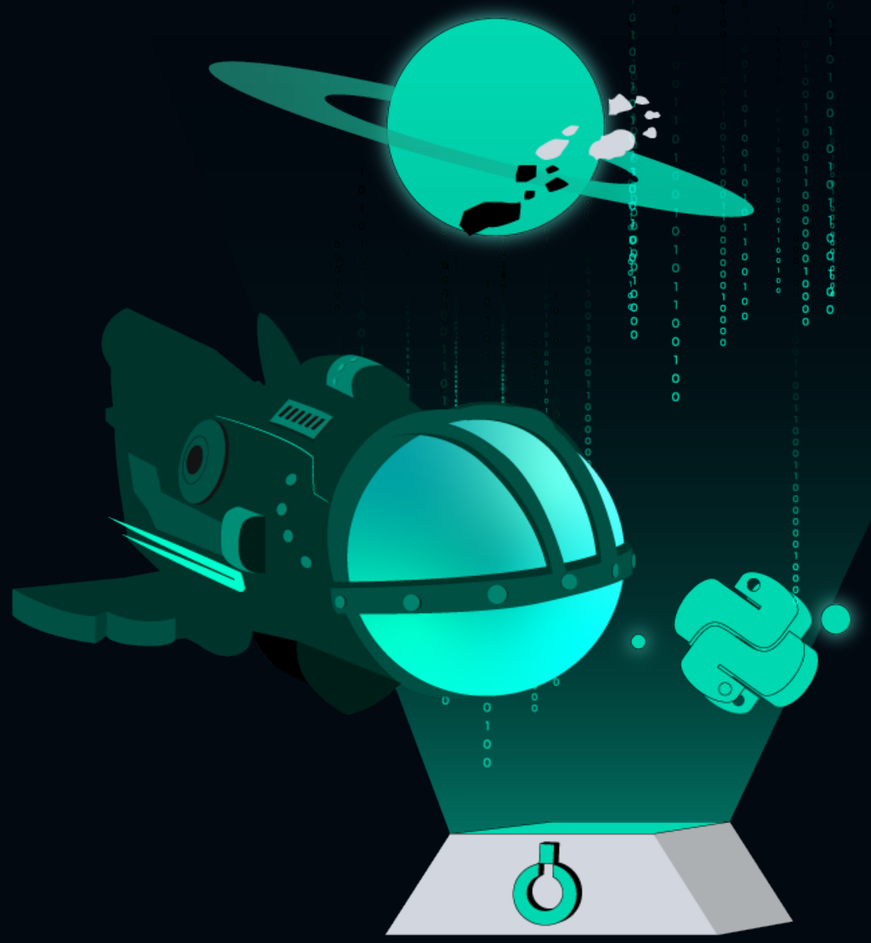
código.py

GALÁXIA 4

Pandas



Galáxia 4

Introdução aos pandas.

Mundo 1

Mundo 2

2.1. pandas.Series

2.2. pandas.Datafame

Mundo 3

3.1. pandas.DataFrame.set_index

3.2. pandas.DataFrame.reset_index

3.3. pandas.date_range

Mundo 4

4.1. pandas.DataFrame.columns

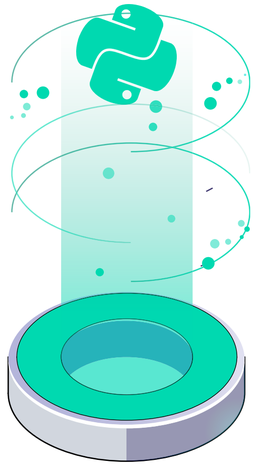
Mundo 5

5.1. pandas.DataFrame.loc

5.2. pandas.DataFrame.at

5.3. pandas.DataFrame.iloc

5.4. pandas.DataFrame.iat



Mundo 6

6.1. pandas.DataFrame.sum

6.2. pandas.DataFrame.cumsum

6.3. pandas.DataFrame.max

6.4. pandas.DataFrame.min

Mundo 7

7.1. pandas.DataFrame.sort_index

7.2. pandas.DataFrame.sort_values

7.3. pandas.DataFrame.rank

Leitura e extração de dados

Mundo 8

8.1. pandas.read_excel

8.2. DataFrame.to_excel

Mundo 9

9.1. pandas.read_csv

9.2. pandas.to_csv

Mundo 10

10.1. Dados que podem ser extraídos do Pandas-Datareader

Mundo 11

11.1. pandasdatareader.get_data_yahoo

Mundo 12

12.1. pandasdatareader.get_data_fred

Mundo 13

13.1. wb.get_indicators

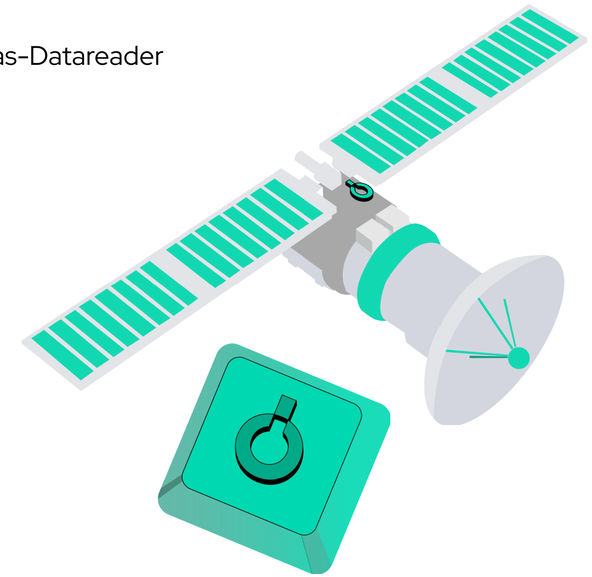
13.2. wb.search

13.3. wb.download

Mundo 14

14.1. pandasdatareader.get_avaible_datasets

14.2. pandasdatareader.get_data_famafrench



Mundo 15

15.1. sqlalchemy.create_engine

15.2. DataFrame.to_sql

15.3. pandas.read_sql

Limpeza e tratamentos de dados

Mundo 16

16.1. DataFrame.dropna

16.2. DataFrame.fillna

Mundo 17

17.1. pandas.unique

17.2. DataFrame.drop_duplicates

Mundo 18

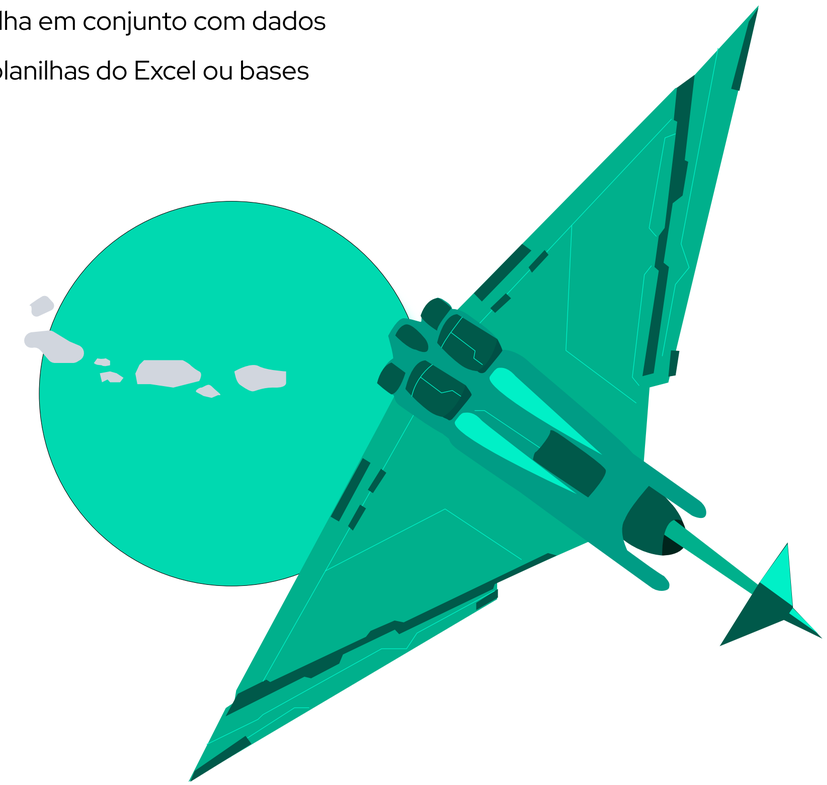
18.1. DataFrame.astype

18.2. DataFrame.to_datetime

18.3. DataFrame.replace

Introdução

Olá, seja bem vindo à Galáxia 4!! Neste módulo iremos abordar sobre a biblioteca Pandas. Ela é uma biblioteca de código aberto do Python que fornece estruturas de dados de alto nível e ferramentas para análise de dados. Esta biblioteca também trabalha em conjunto com dados tabulares, como aqueles encontrados em planilhas do Excel ou bases de dados relacionais.



Mundo 1

O objetivo desta Galáxia é ser uma grande caixa de ferramentas. Neste pdf você vai encontrar as principais funções do pandas traduzidas e explicadas para você. Esta biblioteca é a mais, ou uma das mais importantes, visto que tudo que cerca análises de dados utiliza o pandas. Manipulação, operações de tabelas e criação de séries temporais são algumas das muitas coisas que o pandas pode fazer por você. Além disso, o pandas pode ser aplicado a vários campos diferentes como: finanças, estatística, machine learning.

Mundo 2

Neste mundo abordaremos alguns métodos de como modelar os dados dentro do Python, utilizando a biblioteca pandas.

1. pandas.Series(data = None, index = None, dtype = None, name = None, copy = False, fastpath = False)

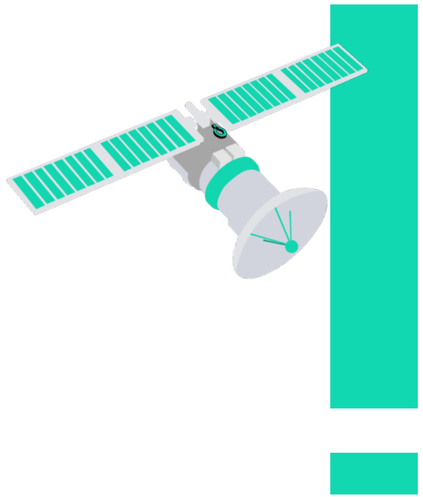
Método que cria matrizes unidimensionais capazes de armazenar dados de todos os tipos (integer, string, float, objetos etc.). Fazendo um comparativo com o Excel, a Series representa uma coluna em uma planilha Excel.

Parâmetros:

Os parâmetros copy e fastpath vêm definidos como False, enquanto os outros vêm definidos como vazios.

data:

São os dados que vão compor a Series.



Esse parâmetro **é obrigatório** e pode ser passado em forma de lista, dicionário ou array (criado a partir do NumPy).

index:

É a rotulação dos dados e deve ter o mesmo tamanho do argumento data.

Esse parâmetro é opcional e pode ser passado em forma de lista.

obs: Quando se cria uma Series a partir de um dicionário, o index vem automaticamente.

dtype:

É o formato dos dados, definido apenas uma vez.

Esse parâmetro é opcional e pode ser: integer, string, float ou date.

name:

É o nome da Series.

Esse parâmetro **é opcional** e é dado em string.

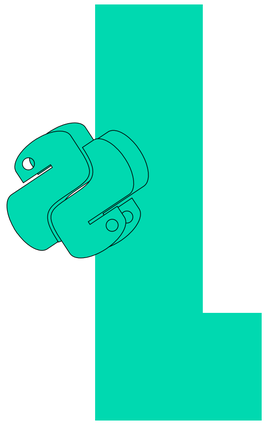
Exemplo de aplicação:

Criando uma Series a partir de um vetor aleatório de 5 itens.

```
import pandas as pd
import numpy as np
serie_inicial = pd.Series(np.random.randn(5), index = ["A", "B", "C", "D", "E"])
print(serie_inicial)
```

Resposta:

```
>>> A   -0.728375
    B    0.445204
    C   -0.424502
    D   -0.582674
    E    0.444478
    dtype: float64
```



2. pandas.DataFrame(data = None, index = None, columns = None, dtype = None, copy = None)

Método que cria estruturas de dados tabular bidimensionais, capazes de armazenar dados de todos os tipos (integer, string, float, objetos, etc...). Fazendo um comparativo com o Excel, o DataFrame representa uma tabela em uma planilha Excel.

Parâmetros:

Todos os parâmetros vêm definidos como vazios.

data:

São os dados que vão compor o DataFrame.

Esse parâmetro **é obrigatório** e pode ser passado em forma de Series, lista, dicionário ou array (criado a partir do NumPy).

index:

É a rotulação dos dados.

Esse parâmetro **é opcional** e pode ser passado em forma de lista.

column:

É o nome das colunas do DataFrame, deve ter a mesma dimensão dos dados.

Esse parâmetro **é opcional** e pode ser passado em forma de lista.

dtype:

É o formato dos dados, definido apenas uma vez.

Esse parâmetro **é opcional** e pode ser: integer, string, float ou date.

copy:

Esse conceito é mais complexo. Ao definir copy=False, estamos definindo que as mudanças se propagarão entre os objetos. Isso quer dizer que se eu mudar um DataFrame criado por um vetor, este vetor mudará também.

obs: Isso só se aplica se tivermos criado um DataFrame utilizando **vetores (criados pela biblioteca NumPy)**.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

Exemplo:

Criando um DataFrame 3x3, escolhendo a nomeação das colunas e definindo qual será o index.

```
import pandas as pd
dicionario_dados = {
   "empresas": ["Wege", "Vale", "Petrobras"],
   "price": [20, 30, 40],
   "volume": [1000, 4000, 7500]
}
dados_diarios_from_dict = pd.DataFrame(dicionario_dados, index = ["2019", "2020", "2021"],
columns = ["volume", "price", "empresas"])
print(dados_diarios_from_dict)
```

Resposta:

```
>>>         volume  price   empresas
2019    1000      20        Wege
2020    4000      30        Vale
2021    7500      40   Petrobras
```

Mundo 3

Neste mundo veremos formas de interagir com o index.

1. pandas.DataFrame.set_index(keys, drop = True, append = False, inplace = False, verify_integrity = False)

É um método utilizado, em DataFrames, para definir index através das colunas existentes.

Parâmetros:

Os parâmetros append, inplace, verify_integrity vêm definidos como False, enquanto o parâmetro drop vem definido como True.

keys:

Esse parâmetro vai definir quem vai se tornar index.

Esse parâmetro **é obrigatório**. Recebe o nome da coluna em string (em caso de 1 index), ou uma lista com os nomes das colunas em string (em caso de um ou mais index).

drop:

Esse parâmetro vai definir se as colunas, escolhidas para se tornarem index, serão deletadas. Por padrão, o pandas considera "drop=True", ou seja, caso não coloque nada o pandas excluirá a coluna definida (pois ela já se tornou um index).

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

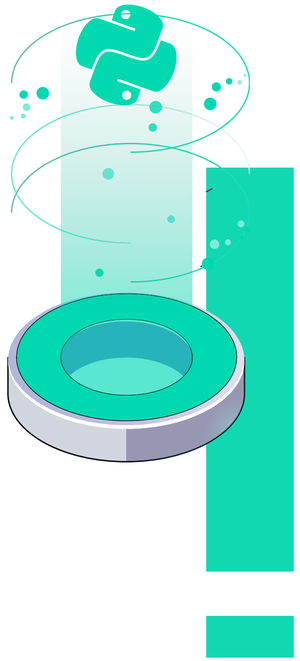
append:

Se o index definido será anexado, ou substituirá o index já existente. Caso não tenha um index ele apenas anexará o novo. Por padrão o pandas considera “append=False”, ou seja, ele substituirá em caso de um index já existente.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

inplace:

Se deve fazer a modificação direto no DataFrame ou não. Esse parâmetro pode ser interpretado por uma forma de salvar as modificações diretamente no DataFrame. Por padrão o pandas considera “inplace=False”, ou seja, ele não salvará as modificações caso você não atribua a um objeto.



Esse parâmetro **é opcional** e recebe um booleano: True ou False.

verify_integrity:

Verifica se o index definido possui duplicatas, se sim ele retornará um erro. Por padrão o pandas considera “verify_integrity=False”, ou seja, ele não verificará a existência de duplicatas, logo não retornará um erro.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

Exemplo:

Neste exemplo é criado um dicionário para ser transformado em **DataFrame**. Depois, a coluna “empresa” é definida como index.

```
import pandas as pd
dicionario = {"empresa": ["Weg", "Petrobras", "Vale"], "cotacao": [20.54, 20, 40]}
dataframe = pd.DataFrame(dicionario)
print(f''' DataFrame sem index definido:
{dataframe}''')
dataframe = dataframe.set_index("empresa")
print(f''' DataFrame com index definido:
{dataframe}''')
```

Resposta:

```
>>>   DataFrame sem index definido:
          empresa  cotacao
      0        Weg    20.54
      1  Petrobras    20.00
      2       Vale    40.00
>>>   DataFrame com index definido:
               cotacao
   empresa
   Weg           20.54
   Petrobras     20.00
   Vale          40.00
```

2. pandas.DataFrame.reset_index(level = None, drop = False, inplace = False, col_level = 0, col_fill = " ", allow_duplicates = False, names = None)

É um método utilizado, em DataFrames, para redefinir o index ou o nível deles. Existe um método parecido, quase idêntico, com esse para Series. Mas para esse PDF nos limitaremos apenas ao método para DataFrames.

Parâmetros:

Caso você não defina um parâmetro, ele redefinirá todos os index existentes no DataFrame.

Os parâmetros drop e inplace, vêm definidos como False, enquanto os parâmetros level e name são vazios.

level:

Esse parâmetro vai definir qual index será redefinido.

Esse parâmetro **é opcional**. Pode ser o nome de um index ou uma lista com nome dos index, em string. Pode ser também a posição de um index(começando no 0) ou uma lista com as posições dos index, em integer.

drop:

Esse parâmetro vai definir se os index escolhidos serão excluídos ou não. Por padrão o pandas considera “drop=False”, ou seja, os index não serão excluídos. Portanto eles irão se tornar colunas que compõe o DataFrame.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

inplace:

Esse parâmetro vai definir se as modificações serão feitas diretamente no DataFrame ou não. Esse parâmetro pode ser interpretado por: uma forma de salvar as modificações diretamente no DataFrame. Por padrão o pandas considera “inplace=False”, ou seja, ele não salvará as modificações caso você não atribua a um objeto.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

allow_duplicates:

Esse parâmetro define se o nome das colunas podem se repetir. Por padrão o pandas considera “allow_duplicates=False”, ou seja, ele não aceitará nomes de colunas iguais.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

names:

Esse parâmetro define como serão chamadas as colunas que antes eram indexadas. Por padrão esse parâmetro vem vazio, ou seja, as novas colunas terão os antigos nomes dos index.

Esse parâmetro **é opcional**. Pode ser o nome de uma coluna ou uma lista com nome das colunas, em string.

col_level:

Esse parâmetro é utilizado quando os títulos das colunas dos DataFrames têm mais de um nível. Esse método especifica para qual dos níveis o index retornará. Por padrão esse parâmetro vem vazio, ou seja, ele considerará o primeiro nível que é o nível mais em cima.

Esse parâmetro **é opcional**, pode ser usado para redefinir um index ou a posição do título de uma coluna. Ele recebe a posição do nível que o index irá, em integer.

Deve ser utilizado em um DataFrame multi-level

Deve ser utilizado em conjunto com o parâmetro: **level**.

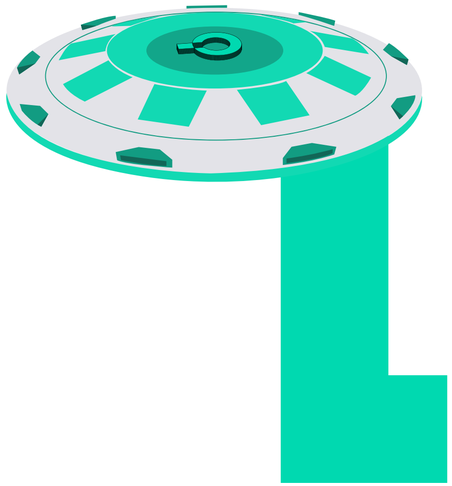
col_fill:

Esse parâmetro define como serão nomeados os outros níveis dos títulos das colunas.

Esse parâmetro **é opcional** e recebe o nome dos níveis em string.

Deve ser utilizado em um DataFrame multi-level

Deve ser utilizado em conjunto com os parâmetros: **level** e **col_level**.



Exemplo:

```python
import pandas as pd
dicionario = {"empresa": ["WEGE3", "PETR4", "VALE3"], "cotacao": [20.54, 20, 40],
"numero": [1,2,3], "segmento": ["Eletrico", "Petroleo", "Minerio"]}
dataframe = pd.DataFrame(dicionario)

dataframe = dataframe.set_index(["empresa", "cotacao"])
print(f'''DataFrame antes das modificações:
{dataframe}''')

dataframe = dataframe.reset_index(level=["empresa",'cotacao'],
names=['acoes','preco'],allow_duplicates=True)
print(f'''DataFrame após as modificações:
{dataframe}''')
```

Resposta:

```
>>> DataFrame antes das modificações:
                   numero  segmento
empresa cotacao
WEGE3   20.54           1  Eletrico
PETR4   20.00           2  Petroleo
VALE3   40.00           3   Minerio

>>> DataFrame após as modificações:
    acoes  preco  numero  segmento
0  WEGE3  20.54       1  Eletrico
1  PETR4  20.00       2  Petroleo
2  VALE3  40.00       3   Minerio
```

3. pandas.date_range(start = None, end = None, periods = None, freq = None, tz = None, normalize = False, name = None, closed = _NoDefault.no_default, inclusive = None, **kwargs)

É um método utilizado para criar intervalos de datas dentro do Python. As datas sempre serão retornadas no formato **aaaa-mm-dd**. Esse é o formato que o Pandas utiliza para trabalhar com datas.

Parâmetros:

Caso você não defina um parâmetro, ele retornará um erro. Obrigatoriamente, 3 dos 4 parâmetros(“**start**”,“**end**”, “**periods**” e “**freq**”) devem ser usados. Além disso, os 4 não podem ser usados de uma vez só. Deve-se mesclar a sintaxe e o sentido de acordo com o que você deseja.

start:

Esse parâmetro vai definir a primeira data do seu intervalo. Deve ser a data mais antiga ou ele retornará um erro.

Esse parâmetro é um dos **4 obrigatórios que devem ser mesclados entre si**, recebe a data no formato string entre aspas ou no formato date. Existem muitos formatos de datas que esse parâmetro reconhece, a seguir os mais famosos: dd/mm/aaaa ; dd/mm/aaaa ; dd-mm-aaaa ; dd--mm-aaaa ; aaaa-mm-dd ; aa-mm-dd.

end:

Esse parâmetro vai definir a última data do seu intervalo. Deve ser a data mais recente ou ele retornará um erro.

Esse parâmetro é um dos **4 obrigatórios que devem ser mesclados entre si**, recebe a data no formato string entre aspas ou no formato date. Existem muitos formatos de datas que esse parâmetro reconhece, a seguir os mais famosos: dd/mm/aaaa ; dd/mm/aaaa ; dd-mm-aaaa ; dd--mm-aaaa ; aaaa-mm-dd ; aa-mm-dd.

periods:

Esse parâmetro vai definir a quantidade de datas retornadas.

Esse parâmetro é um dos **4 obrigatórios que devem ser mesclados entre si**, recebe a quantidade de datas a serem retornadas, em integer.

freq:

Esse parâmetro vai definir a frequência dos intervalos, se será mensal, anual, diário ou apenas dos dias úteis.

Esse parâmetro é um dos 4 obrigatórios que devem ser mesclados entre si, recebe a frequência no formato string entre aspas. Para visualização da tabela.

tz:

Esse parâmetro vai definir o fuso horário das suas datas.

Esse parâmetro **é opcional** e recebe o nome dos fusos horários em string. O pandas determina alguns fuso horários. Para visualização da tabela.

normalize:

Esse parâmetro vai definir começo/fim da sua data como meia noite. Por padrão o pandas considera "normalize=False".

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

name:

Esse parâmetro vai definir o nome do seu intervalo.

Esse parâmetro **é opcional** e é dado em string.

inclusive:

Esse parâmetro vai definir se as datas estarão ou não dentro do intervalo

Esse parâmetro **é opcional** e é dado em string. As opções que o pandas disponibiliza são:

(both = os dois dentro do intervalo ; neither = os dois fora do intervalo ; right = só o da direita dentro do intervalo ; left = só o da esquerda dentro do intervalo)

Exemplo:

Nesse exemplo criaremos um DataFrame onde o intervalo de datas foi criado por nós e definido como index.

```
import pandas as pd

fim_mes_weg = pd.date_range(start = "1/1/2000", end = "30/4/2000", freq = "M")
dicionario = {"cotacao": [20.54, 20, 21, 22]}

cotacao_weg = pd.DataFrame(dicionario, index = fim_mes_weg)
cotacao_weg = cotacao_weg[cotacao_weg.index > "2000-02-25"]
print(cotacao_weg)
```

Resposta:

```
>>>                cotacao
2000-02-29      20.0
2000-03-31      21.0
2000-04-30      22.0
```

Mundo 4

Neste mundo veremos como ordenar e rankear os DataFrames.

1. pandas.DataFrame.columns

Esse é um método do pandas que retorna os rótulos das colunas. Pode ser utilizado para acessar e modificar uma coluna.

Parâmetros:

Não recebe nenhum parâmetro, mas precisa estar atribuído à um pandas DataFrame.

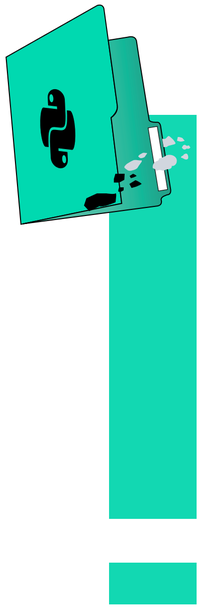
Exemplo:

No caso abaixo, as colunas foram acessadas para depois serem renomeadas.

```python
import pandas as pd

df_cambio = pd.DataFrame({
    "moeda": ['dolar/real', 'libra/real'] * 2,
    "price": [5.24, 4.15, 5.26, 4.23],
    "data": sorted(list(pd.date_range("1/5/2014", periods = 2)) * 2)})
print("Colunas antes da alteração: ", df_cambio.columns)
df_cambio.columns = ['moeda', 'cotacao', 'data']
print("Colunas depois da alteração: ", df_cambio.columns)
```

Resposta:

```
>>>  Colunas antes da alteração:  Index(['moeda', 'price',
'data'], dtype='object')
>>>  Colunas depois da alteração:  Index(['moeda', 'cotacao',
'data'], dtype='object')
```



2. Como selecionar uma coluna?

Uma forma de selecionar uma coluna de um DataFrame de várias colunas é:

DataFrame["Nome_coluna"]

ou

DataFrame.Nome_coluna

Ao selecionar uma coluna dessa forma, o Python retorna uma Series.

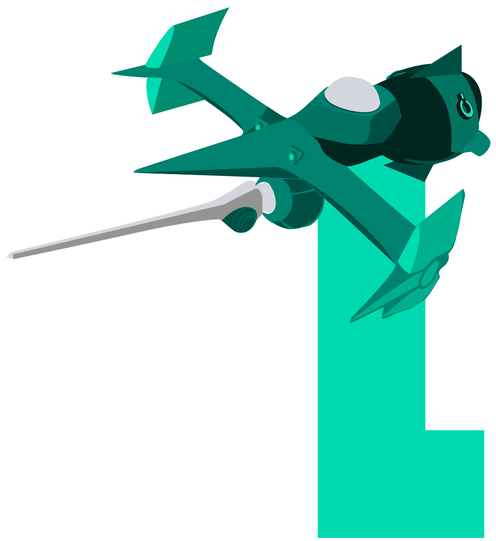
Exemplo:

```python
import pandas as pd

dicionario_dados = {
    "empresas": ["Wege", "Vale", "Petrobras"],
    "price": [20, 30, 40],
    "volume": [1000, 4000, 7500]}
df = pd.DataFrame(dicionario_dados)
coluna1 = df["empresas"]
coluna2 = df.empresas

print(coluna1)
print(coluna2)
```

e

Resposta:

```
>>> 0         Wege
    1         Vale
    2    Petrobras
    Name: empresas, dtype: object

>>> 0         Wege
    1         Vale
    2    Petrobras
    Name: empresas, dtype: object
```



3. Como selecionar uma coluna por condição?

Esse método é muito parecido com o de selecionar colunas, a diferença é que agora vamos definir uma condição. O que estamos fazendo, de fato, é filtrar as linhas no DataFrame existente e retornando um novo DataFrame, por isso é importante que uma nova variável seja atribuída.

Resposta:

3.1 selecionando itens menores que:

Exemplo:

```python
import pandas as pd
dicionario_dados = {
    "empresas": ["Itau", "Wege", "Vale", "Petrobras"],
    "price": [23, 20, 30, 40],
    "volume": [2000, 1000, 4000, 7500]}

df_original = pd.DataFrame(dicionario_dados)
df_menor = df_original[df_original["price"] > 25]
print(df_menor)
```

Resposta:

```
>>>       empresas  price  volume
      2        Vale     30    4000
      3   Petrobras     40    7500
```

Para selecionar números maiores que alguma condição, é só fazer o mesmo método, fazendo a troca somente do “<” para “>”.

3.2 selecionando itens iguais a:

Para selecionar números diferentes a alguma condição, é só fazer o mesmo método, fazendo a troca somente do "==" para "!=".

Exemplo:

```
import pandas as pd

dicionario_dados = {
    "empresas": ["Itau", "Wege", "Vale", "Petrobras"],
    "price": [23, 20, 30, 40],
    "volume": [2000, 1000, 4000, 7500]}
df_original = pd.DataFrame(dicionario_dados)
df_igual = df_original[df_original["empresas"] == "Itau"]
print(df_igual)
```

Resposta:

```
>>>        empresas  price  volume
        0      Itau     23    2000
```

4. Como selecionar uma coluna por mais de uma condição?

Esse método é muito parecido com o de selecionar colunas por condição, a diferença é que vamos definir mais de uma condição com o auxílio do " & " e " | ". O que estamos fazendo, de fato, é filtrar as linhas no DataFrame existente, de acordo com as condições, e retornando um novo DataFrame, por isso é importante que uma nova variável seja atribuída.

4.1 selecionando itens com o &:

Com esse método, você selecionará apenas se as duas condições forem verdadeiras.

