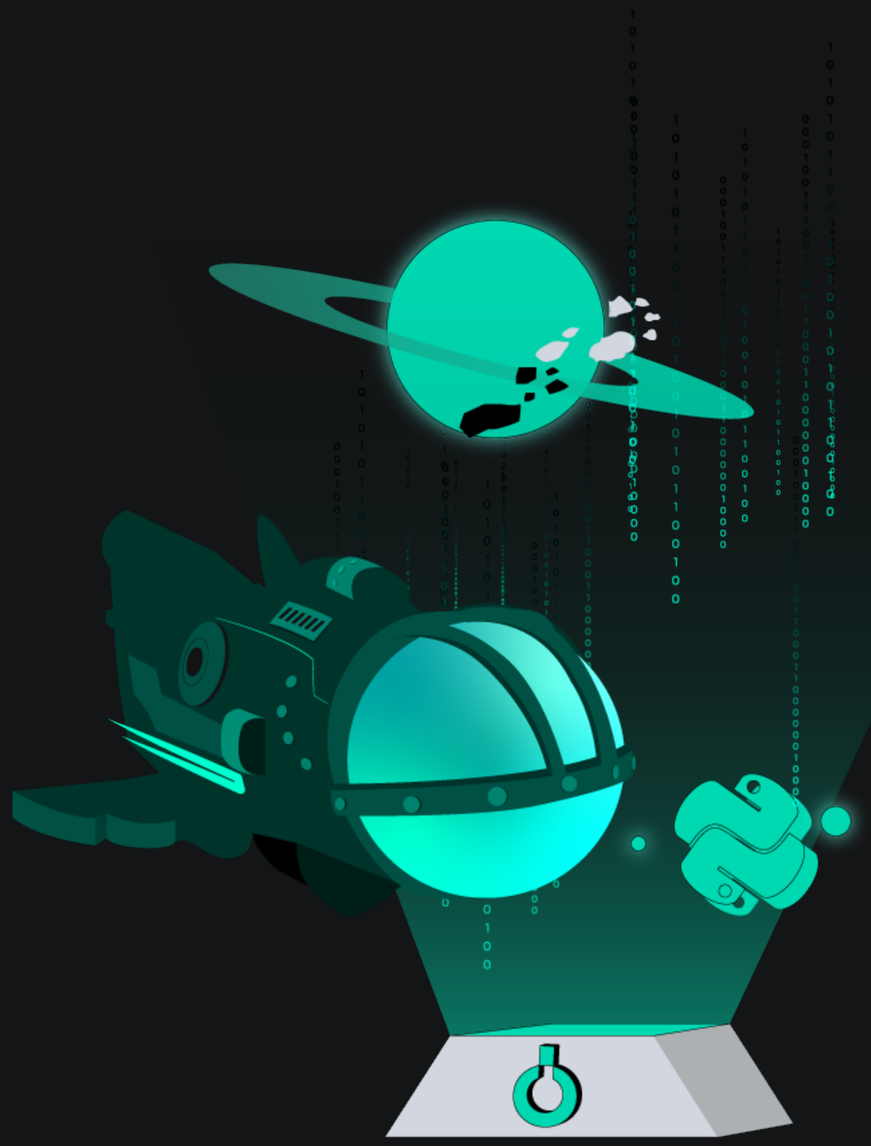
código.py

GALÁXIA 5

Matplotlib



Galáxia 5

Introdução aos pandas.

Mundo 1

O que é o Matplotlib?

Mundo 2

2.1. Pré-configurações

2.2. matplotlib.pyplot.plot

2.3. DataFrame.plot

2.3.1. DataFrame.plot.area

2.3.2. DataFrame.plot.bar

2.3.3. DataFrame.plot.barh

2.3.4. DataFrame.plot.box

2.3.5. DataFrame.plot.density

2.3.6. DataFrame.plot.hexbin

2.3.7. DataFrame.plot.hist

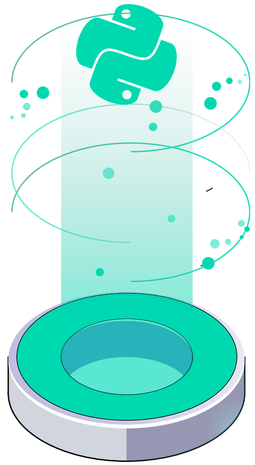
2.3.8. DataFrame.plot.area

2.3.9. DataFrame.plot.kde

2.3.10. DataFrame.plot.line

2.3.11. DataFrame.plot.pie

2.3.12. DataFrame.plot.scatter



Mundo 3

3.1. matplotlib.pyplot.subplots

Mundo 4

4.1. matplotlib.axes.Axes.set_ylabel

4.2. matplotlib.axes.Axes.set_xlabel

Mundo 5

5.1. matplotlib.axes.Axes.set_title

Mundo 6

6.1. matplotlib.axes.Axes.set_xlim

6.2. matplotlib.axes.Axes.set_ylim

6.3. matplotlib.axes.Axes.set_yscale

Mundo 7

7.1. matplotlib.pyplot.legend

Mundo 8

8.1. matplotlib.axes.Axes.tick_params

8.2. matplotlib.axes.Axes.set_major_formatter

8.3. matplotlib.axes.Axes.set_major_locator

Mundo 9

9.1. matplotlib.axes.Axes.set_facecolor

9.2. matplotlib.style.use

Mundo 10

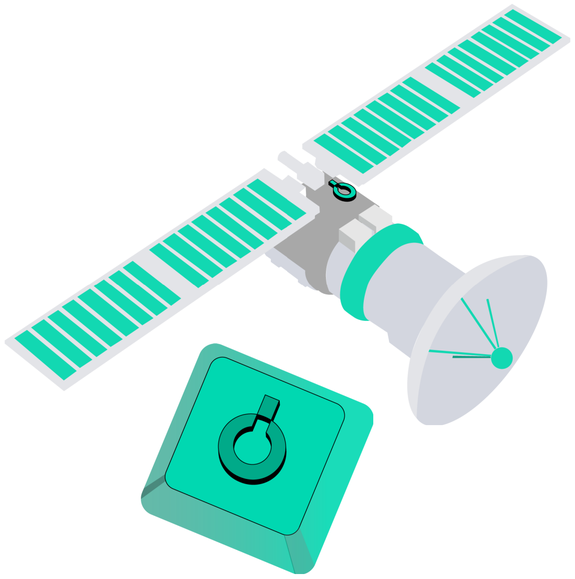
10.1. Plotando um gráfico de barras

Mundo 11

11.1. Plotando um histograma

11.2. Plotando um gráfico de dispersão

11.3. Plotando um gráfico de boxplot



Mundo 12

12.1. seaborn.boxplot

12.2. seaborn.histplot

12.3. seaborn.regplot

12.4. seaborn.lineplot

Mundo 13

13.1. Axes.annotate

Mundo 14

14.1. Pré-configurações

14.2. matplotlib.animation.FuncAnimation

Mundo 15

15.1. Plotando um gráfico de calor

Mundo 16

16.1. Plotando um gráfico de calor

Introdução

Olá, seja bem vindo à Galáxia 5!! Neste módulo iremos abordar sobre a biblioteca matplotlib e alguns módulos auxiliares. Ela é uma biblioteca de código aberto do Python que é utilizada para visualização dos dados. Então simbora que temos muito assunto pela frente.

Observação

Nas aulas deste mundo foi utilizada a biblioteca "pandas_datareader" porém por motivos de mau funcionamento da biblioteca, optamos por utilizar a biblioteca "yfinance". Ela pode ser instalada através do comando:

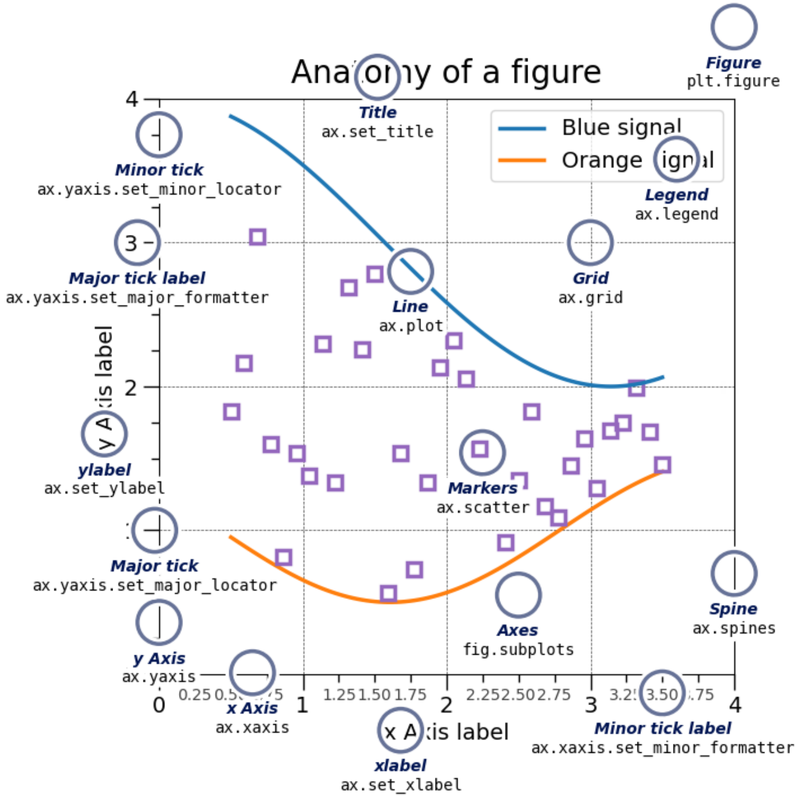
```
pip install yfinance
```

É a mesma dinâmica do "pandas_datareader", então não se preocupe quanto a isso. Vamos parar de enrolação, e bora para o conteúdo.

Mundo 1

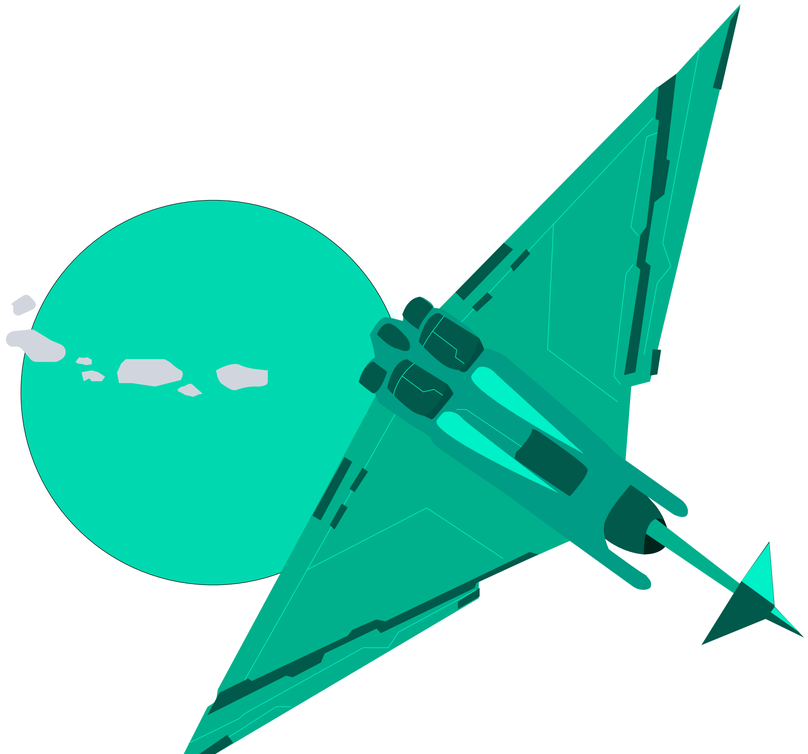
O matplotlib é a biblioteca padrão de gráficos dentro do python. Ao longo do curso serão utilizados outros pacotes como: pandas, seaborn, quantstats, etc... Seja qual for, esse pacote usará como base o matplotlib. Esta biblioteca trabalha com duas dimensões para criação de gráficos, as figuras e os eixos.

Abaixo está a anatomia do matplotlib. Nesta imagem estão contidas as principais características que podem ser atribuídas ao utilizar a biblioteca. Essa imagem é um ótimo instrumento em caso de dúvidas na hora de acessar ou observar alguma propriedade.



O matplotlib é uma biblioteca que possui muitos submódulos, ao todo são 54. Os principais são:

- Pyplot, que é utilizado para plotagem de gráficos no geral
- Animation, que é utilizado para rodar animações nos gráficos
- Axes, que é utilizado para configurar os detalhes dos eixos nos gráficos



Mundo 2

Neste mundo damos nosso primeiro passo em direção ao aprendizado. Para nossas aulas serão utilizadas, além do matplotlib, outras bibliotecas que compõe a análise de dados como: pandas, numpy e pandas_datareader. Então é importante que você já esteja familiarizado com estas bibliotecas.

2.1. Pré-configurações

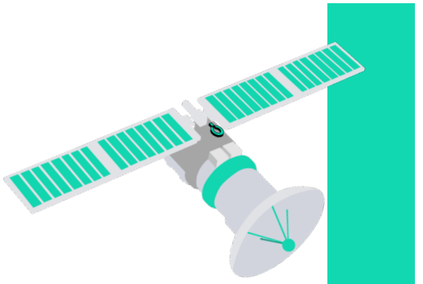
Caso você esteja utilizando um tema dark, existe a possibilidade de que as definições de cores predefinidas do matplotlib te atrapalhem na hora da visualização. Por isso, abaixo está o código que será utilizado para fazer esta mudança.

O código transforma os eixos e o background do gráfico em branco.

Repare também que, não estamos definindo essas propriedades para apenas um gráfico e sim tornando padrão, para que essas sejam replicadas para todos os outros gráficos que iremos fazer.

Exemplo:

```
import matplotlib.pyplot as plt

params = {"ytick.color" : "w",
          "xtick.color" : "w",
          "axes.labelcolor" : "w",
          "axes.edgecolor" : "w"}

plt.rcParams.update(params)
```



2.2. matplotlib.pyplot.plot(x , y , [fmt = ‘b-’] , color = “blue”, marker = None, linestyle = “-”, label = None, linewidth = 1, markersize = 5)

Este método é usado para plotar dados em um gráfico. **Este é um método da biblioteca matplotlib, do submódulo pyplot**.

Parâmetros:

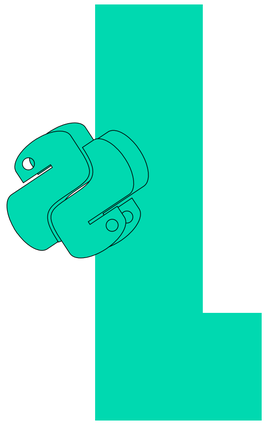
Os únicos parâmetros obrigatórios são: **x ou y**, sendo que eles não precisam ser definidos juntos.

x:

Este parâmetro vai definir as coordenadas do eixo x do gráfico.

É um parâmetro **obrigatório** que deve receber uma lista contendo integers.

y:



Este parâmetro vai definir as coordenadas do eixo y do gráfico.

É um parâmetro **obrigatório** que deve receber uma lista contendo integers.

[fmt]:

Este parâmetro vai definir a formatação da cor da linha, tipo de linha e tipo de marcador. Por padrão, o matplotlib define que a formatação será ‘b-’, ou seja, será da cor azul (b) e será uma linha contínua (-).

É um parâmetro **opcional** que deve receber a formatação em string com formatos pré definidos que obedecem o formato de cores, tipos de linhas e tipos de marcadores dos links anteriores.

color:

Este parâmetro vai definir a cor da série de dados de acordo com o formato de cores. Por padrão o pandas define que a cor padrão é azul.

É um parâmetro **opcional** que deve receber a formatação em string com formatos pré-definidos que obedecem o formato de cores.

marker:

Este parâmetro vai definir o marcador da série de dados de acordo com os tipos de marcadores. Por padrão o pandas define que não há marcador.

É um parâmetro **opcional** que deve receber a formatação em string com formatos pré-definidos que obedecem os tipos de marcadores.

linestyle:

Este parâmetro vai definir a formatação da linha da série de dados de acordo com as formatações das linhas. Por padrão o pandas define que a linha é contínua.

É um parâmetro **opcional** que deve receber a formatação em string com formatos pré-definidos que obedecem os tipos de linhas.

label:

Este parâmetro vai definir o rótulo da sua série. Por padrão, o matplotlib considera label = None.

É um parâmetro **opcional** que deve receber o nome em string.

obs: Você pode estranhar caso o rótulo não apareça no gráfico, isso porque você definiu apenas o nome da sua série de dados. Caso você queira que a legenda apareça, é só colocar "plt.legend()" que isso definirá a legenda como sendo o rótulo predefinido.

linewidth:

Este parâmetro vai definir a largura da sua série de dados. Por padrão o matplotlib define linewidth = 1.

É um parâmetro **opcional** que deve receber a largura da série de dados em integer.

markersize:

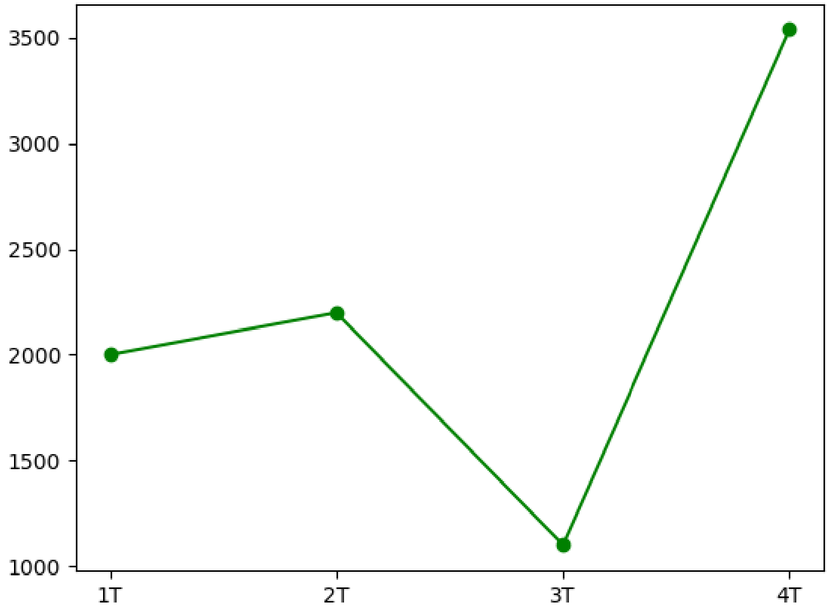
Este parâmetro vai definir a largura do seu marcador. Por padrão, o matplotlib define markersize = 5.

É um parâmetro **opcional** que deve receber a largura do marcador em integer.

Exemplo:

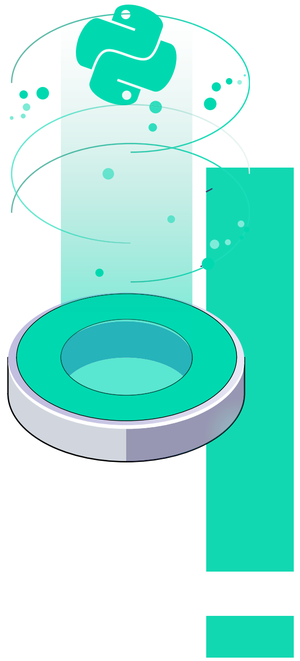
```
import matplotlib.pyplot as plt

x = ["1T", "2T", "3T", "4T"]
y = [2000, 2200, 1100, 3540]

plt.plot(x, y, color="green", marker='o')

plt.show()
```

Resposta:



2.3. DataFrame.plot(x = None, y = None, kind = ‘line’, ax = None, figsize = None, use_index = True, title = None, grid = None, legend = True, style = None, xticks = None, yticks = None, xlabel = None, ylabel = None, rot = None, fontsize = None, colormap = None, colorbar = False, stacked = False)

Este método é utilizado para plotar DataFrames. A princípio este item pode ser parecido com o item 2.2 matplotlib.pyplot.plot(). Porém, enquanto um é um método do matplotlib que é utilizado para plotar séries, o outro é um método do pandas utilizado para plotar DataFrames. Podemos considerar que este item (2.3 Dataframe.plot()) é uma “integração” do pandas com a biblioteca matplotlib.



Parâmetros:

Os únicos parâmetros obrigatórios são: **x ou y** . Sendo que eles não precisam ser definidos juntos, pode ser um ou outro.

x:

Este parâmetro vai definir os dados do eixo x. Neste caso deve ser a coluna de um DataFrame. Só pode receber um valor.

É um parâmetro **obrigatório** que deve receber o nome de uma coluna de um DataFrame em string ou a posição de uma coluna de um DataFrame em integer.

y:

Este parâmetro vai definir os dados do eixo y. Neste caso deve ser a coluna de um DataFrame. Pode receber múltiplos valores.

É um parâmetro **obrigatório** que deve receber o nome de uma, ou mais, colunas de um DataFrame em string ou a posição de uma, ou mais, colunas de um DataFrame em integer. Caso você passe mais de uma coluna, você deverá colocar as posições, ou nomes, dentro de uma lista.

kind:

Este parâmetro vai definir o formato que o gráfico será apresentado, ou em outras palavras, o tipo do gráfico. Por padrão, o pandas define kind = 'line', ou seja, o gráfico será apresentado em formato de linha contínua.

É um parâmetro **opcional** que deve receber o tipo em string com algum dos formatos predefinidos em formatos de gráficos.

ax:

Este parâmetro é utilizado para definir em qual objeto o gráfico plotado deve ser desenhado. Imagine que um eixo teórico "ax" é como se fosse um container, que contém os elementos, título, dados, legendas e etc... já definidos. Caso a gente queira aproveitar esses elementos é só definir na função do gráfico a ser plotado o objeto que será utilizado (no caso o "ax"). Este é o conceito de trabalhar com objetos, conseguimos reutilizar propriedades de outros objetos. Em resumo, é como se estivesse colando um gráfico sem formatação em cima de um gráfico já formatado.

É um parâmetro **opcional** que deve receber o nome do objeto, que contém os elementos, a ser utilizado.

figsize:

Este parâmetro vai definir o tamanho do gráfico que você deseja.

É um parâmetro **opcional** que receberá dois integer em uma tuple (largura , altura).

use_index:

Este parâmetro vai ser utilizado quando se desejar usar o índex do DataFrame no eixo x. Isso porque não é possível definir o index no parâmetro "x". Por padrão o pandas define use_index = False, ou seja, ele não usará o index.

É um parâmetro **opcional** que receberá um booleano com True ou False.

title:

Este parâmetro vai ser definir o título do gráfico a ser plotado. Por padrão, o pandas define title = None, ou seja, nenhum título será definido.

É um parâmetro **opcional** que receberá uma string com o nome do título.

grid:

Este parâmetro vai ser definir se o gráfico terá linhas de grade ou não. Por padrão, o pandas define grid = False, ou seja, não serão definidas linhas de grade.

É um parâmetro **opcional** que receberá um booleano com True ou False.

legend:

Este parâmetro vai definir se o gráfico terá legenda ou não. Por padrão, o pandas define legend = True, ou seja, o gráfico terá a legenda explicando as cores que representam os dados.

É um parâmetro **opcional** que receberá um booleano com True ou False. Pode receber também uma string entre chaves com a palavra "reverse" {"reverse"} caso deseje inverter as cores dos dados.

style:

Este parâmetro vai definir o estilo de cada coluna do DataFrame representada no gráfico. Por padrão, o pandas define style = None, ou seja, será um estilo padrão.

Esse parâmetro obedece a ordem de estilo já pré definida pela biblioteca de Formatação de cor, Formatação de marcador, Formatação de linha.

É um parâmetro **opcional** que receberá um dict com o nome de cada coluna, em string, e a formatação predefinida, em string, ao lado. Exemplo: {"WEGE3.SA" : "ro--" , "PETR4.SA" : "g>-"}.

xticks:

Este parâmetro vai definir os valores que aparecerão no eixo x.

É um parâmetro **opcional** que receberá uma lista com os valores , em integer, a serem definidos no eixo.

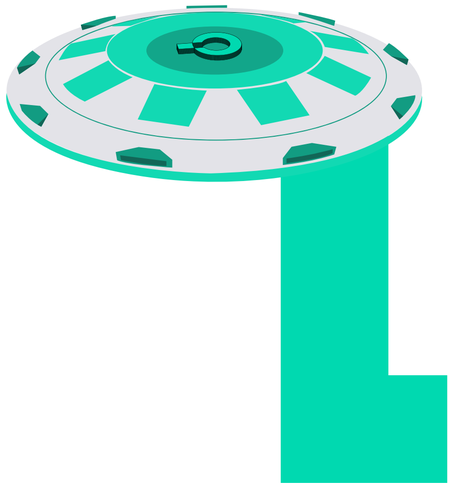
yticks:

Este parâmetro vai definir os valores que aparecerão no eixo y.

É um parâmetro **opcional** que receberá uma lista com os valores , em integer, a serem definidos no eixo.

xlabel:

Este parâmetro vai definir o nome que será dado ao eixo x. Por padrão, o pandas define xlabel = None, ou seja, nenhum nome será definido ao eixo.



É um parâmetro opcional que receberá o nome a ser dado no formato string.

ylabel:

Este parâmetro vai definir o nome que será dado ao eixo y. Por padrão, o pandas define ylabel = None, ou seja, nenhum nome será definido ao eixo.

É um parâmetro **opcional** que receberá o nome a ser dado no formato string.

rot:

Este parâmetro vai fazer com que você defina a rotação dos valores do eixo x (para gráficos verticais) ou dos valores do eixo y (para gráficos horizontais). Este parâmetro utiliza a medida °.

É um parâmetro **opcional** que receberá a rotação a ser dado aos valores no formato integer.

fontsize:

Este parâmetro vai definir o tamanho dos valores dos eixos x e y.

É um parâmetro **opcional** que receberá os tamanhos dos valores de x e y no formato integer.

colormap:

Este parâmetro vai definir as cores do seu gráfico, ela faz isso para todas as linhas, ou barras, ou etc. As cores já são pré-definidas pelo pandas, no total são 124.

É um parâmetro **opcional** que receberá o estilo da cor no formato string de acordo com cores pré-definidas pelo pandas.

colorbar:

Este parâmetro vai definir se terá uma barra de cor por densidade ao lado. Por padrão, o pandas define colorbar = False, ou seja, a barra não aparecerá.

