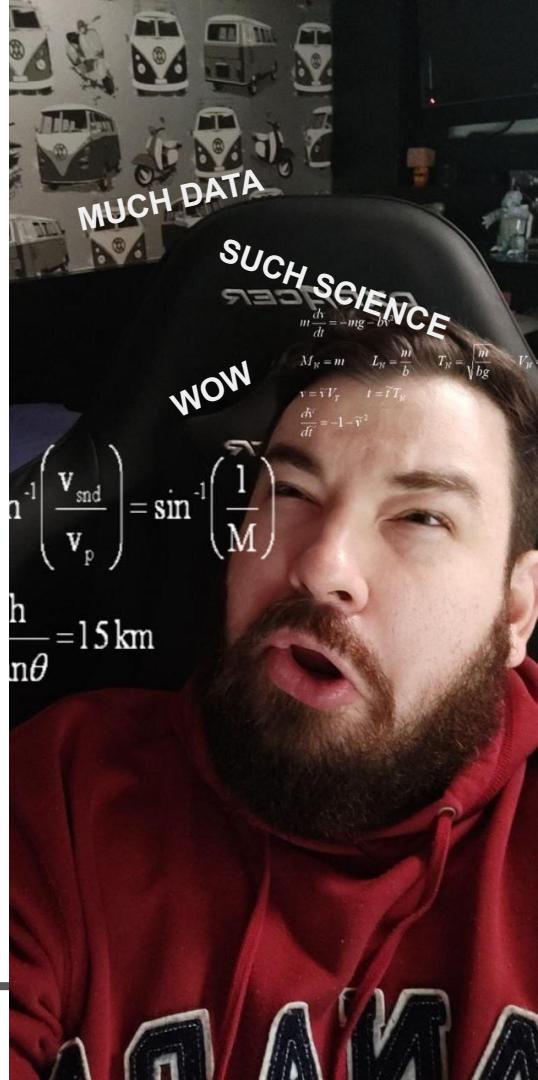


# Ferramentas de Softwares para uso em ciência de Dados II

# OBJETIVO

## Nasser Santiago Boan

- > arquivologia - UNB
- > tecnologia do Petróleo e Gás - UFRJ
- > MBA Processos de Negócio - IBMEC
- > IBM Data Science Professional Certified
- > líder de Ciência de Dados - Certsys
- > Cientista de Dados - Stefanini
- > docente Pós-Graduação Ciência de Dados



# Ementa

- Básico
  - pip
  - conda
  - miniconda
  - jupyter notebook
- Nivelamento 2
  - Variáveis e operações matemáticas
  - Operações com strings
  - Listas
  - Tuplas
  - Dicionários
  - Condições lógicas
  - Loops (for + compreensão de lista + while)
  - Funções (def + lambda)
  - Funções built-in (zip, enumerate, map e filter)



# Ementa

- Files
  - Lendo arquivos com OPEN
  - Escrevendo arquivos com OPEN
- Numpy
  - Numpy array
  - Operações com arrays
  - Indexing e slicing
  - Stacking
  - Funções estatísticas (mean, var, std, median, quantiles )

# Ementa

- Pandas
  - Dataframes
  - read\_ / to\_
  - Indexing e slicing (.loc , .iloc , filter , head, sample )
  - Tipos de dados e datas (to\_datetime)
  - Funções estatísticas (describe, corr, skew, mode, unique, nunique)
  - Transformações (apply + operações com séries)
  - Reshaping e sorting ( pivot + transpose + nlargest + nsmallest + value\_counts + sort\_values )
  - Concat e merge
  - Valores nulos (isna, isnull, fillna-- imputação --, drop, dropna, replace)
  - Method Chaining

# Ementa

- matplotlib
  - Arquitetura (pyplot e axes)
  - Anatomia dos gráficos em matplotlib
  - Plotando (figure, plt.bar, plt.scatter, plt.plot, title, labels, legend, colors, axvline, axhline )
  - Subplots
  - Abrindo imagens
  - Exportando gráficos (plt.savefig)

# Ementa

- Scikit-Learn
  - Pre-processamento (standartscaler, minmaxscaler, onehotencoding)
  - Conceitos de Regressão / Classificação / Clusterização
    - Dataset de treino e teste (train\_test\_split)
    - Underfitting / overfitting
  - Regressão linear simples (na mão + coeficientes explícitos)
    - Métricas de regressão e função de custo
  - Regressão linear múltipla e polynomial features
  - DecisionTree / RandomForest
    - Métricas de classificação
  - KMeans
    - métricas de clusterização
  - Kfold e GridsearchCV

# Avaliação

- Lista 01 ( python )
- Lista 02 ( pandas + matplotlib )
- Projeto 01 ( análise de dados )
- Lista 03 ( conceitos de machine learning 1 )
- Lista 04 ( conceitos de machine learning 2 )
- Projeto 02 ( projeto ML)

# Módulo Básico

# pip

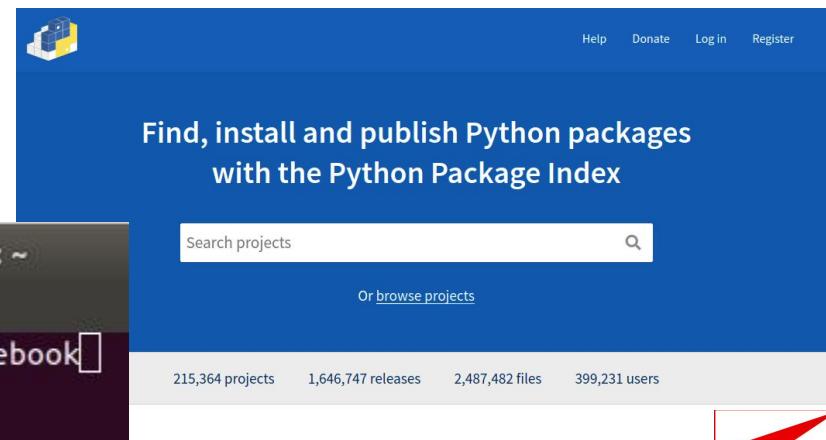
O pip é o gerenciador de pacotes do python.

```
$ pip install <nome do pacote>
```

```
$ pip --version
```

```
$ pip list
```

```
$ pip install -r requirements.txt
```



# conda e miniconda

O conda vai além e gerênciambientes e dependências

```
$ conda create env -n <nome do ambiente>
```

```
$ conda create env -n <nome do ambiente> python=3.6
```

```
$ conda env create -f environment.yml
```

```
$ conda list
```

```
$ conda install <nome do pacote>
```

```
$ conda activate <nome do ambiente>
```

```
$ conda deactivate <nome do ambiente>
```



# conda e miniconda

O conda vai além e gerênciambientes e dependências

```
(base) nzboan@ubutop:~$ conda list
# packages in environment at /home/nzboan/miniconda3:
#
# Name           Version    Build  Channel
_libgcc_mutex   0.1        main
_tflow_select   2.1.0      gpu
absl-py         0.8.1      py37_0
altair          3.2.0      py37_0
asnincrypto     1.2.0      py37_0
astor           0.8.0      py37_0
astroid          2.3.3      py37_0
attrs            19.3.0     py_0
backcall         0.1.0      py37_0
beautifulsoup4  4.8.1      py37_0
biopython        1.74       pypi_0  pypi
blas             1.0        mkl
bleach           3.1.0      py37_0
blosc            1.16.3     hd408876_0
bokeh            1.4.0      py37_0
boto3            1.9.232    pypi_0  pypi
botocore         1.12.232   pypi_0  pypi
branca           0.3.1      py_0
bzzip2           1.0.8      h7b6447c_0
c-ares            1.15.0     h7b6447c_1001
ca-certificates  2019.11.27  0
cachetools       3.1.1      pypi_0  pypi
cairo             1.14.12    h8948797_3
certifi           2019.11.28  py37_0
cffi              1.13.2     py37h2e261b9_0
chardet          3.0.4      py37_1003
click             7.0        py37_0
click-plugins    1.1.1      pypi_0  pypi
cligj             0.5.0      pypi_0  pypi
cloudpickle      1.2.2      py_0
cogroo-interface 0.3        pypi_0  pypi
colorcet          2.0.2      py_0
conda            4.8.1      py37_0
```



# conda e miniconda

O conda vai além e gerênciambientes e dependências

```
(base) nzboan@ubutop:~$ conda activate new
(base) nzboan@ubutop:~$ conda list
# packages in environment at /home/nzboan/miniconda3/envs/new:
#
# Name           Version      Build Channel
(base) nzboan@ubutop:~$
```

```
In [5]: from sklearn.datasets import load_iris
```

```
-----
ModuleNotFoundError          Traceback (most recent call
last)
<ipython-input-5-56d11ecab3a7> in <module>
----> 1 from sklearn.datasets import load_iris

ModuleNotFoundError: No module named 'sklearn'
```

# conda e miniconda

```
(new) nzboan@ubutop:~$ conda install scikit-learn
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

environment location: /home/nzboan/miniconda3/envs/new

added / updated specs:
- scikit-learn

The following packages will be downloaded:

      package          build
-----
ca-certificates-2020.1.1           0
certifi-2019.11.28                  py38_0
ld_impl_linux-64-2.33.1            h53a641e_7
mkl-service-2.3.0                   py38he904b0f_0
mkl_fft-1.0.15                     py38ha843d7b_0
mkl_random-1.1.0                   py38h962f231_0
numpy-1.18.1                       py38hf9e942_0
numpy-base-1.18.1                  py38hde5b4d6_1
openssl-1.1.1d                     h7b6447c_3
pip-20.0.2                          py38_1
python-3.8.1                        h0371630_1
scikit-learn-0.22.1                 py38hd81dba3_0
scipy-1.3.2                         py38h7c811a0_0
setuptools-45.1.0                   py38_0
six-1.14.0                          py38_0
sqlite-3.30.1                      h7b6447c_0
wheel-0.34.1                        py38_0
-----
                                         Total: 8
```

```
In [2]: from sklearn.datasets import load_iris
```

```
In [9]: data = load_iris()
```

```
In [10]: data.target
```



# conda e miniconda

```
! environment.yml
1  name: stats
2  dependencies:
3    - jupyterlab
4    - pandas
5    - numpy
6    - scikit-learn
7
```

```
(base) nzboan@ubutop:~$ conda env create -f environment.yml
(base) nzboan@ubutop:~$ conda env create -f environment.yml
Collecting package metadata (repodata.json): done
Solving environment: done

Downloading and Extracting Packages
pandas-1.0.0      | 8.8 MB      | #####| 100%
jupyterlab-1.2.5  | 2.8 MB      | #####| 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate stats
#
# To deactivate an active environment, use
#
#     $ conda deactivate
(base) nzboan@ubutop:~$
```

# jupyter notebook

O jupyter notebook é documento que contém tanto código quanto elementos de texto. Ele executa código célula a célula, pode ser lido por humanos (human-readable) ou pela máquina (machine-readable).

# google colab

Welcome To Colaboratory

File Edit View Insert Runtime Tools Help

Share Connect Editing

+ Code + Text Copy to Drive

## What is Colaboratory?

Colaboratory, or "Colab" for short, allows you to write and execute Python in your browser, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch [Introduction to Colab](#) to learn more, or just get started below!

### Getting started

The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
[ ] 1 seconds_in_a_day = 24 * 60 * 60
2 seconds_in_a_day
```

86400

To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut "Command/Ctrl+Enter". To edit the code, just click the cell and start editing.

Variables that you define in one cell can later be used in other cells:

```
[ ] 1 seconds_in_a_week = 7 * seconds_in_a_day
2 seconds_in_a_week
```

604800

# referências importantes

## instalação do conda

<https://docs.conda.io/projects/conda/en/latest/user-guide/install/index.html#regular-installation>

## instalação do pip

<https://pip.pypa.io/en/stable/installing/>

## instalação do jupyter notebook

<https://jupyter.org/install.html>

## google colab

<https://colab.research.google.com/>

# Nivelamento (again...)

<https://github.com/aulas-iesb/CastoresIndomaveis>

# Files e Numpy

# File Handling

Em python não há necessidade de importar bibliotecas para ler e escrever arquivos. As funções built-in open e write resolvem bem a grande maioria das necessidades.

The screenshot shows a Jupyter Notebook interface. On the left is a file browser with a folder named 'files-numpy /'. It contains two items: 'mod-files-numpy-file-handling.ipynb' (modified 'seconds ago') and 'primeiro\_arquivo.txt' (modified 'a minute ago'). A red arrow points from the 'primeiro\_arquivo.txt' entry to the code cell on the right. The code cell at the top right contains the Python code: [1]: `f = open('primeiro_arquivo.txt', 'w+')`. Below it is an empty output cell: [ ]:.

# open()

A função open() abre um arquivo e retorna um objeto. A função possui vários modos e o objeto pode ou não ser alterado de acordo com o modo passado. Os modos que a função pode receber:

Modo	Descrição
'r'	Modo padrão da função. Abre um arquivo para leitura.
'w'	Esse modo abre um arquivo para escrita. Se o arquivo não existir esse modo cria o arquivo. Se o arquivo já existe ele substitui o arquivo.
'x'	Cria um arquivo novo. Se o arquivo já existir a função retorna um erro
'a'	Abre um arquivo no modo 'append'. Se o arquivo não existe ele cria um novo.
'+'	Esse modo abre um arquivo para leitura e escrita.

# read()

```
[33]: f = open('lorem.txt','r')
conteudo = f.read()
f.close()
```

```
[35]: print(conteudo)
```

  Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam sagittis quis massa sed euismod. Aenean at mattis nisl. Vestibulum a imperdiet lacus. Duis hendrerit, justo maximus convallis placerat, purus nibh vulputate nibh, ac varius libero est eu orci. Fusce ac dolor id ex dictum mattis. Donec f inibus pellentesque ligula, et semper tellus varius maximus. In rutrum vitae quam nec suscipit. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nullam laoreet dolor non felis tincidunt dignissim. Suspendisse lacinia accumsan risus vitae rhoncus. Nulla luctus con vallis enim, ut blandit arcu dignissim at. Pellentesque pharetra ornare efficitur. In molestie, quam in tristique ullamcorper, purus nulla accumsan m i, et lobortis purus magna quis risus. Cras magna ipsum, accumsan in purus eget, tincidunt feugiat est. Proin laoreet velit a nunc ullamcorper varius.

  Cras vel pharetra lacus. In imperdiet at erat ac suscipit. Donec eget urna vel lectus fermentum maximus. Quisque eu tempus nisl. Sed ultricies lacus q uis condimentum efficitur. Quisque non metus at eros eleifend eleifend. Fusce et semper nisi, lobortis consequat erat. Praesent condimentum pulvinar n unc in scelerisque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

  Duis in maximus lacus, vel eleifend ligula. Integer in elementum leo. Nam interdum pharetra neque, a lacinia arcu. Donec scelerisque velit sit amet ju sto sagittis, eu commodo quam viverra. Vivamus non est viverra mi vehicula scelerisque. Donec id suscipit ligula. Nullam condimentum semper elit, non accumsan justo consectetur a. Quisque non nisl elit. Ut ultrices risus ut tortor tincidunt accumsan. Curabitur aliquam mi nec mi aliquam lacinia. Maec enas eget felis non mi ultrices suscipit.

# readlines()

```
[37]: f = open('lines.txt','r')
conteudo = f.readlines()
for i in conteudo:
    print(i)
```

linha 1

linha 2

linha 3

linha 4

linha 5

linha 6

# write()

```
[51]: with open('primeiro_arquivo.txt', 'w') as f:  
    for i in range(10):  
        f.write(f'{i}\n')
```

The screenshot shows a Jupyter Notebook interface. The top bar has two tabs: 'mod-files-numpy-file-handlir X' and 'primeiro\_arquivo.txt'. The main area displays a code cell with the following content:

```
1 | 0  
2 | 1  
3 | 2  
4 | 3  
5 | 4  
6 | 5  
7 | 6  
8 | 7  
9 | 8  
10| 9  
11|
```

The code cell contains a Python script that uses a 'with' statement to open a file named 'primeiro\_arquivo.txt' in write mode ('w'). It then enters a 'for' loop that iterates 10 times, writing each number from 0 to 9 followed by a new line character ('\n') to the file.

# csv.reader()

```
[38]: import csv

[40]: with open('avocado.csv') as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        for row in csv_reader:
            print(row)

[', 'Date', 'AveragePrice', 'Total Volume', '4046', '4225', '4770', 'Total Bags', 'Small Bags', 'Large Bags', 'XLarge Bags', 'type', 'year', 'region']
['0', '2015-12-27', '1.33', '64236.62', '1036.74', '54454.85', '48.16', '8696.87', '8603.62', '93.25', '0.0', 'conventional', '2015', 'Albany']
['1', '2015-12-20', '1.35', '54876.98', '674.28', '44638.81', '58.33', '9505.56', '9408.07', '97.49', '0.0', 'conventional', '2015', 'Albany']
['2', '2015-12-13', '0.93', '118220.22', '794.7', '109149.67', '130.5', '8145.35', '8042.21', '103.14', '0.0', 'conventional', '2015', 'Albany']
['3', '2015-12-06', '1.08', '78992.15', '1132.0', '71976.41', '72.58', '5811.16', '5677.4', '133.76', '0.0', 'conventional', '2015', 'Albany']
['4', '2015-11-29', '1.28', '51039.6', '941.48', '43838.39', '75.78', '6183.95', '5986.26', '197.69', '0.0', 'conventional', '2015', 'Albany']
```

# csv.reader()

```
[44]: avg_price = []

    with open('avocado.csv') as csv_file:
        csv_reader = csv.reader(csv_file,delimiter=',')
        next(csv_reader)
        for row in csv_reader:
            avg_price.append(row[2])

[47]: avg_price[:10]

[47]: ['1.33',
       '1.35',
       '0.93',
       '1.08',
       '1.28',
       '1.26',
       '0.99',
       '0.98',
       '1.02',
       '1.07']
```

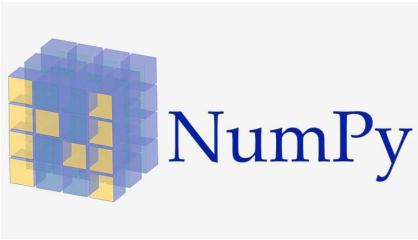
# csv.writer()

```
[76]: import csv

with open('empregados.csv', 'w') as f:
    empregados_writer = csv.writer(f, delimiter=',', quotechar='''')

    empregados_writer.writerow(['nome', 'idade', 'salario'])
    empregados_writer.writerow(['nasser', 29, 1000])
    Delimiter: ,
```

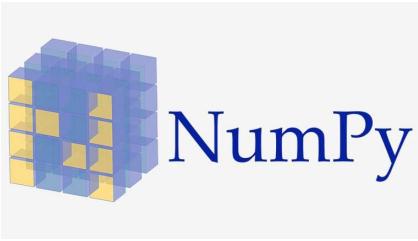
	nome	idade	salario
1	nasser	29	1000
2	joao	20	1500



# NumPy

```
[2]: import numpy as np  
  
a = np.array([[1,2,2],[2,2,2]])  
a
```

```
[2]: array([[1, 2, 2],  
           [2, 2, 2]])
```



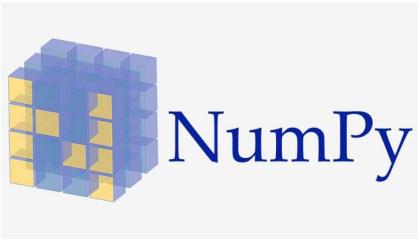
# NumPy

```
[3]: a.ndim
```

```
[3]: 2
```

```
[31]: np.zeros((40,50,50,50)).ndim
```

```
[31]: 4
```



# NumPy

```
[38]: a.shape
```

```
[38]: (2, 3)
```

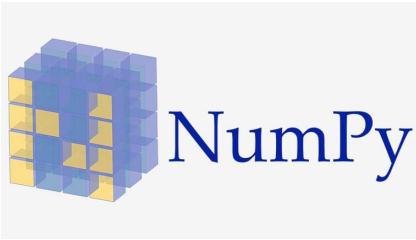
```
[13]: b = np.array([[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15]])
```

```
b
```

```
[13]: array([[ 1,  2,  3,  4,  5],  
           [ 6,  7,  8,  9, 10],  
           [11, 12, 13, 14, 15]])
```

```
[39]: b.shape
```

```
[39]: (3, 5)
```



# NumPy

```
[43]: a.size
```

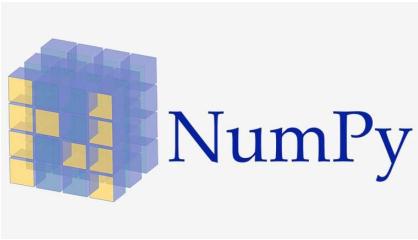
```
[43]: 6
```

```
[44]: b.size
```

```
[44]: 15
```

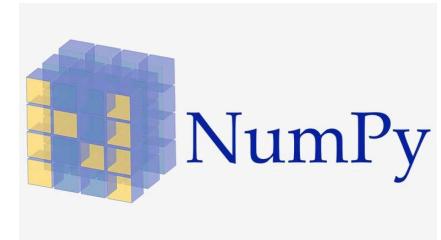
```
[42]: np.zeros((40,50,50,50)).size
```

```
[42]: 5000000
```



```
[61]: a + 1  
  
[61]: array([[2, 3, 3],  
           [3, 3, 3]])  
  
[62]: a ** 2  
  
[62]: array([[1, 4, 4],  
           [4, 4, 4]])  
  
[63]: a - 10  
  
[63]: array([[-9, -8, -8],  
           [-8, -8, -8]])  
  
[66]: (a*3) > 5  
  
[66]: array([[False,  True,  True],  
           [ True,  True,  True]])
```

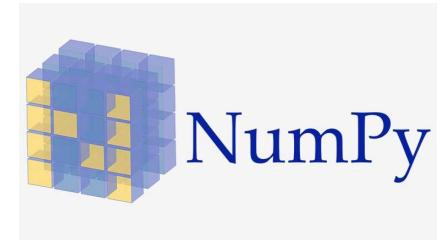
# operações



```
[52]: a + b
-----
ValueError                                Traceback (most recent call last)
<ipython-input-52-bd58363a63fc> in <module>
      1 a + b
ValueError: operands could not be broadcast together with shapes (2,3) (3,5)
```

```
[57]: c = np.array([[2,1,1],[1,1,1]])
c
[57]: array([[2, 1, 1],
           [1, 1, 1]])
```

# operações



```
[57]: c = np.array([[2,1,1],[1,1,1]])
c
```

```
[57]: array([[2, 1, 1],
           [1, 1, 1]])
```

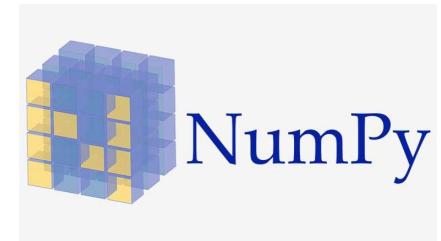
```
[56]: a + c
```

