APOSTILA DO CURSO

CIÊNCIA DE DADOS IMPRESSIONADOR

MÓDULO 1

O que é Ciência de Dados?

Módulo 1 – O que é Ciência de Dados?

Ciência de dados é o **processo** de **exploração**, **manipulação** e **análise** dos dados para a **descoberta e previsão** através da criação **de hipóteses, testes e validação** com o objetivo de **responder perguntas do negócio e/ou fazer recomendações** capazes de serem diferenciais de negócio.

Todo esse processo precisa ter um forte **embasamento estatístico e matemático** e ser **diretamente ligado ao negócio**, além de poder ser feito de forma **escalável e replicável**.

Módulo 1 – O que é Ciência de Dados?

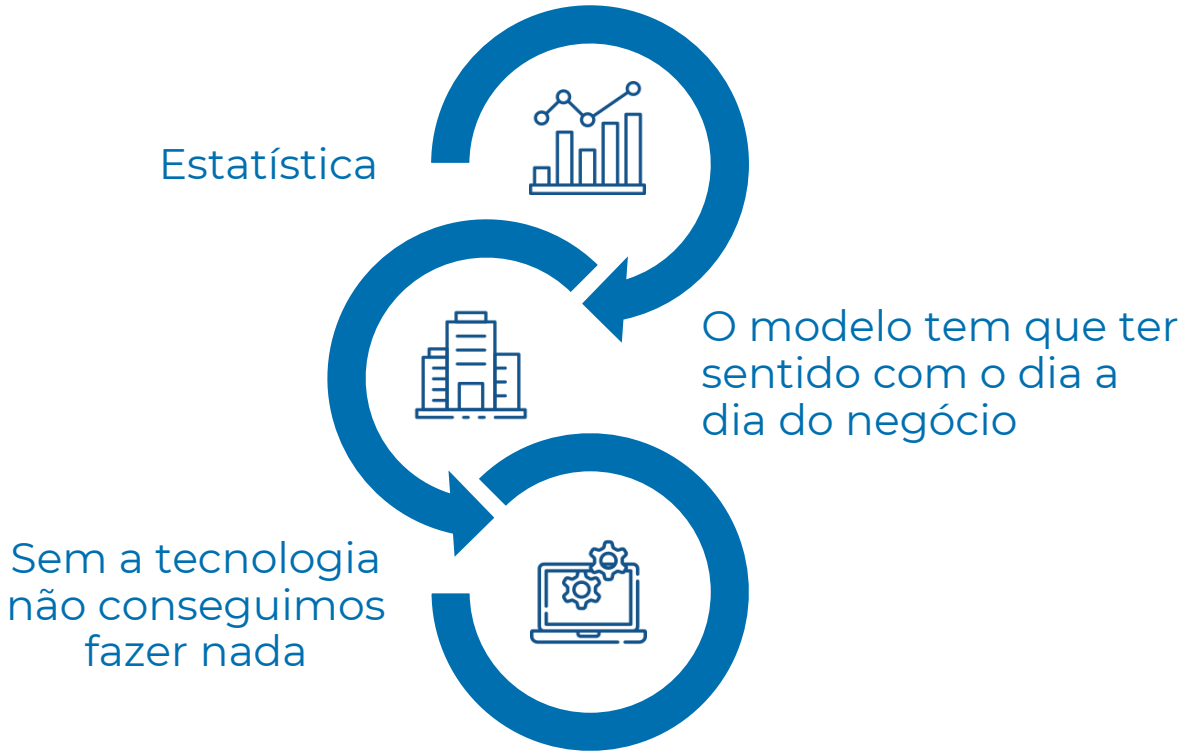
Objetivos da Ciência de Dados:

- Usar dados históricos para fazer previsões sobre o futuro e tentar ser cada vez mais assertivo com a estratégia futura da empresa;
- Descobrir diferenciais para o negócio;
- Encontrar oportunidades ocultas;
- Buscar padrões que não estavam perceptíveis ao olho humano;
- Responder perguntas do negócio e/ou fazer recomendações capazes de serem diferenciais para o negócio.

Módulo 1 – O que é Ciência de Dados?

Todo esse processo precisa ter um forte embasamento estatístico e matemático e ser diretamente ligado ao negócio, além de poder ser feito de forma escalável e replicável.

Na ciência de dados nós temos 3 pilares principais:



Toda essa parte de teste, validação, de fazer as suas próprias perguntas e validar se aquelas perguntas realmente estão acontecendo, isso faz parte do **processo científico**.

Existe um método científico! Observação, hipóteses, testes e validação, análises, monitoramento.

Módulo 1 – O que é Ciência de Dados?

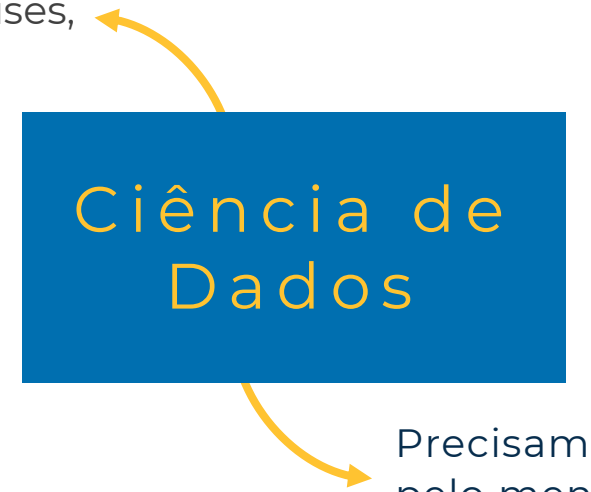
Quando se fala em **dados**, logicamente quer dizer que é necessário a existência deles e que estejam armazenados ou que pelo menos comecem a ser armazenados a partir do início do projeto.

Precisamos que existam dados armazenados (ou pelo menos começar esse armazenamento).
Armazenamento, processamento, visualização.

Módulo 1 – O que é Ciência de Dados?

Então para entregar um bom projeto de *Data Science* é necessário todo um conjunto de **tecnologia** e de **conhecimento**.

Existe um método científico! Observação, hipóteses, testes e validação, análises, monitoramento.



Precisamos que existam dados armazenados (ou pelo menos começar esse armazenamento). Armazenamento, processamento, visualização.

Módulo 1 – O que é Ciência de Dados?

Você com certeza já ouviu falar que o dado é o nosso ativo mais valioso ou que é o **novo petróleo**.

Isso porque quando as empresas sabem usar o dado e usam de forma correta, isso vira um diferencial para o negócio.

Módulo 1 – O que é Ciência de Dados?

Atualmente é necessário encantar e fidelizar o cliente, porque você não precisa mais ir em uma loja, existem sites que vendem o mesmo produto.

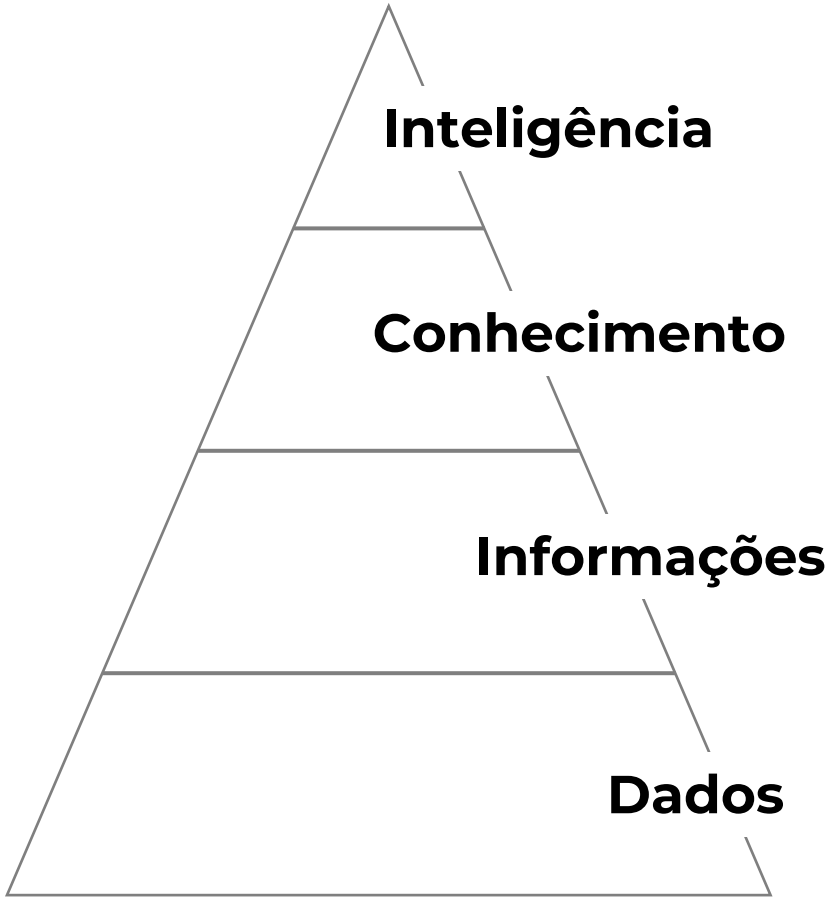
Por isso a importância de **conhecer o cliente**, saber o que, como e quando oferecer. Às vezes o cliente nem queria comprar um produto e decide comprar porque em um momento estratégico você colocou uma propaganda para ele.

Módulo 1 – O que é Ciência de Dados?

Um exemplo do dia a dia é quando você passa do lado de uma academia e o Instagram mostra uma promoção desse local.

Aí surge a pergunta "**como eles sabem disso?**".

A resposta é simples, isso é um dado e é assim que ele está sendo usado hoje em dia pelas empresas. Se a sua empresa ainda não usa, ela pode ficar para trás.

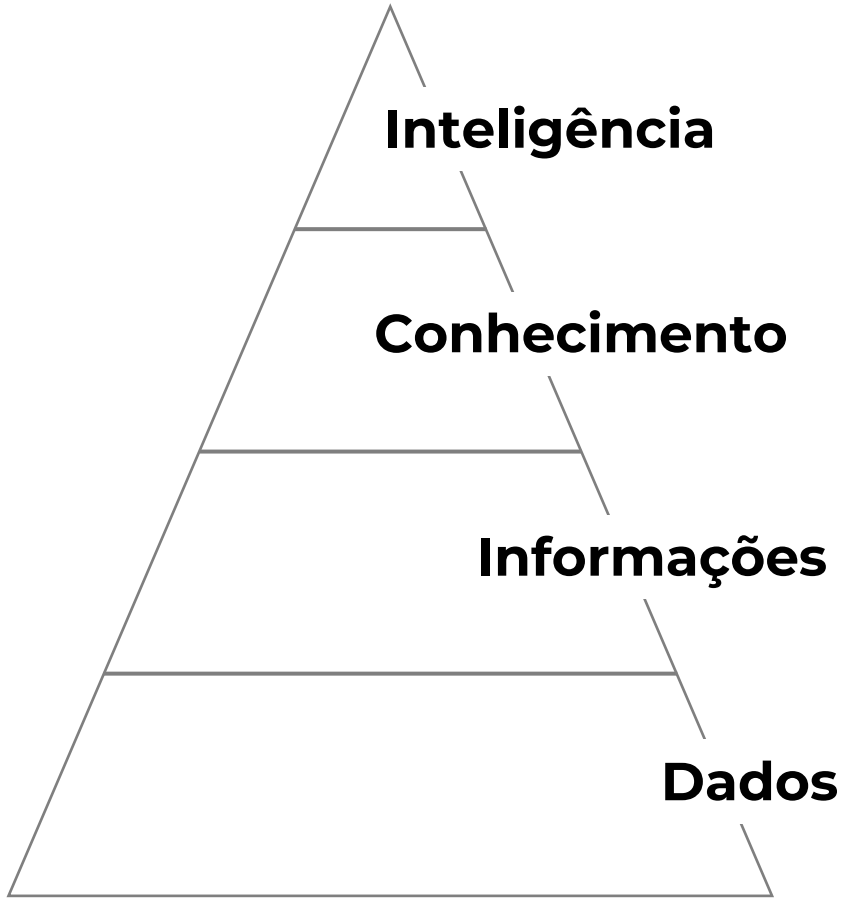


A pirâmide ao lado mostra a **relação entre dados e inteligência**, porque dado qualquer empresa pode ter, às vezes está em uma planilha no Excel ou em papel.

Porém, a maneira como se usa, organiza e transforma esse dado, isso sim é o diferencial.

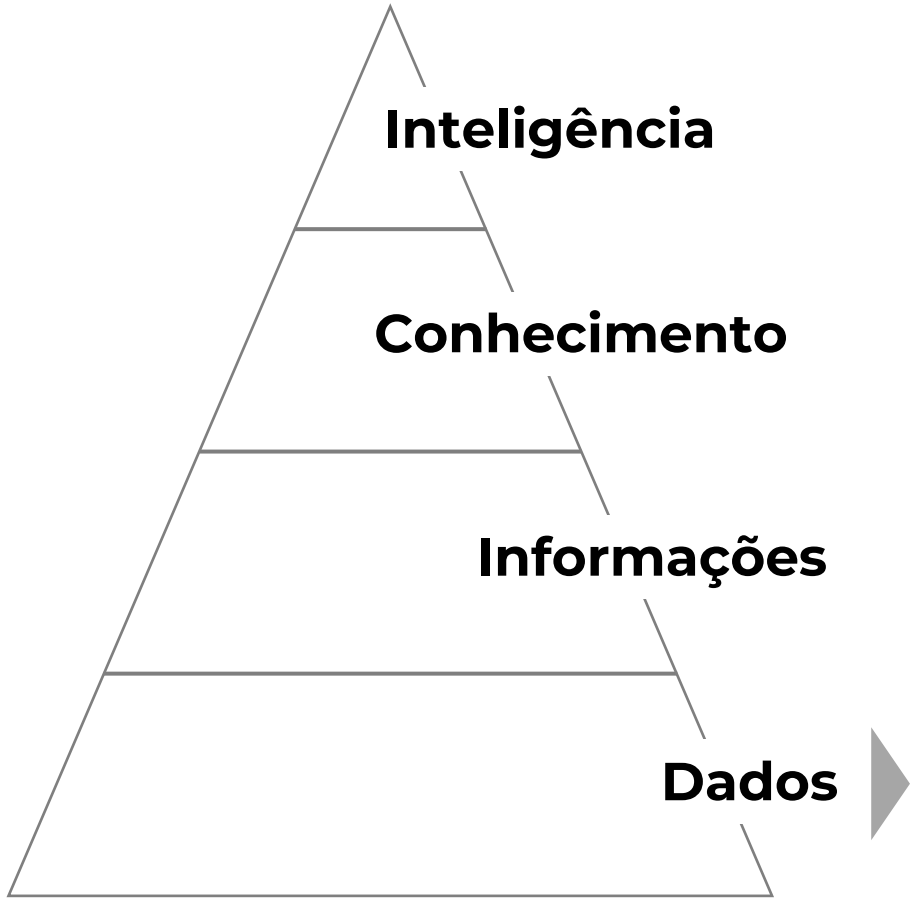
A intenção máxima é **transformar os dados em inteligência** e para isso é necessário ter estatística e um bom conhecimento do negócio, mas isso só é possível se for feito junto com a tecnologia.

Módulo 1 – O que é Ciência de Dados?



Dado é a **informação bruta**, é o registro de que algo aconteceu.

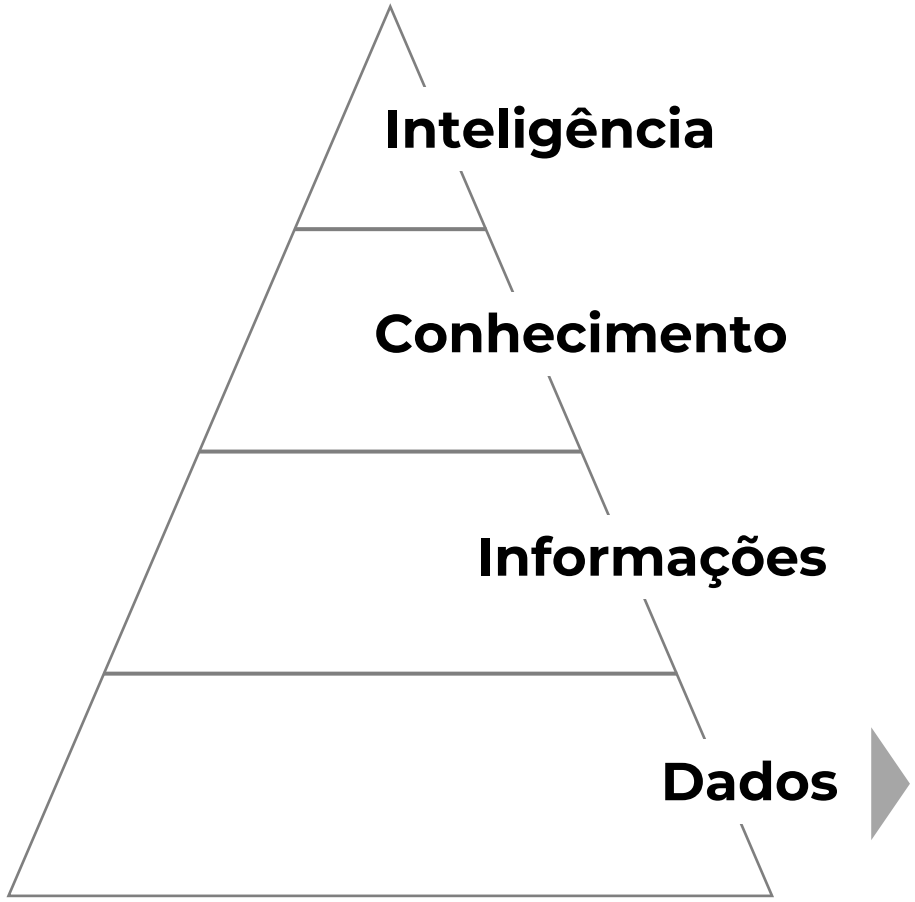
Módulo 1 – O que é Ciência de Dados?



Barulho de apito repetindo 3 vezes

Por exemplo, quando se escuta um barulho apitando 3 vezes, não sabemos de onde está vindo o barulho, nem a que se refere. Apenas que se observou o barulho apitando 3 vezes.

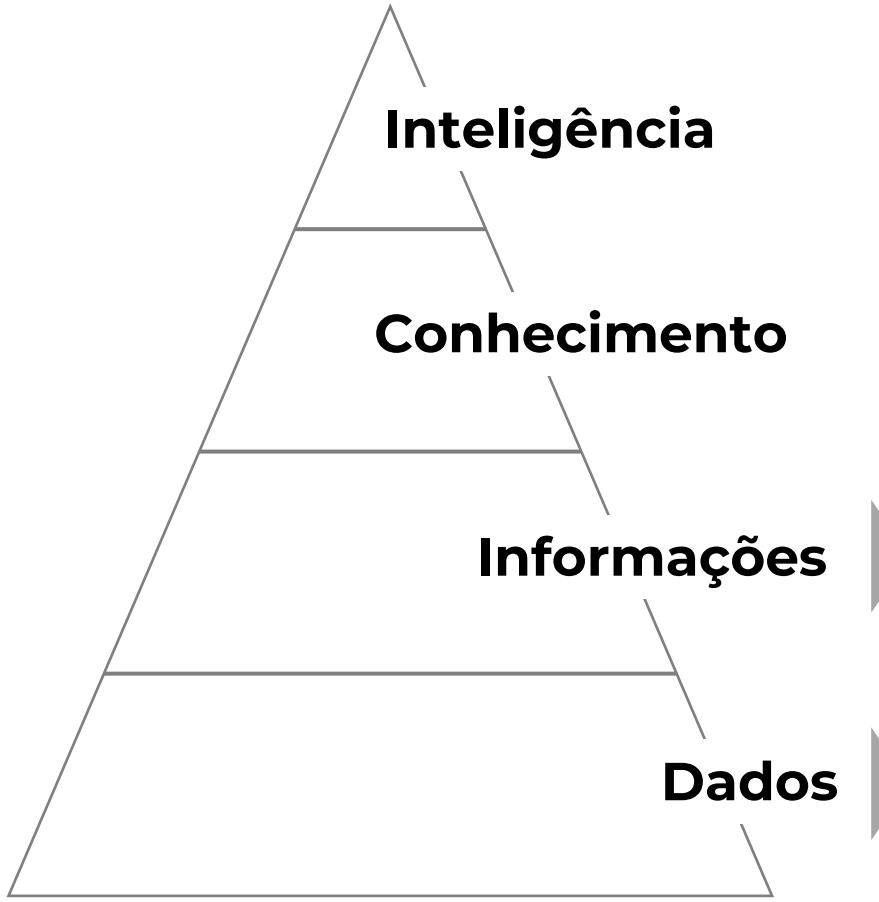
Módulo 1 – O que é Ciência de Dados?



Informação é a contextualização do dado, é o dado depois que passou por algum processamento, ou seja, começa a ter um **significado, relevância e propósito**.

Barulho de apito repetindo 3 vezes

Módulo 1 – O que é Ciência de Dados?

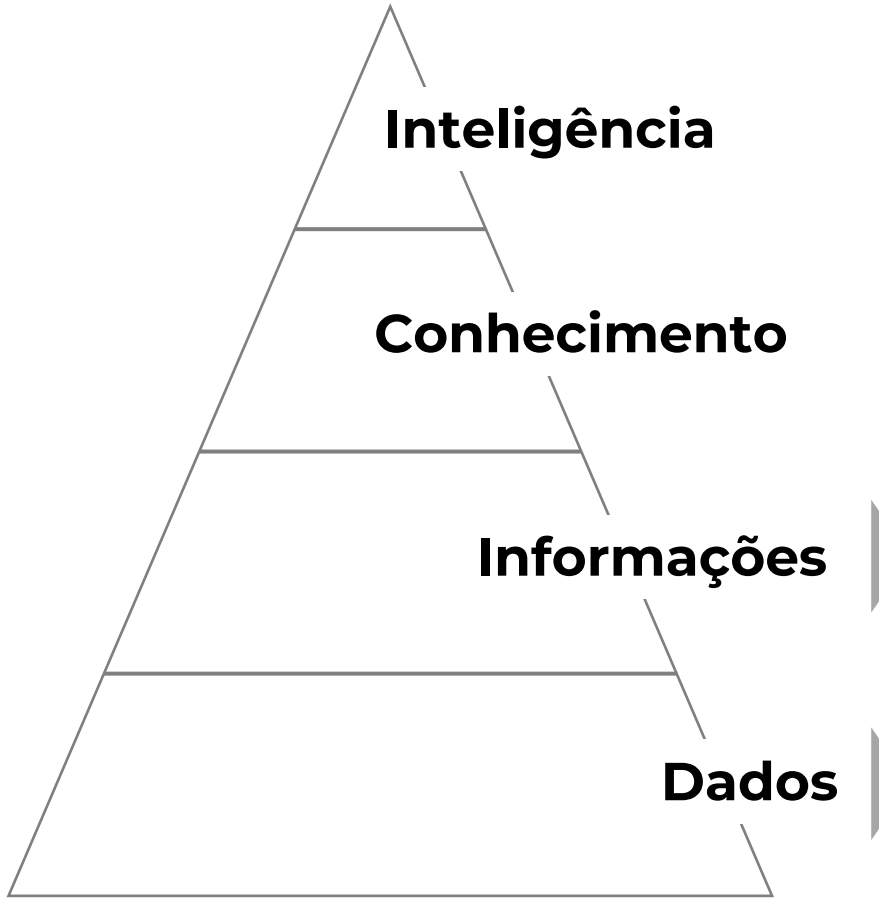


O barulho está vindo do micro-ondas

Barulho de apito repetindo 3 vezes

Por exemplo, ao dizer que o barulho está vindo do micro-ondas, foi contextualizado a que se refere o dado.

Módulo 1 – O que é Ciência de Dados?

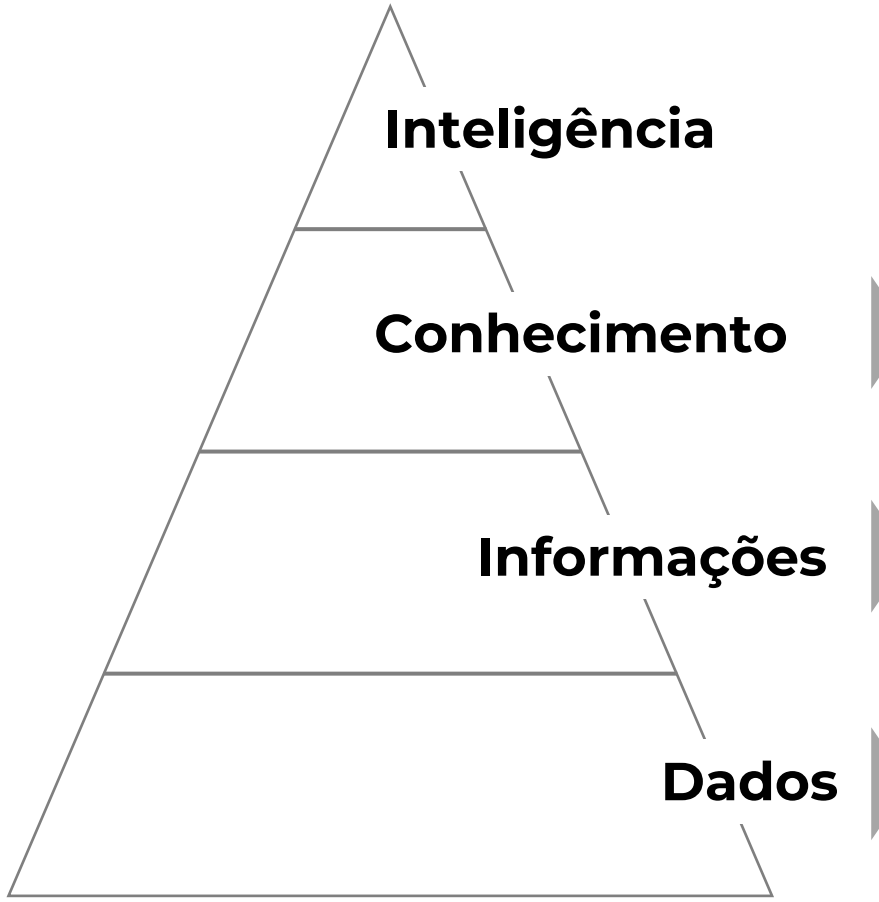


Conhecimento fornece um **significado** para aquele dado, cria uma interpretação, requer uma análise e entendimento.

O barulho está vindo do micro-ondas

Barulho de apito repetindo 3 vezes

Módulo 1 – O que é Ciência de Dados?



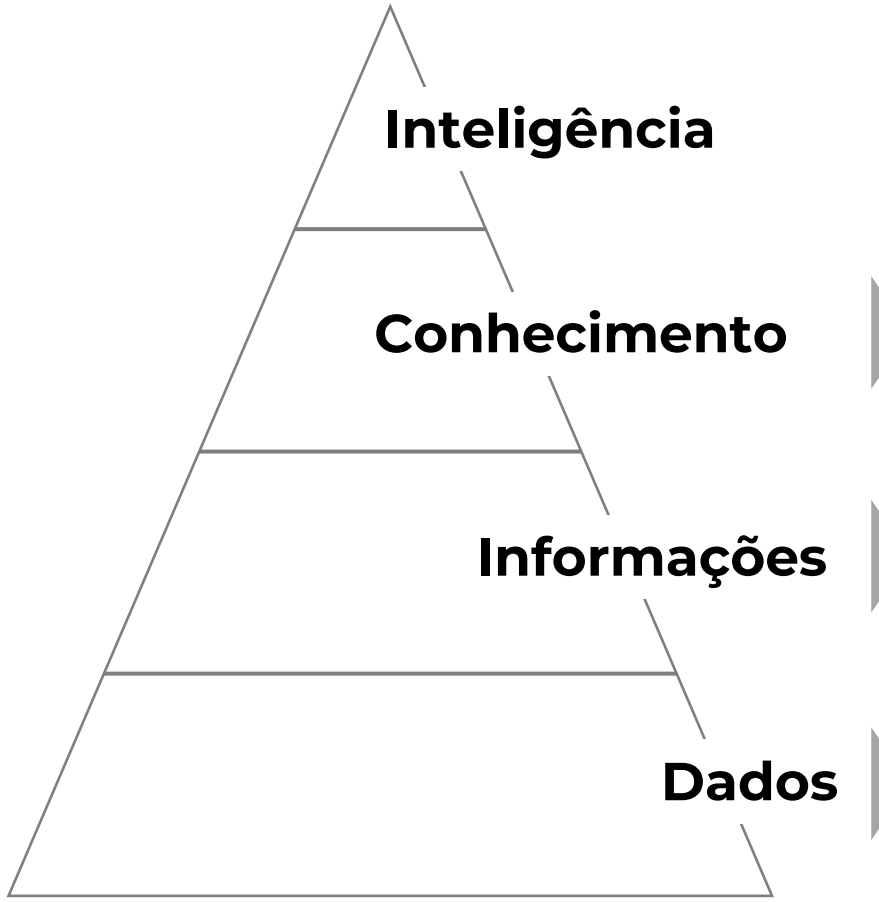
Nesse caso o barulho indica que a comida está pronta, porque eu sei de onde vem aquele barulho e o que significa.

Esse barulho indica que a comida está pronta

O barulho está vindo do micro-ondas

Barulho de apito repetindo 3 vezes

Módulo 1 – O que é Ciência de Dados?



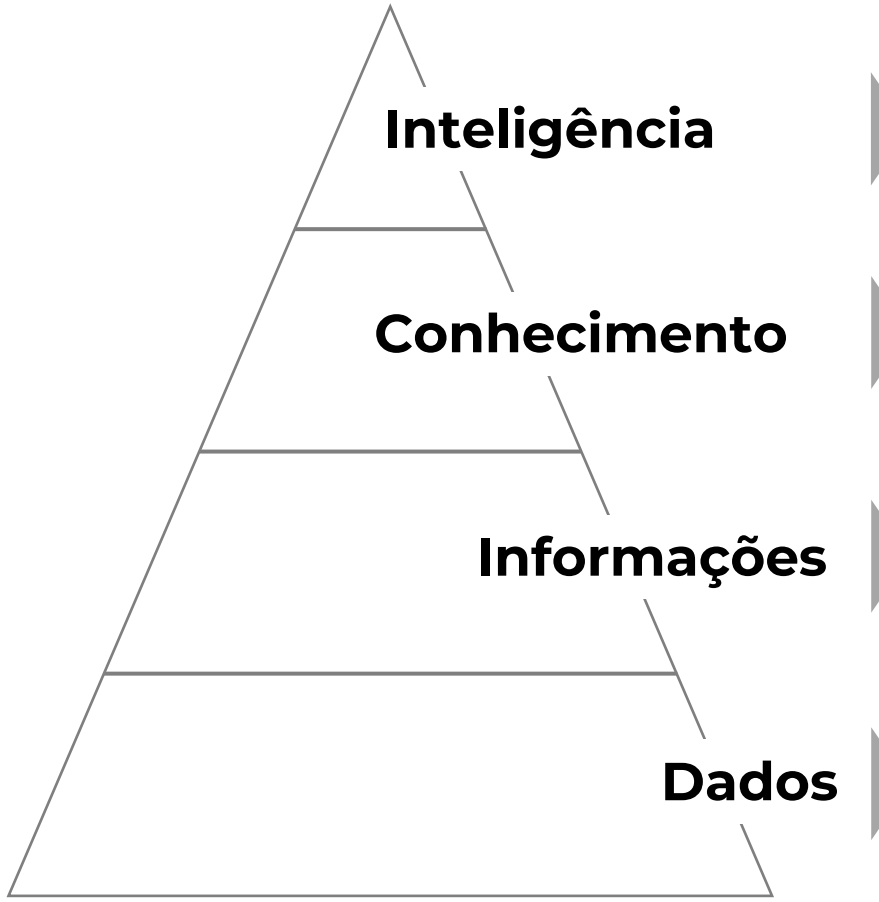
Inteligência é o **uso do conhecimento** aplicado para uma **tomada de decisão** com a finalidade de trazer uma vantagem.

Esse barulho indica que a comida está pronta

O barulho está vindo do micro-ondas

Barulho de apito repetindo 3 vezes

Módulo 1 – O que é Ciência de Dados?



Eu vou retirar a comida e almoçar para que ela esfrie

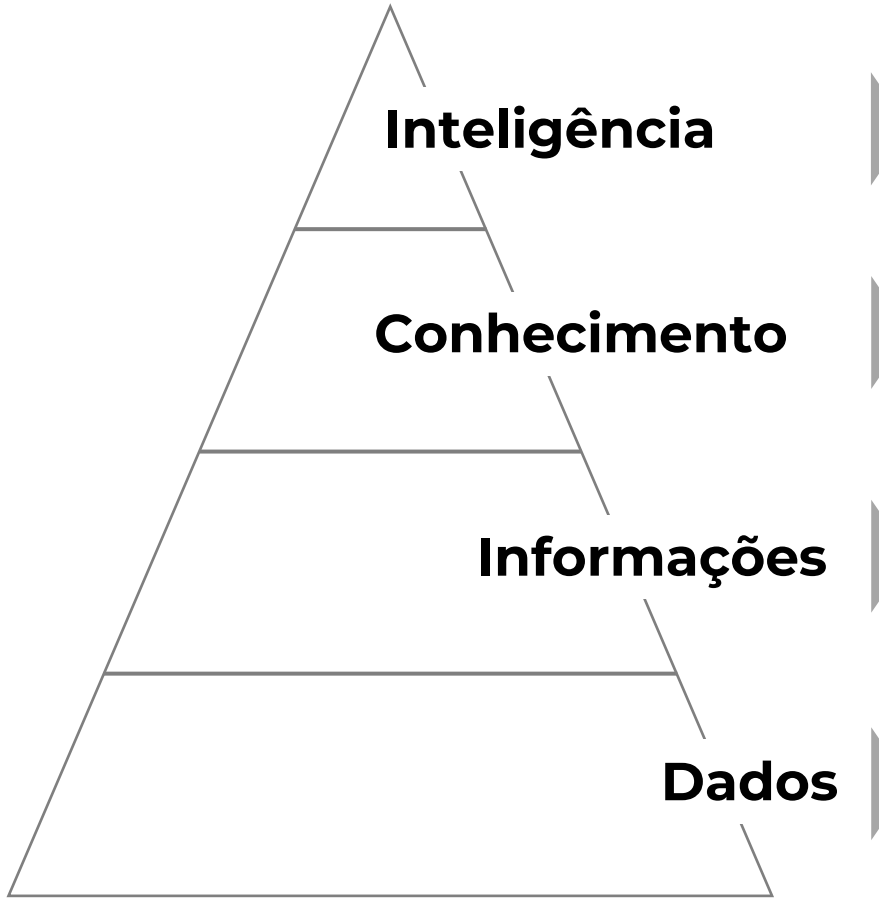
Esse barulho indica que a comida está pronta

O barulho está vindo do micro-ondas

Barulho de apito repetindo 3 vezes

Nesse exemplo, a tomada de decisão é retirar a comida para que ela não esfrie.

Módulo 1 – O que é Ciência de Dados?



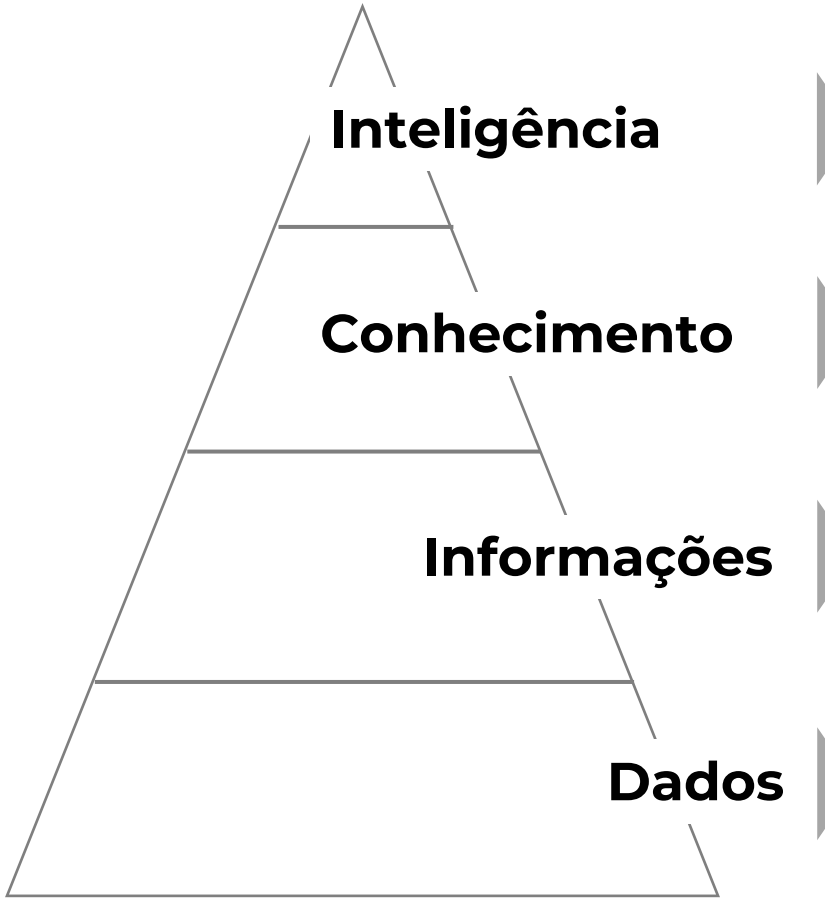
Inteligência	Eu vou retirar a comida e almoçar para que ela esfrie	Conhecimento aplicado, análise para tomada de decisão
Conhecimento	Esse barulho indica que a comida está pronta	Informação após a interpretação, requer análise, entendimento, avaliação e contexto
Informações	O barulho está vindo do micro-ondas	Dados processados, com contexto e significado, possuem relevância e propósito
Dados	Barulho de apito repetindo 3 vezes	Informações brutas, registros de ocorrência, observações

Módulo 1 – O que é Ciência de Dados?

Um mesmo dado pode ter contextos diferentes que vão levar a uma tomada de decisão diferente.

Vamos comparar a seguir a tomada de decisão para um mesmo dado, utilizando como exemplo o alarme de incêndio e o micro-ondas.

Módulo 1 – O que é Ciência de Dados?



Eu vou retirar a comida e almoçar para que ela esfrie

Esse barulho indica que a comida está pronta

O barulho está vindo do micro-ondas

Barulho de apito repetindo 3 vezes

Eu vou correr o mais rápido possível

Esse barulho indica que está acontecendo um incêndio

O barulho está vindo do alarme de incêndio

Barulho de apito repetindo 3 vezes

Então, para todo dado é necessário **avaliar esse conhecimento** para depois aplicar uma **tomada de decisão**.

No micro-ondas o conhecimento é que a minha comida está pronta e a decisão é a retirada da comida para almoçar antes que ela esfrie.

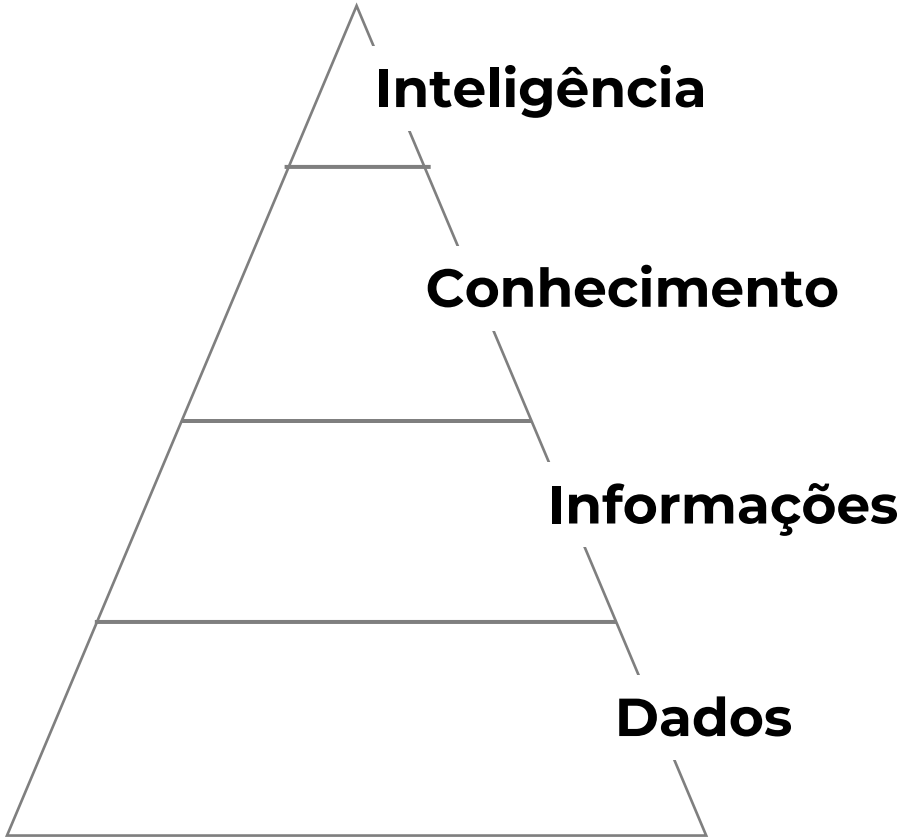
Enquanto no alarme de incêndio, o meu conhecimento é identificar que está acontecendo um incêndio e a minha inteligência será a decisão de correr o mais rápido possível, porque nesse caso é algo muito mais urgente do que a comida esfriar.

Analogamente, imagine um produto que não vende muito. Se em determinado mês a venda dele for baixa, não é necessário ficar preocupado.

Porém, se a venda do meu produto principal for baixa, eu preciso entender para agir o mais rápido possível.

Agora um exemplo prático:

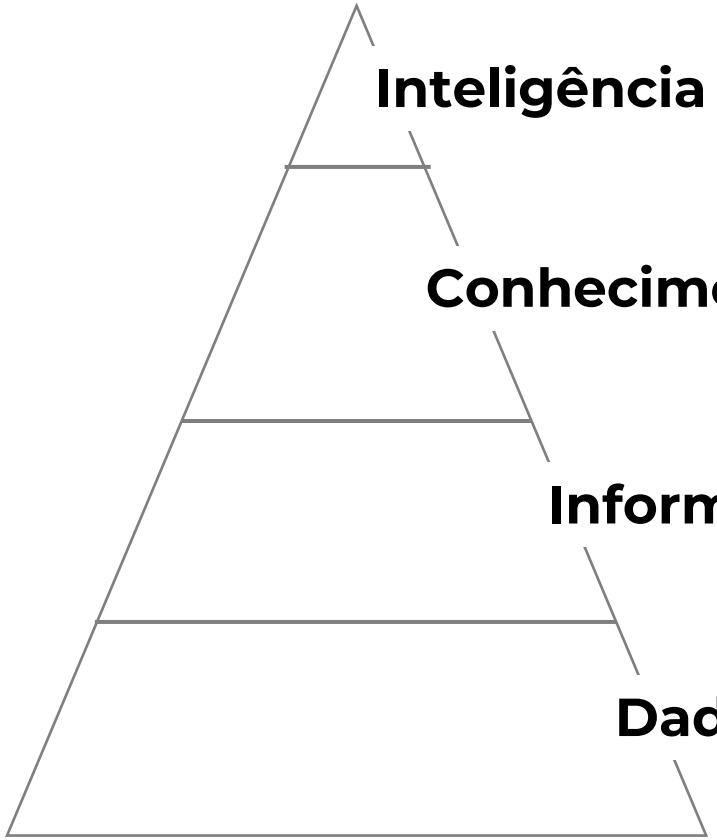
Apenas com o dado não sabemos o significado desses números.



Chocolate X;2000;10000

Informações brutas, registros de ocorrência, observações

Agora um exemplo prático:



Agora com a informação foi colocado um rótulo do que cada coluna está dizendo. Porém ainda não podemos concluir se esse estoque é bom.

Produto: Chocolate X, Estoque: 2000, Venda Mensal: 10000

Informações brutas, registros de ocorrência, observações

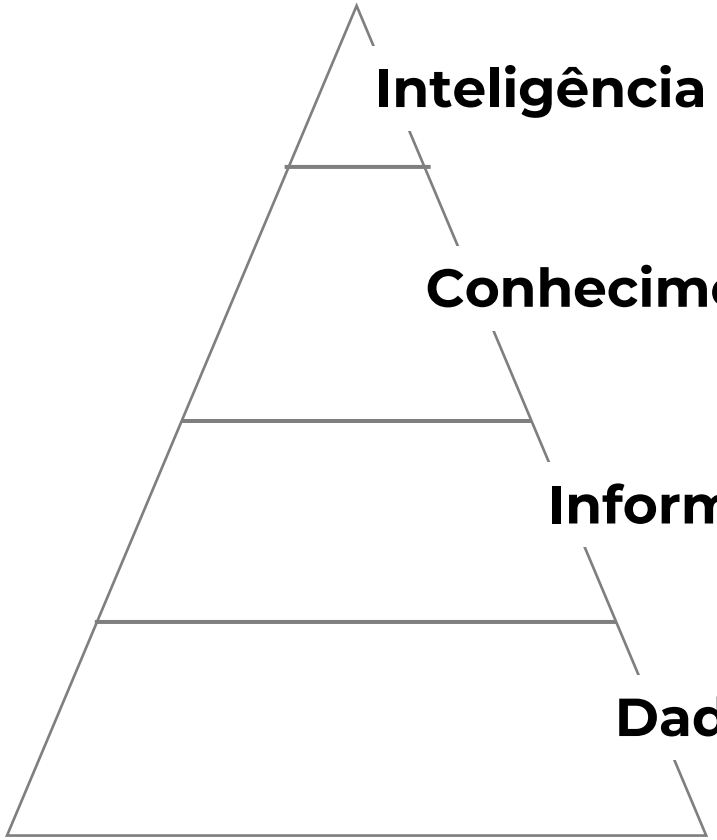
Chocolate X;2000;10000

Informações brutas, registros de ocorrência, observações

Módulo 1 – O que é Ciência de Dados?

Agora um exemplo prático:

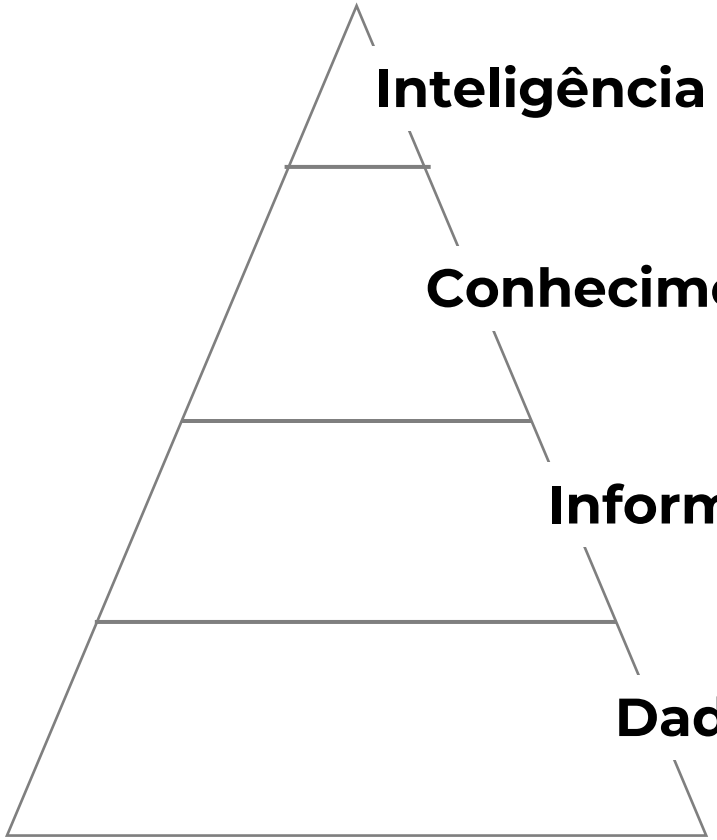
No conhecimento vamos fazer a interpretação, ou seja, fazer a comparação do estoque com a venda mensal.



Conhecimento	**Existe o risco de falta de estoque (ruptura) desse chocolate na loja**	Informação após a interpretação, requer análise, entendimento, avaliação e contexto
Informações	**Produto: Chocolate X, Estoque: 2000, Venda Mensal: 10000**	Informações brutas, registros de ocorrência, observações
Dados	**Chocolate X;2000;10000**	Informações brutas, registros de ocorrência, observações

Módulo 1 – O que é Ciência de Dados?

Agora um exemplo prático:



Inteligência	**Vamos fazer um novo pedido maior ainda desse produto pois a Páscoa está próxima**	Conhecimento aplicado, análise para tomada de decisão
Conhecimento	**Existe o risco de falta de estoque (ruptura) desse chocolate na loja**	Informação após a interpretação, requer análise, entendimento, avaliação e contexto
Informações	**Produto: Chocolate X, Estoque: 2000, Venda Mensal: 10000**	Informações brutas, registros de ocorrência, observações
Dados	**Chocolate X;2000;10000**	Informações brutas, registros de ocorrência, observações

Então uma informação que antes não tinha sentido como “Chocolate X;2000;10000” nos proporcionou o *insight* de enviar mais chocolate à loja para que não falte pois a páscoa está próxima.

Esse *insight* é o que nós vamos fazer em escalas maiores, como por exemplo, colocar um produto do lado de outro porque a probabilidade é maior desses produtos venderem juntos ou oferecer um produto para um cliente porque a chance dele comprar é alta.

Módulo 1 – O que eu quero responder?

São necessárias muitas perguntas e observações para fazer as nossas hipóteses, mas existe uma pergunta fundamental que garante o nosso sucesso na Ciência de Dados:

O QUE EU QUERO

RESPONDER?

Módulo 1 – O que eu quero responder?

Na maioria das vezes em projeto de dados não trabalhamos sozinho, trabalhamos para um cliente ou para um gestor. Sempre vai ter alguém querendo saber alguma coisa.

Por que a venda foi ruim?

Por que eu não consegui vender o meu produto?

Em geral, essas perguntas são muito abertas e não nos dizem o que eles realmente querem saber. É por isso que vamos precisar investigar e procurar um pouco mais.

Por exemplo: a venda geral dos produtos foi ruim ou apenas a venda de um produto em específico?

Módulo 1 – O que eu quero responder?

Se a conclusão apresentada para o chefe foi de que a venda foi ruim em todas as empresas, mas ele queria saber se aquele produto vendeu nas outras empresas, a resposta que ele precisava não foi dita.

A pergunta estava muito clara para ele, mas não respondemos. Por isso, vamos trabalhar no nosso framework de data Science que é: como fazer as perguntas, como mapear o que exatamente a gente precisa descobrir, para depois começar a entrar nos dados.

Módulo 1 – O que eu quero responder?

Um exemplo para responder a pergunta “O que eu quero responder” é o livro “O guia do mochileiro das galáxias”.

No livro querem encontrar a resposta para a pergunta “Qual a resposta da vida, do universo e tudo mais?” e a resposta dada pela máquina construída é 42.

As pessoas falaram que estava errado e a máquina respondeu que não era a resposta que estava errada, e sim que eles não souberam fazer a pergunta.

Módulo 1 – O que eu quero responder?

Ou seja, se você não sabe o que perguntar qualquer resposta que lhe for entregue vai servir para você e se não servir você deve reformular a pergunta.

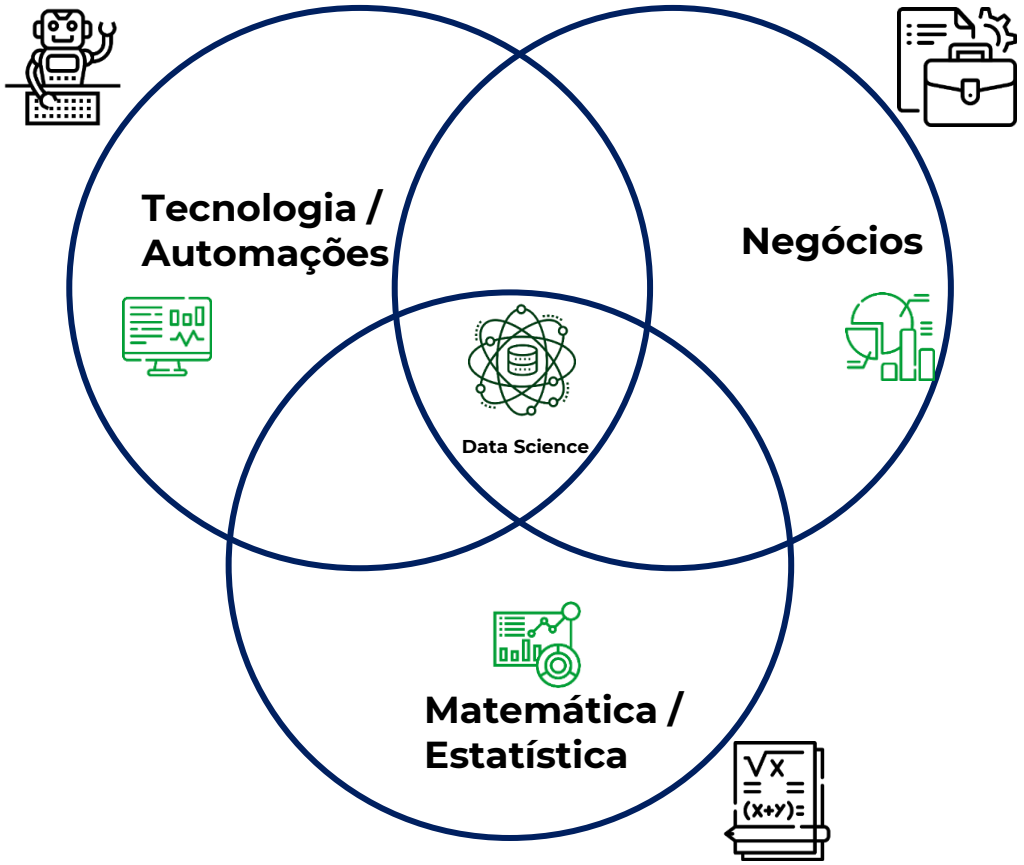
Então, garantam que vocês vão saber o que responder para fazer um projeto de data Science. Pensem muito bem no que vocês querem responder antes de começar.

Módulo 1 – Os Pilares da Ciência de Dados

A ciência de dados tem **3 pilares**: tecnologia, negócio e matemática.

O cientista de dados consegue relacionar essas 3 áreas, mas para isso é necessário ter uma base estatística, tecnológica (Python) e de negócios.

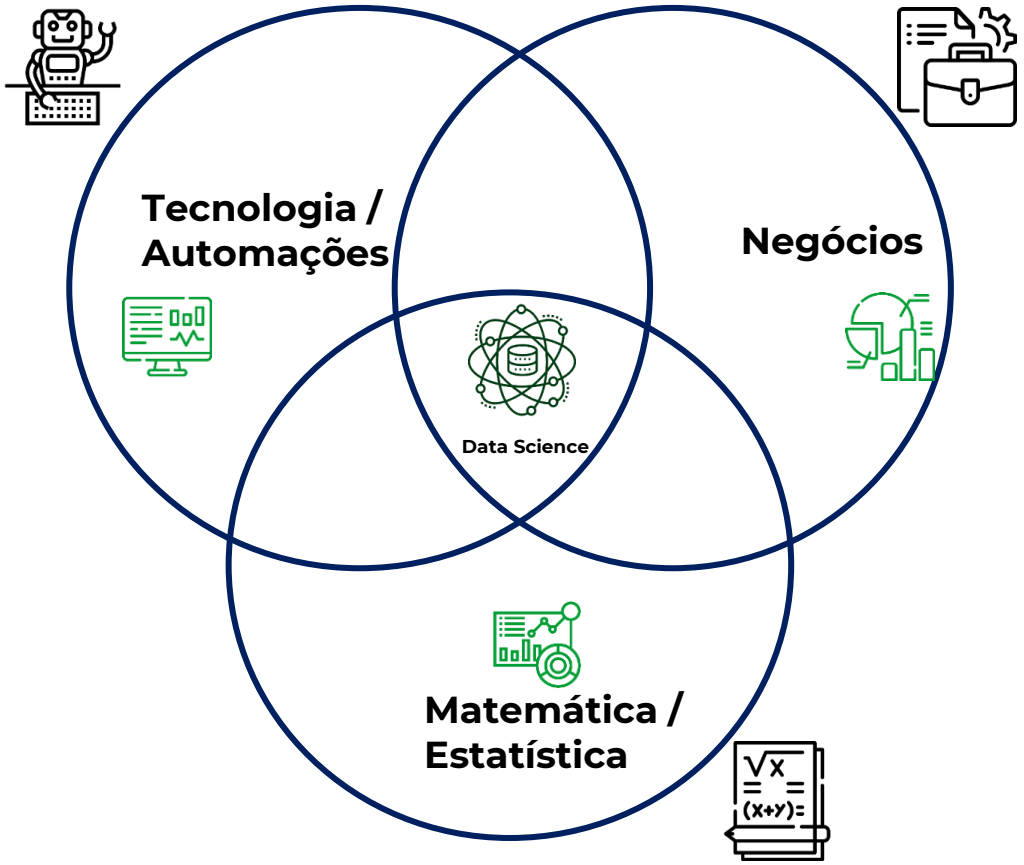
Diferente de um projeto de automação o data Science lida muito com o cliente e com expectativas que não estão claras para ele.



Módulo 1 – Os Pilares da Ciência de Dados

Esse é um projeto muito mais complexo que lida diretamente com o cliente ou chefe.

Às vezes podemos achar um resultado certo, mas que pode não ter validade estatística ou não fazer sentido dentro do dia a dia do negócio.

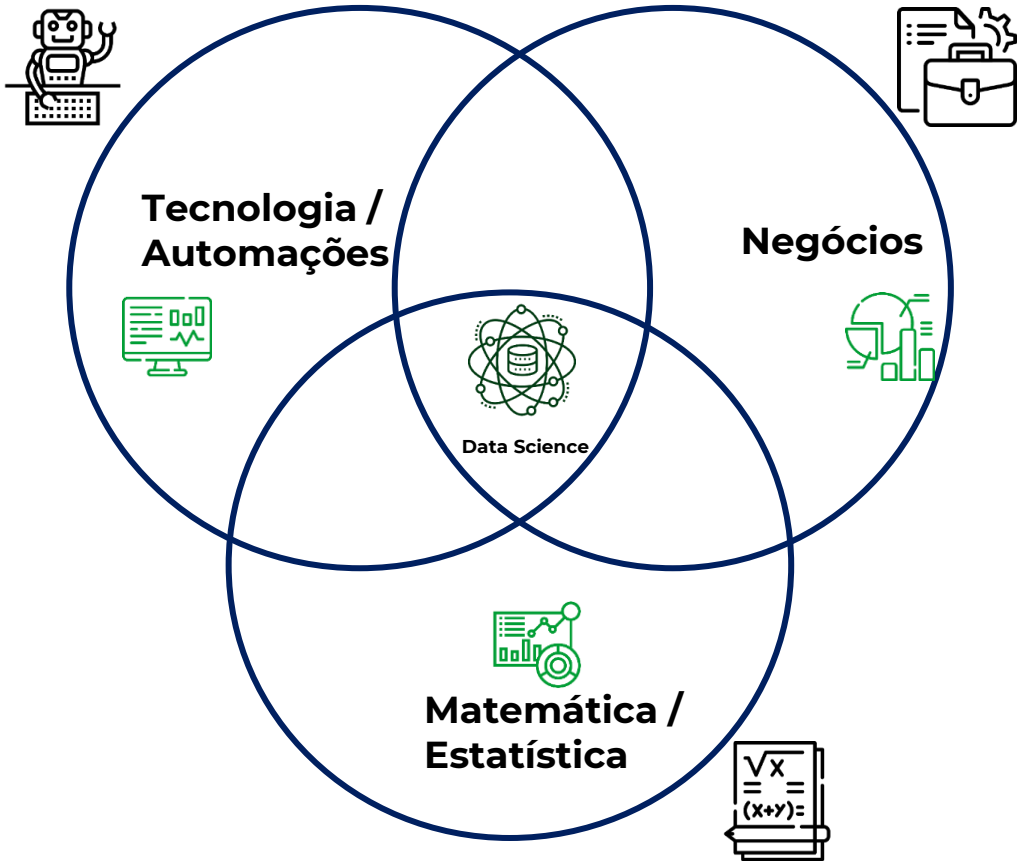


Módulo 1 – Os Pilares da Ciência de Dados

O cientista de dados é um profissional multidisciplinar, que consegue navegar e entender o que as áreas precisam.

Ele possui uma forte base estatística, para verificar se faz sentido o que os modelos estão gerando.

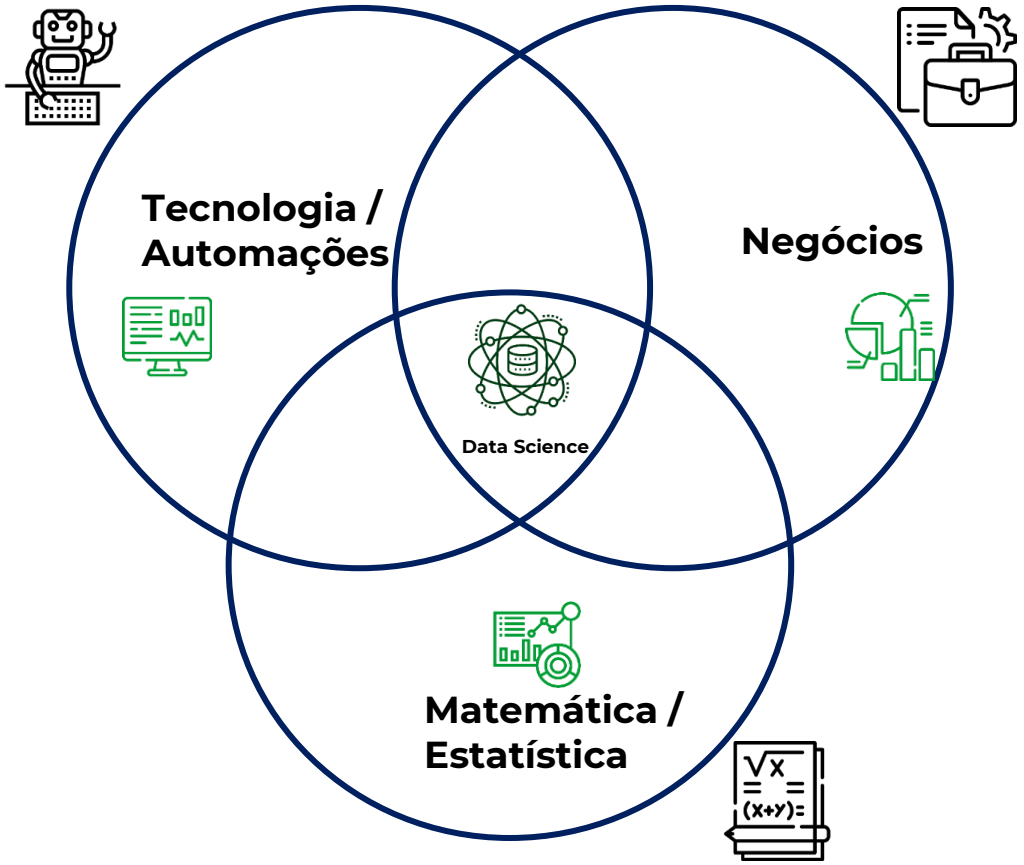
Além de conhecer muito sobre tecnologia, seja para fazer o *upload* dos dados ou tratar os que já estão disponíveis.



Módulo 1 – Os Pilares da Ciência de Dados

Você pode estar se perguntando: “ok, mas para que serve um cientista de dados se já existem softwares que fazem isso e as empresas já usam a estatística tradicional?”

A ciência de dados é a junção da tecnologia, negócio e matemática, para calcular e analisar um resultado com base nesses 3 pilares.

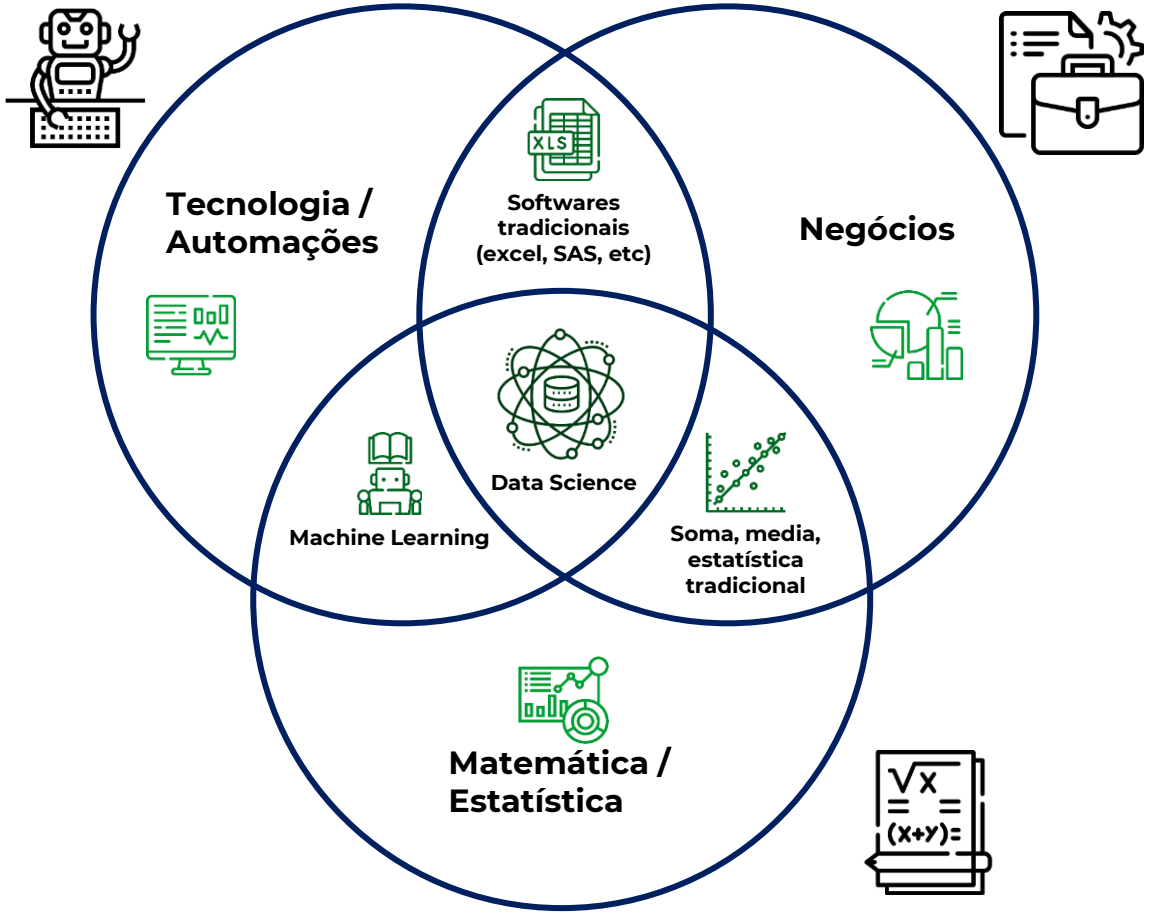


Módulo 1 – Os Pilares da Ciência de Dados

Quando se fala de tecnologia e negócio existem diversas ferramentas e softwares de apoio, como o Excel.

Na matemática e tecnologia temos o modelo *Machine Learning*. Principalmente quando se fala em matemática, estatística e negócio estamos falando de soma, média e estatística tradicional que as empresas já fazem.

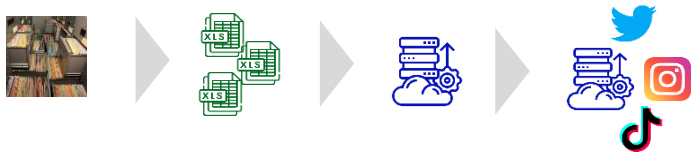
Porém, muitas vezes essas médias não traduzem a realidade, por isso, a importância da área de *Data Science*.

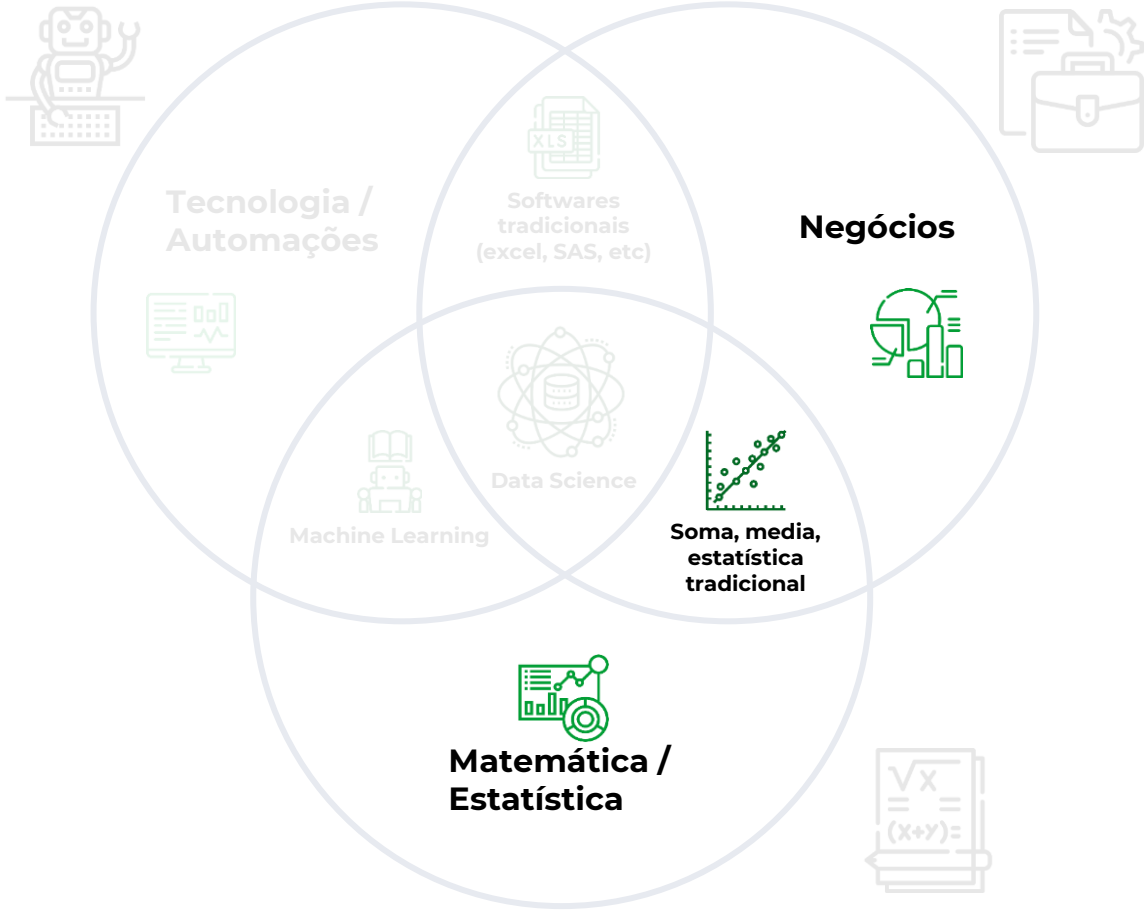


Módulo 1 – Os Pilares da Ciência de Dados

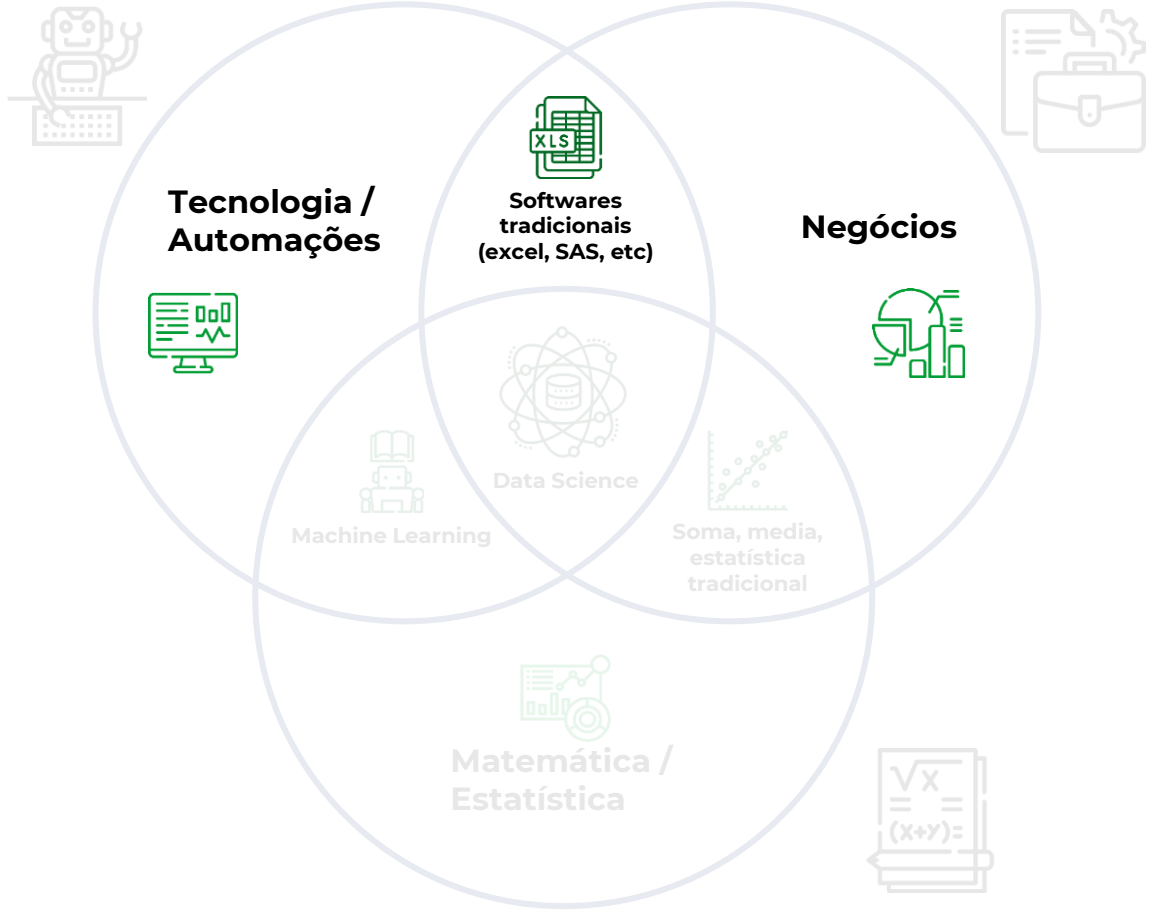
Por que é tão perigoso a gente não usar esses 3 pilares não relacionados?

Sem a tecnologia atrelada não é possível fazer a empresa continuar funcionando na quantidade de dados que temos hoje em dia.



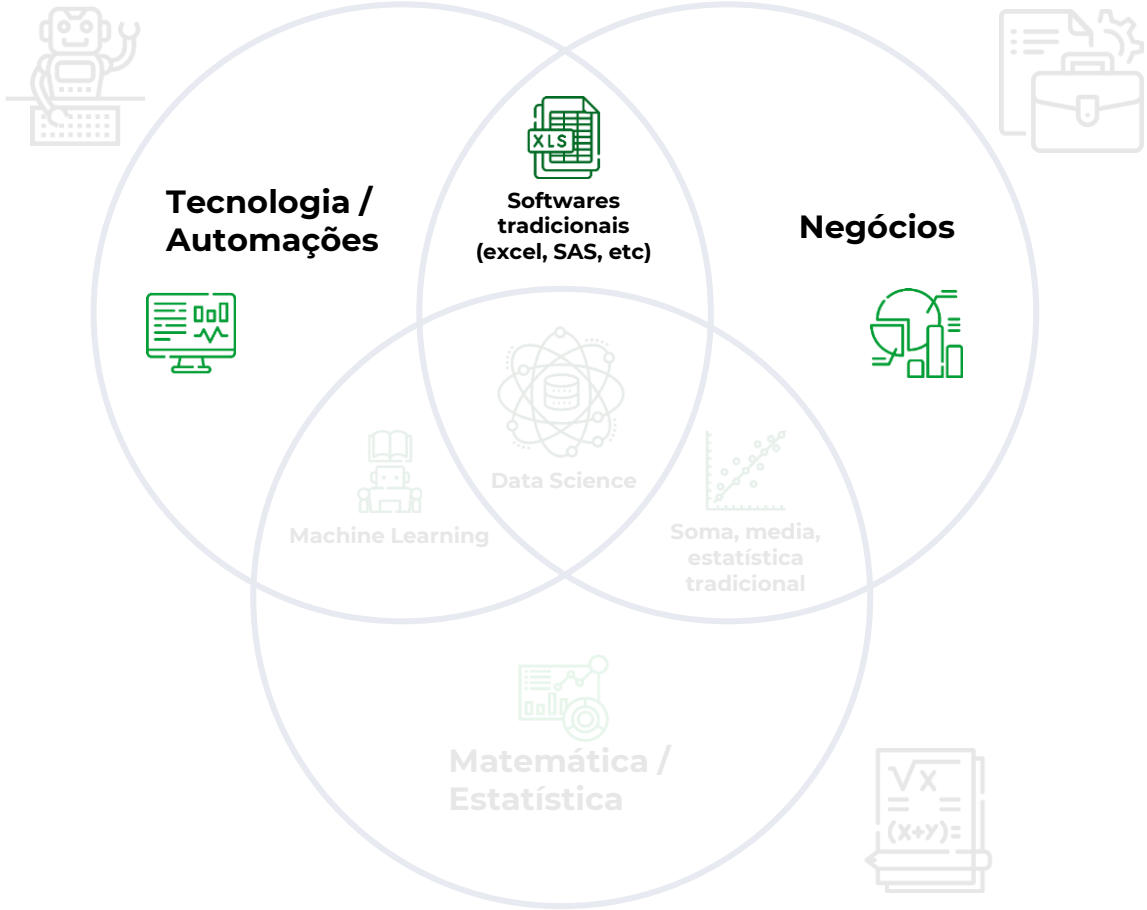


Agora, por exemplo, se só considerarmos os softwares tradicionais, podemos ter conclusões corretas, só que pode ter pouca validade, ou até nenhuma validade estatística e assim também pode não ter tanto significado para o negócio.



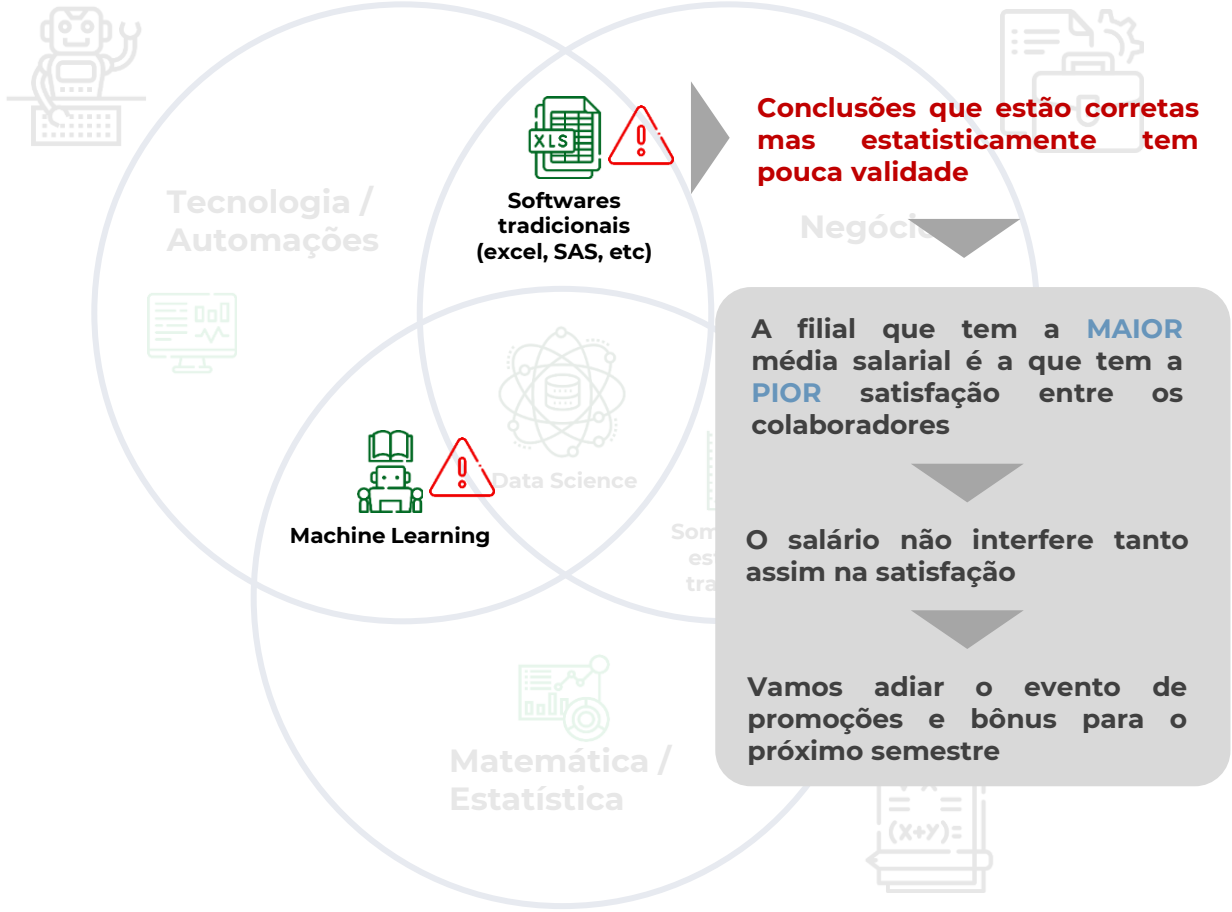
Por exemplo: se a média de vendas desse ano foi melhor que a do ano passado, porém se eu tive muito mais lojas ou muito mais vendedores, essa não é uma informação que tem tanta validade.

Por isso é necessário entender estatisticamente se aquele resultado faz sentido.

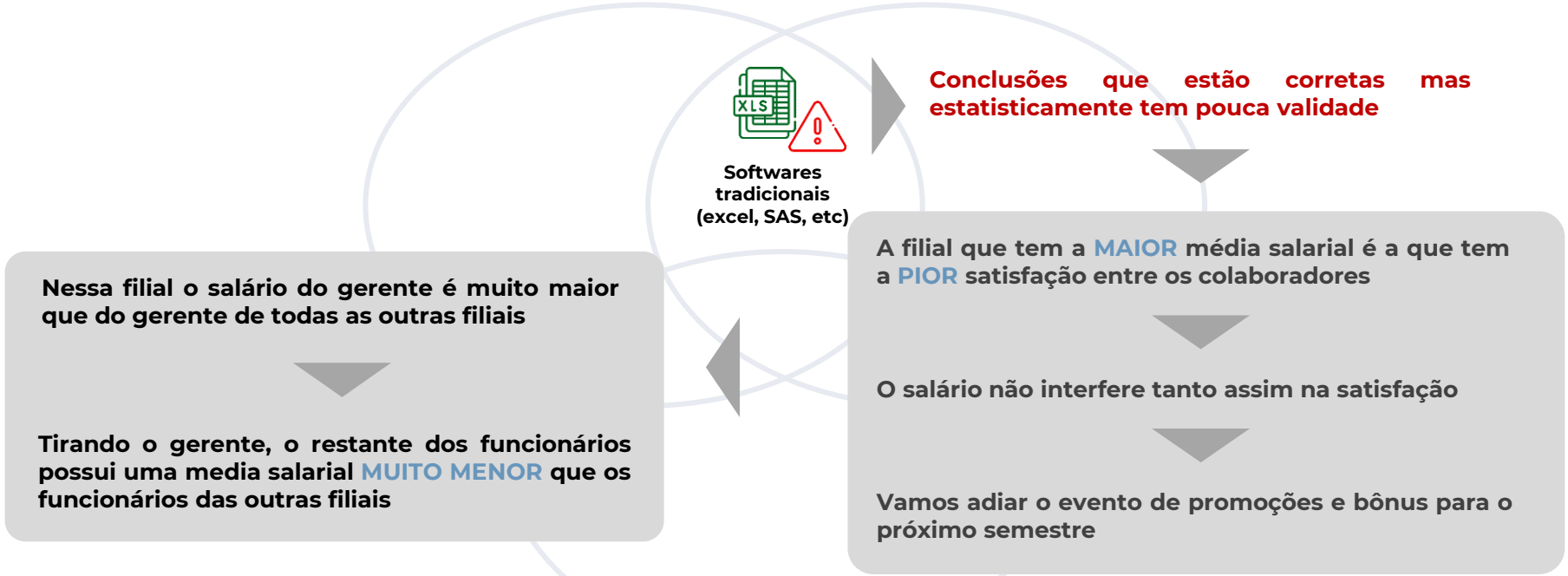


Agora vamos ver um exemplo simples:

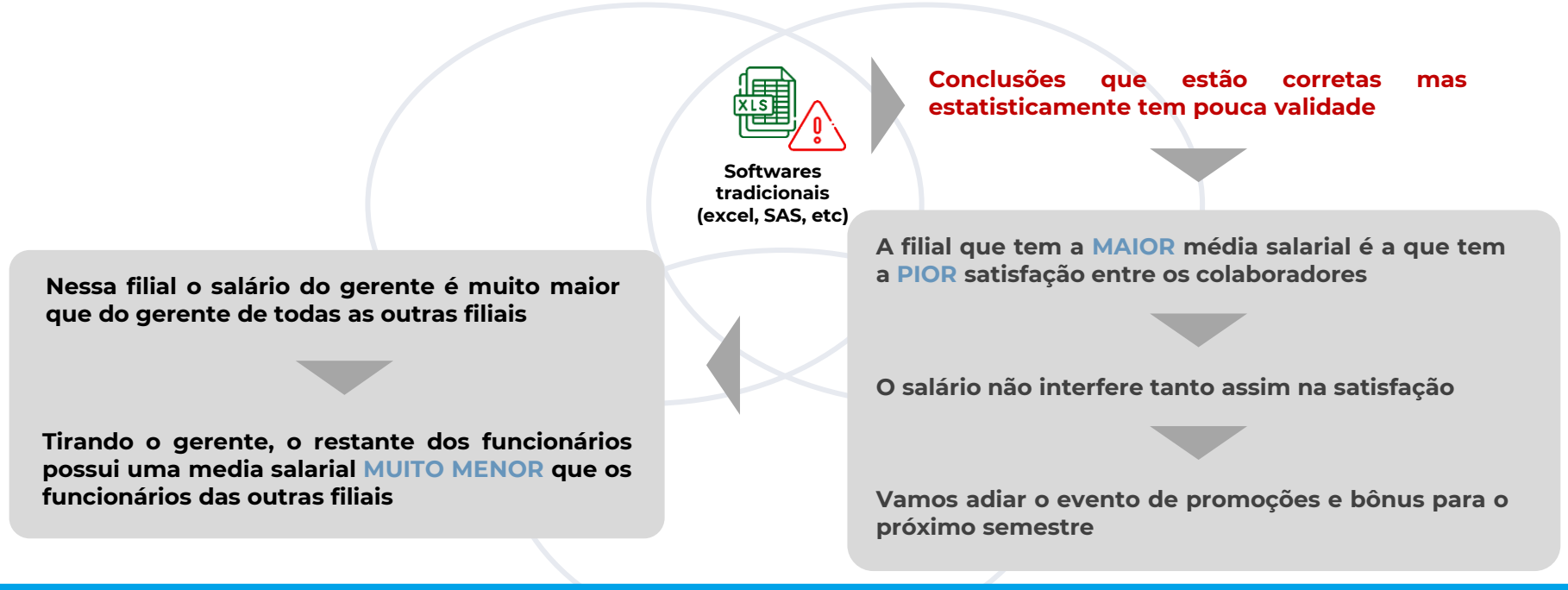
Realizar uma análise de filiais porque o índice de insatisfação está alto. Pela análise da média salarial é possível notar que a filial com pior satisfação é a que possui maior média, com isso, conclui-se que o salário não é o problema e adia-se o evento de promoções.



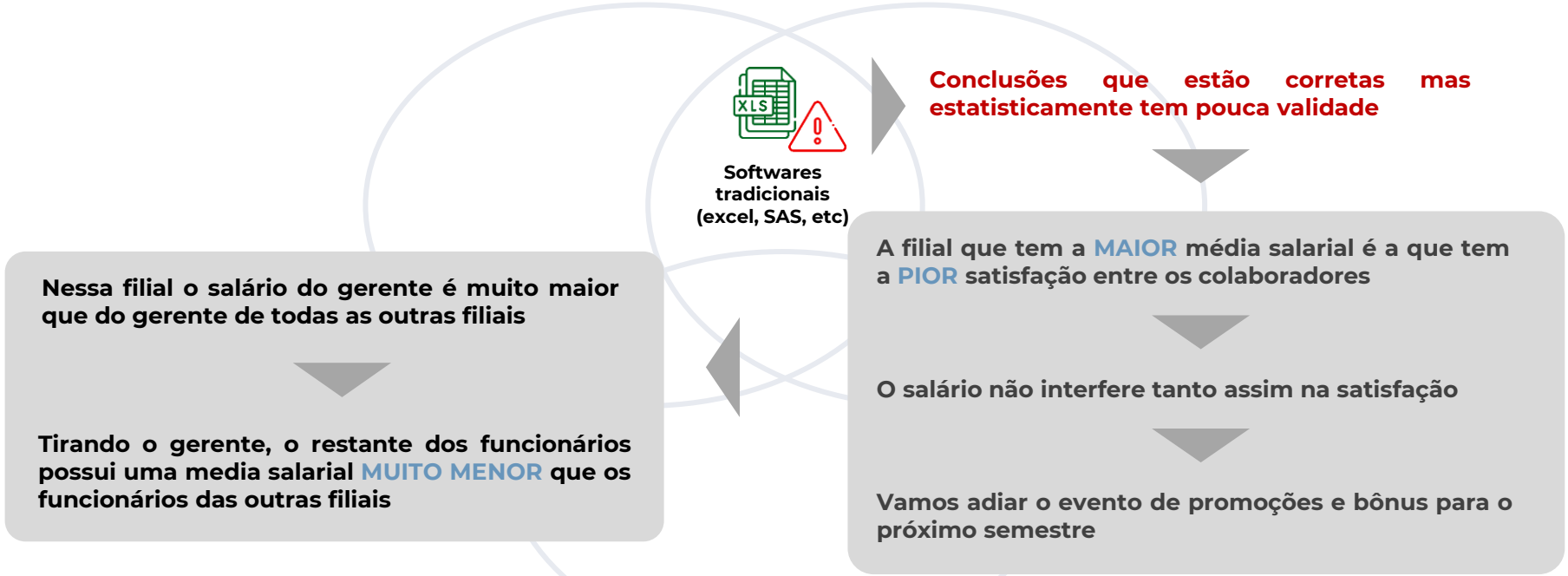
Quando se analisa a estatística dessa informação, nota-se que na verdade nessa filial de maior média salarial, o salário do gerente é muito maior que o das outras filiais. Então, essa média é maior por causa do salário do gerente, porém o salário de todos os outros funcionários é muito menor, isso acontece pois a média pode ser afetada pelos outliers.



Com a conclusão equivocada da fonte do problema, as promoções foram adiadas e vários funcionários pediram demissão. Nesse caso eu gerei uma conclusão que estava certa (a média salarial é maior), só que estatisticamente não foi levado em consideração o desvio padrão.



A decisão foi tomada, mas não tinha validade estatística e a decisão prejudicou a empresa.



Um outro exemplo, se estivéssemos na dúvida entre uma viagem para essas 2 cidades e o nosso objetivo fosse ir à praia. Observamos que a Cidade 2 tem uma temperatura média maior que a Cidade 1, como o objetivo é ir à praia eu vou escolher a Cidade 2 que tem uma média de temperatura maior.

Softwares tradicionais (excel, SAS, etc)

Conclusões que estão corretas mas estatisticamente tem pouca validade

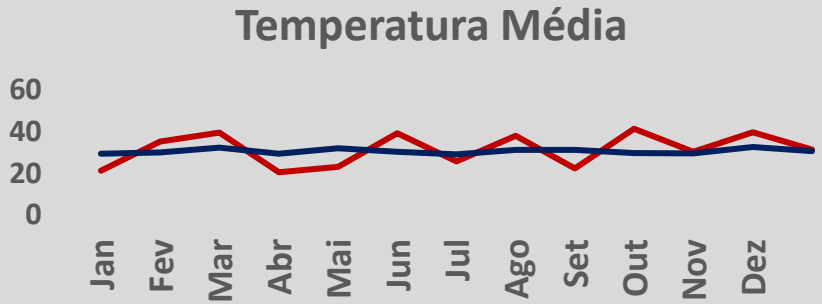
Cidade	Temp Média
Cidade 1	30,5
Cidade 2	31,3

Ao analisar o histórico da temperatura, percebe-se que o histórico da cidade 2 tem uma variação de temperatura alta e por mais que a média seja maior, tem chance de a temperatura estar 20°C, então eu vou acabar sentindo frio na praia.

Cidade	Temp Média
Cidade 1	30,5
Cidade 2	31,3

Softwares tradicionais (excel, SAS, etc)

Conclusões que estão corretas mas estatisticamente tem pouca validade

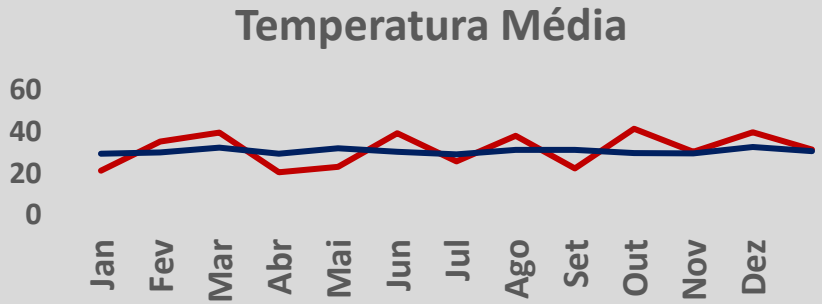


Embora a Cidade 1 tenha uma média menor, a sua temperatura é mais constante. Ou seja, se eu quiser uma segurança de viajar com uma temperatura agradável eu deveria escolher a Cidade 1. Nesse caso, iríamos escolher um lugar que tinha a chance de sentir frio na praia, porque não entendemos esses conceitos básicos de estatística.

Cidade	Temp Média
Cidade 1	30,5
Cidade 2	31,3

Softwares tradicionais (excel, SAS, etc)

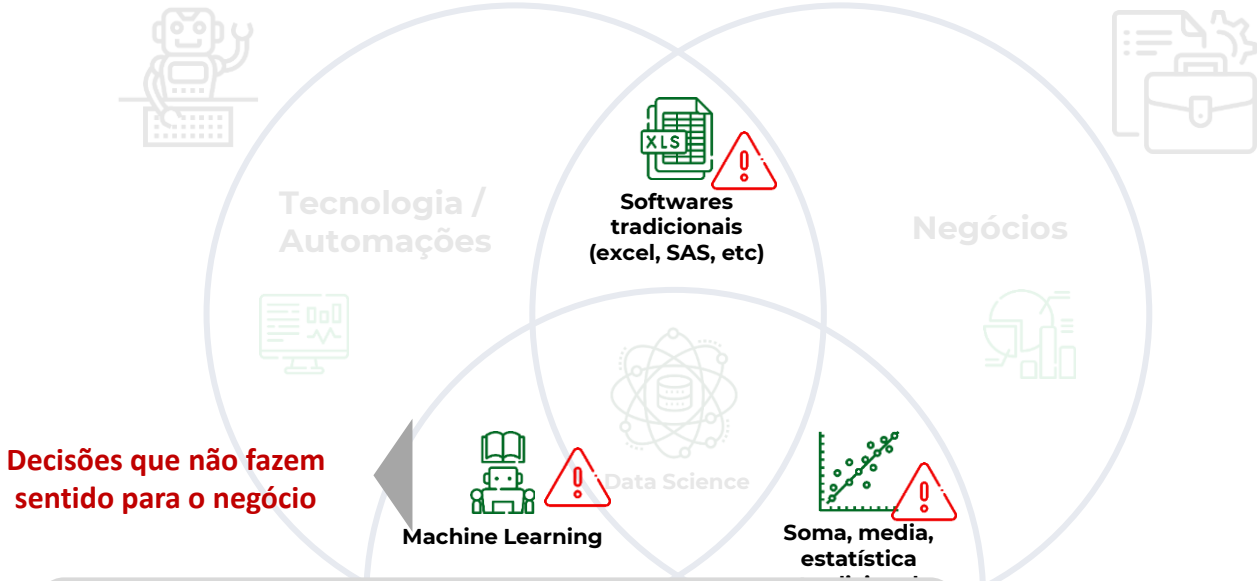
Conclusões que estão corretas mas estatisticamente tem pouca validade



Quando falamos de *Machine Learning* podemos encontrar decisões que não fazem sentido para o negócio.

Por exemplo: análise do canal e o retorno do investimento.

Pelo modelo de *Machine Learning*, a empresa deve usar como principal investimento o Orkut, porém é uma plataforma que não existe mais.



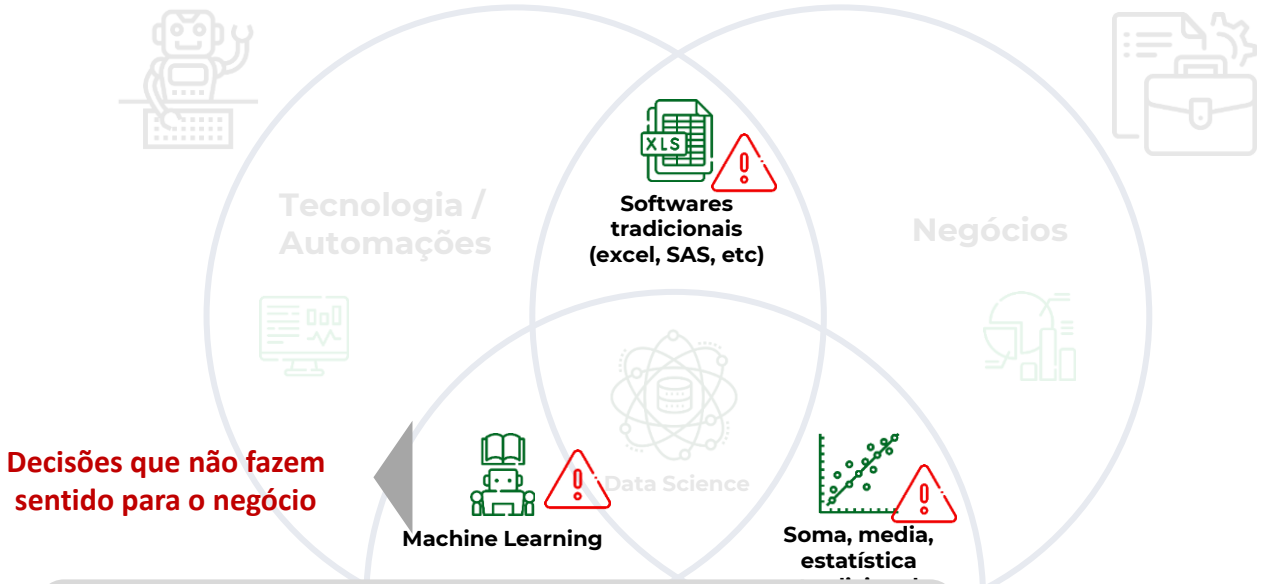
Canal	Retorno do Investimento
TV	0,8
Jornal	1,1
Blogs / Sites	2,7
Orkut	5,3

A empresa deve usar o Orkut como principal investimento para ações de marketing em 2022

Módulo 1 – Os Pilares da Ciência de Dados

O modelo vai usar os dados que ele possui. Então, se foi feito um investimento no Orkut, que em algum momento teve um retorno muito bom, o modelo pode indicar que esse é o melhor investimento.

Então, é necessário ter muito cuidado para garantir que resultado do modelo faça sentido dentro do negócio.



Decisões que não fazem sentido para o negócio

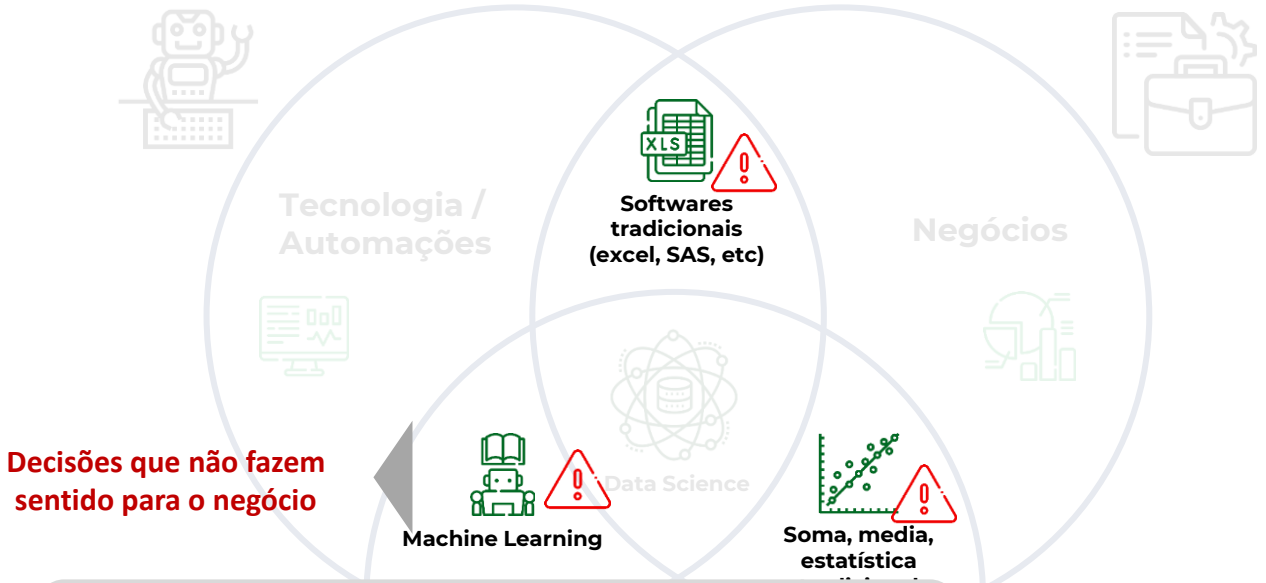
Canal	Retorno do Investimento
TV	0,8
Jornal	1,1
Blogs / Sites	2,7
Orkut	5,3

A empresa deve usar o Orkut como principal investimento para ações de marketing em 2022

Módulo 1 – Os Pilares da Ciência de Dados

Muitas empresas tomam decisões baseado em resultados do Machine Learning sem entender realmente todo o contexto dentro da empresa e pode gerar muito prejuízo.

Nesse exemplo, poderíamos ter evitado essa conclusão criando uma tabela com a informação se a plataforma está ativa.



Decisões que não fazem sentido para o negócio

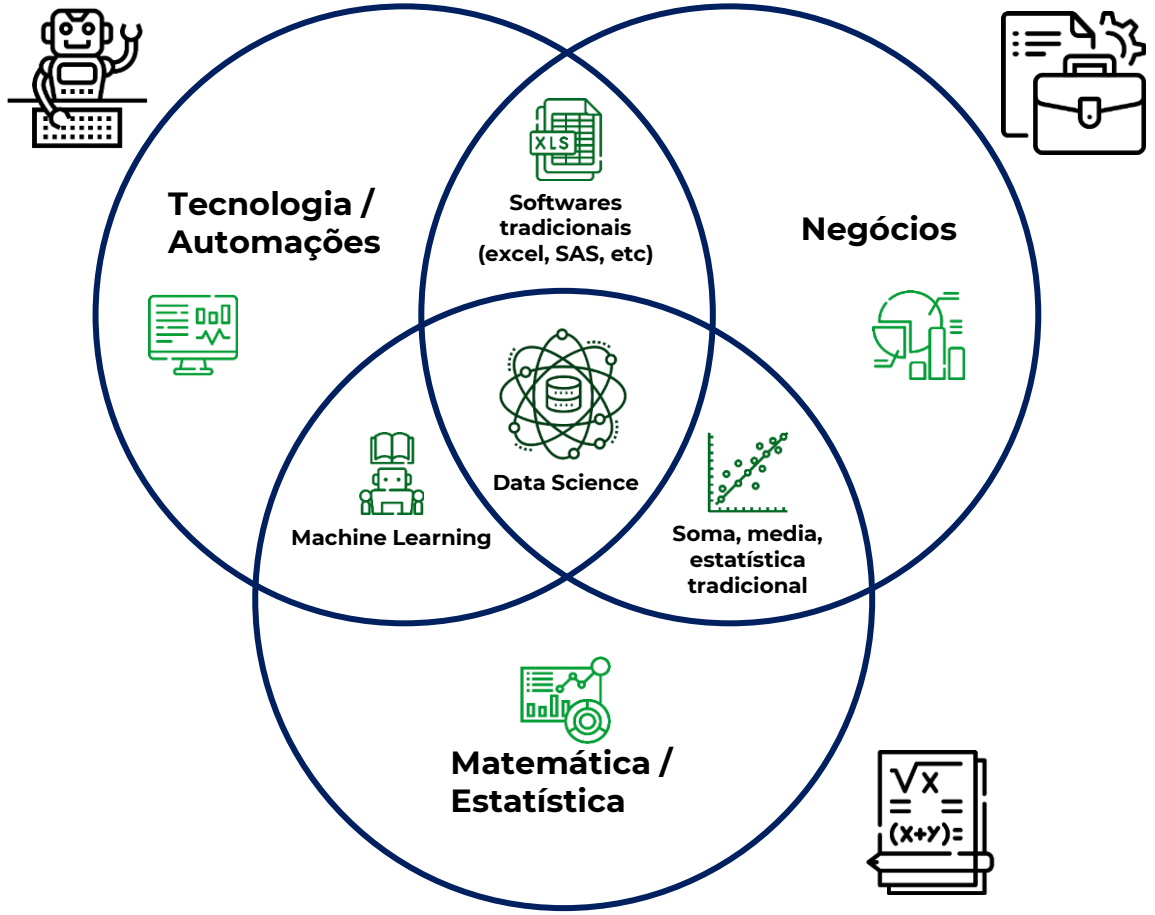
Canal	Retorno do Investimento
TV	0,8
Jornal	1,1
Blogs / Sites	2,7
Orkut	5,3

A empresa deve usar o Orkut como principal investimento para ações de marketing em 2022

Módulo 1 – Os Pilares da Ciência de Dados

O *Data Science* está exatamente na interseção entre essas 3 áreas, não existe uma área que é mais prioritária ou um pilar que é mais importante.

Precisamos entender esses **3 pilares** e como eles trabalham juntos, esse é o grande diferencial no Cientista de Dados.



MÓDULO 2

Introdução a Ciência de Dados

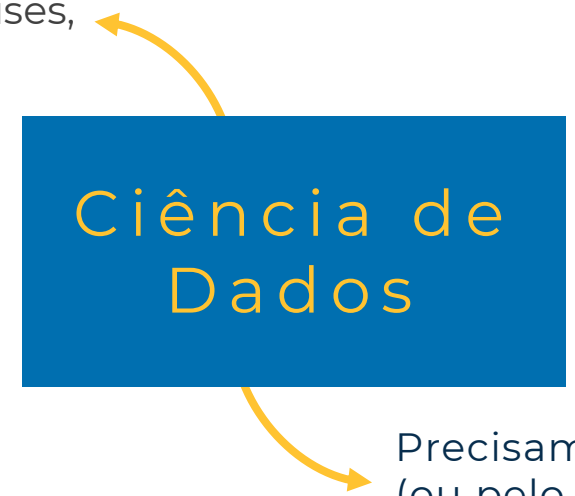
Módulo 2 – O que é ser um cientista?

Ciência é ter um **método científico**, fazer observações e hipóteses.

A primeira coisa do cientista é ele entender que ele não está buscando coisas novas, que ele busca coisas em artigos que já existem, programas que outras pessoas já escreveram.

O **método científico** exige que o nosso estudo consiga ser replicado, ou seja, não adianta fazer um modelo ótimo, nível de certeza alto se aquele modelo não consegue ser replicado.

Existe um método científico! Observação, hipóteses, testes e validação, análises, monitoramento.



Precisamos que existam dados armazenados (ou pelo menos começar esse armazenamento). Armazenamento, processamento, visualização.

Quando falamos em método científico falamos de observação, ou seja, é necessário observar o que está acontecendo na empresa.

O cientista de dados precisa ter muito conhecimento de negócio, precisa ver se o que é observado realmente faz sentido, ou se foi uma coisa muito pontual que não é o padrão.

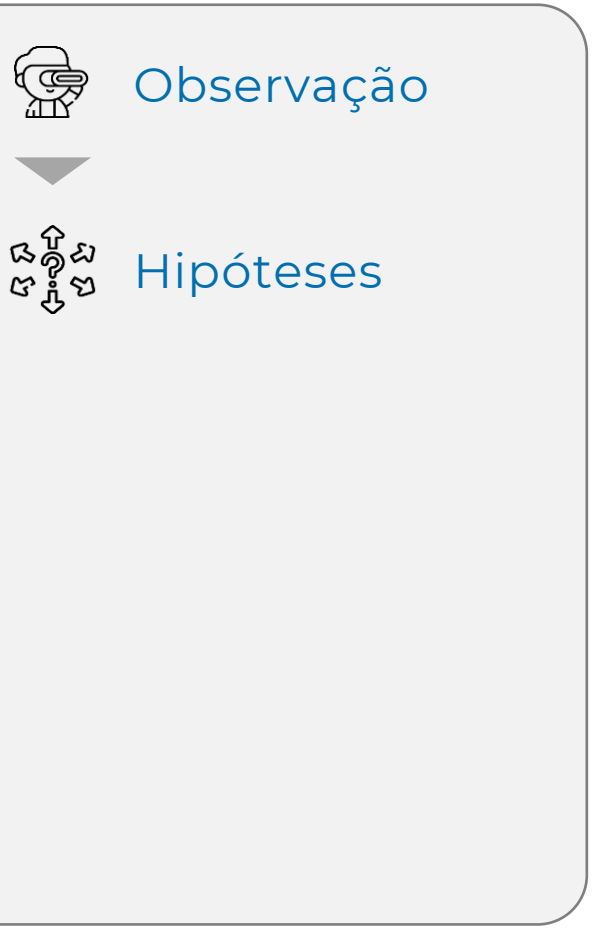
Observação

Módulo 2 – O que é ser um cientista?

É necessário criar hipóteses, pois não podemos fazer várias análises sem realmente pensar em o que queremos analisar.

Então primeiro pensamos em qual hipótese queremos testar e depois vamos analisar essa hipótese, ela pode ser verdadeira ou não.

Se a hipótese não for verdadeira vamos ter que testar outras hipóteses.



Módulo 2 – O que é ser um cientista?

Uma das etapas é fazer experimentos, testar se aquilo realmente funciona ou se vai ser necessário buscar modelos diferentes.

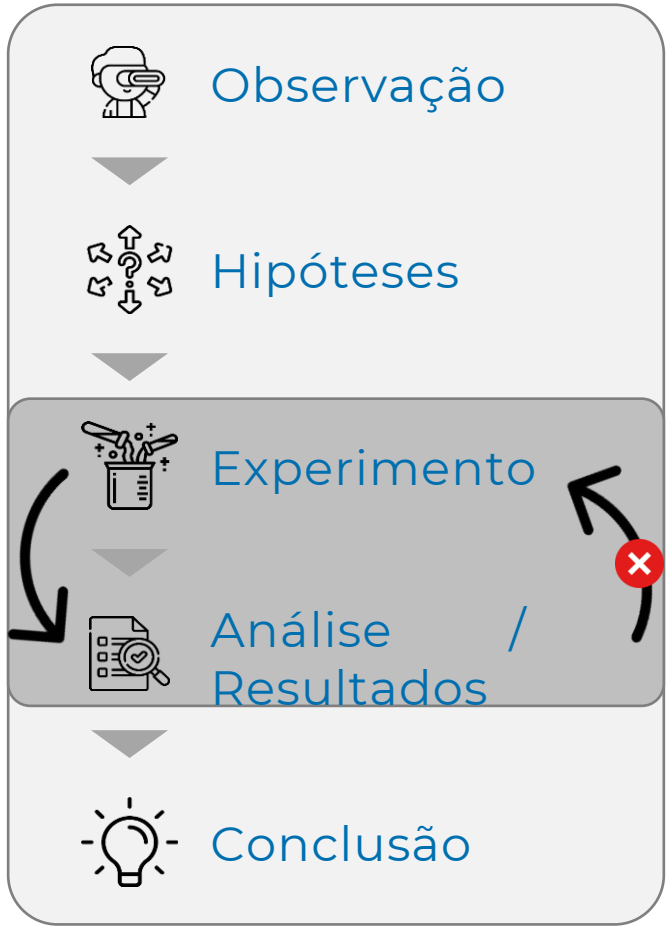
Nessa etapa vamos testar modelos e variáveis.

Depois de testar é necessário analisar e observar quais foram os resultados gerados.

Para então chegar a uma conclusão.

Só que esse processo não é tão linear, nem sempre do resultado vamos direto para a conclusão.

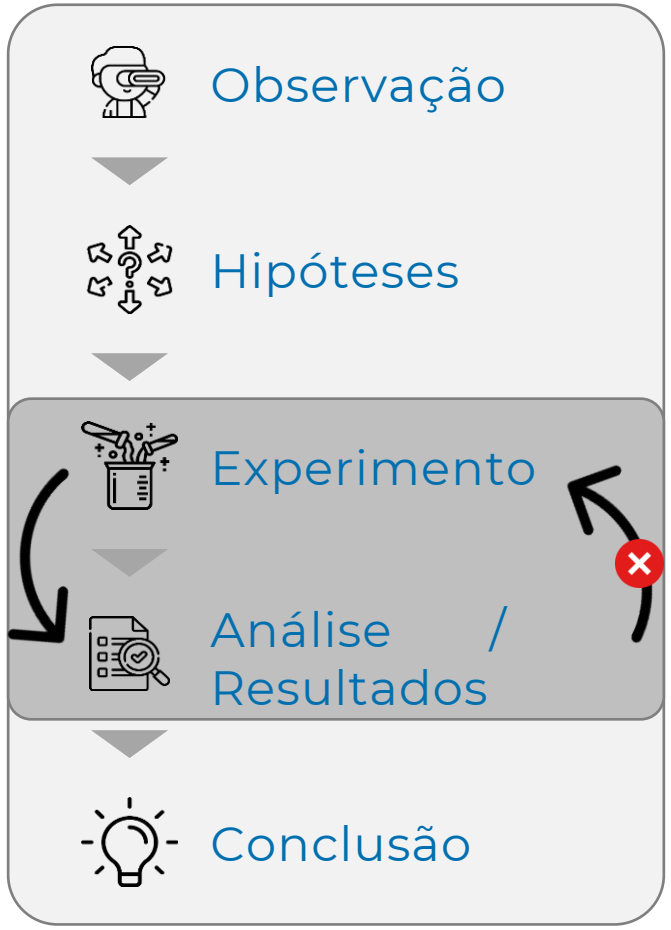
Muitas vezes vamos ficar muito tempo nesse processo de experimentar e analisar. Se o experimento não deu certo, mudamos o modelo ou a variável.



Módulo 2 – O que é ser um cientista?

Esse processo gera uma complexidade muito grande quando falamos de dados, por isso a tecnologia é tão importante para esse processo científico.

Atualmente temos muitos dados por isso precisamos de tecnologia, que no nosso caso será o Python.

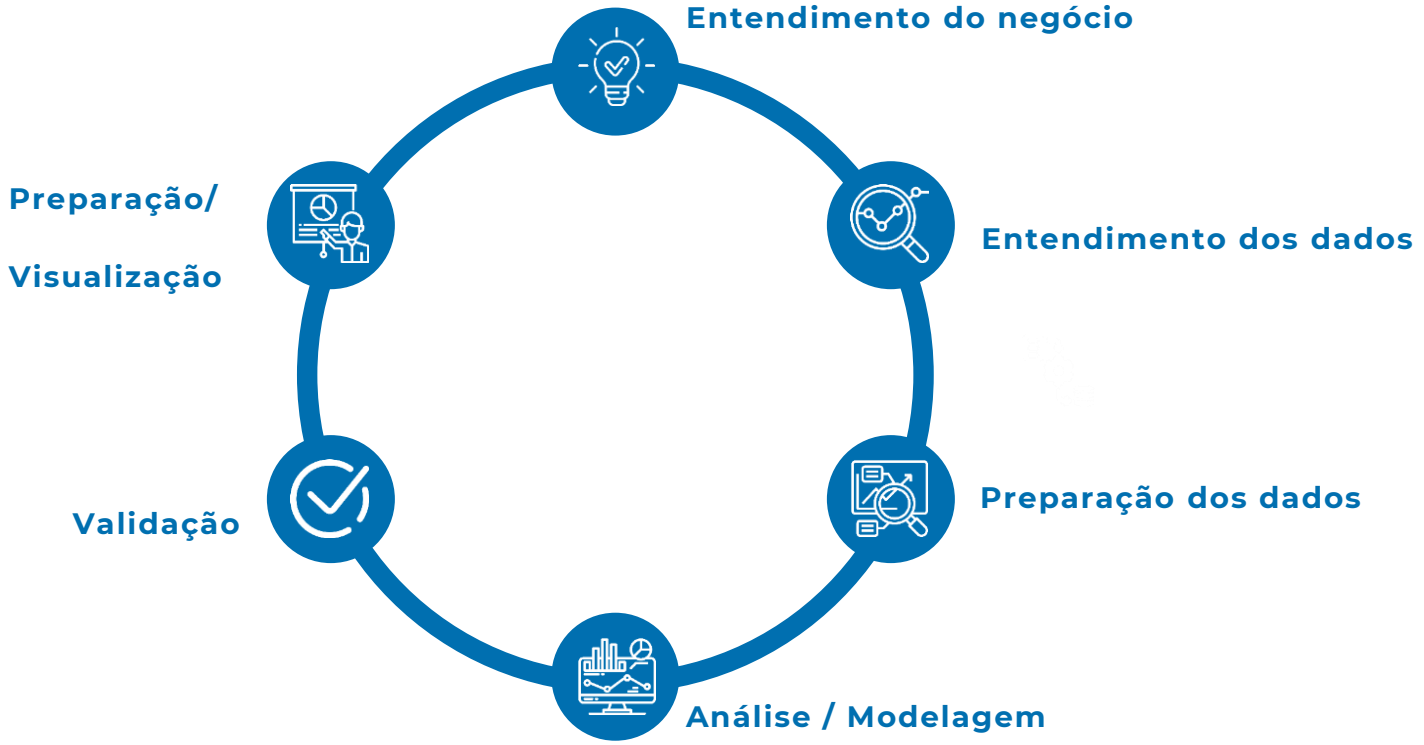


Vamos experimentar muitas coisas usando o modelo de *Machine Learning*, vamos buscar coisas que já existem e bibliotecas já existentes.

Além disso, vamos buscar também um processo que outra pessoa já fez, por exemplo: problema de ruptura na empresa.

Nós vamos buscar em fontes que já tiveram esse mesmo problema, para saber o que a empresa fez, o quê e como ela analisou.

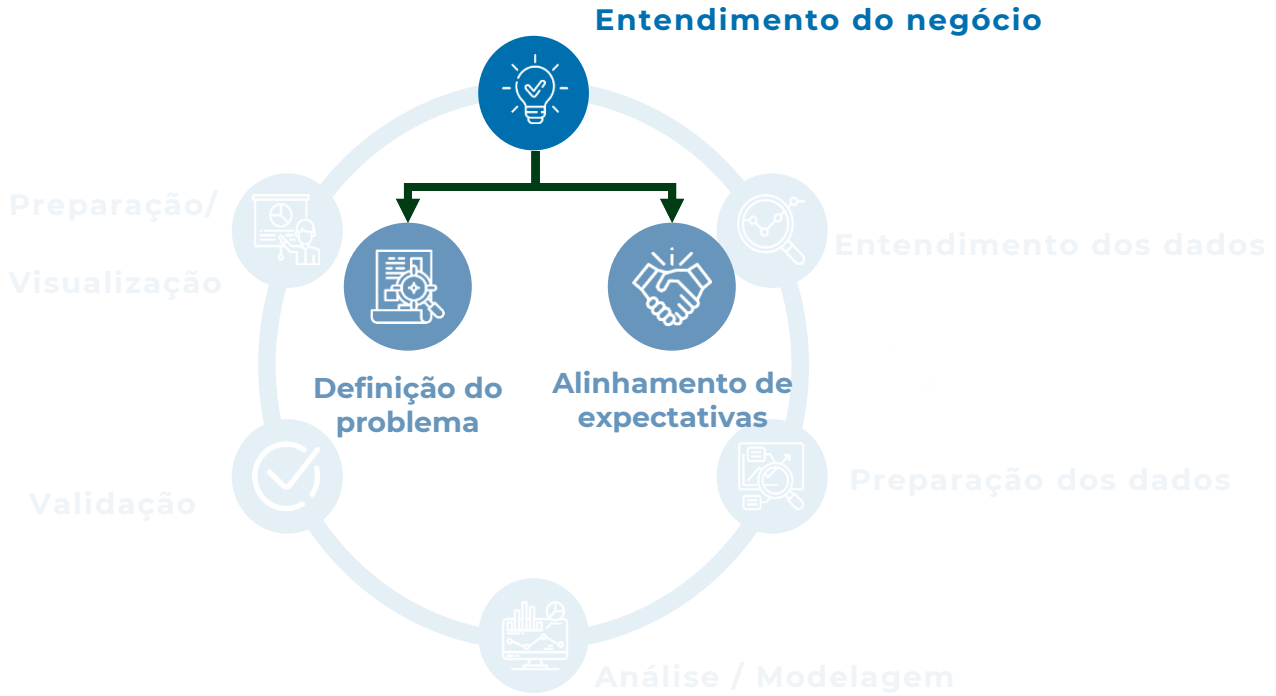
Podemos usar o CRISP-DM (*CRoss Industry Standard Process for Data Mining*) que é um processo existente como base para o nosso framework.



Módulo 2 – Um framework para Ciência de Dados

O entendimento do negócio é divido em 2 etapas:

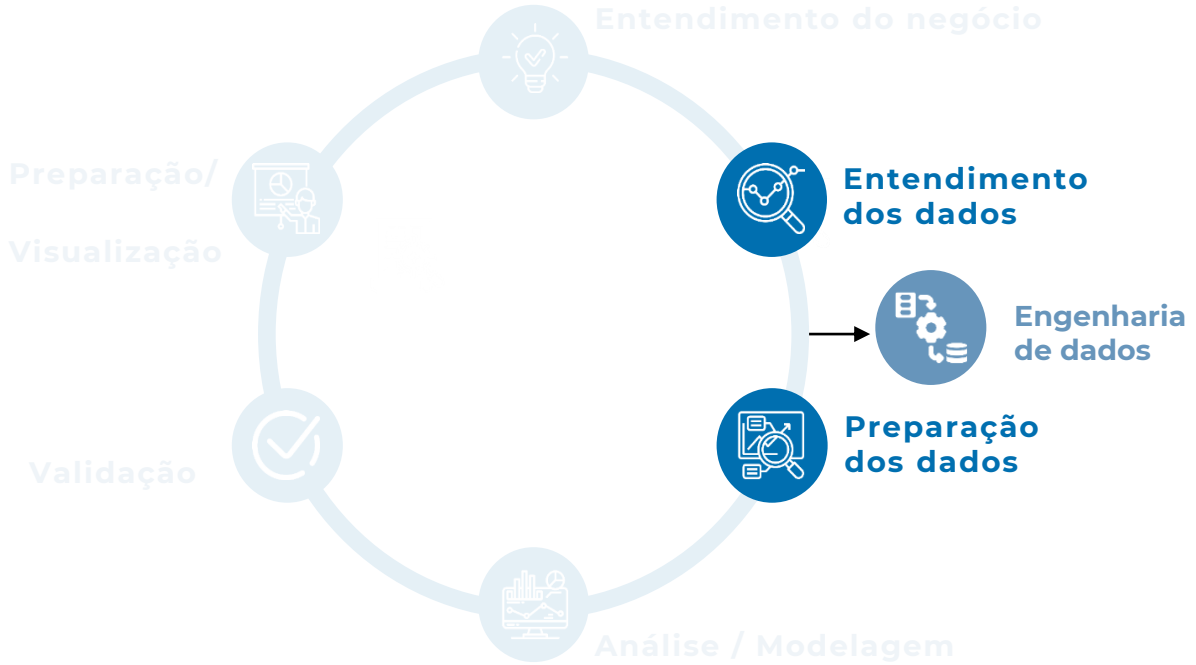
- Definição do problema: entender qual pergunta o seu chefe quer que você responda
- Alinhamento das expectativas: alinhe a precisão para não ter quebra de expectativa.



Módulo 2 – Um framework para Ciência de Dados

Entre o entendimento e a preparação dos dados é necessário fazer a engenharia dos dados, para garantir que a informação vai estar redonda antes de colocar no modelo.

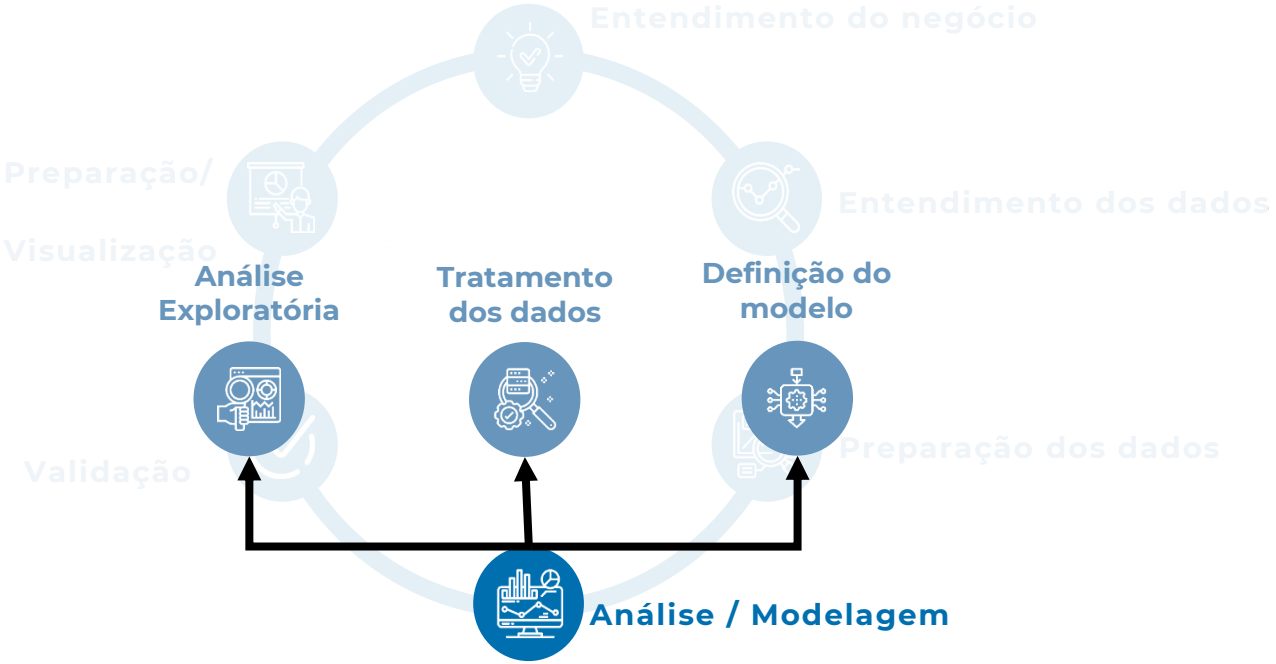
Em seguida tem a preparação de dados para fazer um filtro e diminuir o trabalho no tratamento dos dados.



Módulo 2 – Um framework para Ciência de Dados

A análise e modelagem são dividas em 3 etapas:

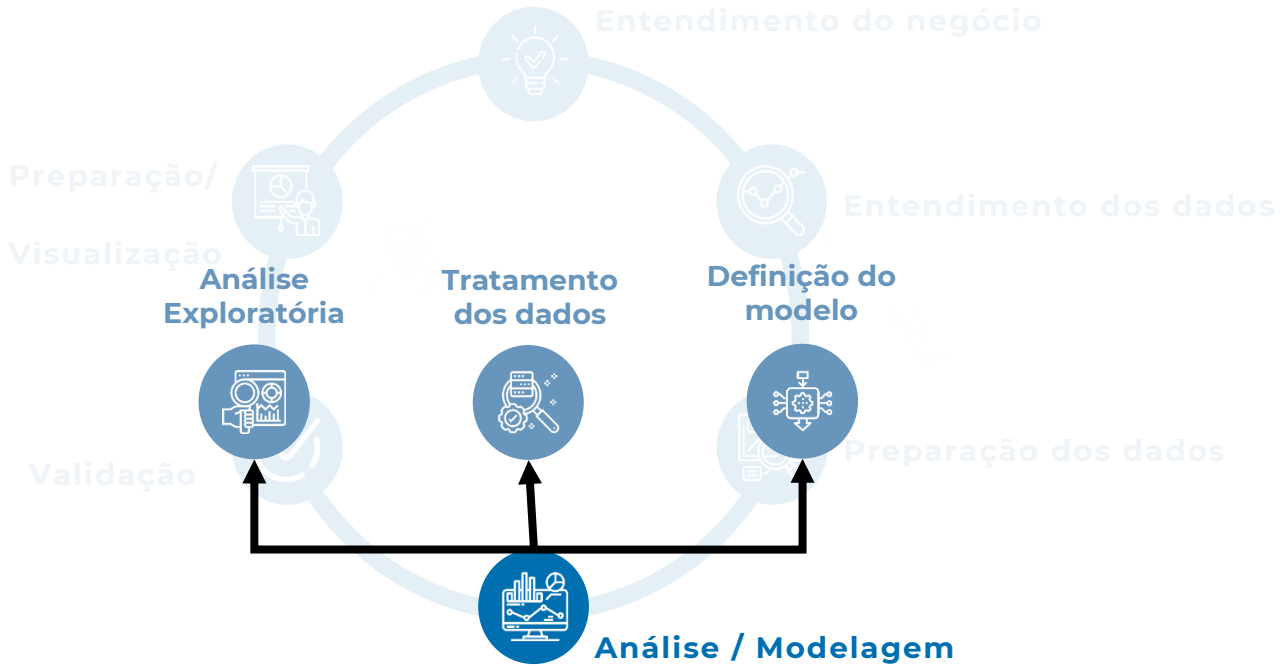
- Análise exploratória
- Tratamento de dados
- Definição do modelo



Módulo 2 – Um framework para Ciência de Dados

Na parte de análise exploratória já conversamos muito com o contratante do projeto.

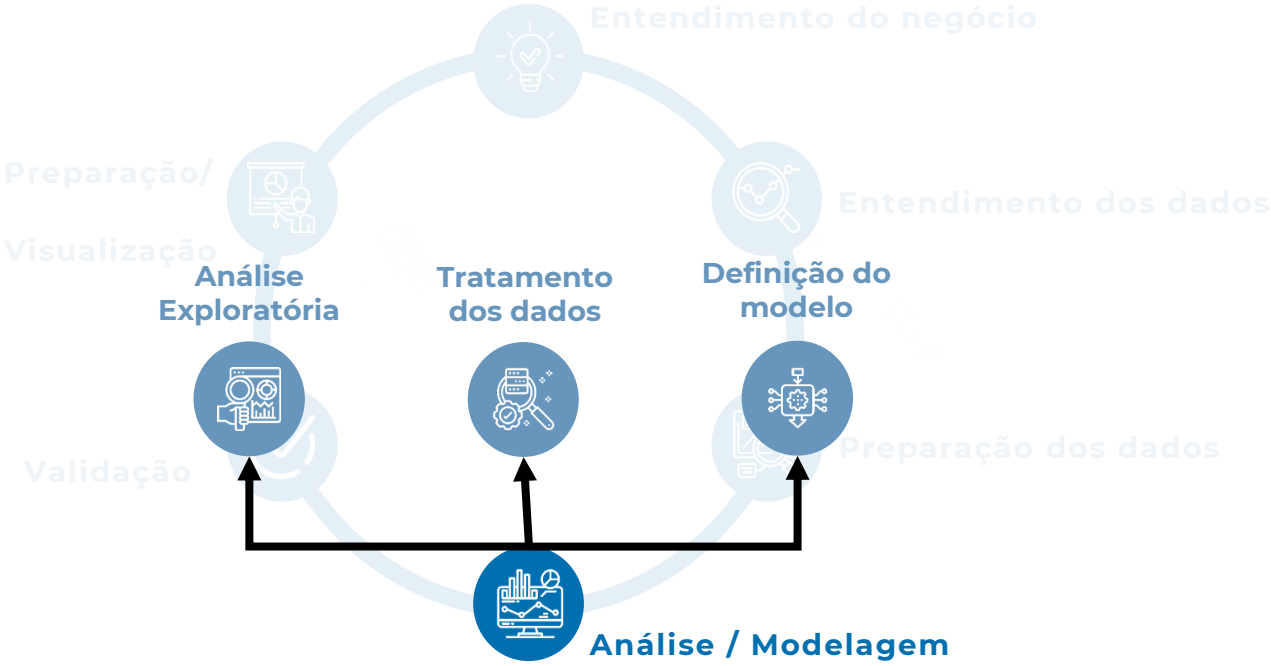
- Essa parte é para entender os dados e o negócio.
- A análise exploratória tem capacidade de começar a gerar valor, por exemplo: descobrir o produto que mais vende.



Módulo 2 – Um framework para Ciência de Dados

Tratamento dos dados é a etapa mais importante, porque se colocarmos dados ruins no modelo, ele vai gerar um resultado ruim.

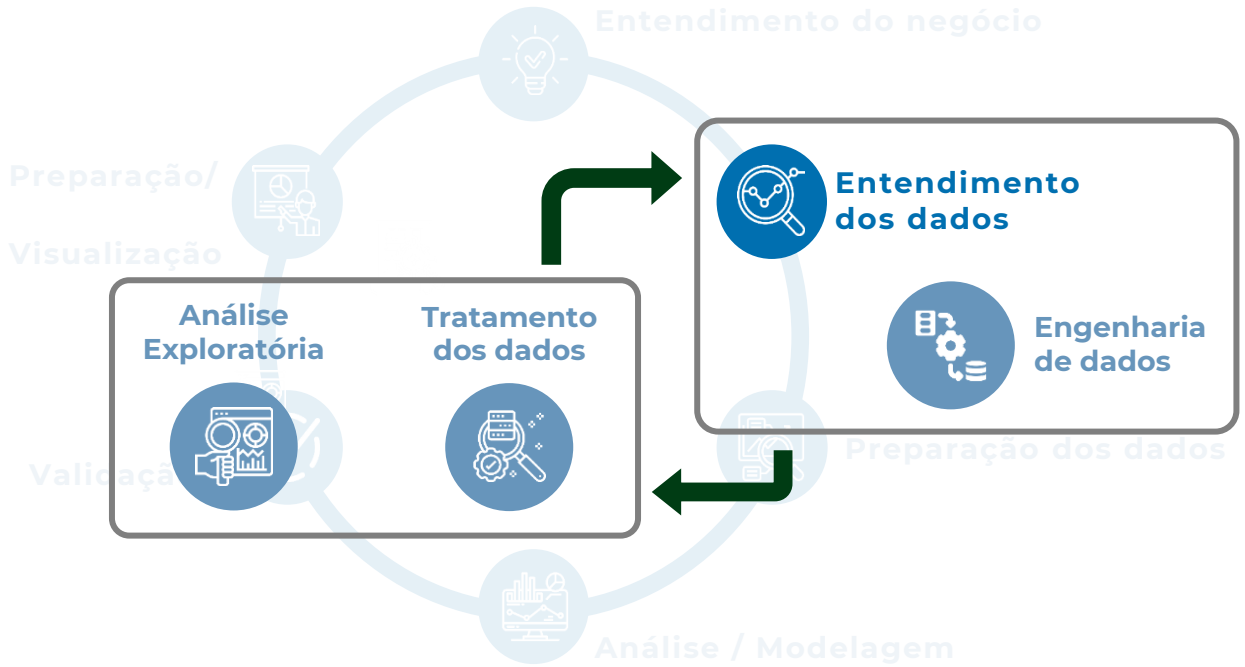
Nessa etapa vamos remover valores nulos que às vezes não foram preenchidos, mas que não podem ser zero. Por exemplo o valor da altura ser zero: nós sabemos que é impossível isso acontecer.



Módulo 2 – Um framework para Ciência de Dados

Essas etapas não são lineares, na maioria das vezes quando fazemos o tratamento de dados percebemos um problema na base ou que precisamos de um dado novo.

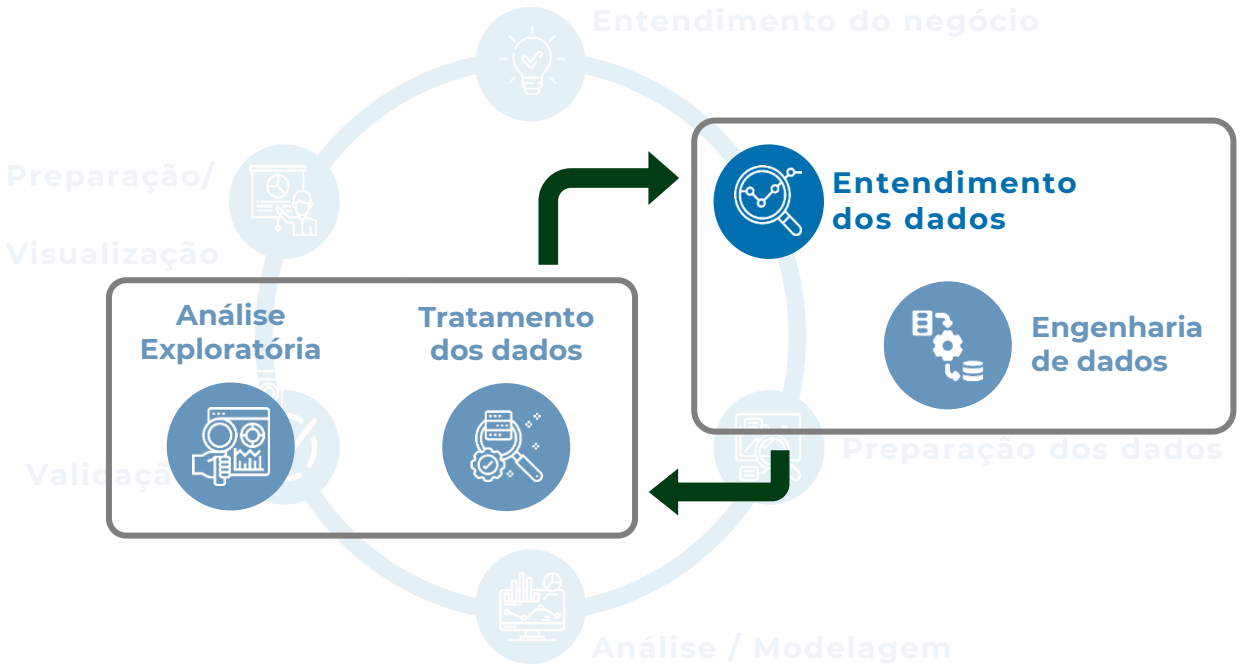
Nesses casos voltamos na etapa de entendimento dos dados, se necessário uma engenharia de dados e refazemos a análise exploratória.



Módulo 2 – Um framework para Ciência de Dados

Essa etapas serão repetidas até alcançarmos a segurança no tratamento de dados para colocar no modelo.

O modelo vai executar, selecionar qual o melhor modelo, selecionar variáveis e escolher o menor erro.



Módulo 2 – Um framework para Ciência de Dados

Depois que o modelo escolhido for testado e validado, vamos para a parte de validação para garantir que o modelo esteja funcionando para os dados de teste e com novos dados.

Vamos fazer isso até achar o modelo perfeito, que não terá uma precisão de 100% e sim a precisão maior que a acordada com o contratante.

Módulo 2 – Um framework para Ciência de Dados

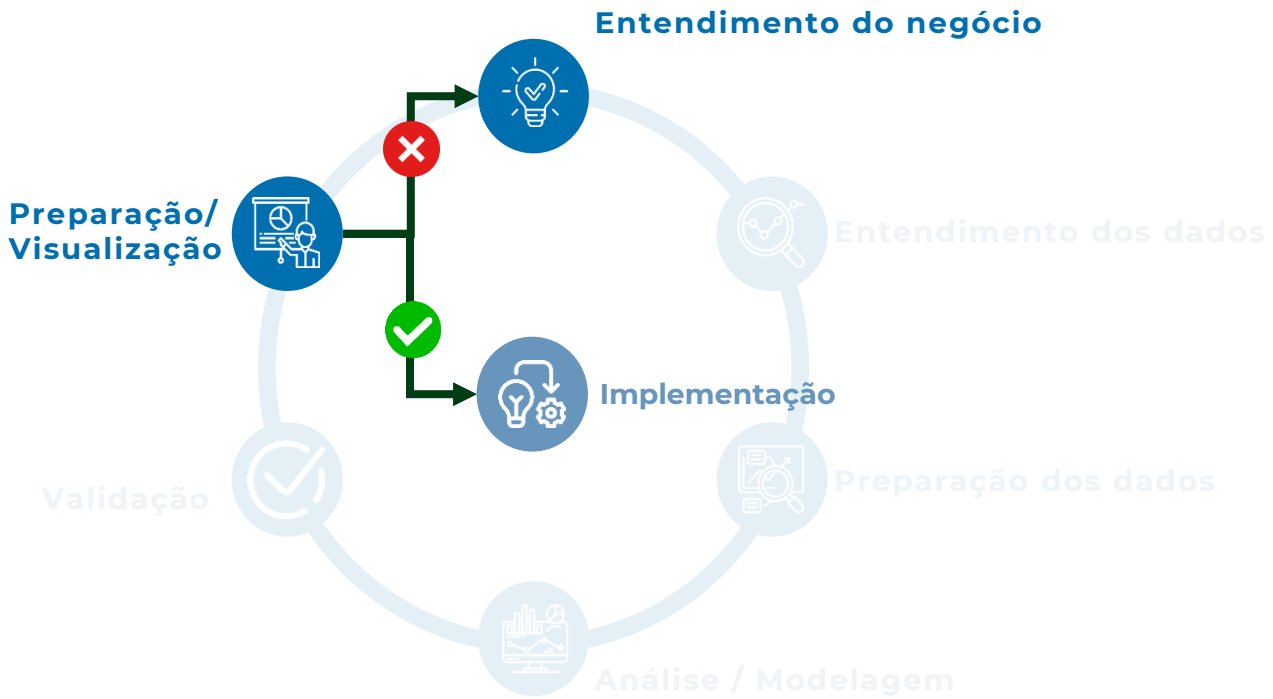
Com o modelo validado, a próxima etapa é apresentar para o contratante com a visualização de dados, gráficos e contar a história.

Nessa etapa não é preciso contar com todos os detalhes os modelos testados, ele quer entender o que vai resolver o problema dele.

Módulo 2 – Um framework para Ciência de Dados

Na apresentação o projeto pode ser aceito ou o contratante pode falar que não era isso que ele queria.

Se for aceita vamos para a implementação, caso contrário precisamos refazer todo o ciclo.



Módulo 2 – Um framework para Ciência de Dados

Com o modelo validado, vamos implementar e colocar na produção, mas por ser um método científico é necessário fazer melhorias e monitorar os dados.

Módulo 2 – Um framework para Ciência de Dados

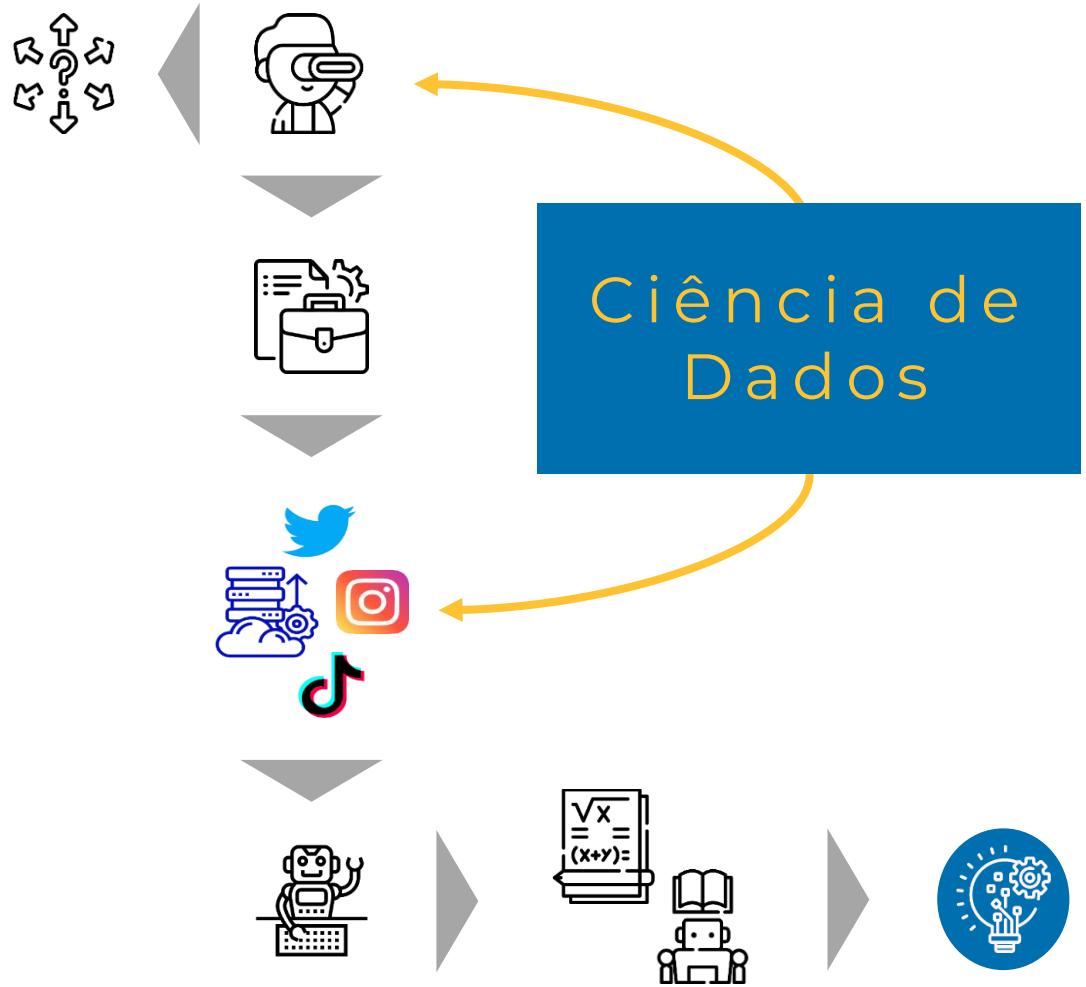
Todas essas etapas são muito importantes para o projeto de *Data Science*.

Módulo 2 – Resumindo Ciência de Dados

A ciência de dados começa na observação e criação de hipóteses, que estão diretamente relacionadas com o negócio.

Depois começamos a trabalhar com as informações utilizando a tecnologia.

Por último, começamos a criar modelos, com a base estatística, para gerar uma conclusão com valor para o negócio.



Módulo 2 – Um framework para Ciência de Dados

A linguagem de programação utilizada durante o curso é o Python, mas você pode usar outro programa pois a estatística é a mesma.

```
In [1]: print("Hello Python!")
        Hello Python!
```

- Mais fácil entendimento e aprendizado
- Comunidade forte, participativa e é mais fácil de achar dúvidas na internet
- Grande crescimento no número de usuários
- Cada vez mais buscada em vagas de trabalho

Módulo 2 – Um framework para Ciência de Dados

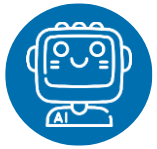
A linguagem de programação utilizada durante o curso é o Python, mas você pode usar outro programa pois a estatística é a mesma.

```
In [1]: print("Hello Python!")
        Hello Python!
```

Usada em todo o processo de Data Science (não precisa de outra linguagem)

Também pode ser usada no processo de ETL (extração, transformação e carregamento)

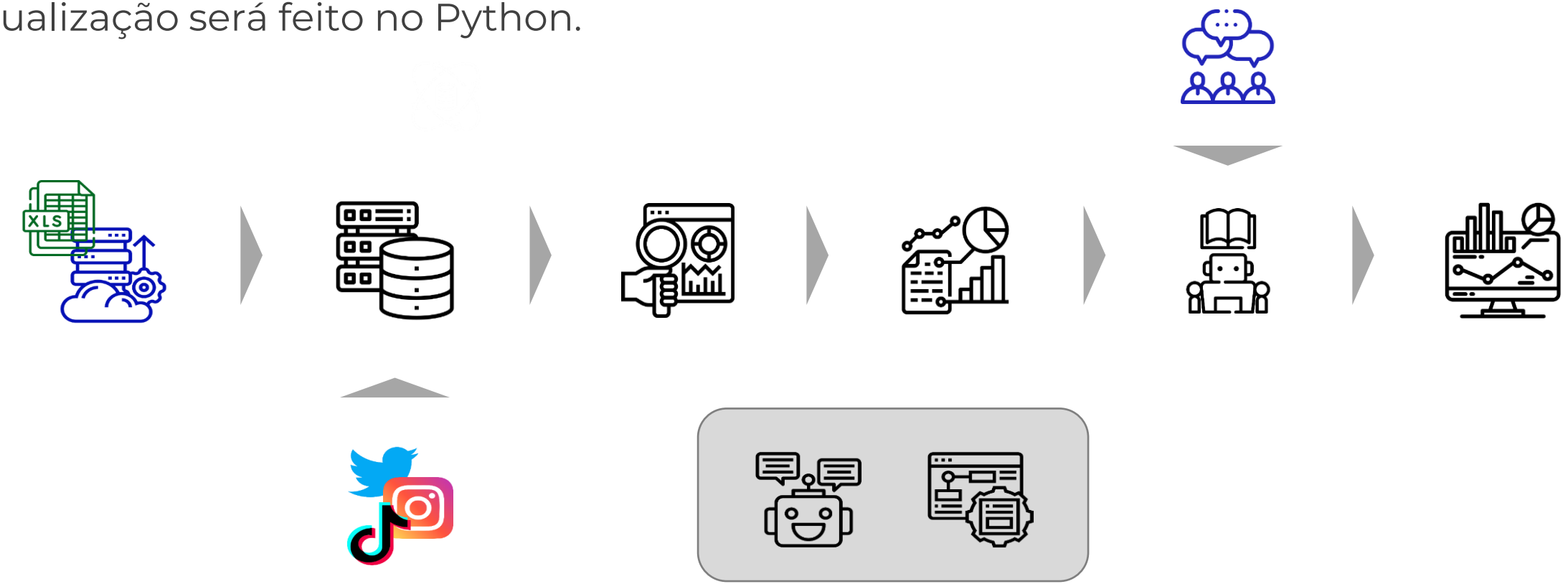
Existência de vários pacotes prontos (como o pandas)



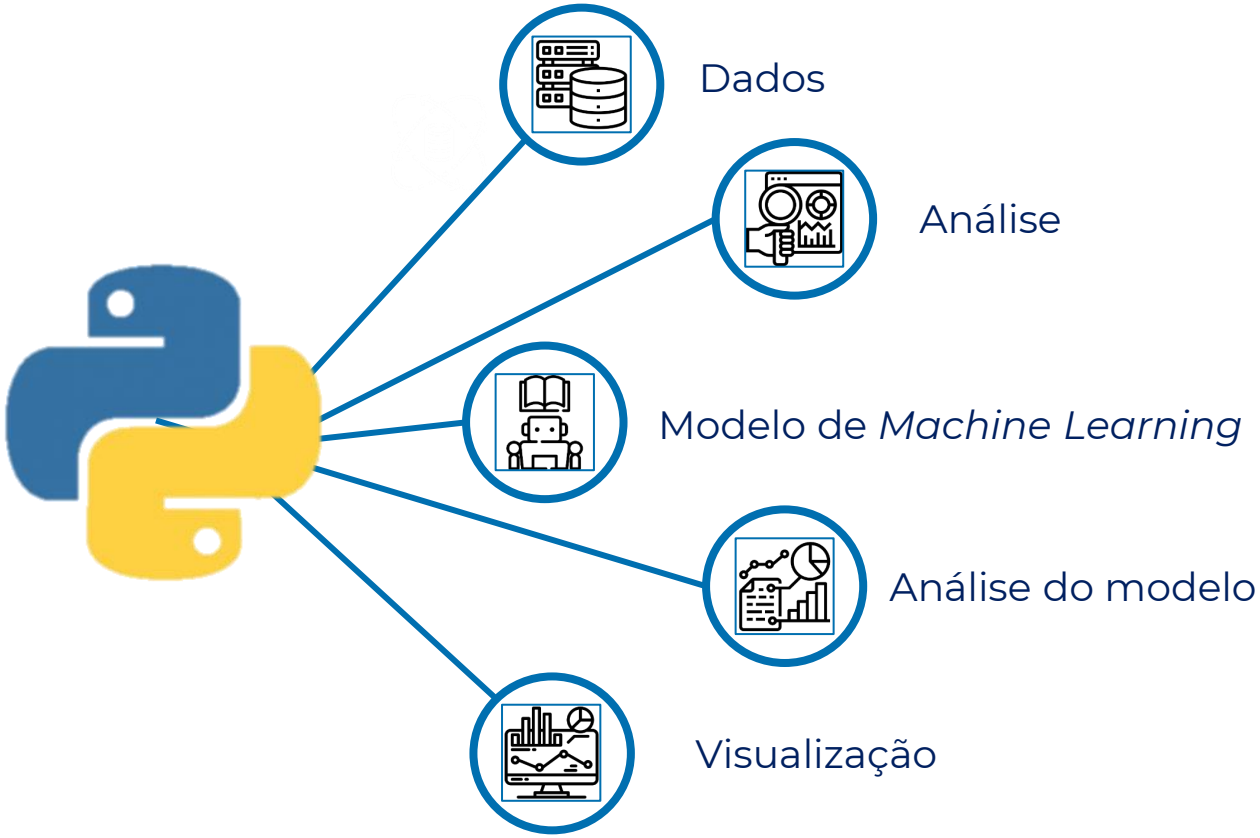
Possibilidade de web scraping, chatbot, etc.

Módulo 2 – Um framework para Ciência de Dados

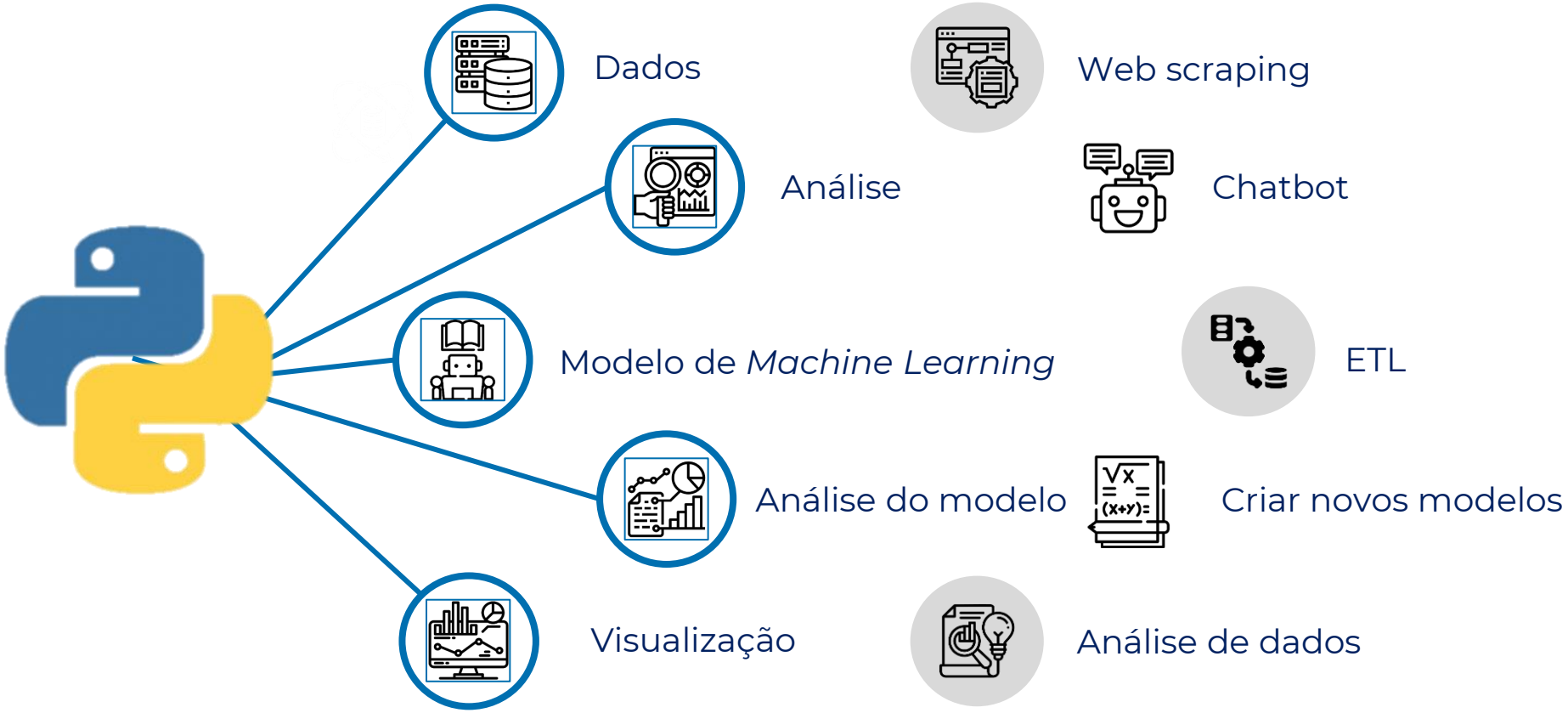
Todo o processo de organização dos dados, análise exploratória, Machine Learning e visualização será feito no Python.



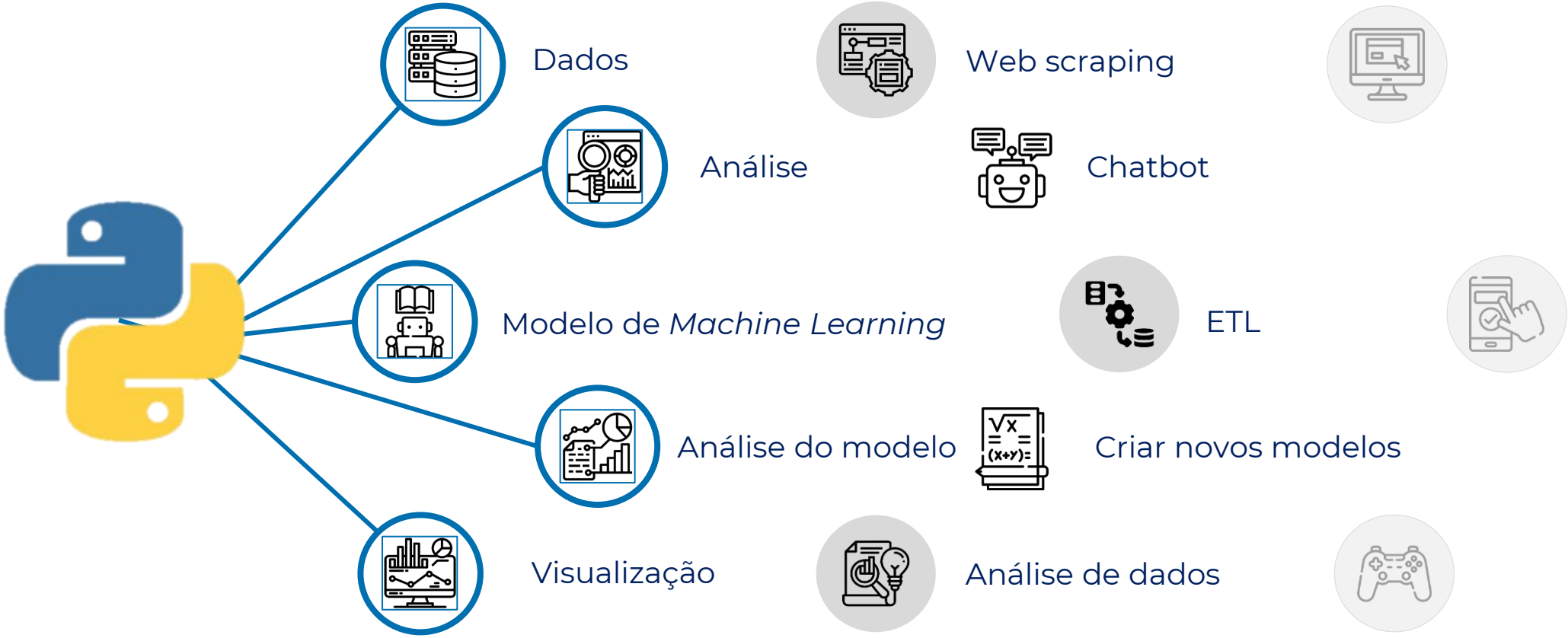
O mercado de trabalho para o cientista de dados e para quem conhece python é muito amplo.



O mercado de trabalho para o cientista de dados e para quem conhece python é muito amplo.



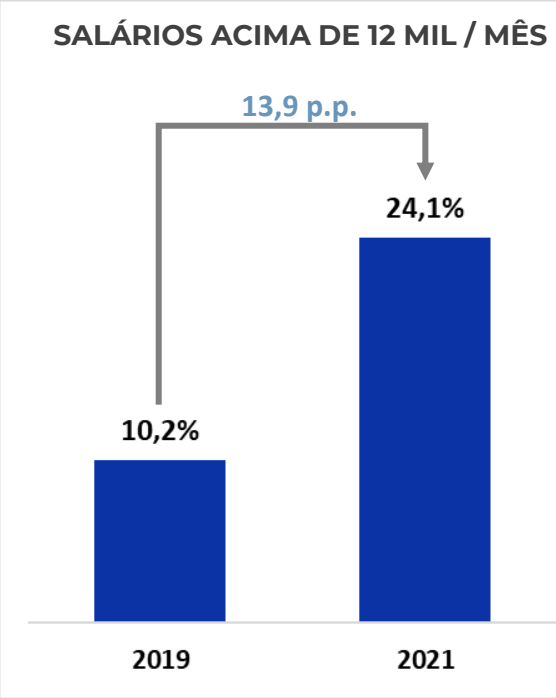
Além disso, com o Python é possível criar jogos, sites e aplicativos.

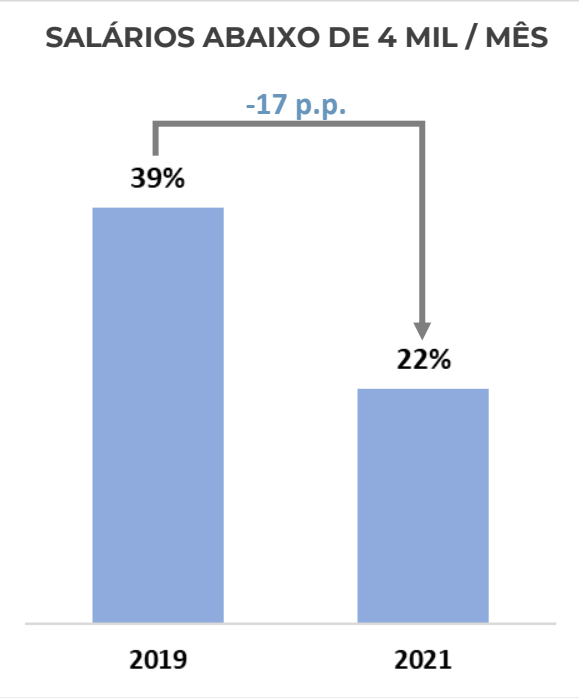


O salário de um cientista de dados teve um aumento de 40%. Além disso, houve um aumento da remunerações de 12 mil e uma redução para remunerações de 4 mil.

40%

de aumento na remuneração entre 2019 e 2021



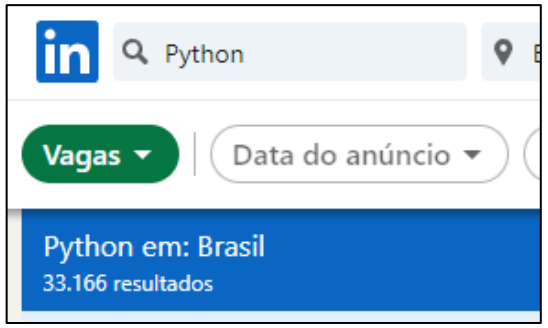


A procura por cientista de dados com conhecimento em Python aumentou, pois é uma linguagem que está crescendo e as empresas estão procurando.

Média salarial mensal de

7.477

+ de 33 mil vagas



Empresas que são referência no mercado:

IBM

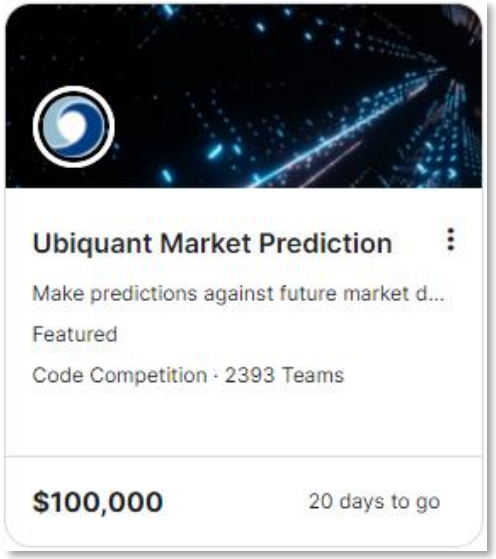
Magalu

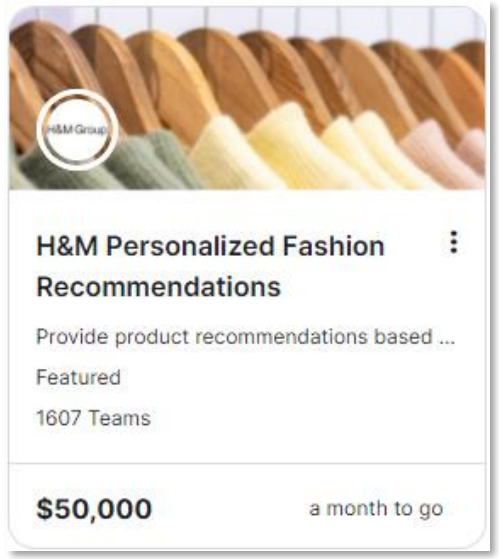
ORACLE

creditas

Além de todas essas áreas, existem competições que dão prêmios em dinheiro.

Desafios de Data Science com remuneração em dinheiro





Competições disponíveis em www.kaggle.com/competitions na data de 29/03/2022

A Netflix deu 1 milhão para quem conseguisse melhorar o seu algoritmo, ou seja, a ciência de dados é muito valorizada e as empresas reconhecem a importância disso, porque elas sabem que isso vai trazer novos clientes.

HOME › GIZMODO ›

Nerds superam sistema de recomendação da Netflix e podem ganhar US$ 1 milhão

Netflix anuncia vencedores do concurso de US$ 1 milhão

22 set 2009 16h03

MÓDULO 3

Python Básico

Módulo 3 – Explicando esse módulo

O módulo 3 tem como objetivo explicar conceitos de Python básico, que serão utilizados para fazer os projetos de Ciência de dados.

Módulo 3 – Instalando o Python no Windows

O Python pode ser instalado pelo link https://www.python.org/downloads/ mas, para esse curso, vamos usar o Anaconda que é o bloco de informações do Python.

Ao instalar o Anaconda será instalado o Jupyter junto, que é onde vamos escrever o código, executar e ele vai nos dizer o que está acontecendo.

No lugar do Jupyter poderia ser utilizado o Google Colab, porém nesse curso vamos utilizar o Jupyter porque ele nos obriga a instalar os programas no computador e não precisa de internet.

Módulo 3 – Instalando o Python no Windows

Para fazer a instalação, entre no link abaixo e escolha o sistema operacional do seu computador:

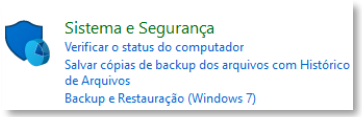
Link: https://www.anaconda.com/products/distribution#Downloads

Módulo 3 – Instalando o Python no Windows

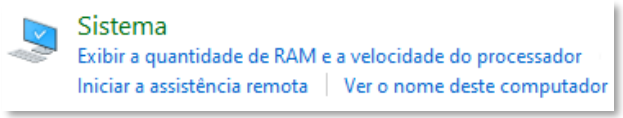
Agora vamos conferir pra ver se está tudo funcionando direitinho:

1. Pesquise por **Painel de Controle** na barra de pesquisa

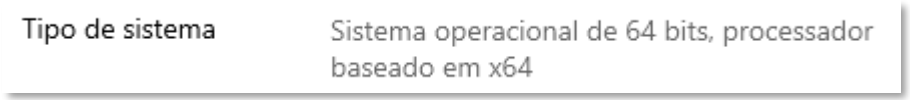
2. Selecione **Sistema e Segurança**



3. Selecione **Sistema**



4. Ao lado de tipo de sistema está escrito o sistema operacional



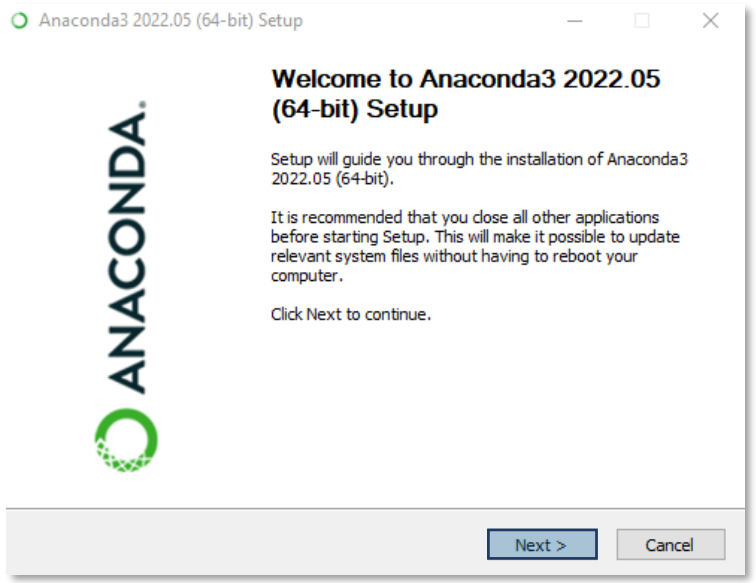
Módulo 3 – Instalando o Python no Windows

Para baixar, clique no **processador** para o seu dispositivo e o download começará diretamente.

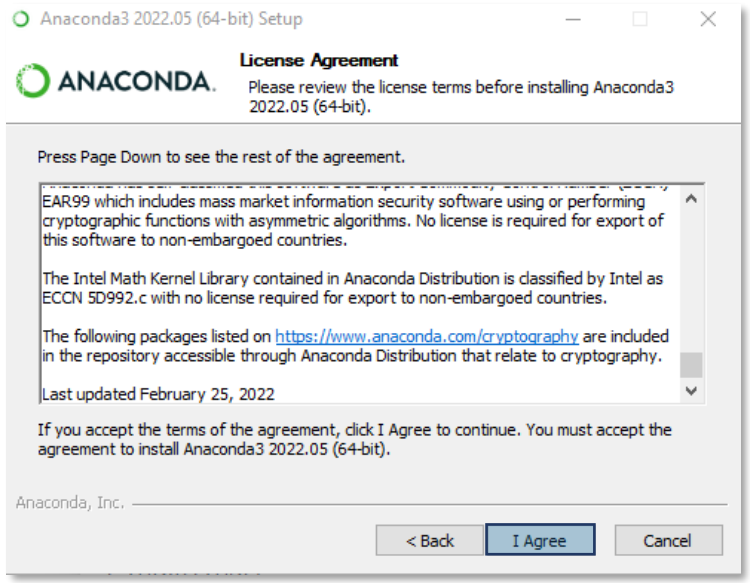
Módulo 3 – Instalando o Python no Windows

Depois que baixar vamos executá-lo no computador.

1. Abra o instalador do Anaconda

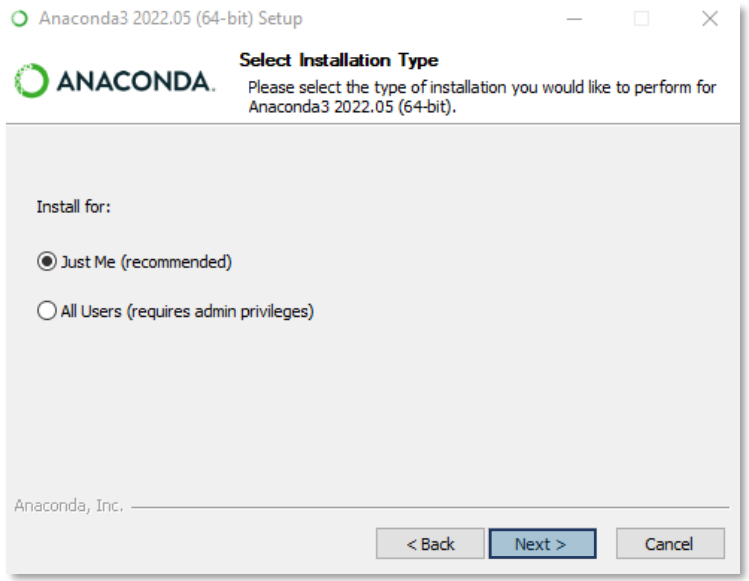


2. Aceite os termos de uso

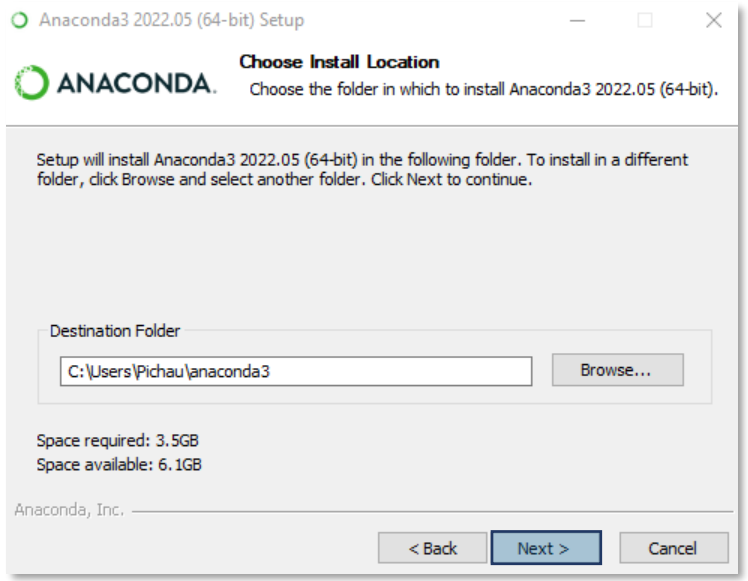


Depois que baixar vamos executá-lo no computador.

3 Selecione o tipo de instalação



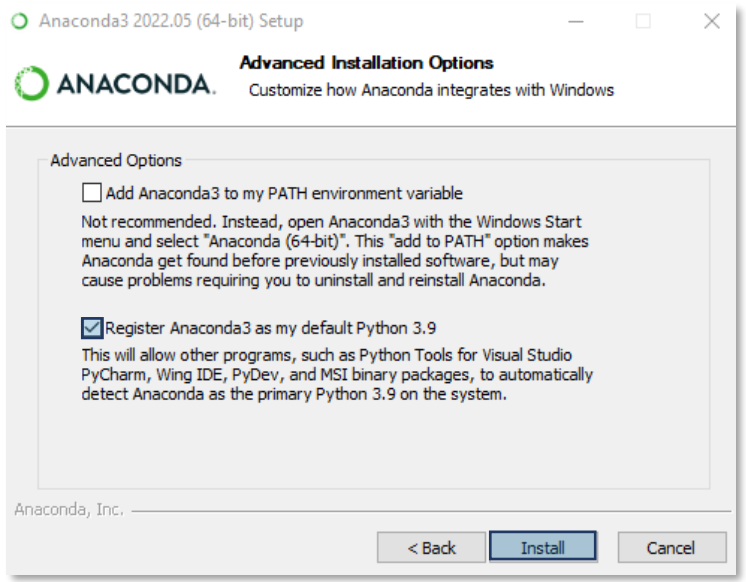
4 Escolha o local de destinação (de preferência o que ele sugerir) e clique em ***next***



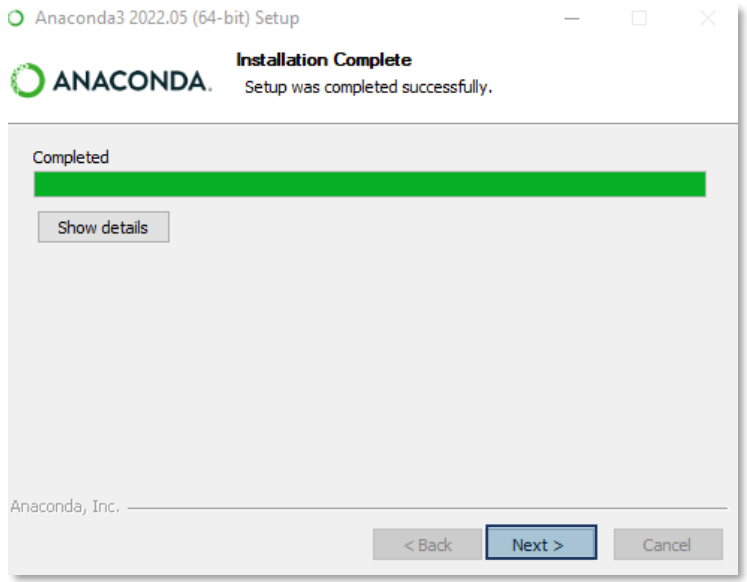
Módulo 3 – Instalando o Python no Windows

Depois que baixar vamos executá-lo no computador.

5 Defina o Anaconda como o seu Python padrão e siga com a instalação clicando em ***install***.

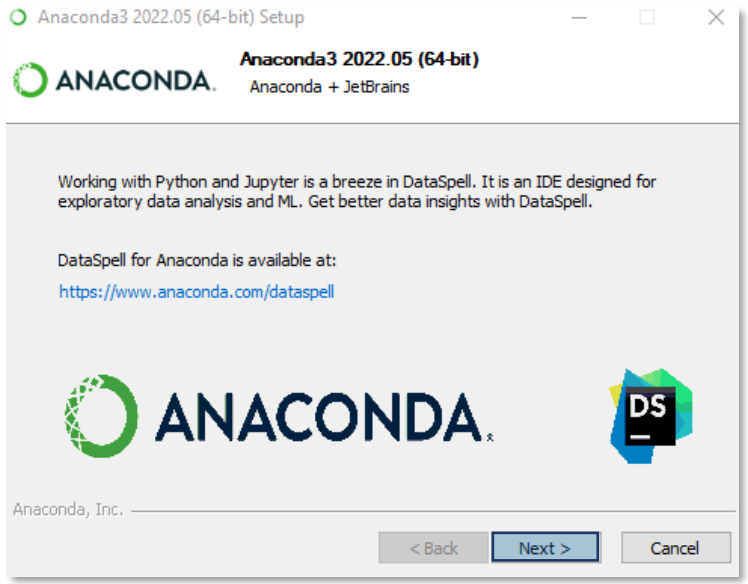


6 Ao fim da instalação, clique em ***next***.

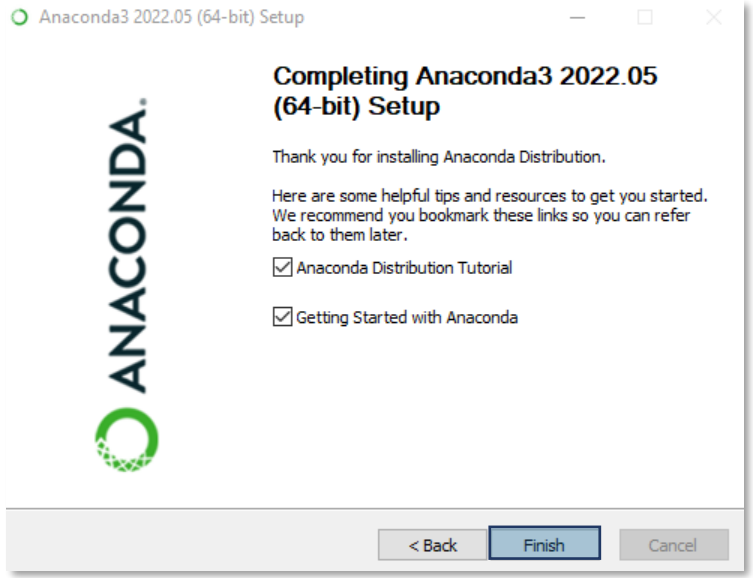


Depois que baixar vamos executá-lo no computador.

7 Clique em ***next***.



8 Ao fim da instalação, clique em ***finish***.



Módulo 3 – Instalando o Python no Windows

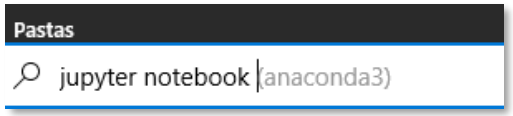
Ao baixar o Anaconda estamos instalando todas essas ferramentas juntas e se precisarmos baixar outra será pelo próprio Python.

Módulo 3 – Instalando o Python no Windows

Agora vamos conferir pra ver se está tudo funcionando direitinho:

1. Na barra de pesquisa, digite Jupyter;

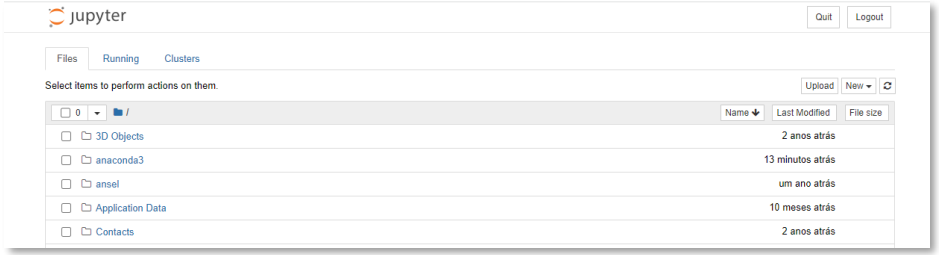
2. Clique no Jupyter Notebook;



3. Vai abrir uma tela preta e carregar algumas informações.

   Não feche a tela preta;

4. O Jupyter irá abrir no seu navegador principal.



Módulo 3 – Problemas na Instalação - Resolvido

Agora vamos resolver os 3 principais problemas que acontecem para abrir o Jupyter:

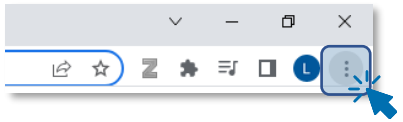
- O Chrome não abre quando eu abro o Jupyter;
- O Jupyter abre na pasta Documentos e eu queria que ele abrisse em outra Pasta;
- Eu instalo o Anaconda e não aparece o atalho do Jupyter

Módulo 3 – Problemas na Instalação - Resolvido

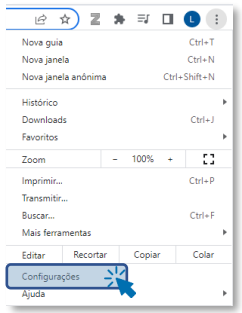
“O meu Jupyter não abre no Google Chrome”

Normalmente o Jupyter abre no seu navegador padrão. Se o Chrome não for o seu navegador padrão, ele não irá abrir no Chrome. Para definir o Chrome como padrão, siga os passos abaixo :

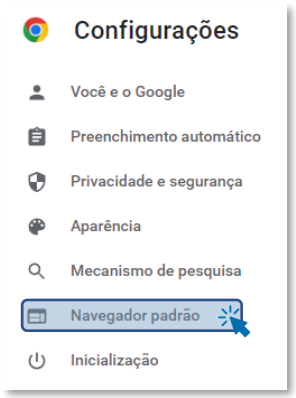
1. Clique nos três pontinhos



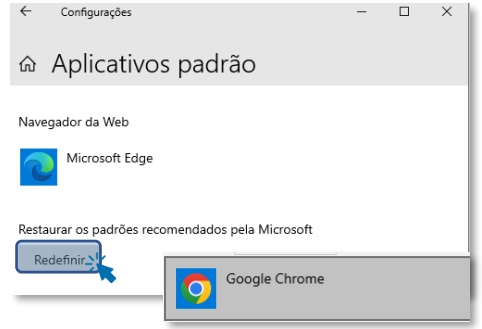
2. Clique em **configurações**



3. Clique em **navegador padrão**



4. Clique em **redefinir** e escolha o **Chrome**

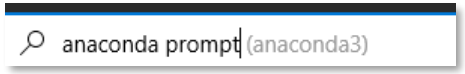


Módulo 3 – Problemas na Instalação - Resolvido

“O meu Jupyter não abre no Google Chrome”

Se mesmo com o procedimento anterior, o Jupyter não abrir no Chrome, então você deve seguir os passos:

1. Abra o Prompt Anaconda



2. Rode o comando:

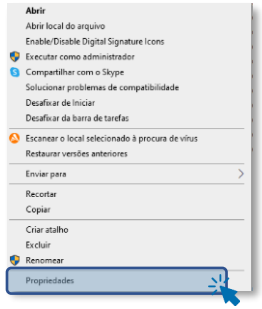
```
jupyter notebook --generate-config
```

3. No bloco de notas abra o arquivo que está no caminho do prompt anaconda

C:\Users\Usuario\.jupyter e abra o arquivo jupyter_notebook_config.py

4. Agora procure o chrome no buscador do Windows, clique com o botão direito do mouse e selecione “Abrir o local do arquivo”.

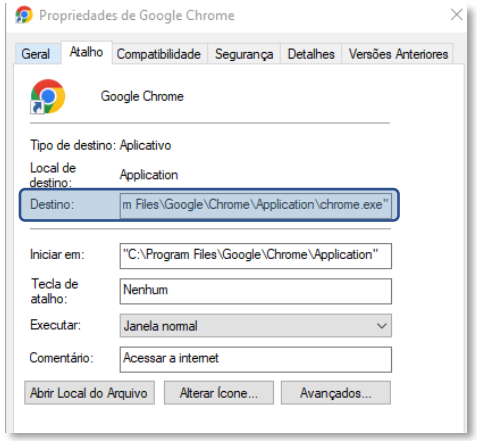
5. Clique com o botão direito do mouse no atalho do chrome e clique em **propriedades.**



Módulo 3 – Problemas na Instalação - Resolvido

“O meu Jupyter não abre no Google Chrome”

6 No campo Destino está exatamente o caminho do local onde está instalado o seu Google Chrome. Copie o caminho todo.



7 Com o arquivo aberto no bloco de notas, procure por “NotebookApp(JupyterApp) configuration” e altere a linha:

c.NotebookApp.browser = ‘’ por c.NotebookApp.browser = ‘ **caminho** %s’

Coloque entre aspas simples **o local** onde está instalado o seu Google Chrome, retire o #, adicione um espaço e %s no final. Salve o arquivo e abra o Jupyter.

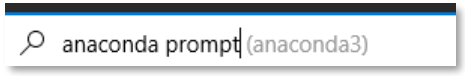
Atenção: Lembre-se de trocar as “\” do caminho por “/”.

Ex: c.NotebookApp.browser = 'C:/Program Files (x86)/Google/Chrome/Application/chrome.exe %s’

Módulo 3 – Problemas na Instalação - Resolvido

“O Jupyter abre na pasta Documentos e eu queria que ele abrisse em outra Pasta”

1. Abra o Prompt Anaconda



2. Rode o comando:

jupyter notebook --generate-config

3. No bloco de notas abra o arquivo que está no caminho do prompt anaconda

C:\Users\Usuario\.jupyter e abra o arquivo jupyter_notebook_config.py

4. Com o arquivo aberto no bloco de notas, procure por “NotebookApp(JupyterApp) configuration” altere a linha:

#c.NotebookApp.notebook_dir = ‘’ por

c.NotebookApp.notebook_dir = r‘**Local**’

Exemplo: c.NotebookApp.notebook_dir = r'**E:**’

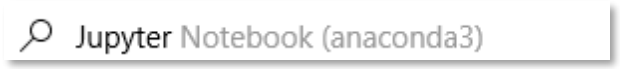
(Você deve colocar entre aspas simples o caminho da pasta onde você quer que o Jupyter abra. No caso do exemplo, o jupyter abrirá no disco E)

5. Salve o arquivo

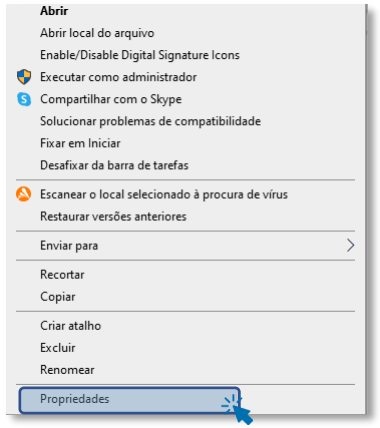
Módulo 3 – Problemas na Instalação - Resolvido

“O Jupyter abre na pasta Documentos e eu queria que ele abrisse em outra Pasta”

6 Salve o arquivo e procure o Jupyter no buscador do Windows, clique com o botão direito do mouse e selecione “Abrir o local do arquivo”.



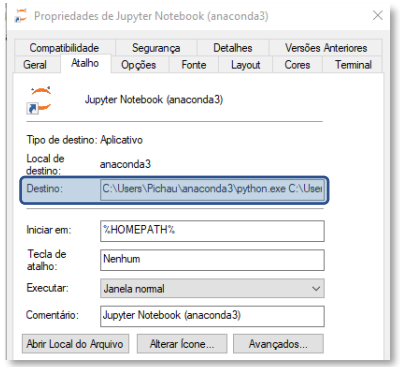
7 Clique com o botão direito do mouse no atalho do Jupyter e clique em **propriedades.**



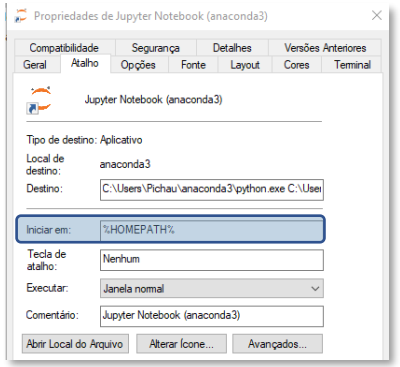
Módulo 3 – Problemas na Instalação - Resolvido

"O Jupyter abre na pasta Documentos e eu queria que ele abrisse em outra Pasta"

9 No campo "Destino" delete "%USERPROFILE%/".



10 No campo "Iniciar em" escreva o caminho do diretório onde deseja iniciar.



11 Feche e abra novamente o Jupyter e tudo deve funcionar normalmente.

Módulo 3 – Problemas na Instalação - Resolvido

"Eu instalo o anaconda e não aparece o atalho do jupyter'"

Geralmente o problema aqui está relacionado a algum problema na versão do anaconda/SO/arquitetura do seu computador.

1. Desintale o Anaconda e **REINICIE O COMPUTADOR.**

   Painel de Controle > Desinstalar

   (É sério, tem que reiniciar o computador)

2. Baixe novamente o Anaconda e instale ele, mas na opção de escolher os usuários escolha a opção **"All Users"** ao invés de "Just me"

3. Caso não resolva desintale o Anaconda e **REINICIE O COMPUTADOR.**

   (É sério, tem que reiniciar o computador)

4. Depois entre nesse link que tem todas as versões que saíram até hoje: https://repo.anaconda.com/archive/

   Geralmente recomendo baixar essa aqui: Anaconda3-2019.10-Windows-x86_64.exe mas se não funcionar continue testando.

Módulo 3 – Problemas na Instalação - Resolvido

"Eu instalo o anaconda e não aparece o atalho do jupyter'"

5 Sempre que for testar uma nova versão é importante que você **desinstale** o Anaconda antes, **reinicie** o computador (sim, toda vez que desinstalar você precisa reiniciar) e instale uma nova versão.

6 Alguma versão (sobre tudo das mais recentes) deve funcionar normalmente no computador.

Módulo 3 –Mac, Linux e Google Colab

O procedimento para fazer a instalação no Mac e no Linux é o mesmo de fazer a instalação no Windows. Para fazer a instalação entre no link abaixo e escolha o sistema operacional do seu computador:

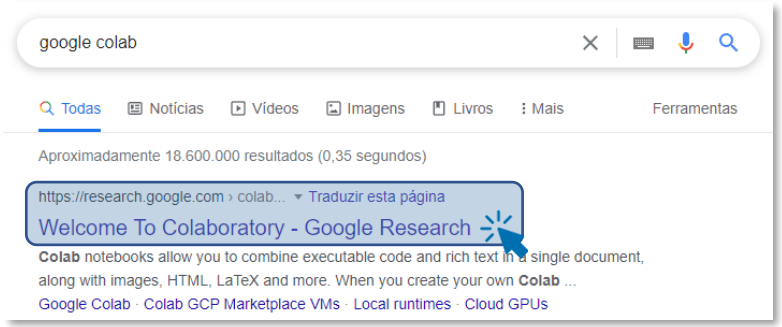
Link: https://www.anaconda.com/products/distribution#Downloads

Opção para instalação mais intuitiva

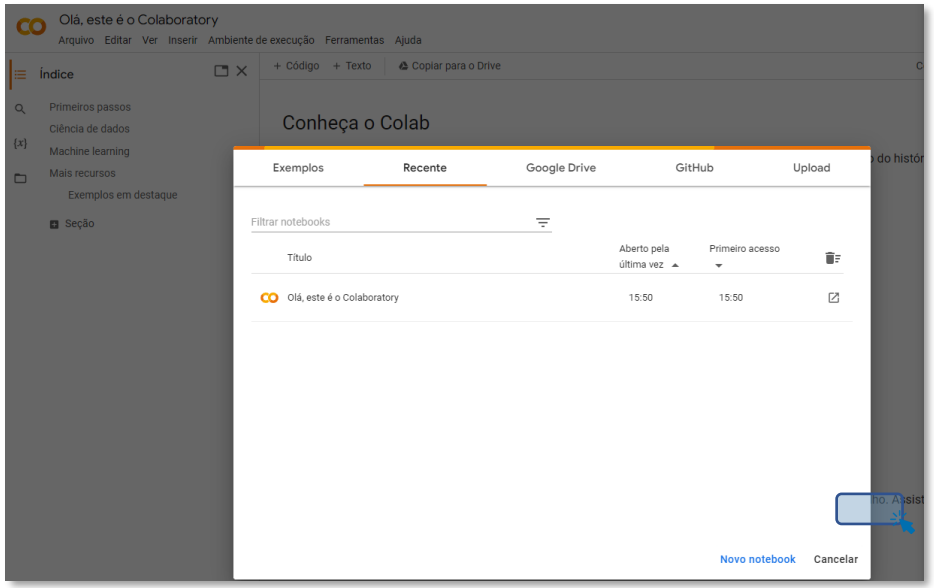
Uma alternativa para quem não quer baixar nenhum programa é o **Google Colab**.

Ele é bem simples de usar e só precisa de **internet** e uma conta no **gmail**.

1. Procure por **google colab** e clique no primeiro site

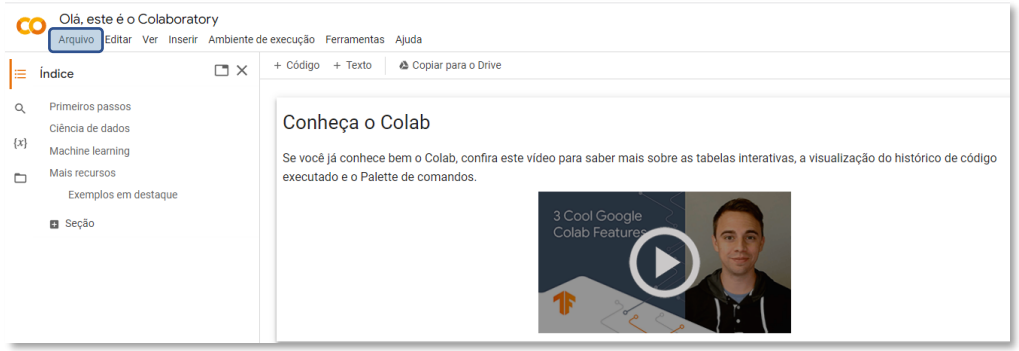


2. Clique em cancelar para ir para a página inicial



Para criar um arquivo basta seguir o passo a passo abaixo:

1 Clique em **arquivo**

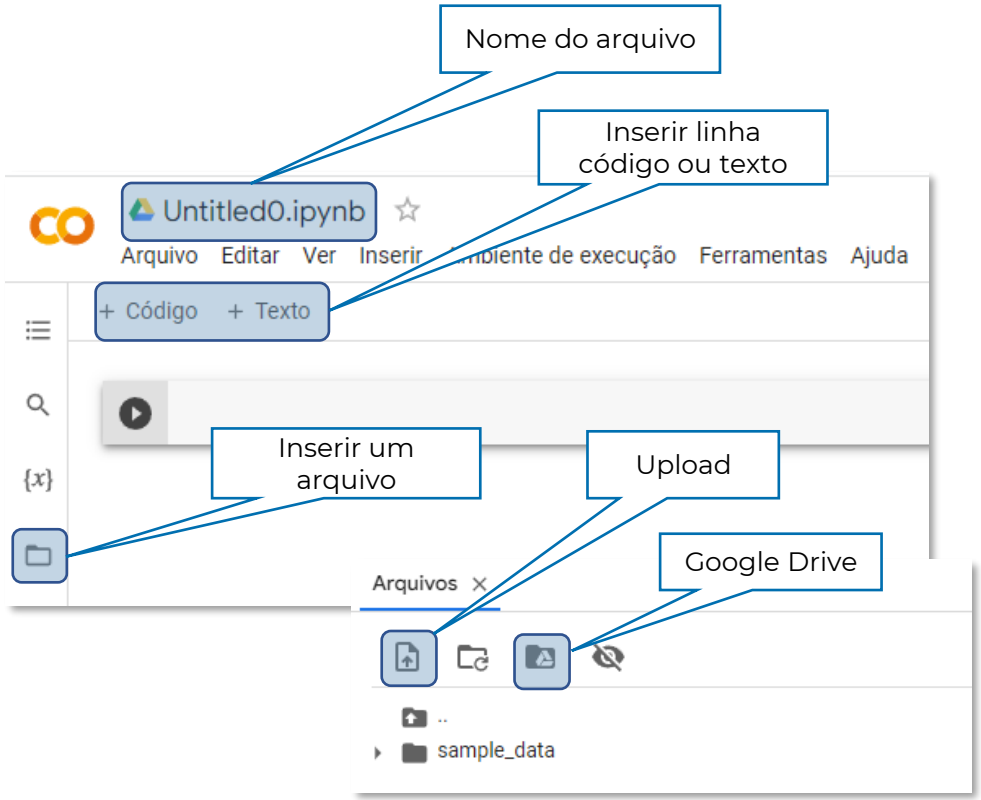


2 Clique em **novo notebook**.

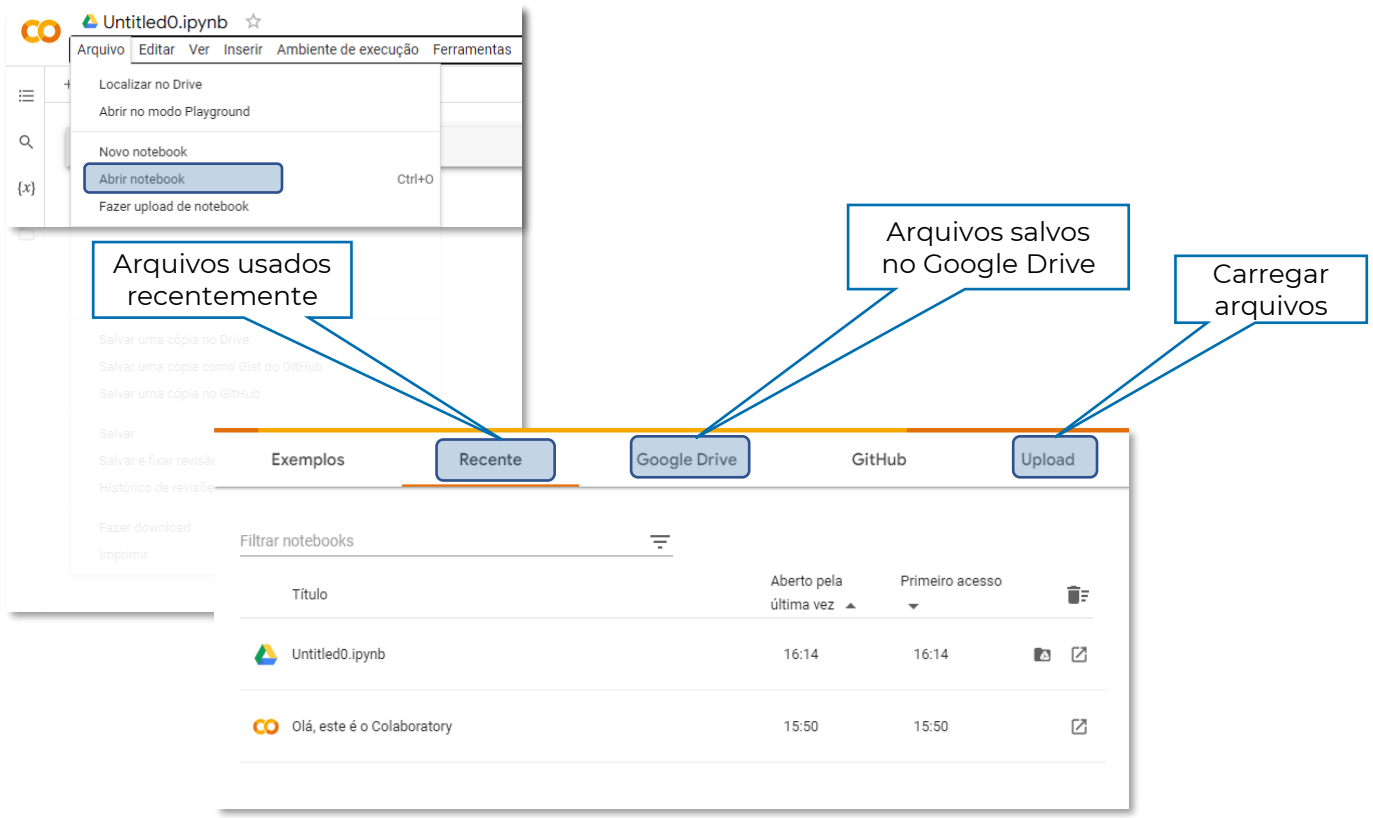
Após a criação, você terá uma tela como a imagem a direita.

Para alterar o **nome do arquivo**, basta clicar no campo indicado e alterar.

O formato **.ipynb** indica que se trata de um formato de notebook. Ele poderá ser lido tanto no Google Colab quanto no Jupyter Notebook.



Uma outra opção para **abrir um arquivo** é clicar em **arquivo** e **abrir notebook**.

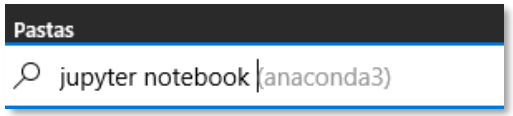


Módulo 3 – Criando seu Primeiro Programa

Agora vamos conferir pra ver se está tudo funcionando direitinho:

1. Na barra de pesquisa, digite Jupyter;

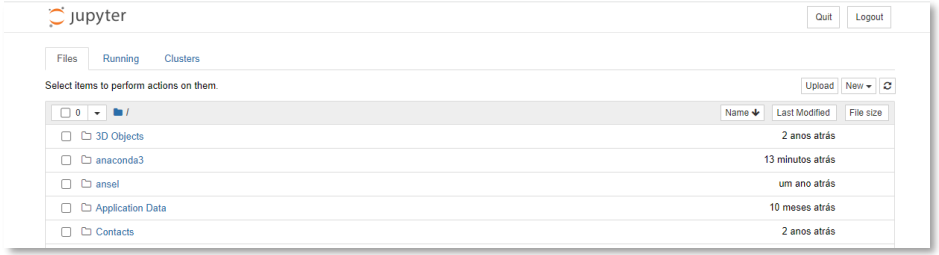
2. Clique no Jupyter Notebook;



3. Vai abrir uma tela preta e carregar algumas informações.

   Não feche a tela preta;

4. O Jupyter irá abrir no seu navegador principal.



Módulo 3 – Criando seu Primeiro Programa

Quando o Jupyter rodar no seu computador, você verá uma tela como a tela ao lado.

Essa tela é basicamente um espelho das pastas do **seu computador**.

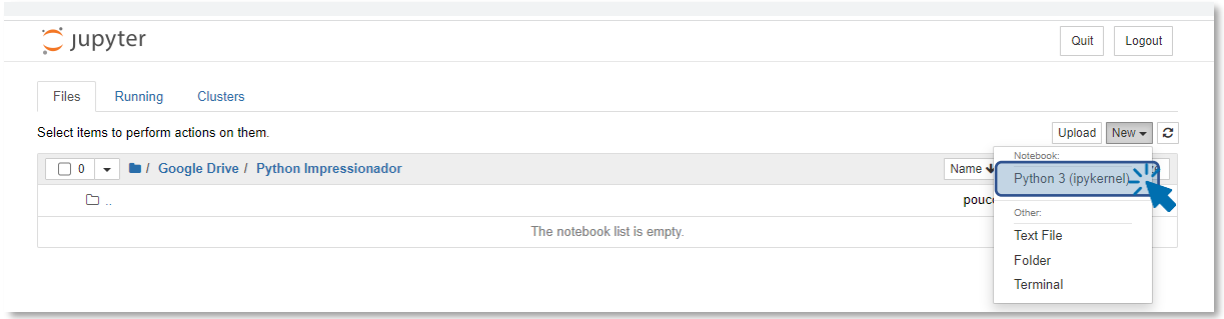
Antes de começar, vamos escolher uma pasta para salvar os arquivos.

No nosso caso, vamos salvar na Pasta Google Drive>Python Impressionador

Módulo 3 – Criando seu Primeiro Programa

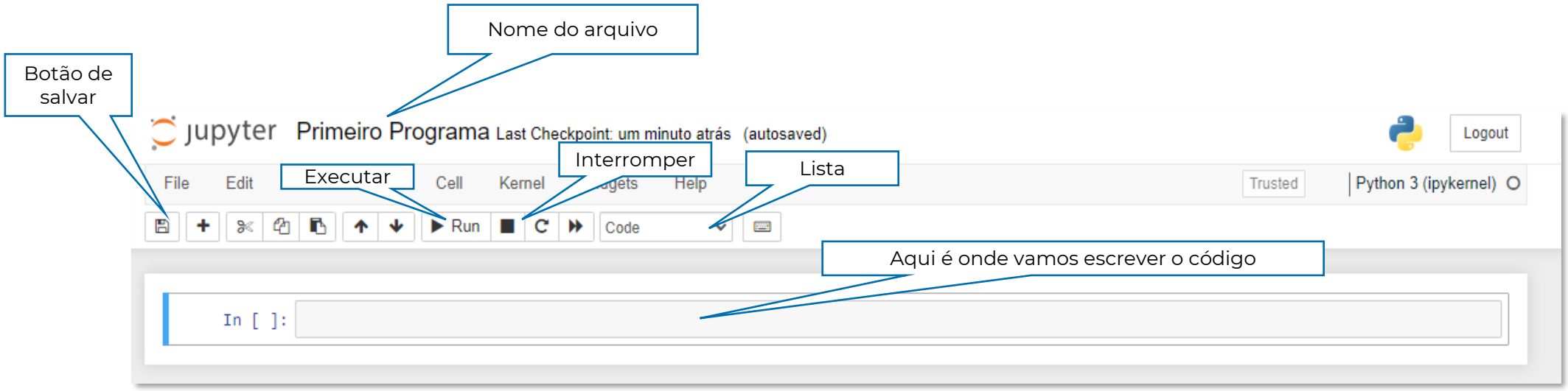
Para criar um novo arquivo do Jupyter do zero, basta clicar em NEW> Python 3.

Uma outra opção é carregar o arquivo em upload.



Módulo 3 – Criando seu Primeiro Programa

Vamos conhecer agora os **principais ícones** da barra de tarefas.



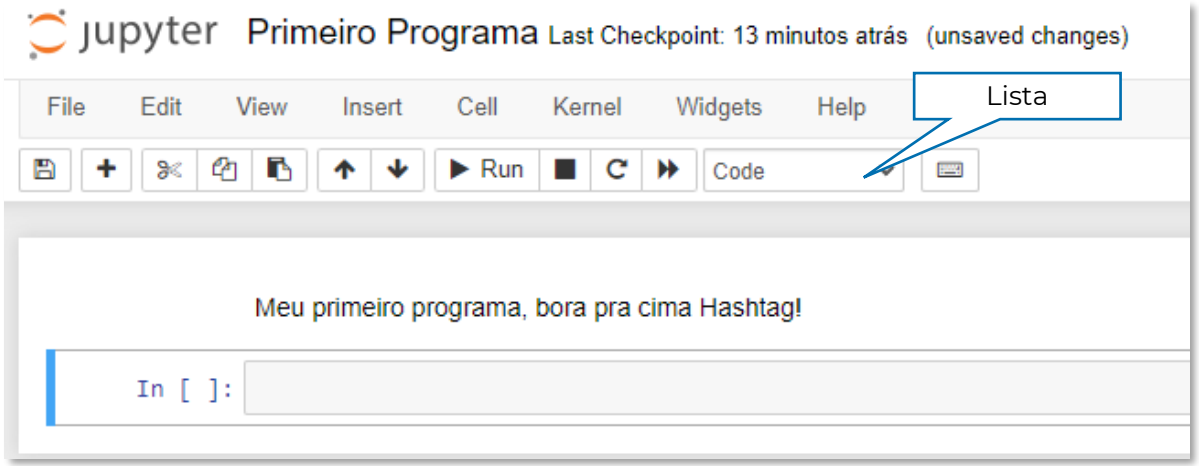
Módulo 3 – Criando seu Primeiro Programa

O Jupyter diferencia um código de um texto pelo item escolhido na lista.

Por exemplo, o **Markdown** é utilizado para escrever **texto**.

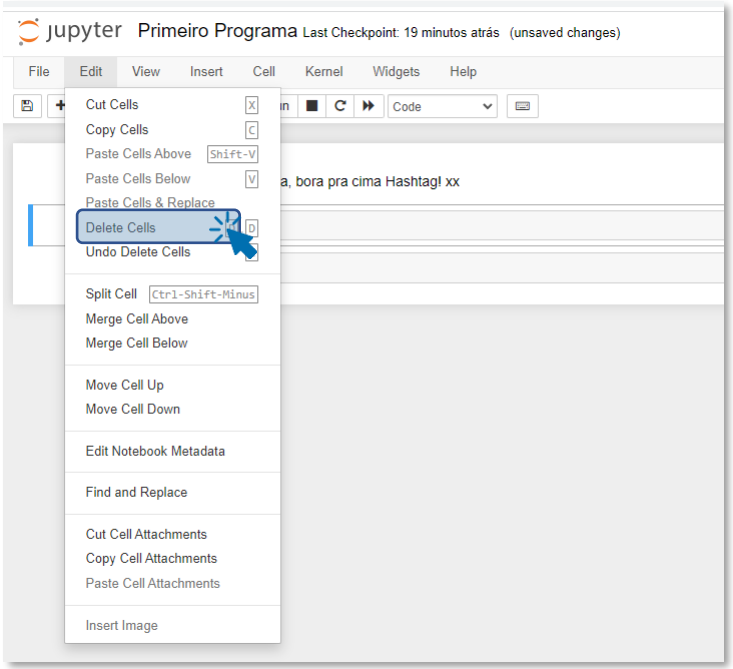
Para executar a célula selecionada, basta clicar no ícone **RUN** ou usar o atalho **CTRL + ENTER**.

Outro atalho é usar **SHIFT + ENTER** quando você quiser executar uma célula e inserir uma nova célula automaticamente.



Módulo 3 – Criando seu Primeiro Programa

Para deletar células, basta selecionar a célula, clicar em ***Edit*** e **deletar**.

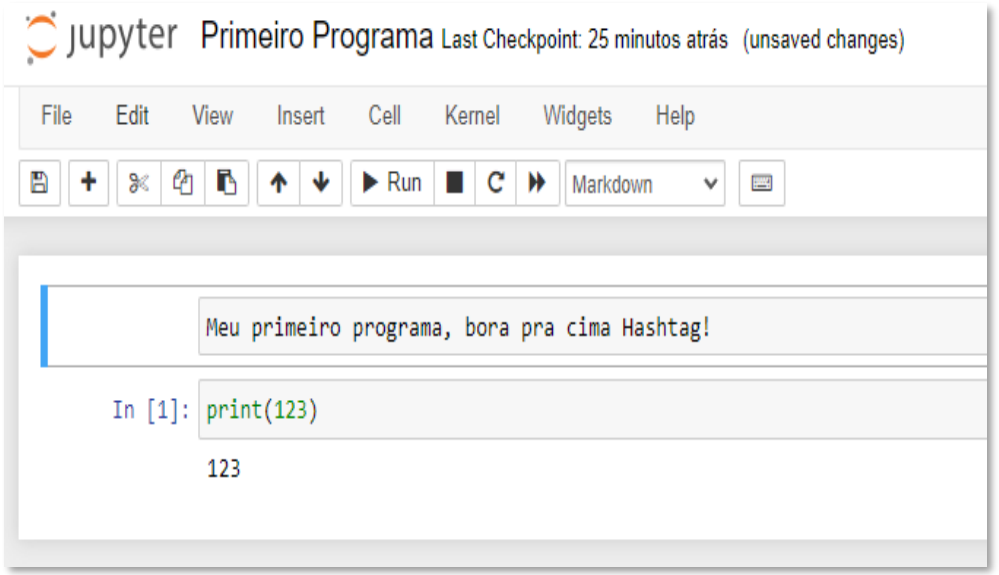


Módulo 3 – Criando seu Primeiro Programa

Agora vamos fazer o nosso primeiro código com a função **print**, que vai nos retornar " o valor que está dentro do parênteses".

Estrutura: **print**(**número**)

Então, a gente escreve o texto e aperta **CTRL + ENTER** para executar.



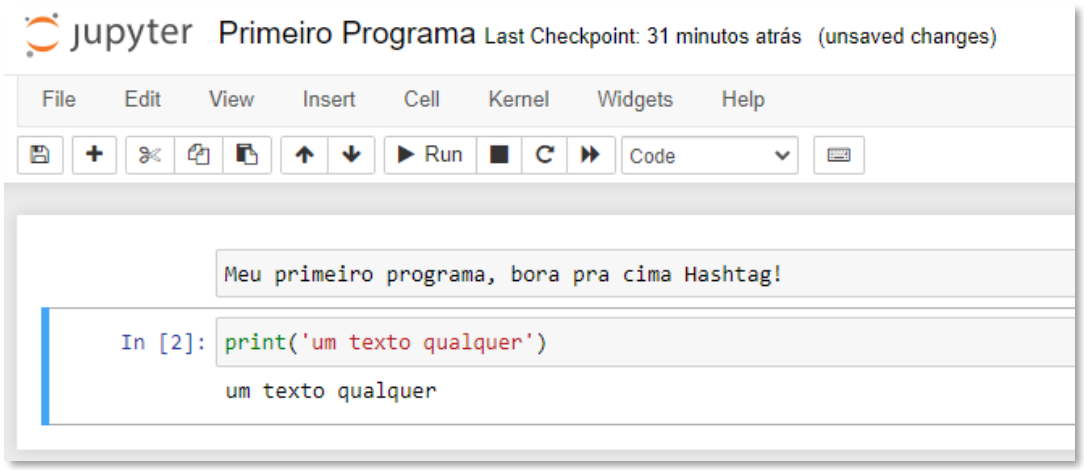
Módulo 3 – Criando seu Primeiro Programa

A função print pode ser usada tanto para número como para textos, mas para texto é muito importante lembrar de utilizar as **aspas**!

O Python vai entender tanto as aspas duplas (“”) quanto as simples (‘’). No curso, vamos padronizar escrever com as simples.

Estrutura: **print**(**“texto”**)

Então, a gente escreve o texto e aperta **CTRL + ENTER** para executar.



Módulo 3 – Variáveis

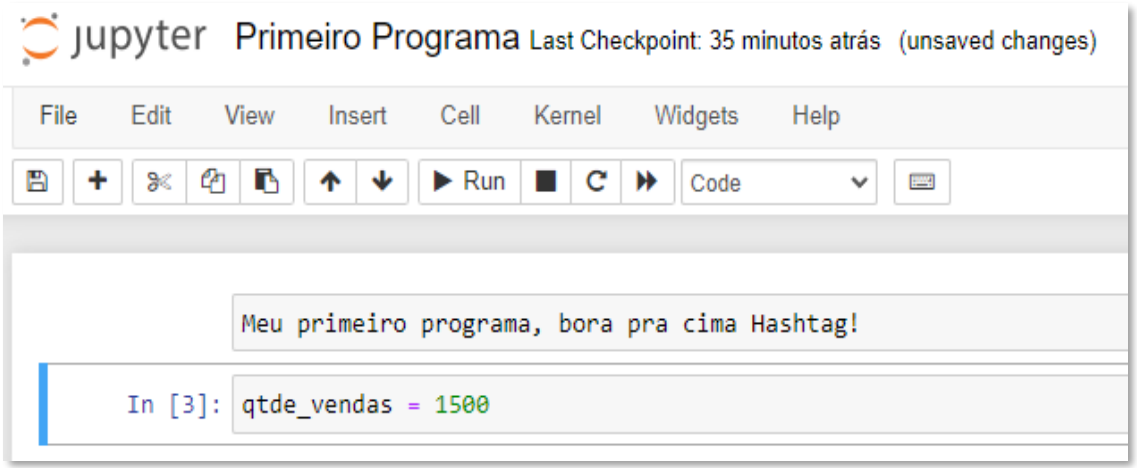
Variável é o nome atribuído para algum valor específico.

Estrutura: **nome_variável** = valor

Por exemplo: as vendas de uma empresa no mês de janeiro foram de 1500.

Então, nesse caso vamos criar uma variável para a quantidade de vendas.

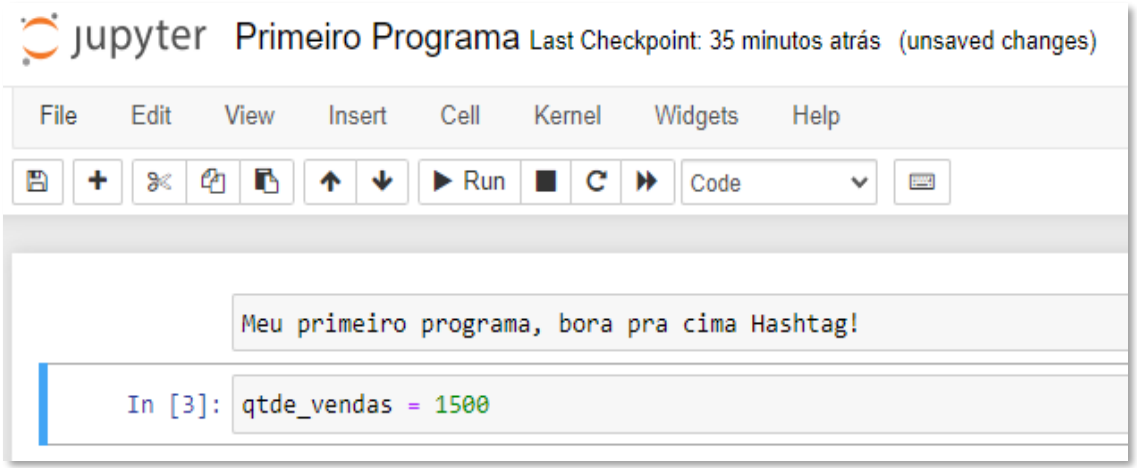
qtde_vendas = 1500



Módulo 3 – Variáveis

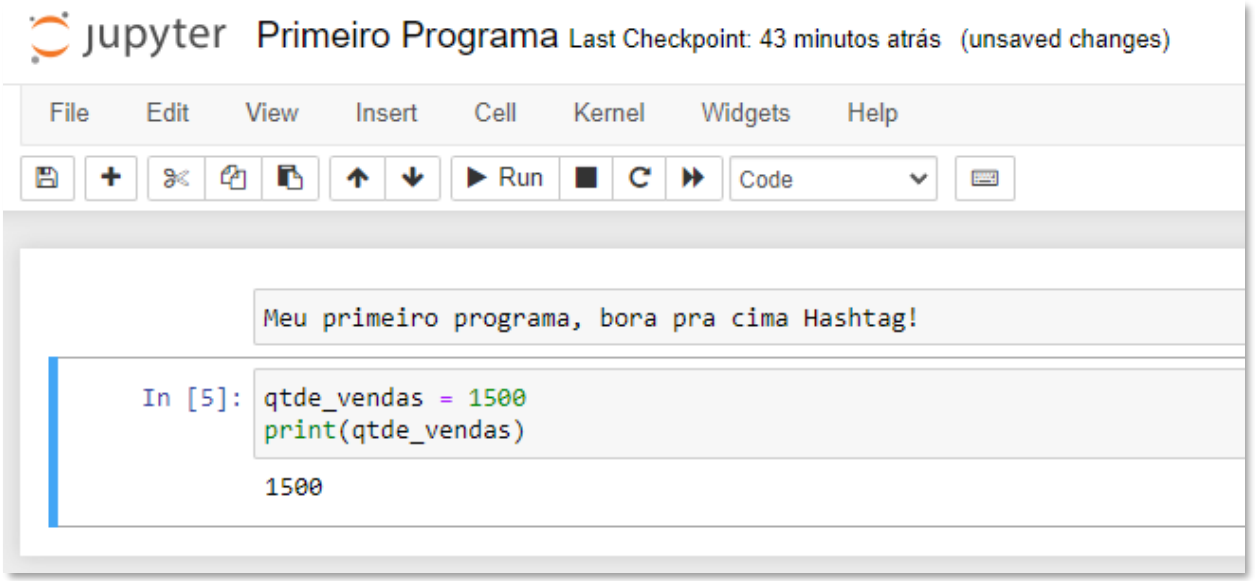
Um ponto importante é que **não pode ter espaço** no nome da variável. Você pode usar o "_" ou pode escrever tudo junto.

No curso vamos padronizar com o uso do "_" para variáveis com mais de uma palavra.



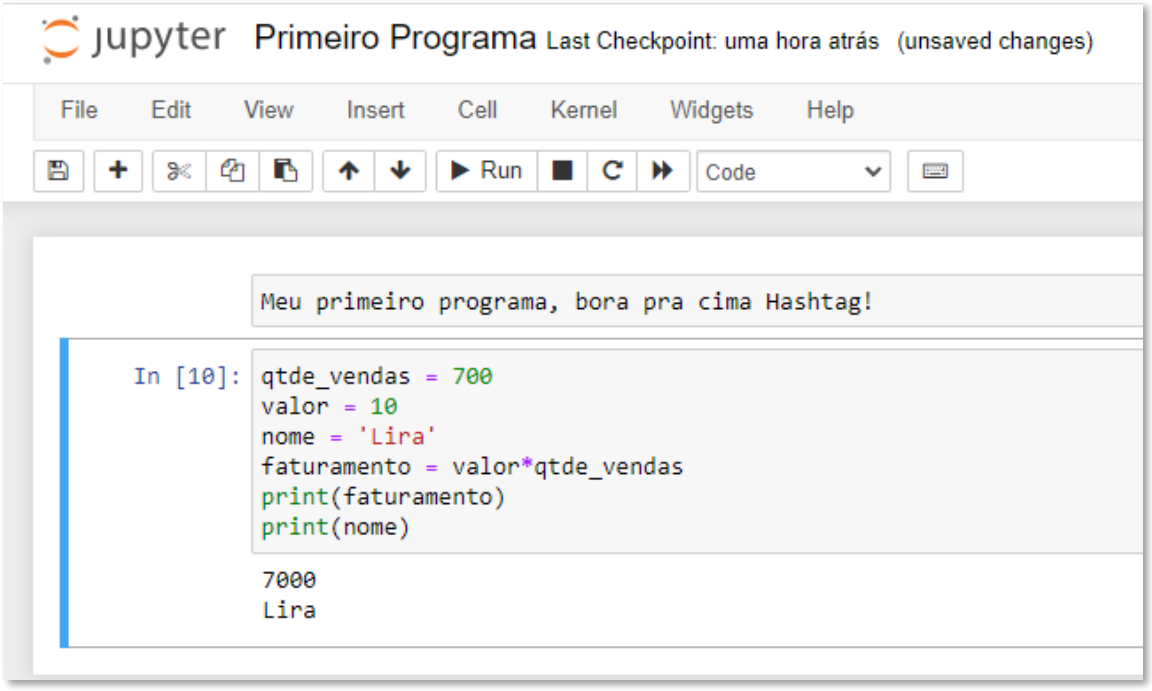
Quando estamos tratando de variáveis, o termo igual (=) não significa igual e sim, **recebe**.

Na programação, sempre o que está à esquerda do "=" **recebe** o valor do que está escrito à direita do " = "



Um vantagem do uso da variável é que se fosse necessário trocar o valor dela, não precisaria alterar o código todo.

No Python a gente não precisa dizer o tipo da variável, pois ele já consegue entender de forma automática.



Módulo 3 – Tipos de Variáveis

Até agora, atribuímos valores para as variáveis sem dizer qual é o tipo delas, isso porque o Python já entende o tipo da variável. Os tipos mais básicos de variáveis são:

- Int → Números inteiros.
- String → Texto. Lembrando que sempre vão estar entre aspas.
- Float → Números com casas decimais (ponto flutuante). Lembrando que no Python a casa decimal é representada pelo ponto.
- Bool ou boolean → Variáveis que só possuem 2 valores possíveis true (verdadeiro) ou false (falso).

Módulo 3 – Tipos de Variáveis

Para saber qual o tipo de uma variável, usamos a função **Type** como nos exemplos ao lado.

Estrutura: **type**(**variável**)

No primeiro exemplo, faturamento é uma variável do tipo inteiro. Já no segundo, como possui casa decimal, temos o tipo float.

No terceiro a variável está entre aspas, então é um texto. Por último, a variável recebe um valor de True, então é do tipo bool.

Módulo 3 – Tipos de Variáveis

Cuidados que você precisa ter com os nomes das variáveis:

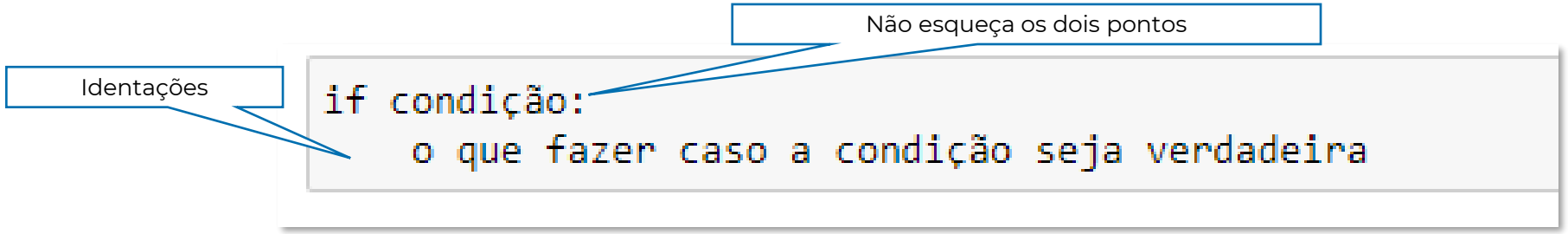
- **Não** escrever variáveis com caracteres especiais;
- Dica: use nomes em português;
- **Não** usar nomes reservados, por exemplo nome de uma função nativa do Python.

Lista com os principais nomes para você **evitar** quando for colocar nome nas variáveis:

https://drive.google.com/file/d/15qWxRRoC5qiGnmORacbQemTsKLcAp5fa/view?usp=sharing

Quase todo programa no Python utiliza a função **if**, uma vez que provavelmente você vai ter que fazer alguma verificação ou tratar alguma condição em algum momento.

E como funciona o **if**?



As identações indicam para o Python quais ações estão dentro daquele **if**.

Módulo 3 – Estrutura do if – Condições no Python

Agora vamos olhar um exemplo prático: você quer exibir uma mensagem caso a quantidade vendida no mês tenha ultrapassado a meta.

Então vamos atribuir valores às variáveis:

meta = 50000
qtde_vendida = 65300

Exemplo Real (informações 100% hipotéticas e inventadas):

Digamos que você trabalha na Amazon (que tem centenas de milhares, se não milhões de produtos) e está analisando o resultado de vendas dos produtos.

Você precisa criar um programa que vai analisar o resultado de vendas dos produtos da Amazon em um mês. Para simplificar vamos pensar em um único produto: um Iphone.

Meta de Vendas do Iphone = 50.000 unidades
Quantidade vendida no Mês = 65.300 unidades

O seu programa deve avisar (usaremos o print por enquanto) caso o produto tenha batido a meta do mês. Então devemos fazer:

- Caso o produto tenha batido a meta, devemos exibir a mensagem: "Batemos a meta de vendas de Iphone, vendemos {} unidades"
- Se ele não bateu a meta do mês, o seu programa não deve fazer nada

```
meta = 50000
qtde_vendida = 653000

if qtde_vendida>meta:
    print('Batemos a meta de vendas de Iphone, vendemos {} unidades'.format(qtde_vendida))
```

```
Batemos a meta de vendas de Iphone, vendemos 653000 unidades
```

Módulo 3 – Estrutura do if – Condições no Python

O segundo passo é construir a estrutura do **if**:

meta = 50000
qtde_vendida = 65300

If qtde_vendida>meta:
print('Batemos a meta de Iphone, vendemos {} unidades'.**format**(**qtde_vendida**)

```
meta = 50000
qtde_vendida = 653000

if qtde_vendida>meta:
    print('Batemos a meta de vendas de Iphone, vendemos {} unidades'.format(qtde_vendida))
```

```
Batemos a meta de vendas de Iphone, vendemos 653000 unidades
```

Módulo 3 – Estrutura do if – Condições no Python

Então basicamente estamos avaliando se a quantidade vendida foi maior que a meta. Se essa condição é verdadeira o Python vai imprimir a frase 'Batemos a meta de vendas de Iphone, vendemos {} unidades'.

Nota: a função **format** é utilizada para substituir uma variável nas chaves ({}).

```
meta = 50000
qtde_vendida = 653000

if qtde_vendida>meta:
    print('Batemos a meta de vendas de Iphone, vendemos {} unidades'.format(qtde_vendida))
```

```
Batemos a meta de vendas de Iphone, vendemos 653000 unidades
```

Atenção: Tudo o que estiver dentro do *tab* ele só vai executar se a condição for verdadeira.

Ambos com *tab*

```
In [2]: meta = 50000
        qtde_vendida = 653000

        if qtde_vendida>meta:
            print('Batemos a meta de vendas de Iphone, vendemos {} unidades'.format(qtde_vendida))
            print('Fim do programa')

        Batemos a meta de vendas de Iphone, vendemos 653000 unidades
        Fim do programa
```

Porém, se não estiver dentro do *tab* e a condição for falsa ele vai executar.

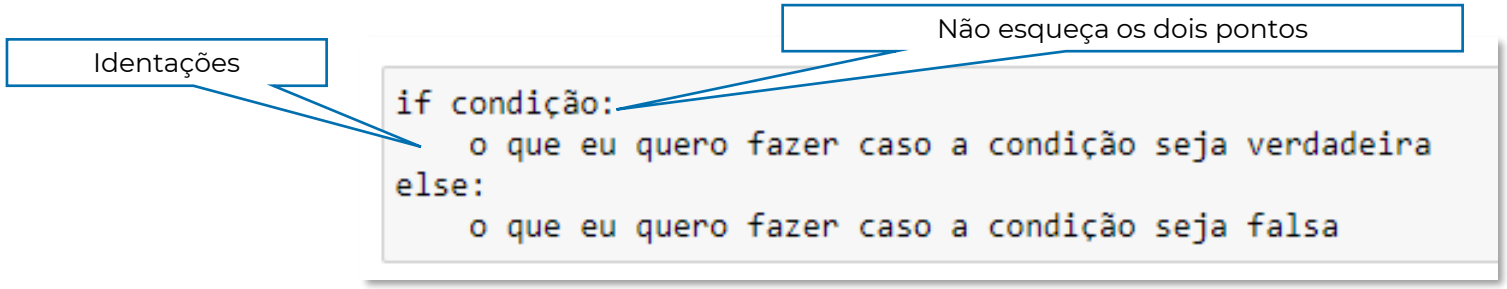
Quantidade vendida menor que a meta

Sem *tab*

```
In [4]: meta = 50000
        qtde_vendida = 3000

        if qtde_vendida>meta:
            print('Batemos a meta de vendas de Iphone, vendemos {} unidades'.format(qtde_vendida))
        print('Fim do programa')

        Fim do programa
```

Módulo 3 – Estrutura do if – Condições no Python

Quando usamos o **if**, nem sempre queremos apenas analisar o caso verdadeiro, em boa parte das vezes queremos fazer alguma coisa caso a condição seja verdadeira e fazer outra coisa caso a condição seja falsa.



Módulo 3 – Estrutura do if – Condições no Python

Utilizando o mesmo exemplo prático, agora nosso programa deve avisar nos 2 casos:

- Caso o produto tenha batido a meta, devemos exibir a mensagem: “Batemos a meta de vendas de Iphone, vendemos {} unidades”

- Se ele não bateu a meta do mês, devemos exibir a mensagem: “Infelizmente não batemos a meta, vendemos {} unidades. A meta era de {} unidades”

O primeiro passo é atribuir valores às variáveis:

meta = 50000
qtde_vendida = 65300

Módulo 3 – Estrutura do if – Condições no Python

O segundo passo é construir a estrutura do **if**:

```
meta = 50000
qtde_vendida = 65300

If qtde_vendida>meta:
        print('Batemos a meta de Iphone, vendemos {} unidades'.format(qtde_vendida)
Else:
        print('Infelizmente não batemos a meta, vendemos {} unidades. A meta era de {} unidades) format(meta)
```

Se essa condição for verdadeira

Caso contrário

Observação: a função **format** pode ser utilizada para substituir mais de uma variável nas chaves ({}).

```
meta = 50000
qtde_vendida = 65300

if qtde_vendida>meta:
    print('Batemos a meta de vendas de Iphone, vendemos {} unidades'.format(qtde_vendida))
else:
    print('Infelizmente não batemos a meta, vendemos {} unidades. A meta era de {} unidades'.format(qtde_vendida,meta))
```

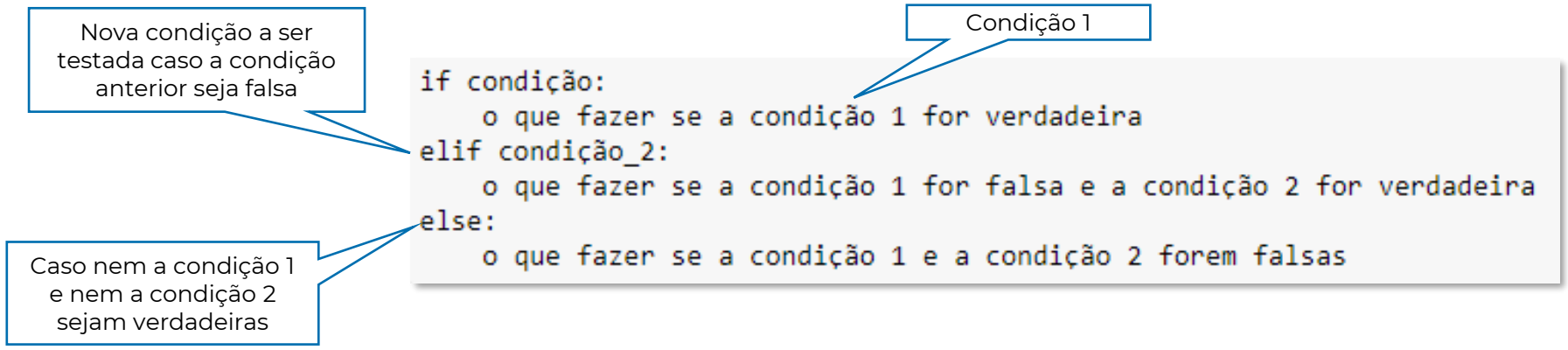
```
Batemos a meta de vendas de Iphone, vendemos 65300 unidades
```

Meta

Quantidade vendida

Módulo 3 – Elif

Agora vamos trabalhar com o acréscimo do if (um if composto), que é um caso mais complexo onde não temos apenas 1 condição, e sim 2 ou mais.




```
if condição:
    o que fazer se a condição 1 for verdadeira
elif condição_2:
    o que fazer se a condição 1 for falsa e a condição 2 for verdadeira
else:
    o que fazer se a condição 1 e a condição 2 forem falsas
```

Módulo 3 – Elif

Vamos para o exemplo prático ao lado.