Exemplo:

```
import pandas as pd

df_cambio = pd.DataFrame({
    "moeda": ['dolar/real', 'libra/real'] * 2,
    "cotacao": [5.24, 4.15, 5.26, 4.23],
    "data": sorted(list(pd.date_range("1/5/2014", periods = 2)) * 2)})
#dolar e maior que 5.25
cambio_dolar_e_maior = df_cambio[(df_cambio['moeda'] == 'dolar/real')
& (df_cambio['cotacao'] > 5.25)]
print(cambio_dolar_e_maior)
```

Resposta:

```
>>>        moeda    cotacao         data
2  dolar/real     5.26   2014-01-06
```

4.2 selecionando itens com o |:

Com esse método, você selecionará se uma das duas condições forem verdadeiras.

Exemplo:

```
import pandas as pd

df_cambio = pd.DataFrame({
    "moeda": ['dolar/real', 'libra/real'] * 2,
    "cotacao": [5.24, 4.15, 5.26, 4.23],
    "data": sorted(list(pd.date_range("1/5/2014", periods = 2)) * 2)})
#libra ou maior que 5.25
cambio_libra_ou_maior = df_cambio[(df_cambio['moeda']

    == 'libra/real') | (df_cambio['cotacao'] > 5.25)]

print(cambio_libra_ou_maior)
```

Resposta:

```
>>>        moeda  cotacao       data
1  libra/real     4.15 2014-01-05
2  dolar/real     5.26 2014-01-06
3  libra/real     4.23 2014-01-06
```

5. Como selecionar um index

Esse método é muito parecido com o de selecionar colunas por condição. Só que utilizaremos o método index, que retornará a coluna index inteira.

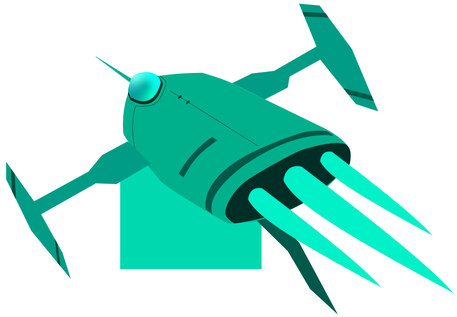
Exemplo:

```
import pandas as pd
cambio = {'moeda': ['dolar', 'dolar', 'libra', 'libra'],
          'cotacao': [5.24, 5.26, 5.89, 5.89]}
df_cambio = pd.DataFrame(
    cambio, index=['2022-10-25', '2022-10-26', '2022-10-25', '2022-10-26'])
df_25_out = df_cambio[(df_cambio.index == "2022-10-25")]
print(df_25_out)
```

Resposta:

```
>>>
                    moeda  cotacao
2022-10-25  dolar     5.24
2022-10-25  libra     5.89
```

Mundo 5



Neste mundo veremos algumas propriedades de seleção de dados dentro dos DataFrames.

1. pandas.DataFrame.loc

Como dito anteriormente, esse método serve para seleção de dados. Sua maior utilização é com o **INDEX**. Sua sintaxe é:

df.loc[<chave_index> , <colunas>]

Esse método é utilizado em caso de datas ou nomes. Para o número das posições o melhor método é o **iloc** (veremos a seguir).

Utilizamos " : " quando queremos escolher um intervalo, seja de linhas ou de colunas. Utilizaremos o seguinte DataFrame para exemplificar o método loc:

```python
import pandas as pd

df_cambio = pd.DataFrame({
"moeda": ['dolar/real', 'libra/real'] * 4,
"price": [5.24, 4.15, 5.26, 4.23, 5.55, 4.02, 5.76, 3.98],
"derivativo": list(["DOLFT3", "LIBFT3"]) * 4},
index = sorted(list(pd.date_range("1/5/2014",periods=4))*2))
```

1.1 selecionando uma chave:

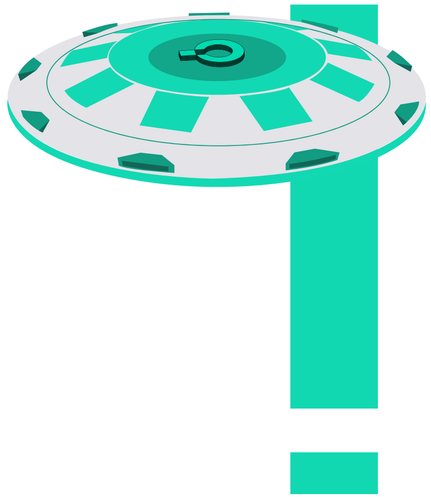
Com esse método, você pode selecionar uma chave para procurar pelo DataFrame. Repare que como não foi especificado, todas as colunas serão selecionadas.

Exemplo:

```
#um único dia por chave
print(df_cambio.loc['2014-01-05'])
```

Resposta:

```
moeda  price derivativo
2014-01-05  dolar/real   5.24     DOLFT3
2014-01-05  libra/real   4.15     LIBFT3
```



1.2 selecionando um intervalo:

Com esse método, você pode selecionar um intervalo para procurar pelo DataFrame. Repare que como não foi especificado, todas as colunas serão selecionadas.

obs: Esse método só pode ser feito com dates e integers.

Exemplo:

```
#algum período
df_cambio.loc['2014-01-05':'2014-01-07']
```

Resposta:

```
moeda  price derivativo
2014-01-05  dolar/real   5.24     DOLFT3
2014-01-05  libra/real   4.15     LIBFT3
2014-01-06  dolar/real   5.26     DOLFT3
2014-01-06  libra/real   4.23     LIBFT3
2014-01-07  dolar/real   5.55     DOLFT3
2014-01-07  libra/real   4.02     LIBFT3
```

1.3 selecionando um intervalo e limitando a coluna:

Com esse método, você pode selecionar um intervalo para procurar pelo DataFrame. Mas dessa vez repare que, a coluna foi especificada, logo ele só retornará os valores dessa coluna.

Repare que ao selecionar uma coluna o retorno será uma Series e não um DataFrame

Exemplo:

```
#um periodo sobre a informação preço
df_cambio.loc['2014-01-05':'2014-01-07', 'price']
```

Resposta:

```
2014-01-05    5.24
2014-01-05    4.15
2014-01-06    5.26
2014-01-06    4.23
2014-01-07    5.55
2014-01-07    4.02
Name: price, dtype: float64
```

1.4 selecionando uma chave e limitando o intervalo da coluna:

Com esse método, você pode selecionar uma chave para procurar pelo DataFrame. Mas dessa vez repare que, como o intervalo da coluna foi especificado, logo ele só retornará os valores deste intervalo.

Exemplo:

```
#um dia sobre a informação preço e moeda
df_cambio.loc['2014-01-05', 'moeda':'price']
```

Resposta:

```
              moeda  price
2014-01-05  dolar/real   5.24
2014-01-05  libra/real   4.15
```

1.5 Substituir dados com loc:

Com esse método, você pode selecionar uma chave específica para substituir.

Exemplos:

```
#se um ativo mudar o código de negociação?
df_cambio.loc[df_cambio['derivativo'] == "DOLFT3", 'derivativo'] = "DOLFT4"
print(df_cambio)
```

Resposta:

```
           moeda  price derivativo
2014-01-05  dolar/real   5.24     DOLFT4
2014-01-05  libra/real   4.15     LIBFT3
2014-01-06  dolar/real   5.26     DOLFT4
2014-01-06  libra/real   4.23     LIBFT3
2014-01-07  dolar/real   5.55     DOLFT4
2014-01-07  libra/real   4.02     LIBFT3
2014-01-08  dolar/real   5.76     DOLFT4
2014-01-08  libra/real   3.98     LIBFT3
```

2. pandas.DataFrame.at

Esse método serve para caso você necessite de um único item na tabela (no caso de itens repetidos, ele retornará uma Series). Podemos fazer uma comparação como se fosse o "proc-v" do Excel. Assim como o **loc**, esse método só pode ser utilizado em conjunto com o **index**. O at nada mais é do que um loc otimizado para puxar um dado específico da tabela. Sua sintaxe é:

df.at[<chave_index> , <coluna>]

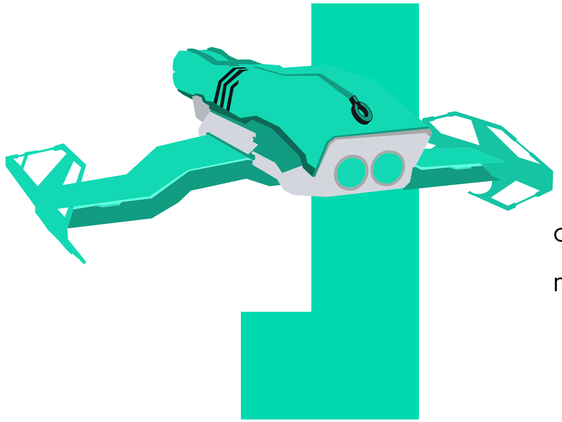
Alguns exemplos da utilização para o seguinte DataFrame:

Exemplo:

```
import pandas as pd

df_cambio = pd.DataFrame({
"moeda": ['dolar/real', 'libra/real'] * 4,
"price": [5.24, 4.15, 5.26, 4.23, 5.55, 4.02, 5.76, 3.98],
"derivativo": list(["DOLFT3", "LIBFT3"]) * 4},

 index = sorted(list(pd.date_range("1/5/2014",periods=4))*2))

print(df_cambio.at['2014-01-05', "price"])
```

Resposta:

```
2014-01-05    5.24
2014-01-05    4.15
Name: price, dtype: float64
```



3. pandas.DataFrame.iloc

Esse método é utilizado para selecionar itens através da posição. Não precisa ser um index para acessar as informações. Sua sintaxe é:

df.iloc[posição_da_linha , posição_da_coluna]

Aceita somente integers

Utilizamos " : " quando queremos escolher um intervalo, seja de linhas ou de colunas. Utilizaremos o seguinte DataFrame para exemplificar o método loc:

Exemplo:

```
import pandas as pd

df_cambio = pd.DataFrame({
"moeda": ['dolar/real', 'libra/real'] * 4,
"price": [5.24, 4.15, 5.26, 4.23, 5.55, 4.02, 5.76, 3.98],
"derivativo": list(["DOLFT3", "LIBFT3"]) * 4},

  index = sorted(list(pd.date_range("1/5/2014",periods=4))*2))
```

3.1 selecionando a posição de uma linha:

Com esse método, você pode selecionar a posição de uma linha no DataFrame.

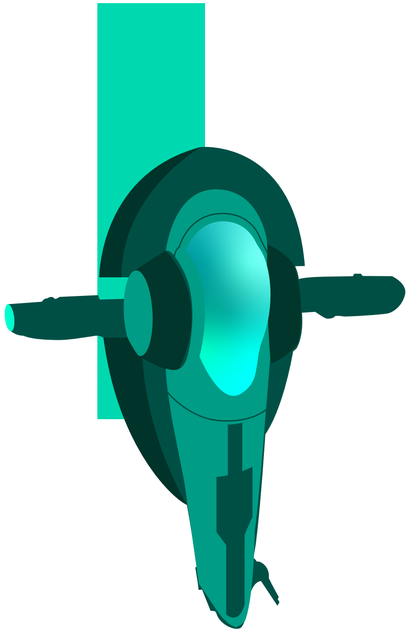
Repare que o retorno dele será uma Series.

Exemplo:

```
#primeira linha
print(df_cambio.iloc[0])
```

Resposta:

```
moeda          dolar/real
price                5.24
derivativo         DOLFT3
Name: 0, dtype: object
```



3.2 selecionando intervalo de uma linha:

Com esse método, você pode selecionar o intervalo de linhas no DataFrame.

Exemplo:

```
#intervalo de linhas
print(df_cambio.iloc[0:3])
```

Resposta:

```
   moeda  price derivativo
0  dolar/real   5.24     DOLFT3
1  libra/real   4.15     LIBFT3
2  dolar/real   5.26     DOLFT3
```

3.3 selecionando uma coluna

Com esse método, você pode selecionar uma coluna inteira.

Repare que o retorno dele será uma Series.

Exemplo:

```
#primeira coluna
print(df_cambio.iloc[:,0])
```

Resposta:

```
0    dolar/real
1    libra/real
2    dolar/real
3    libra/real
4    dolar/real
5    libra/real
6    dolar/real
7    libra/real
Name: moeda, dtype: object
```

3.4 selecionando apenas um item

Neste caso, acessaremos apenas um item. Utilizando a posição da coluna e da linha, método parecido com a função “PROC-V” do Excel.

Por estar considerando a posição do objeto, retornará apenas um item.

Exemplo:

```
#primeiro preço do cambio
print(df_cambio.iloc[0, 1])
```

Resposta:

```
5.24
```

3.5 mudando o valor do item

Neste caso, estamos atribuindo as posições de um item ao valor que desejamos.

Exemplo:

```
#mudando o primeiro valor
df_cambio.iloc[0] = 99
print(df_cambio.iloc[0])
```

Resposta:

```
99.0
```

4. pandas.DataFrame.iat

Esse método é utilizado para selecionar apenas um único item através das posições. Não precisa ser um index para acessar as informações. O método iat é uma otimização do iloc para puxar um único item da tabela. Sua sintaxe é:

df.iat[posição_da_linha , posição_da_coluna]

4.1 selecionando apenas um item

Neste caso, acessaremos apenas um item. Utilizando a posição da coluna e da linha, método parecido com a função "PROC-V" do Excel.

Por estar considerando a posição do objeto, retornará apenas um item.

Exemplo:

```
#primeiro preço do cambio com iat
print(df_cambio.iat[0, 1])
```

Resposta:

```
5.24
```

4.2 mudando o valor do item

Neste caso, estamos atribuindo as posições de um item ao valor que desejamos.

Exemplo:

```
#mudando o valor
df_cambio.iat[0, 1] = 99
print(df_cambio.iat[0, 1])
```

Resposta:

```
99.0
```

Mundo 6

Neste mundo abordaremos algumas propriedades e operações geométricas das Series e DataFrames.

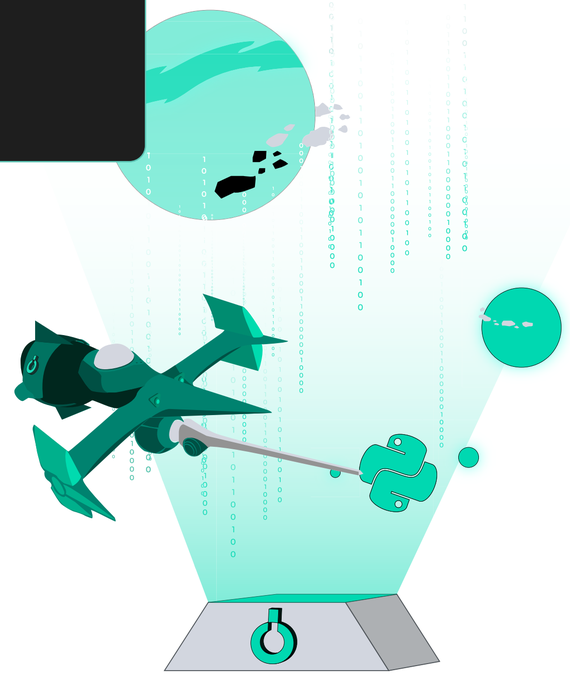
1. Gerador de Series

Para facilitar a compreensão e dar fluidez ao PDF. Foi criada uma função que gera números aleatórios, com uma volatilidade pré-determinada, para um certo intervalo de datas. Retorna uma Series, onde a data é o index.

O objetivo é criar uma Series, para poder exemplificar com maior clareza as propriedades da biblioteca pandas.

1.1 Função geradora de Series

```
import pandas as pd
import numpy as np

def gerador_serie_historica(valor_inicial, volatilidade, periodos, dia_inicial, frequencia = "M"):

    vetor = [valor_inicial]

    for i in range(periodos - 1):

            preco = vetor[i] * (1 + np.random.normal(0, volatilidade))

            vetor.append(preco)


    serie = pd.Series(vetor, index = pd.date_range
(dia_inicial,periods = periodos, freq = frequencia))

    return serie
```



2. Operações aritméticas

As operações aritméticas do pandas são as mesmas que vieram nativas do Python. Relembre algumas delas:

+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Divisão Inteira
**	Exponenciação

As operações aritméticas funcionam em conjunto com o index. Na existência de um index, as operações só irão ocorrer quando existir elementos em **comum** no index das Series, ou DataFrames, que estão sendo utilizadas na operação. Para os itens que não estão em comum, ele retornará um “NaN” no valor.

Caso não haja um index definido, ele fará a operação de acordo com a posição das linhas.

2.1 Operação de multiplicação

No exemplo abaixo, geramos Series com valores das datas diferentes propositalmente. Repare que, na resposta da multiplicação das Series, ele só retornou valores para as datas que ele achou em comum, com as duas Series.

Exemplo:

```
serie_euro = gerador_serie_historica(5, 0.01, 5, "2022-01-01")
serie_cotacoes = gerador_serie_historica(20, 0.05, 5, "2022-03-01")
final = serie_euro * serie_cotacoes

print(final)
```

Resposta:

```
2022-01-31           NaN
2022-02-28           NaN
2022-03-31     97.185975
2022-04-30     99.200609
2022-05-31    103.573968
2022-06-30           NaN
2022-07-31           NaN
Freq: M, dtype: float64
```

2.2 Operação de divisão

No exemplo abaixo, geramos Series com valores das datas diferentes propositalmente. Repare que, na resposta da divisão das Series, ele só retornou valores para as datas que ele achou em comum, com as duas Series.

Exemplo:

```
serie_euro = gerador_serie_historica(5, 0.01, 5, "2022-01-01")
serie_cotacoes = gerador_serie_historica(20, 0.05, 5, "2022-03-01")
final = serie_euro / serie_cotacoes

print(final)
```

Resposta:

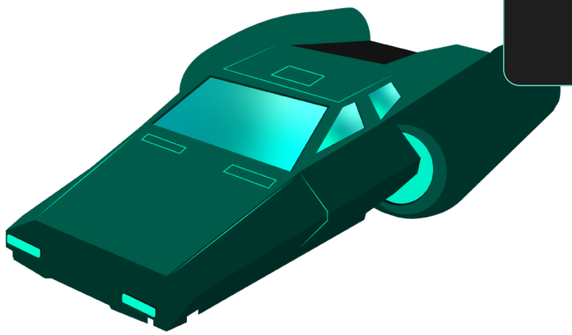
```
2022-01-31         NaN
2022-02-28         NaN
2022-03-31    0.256063
2022-04-30    0.262249
2022-05-31    0.252093
2022-06-30         NaN
2022-07-31         NaN
Freq: M, dtype: float64
```

2.3 Operação de adição

No exemplo abaixo, geramos Series com valores das datas diferentes propositalmente. Repare que, na resposta da soma das Series, ele só retornou valores para as datas que ele achou em comum, com as duas Series.

Exemplo:

```
serie_euro = gerador_serie_historica(5, 0.01, 5, "2022-01-01")
serie_cotacoes = gerador_serie_historica(20, 0.05, 5, "2022-03-01")
final = serie_euro + serie_cotacoes

print(final)
```



Resposta:

```
2022-01-31          NaN
2022-02-28          NaN
2022-03-31    24.962338
2022-04-30    23.227792
2022-05-31    22.975494
2022-06-30          NaN
2022-07-31          NaN
Freq: M, dtype: float64
```

2.4 Operação de subtração

No exemplo abaixo, geramos Series com valores das datas diferentes propositalmente. Repare que, na resposta da diferença das Series, ele só retornou valores para as datas que ele achou em comum, com as duas Series.

Exemplo:

```
serie_euro = gerador_serie_historica(5, 0.01, 5, "2022-01-01")
serie_cotacoes = gerador_serie_historica(20, 0.05, 5, "2022-03-01")
final = serie_euro - serie_cotacoes

print(final)
```

Resposta:

```
2022-01-31          NaN
2022-02-28          NaN
2022-03-31   -14.994343
2022-04-30   -15.584874
2022-05-31   -16.790100
2022-06-30          NaN
2022-07-31          NaN
Freq: M, dtype: float64
```

2.5 Operação de exponenciação

No exemplo abaixo, geramos Series com valores das datas diferentes propositalmente. Repare que, na resposta da exponenciação das Series, ele só retornou valores para as datas que ele achou em comum, com as duas Series.

Exemplo:

```
serie_euro = gerador_serie_historica(5, 0.01, 5, "2022-01-01")
serie_cotacoes = gerador_serie_historica(20, 0.05, 5, "2022-03-01")
final = serie_euro - serie_cotacoes

print(final)
```

Resposta:

```
2022-01-31          NaN
2022-02-28          NaN
2022-03-31   -14.994343
2022-04-30   -15.584874
2022-05-31   -16.790100
2022-06-30          NaN
2022-07-31          NaN
Freq: M, dtype: float64
```

3. Operações aritméticas em colunas

Para operações entre valores de um mesmo DataFrame utilizamos alguns métodos criados pelo pandas. A seguir os principais.

.sum()	Soma
.cumsum()	Acumular Soma
.max()	Valor Máximo
.min()	Valor Mínimo

Para exemplificação a seguir, utilizaremos o DataFrame a seguir, gerado pela função que criamos.

Exemplo:

```
serie_lucro_trimestral = gerador_serie_historica(20000, 2, 4, "2022-01-01", frequencia = "Q")
```

Resposta:

```
2022-03-31    20000.000000
2022-06-30   -12227.802251
2022-09-30   -26138.419417
2022-12-31   -83428.706732
Freq: Q-DEC, dtype: float64
```

3.1 pandas.DataFrame.sum (axis = 0,min_count = 0, skipna = True)

Parâmetros:

Se não definir nenhum parâmetro, o método retornará uma Series, com o nome das colunas e o valor da soma dela.

axis:

Esse parâmetro vai definir em qual direção ocorrerá a soma dos itens, se será verticalmente ou horizontalmente. Por padrão, o pandas define “axis=0”, ou seja, significa que a soma será verticalmente, de cima para baixo.

Esse parâmetro **é opcional**. Pode ser 0, que significa uma soma no sentido vertical, de cima para baixo. Ou pode ser 1, que significa uma soma no sentido horizontal, da esquerda para direita.

min_count:

Esse parâmetro vai definir a quantidade mínima de itens dentro do intervalo para ocorrer a soma. Por padrão, o pandas define “min_count=0”, ou seja, não está definido um valor mínimo para ocorrer a soma.

Esse parâmetro **é opcional** e recebe a quantidade mínima como um integer.

skipna:

Esse parâmetro vai definir se os valores NaN serão ignorados ou não. Por padrão, o pandas define “skipna=True”, ou seja, os valores “NaN” serão ignorados. Caso o “skipna=False”, e um valor NaN seja encontrado, ele retornará o valor NaN.

Esse parâmetro é opcional e recebe um booleano: True ou False.

No exemplo abaixo, geramos uma Series com a função criada. Após isso, soma-se todos os valores da coluna.

Exemplo:

```
lucro_anual = serie_lucro_trimestral.sum()
print(lucro_anual)
```

Resposta:

```
>>>  22192.587795485317
```

3.2 pandas.DataFrame.cumsum (axis = 0,skipna = True)

Parâmetros:

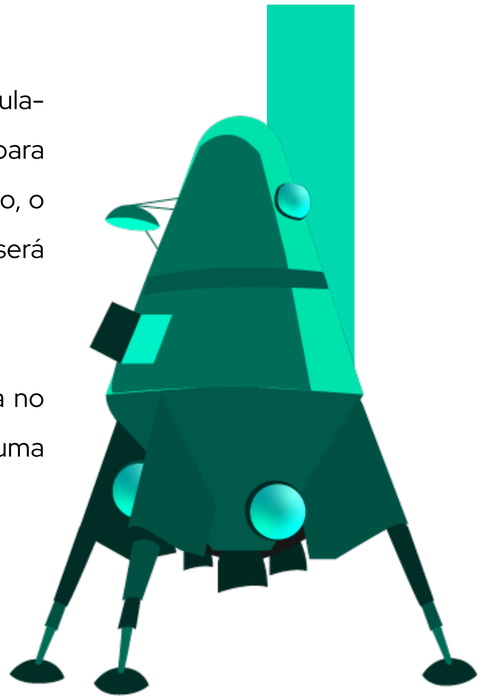
Se não definir nenhum parâmetro, o método retornará o DataFrame, com a soma acumulada por posição.

axis:

Esse parâmetro vai definir em qual direção ocorrerá a soma acumulada dos itens, se será a soma acumulada no sentido vertical, de cima para baixo, ou no sentido horizontal, da esquerda para direita. Por padrão, o pandas define "axis=0", ou seja, significa que a soma acumulada será feita no sentido vertical.

Esse parâmetro **é opcional**. Pode ser 0, que significa uma soma no sentido vertical, de cima para baixo. Ou pode ser 1, que significa uma soma horizontal, da esquerda para direita.

skipna:



Esse parâmetro vai definir se os valores NaN serão ignorados ou não. Por padrão, o pandas define "skipna=True", ou seja, os valores "NaN" serão ignorados. Caso o "skipna=False", e um valor NaN seja encontrado, ele retornará o valor NaN.

Esse parâmetro **é opcional** e recebe um booleano: **True** ou **False**.

No exemplo abaixo, geramos uma Series com a função criada. Após isso, retorna a soma de cada linha com a anterior, fazendo assim a soma acumulada.

Exemplo:

```
crescimento_lucros  = serie_lucro_trimestral.cumsum()
print(crescimento_lucros)
```

Resposta:

```
2022-03-31    20000.000000
2022-06-30    19653.486641
2022-09-30    18839.469557
2022-12-31    16294.924173
Freq: Q-DEC, dtype: float64
```

3.3 pandas.DataFrame.max (axis = 0, skipna = True, numeric_only = True)

Parâmetros:

Se não definir nenhum parâmetro, o método retornará o DataFrame, com o valor máximo com cada coluna.

axis:

Esse parâmetro vai definir em qual direção ocorrerá a análise do valor máximo, se será no sentido horizontal, da esquerda para direita, ou no vertical, de cima para baixo. Por padrão, o pandas define "axis=0", ou seja, significa que será o valor máximo no sentido vertical.

Esse parâmetro **é opcional**. Pode ser 0, que significa o valor máximo analisado verticalmente, de cima para baixo. Ou pode ser 1, que significa o valor máximo analisado horizontalmente, da esquerda para direita.

skipna:

Esse parâmetro vai definir se os valores NaN serão ignorados ou não. Por padrão, o pandas define "skipna=True", ou seja, os valores "NaN" serão ignorados. Caso o "skipna=False", e um valor NaN seja encontrado, ele retornará o valor NaN.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

numeric_only:

Esse parâmetro vai definir que em caso de qualquer outro formato, que não seja integer, ele retornará um erro ou ignorará aquela coluna. Por padrão, o pandas define "numeric_only = True", ou seja, se houver uma string na coluna, ele apenas ignora essa coluna. Caso "numeric_only = False" e tenha uma string na coluna, ele retornará um erro.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

No exemplo abaixo, geramos uma Series com a função criada. Após isso, retoma-se o maior valor da coluna.

Exemplo:

```
lucro_max = serie_lucro_trimestral.max()
print(lucro_max)
```

Resposta:

```
>>>  923.3825678305033
```

3.4 pandas.DataFrame.min (axis = 0, skipna = True, numeric_only = True)

Parâmetros:

* Se não definir nenhum parâmetro, o método retornará o DataFrame, com o valor mínimo com cada coluna.

axis:

Esse parâmetro vai definir em qual direção ocorrerá a análise do valor mínimo, se será no sentido horizontal, da esquerda para direita, ou no vertical, de cima para baixo. Por padrão, o pandas define “axis=0”, ou seja, significa que será o valor mínimo no sentido vertical.

Esse parâmetro **é opcional**. Pode ser 0, que significa o valor mínimo analisado verticalmente, de cima para baixo. Ou pode ser 1, que significa o valor máximo analisado horizontalmente, da esquerda para direita.

skipna:

Esse parâmetro vai definir se os valores NaN serão ignorados ou não. Por padrão, o pandas define “skipna=True”, ou seja, os valores “NaN” serão ignorados. Caso o “skipna=False”, e um valor NaN seja encontrado, ele retornará o valor NaN.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

numeric_only:

Esse parâmetro vai definir que em caso de qualquer outro formato, que não seja integer, ele retornará um erro, ou ignorará aquela coluna. Por padrão, o pandas define “numeric_only = True”, ou seja, se houver uma string na coluna, ele apenas ignora essa coluna. Caso “numeric_only = False” e tenha uma string na coluna, ele retornará um erro.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

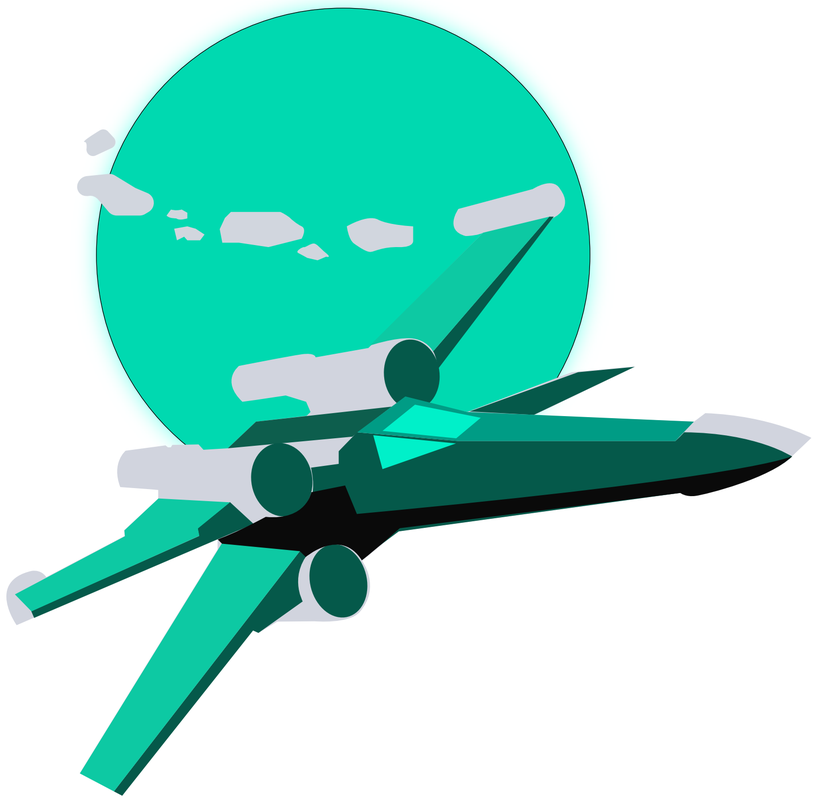
No exemplo abaixo, geramos uma Series com a função criada. Após isso, retoma-se o menor valor da coluna. Neste mundo abordaremos algumas formas de ordenação de dados.