```
[56]: array([[3, 3, 3],
           [3, 3, 3]])
```

```
[67]: a - c
```

```
[67]: array([[-1,  1,  1],
           [ 1,  1,  1]])
```

# operações



```
[69]: array1 = np.array([[1,2,3],[4,5,6],[7,8,9]])  
array1
```

```
[69]: array([[1, 2, 3],  
           [4, 5, 6],  
           [7, 8, 9]])
```

```
[71]: array2 = np.array([[  
array2
```

```
[71]: array([[5, 5, 5, 5,  
           [5, 5, 5, 5,  
           [5, 5, 5, 5,
```

```
[73]: array1@array2
```

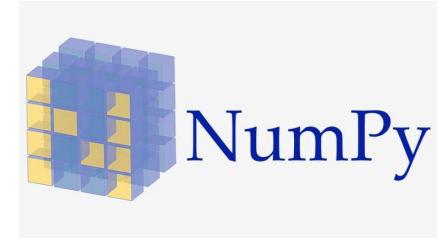
```
[73]: array([[ 30,  30,  30,  30,  30,  30],  
           [ 75,  75,  75,  75,  75,  75],  
           [120, 120, 120, 120, 120, 120]])
```

```
[79]: a = np.arange(12).reshape(3,4)  
a
```

```
[79]: array([[ 0,  1,  2,  3],  
           [ 4,  5,  6,  7],  
           [ 8,  9, 10, 11]])
```

<http://matrixmultiplication.xyz/>

# indexing & slicing



```
[79]: a = np.arange(12).reshape(3,4)  
a
```

```
[79]: array([[ 0,  1,  2,  3],  
           [ 4,  5,  6,  7],  
           [ 8,  9, 10, 11]])
```

```
[82]: a[1][2]
```

```
[82]: 6
```

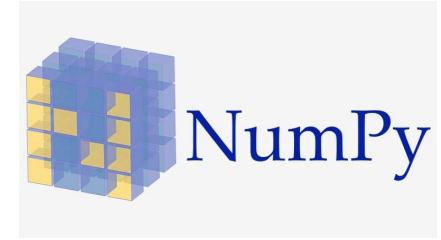
```
[83]: a[:2]
```

```
[83]: array([[0, 1, 2, 3],  
           [4, 5, 6, 7]])
```

```
[86]: a[::-2]
```

```
[86]: array([[ 0,  1,  2,  3],  
           [ 8,  9, 10, 11]])
```

# indexing & slicing



```
[87]: a[:::-1]
```

```
[87]: array([[ 8,  9, 10, 11],
       [ 4,  5,  6,  7],
       [ 0,  1,  2,  3]])
```

```
[89]: index_bol = a > 5
```

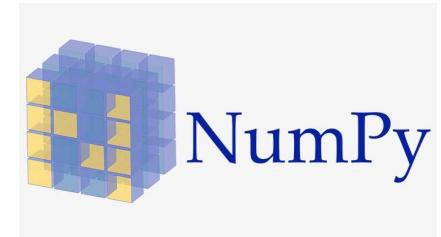
```
[90]: a[index_bol]
```

```
[90]: array([ 6,  7,  8,  9, 10, 11])
```

```
[91]: a
```

```
[91]: array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

# stacking



```
a
```

```
[79]: array([[ 0,  1,  2,  3],  
           [ 4,  5,  6,  7],  
           [ 8,  9, 10, 11]])
```

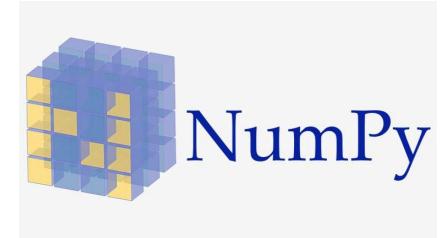
```
[96]: b
```

```
[96]: array([[ 0,  1,  2,  3],  
           [ 4,  5,  6,  7],  
           [ 8,  9, 10, 11],  
           [12, 13, 14, 15]])
```

```
[99]: np.vstack_(a,b)
```

```
[99]: array([[ 0,  1,  2,  3],  
           [ 4,  5,  6,  7],  
           [ 8,  9, 10, 11],  
           [ 0,  1,  2,  3],  
           [ 4,  5,  6,  7],  
           [ 8,  9, 10, 11],  
           [12, 13, 14, 15]])
```

# funções estatísticas

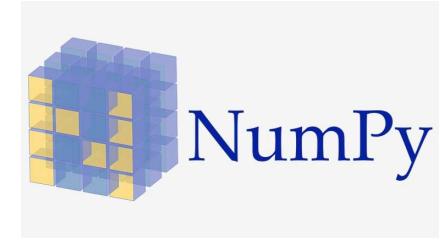


```
[101]: a = np.arange(24).reshape(8,3)

[102]: a

[102]: array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11],
       [12, 13, 14],
       [15, 16, 17],
       [18, 19, 20],
       [21, 22, 23]])
```

# funções estatísticas



```
[103]: a.mean()  
[103]: 11.5  
  
[104]: a.var()  
[104]: 47.916666666666664  
  
[105]: a.std()  
[105]: 6.922186552431729  
  
[106]: np.median(a)  
[106]: 11.5  
  
[109]: np.quantile(a,0.25)  
[109]: 5.75
```



# Pandas

Pandas é o melhor amigo do Cientista de Dados! Um pacote para a manipulação fácil e rápida de estruturas de dados “relacionais” de forma tabular. Suas duas estruturas de dados principais são a série (1d) e o dataframe (2d).

In [33]: data

Out[33]:

	Area Abbreviation	Area Code	Area	Item Code	Item	Element Code	Element	Unit	latitude	longitude	...	Y2004	Y2005	Y2006	Y2007	Y2008	Y2009
0	AF	2	Afghanistan	2511	Wheat and products	5142	Food	1000 tonnes	33.94	67.71	...	3249.0	3486.0	3704.0	4164.0	4252.0	4538.0
1	AF	2	Afghanistan	2805	Rice (Milled Equivalent)	5142	Food	1000 tonnes	33.94	67.71	...	419.0	445.0	546.0	455.0	490.0	415.0
2	AF	2	Afghanistan	2513	Barley and products	5521	Feed	1000 tonnes	33.94	67.71	...	58.0	236.0	262.0	263.0	230.0	379.0
3	AF	2	Afghanistan	2513	Barley and products	5142	Food	1000 tonnes	33.94	67.71	...	185.0	43.0	44.0	48.0	62.0	55.0
4	AF	2	Afghanistan	2514	Maize and products	5521	Feed	1000 tonnes	33.94	67.71	...	120.0	208.0	233.0	249.0	247.0	195.0
5	AF	2	Afghanistan	2514	Maize and products	5142	Food	1000 tonnes	33.94	67.71	...	231.0	67.0	82.0	67.0	69.0	71.0
6	AF	2	Afghanistan	2517	Millet and products	5142	Food	1000 tonnes	33.94	67.71	...	15.0	21.0	11.0	19.0	21.0	18.0
7	AF	2	Afghanistan	2520	Cereals, Other	5142	Food	1000 tonnes	33.94	67.71	...	2.0	1.0	1.0	0.0	0.0	0.0
8	AF	2	Afghanistan	2531	Potatoes and products	5142	Food	1000 tonnes	33.94	67.71	...	276.0	294.0	294.0	260.0	242.0	250.0
9	AF	2	Afghanistan	2536	Sugar cane	5521	Feed	1000 tonnes	33.94	67.71	...	50.0	29.0	61.0	65.0	54.0	114.0
10	AF	2	Afghanistan	2537	Sugar beet	5521	Feed	1000 tonnes	33.94	67.71	...	0.0	0.0	0.0	0.0	0.0	0.0

<https://www.kaggle.com/merishnasuwal/breast-cancer-prediction-dataset>

# dataframe (df)

Um dataframe é uma estrutura de dados única do Pandas. Ela é formada por um conjunto de séries.

## pandas.DataFrame

```
class pandas.DataFrame(data=None, index: Optional[Collection] = None, columns:  
Optional[Collection] = None, dtype: Union[str, numpy.dtype, ExtensionDtype, None] = None,  
copy: bool = False)
```

[\[source\]](#)

# dataframe (df)

Um dataframe é uma estrutura de dados única do Pandas. Ela é formada por um conjunto de séries.

O Dataframe

```
[20]: ## criando um dataframe

data = np.array([['a', '0'], ['b', '1'], ['c', '2'], ['d', '3'], ['e', '4'], ['f', '5'], ['g', '6'], ['h', '7'], ['i', '8']])

df = pd.DataFrame(data,columns=['letras', 'numeros'])
df
```

	letras	numeros
0	a	0
1	b	1
2	c	2
3	d	3
4	e	4
5	f	5
6	g	6
7	h	7
8	i	8

# Series

A série é um conjunto de dados, iterável e composta por um array numpy.

## pandas.Series

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

[source]

# Series

A série é um conjunto de dados, iterável e composta por um array numpy.

```
[21]: ## Acessando a serie 'letras'  
df.letras
```

```
[21]: 0    a  
      1    b  
      2    c  
      3    d  
      4    e  
      5    f  
      6    g  
      7    h  
      8    i  
  
Name: letras, dtype: object
```

```
[23]: ## Acessando a serie 'letras'  
df['letras']
```

```
[23]: 0    a  
      1    b  
      2    c  
      3    d  
      4    e  
      5    f  
      6    g  
      7    h  
      8    i  
  
Name: letras, dtype: object
```

# read\_csv()

## pandas.read\_csv

```
pandas.read_csv(filepath_or_buffer, Union[str, pathlib.Path, IO[~AnyStr]], sep=',',
delimiter=None, header='infer', names=None, index_col=None, usecols=None, squeeze=False,
prefix=None, mangle_dupe_cols=True, dtype=None, engine=None, converters=None,
true_values=None, false_values=None, skipinitialspace=False, skiprows=None, skipfooter=0,
nrows=None, na_values=None, keep_default_na=True, na_filter=True, verbose=False,
skip_blank_lines=True, parse_dates=False, infer_datetime_format=False, keep_date_col=False,
date_parser=None, dayfirst=False, cache_dates=True, iterator=False, chunkszie=None,
compression='infer', thousands=None, decimal: str = '.', lineterminator=None, quotechar='',
quoting=0, doublequote=True, escapechar=None, comment=None, encoding=None,
dialect=None, error_bad_lines=True, warn_bad_lines=True, delim_whitespace=False,
low_memory=True, memory_map=False, float_precision=None)
```

[source]

# read\_csv()

```
[1] 1 import pandas as pd  
  
[7] 1 df = pd.read_csv('/content/Breast_cancer_data.csv')  
2 df.head()
```

	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	diagnosis
0	17.99	10.38	122.80	1001.0	0.11840	0
1	20.57	17.77	132.90	1326.0	0.08474	0
2	19.69	21.25	130.00	1203.0	0.10960	0
3	11.42	20.38	77.58	386.1	0.14250	0
4	20.29	14.34	135.10	1297.0	0.10030	0

# to\_csv()

## pandas.DataFrame.to\_csv

```
DataFrame.to_csv(self, path_or_buf: Union[str, pathlib.Path, IO[~AnyStr], NoneType] = None,  
sep: str = ',', na_rep: str = "", float_format: Union[str, NoneType] = None, columns:  
Union[Sequence[Union[Hashable, NoneType]], NoneType] = None, header: Union[bool, List[str]]  
= True, index: bool = True, index_label: Union[bool, str, Sequence[Union[Hashable, NoneType]],  
NoneType] = None, mode: str = 'w', encoding: Union[str, NoneType] = None, compression:  
Union[str, Mapping[str, str], NoneType] = 'infer', quoting: Union[int, NoneType] = None,  
quotechar: str = "", line_terminator: Union[str, NoneType] = None, chunksize: Union[int,  
NoneType] = None, date_format: Union[str, NoneType] = None, doublequote: bool = True,  
escapechar: Union[str, NoneType] = None, decimal: Union[str, NoneType] = '.') → Union[str,  
NoneType]
```

[source]

# to\_csv()



	3	11.42	20.38	77.58
	4	20.29	14.34	135.10
[8]	1	df.to_csv('meu_dataset.csv',sep=';')		

# indexing/slicing

<nome do dataframe>[<nome da coluna>]

<nome do dataframe>.<nome da coluna>

```
[9] 1 df['mean_radius']

 0    17.99
 1    20.57
 2    19.69
 3    11.42
 4    20.29
 ...
564   21.56
565   20.13
566   16.60
567   20.60
568    7.76
Name: mean_radius, Length: 569, dtype: float64
```

# indexing/slicing

df.loc[ <linha>, <coluna> ]

```
[7] 1 df = pd.read_csv('/content/Breast_cancer_data.csv')
2 df.head()
```

	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	diagnosis
0	17.99	10.38	122.80	1001.0	0.11840	0
1	20.57	17.77	132.90	1326.0	0.08474	0
2	19.69	21.25	130.00	1203.0	0.10960	0
3	11.42	20.38	77.58	386.1	0.14250	0
4	20.29	14.34	135.10	1297.0	0.10030	0

```
▶ 1 df.loc[1, 'mean_texture']
```

```
⌚ 17.77
```

# indexing/slicing

df.iloc[ <linha>, <coluna> ]

```
[7] 1 df = pd.read_csv('/content/Breast_cancer_data.csv')
2 df.head()

mean_radius mean_texture mean_perimeter mean_area mean_smoothness diagnosis
0 17.99 10.38 122.80 1001.0 0.11840 0
1 20.57 17.77 132.90 1326.0 0.08474 0
2 19.69 21.25 130.00 1203.0 0.10960 0
3 11.42 20.38 77.58 386.1 0.14250 0
4 20.29 14.34 135.10 1297.0 0.10030 0

[17] 1 df.iloc[1,1]

17.77
```

# indexing/slicing

df[[<lista com as colunas>]]

	mean_radius	mean_area	diagnosis
0	17.99	1001.0	0
1	20.57	1326.0	0
2	19.69	1203.0	0
3	11.42	386.1	0
4	20.29	1297.0	0
...	...	...	...
564	21.56	1479.0	0
565	20.13	1261.0	0
566	16.60	858.1	0
567	20.60	1265.0	0
568	7.76	181.0	1

569 rows × 3 columns

# indexing/slicing

## condicionais

```
[38]: ## slicing com condicionais  
  
df[df['diagnosis'] == 1]  
  
[38]:   mean_radius  mean_texture  mean_perimeter  mean_area  mean_smoothness  diagnosis  
      19          13.540        14.36          87.46       566.3         0.09779        1  
      20          13.080        15.71          85.63       520.0         0.10750        1  
      21          9.504         12.44          60.34       273.9         0.10240        1  
      37          13.030        18.42          82.61       523.8         0.08983        1  
      46           8.196        16.84          51.71       201.9         0.08600        1  
     ...           ...          ...            ...          ...          ...          ...  
      558          14.590        22.68          96.39       657.1         0.08473        1  
      559          11.510        23.93          74.52       403.5         0.09261        1  
      560          14.050        27.15          91.38       600.4         0.09929        1  
      561          11.200        29.37          70.67       386.0         0.07449        1  
      568          7.760         24.54          47.92       181.0         0.05263        1  
  
357 rows × 6 columns
```

# indexing/slicing

## condicionais

[40]: df[(df['diagnosis'] == 1) & (df['mean_area'] > 500)]						
	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	diagnosis
19	13.54	14.36	87.46	566.3	0.09779	1
20	13.08	15.71	85.63	520.0	0.10750	1
37	13.03	18.42	82.61	523.8	0.08983	1
49	13.49	22.30	86.91	561.0	0.08752	1
51	13.64	16.34	87.21	571.8	0.07685	1
...	...	...	...	...	...	...
545	13.62	23.23	87.19	573.2	0.09246	1
552	12.77	29.43	81.35	507.9	0.08276	1
554	12.88	28.92	82.50	514.3	0.08123	1
558	14.59	22.68	96.39	657.1	0.08473	1
560	14.05	27.15	91.38	600.4	0.09929	1
137 rows × 6 columns						

# indexing/slicing

## condicionais

Operador	Exemplo
& (AND)	<code>df[(df['coluna1'] == 0) &amp; (df['coluna2'] &gt;= 10)]</code>
(OR)	<code>df[(df['coluna2'] &lt;= 5)   (df['coluna2'] &gt;= 10)]</code>

# indexing/slicing

.head()

```
[41]: df.head()
```

	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	diagnosis
0	17.99	10.38	122.80	1001.0	0.11840	0
1	20.57	17.77	132.90	1326.0	0.08474	0
2	19.69	21.25	130.00	1203.0	0.10960	0
3	11.42	20.38	77.58	386.1	0.14250	0
4	20.29	14.34	135.10	1297.0	0.10030	0

.tail()

```
[42]: df.tail()
```

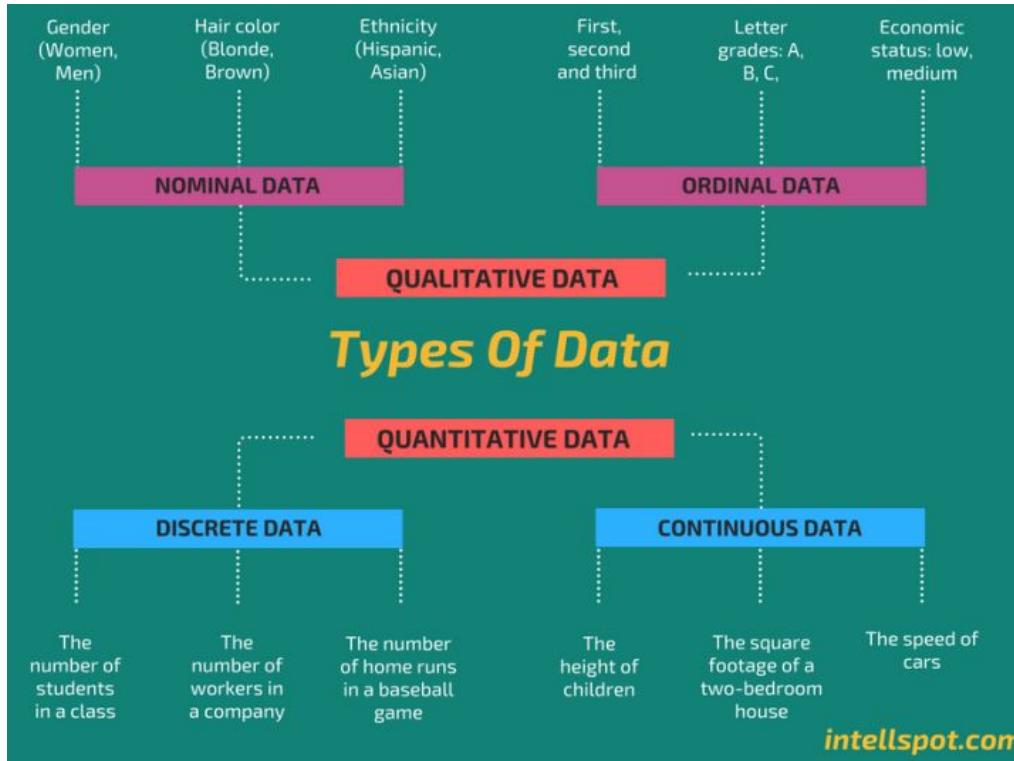
	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	diagnosis
564	21.56	22.39	142.00	1479.0	0.11100	0
565	20.13	28.25	131.20	1261.0	0.09780	0
566	16.60	28.08	108.30	858.1	0.08455	0
567	20.60	29.33	140.10	1265.0	0.11780	0
568	7.76	24.54	47.92	181.0	0.05263	1

# indexing/slicing

.sample()

```
[43]: ## colhendo uma amostra  
df.sample()  
[43]:   mean_radius  mean_texture  mean_perimeter  mean_area  mean_smoothness  diagnosis  
      48           12.05          14.63         78.04        449.3        0.1031            1
```

# tipos de dados



# datas

to\_datetime()

## pandas.to\_datetime

```
pandas.to_datetime(arg, errors='raise', dayfirst=False, yearfirst=False, utc=None, format=None, exact=True, unit=None,  
infer_datetime_format=False, origin='unix', cache=True)
```

[source]

# datas

## to\_datetime()

[46]:

```
df_data = pd.DataFrame(d)
df_data
```

	col1	col2
0	0	10-10-2020
1	1	02-01-2010
2	2	15-10-2015
3	3	15-10-2010
4	4	12-02-2012
5	5	12-12-2022
6	6	15-10-2210
7	7	20-12-2050
8	8	10-10-1910
9	9	15-10-2010

# datas

```
to_datetime [52]: pd.to_datetime(df_data['col2'],infer_datetime_format=True)

[52]: 0    2020-10-10
      1    2010-02-01
      2    2015-10-15
      3    2010-10-31
      4    2012-12-02
      5    2022-12-12
      6    2210-10-15
      7    2050-12-20
      8    1910-10-10
      9    2010-10-15
Name: col2, dtype: datetime64[ns]
```

# datas

## to\_datetime()

```
[52]: pd.to_datetime(df_data['col2'],infer_datetime_format=True)

[52]: 0    2020-10-10
      1    2010-02-01
      2    2015-10-15
      3    2010-10-31
      4    2012-12-02
      5    2022-12-12
      6    2210-10-15
      7    2050-12-20
      8    1910-10-10
      9    2010-10-15
Name: col2, dtype: datetime64[ns]
```

# Funções Estatísticas

df.describe()

## pandas.DataFrame.describe

`DataFrame.describe(self: ~FrameOrSeries, percentiles=None, include=None, exclude=None) → ~FrameOrSeries`

Generate descriptive statistics.

[\[source\]](#)

# Funções Estatísticas

df.describe()

```
[3] 1 df.describe()
```

	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	diagnosis
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.627417
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.483918
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.000000
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.000000
50%	13.370000	18.840000	86.240000	551.100000	0.095870	1.000000
75%	15.780000	21.800000	104.100000	782.700000	0.105300	1.000000
max	28.110000	39.280000	188.500000	2501.000000	0.163400	1.000000

# Funções Estatísticas

df.describe()

```
[3] 1 df.describe()
```

	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	diagnosis
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.627417
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.483918
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.000000
25%	Primeiro Quartil					
50%	Segundo Quartil ou Mediana					
75%	Terceiro Quartil					
max	28.110000	39.280000	188.500000	2501.000000	0.163400	1.000000

# Funções Estatísticas

df.corr()

pandas.DataFrame.corr

`DataFrame.Corr(self, method='pearson', min_periods=1) → 'DataFrame'`

[source]

Compute pairwise correlation of columns, excluding NA/null values.