É um parâmetro **opcional** que receberá um booleano com True ou False.

stacked:

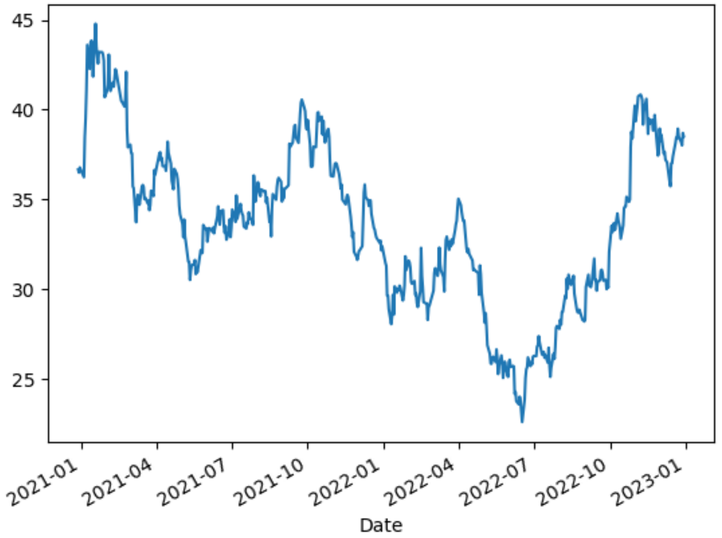
Este parâmetro vai definir se o gráfico será empilhado ou não. Por padrão, nos gráficos de linha e barra, o pandas define stacked = False, ou seja, não estará empilhado. Porém no caso do gráfico de área, o padrão é stacked = True, ou seja, o gráfico de área por padrão virá empilhado.

É um parâmetro **opcional** que receberá um booleano com True ou False.

Exemplo:

```
import yfinance as yf
import matplotlib.pyplot as plt

cotacoes = yf.download("WEGE3.SA","2020-12-28", "2023-01-01")['Adj Close']

cotacoes.plot(x=cotacoes.index, y=cotacoes.values, kind='line')

plt.show()
```

Respostas:



2.3.1. Argumentos adicionais do método DataFrame.plot()

O método DataFrame.plot() pode receber alguns outros argumentos, e alguns desses são os tipos de gráficos, então as explicações abaixo poderiam ser feito do mesmo jeito, se a gente utilizasse DataFrame.plot(kind = "tipo_do_grafico").

2.3.2. DataFrame.plot.area(x = None, y = None, stacked = True, others)

Esse argumento adicional é utilizado para plotar um gráfico de área.

Parâmetros:

Os únicos parâmetros obrigatórios são: **x ou y** . Sendo que eles não precisam ser definidos juntos, pode ser um ou outro.

x:

Este parâmetro vai definir os dados do eixo x. Neste caso deve ser a coluna de um DataFrame. Só pode receber um valor.

É um parâmetro **opcional** que deve receber o nome de uma coluna de um DataFrame em string ou a posição de uma coluna de um DataFrame em integer.

y:

Este parâmetro vai definir os dados do eixo y. Neste caso deve ser a coluna de um DataFrame. Pode receber múltiplos valores.

É um parâmetro **opcional** que deve receber o nome de uma, ou mais, colunas de um DataFrame em string ou a posição de uma, ou mais, colunas de um DataFrame em integer. Caso você passe mais de uma coluna, você deverá colocar as posições, ou nomes, dentro de uma lista.

stacked:

Este parâmetro vai definir se o gráfico será empilhado ou não. Por padrão, nos gráficos de área, o padrão é stacked = True, ou seja, o gráfico de área virá empilhado.

É um parâmetro **opcional** que receberá um booleano com True ou False.

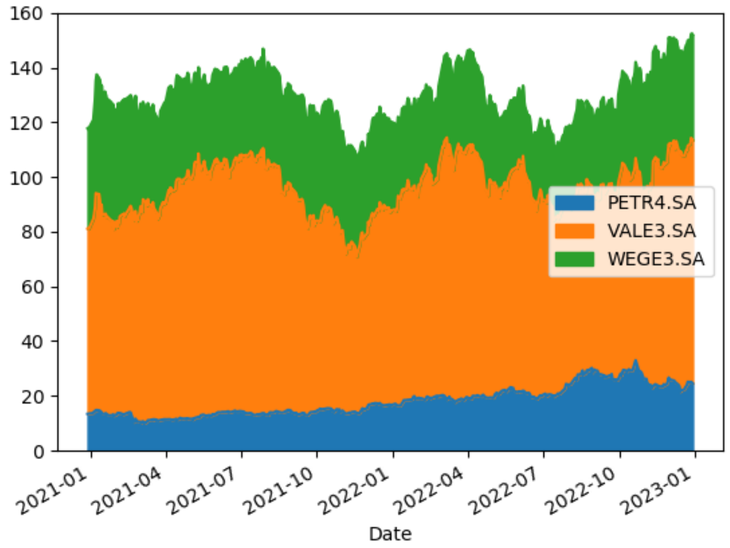
others:

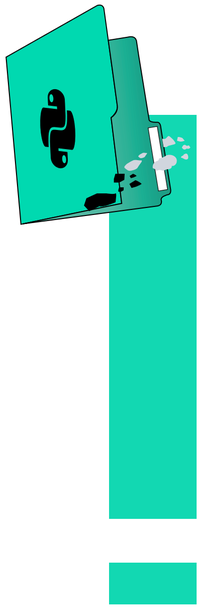
Muitos dos parâmetros do método DataFrame.plot() podem ser utilizados neste método.

Exemplo:

```
import yfinance as yf
import matplotlib.pyplot as plt

cotacoes = yf.download(["WEGE3.SA","PETR4.SA","VALE3.SA"],"2020-12-28", "2023-01-01")['Adj Close']

cotacoes.plot.area()

plt.show()
```

Respostas:





2.3.3. DataFrame.plot.bar(x = None, y = None, color = None, others)

Esse argumento adicional é utilizado para plotar um gráfico de barra vertical.

Parâmetros:

Os únicos parâmetros obrigatórios são: **x ou y**. Sendo que eles não precisam ser definidos juntos.

x:

Este parâmetro vai definir os dados do eixo x. Neste caso deve ser a coluna de um DataFrame. Só pode receber um valor.

É um parâmetro **opcional** que deve receber o nome de uma coluna de um DataFrame em string ou a posição de uma coluna de um DataFrame em integer.

y:

Este parâmetro vai definir os dados do eixo y. Neste caso deve ser a coluna de um DataFrame. Pode receber múltiplos valores.

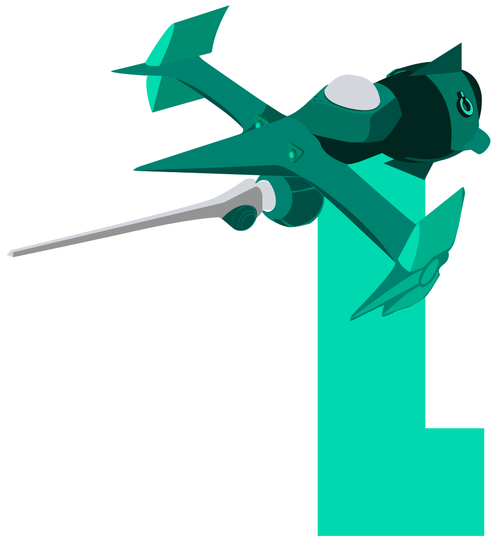
É um parâmetro **opcional** que deve receber o nome de uma, ou mais, colunas de um DataFrame em string ou a posição de uma, ou mais, colunas de um DataFrame em integer. Caso você passe mais de uma coluna, você deverá colocar as posições, ou nomes, dentro de uma lista.

color:

Este parâmetro vai definir as cores de cada coluna do DataFrame representada no gráfico. Por padrão, o pandas irá definir cores aleatórias para cada gráfico.

É um parâmetro **opcional** que receberá uma list com o nome, RGB ou RGBA de cada cor, no formato string. Pode ser também um dict com o nome da coluna e o valor da cor ao lado, no formato string.

others:

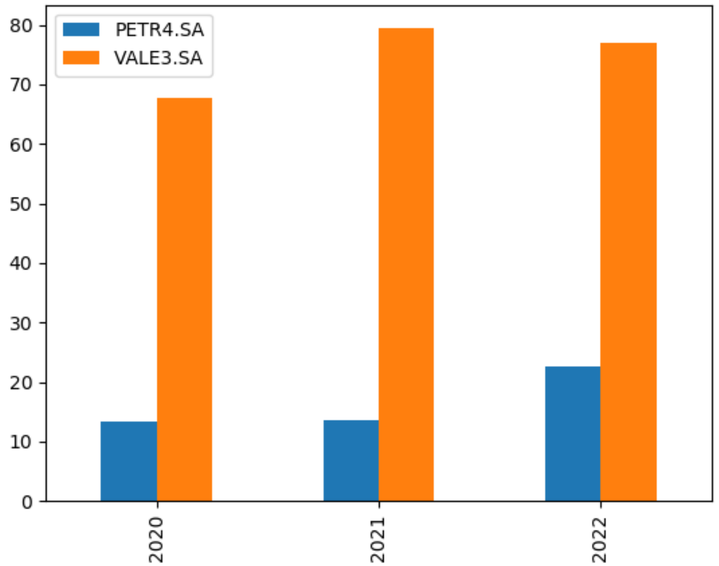


Muitos dos parâmetros do método DataFrame.plot() podem ser utilizados neste método.

Exemplo:

```
import yfinance as yf
import matplotlib.pyplot as plt

cotacoes = yf.download(["PETR4.SA","VALE3.SA"],"2020-12-28", "2023-01-01")['Adj Close']
cotacoes = cotacoes.dropna().resample("Y").mean()
cotacoes.index = ["2020","2021","2022"]
cotacoes.plot.bar()

plt.show()
```

Respostas:



2.3.4. DataFrame.plot.barh(x = None, y = None, color = None, others)

Esse argumento adicional é utilizado para plotar um gráfico de barra **horizontal**.

Parâmetros:

Os únicos parâmetros obrigatórios são: **x ou y** . Sendo que eles não precisam ser definidos juntos, pode ser um ou outro.

x:

Este parâmetro vai definir os dados do eixo x. Neste caso deve ser a coluna de um DataFrame. Só pode receber um valor.

É um parâmetro **opcional** que deve receber o nome de uma coluna de um DataFrame em string ou a posição de uma coluna de um DataFrame em integer.

y:

Este parâmetro vai definir os dados do eixo y. Neste caso deve ser a coluna de um DataFrame. Pode receber múltiplos valores.

É um parâmetro opcional que deve receber o nome de uma, ou mais, colunas de um DataFrame em string ou a posição de uma, ou mais, colunas de um DataFrame em integer. Caso você passe mais de uma coluna, você deverá colocar as posições, ou nomes, dentro de uma lista.

color:

Este parâmetro vai definir as cores de cada coluna do DataFrame representada no gráfico. Por padrão, o pandas irá definir cores aleatórias para cada gráfico.

É um parâmetro **opcional** que receberá uma list com o nome, RGB ou RGBA de cada cor, no formato string. Pode ser também um dict com o nome da coluna e o valor da cor ao lado, no formato string.

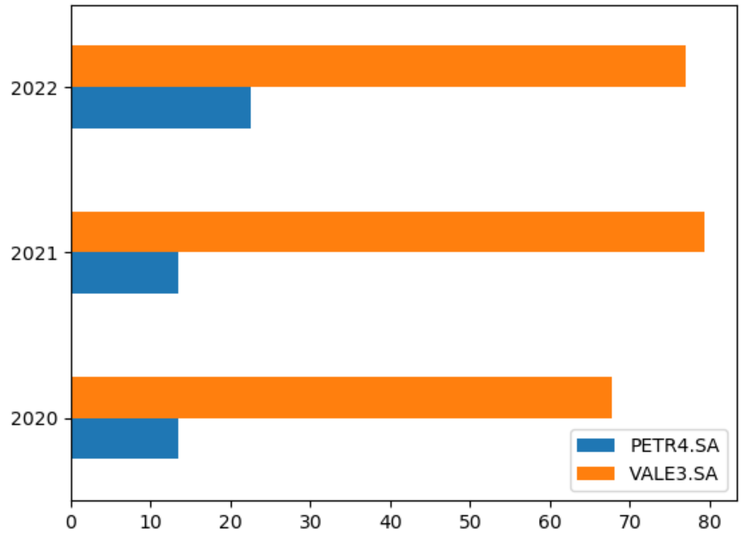
others:

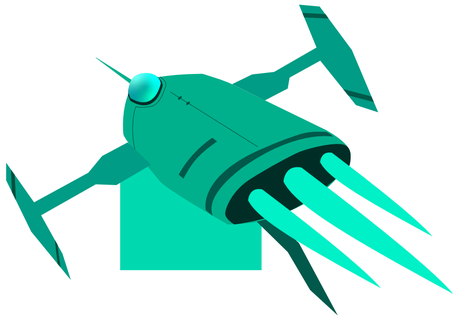
Muitos dos parâmetros do método DataFrame.plot() podem ser utilizados neste método.

Exemplo:

```
import yfinance as yf
import matplotlib.pyplot as plt

cotacoes = yf.download(["PETR4.SA","VALE3.SA"],"2020-12-28", "2023-01-01")['Adj Close']
cotacoes = cotacoes.dropna().resample("Y").mean()
cotacoes.index = ["2020","2021","2022"]
cotacoes.plot.barh()

plt.show()
```

Respostas:





2.3.5. DataFrame.plot.box(by = None, others)

Esse argumento adicional é utilizado para plotar um gráfico boxplot, utilizado para representar grupos de dados numéricos por meio de seus quartis. Sua forma de trabalhar é parecida com o DataFrame.groupby() pois ele faz agrupamento de dados.

Parâmetros:

by:

Este parâmetro vai determinar qual será a coluna do DataFrame a ser feito o agrupamento e depois plotada.

É um parâmetro **opcional** que deve receber o nome de uma, ou mais, colunas de um DataFrame em string.

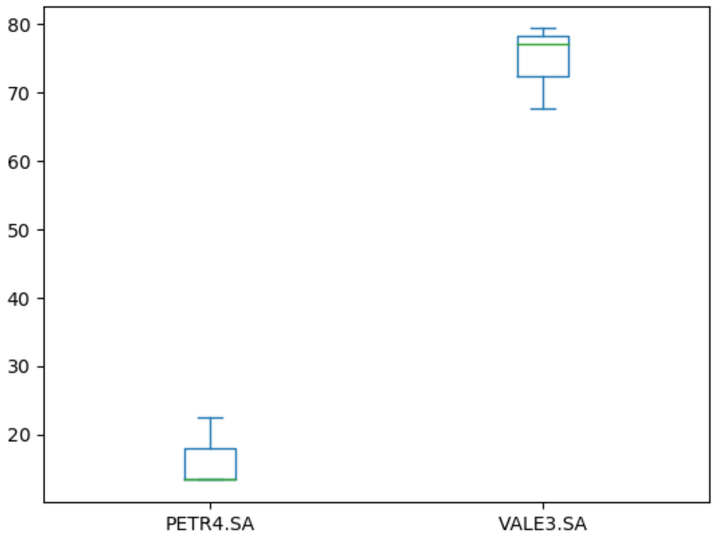
others:

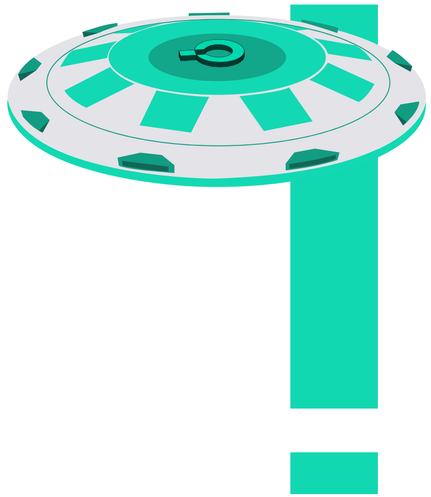
Muitos dos parâmetros do método DataFrame.plot() podem ser utilizados neste método.

Exemplo:

```
import yfinance as yf
import matplotlib.pyplot as plt

cotacoes = yf.download(["PETR4.SA","VALE3.SA"],"2020-12-28", "2023-01-01")['Adj Close']
cotacoes = cotacoes.dropna().resample("Y").mean()
cotacoes.index = ["2020","2021","2022"]
cotacoes.plot.box()

plt.show()
```

Respostas:





2.3.6. DataFrame.plot.density(bw_method = None, ind = None, others)

Esse argumento adicional é utilizado para plotar um gráfico de densidade. Que tem como função avaliar a função probabilidade de uma variável aleatória, esse método utiliza a distribuição de gauss (ou distribuição normal).

bw_method:

Este parâmetro vai determinar qual será a largura da banda do gráfico plotado.

É um parâmetro **opcional** que deve receber como será definido a largura da banda. Pode ser um valor definido por você, no formato integer. Pode ser também um valor pré definido pelo pandas, no formato string. Esses valores são “scott” e “silverman”, esses valores são funções utilizadas para achar a largura da banda.

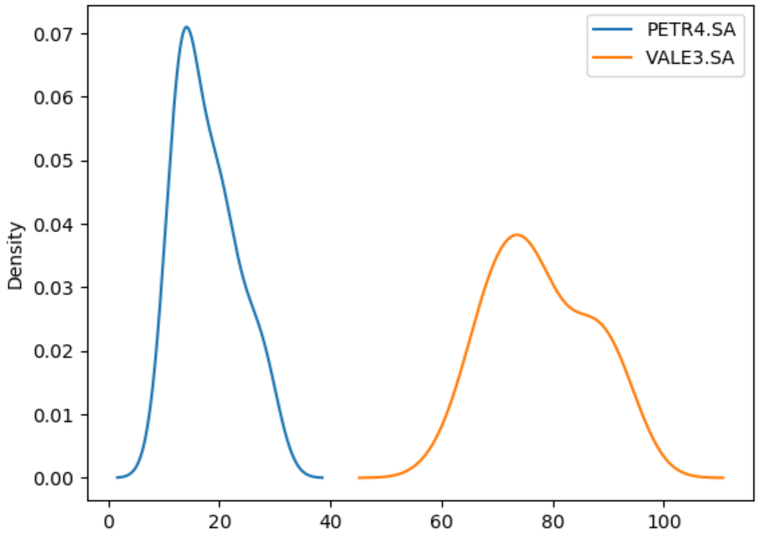
others:

Muitos dos parâmetros do método DataFrame.plot() podem ser utilizados neste método.

Exemplo:

```
import yfinance as yf
import matplotlib.pyplot as plt

cotacoes = yf.download(["PETR4.SA","VALE3.SA"],"2020-12-28", "2023-01-01")['Adj Close']
cotacoes = cotacoes.dropna().resample("m").mean()
cotacoes.plot.density()

plt.show()
```

Respostas:



2.3.7. DataFrame.plot.hexbin(x, y, C = None, gridsize = None, others)

Esse argumento adicional é utilizado para plotar um gráfico hexagonal binning.

x:

Este parâmetro vai definir os dados do eixo x. Neste caso deve ser a coluna de um DataFrame. Só pode receber um valor.

É um parâmetro **opcional** que deve receber o nome de uma coluna de um DataFrame em string ou a posição de uma coluna de um DataFrame em integer.

y:

Este parâmetro vai definir os dados do eixo y. Neste caso deve ser a coluna de um DataFrame. Pode receber múltiplos valores.

É um parâmetro **opcional** que deve receber o nome de uma, ou mais, colunas de um DataFrame em string ou a posição de uma, ou mais, colunas de um DataFrame em integer. Caso você passe mais de uma coluna, você deverá colocar as posições, ou nomes, dentro de uma lista.

gridsize:

Este parâmetro vai definir a quantidade de hexágonos nas direções x e y.

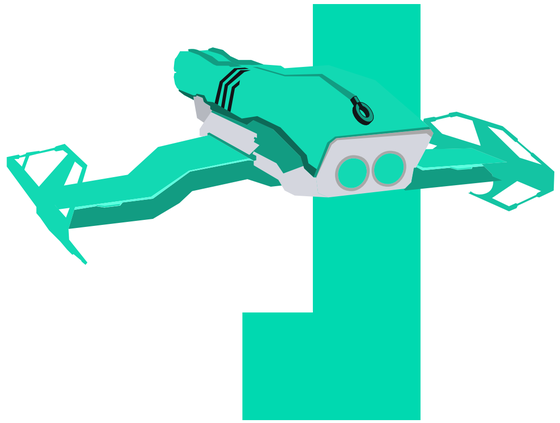
É um parâmetro **opcional** que deve receber uma tuple com a quantidade de hexágonos nas direções x e y, em integer, (x , y).

others:

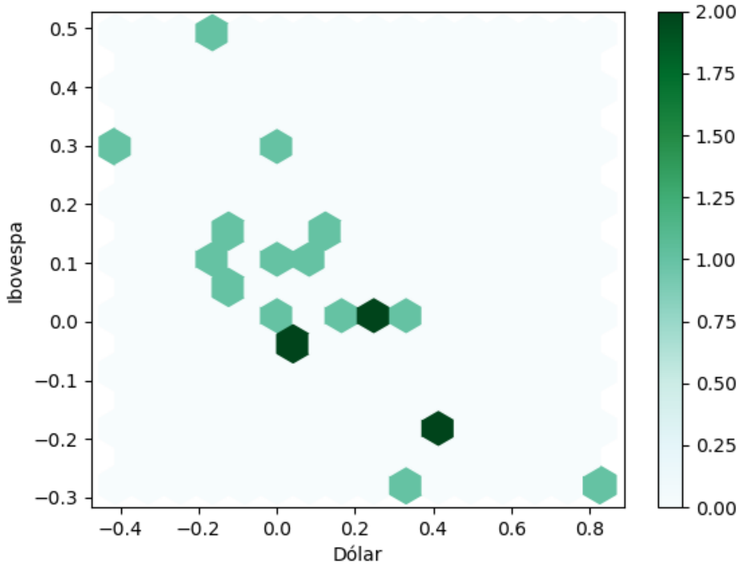
Muitos dos parâmetros do método DataFrame.plot() podem ser utilizados neste método.

Exemplo:

```python
import yfinance as yf
import matplotlib.pyplot as plt

cotacoes = yf.download(["^BVSP", "BRL=X"])['Close']
cotacoes = cotacoes.resample("Y").last().pct_change().dropna()

cotacoes['ano'] = cotacoes.index.year
cotacoes['ano'] = cotacoes['ano'].astype("category")
cotacoes.columns = ["Ibovespa", "Dólar", "Ano"]

cotacoes.plot.hexbin(x = "Dólar", y = "Ibovespa", gridsize=15)

plt.show()
```



Respostas:



2.3.8. DataFrame.plot.hist(by = None, bins = 10, others)

Esse argumento adicional é utilizado para plotar um histograma

by:

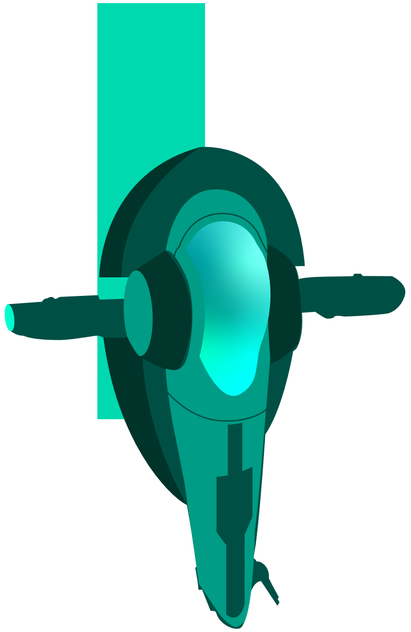
Este parâmetro vai determinar qual será a coluna do DataFrame a ser feito o agrupamento e depois plotada.

É um parâmetro **opcional** que deve receber o nome de uma, ou mais, colunas de um DataFrame em string.

bins:

Este parâmetro vai definir o número de colunas a serem usadas no gráfico plotado. Por padrão, o pandas define bins = 10, ou seja, o gráfico será dividido em 10 colunas.

É um parâmetro **opcional** que recebe o número de colunas no formato integer.



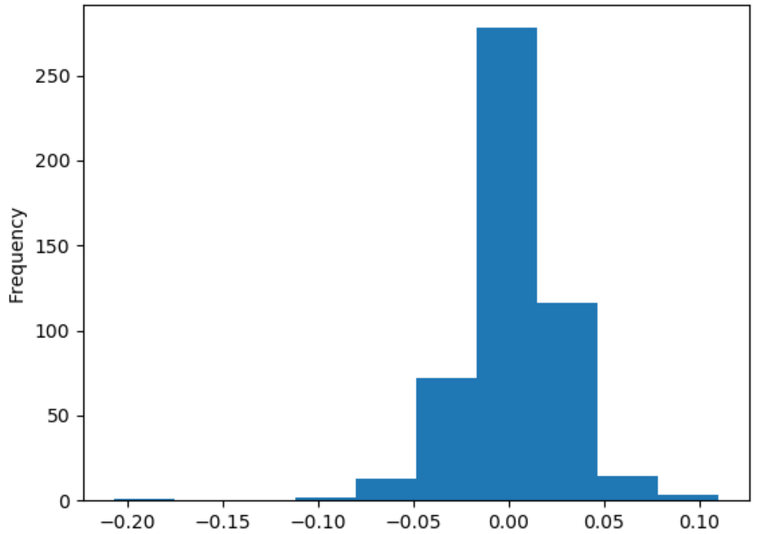
others:

Muitos dos parâmetros do método DataFrame.plot() podem ser utilizados neste método.

Exemplo:

```
import yfinance as yf
import matplotlib.pyplot as plt

cotacoes = yf.download(["PETR4.SA"],"2020-12-28", "2023-01-01")['Adj Close']
retornos_diarios = cotacoes.pct_change().dropna()

retornos_diarios.plot.hist()
plt.show()
```

Respostas:



2.3.9. DataFrame.plot.kde(bw_method = None, ind = None, others)

Esse argumento adicional é utilizado para plotar um gráfico de estimativa de densidade kernel (kde).

bw_method:

Este parâmetro vai determinar qual será a largura da banda do gráfico plotado.

É um parâmetro **opcional** que deve receber como será definido a largura da banda. Pode ser um valor definido por você, no formato integer. Pode ser também um valor pré definido pelo pandas, no formato string. Esses valores são “scott” e “silverman”, esses valores são funções utilizadas para achar a largura da banda.