Exemplo:

```
lucro_min = serie_lucro_trimestral.min()
print(lucro_min)
```

Resposta:

```
>>>  923.3825678305033
```



Mundo 7

Neste mundo abordaremos algumas formas de ordenação de dados.

1. pandas.DataFrame.sort_index(axis = 0, level = None, ascending = True, inplace = False, kind = “quicksort”, na_position = “last”, sort_remaining = True, ignore_index = False, key = None)

Esse método retorna um novo DataFrame com o index organizado.

Parâmetros:

Se não definir nenhum parâmetro, ele organizará o index pela ordem das linhas.

axis:

Esse parâmetro vai definir em qual direção ocorrerá a organização, se será no sentido horizontal, da esquerda para direita, ou no vertical,de cima para baixo. Por padrão, o pandas define “axis=0”, ou seja, significa que será organizado verticalmente.

Esse parâmetro **é opcional**. Pode ser 0, que significa que a organização ocorrerá no sentido vertical, de cima para baixo. Ou pode ser 1, que significa que a organização ocorrerá horizontalmente, da esquerda para direita.

level:

Esse parâmetro é utilizado em DataFrames com colunas multi index, ele vai definir qual index será organizado.

Esse parâmetro **é opcional**. Pode ser o nome de um index ou uma lista com nome dos index, em string. Pode ser também a posição de um index(começando no 0) ou uma lista com as posições dos index, em integer.

ascending:

Esse parâmetro é utilizado para definir a ordem que o DataFrame é organizado. Por padrão, o pandas define "ascending = True", ou seja, ele organizará em ordem crescente. Caso "ascending = False", ele organizaria por ordem decrescente.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

inplace:

Esse parâmetro vai definir se as modificações serão feitas diretamente no DataFrame ou não. Esse parâmetro pode ser interpretado por uma forma de salvar as modificações diretamente no DataFrame. Por padrão, o pandas considera "inplace=False", ou seja, ele não salvará as modificações caso você não atribua a um objeto.

Esse parâmetro **é opcional** e recebe um booleano: **True** ou **False**.

kind:

Esse conceito é mais complexo e menos importante. Mas em resumo, está relacionado ao tipo de algoritmo usado na organização.

na_position:

Este parâmetro define se os valores “NaN” ficarão nos primeiros lugares ou nos últimos. Por padrão, o pandas define “na_position = last, ou seja, os valores “NaN” ficarão nos últimos lugares.

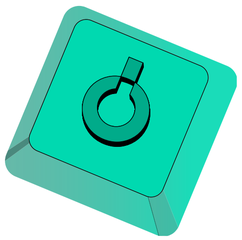
Esse parâmetro **é opcional** e recebe apenas duas strings: “last” ou “first”

sort_remaining:

Este parâmetro é utilizado para DataFrames multiniveis. Ele especifica se deve organizar por outros níveis, após já ter organizado para um nível específico. Por padrão, o pandas considera “sort_remaing = True”, ou seja, ele organizará após já ter organizado para o nível especificado.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

ignore_index:



Este parâmetro define se o índex existente, um index de datas por exemplo, vai voltar ao padrão, onde é especificado por posição (0,1,2,3). Por padrão, o pandas define “ignore_index = False”, ou seja, ele manterá o index como está.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

key:

Este parâmetro é parecido com o key do método sorted(). Ele define uma função a ser aplicada, antes da organização do index. Porém, você não verá essa modificação.

Esse parâmetro **é opcional** e recebe uma função que pode ser nativa do Python ou criada de acordo com a necessidade.

No exemplo abaixo, geramos um DataFrame e organizamos as colunas deles em ordem alfabética-invertida. Além disso, as modificações do DataFrame foram salvas por causa do "inplace = True".

Exemplo:

```
import pandas as pd
import numpy as np

dicionario = {
            "nomes": ["Quero Quero", "Alpargatas", "Alpargatas", "Magazine Luiza"],
            "preco_sobre_lucro": [12, 6, 12, 100],
            "volume": [5000, 1000, 4000, 7000]}

empresas = pd.DataFrame(dicionario, index = ["LJQQ3", "ALPA3", "ALPA4", "MGLU3"])
empresas.sort_index(axis = 1, ascending = False, inplace=True)
print(empresas)
```

Resposta:

```
       volume  preco_sobre_lucro           nomes
LJQQ3    5000                 12     Quero Quero
ALPA3    1000                  6      Alpargatas
ALPA4    4000                 12      Alpargatas
MGLU3    7000                100  Magazine Luiza
```

2. pandas.DataFrame.sort_values(by, axis = 0, ascending = True, inplace = False, kind = "quicksort", na_position = "last", ignore_index = False, key = None)

Esse método retorna um novo DataFrame com o index organizado.

Parâmetros:

O parâmetro "by" é o único obrigatório.

by:

Esse parâmetro vai definir qual a coluna que será ordenada.

Esse parâmetro **é obrigatório**. Pode receber o nome da coluna no formato string, ou pode receber uma lista com os nomes das colunas no formato string.

axis:

Esse parâmetro vai definir em qual direção ocorrerá a organização, se será no sentido horizontal, da esquerda para direita, ou no vertical, de cima para baixo. Por padrão, o pandas define “axis=0”, ou seja, significa que será organizado verticalmente.

Para ser comparado qual item vai vir primeiro, eles devem ter o mesmo formato (string com string e integer com integer)

Esse parâmetro **é opcional**. Pode ser 0, que significa que a organização ocorrerá no sentido vertical(organizando as linhas do index). Ou pode ser 1, que significa que a organização ocorrerá horizontalmente (organizando os rótulos(nome) das colunas)

ascending:

Esse parâmetro é utilizado para definir a ordem que o DataFrame é organizado. Por padrão, o pandas define “ascending = True”, ou seja, ele organizará em ordem crescente. Caso “ascending = False”, ele organizaria por ordem decrescente.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

inplace:

Esse parâmetro vai definir se as modificações serão feitas diretamente no DataFrame ou não. Esse parâmetro pode ser interpretado por: uma forma de salvar as modificações diretamente no DataFrame. Por padrão, o pandas considera “inplace=False”, ou seja, ele não salvará as modificações caso você não atribua a um objeto.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

kind:

Esse conceito é mais complexo e menos importante. Mas em resumo, está relacionado ao tipo de algoritmo usado na organização.

na_position:

Este parâmetro define se os valores “NaN” ficarão nos primeiros lugares ou nos últimos. Por padrão, o pandas define “na_position = last, ou seja, os valores “NaN” ficarão nos últimos lugares.

Esse parâmetro **é opcional** e recebe apenas duas strings: “last” ou “first”

ignore_index:

Este parâmetro define se o índex existente, um index de datas por exemplo, vai voltar ao padrão, onde é especificado por posição (0,1,2,3). Por padrão, o pandas define “ignore_index = False”, ou seja, ele manterá o index como está.

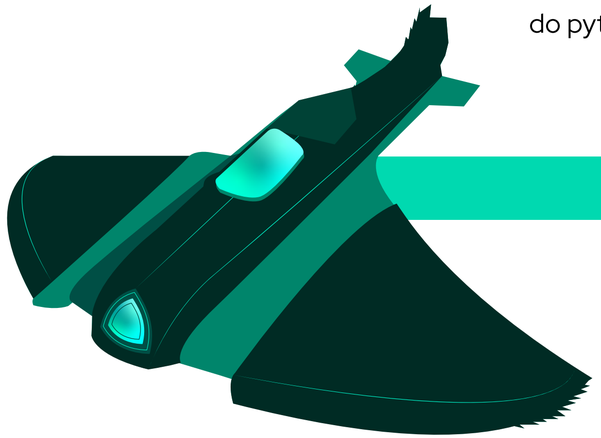
Esse parâmetro **é opcional** e recebe um booleano: True ou False.

key:

Esse parâmetro recebe uma função que transforma cada elemento antes de ordenar, pega este valor e retorna um valor que é então usado dentro de ordenação ao invés do valor original.

Exemplo: Se passarmos o parâmetro “key = len()”, estaremos dando como chave uma função que retorna o tamanho dos elementos. Então serão , estes elementos, ordenados pelo tamanho. Em resumo, Isso significa que as strings seriam classificadas com base em seus comprimentos

Esse parâmetro **é opcional** e recebe uma função que pode ser nativa do python ou criada de acordo com a necessidade.



No exemplo abaixo, geramos um DataFrame, que está organizado por posição. Utilizando o método sortt_values(), organizamos esse DataFrame por valor, em ordem decrescente.

Exemplo:

```
import pandas as pd

acoes = {'Acoes': ['Wege', 'Petrobras', 'Vale', 'PetroRio'],
         'Valor': [35.42, 33.67, 72.22, 33.96]}
df_acoes = pd.DataFrame(acoes, index=[1, 2, 3, 4])
print(df_acoes)

df_acoes.sort_values(by='Valor', ascending=False, inplace=True)
print(df_acoes)
```

Resposta antes do sort_values():

```
     Acoes  Valor
1     Wege  35.42
2  Petrobras  33.67
3     Vale  72.22
4   PetroRio  33.96
```

Resposta depois do sort_values():

```
     Acoes  Valor
3     Vale  72.22
1     Wege  35.42
4   PetroRio  33.96
2  Petrobras  33.67
```

3. pandas.DataFrame.rank(axis = 0, method = “average”, numeric_only = False, na_option = “keep”, ascending = True, pct = False)

Esse método retorna classificação dos dados. Por padrão, caso tenha valores iguais, o pandas retornará a classificação média desses valores.

Parâmetros:

Caso não defina nenhum parâmetro, ele retornará todas as colunas com rankings definidos por valores das linhas.

axis:

Esse parâmetro vai definir em qual direção ocorrerá o ranqueamento, se será no sentido horizontal, da esquerda para direita, ou no vertical, de cima para baixo. Por padrão, o pandas define “axis=0”, ou seja, significa que será ranqueado verticalmente.

Esse parâmetro **é opcional**. Pode ser 0, que significa que o ranqueamento ocorrerá no sentido vertical, de cima para baixo. Ou pode ser 1, que significa que o ranqueamento ocorrerá horizontalmente, da esquerda para direita.

method:

Esse parâmetro vai definir métodos de como será feito o desempate do ranqueamento. Por padrão, o pandas define “method = ‘average’”, ou seja, em caso de empate, pega a média dos empatados.

Esse parâmetro **é opcional**. Existem alguns padrões já estabelecidos pelo pandas, que são:

‘average’ => Em caso de empate, pegar a média dos ranking dos empatados.

‘min’ => Em caso de empate, pegar a menor posição (pulando a posição de cima).

‘max’ => Em caso de empate, pegar a maior posição (pulando a posição de baixo).

‘first’ => Em caso de empate, pegar o primeiro que aparecer.

‘dense’ => Em caso de empate, pegar a menor posição (sem pular uma posição).

numeric_only:

Esse parâmetro vai definir se esse método vai permitir valores que não são integers. Por padrão, o pandas considera “numeric_only = True”, ou seja, ele não irá ranquear valores não numéricos.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

na_option:

Esse parâmetro vai definir o que fazer em caso de valores com “NaN”. Por padrão, o pandas considera “na_option = ‘keep’ ”, ou seja, ela mantém os valores “NaN” sem modificar o ranqueamento.

Esse parâmetro **é opcional**. Existem alguns padrões já estabelecidos pelo pandas, que são:

‘keep => Mantém os valores “NaN” sem interferir no ranqueamento.

‘top’ => Mantém os valores “NaN” atribuindo-os às primeiras posições do ranqueamento.

‘bottom’ => Mantém os valores “NaN” atribuindo-os às últimas posições do ranqueamento.

ascending:

Esse parâmetro é utilizado para definir a ordem que as colunas serão ordenadas. Por padrão, o pandas define “ascending = True”, ou seja, ele organizará em ordem crescente. Caso “ascending = False”, ele organizaria por ordem decrescente. Por padrão, o pandas define “ascending = True”, ou seja, ele organizará as colunas por ordem crescente.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

pct:

Esse parâmetro é utilizado para definir a formatação em porcentagem do ranqueamento. A porcentagem é equidistante de todos os valores. Por padrão, o pandas define “pct = False”, ou seja, ele retornará o ranqueamento em números inteiros.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

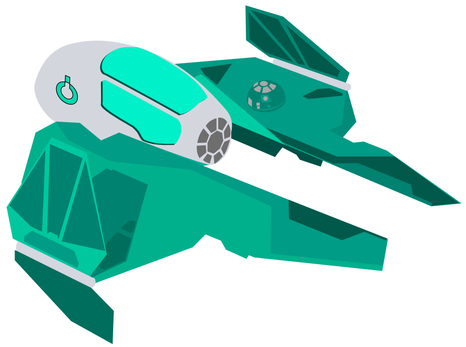
No exemplo abaixo, geramos uma Serie, pelo nosso gerador de Series, e organizamos as linhas, da coluna "PL", em ordem decrescente. Salvamos estas modificações, sem a necessidade de atribuir a um novo DataFrame.

Exemplo:

```
import pandas as pd
import numpy as np

dicionario = {
      "nomes": ["Quero Quero", "Alpargatas", "Alpargatas", "Magazine Luiza", "Vale"],
      "preco_sobre_lucro": [12, 6, None, 100,None],
      "volume": [5000, 1000, 4000, 7000,56]}

empresas = pd.DataFrame(dicionario, index = ["LJQQ3", "ALPA3", "ALPA4", "MGLU3", "VALE3"])
empresas = empresas.rank(method='dense', numeric_only=True, na_option='keep', ascending = False)
print(empresas)
```

Resposta:

```
       preco_sobre_lucro  volume
LJQQ3                2.0     2.0
ALPA3                3.0     4.0
ALPA4                NaN     3.0
MGLU3                1.0     1.0
VALE3                NaN     5.0
```



Mundo 8

Neste mundo abordaremos formas de integrar Excel com a biblioteca pandas.

1. pandas.read_excel(io, sheet_name = 0, names = None, index_col = None, usecols = None, dtype = None, engine = None, skiprows = None, nrows = None, na_values = None, keep_default_na = True, na_filter = True, parse_dates = False, date_parser = None, thousands = None, decimal = ".", comment = None, skipfooter = 0, convert_float = None)

Primeiramente você deve entender o conceito de diretório e caminho do arquivo. Quando trabalhamos com dados dentro do computador, precisamos informar onde está localizado o arquivo. Fazemos isso por meio do caminho do arquivo, uma combinação de textos e endpoints que vão detalhar, para o computador, a localização de um arquivo. Já o diretório é o local final do computador onde o arquivo se encontra.

Quando queremos acessar um arquivo, que está localizado no local onde estamos trabalhando, não precisaremos especificar um caminho, apenas o nome do arquivo. Mas quando estamos trabalhando em um diretório diferente da localização do arquivo, aí sim, precisamos especificar seu caminho.

Esse método aceita os seguintes formatos de arquivos: xls, xlsx, xlsm, xlsb, odf, ods.

Parâmetros:

O parâmetro "io" é o único obrigatório.

io:

Esse parâmetro vai definir onde está localizado o arquivo, através do caminho dele. Vale lembrar que, se o arquivo estiver no mesmo diretório, do arquivo.py, não precisará especificar o caminho, apenas o nome do arquivo a ser lido.

Esse parâmetro **é obrigatório**. Pode receber o caminho, ou nome, do arquivo em string.

sheet_name:

Esse parâmetro vai definir qual aba da planilha que vamos acessar. Por padrão, o pandas define "sheet_name = 0", ou seja, ele usará as informações da primeira aba da planilha.

Esse parâmetro **é opcional**. Recebe o nome das colunas, em string. Pode receber a posição das colunas, em integer. Também pode receber uma lista contendo as posições das colunas e os nomes delas.

names:

Esse parâmetro vai definir o nome das colunas do DataFrame. O tamanho da lista deve conciliar com a quantidade de colunas no DataFrame. Por padrão, o pandas define "names = None", ou seja, ele não irá definir nome para as colunas, utilizará os que estão definidos no Excel.

Esse parâmetro **é opcional**. Recebe o nome das colunas, em string. Também pode receber uma lista contendo os nomes das colunas em string.

index_col:

Esse parâmetro vai definir se vai ser definida uma coluna index. Por padrão, o pandas considera "index_col = None", ou seja, não é definido nenhuma coluna index.

Esse parâmetro **é opcional**. Pode receber a posição da coluna, em integer. Pode receber as posições das colunas, em integer, dentro de uma lista (neste caso irá retornar um DataFrame MultI index). Ou pode receber o nome da coluna, em string.

usecols:

Esse parâmetro vai definir quais colunas serão selecionadas. Por padrão, o pandas define "usecols = None", ou seja, como não foi definido ele passará todas as colunas.

Pode receber também uma função, por exemplo, que só selecione nome das colunas com mais de 10 caracteres:

df = pd.read_csv(„data.csv", usecols = lambda x: len(x) > 10)

Esse parâmetro **é opcional**. Pode receber o nome da coluna a ser passado em string, dentro de uma lista. Ou os nomes das colunas a serem passadas, em string, dentro de uma lista. Pode receber também uma função.

dtype:

Esse parâmetro vai definir o tipo de cada coluna, se são: integers, strings ou floats por exemplo.

Esse parâmetro **é opcional**. Recebe um tipo definido para o DataFrame inteiro, ou um dicionário que vai definir cada coluna como um tipo.

engine:

Esse parâmetro vai definir o tipo de “integração” utilizada para abrir o documento. Por padrão, o pandas define “engine = None”, ou seja, o pandas irá tentar definir uma engine automaticamente. Porém pode ter casos que você precise definir uma engine.

Esse parâmetro **é opcional**. Recebe tipos pré-definidos de ignição, que são:

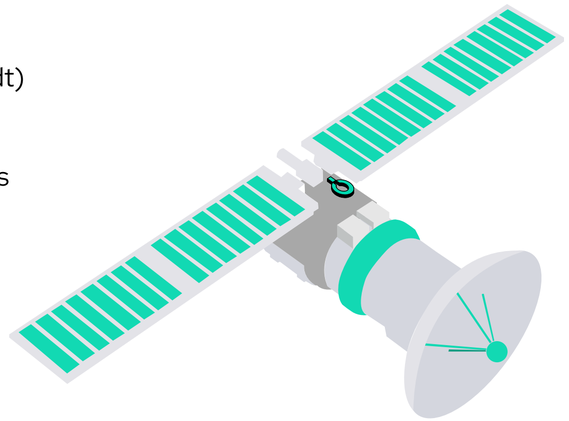
“xlrd” => Utilizado em arquivos Excel antigos

“openpyxl” => Utilizado em arquivos Excel recentes

“odf” => Utilizado para formatos (.odf , .ods , .odt)

“pyxlsb” => Utilizado para arquivos Excel binários

skiprows:



Esse parâmetro vai definir a quantidade de linhas a ser puladas, antes de contabilizar os dados presentes no arquivo. Por padrão, o pandas considera “skiprows = None”, ou seja, nenhuma linha vai ser pulada.

Pode receber também uma função, por exemplo, uma função que pule os números ímpares:

skiprows = lambda x: x%2! = 0

Esse parâmetro **é opcional**. Pode receber: o número das linhas, em integer, dentro de uma lista. Pode receber a quantidade de linhas (começando na posição 0). Ou pode receber uma função que pule linhas de acordo com uma condição.

nrows:

Esse parâmetro vai definir a quantidade de linhas que um DataFrame vai ter. Sua contagem começa da linha 0, então se vc definir “nrows = 5”, serão escolhidas as 5 primeiras linhas do seu DataFrame. Por padrão, o pandas define “nrows = None”, ou seja, ele não define um limite máximo de linha

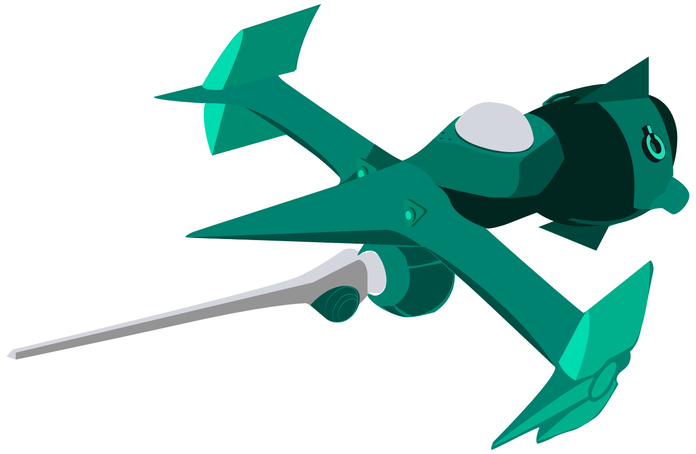
Esse parâmetro **é opcional**. Pode receber apenas a quantidade de linhas máximas de um DataFrame, em interger.

na_values:

Esse parâmetro vai definir valores que serão transformados em vazios (NaN). Muitas vezes, os valores que são definidos como vazios no documento(“nd”,”#NA”,”N/A”), não são reconhecidos pelo Python como tal. Por padrão, o pandas considera “na_values = None”, ou seja, nenhum valor foi definido como vazio.

Esse parâmetro **é opcional**. Pode receber o valor a ser considerado vazio, em string. Ou os valores a serem considerados vazios, em string, dentro de uma lista.

keep_default_na:



Esse parâmetro vai definir se os valores “NaN” serão mantidos no DataFrame ou não. Por padrão, o pandas define “keep_default_na = True”, ou seja, ele manterá os valores “NaN”. Caso seja definido “keep_default_na = False”, ele excluirá o valor “Nan” e deixará um espaço vazio.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

na_filter:

Esse parâmetro vai definir que espaços vazios ou em branco serão representados como valores “NaN”. Por padrão, o pandas define “na_filter = True”, ou seja, em caso de valores em branco, estes serão representados como “NaN”. Caso “na_filter = False”, os valores em branco, continuariam em branco.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

parse_dates:

Esse parâmetro vai definir quais colunas serão definidas como data. Por padrão, o pandas define "parse_dates = None", ou seja, ele não considera nenhuma coluna como data.

obs: Esse parâmetro permite juntar datas presentes em diferentes colunas. Caso o dia, mês e ano estejam em 3 colunas distintas, poderia ser usado a seguinte sintaxe: **parse_dates = [['coluna_dia' , 'coluna_mes' , 'coluna_ano']].**

obs2: Esse parâmetro também permite definir o nome da coluna final da data após coletar dados de colunas diferentes, com a seguinte sintaxe: **parse_dates = {'data_final': 'coluna_dia' , 'coluna_mes': 'coluna_ano'}.**

obs3: Este parâmetro pode receber também um valor booleano e, neste caso, ele definirá o index como data.

Esse parâmetro **é opcional**. Pode receber o nome da coluna em string, a posição da coluna em integer. Ou pode receber uma lista com as posições, em integer, e/ou com os nomes, em string. Pode receber também um booleano: True ou False.

date_parser:

Esse parâmetro é utilizado em conjunto com o "parse_dates", define uma função para converter uma coluna.. Por padrão, o pandas define "date_parsesr = None", ou seja, nenhuma função é definida.

Esse parâmetro **é opcional**. Pode receber uma função lambda com a seguinte sintaxe:

date_parser = lambda x: datetime.strptime(x, „%Y-%d-%m %H:%M:%S")

* Nessa função acima, estamos passando itens que são strings para o formato datetime.

thousands:

Esse parâmetro vai definir o separador **milenar** para números que estão formatados como **texto** no Excel. Os itens do Excel, que já estão formatados como números, vão automaticamente ser formatados como float ou integrer com separador ".".

Esse parâmetro **é opcional**. Pode receber apenas um separador milenar em string.

decimal:

Esse parâmetro vai definir o separador **decimal** para números que estão formatados como **texto** no Excel. Os itens do Excel, que já estão formatados como números, vão automaticamente ser formatados como float ou integrer com separador ".".

Esse parâmetro **é opcional**. Pode receber apenas um separador decimal em string.

comment:

Esse parâmetro vai definir um caractere,uma sequência de caracteres ou uma palavra que será excluída do DataFrame.

Esse parâmetro **é opcional**. Pode receber apenas um caractere,uma sequência de caracteres ou uma palavra como string.

skipfooter:

Esse parâmetro vai definir a quantidade de linhas a serem excluídas do DataFrame, tendo como ponto de partida a última linha do DataFrame.

Esse parâmetro **é opcional**. Pode receber um número, em integer, que determinará a quantidade de linhas.

2. DataFrame.to_excel(Excel_writer, sheet_name = 'Sheet1', na_rep = " ", float_format = None, columns = None, header = True, index = True, index_label = None, startrow = 0, startcol = 0, merge_cells = True, inf_rep = "inf", freeze_panes = None)

Esse método envia um DataFrame, ou Series, para um arquivo Excel. Caso não exista o arquivo Excel, ele criará um.

Esse método aceita os seguintes formatos de arquivos: xls, xlsx, xlsm, xlsb, odf, ods

Parâmetros:

O parâmetro “Excel_writer” é o único obrigatório.

Excel_writer:

Esse parâmetro vai definir o nome do arquivo(xls, xlsx, xlsm, xlsb, odf, ods), no qual o DataFrame vai ser enviado. Note que é nesse parâmetro que o formato do arquivo vai ser definido.

Esse parâmetro **é obrigatório**. Pode receber o nome do arquivo em string.

sheet_name:

Esse parâmetro vai definir para qual aba da planilha vai o DataFrame. Por padrão, o pandas define “sheet_name = ‘Sheet1’ ”, ou seja, ele irá para aba nomeada como “Sheet1”, ou criará uma aba chamada “Sheet1”.

Esse parâmetro **é opcional**. Recebe o nome da aba, em string.

na_rep:

Esse parâmetro vai definir qual será o valor dos ‘NaN’ dentro do excel. Por padrão, o pandas define ‘na_rep = “ “’, ou seja, os valores ‘NaN’ serão exportados para o excel como vazio (sem nada).

Esse parâmetro **é opcional**. Pode receber o valor a ser considerado como ‘NaN’, como string.

float_format:

Esse parâmetro vai definir a quantidade de casas decimais que terão os floats dentro do Excel. Por padrão, o pandas define “float_format = None”, ou seja, será a quantidade de casas decimais definidas no DataFrame.

Esse parâmetro **é opcional**. Recebe a quantidade de casas decimais, em string, no seguinte formato: “%.2f ” (2 casas decimais).

columns:

Esse parâmetro vai definir quais serão as colunas do seu DataFrame que serão exportadas para o Excel, Por padrão, o pandas define "columns = None", ou seja, ele exportará TODAS as colunas do seu DataFrame para o Excel.

Esse parâmetro **é opcional**. Recebe o nome das colunas, em string, dentro de uma lista.

header:

Esse parâmetro vai definir se o cabeçalho do seu DataFrame será exportado junto com a tabela. Por padrão, o pandas define "header = True", ou seja, ele exporta o cabeçalho junto com o DataFrame.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

index:

Esse parâmetro vai definir se o index do seu DataFrame será exportado junto com a tabela. Por padrão, o pandas define "index = True", ou seja, ele exporta o index junto com o DataFrame

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

index_label:

Esse parâmetro vai definir o título do index do seu DataFrame. Por padrão, o pandas define "index-label = None", ou seja, será o nome definido no DataFrame.

Só será visível no Excel, se index = True e header = True

Esse parâmetro **é opcional** e recebe o título do index, como string.

startrow:

Esse parâmetro vai definir a partir de qual linha o DataFrame será exportado. Por padrão, o pandas define “startrow = 0”, ou seja, ele será exportado a partir da primeira linha.

Esse parâmetro **é opcional** e recebe a posição, em integer, da primeira linha que receberá o DataFrame exportado.

startcol:

Esse parâmetro vai definir a partir de qual coluna o DataFrame será exportado. Por padrão, o pandas define “startcol = 0”, ou seja, ele será exportado a partir da primeira coluna.

Esse parâmetro **é opcional** e recebe a posição, em integer, da primeira coluna que receberá o DataFrame exportado.

inf_rep:

Esse parâmetro vai definir como serão representados os infinitos dentro do Excel, já que não existe esse tipo de representação em planilhas. Por padrão, o pandas define ”inf_rep” = “inf” , ou seja, infinitos serão representados como o texto “inf”.

Esse parâmetro **é opcional** e recebe uma string

freeze_panes:

Esse parâmetro vai definir quais linhas e colunas serão congeladas, sendo que as primeiras posições são “1”, enquanto “0” representa nenhuma. Por padrão, o pandas define “freeze_panes = None”, ou seja, nenhuma coluna será congelada.

Esse parâmetro **é opcional** e recebe uma tuple com as posições de linha e coluna.

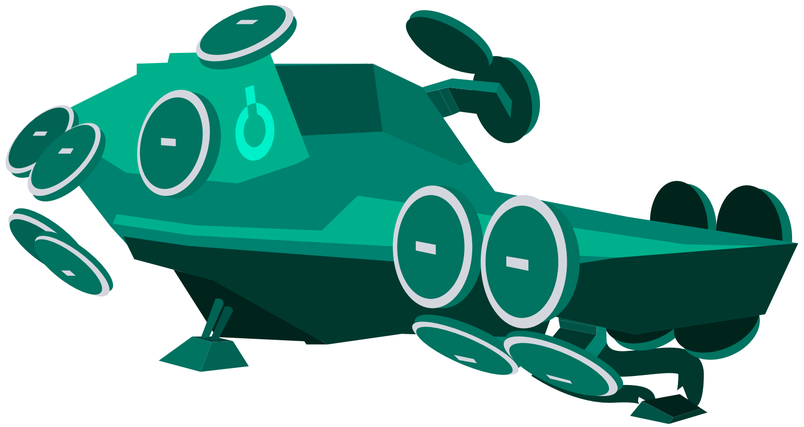
No exemplo abaixo, estamos lendo um arquivo onde precisamos pular as 6 primeiras linhas, definimos coluna “Data” como index, definimos que os valores de “NaN” serão “nd”, o separador decimal como vírgula (,) e pegamos apenas 3 colunas do DataFrame original, que são ‘Data’, ‘Fech Ajustado’, ‘Vol (MM R$).’

Exemplo:

```
import pandas as pd
import numpy as np

dados_petrobras = pd.read_Excel("dados_petro.xlsx",
                                skiprows=6,
                                index_col = "Data",
                               na_values="nd",
                               decimal = ",",
                               usecols = ['Data', 'Fech Ajustado', 'Vol (MM R$)'])


print(dados_petrobras)
```

Resposta:

```
            Fech Ajustado  Vol (MM R$)
Data
2022-09-12      31.580000   2267.814565
2022-09-09      31.790000   1588.898586
2022-09-08      31.800000   2011.363072
2022-09-06      32.100000   3555.963567
2022-09-05      33.330000   2536.813661
   ...            ...           ...
2022-05-20      25.450721   2697.429409
2022-05-19      24.968451   2165.226242
2022-05-18      24.551945   2730.939150
2022-05-17      24.961144   3107.951524
2022-05-16      25.289965   3184.449800
```



Mundo 9

Neste mundo abordaremos formas de integrar csv com a biblioteca pandas.