# Funções Estatísticas

df.corr()

Parameters: method : {‘pearson’, ‘kendall’, ‘spearman’} or callable

Method of correlation:

- pearson : standard correlation coefficient
- kendall : Kendall Tau correlation coefficient
- spearman : Spearman rank correlation
- callable: callable with input two 1d ndarrays

# Funções Estatísticas

df.corr()

```
[5] 1 df.corr(method='pearson')
```

	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	diagnosis
mean_radius	1.000000	0.323782	0.997855	0.987357	0.170581	-0.730029
mean_texture	0.323782	1.000000	0.329533	0.321086	-0.023389	-0.415185
mean_perimeter	0.997855	0.329533	1.000000	0.986507	0.207278	-0.742636
mean_area	0.987357	0.321086	0.986507	1.000000	0.177028	-0.708984
mean_smoothness	0.170581	-0.023389	0.207278	0.177028	1.000000	-0.358560
diagnosis	-0.730029	-0.415185	-0.742636	-0.708984	-0.358560	1.000000

# Funções Estatísticas

df.skew()

## pandas.DataFrame.skew

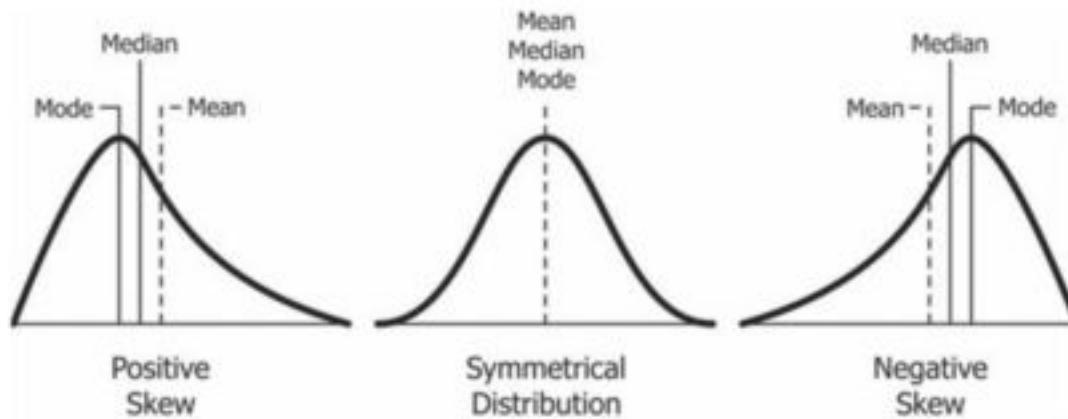
`DataFrame.skew(self, axis=None, skipna=None, level=None, numeric_only=None, **kwargs)`

[\[source\]](#)

Return unbiased `skew` over requested axis.

# Funções Estatísticas

Skewness descreve a simetria ou assimetria de uma distribuição!



# Funções Estatísticas

df.skew()

```
[7] 1 df.skew()

mean_radius      0.942380
mean_texture     0.650450
mean_perimeter   0.990650
mean_area        1.645732
mean_smoothness  0.456324
diagnosis        -0.528461
dtype: float64
```

# Funções Estatísticas

df.mode()

## pandas.DataFrame.mode

`DataFrame.mode(self, axis=0, numeric_only=False, dropna=True) → 'DataFrame'`

[\[source\]](#)

Get the mode(s) of each element along the selected axis.

# Funções Estatísticas

df.mode()

```
[2] 1 df = pd.read_csv('/content/train (3).csv')
```

```
[3] 1 df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	38.0	1	0	PC 17599 STON/O2. 3101282	71.2833 7.9250	C85 NaN	C S
2	3	1	3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	26.0	0	0	113803	53.1000	C123	S
3	4	1	1	Allen, Mr. William Henry	male	35.0	1	0	373450	8.0500	NaN	S
4	5	0	3									

# Funções Estatísticas

df.mode()

```
⌚ male      577  
    female    314  
    Name: Sex, dtype: int64
```

```
[4]   1   df.Sex.mode()
```

```
⌚ 0     male  
    dtype: object
```

# Funções Estatísticas

.unique()

## pandas.Series.unique

`Series.unique(self)`

[source]

Return unique values of Series object.

Uniques are returned in order of appearance. Hash table-based unique, therefore does NOT sort.

Returns: ndarray or ExtensionArray

The unique values returned as a NumPy array. See Notes.

# Funções Estatísticas

.unique()

```
[6] 1 df.Embarked.unique()  
[1] array(['S', 'C', 'Q', nan], dtype=object)
```

# Funções Estatísticas

.nunique()

## pandas.DataFrame.nunique

`DataFrame.nunique(self, axis=0, dropna=True) → pandas.core.series.Series`

[source]

Count distinct observations over requested axis.

Return Series with number of distinct observations. Can ignore NaN values.

**Parameters:** `axis : {0 or 'index', 1 or 'columns'}, default 0`

The axis to use. 0 or 'index' for row-wise, 1 or 'columns' for column-wise.

`dropna : bool, default True`

Don't include NaN in the counts.

**Returns:** Series

# Funções Estatísticas

.nunique()

```
[7] 1 df.nunique()

  ↗ PassengerId    891
  Survived          2
  Pclass            3
  Name              891
  Sex               2
  Age              88
  SibSp             7
  Parch             7
  Ticket            681
  Fare              248
  Cabin             147
  Embarked          3
  dtype: int64
```

# Sorting

.value\_counts()

## pandas.Series.value\_counts

`series.value_counts(self, normalize=False, sort=True, ascending=False, bins=None, dropna=True)`

[source]

Return a Series containing counts of unique values.

The resulting object will be in descending order so that the first element is the most frequently-occurring element. Excludes NA values by default.

# Sorting

.value\_counts()

```
[15] 1 df.diagnosis.value_counts()  
1    357  
0    212  
Name: diagnosis, dtype: int64
```

# Sorting

.value\_counts()

```
[16] 1 df.diagnosis.value_counts(normalize=True)

  ↗ 1    0.627417
  0    0.372583
Name: diagnosis, dtype: float64
```

# Transformações

.apply()

## pandas.DataFrame.apply

`DataFrame.apply(self, func, axis=0, raw=False, result_type=None, args=(), **kwds)`

[source]

Apply a function along an axis of the DataFrame.

Objects passed to the function are Series objects whose index is either the DataFrame's index (`axis=0`) or the DataFrame's columns (`axis=1`). By default (`result_type=None`), the final return type is inferred from the return type of the applied function. Otherwise, it depends on the `result_type` argument.

Parameters: `func : function`

Function to apply to each column or row.

# Transformações

.apply()

```
[11] 1 df = pd.DataFrame([[1,2,3,4,5],[1,2,3,4,5]],columns=['a','b','c','d','e'])  
2 df
```

```
↳      a   b   c   d   e  
0   1   2   3   4   5  
1   1   2   3   4   5
```

```
[12] 1 df.apply(np.sqrt)
```

```
↳      a          b          c          d          e  
0   1.0  1.414214  1.732051  2.0  2.236068  
1   1.0  1.414214  1.732051  2.0  2.236068
```

# Transformações

## .apply()

```
[11] 1 df = pd.DataFrame([[1,2,3,4,5],[1,2,3,4,5]],columns=['a','b','c','d','e'])  
2 df
```

	a	b	c	d	e
0	1	2	3	4	5
1	1	2	3	4	5

```
[13] 1 def cubo(value):  
2     return value**3  
3  
4 df.apply(cubo)
```

	a	b	c	d	e
0	1	8	27	64	125
1	1	8	27	64	125

# Operações com séries

concatenando strings

```
[17] 1 df = pd.DataFrame({'primeiro_nome':['nasser','maria','joão'],'sobrenome':['boan','silva','carvalho']})  
2 df  
3  
4 ➔    primeiro_nome  sobrenome  
5      0           nasser       boan  
6      1           maria        silva  
7      2           joão        carvalho
```

# Operações com séries

concatenando strings

```
[18] 1 df['nome_completo'] = df['primeiro_nome'] + ' ' + df['sobrenome']
      2 df
```

	primeiro_nome	sobrenome	nome_completo
0	nasser	boan	nasser boan
1	maria	silva	maria silva
2	joão	carvalho	joão carvalho

# Operações com séries

```
[19] 1 df = pd.DataFrame([[1,2,3,4,5],[1,2,3,4,5]],columns=['a','b','c','d','e'])  
2 df
```

```
□      a b c d e  
0    1 2 3 4 5  
1    1 2 3 4 5
```

```
[20] 1 df['b'] + df['d']
```

```
□ 0   6  
1   6  
dtype: int64
```

```
[21] 1 df['b'] * df['d']
```

```
□ 0   8  
1   8  
dtype: int64
```

```
[22] 1 df['b'] ** df['d']
```

```
□ 0   16  
1   16  
dtype: int64
```

```
[23] 1 df['b'] / df['d']
```

```
□ 0   0.5  
1   0.5  
dtype: float64
```

# Reshaping

df.transpose()

## pandas.DataFrame.transpose

`DataFrame.transpose(self, *args, copy: bool = False) → 'DataFrame'`

[source]

Transpose index and columns.

Reflect the DataFrame over its main diagonal by writing rows as columns and vice-versa. The property `T` is an accessor to the method `transpose()`.

Parameters: `*args : tuple, optional`

Accepted for compatibility with NumPy.

`copy : bool, default False`

Whether to copy the data after transposing, even for DataFrames with a single dtype.

Note that a copy is always required for mixed dtype DataFrames, or for DataFrames with any extension types.

Returns: `DataFrame`

The transposed DataFrame.

# Reshaping

df.transpose()

```
[24] 1 df = pd.DataFrame([[1,2,3,4,5],[1,2,3,4,5]],columns=['a','b','c','d','e'])
2 df
[25] 1 df.transpose()
```

	a	b	c	d	e
0	1	2	3	4	5
1	1	2	3	4	5

	0	1
a	1	1
b	2	2
c	3	3
d	4	4
e	5	5

# Reshaping

df.pivot\_table()

## pandas.DataFrame.pivot\_table

`DataFrame.pivot_table(self, values=None, index=None, columns=None, aggfunc='mean', fill_value=None, margins=False, dropna=True, margins_name='All', observed=False) → 'DataFrame'` [source]

Create a spreadsheet-style pivot table as a DataFrame.

The levels in the pivot table will be stored in MultiIndex objects (hierarchical indexes) on the index and columns of the result DataFrame.

# Reshaping

## pd.pivot\_table()

```
[36] 1 df = pd.DataFrame({"A": ["foo", "foo", "foo", "foo", "foo",
2                               "bar", "bar", "bar", "bar"],
3                               "B": ["one", "one", "one", "two", "two",
4                               "one", "one", "two", "two"],
5                               "C": ["small", "large", "large", "small",
6                               "small", "large", "small", "small",
7                               "large"],
8                               "D": [1, 2, 2, 3, 3, 4, 5, 6, 7],
9                               "E": [2, 4, 5, 5, 6, 6, 8, 9, 9]})
```

```
[37] 1 df
```

	A	B	C	D	E
0	foo	one	small	1	2
1	foo	one	large	2	4
2	foo	one	large	2	5
3	foo	two	small	3	5
4	foo	two	small	3	6
5	bar	one	large	4	6
6	bar	one	small	5	8
7	bar	two	small	6	9
8	bar	two	large	7	9

# Reshaping

pd.pivot\_table()

```
[9] 1 pd.pivot_table(df, values='D', index='A',columns=['C'],aggfunc='mean')  
C   C  large    small  
A  
bar      5.5  5.500000  
foo      2.0  2.333333
```

# Sorting

.nlargest()

pandas.Series.nlargest

`series.nlargest(self, n=5, keep='first')`

[source]

Return the largest  $n$  elements.

# Sorting

.nlargest()

```
[12] 1 df.mean_area.nlargest(10)

    ↗ 461    2501.0
      212    2499.0
      180    2250.0
      352    2010.0
      82     1878.0
      521    1841.0
      122    1761.0
      339    1747.0
      164    1686.0
      202    1685.0
Name: mean_area, dtype: float64
```

# Sorting

## .nlargest()

	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	diagnosis
461	27.42	26.27	186.9	2501.0	0.10840	0
212	28.11	18.47	188.5	2499.0	0.11420	0
180	27.22	21.87	182.1	2250.0	0.10940	0
352	25.73	17.46	174.2	2010.0	0.11490	0
82	25.22	24.91	171.5	1878.0	0.10630	0
521	24.63	21.60	165.5	1841.0	0.10300	0
122	24.25	20.20	166.2	1761.0	0.14470	0
339	23.51	24.27	155.1	1747.0	0.10690	0
164	23.27	22.04	152.1	1686.0	0.08439	0
202	23.29	26.67	158.9	1685.0	0.11410	0

# Sorting

df.nsmallest()

## pandas.DataFrame.nsmallest

`DataFrame.nsmallest(self, n, columns, keep='first') → 'DataFrame'`

[\[source\]](#)

Return the first  $n$  rows ordered by  $columns$  in ascending order.

Return the first  $n$  rows with the smallest values in  $columns$ , in ascending order. The columns that are not specified are returned as well, but not used for ordering.

This method is equivalent to `df.sort_values(columns, ascending=True).head(n)`, but more performant.

# Sorting

## .nsmallest()

```
[13] 1 df.nsmallest(n=10,columns='mean_area')
```

	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	diagnosis
101	6.981	13.43	43.79	143.5	0.11700	1
539	7.691	25.44	48.34	170.4	0.08668	1
538	7.729	25.49	47.98	178.8	0.08098	1
568	7.760	24.54	47.92	181.0	0.05263	1
46	8.196	16.84	51.71	201.9	0.08600	1
151	8.219	20.70	53.27	203.9	0.09405	1
314	8.597	18.60	54.09	221.2	0.10740	1
525	8.571	13.10	54.53	221.3	0.10360	1
61	8.598	20.98	54.66	221.8	0.12430	1
59	8.618	11.79	54.34	224.5	0.09752	1

# Sorting

.nsmallest()

```
[14] 1 df.mean_area.nsmallest(10)

    ↗ 101    143.5
      539    170.4
      538    178.8
      568    181.0
      46     201.9
      151    203.9
      314    221.2
      525    221.3
      61     221.8
      59     224.5

Name: mean_area, dtype: float64
```

# Sorting

df.sort\_values()

## pandas.DataFrame.sort\_values

`DataFrame.sort_values(self, by, axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last', ignore_index=False)`

[source]

Sort by the values along either axis.

# Sorting

df.sort\_values()

```
[7] 1 df.head()

[8] 1 df.sort_values('mean_texture', ascending=True)
```

	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	diagnosis
0	17.99	10.38	122.80	1001.0	0.11840	0
1	20.57	17.77	132.90	1326.0	0.08474	0
2	19.69	21.25	130.00	1203.0	0.10960	0
3	11.42	20.38	77.58	386.1	0.14250	0
4	20.29	14.34	135.10	1297.0	0.10030	0

	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	diagnosis
166	10.80	9.71	68.77	357.6	0.09594	1
0	17.99	10.38	122.80	1001.0	0.11840	0
313	11.54	10.72	73.73	409.1	0.08597	1
120	11.41	10.82	73.34	403.3	0.09373	1
123	14.50	10.89	94.28	640.7	0.11010	1
...	...	...	...	...	...	...

# Sorting

.sort\_values()

```
[10] 1 df.mean_texture.sort_values(ascending=True)

    ↗ 166      9.71
        0      10.38
        313     10.72
        120     10.82
        123     10.89
        ...
        265     31.12
        219     32.47
        259     33.56
        232     33.81
        239     39.28
Name: mean_texture, Length: 569, dtype: float64
```

# Concat

pd.concat()

## pandas.concat

```
pandas.concat(objs: Union[Iterable[Union[ForwardRef('DataFrame'), ForwardRef('Series')]]], Mapping[Union[Hashable, NoneType], Union[ForwardRef('DataFrame'), ForwardRef('Series')]]], axis=0, join='outer', ignore_index: bool = False, keys=None, levels=None, names=None, verify_integrity: bool = False, sort: bool = False, copy: bool = True) → Union[ForwardRef('DataFrame'), ForwardRef('Series')]
```

[source]

Concatenate pandas objects along a particular axis with optional set logic along the other axes.

Can also add a layer of hierarchical indexing on the concatenation axis, which may be useful if the labels are the same (or overlapping) on the passed axis number.

# Concat

pd.concat()

```
[13] 1  serie1 = pd.Series(['a','b','c','d','e'])
      2  serie2 = pd.Series(['f','g','h','i','j'])
      3
      4  pd.concat([serie1,serie2])
```

0	a
1	b
2	c
3	d
4	e
0	f
1	g
2	h
3	i
4	j

dtype: object

# Concat

pd.concat()

```
[14] 1  serie1 = pd.Series(['a','b','c','d','e'])
      2  serie2 = pd.Series(['f','g','h','i','j'])
      3
      4  pd.concat([serie1,serie2],ignore_index=True)
```

0 a  
1 b  
2 c  
3 d  
4 e  
5 f  
6 g  
7 h  
8 i  
9 j  
dtype: object

# Merge

df.merge()

## pandas.DataFrame.merge

```
DataFrame.merge(self, 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) → 'DataFrame'
```

[source]

Merge DataFrame or named Series objects with a database-style join.

The join is done on columns or indexes. If joining columns on columns, the DataFrame indexes *will be ignored*. Otherwise if joining indexes on indexes or indexes on a column or columns, the index will be passed on.

# Merge

df.merge()

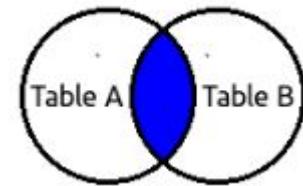
```
[16] 1 df1
```

	lkey	value
0	foo	1
1	bar	2
2	baz	3
3	foo	5

```
[17] 1 df2
```

	rkey	value
0	foo	5
1	bar	6
2	baz	7
3	foo	8



```
[28] 1 ## default é inner join
```

```
2
```

```
3 df1.merge(df2,on='value')
```

	lkey	value	rkey
0	foo	5	foo

# Merge

df.merge()

```
[16] 1 df1
```

	lkey	value
0	foo	1
1	bar	2
2	baz	3
3	foo	5

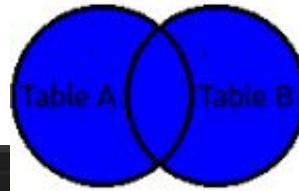
  

```
[17] 1 df2
```

	rkey	value
0	foo	5
1	bar	6
2	baz	7
3	foo	8

```
1 ## outer join
2
3 df1.merge(df2,on='value',how='outer')
```

	lkey	value	rkey
0	foo	1	NaN
1	bar	2	NaN
2	baz	3	NaN
3	foo	5	foo
4	NaN	6	bar
5	NaN	7	baz
6	NaN	8	foo



# Merge

df.merge()

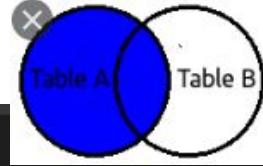
```
[16] 1 df1
```

	lkey	value
0	foo	1
1	bar	2
2	baz	3
3	foo	5

```
[17] 1 df2
```

	rkey	value
0	foo	5
1	bar	6
2	baz	7
3	foo	8



```
[31] 1 ## left
```

```
[32] 2
```

```
[33] 3 df1.merge(df2,on='value',how='left')
```

	lkey	value	rkey
0	foo	1	NaN
1	bar	2	NaN
2	baz	3	NaN
3	foo	5	foo

# Merge

df.merge()

```
[16] 1 df1
```

	lkey	value
0	foo	1
1	bar	2
2	baz	3
3	foo	5

```
[17] 1 df2
```

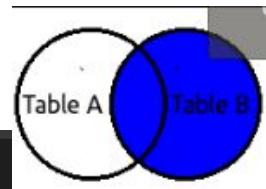
	rkey	value
0	foo	5
1	bar	6
2	baz	7
3	foo	8

```
[32] 1 ## right
```

```
2
```

```
3 df1.merge(df2,on='value',how='right')
```

	lkey	value	rkey
0	foo	5	foo
1	NaN	6	bar
2	NaN	7	baz
3	NaN	8	foo



# Valores Nulos

df.isna()

## pandas.DataFrame.isna

`DataFrame.isna(self) → 'DataFrame'`

[source]

Detect missing values.

Return a boolean same-sized object indicating if the values are NA. NA values, such as None or `numpy.NaN`, gets mapped to True values. Everything else gets mapped to False values. Characters such as empty strings '' or `numpy.inf` are not considered NA values (unless you set `pandas.options.mode.use_inf_as_na = True`).

Returns: DataFrame

Mask of bool values for each element in DataFrame that indicates whether an element is not an NA value.

# Valores Nulos

df.isna()

```
[58] 1 url = 'http://dados.df.gov.br/dataset/3a3b7b40-c715-439d-9dff-f22b47fc5994/resource/f4422c06-a041-46cc-9d89-9d09ffae1c70/download/2018-12-19-infracoes.csv'
2
3 df = pd.read_csv(url,sep=';',nrows=100)

[59] 1 print(df.shape)
2 df.head()

cometimento hora_cometimento auinf_local_rodovia auinf_local_km auinf_local_referencia auinf_local_complemento auinf_local_latitude auinf_local_longitude grav_tipo
01/11/2018 00:01 DF-011 (EPIG) KM 0,8
SENT EIXO
MONUM/EPTG (PRÓ...
NaN NaN NaN NaN NaN NaN NaN NaN Média

01/11/2018 00:05 DF-051 (EPGU) KM 03,4
SENTIDO GUARÁ II AO
ZOOL...
NaN NaN NaN NaN NaN NaN NaN NaN Média

01/11/2018 00:05 DF-079 (EPVP) KM 3,6
SENTIDO DF-075
(EPNB) ...
NaN NaN NaN NaN NaN NaN NaN NaN Média
```

# Valores Nulos

df.isna()

```
[47] 1 df.isna()
```

	tipo_infracao	descricao	tipo_infrator	tipo_veiculo	cometimento	hora_cometimento	auinf_local_rodovia	auinf_local_km	auinf_local_referencia	auinf_local_comple
0	False	False	False	False	False	False	False	True	True	
1	False	False	False	False	False	False	False	True	True	
2	False	False	False	False	False	False	False	True	True	
3	False	False	False	False	False	False	False	True	True	
4	False	False	False	False	False	False	False	True	True	
...	...	...	...	...	...	...	...	...	...	...

# Valores Nulos

df.isna()