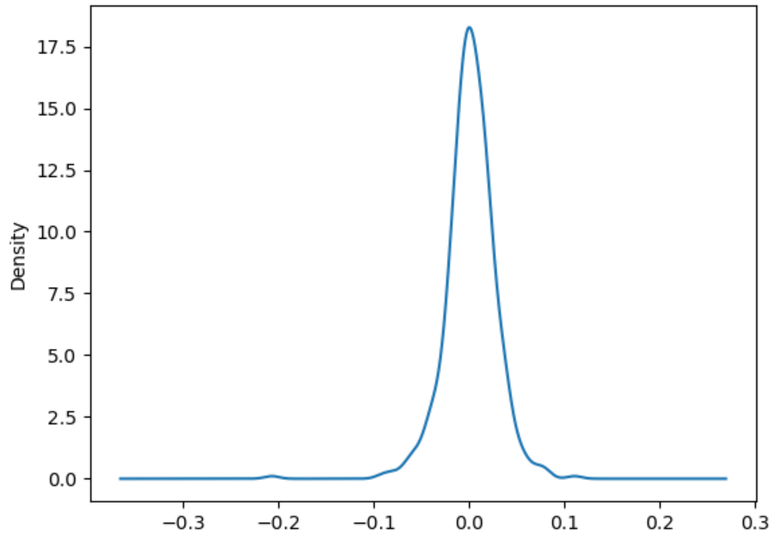
others:

Muitos dos parâmetros do método DataFrame.plot() podem ser utilizados neste método.

Exemplo:

```
import yfinance as yf
import matplotlib.pyplot as plt

cotacoes = yf.download(["PETR4.SA"],"2020-12-28", "2023-01-01")['Adj Close']
retornos_diarios = cotacoes.pct_change().dropna()

retornos_diarios.plot.kde()
plt.show()
```

Respostas:



2.3.10. DataFrame.plot.line(x = None, y = None, color = None, others)

Esse argumento adicional é utilizado para plotar um gráfico de linha.

Parâmetros:

Os únicos parâmetros obrigatórios são: **x ou y**. Sendo que eles não precisam ser definidos juntos, pode ser um ou outro.

x:

Este parâmetro vai definir os dados do eixo x. Neste caso deve ser a coluna de um DataFrame. Só pode receber um valor.

É um parâmetro **opcional** que deve receber o nome de uma coluna de um DataFrame em string ou a posição de uma coluna de um DataFrame em integer.

y:

Este parâmetro vai definir os dados do eixo y. Neste caso deve ser a coluna de um DataFrame. Pode receber múltiplos valores.

É um parâmetro **opcional** que deve receber o nome de uma, ou mais, colunas de um DataFrame em string ou a posição de uma, ou mais, colunas de um DataFrame em integer. Caso você passe mais de uma coluna, você deverá colocar as posições, ou nomes, dentro de uma lista.

color:

Este parâmetro vai definir as cores de cada coluna do DataFrame representada no gráfico. Por padrão, o pandas irá definir cores aleatórias para cada gráfico.

É um parâmetro **opcional** que receberá uma list com o nome, RGB ou RGBA de cada cor, no formato string. Pode ser também um dict com o nome da coluna e o valor da cor ao lado, no formato string.

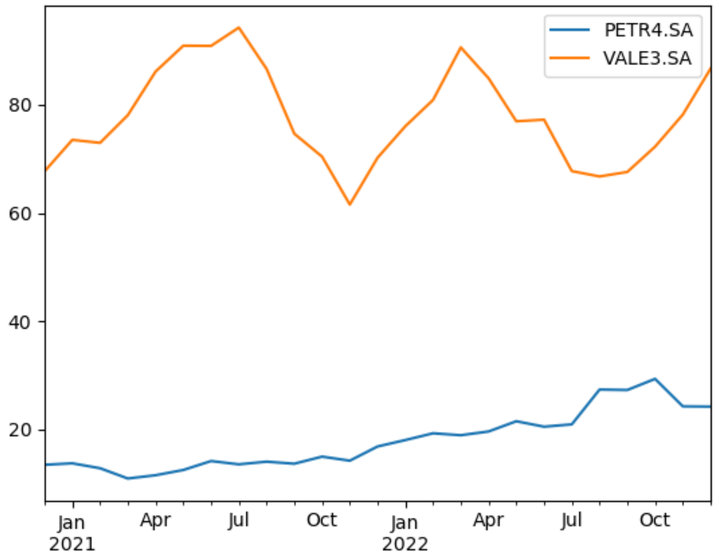
others:

Muitos dos parâmetros do método DataFrame.plot() podem ser utilizados neste método.

Exemplo:

```
import yfinance as yf
import matplotlib.pyplot as plt

cotacoes = yf.download(["PETR4.SA","VALE3.SA"],"2020-12-28", "2023-01-01")['Adj Close']
cotacoes = cotacoes.dropna().resample("m").mean()
cotacoes.plot.line()

plt.show()
```

Respostas:



2.3.11. DataFrame.plot.pie(y = None, others)

Esse argumento adicional é utilizado para plotar um gráfico de pizza.

Parâmetros:

y:

Este parâmetro vai definir os dados do eixo y. Neste caso deve ser a coluna de um DataFrame. Pode receber múltiplos valores.

É um parâmetro **opcional** que deve receber o nome de uma, ou mais, colunas de um DataFrame em string ou a posição de uma, ou mais, colunas de um DataFrame em integer. Caso você passe mais de uma coluna, você deverá colocar as posições, ou nomes, dentro de uma lista.

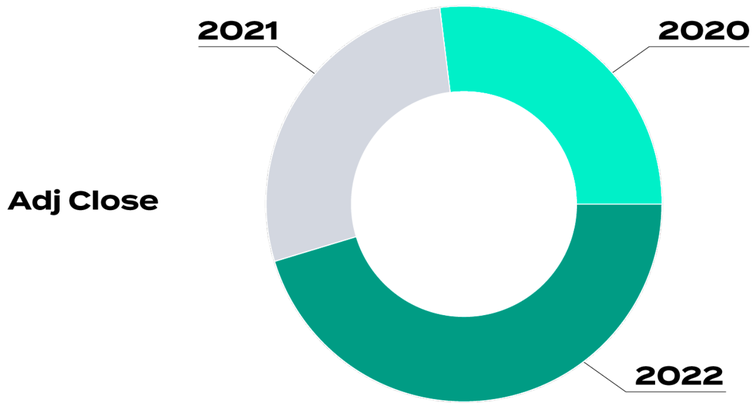
others:

Muitos dos parâmetros do método DataFrame.plot() podem ser utilizados neste método.

Exemplo:

```
import yfinance as yf
import matplotlib.pyplot as plt

cotacoes = yf.download(["PETR4.SA"],"2020-12-28", "2023-01-01")['Adj Close']
cotacoes = cotacoes.dropna().resample("y").mean()
cotacoes.index = ['2020','2021','2022']
cotacoes.plot.pie()

plt.show()
```

Respostas:



2.3.12. DataFrame.plot.scatter(x, y, s = None, c = None, others)

Esse argumento adicional é utilizado para plotar um gráfico de dispersão.

Parâmetros:

x:

Este parâmetro vai definir os dados do eixo x. Neste caso deve ser a coluna de um DataFrame. Só pode receber um valor.

É um parâmetro **opcional** que deve receber o nome de uma coluna de um DataFrame em string ou a posição de uma coluna de um DataFrame em integer.

y:

Este parâmetro vai definir os dados do eixo y. Neste caso deve ser a coluna de um DataFrame. Pode receber múltiplos valores.

É um parâmetro **opcional** que deve receber o nome de uma, ou mais, colunas de um DataFrame em string ou a posição de uma, ou mais, colunas de um DataFrame em integer. Caso você passe mais de uma coluna, você deverá colocar as posições, ou nomes, dentro de uma lista.

s:

Este parâmetro vai definir o tamanho de cada ponto de dispersão.

É um parâmetro **opcional** que deve receber o tamanho de cada ponto de dispersão no formato integer.

c:

Este parâmetro vai definir a cor de cada ponto de dispersão.

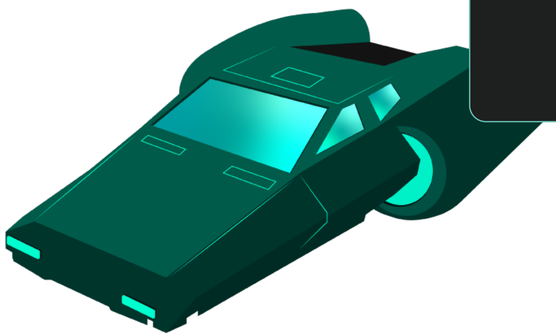
É um parâmetro **opcional** que deve receber o nome, RGB ou RGBA, da cor dos pontos, no formato string.

others:

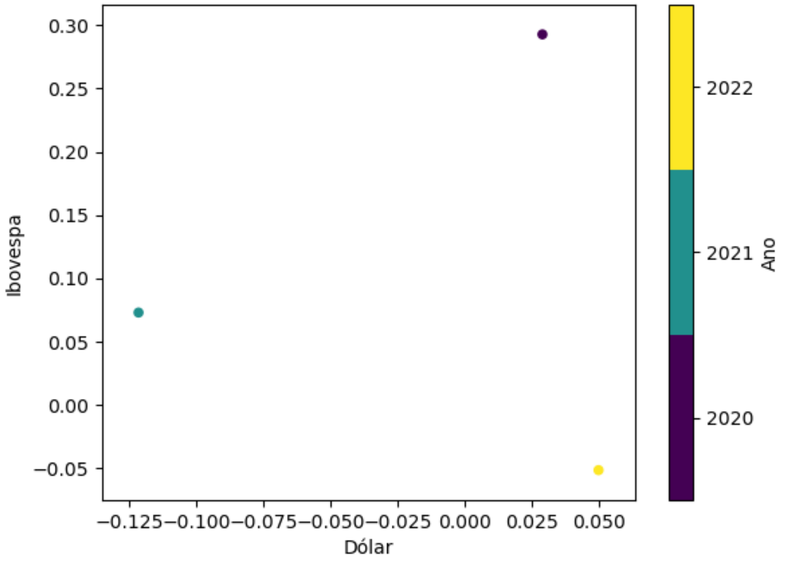
Muitos dos parâmetros do método DataFrame.plot() podem ser utilizados neste método.

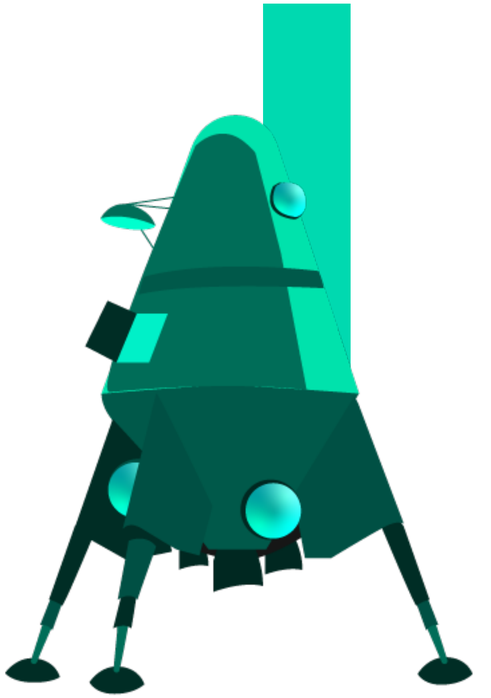
Exemplo:

```
import yfinance as yf
import matplotlib.pyplot as plt

cotacoes = yf.download(["^BVSP", "BRL=X"],"2019-01-01","2023-01-01")['Close']

cotacoes = cotacoes.resample("Y").last().pct_change().dropna()

cotacoes['ano'] = cotacoes.index.year
cotacoes['ano'] = cotacoes['ano'].astype("category")

cotacoes.columns = ["Ibovespa", "Dólar", "Ano"]

cotacoes.plot.scatter(x = "Dólar", y = "Ibovespa", c = "Ano",  cmap = "viridis")

plt.show()
```



Respostas:


0.30
0.25
0.20
0.15
0.10
0.05
0.00
−0.05
Ibovespa
−0.125 −0.100 −0.075 −0.050 −0.025 0.000 0.025 0.050
Dólar
2022
2021
2020
Ano



Mundo 3

Neste mundo aprenderemos sobre criação de gráficos, mas desta vez orientado a objeto. Desta forma há muito mais recursos na hora de plotar um gráfico.

3.1. matplotlib.pyplot.subplots(nrows = 1, ncols = 1, *, sharex = False, sharey = False, squeeze = True, width_ratios = None, height_ratios = None, subplot_kw = None, gridspec_kw = None)

Este método é usado para plotar dados em um gráfico. Este é um método da biblioteca matplotlib, do submódulo pyplot.

Parâmetros:

nrows:

Este parâmetro vai definir quantas linhas de gráfico terão na sua plotagem. Por padrão o pandas define nrows = 1, ou seja, por padrão ele só define 1 linha de gráfico. Este parâmetro trabalha em conjunto com o parâmetro “ncols”, para saber a quantidade de gráficos a serem plotados é só multiplicar o valor de um pelo valor de outro.

É um parâmetro **opcional** que deve receber a quantidade de linhas no formato integer.

ncol:

Este parâmetro vai definir quantas colunas de gráfico terão na sua plotagem. Por padrão o pandas define ncols = 1, ou seja, por padrão ele só define 1 coluna de gráfico. Este parâmetro trabalha em conjunto com o parâmetro “nrows”, para saber a quantidade de gráficos a serem plotados é só multiplicar o valor de um pelo valor de outro.

É um parâmetro **opcional** que deve receber a quantidade de colunas no formato integer.

sharex:

Este parâmetro vai definir se as propriedades do eixo x serão compartilhadas com todos os gráficos, ou seja, se todos eles terão propriedades iguais. Por padrão, o pandas define sharex = False, ou seja, não terão, necessariamente, propriedades iguais no eixo x.

É um parâmetro **opcional** que deve receber um booleano com True ou False. Pode receber também uma string com parâmetros pré definidos onde:

none = Nenhum gráfico terá propriedades compartilhadas

all = Todos os gráficos terão as propriedades compartilhadas

row = Todos os gráficos em linhas terão as propriedades compartilhadas

col = Todos os gráficos em colunas terão as propriedades compartilhadas

sharey:

Este parâmetro vai definir se as propriedades do eixo y serão compartilhadas com todos os gráficos, ou seja, se todos eles terão propriedades iguais. Por padrão, o pandas define sharey = False, ou seja, não terão, necessariamente, propriedades iguais no eixo y.

É um parâmetro **opcional** que deve receber um booleano com True ou False. Pode receber também uma string com parâmetros pré definidos onde:

none = Nenhum gráfico terá propriedades compartilhadas

all = Todos os gráficos terão as propriedades compartilhadas

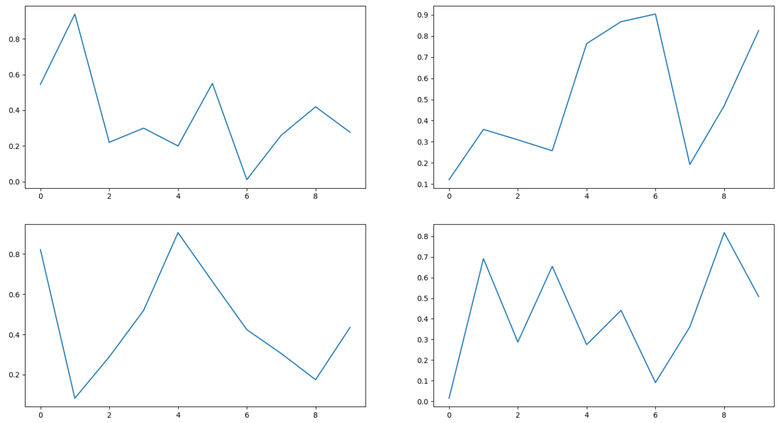
row = Todos os gráficos em linhas terão as propriedades compartilhadas

col = Todos os gráficos em colunas terão as propriedades compartilhadas.

Exemplo:

```
import numpy as np
import matplotlib.pyplot as plt

fig, ax = plt.subplots(2, 2, figsize = (20, 15))

for i in range(2):
    for j in range(2):

        ax[i, j].plot(np.random.random(10))

plt.show()
```

Respostas:



Mundo 4

Neste mundo aprenderemos sobre como modificar rótulos dentro dos gráficos, para os eixos x e y.

4.1. matplotlib.axes.Axes.set_ylabel(ylabel, labelpad = None, loc = None, others)

Este método é usado para colocar rótulos no eixo y em um gráfico. **Este é um método da biblioteca matplotlib**.

Parâmetros:

ylabel:

Este parâmetro vai definir o nome do rótulo do eixo y.

É um parâmetro **opcional** que deve receber o nome do eixo y no formato string.

labelpad:

Este parâmetro vai definir a distância do eixo até o rótulo.

É um parâmetro **opcional** que deve receber a distância no formato integer.

loc:

Este parâmetro vai definir a localização do rótulo no eixo. Por padrão, o pandas define loc = 'center', ou seja, o rótulo vai ficar no centro.

É um parâmetro **opcional** que deve receber a localização em locais pré definidos pelo matplotlib:

center = centro

top = em cima

bottom = embaixo

others:

O python, e principalmente o matplotlib, tem o objetivo de ser o mais prático possível, por isso eles "reutilizam" muitos parâmetros em vários métodos. Neste caso não seria diferente, este método reutiliza classes de textos que servem para controlar a aparência dos rótulos, parâmetros como: color, rotation, fontsize, etc, podem ser utilizados nestes casos.

4.2. matplotlib.axes.Axes.set_xlabel(ylabel, labelpad = None, loc = None, others)

Este método é usado para colocar rótulos no eixo x em um gráfico. **Este é um método da biblioteca matplotlib**.

Parâmetros:

xlabel:

Este parâmetro vai definir o nome do rótulo do eixo x.

É um parâmetro **opcional** que deve receber o nome do eixo y no formato string.

labelpad:

Este parâmetro vai definir a distância do eixo até o rótulo.

É um parâmetro **opcional** que deve receber a distância no formato integer.

loc:

Este parâmetro vai definir a localização do rótulo no eixo. Por padrão, o pandas define loc = 'center', ou seja, o rótulo vai ficar no centro.

É um parâmetro **opcional** que deve receber a localização em locais pré definidos pelo matplotlib:

center = centro

top = em cima

bottom = embaixo

others:

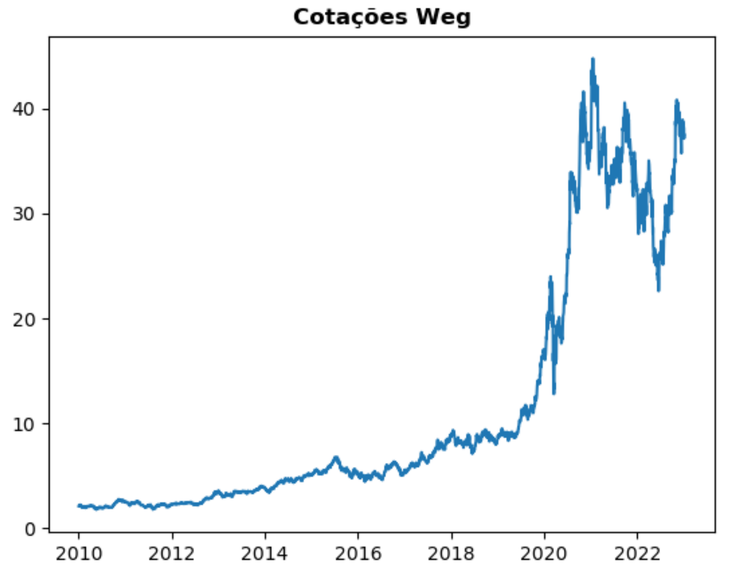
O python, e principalmente o matplotlib, tem o objetivo de ser o mais prático possível, por isso eles “reutilizam” muitos parâmetros em vários métodos. Neste caso não seria diferente, este método reutiliza classes de textos que servem para controlar a aparência dos rótulos, parâmetros como: color, rotation, fontsize, etc... podem ser utilizados nestes casos.

Antes de modificar os eixos:

Exemplo:

```
import yfinance as yf
import matplotlib.pyplot as plt

cotacao_weg = yf.download("WEGE3.SA", "2010-01-01")['Adj Close']

fig, ax = plt.subplots()

ax.plot(cotacao_weg.index, cotacao_weg.values)

plt.show()
```

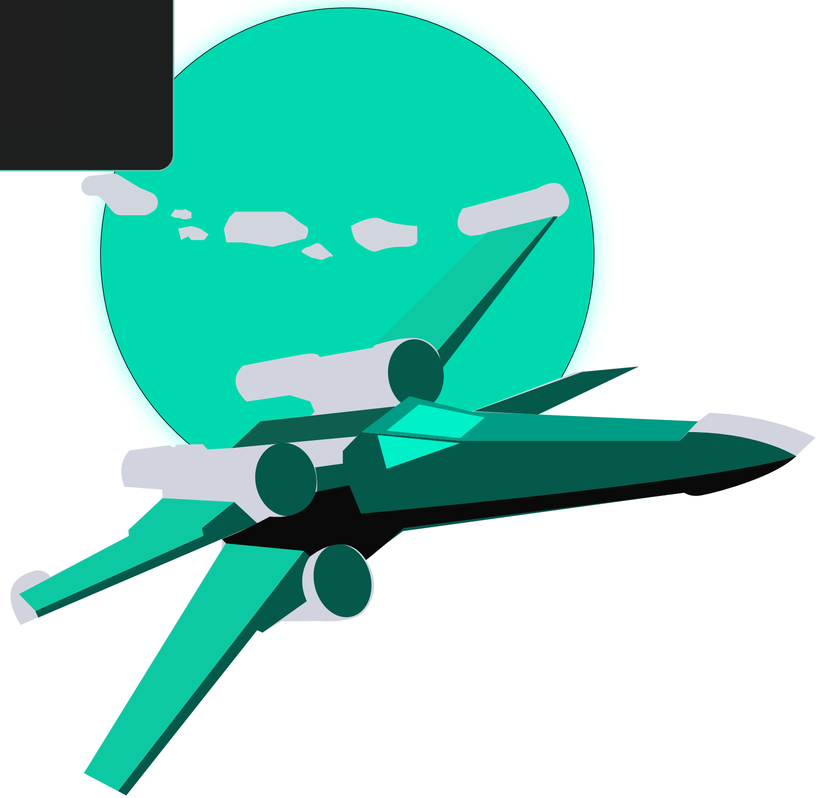
Respostas:



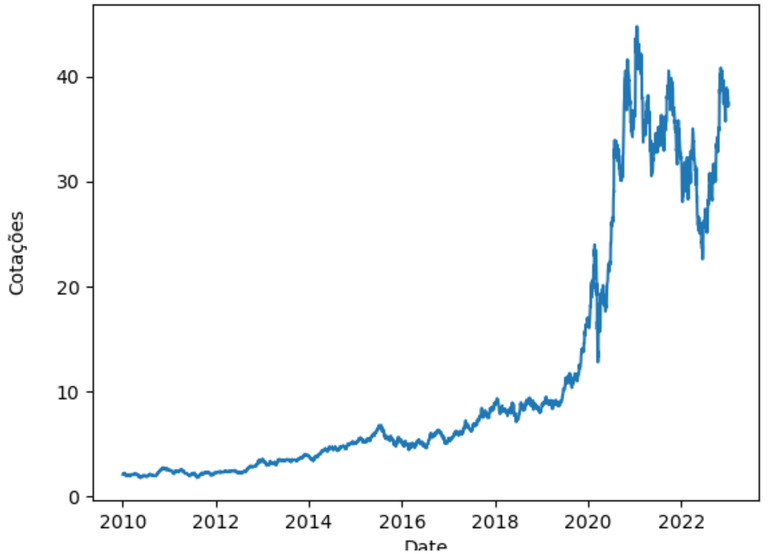
Depois de modificar os eixos:

Exemplo:

```
import yfinance as yf
import matplotlib.pyplot as plt

cotacao_weg = yf.download("WEGE3.SA", "2010-01-01")['Adj Close']

fig, ax = plt.subplots()

ax.plot(cotacao_weg.index, cotacao_weg.values)

ax.set_ylabel("Cotações", labelpad = 15, color='k')
ax.set_xlabel("Date", color='k')

plt.show()
```



Respostas:



Mundo 5

Neste mundo aprenderemos a como colocar legendas dentro de gráficos e como mexer em suas propriedades

5.1. matplotlib.axes.Axes.set_title(label, loc = 'center', fontdict = None, pad = None, others)

Este método é usado para colocar título em um gráfico. **Este é um método da biblioteca matplotlib**.

Parâmetros:

label:

Este parâmetro vai definir o nome do título do gráfico.

É um parâmetro **opcional** que deve receber o nome do título no formato string.

fontdict:

Este parâmetro é quem controla a aparência do título do gráfico. Ele é um dict que tem como padrão os seguintes argumentos:

{'fontsize': **(O tamanho da fonte no formato integer)**,

'fontweight': **(A formatação que pode ser "normal" ou "bold", no formato string)**,

'color': **(A cor que pode ser o nome da cor, RGA no formato string ou uma tuple com o (R,G,B))** ,

'verticalalignment': **(A posição vertical do título que pode receber os seguintes parâmetros no formato string: 'top', 'bottom', 'center', 'baseline', 'center_baseline')**,

'horizontalalignment': **(A posição vertical do título que pode receber os seguintes parâmetros no formato string: 'left', right, 'center')**}

É um parâmetro **opcional** que deve receber um dicionário com os parâmetros pré definidos, os parâmetros não precisam ser definidos todos de uma vez.

pad:

Este parâmetro vai definir a distância do eixo até o título. Por padrão, o matplotlib define que pad = 6, ou seja, a distância do título até o eixo vai ser de 6.

É um parâmetro **opcional** que deve receber a distância no formato integer.

y:

Este parâmetro vai definir a altura da posição do título. O 1 é o marco inicial, onde números menores que 1 abaixam a posição do título e números maiores que 1 aumentam a posição do título.