1. pandas.read_csv(filepath_or_buffer, sep = _NoDefault.no_default, delimiter = None, header = “infer”, names = _NoDefault.no_default, index_col = None, usecols = None, dtype = None, engine = None, converters = None, skiprows = None, skipfooter = 0, nrows = None, na_values = None, keep_default_na = True, na_filter = True, verbose = False, skip_blank_lines = True, parse_dates = None, infer_datetime_format = False, keep_date_col = False, date_parser = None, dayfirst = False, cache_dates = True, iterator = False, chunksize = None, compression = “infer”, thousands = None, decimal = “.”, lineterminator = None, quotechar =“ „ “, quoting = 0, doublequote = True, escapechar = None, comment = None, encoding = None, encoding_errors =“strict”, dialect = None, error_bad_lines = None, warn_bad_lines = None, on_bad_lines = None, delim_whitespace = False, low_memory = True, memory_map = False, float_precision = None, storage_options = None)

Primeiramente você deve entender o conceito de diretório e caminho do arquivo. Quando trabalhamos com dados dentro do computador, precisamos informar onde está localizado o arquivo. Fazemos isso por meio do caminho do arquivo, uma combinação de textos e endpoints que vão detalhar, para o computador, a localização de um arquivo. Já o diretório é o local final do computador onde o arquivo se encontra.

Quando queremos acessar um arquivo, que está localizado no local onde estamos trabalhando, não precisaremos especificar um caminho, apenas o nome do arquivo. Mas quando estamos trabalhando em um diretório diferente da localização do arquivo, aí sim, precisamos especificar seu caminho.

CSV é uma abreviação para **Comma Separated Values** que, em português, significa valores separados por vírgula. É a forma que a maioria dos dados são disponibilizados. No Brasil, como o nosso separador decimal é ” , ”, é utilizado o ” ; ” para separar valores nesse tipo de arquivo.

Parâmetros:

O parâmetro “filepath_or_buffer” é o único obrigatório.

filepath_or_buffer:

Esse parâmetro vai definir onde está localizado o arquivo, através do caminho dele. Vale lembrar que, se o arquivo estiver no mesmo diretório, do arquivo.py, não precisará especificar o caminho, apenas o nome do arquivo a ser lido.

Esse parâmetro **é obrigatório**. Pode receber o caminho, ou nome, do arquivo em string.

sep:

Esse parâmetro vai definir qual será o delimitador para separação dos dados. Na maioria das vezes, o csv não vem formatado tabularmente, e vem separado por um delimitador. No caso da CVM, usualmente eles utilizam o “ ; ”.

Esse parâmetro **é opcional**. Pode receber o delimitador em string.

delimiter:

Faz a mesma função do **sep**

header:

Esse parâmetro vai definir qual será o cabeçalho da sua tabela dentro do csv. Por padrão, o pandas define “header = 0”, ou seja, a primeira linha será o cabeçalho.

Repare que é possível definir mais de uma linha para cabeçalho, neste caso será uma tabela multiindex.

Esse parâmetro **é opcional**. Pode receber o número da posição da linha, em integer, ou pode receber uma lista com os números com as posições das linhas, em integer.

names:

Esse parâmetro vai definir qual será o nome das suas colunas dentro do Excel. Para esse parâmetro funcionar, não deve existir um cabeçalho já definido na coluna e a quantidade de nomes deve ser a mesma da quantidade de coluna. Por padrão, o pandas define "names = None", ou seja, ele vai considerar os nomes já definidos no DataFrame.

Esse parâmetro **é opcional**. Pode receber uma lista com a nova nomeação das colunas, em string.

index_col:

Esse parâmetro vai definir qual será serão as colunas que serão consideradas index. Por padrão, o pandas define "index_col = None", ou seja, não define nenhuma coluna como index.

Esse parâmetro **é opcional**. Pode receber uma lista com o nome das colunas, em string ou pode receber uma lista com a posição das colunas, em integer.

usecols:

Esse parâmetro vai definir quais colunas serão selecionadas. Por padrão, o pandas define "usecols = None", ou seja, como não foi definido ele passará todas as colunas.

Pode receber também uma função, por exemplo, que só selecione nome das colunas com mais de 10 caracteres:

df = pd.read_csv("data.csv", usecols = lambda x: len(x) > 10)

Esse parâmetro **é opcional**. Pode receber o nome da coluna a ser passado em string, dentro de uma lista. Ou os nomes das colunas a serem passadas, em string, dentro de uma lista. Pode receber também uma função

dtype:

Esse parâmetro vai definir o tipo de cada coluna, Você pode defini-las como integers, strings ou floats por exemplo.

Esse parâmetro **é opcional**. Recebe um tipo definido para o DataFrame inteiro, ou um dicionário que vai definir cada coluna como um tipo.

engine:

Esse parâmetro vai definir qual ignição será usada. Por padrão, o pandas define “engine = None”, ou seja, nenhuma ignição será definida.

Esse parâmetro **é opcional**. Recebe alguns formatos que podem ser definidos:

c => É uma inginição mais rápida

pyarrow => Assim como “c” é uma inginição mais rápida

Python => Mais devagar porém mais completa

converters:

Esse parâmetro vai definir uma função para customizar o nome de uma coluna enquanto o pandas lê o arquivo. Um exemplo que pode ser usado é uma função lambda que pega as 3 primeiras letras de cada item de uma coluna:

3_letras = lambda x: x[0:3]

Então, a sintaxe seria assim **converters = { ‘nome_coluna’: 3_letras}**. Por padrão, o pandas define “converter = None”, ou seja, não define nenhuma função a ser usada. Pode ser também uma função que transforma todos em string.

string = lambda x: str(x)

Esse parâmetro **é opcional**. Recebe um dictionary com o nome da coluna e a função a ser utilizada

skiprows:

Esse parâmetro vai definir a quantidade de linhas a ser puladas, antes de contabilizar os dados presentes no arquivo. Por padrão, o pandas considera “skiprows = None”, ou seja, nenhuma linha vai ser pulada.

Pode receber também uma função, por exemplo, uma função que pule os números ímpares:

df = pd.read_csv(”data.csv”, skiprows = lambda x: x%2! = 0)

Esse parâmetro **é opcional**. Pode receber: o número das linhas, em integer, dentro de uma lista. Pode receber a quantidade de linhas (começando na posição 0). Ou pode receber uma função que pule linhas de acordo com uma condição.

skipfooter:

Esse parâmetro vai definir a quantidade de linhas a serem excluídas do DataFrame, tendo como ponto de partida a última linha do DataFrame.

Esse parâmetro **é opcional**. Pode receber um número, em integer, que determinará a quantidade de linhas.

nrows:

Esse parâmetro vai definir a quantidade de linhas que um DataFrame vai ter. Sua contagem começa da linha 0, então se você definir “nrows = 5”, serão escolhidas as 5 primeiras linhas do seu DataFrame. Por padrão, o pandas define “nrows = None”, ou seja, ele não define um limite máximo de linha

Esse parâmetro **é opcional**. Pode receber apenas a quantidade de linhas máximas de um DataFrame, em interger.

na_values:

Esse parâmetro vai definir valores que serão transformados em vazios (NaN). Muitas vezes, os valores que são definidos como vazios no documento(“nd”,”#NA”,”N/A”), não são reconhecidos pelo Python como tal. Por padrão, o pandas considera “na_values = None”, ou seja, nenhum valor foi definido como vazio.

Esse parâmetro **é opcional**. Pode receber o valor a ser considerado vazio, em string. Ou os valores a serem considerados vazios, em string, dentro de uma lista.

keep_default_na:

Esse parâmetro vai definir se os valores “NaN” serão mantidos no DataFrame ou não. Por padrão, o pandas define “keep_default_na = True”, ou seja, ele manterá os valores “NaN”. Caso seja definido “keep_default_na = False”, ele excluirá o valor “Nan” e deixará um espaço vazio.

Esse parâmetro é opcional e recebe um booleano: True ou False.

na_filter:

Esse parâmetro vai definir que espaços vazios ou em branco serão representados como valores “NaN”. Por padrão, o pandas define “na_filter = True”, ou seja, em caso de valores em branco, estes serão representados como “NaN”. Caso “na_filter = False”, os valores em branco, continuariam em branco.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

skip_blank_lines:

Esse parâmetro vai definir se as linhas em branco vão ser puladas ao invés de serem consideradas como “NaN”. Por padrão, o pandas define “skip_blank_lines = True”, ou seja, ele pulará as linhas em branco.

Atenção!!!!

Como se sabe, o csv (comma-separated-values) é um arquivo onde separa os valores por vírgulas (,). Logo, uma linha em branco é diferente de uma linha que não tem os valores.

O exemplo a seguir ilustra bem os 3 casos: O primeiro, de uma linha completa separada por virgulas. O segundo de uma linha sem valores, que serão representados como “NaN”. O terceiro de uma linha vazia, onde então poderá escolher se será pulada ou definida como “NaN”

1| WEGE3,35.90,ENERGIA -> Linha “normal

2| ,,,, -> Linha com valores faltantes

3| -> Linha em branco que será pulada caso o parâmetro seja True

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

parse_dates:

Esse parâmetro vai definir quais colunas serão definidas como data. Por padrão, o pandas define "parse_dates = None", ou seja, ele não considera nenhuma coluna como data.

obs: Esse parâmetro permite juntar datas presentes em diferentes colunas. Caso o dia, mês e ano estejam em 3 colunas distintas, poderia ser usado a seguinte sintaxe: **parse_dates = [['coluna_dia' , 'coluna_mes' , 'coluna_ano']].**

obs2: Esse parâmetro também permite definir o nome da coluna final da data após coletar dados de colunas diferentes, com a seguinte sintaxe: **parse_dates = {'data_final': 'coluna_dia' , 'coluna_mes': 'coluna_ano'}.**

obs3: Este parâmetro pode receber também um valor booleano, True ou False. Em caso de True, ele tentará definir o índex como uma data, que caso tenha um formato de data terá seu valor reconhecido.

Esse parâmetro **é opcional**. Pode receber o nome da coluna em string, a posição da coluna em integer. Ou pode receber uma lista com as posições, em integer, e/ou com os nomes, em string. Pode receber também um booleano: True ou False.

infer_datetime_format:

Esse parâmetro deve ser usado em conjunto com o **parse_dates**. Se " parse_dates = True " e " infer_datetime_format = True", o Python tentará mudar o formato das strings para datas e escolherá o jeito mais rápido de fazer isso. Em alguns casos, isso pode aumentar a velocidade de análise em 5-10x.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

keep_date_col:

Esse parâmetro deve ser usado em conjunto com o **parse_dates**. Se for utilizado o método do parse_dates de juntar colunas e formar uma coluna data, por padrão, o pandas define "keep_date_col = False", ou seja, ele exclui essas colunas deixando apenas a resultante da junção. Caso "keep_date_col = True", o pandas deixaria tanto a coluna resultante como as que foram utilizadas para juntar.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

date_parser:

Esse parâmetro é utilizado em conjunto com o "parse_dates" e define uma função para converter uma coluna. Por padrão, o pandas define "date_parsesr = None", ou seja, nenhuma função é definida.

Esse parâmetro **é opcional**. Pode receber uma função lambda com a seguinte sintaxe:

date_parser = lambda x: datetime.strptime(x, "%Y-%d-%m %H:%M:%S")

Nessa função acima, estamos passando itens que são strings para o formato datetime.

day_first:

Esse parâmetro é utilizado para definir se uma data está no estilo EUA ou no Europeu. Por padrão, o pandas define "dayfirst = False", ou seja, ele considera o estilo EUA de data.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

iterator:

Esse parâmetro é utilizado em conjunto com o **chunksize**. Ele importará o arquivo csv como um TextFileReader, que pode ser acessado através do método df.**get_chunk()**. Por padrão, o pandas define "iterator = False", ou seja, ele retornará um DataFrame.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

chunksize:

Esse parâmetro é utilizado em conjunto com o iterator. Ele definirá a quantidade de itens a serem retornados com o método df.get_chunk().

Esse parâmetro **é opcional** e recebe a quantidade de itens como um integer.

compression:

Esse parâmetro é utilizado quando precisa-se ler arquivos comprimidos. Por padrão, o pandas define "compression = None".

Atenção!!!

Dentro do arquivo comprimido deve-se ter apenas um arquivo ou ele retornará erro de multiplos arquivos.

Esse parâmetro **é opcional** e recebe os formatos já pré-definidos como strings. Os formatos possíveis são ['zip', 'gzip', 'bz2', 'zstd', 'tar'].

thousands:

Esse parâmetro vai definir o separador **milenar** para números que estão formatados como **texto** no csv. Os itens do csv, que já estão formatados como números, vão automaticamente ser formatados como float ou integrer com separador ".".

Esse parâmetro **é opcional**. Pode receber apenas um separador milenar em string.

decimal:

Esse parâmetro vai definir o separador **decimal** para números que estão formatados como t**exto** no csv. Os itens do csv, que já estão formatados como números, vão automaticamente ser formatados como float ou integrer com separador ".".

Esse parâmetro **é opcional**. Pode receber apenas um separador decimal em string.

lineterminator:

Esse parâmetro define um caractere para determinar o fim de uma linha ou coluna.

Esse parâmetro **é opcional**. Pode receber apenas um separador decimal em string.

comment:

Esse parâmetro vai definir um caractere, uma sequência de caracteres ou uma palavra para limitar as strings de um DataFrame. Qualquer coisa depois da sequência definida, será excluído no DataFrame inteiro. As palavras que não tiverem em sua composição a sequência de caracteres definida não serão modificadas. Por exemplo, se tivermos em uma coluna as palavras [WEGE3-ON , WEGE3-PN , PETR4-ON , PETR4-PN] e definirmos 'comment = 'WEGE3, a coluna ficará desse jeito: [WEGE3 , WEGE3, ,].

Esse parâmetro **é opcional**. Pode receber apenas um caractere,uma sequência de caracteres ou uma palavra como string.

encoding:

Esse parâmetro é um pouco mais complexo. Encoding é a forma como o arquivo vai ser extraído. Existem vários tipos de encoding. Não existe um tipo certo de encoding a ser usado, cada caso é um caso.

Como saber qual encoding usar?

Como foi abordado, depende de cada caso, mas pode ser pelo tipo de erro que está dando ou pela instrução do site no qual o csv foi baixado.

Esse parâmetro **é opcional**. Pode receber o nome dos encoding em string. A tabela com os encodings

on_bad_lines:

Esse parâmetro vai definir o que fazer com uma linha ruim, que não consiga ser interpretada pelo Python.

Esse parâmetro **é opcional**. Pode receber uma string, com os seguinte formatos já pré-definidos:

‘error’ => Retornar um erro quando aparecer uma linha ruim e parar o programa

‘warm’ => Retornar um erro porém passar a linha

‘skip’ => Passar a linha

No exemplo abaixo, estamos lendo um arquivo onde precisamos separar por vírgula (,), definir a forma de ser extraído pelo “encoding”, definir qual será o separador decimal, qual será a coluna index e qual coluna usaremos.

Exemplo:

```
import pandas as pd
import numpy as np

dados_bp = pd.read_csv("dfp_cia_aberta_DRE_con_2021.csv",
                       sep = ";",
                       encoding = 'ISO-8859-1',
                      decimal = ",",
                      index_col = "DT_REFER",
                      usecols = ['DT_REFER','CD_CONTA', 'DS_CONTA', 'VL_CONTA'])

print(dados_bp)
```

Resposta:

```
            CD_CONTA                                DS_CONTA                VL_CONTA
DT_REFER
2021-12-31      3.01          Receitas de Intermediação Financeira   98659704.0000000000
2021-12-31      3.01          Receitas de Intermediação Financeira  125947217.0000000000
2021-12-31    3.01.01                       Receita de Juros         98659704.0000000000
2021-12-31    3.01.01                       Receita de Juros        125947217.0000000000
2021-12-31      3.02          Despesas de Intermediação Financeira  -43232120.0000000000
   ...       ...                                   ...                          ...
2021-12-31  3.99.01.01                                    ON          2.4903000000
2021-12-31    3.99.02                  Lucro Diluído por Ação          0.0000000000
2021-12-31    3.99.02                  Lucro Diluído por Ação          0.0000000000
2021-12-31  3.99.02.01                                    ON          0.6532000000
2021-12-31  3.99.02.01                                    ON          2.4754000000
```

2. pandas.to_csv(path_or_buf= None , sep= “,” , na_rep = “ “, float_format = None, columns = None, header = True, index = True, index_label = None, mode = “w”, encoding = None, compression = “infer”, quoting = None, quotechar = “ „ “, lineterminator = None, chunksize = None, date_format = None, doublequote = True, escapechar = None, decimal = “.” , errors = “strict”, storage_options =None)

Esse método envia um DataFrame, ou Series, para um arquivo csv. Caso não exista o arquivo csv, ele criará um.

Parâmetros:

O parâmetro “path_or_buf” é o único obrigatório.

path_or_buf:

Esse parâmetro vai definir o nome do arquivo. Note que é nesse parâmetro que o formato do arquivo vai ser definido.

Esse parâmetro **é obrigatório**. Pode receber o nome do arquivo em string.

sep:

Esse parâmetro vai definir qual será o delimitador para separação dos dados. Na maioria das vezes, o csv não vem formatado tabularmente, e vem separado por um delimitador. No caso da CVM, usualmente eles utilizam o “ ; ”.

Esse parâmetro **é opcional**. Pode receber o delimitador em string.

na_rep:

Esse parâmetro vai definir qual será o valor dos ‘NaN’ dentro do csv. Por padrão, o pandas define ‘na_rep = “ ”‘, ou seja, os valores ‘NaN’ serão exportados para o csv como vazio (sem nada).

Esse parâmetro **é opcional**. Pode receber o valor a ser considerado como ‘NaN’, como string.

float_format:

Esse parâmetro vai definir a quantidade de casas decimais que terão os floats dentro do Excel. Por padrão, o pandas define "float_format = None", ou seja, será a quantidade de casas decimais definidas no DataFrame.

Esse parâmetro **é opcional**. Recebe a quantidade de casas decimais, em string, no seguinte formato: "%.2f " (2 casas decimais).

columns:

Esse parâmetro vai definir quais serão as colunas do seu DataFrame que serão exportadas para o Excel, Por padrão, o pandas define "columns = None", ou seja, ele exportará TODAS as colunas do seu DataFrame para o Excel.

Esse parâmetro **é opcional**. Recebe o nome das colunas, em string, dentro de uma lista.

header:

Esse parâmetro vai definir se o cabeçalho do seu DataFrame será exportado junto com a tabela. Por padrão, o pandas define "header = True", ou seja, ele exporta o cabeçalho junto com o DataFrame

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

index:

Esse parâmetro vai definir se o index do seu DataFrame será exportado junto com a tabela. Por padrão, o pandas define "index = True", ou seja, ele exporta o index junto com o DataFrame

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

index_label:

Esse parâmetro vai definir o título do index do seu DataFrame. Por padrão, o pandas define "index-label = None", ou seja, será o nome definido no DataFrame.

Só será visível no Excel, se index = True e header = True

Esse parâmetro **é opcional** e recebe o título do index, como string.

mode:

Esse parâmetro vai definir o modo como será feita a alteração do seu arquivo. Por padrão, o pandas define "mode = "w"", ou seja, caso não exista um arquivo, ele criará um. Caso já tenha um arquivo, ele fará a substituição.

Esse parâmetro **é opcional** e recebe o modo como uma string, já pré-definida.

"w" => Criará um arquivo, caso não exista. Caso o arquivo exista, ele substituirá por um novo.

"x" => Criará um arquivo, caso não exista. Caso o arquivo exista, ele retornará um erro

"a" => Criará um arquivo, caso não exista. Caso o arquivo exista, ele irá adicionar informações ao final do arquivo.

encoding:

Esse parâmetro é um pouco mais complexo. Encoding é a forma como o arquivo vai ser extraído. Existem vários tipos de encoding. Não existe um tipo certo de encoding a ser usado, cada caso é um caso.

Como saber qual encoding usar?

Como foi abordado, depende de cada caso, mas pode ser pelo tipo de erro que está dando ou pela instrução do site no qual o csv foi baixado.

Esse parâmetro **é opcional**. Pode receber o nome dos encoding em string. A tabela com os encodings

compression:

Esse parâmetro vai definir para qual tipo de arquivo comprimido o arquivo.csv vai ser exportado. Por padrão, o pandas define "compression = None", ou seja, não será exportado como um arquivo comprimido. Para exportar um arquivo comprimido, o **to_csv** respeita a seguinte sintaxe:

df.to_csv(arquivo.zip ,compression = {'method' : ,zip', ,archive_name': arquivo.csv'})

Onde é definido o nome do arquivo e qual será o formato do arquivo comprimido.

Esse parâmetro **é opcional** e recebe os formatos , já pré-definidos, como strings. Os formatos possíveis são ['zip', 'gzip', 'bz2', 'zstd', 'tar'].

lineterminator:

Esse parâmetro vai definir o separador de palavras e separar entre linhas. Por exemplo, se tiver uma coluna com palavras separadas por "/". Se for definido "lineterminator = '/' " , as palavras que estiverem depois do '/' passarão para linha de baixo e para outras colunas.

Esse parâmetro **é opcional**. Pode receber apenas um separador decimal em string.

date_format:

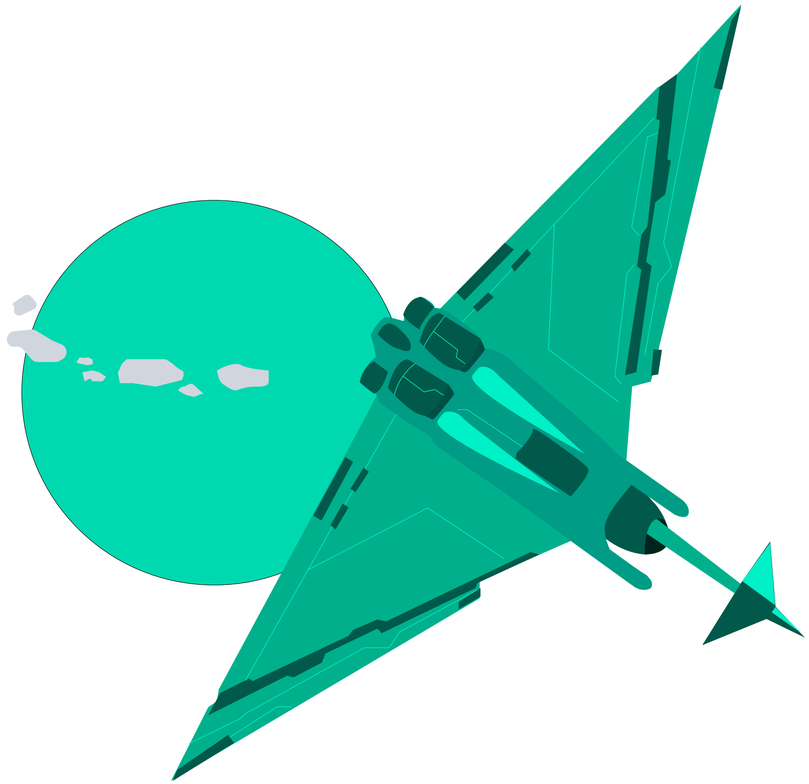
Esse parâmetro vai definir o formato da data a ser passado

Esse parâmetro **é opcional**. Pode receber o formato da data em string

decimal:

Esse parâmetro vai definir o separador **decimal** para números que estão formatados como **texto** no csv. Os itens do csv, que já estão formatados como números, vão automaticamente ser formatados como float ou integrer com separador ".".

Esse parâmetro **é opcional**. Pode receber apenas um separador decimal em string.



Mundo 10

Neste mundo abordaremos sobre a biblioteca pandas datareader

1. pandas datareader

O pandas datareader é uma biblioteca Python criada para extrair dados da internet. É bastante utilizada para retirar dados de ações em tempo real. Basicamente, é uma biblioteca que junta um compilado de informações de sites famosos e conceituados que possuem todos, e quando eu digo todos eu quero dizer TODOS os tipos de informações. São centenas de coleções e obviamente escolhemos as mais importantes.

Tipos de dados que podemos extrair do pandas datareader:

Mercado Financeiro

- AlphaVantage - Equity e FOREX. Dados Intraday dos últimos 3-5 dias.
- The Investors Exchange (IEX) - Dados de ações, ETFs e fundos.
- Moscow Exchange (MOEX) - Dados de ações
- Quandl - Dados de ações, ETFs e fundos.
- Tiingo - Dados de ações, ETFs e fundos.
- Yahoo Finance - Dados de ações, ETFs e fundos.

Dados específicos de finanças:

- Fama-French Data (Ken French's Data Library) - Dados sobre factor investing nos EUA
- NASDAQ - Todos os símbolos negociados na NASDAQ.
- Naver Finance - Dados do mercado coreano.
- Stooq.com - Dados de índices.

Economia

- Federal Reserve Economic Data (FRED) - Dados da economia americana
- Bank of Canada - Economia canadense.
- Econdb - Economia mundial
- Organisation for Economic Co-operation and Development (OECD) - Indicadores econômicos
- World Bank - Indicadores econômicos

Mundo 11

Neste mundo abordaremos formas de extrair dados das ações pelo Yahoo Finance usando o pandas datareader

1. pandasdatareader.get_data_yahoo(symbols = None, start = “ ” , end = “ ”, retry_count = 3, pause = 0.1, adjust_price = False, ret_index = False, chunksize = 25, interval = “d”, get_actions = False, adjust_dividends = False)

Parâmetros:

O parâmetro “symbols” é o único obrigatório.

Obs: Caso só o symbols seja definido, ele retornará as informações dos últimos 5 anos.

symbols:

Esse parâmetro vai definir o ticker de cada empresa, listada na bolsa.

Obs: Para empresas brasileiras é preciso adicionar o sufixo “.SA”. Por exemplo, **”WEGE3.SA”**.

Esse parâmetro **é obrigatório**. Pode receber o ticker de uma empresa, como string, ou receber uma lista com os tickers das empresas, em string.

start:

Esse parâmetro vai definir a data de começo de contagem. Se você definir apenas ele, será retornado as informações do dia informado até o dia de hoje.

Esse parâmetro **é opcional**. Pode receber as datas em formato de string ou datetime.

end:

Esse parâmetro vai definir a data final da contagem.

Esse parâmetro **é opcional**. Pode receber as datas em formato de string ou datetime.

retry_count:

Esse parâmetro vai definir a quantidade de requisições que serão feitas.

Esse parâmetro **é opcional**. Pode receber a quantidade de solicitações em integer.

pause:

Esse parâmetro vai definir qual será o tempo, em segundos, do intervalo entre as solicitações de requisição.

Esse parâmetro **é opcional**. Pode receber a quantidade de segundos em integer.

adjust_price:

Esse parâmetro vai definir se serão ajustados todos os dados baseando-se na coluna "Adj Close". Porém após ajustar os dados, ele adiciona a coluna "Adj_Ratio" no lugar da coluna "Adj Close". Por padrão, o pandas considera "adjust_price = False"

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

ret_index:

Esse parâmetro vai definir se será incluída uma coluna "ret_index" nos dados. Por padrão, o pandas considera "ret_index = False", ou seja, não retornará com a coluna "ret_index".

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

chunksize:

Esse parâmetro vai definir o número de ações a serem carregadas de uma vez só antes de iniciar uma pausa. Por padrão, o pandas define "chunksize = 25", ou seja, serão carregadas 25 ações antes de iniciar uma pausa.

Esse parâmetro **é opcional**. Pode receber a quantidade de ações em integer.

interval:

Esse parâmetro vai definir o intervalo de dados, se serão diariamente (d), semanalmente (w) ou mensalmente (m). Por padrão, o pandas define o intervalo como sendo diário (d).

Esse parâmetro **é opcional**. Pode receber parâmetros pré-definidos no formato string.

d = diariamente
w = semanalmente
m = mensalmente

get_actions:

Esse parâmetro vai definir se os dados serão retornados junto com uma coluna com os dividendos e os desdobramentos. Por padrão, o pandas define "get_actions = False", ou seja, os dados não serão retornados com essas colunas.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

adjust_dividends:

Esse parâmetro vai definir se os dividendos serão ajustados ou não. Por padrão, o pandas define "adjust_dividends = False", ou seja, os dividendos não serão ajustados.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

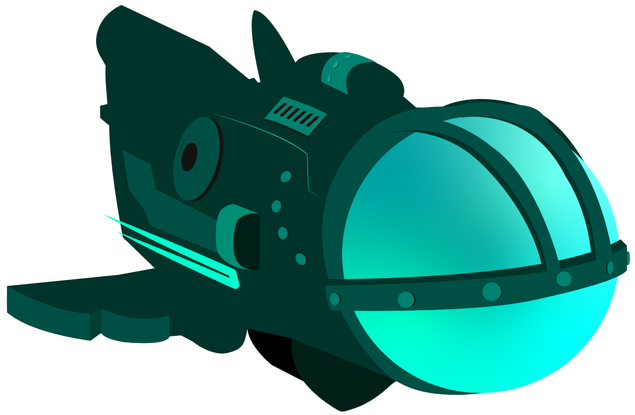
No exemplo abaixo estamos extraindo um DataFrame com o preço ajustado , desde 2018-12-31, de 5 ações da bolsa brasileira.

Exemplo:

```
from pandas_datareader import data as pdr

#puxar ddos várias ações
acao = ["RPAD5.SA", "CASH3.SA", "VALE3.SA", "ATOM3.SA"]
dados = pdr.get_data_yahoo(symbols = acao,
                           start = "2018-12-31")['Adj Close']
print(dados)
```

Resposta:

Symbols	RPAD5.SA	CASH3.SA	VALE3.SA	ATOM3.SA
Date				
2019-01-02	4.451818	NaN	37.180229	2.060647
2019-01-03	4.851542	NaN	35.659252	2.143404
2019-01-04	4.851542	NaN	37.980740	2.102026
2019-01-07	4.851542	NaN	37.776974	2.126853
2019-01-08	4.851542	NaN	38.140846	2.110301
...	...	...	...	...
2022-10-04	7.770000	1.23	75.790001	2.560000
2022-10-05	8.490000	1.27	76.959999	2.500000
2022-10-06	8.490000	1.35	75.230003	2.500000
2022-10-07	8.490000	1.31	75.510002	2.600000
2022-10-10	NaN	1.30	NaN	NaN



Mundo 12

Neste mundo abordaremos formas de extrair dados do Federal Reserve Economic Data (FRED) usando o pandas datareader.