```
[60] 1 df.isna().sum()

  tipo_infracao          0
  descricao               0
  tipo_infrator           0
  tipo_veiculo            0
  cometimento             0
  hora_cometimento        0
  auinf_local_rodovia     0
  auinf_local_km           0
  auinf_local_referencia  100
  auinf_local_complemento 100
  auinf_local_latitude     100
  auinf_local_longitude    100
  grav_tipo                 0
dtype: int64
```

# Valores Nulos

df.isnull()

## pandas.DataFrame.isnull

`DataFrame.isnull(self) → 'DataFrame'`

[\[source\]](#)

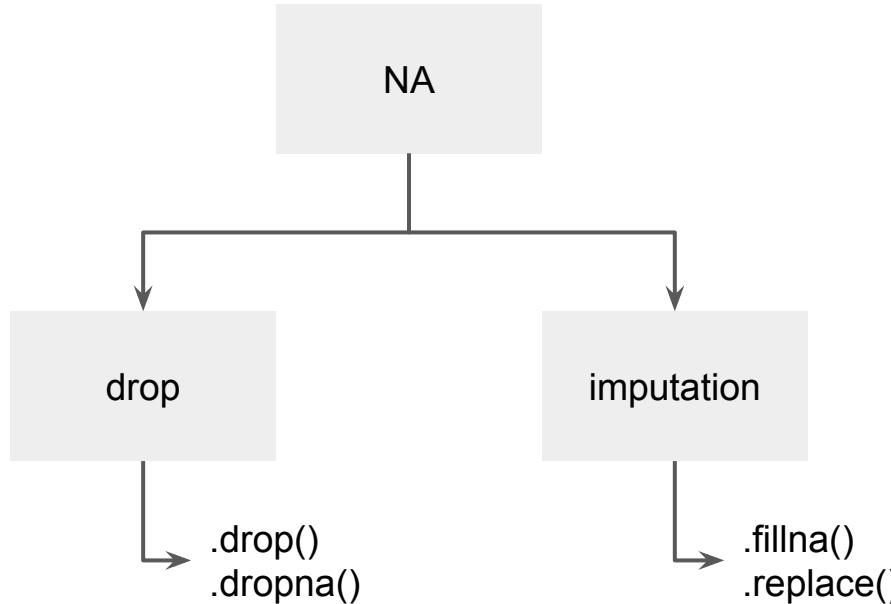
Detect missing values.

Return a boolean same-sized object indicating if the values are NA. NA values, such as None or `numpy.Nan`, gets mapped to True values. Everything else gets mapped to False values. Characters such as empty strings '' or `numpy.inf` are not considered NA values (unless you set `pandas.options.mode.use_inf_as_na = True`).

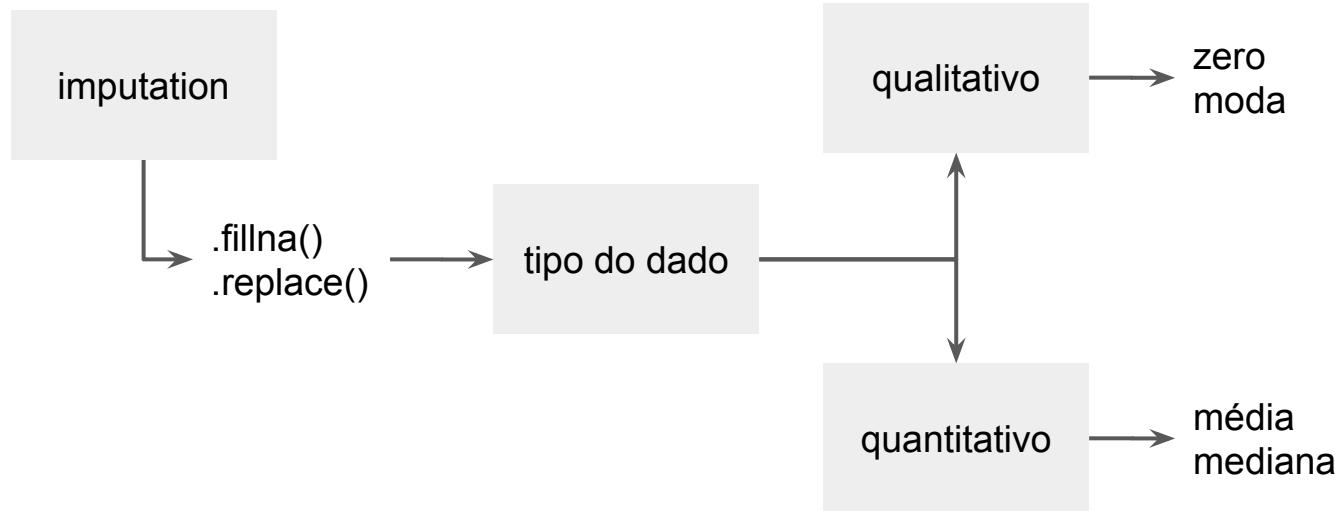
**Returns:** DataFrame

Mask of bool values for each element in DataFrame that indicates whether an element is not an NA value.

# Valores Nulos



# Valores Nulos



# Valores Nulos

.fillna()

pandas.Series.fillna

`Series.fillna(self, value=None, method=None, axis=None, inplace=False, limit=None, downcast=None) → Union[ForwardRef('Series'), NoneType]` [source]

Fill NA/NaN values using the specified method.

# Valores Nulos

.fillna()

```
1 df[['rooms','bathroom']].isna().sum()  
2 rooms      11  
bathroom    11  
dtype: int64
```

# Valores Nulos

df.fillna()

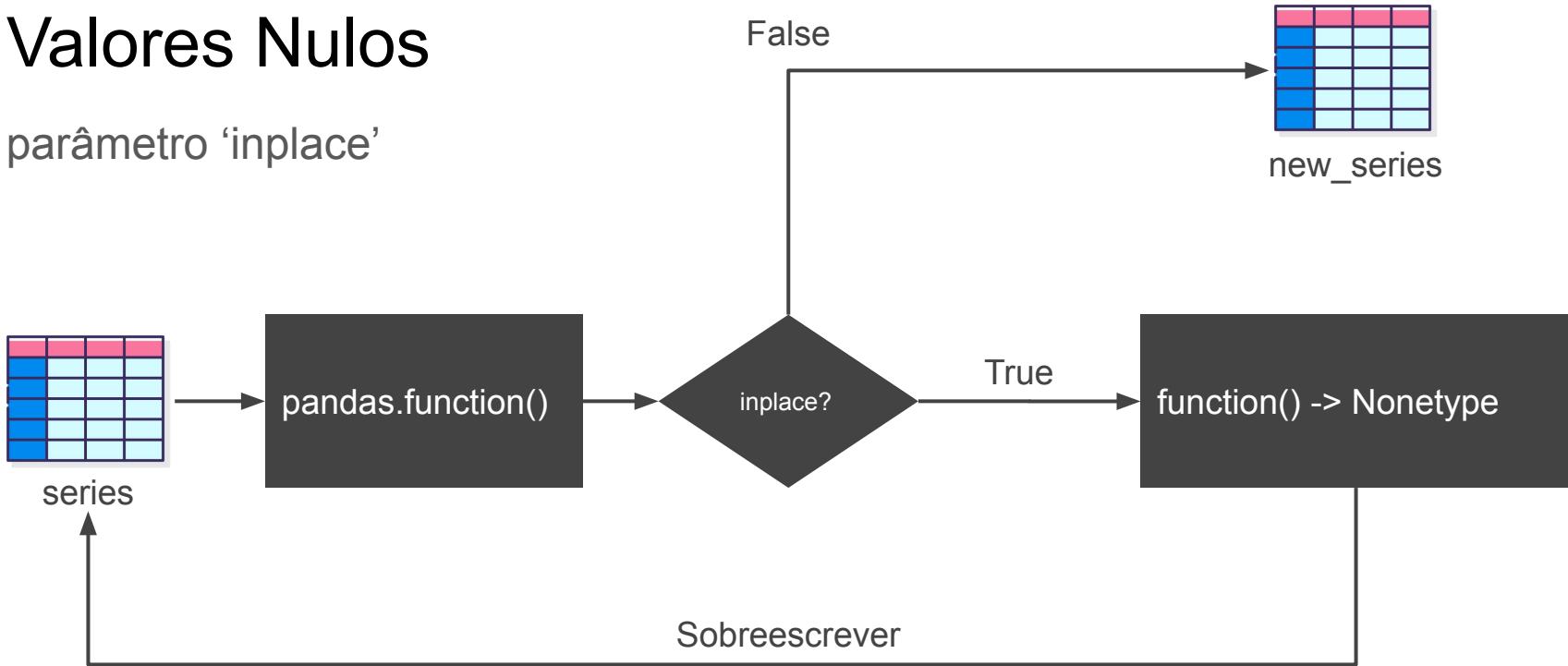
```
[11] 1 ## achando a media de rooms
     2 med_room = df.rooms.mean()
     3
     4 ## achando a media de bathrooms
     5 med_bath = df.bathroom.mean()
     6
     7 ## inputando em cada coluna com a média
     8 df['rooms'].fillna(med_room inplace=True)
     9 df['bathroom'].fillna(med_bath,inplace=True)

[12] 1 ## verificando se as colunas ainda possuem valores nulos
     2 df[['rooms','bathroom']].isna().sum()

   rooms      0
bathroom      0
dtype: int64
```

# Valores Nulos

parâmetro ‘inplace’



# Valores Nulos

df.replace()

## pandas.Series.replace

`Series.replace(self, to_replace=None, value=None, inplace=False, limit=None, regex=False, method='pad')`

[\[source\]](#)

Replace values given in `to_replace` with `value`.

Values of the Series are `replaced` with other values dynamically. This differs from updating with `.loc` or `.iloc`, which require you to specify a location to update with some value.

# Valores Nulos

.replace()

```
1 ## Substituindo os valores utilizando um dicionário
2 df['furniture'].replace(
3     {'furnished':'mobiliado','not furnished':'n mobiliado'},
4     inplace=True)
```

[24] 1 df.head()

bathroom	parking spaces	floor	animal	furniture	hoa	rent amount	property tax
3.000000	4.0	-	acept	mobiliado	R\$0	R\$8,000	R\$1,000
1.000000	1.0	10	acept	n mobiliado	R\$540	R\$820	R\$122
5.000000	4.0	3	acept	mobiliado	R\$4,172	R\$7,000	R\$1,417
2.000000	1.0	12	acept	n mobiliado	R\$700	R\$1,250	R\$150
2.341901	NaN	NaN	NaN	n mobiliado	R\$0	R\$1,200	R\$41

# Valores Nulos

drop

.drop()  
.dropna()

## pandas.DataFrame.drop

DataFrame.drop(self, labels=None, axis=0, index=None, columns=None, level=None, inplace=False, errors='raise')

[source]

Drop specified labels from rows or columns.

Remove rows or columns by specifying label names and corresponding axis, or by specifying directly index or column names. When using a multi-index, labels on different levels can be removed by specifying the level.

## pandas.DataFrame.dropna

DataFrame.dropna(self, axis=0, how='any', thresh=None, subset=None, inplace=False)

[source]

Remove missing values.

See the [User Guide](#) for more on which values are considered missing, and how to work with missing data.

# VAMOS PRATICAR!!



# VAMOS PRATICAR!!

1. Entre em dados.gov.br;
2. Escolha um dataset público de sua preferência (em formato .csv);
3. Baixe o dataset;
4. Leia o csv utilizando pandas;
5. Aplique as seguintes funções:
  - a. apply()
  - b. nlargest()
  - c. nsmallest()
  - d. sort\_values()
6. Descreva quantos valores nulos cada coluna tem;
7. Defina a melhor estratégia para tratamento dos valores nulos;
8. Trate os valores nulos.



# Bônus ...

## pandas.DataFrame.groupby

```
DataFrame.groupby(self, by=None, axis=0, level=None, as_index: bool = True, sort: bool = True,  
group_keys: bool = True, squeeze: bool = False, observed: bool = False) →  
'groupby_generic.DataFrameGroupBy'
```

[source]

Group DataFrame using a mapper or by a Series of columns.

A **groupby** operation involves some combination of splitting the object, applying a function, and combining the results. This can be used to group large amounts of data and compute operations on these groups.

# Bônus ...

```
df.groupby(by="key").sum()
```

# Bônus ...

```
[68] 1 df[['GarageQual','SalePrice']].groupby('GarageQual').mean()
```

```
C↳          SalePrice
```

```
GarageQual
```

```
Ex      241000.000000
```

```
Fa      123573.354167
```

```
Gd      215860.714286
```

```
Po      100166.666667
```

```
TA      187489.836003
```

GarageQual: Garage quality

Ex	Excellent
Gd	Good
TA	Typical/Average
Fa	Fair
Po	Poor
NA	No Garage

# Bônus ...

```
[4] 1 df[['YrSold','SalePrice','LotArea']].groupby('YrSold').agg(  
2 | | {'SalePrice':'sum','LotArea':'mean'})
```

YrSold	SalePrice	LotArea
2006	57320530	10489.458599
2007	61214777	10863.686930
2008	53917695	10587.687500
2009	60648051	10294.248521
2010	31043893	10220.645714

# Bônus ...

	ticketid	dia_da_semana	mes
0	343	quarta	janeiro
1	290	quinta	janeiro
2	192	quinta	janeiro
3	1	sexta	janeiro
4	4	sábado	janeiro
...	...	...	...
171	562	quarta	junho
172	473	quinta	junho
173	481	quinta	junho
174	20	sexta	junho
175	2	sábado	junho

# Bônus ...

```
.groupby(['mes','dia_da_semana'],as_index=False).sum()  
)  
  
] :  
    mes dia_da_semana ticketid  
0   abril       quarta     2375  
1   abril       quinta    3653  
2   abril      segunda    3372  
3   abril      sexta      38  
4   abril      sábado     15  
5   abril      terça     3359  
6  fevereiro    quarta    2112  
7  fevereiro    quinta    3929
```

# Bônus ...

## pandas.DataFrame.drop\_duplicates

```
DataFrame.drop_duplicates(self, subset: Union[Hashable, Sequence[Hashable], NoneType] =  
    None, keep: Union[str, bool] = 'first', inplace: bool = False, ignore_index: bool = False) →  
    Union[ForwardRef('DataFrame'), NoneType]
```

[source]

Return DataFrame with `duplicate` rows removed.

Considering certain columns is optional. Indexes, including time indexes are ignored.

# Bônus ...

```
[26] 1 df.shape
[27] (1460, 81)

[25] 1 df.drop_duplicates()

   Id MSSubClass MSZoning LotFrontage L...
0    1          60       RL      65.0
1    2          20       RL      80.0
2    3          60       RL      68.0
3    4          70       RL      60.0
4    5          60       RL      84.0
...
1455 1456        60       RL      62.0
1456 1457        20       RL      85.0
1457 1458        70       RL      66.0
1458 1459        20       RL      68.0
1459 1460        20       RL      75.0
1460 rows × 81 columns
```

# Bônus ...

```
[36] 1 df.drop_duplicates(subset=['SalePrice'],keep='last')
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street
0	1	60	RL	65.0	8450	Pave
1	2	20	RL	80.0	9600	Pave
6	7	20	RL	75.0	10084	Pave

```
[37] 1 df.drop_duplicates(subset=['SalePrice'],keep='last').SalePrice.nunique()
```

```
663
```

1456	1457	20	RL	85.0	13175	Pave
1457	1458	70	RL	66.0	9042	Pave
1458	1459	20	RL	68.0	9717	Pave
1459	1460	20	RL	75.0	9937	Pave

663 rows × 81 columns

# Bônus ...

## pandas.DataFrame.rename

```
DataFrame.rename(self, mapper=None, index=None, columns=None, axis=None, copy=True,  
inplace=False, level=None, errors='ignore')
```

[\[source\]](#)

Alter axes labels.

Function / dict values must be unique (1-to-1). Labels not contained in a dict / Series will be left as-is. Extra labels listed don't throw an error.

See the [user guide](#) for more.

# Bônus ...

```
[44] 1 novo.rename(columns={'SalePrice':'valor_total_vendido','LotArea':'média_de_square_feet'})
```

```
↳      valor_total_vendido  média_de_square_feet
```

YrSold

2006	57320530	10489.458599
2007	61214777	10863.686930
2008	53917695	10587.687500
2009	60648051	10294.248521
2010	31043893	10220.645714

# Bônus ...

## pandas.DataFrame.assign

`DataFrame.assign(self, **kwargs) → 'DataFrame'`

[\[source\]](#)

Assign new columns to a DataFrame.

Returns a new object with all original columns in addition to new ones. Existing columns that are re-assigned will be overwritten.

Parameters: `**kwargs : dict of {str: callable or Series}`

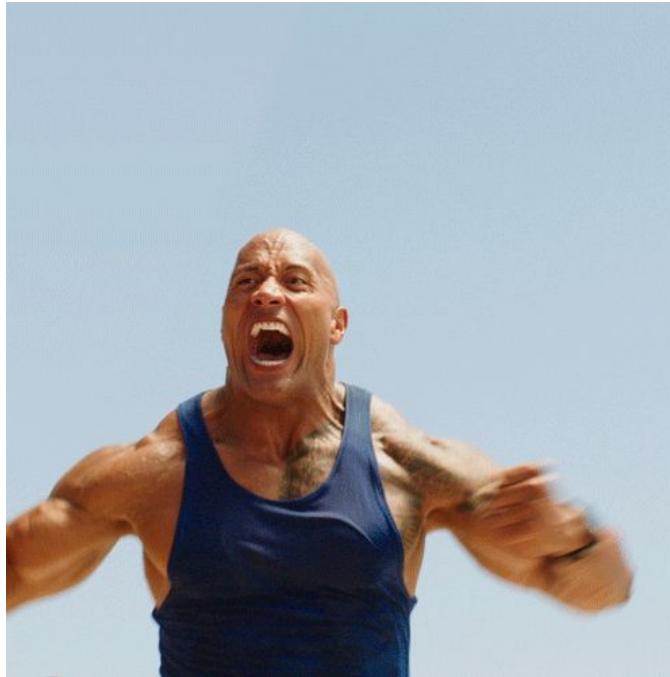
The column names are keywords. If the values are callable, they are computed on the DataFrame and assigned to the new columns. The callable must not change input DataFrame (though pandas doesn't check it). If the values are not callable, (e.g. a Series, scalar, or array), they are simply assigned.

# Bônus ...

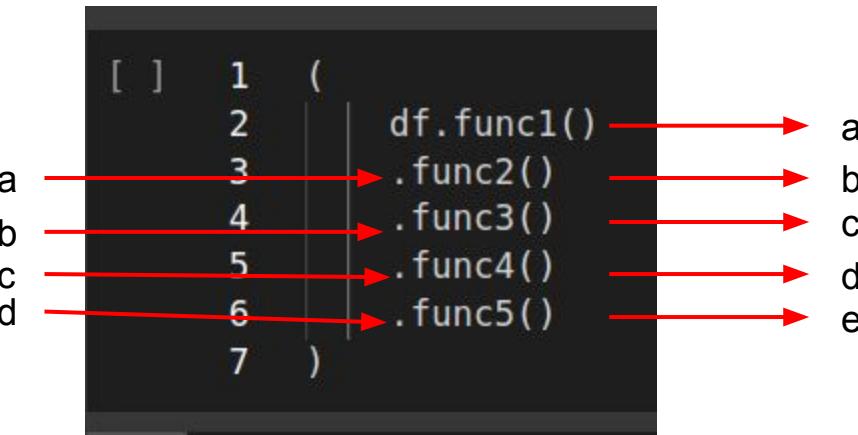
```
[54] 1 novo.assign(valor_total_em_reais = lambda x: (x.valor_total_vendido * 5.1),  
2 | | | | | média_em_m2           = lambda x: (x.média_de_square_feet / 10.764))
```

	valor_total_vendido	média_de_square_feet	valor_total_em_reais	média_em_m2
YrSold				
2006	57320530	10489.458599	292334703.0	974.494481
2007	61214777	10863.686930	312195362.7	1009.261142
2008	53917695	10587.687500	274980244.5	983.620169
2009	60648051	10294.248521	309305060.1	956.359023
2010	31043893	10220.645714	158323854.3	949.521155

# METHOD CHAINING!!!



# METHOD CHAINING!!!



# METHOD CHAINING!!!

```
[57] 1  (
2  |  pd.read_csv('/content/train.csv')
3  )
```

	YrSold	SalePrice	LotArea
0	2008	208500	8450
1	2007	181500	9600
2	2008	223500	11250
3	2006	140000	9550
4	2008	250000	14260
...	...	...	...
1455	2007	175000	7917
1456	2010	210000	13175
1457	2010	266500	9042
1458	2010	142125	9717
1459	2008	147500	9937

1460 rows × 3 columns

# METHOD CHAINING!!!

```
[58] 1  (
2  |  pd.read_csv('/content/train.csv')
3  |  .loc[:,['YrSold','SalePrice','LotArea']]
4  )
```

	YrSold	SalePrice	LotArea
0	2008	208500	8450
1	2007	181500	9600
2	2008	223500	11250
3	2006	140000	9550
4	2008	250000	14260
...	...	...	...
1455	2007	175000	7917
1456	2010	210000	13175
1457	2010	266500	9042
1458	2010	142125	9717
1459	2008	147500	9937

1460 rows × 3 columns

# METHOD CHAINING!!!

```
[61] 1  (
2    pd.read_csv('/content/train.csv')
3    .loc[:,['YrSold','SalePrice','LotArea']]
4    .groupby('YrSold')
5  )
```

```
↳ <pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fbc70af49b0>
```

# METHOD CHAINING!!!

```
[62] 1  (
2  |  pd.read_csv('/content/train.csv')
3  |  .loc[:,['YrSold','SalePrice','LotArea']]
4  |  .groupby('YrSold')
5  |  .agg({'SalePrice':'sum','LotArea':'mean'})
6  )
```

	SalePrice	LotArea
YrSold		
2006	57320530	10489.458599
2007	61214777	10863.686930
2008	53917695	10587.687500
2009	60648051	10294.248521
2010	31043893	10220.645714

# METHOD CHAINING!!!

The screenshot shows a Jupyter Notebook cell with the following code:

```
1  (
2      pd.read_csv('/content/train.csv')
3      .loc[:,['YrSold','SalePrice','LotArea']]
4      .groupby('YrSold')
5      .agg({'SalePrice':'sum','LotArea':'mean'})
6      |.rename(columns={'SalePrice':'valor_total_vendido','LotArea':'média_de_square_feet'})
7  )
```

The output of the code is displayed below the cell, showing a DataFrame with three columns: 'YrSold', 'valor\_total\_vendido', and 'média\_de\_square\_feet'. The data is as follows:

YrSold	valor_total_vendido	média_de_square_feet
2006	57320530	10489.458599
2007	61214777	10863.686930
2008	53917695	10587.687500
2009	60648051	10294.248521
2010	31043893	10220.645714

# METHOD CHAINING!!!

```
[64] 1  (
2      pd.read_csv('/content/train.csv')
3      .loc[:,['YrSold','SalePrice','LotArea']]
4      .groupby('YrSold')
5      .agg({'SalePrice':'sum','LotArea':'mean'})
6      .rename(columns={'SalePrice':'valor_total_vendido','LotArea':'média_de_square_'
7      .assign(valor_total_em_reais = lambda x: (x.valor_total_vendido * 5.1),
8      ||||| média_em_m2           = lambda x: (x.média_de_square_feet / 10.764))
9  )
```

YrSold	valor_total_vendido	média_de_square_feet	valor_total_em_reais	média_em_m2
2006	57320530	10489.458599	292334703.0	974.494481
2007	61214777	10863.686930	312195362.7	1009.261142
2008	53917695	10587.687500	274980244.5	983.620169
2009	60648051	10294.248521	309305060.1	956.359023
2010	31043893	10220.645714	158323854.3	949.521155