Nesta situação, queremos criar um programa que seja capaz de analisar o bônus dos funcionários de uma empresa.

Para isso, temos que considerar alguns pontos importantes.

Exemplo:

Vamos criar um programa para analisar o bônus dos funcionários de uma empresa (pode parecer "simples", mas uma empresa como a Amazon tem 900.000 funcionários)

Para os cargos de vendedores, a regra do bônus é de acordo com a meta de vendas da pessoa:

Se ela vendeu abaixo da meta dela, ela não ganha bônus.

Se ela vendeu acima da meta dela, ela ganha como bônus 3% do valor que ela vendeu.

Se ela vendeu mais do que o dobro da meta dela, ela ganha como bônus 7% do valor que ela vendeu.

Vamos criar um programa para avaliar uma pessoa que tinha como meta de vendas 20.000 reais e calcular o bônus dela de acordo com o valor de vendas que ela tiver.

Módulo 3 – Elif

As condições são:

- Caso a meta não seja atingida, não o vendedor não ganha bônus.
- Caso as vendas superem a meta, ele ganha 3% do valor que ele vendeu.
- Se ele vender o dobro da meta, ele ganha 3% do valor que ele vendeu.

Exemplo:

Vamos criar um programa para analisar o bônus dos funcionários de uma empresa (pode parecer "simples", mas uma empresa como a Amazon tem 900.000 funcionários)

Para os cargos de vendedores, a regra do bônus é de acordo com a meta de vendas da pessoa:

Se ela vendeu abaixo da meta dela, ela não ganha bônus.

Se ela vendeu acima da meta dela, ela ganha como bônus 3% do valor que ela vendeu.

Se ela vendeu mais do que o dobro da meta dela, ela ganha como bônus 7% do valor que ela vendeu.

Vamos criar um programa para avaliar uma pessoa que tinha como meta de vendas 20.000 reais e calcular o bônus dela de acordo com o valor de vendas que ela tiver.

Módulo 3 – Elif

O primeiro passo é atribuir valores às variáveis:

meta = 20000
vendas = 1500

Exemplo:

Vamos criar um programa para analisar o bônus dos funcionários de uma empresa (pode parecer "simples", mas uma empresa como a Amazon tem 900.000 funcionários)

Para os cargos de vendedores, a regra do bônus é de acordo com a meta de vendas da pessoa:

Se ela vendeu abaixo da meta dela, ela não ganha bônus.

Se ela vendeu acima da meta dela, ela ganha como bônus 3% do valor que ela vendeu.

Se ela vendeu mais do que o dobro da meta dela, ela ganha como bônus 7% do valor que ela vendeu.

Vamos criar um programa para avaliar uma pessoa que tinha como meta de vendas 20.000 reais e calcular o bônus dela de acordo com o valor de vendas que ela tiver.

Módulo 3 – Elif

Agora vamos analisar o caso único, se a pessoa ganhou ou não bônus:

meta = 20000
vendas = 1500

```
if vendas < meta:
        print(‘Não ganhou bônus’)
else:
        bônus = 0.03 * vendas
        print(‘Ganhou {} de bônus’.format(bonus))
```

```
meta = 20000
vendas = 1500

if vendas < meta:
    print('Não ganhou bônus')
else:
    bônus = 0.03 * vendas
    print('Ganhou {} de bônus'.format(bonus))
```

```
Não ganhou bônus
```

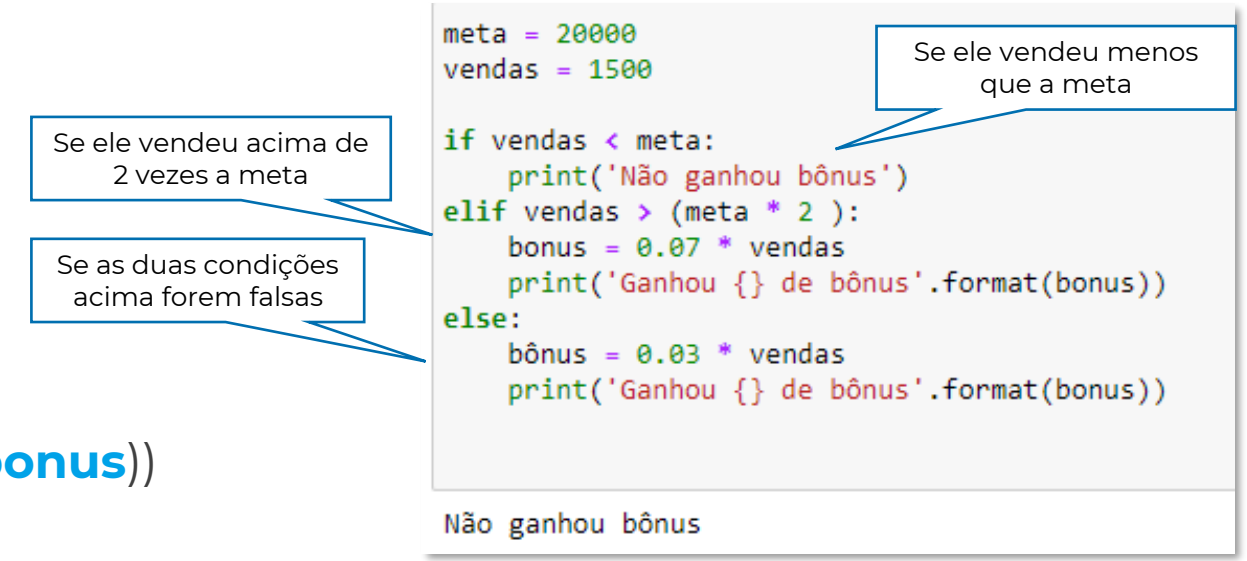
Mas agora a gente não pode falar que o vendedor vai ganhar 3% de bônus porque existem 2 condições.

Módulo 3 – Elif

Agora vamos analisar as 2 condições juntas:

meta = 20000
vendas = 1500

```
if vendas < meta:
        print('Não ganhou bônus')
elif vendas > (meta * 2):
        bônus = 0.07 * vendas
         print('Ganhou {} de bônus'.format(bonus))
else:
        bônus = 0.03 * vendas
        print('Ganhou {} de bônus'.format(bonus))
```



Módulo 3 – Comparadores

Principais comparadores que você pode usar dentro de um if ou elif:

== igual
!= diferente
> maior que (>= maior ou igual)
< menor que (<= menor ou igual)
in texto existe dentro de outro texto
not verifica o contrário da comparação

Observação: Lembre-se que o sinal de igual (=) não significa igual e sim, **recebe**.

Módulo 3 – Comparadores

Vamos ver agora alguns exemplos de comparadores:

No programa ao lado, se o faturamento da loja 1 for igual ao da loja 2, eu quero printar ‘Os faturamentos são iguais’, caso contrário quero printar ‘Os faturamentos são diferentes.’

Nesse exemplo, os faturamentos são diferentes (2500 e 2200).

```
faturamento_loja_1 = 2500
faturamento_loja_2 = 2200
email = "liragmail.com"

print('Programa 1')
if faturamento_loja_1 == faturamento_loja_2:
    print('Os faturamentos são iguais')
else:
    print('Os faturamentos são diferentes')
```

Símbolo de igual

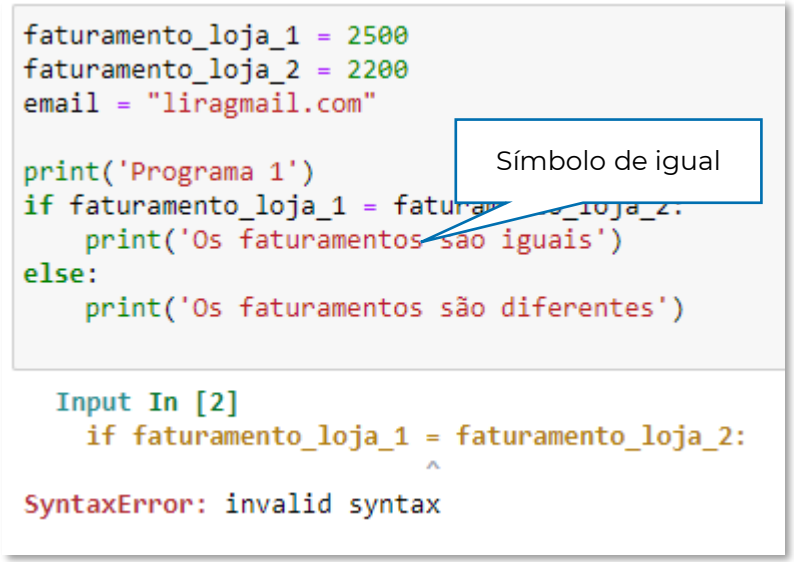
```
Programa 1
Os faturamentos são diferentes
```

Módulo 3 – Comparadores

Agora vamos ver o que aconteceria se ao invés do "**==**" a gente colocasse o símbolo de recebe "**=**".

Note que apareceu uma mensagem de **SyntaxError**, isso porque o Python não entende se você quer colocar um novo valor para o faturamento ou fazer uma comparação.

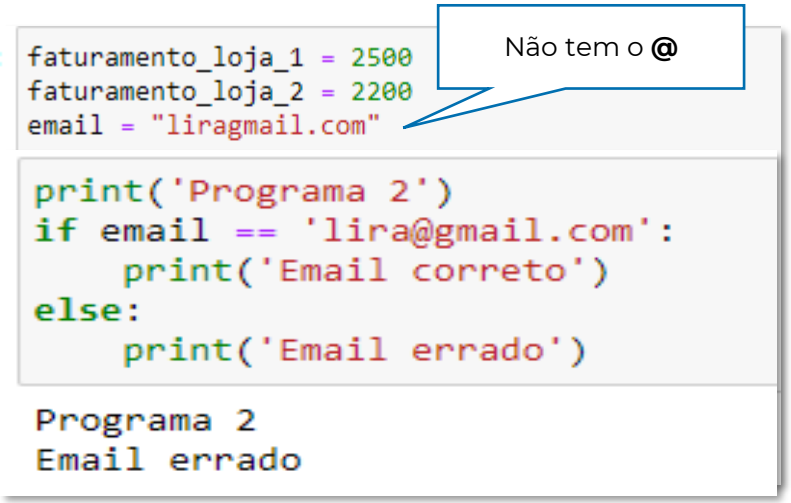
Então, sempre se atente ao sinal de **igual** como ==.



Módulo 3 – Comparadores

Outro exemplo, no programa ao lado eu quero verificar se o email é lira@gmail.com, se for igual eu quero printar “Email correto”, caso contrário ‘Email errado’.

Como o email não possui o **@**, então o email está errado.



Módulo 3 – Comparadores

Por último, no programa ao lado eu quero verificar se o email é válido.

Como podemos fazer isso?

Verificando se o email possui **@** utilizando o comparador ***in***.

O comparador ***in*** permite identificar se algo existe ao menos uma vez em um texto, ou em uma variável, etc.

Nesse caso, como não colocamos o **@,** por isso aparece que o email é inválido.

```
In [*]: print('Programa 3')
email_usuario = input('Insira seu e-mail:')
if not '@' in email_usuario:
    print('Email Inválido')
else:
    print('Email válido')
```

```
Insira seu e-mail: liragmail.com
```

```
In [8]: print('Programa 3')
email_usuario = input('Insira seu e-mail:')
if not '@' in email_usuario:
    print('Email Inválido')
else:
    print('Email válido')
```

```
Programa 3
Insira seu e-mail:liragmail.com
Email Inválido
```

Módulo 3 – Comparadores

Normalmente quando o email é válido não precisa aparecer uma mensagem informando que ele é válido.

Nesse caso podemos usar o comparador ***pass*** porque não queremos fazer nada nessa condição.

```
print('Programa 3')
email_usuario = input('Insira seu e-mail:')
if not '@' in email_usuario:
    print('Email Inválido')
else:
    pass
```

```
Insira seu e-mail: lira@gmail.com
```

```
print('Programa 3')
email_usuario = input('Insira seu e-mail:')
if not '@' in email_usuario:
    print('Email Inválido')
else:
    pass
```

```
Programa 3
Insira seu e-mail:lira@gmail.com
```

Módulo 3 – Comparadores

Nesse exemplo também usamos o comparador NOT, que inverte o sentido da condição.

Ou seja, no exemplo ao lado, temos como condição: se o **@ não** está contido na variável email_usuário.

Se essa condição for verdadeira, o programa imprime “Email Inválido”. Caso contrário, ele não faz nada.

```
print('Programa 3')
email_usuario = input('Insira seu e-mail:')
if not '@' in email_usuario:
    print('Email Inválido')
else:
    pass
```

```
Insira seu e-mail: lira@gmail.com
```

```
print('Programa 3')
email_usuario = input('Insira seu e-mail:')
if not '@' in email_usuario:
    print('Email Inválido')
else:
    pass
```

```
Programa 3
Insira seu e-mail:lira@gmail.com
```

Módulo 3 – And e Or

Até agora utilizamos o **if** para analisar uma condição de cada vez, mas em alguns casos vamos precisar usar conectores **and** e **or** para acrescentar mais condições a um mesmo teste.

Vamos utilizar o **and** para analisar se duas condições são verdadeiras **ao mesmo tempo** e **or** se **pelo menos uma** das duas condições forem verdadeiras.

```
if condicao_1 and condicao_2:
    vai ser executado se as 2 condições forem verdadeiras ao mesmo tempo

outro caso:

if condicao_1 or condicao_2:
    vai ser executado se pelo menos uma das condições forem verdadeiras
```

Módulo 3 – And e Or

Vamos ver um exemplo prático sobre cálculo de meta de vendas dos funcionários:

Aqui o cálculo do bônus depende de 2 condições para ocorrer:

- Se o funcionário vendeu mais do que a meta de vendas **e** a loja bateu a meta de vendas da loja, o funcionário ganha 3% do que ele vendeu em forma de bônus.

- Caso o funcionário tenha batido a meta de vendas individual dele, **mas** a loja não tenha batido a meta de vendas da loja como um todo, o funcionário não ganha bônus.

Módulo 3 – And e Or

No exemplo ao lado, como usamos o **and**, necessariamente as duas condições precisam ser atendidas para o funcionário ganhar bônus.

Nesse caso, temos que **nenhuma das condições é verdadeira**, logo o funcionário **não vai ganhar bônus**.

```
meta_funcionarios = 10000
meta_loja = 250000
vendas_funcionario = 5000
vendas_loja = 200000
if vendas_funcionario > meta_funcionarios and vendas_loja > meta_loja:
    bonus = 0.03 * vendas_funcionario
    print('Bônus do funcionário foi de {}'.format(bonus))
else:
    print('Funcionário não ganhou bônus')
```

Condição falsa

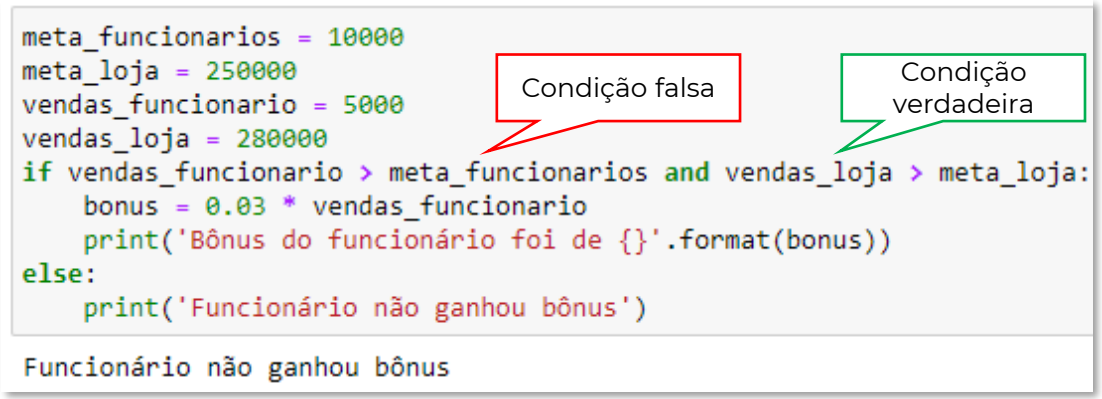
Condição falsa

```
Funcionário não ganhou bônus
```

Módulo 3 – And e Or

Vamos considerar agora que o valor de vendas da loja, ao invés de 200000 (exemplo anterior), passe a ser 280000.

Nesse caso, temos que apenas uma das condições é verdadeira, logo o funcionário **não vai ganhar bônus**.



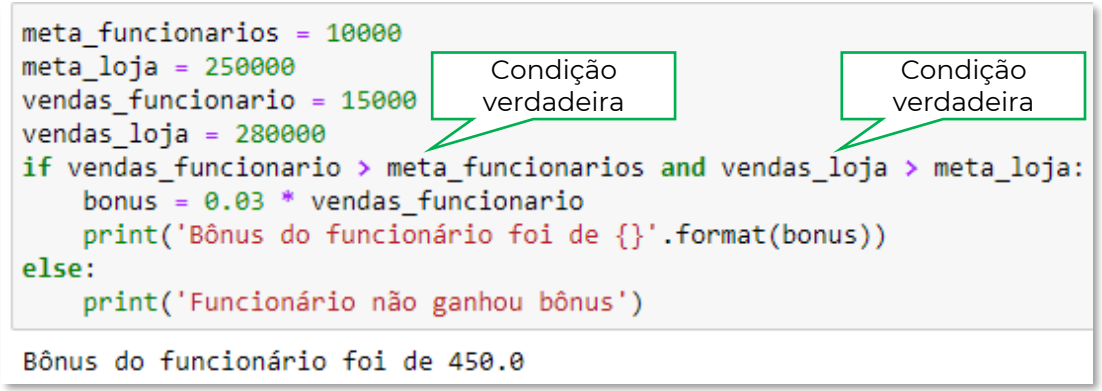

```
meta_funcionarios = 10000
meta_loja = 250000
vendas_funcionario = 5000
vendas_loja = 280000
if vendas_funcionario > meta_funcionarios and vendas_loja > meta_loja:
    bonus = 0.03 * vendas_funcionario
    print('Bônus do funcionário foi de {}'.format(bonus))
else:
    print('Funcionário não ganhou bônus')
```

```
Funcionário não ganhou bônus
```

Módulo 3 – And e Or

Vamos considerar agora que o valor de vendas de um funcionário, ao invés de 5000 (exemplo anterior), passe a ser 15000.

Nesse caso, temos que ambas as condições são verdadeiras, logo o funcionário **vai ganhar bônus**.




```
meta_funcionarios = 10000
meta_loja = 250000
vendas_funcionario = 15000
vendas_loja = 280000
if vendas_funcionario > meta_funcionarios and vendas_loja > meta_loja:
    bonus = 0.03 * vendas_funcionario
    print('Bônus do funcionário foi de {}'.format(bonus))
else:
    print('Funcionário não ganhou bônus')
```

```
Bônus do funcionário foi de 450.0
```

Módulo 3 – And e Or

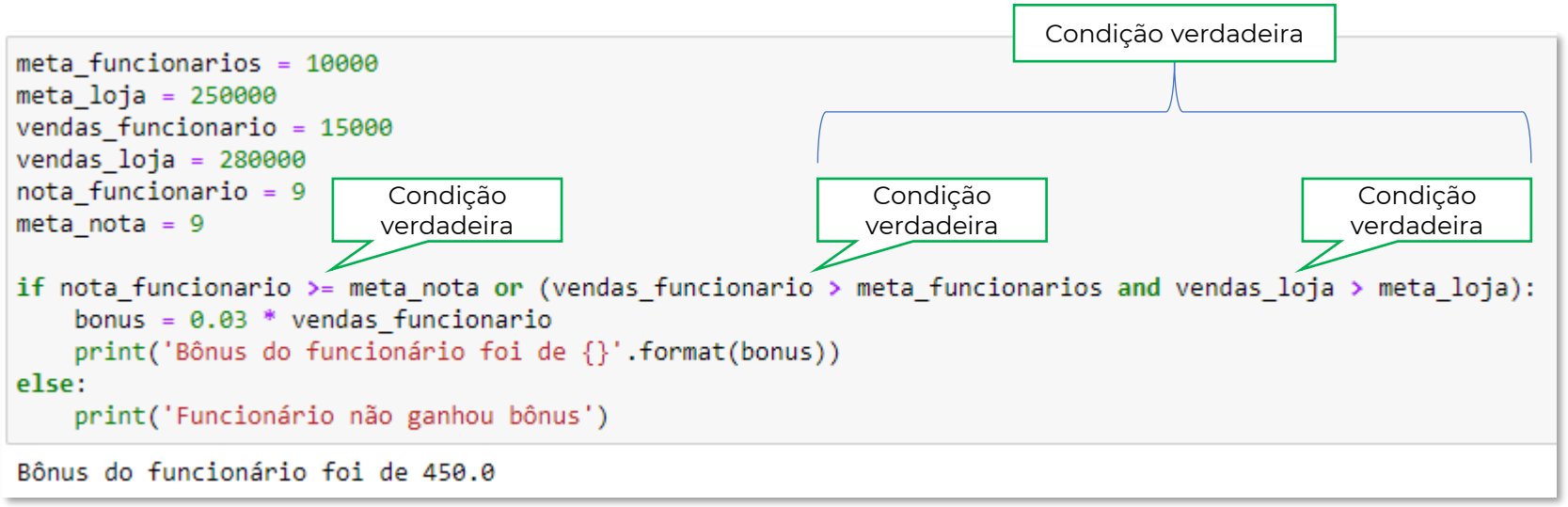
Agora vamos para um outro exemplo e analisar mais a fundo.

Nessa empresa, existe um outro caso também que garante que o funcionário ganhe um bônus, **independente das vendas que ele fez naquele mês**.

Se a nota do funcionário for 9 ou 10, ele também ganha o bônus de 3% do valor de vendas (os bônus não são cumulativos).

No exemplo abaixo, usamos o **and** e **or** ao mesmo tempo. Então, para o funcionário ganhar o bônus uma das condições do **or** precisa ser verdadeira.

Nesse caso, temos que **todas as condições são verdadeiras**, logo o funcionário **vai ganhar bônus**.

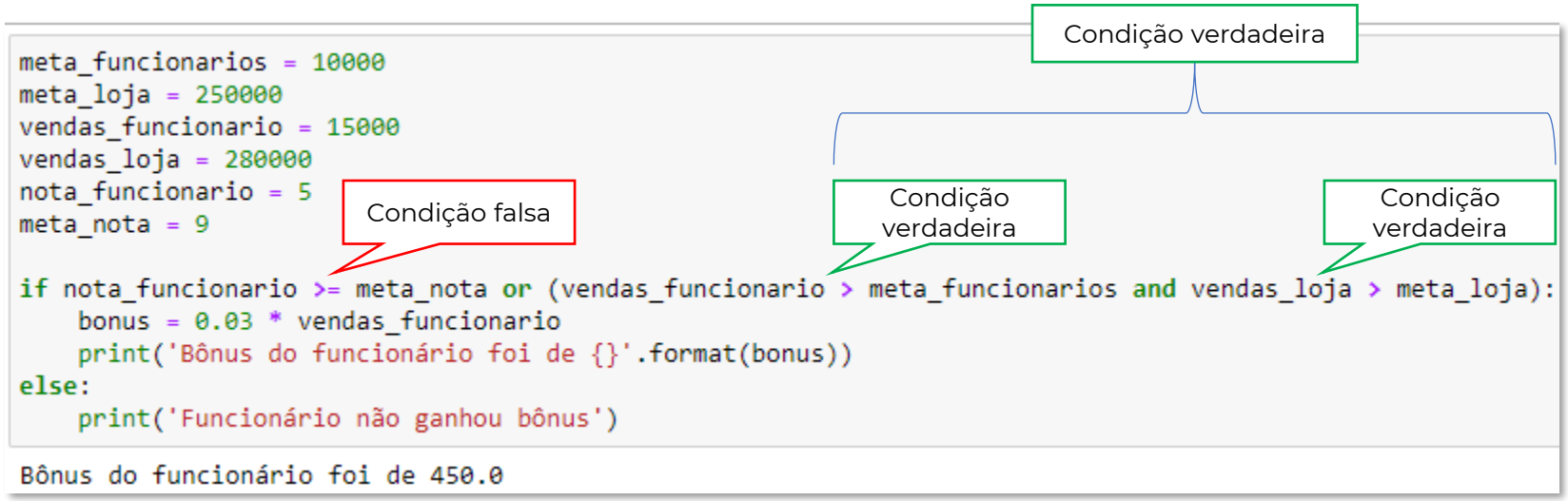



```
meta_funcionarios = 10000
meta_loja = 250000
vendas_funcionario = 15000
vendas_loja = 280000
nota_funcionario = 9
meta_nota = 9

if nota_funcionario >= meta_nota or (vendas_funcionario > meta_funcionarios and vendas_loja > meta_loja):
    bonus = 0.03 * vendas_funcionario
    print('Bônus do funcionário foi de {}'.format(bonus))
else:
    print('Funcionário não ganhou bônus')
```

```
Bônus do funcionário foi de 450.0
```

Agora vamos considerar que o valor da nota, ao invés de 9 (exemplo anterior), passe a ser 5.

Nesse caso, temos que **apenas a condição da nota é falsa**, porém como no **or apenas uma** das condições precisa ser verdadeira, então o funcionário **vai ganhar bônus**.

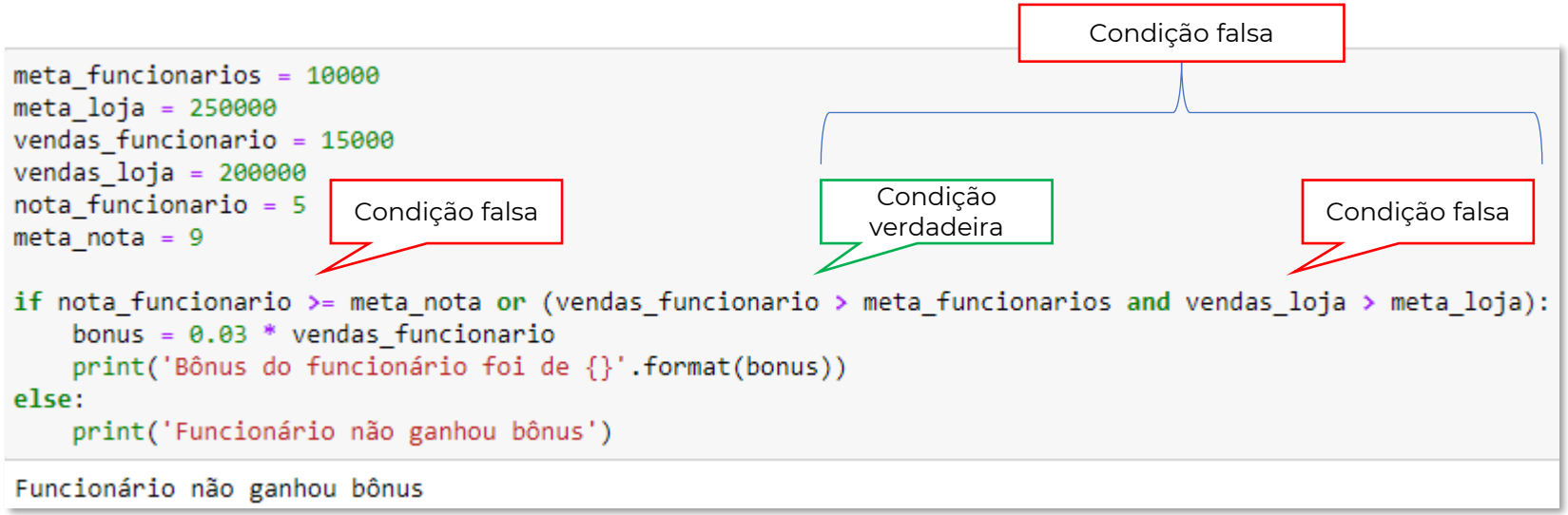



```
meta_funcionarios = 10000
meta_loja = 250000
vendas_funcionario = 15000
vendas_loja = 280000
nota_funcionario = 5
meta_nota = 9

if nota_funcionario >= meta_nota or (vendas_funcionario > meta_funcionarios and vendas_loja > meta_loja):
    bonus = 0.03 * vendas_funcionario
    print('Bônus do funcionário foi de {}'.format(bonus))
else:
    print('Funcionário não ganhou bônus')
```

```
Bônus do funcionário foi de 450.0
```

Agora vamos considerar que o valor das vendas, ao invés de 280000 (exemplo anterior), passe a ser 200000.

No caso do **and** ambas as condições precisam ser verdadeiras, então o resultado do **and** é uma condição **falsa.** Já que a outra condição do **or** também é **falsa,** o funcionário **não vai ganhar bônus.**



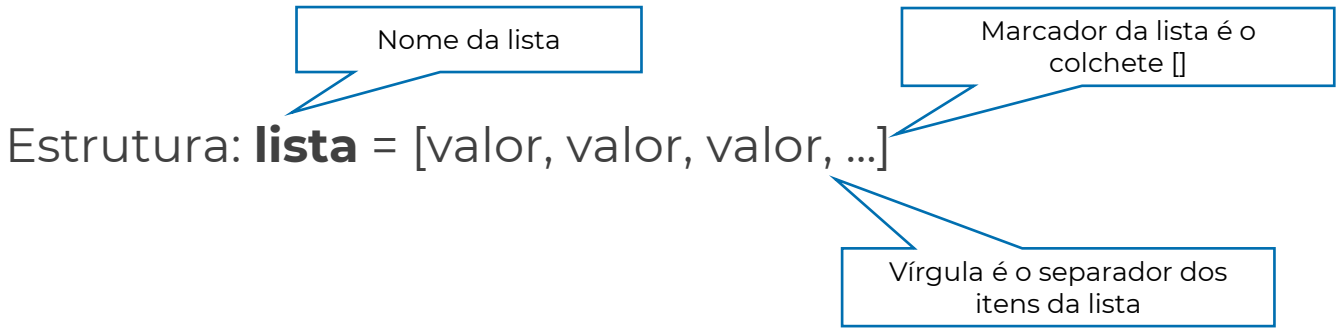

```
meta_funcionarios = 10000
meta_loja = 250000
vendas_funcionario = 15000
vendas_loja = 200000
nota_funcionario = 5
meta_nota = 9

if nota_funcionario >= meta_nota or (vendas_funcionario > meta_funcionarios and vendas_loja > meta_loja):
    bonus = 0.03 * vendas_funcionario
    print('Bônus do funcionário foi de {}'.format(bonus))
else:
    print('Funcionário não ganhou bônus')
```

```
Funcionário não ganhou bônus
```

Módulo 3 – Listas em Python

Agora vamos tratar das Listas, que são objetos muito importantes e serão muito utilizadas no curso. Quando importamos uma base de dados para o Python, normalmente ela é lida como uma "lista" ou como alguma "variação de lista“.

Listas em Python foram feitas para serem homogêneas (apenas número ou texto), apesar de aceitarem valores heterogêneos.



As listas podem ser vazias. Em breve vamos explicar como adicionar itens a elas.

```
lista = []
```

```
nome = 'celular'
produtos = ['tv',nome,'mouse','teclado', 'tablet']
print(produtos)
```

```
['tv', 'celular', 'mouse', 'teclado', 'tablet']
```

Podem ser representadas com todos os itens na mesma linha. Além disso, os valores das listas podem ser textos ou variáveis.

As listas também podem ser representadas separando uma linha para cada item.

Listas de Produtos de uma Loja:

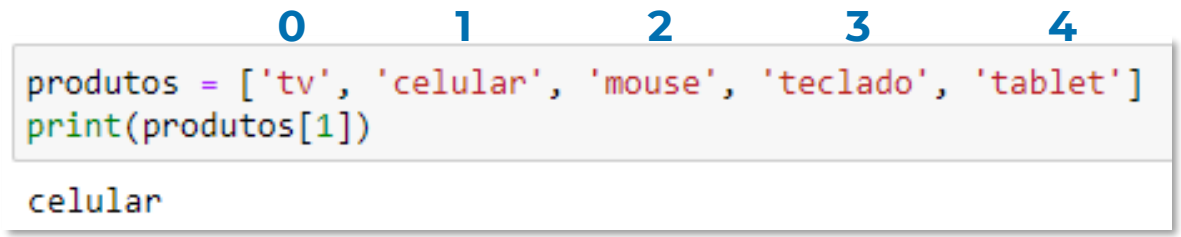
```
produtos = ['tv',
            'celular',
            'mouse',
            'teclado',
            'tablet']
```

Módulo 3 – Índices em Lista, Consultando e Modificando Valores

Agora vamos aprender os índices dentro da lista e como acessar ou modificar o valor de um item dentro da lista.

Se você quiser acessar um valor de um item da lista:
lista = [valor, valor, valor, valor, ...]
lista[i] → é o valor de índice i da lista que você quer como resposta.

Observação: Lembre que no Python o índice começa em 0, então o primeiro item de uma lista é o item lista[0].



Módulo 3 – Índices em Lista, Consultando e Modificando Valores

No exemplo ao lado temos duas listas que são independentes, mas possuem dados complementares.

Então podemos usar a função **format** para acessarmos os dados de produto e venda de forma simples utilizando o índice da lista.

```
produtos = ['tv', 'celular', 'mouse', 'teclado', 'tablet']
              0 ,      1   ,    2   ,     3    ,     4
vendas = [  1000,    1500  ,   350  ,    270   ,    900  ]
```

```
print('Vendas do produto {} foram de {}'.format(produtos[1], vendas[1]))
```

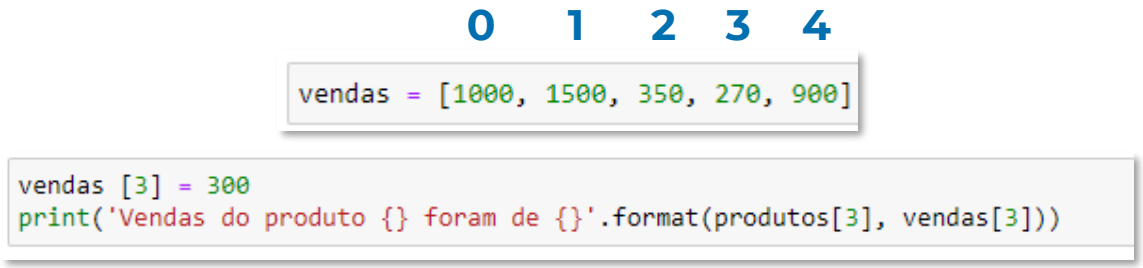
```
Vendas do produto celular foram de 1500
```

Módulo 3 – Índices em Lista, Consultando e Modificando Valores

Agora vamos aprender como substituir um valor de uma lista.

Por exemplo, ao invés de 270 deveria ser 300.

Então, antes do print, vamos colocar o índice que queremos substituir recebendo o valor correto.



Módulo 3 – Índices em Lista, Consultando e Modificando Valores

Uma *string* no Python é como se fosse uma lista de caracteres, então o texto de índice 1 do exemplo ao lado é a letra i.

Uma diferença significativa entre a *string* e a lista é que a *string* no Python é **imutável**.

Na lista é possível substituir o valor, na *string* não. No exemplo ao lado aparece o erro de que a string não suporta o *item assignment*.

```
texto = 'lira@gmail.com'
print(texto[1])
```

```
i
```

```
texto = 'lira@gmail.com'
texto [1] = 'a'
print(texto)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [7], in <cell line: 2>()
      1 texto = 'lira@gmail.com'
----> 2 texto [1] = 'a'
      3 print(texto)

TypeError: 'str' object does not support item assignment
```

Para substituir o valor da *string* podemos utilizar a função *replace*, mas para isso temos que criar uma nova variável.

texto = 'lira@gmail.com'
texto = **texto.replace**('i', 'a')

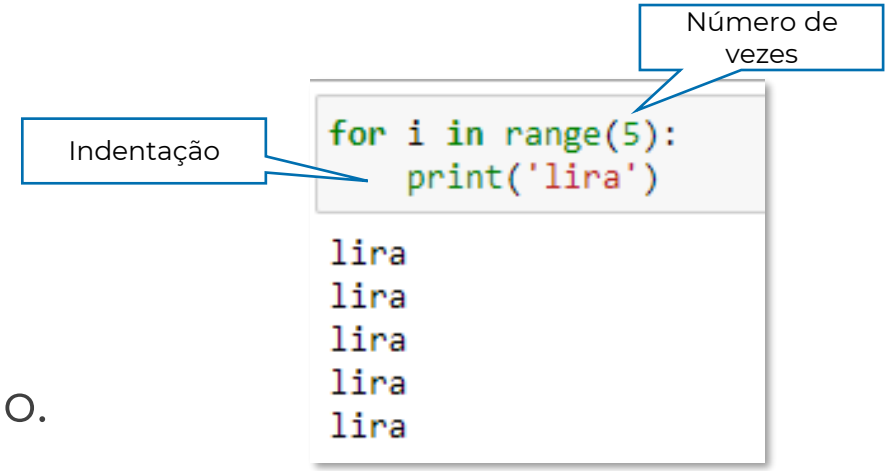
Módulo 3 – Estrutura de Repetição For

Agora vamos utilizar uma estrutura de repetição para conseguir executar códigos de forma repetitiva, ou seja, ao invés de repetir nosso código podemos pedir pra ele olhar uma lista inteira e percorrer toda aquela lista executando alguma ação. Uma das estruturas mais úteis e usadas é a **for**.

Forma mais simples de utilizar o FOR:

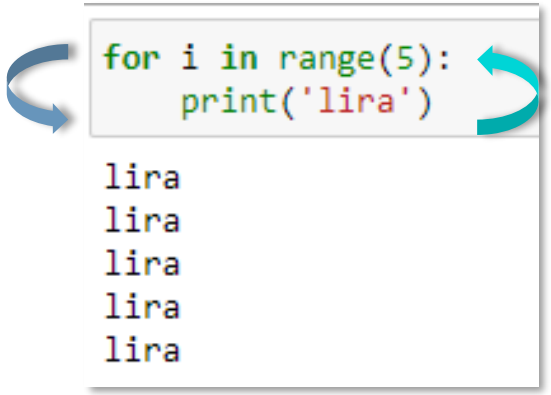
for variável in range(n):
    repetir código n vezes

Vamos fazer um exemplo bem simples ao lado.



Módulo 3 – Estrutura de Repetição For

No exemplo ao lado, o código está executando o print e voltando para a primeira linha, até terminar a contagem.

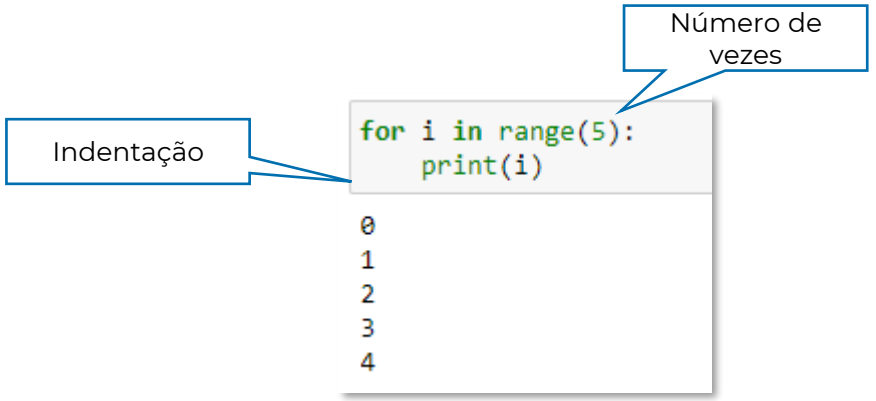


Módulo 3 – Estrutura de Repetição For

O i é um índice que funciona como um contador.

Então no código ao lado, pedimos para ele imprimir o valor de **i** um total de **5 vezes**.

Lembrando que o Python começa a contagem com o número 0.



Módulo 3 – Estrutura de Repetição For

Imagine que você está construindo uma automação para enviar todo dia por e-mail um resumo da produção de uma fábrica. Ao invés de repetir o print 5 vezes, nós vamos colocar isso dentro de um for.

```
produtos = ['coca', 'pepsi', 'guarana', 'sprite', 'fanta']
producao = [15000, 12000, 13000, 5000, 250]

for i in range(5):

    print('{} unidades produzidades de {}'.format(producao[i],produtos[i]))
```

Retornar o valor da tabela produção

Retornar a marca

```
1ª 15000 unidades produzidades de coca
2ª 12000 unidades produzidades de pepsi
3ª 13000 unidades produzidades de guarana
4ª 5000 unidades produzidades de sprite
5ª 250 unidades produzidades de fanta
```

Agora imagine se a lista aumentar ou diminuir, nesse caso nossa estrutura for não comtemplará todos os valores da lista. Para resolver esse problema vamos usar a função **len** que calculará o tamanho da lista e permitirá que mesmo com alteração do número de itens da lista o nosso for sempre vai contemplar todos.

Retorna o tamanho da lista produtos

```
produtos = ['coca', 'pepsi', 'guarana', 'sprite', 'fanta']
producao = [15000, 12000, 13000, 5000, 250]

tamanho = len(produtos)

for i in range(tamanho):

    print('{} unidades produzidades de {}'.format(producao[i],produtos[i]))
```

```
15000 unidades produzidades de coca
12000 unidades produzidades de pepsi
13000 unidades produzidades de guarana
5000 unidades produzidades de sprite
250 unidades produzidades de fanta
```

Módulo 3 – For each - Percorrer cada item de uma lista

Agora a gente vai trabalhar com uma pequena variação do **for**.

No exemplo anterior utilizamos o **range** para informar o número de repetições executadas no for.

Porém, no Python é muito comum usarmos a própria lista para termos essa informação.

Como podemos ver no exemplo, criamos localmente uma variável **item** que irá percorrer todos os itens da lista produtos.

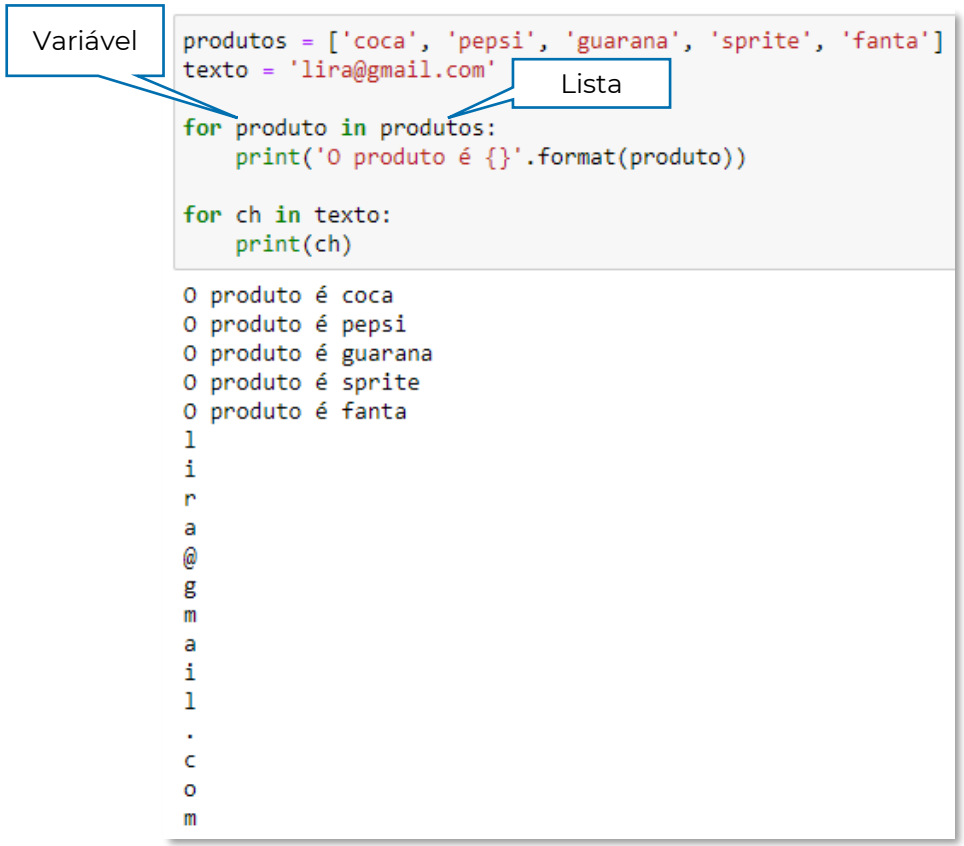
```
produtos = ['coca', 'pepsi', 'guarana', 'sprite', 'fanta']
texto = 'lira@gmail.com'

for item in produtos:
    print(item)
```

```
coca
pepsi
guarana
sprite
fanta
```

Módulo 3 – For each - Percorrer cada item de uma lista

É muito comum no Python utilizarmos a palavra referente à lista no singular (**produto**) e na lista a palavra no plural (**produtos**).

Essa estrutura pode ser lida como:
“Para cada **produto** na lista **produtos**, “printar” o texto”.



Módulo 3 – For e if

Vamos agora combinar o uso do for com o IF.

Note que ao lado apresentamos uma estrutura genérica. Perceba que o If possui uma identação em relação ao FOR e as linhas 3 e 5 possuem uma identação “dupla”, pois pertencem ao IF e ao FOR.

```
for item in lista:
    if condicao:
        faça alguma coisa
    else:
        outra coisa
```

Módulo 3 – For e if

Imagine que a gente esteja analisando a meta de vendas de vários funcionários de uma empresa.

A meta de vendas é de 1000 reais em 1 dia e queremos calcular qual o % de pessoas que bateram a meta.

O primeiro passo que vamos fazer é percorrer essa lista e printar todos os itens da lista.

```
vendas = [1200, 300, 800, 1500, 1900, 2750, 400, 20, 23, 70, 90, 80, 1100, 999, 900, 880, 870, 50, 1111, 120, 300, 450, 800]
meta = 1000

for venda in vendas:
    print(venda)
```

```
1200
300
800
1500
1900
2750
400
20
23
70
90
80
1100
999
900
880
870
50
1111
120
300
450
800
```

Agora vamos analisar se as vendas foram maiores que a meta. Porém, o ideal seria termos um “contador” que acumulasse o número de funcionários que bateram a meta.

Com esse contador será possível, por exemplo, dizer automaticamente a % de atingimento de meta.

```
vendas = [1200, 300, 800, 1500, 1900, 2750, 400, 20, 23, 70, 90, 80, 1100, 999, 900, 880, 870, 50, 1111, 120, 300, 450, 800]
meta = 1000

for venda in vendas:
    if venda >= meta:
        print(venda)
```

```
1200
1500
1900
2750
1100
1111
```

Abaixo indicamos uma variável **qtde_bateu_meta** com valor inicial igual a zero e cada vez que a meta fosse atingida seria somado 1 a esse valor.

```
vendas = [1200, 300, 800, 1500, 1900, 2750, 400, 20, 23, 70, 90, 80, 1100, 999, 900, 880, 870, 50, 1111, 120, 300, 450, 800]
meta = 1000

qtde_bateu_meta = 0

for venda in vendas:
    if venda >= meta:
        qtde_bateu_meta +=1
print(qtde_bateu_meta)
```

qtde_bateu_meta = qtde_bateu_meta + 1

```
6
```

Agora precisamos calcular a quantidade de pessoas totais, ou seja, o tamanho da minha lista. Depois vamos calcular o percentual dividindo a quantidade de pessoas que bateram a meta pela quantidade total de pessoas.

```
vendas = [1200, 300, 800, 1500, 1900, 2750, 400, 20, 23, 70, 90, 80, 1100, 999, 900, 880, 870, 50, 1111, 120, 300, 450, 800]
meta = 1000

qtde_bateu_meta = 0

for venda in vendas:
    if venda >= meta:
        qtde_bateu_meta +
print(qtde_bateu_meta)

qtde_funcioarios = len(vendas)
print('O percentual de pessoas que bateram a meta foi de {:.1%} '.format(qtde_bateu_meta / qtde_funcioarios))
```

```
6
O percentual de pessoas que bateram a meta foi de 26.1%
```

: indica que será o formato de uma forma específica

.1 significa o número de casas decimais do resultado

% indica que o resultado se trata de uma porcentagem

Agora a gente vai estudar uma outra estrutura de repetição: o **while**.

A grande diferença em relação ao for é que no **for** a gente executava a repetição um **número de vezes previamente definido**; já no **while**, a gente vai executar a repetição enquanto **uma determinada condição for verdadeira**.

```
while condicao:
    repete esse código
```

Módulo 3 – Estrutura While

Vamos ver na prática como isso funciona para criar um programa que registre as vendas de uma empresa.

Nele, a pessoa deve inserir o nome do produto e o produto deve ser adicionado na lista de venda. Enquanto o usuário não encerrar o programa, significa que ele está registrando novos produtos, então o programa deve permitir e entrada de quantos produtos o usuário quiser.

```
venda = input('Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vazia')
vendas = []

while venda != '':
    vendas.append(venda)
    venda = input('Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vazia')

print('Registro Finalizado. As vendas cadastradas foram: {}'.format(vendas))
```

Diferente de vazio

```
Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vaziaarroz
Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vaziafeijão
Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vazia
Registro Finalizado. As vendas cadastradas foram: ['arroz', 'feijão']
```

Dica de ouro: sempre que for usar o comando **while**, lembre-se de ter certeza que o programa vai terminar em algum momento.

Se o while não terminar em algum momento ele entra no que chamamos de loop infinito e aí duas opções podem ocorrer:

1) Seu computador vai travar
2) Seu código ficará rodando infinitamente

Módulo 3 – Loop infinito no While

Digamos que temos uma lista de vendedores e as quantidades vendidas e queremos identificar todos os vendedores que bateram a meta de 50 vendas.

Olhando a condição do **while** temos que nosso loop será pausado quando a venda for menor que a meta.

Quando olhamos o código dentro da identação não temos nada que aumente o valor de i, que diferente do **for** aumenta automaticamente.

```
vendas = [941, 852, 783, 714, 697, 686, 685, 670, 631, 453, 386, 371, 294, 269, 259, 218, 208, 163, 125, 102, 87, 47, 7]
vendedores = ['Maria', 'José', 'Antônio', 'João', 'Francisco', 'Ana', 'Luiz', 'Paulo', 'Carlos', 'Manoel', 'Pedro', 'Francisca',
meta = 50

i = 0

while vendas[i] > meta:
    print('{} bater a meta. Vendas: {}'.format(vendedores[i],vendas[i]))
```

Módulo 3 – Loop infinito no While

Isso fará com que **vendas** sempre seja 941 e consequentemente sempre maior que a meta que é 50.

Logos, estamos realizando um loop infinito.

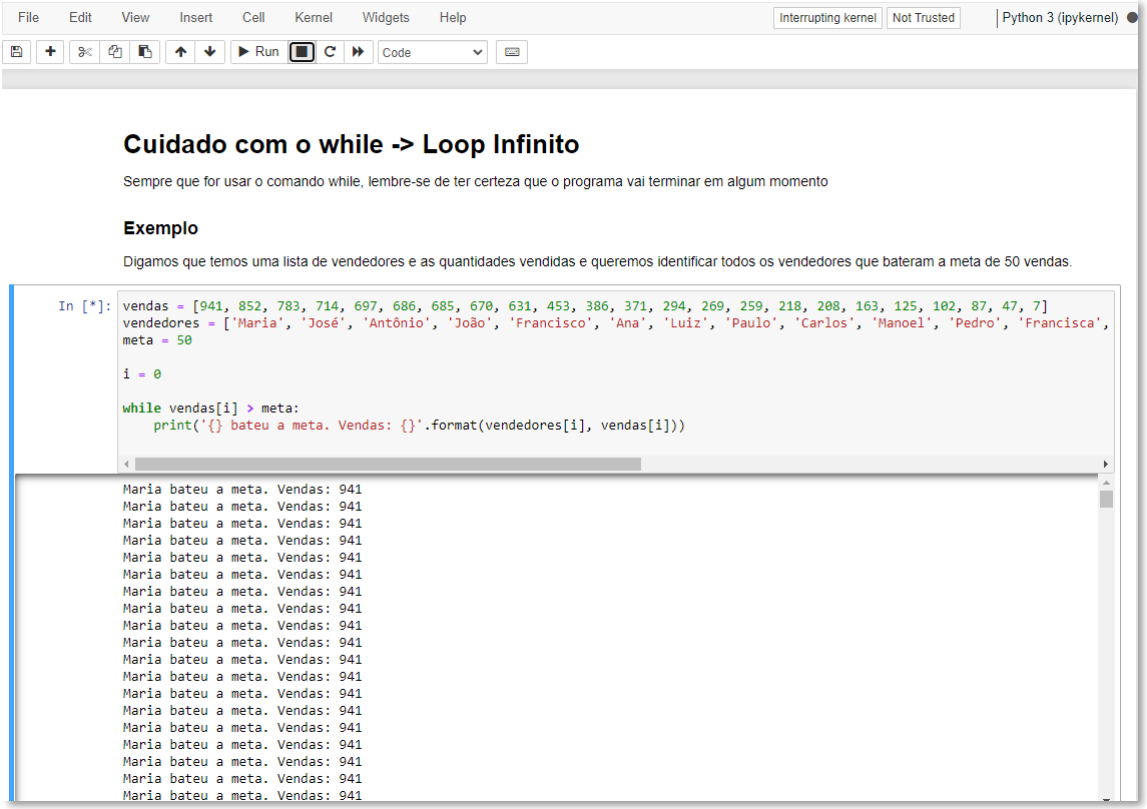
```
In [*]: vendas = [941, 852, 783, 714, 697, 686, 685, 670, 631, 453, 386, 371, 294, 269, 259, 218, 208, 163, 125, 102, 87, 47, 7]
vendedores = ['Maria', 'José', 'Antônio', 'João', 'Francisco', 'Ana', 'Luiz', 'Paulo', 'Carlos', 'Manoel', 'Pedro', 'Francisca',
meta = 50

i = 0

while vendas[i] > meta:
    print('{} bateu a meta. Vendas: {}'.format(vendedores[i], vendas[i]))
```

```
Maria bateu a meta. Vendas: 941
Maria bateu a meta. Vendas: 941
Maria bateu a meta. Vendas: 941
Maria bateu a meta. Vendas: 941
Maria bateu a meta. Vendas: 941
Maria bateu a meta. Vendas: 941
Maria bateu a meta. Vendas: 941
Maria bateu a meta. Vendas: 941
Maria bateu a meta. Vendas: 941
Maria bateu a meta. Vendas: 941
Maria bateu a meta. Vendas: 941
Maria bateu a meta. Vendas: 941
Maria bateu a meta. Vendas: 941
Maria bateu a meta. Vendas: 941
Maria bateu a meta. Vendas: 941
Maria bateu a meta. Vendas: 941
Maria bateu a meta. Vendas: 941
Maria bateu a meta. Vendas: 941
Maria bateu a meta. Vendas: 941
```

Módulo 3 – Loop infinito no While

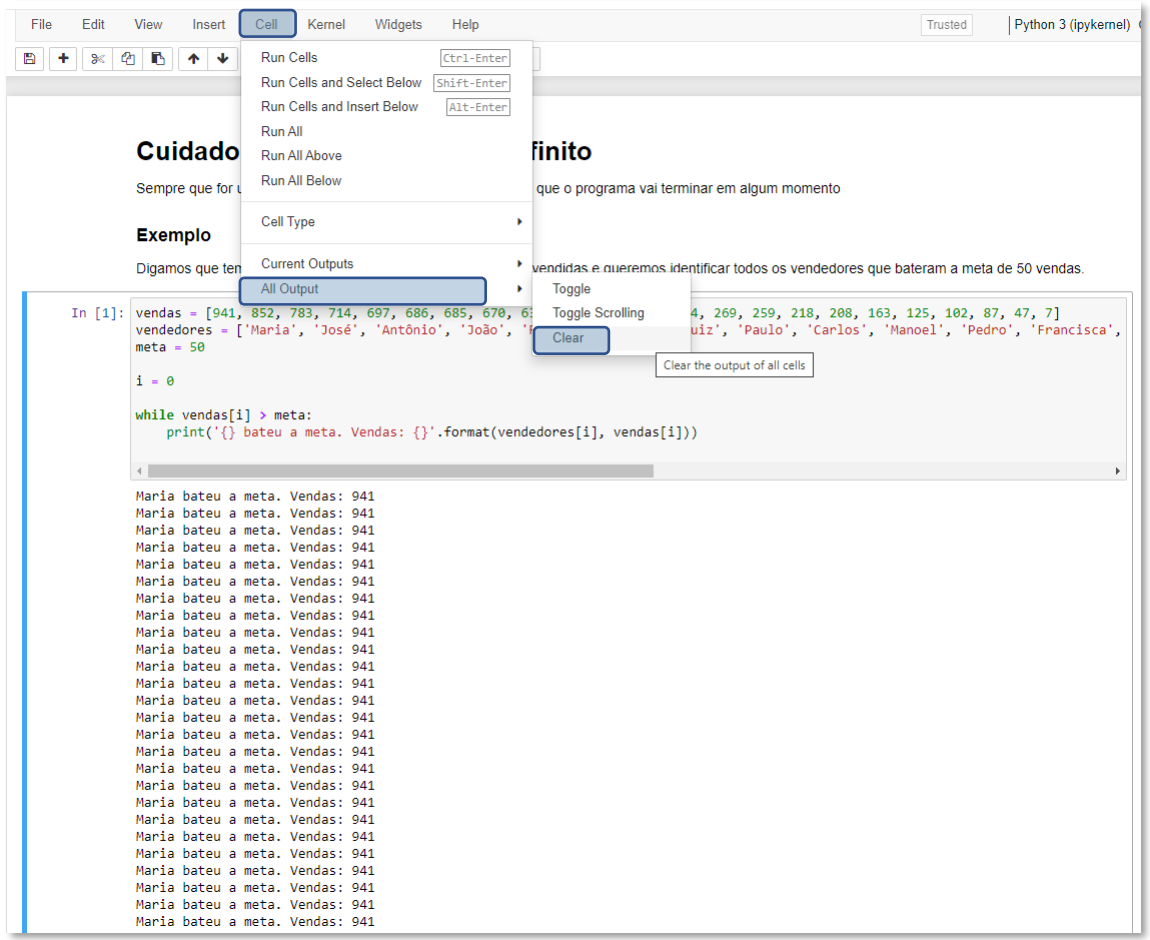
Existem 2 formas de parar esse loop infinito:

- Clicar em stop e corrigir o erro
- Pausar o Jupyter pelo prompt de comando apertando as teclar CTRL + C



Módulo 3 – Loop infinito no While

Para limpar o resultado clique em cell > all Outputs > Clear



Módulo 3 – Loop infinito no While

Agora para a gente corrigir o código vamos colocar para o valor de **i receber i + 1**.

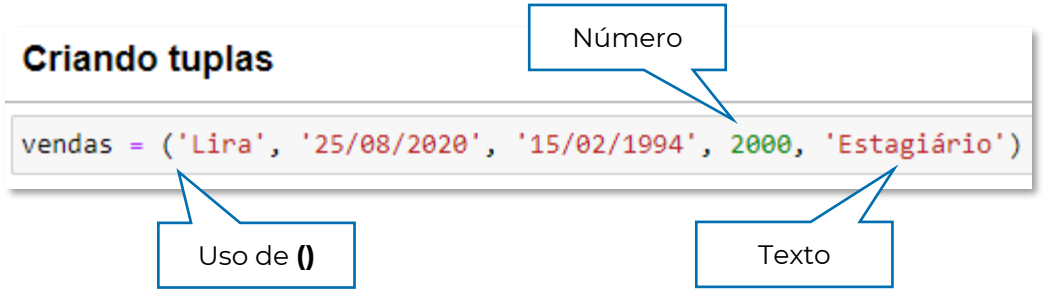
```
vendas = [941, 852, 783, 714, 697, 686, 685, 670, 631, 453, 386, 371, 294, 269, 259, 218, 208, 163, 125, 102, 87, 47, 7]
vendedores = ['Maria', 'José', 'Antônio', 'João', 'Francisco', 'Ana', 'Luiz', 'Paulo', 'Carlos', 'Manoel', 'Pedro', 'Francisca',
meta = 50

i = 0

while vendas[i] > meta:
    print('{} bateu a meta. Vendas: {}'.format(vendedores[i], vendas[i]))
    i += 1
```

Módulo 3 – Loop infinito no While

Agora sim o código será executado sem problemas.

```
vendas = [941, 852, 783, 714, 697, 686, 685, 670, 631, 453, 386, 371, 294, 269, 259, 218, 208, 163, 125, 102, 87, 47, 7]
vendedores = ['Maria', 'José', 'Antônio', 'João', 'Francisco', 'Ana', 'Luiz', 'Paulo', 'Carlos', 'Manoel', 'Pedro', 'Francisca',
meta = 50

i = 0

while vendas[i] > meta:
    print('{} bateu a meta. Vendas: {}'.format(vendedores[i], vendas[i]))
    i += 1
```

```
Maria bateu a meta. Vendas: 941
José bateu a meta. Vendas: 852
Antônio bateu a meta. Vendas: 783
João bateu a meta. Vendas: 714
Francisco bateu a meta. Vendas: 697
Ana bateu a meta. Vendas: 686
Luiz bateu a meta. Vendas: 685
Paulo bateu a meta. Vendas: 670
Carlos bateu a meta. Vendas: 631
Manoel bateu a meta. Vendas: 453
Pedro bateu a meta. Vendas: 386
Francisca bateu a meta. Vendas: 371
Marcos bateu a meta. Vendas: 294
Raimundo bateu a meta. Vendas: 269
Sebastião bateu a meta. Vendas: 259
Antônia bateu a meta. Vendas: 218
Marcelo bateu a meta. Vendas: 208
Jorge bateu a meta. Vendas: 163
Márcia bateu a meta. Vendas: 125
Geraldo bateu a meta. Vendas: 102
Adriana bateu a meta. Vendas: 87
```



A estrutura das Tuplas são parecidas com listas, mas elas têm algumas diferenças.

Estrutura da Tupla:

Tupla = (valor, valor**)**

As tuplas são imutáveis e podem ter dados heterogêneos.

Enquanto nas **listas** utilizávamos **[]**, nas **Tuplas** utilizamos ().

Algumas vantagens e desvantagens das Tuplas:

VANTAGENS	DESVANTAGENS
Proteger os dados e garantir que eles nunca serão modificados dentro do programa	Menos flexível que a lista, ou seja, não é possível alterar valores
Mais rápida que a lista para executar	Imutável
Aplicação para dados heterogêneos	

Para acessar um valor da Tupla fazemos de forma semelhante às listas:

Nome = vendas**[0]**

Apesar de criar a tupla com **()**, sempre que você for acessar o índice de algum dado você vai usar **[]**.

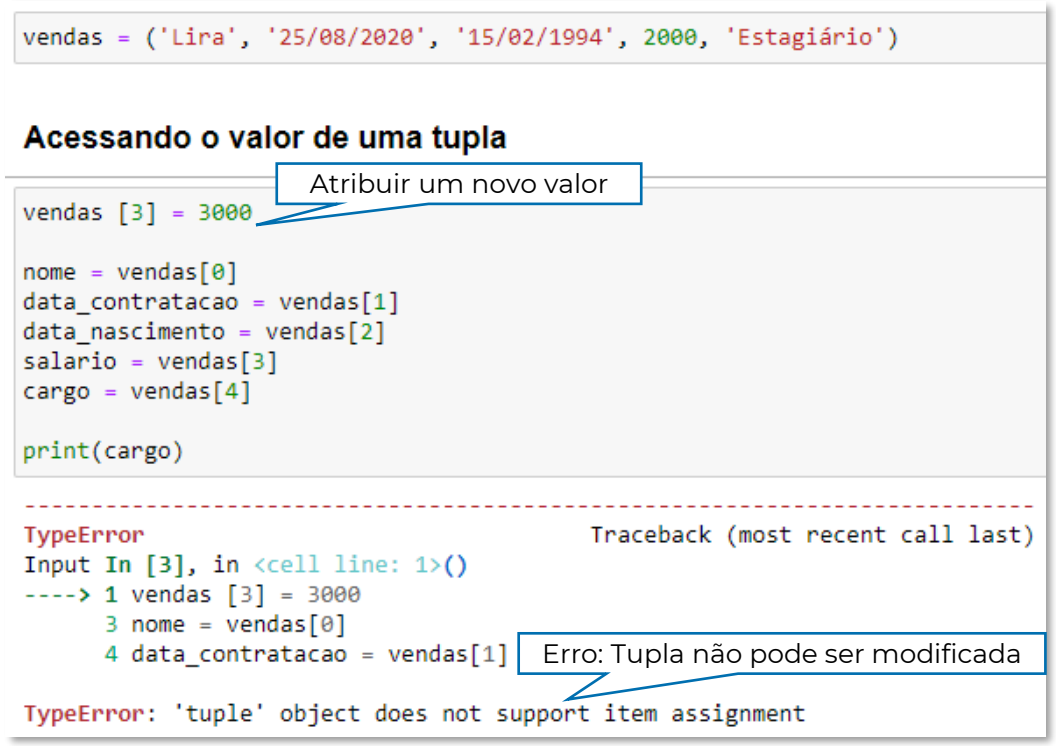
```
vendas = ('Lira', '25/08/2020', '15/02/1994', 2000, 'Estagiário')
```

Acessando o valor de uma tupla

```
nome = vendas[0]
data_contratacao = vendas[1]
data_nascimento = vendas[2]
salario = vendas[3]
cargo = vendas[4]

print(cargo)
```

```
Estagiário
```

Agora, se a gente tentar modificar o salário, por exemplo, de 2000 para 3000, repare que ele indica um erro e informa que a Tupla **não pode ser modificada**.



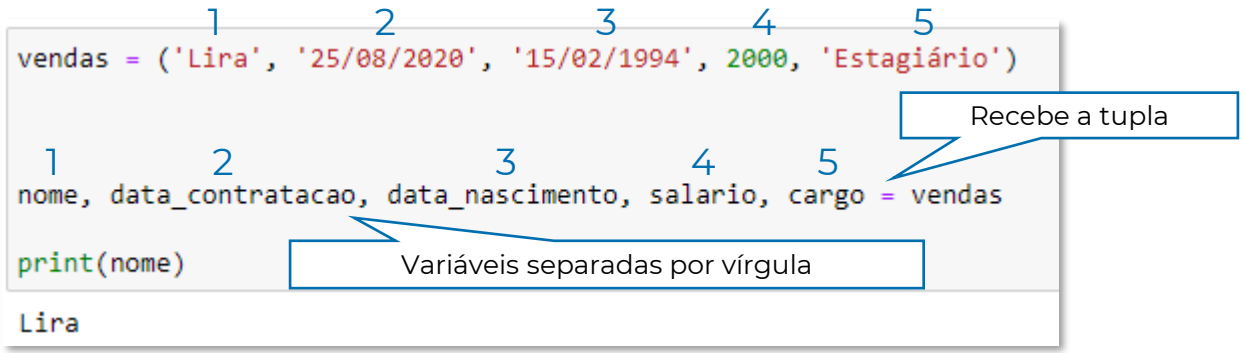
Abaixo listamos as diferenças das Tuplas e Listas:

	Tuplas	Listas
Dados heterogêneos	✓	✗
Mutável	✗	✓
Utiliza **[]** para escrever o código	✗	✓
Utiliza **()** para escrever o código	✓	✗

Módulo 3 – Unpacking em Tuplas

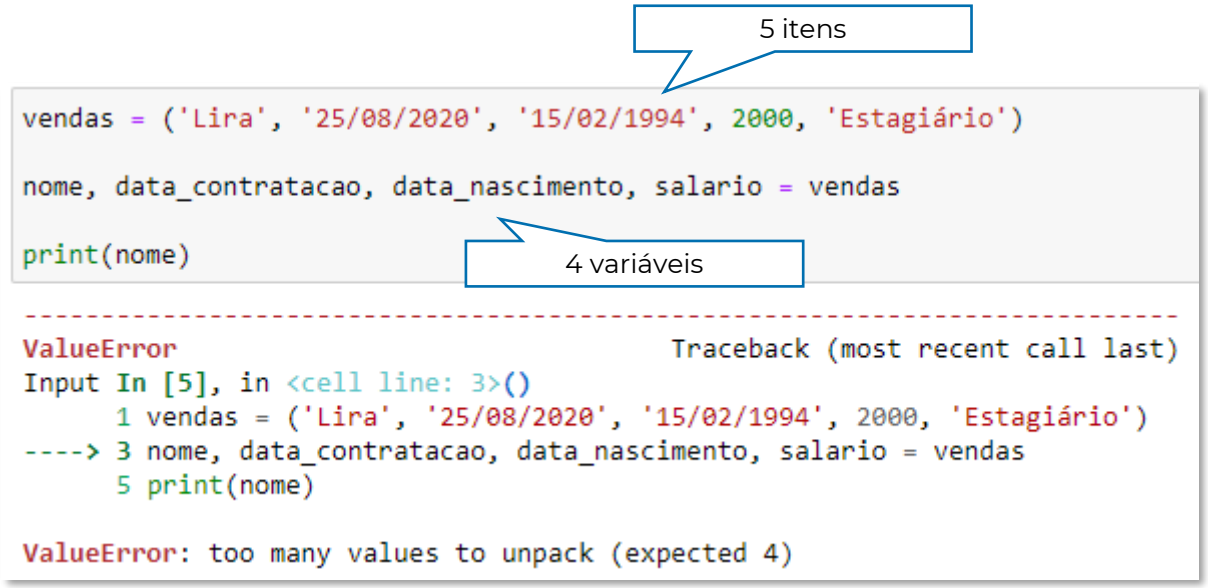
Agora, ao invés identificar o índice da informação, vamos desmembrar a tupla em várias variáveis e em uma única linha de código.

Perceba que a ordem da atribuição corresponde à sequência da tupla.



Atenção: se você tem 5 itens na tupla, você tem que ter 5 variáveis.

Por exemplo, abaixo estamos atribuindo uma tupla com 5 itens a 4 variáveis, dessa forma o Python gera um erro pois o número de itens é maior que o de variáveis.



Módulo 3 – Unpacking em Tuplas

O enumerate fornece 2 informações: **valor** e o **índice** dentro da lista.

No exemplo ao lado, ele vai utilizar a variável **venda** para acessar cada um dos valores da lista **vendas** e o **i** vai puxar o **índice**.

Repare que o **i, venda** é como se a gente estivesse fazendo o **unpacking** na Tupla.

```
vendas = [1000, 2000, 300, 300, 150]
funcionarios = ['João', 'Lira', 'Ana', 'Maria', 'Paula']

for i, venda in enumerate(vendas):
        print('{} vendeu {} unidades'.format(funcionarios[i], venda))
```

```
João vendeu 1000 unidades
Lira vendeu 2000 unidades
Ana vendeu 300 unidades
Maria vendeu 300 unidades
Paula vendeu 150 unidades
```

Módulo 3 – Unpacking em Tuplas

Então o **enumerate** puxa esse valor e transforma em uma tupla, que tem 2 informações: **índice** e o **valor**.

```
vendas = [1000, 2000, 300, 300, 150]
funcionarios = ['João', 'Lira', 'Ana', 'Maria', 'Paula']

for item in enumerate(vendas):
    print(item)
        #print('{} vendeu {} unidades'.format(funcionarios[i], venda))
```

```
(0, 1000)
(1, 2000)
(2, 300)
(3, 300)
(4, 150)
```

Módulo 3 – Unpacking em Tuplas

O enumerate tranforma esses valores em uma tupla que tem 2 informações.

Então, as **tuplas** vão ser muito úteis principalmente quando você estiver trabalhando com uma lista de informações que não representam a mesma coisa, por exemplo: **nomes** e **números**.

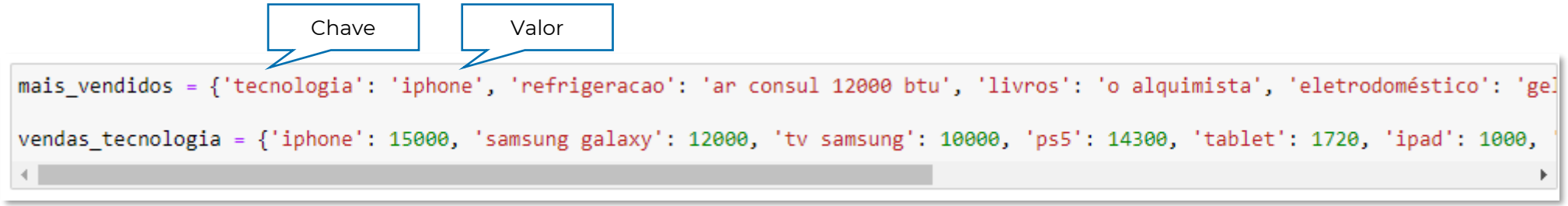
```
vendas = [1000, 2000, 300, 300, 150]
funcionarios = ['João', 'Lira', 'Ana', 'Maria', 'Paula']

for i, venda in enumerate(vendas):
        print('{} vendeu {} unidades'.format(funcionarios[i], venda))
```

```
João vendeu 1000 unidades
Lira vendeu 2000 unidades
Ana vendeu 300 unidades
Maria vendeu 300 unidades
Paula vendeu 150 unidades
```

O dicionário é uma outra estrutura de dados que armazena informações. Até o momento, já trabalhamos com lista e tupla. O dicionário, assim como a tupla, pode ser heterogêneo.

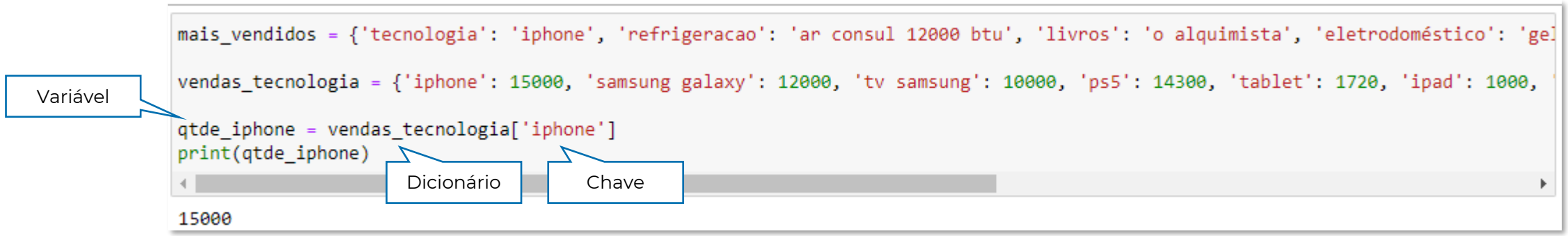
No dicionário, ao invés de você usar o índice para acessar a informação, você vai usar uma **chave**, que é basicamente um nome. Cada item do dicionário é um conjunto de uma chave e um valor.

Dicionario = {**chave**: valor, **chave**: valor, **chave**: valor...}



Imagine que a gente queira saber quanto foi vendido de iphone.

Vamos criar uma variável chamada **qtde_iphone** para receber o valor do dicionário **vendas_tecnologia** com a chave **'iphone'**.

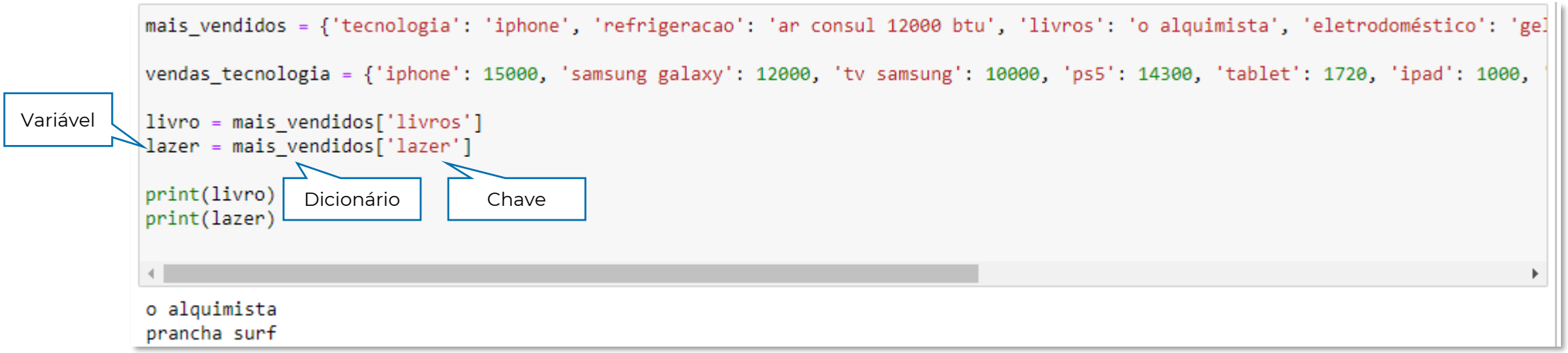


Atenção: Você não pode ter 2 produtos com a mesma chave.

Imagine que a gente queira o item mais vendido nas categorias 'livros' e 'lazer'.

Vamos criar uma variável chamada **livro** para receber o valor do dicionário **mais_vendidos** com a chave **'livros'**. O mesmo para o **lazer**.

A estrutura do código é mostrada abaixo:




```
mais_vendidos = {'tecnologia': 'iphone', 'refrigeracao': 'ar consul 12000 btu', 'livros': 'o alquimista', 'eletrodoméstico': 'ge

vendas_tecnologia = {'iphone': 15000, 'samsung galaxy': 12000, 'tv samsung': 10000, 'ps5': 14300, 'tablet': 1720, 'ipad': 1000,

livro = mais_vendidos['livros']
lazer = mais_vendidos['lazer']

print(livro)
print(lazer)
```

```
o alquimista
prancha surf
```

Módulo 3 – Dicionários em Python

Imagine que a gente queira saber quanto foi vendido de 'notebook asus' e de 'ipad.

Vamos fazer da mesma forma, igual ao exemplo ao lado.

```
mais_vendidos = {'tecnologia': 'iphone', 'refrigeracao': 'ar consul 12000 btu', 'livros': 'o alquimista', 'eletrodoméstico': 'ge

vendas_tecnologia = {'iphone': 15000, 'samsung galaxy': 12000, 'tv samsung': 10000, 'ps5': 14300, 'tablet': 1720, 'ipad': 1000,

livro = mais_vendidos['livros']
lazer = mais_vendidos['lazer']

print(livro)
print(lazer)

qtde_notebook_asus = vendas_tecnologia['not
qtde_ipad = vendas_tecnologia['ipad']

print(qtde_notebook_asus)
print(qtde_ipad)
```

Dicionário

Chave

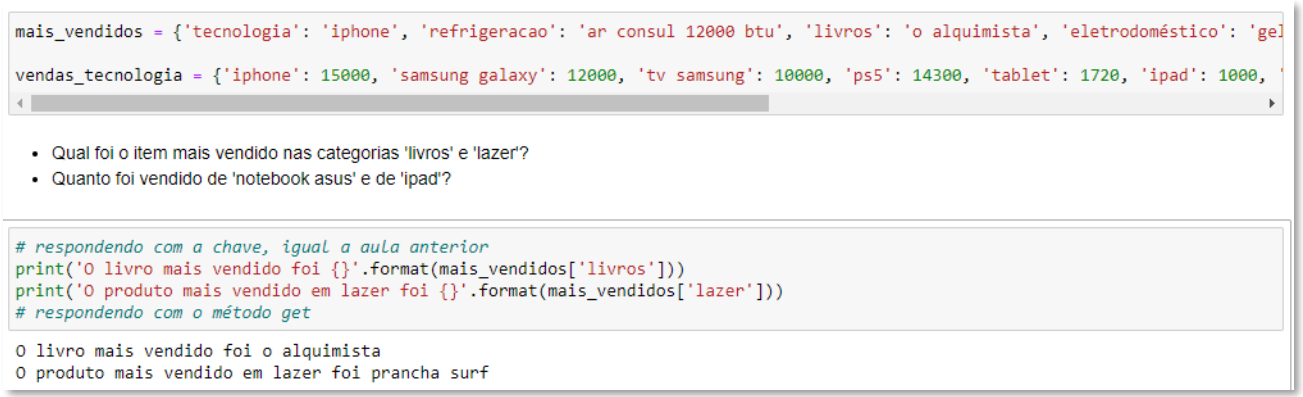
```
o alquimista
prancha surf
2450
1000
```

Módulo 3 – Pegar item Dicionário e Verificar Item Dicionário

Existem duas formas de buscar um valor:

- dicionario[‘CHAVE’]
- .get[‘CHAVE’]

Então, fazendo da forma da aula passada, vamos criar um código com **print**(**dicionário**[‘**chave**’]).



```
mais_vendidos = {'tecnologia': 'iphone', 'refrigeracao': 'ar consul 12000 btu', 'livros': 'o alquimista', 'eletrodoméstico': 'gel

vendas_tecnologia = {'iphone': 15000, 'samsung galaxy': 12000, 'tv samsung': 10000, 'ps5': 14300, 'tablet': 1720, 'ipad': 1000,
```

- Qual foi o item mais vendido nas categorias 'livros' e 'lazer'?
- Quanto foi vendido de 'notebook asus' e de 'ipad'?

```
# respondendo com a chave, igual a aula anterior
print('O livro mais vendido foi {}'.format(mais_vendidos['livros']))
print('O produto mais vendido em lazer foi {}'.format(mais_vendidos['lazer']))
# respondendo com o método get
```

```
O livro mais vendido foi o alquimista
O produto mais vendido em lazer foi prancha surf
```

Módulo 3 – Pegar item Dicionário e Verificar Item Dicionário

Pelo método chamado get(), vamos usar a mesma premissa anterior, só mudando o formato do código.

Estrutura:

print(dicionário.get[‘**chave**’]).

```
mais_vendidos = {'tecnologia': 'iphone', 'refrigeracao': 'ar consul 12000 btu', 'livros': 'o alquimista', 'eletrodoméstico': 'gel

vendas_tecnologia = {'iphone': 15000, 'samsung galaxy': 12000, 'tv samsung': 10000, 'ps5': 14300, 'tablet': 1720, 'ipad': 1000,
```

- Qual foi o item mais vendido nas categorias 'livros' e 'lazer'?
- Quanto foi vendido de 'notebook asus' e de 'ipad'?

```
# respondendo com a chave, igual a aula anterior
print('O livro mais vendido foi {}'.format(mais_vendidos['livros']))
print('O produto mais vendido em lazer foi {}'.format(mais_vendidos['lazer']))
# respondendo com o método get
print(vendas_tecnologia.get('notebook asus'))
print(vendas_tecnologia.get('ipad'))
```

```
O livro mais vendido foi o alquimista
O produto mais vendido em lazer foi prancha surf
2450
1000
```

Módulo 3 – Pegar item Dicionário e Verificar Item Dicionário

O que aconteceria se procurássemos por uma chave que não existe?

O método **.get** retornará **none** como resultado. Ou seja, o Python retorna dizendo que não existe nenhuma chave 'ipads' no dicionário.

Já pelo outro método, ao procurar uma chave inexistente, o Python retorna um **erro**.

GET

```
print(vendas_tecnologia.get('ipads'))
```

```
None
```

[CHAVE]

```
print(vendas_tecnologia['ipads'])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Input In [8], in <cell line: 1>()
----> 1 print(vendas_tecnologia['ipads'])

KeyError: 'ipads'
```

Módulo 3 – Pegar item Dicionário e Verificar Item Dicionário

```
if 'copo' in vendas_tecnologia:
    print(vendas_tecnologia['copo'])
else:
    print('Copo não está dentro da lista de produtos de tecnologia')
```

```
Copo não está dentro da lista de produtos de tecnologia
```

Se tentarmos procurar as vendas de "copo" na lista de vendas tecnologia, o que acontece?

A primeira forma de testar é com o **if**, utilizando o método das **chaves**.

Então, se o **copo** existe dentro do **dicionário**, retorna o valor de **copo**, caso contrário retorna o **print** "Copo não está dentro da lista de produtos de tecnologia"

Módulo 3 – Pegar item Dicionário e Verificar Item Dicionário

```
if  vendas_tecnologia.get('copo') == None:
    print('Copo não está dentro da lista de produtos de tecnologia')
else:
    print(vendas_tecnologia.get('copo'))
```

```
Copo não está dentro da lista de produtos de tecnologia
```

Uma outra forma é utilizando o **if** com o método **get**.

Então, se valor de **dicionário**.get('**copo**') é **None**, retorna o print "Copo não está dentro da lista de produtos de tecnologia". Caso contrário retorna o valor de copo.

Módulo 3 – Range

Agora a gente vai trabalhar com as diferentes opções dentro da estrutura do range:

- range(tamanho)
- range(inicio, fim)
- range(inicio, fim, passo)

Vamos ver na prática cada um desses exemplos.

Módulo 3 – Range

O uso mais comum do **range** é o tamanho, que é muito utilizado com o **for** para percorrer uma lista.

```
#uso mais comum no for:
produtos = ['arroz', 'feijao', 'macarrao', 'atum', 'azeite']
estoque = [50, 100, 20, 5, 80]

for i in range(5):
    print('{}: {} unidades em estoque'.format(produtos[i],estoque[i]))
```

```
arroz: 50 unidades em estoque
feijao: 100 unidades em estoque
macarrao: 20 unidades em estoque
atum: 5 unidades em estoque
azeite: 80 unidades em estoque
```

Módulo 3 – Range

A segunda forma é passar o início e o fim.

Por exemplo:

Print(**range**(1,10))

Ele vai percorrer 9 itens, porque o 10 é excludente.

```
#range com inicio e fim
print(range(1, 10))

#vamos olhar no for para entender

for i in range(1, 10):
    print(i)
```

```
range(1, 10)
1
2
3
4
5
6
7
8
9
```

Módulo 3 – Range

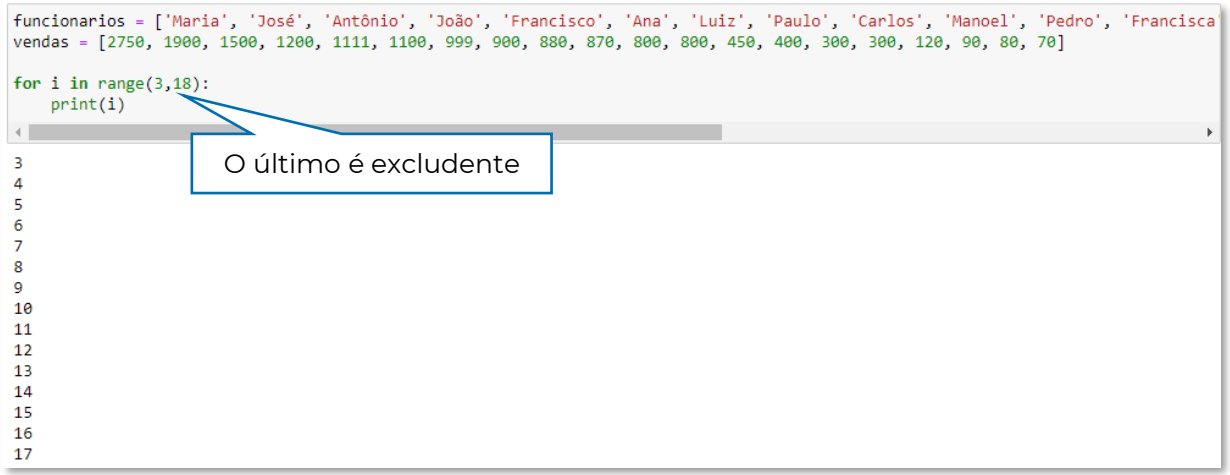
Agora um exemplo prático:

Modelo Jack Welch da G&E

Classe A: 10% melhor
Classe B: 80% mantém/busca melhorar
Classe C: 10% pior

Quem são os funcionários classe B?

Nesse caso, vamos querer as pessoas que estão no meio, entre 3 e 17.



Módulo 3 – Range

Resolvemos o exercício anterior colocando o print para indicar o funcionário e a quantidade de venda.

```
funcionarios = ['Maria', 'José', 'Antônio', 'João', 'Francisco', 'Ana', 'Luiz', 'Paulo', 'Carlos', 'Manoel', 'Pedro', 'Francisca
vendas = [2750, 1900, 1500, 1200, 1111, 1100, 999, 900, 880, 870, 800, 800, 450, 400, 300, 300, 120, 90, 80, 70]

print('Funcionários classe B:')

for i in range(3,18):
    print('{}: fez {} vendas'.format(funcionarios[i],vendas[i]))
```

```
Funcionários classe B:
João: fez 1200 vendas
Francisco: fez 1111 vendas
Ana: fez 1100 vendas
Luiz: fez 999 vendas
Paulo: fez 900 vendas
Carlos: fez 880 vendas
Manoel: fez 870 vendas
Pedro: fez 800 vendas
Francisca: fez 800 vendas
Marcos: fez 450 vendas
Raimundo: fez 400 vendas
Sebastião: fez 300 vendas
Antônia: fez 300 vendas
Marcelo: fez 120 vendas
Jorge: fez 90 vendas
```

Módulo 3 – Range

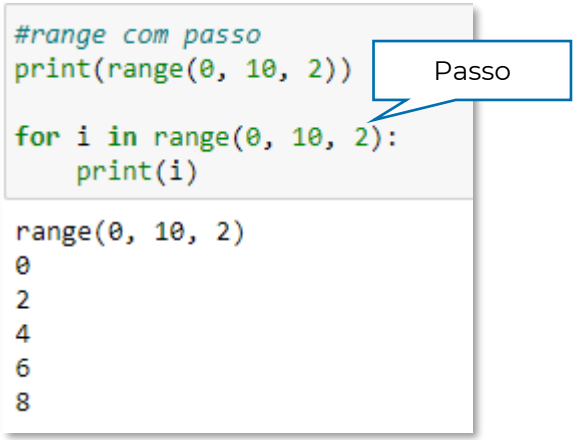
A última opção que a gente tem dentro do range é utilizar ele com o **passo**.

Como assim passo?

Vamos supor que eu quero ir de 0 a 10, mas com intervalo de 2 em 2. Nesse caso, a gente utiliza o range com passo.

Range(início, fim, passo)

Então, o código vai retornar os valores de 0, 2, 4, 6 e 8 (lembrando que o 10 é excludente).



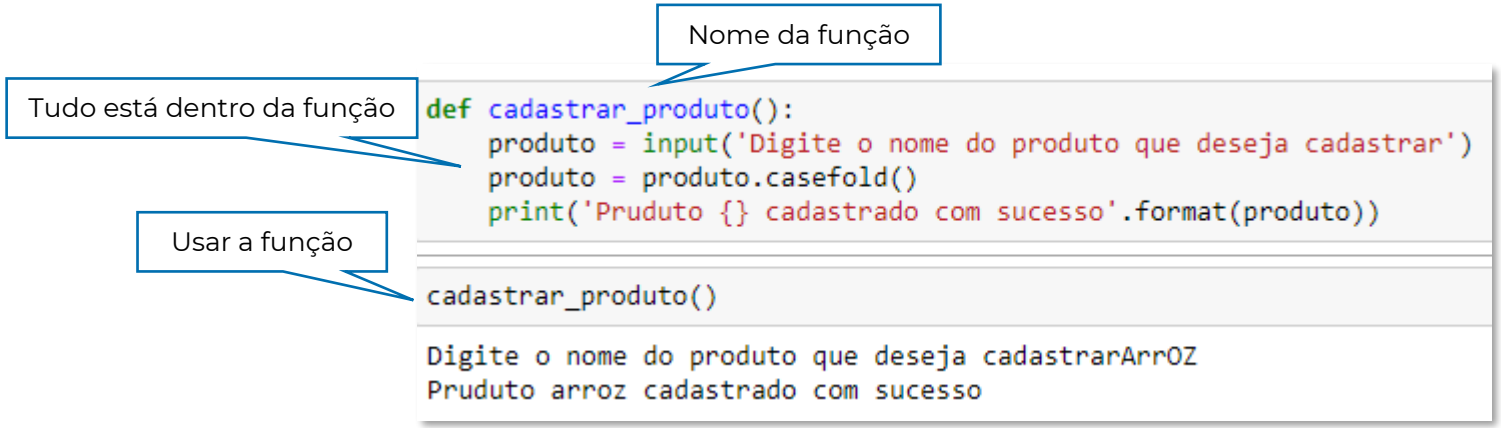
As **functions** são blocos de código que servem para um único propósito: fazem uma ação específica. Por exemplo: o comando print, que vimos algumas vezes anteriormente, é um exemplo de function. Só que ela é uma function nativa do Python, ou seja, ela já estava pronta pra gente usar.

Porém, podemos criar a nossa própria function e personalizar para o que a gente quiser. A estrutura para criar uma function é a seguinte:

Estrutura básica:

```
def nome_funcao():
    faça alguma coisa
    faça outra coisa
    return valor_final
```

Vamos criar uma função de cadastro de um Produto. Essa função deve garantir que o produto cadastrado está em letra minúscula.



Observação: a função .casefold() é utilizada para transformar a palavra em letra minúscula.

Módulo 3 – Retornar um valor na Function

Agora a gente vai retornar o valor para uma function. Mas, primeiro de tudo, o que é retornar um valor?

Na function, nós temos 2 opções: (1) executar as ações e (2) retornar um valor como resposta.

Retornar alguma coisa como resposta significa basicamente que a minha function vai dar um valor como resultado, que poderemos utilizar para alguma finalidade dentro do nosso código.

Módulo 3 – Retornar um valor na Function

Uma variável utilizada dentro de uma function só existe dentro daquela function.

Vejamos o exemplo ao lado, onde criamos uma function para o cadastro de produtos.

Quando utilizamos o código print logo após chamar a function, ele está tentando acessar a variável produto que está dentro da function, por isso aparece um erro.

E como fazemos para acessar o valor de alguma coisa que estava dentro da function?

Utilizamos o **return**.

Dentro da função

```
def cadastrar_produto():
    produto = input('Digite o nome do produto que deseja cadastrar')
    produto = produto.casefold()
    produto = produto.strip()
    #return produto
```

```
cadastrar_produto()

print(produto)
```

```
Digite o nome do produto que deseja cadastrarfeijão

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [2], in <cell line: 3>()
      1 cadastrar_produto()
----> 3 print(produto)

NameError: name 'produto' is not defined
```

Módulo 3 – Retornar um valor na Function

O **return** vai basicamente te dar a resposta da function.

O retorno geralmente é composto por listas, variáveis, dicionários, etc

Obs: Não podemos escrever nada abaixo do return dentro da function. O comando return deve ser o último comando de uma function.

```
def cadastrar_produto():
    produto = input('Digite o nome do produto que deseja cadastrar')
    produto = produto.casefold()
    produto = produto.strip()
    return produto
```

```
variavel_produto = cadastrar_produto()

print(variavel_produto)
```

```
Digite o nome do produto que deseja cadastrar   ArRoz
arroz
```

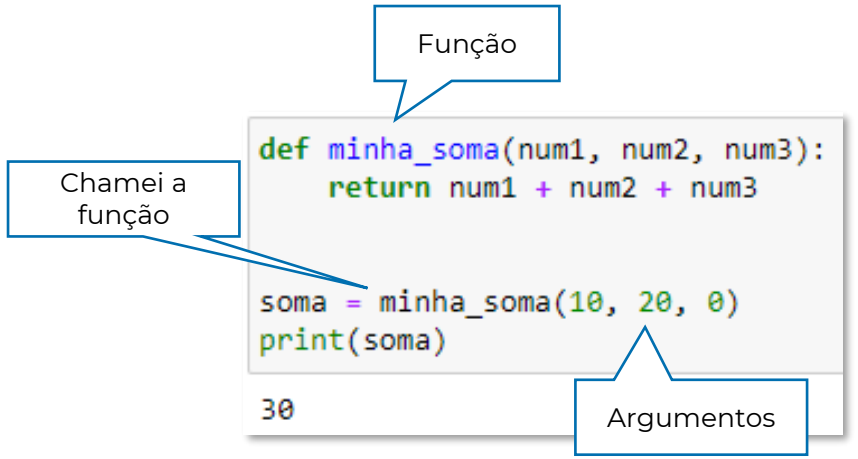
Módulo 3 – Argumentos e Parâmetros numa Function

Agora a gente vai aprender como passar um argumento ou parâmetro para uma função.

Argumento é a informação que você passa dentro dos parênteses. Por exemplo: **print**('**texto**'). Nesse caso, **texto** é o argumento/parâmetro.

O **print** é um exemplo de uma função que pode receber vários parâmetros para funcionar.

Só para mostrar o funcionamento, vamos criar uma função de soma, conforme o exemplo ao lado.



Módulo 3 – (Opcional) Aplicação em um Exemplo de argumento

Agora um exemplo prático:

Digamos que estamos criando um programa para categorizar os produtos de uma revendedora de bebidas.

Cada produto tem um código. O tipo de produto é dado pelas 3 primeiras letras do código. Exemplos:

- Vinho -> BEB12302
- Guaraná -> BSA11104

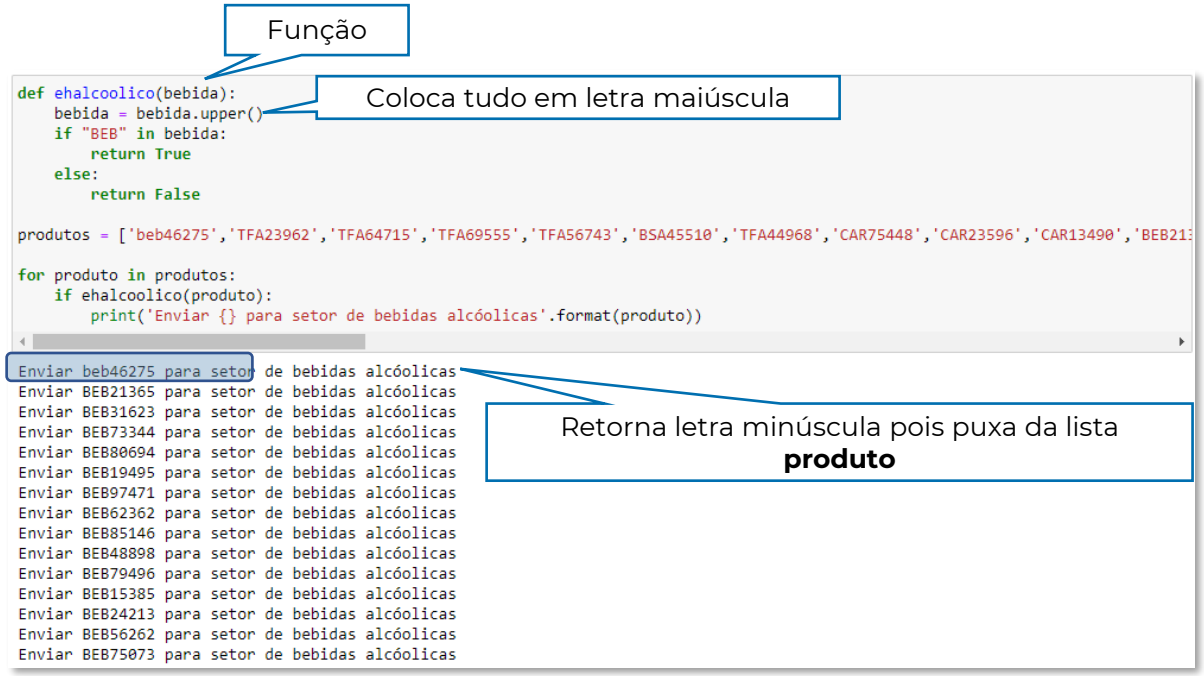
Repare que bebidas não alcóolicas começam com BSA e bebidas alcoolicas começam com BEB.

Módulo 3 – (Opcional) Aplicação em um Exemplo de argumento

Na prática temos uma lista de produtos e eu quero olhar todos os itens.

Antes de escrever o código vamos pensar no que precisamos fazer:

1. Percorrer toda a minha lista de produtos
2. Para cada produto, verificar se ele é bebida alcoólica
3. Se for bebida alcoólica, exibir a mensagem “Enviar...”




```
def ehalcoolico(bebida):
    bebida = bebida.upper()
    if "BEB" in bebida:
        return True
    else:
        return False

produtos = ['beb46275','TFA23962','TFA64715','TFA69555','TFA56743','BSA45510','TFA44968','CAR75448','CAR23596','CAR13490','BEB21

for produto in produtos:
    if ehalcoolico(produto):
        print('Enviar {} para setor de bebidas alcóolicas'.format(produto))
```

```
Enviar beb46275 para setor de bebidas alcóolicas
Enviar BEB21365 para setor de bebidas alcóolicas
Enviar BEB31623 para setor de bebidas alcóolicas
Enviar BEB73344 para setor de bebidas alcóolicas
Enviar BEB80694 para setor de bebidas alcóolicas
Enviar BEB19495 para setor de bebidas alcóolicas
Enviar BEB97471 para setor de bebidas alcóolicas
Enviar BEB62362 para setor de bebidas alcóolicas
Enviar BEB85146 para setor de bebidas alcóolicas
Enviar BEB48898 para setor de bebidas alcóolicas
Enviar BEB79496 para setor de bebidas alcóolicas
Enviar BEB15385 para setor de bebidas alcóolicas
Enviar BEB24213 para setor de bebidas alcóolicas
Enviar BEB56262 para setor de bebidas alcóolicas
Enviar BEB75073 para setor de bebidas alcóolicas
```

Módulo 3 – O que são módulos e qual a importância

Agora a gente vai aprender a importar um módulo no Python. Mas o que é um **módulo**?

Módulo é um arquivo no Python **cheio de código** e que dentro desse código você tem alguns objetos que já foram criados para você usar. Ele não vem por padrão no Python, você precisa importar para o seu código. Por isso, vamos precisar, sempre que necessário, importar esses módulos para dentro de nossos projetos.

Importância do módulo:

- Já tem muita coisa pronta, então você não precisa criar do zero.

- Se você souber usar Módulos e como usar um módulo novo, você vai conseguir fazer praticamente tudo no Python

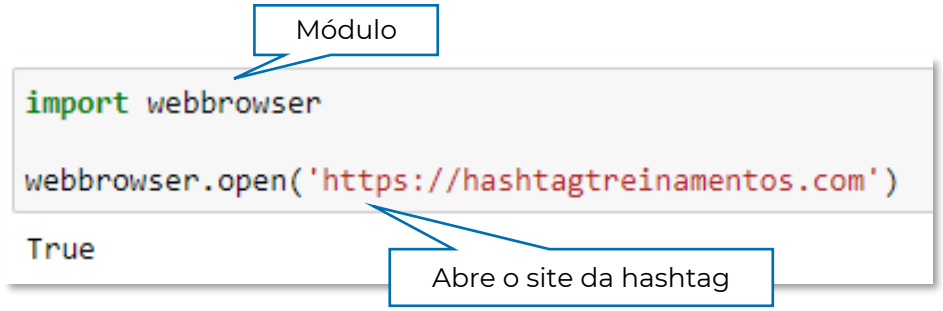
Módulo 3 – O que são módulos e qual a importância

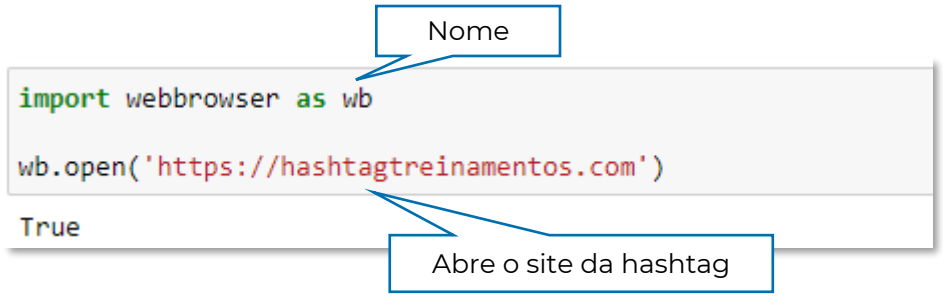
Para você importar um módulo a estrutura é:

Import módulo

Ou

Import módulo as nome (caso você queira mudar o módulo de nome)





MÓDULO 4

Pandas e Numpy: As bibliotecas básicas para Ciência de Dados

Não tem como a gente falar de Data Science sem falar do Pandas. O Pandas substitui um monte de funções que a gente iria ter que criar na mão.

Para eu poder importar qualquer biblioteca do Python a gente segue a estrutura:

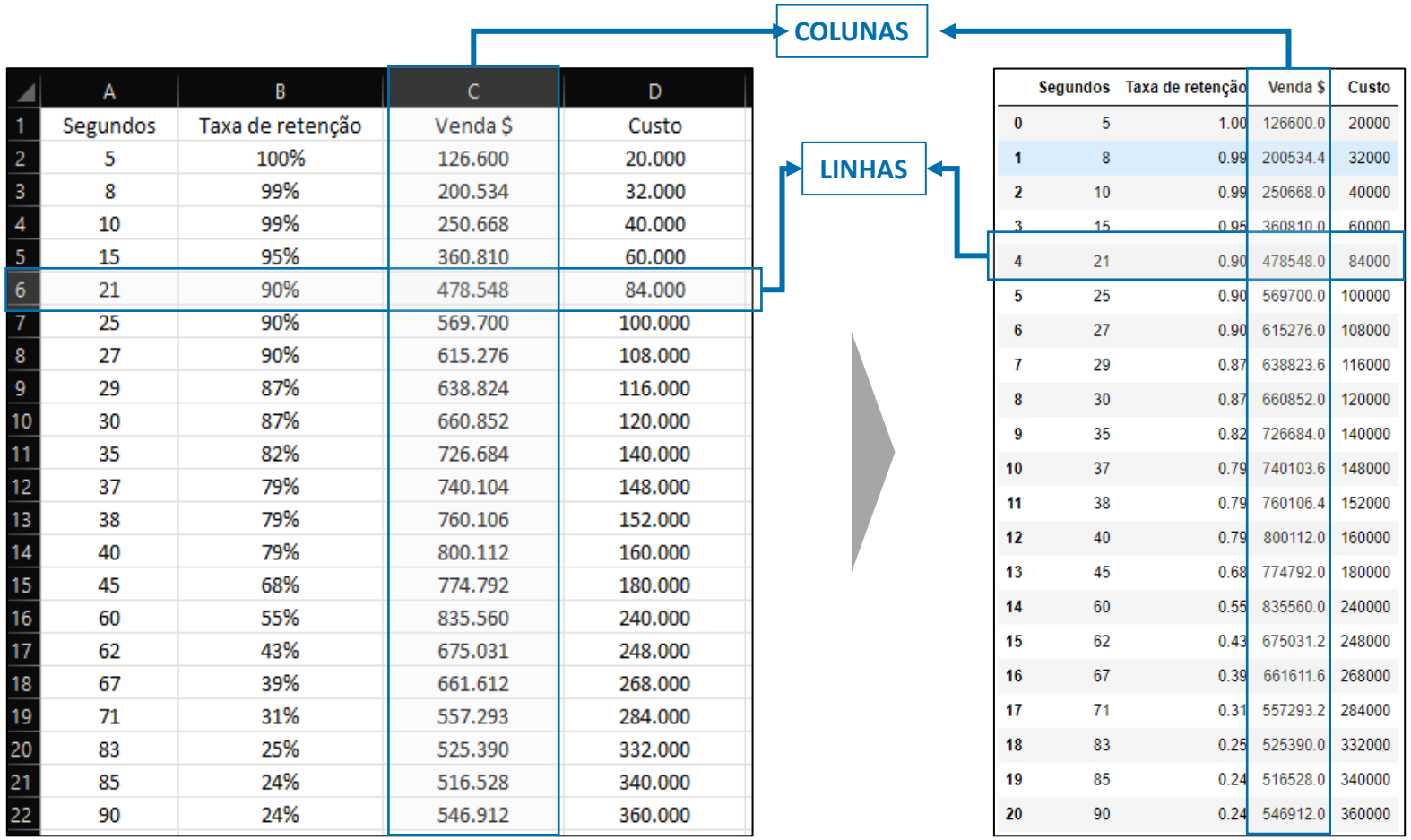
Import pandas **as** pd

Módulo 4 – As bibliotecas básicas para Ciência de Dados / Comparando Pandas e Excel

As estruturas ao lado são muito conhecidas por nós: uma composição de linhas e colunas, que chamamos de tabelas.

Essas estruturas estão presentes em todo tipo de ferramenta, como o Excel e o próprio Python.

A diferença é que no Excel chamamos de **tabela**, enquanto no Pandas (Python) a chamamos de **Dataframe**.



	A	B	C	D
1	Segundos	Taxa de retenção	Venda $	Custo
2	5	100%	126.600	20.000
3	8	99%	200.534	32.000
4	10	99%	250.668	40.000
5	15	95%	360.810	60.000
6	21	90%	478.548	84.000
7	25	90%	569.700	100.000
8	27	90%	615.276	108.000
9	29	87%	638.824	116.000
10	30	87%	660.852	120.000
11	35	82%	726.684	140.000
12	37	79%	740.104	148.000
13	38	79%	760.106	152.000
14	40	79%	800.112	160.000
15	45	68%	774.792	180.000
16	60	55%	835.560	240.000
17	62	43%	675.031	248.000
18	67	39%	661.612	268.000
19	71	31%	557.293	284.000
20	83	25%	525.390	332.000
21	85	24%	516.528	340.000
22	90	24%	546.912	360.000

	Segundos	Taxa de retenção	Venda $	Custo
0	5	1.00	126600.0	20000
1	8	0.99	200534.4	32000
2	10	0.99	250668.0	40000
3	15	0.95	360810.0	60000
4	21	0.90	478548.0	84000
5	25	0.90	569700.0	100000
6	27	0.90	615276.0	108000
7	29	0.87	638823.6	116000
8	30	0.87	660852.0	120000
9	35	0.82	726684.0	140000
10	37	0.79	740103.6	148000
11	38	0.79	760106.4	152000
12	40	0.79	800112.0	160000
13	45	0.68	774792.0	180000
14	60	0.55	835560.0	240000
15	62	0.43	675031.2	248000
16	67	0.39	661611.6	268000
17	71	0.31	557293.2	284000
18	83	0.25	525390.0	332000
19	85	0.24	516528.0	340000
20	90	0.24	546912.0	360000

Cada coluna desse dataframe a gente vai chamar de series.

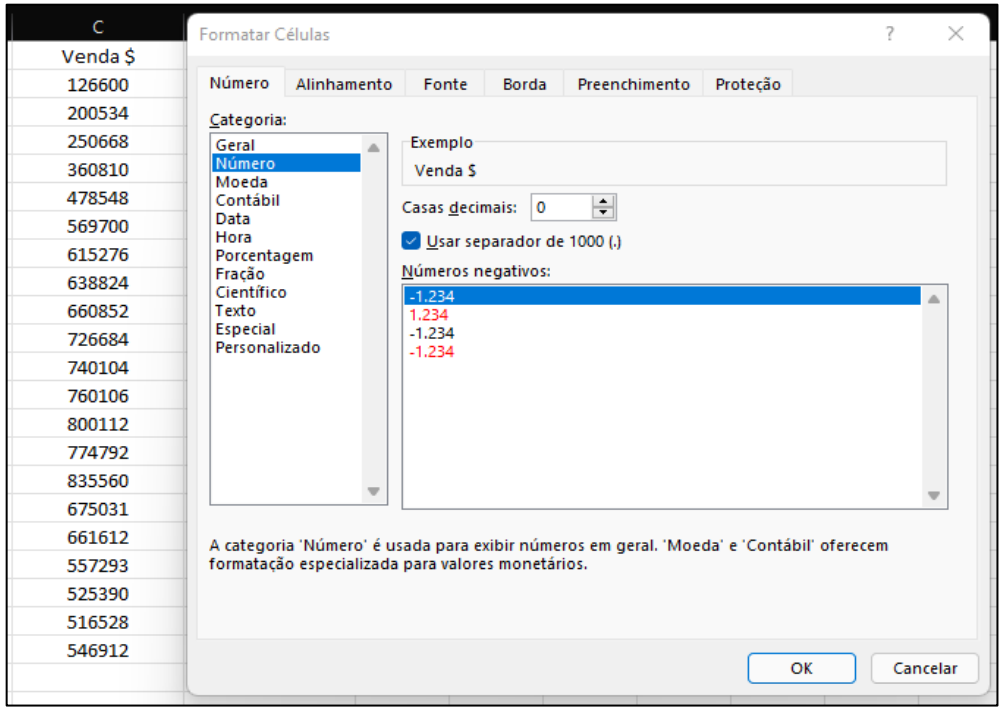
	A	B	C	D
1	Segundos	Taxa de retenção	Venda $	Custo
2	5	100%	126.600	20.000
3	8	99%	200.534	32.000
4	10	99%	250.668	40.000
5	15	95%	360.810	60.000
6	21	90%	478.548	84.000
7	25	90%	569.700	100.000
8	27	90%	615.276	108.000
9	29	87%	638.824	116.000
10	30	87%	660.852	120.000
11	35	82%	726.684	140.000
12	37	79%	740.104	148.000
13	38	79%	760.106	152.000
14	40	79%	800.112	160.000
15	45	68%	774.792	180.000
16	60	55%	835.560	240.000
17	62	43%	675.031	248.000
18	67	39%	661.612	268.000
19	71	31%	557.293	284.000
20	83	25%	525.390	332.000
21	85	24%	516.528	340.000
22	90	24%	546.912	360.000

DataFrame

Series

	Segundos	Taxa de retenção	Venda $	Custo
0	5	1.00	126600.0	20000
1	8	0.99	200534.4	32000
2	10	0.99	250668.0	40000
3	15	0.95	360810.0	60000
4	21	0.90	478548.0	84000
5	25	0.90	569700.0	100000
6	27	0.90	615276.0	108000
7	29	0.87	638823.6	116000
8	30	0.87	660852.0	120000
9	35	0.82	726684.0	140000
10	37	0.79	740103.6	148000
11	38	0.79	760106.4	152000
12	40	0.79	800112.0	160000
13	45	0.68	774792.0	180000
14	60	0.55	835560.0	240000
15	62	0.43	675031.2	248000
16	67	0.39	661611.6	268000
17	71	0.31	557293.2	284000
18	83	0.25	525390.0	332000
19	85	0.24	516528.0	340000
20	90	0.24	546912.0	360000

No Excel colocamos dados para cada célula. Já no pandas, os valores de uma mesma coluna devem respeitar o mesmo tipo de dados.

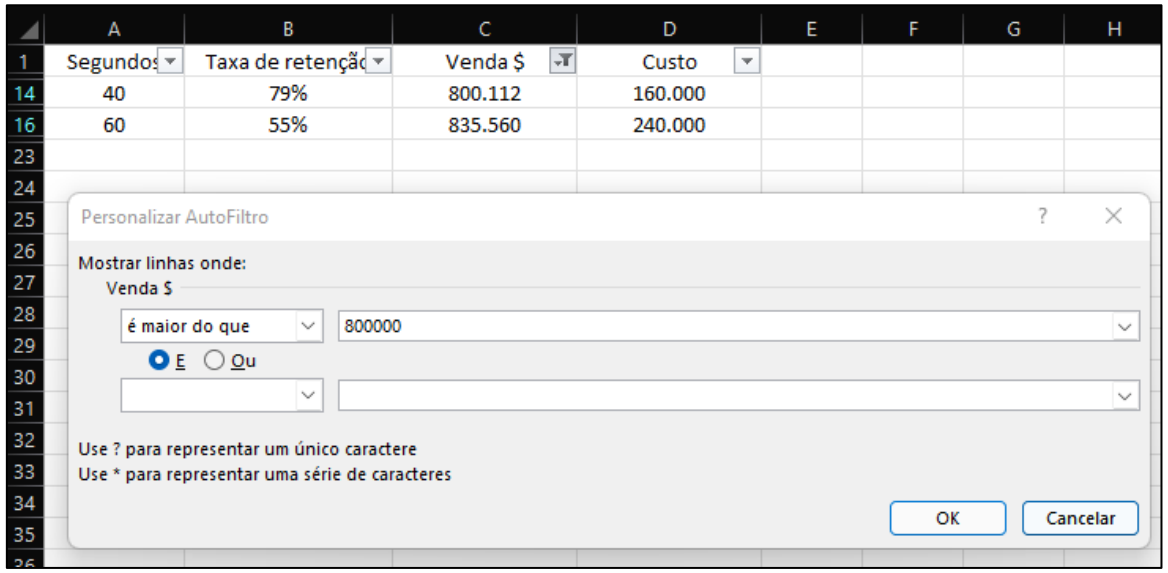



```
In [10]: base.dtypes

Out[10]: Segundos              int64
         Taxa de retenção    float64
         Venda $             float64
         Custo                 int64
         dtype: object
```

Todos os elementos da coluna são do mesmo tipo!

Módulo 4 – As bibliotecas básicas para Ciência de Dados / Comparando Pandas e Excel

A primeira etapa é pensar em como a gente faria no Excel e ver como se faz no Pandas. Por exemplo, no Excel, se quisermos usar um filtro, utilizamos uma ferramenta para isso; enquanto no Pandas, esse mesmo filtro será feito por meio de uma linha de código.




```
In [19]: base[base["Venda $"] > 800000]
```

Out[19]:

	Segundos	Taxa de retenção	Venda $	Custo
12	40	0.79	800112.0	160000
14	60	0.55	835560.0	240000

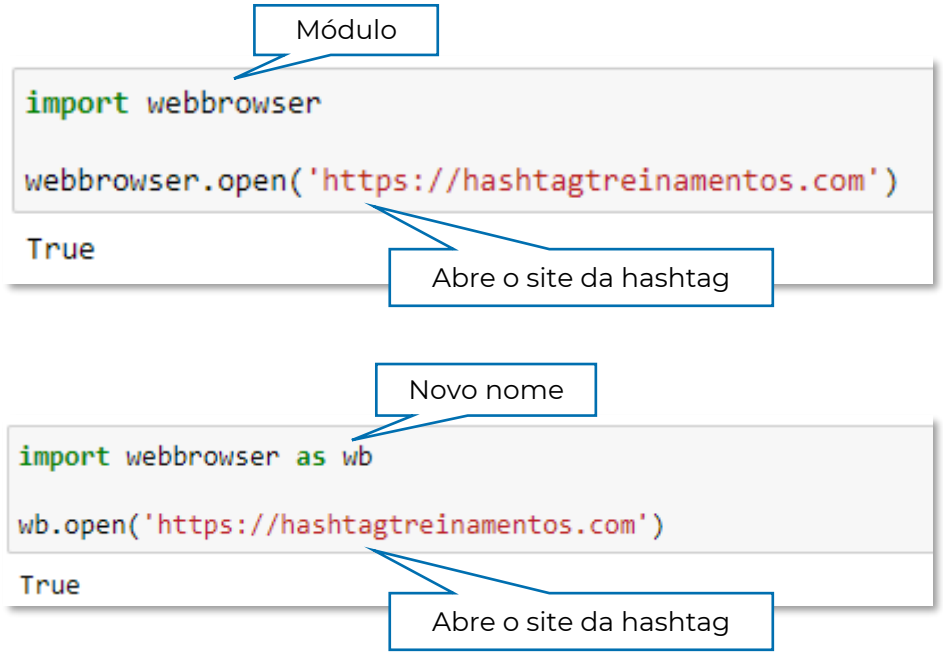
1 linha de comando

Para você importar um módulo a estrutura é:

Import módulo

Ou

Import módulo as nome (caso você queira mudar o módulo de nome)



Agora vamos ver na prática como funciona o Pandas.

Vamos pegar essa tabela que a gente fez no Excel e trazer para o Pandas, mas antes vamos importar a biblioteca utilizando o comando **import**.

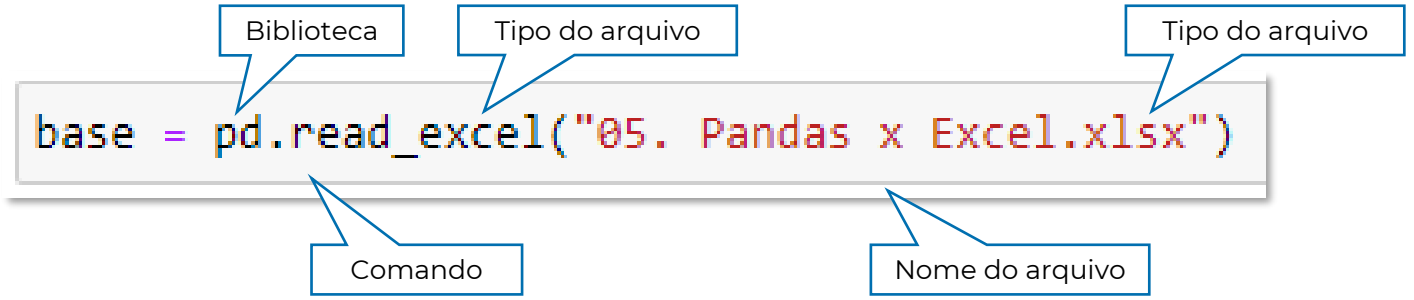
Atenção: Toda vez que a gente for trabalhar com qualquer biblioteca no Python, primeiro temos que importá-la.

Módulo 4 – Comparando Excel e Pandas na prática

Com o Pandas a gente consegue ler qualquer arquivo com vários formatos diferentes (CSV, html, excel, etc) utilizando a função **read**.

A estrutura é simples:
biblioteca.read_tipodearquivo(“**Nomedoarquivo.tipodoarquivo**”)

Nesse caso, estamos trabalhando com um arquivo em Excel, conforme a imagem abaixo:



Módulo 4 – Comparando Excel e Pandas na prática

No Excel a gente chama de tabela, enquanto no Pandas a gente chama de Dataframe.

Para ver a tabela no Excel basta eu abrir a tabela, enquanto no Python é necessário usar o comando **head**.

```
base.head(21)
```

	Segundos	Taxa de retenção	Venda $	Custo
0	5	1.00	126600.0	20000
1	8	0.99	200534.4	32000
2	10	0.99	250668.0	40000
3	15	0.95	360810.0	60000
4	21	0.90	478548.0	84000
5	25	0.90	569700.0	100000
6	27	0.90	615276.0	108000
7	29	0.87	638823.6	116000
8	30	0.87	660852.0	120000
9	35	0.82	726684.0	140000
10	37	0.79	740103.6	148000
11	38	0.79	760106.4	152000
12	40	0.79	800112.0	160000
13	45	0.68	774792.0	180000
14	60	0.55	835560.0	240000
15	62	0.43	675031.2	248000
16	67	0.39	661611.6	268000
17	71	0.31	557293.2	284000
18	83	0.25	525390.0	332000
19	85	0.24	516528.0	340000
20	90	0.24	546912.0	360000

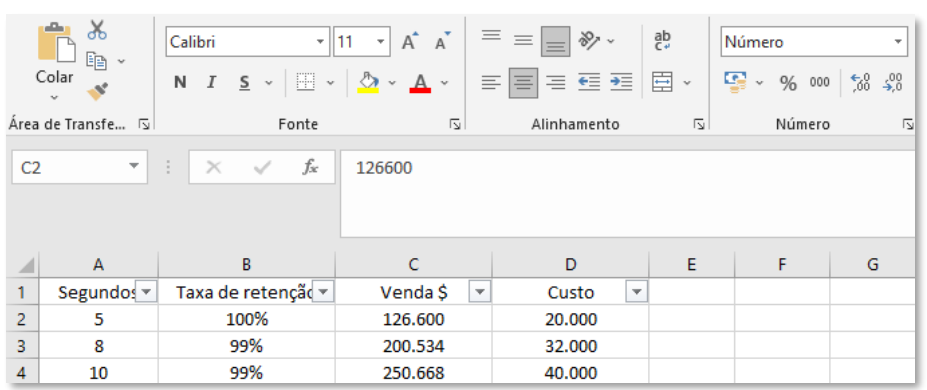
Módulo 4 – Comparando Excel e Pandas na prática

É possível ver o tipo de cada serie no Pandas utilizando o comando **dtypes**.

No Excel é possível mudar o tipo de uma linha, enquanto no Pandas modificamos para a coluna inteira.

Além disso, se um dos dados estiver como texto o Pandas vai entender a coluna inteira como texto, mas existem formas de tratar isso.

```
base.dtypes

Segundos              int64
Taxa de retenção    float64
Venda $             float64
Custo                 int64
dtype: object
```

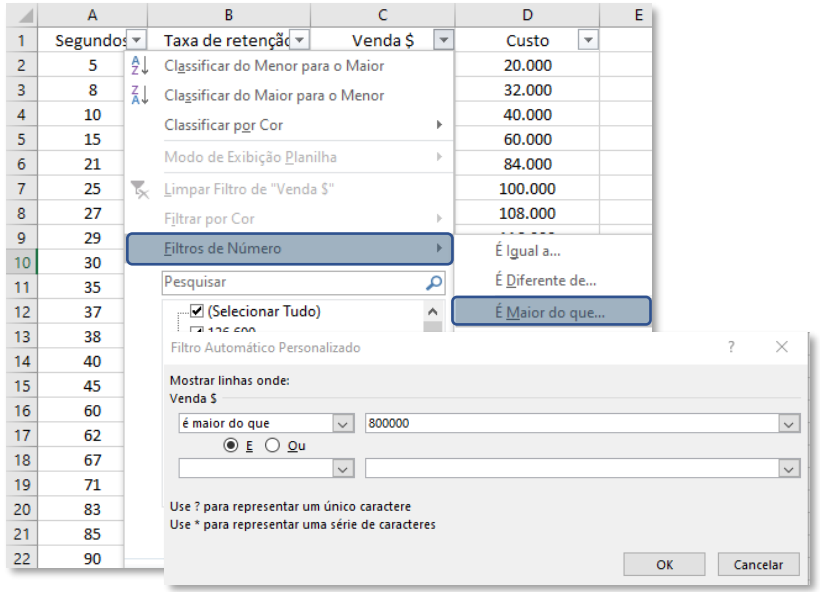


	A	B	C	D
1	Segundos	Taxa de retenção	Venda $	Custo
2	5	100%	126.600	20.000
3	8	99%	200.534	32.000
4	10	99%	250.668	40.000

Módulo 4 – Comparando Excel e Pandas na prática

Agora vamos comparar o uso de filtros:

Excel:



	A	B	C	D
1	Segundos	Taxa de retenção	Venda $	Custo
14	40	79%	800.112	160.000
16	60	55%	835.560	240.000

Pandas:

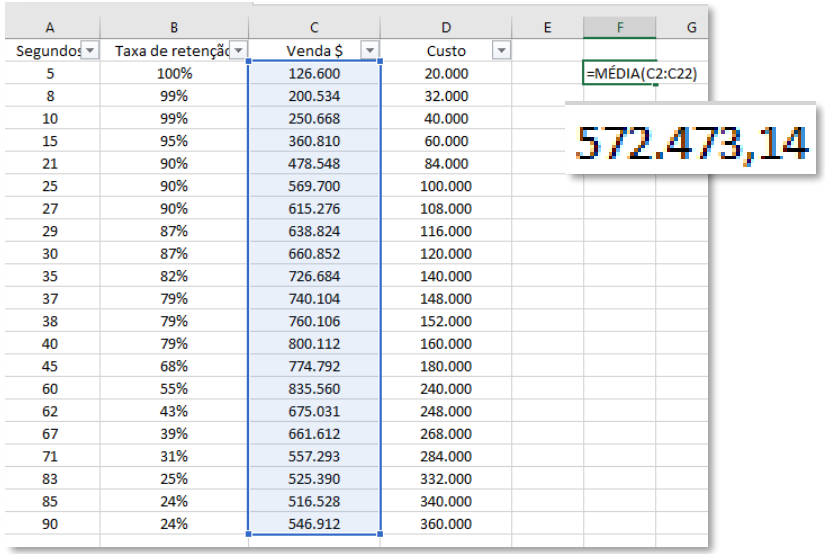
```
base[base["Venda $"] > 800000]
```

	Segundos	Taxa de retenção	Venda $	Custo
12	40	0.79	800112.0	160000
14	60	0.55	835560.0	240000

Módulo 4 – Comparando Excel e Pandas na prática

Ambos podem calcular a média utilizando uma fórmula:

Excel:



A	B	C	D	E	F	G
Segundos	Taxa de retenção	Venda $	Custo			
5	100%	126.600	20.000		=MÉDIA(C2:C22)	
8	99%	200.534	32.000			
10	99%	250.668	40.000			
15	95%	360.810	60.000			
21	90%	478.548	84.000			
25	90%	569.700	100.000			
27	90%	615.276	108.000			
29	87%	638.824	116.000			
30	87%	660.852	120.000			
35	82%	726.684	140.000			
37	79%	740.104	148.000			
38	79%	760.106	152.000			
40	79%	800.112	160.000			
45	68%	774.792	180.000			
60	55%	835.560	240.000			
62	43%	675.031	248.000			
67	39%	661.612	268.000			
71	31%	557.293	284.000			
83	25%	525.390	332.000			
85	24%	516.528	340.000			
90	24%	546.912	360.000			

572.473,14

Pandas:

```
base["Venda $"].mean()
572473.1428571427
```

Módulo 4 – Comparando Excel e Pandas na prática

No Pandas é possível obter todas as informações estatísticas de uma só vez utilizando o comando **describe**.

No Pandas é possível obter algumas informações de forma mais rápida que o Excel.

```
base["Venda $"].describe()
```

```
count         21.000000
mean      572473.142857
std       197752.952190
min       126600.000000
25%       516528.000000
50%       615276.000000
75%       726684.000000
max       835560.000000
Name: Venda $, dtype: float64
```

Módulo 4 – Introdução ao NumPy: A importância do NumPy

O NumPy é a base do Pandas.

É basicamente a biblioteca *Numerical Python*, ou seja, como o próprio nome diz, é uma biblioteca de **computação numérica** e é otimizada para cálculos mais pesados.

No NumPy vamos trabalhar muito com **arrays**, que são mais **rápidos** e mais **performáticos** que as listas.

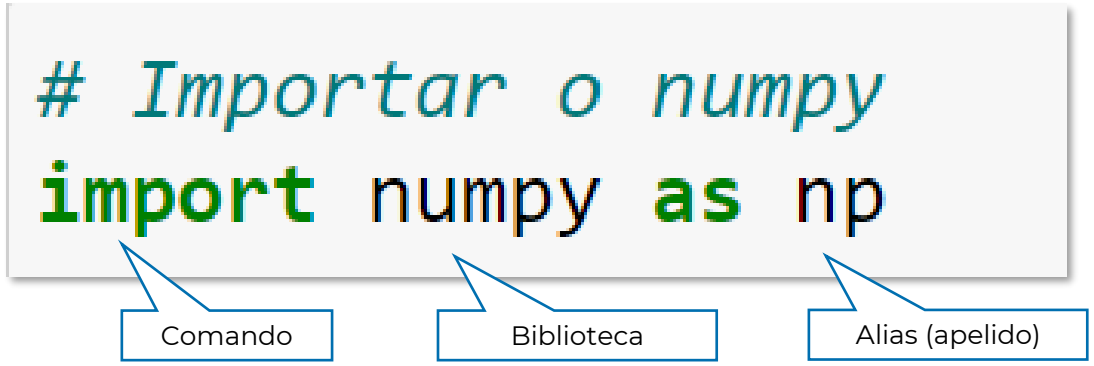
Toda vez que a gente precisa importar uma biblioteca, a gente usa o comando import para importar a biblioteca que a gente deseja, além da possibilidade de adicionar um apelido (alias) que vai ajudar a gente a escrever o código.

Toda vez que precisarmos importar uma biblioteca, vamos usar o comando **import.**

Estrutura:

import biblioteca **as** apelido

Utilizando o apelido fica mais fácil de digitar o código, pois dessa forma não precisamos escrever o nome completo da biblioteca, e sim apenas o apelido, o que otimiza bastante o nosso código.



Módulo 4 – Introdução ao NumPy: A importância do NumPy

Para exemplificar a importância do **numpy** vamos colocar uma lista com 2 valores: venda_valor e comissão.

Vamos tentar calcular quanto cada funcionário vai ganhar de comissão utilizando as listas.

Repare que ocorreu um erro, pois não podemos fazer essa operação entre listas, ou seja, não conseguimos fazer de uma forma rápida.

```
# Venda e comissão
venda_valor = [150000,230000,82000,143000,184000]
comissao = [5,8,8,5,12]
```

```
# Ao tentar fazer essa operação com listas, teremos um erro
venda_valor*(comissao/100)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [3], in <cell line: 2>()
      1 # Ao tentar fazer essa operação com listas, teremos um erro
----> 2 venda_valor*(comissao/100)

TypeError: unsupported operand type(s) for /: 'list' and 'int'
```

Módulo 4 – Introdução ao NumPy: A importância do NumPy

Para resolver esse problema a gente pode transformar essas 2 listas em um array.

Para transformar lista em um array:

meu_array = **np.array**(**minha_lista**)

Agora, com o array conseguimos aplicar aquela função que tentamos anteriormente.

```
# Transformando em um array
venda_valor = np.array(venda_valor)
comissao = np.array(comissao)
```

```
# Agora vamos conseguir fazer sendo um array
venda_valor*(comissao/100)
```

```
array([ 7500., 18400.,  6560.,  7150., 22080.])
```

Módulo 4 – Introdução ao NumPy: Propriedades de um array

Agora vamos entender as propriedades do array e como ele funciona.

Uma informação importante que vamos utilizar muito é a dimensão do array, que é feita pela estrutura abaixo.

Estrutura: **array.shape**

```
print(venda_valor)
```

```
[150000 230000  82000 143000 184000]
```

```
venda_valor.shape
```

```
(5,)
```

Módulo 4 – Introdução ao NumPy: Propriedades de um array

Para saber o tipo dos dados, podemos fazer:

Estrutura: **meu_array.dtype**

No exemplo ao lado, o resultado do tipo foi int32, ou seja, nesse array os nossos valores são inteiros.

```
venda_valor.dtype
```

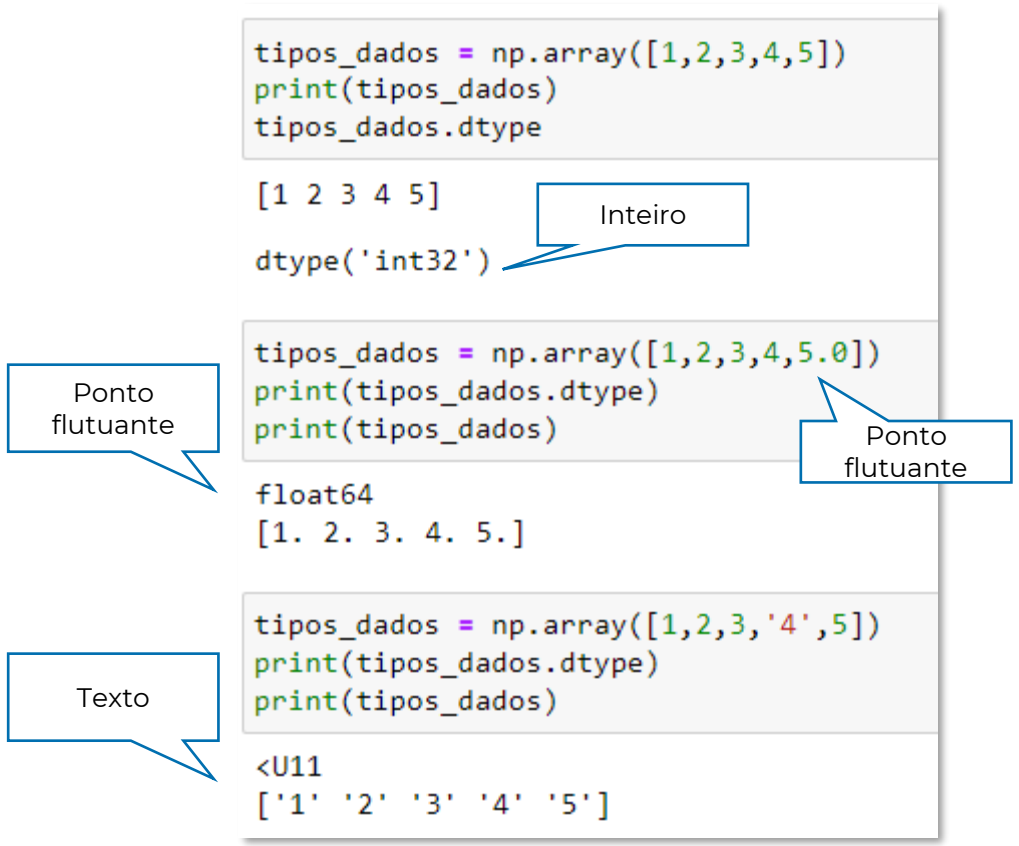
```
dtype('int32')
```

Módulo 4 – Introdução ao NumPy: Propriedades de um array

Se eu tenho dados diferentes tipos (inteiro e texto), ele converte tudo para o mesmo tipo de dado.

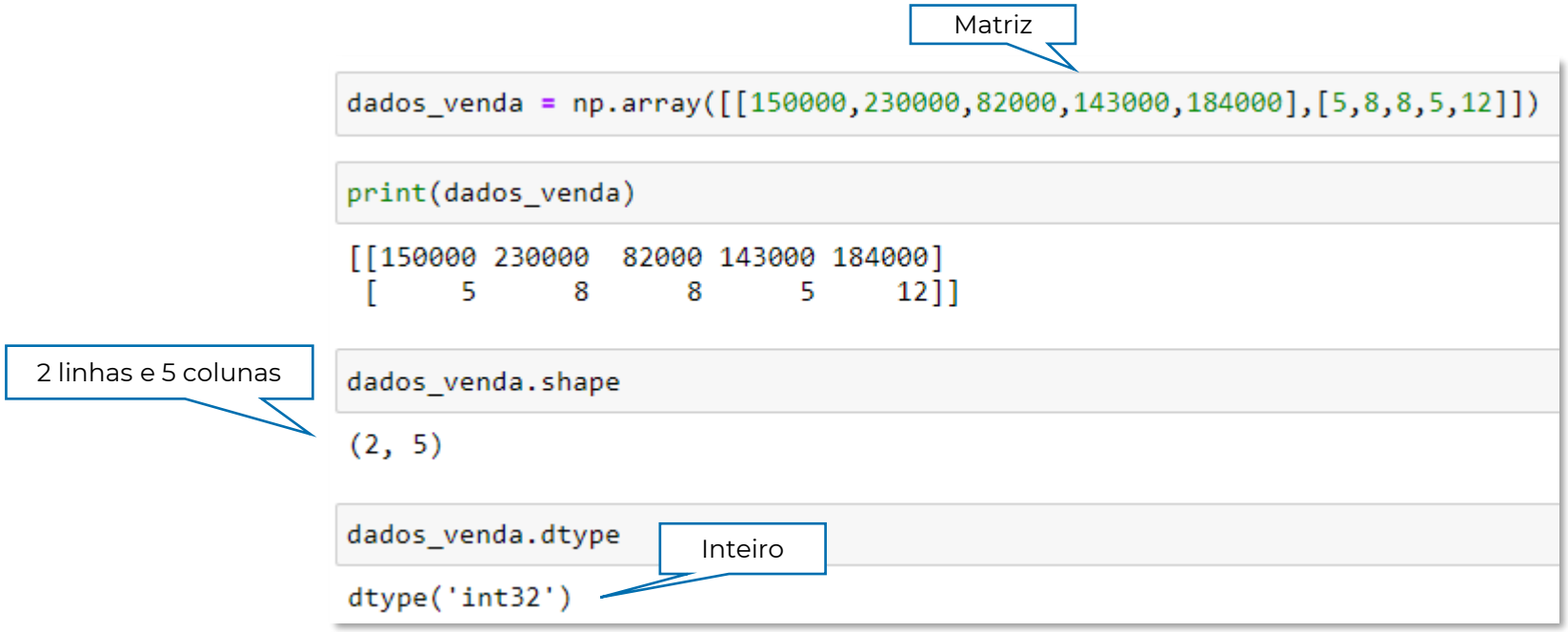
No exemplo ao lado, temos que apenas um valor é um ponto flutuante (float).

Porém, ao converter os valores para o mesmo tipo de dado, todos ficam com casa decimal.



Módulo 4 – Introdução ao NumPy: Propriedades de um array

O array de uma dimensão é o que chamamos de vetor e o de duas dimensões é o que chamamos de matriz.



Módulo 4 – Introdução ao NumPy: Trabalhando com um array

Agora vamos aprender como trabalhamos com um array.

Para buscar elementos em um array utilizamos as estruturas abaixo:

- Buscar por linha:

array[**N**]

- Buscar por linha e coluna:

array[**N**][**N**]

```
dados_venda
```

```
array([['Lucas', 'Bia', 'Jean', 'Gabi', 'Pedro'],
       ['150000', '230000', '82000', '143000', '184000']], dtype='<U11')
```

```
# Buscando a primeira linha (lembrando que o Python começa no 0)
dados_venda[0]
```

```
array(['Lucas', 'Bia', 'Jean', 'Gabi', 'Pedro'], dtype='<U11')
```

```
# Buscando na primeira linha, a segunda coluna
dados_venda[0][1]
```

```
'Bia'
```

Linha Coluna

```
dados_venda[1][2]
```

```
'82000'
```

Além disso, podemos mostrar as informações estatísticas de um array.

Vejamos alguns exemplos:

- Média:

Definição: Soma de todos os valores dividido pela quantidade de valores.

Estrutura: **np.mean(meu_array)**

```
# Média da primeira linha
np.mean(dados_venda[0])
```

```
157800.0
```

Primeira linha

```
# Média da segunda linha
np.mean(dados_venda[1])
```

Segunda linha

- Mediana:

Definição: Valor que está no meio da minha distribuição.

Estrutura: **np.median(meu_array)**

```
# Mediana da primeira linha
np.median(dados_venda[0])
```

```
150000.0
```

Primeira linha

Módulo 4 – Introdução ao NumPy: Trabalhando com um array

- Desvio Padrão:

np.std(meu_array)

```
# Desvio padrão da primeira linha
np.std(dados_venda[0])
```

```
48836.05225650411
```

Primeira linha

Módulo 4 – Introdução ao NumPy: Trabalhando com um array

```
# Considerando esses 2 arrays
array1 = np.array([1,2,3,4,5])
array2 = np.array([7,8,9,10,11])
```

```
# Soma
array1 + array2
```

```
array([ 8, 10, 12, 14, 16])
```

1 + 7

Podemos também realizar operações aritméticas, agregação, comparação, mínimo e máximo, etc.

- Soma:

Estrutura: **array1 + array2**

Módulo 4 – Introdução ao NumPy: Trabalhando com um array

- Multiplicação
  - Por um valor:

    Estrutura: **array1*valor**

  - De arrays:

    Estrutura: **array1* array2**

```
# Considerando esses 2 arrays
array1 = np.array([1,2,3,4,5])
array2 = np.array([7,8,9,10,11])
```

Multiplicação de 2 arrays

```
# Multiplicação por um valor
array1 * 2
```

```
array([ 2,  4,  6,  8, 10])
```

1 x 2

```
# Multiplicação de arrays
array1 * array2
```

```
array([ 7, 16, 27, 40, 55])
```

1 x 7

Módulo 4 – Introdução ao NumPy: Trabalhando com um array

- Potência

  Estrutura: **np.power**(**array2, array1**)

- Soma dos valores da primeira linha

  Estrutura: **np.sum**(**array1**)

```
# Considerando esses 2 arrays
array1 = np.array([1,2,3,4,5])
array2 = np.array([7,8,9,10,11])
```

7 elevado a 1

```
# Potência
np.power(array2,array1)
```

```
array([     7,     64,    729,  10000, 161051], dtype=int32)
```

```
# Soma dos valores da primeira linha
np.sum(array1)
```

```
15
```

1 + 2 + 3 + 4 +5 = 15

Módulo 4 – Introdução ao NumPy: Trabalhando com um array

- Soma acumulada

  Estrutura: **np.cumsum**(**array2, array1**)

- Mínimo

  Estrutura: **array1.min**

- Máximo

  Estrutura: **array1.máx**

```
# Considerando esses 2 arrays
array1 = np.array([1,2,3,4,5])
array2 = np.array([7,8,9,10,11])
```

```
# Soma acumulada dos valores da primeira linha
np.cumsum(array1)
```

```
array([ 1,  3,  6, 10, 15], dtype=int32)
```

1 + 2

3 + 3

```
# Valor mínimo
array1.min()
```

```
1
```

Menor valor do array 1

```
# Valor máximo
array1.max()
```

```
5
```

Maior valor do array 1

Módulo 4 – Introdução ao Pandas: Importando e visualizando uma base

O que é Pandas? Por que ele é importante?

O Pandas possui o Dataframe, que é uma estrutura que vamos utilizar em várias partes do nosso projeto de Data Science. Além disso, com o pandas conseguimos:

- Visualizar bases de dados;
- Analisar dados;
- Visualizar resumo estatístico das informações;
- Obter informações sobre a base de dados importada;
- Fazer gráficos e tabelas.

O pandas tem uma estrutura muito **amigável** e uma **alta performance**, muito fácil para trabalhar com dados e análises exploratórias.

Módulo 4 – Introdução ao Pandas: Importando e visualizando uma base

O primeiro passo é importar o Pandas. Fazemos isso com o seguinte código:

Import pandas **as** pd

```
# Importando o pandas
import pandas as pd
```

Com o Pandas a gente consegue ler qualquer arquivo com vários formatos diferentes (CSV, html, excel, etc), utilizando a função read.

A estrutura é simples:
biblioteca.read_tipodearquivo("**Nomedoarquivo**.**tipodoarquivo**")

Importar uma base em csv:
pd.read_csv("arquivo_excel.xlsx")

```
base = pd.read_csv("07. Pandas.csv")
```

Módulo 4 – Introdução ao Pandas: Importando e visualizando uma base

Fazendo um paralelo com o excel, ao abrir a base no Excel, já era possível visualizar todas a informações; no python não é tão direto assim.

Se a gente digitar apenas o código e não atribuí-lo a uma variável, o pandas só vai fazer a leitura, mas não irá salvar. Por isso, para atribuir a base em uma variável digitamos o nome da variável antes do sinal de igual (=).

Variável = **biblioteca.read_tipodearquivo**(“**Nomedoarquivo.tipodoarquivo**”)

No exemplo abaixo a nossa base está salva na variável base.

```
base = pd.read_csv("07. Pandas.csv")
```

Módulo 4 – Introdução ao Pandas: Importando e visualizando uma base

A função head mostra a visualização das **5 primeiras linhas** da base: títulos das colunas e os dados que existem nelas.

Dessa forma é possível ter uma noção de como está a base de dados importada.

Estrutura: **variável**.**head**()

As 5 primeiras linhas:
base.**head()**

```
# Visualizando as 5 primeiras linhas
base.head()
```

	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
0	2020-04-01	1	1	1	Tiger King: Murder, Mayhem ...	TV Show	Yes	Mar 20, 2020	9	90
1	2020-04-01	2	2	-	Ozark	TV Show	Yes	Jul 21, 2017	5	45
2	2020-04-01	3	3	2	All American	TV Show	NaN	Mar 28, 2019	9	76
3	2020-04-01	4	4	-	Blood Father	Movie	NaN	Mar 26, 2020	5	30
4	2020-04-01	5	5	4	The Platform	Movie	Yes	Mar 20, 2020	9	55

Módulo 4 – Introdução ao Pandas: Importando e visualizando uma base

As 5 primeiras linhas:
base.**head()**

```
# Visualizando as 5 primeiras linhas
base.head()
```

	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
0	2020-04-01	1	1	1	Tiger King: Murder, Mayhem ...	TV Show	Yes	Mar 20, 2020	9	90
1	2020-04-01	2	2	-	Ozark	TV Show	Yes	Jul 21, 2017	5	45
2	2020-04-01	3	3	2	All American	TV Show	NaN	Mar 28, 2019	9	76
3	2020-04-01	4	4	-	Blood Father	Movie	NaN	Mar 26, 2020	5	30
4	2020-04-01	5	5	4	The Platform	Movie	Yes	Mar 20, 2020	9	55

Com essa função já foi possível notar que na coluna "Last Week Rank" existem linhas com traço (-).

O que isso significa esse **traço** (-)?

Em alguns casos significa **coluna vazia**, mas isso vamos ver mais para frente.

Por enquanto vamos guardar a informação que existem algumas colunas com traço.

Módulo 4 – Introdução ao Pandas: Importando e visualizando uma base

```
base.head(15)
```

	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
0	2020-04-01	1	1	1	Tiger King: Murder, Mayhem ...	TV Show	Yes	Mar 20, 2020	9	90
1	2020-04-01	2	2	-	Ozark	TV Show	Yes	Jul 21, 2017	5	45
2	2020-04-01	3	3	2	All American	TV Show	NaN	Mar 28, 2019	9	76
3	2020-04-01	4	4	-	Blood Father	Movie	NaN	Mar 26, 2020	5	30
4	2020-04-01	5	5	4	The Platform	Movie	Yes	Mar 20, 2020	9	55
5	2020-04-01	6	6	-	Car Masters: Rust to Riches	TV Show	Yes	Sep 14, 2018	4	14
6	2020-04-01	7	10	-	Unorthodox	TV Show	Yes	Mar 26, 2020	2	5
7	2020-04-01	8	7	5	Love is Blind	TV Show	Yes	Feb 13, 2020	9	40
8	2020-04-01	9	8	-	Badland	Movie	NaN	Mar 26, 2020	4	11
9	2020-04-01	10	9	-	Uncorked	Movie	Yes	Mar 27, 2020	4	15
10	2020-04-02	1	1	1	Tiger King: Murder, Mayhem ...	TV Show	Yes	Mar 20, 2020	10	100
11	2020-04-02	2	2	-	Ozark	TV Show	Yes	Jul 21, 2017	6	54
12	2020-04-02	3	3	2	All American	TV Show	NaN	Mar 28, 2019	10	84
13	2020-04-02	4	-	-	Nailed It!	TV Show	Yes	Mar 9, 2018	1	7
14	2020-04-02	5	-	-	How to Fix a Drug Scandal	TV Show	Yes	Apr 1, 2020	1	6

O head pode ser utilizado para mostrar outras quantidades de linhas, além do padrão das 5 primeiras linhas.

Como por exemplo: visualizar 15 primeiras.

Vamos usar o código:

base.head(n° de linhas)

Módulo 4 – Introdução ao Pandas: Importando e visualizando uma base

Da mesma forma podemos visualizar as últimas 5 linhas, usando a função tail.

A estrutura é a mesma: **base**.**tail**()

	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
7095	2022-03-11	6	5	1	Worst Roommate Ever	TV Show	Yes	Mar 1, 2022	10	81
7096	2022-03-11	7	7	2	Vikings: Valhalla	TV Show	Yes	Feb 25, 2022	14	100
7097	2022-03-11	8	8	-	Shooter	Movie	NaN	Aug 1, 2014	3	7
7098	2022-03-11	9	9	7	Shrek 2	Movie	NaN	Mar 1, 2022	10	33
7099	2022-03-11	10	10	-	Shrek	Movie	NaN	May 1, 2018	7	12

Módulo 4 – Introdução ao Pandas: Importando e visualizando uma base

Podemos escolher da mesma forma que o **head** o número de linhas que queremos visualizar. Por exemplo: visualizar as últimas 3 linhas.

A estrutura é a mesma: **base**.**tail**(n° de linhas)

	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
7097	2022-03-11	8	8	-	Shooter	Movie	NaN	Aug 1, 2014	3	7
7098	2022-03-11	9	9	7	Shrek 2	Movie	NaN	Mar 1, 2022	10	33
7099	2022-03-11	10	10	-	Shrek	Movie	NaN	May 1, 2018	7	12

Módulo 4 – Introdução ao Pandas: Importando e visualizando uma base

Então basicamente com essas duas funções temos a visualização do topo e o final da nossa base.

Geralmente quando se tem algum problema com dado incompleto ou errado, com as primeiras e últimas linhas já é possível ter noção do erro.

```
In [7]: # Visualizando as 5 primeiras linhas
        base.head()
```

Out[7]:

	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
0	2020-04-01	1	1	1	Tiger King: Murder, Mayhem ...	TV Show	Yes	Mar 20, 2020	9	90
1	2020-04-01	2	2	-	Ozark	TV Show	Yes	Jul 21, 2017	5	45
2	2020-04-01	3	3	2	All American	TV Show	NaN	Mar 28, 2019	9	76
3	2020-04-01	4	4	-	Blood Father	Movie	NaN	Mar 26, 2020	5	30
4	2020-04-01	5	5	4	The Platform	Movie	Yes	Mar 20, 2020	9	55

As 5 últimas linhas:
base.tail()

```
In [6]: # Visualizando as 5 últimas linhas
        base.tail()
```

Out[6]:

	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
7095	2022-03-11	6	5	1	Worst Roommate Ever	TV Show	Yes	Mar 1, 2022	10	81
7096	2022-03-11	7	7	2	Vikings: Valhalla	TV Show	Yes	Feb 25, 2022	14	100
7097	2022-03-11	8	8	-	Shooter	Movie	NaN	Aug 1, 2014	3	7
7098	2022-03-11	9	9	7	Shrek 2	Movie	NaN	Mar 1, 2022	10	33
7099	2022-03-11	10	10	-	Shrek	Movie	NaN	May 1, 2018	7	12

Módulo 4 – Introdução ao Pandas: Importando e visualizando uma base

Uma outra informação importante é saber quantas linhas e colunas temos nessa base e para isso utilizamos a fórmula shape que vai indicar exatamente o **tamanho** da nossa base.

Estrutura: **base.shape**

Nessa caso, a base de dados tem 7100 linhas e 10 colunas.

```
# Tamanho da base
base.shape
```

```
(7100, 10)
```

Linhas

Colunas

Módulo 4 – Introdução ao Pandas: Importando e visualizando uma base

Uma outra opção é a função display, que mostra a visualização dessas três funções juntas (head, tail e shape), ou seja, mostra as primeiras e últimas 5 linhas, número de linhas e colunas.

Estrutura: **display**(**base**)

5 primeiras linhas

5 últimas linhas

	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
0	2020-04-01	1	1	1	Tiger King: Murder, Mayhem ...	TV Show	Yes	Mar 20, 2020	9	90
1	2020-04-01	2	2	-	Ozark	TV Show	Yes	Jul 21, 2017	5	45
2	2020-04-01	3	3	2	All American	TV Show	NaN	Mar 28, 2019	9	76
3	2020-04-01	4	4	-	Blood Father	Movie	NaN	Mar 26, 2020	5	30
4	2020-04-01	5	5	4	The Platform	Movie	Yes	Mar 20, 2020	9	55
...	...	...	...	...	...	...	...	...	...	...
7095	2022-03-11	6	5	1	Worst Roommate Ever	TV Show	Yes	Mar 1, 2022	10	81
7096	2022-03-11	7	7	2	Vikings: Valhalla	TV Show	Yes	Feb 25, 2022	14	100
7097	2022-03-11	8	8	-	Shooter	Movie	NaN	Aug 1, 2014	3	7
7098	2022-03-11	9	9	7	Shrek 2	Movie	NaN	Mar 1, 2022	10	33
7099	2022-03-11	10	10	-	Shrek	Movie	NaN	May 1, 2018	7	12

7100 rows × 10 columns

Linhas

Colunas

Então fiquem a vontade para escolher qual função usar: as três funções (**head, tail e shape**) ou a função **display**. Ambas podem ser utilizadas para começar a ver seus dados.

Então, notem que em todo projeto de ciência de dados será necessário fazer os passos:

A base que importamos está como DataFrame, uma forma fácil de confirmar é digitando a função type:

Estrutura: **type**(**base**)

```
type(base)
```

```
pandas.core.frame.DataFrame
```

Repare que o resultado retornado para o tipo é .**DataFrame**

Isso é muito importante para usar em qualquer parte do projeto de Ciência de Dados, pois as outras bibliotecas que vamos usar esperam receber já um **DataFrame** e se não estiver nesse tipo é necessário converter.

Essa é uma vantagem do pandas porque assim que ele importa a base, ela já é um **DataFrame**.

Supondo que temos um dicionário de funcionários, venda e comissão.

Ao utilizar o código **type**(**dic**) para descobrir o tipo, como mostra a imagem ao lado, o código retorna o resultado do tipo como dicionário (**dict**).

```
# Criando o dicionários
dic = {
    "funcionarios": ['Lucas','Bia','Jean','Gabi','Pedro'],
    "venda_valor": [150000,230000,82000,143000,184000],
    "comissao": [5,8,8,5,12]
}
```

```
type(dic)
```

```
dict
```

Como transformar o dicionário em DataFrame?

O pandas possui uma função:

pd.DataFrame(**variável**)

Essa função transforma automaticamente esses valores em um **DataFrame**.

Então para transformar o dicionário em DataFrame é só utilizar o código:

pd.DataFrame(**dic**)

```
# Transformando esse dicionário em um DataFrame
base_dic = pd.DataFrame(dic)

base_dic
```

	funcionarios	venda_valor	comissao
0	Lucas	150000	5
1	Bia	230000	8
2	Jean	82000	8
3	Gabi	143000	5
4	Pedro	184000	12

Apenas com uma linha de código transformamos um dicionário em um DataFrame, que é muito mais visual. Essa transformação nós vamos fazer muito no projeto de ciência de dados.

Dicionário:

```
dic
```

```
{'funcionarios': ['Lucas', 'Bia', 'Jean', 'Gabi', 'Pedro'],
 'venda_valor': [150000, 230000, 82000, 143000, 184000],
 'comissao': [5, 8, 8, 5, 12]}
```

DataFrame:

```
# Transformando esse dicionário em um DataFrame
base_dic = pd.DataFrame(dic)

base_dic
```

	funcionarios	venda_valor	comissao
0	Lucas	150000	5
1	Bia	230000	8
2	Jean	82000	8
3	Gabi	143000	5
4	Pedro	184000	12

Como dito anteriormente na comparação de Excel e pandas, no **Excel** temos as **colunas** e no **pandas** temos o que chamamos de **series**. Mas o que seriam essas **series**?

Por exemplo, se o nosso objetivo fosse visualizar somente a coluna funcionários desse DataFrame, essa coluna vai ser uma **series** e vai ter suas próprias propriedades.

```
series_funcionarios = base_dic["funcionarios"]

series_funcionarios
```

```
0    Lucas
1      Bia
2     Jean
3     Gabi
4    Pedro
Name: funcionarios, dtype: object
```

Exemplo: na series funcionários temos os valores e um índice, então ela é diferente de um **array** de valores e por isso temos que saber como trabalhar com ela.

Para passar os valores da series funcionários não podemos simplesmente digitar: **series_funcionário**.

Temos que utilizar uma função mais específica: **series_funcionarios.values**.

```
series_funcionarios.values
```

```
array(['Lucas', 'Bia', 'Jean', 'Gabi', 'Pedro'], dtype=object)
```

O mesmo podemos fazer para visualizar somente o índice: **series_funcionarios.index**

Então o índice começa em 0, termina em 5 (o 5 não é incluído) e varia de 1 em 1.

```
series_funcionarios.index
```

```
RangeIndex(start=0, stop=5,
```

É necessário entender as definições de DataFrame e Series para saber como trabalhar.

Agora quando digitamos o tipo da **serie_funcionários**, a função **type** retorna que é um tipo série.

```
type(series_funcionarios)
```

```
pandas.core.series.Series
```

Módulo 4 – Introdução ao Pandas: DataFrame e Series

Para transformar um Serie em DataFrame, vamos passar a coluna com colchetes duplos, ao invés colchetes simples.

Dataframe_funcionários = **base_dic**[[“funcionários”]]
series_funcionários = **base_dic** [“funcionários”]

```
dataframe_funcionarios = base_dic[["funcionarios"]]

dataframe_funcionarios
```

	funcionarios
0	Lucas
1	Bia
2	Jean
3	Gabi
4	Pedro

```
type(dataframe_funcionarios)
```

```
pandas.core.frame.DataFrame
```

```
series_funcionarios = base_dic["funcionarios"]

series_funcionarios
```

```
0    Lucas
1      Bia
2     Jean
3     Gabi
4    Pedro
Name: funcionarios, dtype: object
```

```
type(series_funcionarios)
```

```
pandas.core.series.Series
```

Módulo 4 – Introdução ao Pandas: tipos de dados, valores nulos e seleção de colunas

Podemos aplicar vários conceitos que vimos no NumPy no Pandas, como por exemplo:

- Mostrar o tipo de dados

  Estrutura: **base**.**dtypes**

```
# Mostrando somente o tipo dos dados
base.dtypes
```

```
As of                    object
Rank                      int64
Year to Date Rank        object
Last Week Rank           object
Title                    object
Type                     object
Netflix Exclusive        object
Netflix Release Date     object
Days In Top 10            int64
Viewership Score          int64
dtype: object
```

Objeto

Inteiro

Módulo 4 – Introdução ao Pandas: tipos de dados, valores nulos e seleção de colunas

Uma outra função que podemos utilizar é a **info**, que vai nos mostrar o tipo de dados e os valores vazios.

Estrutura: **base.info**

Entrada de 7100 e 4599 valores não nulos, então temos 2501 valores nulos

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7100 entries, 0 to 7099
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   As of                 7100 non-null   object
 1   Rank                  7100 non-null   int64
 2   Year to Date Rank     7100 non-null   object
 3   Last Week Rank        7100 non-null   object
 4   Title                 7100 non-null   object
 5   Type                  7100 non-null   object
 6   Netflix Exclusive     4599 non-null   object
 7   Netflix Release Date  7100 non-null   object
 8   Days In Top 10        7100 non-null   int64
 9   Viewership Score      7100 non-null   int64
dtypes: int64(3), object(7)
memory usage: 554.8+ KB
```

Módulo 4 – Introdução ao Pandas: tipos de dados, valores nulos e seleção de colunas

A função isnull informa se uma coluna é nula.

Estrutura: **base.isnull()**

```
base.isnull()
```

	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
0	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	True	False	False	False
3	False	False	False	False	False	False	True	False	False	False
4	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...
7095	False	False	False	False	False	False	False	False	False	False
7096	False	False	False	False	False	False	False	False	False	False
7097	False	False	False	False	False	False	True	False	False	False
7098	False	False	False	False	False	False	True	False	False	False
7099	False	False	False	False	False	False	True	False	False	False

Coluna nula

7100 rows × 10 columns

Módulo 4 – Introdução ao Pandas: tipos de dados, valores nulos e seleção de colunas

Para visualizar a coluna nula podemos utilizar a função **head**.

Estrutura: **base.head()**

```
base.head()
```

	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
0	2020-04-01	1	1	1	Tiger King: Murder, Mayhem ...	TV Show	Yes	Mar 20, 2020	9	90
1	2020-04-01	2	2	-	Ozark	TV Show	Yes	Jul 21, 2017	5	45
2	2020-04-01	3	3	2	All American	TV Show	NaN	Mar 28, 2019	9	76
3	2020-04-01	4	4	-	Blood Father	Movie	NaN	Mar 26, 2020	5	30
4	2020-04-01	5	5	4	The Platform	Movie	Yes	Mar 20, 2020	9	55

Coluna nula

Módulo 4 – Introdução ao Pandas: tipos de dados, valores nulos e seleção de colunas

Podemos utilizar a estrutura abaixo para contar os valores vazios por coluna:

Estrutura: **base.isnull().sum()**

Entrada de 7100 e 4599 valores não nulos, então temos 2501 valores nulos

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7100 entries, 0 to 7099
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   As of                 7100 non-null   object
 1   Rank                  7100 non-null   int64
 2   Year to Date Rank     7100 non-null   object
 3   Last Week Rank        7100 non-null   object
 4   Title                 7100 non-null   object
 5   Type                  7100 non-null   object
 6   Netflix Exclusive     4599 non-null   object
 7   Netflix Release Date  7100 non-null   object
 8   Days In Top 10        7100 non-null   int64
 9   Viewership Score      7100 non-null   int64
dtypes: int64(3), object(7)
memory usage: 554.8+ KB
```

Antes de falarmos de estatística, vamos precisar selecionar apenas as colunas que vamos trabalhar para evitar esse erro ao fazer a média.

```
base.mean()
```

```
C:\Users\JEANLU~1\AppData\Local\Temp/ipykernel_4228/4202722464.py:2: FutureWarning: Dropping of nuisance columns in DataFrame r
eductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid columns b
efore calling the reduction.
  base.mean()
```

Esse erro acontece porque estamos tentando tirar a média de colunas que não possuem esse valor.

Então o que nós vamos fazer é pegar apenas as colunas onde é possível tirar a média e separar em uma nova base.

Módulo 4 – Introdução ao Pandas: tipos de dados, valores nulos e seleção de colunas

É muito simples buscar por uma coluna no DataFrame, podemos usar o nome da coluna entre aspas ou usar o ponto.

Estrutura: **base**["**Title**“]

```
# Podemos usar o nome da coluna entre aspas
base["Title"]
```

```
0       Tiger King: Murder, Mayhem …
1                              Ozark
2                      All American
3                      Blood Father
4                      The Platform
                  ...
7095            Worst Roommate Ever
7096              Vikings: Valhalla
7097                        Shooter
7098                        Shrek 2
7099                          Shrek
Name: Title, Length: 7100, dtype: object
```

Estrutura: **base**.**Title**

```
# Ou usar o ponto
base.Title
```

```
0       Tiger King: Murder, Mayhem …
1                              Ozark
2                      All American
3                      Blood Father
4                      The Platform
                  ...
7095            Worst Roommate Ever
7096              Vikings: Valhalla
7097                        Shooter
7098                        Shrek 2
7099                          Shrek
Name: Title, Length: 7100, dtype: object
```

Observação: O ponto só vai funcionar quando o nome da coluna não possuir espaço.

Estrutura: **base**["**Title**“]

```
base['Year to Date Rank']
```

```
0        1
1        2
2        3
3        4
4        5
        ..
7095     5
7096     7
7097     8
7098     9
7099    10
Name: Year to Date Rank, Length: 7100, dtype: object
```

Estrutura: **base**.**Title**

```
base.Year to Date Rank
```

```
  Input In [35]
    base.Year to Date Rank
                  ^
SyntaxError: invalid syntax
```

Módulo 4 – Introdução ao Pandas: tipos de dados, valores nulos e seleção de colunas

Podemos utilizar funções na coluna, como por exemplo, o **value_counts**, que permite contar os valores de uma coluna:

Estrutura: **base**["**Title**“].**value_counts()**

Série mais assistida (428 vezes)

```
base["Title"].value_counts()
```

```
Cocomelon                          428
Ozark                               85
Cobra Kai                           81
Manifest                            80
The Queenâ??s Gambit                73
                                   ...
The Office                          1
Animals on the Loose: A You…        1
Dark                                1
The Secret Life of Pets 2           1
Step Up Revolution                  1
Name: Title, Length: 645, dtype: int64
```

Módulo 4 – Introdução ao Pandas: tipos de dados, valores nulos e seleção de colunas

Para selecionar mais de 1 coluna, basta passar uma lista com os valores que queremos buscar.

Estrutura:

lista_colunas = ["Rank","Days In Top 10","Viewership Score"]

base[**lista_colunas**]

Dessa forma, você consegue filtrar sua base por coluna.

```
# Para selecionar mais de uma coluna
lista_colunas = ["Rank","Days In Top 10","Viewership Score"]
base[lista_colunas]
```

	Rank	Days In Top 10	Viewership Score
0	1	9	90
1	2	5	45
2	3	9	76
3	4	5	30
4	5	9	55
...	...	...	...
7095	6	10	81
7096	7	14	100
7097	8	3	7
7098	9	10	33
7099	10	7	12

7100 rows × 3 columns

Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Da mesma forma que selecionamos as colunas que queríamos trabalhar, podemos fazer o mesmo com a base.

Estrutura:

base_estatistica = **base**[[“Colunas”]]

Para visualizar as primeiras linhas dessa nova base, podemos utilizar a função **head**.

```
# Selecionando a base que vamos mostrar informações estatísticas
base_estatistica = base[["Rank","Days In Top 10","Viewership Score"]]
```

Colunas

```
base_estatistica.head()
```

	Rank	Days In Top 10	Viewership Score
0	1	9	90
1	2	5	45
2	3	9	76
3	4	5	30
4	5	9	55

Para confirmar se todos os valores são inteiros podemos utilizar a função **info.**

Estrutura:

base_estatistica.info()

```
base_estatistica.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7100 entries, 0 to 7099
Data columns (total 3 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Rank              7100 non-null   int64
 1   Days In Top 10    7100 non-null   int64
 2   Viewership Score  7100 non-null   int64
dtypes: int64(3)
memory usage: 166.5 KB
```

7100 entrada = 7100 valores não nulos

Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Como nessa base todos os valores são inteiros, é possível aplicar funções estatísticas.

- Média de cada coluna

Estrutura: **base_estatistica.mean()**

- Contar a quantidade de registros

Estrutura: **base_estatistica.count()**

```
# Mostrando a média
base_estatistica.mean()
```

```
Rank                 5.500000
Days In Top 10      24.123662
Viewership Score   122.790141
dtype: float64
```

Média de cada coluna

```
# Mostrando a contagem de registros
base_estatistica.count()
```

```
Rank                7100
Days In Top 10      7100
Viewership Score    7100
dtype: int64
```

Quantidade de registros

Como nessa base todos os valores são inteiros, é possível aplicar funções estatísticas.

- Mediana

Estrutura: **base_estatistica.median()**

- Desvio Padrão

Estrutura: **base_estatistica.std()**

```
# Mediana
base_estatistica.median()
```

```
Rank                5.5
Days In Top 10      7.0
Viewership Score    50.0
dtype: float64
```

Mediana de cada coluna

```
# Desvio Padrão
base_estatistica.std()
```

```
Rank                2.872484
Days In Top 10      58.473789
Viewership Score    213.861642
dtype: float64
```

Desvio padrão de cada coluna

Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Aqui já podemos começar a fazer uma análise de dados, por exemplo, a média em Day in Top 10 é 24 e a mediana é 7, provavelmente podemos ter um **outlier**.

Além disso, o desvio padrão em Day in Top 10 é um valor alto, então esses dados estão muito distribuídos.

```
# Mostrando a média
base_estatistica.mean()
```

```
Rank                 5.500000
Days In Top 10      24.123662
Viewership Score   122.790141
dtype: float64
```

```
# Mostrando a contagem de registros
base_estatistica.count()
```

```
Rank               7100
Days In Top 10     7100
Viewership Score   7100
dtype: int64
```

```
# Mediana
base_estatistica.median()
```

```
Rank                 5.5
Days In Top 10       7.0
Viewership Score    50.0
dtype: float64
```

```
# Desvio Padrão
base_estatistica.std()
```

```
Rank                 2.872484
Days In Top 10      58.473789
Viewership Score   213.861642
dtype: float64
```

Desvio padrão grande

Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Podemos utilizar a função describe para trazer todo o resumo estatístico.

- Mediana

Estrutura: **base_estatistica.describe()**

```
base_estatistica.describe()
```

	Rank	Days In Top 10	Viewership Score
count	7100.000000	7100.000000	7100.000000
mean	5.500000	24.123662	122.790141
std	2.872484	58.473789	213.861642
min	1.000000	1.000000	1.000000
25%	3.000000	3.000000	19.000000
50%	5.500000	7.000000	50.000000
75%	8.000000	18.000000	128.000000
max	10.000000	428.000000	1474.000000

Valor mínimo

Quartil de 50% é a mediana

Valor máximo

Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Imagine que agora observamos o desvio padrão alto na coluna Days in Top 10 e queremos nos aprofundar nessas colunas.

Para isso nós podemos fazer filtros na base no pandas, da mesma forma que fazemos no Excel.

```
base_estatistica.describe()
```

	Rank	Days In Top 10	Viewership Score
count	7100.000000	7100.000000	7100.000000
mean	5.500000	24.123662	122.790141
std	2.872484	58.473789	213.861642
min	1.000000	1.000000	1.000000
25%	3.000000	3.000000	19.000000
50%	5.500000	7.000000	50.000000
75%	8.000000	18.000000	128.000000
max	10.000000	428.000000	1474.000000

Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Da mesma forma que filtramos coluna, podemos fazer para linha, só que precisamos colocar uma condição:

Estrutura:
base[base[“Coluna”]condição]

É possível notar que a maioria dos títulos que ficaram mais de 100 dias no top 10 é o Cocomelon.

```
# Filtrando apenas programas que ficaram mais de 100 dias no Top 10
base[base["Days In Top 10"] > 100]
```

Coluna

Condição

Maioria com o título Cocomelon

Maior que 100

	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
2896	2021-01-15	7	7	10	Cocomelon	TV Show	NaN	Jun 1, 2020	101	291
2909	2021-01-16	10	7	9	Cocomelon	TV Show	NaN	Jun 1, 2020	102	292
2919	2021-01-17	10	10	9	Cocomelon	TV Show	NaN	Jun 1, 2020	103	293
3019	2021-01-27	10	-	-	Cocomelon	TV Show	NaN	Jun 1, 2020	104	294
3029	2021-01-28	10	10	-	Cocomelon	TV Show	NaN	Jun 1, 2020	105	295
...	...	...	...	...	...	...	...	...	...	...
6674	2022-01-28	5	6	-	Cocomelon	TV Show	NaN	Jun 1, 2020	424	1466
6687	2022-01-29	8	5	8	Cocomelon	TV Show	NaN	Jun 1, 2020	425	1469
6718	2022-02-01	9	-	7	Cocomelon	TV Show	NaN	Jun 1, 2020	426	1471
6959	2022-02-25	10	-	-	Cocomelon	TV Show	NaN	Jun 1, 2020	427	1472
6998	2022-03-01	9	-	-	Cocomelon	TV Show	NaN	Jun 1, 2020	428	1474

328 rows × 10 columns

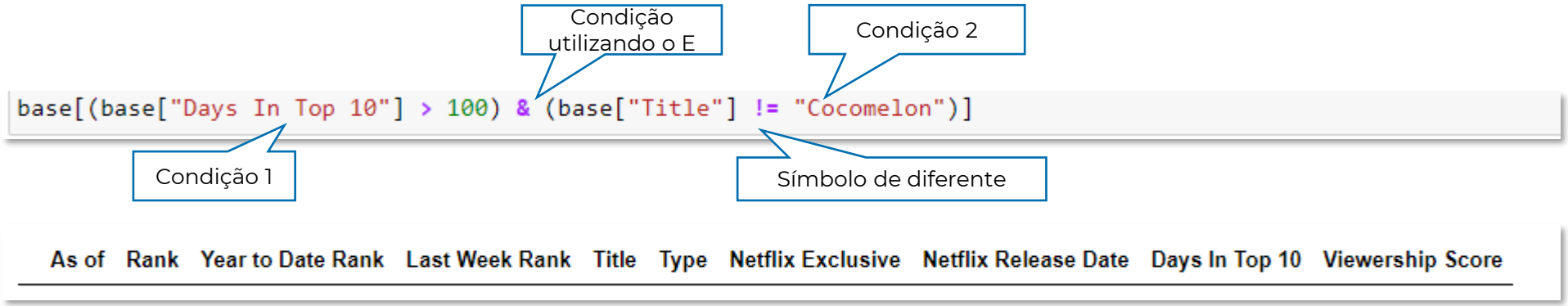
Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Nos filtros é possível colocar mais de uma condição, utilizando o E / OU.

Exemplo: Procurar programas que ficaram mais de 100 dias no Top 10 **e** o título é diferente de Cocomelon.

Estrutura: **base[base**[“**Coluna**”]**condição 1 & condição 2**]

Notamos que apenas o Cocomelon tem o valor maior que 100 na coluna Days in Top 10



Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Exemplo: Procurar o título Squid Game **ou** o título Ozark.

Estrutura: **base[base**[“**Coluna**”]**condição 1 | condição 2**]

Apenas o Ozark apareceu pois o Squid Game está no meio dos intervalos.

Condição 1 | OU | Condição 2

```
base.loc[(base["Title"] == "Squid Game") | (base["Title"] == "Ozark")]
```

	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
1	2020-04-01	2	2	-	Ozark	TV Show	Yes	Jul 21, 2017	5	45
11	2020-04-02	2	2	-	Ozark	TV Show	Yes	Jul 21, 2017	6	54
21	2020-04-03	2	2	-	Ozark	TV Show	Yes	Jul 21, 2017	7	63
31	2020-04-04	2	2	2	Ozark	TV Show	Yes	Jul 21, 2017	8	72
41	2020-04-05	2	2	2	Ozark	TV Show	Yes	Jul 21, 2017	9	81
...	...	...	...	...	...	...	...	...	...	...
6954	2022-02-25	5	5	5	Ozark	TV Show	Yes	Jul 21, 2017	85	524
6967	2022-02-26	8	5	8	Ozark	TV Show	Yes	Jul 21, 2017	86	527
6979	2022-02-27	10	8	7	Ozark	TV Show	Yes	Jul 21, 2017	87	528
6986	2022-02-28	7	10	7	Ozark	TV Show	Yes	Jul 21, 2017	88	532
6996	2022-03-01	7	7	8	Ozark	TV Show	Yes	Jul 21, 2017	89	536

151 rows × 10 columns

Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Uma outra forma de procurarmos uma informação na base é usar o **.loc**

Estrutura: **base.loc**[condição]

Condição 2 | Condição 1

```
base.loc[(base["Title"] == "Squid Game") | (base["Title"] == "Ozark")]
```

	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
1	2020-04-01	2	2	-	Ozark	TV Show	Yes	Jul 21, 2017	5	45
11	2020-04-02	2	2	-	Ozark	TV Show	Yes	Jul 21, 2017	6	54
21	2020-04-03	2	2	-	Ozark	TV Show	Yes	Jul 21, 2017	7	63
31	2020-04-04	2	2	2	Ozark	TV Show	Yes	Jul 21, 2017	8	72
41	2020-04-05	2	2	2	Ozark	TV Show	Yes	Jul 21, 2017	9	81
...	...	...	...	...	...	...	...	...	...	...
6954	2022-02-25	5	5	5	Ozark	TV Show	Yes	Jul 21, 2017	85	524
6967	2022-02-26	8	5	8	Ozark	TV Show	Yes	Jul 21, 2017	86	527
6979	2022-02-27	10	8	7	Ozark	TV Show	Yes	Jul 21, 2017	87	528
6986	2022-02-28	7	10	7	Ozark	TV Show	Yes	Jul 21, 2017	88	532
6996	2022-03-01	7	7	8	Ozark	TV Show	Yes	Jul 21, 2017	89	536

151 rows × 10 columns

Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Uma outra forma de procurarmos informação na base é usar o **.loc**

Estrutura:
base.loc[condição,[coluna]]

```
# Ele permite filtrar colunas de forma muito prática
base.loc[(base["Title"] == "Squid Game") | (base["Title"] == "Ozark"),["As of","Title",'Type']]
```

Condição 1 | Condição 2 | Colunas

	As of	Title	Type
1	2020-04-01	Ozark	TV Show
11	2020-04-02	Ozark	TV Show
21	2020-04-03	Ozark	TV Show
31	2020-04-04	Ozark	TV Show
41	2020-04-05	Ozark	TV Show
...	...	...	...
6954	2022-02-25	Ozark	TV Show
6967	2022-02-26	Ozark	TV Show
6979	2022-02-27	Ozark	TV Show
6986	2022-02-28	Ozark	TV Show
6996	2022-03-01	Ozark	TV Show

151 rows × 3 columns

Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Já o .**iloc** utiliza o índice para filtrar, como se fosse o índice location.

Estrutura: **base.iloc**[linha]

Buscando as linhas de 30 (incluído) e 40 (**não incluído**)

```
base.iloc[30:40]
```

	As of	Rank	Year to Date Rank	Last Week Rank	Title	Type	Netflix Exclusive	Netflix Release Date	Days In Top 10	Viewership Score
30	2020-04-04	1	1	1	Tiger King: Murder, Mayhem ...	TV Show	Yes	Mar 20, 2020	12	120
31	2020-04-04	2	2	2	Ozark	TV Show	Yes	Jul 21, 2017	8	72
32	2020-04-04	3	-	-	Money Heist	TV Show	Yes	Dec 20, 2017	1	8
33	2020-04-04	4	3	3	All American	TV Show	NaN	Mar 28, 2019	12	99
34	2020-04-04	5	-	-	Coffee & Kareem	Movie	Yes	Apr 3, 2020	1	6
35	2020-04-04	6	4	-	How to Fix a Drug Scandal	TV Show	Yes	Apr 1, 2020	3	18
36	2020-04-04	7	6	-	The Roommate	Movie	NaN	Apr 1, 2020	3	13
37	2020-04-04	8	5	-	Nailed It!	TV Show	Yes	Mar 9, 2018	3	16
38	2020-04-04	9	7	-	Unorthodox	TV Show	Yes	Mar 26, 2020	5	14
39	2020-04-04	10	8	-	The Players Club	Movie	NaN	Apr 1, 2020	2	4

Módulo 4 – Introdução ao Pandas: informações estatísticas e filtros na base

Já o .**iloc** utiliza o índice para filtrar, como se fosse o índice location.

Estrutura: **base.iloc**[linha, coluna]

```
# Também posso usar para buscar apenas colunas específicas
base.iloc[30:40,[0,4,5]]
```

Linhas | Colunas

	As of	Title	Type
30	2020-04-04	Tiger King: Murder, Mayhem ...	TV Show
31	2020-04-04	Ozark	TV Show
32	2020-04-04	Money Heist	TV Show
33	2020-04-04	All American	TV Show
34	2020-04-04	Coffee & Kareem	Movie
35	2020-04-04	How to Fix a Drug Scandal	TV Show
36	2020-04-04	The Roommate	Movie
37	2020-04-04	Nailed It!	TV Show
38	2020-04-04	Unorthodox	TV Show
39	2020-04-04	The Players Club	Movie

Módulo 4 – Introdução ao Pandas: criando gráficos

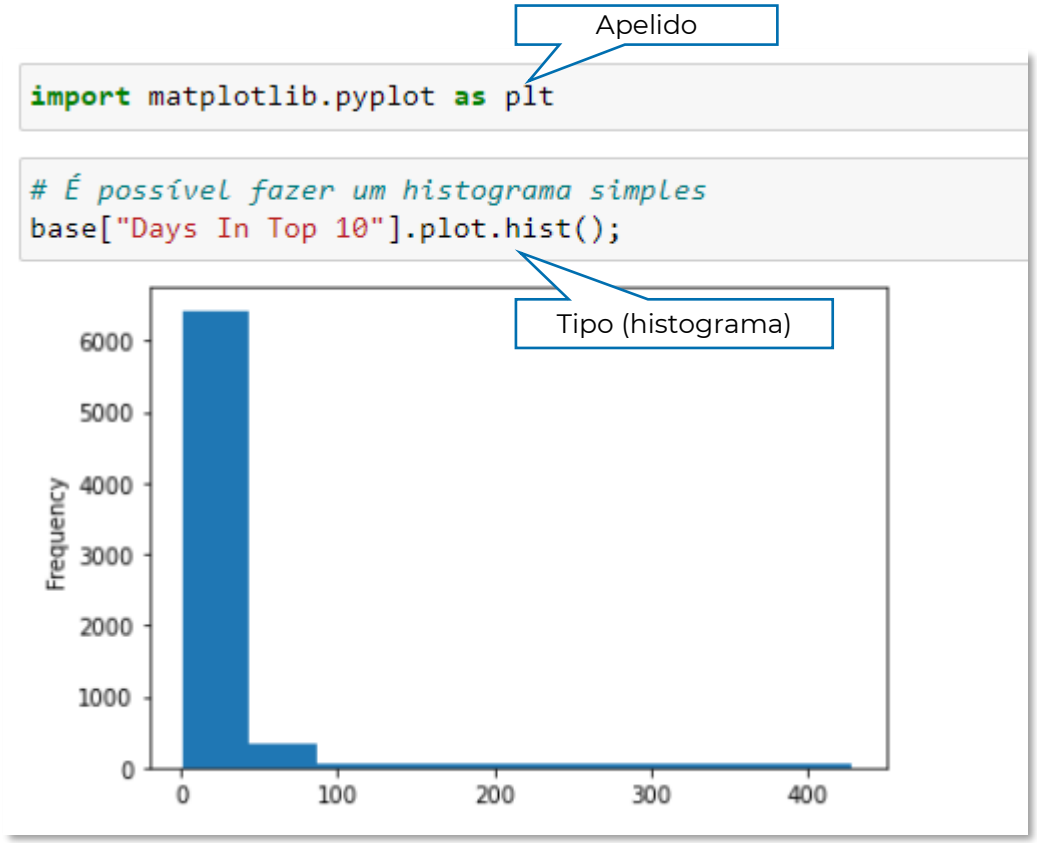
Para criar um gráfico no Pandas é extremamente simples, primeiro vamos precisar importar o **matplotlib.pyplot.**

import matplotlib.pyplot as plt

E depois de uma forma simples plotar um histograma com o código abaixo:

Estrutura: **base**[“Coluna"].**plot.hist()**

Exemplo: **base**["Days In Top 10"].**plot.hist()**



Módulo 4 – Introdução ao Pandas: criando gráficos

Agora, por exemplo, vamos plotar um gráfico de barras de quantas vezes o título aparece.

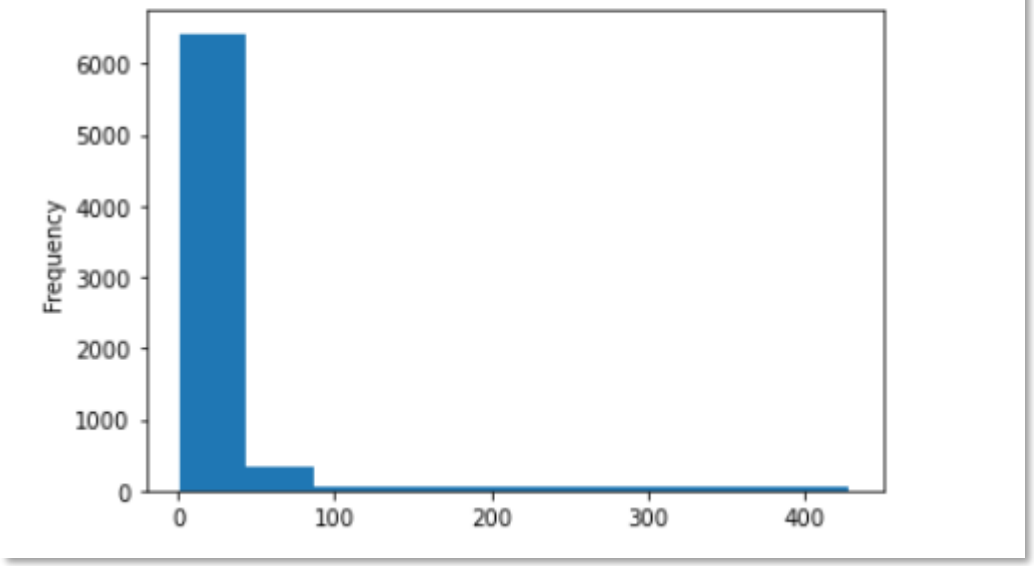
Nesse caso, não é possível fazer direto pois o título não é um valor numérico, para isso vamos fazer o seguinte:

Estrutura: **base**[“Coluna"].**plot.hist()**

Exemplo: **base**["Days In Top 10"].**plot.hist()**

Apelido

```
import matplotlib.pyplot as plt

# É possível fazer um histograma simples
base["Days In Top 10"].plot.hist();
```



MÓDULO 5

Projeto 1 - Analisando o engajamento do Instagram

Módulo 5 – Explicando o Projeto

Agora, vamos aplicar o que aprendemos em um Projeto de Análise do engajamento do Instagram.

O primeiro passo do projeto foi analisar os posts que mais engajam e qual é o melhor conteúdo para sugerir dicas de conteúdos.

Nesse projeto a pergunta que queremos responder é: **Qual tipo de conteúdo mais engaja no Instagram da minha empresa?**

Módulo 5 – Explicando o Projeto

Nesse projeto, vamos considerar que a empresa já possui uma base de dados do Instagram **desde que o usuário começou a postar na marca até o dia 27/março**.

A pessoa que nos contratou enviou alguns direcionamentos:

- Podem ignorar a coluna visualizações, queremos entender apenas **curtidas, comentários e interações.**

- **Tags vazias** é que realmente **não possuem tag** (favor tratar como vazio).

A primeira parte do nosso projeto é importar e visualizar nossa base:

- Importar o Pandas

import pandas **as** pd

- Importar a base em Excel

Estrutura: **base** = **pd.read_tipo('Nome da base.formato')**

base = **pd.read_excel('08. Analisando o engajamento no Instagram.xlsx')**

Vamos importar e visualizar a nossa base

```
# Importando o pandas
import pandas as pd
```

```
# Importar a base em excel
# - Base: 08. Analisando o engajamento no Instagram.xlsx

base = pd.read_excel('08. Analisando o engajamento no Instagram.xlsx')
```

Atenção: o arquivo de Excel tem que estar na mesma pasta do arquivo do jupyter.

Agora o nosso interesse é **ver** e **entender** a nossa base de dados.

Para visualizar as primeiras 5 linhas da base vamos utilizar o comando **head.**

Estrutura: **base.head**()

```
# Visualizando as 5 primeiras linhas
base.head()
```

Desconsiderar visulizações

	Tipo	Data	Curtidas	Comentários	Visualizações	Tags	Pessoas	Campanhas	Carrossel	Interacoes
0	Foto	2021-09-11	2858	16	NaN	Loja	N	N	NaN	2874
1	Foto	2021-09-11	2930	28	NaN	Loja/Produtos	N	N	NaN	2958
2	Foto	2021-09-11	2807	9	NaN	Loja	N	N	NaN	2816
3	Vídeo	2021-09-12	5115	49	82878.0	Produtos	N	N	NaN	5164
4	Foto	2021-09-13	4392	45	NaN	Produtos	S	N	NaN	4437

Valor nulo

Valor nulo

Em projetos muito grandes, quanto mais colunas temos, mais o nosso modelo vai demorar para rodar.

Então, vamos adotar uma boa prática de **apagar as colunas que não vamos usar**. Para isso vamos utilizar o comando **drop**:

Estrutura: **base**.**drop**(**'nome_coluna'**, **axis = 1**)

- **axis = 1** para apagar coluna
- **axis = 0** para apagar linha

Em caso de mais de 1 coluna, passamos a lista entre colchetes.

```
# Apagando a coluna "Visualizações"
base = base.drop("Visualizações",axis=1)
```

Atribuir a base novamente para sobrepor as mudanças

Módulo 5 – Importando e tratando a base com Pandas

```
# Visualizando novamente as 5 primeiras linhas
base.head()
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
0	Foto	2021-09-11	2858	16	Loja	N	N	NaN	2874
1	Foto	2021-09-11	2930	28	Loja/Produtos	N	N	NaN	2958
2	Foto	2021-09-11	2807	9	Loja	N	N	NaN	2816
3	Vídeo	2021-09-12	5115	49	Produtos	N	N	NaN	5164
4	Foto	2021-09-13	4392	45	Produtos	S	N	NaN	4437

```
# Visualizando as 5 últimas linhas
base.tail()
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
47	IGTV	2022-03-12	5489	77	Dicas de como usar/Novos Produtos	S	N	NaN	5566
48	Foto	2022-03-20	29084	479	Datas comemorativas/Promoções	S	S	NaN	29563
49	Foto	2022-03-22	9087	106	NaN	S	S	NaN	9193
50	Foto	2022-03-26	16551	186	NaN	S	N	NaN	16737
51	IGTV	2022-03-27	4934	65	Dicas de como usar/Produtos	S	N	NaN	4999

Para visualizar novamente as primeiras 5 linhas da base vamos utilizar o comando **head.**

Estrutura: **base.head**()

E para visualizar as últimas 5 linhas vamos utilizar o comando **tail**.

Estrutura: **base.tail**()

Módulo 5 – Importando e tratando a base com Pandas

E vamos continuar visualizando a base com o passo a passo básico.

- Tamanho

Estrutura: **base.shape**()

Outra forma de fazer isso é utilizando o comando display.

Estrutura: **display**(**base**)

Se a base for muito **pequena**, o display vai mostrar a **base inteira**.

```
# Tamanho da base
base.shape
```

```
(52, 9)
```

Linhas

Colunas

Para saber as informações (tipo, valores nulos) da base vamos utilizar o comando **info**.

Estrutura: **base.info**()

```
# Verificando as informações
base.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 52 entries, 0 to 51
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Tipo         52 non-null     object
 1   Data         52 non-null     datetime64[ns]
 2   Curtidas     52 non-null     int64
 3   Comentários  52 non-null     int64
 4   Tags         44 non-null     object
 5   Pessoas      52 non-null     object
 6   Campanhas    52 non-null     object
 7   Carrossel    8 non-null      object
 8   Interacoes   52 non-null     int64
dtypes: datetime64[ns](1), int64(3), object(5)
memory usage: 3.8+ KB
```

8 valores não nulos e 44 valores nulos

Módulo 5 – Tratando valores nulos da coluna Corrossel

Agora vamos tratar os valores nulos que aparecem na coluna Carrossel.

Porém, primeiro precisamos entender os valores de carrossel, para isso vamos contar os valores que aparecem nessa coluna.

Estrutura: **base**.**Coluna**.**value_counts**()

Estrutura: **base**.**Carrossel**.**value_counts**()

Só existe o valor Sim (S)

```
# Contando os valores que aparecem na coluna Carrossel
base.Carrossel.value_counts()
```

```
S    8
Name: Carrossel, dtype: int64
```

Na verdade, os valores **nulos** são de postagens que **não são carrossel**.

Sendo assim o nulo deveria ser "N“.

Módulo 5 – Tratando valores nulos da coluna Corrossel

E como que a gente poderia **atribuir um novo valor**?

Para localizar os valores que são nulos vamos utilizar o comando loc.

Estrutura: **base.loc[base.Coluna.isnull()]**

Estrutura: **base.loc[base.Carrossel.isnull()]**

```
# Filtrando os valores em que carrossel é nulo
base.loc[base.Carrossel.isnull()].head()
```

Para poder resumir (5 primeiras linhas)

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
0	Foto	2021-09-11	2858	16	Loja	N	N	NaN	2874
1	Foto	2021-09-11	2930	28	Loja/Produtos	N	N	NaN	2958
2	Foto	2021-09-11	2807	9	Loja	N	N	NaN	2816
3	Vídeo	2021-09-12	5115	49	Produtos	N	N	NaN	5164
4	Foto	2021-09-13	4392	45	Produtos	S	N	NaN	4437

Valor **não**

Módulo 5 – Tratando valores nulos da coluna Corrossel

Para localizar os valores que não são nulos vamos utilizar o comando loc.

Estrutura: **base.loc[base.Coluna.notnull()]**

Estrutura:**base.loc[base.Carrossel.notnull()]**

```
# Buscando valores que NAO sao nulos
base.loc[base.Carrossel.notnull()]
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
5	Foto	2021-09-17	5359	62	Novos Produtos	N	S	S	5421
8	Foto	2021-09-27	6355	89	Produtos	S	N	S	6444
12	Foto	2021-10-21	6166	55	Novos Produtos	S	S	S	6221
21	Foto	2021-12-23	8328	93	Produtos	S	N	S	8421
25	Foto	2022-01-02	12193	138	Novos Produtos	S	N	S	12331
26	Foto	2022-01-08	24585	354	Datas comemorativas	S	S	S	24939
28	Foto	2022-01-15	9936	119	Novos Produtos	S	N	S	10055
40	Foto	2022-02-21	21621	213	Influenciadores	S	S	S	21834

Valor **sim**

Módulo 5 – Tratando valores nulos da coluna Corrossel

Para localizar os valores que são nulos apenas na coluna Carrossel.

Estrutura:

base.loc[base.Coluna.isnull(), 'Coluna']

Estrutura:

base.loc[base.Carrossel.isnull(), 'Carrossel']

```
# Selecionando apenas a coluna Carrossel
base.loc[base.Carrossel.isnull(),'Carrossel']
```

```
0     NaN
1     NaN
2     NaN
3     NaN
4     NaN
6     NaN
7     NaN
9     NaN
10    NaN
11    NaN
13    NaN
14    NaN
15    NaN
16    NaN
17    NaN
18    NaN
19    NaN
20    NaN
22    NaN
23    NaN
24    NaN
27    NaN
29    NaN
30    NaN
31    NaN
32    NaN
33    NaN
34    NaN
35    NaN
36    NaN
37    NaN
38    NaN
39    NaN
41    NaN
42    NaN
43    NaN
44    NaN
45    NaN
46    NaN
47    NaN
48    NaN
49    NaN
50    NaN
51    NaN
Name: Carrossel, dtype: object
```

Módulo 5 – Tratando valores nulos da coluna Corrossel

Agora vamos atribuir o valor N para essa coluna.

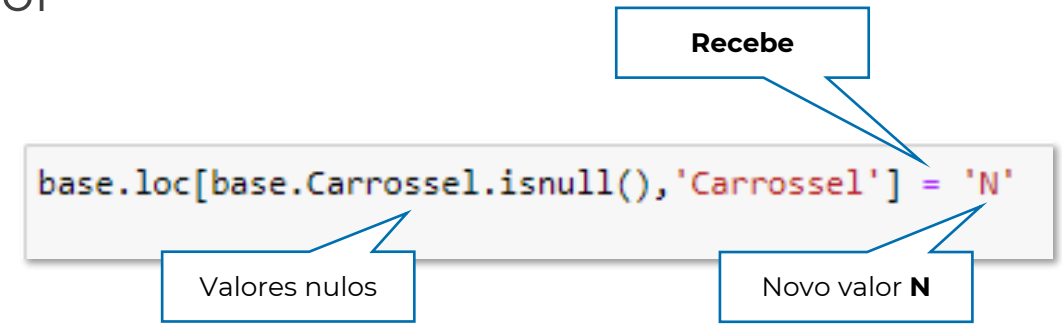
Lembrando que para atribuir um novo valor utilizamos o sinal de **igual**.

Estrutura:

base.loc[base.Coluna.isnull(), 'Coluna'] = 'N'

Estrutura:

base.loc[base.Carrossel.isnull(), 'Carrossel'] = 'N'



Módulo 5 – Tratando valores nulos da coluna Corrossel

Verificando novamente os valores dessa coluna, utilizando o comando **head**.

```
base.head()
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
0	Foto	2021-09-11	2858	16	Loja	N	N	N	2874
1	Foto	2021-09-11	2930	28	Loja/Produtos	N	N	N	2958
2	Foto	2021-09-11	2807	9	Loja	N	N	N	2816
3	Vídeo	2021-09-12	5115	49	Produtos	N	N	N	5164
4	Foto	2021-09-13	4392	45	Produtos	S	N	N	4437

Valor **N**

Verificando novamente os valores dessa coluna, podemos observar que não existe mais valor nulo.

```
base.Carrossel.value_counts()
```

```
N    44
S     8
Name: Carrossel, dtype: int64
```

Módulo 5 – Analisando informações estatísticas e as 5 melhores / 5 piores publicações

Agora, vamos nos aprofundar na nossa base utilizando o **describe**, que nos fornecerá todo o resumo estatístico.

Estrutura: **base**.**describe**()

É importante entender o motivo dos menores/maiores valores, para saber se o máximo é algo **replicável** e **escalável**.

```
# Descrição estatística da base
base.describe()
```

	Curtidas	Comentários	Interacoes
count	52.000000	52.000000	52.000000
mean	12262.730769	189.500000	12452.230769
std	8165.875326	170.687709	8299.390088
min	2807.000000	9.000000	2816.000000
25%	5492.000000	69.500000	5562.500000
50%	9603.000000	128.000000	9773.500000
75%	17621.750000	265.250000	17920.750000
max	37351.000000	852.000000	37853.000000

Valor **mínimo**

Valor **máximo**

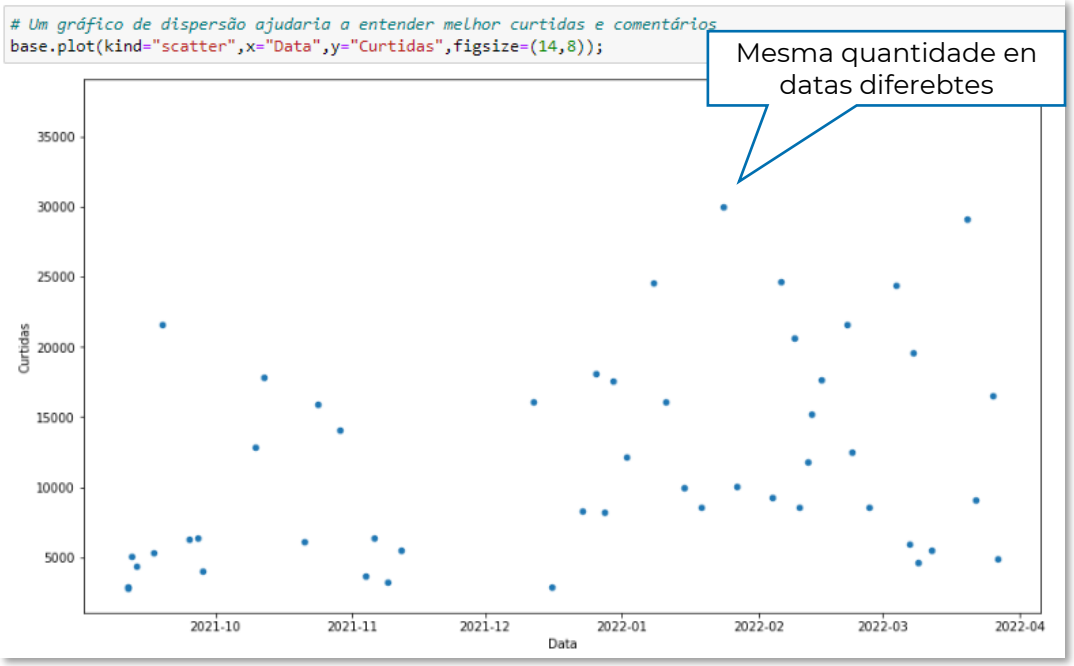
Módulo 5 – Analisando informações estatísticas e as 5 melhores / 5 piores publicações

Podemos também plotar as informações pelo gráfico scatter.

Estrutura:
base.plot(kind="scatter",x="Data",y**="Curtidas"**,**figsize**=(14,8));

base.plot(kind="tipo",x="Data",y**="Curtidas"**, **figsize**=(14,8));

Observa-se que no tempo não tem um aumento de curtidas, então a data não é um fator tão predominante que influencia nas curtidas.

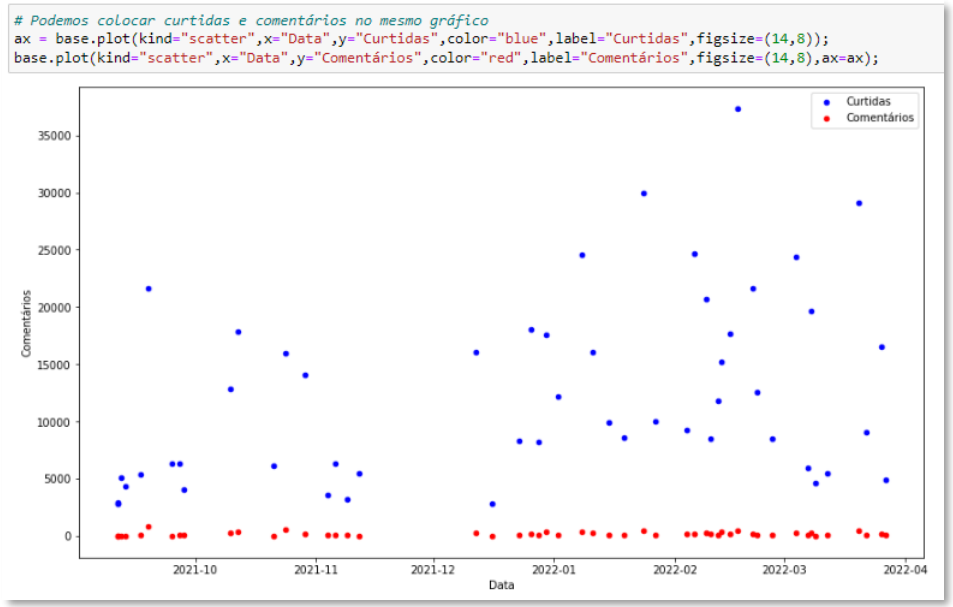


Módulo 5 – Analisando informações estatísticas e as 5 melhores / 5 piores publicações

Nesse gráfico não ficou bom de visualizar curtidas e comentários, porque a escala da curtida não nos permite analisar os comentários.

Estrutura:
Ax=**base**.**plot**(kind="scatter",x="Data",y="**Curtidas**",color="blue",label="Curtidas",figsize=(14,8));
base.**plot**(kind="scatter",x="Data",y="**Comentários**",color="red",label="Comentários",figsize=(14,8),ax=ax);

```
# Podemos colocar curtidas e comentários no mesmo gráfico
ax = base.plot(kind="scatter",x="Data",y="Curtidas",color="blue",label="Curtidas",figsize=(14,8));
base.plot(kind="scatter",x="Data",y="Comentários",color="red",label="Comentários",figsize=(14,8),ax=ax);
```

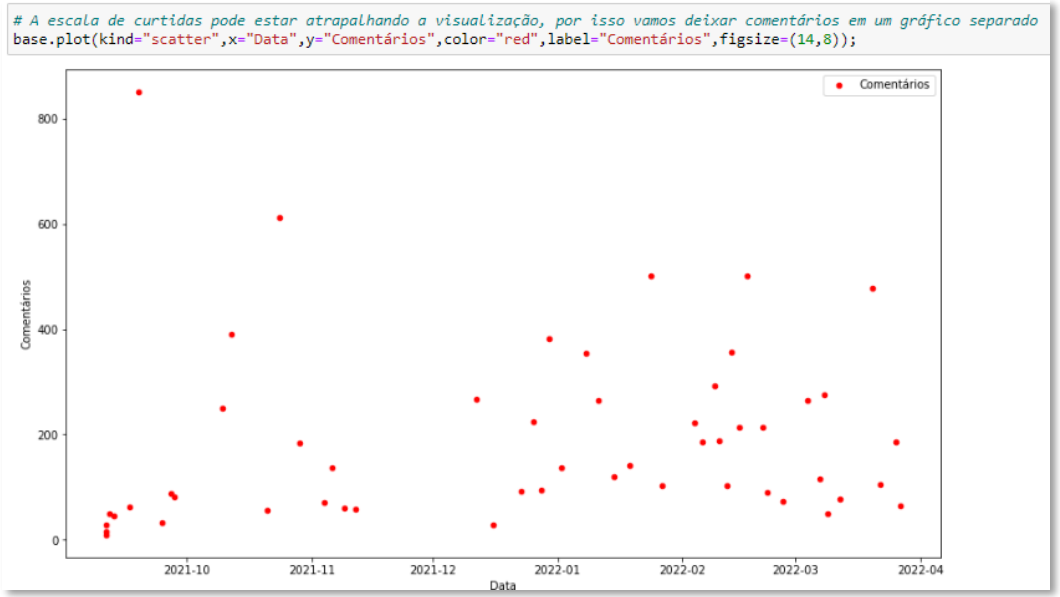


Módulo 5 – Analisando informações estatísticas e as 5 melhores / 5 piores publicações

Criando apenas um gráfico de comentário também observamos que não existe nenhum padrão.

O gráfico e as informações estatísticas não estão dizendo muita coisa pois existe uma grande dispersão entre curtidas e comentários.

Precisamos verificar se existe um padrão usando as outras colunas de informações.

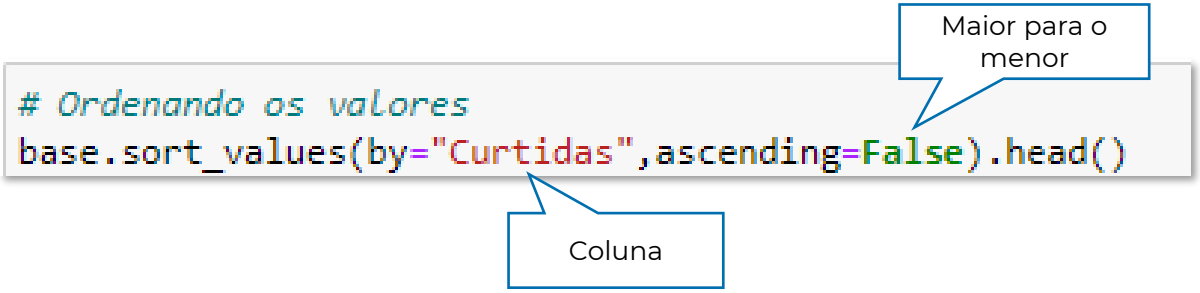


Módulo 5 – Analisando informações estatísticas e as 5 melhores / 5 piores publicações

A primeira coisa que podemos fazer é pegar os **5 primeiros** registros com **mais** e **menos curtidas.**

Para isso vamos utilizar a função **sort_values**, que vai ordenar os meus valores por uma coluna escolhida.

Podendo escolher se vamos ordenar de forma ascendente (menor para maior) ou descendente (maior para o menor).



Módulo 5 – Analisando informações estatísticas e as 5 melhores / 5 piores publicações

A primeira coisa que podemos fazer é pegar os **5 primeiros** registros com **mais curtidas.**

Estrutura: **base**.**sort_values**(by="**Curtidas**",ascending=**False**).**head()**

Estrutura: **base**.**sort_values**(by="**Curtidas**",ascending=**False**).**head()**

```
# Ordenando os valores
base.sort_values(by="Curtidas",ascending=False).head()
```

Tem pessoas e são publicações de campanha

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
39	Foto	2022-02-17	37351	502	Promoções	S	S	N	37853
30	Reels	2022-01-24	29981	502	Trends	S	S	N	30483
48	Foto	2022-03-20	29084	479	Datas comemorativas/Promoções	S	S	N	29563
33	Foto	2022-02-06	24655	186	Influenciadores	S	S	N	24841
26	Foto	2022-01-08	24585	354	Datas comemorativas	S	S	S	24939

Módulo 5 – Analisando informações estatísticas e as 5 melhores / 5 piores publicações

Podemos utilizar o mesmo comando para encontrar os **5** registros com **menos curtidas.**

Estrutura: **base**.**sort_values**(by="**Curtidas**",ascending=**True**).**head()**

Estrutura: **base**.**sort_values**(by="**Curtidas**",ascending= **True**).**head()**

```
# Selecionando os 5 últimos valores
base.sort_values(by="Curtidas",ascending=True).head()
```

Não tem pessoas e **não** são publicações de campanha

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
2	Foto	2021-09-11	2807	9	Loja	N	N	N	2816
0	Foto	2021-09-11	2858	16	Loja	N	N	N	2874
20	Foto	2021-12-16	2881	29	Produtos	N	N	N	2910
1	Foto	2021-09-11	2930	28	Loja/Produtos	N	N	N	2958
17	Vídeo	2021-11-09	3213	60	Produtos	N	N	N	3273

Módulo 5 – Analisando informações estatísticas e as 5 melhores / 5 piores publicações

Uma outra forma de encontrar os registros com **menos curtidas** é utilizando o **tail**.

Estrutura: **base**.**sort_values**(by="**Curtidas**",ascending=**False**).**tail()**

Estrutura: **base**.**sort_values**(by="**Curtidas**",ascending= **False**).**tail()**

```
# Selecionando os 5 últimos valores
base.sort_values(by="Curtidas",ascending=False).tail()
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
17	Vídeo	2021-11-09	3213	60	Produtos	N	N	N	3273
1	Foto	2021-09-11	2930	28	Loja/Produtos	N	N	N	2958
20	Foto	2021-12-16	2881	29	Produtos	N	N	N	2910
0	Foto	2021-09-11	2858	16	Loja	N	N	N	2874
2	Foto	2021-09-11	2807	9	Loja	N	N	N	2816

Módulo 5 – Analisando informações estatísticas e as 5 melhores / 5 piores publicações

Então, podemos observar que no **top 5** todas as postagens tinham **pessoas** e eram fotos de **campanha**.

Nas 5 piores postagens, não haviam pessoas e nem eram postagens de campanhas.

Isso pode ser um indicador que **pessoas** e **campanhas** têm relação com as **curtidas**.

O próximo passo foi marcar todas as outras fotos para saber se tinha pessoa ou era foto de campanha, para saber se isso que observamos se aplicava ao restante da base.

Agora vamos ver como nós vamos comparar as fotos que têm pessoas e as que não têm, vamos utilizar de forma muito semelhante ao que fazemos no Excel com a tabela dinâmica.

Basicamente, o que fizemos foi comprovar a hipótese de que fotos **com pessoas** possuem **mais curtidas**.

Rótulos de Linha	Média de Curtidas
N	4.257
S	14.665
Total Geral	**12.263**

Média até 3x maior

Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Agora vamos ver como fazemos isso no Python.

O **group by** é muito análogo à tabela dinâmica que fazemos no Excel.

O groupby vai agregar pela **Coluna1**

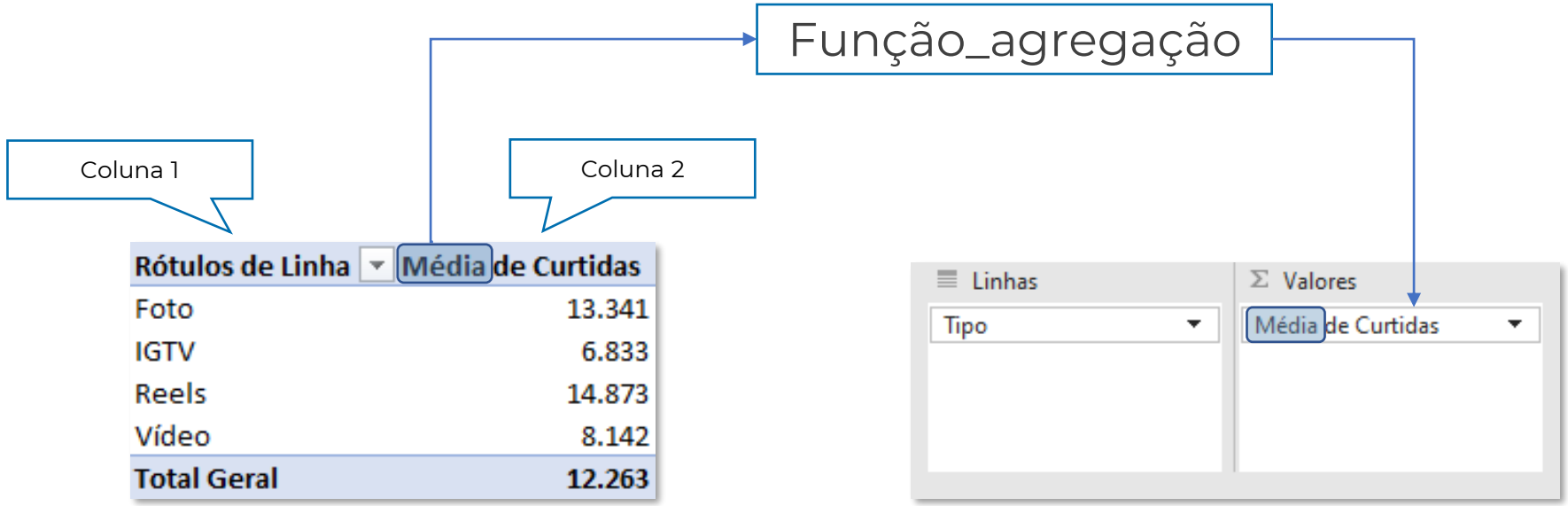
A função de agregação será aplicada na **Coluna2**

```
# Agrupando as informações por tipo
base.groupby("Tipo")["Comentários"].count()
```

```
Tipo
Foto     36
IGTV      5
Reels     5
Vídeo     6
Name: Comentários, dtype: int64
```

Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Estrutura: **base**.**groupby**("**Coluna1**")["**Coluna2**"].**função_agregação()**



Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Aplicando no nosso exemplo temos:

base.groupby("Tipo")["Comentários"].count()

```
# Agrupando as informações por tipo
base.groupby("Tipo")["Comentários"].count()
```

```
Tipo
Foto     36
IGTV      5
Reels     5
Vídeo     6
Name: Comentários, dtype: int64
```

Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Nós podemos adicionar mais colunas para visualizar por tipo e pessoas, por exemplo.

base.groupby(["Tipo“, “Pessoas“])["Curtidas"]. mean()

```
# Agrupando por Tipo e Pessoas
base.groupby(["Tipo","Pessoas"])["C
```

```
Tipo   Pessoas
Foto   N          3,863.50
       S         15,236.67
IGTV   S          6,833.40
Reels  N          5,934.50
       S         20,832.00
Vídeo  N          4,007.50
       S         16,409.50
Name: Curtidas, dtype: float64
```

```
base.groupby(["Tipo","Pessoas"])[["Curtidas",'Comentários']].mean()
```

		Curtidas	Comentários
Tipo	Pessoas		
Foto	N	3,863.50	29.50
	S	15,236.67	226.20
IGTV	S	6,833.40	133.60
Reels	N	5,934.50	98.00
	S	20,832.00	342.00
Vídeo	N	4,007.50	65.25
	S	16,409.50	370.00

Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Podemos também analisar Curtidas e Comentários, ao mesmo tempo.

base.groupby(["Tipo“, “Pessoas“])[["Curtidas“,’Comentários’]]. mean()

```
base.groupby(["Tipo","Pessoas"])[["Curtidas",'Comentários']].mean()
```

Tipo	Pessoas	Curtidas	Comentários
Foto	N	3,863.50	29.50
	S	15,236.67	226.20
IGTV	S	6,833.40	133.60
Reels	N	5,934.50	98.00
	S	20,832.00	342.00
Vídeo	N	4,007.50	65.25
	S	16,409.50	370.00

Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Agora, se a gente quiser incluir a coluna de campanhas basta:

base.groupby(["Tipo“, “Pessoas“,”Campanhas”])[["Curtidas“,’Comentários’]]. mean()

```
# Incluindo a coluna de campanhas
base.groupby(["Tipo","Pessoas","Campanhas"])[["Curtidas",'Comentários']].mean()
```

			Curtidas	Comentários
Tipo	Pessoas	Campanhas		
Foto	N	N	2,869.00	20.50
		S	5,852.50	47.50
	S	N	10,815.29	159.93
		S	19,105.38	284.19
IGTV	S	N	6,833.40	133.60
Reels	N	N	5,934.50	98.00
	S	N	12,894.00	249.00
		S	24,801.00	388.50
Vídeo	N	N	4,007.50	65.25
	S	S	16,409.50	370.00

O groupby já permite ver que publicações de campanha tem um grande engajamento e com foto de pessoas também.

Podemos então fazer os agrupamentos que acharmos melhor para entender os nossos dados.

Agrupamento somente para pessoas:
base.groupby(“Pessoas“)[["Curtidas“,’Comentários’]]. mean()

```
# Somente para pessoas
base.groupby("Pessoas")[["Curtidas",'Comentários']].mean()
```

Pessoas	Curtidas	Comentários
N	4,256.67	52.83
S	14,664.55	230.50

3 vezes maior com pessoas

Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Agrupamento somente para campanhas:
base.**groupby**(“**Campanhas**“)[["**Curtidas“,’Comentários’**]]. **mean()**

```
# Somente para campanhas
base.groupby("Campanhas")[["Curtidas",'Comentários']].mean()
```

Campanhas	Curtidas	Comentários
N	7,928.33	123.17
S	18,173.27	279.95

Postagem de campanha tem mais curtidas

Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Nesse caso devemos **filtrar** apenas as fotos pois só temos carrossel em fotos.

Sem esse filtro estaríamos comparando coisas erradas.

Então temos que **restringir a nossa base**.

base[**base.**Tipo=='Foto'].**groupby**("**Carrossel**")[["**Curtidas**",'**Comentários**']].**mean**()

```
# Podemos também filtrar a base
base[base.Tipo=='Foto'].groupby("Carrossel")[["Curtidas",'Comentários']].mean()
```

Filtro

Carrossel	Curtidas	Comentários
N	13,776.36	208.57
S	11,817.88	140.38

Fotos que não são carrosséis tem mais curtidas

Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Filtrar a base é muito importante, senão podemos dar uma conclusão errada, porque comparamos dados que não podem ser comparados.

```
# Carrossel (sem filtrar a base)
base.groupby("Carrossel")[["Curtidas",'Comentários']].mean()
```

Carrossel	Curtidas	Comentários
N	12,343.61	198.43
S	11,817.88	140.38

Diferença de 500

```
# Podemos também filtrar a base
base[base.Tipo=='Foto'].groupby("Carrossel")[["Curtidas",'Comentários']].mean()
```

Carrossel	Curtidas	Comentários
N	13,776.36	208.57
S	11,817.88	140.38

Diferença de 2000

Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Colocando pessoas e campanhas juntos podemos ver como se dá essa diferença.

A média quando tem **pessoas** **E** é publicação de **campanhas** é de cerca de 19,4 mil curtidas, já quando são apenas pessoas (sem campanha) passa para quase 10 mil e se não tiverem pessoas chega no máximo a 5,9 mil mesmo em campanhas.

Nesse caso a gente já consegue mostrar para a empresa a **importância** de incluir **pessoas** usando os seus produtos, o que gera um aumento considerável no engajamento.

```
# Agregando por pessoas e campanhas
base.groupby(["Pessoas","Campanhas"])[["Curtidas",'Comentários']].mean()
```

Pessoas	Campanhas	Curtidas	Comentários
N	N	3,937.50	53.90
	S	5,852.50	47.50
S	N	9,923.75	157.80
	S	19,405.35	303.20

Nem Pessoas, nem campanhas: 3937 curtidas

Pessoas e campanhas: 19405 curtidas

Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Podemos também analisar por pessoas, campanhas e tipo.

Analisando novamente a questão do vídeo, ele não parece mais tão ruim assim.

Quando feito em **campanha** e usando **pessoas** ele teve um resultado **bom**, inclusive próximo à foto.

```
# Agregando por pessoas, campanhas e tipo
base.groupby(["Pessoas","Campanhas","Tipo"])[["Curtidas",'Comentários']].mean()
```

Pessoas	Campanhas	Tipo	Curtidas	Comentários
N	N	Foto	2,869.00	20.50
		Reels	5,934.50	98.00
		Vídeo	4,007.50	65.25
	S	Foto	5,852.50	47.50
S	N	Foto	10,815.29	159.93
		IGTV	6,833.40	133.60
		Reels	12,894.00	249.00
	S	Foto	19,105.38	284.19
		Reels	24,801.00	388.50
		Vídeo	16,409.50	370.00

Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

O que poderia ter levado a média baixa é que só temos vídeo ou **COM** pessoa e **COM** campanha ou sem nenhum dos dois.

Não temos nenhum vídeo com apenas um dos dois (pessoa ou campanha).

Já IGTV, mesmo tendo pessoa, não teve um resultado tão bom.

```
# Agregando por pessoas, campanhas e tipo
base.groupby(["Pessoas","Campanhas","Tipo"])[["Curtidas",'Comentários']].mean()
```

Pessoas	Campanhas	Tipo	Curtidas	Comentários
N	N	Foto	2,869.00	20.50
		Reels	5,934.50	98.00
		Vídeo	4,007.50	65.25
	S	Foto	5,852.50	47.50
S	N	Foto	10,815.29	159.93
		IGTV	6,833.40	133.60
		Reels	12,894.00	249.00
	S	Foto	19,105.38	284.19
		Reels	24,801.00	388.50
		Vídeo	16,409.50	370.00

Módulo 5 – O group by (groupby) no pandas e a análise do engajamento

Se eu filtrar a base somente por tipo igual a vídeo, percebemos que a loja tentou postar 4 vídeos mostrando seus produtos (sem **nenhuma pessoa**) e o resultado foi **baixo**.

Quando o vídeo foi feito com **pessoas** aproveitando trends e **datas comemorativas** o resultado foi **muito bom**!

```
# Vamos filtrar a base apenas onde o tipo é Vídeo
base[base.Tipo=='Vídeo']
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
3	Vídeo	2021-09-12	5115	49	Produtos	N	N	N	5164
9	Vídeo	2021-09-28	4056	81	Produtos	N	N	N	4137
15	Vídeo	2021-11-04	3646	71	Produtos	N	N	N	3717
17	Vídeo	2021-11-09	3213	60	Produtos	N	N	N	3273
24	Vídeo	2021-12-30	17600	383	Trends	S	S	N	17983
37	Vídeo	2022-02-13	15219	357	Datas comemorativas	S	S	N	15576

Resultado bom com pessoas

Conclusões:

- Em uma análise inicial, postagens incluindo **pessoas engajam** muito mais que aquelas que não possuem ninguém.
- Postagens em **épocas de campanha** também possuem um **melhor engajamento**.
- Nessa base, o **carrossel não foi um diferencial** para melhorar o engajamento da marca.

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

As tags foram criadas para a base não ficar muito grande, um exemplo de tag é o gênero de um filme.

Agora o que queremos responder é: **Qual a tag mais engaja nessas publicações?**

Agora queremos olhar apenas **tags**.

A pessoa que nos contratou também deu alguns direcionamentos:

- Podem ignorar a coluna visualizações, queremos entender apenas curtidas, comentários e interações.

- Tags vazias significa que realmente não possuem tag (tratar como vazio).

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Primeiro vamos importar a nossa base exatamente como fizemos anteriormente:

- Importar o Pandas

import pandas **as** pd

- Importar o Numpy

import numpy **as** np

```
# Importando o pandas
import pandas as pd
import numpy as np
# Usando o mesmo formato dos valores
pd.options.display.float_format = '{:,.2f}'.format
```

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

- Importar a base excel

base = pd.read_excel("08. Analisando o engajamento no Instagram.xlsx")

```
# Importar a base em excel
base = pd.read_excel("08. Analisando o engajamento no Instagram.xlsx")
```

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

- Excluir a coluna Visualizações

base = **base**.**drop**("**Visualizações**",axis=1)

```
# Apagando a coluna "Visualizações"
base = base.drop("Visualizações",axis=1)
```

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

```
# Visualizando novamente as 5 primeiras linhas
base.head()
```

Mais de uma tag

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
0	Foto	2021-09-11	2858	16	Loja	N	N	NaN	2874
1	Foto	2021-09-11	2930	28	Loja/Produtos	N	N	NaN	2958
2	Foto	2021-09-11	2807	9	Loja	N	N	NaN	2816
3	Vídeo	2021-09-12	5115	49	Produtos	N	N	NaN	5164
4	Foto	2021-09-13	4392	45	Produtos	S	N	NaN	4437

Depois vamos visualizar as 5 primeiras linhas utilizando o comando head.

base.head()

Se tem mais uma tag separamos por uma barra.

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Vamos agora aplicar o group by para Tags:

base.**groupby**("**Tags**")["**Curtidas**"].mean()

Repare que Datas comemorativas aparecem em mais de uma tag.

Ou seja, se eu tenho **mais de uma tag** não é possível agrupar.

Então precisamos tratar a nossa base para depois utilizar o group by.

```
# Agrupando por tags
base.groupby("Tags")["Curtidas"].mean()
```

```
Tags
Datas comemorativas                 17,975.00
Datas comemorativas/Promoções       29,084.00
Dicas de como usar/Novos Produtos    5,703.50
Dicas de como usar/Produtos          7,586.67
Influenciadores                     15,197.29
Loja                                 2,832.50
Loja/Produtos                        2,930.00
Novos Produtos                      11,619.57
Produtos                             5,666.92
Promoções                           26,645.50
Trends                              22,400.67
Trends/Produtos                     12,894.00
Name: Curtidas, dtype: float64
```

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Para conseguir analisar separadamente as tags, podemos dividir linhas com 2 tags em 2 linhas.

- Para isso primeiro vamos usar o **split** para **separar** em uma lista com as tags.
- Depois vamos usar o **explode** para transformar as listas com 2 tags em 2 **linhas diferentes**.

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

A primeira coisa que o split vai fazer é separar as informações.

O split separa um texto em uma lista baseado em algum separador, se eu informamos nenhum argumento ele vai separar por espaço.

Estrutura: **texto**.**split**() ou **texto**.**split**('delimitador')

```
texto = "O Curso de Ciência de Dados da Hashtag é top!"

# Se eu não passo nenhum argumento, ele vai separar por espaço
texto.split()
```

```
['O', 'Curso', 'de', 'Ciência', 'de', 'Dados', 'da', 'Hashtag', 'é', 'top!']
```

```
texto = "O-Curso-de-Ciência-de-Dados-da-Hashtag-é-top!"

texto.split()
```

Separar pelo **espaço**

```
['O-Curso-de-Ciência-de-Dados-da-Hashtag-é-top!']

texto.split("-")
```

Separar pelo **-**

```
['O', 'Curso', 'de', 'Ciência', 'de', 'Dados', 'da', 'Hashtag', 'é', 'top!']
```

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Agora vamos utilizar esse comando para a nossa coluna de Tags, transformando a coluna Tags em uma lista de tags e separando as tags por “/”.

base.Tags = **base**.Tags.**str.split**("/")

Para visualizar as primeiras 5 linhas vamos utilizar o comando **head**.

base.head()

```
# Vamos usar isso para a nossa coluna "Tags"
# Transformando a coluna Tags em uma lista de tags
base.Tags = base.Tags.str.split("/")
base.head()
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
0	Foto	2021-09-11	2858	16	[Loja]	N	N	NaN	2874
1	Foto	2021-09-11	2930	28	[Loja, Produtos]	N	N	NaN	2958
2	Foto	2021-09-11	2807	9	[Loja]	N	N	NaN	2816
3	Vídeo	2021-09-12	5115	49	[Produtos]	N	N	NaN	5164
4	Foto	2021-09-13	4392	45	[Produtos]	S	N	NaN	4437

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

O **explode** vai separar uma coluna de um DataFrame em **1 linha para cada elemento da lista**.

Observe que na primeira o elemento A tem 2 elementos.

O **explode** vai transformar a minha primeira linha em **2 linhas**, uma com o primeiro elemento e outra com o segundo.

```
# Criando o dicionários
dic = {
    "A": [[1,2],3,[4,5,6],[]],
    "B": [1,2,3,4],
}

# Transformando esse dicionário em um DataFrame
base_dic = pd.DataFrame(dic)

base_dic
```

2 elementos

3 elementos

	A	B
0	[1, 2]	1
1	3	2
2	[4, 5, 6]	3
3	[]	4

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Estrutura: **base_dic** = **base_dic**.**explode**(‘**Coluna**’)

Aplicando ao nosso exemplo, temos: **base_dic** = **base_dic**.**explode**('**A**’)

```
# Criando o dicionários
dic = {
    "A": [[1,2],3,[4,5,6],[]],
    "B": [1,2,3,4],
}

# Transformando esse dicionário em um DataFrame
base_dic = pd.DataFrame(dic)

base_dic
```

	A	B
0	[1, 2]	1
1	3	2
2	[4, 5, 6]	3
3	[]	4

2 elementos

3 elementos

```
# Usando o explode para separar a coluna A
base_dic = base_dic.explode('A')
base_dic
```

	A	B
0	1	1
0	2	1
1	3	2
2	4	3
2	5	3
2	6	3
3	NaN	4

Cada elemento em uma linha

Elemento é mantido

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Tudo que estiver em lista será separado em 1 linha por elemento da lista.

Se não tiver na lista, o elemento será mantido.

Listas vazias vão ter o valor de NaN.

Para as outras colunas, elas irão repetir os seus valores.

Inclusive o índice também irá repetir.

```
# Usando o explode para separar a coluna A
base_dic = base_dic.explode('A')
base_dic
```

	A	B
0	1	1
0	2	1
1	3	2
2	4	3
2	5	3
2	6	3
3	NaN	4

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Vamos aplicar o explode na nossa coluna de Tag agora que ele já está separado.

base = **base**.**explode**('**Tags**')
base.**head**()

Feito isso agora podemos começar a trabalhar com as nossas Tags.

```
# Separando a coluna Tag em 1 linha para cada elemento da lista
base = base.explode('Tags')
base.head()
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
0	Foto	2021-09-11	2858	16	Loja	N	N	NaN	2874
1	Foto	2021-09-11	2930	28	Loja	N	N	NaN	2958
1	Foto	2021-09-11	2930	28	Produtos	N	N	NaN	2958
2	Foto	2021-09-11	2807	9	Loja	N	N	NaN	2816
3	Vídeo	2021-09-12	5115	49	Produtos	N	N	NaN	5164

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

```
# Repetindo o cálculo da média para pessoas
base.groupby("Pessoas")["Curtidas"].mean()
```

```
Pessoas
N    4,154.62
S   14,100.57
Name: Curtidas, dtype: float64
```

```
base.head()
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
0	Foto	2021-09-11	2858	16	Loja	N	N	NaN	2874
1	Foto	2021-09-11	2930	28	Loja	N	N	NaN	2958
1	Foto	2021-09-11	2930	28	Produtos	N	N	NaN	2958
2	Foto	2021-09-11	2807	9	Loja	N	N	NaN	2816
3	Vídeo	2021-09-12	5115	49	Produtos	N	N	NaN	5164

Valores duplicados

Agora vamos fazer o mesmo groupby que fizemos na aula anterior para o cálculo de média para pessoas.

Repare que temos uma diferença do que vimos na última aula.

Como adicionamos linha, toda análise que estamos fazendo só pode ser feita para essa coluna nova de Tag.

Por exemplo, se eu tinha uma foto com 5000 curtidas e 2 tags, agora ela vai aparecer **2 vezes**.

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Fazendo a mesma análise para Tags, repare que agora com a base tratada temos apenas uma Tag por linha.

base.**groupby**("**Tags**")["**Curtidas**"].mean()

```
# Fazendo para Tag
base.groupby("Tags")["Curtidas"].mean()
```

```
Tags
Datas comemorativas   20,752.25
Dicas de como usar     6,833.40
Influenciadores       15,197.29
Loja                   2,865.00
Novos Produtos        10,304.89
Produtos               6,269.82
Promoções             27,458.33
Trends                20,024.00
Name: Curtidas, dtype: float64
```

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Para melhorar a visualização podemos colocar em ordem decrescente os valores de curtidas. Para isso, vamos utilizar o comando **sort_values**.

sort_values("**Coluna**",ascending=False)

False – maior para o menor
True – menor para o maior

Ordenando a nossa base por curtida temos:

base.**groupby**("**Tags**")[["**Curtidas**","**Comentários**"]
].mean().**sort_values**("**Curtidas**",ascending=False)

```
# Ordenando por curtidas
base.groupby("Tags")[["Curtidas","Comentários"]].mean().sort_values("Curtidas",ascending=False)
```

Tags	Curtidas	Comentários
Promoções	27,458.33	531.00
Datas comemorativas	20,752.25	343.50
Trends	20,024.00	352.25
Influenciadores	15,197.29	161.71
Novos Produtos	10,304.89	198.56
Dicas de como usar	6,833.40	133.60
Produtos	6,269.82	94.12
Loja	2,865.00	17.67

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Dessa forma, é possível notar que postagens de **promoções** são as que mais **engajam**.

Além de promoções, datas comemorativas e trends também possuem um bom engajamento.

```
# Ordenando por curtidas
base.groupby("Tags")[["Curtidas","Comentários"]].mean().sort_values("Curtidas",ascending=False)
```

Tags	Curtidas	Comentários
Promoções	27,458.33	531.00
Datas comemorativas	20,752.25	343.50
Trends	20,024.00	352.25
Influenciadores	15,197.29	161.71
Novos Produtos	10,304.89	198.56
Dicas de como usar	6,833.40	133.60
Produtos	6,269.82	94.12
Loja	2,865.00	17.67

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

E o que está sem tag? Como vamos tratar isso?

O primeiro passo é identificar os valores sem Tag utilizando o filtro.

Estrutura: **base**[**base**.**Tags**.**isnull**()]

```
# Filtrando valores sem tag
base[base.Tags.isnull()]
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
11	Foto	2021-10-12	17831	391	NaN	S	S	NaN	18222
19	Foto	2021-12-12	16086	268	NaN	S	S	NaN	16354
29	Foto	2022-01-19	8612	142	NaN	S	N	NaN	8754
38	Foto	2022-02-15	17687	213	NaN	S	N	NaN	17900
41	Foto	2022-02-22	12530	90	NaN	S	N	NaN	12620
43	Foto	2022-03-04	24399	266	NaN	S	S	NaN	24665
49	Foto	2022-03-22	9087	106	NaN	S	S	NaN	9193
50	Foto	2022-03-26	16551	186	NaN	S	N	NaN	16737

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Da mesma forma que fizemos para Carrossel, podemos fazer para as tags escrevendo "Sem tag" e nesse caso iria aparecer no groupby.

base.**loc**[**base**.**Tags**.**isnull**(),'**Tags**'] = 'Sem Tags'

```
# Atribuindo o texto sem tag para as colunas onde a tag é NaN
base.loc[base.Tags.isnull(),'Tags'] = 'Sem Tags'
```

Valor atribuído

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Agora vamos repetir a tabela de curtidas por Tag considerando valores sem tag.