Documentação: https://fred.stlouisfed.org/categories

1. pandasdatareader.get_data_fred(symbols = None, start = “” , end = “”, retry_count = 3, pause = 0.1,)

Parâmetros:

O parâmetro “symbols” é o único obrigatório.

Obs: Caso só o symbols seja definido, ele retornará as informações dos últimos 5 anos.

symbols:

Esse parâmetro vai definir a identificação de cada classe. Para conseguir essa identificação precisará acessar a documentação do FRED.

Esse parâmetro **é obrigatório**. Pode receber o código da informação, como string, ou receber uma lista com os códigos das informações, em string.

start:

Esse parâmetro vai definir a data de começo de contagem. Se você definir apenas ele, será retornado as informações do dia informado até o dia de hoje.

Esse parâmetro **é opcional**. Pode receber as datas em formato de string ou datetime.

end:

Esse parâmetro vai definir a data final da contagem.

Esse parâmetro **é opcional**. Pode receber as datas em formato de string ou datetime.

retry_count:

Esse parâmetro vai definir a quantidade de requisições que serão feitas.

Esse parâmetro **é opcional**. Pode receber a quantidade de solicitações em integer.

pause:

Esse parâmetro vai definir qual será o tempo, em segundos, do intervalo entre as solicitações de requisição.

Esse parâmetro **é opcional**. Pode receber a quantidade de segundos em integer.

No exemplo abaixo estamos extraindo um DataFrame com o yield dos Estados Unidos, desde 1910-12-31 até os dias atuais.

Exemplo:

```
from pandas_datareader import data as pdr

dado = "TB3MS" #yield de 3 meses americano
inicio = "1910-12-31"
dado_fred = pdr.get_data_fred(dado, inicio)

print(dado_fred)
```

Resposta:

```
            TB3MS
DATE
1934-01-01   0.72
1934-02-01   0.62
1934-03-01   0.24
1934-04-01   0.15
1934-05-01   0.16
...           ...
2022-05-01   0.98
2022-06-01   1.49
2022-07-01   2.23
2022-08-01   2.63
2022-09-01   3.13

[1065 rows x 1 columns]
```

Mundo 13

Neste mundo abordaremos formas de extrair dados do World Bank usando o pandas datareader.

Documentação: https://data.worldbank.org/

1. wb.get_indicators()

Parâmetros:

Não recebe nenhum parâmetro e retorna os indicadores. Existem + 20.000 indicadores a serem usados.

2. wb.search(string = None, field = “name”, case = False)

Parâmetros:

string:

Esse parâmetro vai definir qual informação será extraída. Recebe uma expressão.

Esse parâmetro **é obrigatório**. Pode receber a expressão que retorna as informações desejadas ou um indicador.

field:

Esse parâmetro vai receber o nome e o id do valor que você está procurando.

Esse parâmetro **é obrigatório**. Pode receber a identificação e o nome no formato string.

case:

Esse parâmetro vai definir se as pesquisas serão sensíveis a letras maiúsculas e minúsculas. Por padrão, o pandas define “Case = False”, ou seja, não serão sensíveis às letras maiúscula e minúscula.

Esse parâmetro é opcional e recebe um booleano: True ou False.

No exemplo abaixo estamos extraindo um DataFrame, que demonstra o PIB per capita constante (em dólar) em diferentes anos.

Exemplo:

```
from pandas_datareader import wb

pib_percapta = wb.search('gdp.*capita.*const')
print(pib_percapta)
```

Resposta:

```
                         id  ...            topics
716       6.0.GDPpc_constant  ...  Economy & Growth
10641         NY.GDP.PCAP.KD  ...  Economy & Growth
10643         NY.GDP.PCAP.KN  ...  Economy & Growth
10645      NY.GDP.PCAP.PP.KD  ...  Economy & Growth
10646  NY.GDP.PCAP.PP.KD.87   ...

[5 rows x 7 columns]
vinicius@galaxia1:~/d
```

3. wb.download(country = None, indicator = None, start = “ ”, end = “ ”, freq = None, errors = ’warn’,)

Parâmetros:

indicator:

Esse parâmetro vai definir a identificação do indicador que você deseja extrair. Esse parâmetro pode ser conseguido utilizando os métodos wb.search(), wb.get_indicators() ou pelo site.

Esse parâmetro **é obrigatório**. Pode receber a identificação no formato string.

country:

Esse parâmetro vai definir os países que terão as informações extraídas.

Esse parâmetro **é obrigatório**. Pode receber o código dos países no formato string ou pode receber uma lista com o codigo dos países.

obs: O codigo dos países segue a seguinte formatação.

start:

Esse parâmetro vai definir a data de começo de contagem. Se você definir apenas ele, será retornado as informações do dia informado até o dia de hoje.

Esse parâmetro **é opcional**. Pode receber as datas em formato de string ou datetime.

end:

Esse parâmetro vai definir a data final da contagem.

Esse parâmetro **é opcional**. Pode receber as datas em formato de string ou datetime.

freq:

Esse parâmetro vai definir a frequência de dados, se serão mensalmente (M), anualmente (A) ou Trimestral (Q). Por padrão, o pandas define o intervalo como sendo anual (A).

Esse parâmetro **é opcional**. Pode receber parâmetros pré-definidos no formato string.

M = mensalmente
A = anualmente
Q = trimestralmente

errors:

Esse parâmetro vai definir o que fazer em caso de erro. Por padrão, o pandas define “errors = ‘warn’ “, ou seja, em caso de erro ele retornará um aviso.

Esse parâmetro **é opcional** e recebe dois métodos pré-definidos. Que são:

raise => Em caso de erro, retorna um erro

ignore => Em caso de erro, ignorar o erro e manter os valores originais (para cada item)

warn => Em caso de erro, retorna um aviso

No exemplo abaixo estamos extraindo um DataFrame, com 200 linhas, e agrupando por países , ou seja, estamos somando as informações de cada país. Depois pegamos a primeira informação (mais recente) e a última informação (mais antiga), dividimos uma pela outra para achar o crescimento.

Exemplo:

```
from pandas_datareader import wb
pib_percapta = wb.download(indicator='NY.GDP.PCAP.KD',
                           country=['BRA', 'USA', 'ARG', 'MEX'],
                           start=1980, end=2022)

print(pib_percapta)
```

Resposta:

```
                        NY.GDP.PCAP.KD
country       year
Argentina     2021    12390.808688
              2020    11344.405742
              2019    12712.970738
              2018    13105.397163
              2017    13595.037355
...                             ...
United States 1984    33906.351244
              1983    31893.199049
              1982    30775.441389
              1981    31640.677787
              1980    31161.930725

[168 rows x 1 columns]
```

4. Link com o código dos países:

https://en.wikipedia.org/wiki/List_of_ISO_3166_country_codes

Mundo 14

Neste mundo abordaremos formas de extrair dados de Factor Investing pelo Fama-French Bank usando o pandas datareader.

Documentação: http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html

1. pandasdatareader.get_avaible_datasets()

Esse é um submódulo do pandas_datareader que precisa ser exportado.

from pandas_datareader.famafrench import get_available_datasets

Parâmetros:

Não recebe nenhum parâmetro e retorna os nomes e id's dos datasets(conjunto de dados tabulares) disponíveis. São + de 150 datasets disponíveis.

2. pandasdatareader.get_data_famafrench(symbols = None, start = None, end = None, retry_count = None, pause = None)

Parâmetros:

Recebe o parâmetro 'symbols' como obrigatório.

symbols:

Esse parâmetro vai definir a identificação de cada DataSet. As identificações podem ser conseguidas utilizando o método anterior "get_avaible_datasets()".

Esse parâmetro **é obrigatório**. Pode receber a identificação dos DataSets como uma string.

start:

Esse parâmetro vai definir a data de começo de contagem. Se você definir apenas ele, será retornado as informações do dia informado até o dia de hoje.

Esse parâmetro **é opcional**. Pode receber as datas em formato de string ou datetime.

end:

Esse parâmetro vai definir a data final da contagem.

Esse parâmetro **é opcional**. Pode receber as datas em formato de string ou datetime.

retry_count:

Esse parâmetro vai definir a quantidade de requisições que serão feitas.

Esse parâmetro **é opcional**. Pode receber a quantidade de solicitações em integer.

pause:

Esse parâmetro vai definir qual será o tempo, em segundos, do intervalo entre as solicitações de requisição.

Esse parâmetro **é opcional**. Pode receber a quantidade de segundos em integer.

No exemplo abaixo estamos extraindo um DataFrame com valores desde 1960 sobre factor investing. O nome do DataSet foi adquirido a partir do método **get_avaible_datasets()**

Exemplo:

```
from pandas_datareader import data as pdr

dados_factor_investing = pdr.get_data_famafrench('F-F_Research_Data_Factors', start = 1960)
print(dados_factor_investing)
```

Resposta:

```
{0:          Mkt-RF   SMB   HML    RF
Date
1960-01   -6.98  2.09  2.78  0.33
1960-02    1.17  0.51 -1.93  0.29
1960-03   -1.63 -0.49 -2.94  0.35
1960-04   -1.71  0.32 -2.28  0.19
1960-05    3.12  1.21 -3.70  0.27
...         ...   ...   ...   ...
2022-04   -9.46 -1.41  6.19  0.01
2022-05   -0.34 -1.85  8.41  0.03
2022-06   -8.43  2.09 -5.97  0.06
2022-07    9.57  2.81 -4.10  0.08
2022-08   -3.78  1.39  0.31  0.19

[752 rows x 4 columns],

1:       Mkt-RF    SMB    HML    RF
Date
1960   -1.46  -2.78  -4.62  2.66
1961   24.81   1.28   5.42  2.13
1962  -12.90  -8.29   8.92  2.73
1963   17.84  -6.18  15.69  3.12
1964   12.54  -0.82   9.72  3.54
...      ...    ...    ...   ...
2017   21.51  -4.97 -13.51  0.80
2018   -6.95  -3.21  -9.73  1.83
2019   28.28  -6.10 -10.34  2.15
2020   23.66  13.17 -46.57  0.45
2021   23.56  -3.92  25.57  0.04

[62 rows x 4 columns], 'DESCR': 'F-F Research Data Fac-
tors\n------------------------\n\nThis file was created
by CMPT_ME_BEME_RETS using the 202208 CRSP database. The
1-month TBill return is from Ibbotson and Associates,
Inc. Copyright 2022 Kenneth R. French\n\n  0 : (752 rows
x 4 cols)\n  1 : Annual Factors: January-December (62
rows x 4 cols)'}
```

Mundo 15

Neste mundo abordaremos formas de integrar o SQL com o pandas, utilizando a biblioteca sqlalchemy.

1. sqlalchemy.create_engine(url, echo= ' ', encoding= False)

Parâmetros:

encoding:

Esse parâmetro representa o tipo de codificação utilizado pelo banco de dados.

Esse parâmetro **é opcional**. Pode receber a identificação na formatação de uma string.

echo:

Esse parâmetro determina se as instruções SQL serão exibidas no console. Por padrão, o sqlalchemy define “echo = False”.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

url:

Esse parâmetro é obrigatório e utiliza uma url pré-definida com parâmetros das configurações. É importante notar que cada banco de dados possui uma url diferente. Segue a sintaxe da url:

url: dialect+driver://username:password@host:port/database

Onde:

dialect:

É o nome do serviço utilizado como gerenciamento de banco de dados. Exemplos: **mysql, oracle, postgresql, etc..**

driver:

É o nome da ignição criada para integrar o Python com o gerenciador de banco de dados. No nosso caso, como utilizamos o mysql, poderia ser tanto o **pymysql** ou o **mysqldb**, que pode ser melhor utilizado dependendo da forma que você vai trabalhar.

username:

É o nome do usuário do gerenciador do banco de dados. Por padrão, quando se utiliza o local host, os bancos de dados definem que será **root**.

password:

É a senha definida para acessar o gerenciador de banco de dados.

host:

É o local onde está armazenado seu banco de dados que, no seu caso será, poderá ser o **localhost**, dentro do computador. Entretanto, poderia ser um serviço na nuvem como AWS. Neste caso seria necessário passar a url do banco de dados.

port:

O número da porta é a forma como os bancos de dados se comunicam com o servidor. Ele determina qual programa de banco de dados está sendo usado e quais são as configurações do servidor. Geralmente, um banco de dados utiliza a porta padrão 3306.

database:

É o nome do banco de dados localizado dentro do gerenciador de banco de dados.

Esse parâmetro **é obrigatório**. Pode receber a url no formato string.

No exemplo abaixo estamos criando uma conexão com o banco de dados, é o primeiro passo para integração do gerenciador do banco de dados com o Python. O software que estamos utilizando é o MySQL.

Exemplo:

```
from sqlalchemy import create_engine

user = "root"
senha = "123456789"
banco_de_dados = "edufinance_financeiro"
host = "localhost"
porta = 3306

conexao = create_engine(url=f"mysql+pymysql://{user}:{senha}@{host}:{porta}/{banco_de_dados}",
echo=True)
```

2. Criando uma tabela dentro do banco de dados

Para criar uma tabela dentro do banco de dados, utilizando o Python, será um pouco diferente. Ao invés de utilizar apenas uma função, propriamente dita, utilizaremos uma classe por orientação a objeto (calma que veremos a frente).

Note que é necessário a criação de uma conexão antes de tentar integrar o método ao banco de dados.

No exemplo abaixo, definimos o seguinte código para criação da tabela.

Exemplo:

```python
from sqlalchemy import create_engine
from sqlalchemy orm import declarative_base
from sqlalchemy import Column, Integer, String, Date

conexao = create_engine(url=f"mysql+pymysql://root:123456789@localhost:3306/edufinance_financeiro", echo=True)

Banco_de_dados = declarative_base()

class User(Banco_de_dados):

    __tablename__ = 'nome_da_tabela'

    coluna_1 = Column(Integer, primary_key=True, autoincrement=True)
    coluna_2 = Column(String(50))
    coluna_3 = Column(Integer)
    coluna_4 = Column(Date)

Banco_de_dados.metadata.create_all(conexao)
```

Onde:

tablename:

É o nome da tabela no formato string

coluna_x:

É o nome da coluna que recebe o método columns() com os seguintes parâmetros:

type => O tipo da formatação da coluna. Pode ser integer, string(n° de caracteres), date

primary_key(Apenas para coluna principal) => Se vai ser uma chave única sem repetição. Recebe um booleano com True ou False.

autoincrement(Apenas para coluna principal) => Vai definir se a chave primária será definida automaticamente. Recebe um booleano com True ou False.

3. DataFrame.to_sql(name, con, schema = None, if_exists = "fail", index = True, index_label = None, chunksize = None, dtype = None, method = None)

Esse método é utilizado para enviar informações ao SQL. Caso não exista a tabela dentro do gerenciador, ele criará uma nova. Esse método tem que ser utilizado em conjunto com a conexão ao SQL.

Parâmetros:

Só recebe os parâmetros name e con como obrigatórios.

name:

Esse parâmetro vai definir o nome da tabela para onde serão enviadas as informações. Caso não exista a tabela, ele criará uma nova.

Esse parâmetro **é obrigatório**. Recebe o nome da tabela no formato string.

con:

Esse parâmetro vai definir qual será o conector utilizado para integrar o pandas ao sql.

Esse parâmetro **é obrigatório**. Recebe a variável na qual o conector foi atribuído.

schema:

Esse parâmetro vai definir para qual banco de dados a tabela vai ser enviada. Por padrão, o pandas define "schema = None", ou seja, o pandas vai definir um banco de dados padrão.

Esse parâmetro **é opcional**. Recebe o nome do banco de dados no formato string.

if_exists:

Esse parâmetro vai definir o que fazer quando a tabela enviada já existir. Por padrão,o pandas define "if_existe = 'fail' ", ou seja, o envio retornará um erro se a tabela já existir.

Esse parâmetro **é opcional**. Recebe os comandos já pré-definidos no formato string. Os formatos são:

fail = retornará um erro caso o nome da tabela já exista.

replace = substitui a tabela caso o nome da tabela já exista

append = adiciona informações à tabela caso o nome da tabela já exista

index:

Esse parâmetro vai definir se o index entrará na tabela como uma coluna ou não. Por padrão, o pandas define o "index = True", então ele adiciona o index como uma coluna dentro do SQL.

Esse parâmetro só pode ser usado em caso de uma nova tabela ou substituição de uma tabela antiga

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

index_label:

Esse parâmetro é utilizado em conjunto com o parâmetro “index”. Ele que vai definir o nome da nova coluna que antes era um index. Por padrão, o pandas define o nome da nova coluna como o mesmo nome do index.

Esse parâmetro **é opciona**l. Recebe o nome da nova coluna no formato string.

chuncksize:

Esse parâmetro vai definir quantas linhas serão adicionadas ao banco de dados a cada vez. Por padrão, o pandas define “chuncksize = 1”, ou seja, ele vai adicionar uma linha de cada vez no gerenciador de banco de dados.

Esse parâmetro **é opcional**. Recebe o valor da quantidade de linhas em integer.

dtype:

Esse parâmetro vai definir o tipo de cada coluna. Por padrão, o pandas define o tipo integer para números e o tipo text para os outros.

Esse parâmetro **é opcional**. Recebe os nomes das colunas e o tipo das colunas no formato string dentro de um dicionário.

method:

O parâmetro “method” na função to_sql do pandas determina o método usado para escrever os dados no banco de dados. Os valores possíveis são ‘multi’ para executar várias instruções SQL em paralelo, ‘infer’ para tentar inferir o melhor método a partir do tipo de dados e ‘None’ para usar o método padrão.

Esse parâmetro **é opcional**. Recebe o método utilizado no formato string.

No exemplo abaixo estamos enviando um DataFrame para o gerenciador de banco de dados. Adicionando, caso exista, as informações na tabela “price”. Por fim, iremos enviar as informações sem os rótulos das colunas, ou seja, tirando o index.

Exemplo:

```
import pandas as pd
from sqlalchemy import create_engine

conexao = create_engine(url=f"mysql+pymysql://root:123456789@localhost:3306/edufinance_financeiro")

nome_tabela = 'price'
df_teste = pd.DataFrame({"empresas": ['WEGE3', 'PCAR4', 'PETR4'],
                         "cotacao": [20, 30, 40]}, index = pd.date_range("2022-01-01", periods = 3))

df_teste.to_sql(nome_tabela, conexao, index=False, if_exists='append')
```

4. pandas.read_sql(sql, con, index_col = None, coerce_float = True, parse_dates = None, columns = None, chunksize = None)

Esse método é utilizado para ler informações do SQL, caso não exista a tabela dentro do gerenciador, ele retornará um erro. Esse método tem que ser utilizado em conjunto com a conexão ao SQL.

Parâmetros:

Só recebe os parâmetros sql e con como obrigatórios.

sql:

Esse parâmetro vai definir o nome da tabela que terá as informações lidas ou os códigos em sql a serem interpretados. Caso não exista a tabela, ele retornará um erro.

Esse parâmetro **é obrigatório**. Recebe o nome da tabela no formato string ou os comandos em sql no formato string.

con:

Esse parâmetro vai definir qual será o conector utilizado para integrar o pandas ao sql.

Esse parâmetro **é obrigatório**. Recebe a variável na qual o conector foi atribuído.

index_col:

Esse parâmetro vai definir quais as colunas da tabela do SQL serão definidas como index. Se for mais de uma, irá retornar um DataFrame multindex. Por padrão, o pandas define "index_col = False",ou seja, não definirá um index.

Esse parâmetro **é opcional**. Recebe o nome da coluna no formato string ou os nomes das colunas dentro de uma lista no formato string.

coerce_float:

Esse parâmetro vai definir que números que não estão no formato float, tentarão ser passados para esse formato. Por padrão, o pandas considera "coerce_float = True", ou seja, tentarão transformar esses objects em float.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

parse_dates:

Esse parâmetro vai definir quais colunas selecionadas terão seus tipos convertidos para o formato datetime. Por padrão, o pandas define "parse_dates = None", ou seja, nenhuma coluna será definida

Esse parâmetro **é opcional**. Recebe o nome da coluna no formato string ou os nomes das colunas dentro de uma lista no formato string.

columns:

Esse parâmetro vai definir quais colunas do sql serão selecionadas para o DataFrame. Por padrão, o pandas define “columns = None”, ou seja, serão selecionadas todas as colunas.

Esse parâmetro **é opcional**. Recebe o nome da coluna no formato string ou os nomes das colunas dentro de uma lista no formato string.

chunksize:

Esse parâmetro vai definir a quantidade de linhas que serão lidas de uma vez. O objetivo é diminuir a utilização de memória do computador que esse método pode gerar. O pandas define “chunksize = None”, ou seja, não define o limite, lê tudo de uma vez.

Esse parâmetro **é opcional**. Recebe a quantidade de linhas a serem lidas no formato string.

No exemplo abaixo estamos extraindo um DataFrame e atribuindo à variável “df”.

Exemplo:

```python
import pandas as pd
from sqlalchemy import create_engine

conexao = create_engine(url=f"mysql+pymysql://root:123456789@localhost:3306/edufinance_financeiro")

nome_tabela = 'price'
comando_sql = f"SELECT * FROM {nome_tabela};" #tem que saber SQL
df = pd.read_sql(comando_sql,con=conexao)

print(df)
```

Resposta:

```
  empresas  cotacao
0    WEGE3       20
1    PCAR4       30
2    PETR4       40
```

Mundo 16

Neste mundo abordaremos formas de tratar os dados que são importados no Python.

1. DataFrame.dropna(axis = 0, how = None, thresh = None, subset = None, inplace = False)

Esse método é utilizado para descartar valores “NaN” dentro de DataFrames.

Parâmetros:

Nenhum parâmetro é obrigatório.

axis:

Esse parâmetro vai definir em qual direção ocorrerá a exclusão, se será no sentido horizontal, da esquerda para direita, ou no vertical, de cima para baixo. Por padrão, o pandas define “axis=0”, ou seja, significa que será excluído verticalmente.

Esse parâmetro **é opcional**. Pode ser 0, que significa que a exclusão ocorrerá no sentido vertical, de cima para baixo. Ou pode ser 1, que significa que a exclusão ocorrerá horizontalmente, da esquerda para direita.

how:

Esse parâmetro vai definir quais são as condições de exclusão. Por padrão, o pandas define “how = any”, ou seja, em caso de qualquer item , da coluna ou linha, ser NaN, ele excluirá a linha, ou coluna, inteira.

Esse parâmetro **é opcional**. Pode receber strings pré-definidas pelo pandas, que são:

any => Se qualquer item NaN está presente na linha, ou na coluna, ela será excluída.

all => Se TODOS os itens da linha, ou coluna, forem NaN, ela será excluída.

tresh:

Esse parâmetro vai definir o número máximo de valores “NaN” que uma coluna, ou linha, podem ter. Por padrão, o pandas considera “tresh = 0”, ou seja, tolerância zero.

Esse parâmetro **é opcional**. Pode receber a quantidade mínima de valores “NaN” no formato integer.

subset:

Esse parâmetro vai definir quais linhas, ou colunas, serão consideradas na exclusão dos valores NaN. Você vai escolher uma coluna para ser analisada, e o pandas só considerará essa coluna. Ou seja, se tiverem valores NaN fora da coluna escolhida para análise, eles continuarão lá, pois estão fora do intervalo de análise.

Esse parâmetro **é opcional**. Pode receber o nome das colunas no formato string dentro de uma lista.

inplace:

Esse parâmetro vai definir se as modificações serão feitas diretamente no DataFrame ou não. Esse parâmetro pode ser interpretado por uma forma de salvar as modificações diretamente no DataFrame. Por padrão, o pandas considera “inplace=False”, ou seja, ele não salvará as modificações caso você não atribua a um objeto.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

No exemplo abaixo estamos excluindo as linhas das colunas, empresa e cotacao, que possuem valores “NaN” ao mesmo tempo.

Exemplo:

```
import pandas as pd

dados_teste = pd.DataFrame({'empresa': ['Weg', 'Petrobras', 'Vale', pd.NA],
                            'cotacao': [20, 50, pd.NA, pd.NA],
                            'volume': [3000, pd.NA, pd.NA ,pd.NA]})

dados_teste = dados_teste.dropna(subset = ['empresa', 'cotacao'], how='all')

print(dados_teste)
```

Resposta:

```
  empresa cotacao volume
0        Weg      20   3000
1  Petrobras      50   <NA>
2       Vale    <NA>   <NA>
```

2. DataFrame.fillna(value = None, method = None, axis = None, inplace = False, limit = None, downcast = None)

Esse método é utilizado para substituir valores “NaN” ,dentro de DataFrames, por outros valores.

Parâmetros:

O parâmetro “value” é obrigatório.

value:

Esse parâmetro vai definir qual será o valor que substituirá os valores “NaN”

Esse parâmetro **é obrigatório, caso não defina o method**. Pode ser uma palavra no formato string ou um número no formato integer.

method:

Esse parâmetro vai definir como serão feitas as substituições dos valores.

Esse parâmetro **é obrigatório, caso não defina o value**. Pode receber strings pré-definidas pelo pandas, que são:

pad / ffill => Ele repetirá o último valor de cima para baixo para os valores Nan

backfill / bfill => Ele repetirá o último valor de baixo para cima para os valores Nan

axis:

Esse parâmetro vai definir em qual direção ocorrerá a substituição dos itens, se será no sentido horizontal, da esquerda para direita, ou no vertical, de cima para baixo. Por padrão, o pandas define "axis=0", ou seja, significa que a substituição dos itens ocorrerá verticalmente

Esse parâmetro **é opcional**. Pode ser 0, que significa que a substituição dos itens ocorrerá no sentido vertical, de cima para baixo. Ou pode ser 1, que significa que a substituição dos itens ocorrerá horizontalmente, da esquerda para direita.

inplace:

Se deve fazer a modificação direto no DataFrame ou não. Esse parâmetro pode ser interpretado por uma forma de salvar as modificações diretamente no DataFrame. Por padrão o pandas considera "inplace=False", ou seja, ele não salvará as modificações caso você não atribua a um objeto.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

limit:

Esse parâmetro é utilizado em conjunto com **method**. Vai definir um limite de substituições. Por padrão, o pandas define "limit = None", ou seja, não é definido um limite. Caso seja definido igual 2, o Python irá preencher até dois dados NaN após a última informação.

Esse parâmetro **é opcional**. Recebe um valor limite no formato integer.

downcast:

Esse parâmetro é utilizado para definir o formato dos valores substituídos. Por padrão, o pandas define "downcast = None", ou seja, ele definirá formato "object".

Esse parâmetro **é opcional**. Recebe os valores dentro de um dicionário com os valores a serem substituídos e o formato deles (float64 ou int64).

No exemplo abaixo estamos substituindo os valores “NaN” das colunas empresa, cotacao e volume respectivamente por “Nenhuma”, 0 e -10.

Exemplo:

```
import pandas as pd

dados_teste = pd.DataFrame({'empresa': ['Weg', 'Petrobras', 'Vale', pd.NA],
                            'cotacao': [20, 50, pd.NA, pd.NA],
                            'volume': [3000, pd.NA, pd.NA ,pd.NA]})

dados_teste = dados_teste.fillna({'empresa':'Nenhuma',
                                  'cotacao': 0,
                                  'volume': -10})

print(dados_teste)
```

Resposta:

```
    empresa  cotacao  volume
0       Weg       20    3000
1  Petrobras      50     -10
2      Vale        0     -10
3   Nenhuma        0     -10
```

Mundo 17

Neste mundo abordaremos formas de tratar os dados que são importados no Python.

1. pandas.unique(values)

Esse método recebe uma Series, ou a coluna de um DataFrame, com valores duplicados e retorna um vetor com elementos únicos.

Parâmetros:

O parâmetro “value” é obrigatório.

value:

Esse parâmetro vai definir o intervalo de valores que sofrerá a análise de itens repetidos. Esse método pode ser utilizado com duas sintaxes diferentes.

pd.unique(Series)

Series.unique()

Esse parâmetro **é obrigatório**. Recebe o intervalo a ser analisado no formato de uma Series.

Exemplo:

No caso abaixo, o método **.unique()** foi utilizado como forma de adquirir um vetor com valores únicos que, em seguida, foram utilizados em um for loop. Dessa forma, o loop irá percorrer apenas valores únicos.

```
import pandas as pd

df_cambio = pd.DataFrame({
                    "moeda": ['dolar/real', 'libra/real'] * 4,
                    "price": [5.24, 4.15, 5.26, 4.23, 5.55, 4.02, 5.76, 3.98],
                    "derivativo": list(["DOLFT3", "LIBFT3"]) * 4},
                  index = sorted(list(pd.date_range("1/5/2014", periods = 4)) * 2))

ativos = df_cambio['derivativo'].unique()
print(ativos)
```

Resposta:

```
['DOLFT3' 'LIBFT3']
```

2. DataFrame.drop_duplicates(subset = None, keep = ‚first', inplace = False, ignore_index = False)

Esse método recebe uma Series, ou a coluna de um DataFrame, com valores duplicados e retorna um vetor com elementos únicos.

Parâmetros:

Não existe parâmetro obrigatório

subset:

Esse parâmetro vai definir quais colunas considerar na hora da análise das duplicatas. Por padrão, o pandas define “subset = None”, ou seja, todas as colunas serão consideradas.

Esse parâmetro **é opcional**. Recebe o nome de uma coluna no formato string ou recebe o nome das colunas no formato string dentro de uma lista.

keep:

Esse parâmetro vai definir o que fazer com as duplicatas. Por padrão, o pandas define " keep = "first ", ou seja, ele vai manter a primeira aparição , de cima para baixo.