# METHOD CHAINING!!!

```
[65] 1  (
2      pd.read_csv('/content/train.csv')
3      .loc[:,['YrSold','SalePrice','LotArea']]
4      .groupby('YrSold')
5      .agg({'SalePrice':'sum','LotArea':'mean'})
6      .rename(columns={'SalePrice':'valor_total_vendido','LotArea':'média_de_square_'
7      .assign(valor_total_em_reais = lambda x: (x.valor_total_vendido * 5.1),
8          média_em_m2           = lambda x: (x.média_de_square_feet / 10.764))
9      .drop(['valor_total_vendido','média_de_square_feet'],axis=1)
10     )
```

	valor_total_em_reais	média_em_m2
YrSold		
2006	292334703.0	974.494481
2007	312195362.7	1009.261142
2008	274980244.5	983.620169
2009	309305060.1	956.359023
2010	158323854.3	949.521155

# METHOD CHAINING!!!

Upload Refresh Mount Drive

..

sample\_data

relatório.csv

train.csv

```
[66] 1  (
2      pd.read_csv('/content/train.csv')
3      .loc[:,['YrSold','SalePrice','LotArea']]
4      .groupby('YrSold')
5      .agg({'SalePrice':'sum','LotArea':'mean'})
6      .rename(columns={'SalePrice':'valor_total_vendido','LotArea':'média_de_square_feet'})
7      .assign(valor_total_em_reais = lambda x: (x.valor_total_vendido * 5.1),
8             [mádia_em_m2 = lambda x: (x.média_de_square_feet / 10.764)])
9      .drop(['valor_total_vendido','média_de_square_feet'],axis=1)
10     .to_csv('relatório.csv',sep=';')
11 )
```

# METHOD CHAINING!!!

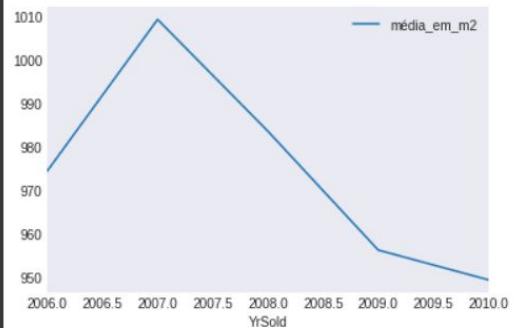
```
[71] 1 def criar_relatorio(path_arquivo):
2     (
3         pd.read_csv(path_arquivo)
4             .loc[:,['YrSold','SalePrice','LotArea']]
5             .groupby('YrSold')
6             .agg({'SalePrice':'sum','LotArea':'mean'})
7             .rename(columns={'SalePrice':'valor_total_vendi
8             .assign(valor_total_em_reais = lambda x: (x.val
9                 média_em_m2           = lambda x: (x.médi
10                .drop(['valor_total_vendido','média_de_square_f
11                .to_csv('relatório.csv',sep=';')
12
13
14
15
[72] 1 criar_relatorio('/content/train.csv')
```



# METHOD CHAINING!!!

```
[88] 1  (
2      pd.read_csv('/content/train.csv')
3      .loc[:,['YrSold','SalePrice','LotArea']]
4      .groupby('YrSold')
5      .agg({'SalePrice':'sum','LotArea':'mean'})
6      .rename(columns={'SalePrice':'valor_total_vendido','LotArea':'média_de_sq
7      .assign(valor_total_em_reais = lambda x: (x.valor_total_vendido * 5.1),
8          |     | média_em_m2 = lambda x: (xmédia_de_square_feet / 10.764)
9      .drop(['valor_total_vendido','média_de_square_feet','valor_total_em_reais'
10     .plot()
11   )
```

↳ <matplotlib.axes.\_subplots.AxesSubplot at 0x7fb6ea65c88>



# Plotando com pandas

## pandas.Series.plot

`Series.plot(self, *args, **kwargs)`

[source]

Make plots of Series or DataFrame.

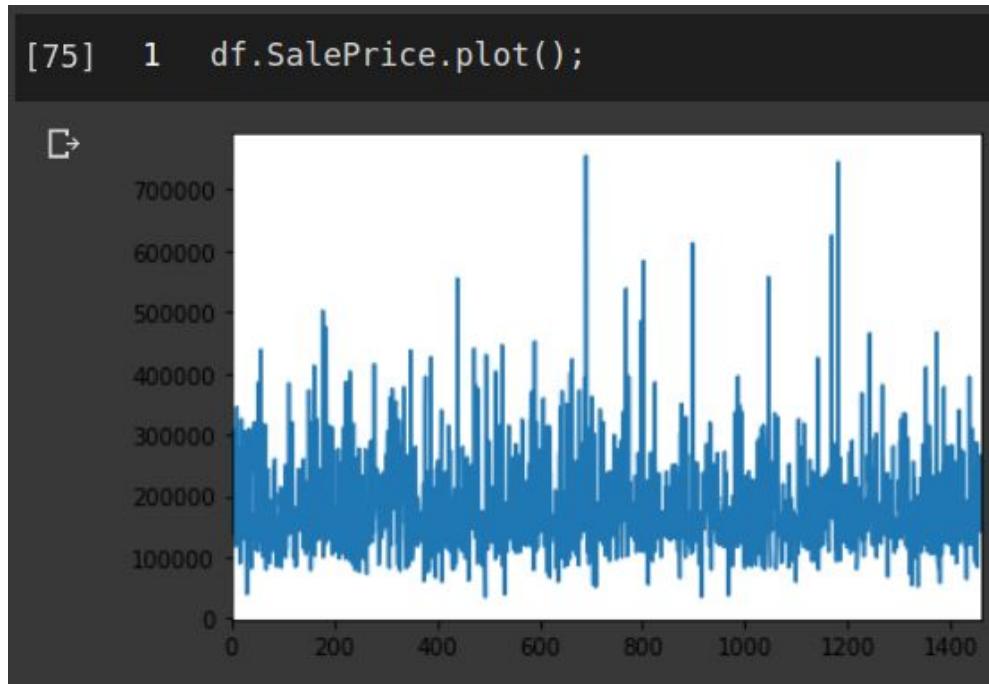
Uses the backend specified by the option `plotting.backend`. By default, `matplotlib` is used.

# Plotando com pandas

```
[73] 1 df.head()
```

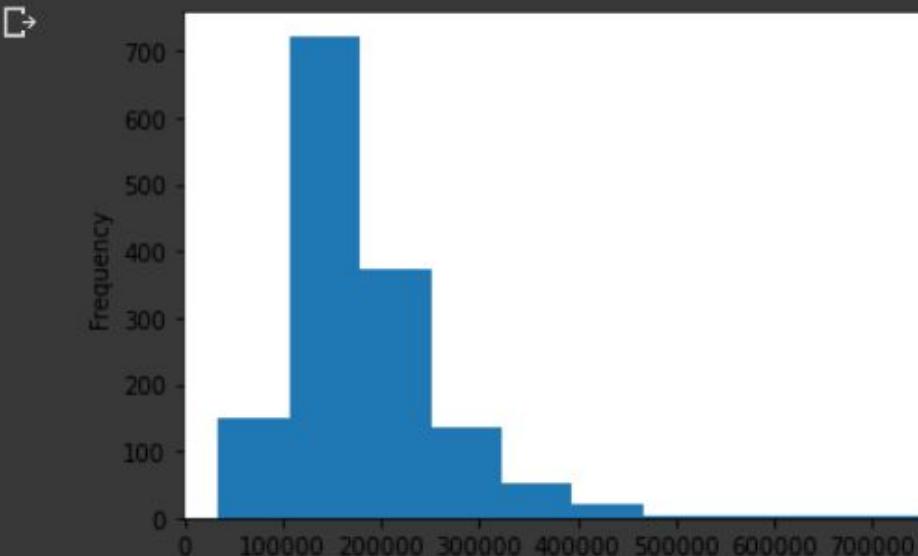
dPorch	3SsnPorch	ScreenPorch	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
0	0	0	0	NaN	NaN	NaN	0	2	2008	WD	Normal	208500
0	0	0	0	NaN	NaN	NaN	0	5	2007	WD	Normal	181500
0	0	0	0	NaN	NaN	NaN	0	9	2008	WD	Normal	223500
272	0	0	0	NaN	NaN	NaN	0	2	2006	WD	Abnorml	140000
0	0	0	0	NaN	NaN	NaN	0	12	2008	WD	Normal	250000

# Plotando com pandas



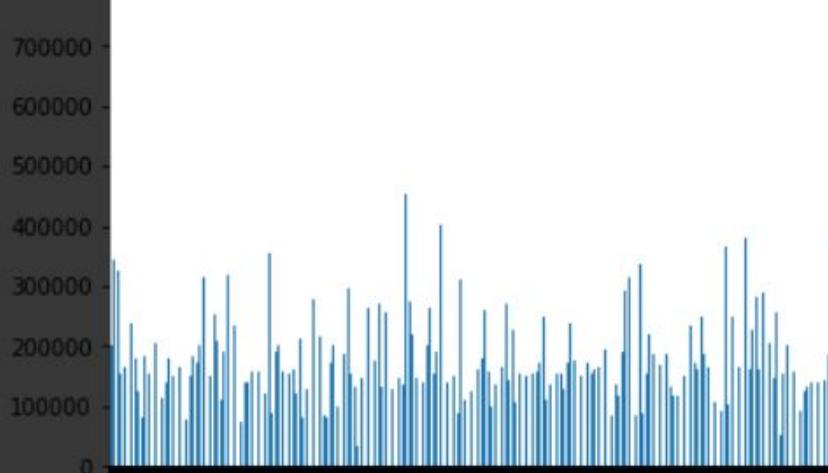
# Plotando com pandas

```
[77] 1 df.SalePrice.plot.hist();
```



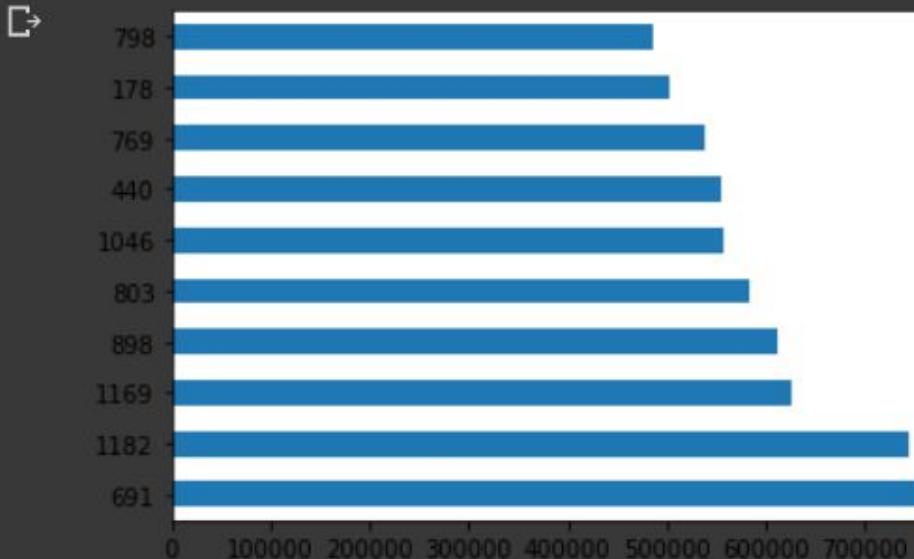
# Plotando com pandas

```
[80] 1 df.SalePrice.plot.bar();
```



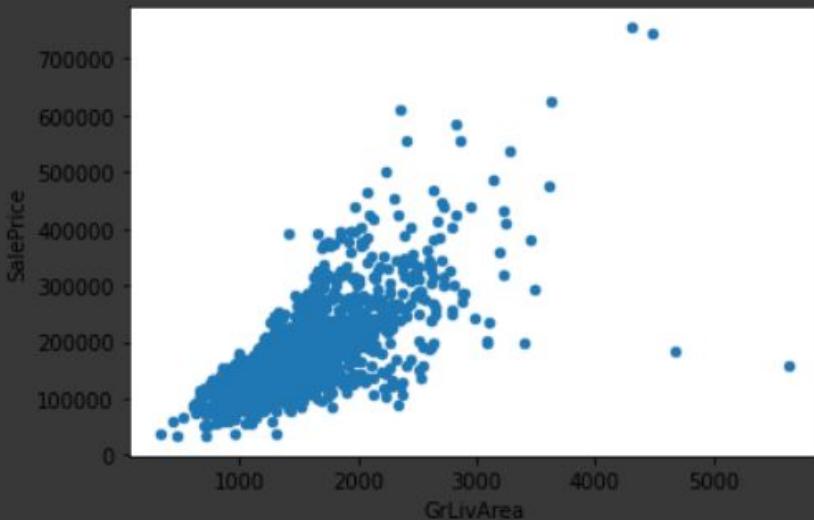
# Plotando com pandas

```
[83] 1 df.SalePrice.nlargest(10).plot.barh();
```



# Plotando com pandas

```
[34] 1 df[['GrLivArea','SalePrice']].plot.scatter(x='GrLivArea',y='SalePrice');
```



# pandas.DataFrame.plot.scatter

`DataFrame.plot.scatter(self, x, y, s=None, c=None, **kwargs)`

[source]

Create a scatter plot with varying marker point size and color.

The coordinates of each point are defined by two dataframe columns and filled circles are used to represent each point. This kind of plot is useful to see complex correlations between two variables. Points could be for instance natural 2D coordinates like longitude and latitude in a map or, in general, any pair of metrics that can be plotted against each other.

Parameters:

`x : int or str`  
The column name or column position to be used as horizontal coordinates for each point.

`y : int or str`

The column name or column position to be used as vertical coordinates for each point.

`s : scalar or array_like, optional`

The size of each point. Possible values are:

- A single scalar so all points have the same size.
- A sequence of scalars, which will be used for each point's size recursively. For instance, when passing [2,14] all points size will be either 2 or 14, alternatively.

`c : str, int or array_like, optional`

The color of each point. Possible values are:

- A single color string referred to by name, RGB or RGBA code, for instance 'red' or '#a98d19'.
- A sequence of color strings referred to by name, RGB or RGBA code, which will be used for each point's color recursively. For instance ['green','yellow'] all points will be filled in green or yellow, alternatively.
- A column name or position whose values will be used to color the marker points according to a colormap.

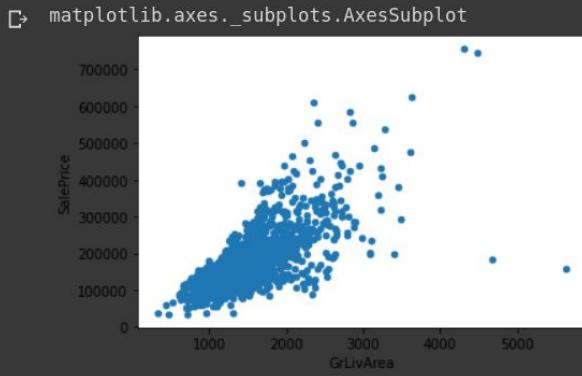
`**kwargs`

Keyword arguments to pass on to `DataFrame.plot()`.

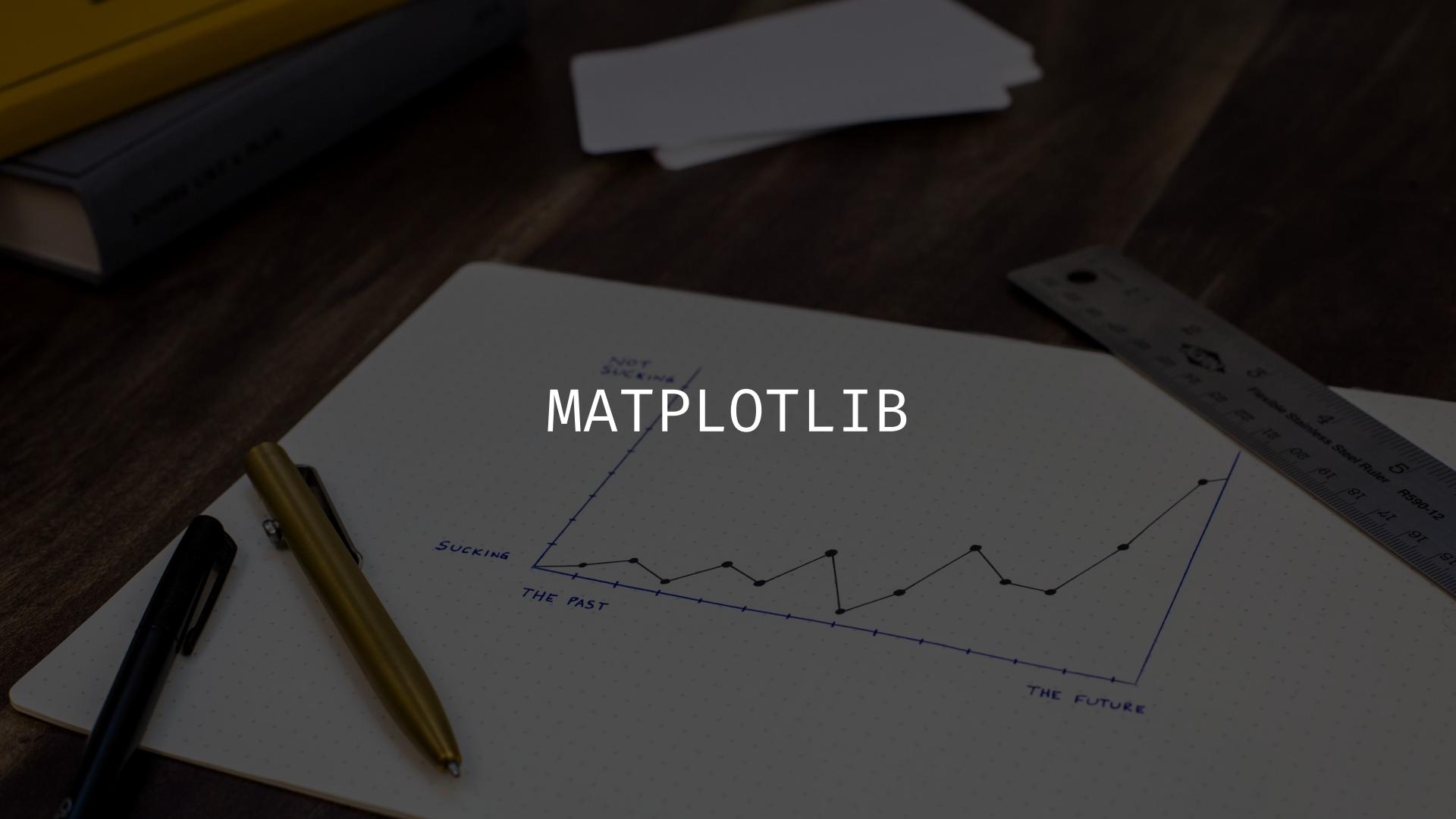
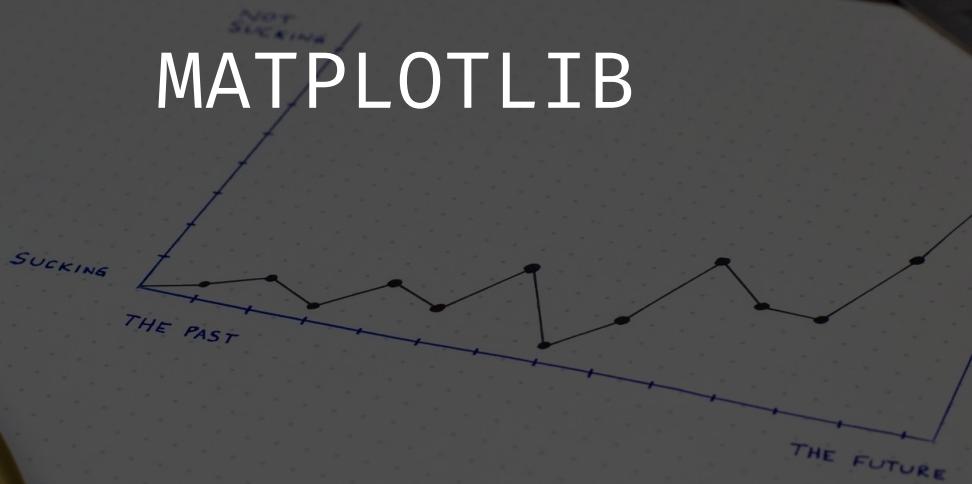
Returns: `matplotlib.axes.Axes or numpy.ndarray of them`

# Plotando com pandas

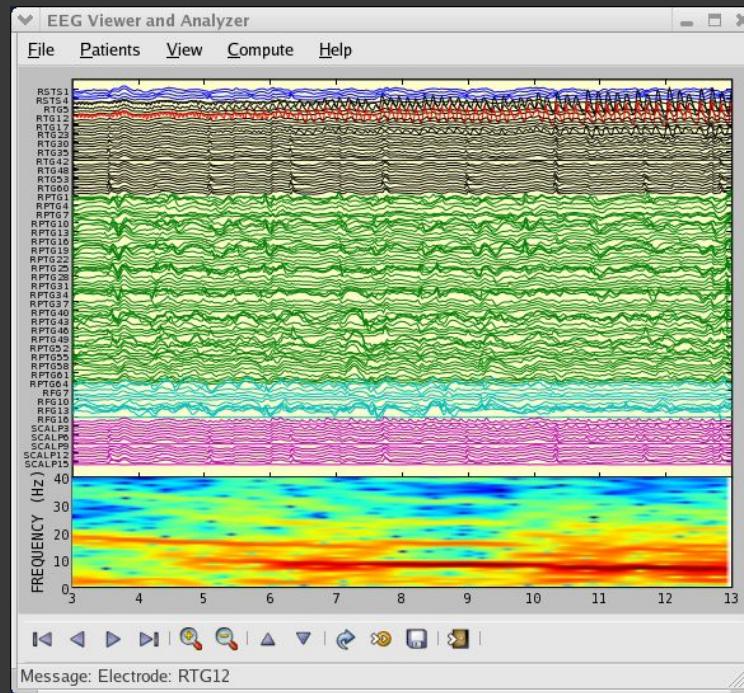
```
[40] 1 a = df[['GrLivArea','SalePrice']].plot.scatter(x='GrLivArea',y='SalePrice')
2 type(a)
```