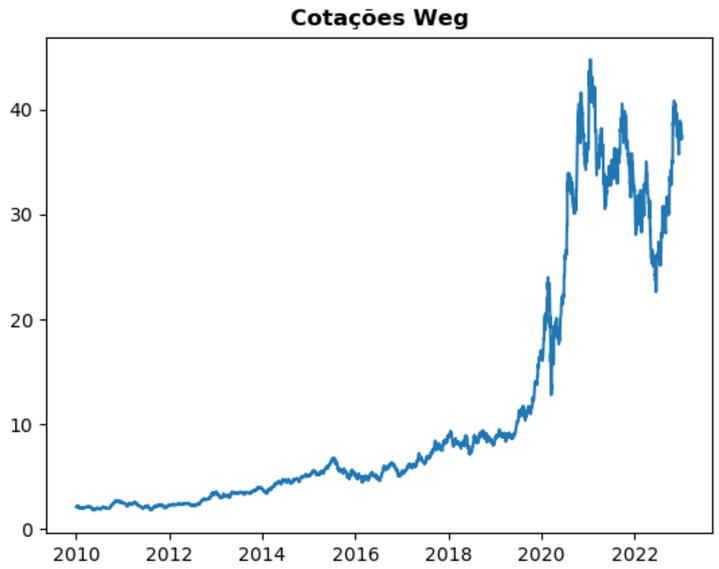
obs: Trabalhar com números perto de 1 por se tratar de uma escala sensível.

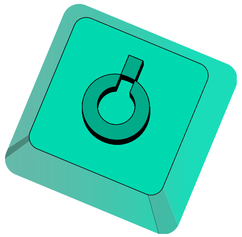
É um parâmetro **opcional** que deve receber a posição do título no formato float.

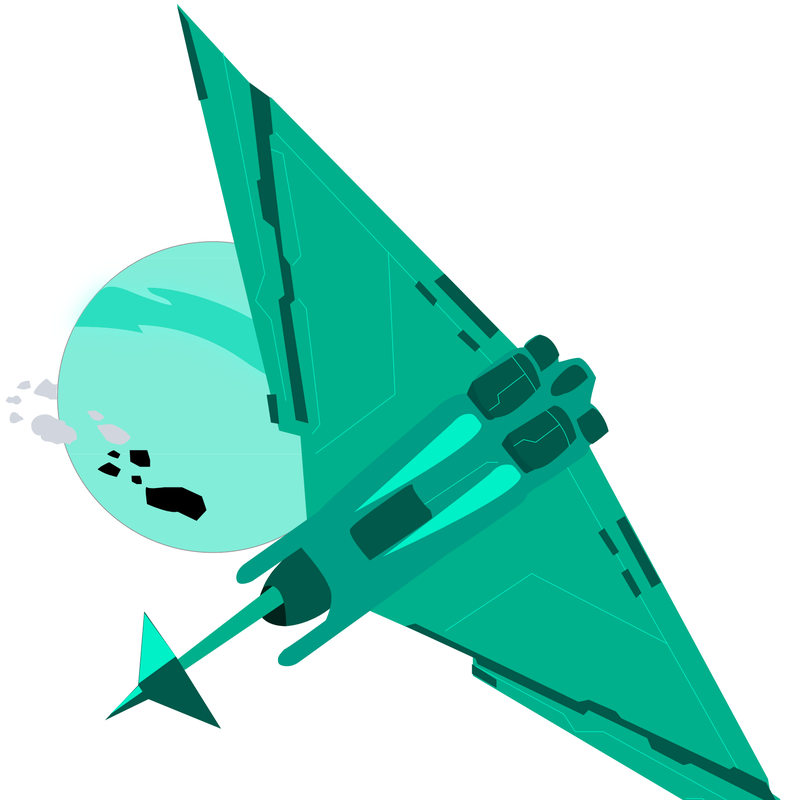
Exemplo:

```
import yfinance as yf
import matplotlib.pyplot as plt

cotacao_weg = yf.download("WEGE3.SA", "2010-01-01")['Adj Close']

fig, ax = plt.subplots()

ax.plot(cotacao_weg.index, cotacao_weg.values)

ax.set_ylabel("Cotações", labelpad = 15, color='k')
ax.set_xlabel("Date", color='k')

plt.show()
```

Respostas:







Mundo 6

Neste mundo aprenderemos a como criar eixos fixos e colocar o gráfico em escala logarítmica.

6.1. matplotlib.axes.Axes.set_xlim(left = None, right = None, auto = False, others)

Este método é usado para definir o intervalo de um gráfico. **Este é um método da biblioteca matplotlib**.

Parâmetros:

left:

Este parâmetro vai definir o começo do intervalo.

É um parâmetro **opcional** que deve receber o valor limite a ser definido no formato float.

right:

Este parâmetro vai definir o final do intervalo.

É um parâmetro **opcional** que deve receber o valor limite a ser definido no formato float.

auto:

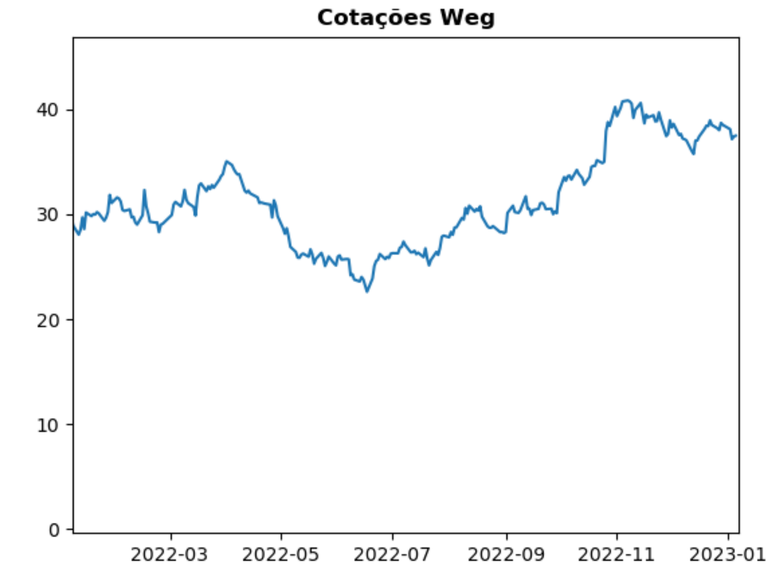
Este parâmetro vai definir o final do intervalo.

É um parâmetro **opcional** que deve receber o valor limite a ser definido no formato float.

Exemplo:

```python
import yfinance as yf
import matplotlib.pyplot as plt
from datetime import datetime, timedelta

cotacao_weg = yf.download("WEGE3.SA", "2010-01-01")['Adj Close']

um_ano_atras = datetime.now() - timedelta(days = 365)
hoje = datetime.now()

fig, ax = plt.subplots()

ax.plot(cotacao_weg.index, cotacao_weg.values)

ax.set_title("Cotações Weg", fontweight ="bold", color = "k")

ax.set_xlim(um_ano_atras,hoje)

plt.show()
```

Respostas:



6.2. matplotlib.axes.Axes.set_ylim(left = None, right = None, auto = False, others)

Este método é usado para definir o intervalo de um gráfico. **Este é um método da biblioteca matplotlib**.

Parâmetros:

left:

Este parâmetro vai definir o começo do intervalo.

É um parâmetro **opcional** que deve receber o valor limite a ser definido no formato float.

right:

Este parâmetro vai definir o final do intervalo.

É um parâmetro **opcional** que deve receber o valor limite a ser definido no formato float.

auto:

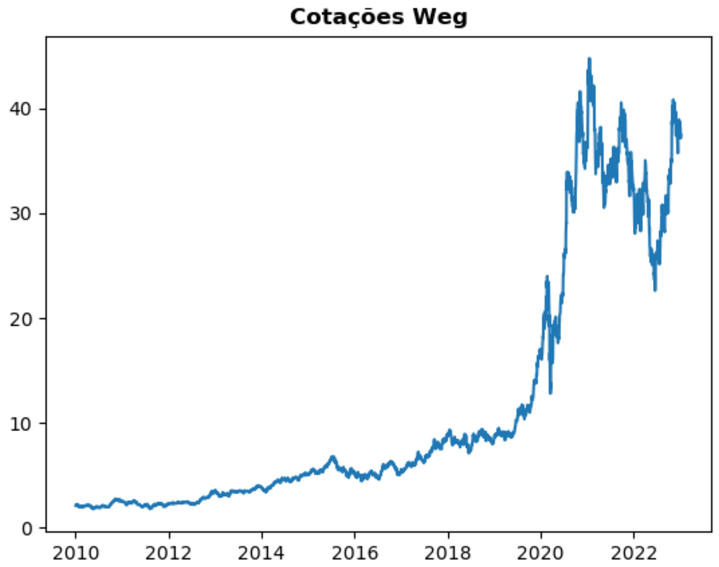
Este parâmetro vai definir o final do intervalo.

É um parâmetro **opcional** que deve receber o valor limite a ser definido no formato float.

Exemplo:

```
import yfinance as yf
import matplotlib.pyplot as plt

cotacao_weg = yf.download("WEGE3.SA", "2010-01-01")['Adj Close']

fig, ax = plt.subplots()

ax.plot(cotacao_weg.index, cotacao_weg.values)

ax.set_ylim(0,60)

plt.show()
```

Respostas:



6.3. matplotlib.axes.Axes.set_yscale(value = None)

Este método é usado para definir a escala a ser utilizada no seu gráfico. **Este é um método da biblioteca matplotlib**.

Parâmetros:

value:

Este parâmetro vai definir o começo do intervalo a ser definido.

É um parâmetro **opcional** que deve receber valores pré definidos no formato string.

linear = gráfico na escala linear

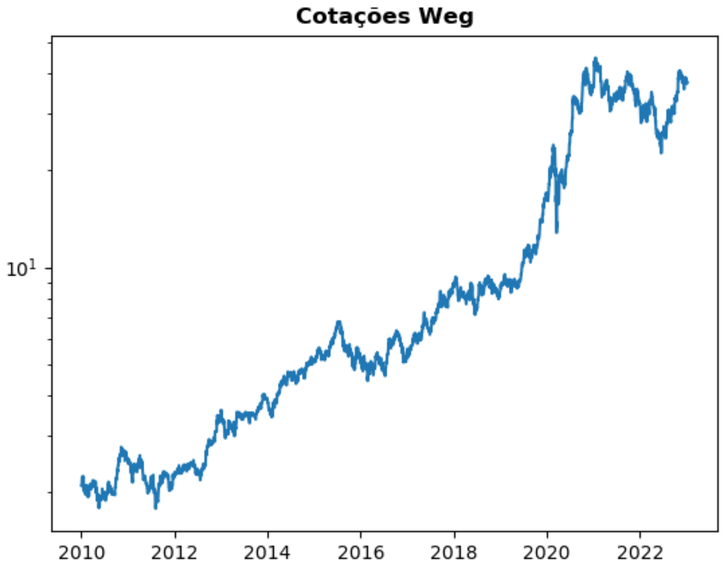
log = gráfico na escala logarítmica (os valores positivos)

symlog = gráfico na escala symlogarítmica (valores positivos e negativos)

Exemplo:

```
import yfinance as yf
import matplotlib.pyplot as plt
from datetime import datetime, timedelta

cotacao_weg = yf.download("WEGE3.SA", "2010-01-01")['Adj Close']

fig, ax = plt.subplots()

ax.plot(cotacao_weg.index, cotacao_weg.values)

ax.set_yscale(value="log")

plt.show()
```

Respostas:



Mundo 7

Neste mundo aprenderemos a como criar legendas para as informações do gráfico.

6.2. matplotlib.pyplot.legend(loc = 'upper right', bbox_to_anchor = None, fontsize = None, right = None, auto = False, prop = None, title = None, title_fontsize = None, border_pad = None, labelspacing = None, handlelenght = None, handleweight = None, handletextpad = None, borderaxespad = None, others)

Este método é usado para criação e edição das legendas das informações do gráfico. **Este é um método da biblioteca matplotlib**.

Parâmetros:

loc:

Este parâmetro vai definir a localização da legenda das informações no gráfico .

É um parâmetro **opcional** que deve receber a localização de acordo com a tabela predefinida que está no formato integer.

bbox_to_anchor:

Este parâmetro vai definir a posição da legenda no gráfico.

É um parâmetro **opcional** que deve receber uma tuple com a posição da legenda (para esquerda, para cima) no formato integer.

obs: Procure trabalhar com valores próximo de 1 porque os tamanhos são muito sensíveis.

fontsize:

Este parâmetro vai definir o tamanho da legenda das informações do gráfico.

É um parâmetro **opcional** que deve receber o tamanho em integer ou receber tamanhos de acordo com a tabela predefinida que está no formato string.

prop:

Este parâmetro vai definir as propriedades da legenda (tamanho, estilo, etc).

É um parâmetro **opcional** que deve receber um dict com as propriedades que deseja modificar no formato string e seus respectivos valores.

{
"style": Estilo das letras que podem ser: "normal", "italic" ou "oblique",

"size": Tamanho das letras no formato string de acordo com a tabela predefinida ou o tamanho no formato integer,

"weight": A densidade das letras que podem ser: "normal" ou "bold"
}

title:

Este parâmetro vai definir qual será o nome da legenda.

É um parâmetro **opcional** que deve receber o valor no formato string.

title_fontsize:

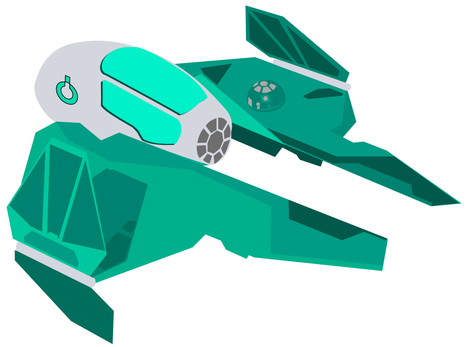
Este parâmetro vai definir o tamanho da legenda das informações do gráfico .

É um parâmetro **opcional** que deve receber o tamanho em integer ou receber tamanhos de acordo com a tabela predefinida que está no formato string.

borderpad:

Este parâmetro vai definir o espaçamento da borda dentro da legenda. Por padrão, o matplotlib define borderpad = 0.4.

É um parâmetro **opcional** que deve receber um número no formato float.



labelspacing:

Este parâmetro vai definir o espaçamento entre as informações da legenda. Por padrão o pandas define labelspacing = 0.5.

É um parâmetro **opcional** que deve receber um número no formato float.

handlelenght:

Este parâmetro vai definir o tamanho das linhas que representam o gráfico. Por padrão, o pandas define handlelenght = 2.0.

É um parâmetro **opcional** que deve receber um número no formato float.

handleheight:

Este parâmetro vai definir o tamanho das distâncias das linhas que representam o gráfico. Por padrão, o pandas define handleheight = 0.7.

É um parâmetro **opcional** que deve receber um número no formato float.

handletextpad:

Este parâmetro vai definir o tamanho das distâncias das linhas que representam o gráfico e os nomes das linhas. Por padrão, o pandas define handletextpad = 0.8.

É um parâmetro **opcional** que deve receber um número no formato float.

borderaxespad:

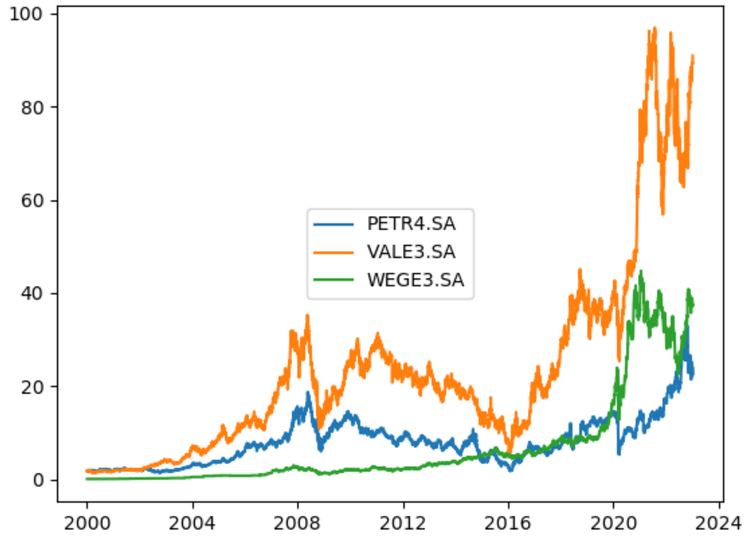
Este parâmetro vai definir o tamanho das distâncias entre o eixo superior e o eixo da direita. Por padrão, o pandas define handletextpad = 0.5.

É um parâmetro **opcional** que deve receber um número no formato float.

Exemplo:

```
import yfinance as yf
import matplotlib.pyplot as plt

cotacoes = yf.download(["WEGE3.SA", "PETR4.SA", "VALE3.SA"])['Adj Close']

fig, ax = plt.subplots()

for empresa in cotacoes.columns:

  ax.plot(cotacoes.index, cotacoes[empresa].values, label = empresa)

ax.legend(loc = 10)

plt.show()
```

Respostas:



Mundo 8

Neste mundo aprenderemos a como modificar itens presentes nos eixos.

8.1. matplotlib.axes.Axes.tick_params(axis = “both”, direction = “out”, color = “r”, pad = None, labelsize = None, labelsize = “black”, top = False, bottom = True, left = True, right = False, labeltop = False, labelbottom = True, labelleft = False, labelright = True)

Este método é usado para edição dos eixos do gráfico. **Este é um método da biblioteca matplotlib**.

Parâmetros:

axis:

Este parâmetro vai definir o eixo que sofrerá a modificação.

É um parâmetro **opcional** que deve receber valores pré definidos no formato string (“x” , “y” , “both”), sendo “x” em relação ao eixo x, sendo “y” em relação ao eixo y e sendo “both” em relação aos dois eixos.

direction:

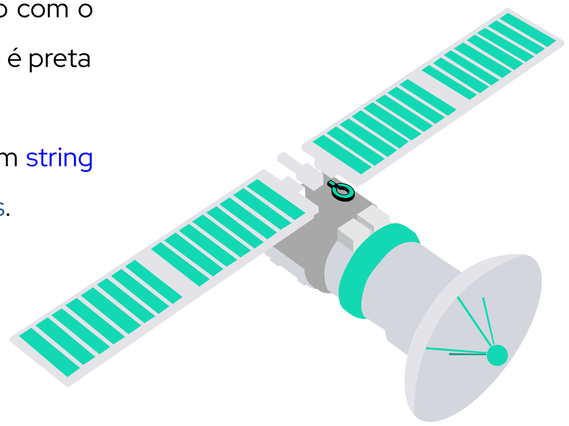
Este parâmetro vai definir a onde o traço que referencia os valores estará.

É um parâmetro **opcional** que deve receber valores pré definidos no formato string (“out” , “in” , “inout”), sendo “out” o traço fora da grade, sendo “in” o traço na grade e sendo “inout” o traço na grade.

color:

Este parâmetro vai definir a cor da série de dados de acordo com o formato de cores. Por padrão o pandas define que a cor padrão é preta

É um parâmetro **opcional** que deve receber a formatação em string com formatos pré-definidos que obedecem o formato de cores.



pad:

Este parâmetro vai definir o espaçamento dos valores do gráfico.

É um parâmetro **opcional** que deve receber o valor da distância no formato float.

labelsize:

Este parâmetro vai definir o tamanho dos valores do eixo (apenas os números).

É um parâmetro **opcional** que deve receber o tamanho no formato float.

labelcolor:

Este parâmetro vai definir a cor dos valores do eixo (apenas os números).

É um parâmetro **opcional** que deve receber a formatação em string com formatos pré-definidos que obedecem o formato de cores.

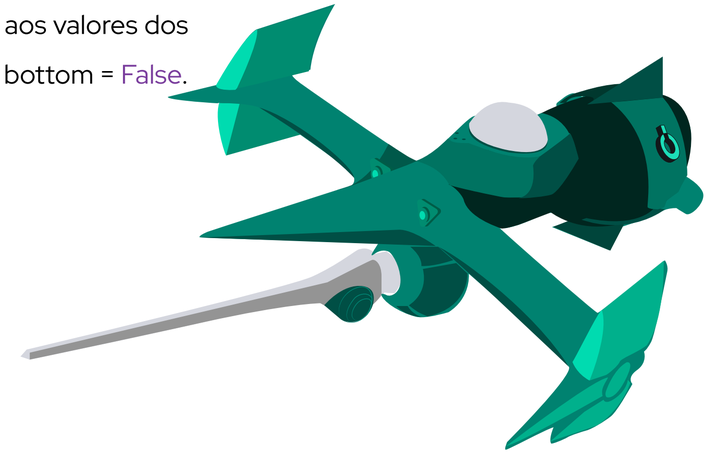
top:

Este parâmetro vai definir se terá traços referente aos valores dos eixos na parte de cima. Por padrão, o pandas define top = False.

É um parâmetro **opcional** que receberá um booleano com True ou False.

bottom:

Este parâmetro vai definir se terá traços referente aos valores dos eixos na parte de baixo. Por padrão, o pandas define bottom = False.



É um parâmetro **opcional** que receberá um booleano com True ou False.

left:

Este parâmetro vai definir se terá traços referente aos valores dos eixos na parte da esquerda. Por padrão, o pandas define left = True.

É um parâmetro **opcional** que receberá um booleano com True ou False.

right:

Este parâmetro vai definir se terá traços referente aos valores dos eixos na parte de cima. Por padrão, o pandas define right = False.

É um parâmetro **opcional** que receberá um booleano com True ou False.

labeltop:

Este parâmetro vai definir se terá os valores dos eixos na parte de cima. Por padrão, o pandas define top = False.

É um parâmetro **opcional** que receberá um booleano com True ou False.

labelbottom:

Este parâmetro vai definir se terá os valores dos eixos na parte de baixo. Por padrão, o pandas define bottom = False.

É um parâmetro opcional que receberá um booleano com True ou False.

labelleft:

Este parâmetro vai definir se terá os valores dos eixos na parte da esquerda. Por padrão, o pandas define left = True.

É um parâmetro **opcional** que receberá um booleano com True ou False.

labelright:

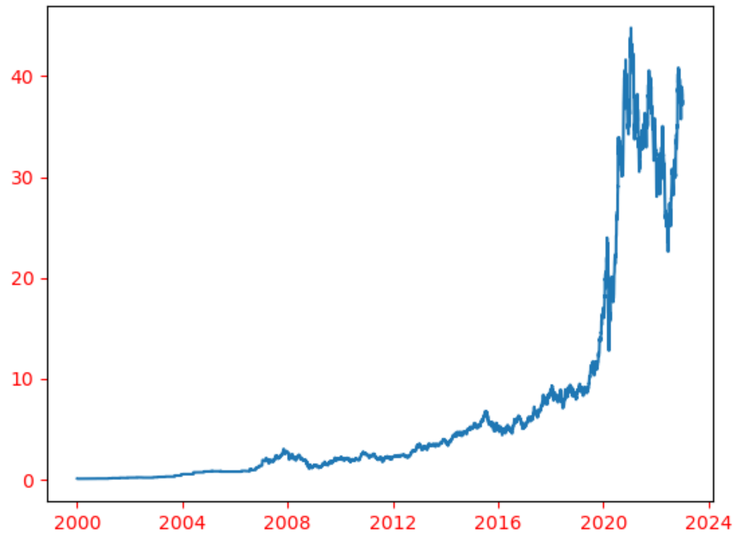
Este parâmetro vai definir se terá os valores dos eixos na parte de cima. Por padrão, o pandas define right = False.

É um parâmetro **opcional** que receberá um booleano com True ou False.

Exemplo:

```
import yfinance as yf
import matplotlib.pyplot as plt

cotacoes = yf.download(["WEGE3.SA"])['Adj Close']

fig, ax = plt.subplots()

ax.plot(cotacoes.index, cotacoes.values)

ax.tick_params(axis='x', colors='r')
ax.tick_params(axis='y', colors='r')

plt.show()
```

Respostas:



8.2. matplotlib.axes.Axes.set_major_formatter(formatter = None)

Este método é usado para definir a formatação dos valores dos eixos. **Este é um método da biblioteca matplotlib**.

Neste método será apresentado duas formas de resolução. Pois dependendo da versão da biblioteca matplotlib instalada no seu computador, uma das duas versões podem não funcionar.

1° Forma

Parâmetros:

formatter:

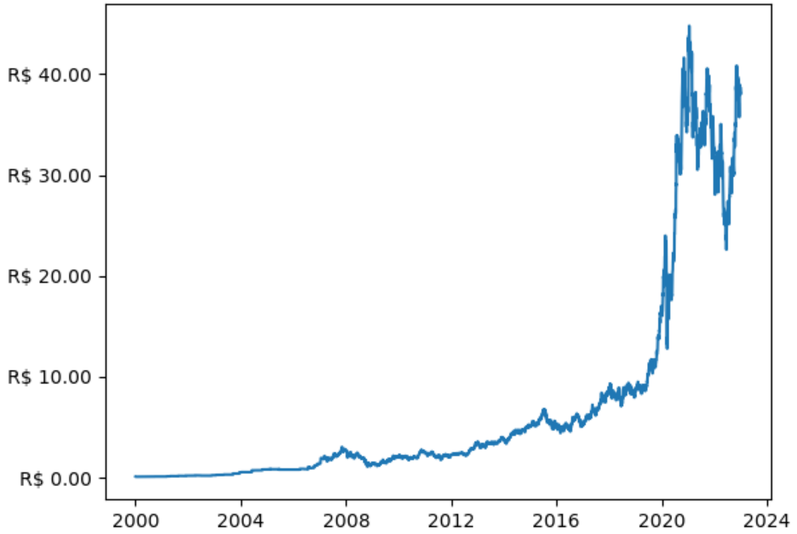
Este parâmetro vai definir o formato que você deseja utilizar. Neste primeiro caso só passar a forma em que deseja visualizar o número, sendo "x" a representação do seu número. Então por exemplo, se deseja seus valores em reais, apenas defina formatter = "R$ {x}", se deseja em quilômetros, apenas defina formatter = "{x} km".

É um parâmetro **opcional** que deve receber a formatação desejada dos valores no formato string.

Exemplo:

```
import matplotlib.pyplot as plt
import yfinance as yf


cotacoes = yf.download(["WEGE3.SA"])['Adj Close']
fig, ax = plt.subplots()
ax.plot(cotacoes.index, cotacoes.values)

ax.yaxis.set_major_formatter(formatter='R${x:1.0f}')
plt.show()
```

Respostas:



2° Forma

Parâmetros:

formatter:

Este parâmetro vai definir o formato que você deseja utilizar. Neste segundo caso você precisará criar uma função que retorne a formatação desejada. Após isso, precisará utilizar um submódulo do matplotlib para que consiga interpretar essa função do jeito desejado. Então por exemplo, se deseja seus valores em reais, você deve fazer uma função que leia um valor e retorne "R$ {x}", se deseja em quilômetros, faça o mesmo mas retornando "{x} km".