Esse parâmetro **é opcional**. Recebe métodos pré-definidos pelo pandas. São eles:

' first ' => Manter a primeira aparição, de cima para baixo

' last ' => Manter a última aparição, de cima para baixo

False => Um booleano, não mantém nenhuma duplicata

inplace:

Se deve fazer a modificação direto no DataFrame ou não. Esse parâmetro pode ser interpretado por uma forma de salvar as modificações diretamente no DataFrame. Por padrão o pandas considera "inplace=False", ou seja, ele não salvará as modificações caso você não atribua a um objeto.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

ignore_index:

Esse parâmetro vai definir se o index pré-estabelecido vai ser ignorado ou não. Por padrão, o pandas define "ignore_index = False", ou seja, ele manterá o index de cada linha. Caso, "ignore_index = True", o index antigo seria excluído e substituído por um novo com a contagem das linhas começando por 0.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

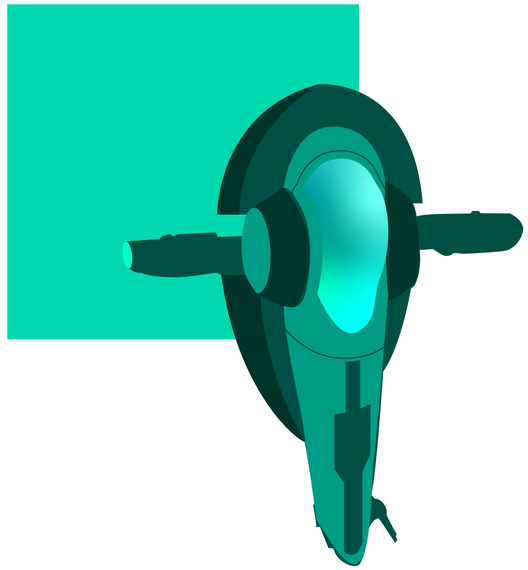
No caso abaixo, o DataFrame é organizado de acordo com a coluna "Volume". O objetivo é organizar do maior para o menor, de cima para baixo, para excluir os valores duplicados com menor liquidez (menor volume) e manter apenas o código de negociação mais líquido na tabela. Lembrando que a função drop_duplicate(), por padrão, mantém os primeiros itens.

Exemplo:

```
import pandas as pd

dados_mercado = pd.DataFrame({'Nome': ['Lojas Quero Quero', 'Alpargatas', 'Alpargatas', 'Vale', 'C&A'],
'Codigos': ['LJQQ3', 'ALPA3', 'ALPA4', 'VALE3', 'CEAB3'],
'Cotacoes': [20, 25, 30, 21, 23],
'Volume': [1000, 2000, 3500, 10000, 5000]},
index=['um','dois','tres','quatro','cinco'])

#ele vai ver o primeiro, quando ver o segundo vai sumir da lista.

dados_mercado = dados_mercado.sort_values(by = 'Volume', ascending = False).drop_duplicates(subset = 'Nome')

print(dados_mercado)
```

Resposta:

```
      Nome Codigos  Cotacoes  Volume
quatro                 Vale    VALE3        21   10000
cinco                   C&A    CEAB3        23    5000
tres             Alpargatas    ALPA4        30    3500
um      Lojas Quero Quero    LJQQ3        20    1000
```



Mundo 18

Neste mundo abordaremos formas de tratar os dados que são importados no Python.

1. DataFrame.astype(dtype, copy = True, errors = (“raise”)

Esse método recebe uma Series, ou a coluna de um DataFrame, e muda o tipo dela inteira.

Parâmetros:

O parâmetro “dtype” é obrigatório.

dtype:

Esse parâmetro vai definir o novo tipo da coluna. Esse tipo pode ser um tipo padrão do Python ou pode ser um tipo do NumPy.

Esse parâmetro **é opcional**. Recebe o tipo no formato string, que são:

Object ou strings => Usado para texto

np.Int64 ou integer => Usado para números inteiros

np.float64 ou float => Usado para números decimais

bool => Usado para números booleanos

np.Datetime64 => Usado para data e hora (Não separadamente)

copy:

Esse conceito é mais complexo. Ao definir copy=False, estamos definindo que as mudanças se propagarão entre os objetos. Isso quer dizer que se eu mudar uma Series criada por um vetor, este vetor mudará também.

obs: Isso só se aplica se tivermos criado uma Series utilizando **vetores (criados pela biblioteca NumPy)**.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

errors:

Esse parâmetro vai definir o que fazer em caso de erro. Por padrão, o pandas define “errors = ‘raise’ “, ou seja, em caso de erro ele retornará um erro.

Esse parâmetro **é opcional** e recebe dois métodos pré-definidos. Que são:

raise => retornar um erro em caso de erro

ignore => Em caso de erro, ignorar o erro e manter os valores originais (para cada item)

No caso abaixo é extraído um DataFrame do Yahoo Finance. Deste DataFrame modificamos os formatos das colunas: “High” e “Close”

Exemplo:

```
import pandas as pd
from pandas_datareader import data as pdr

dados_ambev = pdr.get_data_yahoo("ABEV3.SA")

dados_ambev['High'] = dados_ambev['High'].astype(int)
dados_ambev['Close'] = dados_ambev['Close'].astype(str)

print(dados_ambev)
```

Resposta:

```
High        Low       Open              Close      Volume  Adj Close
Date
2017-10-16    22  21.760000  22.250000  21.920000076293945  13271300.0  19.015736
2017-10-17    21  21.639999  21.920000               21.75   7051900.0  18.868258
2017-10-18    21  21.680000  21.799999  21.780000686645508  14515700.0  18.894283
2017-10-19    21  21.459999  21.700001  21.829999923706055   7696100.0  18.937660
2017-10-20    21  21.459999  21.830000  21.530000686645508  11901300.0  18.677404
...          ...        ...        ...                 ...         ...        ...
2022-10-07    16  15.300000  15.960000  15.420000076293945  40213400.0  15.420000
2022-10-10    15  15.060000  15.460000  15.199999809265137  29475700.0  15.200000
2022-10-11    15  14.940000  15.180000  14.989999771118164  30204200.0  14.990000
2022-10-13    15  14.700000  14.820000  15.010000228881836  25868700.0  15.010000
2022-10-14    15  14.850000  15.020000  15.020000457763672   7989600.0  15.020000
```

2. DataFrame.to_datetime(arg, errors = “raise”, dayfirst = False, yearfirst = False, utc = False, format = None, exact = True, unit = None , infer_datetime_format = False, origin = “unix”, cache = True)

Esse método recebe uma Series ou a coluna de um DataFrame e muda o tipo dela inteira para o formato datetime.

Parâmetros:

arg:

Esse parâmetro vai definir o objeto a ser transformado em datetime.

Esse parâmetro **é obrigatório**. Pode receber: integer, float, string, datetime, lista, tupla, Series e DataFrame.

errors:

Esse parâmetro vai definir o que fazer em caso de erro. Por padrão, o pandas define “errors = ‘raise’ ”, ou seja, em caso de erro ele retornará um erro.

Esse parâmetro **é opcional** e recebe dois métodos pré-definidos.

Que são:

raise => retornar um erro em caso de erro

coerce => Em caso de erro, retornará um NaT (para cada item)

ignore => Em caso de erro,Ignorar o erro, e manter os valores originais (para cada item)

dayfirst:

Esse parâmetro é utilizado para informar se a data passada para o pandas está na formatação de data com o **dia** em primeiro lugar.

Uma data que está no formato "10/11/12" ficará no formato 2010-11-12.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

yearfirst:

Esse parâmetro é utilizado para informar se a data passada para o pandas, está na formatação de data com o **ano** em primeiro lugar.

Data está assim: "10/11/12" => Vai ficar assim: 2012-11-10.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

utc:

Esse parâmetro vai definir se uma data será retornada com fuso horário ou não. Por padrão, o pandas considera "utc = False", ou seja, ele não considerará o fuso horário.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

format:

Esse parâmetro é utilizado para informar a formatação em que os dados estão sendo passados. Por padrão, o pandas define "format = None", ou seja, o pandas tentará reconhecer o formato, porém é sempre melhor informar o formato que está sendo utilizado.

Esse parâmetro **é opcional** e recebe a formatação da data em formato de string.

exact:

Esse parâmetro funciona da seguinte maneira: ele especifica se a data / hora deve ser considerada exata ou aproximada. Por padrão, o pandas define "exact = True", ou seja, a data / hora vai ser exata (ex: 2018-01-01 12:00:00). Caso "exact = False", ele definiria a hora quebrada (ex: 2018-01-01 12:00:11)

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

unit:

Esse parâmetro informa ao pandas qual a unidade que os dados estão sendo passados (no caso de integer ou float). Por padrão, o pandas define “unit = ‘ms’ “, ou seja, ele lerá os dados em milissegundos.

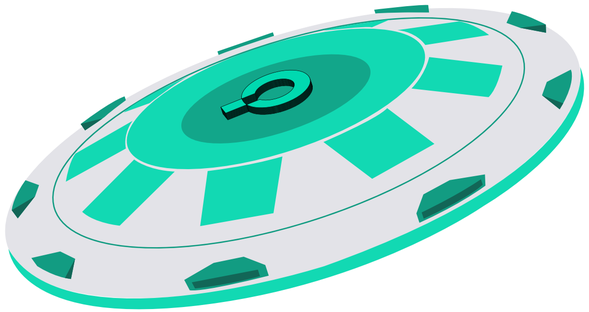
Esse parâmetro **é opcional** e recebe formatos pré-definidos como string. São eles:

d = dias

s = segundos

ms = milissegundos

us => microssegundos



ns => nanossegundos

infer_datetime_format:

Esse parâmetro vai definir que o pandas analisará a amostra de dados passada e tentará definir um formato automaticamente, se “format = None”. Por padrão, o pandas considera “infer_datetime_format = False”, ou seja, ele não tentará definir um formato automaticamente.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

origin:

Esse parâmetro vai definir a data referência a ser considerada como origem. Por padrão, o pandas considera “origin = ‘unix’ “, ou seja, ele considera a contagem das datas a partir de 1970-01-01. Se, por exemplo, um valor de data e hora for fornecido como 100, isso será interpretado como 100 segundos após a data e hora de origem.

Esse parâmetro **é opcional** e pode receber “unix” e “julian” como formato pré-definido.

cache:

Esse conceito é um pouco mais complexo.

Quando você chama o método to_datetime() pela primeira vez, ele analisa a string de data e a converte para um objeto datetime. Esse objeto é armazenado em um cache interno.

Quando você chama o método to_datetime() pela segunda vez com a mesma string de data, o objeto datetime é recuperado do cache, o que é mais rápido do que analisar e converter a string novamente.

Isso é útil se você estiver lidando com uma grande quantidade de dados e precisar converter muitas strings de data para objetos datetime.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

No caso abaixo, é criado um DataFrame de exemplo. Passamos a coluna “datas” deste DataFrame para o formato datetime, informando o formato que está sendo passada a data. Como é uma data, podemos pegar algumas informações como: dia, mês e ano. Neste caso, decidimos pegar o ano.

Exemplo:

```
import pandas as pd

df_data_teste = pd.DataFrame({'datas': ["3/12/2020", "4/12/2020"],
                              'cotacoes': [20, 21]})
df_data_teste['datas'] = pd.to_datetime(df_data_teste['datas'], format = "%d/%m/%Y")
print(df_data_teste['datas'].iloc[0].year)
```

Resposta:

```
2020
```

3. DataFrame.replace(to_replace = None, value = _NoDefault.no_default, inplace = False, limit = None, regex = False, method = _NoDefault.no_default)

Esse método recebe uma Series, coluna de um DataFrame ou DataFrame e substitui um certo valor.

Parâmetros:

to_replace:

Esse parâmetro é usado para especificar o que, e pelo que, devem ser substituídos os valores na matriz de substituição.

Esse parâmetro **é obrigatório**. Pode receber: uma lista, um ndarray, um dicionário ou um objeto callable. Se for uma lista, ndarray ou um dicionário, os valores devem ser do mesmo tipo que os valores da matriz de substituição.

value:

Esse parâmetro é utilizado em conjunto com o "**to_replace**".

Esse parâmetro **é obrigatório**. Pode receber: uma lista, um ndarray, um dicionário ou um objeto callable. Se for uma lista,ndarray ou um dicionário, os valores devem ser do mesmo tipo que os valores da matriz de substituição.

inplace:

Se deve fazer a modificação direto no DataFrame ou não. Esse parâmetro pode ser interpretado por uma forma de salvar as modificações diretamente no DataFrame. Por padrão o pandas considera "inplace=False", ou seja, ele não salvará as modificações caso você não atribua a um objeto.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

limit:

Esse parâmetro vai definir um limite de substituições. Por padrão, o pandas define “limit = None”, ou seja, não é definido um limite.

Esse parâmetro **é opcional**. Recebe um valor limite no formato integer.

regex:

Esse parâmetro vai determinar que a busca será feita utilizando expressões regulares.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

method:

Esse parâmetro vai definir como serão feitas as substituições dos valores.

Esse parâmetro **é obrigatório, caso não defina o value e o to_replace**. Pode receber strings pré-definidas pelo pandas, que são:

pad / ffill => Ele repetirá o último valor de cima para baixo para os valores Nan

bfill => Ele repetirá o último valor de baixo para cima para os valores Nan

No caso abaixo, estamos importando um DataFrame com os dias dos dividendos da ação “ABEV3.SA”. E dentro desse DataFrame com os dividendos, estamos substituindo todos os valores “NaN” por “0”.

Exemplo:

```
from pandas_datareader import data as pdr
import numpy as np

dados_ambev = pdr.get_data_yahoo("ABEV3.SA", get_actions=True)['Dividends']
print(dados_ambev)

dados_ambev.replace(np.NAN, 0, inplace=True)
print(dados_ambev)
```

Antes do replace:

```
Date
2017-11-09   NaN
2017-11-10   NaN
2017-11-13   NaN
2017-11-14   NaN
2017-11-15   NaN
              ..
2022-11-01   NaN
2022-11-03   NaN
2022-11-04   NaN
2022-11-07   NaN
2022-11-08   NaN
Name: Dividends, Length: 1240, dtype: float64
```

Depois do replace:

```
Date
2017-11-09    0.0
2017-11-10    0.0
2017-11-13    0.0
2017-11-14    0.0
2017-11-15    0.0
             ...
2022-11-01    0.0
2022-11-03    0.0
2022-11-04    0.0
2022-11-07    0.0
2022-11-08    0.0
Name: Dividends, Length: 1240, dtype: float64
```

Mundo 19

Neste mundo abordaremos formas de alterar dados dentro dos DataFrames utilizando a função .map().

1. Series.map(fun , iter)

O método map() é usado para aplicar uma função a cada elemento de um iterável e retornar os resultados em um novo objeto. Essa função aplicada, muitas das vezes, é a função lambda, criada por você.

Parâmetros:

fun:

Esse parâmetro é a função que vai ser utilizada.

Esse parâmetro **é obrigatório**. Pode ser uma função lambda criada para um específico caso ou pode ser funções nativas do Python.

iter:

Esse parâmetro é o objeto a ser percorrido.

Esse parâmetro **é obrigatório**. Pode receber uma Series, Coluna de DataFrame, Lista, Tupla e etc...

No caso abaixo estamos importando as cotações da "ABEV3, MGLU3, WEGE3, VALE3" do Yahoo Finance. Logo em seguida, é calculado o retorno com .pct_change(), excluído os dados faltantes com .dropna() e calculado a média de retornos com mean(). Após ter o DataFrame com o retorno médio de cada empresa, é utilizado a função map, que tem como seu parâmetro uma função lambda que transforma o retorno mensal em retorno anual.

Exemplo:

```python
import pandas as pd
from pandas_datareader import data as pdr

dados_acoes = pdr.get_data_yahoo(["ABEV3.SA", "MGLU3.SA", "WEGE3.SA", "VALE3.SA"])['Adj Close']
media_retornos_diarios = dados_acoes.pct_change().dropna().mean()
dados_anuais = media_retornos_diarios.map(lambda x: (1 + x) ** 252 - 1)

print(dados_anuais)
```

Resposta:

```
Symbols
ABEV3.SA   -0.002562
MGLU3.SA    0.404994
WEGE3.SA    0.426925
VALE3.SA    0.360302
dtype: float64
```

2. Series.to_frame(name = None)

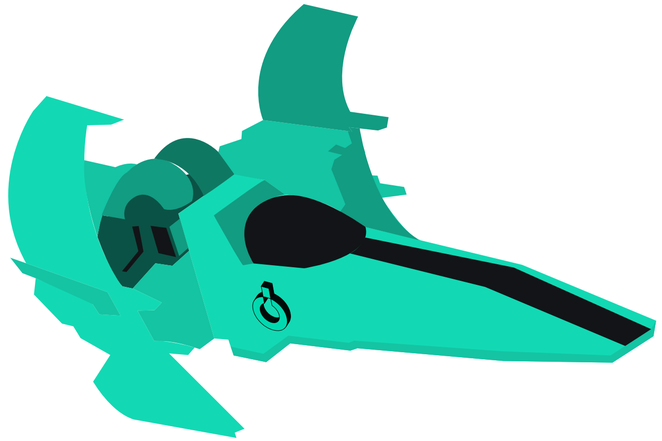
O método ".to_frame" transforma um objeto em um DataFrame.

Parâmetros:

name:

Esse parâmetro é a Series que vai ser transformada em DataFrame.

Esse parâmetro **é obrigatório**. Recebe uma Series



Mundo 20

Neste mundo abordaremos formas de alterar dados dentro dos DataFrames.

1. pandas.cut(x , bins, right = True, labels = None, retbins = False, precision = 3, include_lowest = False, duplicates = "raise", ordered = True)

O método cut() separa um conjunto de dados em vários pedaços e, em seguida, divide cada um desses pedaços em grupos definidos pelo argumento "bins".

Parâmetros:

x:

Esse parâmetro é o vetor com os valores que serão divididos em grupos.

Esse parâmetro **é obrigatório**. Recebe um vetor de uma dimensão com os valores.

bins:

Esse parâmetro vai definir o intervalo dos grupos.

Esse parâmetro **é obrigatório**. Pode receber:

integer => Neste caso vai dividir em intervalos de iguais tamanhos

lista => Neste caso vai dividir em intervalos de acordo com os intervalos da lista (não necessariamente de tamanhos iguais)

right:

Esse parâmetro vai definir se os valores vão incluir os números no intervalo à direita ou não, ou seja, se o intervalo a direita vai ser aberto ou fechado. Por padrão, o pandas define "right = True", ou seja, o intervalo à direita está incluso.

Esse método não é utilizado quando "bins" é uma lista.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

labels:

Esse parâmetro é utilizado quando "bins = integer". Ele vai definir como serão nomeados os intervalos e, por padrão, o pandas define os últimos números do intervalo como seu nome.

O tamanho da lista deve ter o mesmo tamanho da quantidade de intervalos

Esse parâmetro **é opcional** e recebe uma lista com os nomes dos rótulos no formato string.

retbins:

Esse parâmetro vai definir se na resposta será retornado os valores dos intervalos. Por padrão, o pandas define "retbins = False", ou seja, não retornará o valor do intervalo na resposta.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

precision:

Esse parâmetro vai definir a quantidade de casas decimais utilizadas nos intervalos. Por padrão, o pandas define que o número de casas decimais será a mesma já definida nos números.

Esse parâmetro **é opcional** e recebe o números de casas decimais no formato integer.

include_lowest:

Esse parâmetro vai definir se os valores vão incluir os números no intervalo à esquerda ou não, ou seja, se o intervalo à esquerda vai ser aberto ou fechado. Por padrão, o pandas define "right = True", ou seja, o intervalo à esquerda está incluso.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

duplicates:

Esse parâmetro vai definir o que fazer com os intervalos duplicados, se retornará um erro, ou se serão excluídos. Por padrão, o pandas define "duplicates = 'raise' ", ou seja, elas serão mantidas.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

No caso abaixo, estamos criando um DataFrame com o nome da empresa, nome do ceo e idade. Depois criamos uma lista que vai servir como separação de 5 intervalos. Utilizando o método "cut", a gente separa as idades nos intervalos criados.

Exemplo:

```
import pandas as pd

df_ceos_empresas = pd.DataFrame({"Empresa": ['Weg', 'Vale', 'Petrobras', 'Renner',
                                             'Assai', 'Nubank', 'Inter', 'BTG'],
                                 'Nome_ceo': ['Pedro', 'Nelson', 'Lucas', 'Stefany',
                                              'Mateus', 'Aline', 'Raquel', 'Paloma'],
                                 'Idade': [36, 45, 55, 60, 62, 23, 31, 32]})

faixa_etaria = [0, 18, 25, 35, 50, 65]
ceos_por_idade = pd.cut(df_ceos_empresas['Idade'], faixa_etaria)

print(ceos_por_idade)
```

Resposta:

```
0    (35, 50]
1    (35, 50]
2    (50, 65]
3    (50, 65]
4    (50, 65]
5    (18, 25]
6    (25, 35]
7    (25, 35]
Name: Idade, dtype: category
Categories (5, interval[int64, right]): [(0, 18] < (18, 25] <
(25, 35] < (35, 50] < (50, 65]]
```

2. pandas.value_counts(normalize = False, sort = True, ascending = False, bins = None, dropna = True)

O método value_counts() retorna uma Series com contagem de valores exclusivos.

Parâmetros:

normalize:

Este parâmetro define na função "values_count" do pandas a divisão dos valores da contagem pelo número total de linhas.

Por exemplo, se o seu DataFrame tem 100 linhas e a função "values_count" mostra que há 10 valores "1", o parâmetro "normalized" vai dividir esse valor (10) pelo número total de linhas (100), resultando em 0,1. Dessa forma, os dados serão normalizados entre 0 e 1.

Por padrão, o pandas define, "normalize = False".

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

sort:

Este parâmetro define se os valores serão classificados em ordem decrescente ou crescente. Por padrão, o pandas define "sort = True", ou seja, ele será classificado em ordem decrescente.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

ascending:

Este parâmetro define se os valores serão classificados em ordem crescente. Por padrão, o pandas define "ascending = False", ou seja, ele será classificado em ordem decrescente.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

bins:

Este parâmetro vai definir se os valores serão agrupados em compartimentos ou não e só funciona para valores numéricos. Por padrão, o pandas define o "bins = False".

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

dropna:

Este parâmetro vai definir se os valores que são nulos, serão excluídos do resultado final. Por padrão, o pandas define "dropna = True", ou seja, ele vai excluir os valores nulos.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

No caso abaixo, estamos criando um DataFrame com o nome da empresa, nome do ceo e idade. Depois criamos uma lista que vai servir como separação de 5 intervalos. Utilizando o método "cut", a gente separa as idades nos intervalos criados.

Utilizando o método "value_counts" ele irá contar o número de ocorrências em cada intervalo.

Exemplo:

```
import pandas as pd

df_ceos_empresas = pd.DataFrame({"Empresa": ['Weg', 'Vale', 'Petrobras', 'Renner',
                                             'Assai', 'Nubank', 'Inter', 'BTG'],
                                 'Nome_ceo': ['Pedro', 'Nelson', 'Lucas', 'Stefany',
                                              'Mateus', 'Aline', 'Raquel', 'Paloma'],
                                 'Idade': [36, 45, 55, 60, 62, 23, 31, 32]})

faixa_etaria = [0, 18, 25, 35, 50, 65]
ceos_por_idade = pd.cut(df_ceos_empresas['Idade'], faixa_etaria)
numero_de_amostras_por_categoria = pd.value_counts(ceos_por_idade)

print(numero_de_amostras_por_categoria)
```

Resposta:

```
(50, 65]    3
(25, 35]    2
(35, 50]    2
(18, 25]    1
(0, 18]     0
Name: Idade, dtype: int64
```

3. pandas.get_dummies(data, prefix = None, prefix_sep = "_", dummy_na = False, columns = None, sparse = False, drop_first = False, dtype = None)

O método get_dummies() retorna um DataFrame de 0 e 1, onde 0 é False e 1 é True, que indica a qual categoria cada dado pertence.

Parâmetros:

data:

Este parâmetro define os valores das categorias.

Esse parâmetro **é obrigatório** e recebe uma lista,uma Series, ou um DataFrame com os intervalos.

prefix:

Este parâmetro especifica um prefixo a ser adicionado aos nomes das colunas que são criadas quando o método é executado. Por padrão, o pandas define "prefix = None", ou seja, nenhum prefixo será definido. O nome da coluna nova se dará pelo valor + o nome da coluna antiga

Esse parâmetro **é opcional**. Pode receber o prefixo como uma string, como uma lista com strings dentro ou como um dicionário com strings.

prefix_sep:

Este parâmetro especifica um separador a ser usado, como nome da coluna, entre o nome da coluna e o valor da coluna, do antigo DataFrame. Por padrão, o pandas define "prefix = '_' ", ou seja, o separador será o '_'

Esse parâmetro **é opcional** e recebe o separador no formato string.

dummy_na:

Este parâmetro especifica que os valores NaN sejam considerados como um valor válido para criação de um novo dummie. Por padrão, o pandas define "dummy_na= False", ou seja, valores NaN serão ignorados

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

columns:

Este parâmetro especifica quais serão as colunas a serem consideradas, na hora da criação do novo DataFrame. Por padrão o pandas define "columns = 'None' ", ou seja, ela considerará TODAS as colunas.

Esse parâmetro **é opcional** e recebe uma lista com os nomes das colunas no formato string.

sparse:

Este parâmetro especifica se retornará uma matriz esparsa (formato de dados que ocupam menos espaço na memória do que uma matriz densa, porque muitos dos seus elementos são zero). Se o parâmetro for definido como True, a saída será uma matriz esparsa. Se o parâmetro for definido como False (padrão), a saída será uma matriz densa. Por padrão, o pandas define "sparse = False", ou seja, será uma matriz densa.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

drop_first:

Este parâmetro especifica que a primeira coluna deve ser descartada após a codificação. Por padrão, o pandas define "drop_first = False", ou seja, todas as colunas codificadas serão mantidas.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

dtype:

Este parâmetro especifica o tipo de dado que será retornado. Por padrão, o pandas define “dtype = ‘None’ “, ou seja, ele considerará o formato já definido.

Esse parâmetro **é opcional** e recebe os formatos (**integer** ou **strings**) como uma string.

No caso abaixo, estamos criando um DataFrame com o nome da empresa, nome do ceo e idade. Depois criamos uma lista que vai servir como separação de 5 intervalos. Utilizando o método “cut”, a gente separa as idades nos intervalos criados.

Utilizando o método “get_dummies” ele irá retornar um DataFrame com valores de 0 e 1.

Exemplo:

```
import pandas as pd

df_ceos_empresas = pd.DataFrame({"Empresa": ['Weg', 'Vale', 'Petrobras', 'Renner',
                                             'Assai', 'Nubank', 'Inter', 'BTG'],
                                 'Nome_ceo': ['Pedro', 'Nelson', 'Lucas', 'Stefany',
                                              'Mateus', 'Aline', 'Raquel', 'Paloma'],
                                 'Idade': [36, 45, 55, 60, 62, 23, 31, 32]})

faixa_etaria = [0, 18, 25, 35, 50, 65]
ceos_por_idade = pd.cut(df_ceos_empresas['Idade'], faixa_etaria)
numero_de_amostras_por_categoria = pd.get_dummies(ceos_por_idade)

print(numero_de_amostras_por_categoria)
```

Resposta:

```
   (0, 18]  (18, 25]  (25, 35]  (35, 50]  (50, 65]
0        0         0         0         1         0
1        0         0         0         1         0
2        0         0         0         0         1
3        0         0         0         0         1
4        0         0         0         0         1
5        0         1         0         0         0
6        0         0         1         0         0
7        0         0         1         0         0
```

Mundo 21

Neste mundo abordaremos formas de formatar seus dados para colocar em um Banco de Dados.

1. pandas.melt(frame, id_vars = None, value_vars = None, var_name = None, value_name = “value”, col_level = None, ignore_index = True)

O método melt() é utilizado para transformar uma tabela cumprida em uma tabela longa. Resumidamente, ele irá redimensionar o número de linhas e colunas. Muitas colunas se transformarão em linhas.

Parâmetros:

O método pode ser utilizado sem nenhum parâmetro definido, neste caso, o pandas tentará redimensionar na maneira que preferir.

frame:

Esse parâmetro é o DataFrame com os valores a serem redimensionados.

Esse parâmetro **é opcional**. Recebe um DataFrame.

id_vars:

Esse parâmetro vai definir os valores que serão utilizados como referência para redimensionar a tabela.

Esse parâmetro **é opcional**. Recebe uma lista, ou uma tuple com o nome das colunas no formato string.

value_vars:

Esse parâmetro vai definir as colunas que irão se tornar valores.

Esse parâmetro **é opcional**. Recebe uma lista, ou uma tuple com o nome das colunas no formato string.

var_name:

Esse parâmetro vai definir o nome da coluna das váriaveis. Por padrão, o pandas considera 'var_name = 'variable' ''

Esse parâmetro **é opcional**. Recebe uma lista, ou uma tuple com o nome das colunas no formato string.

col_level:

Esse parâmetro vai definir o nome da coluna dos valores. Por padrão, o pandas considera 'value_name = 'values' ''.

Esse parâmetro **é opcional**. Recebe uma lista, ou uma tuple com o nome das colunas no formato string.

ignore_index:

Esse parâmetro tem como objetivo fazer com que os índices sejam ignorados e uma nova sequência de índices seja criada a partir do zero. Por padrão, o pandas define "ignore_index = True", ou seja, ele criará uma nova sequência.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

No caso abaixo, estamos importando o preço ajustado, das empresas que selecionamos. Resetamos o index e aplicamos a função melt para redimensionar a tabela.

Exemplo:

```
import pandas as pd
from pandas_datareader import data as pdr

acoes = pdr.get_data_yahoo(['WEGE3.SA', 'VALE3.SA', 'PCAR3.SA', 'LREN3.SA',
                            'PETR4.SA', 'EQTL3.SA', 'EGIE3.SA', 'ELET3.SA'])['Adj Close']
print(acoes)

acoes = acoes.reset_index()
dados_no_melzinho = pd.melt(acoes, id_vars = 'Date', var_name = 'empresa', value_name = 'price')

print(dados_no_melzinho)
```

DataFrame Original:

```
Symbols      WEGE3.SA   VALE3.SA   PCAR3.SA   LREN3.SA   PETR4.SA   EQTL3.SA   EGIE3.SA   ELET3.SA
Date
2017-10-20   8.022429  22.801861  76.293755  30.575420   8.169018  11.028064  20.179123  17.225563
2017-10-23   7.750243  22.635775  76.293755  29.970547   8.158946  10.755589  19.919071  17.853825
2017-10-24   7.983036  23.410833  76.293755  30.741150   8.315073  10.913338  20.046329  17.965166
2017-10-25   7.968709  23.217066  76.293755  31.155451   8.420838  11.038820  20.029734  18.585474
2017-10-26   7.718007  22.608095  76.293755  30.094837   8.425874  10.934848  19.913540  18.665003
...               ...        ...        ...        ...        ...        ...        ...        ...
2022-10-13  33.470001  72.180000  20.230000  29.180000  33.939999  27.620001  38.279999  45.000000
2022-10-14  32.860001  69.830002  19.719999  28.540001  33.419998  27.280001  38.369999  43.900002
2022-10-17  33.610001  70.930000  20.129999  29.090000  33.389999  27.570000  38.689999  46.049999
2022-10-18  34.560001  71.959999  20.389999  29.469999  34.209999  28.020000  38.700001  47.020000
2022-10-19  34.599998  71.209999  20.330000  29.350000  34.970001  28.139999  38.610001  47.099998

[1241 rows x 8 columns]
```

DataFrame Modificado:

```
                    Date    empresa      price
0    2017-10-20  WEGE3.SA   8.022429
1    2017-10-23  WEGE3.SA   7.750243
2    2017-10-24  WEGE3.SA   7.983036
3    2017-10-25  WEGE3.SA   7.968709
4    2017-10-26  WEGE3.SA   7.718007
...         ...       ...        ...
9923 2022-10-13  ELET3.SA  45.000000
9924 2022-10-14  ELET3.SA  43.900002
9925 2022-10-17  ELET3.SA  46.049999
9926 2022-10-18  ELET3.SA  47.020000
9927 2022-10-19  ELET3.SA  47.099998

[9928 rows x 3 columns]
```

2. pandas.pivot_table(data, values = None, index = None, columns = None, aggfunc = "mean", fill_value = None, margins = False, dropna = True, sort = True)

O método pivot_table() é utilizado para transformar uma tabela longa em uma tabela cumprida. Resumidamente, ele irá redimensionar o número de linhas e colunas. Muitas linhas se tornarão colunas.