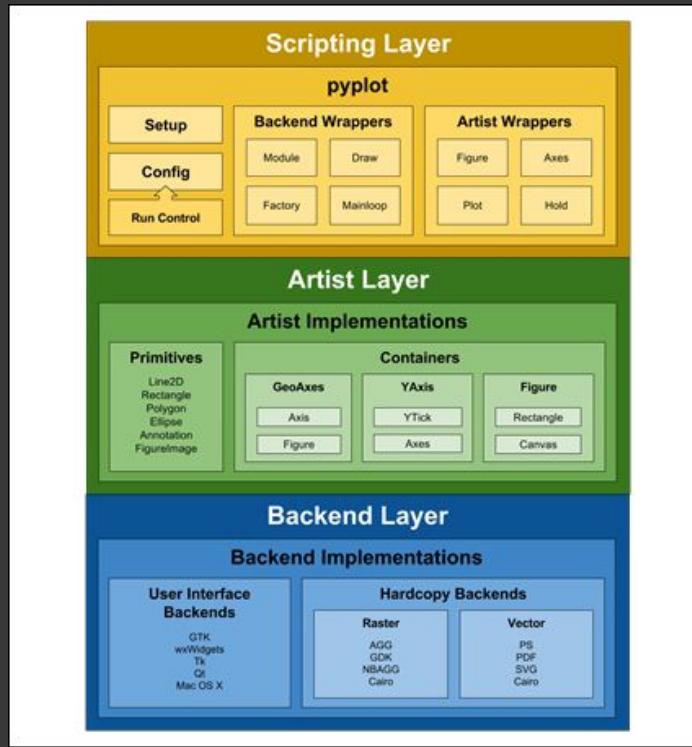
# MATPLOTLIB



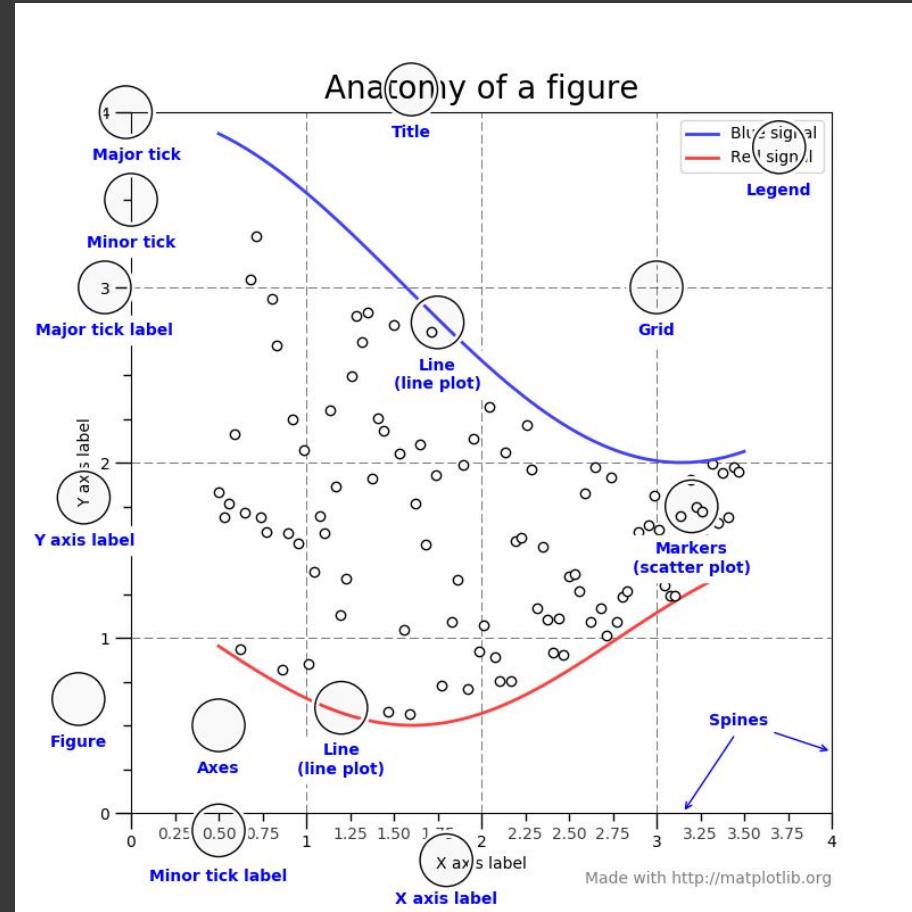
# MATPLOTLIB



# MATPLOTLIB



# MATPLOTLIB



# MATPLOTLIB

```
[21] 1 df = pd.read_csv('avocado.csv')
2 df.head()
```

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Lar Ba
0	0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.
1	1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.
2	2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.
3	3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.
4	4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.



# MATPLOTLIB

```
plt.figure()
```

## matplotlib.pyplot.figure

```
matplotlib.pyplot.figure(num=None, figsize=None, dpi=None, facecolor=None,  
edgecolor=None, frameon=True, FigureClass=<class 'matplotlib.figure.Figure'>, clear=False, **kwargs) [source]
```

# MATPLOTLIB

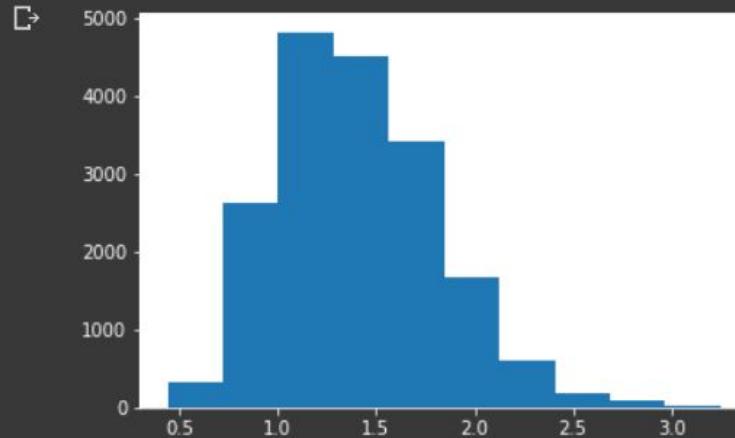
```
plt.figure()
```

```
[2] 1 plt.figure();  
[2]: <Figure size 432x288 with 0 Axes>
```

# MATPLOTLIB

```
plt.figure()
```

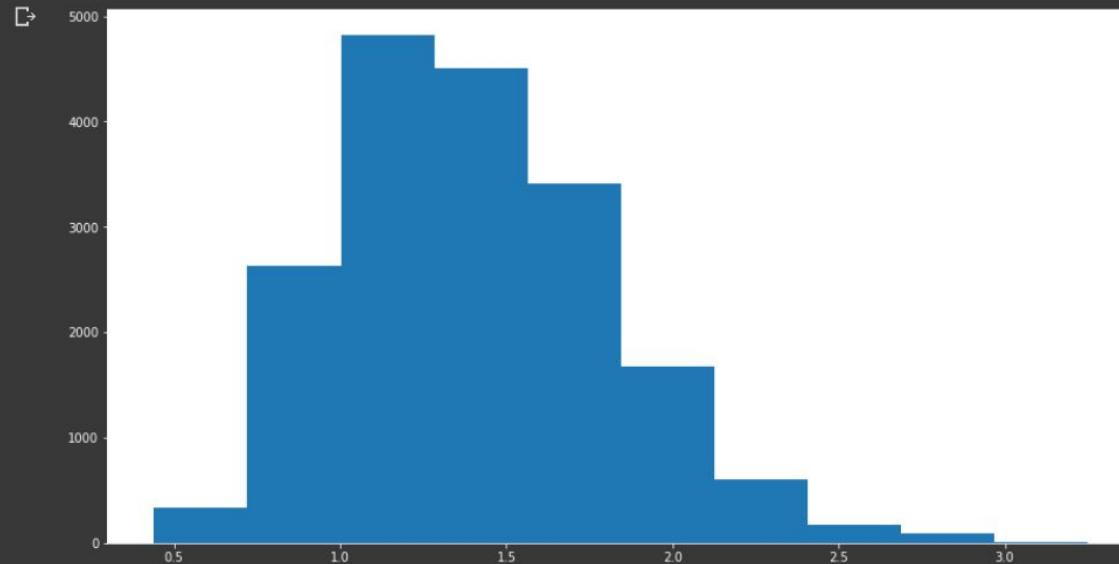
```
[24] 1 plt.figure()  
2 plt.hist(df.AveragePrice);
```



# MATPLOTLIB

```
plt.figure()
```

```
[19] 1 plt.figure(figsize=(15,8))  
2 plt.hist(df.AveragePrice);
```



# MATPLOTLIB

plt.bar()

## matplotlib.pyplot.bar

```
matplotlib.pyplot.bar(x, height, width=0.8, bottom=None, *, align='center', data=None, **kwargs)
```

[source]

# MATPLOTLIB

```
plt.bar()
```

```
[25] 1   data = (
2
3   |   df.loc[:,['year','AveragePrice']]
4   |   .groupby('year')
5   |   .mean()
6
7 )
```

AveragePrice	
year	
2015	1.375590
2016	1.338640
2017	1.515128
2018	1.347531

# MATPLOTLIB

```
plt.bar()
```

```
[56] 1  tamanho = data.values.flatten()  
2  tamanho
```

```
↳ array([1.37559038, 1.3386396 , 1.51512758, 1.34753086])
```

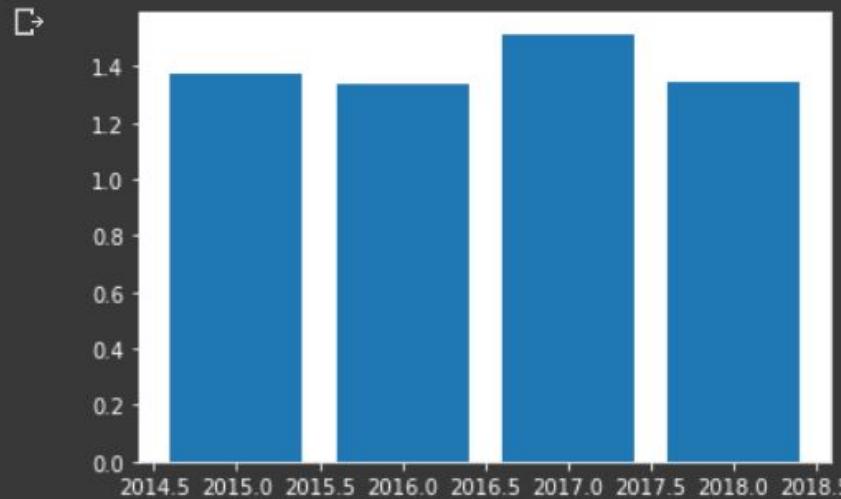
```
[59] 1  x = list(data.index)  
2  x
```

```
↳ [2015, 2016, 2017, 2018]
```

# MATPLOTLIB

```
plt.bar()
```

```
[61] 1 plt.figure()  
2 plt.bar(x = x, height= tamanho)  
3 plt.show()
```



# MATPLOTLIB

plt.scatter()

## matplotlib.pyplot.scatter

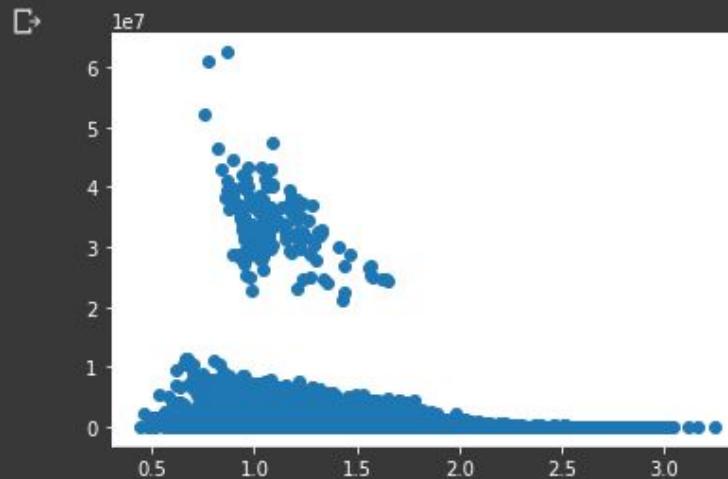
```
matplotlib.pyplot.scatter(x, y, s=None, c=None, marker=None, cmap=None, norm=None, vmin=None,  
vmax=None, alpha=None, linewidths=None, verts=<deprecated parameter>, edgecolors=None, *,  
plotnonfinite=False, data=None, **kwargs)
```

[source]

# MATPLOTLIB

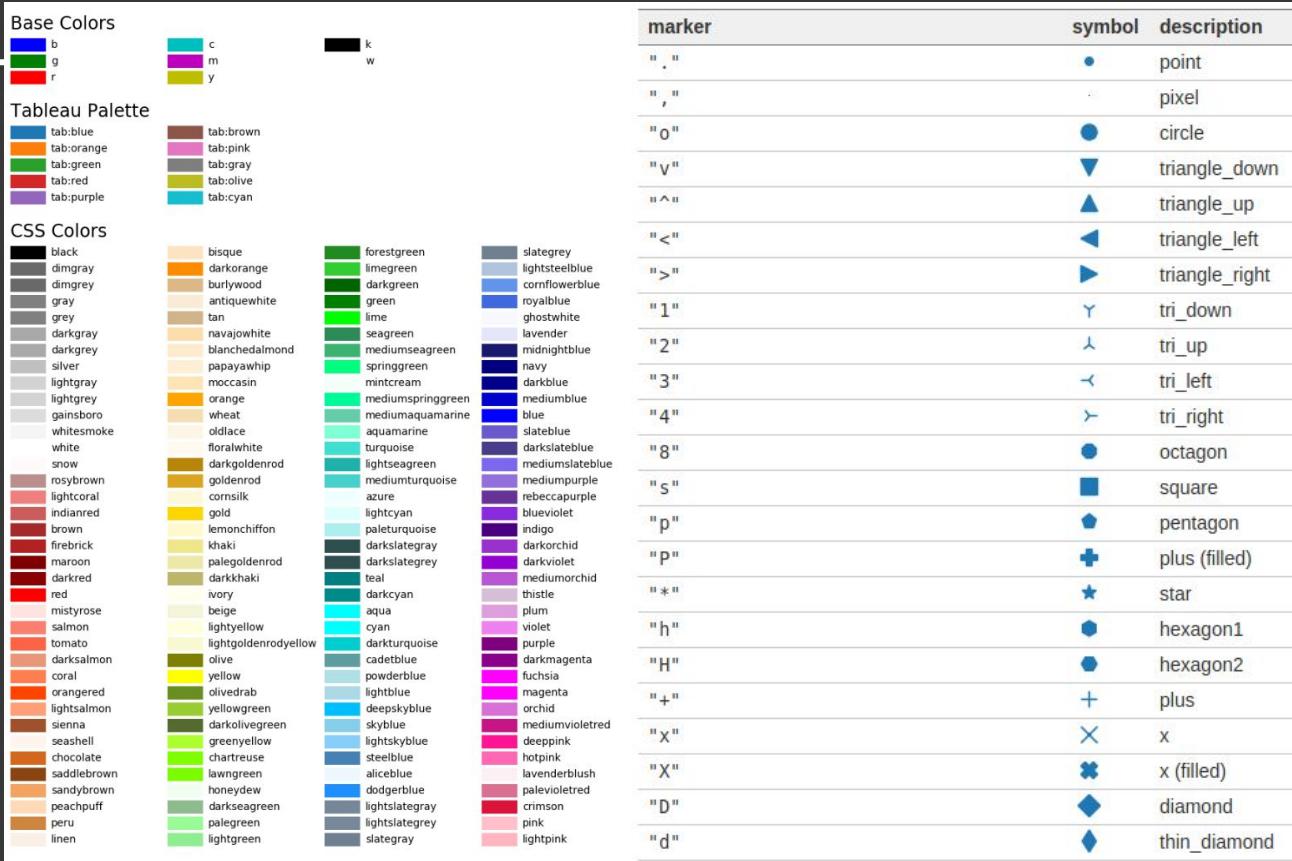
```
plt.scatter()
```

```
[15] 1 plt.figure()  
2 plt.scatter(x=df['AveragePrice'],y=df['Total Volume'])  
3 plt.show()
```



# MATPLOT

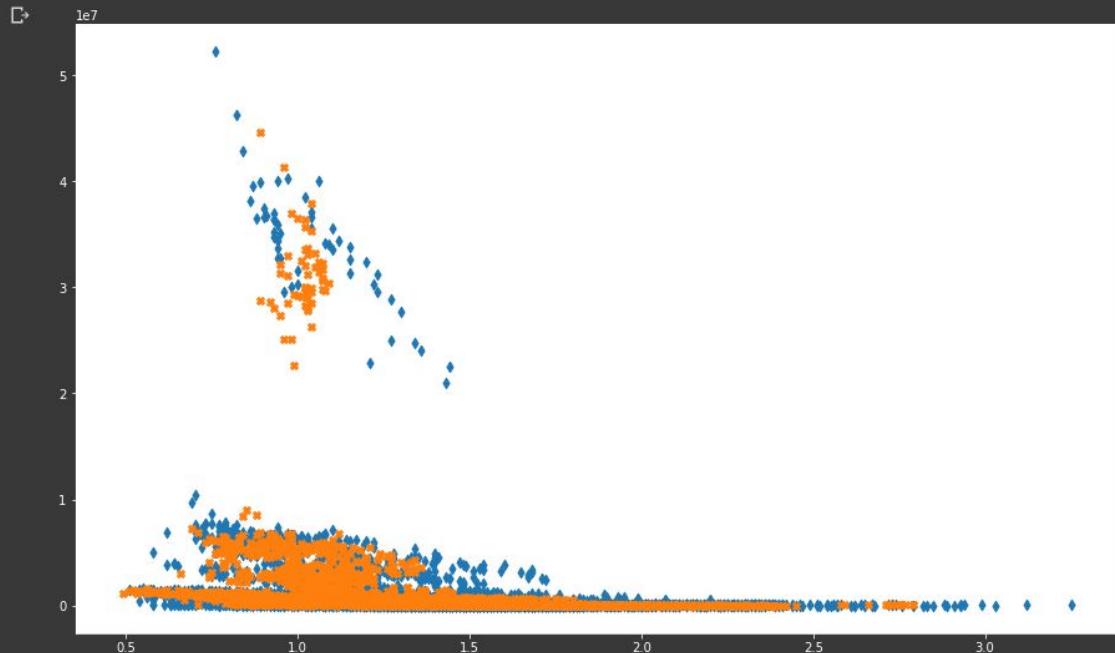
`plt.scatter()`



# MATPLOTLIB

plt.scatter()

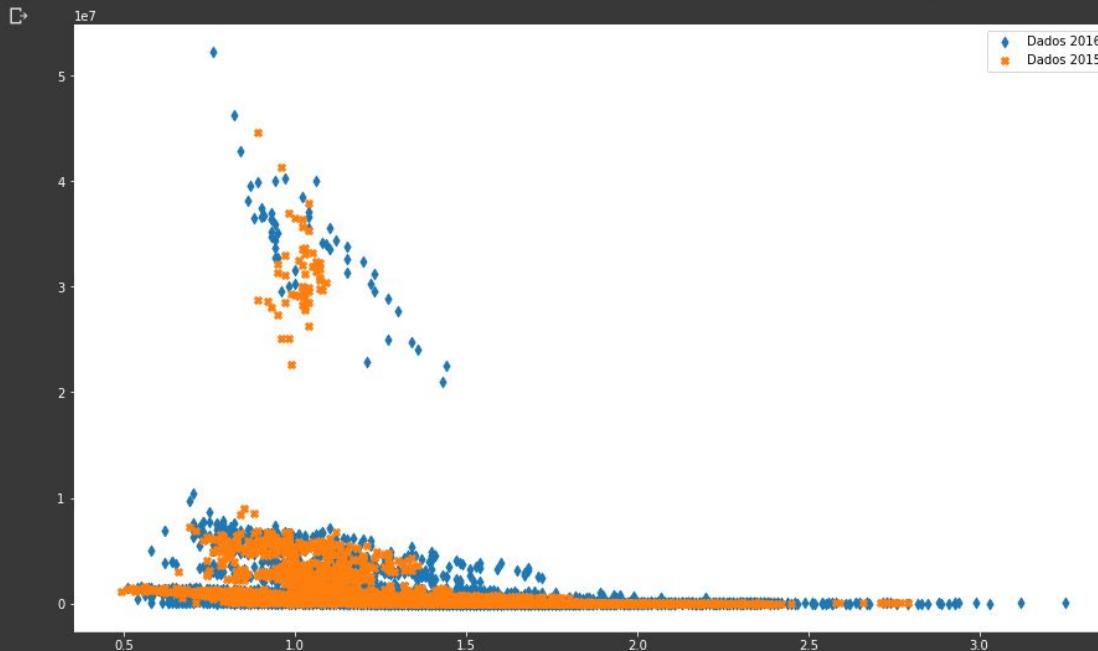
```
[22] 1 plt.figure(figsize=(15,9))
2 plt.scatter(x = df[df.year == 2016].AveragePrice, y = df[df.year == 2016]['Total Volume'],
3             c = 'tab:blue' , marker = 'd' )
4 plt.scatter(x = df[df.year == 2015].AveragePrice, y = df[df.year == 2015]['Total Volume'],
5             c = 'tab:orange' , marker = 'X' )
6 plt.show()
```



# MATPLOT

```
plt.legend()
```

```
[29] 1 ## adicionando legendas
2
3 plt.figure(figsize=(15,9))
4 plt.scatter(x = df[df.year == 2016].AveragePrice, y = df[df.year == 2016]['Total Volume'],
5             c = 'tab:blue' , marker = 'd' , label='Dados 2016')
6 plt.scatter(x = df[df.year == 2015].AveragePrice, y = df[df.year == 2015]['Total Volume'],
7             c = 'tab:orange' , marker = 'X', label='Dados 2015')
8
9 plt.legend()
10 plt.show()
```

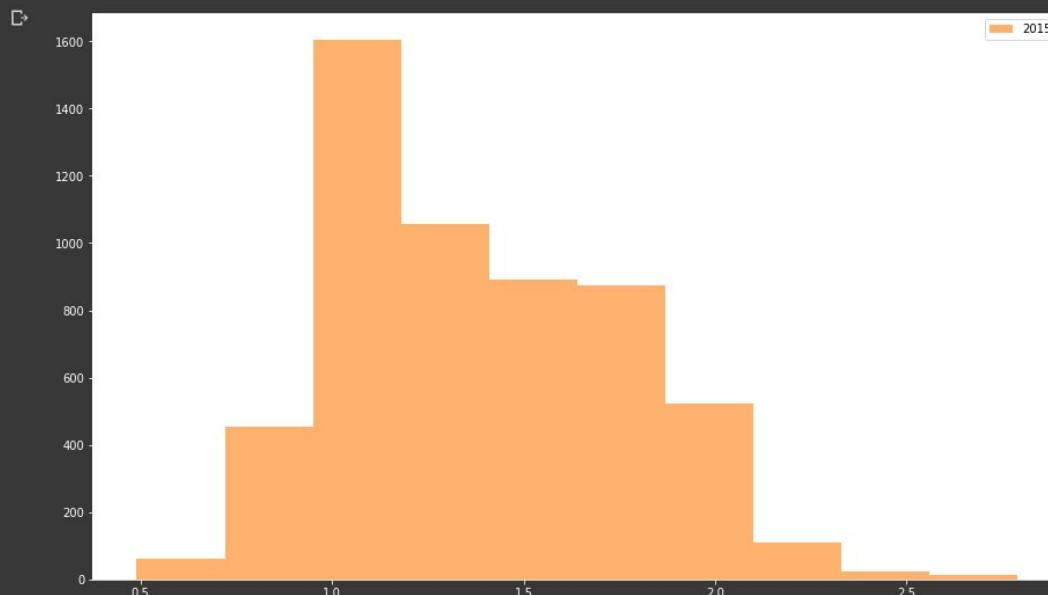


this method.