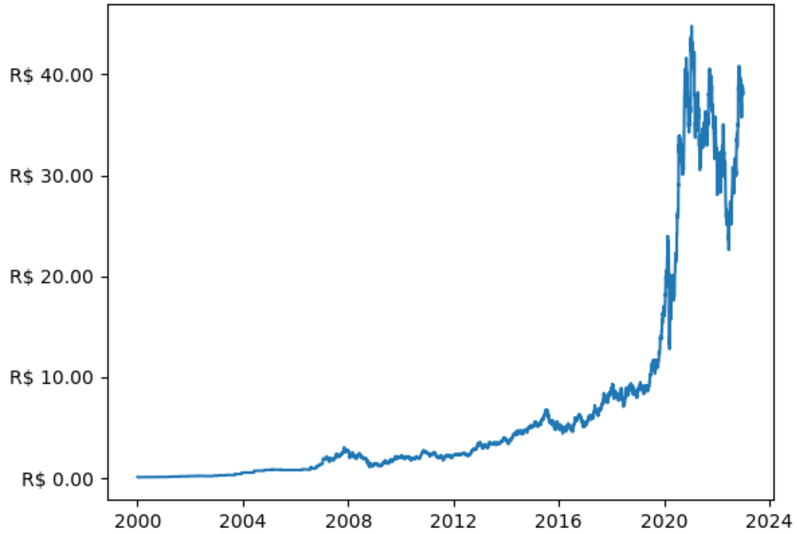
É um parâmetro **opcional** que deve receber uma função que retorne a formatação desejada.

Quando você deseja colocar o eixo em reais:

Exemplo:

```python
import matplotlib.pyplot as plt
import yfinance as yf
import matplotlib.ticker as ticker

def my_formatter(x, pos):
    return f'R$ {x:.2f}'


cotacoes = yf.download(["WEGE3.SA"])['Adj Close']
fig, ax = plt.subplots()
ax.plot(cotacoes.index, cotacoes.values)

ax.yaxis.set_major_formatter(formatter=ticker.FuncFormatter(my_formatter))

plt.show()
```

Respostas:

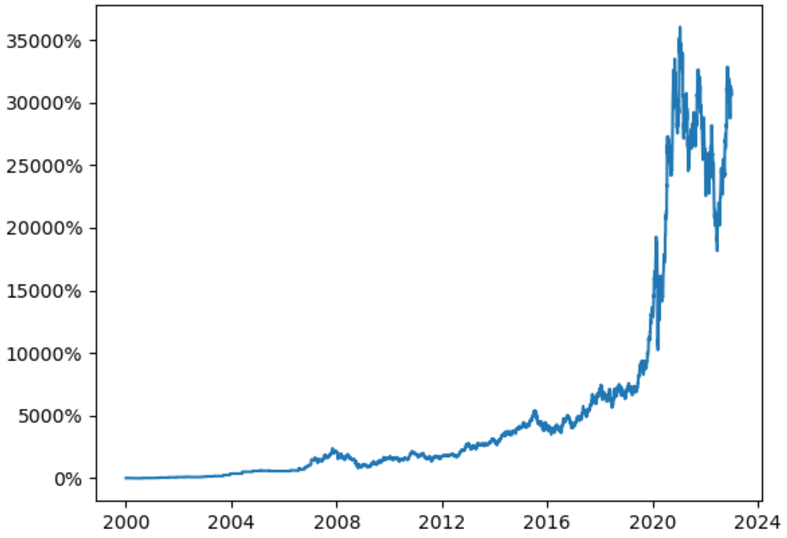


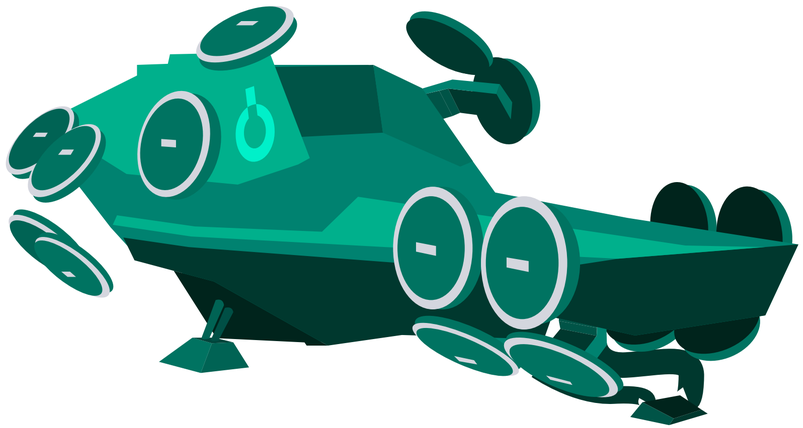
Quando você deseja calcular o retorno acumulado e colocá-lo em porcentagem:

Exemplo:

```
import matplotlib.pyplot as plt
import yfinance as yf
import matplotlib.ticker as ticker

cotacoes = yf.download(["WEGE3.SA"])['Adj Close']
fig, ax = plt.subplots()

retornos = cotacoes.pct_change().dropna()
retornos_acumulados = (1 + retornos).cumprod() - 1
ax.plot(retornos_acumulados.index, retornos_acumulados.values)

ax.yaxis.set_major_formatter(ticker.PercentFormatter(1))
plt.show()
```

Respostas:





8.3. matplotlib.axes.Axes.set_major_locator(locator = None)

Este método é usado para definir a formata**ção dos valores dos eixos.** Este é um método da biblioteca matplotlib.

Parâmetros:

locator:

Este parâmetro é utilizado em conjunto com um submódulo da biblioteca matplotlib. Ele é utilizado quando se quer definir intervalos de datas nos eixos.

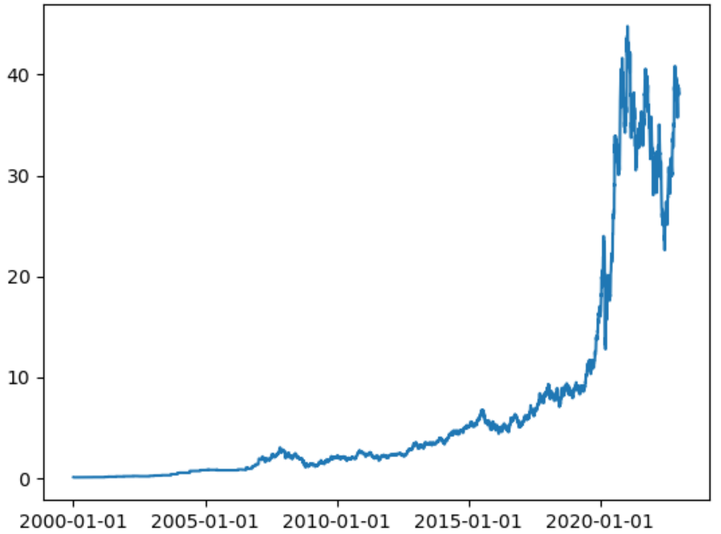
É um parâmetro **opcional** que deve receber um método do submódulo em conjunto com o intervalo de tempo definido no formato integer.

Quando você deseja definir intervalos do eixo x de 5 em 5 anos:

Exemplo:

```
import matplotlib.pyplot as plt
import yfinance as yf
import matplotlib.ticker as ticker

def my_formatter(x, pos):
    return f'R$ {x:.2f}'


cotacoes = yf.download(["WEGE3.SA"])['Adj Close']
fig, ax = plt.subplots()
ax.plot(cotacoes.index, cotacoes.values)

ax.xaxis.set_major_locator(mdate.YearLocator(5))
plt.show()
```

Respostas:

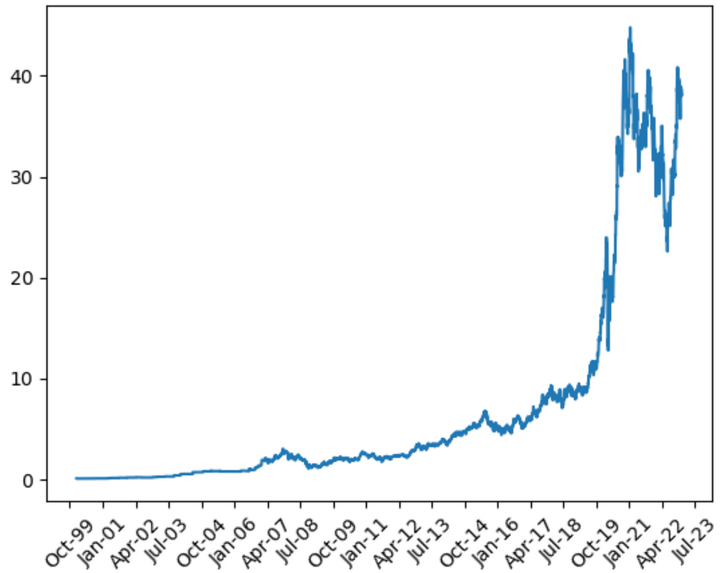


Quando você deseja definir intervalos do eixo x de 15 em 15 meses e define a formatação dele sendo mmm-yy, e dá uma rotação de 45° para as informações não coincidirem :

Exemplo:

```
import matplotlib.pyplot as plt
import yfinance as yf
import matplotlib.dates as mdate

cotacoes = yf.download(["WEGE3.SA"])['Adj Close']
fig, ax = plt.subplots()
ax.plot(cotacoes.index, cotacoes.values)

minha_formatacao = mdate.DateFormatter('%b-%y')
ax.xaxis.set_major_formatter(minha_formatacao)

plt.xticks(rotation=45)
ax.xaxis.set_major_locator(mdate.MonthLocator(interval = 15))

plt.show()
```

Respostas:



Mundo 9

Neste mundo aprenderemos a como estilizar gráficos.

9.1. matplotlib.axes.Axes.set_facecolor(color)

Este método é usado para definir a coloração do fundo do gráfico. **Este é um método da biblioteca matplotlib**.

Parâmetros:

color:

Este parâmetro é utilizado para definir a coloração do fundo do gráfico.

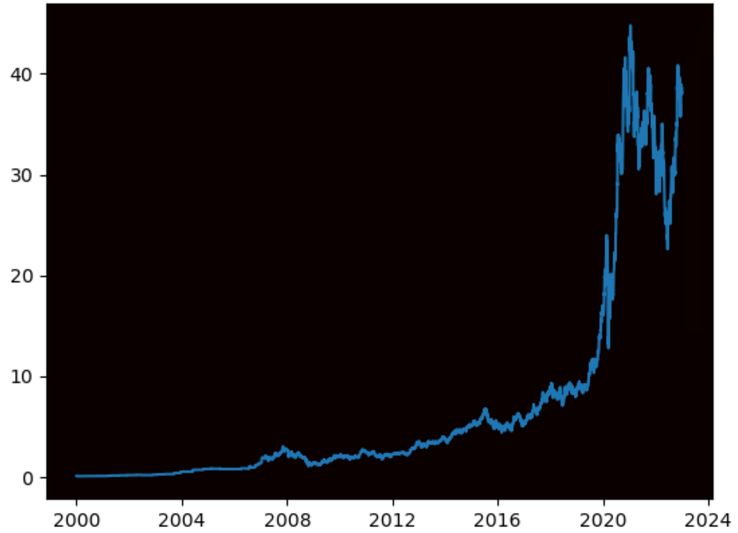
É um parâmetro **opcional** que deve receber uma cor, no formato string, de acordo com a tabela predefinida, pode receber também hexadecimal no formato string, ou também uma tuple com o formato RGB ou RGBA.

obs: O formato RGB/RGBA do python não é o mesmo que o normal, para isso você precisa dividir os números por 255, para que consiga acessar a cor desejada de acordo com o obtido na internet.

Exemplo:

```
import matplotlib.pyplot as plt
import yfinance as yf

cotacoes = yf.download(["WEGE3.SA"])['Adj Close']
fig, ax = plt.subplots()

ax.plot(cotacoes.index, cotacoes.values)
ax.set_facecolor((10/255, 0/255, 0/255))
plt.show()
```

Respostas:



9.2. matplotlib.style.use(style)

Este método é usado para definir o estilo de cores do gráfico, são estilos predefinidos. **Este é um método da biblioteca matplotlib**.

Parâmetros:

style:

Este parâmetro é utilizado para definir o estilo de cores do gráfico.

É um parâmetro **opcional** que deve receber um estilo na formatação string de acordo com a tabela predefinida. Também há a opção de baixar novos estilos que não vêm instalados com o Python, como o "cyberpunk" por meio do comando "pip install -q cyberpunk"

Exemplo:

```
import matplotlib.pyplot as plt
import yfinance as yf
import mplcyberpunk

plt.style.use("cyberpunk")

cotacoes = yf.download(["WEGE3.SA", "PETR4.SA"])['Adj Close']
fig, ax = plt.subplots()

ax.plot(cotacoes.index, cotacoes.values)

plt.show()
```

Respostas:

Mundo 10

Neste mundo aprenderemos a fazer gráficos de barras orientado a objetos.

10.1. Plotando um gráfico de barras

1° passo:

Antes de tudo, vamos importar todos os módulos necessários.

```
import matplotlib.pyplot as plt
from bcb import sgs
import matplotlib.ticker as mtick
import matplotlib.dates as mdate
from datetime import datetime, timedelta
```

2° passo:

Depois vamos importar os dados da inflação, diretamente da API do Banco Central do Brasil. Para isso, precisamos instalar a biblioteca por meio do comando "pip install python-bcb'". Seguidamente, exportar os dados através do método "get" (Não se preocupe quanto a essa parte, teremos um módulo que abordará a API do Banco Central).

Exemplo:

```
inflacao = sgs.get({'ipca': 433,'igp-m': 189}, start = datetime.now() - timedelta(days = 180))
```

Respostas:

```
                ipca  igp-m
Date
2022-07-01 -0.68   0.21
2022-08-01 -0.36  -0.70
2022-09-01 -0.29  -0.95
2022-10-01  0.59  -0.97
2022-11-01  0.41  -0.56
2022-12-01   NaN   0.45
```

3° passo:

Vamos criar um vetor numérico a partir das datas extraídas do Data-Frame das inflações nos últimos 6 meses. Utilizaremos esses números como forma de localizar a posição das informações e utilizá-las depois.

Exemplo:

```
datas_numericas = mdate.date2num(inflacao.index)
```

Respostas:

```
[738337. 738368. 738399. 738429. 738460. 738490.]
```

4° passo:

Vamos criar dois gráficos de barras com a data numérica como eixo x, o primeiro utilizando as informações do ipca e o segundo, as informações do igp-m.

parâmetro x:

Se plotarmos estes gráficos sem os devidos cuidados, plotaremos um sobreposto ao outro. Por isso, transformamos as datas em números, para utilizarmos como forma de posicioná-los no gráfico.

parâmetro width:

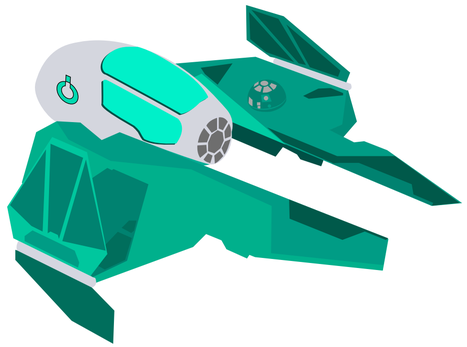
Após isso, definiremos uma largura para o gráfico, e utilizaremos para ajustar a posição do outro gráfico, para que um não sobreponha o outro.

parâmetro height:

Depois de já definido as larguras do eixo “x”, definiremos o eixo “y” definindo que height = valores de “ipca”” e “igp-m”.

parâmetro label:

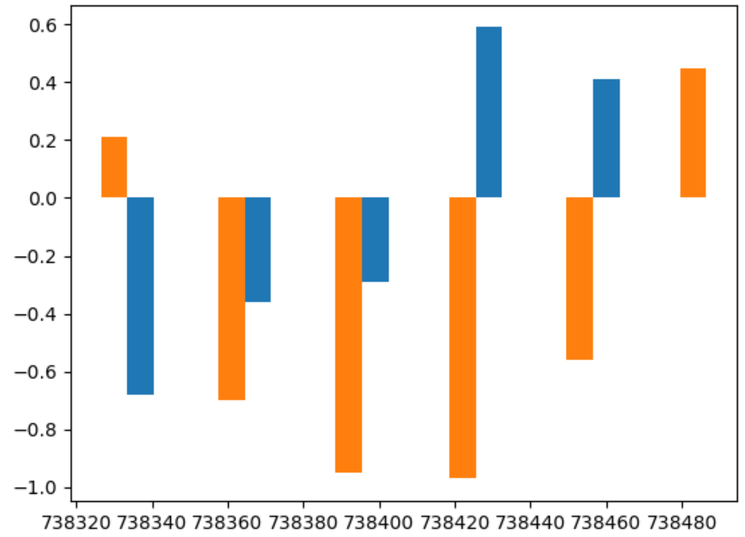
E por último, definimos quais serão os nomes deles no rótulo, usado para identificação, por meio da cor, de cada barra no gráfico.



Exemplo:

```
fig, ax = plt.subplots()
largura = 7
ax.bar(x=datas_numericas, height=inflacao['ipca'], label = "IPCA", width=largura)
ax.bar(x=datas_numericas - largura, height=inflacao['igp-m'], label = "IGP-M", width=largura)
```

Respostas:



5° passo:

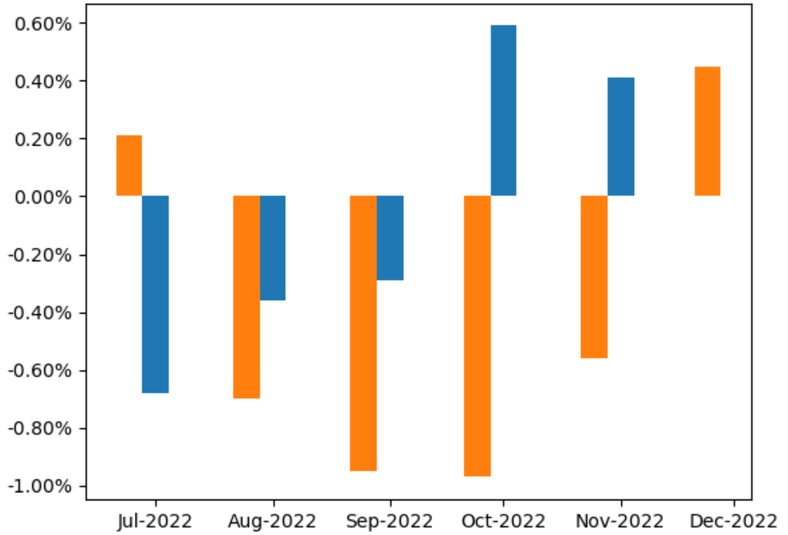
Vamos transformar o formato dos valores do eixo y para porcentagem.

```
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
```

Vamos transformar o formato dos valores do eixo para datas (mmm--yy)

```
myFmt = mdate.DateFormatter('%b-%Y')
ax.xaxis.set_major_formatter(myFmt)
ax.xaxis_date()
```

Respostas:



6° passo:

Por fim, vamos criar uma linha preta para que a gente utilize ela como início do eixo x.

```
plt.axhline(y = 0, color = "black")
```

E definiremos a legenda.

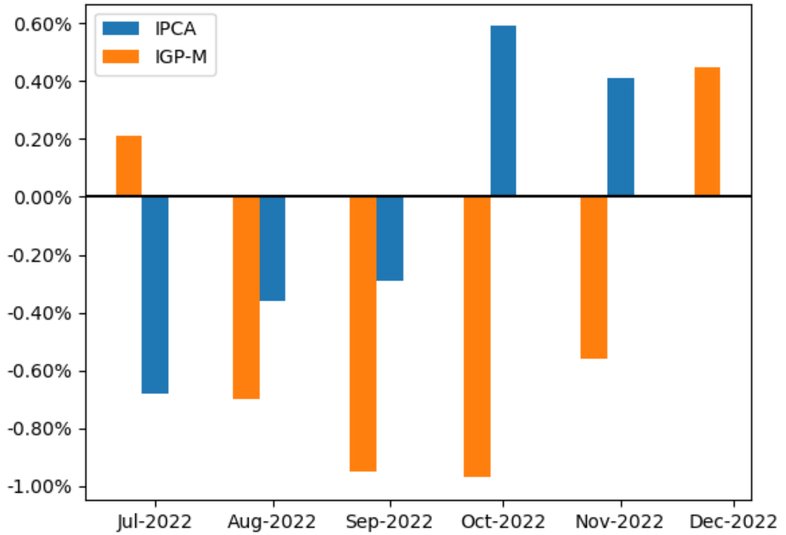
```
plt.legend()
```

Resultado final:

Exemplo:

```
import matplotlib.pyplot as plt
from bcb import sgs
import matplotlib.ticker as mtick
import matplotlib.dates as mdate
from datetime import datetime, timedelta

inflacao = sgs.get({'ipca': 433,
                    'igp-m': 189}, start = datetime.now() - timedelta(days = 180))
datas_numericas = mdate.date2num(inflacao.index)

fig, ax = plt.subplots()
largura = 7
ax.bar(x=datas_numericas, height=inflacao['ipca'], label = "IPCA", width=largura)
ax.bar(x=datas_numericas - largura, height=inflacao['igp-m'], label = "IGP-M", width=largura)

ax.yaxis.set_major_formatter(mtick.PercentFormatter())
myFmt = mdate.DateFormatter('%b-%Y')
ax.xaxis.set_major_formatter(myFmt)
ax.xaxis_date()
plt.axhline(y = 0, color = "black")
plt.legend()

plt.show()
```

Respostas:


IPCA
IGP-M
0.60%
0.40%
0.20%
0.00%
-0.20%
-0.40%
-0.60%
-0.80%
-1.00%
Jul-2022
Aug-2022
Sep-2022
Oct-2022
Nov-2022
Dec-2022

Mundo 11

Neste mundo aprenderemos a fazer gráficos de dispersão, boxplot e histograma orientado a objetos.

11.1. Plotando um histograma

1° passo:

Antes de tudo, vamos importar todos os módulos necessários.

```
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
from datetime import datetime
from datetime import timedelta
import yfinance as yf
```

2° passo:

Vamos importar as cotações ajustadas da WEGE3 e calcular o retorno diário.

Exemplo:

```
cotacoes = yf.download(["WEGE3.SA"], end=datetime.today(), start=datetime.today()- timedelta(days=480))['Adj Close']
retornos_diarios = cotacoes.pct_change().dropna()
```

Respostas:

```
Date
2021-09-13                                0.006208
2021-09-14                                0.012339
2021-09-15                                0.011681
2021-09-16                                0.001506
2021-09-17                               -0.017794
                                             ...
2022-12-26                               -0.008052
2022-12-27                               -0.004713
2022-12-28                                0.018153
2022-12-29                               -0.004910
2023-01-02                               -0.010647
Name: Adj Close, Length: 326, dtype: float64
```

3° passo:

Vamos criar um histograma com os retornos diários.

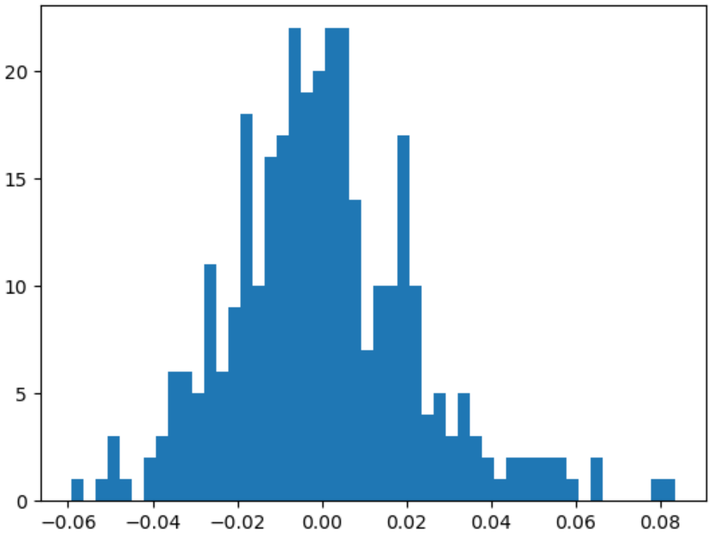
parâmetro bins:

Esse parâmetro vai definir a quantidade de barras que os dados serão distribuídos.

Exemplo:

```
fig, ax = plt.subplots()
ax.hist(retornos_diarios.values, bins = 50)
```

Respostas:



4° passo:

Vamos mudar a formatação do eixo x para porcentagem

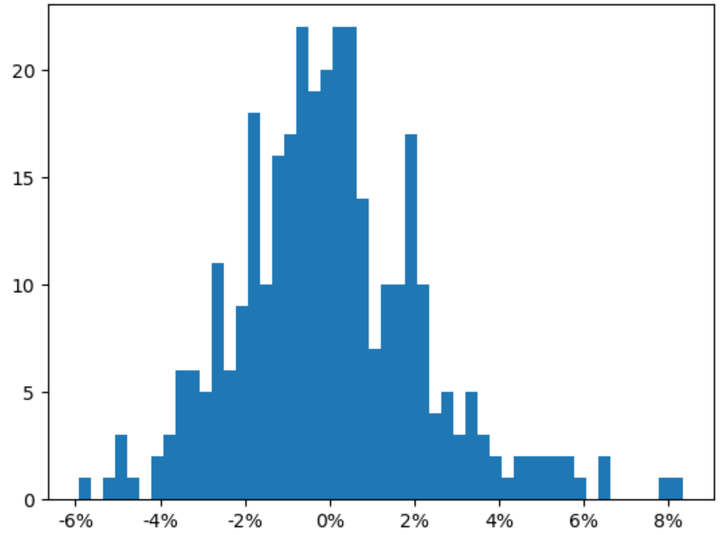
```
ax.xaxis.set_major_formatter(mtick.PercentFormatter(1, decimals = 0))
```

Resultado final:

Exemplo:

```
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
from datetime import datetime
from datetime import timedelta
import yfinance as yf

cotacoes = yf.download(["WEGE3.SA"], end=datetime.today(), start=datetime.today()- timedelta(days=480))['Adj Close']
retornos_diarios = cotacoes.pct_change().dropna()

fig, ax = plt.subplots()
ax.hist(retornos_diarios.values, bins = 50)

ax.xaxis.set_major_formatter(mtick.PercentFormatter(1, decimals = 0))

plt.show()
```

Respostas:



11.2. Plotando um gráfico de dispersão

1° passo:

Antes de tudo, vamos importar todos os módulos necessários.

```
import matplotlib.ticker as mtick
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.pylab as lab
import yfinance as yf
import mplcyberpunk
from datetime import datetime, timedelta
```

2° passo:

Vamos importar as cotações ajustadas da WEGE3 e do Ibovespa e calcular o retorno anual, criando uma coluna auxiliar ao lado que define o ano referente ao retorno.