O pivot_table é a operação inversa do método melt.

Parâmetros:

O parâmetro data é o único obrigatório.

frame:

Esse parâmetro é o DataFrame com os valores a serem redimensionados.

Esse parâmetro **é obrigatório**. Recebe um DataFrame.

index:

Esse parâmetro vai definir qual será o index do novo DataFrame.

Esse parâmetro **é obrigatório**. Recebe o nome de uma coluna no formato string, ou uma lista com os nomes das colunas no formato string.

columns:

Esse parâmetro vai definir a coluna a ser analisada. Por padrão, o pandas define “columns = None”, ou seja, ele analisará todas as colunas.

Esse parâmetro **é obrigatório**. Recebe o nome de uma coluna no formato string, ou uma lista com os nomes das colunas no formato string.

aggfunc:

Esse parâmetro vai ser usado para aplicar uma função de agregação aos dados. Um exemplo seria a função ”sum”, que somaria os valores de todas as colunas selecionadas. Por padrão, o pandas define “aggfunc = None”, ou seja, nenhuma função será definida.

Esse parâmetro **é opcional**. Recebe o nome da função, que pode ser sum(), mean()...

formato string.

fill_value:

Esse parâmetro é usado para especificar um valor a ser usado para substituir valores ausentes. Por padrão, o pandas define “fill_value = None”, ou seja, os valores faltantes serão “NaN”

Esse parâmetro **é opcional**. Recebe o valor a ser considerado no formato string.

dropna:

Este parâmetro vai definir se os valores que são nulos, serão excluídos do resultado final. Por padrão, o pandas define “dropna = True”, ou seja, ele vai excluir os valores nulos.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

margins_name:

Esse parâmetro é usado para especificar um valor a ser usado para substituir valores ausentes. Por padrão, o pandas define “fill_value = None”, ou seja, os valores faltantes serão “NaN”.

Esse parâmetro **é opcional**. Recebe o valor a ser considerado no formato string.

sort:

Esse parâmetro vai definir se os resultados estarão ordenados ou não. Por padrão, o pandas considera “sort = True”, ou seja, ele ordenará em ordem crescente, caso “sort = False” ele apenas não ordenará.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

No caso abaixo, estamos criando um DataFrame com as ações da WEGE3, PETR4 e VALE3, e redimensionando esse DataFrame, fazendo com que as empresas presentes na coluna “empresas” se tornassem colunas próprias.

Exemplo:

```
import pandas as pd

dict = {'Empresas': ['WEGE3','PETR4','VALE3'],'Preço':[20.28, 29.81, 67.13]}
df_empresas = pd.DataFrame(dict, index=['2022-11-1','2022-11-1','2022-11-1'])
print(df_empresas)

dados_pivotados = pd.pivot_table(df_empresas, columns='Empresas', index= df_empresas.index)
print(dados_pivotados)
```

DataFrame antes do pivot_table():

```
            Empresas  Preço
2022-11-1     WEGE3  20.28
2022-11-1     PETR4  29.81
2022-11-1     VALE3  67.13
```

DataFrame após o pivot_table():

```
            Preço
Empresas    PETR4  VALE3  WEGE3
2022-11-1  29.81  67.13  20.28
```

3. pandas.pct_change(periods=1 , fill_method = "pad", limit = None, freq = None)

O método pct_change () calcula a variação percentual de um determinado valor em relação ao valor anterior. Muito utilizado para calcular os retornos dos ativos.

Parâmetros:

Não existe parâmetro obrigatório.

periods:

Esse parâmetro vai definir o intervalo que ocorrerá a variação percentual. Por padrão, o pandas define "periods = 1". Em um dataframe com o fechamento diário, o retorno seria uma variação diária. Com "periods = 7", seria uma variação semanal.

Esse parâmetro **é opcional**. Recebe o intervalo como integer.

fill_method:

Esse parâmetro vai definir por qual valor os valores faltantes serão substituídos. Por padrão, o pandas define " fill_method = 'pad' ", ou seja, ele substituirá pela última ocorrência.

Esse parâmetro **é opcional**. Recebe o valor, ou método, a ser substituído no formato string.

limit:

Esse parâmetro vai definir o número de NAs consecutivos a serem preenchidos antes de parar. Por padrão, o pandas define "limit = 'None' ", ou seja, não existe um limite.

Esse parâmetro **é opcional**. Recebe o intervalo como integer.

freq:

Esse parâmetro vai definir a frequência dos intervalos, se será mensal, anual, diário ou apenas dos dias úteis.

Esse parâmetro **é opcional**. Recebe a frequência no formato string entre aspas. Para visualização da tabela.

No caso abaixo, estamos importando o preço ajustado, da WEGE3, e aplicando o método pct_change() para conseguir o retorno diário da cotação ajustada.

Exemplo:

```
import pandas as pd
from pandas_datareader import data as pdr

acoes = pdr.get_data_yahoo(['WEGE3.SA'])['Adj Close']

print(acoes.pct_change())
```

Resposta:

```
Symbols     WEGE3.SA
Date
2017-10-23       NaN
2017-10-24  0.030037
2017-10-25 -0.001794
2017-10-26 -0.031461
2017-10-27  0.001856
...              ...
2022-10-14 -0.018225
2022-10-17  0.022824
2022-10-18  0.028265
2022-10-19  0.002604
2022-10-20 -0.006638

[1241 rows x 1 columns]
```

4. pandas.droplevel(level, axis = 0)

O método droplevel() deleta níveis de index ou colunas.

Parâmetros:

O parâmetro level é o único obrigatório.

level:

Esse parâmetro remove o nível especificado de um índice hierárquico. Por exemplo, se você tem um índice hierárquico com três níveis e você especificar o parâmetro "level" como 2, o método "droplevel" irá remover o segundo nível do índice e retornar um índice com apenas dois níveis.

Esse parâmetro **é obrigatório**. Recebe o intervalo com a posição do nível, a ser excluído, no formato integer ou recebe o nome do nível, a ser excluído, no formato string. Pode receber também uma lista contendo posições ou nome dos níveis.

axis:

Esse parâmetro vai definir em qual direção ocorrerá a exclusão dos itens, se será no sentido horizontal, da esquerda para direita, ou no vertical, de cima para baixo. Por padrão, o pandas define "axis=0", ou seja, significa que a exclusão dos itens ocorrerá verticalmente.

Esse parâmetro **é opcional**. Pode ser 0, que significa que a exclusão dos itens ocorrerá no sentido vertical, de cima para baixo. Ou pode ser 1, que significa que a exclusão dos itens dos itens ocorrerá horizontalmente, da esquerda para direita.

No caso abaixo, estamos criando um DataFrame multiindex e excluindo um dos níveis do index dele, que neste caso é o "Empresas".

Exemplo:

```
import pandas as pd

cols = pd.MultiIndex.from_tuples([("Empresas","Energia", "WEGE3"), ("Empresas","Energia","VALE3")])
df_empresas = pd.DataFrame([[39.51,69.28], [39.45,68.67]], columns=cols, index=['2022-11-01','2022-11-02'])
print(df_empresas)

df_empresas = df_empresas.droplevel(level=0, axis=1)
print(df_empresas)
```

DataFrame antes do droplevel():

```
           Empresas
             Energia
               WEGE3  VALE3
2022-11-01     39.51  69.28
2022-11-02     39.45  68.67
```

DataFrame após do droplevel():

```
            Energia
              WEGE3  VALE3
2022-11-01    39.51  69.28
2022-11-02    39.45  68.67
```

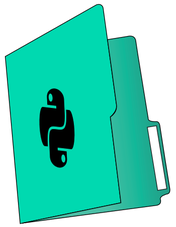
Mundo 22

Neste mundo abordaremos formas de concatenar dados.

1. pandas.concat(objs, *, axis = 0, join = "outer", ignore_index = False, keys = None, levels = None, names = None, verify_integrity = False, sort = False, copy = True)

O método "concat" junta dois objetos do pandas (Series ou DataFrames), concatenando-os na vertical (por linha) ou na horizontal (por coluna).

Parâmetros:



O parâmetro objs é o único obrigatório.

objs:

Esse parâmetro são os DataFrames, ou Series, que você deseja concatenar.

Esse parâmetro **é obrigatório**. Recebe uma lista com os DataFrames, ou Series, dentro.

axis:

Esse parâmetro vai definir em qual direção ocorrerá a concatenação dos dados, se será a concatenação horizontalmente, da esquerda para direita, ou verticalmente, de cima para baixo. Por padrão, o pandas define "axis=0", ou seja, significa que a concatenação será vertical.

Esse parâmetro **é opcional**. Pode ser 0, que significa uma concatenação vertical, de cima para baixo. Ou pode ser 1, que significa uma concatenação horizontal, da esquerda para direita.

join:

Esse parâmetro vai definir qual vai ser a forma em que os dados serão concatenador. Por padrão, o pandas define, "join = 'outer' ", ou seja, todos os dados serão considerados na hora da concatenação, e dados que não estão em comum serão apresentados com valores NaN.

Esse parâmetro **é opcional**. Recebe parâmetros pré-definidos no formato string, que são:

outer => Concatenar e apresentar todos os dados, com os dados faltantes em NaN

inner => Concatenar e apresentar apenas os dados que estão presentes na intersecção.

ignore_index:

Esse parâmetro vai definir se o index pré-estabelecido vai ser excluído ou não. Por padrão, o pandas define "ignore_index = False", ou seja, ele manterá o index de cada linha. Caso, "ignore_index = True", o index antigo seria excluído e substituído por um novo com a contagem das linhas começando por 0.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

keys:

Esse parâmetro vai definir um identificador para o index do resultado.

Esse parâmetro **é opcional** e recebe uma lista com os nomes dos identificadores no formato de string.

levels:

Esse parâmetro vai definir quais níveis devem ser concatenados.

Esse parâmetro **é opcional** e recebe uma lista com as posições dos níveis que devem ser concatenados.

names:

Esse parâmetro vai definir os nomes dos níveis dos index.

Esse parâmetro **é opcional** e recebe uma lista com os nomes dos níveis.

verify_integrity:

Verifica se o index definido possui duplicatas, se sim ele retornará um erro. Por padrão o pandas considera "verify_integrity=False", ou seja, ele não verificará a existência de duplicatas, logo não retornará um erro.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

sort:

Esse parâmetro vai definir se os index estarão ordenados ou não. Por padrão, o pandas considera "sort = True", ou seja, ele ordenará em ordem crescente, caso "sort = False" ele apenas não ordenará.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

copy:

Esse conceito é mais complexo. Ao definir copy=False, estamos definindo que as mudanças se propagarão entre os objetos. Isso quer dizer que se eu mudar uma Series criada por um vetor, este vetor mudará também.

obs: Isso só se aplica se tivermos criado uma Series utilizando **vetores (criados pela biblioteca NumPy)**.

Esse parâmetro **é opcional** e recebe um booleano: True ou False

No caso abaixo, estamos importando o preço ajustado, das empresas VALE3 e ABEV3. Criamos uma coluna adicional nos dois DataFrames importados com o ticker das empresas. Após isso, é utilizado pd.concat() ignorando o index.

Exemplo:

```
import pandas as pd
from pandas_datareader import data as pdr
import numpy as np

dados_ambev = pdr.get_data_yahoo("ABEV3.SA")
dados_vale = pdr.get_data_yahoo("VALE3.SA")
dados_ambev['empresa'] = ["ABEV3"] * len(dados_ambev)
dados_vale['empresa'] = ["VALE3"] * len(dados_vale)
dados_juntos = pd.concat([dados_ambev, dados_vale], ignore_index = True)
print(dados_juntos)
```

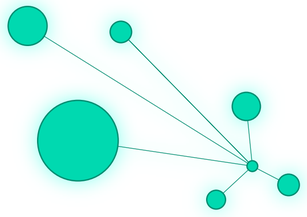
Resposta:

```
      High        Low       Open      Close      Volume  Adj Close empresa
0    21.650000  21.180000  21.650000  21.299999   8081700.0  18.477882   ABEV3
1    21.440001  21.240000  21.440001  21.320000  11173800.0  18.495232   ABEV3
2    21.450001  20.820000  21.400000  21.190001  15875200.0  18.382456   ABEV3
3    21.299999  20.850000  21.280001  21.040001  11557100.0  18.252329   ABEV3
4    21.330000  20.889999  21.160000  20.910000  14504400.0  18.139551   ABEV3
...        ...        ...        ...        ...        ...        ...     ...
2477 73.360001  69.599998  72.610001  69.830002  34091700.0  69.830002   VALE3
2478 71.239998  69.260002  70.209999  70.930000  25840500.0  70.930000   VALE3
2479 72.830002  70.739998  72.300003  71.959999  27494000.0  71.959999   VALE3
2480 72.129997  70.580002  71.629997  71.110001  22651000.0  71.110001   VALE3
2481 73.190002  70.320000  70.980003  72.220001  21145600.0  72.220001   VALE3

[2482 rows x 7 columns]
```

2. pandas.merge(right, how = “inner”, on = None, left_on = None, right_on = None, left_index = False, right_index = False, sort = False, suffixes = (“_x”, “_y”), copy = True, indicator = False, validate = None)

O método “merge” junta dois objetos , por elementos comuns, através de colunas ou index, sempre no sentido horizontal. A operação do merge equivale ao “join” do SQL. Essa função é sempre utilizada em conjunto com outro dataframe.

Parâmetros:



O parâmetro objs é o único obrigatório.

right:

Esse parâmetro define o DataFrames, ou Series, que você deseja juntar ao dataframe que está sendo aplicado a função.

Esse parâmetro **é obrigatório**. Recebe uma lista com os DataFrames, ou Series, dentro.

how:

Esse parâmetro vai definir qual vai ser a forma em que os dados serão concatenados. Por padrão, o pandas define, “how = ‘inner’ “, ou seja, só serão retornados os dados em comum com os dois DataFrames.

Esse parâmetro **é opcional**. Recebe parâmetros pré-definidos no formato string, que são:

- “how = ‘left’” - os dados do primeiro DataFrame são mantidos e os valores do segundo DataFrame são inseridos em branco nas linhas em que não houver correspondência.

- “how = ‘right’” - os dados do segundo DataFrame são mantidos e os valores do primeiro DataFrame são inseridos em branco nas linhas em que não houver correspondência.

- “how = ‘outer’” - todos os dados de ambos os DataFrames são mantidos e os valores em branco são inseridos nas linhas em que não houver correspondência.

- “how = ’inner’” - apenas os dados que estão presentes em ambos os DataFrames são mantidos.

on:

Esse parâmetro especifica o nome da coluna, ou index, que será usado para fazer a junção dos dois DataFrames. **Esse nome deve estar presente nas duas tabelas**. Por padrão, pandas define “on = None”.

Esse parâmetro **é opcional**. Recebe o nome da coluna no formato string.

left_on:

Esse parâmetro especifica o nome da coluna que será usada como chave primária no DataFrame da esquerda para fazer a junção dos dois DataFrames. **Diferente do parâmetro “on”, deve ser usado , em conjunto com o “right_on” quando o nome das colunas são diferentes**. Por padrão, pandas define “left_on = None”

Esse parâmetro **é opcional**. Recebe o nome da coluna no formato string.

right_on:

Esse parâmetro especifica o nome da coluna que será usada usada como chave primária no DataFrame da direita para fazer a junção dos dois DataFrames. **Diferente do parâmetro “on”, deve ser usado , em conjunto com o “left_on” quando o nome das colunas são diferentes**. Por padrão, pandas define “right_on = None”.

Esse parâmetro **é opcional**. Recebe o nome da coluna no formato string.

left_index:

Esse parâmetro especifica se o index será usado como chave primária no DataFrame da esquerda para fazer a junção dos dois DataFrames. Por padrão, pandas define “left_index = False”.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

right_index:

Esse parâmetro especifica se o index será usado como chave primária no DataFrame da direita para fazer a junção dos dois DataFrames. Por padrão, pandas define "right_index = False".

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

sort:

Esse parâmetro especifica se o resultado virá ordenado em ordem crescente ou não, de acordo com a chave de junção. Por padrão, o pandas considera "sort = False", ou seja, ele não ordena.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

suffixes:

Esse parâmetro serve para adicionar sufixos aos nomes das colunas, para identificação, de acordo com a fonte de dados.Por padrão, o pandas define "suffixes = ('_x', '_y')", ou seja, as colunas do DataFrame a esquerda receberão o sufixo "_x" enquanto as da direita, receberão "_y".

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

indicators:

Esse parâmetro vai definir se uma coluna adicional, com as informações de origem de cada linha, será criada. Por padrão, o pandas define "indicators = False", ou seja, não será criada uma coluna adicional com a origem de cada linha.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

validate:

Esse parâmetro vai checar se a correspondência vai ser: uma para uma, muitas para uma, uma para muitas ou muitas para muitas.

Esse parâmetro **é opcional** e recebe parâmetros pré definidos:

"one_to_one" or "1:1": Checa se a correspondência dos dois DataFrame são únicas, ou seja, uma para uma.

"one_to_many" or "1:m": Checa se a correspondência dos DataFrame é de uma (para o DataFrame da esquerda) para muitas (para o DataFrame da direita).

"many_to_one" or "m:1": Checa se a correspondência dos DataFrame é de muitas (para o DataFrame da esquerda) para uma (para o DataFrame da direita).

"many_to_many" or "m:m": Checa se há múltiplas correspondências nos dois DataFrame, ou seja, se é de muitas para muitas

No caso abaixo, estamos criando dois DataFrames e dando merge através da coluna "ticker".

Exemplo:

```
import pandas as pd

dados_roe = pd.DataFrame({"ticker": ['WEGE3', 'VALE3', 'PCAR3', 'ELET3'],
                          "ROE": [20, 9, 19, 3]})
dados_roic = pd.DataFrame({"ticker": ['WEGE3', 'VALE3', 'PCAR3', 'ELET3'],
                           "ROIC": [25, 6, 13, 1]})
indicadores = pd.merge(dados_roe, dados_roic, how = "inner",
              on = "ticker")

print(indicadores)
```

Resposta:

```
  ticker  ROE  ROIC
0  WEGE3   20    25
1  VALE3    9     6
2  PCAR3   19    13
3  ELET3    3     1
```

3. DataFrame.join(other, on = None, how = “left”, lsuffix = “ “, rsuffix = “ “, sort = False, validate = None)

O método “join()” junta dois objetos , por elementos comuns, através de colunas ou index, sempre no sentido horizontal. É um caso específico da função “merge”.

Parâmetros:

O parâmetro objs é o único obrigatório.

other:

Esse parâmetro vai definir o DataFrame, ou Series, que você deseja concatenar.

Esse parâmetro **é obrigatório**. Recebe um DataFrame, ou Series, dentro.

on:

Esse parâmetro especifica o nome da coluna, ou index, que será usado para fazer a junção dos dois DataFrames. Esse nome deve estar presente nos dois DataFrames. Por padrão, pandas define “on = None”.

Esse parâmetro **é opcional**. Recebe o nome da coluna no formato string.

how:

Esse parâmetro vai definir qual vai ser a forma em que os dados serão concatenados. Por padrão, o pandas define, “how = ‘inner’ “, ou seja, só serão retornados os dados em comum com os dois DataFrames.

Esse parâmetro **é opcional**. Recebe parâmetros pré-definidos no formato string, que são:

- “how = ‘left’” - os dados do primeiro DataFrame são mantidos e os valores do segundo DataFrame são inseridos em branco nas linhas em que não houver correspondência.

- “how = ’right’” - os dados do segundo DataFrame são mantidos e os valores do primeiro DataFrame são inseridos em branco nas linhas em que não houver correspondência.

- “how = ’outer’” - todos os dados de ambos os DataFrames são mantidos e os valores em branco são inseridos nas linhas em que não houver correspondência.

- “how = ’inner’” - apenas os dados que estão presentes em ambos os DataFrames são mantidos.

rsuffix:

Esse parâmetro especifica o sufixo a ser usado nas colunas sobrepostas a direita.

Esse parâmetro **é opcional**. Recebe o sufixo no formato string.

lsuffix:

Esse parâmetro especifica o sufixo a ser usado nas colunas sobrepostas a esquerda.

Esse parâmetro **é opcional**. Recebe o sufixo no formato string.

sort:

Esse parâmetro especifica se o resultado virá ordenado em ordem crescente ou não, de acordo com a chave de junção. Por padrão, o pandas considera “sort = False”, ou seja, ele não ordena.

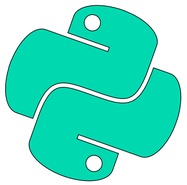
Esse parâmetro **é opcional** e recebe um booleano: True ou False.

No caso abaixo, estamos importando os dados da ABEV3 e criando um DataFrame adicional, com a data como index e as colunas ticker e P/L. Nesse caso, o P/L foi gerado randomicamente. Podemos juntar essas 2 tabelas com o join, através do index.

Exemplo:

```
import numpy as np
import pandas as pd
import pandas_datareader as pdr

dados_cotacoes = pdr.get_data_yahoo("ABEV3.SA")
dados_pl = pd.DataFrame({"ticker": ['ABEV3'] * len(dados_cotacoes),
                         "PL": np.random.randint(10, 20, len(dados_cotacoes))},
                         index = dados_cotacoes.index)

dados_finais = dados_cotacoes.join(dados_pl)

print(dados_finais)
```

Resposta:



```
              High        Low       Open      Close      Volume  Adj Close ticker  PL
Date
2017-10-23  21.650000  21.180000  21.650000  21.299999   8081700.0  18.477879  ABEV3  19
2017-10-24  21.440001  21.240000  21.440001  21.320000  11173800.0  18.495232  ABEV3  10
2017-10-25  21.450001  20.820000  21.400000  21.190001  15875200.0  18.382454  ABEV3  12
2017-10-26  21.299999  20.850000  21.280001  21.040001  11557100.0  18.252329  ABEV3  16
2017-10-27  21.330000  20.889999  21.160000  20.910000  14504400.0  18.139551  ABEV3  14
...               ...        ...        ...        ...         ...        ...    ...  ..
2022-10-17  15.060000  14.770000  14.920000  14.920000  11542800.0  14.920000  ABEV3  13
2022-10-18  15.130000  14.820000  15.060000  14.900000  24386300.0  14.900000  ABEV3  11
2022-10-19  15.040000  14.780000  14.930000  14.890000  31016000.0  14.890000  ABEV3  18
2022-10-20  15.110000  14.900000  14.960000  15.040000  25176800.0  15.040000  ABEV3  13
2022-10-21  15.100000  14.950000  14.990000  15.100000   3285800.0  15.100000  ABEV3  15

[1242 rows x 8 columns]
```

Mundo 23

Neste mundo abordaremos formas de redimensionar os DataFrames.

1. pandas.resample(rule, axis = 0, closed = None, convention = “start”, label = None, kind = None, on = None, level = None, origin = “start_day”, offset = None)

O método “resample” funciona como forma de mudar a frequência com que os dados aparecem. Você pode agrupar os dados, transformando dados mensais em dados semanais, ou pode separar os dados, transformando dados semanais em dados mensais. Nos dois casos o pandas utilizaria o método de interpolação.

Esse método é utilizado em conjunto com outros métodos como: last(), mean(), sum(), etc...

Parâmetros:

O parâmetro objs é o único obrigatório.

rule:

Esse parâmetro vai definir a forma em que seus dados vão ser apresentados. Podemos definir essa parâmetro através da **Tabela de parâmetro da frequência** presente nas referências desse PDF.

Esse parâmetro **é obrigatório**.

axis:

Esse parâmetro vai definir em qual direção ocorrerá o redimensionamento dos itens, se será horizontalmente, da esquerda para direita, ou verticalmente, de cima para baixo. Por padrão, o pandas define “axis=0”, ou seja, significa que o redimensionamento vertical.

Esse parâmetro **é opcional**. Pode ser 0, que significa um redimensionamento vertical, de cima para baixo. Ou pode ser 1, que significa um redimensionamento horizontal, da esquerda para direita.

closed:

O parâmetro “closed” na função “resample” da biblioteca pandas especifica se o intervalo de amostragem deve ser fechado no início ou no final. Se o parâmetro “closed” for “left”, o intervalo será fechado no início; Se o parâmetro “closed” for “right”, o intervalo será fechado no final.

Esse parâmetro **é opcional**. Pode receber parâmetros pré-definidos, que são:

right => intervalo fechado final

left => intervalo fechado no início

label:

O parâmetro “label” no método “resample” é usado para determinar como os rótulos das linhas serão manipulados durante a amostragem. Se “label” for “None”, os rótulos serão mantidos como estão. Se “label” for “left”, os rótulos serão os mesmos do início. Se “label” for “right”, os rótulos serão os mesmos do final.

Esse parâmetro **é opcional**. Pode receber parâmetros pré-definidos, que são:

right => intervalo fechado final

left => intervalo fechado no início

convention:

Este parâmetro pode ser usado quando existe um formato datetime no index. Ele vai definir se os dados vão ser agrupados nos primeiros, ou nos últimos, meses de um período. Por padrão, o pandas define "convention = " start " ", ou seja, serão agrupados de acordo com o início do período.

Esse parâmetro **é opcional**. Pode receber parâmetros pré-definidos, que são:

"end" ou "e" => Os dados serão agrupados de acordo com o final do período

"start" ou "s" => Os dados serão agrupados de acordo com o início do período.

kind:

Este parâmetro vai definir como converter o index.

Esse parâmetro **é opcional**. Pode receber parâmetros pré-definidos, que são:

" timestamp " => Vai converter o index para DateTimeIndex

" period " => Vai converter o index para PeriodIndex

on:

Este parâmetro vai ser definido quando se deseja utilizar uma coluna do DataFrame, ao invés do index.

Esse parâmetro **é opcional**. Recebe o nome da coluna no formato string.

level:

Este parâmetro é utilizado em um DataFrame Multindex e vai ser definido quando se deseja utilizar um nível do DataFrame. Esse nível deve ser do tipo DateTime.

Esse parâmetro **é opcional**. Recebe o nome da coluna no formato string.

origin:

Esse parâmetro vai definir a data referência a ser considerada como origem. Por padrão, o pandas considera "origin = start_day' ", ou seja, ele considera o primeiro valor, às meia noite da data da Séries.

Esse parâmetro **é opcional**. Pode receber parâmetros pré-definidos, que são:

"epoch" => Considera a origem 1970-01-01

"start" => Considera a origem como primeiro valor da Series de data

"start_day" => Considera a origem como meia noite do primeiro valor da Series de data

"end" Considera a origem como último valor da Series de data

"end_day" Considera a origem como meia noite do último valor da Series de data

offset:

Esse parâmetro vai definir um deslocamento, de tempo, a ser adicionado à origem da data.

Esse parâmetro **é opcional**. Recebe um valor em TimeDelta, ou um valor em string.

No caso abaixo, estamos importando os dados de cotação ajustada da WEGE3 e redimensionando o DataFrame, passando a frequência dele de diária para mensal. Utilizaremos também o método last() para pegar a data de fechamento mensal.

Exemplo:

```
import pandas_datareader as pdr

dados_weg = pdr.get_data_yahoo("WEGE3.SA")['Adj Close']
print(dados_weg)

dados_weg_mensal = dados_weg.resample("M").last()
print(dados_weg_mensal)
```

Dados Diários (antes de resample):

```
Date
2017-11-02     7.581912
2017-11-03     7.682194
2017-11-06     7.775312
2017-11-07     7.564006
2017-11-08     7.786054
                 ...
2022-10-26    37.980000
2022-10-27    38.830002
2022-10-28    38.470001
2022-10-31    40.279999
2022-11-01    39.520000
Name: Adj Close, Length: 1241, dtype: float64
```

Dados Mensais (depois do resample):

```
Date
2017-11-30     8.194343
2017-12-31     8.655786
2018-01-31     8.508590
2018-02-28     8.404476
2018-03-31     8.164780
                  ...
2022-07-31    27.999533
2022-08-31    28.249172
2022-09-30    32.139999
2022-10-31    40.279999
2022-11-30    39.520000
Freq: M, Name: Adj Close, Length: 61, dtype: float64
```

Mundo 24

Neste mundo abordaremos formas de criar janelas para o cálculo das médias móveis ou volatilidade, por exemplo.

1. pandas.rolling(window, min_periods = None, center = False, win_type = None, on = None, axis = 0, closed = None, step = None, method = “single”)

O método “rolling” é utilizado para criar janelas móveis. É utilizado em conjunto com outros métodos como: std(), sum(), mean(), etc.

Parâmetros:

O parâmetro window é o único obrigatório.

window:

Esse parâmetro determina o tamanho da janela que será usada para cada iteração.

Esse parâmetro **é obrigatório**. Recebe um inteiro que determina a quantidade, do intervalo (dias, meses ou anos) a ser considerado.

min_periods:

O parâmetro “min_periods” especifica o número mínimo de **períodos** necessários para que a rolagem seja aplicada. Por padrão, o pandas define “min_periods = 1”, ou seja, o mínimo de períodos necessários é um.

Esse parâmetro **é opcional**. Recebe o número de períodos no formato integer.

center:

Esse parâmetro vai definir se a janela será alinhada pelo centro ou pela esquerda do intervalo. Por padrão, o pandas define “center = False “, ou seja, os intervalos vão ser definidos pela esquerda.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

on:

Esse parâmetro vai definir qual coluna utilizar para calcular a janela contínua. Por padrão, se utiliza o index.

Esse parâmetro **é opcional**. Recebe o nome da coluna no formato string.

axis:

Esse parâmetro vai definir se as janelas serão calculadas no sentido vertical ou horizontal. Por padrão, o pandas define “axis = 0”, ou seja, o cálculo será feito no sentido vertical.