Observe que a categoria Sem Tags aparece na quarta posição. Uma interpretação possível é que podem ser fotos muito específicas ou quando faço algo muito genérico e tem um engajamento bom.

```
# Mostrando novamente a tabela de curtidas por tag
base.groupby("Tags")[["Curtidas","Comentários"]].mean().sort_values("Curtidas",ascending=False)
```

Tags	Curtidas	Comentários
Promoções	27,458.33	531.00
Datas comemorativas	20,752.25	343.50
Trends	20,024.00	352.25
Sem Tags	15,347.88	207.75
Influenciadores	15,197.29	161.71
Novos Produtos	10,304.89	198.56
Dicas de como usar	6,833.40	133.60
Produtos	6,269.82	94.12
Loja	2,865.00	17.67

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Podemos voltar como NaN caso a gente queira somente ignorar esses valores conforme orientado.

Podemos utilizar o Numpy:

import numpy **as** np
base.**loc**[**base**.**Tags** == 'Sem Tags','**Tags**'] = np.nan

E essas linhas novamente param de ser consideradas na agregação.

```
# Podemos voltar como NaN caso a gente queira somente ignorar esses valores conforme orientado
import numpy as np
base.loc[base.Tags == 'Sem Tags','Tags'] = np.nan

# E voltamos com as colunas com valores nulos
base[base.Tags.isnull()]
```

Transforma em valor nulo

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
11	Foto	2021-10-12	17831	391	NaN	S	S	NaN	18222
19	Foto	2021-12-12	16086	268	NaN	S	S	NaN	16354
29	Foto	2022-01-19	8612	142	NaN	S	N	NaN	8754
38	Foto	2022-02-15	17687	213	NaN	S	N	NaN	17900
41	Foto	2022-02-22	12530	90	NaN	S	N	NaN	12620
43	Foto	2022-03-04	24399	266	NaN	S	S	NaN	24665
49	Foto	2022-03-22	9087	106	NaN	S	S	NaN	9193
50	Foto	2022-03-26	16551	186	NaN	S	N	NaN	16737

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Agora que a gente já tem a nossa base de Tag, podemos relacionar pessoas com Tag:

base.groupby(['Pessoas',"Tags"])[["Curtidas","Comentários"]].mean()

Podemos concluir que todas as tags com pessoas têm um valor bom.

Comparando Novos Produtos com pessoas e sem pessoas, é possível observar que ele funciona muito melhor com pessoas.

```
# Fazendo para Pessoas e Tag
base.groupby(['Pessoas',"Tags"])[["Curtidas","Comentários"]].mean()
```

Pessoas	Tags	Curtidas	Comentários
N	Loja	2,865.00	17.67
	Novos Produtos	5,359.00	62.00
	Produtos	4,450.67	60.78
S	Datas comemorativas	20,752.25	343.50
	Dicas de como usar	6,833.40	133.60
	Influenciadores	15,197.29	161.71
	Novos Produtos	10,923.12	215.62
	Produtos	8,316.38	131.62
	Promoções	27,458.33	531.00
	Trends	20,024.00	352.25

Sem pessoas

Com pessoas

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Se a gente quiser ordenar por curtidas podemos utilizar o sort_values.

base.**groupby**(['**Pessoas**',"**Tags**"])[["**Curtidas**","**Comentários**"]].mean().**sort_values**("**Curtidas**",ascending=False)

Observe que tudo o que tem pessoas tem uma boa quantidade de curtidas.

```
# Também podemos ordenar por curtidas
base.groupby(['Pessoas',"Tags"])[["Curtidas","Comentários"]].mean().sort_values("Curtidas",ascending=False)
```

		Curtidas	Comentários
Pessoas	Tags		
S	Promoções	27,458.33	531.00
	Datas comemorativas	20,752.25	343.50
	Trends	20,024.00	352.25
	Influenciadores	15,197.29	161.71
	Novos Produtos	10,923.12	215.62
	Produtos	8,316.38	131.62
	Dicas de como usar	6,833.40	133.60
N	Novos Produtos	5,359.00	62.00
	Produtos	4,450.67	60.78
	Loja	2,865.00	17.67

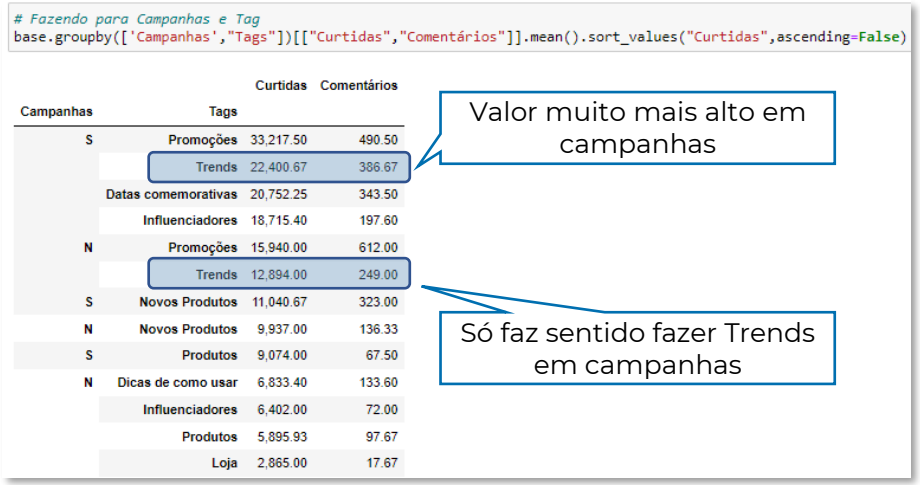
Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Podemos aplicar o mesmo para Campanhas e Tags.

base.**groupby**(['**Campanhas**',"**Tags**"])[["**Curtidas**","**Comentários**"]].mean().**sort_values**("**Curtidas**", ascending=False)

Observe que algumas vezes, mesmo não tendo campanha, promoção funciona.

Campanhas não é tão determinante assim em algumas Tags.



```
# Fazendo para Campanhas e Tag
base.groupby(['Campanhas',"Tags"])[["Curtidas","Comentários"]].mean().sort_values("Curtidas",ascending=False)
```

Campanhas	Tags	Curtidas	Comentários
S	Promoções	33,217.50	490.50
	Trends	22,400.67	386.67
	Datas comemorativas	20,752.25	343.50
	Influenciadores	18,715.40	197.60
N	Promoções	15,940.00	612.00
	Trends	12,894.00	249.00
S	Novos Produtos	11,040.67	323.00
N	Novos Produtos	9,937.00	136.33
S	Produtos	9,074.00	67.50
N	Dicas de como usar	6,833.40	133.60
	Influenciadores	6,402.00	72.00
	Produtos	5,895.93	97.67
	Loja	2,865.00	17.67

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

Com tudo isso que observamos, podemos fechar a nosso análise com algumas conclusões:

- Ter o **rosto de outras pessoas é fundamental para um bom engajamento** na publicação.
    - Em todas as tags, quando havia o rosto, o resultado foi muito melhor.
- Criar **campanhas ajuda muito na divulgação** da marca.
- **Promoções** tiveram um desempenho absurdamente **maior que qualquer outra tag**.
    - Porém é uma tag que pode ter **custo para a loja**, o que deve ser analisado.

Módulo 5 – Analisando Tags: Separando valores de uma coluna em linhas diferentes (split e explode)

- Usar conteúdos que estão em **trend** também ajudam na divulgação da marca, **mesmo a trend sendo de outros nichos** .
- A melhor maneira de **mostrar produtos** é através de outras **pessoas utilizando-os**, e se possível em campanhas de datas especiais.
- Para **novos produtos** a inclusão de **pessoas** é mais crítica ainda, sendo quase o dobro quando há um rosto junto ao produto.
- **Não** podemos afirmar que a **tag Loja é ruim** até testarmos essa tag incluindo pessoas ou em uma campanha. Vale o teste para verificar.
- Continuaremos a monitorar as postagens para encontrar novos padrões, dado que ainda temos poucas informações da base.

MÓDULO 6

Introdução a Estatística

O que é estatística?

Descrição

```
base.describe()
```

	altura
count	8.000000
mean	1.728750
std	0.058661
min	1.650000
25%	1.695000
50%	1.720000
75%	1.755000
max	1.840000

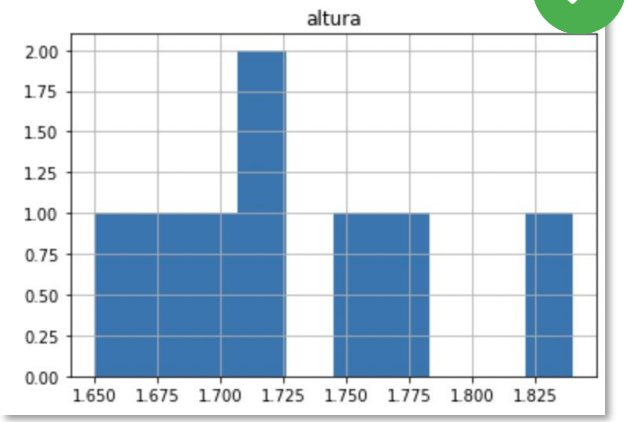
Tabela de Frequência

```
pd.crosstab(index=base.altura,columns='count')
```

col_0	count
altura	
1.65	1
1.68	1
1.70	1
1.72	2
1.75	1
1.77	1
1.84	1

Histograma

```
base.hist();
```



Amostragem

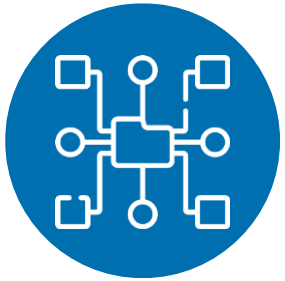
```
base.sample(3)
```

	alunos	altura
2	Jean	1.70
6	Vanessa	1.77
0	Lucas	1.65

Podemos dizer que tudo isso é estatística, desde a parte de coleta até a análise de dados.

COLETA

ORGANIZAÇÃO



APRESENTAÇÃO

INTERPRETAÇÃO

ANÁLISE

Na estatística também podemos fazer:

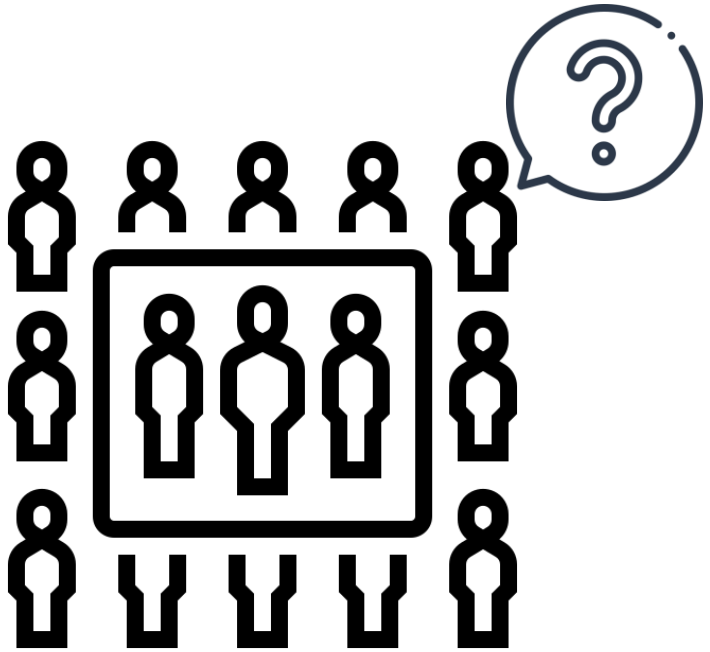
- Inferências
- Amostragem: para obter uma confiança da amostra que estamos trabalhando.
- Machine learning: utiliza estatística para fazer previsões.

INFERÊNCIAS

AMOSTRAGEM

MACHINE LEARNING

Com a estatística nós vamos representar todas as informações que estão em 1000 linhas através de **indicadores** (média, mediana, tabela de frequência), de forma que seja **prático** trabalhar com essas informações.



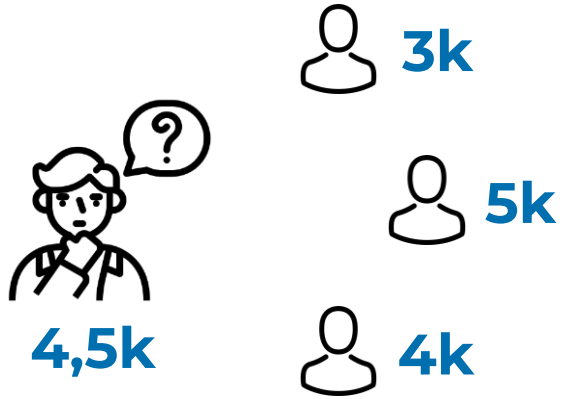
Estatística descritiva é a **descrição** e **apresentação** dos dados de forma a facilitar o entendimento de um grande conjunto de dados

Fornece um resumo sobre os dados apresentados, podendo ser **visual** (tabelas de frequência, histogramas, gráficos) ou **quantitativo** (media, mediana, moda)

Vamos analisar um exemplo: supondo que você recebeu uma promoção e seu salário a partir de agora é de 4,5K.

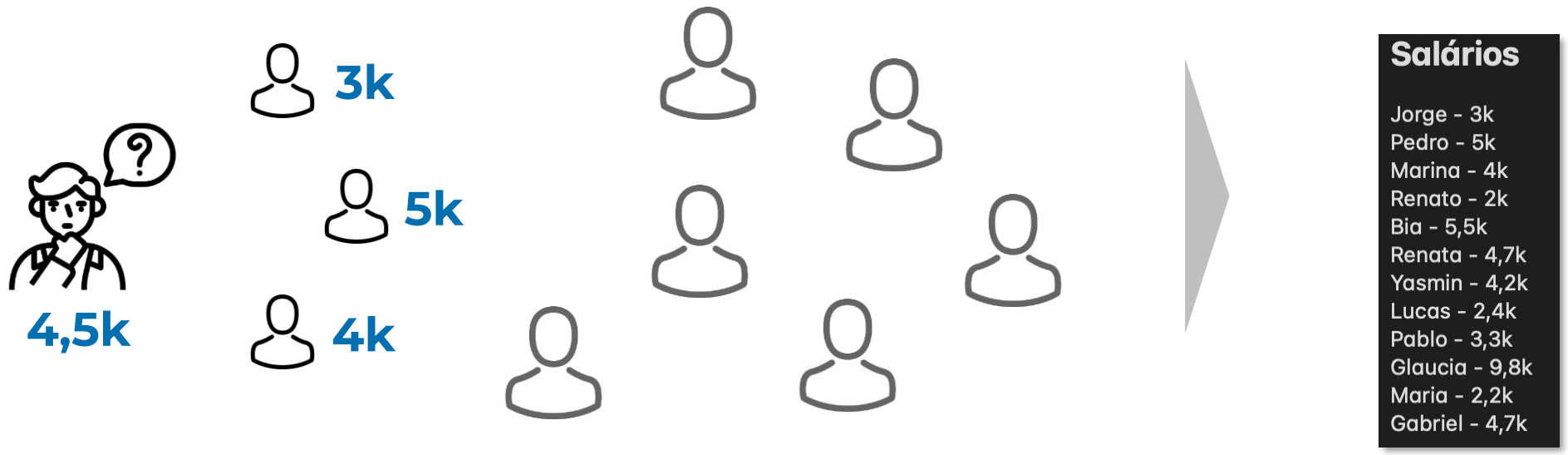
E aí surgem os questionamentos: **É um salário bom? Faz sentido esse salário?**

Perguntando a 3 amigos, você percebe que o seu salário é um valor bom.



Você decide continuar perguntando e tem uma lista maior de dados. Como você vai comparar o seu salário agora?

Fica **inviável** fazer comparações listando todos os dados.



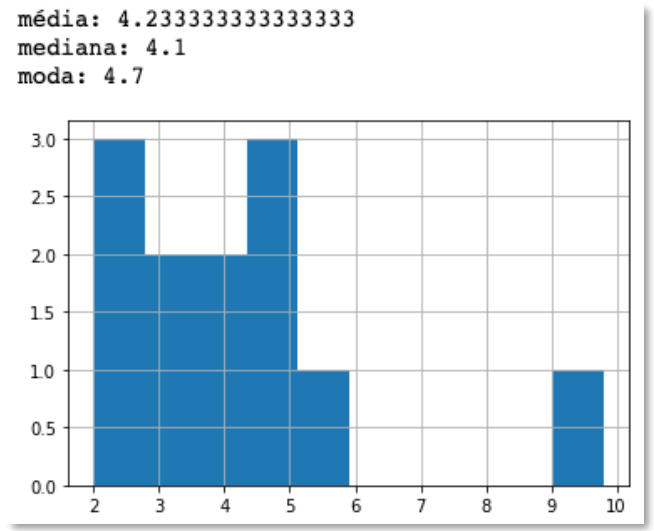
Quando transformamos a nossa tabela em um histograma e colocamos indicadores (média, mediana e moda), podemos concluir que o seu salário está acima da média.

A **estatística** fornece com poucos indicadores **informações** muito importantes sobre a base inteira.

4,5k

Salários

Jorge - 3k
Pedro - 5k
Marina - 4k
Renato - 2k
Bia - 5,5k
Renata - 4,7k
Yasmin - 4,2k
Lucas - 2,4k
Pablo - 3,3k
Glaucia - 9,8k
Maria - 2,2k
Gabriel - 4,7k



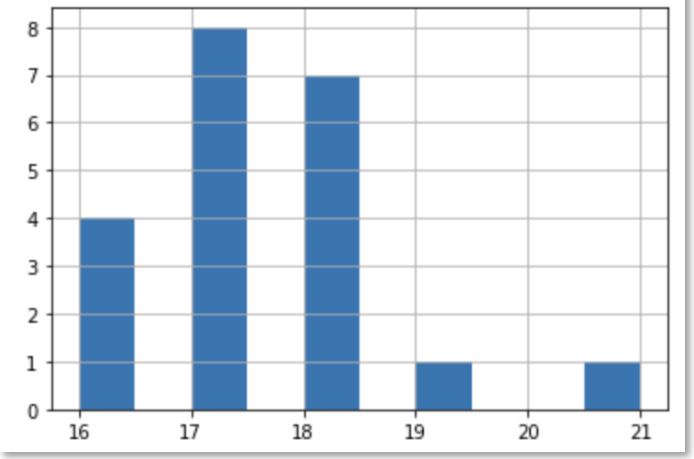
Módulo 6 – Tabela de frequência e histograma

Agora que a gente já entendeu que a estatística é utilizada para explicar dados sobre uma base e mostrar de forma gráfica ou quantitativa informações sobre uma base, vamos mostrar como podemos fazer isso utilizando a **tabela de frequência** e o **histograma**.

```
base = {
    "id_aluno": [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21],
    "idades": [16,16,16,16,17,17,17,17,17,17,17,17,18,18,18,18,18,18,18,19,21]
}

base = pd.DataFrame(base)

print(pd.crosstab(index=base.idades,columns="contagem"))
```

```
col_0    contagem
idades
16              4
17              8
18              7
19              1
21              1
```

```
base_alunos.idades.hist();
```



Módulo 6 – Tabela de frequência e histograma

A tabela de frequência apresenta os dados e a **quantidade de ocorrência** / **repetições daquele dado**.

Vamos supor que você tem a idade dos alunos e deseja informar quantos alunos possuem 17 anos.

Olhando para a lista você não consegue dizer rapidamente.

Agora, criando um tabela de frequência, listando a idade e o total de alunos, já temos muita informação sobre a base.

Difícil visualizar informação

```
base = {
    "id_aluno": [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21],
    "idades": [16,16,16,16,17,17,17,17,17,17,17,17,18,18,18,18,18,18,18,19,21]
}

base = pd.DataFrame(base)

print(pd.crosstab(index=base.idades,columns="contagem"))
```

```
col_0    contagem
idades
16              4
17              8
18              7
19              1
21              1
```

Informação resumida

ALUNOS	ALUNOS
16	4
17	8
18	7
19	1
21	1

Vejamos um outro exemplo: em 2018, cerca de 30% dos funcionários de uma determinada empresa recebiam um salário de 4000 a 5500.

Em 2022, esse número diminuiu, mas em compensação o número de pessoas com salário acima de 4000 aumentou.

Podemos concluir que a empresa está aumentando os salários.

FAIXA SALARIAL	FREQUÊNCIA		FREQUÊNCIA ACUMULADA	
	2018	2022	2018	2022
1.000 a 2.500	5%	3%	5%	3%
2.500 a 4.000	21%	18%	26%	21%
4.000 a 5.500	38%	25%	64%	46%
5.500 a 7.000	14%	25%	78%	71%
7.000 a 8.500	12%	17%	90%	88%
8.500 a 10.000	7%	9%	97%	97%
acima 10.000	3%	3%	100%	100%

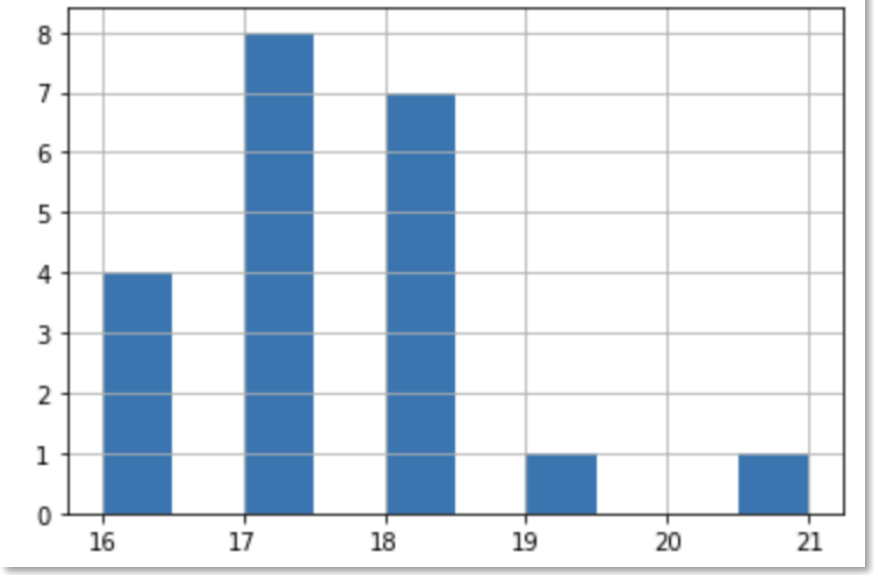
Módulo 6 – Tabela de frequência e histograma

O **histograma** é basicamente a **exibição gráfica** de uma distribuição de frequências.

Pelo histograma ao lado podemos observar a distribuição de alunos.

Podemos escolher o histograma ou a tabela de frequência para representar distribuição, escolher com base na análise que estamos criando.

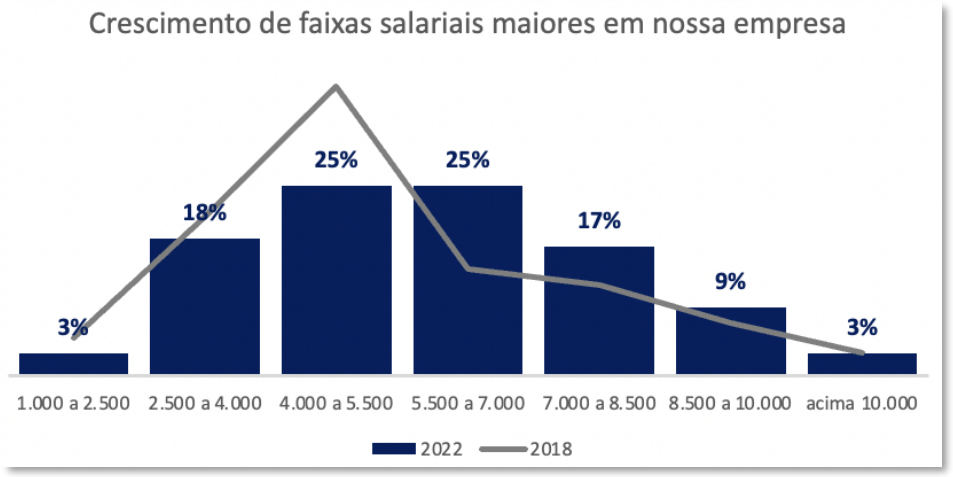
```
base_alunos.idades.hist();
```



Módulo 6 – Tabela de frequência e histograma

Podemos ver que tanto a tabela de frequência quanto o histograma passam a mensagem que os salários estão melhorando.

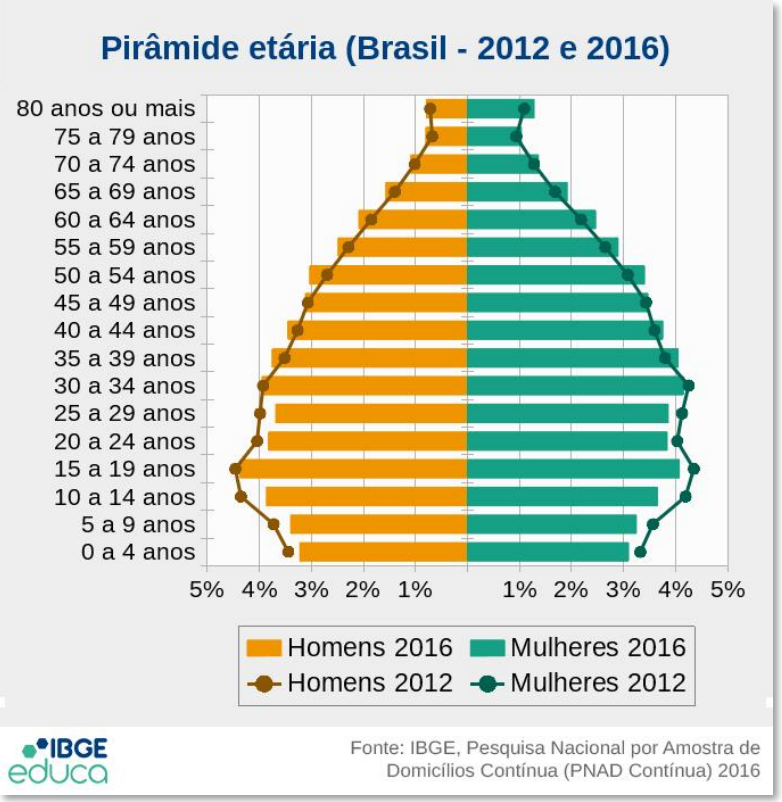
FAIXA SALARIAL	FREQUÊNCIA	
	2018	2022
1.000 a 2.500	5%	3%
2.500 a 4.000	21%	18%
4.000 a 5.500	38%	25%
5.500 a 7.000	14%	25%
7.000 a 8.500	12%	17%
8.500 a 10.000	7%	9%
acima 10.000	3%	3%



Utilizamos o histograma também para mostrar a população.

Por exemplo, no histograma de pirâmide etária é possível notar que em 2012 tínhamos mais pessoas mais novas, agora o numero de pessoas mais novas caiu.

Pode ser que poucas pessoas estão nascendo ou menos pessoas querendo ter filhos.



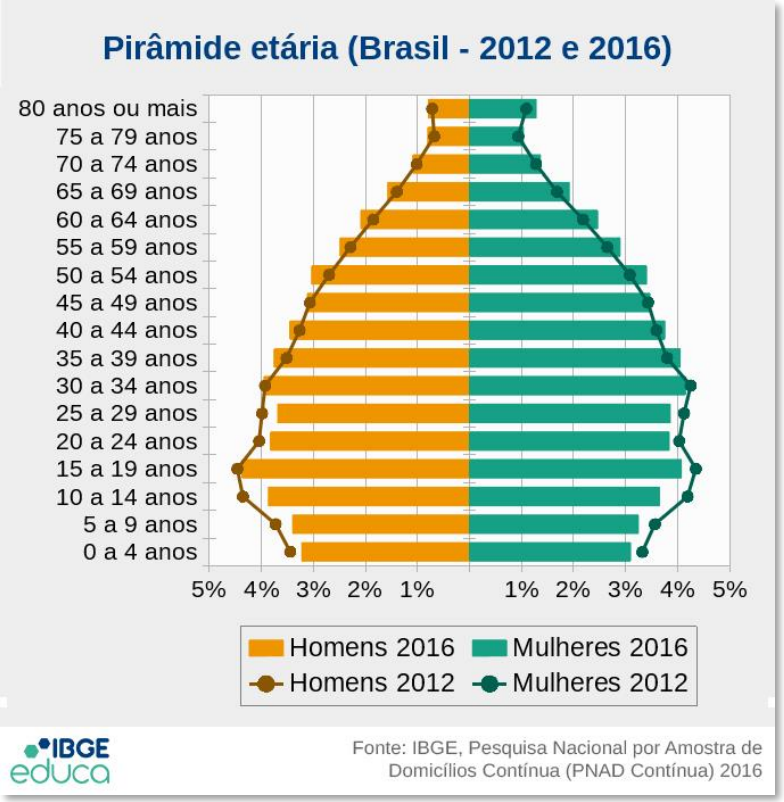
Módulo 6 – Tabela de frequência e histograma

Em compensação as idades maiores estão aumentando ao longo dos anos.

Então a expectativa de vida está maior e as pessoas estão vivendo mais tempo.

Podemos usar essa informação para entender a questão de aposentadoria.

Se temos menos pessoas contribuindo e mais pessoas estão ficando mais velhas, precisamos entender melhor como vamos fazer um plano de aposentadoria que consiga atender a todos.

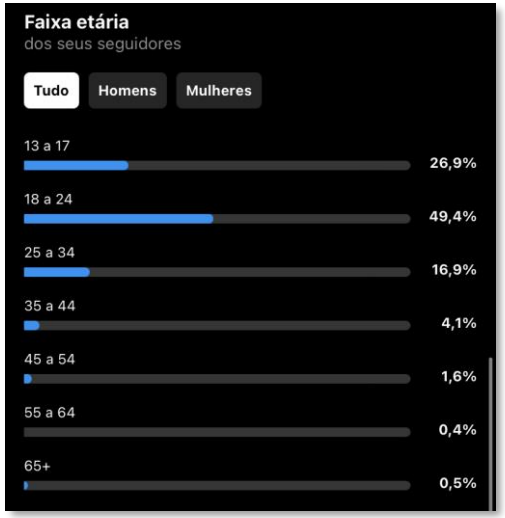


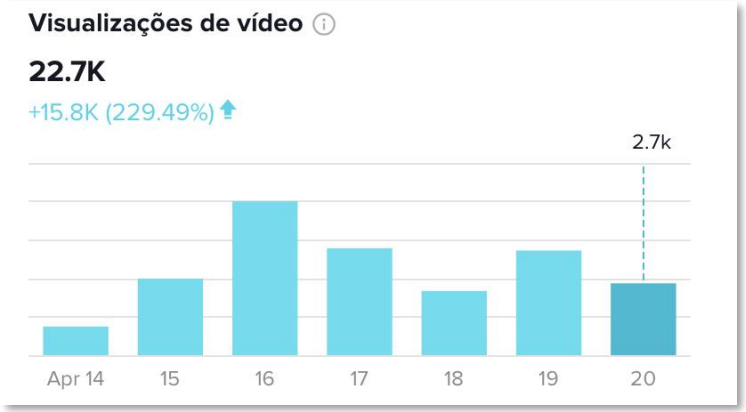
Módulo 6 – Tabela de frequência e histograma

Podemos utilizar informações estatísticas para rede social para aumentar o engajamento.

Podemos visualizar a faixa etária dos seguidores, períodos mais ativos, melhor dia e horário para postar.

Todas essas informações ajudam a aumentar o engajamento.





Módulo 6 – Entendendo o conceito de média

Agora vamos entender como mostrar informações estatísticas de forma quantitativa.

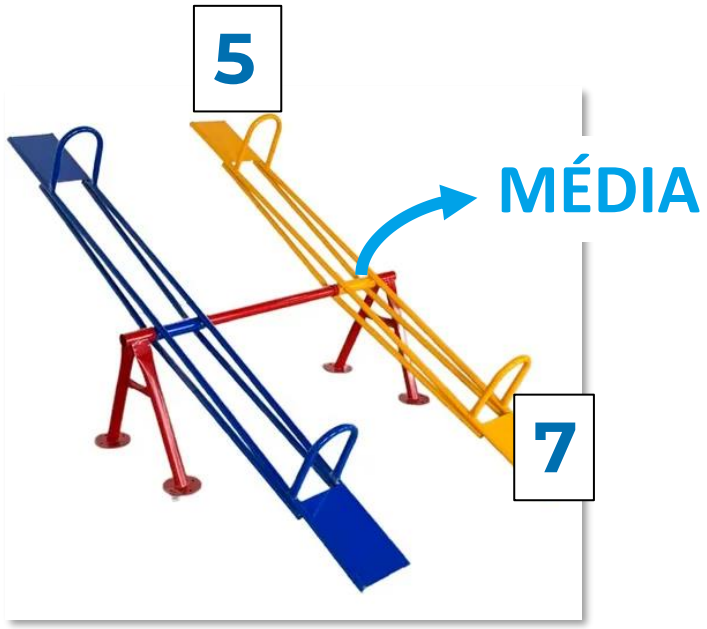
Resumo estatístico é uma forma de apresentar os principais conceitos sobre a distribuição dos dados de forma resumida, usando para isso:

- Média
- Mediana
- Moda
- Desvio Padrão

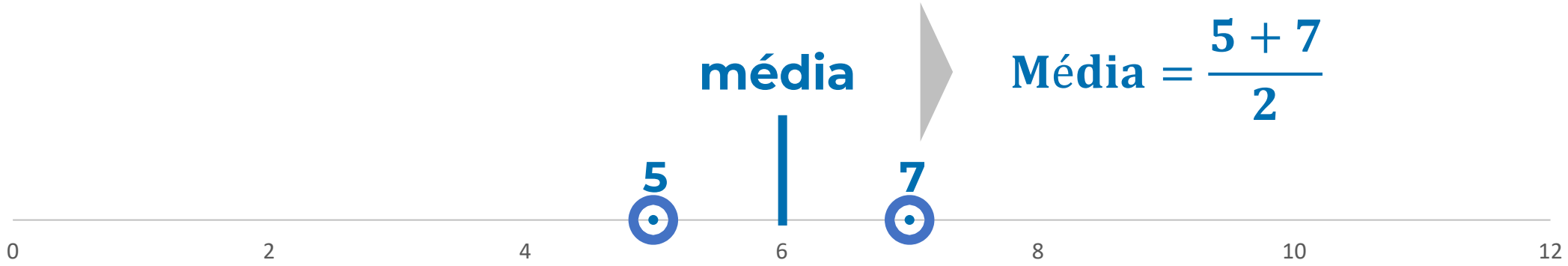
É basicamente a posição central de determinado conjunto de dados, é análogo ao “meio” (ou o centro de massa físico).

Vamos supor que estamos com dois números: 5 e 7.

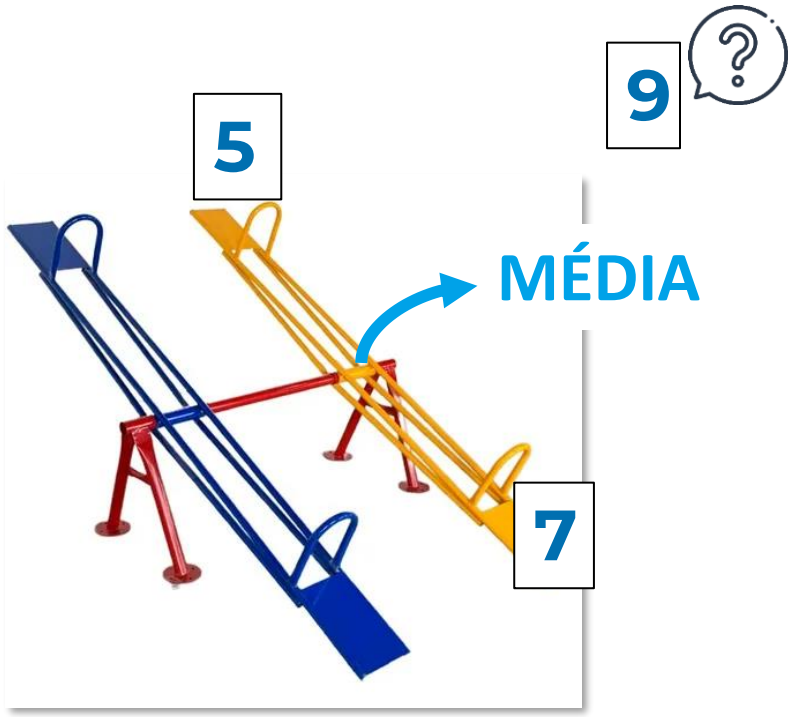
Como que conseguiríamos o ponto de equilíbrio entre esses dois números?



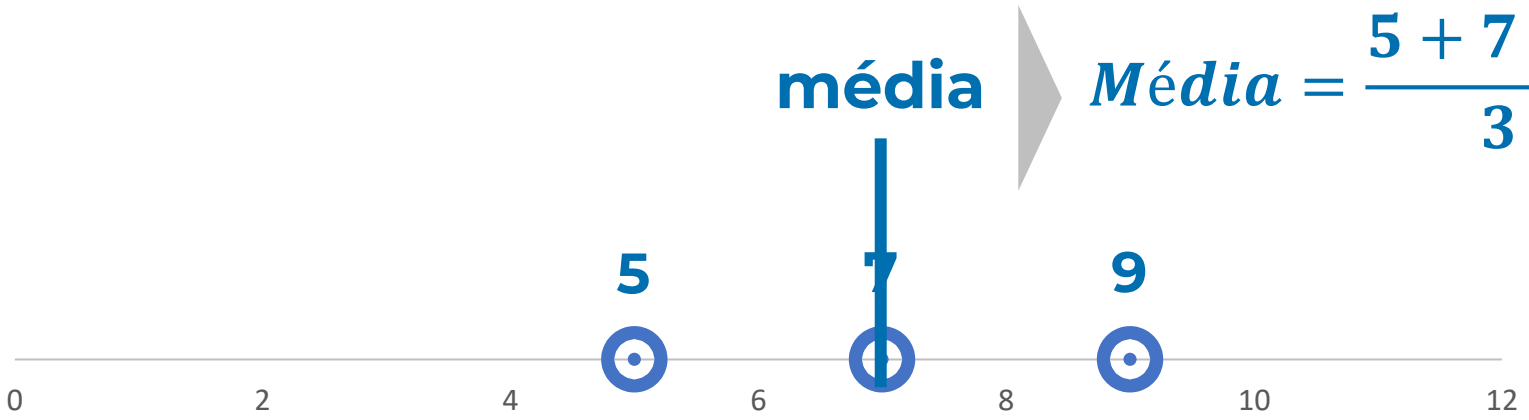
A média que estamos falando desse sistema seria exatamente o 6, que é o meio desses 2 números.

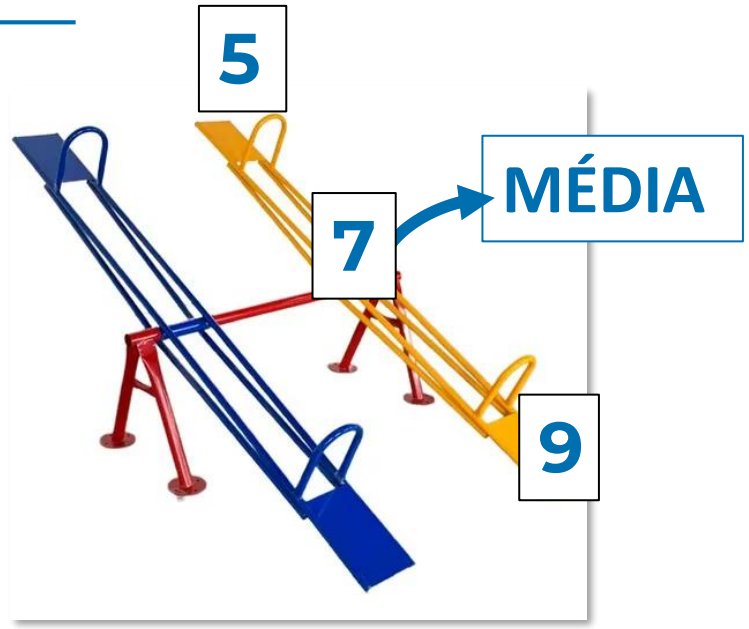


Vamos supor que colocamos um outro número. Como calculamos a média?



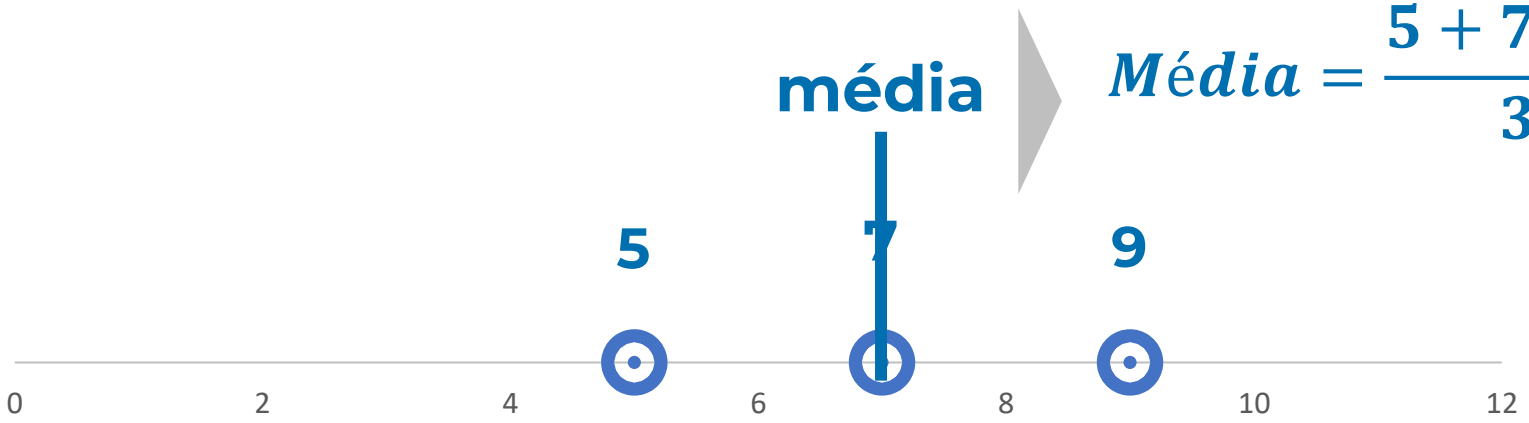
Vamos supor que colocamos um outro número. Como calculamos a média?



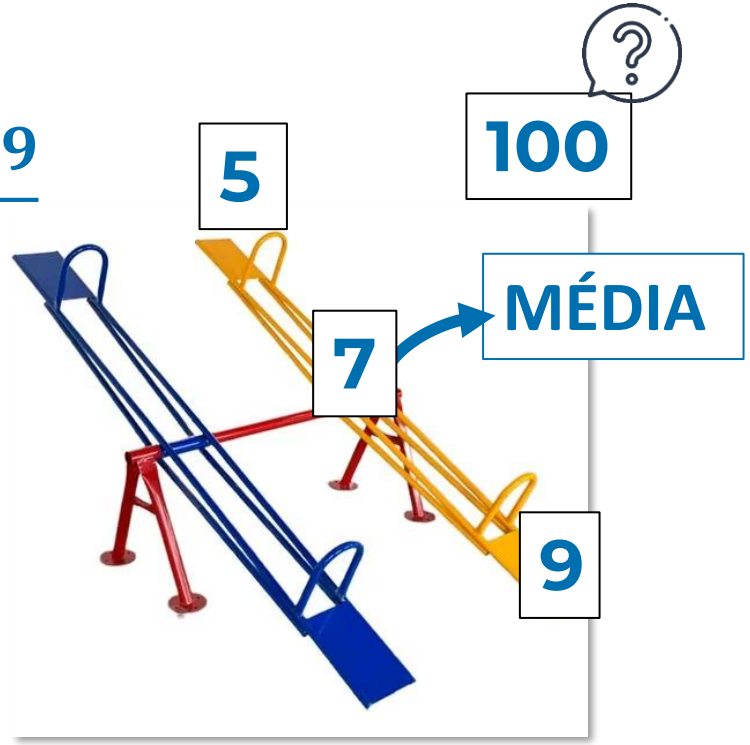


E se agora tivéssemos um *outlier*, por exemplo, o número 100.

Como calculamos a média?

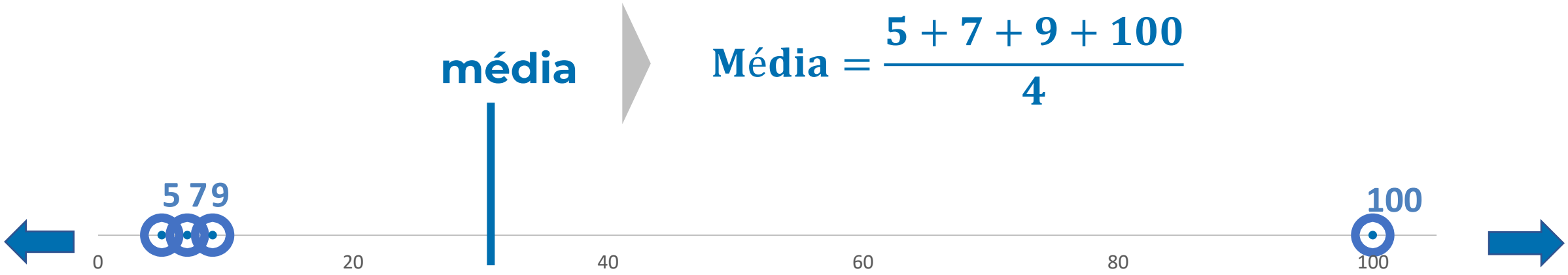


$$Média = \frac{5 + 7 + 9}{3}$$



Observa que o 100 é um número mais distante, então ele deslocou muito a média por ser um valor muito maior se comparado aos demais.

A **média é muito sensível a valores muito altos** e nós temos que tomar cuidado para não interpretar resultados de forma imprecisa apenas utilizando a média.



Módulo 6 – Entendendo o conceito de média

Para fazer a média no Pandas, basta utilizar o comando **mean**.

Estrutura: **base**.**mean**()

A média nos ajuda a **começar a entender a distribuição dos nossos dados**

```
base = {
    "valores": [5,7]
}

base = pd.DataFrame(base)

print(base.mean())
```

```
valores    6.0
dtype: float64
```

```
base = {
    "valores": [5,7,9]
}

base = pd.DataFrame(base)

print(base.mean())
```

```
valores    7.0
dtype: float64
```

```
base = {
    "valores": [5,7,9,100]
}

base = pd.DataFrame(base)

print(base.mean())
```

```
valores    30.25
dtype: float64
```

A média é muito **influenciada por outliers**!

Módulo 6 – Mediana e sua relação com a média

A mediana é o valor central de um conjunto de dados (ou seja, o valor que está no meio).

Obs: para isso, os dados precisam estar **ordenados**!

[1 , 5 , 100 , 12 , 28 , 37 , 9]

O que a mediana vai nos dizer?

A **mediana** vai nos dizer metade dos valores abaixo da mediana e metade acima, então se eu tenho **outlier não vai influenciar tanto**.

Vamos observar um exemplo com a lista de valores abaixo:

[1 , 5 , 100 , 12 , 28 , 37 , 9]

O primeiro passo é **ORDENAR OS DADOS**!

[1 , 5 , 9 , 12 , 28 , 37 , 100]

A **MEDIANA** será o valor na posição **central** do nosso conjunto de dados

Então, podemos dizer que temos 3 valores antes do 12 e 3 valores depois.

Agora, por exemplo, se tivéssemos uma quantidade par na nossa lista.

[1 , 5 , 9 , 12 , 15 , 28 , 37 , 100]

No caso de um número par de elementos, **a mediana será a media entre os 2 elementos centrais**

$$\textbf{Mediana} = \frac{12 + 15}{2}$$

Módulo 6 – Mediana e sua relação com a média

Como que podemos relacionar a mediana com a média?

Média igual a mediana

```
base = {
    "valores": [5,7]
}

base = pd.DataFrame(base)

print(base.median())
```

```
valores    6.0
dtype: float64
```

Média igual a mediana

```
base = {
    "valores": [5,7,9]
}

base = pd.DataFrame(base)

print(base.median())
```

```
valores    7.0
dtype: float64
```

Média é de 30,25 e mediana 8.

```
base = {
    "valores": [5,7,9,100]
}

base = pd.DataFrame(base)

print(base.median())
```

```
valores    8.0
dtype: float64
```

A mediana é **pouco influenciada por outliers**

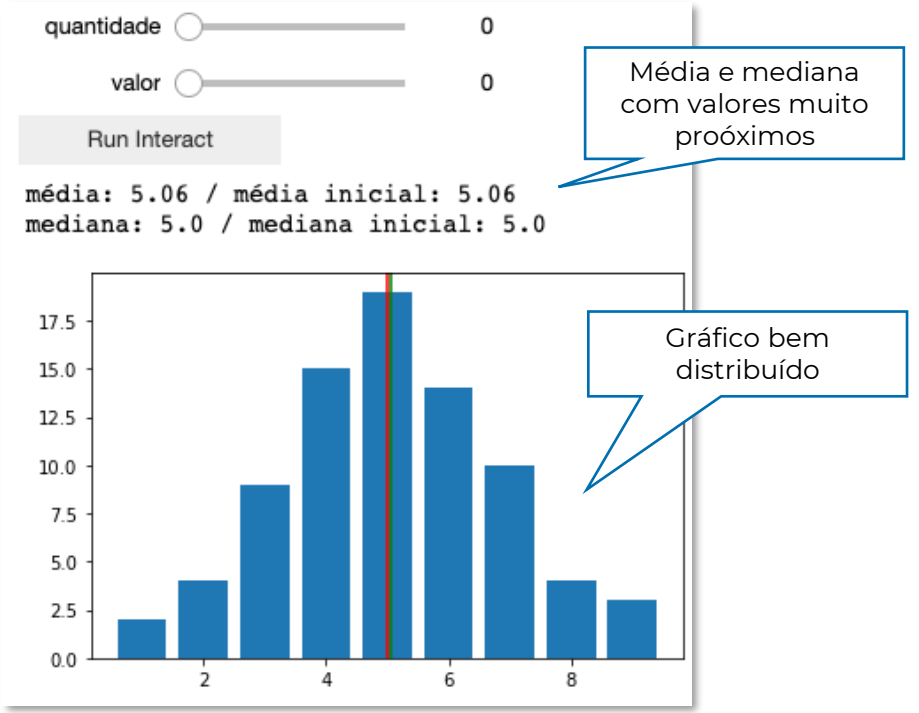
Módulo 6 – Mediana e sua relação com a média

Utilizamos média e mediana no nosso dia a dia:

- Avaliação de loja no Reclame aqui
- Avaliação do motorista da uber

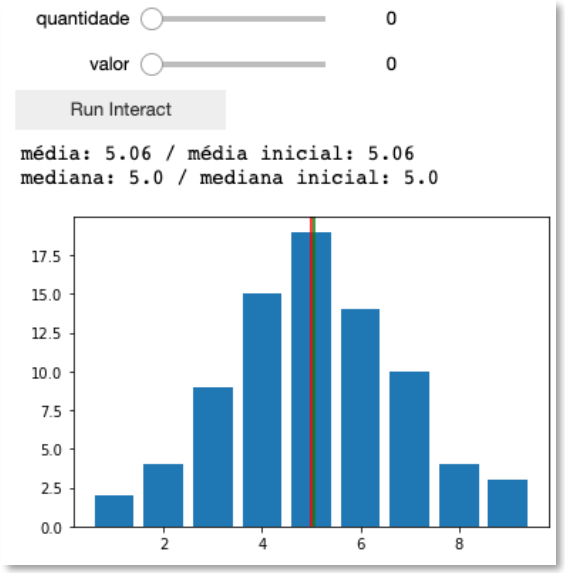
Módulo 6 – Usando Python para entender a relação entre média e mediana

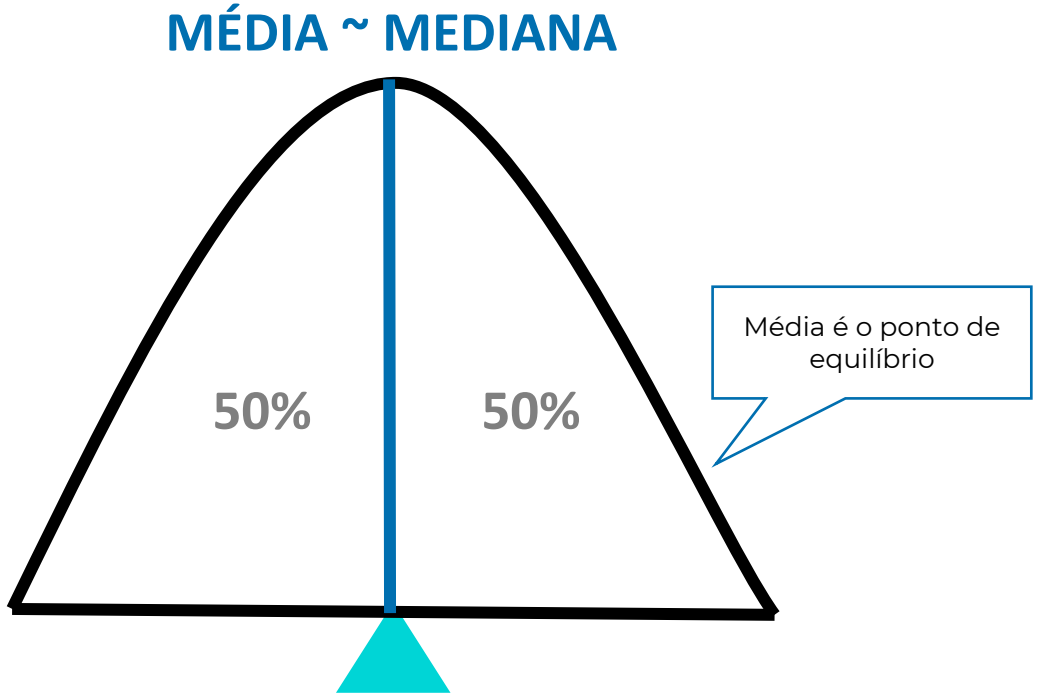
Ao começar a relacionar mediana com a média, já começamos a ter uma ideia da distribuição dos nossos dados



Módulo 6 – Usando Python para entender a relação entre média e mediana

Ao fazer um desenho passando por todos os pontos de maior frequência, obtemos algo muito próximo de um sino, conforme imagem à direita.





Módulo 6 – Usando Python para entender a relação entre média e mediana

Vamos ver agora como alterar a **média** e **mediana**.

Supondo que temos a base abaixo onde todos os valores são 5, nesse caso a **média** e **mediana** são **iguais**.

```
                                                  1 2 3 4 5 6 7 8 9 10 11
notas2 = np.array([5,5,5,5,5,5,5,5,5,5,5])
```

Mediana é o valor da posição 6

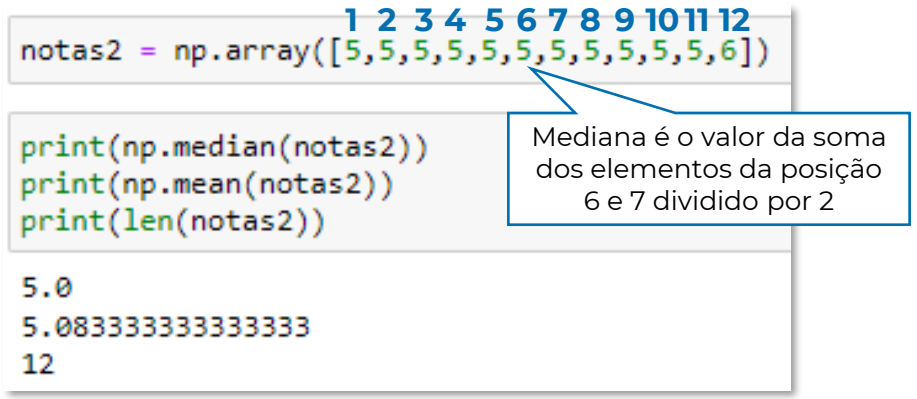
```
print(np.median(notas2))
print(np.mean(notas2))
print(len(notas2))
```

```
5.0
5.0
11
```

Módulo 6 – Usando Python para entender a relação entre média e mediana

Agora, se adicionarmos um valor 6 nessa base, podemos notar que apenas a **média** altera.

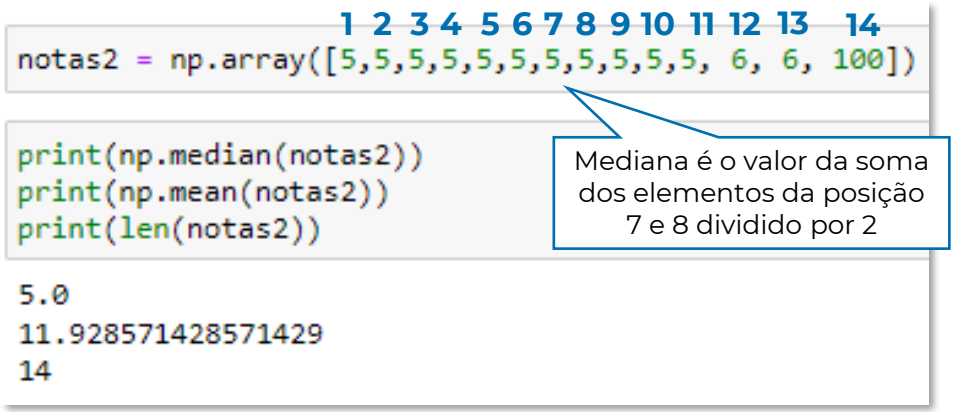
Como a **mediana** é o valor da posição central, por mais que o valor não tenha sido alterado, observe que a posição foi.



Módulo 6 – Usando Python para entender a relação entre média e mediana

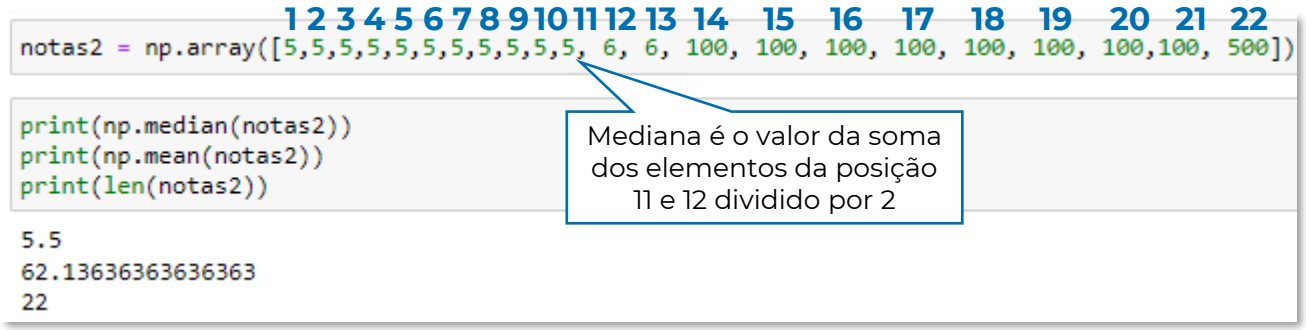
Observe que para alterar a **média** foi necessário apenas acrescentar um valor, enquanto a **mediana** alterou somente a posição do elemento.

Então, se adicionarmos um valor de 100 na nossa base, ele afeta a **média** e a **mediana** continua com o mesmo valor.



Módulo 6 – Usando Python para entender a relação entre média e mediana

Só vamos conseguir alterar o valor da **mediana** se adicionarmos muitos valores a ponto de colocar o 6 como a posição central.

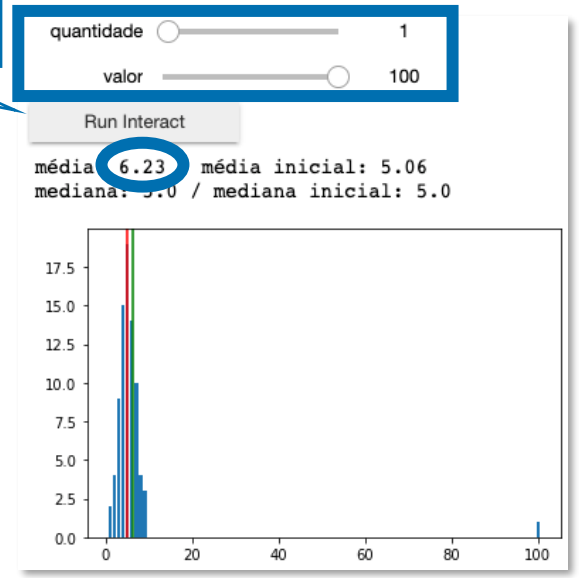


Módulo 6 – Usando Python para entender a relação entre média e mediana

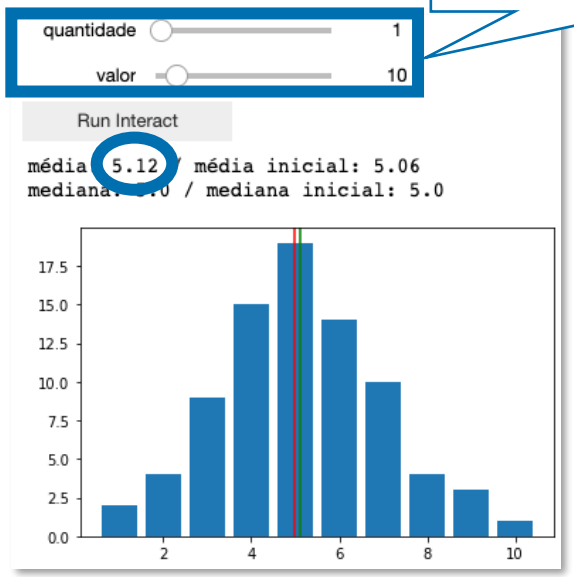
Então, a **média** alteramos com a **adição de um único valor.**

Quanto mais alto for o valor, mais alta a minha **média**.

1 único outlier já é suficiente para mudar significativamente a média

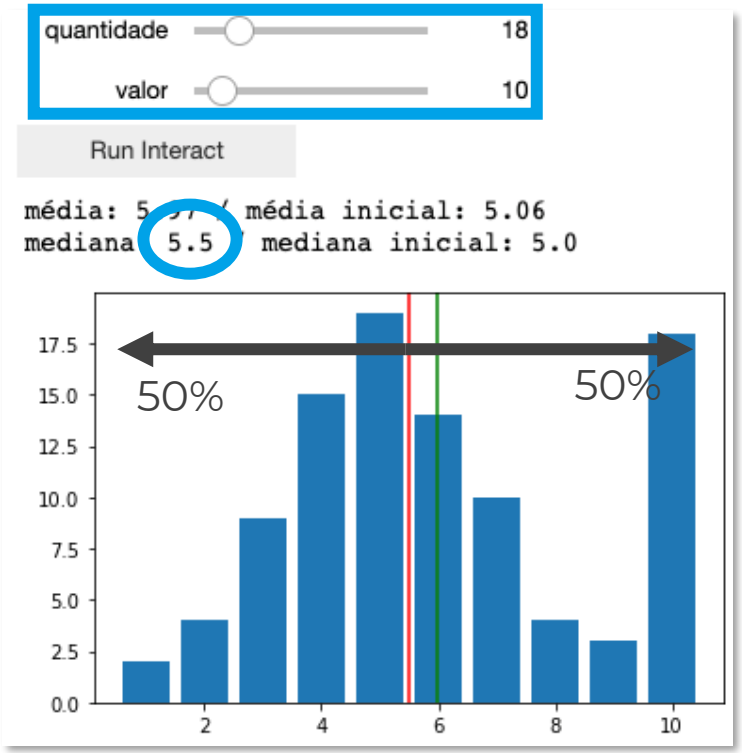


Toda vez que adicionarmos novos valores vamos mudar a média



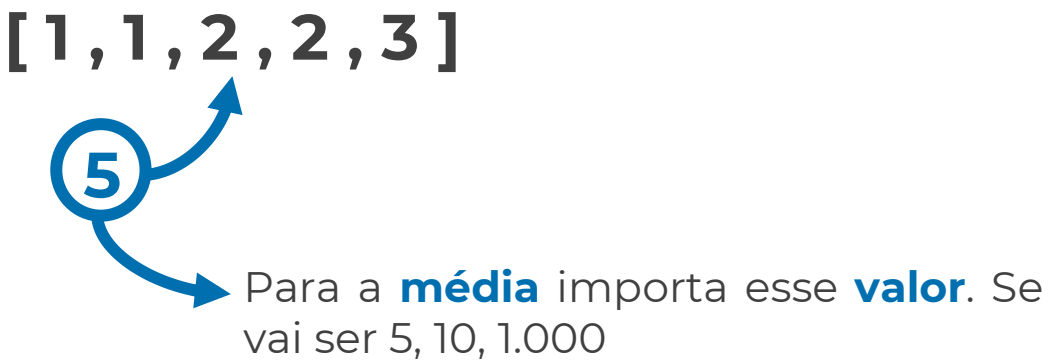
Módulo 6 – Usando Python para entender a relação entre média e mediana

Só vamos mudar a **mediana** quando mudarmos a **quantidade de dados** adicionada a base.



Imagine que temos essa distribuição de dados abaixo e queremos adicionar o 5.

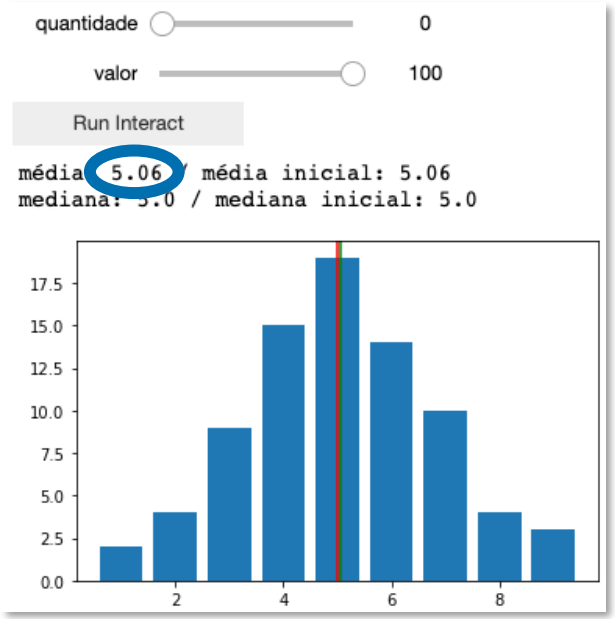
O que isso impacta na **média** e **mediana**?



Imagine que temos essa distribuição de dados abaixo e queremos adicionar o 5.

O que isso impacta na **média** e **mediana**?

Módulo 6 – Média, mediana e moda

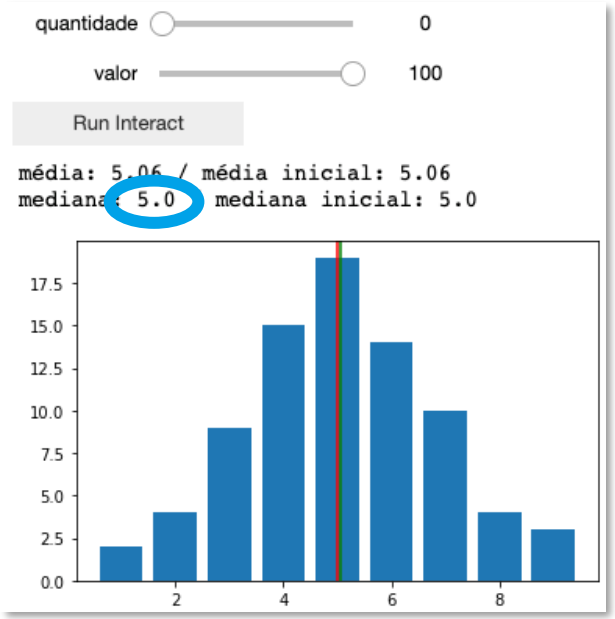


Podemos utilizar a média para **analisar a avaliação de empresas.**

Quando escolhemos um range para a nota, tornamos ele **menos sensível a outliers**.

Por exemplo: a nota varia de 0 a 10 ou de 1 a 5

Módulo 6 – Média, mediana e moda

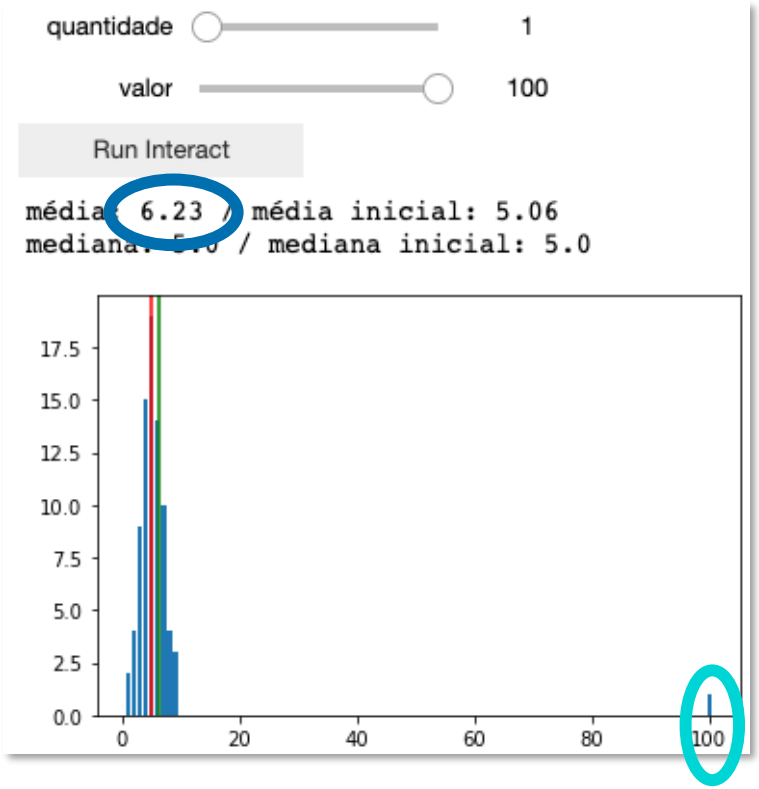


Já a **mediana** podemos usar para avaliar o **tempo de resposta de uma solicitação**.

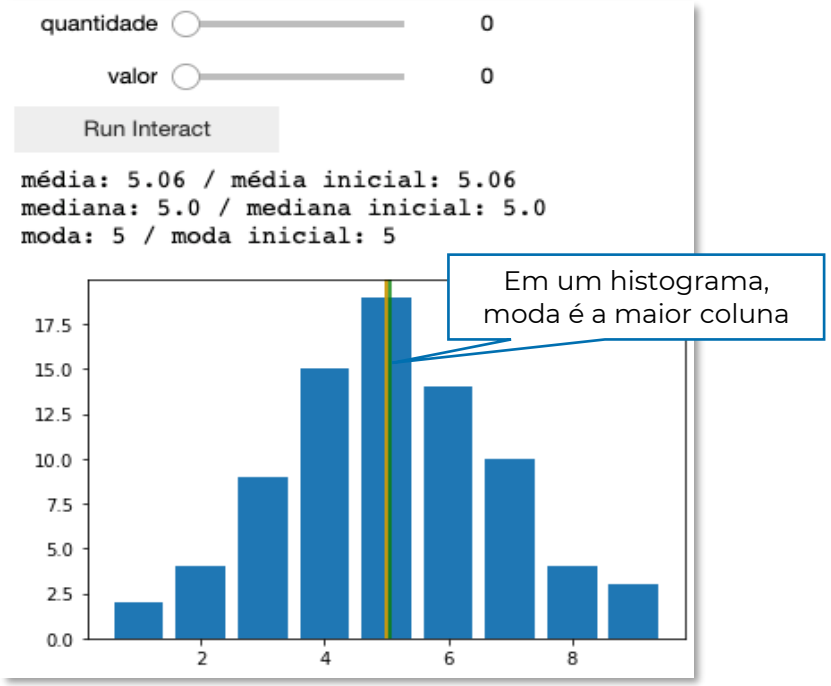
Se ocorreu um problema no sistema que a solicitação não foi fechada, isso não vai penalizar a empresa

Podemos usar a **média** e os **outliers** para identificação de fraudes.

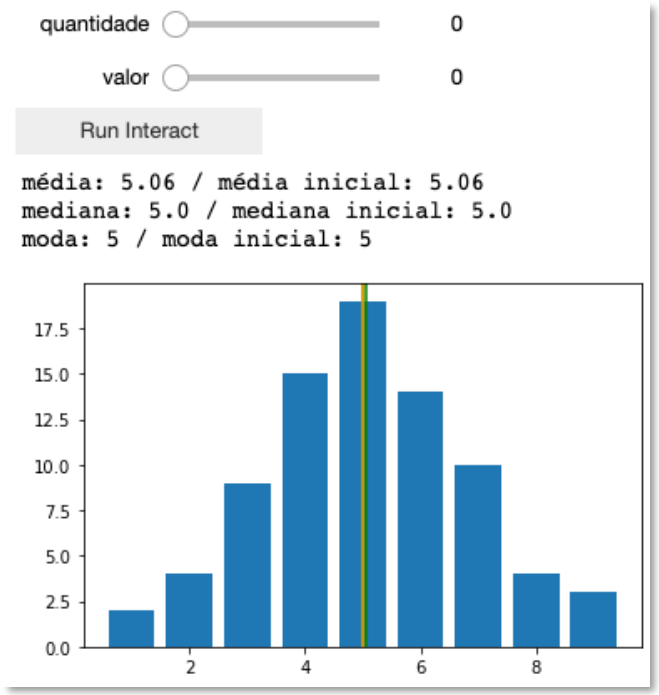
Um outro exemplo é para verificar o que está fora do padrão dos dados.

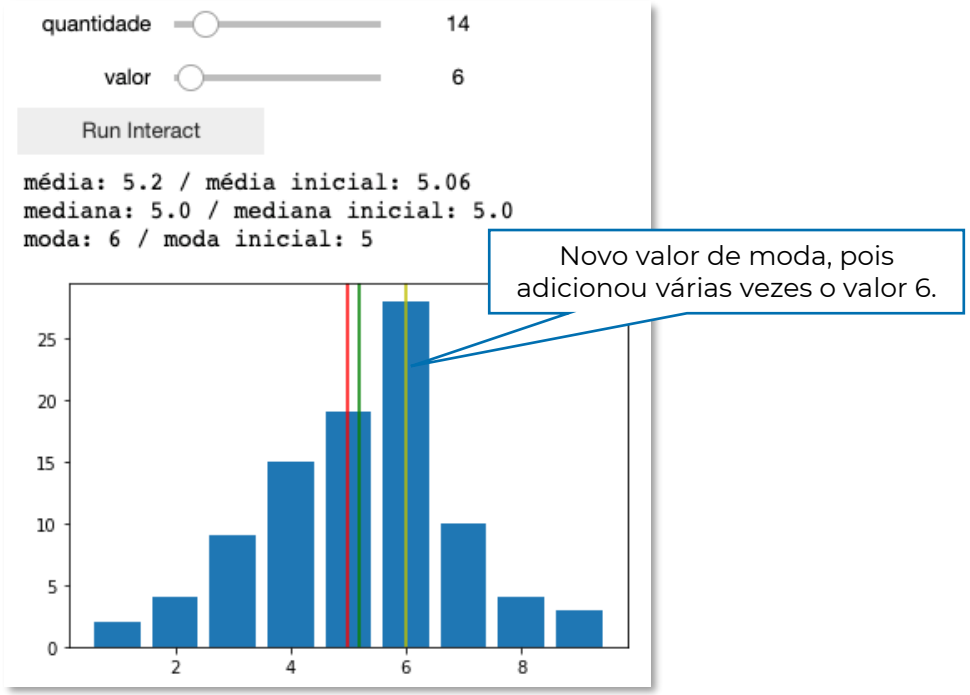


Vamos agora falar de Moda. Consiste no **valor mais frequente** do conjunto de dados.



Só vamos conseguir mudar a moda se a quantidade de registros adicionado for suficiente para fazer **esse valor ser o mais frequente**.





Então relembrando:

Tipo	DESVANTAGENS
Média	É influenciada pela **adição de qualquer valor**.
Mediana	É influenciada pela **quantidade de valores**, independente do valor.
Moda	É influenciada pela quantidade de **valores iguais**.

Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Agora vamos ver alguns exemplos práticos:

Média: 1.67
Mediana: 1.5
Moda: 1
Registros: 6
Mínimo: 1
Máximo: 3

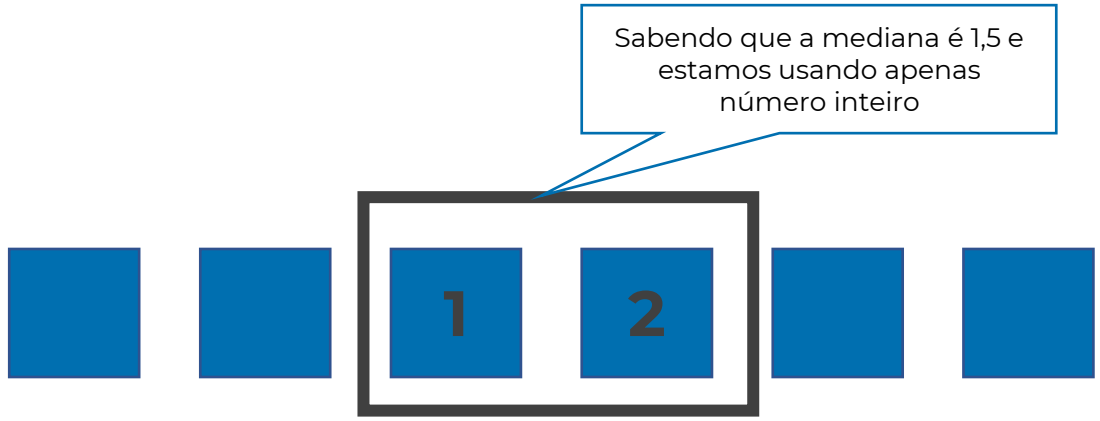
Todos são valores inteiros

Com essas informações conseguimos entender como está a base.

Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Média: 1.67
Mediana: 1.5
Moda: 1
Registros: 6
Mínimo: 1
Máximo: 3

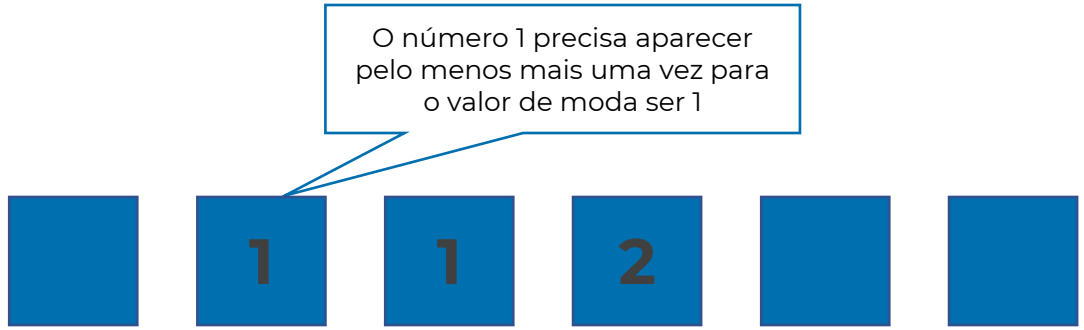
Todos são valores inteiros



Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Média: 1.67
Mediana: 1.5
Moda: 1
Registros: 6
Mínimo: 1
Máximo: 3

Todos são valores inteiros



Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Média: 1.67
Mediana: 1.5
Moda: 1
Registros: 6
Mínimo: 1
Máximo: 3

Todos são valores inteiros

Média: 1.67
Mediana: 1.5
Moda: 1
Registros: 6
Mínimo: 1
Máximo: 3

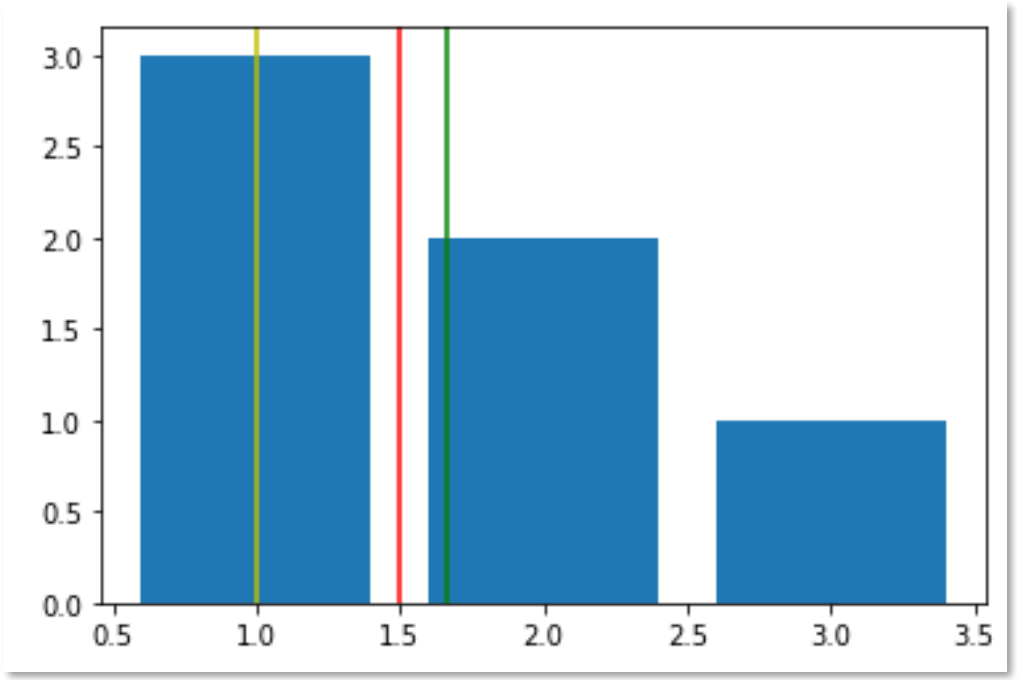
Todos são valores inteiros

Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Média: 1.67
Mediana: 1.5
Moda: 1
Registros: 6
Mínimo: 1
Máximo: 3

Todos são valores inteiros

1	1	1	2	2	3



Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

```
base.mean()
```

```
Produto1    6.9
Produto2    6.7
Produto3    7.4
Produto4    9.0
Produto5    2.8
Produto6    2.0
dtype: float64
```

Agora, vamos ver uma base de avaliação de produtos, conforme mostrado ao lado.

Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

```
base.mean()
```

```
Produto1     6.9
Produto2     6.7
Produto3     7.4
Produto4     9.0
Produto5     2.8
Produto6     2.0
dtype: float64
```

Você precisa **retirar um produto de linha**!

Com as avaliações do produto mostradas ao lado, **qual você escolheria**?

Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

```
base.mean()
```

```
Produto1     6.9
Produto2     6.7
Produto3     7.4
Produto4     9.0
Produto5     2.8
Produto6     2.0
dtype: float64
```

Apenas com a média, nenhum!

Precisamos de **mais informações** para responder a essa pergunta!

Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Com as informações de média, mediana e moda, podemos ver que o produto 6 é ó pior.

Porém, repare que os valores são **exatamente iguais**

	produtos	média	mediana	moda
0	Produto1	6.9	7.0	6.0
1	Produto2	6.7	7.0	9.0
2	Produto3	7.4	7.0	7.0
3	Produto4	9.0	9.0	9.0
4	Produto5	2.8	2.0	2.0
5	Produto6	2.0	2.0	2.0

Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Agora, avaliando o número de registros, mínimo e máximo, podemos observar que o produto 6 só teve **uma avaliação.**

Como a avaliação é mensal, então ele é um **produto novo** e talvez as pessoas não tiveram tempo de conhecer.

	produtos	média	mediana	moda	registros	mínimo	máximo
0	Produto1	6.9	7.0	6.0	10	6.0	8.0
1	Produto2	6.7	7.0	9.0	10	3.0	10.0
2	Produto3	7.4	7.0	7.0	10	5.0	9.0
3	Produto4	9.0	9.0	9.0	1	9.0	9.0
4	Produto5	2.8	2.0	2.0	10	1.0	10.0
5	Produto6	2.0	2.0	2.0	1	2.0	2.0

Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

O produto 5 possui 10 registros e uma **média bem abaixo** dos outros produtos.

Então, provavelmente ele seria o escolhido.

	produtos	média	mediana	moda	registros	mínimo	máximo
0	Produto1	6.9	7.0	6.0	10	6.0	8.0
1	Produto2	6.7	7.0	9.0	10	3.0	10.0
2	Produto3	7.4	7.0	7.0	10	5.0	9.0
3	Produto4	9.0	9.0	9.0	1	9.0	9.0
4	Produto5	2.8	2.0	2.0	10	1.0	10.0
5	Produto6	2.0	2.0	2.0	1	2.0	2.0

Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Outro exemplo que podemos analisar é a média de salário de duas empresas.

Qual empresa você preferiria?

Se olharmos apenas para a média, escolheríamos a empresa 1. Porém vamos analisar os outros dados.

	empresas	média	mediana	moda	registros	mínimo	máximo
0	1	6954.2	2155.0	5000	12	1000	57000
1	2	5983.3	5100.0	4200	12	3000	15000

Esse valor 57.000 está fazendo **a média ser mais alta** (outlier)!

	empresas	média	mediana	moda	registros	mínimo	máximo
0	1	6954.2	2155.0	5000	12	1000	57000
1	2	5983.3	5100.0	4200	12	3000	15000

Se retirarmos esse valor de 57.000, a **média** seria de aproximadamente 2.404,6.

	empresas	média	mediana	moda	registros	mínimo	máximo
0	1	6954.2	2155.0	5000	12	1000	57000
1	2	5983.3	5100.0	4200	12	3000	15000

Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Na empresa 1, metade das pessoas ganha menos que 2.155 reais, já na empresa 2 a metade ganha até 5.100 reais.

	empresas	média	mediana	moda	registros	mínimo	máximo
0	1	6954.2	2155.0	5000	12	1000	57000
1	2	5983.3	5100.0	4200	12	3000	15000

Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Na empresa 2, você vai receber **pelo menos** 3 mil reais, já na empresa 1 você pode começar ganhando mil reais.

Então, na empresa 2 metade dessa pessoas ganha entre 3000 e 5100, enquanto na empresa 1 metade ganha entre 1000 e 2155.

	empresas	média	mediana	moda	registros	mínimo	máximo
0	1	6954.2	2155.0	5000	12	1000	57000
1	2	5983.3	5100.0	4200	12	3000	15000

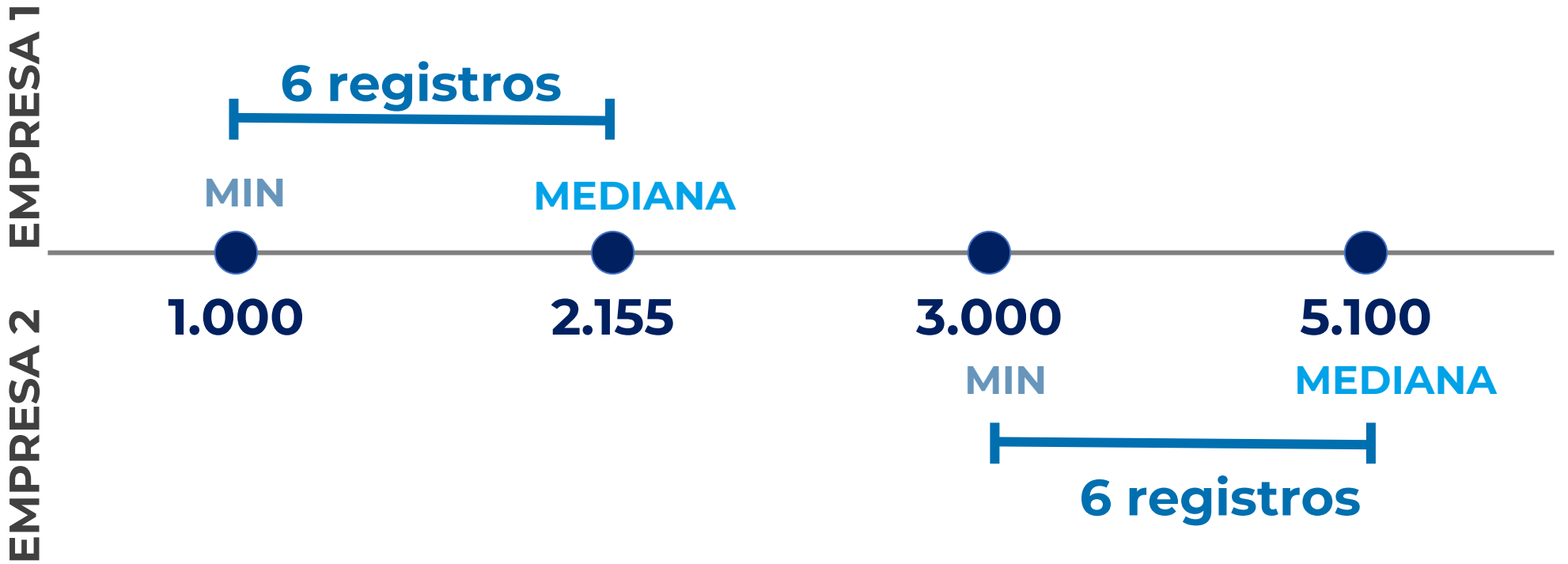
Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Na empresa 2, o **menor salário é MAIOR que a mediana** da empresa 1.

Ou seja, a pessoa que menos ganha na empresa 2, ganha mais que a metade da empresa 1.

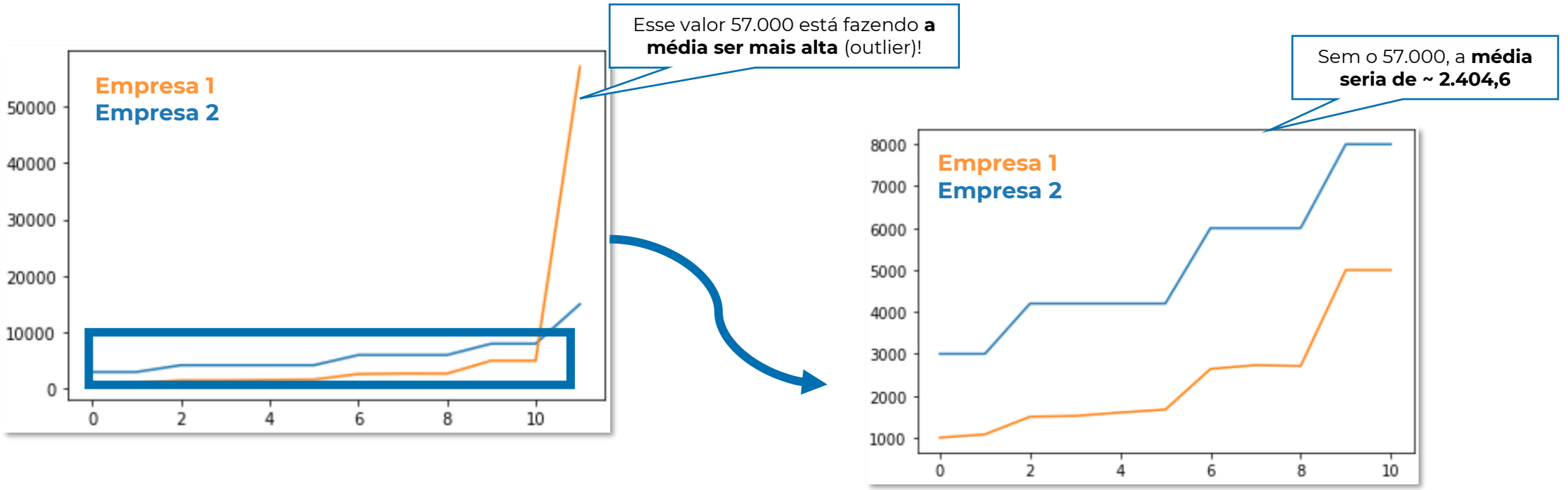
	empresas	média	mediana	moda	registros	mínimo	máximo
0	1	6954.2	2155.0	5000	12	1000	57000
1	2	5983.3	5100.0	4200	12	3000	15000

Podemos ver de uma forma mais visual e nesse caso já podemos perceber que a **empresa 2 é muito mais vantajosa**.



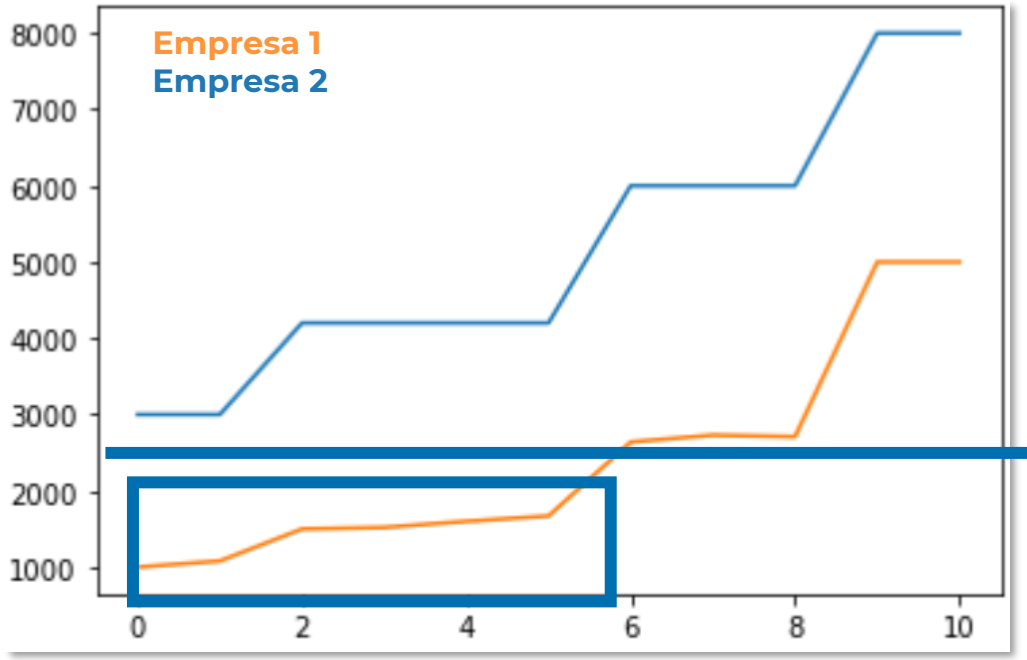
Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Se olharmos para os gráficos de salário, podemos observar que o outlier na empresa 1 está fazendo a média ser mais alta.

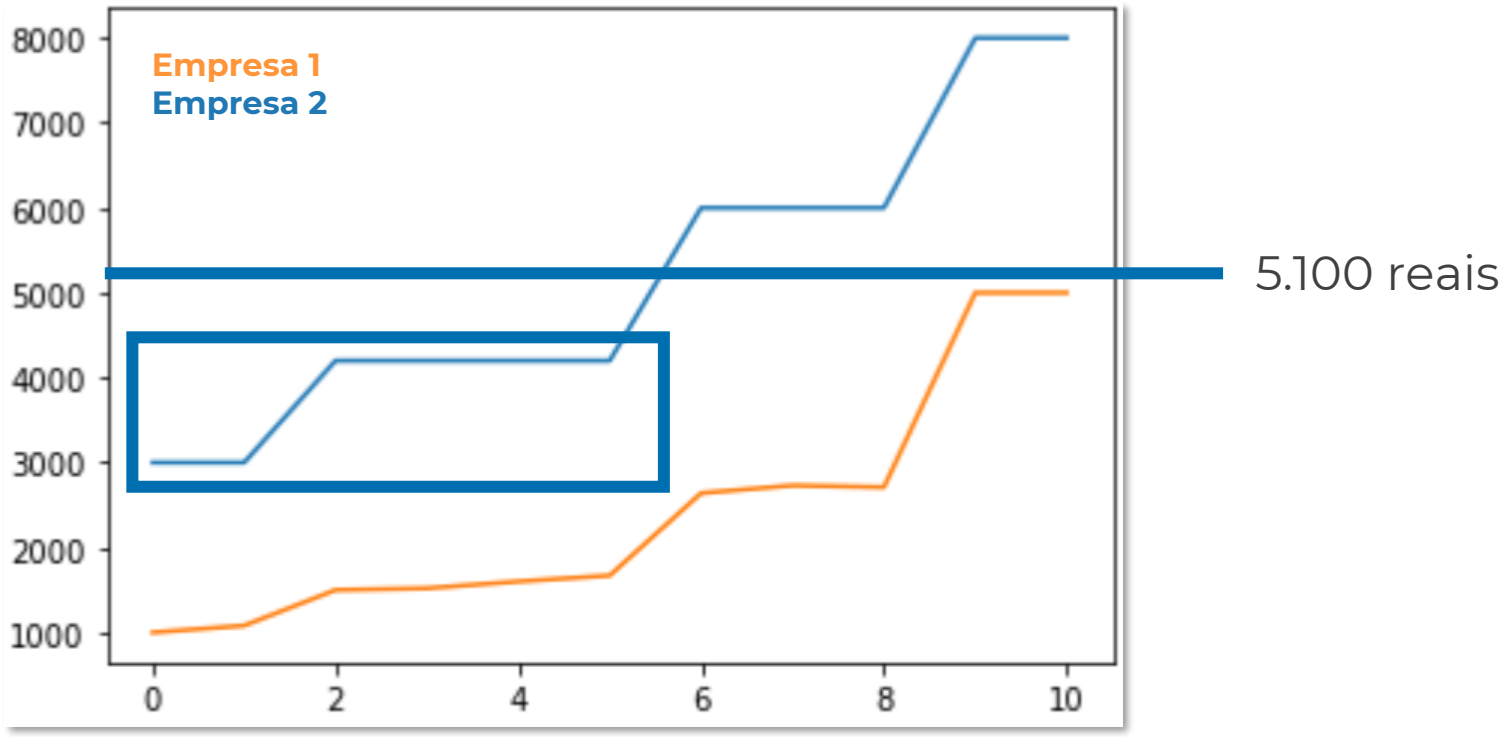


Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Na empresa 1, metade das pessoas ganha menos que 2.155 reais.

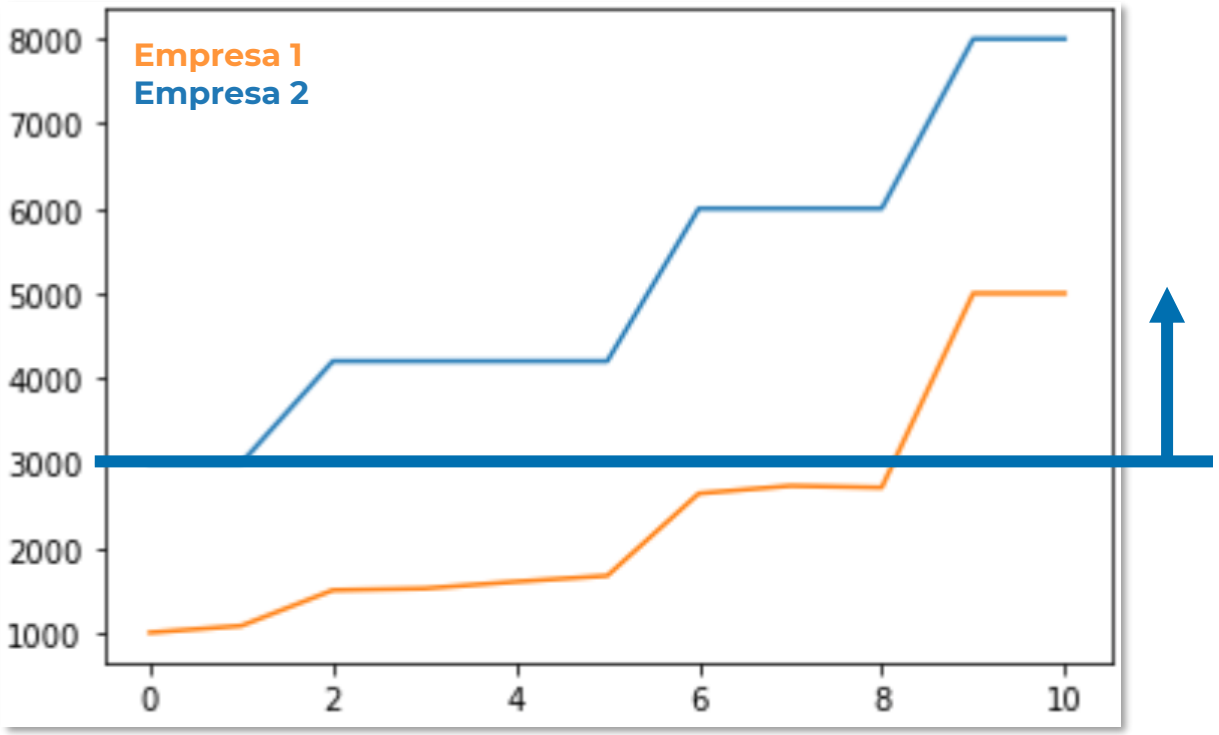


Já na empresa 2, a metade ganha até 5.100 reais.



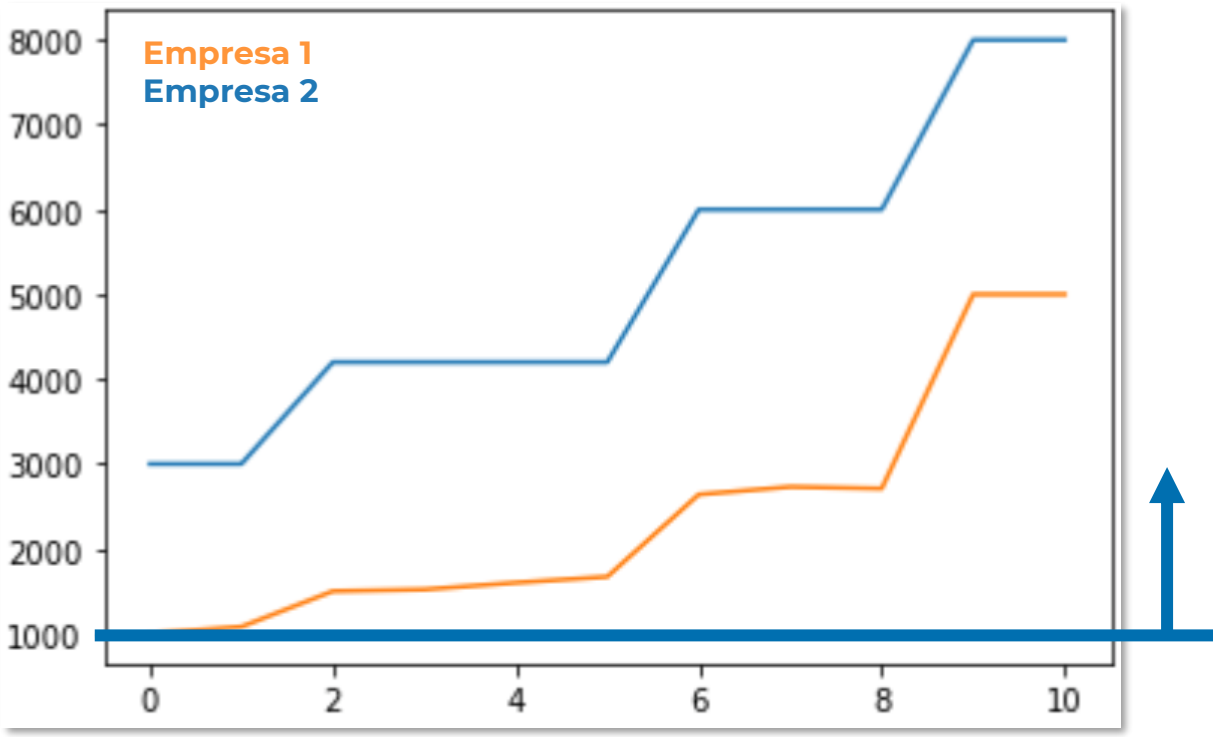
Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Na empresa 2, você vai receber **pelo menos 3 mil reais**.

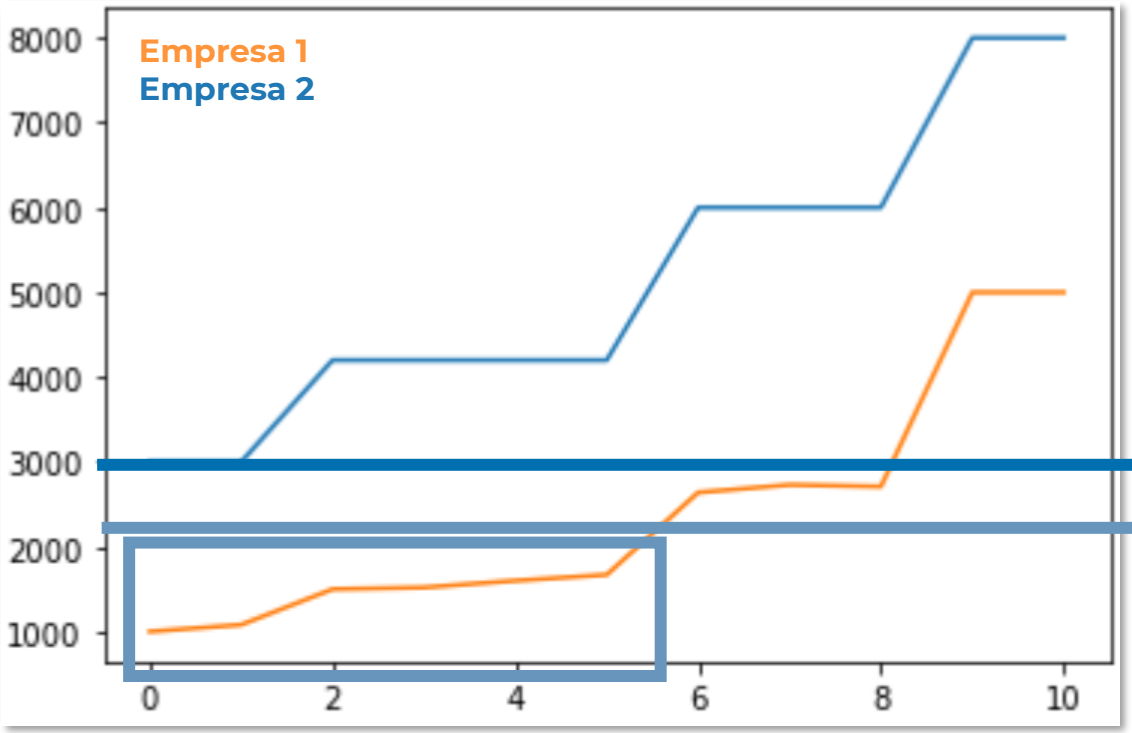


Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Já na empresa 1, você pode começar ganhando **mil reais** e chegar a 9 graduações de cargo para receber mais que 3000 reais.



Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda



Na empresa 2, o **menor salário é MAIOR que a mediana** da empresa 1.

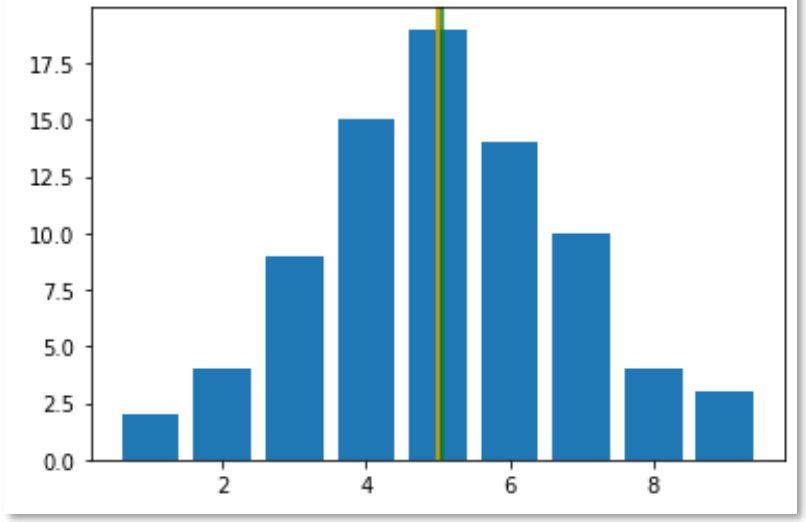
MÍNIMO DA EMPRESA 2

MEDIANA DA EMPRESA 1

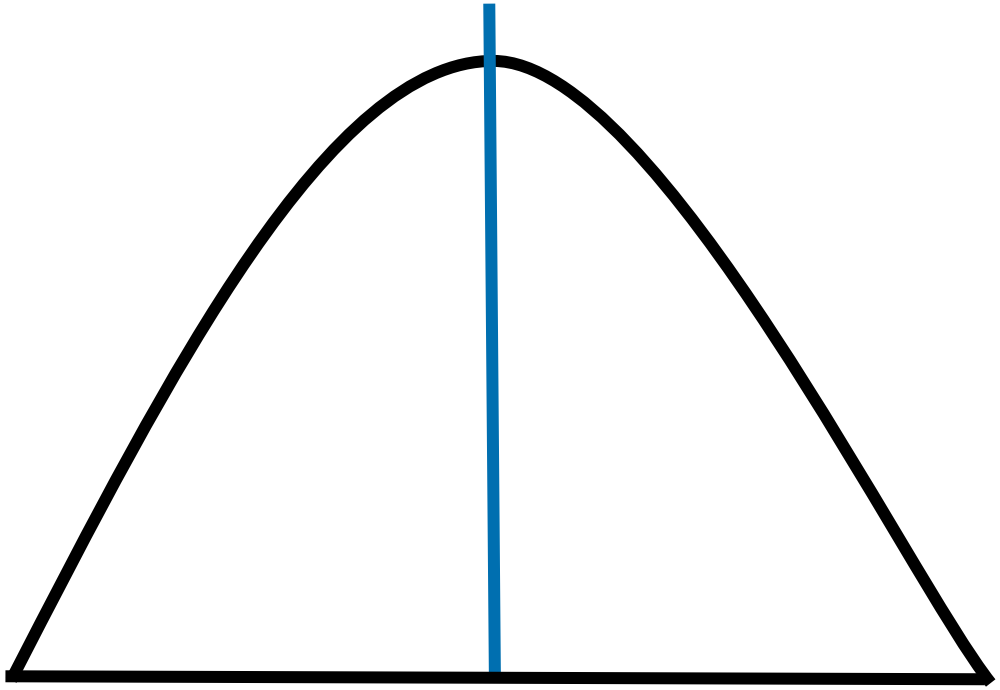
Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Quando temos muitos dados, vemos que eles estão muito bem distribuídos.

```
registros: 80
média: 5.06 / média inicial: 5.06
mediana: 5.0 / mediana inicial: 5.0
moda: 5 / moda inicial: 5
```



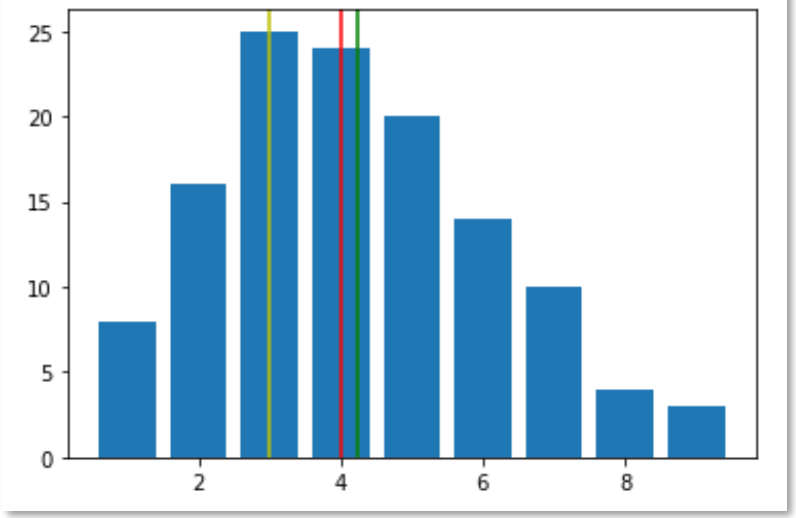
MÉDIA = MEDIANA = MODA



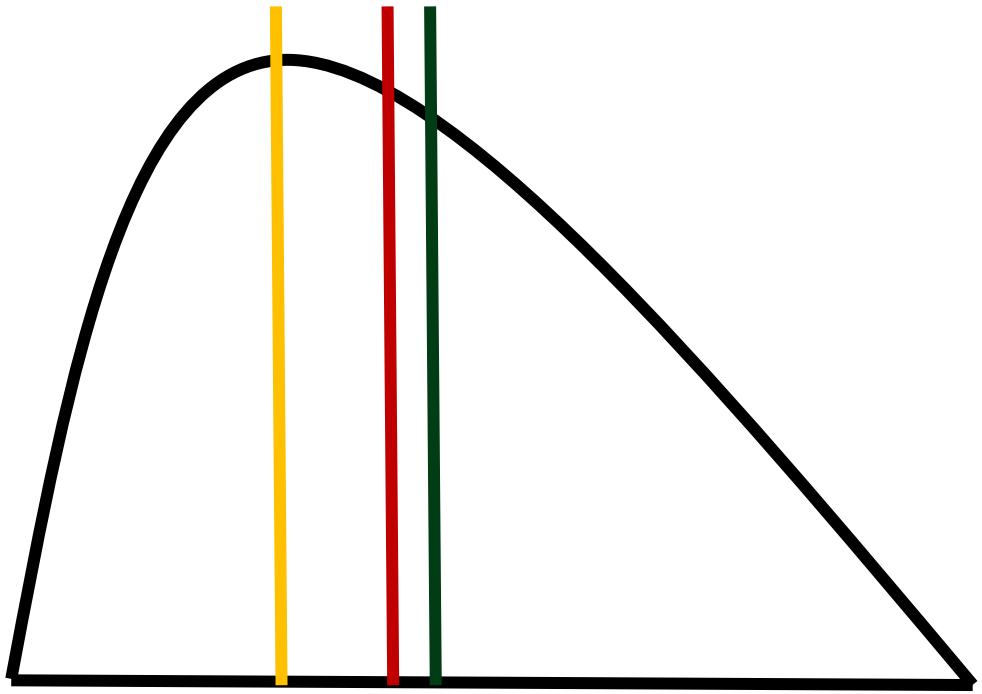
Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Já quando temos média maior que mediana e moda, os dados tendem mais para os valores menores.

```
registros: 124
média: 4.23 / média inicial: 5.06
mediana: 4.0 / mediana inicial: 5.0
moda: 3 / moda inicial: 5
```



MODA < MEDIANA < MÉDIA

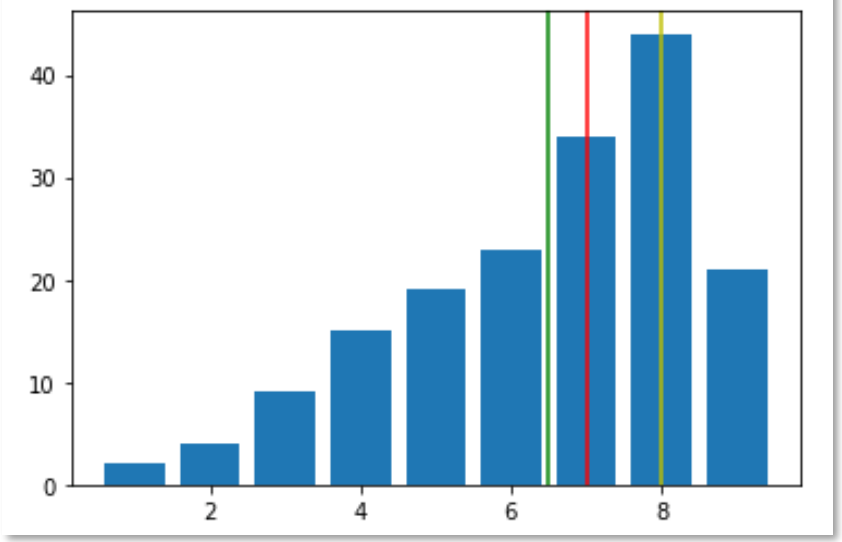


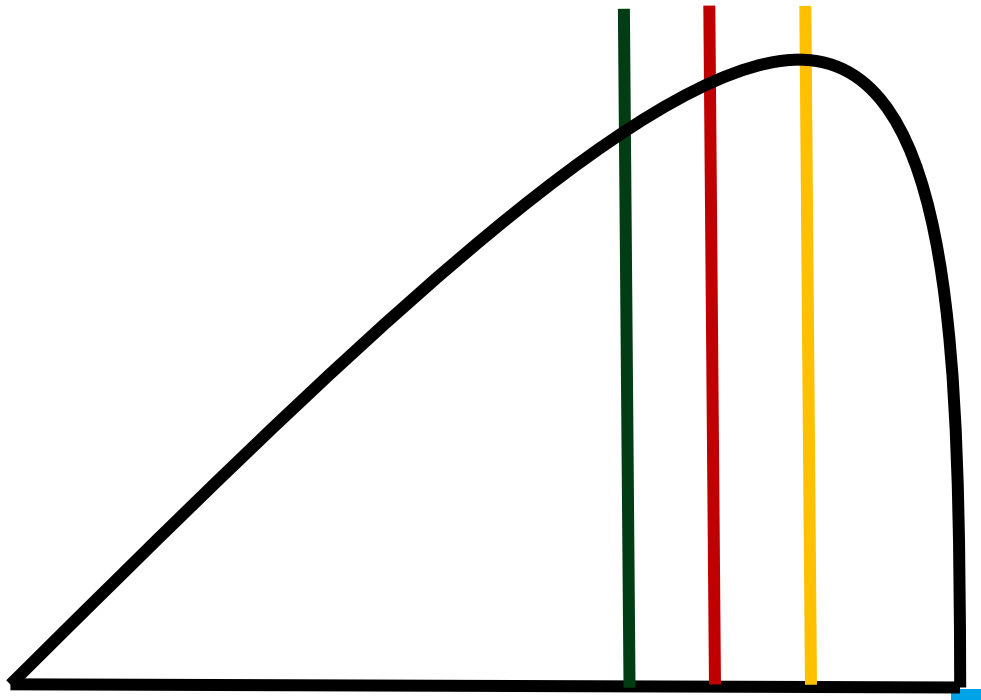
Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Quando temos média menor que mediana e moda, os dados tendem mais para os valores maiores.

MÉDIA < MEDIANA < MODA

```
registros: 171
média: 6.49 / média inicial: 5.06
mediana: 7.0 / mediana inicial: 5.0
moda: 8 / moda inicial: 5
```



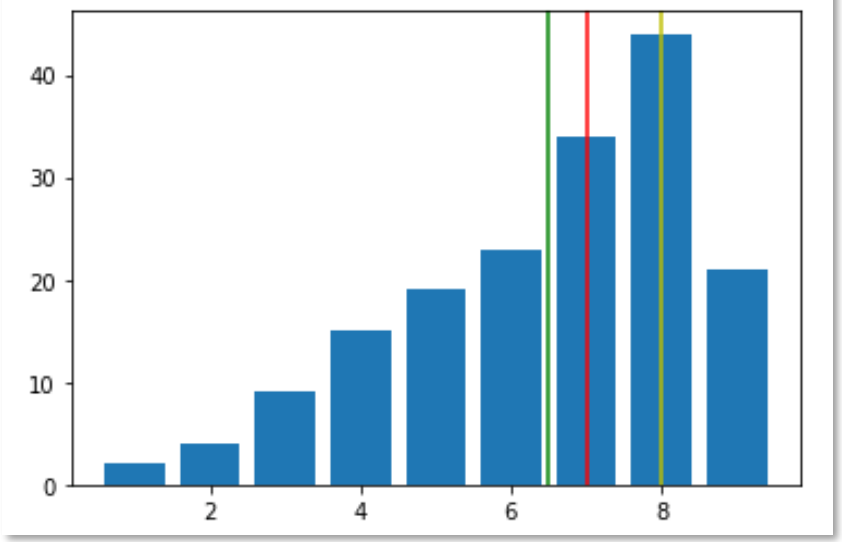


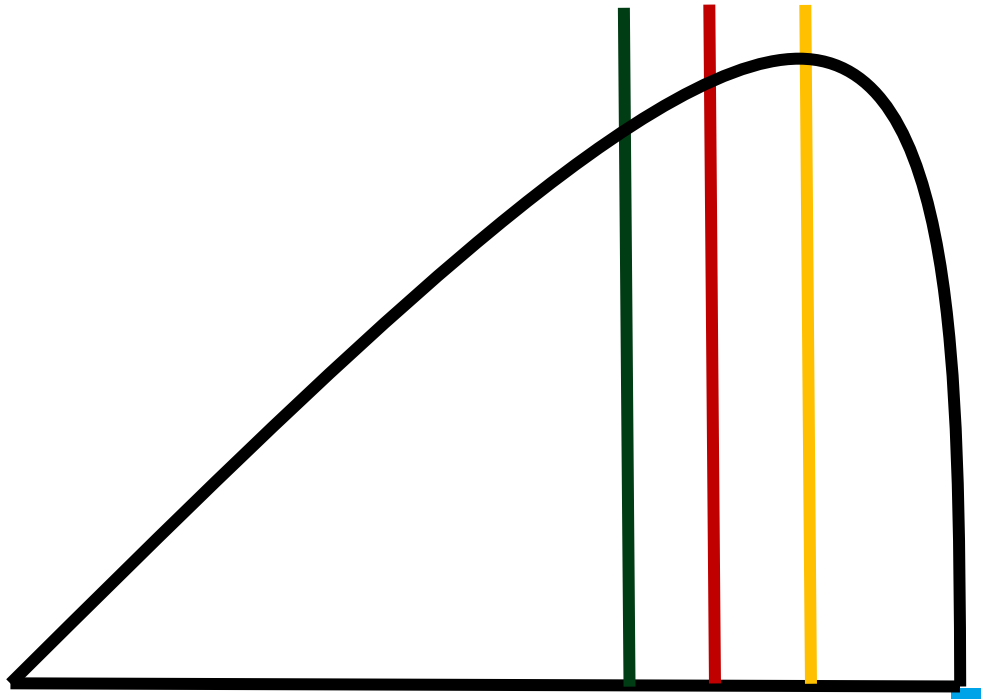
Módulo 6 – Entendendo de forma prática a relação entre média, mediana e moda

Agora, apenas com a média, mediana e moda você já consegue ter uma boa noção de como está a **distribuição dos seus dados**!

MÉDIA < MEDIANA < MODA

```
registros: 171
média: 6.49 / média inicial: 5.06
mediana: 7.0 / mediana inicial: 5.0
moda: 8 / moda inicial: 5
```



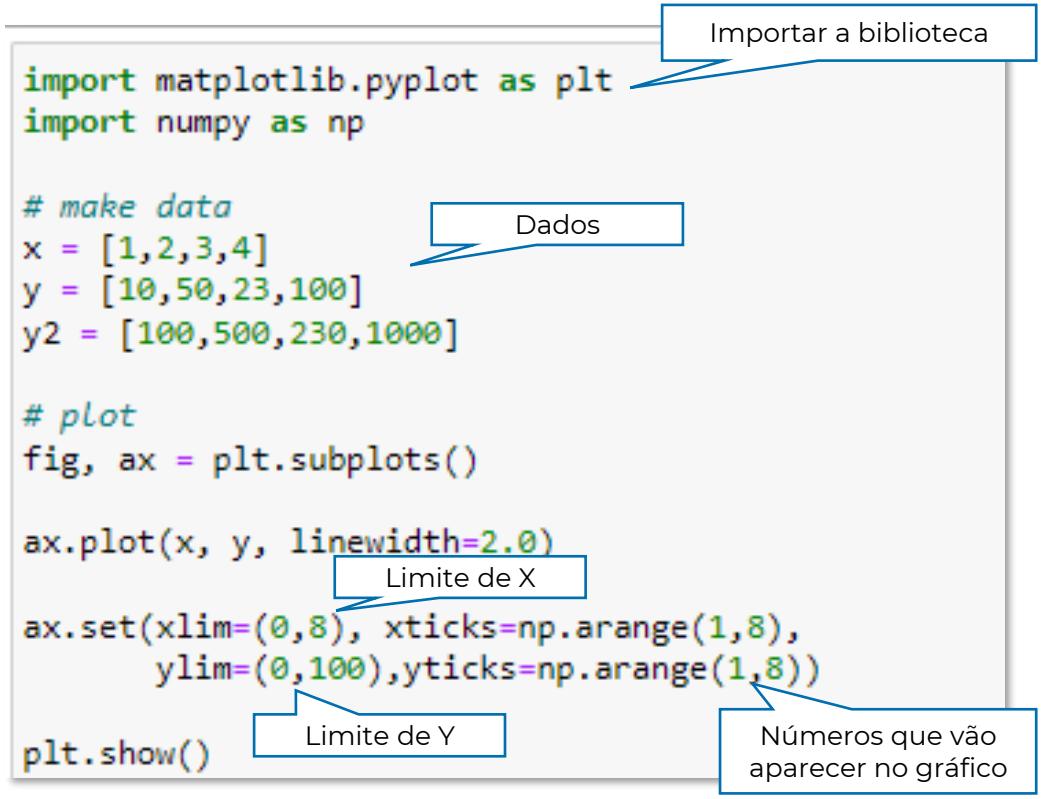


MÓDULO 7

Matplotlib: Criando gráficos em Python

Módulo 7 – Apresentando o Matplotlib: Criando gráficos em Python

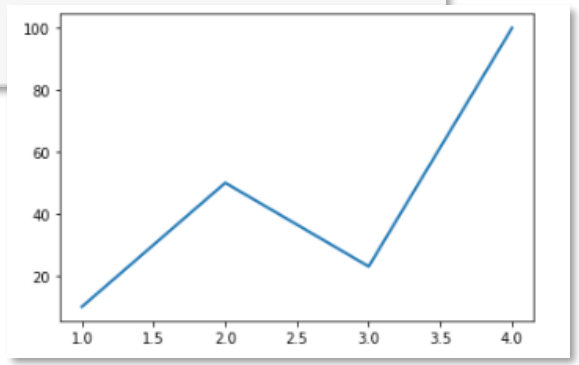
Nesse módulo vamos falar do Matplotlib, que é uma biblioteca que nos ajuda a criar gráficos no Python.



Se removermos os limites dos eixos o próprio Matplotlib vai procurar a melhor forma de mostrar o gráfico.

Mas antes disso vamos entender o que significa os termos do exemplo ao lado.

```
import matplotlib.pyplot as plt
import numpy as np

# make data
x = [1,2,3,4]
y = [10,50,23,100]

# plot
fig, ax = plt.subplots()

ax.plot(x, y, linewidth=2.0)

plt.show()
```

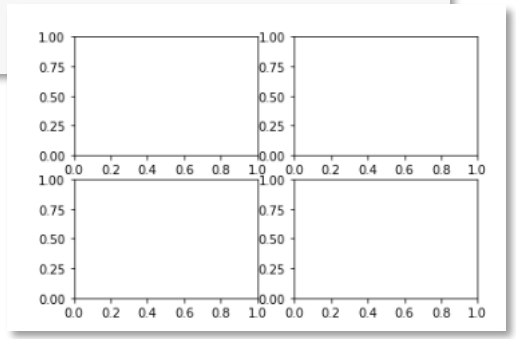


A função **plt.subplot** retorna 2 argumentos: fig (figura) e ax (eixo).

Por exemplo: **plt.subplot(nrows=2, ncols = 2)**

Então tudo isso vai aparecer dentro da figura.

```
import matplotlib.pyplot as plt
import numpy as np

# make data
x = [1,2,3,4]
y = [10,50,23,100]

# plot
fig, ax = plt.subplots(nrows=2, ncols = 2)

ax.plot(x, y, linewidth=2.0)

plt.show()
```



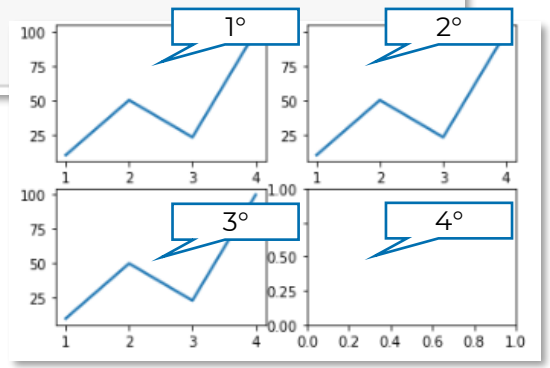
Agora cada quadrante é uma área de plotagem, então podemos fazer:

Para falar do primeiro elemento:

1 - Ax[**0**][**0**].plot(**x, y, linewidth=2.0**)
2 - Ax[**0**][**1**]. plot(**x, y, linewidth=2.0**)
3 - Ax[**1**][**0**].plot(**x, y, linewidth=2.0**)

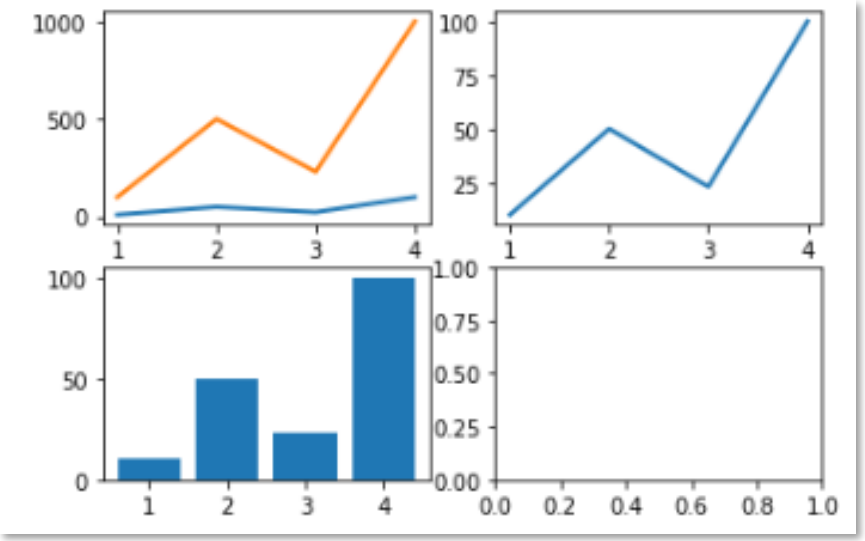
Dessa forma, é possível criar diferentes gráficos.

```
import matplotlib.pyplot as plt
import numpy as np

# make data
x = [1,2,3,4]
y = [10,50,23,100]

# plot
fig, ax = plt.subplots(nrows=2, ncols = 2)

ax[0][0].plot(x, y, linewidth=2.0)
ax[0][1].plot(x, y, linewidth=2.0)
ax[1][0].plot(x, y, linewidth=2.0)

plt.show()
```



Além disso, é possível criar diferentes gráficos e colocar 2 dados em um mesmo gráfico.

```
import matplotlib.pyplot as plt
import numpy as np

# make data
x = [1,2,3,4]
y = [10,50,23,100]
y2 = [100,500,230,1000]

# plot
fig, ax = plt.subplots(nrows=2, ncols = 2)

ax[0][0].plot(x, y, linewidth=2.0)
ax[0][0].plot(x, y2, linewidth=2.0)
ax[0][1].plot(x, y, linewidth=2.0)
ax[1][0].bar(x, y, linewidth=2.0)

plt.show()
```



Nós podemos mexer nos parâmetros também. Por exemplo, mudar a cor do eixo x para vermelho.

Ax.tick_params(**axis='x'**, **labelcolor='r'**)

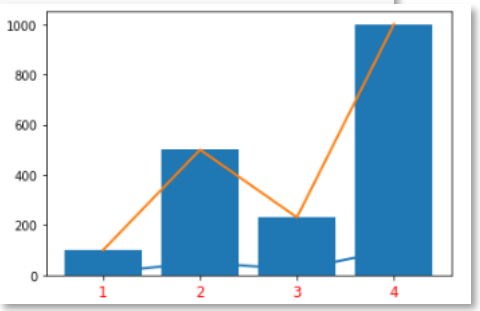
Se colocarmos apenas a função color estamos falando dos traços que acompanham o número.

Ax.tick_params(**axis='x'**, **color='r'**)

E para aumentar o tamanho do rótulo utilizamos a função **label**.

Ax.tick_params(**axis='x'**, **labelcolor='r', labelsize=12**)

```
import matplotlib.pyplot as plt
import numpy as np

# make data
x = [1,2,3,4]
y = [10,50,23,100]
y2 = [100,500,230,1000]

# plot
fig, ax = plt.subplots()

ax.plot(x, y, linewidth=2.0)
ax.plot(x, y2, linewidth=2.0)
ax.bar(x, y2, linewidth=2.0)

ax.set(xticks=np.arange(1, 5))

ax.tick_params(axis='x',labelcolor='r',labelsize=12)

plt.show()
```

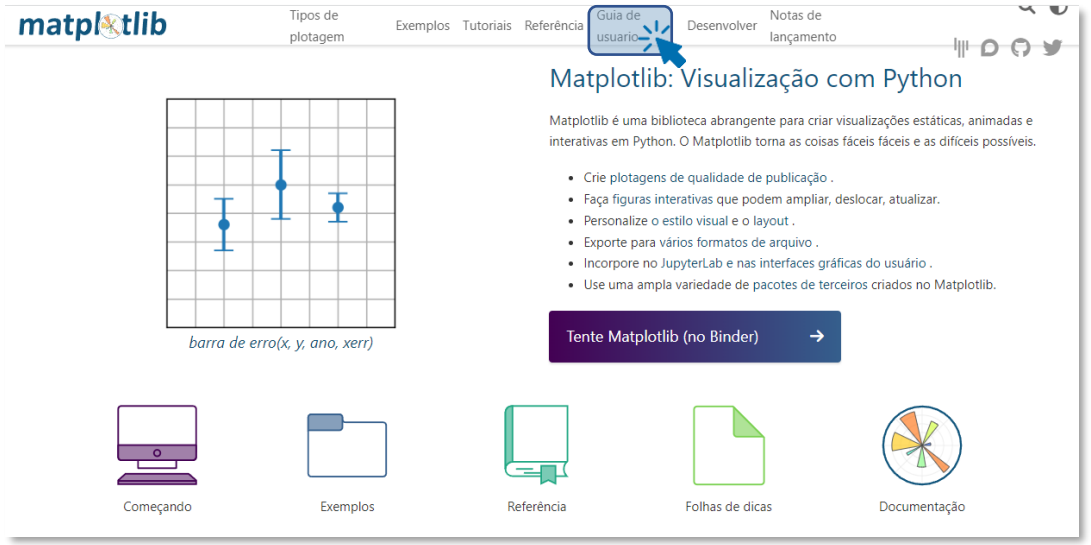


Módulo 7 – Introdução ao Matplotlib

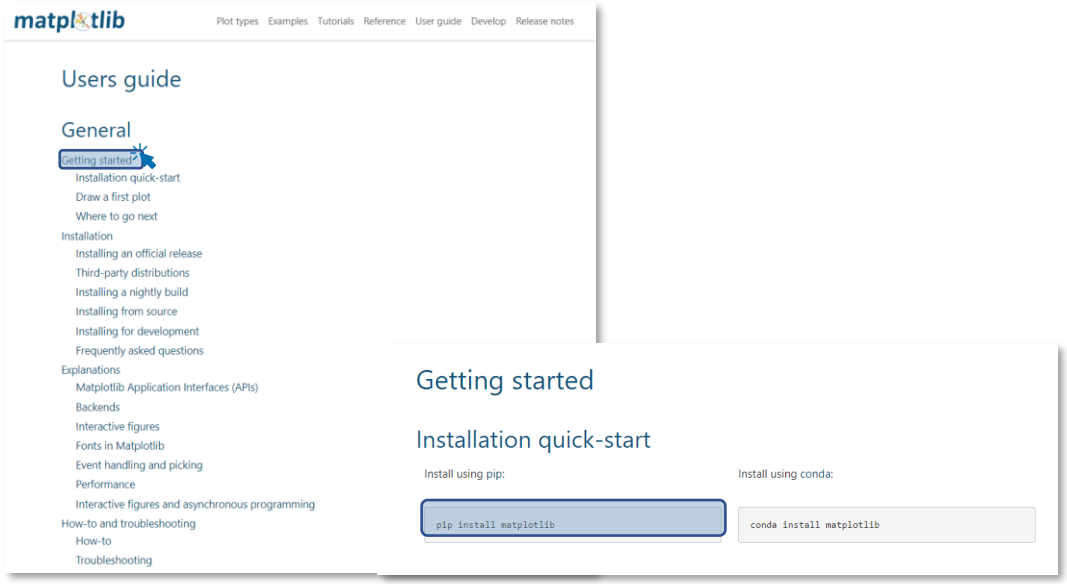
Existem várias bibliotecas no Python que nos ajudam a fazer gráficos, mas a Matplotlib é a mais utilizada.

A documentação do Matplotlib está disponível no link.

1

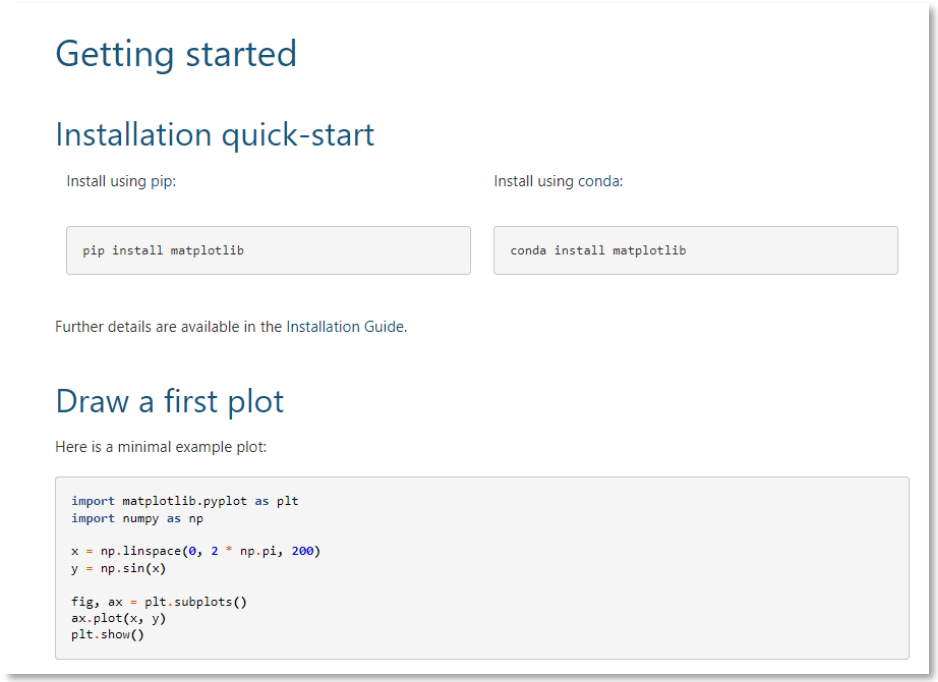


2



Módulo 7 – Introdução ao Matplotlib

Se você estiver utilizando o Jupyter Notebook ele já está instalado, caso contrário é necessário fazer o pip install matplotlib.



Getting started

Installation quick-start

Install using pip:

```
pip install matplotlib
```

Install using conda:

```
conda install matplotlib
```

Further details are available in the Installation Guide.

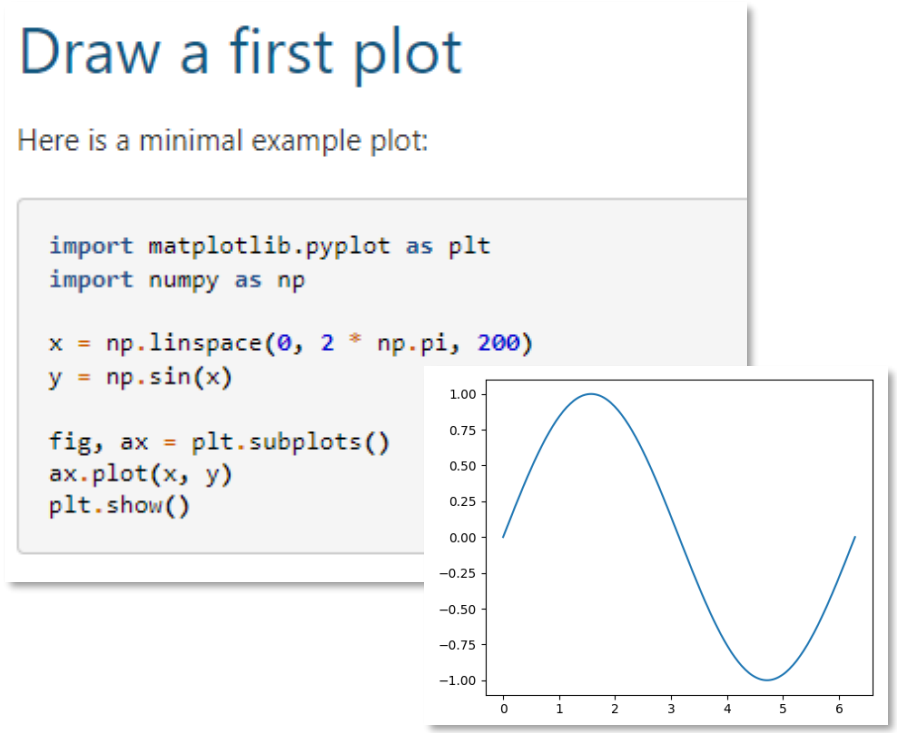
Draw a first plot

Here is a minimal example plot:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2 * np.pi, 200)
y = np.sin(x)

fig, ax = plt.subplots()
ax.plot(x, y)
plt.show()
```

Para fazer o primeiro plot utilizamos o código abaixo:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2 * np.pi, 200)
y = np.sin(x)

fig, ax = plt.subplots()
ax.plot(x, y)
plt.show()
```



O Matplotlib **permite desde a criação de visualizações** estáticas mais simples até os gráficos mais complexos e elaborados.

Como diz a própria documentação, ele torna o que é fácil, fácil e as coisas difíceis, possíveis de serem feitas!

Ela é bem **completa** e também é útil para **analytics**.

Não possui tanta customização quanto outras bibliotecas e **pode deixar a desejar no apelo estético em alguns casos**.

Módulo 7 – Introdução ao Matplotlib

O **Pyplot** é o módulo do Matplotlib que possui os **gráficos básicos** normalmente usados nessa biblioteca.

Vamos usar bastante o **pyplot** e, por isso, vamos importar diretamente esse módulo.

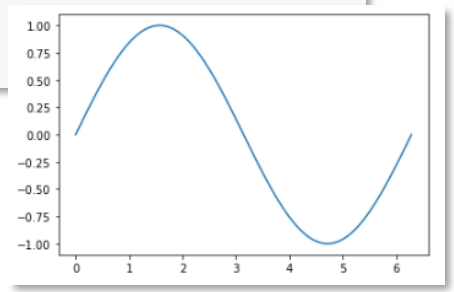
Então vamos copiar exatamente o código disponível na documentação:

import matplotlib.pyplot **as** plt
import numpy **as** np

x = **np.linspace**(0, 2 * np.pi, 200)
y = **np.sin**(x)

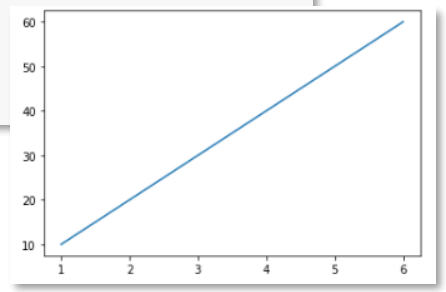
fig, ax = **plt.subplots**()
ax.plot(x, y)
plt.show()

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2 * np.pi, 200)
y = np.sin(x)

fig, ax = plt.subplots()
ax.plot(x, y)
plt.show()
```



Agora, vamos supor que nós não queremos utilizar o numpy, então podemos alterar os eixos para:

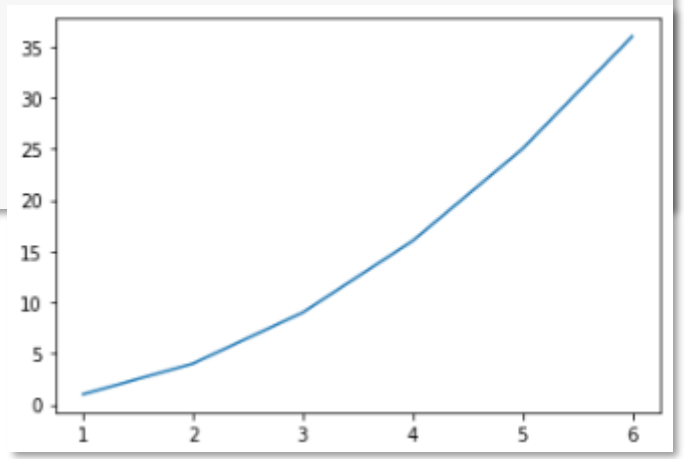
```
import matplotlib.pyplot as plt
import numpy as np

x =[1,2,3,4,5,6]
y =[10,20,30,40,50,60]

fig, ax = plt.subplots()
ax.plot(x, y)
plt.show()
```



Agora, vamos supor que temos uma função quadrática:

```
import matplotlib.pyplot as plt
import numpy as np

x =[1,2,3,4,5,6]
y =[1,4,9,16,25,36]

fig, ax = plt.subplots()
ax.plot(x, y)
plt.show()
```

```
# Criando nosso primeiro plot
# Disponível em: https://matplotlib.org/stable/users/getting_started/
import matplotlib.pyplot as plt

x = [1,2,3,4,5,6]
y = [1,4,9,16,25,36]

fig, ax = plt.subplots()
ax.plot(x, y)
plt.show()
```



Para esse módulo vamos utilizar a nossa base do Projeto do Intagram.

Para isso vamos importar as bibliotecas utilizando a estrutura abaixo:

Import biblioteca as apelido

```
# Importando o pyplot
import matplotlib.pyplot as plt

# Importando as outras bibliotecas
import numpy as np
import pandas as pd
import datetime as dt
```

Vamos importar e tratar a nossa base em Excel.

Para importar a base utilizamos:

Base = **pd.read_tipodearquivo**(‘**Nome do arquivo**’.**tipodoarquivo**)

```
# Importar a base em excel
base = pd.read_excel("08. Analisando o engajamento no Instagram.xlsx")
# Apagando a coluna "Visualizações"
base = base.drop("Visualizações",axis=1)
# Agora vamos atribuir o valor N para essa coluna
base.loc[base.Carrossel.isnull(),"Carrossel"] = "N"
```

Nome da base

Tipo de arquivo

Para apagar uma coluna utilizamos:

base = **base.drop**(‘**Coluna**’,axil=1)

E para atribuir um valor N para essa coluna utilizamos a estrutura abaixo:

base.loc[**base**. **Coluna**.**isnull**(),"**Coluna**"] = "N"

```
# Apagando a coluna "Visualizações"
base = base.drop("Visualizações",axis=1)

# Agora vamos atribuir o valor N para essa coluna
base.loc[base.Carrossel.isnull(),"Carrossel"] = "N"
```

```
base.head()
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes
0	Foto	2021-09-11	2858	16	Loja	N	N	N	2874
1	Foto	2021-09-11	2930	28	Loja/Produtos	N	N	N	2958
2	Foto	2021-09-11	2807	9	Loja	N	N	N	2816
3	Vídeo	2021-09-12	5115	49	Produtos	N	N	N	5164
4	Foto	2021-09-13	4392	45	Produtos	S	N	N	4437

Para visualizarmos as primeiras 5 linhas utilizamos o comando **head**.

Base.head()

Módulo 7 – Usando a documentação para criar nosso primeiro gráfico (gráfico de linha)

Agora que já temos a nossa base importada vamos começar a fazer os plots.

Vamos começar com um exemplo da documentação:

```
import matplotlib.pyplot as plt
import numpy as np

# make data
x = np.linspace(0, 10, 100)
y = 4 + 2 * np.sin(2 * x)

# plot
fig, ax = plt.subplots()

ax.plot(x, y,'r--', linewidth=5.0)

ax.set(xlim=(0,8),xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()
```

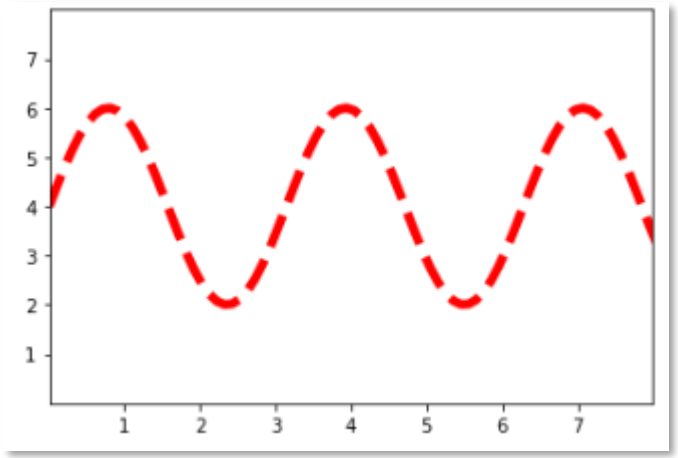
Importando as bibliotecas

Criando dados

Tipo de gráfico

Limites dos eixos

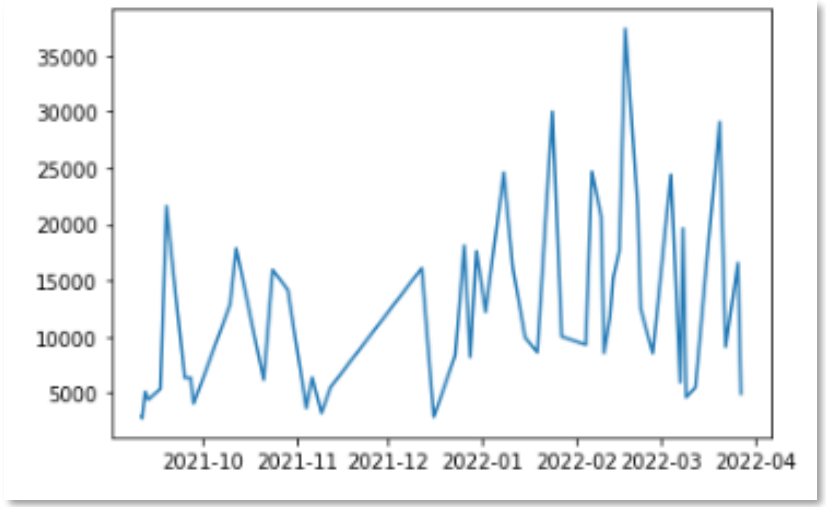
Comando para mostrar o gráfico



Módulo 7 – Usando a documentação para criar nosso primeiro gráfico (gráfico de linha)

Agora vamos aplicar esse mesmo gráfico para a nossa base.

```
x = base.Data
y = base.Curtidas

fig, ax = plt.subplots()

ax.plot(x, y)

plt.show()
```



Módulo 7 – Usando a documentação para criar nosso primeiro gráfico (gráfico de linha)

E se quiséssemos mostrar esse mesmo gráfico só que no modelo de dispersão? O que mudaria?

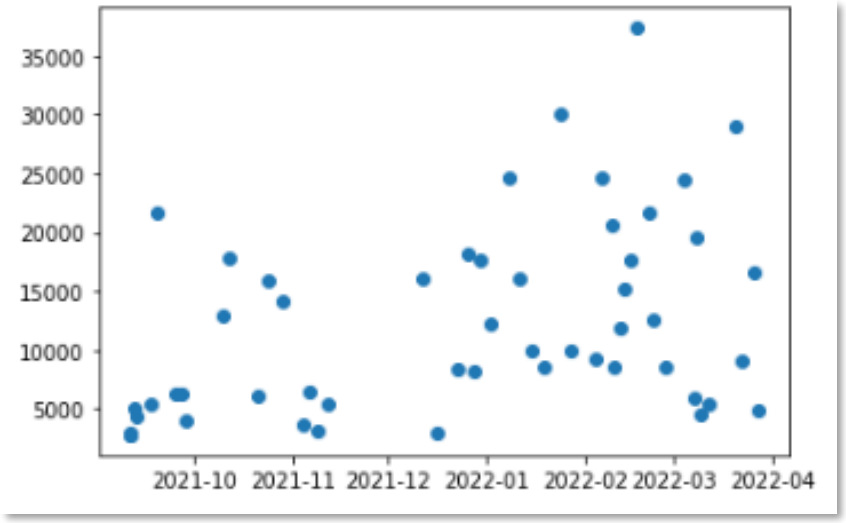
Trocaríamos apenas o **ax.plot** por **ax.scatter**:

x = **base**.Data
y = **base**.Curtidas

fig, ax = **plt.subplots**()

ax.scatter(**x**, **y**)

plt.show()



Atenção!
Confira na documentação os **argumentos** de cada gráfico!

Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

Agora vamos entender mais um pouco sobre como utilizar a documentação do Matplotlib.

Caso você tenha dificuldade em inglês, existe a opção de traduzir a página.

1) Clique com o botão direito.

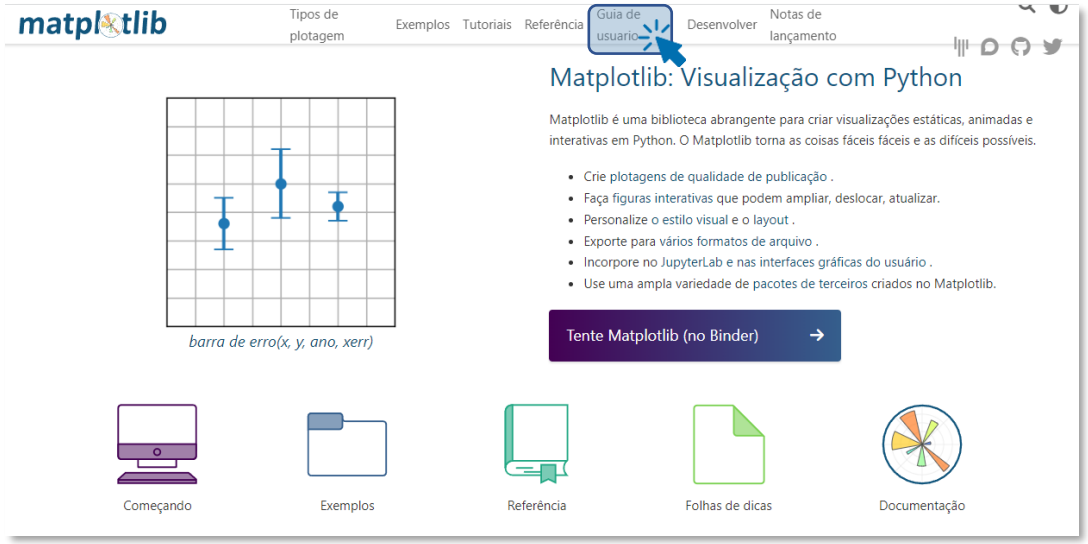
2) Clique em traduzir para o português.

Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

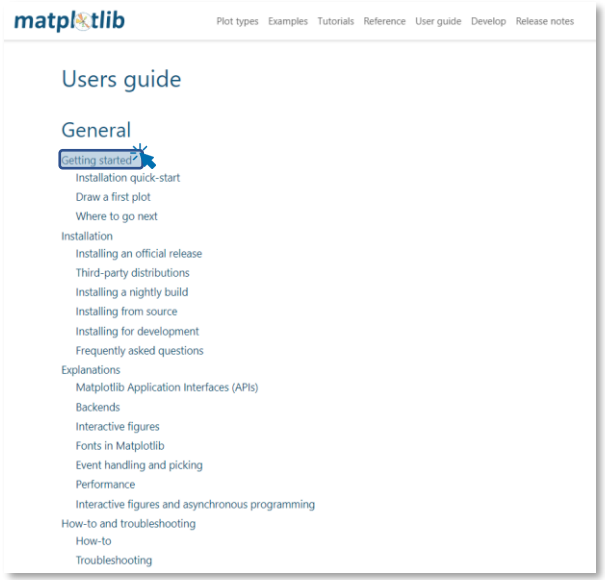
Como podemos começar a utilizar a biblioteca do Matplotlib?

Clicamos em ‘Guia do usuário’ e em seguida: General > Getting started.

1

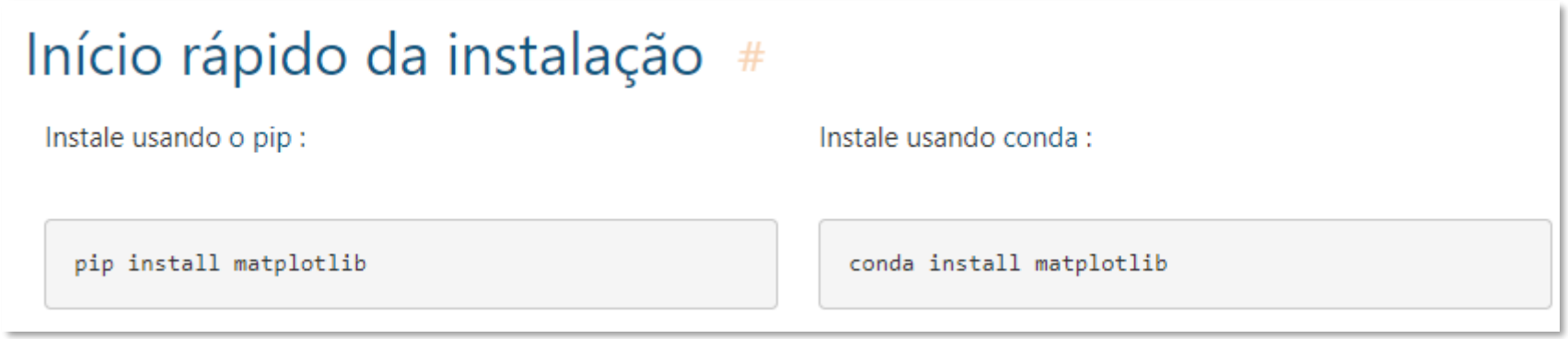


2



Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

Se você estiver utilizando o Anaconda ele já está instalado, caso contrário é necessário fazer o pip install matplotlib.



Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

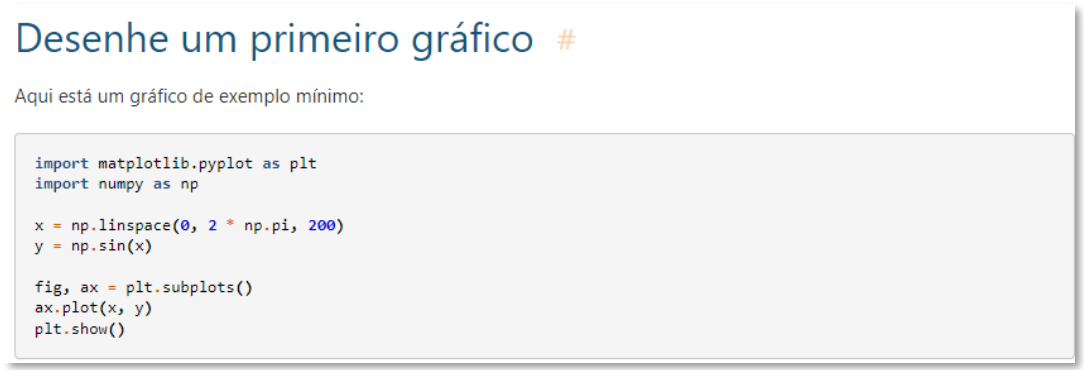
Então vamos entender linha a linha o primeiro gráfico que copiamos da documentação:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2 * np.pi, 200)
y = np.sin(x)

fig, ax = plt.subplots()
ax.plot(x, y)
plt.show()
```

Importando as bibliotecas

Criandos os dados

Fazendo o plot

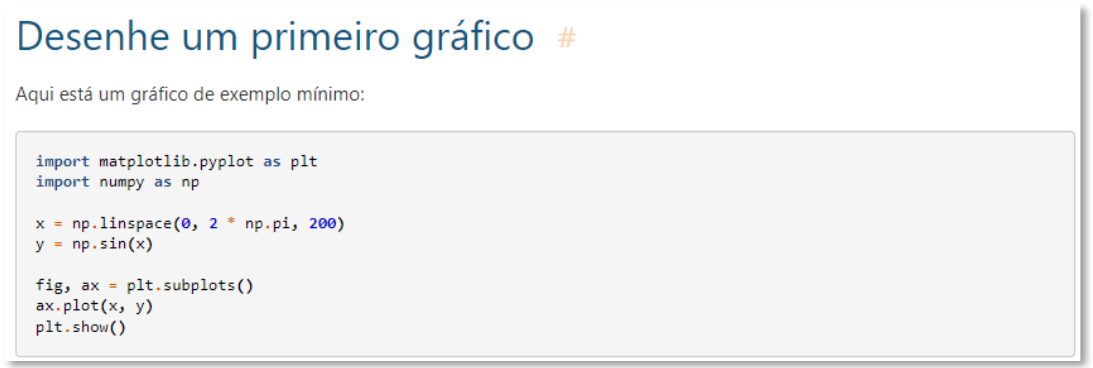


Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

Vamos substituir os nossos dados por números para facilitar:

```
import matplotlib.pyplot as plt

x = [1,2,3,4,5]
y = [1,4,17,50,92]

fig, ax = plt.subplots()
ax.plot(x, y)
plt.show()
```

Nossos dados



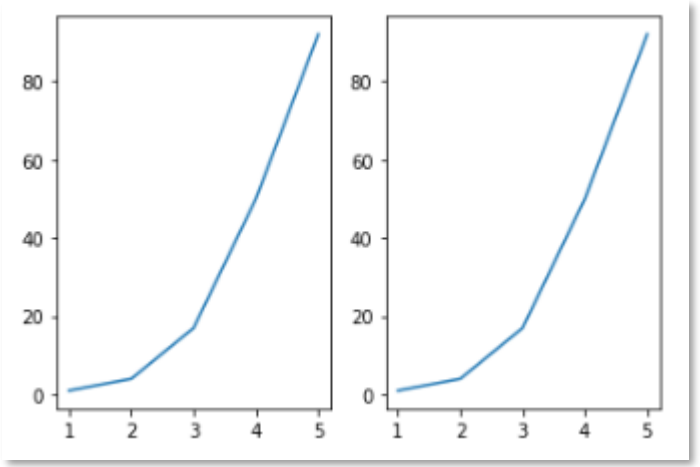
Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

Agora vamos entender a parte do gráfico.

O comando abaixo cria uma figura, então com ele nós criamos 2 imagens diferentes dentro da nossa figura.

```
import matplotlib.pyplot as plt

x = [1,2,3,4,5]
y = [1,4,17,50,92]

fig, ax = plt.subplots(nrows=1,ncols=2)
Ax[0].plot(x, y)
Ax[1].plot(x, y)

plt.show()
```

Cada um dos gráficos que estamos criando

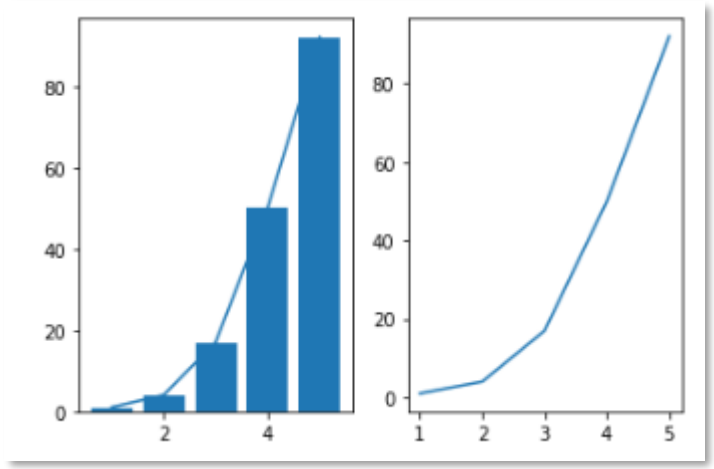


Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

Podemos colocar mais de um gráfico em uma coluna, por exemplo:

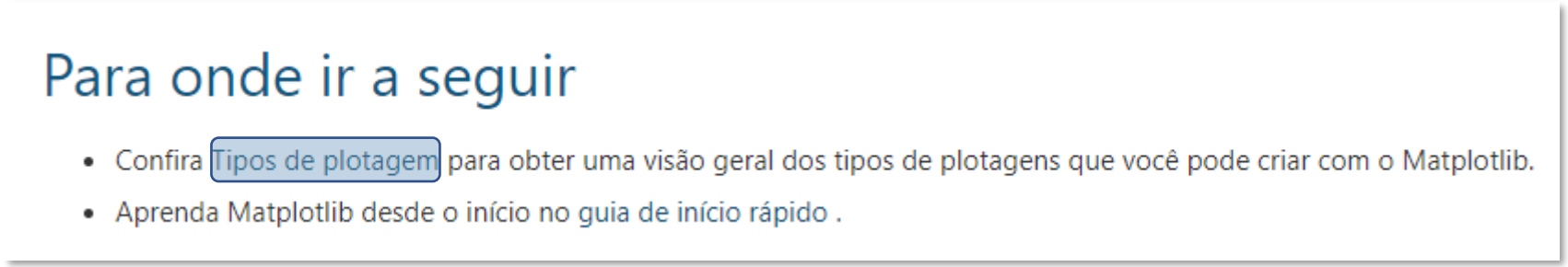
```
import matplotlib.pyplot as plt

x = [1,2,3,4,5]
y = [1,4,17,50,92]

fig, ax = plt.subplots(nrows=1,ncols=2)

ax[0].plot(x, y)
ax[0].bar(x, y)
ax[1].plot(x, y)

plt.show()
```

Cada um dos gráficos que estamos criando



Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

Agora vamos olhar os tipos de gráficos clicando em **Tipos de plotagem**.

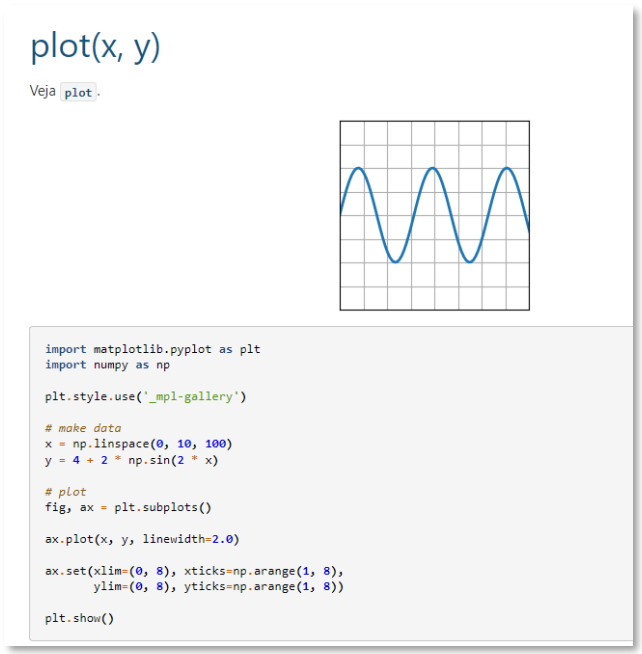


Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

Agora vamos clicar no gráfico de linha para entendermos como ele funciona.

1

2



Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

Copiando o código da documentação temos:

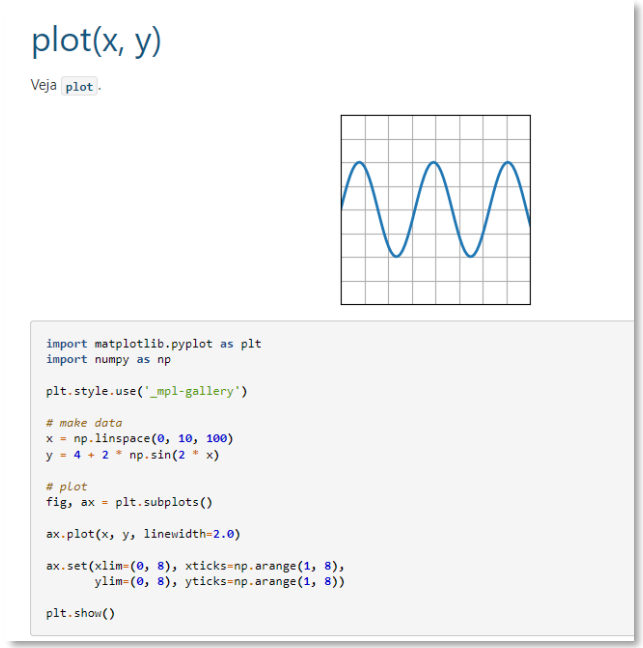
```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 10, 100)
y = 4 + 2 * np.sin(2 * x)
fig, ax = plt.subplots()

ax.plot(x, y, linewidth=2.0)

ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))
plt.show()
```

Definindo os valores de X

Definindo os valores de Y

Espessura da linha

Limites dos eixos

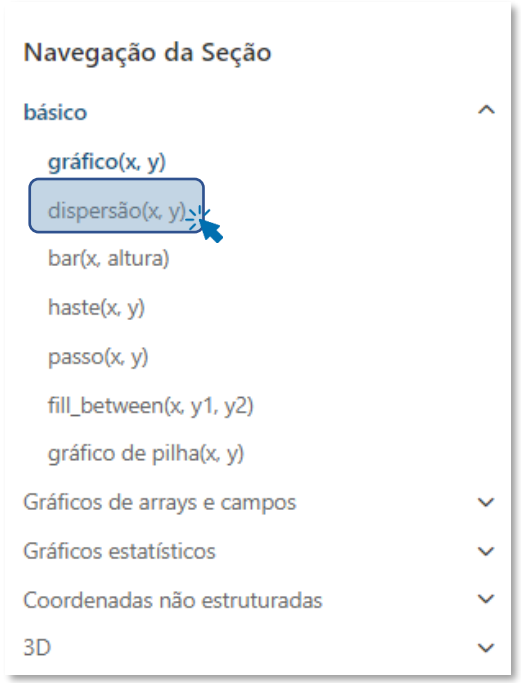


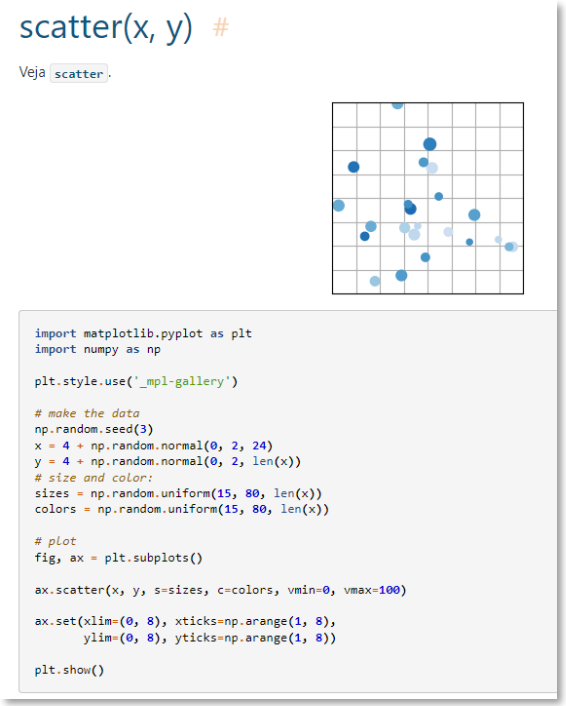
Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

Para aprender mais sobre um argumento, podemos pesquisar na documentação

Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

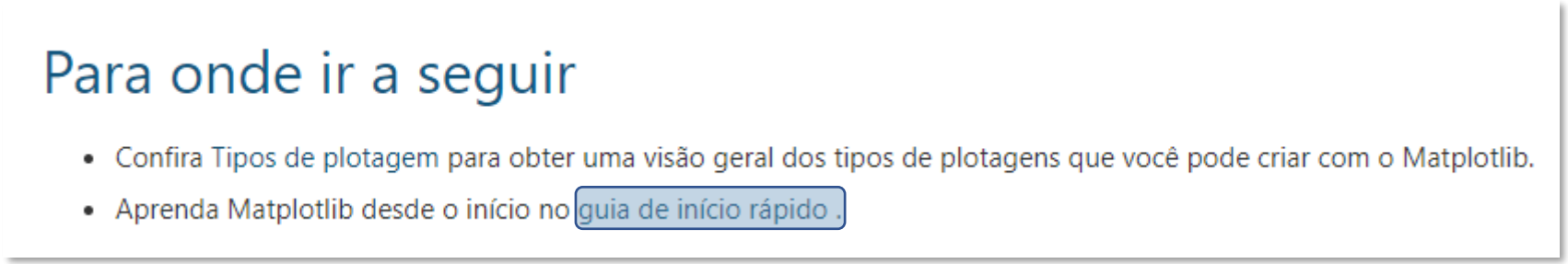
Podemos aplicar isso para outros tipos de gráficos também e vamos notar que o código para o gráfico fica parecido.





Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

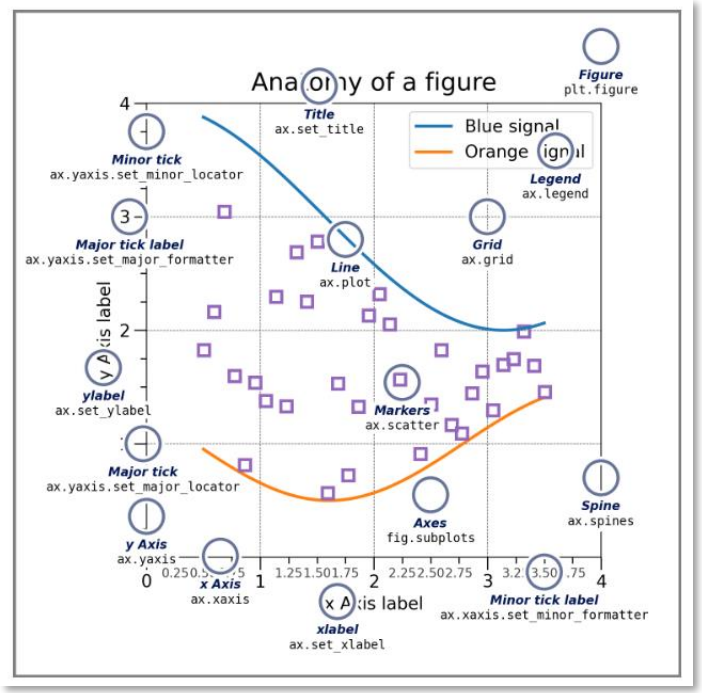
Agora vamos voltar para a aba *Getting started* e vamos clicar em Guia do início rápido.



Módulo 7 – (Opcional) Entendo a documentação do Matplotlib

Nessa opção, além de ter a forma e um exemplo simples, temos a parte da figura.

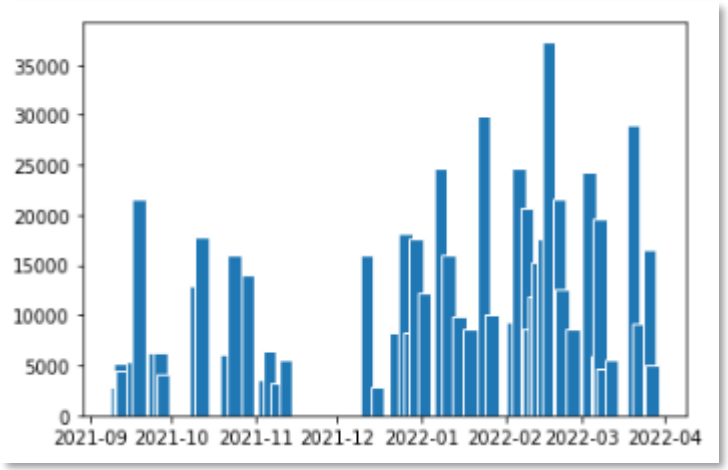
A parte da figura mostra como mexer na linha, bordas, etc.



Módulo 7 – Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)

Os gráficos vão nos ajudar a entender se existe uma relação entre os dados, comportamentos que podem prever o futuro.

A parte da figura mostra como mexer na linha, bordas, etc.



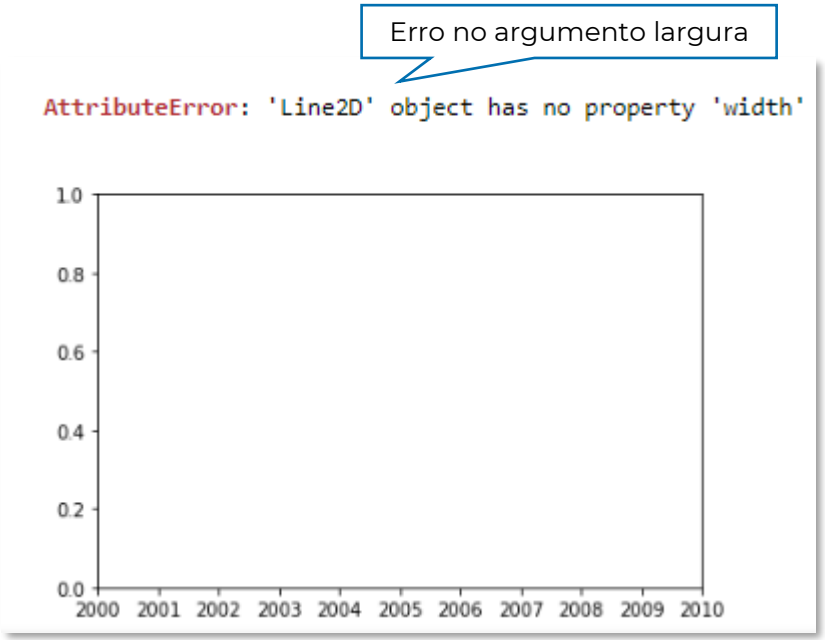
Módulo 7 – Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)

Agora repare que ao trocar o gráfico de barra para um de linha ocorre um erro no nosso código.

Para resolvermos esse problema, basta apagar o argumento **width**.

```
# make data
x = base.Data
y = base.Curtidas

# plot
fig, ax = plt.subplots()

ax.plot(x, y,width=5, edgecolor='white')

plt.show()
```

Erro no argumento largura



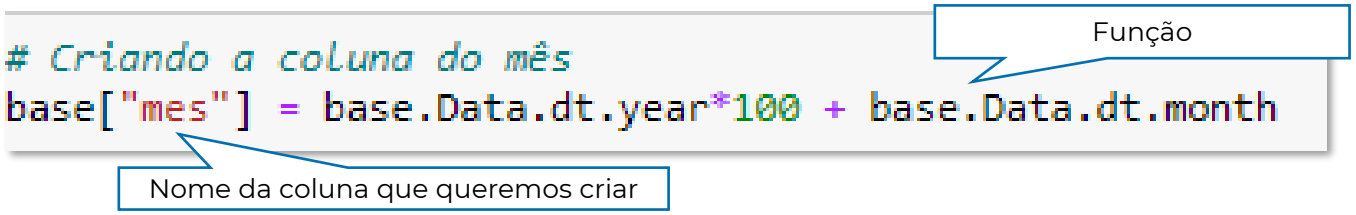
Módulo 7 – Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)

Será que existe uma tendência de queda nas curtidas?

- Vamos fazer uma análise mensal
  - Para isso precisamos criar a coluna mês/ano:

Criando a coluna do mês:

base[“**coluna_que_queremos_criar**"] = **função_que_vai_criar_essa_coluna**



Módulo 7 – Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)

Vamos importar uma biblioteca que vai nos ajudar a trabalhar com datas: **datetime**.

Antes de mais nada, utilizamos o comando import datetime as dt, conforme mostrado na imagem abaixo.

```
import datetime as dt
```

Módulo 7 – Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)

Em seguida, complementamos com o seguinte código:

```
import datetime as dt
```

```
# Criando a coluna do mês
base["mes"] = base.Data.dt.year*100 + base.Data.dt.month

# Verificando a coluna criada
base.head()
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes	mes
0	Foto	2021-09-11	2858	16	Loja	N	N	N	2874	202109
1	Foto	2021-09-11	2930	28	Loja/Produtos	N	N	N	2958	202109
2	Foto	2021-09-11	2807	9	Loja	N	N	N	2816	202109
3	Vídeo	2021-09-12	5115	49	Produtos	N	N	N	5164	202109
4	Foto	2021-09-13	4392	45	Produtos	S	N	N	4437	202109

Módulo 7 – Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)

Agora como poderíamos colocar essa tabela na forma de um gráfico?

Vamos utilizar o comando **groupby** para somar os valores do mês.

Relembrando o groupby:

Variável = base.groupby("**Coluna**")["**Coluna_que_queremos_agrupar**"].função()

```
media_mensal = base.groupby("mes")["Curtidas"].mean()
print(media_mensal)
```

```
mes
202109     6181.500000
202110    13390.400000
202111     4682.000000
202112    11863.833333
202201    15913.285714
202202    17081.363636
202203    13299.555556
Name: Curtidas, dtype: float64
```

Módulo 7 – Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)

Se a gente quiser o tipo da média mensal utilizando o comando type, vamos observar que ele é uma série, que possui 2 elementos: índice e os valores.

O índice vai ser o mês e os valores vão ser as médias.

E podemos usar exatamente essas informações para criar o nosso gráfico.

```
type(media_mensal)
```

```
pandas.core.series.Series
```

```
# O groupby vai ter o index
media_mensal.index
```

```
Int64Index([202109, 202110, 202111, 202112, 202201, 202202, 202203], dtype='int64', name='mes')
```

```
# E os valores
media_mensal.values
```

```
array([ 6181.5       , 13390.4       ,  4682.        , 11863.83333333,
       15913.28571429, 17081.36363636, 13299.55555556])
```

Módulo 7 – Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)

Então vamos utilizar o nosso código e substituir os valores de x e y:

x = media_mensal.index
y = media_mensal.values

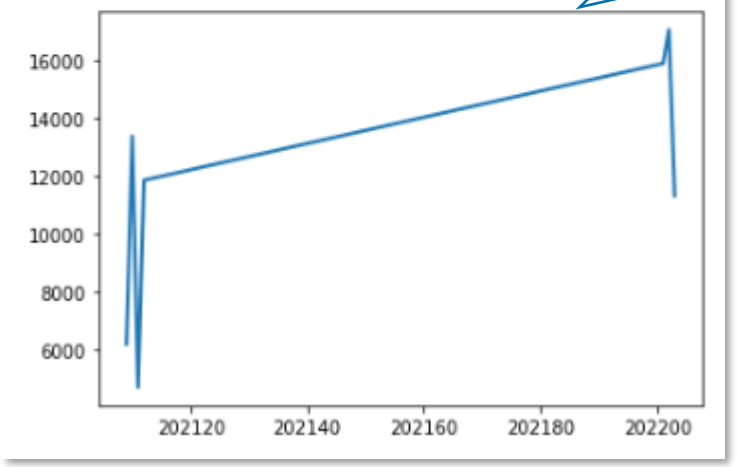
fig, ax = **plt.subplots**()

ax.plot(**x**, **y**, linewidth=2.0)

plt.show()

```
# make data
x = media_mensal.index
y = media_mensal.values

# plot
fig, ax = plt.subplots()

ax.plot(x, y, linewidth=2.0)

plt.show()
```

Repara que o nosso gráfico ficou estranho!



Módulo 7 – Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)

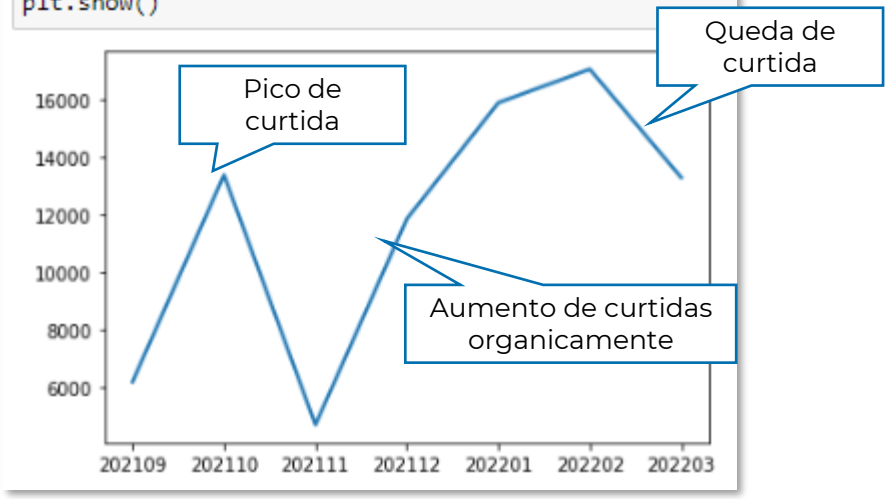
Vamos entender o motivo do gráfico ter ficado assim, se observarmos os valores do índice vamos notar que existe um gap entre eles.

Como podemos corrigir?

Considerar o índice como um **texto** e não um inteiro.

Agora conseguimos mostrar de forma visual as curtidas de cada mês.

```
# make data
x = media_mensal.index.astype(str)
y = media_mensal.values

# plot
fig, ax = plt.subplots()

ax.plot(x, y, linewidth=2.0)

plt.show()
```



Módulo 7 – Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)

Como já vimos a média é interferida pelos outliers, então vamos aplicar novos dados nesse gráfico:

minimo = **base**.**groupby**("**mes**")["**Curtidas**"].**min**()
maximo = **base**.**groupby**("**mes**")["**Curtidas**"].**max**()

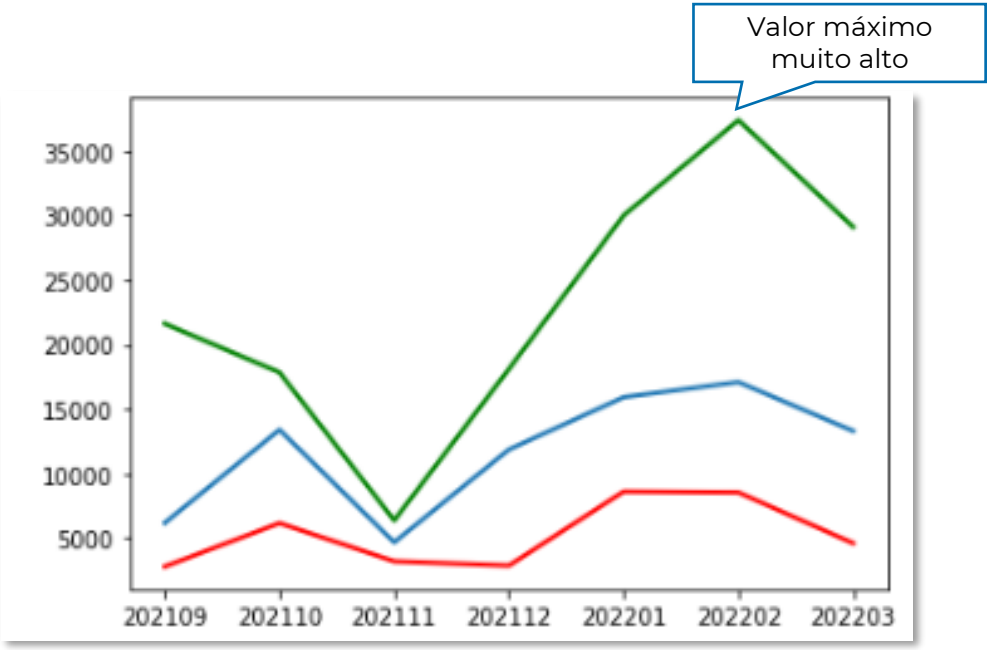
```
# Fazendo para o mínimo e máximo de curtidas
minimo = base.groupby("mes")["Curtidas"].min()
maximo = base.groupby("mes")["Curtidas"].max()
```

Módulo 7 – Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)

Plotando esse gráfico:

```
x = media_mensal.index.astype(str)
y = media_mensal.values
x1 = minimo.index.astype(str)
y1 = minimo.values
x2 = maximo.index.astype(str)
y2 = maximo.values

fig, ax = plt.subplots()
ax.plot(x, y, linewidth=2.0)
ax.plot(x1, y1, linewidth=2.0, color="r")
ax.plot(x2, y2, linewidth=2.0, color='g')
plt.show()
```

Argumento da cor



Módulo 7 – Usando gráficos (de linha) para entender os dados (máximo, mínimo e média mensal de curtidas)

Agora vamos fazer uma visualização ordenando por mês e curtidas para entender nosso outlier:

```
base[(base.mes >= 202202)].sort_values(["mes","Curtidas"],ascending=False)
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes	mes
48	Foto	2022-03-20	29084	479	Datas comemorativas/Promoções	S	S	N	29563	202203
43	Foto	2022-03-04	24399	266	NaN	S	S	N	24665	202203
45	Reels	2022-03-08	19621	275	Trends	S	S	N	19896	202203
50	Foto	2022-03-26	16551	186	NaN	S	N	N	16737	202203
49	Foto	2022-03-22	9087	106	NaN	S	S	N	9193	202203
44	IGTV	2022-03-07	5918	116	Dicas de como usar/Novos Produtos	S	N	N	6034	202203
47	IGTV	2022-03-12	5489	77	Dicas de como usar/Novos Produtos	S	N	N	5566	202203
51	IGTV	2022-03-27	4934	65	Dicas de como usar/Produtos	S	N	N	4999	202203
46	Foto	2022-03-09	4613	50	Influenciadores	S	N	N	4663	202203
39	Foto	2022-02-17	37351	502	Promoções	S	S	N	37853	202202
33	Foto	2022-02-06	24655	186	Influenciadores	S	S	N	24841	202202
40	Foto	2022-02-21	21621	213	Influenciadores	S	S	S	21834	202202
34	Foto	2022-02-09	20660	292	Influenciadores	S	S	N	20952	202202
38	Foto	2022-02-15	17687	213	NaN	S	N	N	17900	202202
37	Vídeo	2022-02-13	15219	357	Datas comemorativas	S	S	N	15576	202202
41	Foto	2022-02-22	12530	90	NaN	S	N	N	12620	202202
36	Foto	2022-02-12	11802	102	Produtos	S	S	N	11904	202202
32	IGTV	2022-02-04	9270	222	Dicas de como usar/Produtos	S	N	N	9492	202202
35	IGTV	2022-02-10	8556	188	Dicas de como usar/Produtos	S	N	N	8744	202202
42	Foto	2022-02-26	8544	72	Influenciadores	S	S	N	8616	202202

Nosso outlier foi uma **promoção**

Módulo 7 – Filtrando a base usando o contains (e fillna para tratar valores vazios)

Notamos na aula passada que em uma publicação de promoção foi onde tivemos o maior número de curtidas.

Mas será que essa foi a única promoção que a gente fez?

Para verificar se tivemos outras promoções vamos utilizar o **contains**.

```
base[(base.mes >= 202202)].sort_values(["mes","Curtidas"],ascending=False)
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes	mes
48	Foto	2022-03-20	29084	479	Datas comemorativas/Promoções	S	S	N	29563	202203
43	Foto	2022-03-04	24399	266	NaN	S	S	N	24665	202203
45	Reels	2022-03-08	19621	275	Trends	S	S	N	19896	202203
50	Foto	2022-03-26	16551	186	NaN	S	N	N	16737	202203
49	Foto	2022-03-22	9087	106	NaN	S	S	N	9193	202203
44	IGTV	2022-03-07	5918	116	Dicas de como usar/Novos Produtos	S	N	N	6034	202203
47	IGTV	2022-03-12	5489	77	Dicas de como usar/Novos Produtos	S	N	N	5566	202203
51	IGTV	2022-03-27	4934	65	Dicas de como usar/Produtos	S	N	N	4999	202203
46	Foto	2022-03-09	4613	50	Influenciadores	S	N	N	4663	202203
39	Foto	2022-02-17	37351	502	Promoções	S	S	N	37853	202202
33	Foto	2022-02-06	24655	186	Influenciadores	S	S	N	24841	202202
40	Foto	2022-02-21	21621	213	Influenciadores	S	S	S	21834	202202
34	Foto	2022-02-09	20660	292	Influenciadores	S	S	N	20952	202202
38	Foto	2022-02-15	17687	213	NaN	S	N	N	17900	202202
37	Vídeo	2022-02-13	15219	357	Datas comemorativas	S	S	N	15576	202202
41	Foto	2022-02-22	12530	90	NaN	S	N	N	12620	202202
36	Foto	2022-02-12	11802	102	Produtos	S	S	N	11904	202202
32	IGTV	2022-02-04	9270	222	Dicas de como usar/Produtos	S	N	N	9492	202202
35	IGTV	2022-02-10	8556	188	Dicas de como usar/Produtos	S	N	N	8744	202202
42	Foto	2022-02-26	8544	72	Influenciadores	S	S	N	8616	202202

Módulo 7 – Filtrando a base usando o contains (e fillna para tratar valores vazios)

Nós vamos utilizar o **contains** para verificar se existe uma palavra dentro de um texto.

Porém, o **contains** **não aceita se tiver valores NaN**, então precisamos tratar isso antes

```
# Tentando usar o contains sem tratar os valores NaN
base[base.Tags.str.contains("Promoções")]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [25], in <cell line: 2>()
      1 # Tentando usar o contains sem tratar os valores NaN
----> 2 base[base.Tags.str.contains("Promoções")]

File ~\anaconda3\lib\site-packages\pandas\core\frame.py:3495, in DataFrame.__getitem__(self, key)
   3492     return self.where(key)
   3494 # Do we have a (boolean) 1d indexer?
-> 3495 if com.is_bool_indexer(key):
   3496     return self._getitem_bool_array(key)
   3498 # We are left with two options: a single key, and a collection of keys,
   3499 # We interpret tuples as collections only for non-MultiIndex

File ~\anaconda3\lib\site-packages\pandas\core\common.py:144, in is_bool_indexer(key)
    140     na_msg = "Cannot mask with non-boolean array containing NA / NaN values"
    141     if lib.infer_dtype(key) == "boolean" and isna(key).any():
    142         # Don't raise on e.g. ["A", "B", np.nan], see
    143         #  test_loc_getitem_list_of_labels_categoricalindex_with_na
--> 144         raise ValueError(na_msg)
    145     return False
    146 return True

ValueError: Cannot mask with non-boolean array containing NA / NaN values
```

Não aceita se tiver valores **nulos**

Módulo 7 – Filtrando a base usando o contains (e fillna para tratar valores vazios)

Para corrigir os valores nulos vamos utilizar o **.fillna**() que vai substituir os valores NaN pelo novo valor que passarmos.

base.**fillna**("**novo_valor**")

Então vamos substituir os valores nulos para vazios:

base.**fillna**("")

```
# O `.fillna()` vai substituir os valores NaN pelo novo valor que passarmos
base.fillna("").tail()
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes	mes
47	IGTV	2022-03-12	5489	77	Dicas de como usar/Novos Produtos	S	N	N	5566	202203
48	Foto	2022-03-20	29084	479	Datas comemorativas/Promoções	S	S	N	29563	202203
49	Foto	2022-03-22	9087	106		S	S	N	9193	202203
50	Foto	2022-03-26	16551	186		S	N	N	16737	202203
51	IGTV	2022-03-27	4934	65	Dicas de como usar/Produtos	S	N	N	4999	202203

Vazio

Módulo 7 – Filtrando a base usando o contains (e fillna para tratar valores vazios)

Então podemos usar o vazio apenas para visualizar todas as tags que possuem marcação de patrocinado.

```
base[base.fillna("").Tags.str.contains("Promoções")]
```

Então, podemos observar que temos 3 resultados que contém promoções

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes	mes
13	Foto	2021-10-24	15940	612	Promoções	S	N	N	16552	202110
39	Foto	2022-02-17	37351	502	Promoções	S	S	N	37853	202202
48	Foto	2022-03-20	29084	479	Datas comemorativas/Promoções	S	S	N	29563	202203

Módulo 7 – Filtrando a base usando o contains (e fillna para tratar valores vazios)

Agora vamos excluir da nossa base as linhas de promoções:

```
base = base.drop([13,39,48],axis=0)
```

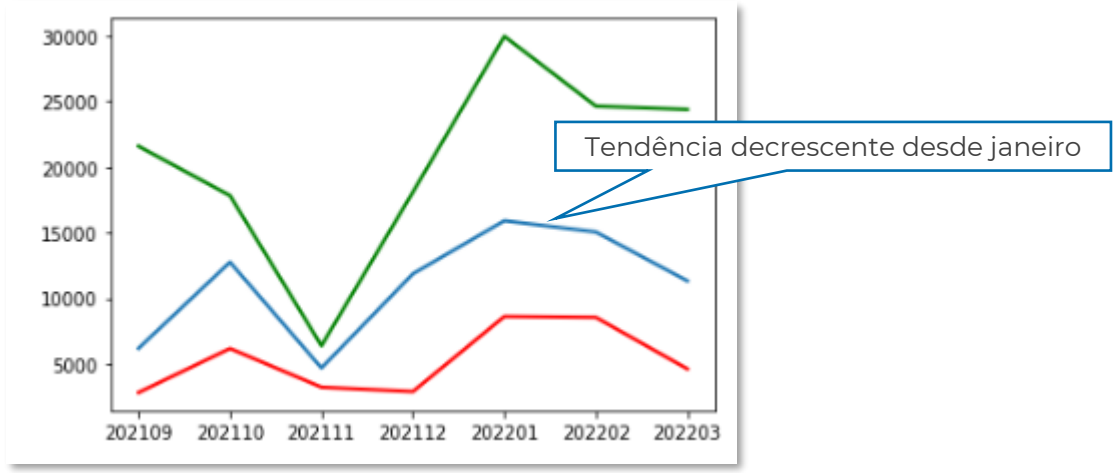
Linhas

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes	mes
13	Foto	2021-10-24	15940	612	Promoções	S	N	N	16552	202110
39	Foto	2022-02-17	37351	502	Promoções	S	S	N	37853	202202
48	Foto	2022-03-20	29084	479	Datas comemorativas/Promoções	S	S	N	29563	202203

Módulo 7 – Filtrando a base usando o contains (e fillna para tratar valores vazios)

Refazendo os cálculos pra base sem as promoções e refazendo o plot:

```
media_mensal = base.groupby("mes")["Curtidas"].mean()
minimo = base.groupby("mes")["Curtidas"].min()
maximo = base.groupby("mes")["Curtidas"].max()

# make data
x = media_mensal.index.astype(str)
y = media_mensal.values
x1 = minimo.index.astype(str)
y1 = minimo.values
x2 = maximo.index.astype(str)
y2 = maximo.values

# plot
fig, ax = plt.subplots()

ax.plot(x, y, linewidth=2.0)
ax.plot(x1, y1, linewidth=2.0, color="r")
ax.plot(x2, y2, linewidth=2.0, color='g')


plt.show()
```



Módulo 7 – Criando e ajustando o visual (rotacionando o eixo x) de um gráfico de barras

Como podemos visualizar as tags que mais impactaram o negócio?

Para isso vamos precisar fazer o tratamento da nossa base e separar a linha de tag em duas, utilizando o **split**.

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes	mes
0	Foto	2021-09-11	2858	16	Loja	N	N	N	2874	202109
1	Foto	2021-09-11	2930	28	Loja/Produtos	N	N	N	2958	202109
2	Foto	2021-09-11	2807	9	Loja	N	N	N	2816	202109
3	Vídeo	2021-09-12	5115	49	Produtos	N	N	N	5164	202109
4	Foto	2021-09-13	4392	45	Produtos	S	N	N	4437	202109

Módulo 7 – Criando e ajustando o visual (rotacionando o eixo x) de um gráfico de barras

Separando a linha em 2:

baseTags = **base**

baseTags.Tags = **baseTags**.Tags.**str.split**("/")

Elemento de separação

baseTags = **baseTags**.**explode**('Tags')

baseTags.**head**()

```
baseTags = base
baseTags.Tags = baseTags.Tags.str.split("/")
baseTags = baseTags.explode('Tags')
baseTags.head()
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes	mes
0	Foto	2021-09-11	2858	16	Loja	N	N	N	2874	202109
1	Foto	2021-09-11	2930	28	Loja	N	N	N	2958	202109
1	Foto	2021-09-11	2930	28	Produtos	N	N	N	2958	202109
2	Foto	2021-09-11	2807	9	Loja	N	N	N	2816	202109
3	Vídeo	2021-09-12	5115	49	Produtos	N	N	N	5164	202109

Módulo 7 – Criando e ajustando o visual (rotacionando o eixo x) de um gráfico de barras

Vamos agrupar Curtidas por Tags.

```
baseTags.groupby("Tags")["Curtidas"].mean()
```

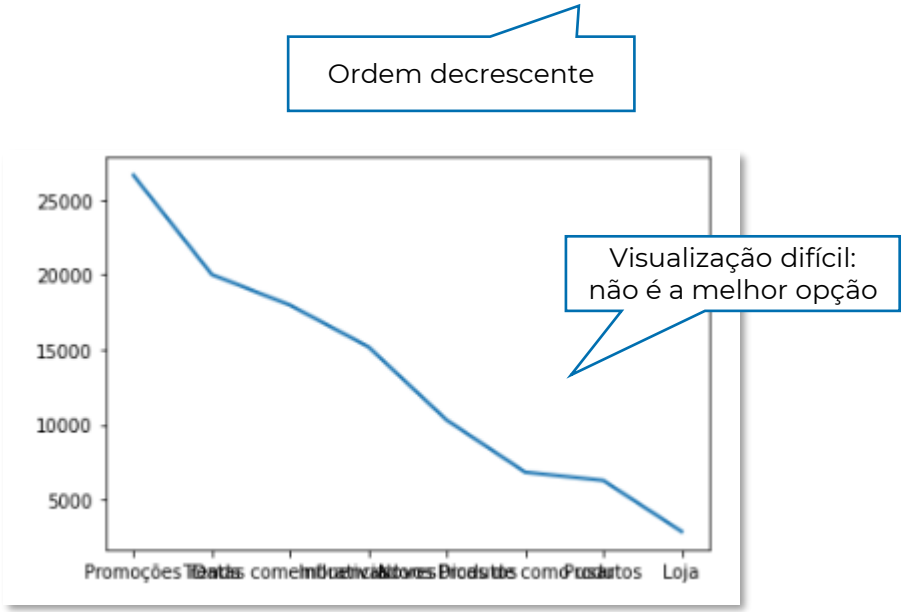
Só que a visualização dessa tabela é difícil para o nosso cliente.

```
Tags
Datas comemorativas     17975.000000
Dicas de como usar       6833.400000
Influenciadores         15197.285714
Loja                     2865.000000
Novos Produtos          10304.888889
Produtos                 6269.823529
Promoções               26645.500000
Trends                  20024.000000
Name: Curtidas, dtype: float64
```

Módulo 7 – Criando e ajustando o visual (rotacionando o eixo x) de um gráfico de barras

Então vamos colocar em um gráfico para uma melhor visualização:

```
media = baseTags.groupby ("Tags")["Curtidas"].mean() .sort_values(ascending=False)

x = media.index
y = media.values

fig, ax = plt.subplots()

ax.plot(x, y, linewidth=2.0)

plt.show()
```

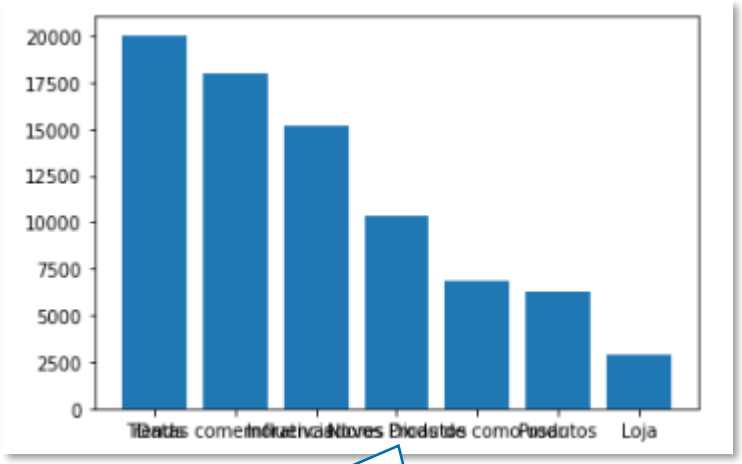


Módulo 7 – Criando e ajustando o visual (rotacionando o eixo x) de um gráfico de barras

Criamos um gráfico de barra com o auxílio da documentação:

```
media = baseTags.groupby ("Tags")["Curtidas"].mean() .sort_values(ascending=False)

x = media.index
y = media.values

fig, ax = plt.subplots()

ax.bar(x, y, linewidth=2.0)

plt.show()
```

Alterou **plot** para **bar**

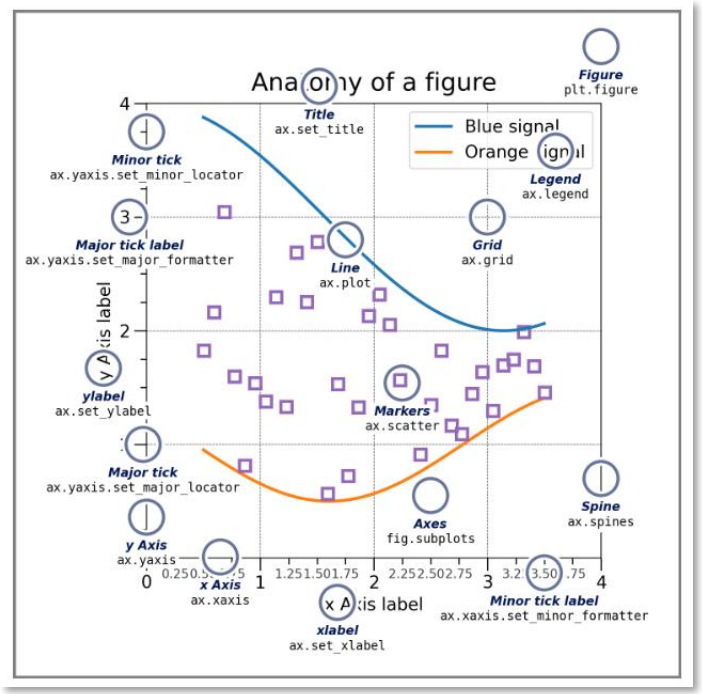


Difícil de identificar o eixo x

Módulo 7 – Criando e ajustando o visual (rotacionando o eixo x) de um gráfico de barras

Então agora vamos melhorar a visualização do gráfico:

O próprio matplotlib já mostra como podemos mudar a linha, a borda, o eixo, etc.



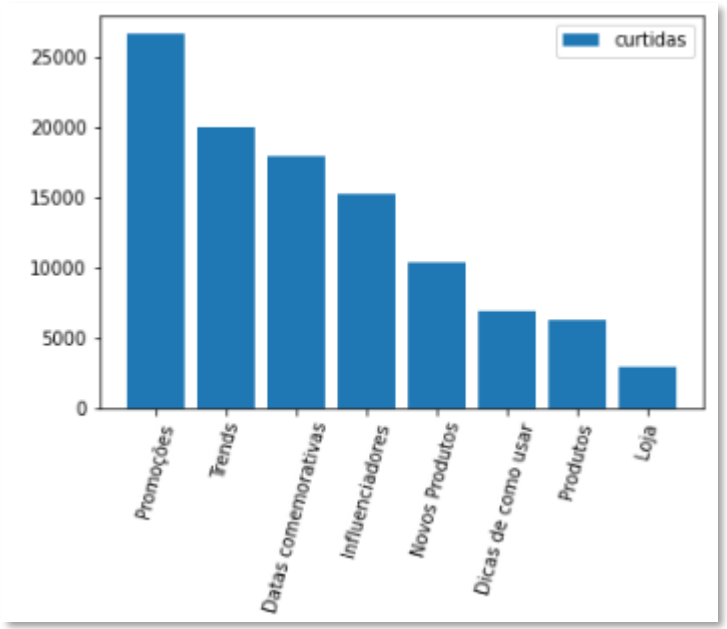
Módulo 7 – Criando e ajustando o visual (rotacionando o eixo x) de um gráfico de barras

Vamos rotacionar o eixo x utilizando o **tick_params**, ou seja, os parâmetros dos rótulos:

ax.**tick_params**('**eixo**',labelrotation=**valor_da_rotação**)

Aplicando no nosso exemplo:

```
fig, ax = plt.subplots()

ax.bar(x, y, label="curtidas")
ax.tick_params('x',labelrotation=75)

ax.legend()
plt.show()
```



Módulo 7 – Usando o annotate para adicionar rótulos de dados no gráfico

O annotate vai servir para a gente colocar qualquer texto no nosso gráfico.

Estrutura:

plt.annotate("**TEXTO**",(**x**,**y**))

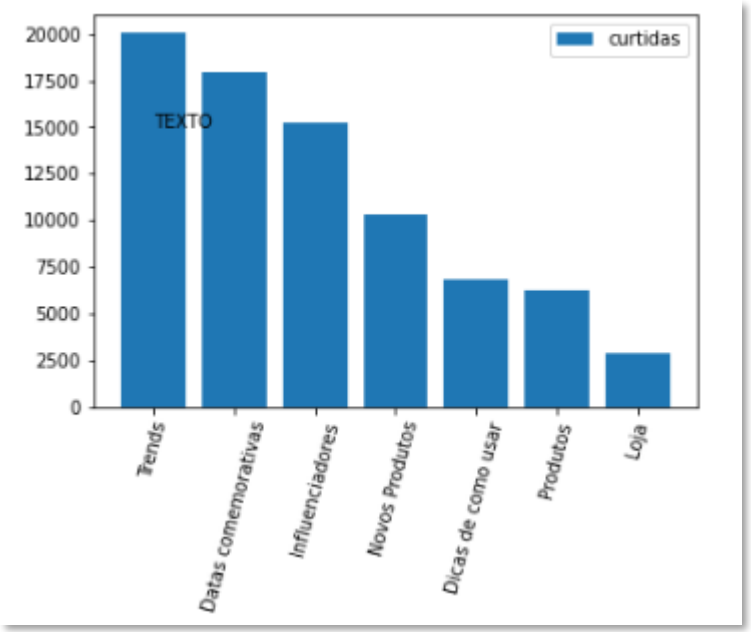
Precisamos passar os argumentos:

- Texto que vai ser escrito
- Posição (x,y) onde o texto vai estar

Módulo 7 – Usando o annotate para adicionar rótulos de dados no gráfico

Vejamos um exemplo colocando a palavra texto na posição x=0 e y=15000.

```
fig, ax = plt.subplots()

ax.bar(x, y, label="curtidas")
ax.tick_params('x',labelrotation=75)

ax.legend()

plt.annotate('TEXTO',(0,15000))

plt.show()
```



Módulo 7 – Usando o annotate para adicionar rótulos de dados no gráfico

Melhorando a formatação do annotate:

```
for i in np.arange(0,8):
    plt.annotate('{:,.0f}'.format(y[i]),
                 (i,y[i]),
                 ha="center",
                 xytext=(0,5),
                 textcoords="offset points"
                )

ax.set(ylim=(0, 30000))
```

formatando o número

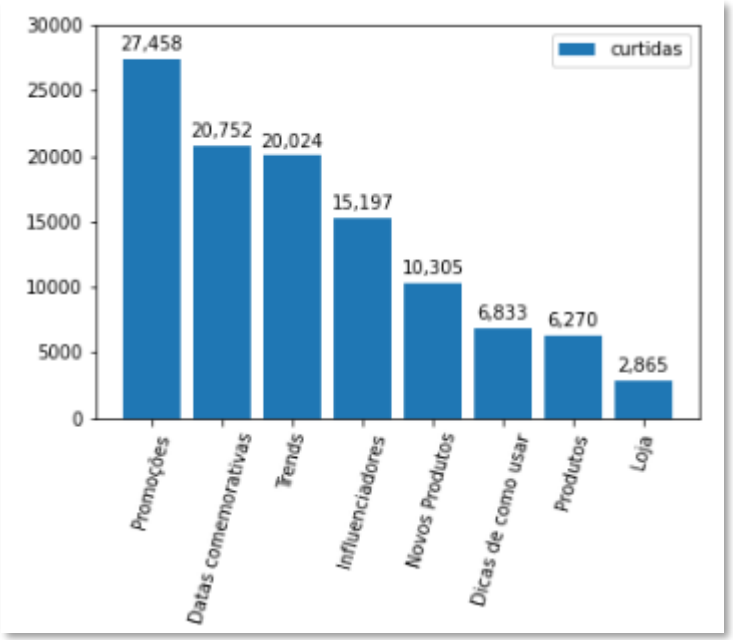
alinhamento horizontal central

colocando a posição do elemento

deslocamento x,y do texto

referencial que vamos fazer o deslocamento

Limite dos eixos



Módulo 7 – Criando um scatter plot usando apenas a documentação

Agora vamos fazer um exemplo de gráfico de dispersão a partir da documentação.

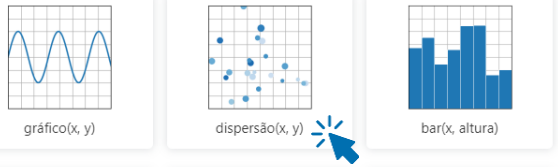
1

matplotlib | Tipos de plotagem | Exemplos | Tutoriais | Referência | Guia de usuario | Desenvolver | Notas de lançamento

2

Básico

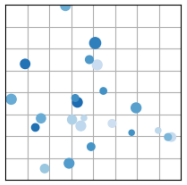
Tipos básicos de gráficos, geralmente y versus x.



3

scatter(x, y)

Veja `scatter`.



```
import matplotlib.pyplot as plt
import numpy as np

plt.style.use('_mpl-gallery')

# make the data
np.random.seed(3)
x = 4 + np.random.normal(0, 2, 24)
y = 4 + np.random.normal(0, 2, len(x))
# size and color:
sizes = np.random.uniform(15, 80, len(x))
colors = np.random.uniform(15, 80, len(x))

# plot
fig, ax = plt.subplots()

ax.scatter(x, y, s=sizes, c=colors, vmin=0, vmax=100)

ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()
```

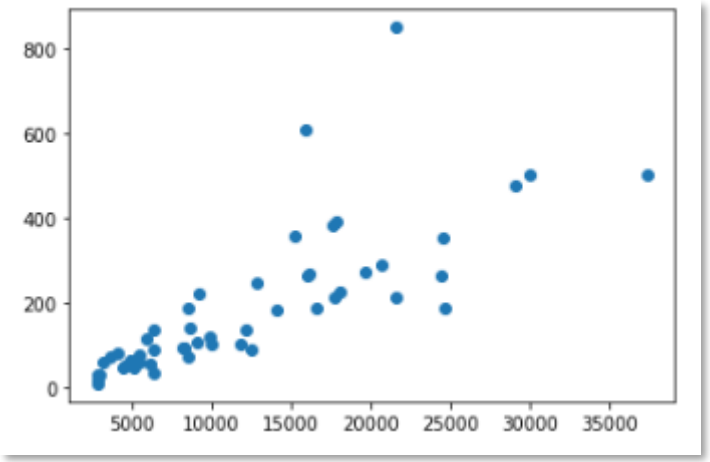
Módulo 7 – Criando um scatter plot usando apenas a documentação

Vamos copiar o código da documentação e aplicar esse gráfico a nossa base, utilizando a relação curtidas e comentários.

```
x = base.Curtidas
y = base.Comentários
```

Depois já podemos ir direto para o plot, usando apenas o básico:

```
fig, ax = plt.subplots()

ax.scatter(x, y)

plt.show()
```

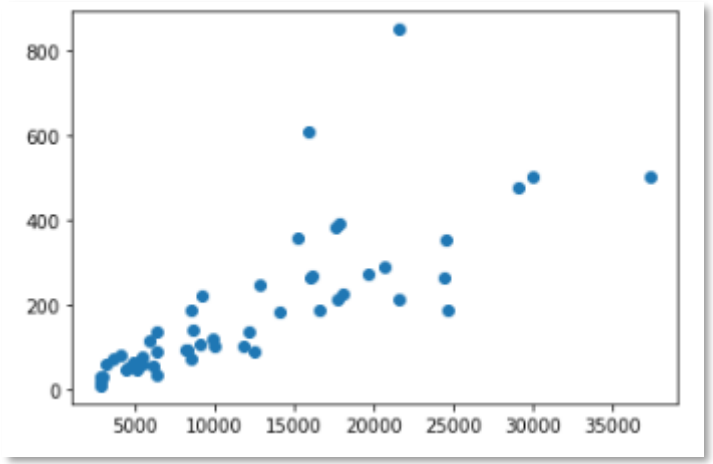


Módulo 7 – Criando um scatter plot usando apenas a documentação

Agora vamos supor que precisamos diferenciar foto, vídeo, reels e IGTV. Vamos atribuir um número para cada tipo.

```
base.Tipo.unique()
```

```
def tipo_para_numero(tipo):
    if(tipo == 'Foto'):
        return 1
    elif(tipo == 'Vídeo'):
        return 2
    elif(tipo == 'Reels'):
        return 3
    elif(tipo == 'IGTV'):
        return 4
```



Módulo 7 – Criando um scatter plot usando apenas a documentação

Aplicando a função dentro da nossa base utilizando o apply, teremos o seguinte.

```
base['NrTipo'] = base.Tipo.apply(tipo_para_numero)
```

```
base.head()
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes	mes	NrTipo
0	Foto	2021-09-11	2858	16	[Loja]	N	N	N	2874	202109	1
1	Foto	2021-09-11	2930	28	[Loja, Produtos]	N	N	N	2958	202109	1
2	Foto	2021-09-11	2807	9	[Loja]	N	N	N	2816	202109	1
3	Vídeo	2021-09-12	5115	49	[Produtos]	N	N	N	5164	202109	2
4	Foto	2021-09-13	4392	45	[Produtos]	S	N	N	4437	202109	1

Módulo 7 – Criando um scatter plot usando apenas a documentação

Agora podemos utilizar a coluna '**NrTipo**' para a nossa variável **colors**.

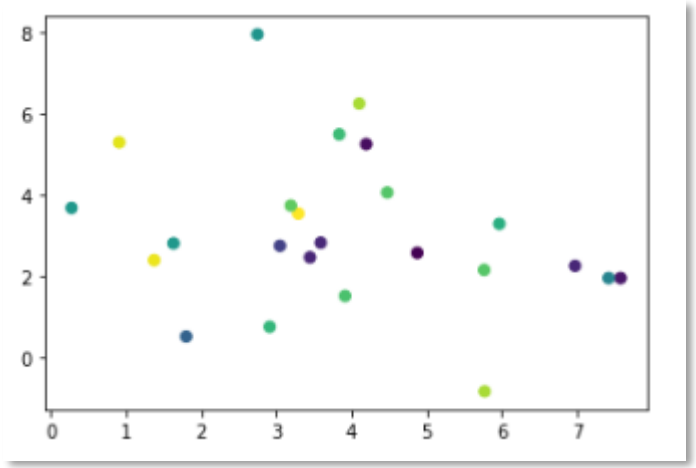
colors = base.**NrTipo**

Depois já podemos ir direto para o plot, usando apenas o básico:

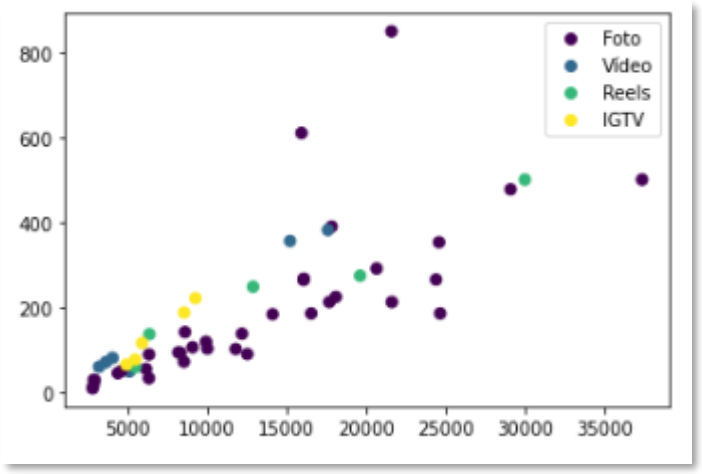
fig, ax = **plt.subplots**()

ax.**scatter**(**x**, **y**, c=**colors**)

plt.show()



Módulo 7 – Criando um scatter plot usando apenas a documentação



DICA: Também podemos adicionar a legenda em relação a cada um dos pontos utilizando o código abaixo:

```
fig, ax = plt.subplots()

scatter_plot = ax.scatter(x, y, c=colors)

ax.legend(handles=scatter_plot.legend_elements()[0],
labels=['Foto','Vídeo','Reels','IGTV'])

plt.show()
```

Módulo 7 – (Opcional) Revisando o datetime e o astype

Nessa aula vamos entender passo a passo como fazer os ajustes nos nossos cálculos de data. O primeiro passo é importar a biblioteca.

Import dateteime **as** dt

Base[‘**Mes**’] = **base**.Data.**dt**.month

Mas, repare que nós temos valores de 2021 e 2022. Quando utilizamos a função mês ele somaria o mesmo mês dos dois anos.

Uma forma de resolver esse problema é considerar o ano.

Base[‘**Ano**’] = **base**.Data.**dt**.year

Módulo 7 – (Opcional) Revisando o datetime e o astype

Só que existe uma forma melhor que é colocando **Ano_mês**, pois dessa forma ele já fica ordenado.

Por exemplo:

- 202109
- 202110
- 202201

Caso contrário não seria possível ordenar, por exemplo:

- 92021
- 102021
- 12022

Módulo 7 – (Opcional) Revisando o datetime e o astype

Então nós queremos:
Ano mês
2021 001 → multiplicamos por 100

E como que podemos fazer essa fórmula?

base[“**DataAjustada**"] = **base**.[‘**Ano**’]*100 + **base**[**'Mes’**]

```
base.head()
```

	Tipo	Data	Curtidas	Comentários	Tags	Pessoas	Campanhas	Carrossel	Interacoes	Mes	Ano	DataAjustada
0	Foto	2021-09-11	2858	16	Loja	N	N	N	2874	9	2021	202109
1	Foto	2021-09-11	2930	28	Loja/Produtos	N	N	N	2958	9	2021	202109
2	Foto	2021-09-11	2807	9	Loja	N	N	N	2816	9	2021	202109
3	Vídeo	2021-09-12	5115	49	Produtos	N	N	N	5164	9	2021	202109
4	Foto	2021-09-13	4392	45	Produtos	S	N	N	4437	9	2021	202109

Módulo 7 – (Opcional) Revisando o datetime e o astype

Utilizamos o **astype** para transformar qualquer tipo de dado.

```
base.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 52 entries, 0 to 51
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Tipo          52 non-null     object
 1   Data          52 non-null     datetime64[ns]
 2   Curtidas      52 non-null     int64
 3   Comentários   52 non-null     int64
 4   Tags          44 non-null     object
 5   Pessoas       52 non-null     object
 6   Campanhas     52 non-null     object
 7   Carrossel     52 non-null     object
 8   Interacoes    52 non-null     int64
 9   Mes           52 non-null     int64
 10  Ano           52 non-null     int64
 11  DataAjustada  52 non-null     int64
dtypes: datetime64[ns](1), int64(6), object(5)
memory usage: 5.0+ KB
```

Inteiro

```
base['DataAjustada'] = base['DataAjustada'].astype(str)
```

```
base.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 52 entries, 0 to 51
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Tipo          52 non-null     object
 1   Data          52 non-null     datetime64[ns]
 2   Curtidas      52 non-null     int64
 3   Comentários   52 non-null     int64
 4   Tags          44 non-null     object
 5   Pessoas       52 non-null     object
 6   Campanhas     52 non-null     object
 7   Carrossel     52 non-null     object
 8   Interacoes    52 non-null     int64
 9   Mes           52 non-null     int64
 10  Ano           52 non-null     int64
 11  DataAjustada  52 non-null     object
dtypes: datetime64[ns](1), int64(5), object(6)
memory usage: 5.0+ KB
```

Objeto

Módulo 7 – (Opcional) Adicionando rótulo para cores de um scatter plot

Vamos aprender agora como inserir rótulos em um gráfico de dispersão.

O primeiro passo é importar a base do Excel:

Import pandas **as** pd

base = **pd.read**_excel('Analisando o engajamento no Instagram.xlsx')

Módulo 7 – (Opcional) Adicionando rótulo para cores de um scatter plot

Vamos criar a coluna do tipo de publicação, como feito anteriormente:

```
def tipo_para_numero(tipo):
    if(tipo == 'Foto'):
        return 1
    elif(tipo == 'Vídeo'):
        return 2
    elif(tipo == 'Reels'):
        return 3
    elif(tipo == 'IGTV'):
        return 4

base['NrTipo'] = base.Tipo.apply(tipo_para_numero)
```

Módulo 7 – (Opcional) Adicionando rótulo para cores de um scatter plot

Vamos visualizar a nossa base:

base.head

```
base.head()
```

	Tipo	Data	Curtidas	Comentários	Visualizações	Tags	Pessoas	Campanhas	Carrossel	Interacoes	NrTipo
0	Foto	2021-09-11	2858	16	NaN	Loja	N	N	NaN	2874	1
1	Foto	2021-09-11	2930	28	NaN	Loja/Produtos	N	N	NaN	2958	1
2	Foto	2021-09-11	2807	9	NaN	Loja	N	N	NaN	2816	1
3	Vídeo	2021-09-12	5115	49	82878.0	Produtos	N	N	NaN	5164	2
4	Foto	2021-09-13	4392	45	NaN	Produtos	S	N	NaN	4437	1

Módulo 7 – (Opcional) Adicionando rótulo para cores de um scatter plot

Vamos criar um gráfico de dispersão a partir do código da documentação:

```
fig, ax = plt.subplots()

ax.scatter(x, y, s=sizes, c=colors, vmin=0, vmax=100)

ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()
```

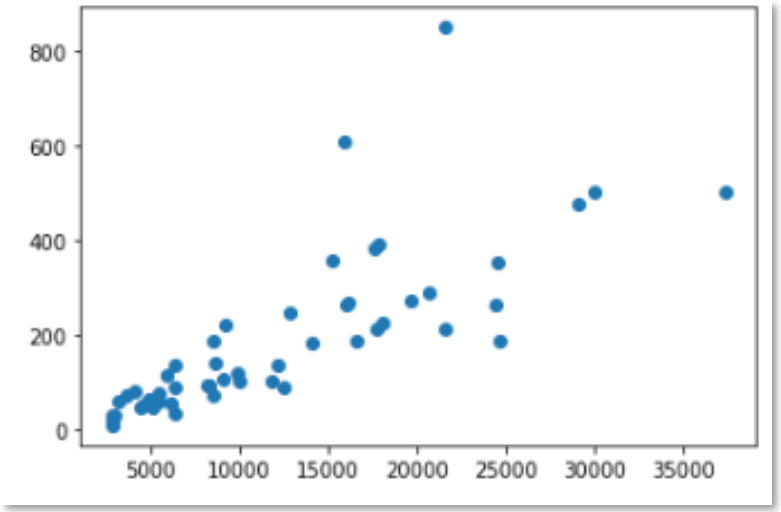
Vamos deixar só os eixos

Podemos excluir

Módulo 7 – (Opcional) Adicionando rótulo para cores de um scatter plot

Agora, basta substituirmos o x para curtidas e o y para comentários.

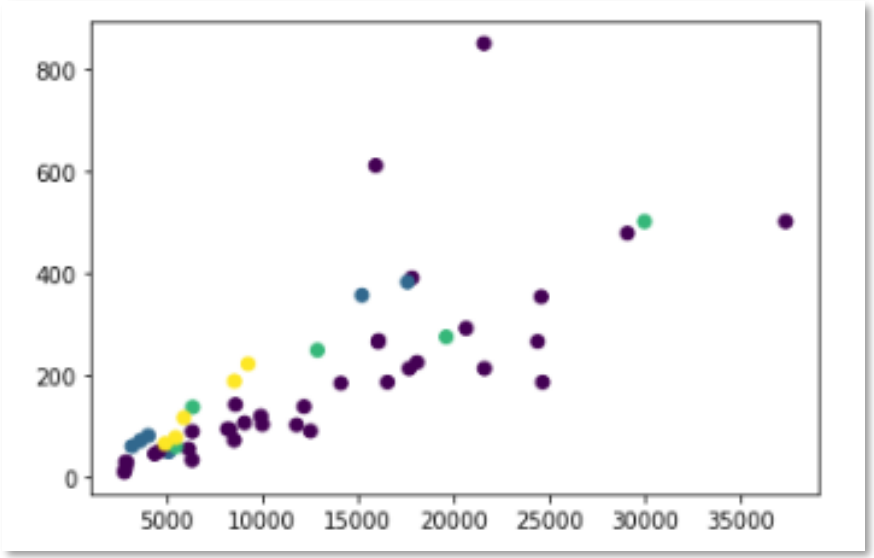
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

x = base.Curtidas
y = base.Comentários

ax.scatter(x, y)

plt.show()
```



Módulo 7 – (Opcional) Adicionando rótulo para cores de um scatter plot

Separando os tipos por cores:

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

x = base.Curtidas
y = base.Comentários

ax.scatter(x, y, c= base.NrTipo)

plt.show()
```



Módulo 7 – (Opcional) Adicionando rótulo para cores de um scatter plot

Vamos salvar esse nosso gráfico em uma variável **scatter_plot** que será um **pathcolection**.

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

x = base.Curtidas
y = base.Comentários

scatter_plot = ax.scatter(x, y, c= base.NrTipo)

plt.show()
```

Atribuir a uma variável

```
scatter_plot
<matplotlib.collections.PathCollection at 0x1ed79031e80>
```

Módulo 7 – (Opcional) Adicionando rótulo para cores de um scatter plot

Paths: um módulo para lidar com polilinhas usadas no matplotlib

- Quase todos os desenhos vetoriais usam Paths em algum lugar no pipeline de desenho

legend_elements()

- Crie identificadores e rótulos de legenda para um PathCollection
- Retornos: handles (elementos 2D) e labels (texto dos rótulos)

Módulo 7 – (Opcional) Adicionando rótulo para cores de um scatter plot

Aplicando o **legend_elements** para a nossa variável do gráfico temos como retorno os pontos 2D e o texto.