# MATPLOTLIB

```
plt.hist()
```

```
[64] 1 ## criando um histograma
2
3 plt.figure(figsize=(15,9))
4 plt.hist(x = df[df['year'] == 2015]['AveragePrice'],color='tab:orange',alpha=0.6,label='2015')
5 plt.legend()
6 plt.show()
```



# MATPLOTLIB

plt.axvline()

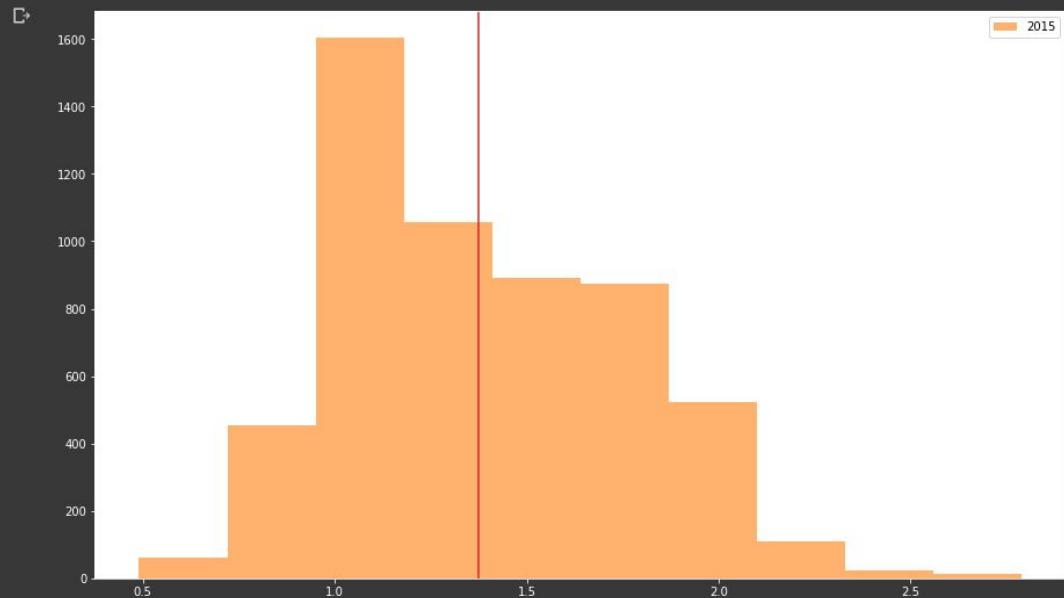
## matplotlib.pyplot.axvline

```
matplotlib.pyplot.axvline(x=0, ymin=0, ymax=1, **kwargs)
```

# MATPLOTLIB

plt.axvline()

```
[68] 1 ## encontrado a media dos valores
2
3 plt.figure(figsize=(15,9))
4 plt.hist(df[df['year'] == 2015]['AveragePrice'],color='tab:orange',alpha=0.5,label='2015')
5 plt.axvline(x = df[df['year'] == 2015]['AveragePrice'].mean(),color='tab:red')
6 plt.legend()
7 plt.show()
8
9
```



# MATPLOTLIB

plt.text()

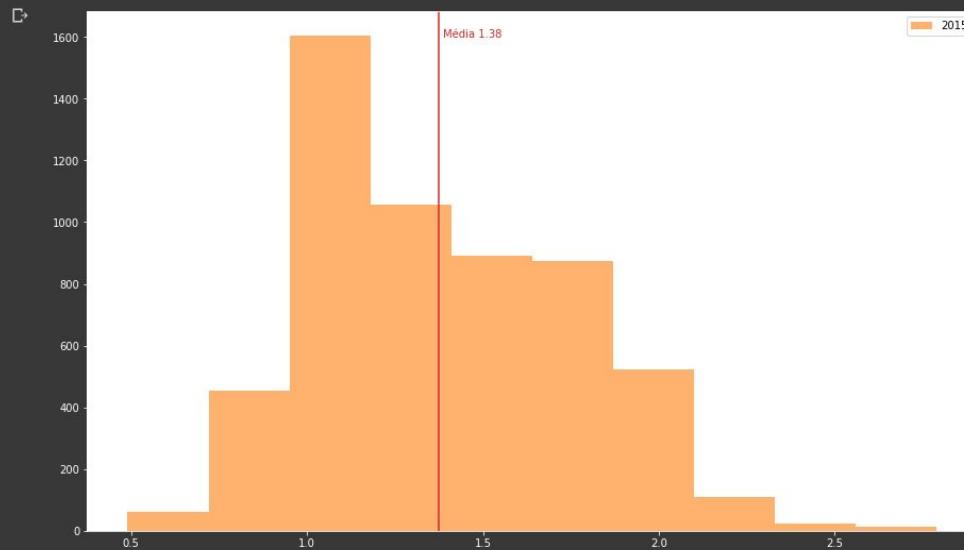
## matplotlib.pyplot.text

```
matplotlib.pyplot.text(x, y, s, fontdict=None, withdash=<deprecated parameter>, **kwargs) [source]
```

# MATPLOTLIB

plt.text()

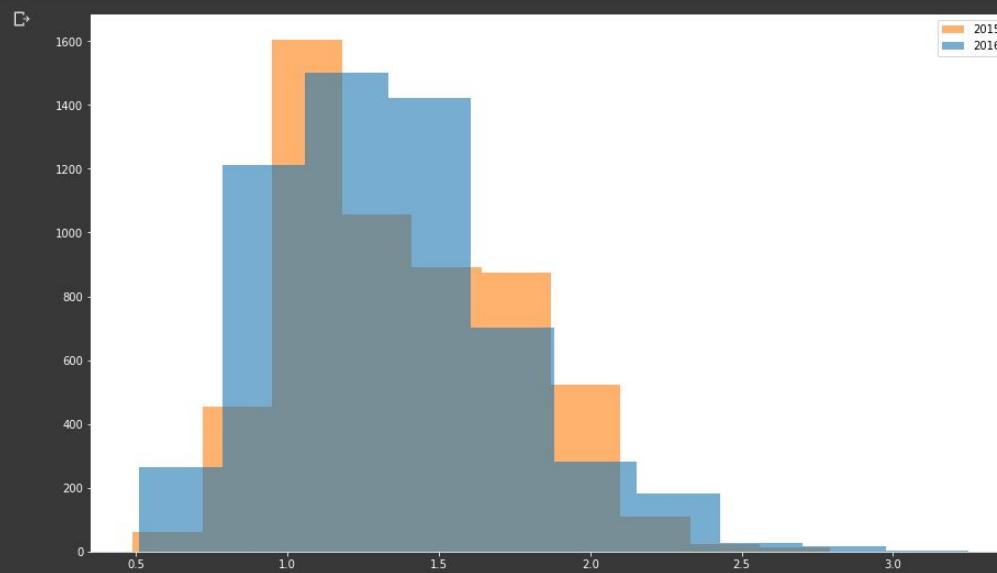
```
[77] 1 ## encontrado a media dos valores
2
3 media = df[df['year'] == 2015]['AveragePrice'].mean()
4
5 plt.figure(figsize=(15,9))
6 plt.hist(x = df[df['year'] == 2015]['AveragePrice'],color='tab:orange',alpha=0.6,label='2015')
7 plt.axvline(x = media ,color= 'tab:red')
8 plt.text(x = media + 0.01, y = 1600, s = f'Média {round(media,2)}', fontdict = {'color':'tab:red'})
9
10 plt.legend()
11 plt.show()
12
13
```



# MATPLOTLIB

plt.hist()

```
[65] 1 ## criando mais de um histograma na mesma figura
2
3 plt.figure(figsize=(15,9))
4 plt.hist(x = df[df['year'] == 2015]['AveragePrice'],color='tab:orange',alpha=0.6,label='2015')
5 plt.hist(x = df[df['year'] == 2016]['AveragePrice'],color='tab:blue',alpha=0.6,label='2016')
6 plt.legend()
7 plt.show()
```



# MATPLOTLIB

plt.title()

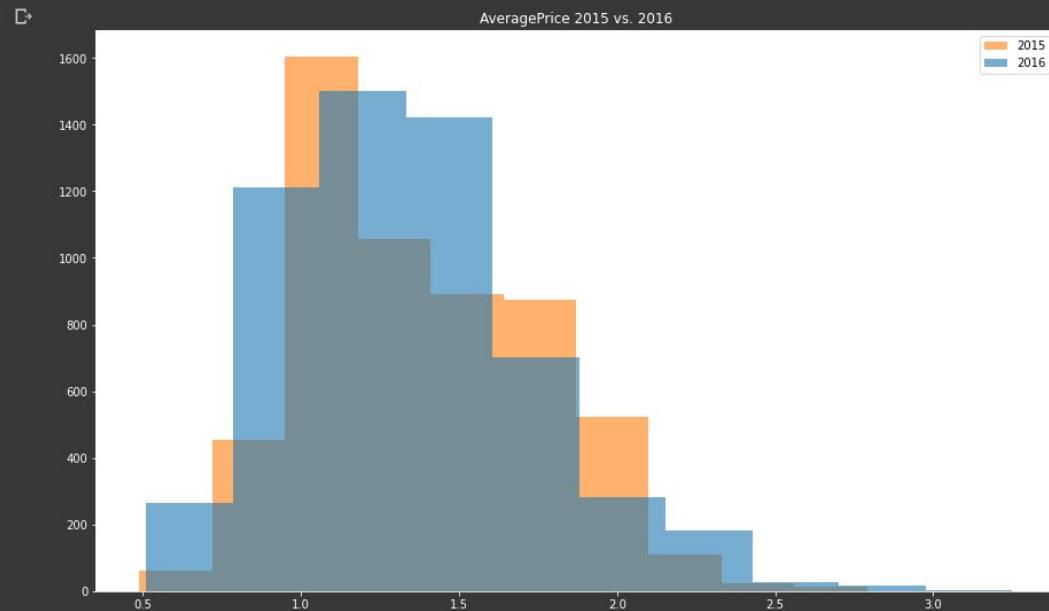
## matplotlib.pyplot.title

```
matplotlib.pyplot.title(label, fontdict=None, loc='center', pad=None, **kwargs)
```

# MATPLOTLIB

```
plt.title()
```

```
[81] 1 ## gerando o gráfico acima com título  
2  
3 plt.figure(figsize=(15,9))  
4 plt.hist(x = df[df['year'] == 2015]['AveragePrice'],color='tab:orange',alpha=0.6,label='2015')  
5 plt.hist(x = df[df['year'] == 2016]['AveragePrice'],color='tab:blue',alpha=0.6,label='2016')  
6 plt.title('AveragePrice 2015 vs. 2016',fontdict={'color':'white'})  
7 plt.legend()  
8 plt.show()
```



# MATPLOTLIB

plt.plot()

## matplotlib.pyplot.plot

```
matplotlib.pyplot.plot(*args, scalex=True, scaley=True, data=None, **kwargs)
```

Plot y versus x as lines and/or markers.

### Parameters:

x, y : array-like or scalar

The horizontal / vertical coordinates of the data points. x values are optional and default to range(len(y)).

Commonly, these parameters are 1D arrays.

They can also be scalars, or two-dimensional (in that case, the columns represent separate data sets).

These arguments cannot be passed as keywords.

# MATPLOTLIB

plt.plot()

```
1 ## gerando os dados necessários para o próximo gráfico
2
3 dados = (
4
5     df[['Date','Total Bags','year','region']]
6     .query('year == 2015')
7     .query("region == 'LasVegas'")
8     .drop(['year','region'],axis=1)
9     .assign(Date = lambda x: pd.to_datetime(x.Date,infer_datetime_format=True))
10    .sort_values('Date')
11
12
13
14 )
```

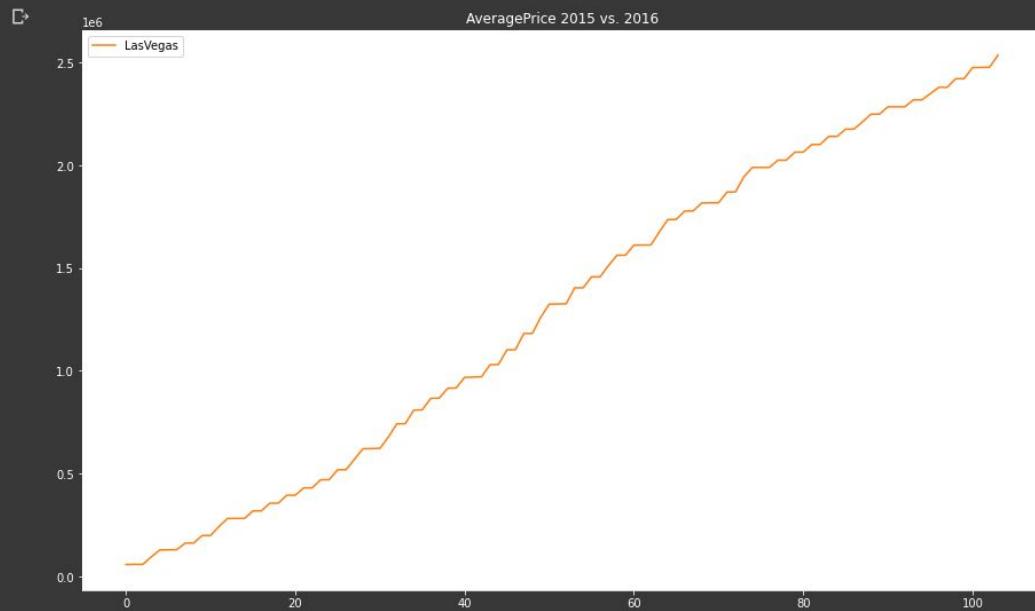
	Date	Total Bags
1143	2015-01-04	58638.68
10269	2015-01-04	457.12
10268	2015-01-11	33.33
1142	2015-01-11	37144.36
1141	2015-01-18	33175.55
...	...	...
10220	2015-12-13	509.14
1093	2015-12-20	54103.94
10219	2015-12-20	407.88
10218	2015-12-27	1015.50
1092	2015-12-27	58612.10

104 rows × 2 columns

# MATPLOTLI

```
plt.plot()
```

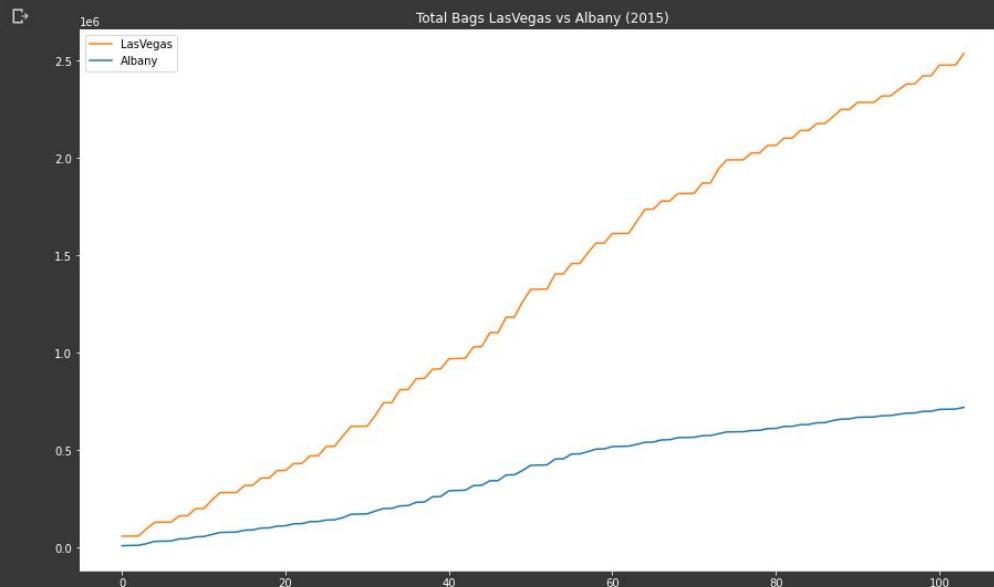
```
[117] 1 ## gerando o gráfico de linhas com a soma cumulativa
2
3 soma_cumulativa = dados['Total Bags'].cumsum()
4
5 plt.figure(figsize=(15,9))
6 plt.plot(range(104),soma_cumulativa,color='tab:orange',label='LasVegas')
7 plt.title('AveragePrice 2015 vs. 2016',fontdict={'color':'white'})
8 plt.legend()
9 plt.show()
```



# MATPLOTLIB

plt.plot()

```
[20] 1 ## gerando o gráfico de linhas com a soma cumulativa
2
3 soma_cumulativa = dados['Total Bags'].cumsum()
4 soma_cumulativa_albany = dados_albany['Total Bags'].cumsum()
5
6 plt.figure(figsize=(15,9))
7 plt.plot(range(104),soma_cumulativa,color='tab:orange',label='LasVegas')
8 plt.plot(range(104),soma_cumulativa_albany,color='tab:blue',label='Albany')
9 plt.title('Total Bags LasVegas vs Albany (2015)',fontdict={'color':'white'})
10 plt.legend()
11 plt.show()
```



# MATPLOTLIB

plt.axhline()

## matplotlib.pyplot.axhline

```
matplotlib.pyplot.axhline(y=0, xmin=0, xmax=1, **kwargs)
```

[\[source\]](#)

Add a horizontal line across the axis.

### Parameters:

**y** : scalar, optional, default: 0

y position in data coordinates of the horizontal line.

**xmin** : scalar, optional, default: 0

Should be between 0 and 1, 0 being the far left of the plot, 1 the far right of the plot.

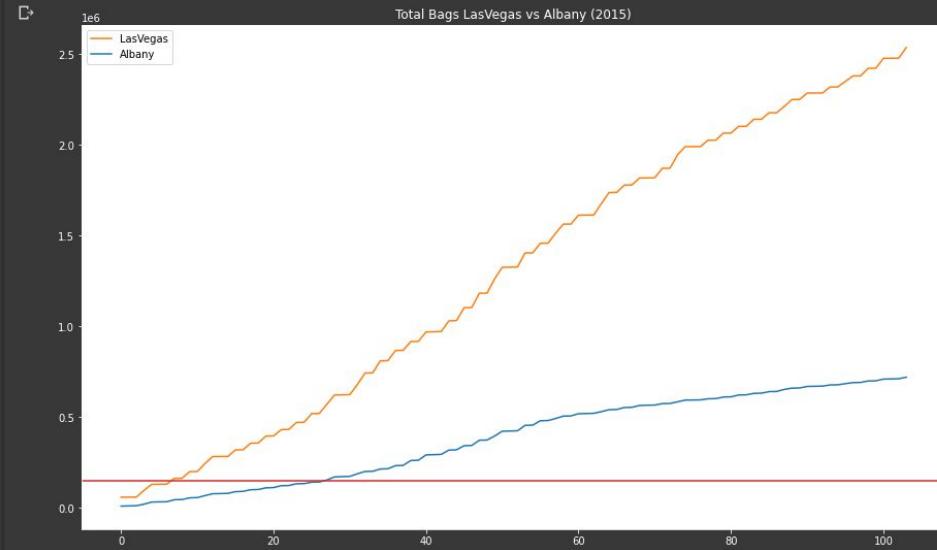
**xmax** : scalar, optional, default: 1

Should be between 0 and 1, 0 being the far left of the plot, 1 the far right of the plot.

# MATPLOTLIB

plt.axhline()

```
1 ## adicionando uma linha horizontal no valor 150000
2
3 soma_cumulativa = dados['Total Bags'].cumsum()
4 soma_cumulativa_albany = dados_albany['Total Bags'].cumsum()
5
6 plt.figure(figsize=(15,9))
7 plt.plot(range(104),soma_cumulativa,color='tab:orange',label='LasVegas')
8 plt.plot(range(104),soma_cumulativa_albany,color='tab:blue',label='Albany')
9 plt.title('Total Bags LasVegas vs Albany (2015)',fontdict={'color':'white'})
10
11 plt.axhline(y=150000,color='tab:red')
12
13 plt.legend()
14 plt.show()
```

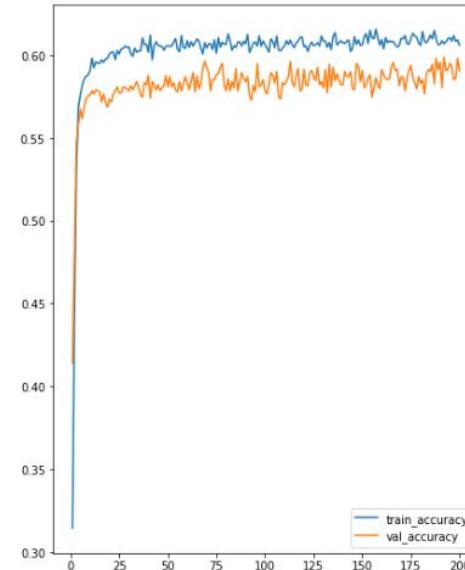
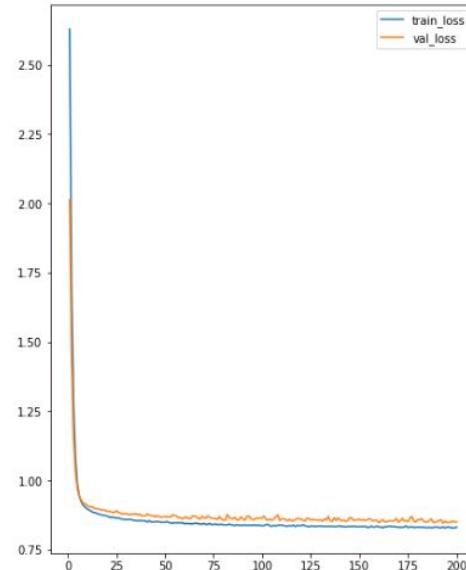


# MATPLOTLIB

```
plt.subplots()
```

```
In [10]: ## gerando as curvas
```

```
model.generate_loss_curves()
```



# MATPLOTLIB

```
plt.subplots()
```

## matplotlib.pyplot.subplots

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

[source]

Create a figure and a set of subplots.

This utility wrapper makes it convenient to create common layouts of subplots, including the enclosing figure object, in a single call.

**Returns:**

`fig : Figure`

`ax : axes.Axes` object or array of Axes objects.

`ax` can be either a single `Axes` object or an array of Axes objects if more than one subplot was created. The dimensions of the resulting array can be controlled with the `squeeze` keyword, see above.