Exemplo:

```
cotacoes = yf.download(["WEGE3.SA", "^BVSP"], end=datetime.today(), start=datetime.today()- timedelta(days=1390))['Adj Close']

retornos_anuais= cotacoes.resample("Y").last().pct_change().
dropna()
retornos_anuais['Ano'] = retornos_anuais.index.year
```

Respostas:

Date	WEGE3.SA	^BVSP	Ano
2020-12-31	1.202823	0.028819	2020
2021-12-31	-0.119122	-0.121402	2021
2022-12-31	0.188505	0.049694	2022
2023-12-31	-0.010647	-0.033218	2023

3° passo:

Vamos selecionar o estilo “cyberpunk” (baixado no mundo 9) para o nosso gráfico.

```
plt.style.use("cyberpunk")
```

4° passo:

Vamos criar uma barra discreta. Que é nada mais do que uma legenda com a identificação dos pontos e seus significados. Como pode ver, é uma função que cria uma barra discreta e seleciona a cor para cada ano do DataFrame. Onde passaremos a coluna dos anos como nosso “vetor_categoria”.

```
def barra_discreta(vetor_categoria):

    cmap = lab.cm.cool  # define a cor
    cmaplist = [cmap(i) for i in range(cmap.N)]
    cmap = mpl.colors.LinearSegmentedColormap.from_list(
    'cmap escolhido', cmaplist, cmap.N)
    bounds = np.linspace(np.min(vetor_categoria),     np.max
    (vetor_categoria) + 0.5, len(vetor_categoria) + 1)
    norm = mpl.colors.BoundaryNorm(bounds, cmap.N)

    return bounds, norm
```

5° passo:

Criando um gráfico de dispersão com as informações dos retornos anuais do Ibovespa e da WEGE3.

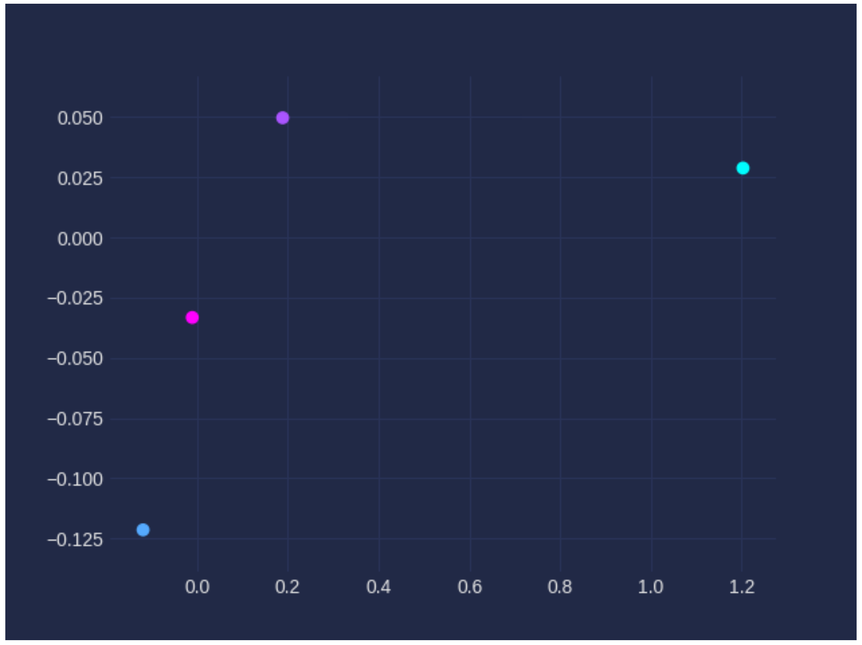
parâmetro cmap:

Esse parâmetro vai definir a coloração dos pontos de dispersão

Exemplo:

```
fig, ax = plt.subplots()
ax.scatter(retornos_anuais['WEGE3.SA'], retornos_anuais["^BVSP"], c = retornos_anuais['Ano'], cmap="cool")
```

Respostas:



6° passo:

Criando a barra discreta, por enquanto transparente, de acordo com a função criada anteriormente, passando os anos como a informação a ser usada.

```
ax2 = fig.add_axes([0.95, 0.1, 0.03, 0.8])
```

7° passo:

Fixando a barra discreta na posição desejada.

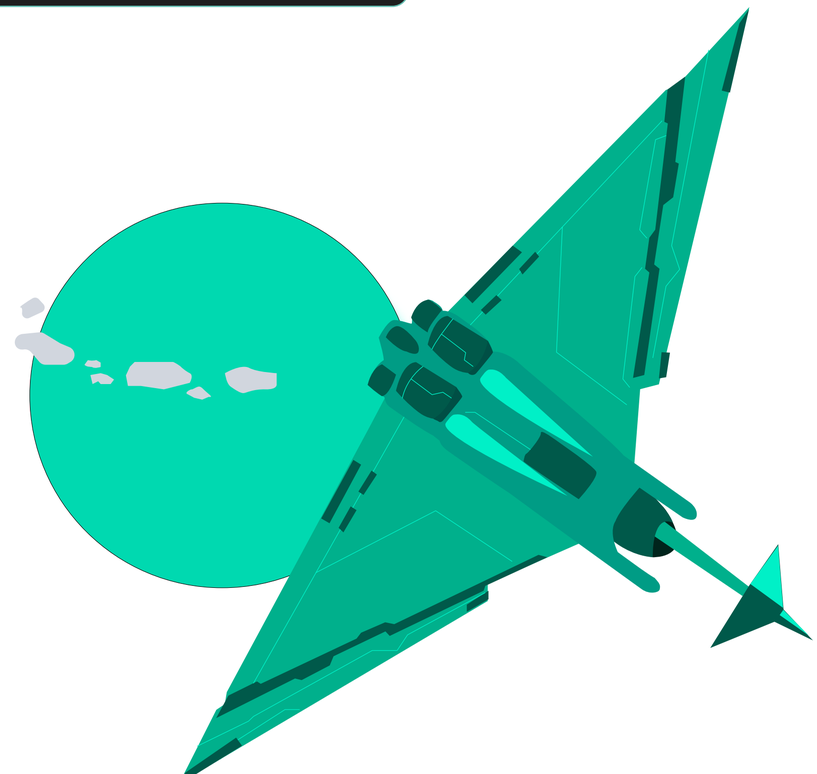
```
ax2 = fig.add_axes([0.95, 0.1, 0.03, 0.8])
```

8° passo:

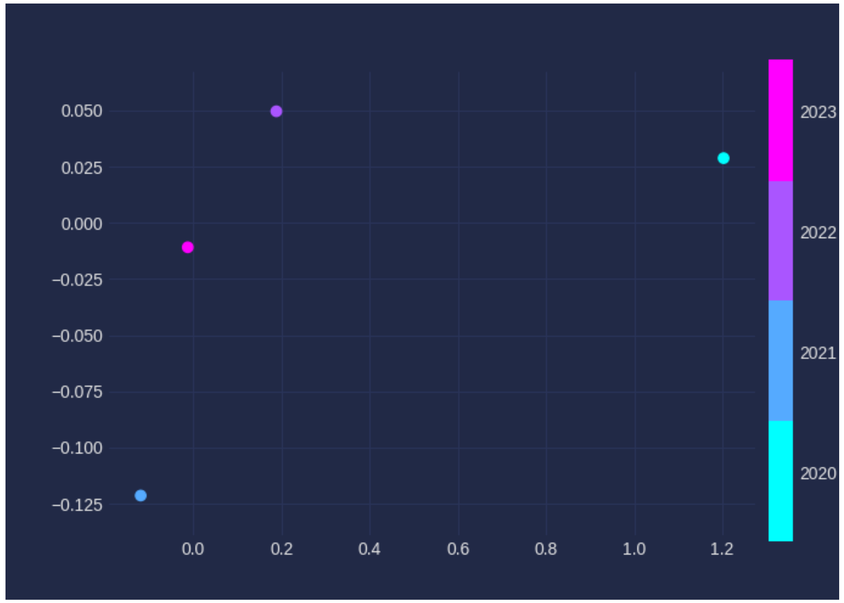
Definindo cor, proporção e informações para a barra discreta.

Exemplo:

```
cb = mpl.colorbar.ColorbarBase(ax2, cmap="cool", norm=norm,
spacing='proportional', ticks=bounds + 0.5,boundaries=bounds,
format='%1i')
```



Respostas:



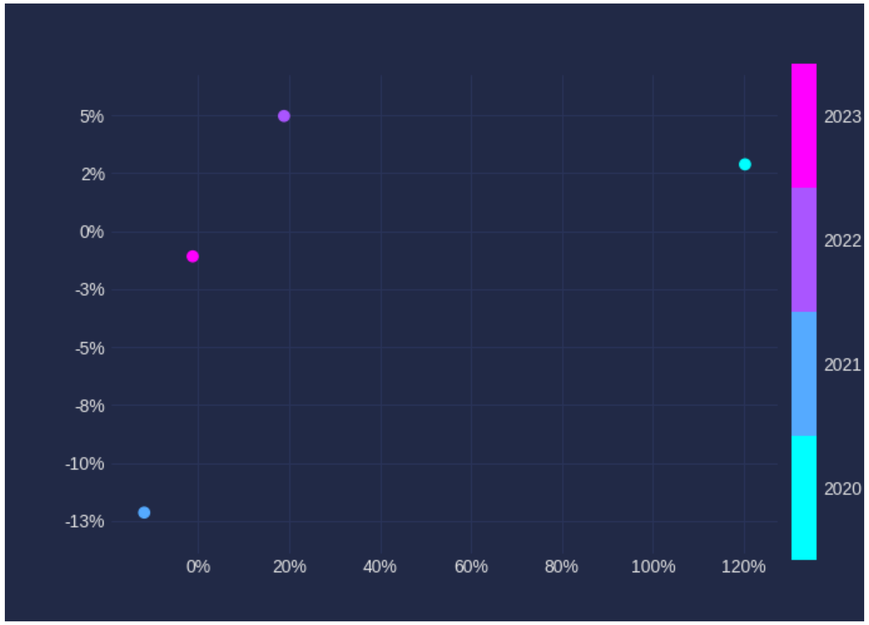
9° passo:

Mudando a formatação do eixo “x” e do eixo “y” para porcentagem.

Exemplo:

```
ax.xaxis.set_major_formatter(mtick.PercentFormatter(1, decimals = 0))
ax.yaxis.set_major_formatter(mtick.PercentFormatter(1, decimals = 0))
```

Respostas:



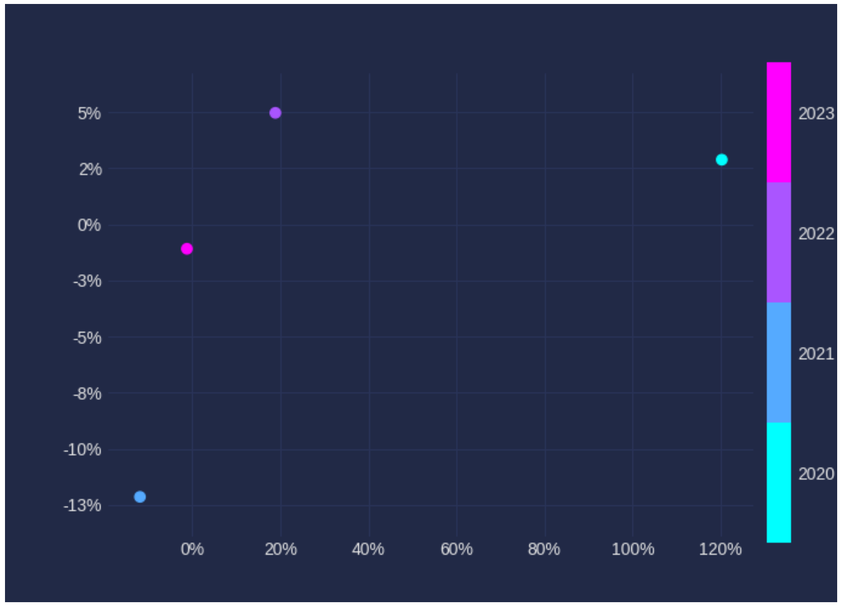
10° passo:

Colocando legenda no eixo “x” e no eixo “y” e definindo título do gráfico.

Exemplo:

```
ax.set_ylabel("Ibovespa")
ax.set_xlabel("WEGE3")
ax.set_title("Retornos anuais WEGE3 X Ibovespa")
```

Respostas:

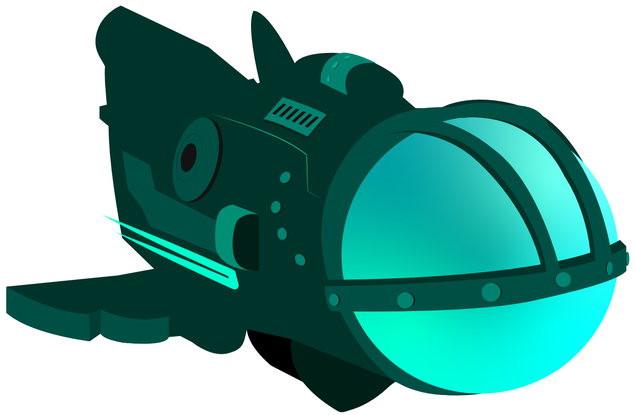


11° passo:

Colocando linhas que definem o começo do eixo “x” e do eixo “y”

Exemplo:

```
ax.axhline(y = 0)
ax.axvline(x = 0)
```



Resultado final:

Exemplo:

```
import matplotlib.ticker as mtick
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.pylab as lab
import yfinance as yf
import mplcyberpunk
from datetime import datetime, timedelta

def barra_discreta(vetor_categoria):

    cmap = lab.cm.cool  # define a cor
    cmaplist = [cmap(i) for i in range(cmap.N)]
    cmap = mpl.colors.LinearSegmentedColormap.from_list(
        'cmap escolhido', cmaplist, cmap.N)
    bounds = np.linspace(np.min(vetor_categoria), np.max(vetor_categoria) + 0.5, len(vetor_categoria) + 1)
    norm = mpl.colors.BoundaryNorm(bounds, cmap.N)

    return bounds, norm

cotacoes = yf.download(["WEGE3.SA", "^BVSP"], end=datetime.today(), start=datetime.today()- timedelta(days=1390))['Adj Close']
retornos_anuais = cotacoes.resample("Y").last().pct_change().dropna()
retornos_anuais['Ano'] = retornos_anuais.index.year
print(retornos_anuais)
plt.style.use("cyberpunk")

fig, ax = plt.subplots()
ax.scatter(retornos_anuais['WEGE3.SA'], retornos_anuais["^BVSP"], c = retornos_anuais['Ano'], cmap="cool")

bounds, norm = barra_discreta(retornos_anuais['Ano'])
ax2 = fig.add_axes([0.95, 0.1, 0.03, 0.8])

cb = mpl.colorbar.ColorbarBase(ax2, cmap="cool", norm=norm,
    spacing='proportional', ticks=bounds + 0.5, boundaries=bounds, format='%1i')

ax.xaxis.set_major_formatter(mtick.PercentFormatter(1, decimals = 0))
ax.yaxis.set_major_formatter(mtick.PercentFormatter(1, decimals = 0))
ax.set_ylabel("Ibovespa")
ax.set_xlabel("WEGE3")
ax.set_title("Retornos anuais WEGE3 X Ibovespa")
ax.axhline(y = 0)
ax.axvline(x = 0)

plt.show()
```

Respostas:

11.3. Plotando um gráfico de boxplot

1° passo:

Antes de tudo, vamos importar todos os módulos necessários.

```
import matplotlib.ticker as mtick
import matplotlib.pyplot as plt
import yfinance as yf
from datetime import datetime, timedelta
import yfinance as yf
```

2° passo:

Vamos importar as cotações ajustadas da WEGE3 e do Ibovespa calcular o retorno mensal.

Exemplo:

```
cotacoes = yf.download(["WEGE3.SA", "^BVSP"],end=datetime.today
(), start=datetime.today()- timedelta(days=480))['Adj Close']
retornos_mensais = cotacoes.resample("M").last().pct_
change().dropna()
```

Respostas:

```
DATE              WEGE3.SA          ^BVSP
Date
2021-10-31       -0.066364       -0.067382
2021-11-30       -0.128919       -0.015324
2021-12-31        0.024211        0.028524
2022-01-31       -0.024863        0.072180
                                      ...
2022-10-31        0.253267        0.054527
2022-11-30       -0.031778       -0.030602
2022-12-31       -0.011129       -0.021825
2023-01-31       -0.010647       -0.033218
```

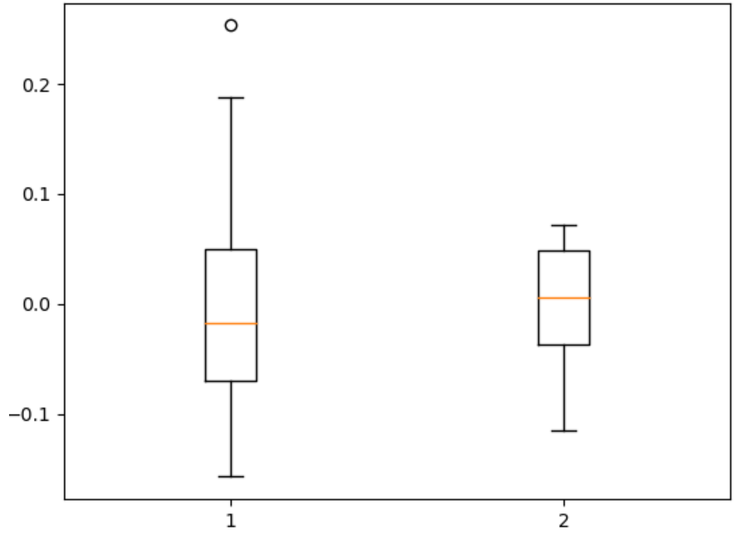
3° passo:

Vamos criar um gráfico de boxplot com os retornos mensais

Exemplo:

```
fig, ax = plt.subplots()
ax.boxplot(x = [retornos_mensais['WEGE3.SA'],
retornos_mensais['^BVSP']])
```

Respostas:



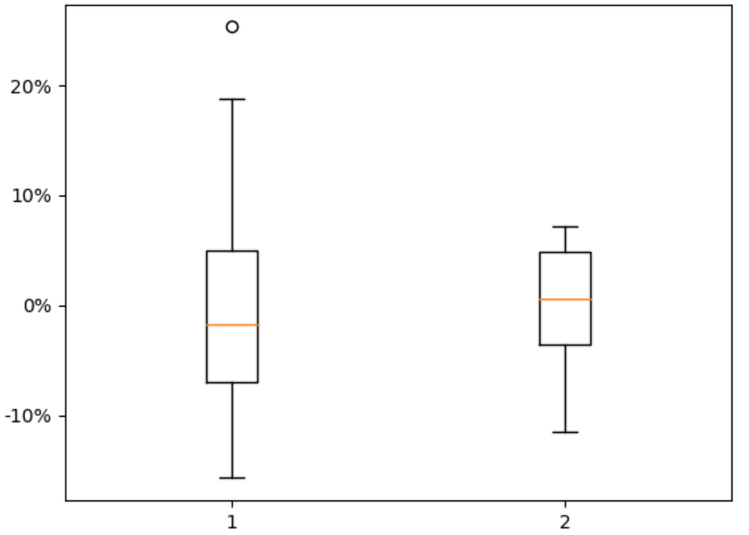
4° passo:

Vamos transformar a formatação do eixo y para porcentagem

Exemplo:

```
ax.yaxis.set_major_formatter(mtick.PercentFormatter(1, decimals = 0))
```

Respostas:



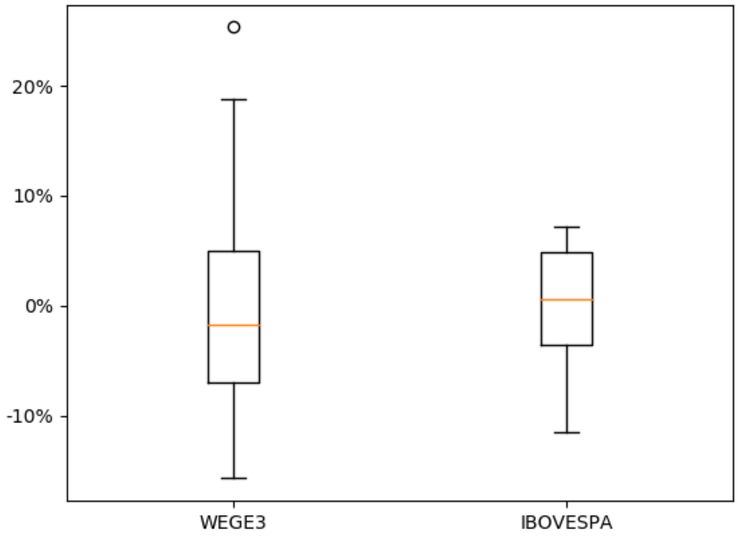
5° passo:

Vamos renomear os valores do eixo x

Exemplo:

```
ax.set_xticks([1,2],["WEGE3","IBOVESPA"])
```

Respostas:

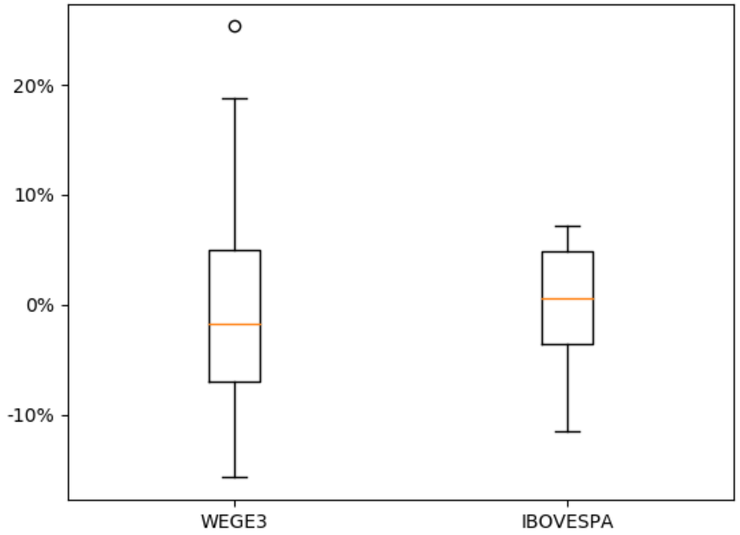


Resultado final:

Exemplo:

```
import matplotlib.ticker as mtick
import matplotlib.pyplot as plt
import yfinance as yf
from datetime import datetime, timedelta
import yfinance as yf

cotacoes = yf.download(["WEGE3.SA", "^BVSP"], end=datetime.today(), start=datetime.today()- timedelta(days=480))['Adj Close']
retornos_mensais = cotacoes.resample("M").last().pct_change().dropna()

fig, ax = plt.subplots()
ax.boxplot(x = [retornos_mensais['WEGE3.SA'],retornos_mensais['^BVSP']])

ax.yaxis.set_major_formatter(mtick.PercentFormatter(1, decimals = 0))

ax.set_xticks([1,2],["WEGE3","IBOVESPA"])
plt.show()
```

Respostas:



Mundo 12

Neste mundo aprenderemos a utilizar e integrar o pacote de gráficos seaborn junto ao matplotlib.

12.1. seaborn.boxplot(data = None, x = None, y = None, hue = None, order = None, orient = None, color = None, palette = None, saturation = 0.75, width = 0.8, fliersize = 5, linewidth = None, others)

Este método é usado para a criação de um gráfico de boxplot. **Este é um método da biblioteca Seaborn**.

Parâmetros:

data:

Este parâmetro vai definir os dados do gráfico boxplot.

É um parâmetro **opcional** que pode receber um DataFrame, um vetor ou uma lista com os vetores dentro.

x:

Este parâmetro vai definir os valores do eixo x, se o parâmetro **data** = None. Caso contrário, ele definirá quais serão as informações usadas dentro do DataFrame.

É um parâmetro **opcional** que pode receber o nome do eixo no formato string ou um vetor com os dados no formato integer.

y:

Este parâmetro vai definir os valores do eixo y, se o parâmetro **data** = None. Caso contrário, ele definirá quais serão as informações usadas dentro do DataFrame.

É um parâmetro **opcional** que pode receber o nome do eixo no formato string ou um vetor com os dados no formato integer.

hue:

Este parâmetro vai definir o mapeamento das variáveis, se o parâmetro **data** = None. Caso contrário, ele definirá quais serão as informações usadas dentro do DataFrame.

É um parâmetro **opcional** que pode receber o nome do eixo no formato string ou um vetor com os dados no formato integer.

orient:

Este parâmetro vai definir a orientação do seu gráfico. Se será verticalmente ou horizontalmente.

É um parâmetro **opcional** que deve receber valores predefinidos, "h" para horizontal e "v" para vertical, no formato string.

palette:

Este parâmetro vai definir qual será a paleta de cores utilizada para pintar o seu gráfico.

É um parâmetro **opcional** que deve receber valores predefinidos, de acordo com a tabela predefinida, no formato string.

saturation:

Este parâmetro vai definir qual será a saturação das cores dentro do gráfico.

É um parâmetro **opcional** que deve receber um valor entre 0 e 1, no formato float.

width:

Este parâmetro vai definir qual será a largura do gráfico.

É um parâmetro **opcional** que deve receber um valor no formato float.

filersize:

Este parâmetro vai definir qual será o tamanho dos marcadores.

É um parâmetro **opcional** que deve receber um valor no formato float.

linewidth:

Este parâmetro vai definir qual será a largura da linha do gráfico.