Então vamos definir isso como uma nova variável:

handles, labels = **scatter_plot**.**legend_elements**()

```
scatter_plot.legend_elements()

([<matplotlib.lines.Line2D at 0x1ed79038160>,
  <matplotlib.lines.Line2D at 0x1ed79038430>,
  <matplotlib.lines.Line2D at 0x1ed79038580>,
  <matplotlib.lines.Line2D at 0x1ed78f43ac0>],
 ['$\\mathdefault{1}$',
  '$\\mathdefault{2}$',
  '$\\mathdefault{3}$',
  '$\\mathdefault{4}$'])
```

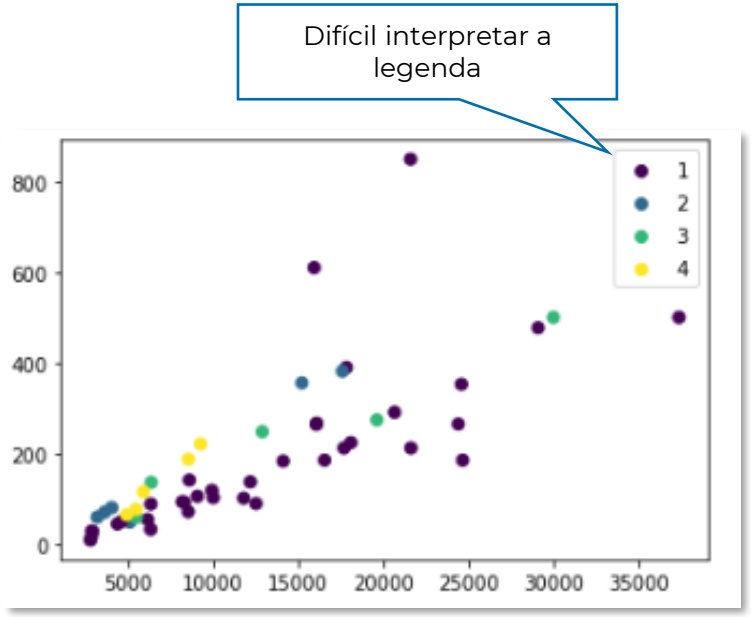
Pontos 2D

Textos

Módulo 7 – (Opcional) Adicionando rótulo para cores de um scatter plot

Vamos colocar o **ax.legend** no nosso código do gráfico utilizando como argumento a nossa variável **handles, labels** :

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

x = base.Curtidas
y = base.Comentários

scatter_plot = ax.scatter(x, y, c= base.NrTipo)
ax.legend(handles, labels)

plt.show()
```



Módulo 7 – (Opcional) Adicionando rótulo para cores de um scatter plot

Alterando a legenda para os nossos valores:

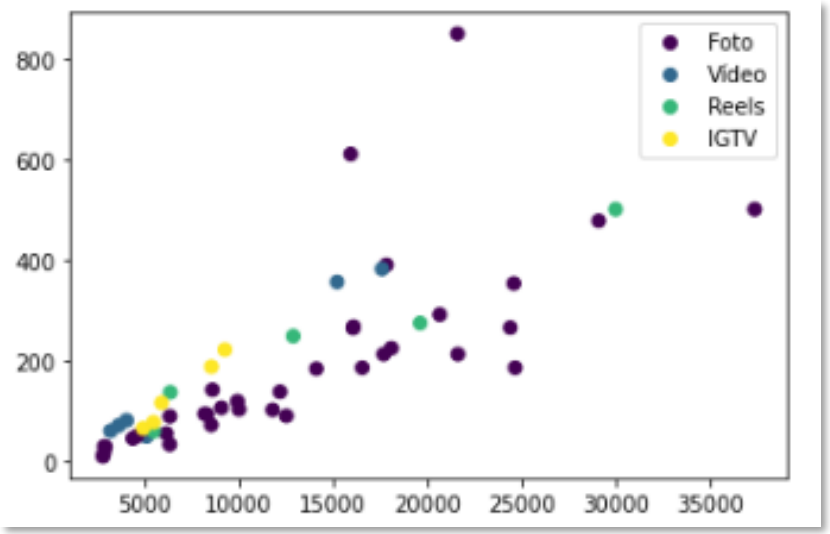
legendas = [‘Foto’, ‘Vídeo’, ‘Reels’, ‘IGTV’]

fig, ax = plt.subplots()

x = **base**.**Curtidas**
y = **base**.**Comentários**

scatter_plot = ax.**scatter**(**x**, **y**, c= **base.NrTipo**)
ax.legend(handles=**handles**, labels=**legendas**)

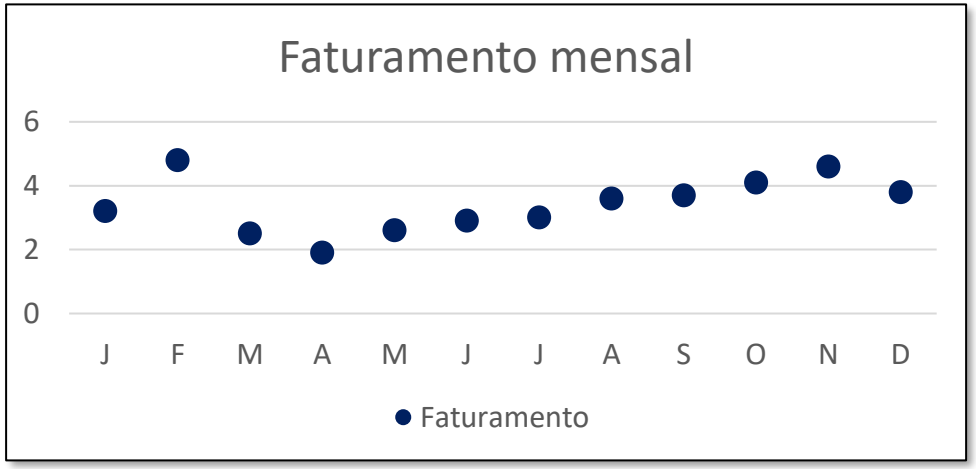
plt.show()

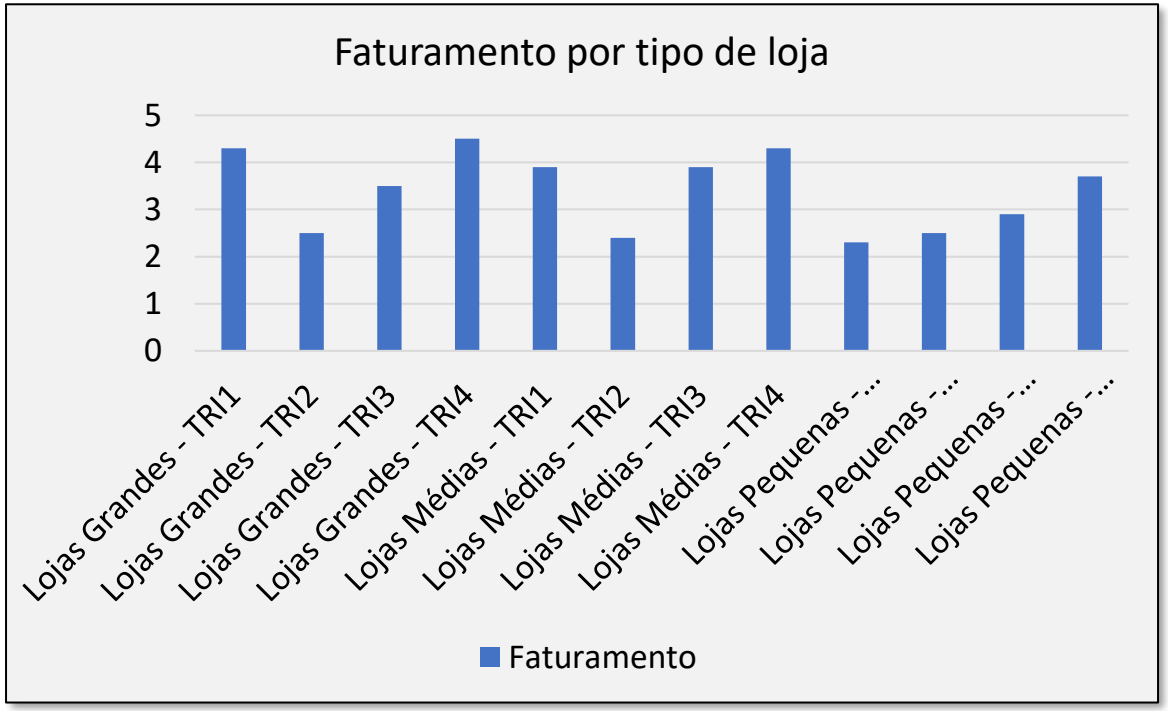


MÓDULO 8

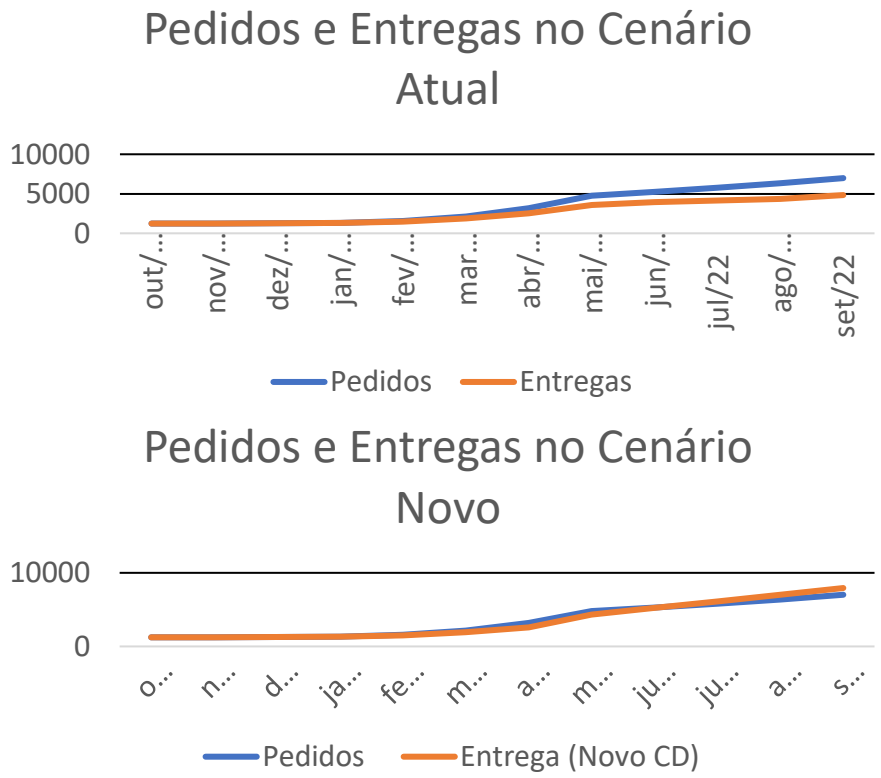
Boas práticas para apresentação de dados

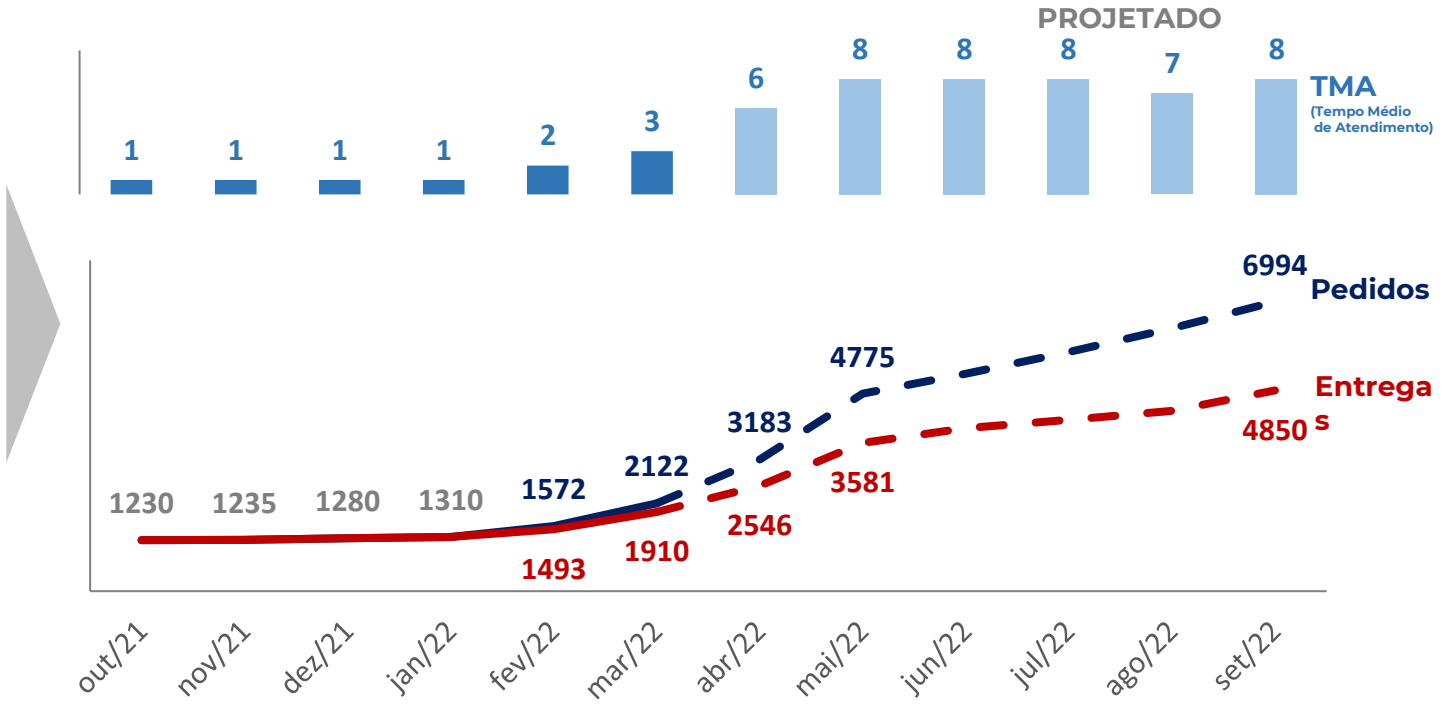
Todas as pessoas conseguem apresentar dados. Porém, a forma como você **apresenta**, a forma como esses **gráficos estão mostrados**, isso sim vai ser o nosso diferencial como um **bom cientista de dados**.



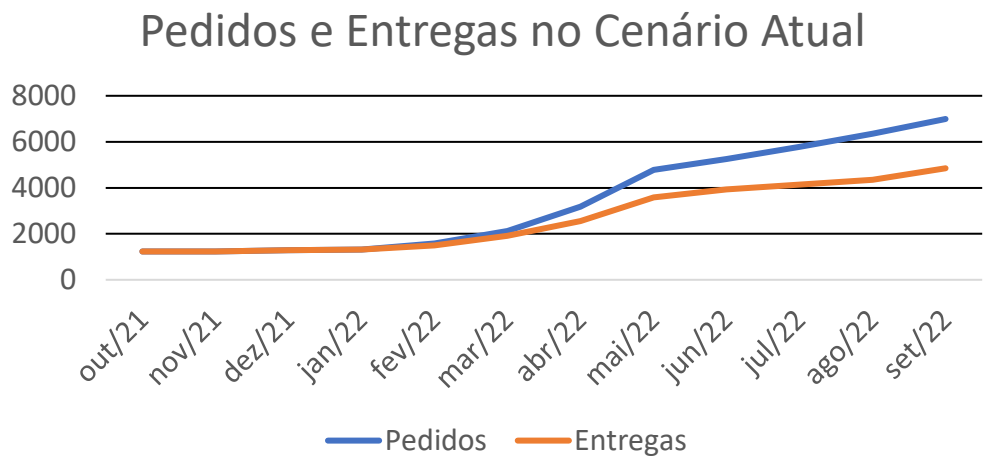


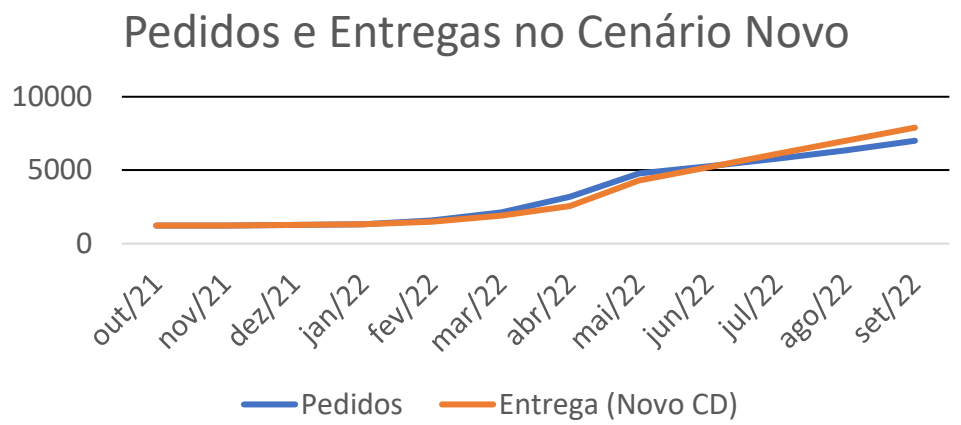
Repare as diferenças dos gráficos abaixo:



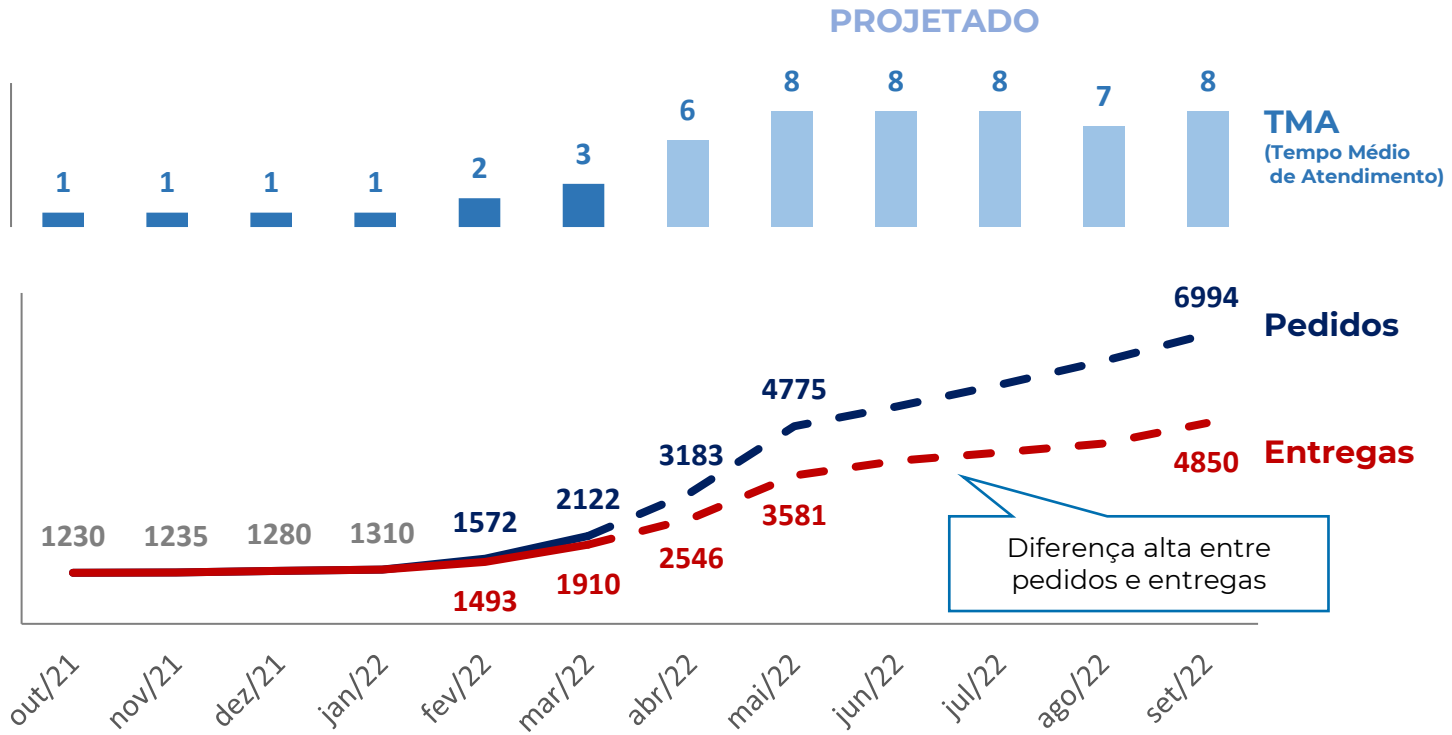


Repare que nos gráficos abaixo é **mais difícil de visualizar os valores**.

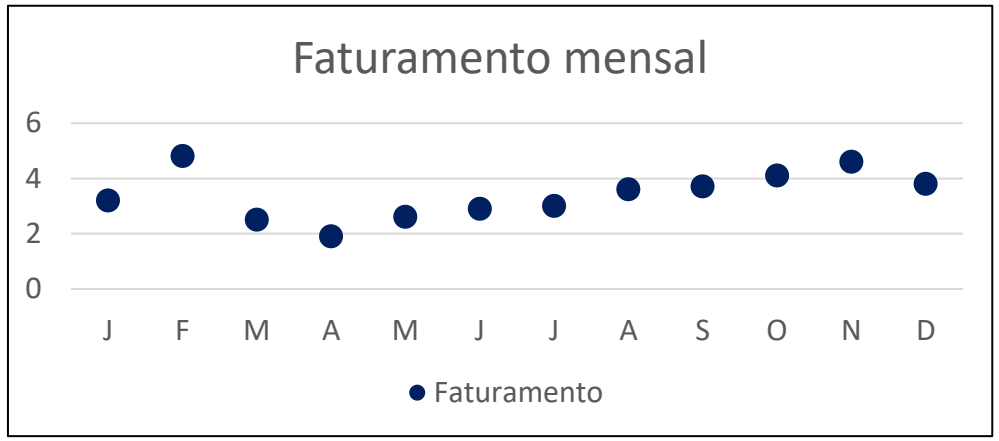


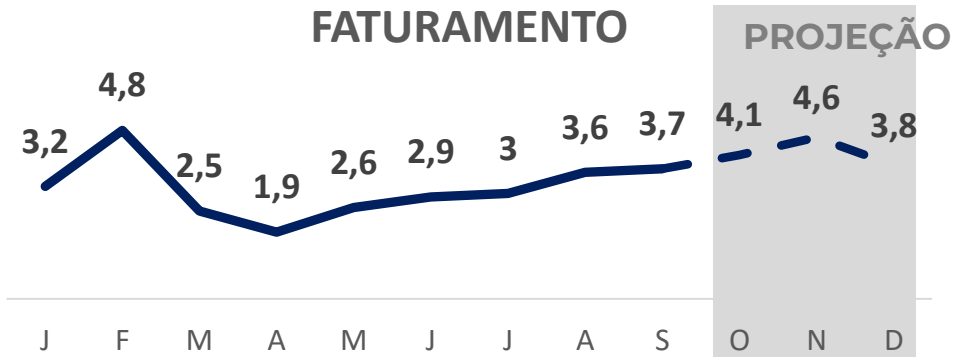


Note que agora é **mais fácil de observar os valores**, a diferença do real e do projetado.

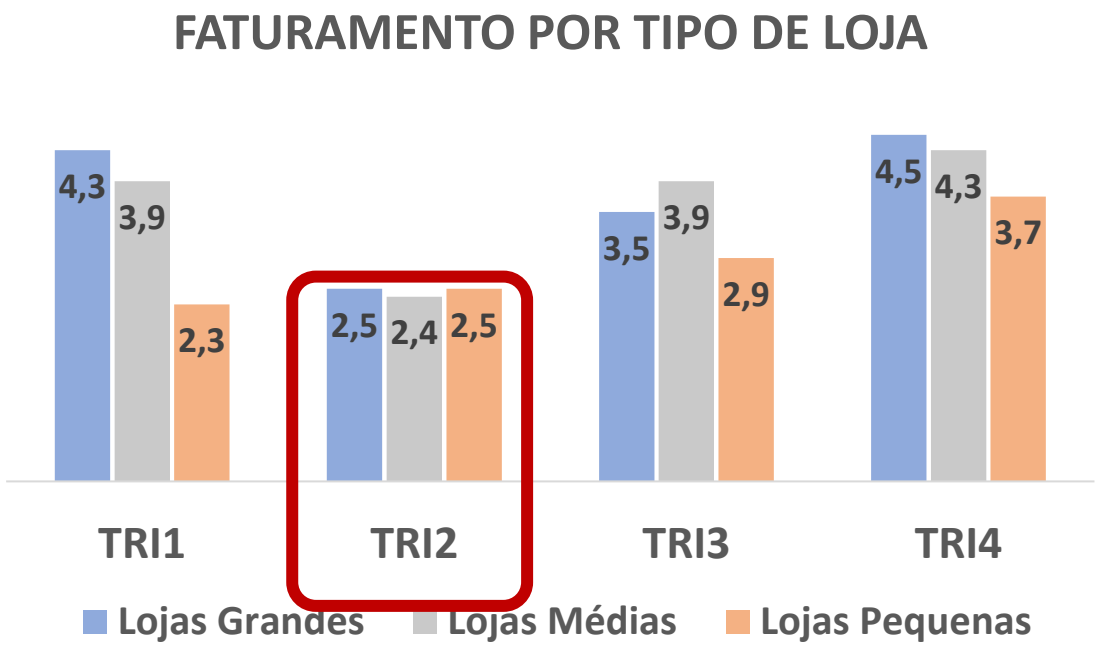


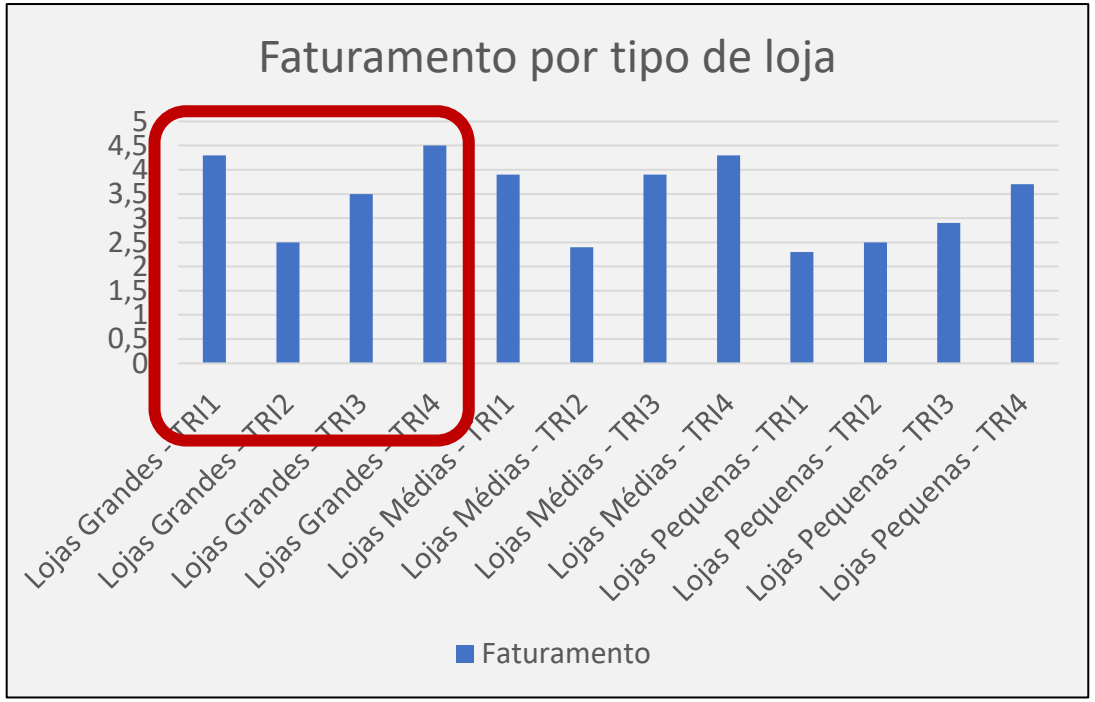
Repare que quando **ligamos os pontos** fica muito **mais fácil de visualizar** o faturamento dos meses.





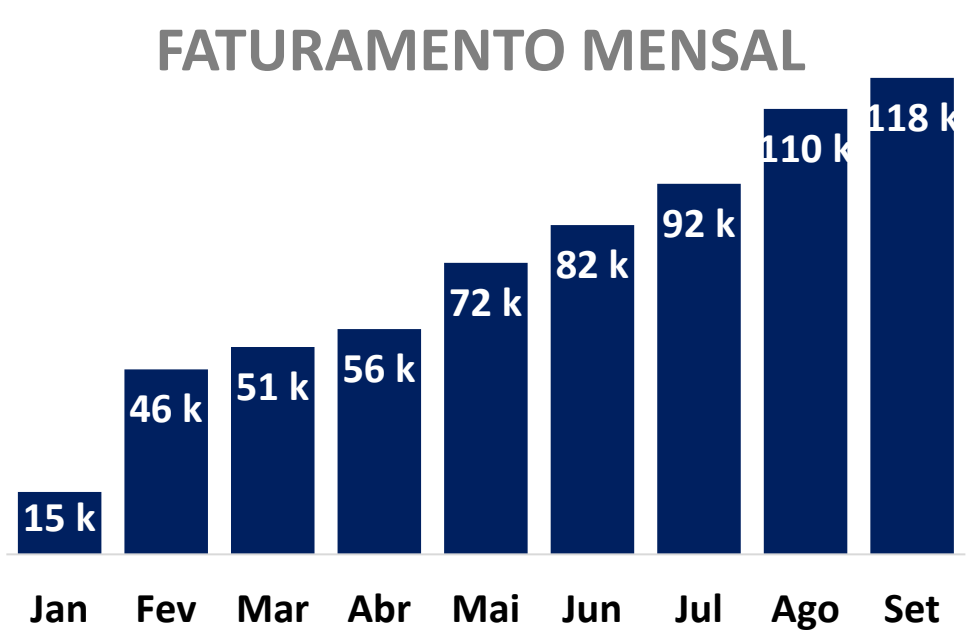
Repare que no gráfico da esquerda **não tem uma sequência cronológica**, enquanto no da direita é **mais nítido visualizar cada trimestre**.

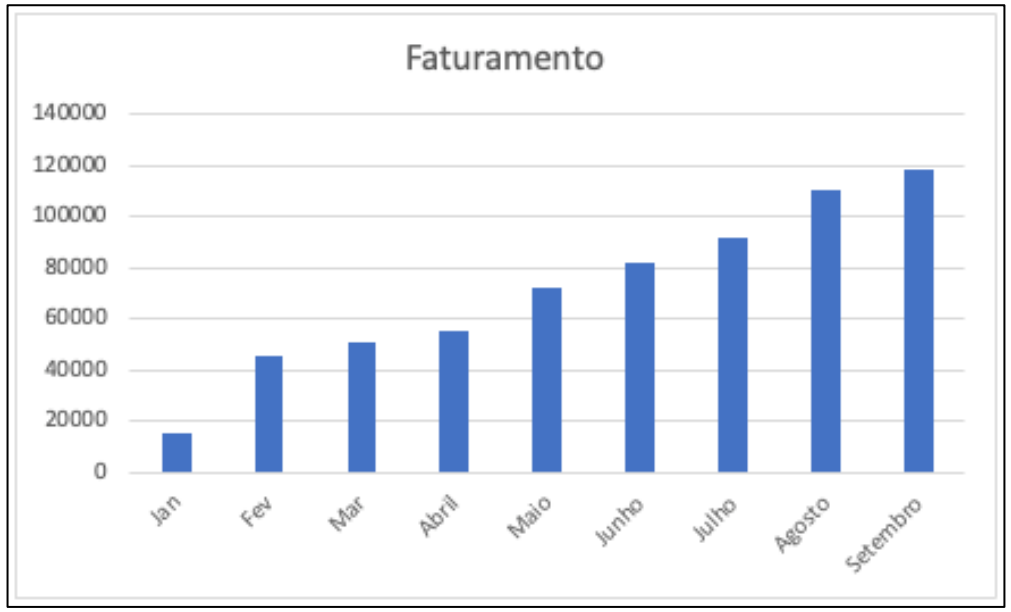


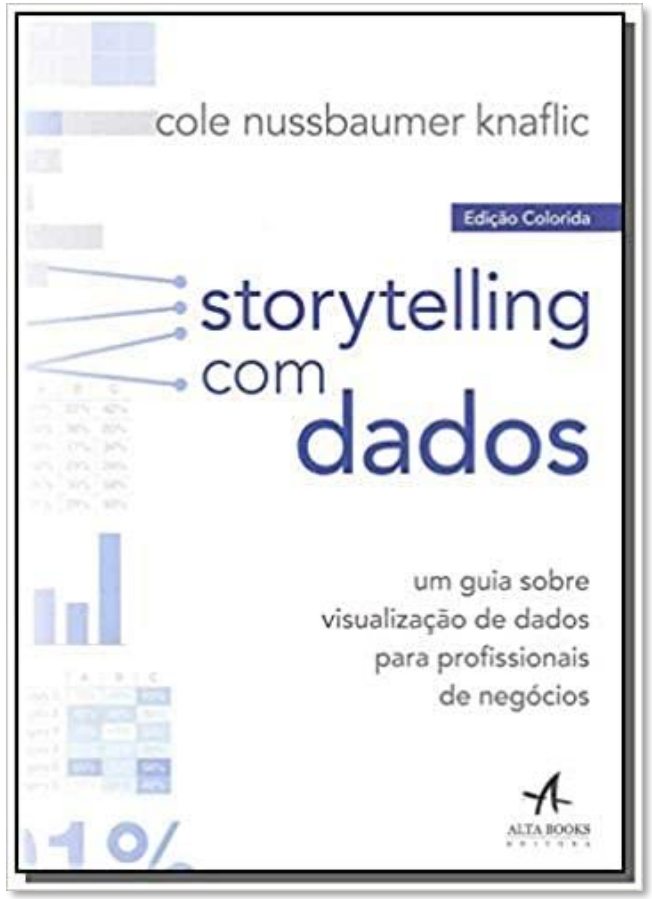


Repare que agora a informação é a mesma, porém repare na **diferença visual**.

O gráfico fica mais **confortável** para as pessoas.







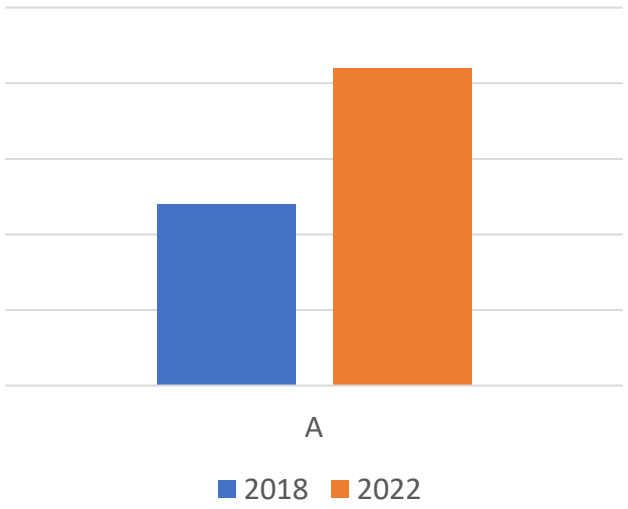
Storytelling com dados – Cole Nussbaumer Knaflic

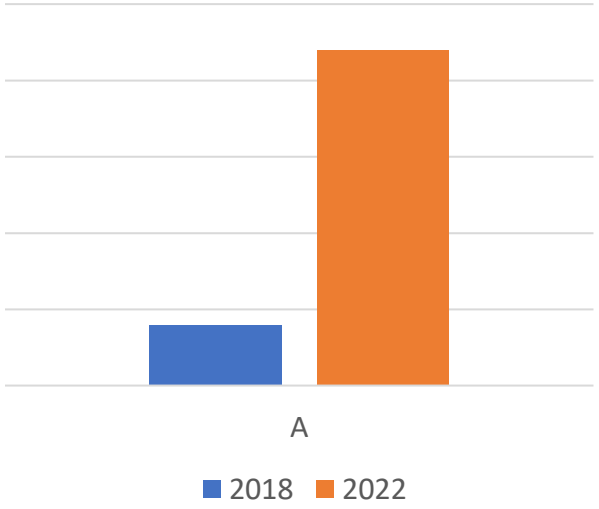
Essas dicas foram retiradas do livro “Storytelling com dados”.

Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

Vamos falar de coisas que são bem importantes antes de entrar nos detalhes visuais.
No gráfico abaixo qual dos dois apresenta um maior crescimento?

Essa pergunta deve ser respondida muito facilmente por qualquer pessoa que visualize esses gráficos.

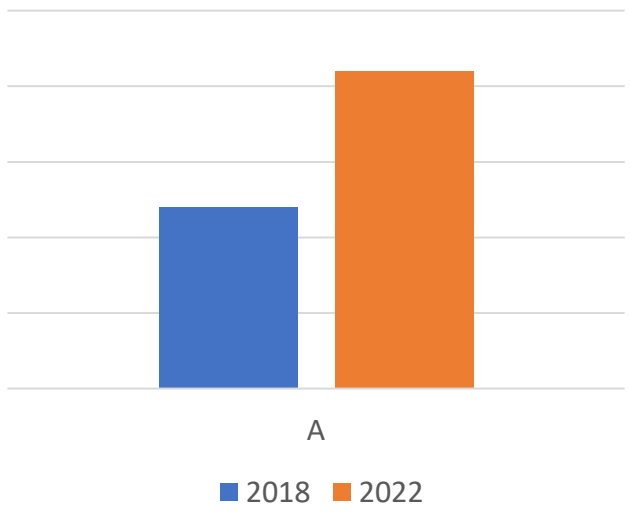


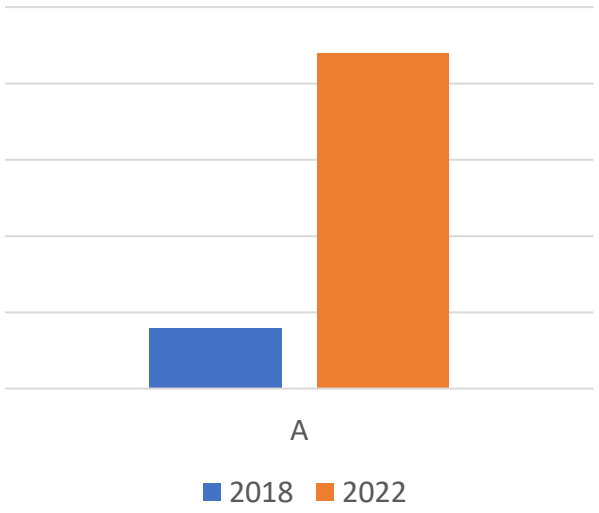


Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

Se o seu público fica em dúvida, ele pode interpretar de uma maneira errada.

As pessoas podem achar que você está querendo enganá-las ou ocultar uma informação.

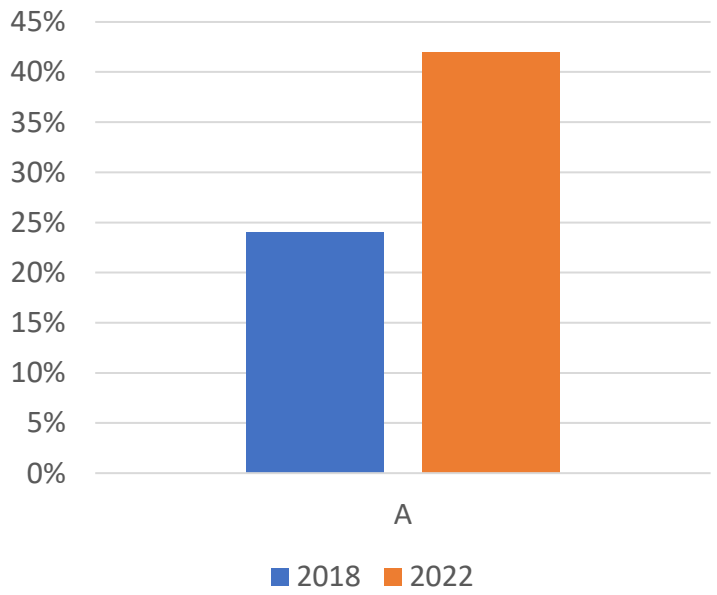


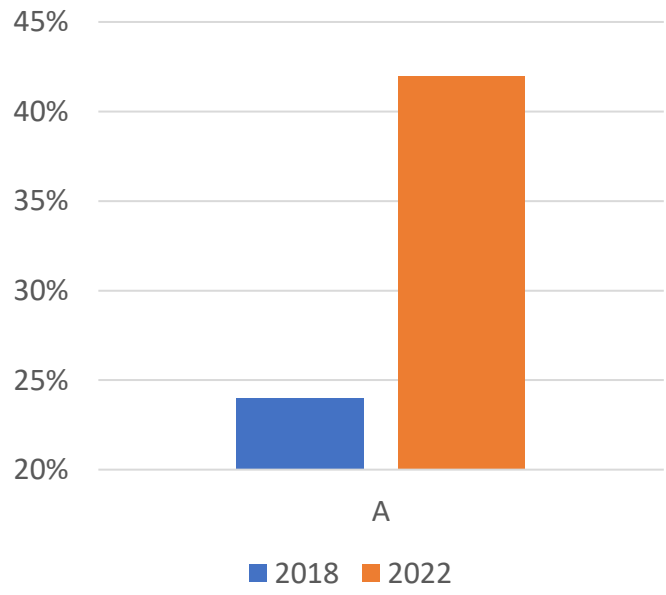


Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

Observe que ao colocar o valor dos eixos, um dos gráficos começa em 0% e o outro em 20%.

Parecia que estávamos mostrando um comportamento que não é verdade.

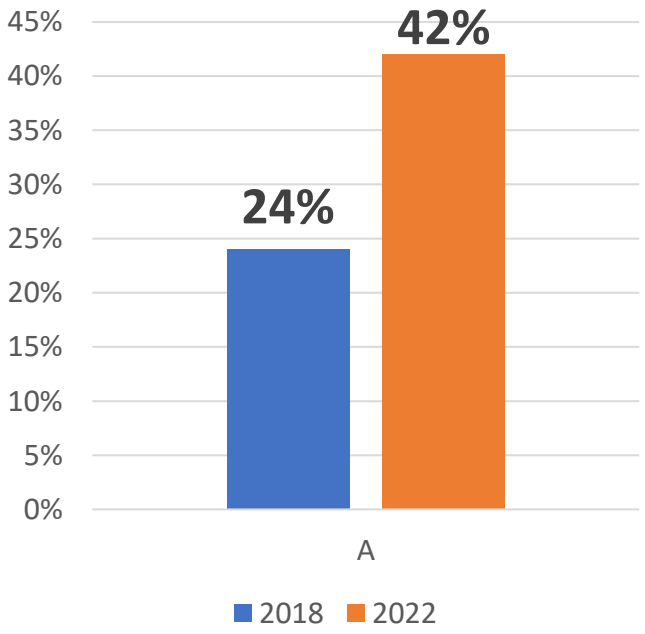


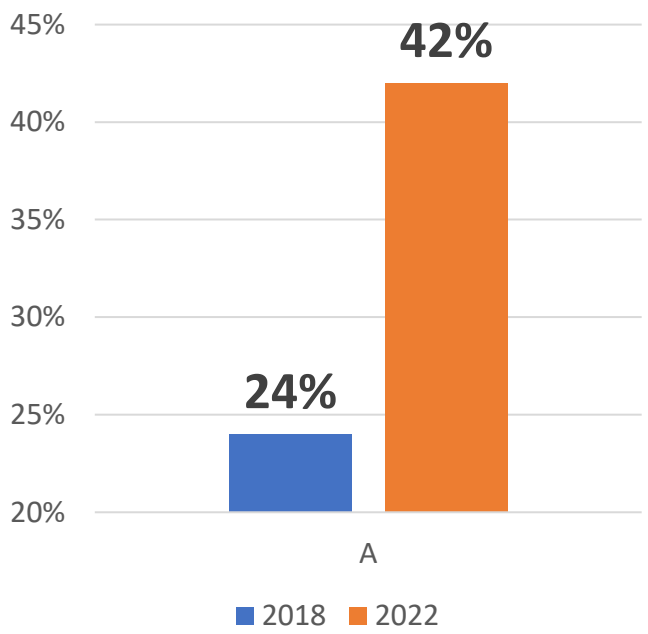


Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

Quando colocamos o rótulo de dados, percebemos que as informações são iguais.

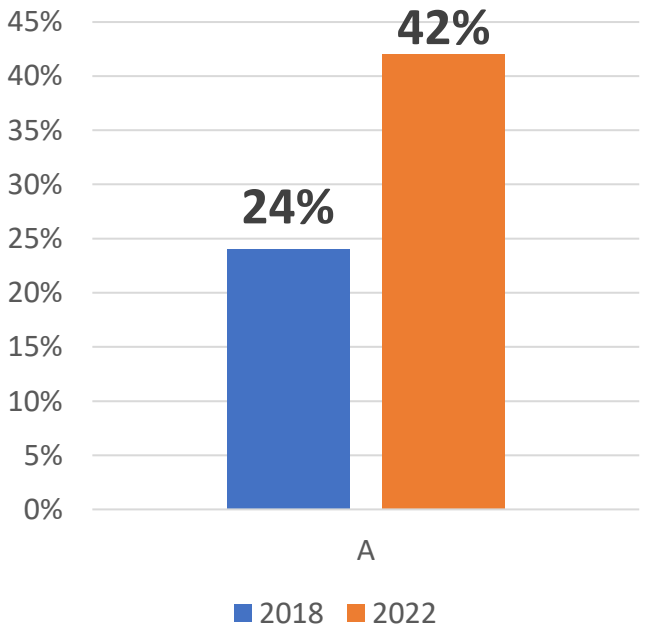
Mas, repare como visualmente não parecem iguais.

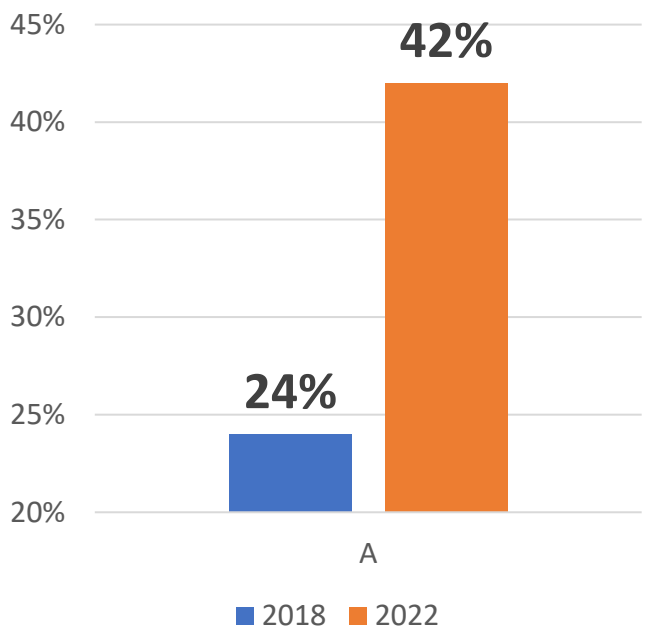




Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

Como o nosso cérebro vai comparar o tamanho das barras, é importante que a gente sempre **comece do zero** para que essa comparação seja feita da forma correta.



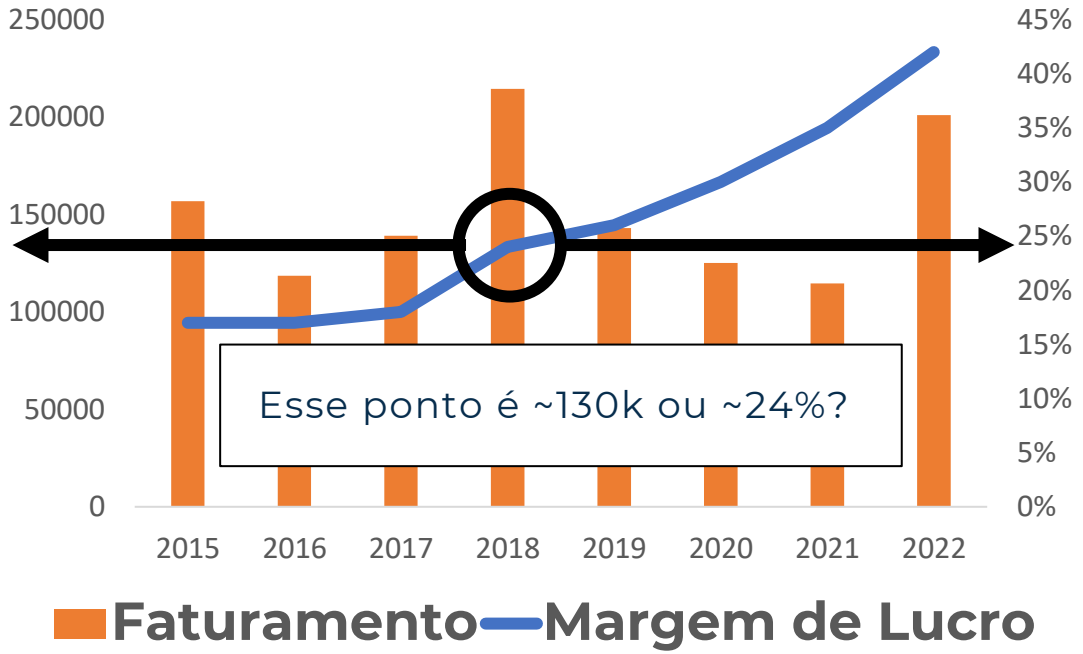


Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

Geralmente, o **eixo secundário não é uma boa ideia** pois é difícil identificar a informação.

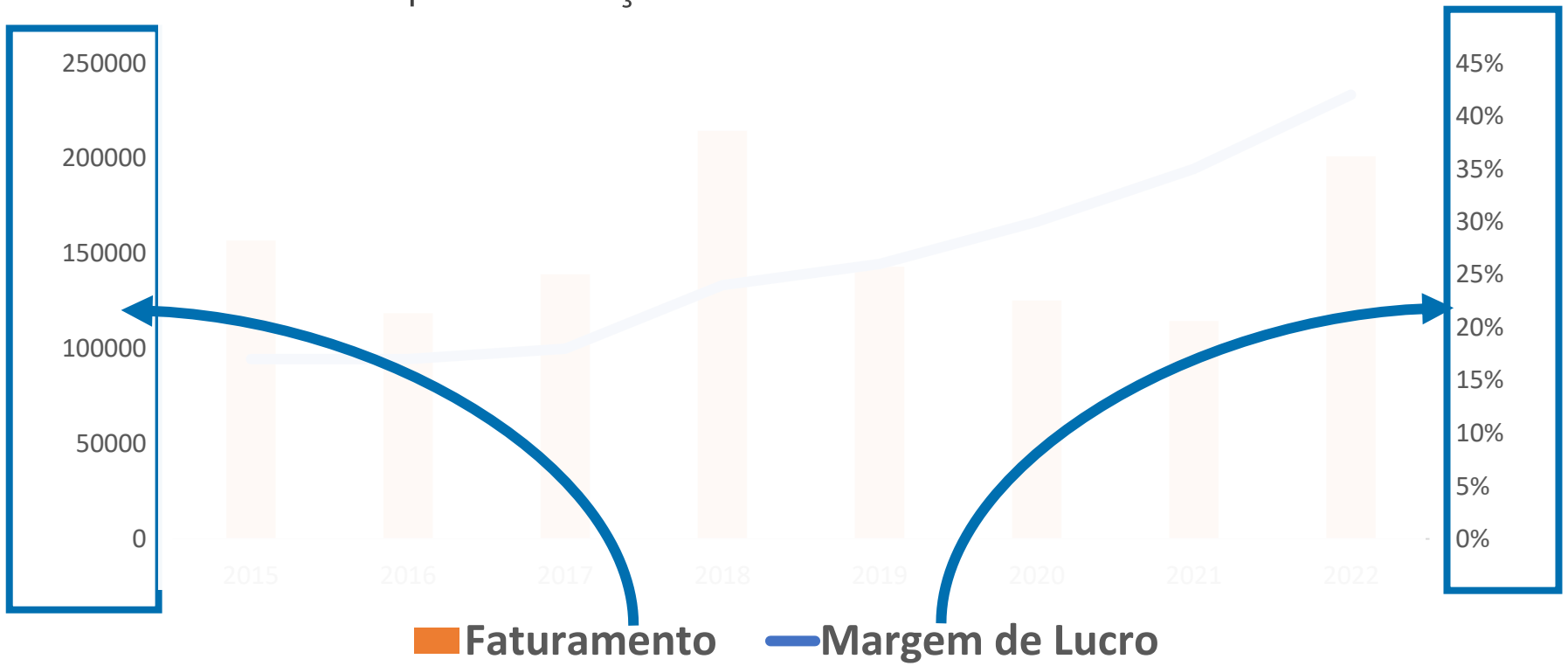
Se não está tão claro para você, vai estar menos ainda pro seu público.

Dica: faça a sua apresentação voltada para o seu público, com os gráficos que as pessoas estão acostumadas a ver.



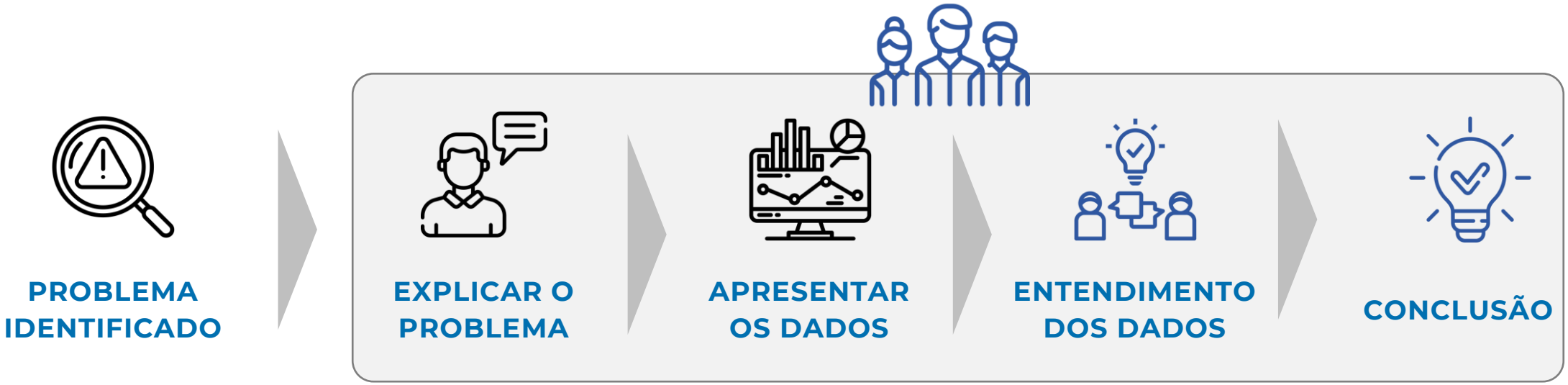
Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

Você pode pensar que é muito fácil identificar os eixos, mas será que todas as pessoas que estão assistindo a sua apresentação sabem disso?



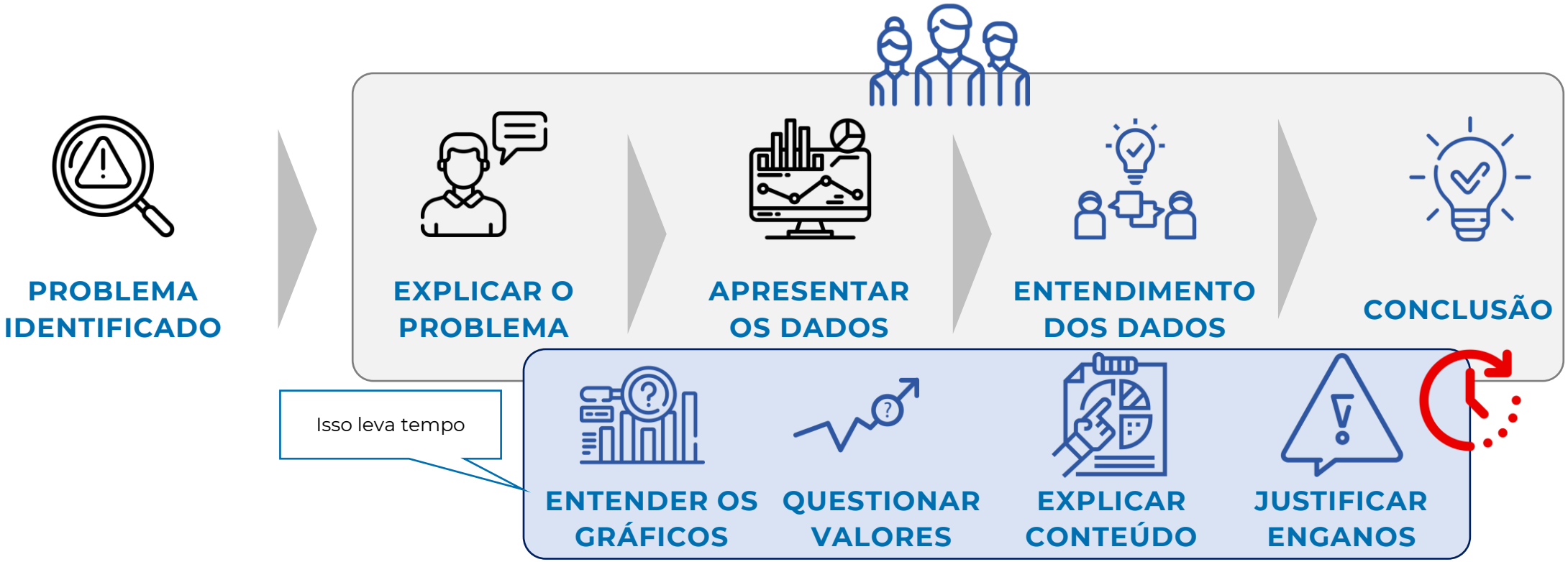
Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

Nós começamos a montar uma apresentação identificando e explicando o problema, apresentamos os dados e as pessoas vão ter que entender os dados para chegar em uma conclusão.



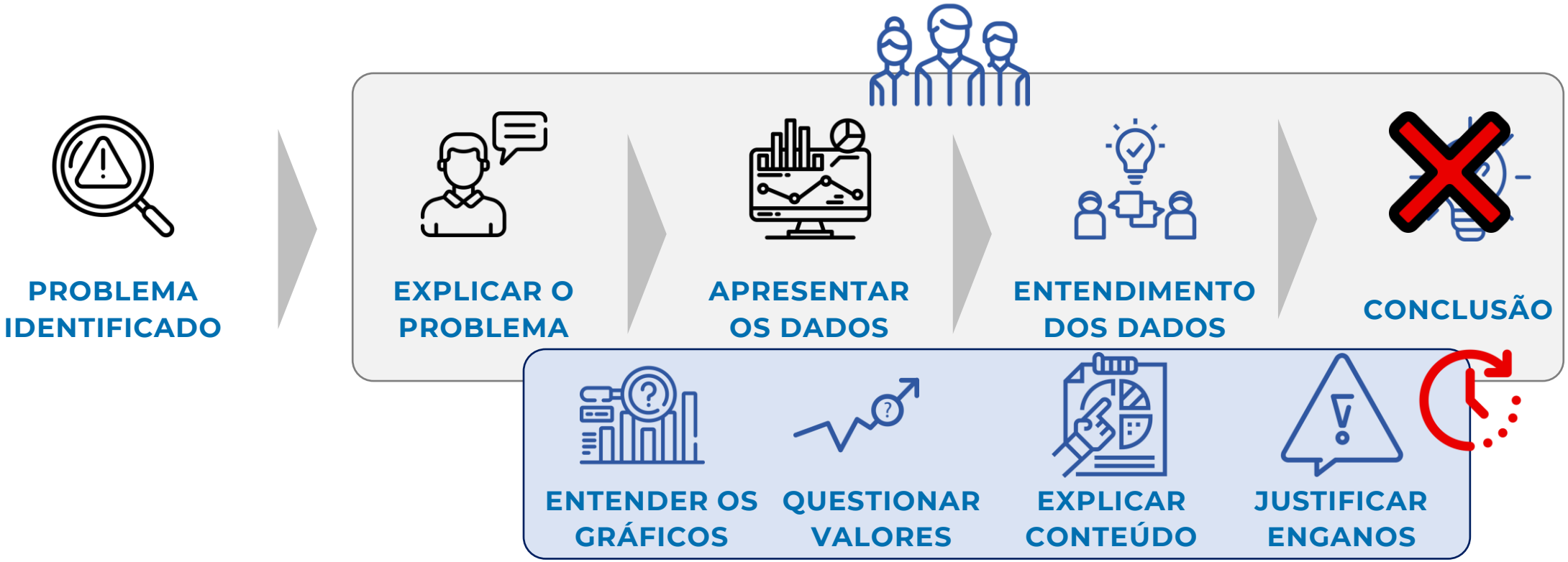
Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

Basicamente vamos adicionar etapas no processo de apresentação e entendimento dos dados.



Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

Isso pode fazer com que a nossa reunião não tenha uma conclusão.



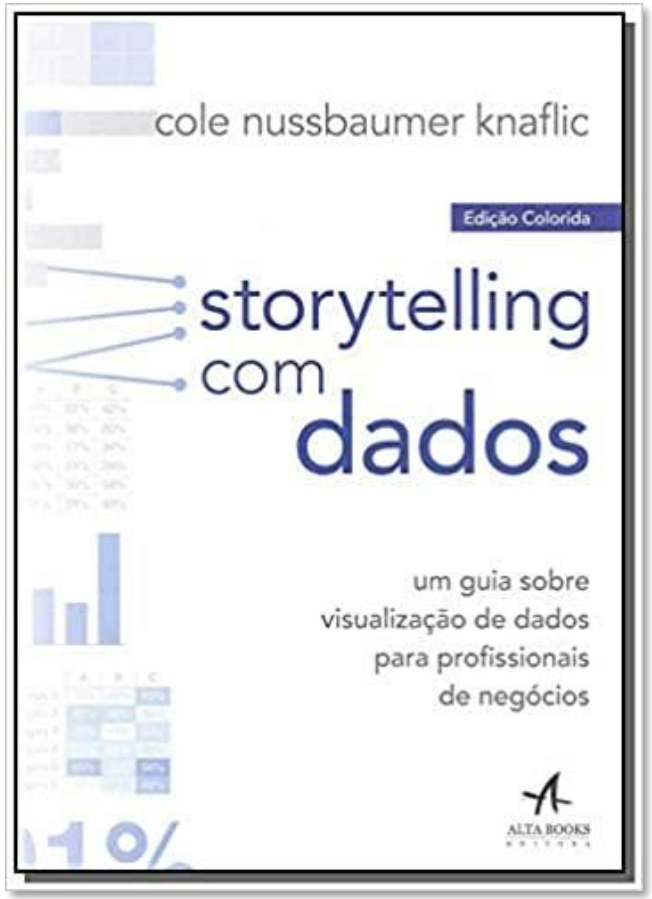
Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

Então façam com que a etapa entre apresentar os dados e o entendimento dos dados seja **a mais tranquila possível para o usuário**.

Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

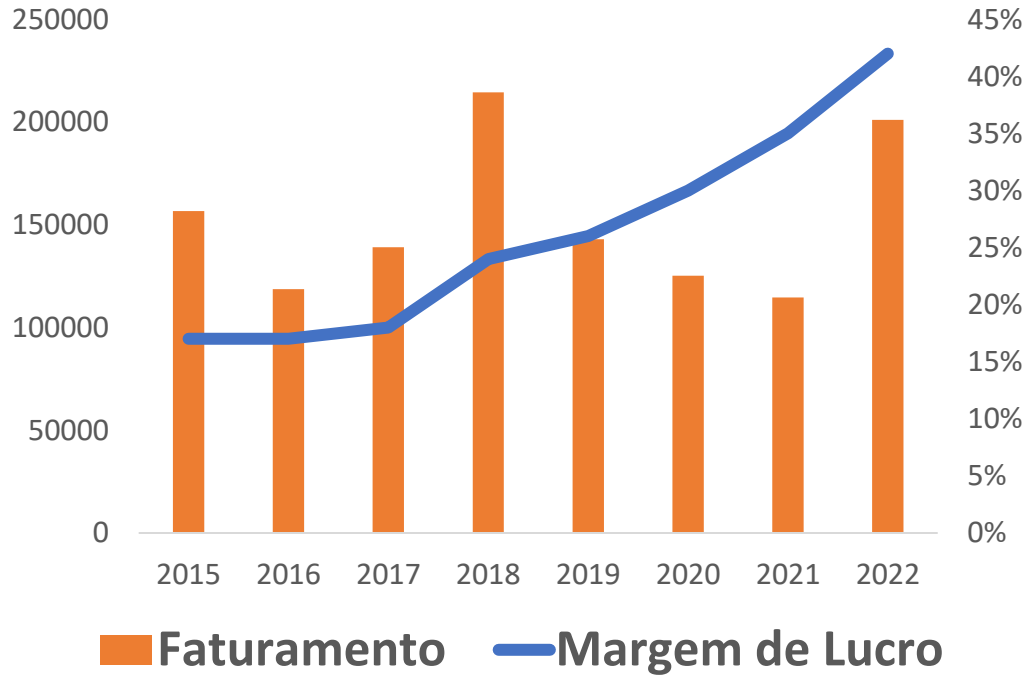
"No que diz respeito às nossas comunicações visuais, o que mais importa é a carga cognitiva **percebida** por parte do nosso público: o quanto ele **acredita que precisará se esforçar** para entender a informação de sua comunicação"

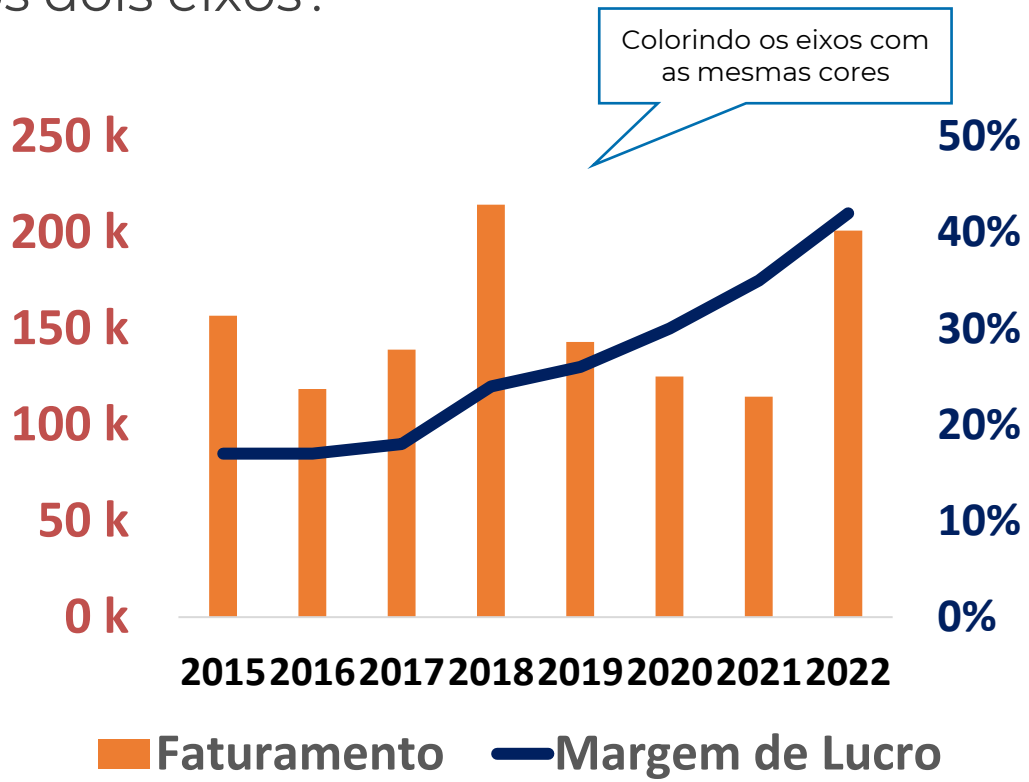
Storytelling com Dados - Cole Nussbaumer Knaflic



Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

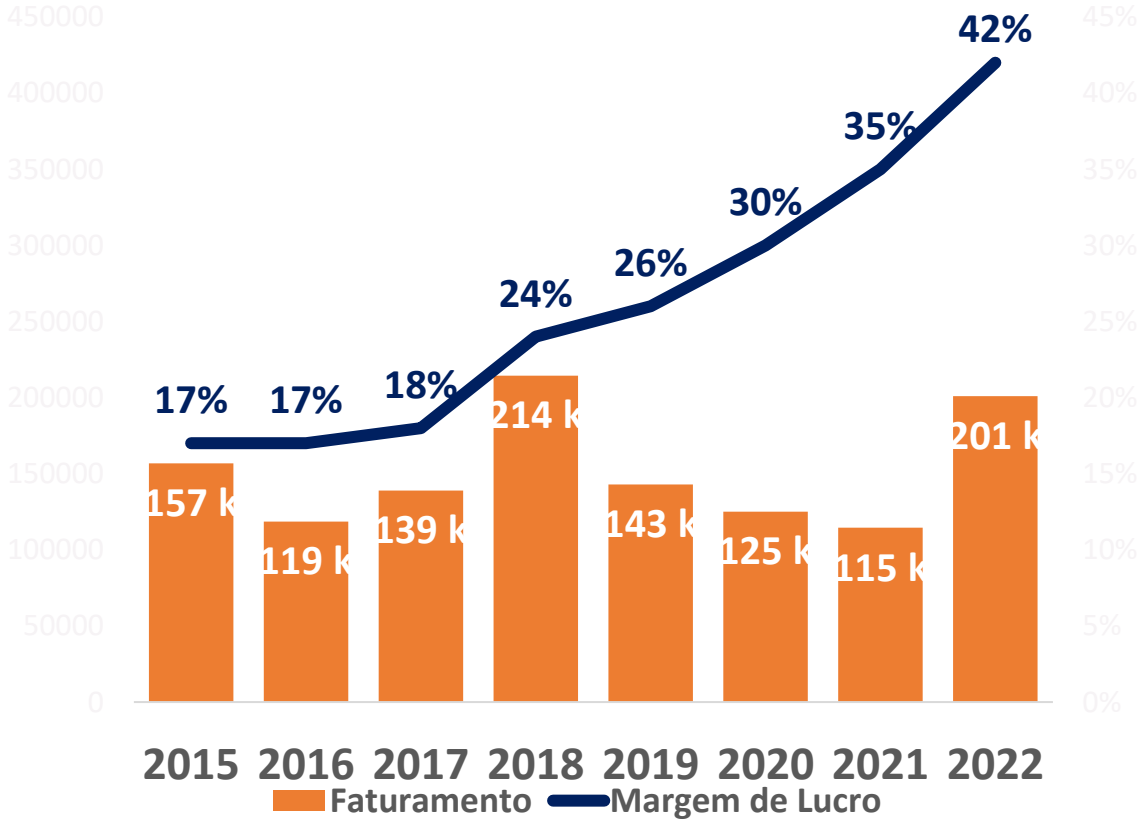
Então como poderíamos resolver o problema dos dois eixos?

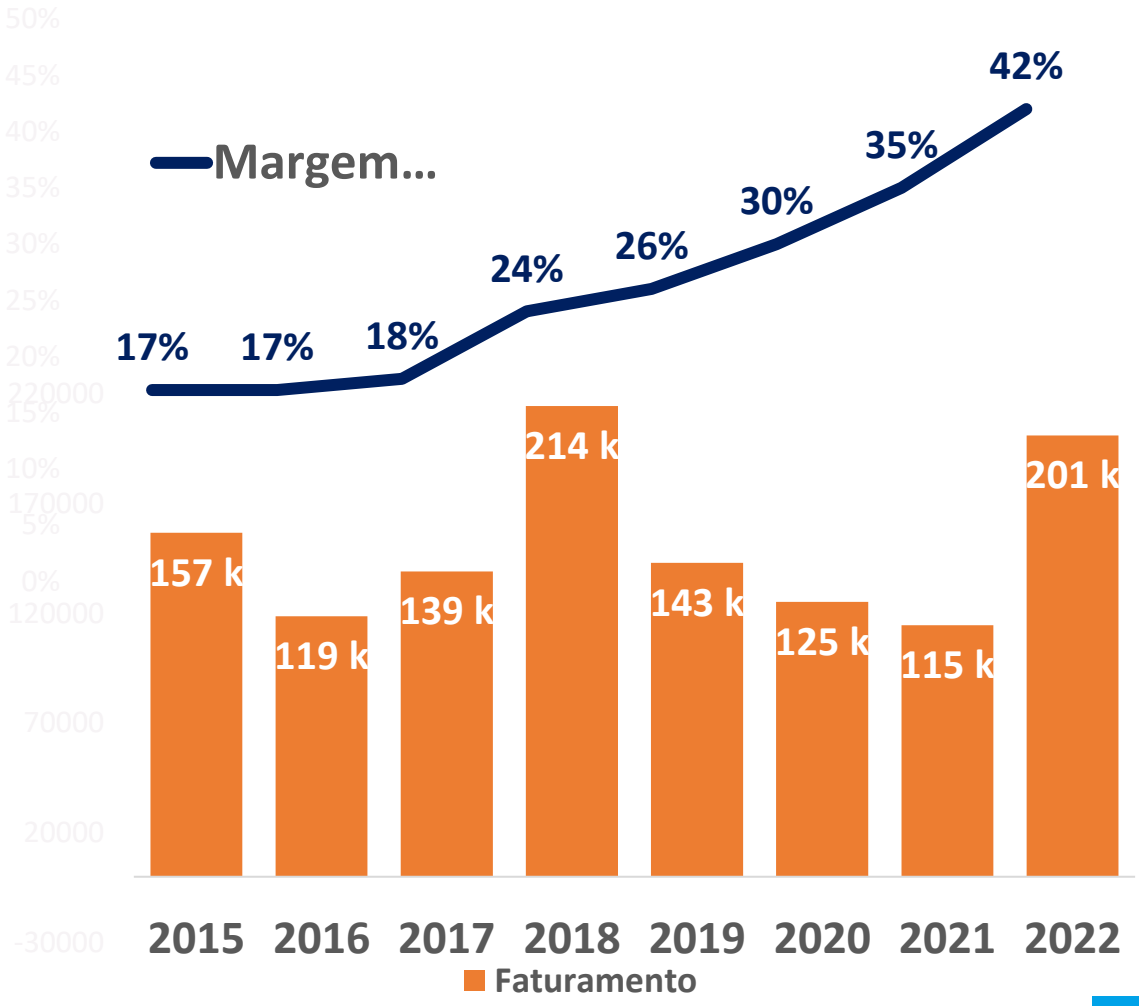




Módulo 8 – Reduzindo o esforço para entender sua apresentação (eixo Y começando no zero e eixos secundários)

Ou podemos separar em 2 gráficos:





Módulo 8 – Melhorando o seu visual (Proximidade e Similaridade)

Quando se fala em **melhorar o visual** estamos falando diretamente sobre evitar a saturação, ou seja, tudo o que está no seu gráfico ou imagem que ocupa espaço mas não aumenta o entendimento do leitor.

Por exemplo, ao inserir um texto em um visual que não acrescenta no entendimento para o leitor.

Tudo que estiver na sua **apresentação precisa ter um objetivo** muito claro, você precisa colocar aquilo de forma eficiente.

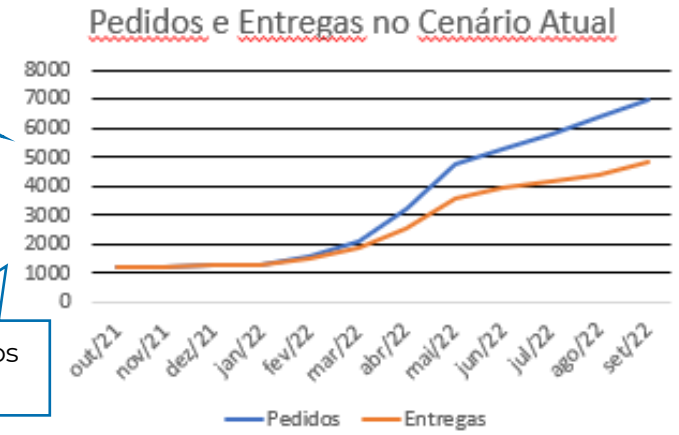
No próximo slide temos um exemplo de saturações no gráfico.

Saturação são elementos visuais que **ocupam espaço**, mas não aumentam o entendimento. Por exemplo:

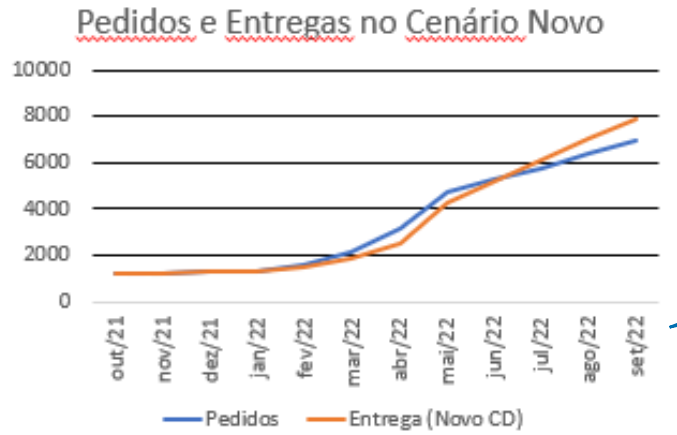
Muito texto

A situação é extremamente crítica pois nossas vendas estão crescendo muito e nossa distribuição não consegue acompanhar. Dessa forma a entrega está ficando muito atrasada e a previsão é que os pedidos de abril possam levar muito mais tempo para serem entregues, prejudicando muito os nossos clientes
Por isso, é fundamental o investimento no novo Centro de Distribuição para resolver esse problema. Com o novo CD a situação será normalizada até setembro!

Linhas de grade muito fortes



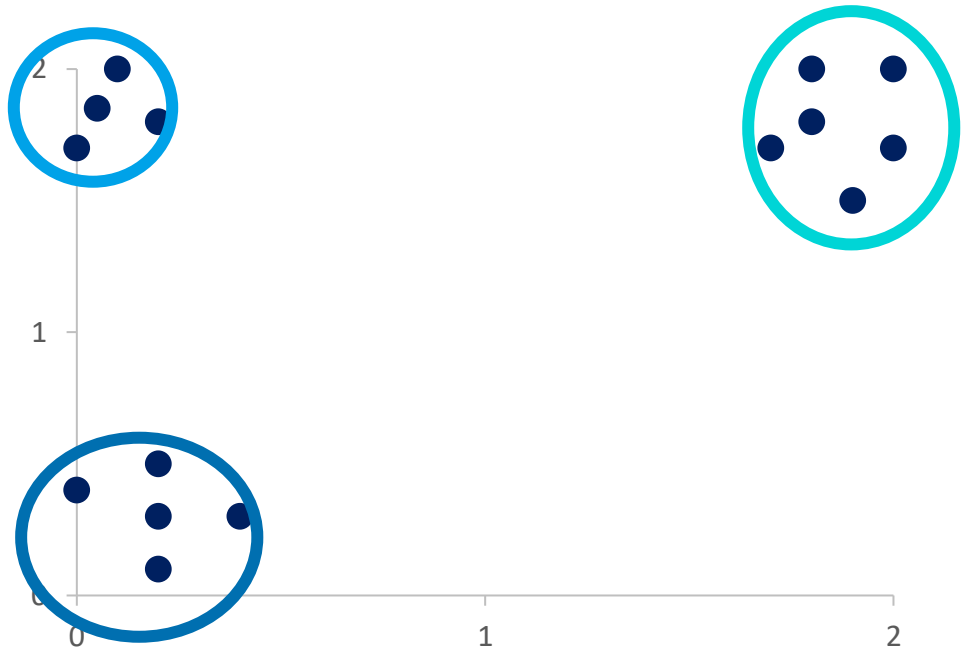
Eixo y com muitos zeros



Gráficos com eixos diferentes

Existem alguns princípios que são usados para reduzir essa saturação e melhorar a visualização do que está sendo apresentado.

O princípio de **proximidade** diz que o nosso cérebro entende **objetos próximos como parte de um grupo**, por exemplo, quando se fala de cluster de cliente ou quando eu quero separar entre tipos de planta.



Módulo 8 – Melhorando o seu visual (Proximidade e Similaridade)

O mesmo se aplica à tabela, quando aumentamos o espaçamento entre as colunas, conseguimos focar mais nas colunas e essa é uma forma de direcionar o usuário para que eu ele leia as informações da forma que queremos.

Produto	Mês	Venda $	Venda Qtd
A	Jan	2546	37
A	Fev	2027	22
A	Mar	2470	31
B	Jan	1787	21
B	Fev	2664	44
B	Mar	2288	46
C	Jan	2531	30
C	Fev	2173	42
C	Mar	2660	38

Produto	Mês	Venda $	Venda Qtd
A	Jan	2546	37
A	Fev	2027	22
A	Mar	2470	31
B	Jan	1787	21
B	Fev	2664	44
B	Mar	2288	46
C	Jan	2531	30
C	Fev	2173	42
C	Mar	2660	38

Módulo 8 – Melhorando o seu visual (Proximidade e Similaridade)

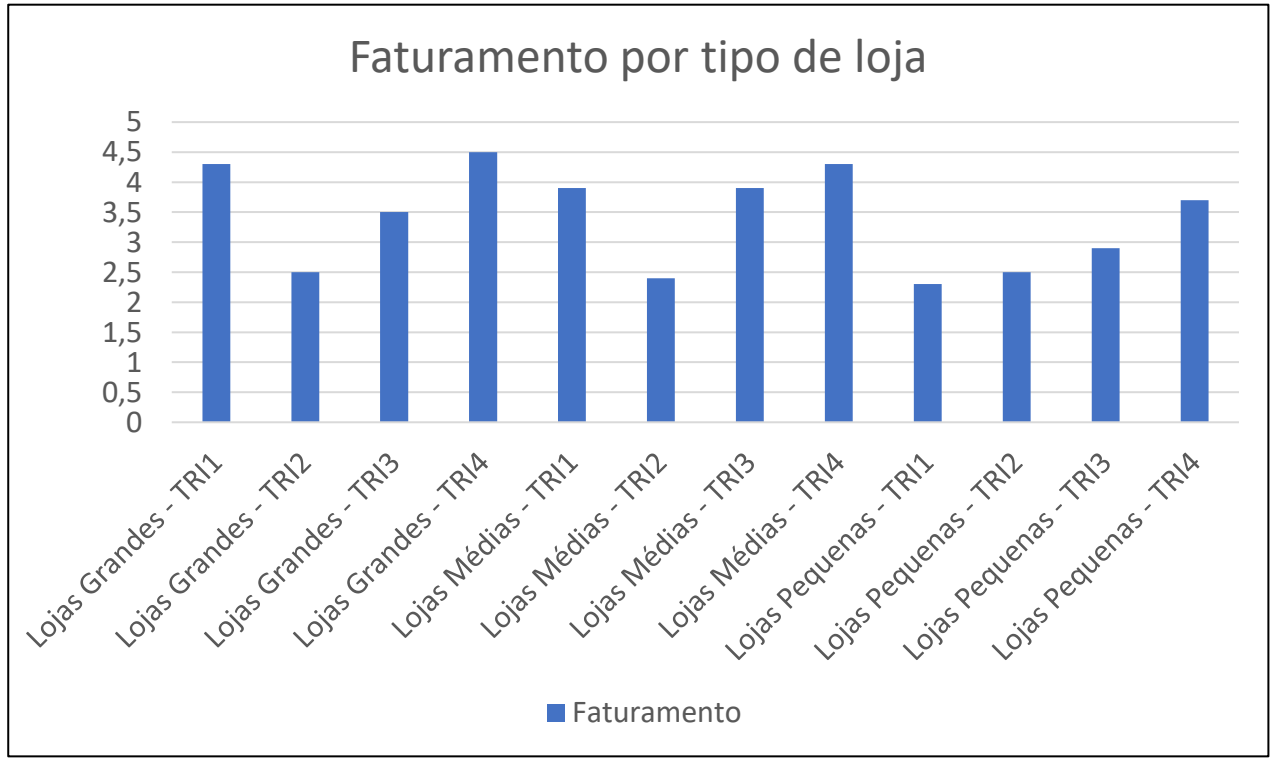
Agora, por outro lado, queremos focar nas linhas. O leitor primeiro vai ler as informações do produto A, depois o B e por último o C.

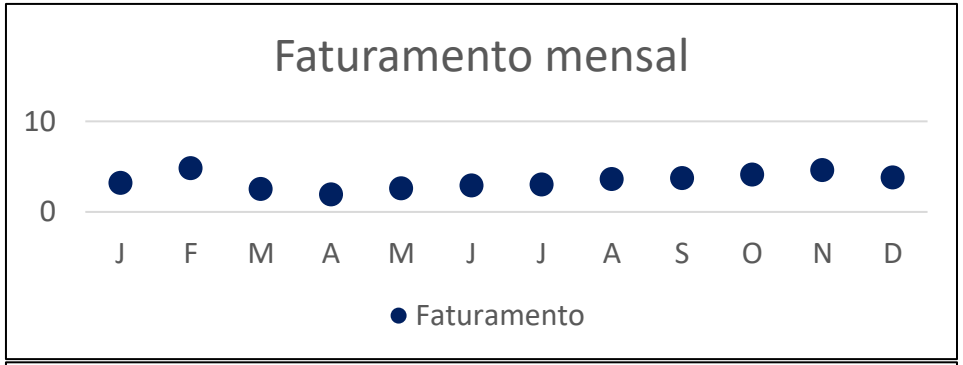
Para fazer essa lógica basta **aumentarmos o espaçamento das linhas**.

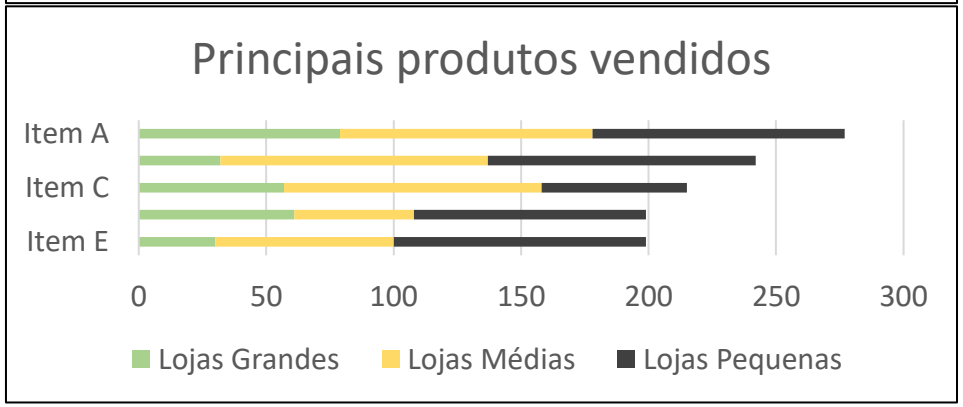
Além disso, a proximidade entre o A, B e o C faz com que o leitor siga essa lógica.

Produto	Mês	Venda $	Venda Qtd
A	Jan	2546	37
A	Fev	2027	22
A	Mar	2470	31
B	Jan	1787	21
B	Fev	2664	44
B	Mar	2288	46
C	Jan	2531	30
C	Fev	2173	42
C	Mar	2660	38

Entendemos objetos fisicamente próximos como parte de um grupo.





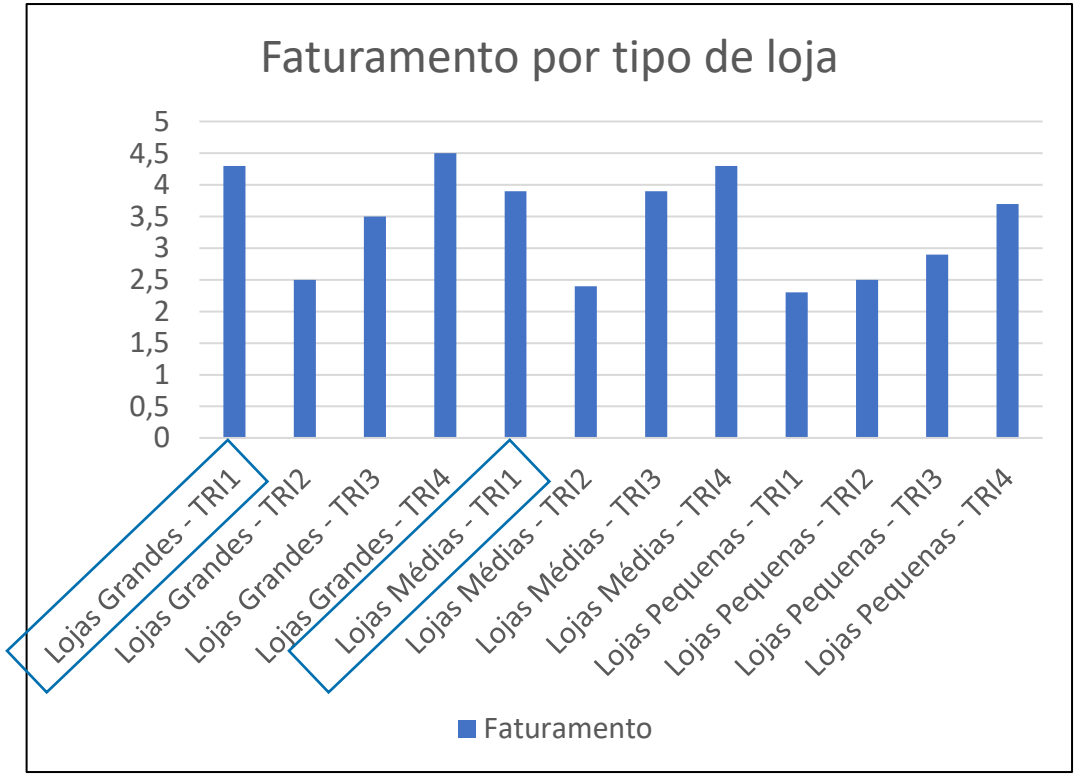


Vamos utilizar esse princípio para melhorar o gráfico ao lado.

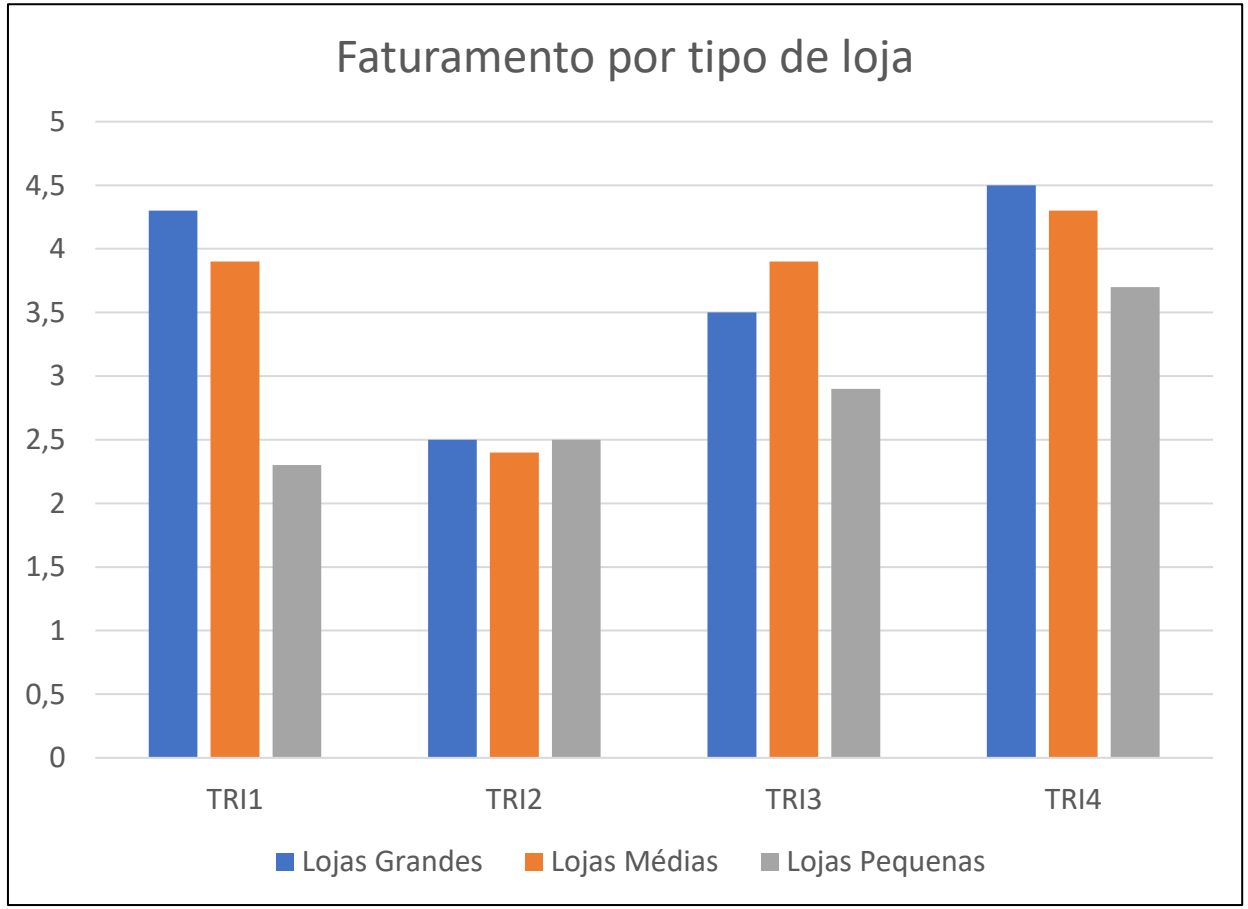
Observem que no eixo x, as informações estão por cada loja e em cada trimestre.

Porém, não é favorável para a leitura ler o 4° Trimestre da Loja grande e em seguida ler o 1° da Loja Média.

Além de não ter uma separação por cor para lojas diferentes.



Módulo 8 – Melhorando o seu visual (Proximidade e Similaridade)

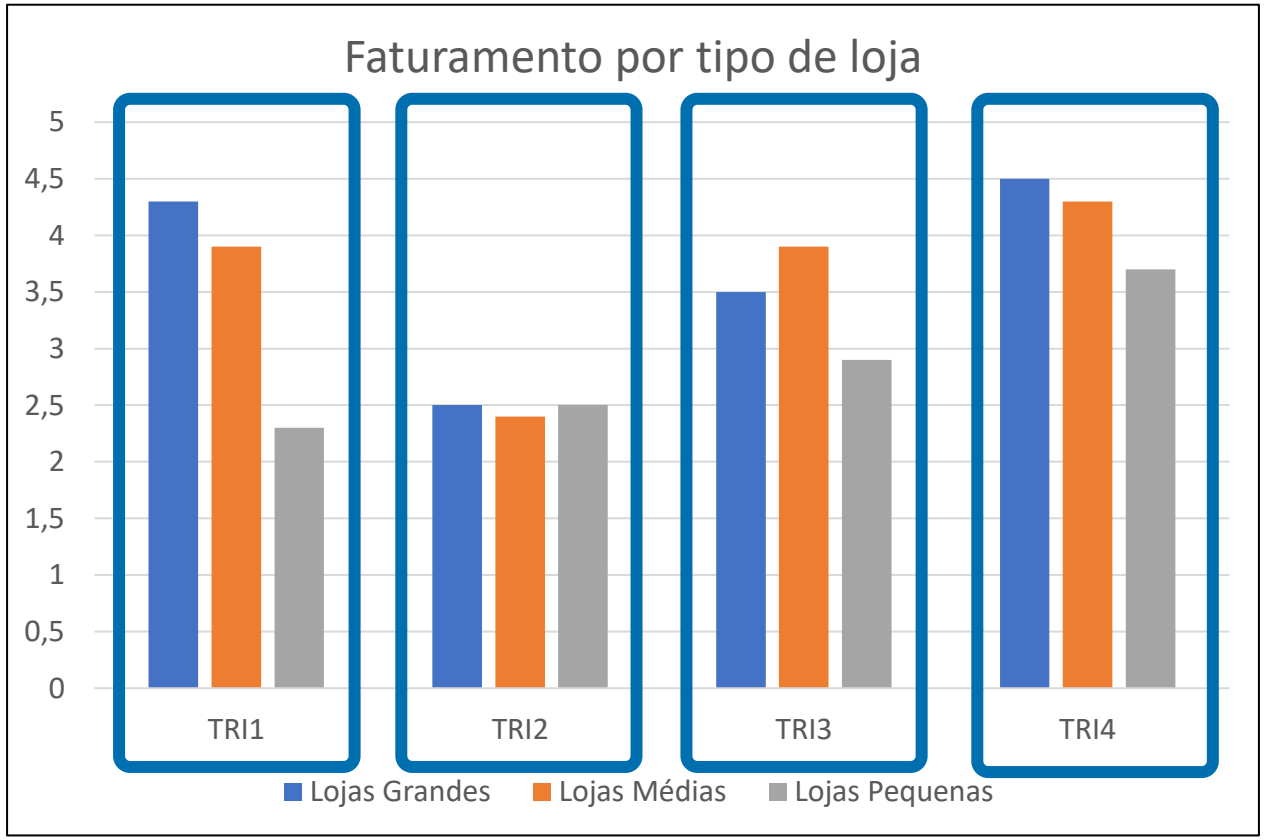


Utilizando o princípio da **proximidade** agora temos uma linha temporal: trimestres 1, 2 e 3.

As informações foram **aproximadas** e colocadas com uma **lógica temporal** entre elas.

Dessa forma fica **mais fácil de entender o gráfico** e com uma visualização mais amigável.

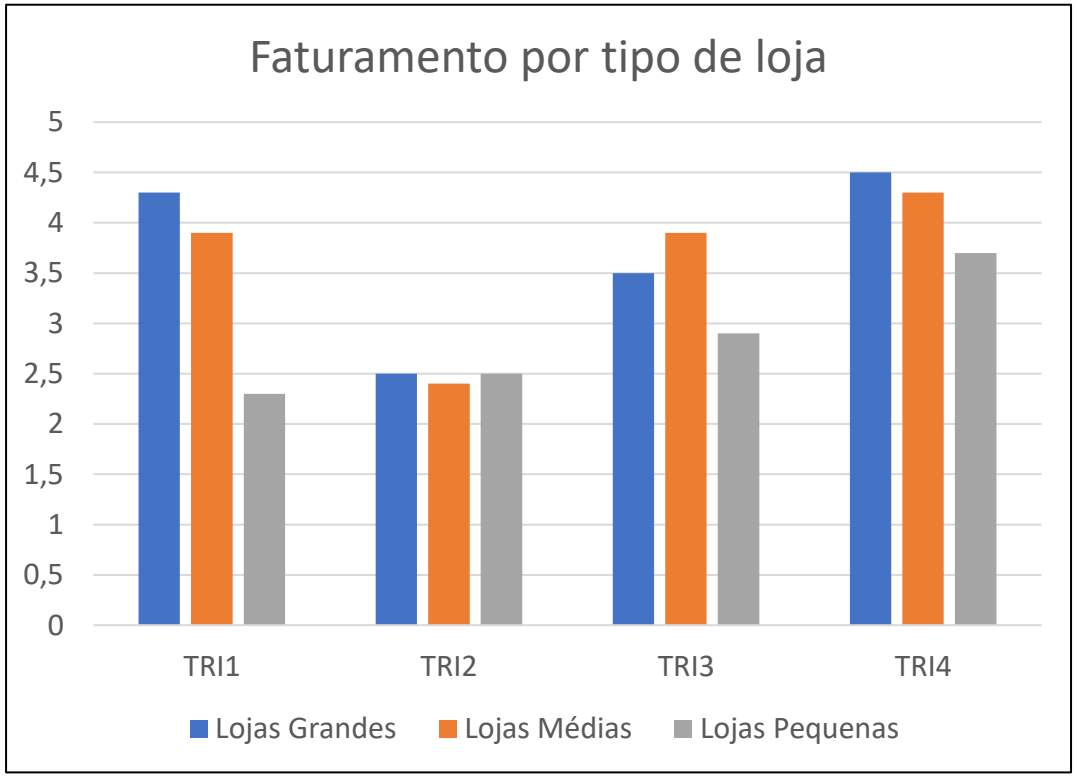
Uma outra forma de utilizar esse princípio é agrupando pelas lojas



A forma como escolher e separar depende do que queremos mostrar.

Por exemplo, se o foco é evolução no tempo, então provavelmente o uso da linha temporal por trimestre é a melhor opção.

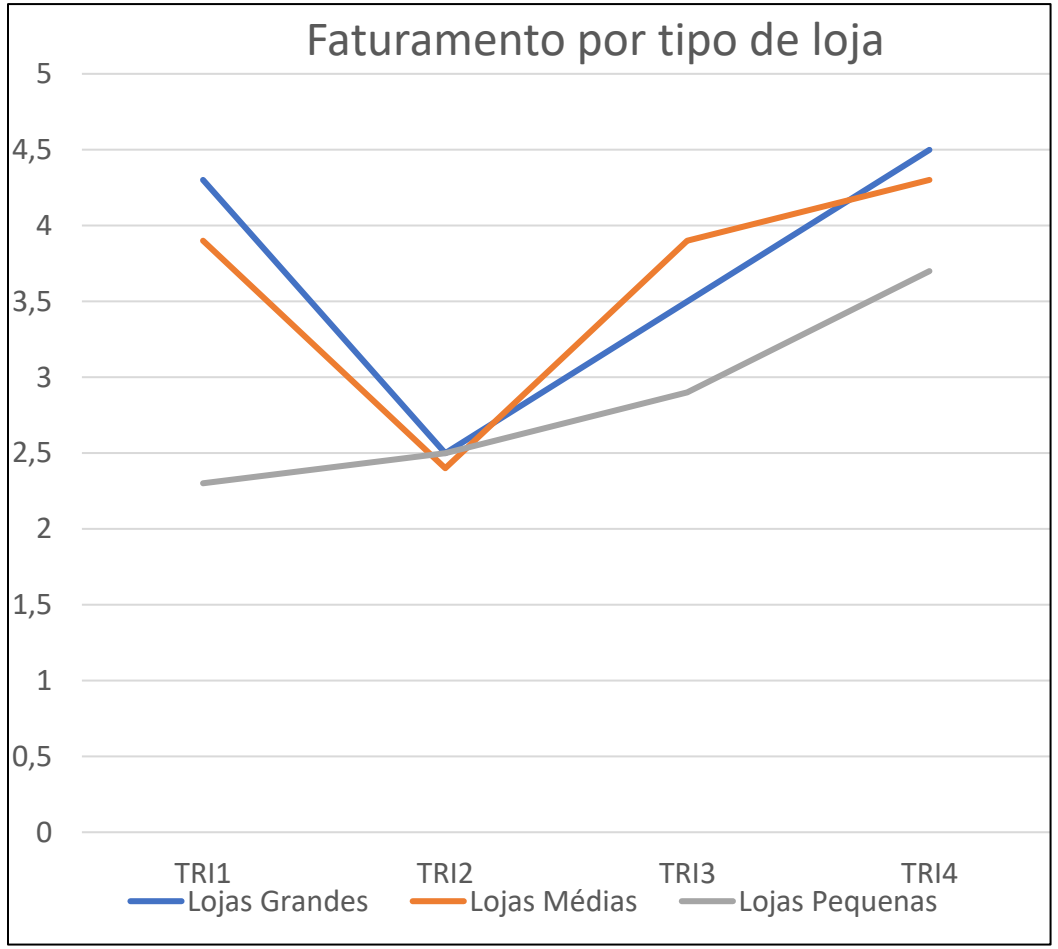
Por outro lado, se o foco é nas lojas, o agrupamento por lojas é a melhor opção.



Módulo 8 – Melhorando o seu visual (Proximidade e Similaridade)

Se o objetivo é **mostrar a evolução**, a melhor forma é a separação por loja utilizando **gráfico de linhas**.

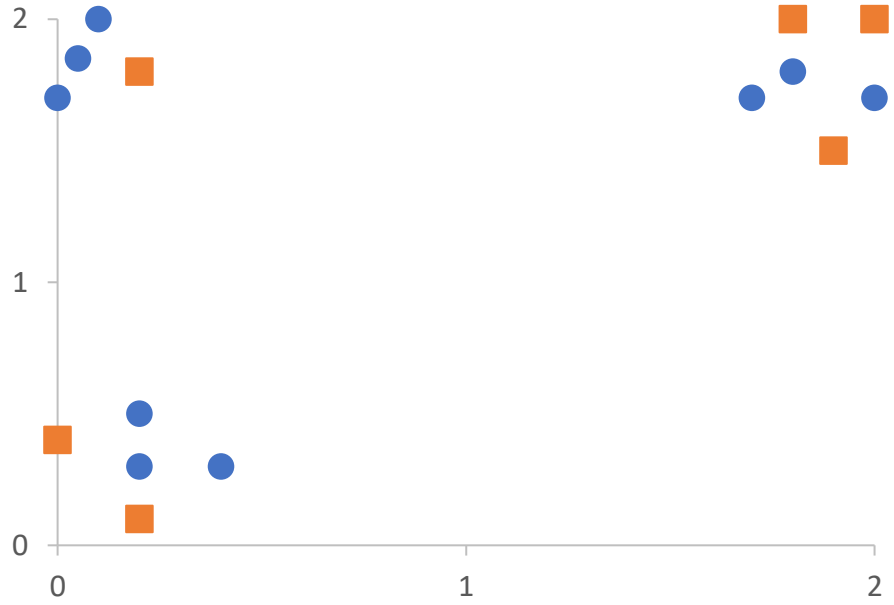
Faça isso de forma consciente e use esse princípio para mostrar o seu objetivo com uma visualização mais amigável.



Objetos com a mesma cor, tamanho e forma são percebidos como relacionados.

Por mais que os pontos estejam próximos, quando usamos esse princípio da **similaridade**, o nosso cérebro entende que existe uma hierarquia.

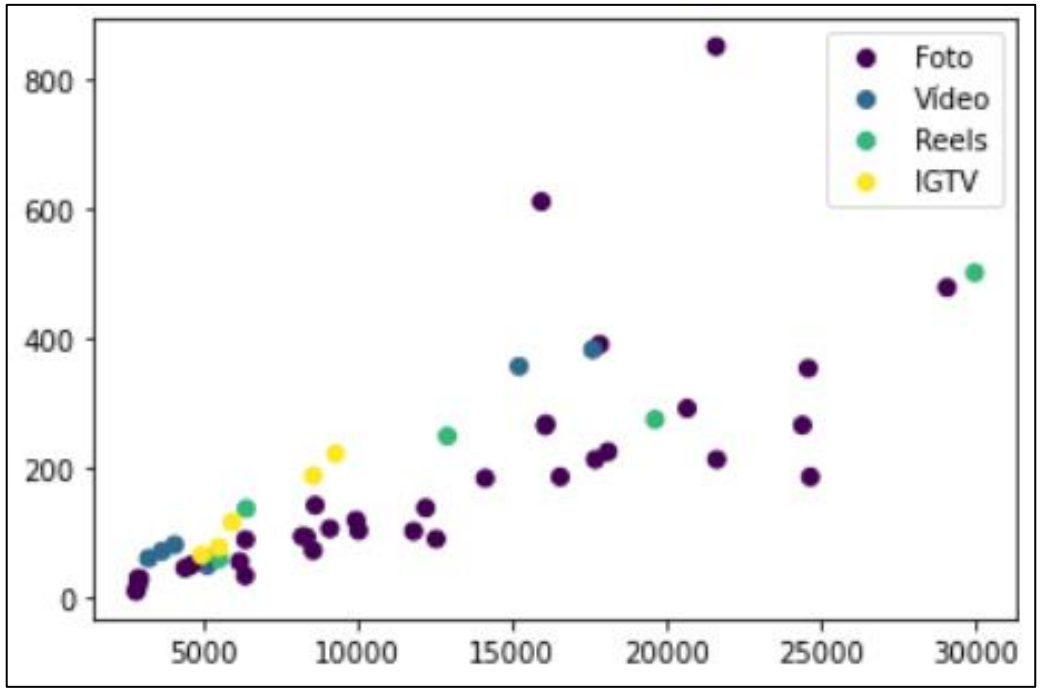
Primeiro observamos os **similares** e depois os que estão próximos. Portanto, quando a gente observa o gráfico ao lado, entendemos que o ponto vermelho é uma coisa e o azul outra.



Observe como ao organizar as linhas dessa forma fica muito mais fácil de entender e mais confortável para o usuário realizar a leitura.

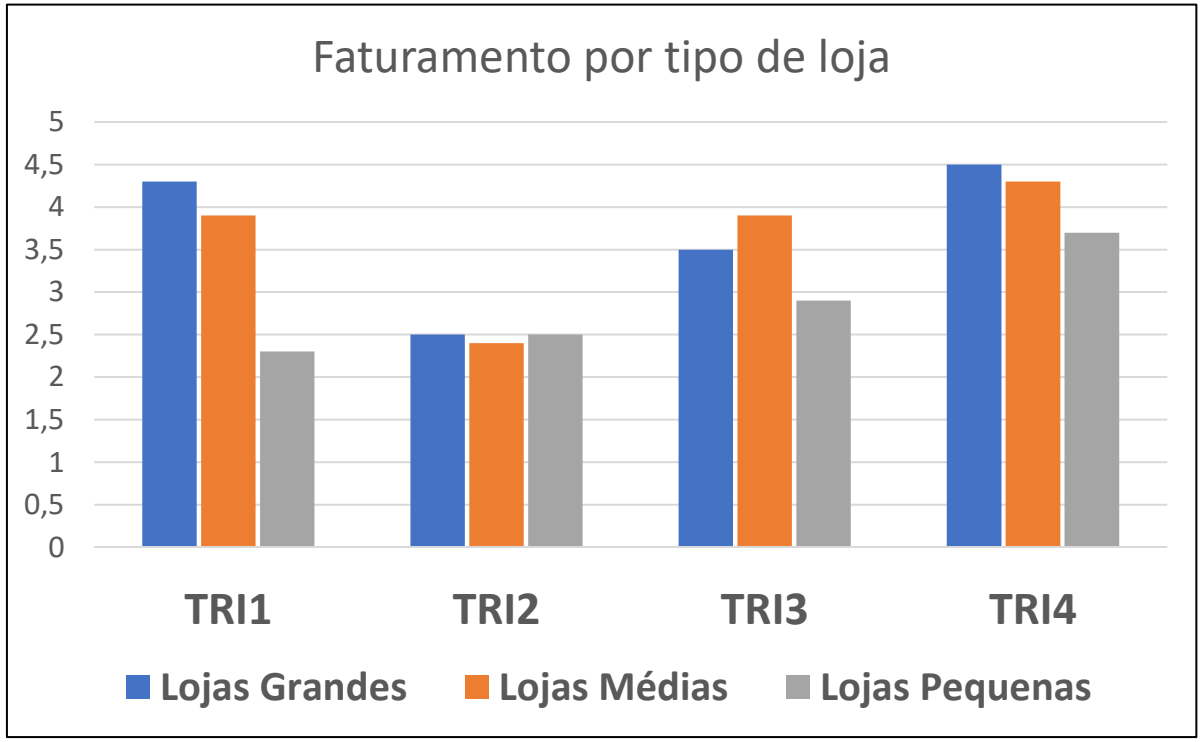
Produto	Mês	Venda $	Venda Qtd
A	Jan	2546	37
A	Fev	2027	22
A	Mar	2470	31
B	Jan	1787	21
B	Fev	2664	44
B	Mar	2288	46
C	Jan	2531	30
C	Fev	2173	42
C	Mar	2660	38

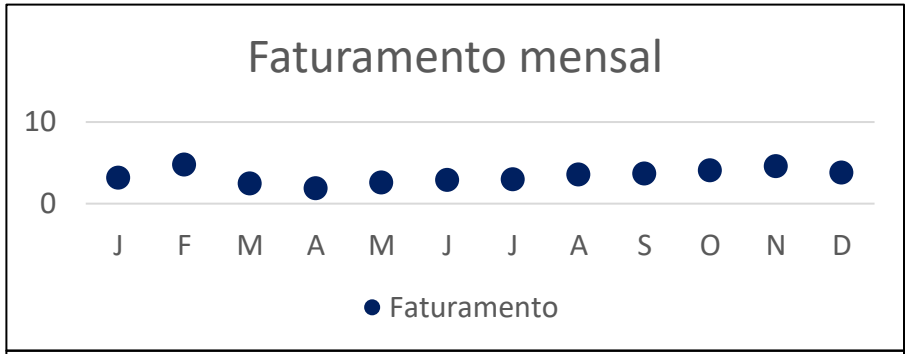
No gráfico abaixo colocamos **cores diferentes para tipos distintos** de mídia e dessa forma **facilita a leitura do gráfico**.

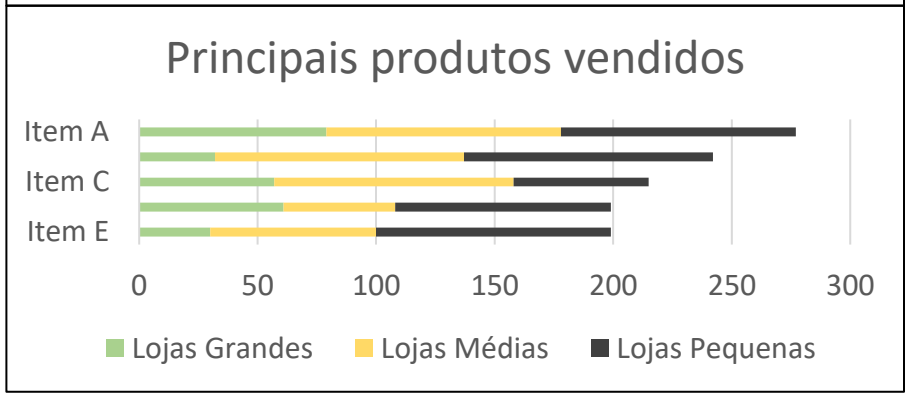


Módulo 8 – Melhorando o seu visual (Proximidade e Similaridade)

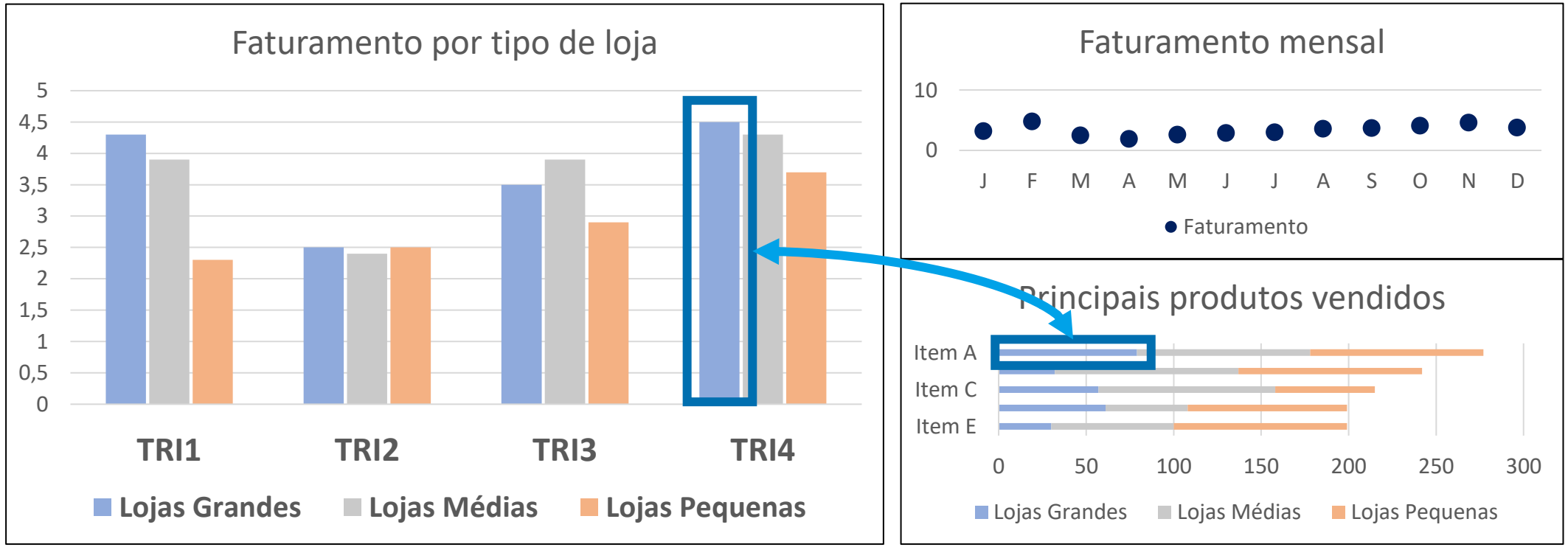
Observe que no visual abaixo temos 6 cores para 3 lojas.







Uma forma de melhorar a visualização é **padronizar as cores para as lojas** nos gráficos.



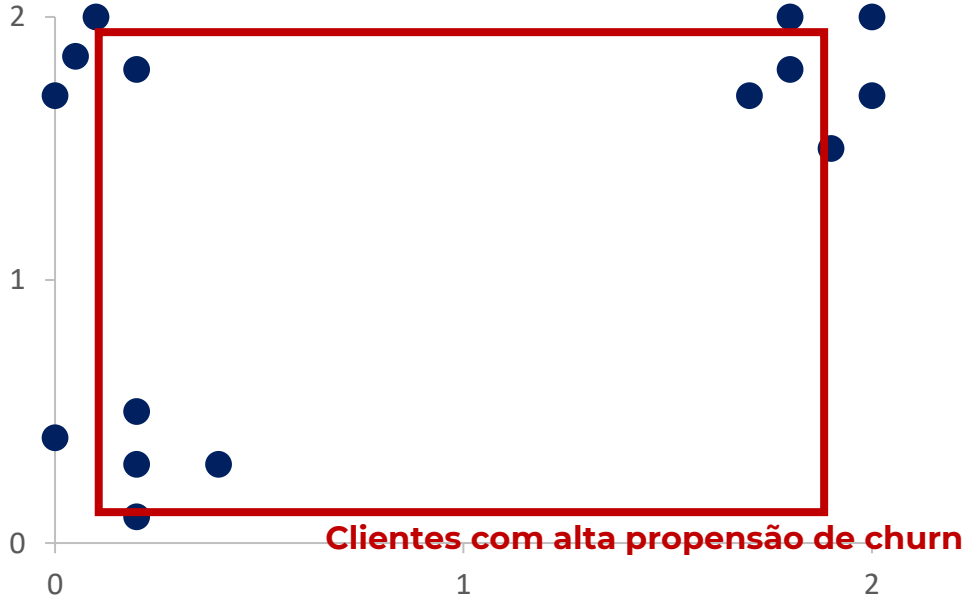
Com isso, nós utilizamos 2 princípios muito importantes para melhorar o visual do gráfico:

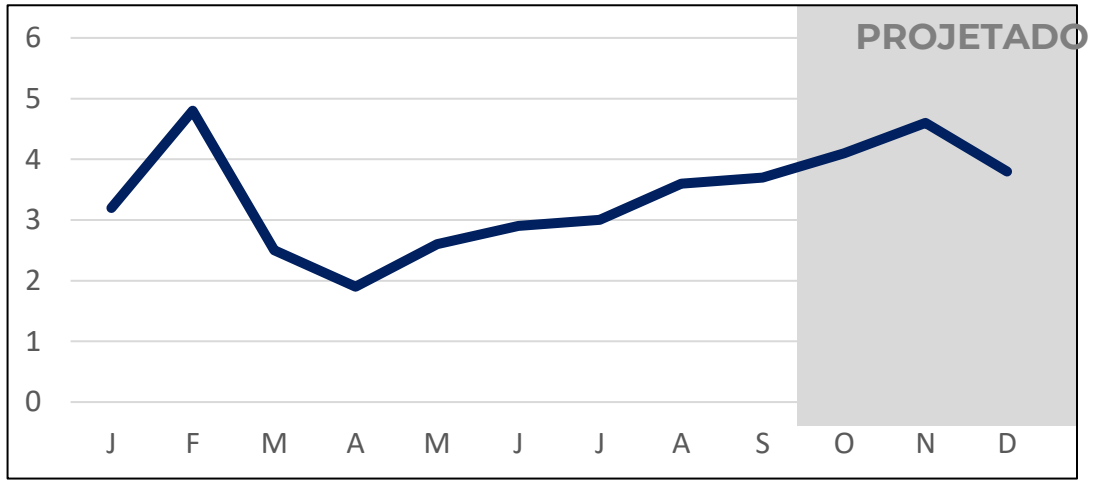
Proximidade: entendemos **objetos fisicamente próximos** como parte de um grupo.

Similaridade: objetos com a **mesma cor**, **tamanho**, **forma** são percebidos como relacionados.

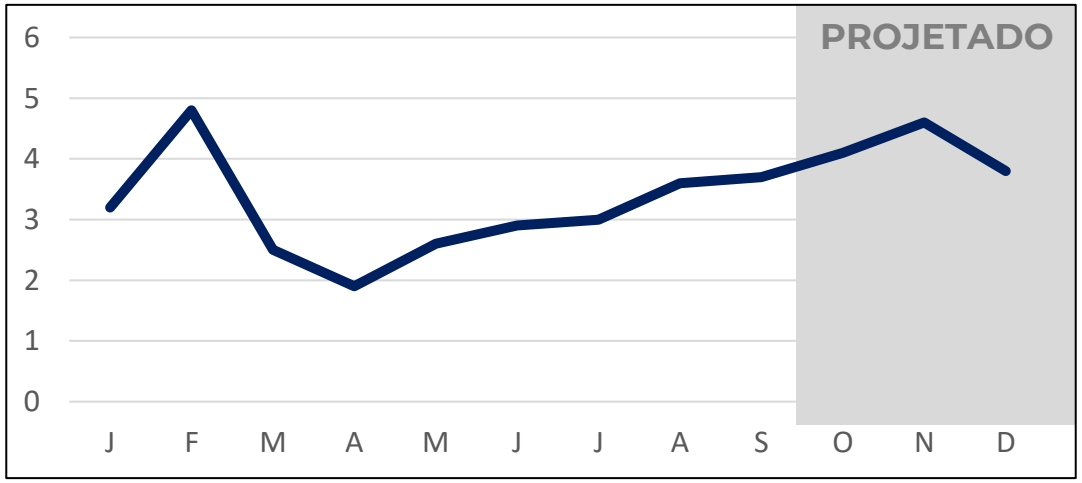
Além de notar que existe uma hierarquia em primeiro vermos a similaridade e depois a proximidade.

Um outro princípio é o acercamento que entende objetos fisicamente delimitados como fazendo parte de um grupo.



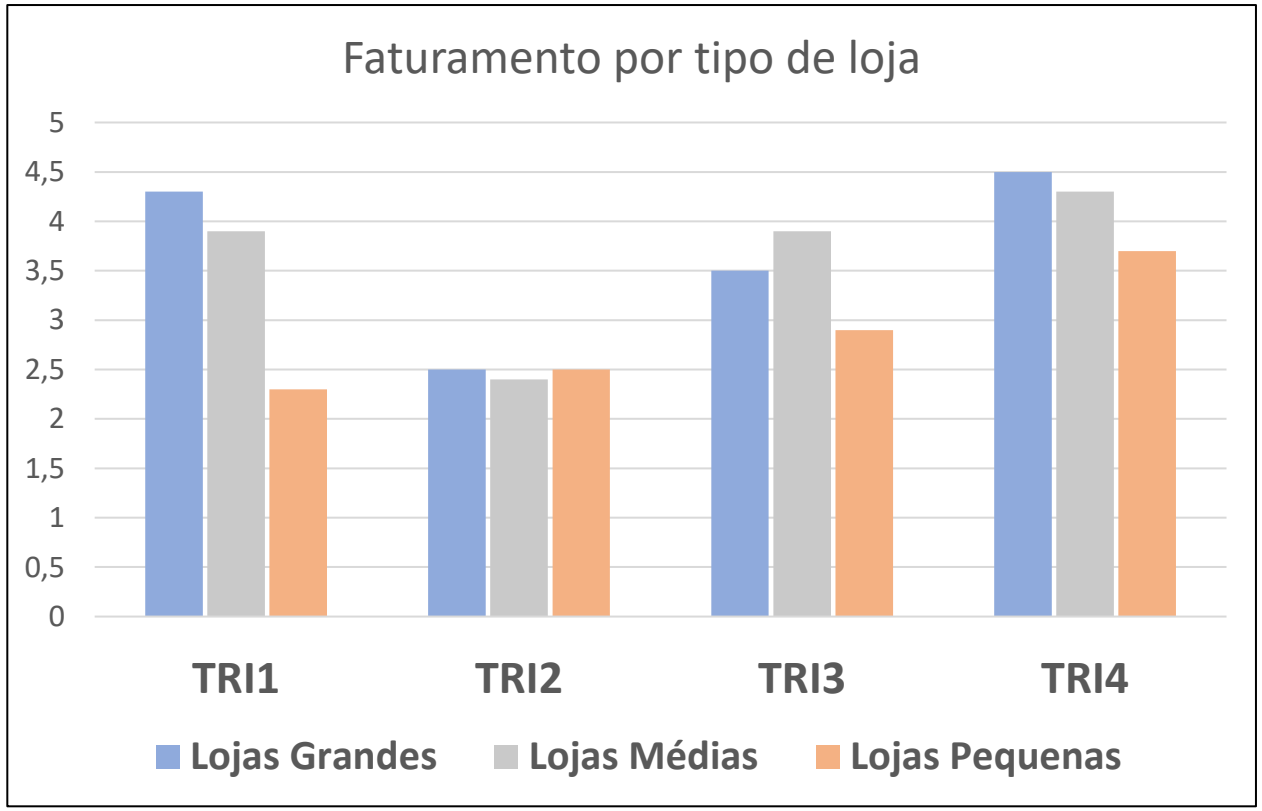


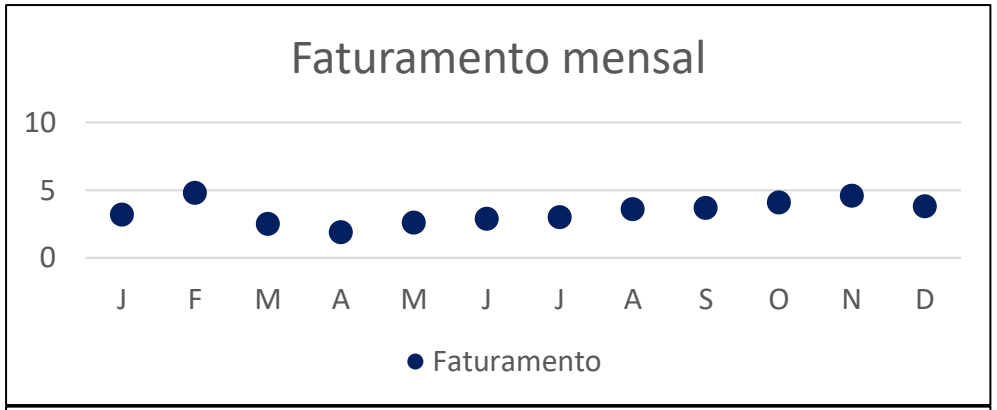
Utilizamos esse princípio para dar ênfase em alguma área, por exemplo, separar o realizado do projetado.

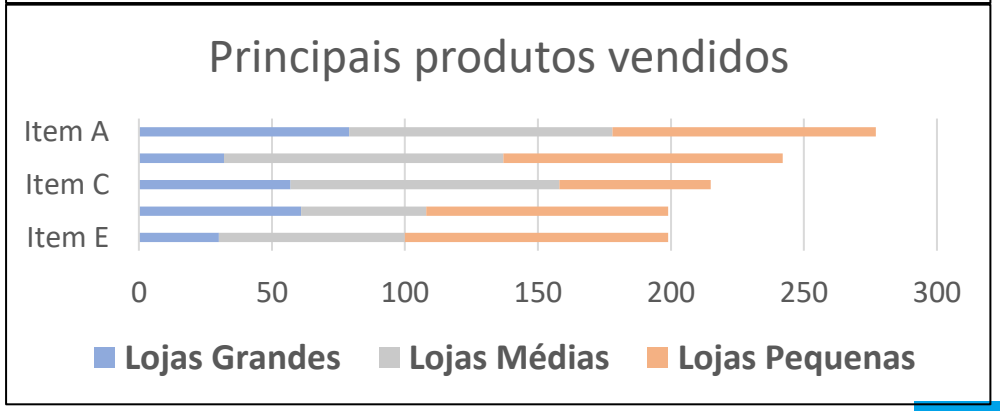


Módulo 8 – Melhorando o seu visual (Acercamento, Fechamento, Continuidade e Conexão)

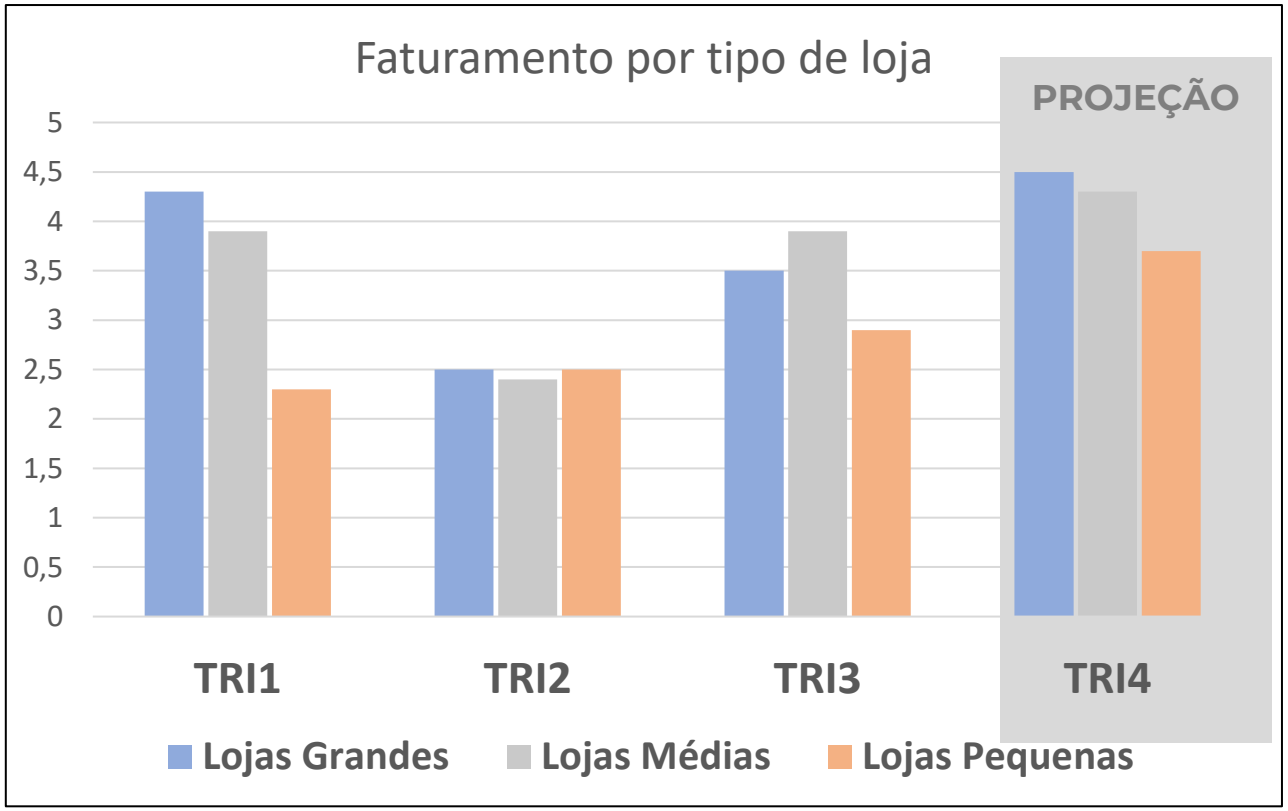
Como que poderíamos usar o acercamento nesse gráfico?

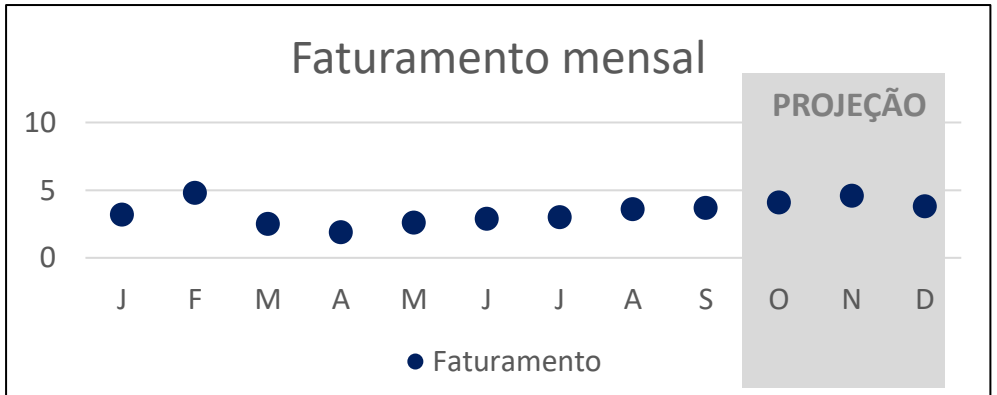


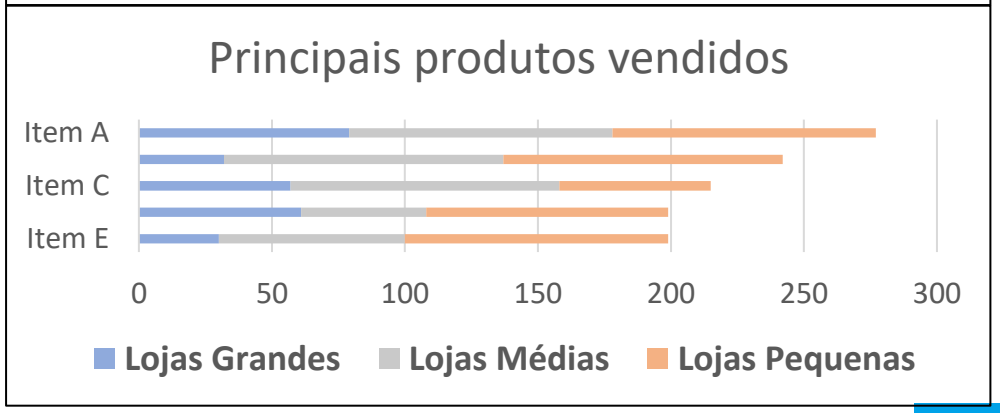




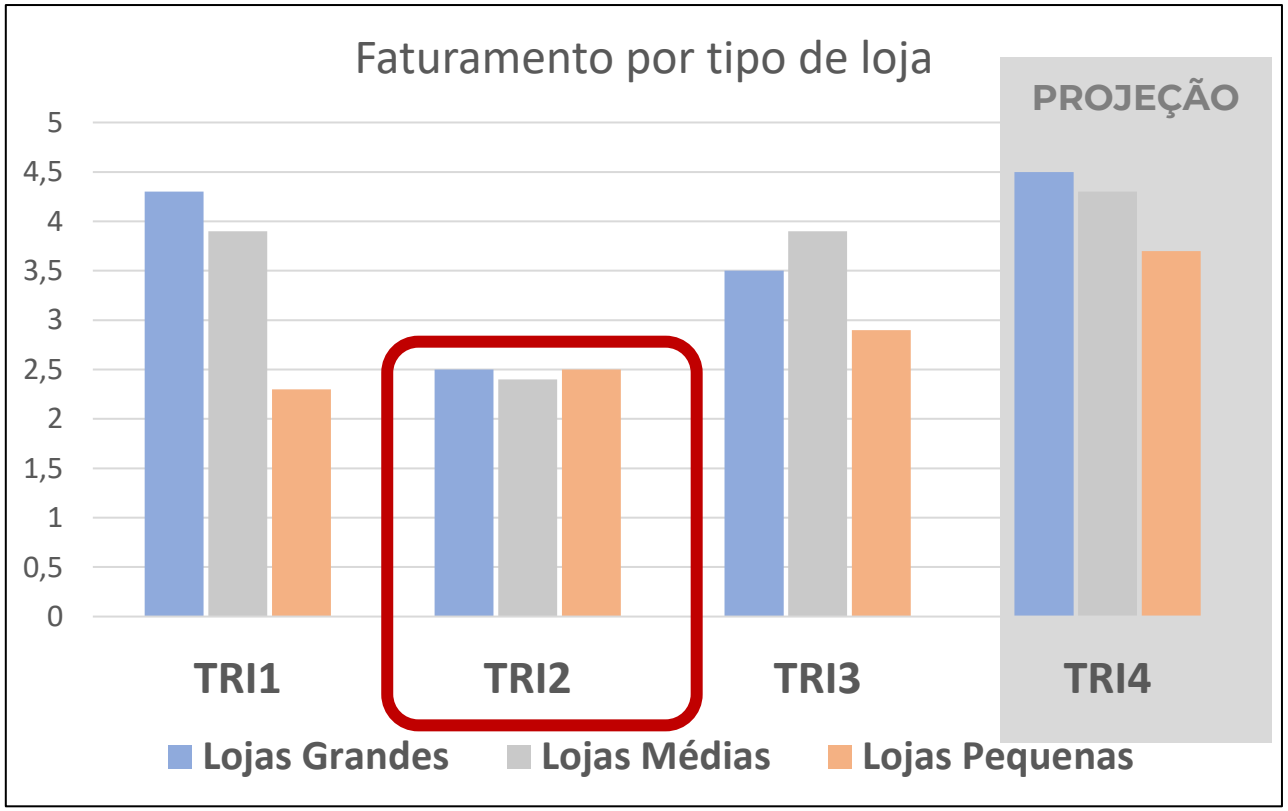
Podemos considerar que o 4° trimestre ainda não finalizou

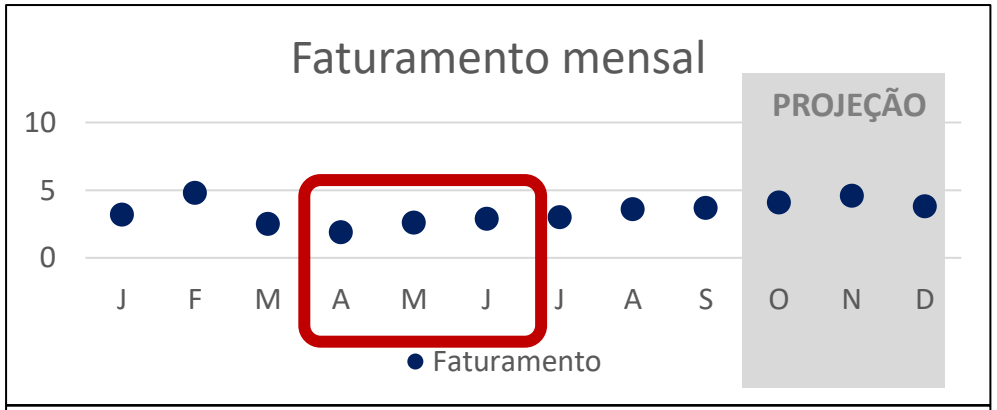


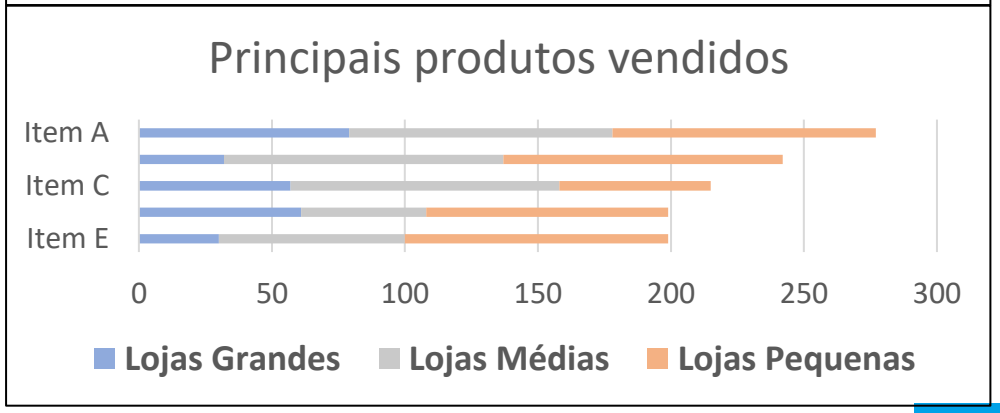




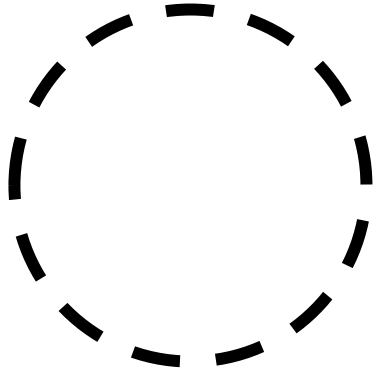
E se quisermos destacar o trimestre com menor faturamento?



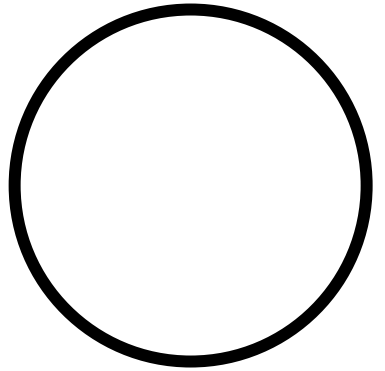




O princípio do **fechamento** diz que quando partes de um todo estão faltando, nossos **olhos preenchem as lacunas**

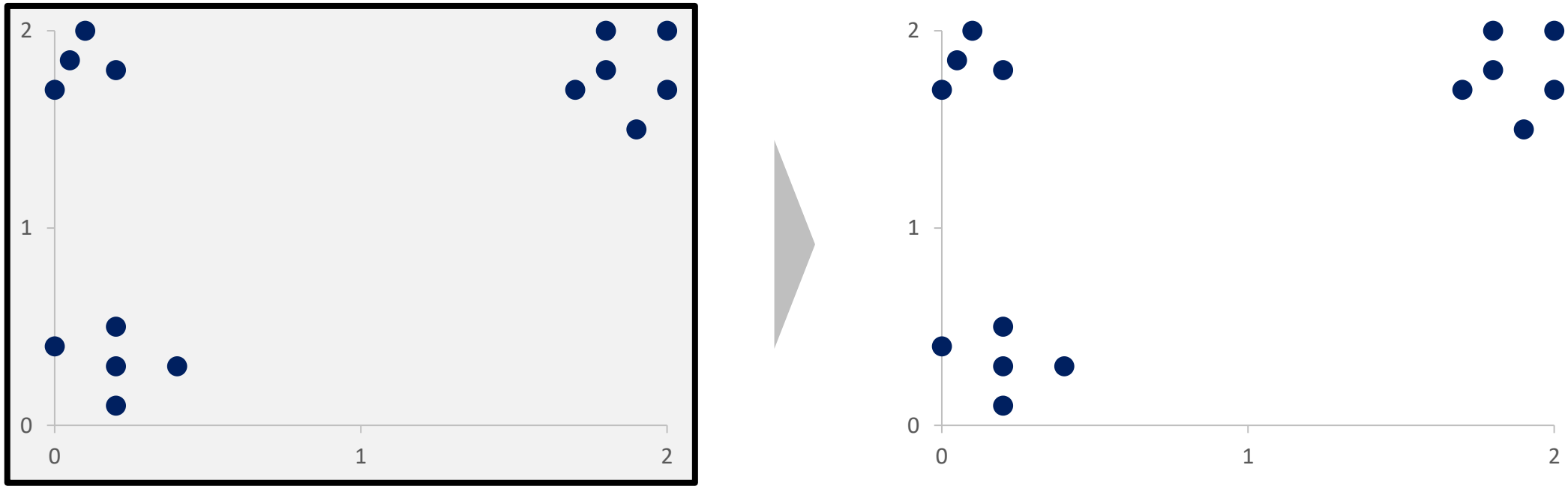


"Nosso cérebro procura simplificar as coisas e tenta **relacionar a algo que já conhecemos**. Tendemos a perceber um conjunto de elementos individuais como uma única forma reconhecível"



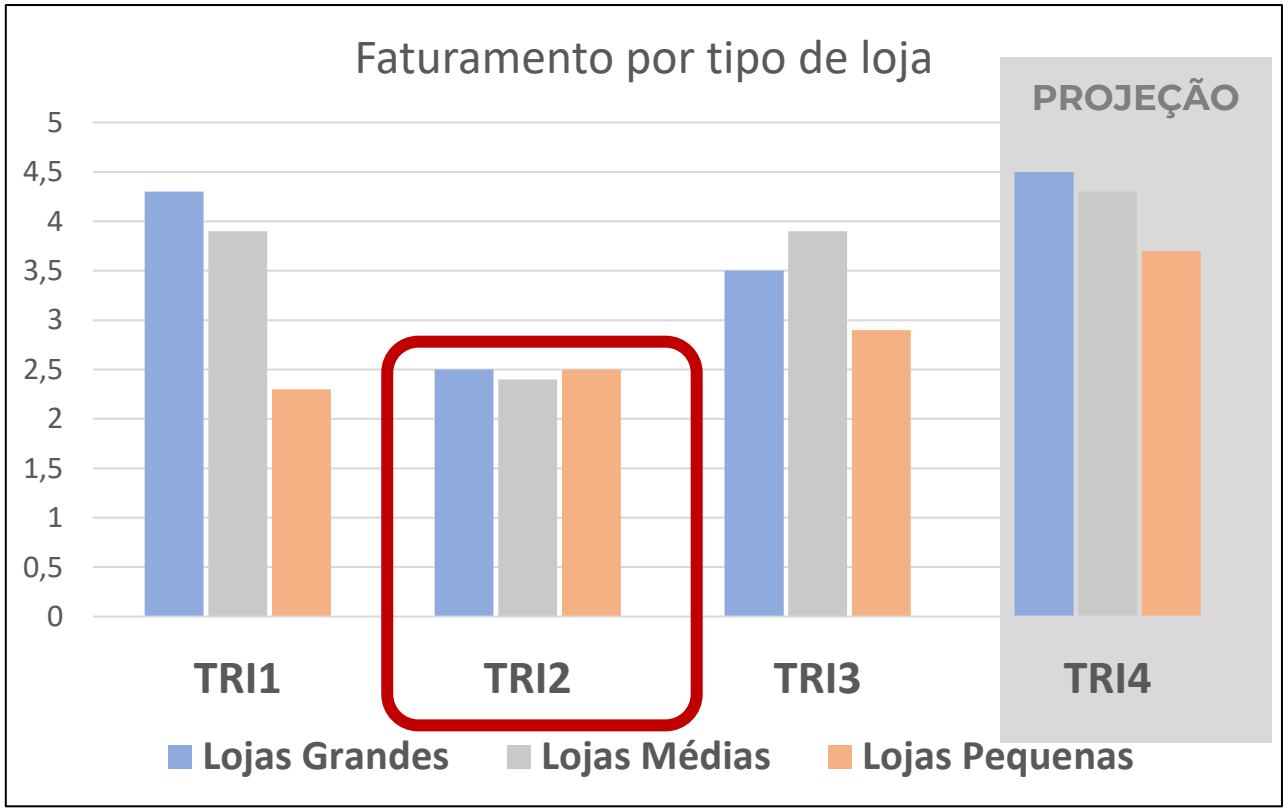
Módulo 8 – Melhorando o seu visual (Acercamento, Fechamento, Continuidade e Conexão)

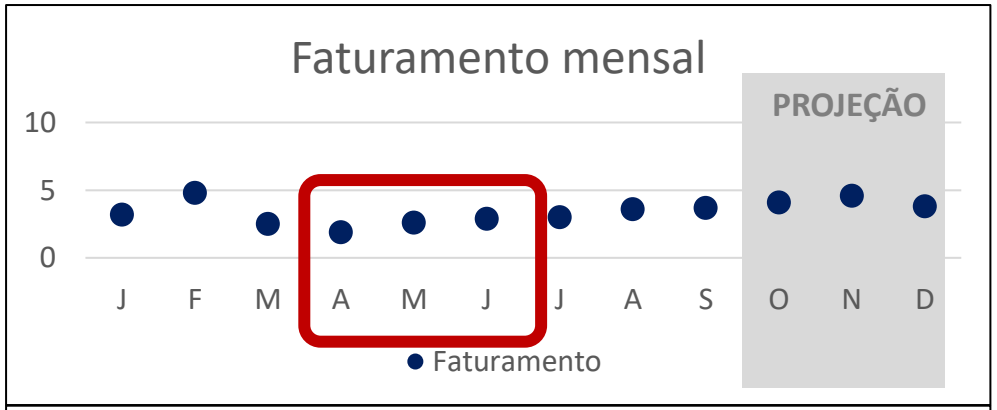
Vamos usar esse princípio para tirar alguns detalhes, por exemplo a borda.

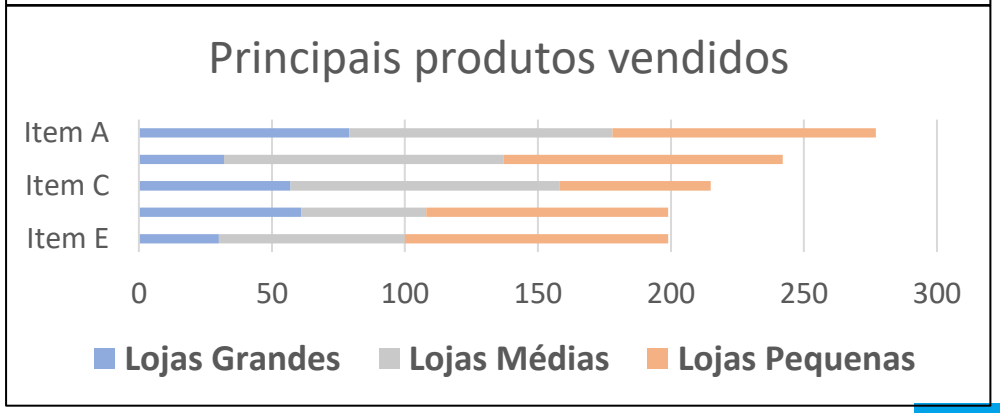


Módulo 8 – Melhorando o seu visual (Acercamento, Fechamento, Continuidade e Conexão)

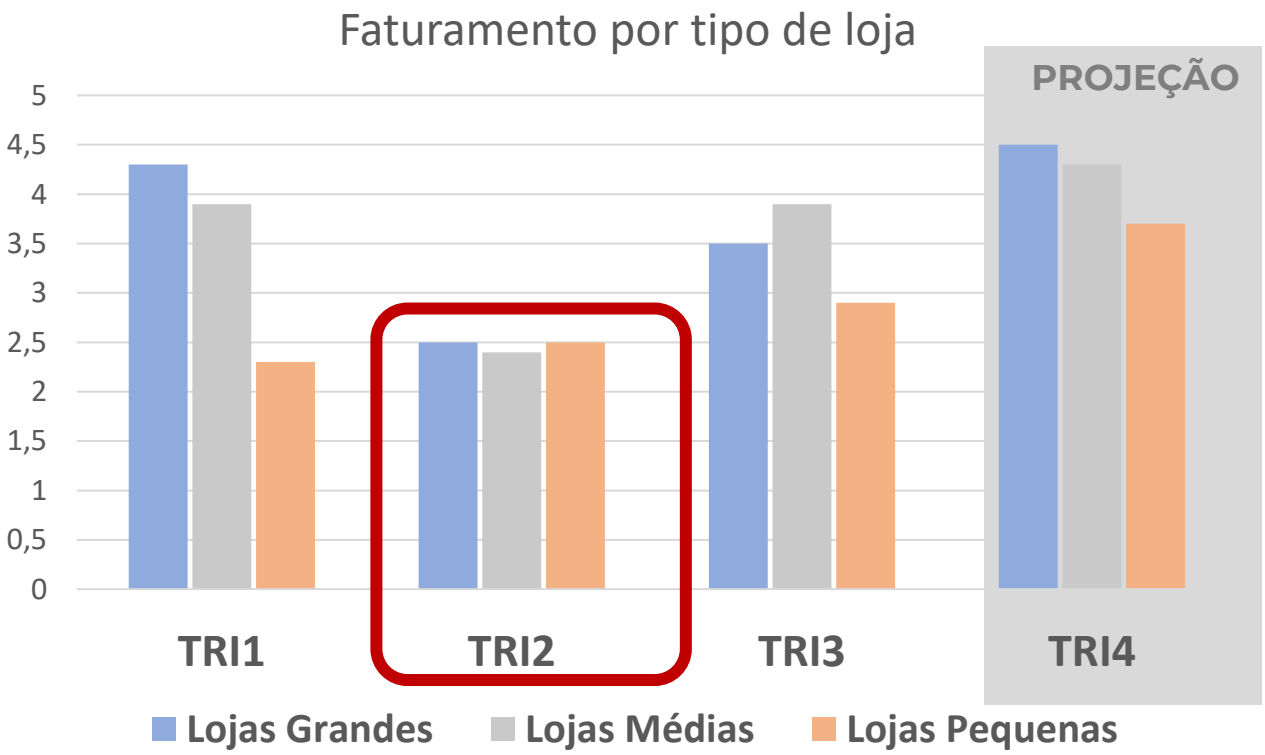
Como a gente poderia melhorar essa nossa apresentação com o fechamento?

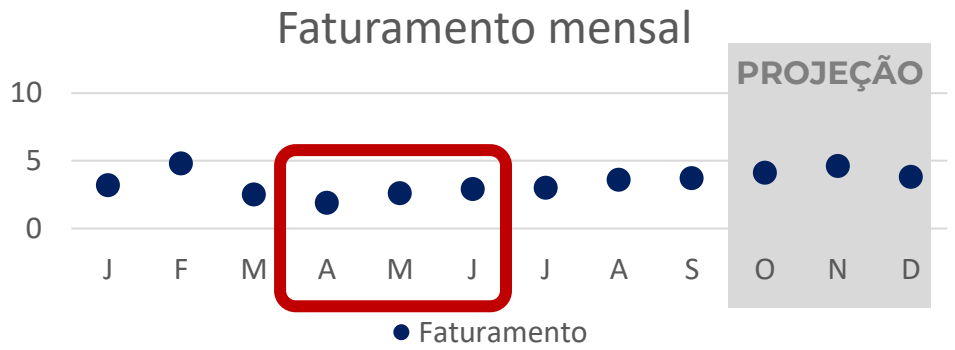


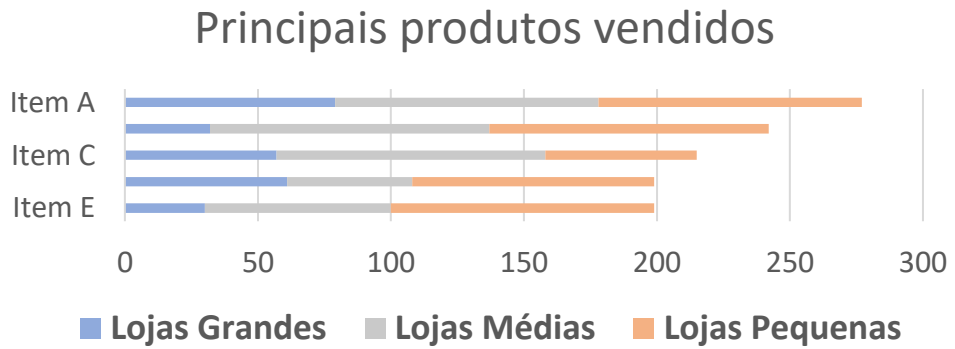




Retirando as bordas, o gráfico fica mais amigável.

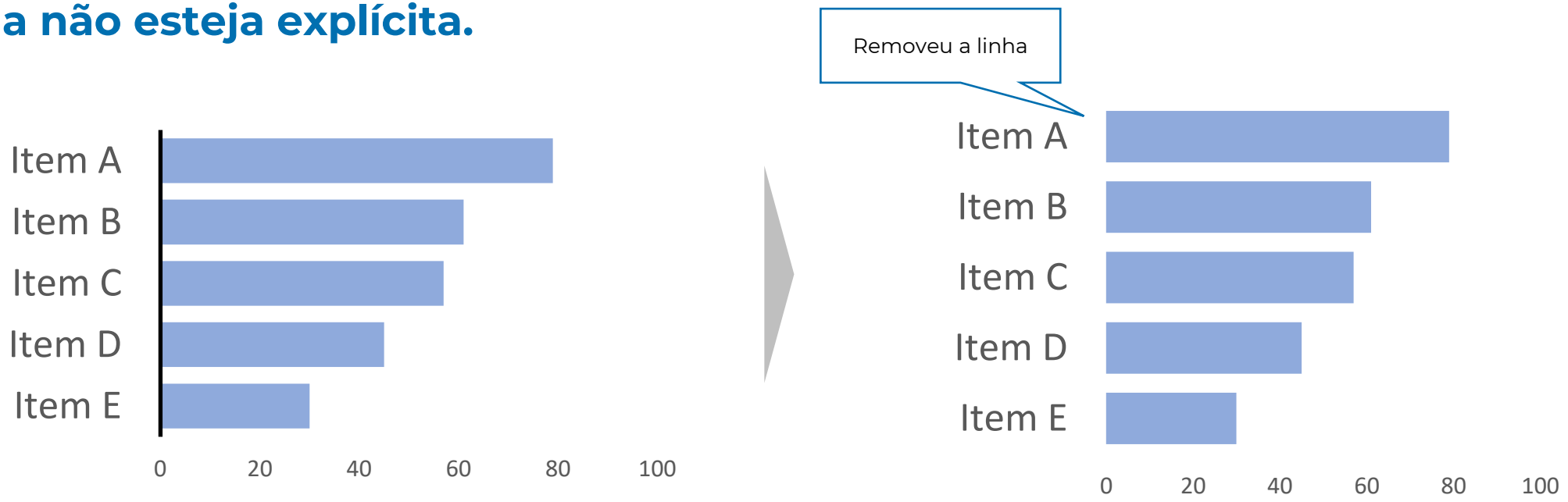




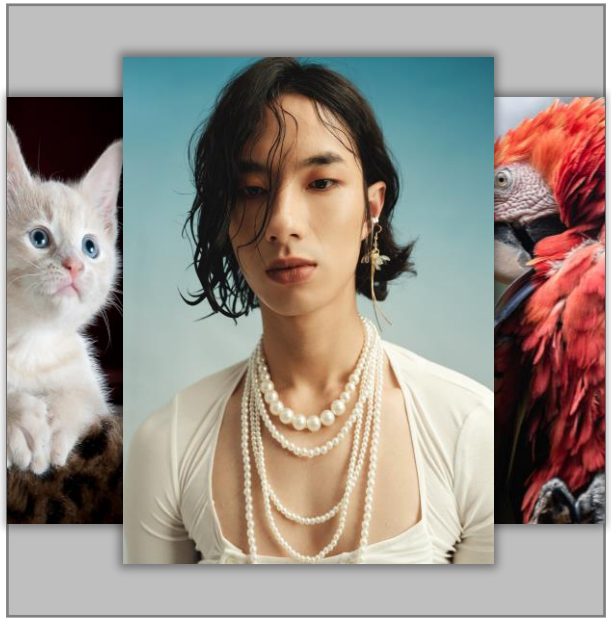


Outro princípio que podemos utilizar para reduzir a saturação é a continuidade.

O princípio da **continuidade** diz que os nossos olhos vão criar **continuidade mesmo que ela não esteja explícita.**

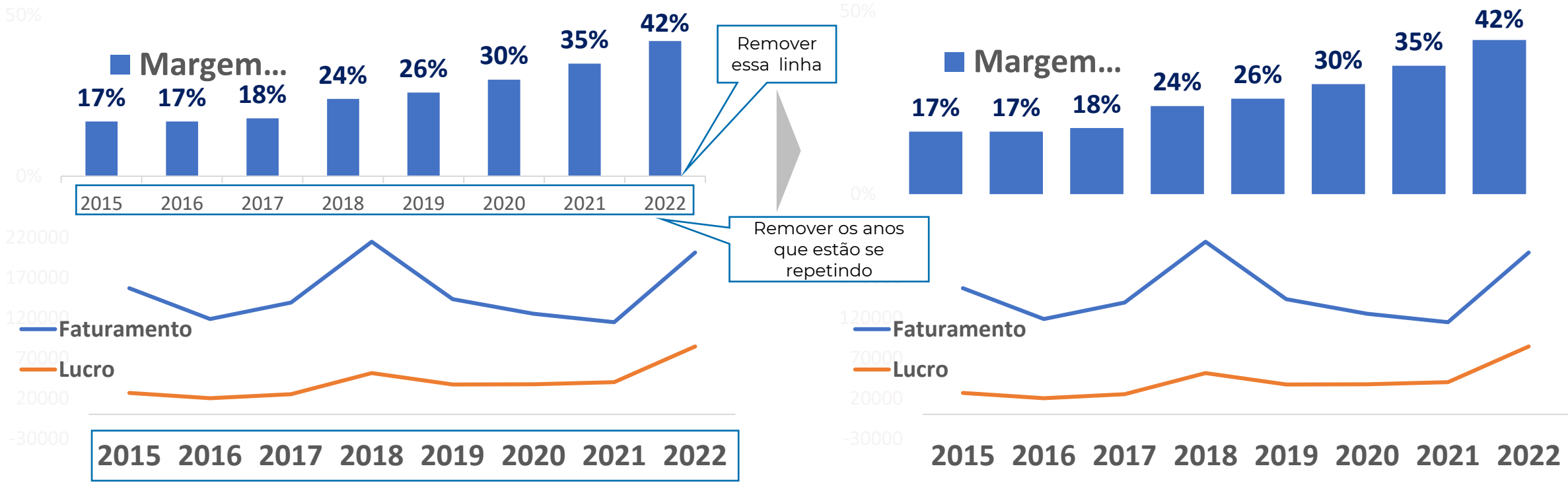


Esse o princípio é muito utilizado em aplicativos, filmes e sites de catálogos de produtos



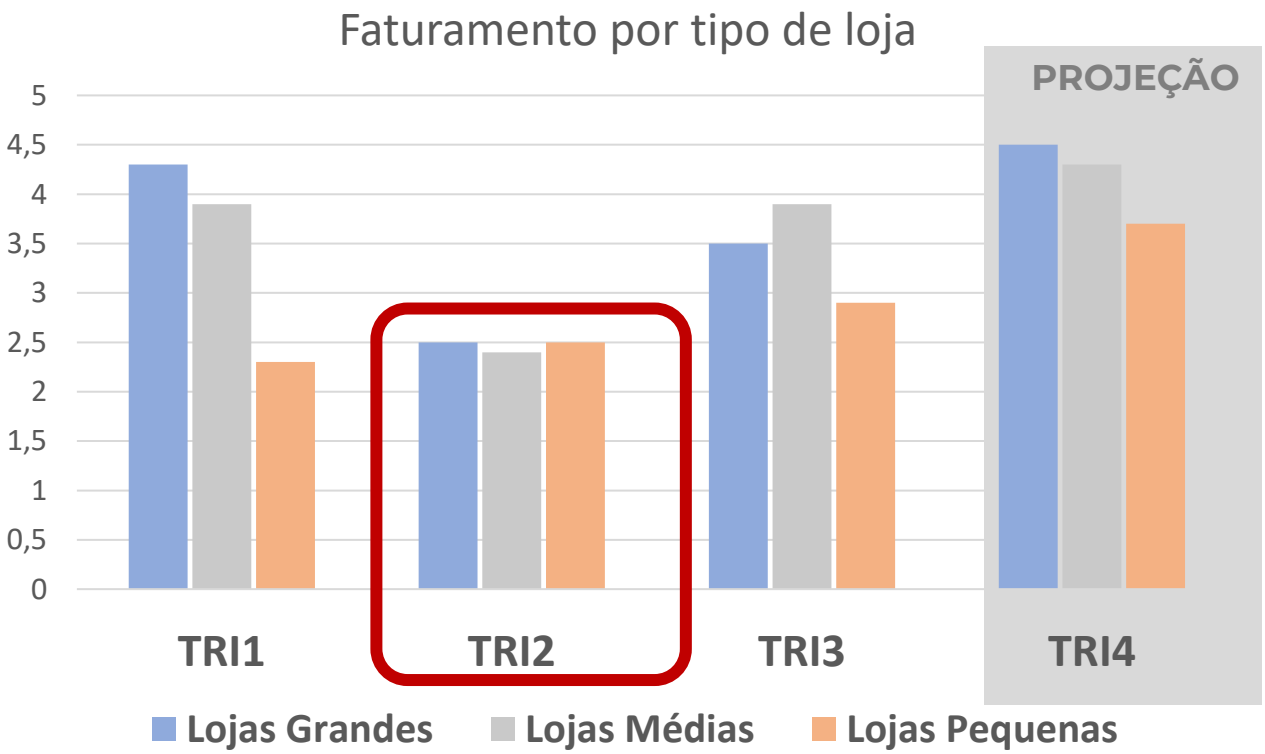
Módulo 8 – Melhorando o seu visual (Acercamento, Fechamento, Continuidade e Conexão)

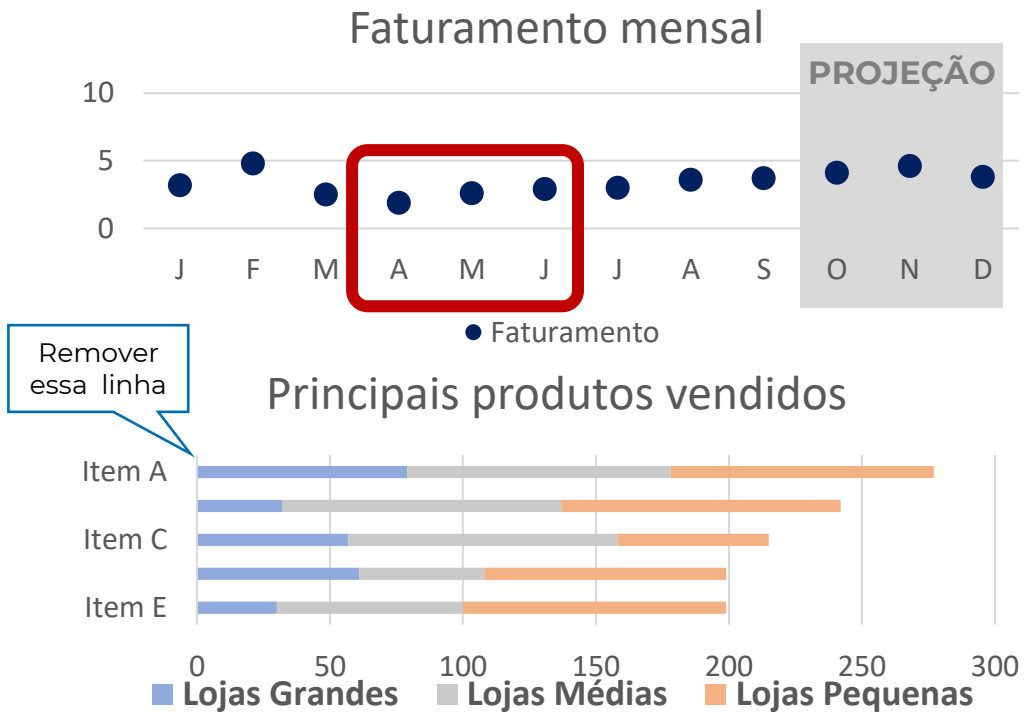
Pensando em como utilizar isso em nossas apresentações, vamos ver o exemplo abaixo:



Módulo 8 – Melhorando o seu visual (Acercamento, Fechamento, Continuidade e Conexão)

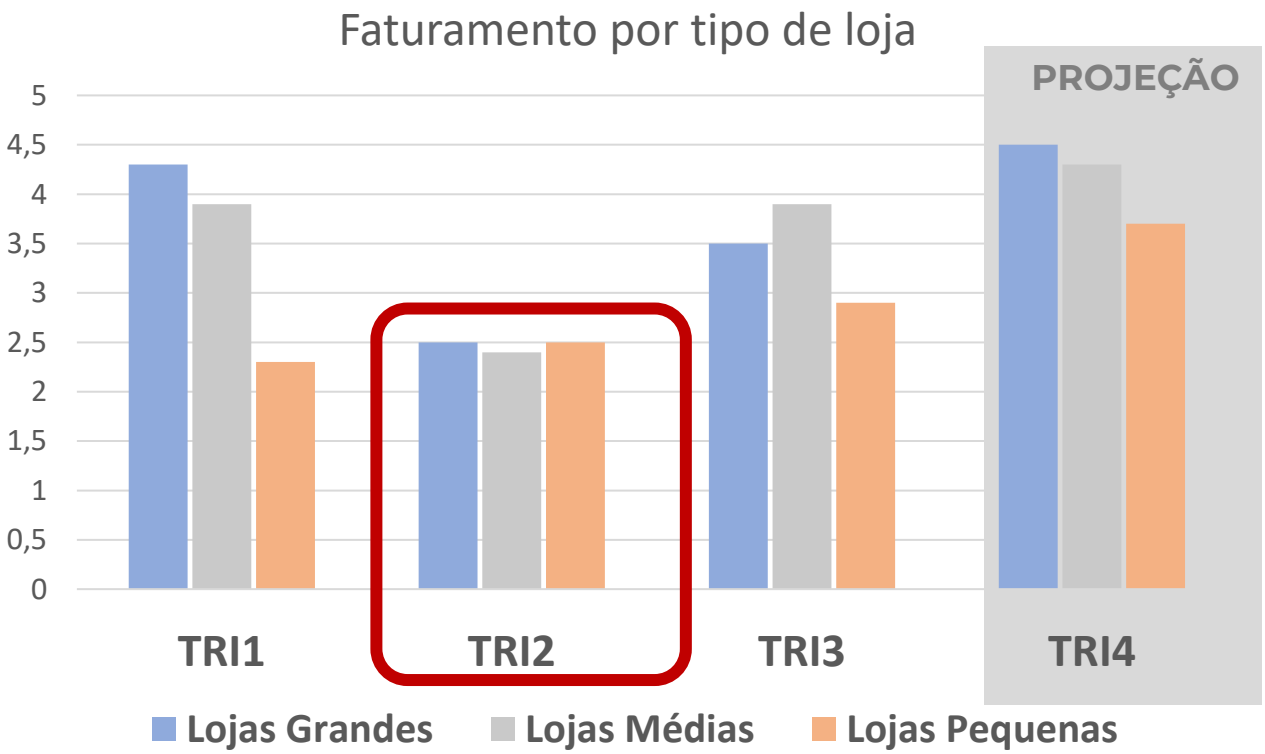
Como que eu vou utilizar isso nesse modelo que estamos criando?

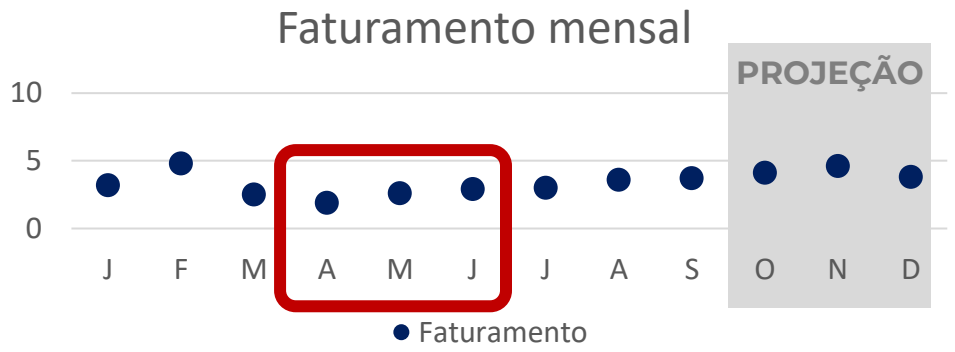


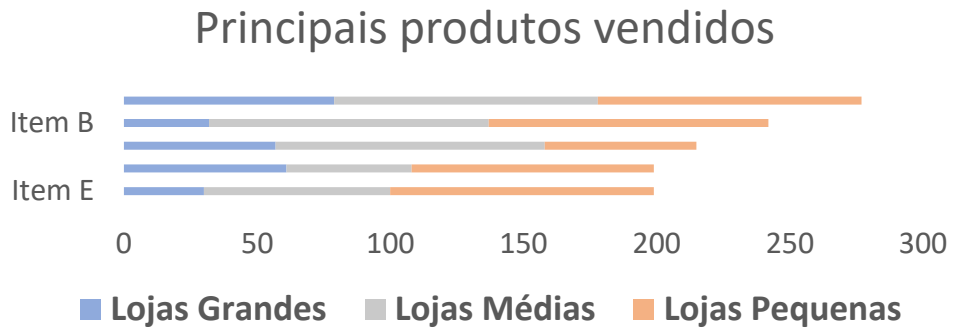


Módulo 8 – Melhorando o seu visual (Acercamento, Fechamento, Continuidade e Conexão)

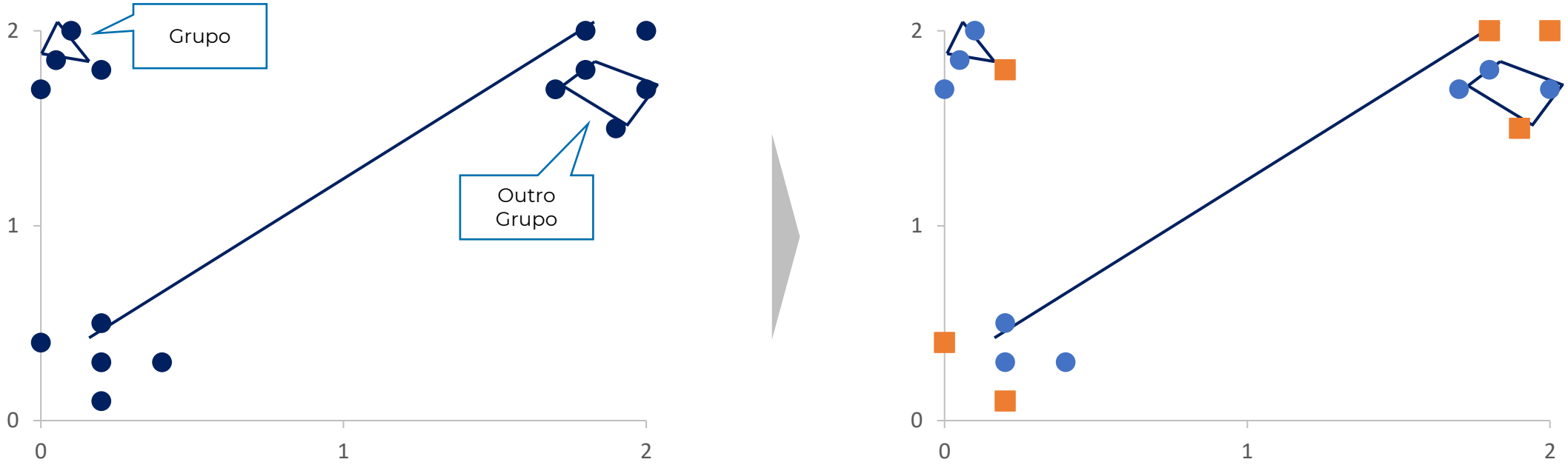
Então vamos remover a linha do eixo y e continuamos entendendo o gráfico.



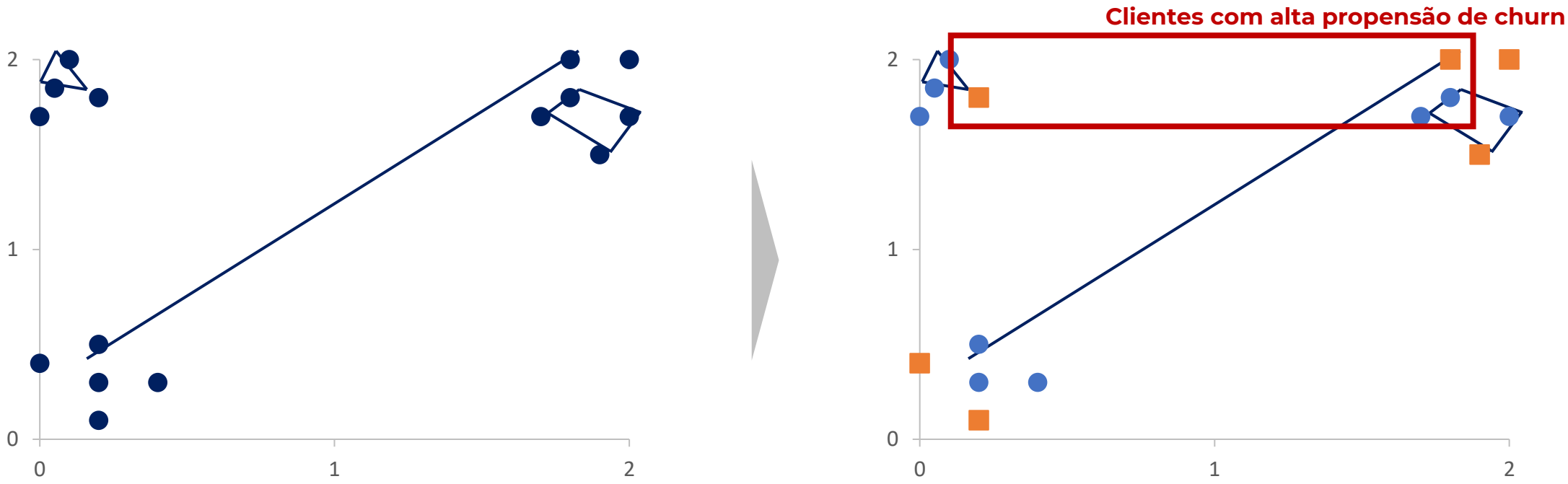




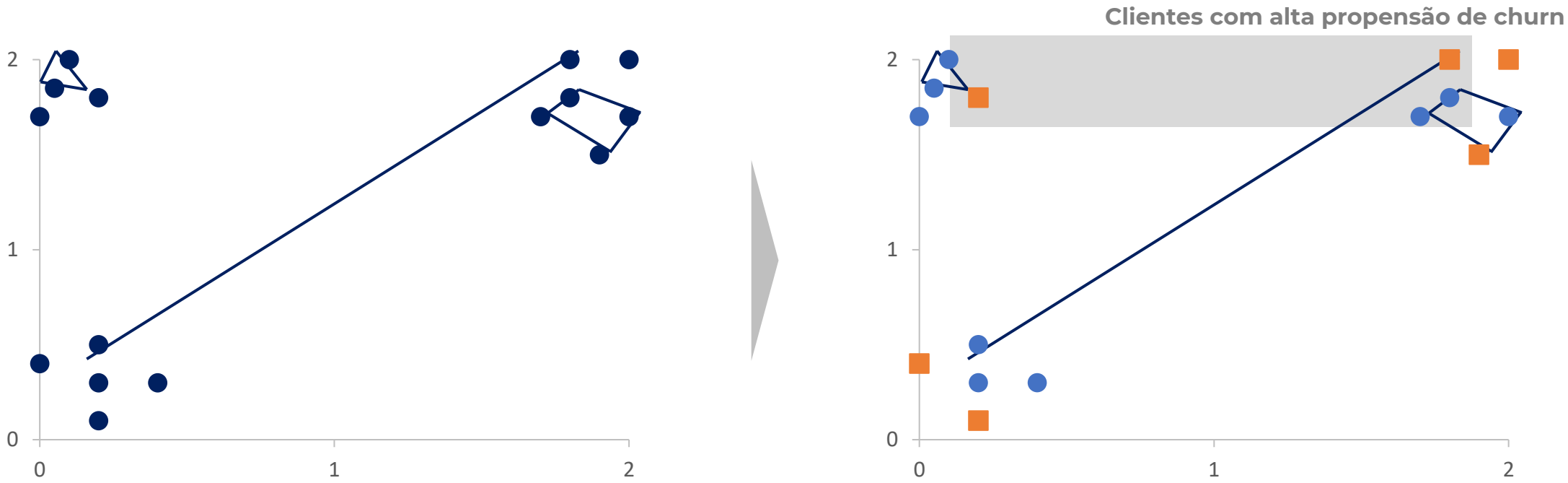
Um outro princípio que podemos considerar é a conexão: entendemos objetos fisicamente conectados como parte de um grupo.



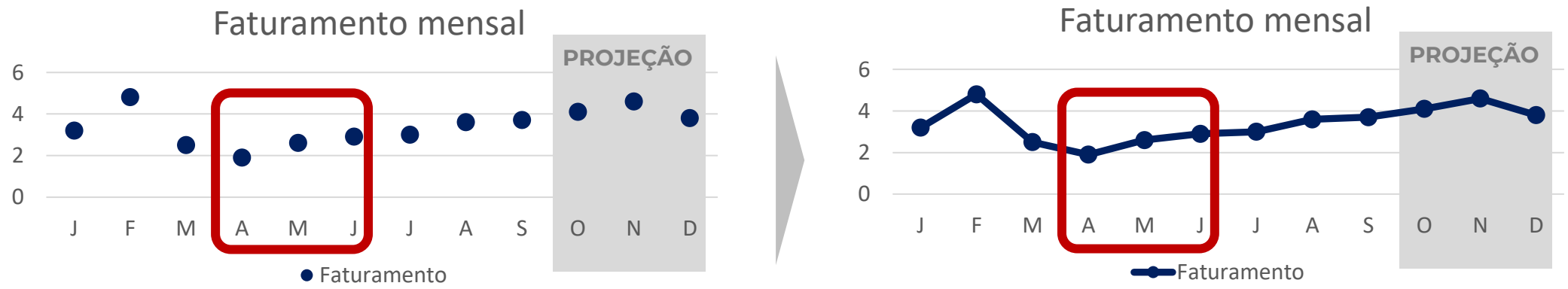
Podemos ver que os 2 clientes estão relacionados.



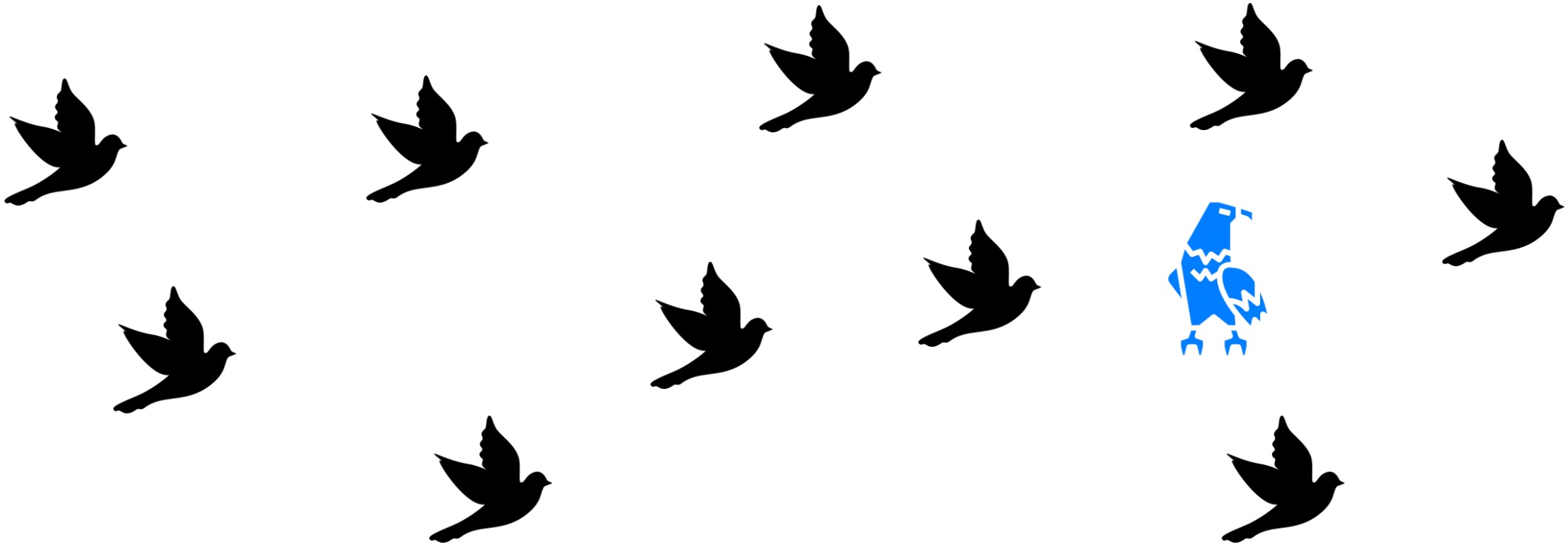
Podemos utilizar o mesmo princípio, porém, colocando uma cor diferente.



Em gráficos de pontos, quando traçamos uma reta, fica muito mais claro para interpretar os dados.

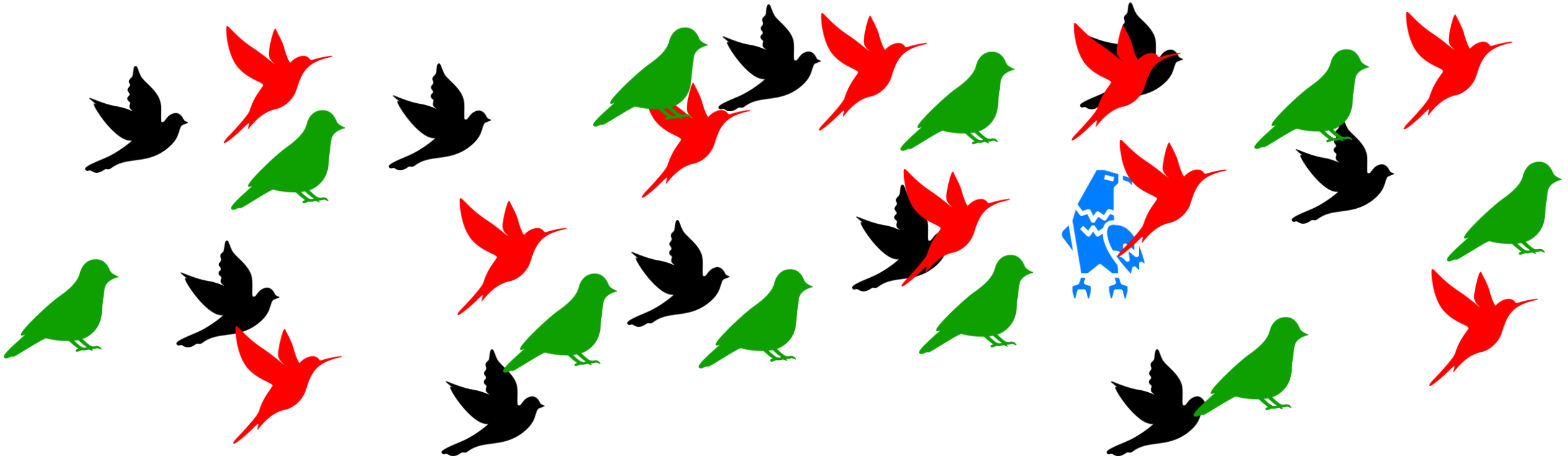


Devemos fazer o uso estratégico do **contraste** para direcionar os olhos do nosso público para o que **ele deve ver**. Repare que no momento é fácil achar o falcão.

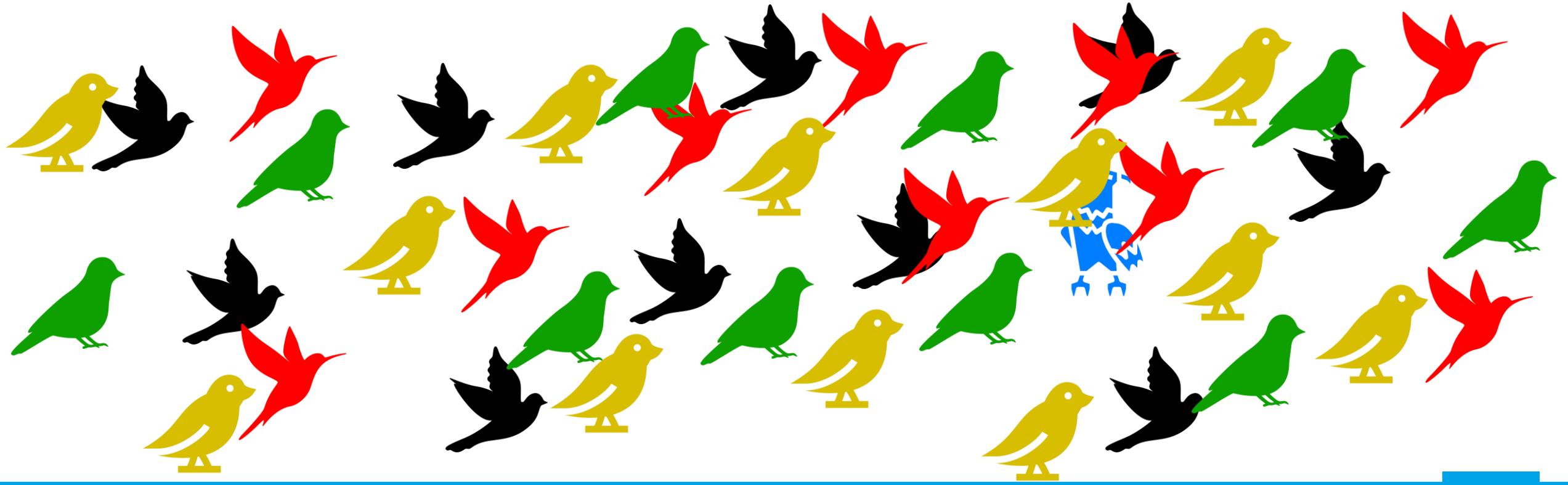


Porém quando adicionamos outro pássaro já fica mais difícil de visualizar.

Quando adicionamos mais um pássaro fica mais difícil de visualizar.



Quando adicionamos mais um pássaro fica mais difícil de visualizar, o mesmo se aplica aos seus dados.

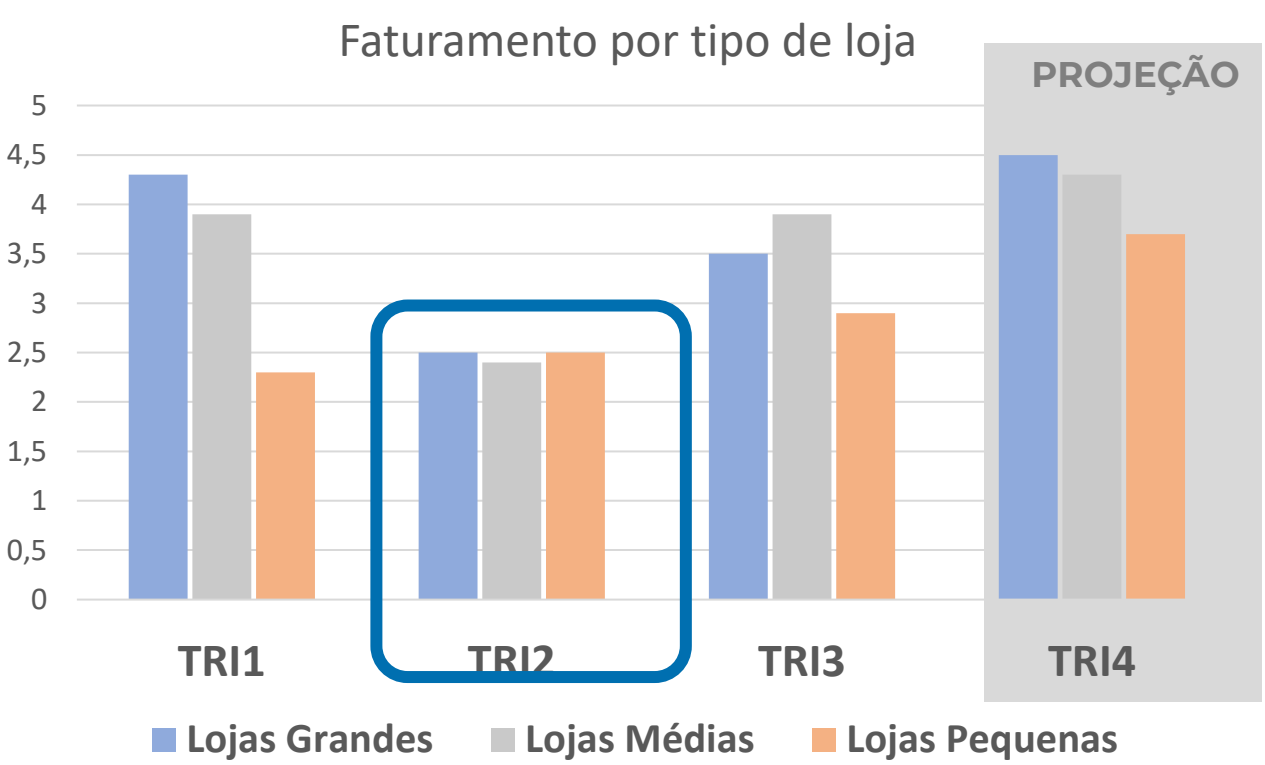


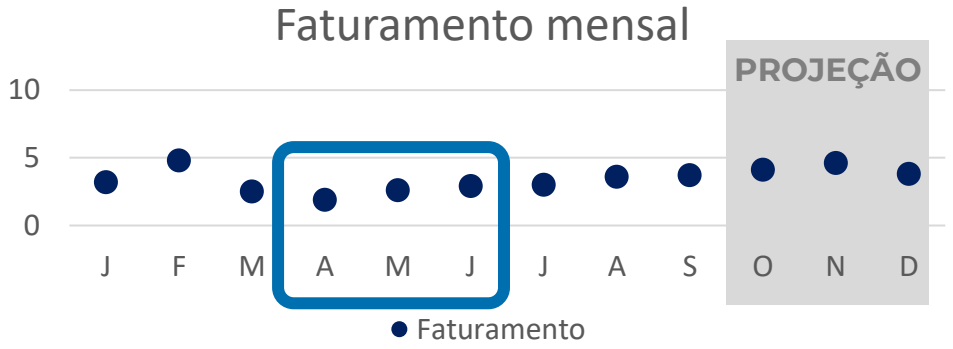
Repare que ao colocarmos o falcão em destaque é muito mais fácil de visualizá-lo.

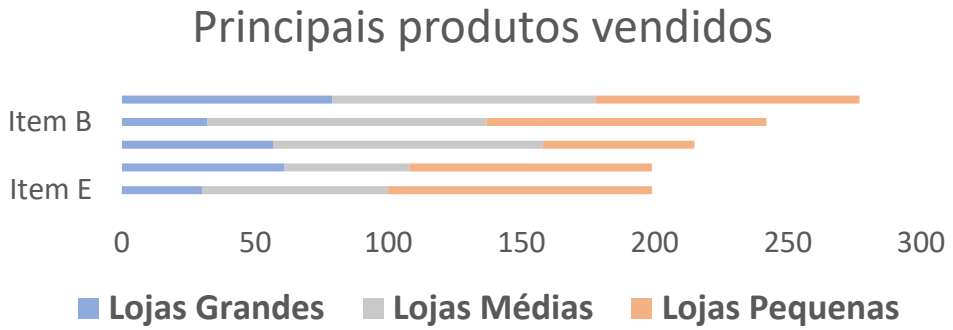
Retire o que não for necessário e dê destaque ao que você quer mostrar.

Vamos supor que isso é uma tabela e que apenas 2 dados são importantes

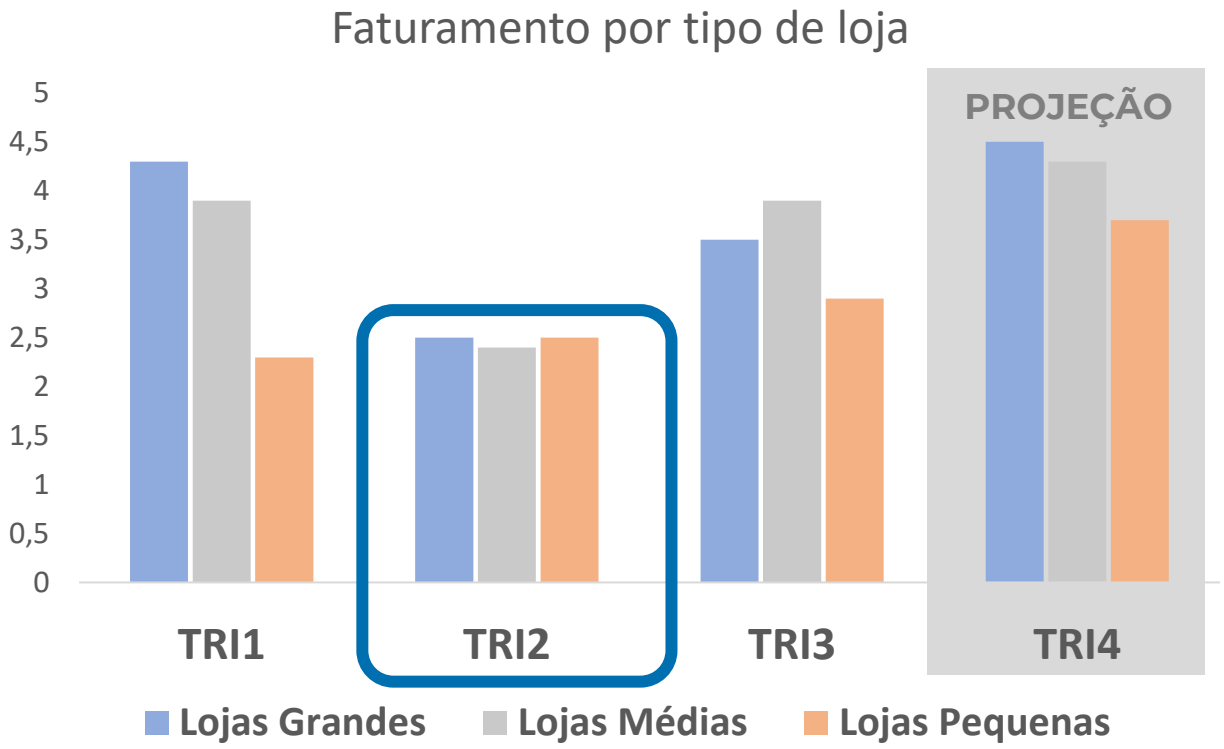
Como poderíamos melhorar o contraste nesses gráficos? Um exemplo muito prático é retirando a linha de grade.

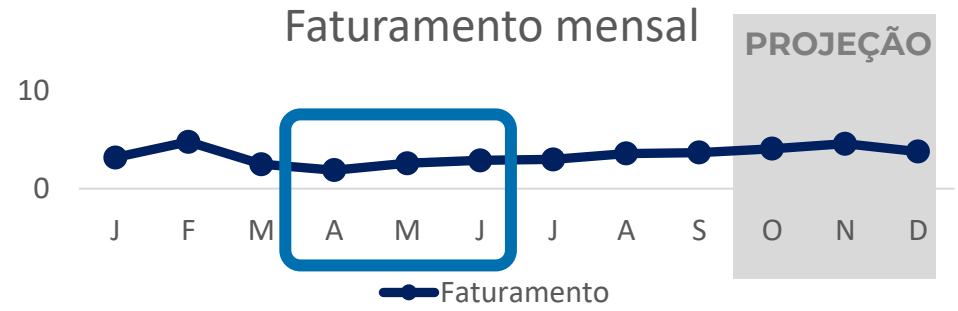


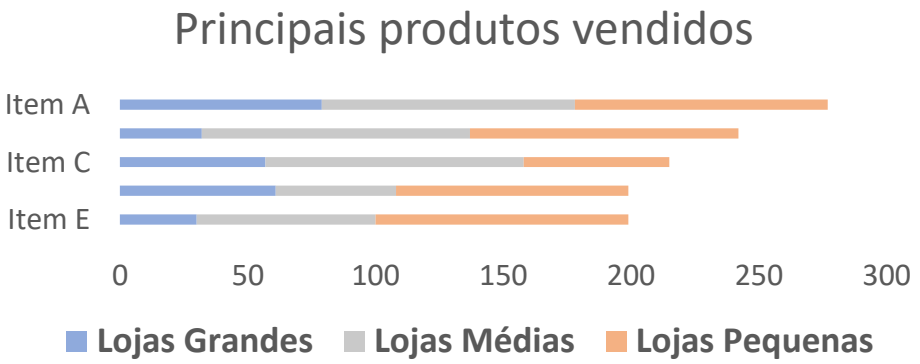




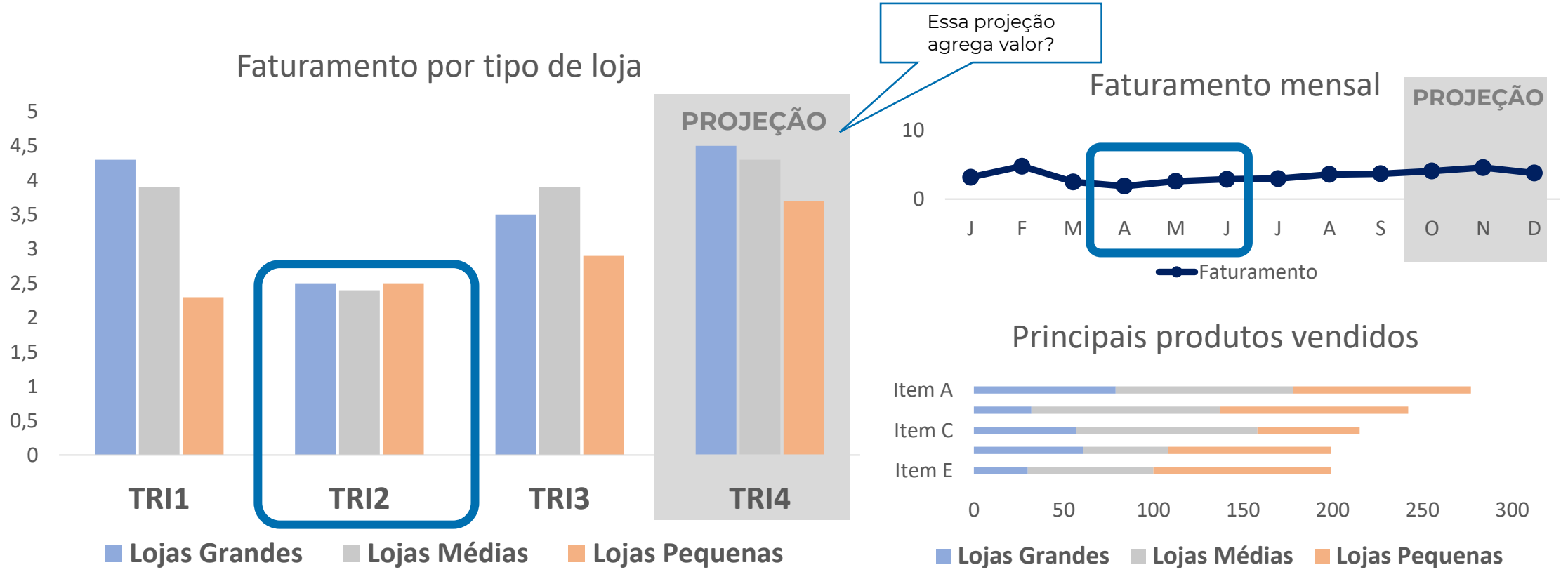
Retirando a linha de grade o nosso gráfico fica **mais limpo**.



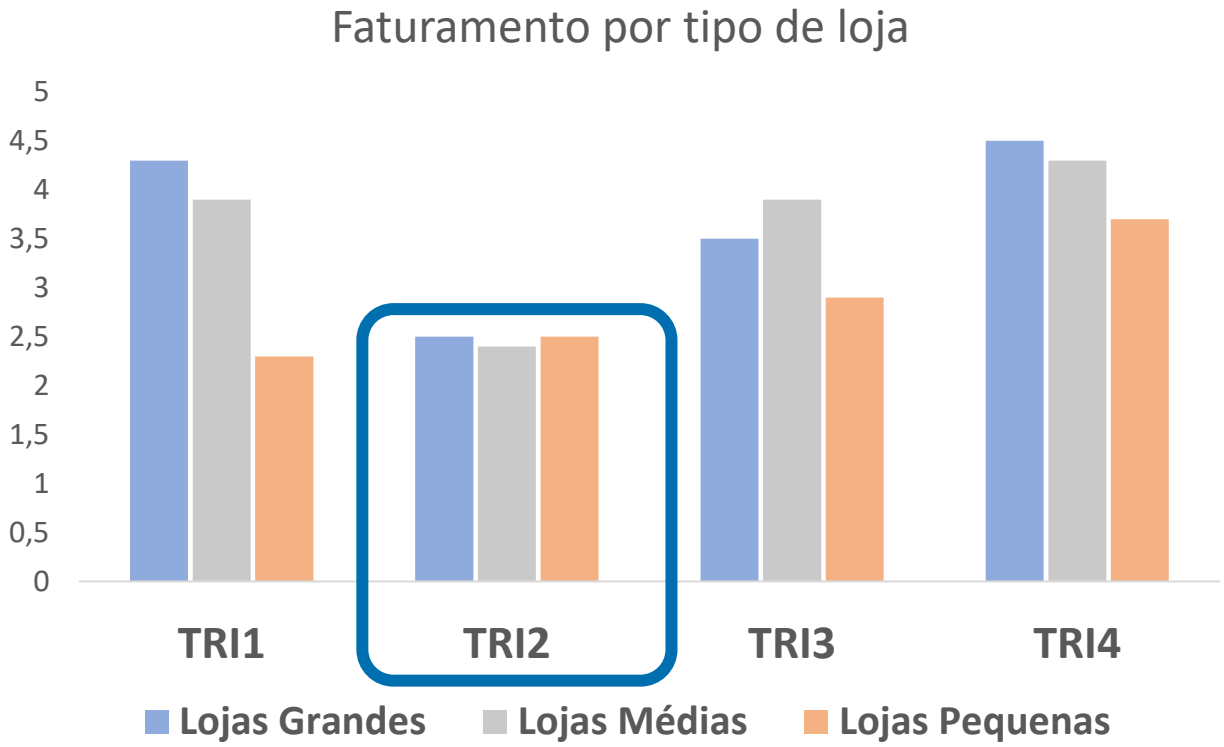


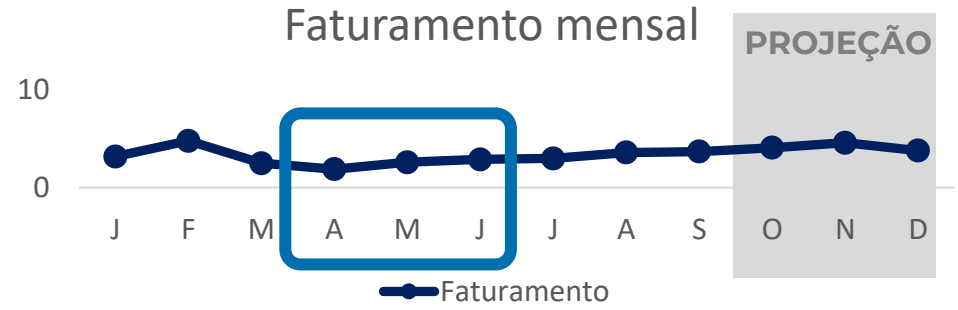


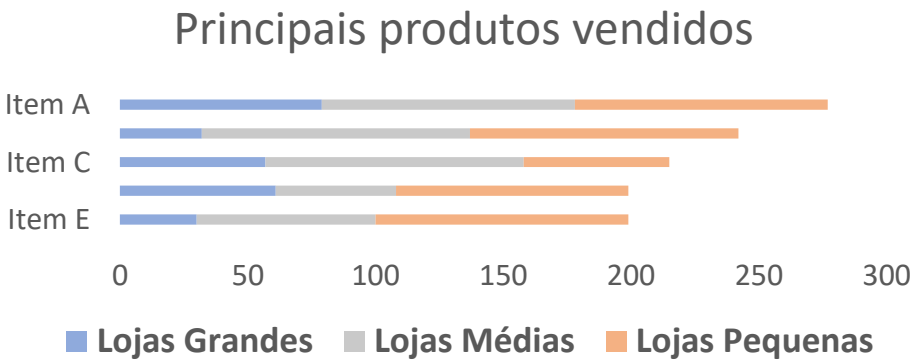
Até em elementos que criamos vamos verificar se todos agregam valor.



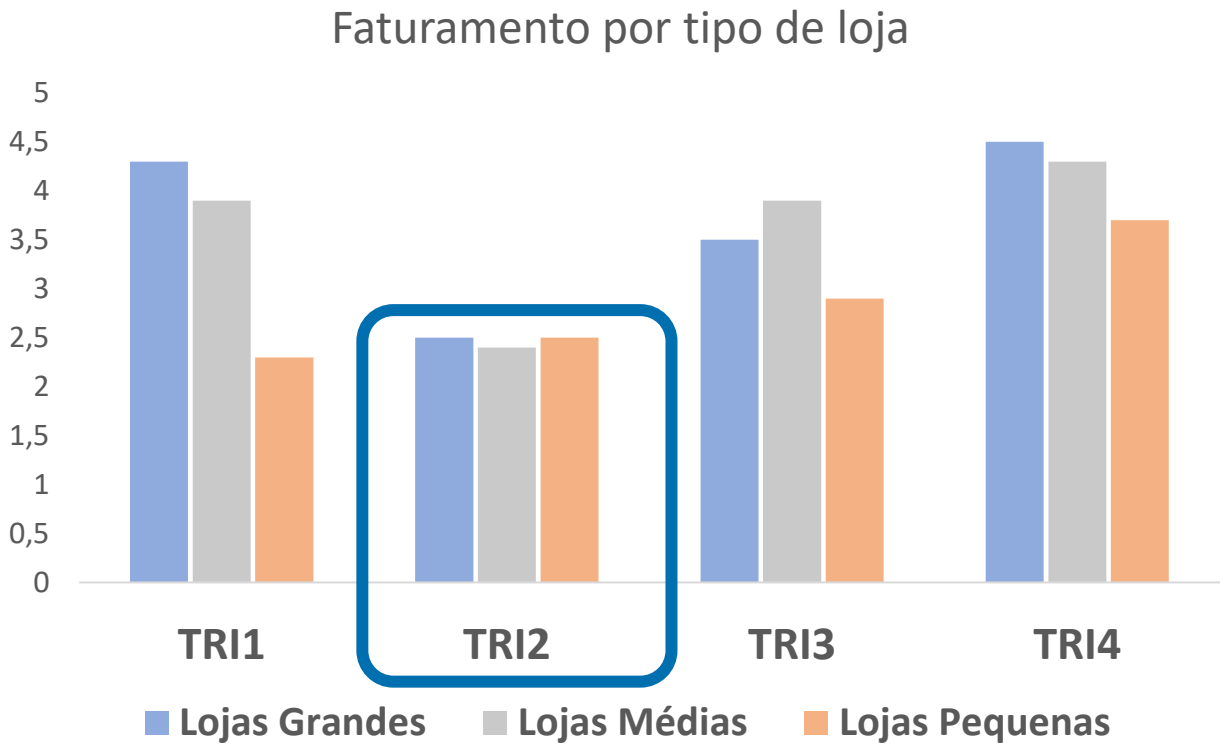
Aquela projeção não agregava valor, pois em outro gráfico os dados de outubro, novembro e dezembro estão indicados como projeção.

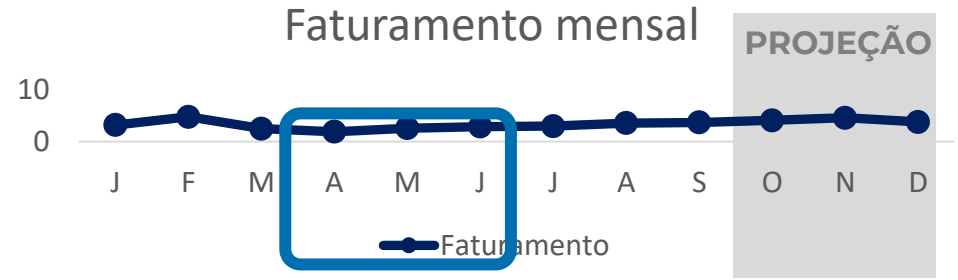


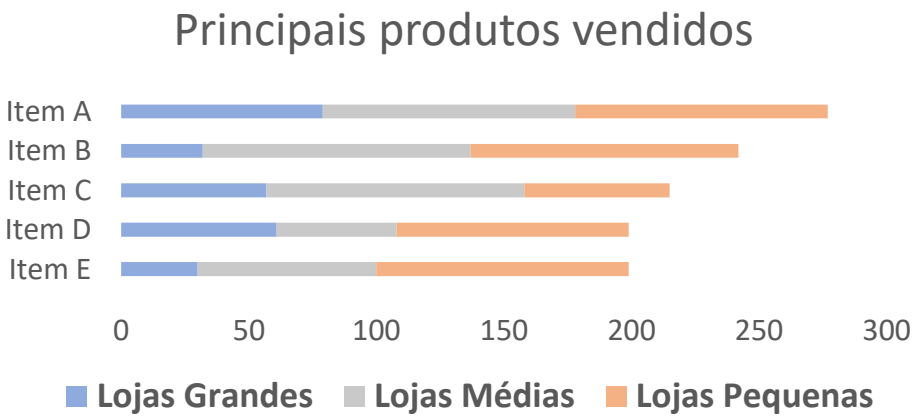




Nosso olhos tendem a ser atraídos pelo que está diferente no grupo.
E se quisermos dar um destaque para o item B?

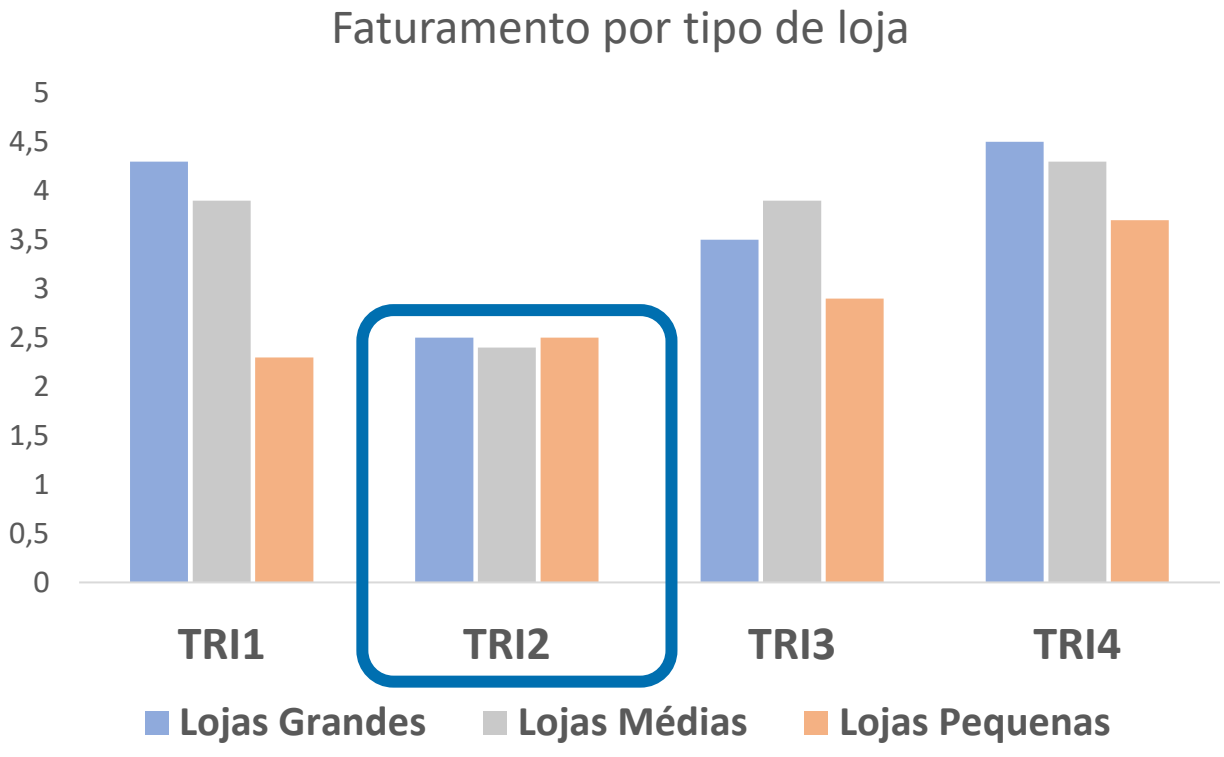


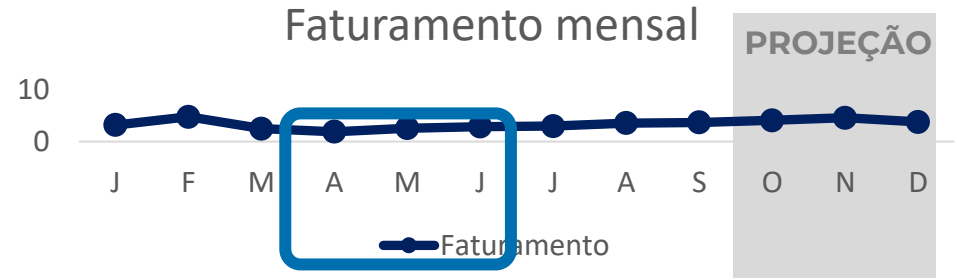


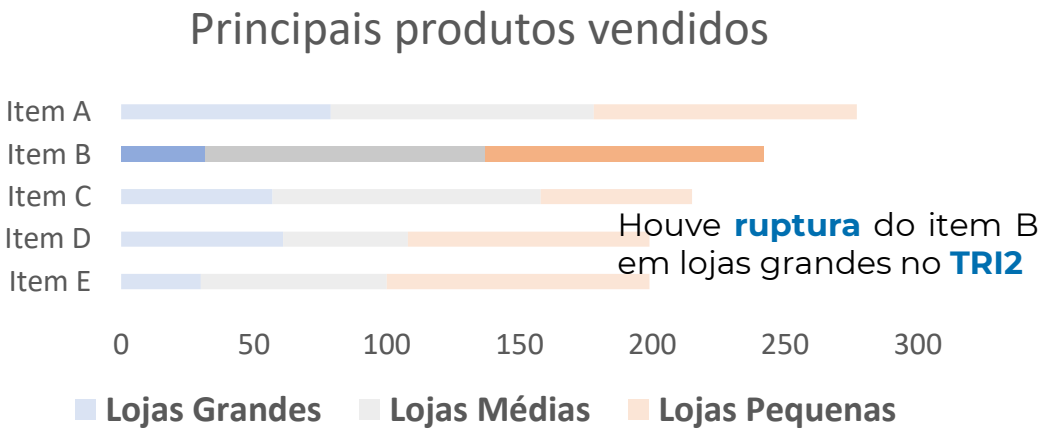


Módulo 8 – Contraste e atributos pré-atentivos

Podemos colocar a cor do item B mais escuro que as demais.

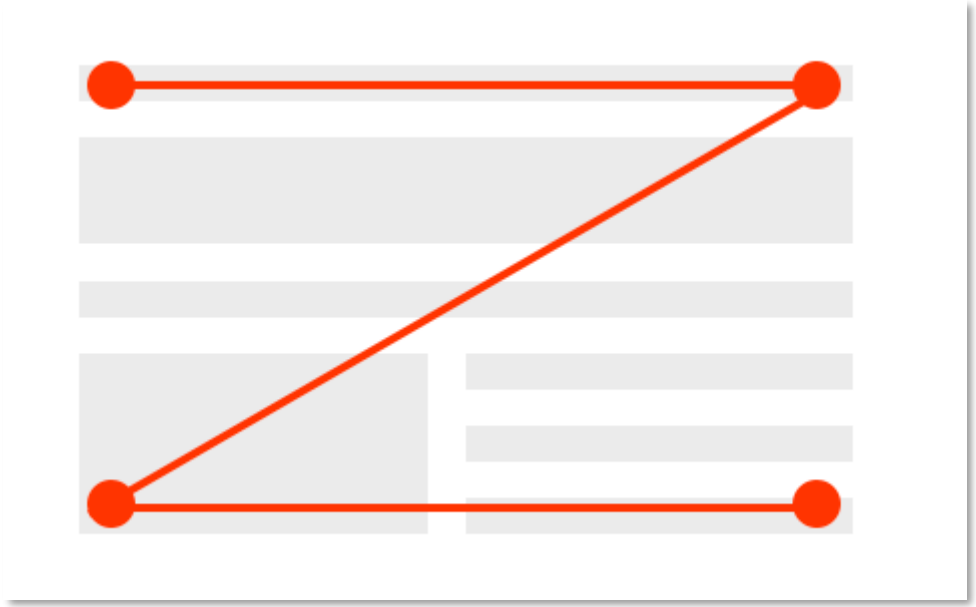






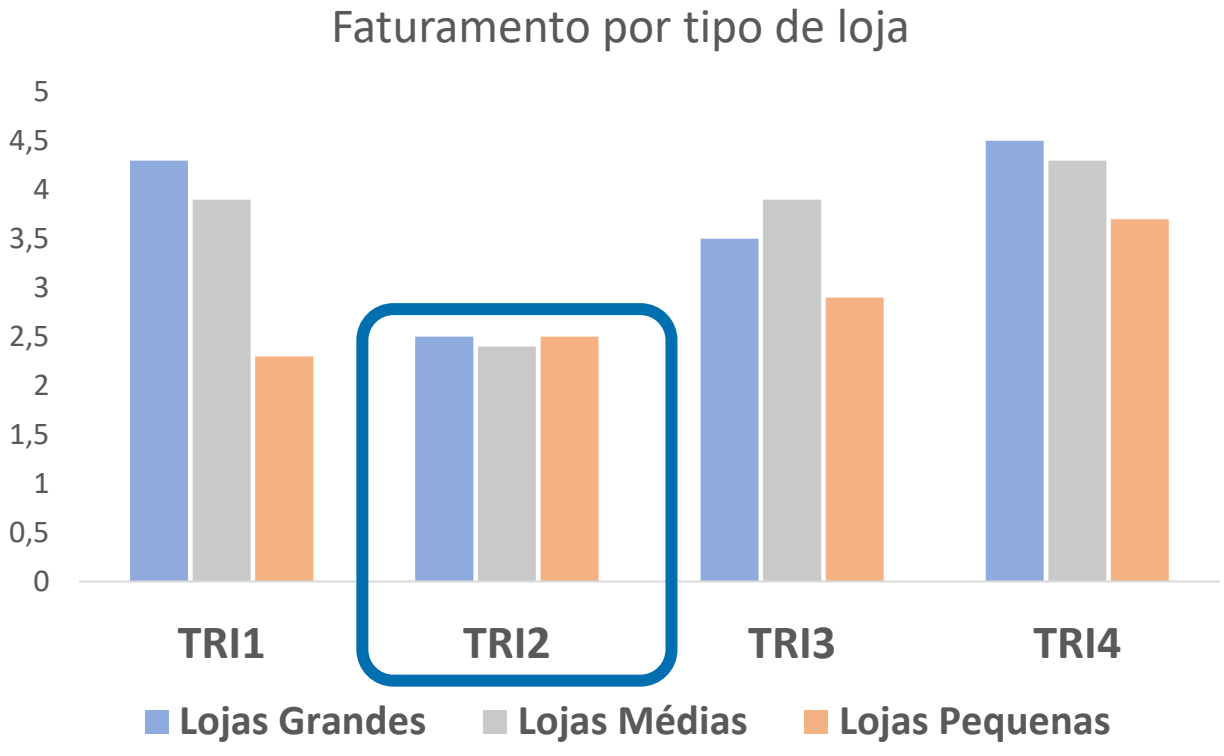
Houve **ruptura** do item B em lojas grandes no **TRI2**

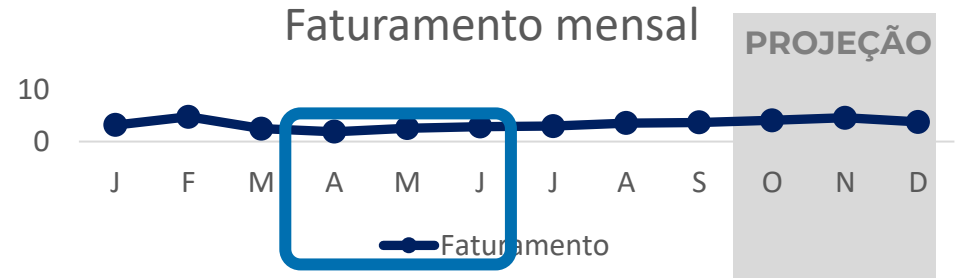
As pessoas tendem a ler a apresentação em formato Z, então podemos organizar os nossos dados dessa forma, colocando o principal na esquerda.

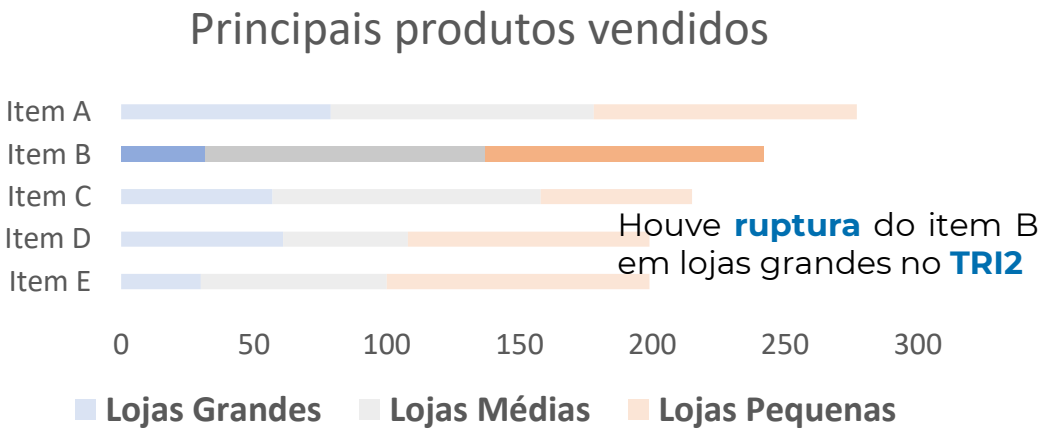


Módulo 8 – Contraste e atributos pré-atentivos

As cores podem ser usadas para destacar parte do texto / gráfico. Repare que o texto em azul se comunica com o retângulo em azul.

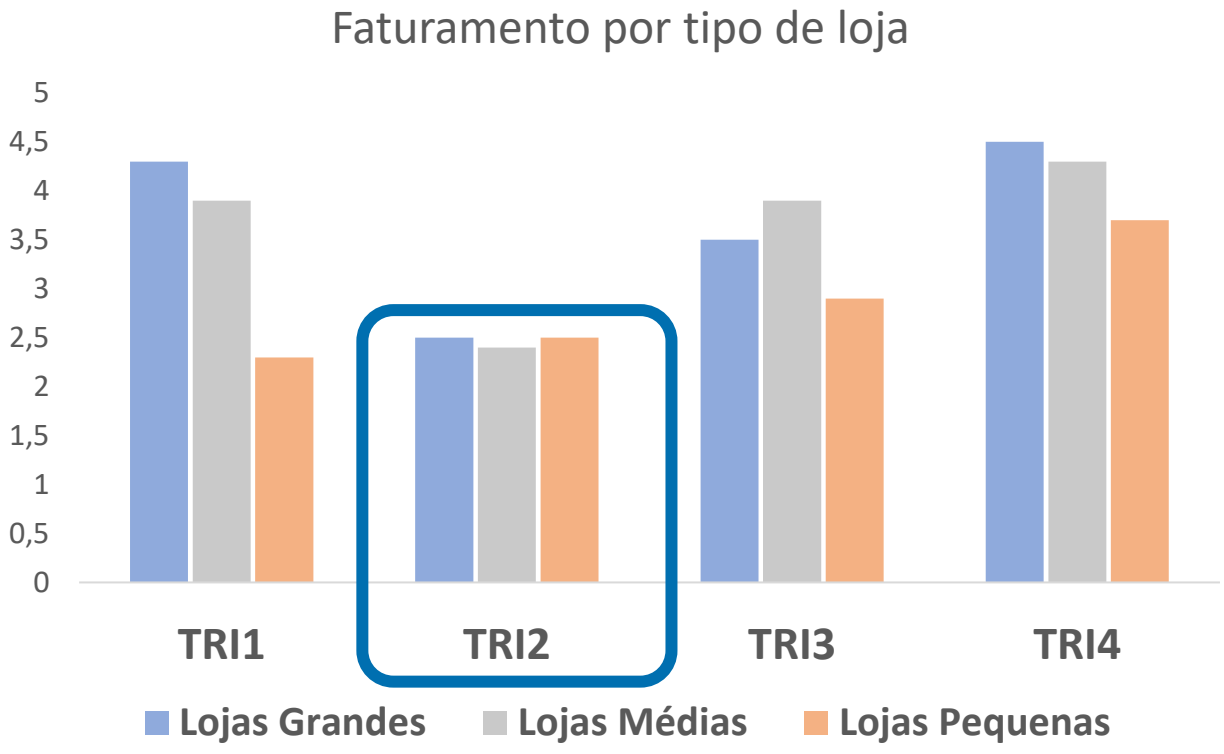


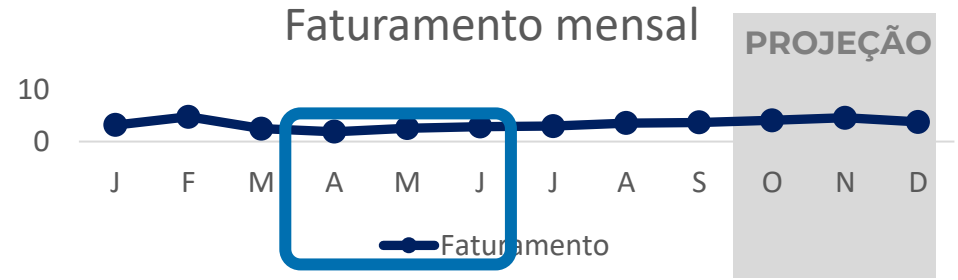


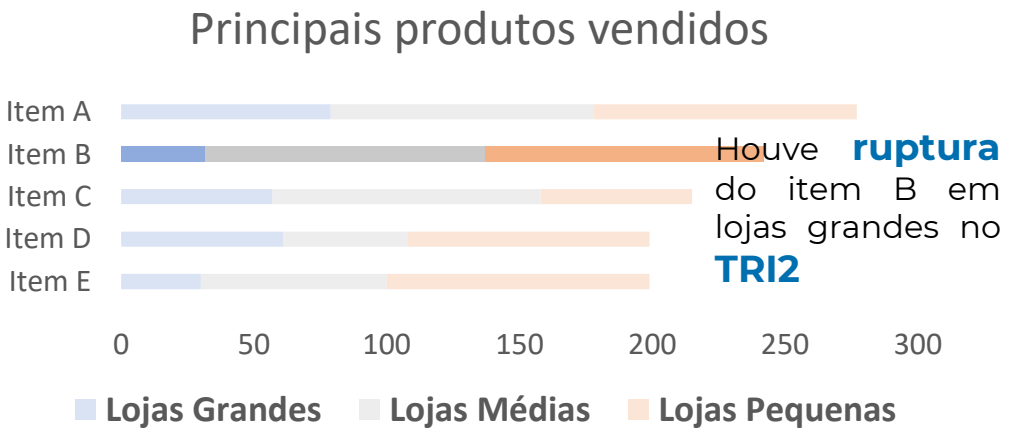


Houve **ruptura** do item B em lojas grandes no **TRI2**

Da mesma forma que as cores, os tamanhos também dão destaque

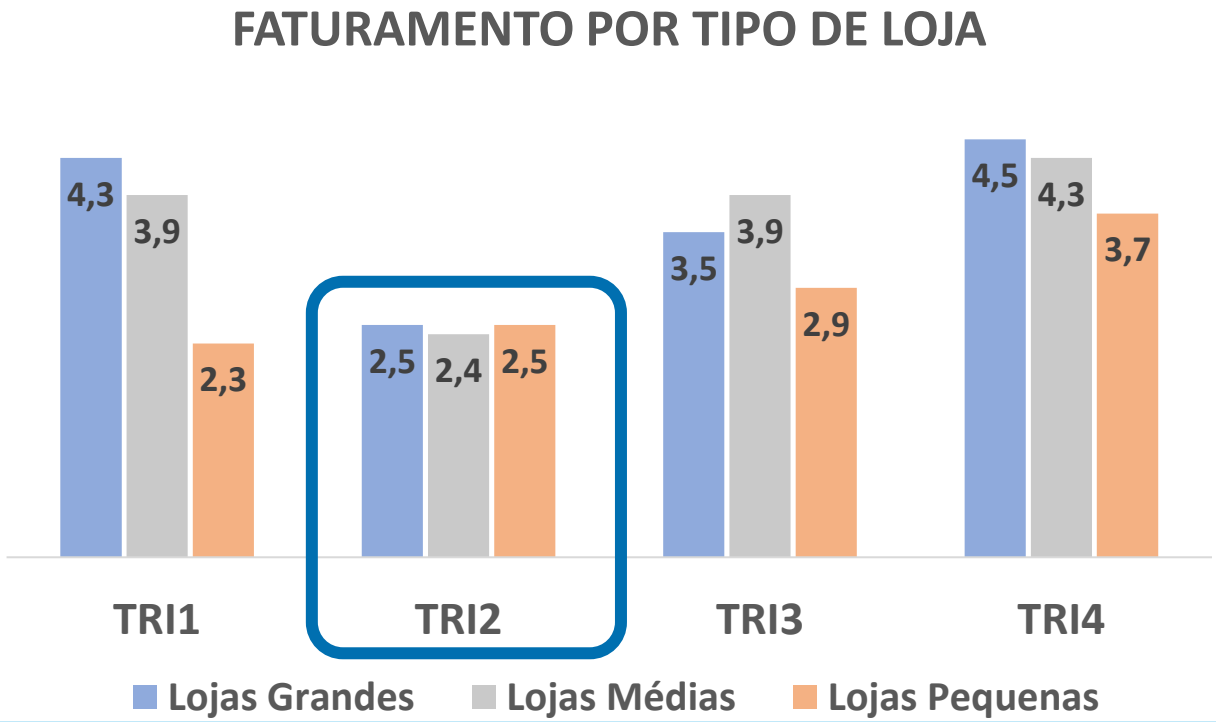


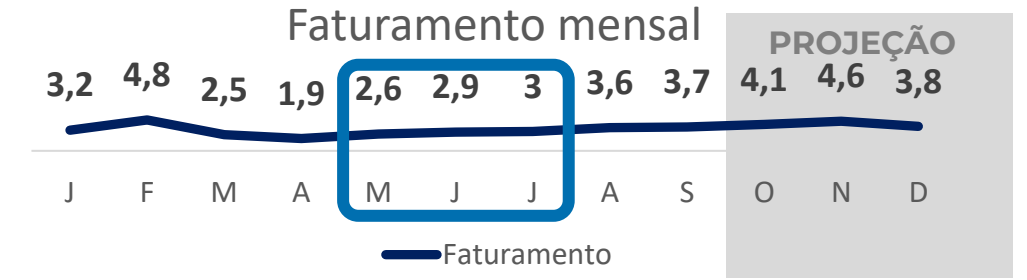


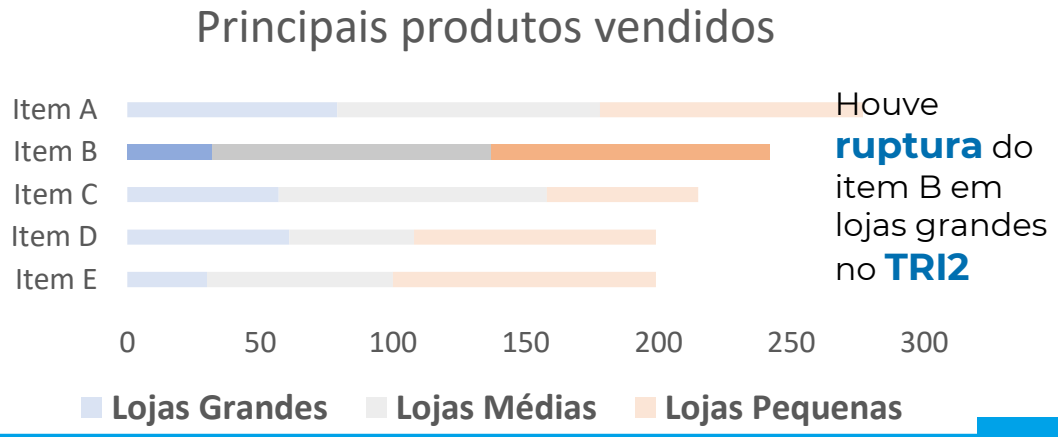


Houve **ruptura** do item B em lojas grandes no **TRI2**

Faça de tudo para tornar seu visual o mais fácil possível de compreender.

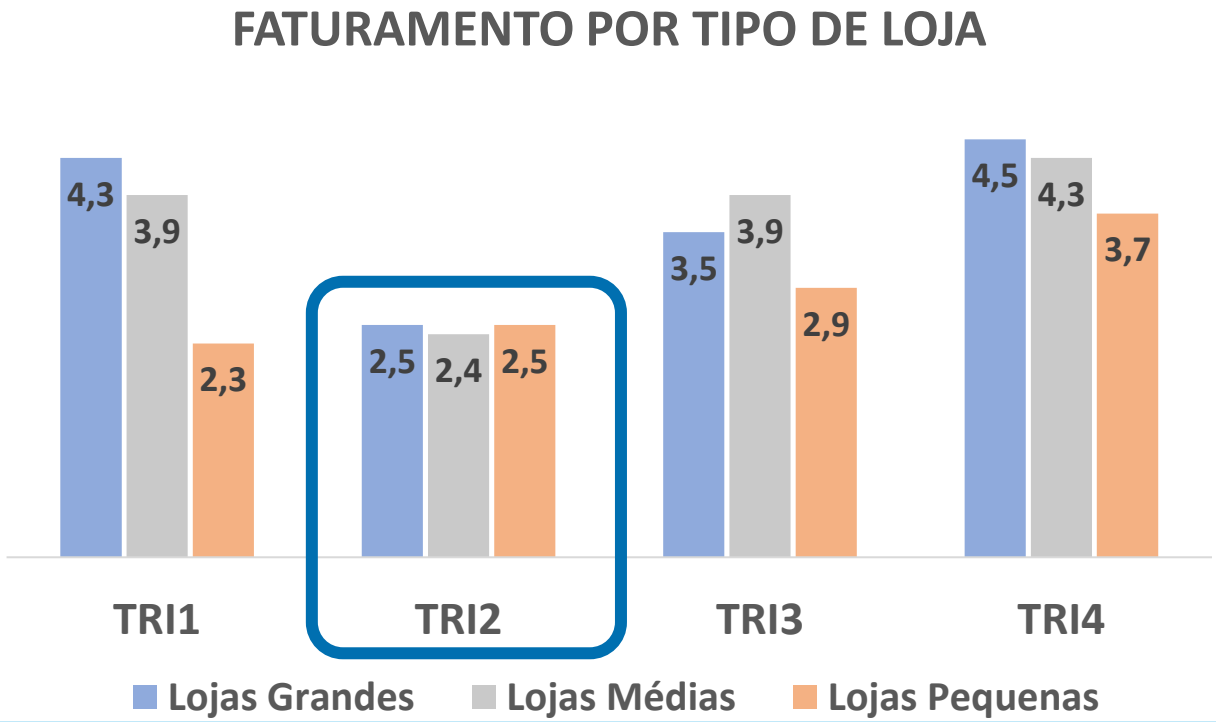


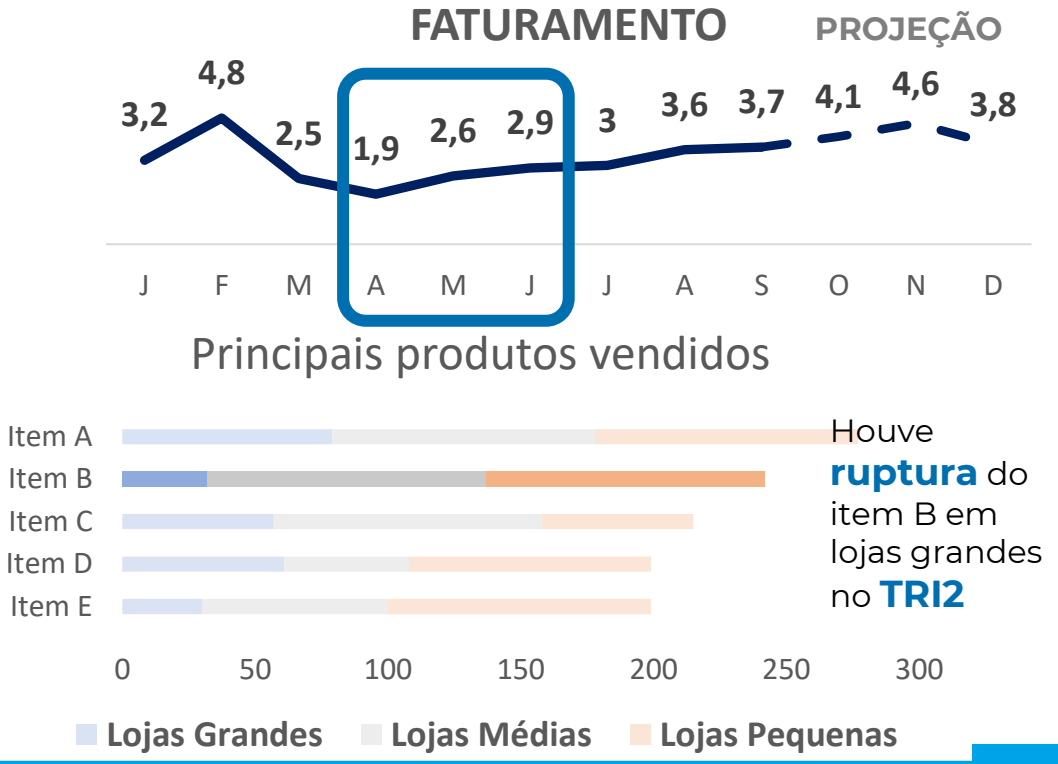




Houve **ruptura** do item B em lojas grandes no **TRI2**

Podemos também remover esse retângulo da projeção e colocar uma linha tracejada, que indica que é uma formação que ainda está sendo executada.





Agora vamos aplicar as boas práticas que falamos no PowertPoint e aplicar no matplotlib. Para começar vamos utilizar uma base de dados de x, y e z.