Esse parâmetro **é opcional**. Recebe 0 se deseja fazer o cálculo da janela no sentido vertical, ou 1 se deseja fazer o cálculo da janela no sentido horizontal.

closed:

Esse parâmetro vai definir se os extremos do eixo serão incluídos no cálculo da média móvel. Por padrão, o pandas define “closed = ‘ right’ “.

Esse parâmetro **é opcional**. Recebe parâmetros pré-definidos:

“right” => O primeiro ponto da janela será excluído dos cálculos

“left” => O último ponto da janela será excluído dos cálculos

“both” => Não terão pontos excluídos dos cálculos

“neither” => O primeiro e o último pontos da janela serão excluídos dos cálculos

step:

Esse parâmetro determina o tamanho do intervalo entre os dados. Em outras palavras, ele determina o número de dados que serão usados para cada cálculo.

Esse parâmetro **é opcional**. Recebe o tamanho do intervalo a ser calculado.

method:

Esse parâmetro vai definir se as operações serão feitas em uma única coluna, ou linha, ou na tabela inteira. Por padrão, o pandas define "method = 'single' ".

Esse parâmetro **é opcional**. Recebe parâmetros pré-definidos:

single => Para fazer a operação linha por linha

table => Para fazer a operação na tabela inteira

No caso abaixo, estamos importando os dados de cotação ajustada da PETR4. Estamos criando uma janela móvel de 250 dias e calculando sua média.

Exemplo:

```
import pandas as pd
from pandas_datareader import data as pdr
import numpy as np

dados_acao = pdr.get_data_yahoo("PETR4.SA", "2010-01-01", "2022-08-31")['Close']
dados_acao = dados_acao.to_frame()
media_movel_250d = dados_acao.rolling(250).mean()

print(media_movel_250d)
```

Resposta:

```
                Close
Date
2010-01-04        NaN
2010-01-05        NaN
2010-01-06        NaN
2010-01-07        NaN
2010-01-08        NaN
...               ...
2022-08-25   30.23972
2022-08-26   30.26432
2022-08-29   30.28828
2022-08-30   30.30480
2022-08-31   30.32896

[3143 rows x 1 columns]
```

2. pandas.ewm(com = None , span = None , halflife = None, alpha = None, min_periods = 0, adjust = True, ignore_na = False, axis = 0, times = None, method = “single”)

O método “ewm” O método EWM (Exponentially Weighted Windows) é utilizado para criar janelas móveis exponenciais. É muito utilizado em conjunto com os seguintes métodos: mean(), var(), std(), corr() e cov().

Parâmetros:

Por padrão, o pandas considera a seguinte equação para o cálculo do método EWM:

Se adjust = True, definido pelo pandas inicialmente:

$$y_t = \frac{x_t + (1-\alpha)x_{t-1} + (1-\alpha)^2 x_{t-2} + \ldots + (1-\alpha)^t x_0}{1 + (1-\alpha) + (1-\alpha)^2 + \ldots + (1-\alpha)^t}$$

OU

Se adjust = False:

$$y_0 = x_0$$
$$y_t = (1-\alpha)y_{t-1} + \alpha x_t,$$

O alfa é o valor de suavização e pode ser definido pelos cálculos: “com”, “spam”,”halflife” ou definido diretamente através do “alpha”.

Se o parâmetro “times” não for definido, um dos seguintes parâmetros **DEVEM** ser definidos: “com”, “spam”,”halflife” ou “alpha”. Caso o parâmetro “times” for definido, os seguintes parâmetros **PODEM** ser definidos: “com”, “spam”,”halflife” ou “alpha”.

com:

Esse parâmetro vai definir o alfa, que é o coeficiente de suavização, através da seguinte fórmula:

$$\alpha = \frac{1}{(1 + com)} \; ; \; com > 0$$

Esse parâmetro **é um dos obrigatórios**. Recebe um número inteiro maior que 0.

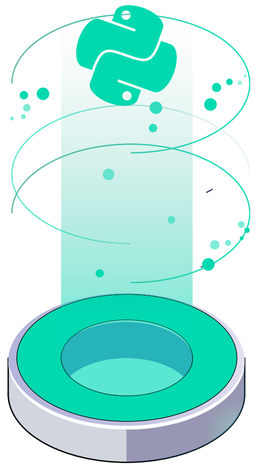
span:

Esse parâmetro vai definir o alfa, que é o coeficiente de suavização, através da seguinte fórmula:

$$\alpha = \frac{2}{(span \; + 1)} \; ; \; span \; > 1$$

Esse parâmetro **é um dos obrigatórios**. Recebe um número inteiro maior que 1

halflife:



Esse parâmetro vai definir o alfa, que é o coeficiente de suavização, através da seguinte fórmula:

$$\alpha = \; 1 - \exp\left(\frac{-\ln(2)}{halflife}\right) \; ; \; halflife > 0$$

Ele é aplicável apenas para o método .mean().

Esse parâmetro **é um dos obrigatórios**. Recebe um número inteiro maior que 0.

alpha:

Esse parâmetro vai definir diretamente o alpha, tendo que ser um número entre 0 e 1.

Esse parâmetro **é um dos obrigatórios**. Sendo 0 < alpha < 1

min_periods:

O parâmetro “min_periods” especifica o número mínimo de **períodos** necessários para que a rolagem seja aplicada. Por padrão, o pandas define “min_periods = 0”, ou seja, o mínimo de períodos necessários é zero. Caso o número de períodos for menor que o definido, o resultado será o valor Nan.

Esse parâmetro **é opcional**. Recebe o número de períodos no formato integer.

adjust:

Essse parâmetro vai definir a forma em que o ajuste será feito.

Se “adjust = True” então:

$$y_t = \frac{x_t + (1-\alpha)x_{t-1} + (1-\alpha)^2 x_{t-2} + \ldots + (1-\alpha)^t x_0}{1 + (1-\alpha) + (1-\alpha)^2 + \ldots + (1-\alpha)^t}$$

Se “adjust = False” então:

$$y_0 = x_0$$
$$y_t = (1-\alpha)y_{t-1} + \alpha x_t,$$

Por padrão, o pandas define “adjuste = True”

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

ignore_na:

Esse parâmetro vai definir se os valores NaN serão calculados na hora de contabilizar o peso dos valores. Por padrão o pandas define “ignore_na = False”, ou seja, o pandas não contabiliza os valores NaN.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

axis:

Esse parâmetro vai definir se as janelas serão calculadas no sentido vertical, de cima para baixo, ou horizontal, da esquerda para direita. Por padrão, o pandas define “axis = 0”, ou seja, o cálculo será feito no sentido vertical.

Esse parâmetro **é opcional**. Recebe 0 se deseja fazer o cálculo da janela no sentido vertical, de cima para baixo, ou 1 se deseja fazer o cálculo da janela no sentido horizontal, da esquerda para direita.

method:

Esse parâmetro vai definir se as operações serão feitas em uma única coluna, ou linha, ou na tabela inteira. Por padrão, o pandas define “method = ‘single’ ”.

Esse parâmetro **é opcional**. Recebe parâmetros pré-definidos:

single => Para fazer a operação linha por linha

table => Para fazer a operação na tabela inteira

No caso abaixo, estamos importando os dados de cotação ajustada da PETR4. Estamos criando uma janela móvel de 252 dias e calculando sua média exponencial.

Exemplo:

```
import pandas as pd
from pandas_datareader import data as pdr
import numpy as np

dados_acao = pdr.get_data_yahoo("PETR4.SA", "2010-01-01", "2022-08-31")['Close']
dados_acao = dados_acao.to_frame()
media_movel_exp = dados_acao.ewm(span = 252).mean()


print(media_movel_exp)
```

Resposta:

```
                Close
Date
2010-01-04  37.320000
2010-01-05  37.159365
2010-01-06  37.273812
2010-01-07  37.242490
2010-01-08  37.183060
...               ...
2022-08-25  30.192833
2022-08-26  30.220084
2022-08-29  30.253759
2022-08-30  30.270962
2022-08-31  30.294354

[3143 rows x 1 columns]
```

Mundo 25

Neste mundo abordaremos os métodos de covariância e correlação.

1. pandas.cov(min_periods = None, ddof = 1)

O método “cov” é utilizado para achar a covariância entre duas Series.

Parâmetros:

Não existe parâmetro obrigatório.

min_periods:

O parâmetro “min_periods” especifica o número mínimo de **períodos** necessários para que a rolagem seja aplicada. Por padrão, o pandas define “min_periods = 0”, ou seja, o mínimo de períodos necessários é zero. Caso o número de períodos for menor que o definido, o resultado será o valor Nan.

Esse parâmetro **é opcional**. Recebe o número de períodos no formato integer.

ddof:

Esse parâmetro representa o grau de liberdade.

Esse parâmetro **é opcional**. Recebe um número no formato integer.

No caso abaixo, estamos importando os dados de cotação ajustada das ações: ’WEGE3.SA’, ’VALE3.SA’, ’PETR4.SA’, ’LREN3.SA’, ’PETR3.SA’. Depois calculamos os retornos diários dela, excluindo os valores NaN, e no fim calculamos a covariância entre elas.

Exemplo:

```
import pandas as pd
from pandas_datareader import data as pdr
import numpy as np

acoes = ['WEGE3.SA', 'VALE3.SA', 'PETR4.SA', 'LREN3.SA', 'PETR3.SA']
dados_acao = pdr.get_data_yahoo(acoes, "2019-08-31", "2022-08-31")['Adj Close']
retornos = dados_acao.pct_change().dropna()

print(retornos.cov())
```

Resposta:

Symbols	WEGE3.SA	VALE3.SA	PETR4.SA	LREN3.SA	PETR3.SA
Symbols					
WEGE3.SA	0.000760	0.000207	0.000339	0.000312	0.000347
VALE3.SA	0.000207	0.000729	0.000472	0.000218	0.000495
PETR4.SA	0.000339	0.000472	0.001018	0.000458	0.001034
LREN3.SA	0.000312	0.000218	0.000458	0.000930	0.000482
PETR3.SA	0.000347	0.000495	0.001034	0.000482	0.001099

2. pandas.corr(method = “pearson”, min_periods = 1)

O método “corr” é utilizado para achar a correlação, excluindo os valores NaN.

Parâmetros:

Não existe parâmetro obrigatório.

method:

Esse parâmetro vai definir por qual método de correlação será calculado a correlação. Por padrão, o pandas define “method = ‘pearson’ “, ou seja, ele calcula a correlação por meio do método ‘pearson’.

Esse parâmetro **é opcional**. Recebe parâmetros pré definidos. Que são:

pearson => utiliza o coeficiente de correlação de pearson

kendall => utiliza o coeficiente de correlação de kendall

spearman => utiliza o coeficiente de correlação de spearman

min_periods:

O parâmetro “min_periods” especifica o número mínimo de **períodos** necessários para que a rolagem seja aplicada. Por padrão, o pandas define “min_periods = 0”, ou seja, o mínimo de períodos necessários é zero. Caso o número de períodos for menor que o definido, o resultado será o valor Nan.

Esse parâmetro **é opcional**. Recebe o número de períodos no formato integer.

No caso abaixo, estamos importando os dados de cotação ajustada das ações: ’WEGE3.SA’, ’VALE3.SA’, ’PETR4.SA’, ’LREN3.SA’, ’PETR3.SA’. Depois calculamos os retornos diários dela, excluindo os valores NaN, e no fim calculamos a correlação entre elas.

Exemplo:

```
import pandas as pd
from pandas_datareader import data as pdr
import numpy as np

acoes = ['WEGE3.SA', 'VALE3.SA', 'PETR4.SA', 'LREN3.SA', 'PETR3.SA']
dados_acao = pdr.get_data_yahoo(acoes, "2019-08-31", "2022-08-31")['Adj Close']
retornos = dados_acao.pct_change().dropna()

print(retornos.corr())
```

Resposta:

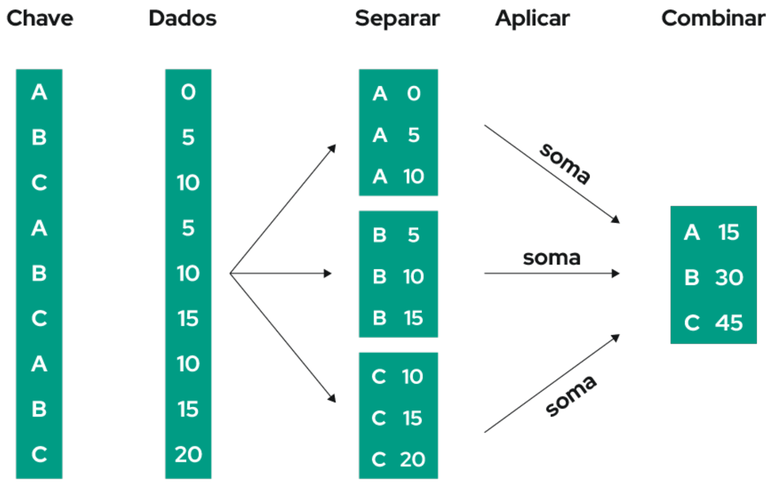
```
Symbols    WEGE3.SA  VALE3.SA  PETR4.SA  LREN3.SA  PETR3.SA
Symbols
WEGE3.SA   1.000000  0.278516  0.385526  0.371410  0.379649
VALE3.SA   0.278516  1.000000  0.547843  0.264999  0.553386
PETR4.SA   0.385526  0.547843  1.000000  0.470704  0.977138
LREN3.SA   0.371410  0.264999  0.470704  1.000000  0.476779
PETR3.SA   0.379649  0.553386  0.977138  0.476779  1.000000
```

Mundo 26

Neste mundo abordaremos formas de agrupar dados utilizando o método groupby.

1. pandas.groupby(by = None, axis = 0 , level = None , as_index = True , sort = True, dropna = True)

O método “groupby” é utilizado para aplicar operações matemáticas em diferentes grupos de uma mesma tabela. Em resumo, esse método separa as informações desejadas, agrupa essas informações e utiliza alguma operação matemática para essse grupo específico. É utilizado em conjunto com os métodos: sum(), mean(), var(), std(), corr() e cov().



Parâmetros:

O único parâmetro obrigatório é o by.

by:

Esse parâmetro vai definir os dados a serem agrupados.

Esse parâmetro **é obrigatório**. Pode receber:

Função => Uma função criada através do método lambda, por exemplo, uma função que só agrupa os pares: lambda x: x % 2 != 0

Lista => Uma lista que contém o nome de uma, ou mais, colunas no formato string.

axis:

Esse parâmetro vai definir se as informações serão agrupadas no sentido vertical, de cima para baixo, ou horizontal, da esquerda para direita. Por padrão, o pandas define “axis = 0”, ou seja, o cálculo será feito no sentido vertical.

Esse parâmetro **é opcional**. Recebe 0 se deseja fazer o agrupamento no sentido vertical, de cima para baixo, ou 1 se deseja fazer o agrupamento no sentido horizontal, da esquerda para direita.

level:

Este parâmetro é utilizado em um DataFrame Multindex e vai ser definido quando se deseja utilizar o group by a partir de um determinado nível.

Esse parâmetro **é opcional**. Recebe o nome do nível no formato string ou o número da posição do nível no formato integer.

as_index:

Este parâmetro vai definir se as colunas do DataFrame resultante do agrupamento, virão como index ou não. Por padrão, o pandas define "as_index = True", ou seja, elas vêm no formato index.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

sort:

Esse parâmetro especifica se os dados devem ser classificados antes de agrupar. Por padrão, o pandas define "sort = True", para uma melhor performance, definir "sort = False".

Obs: Isso não mudará a ordem dos dados no DataFrame.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

dropna:

Esse parâmetro determina se os valores faltantes serão excluídos do agrupamento. Se "dropna" for True, os valores faltantes serão excluídos. Se "dropna" for False, os valores faltantes serão mantidos no agrupamento. Por padrão, o pandas define "dropna = True", ou seja, ele vai excluir as faltantes.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

No caso abaixo, estamos criando um DataFrame com o lucro da Petrobras e Wege no 1°, 2°, 3° e 4° trimestre. Agrupa-se por empresas e depois soma o resultado.

Exemplo:

```
import pandas as pd
from pandas_datareader import data as pdr
import numpy as np

dados_fundamentalistas = pd.DataFrame({"empresa": ['WEGE', 'WEGE', 'WEGE', 'WEGE',
                                        'PETROBRAS', 'PETROBRAS', 'PETROBRAS', 'PETROBRAS'],
                                        "trimestre": ["1TRI", "2TRI", "3TRI", "4TRI"] * 2 ,
                                        "lucro": [20, 30, 40, 55, 90, 95, 105, 123]})

print(dados_fundamentalistas.groupby("empresa")['lucro'].sum())
```

Resposta:

```
empresa
PETROBRAS    413
WEGE         145
Name: lucro, dtype: int64
```

Mundo 27

Neste mundo abordaremos formas de agrupar dados utilizando o método groupby.

1. pandas.apply(func, axis = 0, raw = False, result_type = None)

O método "apply" é utilizado para aplicar funções em um certo intervalo do DataFrame. Esse método é muito utilizado em conjunto com o método groupby(), que é utilizado para separar e agrupar dados. Além de poder aplicar funções proprietárias ao conjunto de dados escolhido, esse método facilita na hora de aplicar múltiplas funções em um grupo.

Parâmetros:

O único parâmetro obrigatório é func.

func:

Este parâmetro recebe a função que será utilizada, que pode ser nativa do Python ou criada a partir de uma função lambda.

Esse parâmetro **é obrigatório** e recebe a função a ser utilizada.

axis:

Esse parâmetro vai definir se a função será aplicada no sentido vertical, de cima para baixo, ou horizontal, da esquerda para direita. Por padrão, o pandas define "axis = 0", ou seja, a função será aplicada no sentido vertical.

Esse parâmetro **é opcional**. Recebe 0 se deseja fazer a aplicação no sentido vertical, de cima para baixo, ou 1 se deseja fazer a aplicação no sentido horizontal, da esquerda para direita.

raw:

Este parâmetro vai definir se a função vai ser passada como um objeto narray. Por padrão, o pandas define "raw = False", ou seja, os objetos serão passados como uma Series.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

result_type:

Este parâmetro só vai funcionar quando "axis = 1". Ele vai definir como será retornado o resultado.

Esse parâmetro **é opcional** e recebe parâmetros pré-definidos:

expand => Resultados, semelhantes a listas, serão transformados em colunas.

reduce => Retorna uma Series, ao invés de expandir os dados transformando em colunas, o oposto do "expand".

broadcast => Os resultados serão transmitidos para a forma original do DataFrame.

No caso abaixo, estamos criando uma função que retorna os valores mínimos, máximos, médias e volumes. Essa função será utilizada no método apply().

Exemplo:

```
import pandas as pd
from pandas_datareader import data as pdr
import numpy as np

def estatisticas_ibov(agrupamento):

    return {'min': agrupamento.min() * 100, 'max': agrupamento.max() * 100,
            'media': agrupamento.mean() * 100, 'vol': agrupamento.std() * np.sqrt(252)}

cotacoes = pdr.get_data_yahoo("^BVSP", "2021-08-31", "2022-08-31")['Close']
cotacoes = cotacoes.pct_change().dropna().to_frame()
cotacoes['ano'] = cotacoes.index.year
cotacoes['mes'] = cotacoes.index.month
estatisticas_por_ano = cotacoes.groupby('ano')['Close'].apply(estatisticas_ibov)

print(estatisticas_por_ano)
```

Resposta:

```
                empresa
                PETROBRAS    413
                WEGE         145
                Name: lucro, dtype: int64
```

2. pandas.transform(func, axis = 0)

O método “transform” é utilizado para aplicar funções em todos os itens de um DataFrame, ou Series. Ele pode ser utilizado com funções nativas do Python ou funções proprietárias, criada por você.

Parâmetros:

O único parâmetro obrigatório é func.

func:

Este parâmetro recebe a função que será utilizada, que pode ser nativa do Python ou criada a partir de uma função lambda.

Esse parâmetro **é obrigatório** e recebe a função a ser utilizada.

axis:

Esse parâmetro vai definir se a função será aplicada no sentido vertical, de cima para baixo, ou horizontal, da esquerda para direita. Por padrão, o pandas define “axis = 0”, ou seja, a função será aplicada no sentido vertical.

Esse parâmetro **é opcional**. Recebe 0 se deseja fazer a aplicação no sentido vertical, de cima para baixo, ou 1 se deseja fazer a aplicação no sentido horizontal, da esquerda para direita.

No caso abaixo, estamos importando a cotação ajustada da “WEGE3” e utilizando a função transform para normalizar os dados.

Exemplo:

```
from pandas_datareader import data as pdr

cotacoes = pdr.get_data_yahoo("WEGE3.SA", "2019-08-31", "2022-08-31")['Adj Close']
cotacoes_normalizadas = cotacoes.transform(lambda x: (x - x.mean())/x.std() )

print(cotacoes_normalizadas)
```

Resposta:

```
Date
2019-09-02   -2.084735
2019-09-03   -2.117794
2019-09-04   -2.093140
2019-09-05   -2.084735
2019-09-06   -2.081374
                ...
2022-08-25    0.005460
2022-08-26   -0.010820
2022-08-29   -0.061988
2022-08-30   -0.059662
2022-08-31   -0.071291
Name: Adj Close, Length: 745, dtype: float64
```

3. pandas.describe(percentiles = None, include = None, exclude = None, datetime_is_numeric = False)

O método “describe” vai fornecer estatísticas descritivas dependendo do seu conjunto de dados.

Parâmetros:

O único parâmetro obrigatório é func.

percentiles:

Este parâmetro vai definir os percentis a ser retornado. Por padrão, o pandas define “percentiles = [0.25, 0.50, 0.75]”.

Esse parâmetro **é opcional** e recebe uma lista com os percentis maiores que 0 e menores que 1.

include:

Este parâmetro vai definir as colunas a serem incluídas no cálculo e no resultado. Por padrão, o pandas define “include = ‘None’, ou seja, o resultado incluirá apenas as colunas numéricas, ou seja, as colunas strings serão excluídas do cálculo e do DataFrame final.

Esse parâmetro **é opcional** e recebe parâmetros pré-definidos. Que são:

all => Todas as colunas de entrada serão incluídas na saída

NumPy.number => Para limitar as saídas apenas para entradas numéricas

NumPy.object => Para limitar as saídas apenas para entradas do tipo string

exclude:

Este parâmetro vai definir as colunas a serem excluídas no cálculo e no resultado. Por padrão, o pandas define “exclude = ‘None’, ou seja, o resultado não exclui nada.

Esse parâmetro **é opcional** e recebe parâmetros pré-definidos. Que são:

NumPy.number => Para excluir as saídas apenas para entradas numéricas

NumPy.object => Para excluir as saídas apenas para entradas do tipo string

datetime_is_numeric:

Este parâmetro vai definir se os formatos datetime devem ser tratados como números.

Esse parâmetro **é opcional** e recebe um booleano: True ou False.

No caso abaixo, estamos importando a cotação ajustada da "WEGE3" e utilizando a função transform para normalizar os dados. Ao final, utilizamos o método describe() para pegar as estatísticas descritivas básicas

Exemplo:

```
from pandas_datareader import data as pdr

cotacoes = pdr.get_data_yahoo("WEGE3.SA", "2019-08-31", "2022-08-31")['Adj Close']
cotacoes_normalizadas = cotacoes.transform(lambda x: (x - x.mean())/x.std() )
cotacoes_normalizadas = cotacoes_normalizadas.describe()

print(cotacoes_normalizadas)
```

Resposta:

```
count    7.450000e+02
mean     7.629989e-17
std      1.000000e+00
min     -2.117794e+00
25%     -7.765781e-01
50%      2.813382e-01
75%      7.337506e-01
max      1.861732e+00
Name: Adj Close, dtype: float64
```

Referências

Mundo 3

Tabela de parâmetro da frequência.

Formatos	Descrição
B	Dias úteis a partir da data
C	Dias úteis personalizados a partir da data
D	Diários a partir da data
W	Semanal a partir da data
M	Mensal a partir da data
SM	Quinzenal a partir da data
BM	Último dia útil de cada mês
CBM	Último dia útil personalizado de cada mês
MS	Primeiro dia de cada mês
SMS	Quinzenalmente pegando o 1 e o 15 dia do mês
BMS	Primeiro dia útil de cada mês
CBMS	Primeiro dia útil personalizado de cada mês
Q	Trimestralmente com o último dia do mês
BQ	Trimestralmente com o último dia útil do mês
QS	Trimestralmente com o primeiro dia do mês
BQS	Trimestralmente com o primeiro dia útil do mês
A, Y	Anualmente com o último dia do ano
BA, BY	Anualmente com o último dia útil do ano
AS, YS	Anualmente com o primeiro dia do ano
BAS, BYS	Anualmente com o primeiro dia útil do ano
BH	De hora em hora apenas com as horas úteis
H	De hora em hora
T, min	De minuto a minuto
S	De segundo a segundo

L, ms	De milisegundo a milisegundo
U, us	De microsegundo a microseungo
N	De nanosegundo a nanosegundo

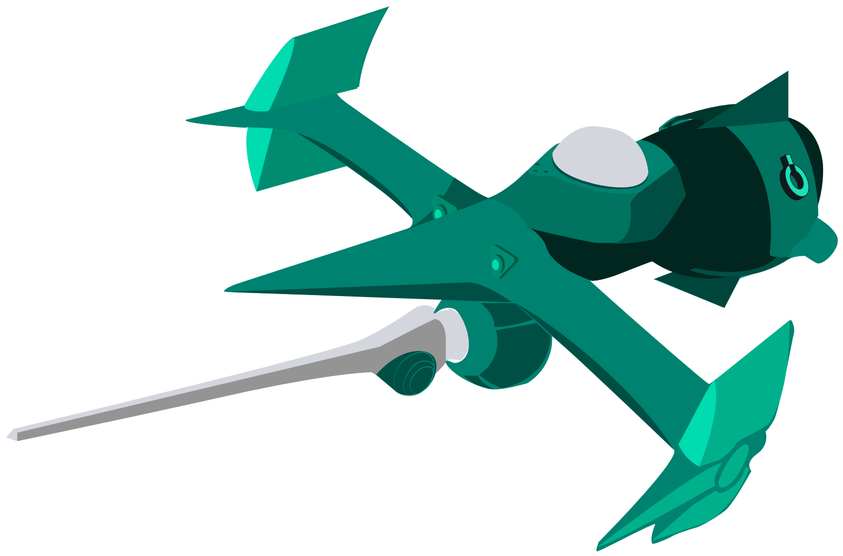


Tabela de Fusos Horários

Formato			
America/Bogota	America/Monterrey	Australia/Sydney	Europe/Monaco
America/Boise	America/New_York	Brazil/Acre	Europe/Moscow
America/Buenos_Aires	America/Noronha	Brazil/DeNoronha	Europe/Paris
America/Guayaquil	America/North_Dakota/Beulah	Brazil/East	Europe/Rome
America/Guyana	America/Recife	Brazil/West	Europe/Vienna
America/Lima	America/Sao_Paulo	Europe/Amsterdam	
America/Los_Angeles	America/Vancouver	Europe/Berlin	
America/Maceio	Asia/Dubai	Europe/Kiev	
America/Manaus	Asia/Shanghai	Europe/London	
America/Mexico_City	Asia/Tokyo	Europe/Madrid	

Fusos Horários conseguidos a partir desse código.

```
import pytz
for tz in pytz.all_timezones:
    print(tz)
```

Mundo 9

Tabela de Encoding

Codec	Languages
ascii	English
big5	Traditional Chinese
big5hkscs	Traditional Chinese
cp037	English
cp273	German New in version 3.4.
cp424	Hebrew
cp437	English
cp500	Western Europe
cp720	Arabic
cp737	Greek
cp775	Baltic languages
cp850	Western Europe
cp852	Central and Eastern Europe
cp855	Bulgarian, Byelorussian, Macedonian, Russian, Serbian
cp856	Hebrew
cp857	Turkish
cp858	Western Europe
cp860	Portuguese
cp861	Icelandic
cp862	Hebrew
cp863	Canadian
cp864	Arabic
cp865	Danish, Norwegian
cp866	Russian
cp869	Greek
cp874	Thai
cp875	Greek
cp932	Japanese

cp949	Korean
cp950	Traditional Chinese
cp1006	Urdu
cp1026	Turkish
cp1125	Ukrainian New in version 3.4.
cp1140	Western Europe
cp1250	Central and Eastern Europe
cp1251	Bulgarian, Byelorussian, Macedonian, Russian, Serbian
cp1252	Western Europe
cp1253	Greek
cp1254	Turkish
cp1255	Hebrew
cp1256	Arabic
cp1257	Baltic languages
cp1258	Vietnamese
euc_jp	Japanese

euc_jis_2004	Japanese
euc_jisx0213	Japanese
euc_kr	Korean
gb2312	Simplified Chinese
gbk	Unified Chinese
gb18030	Unified Chinese
hz	Simplified Chinese
iso2022_jp	Japanese
iso2022_jp_1	Japanese
iso2022_jp_2	Japanese, Korean, Simplified Chinese, Western Europe, Greek
iso2022_jp_2004	Japanese
iso2022_jp_3	Japanese
iso2022_jp_ext	Japanese
iso2022_kr	Korean
latin_1	Western Europe
iso8859_2	Central and Eastern Europe

iso8859_3	Esperanto, Maltese
iso8859_4	Baltic languages
iso8859_5	Bulgarian, Byelorussian, Macedonian, Russian, Serbian
iso8859_6	Arabic
iso8859_7	Greek
iso8859_8	Hebrew
iso8859_9	Turkish
iso8859_10	Nordic languages
iso8859_11	Thai languages
iso8859_13	Baltic languages
iso8859_14	Celtic languages
iso8859_15	Western Europe
iso8859_16	South-Eastern Europe
johab	Korean
koi8_r	Russian
koi8_t	Tajik New in version 3.5.
koi8_u	Ukrainian
kz1048	Kazakh New in version 3.5.
mac_cyrillic	Bulgarian, Byelorussian, Macedonian, Russian, Serbian
mac_greek	Greek
mac_iceland	Icelandic
mac_latin2	Central and Eastern Europe
mac_roman	Western Europe
mac_turkish	Turkish
ptcp154	Kazakh
shift_jis	Japanese
shift_jis_2004	Japanese
shift_jisx0213	Japanese
utf_32	all languages
utf_32_be	all languages
utf_32_le	all languages
utf_16	all languages

utf_16_be	all languages
utf_16_le	all languages
utf_7	all languages
utf_8	all languages
utf_8_sig	all languages