É um parâmetro **opcional** que deve receber um valor no formato float.

others:

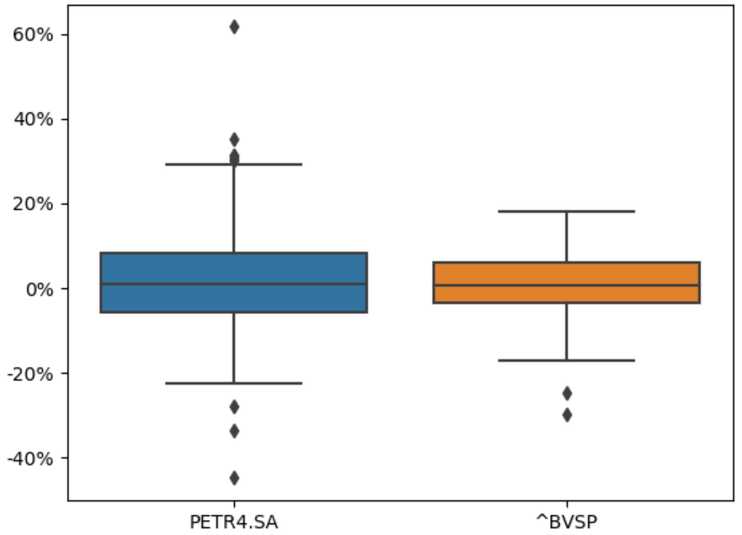
Parâmetros do matplotlib.axes.Axes.boxplot(). Podem ser utilizados neste método.

Documentação: https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.boxplot.html#matplotlib.axes.Axes.boxplot

Exemplo:

```python
import matplotlib.ticker as mtick
import matplotlib.pyplot as plt
import yfinance as yf
import seaborn as sns

cotacoes = yf.download(["PETR4.SA", "^BVSP"])['Adj Close']
retornos_mensais = cotacoes.resample("M").last().pct_change().dropna()

fig, ax = plt.subplots()

sns.boxplot(data = retornos_mensais, orient = "v")
ax.yaxis.set_major_formatter(mtick.PercentFormatter(1, decimals = 0))

plt.show()
```

Respostas:



12.2. seaborn.histplot(data = None, x = None, y = None, hue = None, stat = 'count', bins = 'auto', binwidth = None, binrange = None, cumulative = False, element = 'bars', fill = True, kde = False, palette = None, color = None, others)

Este método é usado para a criação de um gráfico de boxplot. **Este é um método da biblioteca Seaborn**.

Parâmetros:

data:

Este parâmetro vai definir os dados do gráfico boxplot.

É um parâmetro **opcional** que pode receber um DataFrame, um vetor ou uma lista com os vetores dentro.

x:

Este parâmetro vai definir os valores do eixo x, se o parâmetro **data** = None. Caso contrário, ele definirá quais serão as informações usadas dentro do DataFrame.

É um parâmetro **opcional** que pode receber o nome do eixo no formato string ou um vetor com os dados no formato integer.

y:

Este parâmetro vai definir os valores do eixo y, se o parâmetro **data** = None. Caso contrário, ele definirá quais serão as informações usadas dentro do DataFrame.

É um parâmetro **opcional** que pode receber o nome do eixo no formato string ou um vetor com os dados no formato integer.

hue:

Este parâmetro vai definir o mapeamento das variáveis se o parâmetro **data** = None. Caso contrário, ele definirá quais serão as informações usadas dentro do DataFrame.

É um parâmetro **opcional** que pode receber o nome do eixo no formato string ou um vetor com os dados no formato integer.

stat:

Este parâmetro vai definir como serão agregados os dados dentro do gráfico.

É um parâmetro **opcional** que deve receber valores predefinidos, no formato string.

count => Número de resultados em cada bin

frequency => Número de observações dividido pela largura do bin

probability => Normalizar de modo que a soma das alturas das barras seja 1

percent => Normalizar de modo que a soma das alturas das barras seja 100

densidade => Normalizar de modo que a área total do histograma seja 1

bins:

Este parâmetro vai definir o número de colunas a serem usadas no gráfico plotado.

É um parâmetro **opcional** que recebe o número de colunas no formato integer.

binwidth:

Este parâmetro vai definir a largura de cada bin.

É um parâmetro **opcional** que recebe a largura dos bins no formato float.

binrange:

Este parâmetro vai definir o menor e o maior valor para as bordas.

É um parâmetro **opcional** que recebe uma list com valores dos extremos no formato float.

cumulative:

Este parâmetro vai definir se as barras serão disponibilizadas de forma cumulativa.

É um parâmetro **opcional** que receberá um booleano com True ou False.

element:

Este parâmetro vai definir qual formato com que os dados serão disponibilizados.

É um parâmetro **opcional** que receberá formatos predefinidos no formato string.

bars => Separa cada barra

step => Sem a separação das barras, parece uma "escada"

poly => Formato de pirâmide, ligando cada ponto

fill:

Este parâmetro vai definir se o gráfico será preenchido ou não

É um parâmetro **opcional** que receberá um booleano com True ou False.

kde:

Este parâmetro vai calcular o KDE e demonstrar através da suavização de uma ou mais linhas.

É um parâmetro **opcional** que receberá um booleano com True ou False.

palette:

Este parâmetro vai definir qual será a paleta de cores utilizada para pintar o seu gráfico.

É um parâmetro **opcional** que deve receber valores predefinidos, de acordo com a tabela predefinida, no formato string.

color:

Este parâmetro é utilizado para definir a coloração do fundo do gráfico.

É um parâmetro **opcional** que deve receber uma cor, no formato string, de acordo com a tabela predefinida, pode receber também hexadecimal no formato string, ou também uma tuple com o formato RGB ou RGBA.

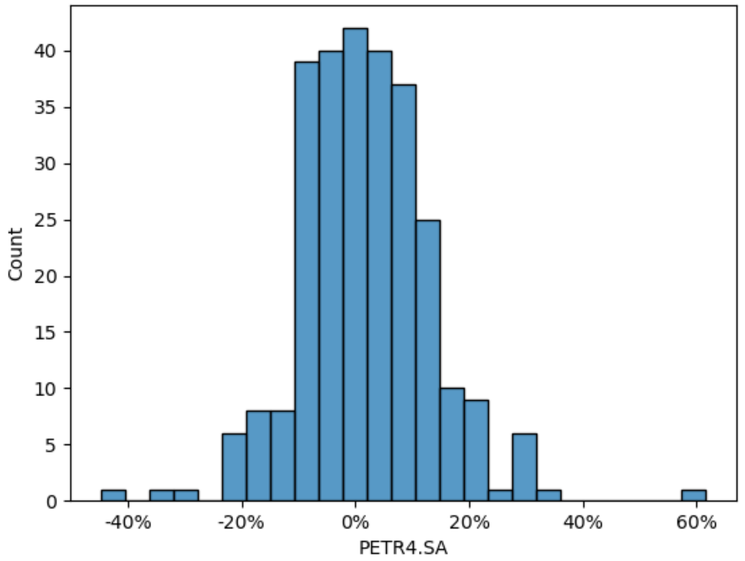
others:

Parâmetros do matplotlib.axes.Axes.bar(), matplotlib.axes.Axes.fill_between(), matplotlib.axes.Axes.plot(), matplotlib.axes.Axes.pcolormesh(). Podem ser utilizados neste método.

Exemplo:

```
import matplotlib.ticker as mtick
import matplotlib.pyplot as plt
import yfinance as yf
import seaborn as sns

cotacoes = yf.download(["PETR4.SA", "^BVSP"])['Adj Close']
retornos_mensais = cotacoes.resample("M").last().pct_change().dropna()

fig, ax = plt.subplots()

sns.histplot(retornos_mensais['PETR4.SA'])
ax.xaxis.set_major_formatter(mtick.PercentFormatter(1, decimals = 0))

plt.show()
```

Respostas:



12.3. seaborn.regplot(data = None, x = None, y = None, scatter = True, fit_reg = True, truncate = True, marker = 'o', scatter_kws = None, line_kws = None)

Este método é usado para a criação de um gráfico de boxplot. **Este é um método da biblioteca Seaborn**.

Parâmetros:

data:

Este parâmetro vai definir os dados do gráfico boxplot.

É um parâmetro **opcional** que pode receber um DataFrame, um vetor ou uma lista com os vetores dentro.

x:

Este parâmetro vai definir os valores do eixo x, se o parâmetro **data** = None. Caso contrário, ele definirá quais serão as informações usadas dentro do DataFrame.

É um parâmetro **opcional** que pode receber o nome do eixo no formato string ou um vetor com os dados no formato integer.

y:

Este parâmetro vai definir os valores do eixo y, se o parâmetro **data** = None. Caso contrário, ele definirá quais serão as informações usadas dentro do DataFrame.

É um parâmetro **opcional** que pode receber o nome do eixo no formato string ou um vetor com os dados no formato integer.

hue:

Este parâmetro vai definir o mapeamento das variáveis se o parâmetro **data** = None. Caso contrário, ele definirá quais serão as informações usadas dentro do DataFrame.

É um parâmetro **opcional** que pode receber o nome do eixo no formato string ou um vetor com os dados no formato integer.

scatter:

Este parâmetro vai definir se os pontos aparecerão na hora de plotar o gráfico.Por padrão, o seaborn define scatter = True.

É um parâmetro **opcional** que receberá um booleano com True ou False.

fit_reg:

Este parâmetro vai definir se a reta da regressão linear aparecerá na hora de plotar o gráfico. Por padrão, o seaborn define fit_reg = True.

É um parâmetro **opcional** que receberá um booleano com True ou False.

truncate:

Este parâmetro vai definir se a linha vai parar antes ou depois do fim do eixo x

É um parâmetro **opcional** que receberá um booleano com True ou False.

marker:

Este parâmetro vai definir o formato dos marcadores.

É um parâmetro **opcional** que deve receber valores no formato string de acordo com a tabela predefinida.

line_kws:

Este parâmetro vai definir a cor da linha que representa a regressão linear.

É um parâmetro **opcional** que vai receber um dict com o valor da cor a ser escolhida de acordo com a tabela predefinida.

scatter_kws:

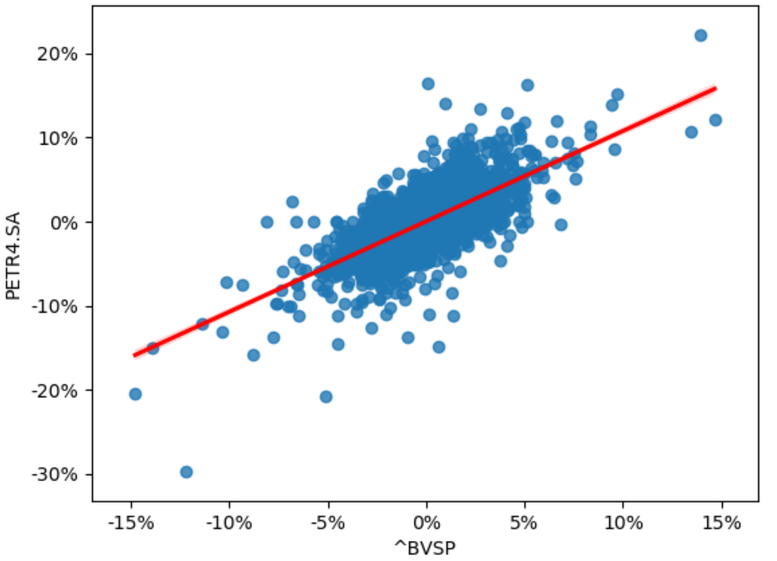
Este parâmetro vai definir a cor dos marcadores.

É um parâmetro **opcional** que vai receber um dict com o valor da cor a ser escolhida de acordo com a tabela predefinida.

Exemplo:

```
import matplotlib.ticker as mtick
import matplotlib.pyplot as plt
import yfinance as yf
import seaborn as sns

cotacoes = yf.download(["PETR4.SA", "^BVSP"])['Adj Close']
retornos = cotacoes.pct_change().dropna()

fig, ax = plt.subplots()
sns.regplot(x = retornos['^BVSP'], y = retornos['PETR4.SA'], line_kws={"color" : "red" })

ax.xaxis.set_major_formatter(mtick.PercentFormatter(1, decimals = 0))
ax.yaxis.set_major_formatter(mtick.PercentFormatter(1, decimals = 0))

plt.show()
```

Respostas:



12.4. seaborn.lineplot(data = None, x = None, y = None, hue = None, palette = None, dashes = True, markers = None, others)

Este método é usado para a criação de um gráfico de boxplot. **Este é um método da biblioteca Seaborn**.

Parâmetros:

data:

Este parâmetro vai definir os dados do gráfico boxplot.

É um parâmetro **opcional** que pode receber um DataFrame, um vetor ou uma lista com os vetores dentro.

x:

Este parâmetro vai definir os valores do eixo x, se o parâmetro **data** = None. Caso contrário, ele definirá quais serão as informações usadas dentro do DataFrame.

É um parâmetro **opcional** que pode receber o nome do eixo no formato string ou um vetor com os dados no formato integer.

y:

Este parâmetro vai definir os valores do eixo y, se o parâmetro **data** = None. Caso contrário, ele definirá quais serão as informações usadas dentro do DataFrame.

É um parâmetro **opcional** que pode receber o nome do eixo no formato string ou um vetor com os dados no formato integer.

hue:

Este parâmetro vai definir o mapeamento das variáveis se o parâmetro **data** = None. Caso contrário, ele definirá quais serão as informações usadas dentro do DataFrame.

É um parâmetro **opcional** que pode receber o nome do eixo no formato string ou um vetor com os dados no formato integer.

palette:

Este parâmetro vai definir qual será a paleta de cores utilizada para pintar o seu gráfico.

É um parâmetro **opcional** que deve receber valores predefinidos, de acordo com a tabela predefinida, no formato string.

marker:

Este parâmetro vai definir o marcador da série de dados de acordo com os tipos de marcadores.

É um parâmetro **opcional** que deve receber a formatação em string com formatos pré-definidos que obedecem os tipos de marcadores. Pode receber um booleano com True ou False, para definir se quer que marcadores apareçam.

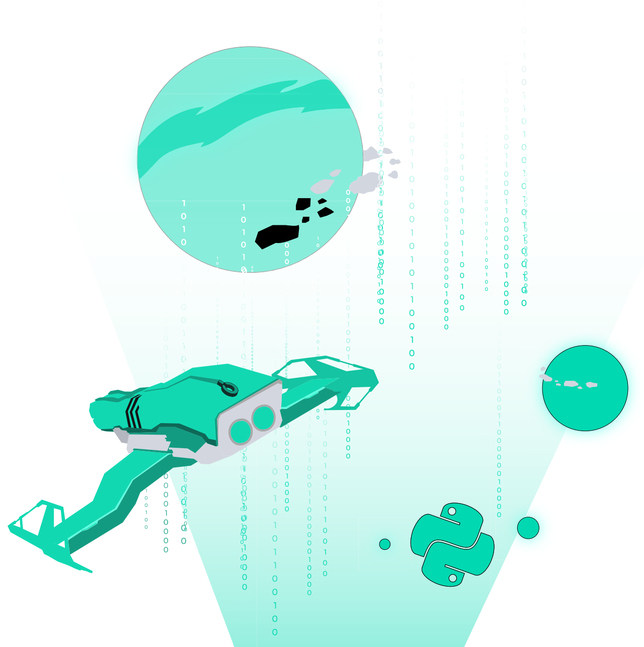
dashes:

Este parâmetro vai definir se uma das linhas terá a formatação de tracejada.

É um parâmetro **opcional** que receberá um booleano com True ou False.

others:

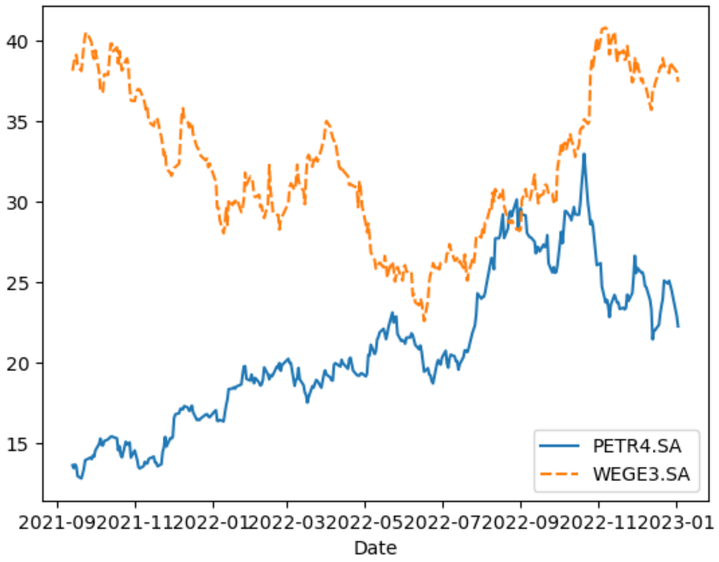
Parâmetros do matplotlib.axes.Axes.plot(). Podem ser utilizados neste método.



Exemplo:

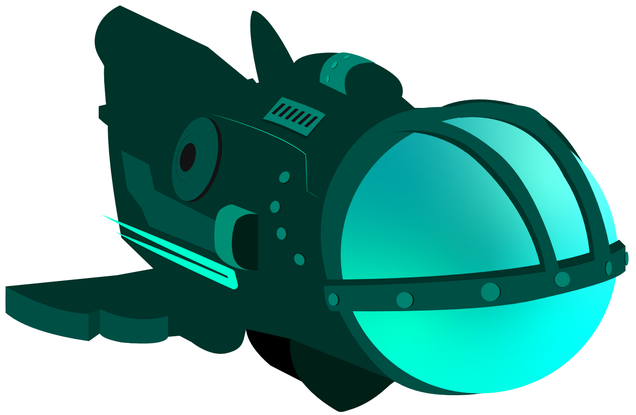
```
import matplotlib.pyplot as plt
import yfinance as yf
import seaborn as sns
from datetime import datetime, timedelta

cotacoes = yf.download(["PETR4.SA", "WEGE3.SA"] , end=datetime.today
(), start = datetime.today() - timedelta(days=480))['Adj Close']

fig, ax = plt.subplots()

sns.lineplot(data = cotacoes)

plt.show()
```

Respostas:



others:

Parâmetros do matplotlib.axes.Axes.plot(). Podem ser utilizados neste método.



Mundo 13

Neste mundo aprenderemos a fazer anotações em gráficos.

13.1. Axes.annotate(text, xy, xytext = None, xycoords = 'data', textcoords = None, arrowprops = None, annotation_clip = None, others)

Este método é usado para fazer anotações em gráficos. **Este é um método da biblioteca matplotlib**.

Parâmetros:

text:

Este parâmetro vai definir o texto da anotação.

É um parâmetro **opcional** que deve receber o valor da anotação no formato string.

xy:

Este parâmetro vai definir as coordenadas de anotação. Essa é a coordenada usada para inserir setas.

É um parâmetro **opcional** que deve receber uma tuple com valores (“coordenadas x” , “coordenadas y”) no formato float.

xytext:

Este parâmetro vai definir as coordenadas de anotação do texto. Essa é a coordenada usada para inserir o texto.

É um parâmetro **opcional** que deve receber uma tuple com valores (“coordenadas x” , “coordenadas y”) no formato float.

arrowprops:

Este parâmetro vai definir as propriedades da seta por meio de um dict.

{
width = largura da seta no formato float ,

headwidth = largura da cabeça da seta no formato float ,

headlenght = comprimento da cabeça da seta no formato float ,

facecolor = cor da seta no formato string de acordo com a tabela predefinida
}

É um parâmetro **opcional** que deve receber uma tuple com valores (“coordenadas x” , “coordenadas y”) no formato float.

horizontalalignment:

Este parâmetro vai definir como será feito o alinhamento horizontal das palavras

É um parâmetro **opcional** que deve receber valores predefinidos no formato string.

left => Alinhamento para esquerda

center => Alinhamento no centro

right => Alinhamento para direita

verticalalalignment:

Este parâmetro vai definir como será feito o alinhamento vertical das palavras

É um parâmetro **opcional** que deve receber valores predefinidos no formato string.

left => Alinhamento para esquerda

center => Alinhamento no centro

right => Alinhamento para direita

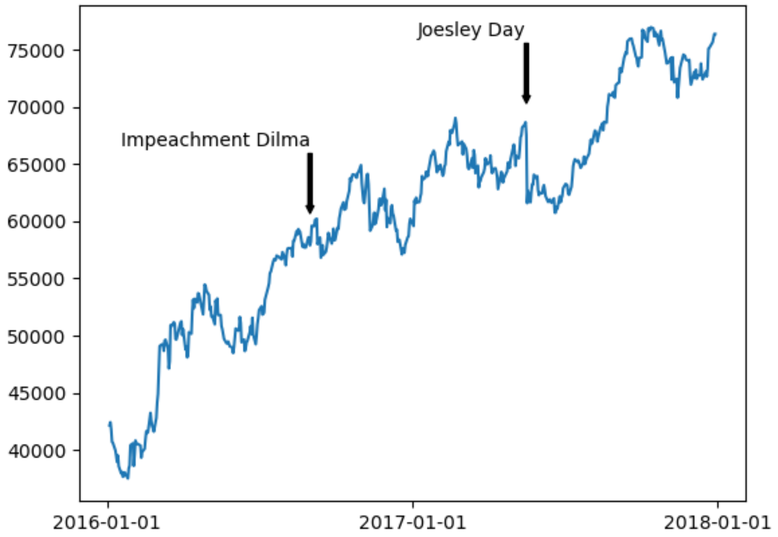
others:

Parâmetros do Text(). Podem ser utilizados neste método.

Exemplo:

```
import matplotlib.pyplot as plt
import matplotlib.dates as mdate
from datetime import datetime
import yfinance as yf
import pandas as pd

cotacoes = yf.download(["^BVSP"], "2016-01-01", "2018-01-01")['Adj Close']
data_crises = [(datetime(2017, 5, 17), "Joesley Day"),
               (datetime(2016, 8, 31), "Impeachment Dilma")]

fig, ax = plt.subplots()
ax.plot(cotacoes)
for data, evento in data_crises:
    ax.annotate(evento, xy = (data, cotacoes.asof(data) + 2800), #posição comeco da seta
                xytext = (data, cotacoes.asof(data) + 10000), #posição texto da seta
                arrowprops = dict(facecolor = 'black', headwidth = 4,
                width = 2, headlength = 4), #propriedades da seta
                horizontalalignment = "right", verticalalignment = "top") #alinhamento do texto
 ax.xaxis.set_major_locator(mdate.YearLocator(1))
plt.show()
```

Respostas:



Mundo 14

Neste mundo aprenderemos a criar gráficos animados.

14.1. Pré-configurações

Antes de tudo, você deve entender que: Animações em gráficos são vídeos normais. Vídeos são dezenas de fotos em sequência e, no caso dos gráficos, criaremos um gráfico para cada frame do vídeo. Para isso, utilizaremos um submódulo da biblioteca matplotlib que é indicado para isso.

Se você estiver utilizando o Jupyter Notebook, você precisa, antes de tudo, escrever essa linha de código antes de qualquer código.

%matplotlib notebook

Essa linha de código vai fazer com que seja possível a visualização de gráficos animados dentro do Jupyter.

14.2. matplotlib.animation.FuncAnimation(fig, func, frames = None, init_func = None, fargs = None, save_count = None, cache_frame_data = True, others)

Este método é usado para fazer animações dentro dos gráficos. **Este é um método da biblioteca matplotlib**.

Parâmetros:

fig:

Este parâmetro vai definir o objeto da figura. O objeto da figura nada mais é que a figura do objeto, obtido por:

```
fig, ax = plt.subplots()
```

sendo:

fig => Figura, utilizada quando queremos acessar atributos da imagem ou salvar uma imagem.

ax => Objetos do eixo, quando queremos acessar atributos dos eixos.

É um parâmetro **opcional** que deve receber a variável fig já definida anteriormente.

fig:

Este parâmetro vai definir a função animação. É uma função que você vai criar e pode conter o que você desejar dentro dela.

É um parâmetro **opcional** que deve receber a função já pronta a ser utilizada. Pense nessa função sendo uma estrutura de repetição, dessa forma você terá mais facilidade em trabalhar com ela.

frames:

Este parâmetro vai definir a quantidade de frames a serem utilizados no gráfico. Pense neles como se fossem cada ponto que você deva ter no gráfico. Se há um DataFrame com 100 valores, você pode ter até 100 frames.

É um parâmetro **opcional** que deve receber um valor no formato integer. Pode ser também uma função do Python, como len(DataFrame) para se obter o tamanho do DataFrame.

interval:

Este parâmetro vai definir o intervalo entre as aparições de cada frame.

É um parâmetro **opcional** que deve receber um valor em milissegundos no formato integer.

repeat:

Este parâmetro vai definir se a plotagem vai se repetir depois da animação.

É um parâmetro **opcional** que receberá um booleano com True ou False.

repeat_delay:

Este parâmetro vai definir o intervalo entre cada repetição. Só funcionará se repeat = True.

É um parâmetro **opcional** que deve receber um valor em milissegundos no formato integer.

14.2. Plotando um gráfico com animação

1° passo:

Antes de tudo, vamos importar todos os módulos necessários.

Exemplo:

```
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from datetime import date
import yfinance as yf
```

2° passo:

Vamos importar os dados diários do ibovespa e transformar em mensais. Porque na hora de plotar a animação serão menos frames, e por isso, fará com que a visualização fique melhor.

Exemplo:

```
cotacoes = yf.download(["^BVSP","WEGE3.SA"], "2001-12-28", "2023-01-01")['Adj Close']
cotacoes_ani = cotacoes.resample("M").last()["^BVSP"]
```

Respostas:

```
Date
2001-12-31                                13578.0
2002-01-31                                12721.0
2002-02-28                                14033.0
2002-03-31                                13255.0
2002-04-30                                13085.0
                                             ...
2022-08-31                               109523.0
2022-09-30                               110037.0
2022-10-31                               116037.0
2022-11-30                               112486.0
2022-12-31                               110031.0
Freq: M, Name: ^BVSP, Length: 253, dtype: float64
```

3° passo:

Vamos criar um gráfico e definir limites para os eixos x e eixo y. Isso porque na hora de criar a animação, por serem vários gráficos, um sobreposto ao outro, ele define limites diferentes, o que faz com que a visualização seja ruim.

Exemplo:

```
fig, ax = plt.subplots()
ax.set_xlim(date(2001, 1, 1), date(2023, 12, 31))
ax.set_ylim(6000, 130000)
```

Respostas:

4° passo:

Vamos criar uma função que definirá como será nosso gráfico a cada frame. Repare que nessa função, na hora de plotar, definimos a cor como azul. Isso porque caso a gente não tivesse definido, o matplot definiria uma cor diferente para cada frame.

Exemplo:

```
def animate(i):

    data = cotacoes_ani.iloc[:int(i+1)] #select data range
    ax.plot(data.index, data.values, color = "b")
```

5° passo:

E por fim, vamos utilizar a função FuncAnimation(). Onde definiremos a nossa figura do gráfico, a função que será utilizada para plotagem, a quantidade de frames, o intervalo a cada frame e se terá repetição ou não.