```
import pandas as pd
import matplotlib.pyplot as plt

dados = {
    'X': [1,2,3,4,5,6],
    'Y': [120,110,130,145,118,125],
    'Z': [95,54,86,77,90,81]
}

base = pd.DataFrame(dados)
base.head()
```

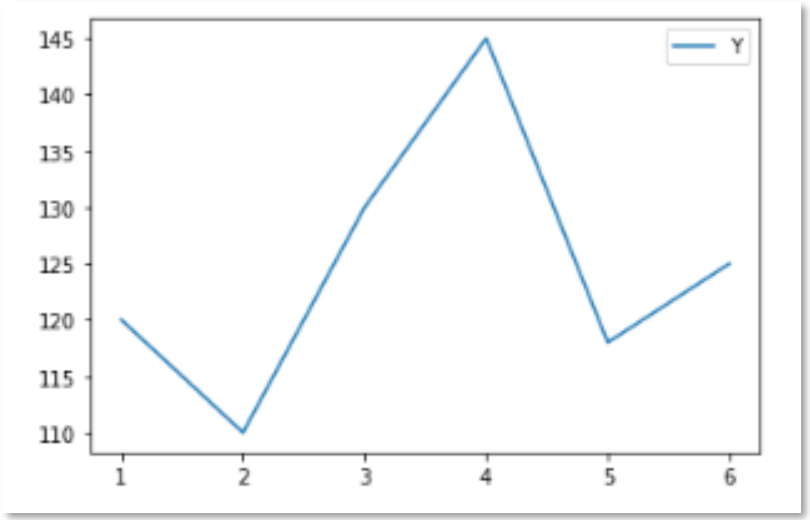
```
base.head()
```

	X	Y	Z
0	1	120	95
1	2	110	54
2	3	130	86
3	4	145	77
4	5	118	90

E aí como que criamos um plot?

Podemos consultar na documentação e copiar o código.

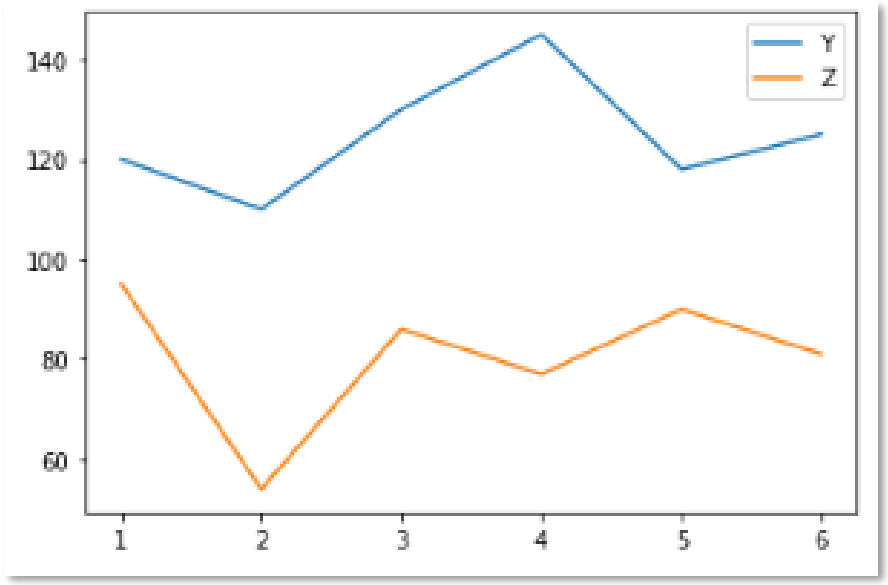
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

ax.plot(base.X,base.Y, label="Y")

plt.legend
plt.show()
```



Como podemos plotar 2 linhas nesse mesmo gráfico?

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

ax.plot(base.X,base.Y, label="Y")
ax.plot(base.X,base.Z, label="Z")

plt.legend
plt.show()
```

Adicionamos os valores de Z



E se quiséssemos separar em 2 gráficos?

Como argumento do **plt.subplots** podemos passar o número de linhas e colunas que queremos no nosso plot.

- plt.subplots(nrows= ,ncols=)

Nesse caso vamos precisar passar um índice para o ax (ax[0], ax[1])

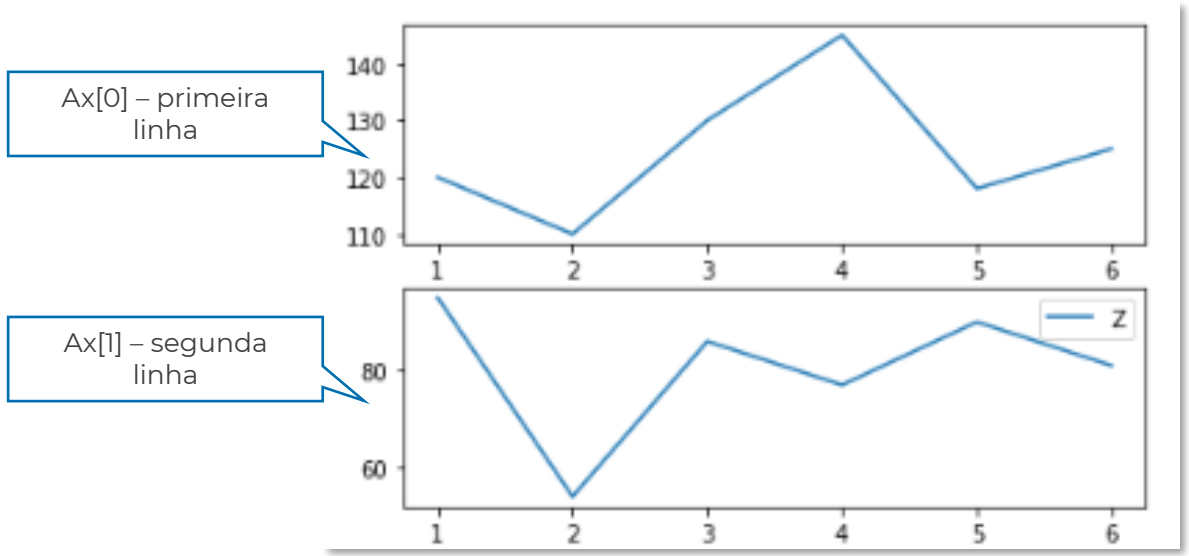
Vamos aplicar essa esses argumentos do **plt.subplots** para o nosso código:

fig, ax = **plt.subplots**(nrows=2,ncols=1)

ax[0].plot(base.X,base.Y, label="Y")
ax[1].plot(base.X,base.Z, label="Z")

plt.legend()

plt.show()



Módulo 8 – Visualização de dados no Python: Ajustando o plot e colocando barras lado a lado em um gráfico de barras

Para melhorar o visual do nosso gráfico podemos:

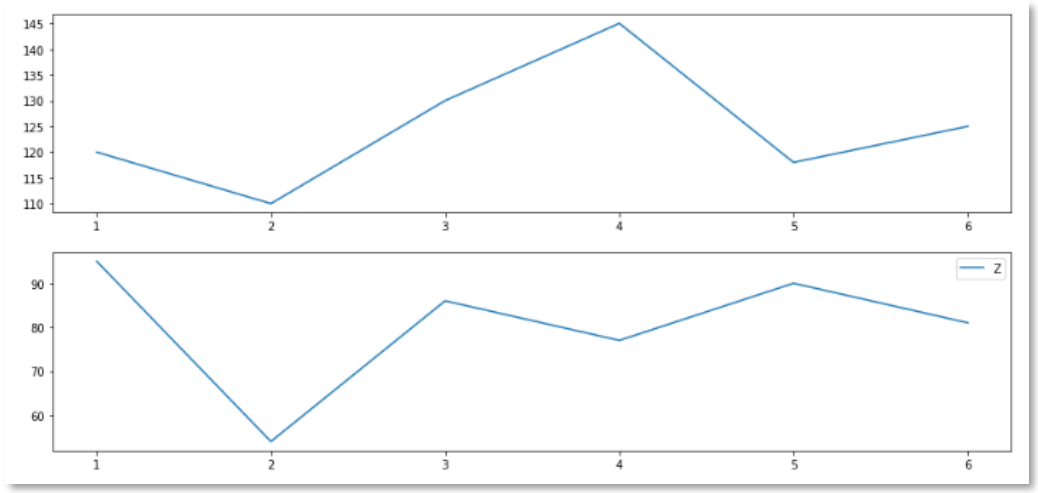
- Aumentar o tamanho do gráfico usando o figsize=(x,y)

fig, ax = **plt.subplots**(nrows=2,ncols=1,**figsize**=(15,7))

ax[0].**plot**(**base**.X, **base**.Y, label="Y")
ax[1].**plot**(**base**.X, **base**.Z, label="Z")

plt.legend()

plt.show()



Módulo 8 – Visualização de dados no Python: Ajustando o plot e colocando barras lado a lado em um gráfico de barras

Para melhorar o visual do nosso gráfico podemos:

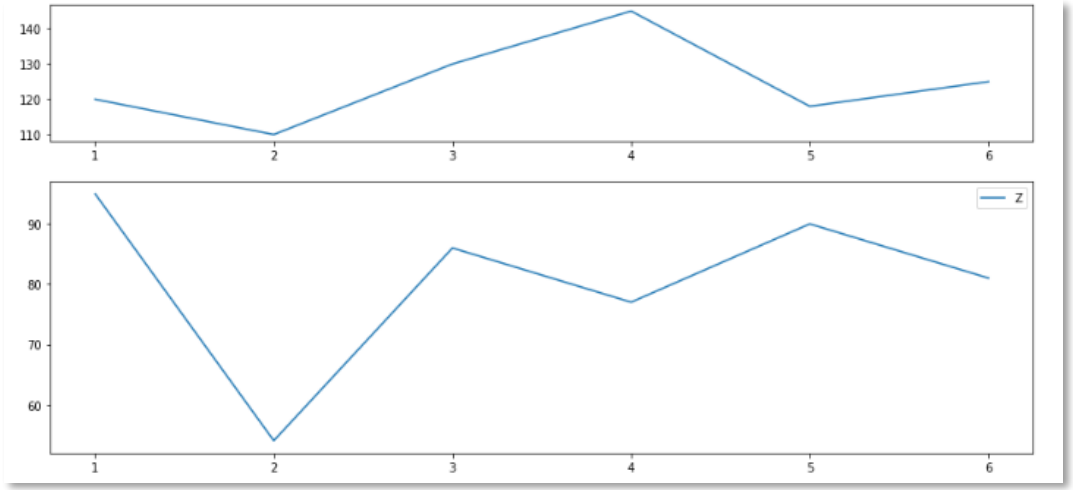
- mudar a proporção dos gráficos, podemos usar o **gridspec_kw**={'height_ratios': []}

fig, ax = **plt.subplots**(nrows=2,ncols=1,**figsize**=(15,7), **gridspec_kw**={'height_ratios': [1,2]})

ax[0].**plot**(**base**.X, **base**.Y, label="Y")
ax[1].**plot**(**base**.X, **base**.Z, label="Z")

plt.legend()

plt.show()



Módulo 8 – Visualização de dados no Python: Ajustando o plot e colocando barras lado a lado em um gráfico de barras

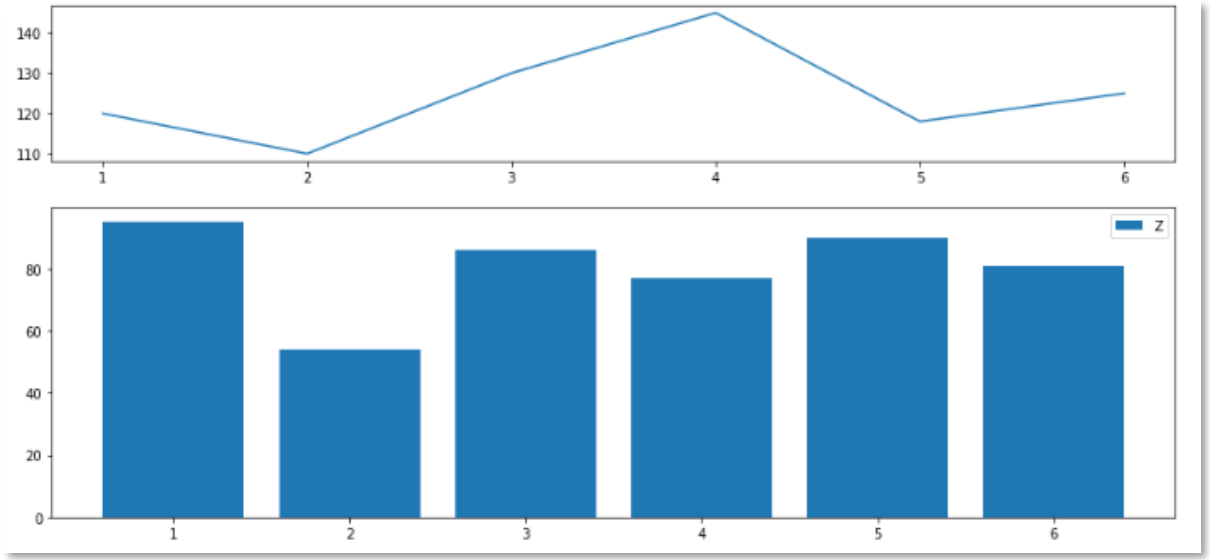
Para melhorar o visual do nosso gráfico podemos:

- Alterar o tipo de gráfico

ax[0].**plot**(**base**.X, **base**.Y, label="Y")
ax[1].**bar**(**base**.X, **base**.Z, label="Z")

plt.legend()

plt.show()

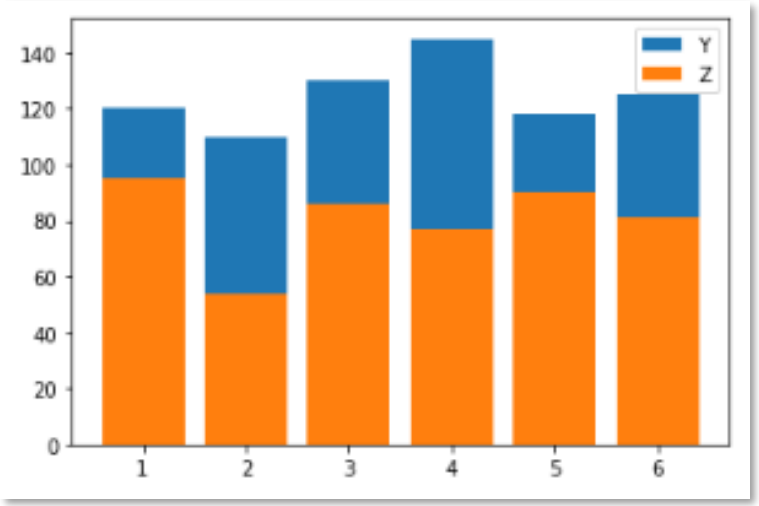


Atenção: ao trocar um tipo de gráfico, verifique os argumentos do gráfico.
Por exemplo, gráfico de linha não possui largura.

Módulo 8 – Visualização de dados no Python: Ajustando o plot e colocando barras lado a lado em um gráfico de barras

Vamos ajustar o gráfico de barras ao lado para as colunas ficarem lado a lado.

```
fig, ax = plt.subplots()

ax.bar(base.X,base.Y, label="Y")
ax.bar(base.X,base.Z, label="Z")

plt.legend()

plt.show()
```

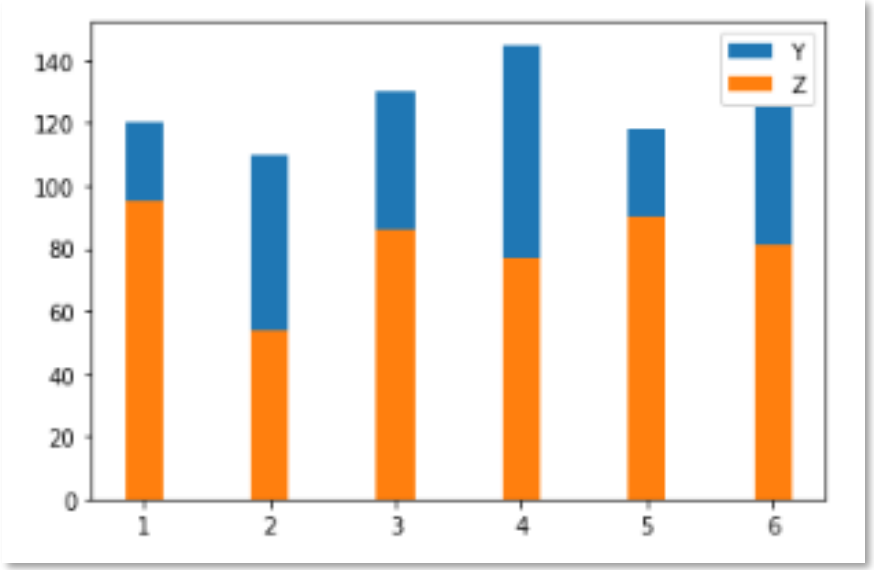


Módulo 8 – Visualização de dados no Python: Ajustando o plot e colocando barras lado a lado em um gráfico de barras

Vamos ajustar as barras para ficarem lado a lado, para isso:

- Vamos definir a largura das barras (w)

```
fig, ax = plt.subplots()

w=0.3

ax.bar(base.X,base.Y, label="Y“, width=w)
ax.bar(base.X,base.Z, label="Z“, width=w)

plt.legend()

plt.show()
```



Módulo 8 – Visualização de dados no Python: Ajustando o plot e colocando barras lado a lado em um gráfico de barras

Vamos ajustar as barras para ficarem lado a lado., para isso:

- Depois vamos deslocar o gráfico para os lados

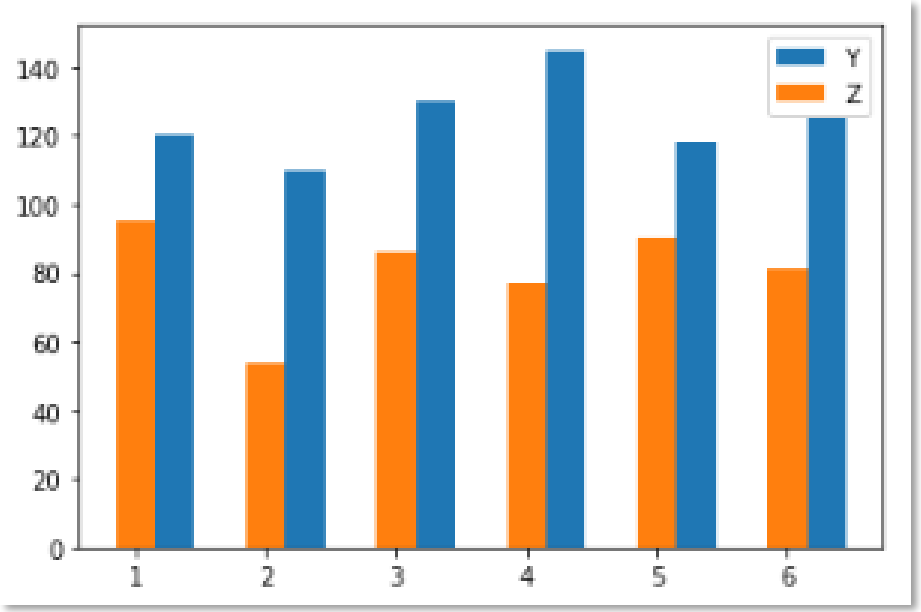
fig, ax = **plt.subplots**()

w=0.3

ax.**bar**(base.X+w,base.Y, label="Y“, width=w)
ax.**bar**(base.X,base.Z, label="Z“, width=w)

plt.legend()

plt.show()



Módulo 8 – Visualização de dados no Python: Ajustando o plot e colocando barras lado a lado em um gráfico de barras

Vamos ajustar as barras para ficarem lado a lado., para isso:

- Depois vamos deslocar o gráfico para os lados utilizando w/2

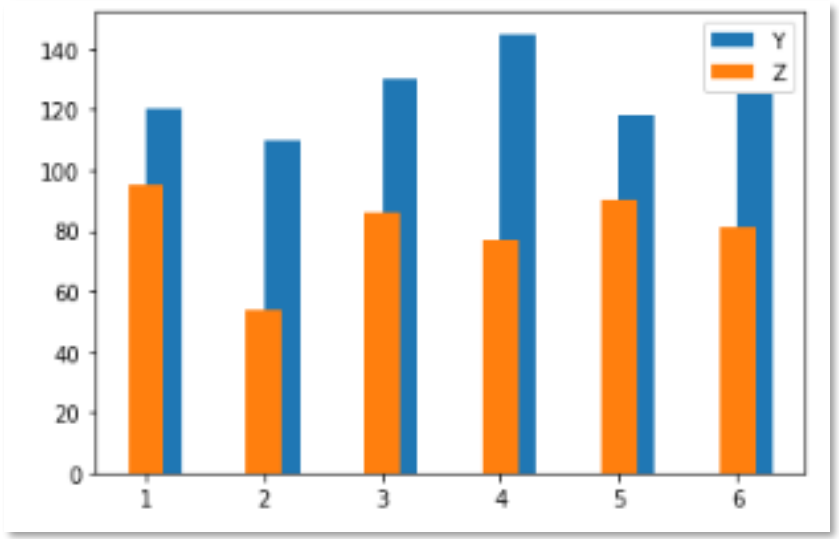
fig, ax = **plt.subplots**()

w=0.3

ax.**bar**(base.X+w/2,base.Y, label="Y“, width=w)
ax.**bar**(base.X,base.Z, label="Z“, width=w)

plt.legend()

plt.show()



Módulo 8 – Visualização de dados no Python: Ajustando o plot e colocando barras lado a lado em um gráfico de barras

Vamos ajustar as barras para ficarem lado a lado., para isso:

- Depois vamos deslocar o gráfico para os lados utilizando w/2

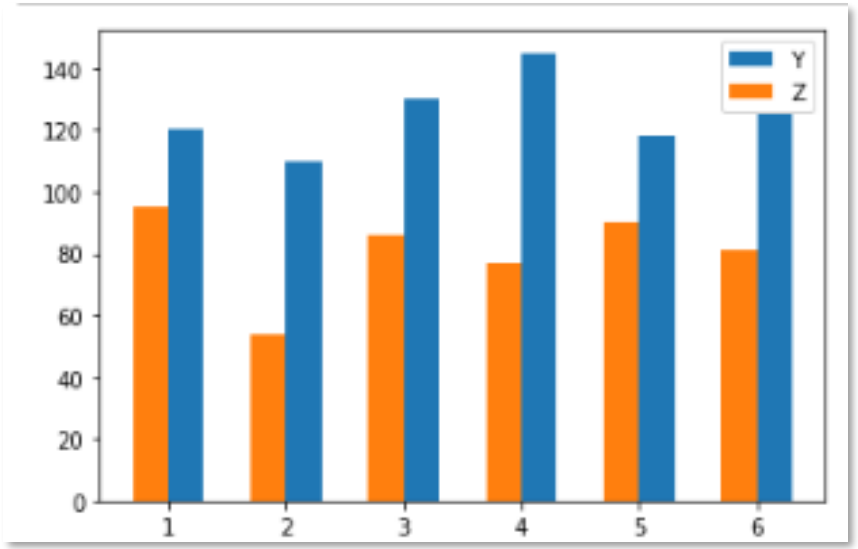
fig, ax = **plt.subplots**()

w=0.3

ax.**bar**(base.X+w/2,base.Y, label="Y“, width=w)
ax.**bar**(base.X-w/2,base.Z, label="Z“, width=w)

plt.legend()

plt.show()



Módulo 8 – Visualização de dados no Python: Adicionando rótulo nos dados (annotate)

Agora que já vimos como **separar as barras** e **organizar melhor o nosso visual**, vamos aprender a colocar o rótulo de dados utilizando a função **annotate**.

O annotate é utilizado para colocar qualquer texto.

Como funciona essa função ?

A estrutura do annotate é:

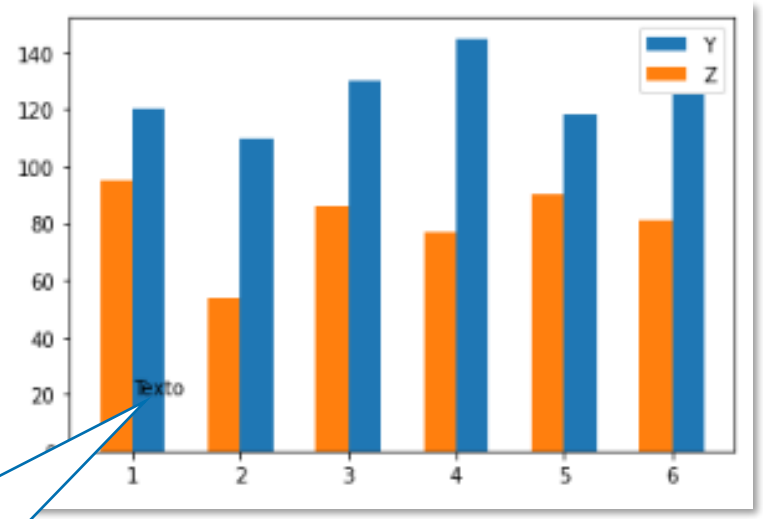
.**annotate**("<Texto>",<posicao(x,y)>)

Módulo 8 – Visualização de dados no Python: Adicionando rótulo nos dados (annotate)

Vamos fazer o rótulo de dados passo a passo no nosso código do gráfico:

```
fig, ax = plt.subplots()

w=0.3

ax.bar(base.X+w/2, base.Y, label="Y",width=w)
ax.bar(base.X-w/2, base.Z, label="Z",width=w)

ax.annotate("Texto", (1,20))

plt.legend()

plt.show()
```

Vamos substituir primeiramente por valores aleatórios



"Texto" na posição (1,20)

Módulo 8 – Visualização de dados no Python: Adicionando rótulo nos dados (annotate)

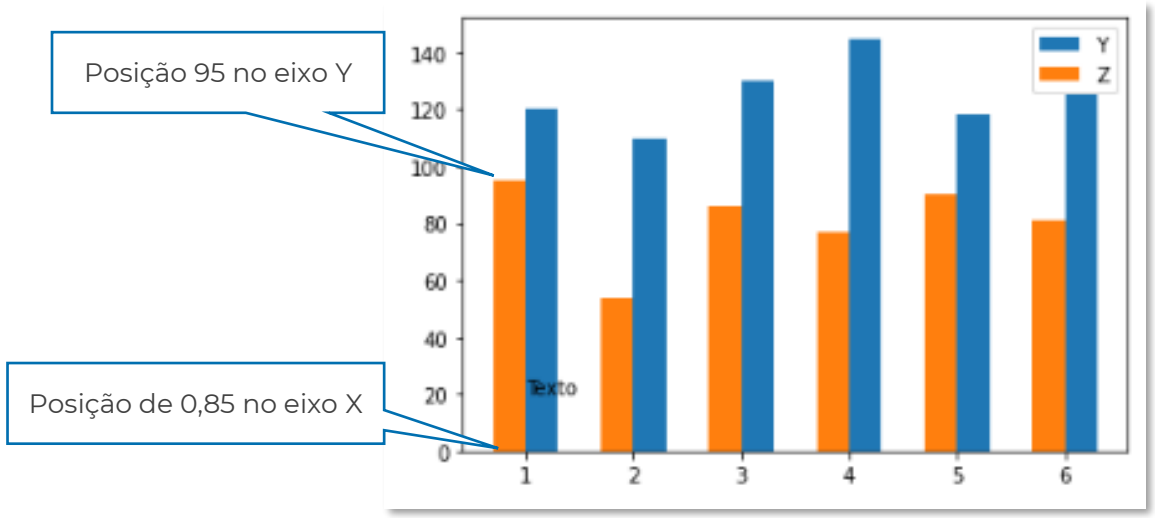
Para descobrir a posição do primeiro ponto laranja basta digitarmos o código:

base.X[0]-**w**/2

```
base.X[0]-w/2
0.85
```

base.Z[0]

```
base.Z[0]
95
```



Vamos colocar exatamente esses dois pontos como posição X e Y na função **annotate**.

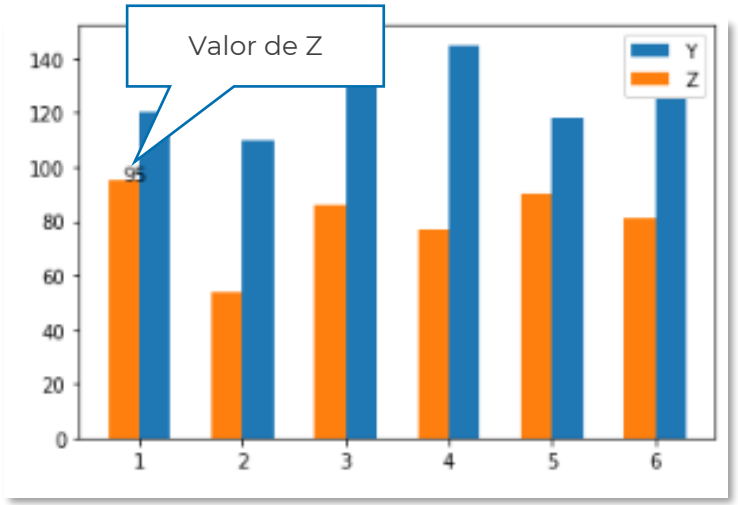
Módulo 8 – Visualização de dados no Python: Adicionando rótulo nos dados (annotate)

Substituindo os valores de X e Y pela posição do primeiro valor de Z no código:

```
fig, ax = plt.subplots()

w=0.3

ax.bar(base.X+w/2, base.Y, label="Y",width=w)
ax.bar(base.X-w/2, base.Z, label="Z",width=w)

ax.annotate(base.Z[0], (base.X[0]-w/2,base.Z[0]))

plt.legend()

plt.show()
```

Posição X

Posição Y



Módulo 8 – Visualização de dados no Python: Adicionando rótulo nos dados (annotate)

Usando o **for**, é possível aplicar o **annotate** para todas as barras.

O **for** percorre uma lista, por exemplo:

for i **in** range(0,6)**:**
    print(i)

```
for i in range(0,6):
    print(i)
```

Percorre todos os valores de 0 a 6

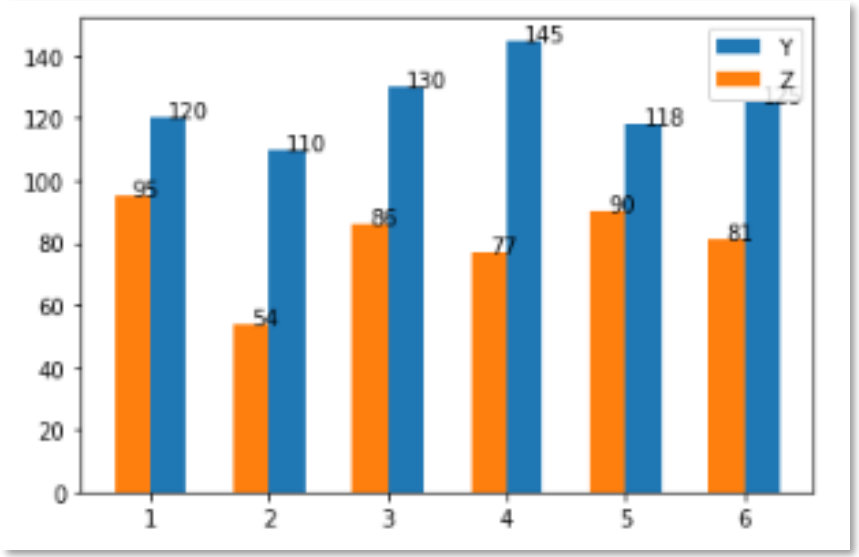
```
0
1
2
3
4
5
```

Módulo 8 – Visualização de dados no Python: Adicionando rótulo nos dados (annotate)

Utilizando o for na função annotate

```
fig, ax = plt.subplots()
w=0.3

ax.bar(base.X+w/2, base.Y, label="Y",width=w)
ax.bar(base.X-w/2, base.Z, label="Z",width=w)

for i in range(0,6):

        ax.annotate(base.Z[i],(base.X[i]-w/2,base.Z[i]))
        ax.annotate(base.Y[i],(base.X[i]+w/2,base.Y[i]))

plt.legend()
plt.show()
```

No **eixo Y**, ao invés de ser –w/2 é **+w/2**



Módulo 8 – Visualização de dados no Python: Adicionando rótulo nos dados (annotate)

Por fim, vamos usar os parâmetros do annotate para melhorar a visualização:

ha → alinhamento horizontal

xytext → deslocamento x,y do texto em relação a um referencial

textcoords → referencial que vamos fazer o deslocamento acima

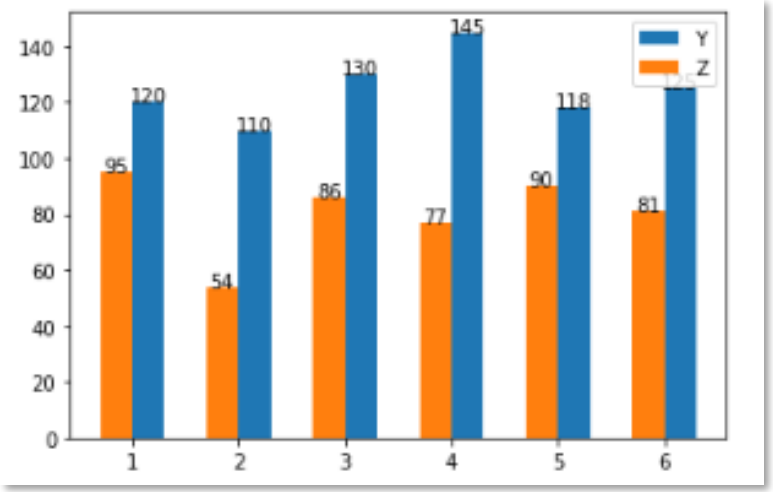
fontsize → tamanho da fonte

fontweight → colocando em negrito

Módulo 8 – Visualização de dados no Python: Adicionando rótulo nos dados (annotate)

Vamos centralizar os valores utilizando o **ha**:

```
fig, ax = plt.subplots()
w=0.3

ax.bar(base.X+w/2, base.Y, label="Y",width=w)
ax.bar(base.X-w/2, base.Z, label="Z",width=w)

for i in range(0,6):

        ax.annotate(base.Z[i],(base.X[i]-w/2,base.Z[i]),ha="center“)
        ax.annotate(base.Y[i],(base.X[i]+w/2,base.Y[i]), ha="center“)

plt.legend()
plt.show()
```



Centralizando os valores

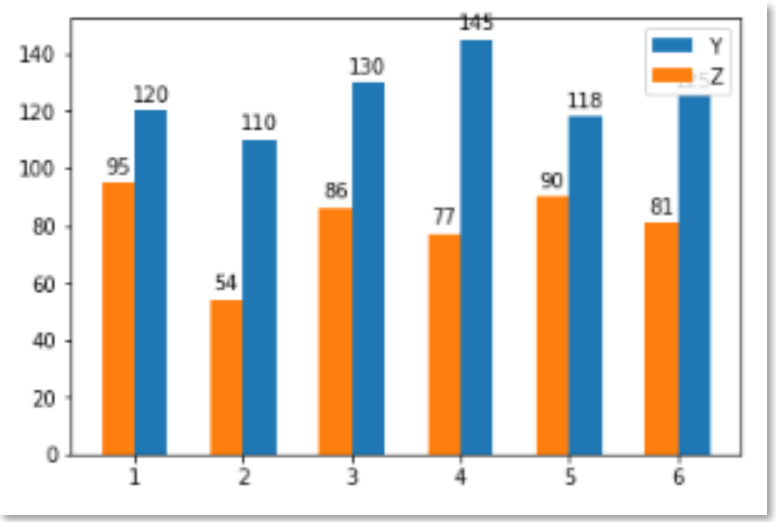
Módulo 8 – Visualização de dados no Python: Adicionando rótulo nos dados (annotate)

Vamos utilizar o **xytext** para deslocar a posição:

Estrutura: **xytext**(x,y),textcoords="offset points" ou "offset pixels"

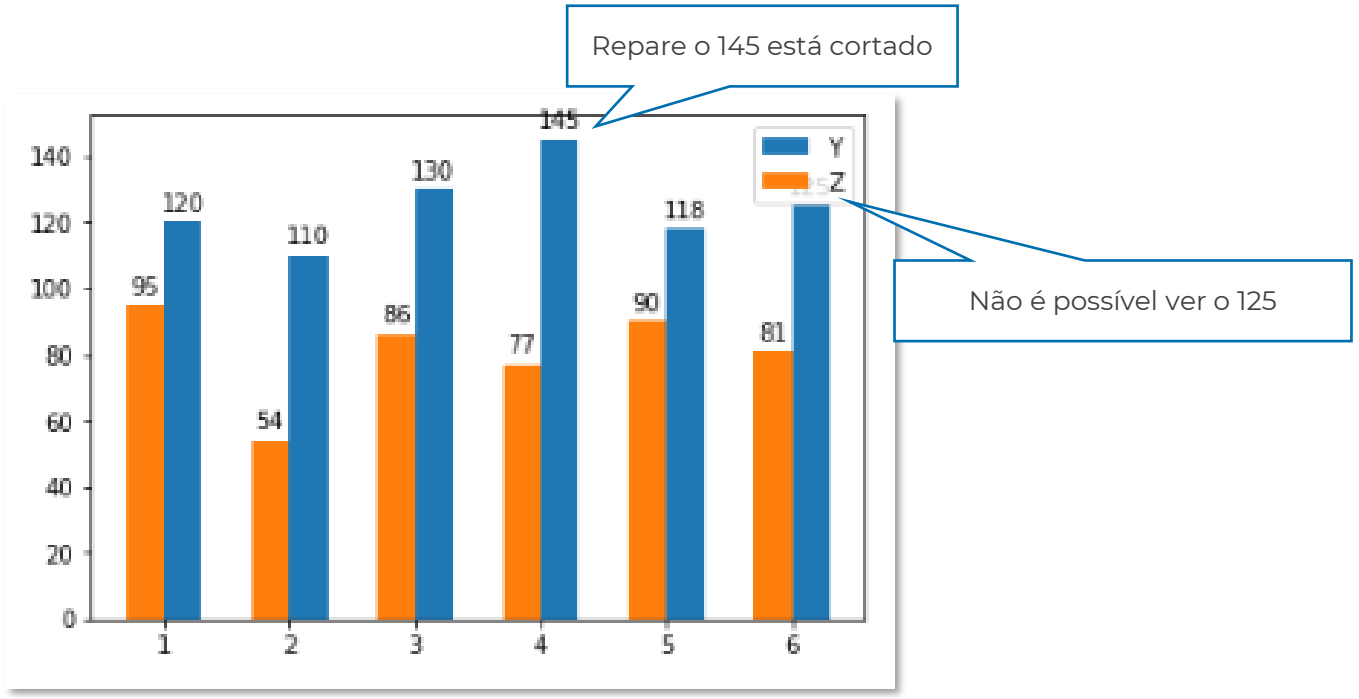
Aplicando o **xytext** na parte do **annotate** do nosso código:

```
ax.annotate(base.Z[i],(base.X[i]-w/2,base.Z[i]),ha="center",xytext=(0,5),textcoords="offset points")

ax.annotate(base.Y[i],(base.X[i]+w/2,base.Y[i]),ha="center",xytext=(0,5),textcoords="offset points")
```



Módulo 8 – Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado

Agora que colocamos o rótulo de dados, vamos melhorar essa visualização.

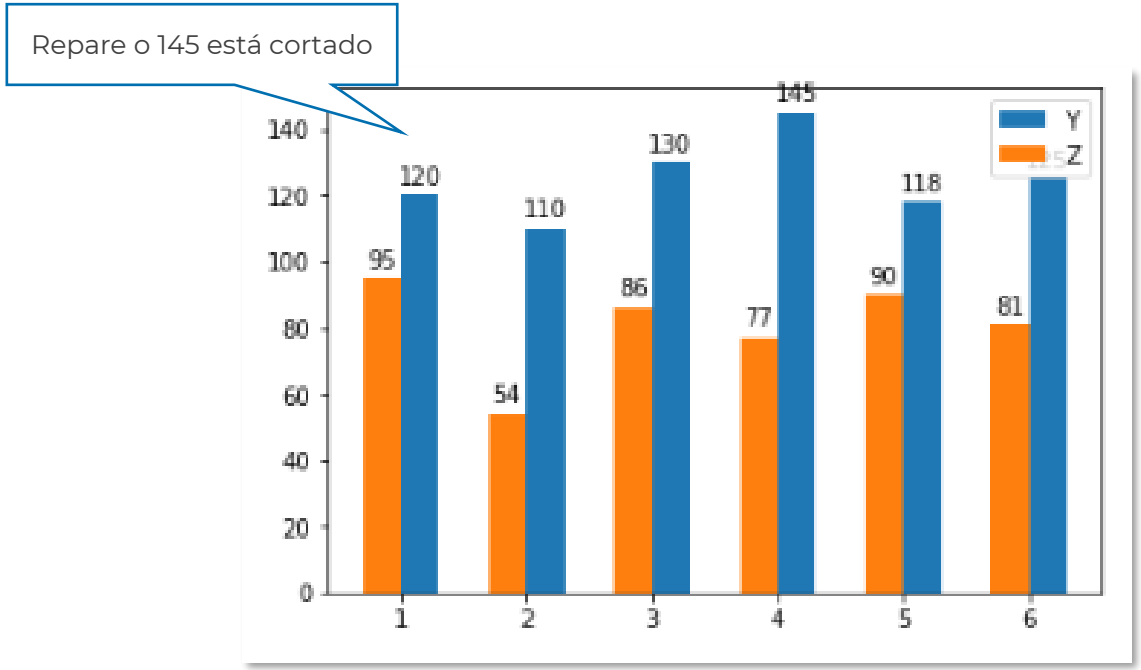


Módulo 8 – Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado

Vamos ajustar o limite do eixo para 150 utilizando o código **.set_yticks que** vai permitir alterar o intervalo do eixo y.

Estrutura:
ax**.set_yticks**([início,final])

Vamos aplicar o **.set_yticks n**o nosso código:

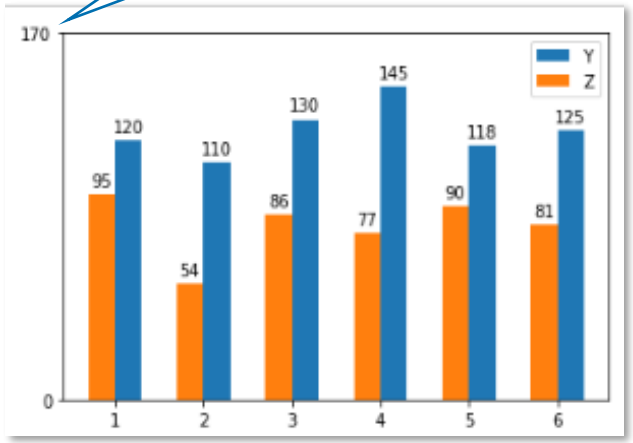


Módulo 8 – Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado

```
fig, ax = plt.subplots()
w=0.3
ax.bar(base.X+w/2, base.Y, label="Y",width=w)
ax.bar(base.X-w/2, base.Z, label="Z",width=w)
for i in range(0,6):
        ax.annotate(base.Z[i],(base.X[i]-w/2,base.Z[i]),ha="center“,
xytext=(0,5),textcoords="offset points")
        ax.annotate(base.Y[i],(base.X[i]+w/2,base.Y[i]),ha="center“
,xytext=(0,5),textcoords="offset points")

ax.set_yticks([0,170])

plt.legend()
plt.show()
```

Até 170 no eixo Y

Eixo y até 170

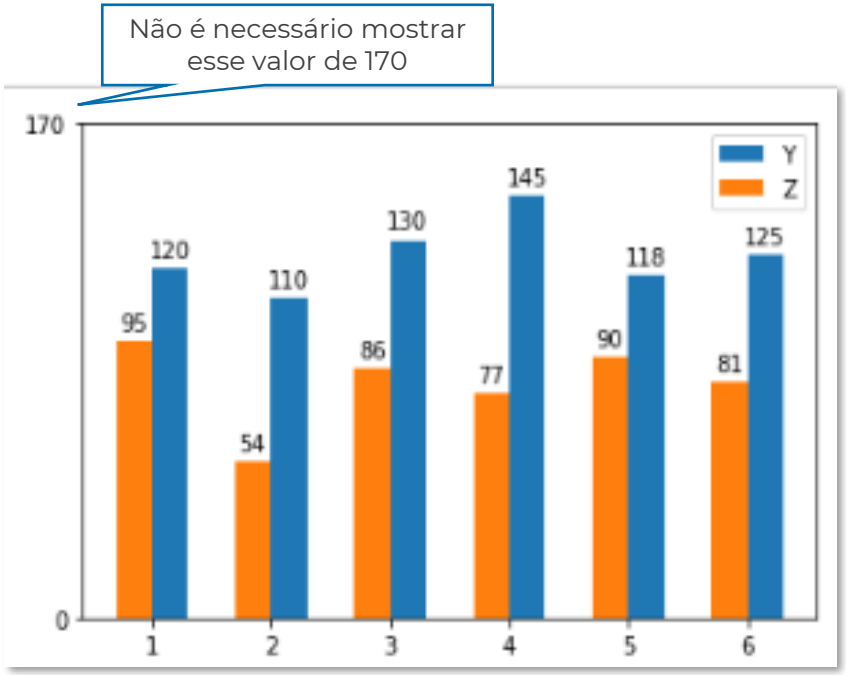


Módulo 8 – Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado

Já que estamos com todos os rótulos de dados, esse eixo y tona-se desnecessário. Para isso vamos utilizar a função **.yaxis.set_visible**

Estrutura:
ax.**yaxis.set_visible**(False)

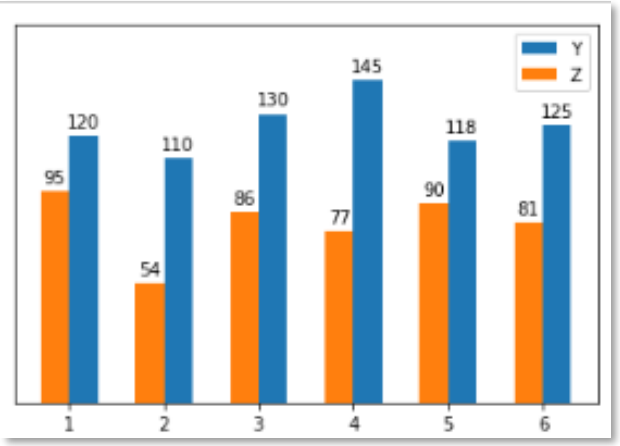
Vamos aplicar o **. yaxis.set_visible** no nosso código:



Módulo 8 – Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado

```
fig, ax = plt.subplots()
w=0.3
ax.bar(base.X+w/2, base.Y, label="Y",width=w)
ax.bar(base.X-w/2, base.Z, label="Z",width=w)
for i in range(0,6):
        ax.annotate(base.Z[i],(base.X[i]-w/2,base.Z[i]),ha="center“,
xytext=(0,5),textcoords="offset points")
        ax.annotate(base.Y[i],(base.X[i]+w/2,base.Y[i]),ha="center“
,xytext=(0,5),textcoords="offset points")

ax.set_yticks([0,170])
ax.yaxis.set_visible(False)
plt.legend()
plt.show()
```

Retira o valor do eixo Y

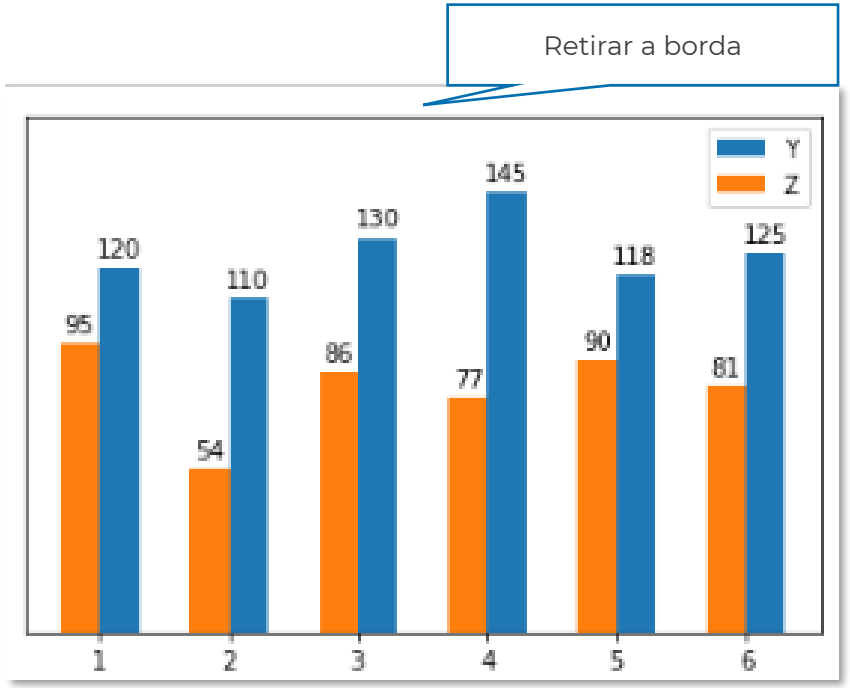


Módulo 8 – Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado

Além disso, podemos retirar a borda do gráfico utilizando a função **.spines[].set_visible**

Estrutura:
ax**.spines[].set_visible**(False)

Vamos aplicar essa função no nosso código:



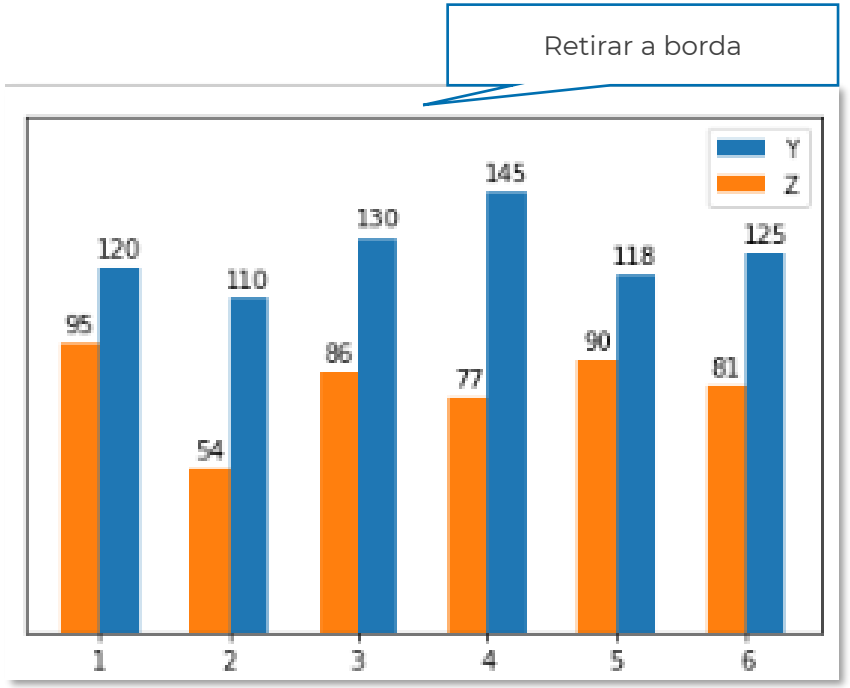
Módulo 8 – Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado

Além disso, podemos retirar a borda do gráfico utilizando a função **.spines[].set_visible**

Estrutura:
ax**.spines[**posição**].set_visible**(False)

Posições:
- Right
- Top
- Left
- Bottom

Vamos aplicar essa função no nosso código:

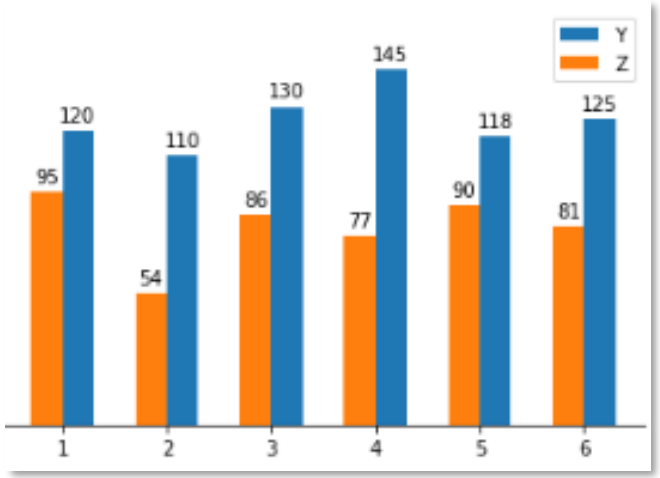


Módulo 8 – Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado

Aplicando no código abaixo:

```
fig, ax = plt.subplots()

w=0.3

ax.bar(base.X+w/2,base.Y, label="Y",width=w)
ax.bar(base.X-w/2,base.Z, label="Z",width=w)

for i in range(0,6):
    ax.annotate(base.Z[i],(base.X[i]-w/2,base.Z[i]),ha="center",xytext=(0,5),textcoords="offset points")
    ax.annotate(base.Y[i],(base.X[i]+w/2,base.Y[i]),ha="center",xytext=(0,5),textcoords="offset points")

ax.set_yticks([0,170])
ax.yaxis.set_visible(False)

ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['left'].set_visible(False)

plt.legend()

plt.show()
```

Retirando a borda superior, direita e esquerda

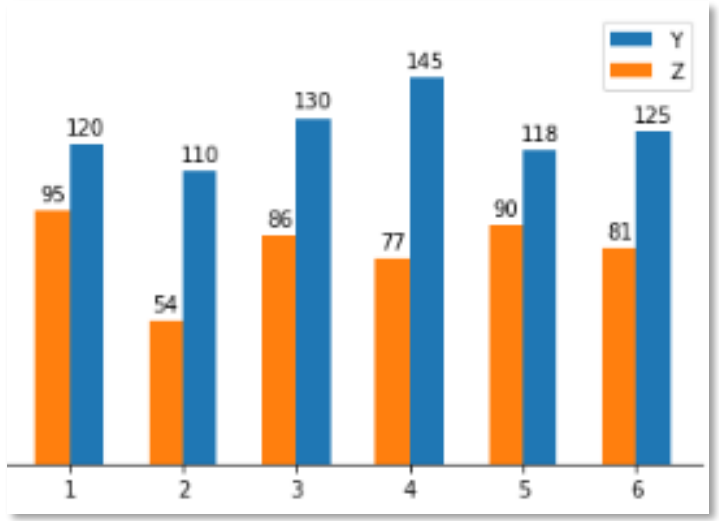


Módulo 8 – Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado

Vamos supor que os valores 5 e 6 do gráfico são projetados e o seu chefe pediu para você diferenciá-los, mudando as cores das barras.

Cores: azul (#C6F0F5) e laranja (#FFC174)

Para isso vamos separar a nossa base em duas.



Módulo 8 – Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado

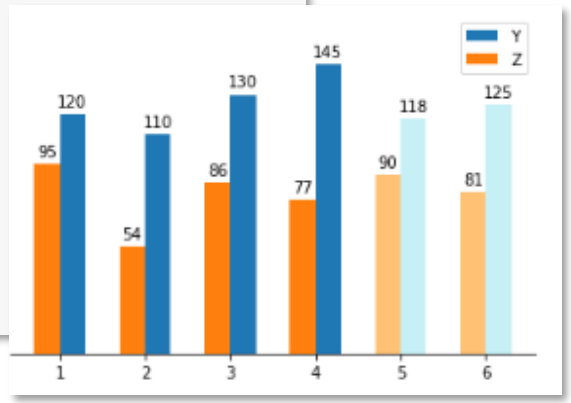
Aplicando no código abaixo:

Passando os 4 primeiros elementos

Passando os 2 últimos elementos

Alterando a cor do gráfico

```
fig, ax = plt.subplots()

w=0.3

ax.bar(base.X[0:4]+w/2,base.Y[0:4], label="Y",width=w)
ax.bar(base.X[0:4]-w/2,base.Z[0:4], label="Z",width=w)
ax.bar(base.X[4:6]+w/2,base.Y[4:6],width=w,color="#C6F0F5")
ax.bar(base.X[4:6]-w/2,base.Z[4:6],width=w,color="#FFC174")

for i in range(0,6):
    ax.annotate(base.Z[i],(base.X[i]-w/2,base.Z[i]),ha="center",xytext=(0,5),textcoords="offset points")
    ax.annotate(base.Y[i],(base.X[i]+w/2,base.Y[i]),ha="center",xytext=(0,5),textcoords="offset points")

ax.set_yticks([0,170])
ax.yaxis.set_visible(False)

ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['left'].set_visible(False)

plt.legend()

plt.show()
```



Módulo 8 – Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado

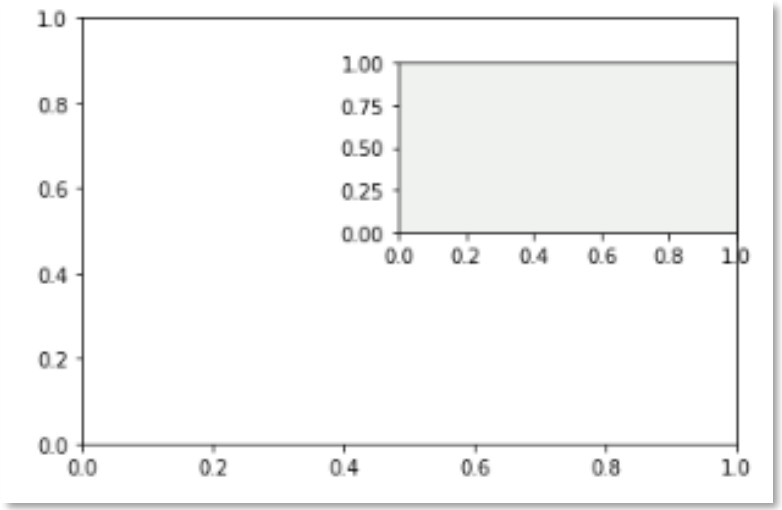
EXTRA: Adicionando retângulos no gráfico para destacar áreas

```
fig, ax = plt.subplots()

ax1 = fig.add_axes([0.5,0.5,0.4,0.3])

rect = ax1.patch

rect.set_facecolor('#E0E6DE')

rect.set_alpha(0.5)

plt.show()
```

[esquerda,fundo,largura,altura] em fração da figura

Fececolor altera a cor do retângulo

Alpha altera a transparência



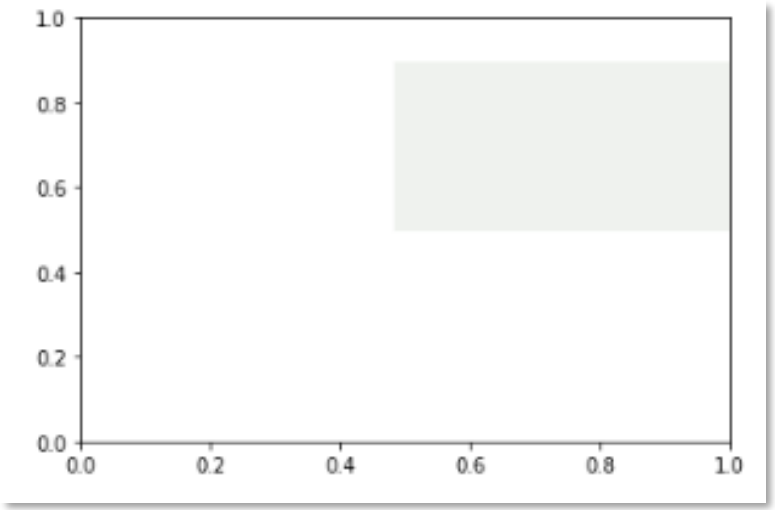
Módulo 8 – Visualização de dados no Python: Retirando as bordas, ajustando os eixos e separando realizado x projetado

EXTRA: Adicionando retângulos no gráfico para destacar áreas

```
fig, ax = plt.subplots()
ax1 = fig.add_axes([0.5,0.5,0.4,0.3])
rect = ax1.patch
rect.set_facecolor('#E0E6DE')
rect.set_alpha(0.5)
ax1.xaxis.set_visible(False)
ax1.yaxis.set_visible(False)
ax1.spines['top'].set_visible(False)
ax1.spines['left'].set_visible(False)
ax1.spines['right'].set_visible(False)
ax1.spines['bottom'].set_visible(False)
plt.show()
```

Retira os valores do retângulo

.set_visible retira as bordas



Módulo 8 – Boas práticas de visualização no Python: Separando em dois gráficos e alterando o tipo de gráfico

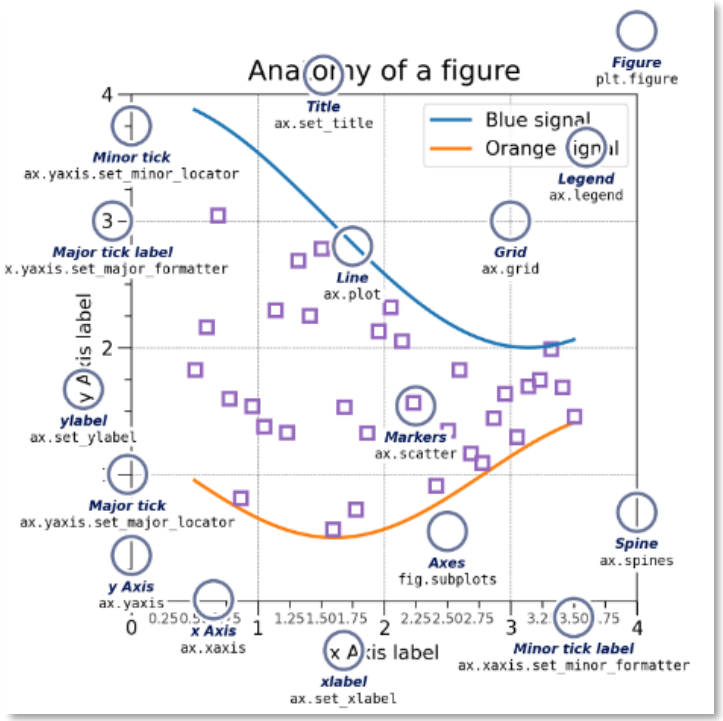
Agora vamos montar um visual de apresentação:

O primeiro passo que vamos fazer é importar as bibliotecas e vamos utilizar essa figura ao lado como base:

import pandas **as** pd

import matplotlib.pyplot **as** plt

import numpy **as** np



Módulo 8 – Boas práticas de visualização no Python: Separando em dois gráficos e alterando o tipo de gráfico

Vamos importar a nossa base visualizar utilizando **head**:

base = **pd.read_**excel("Visualização de dados no Python.xlsx")

base.**head**()

Tempo médio de atendimento

	Data	Pedidos	Entregas	EntregaNovoCD	TMAantigoCD	TMAnovoCD
0	2021-10-01	1230	1230	1230	1	1
1	2021-11-01	1235	1235	1235	1	1
2	2021-12-01	1280	1280	1280	1	1
3	2022-01-01	1310	1310	1310	1	1
4	2022-02-01	1572	1493	1493	2	2

Módulo 8 – Boas práticas de visualização no Python: Separando em dois gráficos e alterando o tipo de gráfico

Vamos entender um pouco a nossa base digitando o código abaixo:

base

	Data	Pedidos	Entregas	EntregaNovoCD	TMAantigoCD	TMAnovoCD
0	2021-10-01	1230	1230	1230	1	1
1	2021-11-01	1235	1235	1235	1	1
2	2021-12-01	1280	1280	1280	1	1
3	2022-01-01	1310	1310	1310	1	1
4	2022-02-01	1572	1493	1493	2	2
5	2022-03-01	2122	1910	1910	3	3
6	2022-04-01	3183	2546	2546	6	4
7	2022-05-01	4775	3581	4300	8	3
8	2022-06-01	5251	3938	5200	8	3
9	2022-07-01	5779	4136	6100	8	2
10	2022-08-01	6355	4356	7000	7	2
11	2022-09-01	6994	4850	7900	8	1

Tempo médio de atendimento menor

Módulo 8 – Boas práticas de visualização no Python: Separando em dois gráficos e alterando o tipo de gráfico

Antes de ir para os gráficos vamos verificar se existe alguma informação vazia na nossa base:

base.**info**()

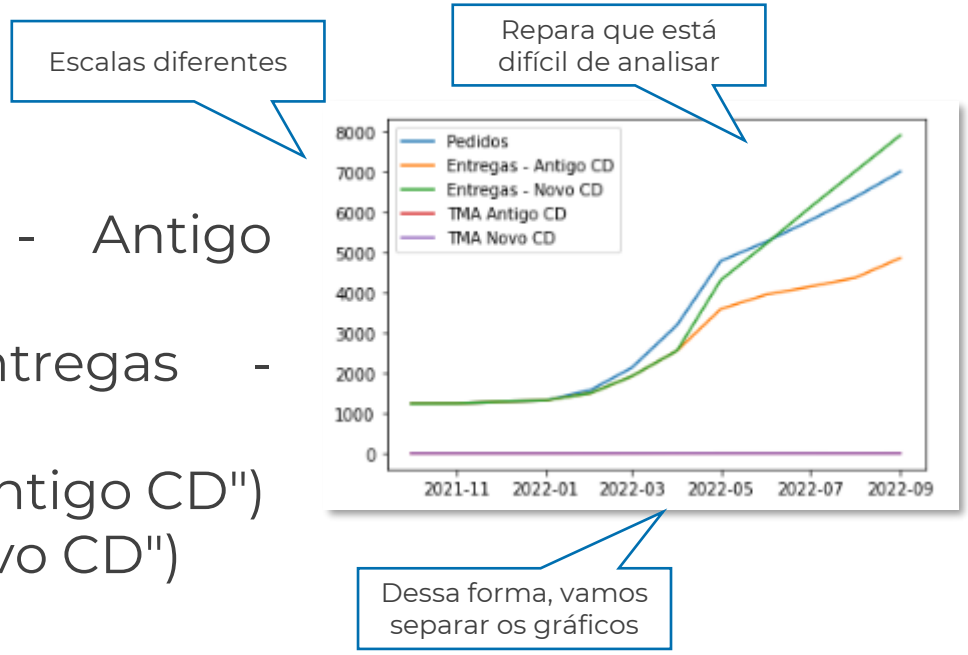
Nenhuma informação vazia

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12 entries, 0 to 11
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Data           12 non-null     datetime64[ns]
 1   Pedidos        12 non-null     int64
 2   Entregas       12 non-null     int64
 3   EntregaNovoCD  12 non-null     int64
 4   TMAantigoCD    12 non-null     int64
 5   TMAnovoCD      12 non-null     int64
dtypes: datetime64[ns](1), int64(5)
memory usage: 704.0 bytes
```

Módulo 8 – Boas práticas de visualização no Python: Separando em dois gráficos e alterando o tipo de gráfico

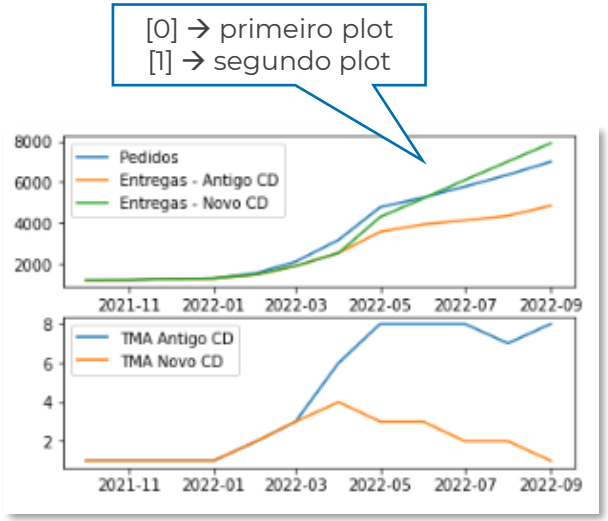
Vamos criar um plot simples de todos os dados:

```
fig, ax = plt.subplots()

ax.plot(base.Data,base.Pedidos, label="Pedidos")
ax.plot(base.Data,base.Entregas, label="Entregas - Antigo CD")
ax.plot(base.Data,base.EntregaNovoCD, label="Entregas - Novo CD")
ax.plot(base.Data,base.TMAantigoCD, label="TMA Antigo CD")
ax.plot(base.Data,base.TMAnovoCD, label="TMA Novo CD")

plt.legend()
plt.show()
```



Módulo 8 – Boas práticas de visualização no Python: Separando em dois gráficos e alterando o tipo de gráfico

```
fig, ax = plt.subplots(nrows=2,ncols=1)

ax[0].plot(base.Data,base.Pedidos, label="Pedidos")
ax[0].plot(base.Data,base.Entregas, label="Entregas - Antigo CD")
ax[0].plot(base.Data,base.EntregaNovoCD, label="Entregas - Novo CD")
ax[0].legend()

ax[1].plot(base.Data,base.TMAantigoCD, label="TMA Antigo CD")
ax[1].plot(base.Data,base.TMAnovoCD, label="TMA Novo CD")
ax[1].legend()
plt.show()
```



Módulo 8 – Boas práticas de visualização no Python: Separando em dois gráficos e alterando o tipo de gráfico

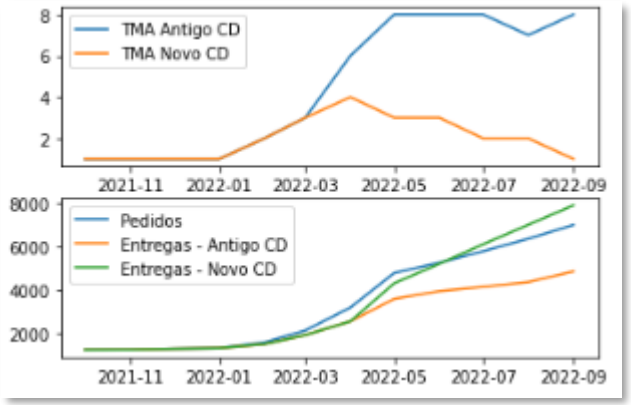
Vamos melhorar a visualização fazendo os seguintes itens:

- Vamos alterar a ordem dos gráficos (colocar o de TMA por cima)
- Podemos aumentar o tamanho dos gráficos
- Vamos aumentar proporcionalmente o gráfico de Pedidos x Entregas em relação ao de TMA

Módulo 8 – Boas práticas de visualização no Python: Separando em dois gráficos e alterando o tipo de gráfico

- Para alterar a ordem do gráfico vamos trocar o que estava com [0] por um e vice-versa.

```
fig, ax = plt.subplots(nrows=2,ncols=1)

ax[1].plot(base.Data,base.Pedidos,label="Pedidos")   ax[0].plot(base.Data,base.Entregas,
label="Entregas - Antigo CD")
ax[1].plot(base.Data,base.EntregaNovoCD, label="Entregas - Novo CD")
ax[1].legend()

ax[0].plot(base.Data,base.TMAantigoCD, label="TMA Antigo CD")
ax[0].plot(base.Data,base.TMAnovoCD, label="TMA Novo CD")
ax[0].legend()
plt.show()
```



Módulo 8 – Boas práticas de visualização no Python: Separando em dois gráficos e alterando o tipo de gráfico

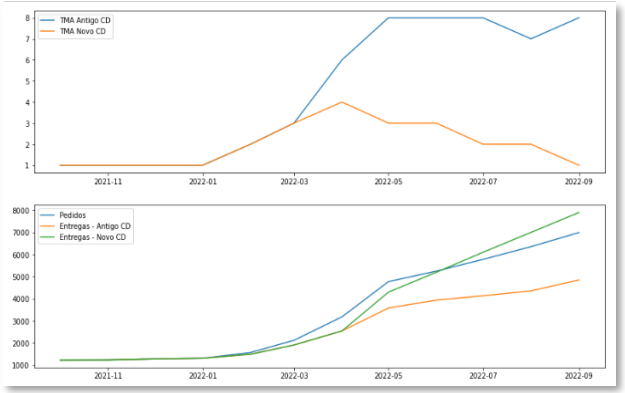
- Para aumentar o tamanho do gráfico:

fig, ax = **plt.subplots**(nrows=2,ncols=1, **figsize**=(15,9))

Aumenta o tamanho do gráfico

ax[1].plot(**base**.**Data**,**base**.**Pedidos**,label="Pedidos") ax[0].plot(**base**.**Data**,**base**.**Entregas**, label="Entregas - Antigo CD")
ax[1].plot(**base**.**Data**,**base**.**EntregaNovoCD**, label="Entregas - Novo CD")
ax[1].**legend**()

ax[0].**plot**(base.**Data**,**base**.**TMAantigoCD**, label="TMA Antigo CD")
ax[0].**plot**(**base**.**Data**,**base**.**TMAnovoCD**, label="TMA Novo CD")
ax[0].**legend**()
plt.show()

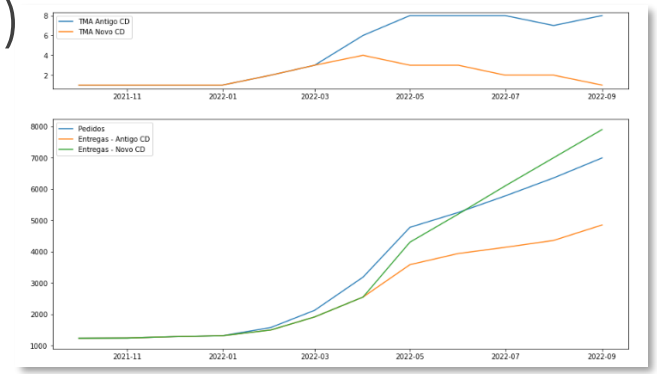


Módulo 8 – Boas práticas de visualização no Python: Separando em dois gráficos e alterando o tipo de gráfico

- Para aumentar proporcionalmente o tamanho do gráfico:

Aumenta o tamanho do gráfico proporcionalmente

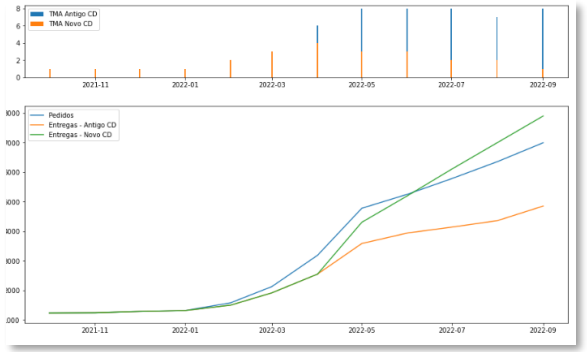
```
fig, ax = plt.subplots(nrows=2,ncols=1, figsize=(15,9), gridspec_kw={'height_ratios': [1,3]})

ax[1].plot(base.Data,base.Pedidos,label="Pedidos")   ax[0].plot(base.Data,base.Entregas,
label="Entregas - Antigo CD")
ax[1].plot(base.Data,base.EntregaNovoCD, label="Entregas - Novo CD")
ax[1].legend()

ax[0].plot(base.Data,base.TMAantigoCD, label="TMA Antigo CD")
ax[0].plot(base.Data,base.TMAnovoCD, label="TMA Novo CD")
ax[0].legend()
plt.show()
```



Módulo 8 – Boas práticas de visualização no Python: Separando em dois gráficos e alterando o tipo de gráfico

- Para alterar o tipo de gráfico basta trocar **plot** por **bar**:

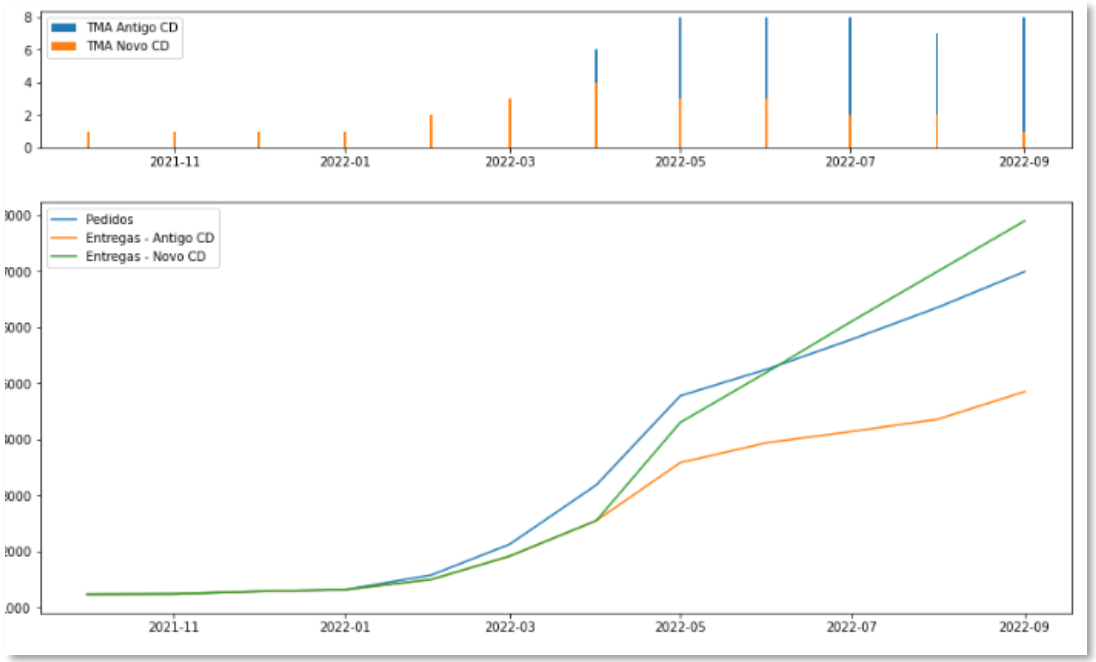
```
fig, ax = plt.subplots(nrows=2,ncols=1, figsize=(15,9), gridspec_kw={'height_ratios': [1,3]})

ax[1].bar(base.Data,base.Pedidos,label="Pedidos")    ax[0].plot(base.Data,base.Entregas,
label="Entregas - Antigo CD")
ax[1].bar(base.Data,base.EntregaNovoCD, label="Entregas - Novo CD")
ax[1].legend()

ax[0].plot(base.Data,base.TMAantigoCD, label="TMA Antigo CD")
ax[0].plot(base.Data,base.TMAnovoCD, label="TMA Novo CD")
ax[0].legend()
plt.show()
```



Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Repare que no gráfico cada barra é um dia, como vamos melhorar isso?

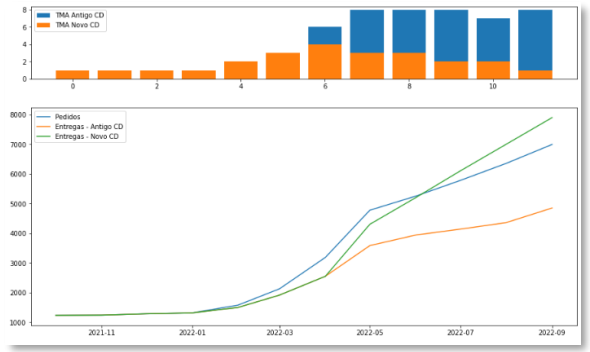
Vamos transformar essas datas em valores inteiros.



Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

```
fig,ax=plt.subplots(nrows=2,ncols=1,figsize=(15,9),gridspec_kw={'height_ratios': [1,3]})

ax[0].bar(np.arange(len(base)),base.TMAantigoCD, label="TMA Antigo CD")
ax[0].bar(np.arange(len(base)),base.TMAnovoCD, label="TMA Novo CD")
ax[0].legend()

ax[1].plot(base.Data,base.Pedidos, label="Pedidos")
ax[1].plot(base.Data,base.Entregas, label="Entregas - Antigo CD")
ax[1].plot(base.Data, base.EntregaNovoCD, label="Entregas - Novo
ax[1].legend()

plt.show()
```

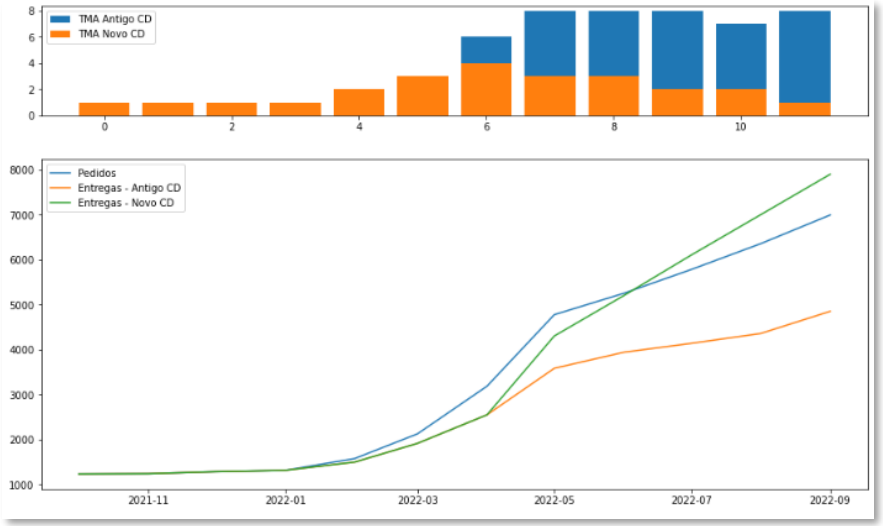
Funciona mesmo se a nossa base aumentar



Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

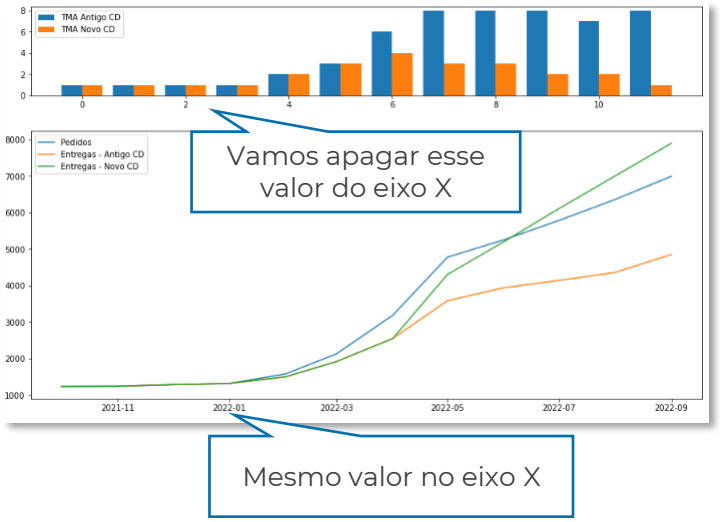
Agora podemos definir o width da coluna e deslocar as colunas em relação a esse width.

Vamos supor um valor de **wid** = 0.4 e aplicar na primeira parte do código.



Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

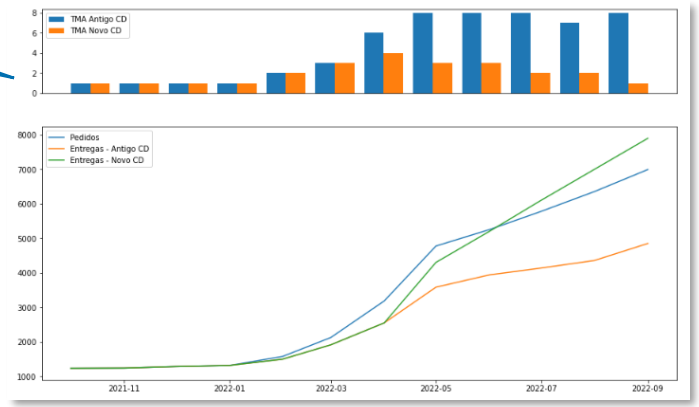
```
fig,ax=plt.subplots(nrows=2,ncols=1,figsize=(15,9),gridspec_kw={'height_ratios': [1,3]})

wid = 0.4

ax[0].bar(np.arange(len(base))-wid/2,base.TMAantigoCD,width=wid,label="TMA Antigo CD“)
ax[0].bar(np.arange(len(base))+wid/2,base.TMAnovoCD,width=wid,label="TMA Novo CD")
ax[0].legend()

plt.show()
```



Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

```
fig,ax=plt.subplots(nrows=2,ncols=1,figsize=(15,9),gridspec_kw={'height_ratios': [1,3]})

wid = 0.4

ax[0].bar(np.arange(len(base))-wid/2,base.TMAantigoCD,width=wid,label="TMA Antigo CD“)
ax[0].bar(np.arange(len(base))+wid/2,base.TMAnovoCD,width=wid,label="TMA Novo CD")
ax[0].legend()
ax[0].xaxis.set_visible(False)
plt.show()
```

Eixo y não está tão claro



Retira o no eixo X

Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Vamos utilizar o **annotate** e todas as outras funções para colocar um rótulo de dados.

- ha → alinhamento horizontal
- xytext → deslocamento x,y do texto em relação a um referencial
- Textcoords → referencial que vamos fazer o deslocamento acima
- fontsize → tamanho da fonte
- fontweight → colocando em negrito

Vamos utilizar o **annotate** e todas as outras funções para colocar um rótulo de dados.
Para a primeira parte do código:

```
for i in np.arange(0,12):
    ax[0].annotate('{:,.0f}'.format(base.TMAantigoCD[i],(iwid/2,base.TMAantigoCD[i]),
    ha="center",
    xytext=(0,3),
    textcoords="offset points",
    fontsize=10,
    fontweight='bold')
```

Alinhamento

Formatando o número

Colocando a posição do elemtno

Deslocamento x,y do texto

Referencial que vamos fazer o deslocamento

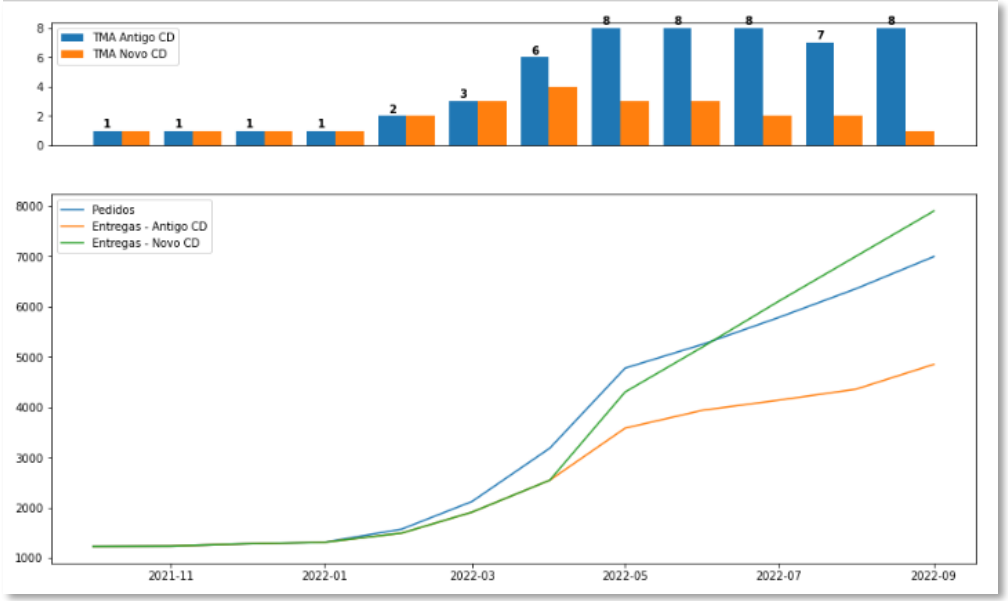
Tamanho da fonte

Colocar em negrito

Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Vamos utilizar o **annotate** e todas as outras funções para colocar um rótulo de dados.
Para a primeira parte do código:

```
for i in np.arange(0,12):
    ax[0].annotate('{:,.0f}'.format(base.TMAantigoCD[i],(iwid/2,base.TMAantigoCD[i]),

    ha="center",

    xytext=(0,3),

    textcoords="offset points",

    fontsize=10,

    fontweight='bold')
```



Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Vamos utilizar o **annotate** e todas as outras funções para colocar um rótulo de dados. Para a segunda parte do código:

```
for i in np.arange(0,12):
    ax[0].annotate('{:,.0f}'.format(base. TMAnovoCD[i],(iwid/2,base. TMAnovoCD[i]),
    ha="center",
    xytext=(0,3),
    textcoords="offset points",
    fontsize=10,
    fontweight='bold')
```

Formatando o número

Colocando a posição do elemtno

Alinhamento

Deslocamento x,y do texto

Referencial que vamos fazer o deslocamento

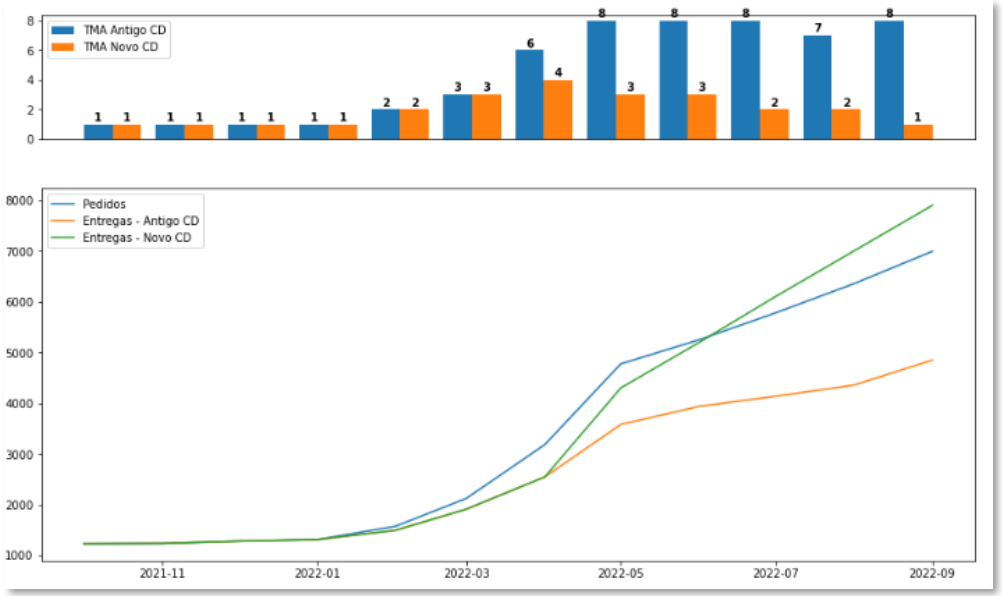
Tamanho da fonte

Colocar em negrito

Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Vamos utilizar o **annotate** e todas as outras funções para colocar um rótulo de dados. Para a segunda parte do código:

```
for i in np.arange(0,12):
    ax[0].annotate('{:,.0f}'.format(base. TMAnovoCD[i],(iwid/2,base. TMAnovoCD[i]),

    ha="center",

    xytext=(0,3),

    textcoords="offset points",

    fontsize=10,

    fontweight='bold')
```



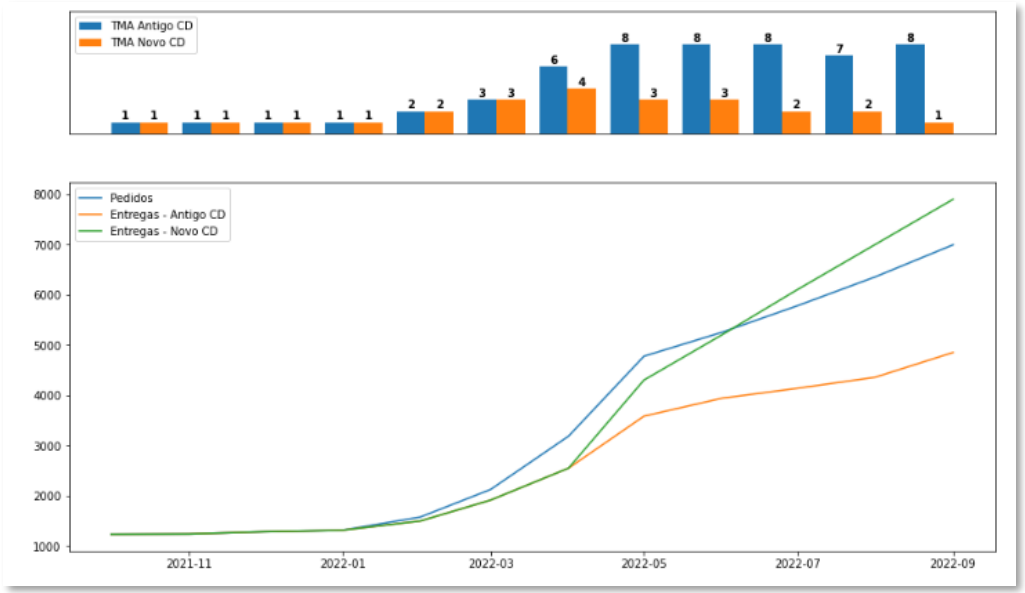
Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Vamos utilizar aumentar o tamanho do eixo y para ajustar o gráfico e retirando os valores:

Para isso colocamos no nosso código os seguintes comando:

ax[0].**set_yticks**(**np.arange**(0,12))

ax[0].**yaxis.set_visible**(False)



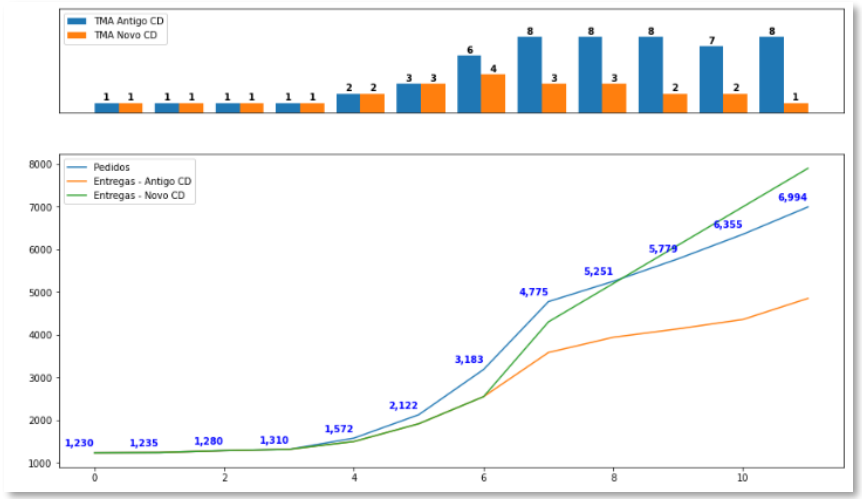
Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Da mesma forma que utilizamos o annotate para o gráfico de barra vamos utilizar para o de linha. Para a primeira linha do gráfico:

```
for i in np.arange(0,12):

    ax[1].annotate('{:,.0f}'.format(base.Pedidos[i]),(i,base.Pedidos[i]),
                ha="right",
                va="top",
                xytext=(0,+15),
                textcoords="offset points",
                fontsize=10,
                fontweight='bold',
                color="blue“)
```

Posição do rótulo em relação a linha

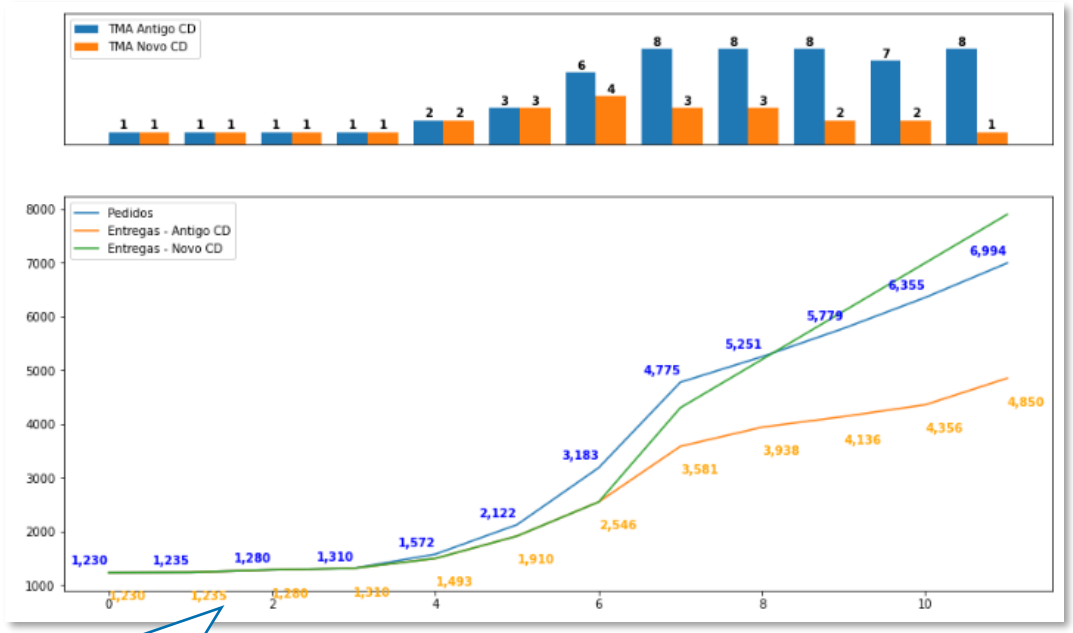
Cor do rótulo



Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Para a segunda linha do gráfico:

```
for i in np.arange(0,12):


ax[1].annotate('{:,.0f}'.format(base.EntregaNovoCD[i]),(i,base. EntregaNovoCD[i]),
                ha="left",
                va="top",
                xytext=(0,-15),
                textcoords="offset points",
                fontsize=10,
                fontweight='bold',
                color="orange")
```



Repare que esses números estão poluindo o gráfico

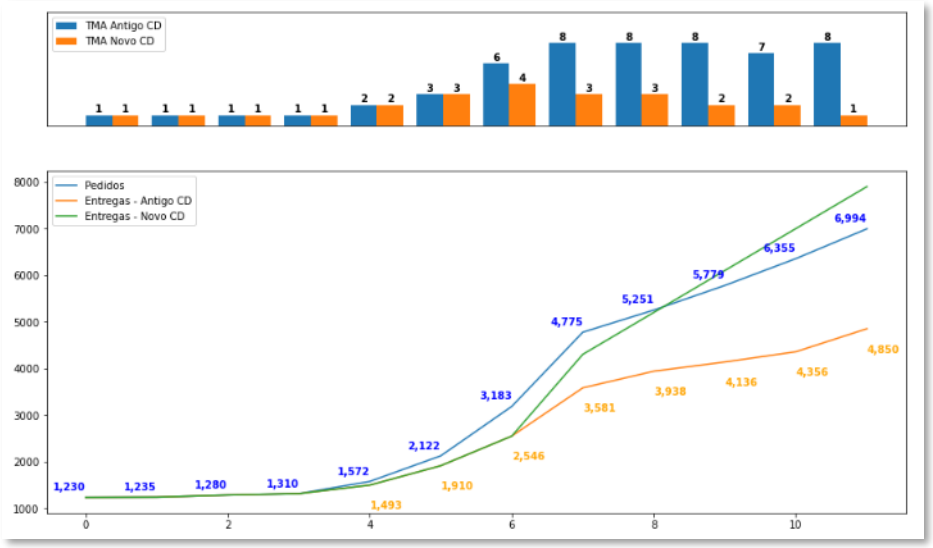
Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Para a segunda linha do gráfico:

```
for i in np.arange(4,12):
```

Vamos alterar a primeira posição do range

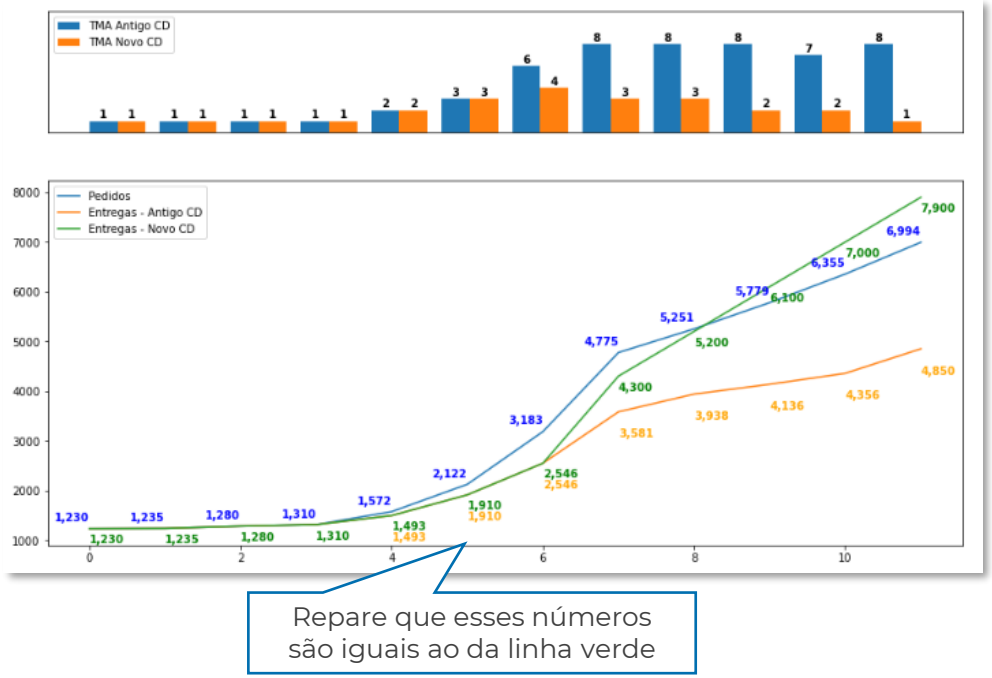
```
ax[1].annotate('{:,.0f}'.format(base.EntregaNovoCD[i]),
(i,base. EntregaNovoCD[i]),
            ha="left",
            va="top",
            xytext=(0,-15),
            textcoords="offset points",
            fontsize=10,
            fontweight='bold',
            color="orange")
```



Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Para a terceira linha do gráfico:

```
for i in np.arange(0,12):


ax[1].annotate('{:,.0f}'.format(base.EntregaNovoCD
[i]),(i,base. EntregaNovoCD[i]),
                ha="left",
                va="bottom",
                xytext=(0,-15),
                textcoords="offset points",
                fontsize=10,
                fontweight='bold',
                color="green")
```



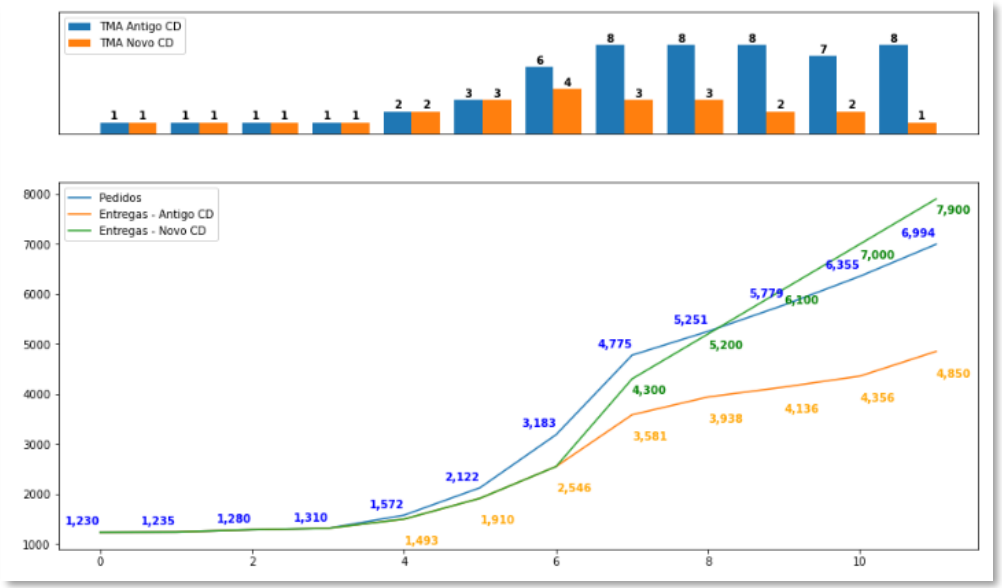
Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Para a terceira linha do gráfico:

```
for i in np.arange(7,12):
```

Vamos alterar a primeira posição do range

```
ax[1].annotate('{:,.0f}'.format(base.EntregaNovoCD
[i]),(i,base. EntregaNovoCD[i]),
            ha="left",
            va="bottom",
            xytext=(0,-15),
            textcoords="offset points",
            fontsize=10,
            fontweight='bold',
            color="green")
```

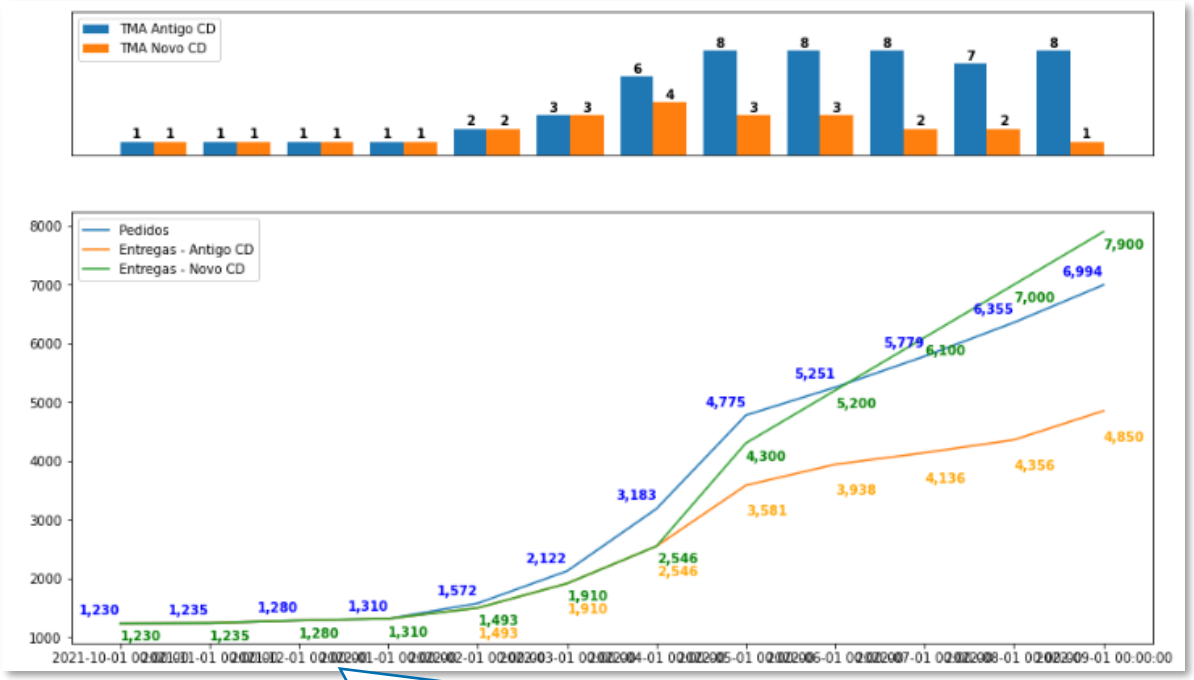


Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Como transformamos a data para número, para ela voltar a aparecer no nosso gráfico como data precisamos utilizar o código abaixo:

ax[1].**xaxis.set_ticks**(**np.arange**(0,12))

ax[1].**set_xticklabels**(**base**.**Data**)

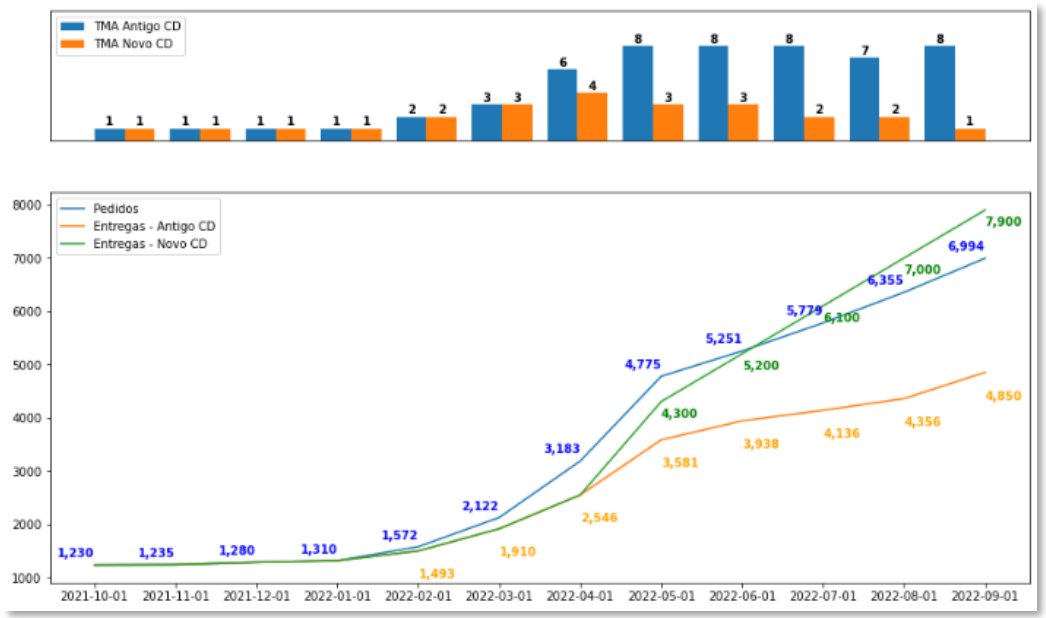


Módulo 8 – Boas práticas de visualização no Python: Ajustando as barras e adicionando rótulo de dados nos gráficos de barra e de linha

Para melhorar a visualização das datas vamos transformar ela em texto.

ax[1].**set_xticklabels**(**np.datetime_as_string**(**base**.**Data**.**values**,unit='D'))

Unida D para ficar ano, dia e mês

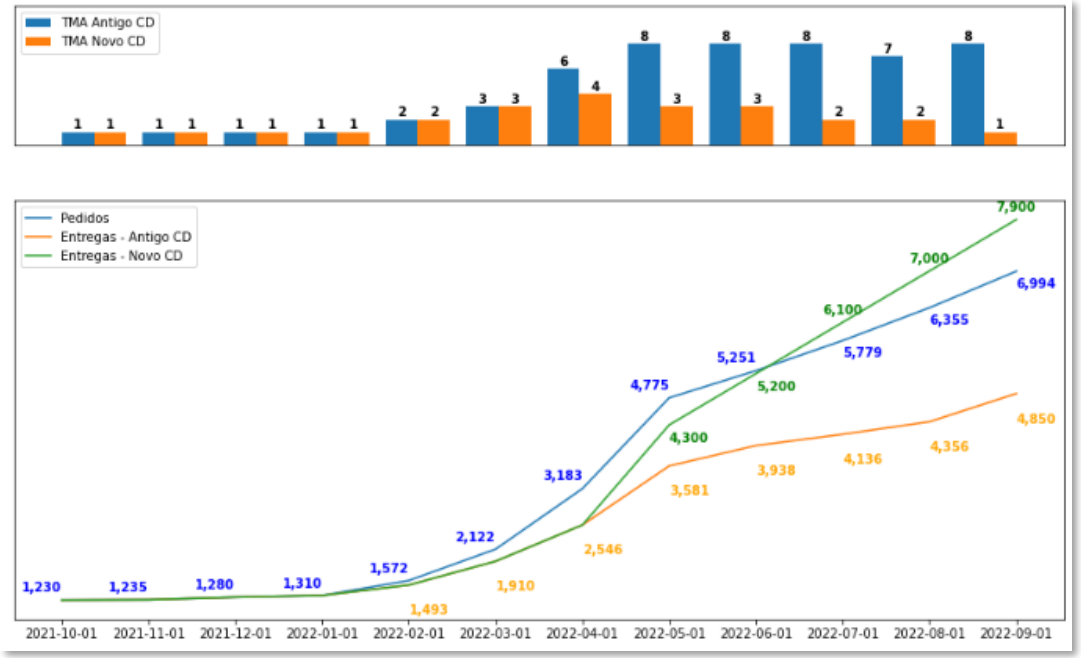


Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

Agora que já colocamos os rótulos de dados para o gráfico de linha podemos melhorar o visual dele:

- Retirar o eixo y utilizando o código abaixo:

ax[1].**yaxis.set_visible**(False)



Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

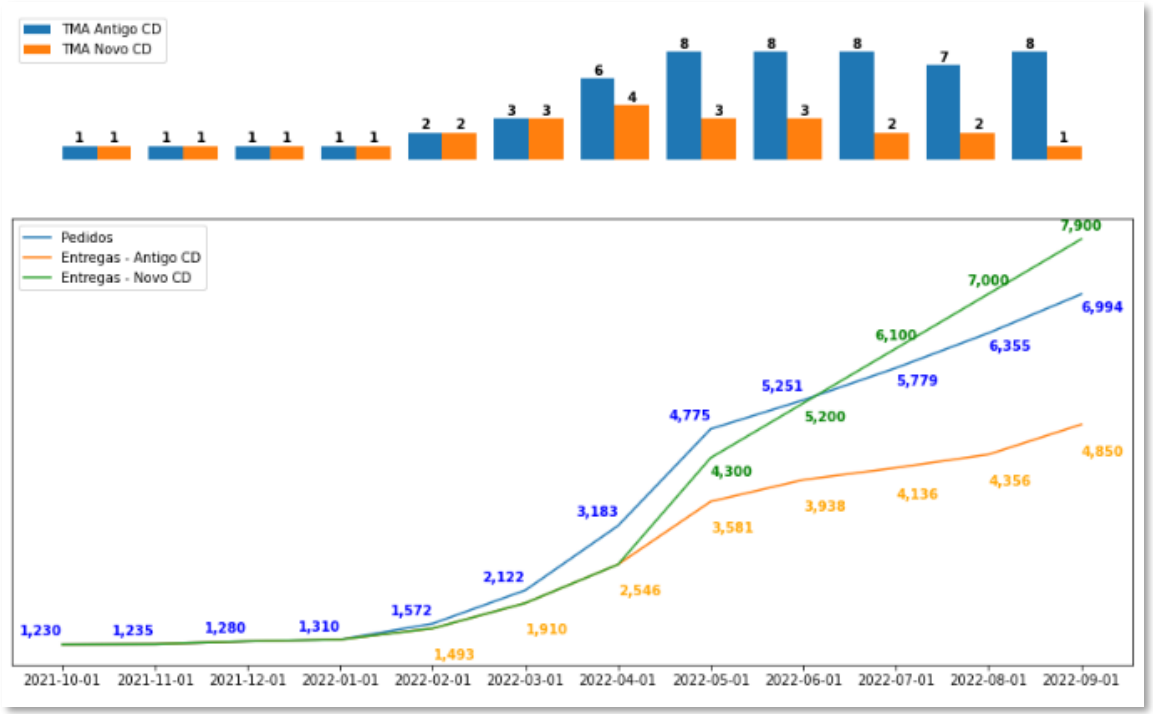
- Retirando a borda do gráfico de barra:

ax[0].**spines**['top'].**set_visible**(False)

ax[0].**spines**['left'].**set_visible**(False)

ax[0].**spines**['right'].**set_visible**(False)

ax[0].**spines**['bottom'].**set_visible**(False)



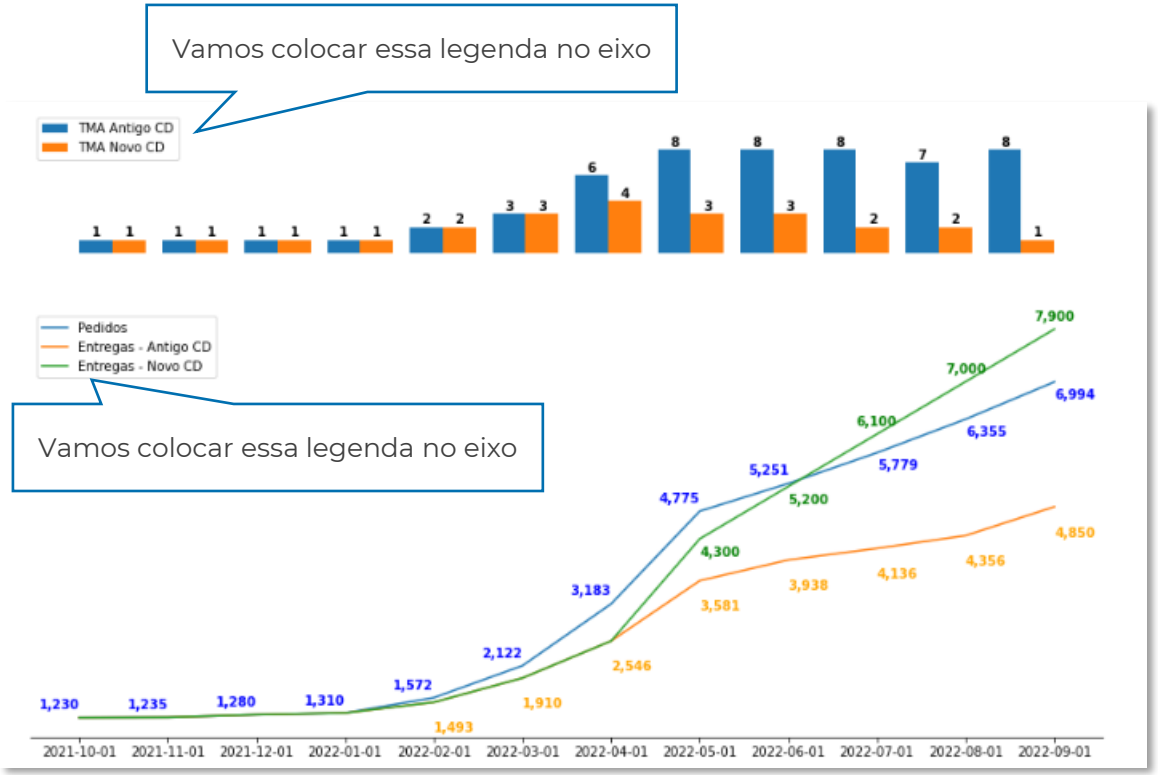
Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

- Retirando a borda do gráfico de linha:

ax[1].**spines**['top'].**set_visible**(False)

ax[1].**spines**['left'].**set_visible**(False)

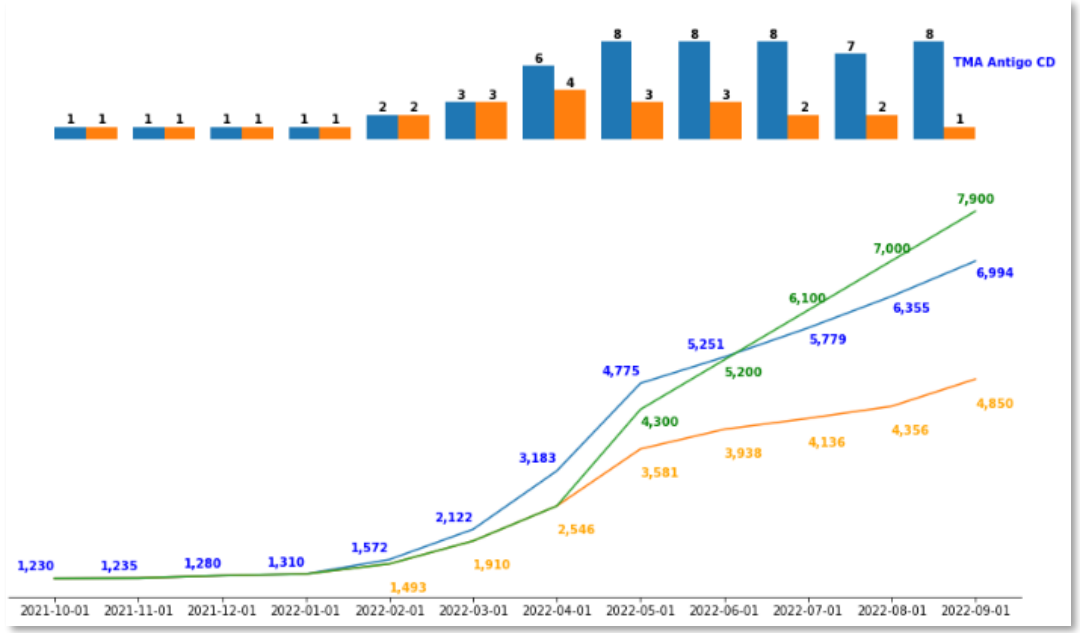
ax[1].**spines**['right'].**set_visible**(False)



Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

- Usando o **annotate** para colocar as legendas na frente dos dados:

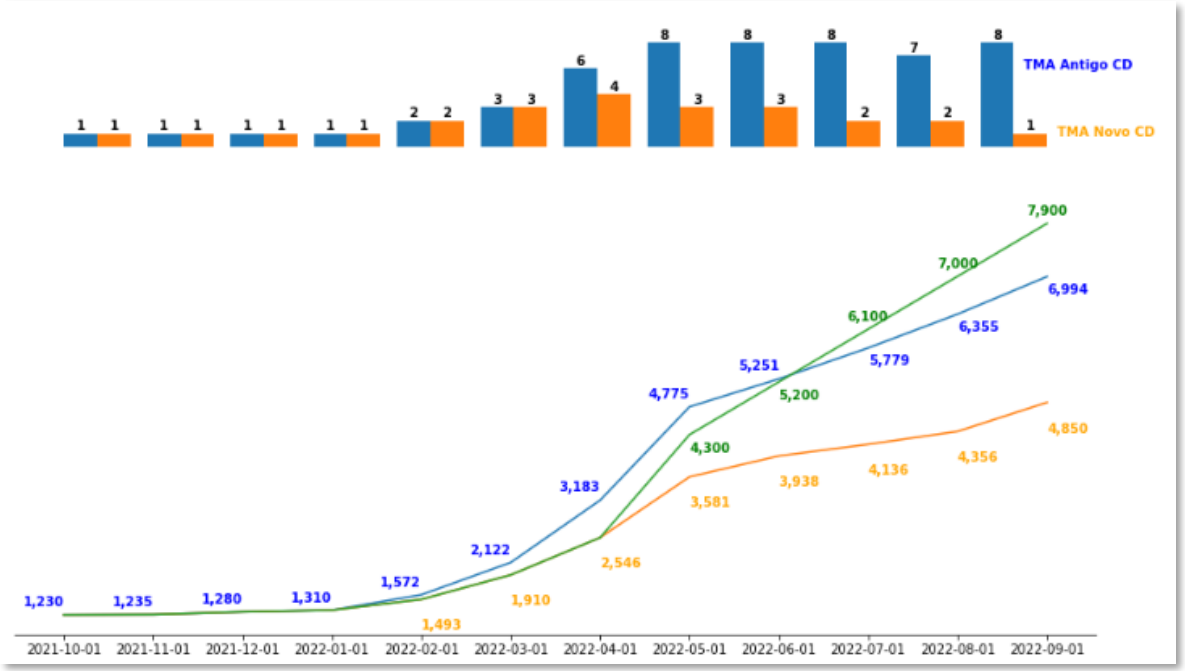
```
ax[0].annotate("TMA Antigo CD",
        (i,base.TMAantigoCD[11]),
        ha="left",
        xytext=(8,-20),
        textcoords="offset points",
        fontsize=10,
        fontweight='bold',
        color="blue"
    )
```



Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

- Usando o **annotate** para colocar as legendas na frente dos dados:

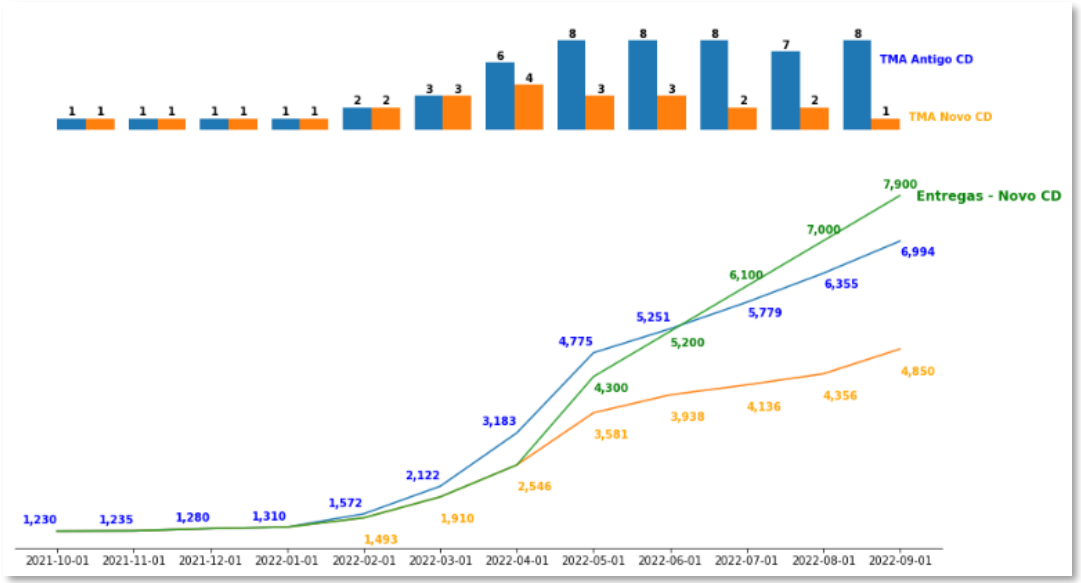
```
ax[0].annotate("TMA Novo CD",
        (i,base. TMAnovoCD[11]),
        ha="left",
        xytext=(8,-20),
        textcoords="offset points",
        fontsize=10,
        fontweight='bold',
        color="orange"
    )
```



Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

- Usando o **annotate** para colocar as legendas na frente dos dados do gráfico das retas:

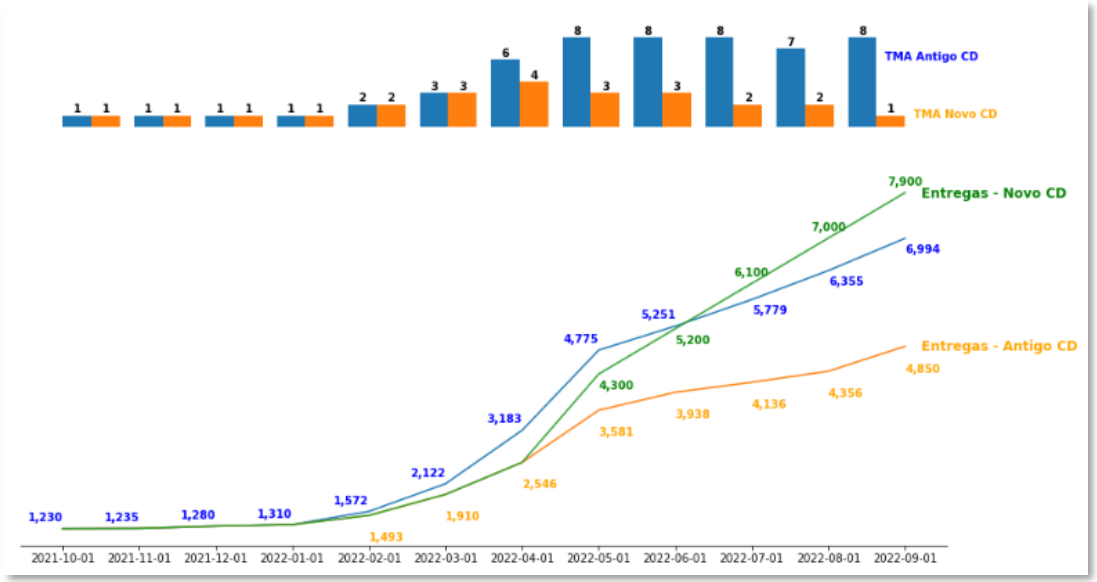
```
ax[1].annotate("Entregas - Novo CD",
        (i,base. EntregaNovoCD[11]),
        ha="left",
        va="center”,
        xytext=(+15,0),
        textcoords="offset points",
        fontsize=12,
        fontweight='bold',
        color=“green“)
```



Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

- Usando o **annotate** para colocar as legendas na frente dos dados do gráfico das retas:

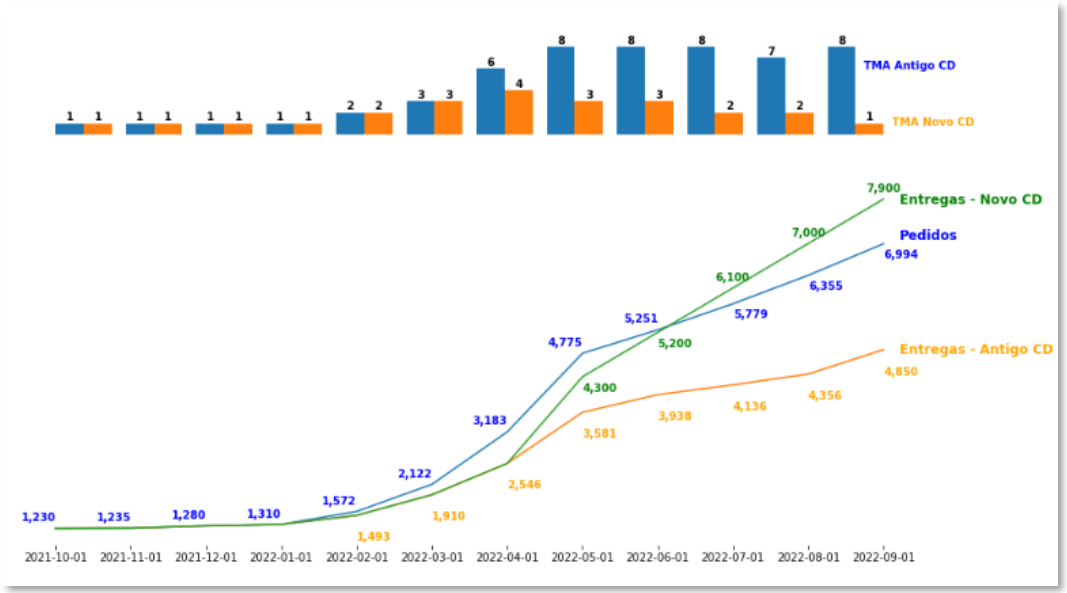
```
ax[1].annotate("Entregas - Antigo CD",
        (i,base. Entregas[11]),
        ha="left",
       va="center”,
       xytext=(+15,0),
       textcoords="offset points",
       fontsize=12,
       fontweight='bold',
       color=“orange")
```



Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

- Usando o **annotate** para colocar as legendas na frente dos dados do gráfico das retas:

```
ax[1].annotate("Pedidos",
        (i,base. Pedidos[11]),
        ha="left",
        va="center",
        xytext=(+15,+8),
        textcoords="offset points",
        fontsize=12,
        fontweight='bold',
        color="blue")
```



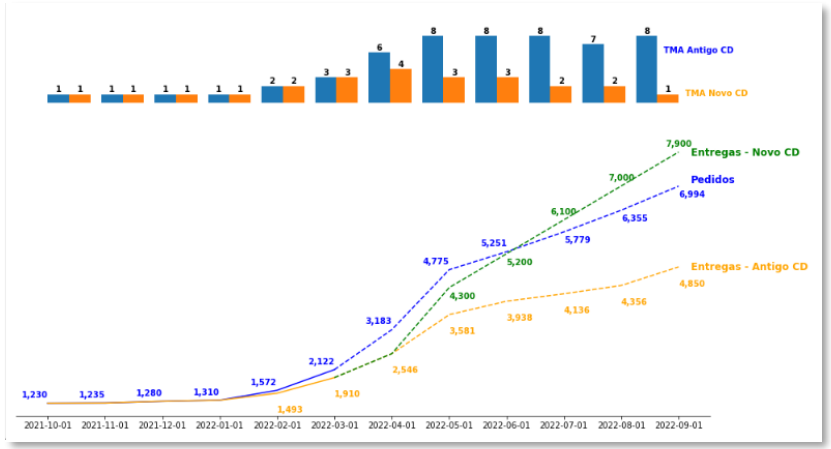
Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

- Por fim, podemos separar entre realizado e projetado:

```
ax[1].plot(np.arange(len(base))[:6],base.Pedidos[:6],color="blue")
ax[1].plot(np.arange(len(base))[-7:],base.Pedidos[-7:], "--",color="blue")
ax[1].plot(np.arange(len(base))[:6],base.Entregas[:6],color="orange")
ax[1].plot(np.arange(len(base))[-7:],base.Entregas[-7:], "--",color="orange")
ax[1].plot(np.arange(len(base))[-7:],base.EntregaNovoCD[-7:], "--",color="green")
```

Diferenciar o projetado colocando o tracejado “—”

Separando em 2 gráficos pelo comprimento da base



Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

- Por fim, podemos separar entre realizado e projetado:

```
ax[0].bar(np.arange(len(base))[:7]-wid/2,
          base.TMAantigoCD[:7],
          width=wid,
          label="TMA Antigo CD",
          color="orange")

ax[0].bar(np.arange(len(base))[-6:]-wid/2,
          base.TMAantigoCD[-6:],
          width=wid,
          label="TMA Antigo CD",
          color="#FFB973")
```

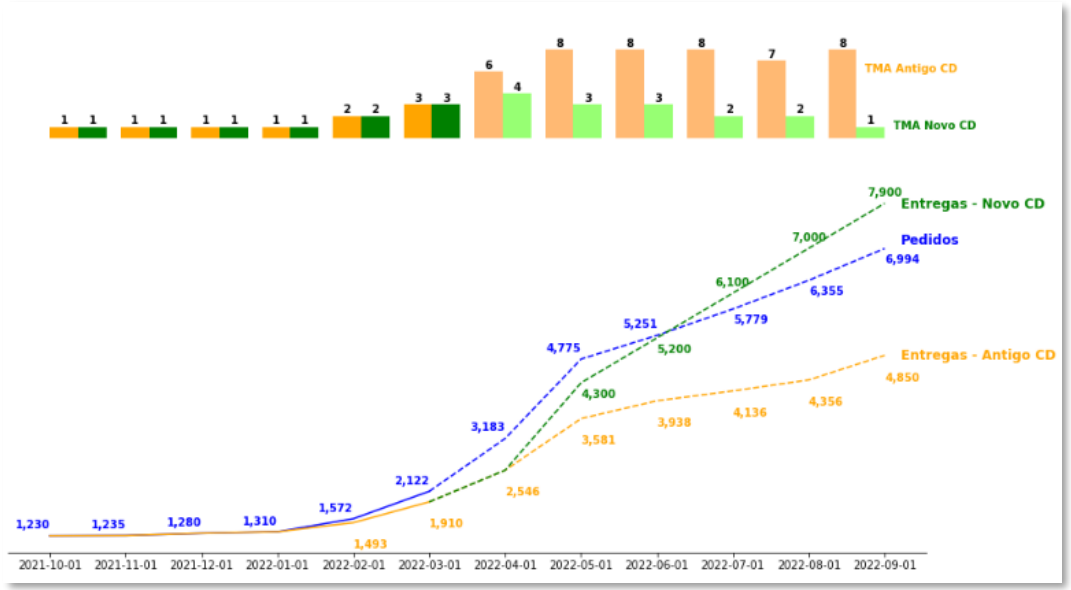
Definindo divisão do que é projetado

Definindo nova cor do projetado

Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

- Por fim, podemos separar entre realizado e projetado:

```
ax[0].bar(np.arange(len(base))[:7]+wid/2,
          base.TMAnovoCD[:7],
          width=wid,
          label="TMA Novo CD",  color="green")

ax[0].bar(np.arange(len(base))[-6:]+wid/2,
          base.TMAnovoCD[-6:],
          width=wid,
          label="TMA Novo CD",
          color="#97FF73")
```



Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

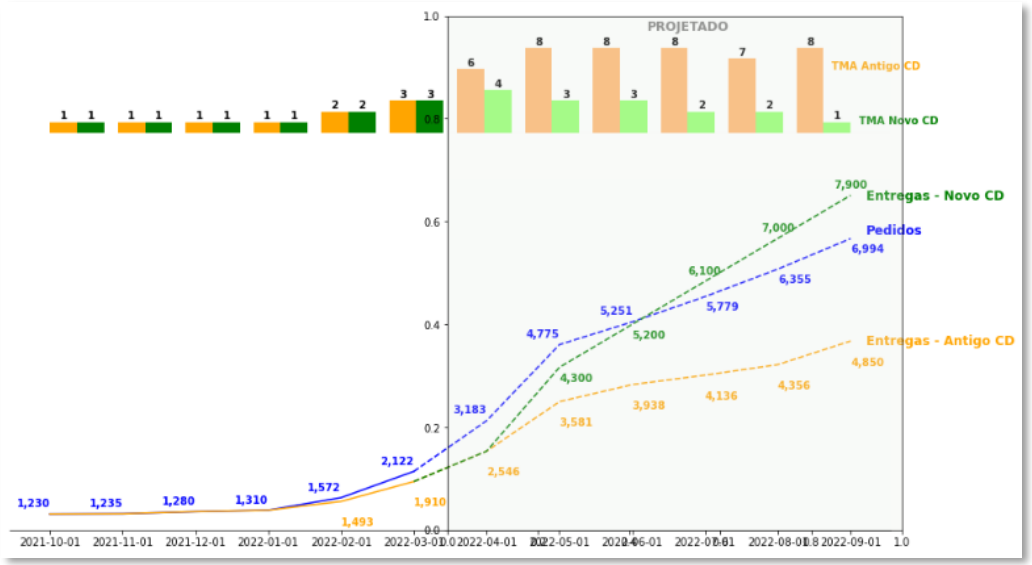
- Por fim, podemos destacar a área do projetado com o retângulo:

Ax1 = **fig.add_axes**([0.51,0.125,0.4,0.755])
rect = ax1.patch

rect.set_edgecolor(None)

rect.set_facecolor('#E0E6DE')

rect.set_alpha(0.2)



ax[0].**annotate**("PROJETADO",(9,10),**ha**="center",**va**="center",**xytext**=(0,0),
textcoords="offset points", **fontsize**=12, **fontweight**='bold',**color**="gray")

Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

- Por fim, podemos retirar as bordas e os valores dos eixos:

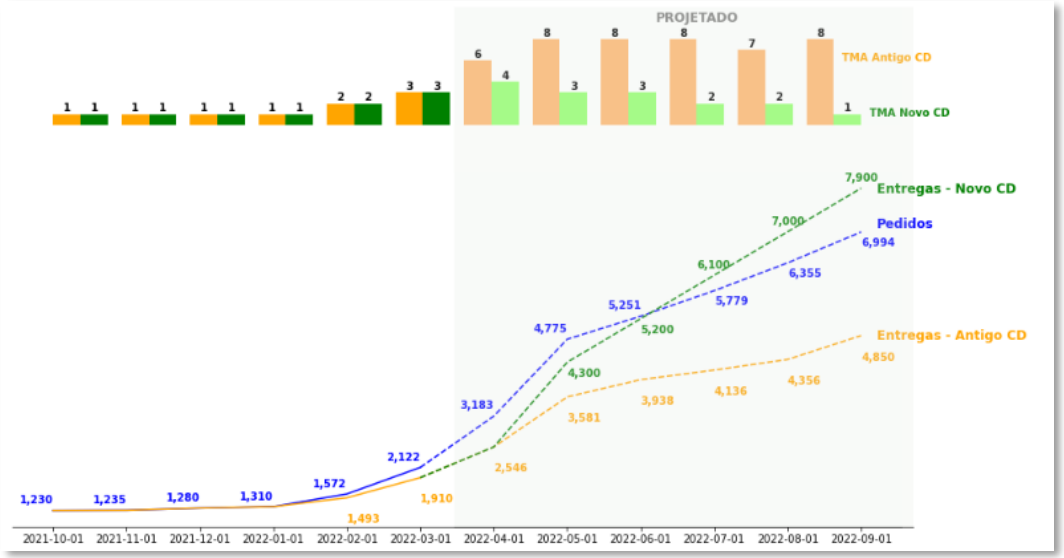
ax1.**xaxis.set_visible**(False)

ax1.**yaxis.set_visible**(False)

ax1.**spines**['top'].**set_visible**(False)

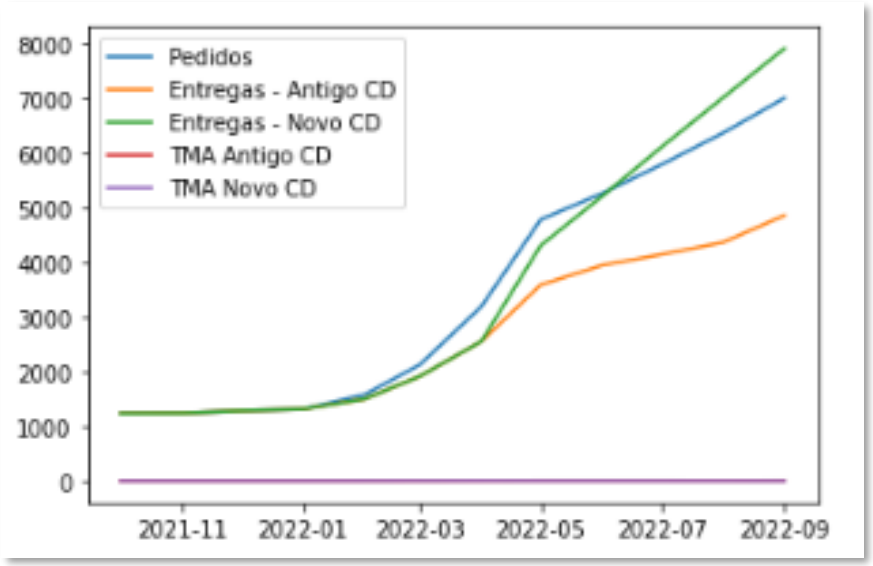
ax1.**spines**['left'].**set_visible**(False)

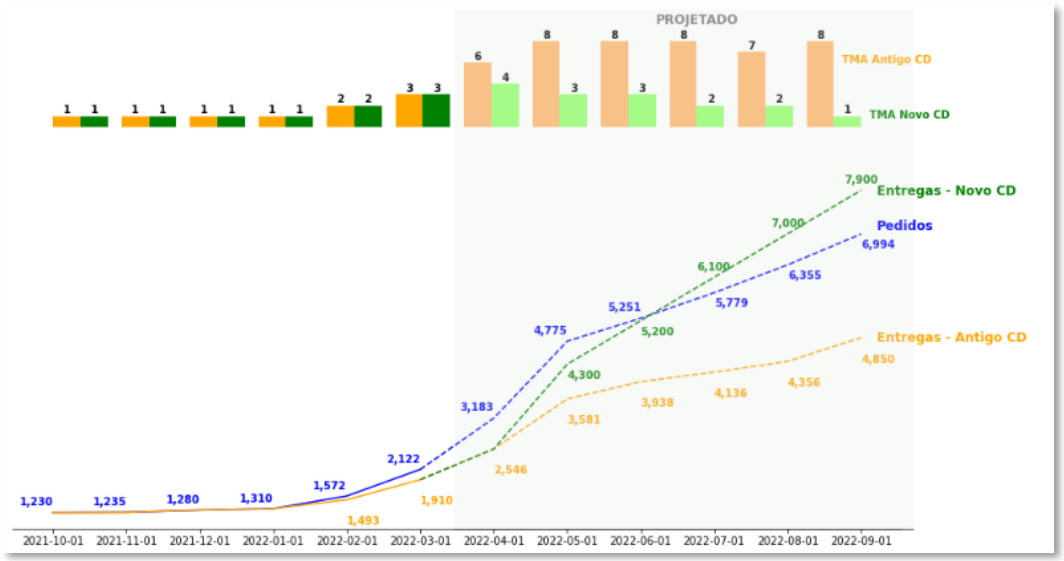
ax1.**spines**['right'].**set_visible**(False)



Módulo 8 – Boas práticas de visualização no Python: Melhorando o visual do gráfico de linhas e separando realizado x projetado

Observem a diferença do primeiro gráfico da aula para o último.





Mais fácil de visualizar os dados

MÓDULO 9

Criando uma apresentação executiva

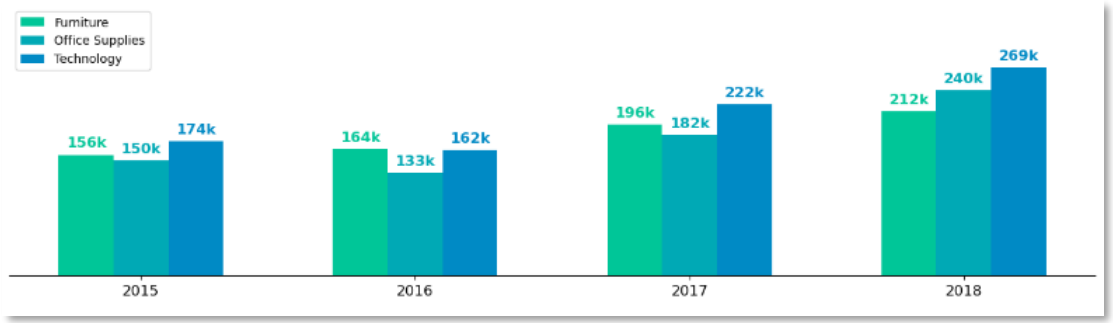
Agora que já vimos como usar o Matplolib e as boas práticas de apresentação, já podemos fazer uma apresentação executiva, que seja realmente **bonita**, **visual** e que traga valor para a nossa empresa.

Nesse projeto vamos fazer as discussões sobre os dados tratados.

Começando o nosso projeto, nós vamos utilizar a base do kaggle de vendas.

O que queremos responder nesse projeto?

- Como foi a **venda** nesse período?
- Qual foi a **categoria mais vendida**?
- Qual foi o **item mais vendido**?



O primeiro passo para começar o nosso projeto é importar as bibliotecas com os código abaixo:

import pandas **as** pd

import numpy **as** np

Apelido da biblioteca

```
# Importando as bibliotecas e a base
import pandas as pd
import numpy as np
base = pd.read_csv("Criando uma apresentação executiva.csv")
```

Para começar o nosso projeto vamos importar a base de dados.

Estrutura:

base = **pd**.**read**_ tipodoarquivo(“**Nome do arquivo**.**tipodoarquivo**”)

base = **pd**.**read**_csv(“**Criando uma apresentação executiva**.**csv**”

```
# Importando as bibliotecas e a base
import pandas as pd
import numpy as np
base = pd.read_csv("Criando uma apresentação executiva.csv")
```

Módulo 9 – Importando e analisando a base

Uma forma de verificar se a base foi importada é utilizando o começo **.head** para visualizar as 5 primeiras linhas.

Estrutura:

base.**head**()

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category
0	1	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-BO-10001798	Furniture
1	2	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-CH-10000454	Furniture
2	3	CA-2017-138688	12/06/2017	16/06/2017	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	California	90036.0	West	OFF-LA-10000240	Office Supplies
3	4	US-2016-108966	11/10/2016	18/10/2016	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311.0	South	FUR-TA-10000577	Furniture
4	5	US-2016-108966	11/10/2016	18/10/2016	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311.0	South	OFF-ST-10000760	Office Supplies

Podemos também utilizar o **.tail** para visualizar as 5 últimas linhas.

Estrutura:

base.**tail**()

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID
9795	9796	CA-2017-125920	21/05/2017	28/05/2017	Standard Class	SH-19975	Sally Hughsby	Corporate	United States	Chicago	Illinois	60610.0	Central	OFF-BI-10003429
9796	9797	CA-2016-128608	12/01/2016	17/01/2016	Standard Class	CS-12490	Cindy Schnelling	Corporate	United States	Toledo	Ohio	43615.0	East	OFF-AR-10001374
9797	9798	CA-2016-128608	12/01/2016	17/01/2016	Standard Class	CS-12490	Cindy Schnelling	Corporate	United States	Toledo	Ohio	43615.0	East	TEC-PH-10004977
9798	9799	CA-2016-128608	12/01/2016	17/01/2016	Standard Class	CS-12490	Cindy Schnelling	Corporate	United States	Toledo	Ohio	43615.0	East	TEC-PH-10000912
9799	9800	CA-2016-128608	12/01/2016	17/01/2016	Standard Class	CS-12490	Cindy Schnelling	Corporate	United States	Toledo	Ohio	43615.0	East	TEC-AC-10000487

Módulo 9 – Importando e analisando a base

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category
0	1	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-BO-10001798	Furniture
1	2	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-CH-10000454	Furniture
2	3	CA-2017-138688	12/06/2017	16/06/2017	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	California	90036.0	West	OFF-LA-10000240	Office Supplies
3	4	US-2016-108966	11/10/2016	18/10/2016	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311.0	South	FUR-TA-10000577	Furniture
4	5	US-2016-108966	11/10/2016	18/10/2016	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311.0	South	OFF-ST-10000760	Office Supplies
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9795	9796	CA-2017-125920	21/05/2017	28/05/2017	Standard Class	SH-19975	Sally Hughsby	Corporate	United States	Chicago	Illinois	60610.0	Central	OFF-BI-10003429	Office Supplies
9796	9797	CA-2016-128608	12/01/2016	17/01/2016	Standard Class	CS-12490	Cindy Schnelling	Corporate	United States	Toledo	Ohio	43615.0	East	OFF-AR-10001374	Office Supplies
9797	9798	CA-2016-128608	12/01/2016	17/01/2016	Standard Class	CS-12490	Cindy Schnelling	Corporate	United States	Toledo	Ohio	43615.0	East	TEC-PH-10004977	Technology
9798	9799	CA-2016-128608	12/01/2016	17/01/2016	Standard Class	CS-12490	Cindy Schnelling	Corporate	United States	Toledo	Ohio	43615.0	East	TEC-PH-10000912	Technology
9799	9800	CA-2016-128608	12/01/2016	17/01/2016	Standard Class	CS-12490	Cindy Schnelling	Corporate	United States	Toledo	Ohio	43615.0	East	TEC-AC-10000487	Technology

9800 rows × 18 columns

Número de linhas e colunas

Além disso, podemos usar o comando **display** para visualizar o formato da base, as 5 primeiras e últimas linhas

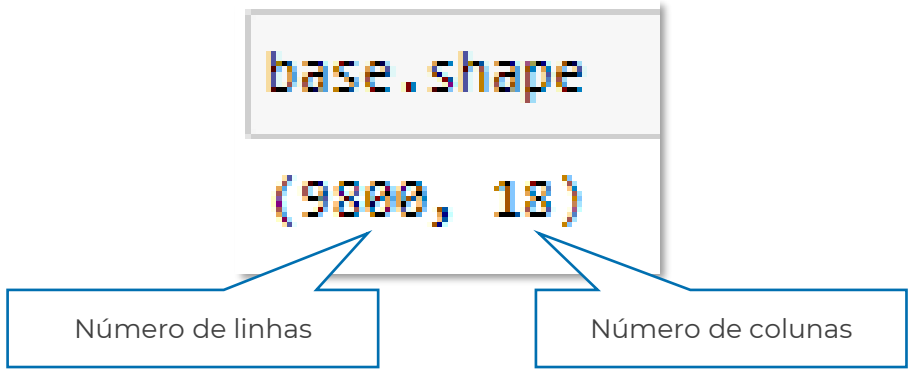
Estrutura:

display(**base**)

Outra forma de visualizar o formato da base é utilizando o comando **shape**.

Estrutura:

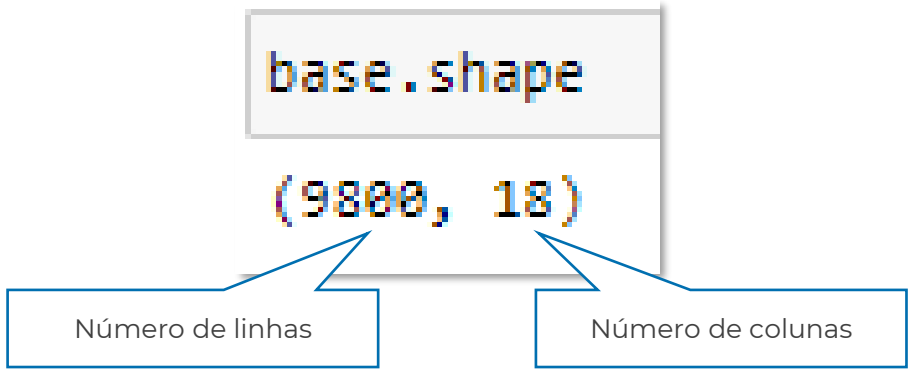
base.**shape**()



Outra forma de visualizar o formato da base é utilizando o comando **shape**.

Estrutura:

base.**shape**()



Vamos verificar se tem alguma informação faltando, algum dado no formato errado ou uma informação nula.

Estrutura:

base.info()

Vamos ajustar as informações de data que não estão com o formato de data.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9800 entries, 0 to 9799
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Row ID         9800 non-null   int64
 1   Order ID       9800 non-null   object
 2   Order Date     9800 non-null   object
 3   Ship Date      9800 non-null   object
 4   Ship Mode      9800 non-null   object
 5   Customer ID    9800 non-null   object
 6   Customer Name  9800 non-null   object
 7   Segment        9800 non-null   object
 8   Country        9800 non-null   object
 9   City           9800 non-null   object
 10  State          9800 non-null   object
 11  Postal Code    9789 non-null   float64
 12  Region         9800 non-null   object
 13  Product ID     9800 non-null   object
 14  Category       9800 non-null   object
 15  Sub-Category   9800 non-null   object
 16  Product Name   9800 non-null   object
 17  Sales          9800 non-null   float64
dtypes: float64(2), int64(1), object(15)
memory usage: 1.3+ MB
```

Order date e ship date estão como objeto

Observe que temos 1 valor nulo

Módulo 9 – Importando e analisando a base

Apenas código postal possui valores nulos, podemos visualizar quais linhas não possuem esse valor.

Verificando os registros com Postal Code vazio com o código abaixo:

```
base[base['Postal Code'].isnull()]
```

Filtrando a base pela coluna Postal Code

Todos os registros sem código postal são da cidade de Burlington, em Vermont.

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category
2234	2235	CA-2018-104066	05/12/2018	10/12/2018	Standard Class	QJ-19255	Quincy Jones	Corporate	United States	Burlington	Vermont	NaN	East	TEC-AC-10001013	Technology
5274	5275	CA-2016-162887	07/11/2016	09/11/2016	Second Class	SV-20785	Stewart Visinsky	Consumer	United States	Burlington	Vermont	NaN	East	FUR-CH-10000595	Furniture
8798	8799	US-2017-150140	06/04/2017	10/04/2017	Standard Class	VM-21685	Valerie Mitchum	Home Office	United States	Burlington	Vermont	NaN	East	TEC-PH-10002555	Technology
9146	9147	US-2017-165505	23/01/2017	27/01/2017	Standard Class	CB-12535	Claudia Bergmann	Corporate	United States	Burlington	Vermont	NaN	East	TEC-AC-10002926	Technology
9147	9148	US-2017-165505	23/01/2017	27/01/2017	Standard Class	CB-12535	Claudia Bergmann	Corporate	United States	Burlington	Vermont	NaN	East	OFF-AR-10003477	Office Supplies
9148	9149	US-2017-165505	23/01/2017	27/01/2017	Standard Class	CB-12535	Claudia Bergmann	Corporate	United States	Burlington	Vermont	NaN	East	OFF-ST-10001526	Office Supplies
9386	9387	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States	Burlington	Vermont	NaN	East	OFF-PA-10000157	Office Supplies
9387	9388	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States	Burlington	Vermont	NaN	East	OFF-PA-10001970	Office Supplies
9388	9389	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States	Burlington	Vermont	NaN	East	OFF-AP-10000828	Office Supplies
9389	9390	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States	Burlington	Vermont	NaN	East	OFF-EN-10001509	Office Supplies
9741	9742	CA-2016-117086	08/11/2016	12/11/2016	Standard Class	QJ-19255	Quincy Jones	Corporate	United States	Burlington	Vermont	NaN	East	FUR-BO-10004834	Furniture

Valor nulo quando City é Burlington

Módulo 9 – Importando e analisando a base

Vamos verificar se existe algum registro de Burlington, Vermont com Postal Code.

```
base[(base.City == 'Burlington')]
```

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category
683	684	US-2018-168116	04/11/2018	04/11/2018	Same Day	GT-14635	Grant Thornton	Corporate	United States	Burlington	North Carolina	27217.0	South	TEC-MA-10004125	Technology
684	685	US-2018-168116	04/11/2018	04/11/2018	Same Day	GT-14635	Grant Thornton	Corporate	United States	Burlington	North Carolina	27217.0	South	OFF-AP-10002457	Office Supplies
1008	1009	US-2018-106705	26/12/2018	01/01/2019	Standard Class	PO-18850	Patrick O'Brill	Consumer	United States	Burlington	Iowa	52601.0	Central	OFF-PA-10001509	Office Supplies
1038	1039	CA-2018-121818	20/11/2018	21/11/2018	First Class	JH-15430	Jennifer Halladay	Consumer	United States	Burlington	North Carolina	27217.0	South	OFF-AR-10000203	Office Supplies
1039	1040	CA-2018-121818	20/11/2018	21/11/2018	First Class	JH-15430	Jennifer Halladay	Consumer	United States	Burlington	North Carolina	27217.0	South	OFF-AR-10004790	Office Supplies
1393	1394	CA-2018-124828	03/07/2018	04/07/2018	First Class	YS-21880	Yana Sorensen	Corporate	United States	Burlington	North Carolina	27217.0	South	OFF-AR-10003514	Office Supplies
2234	2235	CA-2018-104066	05/12/2018	10/12/2018	Standard Class	QJ-19255	Quincy Jones	Corporate	United States	Burlington	Vermont	5401.0	East	TEC-AC-10001013	Technology
2928	2929	US-2018-120390	19/10/2018	26/10/2018	Standard Class	TH-21550	Tracy Hopkins	Home Office	United States	Burlington	North Carolina	27217.0	South	OFF-BI-10004995	Office Supplies
5065	5066	CA-2018-142090	30/11/2018	07/12/2018	Standard Class	SC-20380	Shahid Collister	Consumer	United States	Burlington	North Carolina	27217.0	South	TEC-AC-10002001	Technology
5066	5067	CA-2018-142090	30/11/2018	07/12/2018	Standard Class	SC-20380	Shahid Collister	Consumer	United States	Burlington	North Carolina	27217.0	South	FUR-TA-10001889	Furniture

Vamos verificar se existe algum registro de Burlington, Vermont com Postal Code.

```
base[(base.City == 'Burlington') & (base.State == 'Vermont') & (base['Postal Code'].notnull())]:
```

Concatenando dois filtros, se fosse OU utilizaríamos o sinal |

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category	Sub-Category	Product Name	Sales

Módulo 9 – Tratando valores vazios

Agora que já identificamos que a nossa base possui valores vazios, nós vamos tratar eles.

Então, primeiro vamos observar os valores que são nulos:

```
base[(base.City == 'Burlington') & (base.State == 'Vermont') & (base['Postal Code'].isnull())]
```

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category
2234	2235	CA-2018-104066	05/12/2018	10/12/2018	Standard Class	QJ-19255	Quincy Jones	Corporate	United States	Burlington	Vermont	NaN	East	TEC-AC-10001013	Technology
5274	5275	CA-2016-162887	07/11/2016	09/11/2016	Second Class	SV-20785	Stewart Visinsky	Consumer	United States	Burlington	Vermont	NaN	East	FUR-CH-10000595	Furniture
8798	8799	US-2017-150140	06/04/2017	10/04/2017	Standard Class	VM-21685	Valerie Mitchum	Home Office	United States	Burlington	Vermont	NaN	East	TEC-PH-10002565	Technology
9146	9147	US-2017-165505	23/01/2017	27/01/2017	Standard Class	CB-12535	Claudia Bergmann	Corporate	United States	Burlington	Vermont	NaN	East	TEC-AC-10002926	Technology
9147	9148	US-2017-165505	23/01/2017	27/01/2017	Standard Class	CB-12535	Claudia Bergmann	Corporate	United States	Burlington	Vermont	NaN	East	OFF-AR-10003477	Office Supplies
9148	9149	US-2017-165505	23/01/2017	27/01/2017	Standard Class	CB-12535	Claudia Bergmann	Corporate	United States	Burlington	Vermont	NaN	East	OFF-ST-10001526	Office Supplies
9386	9387	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States	Burlington	Vermont	NaN	East	OFF-PA-10000157	Office Supplies
9387	9388	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States	Burlington	Vermont	NaN	East	OFF-PA-10001970	Office Supplies
9388	9389	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States	Burlington	Vermont	NaN	East	OFF-AP-10000828	Office Supplies
9389	9390	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States	Burlington	Vermont	NaN	East	OFF-EN-10001509	Office Supplies
9741	9742	CA-2016-117086	08/11/2016	12/11/2016	Standard Class	QJ-19255	Quincy Jones	Corporate	United States	Burlington	Vermont	NaN	East	FUR-BO-10004834	Furniture

Existe um forma que podemos buscar só pelo Postal Code.

```
base[(base.City == 'Burlington') & (base.State == 'Vermont') & (base['Postal Code'].isnull())] ['Postal Code']
```

```
2234   NaN
5274   NaN
8798   NaN
9146   NaN
9147   NaN
9148   NaN
9386   NaN
9387   NaN
9388   NaN
9389   NaN
9741   NaN
Name: Postal Code, dtype: float64
```

Módulo 9 – Tratando valores vazios

Então vamos utilizar o **loc** para localizar esses valores. = 5401

base.loc[(**base**.City == 'Burlington') **&** (**base**.State == 'Vermont') **&** (**base**['Postal Code'].**isnull**())]

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category
2234	2235	CA-2018-104066	05/12/2018	10/12/2018	Standard Class	QJ-19255	Quincy Jones	Corporate	United States	Burlington	Vermont	NaN	East	TEC-AC-10001013	Technology
5274	5275	CA-2016-162887	07/11/2016	09/11/2016	Second Class	SV-20785	Stewart Visinsky	Consumer	United States	Burlington	Vermont	NaN	East	FUR-CH-10000595	Furniture
8798	8799	US-2017-150140	06/04/2017	10/04/2017	Standard Class	VM-21685	Valerie Mitchum	Home Office	United States	Burlington	Vermont	NaN	East	TEC-PH-10002555	Technology
9146	9147	US-2017-165505	23/01/2017	27/01/2017	Standard Class	CB-12535	Claudia Bergmann	Corporate	United States	Burlington	Vermont	NaN	East	TEC-AC-10002926	Technology
9147	9148	US-2017-165505	23/01/2017	27/01/2017	Standard Class	CB-12535	Claudia Bergmann	Corporate	United States	Burlington	Vermont	NaN	East	OFF-AR-10003477	Office Supplies
9148	9149	US-2017-165505	23/01/2017	27/01/2017	Standard Class	CB-12535	Claudia Bergmann	Corporate	United States	Burlington	Vermont	NaN	East	OFF-ST-10001526	Office Supplies
9386	9387	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States	Burlington	Vermont	NaN	East	OFF-PA-10000157	Office Supplies
9387	9388	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States	Burlington	Vermont	NaN	East	OFF-PA-10001970	Office Supplies
9388	9389	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States	Burlington	Vermont	NaN	East	OFF-AP-10000828	Office Supplies
9389	9390	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States	Burlington	Vermont	NaN	East	OFF-EN-10001509	Office Supplies
9741	9742	CA-2016-117086	08/11/2016	12/11/2016	Standard Class	QJ-19255	Quincy Jones	Corporate	United States	Burlington	Vermont	NaN	East	FUR-BO-10004834	Furniture

Módulo 9 – Tratando valores vazios

Agora se quisermos apenas da Coluna Postal Code basta digitar:

```
base.loc[(base.City == 'Burlington') & (base.State == 'Vermont') & (base['Postal Code'].isnull()), ['Postal Code']
```

```
2234   NaN
5274   NaN
8798   NaN
9146   NaN
9147   NaN
9148   NaN
9386   NaN
9387   NaN
9388   NaN
9389   NaN
9741   NaN
Name: Postal Code, dtype: float64
```

Módulo 9 – Tratando valores vazios

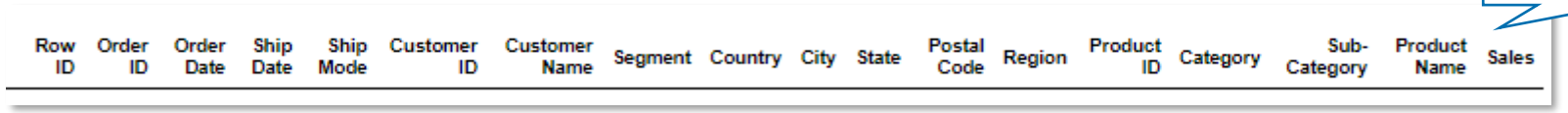
Vamos atribuir o código postal de 05401 para essa coluna:

```
base.loc[(base.City == 'Burlington') & (base.State == 'Vermont') & (base['Postal Code'].isnull()), ['Postal Code']= 05401
```

Vamos procurar esses valores para verificar se foi atualizado:

```
base.loc[(base.City == 'Burlington') & (base.State == 'Vermont') & (base['Postal Code'].isnull())]
```

Nenhum valor em branco



Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category	Sub-Category	Product Name	Sales

Vamos utiliza o comando .info para verificar essa base:

base.info

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9800 entries, 0 to 9799
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Row ID         9800 non-null   int64
 1   Order ID       9800 non-null   object
 2   Order Date     9800 non-null   object
 3   Ship Date      9800 non-null   object
 4   Ship Mode      9800 non-null   object
 5   Customer ID    9800 non-null   object
 6   Customer Name  9800 non-null   object
 7   Segment        9800 non-null   object
 8   Country        9800 non-null   object
 9   City           9800 non-null   object
 10  State          9800 non-null   object
 11  Postal Code    9800 non-null   float64
 12  Region         9800 non-null   object
 13  Product ID     9800 non-null   object
 14  Category       9800 non-null   object
 15  Sub-Category   9800 non-null   object
 16  Product Name   9800 non-null   object
 17  Sales          9800 non-null   float64
dtypes: float64(2), int64(1), object(15)
memory usage: 1.3+ MB
```

Nenhum valor nulo

Módulo 9 – Tratando valores vazios

Se fosse necessário excluir a coluna, poderíamos utilizar o comando **drop**

base = base.drop('**Postal Code**', axis = 1)

base.head()

Porém não vamos utilizar isso no nosso projeto

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category
0	1	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-BO-10001798	Furniture
1	2	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-CH-10000454	Furniture
2	3	CA-2017-138688	12/06/2017	16/06/2017	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	California	90036.0	West	OFF-LA-10000240	Office Supplies
3	4	US-2016-108966	11/10/2016	18/10/2016	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311.0	South	FUR-TA-10000577	Furniture
4	5	US-2016-108966	11/10/2016	18/10/2016	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311.0	South	OFF-ST-10000760	Office Supplies

Agora vamos para etapa de visualização. Aqui temos 2 questões que precisamos pensar:

Quais **colunas** vamos usar?

- Podemos verificar a venda na coluna "**Sales**“
- O período nós podemos usar a coluna "**Order Date**", mas precisamos entender melhor essa coluna

Qual o período queremos exibir?

Observe que Order Date está como objeto

Atenção: Se a coluna de data estiver como texto e não como data, a ordenação vai ser pelo texto e não pela data!

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9800 entries, 0 to 9799
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Row ID         9800 non-null   int64
 1   Order ID       9800 non-null   object
 2   Order Date     9800 non-null   object
 3   Ship Date      9800 non-null   object
 4   Ship Mode      9800 non-null   object
 5   Customer ID    9800 non-null   object
 6   Customer Name  9800 non-null   object
 7   Segment        9800 non-null   object
 8   Country        9800 non-null   object
 9   City           9800 non-null   object
 10  State          9800 non-null   object
 11  Postal Code    9800 non-null   float64
 12  Region         9800 non-null   object
 13  Product ID     9800 non-null   object
 14  Category       9800 non-null   object
 15  Sub-Category   9800 non-null   object
 16  Product Name   9800 non-null   object
 17  Sales          9800 non-null   float64
dtypes: float64(2), int64(1), object(15)
memory usage: 1.3+ MB
```

Order date está como objeto

Podemos converter uma coluna para data usando o **to_datetime** do pandas

Estrutura:

base["**coluna**"] = **pd.to_datetime**(**base**["**coluna**"])

No **datetime**, caso tenha apenas a data, a hora ficaria como 00:00:00

Se quisermos considerar apenas a data, podemos usar o **.date** da biblioteca datetime.

Vamos aplicar no nosso projeto

```
Import datetime as dt

base["Order Date"] = pd.to_datetime(base["Order Date"])

base["Ship Date"] = pd.to_datetime(base["Ship Date"])

base.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9800 entries, 0 to 9799
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Row ID         9800 non-null   int64
 1   Order ID       9800 non-null   object
 2   Order Date     9800 non-null   datetime64[ns]
 3   Ship Date      9800 non-null   datetime64[ns]
 4   Ship Mode      9800 non-null   object
 5   Customer ID    9800 non-null   object
 6   Customer Name  9800 non-null   object
 7   Segment        9800 non-null   object
 8   Country        9800 non-null   object
 9   City           9800 non-null   object
 10  State          9800 non-null   object
 11  Postal Code    9800 non-null   float64
 12  Region         9800 non-null   object
 13  Product ID     9800 non-null   object
 14  Category       9800 non-null   object
 15  Sub-Category   9800 non-null   object
 16  Product Name   9800 non-null   object
 17  Sales          9800 non-null   float64
dtypes: datetime64[ns](2), float64(2), int64(1), object(13)
memory usage: 1.3+ MB
```

Formato de data

Agora podemos verificar qual a data mínima e a máxima:

base[**Order Date**'].**min**()

```
base['Order Date'].min()
```

```
'01/01/2018'
```

base[**Order Date**'].**max**()

```
base['Order Date'].max()
```

```
'31/12/2017'
```

Como estamos falando de 4 anos de análise, começando em jan/2015 e indo até dez/2018.

Estamos trabalhando com um período grande de anos, podemos começar a visualização pelos anos.

Para isso, podemos usar o .**year** da biblioteca datetime para criar uma nova coluna apenas com o ano dessa base

Módulo 9 – Usando o datetime para tratar datas

Podemos criar uma coluna com o ano:

base['Ano'] = base['Order Date'].dt.year

Biblioteca date.time

Para visualizar:

base.head()

Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category	Sub-Category	Product Name	Sales	Ano
2017-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-BO-10001798	Furniture	Bookcases	Bush Somerset Collection Bookcase	261.9600	2017
2017-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-CH-10000454	Furniture	Chairs	Hon Deluxe Fabric Upholstered Stacking Chairs,...	731.9400	2017
2017-06-16	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	California	90036.0	West	OFF-LA-10000240	Office Supplies	Labels	Self-Adhesive Address Labels for Typewriters b...	14.6200	2017
2016-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311.0	South	FUR-TA-10000577	Furniture	Tables	Bretford CR4500 Series Slim Rectangular Table	957.5775	2016
2016-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311.0	South	OFF-ST-10000760	Office Supplies	Storage	Eldon Fold 'N Roll Cart System	22.3680	2016

Vamos gerar um gráfico de quando a compra foi feita, mas para isso precisamos agregar as informações utilizando o **group by**.

base.groupby('Ano')['Sales'].sum()

Qual coluna vamos usar para fazer a função de agregação

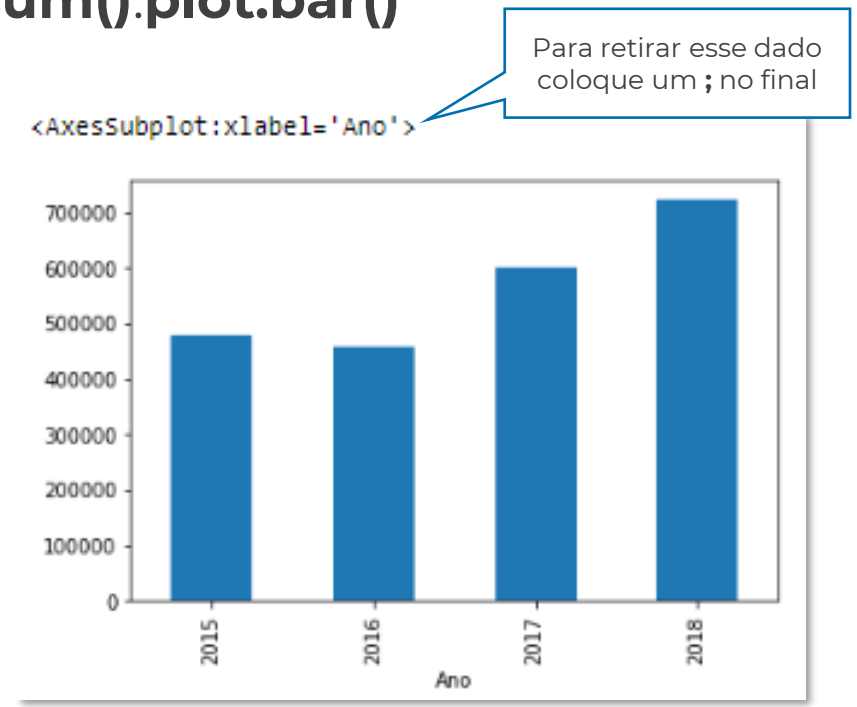
Função de agregação

```
Ano
2015    479856.2081
2016    459436.0054
2017    600192.5500
2018    722052.0192
Name: Sales, dtype: float64
```

Então o resultado é a soma das vendas por cada ano.

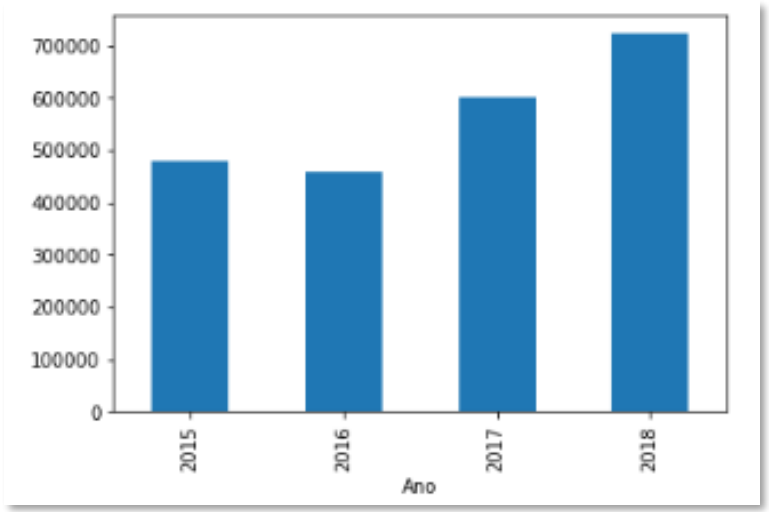
Agora vamos transformar o dado anterior em um gráfico.

base.groupby('Ano')['Sales'].sum().plot.bar()



Agora vamos retirar aquela informação antes do gráfico.

base.groupby('Ano')['Sales'].sum().plot.bar();



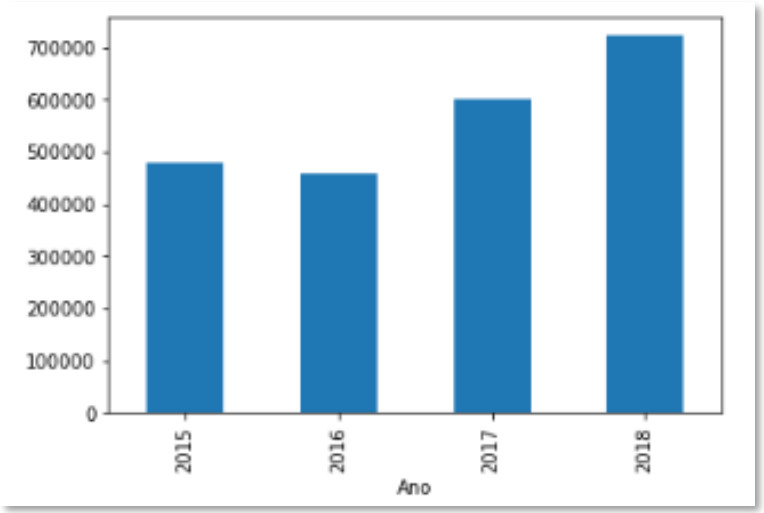
Módulo 9 – Criando um gráfico de barras no matplotlib

A visualização ainda não está tão boa, porque não conseguimos visualizar os valores exatos de cada ano.

Vamos melhorar esse visual utilizando o **matplotlib**, para isso vamos importar as bibliotecas:

import matplotlib.pyplot **as** plt

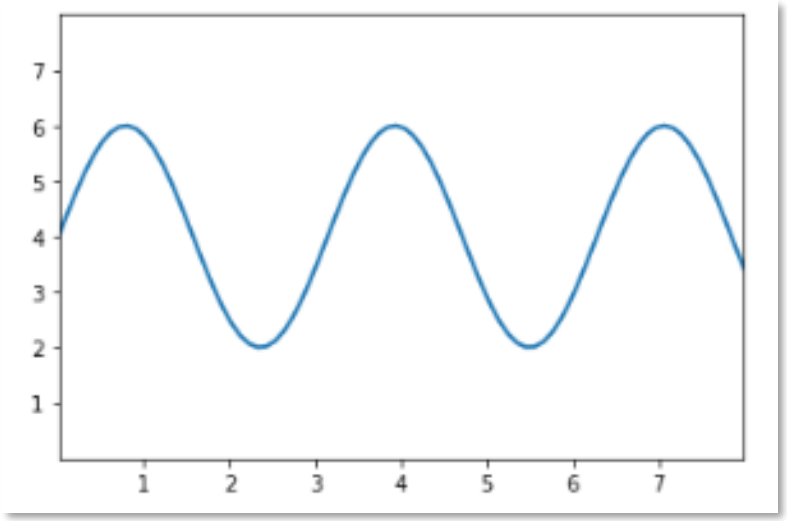
import numpy **as** np



Módulo 9 – Criando um gráfico de barras no matplotlib

Vamos utilizar o código padrão da documentação do matplotlib:

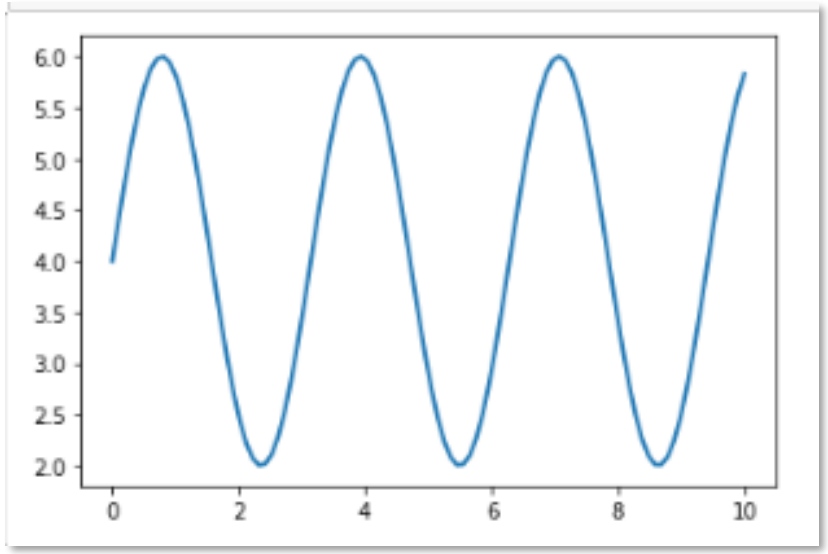
```
x = np.linspace(0, 10, 100)
y = 4 + 2 * np.sin(2 * x)

fig, ax = plt.subplots()

ax.plot(x, y, linewidth=2.0)

ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
       ylim=(0, 8), yticks=np.arange(1, 8))

plt.show()
```



Módulo 9 – Criando um gráfico de barras no matplotlib

Na verdade para realizar um gráfico basta o código abaixo:

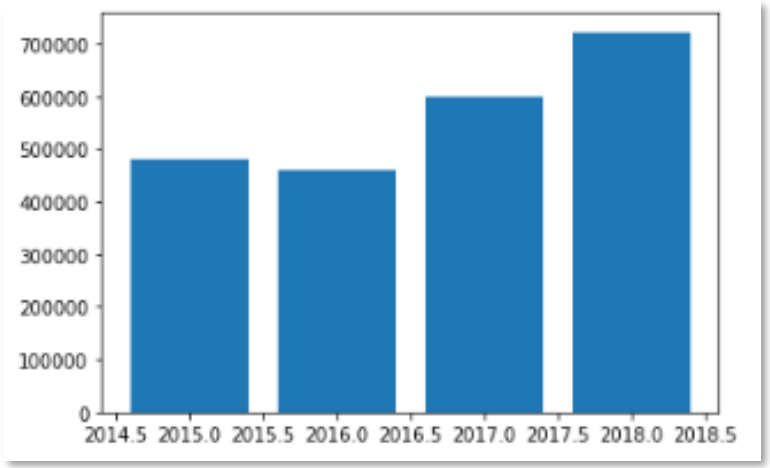
```
fig, ax = plt.subplots()

ax.plot(x, y, linewidth=2.0)

plt.show()
```



Módulo 9 – Criando um gráfico de barras no matplotlib

Agora vamos substituis os nossos valores em x e y:

```
fig, ax = plt.subplots()

ax.bar(base.groupby("Ano")["Sales"].sum().index,base.groupby("Ano")["Sales"].sum().values,linewidth=2.0)

plt.show()
```



Módulo 9 – Criando um gráfico de barras no matplotlib

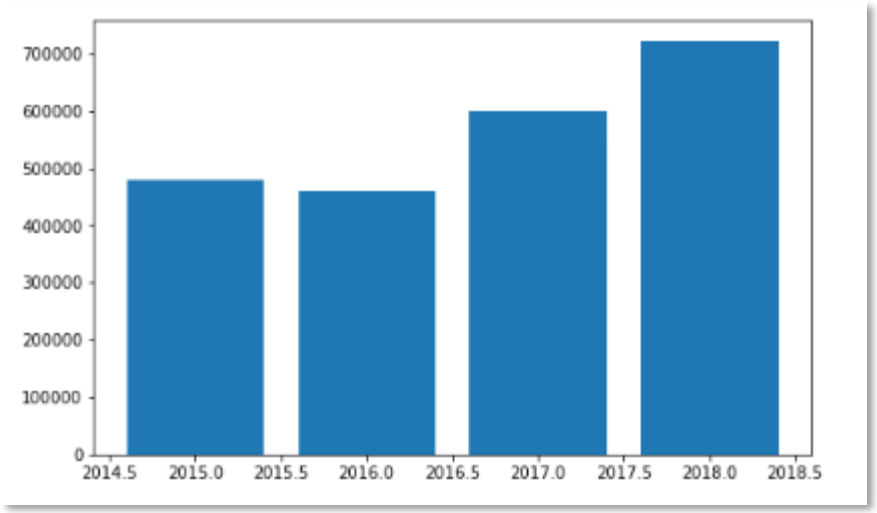
Agora vamos mudar o tamanho da figura:

fig, ax = **plt.subplots**(figsize=(8,5))

Tamanho do gráfico

ax.**bar**(**base**.**groupby**("**Ano**")["**Sales**"].**sum**().**index**,**base**.**groupby**("**Ano**")["**Sales**"].**sum**().**values**, **linewidth**=2.0)

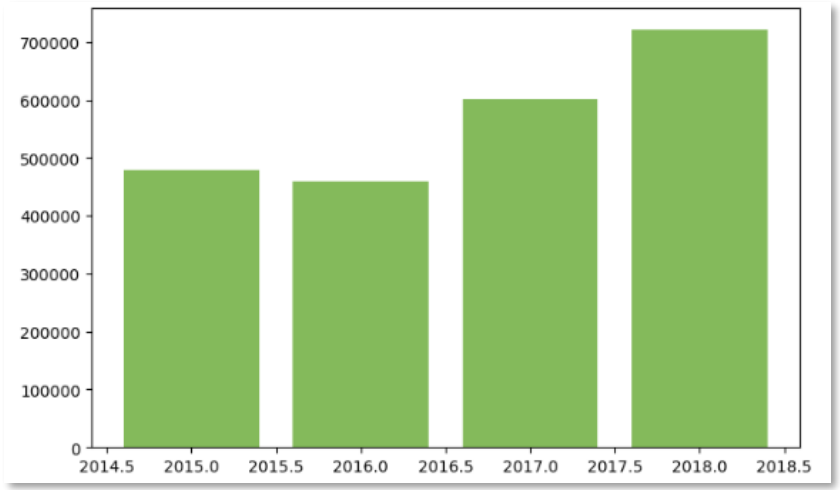
plt.show()



Módulo 9 – Criando um gráfico de barras no matplotlib

Agora vamos alterar a cor do gráfico:

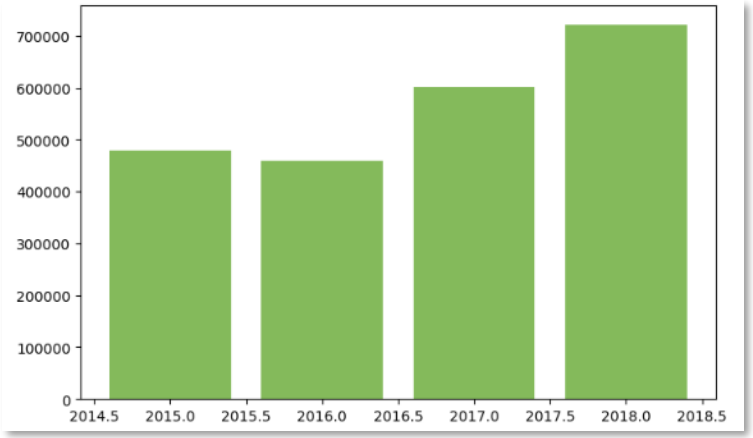
```
fig, ax = plt.subplots(figsize=(8,5))

ax.bar(base.groupby("Ano")["Sales"].sum().index,base.groupby("Ano")["Sales"].sum().values,linewidth=2.0, color='#84ba5b')

plt.show()
```



Módulo 9 – Adicionando título no gráfico e ajustando o eixo x

Para melhorar o nosso código vamos colocar o groupby dentro de uma variável:

```
fig, ax = plt.subplots(figsize=(8,5))

soma_ano = base.groupby("Ano")["Sales"].sum()

ax.bar(soma_ano .index, soma_ano .values,linewidth=2.0, color='#84ba5b')

plt.show()
```



Módulo 9 – Adicionando título no gráfico e ajustando o eixo x

Vamos adicionar o título:

- O **.set_title** vai permitir colocar título e ajustar os parâmetros desse títulos:

  fontsize: Tamanho da fonte

  fontweight: Espessura da fonte

  color: Cor da fonte

Módulo 9 – Adicionando título no gráfico e ajustando o eixo x

Vamos adicionar melhorar o eixo x utilizando os códigos:

- O **xaxis.set_ticks** vai definir os valores mostrados no eixo x

- O **.tick_params**(axis='x') vai ajustar o parâmetro do eixo x

  labelsize: tamanho do rótulo de dados

Módulo 9 – Adicionando título no gráfico e ajustando o eixo x

```
fig, ax = plt.subplots(figsize=(8,5))
soma_ano = base.groupby("Ano")["Sales"].sum()

ax.set_title("VENDAS POR ANO",
             fontsize=20,
             fontweight="bold",
             color="#84ba5b“)
```

Ajustando os parâmetros do título do gráfico

Valores de ano que não existem, ex: 2018,5

```
ax.bar(soma_ano .index, soma_ano .values,linewidth=2.0, color='#84ba5b')

plt.show()
```

Módulo 9 – Adicionando título no gráfico e ajustando o eixo x

Vamos ajustar os valores dos anos:

```
fig, ax = plt.subplots(figsize=(8,5))
soma_ano = base.groupby("Ano")["Sales"].sum()

ax.set_title("VENDAS POR ANO",
             fontsize=20,
             fontweight="bold",
             color="#84ba5b“)

ax.bar(soma_ano .index, soma_ano .values,linewidth=2.0, color='#84ba5b’)

ax.xaxis.set_ticks([2015,2016,2017,2018])
plt.show()
```

Ajustando os valores dos anos

Módulo 9 – Adicionando título no gráfico e ajustando o eixo x

```
fig, ax = plt.subplots(figsize=(8,5))
soma_ano = base.groupby("Ano")["Sales"].sum()

ax.set_title("VENDAS POR ANO",
             fontsize=20,
             fontweight="bold",
             color="#84ba5b“)

ax.bar(soma_ano .index, soma_ano .values,linewidth=2.0, color='#84ba5b’)

ax.xaxis.set_ticks([2015,2016,2017,2018])

ax.tick_params(axis='x',labelsize=14)
plt.show()
```

Ajustando o tamanho do rótulo de dados dos anos

Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Para adicionar rótulo de dados, vamos usar o **.annotate** como vimos nas outras aulas

```
fig, ax = plt.subplots(figsize=(8,5))
soma_ano = base.groupby("Ano")["Sales"].sum()
ax.set_title("VENDAS POR ANO",
        fontsize=20,
        fontweight="bold",
        color="#84ba5b“)

ax.bar(soma_ano .index, soma_ano .values,linewidth=2.0, color='#84ba5b’)
ax.xaxis.set_ticks([2015,2016,2017,2018])
ax.tick_params(axis='x',labelsize=14)
ax.annotate(“TESTE”,(soma_ano.index[0],100000)

plt.show()
```

Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Porém, no caso anterior só tínhamos o valor de 1 ano no rótulo de dados.

Para fazer com todos os valores vamos utilizar o **for**.

Relembrando:

For i **in** range (0,4)**:**
    print (i)

```
for i in range (0,4):
    print (i)
```

```
0
1
2
3
```

Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Agora vamos focar no código do **annotate** utilizando o for:

```
For i in range (0,4):
        ax.annotate('TESTE',(soma_ano.index[i],100000))
```

Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Agora vamos alterar os valores do eixo y no **annotate**:

```
For i in range (0,4):
        ax.annotate('TESTE',(soma_ano.index[i], soma_ano.values[i]))
```

Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Agora vamos alterar o texto TESTE para exatamente o valor de y no **annotate**:

```
For i in range (0,4):
        ax.annotate(soma_ano.values[i],(soma_ano.index[i], soma_ano.values[i]))
```

Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Agora vamos ajustar os valores dos rótulos de dados para que eles fiquem centralizados, para isso vamos utilizar o **ha** = “center”:

```
For i in range (0,4):
        ax.annotate(soma_ano.values[i],(soma_ano.index[i], soma_ano.values[i]),
        ha="center",
        )
```

Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Agora vamos ajustar a posição dos valores dos rótulos de dados, utilizando o **xytext** e o **textcoords**:

```
For i in range (0,4):
        ax.annotate(soma_ano.values[i],(soma_ano.index[i], soma_ano.values[i]),
        ha="center",
        xytext=(0,5),
        textcoords="offset points",
        )
```

Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Agora vamos aumentar o tamanho da fonte com o **fontsize**:

```
For i in range (0,4):
        ax.annotate(soma_ano.values[i],(soma_ano.index[i], soma_ano.values[i]),
        ha="center",
        xytext=(0,5),
        textcoords="offset points",
        fontsize=12,
        )
```

Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

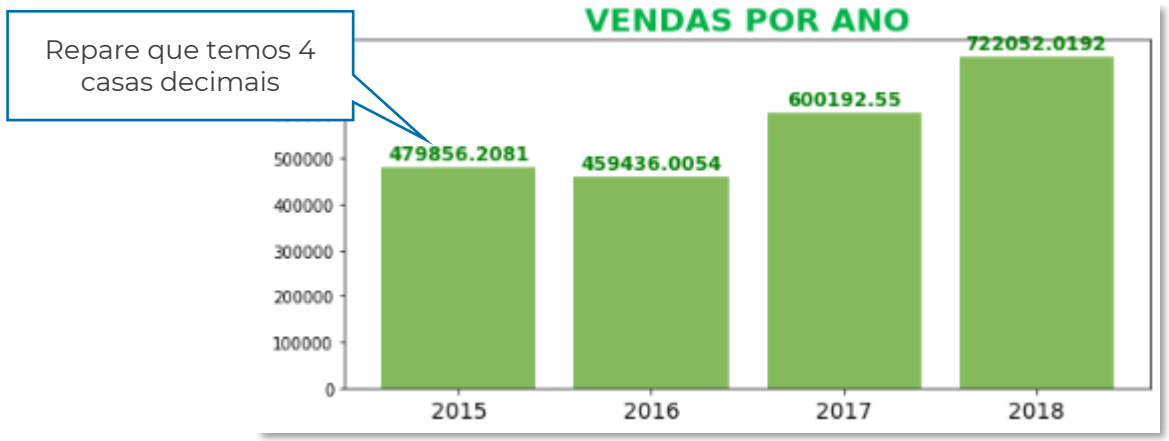
Agora vamos ajustar alterar a cor do rótulo de dados utilizando o **color**:

```
For i in range (0,4):
        ax.annotate(soma_ano.values[i],(soma_ano.index[i], soma_ano.values[i]),
        ha="center",
        xytext=(0,5),
        textcoords="offset points",
        fontsize=12,
        color="green",
)
```

Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Por último, vamos colocar o rótulo de dados em negrito utilizando o **fontweight**

```
For i in range (0,4):
        ax.annotate(soma_ano.values[i],(soma_ano.index[i], soma_ano.values[i]),
        ha="center",
        xytext=(0,5),
        textcoords="offset points",
        fontsize=12,
        color="green",
        fontweight="bold"
)
```



Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Para melhorar a visualização vamos formatar as casas decimais utilizando o **format**

Quantidade de casas decimais após o ponto

```
For i in range (0,4):
        ax.annotate('{:.2f}'.format(soma_ano.values[i]),
        (soma_ano.index[i], soma_ano.values[i]),

        ha="center",
        xytext=(0,5),
        textcoords="offset points",
        fontsize=12,
        color="green",
        fontweight="bold"
)
```

Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Para melhorar a visualização vamos formatar as casas decimais utilizando o **format**

Quantidade de casas decimais após o ponto

```
For i in range (0,4):
        ax.annotate('{:.0f}'.format(soma_ano.values[i]),
        (soma_ano.index[i], soma_ano.values[i]),

        ha="center",
        xytext=(0,5),
        textcoords="offset points",
        fontsize=12,
        color="green",
        fontweight="bold"
)
```

Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Vamos utilizar um separador decimal

Incluir a vírgula como separador decimal

```
For i in range (0,4):
        ax.annotate('{:,.0f}'.format(soma_ano.values[i]),
        (soma_ano.index[i], soma_ano.values[i]),

        ha="center",
        xytext=(0,5),
        textcoords="offset points",
        fontsize=12,
        color="green",
        fontweight="bold"
)
```

Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Vamos trocar o separador decimal para ponto utilizando a função **replace**

```
For i in range (0,4):
        ax.annotate('{:.0f}'.format(soma_ano.values[i]).replace(',','.'),
        (soma_ano.index[i], soma_ano.values[i]),
        ha="center",
        xytext=(0,5),
        textcoords="offset points",
        fontsize=12,
        color="green",
        fontweight="bold"
)
```

Substitui a vírgula por ponto

Repare que esse valor está cortado

Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Para o eixo y, podemos fazer igual fizemos para o eixo x

- Com o .set_yticks vamos ajustar os valores do eixo y

ax**.set_yticks**([0,780000])

Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

O **.yaxis.set_visible**(False) permite retirar os valores do eixo y, tornando nosso visual mais claro

ax**.set_yticks**([0,780000])

ax**.yaxis.set_visible**(False)

Módulo 9 – Adicionando e formatando rótulo de dados, ajustando o eixo y e retirando bordas

Já para retirar as bordas do gráfico, basta usar o **.spines**[‘posição**'].set_visible**(False)

ax**.set_yticks**([0,780000])

ax**.yaxis.set_visible**(False)

ax**.spines**['top'].**set_visible**(False)

ax**.spines**['left'].**set_visible**(False)

ax**.spines**['right'].**set_visible**(False)

Módulo 9 – Vendas por mês e transformando índices em colunas com o reset_index

Da mesma forma que fizemos as vendas por ano, também podemos mostrar essa visão por meses.

- Para isso, podemos usar o .**month** da biblioteca datetime para criar uma nova coluna apenas com o mês dessa base

Então vamos criar uma coluna com o mês:

```
base['Mes'] = base['Order Date'].dt.month
```

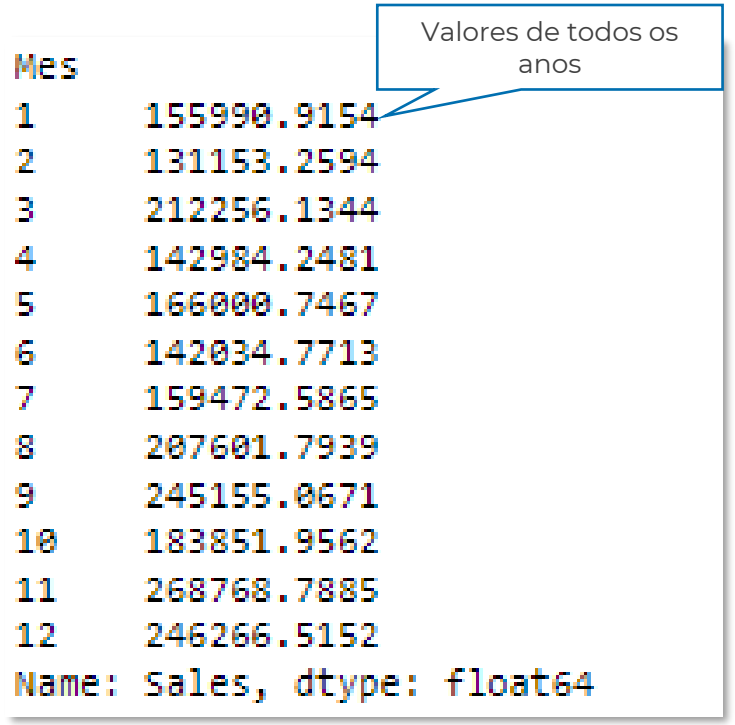
```
base.head()
```

Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category	Sub-Category	Product Name	Sales	Ano	Mes
Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-BO-10001798	Furniture	Bookcases	Bush Somerset Collection Bookcase	261.9600	2017	8
Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-CH-10000454	Furniture	Chairs	Hon Deluxe Fabric Upholstered Stacking Chairs,...	731.9400	2017	8
Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	California	90036.0	West	OFF-LA-10000240	Office Supplies	Labels	Self-Adhesive Address Labels for Typewriters b...	14.6200	2017	12
Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311.0	South	FUR-TA-10000577	Furniture	Tables	Bretford CR4500 Series Slim Rectangular Table	957.5775	2016	11
Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311.0	South	OFF-ST-10000760	Office Supplies	Storage	Eldon Fold 'N Roll Cart System	22.3680	2016	11

Módulo 9 – Vendas por mês e transformando índices em colunas com o reset_index

Agora vamos utilizar a função group by para agrupar os valores:

base.groupby('**Mes**')['**Sales**'].**sum**()



Módulo 9 – Vendas por mês e transformando índices em colunas com o reset_index

Agora vamos utilizar a função group by para agrupar os valores:

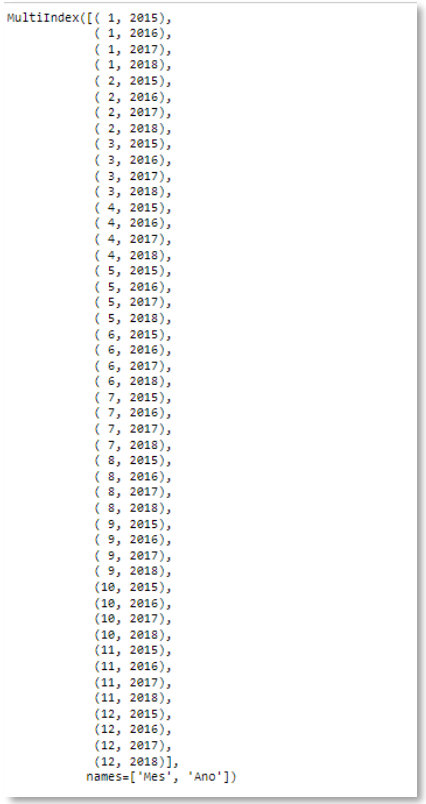
base.groupby('**Mes**')['**Sales**'].**sum**()

Para corrigir esse problema vamos utilizer duas colunas no groupby:

base.groupby(['**Mes**','**Ano**'])['**Sales**'].**sum**()

```
Mes  Ano
1    2015    28828.2540
     2016    29347.3864
     2017    38048.1840
     2018    59767.0910
2    2015    12588.4840
     2016    20728.3520
     2017    48907.5900
     2018    48928.8334
3    2015    54027.6920
     2016    34489.6776
     2017    48990.1410
     2018    74748.6238
4    2015    24710.0160
     2016    38056.9685
     2017    42368.0480
     2018    37849.2156
5    2015    29520.4900
     2016    30761.5585
     2017    64836.2518
     2018    40882.4464
6    2015    29181.3346
     2016    28515.9082
     2017    37424.6810
     2018    46912.8475
7    2015    35194.5580
     2016    28573.3100
     2017    41761.9430
     2018    53942.7755
8    2015    37349.2655
     2016    49076.9300
     2017    45766.8144
     2018    75408.7840
9    2015    65956.3998
     2016    65352.9970
     2017    40692.3063
     2018    73153.3640
10   2015    34561.9470
     2016    31631.8890
     2017    52156.9580
     2018    65501.1622
11   2015    64369.4565
     2016    50009.1450
     2017    66392.5470
     2018    87997.6400
12   2015    63568.3107
     2016    52891.8832
     2017    72847.0855
     2018    56959.2358
Name: Sales, dtype: float64
```

Módulo 9 – Vendas por mês e transformando índices em colunas com o reset_index



```
MultiIndex([( 1, 2015),
            ( 1, 2016),
            ( 1, 2017),
            ( 1, 2018),
            ( 2, 2015),
            ( 2, 2016),
            ( 2, 2017),
            ( 2, 2018),
            ( 3, 2015),
            ( 3, 2016),
            ( 3, 2017),
            ( 3, 2018),
            ( 4, 2015),
            ( 4, 2016),
            ( 4, 2017),
            ( 4, 2018),
            ( 5, 2015),
            ( 5, 2016),
            ( 5, 2017),
            ( 5, 2018),
            ( 6, 2015),
            ( 6, 2016),
            ( 6, 2017),
            ( 6, 2018),
            ( 7, 2015),
            ( 7, 2016),
            ( 7, 2017),
            ( 7, 2018),
            ( 8, 2015),
            ( 8, 2016),
            ( 8, 2017),
            ( 8, 2018),
            ( 9, 2015),
            ( 9, 2016),
            ( 9, 2017),
            ( 9, 2018),
            (10, 2015),
            (10, 2016),
            (10, 2017),
            (10, 2018),
            (11, 2015),
            (11, 2016),
            (11, 2017),
            (11, 2018),
            (12, 2015),
            (12, 2016),
            (12, 2017),
            (12, 2018)],
           names=['Mes', 'Ano'])
```

Para facilitar o nosso código podemos colocar esses valores agrupados em uma variável:

soma_mes = **base**.groupby(['**Mes**','**Ano**'])['**Sales**'].**sum**()

Agora, o **soma_mes**.**index** é uma dupla de valores

Módulo 9 – Vendas por mês e transformando índices em colunas com o reset_index

Para corrigir esse índice vamos transformar ele em **dataframe**.

Com o .**reset_index**() conseguimos transformar os índices em colunas e então aplicar todas as propriedades que conhecemos para o **DataFrame**.

```
soma_mes = soma_mes.reset_index()

soma_mes.head()
```

	level_0	index	Mes	Ano	Sales
0	0	0	1	2015	28828.2540
1	1	1	1	2016	29347.3864
2	2	2	1	2017	38048.1840
3	3	3	1	2018	59767.0910
4	4	4	2	2015	12588.4840

Módulo 9 – Vendas por mês e transformando índices em colunas com o reset_index

Supondo que queremos filtrar apenas o ano de 2015, como poderíamos fazer?

soma_mes[**soma_mes**.**Ano** == 2015]

```
soma_mes[soma_mes.Ano == 2015]
```

	level_0	index	Mes	Ano	Sales
0	0	0	1	2015	28828.2540
4	4	4	2	2015	12588.4840
8	8	8	3	2015	54027.6920
12	12	12	4	2015	24710.0160
16	16	16	5	2015	29520.4900
20	20	20	6	2015	29181.3346
24	24	24	7	2015	35194.5580
28	28	28	8	2015	37349.2655
32	32	32	9	2015	65956.3998
36	36	36	10	2015	34561.9470
40	40	40	11	2015	64369.4565
44	44	44	12	2015	63568.3107

Módulo 9 – Vendas por mês e transformando índices em colunas com o reset_index

Supondo que queremos filtrar apenas o mês 1, como poderíamos fazer?

soma_mes[**soma_mes**.**Mes** == 1]

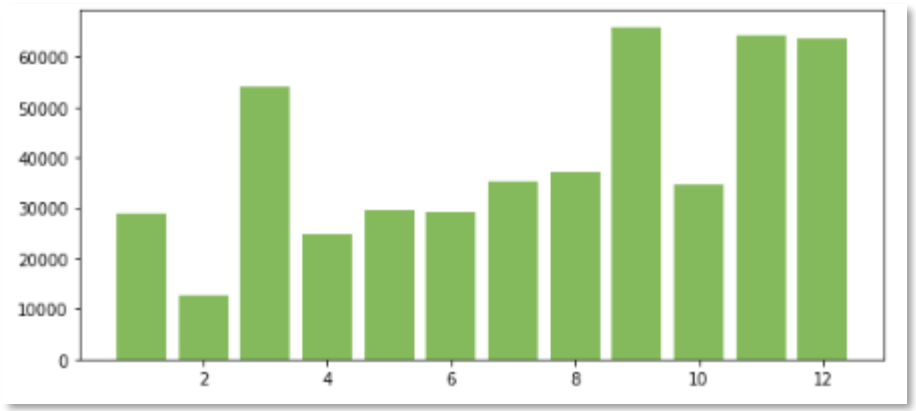
```
soma_mes[soma_mes.Mes == 1]
```

	level_0	index	Mes	Ano	Sales
0	0	0	1	2015	28828.2540
1	1	1	1	2016	29347.3864
2	2	2	1	2017	38048.1840
3	3	3	1	2018	59767.0910

Módulo 9 – Entendendo o deslocamento das barras em um único gráfico de barras horizontais

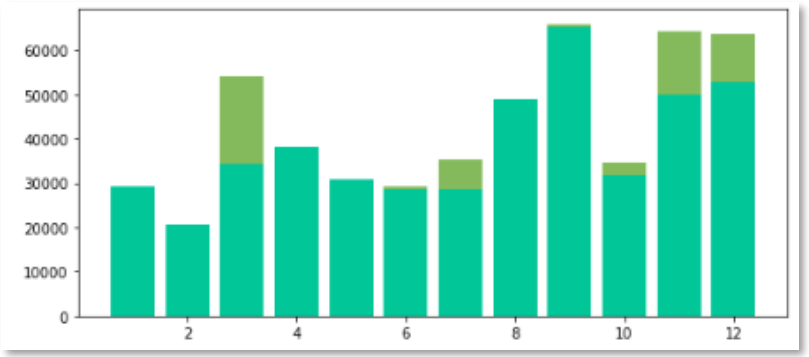
Vamos realizar um gráfico, para isso vamos usar como base o nosso gráfico anterior substituindo os valores de x e y:

```
fig,ax = plt.subplots(figsize = (9,4))

ax.bar(soma_mes[soma_mes.Ano == 2015].Mes.values,
       soma_mes[soma_mes.Ano == 2015].Sales.values,
       color="#84ba5b", label=2015)

plt.show()
```



Como poderíamos fazer o mesmo gráfico para 2016?

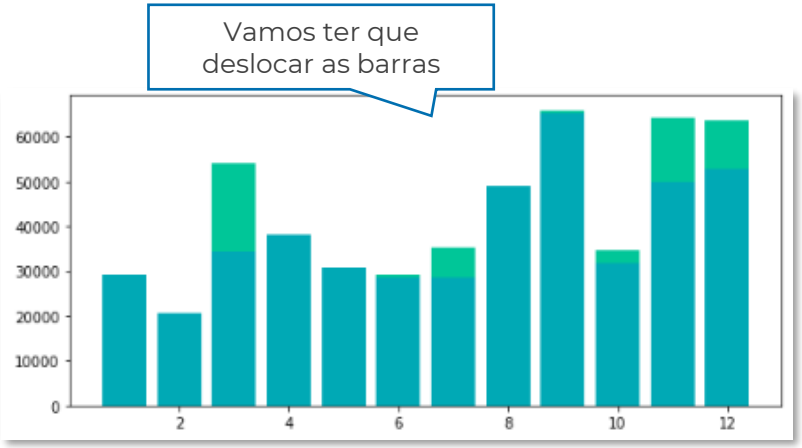
```
fig,ax = plt.subplots(figsize = (9,4))

ax.bar(soma_mes[soma_mes.Ano == 2015].Mes.values,
       soma_mes[soma_mes.Ano == 2015].Sales.values,
       color="#84ba5b", label=2015)

ax.bar(soma_mes[soma_mes.Ano == 2016].Mes.values,
       soma_mes[soma_mes.Ano == 2016].Sales.values,
       color="#00c698", label=2016)

plt.show()
```



Módulo 9 – Entendendo o deslocamento das barras em um único gráfico de barras horizontais

Como poderíamos fazer o mesmo gráfico para 2016?

```
fig,ax = plt.subplots(figsize = (9,4))

ax.bar(soma_mes[soma_mes.Ano == 2015].Mes.values,
       soma_mes[soma_mes.Ano == 2015].Sales.values,
       color="#00c698", label=2015)

ax.bar(soma_mes[soma_mes.Ano == 2016].Mes.values,
       soma_mes[soma_mes.Ano == 2016].Sales.values,
       color="#00a9b5", label=2016)

plt.show()
```



Módulo 9 – Entendendo o deslocamento das barras em um único gráfico de barras horizontais

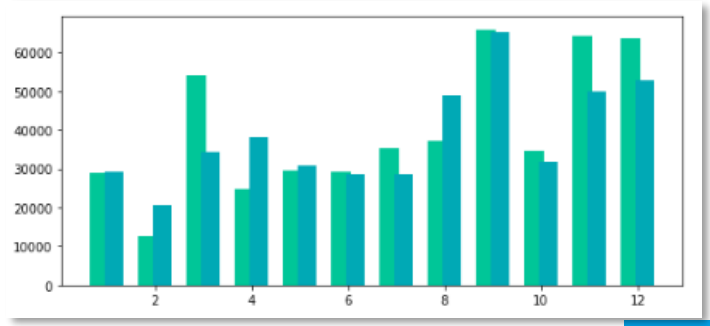
Vamos ajustar essa parte do código adicionando as partes do tamanho das barras

fig,ax = **plt.subplots**(**figsize** = (9,4))

Deslocando a barra para esquerda

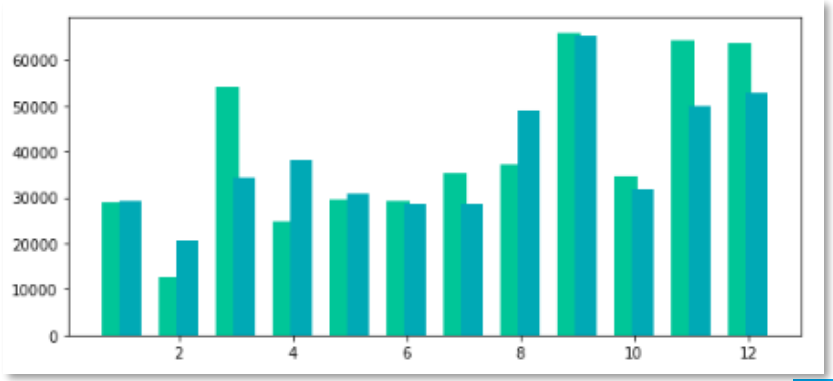
ax.**bar**(**soma_mes**[**soma_mes**.**Ano**== 2015].**Mes**.values-0.4/2, **soma_mes**[**soma_mes**.**Ano** == 2015].**Sales**.**values**, **color**="#00c698", **label**=2015, width = 0.4)

ax.**bar**(**soma_mes**[**soma_mes**.**Ano**==2016].**Mes**.values+0.4/2, **soma_mes**[**soma_mes**.**Ano** == 2016].**Sales**.**values**, **color**="#00a9b5", **label**=2016, width = 0.4)
plt.show()



Módulo 9 – Entendendo o deslocamento das barras em um único gráfico de barras horizontais

```
fig,ax = plt.subplots(figsize = (9,4))
wid = 0.4

ax.bar(soma_mes[soma_mes.Ano==2015].Mes.values-wid/2,soma_mes[soma_mes.Ano == 2015].Sales.values, color="#00c698", label=2015, width = wid)

ax.bar(soma_mes[soma_mes.Ano==2016].Mes.values+wid/2,soma_mes[soma_mes.Ano == 2016].Sales.values,        color="#00a9b5", label=2016, width = wid)

plt.show()
```



Módulo 9 – Adicionando todos os anos no gráfico de barras e colocando rótulo de dados

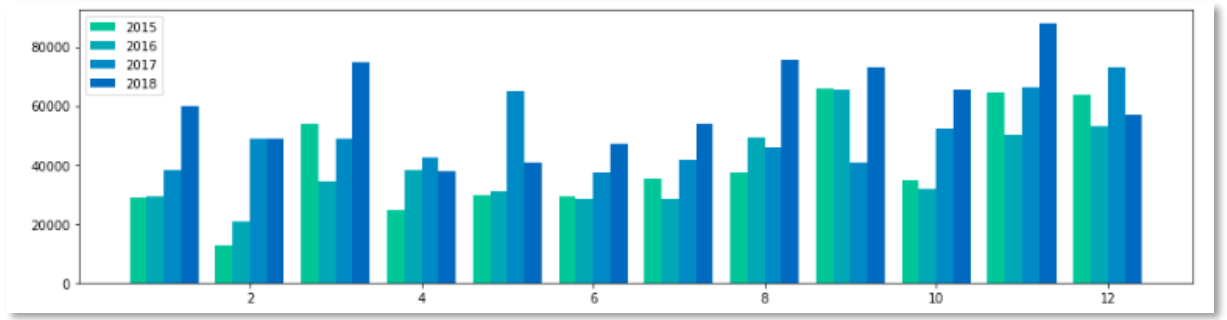
Agora vamos fazer esse mesmo gráfico para os anos de 2017 e 2018:

```
fig,ax = plt.subplots(
            figsize = (16,4) # Tamanho da figura
        )

wid = 0.2 # Tamanho das barras

ax.bar(soma_mes[soma_mes.Ano == 2015].Mes.values-wid-wid/2, # Ajuste da posição das barras
       soma_mes[soma_mes.Ano == 2015].Sales.values,
       color="#00c698", # Cor das barras
       label=2015, # Label dos dados
       width=wid # Largura das barras
      )
ax.bar(soma_mes[soma_mes.Ano == 2016].Mes.values-wid/2, # Ajuste da posição das barras
       soma_mes[soma_mes.Ano == 2016].Sales.values,
       color="#00a9b5",
       label=2016,
       width=wid
      )

# Adicionando o restante das barras
ax.bar(soma_mes[soma_mes.Ano == 2017].Mes.values+wid/2, # Ajuste da posição das barras
       soma_mes[soma_mes.Ano == 2017].Sales.values,
       color="#008ac5",
       label=2017,
       width=wid
      )
ax.bar(soma_mes[soma_mes.Ano == 2018].Mes.values+wid+wid/2, # Ajuste da posição das barras
       soma_mes[soma_mes.Ano == 2018].Sales.values,
       color="#0069c0",
       label=2018,
       width=wid
      )
ax.legend()

plt.show()
```

Ajuste da largura

Colocar o 2016 e 2017 no centro

Ajuste da largura



Módulo 9 – Adicionando todos os anos no gráfico de barras e colocando rótulo de dados

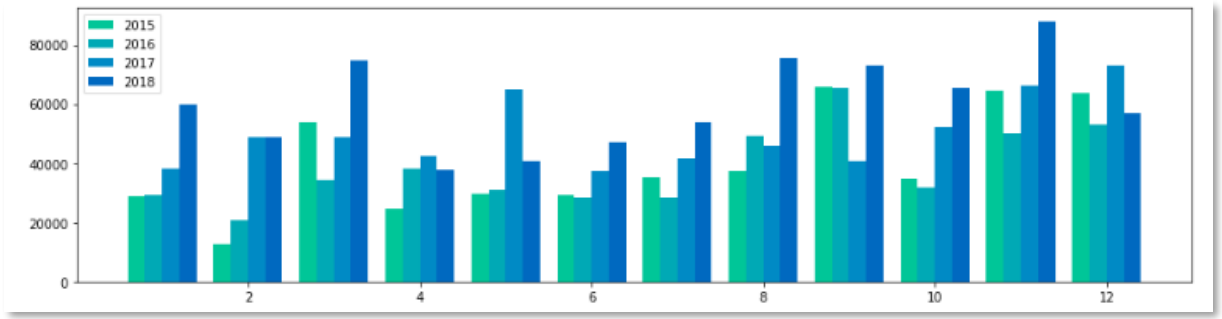
Agora vamos fazer esse mesmo gráfico para os anos de 2017 e 2018:

```
fig,ax = plt.subplots(
            figsize = (16,4) # Tamanho da figura
        )

wid = 0.2 # Tamanho das barras

ax.bar(soma_mes[soma_mes.Ano == 2015].Mes.values-wid-wid/2, # Ajuste da posição das barras
       soma_mes[soma_mes.Ano == 2015].Sales.values,
       color="#00c698", # Cor das barras
       label=2015, # Label dos dados
       width=wid # Largura das barras
      )
ax.bar(soma_mes[soma_mes.Ano == 2016].Mes.values-wid/2, # Ajuste da posição das barras
       soma_mes[soma_mes.Ano == 2016].Sales.values,
       color="#00a9b5",
       label=2016,
       width=wid
      )

# Adicionando o restante das barras
ax.bar(soma_mes[soma_mes.Ano == 2017].Mes.values+wid/2, # Ajuste da posição das barras
       soma_mes[soma_mes.Ano == 2017].Sales.values,
       color="#008ac5",
       label=2017,
       width=wid
      )
ax.bar(soma_mes[soma_mes.Ano == 2018].Mes.values+wid+wid/2, # Ajuste da posição das barras
       soma_mes[soma_mes.Ano == 2018].Sales.values,
       color="#0069c0",
       label=2018,
       width=wid
      )
ax.legend()

plt.show()
```

Ajuste da largura

Colocar o 2016 e 2017 no centro

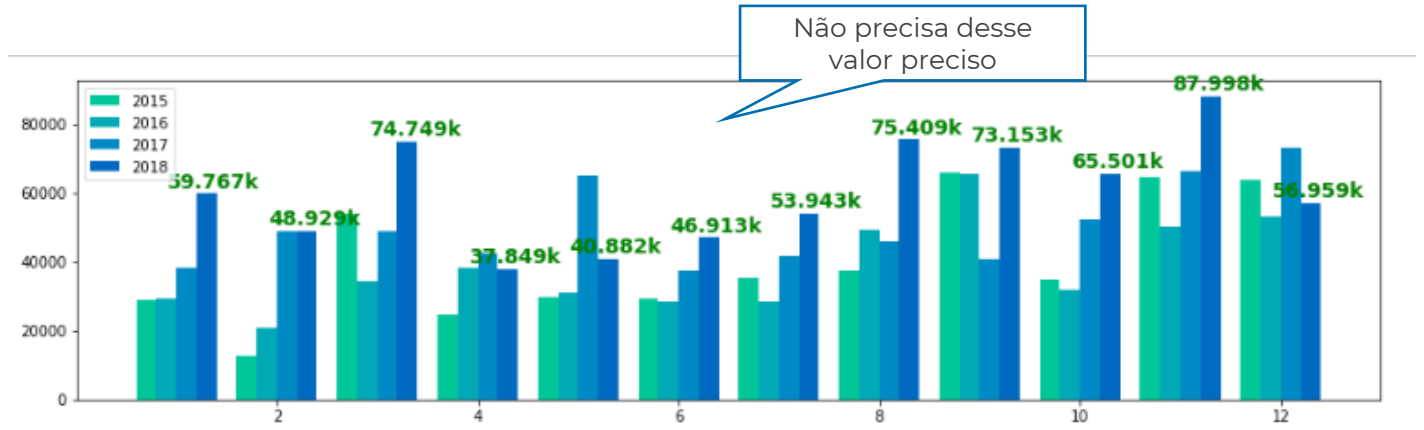
Ajuste da largura



Módulo 9 – Adicionando todos os anos no gráfico de barras e colocando rótulo de dados

Então vamos utilizar o código que já utilizamos do **annotate** apenas em 2018, para o gráfico não ficar muito poluído.

```
for i in np.arange(0,12):
ax.annotate('{:,.0f}k'.format(soma_mes[soma_mes.Ano==2018].Sales.values[i])
.replace(',','.'), (soma_mes[soma_mes.Ano==2018].Mes.values[i]+wid+wid/2,
        soma_mes[soma_mes.Ano == 2018].Sales.values[i]),
        ha="center",
        va="top",
        xytext=(0,+15),
        textcoords="offset points",
        fontsize=14,
        fontweight='bold',
        color="green")
```

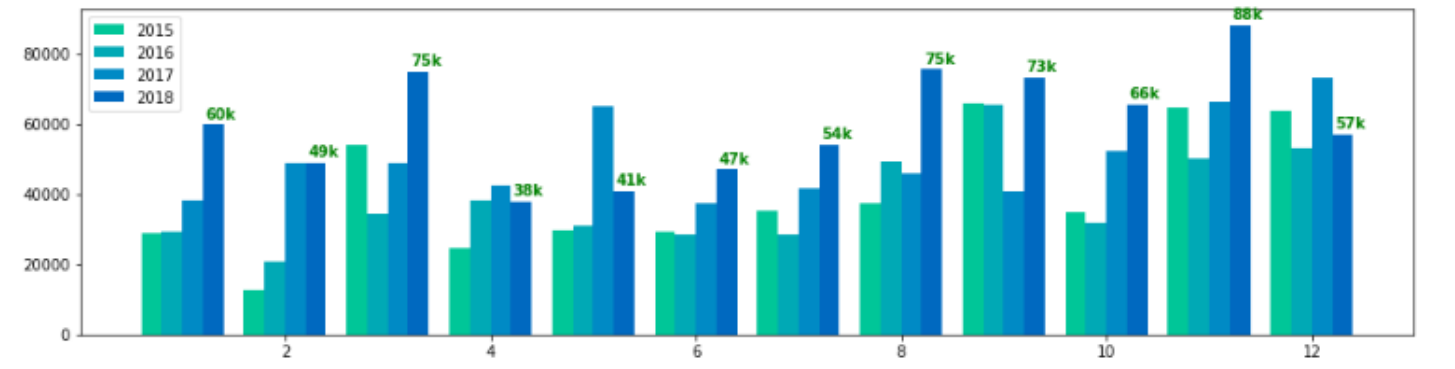


Módulo 9 – Adicionando todos os anos no gráfico de barras e colocando rótulo de dados

```
for i in np.arange(0,12):

ax.annotate('{:,.0f}k'.format(soma_mes[soma_mes.Ano==2018].Sales.values[i]/1000),
            (soma_mes[soma_mes.Ano==2018].Mes.values[i]+wid+wid/2,
            soma_mes[soma_mes.Ano == 2018].Sales.values[i]),
            ha="center",
            va="top",
            xytext=(0,+15),
            textcoords="offset points",
            fontsize=14,
            fontweight='bold',
            color="green")
```

Colocar o K de mil

Dividindo o valor de vendas por 1000

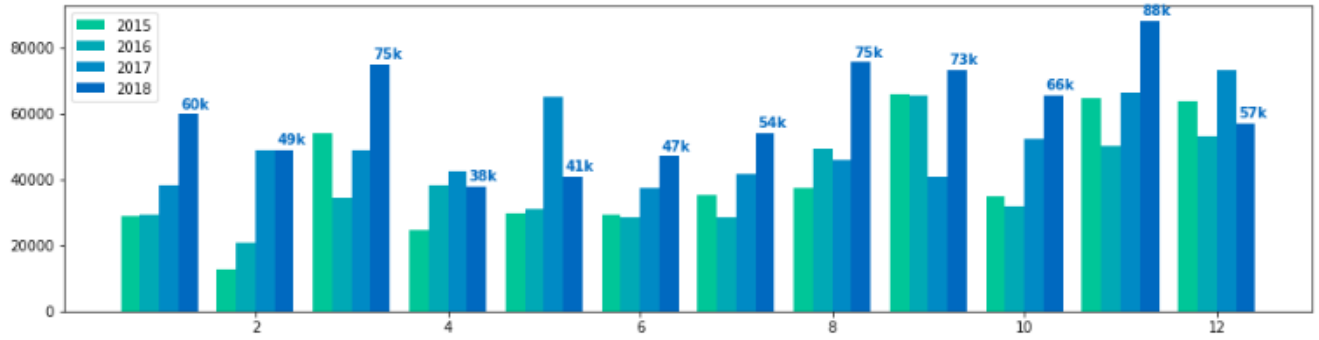


Módulo 9 – Adicionando todos os anos no gráfico de barras e colocando rótulo de dados

```
for i in np.arange(0,12):

ax.annotate('{:,.0f}k'.format(soma_mes[soma_mes.Ano==2018].Sales.values[i]/1000),
            (soma_mes[soma_mes.Ano==2018].Mes.values[i]+wid+wid/2,
            soma_mes[soma_mes.Ano == 2018].Sales.values[i]),
            ha="center",
            va="top",
            xytext=(5,12),
            textcoords="offset points",
            fontsize=10,
            fontweight='bold',
            color="#0069c0")
```

Alteração da posição

Alteração da cor

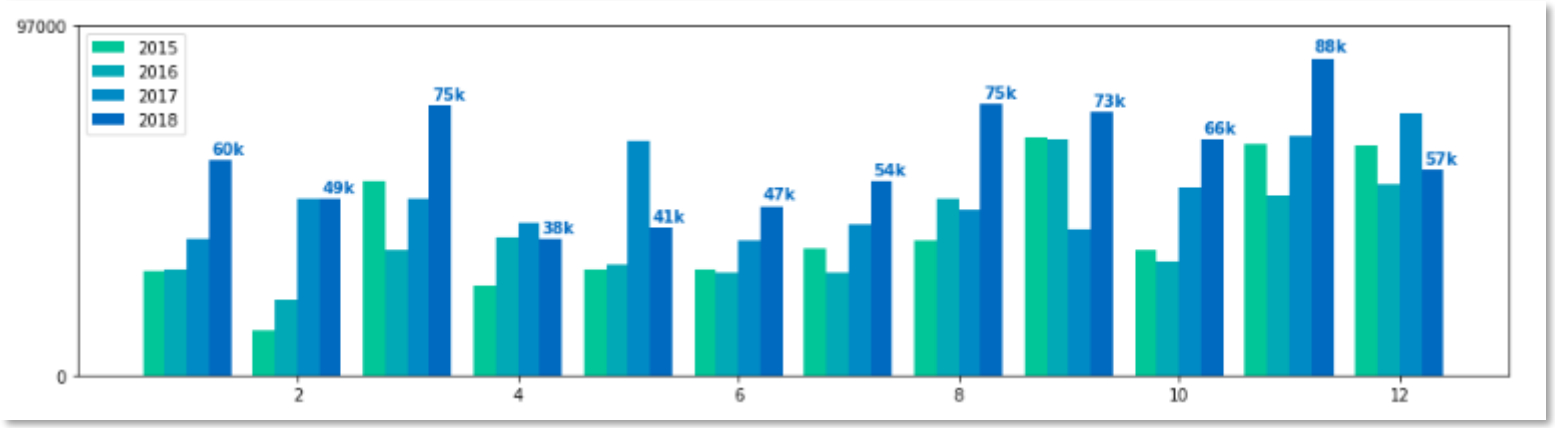


Módulo 9 – Mudando os rótulos do eixo x e finalizando o visual de venda do mês

Agora vamos finalizar o visual, para isso vamos aproveitar o código que fizemos para o ano:

Vamos aumentar o eixo y:

ax.**set_yticks**(**np.array**([0,97000]))



Módulo 9 – Mudando os rótulos do eixo x e finalizando o visual de venda do mês

Agora vamos retirar as bordas:

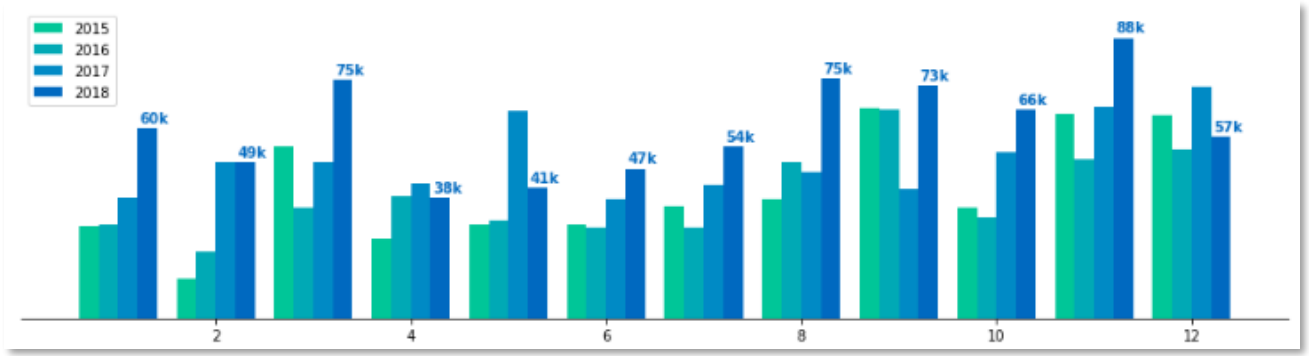
ax.**set_yticks**(**np.array**([0,97000]))

ax.**yaxis.set_visible**(False)

ax.**spines**['top'].**set_visible**(False)

ax.**spines**['left'].**set_visible**(False)

ax.**spines**['right'].**set_visible**(False)

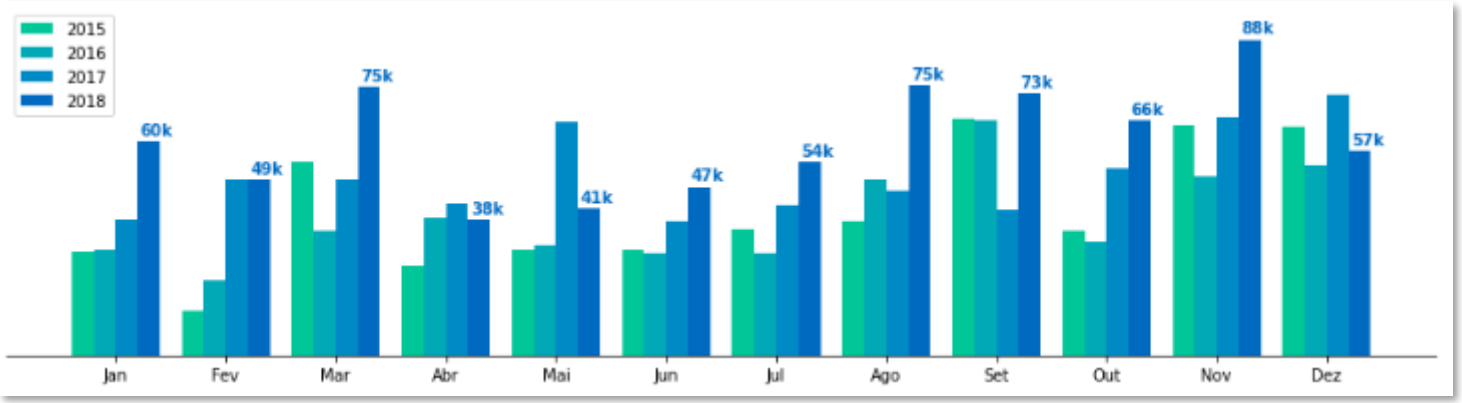


Módulo 9 – Mudando os rótulos do eixo x e finalizando o visual de venda do mês

Agora vamos ajustar os valores de x:

ax.**xaxis.set_ticks**(**np.arange**(1,13))

ax.**set_xticklabels**(['Jan','Fev','Mar','Abr','Mai','Jun','Jul','Ago','Set','Out','Nov','Dez'])



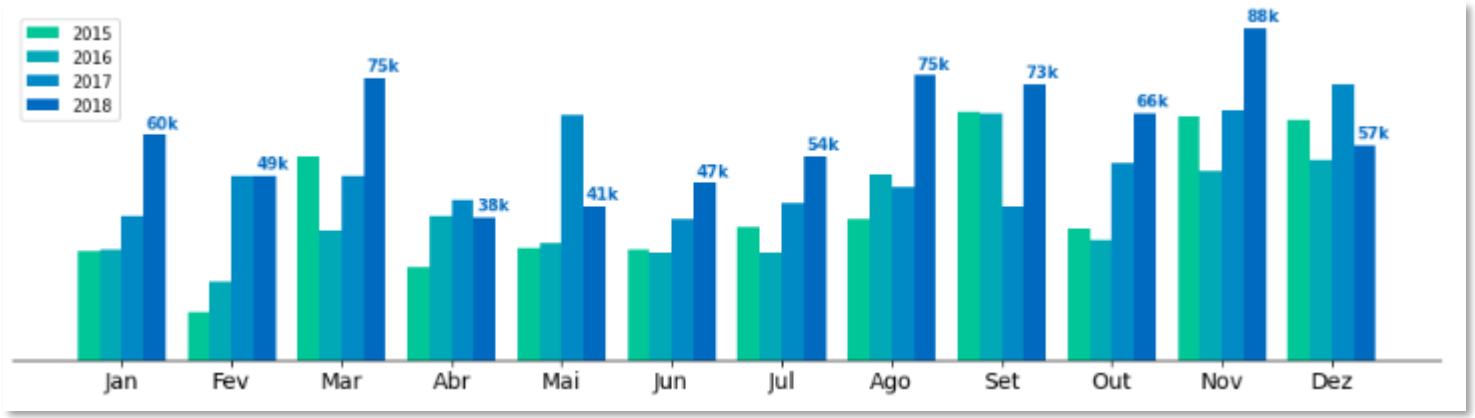
Módulo 9 – Mudando os rótulos do eixo x e finalizando o visual de venda do mês

Aumentando a fonte do eixo x:

ax.**xaxis.set_ticks**(**np.arange**(1,13))

ax.**set_xticklabels**(['Jan','Fev','Mar','Abr','Mai','Jun','Jul','Ago','Set','Out','Nov','Dez'])

ax.**tick_params**(axis='x',**labelsize**=14)



Módulo 9 – Respondendo qual foi a categoria mais vendida

O primeiro passo para responder qual a categoria mais vendida é verificar na base qual coluna poderíamos utilizar:

base.head(2)

Então, podemos ver que já existe uma coluna para categoria

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category
1	CA-2017-152156	2017-08-11	2017-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-BO-10001798	Furniture
2	CA-2017-152156	2017-08-11	2017-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-CH-10000454	Furniture

Módulo 9 – Respondendo qual foi a categoria mais vendida

O primeiro passo para responder qual a categoria mais vendida é verificar na base qual coluna poderíamos utilizar:

base.head(2)

Então, podemos ver que já existe uma coluna para categoria

Coluna que vamos utilizar

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category
1	CA-2017-152156	2017-08-11	2017-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-BO-10001798	Furniture
2	CA-2017-152156	2017-08-11	2017-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-CH-10000454	Furniture

Módulo 9 – Respondendo qual foi a categoria mais vendida

Então, o segundo passo é utilizar novamente o group by:

base.**groupby**("**Category**")["**Sales**"].**sum**()

Dessa forma, ainda não conseguimos passar a informação de categoria por ano, para isso vamos utilizer um group by com o ano também

```
Category
Furniture          728658.5757
Office Supplies    705422.3340
Technology         827455.8730
Name: Sales, dtype: float64
```

Módulo 9 – Respondendo qual foi a categoria mais vendida

Então alterando o nosso group by para:

categoria = **base**.**groupby**([" **Category** ","**Ano**"])[" **Sales** "].**sum**()

categoria

```
categoria = base.groupby(["Category","Ano"])["Sales"].sum(
categoria
```

```
Category         Ano
Furniture        2015    156477.8811
                 2016    164053.8674
                 2017    195813.0400
                 2018    212313.7872
Office Supplies  2015    149512.8200
                 2016    133124.4070
                 2017    182417.5660
                 2018    240367.5410
Technology       2015    173865.5070
                 2016    162257.7310
                 2017    221961.9440
                 2018    269370.6910
Name: Sales, dtype: float64
```

Vamos ter que transformar em **Data Frame**

Módulo 9 – Respondendo qual foi a categoria mais vendida

Transformando em Data Frame utilizando o reset_index:

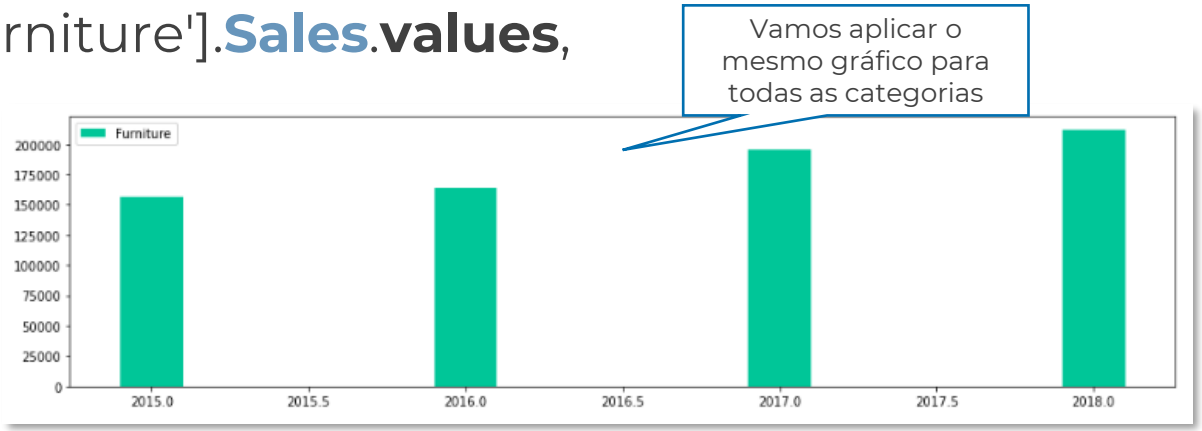
```
categoria = categoria.reset_index()

categoria
```

	index	Category	Ano	Sales
0	0	Furniture	2015	156477.8811
1	1	Furniture	2016	164053.8674
2	2	Furniture	2017	195813.0400
3	3	Furniture	2018	212313.7872
4	4	Office Supplies	2015	149512.8200
5	5	Office Supplies	2016	133124.4070
6	6	Office Supplies	2017	182417.5660
7	7	Office Supplies	2018	240367.5410
8	8	Technology	2015	173865.5070
9	9	Technology	2016	162257.7310
10	10	Technology	2017	221961.9440
11	11	Technology	2018	269370.6910

Módulo 9 – Respondendo qual foi a categoria mais vendida

Agora vamos copiar exatamente o código anterior para editar e criar o nosso gráfico de vendas. Vamos ajustar os valores de x e y:

```
fig,ax = plt.subplots(figsize = (16,4))
wid = 0.2

ax.bar(categoria[categoria.Category == 'Furniture'].Ano.values,
       categoria[categoria.Category == 'Furniture'].Sales.values,
       color="#00c698",
       label="Furniture" ,
       width=wid)

ax.legend()
plt.show()
```



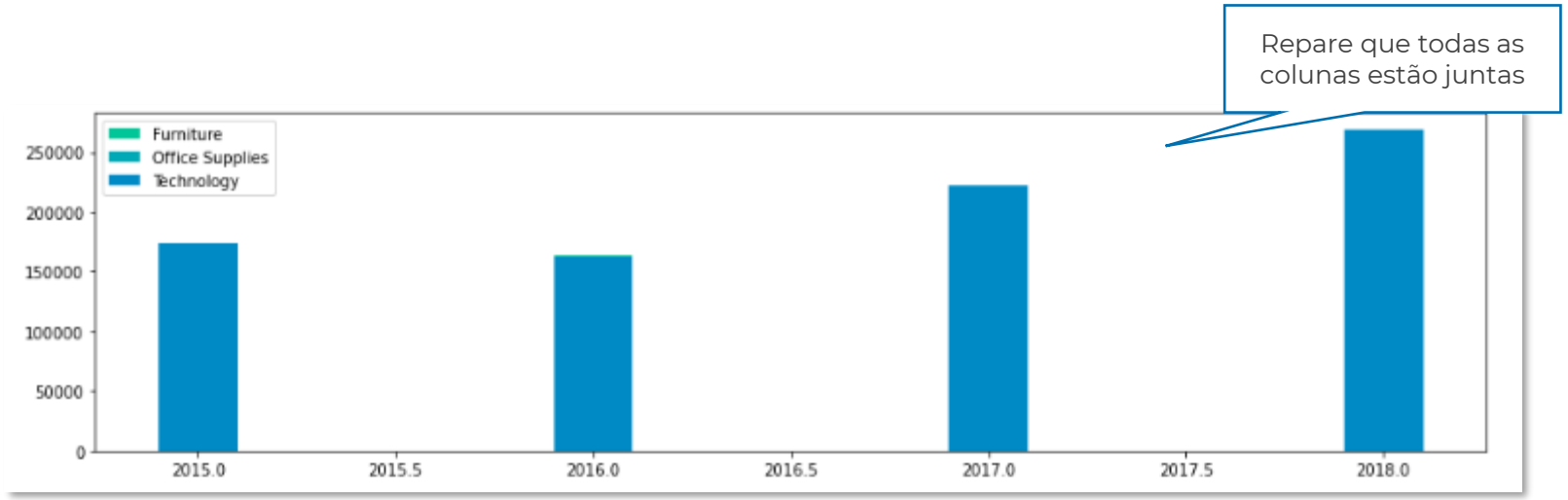
Módulo 9 – Respondendo qual foi a categoria mais vendida

```
fig,ax = plt.subplots(figsize = (16,4))
ax.bar(categoria[categoria.Category == 'Furniture'].Ano.values,
       categoria[categoria.Category == 'Furniture'].Sales.values,
       color="#00c698", label="Furniture", width=wid)

ax.bar(categoria[categoria.Category == 'Office Supplies'].Ano.values,
 categoria[categoria.Category == 'Office Supplies'].Sales.values,
       color="#00a9b5", abel="Office Supplies", width=wid)

ax.bar(categoria[categoria.Category == 'Technology'].Ano.values,
 categoria[categoria.Category == 'Technology'].Sales.values,
color="#008ac5", label="Technology", width=wid)
ax.legend()
plt.show()
```

Módulo 9 – Respondendo qual foi a categoria mais vendida

Como resultado do código anterior temos.

Agora vamos precisar deslocar uma coluna para direita e outra para a esquerda



Módulo 9 – Respondendo qual foi a categoria mais vendida

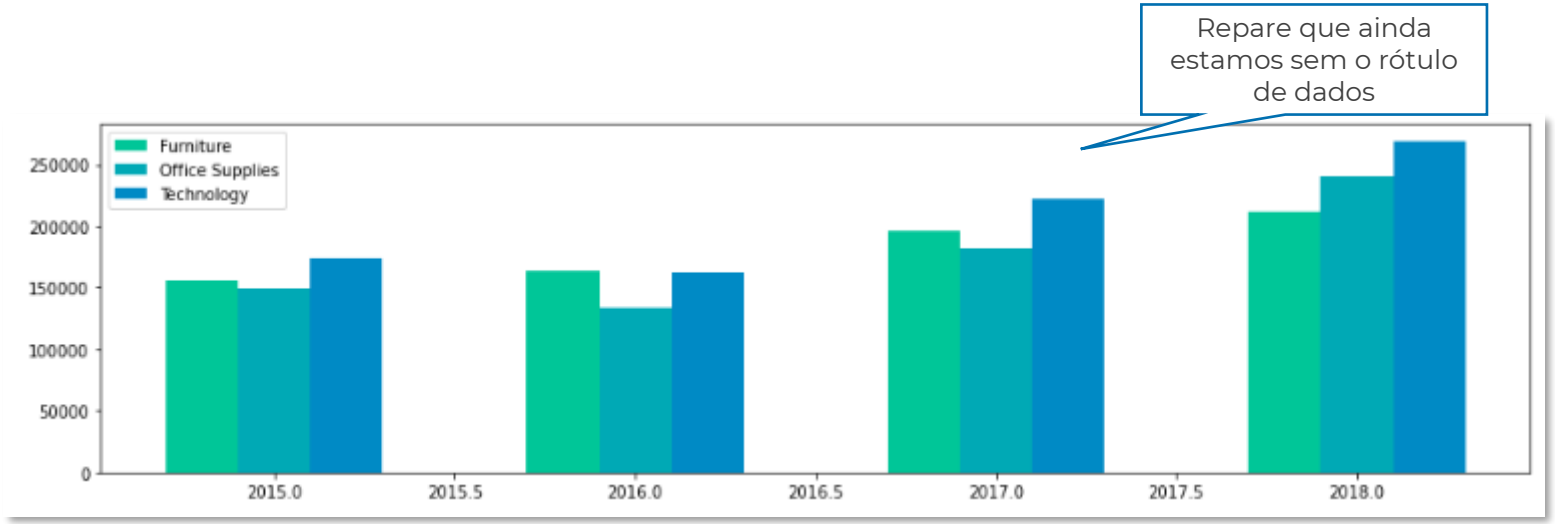
```
fig,ax = plt.subplots(figsize = (16,4))
wid = 0.2
ax.bar(categoria[categoria.Category == 'Furniture'].Ano.values-wid,
       categoria[categoria.Category == 'Furniture'].Sales.values,
       color="#00c698", label="Furniture", width=wid)

ax.bar(categoria[categoria.Category == 'Office Supplies'].Ano.values,
 categoria[categoria.Category == 'Office Supplies'].Sales.values,
       color="#00a9b5", abel="Office Supplies", width=wid)
ax.bar(categoria[categoria.Category == 'Technology'].Ano.values+wid,
 categoria[categoria.Category == 'Technology'].Sales.values,
color="#008ac5", label="Technology", width=wid)
ax.legend()
plt.show()
```

Deslocar coluna para a esquerda

Deslocar coluna para a direita

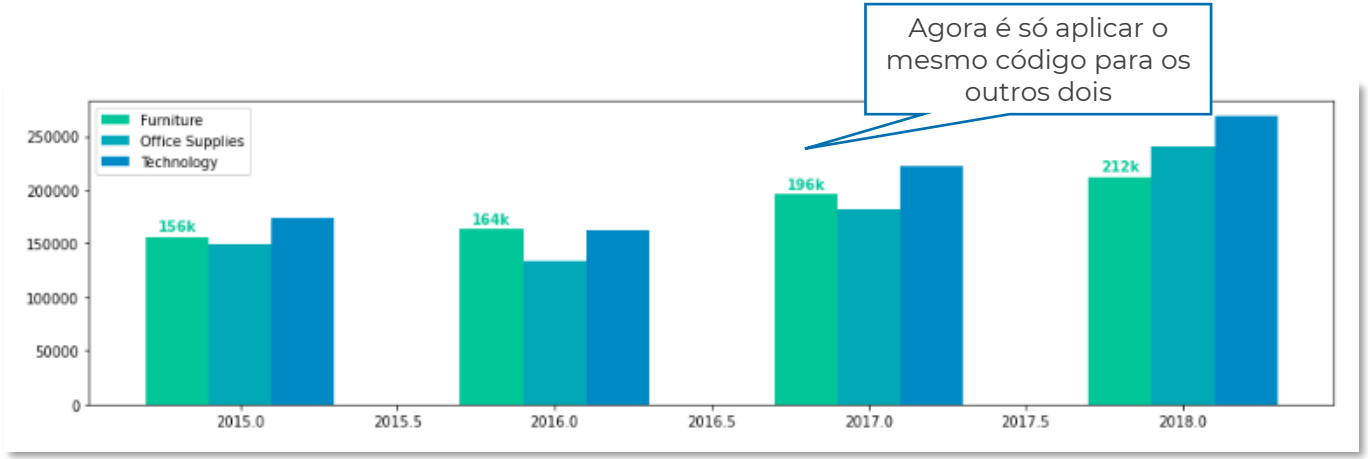
Então, ficamos com o gráfico abaixo como resultado.

O próximo passo vai ser utilizando o **annotate** do outro gráfico para colocar os rótulos de dados e alterando os valore de x e y:

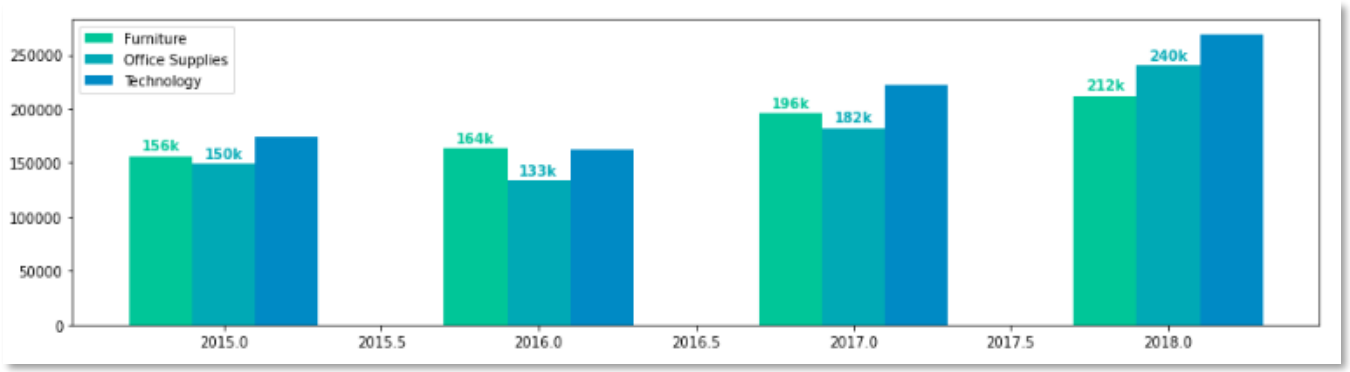


Módulo 9 – Respondendo qual foi a categoria mais vendida

```
for i in np.arange(0,4):

ax.annotate('{:,.0f}k'.format(categoria[categoria.Category=='Furniture'].Sales.values[i]
/1000),
        (categoria[categoria.Category == 'Furniture'].Ano.values[i]-wid,
        categoria[categoria.Category == 'Furniture'].Sales.values[i]),
        ha="center“,
        va="top",
        xytext=(0,12),
        textcoords="offset points",
        fontsize=10,
        fontweight='bold’,
        color="#00c698")
```

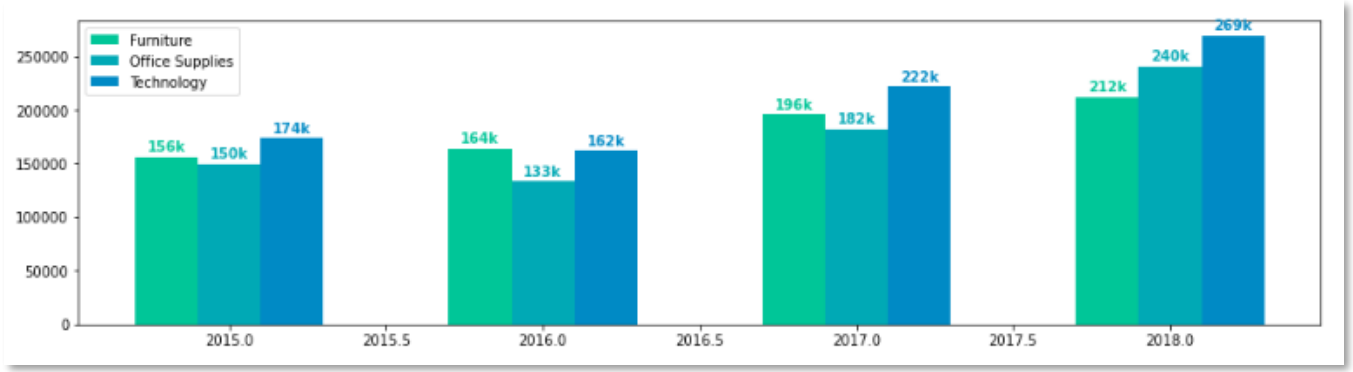


```
for i in np.arange(0,4):

    ax.annotate('{:,.0f}k'.format(categoria[categoria.Category=='Office     Supplies'].Sales.
values[i]/1000),
            (categoria[categoria.Category == 'Office Supplies'].Ano.values[i]-wid,
            categoria[categoria.Category == 'Office Supplies'].Sales.values[i]),
              ha="center“,
            va="top",
            xytext=(0,12),
            textcoords="offset points",
            fontsize=10,
            fontweight='bold’,
            color="#00a9b5")
```



Módulo 9 – Respondendo qual foi a categoria mais vendida

```
for i in np.arange(0,4):

ax.annotate('{:,.0f}k'.format(categoria[categoria.Category=='Technology'].Sales.values[
i]/1000),
            (categoria[categoria.Category == 'Technology'].Ano.values[i]-wid,
            categoria[categoria.Category == 'Technology'].Sales.values[i]),
              ha="center“,
            va="top",
            xytext=(0,12),
            textcoords="offset points",
            fontsize=10,
            fontweight='bold’,
            color="#008ac5")
```

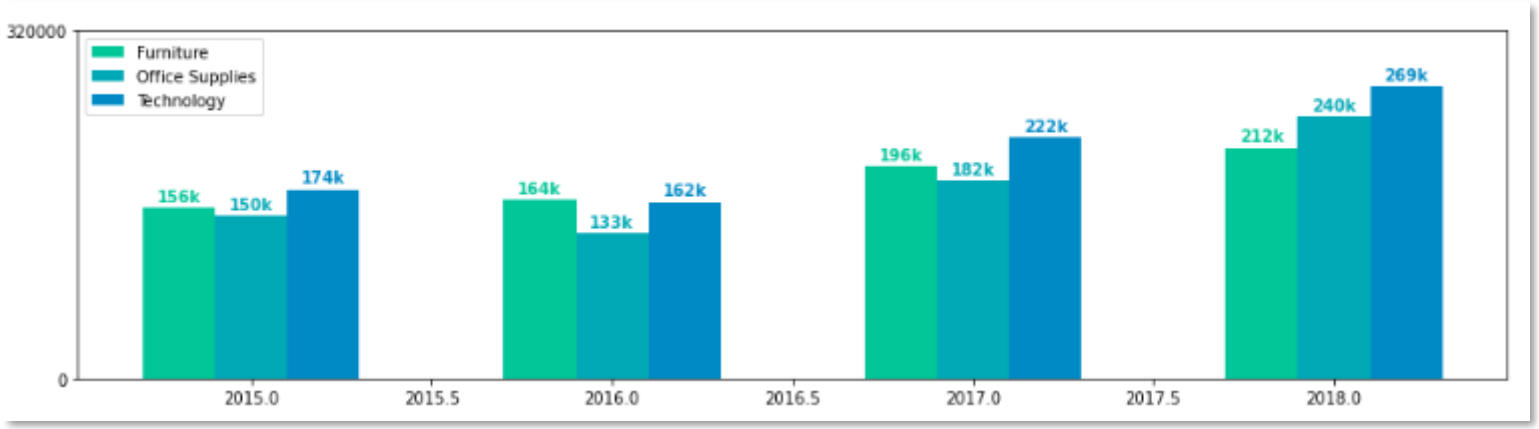


Módulo 9 – Respondendo qual foi a categoria mais vendida

Agora vamos melhorar os valores do eixo y:

- Diminuindo o tamanho das barras para não cortar o rótulo

ax.**set_yticks**(**np.array**([0,320000]))



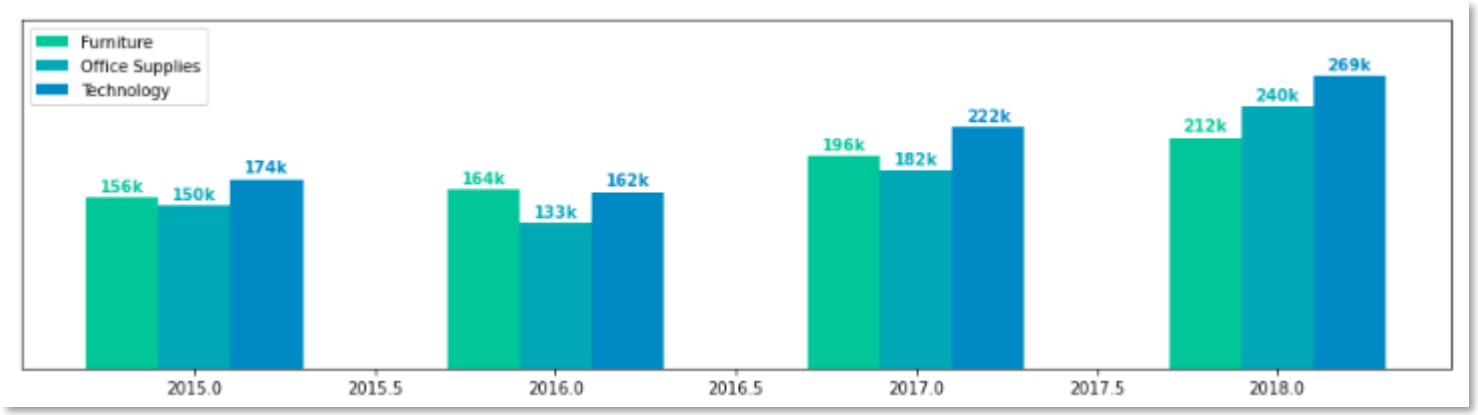
Módulo 9 – Respondendo qual foi a categoria mais vendida

Agora vamos melhorar os valores do eixo y:

- Retirando os valores do eixo y

ax.**set_yticks**(**np.array**([0,320000]))

ax.**yaxis.set_visible**(False



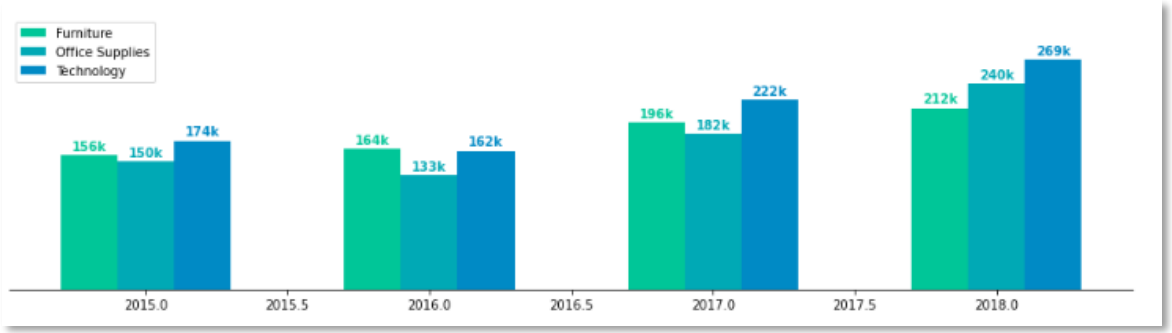
Módulo 9 – Respondendo qual foi a categoria mais vendida

Agora vamos retirar as bordas do gráfico:

ax.**spines**['top'].**set_visible**(False)

ax.**spines**['left'].**set_visible**(False)

ax.**spines**['right'].**set_visible**(False)

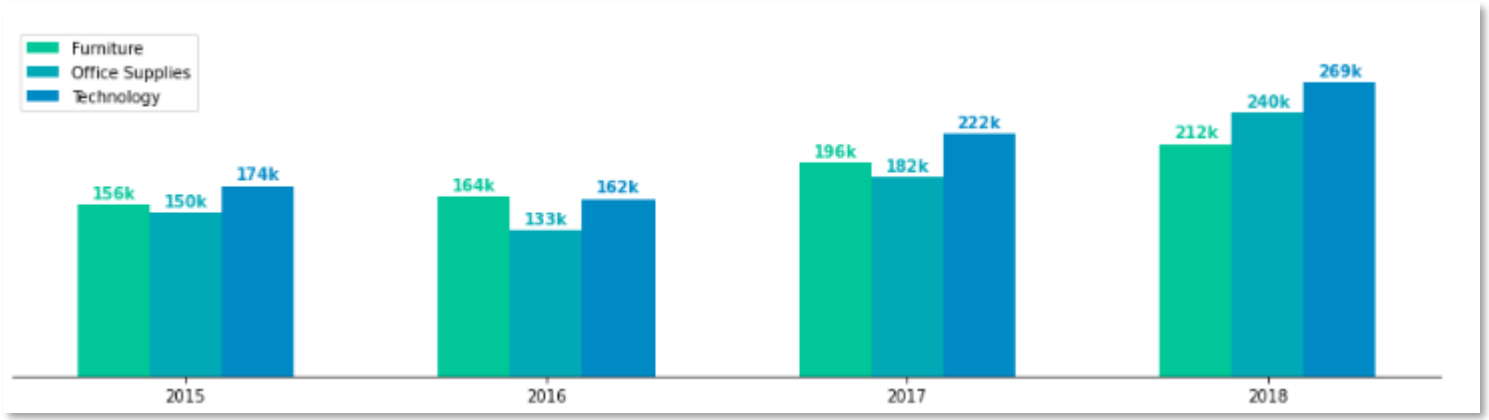


Módulo 9 – Respondendo qual foi a categoria mais vendida

Agora vamos melhorar os valores do eixo x:

- Ajustando os valores de x

ax.**xaxis.set_ticks**([2015,2016,2017,2018])

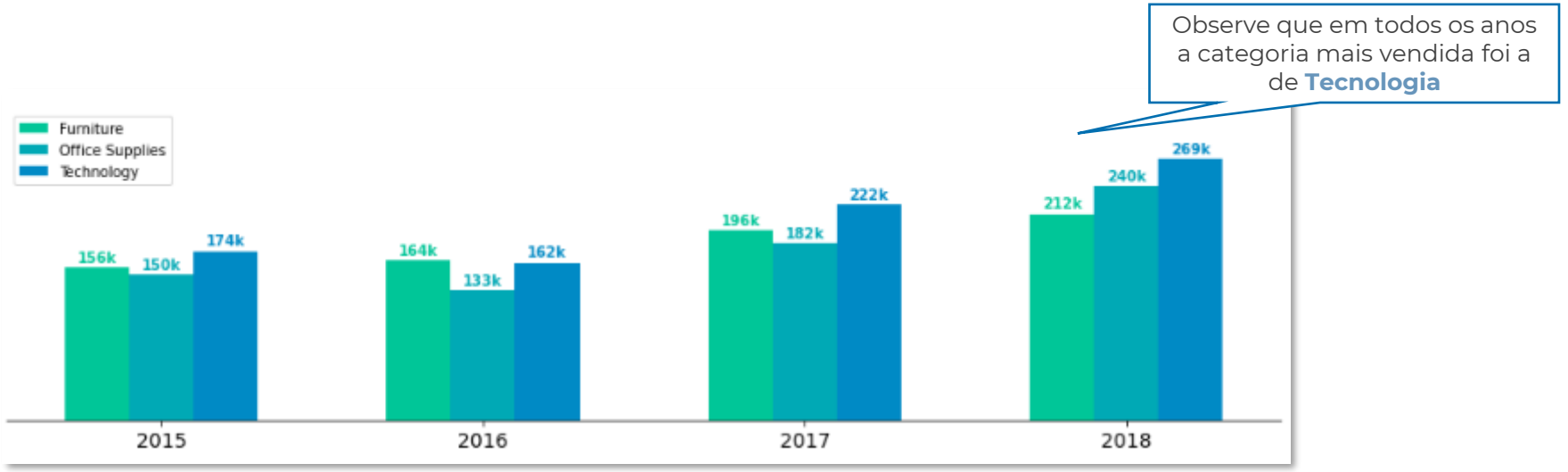


Módulo 9 – Respondendo qual foi a categoria mais vendida

Agora vamos melhorar os valores do eixo x:

- Melhorando o visual do eixo x

ax.**tick_params**(axis='x',**labelsize**=14



Módulo 9 – Criando um gráfico de barras horizontais para o top N itens

O primeiro passo para responder qual foi o item mais vendido é verificar na nossa base qual a coluna nós vamos utilizar.

Item

Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category	Sub-Category	Product Name	Sales	Ano	Mes
Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-BO-10001798	Furniture	Bookcases	Bush Somerset Collection Bookcase	261.96	2017	8
Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-CH-10000454	Furniture	Chairs	Hon Deluxe Fabric Upholstered Stacking Chairs,...	731.94	2017	8

Módulo 9 – Criando um gráfico de barras horizontais para o top N itens

Podemos fazer um group by para os itens, utilizando a mesma lógica:

```
itens = base.groupby("Product Name")["Sales"].sum()

itens = itens.reset_index()

itens.head(10)
```

Repare que os itens não estão ordenados

	Product Name	Sales
0	"While you Were Out" Message Book, One Form pe...	25.228
1	#10 Gummed Flap White Envelopes, 100/Box	41.300
2	#10 Self-Seal White Envelopes	108.682
3	#10 White Business Envelopes,4 1/8 x 9 1/2	379.214
4	#10- 4 1/8" x 9 1/2" Recycled Envelopes	286.672
5	#10- 4 1/8" x 9 1/2" Security-Tint Envelopes	146.688
6	#10-4 1/8" x 9 1/2" Premium Diagonal Seam Enve...	176.288
7	#6 3/4 Gummed Flap White Envelopes	71.280
8	1.7 Cubic Foot Compact "Cube" Office Refrigera...	2706.080
9	1/4 Fold Party Design Invitations & White Enve...	49.980

Módulo 9 – Criando um gráfico de barras horizontais para o top N itens

Vamos colocar em ordem decrescente utilizando o **sort_values**.

```
itens = base.groupby("Product Name")["Sales"].sum()

itens = itens.reset_index()

itens = itens.sort_values("Sales",ascending=False)

itens.head(10)
```

Decrescente

	Product Name	Sales
404	Canon imageCLASS 2200 Advanced Copier	61599.824
649	Fellowes PB500 Electric Punch Plastic Comb Bin...	27453.384
444	Cisco TelePresence System EX90 Videoconferenci...	22638.480
785	HON 5400 Series Task Chairs for Big and Tall	21870.576
685	GBC DocuBind TL300 Electric Binding System	19823.479
687	GBC Ibimaster 500 Manual ProClick Binding System	19024.500
804	Hewlett Packard LaserJet 3310 Copier	18839.686
786	HP Designjet T520 Inkjet Large Format Printer ...	18374.895
682	GBC DocuBind P400 Electric Binding System	17965.068
812	High Speed Automatic Electric Letter Opener	17030.312

Módulo 9 – Criando um gráfico de barras horizontais para o top N itens

Uma forma de resolver a quantidade de itens seria fazer apenas para os top 10 itens

```
top_20_itens = itens.head(20)
```

```
top_20_itens
```

	Product Name	Sales
404	Canon imageCLASS 2200 Advanced Copier	61599.8240
649	Fellowes PB500 Electric Punch Plastic Comb Bin...	27453.3840
444	Cisco TelePresence System EX90 Videoconferenci...	22638.4800
785	HON 5400 Series Task Chairs for Big and Tall	21870.5760
685	GBC DocuBind TL300 Electric Binding System	19823.4790
687	GBC Ibimaster 500 Manual ProClick Binding System	19024.5000
804	Hewlett Packard LaserJet 3310 Copier	18839.6860
786	HP Designjet T520 Inkjet Large Format Printer ...	18374.8950
682	GBC DocuBind P400 Electric Binding System	17965.0680
812	High Speed Automatic Electric Letter Opener	17030.3120
984	Lexmark MX611dhe Monochrome Laser Printer	16829.9010
1042	Martin Yale Chadless Opener Electric Letter Op...	16656.2000
894	Ibico EPK-21 Electric Binding System	15875.9160
1350	Riverside Palais Royal Lawyers Bookcase, Royal...	15610.9656
19	3D Systems Cube Printer, 2nd Generation, Magenta	14299.8900
1387	Samsung Galaxy Mega 6.3	13943.6680
155	Apple iPhone 5	12996.6000
368	Bretford Rectangular Conference Table Tops	12995.2915
768	Global Troy Executive Leather Low-Back Tilter	12975.3820
1362	SAFCO Arco Folding Chair	11572.7800

Módulo 9 – Criando um gráfico de barras horizontais para o top N itens

A cada mudança do número de itens seria necessário alterar a variável, para isso não termos esse problema vamos utilizar o top N itens:

n = 10

top_n_itens = **itens**.**head**(n)

top_n_itens

	Product Name	Sales
404	Canon imageCLASS 2200 Advanced Copier	61599.824
649	Fellowes PB500 Electric Punch Plastic Comb Bin...	27453.384
444	Cisco TelePresence System EX90 Videoconferenci...	22638.480
785	HON 5400 Series Task Chairs for Big and Tall	21870.576
685	GBC DocuBind TL300 Electric Binding System	19823.479
687	GBC Ibimaster 500 Manual ProClick Binding System	19024.500
804	Hewlett Packard LaserJet 3310 Copier	18839.686
786	HP Designjet T520 Inkjet Large Format Printer ...	18374.895
682	GBC DocuBind P400 Electric Binding System	17965.068
812	High Speed Automatic Electric Letter Opener	17030.312

Módulo 9 – Criando um gráfico de barras horizontais para o top N itens

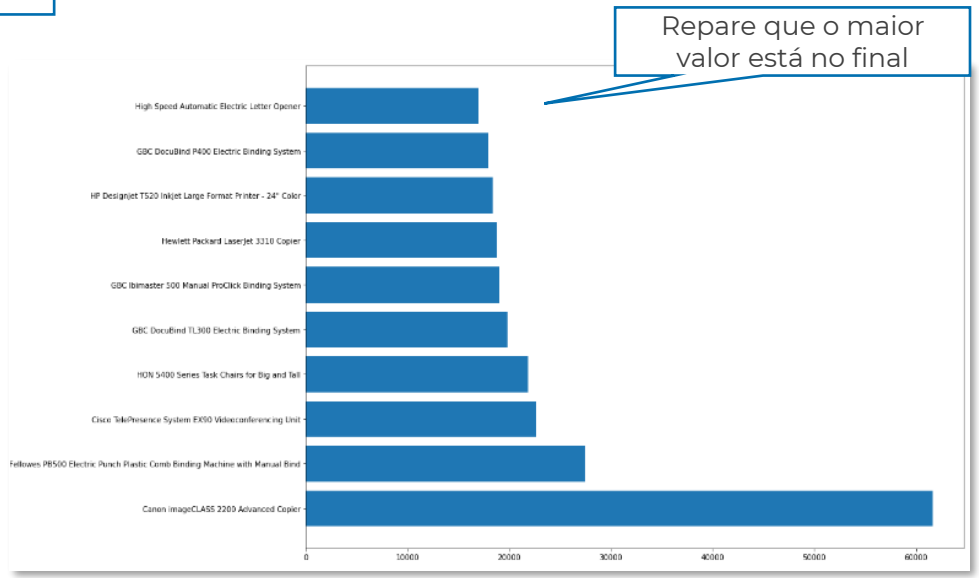
Vamos traçar um gráfico de barras horizontais para mostrar o top 10 itens. Vamos utilizar como base para o nosso gráfico a documentação.

```
fig,ax = plt.subplots(figsize = (16,12))

ax.barh(top_n_itens['Product Name'].values,
        top_n_itens['Sales'].values,
        align='center')

plt.show()
```

Valor do eixo Y

Valor do eixo X

Repare que o maior valor está no final

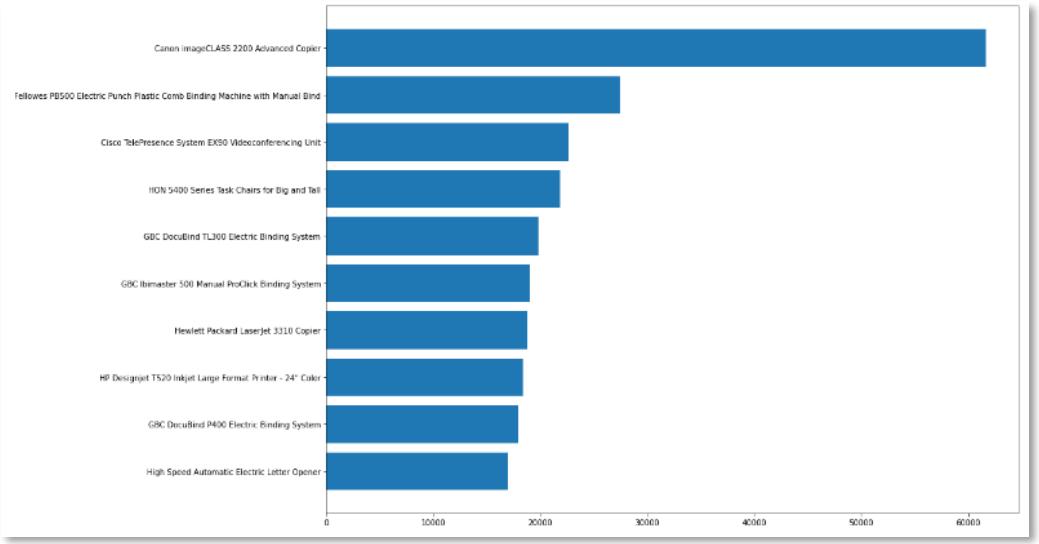


Módulo 9 – Criando um gráfico de barras horizontais para o top N itens

Vamos traçar um gráfico de barras horizontais para mostrar o top 10 itens. Vamos utilizar como base para o nosso gráfico a documentação.

```
fig,ax = plt.subplots(figsize = (16,12))

ax.barh(top_n_itens['Product Name'].values,
        top_n_itens['Sales'].values,
        align='center')

ax.invert_yaxis()

plt.show()
```

Intervertendo o eixo Y



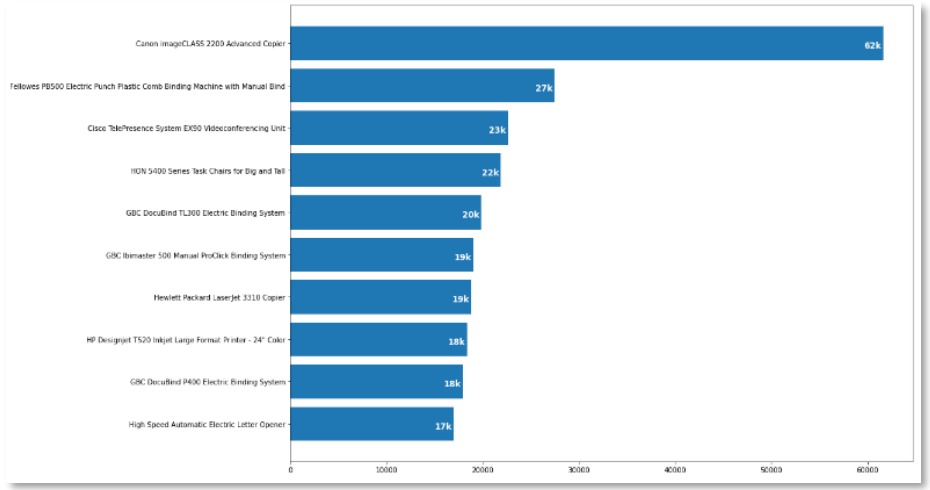
Módulo 9 – Criando um gráfico de barras horizontais para o top N itens

Para melhorar a visualização do gráfico vamos fazer o **annotate**:

Annotate vai depender de **n**

```
for i in np.arange(0,n):

    ax.annotate('{:,.0f}k'.format(top_n_itens['Sales'].values[i]/1000),
                (top_n_itens['Sales'].values[i],i),
                ha="center",
                va="top",
                xytext=(-15,3),
                textcoords="offset points",
                fontsize=12,
                fontweight='bold',
                color="white")
```

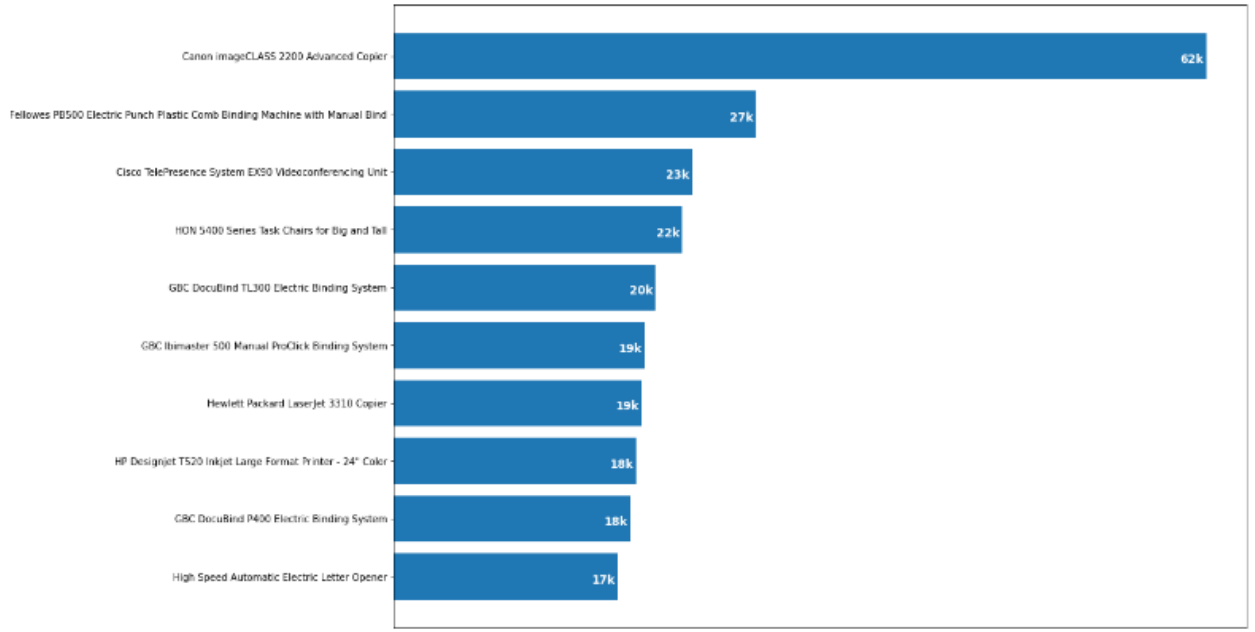
Não pode passar o valor texto, por isso utilizamos i



Módulo 9 – Criando um gráfico de barras horizontais para o top N itens

Vamos retirar os valores do eixo X:

ax.**xaxis.set_visible**(False)



Módulo 9 – Criando um gráfico de barras horizontais para o top N itens

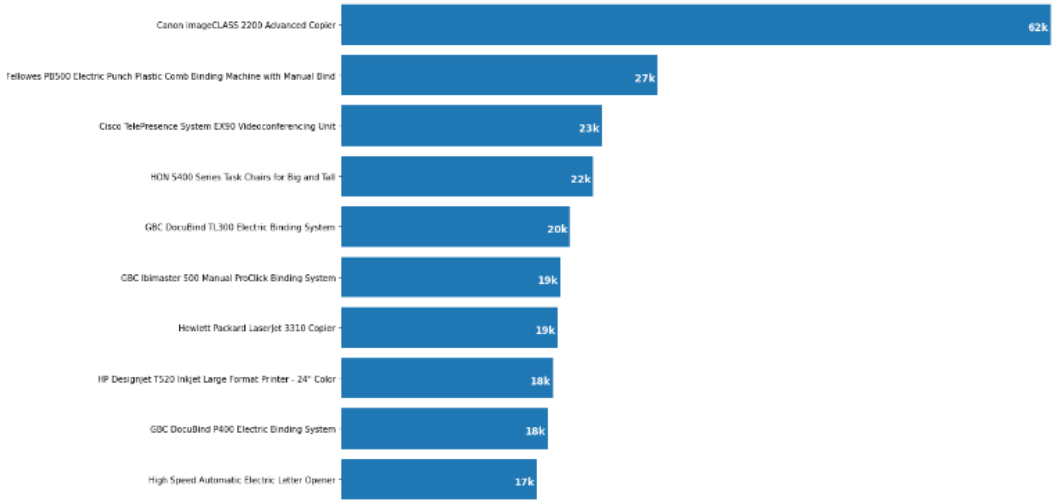
Vamos retirar as bordas:

ax.**spines**['top'].**set_visible**(False)

ax.**spines**['left'].**set_visible**(False)

ax.**spines**['right'].**set_visible**(False)

ax.**spines**['bottom'].**set_visible**(False)



Módulo 9 – Usando o merge para unir 2 bases no pandas

Na nossa apresentação ainda não conseguimos responder o item mais vendido em um determinado ano.

Então para responder essa pergunta primeiro vamos fazer um **groupby** de item e ano

```
itens_ano = base.groupby(["Product Name","Ano"])["Sales"].sum()

itens_ano = itens_ano.reset_index()

itens_ano
```

	Product Name	Ano	Sales
0	"While you Were Out" Message Book, One Form pe...	2018	25.228
1	#10 Gummed Flap White Envelopes, 100/Box	2016	6.608
2	#10 Gummed Flap White Envelopes, 100/Box	2017	24.780
3	#10 Gummed Flap White Envelopes, 100/Box	2018	9.912
4	#10 Self-Seal White Envelopes	2017	86.502
...	...	...	...
5269	invisibleSHIELD by ZAGG Smudge-Free Screen Pro...	2018	205.086
5270	netTALK DUO VoIP Telephone Service	2015	335.936
5271	netTALK DUO VoIP Telephone Service	2016	230.956
5272	netTALK DUO VoIP Telephone Service	2017	377.928
5273	netTALK DUO VoIP Telephone Service	2018	167.968

5274 rows × 3 columns

Módulo 9 – Usando o merge para unir 2 bases no pandas

Agora vamos supor que queremos visualizar alguns dos itens do TOP N.

itens_ano[**itens_ano**["**Product Name**"] == **top_itens**[[2,0]]

Linha

Coluna

```
itens_ano[itens_ano["Product Name"] == top_n_itens.iloc[2,0]]
```

	Product Name	Ano	Sales
1270	Cisco TelePresence System EX90 Videoconferenci...	2015	22638.48

Foi vendido em apenas um ano

Módulo 9 – Usando o merge para unir 2 bases no pandas

Agora vamos criar um gráfico que mostre como foi a nossa venda anual utilizando o **merge** que irá juntar duas bases. Para isso, devemos passar:

- base 1
- base 2
- **how**: forma que iremos fazer essa junção das bases
  - inner: o que tiver em comum entre as 2 bases (base 1 E base 2)
  - outer: tudo o que tiver nas 2 bases (base 1 OU base 2)
  - left: tudo o que tem na **PRIMEIRA** base, juntando com o que tiver na segunda
  - right: tudo o que tem na **SEGUNDA** base, juntando com o que tiver na primeira
- **on**: colunas que vamos usar para fazer a junção da base

Para ficar mais claro vamos fazer um exemplo e criar 2 dataframes:

```
dic1 = {
    "nomes": ['Nome1','Nome2','Nome3'],
    "valores": [1,2,3]
}

base_dic1 = pd.DataFrame(dic1)

dic2 = {
    "nomes": ['Nome1','Nome2','Nome4'],
    "valores": [9,8,7]
}

base_dic2 = pd.DataFrame(dic2)
```

Para ficar mais claro vamos fazer um exemplo e criar 2 dataframes:

```
base_merge = pd.merge(
    base_dic1,
    base_dic2,
    how='outer',
on="nomes" das bases
)
display(base_merge)
```

Primeira base

Segunda base

Tipo de junção que vamos fazer

coluna que vamos usar para fazer essa junção das bases

	nomes	valores_x	valores_y
0	Nome1	1.0	9.0
1	Nome2	2.0	8.0
2	Nome3	3.0	NaN
3	Nome4	NaN	7.0

Módulo 9 – Usando o merge para criar a relação de top N itens pelos anos

Agora vamos unir a nossa base de TOP N com uma base de datas utilizando o **merge**.

Como podemos cruzar a base com todos os anos de 2015 a 2018?

Basta transformar as datas em um **DataFrame** e usar o **merge** para fazer essa união.

df_datas = **pd.DataFrame**([2018,2017,2016,2015])

df_datas.columns = ['Ano']

df_datas

	Ano
0	2018
1	2017
2	2016
3	2015

Módulo 9 – Usando o merge para criar a relação de top N itens pelos anos

Só que pra conseguir unir as bases, precisamos ter uma coluna em comum entre elas

Então podemos, nas duas bases, criar uma coluna chamada uniao (ou qualquer outro nome) com o mesmo valor.

top_n_itens['**uniao**'] = 'unir'

df_datas['**uniao**'] = 'unir'

Todos os valores vão ser 'unir' para conseguir juntar as duas colunas

Deu um erro porque tentamos adicionar uma coluna no TOP N

```
top_n_itens['uniao'] = 'unir'
df_datas['uniao'] = 'unir'
```

```
C:\Users\Pichau\AppData\Local\Temp\ipykernel_16868\2128599823.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  top_n_itens['uniao'] = 'unir'
```

Módulo 9 – Usando o merge para criar a relação de top N itens pelos anos

Agora que temos uma coluna que conseguimos relacionar, podemos fazer o merge das bases.

```
produto_ano = pd.merge(
    top_n_itens,
    df_datas,
    how='outer
    on="uniao"
)
produto_ano.head()
```

Primeira base

Segunda base

Tipo de junção que vamos fazer

coluna que vamos usar para fazer essa junção das bases

	Product Name	Sales	uniao	Ano
0	Canon imageCLASS 2200 Advanced Copier	61599.824	unir	2018
1	Canon imageCLASS 2200 Advanced Copier	61599.824	unir	2017
2	Canon imageCLASS 2200 Advanced Copier	61599.824	unir	2016
3	Canon imageCLASS 2200 Advanced Copier	61599.824	unir	2015
4	Fellowes PB500 Electric Punch Plastic Comb Bin...	27453.384	unir	2018

Módulo 9 – Usando o merge para criar a relação de top N itens pelos anos

Agora que temos uma coluna que conseguimos relacionar, podemos fazer o merge das bases.

Para fazer isso podemos utilizar o **itens_ano**

display(**itens_ano**)

	Product Name	Ano	Sales
0	"While you Were Out" Message Book, One Form pe...	2018	25.228
1	#10 Gummed Flap White Envelopes, 100/Box	2016	6.608
2	#10 Gummed Flap White Envelopes, 100/Box	2017	24.780
3	#10 Gummed Flap White Envelopes, 100/Box	2018	9.912
4	#10 Self-Seal White Envelopes	2017	86.502
...	...	...	...
5269	invisibleSHIELD by ZAGG Smudge-Free Screen Pro...	2018	205.086
5270	netTALK DUO VoIP Telephone Service	2015	335.936
5271	netTALK DUO VoIP Telephone Service	2016	230.956
5272	netTALK DUO VoIP Telephone Service	2017	377.928
5273	netTALK DUO VoIP Telephone Service	2018	167.968

5274 rows × 3 columns

Módulo 9 – Usando o merge para criar a relação de top N itens pelos anos

Então, vamos fazer novamente o merge para colocar a coluna de vendas

```
itens_ano_grafico = pd.merge(
    produto_ano,
    itens_ano,
    how='left',
    on=['Product Name','Ano']
)
itens_ano_grafico.head()
```

Primeira base

Segunda base

Para manter a primeira base

colunas que vamos usar para fazer essa junção das bases

	Product Name	Sales_x	uniao	Ano	Sales_y
0	Canon imageCLASS 2200 Advanced Copier	61599.824	unir	2018	35699.898
1	Canon imageCLASS 2200 Advanced Copier	61599.824	unir	2017	25899.926
2	Canon imageCLASS 2200 Advanced Copier	61599.824	unir	2016	NaN
3	Canon imageCLASS 2200 Advanced Copier	61599.824	unir	2015	NaN
4	Fellowes PB500 Electric Punch Plastic Comb Bin...	27453.384	unir	2018	7371.742

Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

Vamos começar a criar o gráfico baseado n e DataFrame que acabamos de criar:

Os valore de y nós vamos utilizar:

itens_ano_grafico[**itens_ano_grafico**.**Ano** == 2018][**'Product Name'**].**values**

```
# Vamos criar o gráfico baseado nesse DataFrame que acabamos de criar
itens_ano_grafico[itens_ano_grafico.Ano == 2018]['Product Name'].values
```

```
array(['Canon imageCLASS 2200 Advanced Copier',
       'Fellowes PB500 Electric Punch Plastic Comb Binding Machine with Manual Bind',
       'Cisco TelePresence System EX90 Videoconferencing Unit',
       'HON 5400 Series Task Chairs for Big and Tall',
       'GBC DocuBind TL300 Electric Binding System',
       'GBC Ibimaster 500 Manual ProClick Binding System',
       'Hewlett Packard LaserJet 3310 Copier',
       'HP Designjet T520 Inkjet Large Format Printer - 24" Color',
       'GBC DocuBind P400 Electric Binding System',
       'High Speed Automatic Electric Letter Opener'], dtype=object)
```

Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

E para os valores de y nós não podemos utilizar a coluna Sales_x porque esses valores são as somas da venda, para o nosso gráfico vamos utilizar o Sales_y que é a venda ano a ano.

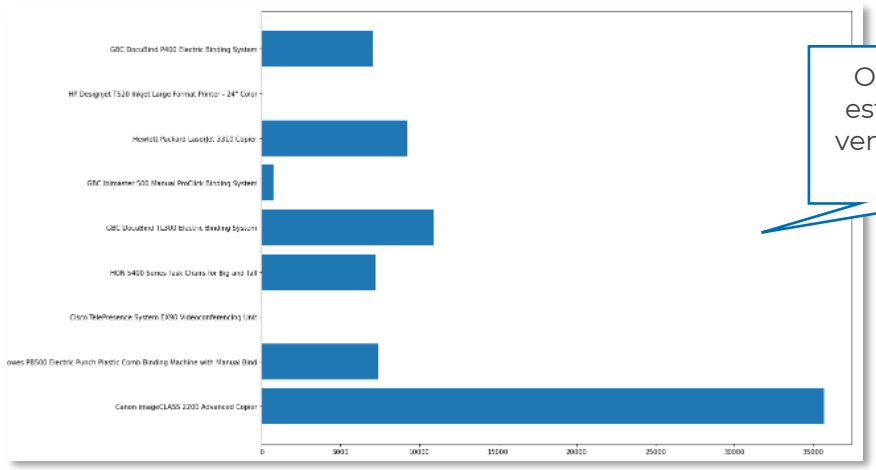
itens_ano_grafico[**itens_ano_grafico**.**Ano** == 2018][**'Sales_y**'].**values**

	Product Name	Sales_x	uniao	Ano	Sales_y
0	Canon imageCLASS 2200 Advanced Copier	61599.824	unir	2018	35699.898
1	Canon imageCLASS 2200 Advanced Copier	61599.824	unir	2017	25899.926
2	Canon imageCLASS 2200 Advanced Copier	61599.824	unir	2016	NaN
3	Canon imageCLASS 2200 Advanced Copier	61599.824	unir	2015	NaN
4	Fellowes PB500 Electric Punch Plastic Comb Bin...	27453.384	unir	2018	7371.742

Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

Agora colocando essas informações no código do gráfico:

```
fig,ax = plt.subplots(figsize = (16,12))

ax.barh(itens_ano_grafico[itens_ano_grafico.Ano == 2018]['Product Name'].values,
        itens_ano_grafico[itens_ano_grafico.Ano == 2018]['Sales_y'].values,
        align='center'
        )

plt.show()
```

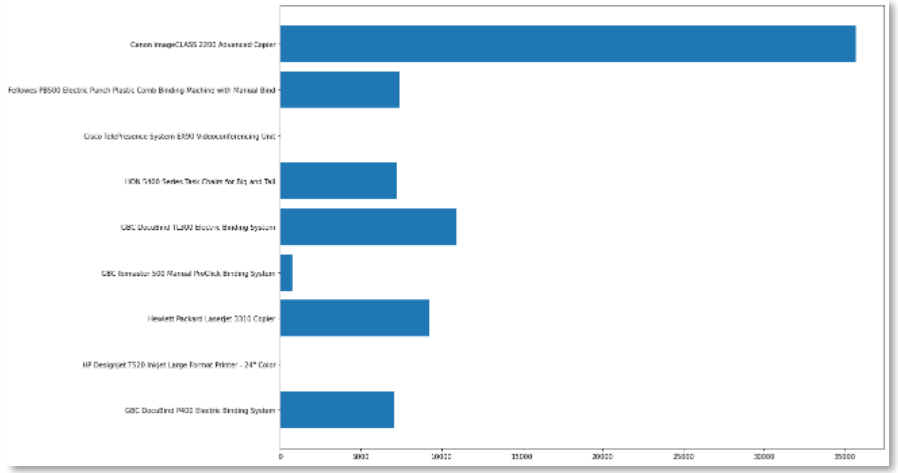


Observe que ele está mostrando a venda apenas para 2018

Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

Vamos fazer a inversão do gráfico que fizemos anteriormente:

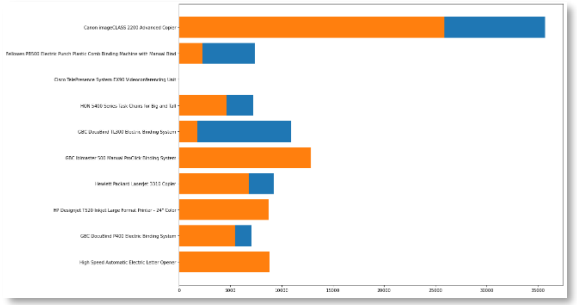
```
fig,ax = plt.subplots(figsize = (16,12))

ax.barh(itens_ano_grafico[itens_ano_grafico.Ano == 2018]['Product Name'].values,
        itens_ano_grafico[itens_ano_grafico.Ano == 2018]['Sales_y'].values,
        align='center'
        )

ax.invert_yaxis()

plt.show()
```



Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

Vamos aplicar o mesmo código para 2017:

```
fig,ax = plt.subplots(figsize = (16,12))

ax.barh(itens_ano_grafico[itens_ano_grafico.Ano == 2017]['Product Name'].values,
        itens_ano_grafico[itens_ano_grafico.Ano == 2017]['Sales_y'].values,
align='center')

ax.barh(itens_ano_grafico[itens_ano_grafico.Ano == 2018]['Product Name'].values,
itens_ano_grafico[itens_ano_grafico.Ano == 2018]['Sales_y'].values, align='center')

ax.invert_yaxis()

plt.show()
```



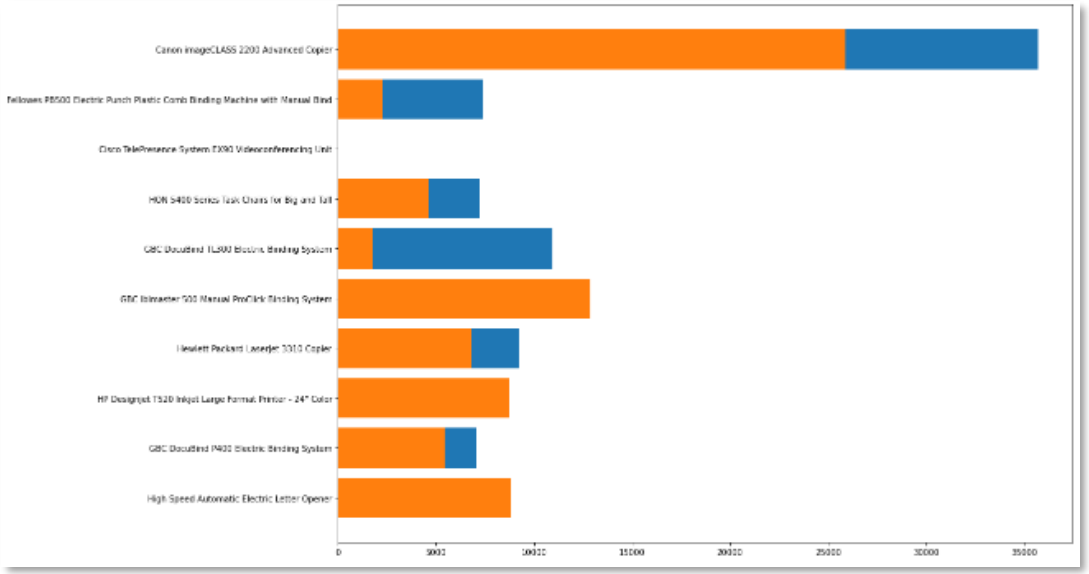
Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

Agora vamos deslocar a coluna igual fizemos para o gráfico de barras, só que agora vamos utilizar a variável com o comprimento.

Vamos utilizar:

hgt = 0.2

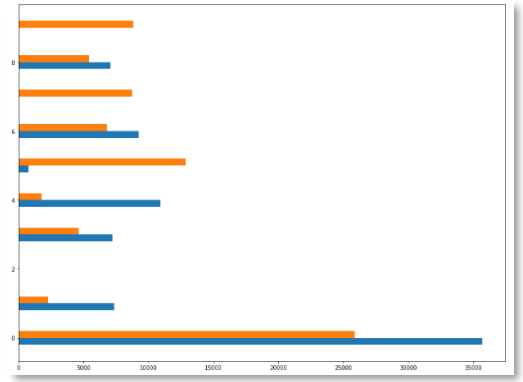
Atenção: como estamos trabalhando com o texto é necessário transformar ele em uma lista de valores.



Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

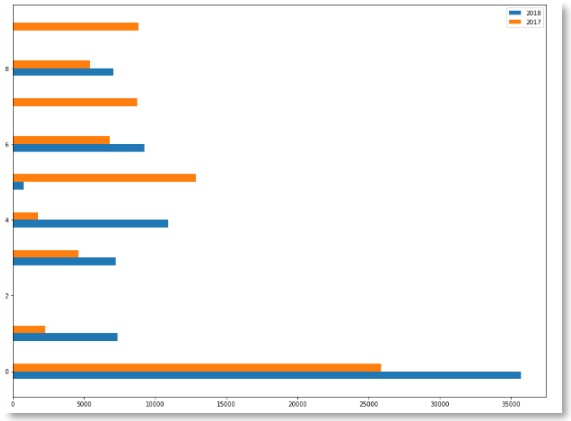
Utilizando o comprimento para ajustar:

```
fig,ax = plt.subplots(figsize = (16,12))
hgt = 0.2

ax.barh(np.arange(0,qtd_itens+hgt/2,
itens_ano_grafico[itens_ano_grafico.Ano==2017]['Sales_y'].values, align='center')
ax.barh(np.arange(0,qtd_itens)-
hgt/2,itens_ano_grafico[itens_ano_grafico.Ano==2018]['Sales_y'].values,
align='center')

ax.invert_yaxis()
plt.show()
```



Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

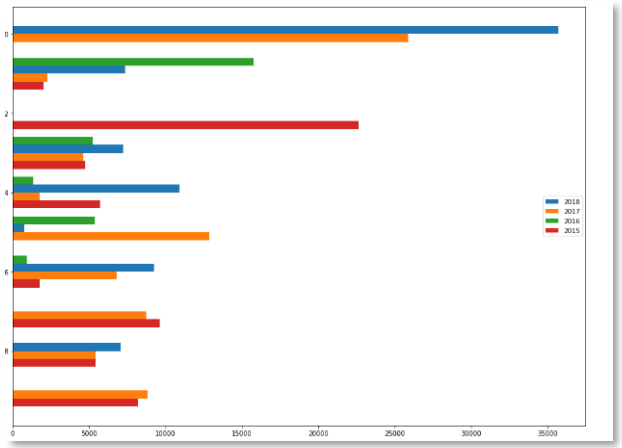
```
fig,ax = plt.subplots(figsize = (16,12))
hgt = 0.2

ax.barh(np.arange(0,qtd_itens)+hgt/2,itens_ano_grafico[itens_ano_grafico.Ano ==
2017]['Sales_y'].values, align='center', label='2017')

ax.barh(np.arange(0,qtd_itens)-hgt/2,itens_ano_grafico[itens_ano_grafico.Ano ==
2018]['Sales_y'].values, align='center', label='2018'))

ax.invert_yaxis()

ax.legend()

plt.show()
```



Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

Vamos incluir no código anterior as informações de 2016 e 2015:

```
ax.barh(np.arange(0,qtd_itens)-hgt-hgt/2,itens_ano_grafico[itens_ano_grafico.Ano == 2016]['Sales_y'].values, align='center', label='2016')

ax.barh(np.arange(0,qtd_itens)-hgt-hgt/2,itens_ano_grafico[itens_ano_grafico.Ano == 2015]['Sales_y'].values, align='center', label='2015'))

ax.invert_yaxis()

ax.legend()

plt.show()
```



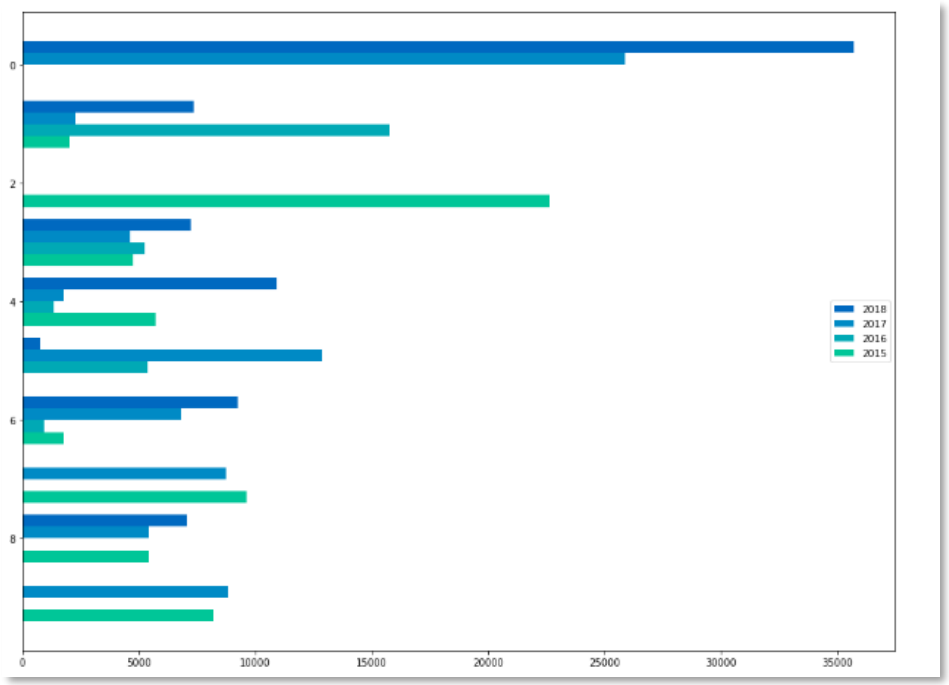
Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

Vamos padronizar as cores:

```
ax.barh(np.arange(0,qtd_itens)-hgt-hgt/2,itens_ano_grafico[itens_ano_grafico.Ano == 2018].Sales_y.values, align='center', height=hgt, label=2018,color="#0069c0")

ax.barh(np.arange(0,qtd_itens)-hgt/2,itens_ano_grafico[itens_ano_grafico.Ano == 2017].Sales_y.values, align='center',height=hgt,label=2017,color="#008ac5")

ax.barh(np.arange(0,qtd_itens)+hgt/2,itens_ano_grafico[itens_ano_grafico.Ano == 2016].Sales_y.values, align='center',height=hgt,label=2016,color="#00a9b5")

ax.barh(np.arange(0,qtd_itens)+hgt+hgt/2,itens_ano_grafico[itens_ano_grafico.Ano == 2015].Sales_y.values, align='center',height=hgt,label=2015,color="#00c698"
```

Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

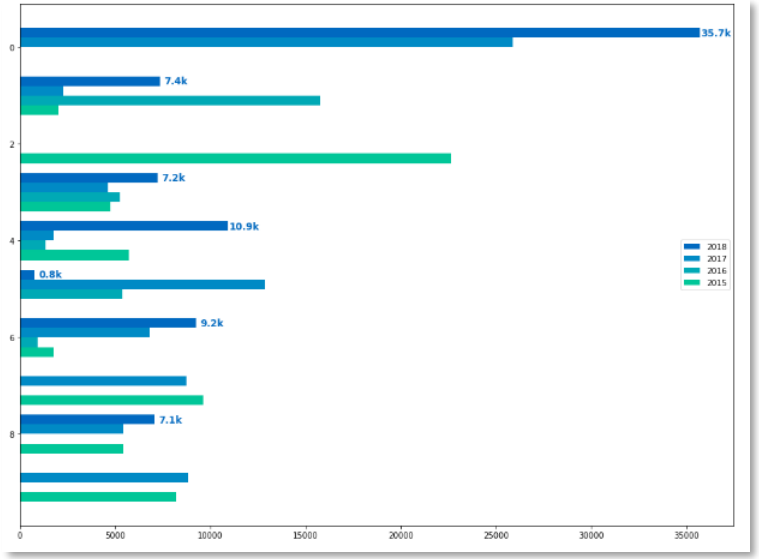
Padronizando as cores temos o gráfico abaixo:



Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

Vamos aplicar o annotate para 2018 no nosso código:

```
for i in np.arange(0,qtd_itens):

        ax.annotate('{:,.1f}k'.format(itens_ano_grafico[itens_ano_grafico.Ano==2018].Sales_y.values[i]/1000),
        (itens_ano_grafico[itens_ano_grafico.Ano==2018].Sales_y.values[i],i-hgt-hgt/2),
        ha="center", va="top", xytext=(20,6), textcoords="offset points", fontsize=12, fontweight='bold', color="#0069c0")
```



Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

Vamos com os valores do nosso eixo x no código:

```
ax.yaxis.set_ticks(np.arange(0,qtd_itens))
```

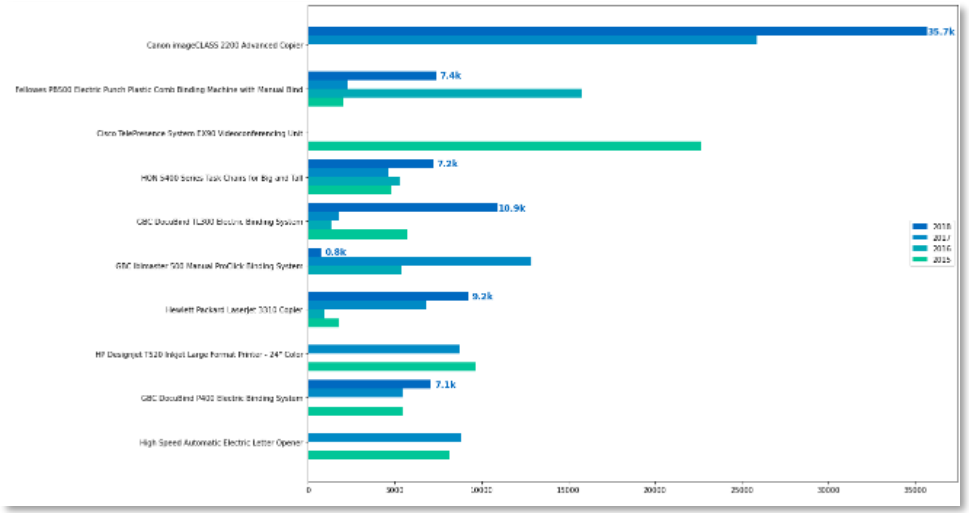
Ajustando os valores de x

```
ax.set_yticklabels(top_n_itens["Product Name"])
```

Colocando o rótulo no eixo

```
ax.tick_params(axis='y',labelsize=10)
```

Melhorando o visual do eixo x



Módulo 9 – Criando o gráfico de barras horizontais do top N itens pelos anos

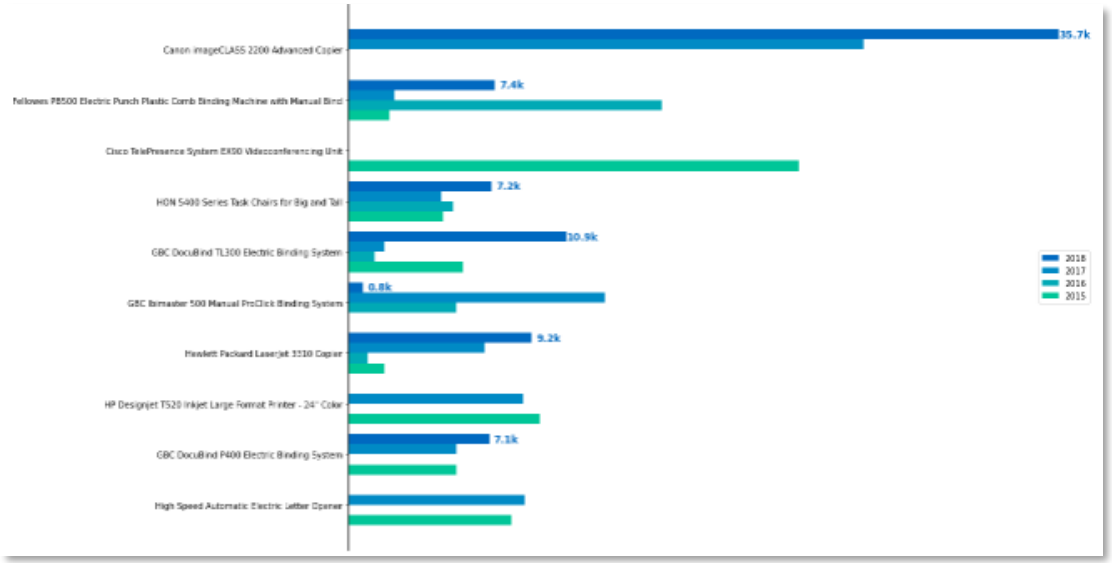
Para finalizar vamos retirar as bordas:

ax.**xaxis.set_visible**(False)

ax.**spines**['top'].**set_visible**(False)

ax.**spines**['right'].**set_visible**(False)

ax.**spines**['bottom'].**set_visible**(False)



Para fechar esse projeto vamos colocar todos os gráficos um embaixo do outro para ficar mais resumido e ficar mais fácil de achar os gráficos.

Então apenas vamos copiar os códigos dos gráficos:

- Vendas por ano
- Vendas por mês
- Vendas por categoria

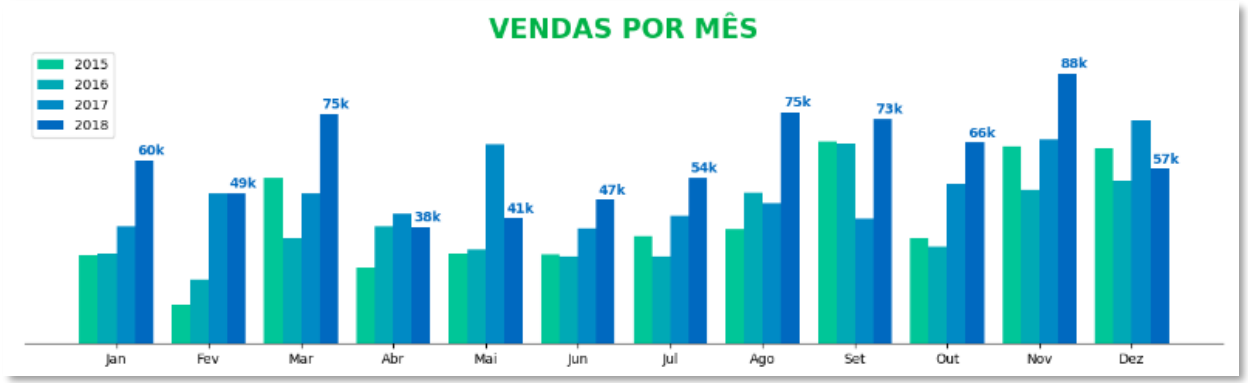
- Vendas por ano

```
fig,ax = plt.subplots(
            figsize = (9,4)
        )

soma_ano = base.groupby("Ano")["Sales"].sum()

ax.bar(soma_ano.index,soma_ano.values,
       color="#84ba5b"
      )

ax.set_title("VENDAS POR ANO",
             fontsize=20,
             fontweight='bold',
             color='#00b247'
            )

ax.xaxis.set_ticks([2015,2016,2017,2018])
ax.tick_params(axis='x',labelsize=14)

for i in np.arange(0,4):
    ax.annotate('{:,.0f}'.format(soma_ano.values[i]).replace(',', '.'),
                (soma_ano.index[i],soma_ano.values[i]),
                ha="center",
                va="top",
                xytext=(0,+15),
                textcoords="offset points",
                fontsize=14,
                fontweight='bold',
                color="green"
            )

ax.set_yticks(np.array([0,790000]))
ax.yaxis.set_visible(False)

ax.spines['top'].set_visible(False)
ax.spines['left'].set_visible(False)
ax.spines['right'].set_visible(False)

plt.show()
```

Módulo 9 – Concluindo o projeto e respondendo as informações do negócio

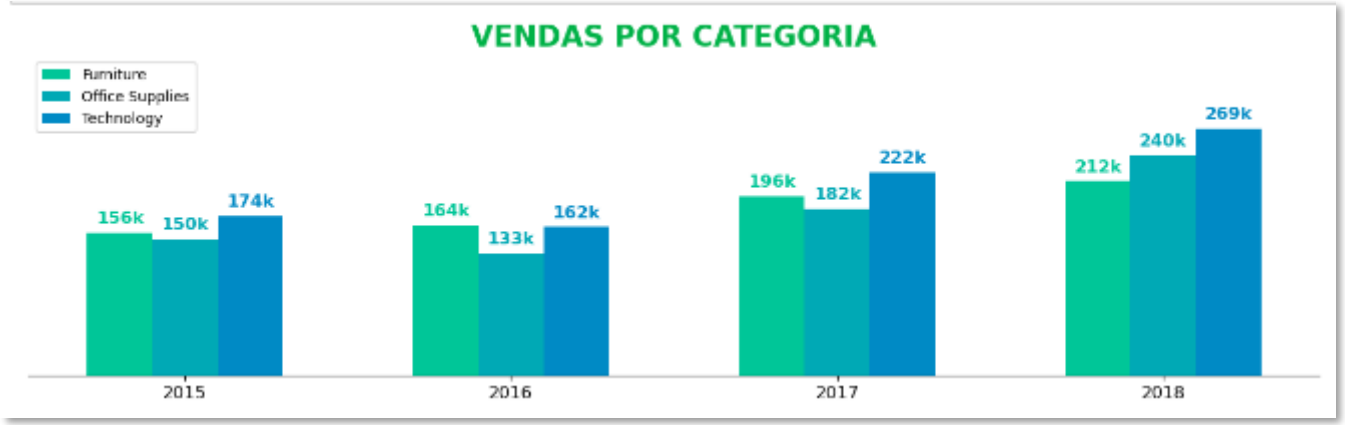
- Vendas por mês

```
fig,ax = plt.subplots(
            figsize = (16,4)
        )

ax.set_title("VENDAS POR MÊS",
             fontsize=20,
             fontweight='bold',
             color='#00b247'
            )

wid = 0.2 # Tamanho das barras

ax.bar(soma_mes[soma_mes.Ano == 2015].Mes.values-wid-wid/2,
       soma_mes[soma_mes.Ano == 2015].Sales.values,
       color="#00c698", # Cor das barras
       label=2015, # Label dos dados
       width=wid # Largura das barras
       )
ax.bar(soma_mes[soma_mes.Ano == 2016].Mes.values-wid/2,
       soma_mes[soma_mes.Ano == 2016].Sales.values,
       color="#00a9b5",
       label=2016,
       width=wid
      )

# Adicionando o restante das barras
ax.bar(soma_mes[soma_mes.Ano == 2017].Mes.values+wid/2,
       soma_mes[soma_mes.Ano == 2017].Sales.values,
       color="#0088ac5",
       label=2017,
       width=wid
      )
ax.bar(soma_mes[soma_mes.Ano == 2018].Mes.values+wid+wid/2,
       soma_mes[soma_mes.Ano == 2018].Sales.values,
       color="#0069c0",
       label=2018,
       width=wid
      )
ax.legend()

# Adicionando o rótulo dos dados para 2018
for i in np.arange(0,12):
    ax.annotate('{:,.0f}k'.format(soma_mes[soma_mes.Ano == 2018].Sales.values[i]/1000),
                (soma_mes[soma_mes.Ano == 2018].Mes.values[i]+wid+wid/2,soma_mes[soma_mes.Ano == 2018].Sales.values[i]),
                ha="center",
                va="top",
                xytext=(5,12),
                textcoords="offset points",
                fontsize=10,
                fontweight='bold',
                color='#0069c0'
            )

ax.set_yticks(np.array([0,97000]))
ax.yaxis.set_visible(False)

ax.spines['top'].set_visible(False)
ax.spines['left'].set_visible(False)
ax.spines['right'].set_visible(False)

ax.xaxis.set_ticks(np.arange(1,13))
ax.set_xticklabels(['Jan','Fev','Mar','Abr','Mai','Jun','Jul','Ago','Set','Out','Nov','Dez'])
ax.tick_params(axis='x',labelsize=10)

plt.show()
```



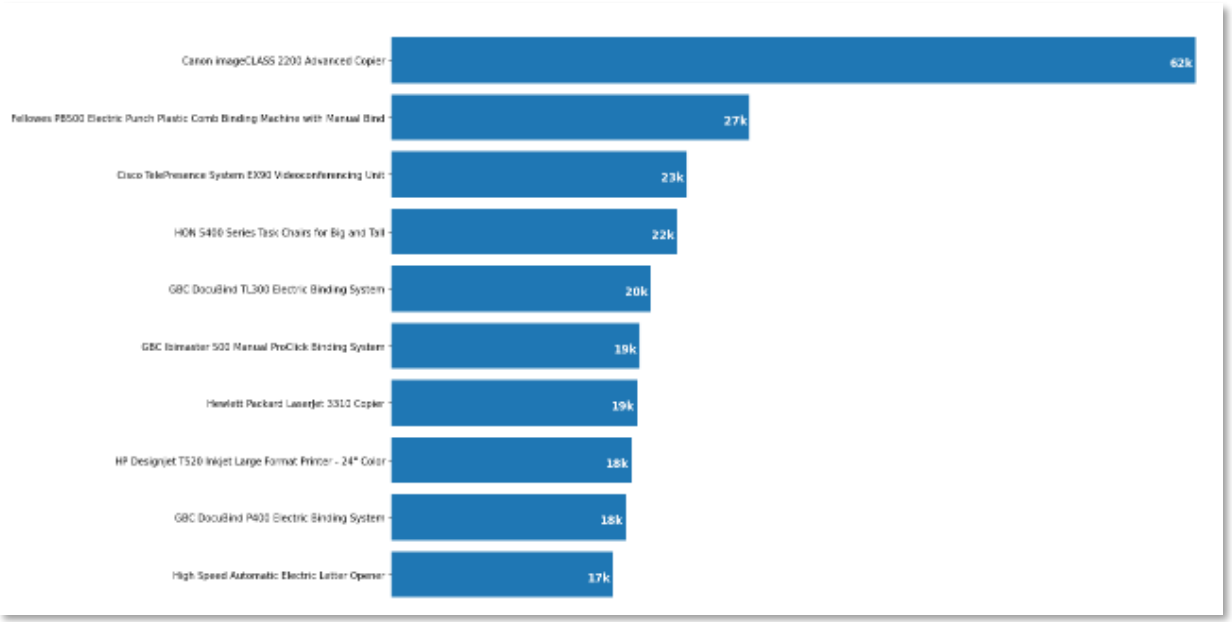
- Vendas por categoria

```
fig,ax = plt.subplots(
            figsize = (16,4)
        )

ax.set_title("VENDAS POR CATEGORIA",
             fontsize=20,
             fontweight='bold',
             color='#00b247'
            )

wid = 0.2

ax.bar(categoria[categoria.Category == 'Furniture'].Ano.values-wid,
       categoria[categoria.Category == 'Furniture'].Sales.values,
       color="#00c698",
       label='Furniture',
       width=wid
      )

ax.bar(categoria[categoria.Category == 'Office Supplies'].Ano.values,
       categoria[categoria.Category == 'Office Supplies'].Sales.values,
       color="#00a9b5",
       label='Office Supplies',
       width=wid
      )

ax.bar(categoria[categoria.Category == 'Technology'].Ano.values+wid,
       categoria[categoria.Category == 'Technology'].Sales.values,
       color="#008ac5",
       label='Technology',
       width=wid
      )

ax.legend()

for i in np.arange(0,4):
    ax.annotate('{:,.0f}k'.format(categoria[categoria.Category == 'Furniture'].Sales.values[i]/1000),
                (categoria[categoria.Category == 'Furniture'].Ano.values[i]-wid,categoria[categoria.Category == 'Furniture'].Sal
                ha="center", va="top", xytext=(0,15), textcoords="offset points",
                fontsize=12, fontweight='bold', color="#00c698")
    ax.annotate('{:,.0f}k'.format(categoria[categoria.Category == 'Office Supplies'].Sales.values[i]/1000),
                (categoria[categoria.Category == 'Office Supplies'].Ano.values[i],categoria[categoria.Category == 'Office Suppli
                ha="center", va="top", xytext=(0,15), textcoords="offset points",
                fontsize=12, fontweight='bold', color="#00a9b5")
    ax.annotate('{:,.0f}k'.format(categoria[categoria.Category == 'Technology'].Sales.values[i]/1000),
                (categoria[categoria.Category == 'Technology'].Ano.values[i]+wid,categoria[categoria.Category == 'Technology'].S
                ha="center", va="top", xytext=(0,15), textcoords="offset points",
                fontsize=12, fontweight='bold', color="#008ac5")

ax.set_yticks(np.array([0,350000]))
ax.yaxis.set_visible(False)

ax.spines['top'].set_visible(False)
ax.spines['left'].set_visible(False)
ax.spines['right'].set_visible(False)

ax.xaxis.set_ticks(np.arange(2015,2019))
ax.tick_params(axis='x',labelsize=12)

plt.show()
```



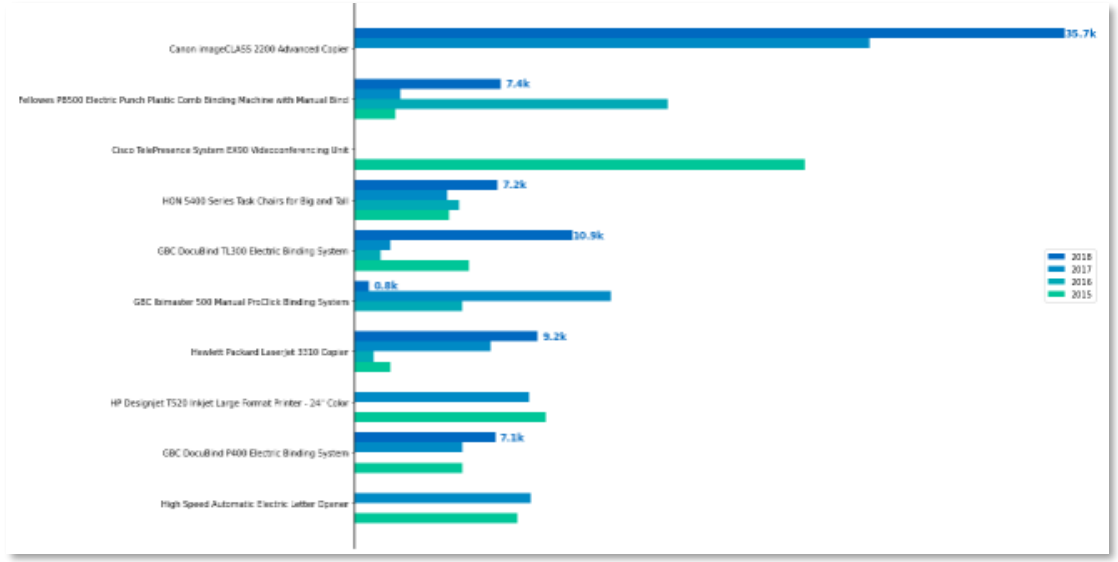
- TOP itens

```
# Vamos traçar um gráfico de barras horizontais para mostrar o top itens
fig,ax = plt.subplots(
            figsize = (16,12)
        )

ax.barh(top_n_itens['Product Name'].values,
        top_n_itens['Sales'].values,
        align='center')

ax.invert_yaxis()

for i in np.arange(0,n):
    ax.annotate('{:,.0f}k'.format(top_n_itens['Sales'].values[i]/1000),
                (top_n_itens['Sales'].values[i],i),
                ha="center", va="top", xytext=(-15,3), textcoords="offset points",
                fontsize=12, fontweight='bold', color="white")

ax.xaxis.set_visible(False)

ax.spines['top'].set_visible(False)
ax.spines['left'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['bottom'].set_visible(False)

plt.show()
```



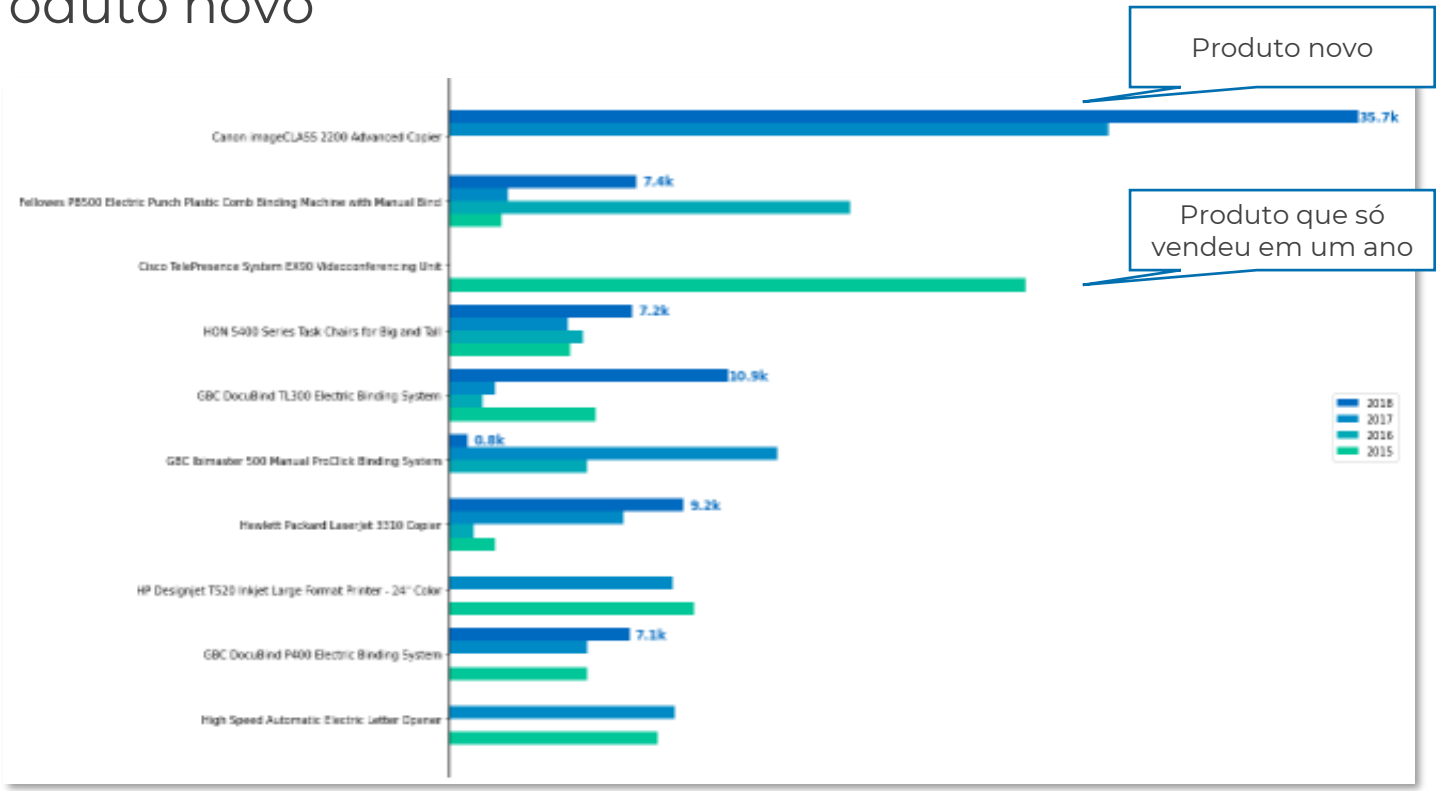
- Itens x anos

```
# Vamos criar o gráfico baseado nesse DataFrame que acabamos de criar
fig,ax = plt.subplots(
         figsize = (16,12)
        )

hgt = 0.2

qtd_itens = len(top_n_itens)

ax.barh(np.arange(0,qtd_itens)-hgt-hgt/2,
        itens_ano_grafico[itens_ano_grafico.Ano == 2018].Sales_y.values,
        align='center',
        height=hgt,
        label=2018,
        color="#0069c0"
       )
ax.barh(np.arange(0,qtd_itens)-hgt/2, #Valores de y
        itens_ano_grafico[itens_ano_grafico.Ano == 2017].Sales_y.values,
        align='center',height=hgt,label=2017,color="#008ac5")
ax.barh(np.arange(0,qtd_itens)+hgt/2, #Valores de y
        itens_ano_grafico[itens_ano_grafico.Ano == 2016].Sales_y.values,
        align='center',height=hgt,label=2016,color="#00a9b5")
ax.barh(np.arange(0,qtd_itens)+hgt+hgt/2, #Valores de y
        itens_ano_grafico[itens_ano_grafico.Ano == 2015].Sales_y.values,
        align='center',height=hgt,label=2015,color="#00c698")

ax.invert_yaxis()
ax.legend()

for i in np.arange(0,qtd_itens):
    ax.annotate('{:,.1f}k'.format(itens_ano_grafico[itens_ano_grafico.Ano == 2018].Sales_y.values[i]/1000),
                (itens_ano_grafico[itens_ano_grafico.Ano == 2018].Sales_y.values[i],i-hgt-hgt/2),
                ha="center", va="top", xytext=(20,6), textcoords="offset points",
                fontsize=12, fontweight='bold', color="#0069c0")

ax.yaxis.set_ticks(np.arange(0,qtd_itens))
ax.set_yticklabels(top_n_itens["Product Name"])
ax.tick_params(axis='y',labelsize=10)

ax.xaxis.set_visible(False)

ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['bottom'].set_visible(False)

plt.show()
```



Módulo 9 – Concluindo o projeto e respondendo as informações do negócio

- Podemos verificar que um produto só vendeu em um ano, também é possível observar um produto novo

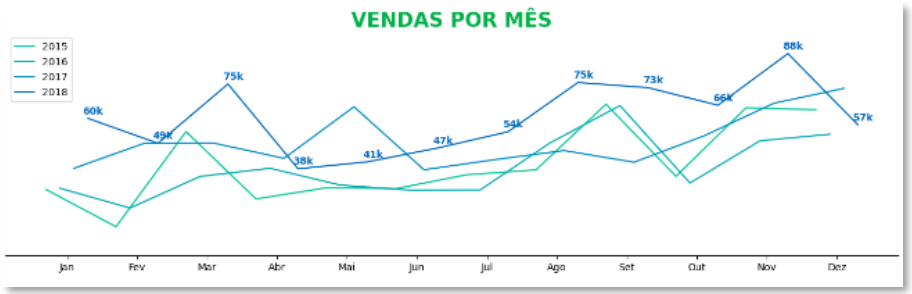


Para trocar um gráfico de barra para um linha, basta substituir o termo **bar** do código por **plot**.

Vamos **excluir a largura do gráfico**, já que o gráfico de linha não possui largura.

Aplicando essas mudanças em uma parte do nosso código:

```
ax.plot(soma_mes[soma_mes.Ano == 2018].Mes.values,
        soma_mes[soma_mes.Ano == 2018].Sales.values,
        color="#0069c0",
        label=2018)
```



Nessa aula vamos aprender como colocar a imagem no PowerPoint.

Vamos utilizar o modelo de PowerPoint da Hashtag que está na descrição do vídeo.

Para salvar um gráfico como imagem temos que colocar no código a função:

plt.savefig('Título')

O arquivo ficará salvo na pasta em que o seu arquivo do Jupyter está armazenado.

Depois de salvar a imagem e importá-la para o PowerPoint o fundo branco do gráfico não faz sentido, já que o nosso fundo é cinza.

Para resolver esse problema vamos alterar o código para:

plt.savefig('Vendas por ano', transparent = True)

O arquivo antigo será substituído pelo atual após rodar o comando.

Repare como melhorou a visualização da imagem

Repare como melhorou a visualização da imagem, vamos fazer esse mesmo processo para todas as imagens.

Quando vamos incluir o gráfico de top itens na apresentação ele vai cortar, mas para resolver isso basta inserir no código o comando **plt.tight_layout** ele vai fazer com que o gráfico se ajuste, por exemplo :

plt.tight_layout()

plt.savefig('TOP ITENS')

Módulo 9 – Corrigindo o erro na transformação da data

Durante o nosso projeto quando falamos de transformação de datas, o Python considerou em algumas datas o mês e o ano de forma correta, enquanto em outras ele considerou de forma invertida.

Repare que na nossa base o formato da data é DD/MM/AAAA

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Product ID	Category
0	1	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-BO-10001798	Furniture
1	2	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR-CH-10000454	Furniture

Módulo 9 – Corrigindo o erro na transformação da data

Vamos fazer a transformação das datas criando uma nova coluna.

```
base["Order Date 2"] = pd.to_datetime(base["Order Date"])
```

```
base[["Order Date","Order Date 2"]]
```

Aqui o Pyhon entende como: mês/dia/ano

```
base["Order Date 2"] = pd.to_datetime(base["Order Date"])
base[["Order Date","Order Date 2"]]
```

	Order Date	Order Date 2
0	2017-08-11	2017-08-11
1	2017-08-11	2017-08-11
2	2017-12-06	2017-12-06
3	2016-11-10	2016-11-10
4	2016-11-10	2016-11-10
...	...	...
9795	2017-05-21	2017-05-21
9796	2016-12-01	2016-12-01
9797	2016-12-01	2016-12-01
9798	2016-12-01	2016-12-01
9799	2016-12-01	2016-12-01

9800 rows × 2 columns

Aqui o Pyhon entende como: dia/ mês/ano

Módulo 9 – Corrigindo o erro na transformação da data

Para confirmar esse erro podemos criar uma coluna com dia e outra com o mês.

base['Dia']=base['Order Date 2'].dt.day

base['Mes']=base['Order Date 2'].dt.month

base[["Order Date","Order Date 2","Dia","Mes"]]

```
base['Dia']=base['Order Date 2'].dt.day
base['Mes']=base['Order Date 2'].dt.month
base[["Order Date","Order Date 2","Dia","Mes"]]
```

	Order Date	Order Date 2	Dia	Mes
0	2017-08-11	2017-08-11	11	8
1	2017-08-11	2017-08-11	11	8
2	2017-12-06	2017-12-06	6	12
3	2016-11-10	2016-11-10	10	11
4	2016-11-10	2016-11-10	10	11
...	...	...	...	...
9795	2017-05-21	2017-05-21	21	5
9796	2016-12-01	2016-12-01	1	12
9797	2016-12-01	2016-12-01	1	12
9798	2016-12-01	2016-12-01	1	12
9799	2016-12-01	2016-12-01	1	12

9800 rows × 4 columns

O 8 foi considerado como mês

O 21 foi considerado como dia

Módulo 9 – Corrigindo o erro na transformação da data

Para resolver esse problema podemos utilizar o **format** e dizer o formato da nossa base.

```
base["Order Date 2"] = pd.to_datetime(base["Order Date"],format ='$d/$m/$Y')

base['Dia']=base['Order Date 2'].dt.day

base['Mes']=base['Order Date 2'].dt.month

base[["Order Date","Order Date 2","Dia","Mes"]]
```

```
base["Order Date 2"] = pd.to_datetime(base["Order Date"],format='$d/$m/$Y')
base['Dia']=base['Order Date 2'].dt.day
base['Mes']=base['Order Date 2'].dt.month
base[["Order Date","Order Date 2","Dia","Mes"]]
```

	Order Date	Order Date 2	Dia	Mes
0	2017-08-11	2017-08-11	11	8
1	2017-08-11	2017-08-11	11	8
2	2017-12-06	2017-12-06	6	12
3	2016-11-10	2016-11-10	10	11
4	2016-11-10	2016-11-10	10	11
...	...	...	...	...
9795	2017-05-21	2017-05-21	21	5
9796	2016-12-01	2016-12-01	1	12
9797	2016-12-01	2016-12-01	1	12
9798	2016-12-01	2016-12-01	1	12
9799	2016-12-01	2016-12-01	1	12

9800 rows × 4 columns

O 11 foi considerado como dia

O 21 foi considerado como dia

MÓDULO 10

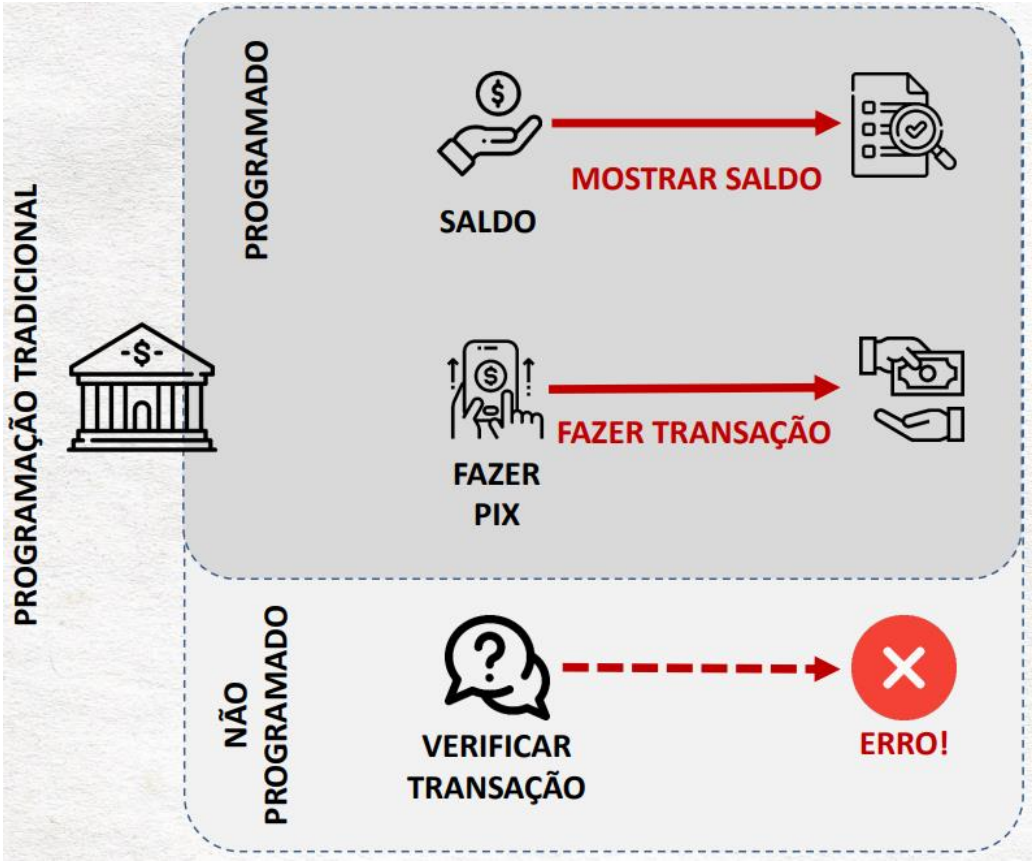
Introdução ao aprendizado de máquinas

Módulo 10 – O que é aprendizado de máquinas (Machine Learning)?

É o campo de estudo que dá aos computadores a capacidade de aprender sem serem explicitamente programados.

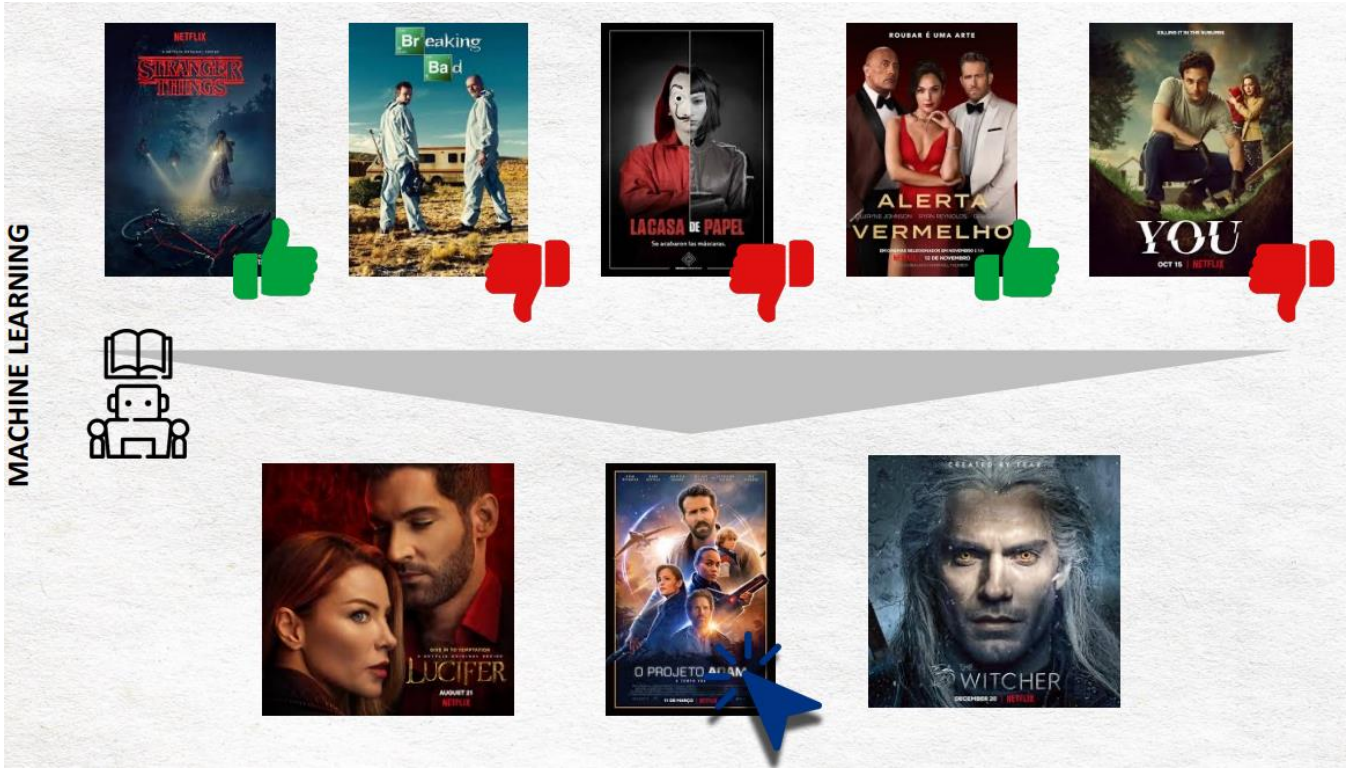
Na programação tradicional, o programa fará apenas aquilo para o qual foi estritamente programado.

Para uma dada entrada, é possível saber exatamente qual será a saída.

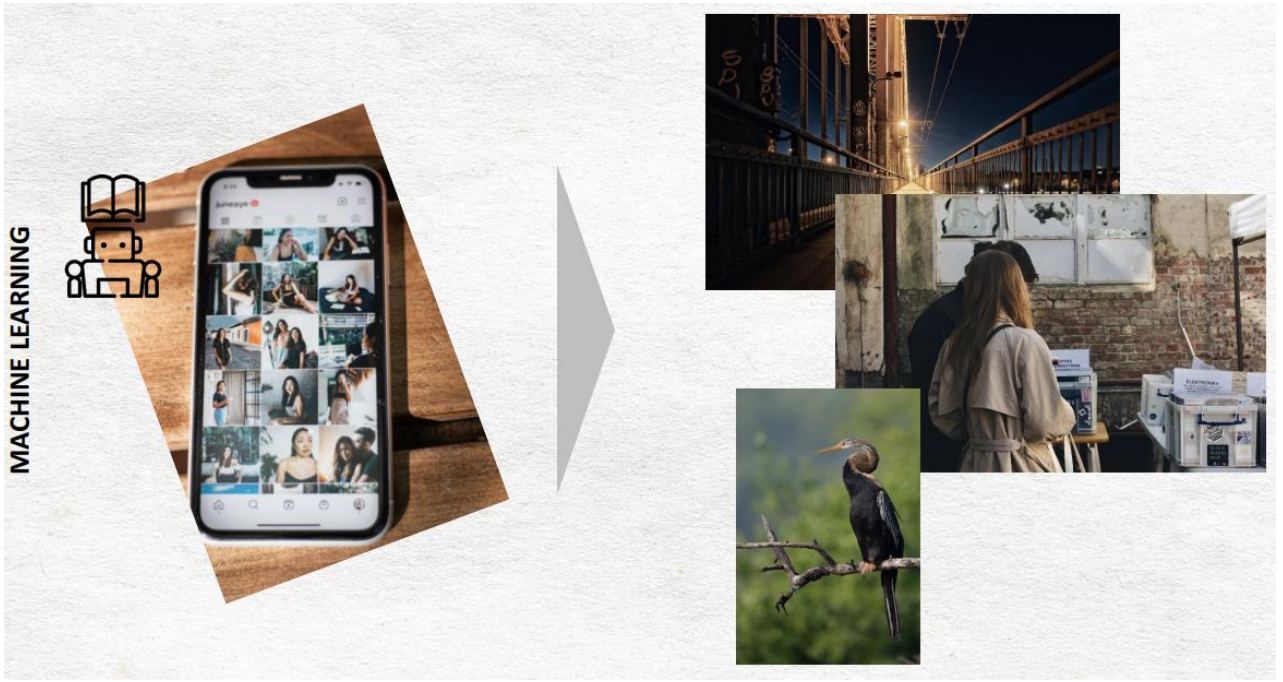


Módulo 10 – O que é aprendizado de máquinas (Machine Learning)?

Com Machine Learning, o programa vai aprendendo a cada interação. Por exemplo, as recomendações de filmes na Netflix dependem de como você interage e avalia outros filmes.



Da mesma forma, o feed do Instagram de cada pessoa depende da forma que ela usa o aplicativo. Se ela curte mais fotos ou vídeos; se interage mais com amigos ou com perfis diversos; se costuma comentar, dentre diversos outros critérios.

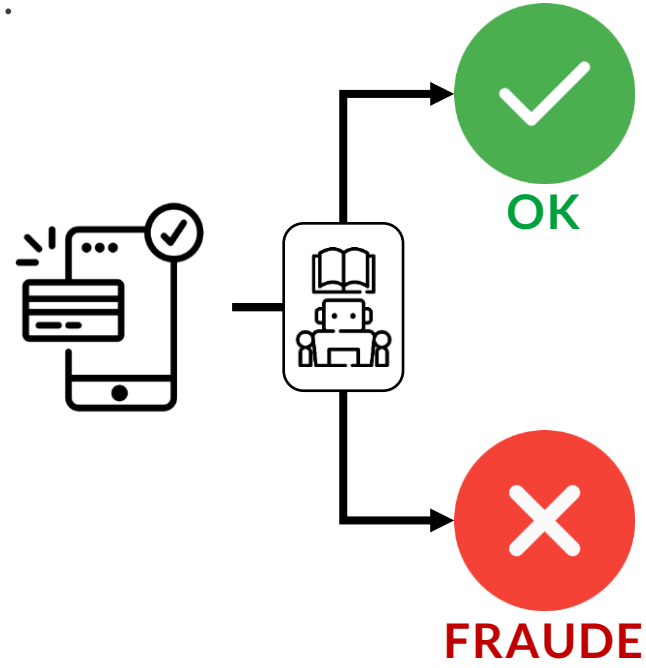


Módulo 10 – Como funciona um modelo de aprendizado de máquinas?

Primeiro, precisamos lembrar que, para ter um projeto de ciência de dados, precisamos de... **DADOS**!

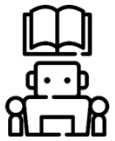
Com os dados, conseguimos encontrar padrões que antes seriam imperceptíveis, e usar esses padrões para gerar diferenciais de negócio.

Para entender, vamos pegar um exemplo clássico de aprendizado de máquinas: a identificação de **fraudes** no setor bancário.



Módulo 10 – Como funciona um modelo de aprendizado de máquinas?

Como precisamos de dados, vamos fazer um levantamento do que é possível obter da base de dados de um sistema bancário:



INFORMAÇÕES HISTÓRICAS

- Valor médio das transações anteriores
- Valor máximo entre os últimos pagamentos
- Perfil de compra
- Lojas que o cliente costuma comprar
- ...

- Já ocorreu algum problema nessa mesma loja?
- A loja vende produtos confiáveis?
- Outros clientes tiveram problema no estabelecimento?
- ...

- Hora das transações anteriores
- Preferência por transações presenciais ou online
- Histórico de parcelamento
- Frequência de compras
- ...

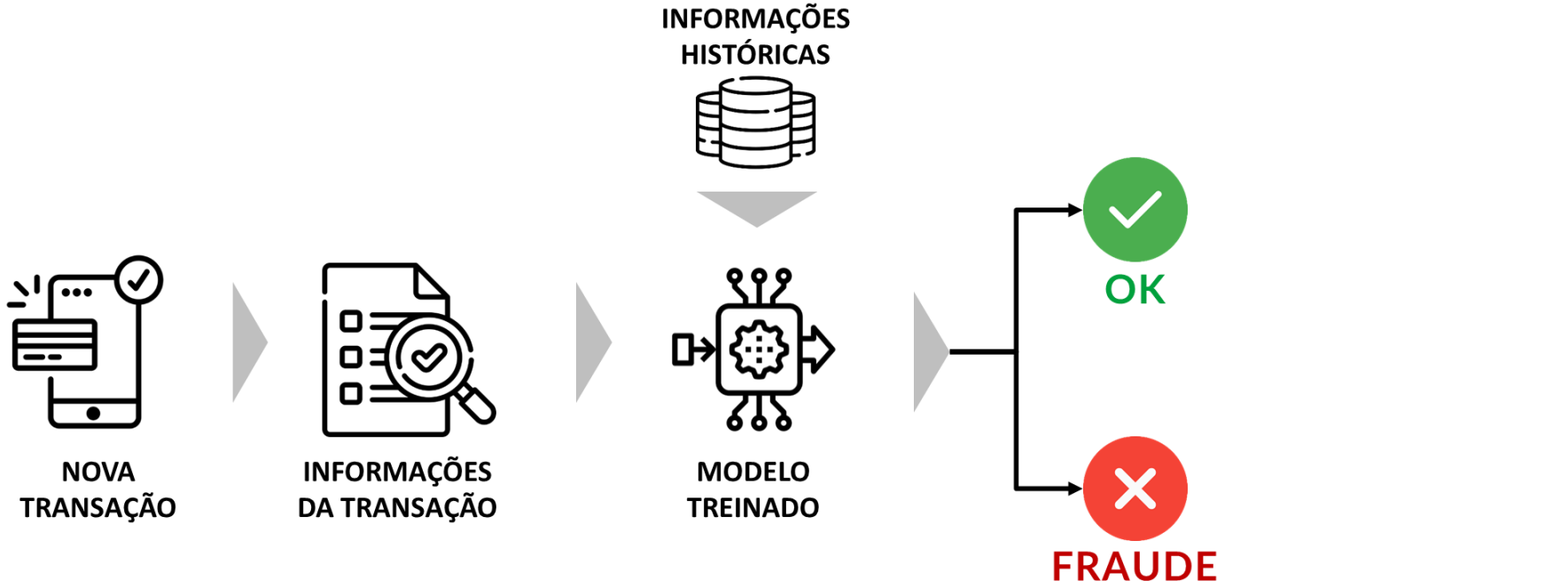
- Lojas que são facilmente fraudadas
- Perfil dos clientes que já sofreram fraudes
- Regiões / sites mais propensos a fraudes
- ...

Módulo 10 – Como funciona um modelo de aprendizado de máquinas?

Com os dados históricos, um modelo de aprendizado de máquinas pode detectar padrões e identificar quais dos dados são mais relevantes, ranqueando-os por relevância. Na imagem do nosso exemplo abaixo, uma cor verde mais forte indica maior relevância. Um risco indica dados que o modelo considera irrelevante.

Módulo 10 – Como funciona um modelo de aprendizado de máquinas?

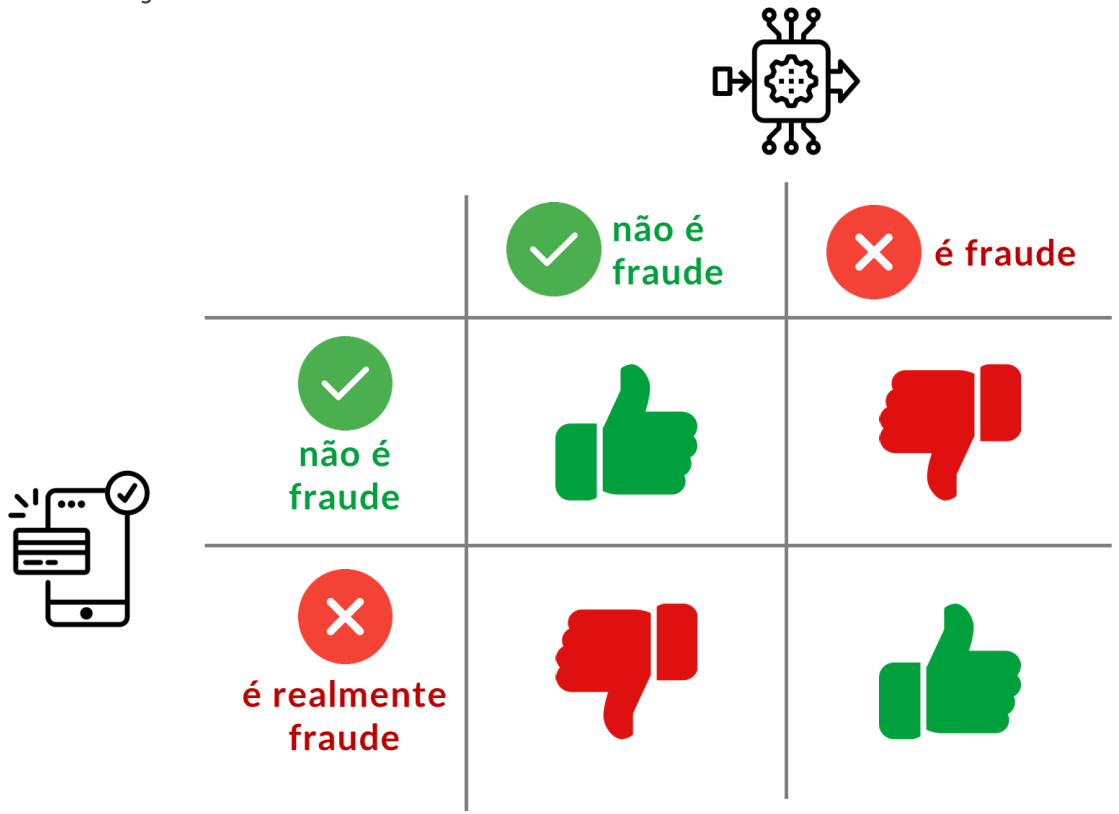
Agora que nosso modelo já foi treinado com dados históricos da base do banco, ele pode analisar novas transações e classificá-las como fraudulentas ou não:



Esse modelo não vai ser 100% preciso, queremos que ele seja **o mais preciso possível**!

Módulo 10 – Como funciona um modelo de aprendizado de máquinas?

Na vida real, é muito difícil que um modelo acerte todas as previsões. Assim, teremos os chamados falsos positivos (transações não fraudulentas erroneamente classificadas como fraude) e os falsos negativos (transações fraudulentas erroneamente classificadas como legítimas):



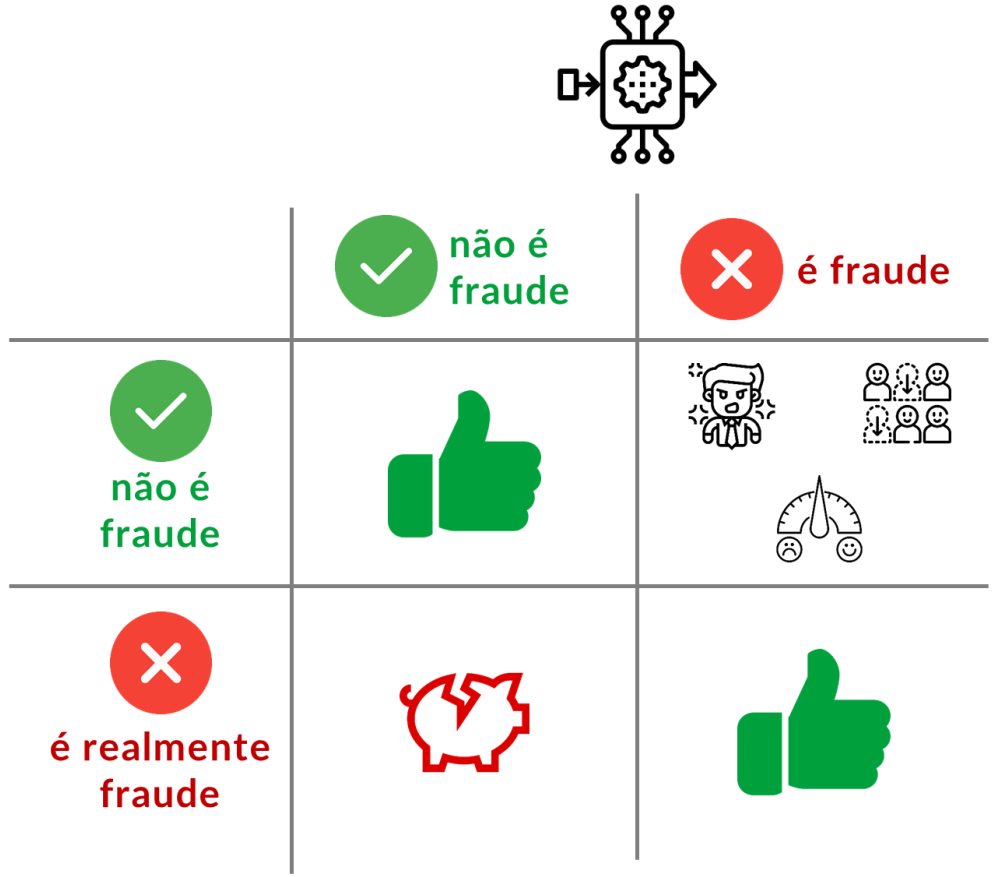
Módulo 10 – Como funciona um modelo de aprendizado de máquinas?

É importante entender a consequência desses erros:

- Ao classificar erroneamente uma transação fraudulenta como legítima, um usuário teve seu dinheiro roubado via, por exemplo, clonagem do cartão. A companhia do cartão terá que reembolsar o usuário.

- Por outro lado, uma transação legítima ser considerada fraudulenta significa que o usuário terá seu cartão bloqueado indevidamente, causando conflito e desconforto no cliente.

A empresa precisa definir os parâmetros do modelo de forma a tentar minimizar os custos decorrentes de ambos os erros.

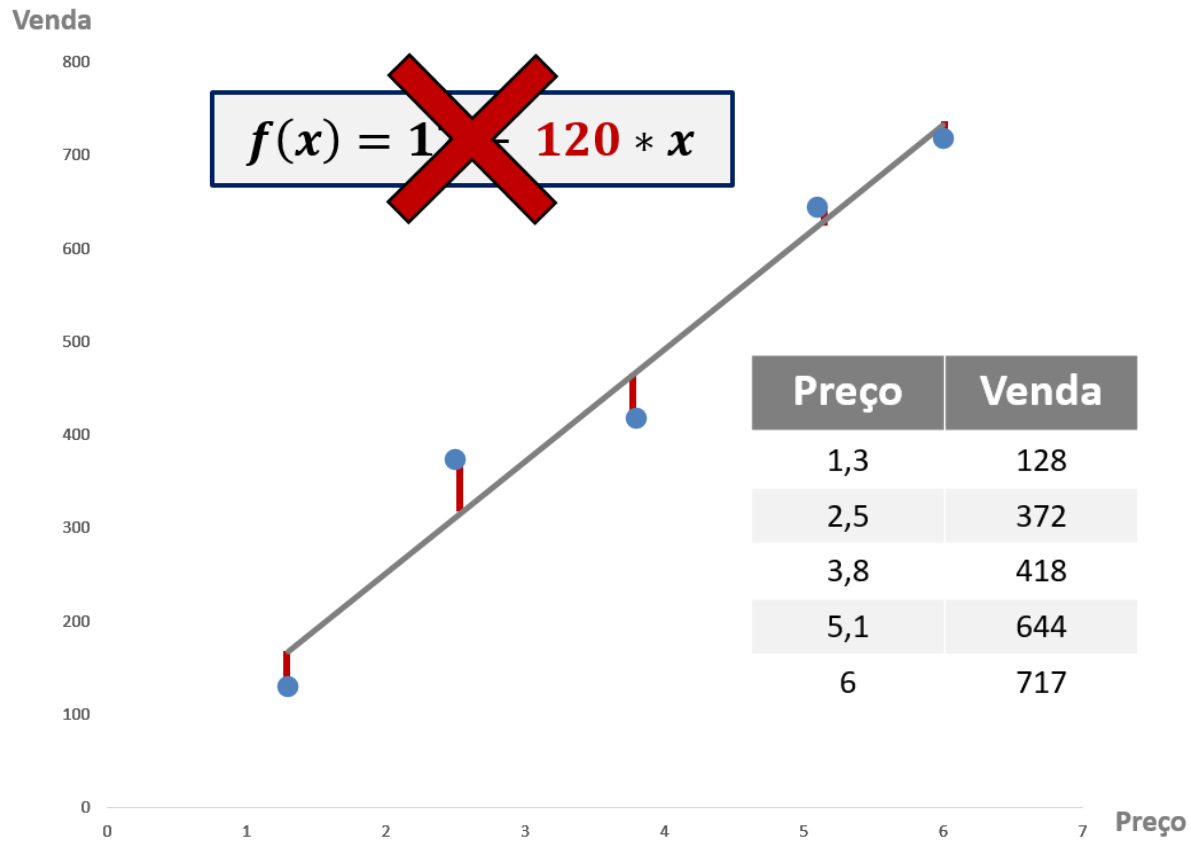


Módulo 10 – Como funciona um modelo de aprendizado de máquinas?

Tudo que vimos conceitualmente para um problema de classificação como o de fraudes também vale para um problema de regressão.

Considere um exemplo simples, onde se busca entender se a relação entre preço e venda é linear, como na figura.

Muito dificilmente se encontrará uma reta que passa por todos os pontos. O que se busca é uma reta que represente bem a tendência. Observe na figura que se marcou em vermelho a distância de cada ponto à reta. Assim, pode-se pensar que uma boa reta seria aquela que minimiza esta distância, que pode ser entendida como o erro da regressão.



Preço	Venda
1,3	128
2,5	372
3,8	418
5,1	644
6	717

Agora que já temos um bom embasamento teórico, vamos aplicar.

Comecemos com um exemplo simples, de dados que queremos verificar se possuem uma tendência linear.

Vamos começar importando as bibliotecas que precisaremos para criar uma base com esses dados e gráficos: NumPy, Pandas e Matplotlib.

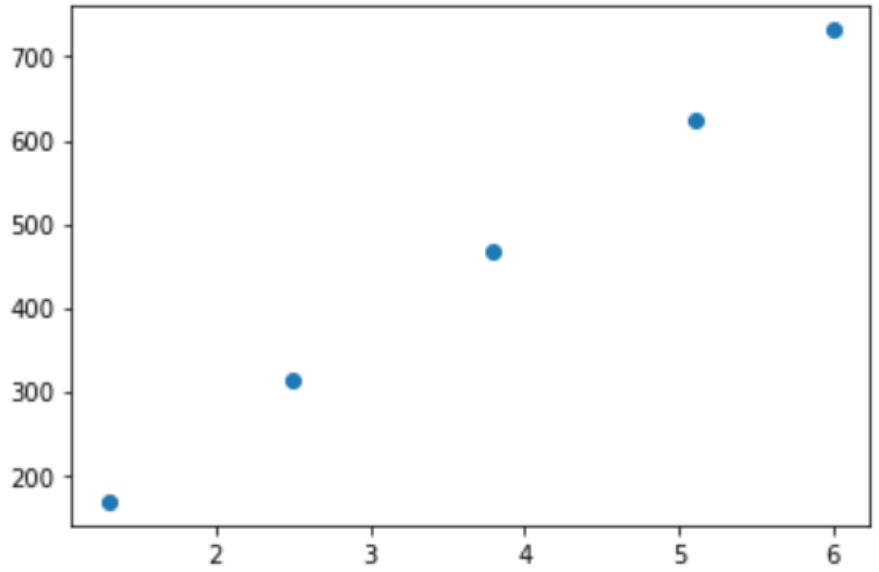
```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt
        import numpy as np
```

```
In [2]: dados = {
            'X': [1.3,2.5,3.8,5.1,6],
            'Y': [168,312,468,624,732]
        }

        dados = pd.DataFrame(dados)
```

Prosseguimos fazendo um gráfico de pontos com estes dados.

Veja que, por simples observação, já percebemos um comportamento linear.

```
In [3]: fig,ax = plt.subplots()

        ax.scatter(dados.X,dados.Y)

        plt.show()
```



Podemos pegar as coordenadas dos pontos inicial `(dados.X[0], dados.Y[0])` e final `(dados.X[4], dados.Y[4])` e plotar uma reta.

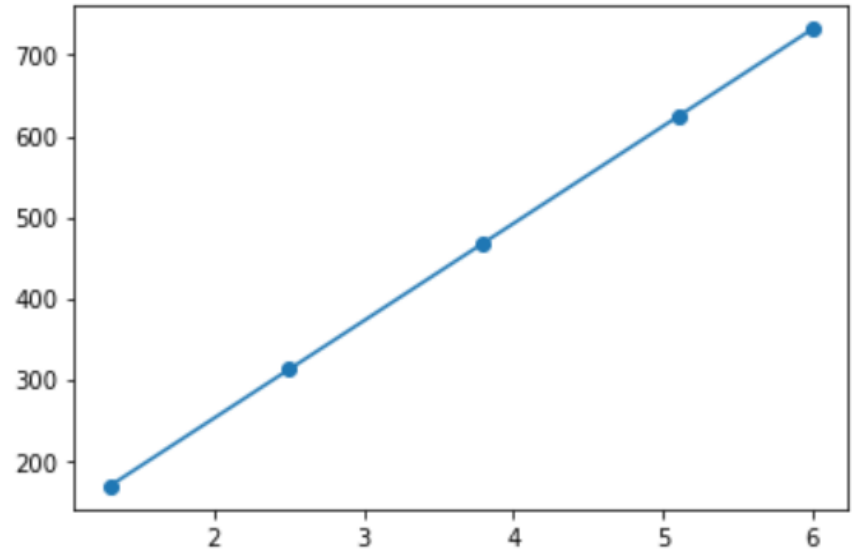
A equação dessa reta é:

$$f(x) = 12 + 120x$$

Se quisermos estimar o valor para, por exemplo, `x= 4,5`, basta substituir o valor na equação:

$$f(4,5) = 12 + 120 \cdot (4,5) = 552$$

```
In [4]: fig,ax = plt.subplots()

        ax.scatter(dados.X,dados.Y)
        x = [dados.X[0],dados.X[4]]
        y = [dados.Y[0],dados.Y[4]]
        ax.plot(x,y)

        plt.show()
```



Agora, não faz muito sentido fazermos a equação e a substituição manualmente se temos o Python.

Podemos importar do Scikit-Learn o modelo linear LinearRegression.

Passamos, então, para o método fit, o valores de X e Y:

```
In [5]: from sklearn.linear_model import LinearRegression
        modelo = LinearRegression()
        modelo.fit(dados.X.values.reshape(-1, 1),dados.Y)

Out[5]: LinearRegression()
```

Agora, o modelo pode nos retornar os coeficientes linear e angular:

O coeficiente linear desse modelo é dado por:

```
In [6]: # Coeficiente linear
        modelo.intercept_

Out[6]: 12.0
```

E o coeficiente angular:

```
In [7]: # Coeficiente angular
        modelo.coef_

Out[7]: array([120.])
```

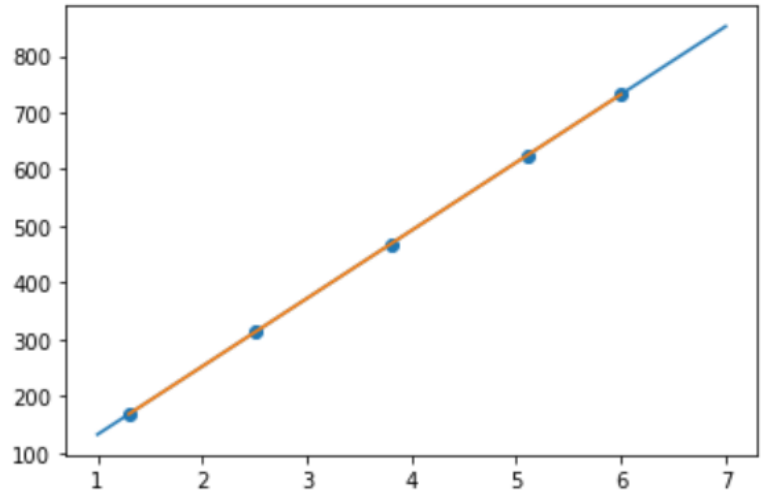
Lembre-se que a equação básica de uma reta é `y = ax + b`, sendo `a` o coeficiente angular, e `b` o coeficiente linear. Assim, podemos criar um conjunto de valores `x` com o método `arange` do NumPy e seus respectivos valores de `y`.

Com isso, podemos pegar nossos pontos originais e plotar a reta do nosso modelo.

Na figura, em laranja está a reta que havíamos criado anteriormente com os pontos inicial e final; e, em azul, a reta do nosso modelo.

```
In [9]: x = np.arange(1,8)
        y = modelo.intercept_ + x*modelo.coef_[0]
```

```
In [10]: fig,ax = plt.subplots()

         ax.scatter(dados.X,dados.Y)
         ax.plot(x,y)
         x2 = [dados.X[0],dados.X[4]]
         y2 = [dados.Y[0],dados.Y[4]]
         ax.plot(x2,y2)

         plt.show()
```



Por fim, com o método **predict**, podemos estimar o valor de y para um dado valor de x:

```
In [11]: # Prevendo algum valor usando o modelo que acabamos de criar
         modelo.predict(pd.DataFrame([4.5]))

Out[11]: array([552.])
```

Dessa forma, vimos como reproduzir com Python tudo que fizemos anteriormente de forma manual.

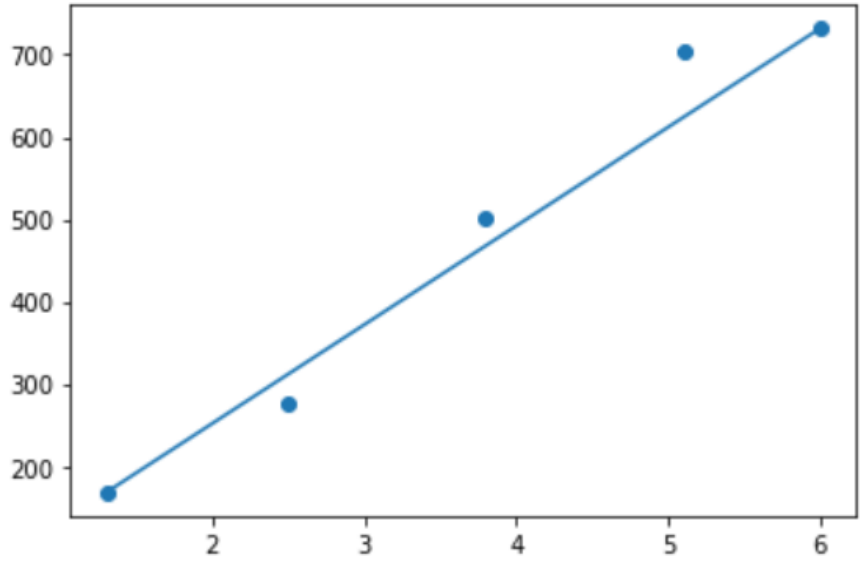
Vimos na parte teórica que, na vida real, usualmente não temos pontos tão perfeitamente lineares. Vamos gerar um conjunto de dados que seja mais próximo do que efetivamente encontramos na realidade:

In [12]:
```
dados2 = {
    'X': [1.3,2.5,3.8,5.1,6],
    'Y': [168,275,501,705,732]
}

dados2 = pd.DataFrame(dados2)
```

Veja o perfil dos pontos graficamente. A reta azul liga o ponto inicial ao ponto final.

Mas será essa a melhor reta que descreve estes pontos?

In [13]:
```
fig,ax = plt.subplots()

ax.scatter(dados2.X,dados2.Y)
x = [dados.X[0],dados.X[4]]
y = [dados.Y[0],dados.Y[4]]
ax.plot(x,y)

plt.show()
```

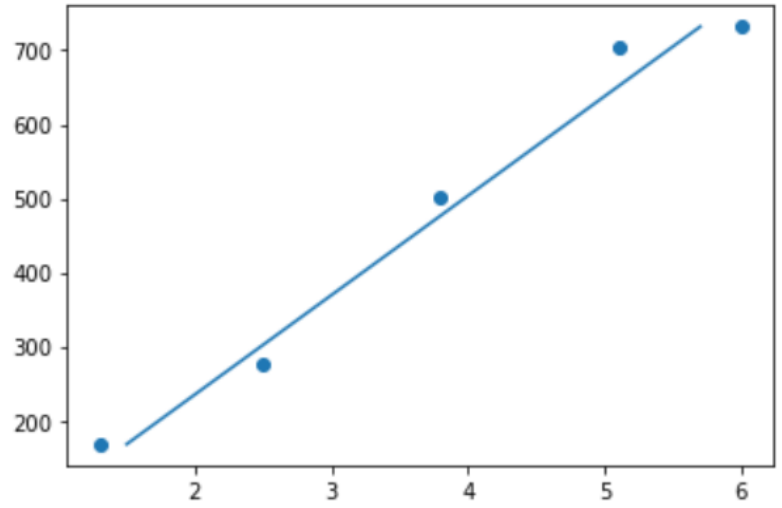


Podemos, por exemplo, modificar levemente essa reta deslocando-a. E agora? Essa reta é melhor que a anterior?

Como vimos na discussão inicial, podemos considerar que a melhor reta será aquela que minimiza a distância dos pontos à reta.

O Scikit-Learn nos permite encontrar esta reta com mínimo esforço.

In [14]:
```
fig,ax = plt.subplots()

ax.scatter(dados2.X,dados2.Y)
x = [dados.X[0]+0.2,dados.X[4]-0.3]
y = [dados.Y[0],dados.Y[4]]
ax.plot(x,y)

plt.show()
```



Primeiro, importamos o modelo `LinearRegression`. Então, passamos nossos valores de X e Y para o método `fit` do modelo:

```
In [15]: from sklearn.linear_model import LinearRegression
         modelo = LinearRegression()
         modelo.fit(dados2.X.values.reshape(-1, 1),dados2.Y)

Out[15]: LinearRegression()
```

Agora conseguimos obter o intercepto e o coeficiente angular de nossa reta:

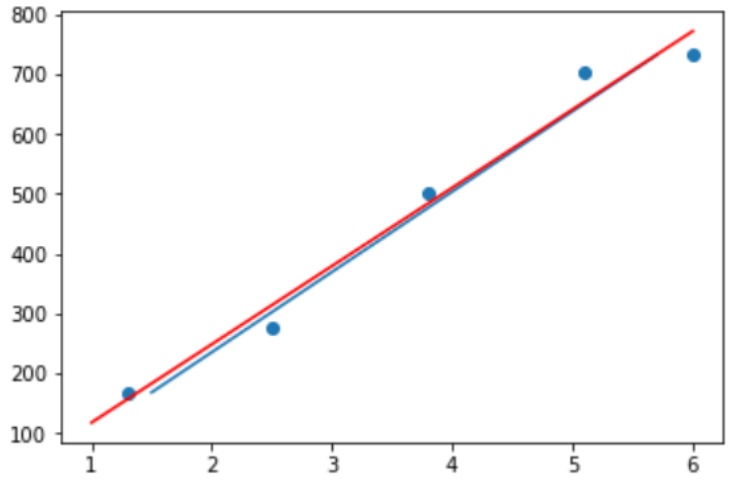
```
In [16]: # Coeficiente linear
         print(modelo.intercept_)
         # Coeficiente angular
         print(modelo.coef_[0])

         -13.493634099086592
         130.93412676446167
```

Veja que a reta gerada pelo modelo, em vermelho, nem é a união dos pontos inicial e final, nem coincide com a reta que fizemos manualmente.

O algoritmo do modelo irá obter a reta que melhor descreve os pontos.

In [17]:
```
x_ML = np.arange(1,7)
y_ML = modelo.intercept_ + x_ML*modelo.coef_[0]
```

In [18]:
```
fig,ax = plt.subplots()

ax.scatter(dados2.X,dados2.Y)
x = [dados.X[0]+0.2,dados.X[4]-0.3]
y = [dados.Y[0],dados.Y[4]]
ax.plot(x,y)
ax.plot(x_ML,y_ML,c="red")

plt.show()
```



Tudo que foi feito para pontos em duas dimensões também funciona em 3 dimensões.

Vamos gerar um conjunto de dados:

In [26]:
```
dados4 = {
    'X': [1.3,2.5,3.8,5.1,6],
    'Z': [0.2,0.1,0.3,0.2,0.1],
    'Y': [158,315,450,630,700]
}

dados4 = pd.DataFrame(dados4)
```

E, da mesma forma que antes, passar para o método `fit`:

In [28]:
```
from sklearn.linear_model import LinearRegression
modelo = LinearRegression()
modelo.fit(dados4.drop("Y",axis=1),dados4.Y)
```

Out[28]:
```
LinearRegression()
```

E, assim como visto quando tínhamos duas dimensões, podemos obter nossos coeficientes. Só que agora será uma reta em três dimensões:

```
In [29]: # Coeficiente linear
         print(modelo.intercept_)
         # Coeficiente angular
         print(modelo.coef_)

         14.948614201901592
         [116.86425248  -7.8939915 ]
```

A equação dessa reta é, portanto, aproximadamente:

$$y = 14.9486 + 116.8642x - 7.89399z$$

Utilizando o método `predict`, podemos ver como nosso modelo prevê valores de Y e comparar com os valores iniciais:

```
In [30]: dados4['Y_predicao'] = modelo.predict(dados4.drop("Y",axis=1))
```

```
In [31]: display(dados4)
```

	X	Z	Y	Y_predicao
0	1.3	0.2	158	165.293344
1	2.5	0.1	315	306.319846
2	3.8	0.3	450	456.664576
3	5.1	0.2	630	609.377504
4	6.0	0.1	700	715.344730

Agora que temos um embasamento de como funciona com pontos fictícios, vamos pegar uma base mais próxima do real, com dados de vendas de uma loja. Nosso objetivo é estimar o estoque ideal para a loja.

Primeiro, vamos começar importando o Pandas, que será a biblioteca que utilizaremos para criar nossa base de dados a partir de uma planilha Excel.

Lembrando, o método **head** nos permite visualizar as primeiras entradas da base.

```
# Importando a base de dados
# - Base de dados: "dadosVenda.xlsx"
import pandas as pd
dados = pd.read_excel("dadosVenda.xlsx")
```

```
# Visualizando os dados
dados.head()
```

	IDRegistro	PrecoVenda	PrecoOriginal	Desconto	VendaQtd
0	1	21.85	23	1.15	81
1	2	4.30	5	0.70	3361
2	3	13.65	15	1.35	1551
3	4	4.97	7	2.03	3036
4	5	9.60	10	0.40	2436

O método `info` nos fornece informações básicas para cada coluna do dataframe. Por exemplo, vemos que há 1634 linhas. No entanto, na coluna `Desconto` há apenas 1606 valores. Logo, há valores vazios nessa coluna.

É muito importante fazer essa checagem sempre, logo no início. Para definir qual a estratégia que será utilizada para tratar estas situações.

```
# Verificando as informações da base
dados.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1634 entries, 0 to 1633
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   IDRegistro     1634 non-null   int64
 1   PrecoVenda     1634 non-null   float64
 2   PrecoOriginal  1634 non-null   int64
 3   Desconto       1606 non-null   float64
 4   VendaQtd       1634 non-null   int64
dtypes: float64(2), int64(3)
memory usage: 64.0 KB
```

O método `describe` também é muito útil, dando um resumo estatístico de cada coluna. Aqui vemos também que há valores ausentes na coluna `Desconto`.

Em breve, ainda nesse módulo, entenderemos melhor o significado de cada descrição estatística dessa tabela.

```
dados.describe()
```

	IDRegistro	PrecoVenda	PrecoOriginal	Desconto	VendaQtd
count	1634.000000	1634.000000	1634.000000	1606.000000	1634.000000
mean	817.500000	11.152889	13.913097	2.808331	1793.477356
std	471.839485	4.675302	5.422824	2.055469	960.099218
min	1.000000	2.820000	3.000000	0.050000	45.000000
25%	409.250000	7.200000	9.000000	1.140000	995.000000
50%	817.500000	10.920000	14.000000	2.250000	1753.000000
75%	1225.750000	14.520000	19.000000	4.200000	2646.000000
max	1634.000000	23.000000	23.000000	9.200000	3711.000000

No nosso caso, os valores vazios simplesmente significam situações onde não houve desconto. Logo, podemos com o método `loc` substituir essas situações pelo valor zero.

Após a substituição, vemos com o método `info` que há 1634 valores na coluna `Desconto`:

```
# Ajustando os dados com valores de desconto vazio
dados.loc[dados.Desconto.isnull(),"Desconto"] = 0
```

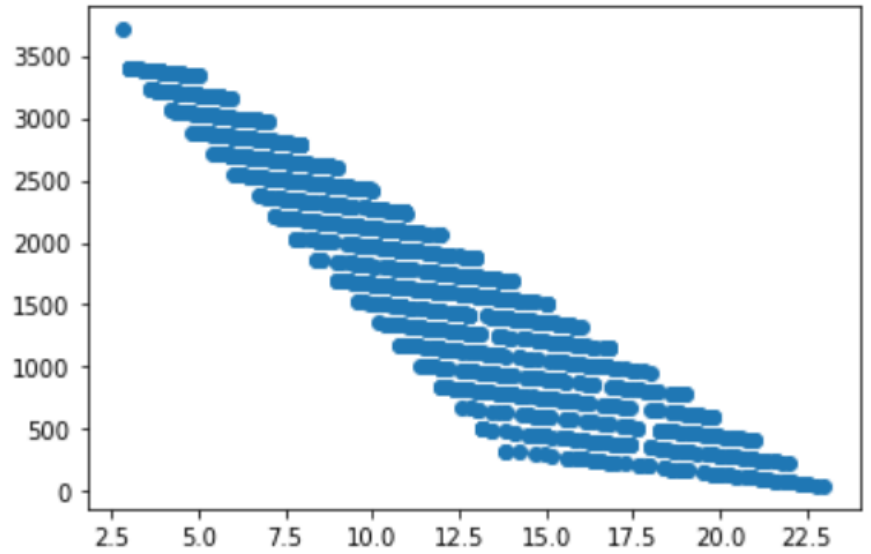
```
# Verificando novamente as informações da base
dados.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1634 entries, 0 to 1633
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   IDRegistro     1634 non-null   int64
 1   PrecoVenda     1634 non-null   float64
 2   PrecoOriginal  1634 non-null   int64
 3   Desconto       1634 non-null   float64
 4   VendaQtd       1634 non-null   int64
dtypes: float64(2), int64(3)
memory usage: 64.0 KB
```

Módulo 10 – Regressão linear no Scikit-Learn

Uma boa forma de entender os dados, é fazer um gráfico de dispersão, *scatter plot* em inglês. Para isso, importamos o Matplotlib e utilizamos o método `scatter`, passando nosso X e Y desejado. No caso, preço de vendas e quantidade de vendas.

```
# Podemos fazer o scatter plot
import matplotlib.pyplot as plt

# plot
fig, ax = plt.subplots()

ax.scatter(dados.PrecoVenda, dados.VendaQtd)

plt.show()
```



Após ajustar os dados e visualizá-los, podemos partir para nosso modelo de machine learning. Assim como já fizemos antes, importaremos o `LinearRegression` do Scikit-Learn e passaremos nosso X e nosso y desejados:

```
# Importando o LinearRegression
from sklearn.linear_model import LinearRegression
```

```
# Criando os dados
X = dados.PrecoVenda
y = dados.VendaQtd
```

```
# Fazendo o fit do modelo
reg = LinearRegression().fit(X.values.reshape(-1, 1), y)
```

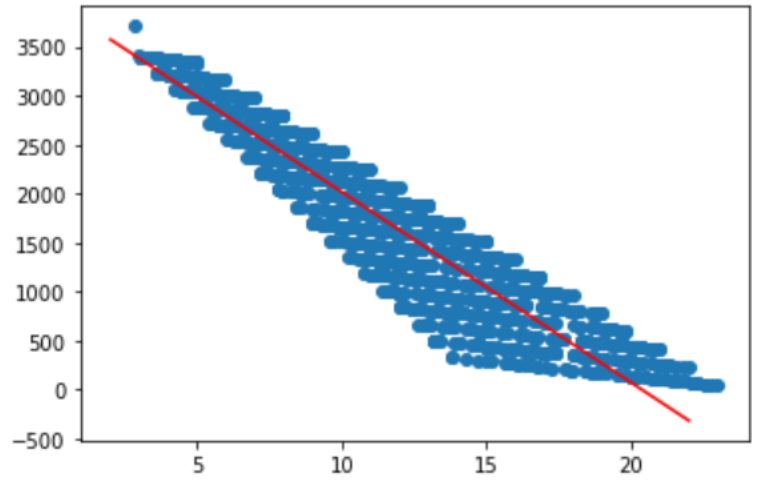
Para verificar o quão bom um modelo linear descreve nossos dados, usamos o método `score`. Quanto mais próximo de 1, melhor o modelo linear descreve nossos dados:

```
reg.score(X.values.reshape(-1, 1), y)
```

```
0.895643694044705 9
```

Vimos, também, que podemos obter o coeficiente angular da reta e o intercepto com `coef_` e `Intercept_`. Assim, podemos criar nossa reta graficamente:

```
# Podemos fazer o scatter plot
import matplotlib.pyplot as plt
import numpy as np

# plot
fig, ax = plt.subplots()

ax.scatter(dados.PrecoVenda, dados.VendaQtd)
x_plot = np.arange(2,23)
y_plot = reg.coef_[0]*x_plot + reg.intercept_
ax.plot(x_plot,y_plot,c='r')

plt.show()
```



Vamos criar um conjunto de valores para os quais queremos fazer nossa previsão:

```
# Podemos definir uma lista de valores para fazermos a previsão
valores = {
    'valores': [17.50, 17.40, 17.30, 17.20, 17.10]
}

valores = pd.DataFrame(valores)
```

Com o método `predict`, podemos verificar qual a previsão de vendas para cada preço:

```
# Fazendo o predict
reg.predict(valores)
```

```
array([559.94606196, 579.3805924 , 598.81512285, 618.2496533 ,
       637.68418374])
```

Será que essa previsão faz sentido? Veja abaixo que, pegando apenas o valor de `17.10`, o valor previsto (~637) é bem distante do valor que realmente foi vendido:

```
# Verificando os dados para esse mesmo preço de venda
dados[dados.PrecoVenda == 17.10]
```

	IDRegistro	PrecoVenda	PrecoOriginal	Desconto	VendaQtd
397	398	17.1	19	1.9	836
492	493	17.1	19	1.9	836
1414	1415	17.1	18	0.9	988

Podemos olhar com mais cuidado a faixa entre 17 e 18 de preço original, como abaixo. Os valores de venda são bem distintos do previsto. Ou seja, erramos no nosso modelo. Afinal, fizemos apenas com o preço de venda, mas talvez seja interessante considerar o preço original e o desconto.

```
# Verificando os dados para um range de preços e descontos
dados[(dados.PrecoOriginal >= 17) & (dados.PrecoOriginal <= 18) & (dados.Desconto >= 0) & (dados.Desconto <= 0.4)]
```

	IDRegistro	PrecoVenda	PrecoOriginal	Desconto	VendaQtd
108	109	16.83	17	0.17	1148
284	285	16.66	17	0.34	1154
452	453	18.00	18	0.00	960
989	990	16.83	17	0.17	1148
998	999	16.83	17	0.17	1148
1115	1116	17.82	18	0.18	966
1220	1221	17.64	18	0.36	971
1539	1540	16.66	17	0.34	1154
1548	1549	16.66	17	0.34	1154
1576	1577	16.83	17	0.17	1148

Módulo 10 – Regressão linear no Scikit-Learn

Vamos então, recriar nosso modelo colocando como X as colunas `PrecoOriginal` e `Desconto`. Veja que é basicamente a mesma estrutura de código anterior.

Perceba como o valor do `score` melhorou bastante, mostrando que o novo modelo descreve muito melhor este conjunto de dados.

```
# Importar o algoritmo
from sklearn.linear_model import LinearRegression
```

```
# Dados
X2 = dados[["PrecoOriginal","Desconto"]]
y2 = dados.VendaQtd
```

```
# Fazendo o fit
reg2 = LinearRegression().fit(X2, y2)
```

```
reg2.score(X2, y2)
```

```
0.9999999083594651
```

```
reg.score(X.values.reshape(-1, 1),y)
```

```
0.8956436940447059
```

```
# Coeficiente angular
reg2.coef_
```

```
array([-182.99974391,   31.00137454])
```

```
# Coeficiente linear
reg2.intercept_
```

```
4254.00024394365
```

Veja que, com esse novo modelo, passando valores para previsão, temos valores mais próximos aos que efetivamente ocorreram:

```
valores = {
    'valores': [17.50, 17.40, 17.30, 17.20, 17.10],
    'precos': [17.50,17.50,17.50,17.50,17.50],
    'descontos': [0,0.1,0.2,0.3,0.4]
}

valores = pd.DataFrame(valores)
```

```
# Fazendo a predição
reg2.predict(valores[["precos","descontos"]])
```

```
array([1051.50472554, 1054.604863  , 1057.70500045, 1060.8051379 ,
       1063.90527536])
```

Compare a última coluna com as previsões anteriores:

```
# Verificando novamente os dados para um range de preços e descontos
dados[(dados.PrecoOriginal >= 17) & (dados.PrecoOriginal <= 18) & (dados.Desconto >= 0) & (dados.Desconto <= 0.4)]
```

	IDRegistro	PrecoVenda	PrecoOriginal	Desconto	VendaQtd
108	109	16.83	17	0.17	1148
284	285	16.66	17	0.34	1154
452	453	18.00	18	0.00	960
989	990	16.83	17	0.17	1148
998	999	16.83	17	0.17	1148
1115	1116	17.82	18	0.18	966
1220	1221	17.64	18	0.36	971
1539	1540	16.66	17	0.34	1154
1548	1549	16.66	17	0.34	1154
1576	1577	16.83	17	0.17	1148

Módulo 10 – Regressão linear no Scikit-Learn

Vamos comparar as previsões dos dois modelos lado a lado, criando colunas no nosso dataframe.

Novamente, veja como o segundo modelo apresenta valores muito mais próximos para a quantidade de vendas real.

```
dados['y_model1'] = reg.predict(dados.PrecoVenda.values.reshape(-1, 1))
dados['y_model2'] = reg2.predict(dados[["PrecoOriginal","Desconto"]])
```

```
dados.head()
```

	IDRegistro	PrecoVenda	PrecoOriginal	Desconto	VendaQtd	y_model1	y_model2
0	1	21.85	23	1.15	81	-285.456012	80.657715
1	2	4.30	5	0.70	3361	3125.304081	3360.702487
2	3	13.65	15	1.35	1551	1308.175484	1550.855941
3	4	4.97	7	2.03	3036	2995.092727	3035.934827
4	5	9.60	10	0.40	2436	2095.273967	2436.403355

```
dados.tail()
```

	IDRegistro	PrecoVenda	PrecoOriginal	Desconto	VendaQtd	y_model1	y_model2
1629	1630	11.96	13	1.04	1907	1636.619049	1907.245003
1630	1631	7.40	10	2.60	2505	2522.833637	2504.606379
1631	1632	12.15	15	2.85	1597	1599.693441	1597.358003
1632	1633	13.44	16	2.56	1405	1348.987998	1405.367860
1633	1634	2.82	3	0.18	3711	3412.935132	3710.581260

Módulo 10 – Regressão linear no Scikit-Learn

No Scikit-Learn, estão disponíveis muitas métricas para calcular erros de modelos. No momento, não nos aprofundaremos no significado de cada uma delas. Mais veja como, nas duas que importamos, o segundo modelo tem valores bem menores que o primeiro modelo.

À medida que formos avançando no curso, nos aprofundaremos no significado de cada uma dessas métricas de erro. Nesse caso, podemos interpretar como sendo a distância dos pontos à reta calculada. O segundo modelo tem uma reta que descreve melhor nossos dados, tendo estes uma menor distância à reta.

Para começar a ter o embasamento necessário para o entendimento dessas métricas, vejamos um pouco de estatística, aproveitando para aprender mais sobre Pandas, até para entender também o método `describe` visto anteriormente.

```
from sklearn.metrics import mean_absolute_error
```

```
# para o modelo 1
mean_absolute_error(dados.VendaQtd, dados.y_model1)
```

```
254.3109951899015
```

```
# para o modelo 2
mean_absolute_error(dados.VendaQtd, dados.y_model2)
```

```
0.24931777290882204
```

```
from sklearn.metrics import mean_absolute_percentage_error
```

```
# para o modelo 1
mean_absolute_percentage_error(dados.VendaQtd, dados.y_model1)*100
```

```
33.713612693046244
```

```
# para o modelo 2
mean_absolute_percentage_error(dados.VendaQtd, dados.y_model2)*100
```

```
0.027306697265634318
```

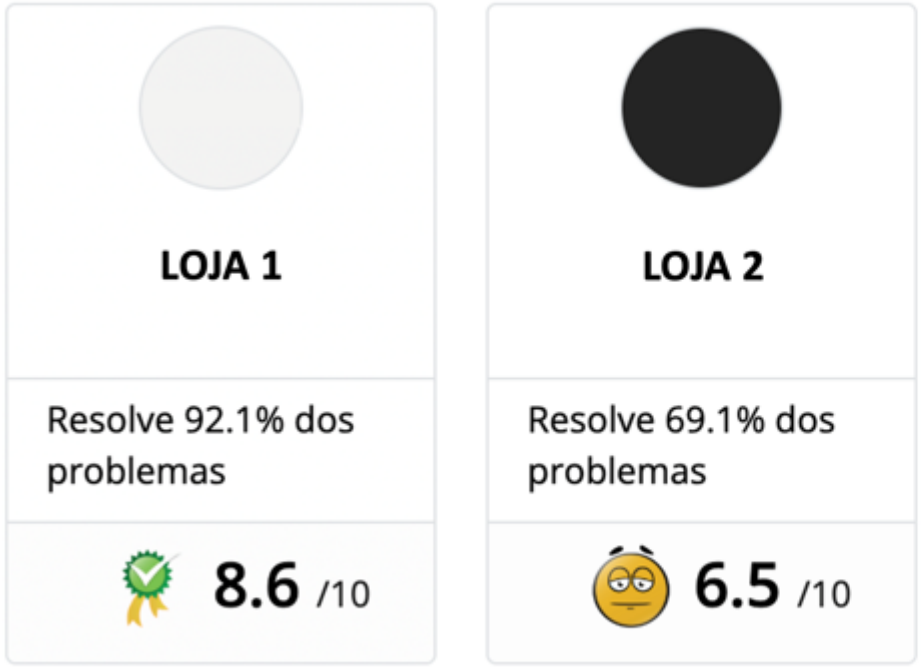
Módulo 10 – Descrição estatística com Pandas

Para começar nosso estudo, vamos considerar o seguinte exemplo, de notas de usuários quanto ao atendimento de problemas de duas lojas de carros.

Qual das duas empresas abaixo você escolheria pra comprar seu carro? Sabendo que a nota apresentada é uma média das notas dos usuários.

A média é insuficiente para fazermos uma boa escolha! Nosso objetivo é entender o porquê.

Temos diversas medidas estatísticas que nos permitem uma análise mais completa de dados:



- Média
- Mediana
- Moda

Medidas de Tendência Central

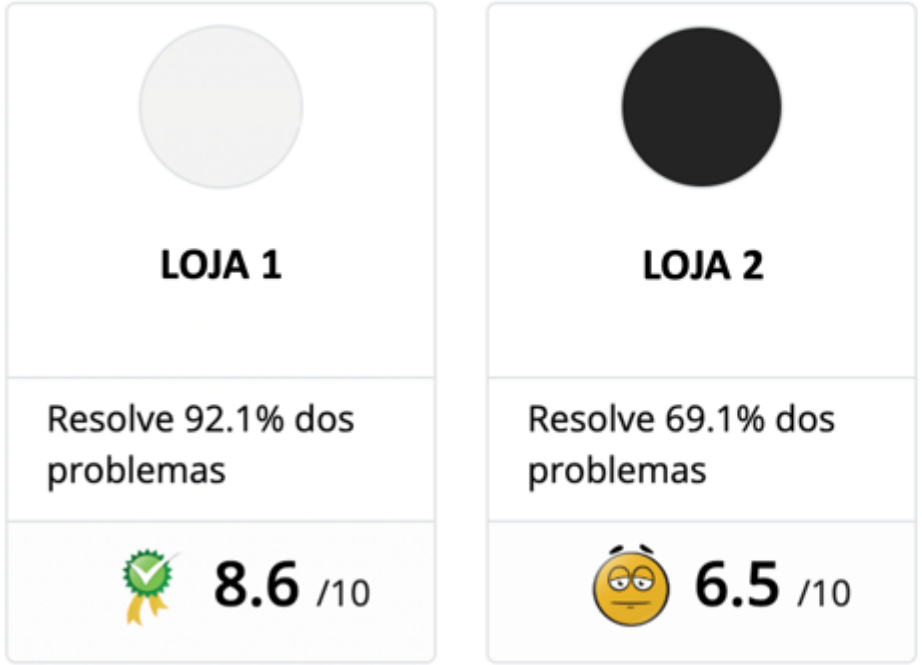
Medidas de Dispersão

- Range
- Variância
- Desvio Padrão

Medidas de Tendência Central

- Quartis

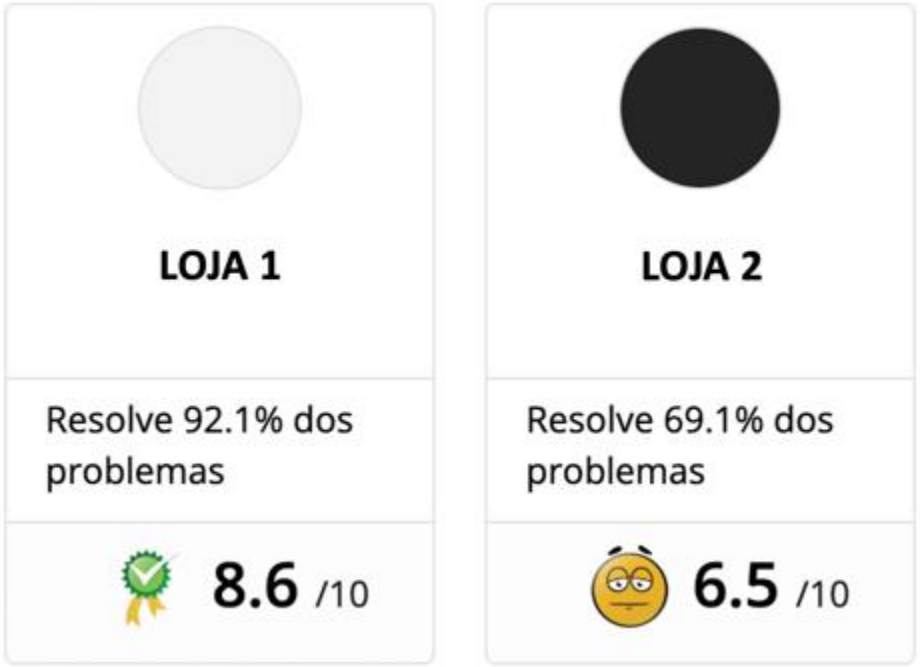
Não é necessário pegar cada valor individualmente, embora seja possível com métodos específicos:




```
base.empresa1.mean()
3.0

base.empresa1.median()
3.0

base.empresa1.std()
2.1908902300206643

base.empresa1.max()
5

base.empresa1.min()
1
```

Módulo 10 – Descrição estatística com Pandas

Como vimos anteriormente, o método `describe` do Pandas já apresenta as principais métricas todas resumidas em um dataframe fácil de compreender. Veja que, no nosso exemplo, há 6 avaliações para cada empresa. A **Empresa 1** apresenta valor de nota mínima 1 e máxima 5, enquanto que a **Empresa 2** apresenta todas as notas iguais a 3.




```
base.describe()
```

	empresa1	empresa2
count	6.00000	6.0
mean	3.00000	3.0
std	2.19089	0.0
min	1.00000	3.0
25%	1.00000	3.0
50%	3.00000	3.0
75%	5.00000	3.0
max	5.00000	3.0

Uma curiosidade nesse caso, é que ambas as empresas possuem mesma média e mesma mediana:

	Empresa 1	Empresa 2
Média	3	3
Mediana	3	3

```
base.empresa1.mean()
```

```
3.0
```

```
base.empresa1.median()
```

```
3.0
```

```
base.head()
```

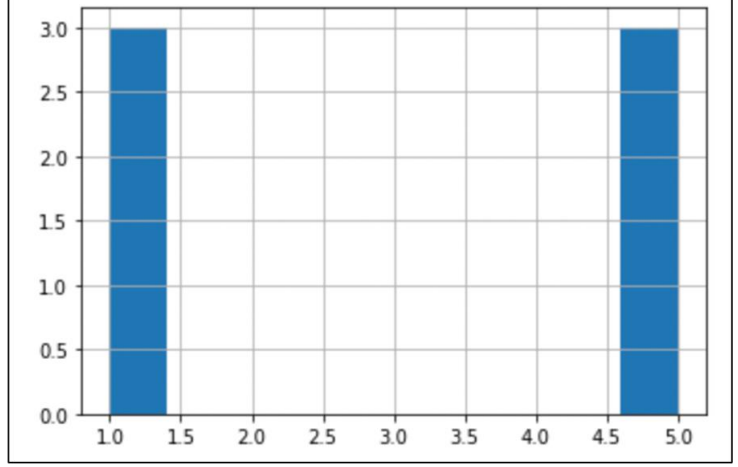
	empresa1	empresa2
0	1	3
1	1	3
2	1	3
3	5	3
4	5	3

Ora, então o que as difere? A **dispersão** dos dados. Podemos construir histogramas para as duas empresas, seja pelo método `hist` do Pandas, seja pelo `bar` do Matplotlib. Abaixo, as duas formas para a **Empresa 1**.

```
display(base)
```

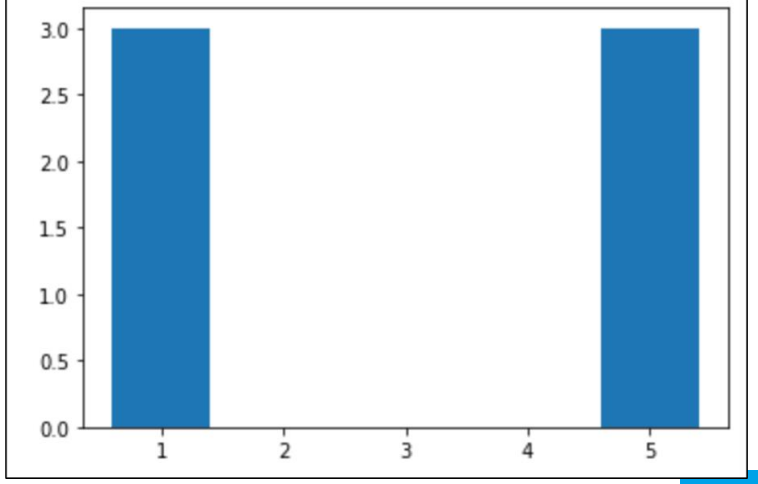
	empresa1	empresa2
0	1	3
1	1	3
2	1	3
3	5	3
4	5	3
5	5	3

```
base.empresa1.hist();
```

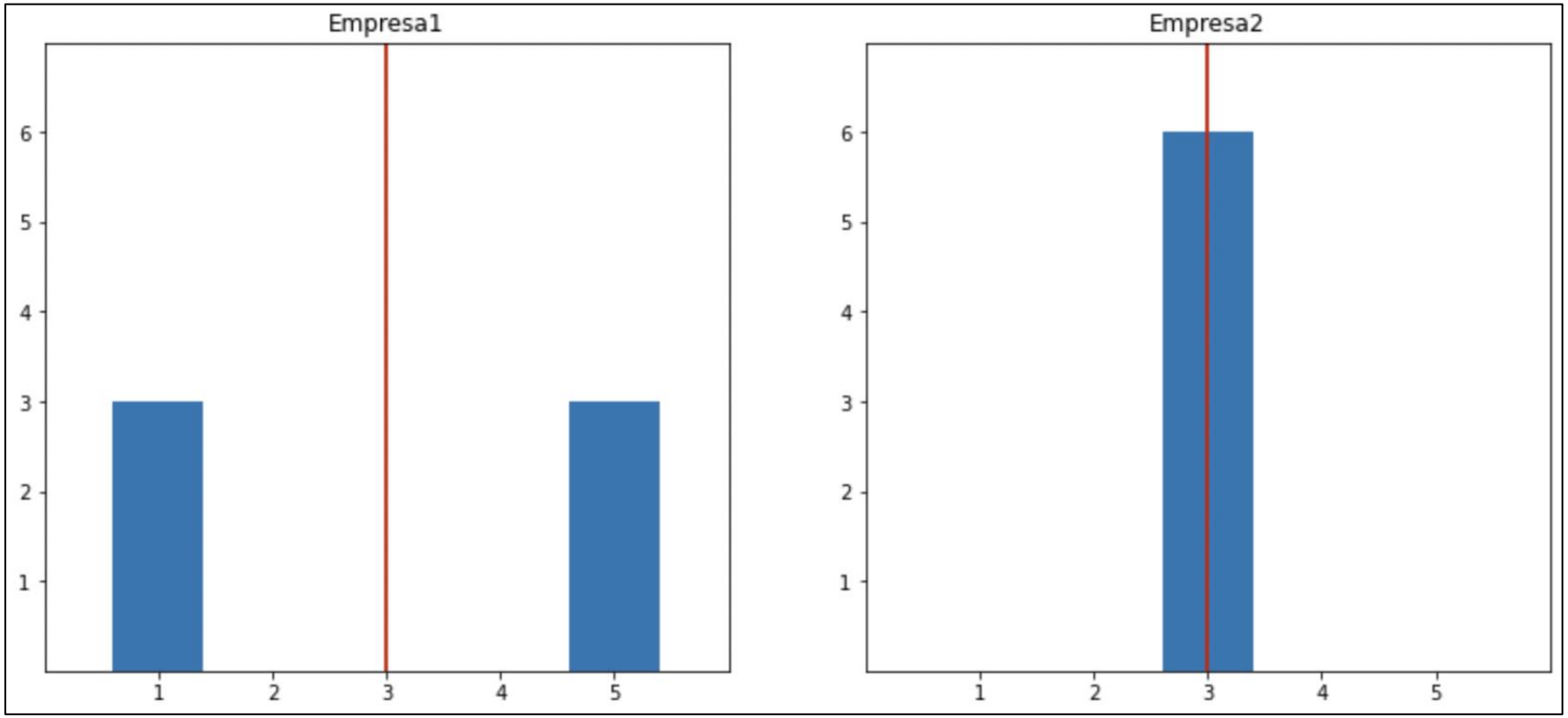



```
x = base.empresa1.value_counts().index
y = base.empresa1.value_counts().values
```

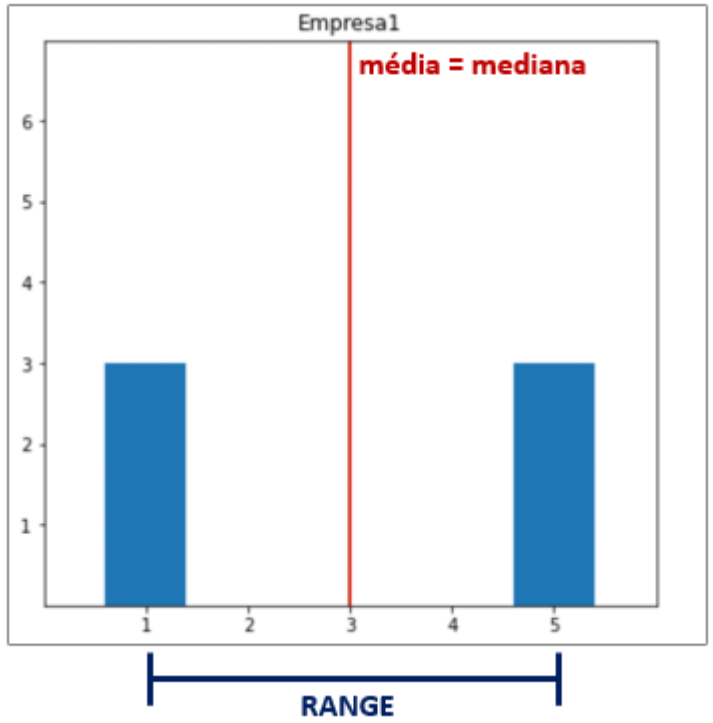
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.bar(x, y)
plt.show()
```



Observe que a **Empresa 2** é consistente nas avaliações, enquanto que a **Empresa 1** possui avaliações muito ruins e muito boas. Logo, o que você prefere? Consistência de um atendimento regular, ou a chance de ter um atendimento muito bom ou muito ruim?



Vemos na **Empresa 1**, que há uma diferença entre os valores máximo e mínimo, que chamamos de *range*, ou amplitude. A amplitude é muito sensível a outliers e, consequentemente, a média também.



RANGE (ou amplitude):

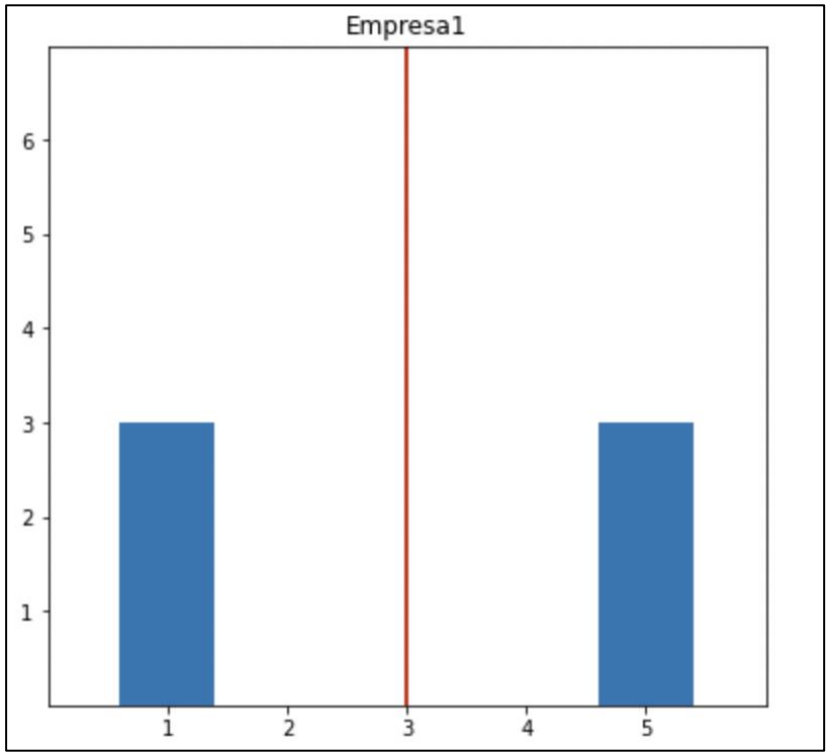
$$R = max - min$$

```
max1 = base.empresa1.max()
min1 = base.empresa1.min()
```

```
R = max1 - min1
print(R)
```

```
4
```

Uma forma de detectar a dispersão de dados, é utilizar a distância de cada ponto a média. Como há distâncias positivas e negativas, elevamos ao quadrado os valores. O nome dessa métrica é **variância amostral:**



x	$x - média$	$(x - média)^2$
1	1 – 3 = -2	4
1	1 - 3 = -2	4
1	1 - 3 = -2	4
5	5 - 3 = +2	4
5	5 - 3 = +2	4
5	5 - 3 = +2	4

VARIÂNCIA AMOSTRAL:

$$s^2 = \frac{\sum(x - \bar{x})^2}{n - 1}$$

$s^2 = 4,8$

O **desvio padrão** é simplesmente a raiz quadrada da variância amostral. É o **std** que aparece no `describe` do Pandas:

DESVIO PADRÃO AMOSTRAL:

$$s = \sqrt{S^2}$$

$$s = \sqrt{4{,}8}$$

$$s \simeq 2{,}19$$

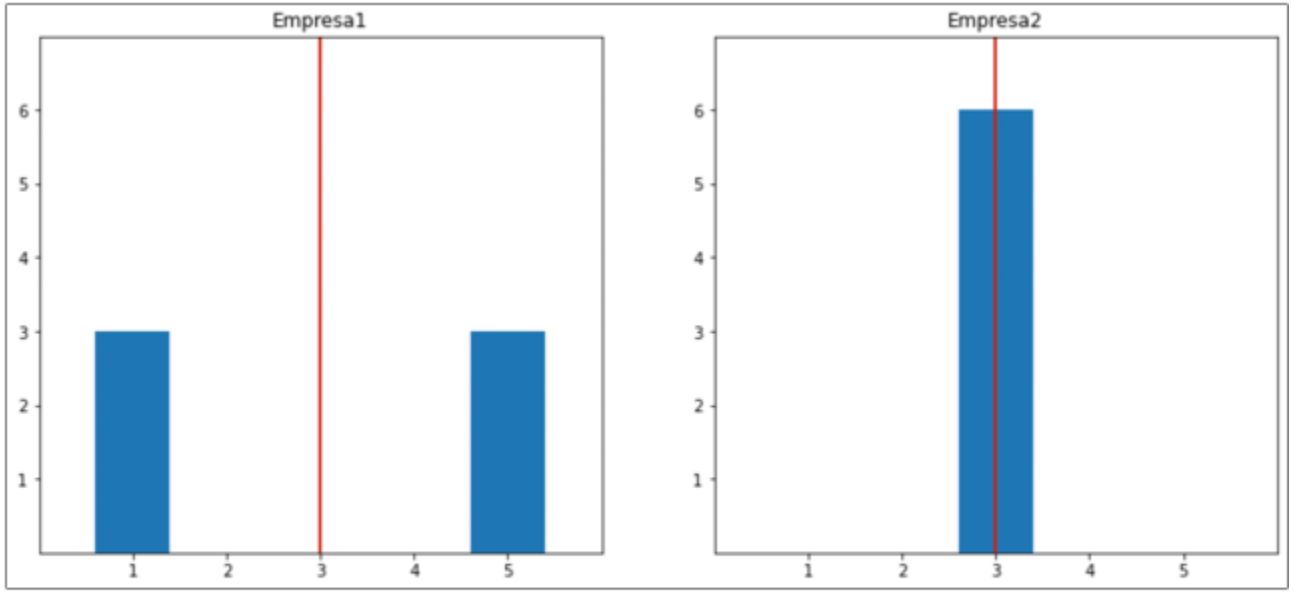
```
desvio_padrao = base.empresa1.std()
print(desvio_padrao)
```

```
2.1908902300206643
```

```
variancia = (desvio_padrao)**2
print(variancia)
```

```
4.799999999999999
```

Veja como o desvio padrão da **Empresa 1** é maior que o da **Empresa 2**, mostrando uma maior dispersão das notas para a primeira empresa.




```
desvio_padrao = base.empresa1.std()
print(desvio_padrao)
```

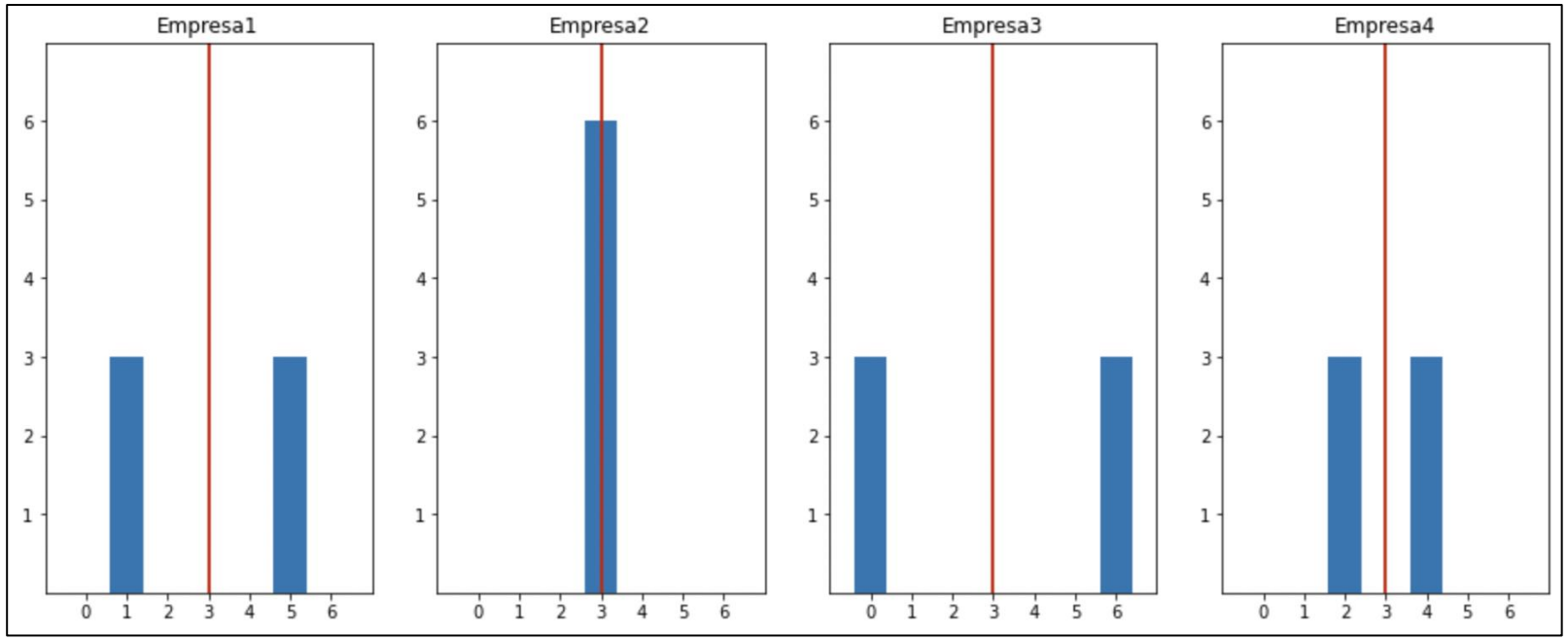
```
2.1908902300206643
```

```
desvio_padrao = base.empresa2.std()
print(desvio_padrao)
```

```
0.0
```

Módulo 10 – Descrição estatística com Pandas

Apenas para reforçar o conceito, vamos adicionar mais duas empresas que, também, possuem média de notas igual a 3. Observe como a **Empresa 3** possui uma maior dispersão que as demais.




```
base.describe()
```

	empresa1	empresa2	empresa3	empresa4
count	6.00000	6.0	6.000000	6.000000
mean	3.00000	3.0	3.000000	3.000000
std	2.19089	0.0	3.286335	1.095445

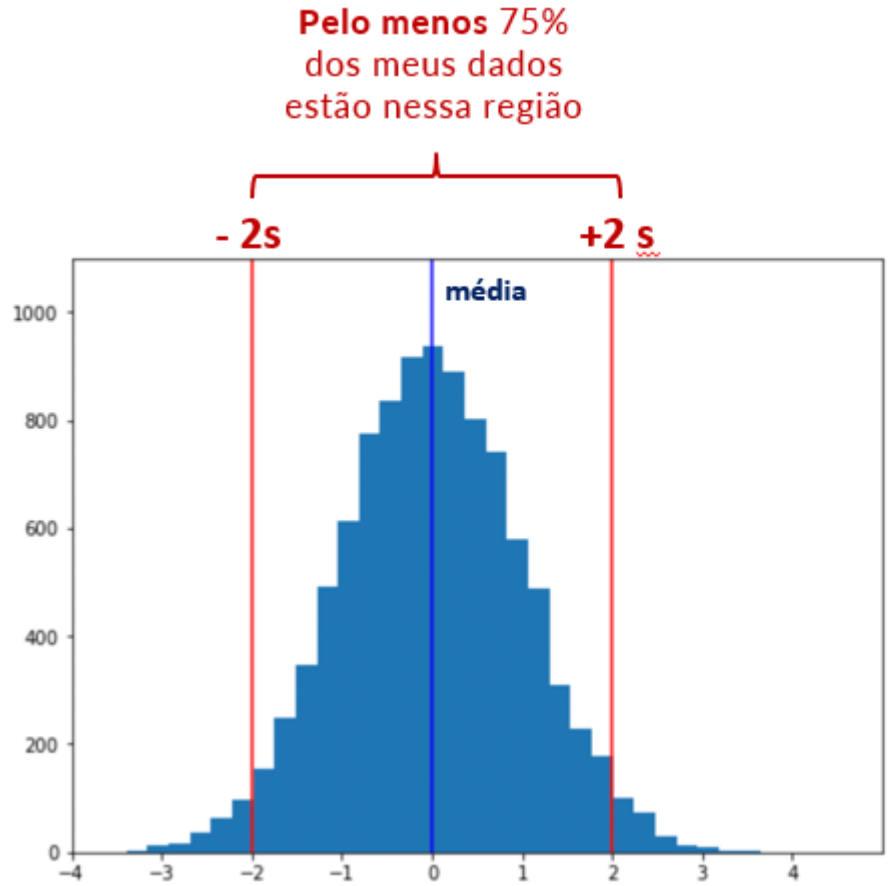
Além das dispersão de valores, o desvio padrão também pode me dar uma ideia de proporção. É sobre isso que fala o **Teorema de Chebyshev**.

Teorema de Chebyshev: para qualquer distribuição de dados, a *proporção mínima* de dados que está a menos de **k** desvios padrões é dada por

$$\% = 1 - \frac{1}{k^2}$$

Vamos ter uma ideia visual deste teorema.

Com base no teorema, temos **pelo menos**, 75 % dos nossos dados entre -2 e 2 desvios padrão:

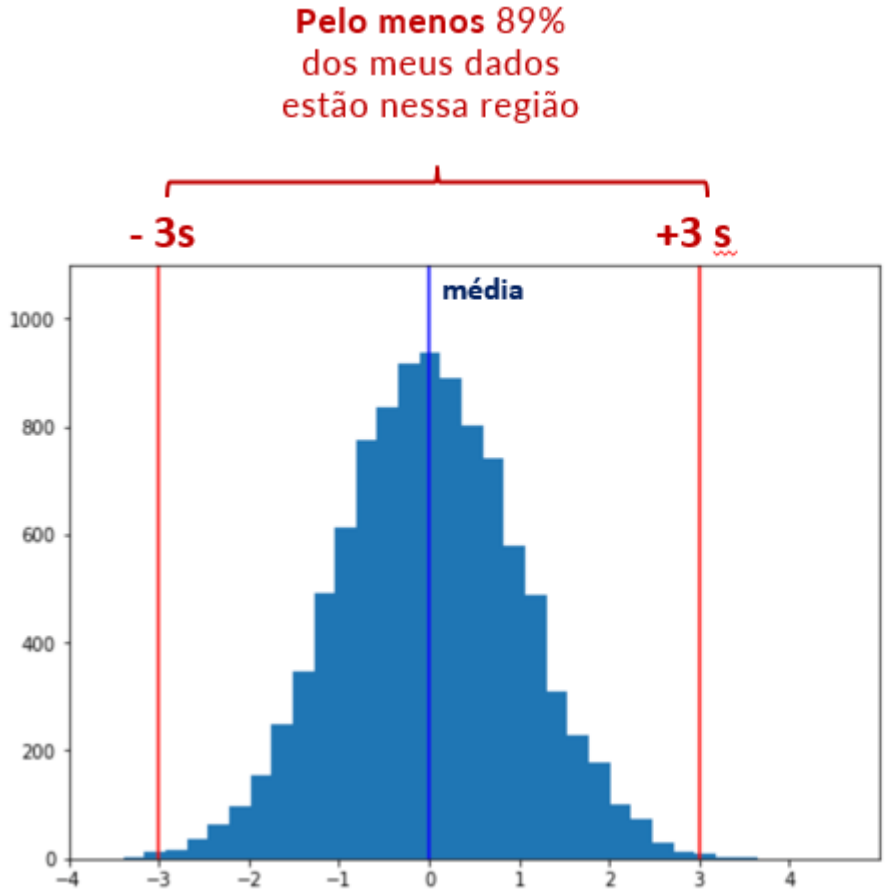


$$k = 2$$

$$\% = 1 - \frac{1}{2^2}$$

$$\% = 75\%$$

Com base no teorema, temos **pelo menos**, 89 % dos nossos dados entre -3 e 3 desvios padrão:



$$k = 3$$

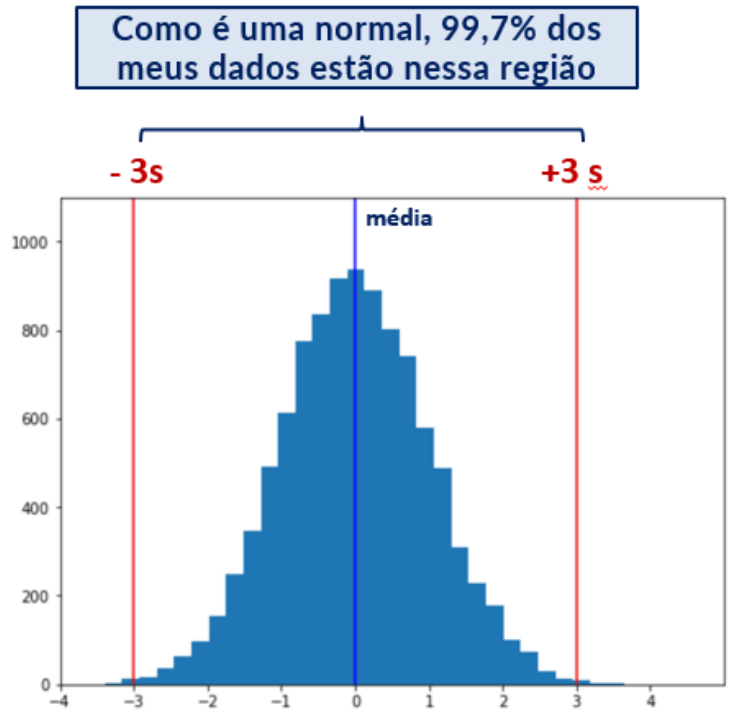
$$\% = 1 - \frac{1}{3^2}$$

$$\% = 89\%$$

Observe que a definição cita “proporção mínima”. E, nas imagens, colocamos “pelo menos”. Isto é, para uma dada distribuição, no mínimo aquela porcentagem será encontrada.

Veremos com frequência distribuições normais. Assim, vejamos os valores realmente encontrados em distribuições normais:

k	% mínima dos dados	% dos dados (normal)
1	-	68,3%
2	75%	95,5%
3	~ 88,89%	99,7%
4	93,75%	
5	96%	
...	...	



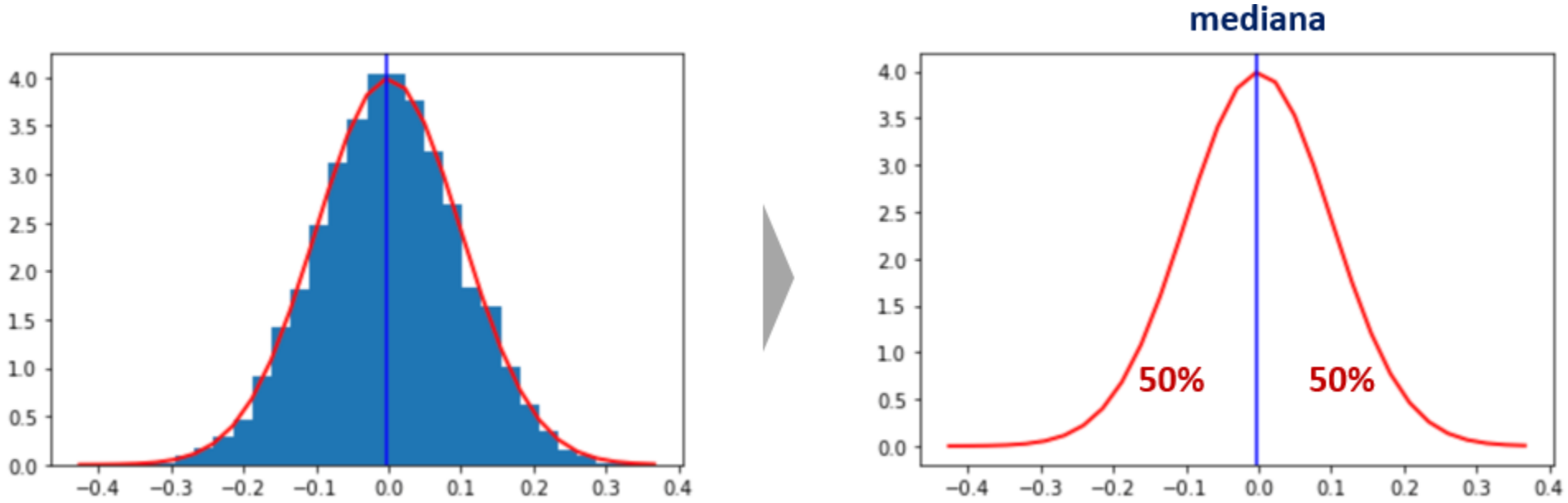
$$k = 3$$

$$\% = 1 - \frac{1}{3^2}$$

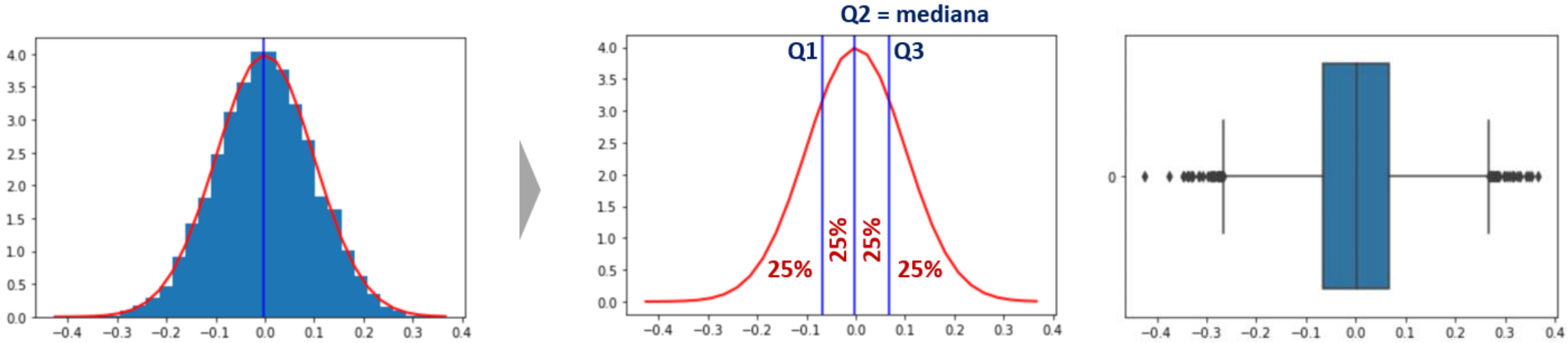
$$\% = 89\%$$

Separatrizes são pontos que dividem o conjunto de dados em partes iguais.

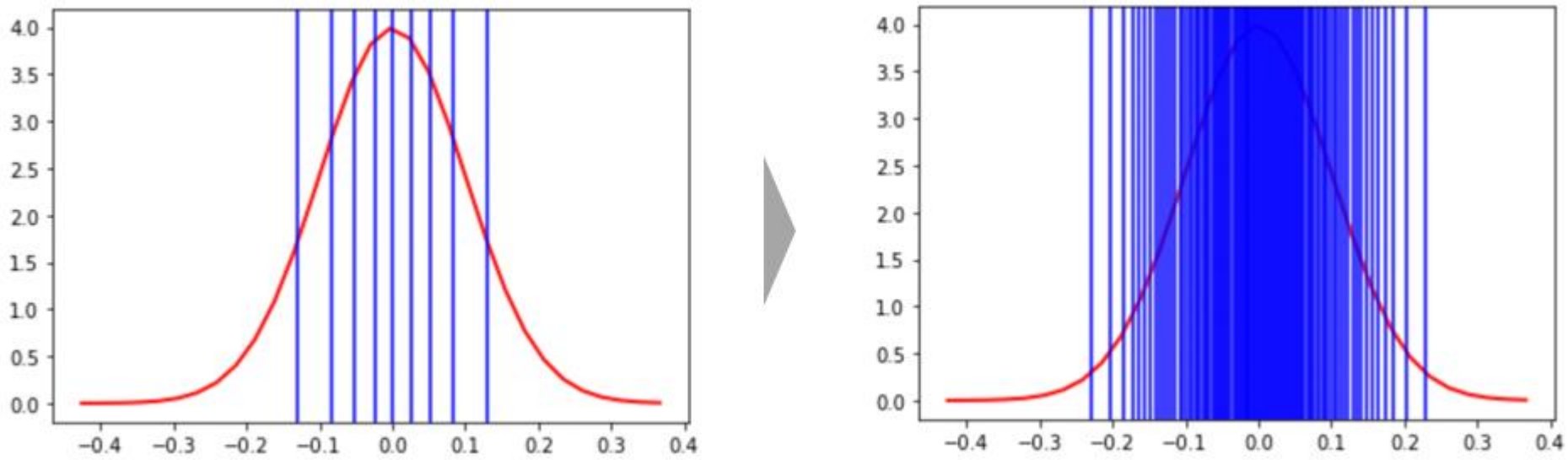
A **mediana** é uma separatriz, que divide o conjunto pela metade:



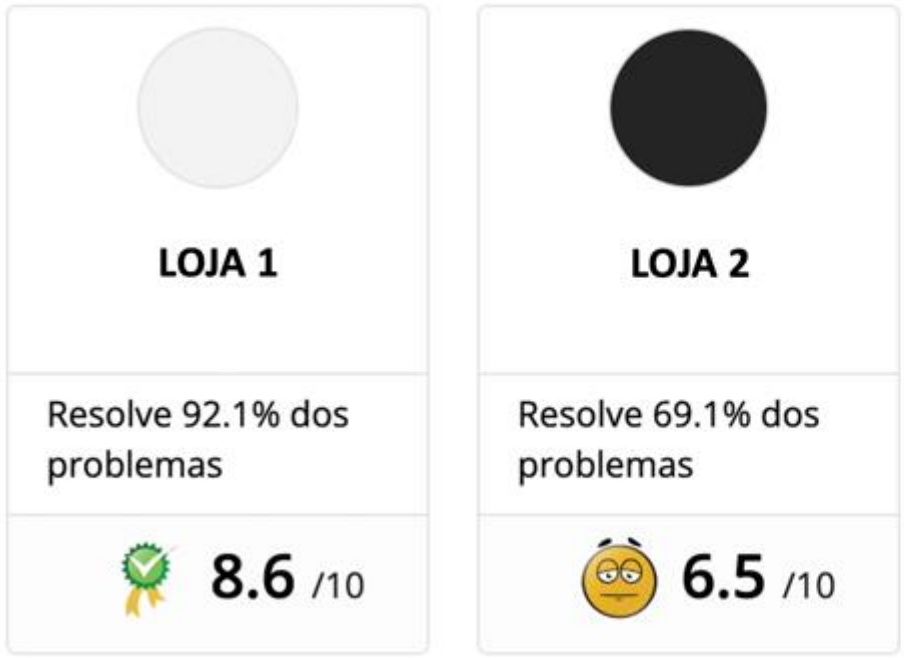
Quartis dividem a base em 4 partes iguais. É um conceito que aparecerá mais vezes por estar intimamente relacionado com os gráficos de boxplot:



Decis dividem a base em 10, **centis** em 100, ...



E agora? Qual empresa? Já vimos o significado de cada entrada que aparece no retorno no `describe`. Com isto, você pode escolher a empresa com base no que prefere: consistência no atendimento, ou a chance de ter um atendimento muito bom ou muito ruim.




```
base.describe()
```

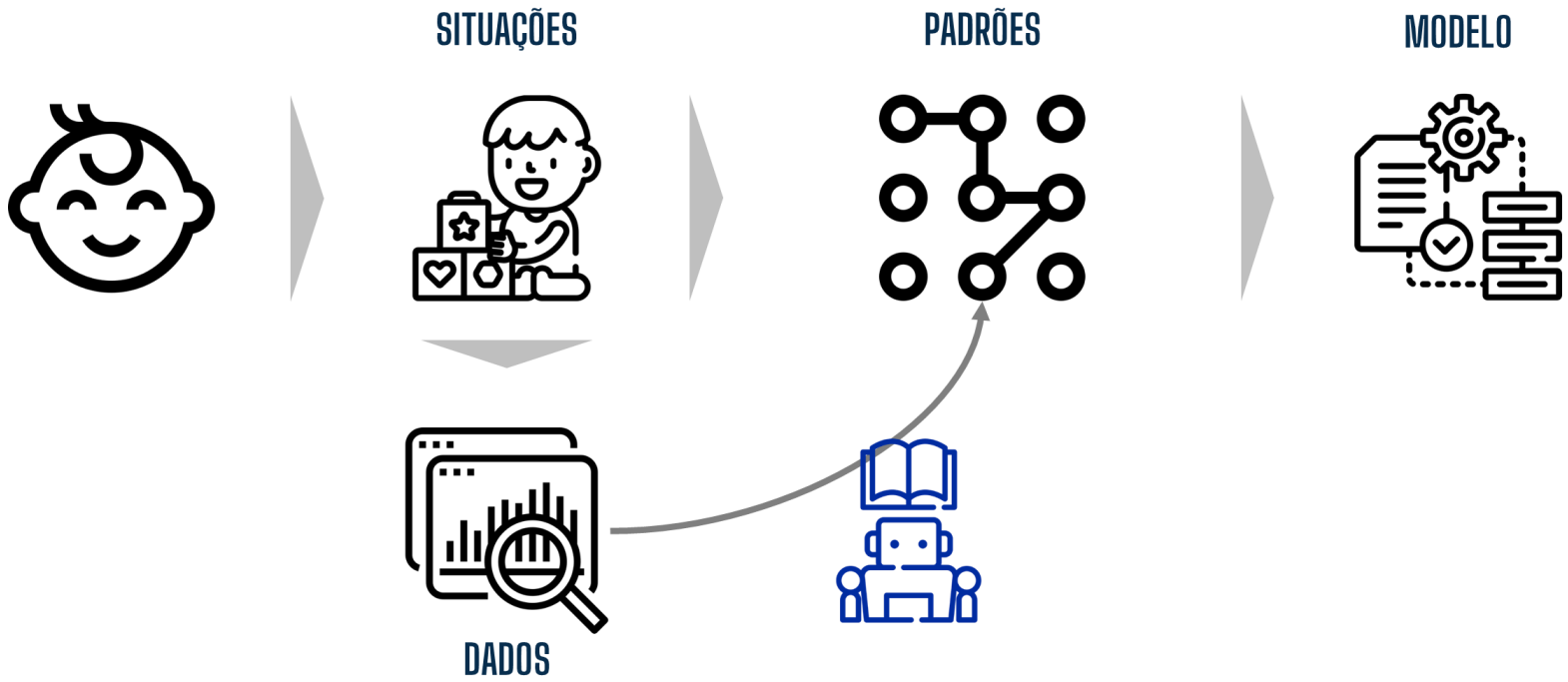
	empresa1	empresa2
count	6.00000	6.0
mean	3.00000	3.0
std	2.19089	0.0
min	1.00000	3.0
25%	1.00000	3.0
50%	3.00000	3.0
75%	5.00000	3.0
max	5.00000	3.0

MÓDULO 11

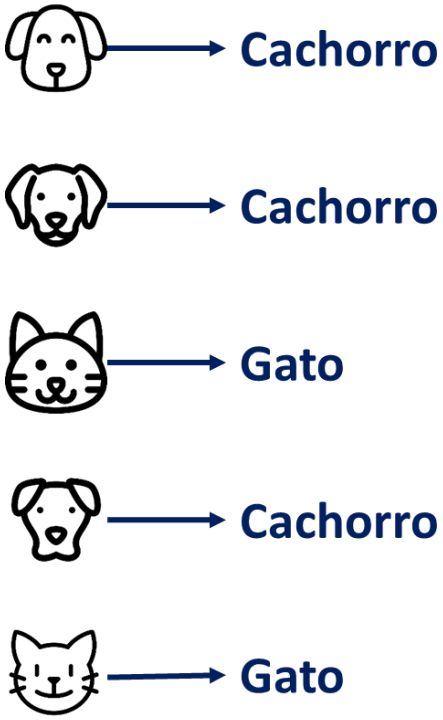
Como as máquinas aprendem?

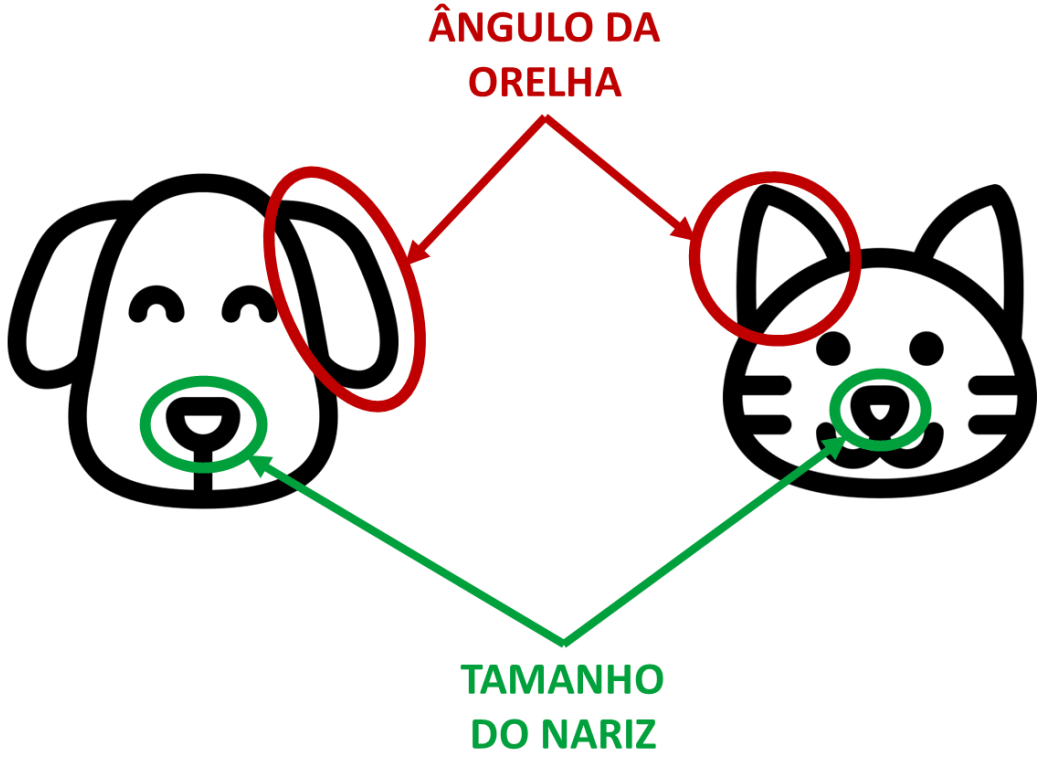
Módulo 11 – Tipos de aprendizado

Como já vimos no módulo anterior, nosso modelo precisa de dados para aprender reconhecendo padrões. Muito similar a um aprendizado de uma criança.



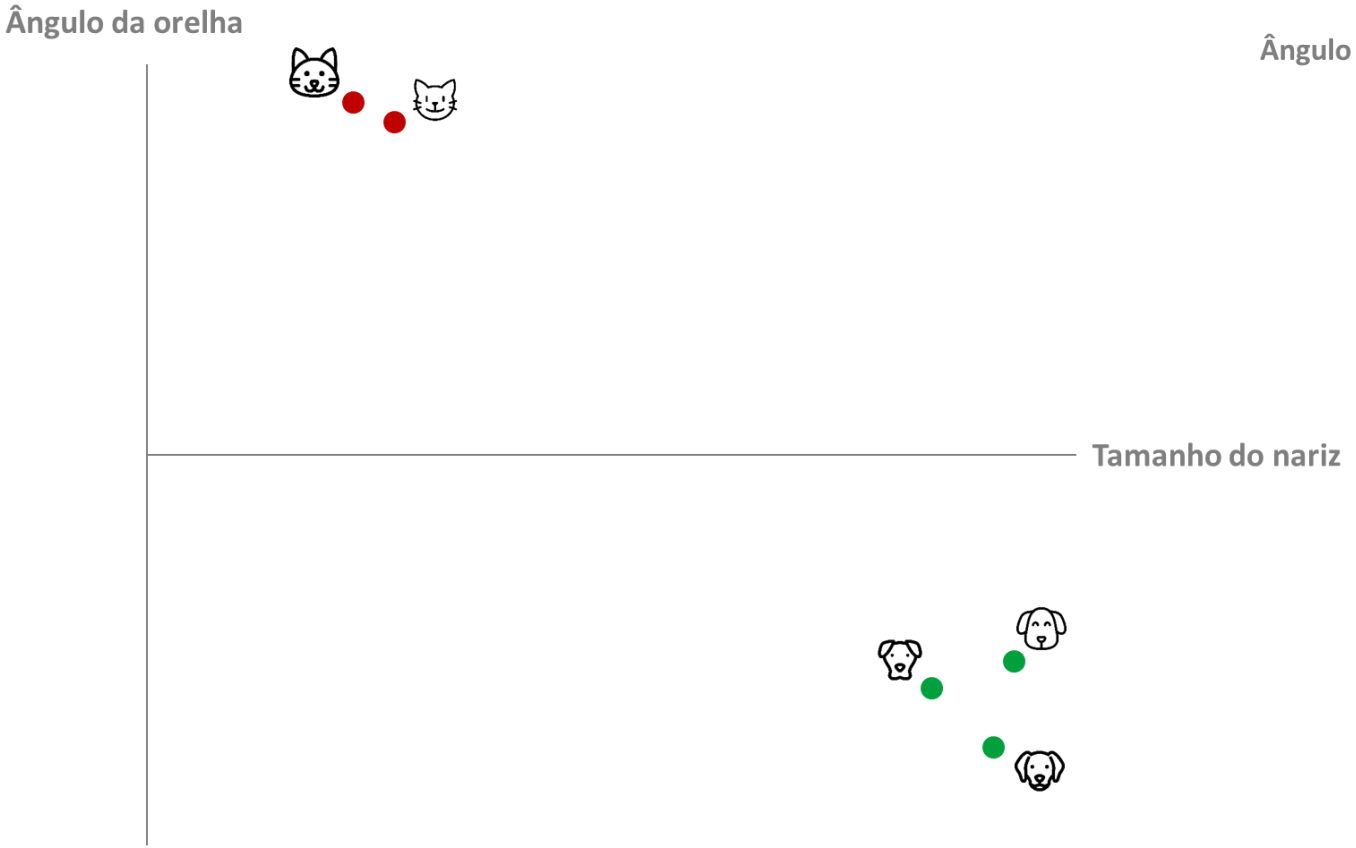
Considere um exemplo onde queremos criar um modelo que classifique corretamente imagens de animais como sendo cachorro ou gato. O modelo pode encontrar padrões nas imagens como, por exemplo, o formato da orelha e o tamanho do nariz:

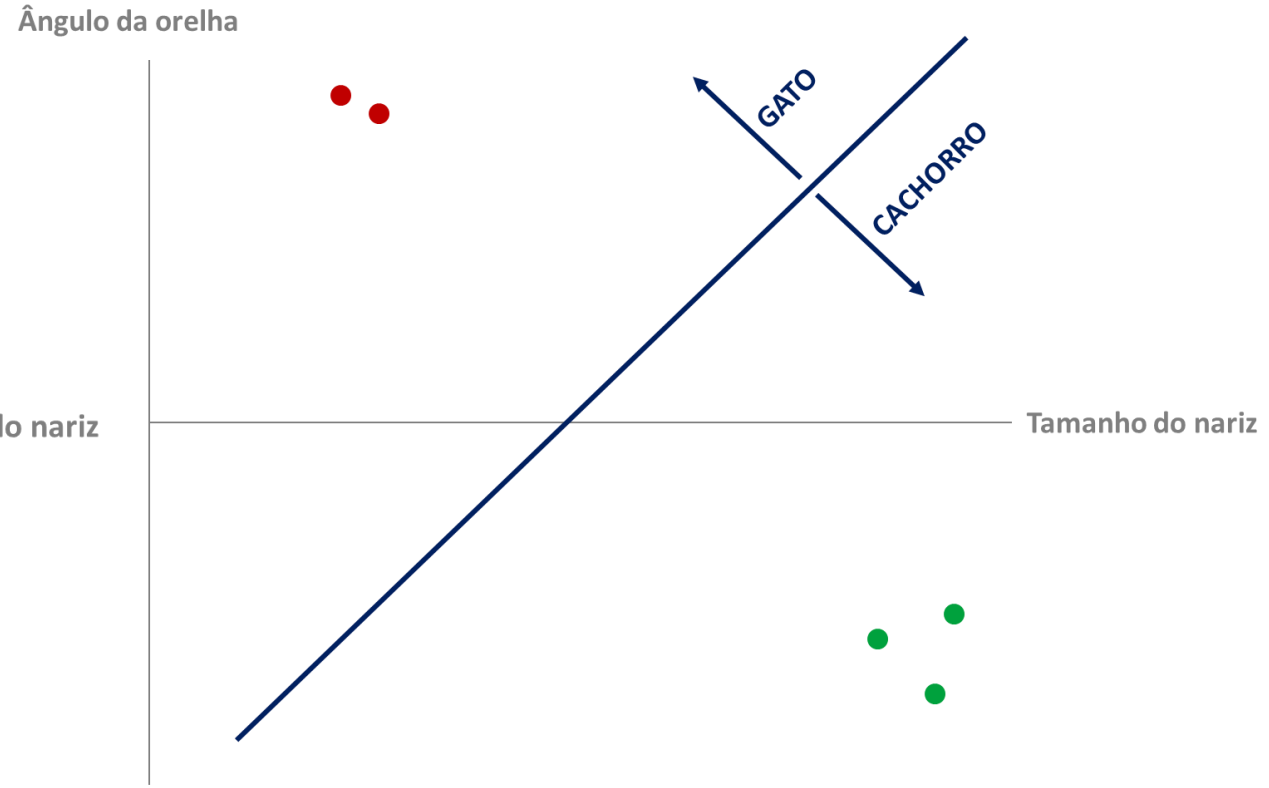




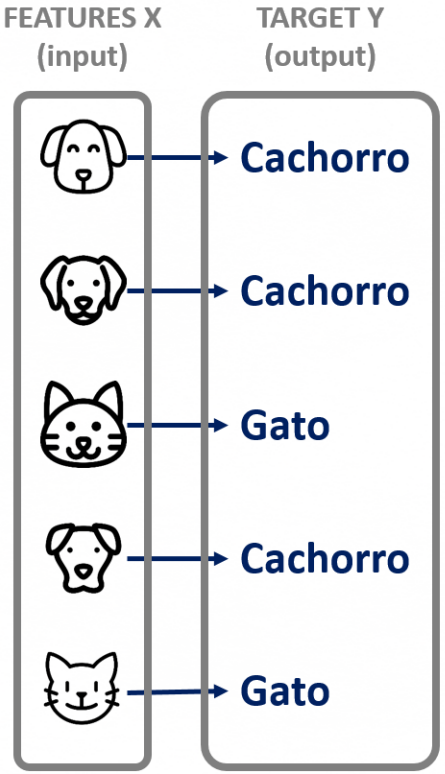
Módulo 11 – Tipos de aprendizado – Aprendizado supervisionado

Com base nestes padrões, o modelo pode definir um limite, visualizado como uma reta abaixo, separando o que ele classificará como cachorro e o que será classificado como gato.





Repare, nesse nosso exemplo, que os dados de partida são rotulados. Desde o início nosso modelo já sabe que há duas classificações possíveis e fornecemos exemplos de cada uma das classificações ao modelo. Esse tipo de aprendizado é chamado de **supervisionado**.

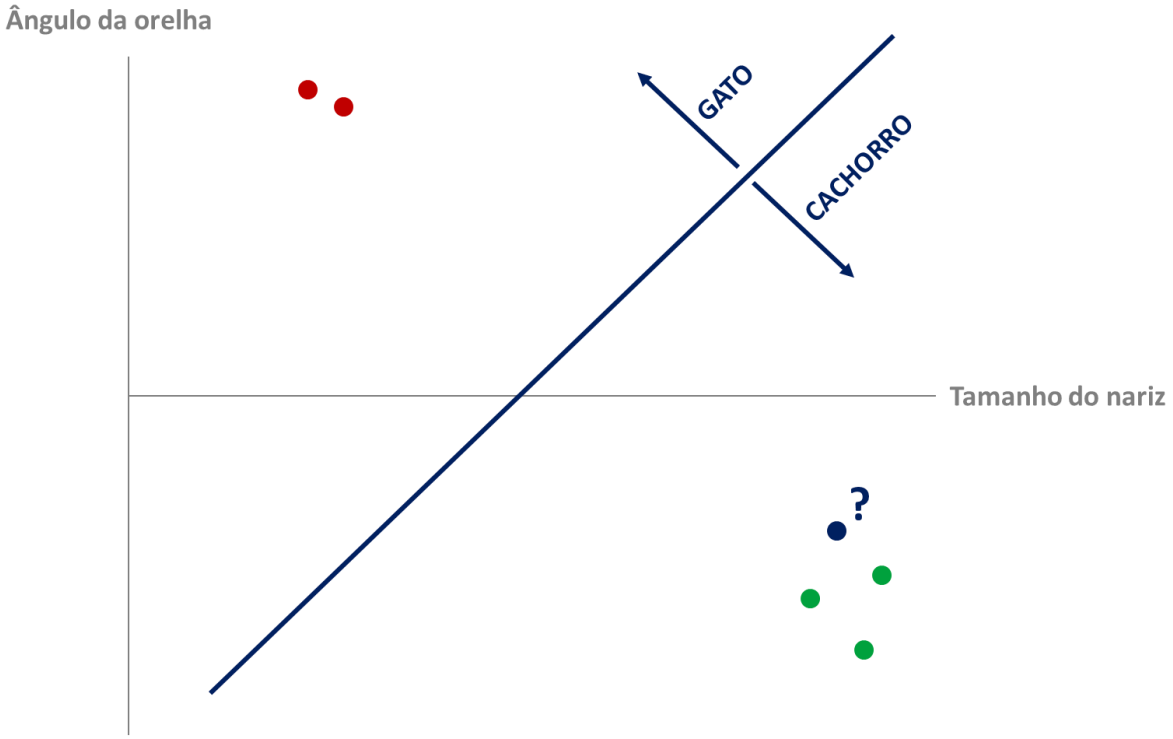


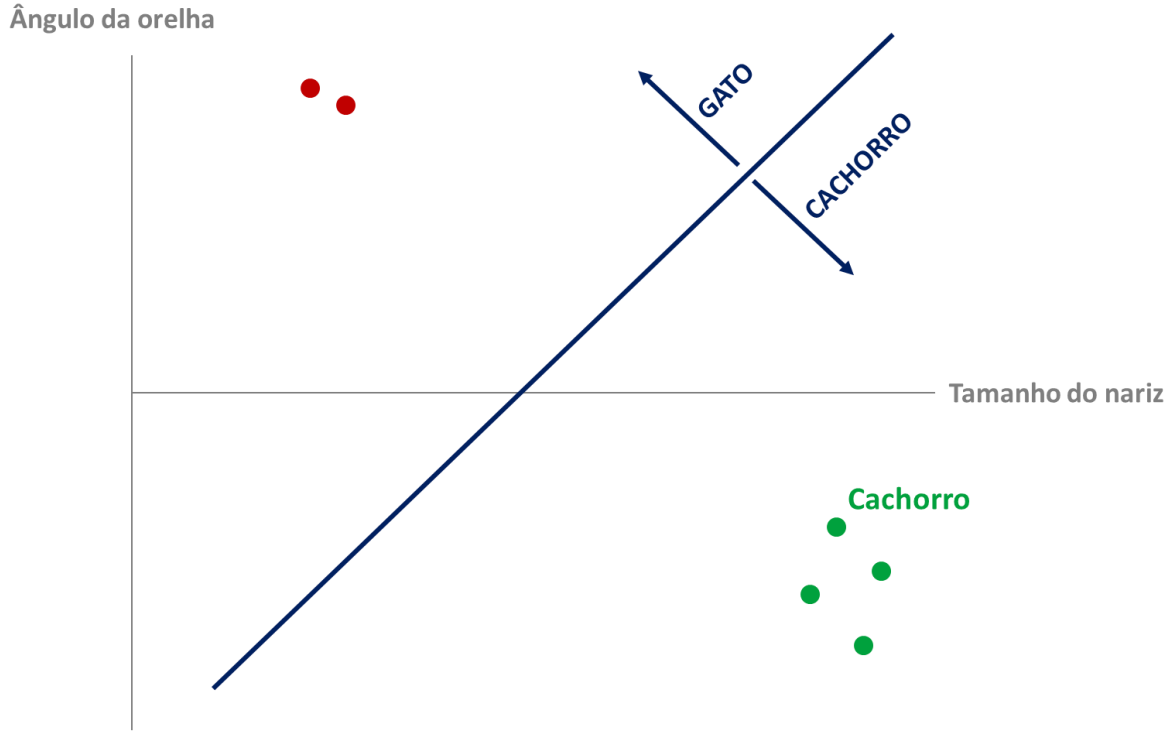
OS DADOS JÁ ESTÃO ROTULADOS!

(eu já digo o que é gato e o que é cachorro)

APRENDIZADO SUPERVISIONADO

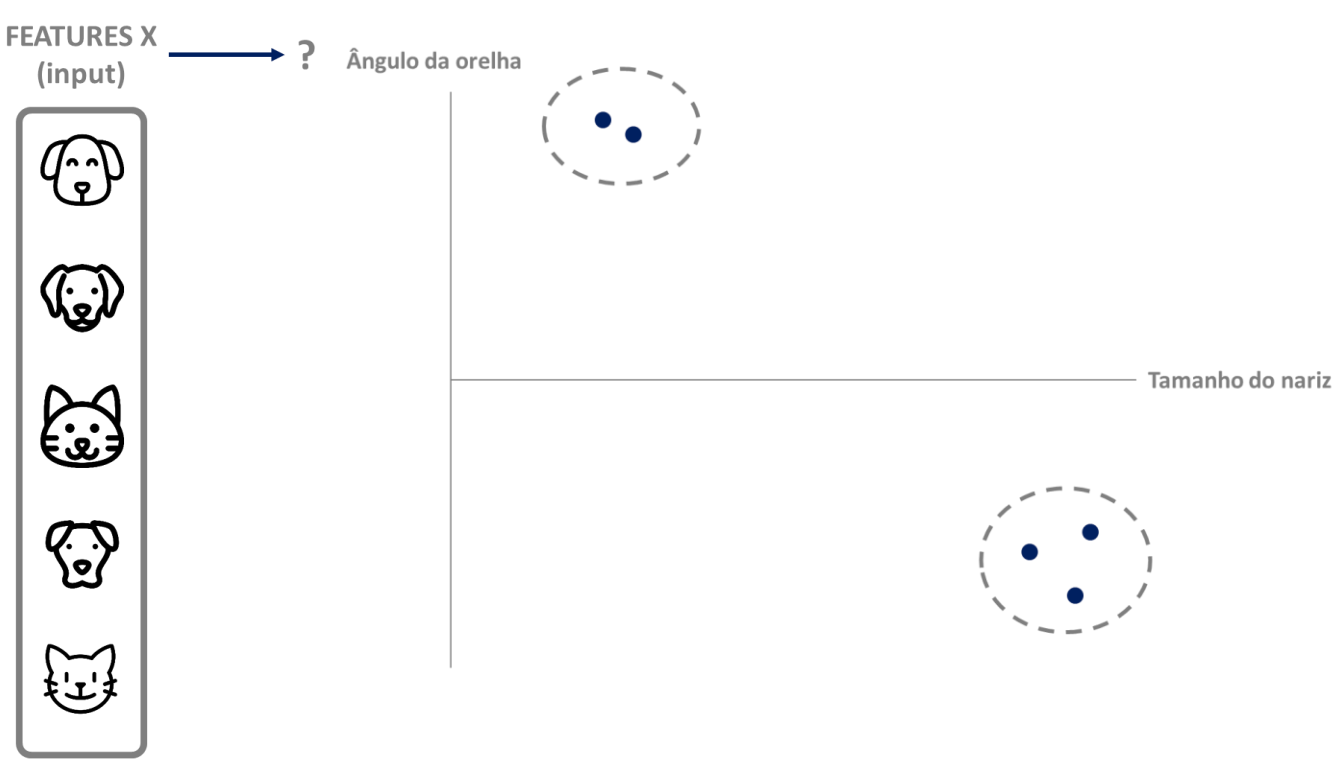
Neste caso, ao receber uma nova imagem, que não estava dentre as que foram usadas para treinar o modelo, o modelo reconhecerá os padrões na imagem e já será capaz de fazer a classificação:

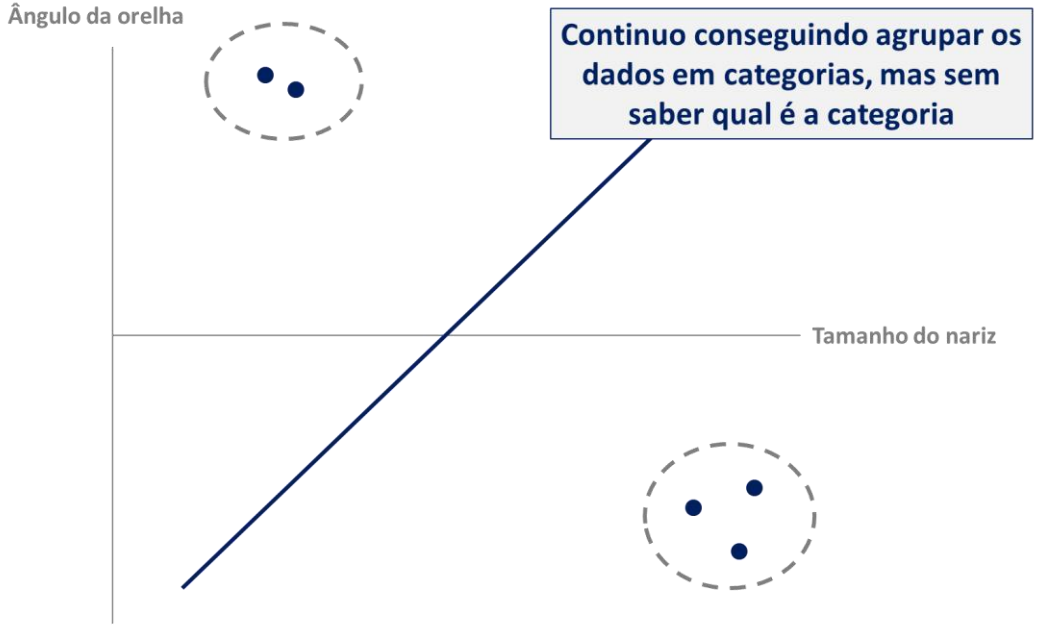




Módulo 11 – Tipos de aprendizado – Aprendizado não supervisionado

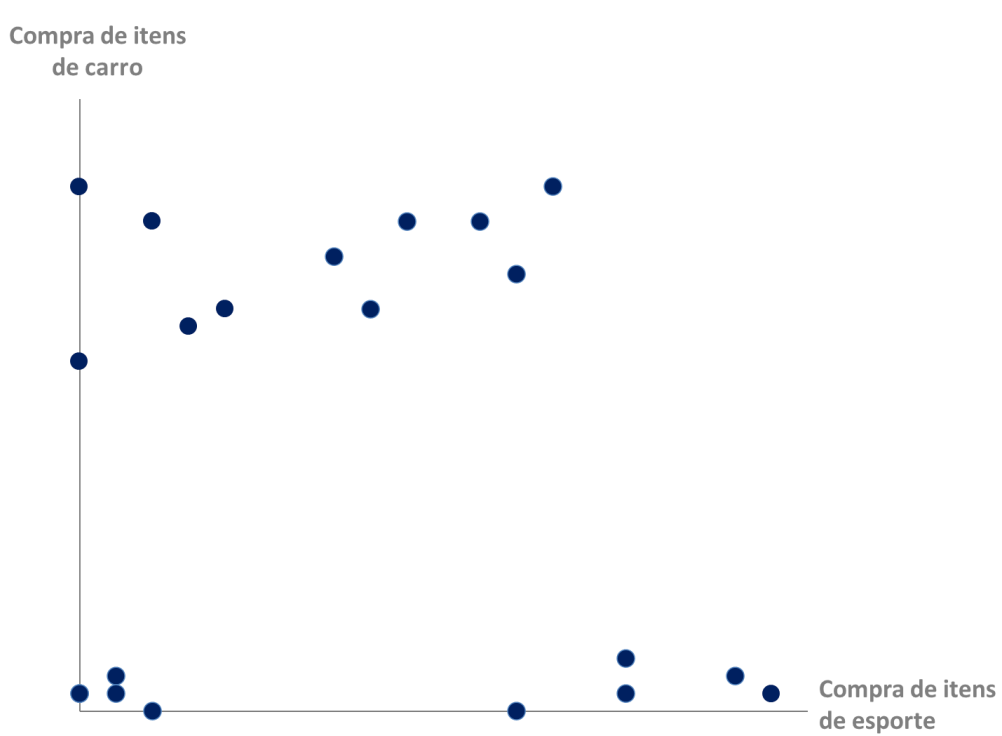
Agora, vamos supor uma situação onde não passamos ao nosso modelo as classificações possíveis. O modelo ainda será capaz de reconhecer padrões e agrupar os dados. Mas não saberá o nome dos grupos.

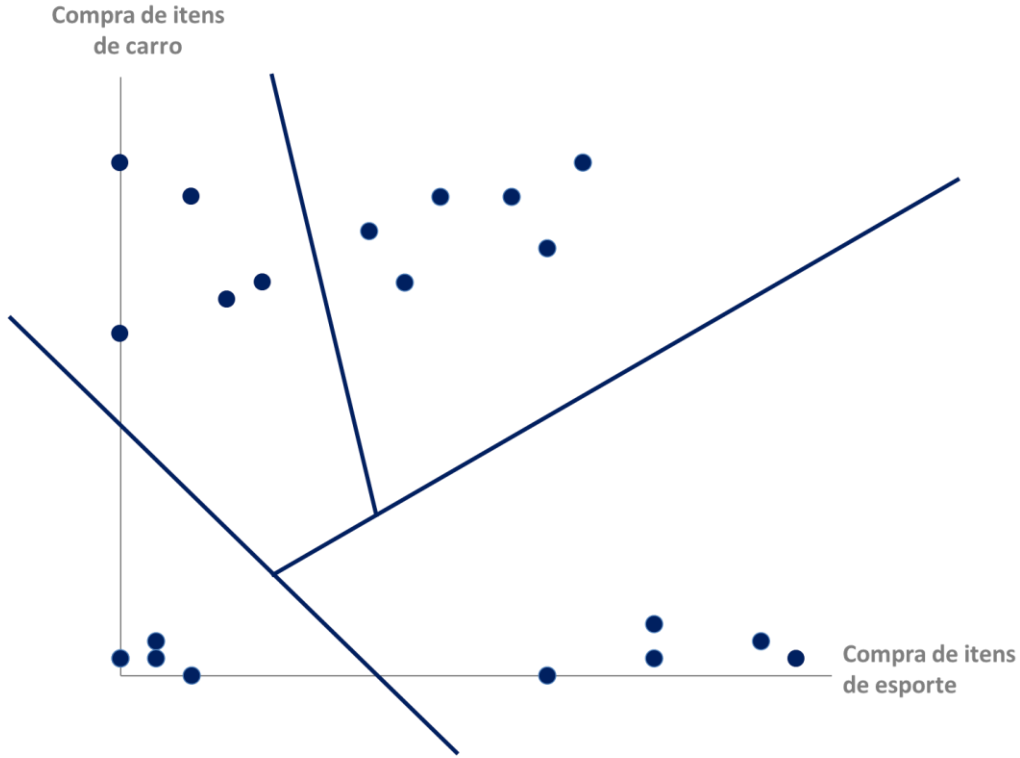




Módulo 11 – Tipos de aprendizado – Aprendizado não supervisionado

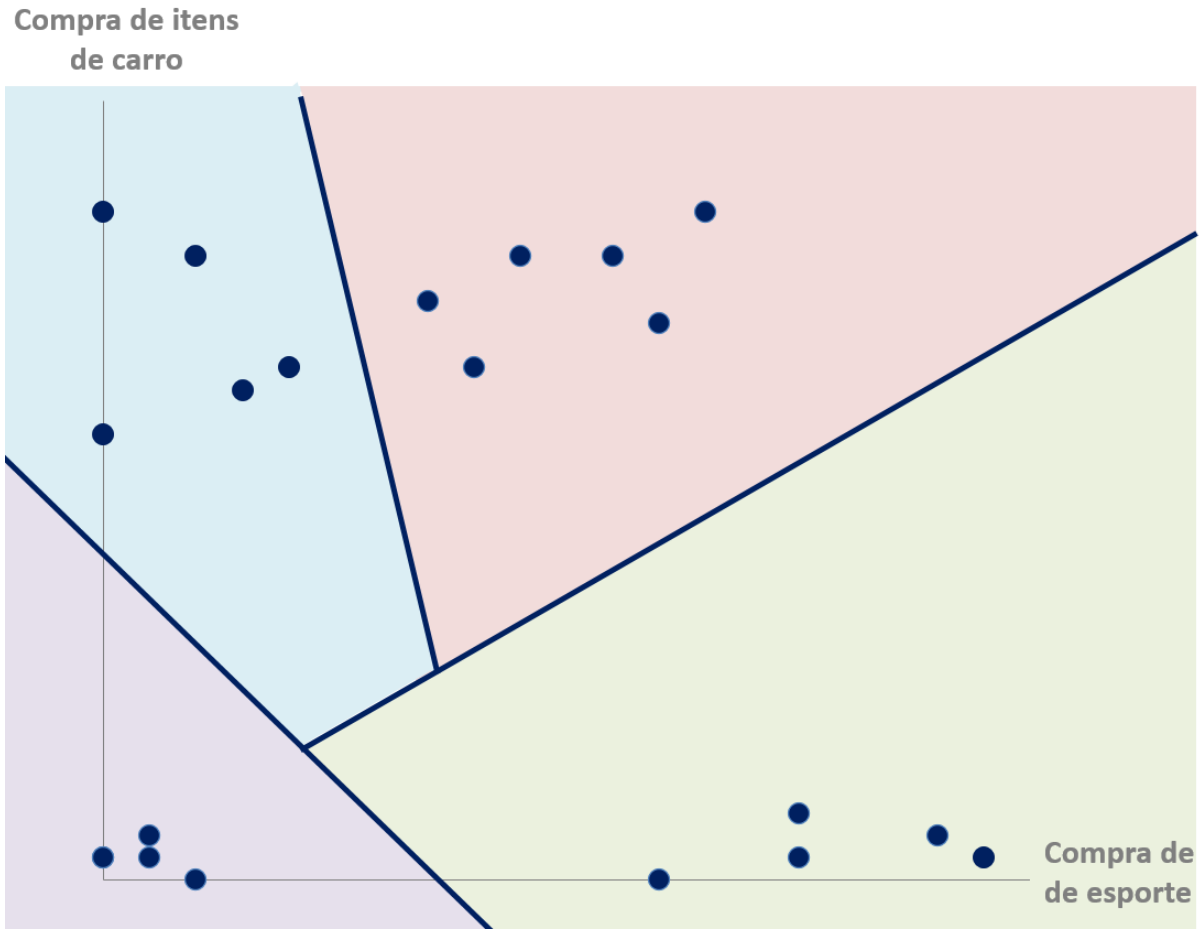
Na vida real, não saber exatamente uma classificação é comum. Por exemplo, avaliando o perfil de compra de diversos clientes, em um primeiro momento podemos não saber como rotulá-los. E, na prática, o importante é saber reconhecer grupos de clientes para entender seus perfis e o que pode ser oferecido a cada grupo. Na figura abaixo, o modelo inicialmente dividiu em 4 grupos:





Módulo 11 – Tipos de aprendizado – Aprendizado não supervisionado

Com esses grupos criados pelo modelo, podemos tentar entender as relações dentro de cada grupo

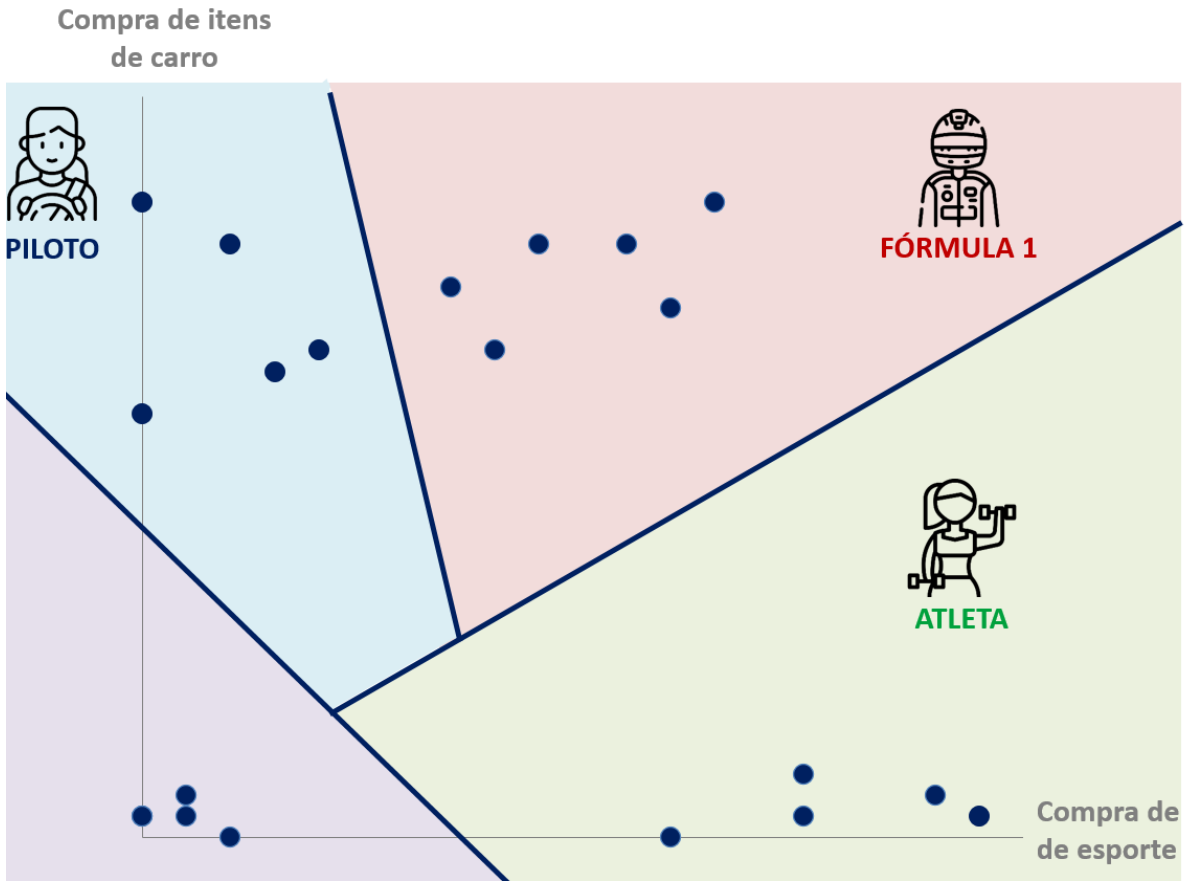


Eu não sei quem é cada um dos clientes, mas eu consigo criar relação entre eles!