Exemplo:

```
ani = FuncAnimation(fig, animate, frames = range(0, len(cotacoes_ani)), interval=50, repeat = False)
```

Resultado final:

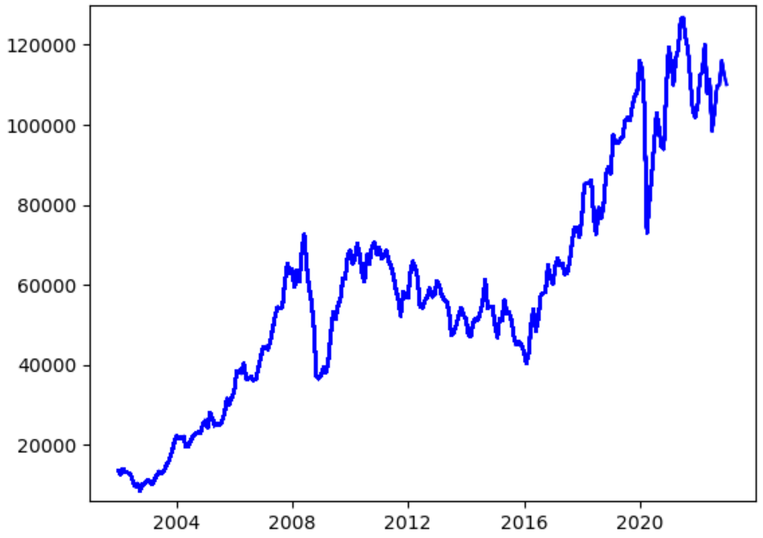
Exemplo:

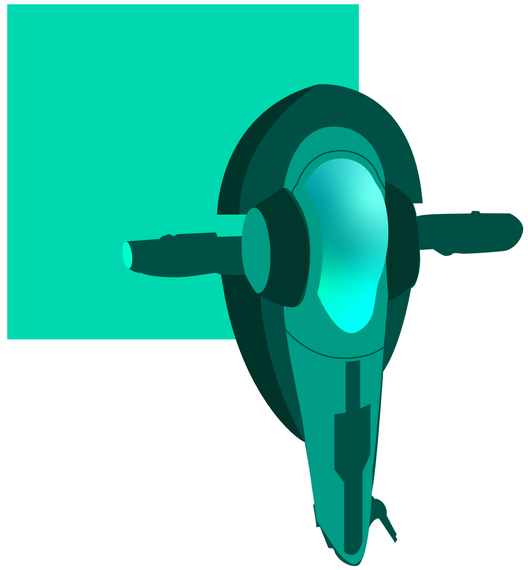
```
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from datetime import date
import yfinance as yf

cotacoes = yf.download(["^BVSP","WEGE3.SA"], "2001-12-28", "2023-01-01")['Adj Close']
cotacoes_ani = cotacoes.resample("M").last()["^BVSP"]

fig, ax = plt.subplots()
ax.set_xlim(date(2001, 1, 1), date(2023, 12, 31))
ax.set_ylim(6000, 130000)

def animate(i):

    data = cotacoes_ani.iloc[:int(i+1)] #select data range
    ax.plot(data.index, data.values, color = "b")


ani = FuncAnimation(fig, animate, frames = range(0, len(cotacoes_ani)), interval=50, repeat = False)

plt.show()
```

Respostas:

Esse script gerará uma animação na qual terá esse gráfico como resultado final.

Próximo





Mundo 15

Neste mundo aprenderemos a criar gráficos de calor através da biblioteca Seaborn.

15.1. Plotando um gráfico de calor

1° passo:

Antes de tudo, vamos importar todos os módulos necessários.

Exemplo:

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
import numpy as np
import matplotlib.dates as mdate
from pandas_datareader import data as pdr
from datetime import datetime
import seaborn as sns
```

2° passo:

Vamos importar os dados diários das ações presentes na lista_acoes_inicial mas repare que para importar essas informações do yahoo finance, os tickers devem ter o sufixo **".SA"** e o ibovespa será chamado de **"^BVSP"**. Pois é assim que o Yahoo Finance reconhece esses tickers.

Exemplo:

```
lista_acoes_inicial = ["WEGE3", "VALE3", "PETR4", "ABEV3", "IBOV"]

lista_acoes = [acao + ".SA" for acao in lista_acoes_inicial]

if "IBOV.SA" in lista_acoes:

    lista_acoes.remove("IBOV.SA")
    lista_acoes.append("^BVSP")

cotacoes = yf.download(lista_acoes ,"2008-12-28", "2023-01-01")['Adj Close']
```

Respostas:

```
            ABEV3.SA   PETR4.SA  ...   WEGE3.SA     ^BVSP
Date                               ...
2008-12-29  1.981542   8.069036  ...   1.351673   37060.0
2008-12-30  2.048963   8.183696  ...   1.386047   37550.0
2009-01-02  2.150698   8.792815  ...   1.421529   40244.0
2009-01-05  2.078204   8.993464  ...   1.457012   41519.0
2009-01-06  2.053313   9.118871  ...   1.485842   42312.0
...              ...        ...  ...        ...       ...
2022-12-23  14.620000  25.120001  ...  38.500000  109698.0
2022-12-26  14.520000  24.940001  ...  38.189999  108738.0
2022-12-27  14.520000  25.110001  ...  38.009998  108347.0
2022-12-28  14.550000  24.799999  ...  38.700001  110237.0
2022-12-29  14.520000  24.500000  ...  38.509998  110031.0

[3475 rows x 5 columns]
```

3° passo:

Agora criaremos uma função que calcula retornos acumulados de acordo com as janelas móveis e depois anualizar esses retornos. Isso para que a gente possa conferir o retorno acumulado anual em diferentes intervalos de tempo. No caso abaixo para os primeiros 1,3,5 e 10 anos.

Exemplo:

```python
def janelas_moveis_retorno_acumulado(df, dias, anos):

    janela_movel = df.pct_change(periods = dias).dropna()

    lista_retornos = []

    for coluna in df.columns:

        media_retorno_acumulado = janela_movel[coluna].mean()
        valor_ao_ano = (1 + media_retorno_acumulado)**(1/anos) - 1
        lista_retornos.append(valor_ao_ano)

    return lista_retornos

janela_1_ano_geral = janelas_moveis_retorno_acumulado(cotacoes, 250, anos = 1)
janela_3_anos_geral = janelas_moveis_retorno_acumulado(cotacoes, 250 * 3, anos = 3)
janela_5_anos_geral = janelas_moveis_retorno_acumulado(cotacoes, 250 * 5, anos = 5)
janela_10_anos_geral = janelas_moveis_retorno_acumulado(cotacoes, 250 * 10, anos = 10)
```

4° passo:

Depois de ter calculado o retorno acumulado para cada intervalo de tempo definido. Criaremos um DataFrame com cada retorno respectivamente a sua empresa, para que a gente possa fazer a análise no geral, comparando ano a ano.

E abaixo, na estrutura de repetição, estaremos fazendo um arredondamento e colocando tudo em porcentagem.

Exemplo:

```
tabela_janelas_moveis = pd.DataFrame(data = {'1 ano': janela_1_ano_geral,
                                             '3 anos': janela_3_anos_geral,
                                             '5 anos': janela_5_anos_geral,
                                             '10 anos': janela_10_anos_geral},
                                                index=lista_acoes_inicial)

for coluna in tabela_janelas_moveis.columns:

  tabela_janelas_moveis[f'{coluna}'] = (tabela_janelas_moveis[f'{coluna}'].round(2)) * 100
```

Respostas:

```
        1 ano  3 anos  5 anos  10 anos
WEGE3    17.0    16.0    15.0     13.0
VALE3    13.0    10.0    10.0      5.0
PETR4    20.0    18.0    19.0     10.0
ABEV3    31.0    32.0    29.0     28.0
IBOV      8.0     7.0     8.0      6.0
```

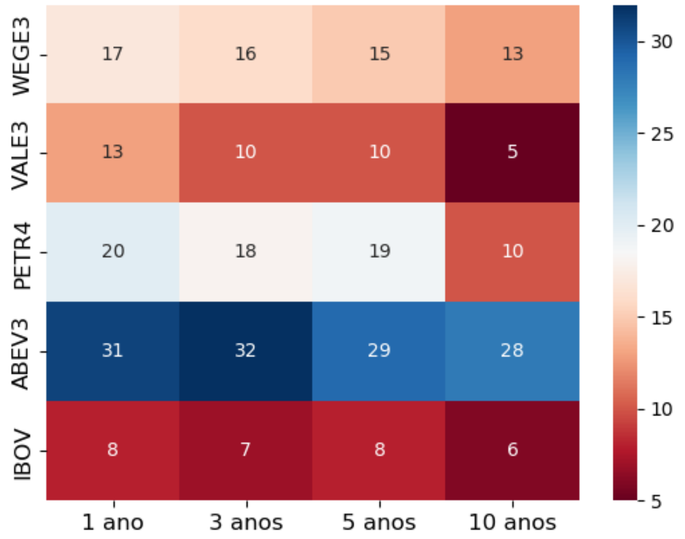
5° passo:

Após a conclusão do DataFrame, utilizaremos a biblioteca Seaborn para criação do gráfico de calor. E logo abaixo já definiremos um título e o tamanho da fonte das palavras, e dos números, do eixo x e eixo y.

Exemplo:

```
ax = sns.heatmap(tabela_janelas_moveis, annot=True, cmap="RdBu", fmt = "g")

plt.title("Mapa de calor do retorno médio (ao ano) de cada ativo", color = "w")
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
```

Respostas:



6° passo:

Agora é só adicionar "%" aos números que estão no gráfico.

Exemplo:

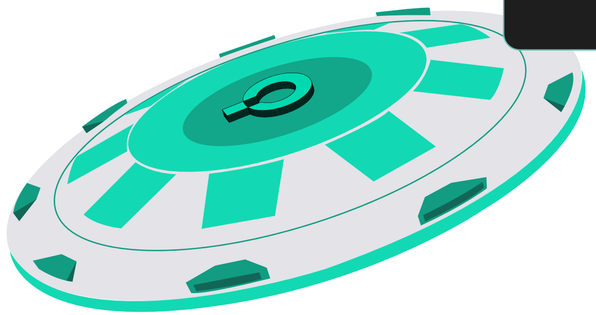
```
for t in ax.texts:
    t.set_text(t.get_text() + "%")
```

Resultado final:

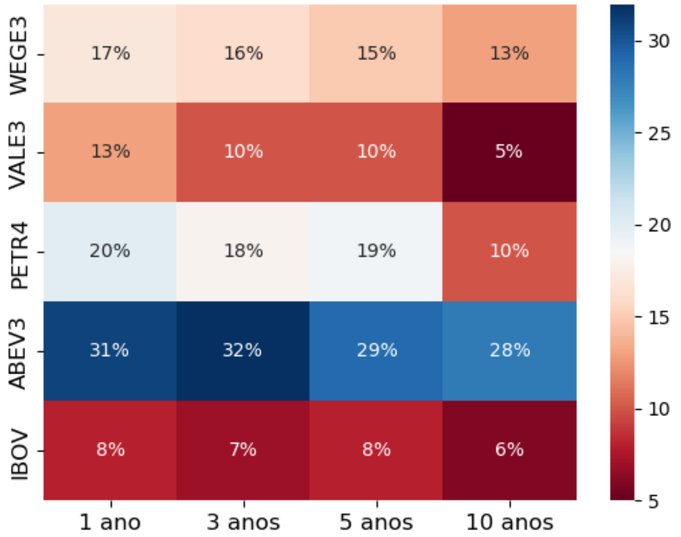
Exemplo:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import yfinance as yf

lista_acoes_inicial = ["WEGE3", "VALE3", "PETR4", "ABEV3", "IBOV"]

lista_acoes = [acao + ".SA" for acao in lista_acoes_inicial]

if "IBOV.SA" in lista_acoes:

    lista_acoes.remove("IBOV.SA")
    lista_acoes.append("^BVSP")

cotacoes = yf.download(lista_acoes ,"2008-12-28", "2023-01-01")['Adj Close']

def janelas_moveis_retorno_acumulado(df, dias, anos):

    janela_movel = df.pct_change(periods = dias).dropna()

    lista_retornos = []

    for coluna in df.columns:

        media_retorno_acumulado = janela_movel[coluna].mean()
        valor_ao_ano = (1 + media_retorno_acumulado)**(1/anos) - 1
        lista_retornos.append(valor_ao_ano)

    return lista_retornos
```

```
janela_1_ano_geral = janelas_moveis_retorno_acumulado(cotacoes, 250, anos = 1)
janela_3_anos_geral = janelas_moveis_retorno_acumulado(cotacoes, 250 * 3, anos = 3)
janela_5_anos_geral = janelas_moveis_retorno_acumulado(cotacoes, 250 * 5, anos = 5)
janela_10_anos_geral = janelas_moveis_retorno_acumulado(cotacoes, 250 * 10, anos = 10)

tabela_janelas_moveis = pd.DataFrame(data = {'1 ano': janela_1_ano_geral,
                                             '3 anos': janela_3_anos_geral,
                                             '5 anos': janela_5_anos_geral,
                                             '10 anos': janela_10_anos_geral},
                                                index=lista_acoes_inicial)

for coluna in tabela_janelas_moveis.columns:

    tabela_janelas_moveis[f'{coluna}'] = (tabela_janelas_moveis[f'{coluna}'].round(2)) * 100

ax = sns.heatmap(tabela_janelas_moveis, annot=True, cmap="RdBu", fmt = "g")

plt.title("Mapa de calor do retorno médio (ao ano) de cada ativo", color = "w")
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

for t in ax.texts:
    t.set_text(t.get_text() + "%")

plt.show()
```



Respostas:


WEGE3
VALE3
PETR4
ABEV3
IBOV
17%
16%
15%
13%
13%
10%
10%
5%
20%
18%
19%
10%
31%
32%
29%
28%
8%
7%
8%
6%
1 ano
3 anos
5 anos
10 anos
30
25
20
15
10
5

Mundo 16

Neste mundo aprenderemos a criar gráficos de calor através da biblioteca Quantstats.

16.1. Plotando um gráfico de calor

1° passo:

Antes de tudo, vamos importar todos os módulos necessários.

Exemplo:

```
import quantstats as qs
import yfinance as yf
import pandas as pd
```

2° passo:

Vamos importar os dados diários das ações da PETR4.SA. E transformá-los em retornos mensais.

Exemplo:

```
cotacoes = pdr.get_data_yahoo('PETR4.SA' ,"2008-12-28", "2023-01-01")['Adj Close']

retornos = cotacoes.resample("M").last().pct_change().dropna()
```

Resposta:

```
 Date
2009-01-31                                0.095884
2009-02-28                                0.054734
2009-03-31                                0.081439
2009-04-30                                0.046996
2009-05-31                                0.166216
                                             ...
2022-08-31                                0.194980
2022-09-30                               -0.103220
2022-10-31                                0.000336
2022-11-30                                0.022153
2022-12-31                               -0.081020
Freq: M, Name: Adj Close, Length: 168, dtype: float64
```

3° passo:

Por meio deste método, iremos expandir as propriedades do pandas dentro da biblioteca quantstats.

Exemplo:

```
qs.extend_pandas()
```

4° passo:

Por fim, plotamos o gráfico de calor através da biblioteca pandas com a quantstats.

Exemplo:

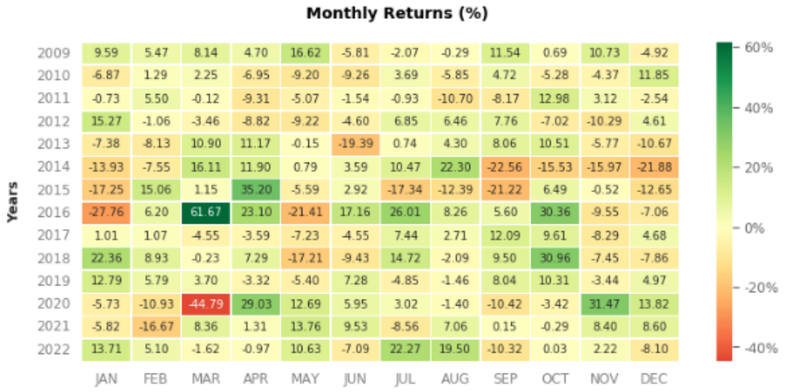
```
retornos.plot_monthly_heatmap()
```

Resultado final:

Exemplo:

```
import quantstats as qs
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt

cotacoes = yf.download('PETR4.SA' ,"2008-12-28", "2023-01-01")['Adj Close']
retornos = cotacoes.resample("M").last().pct_change().dropna()

print(retornos)

qs.extend_pandas()

retornos.plot_monthly_heatmap()

plt.show()
```

Respostas:


Monthly Returns (%)
Years
2009 9.59 5.47 8.14 4.70 16.62 -5.81 -2.07 -0.29 11.54 0.69 10.73 -4.92
2010 -6.87 1.29 2.25 -6.95 -9.20 -9.26 3.69 -5.85 4.72 -5.28 -4.37 11.85
2011 -0.73 5.50 -0.12 -9.31 -5.07 -1.54 -0.93 -10.70 -8.17 12.98 3.12 -2.54
2012 15.27 -1.06 -3.46 -8.82 -9.22 -4.60 6.85 6.46 7.76 -7.02 -10.29 4.61
2013 -7.38 -8.13 10.90 11.17 -0.15 -19.39 0.74 4.30 8.06 10.51 -5.77 -10.67
2014 -13.93 -7.55 16.11 11.90 0.79 3.59 10.47 22.30 -22.56 -15.53 -15.97 -21.88
2015 -17.25 15.06 1.15 35.20 -5.59 2.92 -17.34 -12.39 -21.22 6.49 -0.52 -12.65
2016 -27.76 6.20 61.67 23.10 -21.41 17.16 26.01 8.26 5.60 30.36 -9.55 -7.06
2017 1.01 1.07 -4.55 -3.59 -7.23 -4.55 7.44 2.71 12.09 9.61 -8.29 4.68
2018 22.36 8.93 -0.23 7.29 -17.21 -9.43 14.72 -2.09 9.50 30.96 -7.45 -7.86
2019 12.79 5.79 3.70 -3.32 -5.40 7.28 -4.85 -1.46 8.04 10.31 -3.44 4.97
2020 -5.73 -10.93 -44.79 29.03 12.69 5.95 3.02 -1.40 -10.42 -3.42 31.47 13.82
2021 -5.82 -16.67 8.36 1.31 13.76 9.53 -8.56 7.06 0.15 -0.29 8.40 8.60
2022 13.71 5.10 -1.62 -0.97 10.63 -7.09 22.27 19.50 -10.32 0.03 2.22 -8.10
JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC
60%
40%
20%
0%
-20%
-40%

Conclusão

Com isso chegamos ao fim do nosso módulo de matplotlib. Aprendemos como apresentar, analisar visualmente dados que antes eram apenas apresentados numericamente. Use esse pdf sempre que quiser consultar ou precisar de ajuda e bora para a próxima galáxia.

Referências

1. Formatação de cor

Caractere	Cor
b	Azul
g	Verde
r	Vermelho
c	Ciano
m	Magenta
y	Amarelo
k	Preto
w	Branco

2. Formatação da linha

Caractere	Tipo de Linha
-	Linha cheia
--	Linha Tracejada
:	Linha traço-ponto
:-	Linha pontilhada

3. Formatação dos marcadores

Caractere	Tipos dos Marcadores
.	Ponto
,	Pixel
o	Círculo
v	Triângulo
^	Triângulo
<	Triângulo
>	Triângulo
1	Y para baixo
2	Y para cima
3	Y para esqueda
4	Y para direita
s	Quadrado
p	Pentágono
*	Estrela
h	Hexágono
H	Hexágono
+	Sinal de mais
x	Sinal cruzado
D	Losango largo
d	Losango estreito
|	Linha vertical
_	Linha horizontal

4. Formatação dos gráficos

kind	Tipos de Gráficos
line	Gráfico de linha
bar	Gráfico de barras
barh	Gráfico de barras horizontal
hist	Gráfico de histograma
box	Gráfico de diagrama de caixa
kde	Gráfico de estimativa de densidade
density	Gráfico de função densidade
area	Gráfico de área
pie	Gráfico de pizza
scatter	Gráfico de dispersão
hexbin	Gráfico hexagonal Binning

5. Formatação do colormap

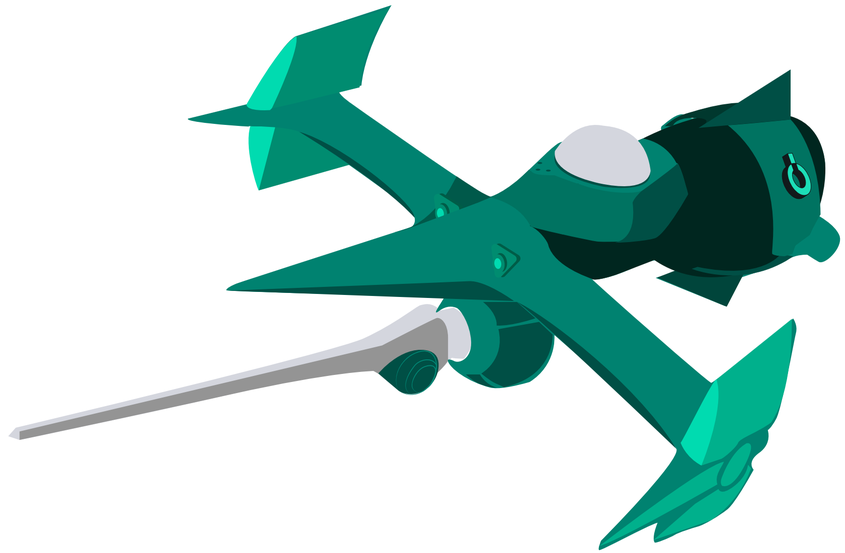
Nº	Cores	Nº	Cores	Nº	Cores	Nº	Cores
1	Accent	11	CMRmap	21	OrRd	31	Pastel2
2	Accent_r	12	CMRmap_r	22	OrRd_r	32	Pastel2_r
3	Blues	13	Dark2	23	Oranges	33	PiYG
4	Blues_r	14	Dark2_r	24	Oranges_r	34	PiYG_r
5	BrBG	15	GnBu	25	PRGn	35	PuBu
6	BrBG_r	16	GnBu_r	26	PRGn_r	36	PuBuGn
7	BuGn	17	Greens	27	Paired	37	PuBuGn_r
8	BuGn_r	18	Greens_r	28	Paired_r	38	PuBu_r
9	BuPu	19	Greys	29	Pastel1	39	PuOr
10	BuPu_r	20	Greys_r	30	Pastel1_r	40	PuOr_r

41	PuRd	51	RdYlBu	61	37865	71	YlOrBr
42	PuRd_r	52	RdYlBu_r	62	Set3_r	72	YlOrBr_r
43	Purples	53	RdYlGn	63	Spectral	73	YlOrRd
44	Purples_r	54	RdYlGn_r	64	Spectral_r	74	YlOrRd_r
45	RdBu	55	Reds	65	Wistia	75	afmhot
46	RdBu_r	56	Reds_r	66	Wistia_r	76	afmhot_r
47	RdGy	57	37135	67	YlGn	77	autumn
48	RdGy_r	58	Set1_r	68	YlGnBu	78	autumn_r
49	RdPu	59	37500	69	YlGnBu_r	79	binary
50	RdPu_r	60	Set2_r	70	YlGn_r	80	binary_r

81	bone	91	coolwarm	101	gist_gray	111	gist_yarg
82	bone_r	92	coolwarm_r	102	gist_gray_r	112	gist_yarg_r
83	brg	93	copper	103	gist_heat	113	gnuplot
84	brg_r	94	copper_r	104	gist_heat_r	114	gnuplot2
85	bwr	95	cubehelix	105	gist_ncar	115	gnuplot2_r
86	bwr_r	96	cubehelix_r	106	gist_ncar_r	116	gnuplot_r
87	cividis	97	flag	107	gist_rainbow	117	gray
88	cividis_r	98	flag_r	108	gist_rainbow_r	118	gray_r
89	cool	99	gist_earth	109	gist_stern	119	hot
90	cool_r	100	gist_earth_r	110	gist_stern_r	120	hot_r

121	hsv	131	ocean	141	seismic	151	tab20b
122	hsv_r	132	ocean_r	142	seismic_r	152	tab20b_r
123	inferno	133	pink	143	spring	153	tab20c
124	inferno_r	134	pink_r	144	spring_r	154	tab20c_r
125	jet	135	plasma	145	summer	155	terrain
126	jet_r	136	plasma_r	146	summer_r	156	terrain_r
127	magma	137	prism	147	tab10	157	twilight
128	magma_r	138	prism_r	148	tab10_r	158	twilight_r
129	nipy_spectral	139	rainbow	149	tab20	159	twilight_shifted
130	nipy_spectral_r	140	rainbow_r	150	tab20_r	160	twilight_shifted_r

161	viridis	162	viridis_r	163	winter	164	winter_r



6. Tabela para localização da legenda

Localização	Código
best	0
upper right	1
upper left	2
lower left	3
lower right	4
right	5
center left	6
center right	7
lower center	8
upper center	9
center	10

7. Tabela para o tamanho da legenda

Formato	Tamanho
xx-small	Super pequeno
x-small	Muito pequeno
small	Pequeno
medium	Médio
large	Grande
x-large	Muito grande
xx-large	Super grande

8. Tabela de estilos disponíveis padrão do matplotlib

Nº	Estilos	Nº	Estilos
1	Solarize_Light2	15	seaborn-dark
2	_classic_test_patch	16	seaborn-dark-palette
3	_mpl-gallery	17	seaborn-darkgrid
4	_mpl-gallery-nogrid	18	seaborn-deep
5	bmh	19	seaborn-muted
6	classic	20	seaborn-notebook
7	dark_background	21	seaborn-paper
8	fast	22	seaborn-colorblind
9	fivethirtyeight	23	seaborn-poster
10	ggplot	24	seaborn-talk
11	grayscale	25	seaborn-ticks
12	seaborn	26	seaborn-white
13	seaborn-bright	27	seaborn-whitegrid
14	seaborn-colorblind	28	tableau-colorblind10

9. Tabela de paleta de cores seaborn

Nº	Cores
1	pastel
2	husl
3	Set2
4	Spectral
5	flare