(na verdade, eu nem preciso dessa informação, apenas a relação entre os clientes já é útil)

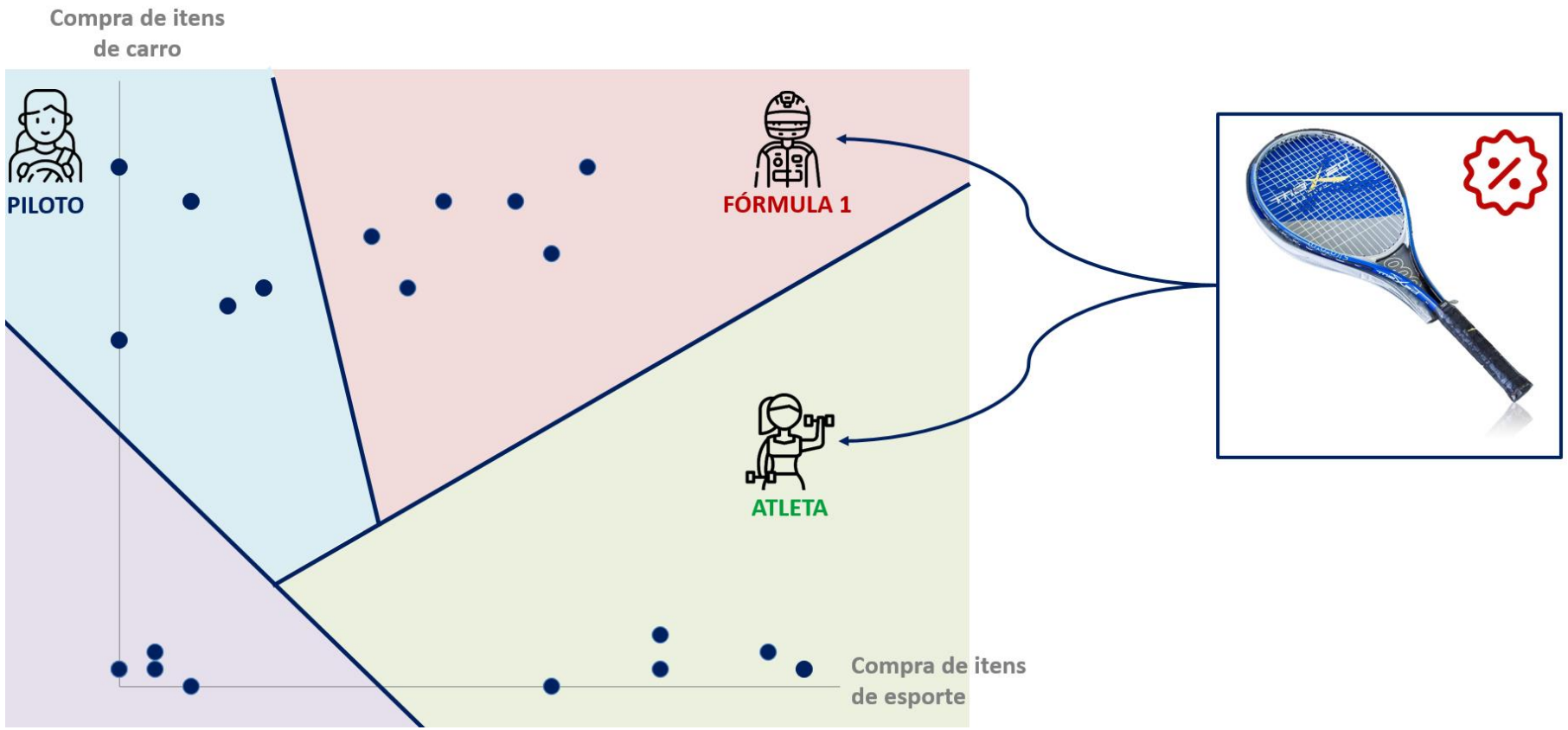
Módulo 11 – Tipos de aprendizado – Aprendizado não supervisionado

Pode ser interessante associar personas a cada grupo, com base nas relações que vão sendo percebidas após o modelo ter feito o agrupamento:



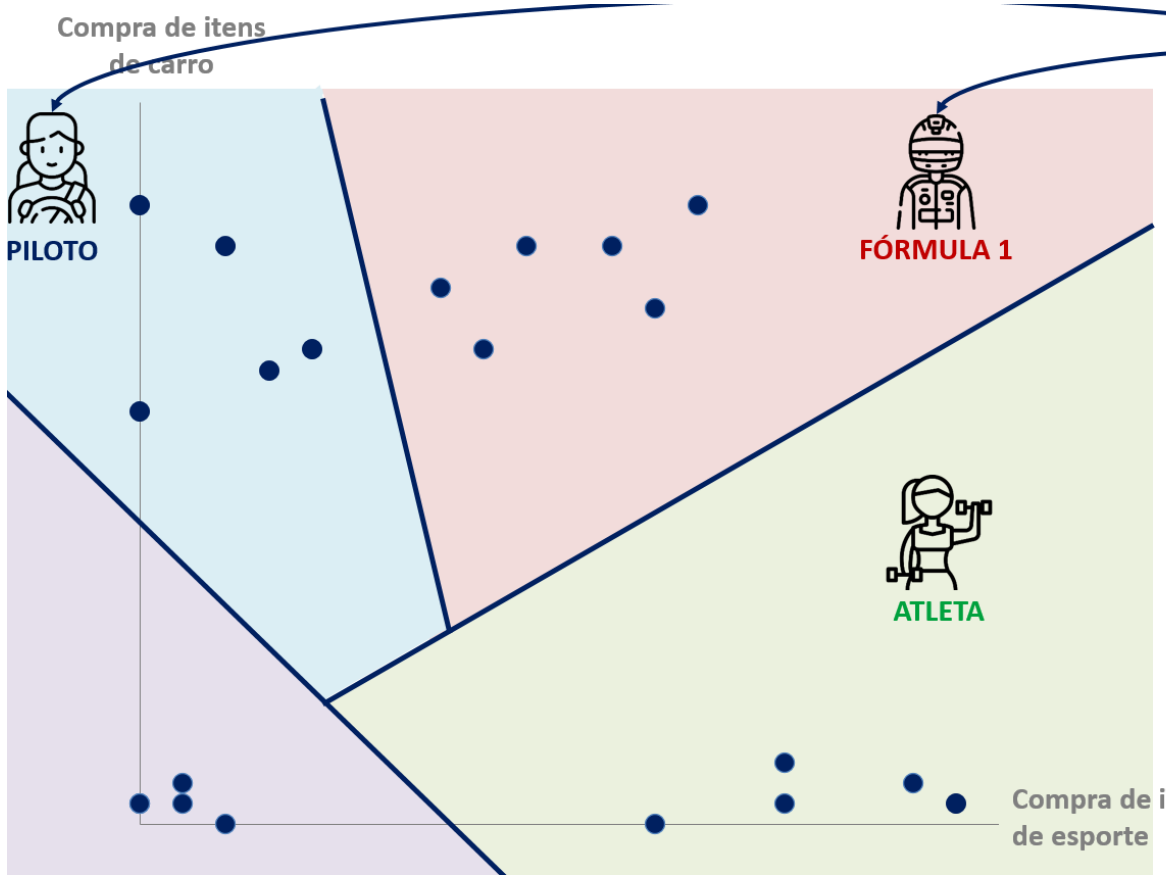
Podemos criar personas utilizando esses clusters de clientes, e usar isso para fazer recomendações de conteúdo / produto

Essas personas podem ajudar nas sugestões de produtos. Tanto um atleta quanto um piloto de Fórmula 1 podem se interessar por uma raquete, pelo perfil esportivo de ambos:

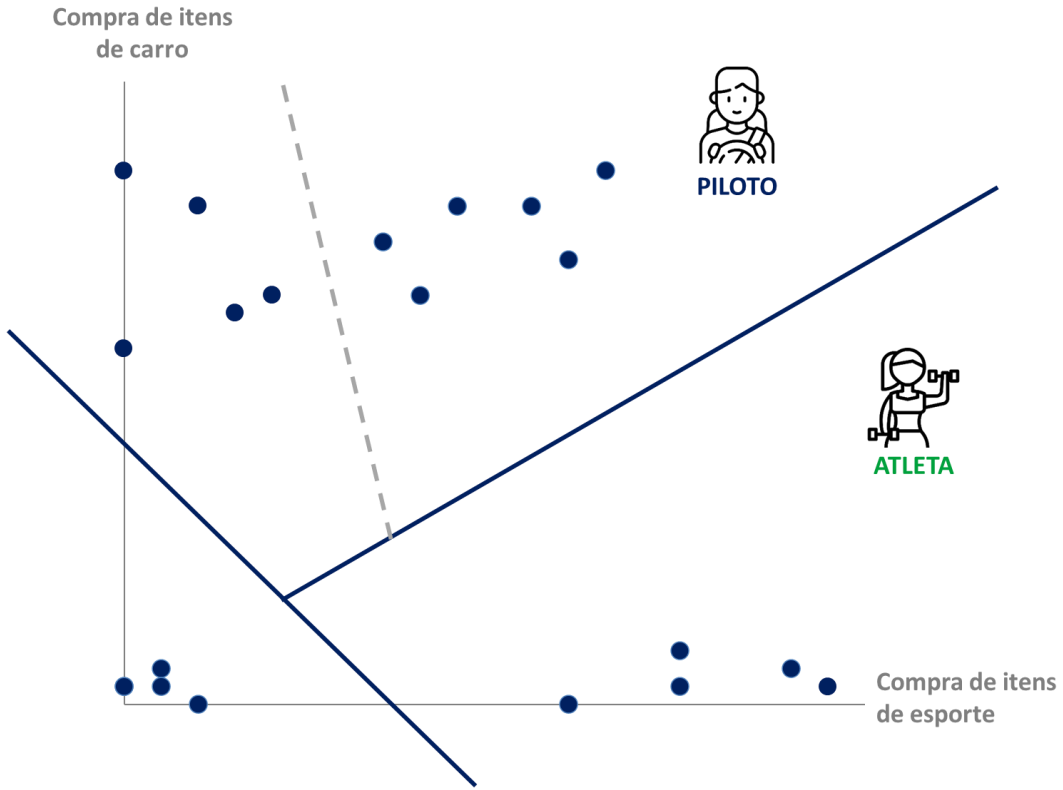


Módulo 11 – Tipos de aprendizado – Aprendizado não supervisionado

Já um piloto de Fórmula 1 e um de outra categoria podem se interessar por pneus de alta performance:

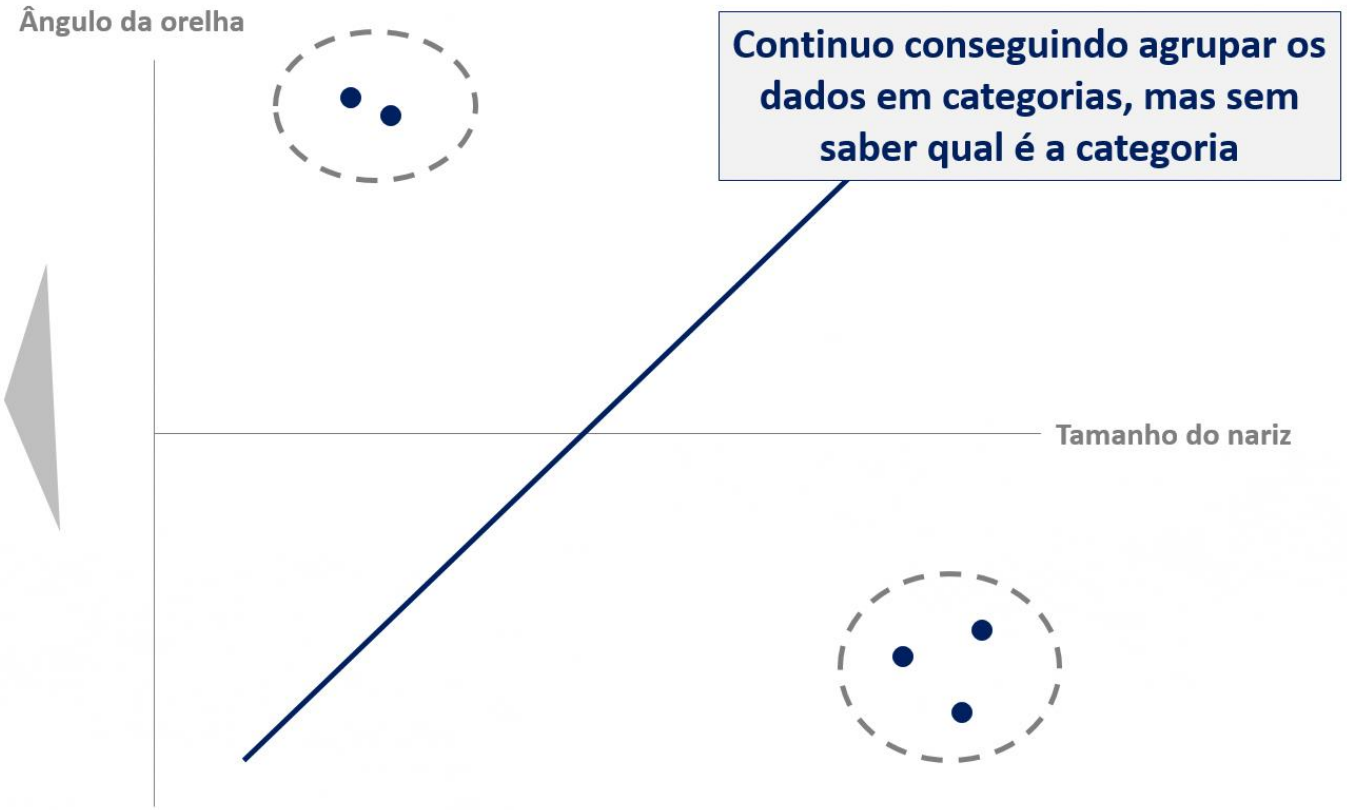


Inclusive, podemos instruir nosso modelo a não separar estes dois últimos grupos, considerando apenas uma classificação única como “piloto”. Esta decisão pode ser tomada com base em testes, avaliando se há alteração na performance de indicações e vendas com base na modificação de grupos.



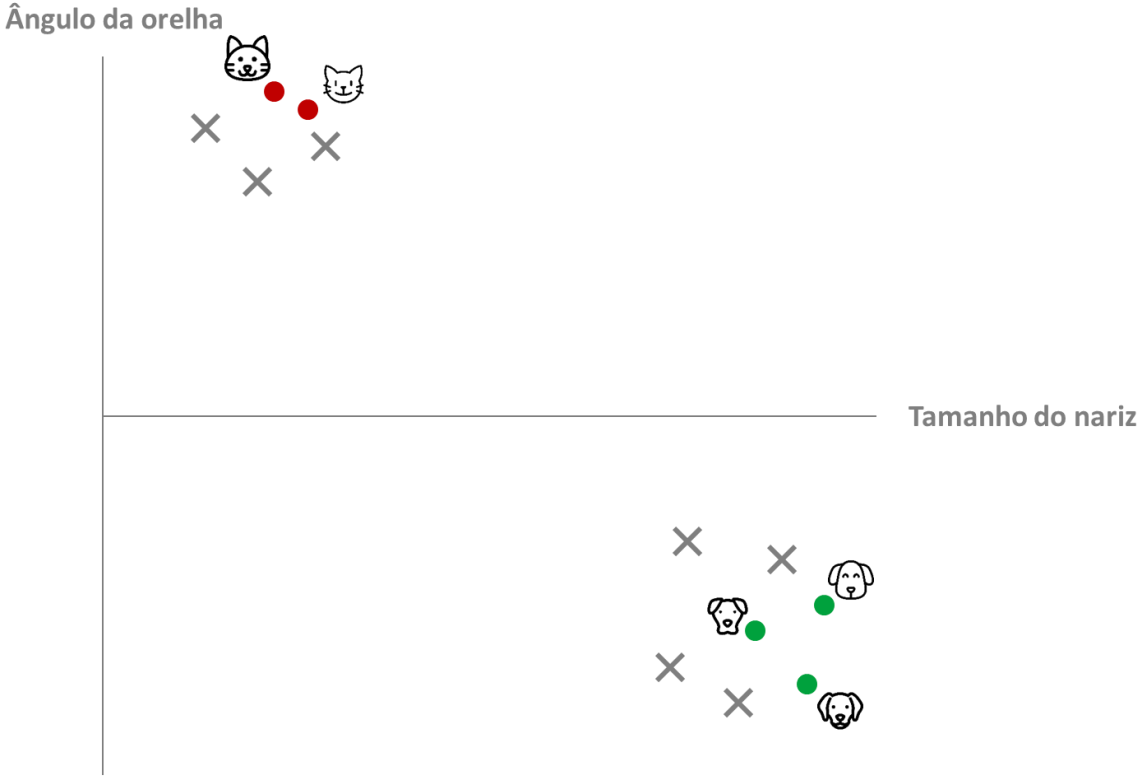
Este exemplo é o que chamamos de **aprendizado não supervisionado**. Quando delegamos ao modelo a classificação, sem passar rótulos previamente.

APRENDIZADO NÃO SUPERVISIONADO



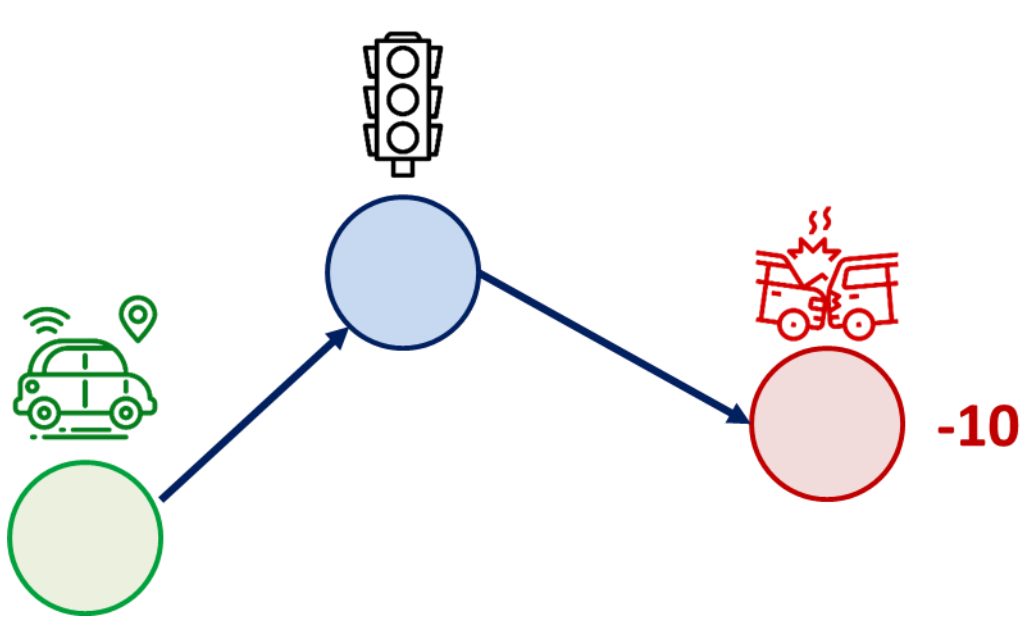
Módulo 11 – Tipos de aprendizado – Aprendizado semi-supervisionado

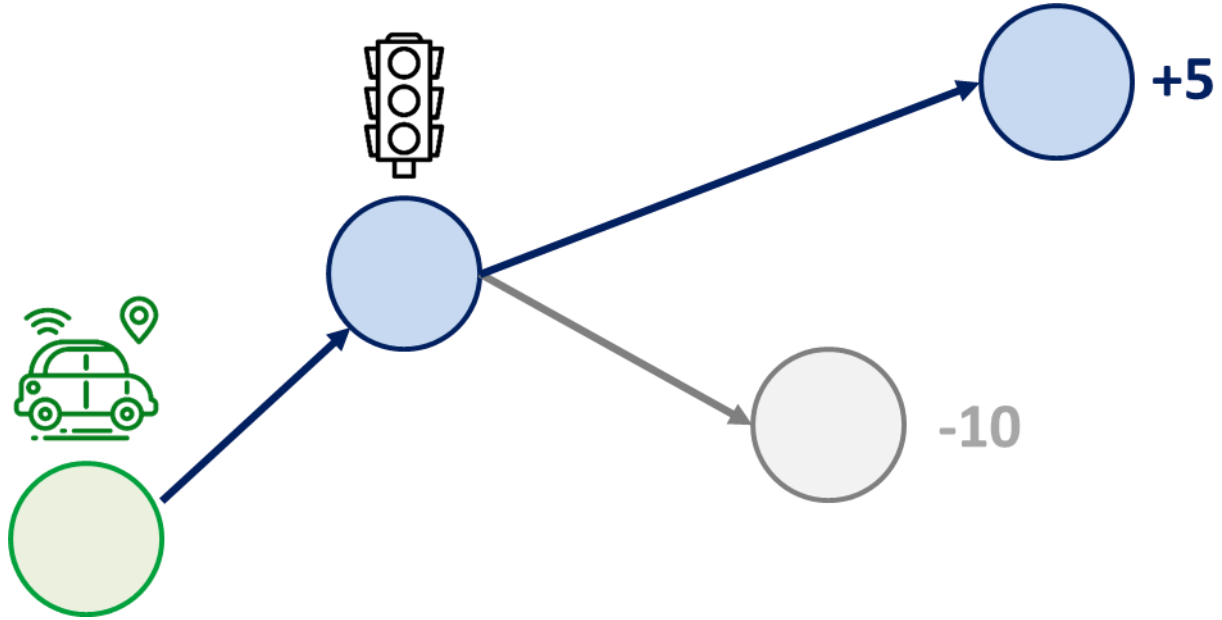
No **aprendizado semi-supervisionado** temos alguns dados rotulados e outros não rotulados. Este tipo de aprendizado é comum em bases muito grandes. Por exemplo, de imagens, onde não se tem meios de rotular toda a base de treino. Rotula-se alguns e outros são deixados sem rótulo.



Módulo 11 – Tipos de aprendizado – Aprendizado por reforço

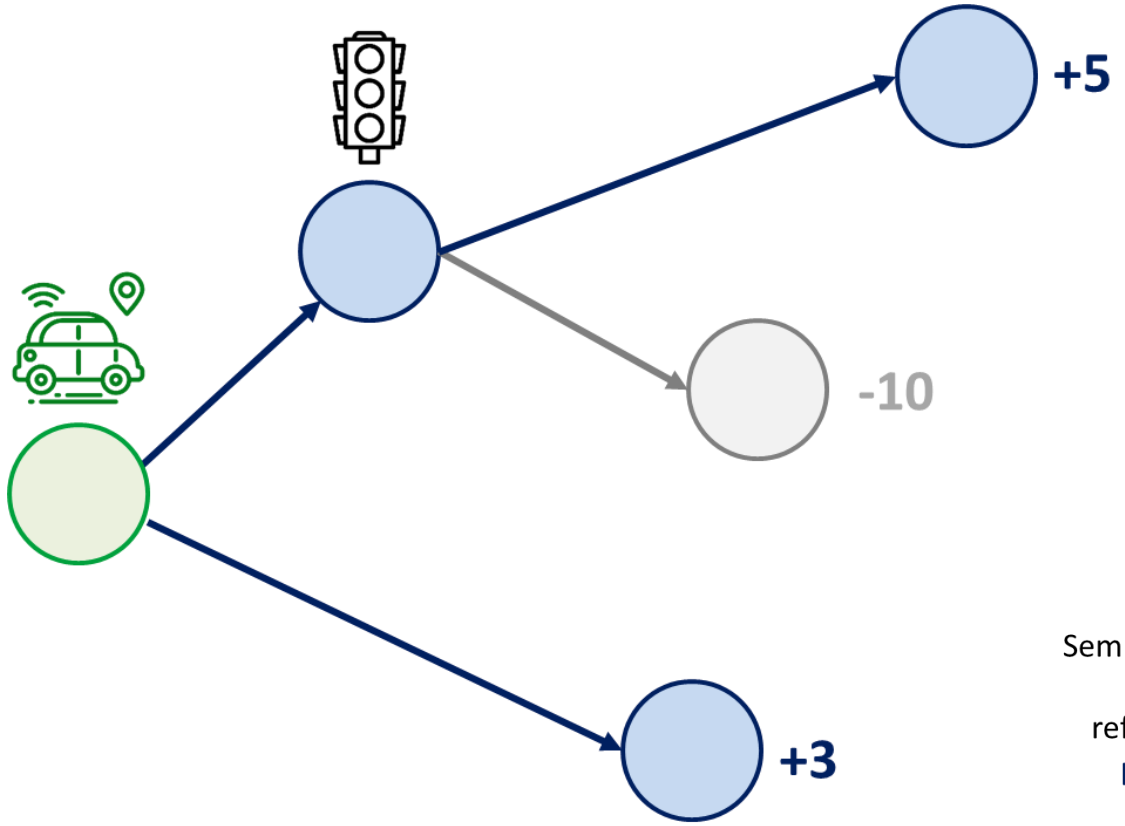
O **aprendizado por reforço** é aquele que vai aprendendo por bonificação ou punição de acordo com escolhas. Por exemplo, um carro autônomo. Na figura, quando um carro atravessa o sinal vermelho, acaba batendo e isso é penalizado no modelo. Quando para no sinal e só avança no verde, sem acidentes, o modelo armazena como algo positivo.

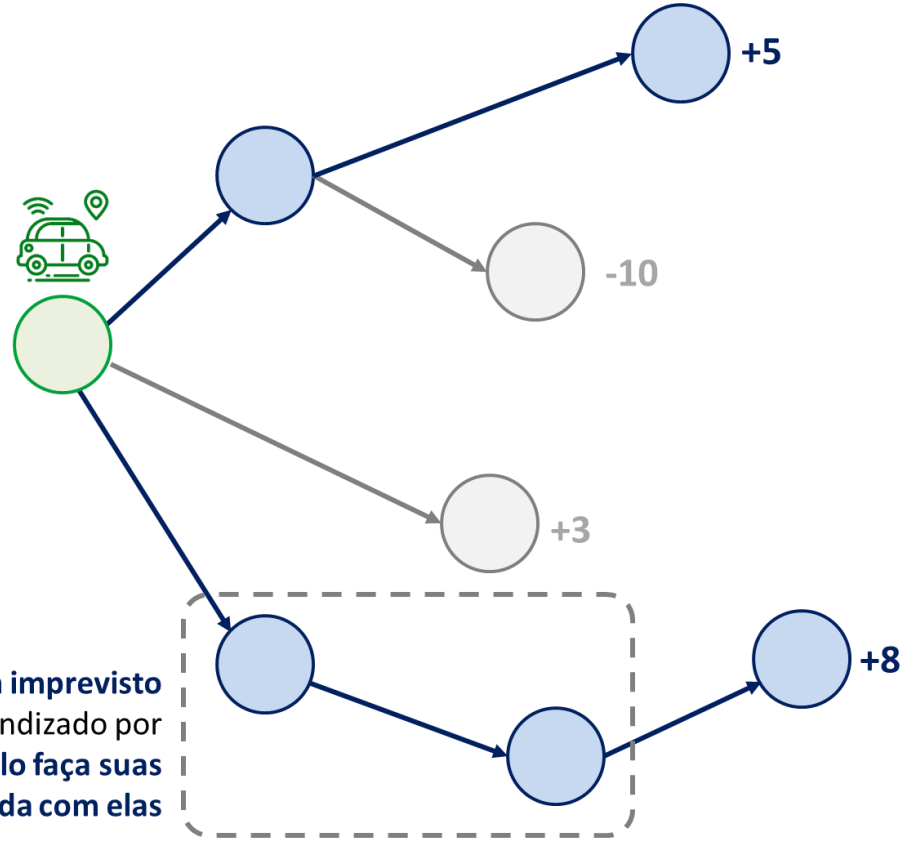




Módulo 11 – Tipos de aprendizado – Aprendizado por reforço

Durante a fase de treinos, o modelo vai fazendo diversas escolhas, diversos caminhos, e sendo bonificado ou penalizado de acordo. O modelo treinado vai sempre buscar o caminho com maior pontuação.

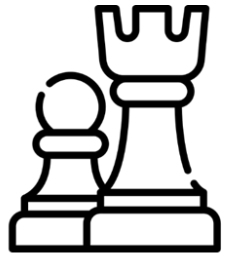




Sempre vai poder surgir **algum imprevisto** na estrada, então o aprendizado por reforço permite que **o modelo faça suas próprias escolhas e aprenda com elas**

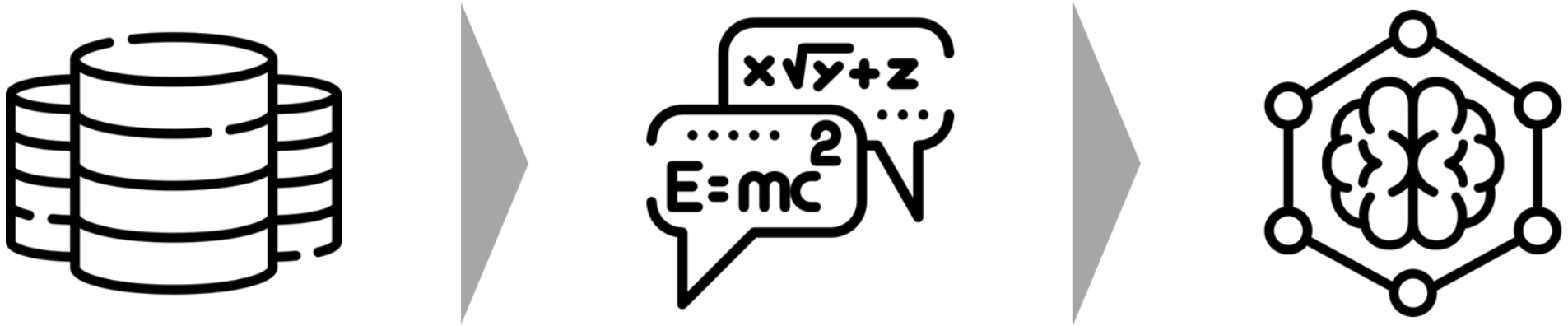
Módulo 11 – Tipos de aprendizado – Aprendizado por reforço

Esse tipo de aprendizado é comum em jogos, como xadrez; em sistemas de determinação de rotas de trânsito; em sistemas de feed de notícias; e em programas de análise de mercados.



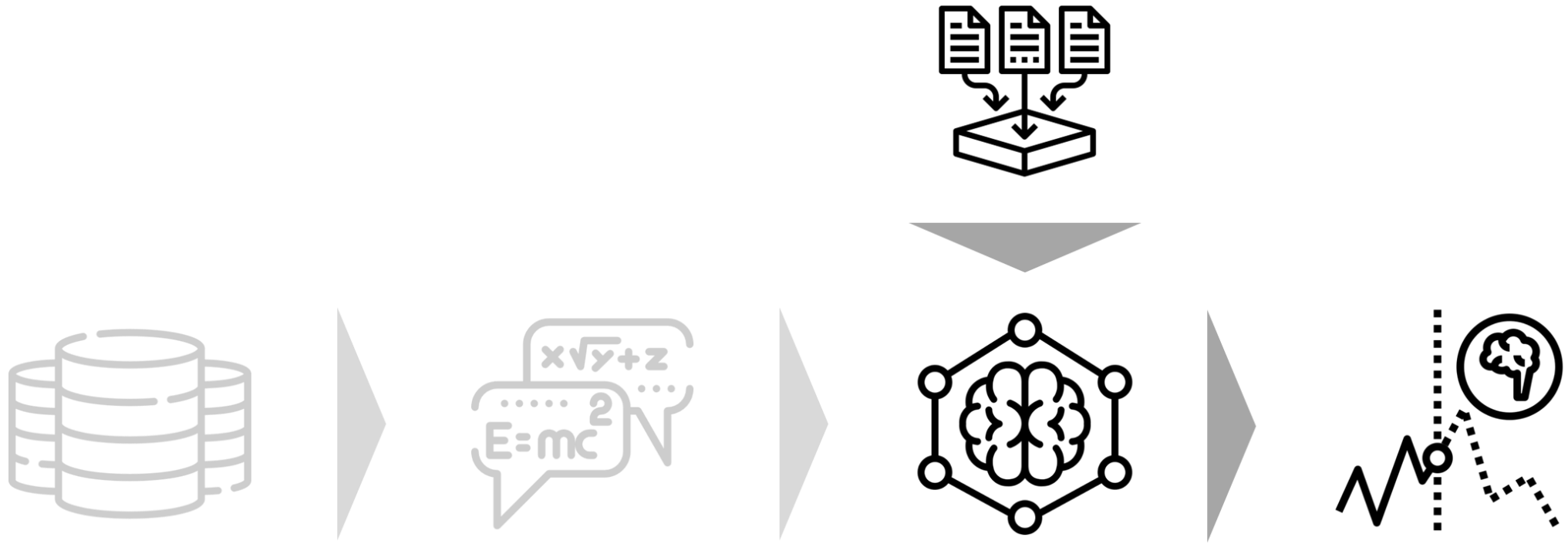
Módulo 11 – Diferença entre aprender e decorar

Aprender é permitir que o nosso algoritmo **busque padrões nos dados** que poderão ser usados para **prever comportamentos futuros.**



Módulo 11 – Diferença entre aprender e decorar

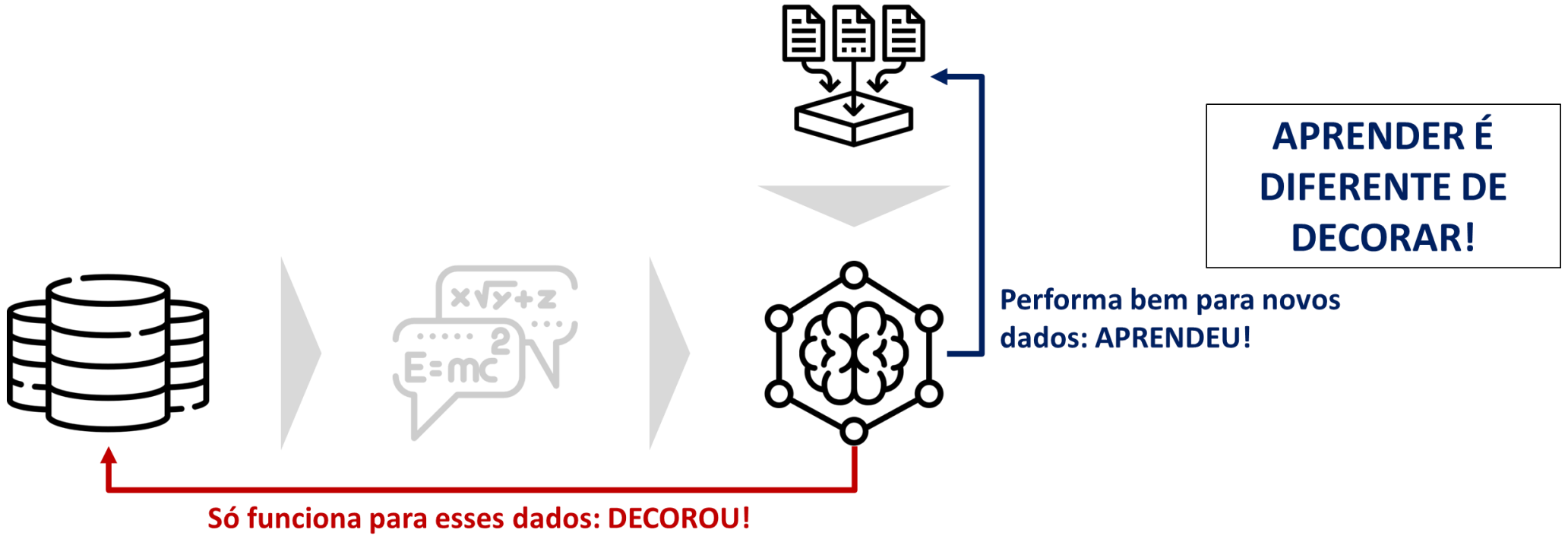
O que esperamos é que nosso modelo faça previsões corretas com novos dados. Ou seja, que os dados de treino tenham sido efetivos em criar um modelo que funcione bem com dados novos, que o modelo não tenha tido contato anteriormente.



Módulo 11 – Diferença entre aprender e decorar

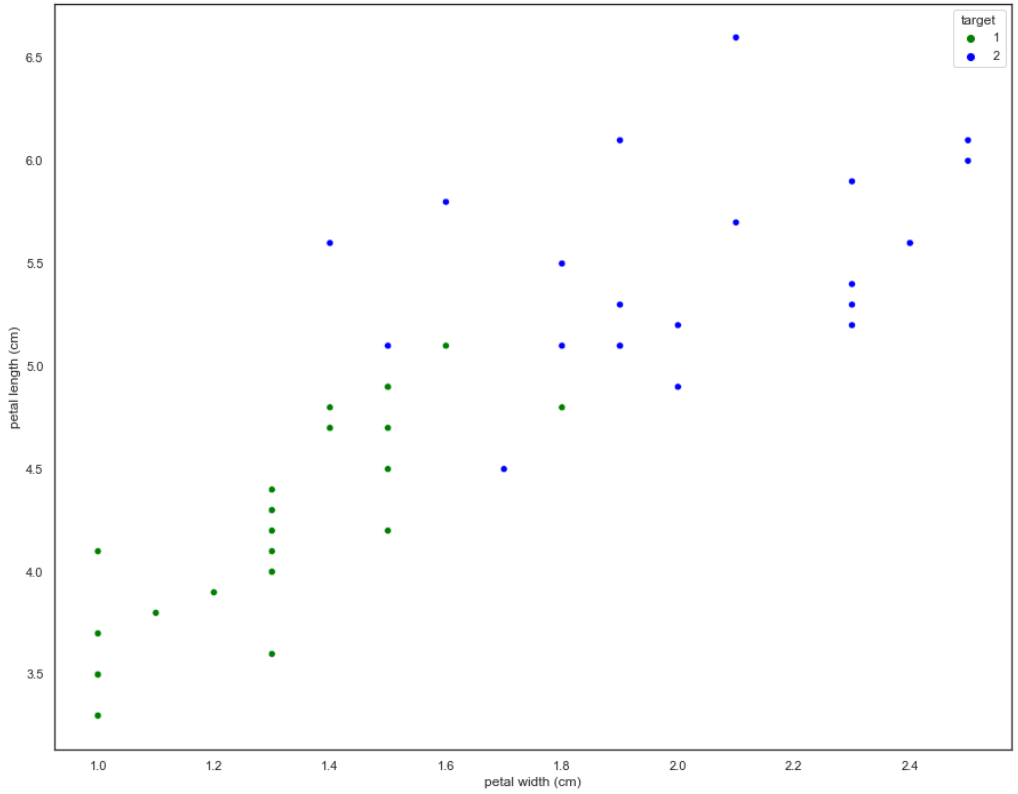
No entanto, pode ocorrer que o modelo performe bem na base de testes, mas não tenha boa performance com novos dados.

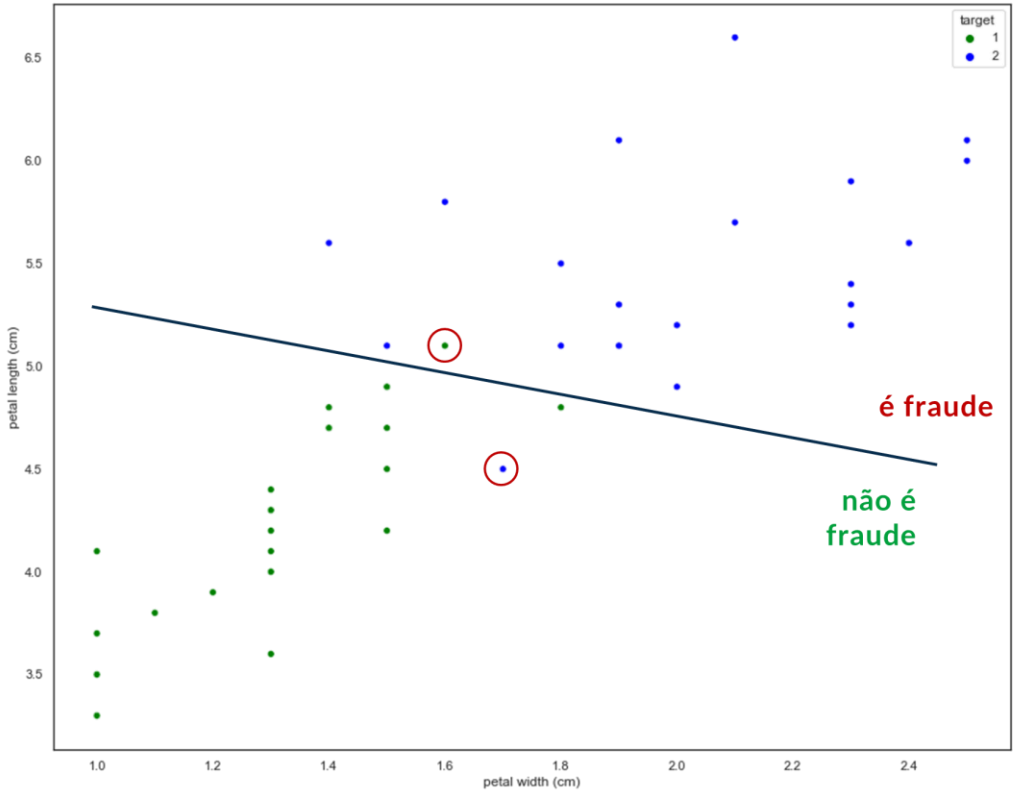
É como se o modelo tivesse “decorado” os dados de teste. Vamos aprender como isso pode ocorrer e como podemos evitar estas situações.



Módulo 11 – Diferença entre aprender e decorar

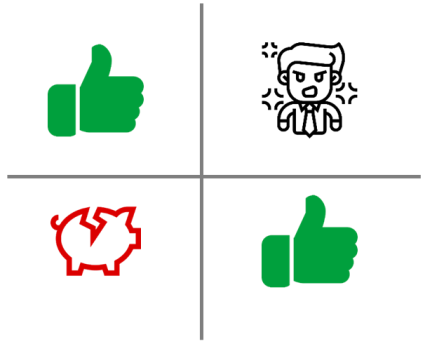
Vamos nos lembrar do exemplo da fraude do módulo anterior. Suponha que um dado modelo tenha criado a seguinte reta de separação das classificações. Repare que há um caso de falso positivo e um de falso negativo.

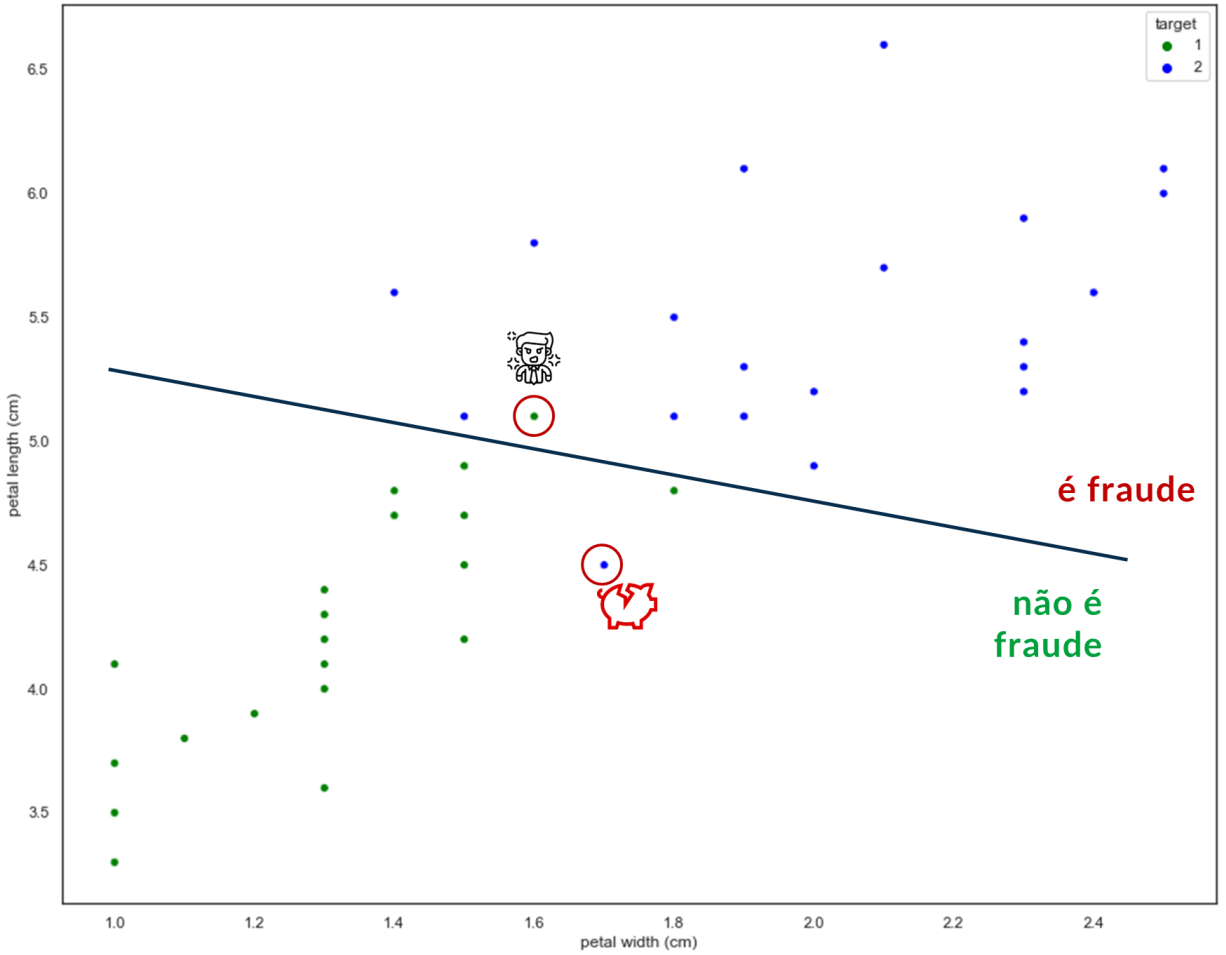




Módulo 11 – Diferença entre aprender e decorar

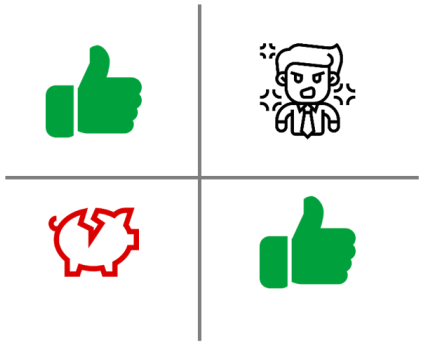
Lembre-se que estas classificações erradas são prejudiciais, pois podem levar a clientes insatisfeitos e a perdas financeiras para a empresa, como já vimos no módulo anterior.

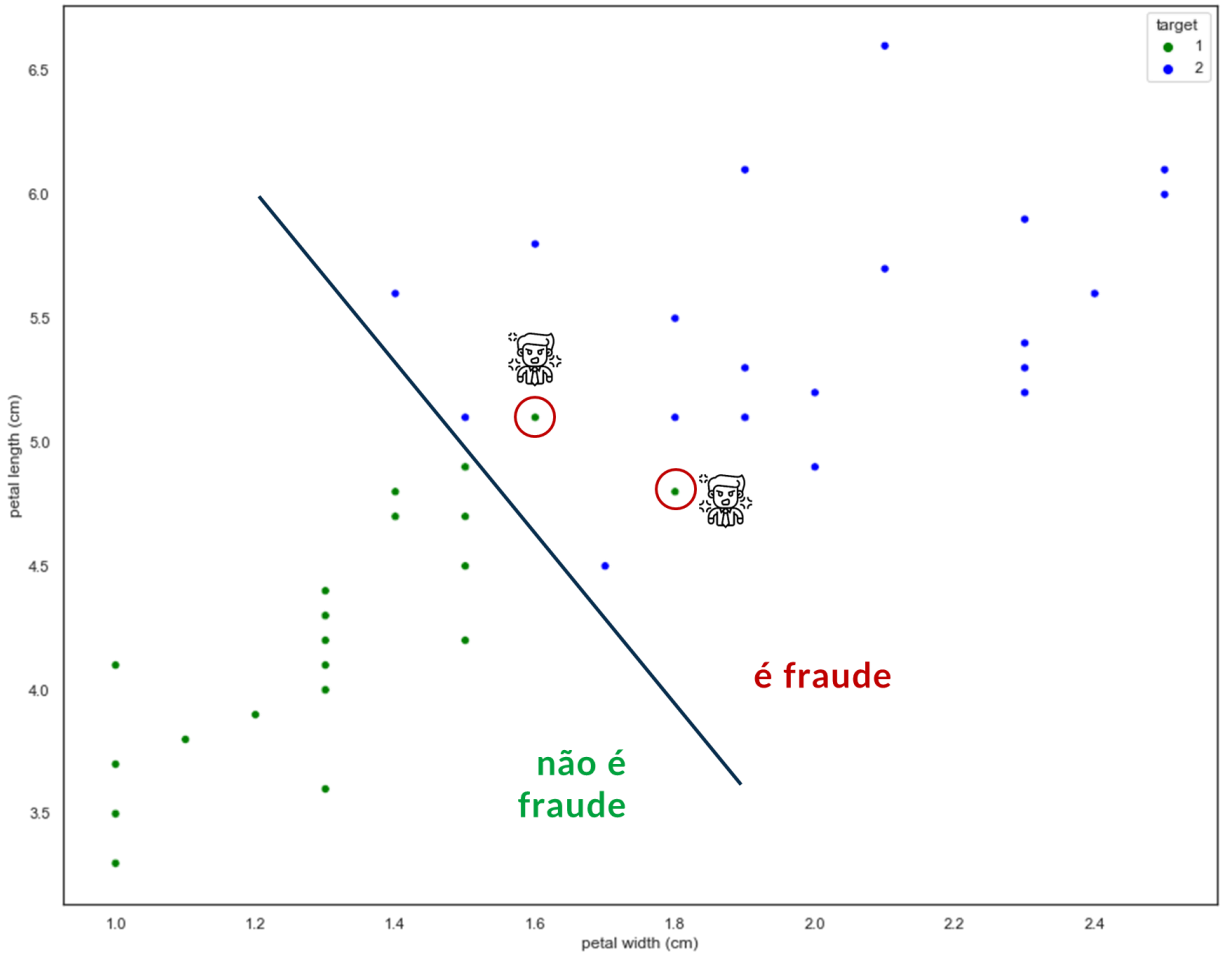




Módulo 11 – Diferença entre aprender e decorar

Uma escolha que pode ser feita é minimizar algum dos erros. Por exemplo, a empresa pode achar mais interessante não ter perdas financeiras a custa de eventualmente ter mais problemas com clientes. Veja que esta escolha vai acabar mudando a reta no gráfico.



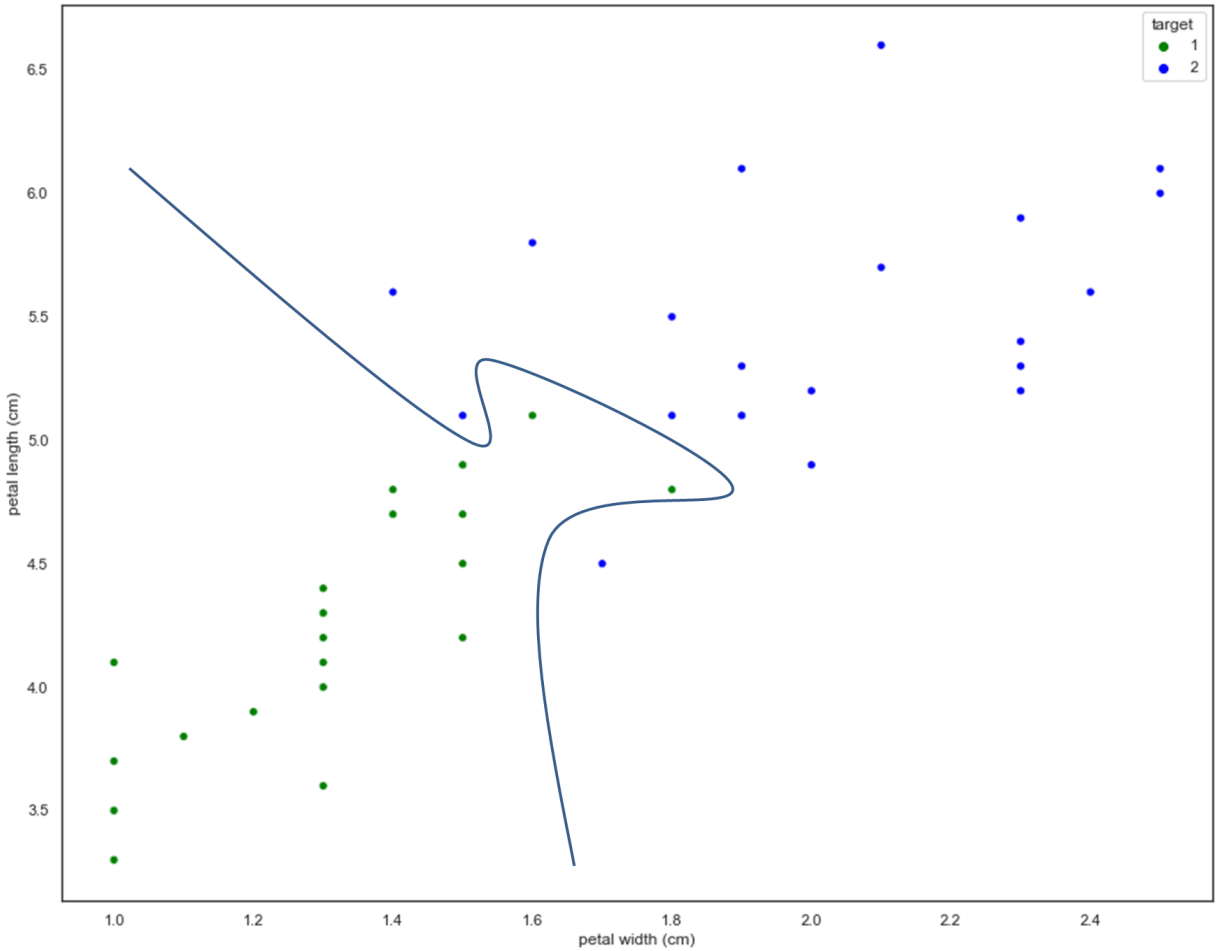


Módulo 11 – Diferença entre aprender e decorar

Agora, suponha que se busque uma situação super ideal, onde há 100 % de acerto do modelo. Toda vez que ver 100 % de acerto, fique atento!

Veja que, teoricamente, nosso modelo é perfeito, criou uma curva que separa perfeitamente os casos de fraude dos que não são fraude. Mas, lembre-se, o modelo foi criado com os dados de treino, os dados prévios.

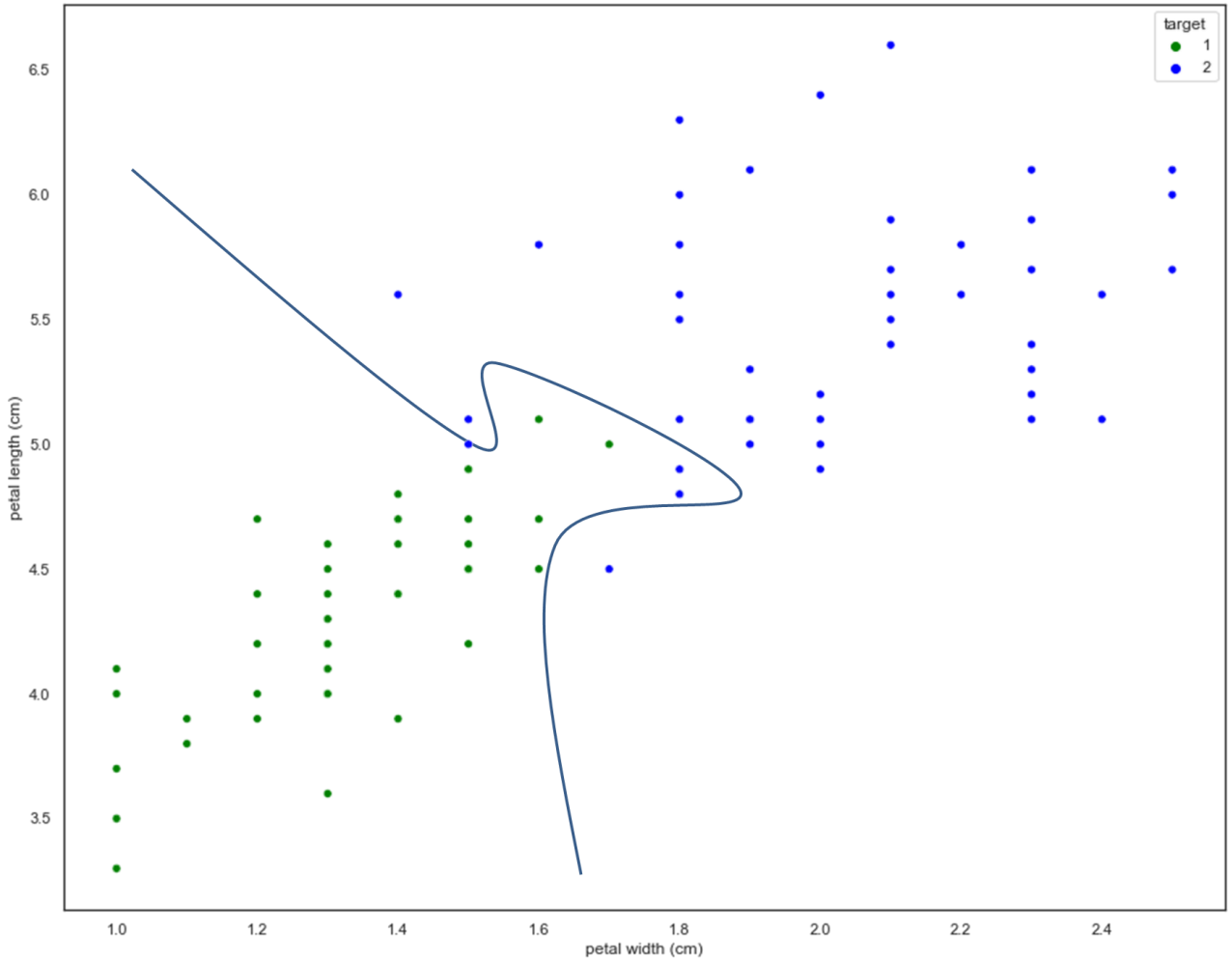
E o que pode ocorrer quando novos casos reais forem apresentados ao modelo?



Módulo 11 – Diferença entre aprender e decorar

Como deixamos nosso modelo super otimizado para os dados de treino, com uma curva bem específica para aqueles dados, ao ser apresentado a novos dados ele acabou errando muito.

Perceba na imagem, pela coloração dos pontos, que temos muito mais erros agora do que com as duras retas que vimos anteriormente.

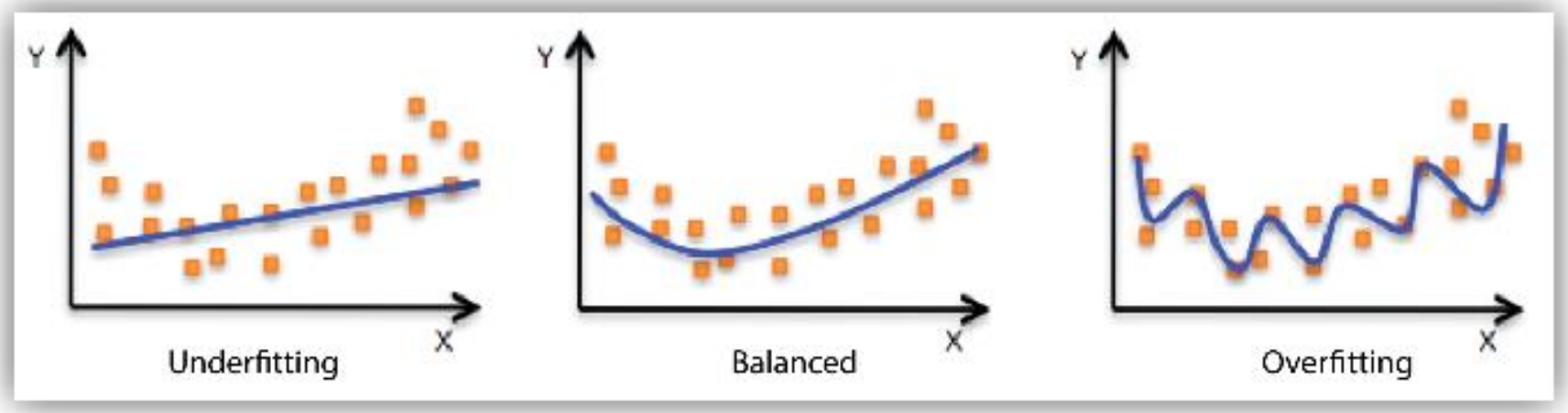


Módulo 11 – Diferença entre aprender e decorar

Quando nosso modelo está muito mal ajustado aos dados, temos um caso de **underfitting.**

Um modelo que tenta passar por todos os pontos, tenta chegar em uma situação perfeita, caracteriza um caso de **overfitting**. O que ocorreu nas últimas páginas.

O ideal é que nosso modelo seja balanceado. Por mais que eventualmente erre, evita erros grosseiros dos outros extremos. E veremos durante o curso como podemos estimar uma margem de erro aceitável.

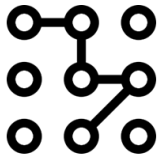


Módulo 11 – Considerações importantes

Machine Learning sempre vai ser a melhor alternativa?

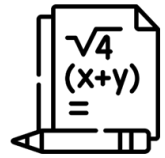
NÃO. Machine Learning será útil se:

Temos **dados disponíveis**



Existe um **padrão** nos dados

O **problema a ser resolvido** está bem definito



Não existe uma fórmula matemática para determinar o resultado

Módulo 11 – Considerações importantes

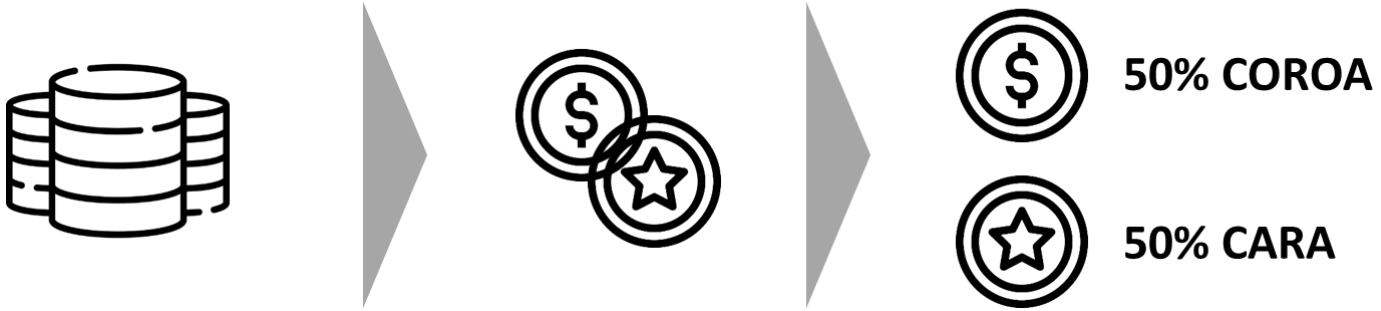
Para entender a importância de dados disponíveis, vamos para um exemplo bem simples: lançamento de moeda.

Sabemos que a probabilidade é de 50 % para cara e para coroa. No entanto, pode ser que em 10 lançamentos, 7 sejam cara e 3 sejam coroa. Se nosso modelo for treinado com apenas esses 10 lançamentos, terá resultados completamente errados.

A tendência é que, com cada vez mais lançamentos, a proporção se aproxime da probabilidade esperada.

A mesma linha de raciocínio se aplica a dados reais. Precisamos de uma quantidade de dados significativa para que padrões sejam percebidos corretamente.

Amostra suficientemente grande!



Módulo 11 – Considerações importantes

Agora, por vezes temos uma limitação no tamanho da base. Por exemplo, se estivermos avaliando o comportamento de vendas de uma empresa, não temos como aumentar a quantidade de vendas do passado. O máximo que podemos fazer é informar ao cliente que a quantidade pode não ser a ideal para uma boa previsão e estipular uma margem de erro ou mudar a forma de estudar o comportamento.

Em geral, não temos controle sobre o tamanho da base

Temos **dados disponíveis**

50% COROA

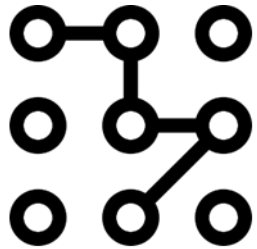
50% CARA

A quantidade que estiver disponível no cliente / empresa

Módulo 11 – Considerações importantes

Já vimos que é necessário haver um padrão para treinar modelos. Mas quando parar de melhorar o modelo?

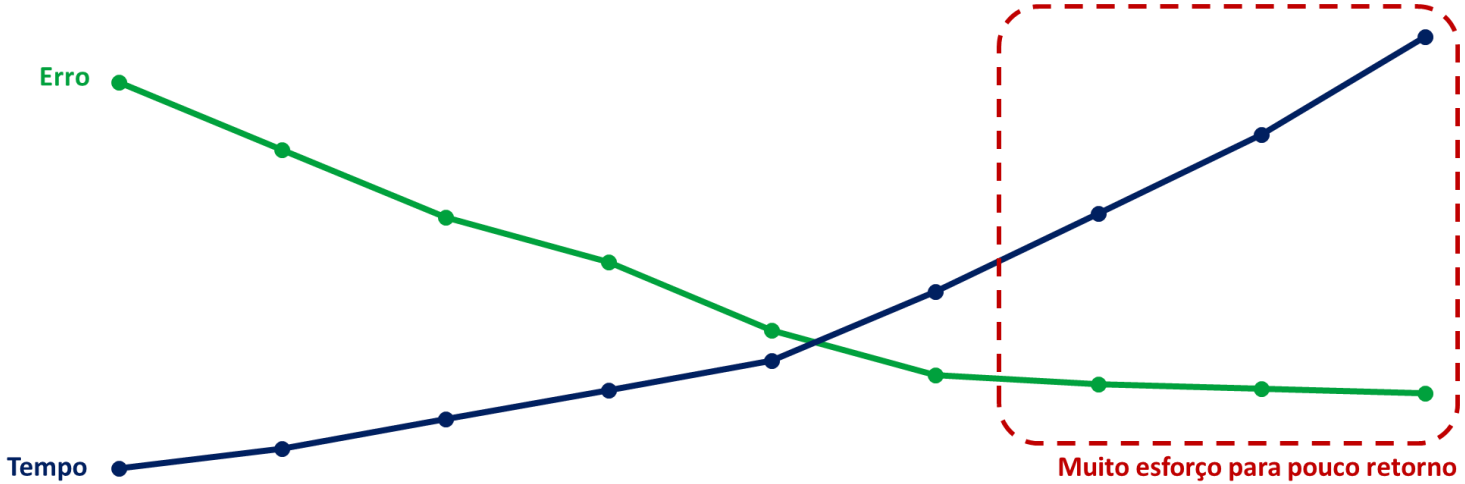
Depende! Você, como cientista, precisa conseguir **ponderar o tempo gasto** para implementar uma melhoria no seu algoritmo com o **resultado** que ela vai trazer!



Existe um **padrão** nos dados

Módulo 11 – Considerações importantes

Perceba, nesse exemplo ilustrativo, que diminuir o erro a partir de um determinado momento, passa a levar muito tempo e pode não ser viável economicamente.



ALGORITMOS	1	1	1	2	2	3	4	5	6
TRATAMENTO DE DADOS	N	S	S	S	S	S	S	S	S
SELEÇÃO DE VARIÁVEIS	N	N	S	S	S	S	S	S	S
AJUSTE DE PARÂMETROS	N	N	N	N	S	S	S	S	S

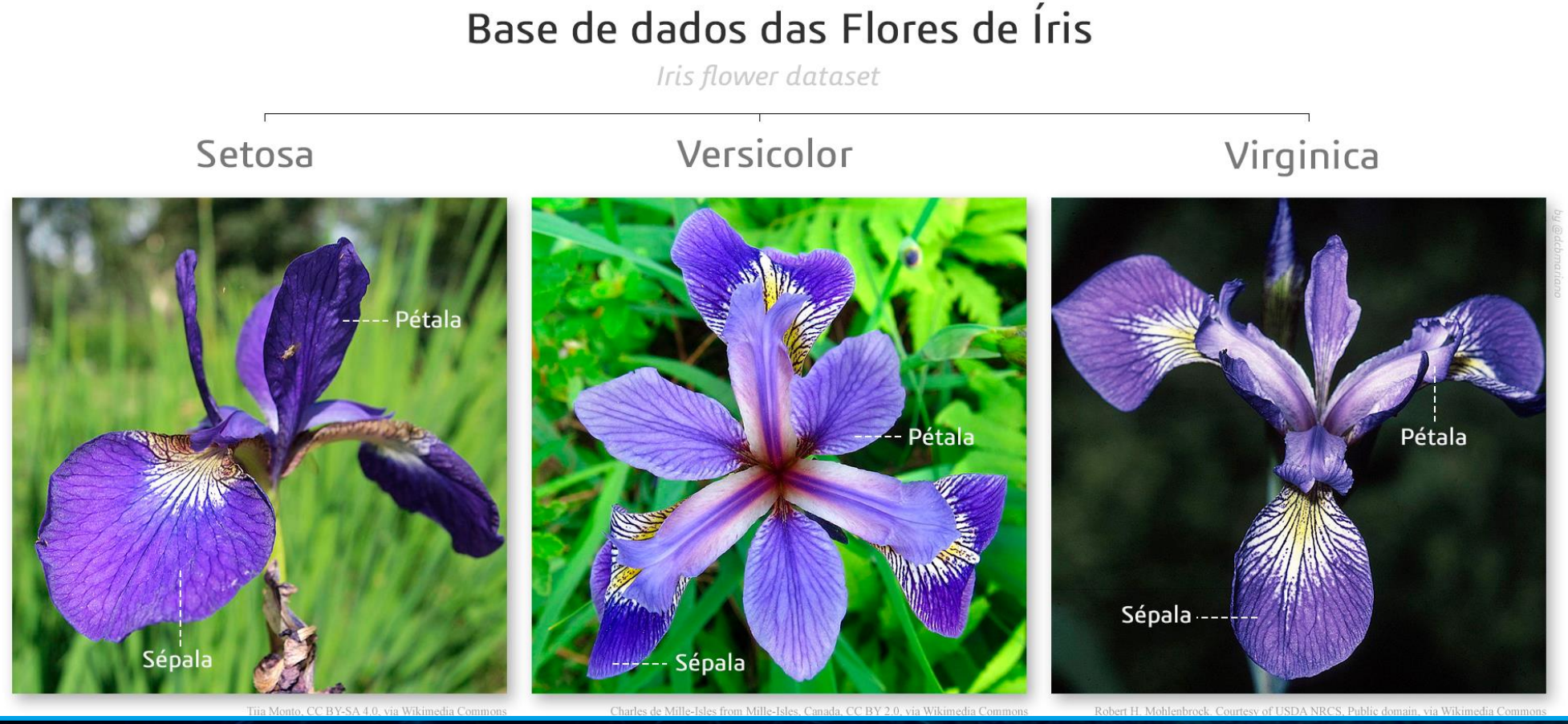
Meramente ilustrativo, apenas para fins didáticos

MÓDULO 12

Criando um modelo de classificação

Módulo 12 – Entendendo o dataset iris

Neste módulo, utilizaremos a famosa base de dados Íris. Esta consiste de 50 amostras de cada espécie da planta Íris, com dados de comprimento e largura das pétalas e sépalas como mostrado na imagem:



Tiia Monto, CC BY-SA 4.0, via Wikimedia Commons

Charles de Mille-Isles from Mille-Isles, Canada, CC BY 2.0, via Wikimedia Commons

Robert H. Mohlenbrock. Courtesy of USDA NRCS. Public domain, via Wikimedia Commons

Módulo 12 – Entendendo o dataset iris

A base de dados pode ser importada diretamente do Scikit-Learn, que possui um conjunto de bases via `sklearn.datasets`.

Vamos aproveitar e importar as demais bibliotecas que usaremos: Pandas, NumPy e Matplotlib.

E carregar a base de dados na variável `data`.

```
# Importando as bibliotecas (pandas, matplotlib, numpy)
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
# Importando o dataset iris
from sklearn.datasets import load_iris
```

```
# Obtendo os dados do dataset
data = load_iris()
```

Módulo 12 – Entendendo o dataset iris

Por padrão, o Scikit-Learn armazena as bases de dados como arrays do NumPy. No caso em questão, um array para os atributos, que estamos chamando de X desde os últimos módulos; e um array para nossa variável alvo, o y dos últimos módulos. Podemos acessar com os atributos `data` e `target` de nossa variável:

```
# Valores de X
data.data
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
```

```
# Valores de Y
data.target
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

Seria mais conveniente utilizar objetos do Pandas. Assim, vamos utilizar o construtor `DataFrame` e instruir que o as colunas tenham os nomes corretos, armazenados na variável `data` e que podem ser consultados com o atributo `feature_names`:

```
# Transformando em um DataFrame
iris = pd.DataFrame(data.data)
iris.head()
```

	0	1	2	3
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
# Nome das colunas
iris.columns = data.feature_names
iris.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

A variável alvo também pode ser colocada no dataframe, na última coluna:

```
# Adicionando a coluna targer ao DataFrame
iris['Target'] = data.target
iris.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Como discutido no início do módulo, a base de dados apresenta 50 amostras de cada uma das espécies da planta. Podemos verificar isto com o método `value_counts()` na coluna da variável alvo `Target`.

Para facilitar a compreensão, vamos nos restringir a apenas duas das espécies, retirando a com classificação “2”.

```
# Contando a quantidade de cada um dos Targets
iris['Target'].value_counts()
```

```
0    50
1    50
2    50
Name: Target, dtype: int64
```

```
# Retirando Target = 2
iris = iris[iris.Target != 2]
```

```
# Contando novamente os valores
iris['Target'].value_counts()
```

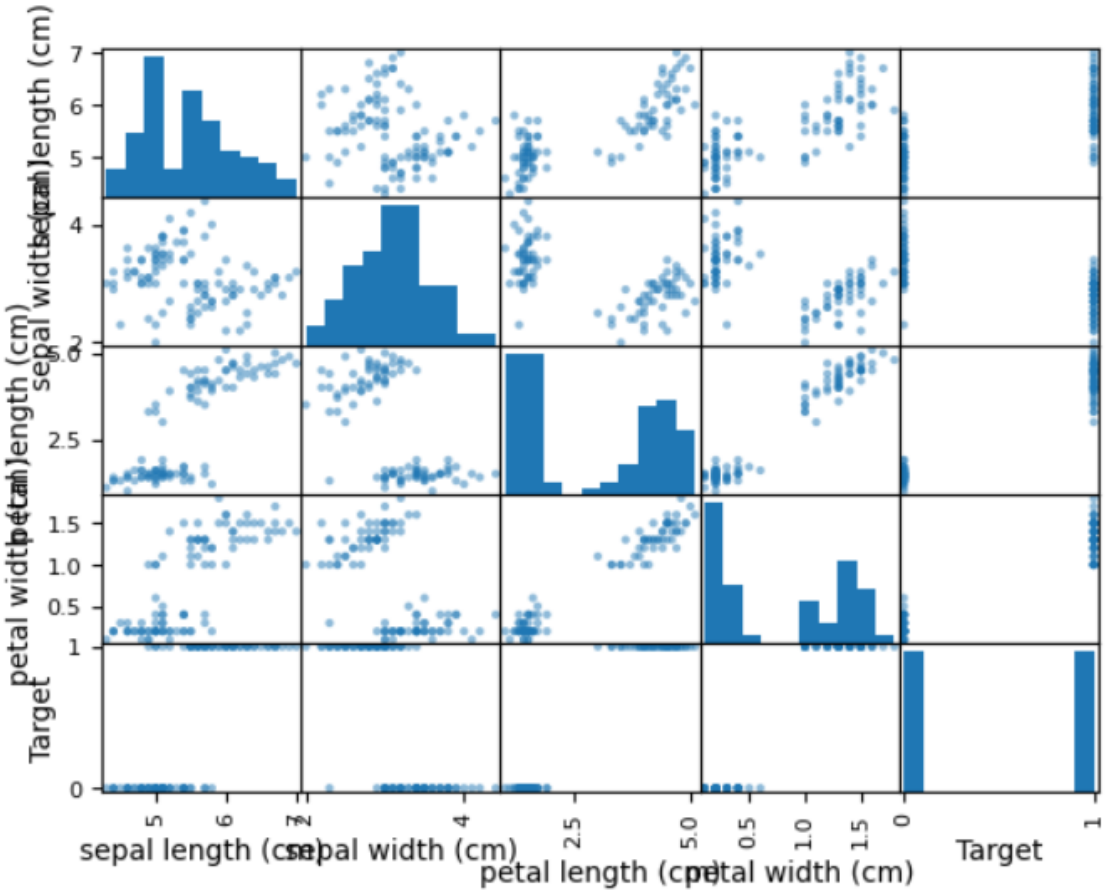
```
0    50
1    50
Name: Target, dtype: int64
```

Módulo 12 – Visualizando os dados

Agora que já arrumamos os dados, vamos buscar entender como as variáveis se relacionam. Uma forma é através de matrizes de distribuição, como no Pandas `scatter_matrix`, que mostra a relação entre as variáveis numéricas.

Na diagonal da matriz, vemos a distribuição de valores da cada variável na forma de histograma. Perceba, por exemplo, que a última posição da matriz mostra que há apenas dois valores na coluna `Target`, com iguais quantidades.

```
pd.plotting.scatter_matrix(iris);
```

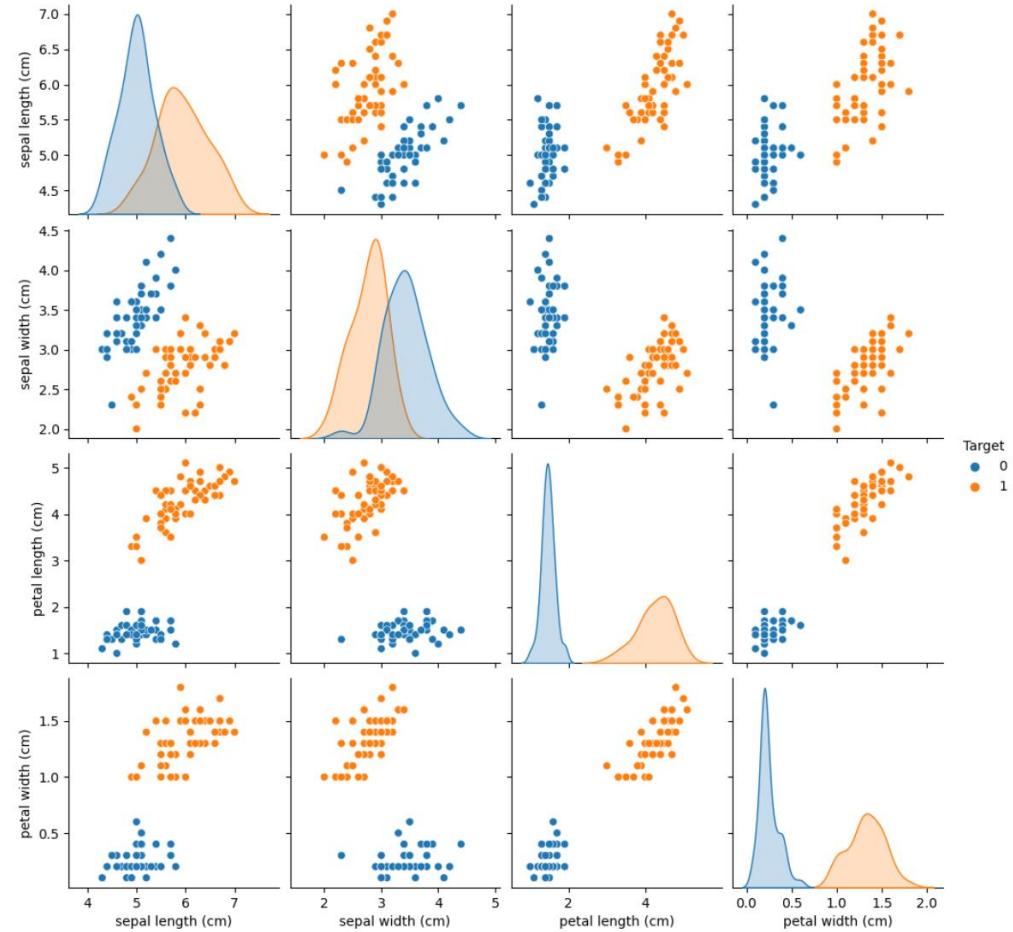


Módulo 12 – Visualizando os dados

O `scatter_matrix` do Pandas é muito útil, mas há uma ferramenta com visualizações ainda melhores e personalizáveis, o **`pairplot`** do Searbon.

Em um problema de classificação, é útil verificar as distribuições comparando cada classe. Veja como se passou a coluna **`Target`** para o parâmetro **`hue`**. Com isto, conseguimos verificar como as variáveis se relacionam já destacando cada classificação. Perceba como os pontos de cada classe são bem distinguíveis.

```
import seaborn as sns
sns.pairplot(iris,hue="Target");
```

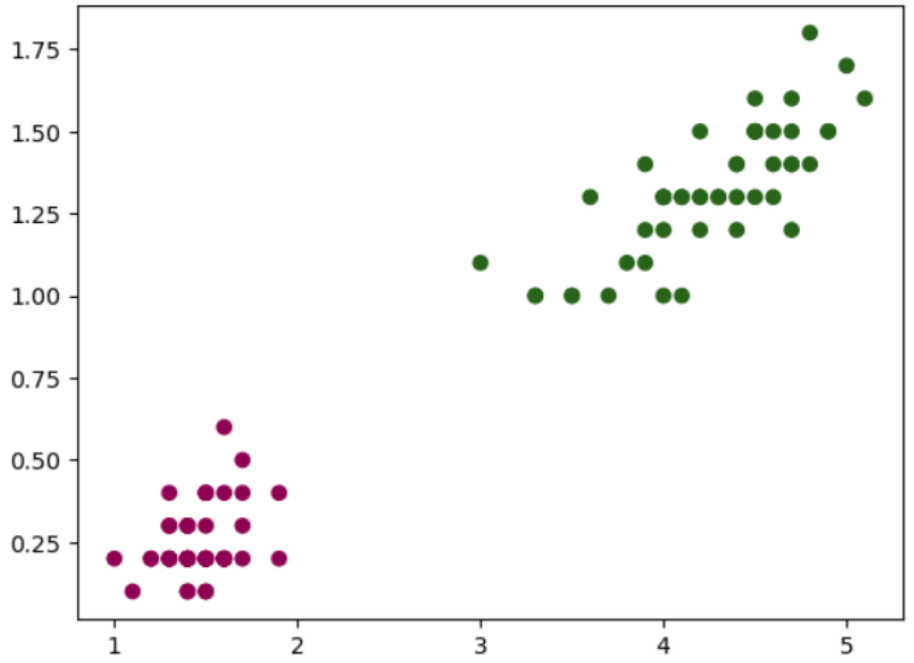


Módulo 12 – Visualizando os dados

Vamos pegar um par de atributos, no caso comprimento e largura das pétalas, para olhar mais atentamente. Podemos utilizar o `scatter` do Matplotlib, passando a coluna `Target` para usar cores distintas para cada classificação. O `PiYG` é uma escala de cores, que estamos utilizando apenas para que tenham conhecimento que há escalas de cores pré-definidas no Matplotlib e que podem ser consultadas em:

https://matplotlib.org/stable/tutorials/colors/colormaps.html

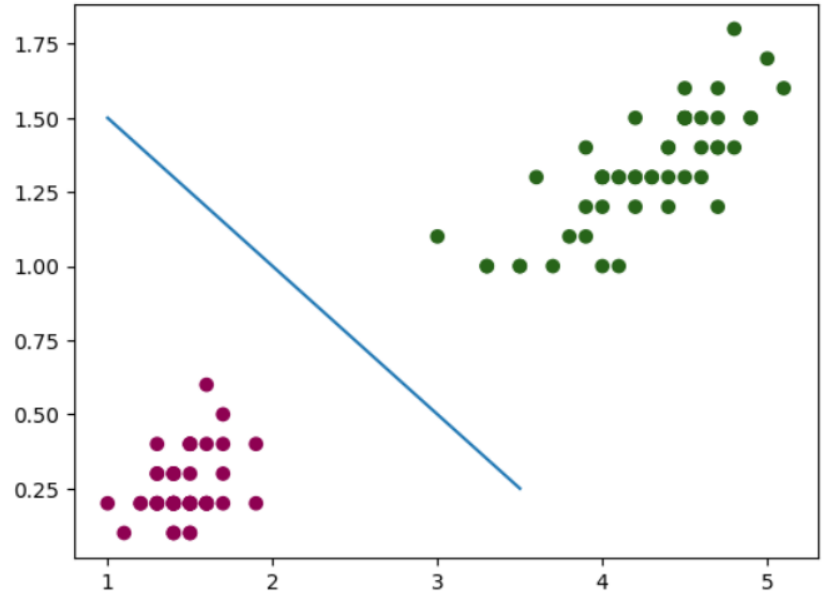
```
# Traçando um scatter do matplotlib das colunas "petal length (cm)" e "petal width (cm)"
fig, ax = plt.subplots()

x = iris["petal length (cm)"]
y = iris["petal width (cm)"]

ax.scatter(x,y,c=iris.Target,cmap='PiYG')

plt.show()
```



Módulo 12 – Criando uma reta capaz de separar os dados do modelo

Assim como já vimos em módulos anteriores, podemos traçar uma reta que separa as duas classes. No código ao lado, criamos uma reta manualmente, com base em critérios visuais:

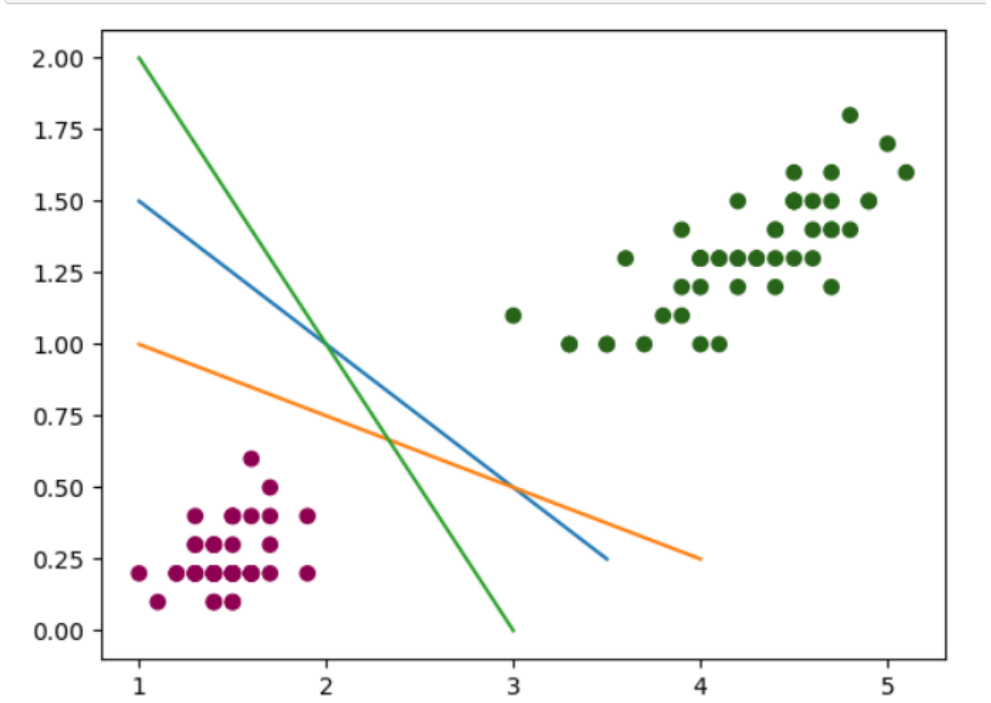
```
# Traçando o scatter plot e a reta
fig, ax = plt.subplots()

# Aqui estamos fazendo um gráfico de dispersão
x = iris["petal length (cm)"]
y = iris["petal width (cm)"]
ax.scatter(x,y,c=iris.Target,cmap='PiYG')

# Aqui estamos fazendo um gráfico de linha
x_reta = [1,3.5]
y_reta = [1.5,0.25]
ax.plot(x_reta,y_reta)

plt.show()
```



Módulo 12 – Criando uma reta capaz de separar os dados do modelo

No entanto, várias retas poderiam ser utilizadas. Como vemos, qualquer uma das retas a seguir separa as duas classes:

```
# Buscando por melhores retas
fig, ax = plt.subplots()

# Aqui estamos fazendo um gráfico de dispersão
x = iris["petal length (cm)"]
y = iris["petal width (cm)"]
ax.scatter(x,y,c=iris.Target,cmap='PiYG')

# Aqui estamos fazendo um gráfico de linha
x_reta = [1,3.5]
y_reta = [1.5,0.25]
ax.plot(x_reta,y_reta)

# Aqui estamos fazendo um gráfico de linha
x_reta2 = [1,4]
y_reta2 = [1,0.25]
ax.plot(x_reta2,y_reta2)

# Aqui estamos fazendo um gráfico de linha
x_reta3 = [1,3]
y_reta3 = [2,0]
ax.plot(x_reta3,y_reta3)

plt.show()
```



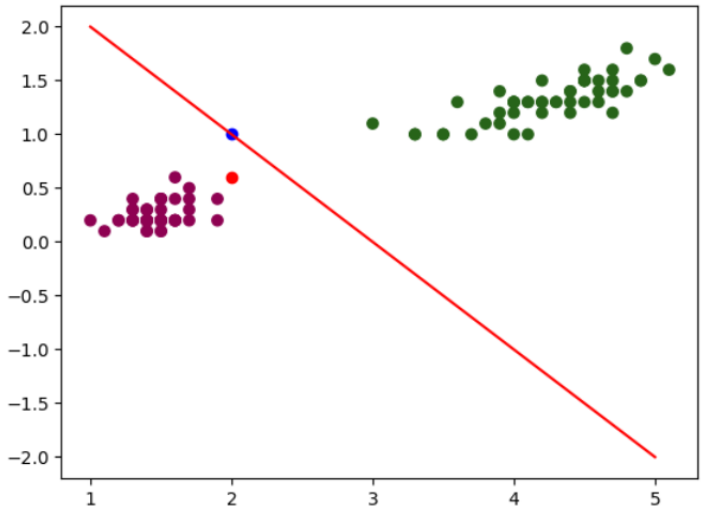
Módulo 12 – Criando uma reta capaz de separar os dados do modelo

Vamos pegar uma dessas retas e alongá-la um pouco mais. E criar dois pontos, com cores vermelha e azul.

Perceba que o ponto vermelho está abaixo de nossa reta, mais próximo a um dos grupos de pontos, de cor roxa.

O ponto azul está levemente acima da reta. Como o objetivo da reta é separar as classes, esse ponto será classificado como pertencente ao grupo dos pontos verdes, mesmo estando relativamente distante destes. Daí a importância de se ter uma boa reta.

```
# Traçando novamente a reta e adicionando o ponto que acima
fig, ax = plt.subplots()

# Aqui estamos fazendo um gráfico de dispersão
x = iris["petal length (cm)"]
y = iris["petal width (cm)"]
ax.scatter(x,y,c=iris.Target,cmap='PiYG')

# Aqui estamos fazendo um gráfico de linha
x_reta3 = np.arange(1,6)
y_reta3 = (-1)*x_reta3+3
ax.plot(x_reta3,y_reta3,c='r')

# Fazendo o gráfico de dispersão somente desse ponto
ax.scatter(2,+0.6,c='r')

# Fazendo o gráfico de dispersão somente desse ponto
ax.scatter(2,+1,c='b')

plt.show()
```



Módulo 12 – Criando uma função classificadora

O “acima” e “abaixo” da página anterior se refere ao valor da ordenada do ponto, o y, que pode ser maior ou menor que o y da reta para um determinado valor de x.

Assim, podemos criar uma função que classifica um par (x, y) verificando se tal ponto está acima ou abaixo de uma dada reta. A reta da página anterior tem equação y = -x + 3, daí o seu uso no corpo da função:

```
def classifica_modelo(x,y):
    y_equacao = -x + 3
    y_ponto = y

    if y_ponto > y_equacao:
        return 1
    elif y_ponto < y_equacao:
        return 0
    else:
        return -1
```

```
# (3.5,0.7)
classifica_modelo(3.5,0.7)
```

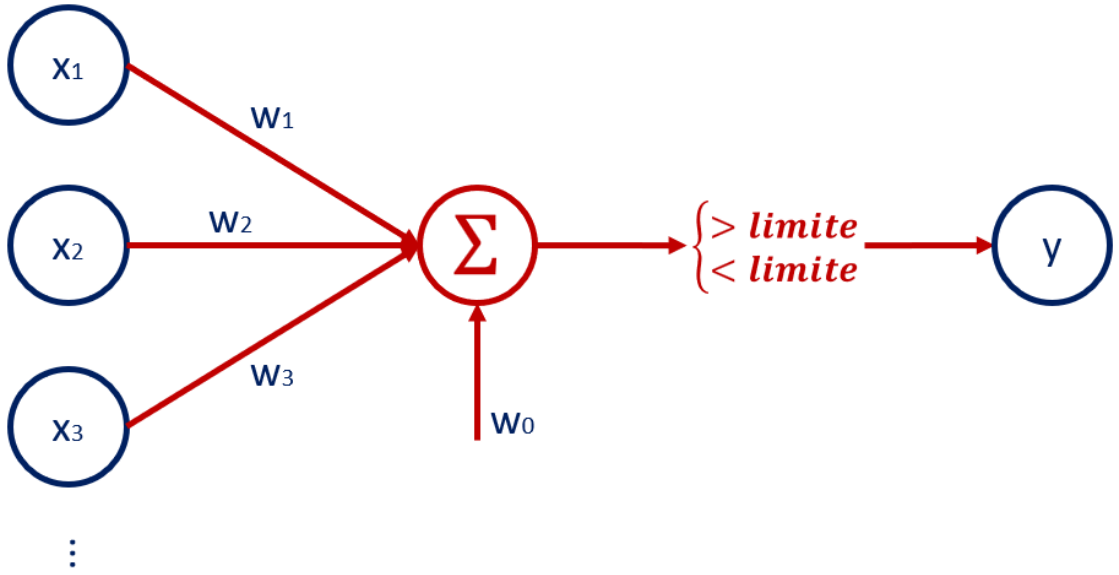
```
1
```

```
# (2,0.6)
classifica_modelo(2,0.6)
```

```
0
```

Módulo 12 – Perceptron

Tudo que fizemos manualmente pode ser automatizado com modelos do Scikit-Learn. E não apenas para duas variáveis, podendo ter diversas variáveis como mostrado na figura a seguir. A lógica é a mesma vista, mas a vantagem é delegar todo o trabalho que tivemos manualmente para um modelo. No caso, o modelo Perceptron.




```
# Importando o perceptron
from sklearn.linear_model import Perceptron
clf = Perceptron(tol=1e-3, random_state=0)
```

Módulo 12 – Perceptron

Agora, precisamos de nossos valores de X e de y. Passamos para o `fit` do modelo e podemos obter os coeficientes e o intercepto.

Ou seja, tudo o feito anteriormente, mas de forma bem mais rápida e reprodutível.

```
# Selecionando o X e o y
X = iris[['petal length (cm)','petal width (cm)']]
y = iris.Target
```

```
# Fazendo o fit com o modelo
clf.fit(X, y)
```

```
Perceptron()
```

```
# w1 e w2
clf.coef_
```

```
array([[0.9, 1.7]])
```

```
# w0
clf.intercept_
```

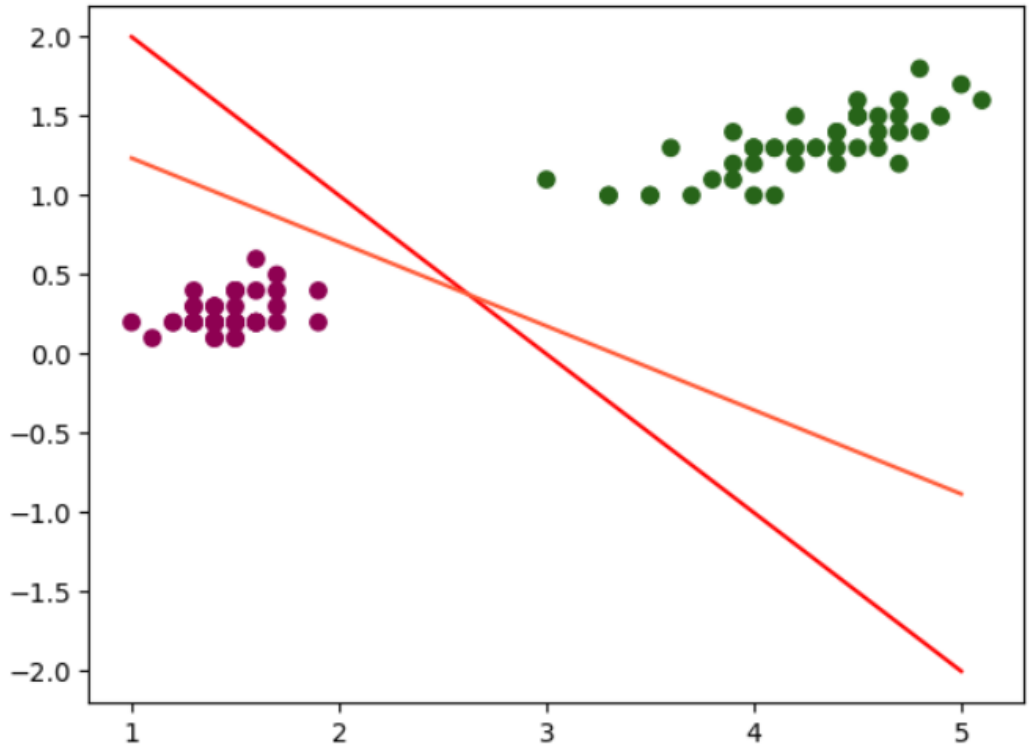
```
array([-3.])
```

```
clf.coef_[0][1]
```

```
1.7
```

Com os coeficientes e o intercepto, podemos traçar graficamente a reta do modelo. Abaixo, a reta vermelha é que fizemos manualmente e a laranja a reta do modelo. Veja que há uma diferença significativa entre as duas.

```
# Traçando novamente a reta e adicionando o ponto que acima
fig, ax = plt.subplots()

# Aqui estamos fazendo um gráfico de dispersão
x = iris["petal length (cm)"]
y = iris["petal width (cm)"]
ax.scatter(x,y,c=iris.Target,cmap='PiYG')

# Aqui estamos fazendo um gráfico de linha
x_reta3 = np.arange(1,6)
y_reta3 = (-1)*x_reta3+3
ax.plot(x_reta3,y_reta3,c='r')

# Aqui estamos fazendo um gráfico de linha
x_perc = np.arange(1,6)
y_perc = (-clf.intercept_-clf.coef_[0][0]*x_perc)/clf.coef_[0][1]
ax.plot(x_perc,y_perc,c='#FF5733')

plt.show()
```



Módulo 12 – Perceptron

Para entender a importância de ter uma boa reta de separação, vamos colocar manualmente dois novos pontos em nossa base de dados.

```
# Adicionando uma nova base com os dados mostrados abaixo
base2 = pd.DataFrame([[0,0,3,0.75,0], [0,0,2.5,1,0]])
base2.columns = ['sepal length (cm)','sepal width (cm)','petal length (cm)','petal width (cm)','Target']
```

```
base2.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Target
0	0	0	3.0	0.75	0
1	0	0	2.5	1.00	0

```
# Adicionado esses dados ao dataset iris
iris = iris.append(base2)
```

```
iris.tail()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Target
97	6.2	2.9	4.3	1.30	1
98	5.1	2.5	3.0	1.10	1
99	5.7	2.8	4.1	1.30	1
0	0.0	0.0	3.0	0.75	0
1	0.0	0.0	2.5	1.00	0

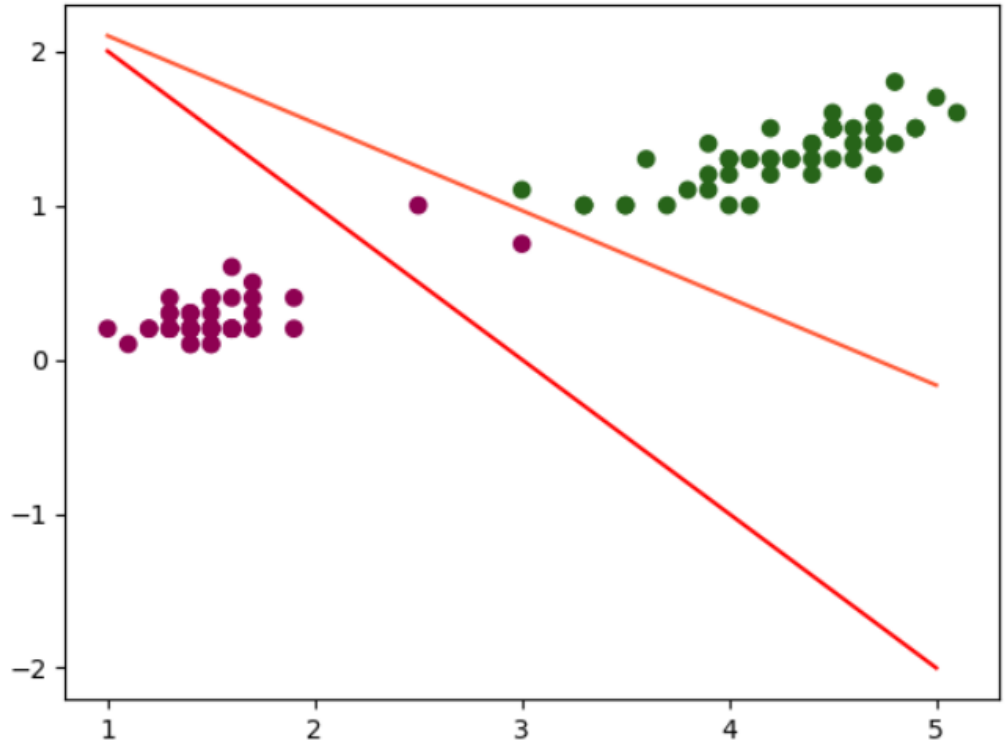
Agora, vamos treinar nosso modelo com essa nova base de dados, com os dois pontos adicionados:

```
# Executando o Perceptron pra essa nova base
X = iris[["petal length (cm)","petal width (cm)"]]
y = iris.Target

clf = Perceptron(tol=1e-3, random_state=0)
clf.fit(X, y)
```

```
Perceptron()
```

Veja que a reta do modelo (a laranja) é diferente da reta do modelo anterior, quando não havia os dois pontos. O modelo efetivamente foi alterado pela presença destes pontos na base de dados. Enquanto isso, nossa reta criada manualmente (a vermelha) permanece inalterada e classifica os novos pontos erroneamente. Percebemos, portanto, como o machine learning é importante por efetivamente aprender com novos dados.

```
# Traçando novamente a reta e adicionando o ponto que acima
fig, ax = plt.subplots()

# Aqui estamos fazendo um gráfico de dispersão
x = iris["petal length (cm)"]
y = iris["petal width (cm)"]
ax.scatter(x,y,c=iris.Target,cmap='PiYG')

# Aqui estamos fazendo um gráfico de linha
x_reta3 = np.arange(1,6)
y_reta3 = (-1)*x_reta3+3
ax.plot(x_reta3,y_reta3,c='r')

# Aqui estamos fazendo um gráfico de linha
x_perc = np.arange(1,6)
y_perc = (-clf.intercept_-clf.coef_[0][0]*x_perc)/clf.coef_[0][1]
ax.plot(x_perc,y_perc,c='#FF5733')

plt.show()
```



MÓDULO 13

Utilizando o aprendizado de máquinas

Módulo 13 – Matriz de confusão

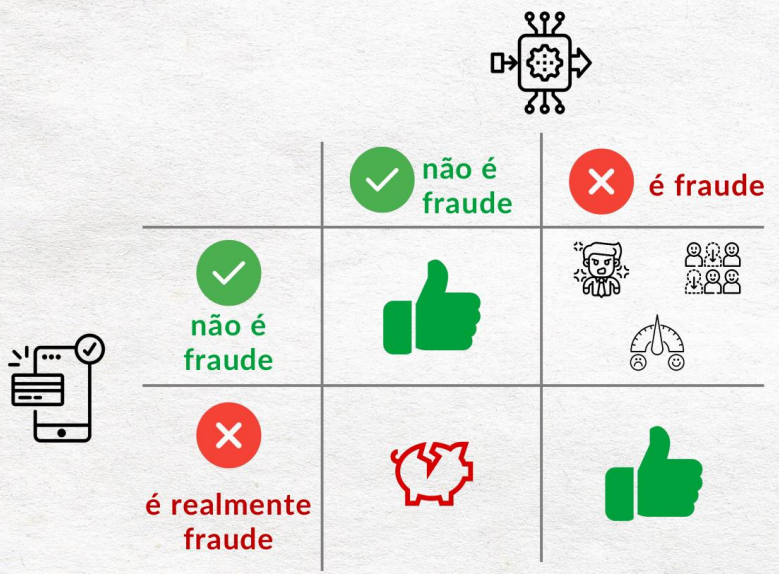
Em módulos anteriores, vimos que podemos classificar transações financeiras como fraudulentas ou não.

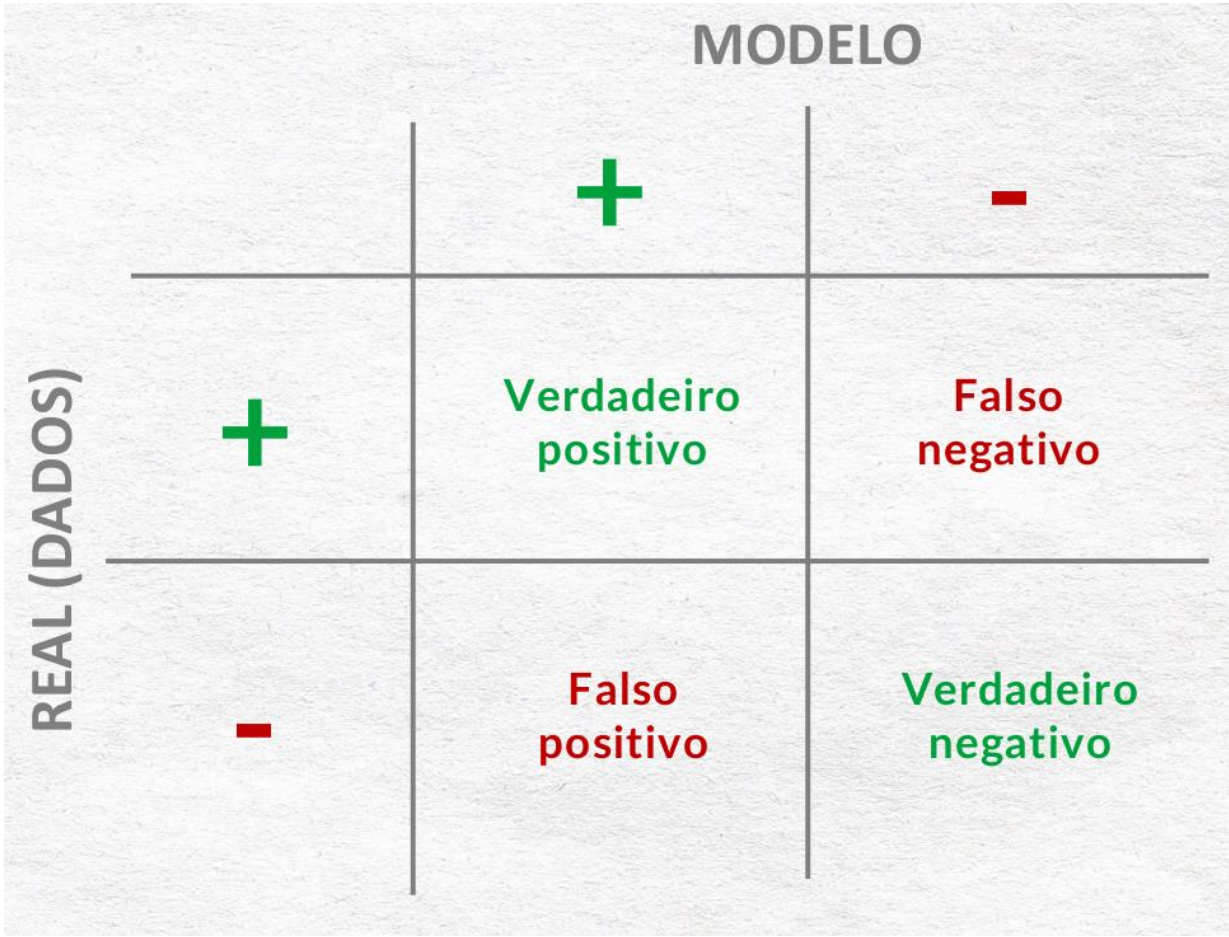
Vamos retomar esta aplicação para conhecer melhor alguns termos utilizados em aprendizado de máquinas.

Começaremos entendendo o que são resultados positivos e negativos em uma matriz de confusão.

Módulo 13 – Matriz de confusão

Na vida real, é muito difícil que um modelo acerte todas as previsões. Assim, teremos os chamados falsos positivos (transações não fraudulentas erroneamente classificadas como fraude) e os falsos negativos (transações fraudulentas erroneamente classificadas como legítimas):





Módulo 13 – Matriz de confusão

Esse arranjo das figuras apresentadas é conhecido como matriz de confusão. Vamos construir um pequeno código Python importando o método `confusion_matrix` do Scikit-Learn. Passaremos 6 dados reais (`y_true`) e 6 resultados de um modelo de previsão (`y_pred`).

```
from sklearn.metrics import confusion_matrix
y_true = ['-','-','-','+','+','+']
y_pred = ['-','-','+','+','-','-']
confusion_matrix(y_true, y_pred,labels=['+','-'])
```

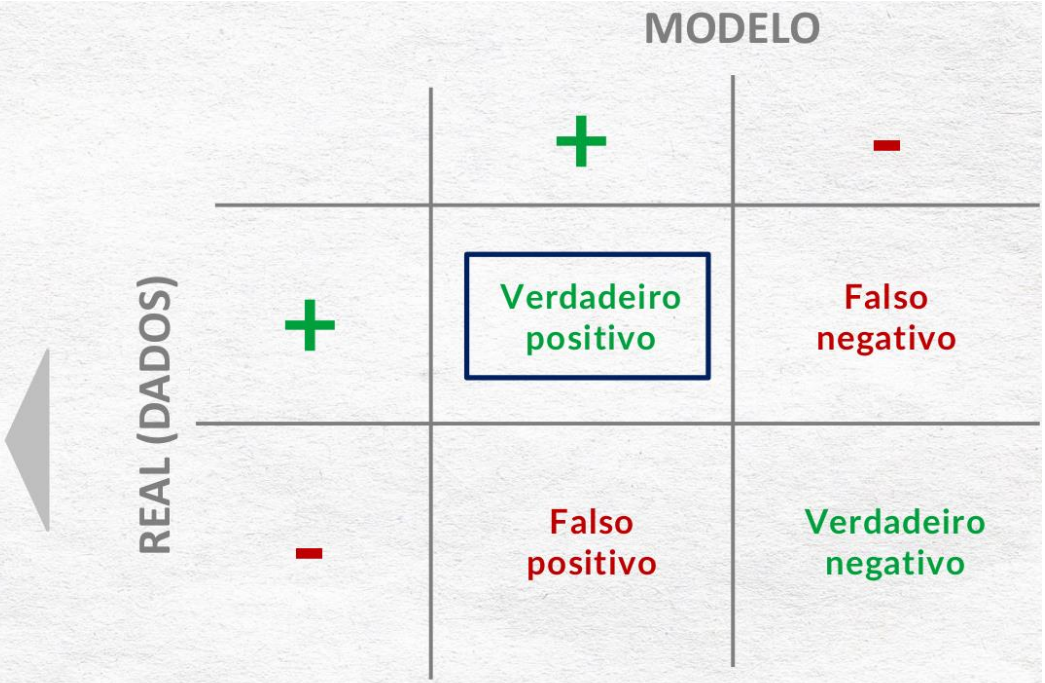
```
array([[1, 2],
       [1, 2]], dtype=int64)
```

Módulo 13 – Matriz de confusão

Observe, comparando visualmente as duas listas, que há apenas uma situação com sinal positivo nos dados que também apresenta sinal positivo na previsão. Logo, há um verdadeiro positivo.

```
from sklearn.metrics import confusion_matrix
y_true = ['-','-','-','+','+','+']
y_pred = ['-','-','+','+','-','-']
confusion_matrix(y_true, y_pred,labels=['+','-'])
```

```
array([[1, 2],
       [1, 2]], dtype=int64)
```

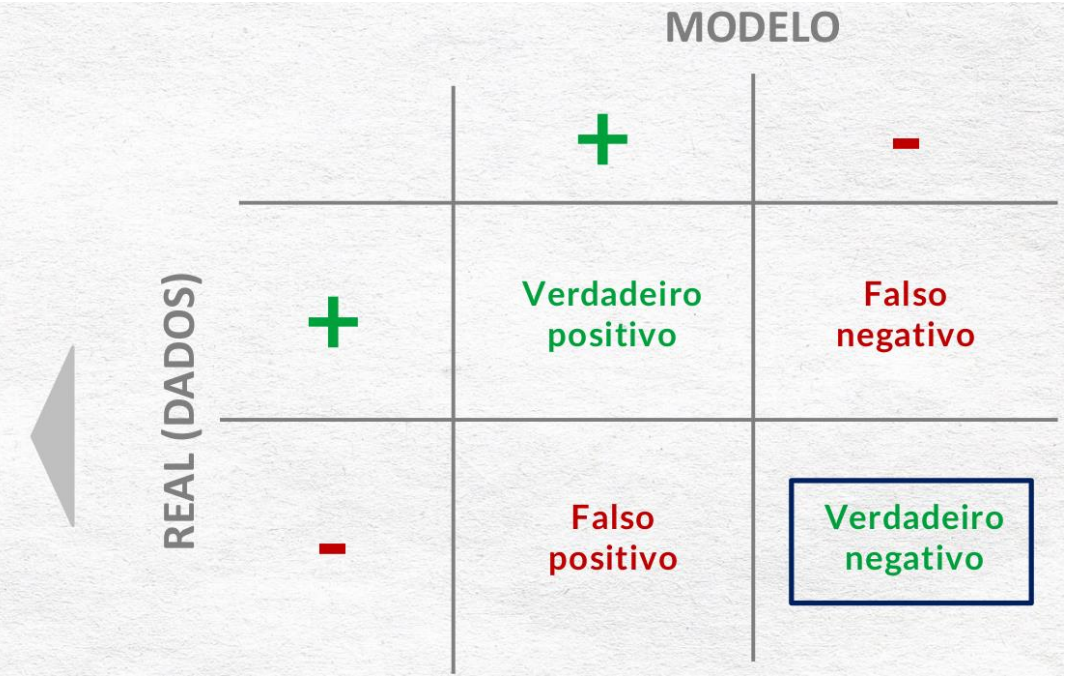


Módulo 13 – Matriz de confusão

Seguindo, vemos que há duas situações coincidentes de sinais negativos. Logo, há 2 verdadeiros negativos.

```
from sklearn.metrics import confusion_matrix
y_true = ['-','-','-','+','+','+']
y_pred = ['-','-','+','+','-','-']
confusion_matrix(y_true, y_pred,labels=['+','-'])
```

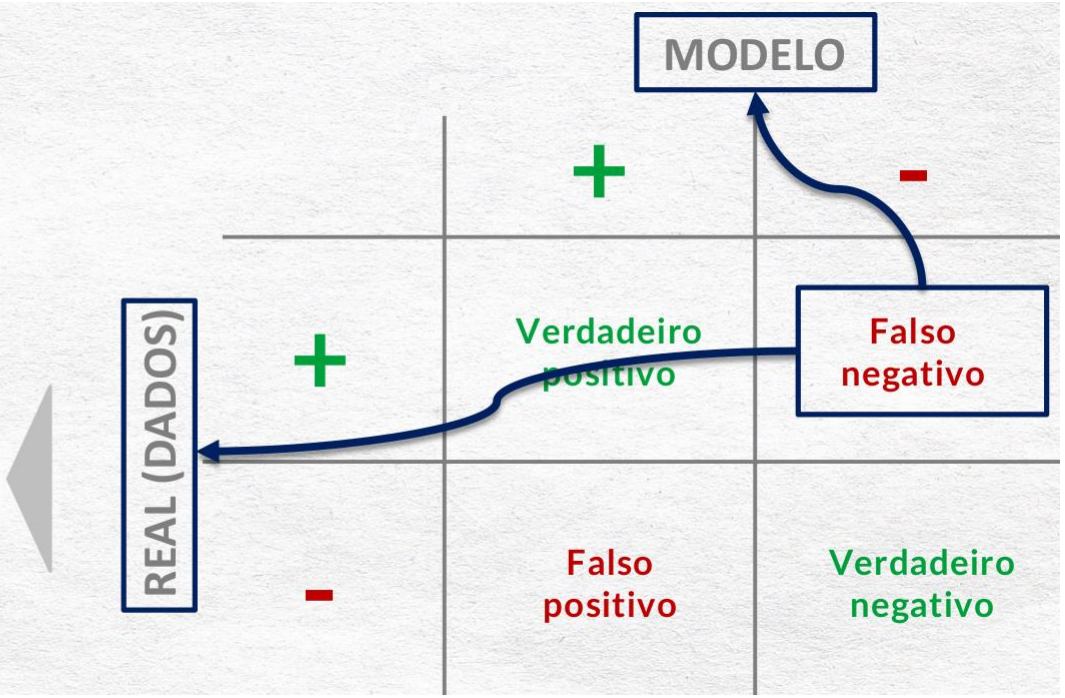
```
array([[1, 2],
       [1, 2]], dtype=int64)
```



Módulo 13 – Matriz de confusão

Agora, temos os erros do modelo. Há duas situações que são de sinal positivo na vida real, mas que o modelo previu como negativas. Logo, há dois falsos negativos.

```
from sklearn.metrics import confusion_matrix
y_true = ['-','-','-','+','+','+']
y_pred = ['-','-','+','+','-','-']
confusion_matrix(y_true, y_pred,labels=['+','-'])

array([[1, 2],
       [1, 2]], dtype=int64)
```

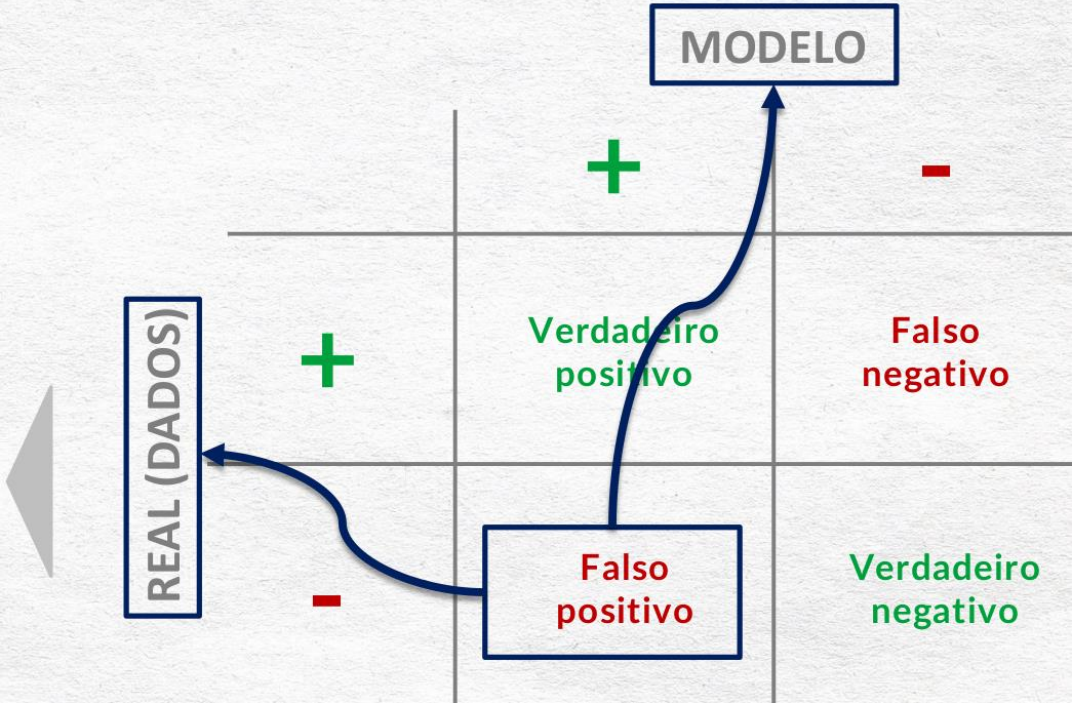


Módulo 13 – Matriz de confusão

Há uma situação de sinal negativo na vida real que o modelo classificou como positivo. Logo, há um falso positivo.

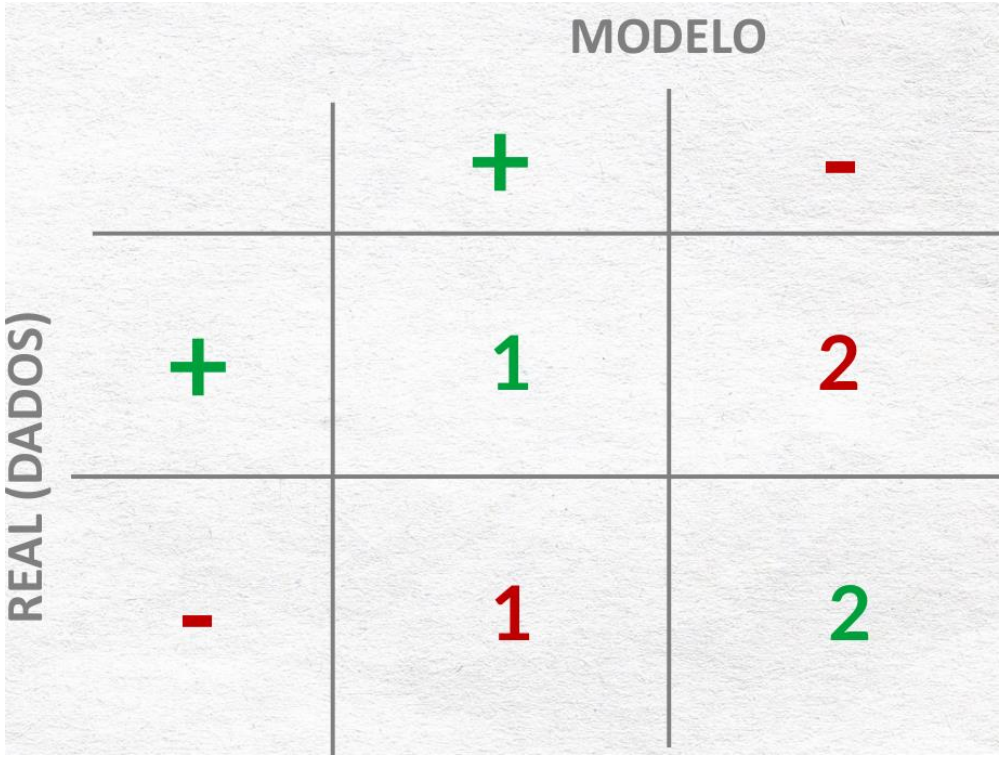
```
from sklearn.metrics import confusion_matrix
y_true = ['-','-','-','+','+','+']
y_pred = ['-','-','+','+','-','-']
confusion_matrix(y_true, y_pred,labels=['+','-'])
```

```
array([[1, 2],
       [1, 2]], dtype=int64)
```



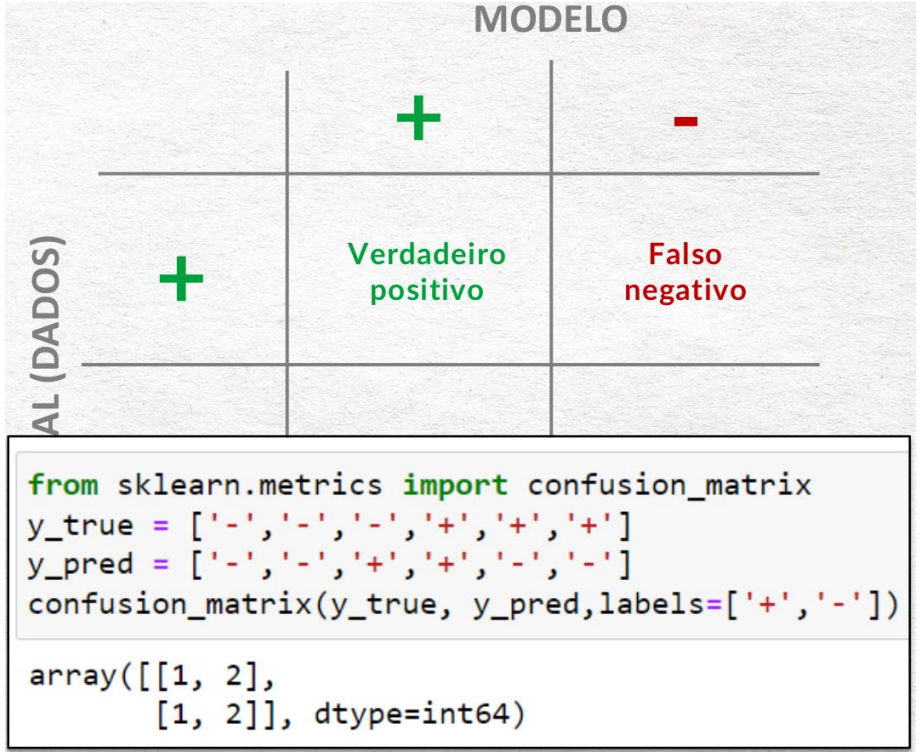
Módulo 13 – Matriz de confusão

Observe nas imagens que o Scikit-Learn apresenta os números na forma de uma matriz. Essa é a forma usual de representar, com os números indicando cada situação.



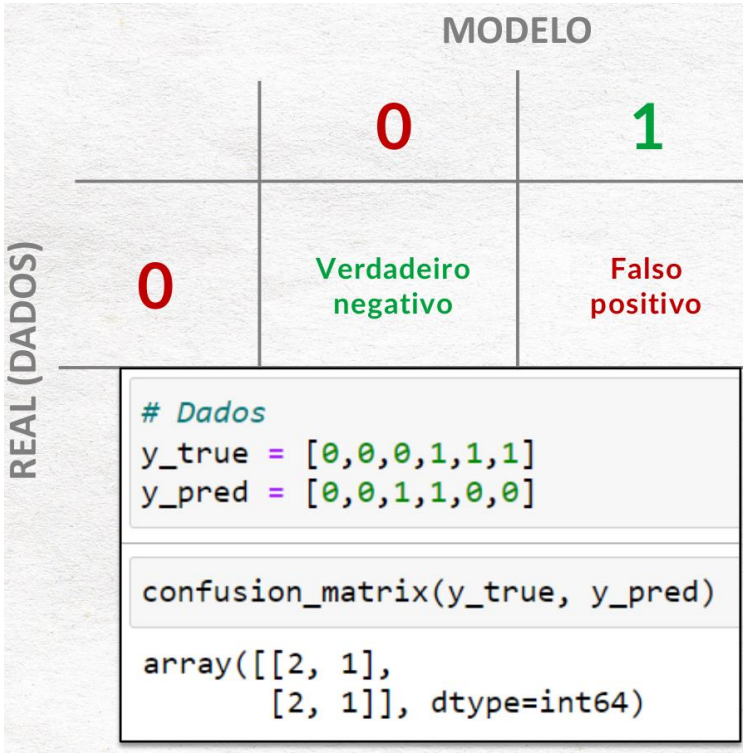
Módulo 13 – Matriz de confusão

Nas páginas anteriores, usamos os símbolos + e – por questões didáticas. Em projetos reais, usualmente as classificações são representadas pelos números 0 e 1. Perceba que, por padrão, o Scikit-Learn coloca o 0 antes do 1, trocando as posições das classificações. Não há problema, o importante é saber o significado da matriz e estar atento a este detalhe.




```
from sklearn.metrics import confusion_matrix
y_true = ['-','-','-','+','+','+']
y_pred = ['-','-','+','+','-','-']
confusion_matrix(y_true, y_pred,labels=['+','-'])
```

```
array([[1, 2],
       [1, 2]], dtype=int64)
```




```
# Dados
y_true = [0,0,0,1,1,1]
y_pred = [0,0,1,1,0,0]
```

```
confusion_matrix(y_true, y_pred)
```

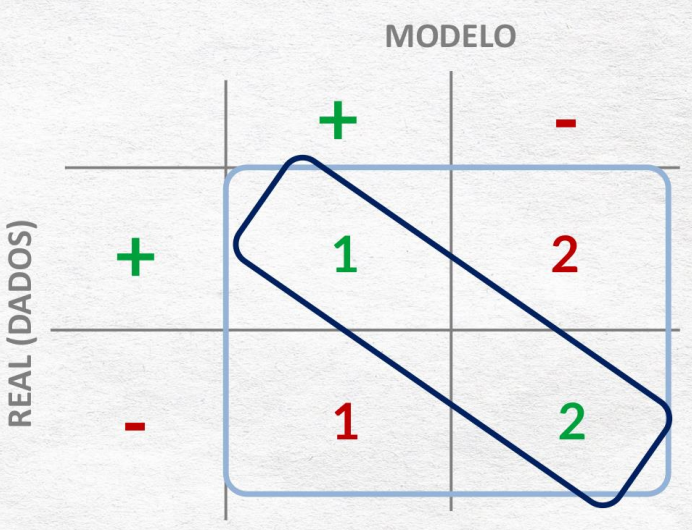
```
array([[2, 1],
       [2, 1]], dtype=int64)
```

Módulo 13 – Acurácia de um modelo de classificação

Quando desenvolvemos um modelo, nossa maior preocupação é saber o quão bom ele é. Daí a necessidade de métricas. A matriz de confusão nos ajuda a entender algumas das principais métricas.

Comecemos com a acurácia: $$\text{Acurácia} = \frac{\text{Verdadeiros positivos} + \text{Verdadeiros negativos}}{\text{Total}}$$

Com os valores do nosso exemplo: $$\text{Acurácia} = \frac{1+2}{1+2+1+2} = 0,5 \therefore 50\%$$



O Scikit-Learn possui uma vasta coleção de métodos para métricas. Podemos importar o `accuracy_score` e obter o mesmo valor acima:

```
from sklearn.metrics import accuracy_score
accuracy_score(y_true, y_pred)
```

```
0.5
```

Módulo 13 – Acurácia de um modelo de classificação

Um cuidado que devemos ter é o de não confiar cegamente em uma única métrica. Por exemplo, uma acurácia de 90 % é boa?

$$\text{Acurácia} = \frac{\text{Verdadeiros positivos} + \text{Verdadeiros negativos}}{\text{Total}}$$

Observe acima novamente a definição de acurácia. Perceba que é uma métrica que considera a fração de acertos no total de predições. Logo, se a base de dados for muito desbalanceada, como no nosso caso ao lado onde 90 % das entradas são de uma das classes, altas acurácias serão obtidas facilmente. Afinal, basta prever que todas as entradas são da classe dominante e já teremos acurácias dessa ordem de grandeza.

Este fenômeno é chamado de **paradoxo da acurácia**. Vale uma busca para ver mais casos interessantes.

```
# Dados
y_true = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1]
y_pred = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_true,y_pred,labels=[1,0])
```

```
array([[ 0,  2],
       [ 0, 18]], dtype=int64)
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_true,y_pred)
```

```
0.9
```

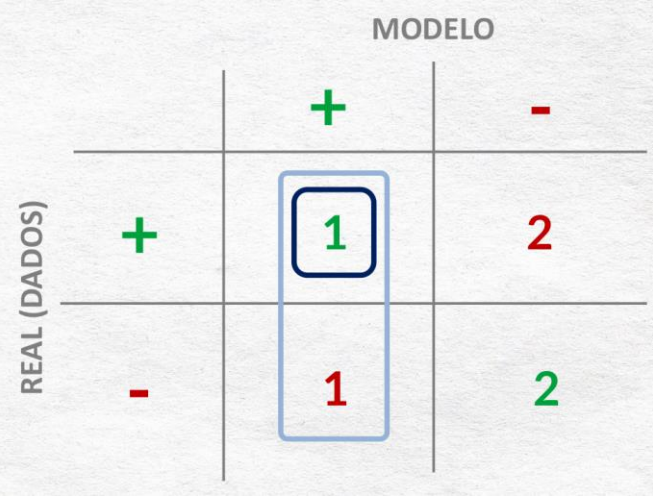
1: é fraude
0: não é fraude

Módulo 13 – Precisão de um modelo de classificação

A precisão é a fração de verdadeiros positivos frente o total de previsões positivas.

$$\text{Precisão} = \frac{\text{Verdadeiros positivos}}{\text{Verdadeiros positivos} + \text{Falsos positivos}}$$

Com os valores do nosso exemplo: $\text{Precisão} = \frac{1}{1+1} = 0,5 \therefore 50\%$



No Scikit-Learn, podemos importar o `precision_score` e obter o mesmo valor acima:

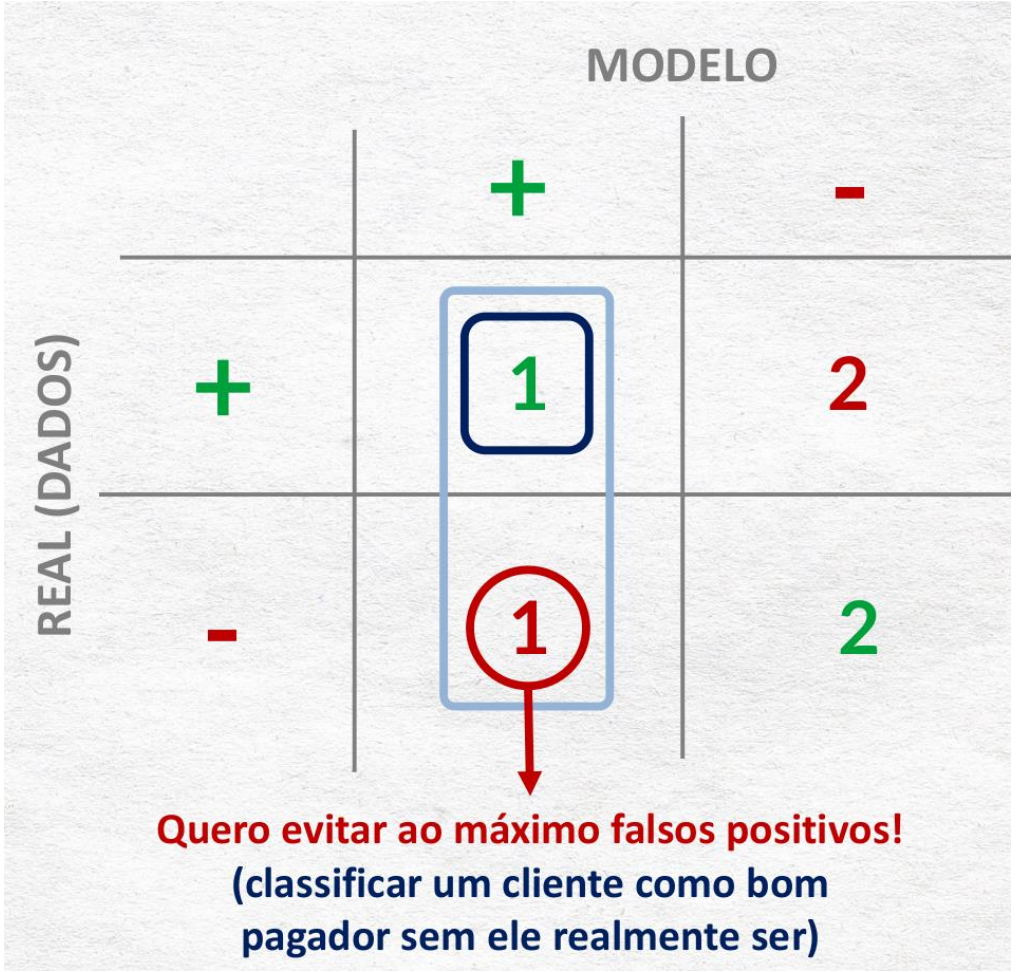
```
from sklearn.metrics import precision_score
precision_score(y_true, y_pred,pos_label='+')
```

```
0.5
```

Módulo 13 – Precisão de um modelo de classificação

Pela definição, vemos que a precisão mede o quanto seu modelo é bom em prever positivos. Buscar maximizar a precisão irá minimizar o número de erros de falsos positivos.

No nosso contexto, uma transação legítima ser considerada fraudulenta significa que o usuário terá seu cartão bloqueado indevidamente, causando conflito e desconforto no cliente.

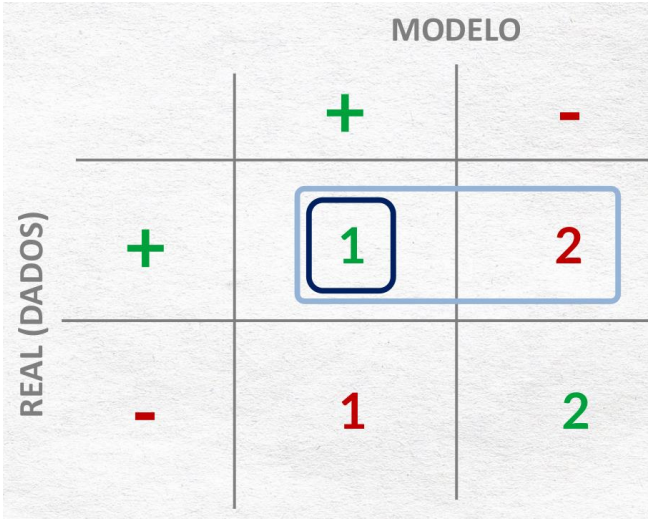


Módulo 13 – Recall de um modelo de classificação

O recall é fração de verdadeiros positivos frente ao total de positivos reais. Afinal, os falsos negativos são, na realidade, positivos.

$$\text{Recall} = \frac{\text{Verdadeiros positivos}}{\text{Verdadeiros positivos} + \text{Falsos negativos}}$$

Com os valores de nosso exemplo: $\text{Recall} = \frac{1}{1+2} = 0,333\ldots \quad \therefore 33,3\%$



No Scikit-Learn, podemos importar o `recall_score` e obter o mesmo valor acima:

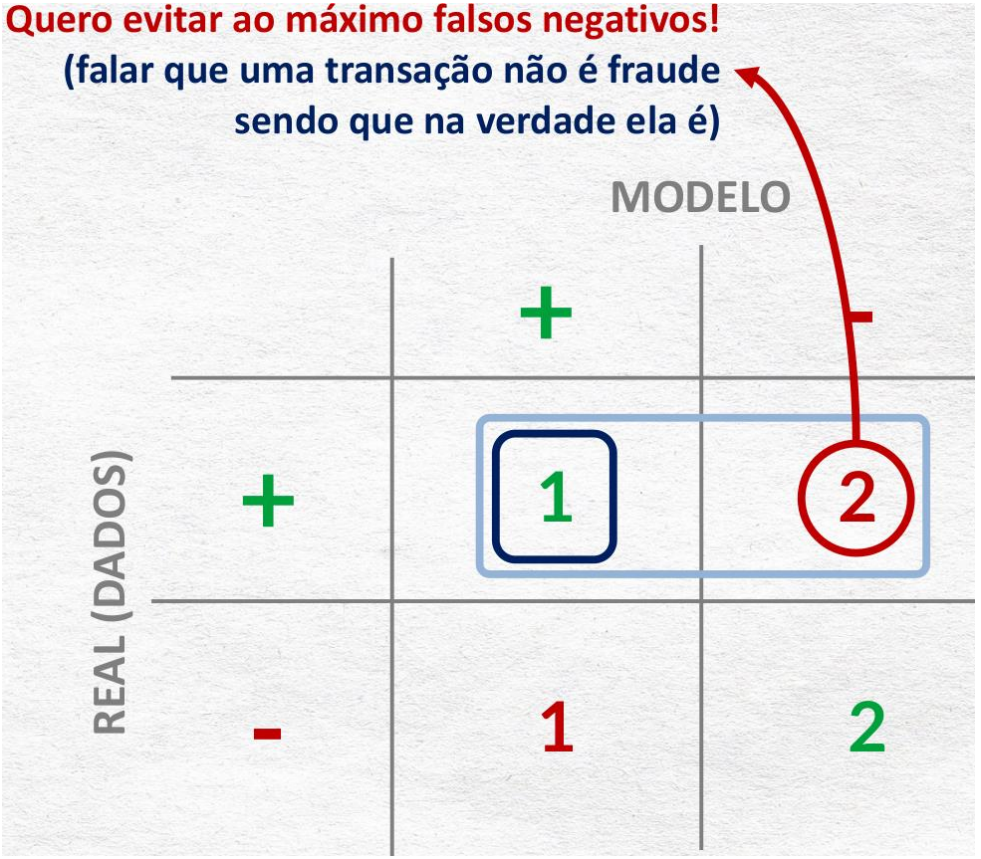
```
from sklearn.metrics import recall_score
recall_score(y_true, y_pred,pos_label='+')
```

```
0.3333333333333333
```

Módulo 13 – Recall de um modelo de classificação

Pela definição, vemos que o recall quantifica a proporção de reais positivos que foram identificados. Buscar maximizar o recall irá minimizar o número de erros de falsos negativos.

No nosso contexto, ao classificar erroneamente uma transação fraudulenta como legítima, um usuário teve seu dinheiro roubado via, por exemplo, clonagem do cartão. A companhia do cartão terá que reembolsar o usuário.



Módulo 13 – Recall de um modelo de classificação

Como já visto, não se deve confiar em uma única métrica. Vamos considerar um caso onde temos um modelo com recall de 100 %. Será que é um bom modelo?

Pela definição, vemos que o recall quantifica a proporção de reais positivos que foram identificados. Buscar maximizar o recall irá minimizar o número de erros de falsos negativos. No nosso caso, maximizar a detecção de fraudes.

Mas observe como a precisão foi muito penalizada. Ou seja, o número de falsos positivos aumentou. Isto também possui um custo associado. Logo, a empresa deve encontrar um modelo que seja um bom custo-benefício entre precisão e recall.

```
# Dados
y_true = [0,0,0,1,1,1]
y_pred = [1,1,1,1,1,1]
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_true,y_pred,labels=[1,0])
```

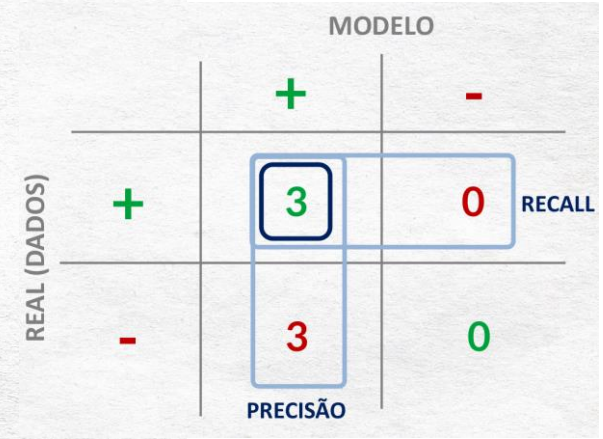
```
array([[3, 0],
       [3, 0]], dtype=int64)
```

```
from sklearn.metrics import precision_score
precision_score(y_true,y_pred)
```

```
0.5
```

```
from sklearn.metrics import recall_score
recall_score(y_true,y_pred)
```

```
1.0
```



1: é fraude
0: não é fraude

Módulo 13 – Conclusão de métricas de um modelo de classificação

Como conclusão dessa parte de métricas, temos que nenhuma métrica possui valor isoladamente.

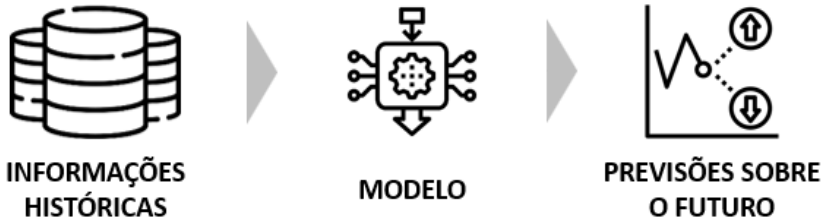
Acabamos de ver as três principais métricas e já percebemos a importância de considerar o problema a ser resolvido e os eventuais custos de cada decisão. Isto é uma constante em ciência de dados e sempre será discutido quando aprendermos novas métricas no decorrer do curso.

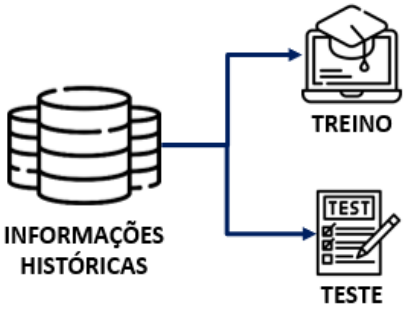
Percebemos claramente a importância de sermos cientistas de dados impressionadores!

Módulo 13 – Separando os dados em treino e teste

Durante toda a explicação até o momento, comparamos um modelo com dados reais. Mas como fazer essa comparação quando recebemos uma base de dados fechada, sem ter uma forma de adquirir novos dados reais posteriormente?

Uma forma é separar a base em partes, uma para treinar o modelo e outra para testar o modelo, esta última fazendo o papel do que seria os novos dados reais. Os dados de treino farão o papel das informações históricas que treinam o modelo enquanto que os de teste irão servir para avaliar qual a qualidade do modelo para previsões.





Módulo 13 – Separando os dados em treino e teste

Para ver como fazer essa separação com o Scikit-Learn, vamos usar uma base de dados que mostra se determinados clientes de um banco são inadimplentes ou não junto com a informação de saldo em conta e de investimentos.

Começamos importando o Pandas e a base da mesma forma que fizemos em outros módulos:

```
1  import pandas as pd
2  base = pd.read_excel("BaseInadimplencia.xlsx")
3  base.head()
```

	SaldoConta	SaldoInvestimento	Situacao
0	-1.365390	-3.280278	0
1	-1.992492	-4.158429	0
2	-3.910816	-0.874096	0
3	-2.745822	-2.250098	0
4	-1.352205	-1.280924	0

Módulo 13 – Separando os dados em treino e teste

Assim como já feito em outros módulos, separamos a base em X (atributos) e y (alvo) e, com o método train_test_split, separamos em dados de treino (*train*) e teste (*test*):

```
1  from sklearn.model_selection import train_test_split
2  X = baseCompleta[['SaldoConta','SaldoInvestimento']]
3  y = baseCompleta['Situacao']
4  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

Podemos verificar o tamanho de cada divisão feita:

```
1  X_train.shape
```

```
(670, 2)
```

```
1  X_test.shape
```

```
(330, 2)
```

Módulo 13 – Avaliando nosso modelo

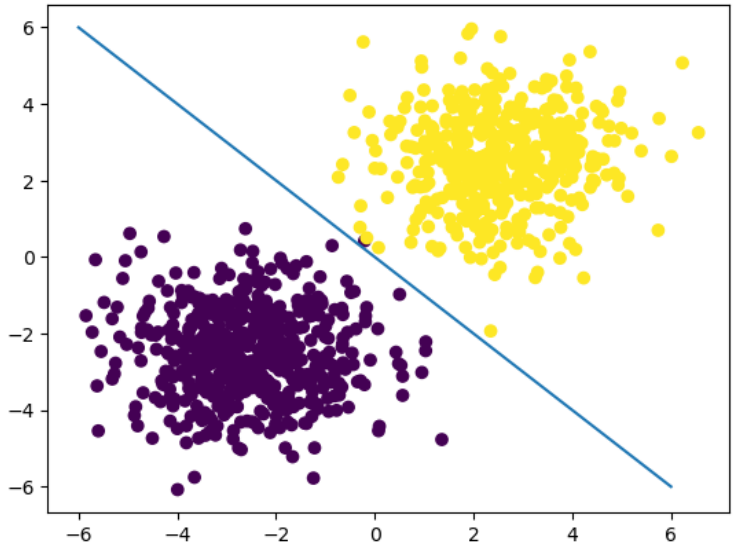
Vamos usar a mesma ideia de traçar uma reta que pode ser utilizada como fronteira de classificação dos nossos dados.

Essa reta pode ser transformada em função e aplicada nos dados, gerando previsões em cima da base de treino e da base de testes:

```python
# Transformando essa reta em uma função para classificar os pontos
def clf(x,y):
    # y_modelo = -x_modelo
    y_modelo = -x

    # Se y_modelo > y_dado -> class: 0
    if y_modelo >= y:
        return 0
    # Se y_modelo < y_dado -> class: 1
    elif y_modelo < y:
        return 1
```

```python
# Aplicando a função aos dados de treino
y_pred_train = X_train.apply(lambda x:clf(x['SaldoConta'],x['SaldoInvestimento']),axis=1)
y_pred_test = X_test.apply(lambda x:clf(x['SaldoConta'],x['SaldoInvestimento']),axis=1)
```

```python
# Criando uma reta capaz de separar esses pontos
import matplotlib.pyplot as plt
fig, ax = plt.subplots()

ax.scatter(X_train.SaldoConta, X_train['SaldoInvestimento'],c=y_train)
ax.scatter(X_test.SaldoConta, X_test['SaldoInvestimento'],c=y_test)
x = [6,-6]
y = [-6,6]
ax.plot(x, y)

plt.show()
```



Módulo 13 – Avaliando nosso modelo

Com o método **head** do Pandas, podemos ver alguns exemplos da classificação feita em cada base. E, com o método **confusion_matrix** do Scikit-Learn, conseguimos construir as matrizes de confusão para treino e teste.

```
# Verificando a base com a nova coluna
y_pred_train.head(5)
```

```
703    1
311    0
722    1
629    1
0      0
dtype: int64
```

```
y_pred_test.head(5)
```

```
521    0
737    1
740    1
660    1
411    0
dtype: int64
```

```
from sklearn.metrics import confusion_matrix
```

```
# Matriz de confusão para a base de treino
confusion_matrix(y_train,y_pred_train)
```

```
array([[359,   1],
       [  0, 310]])
```

```
# Matriz de confusão para a base de teste
confusion_matrix(y_test,y_pred_test)
```

```
array([[183,   0],
       [  0, 147]])
```

Módulo 13 – Avaliando nosso modelo

Agora, podemos utilizar os métodos das métricas que estudamos anteriormente no módulo.

```
1  from sklearn.metrics import accuracy_score
2  # Acurácia para a base de treino
3  accuracy_score(y_train,y_pred_train)
```

```
0.9985074626865672
```

```
1  from sklearn.metrics import precision_score
2  # Precisão para a base de treino
3  precision_score(y_train,y_pred_train)
```

```
0.9967845659163987
```

```
1  from sklearn.metrics import recall_score
2  # Recall para a base de treino
3  recall_score(y_train,y_pred_train)
```

```
1.0
```

```
1  # Acurácia para a base de teste
2  accuracy_score(y_test,y_pred_test)
```

```
1.0
```

```
1  # Precisão para a base de teste
2  precision_score(y_test,y_pred_test)
```

```
1.0
```

```
1  # Recall para a base de teste
2  recall_score(y_test,y_pred_test)
```

```
1.0
```

Como vemos, os valores são elevados, na base de teste todas as métricas foram de 100 %. Isto é coerente com o que vimos graficamente, onde a reta separava bem as duas classes.

Mas será esse o melhor modelo? Como podemos comparar modelos?

Módulo 13 – Avaliando nosso modelo

Em um projeto real, usualmente testamos mais de um modelo para vermos qual é o mais adequado. Não apenas sob o ponto de vista de métricas, mas também do ponto de vista de performance (velocidade e consumo de recursos computacionais).

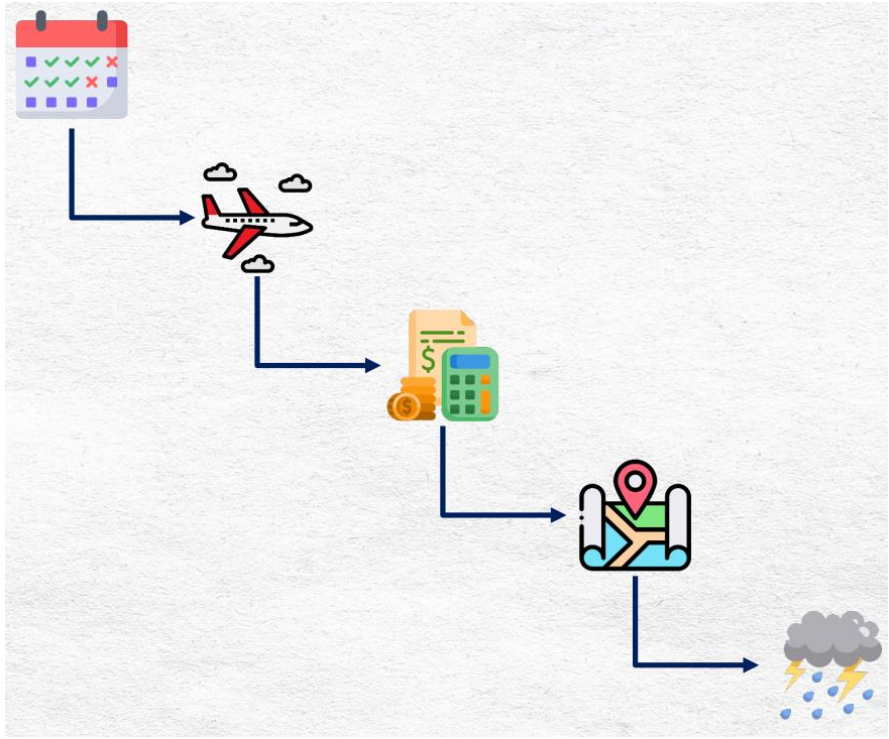
Agora, iremos conhecer um outro modelo, o de árvore de decisão, e comparar as métricas obtidas.

Módulo 13 – O que é uma árvore de decisão?

O modelo que vamos estudar agora é o árvore de decisão.

Para entender a ideia geral do modelo, imagine que você foi convidado para uma festa. Provavelmente, você irá considerar alguns fatores para decidir se vai ou não ao evento. Por exemplo:

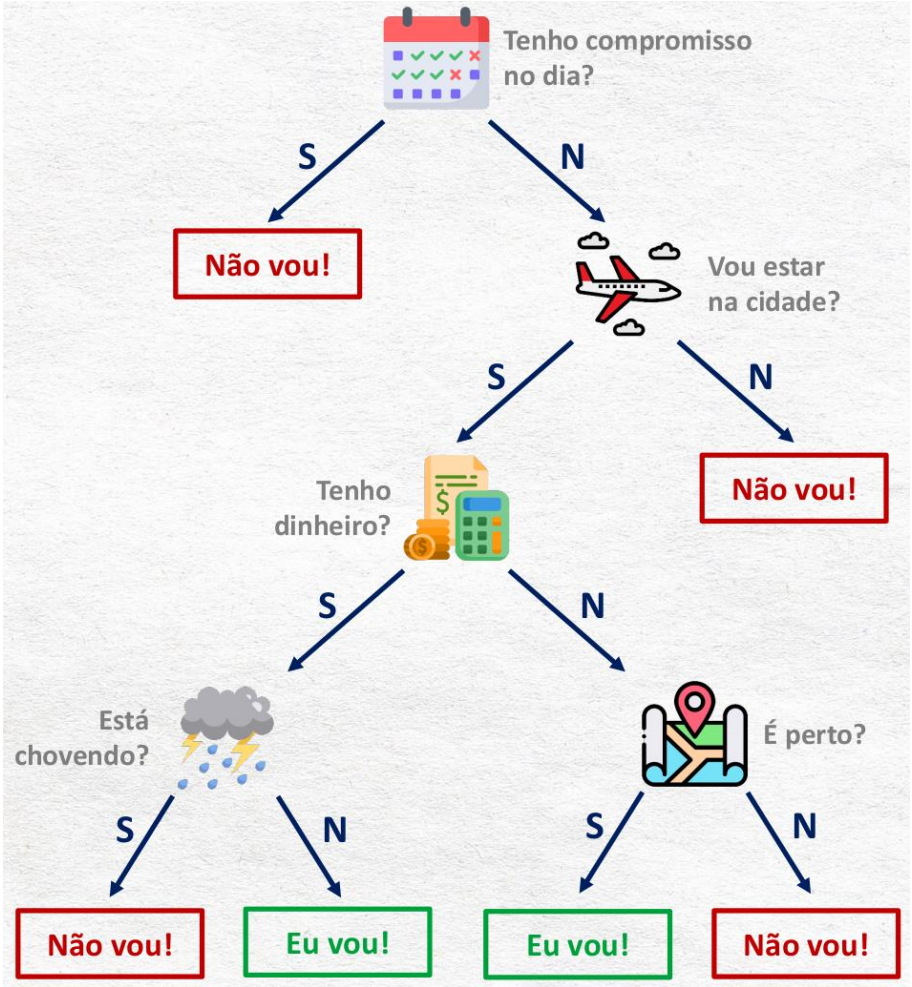
- Está livre no dia?
- Precisa viajar?
- Tem dinheiro?
- O local é de fácil acesso e seguro?
- Qual a previsão do tempo para o dia?



Módulo 13 – O que é uma árvore de decisão?

Podemos elencar estes fatores na forma de uma árvore de decisões, cujos **ramos** vão crescendo a depender das respostas que damos a cada um dos questionamentos feitos nos **nós**.

Temos o **nó raiz** (compromisso no dia?), passando pelos **nós folha** (perguntas intermediárias), até chegar aos **nós finais** (perguntas finais). Também são utilizados os termos *nós internos* e *nós terminais* para os nós folha e finais, respectivamente.



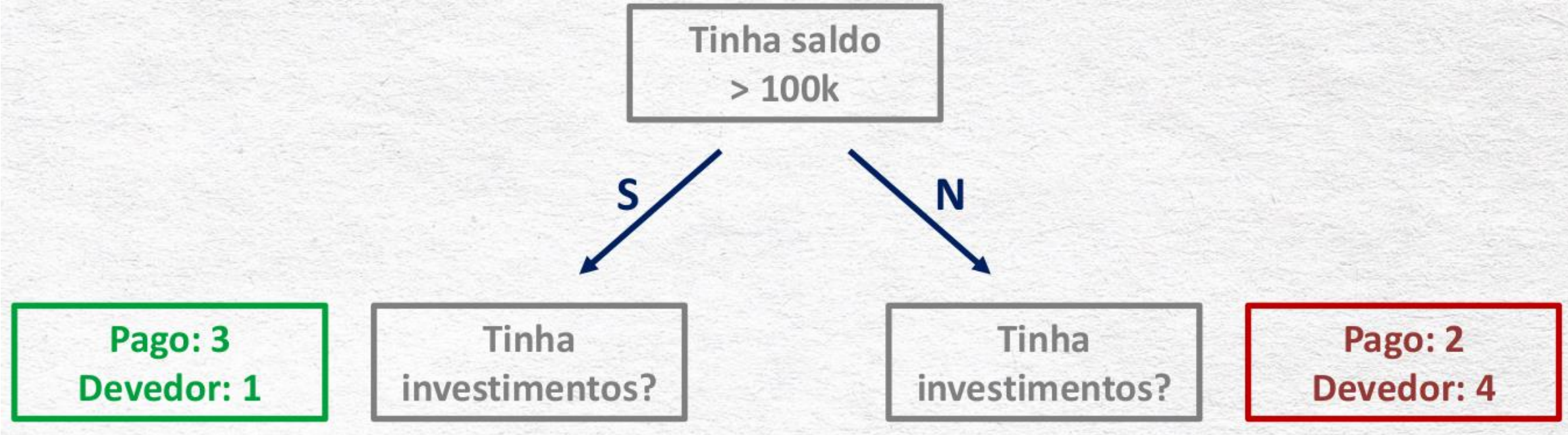
Módulo 13 – O que é uma árvore de decisão?

Vamos considerar o seguinte exemplo: uma instituição financeira quer desenvolver um modelo de previsão de possíveis clientes devedores. Ela possui dados históricos de clientes de saldo em conta, se os clientes possuem investimentos ou não, e se possuem casa própria ou não.

Tinha saldo > 100k	Tinha investimentos	Tinha casa própria	Situação empréstimo
Sim	Sim	Sim	Pago
Não	Sim	Sim	Pago
Não	Não	Sim	Pago
Sim	Não	Sim	Pago
Não	Não	Não	Devedor
Não	Sim	Não	Devedor
Não	Sim	Não	Devedor
Não	Não	Não	Devedor
Sim	Sim	Não	Pago
Sim	Não	Não	Devedor

Módulo 13 – O que é uma árvore de decisão?

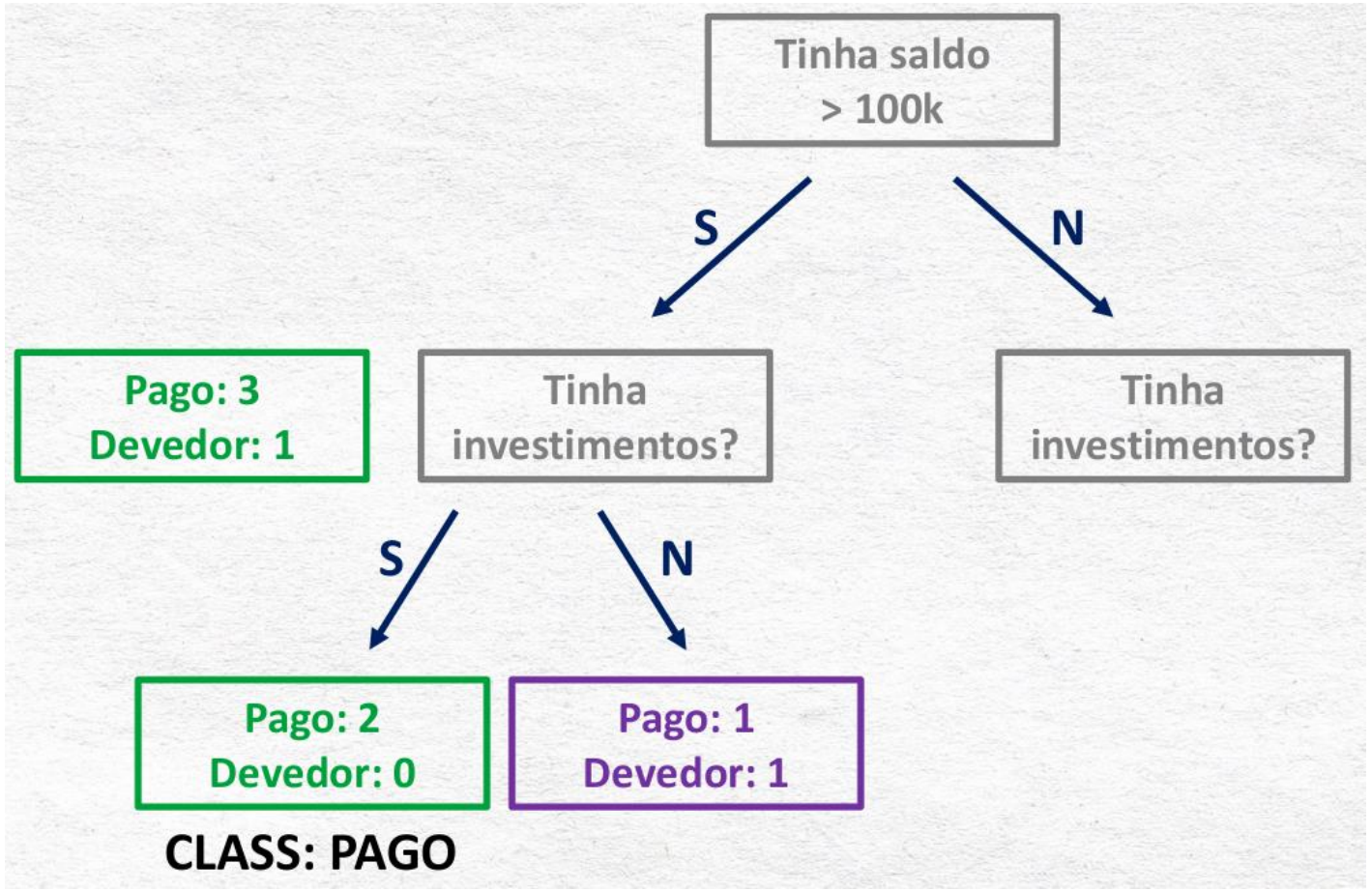
Podemos tentar construir uma árvore de decisão com base nos dados que temos. Primeiro, temos que decidir qual será nossa primeira pergunta. Vamos começar com o saldo bancário:



Agora temos dois ramos e nenhum deles chegou em uma classificação definitiva.

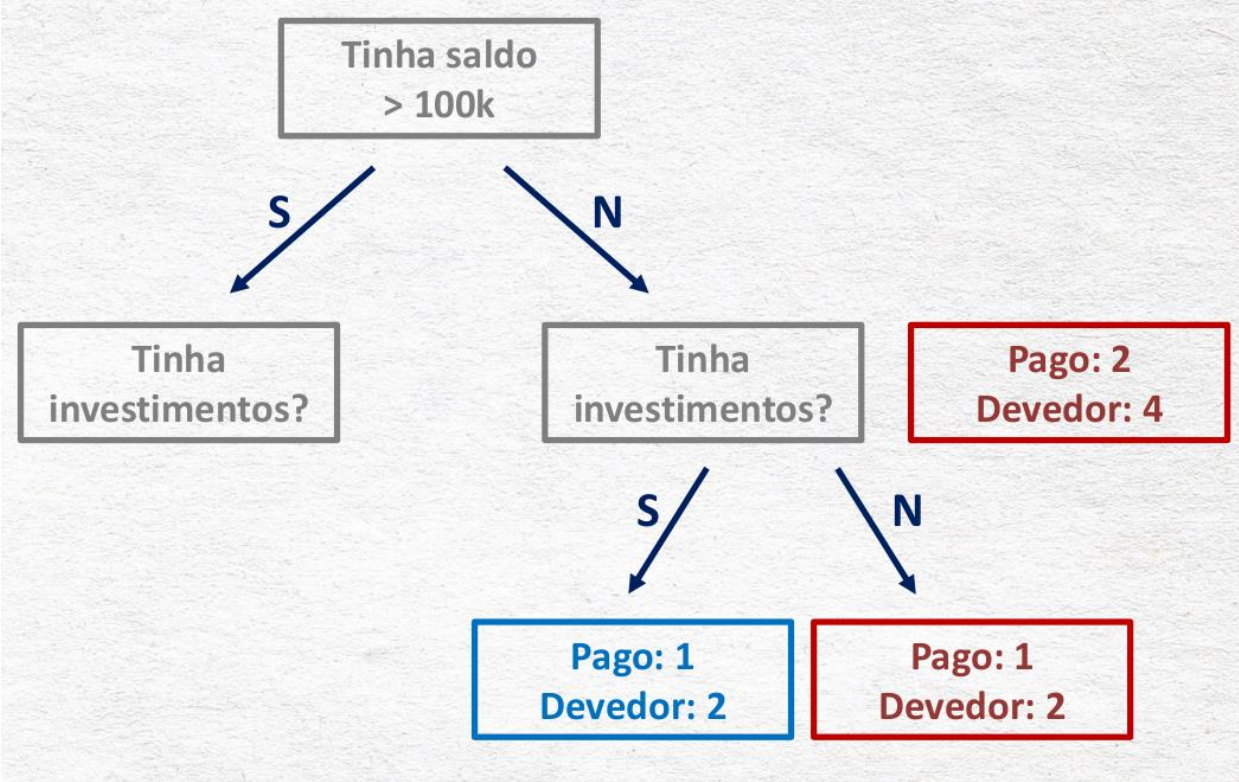
Módulo 13 – O que é uma árvore de decisão?

Se estendermos o ramo da esquerda, chegamos em uma classificação definitiva, onde identificamos um caminho que permite identificar clientes bom pagadores.



Módulo 13 – O que é uma árvore de decisão?

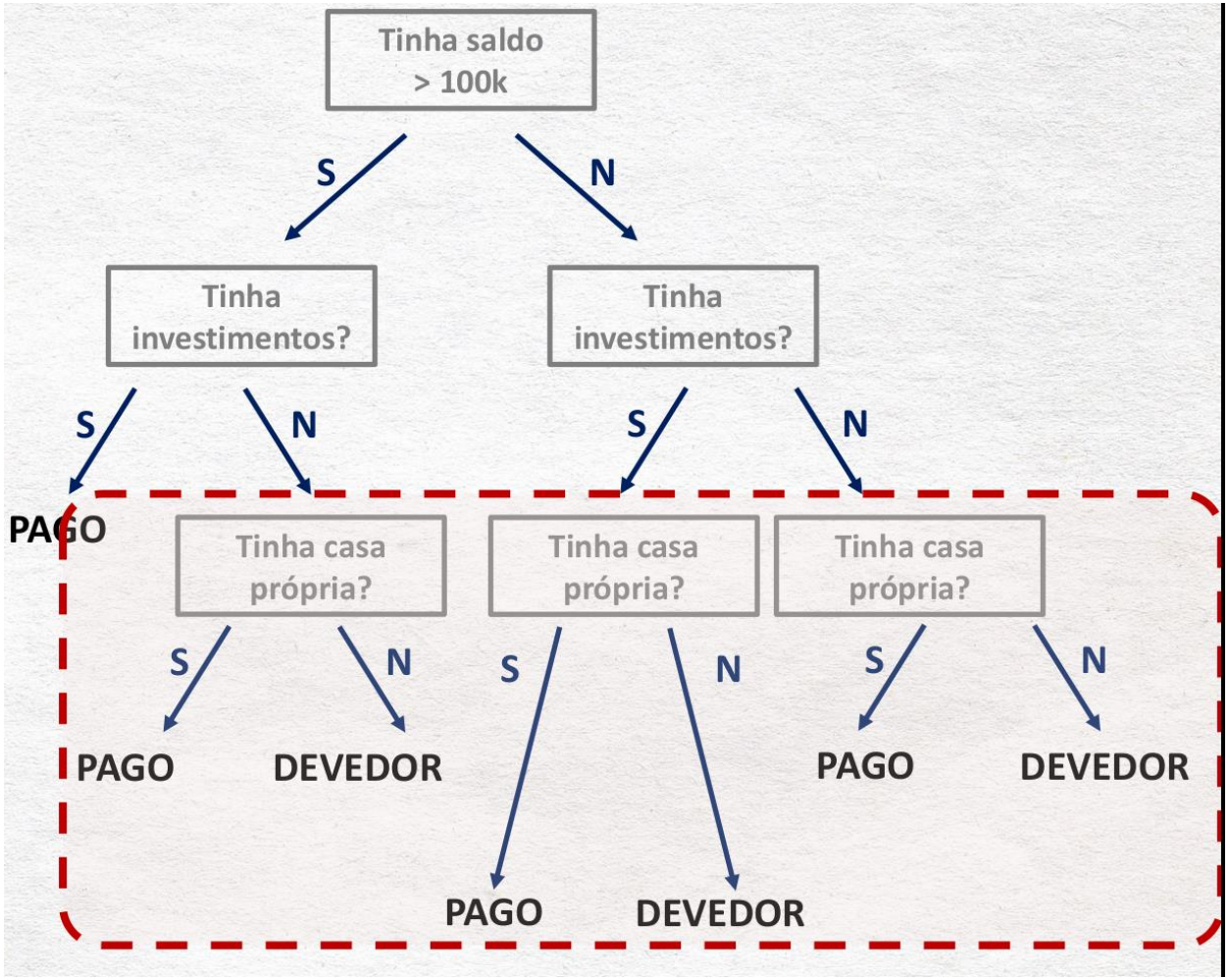
Indo para o ramo da direita, ainda não chegamos em uma classificação final ao responder o questionamento sobre investimentos. Ainda há mais um parâmetro para adicionar à árvore: a posse ou não de casa própria pelo cliente.



Módulo 13 – O que é uma árvore de decisão?

Veja que a adição destes nós permitiu classificar todos os clientes.

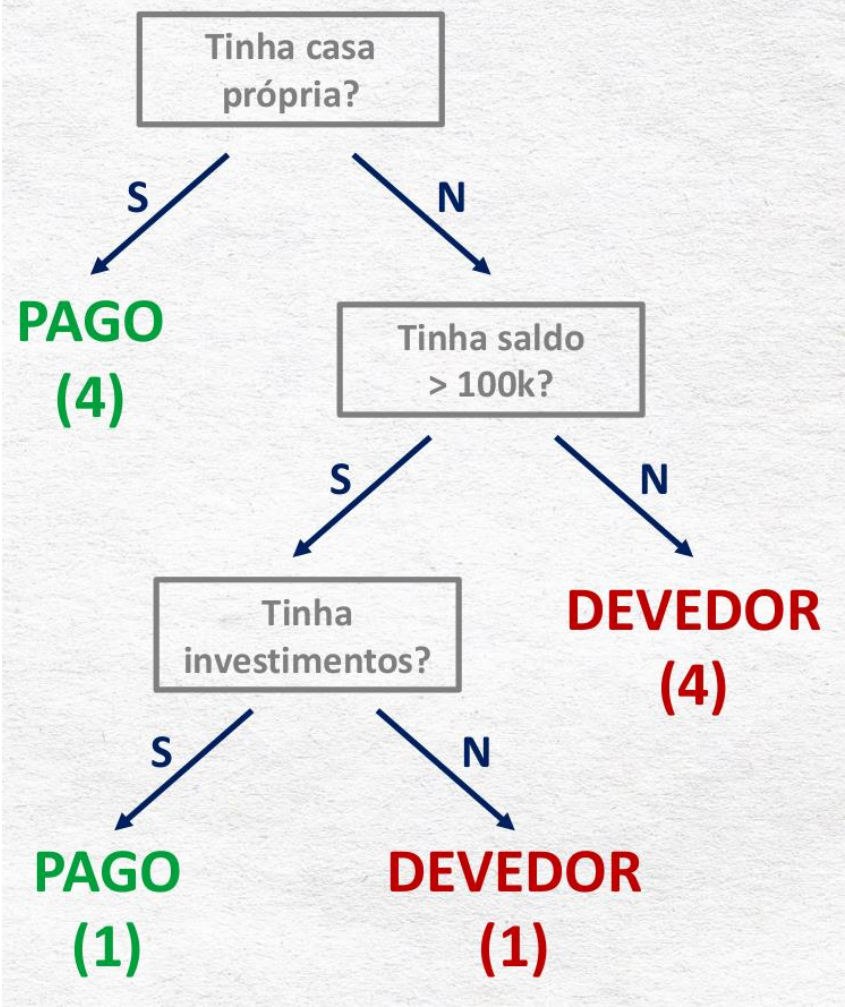
Isso nos faz pensar: será que se essa pergunta fosse nosso nó raiz, teríamos uma árvore menor?



Módulo 13 – O que é uma árvore de decisão?

Veja como colocando a análise da casa logo no início, temos quase todos os bons pagadores classificados rapidamente.

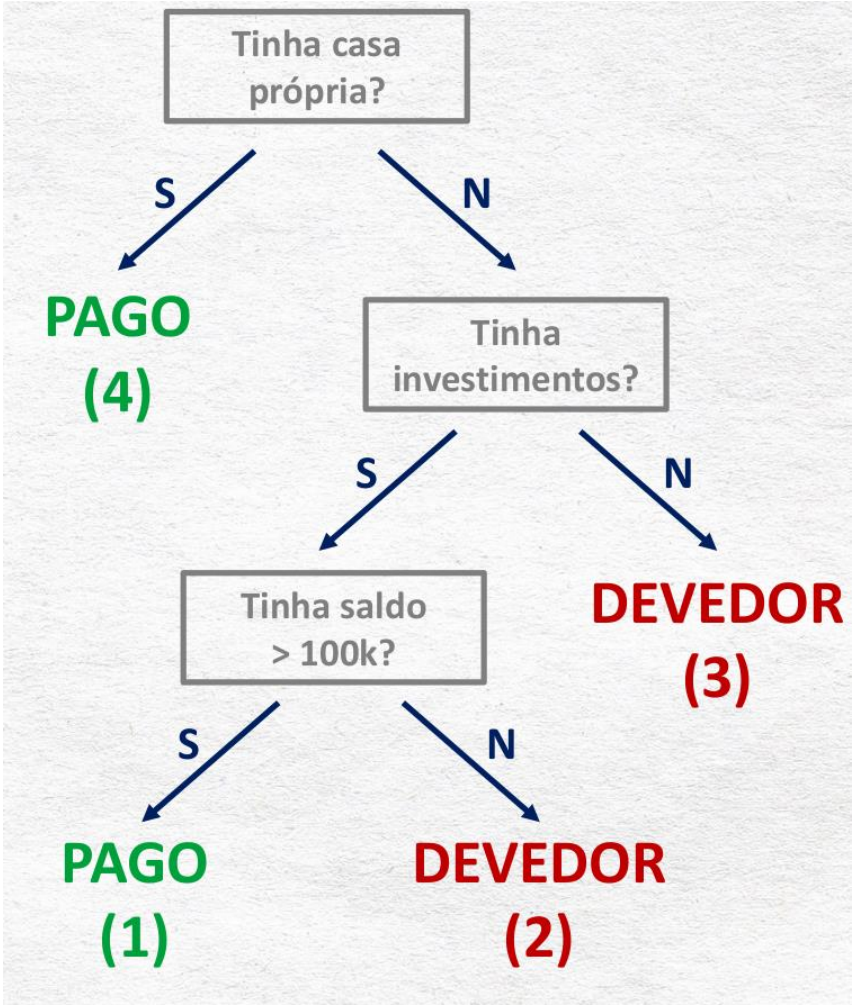
Ao questionar o saldo de investimentos, todos com menos de 100 mil são devedores. Por fim, a análise de investimentos classifica os demais.



Módulo 13 – O que é uma árvore de decisão?

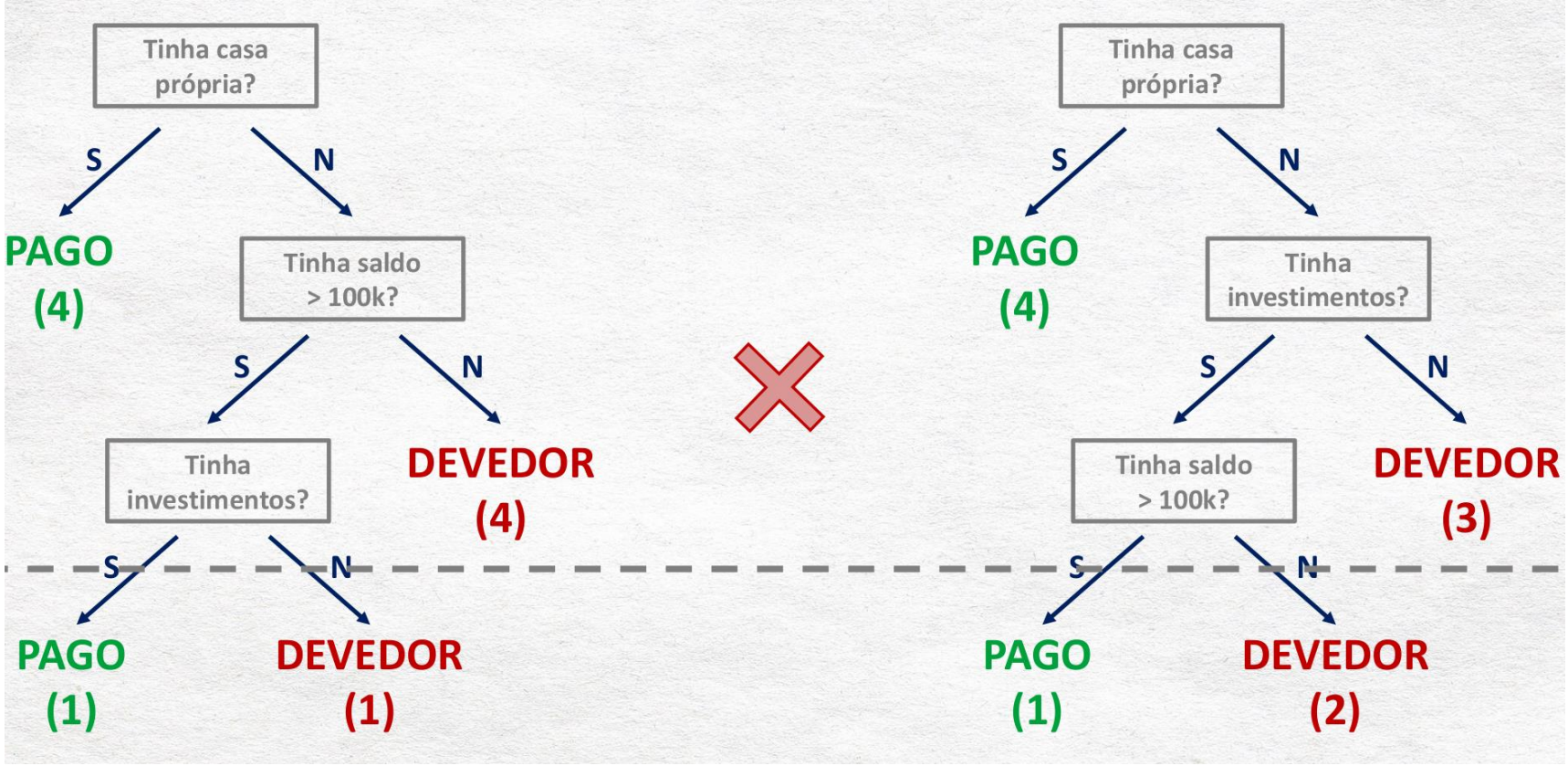
Trocando a ordem dos dois últimos nós, temos um resultado similar.

Mas qual é o melhor modelo? O da página anterior ou o ao lado?



Módulo 13 – O que é uma árvore de decisão?

Considere os dois lado a lado. Vemos que o da esquerda com duas perguntas já classificou 8 dos 10 clientes. Isto significa que, se precisarmos diminuir o número de nós para tornar o modelo mais rápido, ele será um modelo provavelmente mais eficaz.



Módulo 13 – Reutilizando o dateset íris

Vamos novamente importar o dataset íris para fazer a comparação entre o modelo de regressão linear e o de árvore de decisão.

Assim como anteriormente, vamos deixar com duas classes apenas e fazer nossa separação entre base de treino e de teste. A única diferença é que desta vez excluiremos a classe 0, pois as 1 e 2 não são tão facilmente separadas por uma reta. Dessa forma, será uma boa forma de comparar os modelos:

```
import pandas as pd
```

```
# Importando o dataset iris
from sklearn.datasets import load_iris
data = load_iris()
```

```
# Transformando em um DataFrame
iris = pd.DataFrame(data.data)
iris.columns = data.feature_names
iris['target'] = data.target
iris = iris[iris.target != 0]
iris.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
50	7.0	3.2	4.7	1.4	1
51	6.4	3.2	4.5	1.5	1
52	6.9	3.1	4.9	1.5	1
53	5.5	2.3	4.0	1.3	1
54	6.5	2.8	4.6	1.5	1

```
X = iris.drop('target',axis=1)
y = iris.target
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Módulo 13 – Reutilizando o dateset íris

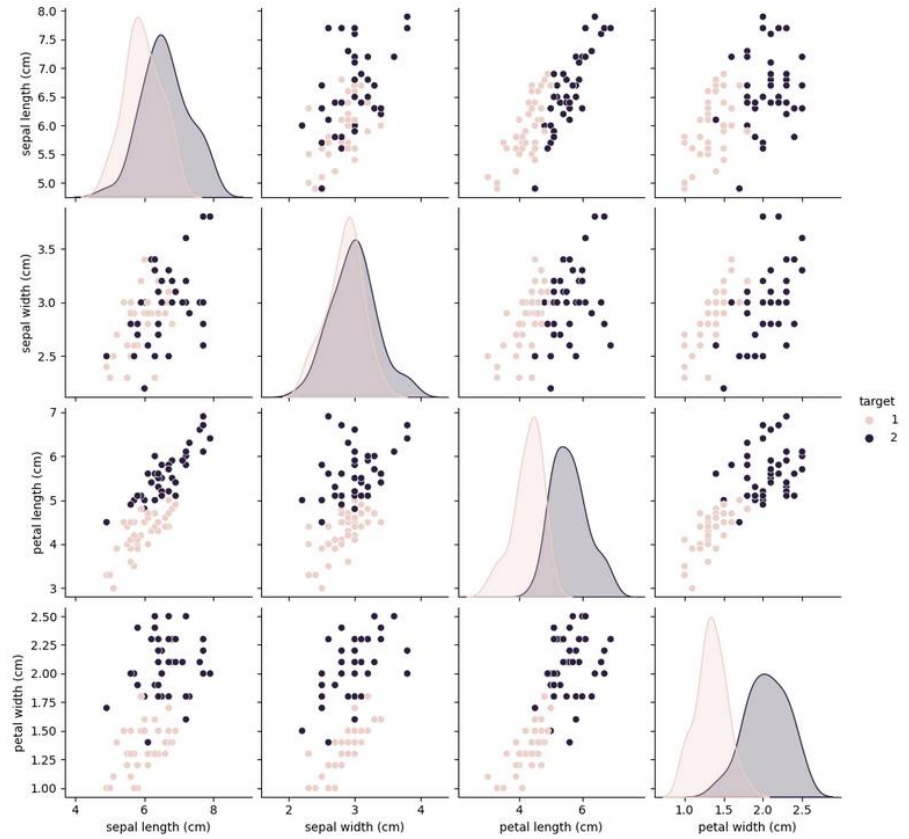
Apenas para visualização, vamos concatenar os X e y de teste e criar um `pairplot`.

Vemos como, de uma forma geral, é possível distinguir as duas classes, com pouca sobreposição das mesmas.

```
1  X_view = pd.concat([X_train,y_train],axis=1)
2  X_view.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
105	7.6	3.0	6.6	2.1	2
138	6.0	3.0	4.8	1.8	2
76	6.8	2.8	4.8	1.4	1
92	5.8	2.6	4.0	1.2	1
119	6.0	2.2	5.0	1.5	2

```
1  import seaborn as sns
2  sns.pairplot(X_view,hue='target');
```

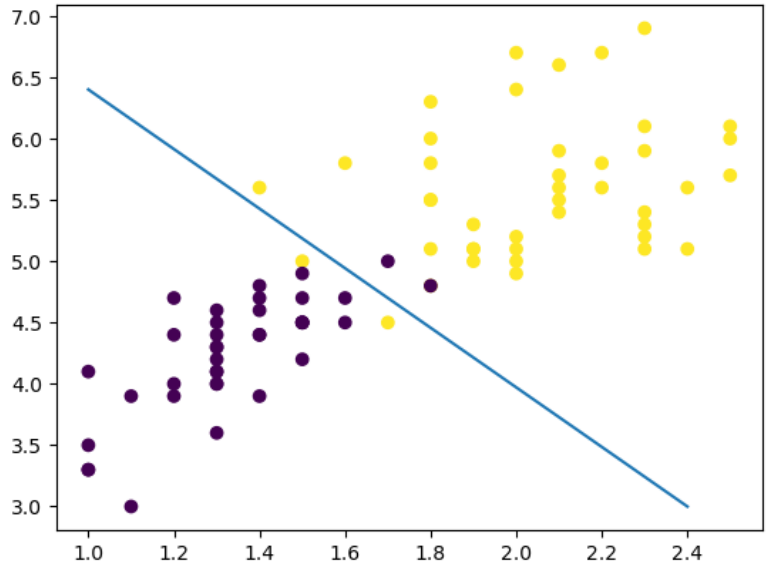


Módulo 13 – Criando uma reta

De forma similar ao que já fizemos em outros datasets, vamos visualmente traçar uma reta que separa boa parte dos pontos de acordo com a coloração das classes e criar uma função de classificação de acordo com a posição dos pontos:

```python
a=-1.7/0.7
b=3-2.4*a

def clf_reta(x,y):
    y_modelo = a*x + b

    # Se y_ponto > y_funcao: class 2
    if y >= y_modelo:
        return 2
    # Se y_ponto < y_funcao: class 1
    elif y < y_modelo:
        return 1
```

```python
# plot
fig, ax = plt.subplots()

ax.scatter(X_train['petal width (cm)'], X_train['petal length (cm)'], c=y_train.values)

# Criar o plot de uma reta
x_reta = [2.4,1]
y_reta = [3,6.4]
ax.plot(x_reta,y_reta)

plt.show()
```



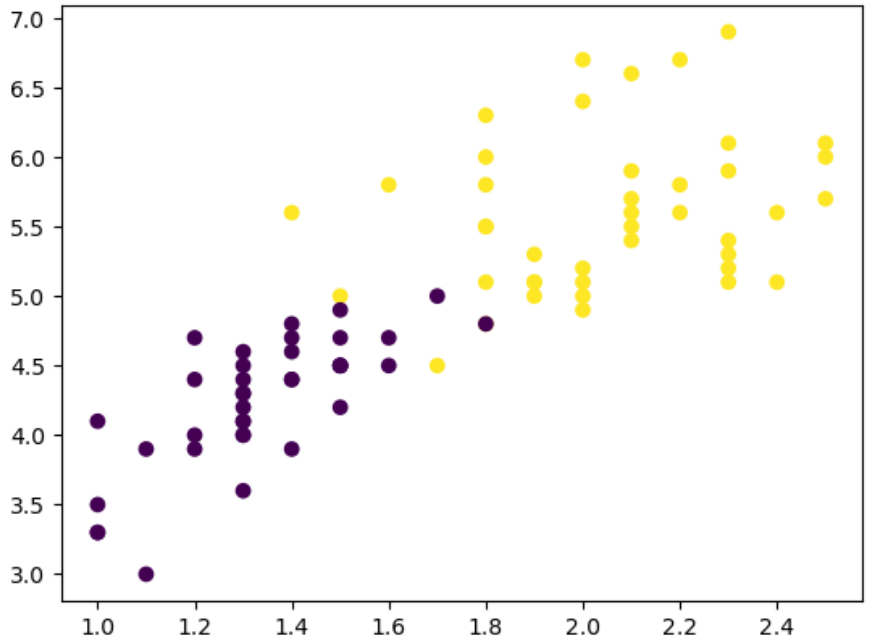
Módulo 13 – Criando uma árvore de decisão

Para a árvore de decisões, precisamos definir alguns nós. Ou seja, alguns valores de x ou y que permitem separar nossas classes.

Visualmente, vemos que sempre que x >= 1.9 temos a classe amarela (2) e sempre que x <= 1.3 temos a classe roxa (1). Como essas duas comparações separam as classes por completo, colocamos elas no início, como os nós de cima da árvore. Para y, vemos que o valor de y = 5 separa razoavelmente as classes. Mas colocando essa comparação ao final, as comparações de x já terão sido feitas e temos uma classificação boa das classes:

```python
def clf_arvore(x,y):
    if x >= 1.9:
        return 2
    elif x <= 1.3:
        return 1
    elif y <= 5:
        return 1
    elif y > 5:
        return 2
```

```python
# plot
fig, ax = plt.subplots()

ax.scatter(X_train['petal width (cm)'], X_train['petal length (cm)'], c=y_train.values)

# ax.set(xlim=(1.35,1.85),ylim=(5, 8))

plt.show()
```



Módulo 13 – Comparando os modelos

Com nossas funções criadas, aplicamos aos nossos dados de treino e de teste e podemos, finalmente, começar a avaliar nossas métricas:

```
# Dados de treino
y_modelo_reta_treino = X_train.apply(lambda x:clf_reta(x['petal width (cm)'],x['petal length (cm)']),axis=1)
y_modelo_arvore_treino = X_train.apply(lambda x:clf_arvore(x['petal width (cm)'],x['petal length (cm)']),axis=1)
```

```
# Dados de teste
y_modelo_reta_teste = X_test.apply(lambda x:clf_reta(x['petal width (cm)'],x['petal length (cm)']),axis=1)
y_modelo_arvore_teste = X_test.apply(lambda x:clf_arvore(x['petal width (cm)'],x['petal length (cm)']),axis=1)
```

Como vimos, as métricas são decorrentes da matriz de confusão. Desta forma, vamos começar construindo as matrizes para os modelos:

```
# Para a reta
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_modelo_reta_teste)
```

```
array([[11,  1],
       [ 1,  7]])
```

```
# Para a árvore
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_modelo_arvore_teste)
```

```
array([[11,  1],
       [ 3,  5]])
```

Módulo 13 – Comparando os modelos

Agora, podemos verificar as três métricas que estudamos: acurácia, precisão e recall.

Como vemos, a reta é consistentemente melhor que a árvore em todas as métricas. Ou seja, para este caso específico, comparando as funções que criamos, a função de separação linear ter melhor resultado que a função de árvore de decisão.

```
# Para a reta
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_modelo_reta_teste)
```

```
0.9
```

```
# Para a árvore
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_modelo_arvore_teste)
```

```
0.8
```

```
# Para a reta
from sklearn.metrics import precision_score
precision_score(y_test, y_modelo_reta_teste,pos_label=2)
```

```
0.875
```

```
# Para a árvore
from sklearn.metrics import precision_score
precision_score(y_test, y_modelo_arvore_teste,pos_label=2)
```

```
0.8333333333333334
```

```
# Para a reta
from sklearn.metrics import recall_score
recall_score(y_test, y_modelo_reta_teste,pos_label=2)
```

```
0.875
```

```
# Para a árvore
from sklearn.metrics import recall_score
recall_score(y_test, y_modelo_arvore_teste,pos_label=2)
```

```
0.625
```

Isto não significa que sempre será assim. Estamos olhando estas funções neste dataset. Mudar as funções e o dataset podem gerar resultados bem distintos.

Daí o papel do cientista de dados, avaliar os resultados e a necessidade de novos estudos a depender dos resultados e da necessidade do cliente.

MÓDULO 14

Análise exploratória de dados

Módulo 14 – Entendendo o dataset Titanic

Neste módulo, utilizaremos a base de dados de passageiros do famoso Titanic. É uma excelente base para aprendermos o básico de tratamento de dados e de visualização.

Módulo 14 – Entendendo o dataset Titanic

O primeiro passo é conhecer o dicionário de dados da base. O dicionário descreve o nome de cada coluna e sua respectiva descrição. Como mostrado na aula, tal base foi obtida do Kaggle, de onde traduzimos o conteúdo. As colunas dessa base de dados e seus significados são:

- **Passenger ID**: ID do passageiro (número único para cada um dos passageiros)
- **Survived**: sobrevivente (0 = Não, 1 = Sim)
- **Pclass**: Classe da passagem (1 = primeira classe, 2 = segunda classe, 3 = terceira classe)
- **Name**: nome do passageiro
- **Sex**: Gênero do passageiro
- **Age**: Idade (em anos) do passageiro
- **SibSp**: número de irmãos / cônjuges a bordo do Titanic
- **Parch**: número de pais / filhos a bordo do Titanic
- **Ticket**: código do bilhete de embarque
- **Fare**: tarifa da passagem
- **Cabin**: código da cabine
- **Embarked**: porto de embarque (C = Cherbourg, Q = Queenstown, S = Southampton)

Módulo 14 – Analisando as informações da base

Já vimos alguns projetos antes, e o começo é igual aos demais. Iremos importar nossa base com o `read_csv` do Pandas:

```
# Importando o pandas
import pandas as pd
```

```
# Importando a base de dados
base = pd.read_csv('train.csv')
```

E agora começaremos um procedimento de reconhecimento de nossa base, que deve ser feito sempre, em qualquer projeto de ciência de bases que estiver trabalhando.

Módulo 14 – Analisando as informações da base

É sempre importante verificar se a importação foi bem feita. Para isso, olhamos as primeiras e últimas linhas do dataframe criado com o intuito de ver se há algo estranho que possa indicar problemas no arquivo de origem da base. No caso, aparentemente está tudo dentro do que se espera para o conteúdo das colunas conforme suas definições:

```
1  # Visualizando as 3 primeiras linhas
2  base.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S

```
1  # Visualizando as 3 últimas linhas
2  base.tail(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.45	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75	NaN	Q

Módulo 14 – Analisando as informações da base

Outra verificação importante é o tamanho da base com o atributo **shape**. Usualmente recebemos esta informação da fonte da base, de forma que temos mais um indício se a importação funcionou.

Além disso, com o método **info**, podemos verificar o tipo de cada coluna. Isto é importante, especialmente para se certificar de que colunas numéricas foram efetivamente reconhecidas como tal (inteiro, *int64*, ou float, *float64*), para fazer operações numéricas com as mesmas. Colunas de texto são ditas tipo *object* no Pandas. Compare os tipos com as descrições das colunas para se certificar de que estão corretas.

```
1  # Verificando o tamanho da base
2  base.shape
```

```
(891, 12)
```

```
1  # Verificando as informações
2  base.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Módulo 14 – Analisando as informações da base

Agora, vejamos se há valores nulos nas colunas. Dessa forma, podemos verificar se nossa base tem muitos dados ausentes e começar a pensar em como tratar estas situações. Como vemos, a coluna *Cabin* possui diversos valores ausentes, e também há na coluna *Age*:

```
1  # Contando a quantidade de valores nulos
2  base.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

Módulo 14 – Analisando as informações da base

Uma atenção especial deve ser dada aos dados numéricos. É sempre importante verificar se há valores negativos, zeros, muito pequenos, muito grandes, dispersos... Uma primeira noção pode ser obtida com o método `describe`. Ele apresenta uma estatística básica das colunas numéricas com: contagem, média, desvio padrão, valor mínimo, quartis, e valor máximo.

```
1  # Verificando as informações estatísticas
2  base.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Módulo 14 – Analisando as informações da base

Outra análise importante é de valores únicos por coluna. Aqui, verificamos que algumas colunas possuem apenas 2 ou 3 valores, coerente com o dicionário de dados, e que a coluna *PassengerId* é realmente uma identificação única dos passageiros, sem duplicatas, já que o tamanho da base é de 891 linhas.

```
1  # Verificando a quantidade de valores diferentes na base
2  base.nunique()
```

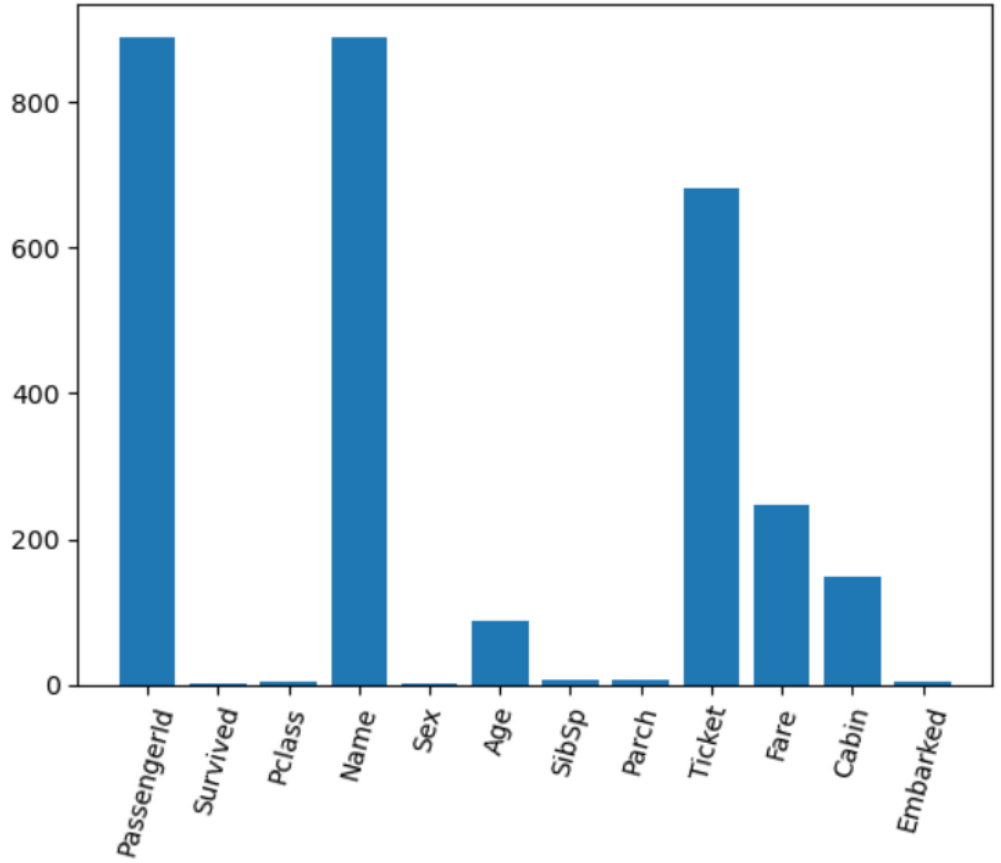
```
PassengerId    891
Survived         2
Pclass           3
Name           891
Sex              2
Age             88
SibSp            7
Parch            7
Ticket         681
Fare           248
Cabin          147
Embarked         3
dtype: int64
```

Módulo 14 – Criando visualizações

Aqui vamos fazer nossa primeira visualização. Podemos pegar o `index` e o `values` dos valores únicos e construir um gráfico de barras de valores únicos por coluna:

```
# Importando o matplotlib
import matplotlib.pyplot as plt
```

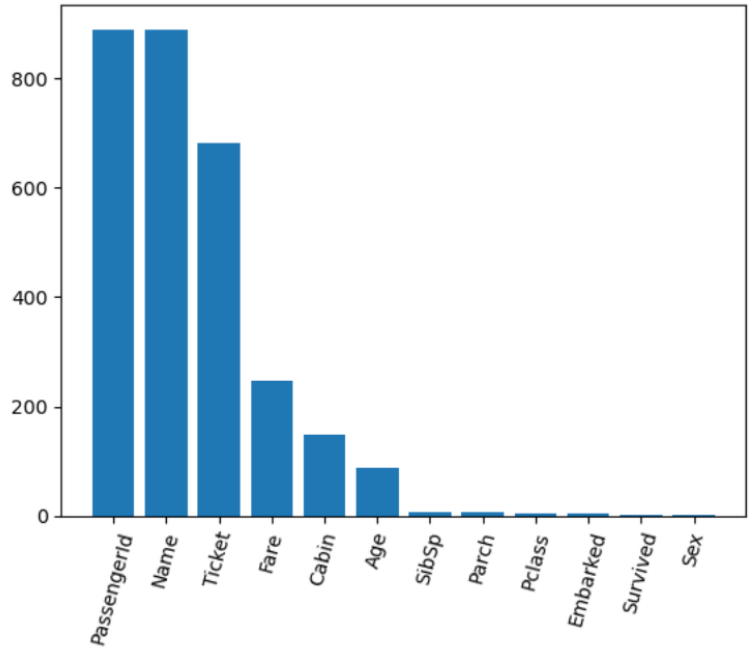
```
# Visualizando a informação acima de forma gráfica
fig, ax = plt.subplots()

ax.bar(base.nunique().index, base.nunique().values)

ax.tick_params(axis='x',labelrotation=75)

plt.show()
```



Módulo 14 – Criando visualizações

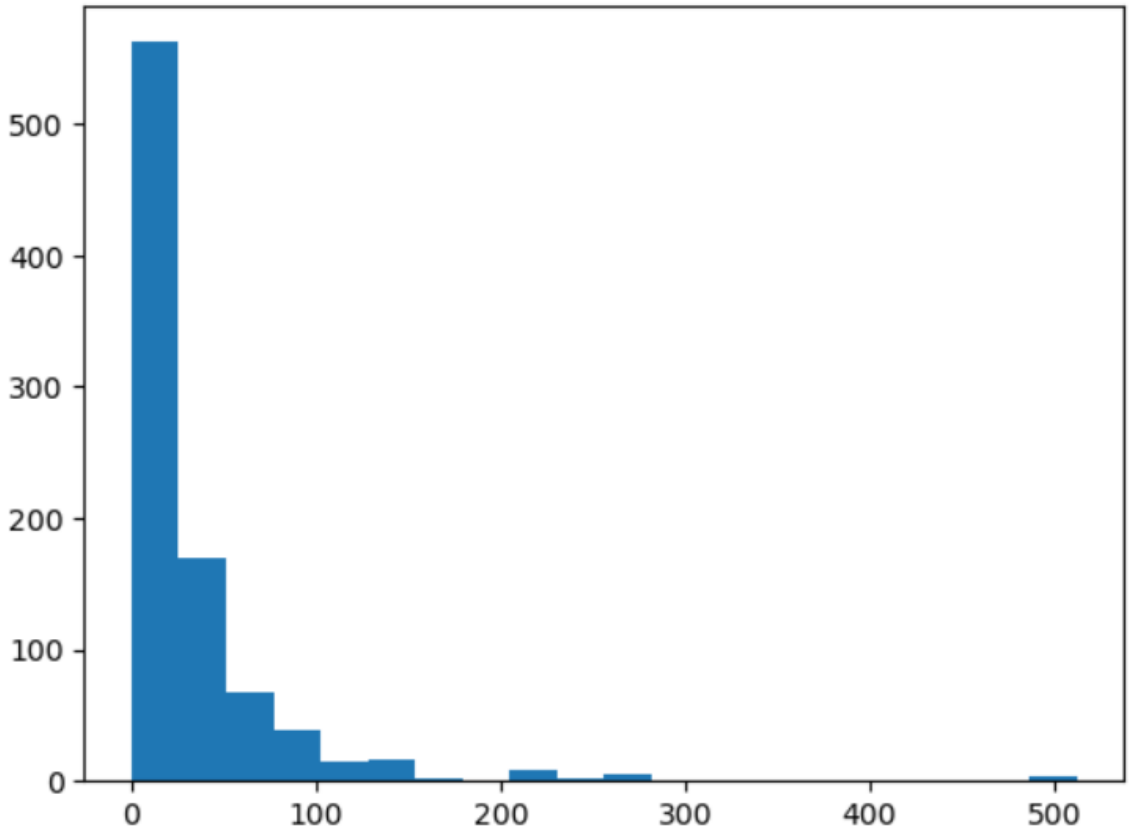
A visualização é mais efetiva se as barras estiverem em ordem, seja crescente ou decrescente. No caso, com o parâmetro `ascending=False`, as barras ficarão em ordem decrescente de valores únicos.

```
1  # Visualizando a informação acima de forma gráfica em ordem decrescente
2  fig, ax = plt.subplots()
3
4  ax.bar(base.nunique().sort_values(ascending=False).index, base.nunique().sort_values(ascending=False).values)
5
6  ax.tick_params(axis='x',labelrotation=75)
7
8  plt.show()
```



Uma análise interessante é a de preço das passagens. O Matplotlib possui o método `hist` para construir histogramas. No código abaixo, construímos um histograma solicitando que os valores da coluna *Fare* sejam divididos em 20 grupos (`bins`) de igual largura.

```
# Verificando o histograma das tarifas
fig, ax = plt.subplots()

ax.hist(base.Fare, bins=20)

plt.show()
```

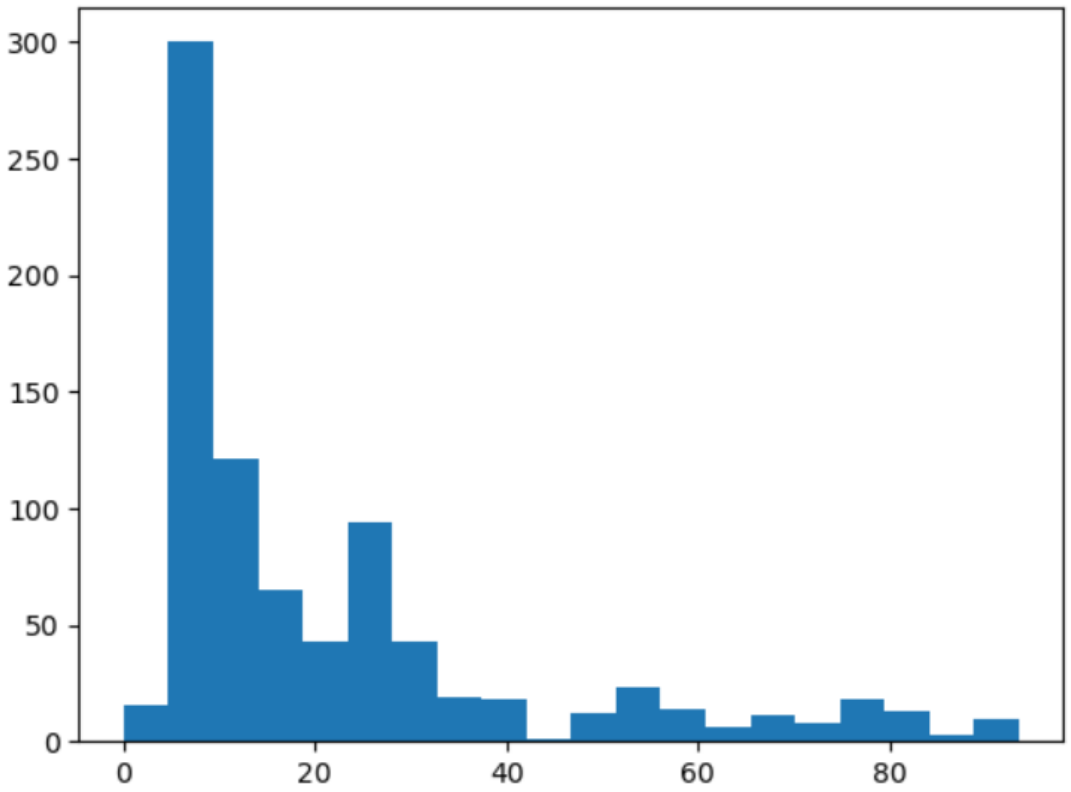


Observe como há uma grande concentração de valores abaixo de 100. Podemos, então, avaliar com mais detalhes essa faixa de valores.

Módulo 14 – Criando visualizações

Com o conhecimento de Pandas que já adquirimos em outros módulos, podemos aplicar um filtro na coluna *Fare* de forma a pegar apenas valores menores do que 100. Assim, teremos um histograma para essa faixa de valores, permitindo um maior detalhamento. Veja como há uma maior concentração em valores abaixo de 20, com o pico por volta de 5.

```
1  # Verificando o histograma das tarifas apenas para tarifas menores que 100
2  fig, ax = plt.subplots()
3
4  ax.hist(base[base.Fare <100].Fare, bins=20)
5
6  plt.show()
```



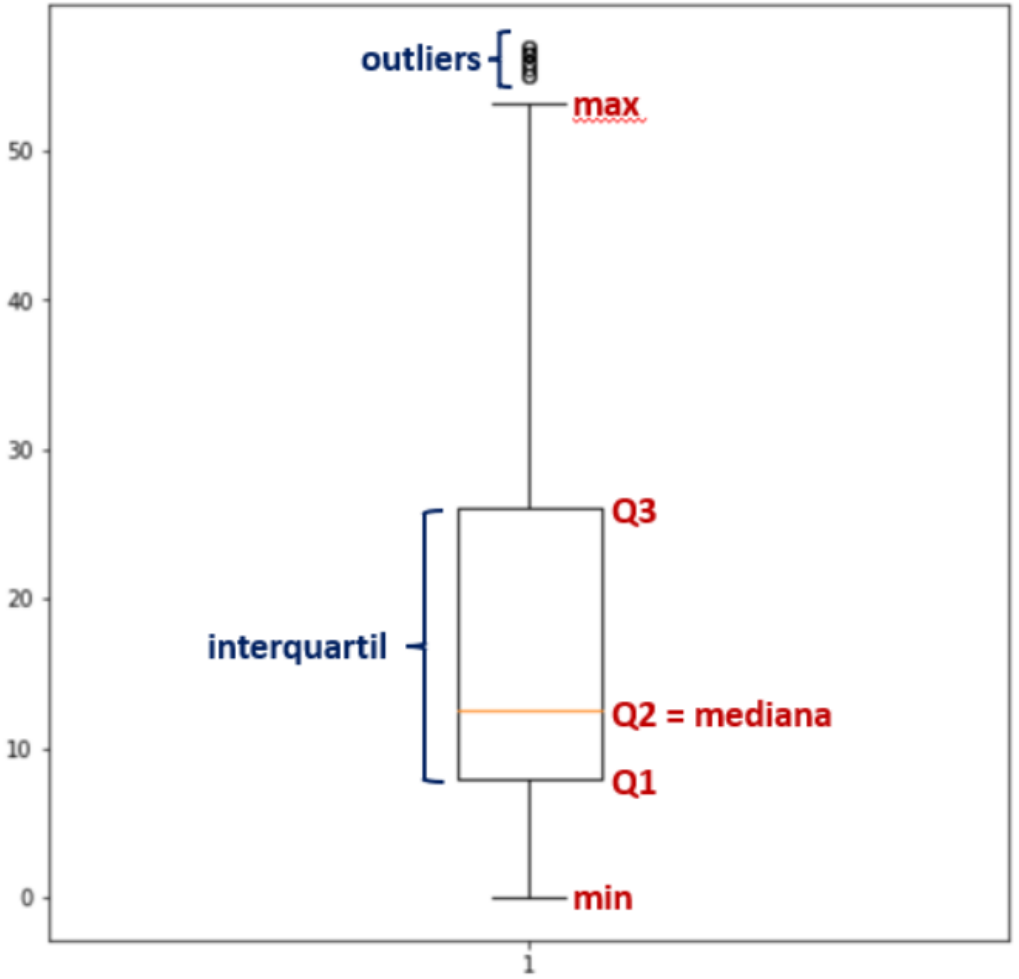
Módulo 14 – Criando visualizações

Vimos que o método `describe` fornece um resumo estatístico das colunas numéricas de nossa base na forma de tabela. Um gráfico **boxplot** é uma maneira visual de obter boa parte destas informações estatísticas.

Em um boxplot temos um resumo visual da distribuição dos dados a partir de 5 valores numéricos: limite mínimo, quartis 1, 2 (mediana) e 3, e limite máximo.

A amplitude interquartil, AIQ, é a diferença *Q3-Q1*, representando 50 % de todos os valores observados, concentradas na tendência central dos valores.

Os chamados *outliers* são valores que se encontram abaixo ou acima de *1,5 x AIQ*. No exemplo ao lado, temos outliers superiores, acima do limite máximo.

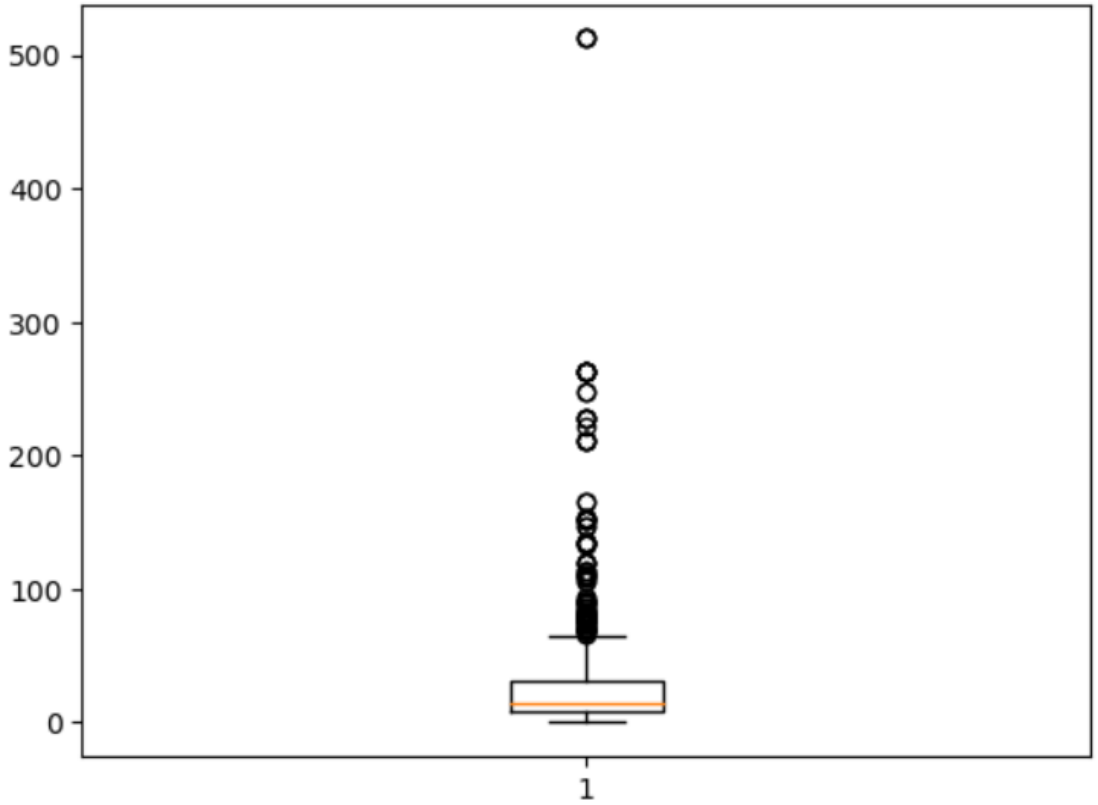


Módulo 14 – Criando visualizações

O Matplotlib possui o método `boxplot`, para o qual podemos passar a coluna da nossa base que queremos estudar.

Veja se consegue interpretar o boxplot com base no que foi resultado do método `describe` anteriormente e com base no histograma já visto. Na próxima página, esta comparação será feita e você pode verificar se sua interpretação foi correta.

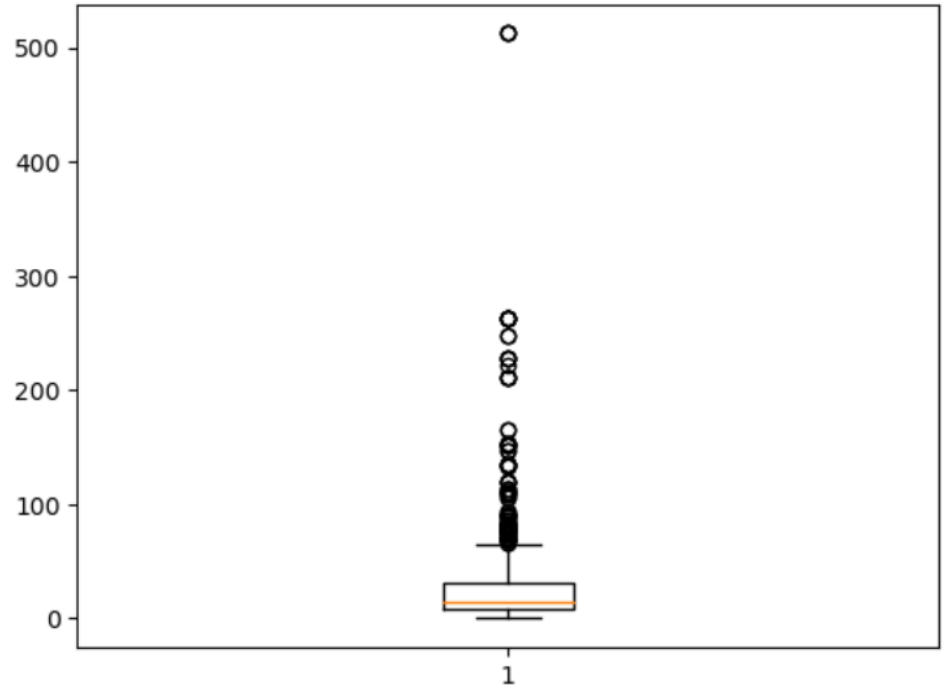
```
1  # Verificando para a coluna Fare
2  fig, ax = plt.subplots()
3
4  ax.boxplot(base.Fare)
5
6  plt.show()
```



Compare o gráfico com a tabela do `describe` que vimos anteriormente. Verifique que há uma concordância entre os valores da tabela e as linhas do boxplot. Observe, também, que o boxplot indica que mais de 50 % dos valores são baixos, algo que o histograma já nos dava indícios. Comece a se acostumar a reconhecer essa complementariedade das diferentes visualizações.

```
1  # Podemos acessar qualquer um dos valores do .describe()
2  base.Fare.describe()
```

```
count    891.000000
mean      32.204208
std       49.693429
min        0.000000
25%        7.910400
50%       14.454200
75%       31.000000
max      512.329200
Name: Fare, dtype: float64
```



Na página onde apresentamos um boxplot, falamos do conceito de amplitude interquartil e de como ele se relaciona com o conceito de outliers. Podemos calcular esta amplitude e verificar qual seria o limite máximo de valor que não se considera outlier:

```
# Determinando o interquartil
Q1 = base.Fare.describe()['25%']
Q3 = base.Fare.describe()['75%']
interquartil = Q3 - Q1

# Calculando o valor máximo
vlr_max = Q3 + 1.5*interquartil
```

Com isso, podemos olhar quais entradas de nossa base são outliers.

Módulo 14 – Criando visualizações

Para olhar os outliers, basta filtrar com o Pandas por valores acima deste limite máximo:

```
1  # Agora filtrando os valores acima do máximo
2  base[base.Fare > vlr_max]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
27	28	0	1	Fortune, Mr. Charles Alexander	male	19.0	3	2	19950	263.0000	C23 C25 C27	S
31	32	1	1	Spencer, Mrs. William Augustus (Marie Eugenie)	female	NaN	1	0	PC 17569	146.5208	B78	C
34	35	0	1	Meyer, Mr. Edgar Joseph	male	28.0	1	0	PC 17604	82.1708	NaN	C
52	53	1	1	Harper, Mrs. Henry Sleeper (Myna Haxtun)	female	49.0	1	0	PC 17572	76.7292	D33	C
...	...	...	...	...	...	...	...	...	...	...	...	...
846	847	0	3	Sage, Mr. Douglas Bullen	male	NaN	8	2	CA. 2343	69.5500	NaN	S
849	850	1	1	Goldenberg, Mrs. Samuel L (Edwiga Grabowska)	female	NaN	1	0	17453	89.1042	C92	C
856	857	1	1	Wick, Mrs. George Dennick (Mary Hitchcock)	female	45.0	1	1	36928	164.8667	NaN	S
863	864	0	3	Sage, Miss. Dorothy Edith "Dolly"	female	NaN	8	2	CA. 2343	69.5500	NaN	S
879	880	1	1	Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)	female	56.0	0	1	11767	83.1583	C50	C

116 rows × 12 columns

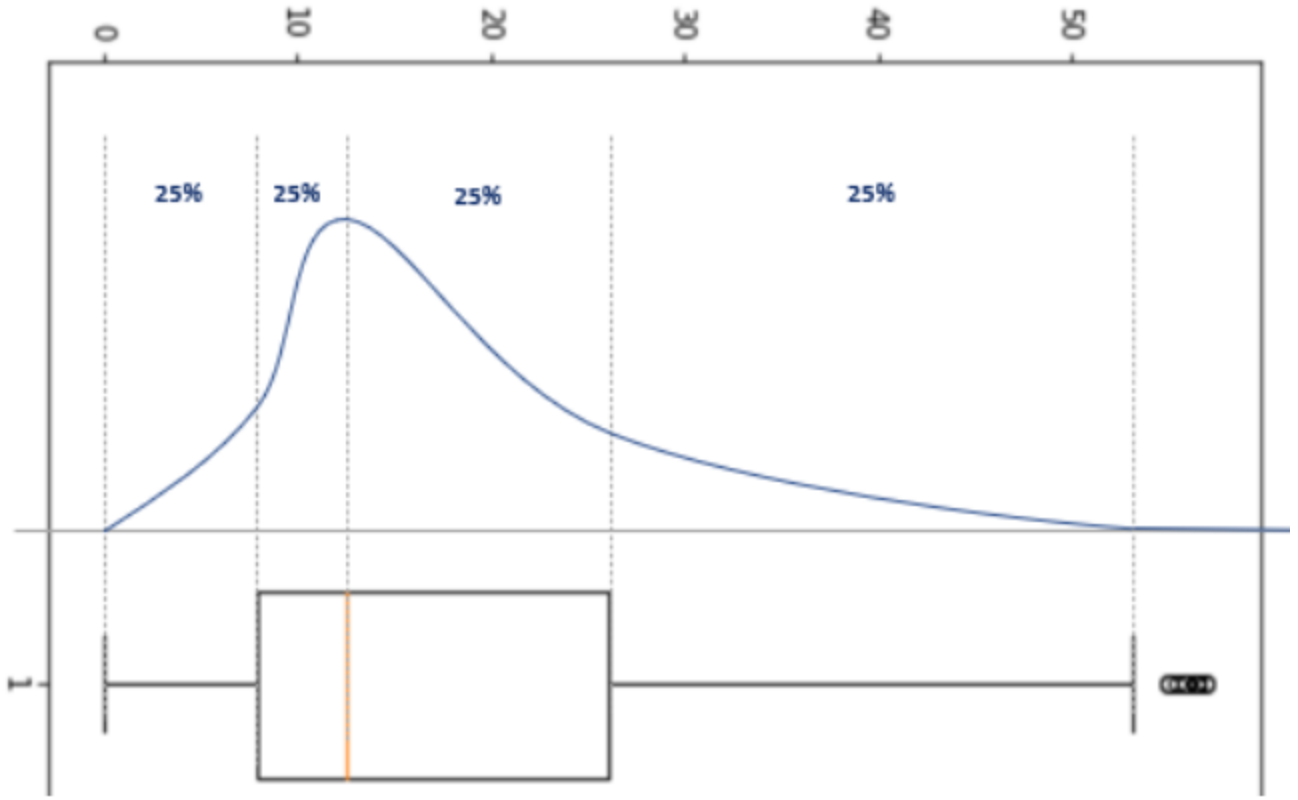
Módulo 14 – Criando visualizações

Também podemos olhar para cada intervalo entre os quartis de nossos valores:

```
1  # Verificando a quantidade de registros em cada intervalo (<Q1, entre Q1 e Q2, entre Q2 e Q3,>Q3)
2  Q2 = base.Fare.describe()['50%']
3  base[(base.Fare > Q2) & (base.Fare < Q3)]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C
10	11	1	3	Sandstrom, Miss. Marguerite Rut	female	4.0	1	1	PP 9549	16.7000	G6	S
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500	C103	S
15	16	1	2	Hewlett, Mrs. (Mary D Kingcome)	female	55.0	0	0	248706	16.0000	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
880	881	1	2	Shelley, Mrs. William (Imanita Parrish Hall)	female	25.0	0	1	230433	26.0000	NaN	S
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	382652	29.1250	NaN	Q
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S

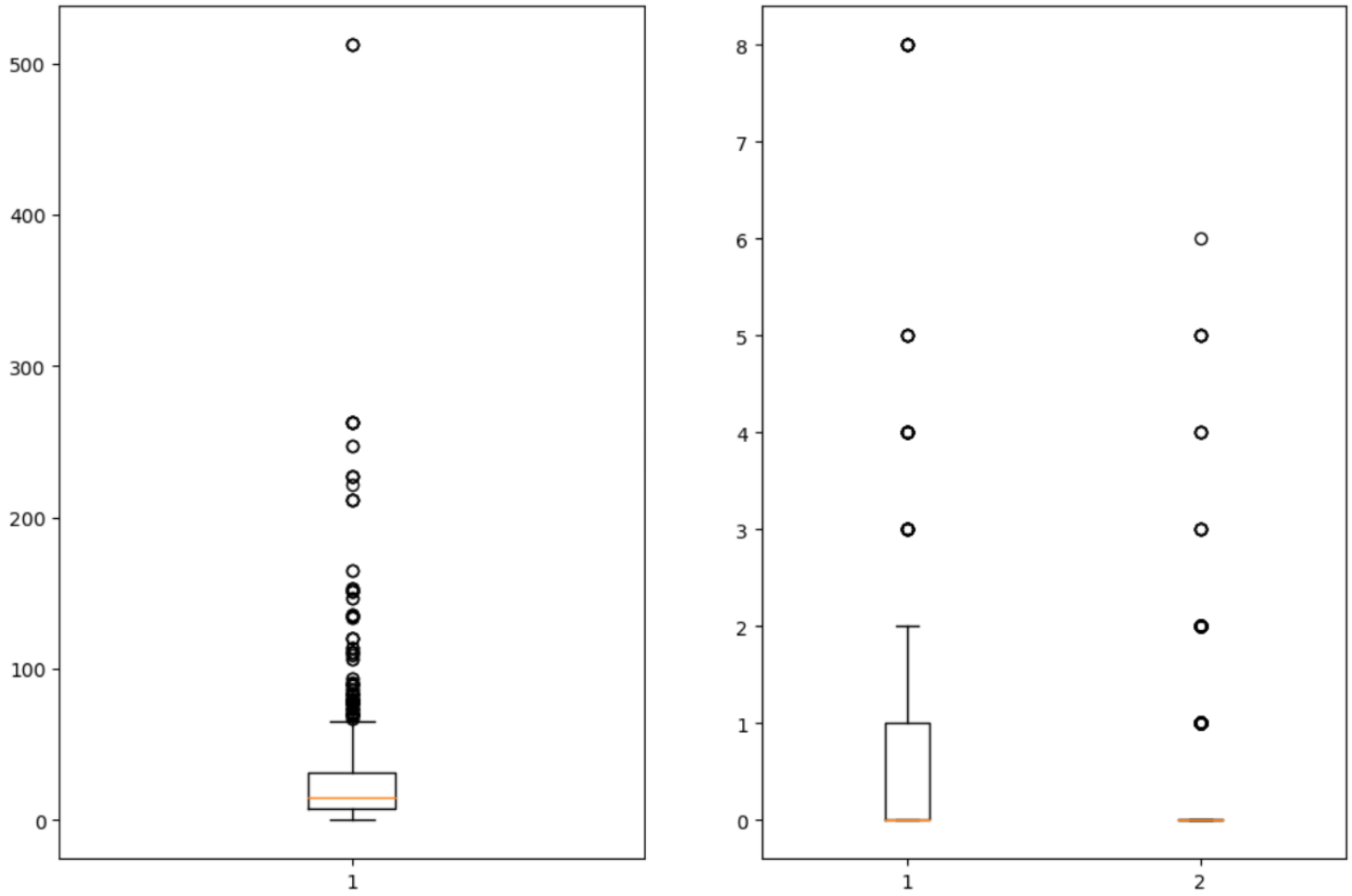
O nome “quartil” vem do fato de estarmos dividindo nossa base em 4 partes. Cada parte contendo 25 % dos dados, como mostrado a seguir. A curva azul indica a densidade dos dados, e veremos como a construir mais adiante.



Módulo 14 – Criando visualizações

Tudo que fizemos pode ser reproduzido para outras colunas numéricas que desejarmos. Observe para as colunas referentes a familiares a bordo e compare com o resumo estatístico fornecido pelo `describe` para cada coluna.

```
# Verificando para as colunas SibSp e Parch
fig, ax = plt.subplots(ncols=2,figsize=(12,8))

ax[0].boxplot(base.Fare)
ax[1].boxplot([base.SibSp,base.Parch])

plt.show()
```



Módulo 14 – Criando visualizações

Como já vimos, o `describe` pode mostrar um resumo estatístico por coluna. Para uma simples contagem de cada valor possível em uma coluna, pode-se utilizar o método `value_counts`. Perceba que, para a coluna *SibSp*, parte significativa das entradas são para os valores 0 e 1, justificando o boxplot visto na página anterior.

```
1  # Verificando o resumo estatístico para essa coluna
2  base.SibSp.describe()
```

```
count    891.000000
mean       0.523008
std        1.102743
min        0.000000
25%        0.000000
50%        0.000000
75%        1.000000
max        8.000000
Name: SibSp, dtype: float64
```

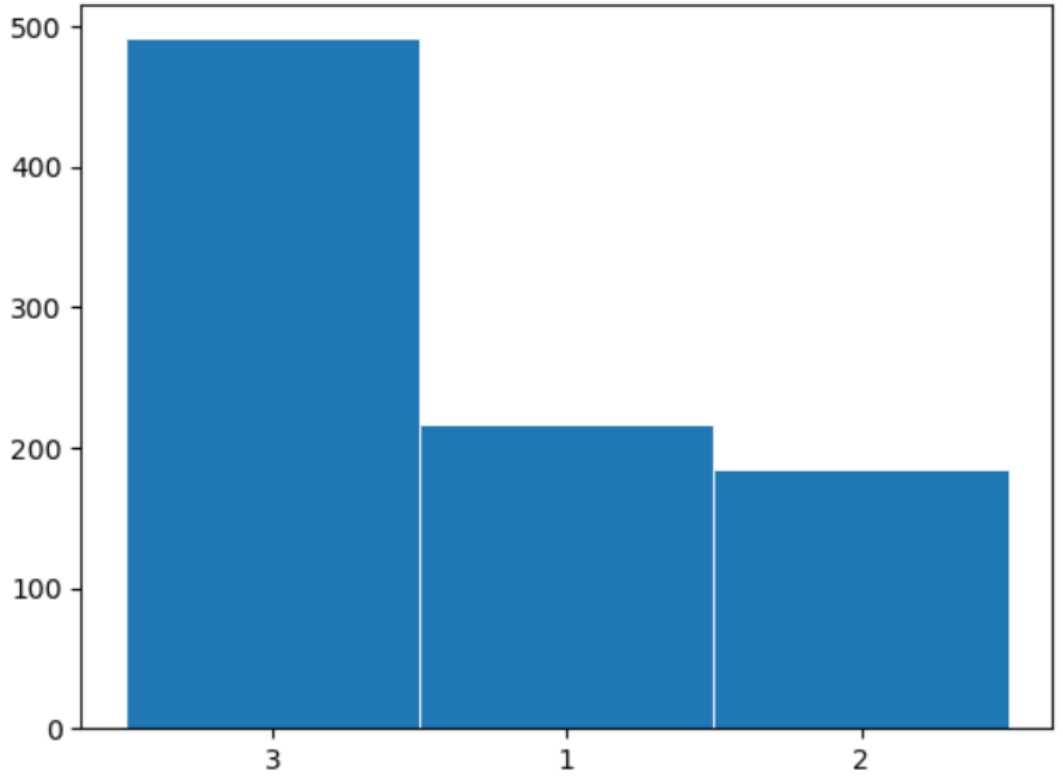
```
1  # Verificando o value_counts()
2  base.SibSp.value_counts()
```

```
0    608
1    209
2     28
4     18
3     16
8      7
5      5
Name: SibSp, dtype: int64
```

Módulo 14 – Criando visualizações

Com o `value_counts`, podemos construir gráficos de barras para as colunas de interesse. Por exemplo, podemos ter uma visualização mostrando qual a classe que tinha mais passageiros.

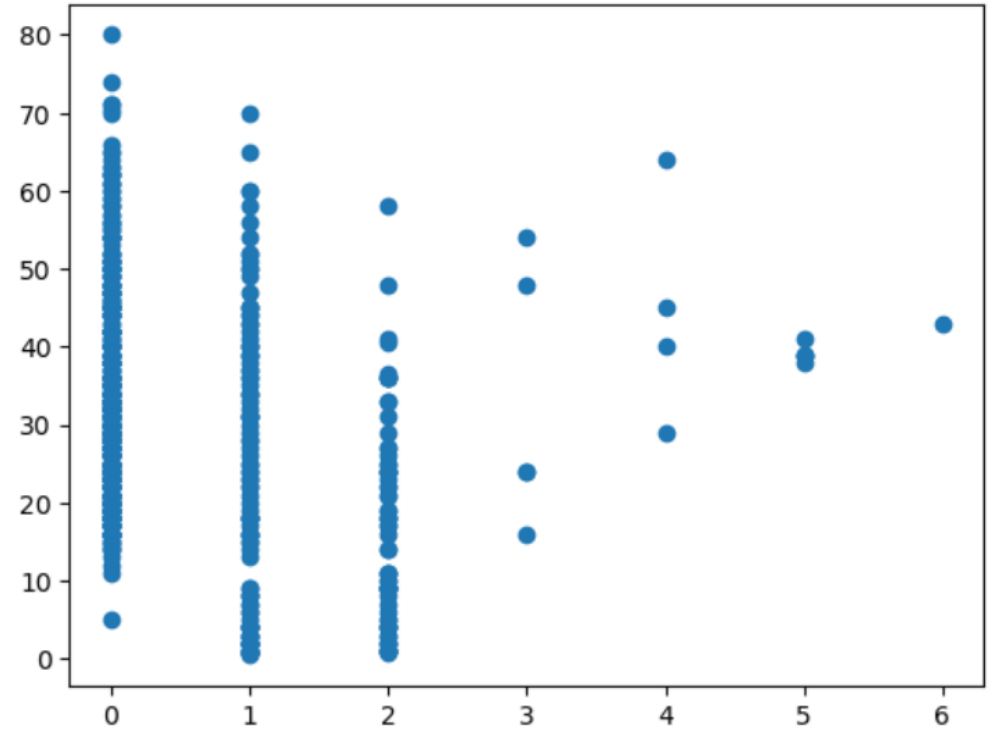
```
# Gerando um gráfico de barras
fig, ax = plt.subplots()

x=base.Pclass.value_counts().index.astype('str')
y=base.Pclass.value_counts().values

ax.bar(x, y, width=1, edgecolor="white", linewidth=0.7)

plt.show()
```



Módulo 14 – Correlação entre colunas

Em Ciência de Dados, uma parte importante da análise é a de relação entre colunas. Um bom gráfico para isso é o de dispersão, *scatter* em inglês. Com ele, podemos ver o número de registros a partir de pontos. Por exemplo, podemos verificar que famílias maiores usualmente possuem alguém com mais idade e são menos frequentes:

```
# Verificando se existe relação entre a coluna Parch e a idade
fig, ax = plt.subplots()

x=base.Parch
y=base.Age

ax.scatter(x, y)

plt.show()
```



Módulo 14 – Correlação entre colunas

Ao invés de fazer esta análise coluna a coluna, podemos utilizar o método `scatter_matrix`, que constrói esta análise para os pares de colunas da base de dados. Na diagonal, onde haveria repetição das colunas, é mostrado um histograma dos valores da coluna em questão.

```
1  # Utilizando o pandas
2  pd.plotting.scatter_matrix(base,figsize=(16,12))
3  plt.show()
```

Módulo 14 – Correlação entre colunas

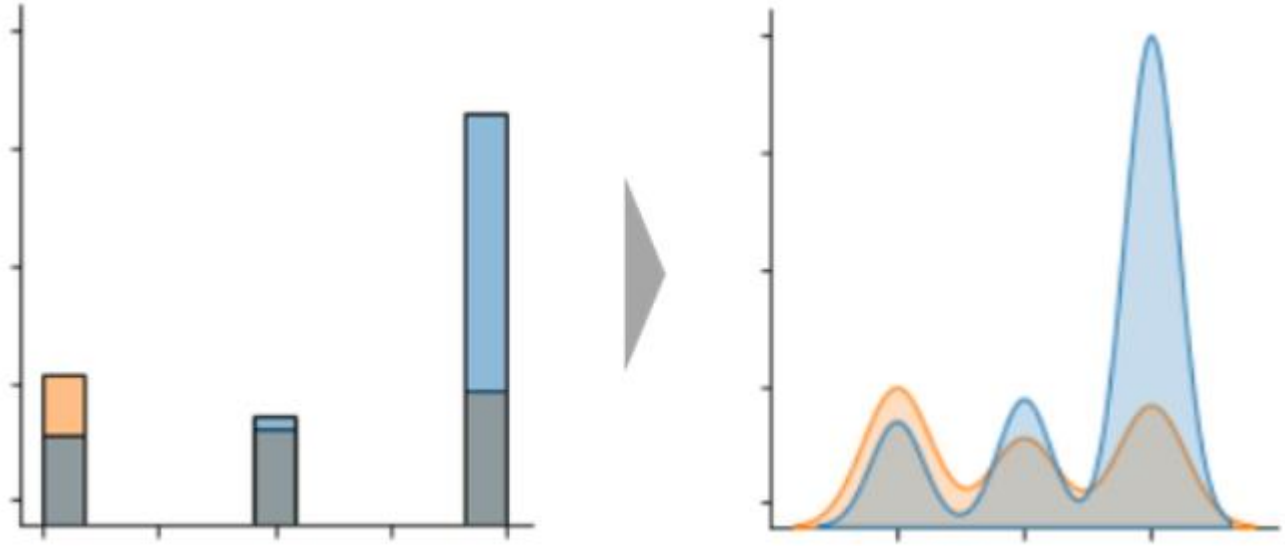
Um dos aspectos mais interessantes desta base de dados é entender quais aspectos foram os mais importantes para a sobrevivência dos passageiros quando houve o acidente. O Seaborn possui o método **`pairplot`**, que também faz uma matriz de gráficos de dispersão, mas permite diferenciar os pontos com cores a partir de uma coluna. Assim, podemos passar a coluna *Survived*. Veja como há mais coloração laranja (sobrevivente) em tarifas maiores e na primeira classe.

```
1  # Utilizando o seaborn
2  import seaborn as sns
3  sns.pairplot(base,hue="Survived")# ,diag_kind='hist')
4  plt.show()
```

Módulo 14 – Correlação entre colunas

Observe, na página anterior, que a diagonal não apresenta mais um histograma* mas, sim, um gráfico KDE. KDE vem do inglês *Kernel Density Estimation,* e mede a chance de uma variável aleatória assumir um determinado valor. De forma simplificada, é como se fosse um histograma suavizado. Observe na figura abaixo um histograma e o KDE correspondente lado a lado.

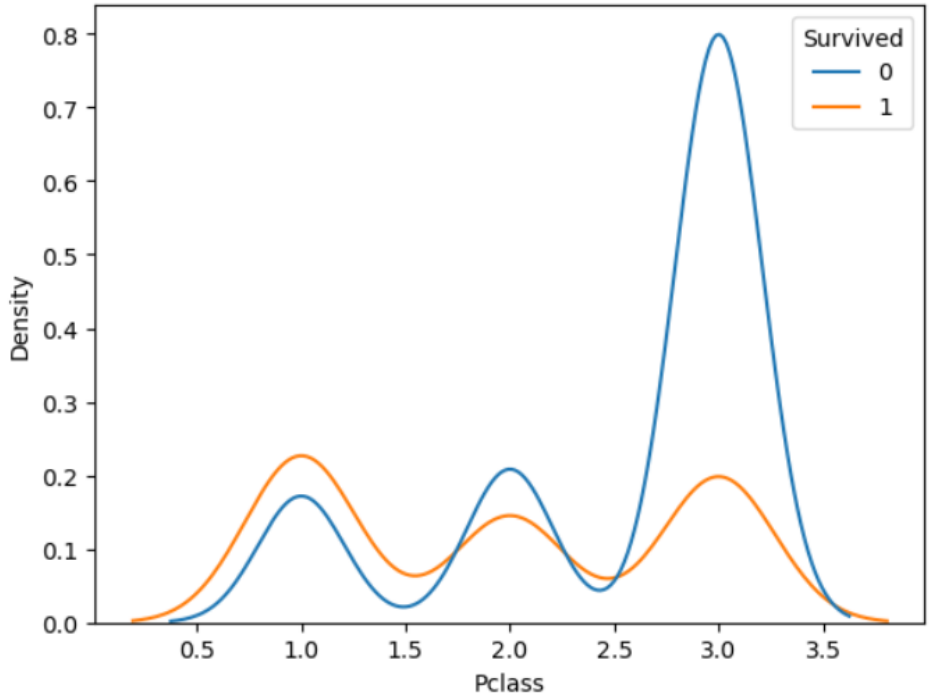
*Para exibir um histograma, passe o parâmetro `diag_kind='hist'`.



Módulo 14 – Correlação entre colunas

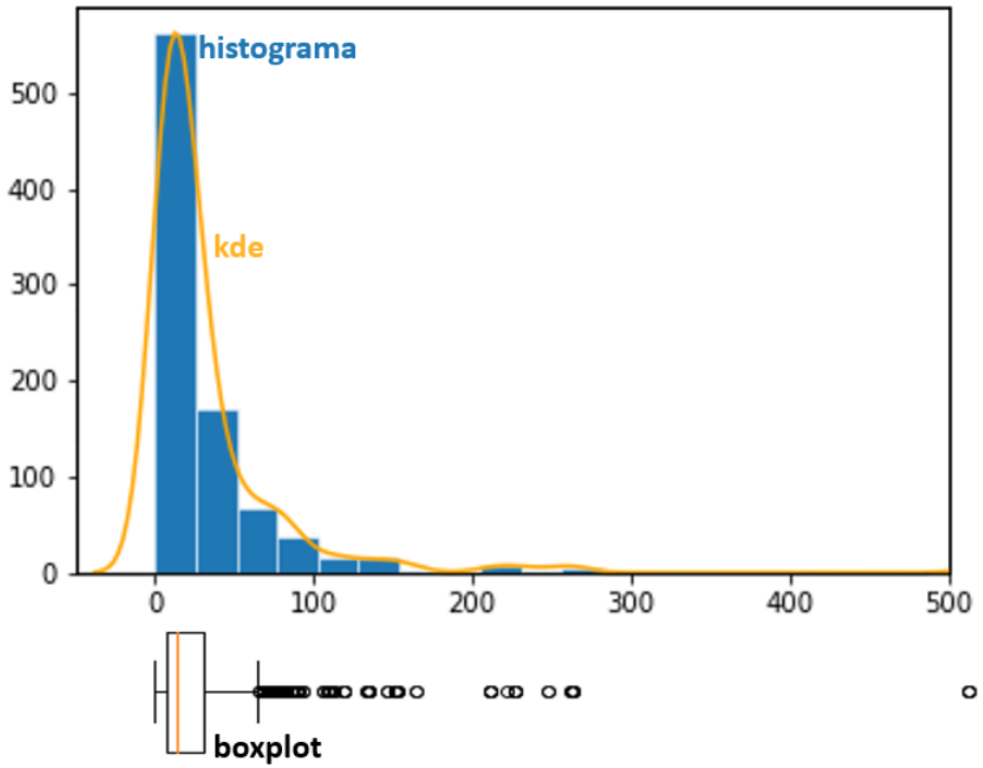
Podemos construir um gráfico KDE para cada coluna com o método `kdeplot`. Por exemplo, podemos ver como é o gráfico KDE da coluna correspondente à classe de cada passageiro, verificando novamente a diferença na sobrevivência ao acidente a depender da classe.

```
1  # Fazendo o kde para o Pclass
2  sns.kdeplot(base.Pclass,hue=base.Survived)
3  plt.show()
```



Módulo 14 – Correlação entre colunas

O KDE é mais uma visualização que complementa o visto nos histogramas e nos boxplots. Por exemplo, para a coluna *Fare*, vemos que o KDE acompanha o histograma, suavizando-o, e vemos que onde há baixos valores eram os outliers do boxplot.



Módulo 14 – Correlação entre colunas

Em uma análise exploratória, é sempre importante verificar se as colunas possuem algum tipo de correlação. Para isso, podemos utilizar o método `corr` do Pandas.

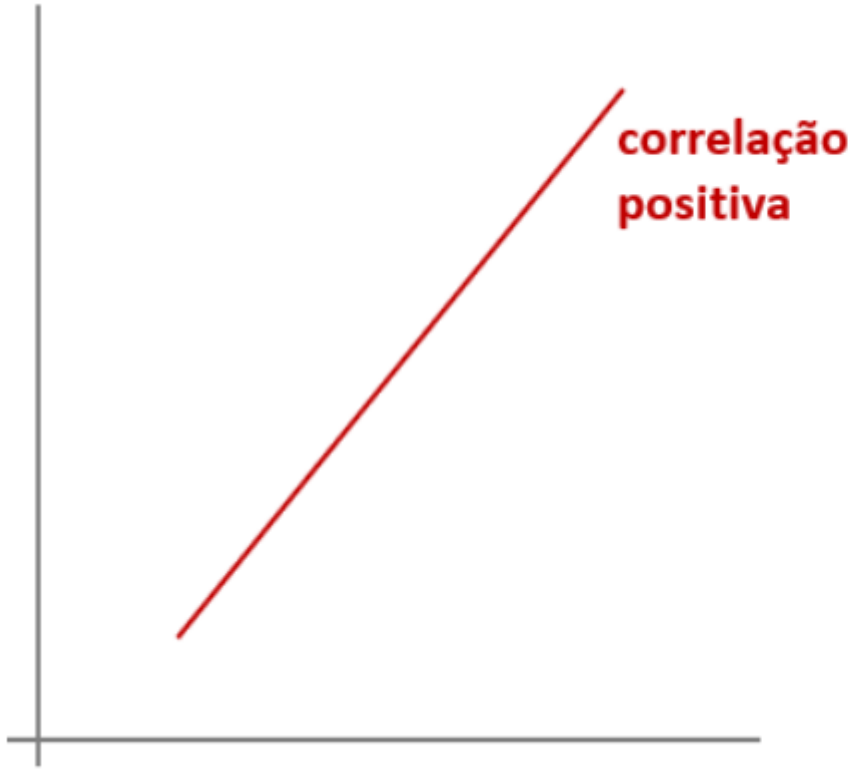
```
1  # Verificando a correlação entre os dados
2  base.corr()
```

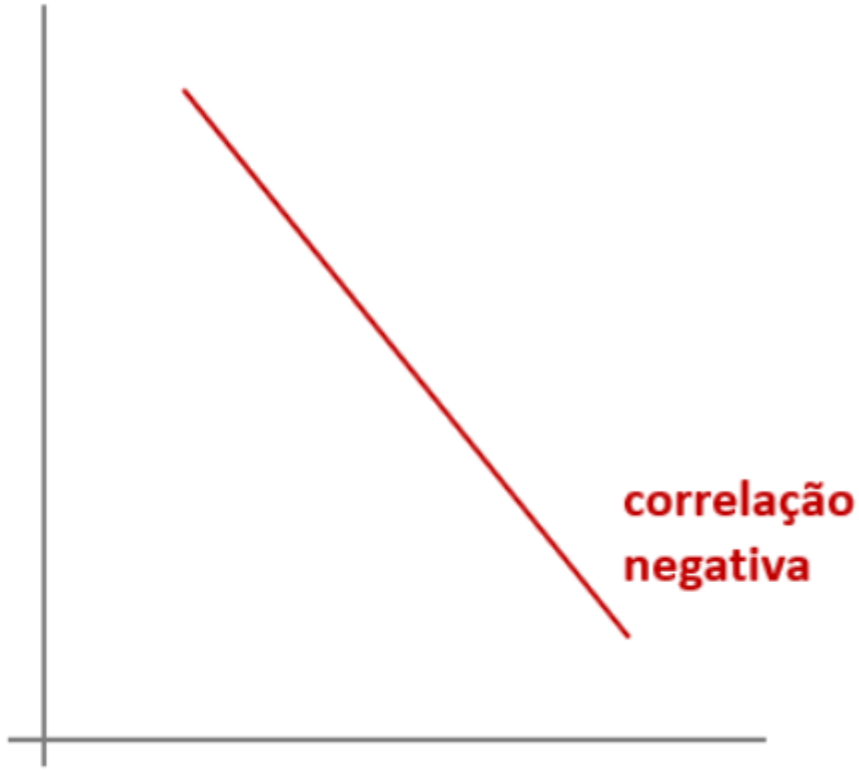
	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
PassengerId	1.000000	-0.005007	-0.035144	0.036847	-0.057527	-0.001652	0.012658
Survived	-0.005007	1.000000	-0.338481	-0.077221	-0.035322	0.081629	0.257307
Pclass	-0.035144	-0.338481	1.000000	-0.369226	0.083081	0.018443	-0.549500
Age	0.036847	-0.077221	-0.369226	1.000000	-0.308247	-0.189119	0.096067
SibSp	-0.057527	-0.035322	0.083081	-0.308247	1.000000	0.414838	0.159651
Parch	-0.001652	0.081629	0.018443	-0.189119	0.414838	1.000000	0.216225
Fare	0.012658	0.257307	-0.549500	0.096067	0.159651	0.216225	1.000000

Observe que há valores positivos e negativos. Estes valores podem variar de -1 a +1. Um valor positivo indica correlação positiva (quando um valor aumenta, o outro também aumenta). E um valor negativo indica o oposto (quando um valor aumenta, o outro diminui). Quanto mais se aproximam de -1 ou +1 maior a correlação, seja negativa ou positiva.

Módulo 14 – Correlação entre colunas

Graficamente, as correlações seriam como a inclinação de uma reta. O valor da inclinação indica o quanto de correlação há entre as variáveis.

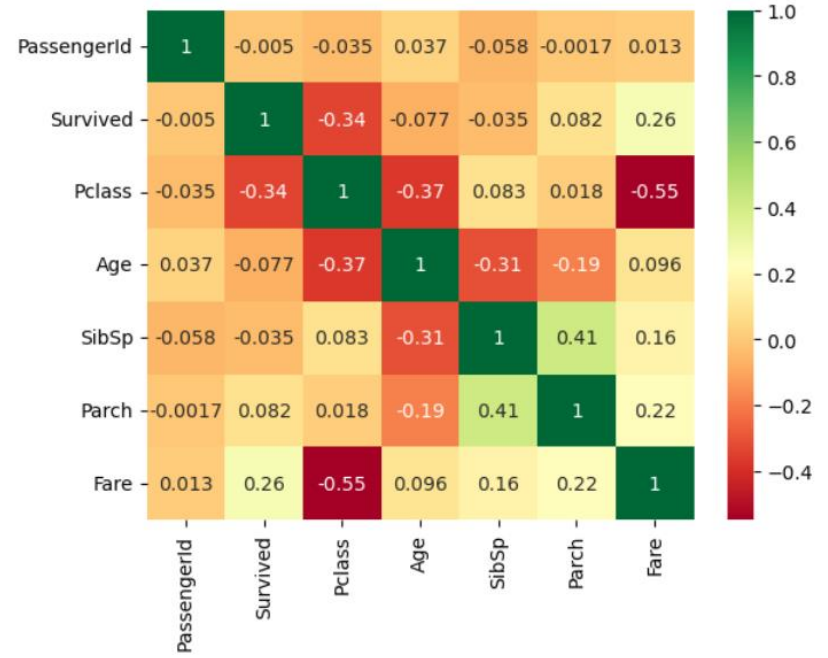




Módulo 14 – Correlação entre colunas

Podemos passar o dataframe de correlação para o método heatmap do Seaborn para facilitar a visualização com cores. As cores indicam a correlação, se positiva ou negativa, e a intensidade mostra a intensidade da correlação. Por exemplo, veja como há uma correlação negativa entre *Pclass* e *Fare*. Faz sentido, pois as passagens mais caras correspondem à classe 1, menor número entre as classes.

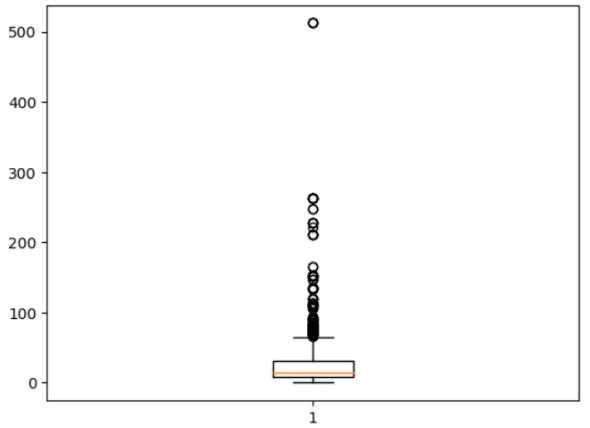
```
1  # Tornando visual
2  # cmap: https://matplotlib.org/stable/tutorials/colors/colormaps.html
3  sns.heatmap(base.corr(),annot=True,cmap='RdYlGn')
4  plt.show()
```



Módulo 14 – Tratando valores ausentes e outliers

Agora que já vimos o básico de visualizações, vamos abordar outro assunto muito importante de análise de dados, o de tratamento de valores vazios e outliers.

```
# Verificando para a coluna Fare
fig, ax = plt.subplots()

ax.boxplot(base.Fare)

plt.show()
```



Por exemplo, já vimos que a coluna *Fare* possui outliers. Mas será que são valores que fazem sentido? Procurando no dataframe, vemos que são pessoas da primeira classe, que realmente possui valores maiores, e com nome. Logo, parecem ser registros autênticos e não algum erro de preenchimento da base. Inclusive, se buscarmos na internet o nome dessas pessoas, achamos suas informações e das cabines de luxo.

```
# Verificando para a coluna Fare
base[base.Fare > 300]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
258	259	1	1	Ward, Miss. Anna	female	35.0	0	0	PC 17755	512.3292	NaN	C
679	680	1	1	Cardeza, Mr. Thomas Drake Martinez	male	36.0	0	1	PC 17755	512.3292	B51 B53 B55	C
737	738	1	1	Lesurer, Mr. Gustave J	male	35.0	0	0	PC 17755	512.3292	B101	C

Módulo 14 – Tratando valores ausentes e outliers

Seguindo a mesma lógica, podemos ver se há valores muito baixos de *Fare*. Aqui, temos algo estranho, já que há valores 0 mesmo para primeira classe.

```
1  base[base.Fare < 1]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
179	180	0	3	Leonard, Mr. Lionel	male	36.0	0	0	LINE	0.0	NaN	S
263	264	0	1	Harrison, Mr. William	male	40.0	0	0	112059	0.0	B94	S
271	272	1	3	Tornquist, Mr. William Henry	male	25.0	0	0	LINE	0.0	NaN	S
277	278	0	2	Parkes, Mr. Francis "Frank"	male	NaN	0	0	239853	0.0	NaN	S
302	303	0	3	Johnson, Mr. William Cahoone Jr	male	19.0	0	0	LINE	0.0	NaN	S
413	414	0	2	Cunningham, Mr. Alfred Fleming	male	NaN	0	0	239853	0.0	NaN	S
466	467	0	2	Campbell, Mr. William	male	NaN	0	0	239853	0.0	NaN	S
481	482	0	2	Frost, Mr. Anthony Wood "Archie"	male	NaN	0	0	239854	0.0	NaN	S
597	598	0	3	Johnson, Mr. Alfred	male	49.0	0	0	LINE	0.0	NaN	S
633	634	0	1	Parr, Mr. William Henry Marsh	male	NaN	0	0	112052	0.0	NaN	S
674	675	0	2	Watson, Mr. Ennis Hastings	male	NaN	0	0	239856	0.0	NaN	S
732	733	0	2	Knight, Mr. Robert J	male	NaN	0	0	239855	0.0	NaN	S
806	807	0	1	Andrews, Mr. Thomas Jr	male	39.0	0	0	112050	0.0	A36	S
815	816	0	1	Fry, Mr. Richard	male	NaN	0	0	112058	0.0	B102	S
822	823	0	1	Reuchlin, Jonkheer. John George	male	38.0	0	0	19972	0.0	NaN	S

Módulo 14 – Tratando valores ausentes e outliers

Provavelmente estes valores não são corretos, talvez na realidade não se tinha essa informação e foi colocado zero no lugar. Logo, podemos substituir esses valores por algo mais razoável como, por exemplo, a média:

```
# Calculando a média
mean_Fare = base.Fare.mean()
```

```
# Substituindo esses valores
base.loc[base.Fare < 1,'Fare'] = mean_Fare
```

Após a substituição, não mais registros desses valores baixos:

```
base[base.Fare < 1]
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked

Módulo 14 – Tratando valores ausentes e outliers

Anteriormente, vimos que há valores vazios/nulos na coluna *Age*:

```
1  # Verificando a base
2  base[base.Age.isnull()]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
17	18	1	2	Williams, Mr. Charles Eugene	male	NaN	0	0	244373	13.0000	NaN	S
19	20	1	3	Masselmani, Mrs. Fatima	female	NaN	0	0	2649	7.2250	NaN	C
26	27	0	3	Emir, Mr. Farred Chehab	male	NaN	0	0	2631	7.2250	NaN	C
28	29	1	3	O'Dwyer, Miss. Ellen "Nellie"	female	NaN	0	0	330959	7.8792	NaN	Q
...	...	...	...	...	...	...	...	...	...	...	...	...
859	860	0	3	Razi, Mr. Raihed	male	NaN	0	0	2629	7.2292	NaN	C
863	864	0	3	Sage, Miss. Dorothy Edith "Dolly"	female	NaN	8	2	CA. 2343	69.5500	NaN	S
868	869	0	3	van Melkebeke, Mr. Philemon	male	NaN	0	0	345777	9.5000	NaN	S
878	879	0	3	Laleff, Mr. Kristo	male	NaN	0	0	349217	7.8958	NaN	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S

177 rows × 12 columns

Podemos verificar se há mais valores ausentes a depender do status de sobrevivência:

```
1  # Entendendo melhor os valores
2  base.loc[base.Age.isnull(),'Survived'].value_counts()
```

```
0    125
1     52
Name: Survived, dtype: int64
```

Veja como há mais valores ausentes entre os que não sobreviveram.

Assim como anteriormente, vamos substituir estes valores ausentes pela média da coluna:

```
1  # Calculando a média das idades
2  base.Age.mean()
```

```
29.69911764705882
```

```
1  # Substituindo os valores nulos por essa média
2  base.loc[base.Age.isnull(),'Age'] = base.Age.mean()
```

```
1  # Verificando novamente a base
2  base[base.Age.isnull()]
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked

É importante ressaltar que pode-se escolher outro critério de substituição que não seja pela média.

Módulo 14 – Pandas Profiling

Tudo que vimos até agora é o básico que deve ser feito em todo início de uma análise de base de dados. E é muito importante ter esses fundamentos. Mas, se é algo que sempre se faz, é natural que tenham desenvolvido uma biblioteca que automatiza esta análise toda. É a biblioteca *ydata-profiling*, mais conhecida como *Pandas Profiling*, já que ela mudou de nome há pouco tempo. Tal biblioteca gera um relatório com uma análise exploratória básica de toda a base de dados.

Para instalar, basta usar `pip install -U ydata-profiling`

Após instalar, importe a classe `ProfileReport`:

```
# Importando o pandas e o pandas profiling
import pandas as pd
from pandas_profiling import ProfileReport
```

Então, importe a base de dados para o Pandas e a passe para o `ProfileReport`, com um título de sua escolha para o relatório:

```
# Importando a base de dados
base = pd.read_csv('train.csv')
```

```
# Gerando o relatório
profile = ProfileReport(base, title="Pandas Profiling Report")
```

Módulo 14 – Pandas Profiling

Agora, basta chamar a variável criada e um relatório será apresentado dentro de seu notebook:

```
# Visualizar o relatório
profile
```

```
Summarize dataset: 100% 47/47 [00:03<00:00, 8.19it/s, Completed]
Generate report structure: 100% 1/1 [00:02<00:00, 2.47s/it]
Render HTML: 100% 1/1 [00:01<00:00, 1.32s/it]
```

Pandas Profiling Report — Overview | Variables | Interactions | Correlations | Missing values | Sample

Overview

Overview | Alerts 16 | Reproduction

Dataset statistics

Number of variables	12
Number of observations	891
Missing cells	866
Missing cells (%)	8.1%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	83.7 KiB
Average record size in memory	96.1 B

Variable types

Numeric	5
Categorical	7

Módulo 14 – Pandas Profiling

O relatório pode ser extenso e seria melhor abri-lo em um arquivo a parte. A estrutura do relatório é em HTML, de forma que podemos usar o método `to_file` para exportar para um arquivo HTML que pode ser aberto em um navegador:

```
1  profile.to_file("analise.html")
```

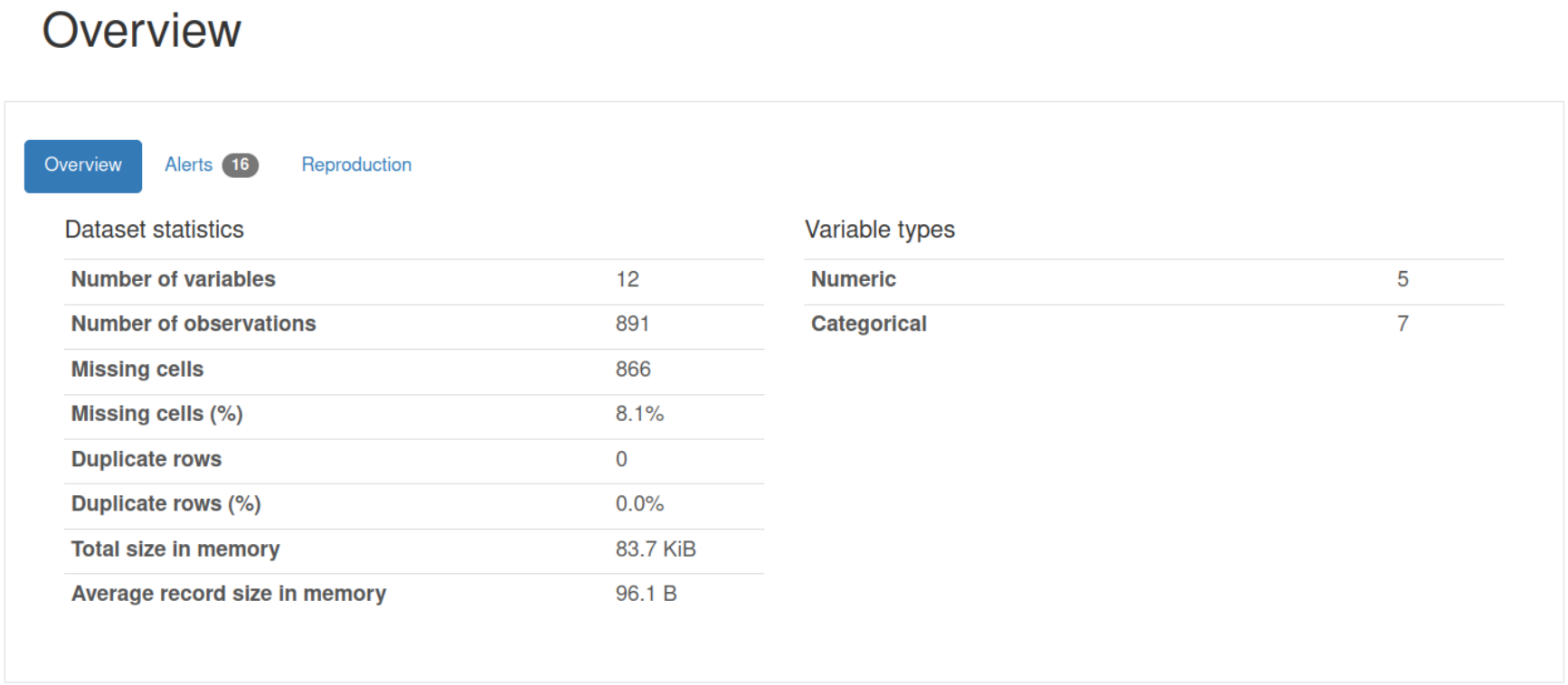
```
Export report to file: 100% |████████████████████| 1/1 [00:00<00:00, 22.63it/s]
```

Após o final da exportação, abra o arquivo em seu navegador de preferência.

Módulo 14 – Pandas Profiling

O relatório gerado tem algumas seções. Recomendamos fortemente que veja todas com detalhes. Aqui iremos apresentar o básico de cada seção, destacando os conceitos que vimos passo a passo no decorrer deste módulo.

A primeira seção é um resumo, mostrando o número de colunas, de linhas, de valores ausentes e de duplicatas. Também mostra o tipo das colunas presentes.



A próxima seção mostra, para cada coluna, um breve descrição estatística. Com valores ausentes e, no caso de colunas numéricas, valores mínimo, máximo, zeros e negativos, e média. Apresenta, também, um histograma. Abaixo, a coluna *PassengerId*.

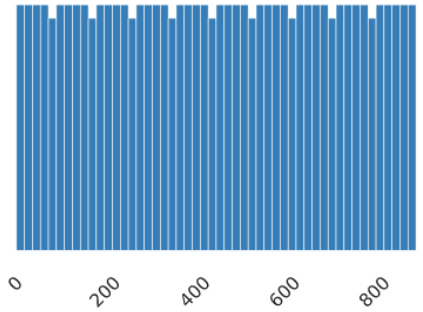
Variables

Select Columns

PassengerId
Real number (ℝ)

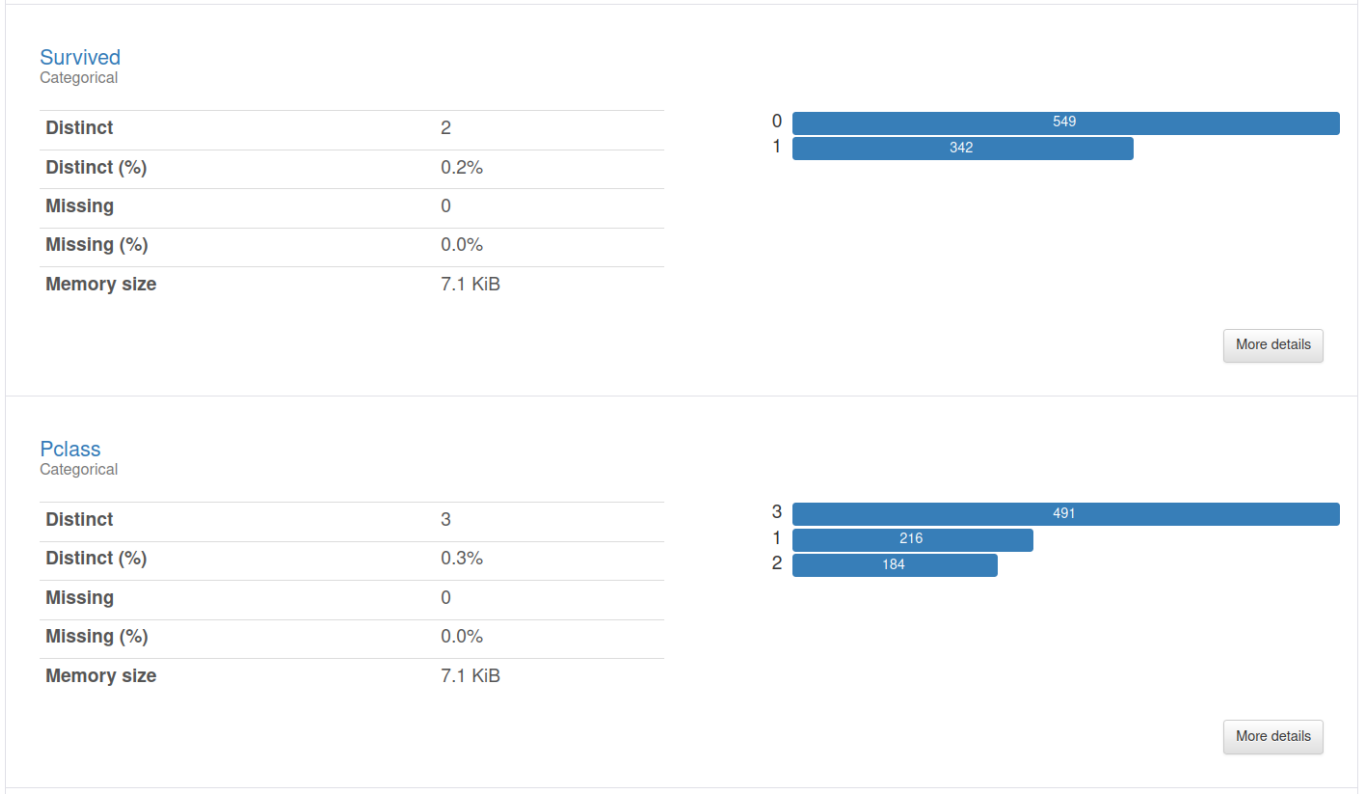
UNIFORM UNIQUE

Distinct	891	**Minimum**	1
Distinct (%)	100.0%	**Maximum**	891
Missing	0	**Zeros**	0
Missing (%)	0.0%	**Zeros (%)**	0.0%
Infinite	0	**Negative**	0
Infinite (%)	0.0%	**Negative (%)**	0.0%
Mean	446	**Memory size**	7.1 KiB



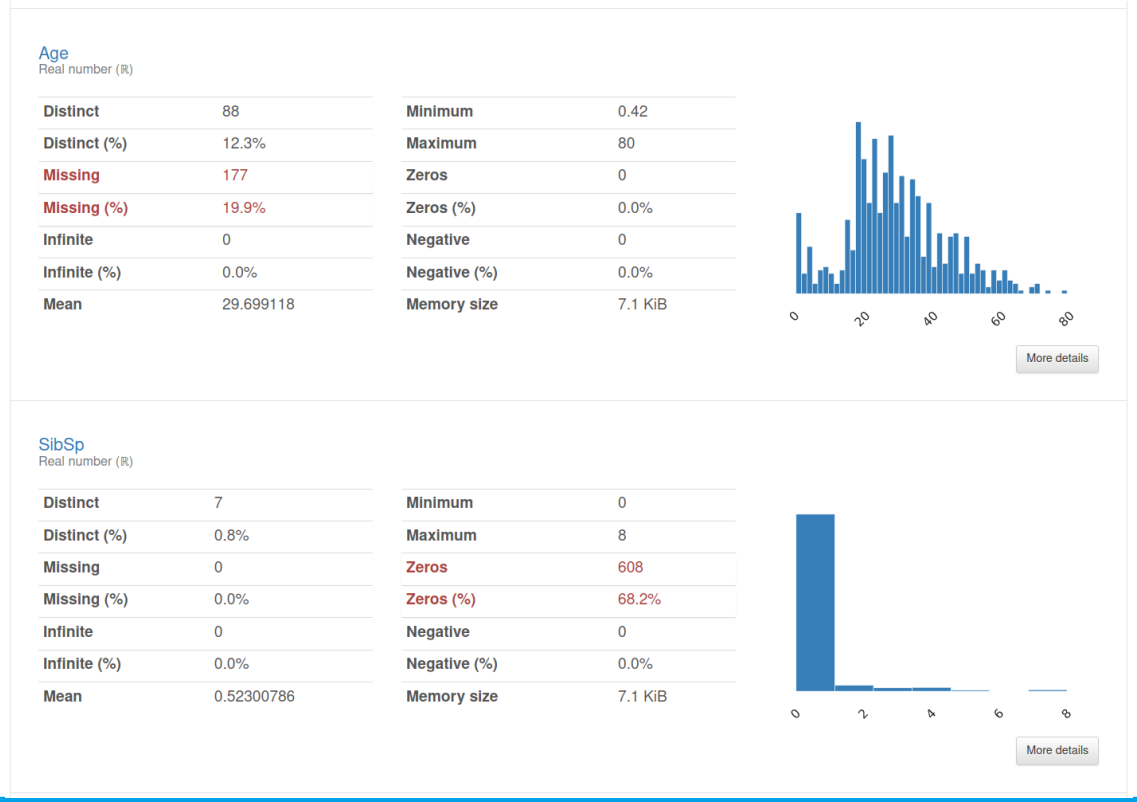
More details

Observe que a biblioteca reconhece quando as colunas são categóricas, ou seja, quando possuem poucos valores que, na realidade, descrevem categorias. Abaixo, vemos as informações das colunas *Survived* e *Pclass*. Os gráficos já mostram claramente que temos mais mortos que sobreviventes e muito mais passageiros da classe 3 que das demais.



Módulo 14 – Pandas Profiling

Abaixo, temos as colunas *Age* e *SibSp*. Observe que a biblioteca destaca estatísticas que podem ser importantes para o analista. Na coluna *Age*, destacou a quantidade de valores ausentes e, na coluna *SibSp*, a quantidade de valores erro. Desta forma, facilita o trabalho do analista e já incentiva o mesmo a pensar em estratégias para lidar com estes casos, como fizemos no decorrer do módulo.



Age
Real number (ℝ)

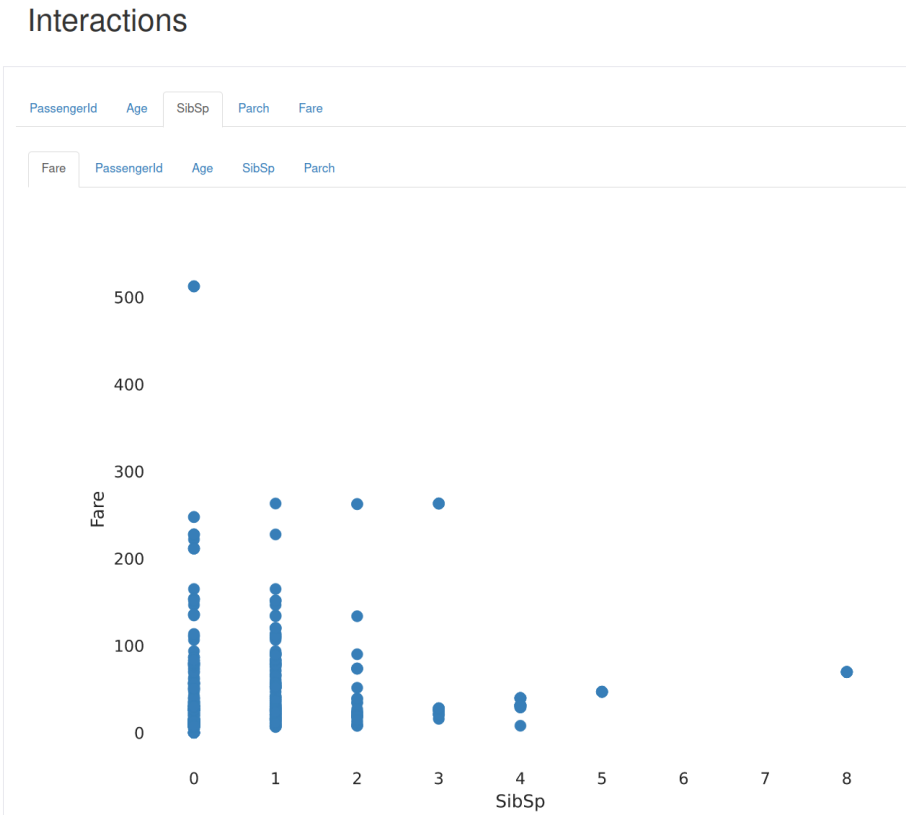
Distinct	88	**Minimum**	0.42
Distinct (%)	12.3%	**Maximum**	80
Missing	177	**Zeros**	0
Missing (%)	19.9%	**Zeros (%)**	0.0%
Infinite	0	**Negative**	0
Infinite (%)	0.0%	**Negative (%)**	0.0%
Mean	29.699118	**Memory size**	7.1 KiB

SibSp
Real number (ℝ)

Distinct	7	**Minimum**	0
Distinct (%)	0.8%	**Maximum**	8
Missing	0	**Zeros**	608
Missing (%)	0.0%	**Zeros (%)**	68.2%
Infinite	0	**Negative**	0
Infinite (%)	0.0%	**Negative (%)**	0.0%
Mean	0.52300786	**Memory size**	7.1 KiB

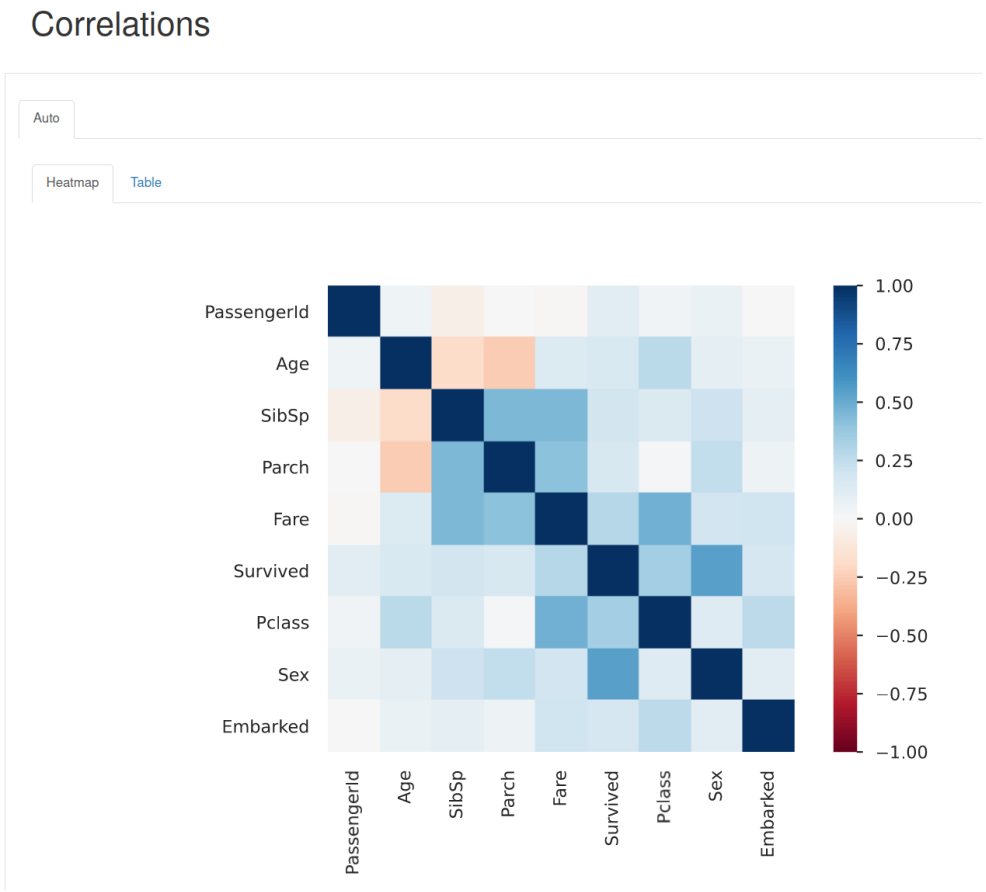
Módulo 14 – Pandas Profiling

No decorrer do módulo, discutimos sobre correlações e gráficos de dispersão. Na seção *Interactions*, a biblioteca gera gráficos de dispersão para cada par de coluna numérica. Abaixo, um dos gráficos que construímos durante o módulo, construído automaticamente pela biblioteca.



Módulo 14 – Pandas Profiling

Da mesma forma, constrói automaticamente mapas de calor:



Módulo 14 – Finalizando a análise e apresentado os resultados

O relatório do Pandas Profiling pode ser personalizado de diversas formas, basta ver a documentação. Pode ser uma boa primeira apresentação aos clientes finais de sua análise.

Também é importante ir documentando sua análise no Jupyter Notebook, usando as células de texto Markdown. Assim, pode-se gerar um relatório bem explicado aos interessados. Neste caso, para apresentar a outros, pode-se remover células de código intermediário, deixando só as importantes para o contexto do cliente final. O Jupyter Notebook permite exportar para diversos formatos como, por exemplo, PDF. Dessa forma, o cliente final pode abrir sem ter a necessidade de precisar instalar o programa.

Tudo que foi visto neste módulo servirá de base aos demais, já que toda nova base deve ser explorada inicialmente, antes de se realizar estudos mais detalhados como aprendizado de máquinas. Assim, treine bem o conteúdo deste módulo! Bons estudos!

MÓDULO 15

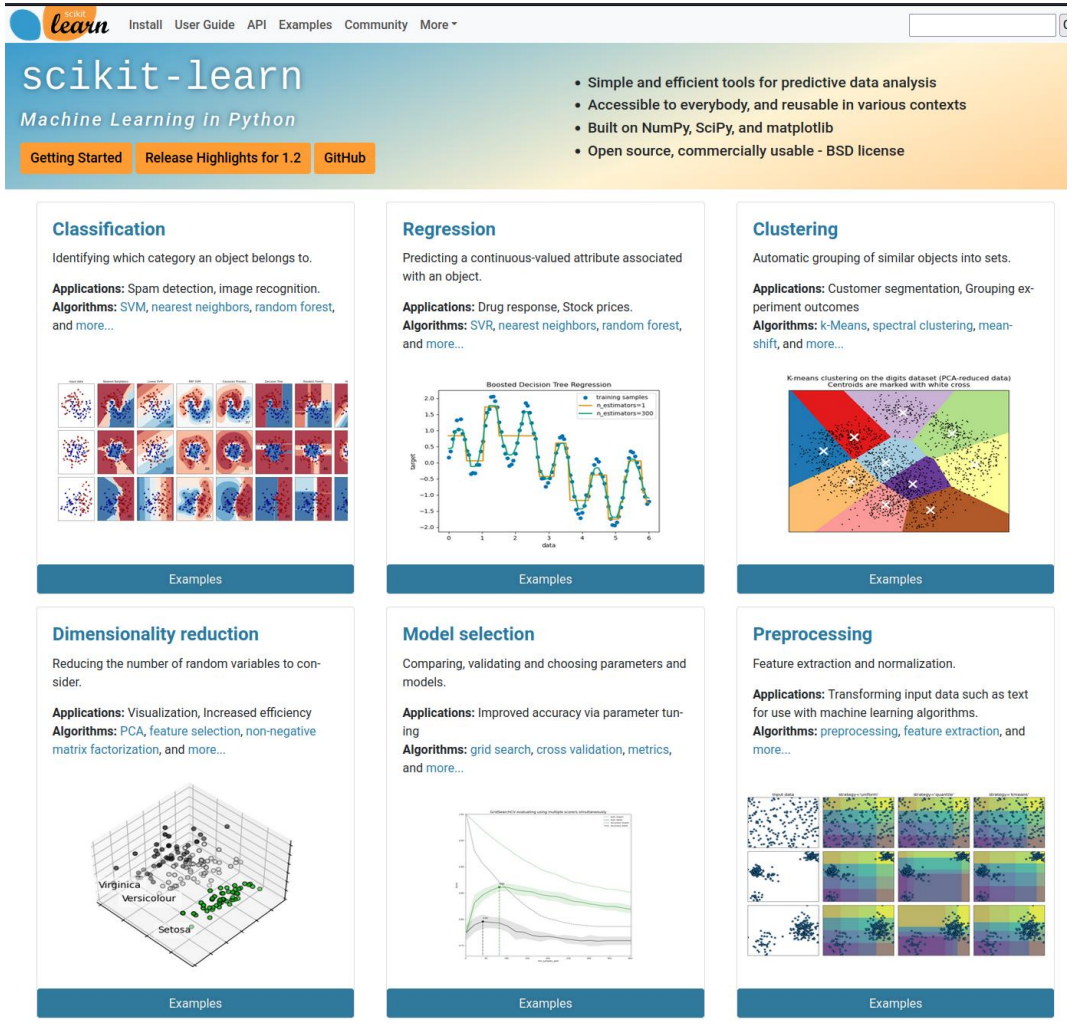
O Scikit-Learn

Módulo 15 – Conhecendo o Scikit-Learn

O Scikit-Learn é uma biblioteca open-source essencial para cientistas de dados que buscam implementar e aprimorar modelos de aprendizado de máquina.

Com uma ampla gama de algoritmos e ferramentas de pré-processamento, a biblioteca facilita a criação e a avaliação de modelos eficientes e robustos. Resumindo:

- Biblioteca de aprendizado de máquina em Python
- Ferramentas eficientes e fáceis de usar
- Aplicações em classificação, regressão, clusterização e redução de dimensionalidade
- Baseada em NumPy, SciPy e matplotlib

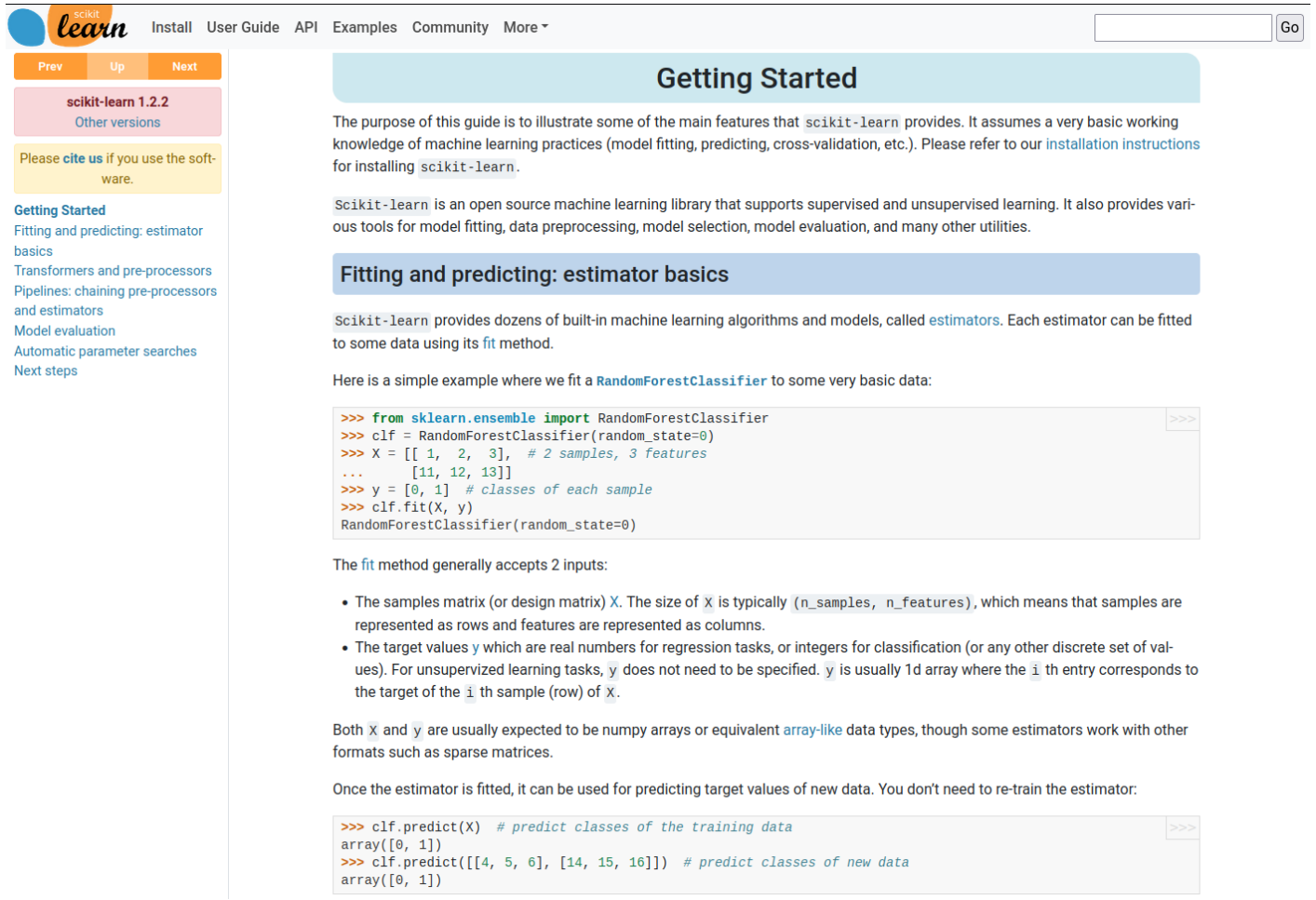


Módulo 15 – Conhecendo o Scikit-Learn

A documentação do Scikit-Learn é notável por sua abrangência e clareza, fornecendo exemplos práticos, explicações detalhadas e orientações passo a passo. Isso facilita o aprendizado e a aplicação efetiva da biblioteca, mesmo para iniciantes na área de Ciência de Dados e aprendizado de máquina.

Em nossos vídeos, sempre mostramos como encontrar as informações na documentação.

Lembrando, se tiver dificuldade em inglês, clique com o botão direito do mouse na página e selecione a opção de tradução do navegador.



Módulo 15 – A base de dados Íris

No módulo 12, construímos um modelo de classificação manualmente, usando o dataset íris.

Recordando, naquele módulo retiramos uma das classes e mostramos que era possível separar as duas restantes a partir de uma reta.

No código ao lado, carregamos o dataset, removemos a classe 2, e mostramos as 5 primeiras linhas de nossa base.

```python
# Importando o pandas
import pandas as pd
```

```python
# Importando o dataset iris
from sklearn.datasets import load_iris
data = load_iris()
```

```python
# Transformando em um dataframe e usando somente target 0 e 1
iris = pd.DataFrame(data.data)
iris.columns = data.feature_names
iris['target'] = data.target
iris = iris[iris.target != 2]
iris.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Módulo 15 – Entendendo o Perceptron

No citado módulo, após fazermos a separação com uma reta manualmente, vimos que há no Scikit-Learn o algoritmo Perceptron. Este algoritmo ajusta uma reta e permite classificar com base se o valor resultante está acima ou abaixo do limite imposto por esta reta.

Lembrando, no contexto do Perceptron, os símbolos *x*, *w* e *y* representam:

•x: vetor de entrada, composto pelos atributos das amostras de dados (ex: $x = [x_1, x_2, ..., x_n]$)

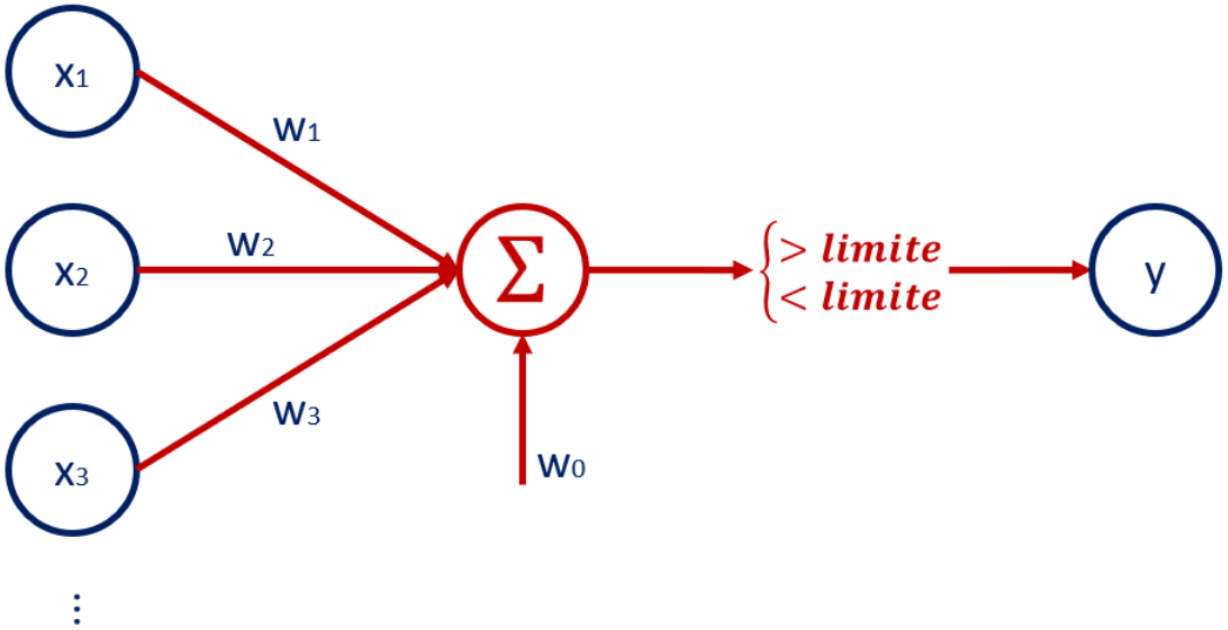
•w: vetor de pesos, que são ajustados durante o treinamento do modelo (ex: $w = [w_0, w_1, w_2, ..., w_n]$)

•y: variável alvo, representando as classes (labels) das amostras de dados

A função linear do Perceptron é definida como:

$$y = w_i \cdot x + w_0$$

onde "·" denota um produto escalar.



Módulo 15 – Entendendo o Perceptron

Vamos reproduzir rapidamente o que vimos no módulo 12. Abaixo, separamos duas colunas de nossa base de dados, que servirão de entrada para nosso modelo, e a coluna de classes:

```
1  # Importando o perceptron do scikit-learn
2  from sklearn.linear_model import Perceptron
```

```
1  iris.head(2)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0

```
1  X = iris[['petal length (cm)','petal width (cm)']]
2  y = iris.target
```

Módulo 15 – Entendendo o Perceptron

Sem fazer a separação entre treino e teste (faremos mais adiante), vemos que nosso modelo é capaz de separar por completo as duas classes, já que resultou em um *score* de valor 1:

```
clf = Perceptron()
```

```
# Fazendo o fit do modelo
clf.fit(X,y)
```

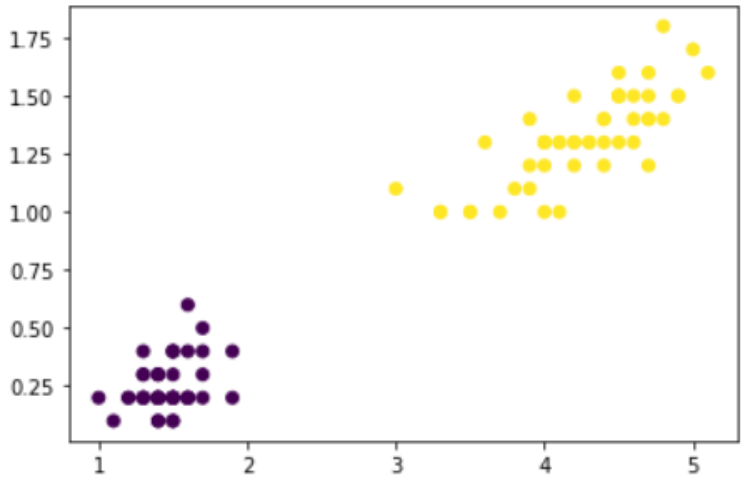
```
Perceptron()
```

```
# Avaliando o modelo para a base completa
# (não separamos ainda em treino e teste)
clf.score(X, y)
```

```
1.0
```

O significado disto pode ser averiguado visualmente. Abaixo, criamos um gráfico de dispersão das duas classes. Perceba que as duas são bem afastadas, permitindo uma completa separação por uma reta.

```
1  import matplotlib.pyplot as plt
2  fig, ax = plt.subplots()
3
4  ax.scatter(iris['petal length (cm)'], iris['petal width (cm)'],c=iris.target)
5
6  plt.show()
```



Módulo 15 – Entendendo o Perceptron

O algoritmo do Perceptron irá definir os parâmetros da reta que melhor separa as duas classes. Esses parâmetros podem ser obtidos com os atributos `coef_` e `interecept_`. De posse destes parâmetros, podemos construir a reta graficamente:

```
1  # w1 e w2
2  clf.coef_
```

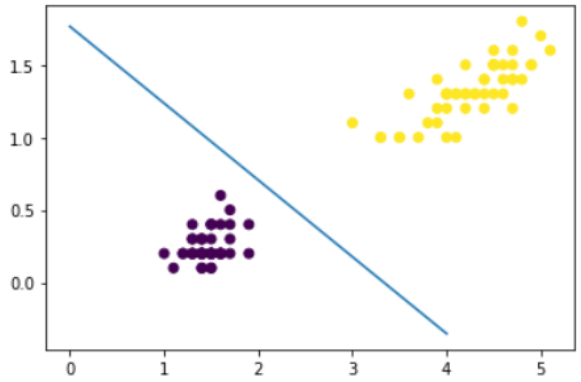
```
array([[0.9, 1.7]])
```

```
1  # w0
2  clf.intercept_
```

```
array([-3.])
```

```
1  # Criando a reta
2  import numpy as np
3
4  w1 = clf.coef_[0][0]
5  w2 = clf.coef_[0][1]
6  w0 = clf.intercept_[0]
7
8  x = np.arange(0,5)
9  y = (-w1*x-w0)/w2
```

```
1  # Visualizando de forma gráfica
2  fig, ax = plt.subplots()
3
4  ax.scatter(iris['petal length (cm)'], iris['petal width (cm)'],c=iris.target)
5  ax.plot(x,y)
6
7  plt.show()
```



Observe como a reta separa as duas classes perfeitamente, daí o *score* 1 apresentado anteriormente.

Dizemos que estas classes são linearmente separáveis, já que é possível traçar uma reta as separando.

Módulo 15 – Entendendo árvore de decisão

Ainda com o módulo íris, no módulo 13 estudamos árvore de decisão. Veja, ao lado, que com as mesmas colunas a árvore também consegue separar completamente as classes:

```
# Importando a árvore de decisão
from sklearn import tree
```

```
# Definindo os dados
X = iris[['petal length (cm)','petal width (cm)']]
y = iris.target
```

```
# Criando o nosso classificador
clf = tree.DecisionTreeClassifier()
```

```
# Fazendo o fit com os dados
clf.fit(X,y)
```

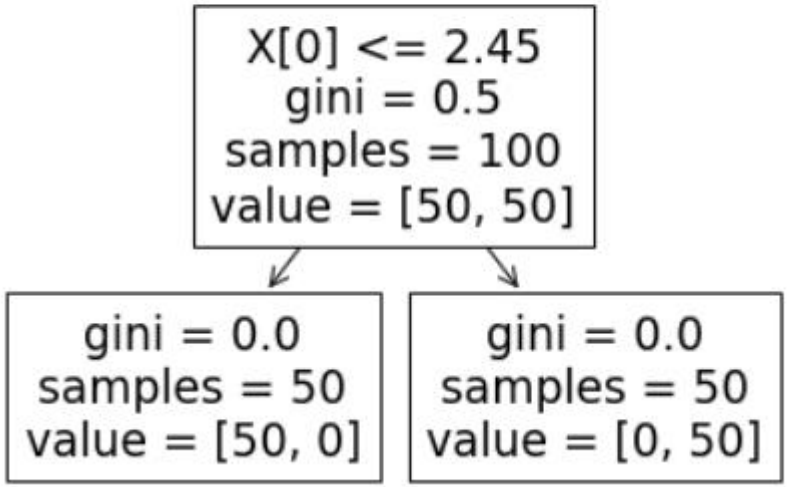
```
DecisionTreeClassifier()
```

```
# Avaliando o modelo
clf.score(X, y)
```

```
1.0
```

Essa árvore pode ser visualizado com método plot_tree. Observe como com uma simples decisão já é possível separar as classes.

```
1  # Visualizando as decisões tomadas por essa árvore
2  tree.plot_tree(clf);
```



Para facilitar a compreensão, usamos até este momento de nossos estudos apenas duas das três classes. Assim, nestes casos onde as classes são bem distintas, os modelos conseguem resolver rapidamente. Está na hora de colocarmos todas as três classes.

Módulo 15 – Comparando os modelos com mais uma classe

Vamos importar novamente nossa base de dados, mas agora não iremos mais remover a classe 2.

Agora vamos resolver nosso problema de classificação, camada por camada.

```
# Importando o pandas
import pandas as pd
```

```
# Importando o dataset iris
from sklearn.datasets import load_iris
data = load_iris()
```

```
# Transformando em um dataframe e usando somente target 0 e 1
iris = pd.DataFrame(data.data)
iris.columns = data.feature_names
iris['target'] = data.target
# iris = iris[iris.target != 2]
iris.head(2)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0

Módulo 15 – Comparando os modelos com mais uma classe

Comecemos verificando que realmente estamos com nossas 3 classes, cada uma com 50 amostras.

Novamente, criaremos nossos *X* e *y* para passar para o modelo. Usaremos as mesmas duas colunas de antes, para comparar os resultados.

```
1  # Verificando os valores na base
2  iris.target.value_counts()
```

```
2    50
1    50
0    50
Name: target, dtype: int64
```

```
1  # Definindo os valores de X e y do modelo
2  X = iris[['petal length (cm)','petal width (cm)']]
3  y = iris.target
```

Módulo 15 – Comparando os modelos com mais uma classe

Veja como agora os modelos não possuem mais o *score* com valor 1. Ou seja, a presença de mais uma classe já dificulta, neste caso, a separação completa das classes.

```
# Importação do perceptron
from sklearn.linear_model import Perceptron
```

```
# Criando o nosso classificador
clfPercep = Perceptron()
```

```
# Fazendo o fit do modelo
clfPercep.fit(X, y)
```

```
Perceptron()
```

```
# Avaliando o modelo para a base completa
# (não separamos ainda em treino e teste)
clfPercep.score(X, y)
```

```
0.8733333333333333
```

```
# Importando a árvore de decisão
from sklearn import tree
```

```
# Criando o nosso classificador
clfArvore = tree.DecisionTreeClassifier()
```

```
# Fazendo o fit com os dados
clfArvore.fit(X,y)
```

```
DecisionTreeClassifier()
```

```
# Avaliando o modelo
clfArvore.score(X, y)
```

```
0.9933333333333333
```

Módulo 15 – Comparando os modelos com mais uma classe

Aproveitando o conhecimento de módulos anteriores, podemos construir uma matriz de confusão para os dois modelos.

Veja que ambos os modelos classificaram corretamente as 50 amostras da classe 0. Já para a classe 1, a árvore de decisão foi bem mais performática que o perceptron. A árvore de decisão classificou corretamente as 50 amostras da classe 2.

```
1  from sklearn.metrics import confusion_matrix
```

```
1  predictArvore = clfArvore.predict(X)
2  predictPercep = clfPercep.predict(X)
```

```
1  # Perceptron
2  confusion_matrix(iris.target, predictPercep)
```

```
array([[50,  0,  0],
       [ 1, 33, 16],
       [ 0,  2, 48]])
```

```
1  # Árvore de Decisão
2  confusion_matrix(iris.target, predictArvore)
```

```
array([[50,  0,  0],
       [ 0, 49,  1],
       [ 0,  0, 50]])
```

Módulo 15 – Comparando os modelos com mais uma classe

Já vimos o conceito de três métricas: acurácia, precisão e recall. O método `score` dos classificadores, em geral, equivale à acurácia. Podemos perceber isto abaixo, onde importamos o `accuracy_score` e obtemos os mesmos valores vistos anteriormente:

```
1  from sklearn.metrics import accuracy_score
```

```
1  # Perceptron
2  accuracy_score(iris.target, predictPercep)
```

```
0.8733333333333333
```

```
1  # Árvore de Decisão
2  accuracy_score(iris.target, predictArvore)
```

```
0.9933333333333333
```

Módulo 15 – Comparando os modelos com mais uma classe

No caso da precisão, haverá uma pequena diferença comparando com o visto em módulos passados. Agora, temos 3 classes ao invés de 2, de forma que precisamos informar ao método como considerar os verdadeiros e falsos positivos, se por classe ou globalmente. Descreveremos melhor este aspecto mais adiante, por ora, passando o parâmetro `average='micro'` teremos a contagem feita globalmente:

```
1  from sklearn.metrics import precision_score
```

```
1  # Perceptron
2  precision_score(iris.target, predictPercep,average='micro')
```

```
0.8733333333333333
```

```
1  # Árvore de Decisão
2  precision_score(iris.target, predictArvore,average='micro')
```

```
0.9933333333333333
```

Módulo 15 – Comparando os modelos com mais uma classe

Seguindo a mesma lógica, no recall também é necessário passar o parâmetro **average**:

```
1  from sklearn.metrics import recall_score
```

```
1  # Perceptron
2  recall_score(iris.target, predictPercep,average='micro')
```

```
0.8733333333333333
```

```
1  # Árvore de Decisão
2  recall_score(iris.target, predictArvore,average='micro')
```

```
0.9933333333333333
```

Observe que, nas três métricas apresentadas, a árvore de decisão apresentou melhores resultados.

Módulo 15 – Entendendo o parâmetro *average*

Para entender um pouco melhor o parâmetro **average**, vamos criar alguns dados e avaliar a mudança na métrica a cada mudança deste parâmetro. Consideraremos 3 possíveis classes:

```
1  from sklearn.metrics import confusion_matrix
2  from sklearn.metrics import precision_score
```

```
1  y_true2 = [0, 0, 0, 1, 1, 1, 2, 2]
2  y_pred2 = [0, 1, 2, 1, 2, 2, 2, 2]
```

```
1  confusion_matrix(y_true2, y_pred2)
```

```
array([[1, 1, 1],
       [0, 1, 2],
       [0, 0, 2]])
```

Módulo 15 – Entendendo o parâmetro *average*

Um dos possíveis valores para o parâmetro é `None`. Neste caso, a pontuação de cada classe é retornada. Relembrando do conceito de precisão:

$$\text{Precisão} = \frac{\text{Verdadeiros positivos}}{\text{Verdadeiros positivos} + \text{Falsos positivos}}$$

```
1  confusion_matrix(y_true2, y_pred2)
```

```
array([[1, 1, 1],
       [0, 1, 2],
       [0, 0, 2]])
```

```
1  precision_score(y_true2, y_pred2,average=None)
```

```
array([1. , 0.5, 0.4])
```

Assim, vemos que para a primeira classe, temos 1 verdadeiro positivo (VP) e nenhum falso positivo (FP). Daí a pontuação 1. Para a segunda classe, temos 1 VP e 1 FP, resultando em 0.5. E, para a classe 3, temos 2 VP 3 FP, chegando em 0.4.

Módulo 15 – Entendendo o parâmetro *average*

Quando o parâmetro tem valor `micro`, as métricas são calculadas globalmente, considerando o total de VPs e FPs.

```
1  confusion_matrix(y_true2, y_pred2)
```

```
array([[1, 1, 1],
       [0, 1, 2],
       [0, 0, 2]])
```

```
1  precision_score(y_true2, y_pred2,average='micro')
```

```
0.5
```

```
1  4/8
```

```
0.5
```

Veja, na matriz de confusão, que o modelo possui 4 VP em um total global de 8 amostras. Daí o resultado de 0.5.

Módulo 15 – Entendendo o parâmetro *average*

Quando `macro`, calcula as métricas para cada classe e encontra sua média não ponderada. Na prática, é o valor médio daqueles encontrados em `None`:

```
1  confusion_matrix(y_true2, y_pred2)
```

```
array([[1, 1, 1],
       [0, 1, 2],
       [0, 0, 2]])
```

```
1  precision_score(y_true2, y_pred2,average='macro')
```

```
0.6333333333333333
```

```
1  (1+0.5+0.4)/3
```

```
0.6333333333333333
```

Módulo 15 – Entendendo o parâmetro *average*

Por fim, quando o parâmetro é passado como `weighted`, o método calcula as métricas para cada classe resultando em sua média ponderada pelo suporte. Suporte é o nome dado ao número de instâncias verdadeiras para cada classe. Vejamos a seguir:

```
1  confusion_matrix(y_true2, y_pred2)
```

```
array([[1, 1, 1],
       [0, 1, 2],
       [0, 0, 2]])
```

```
1  precision_score(y_true2, y_pred2,average='weighted')
```

```
0.6625
```

```
1  (3*1+3*0.5+2*0.4)/8
```

```
0.6625
```

Veja que há 3 valores verdadeiros para a classe 0, igualmente para a classe 1, e 2 valores verdadeiros para a classe 2. Ao multiplicarmos estes valores pelos verdadeiros positivos de cada classe e dividirmos pelo total de amostras, chegamos ao mesmo valor do método.

Módulo 15 – Separando os dados em treino e teste e avaliando os modelos

Até o momento expandimos nosso conhecimento de módulos anteriores sobre métodos de classificação e métricas para um caso multiclasse, com 3 classes.

No entanto, também vimos anteriormente que o mais correto em um projeto é separar a base em dados de treino e de teste, para se assegurar que o modelo não "decorou" a base de treino.

Assim, vamos acrescentar mais esta camada em nosso procedimento. O início é o mesmo, importar as bibliotecas e a base de dados:

```
# Importando o pandas
import pandas as pd
```

```
# Importando o dataset iris
from sklearn.datasets import load_iris
data = load_iris()
```

```
# Transformando em um dataframe e usando somente target 0 e 1
iris = pd.DataFrame(data.data)
iris.columns = data.feature_names
iris['target'] = data.target
# iris = iris[iris.target != 2]
iris.head(2)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0

```
# Verificando os valores na base
iris.target.value_counts()
```

```
2    50
1    50
0    50
Name: target, dtype: int64
```

```
# Definindo os valores de X e y do modelo
X = iris[['petal length (cm)','petal width (cm)']]
y = iris.target
```

Módulo 15 – Separando os dados em treino e teste e avaliando os modelos

Agora, com o train_test_split, separamos nossa base em treino e teste e passamos os dados de treino para nossos modelos:

```
1  from sklearn.model_selection import train_test_split
```

```
1  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
1  # Importação do perceptron
2  from sklearn.linear_model import Perceptron
```

```
1  # Criando o nosso classificador
2  clfPercep = Perceptron()
```

```
1  # Fazendo o fit do modelo
2  clfPercep.fit(X_train, y_train)
```

```
Perceptron()
```

```
1  # Avaliando o modelo para a base completa
2  # (não separamos ainda em treino e teste)
3  clfPercep.score(X, y)
```

```
0.6666666666666666
```

```
1  # Importando a árvore de decisão
2  from sklearn import tree
```

```
1  # Criando o nosso classificador
2  clfArvore = tree.DecisionTreeClassifier()
```

```
1  # Fazendo o fit com os dados
2  clfArvore.fit(X_train, y_train)
```

```
DecisionTreeClassifier()
```

```
1  # Avaliando o modelo
2  clfArvore.score(X, y)
```

```
0.9933333333333333
```

Módulo 15 – Separando os dados em treino e teste e avaliando os modelos

Com os modelos treinados, podemos aplica-los em nossos dados de teste e verificar as métricas.

Comecemos com as matrizes de confusão. Observe como os dois modelos classificam corretamente as classes 0 e 2. No entanto, o Perceptron não é um bom modelo para classificar a classe 1, enquanto que a árvore de decisão classifica corretamente todas as amostras da classe 1, assim como das demais.

```
1  from sklearn.metrics import confusion_matrix
```

```
1  predictArvore = clfArvore.predict(X_test)
2  predictPercep = clfPercep.predict(X_test)
```

```
1  # Perceptron
2  confusion_matrix(y_test, predictPercep)
```

```
array([[19,  0,  0],
       [ 7,  0,  6],
       [ 0,  0, 13]])
```

```
1  # Árvore de Decisão
2  confusion_matrix(y_test, predictArvore)
```

```
array([[19,  0,  0],
       [ 0, 13,  0],
       [ 0,  0, 13]])
```

Módulo 15 – Separando os dados em treino e teste e avaliando os modelos

Agora, podemos avaliar as três métricas que vimos estudando: acurácia, precisão e recall. Veja que a árvore de decisão apresenta valor 1 em todas, já que classificou corretamente todas as amostras de todas as classes.

```
from sklearn.metrics import accuracy_score
```

```
# Perceptron
accuracy_score(y_test, predictPercep)
```

```
0.7111111111111111
```

```
# Árvore de Decisão
accuracy_score(y_test, predictArvore)
```

```
1.0
```

```
from sklearn.metrics import precision_score
```

```
# Perceptron
precision_score(y_test, predictPercep,average='micro')
```

```
0.7111111111111111
```

```
# Árvore de Decisão
precision_score(y_test, predictArvore,average='micro')
```

```
1.0
```

```
from sklearn.metrics import recall_score
```

```
# Perceptron
recall_score(y_test, predictPercep,average='micro')
```

```
0.7111111111111111
```

```
# Árvore de Decisão
recall_score(y_test, predictArvore,average='micro')
```

```
1.0
```

Módulo 15 – Adicionando um novo modelo – Regressão logística

Agora temos um procedimento padronizado, sendo bem simples aplicar a um novo modelo.

Suponha, por exemplo, que a árvore de decisão que vimos ter um score perfeito anteriormente, tenha se mostrada muito lenta em um ambiente de produção, seja pelo volume de dados ou por ter gerado muitos nós de decisão. Podemos facilmente treinar um novo modelo agora que criamos esta estrutura inicial de um projeto de ciência de dados.

Vamos ver o modelo de regressão logística, também muito aplicado em ciência de dados. O início é o mesmo, importar bibliotecas e nossa base de dados:

```
# Importando o pandas
import pandas as pd
```

```
# Importando o dataset iris
from sklearn.datasets import load_iris
data = load_iris()
```

```
# Transformando em um dataframe e usando somente target 0 e 1
iris = pd.DataFrame(data.data)
iris.columns = data.feature_names
iris['target'] = data.target
# iris = iris[iris.target != 2]
iris.head(2)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0

```
# Verificando os valores na base
iris.target.value_counts()
```

```
2    50
1    50
0    50
Name: target, dtype: int64
```

```
# Definindo os valores de X e y do modelo
X = iris[['petal length (cm)','petal width (cm)']]
y = iris.target
```

```
# Definindo os valores de X e y do modelo
X2 = iris.drop('target',axis=1)
y2 = iris.target
```

Módulo 15 – Adicionando um novo modelo – Regressão logística

Depois, separamos nossa base em treino e teste e treinamos nossos modelos:

```
1  from sklearn.model_selection import train_test_split
```

```
1  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
1  # Importação do perceptron
2  from sklearn.linear_model import Perceptron
```

```
1  # Criando o nosso classificador
2  clfPercep = Perceptron()
```

```
1  # Fazendo o fit do modelo
2  clfPercep.fit(X_train2, y_train2)
```

```
Perceptron()
```

```
1  # Avaliando o modelo para a base completa
2  # (não separamos ainda em treino e teste)
3  clfPercep.score(X_train2, y_train2)
```

```
0.8
```

```
1  # Importando a árvore de decisão
2  from sklearn import tree
```

```
1  # Criando o nosso classificador
2  clfArvore = tree.DecisionTreeClassifier()
```

```
1  # Fazendo o fit com os dados
2  clfArvore.fit(X_train, y_train)
```

```
DecisionTreeClassifier()
```

```
1  # Avaliando o modelo
2  clfArvore.score(X_train, y_train)
```

```
0.9904761904761905
```

```
1  # Importar a regressão logística
2  from sklearn.linear_model import LogisticRegression
```

```
1  # Criando o nosso classificador
2  clfLogist = LogisticRegression(random_state=0).fit(X_train, y_train)
```

```
1  # Avaliando o modelo
2  clfLogist.score(X_train, y_train)
```

```
0.9523809523809523
```

Módulo 15 – Adicionando um novo modelo – Regressão logística

Por fim, construímos as matrizes de confusão e avaliamos as métricas. Perceba que a regressão logística também classifica todas as classes corretamente. Logo, se ela tiver melhor eficiência em produção, pode substituir a árvore de decisão. Observe, também, como foi simples adaptar a estrutura para estudar um novo modelo.

```
from sklearn.metrics import confusion_matrix
```

```
predictArvore = clfArvore.predict(X_test)
predictPercep = clfPercep.predict(X_test2)
predictLogist = clfLogist.predict(X_test)
```

```
# Perceptron
confusion_matrix(y_test2, predictPercep)
```

```
array([[19,  0,  0],
       [ 0,  5,  8],
       [ 0,  0, 13]])
```

```
# Árvore de Decisão
confusion_matrix(y_test, predictArvore)
```

```
array([[19,  0,  0],
       [ 0, 13,  0],
       [ 0,  0, 13]])
```

```
# Regressão Logística
confusion_matrix(y_test, predictLogist)
```

```
array([[19,  0,  0],
       [ 0, 13,  0],
       [ 0,  0, 13]])
```

```
from sklearn.metrics import accuracy_score
```

```
# Perceptron
accuracy_score(y_test2, predictPercep)
```

```
0.8222222222222222
```

```
# Árvore de Decisão
accuracy_score(y_test, predictArvore)
```

```
1.0
```

```
# Regressão Logística
accuracy_score(y_test, predictLogist)
```

```
1.0
```

```
from sklearn.metrics import precision_score
```

```
# Perceptron
precision_score(y_test2, predictPercep,average='micro')
```

```
0.8222222222222222
```

```
# Árvore de Decisão
precision_score(y_test, predictArvore,average='micro')
```

```
1.0
```

```
# Regressão Logística
precision_score(y_test, predictLogist,average='micro')
```

```
1.0
```

```
from sklearn.metrics import recall_score
```

```
# Perceptron
recall_score(y_test2, predictPercep,average='micro')
```

```
0.8222222222222222
```

```
# Árvore de Decisão
recall_score(y_test, predictArvore,average='micro')
```

```
1.0
```

```
# Regressão Logística
recall_score(y_test, predictLogist,average='micro')
```

```
1.0
```

Módulo 15 – Entendendo regressão linear

Já nos aprofundamos em modelos de classificação. Está no momento de nos aprofundarmos no modelo de regressão linear, que vimos de forma introdutória em módulos passados.

A regressão linear é um método estatístico que busca modelar a relação entre uma variável dependente (y) e uma ou mais variáveis independentes (x). O objetivo é encontrar a melhor linha reta (ou hiperplano, no caso de múltiplas variáveis) que minimize a soma dos erros quadráticos entre os valores reais e os valores previstos pelo modelo. A regressão linear é amplamente utilizada na análise e previsão de dados devido à sua simplicidade, interpretabilidade e eficiência computacional.

Veja ao lado que criamos 3 pares (x, y). No caso, eles são descritos perfeitamente pela reta `y = 2500 x + 4100`. Usamos o `LinearRegression` do Scikit-Learn e os atributos `coef_` e `intercept_` para descobrir os coeficientes angular e linear, respectivamente, da reta. E percebemos que são perfeitamente lineares pelo `score` de valor 1.

```
import pandas as pd
```

```
# Dados 1
dados1 = {
    'X': [1,2,3],
    'Y': [6600,9100,11600]
}

dados1 = pd.DataFrame(dados1)
```

```
# Importando a Regressão Linear
from sklearn import linear_model
```

```
# Criando nosso algoritmo de regressão
reg = linear_model.LinearRegression()
reg.fit(dados1.X.values.reshape(-1, 1),dados1.Y)
```

```
LinearRegression()
```

```
# Avaliando o modelo
reg.score(dados1.X.values.reshape(-1, 1),dados1.Y)
```

```
1.0
```

```
# Determinando o coeficiente angular
reg.coef_
```

```
array([2500.])
```

```
# Determinando o coeficiente linear
reg.intercept_
```

```
4100.000000000001
```

Módulo 15 – Entendendo regressão linear

O que ocorre se mudar um dos valores? Veja que, alterando o último valor da lista *Y*, já não temos mais uma relação perfeitamente linear pelo valor do `score`. No entanto, o valor ainda é alto o suficiente para podermos considerar que um modelo linear descreve bem o conjunto de pontos.

Podemos verificar de forma visual, a partir dos coeficientes.

```
# Dados 2
dados2 = {
    'X': [1,2,3],
    'Y': [6600,9100,11000]
}

dados2 = pd.DataFrame(dados2)
```

```
# Criando nosso algoritmo de regressão
reg2 = linear_model.LinearRegression()
reg2.fit(dados2.X.values.reshape(-1, 1),dados2.Y)
```

```
LinearRegression()
```

```
# Avaliando o modelo
reg2.score(dados2.X.values.reshape(-1, 1),dados2.Y)
```

```
0.9938398357289527
```

```
# Determinando o coeficiente angular
reg2.coef_
```

```
array([2200.])
```

```
# Determinando o coeficiente linear
reg2.intercept_
```

```
4500.000000000001
```

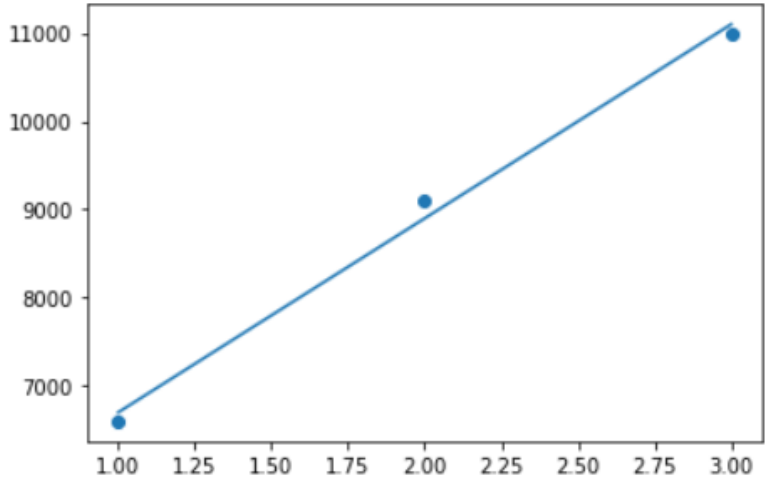
Módulo 15 – Entendendo regressão linear

Observe que a reta, construída a partir dos coeficientes fornecidos pelo modelo, não passa por nenhum dos pontos, mas descreve uma tendência destes.

O que vimos nestas últimas páginas é similar ao que já havíamos visto em módulos anteriores. Agora, vamos aplicar o método de regressão linear em dados reais.

```
a = reg2.coef_[0]
b = reg2.intercept_
```

```
# plot
import matplotlib.pyplot as plt
import numpy as np
fig, ax = plt.subplots()

ax.scatter(dados2.X,dados2.Y)
x=np.arange(1,4)
y=a*x+b
ax.plot(x,y)

plt.show()
```



Módulo 15 – Regressão linear no mercado de ações

Vamos pegar uma base de dados contendo a cotação das ações da empresa Tesla na bolsa americana entre 2010 e 2020. Nosso objetivo é estudar se há alguma relação entre o valor da ação e o volume negociado. Nesta base temos:

- Date: data
- Open: preço na abertura do pregão
- High: maior preço durante o dia
- Low: menor preço durante o dia
- Close: preço no fechamento do pregão
- Adj Close: preço após ajustes (split, dividendos, etc)
- Volume: volume de negociações

```
1  import pandas as pd
```

```
1  # Importando a base
2  base = pd.read_csv('TSLA.csv',parse_dates=['Date'])
```

```
1  # Visualizando as primeiras linhas
2  base.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2010-06-29	19.000000	25.00	17.540001	23.889999	23.889999	18766300
1	2010-06-30	25.790001	30.42	23.299999	23.830000	23.830000	17187100
2	2010-07-01	25.000000	25.92	20.270000	21.959999	21.959999	8218800
3	2010-07-02	23.000000	23.10	18.709999	19.200001	19.200001	5139800
4	2010-07-06	20.000000	20.00	15.830000	16.110001	16.110001	6866900

Como sempre, vamos começar verificando a integridade da base e os tipos de dados. No caso, vemos que a coluna de data está no formato adequado, acima como as colunas de preços e de volume. E não há problemas de duplicatas.

```
1  # Visualizando as informações da base
2  base.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2416 entries, 0 to 2415
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       2416 non-null   datetime64[ns]
 1   Open       2416 non-null   float64
 2   High       2416 non-null   float64
 3   Low        2416 non-null   float64
 4   Close      2416 non-null   float64
 5   Adj Close  2416 non-null   float64
 6   Volume     2416 non-null   int64
dtypes: datetime64[ns](1), float64(5), int64(1)
memory usage: 132.2 KB
```

```
1  # Verificando valores duplicados
2  base.duplicated().sum()
```

```
0
```

Continuando, com o `describe` temos um resumo estatístico de nossa base. Observe, pela diferença entre os valores mínimos e máximos, que as ações valorizaram significativamente no período. E, também, que houve um aumento no volume negociado.

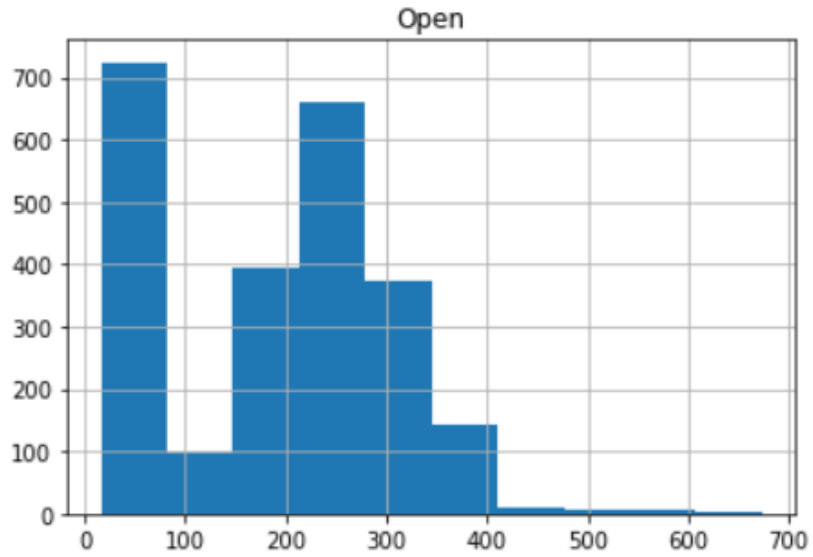
```
1  # Obtendo as informações estatísticas
2  base.describe()
```

	Open	High	Low	Close	Adj Close	Volume
count	2416.000000	2416.000000	2416.000000	2416.000000	2416.000000	2.416000e+03
mean	186.271147	189.578224	182.916639	186.403651	186.403651	5.572722e+06
std	118.740163	120.892329	116.857591	119.136020	119.136020	4.987809e+06
min	16.139999	16.629999	14.980000	15.800000	15.800000	1.185000e+05
25%	34.342498	34.897501	33.587501	34.400002	34.400002	1.899275e+06
50%	213.035003	216.745002	208.870002	212.960007	212.960007	4.578400e+06
75%	266.450012	270.927513	262.102501	266.774994	266.774994	7.361150e+06
max	673.690002	786.140015	673.520020	780.000000	780.000000	4.706500e+07

Cada uma das colunas pode ser estudada graficamente, utilizando todas as visualizações que estudamos. Por exemplo, abaixo temos o histograma da coluna *Open*, mostrando que a maior parte das vezes o valor da ação na abertura foi abaixo de 100, mas há uma quantidade significativa entre 150 e 400. Na realidade, somando as barras, esta faixa de valores possui mais entradas correspondentes. Pratique fazendo o histograma das demais colunas.

```
1  # Visualizando o histograma dos dados
2  base.hist('Open')
```

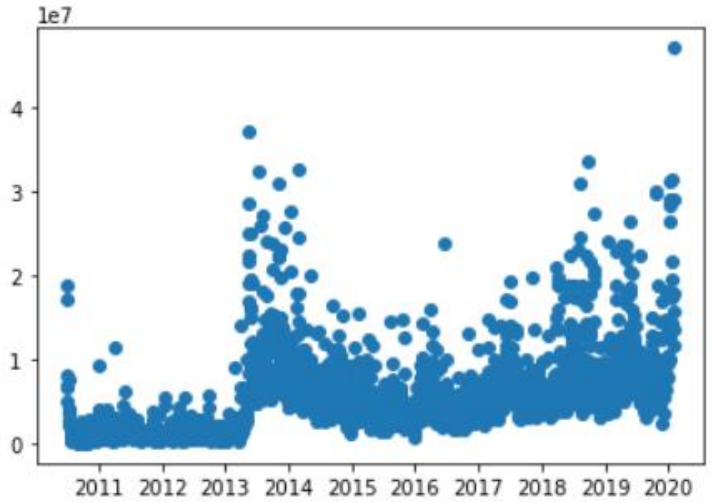
```
array([[<AxesSubplot:title={'center':'Open'}>]], dtype=object)
```



Módulo 15 – Regressão linear no mercado de ações

O histograma avalia a coluna isoladamente. Podemos fazer gráficos de dispersão para avaliar pares de colunas, estudando a relação entre as mesmas. Por exemplo, considerando a variação do volume negociado no tempo, vemos um aumento significativo por volta de 2013 e uma tendência de alta ao se aproximar de 2020:

```
1  # Gerando um gráfico de dispersão para verificar a relação entre as informações
2  import matplotlib.pyplot as plt
3  fig, ax = plt.subplots()
4
5  ax.scatter(base.Date, base.Volume)
6
7  plt.show()
```



Módulo 15 – Regressão linear no mercado de ações

Como queremos estudar a relação entre preço e volume, vamos criar um modelo de regressão. Vamos considerar o preço de abertura num primeiro momento. Considerando o que já aprendemos, vamos começar separando em base de treino e de teste:

```
1  from sklearn.model_selection import train_test_split
2  X_train, X_test, y_train, y_test = train_test_split(base.Open,base.Volume, test_size=0.33, random_state=42)
```

```
1  X_train
```

```
1572    206.500000
393      27.190001
998     224.110001
534      29.520000
30       18.690001
           ...
1638    218.559998
1095    243.000000
1130    223.809998
1294    260.329987
860     119.379997
Name: Open, Length: 1618, dtype: float64
```

Módulo 15 – Regressão linear no mercado de ações

Agora, baste treinar o modelo e avaliar os resultados na base de teste. Observe que tivemos valores de score relativamente baixos, mas o modelo performou melhor, considerando esta métrica, na base de teste:

```
# Importando a Regressão Linear
from sklearn.linear_model import LinearRegression
```

```
# Criando nosso algoritmo de regressão
reg = LinearRegression().fit(X_train.values.reshape(-1,1), y_train)
```

```
# Avaliando o modelo nos dados de TREINO
reg.score(X_train.values.reshape(-1,1), y_train)
```

```
0.21503603604976262
```

```
# Avaliando o modelo nos dados de TESTE
reg.score(X_test.values.reshape(-1,1), y_test)
```

```
0.3284195874914201
```

```
# Determinando o coeficiente angular
reg.coef_[0]
```

```
20195.187973651628
```

```
# Determinando o coeficiente linear
reg.intercept_
```

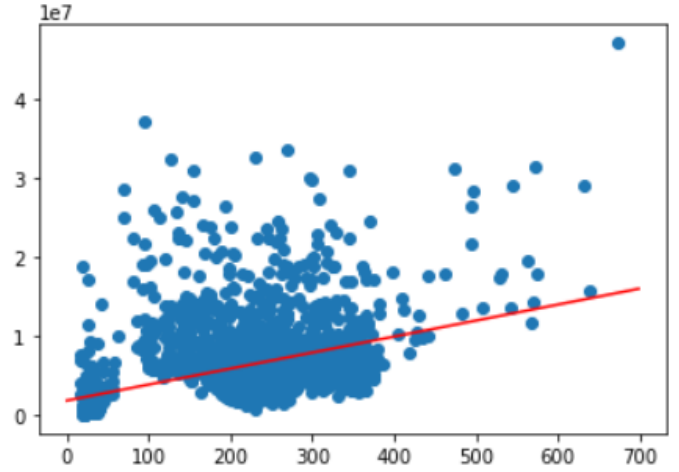
```
1904679.7631864208
```

Módulo 15 – Regressão linear no mercado de ações

Com os valores dos coeficientes, podemos visualizar a reta do nosso modelo.

Como vemos, a reta mostra a tendência crescente dos valores, mas provavelmente geraria muitos erros se fosse utilizada isoladamente para previsões. Podemos aumentar a complexidade do nosso modelo passando mais colunas para treiná-lo.

```
1  # Visualizando de forma gráfica
2  import numpy as np
3
4  a = reg.coef_[0]
5  b = reg.intercept_
6
7  x = np.arange(0,700)
8  y = a*x+b
9
10 fig, ax = plt.subplots()
11
12 ax.scatter(base.Open, base.Volume)
13 ax.plot(x,y,c='r')
14
15 plt.show()
```



Vamos, por exemplo, passar mais uma coluna, a *High*.

```
1  base.head(2)
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2010-06-29	19.000000	25.00	17.540001	23.889999	23.889999	18766300
1	2010-06-30	25.790001	30.42	23.299999	23.830000	23.830000	17187100

```
1  # Definindo X e y
2  X = base[['Open','High']]
3  y = base.Volume
```

Observe que a simples adição de mais de uma coluna já fez melhorar o nosso `score` (compare com os valores das páginas anteriores). Você pode fazer de exercício a adição de mais colunas e, inclusive, passar todas as colunas da base.

```
# Separando novamente em treino e teste
from sklearn.model_selection import train_test_split
X_train2, X_test2, y_train2, y_test2 = train_test_split(X,y, test_size=0.33, random_state=42)
```

```
# Usando a regressão linear
reg2 = LinearRegression().fit(X_train2, y_train2)
```

```
# Avaliando nos dados de treino
reg2.score(X_train2, y_train2)
```

```
0.3492902359670137
```

```
# Avaliando nos dados de teste
reg2.score(X_test2, y_test2)
```

```
0.39825327242737796
```

Módulo 15 – Regressão linear no mercado de ações

Agora que temos uma regressão linear múltipla, é um pouco mais difícil ter uma visualização. Mas a lógica é a mesmo para uma regressão linear simples. Podemos obter os coeficientes:

```
1  # Coeficiente angular
2  reg2.coef_
```

```
array([-408577.50212954,  421052.85560007])
```

```
1  # Coeficiente linear
2  reg2.intercept_
```

```
1928821.1465417575
```

Cada coeficiente angular é um peso passado para uma das colunas *X*, e temos nosso intercepto. Assim, nossa equação se torna

$$y = w_i \cdot x + w_0$$

Uma forma que já vimos em nosso material anteriormente.

MÓDULO 16

Criando um algoritmo de regressão

Módulo 16 – Conhecendo a base de casas da Califórnia

Neste módulo, utilizaremos a base de dados California Housing, presente no Scikit-Learn.

Módulo 16 – Conhecendo a base de casas da Califórnia

A base de dados California Housing é um conjunto de dados amplamente utilizado para estudar e aplicar técnicas de regressão. O conjunto contém informações demográficas e habitacionais de diferentes regiões da Califórnia, coletadas no Censo de 1990. As variáveis incluem características como renda média, idade média da habitação, população e número de cômodos. O objetivo é prever o valor médio de uma casa em uma determinada região.

```
1  # Importando a base
2  from sklearn.datasets import fetch_california_housing
```

```
1  # Visualizando a base importada
2  retorno = fetch_california_housing()
3  retorno.DESCR
```

```
'.. _california_housing_dataset:\n\nCalifornia Housing dataset\n--------------------------\n\n**Data Set Characteristics:**\n\n    :Number of Instances: 20640\n\n    :Number of Attributes:
8 numeric, predictive attributes and the target\n\n    :Attribute Information:\n        - MedInc        median income in block group\n        - HouseAge      median house age in block grou
p\n        - AveRooms      average number of rooms per household\n        - AveBedrms     average number of bedrooms per household\n        - Population    block group population\n
- AveOccup      average number of household members\n        - Latitude      block group latitude\n        - Longitude     block group longitude\n\n    :Missing Attribute Values: None\n\nT
his dataset was obtained from the StatLib repository.\nhttps://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html\n\nThe target variable is the median house value for California district
s,\nexpressed in hundreds of thousands of dollars ($100,000).\n\nThis dataset was derived from the 1990 U.S. census, using one row per census\nblock group. A block group is the smallest ge
ographical unit for which the U.S.\nCensus Bureau publishes sample data (a block group typically has a population\nof 600 to 3,000 people).\n\nAn household is a group of people residing wi
thin a home. Since the average\nnumber of rooms and bedrooms in this dataset are provided per household, these\ncolumns may take surpinsingly large values for block groups with few househo
lds\nand many empty houses, such as vacation resorts.\n\nIt can be downloaded/loaded using the\n:func:`sklearn.datasets.fetch_california_housing` function.\n\n.. topic:: References\n\n
- Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,\n      Statistics and Probability Letters, 33 (1997) 291-297\n'
```

Com o atributo **DESCR** vemos, em forma de texto, o conteúdo. O que precisamos é transformar em forma de dataframe do Pandas.

Módulo 16 – Conhecendo a base de casas da Califórnia

Outros atributos que utilizamos para a construção do dataframe: `data`, `feature_names` e `target`. Respectivamente, fornecendo as variáveis, seus nomes, e a variável alvo.

```
1  # Transformando em um DataFrame
2  import pandas as pd
3  casas = pd.DataFrame(retorno.data)
4  casas.columns = retorno.feature_names
5  casas['MedHouseVal'] = retorno.target
6  casas.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

```
1  # Criando as variáveis X e y
2  X = casas.drop('MedHouseVal',axis=1)
3  y = casas.MedHouseVal
```

Módulo 16 – Conhecendo a base de casas da Califórnia

As colunas da base de dados e seus significados são:

- *MedInc*: renda média dos residentes de uma região (em dezenas de milhares de dólares)
- *HouseAge*: idade média das casas em uma região (em anos)
- *AveRooms*: número médio de cômodos por casa em uma região
- *AveBedrms*: número médio de quartos por casa em uma região
- *Population*: número total de pessoas residentes em uma região
- *AveOccup*: número médio de residents por casa em uma região
- *Latitude*: latitude da região
- *Longitude*: longitude da região
- *MedHouseVal*: valor médio das casas em uma região (em unidades de 100.000 dólares) – variável alvo a ser prevista

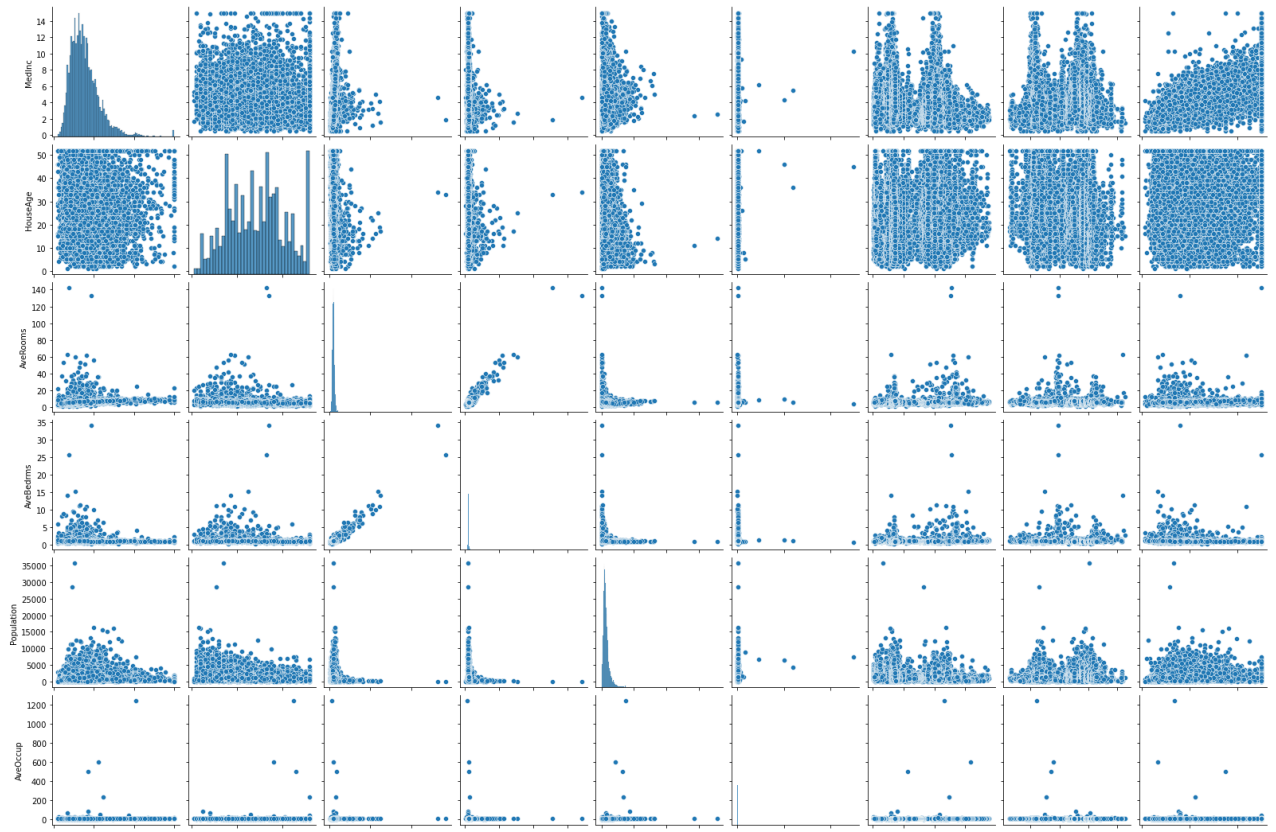
Módulo 16 – Visualizando os dados

Aproveitando o que já vimos nos últimos módulos, podemos usar o método `pairplot` do Seaborn para ter uma ideia da distribuição dos valores de cada coluna.

Nesta visualização, busque identificar o formato das distribuições e os pontos que se destacam como outliers, praticando o que estamos estudando seguidamente no curso.

```
# Utilizando o pairplot do seaborn
import seaborn as sns
sns.pairplot(casas)
```

```
<seaborn.axisgrid.PairGrid at 0x7fae202eca30>
```

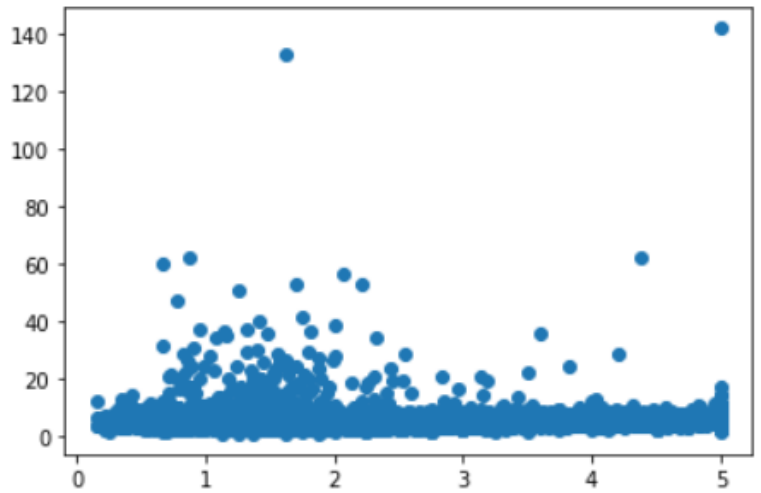


Módulo 16 – Visualizando os dados

Para mais detalhes, podemos construir os gráficos de dispersão de pares específicos. Por exemplo, podemos verificar se há alguma relação entre o preço médio de uma casa e o número de cômodos.

O preço médio está no eixo horizontal, lembrando que está expresso em centenas de milhares de dólares (ou seja, 1 significa $100.000,00). No eixo vertical, temos a média de cômodos. Observe que há valores elevados de cômodos para alguns pontos. Isto pode ser devido a problemas na base ou a situações específicas como, por exemplo citado na documentação, casas com quartos para alugar ou casas de repouso.

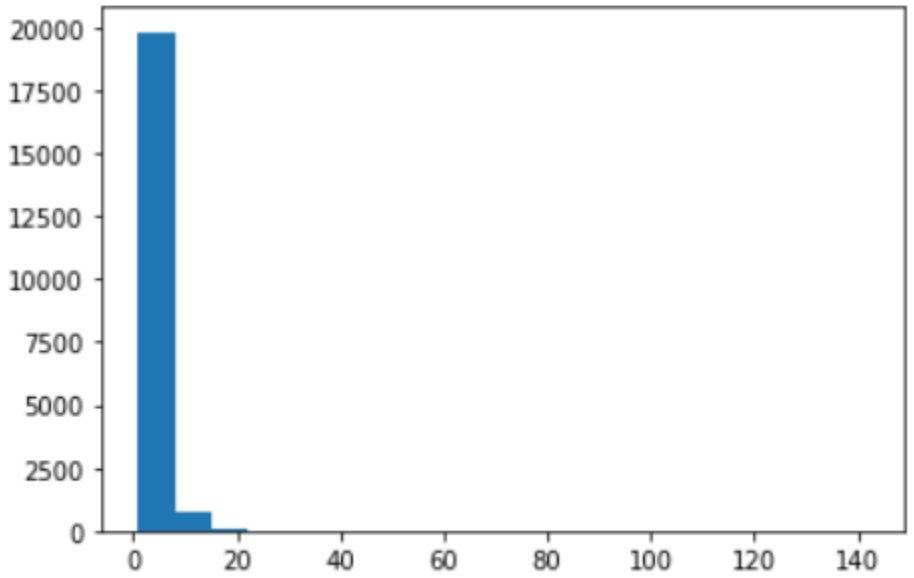
```
1  # Criando uma scatter plot
2  import matplotlib.pyplot as plt
3  fig,ax = plt.subplots()
4
5  ax.scatter(casas.MedHouseVal,casas.AveRooms)
6
7  plt.show()
```



Módulo 16 – Visualizando os dados

Podemos perceber melhor esta questão dos cômodos com um histograma. Veja como boa parte está na faixa usual de valores baixos.

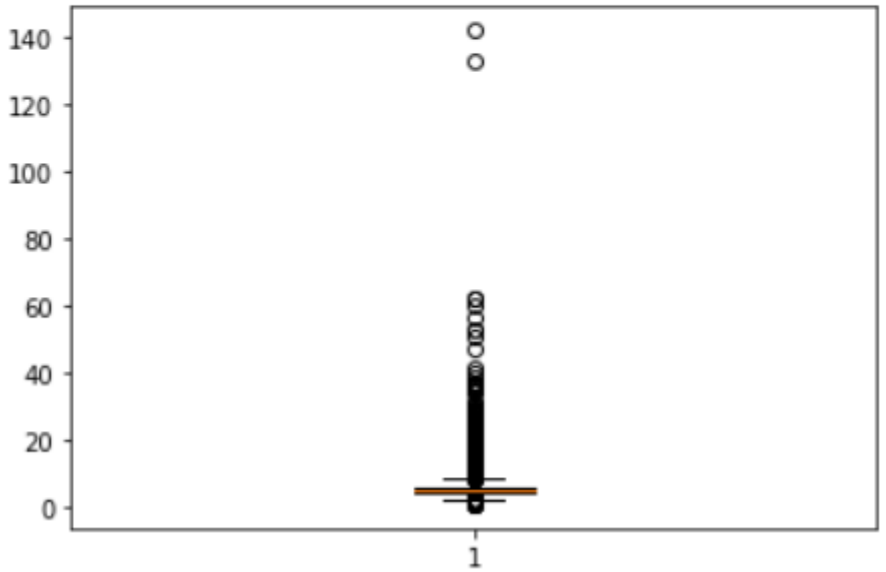
```
1  # Visualizando o histograma dos dados
2  fig,ax = plt.subplots()
3
4  ax.hist(casas.AveRooms,bins=20)
5
6  plt.show()
```



Módulo 16 – Visualizando os dados

Outra visualização que mostra a distribuição de cômodos é o boxplot. Nele, fica evidente que há maior parte dos dados em valores pequenos e há dois pontos com valores muito elevados.

```
1  # Verificando outliers com o boxplot
2  fig,ax = plt.subplots()
3
4  ax.boxplot(casas.AveRooms)
5
6  plt.show()
```



Módulo 16 – Visualizando os dados

Com o método **`describe`**, vemos que 75 % dos dados estão em até cerca de 6 cômodos.

```
1  casas.AveRooms.describe()
```

```
count    20640.000000
mean         5.429000
std          2.474173
min          0.846154
25%          4.440716
50%          5.229129
75%          6.052381
max        141.909091
Name: AveRooms, dtype: float64
```

Módulo 16 – Visualizando os dados

Com o método `corr`, podemos buscar quais variáveis tem maior correlação com a variável alvo.

```
1  # Criando uma matriz de correlação
2  casas.corr()
```

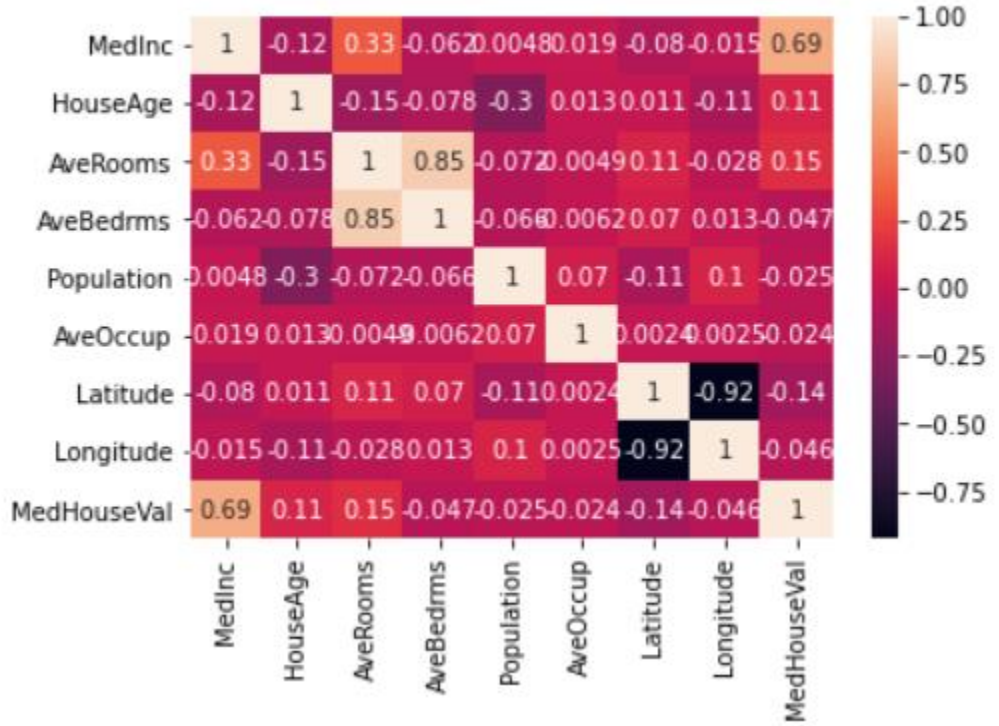
	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
MedInc	1.000000	-0.119034	0.326895	-0.062040	0.004834	0.018766	-0.079809	-0.015176	0.688075
HouseAge	-0.119034	1.000000	-0.153277	-0.077747	-0.296244	0.013191	0.011173	-0.108197	0.105623
AveRooms	0.326895	-0.153277	1.000000	0.847621	-0.072213	-0.004852	0.106389	-0.027540	0.151948
AveBedrms	-0.062040	-0.077747	0.847621	1.000000	-0.066197	-0.006181	0.069721	0.013344	-0.046701
Population	0.004834	-0.296244	-0.072213	-0.066197	1.000000	0.069863	-0.108785	0.099773	-0.024650
AveOccup	0.018766	0.013191	-0.004852	-0.006181	0.069863	1.000000	0.002366	0.002476	-0.023737
Latitude	-0.079809	0.011173	0.106389	0.069721	-0.108785	0.002366	1.000000	-0.924664	-0.144160
Longitude	-0.015176	-0.108197	-0.027540	0.013344	0.099773	0.002476	-0.924664	1.000000	-0.045967
MedHouseVal	0.688075	0.105623	0.151948	-0.046701	-0.024650	-0.023737	-0.144160	-0.045967	1.000000

Lembrando do que já estudamos, um valor positivo indica correlação positiva (quando um valor aumenta, o outro também aumenta). E um valor negativo indica o oposto (quando um valor aumenta, o outro diminui). Quanto mais se aproximam de -1 ou +1 maior a correlação, seja negativa ou positiva.

Veja a correlação elevada do valor médio da casa com a renda média.

Visualmente, as correlações ficam mais evidentes com um mapa de calor, criado a partir do método **heatmap** do Seaborn.

```
1  sns.heatmap(casas.corr(),annot=True);
```



Módulo 16 – Verificando valores duplicados e outliers

Agora que temos uma noção visual da base, vamos verificar o quanto a base está completa, se há valores duplicados e se iremos ou não remover outliers.

Com o método `info`, vemos que todas as colunas estão com os tipos numéricos e sem valores ausentes.

```
1  # Verificando as informações da base
2  casas.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   MedInc       20640 non-null  float64
 1   HouseAge     20640 non-null  float64
 2   AveRooms     20640 non-null  float64
 3   AveBedrms    20640 non-null  float64
 4   Population   20640 non-null  float64
 5   AveOccup     20640 non-null  float64
 6   Latitude     20640 non-null  float64
 7   Longitude    20640 non-null  float64
 8   MedHouseVal  20640 non-null  float64
dtypes: float64(9)
memory usage: 1.4 MB
```

Com o método **`duplicated`**, vemos que não há linhas duplicadas nesta base de dados.

```
1  # Verificando se existe algum valor duplicado
2  casas.duplicated().sum()
```

```
0
```

Módulo 16 – Verificando valores duplicados e outliers

Com o `describe`, temos um resumo estatístico de cada coluna. Perceba que em algumas há uma grande diferença entre o valor na linha *75%* e o valor na linha *max*, o que talvez indique outliers superiores. Uma grande diferença entre *25%* e *min* pode indicar outliers inferiores. Diagramas de boxplot podem confirmar se realmente são, mas aqui já temos alguns indícios.

```
1  # Adicionando as informações estatísticas
2  casas.describe()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	3.870671	28.639486	5.429000	1.096675	1425.476744	3.070655	35.631861	-119.569704	2.068558
std	1.899822	12.585558	2.474173	0.473911	1132.462122	10.386050	2.135952	2.003532	1.153956
min	0.499900	1.000000	0.846154	0.333333	3.000000	0.692308	32.540000	-124.350000	0.149990
25%	2.563400	18.000000	4.440716	1.006079	787.000000	2.429741	33.930000	-121.800000	1.196000
50%	3.534800	29.000000	5.229129	1.048780	1166.000000	2.818116	34.260000	-118.490000	1.797000
75%	4.743250	37.000000	6.052381	1.099526	1725.000000	3.282261	37.710000	-118.010000	2.647250
max	15.000100	52.000000	141.909091	34.066667	35682.000000	1243.333333	41.950000	-114.310000	5.000010

Módulo 16 – Verificando valores duplicados e outliers

Uma outra análise interessante é a de valores únicos por coluna. Observe como há poucos valores únicos na coluna referente a idade da casa, enquanto que outras colunas como a de média de cômodos possui diversos valores.

Esta análise é interessante pois, em algumas bases de dados, algumas colunas possuem poucos valores únicos que podem ser trabalhadas como colunas de variáveis categóricas. Não é o caso desta base, mas aparecerão outras no curso.

```
1  # Verificando a cardinalidade
2  casas.nunique()
```

```
MedInc          12928
HouseAge           52
AveRooms        19392
AveBedrms       14233
Population       3888
AveOccup        18841
Latitude          862
Longitude         844
MedHouseVal      3842
dtype: int64
```

Módulo 16 – Verificando valores duplicados e outliers

O método `value_counts` permite contar quantas vezes um determinado valor aparece. Abaixo, temos os 5 valores que mais aparecem (*head*) e os 5 que menos aparecem (*tail*) na coluna *HouseAge*. Ou seja, temos 1273 casas com 52 anos de idade, e 4 casas com 1 ano de idade.

```
1  casas.HouseAge.value_counts().head()
```

```
52.0    1273
36.0     862
35.0     824
16.0     771
17.0     698
Name: HouseAge, dtype: int64
```

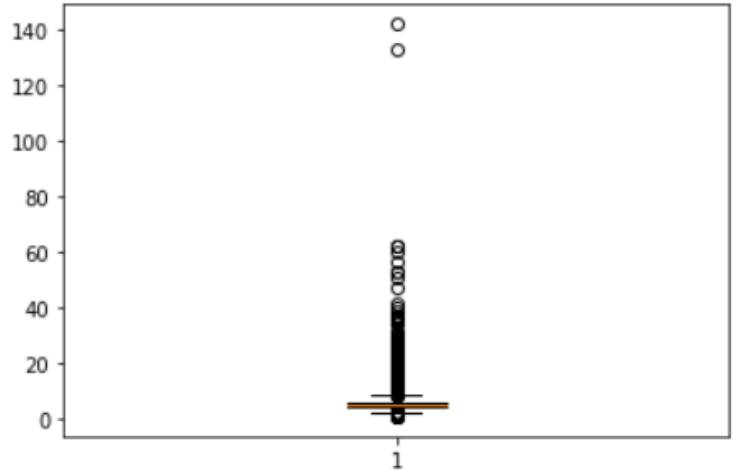
```
1  casas.HouseAge.value_counts().tail()
```

```
49.0    134
3.0      62
2.0      58
51.0     48
1.0       4
Name: HouseAge, dtype: int64
```

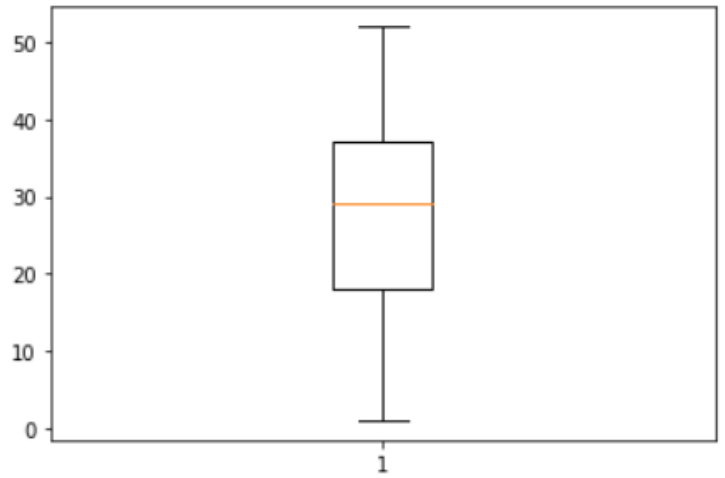
Módulo 16 – Verificando valores duplicados e outliers

Como vimos no `describe`, algumas colunas parecem indicar a presença de outliers. A seguir, comparamos os diagramas de boxplot de *AveRooms* e de *HouseAge*. Compare os diagramas com o `describe` e perceba a diferença nos outliers entre os diagramas.

```
1  # Visualizando o boxplot para a coluna 'AveRooms'
2  fig,ax = plt.subplots()
3
4  ax.boxplot(casas.AveRooms)
5
6  plt.show()
```




```
1  # Visualizando para a coluna 'HouseAge'
2  fig,ax = plt.subplots()
3
4  ax.boxplot(casas.HouseAge)
5
6  plt.show()
```



Módulo 16 – Verificando valores duplicados e outliers

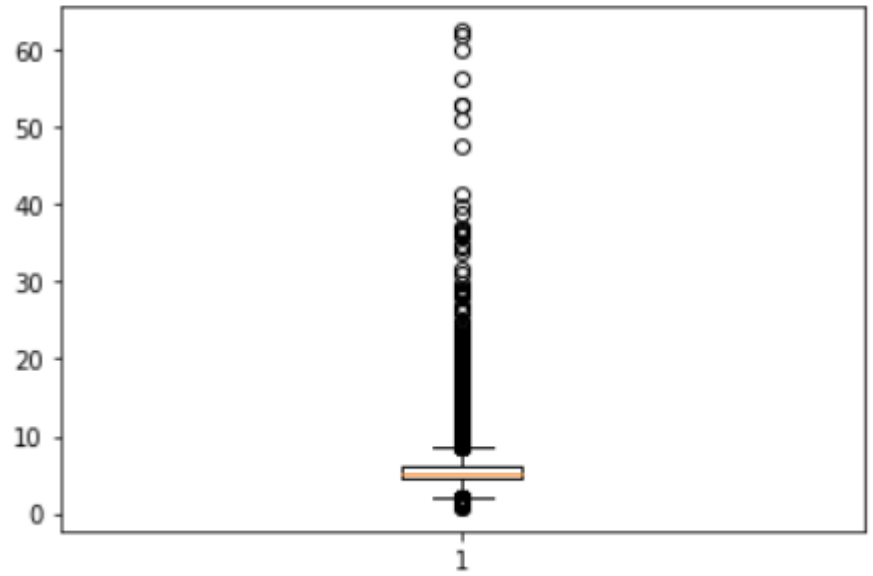
No diagrama de boxplot de *AveRooms*, vemos dois pontos muito acima dos demais. Podemos retirar estes de nossa base de dados:

```
# Tratando os dados
casas = casas[casas.AveRooms < 100]
```

```
casas.shape
```

```
(20638, 9)
```

```
# Visualizando novamente
fig,ax = plt.subplots()

ax.boxplot(casas.AveRooms)

plt.show()
```



Módulo 16 – Separando a base em treino e teste e usando regressão linear simples

Agora que já fizemos o reconhecimento de nossa base e tratamento de dados básico, podemos começar o procedimento de treinamento de modelos. Como visto nos módulos anteriores, começando separando nossa base em duas, uma base de treino e outra base de teste:

```
1  from sklearn.model_selection import train_test_split
```

```
1  casas.head(2)
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	8.3252	41.0	6.984127	1.02381	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.97188	2401.0	2.109842	37.86	-122.22	3.585

```
1  X = casas.drop('MedHouseVal',axis=1)
2  y = casas.MedHouseVal
```

```
1  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

Módulo 16 – Separando a base em treino e teste e usando regressão linear simples

Quando fizemos a visualização de dispersão de nossa base, com o pairplot, vimos que algumas colunas tem comportamento aparentemente mais linear com a variável alvo do que outras. Podemos, por exemplo, comparar *MedInc* com *AveRooms* utilizando o modelo de regressão linear. Perceba como o score para *MedInc* é significativamente maior:

```
# Criando nossa regressão e fazendo o fit com os dados
from sklearn.linear_model import LinearRegression
RegSim1 = LinearRegression().fit(X_train.MedInc.values.reshape(-1,1), y_train)
```

```
# Verificando o coeficiente de determinação
print(RegSim1.score(X_train.MedInc.values.reshape(-1,1), y_train))
print(RegSim1.score(X_test.MedInc.values.reshape(-1,1), y_test))
```

```
0.4722764318735512
0.4780251100283853
```

```
# Escolhendo outra coluna e verificando se o resultado está melhor
RegSim2 = LinearRegression().fit(X_train.AveRooms.values.reshape(-1,1), y_train)
print(RegSim2.score(X_train.AveRooms.values.reshape(-1,1), y_train))
print(RegSim2.score(X_test.AveRooms.values.reshape(-1,1), y_test))
```

```
0.027892442711821674
0.033409092499478765
```

Módulo 16 – Separando a base em treino e teste e usando regressão linear simples

Com um loop for, podemos fazer a análise para todas as colunas. Observe como a única que apresenta um valor significativo é a *MedInc*:

```
# Usando o for para fazer a regressão de todas as colunas
lista = X_train.columns

for i in lista:
    print(i)
    RegSim3 = LinearRegression().fit(X_train[i].values.reshape(-1,1), y_train)
    print(RegSim3.score(X_train[i].values.reshape(-1,1), y_train))
    print(RegSim3.score(X_test[i].values.reshape(-1,1), y_test))
```

```
MedInc
0.4722764318735512
0.4780251100283853
HouseAge
0.009901923713279026
0.01435329520341111
AveRooms
0.027892442711821674
0.033409092499478765
AveBedrms
0.0039958828973664096
0.004431841062501274
Population
0.0004691454405161277
0.0005852769184258033
AveOccup
0.000999915854954958
-0.0037063533253798298
Latitude
0.021513560930008224
0.018471709820245708
Longitude
0.0017303702792554887
0.002805043235198257
```

Módulo 16 – Entendendo o coeficiente de determinação

Mostramos os valores do método `score` da classe `LinearRegression`. Esse método fornece os mesmos valores que seriam obtidos utilizando diretamente o método `r2_score` do Scikit-Learn. Ou seja, estamos utilizando o **coeficiente de determinação**, R ao quadrado, como métrica. Veja abaixo que os valores obtidos pelos métodos são idênticos. Observe, também, que um dos valores é negativo, o que significa que o modelo é pior do que se considerássemos a média dos dados:

```
# Criando o modelo para o AveOccup
from sklearn.metrics import r2_score
RegSim4 = LinearRegression().fit(X_train.AveOccup.values.reshape(-1,1), y_train)
```

```
# Avaliando o score
print(RegSim4.score(X_train.AveOccup.values.reshape(-1,1), y_train))
print(RegSim4.score(X_test.AveOccup.values.reshape(-1,1), y_test))
```

```
0.000999915854954958
-0.0037063533253798298
```

```
predict_test = RegSim4.predict(X_test.AveOccup.values.reshape(-1,1))
predict_treino = RegSim4.predict(X_train.AveOccup.values.reshape(-1,1))
```

```
# Usando o r2_score do Scikit-Learn
r2_score(y_train, predict_treino)
```

```
0.000999915854954958
```

```
r2_score(y_test, predict_test)
```

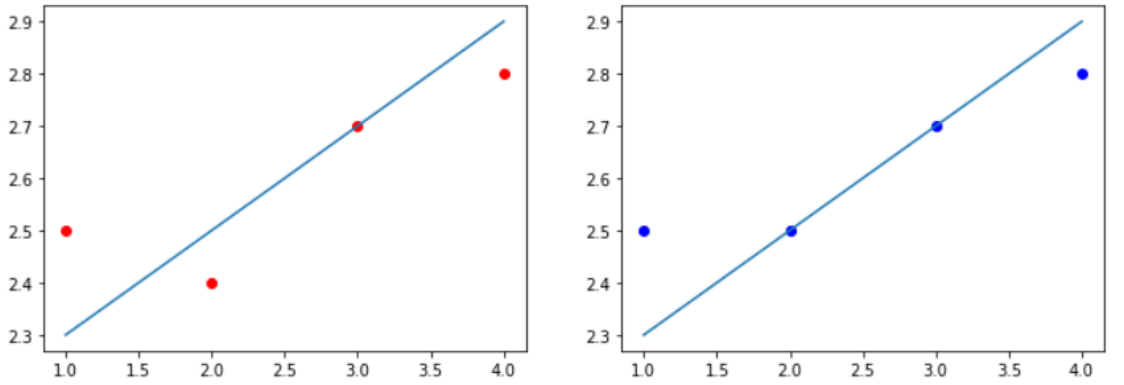
```
-0.0037063533253798298
```

Módulo 16 – Entendendo o coeficiente de determinação

No entanto, não necessariamente esta é a melhor métrica para uma regressão linear. Na página anterior, vimos que o modelo pode até ser pior que a média dos dados. Veja abaixo que criamos dois conjuntos de dados fictícios, sendo a diferença entre eles no segundo ponto apenas. Observe que a segunda reta passa por mais pontos da base, e para os demais pontos as distâncias são as mesmas da primeira reta, mas possui menor valor de coeficiente de determinação.

```
x = [1,2,3,4]
y_true1 = [2.5,2.4,2.7,2.8]
y_true2 = [2.5,2.5,2.7,2.8]
y_pred = [2.3,2.5,2.7,2.9]
```

```
import matplotlib.pyplot as plt
fig,ax = plt.subplots(ncols=2,figsize=(12,4))

ax[0].scatter(x,y_true1,c='r')
ax[0].plot(x,y_pred)
ax[1].scatter(x,y_true2,c='b')
ax[1].plot(x,y_pred)

plt.show()
```




```
# Gerando o r2_score para os 2 conjuntos de dados
print(r2_score(y_true1,y_pred))
print(r2_score(y_true2,y_pred))
```

```
0.399999999999999
0.2592592592592575
```

Módulo 16 – Entendendo o coeficiente de determinação

Como isto é possível? Vejamos a seguir a definição do coeficiente. Repare que no numerador comparamos os dados com os valores previstos, enquanto que no denominador comparamos com a média. Ou seja, a razão envolve uma comparação com o valor médio.

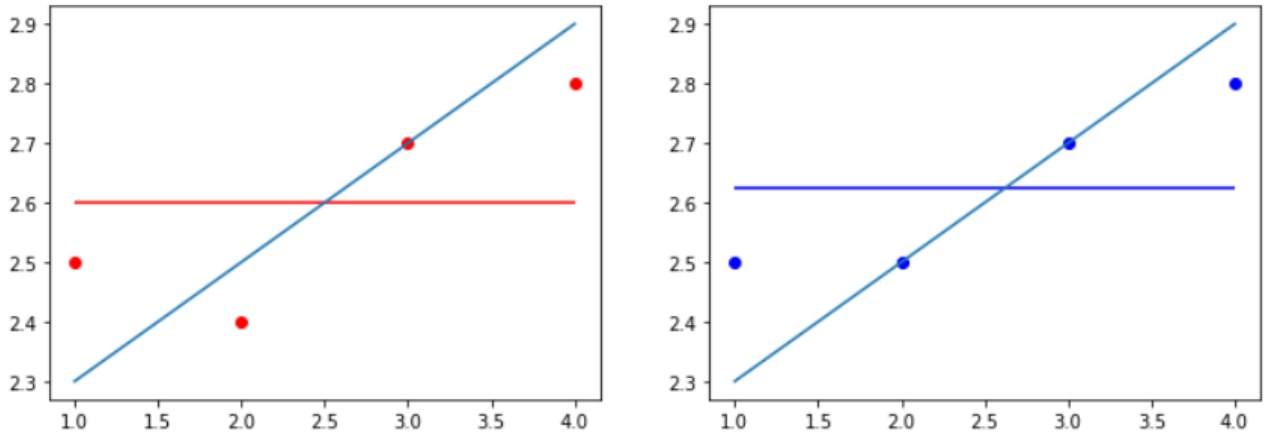
$$R^2 = 1 - \frac{\sum(y - \hat{y})^2}{\sum(y - \bar{y})^2}$$

Valor previsto

Média

Agora faz sentido a primeira reta ter maior coeficiente. Observe que ela melhora significativamente a previsão versus a reta que representa a média (observe, por exemplo, que a distância do segundo ponto para reta é menor que para a média).

```
1  import numpy as np
2  fig,ax = plt.subplots(ncols=2,figsize=(12,4))
3
4  ax[0].scatter(x,y_true1,c='r')
5  ax[0].plot(x,y_pred)
6  ax[0].hlines(y=np.mean(y_true1),xmin=1,xmax=4,color='r')
7  ax[1].scatter(x,y_true2,c='b')
8  ax[1].plot(x,y_pred)
9  ax[1].hlines(y=np.mean(y_true2),xmin=1,xmax=4,color='b')
10
11 plt.show()
```



Módulo 16 – Métricas de erro para regressão

Sabendo que o coeficiente de determinação não é necessariamente a melhor métrica, vejamos o que fazer neste tipo de situação. Primeiro, vamos gerar um novo conjunto de dados de exemplo:

```
1  import pandas as pd
2  dados = {
3      'X': [1,2,3,4],
4      'Y': [5,7,10,11]
5  }
6
7  dados = pd.DataFrame(dados)
8
9  X = dados.X.values.reshape(-1,1)
10 y = dados.Y
```

```
1  dados.shape[0] # Número de linhas da base
```

```
4
```

```
1  X.shape[1] # Número de colunas do modelo
```

```
1
```

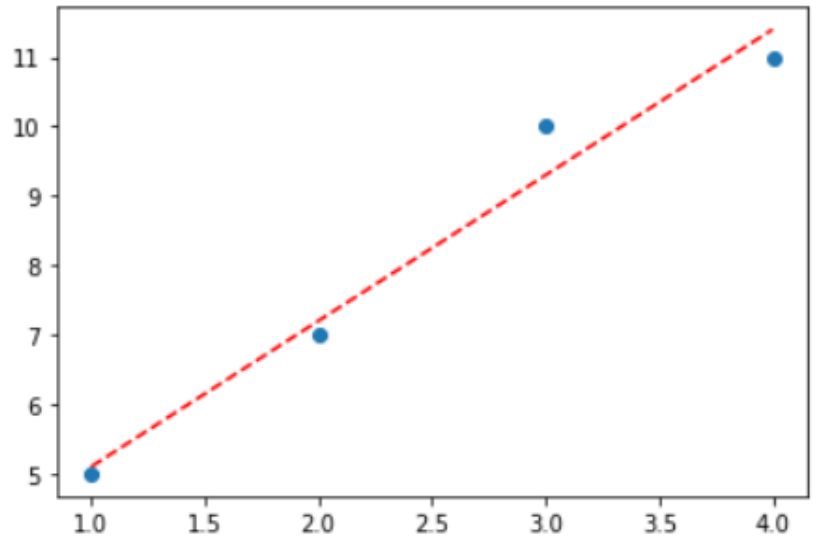
Módulo 16 – Métricas de erro para regressão

Vamos fazer uma regressão no conjunto de dados e visualizar a reta gerada:

```
# Regressão Linear
from sklearn import linear_model
regLinear = linear_model.LinearRegression()
regLinear.fit(X,y)
```

```
LinearRegression()
```

```
# Visualizando essa reta
a=regLinear.coef_[0]
b=regLinear.intercept_
y_reta = a*X+b

import matplotlib.pyplot as plt
fig,ax = plt.subplots()

ax.scatter(X,y)
ax.plot(X,y_reta,'--r')

plt.show()
```




```
y_pred_reg = regLinear.predict(X)
```

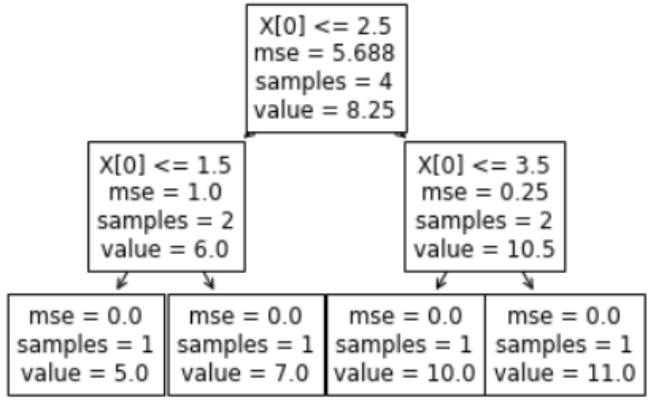
Módulo 16 – Métricas de erro para regressão

Como já estudamos árvores em módulos anteriores, podemos comparar nosso modelo linear com um modelo de árvore de regressão. Desta forma, temos um padrão de comparação. Para este conjunto de exemplo, a árvore gerada é:

```
# Árvore de Decisão
from sklearn.tree import DecisionTreeRegressor
regArvore = DecisionTreeRegressor(random_state=0)
regArvore.fit(X,y)
```

```
DecisionTreeRegressor(random_state=0)
```

```
from sklearn import tree
tree.plot_tree(regArvore);
```




```
y_pred_arvore = regArvore.predict(X)
```

Módulo 16 – Métricas de erro para regressão

Observe que o score da árvore é melhor que o da regressão linear:

```
1  from sklearn.metrics import r2_score
```

```
1  y_true = y
```

```
1  # Regressão Linear
2  r2_score(y_true, y_pred_reg)
```

```
0.9692307692307693
```

```
1  # Árvore de Decisão
2  r2_score(y_true, y_pred_arvore)
```

```
1.0
```

Uma outra métrica é o chamado coeficiente de determinação ajustado. Ele leva em consideração o tamanho da amostra e o número de variáveis utilizadas na previsão:

$$R^2_{Ajustado} = 1 - \frac{\sum(y - \hat{y})^2}{\sum(y - \bar{y})^2} * \frac{n - 1}{n - p - 1}$$

Tamanho da amostra

Número de variáveis preditoras

Módulo 16 – Métricas de erro para regressão

Como a árvore já tinha valor 1 de score, este se mantém. O score do modelo de regressão muda um pouco com esta nova métrica:

```
# Regressão Linear
r2_score_modelo = r2_score(y_true, y_pred_reg)
somatorio = 1 - r2_score_modelo
n = dados.shape[0]
p = X.shape[1]


r2_ajus = 1 - somatorio*((n-1)/(n-p-1))
print(r2_ajus)
```

```
0.953846153846154
```

```
# Árvore de Decisão
r2_score_modelo = r2_score(y_true, y_pred_arvore)
somatorio = 1 - r2_score_modelo
n = dados.shape[0]
p = X.shape[1]


r2_ajus = 1 - somatorio*((n-1)/(n-p-1))
print(r2_ajus)
```

```
1.0
```

Módulo 16 – Métricas de erro para regressão

Existem, ainda, duas métricas muito relevantes para modelos lineares: o erro médio absoluto (EMA), e o erro quadrático médio (EQM). Diferentemente do coeficiente de determinação, eles consideram a distância dos pontos à reta de previsão (e não à média). No caso do EQM, as distâncias são elevadas ao quadrado, penalizando mais outliers presentes na amostra:

$$EMA = \frac{1}{n}\sum|\hat{y} - y|$$

$$EQM = \frac{1}{n}\sum(\hat{y} - y)^2$$

```
1  from sklearn.metrics import mean_absolute_error
```

```
1  # Reg Linear
2  mean_absolute_error(y_true, y_pred_reg)
```

```
0.34999999999999964
```

```
1  # Árvore de Decisão
2  mean_absolute_error(y_true, y_pred_arvore)
```

```
0.0
```

```
1  from sklearn.metrics import mean_squared_error
```

```
1  # Reg Linear
2  mean_squared_error(y_true, y_pred_reg)
```

```
0.1749999999999995
```

```
1  # Árvore de Decisão
2  mean_squared_error(y_true, y_pred_arvore)
```

```
0.0
```

Módulo 16 – Métricas de erro para regressão

Assim como para o coeficiente de determinação, o Scikit-Learn possui métodos para estas duas métricas:

```
1  # Criando um novo modelo de regressão para a coluna 'MedInc'
2  RegSim4 = LinearRegression().fit(X_train.MedInc.values.reshape(-1,1), y_train)
```

```
1  # Fazendo o predict
2  y_pred = RegSim4.predict(X_test.MedInc.values.reshape(-1,1))
```

```
1  # Avaliando o erro médio absoluto dos dados de TESTE
2  from sklearn.metrics import mean_absolute_error
3  mean_absolute_error(y_test, y_pred)
```

```
0.6294060850621533
```

```
1  # Avaliando o erro quadrático médio para os dados de TESTE
2  from sklearn.metrics import mean_squared_error
3  mean_squared_error(y_test,y_pred)
```

```
0.700386804985532
```

Módulo 16 – Métricas de erro para regressão

Seguindo a mesma rotina que fizemos anteriormente, podemos fazer um loop para verificar as métricas para cada um das colunas de nossa base de dados:

```
lista = X_train.columns

for i in lista:
    print(i)
    RegSim4 = LinearRegression().fit(X_train[i].values.reshape(-1,1), y_train)
    y_pred = RegSim4.predict(X_test[i].values.reshape(-1,1))
    print(mean_absolute_error(y_test, y_pred))
    print(mean_squared_error(y_test,y_pred))
```

```
MedInc
0.6294060850621533
0.700386804985532
HouseAge
0.9112472186113978
1.3225424434775808
AveRooms
0.8980334210700883
1.2969733418961416
AveBedrms
0.9164225728049346
1.3358550677054248
Population
0.9184219850918659
1.3410163940887445
AveOccup
0.9186157888878567
1.3467749109300464
Latitude
0.9102081408848934
1.3170163476624213
Longitude
0.9150371204426578
1.3380379078276463
```

Módulo 16 – Regressão linear múltipla

Até o momento, sempre escolhemos colunas individualmente para treinar nosso modelo. Mas, nada impede que sejam escolhidas mais colunas. Nestes casos, dizemos que estamos realizando uma regressão linear múltipla. A organização do código é similar a que vimos anteriormente, bastando passar uma lista com as colunas desejadas:

```
# Visualizando o X_train
X_train.head(2)
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
18368	5.3468	42.0	6.364322	1.087940	957.0	2.404523	37.16	-121.98
19197	3.9191	36.0	6.110063	1.059748	711.0	2.235849	38.45	-122.69

```
# Escolhendo 2 variáveis para fazer a regressão
X_Mult_train = X_train[['AveRooms','MedInc']]
X_Mult_test = X_test[['AveRooms','MedInc']]
```

```
# Fazendo a regressão dessas duas variáveis
regMult = LinearRegression().fit(X_Mult_train, y_train)
y_pred = regMult.predict(X_Mult_test)
```

```
# Avaliando o erro
print(mean_absolute_error(y_test, y_pred))
print(mean_squared_error(y_test,y_pred))
```

```
0.6198538766617373
0.6836236839427631
```

Módulo 16 – Regressão linear múltipla

Com um loop, podemos verificar qual par de variáveis performa melhor:

```python
# Usando o for para testar todos os pares de variáveis
lista = X_train.columns

for i in range(0,len(lista)):
    for j in range(i+1,len(lista)):
        print(lista[i],lista[j])

        X_Mult_train = X_train[[lista[i],lista[j]]]
        X_Mult_test = X_test[[lista[i],lista[j]]]

        regMult = LinearRegression().fit(X_Mult_train, y_train)
        y_pred = regMult.predict(X_Mult_test)

        print(mean_absolute_error(y_test, y_pred))
        print(mean_squared_error(y_test,y_pred))
```

```
MedInc HouseAge
0.6031032736795487
0.6469498044744214
MedInc AveRooms
0.6198538766617373
0.6836236839427631
MedInc AveBedrms
0.6293603474891636
0.7000859500219594
MedInc Population
0.629263237936552
0.6989716091404197
MedInc AveOccup
0.6285791640431908
0.6989059866213563
MedInc Latitude
0.6234988360932905
0.6919213687862203
MedInc Longitude
0.6280224867409706
0.6977953219825376
HouseAge AveRooms
0.8851613422391501
1.2639976421592691
HouseAge AveBedrms
0.9097189234157886
1.3181806604731452
HouseAge Population
0.9110706637950305
1.3225596984147312
HouseAge AveOccup
0.9112283717210554
1.3271932563519058
HouseAge Latitude
0.9036568258128097
1.2966063624435409
HouseAge Longitude
0.9087432485078941
1.3200482235071982
```

E, claro, podemos treinar o modelo com todas as variáveis existentes:

```
1  # Usando todas as variáveis existentes no modelo
2  regMult = LinearRegression().fit(X_train, y_train)
3  y_pred = regMult.predict(X_test)
4
5  print(mean_absolute_error(y_test, y_pred))
6  print(mean_squared_error(y_test,y_pred))
```

```
0.5269635378186492
0.5122113387754117
```

Perceba que os erros são menores do que os vistos anteriormente com variáveis individuais e com pares de variáveis. No entanto, não necessariamente isto é verdade em todos os casos. Em bases de dados muito grandes, treinar o modelo com todas as variáveis pode ter um custo elevado. Nestes casos, faz sentido realizar estudos preliminares buscando selecionar melhor as variáveis. Deve-se também ter cuidado com overfitting ao se utilizar todas as variáveis.

Módulo 16 – Comparando modelos

Para finalizar este módulo, vamos comparar nossa regressão com dois outros modelos. Primeiro, com uma árvore de regressão, utilizando todas as nossas variáveis:

```
# Importando a árvore de regressão
from sklearn import tree
```

```
# Criando o regressor
regArvore = tree.DecisionTreeRegressor(random_state=42)
```

```
# Fazendo o fit
regArvore = regArvore.fit(X_train, y_train)
```

```
# Avaliando o modelo
y_predArvore = regArvore.predict(X_test)

print(mean_absolute_error(y_test, y_predArvore))
print(mean_squared_error(y_test,y_predArvore))
```

```
0.4639818100775194
0.5301621113025969
```

Veja que nosso erro média absoluto é menor que o da regressão linear, mas o erro quadrático médio maior. Como o EQM é mais sensível a outliers, isto pode indicar que há alguns outliers na base que estão diminuindo a performance deste modelo.

Módulo 16 – Comparando modelos

Vamos considerar agora outro modelo, o SVR – *Support Vector Regression*. Veremos detalhes deste modelo em módulos futuros, nosso objetivo no momento é apenas mostrar como é simples modificar o código para treinar um novo modelo. Veja como a estrutura é a mesma, precisamos apenas importar o regressor e fazer as devidas substituições no código:

```
# Importando o SVR
from sklearn import svm
```

```
# Criando o regressor
regSVR = svm.SVR()
```

```
# Fazendo o fit
regSVR.fit(X_train, y_train)
```

```
SVR()
```

```
# Avaliando o modelo
y_predSVR = regSVR.predict(X_test)

print(mean_absolute_error(y_test, y_predSVR))
print(mean_squared_error(y_test,y_predSVR))
```

```
0.8701841836434215
1.36186322768604
```

Observe como os erros foram significativamente maiores que os dos outros modelos. Desta forma, para esta base, os outros modelos são mais adequados.

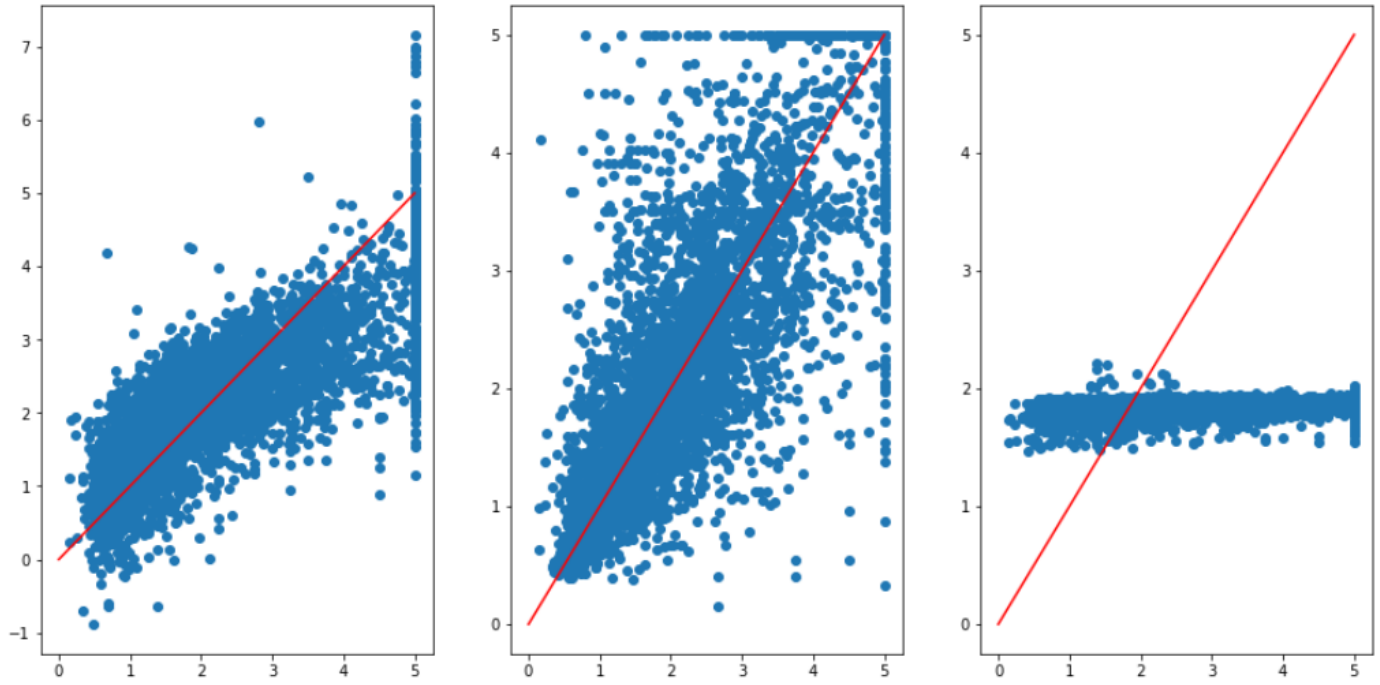
Em módulos futuros, estudaremos o SVR e seu significado.

Módulo 16 – Comparando modelos

Por fim, podemos visualizar as retas de cada modelo. Observe que o SVR realmente possui um pior ajuste que os demais. A árvore possui pontos mais dispersos que a regressão linear, daí o EQM maior da árvore comparada ao modelo de regressão linear.

```
import numpy as np
```

```
fig,ax = plt.subplots(ncols=3,figsize=(16,8))

x = np.arange(0,6)
y = x
ax[0].scatter(y_test,y_pred)
ax[0].plot(x,y,'r')
ax[1].scatter(y_test,y_predArvore)
ax[1].plot(x,y,'r')
ax[2].scatter(y_test,y_predSVR)
ax[2].plot(x,y,'r')

plt.show()
```



MÓDULO 17

Conceitos básicos de SQL para Ciência de Dados

Módulo 17 – O que é SQL?

SQL vem de *Structured Query Language*, ou Linguagem de Consulta Estruturada. É uma linguagem usada para gerenciar e manipular bancos de dados relacionais.

- Permite criar, consultar, modificar e excluir tabelas e dados
- Possibilita filtrar, ordenar e agrupar dados
- Facilita a combinação e comparação de informações entre tabelas
- Funções e operadores para cálculos e manipulação de texto
- Segurança e controle de acesso aos dados
- Amplamente utilizada e suportada por diversos sistemas de gerenciamento de bancos de dados (SGBDs)

Neste módulo, usaremos o banco SQLite por ser mais simples, mas os fundamentos servem para qualquer outra vertente de SQL. A biblioteca sqlite3, padrão de uma distribuição Python, possibilita uma integração com este tipo de banco de dados, de forma que podemos continuar a utilizar o Jupyter Notebook como viemos fazendo no decorrer do curso.

Módulo 17 – Usando SQL com o Pandas

Os primeiros passos para trabalhar com o SQLite são:

- Conectar ao banco de dados
- Criar um cursor

Com o cursor criado, podemos executar comandos SQL. E, com o método `fetchall`, resgatar todos os registros retornados pelo comando.

O comando executado no exemplo é

```
SELECT * FROM dados
```

O asterisco significa todas as colunas. Logo, estamos solicitando literalmente que selecione os registros (linhas) de todas as colunas da tabela `dados`.

Pode parecer confuso, mas o retorno é basicamente uma lista de tuplas. Cada tupla representa uma linha da tabela consultada. E esse retorno pode facilmente ser transformado em um dataframe do Pandas.

```python
# Criando a conexão
import sqlite3
con = sqlite3.connect('BaseDados.db')
```

```python
# Criando um cursor
cur = con.cursor()
```

```python
# Buscando todas as informações da tabela
cur.execute('SELECT * FROM dados').fetchall()
```

```
[(0,
  1,
  'Maria Eduarda da Rocha',
  38273,
  'maria@hashtag.com',
  0,
  1,
  12,
  1,
  1,
  9.0),
 (1, 2, 'Bárbara da Cunha', 63546, 'barbara@hashtag.com', 0, 1, 2, 1, 1, 10.0),
 (2, 3, 'Kevin Melo', 80515, 'kevin@hashtag.com', 1, 1, 3, 1, 1, 7.0),
 (3,
  4,
  'Pedro Henrique da Costa',
  68004,
  'pedro@hashtag.com',
  1,
  1,
  1,
  1,
  1,
  4.0),
 (4, 5, 'Mirella Viana', 28421, 'mirella@hashtag.com', 1, 1, 1, 1, 1, 7.0),
 (5, 6, 'Lívia Jesus', 22284, 'lívia@hashtag.com', 0, 1, 14, 1, 1, 8.0),
 (6, 7, 'Lara Lopes', 38362, 'lara@hashtag.com', 1, 1, 1, 1, 0, None),
 (7, 8, 'Lucca Cardoso', 45109, 'lucca@hashtag.com', 1, 1, 3, 1, 0, None),
 (8, 9, 'Isabelly Souza', 31859, 'isabelly@hashtag.com', 1, 1, 3, 1, 1, 7.0),
 (9, 10, 'Cauã Porto', 42674, 'cauã@hashtag.com', 0, 1, 2, 1, 0, None),
 (10, 11, 'Maria Ferreira', 86563, 'maria@hashtag.com', 0, 1, 1, 1, 1, 7.0),
 (11, 12, 'Júlia Pinto', 47086, 'julia@hashtag.com', 1, 1, 2, 1, 1, 6.0),
```

Módulo 17 – Usando SQL com o Pandas

Armazenando o retorno do comando um variável, **`resultado`**, podemos passar para um DataFrame do Pandas. O objeto cursor possui um atributo, **`description`**, que armazena diversas informações, dentre elas o nome das colunas. Assim, eis as 5 primeiras linhas de nossa tabela do banco de dados na forma de um DataFrame Pandas:

```
# Buscando todas as informações da tabela
cur.execute('SELECT * FROM dados')
resultado = cur.fetchall()
```

```
# Salvando em um DataFrame do pandas
import pandas as pd
resultado = pd.DataFrame(resultado)
resultado.columns = [i[0] for i in cur.description]
resultado.head()
```

	index	id_aluno	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado	dias_ultimo_acesso	nr_prova	prova_feita	nota_prova
0	0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	1	1	9.0
1	1	2	Bárbara da Cunha	63546	barbara@hashtag.com	0	1	2	1	1	10.0
2	2	3	Kevin Melo	80515	kevin@hashtag.com	1	1	3	1	1	7.0
3	3	4	Pedro Henrique da Costa	68004	pedro@hashtag.com	1	1	1	1	1	4.0
4	4	5	Mirella Viana	28421	mirella@hashtag.com	1	1	1	1	1	7.0

Esta é uma tabela ***fictícia*** de alunos da Hashtag. Apresenta uma ID, o nome, a matrícula, e o e-mail. Após, duas colunas representando se o aluno tem acesso à plataforma e se este está liberado, onde 0 é “não” e 1 é “sim”. Então, uma coluna contando os dias desde o último acesso do aluno. Por fim, colunas sobre provas com o número da prova feita (1, 2, 3, ...), se a prova foi feita ou não (0 “não” ou 1 “sim”) e a nota da prova.

Módulo 17 – Usando SQL com o Pandas

Como de costume, de nossos outros módulos, podemos olhar também os últimos 5 registros, para se certificar que toda a base de dados foi obtida com sucesso. E olhar o atributo **shape** para se certificar do tamanho da base:

```
1  # Visualizando os resultados
2  resultado.tail()
```

	index	id_aluno	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado	dias_ultimo_acesso	nr_prova	prova_feita	nota_prova
38	38	18	Bárbara Freitas	19442	barbara@hashtag.com	0	1	12	3	0	NaN
39	39	19	Melissa Ribeiro	38438	melissa@hashtag.com	1	1	3	3	0	NaN
40	40	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	4	0	NaN
41	41	16	Eloah Aragão	65749	eloah@hashtag.com	1	1	3	4	1	7.0
42	42	16	Eloah Aragão	65749	eloah@hashtag.com	1	1	3	5	0	NaN

```
1  # Obtendo o tamanho da base
2  resultado.shape
```

```
(43, 11)
```

Módulo 17 – Selecionando colunas específicas com SELECT

Ao invés de trazermos toda a base, podemos passar ao SELECT apenas as colunas desejadas. Por exemplo:

```
# Selecionando apenas as colunas nome_aluno e cod_matricula
cur.execute('SELECT nome_aluno,cod_matricula FROM dados')
resultado = cur.fetchall()
resultado = pd.DataFrame(resultado)
resultado.columns = [i[0] for i in cur.description]
resultado.head()
```

	nome_aluno	cod_matricula
0	Maria Eduarda da Rocha	38273
1	Bárbara da Cunha	63546
2	Kevin Melo	80515
3	Pedro Henrique da Costa	68004
4	Mirella Viana	28421

Na prática, quando lidamos com bases de dados muito extensas, este é o procedimento mais usual.

Módulo 17 – Selecionando colunas específicas com SELECT

Algumas vezes o nome de uma dada coluna pode ser muito longo ou conter caracteres que causam problemas com demais comandos a serem utilizados. Assim, podemos utilizar o comando `AS` para renomear colunas. No exemplo, renomeamos `e-mail` para `email`, retirando o hífen que é inconveniente no código:

```
1  # Selecionando apenas as colunas nome_aluno e e-mail
2  comando = 'SELECT nome_aluno,[e-mail] as email FROM dados'
3  cur.execute(comando)
4  resultado = cur.fetchall()
5  resultado = pd.DataFrame(resultado)
6  resultado.columns = [i[0] for i in cur.description]
7  resultado.head()
```

	nome_aluno	email
0	Maria Eduarda da Rocha	maria@hashtag.com
1	Bárbara da Cunha	barbara@hashtag.com
2	Kevin Melo	kevin@hashtag.com
3	Pedro Henrique da Costa	pedro@hashtag.com
4	Mirella Viana	mirella@hashtag.com

Módulo 17 – Criando uma função para nossos estudos de SQL com Pandas

Observe nas páginas anteriores que estamos sempre usando a mesma estrutura de criar um comando e gerar um dataframe, exibindo as 5 primeiras linhas deste dataframe. Para evitar mais repetição, podemos encapsular esta lógica em uma função. Tal função recebe o comando SQL, exibe as 5 primeiras linhas de um dataframe criado a partir do comando, assim como o atributo **shape** deste, e retorna o resultado completo caso este queira ser utilizado pelo usuário:

```
# Criando uma função para consultar os dados
def executa_sql(comando):
    cur.execute(comando)
    resultado = cur.fetchall()
    resultado = pd.DataFrame(resultado)
    resultado.columns = [i[0] for i in cur.description]
    print(resultado.shape)
    display(resultado.head())
    return resultado
```

Módulo 17 – Criando uma função para nossos estudos de SQL com Pandas

Assim, podemos repetir nosso último exemplo passando o comando SQL para nossa função:

```
1  resultado = executa_sql('SELECT nome_aluno,[e-mail] as email FROM dados')
```

```
(43, 2)
```

	nome_aluno	email
0	Maria Eduarda da Rocha	maria@hashtag.com
1	Bárbara da Cunha	barbara@hashtag.com
2	Kevin Melo	kevin@hashtag.com
3	Pedro Henrique da Costa	pedro@hashtag.com
4	Mirella Viana	mirella@hashtag.com

Temos a informação de que o comando retorna 43 linhas e 2 colunas no total. E exibimos os 5 primeiros resultados.

Utilizaremos esta função no restante do módulo para focarmos nos comandos SQL apenas.

Módulo 17 – Criando uma função para nossos estudos de SQL com Pandas

A função exibe as 5 primeiras linhas, mas retorna o dataframe completo. Assim, se armazenarmos o resultado em uma variável, `resultado` no exemplo, podemos interagir com o dataframe. Abaixo, solicitamos com o método `tail` os 5 últimos registros do dataframe criado a partir de todas as colunas de nossa base de dados armazenado na variável `resultado`.

```
1 resultado = executa_sql('SELECT * FROM dados')
```

(43, 11)

	index	id_aluno	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado	dias_ultimo_acesso	nr_prova	prova_feita	nota_prova
0	0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	1	1	9.0
1	1	2	Bárbara da Cunha	63546	barbara@hashtag.com	0	1	2	1	1	10.0
2	2	3	Kevin Melo	80515	kevin@hashtag.com	1	1	3	1	1	7.0
3	3	4	Pedro Henrique da Costa	68004	pedro@hashtag.com	1	1	1	1	1	4.0
4	4	5	Mirella Viana	28421	mirella@hashtag.com	1	1	1	1	1	7.0

```
1 resultado.tail()
```

	index	id_aluno	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado	dias_ultimo_acesso	nr_prova	prova_feita	nota_prova
38	38	18	Bárbara Freitas	19442	barbara@hashtag.com	0	1	12	3	0	NaN
39	39	19	Melissa Ribeiro	38438	melissa@hashtag.com	1	1	3	3	0	NaN
40	40	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	4	0	NaN
41	41	16	Eloah Aragão	65749	eloah@hashtag.com	1	1	3	4	1	7.0
42	42	16	Eloah Aragão	65749	eloah@hashtag.com	1	1	3	5	0	NaN

Módulo 17 – Lidando com valores repetidos com DISTINCT

Lembrando de nossos módulos anteriores, um dos primeiros cuidados que devemos ter é verificar se há dados duplicados em nossa base. No Pandas, utilizávamos o método `drop_duplicates`:

```
# Selecionando apenas valores distindos da base completa utilizando o pandas (drop_duplicates)
resultado.drop_duplicates()
```

	index	id_aluno	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado	dias_ultimo_acesso	nr_prova	prova_feita	nota_prova
0	0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	1	1	9.0
1	1	2	Bárbara da Cunha	63546	barbara@hashtag.com	0	1	2	1	1	10.0
2	2	3	Kevin Melo	80515	kevin@hashtag.com	1	1	3	1	1	7.0
3	3	4	Pedro Henrique da Costa	68004	pedro@hashtag.com	1	1	1	1	1	4.0
4	4	5	Mirella Viana	28421	mirella@hashtag.com	1	1	1	1	1	7.0
5	5	6	Lívia Jesus	22284	lívia@hashtag.com	0	1	14	1	1	8.0
6	6	7	Lara Lopes	38362	lara@hashtag.com	1	1	1	1	0	NaN
7	7	8	Lucca Cardoso	45109	lucca@hashtag.com	1	1	3	1	0	NaN
8	8	9	Isabelly Souza	31859	isabelly@hashtag.com	1	1	3	1	1	7.0
9	9	10	Cauã Porto	42674	cauã@hashtag.com	0	1	2	1	0	NaN
10	10	11	Maria Ferreira	86563	maria@hashtag.com	0	1	1	1	1	7.0
11	11	12	Júlia Pinto	47086	julia@hashtag.com	1	1	2	1	1	6.0
12	12	13	Antônio Azevedo	29022	lovedogs@hashtag.com	0	1	14	1	1	6.0
13	13	14	Laura Melo	50966	laura@hashtag.com	1	1	2	1	1	6.0
14	14	15	Gabriela Costela	21262	gabriela@hashtag.com	1	1	2	1	1	8.0
15	15	16	Eloah Aragão	65749	eloah@hashtag.com	1	1	3	1	1	8.0
16	16	17	Francisco Pires	59518	x!c0@hashtag.com	0	1	13	1	0	NaN
17	17	18	Bárbara Freitas	19442	barbara@hashtag.com	0	1	12	1	1	6.0
18	18	19	Melissa Ribeiro	38438	melissa@hashtag.com	1	1	3	1	1	9.0
19	19	20	Lavínia Carvalho	89428	lavinia@hashtag.com	1	1	1	1	0	NaN
20	20	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	2	1	8.0

Repare acima que ainda aparecem nomes e matrículas repetidos. Isto porque um mesmo aluno pode ter feito mais de uma prova, como denotado na coluna *nr_prova*.

Módulo 17 – Lidando com valores repetidos com DISTINCT

Por exemplo, vemos abaixo o registro para um estudante específico:

```
1  resultado[resultado.nome_aluno == 'Eloah Aragão']
```

	index	id_aluno	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado	dias_ultimo_acesso	nr_prova	prova_feita	nota_prova
15	15	16	Eloah Aragão	65749	eloah@hashtag.com	1	1	3	1	1	8.0
31	31	16	Eloah Aragão	65749	eloah@hashtag.com	1	1	3	2	1	6.0
37	37	16	Eloah Aragão	65749	eloah@hashtag.com	1	1	3	3	1	9.0
41	41	16	Eloah Aragão	65749	eloah@hashtag.com	1	1	3	4	1	7.0
42	42	16	Eloah Aragão	65749	eloah@hashtag.com	1	1	3	5	0	NaN

Se quisermos considerar apenas o nome e matrícula como critério para descartar duplicatas, no Pandas fazemos:

```
1  # Selecionando apenas as colunas nome_aluno e cod_matricula e selecionando
2  # valores distintos com o pandas
3  resultado = resultado.loc[:,['nome_aluno','cod_matricula']]
4  resultado = resultado.drop_duplicates()
5  resultado.shape
```

```
(20, 2)
```

Módulo 17 – Lidando com valores repetidos com DISTINCT

A lógica pode ser facilmente reproduzida com SQL. Basta passarmos o comando `DISTINCT` após `SELECT` e antes dos nomes das colunas. Assim, expressamos o interesse em selecionar apenas os dados distintos entre si das referidas colunas, o que descarta repetições:

```
1  # Selecionando valores distintos utilizando o SQL
2  resultado = executa_sql('SELECT DISTINCT nome_aluno,cod_matricula FROM dados')
```

```
(20, 2)
```

	nome_aluno	cod_matricula
0	Maria Eduarda da Rocha	38273
1	Bárbara da Cunha	63546
2	Kevin Melo	80515
3	Pedro Henrique da Costa	68004
4	Mirella Viana	28421

Módulo 17 – Filtros com WHERE e os operadores OR, AND, e NOT

Vamos novamente pegar toda a base:

```
1  # Filtrando apenas alunos com problema de acesso usando o pandas
2  resultado = executa_sql('SELECT * FROM dados')
```

```
(43, 11)
```

	index	id_aluno	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado	dias_ultimo_acesso	nr_prova	prova_feita	nota_prova
0	0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	1	1	9.0
1	1	2	Bárbara da Cunha	63546	barbara@hashtag.com	0	1	2	1	1	10.0
2	2	3	Kevin Melo	80515	kevin@hashtag.com	1	1	3	1	1	7.0
3	3	4	Pedro Henrique da Costa	68004	pedro@hashtag.com	1	1	1	1	1	4.0
4	4	5	Mirella Viana	28421	mirella@hashtag.com	1	1	1	1	1	7.0

Por vezes, queremos selecionar parte dos dados a partir de um filtro. Por exemplo, com o Pandas, se quiséssemos selecionar apenas os alunos sem acesso à plataforma, faríamos:

```
1  resultado = resultado[resultado.acesso_plataforma == 0]
2  resultado.shape
```

```
(18, 11)
```

Módulo 17 – Filtros com WHERE e os operadores OR, AND, e NOT

Com o SQL, temos o comando `WHERE`:

```
1  # Fazendo esse mesmo filtro utilizando o WHERE do SQL
2  resultado = executa_sql('SELECT * FROM dados WHERE acesso_plataforma = 0')
```

```
(18, 11)
```

	index	id_aluno	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado	dias_ultimo_acesso	nr_prova	prova_feita	nota_prova
0	0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	1	1	9.0
1	1	2	Bárbara da Cunha	63546	barbara@hashtag.com	0	1	2	1	1	10.0
2	5	6	Lívia Jesus	22284	lívia@hashtag.com	0	1	14	1	1	8.0
3	9	10	Cauã Porto	42674	cauã@hashtag.com	0	1	2	1	0	NaN
4	10	11	Maria Ferreira	86563	maria@hashtag.com	0	1	1	1	1	7.0

Módulo 17 – Filtros com WHERE e os operadores OR, AND, e NOT

Podemos restringir ainda mais nosso filitro com o operador **AND**. No caso, além de o aluno não ter mais acesso à plataforma, queremos que o último acesso tenha sido até 2 dias atrás:

```
1  # Verificando se existe algum aluno com problema de acesso E que entrou nos últimos 2 dias
2  resultado = executa_sql('SELECT DISTINCT nome_aluno, cod_matricula, [e-mail] as email FROM dados \
3                           WHERE acesso_plataforma = 0 AND dias_ultimo_acesso <= 2')
```

```
(3, 3)
```

	nome_aluno	cod_matricula	email
0	Bárbara da Cunha	63546	barbara@hashtag.com
1	Cauã Porto	42674	cauã@hashtag.com
2	Maria Ferreira	86563	maria@hashtag.com

Módulo 17 – Filtros com WHERE e os operadores OR, AND, e NOT

Também podemos ter filtros com alternativas usando o operador **OR**. Abaixo, selecionamos alunos sem acesso à plataforma ou que tenham acessado a mesma há mais de 10 dias:

```
# Verificando problema de acesso OU acesso acima de 10 dias
resultado = executa_sql('SELECT DISTINCT nome_aluno, cod_matricula, [e-mail]  as email, \
                         acesso_plataforma, dias_ultimo_acesso FROM dados \
                         WHERE acesso_plataforma = 0 OR dias_ultimo_acesso > 10')
```

```
(8, 5)
```

	nome_aluno	cod_matricula	email	acesso_plataforma	dias_ultimo_acesso
0	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	12
1	Bárbara da Cunha	63546	barbara@hashtag.com	0	2
2	Lívia Jesus	22284	lívia@hashtag.com	0	14
3	Cauã Porto	42674	cauã@hashtag.com	0	2
4	Maria Ferreira	86563	maria@hashtag.com	0	1

Módulo 17 – Filtros com WHERE e os operadores OR, AND, e NOT

Também podemos querer o contrário de um filtro, ou seja, sua negação. Para isso serve o operador `NOT`. Observe abaixo a comparação entre os dois comandos SQL. No primeiro, selecionamos registros com 10 dias ou mais do último acesso. No segundo, negamos o filtro, o que retorna apenas registros com menos de 10 dias do último acesso:

```
# Buscando alunos com mais de 10 dias de acesso
resultado = executa_sql('SELECT DISTINCT nome_aluno, cod_matricula, [e-mail]  as email, \
                         acesso_plataforma, dias_ultimo_acesso FROM dados \
                         WHERE dias_ultimo_acesso >= 10')
```

(5, 5)

	nome_aluno	cod_matricula	email	acesso_plataforma	dias_ultimo_acesso
0	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	12
1	Lívia Jesus	22284	lívia@hashtag.com	0	14
2	Antônio Azevedo	29022	lovedogs@hashtag.com	0	14
3	Francisco Pires	59518	x!c0@hashtag.com	0	13
4	Bárbara Freitas	19442	barbara@hashtag.com	0	12

```
# Negando essa condição acima com NOT
resultado = executa_sql('SELECT DISTINCT nome_aluno, cod_matricula, [e-mail]  as email, \
                         acesso_plataforma, dias_ultimo_acesso FROM dados \
                         WHERE NOT dias_ultimo_acesso >= 10')
```

(15, 5)

	nome_aluno	cod_matricula	email	acesso_plataforma	dias_ultimo_acesso
0	Bárbara da Cunha	63546	barbara@hashtag.com	0	2
1	Kevin Melo	80515	kevin@hashtag.com	1	3
2	Pedro Henrique da Costa	68004	pedro@hashtag.com	1	1
3	Mirella Viana	28421	mirella@hashtag.com	1	1
4	Lara Lopes	38362	lara@hashtag.com	1	1

Módulo 17 – Filtros com WHERE e os operadores OR, AND, e NOT

Neste caso específico, também poderíamos ter feito um comando utilizando o sinal de menor e passando o valor 10:

```
1 resultado = executa_sql('SELECT DISTINCT nome_aluno, cod_matricula, [e-mail]  as email, \
2                          acesso_plataforma, dias_ultimo_acesso FROM dados \
3                          WHERE dias_ultimo_acesso < 10')
```

```
(15, 5)
```

	nome_aluno	cod_matricula	email	acesso_plataforma	dias_ultimo_acesso
0	Bárbara da Cunha	63546	barbara@hashtag.com	0	2
1	Kevin Melo	80515	kevin@hashtag.com	1	3
2	Pedro Henrique da Costa	68004	pedro@hashtag.com	1	1
3	Mirella Viana	28421	mirella@hashtag.com	1	1
4	Lara Lopes	38362	lara@hashtag.com	1	1

Módulo 17 – Filtros com WHERE e os operadores OR, AND, e NOT

Vamos praticar mais um pouco a seleção com filtros e operadores. Vamos buscar os alunos que fizeram a prova 4:

```
# Filtrando apenas os alunos que FIZERAM a 4 prova
resultado_sql = executa_sql('SELECT nome_aluno, cod_matricula, nr_prova, prova_feita, nota_prova \
                             FROM dados WHERE nr_prova = 4 AND prova_feita = 1')
```

```
(1, 5)
```

	nome_aluno	cod_matricula	nr_prova	prova_feita	nota_prova
0	Eloah Aragão	65749	4	1	7.0

Agora, alunos que fizeram as provas 3 ou 4. Observe os parênteses utilizados, para deixar claro que as provas são alternativas, mas filtro de que a prova, seja a 3 ou a 4, tenha sido feita é uma restrição:

```
# Filtrando alunos que fizeram a 3 ou 4 prova
resultado_sql = executa_sql('SELECT nome_aluno, cod_matricula, nr_prova, prova_feita, nota_prova \
                             FROM dados WHERE (nr_prova = 4 OR nr_prova = 3) AND prova_feita = 1')
```

```
(3, 5)
```

	nome_aluno	cod_matricula	nr_prova	prova_feita	nota_prova
0	Maria Eduarda da Rocha	38273	3	1	6.0
1	Eloah Aragão	65749	3	1	9.0
2	Eloah Aragão	65749	4	1	7.0

Módulo 17 – Filtros com WHERE e os operadores OR, AND, e NOT

Por fim, vejamos alunos que tiveram nota maior que 5 na terceira prova:

```
1  # Filtrando os alunos com nota maior que 5 na terceira prova
2  resultado_sql = executa_sql('SELECT nome_aluno, cod_matricula, nr_prova, prova_feita, nota_prova \
3                               FROM dados WHERE nr_prova = 3 AND nota_prova > 5')
```

```
(2, 5)
```

	nome_aluno	cod_matricula	nr_prova	prova_feita	nota_prova
0	Maria Eduarda da Rocha	38273	3	1	6.0
1	Eloah Aragão	65749	3	1	9.0

Acrescentando o operador NOT, teríamos os com nota menor do que 5. No caso, nenhum aluno.

```
1  # Algum aluno tirou menos de 5 na terceira prova?
2  resultado_sql = executa_sql('SELECT nome_aluno, cod_matricula, nr_prova, prova_feita, nota_prova \
3                               FROM dados WHERE nr_prova = 3 AND NOT nota_prova > 5')
```

Módulo 17 – Funções de agregação

Vamos olhar nossa base novamente, especificamente para a coluna de dias do última acesso:

```
# Visualizando a base
resultado_sql = executa_sql('SELECT * FROM dados')
```

(43, 11)

	index	id_aluno	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado	dias_ultimo_acesso	nr_prova	prova_feita	nota_prova
0	0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	1	1	9.0
1	1	2	Bárbara da Cunha	63546	barbara@hashtag.com	0	1	2	1	1	10.0
2	2	3	Kevin Melo	80515	kevin@hashtag.com	1	1	3	1	1	7.0
3	3	4	Pedro Henrique da Costa	68004	pedro@hashtag.com	1	1	1	1	1	4.0
4	4	5	Mirella Viana	28421	mirella@hashtag.com	1	1	1	1	1	7.0

Por vezes, precisamos de um estudo quantitativo de comportamento de usuários de uma determinada plataforma. Podemos estar interessados em verificar se os mesmos estão acessando a plataforma recorrentemente. Para isso, podemos utilizar funções de agregação, para verificar o valor mínimo, máximo, a média, dentre outras métricas. No nosso exemplo, vemos que o aluno que está a mais dias sem acessar não acessa a plataforma há 14 dias. E a média é de aproximadamente 5 dias:

```
# Verificando o valor MÍNIMO da coluna dias_ultimo_acesso
resultado_sql = executa_sql('SELECT MIN(dias_ultimo_acesso) as minimo FROM dados')
```

(1, 1)

	minimo
0	1

```
# Verificando o valor MÍNIMO da coluna dias_ultimo_acesso
resultado_sql = executa_sql('SELECT MAX(dias_ultimo_acesso) as maximo FROM dados')
```

(1, 1)

	maximo
0	14

```
# Verificando a MÉDIA da coluna dias_ultimo_acesso
resultado_sql = executa_sql('SELECT AVG(dias_ultimo_acesso) as media FROM dados')
```

(1, 1)

	media
0	5.116279

Obviamente, podemos ter todas as funções utilizadas num mesmo retorno, como colunas distintas:

```
resultado_sql = executa_sql('SELECT AVG(dias_ultimo_acesso) as media, \
                                    MIN(dias_ultimo_acesso) as minimo, \
                                    MAX(dias_ultimo_acesso) as maximo \
                             FROM dados')
```

(1, 3)

	media	minimo	maximo
0	5.116279	1	14

Módulo 17 – Funções de agregação

No Pandas, para fazer agrupamentos, podemos utilizar o método **groupby**. Por exemplo, agrupando por aluno, contando quantas vezes cada aluno aparece na base de dados com a função de agregação **count**:

```
1  # Relembrando o groupby no pandas para verificar quantas vezes cada aluno aparece
2  resultado_sql.groupby('nome_aluno')['id_aluno'].count()
```

```
nome_aluno
Antônio Azevedo           2
Bárbara Freitas           3
Bárbara da Cunha          2
Cauã Porto                1
Eloah Aragão              5
Francisco Pires           1
Gabriela Costela          2
Isabelly Souza            2
Júlia Pinto               3
Kevin Melo                2
Lara Lopes                1
Laura Melo                2
Lavínia Carvalho          1
Lucca Cardoso             1
Lívia Jesus               2
Maria Eduarda da Rocha    4
Maria Ferreira            3
Melissa Ribeiro           3
Mirella Viana             2
Pedro Henrique da Costa   1
Name: id_aluno, dtype: int64
```

Módulo 17 – Funções de agregação

No SQL é bastante similar. Passamos a instrução com a função agregadora e, após, o comando `GROUP BY` com o nome da coluna:

```
1  # Fazendo o group by no SQL para contar a quantidade de linhas por alunos
2  resultado_sql = executa_sql('SELECT nome_aluno, COUNT(id_aluno) as registros FROM dados GROUP BY nome_aluno')
```

```
(20, 2)
```

	nome_aluno	registros
0	Antônio Azevedo	2
1	Bárbara Freitas	3
2	Bárbara da Cunha	2
3	Cauã Porto	1
4	Eloah Aragão	5

Módulo 17 – Funções de agregação

O uso de filtros também é possível. Veja, no exemplo, que estamos solicitando a média das provas com a condição de que exista uma nota (`IS NOT NULL` significa pegar apenas valores não nulos, ou seja, deve haver uma nota). Lembre-se que, em programação, zero e nulo são coisas distintas.

```
1  # Fazendo o group by para calcular a média das provas
2  resultado_sql = executa_sql('SELECT nome_aluno, AVG(nota_prova) as media FROM dados \
3                               where nota_prova is not null \
4                               GROUP BY nome_aluno')
```

```
(15, 2)
```

	nome_aluno	media
0	Antônio Azevedo	6.0
1	Bárbara Freitas	8.0
2	Bárbara da Cunha	10.0
3	Eloah Aragão	7.5
4	Gabriela Costela	8.0

Módulo 17 – Ordenando registros com ORDER BY

Seria interessante poder ordenar os resultados. Para isso, podemos utilizar o comando `ORDER BY` seguido do que queremos ordenar e de que forma. No caso, queremos ordenar a média de notas `AVG(nota_prova)` de forma decrescente `DESC`. Se quiséssemos crescente, utilizaríamos o comando `ASC`.

```
1  # Ordenando as notas da maior para a menor
2  resultado_sql = executa_sql('SELECT nome_aluno, AVG(nota_prova) as media FROM dados \
3                               where nota_prova is not null \
4                               GROUP BY nome_aluno \
5                               ORDER BY AVG(nota_prova) desc')
```

```
(15, 2)
```

	nome_aluno	media
0	Bárbara da Cunha	10.0
1	Melissa Ribeiro	8.0
2	Lívia Jesus	8.0
3	Júlia Pinto	8.0
4	Gabriela Costela	8.0

Módulo 17 – Limitando a base com LIMIT

Observe na página anterior que há 15 registros no total. Poderíamos limitar nosso retorno aos 5 primeiros com o comando `LIMIT 5`:

```
resultado_sql = executa_sql('SELECT nome_aluno, cod_matricula, AVG(nota_prova) as media_prova, \
                             count(nota_prova) as qtd_provas FROM dados \
                             WHERE nota_prova IS NOT NULL \
                             GROUP BY nome_aluno,cod_matricula \
                             ORDER BY AVG(nota_prova) desc \
                             LIMIT 5')
```

(5, 4)

	nome_aluno	cod_matricula	media_prova	qtd_provas
0	Bárbara da Cunha	63546	10.0	1
1	Bárbara Freitas	19442	8.0	2
2	Gabriela Costela	21262	8.0	1
3	Júlia Pinto	47086	8.0	2
4	Lívia Jesus	22284	8.0	1

Módulo 17 – Filtros com HAVING

Vejamos o que ocorre quando tentamos realizar uma agregação utilizando `WHERE` na mesma query:

```
1  # Utilizando o WHERE para selecionar apenas alunos com média maior que 5
2  resultado_sql = executa_sql('SELECT nome_aluno, cod_matricula, AVG(nota_prova) as media_prova, \
3                               count(nota_prova) as qtd_provas FROM dados \
4                               WHERE nota_prova IS NOT NULL AND AVG(nota_prova) > 5 \
5                               GROUP BY nome_aluno,cod_matricula \
6                               ORDER BY AVG(nota_prova) desc \
7                               LIMIT 5')
```

```
---------------------------------------------------------------------------
OperationalError                          Traceback (most recent call last)
Cell In[13], line 2
      1 # Utilizando o WHERE para selecionar apenas alunos com média maior que 5
----> 2 resultado_sql = executa_sql('SELECT nome_aluno, cod_matricula, AVG(nota_prova) as media_prova, \
      3                               count(nota_prova) as qtd_provas FROM dados \
      4                               WHERE nota_prova IS NOT NULL AND AVG(nota_prova) > 5 \
      5                               GROUP BY nome_aluno,cod_matricula \
      6                               ORDER BY AVG(nota_prova) desc \
      7                               LIMIT 5')

Cell In[4], line 3, in executa_sql(comando)
      2 def executa_sql(comando):
----> 3     cur.execute(comando)
      4     resultado = cur.fetchall()
      5     resultado = pd.DataFrame(resultado)

OperationalError: misuse of aggregate: AVG()
```

Módulo 17 – Filtros com HAVING

Isto ocorre pois há uma ordem de execução dos comandos em um query SQL:

Observe que o `WHERE` é avaliado antes de agrupamentos. Para resolver o erro, precisamos utilizar o comando `HAVING`:

```
# Trocando o WHERE pelo HAVING
resultado_sql = executa_sql('SELECT nome_aluno, cod_matricula, AVG(nota_prova) as media_prova, \
                             count(nota_prova) as qtd_provas FROM dados \
                             WHERE nota_prova IS NOT NULL \
                             GROUP BY nome_aluno,cod_matricula \
                             HAVING AVG(nota_prova) > 7 \
                             ORDER BY AVG(nota_prova) desc')
```

(8, 4)

	nome_aluno	cod_matricula	media_prova	qtd_provas
0	Bárbara da Cunha	63546	10.0	1
1	Bárbara Freitas	19442	8.0	2
2	Gabriela Costela	21262	8.0	1
3	Júlia Pinto	47086	8.0	2
4	Lívia Jesus	22284	8.0	1

Módulo 17 – Filtros com HAVING

Podemos visualizar todos os registros para nos certificar que o filtro foi corretamente aplicado:

```
1  display(resultado_sql)
```

	nome_aluno	cod_matricula	media_prova	qtd_provas
0	Bárbara da Cunha	63546	10.000000	1
1	Bárbara Freitas	19442	8.000000	2
2	Gabriela Costela	21262	8.000000	1
3	Júlia Pinto	47086	8.000000	2
4	Lívia Jesus	22284	8.000000	1
5	Melissa Ribeiro	38438	8.000000	2
6	Maria Eduarda da Rocha	38273	7.666667	3
7	Eloah Aragão	65749	7.500000	4

Módulo 17 – Condicionais com CASE

Assim como em Python, com IF/ELIF/ELSE, também é possível construir estruturas condicionais em SQL com cláusulas `CASE`. A primeira condição que for verdadeira será retornada. Cada condição tem o formato

```
WHEN <condição> THEN <valor>
```

Após as condições, podemos colocar um `ELSE` caso nenhuma das condições anteriores seja verdadeira. E precisamos terminar um `CASE` com `END`. Vamos definir a seguinte regra para aplicar na nossa base de dados:

- Nota > 7: aprovado
- Nota > 5 e qtd_provas = 1: fazer prova 2
- Nota > 5 e qtd_provas = 2: fazer prova final
- Nota > 5 e qtd_provas > 2: revisar matéria
- Nota < 5 e qtd_provas = 1: revisar matéria
- Nota < 5 e qtd_provas > 1: lista de exercícios

Módulo 17 – Condicionais com CASE

Para começar, vamos implementar a primeira condição:

```
# Vamos utilizar o CASE para colocar apenas a primeira condição
resultado_sql = executa_sql('SELECT nome_aluno, cod_matricula, \
                             AVG(nota_prova) as media_prova, \
                             count(nota_prova) as qtd_provas, \
                             (CASE \
                                 WHEN AVG(nota_prova) > 7 THEN "Aprovado" \
                             END) as situacao_aluno \
                             FROM dados \
                             WHERE nota_prova IS NOT NULL \
                             GROUP BY nome_aluno,cod_matricula \
                             ORDER BY AVG(nota_prova) desc')
```

```
(15, 5)
```

	nome_aluno	cod_matricula	media_prova	qtd_provas	situacao_aluno
0	Bárbara da Cunha	63546	10.0	1	Aprovado
1	Bárbara Freitas	19442	8.0	2	Aprovado
2	Gabriela Costela	21262	8.0	1	Aprovado
3	Júlia Pinto	47086	8.0	2	Aprovado
4	Lívia Jesus	22284	8.0	1	Aprovado

Módulo 17 – Condicionais com CASE

Visualizando todo o retorno, vemos que os casos não verdadeiros para nossa condição ficaram com o valor *None*, já que não especificamos o valor desejado:

```
1  # Visualizando o resultado
2  display(resultado_sql)
```

	nome_aluno	cod_matricula	media_prova	qtd_provas	situacao_aluno
0	Bárbara da Cunha	63546	10.000000	1	Aprovado
1	Bárbara Freitas	19442	8.000000	2	Aprovado
2	Gabriela Costela	21262	8.000000	1	Aprovado
3	Júlia Pinto	47086	8.000000	2	Aprovado
4	Lívia Jesus	22284	8.000000	1	Aprovado
5	Melissa Ribeiro	38438	8.000000	2	Aprovado
6	Maria Eduarda da Rocha	38273	7.666667	3	Aprovado
7	Eloah Aragão	65749	7.500000	4	Aprovado
8	Isabelly Souza	31859	7.000000	1	None
9	Kevin Melo	80515	7.000000	1	None
10	Mirella Viana	28421	7.000000	1	None
11	Maria Ferreira	86563	6.500000	2	None
12	Antônio Azevedo	29022	6.000000	1	None
13	Laura Melo	50966	6.000000	1	None
14	Pedro Henrique da Costa	68004	4.000000	1	None

Módulo 17 – Condicionais com CASE

Vamos adicionar todas as condições. Observe que colocamos um **ELSE** para o caso de haver alguma condição que não se encaixe nas demais listadas.

```
# Adicionando TODAS as condições
resultado_sql = executa_sql('SELECT nome_aluno, cod_matricula, \
                             AVG(nota_prova) as media_prova, \
                             count(nota_prova) as qtd_provas, \
                             (CASE \
                                 WHEN AVG(nota_prova) > 7 THEN "Aprovado" \
                                 WHEN AVG(nota_prova) > 5 AND count(nota_prova) = 1 THEN "Fazer prova 2" \
                                 WHEN AVG(nota_prova) > 5 AND count(nota_prova) = 2 THEN "Fazer prova final" \
                                 WHEN AVG(nota_prova) > 5 AND count(nota_prova) > 2 THEN "Revisar matéria" \
                                 WHEN AVG(nota_prova) < 5 AND count(nota_prova) = 1 THEN "Revisar matéria" \
                                 WHEN AVG(nota_prova) < 5 AND count(nota_prova) > 1 THEN "Lista de exercícios" \
                                 ELSE "Verificar manualmente" \
                             END) as situacao_aluno \
                             FROM dados \
                             WHERE nota_prova IS NOT NULL \
                             GROUP BY nome_aluno,cod_matricula \
                             ORDER BY AVG(nota_prova) desc')
```

```
(15, 5)
```

	nome_aluno	cod_matricula	media_prova	qtd_provas	situacao_aluno
0	Bárbara da Cunha	63546	10.0	1	Aprovado
1	Bárbara Freitas	19442	8.0	2	Aprovado
2	Gabriela Costela	21262	8.0	1	Aprovado
3	Júlia Pinto	47086	8.0	2	Aprovado
4	Lívia Jesus	22284	8.0	1	Aprovado

Módulo 17 – Condicionais com CASE

Observe, agora, que todas as situações foram contempladas com alguma condição:

```
1  display(resultado_sql)
```

	nome_aluno	cod_matricula	media_prova	qtd_provas	situacao_aluno
0	Bárbara da Cunha	63546	10.000000	1	Aprovado
1	Bárbara Freitas	19442	8.000000	2	Aprovado
2	Gabriela Costela	21262	8.000000	1	Aprovado
3	Júlia Pinto	47086	8.000000	2	Aprovado
4	Lívia Jesus	22284	8.000000	1	Aprovado
5	Melissa Ribeiro	38438	8.000000	2	Aprovado
6	Maria Eduarda da Rocha	38273	7.666667	3	Aprovado
7	Eloah Aragão	65749	7.500000	4	Aprovado
8	Isabelly Souza	31859	7.000000	1	Fazer prova 2
9	Kevin Melo	80515	7.000000	1	Fazer prova 2
10	Mirella Viana	28421	7.000000	1	Fazer prova 2
11	Maria Ferreira	86563	6.500000	2	Fazer prova final
12	Antônio Azevedo	29022	6.000000	1	Fazer prova 2
13	Laura Melo	50966	6.000000	1	Fazer prova 2
14	Pedro Henrique da Costa	68004	4.000000	1	Revisar matéria

Módulo 17 – Subquery

Vamos simplificar um pouco nossa condicional, conforme segue:

```
# Utilizando parte do CASE que vimos na última aula
resultado_sql = executa_sql('SELECT nome_aluno, cod_matricula, \
                             AVG(nota_prova) as media_prova, \
                             count(nota_prova) as qtd_provas, \
                             (CASE \
                                 WHEN AVG(nota_prova) > 7 THEN "Aprovado" \
                                 WHEN AVG(nota_prova) > 5 AND count(nota_prova) = 1 THEN "Fazer prova 2" \
                             END) as situacao_aluno \
                             FROM dados \
                             WHERE nota_prova IS NOT NULL \
                             GROUP BY nome_aluno,cod_matricula \
                             ORDER BY AVG(nota_prova) desc')
```

```
(15, 5)
```

	nome_aluno	cod_matricula	media_prova	qtd_provas	situacao_aluno
0	Bárbara da Cunha	63546	10.0	1	Aprovado
1	Bárbara Freitas	19442	8.0	2	Aprovado
2	Gabriela Costela	21262	8.0	1	Aprovado
3	Júlia Pinto	47086	8.0	2	Aprovado
4	Lívia Jesus	22284	8.0	1	Aprovado

Módulo 17 – Subquery

Poderíamos estruturar a query conforme a seguir. Observe que estamos aplicando a estrutura condicional ao resultado de uma query apresentada após o comando `FROM`. Nestes casos, dizemos estar utilizando uma *subquery*. Uma subquery pode aparecer no lugar de uma expressão, se esta retornar um valor único.

```
# Transformando a query sem o CASE em uma subquery
resultado_sql = executa_sql('SELECT nome_aluno, cod_matricula, media_prova, qtd_provas, \
                             (CASE \
                                 WHEN media_prova > 7 THEN "Aprovado" \
                                 WHEN media_prova > 5 AND qtd_provas = 1 THEN "Fazer prova 2" \
                              END) as situacao_aluno \
                             FROM (SELECT nome_aluno, cod_matricula, \
                             AVG(nota_prova) as media_prova, \
                             count(nota_prova) as qtd_provas \
                             FROM dados \
                             WHERE nota_prova IS NOT NULL \
                             GROUP BY nome_aluno,cod_matricula \
                             ORDER BY AVG(nota_prova) desc)')
```

```
(15, 5)
```

	nome_aluno	cod_matricula	media_prova	qtd_provas	situacao_aluno
0	Bárbara da Cunha	63546	10.0	1	Aprovado
1	Bárbara Freitas	19442	8.0	2	Aprovado
2	Gabriela Costela	21262	8.0	1	Aprovado
3	Júlia Pinto	47086	8.0	2	Aprovado
4	Lívia Jesus	22284	8.0	1	Aprovado

Módulo 17 – Outros filtros - IN

Em alguns contextos, gostaríamos de fazer filtros de valores dentro de um dado intervalo ou conjunto. Com o visto até o momento, poderíamos fazer utilizando `WHERE` para dias do último acesso entre 10 e 12:

```
# Verificando os dias de último acesso iguais a 10,11,12 usando o WHERE + AND
resultado_sql = executa_sql('SELECT * FROM dados WHERE dias_ultimo_acesso >= 10 AND dias_ultimo_acesso <=12')
```

(7, 11)

	index	id_aluno	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado	dias_ultimo_acesso	nr_prova	prova_feita	nota_prova
0	0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	1	1	9.0
1	17	18	Bárbara Freitas	19442	barbara@hashtag.com	0	1	12	1	1	6.0
2	20	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	2	1	8.0
3	32	18	Bárbara Freitas	19442	barbara@hashtag.com	0	1	12	2	1	10.0
4	34	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	3	1	6.0

Outra forma de fazer seria utilizar o comando `IN` e passar uma listagem de valores, indicando que gostaríamos de filtrar nossa base para apenas casos onde tal coluna apresenta valores neste conjunto:

```
# Alterando para usar o IN
resultado_sql = executa_sql('SELECT * FROM dados \
                             WHERE dias_ultimo_acesso IN (10,11,12)')
```

(7, 11)

	index	id_aluno	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado	dias_ultimo_acesso	nr_prova	prova_feita	nota_prova
0	0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	1	1	9.0
1	17	18	Bárbara Freitas	19442	barbara@hashtag.com	0	1	12	1	1	6.0
2	20	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	2	1	8.0
3	32	18	Bárbara Freitas	19442	barbara@hashtag.com	0	1	12	2	1	10.0
4	34	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	12	3	1	6.0

Módulo 17 – Outros filtros - LIKE

Combinando com `DISTINCT`, podemos pegar alunos por número de matrícula ou por nome:

```
# Utilizando o IN para filtrar as matrículas 28421,38273,42674,65749
resultado_sql = executa_sql('SELECT DISTINCT nome_aluno, cod_matricula FROM dados \
                             WHERE cod_matricula IN (28421,38273,42674,65749)')
```

(4, 2)

	nome_aluno	cod_matricula
0	Maria Eduarda da Rocha	38273
1	Mirella Viana	28421
2	Cauã Porto	42674
3	Eloah Aragão	65749

```
# Verificando alunos que o nome seja igual Maria Ferreira utilizando o IN
resultado_sql = executa_sql('SELECT DISTINCT nome_aluno, cod_matricula FROM dados \
                             WHERE nome_aluno IN ("Maria Ferreira")')
```

(1, 2)

	nome_aluno	cod_matricula
0	Maria Ferreira	86563

Módulo 17 – Outros filtros - LIKE

O comando `LIKE` no SQL é utilizado para buscar registros em uma tabela que correspondam a um padrão específico. Ele é frequentemente usado em combinação com a cláusula `WHERE` para filtrar resultados com base em colunas de texto. Assim, o mesmo último exemplo da página anterior poderia ser feito como:

```
1  # Fazendo a mesma consulta utilizando o LIKE
2  resultado_sql = executa_sql('SELECT DISTINCT nome_aluno, cod_matricula FROM dados \
3                               WHERE nome_aluno LIKE "Maria Ferreira"')
```

```
(1, 2)
```

	nome_aluno	cod_matricula
0	Maria Ferreira	86563

Módulo 17 – Outros filtros - LIKE

Mas a vantagem do LIKE é possibilidade de utilizar caracteres curinga para definir padrões:

- %: Representa qualquer sequência de caracteres (incluindo nenhuma sequência)
- _: Representa exatamente um caractere

```
# Verificando todas as alunas chamadas Maria
resultado_sql = executa_sql('SELECT DISTINCT nome_aluno, cod_matricula FROM dados \
                             WHERE nome_aluno LIKE "Maria%"')
```

(2, 2)

	nome_aluno	cod_matricula
0	Maria Eduarda da Rocha	38273
1	Maria Ferreira	86563

```
# Verificando se o e-mail possui o caracter !
resultado_sql = executa_sql('SELECT DISTINCT nome_aluno, cod_matricula,[e-mail],acesso_plataforma,acesso_liberado FROM dados \
                             WHERE [e-mail] LIKE "%!%"')
```

(1, 5)

	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado
0	Francisco Pires	59518	x!c0@hashtag.com	0	1

```
# Verificando se possui !,ã ou í
resultado_sql = executa_sql('SELECT DISTINCT nome_aluno, cod_matricula,[e-mail],acesso_plataforma,acesso_liberado FROM dados \
                             WHERE [e-mail] LIKE "%!%" \
                             OR [e-mail] LIKE "%ã%" \
                             OR [e-mail] LIKE "%í%"')
```

(3, 5)

	nome_aluno	cod_matricula	e-mail	acesso_plataforma	acesso_liberado
0	Lívia Jesus	22284	lívia@hashtag.com	0	1
1	Cauã Porto	42674	cauã@hashtag.com	0	1
2	Francisco Pires	59518	x!c0@hashtag.com	0	1

Módulo 17 – Unindo bases com JOIN

Em uma situação real, por vezes temos mais de uma tabela, cada uma com dados dentro de um determinado contexto de nosso problema. Por exemplo, nossa base provavelmente seria melhor aplicada numa situação real com três tabelas. Uma para dados cadastrais dos alunos, outra para informações sobre a plataforma, e outra para informações sobre provas:

```
1  # Visualizando a tabela alunos
2  alunos = executa_sql('SELECT * FROM alunos')
```

(20, 5)

	index	id_aluno	nome_aluno	cod_matricula	e-mail
0	0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com
1	1	2	Bárbara da Cunha	63546	barbara@hashtag.com
2	2	3	Kevin Melo	80515	kevin@hashtag.com
3	3	4	Pedro Henrique da Costa	68004	pedro@hashtag.com
4	4	5	Mirella Viana	28421	mirella@hashtag.com

```
1  # Visualizando a tabela plataforma
2  plataforma = executa_sql('SELECT * FROM plataforma')
```

(20, 6)

	index	id_plataforma	cod_matricula	acesso_plataforma	acesso_liberado	dias_ultimo_acesso
0	0	1	38273	0	1	12
1	1	2	63546	0	1	2
2	2	3	80515	1	1	3
3	3	4	68004	1	1	1
4	4	5	28421	1	1	1

```
1  # Visualizando a tabela provas
2  provas = executa_sql('SELECT * FROM provas')
```

(43, 6)

	index	id_prova	cod_matricula	nr_prova	prova_feita	nota_prova
0	0	1	38273	1	1	9.0
1	1	2	63546	1	1	10.0
2	2	3	80515	1	1	7.0
3	3	4	68004	1	1	4.0
4	4	5	28421	1	1	7.0

Módulo 17 – Unindo bases com JOIN

Esta divisão permite trabalhar com cada tabela isoladamente de acordo com o contexto desejado, diminuindo riscos de alterações indevidas em dados não diretamente relacionados a um determinado contexto. Observe que, em cada tabela, há uma coluna de identificação *id_*, e que todas elas podem ser relacionadas a partir do código de matrícula do aluno:

```
# Visualizando a tabela alunos
alunos = executa_sql('SELECT * FROM alunos')
```

(20, 5)

	index	id_aluno	nome_aluno	cod_matricula	e-mail
0	0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com
1	1	2	Bárbara da Cunha	63546	barbara@hashtag.com
2	2	3	Kevin Melo	80515	kevin@hashtag.com
3	3	4	Pedro Henrique da Costa	68004	pedro@hashtag.com
4	4	5	Mirella Viana	28421	mirella@hashtag.com

```
# Visualizando a tabela plataforma
plataforma = executa_sql('SELECT * FROM plataforma')
```

(20, 6)

	index	id_plataforma	cod_matricula	acesso_plataforma	acesso_liberado	dias_ultimo_acesso
0	0	1	38273	0	1	12
1	1	2	63546	0	1	2
2	2	3	80515	1	1	3
3	3	4	68004	1	1	1
4	4	5	28421	1	1	1

```
# Visualizando a tabela provas
provas = executa_sql('SELECT * FROM provas')
```

(43, 6)

	index	id_prova	cod_matricula	nr_prova	prova_feita	nota_prova
0	0	1	38273	1	1	9.0
1	1	2	63546	1	1	10.0
2	2	3	80515	1	1	7.0
3	3	4	68004	1	1	4.0
4	4	5	28421	1	1	7.0

Módulo 17 – Unindo bases com JOIN

Seguindo o padrão que adotamos até aqui, vamos verificar um exemplo de junção de duas tabelas com o Pandas e, depois, replicar o conceito com SQL.

Para começar, vamos criar dois dataframes simples:

```
# Criando 2 dataframes
dic1 = {
    "nomes": ['Nome1','Nome2','Nome3'],
    "valores": [1,2,3]
}

base_dic1 = pd.DataFrame(dic1)

dic2 = {
    "nomes": ['Nome1','Nome2','Nome4'],
    "valores": [9,8,7]
}

base_dic2 = pd.DataFrame(dic2)
```

```
base_dic2
```

	nomes	valores
0	Nome1	9
1	Nome2	8
2	Nome4	7

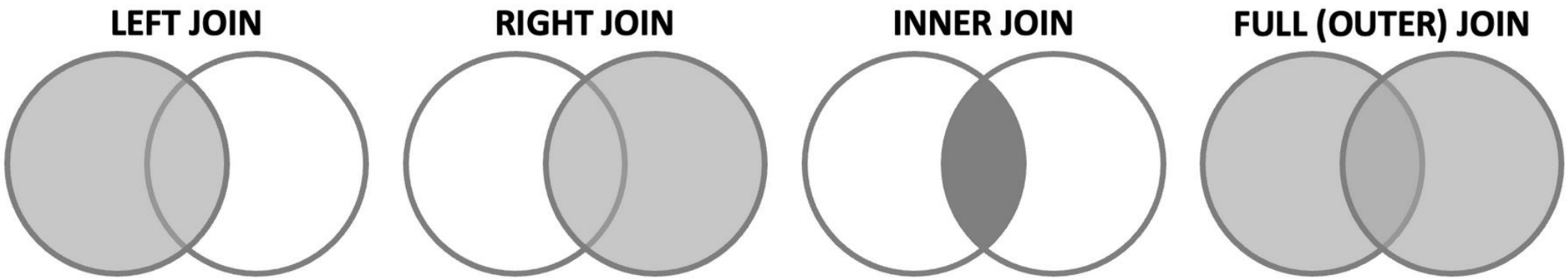
Módulo 17 – Unindo bases com JOIN

No Pandas, há o método `merge`, que permite juntar, ou mergear, dois dataframes. Passamos cada dataframe, o tipo de junção, e a coluna que deve ser considerada referência para junção.

No caso da junção do tipo `outer`, temos uma união das duas bases (diagrama abaixo).

```
base_merge = pd.merge(
    base_dic1, # <- primeira base
    base_dic2, # <- segunda base
    how='outer', # <- tipo de junção que vamos fazer
    on="nomes" # <- coluna que vamos usar para fazer essa junção das bases
)

display(base_merge)
```

	nomes	valores_x	valores_y
0	Nome1	1.0	9.0
1	Nome2	2.0	8.0
2	Nome3	3.0	NaN
3	Nome4	NaN	7.0



Módulo 17 – Unindo bases com JOIN

Replicando a lógica para nosso caso de estudo, podemos unir as bases *alunos* e *plataforma*, considerando a coluna *cod_matricula*.

Observe, ao lado, que fizemos uma junção do tipo `right`. Ou seja, retorna todos os registros da segunda tabela (*plataforma*) e os correspondentes da primeira tabela (*alunos*). Se não houver correspondência, os campos da primeira tabela aparecerão como nulos no resultado.

```
# Unindo a tabela alunos com a tabela plataforma
base_merge = pd.merge(
    alunos, # <- primeira base
    plataforma, # <- segunda base
    how='right', # <- tipo de junção que vamos fazer
    on="cod_matricula" # <- coluna que vamos usar para fazer essa junção das bases
)

display(base_merge)
```

	index_x	id_aluno	nome_aluno	cod_matricula	e-mail	index_y	id_plataforma	acesso_plataforma	acesso_liberado	dias_ultimo_acesso
0	0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com	0	1	0	1	12
1	1	2	Bárbara da Cunha	63546	barbara@hashtag.com	1	2	0	1	2
2	2	3	Kevin Melo	80515	kevin@hashtag.com	2	3	1	1	3
3	3	4	Pedro Henrique da Costa	68004	pedro@hashtag.com	3	4	1	1	1
4	4	5	Mirella Viana	28421	mirella@hashtag.com	4	5	1	1	1
5	5	6	Lívia Jesus	22284	lívia@hashtag.com	5	6	0	1	14
6	6	7	Lara Lopes	38362	lara@hashtag.com	6	7	1	1	1
7	7	8	Lucca Cardoso	45109	lucca@hashtag.com	7	8	1	1	3
8	8	9	Isabelly Souza	31859	isabelly@hashtag.com	8	9	1	1	3
9	9	10	Cauã Porto	42674	cauã@hashtag.com	9	10	0	1	2
10	10	11	Maria Ferreira	86563	maria@hashtag.com	10	11	0	1	1
11	11	12	Júlia Pinto	47086	julia@hashtag.com	11	12	1	1	2
12	12	13	Antônio Azevedo	29022	lovedogs@hashtag.com	12	13	0	1	14
13	13	14	Laura Melo	50966	laura@hashtag.com	13	14	1	1	2
14	14	15	Gabriela Costela	21262	gabriela@hashtag.com	14	15	1	1	2
15	15	16	Eloah Aragão	65749	eloah@hashtag.com	15	16	1	1	3
16	16	17	Francisco Pires	59518	x!c0@hashtag.com	16	17	0	1	13
17	17	18	Bárbara Freitas	19442	barbara@hashtag.com	17	18	0	1	12
18	18	19	Melissa Ribeiro	38438	melissa@hashtag.com	18	19	1	1	3
19	19	20	Lavínia Carvalho	89428	lavinia@hashtag.com	19	20	1	1	1

Módulo 17 – Unindo bases com JOIN

Podemos verificar que a junção partiu da tabela *plataforma* verificando que não há registros nulos oriundo desta tabela na base após o merge. Podemos, também, verificar que uma determinada matrícula, que está em *plataforma*, não está em *alunos*:

```
# Verificando se algum aluno não possui registro na tabela plataforma
base_merge[base_merge.id_plataforma.isnull()]
```

index_x	id_aluno	nome_aluno	cod_matricula	e-mail	index_y	id_plataforma	acesso_plataforma	acesso_liberado	dias_ultimo_acesso

```
# Verificando se algum registro da tabela plataforma não possui a matrícula na tabela aluno
base_merge[base_merge.id_aluno.isnull()]
```

index_x	id_aluno	nome_aluno	cod_matricula	e-mail	index_y	id_plataforma	acesso_plataforma	acesso_liberado	dias_ultimo_acesso

```
alunos[alunos.cod_matricula == 91047]
```

index	id_aluno	nome_aluno	cod_matricula	e-mail

```
base_merge.shape
```

```
(20, 10)
```

Módulo 17 – Unindo bases com JOIN

Agora que vimos no Pandas, vamos ver no SQL. Abaixo, estamos fazendo um **LEFT JOIN** da tabela *alunos*, chamada de *a*, com a tabela *plataforma*, chamada de *p*. Estes "apelidos" *a* e *p* ajudam a indicar a origem das colunas em nossa base após a junção. Observe que indicamos no **SELECT** a origem de cada coluna com *a.* ou *p.* antes do nome da coluna. O comando **ON** serve para indicar a coluna que servirá de referência para a junção.

```
# Trazendo as informações da tabela aluno e fazendo o join com a tabela plataforma
aluno_plataforma = executa_sql('SELECT a.nome_aluno, a.[e-mail], p.acesso_plataforma, \
                                a.cod_matricula \
                                FROM alunos a \
                                LEFT JOIN plataforma p \
                                ON a.cod_matricula = p.cod_matricula')
```

```
(20, 4)
```

	nome_aluno	e-mail	acesso_plataforma	cod_matricula
0	Maria Eduarda da Rocha	maria@hashtag.com	0	38273
1	Bárbara da Cunha	barbara@hashtag.com	0	63546
2	Kevin Melo	kevin@hashtag.com	1	80515
3	Pedro Henrique da Costa	pedro@hashtag.com	1	68004
4	Mirella Viana	mirella@hashtag.com	1	28421

Módulo 17 – Unindo bases com JOIN

No exemplo abaixo, trocamos a ordem das tabelas plataforma e alunos. Lembre-se que há uma matrícula presente em plataforma que não está presente em alunos. Podemos indicar em nossa query que queremos que apareça uma matrícula, independente de qual a tabela de origem. O **COALESCE** no SQL é uma função que ajuda a lidar com valores **NULL** em consultas. Ele recebe uma lista de argumentos e retorna o primeiro valor não **NULL** encontrado na lista.

```
# Trazendo as informações da tabela plataforma e fazendo o join com a tabela alunos
plataforma_aluno = executa_sql('SELECT a.nome_aluno, a.[e-mail], p.acesso_plataforma, \
                                coalesce(a.cod_matricula,p.cod_matricula) as cod_matricula \
                                FROM plataforma p \
                                LEFT JOIN alunos a \
                                ON a.cod_matricula = p.cod_matricula')
```

```
(20, 4)
```

	nome_aluno	e-mail	acesso_plataforma	cod_matricula
0	Maria Eduarda da Rocha	maria@hashtag.com	0	38273
1	Bárbara da Cunha	barbara@hashtag.com	0	63546
2	Kevin Melo	kevin@hashtag.com	1	80515
3	Pedro Henrique da Costa	pedro@hashtag.com	1	68004
4	Mirella Viana	mirella@hashtag.com	1	28421

Módulo 17 – Unindo bases com JOIN

Verifique que não há matrículas nulas em nosso retorno:

```
1 display(plataforma_aluno)
```

	nome_aluno	e-mail	acesso_plataforma	cod_matricula
0	Maria Eduarda da Rocha	maria@hashtag.com	0	38273
1	Bárbara da Cunha	barbara@hashtag.com	0	63546
2	Kevin Melo	kevin@hashtag.com	1	80515
3	Pedro Henrique da Costa	pedro@hashtag.com	1	68004
4	Mirella Viana	mirella@hashtag.com	1	28421
5	Lívia Jesus	lívia@hashtag.com	0	22284
6	Lara Lopes	lara@hashtag.com	1	38362
7	Lucca Cardoso	lucca@hashtag.com	1	45109
8	Isabelly Souza	isabelly@hashtag.com	1	31859
9	Cauã Porto	cauã@hashtag.com	0	42674
10	Maria Ferreira	maria@hashtag.com	0	86563
11	Júlia Pinto	julia@hashtag.com	1	47086
12	Antônio Azevedo	lovedogs@hashtag.com	0	29022
13	Laura Melo	laura@hashtag.com	1	50966
14	Gabriela Costela	gabriela@hashtag.com	1	21262
15	Eloah Aragão	eloah@hashtag.com	1	65749
16	Francisco Pires	x!c0@hashtag.com	0	59518
17	Bárbara Freitas	barbara@hashtag.com	0	19442
18	Melissa Ribeiro	melissa@hashtag.com	1	38438
19	Lavínia Carvalho	lavinia@hashtag.com	1	89428

Módulo 17 – O UNION e o FULL JOIN no SQL

Vamos relembrar nossa base *alunos*. Observe que há 21 registros:

```
1  # Visualizando a tabela alunos
2  alunos = executa_sql("SELECT * FROM alunos")
```

(21, 4)

	id_aluno	nome_aluno	cod_matricula	e-mail
0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com
1	2	Bárbara da Cunha	63546	barbara@hashtag.com
2	3	Kevin Melo	80515	kevin@hashtag.com
3	4	Pedro Henrique da Costa	68004	pedro@hashtag.com
4	5	Mirella Viana	28421	mirella@hashtag.com

```
1  display(alunos)
```

	id_aluno	nome_aluno	cod_matricula	e-mail
0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com
1	2	Bárbara da Cunha	63546	barbara@hashtag.com
2	3	Kevin Melo	80515	kevin@hashtag.com
3	4	Pedro Henrique da Costa	68004	pedro@hashtag.com
4	5	Mirella Viana	28421	mirella@hashtag.com
5	6	Lívia Jesus	22284	lívia@hashtag.com
6	7	Lara Lopes	38362	lara@hashtag.com
7	8	Lucca Cardoso	45109	lucca@hashtag.com
8	9	Isabelly Souza	31859	isabelly@hashtag.com
9	10	Cauã Porto	42674	cauã@hashtag.com
10	11	Maria Ferreira	86563	maria@hashtag.com
11	12	Júlia Pinto	47086	julia@hashtag.com
12	13	Antônio Azevedo	29022	lovedogs@hashtag.com
13	14	Laura Melo	50966	laura@hashtag.com
14	15	Gabriela Costela	21262	gabriela@hashtag.com
15	16	Eloah Aragão	65749	eloah@hashtag.com
16	17	Francisco Pires	59518	x!c0@hashtag.com
17	18	Bárbara Freitas	19442	barbara@hashtag.com
18	19	Melissa Ribeiro	38438	melissa@hashtag.com
19	20	Lavínia Carvalho	89428	lavinia@hashtag.com
20	21	Jorge Henrique Gomes	89791	jorge@hashtag.com

Considere que há uma outra base com o registro de novos alunos e que você deseja adicionar estes registros à base *alunos*, atualizando-a. Observe que o aluno Jorge Henrique Gomes, presente nesta tabela de alunos novos, já está presente na tabela *alunos* mostrada na página anterior:

```
1  # Visualizando a tabela alunos_novos
2  alunos_novos = executa_sql("SELECT * FROM alunos_novos")
```

(4, 4)

	id_aluno	nome_aluno	cod_matricula	e-mail
0	1	Pedro Carlos	115640	pedro@hashtag.com
1	2	Beatriz Rezende	100310	bia@hashtag.com
2	3	Sheila Costa	112366	sheila@hashtag.com
3	21	Jorge Henrique Gomes	89791	jorge@hashtag.com

Módulo 17 – O UNION e o FULL JOIN no SQL

O `UNION` é um operador usado para combinar os resultados de duas ou mais consultas `SELECT` em um único conjunto de resultados. Ele elimina linhas duplicadas, retornando apenas registros únicos. Antes do `UNION`, tínhamos 21 registros. Agora temos 24, pois o registro do aluno José Henrique Gomes já existia e não foi duplicado.

```
1  # Unindo as duas tabelas de alunos utilizando o UNION
2  alunos = executa_sql("SELECT * FROM alunos \
3                        UNION \
4                        SELECT * FROM alunos_novos")
```

(24, 4)

	id_aluno	nome_aluno	cod_matricula	e-mail
0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com
1	1	Pedro Carlos	115640	pedro@hashtag.com
2	2	Beatriz Rezende	100310	bia@hashtag.com
3	2	Bárbara da Cunha	63546	barbara@hashtag.com
4	3	Kevin Melo	80515	kevin@hashtag.com

```
1  display(alunos)
```

	id_aluno	nome_aluno	cod_matricula	e-mail
0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com
1	1	Pedro Carlos	115640	pedro@hashtag.com
2	2	Beatriz Rezende	100310	bia@hashtag.com
3	2	Bárbara da Cunha	63546	barbara@hashtag.com
4	3	Kevin Melo	80515	kevin@hashtag.com
5	3	Sheila Costa	112366	sheila@hashtag.com
6	4	Pedro Henrique da Costa	68004	pedro@hashtag.com
7	5	Mirella Viana	28421	mirella@hashtag.com
8	6	Lívia Jesus	22284	lívia@hashtag.com
9	7	Lara Lopes	38362	lara@hashtag.com
10	8	Lucca Cardoso	45109	lucca@hashtag.com
11	9	Isabelly Souza	31859	isabelly@hashtag.com
12	10	Cauã Porto	42674	cauã@hashtag.com
13	11	Maria Ferreira	86563	maria@hashtag.com
14	12	Júlia Pinto	47086	julia@hashtag.com
15	13	Antônio Azevedo	29022	lovedogs@hashtag.com
16	14	Laura Melo	50966	laura@hashtag.com
17	15	Gabriela Costela	21262	gabriela@hashtag.com
18	16	Eloah Aragão	65749	eloah@hashtag.com
19	17	Francisco Pires	59518	x!c0@hashtag.com
20	18	Bárbara Freitas	19442	barbara@hashtag.com
21	19	Melissa Ribeiro	38438	melissa@hashtag.com
22	20	Lavínia Carvalho	89428	lavinia@hashtag.com
23	21	Jorge Henrique Gomes	89791	jorge@hashtag.com

Módulo 17 – O UNION e o FULL JOIN no SQL

Se, por qualquer motivo, você quiser atualizar com todos os registros, mesmo duplicados, há o **UNION ALL**. Observe que o registro do aluno Jorge Henrique Gomes está duplicado agora:

```
1  # Unindo as duas tabelas de alunos utilizando o UNION ALL
2  alunos = executa_sql("SELECT * FROM alunos \
3                        UNION ALL \
4                        SELECT * FROM alunos_novos")
```

(25, 4)

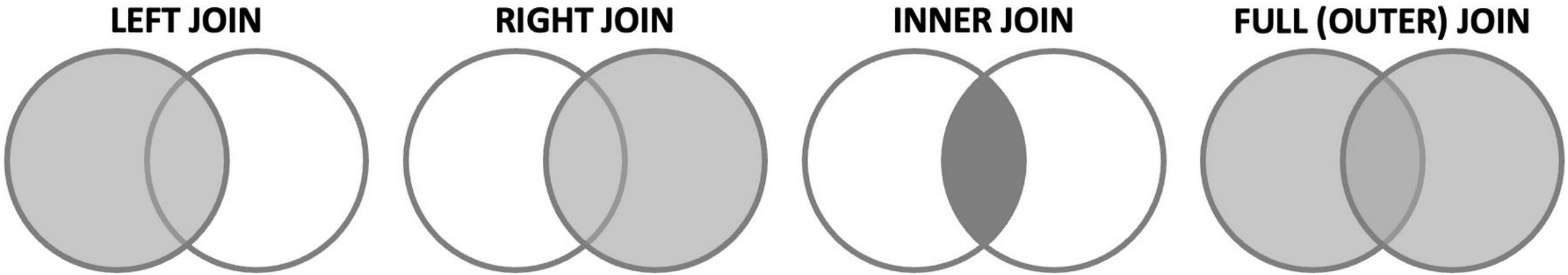
	id_aluno	nome_aluno	cod_matricula	e-mail
0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com
1	2	Bárbara da Cunha	63546	barbara@hashtag.com
2	3	Kevin Melo	80515	kevin@hashtag.com
3	4	Pedro Henrique da Costa	68004	pedro@hashtag.com
4	5	Mirella Viana	28421	mirella@hashtag.com

```
1  display(alunos)
```

	id_aluno	nome_aluno	cod_matricula	e-mail
0	1	Maria Eduarda da Rocha	38273	maria@hashtag.com
1	2	Bárbara da Cunha	63546	barbara@hashtag.com
2	3	Kevin Melo	80515	kevin@hashtag.com
3	4	Pedro Henrique da Costa	68004	pedro@hashtag.com
4	5	Mirella Viana	28421	mirella@hashtag.com
5	6	Lívia Jesus	22284	lívia@hashtag.com
6	7	Lara Lopes	38362	lara@hashtag.com
7	8	Lucca Cardoso	45109	lucca@hashtag.com
8	9	Isabelly Souza	31859	isabelly@hashtag.com
9	10	Cauã Porto	42674	cauã@hashtag.com
10	11	Maria Ferreira	86563	maria@hashtag.com
11	12	Júlia Pinto	47086	julia@hashtag.com
12	13	Antônio Azevedo	29022	lovedogs@hashtag.com
13	14	Laura Melo	50966	laura@hashtag.com
14	15	Gabriela Costela	21262	gabriela@hashtag.com
15	16	Eloah Aragão	65749	eloah@hashtag.com
16	17	Francisco Pires	59518	xlc0@hashtag.com
17	18	Bárbara Freitas	19442	barbara@hashtag.com
18	19	Melissa Ribeiro	38438	melissa@hashtag.com
19	20	Lavínia Carvalho	89428	lavinia@hashtag.com
20	21	Jorge Henrique Gomes	89791	jorge@hashtag.com
21	1	Pedro Carlos	115640	pedro@hashtag.com
22	2	Beatriz Rezende	100310	bia@hashtag.com
23	3	Sheila Costa	112366	sheila@hashtag.com
24	21	Jorge Henrique Gomes	89791	jorge@hashtag.com

Módulo 17 – O UNION e o FULL JOIN no SQL

O sqlite3 não possui nativamente um comando que possibilite fazer o OUTER JOIN, mostrado no diagrama abaixo. Mas, com o que vimos de UNION, podemos construir uma query que gera exatamente o mesmo resultado que seria obtido por tal tipo de junção.



Módulo 17 – O UNION e o FULL JOIN no SQL

Já vimos como realizar um `LEFT JOIN` entre as tabelas *alunos* e *plataforma*. Observe o valor nulo na coluna *acesso_plataforma* para o aluno Jorge Henrique Gomes.

```
# Vamos selecionar todos os valores da tabela alunos fazendo o join com a tabela plataforma
aluno_plataforma = executa_sql('SELECT a.nome_aluno, a.[e-mail], p.acesso_plataforma, \
                                a.cod_matricula \
                                FROM alunos a \
                                LEFT JOIN plataforma p \
                                ON a.cod_matricula = p.cod_matricula')
```

(21, 4)

	nome_aluno	e-mail	acesso_plataforma	cod_matricula
0	Maria Eduarda da Rocha	maria@hashtag.com	0.0	38273
1	Bárbara da Cunha	barbara@hashtag.com	0.0	63546
2	Kevin Melo	kevin@hashtag.com	1.0	80515
3	Pedro Henrique da Costa	pedro@hashtag.com	1.0	68004
4	Mirella Viana	mirella@hashtag.com	1.0	28421

```
display(aluno_plataforma)
```

	nome_aluno	e-mail	acesso_plataforma	cod_matricula
0	Maria Eduarda da Rocha	maria@hashtag.com	0.0	38273
1	Bárbara da Cunha	barbara@hashtag.com	0.0	63546
2	Kevin Melo	kevin@hashtag.com	1.0	80515
3	Pedro Henrique da Costa	pedro@hashtag.com	1.0	68004
4	Mirella Viana	mirella@hashtag.com	1.0	28421
5	Lívia Jesus	lívia@hashtag.com	0.0	22284
6	Lara Lopes	lara@hashtag.com	1.0	38362
7	Lucca Cardoso	lucca@hashtag.com	1.0	45109
8	Isabelly Souza	isabelly@hashtag.com	1.0	31859
9	Cauã Porto	cauã@hashtag.com	0.0	42674
10	Maria Ferreira	maria@hashtag.com	0.0	86563
11	Júlia Pinto	julia@hashtag.com	1.0	47086
12	Antônio Azevedo	lovedogs@hashtag.com	0.0	29022
13	Laura Melo	laura@hashtag.com	1.0	50966
14	Gabriela Costela	gabriela@hashtag.com	1.0	21262
15	Eloah Aragão	eloah@hashtag.com	1.0	65749
16	Francisco Pires	x!c0@hashtag.com	0.0	59518
17	Bárbara Freitas	barbara@hashtag.com	0.0	19442
18	Melissa Ribeiro	melissa@hashtag.com	1.0	38438
19	Lavínia Carvalho	lavinia@hashtag.com	1.0	89428
20	Jorge Henrique Gomes	jorge@hashtag.com	NaN	89791

Módulo 17 – O UNION e o FULL JOIN no SQL

Assim como já vimos o inverso, um `LEFT JOIN` entre as tabelas *plataforma* e *alunos*. Observe os valores nulos na linha da matrícula *91047*.

```
# Agora vamos fazer o join da tabela plataforma com a tabela aluno
plataforma_aluno = executa_sql('SELECT a.nome_aluno, a.[e-mail], p.acesso_plataforma, \
                                coalesce(a.cod_matricula,p.cod_matricula) as cod_matricula \
                                FROM plataforma p \
                                LEFT JOIN alunos a \
                                ON a.cod_matricula = p.cod_matricula')
```

(21, 4)

	nome_aluno	e-mail	acesso_plataforma	cod_matricula
0	Maria Eduarda da Rocha	maria@hashtag.com	0	38273
1	Bárbara da Cunha	barbara@hashtag.com	0	63546
2	Kevin Melo	kevin@hashtag.com	1	80515
3	Pedro Henrique da Costa	pedro@hashtag.com	1	68004
4	Mirella Viana	mirella@hashtag.com	1	28421

```
display(plataforma_aluno)
```

	nome_aluno	e-mail	acesso_plataforma	cod_matricula
0	Maria Eduarda da Rocha	maria@hashtag.com	0	38273
1	Bárbara da Cunha	barbara@hashtag.com	0	63546
2	Kevin Melo	kevin@hashtag.com	1	80515
3	Pedro Henrique da Costa	pedro@hashtag.com	1	68004
4	Mirella Viana	mirella@hashtag.com	1	28421
5	Lívia Jesus	lívia@hashtag.com	0	22284
6	Lara Lopes	lara@hashtag.com	1	38362
7	Lucca Cardoso	lucca@hashtag.com	1	45109
8	Isabelly Souza	isabelly@hashtag.com	1	31859
9	Cauã Porto	cauã@hashtag.com	0	42674
10	Maria Ferreira	maria@hashtag.com	0	86563
11	Júlia Pinto	julia@hashtag.com	1	47086
12	Antônio Azevedo	lovedogs@hashtag.com	0	29022
13	Laura Melo	laura@hashtag.com	1	50966
14	Gabriela Costela	gabriela@hashtag.com	1	21262
15	Eloah Aragão	eloah@hashtag.com	1	65749
16	Francisco Pires	x!c0@hashtag.com	0	59518
17	Bárbara Freitas	barbara@hashtag.com	0	19442
18	Melissa Ribeiro	melissa@hashtag.com	1	38438
19	Lavínia Carvalho	lavinia@hashtag.com	1	89428
20	None	None	0	91047

Módulo 17 – O UNION e o FULL JOIN no SQL

O **UNION** dos JOINs anteriores equivale a um OUTER JOIN das duas tabelas. Observe que duas as linhas destacadas nas páginas anteriores estão presentes no resultado:

```
# Para fazer o full join, podemos exatamente pegar esses dois left joins e fazer um UNION
full_join = executa_sql('SELECT a.nome_aluno, a.[e-mail], p.acesso_plataforma, \
                                coalesce(a.cod_matricula,p.cod_matricula) as cod_matricula \
                                FROM plataforma p \
                                LEFT JOIN alunos a \
                                ON a.cod_matricula = p.cod_matricula \
                         UNION \
                         SELECT a.nome_aluno, a.[e-mail], p.acesso_plataforma, \
                                a.cod_matricula \
                                FROM alunos a \
                                LEFT JOIN plataforma p \
                                ON a.cod_matricula = p.cod_matricula')
```

```
(22, 4)
```

	nome_aluno	e-mail	acesso_plataforma	cod_matricula
0	None	None	0.0	91047
1	Antônio Azevedo	lovedogs@hashtag.com	0.0	29022
2	Bárbara Freitas	barbara@hashtag.com	0.0	19442
3	Bárbara da Cunha	barbara@hashtag.com	0.0	63546
4	Cauã Porto	cauã@hashtag.com	0.0	42674

```
# Visualizando o full join
display(full_join)
```

	nome_aluno	e-mail	acesso_plataforma	cod_matricula
0	None	None	0.0	91047
1	Antônio Azevedo	lovedogs@hashtag.com	0.0	29022
2	Bárbara Freitas	barbara@hashtag.com	0.0	19442
3	Bárbara da Cunha	barbara@hashtag.com	0.0	63546
4	Cauã Porto	cauã@hashtag.com	0.0	42674
5	Eloah Aragão	eloah@hashtag.com	1.0	65749
6	Francisco Pires	x!c0@hashtag.com	0.0	59518
7	Gabriela Costela	gabriela@hashtag.com	1.0	21262
8	Isabelly Souza	isabelly@hashtag.com	1.0	31859
9	Jorge Henrique Gomes	jorge@hashtag.com	NaN	89791
10	Júlia Pinto	julia@hashtag.com	1.0	47086
11	Kevin Melo	kevin@hashtag.com	1.0	80515
12	Lara Lopes	lara@hashtag.com	1.0	38362
13	Laura Melo	laura@hashtag.com	1.0	50966
14	Lavínia Carvalho	lavinia@hashtag.com	1.0	89428
15	Lucca Cardoso	lucca@hashtag.com	1.0	45109
16	Lívia Jesus	lívia@hashtag.com	0.0	22284
17	Maria Eduarda da Rocha	maria@hashtag.com	0.0	38273
18	Maria Ferreira	maria@hashtag.com	0.0	86563
19	Melissa Ribeiro	melissa@hashtag.com	1.0	38438
20	Mirella Viana	mirella@hashtag.com	1.0	28421
21	Pedro Henrique da Costa	pedro@hashtag.com	1.0	68004