# MATPLOTLIB

```
plt.subplots()
```

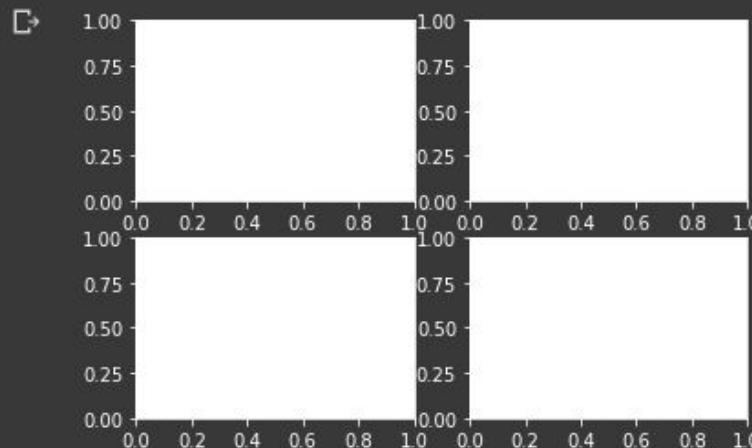
```
[39] 1 ## criando um subplot vazio  
2  
3 plt.subplots()  
4 plt.show()
```



# MATPLOTLIB

```
plt.subplots()
```

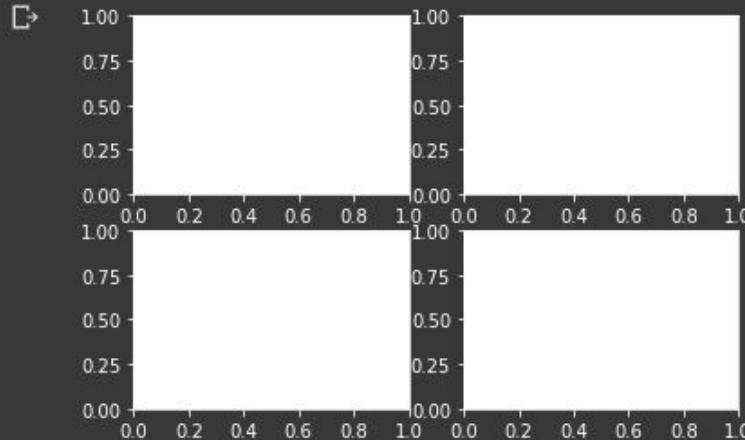
```
[44] 1 ## criando um subplot com 6 axes (2 linhas e 2 colunas)
2
3 plt.subplots(2,2)
4 plt.show()
```



# MATPLOTLIB

```
plt.subplots()
```

```
[48] 1 ## guardando os retornos em variave
      2
      3 f, ax = plt.subplots(2,2)
      4 plt.show()
```



```
[50] 1 ## verificando o tipo de f e ax
      2
      3 print(type(f))
      4 print(type(ax))
```

```
<class 'matplotlib.figure.Figure'>
<class 'numpy.ndarray'>
```

# MATPLOTLIB

```
plt.subplots()
```

```
[50] 1 ## verificando o tipo de f e ax  
2  
3 print(type(f))  
4 print(type(ax))
```

```
↳ <class 'matplotlib.figure.Figure'>  
<class 'numpy.ndarray'>
```

# MATPLOTLIB

```
plt.subplots()
```

## matplotlib.figure.Figure

```
class matplotlib.figure.Figure(figsize=None, dpi=None, facecolor=None, edgecolor=None,  
    linewidth=0.0, frameon=None, subplotpars=None, tight_layout=None, constrained_layout=None)[source]
```

Bases: [matplotlib.artist.Artist](#)

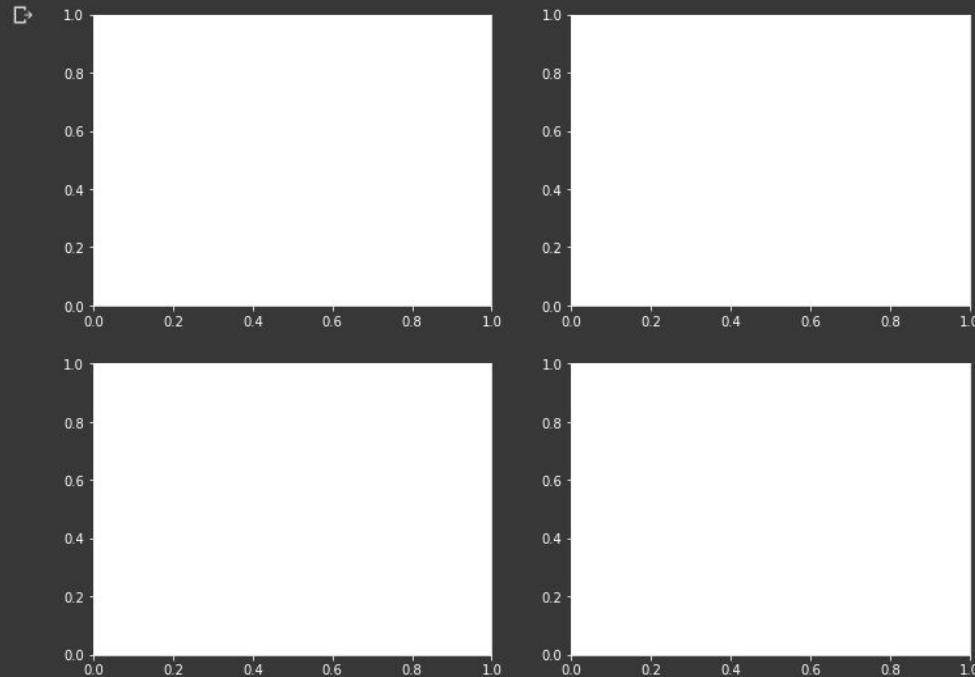
The top level container for all the plot elements.

The Figure instance supports callbacks through a `callbacks` attribute which is a [CallbackRegistry](#) instance. The events you can connect to are '`dpi_changed`', and the callback will be called with `func(fig)` where `fig` is the [Figure](#) instance.

# MATPLOTLIB

```
plt.subplots()
```

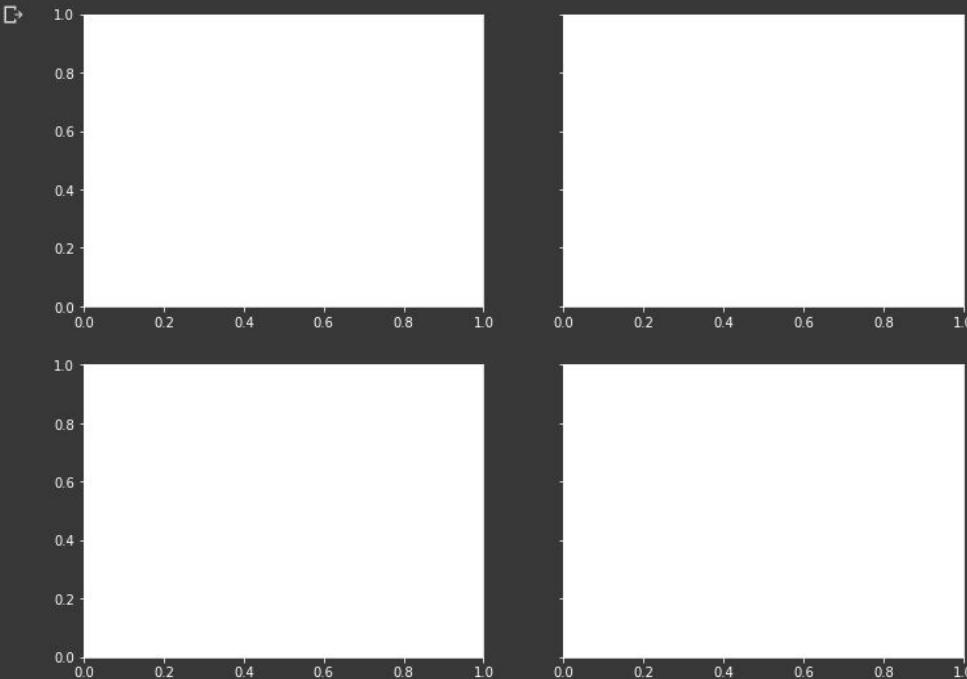
```
[51] 1 ## aumentado o tamanho do subplots  
2  
3 f, ax = plt.subplots(2,2,figsize=(12,9))  
4 plt.show()
```



# MATPLOTL

```
plt.subplots()
```

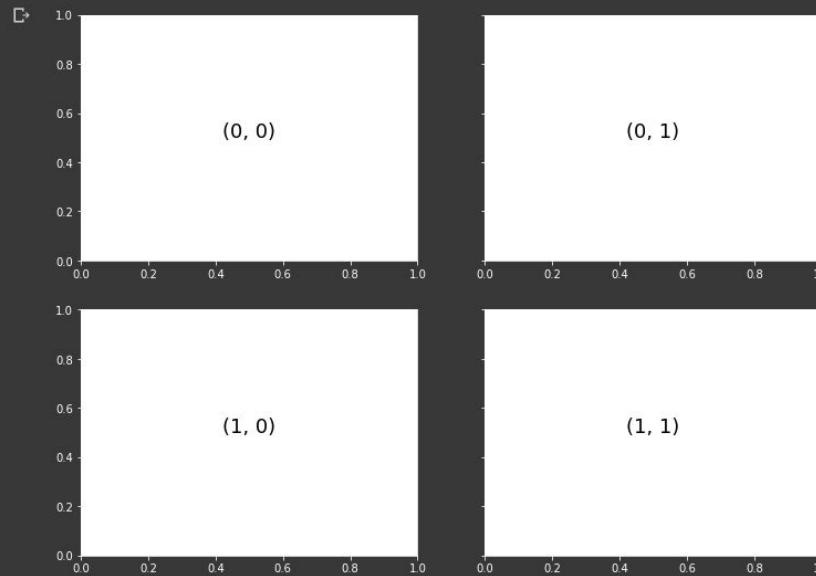
```
[52] 1 ## compartilhando os eixos y
      2
      3 f, ax = plt.subplots(2,2,figsize=(12,9),sharey=True)
      4 plt.show()
```



# MATPLOT

```
plt.subplots()
```

```
[70]: 1 ## acessando os gráficos e adicionando os textos
2
3 f, ax = plt.subplots(2,2,figsize=(12,9),sharey=True)
4
5
6 for i in range(2):
7     for j in range(2):
8         ax[i,j].text(0.5, 0.5, str((i, j)), fontsize=18, ha='center')
9
10 plt.show()
```



# MATPLOTLIB

```
[47] 1 ## definindo os dados que vamos usar  
2 subplot_data = df[['AveragePrice','year']]  
3 subplot_data  
plt.subplots()
```

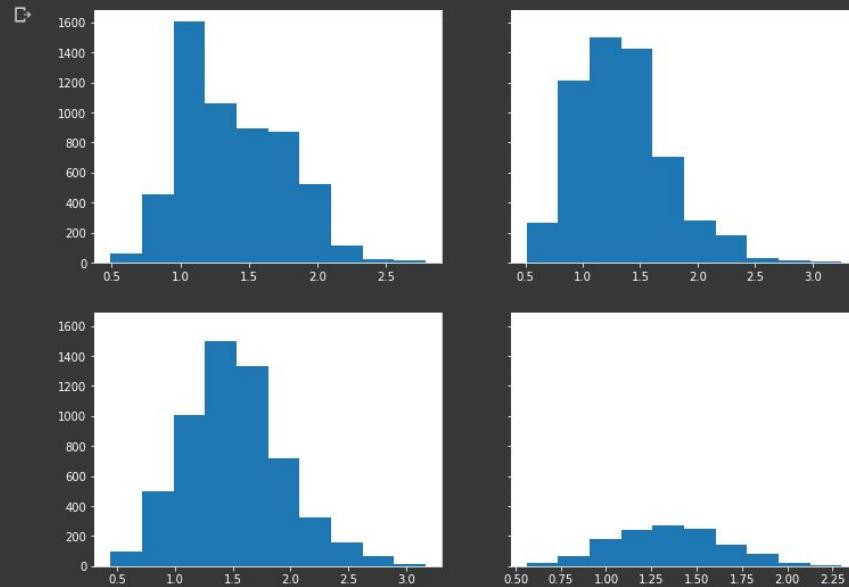
	AveragePrice	year
0	1.33	2015
1	1.35	2015
2	0.93	2015
3	1.08	2015
4	1.28	2015
...	...	...
18244	1.63	2018
18245	1.71	2018
18246	1.87	2018
18247	1.93	2018
18248	1.62	2018

18249 rows × 2 columns

# MATPLOTLIB

plt.subplots()

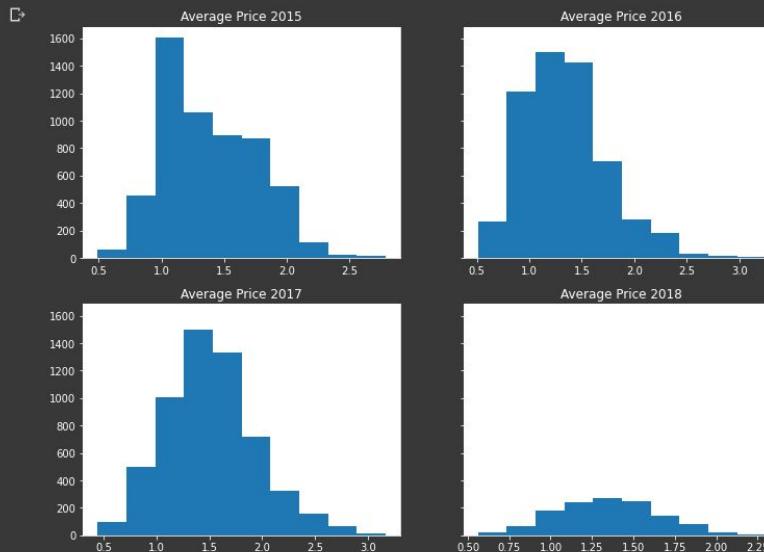
```
[74] 1 ## criando os gráficos em cada um dos eixos
2
3 f, ax = plt.subplots(2,2,figsize=(12,9),sharey=True)
4
5 ax[0,0].hist(subplot_data.query('year == 2015').AveragePrice)
6 ax[0,1].hist(subplot_data.query('year == 2016').AveragePrice)
7 ax[1,0].hist(subplot_data.query('year == 2017').AveragePrice)
8 ax[1,1].hist(subplot_data.query('year == 2018').AveragePrice)
9
10 plt.show()
```



# MATPLOTLIB

plt.subplots()

```
[77] 1 ## adicionando titulos
2
3 f, ax = plt.subplots(2,2,figsize=(12,9),sharey=True)
4
5 ax[0,0].hist(subplot_data.query('year == 2015').AveragePrice)
6 ax[0,0].set_title('Average Price 2015',fontdict={'color':'white'})
7
8 ax[0,1].hist(subplot_data.query('year == 2016').AveragePrice)
9 ax[0,1].set_title('Average Price 2016',fontdict={'color':'white'})
10
11 ax[1,0].hist(subplot_data.query('year == 2017').AveragePrice)
12 ax[1,0].set_title('Average Price 2017',fontdict={'color':'white'})
13
14 ax[1,1].hist(subplot_data.query('year == 2018').AveragePrice)
15 ax[1,1].set_title('Average Price 2018',fontdict={'color':'white'})
16
17 plt.show()
```



# MATPLOTLIB

[https://matplotlib.org/api/axes\\_api.html#plotting](https://matplotlib.org/api/axes_api.html#plotting)

plt.subplots()

## matplotlib.axes

### Table of Contents

- The Axes class
- Subplots
- Plotting
  - Basic
  - Spans
  - Spectral
  - Statistics
  - Binned
  - Contours
  - Array
  - Unstructured Triangles
  - Text and Annotations
  - Fields
- Clearing
- Appearance
- Property cycle
- Axis / limits
  - Axis Limits and direction
  - Axis Labels, title, and legend
  - Axis scales
- Autoscaling and margins
- Aspect ratio
- Ticks and tick labels
- Units
- Adding Artists
- Twinning
- Axes Position
- Async/Event based
- Interactive
- Children
- Drawing
- Bulk property manipulation
- General Artist Properties
- Artist Methods
- Projection
- Other
- Inheritance

# MATPLOTLIB

mpimg.imread()

```
[79] 1 ## importando o modulo image
      2
      3 import matplotlib.image as mpimg
```

# MATPLOTLIB

```
mpimg.imread()
```



# MATPLOTLIB

`mpimg.imread()`

```
[81] 1  ## abrindo a imagem
2
3  img = mpimg.imread('/content/ceu.jpeg')
4  img
D> array([[[ 58,  58,  92],
   [ 56,  56,  90],
   [ 55,  55,  89],
   ...,
   [ 22,  27,  56],
   [ 22,  27,  56],
   [ 22,  27,  56]],

   [[ 58,  58,  92],
   [ 57,  57,  91],
   [ 56,  56,  90],
   ...,
   [ 23,  28,  57],
   [ 23,  28,  57],
   [ 23,  28,  57]],

   [[ 59,  59,  93],
   [ 58,  58,  92],
   [ 57,  57,  91],
   ...,
   [ 22,  29,  58],
   [ 22,  29,  58],
   [ 22,  29,  58]],

   ...,

   [[ 85,  53,  38],
   [ 91,  59,  44],
   [ 94,  62,  47],
   ...,
   [ 58,  55,  62],
   [ 52,  50,  55],
   [ 47,  45,  50]],

   [[ 97,  66,  45],
   [106,  75,  54],
   [109,  78,  58],
   ...,
   [ 69,  64,  70],
   [ 65,  60,  66],
   [ 62,  57,  61]],

   [[[104,  70,  58],
   [113,  79,  67],
   [118,  84,  72],
   ...,
   [ 42,  41,  55],
   [ 44,  43,  57],
   [ 46,  46,  58]]], dtype=uint8)
```

# MATPL

```
[84] 1 ## abrindo o array com a imagem
2
3 plt.figure(figsize=(15,8))
4 plt.imshow(img)
5 plt.show()
```



# MATPLOTLIB

plt.savefig()

## matplotlib.pyplot.savefig

`matplotlib.pyplot.savefig(*args, **kwargs)`

[\[source\]](#)

Save the current figure.

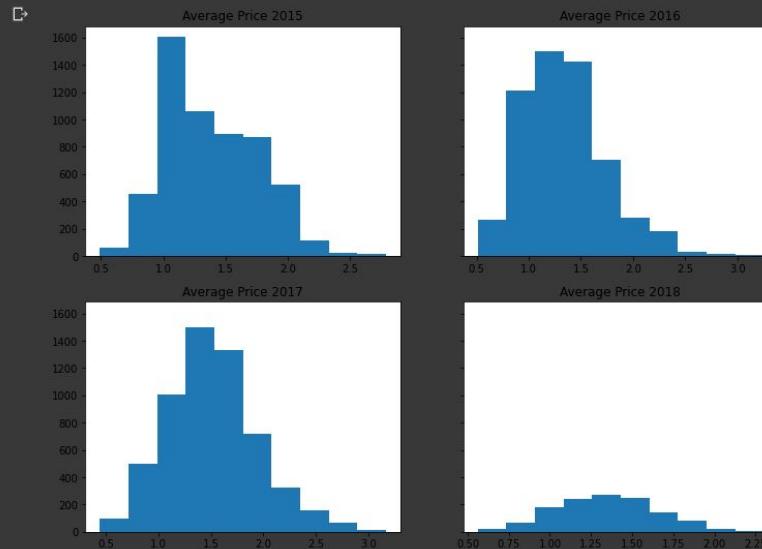
Call signature:

```
savefig(fname, dpi=None, facecolor='w', edgecolor='w',
        orientation='portrait', papertype=None, format=None,
        transparent=False, bbox_inches=None, pad_inches=0.1,
        frameon=None, metadata=None)
```

# MATPLOTLIB

```
plt.savefig()
```

```
[96] 1 ## salvando o subplot
2
3 f, ax = plt.subplots(2,2,figsize=(12,9),sharey=True)
4
5 ax[0,0].hist(subplot_data.query('year == 2015').AveragePrice)
6 ax[0,0].set_title('Average Price 2015',fontdict={'color':'black'})
7
8 ax[0,1].hist(subplot_data.query('year == 2016').AveragePrice)
9 ax[0,1].set_title('Average Price 2016',fontdict={'color':'black'})
10
11 ax[1,0].hist(subplot_data.query('year == 2017').AveragePrice)
12 ax[1,0].set_title('Average Price 2017',fontdict={'color':'black'})
13
14 ax[1,1].hist(subplot_data.query('year == 2018').AveragePrice)
15 ax[1,1].set_title('Average Price 2018',fontdict={'color':'black'})
16
17 plt.savefig(fname='subplot_bonitao',format='png')
18 plt.show()
```



# machine learning scikit-learn



# machine learning

*“Machine learning is the science of getting computers to act without being explicitly programmed.”*

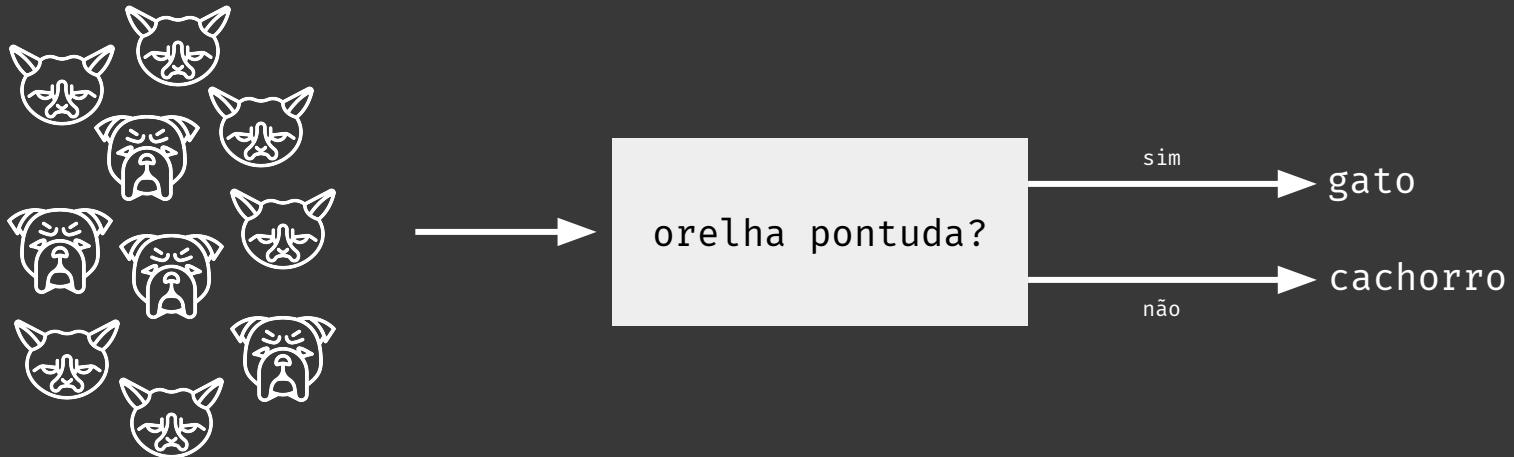
# machine learning

*“Machine learning is the science of getting computers to act without being explicitly programmed.”*



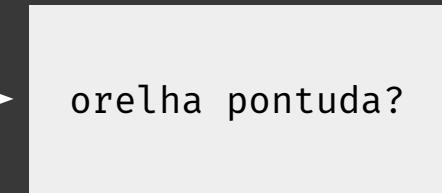
# machine learning

*“Machine learning is the science of getting computers to act without being explicitly programmed.”*



# machine learning

*“Machine learning is the science of getting computers to act without being explicitly programmed.”*



sim

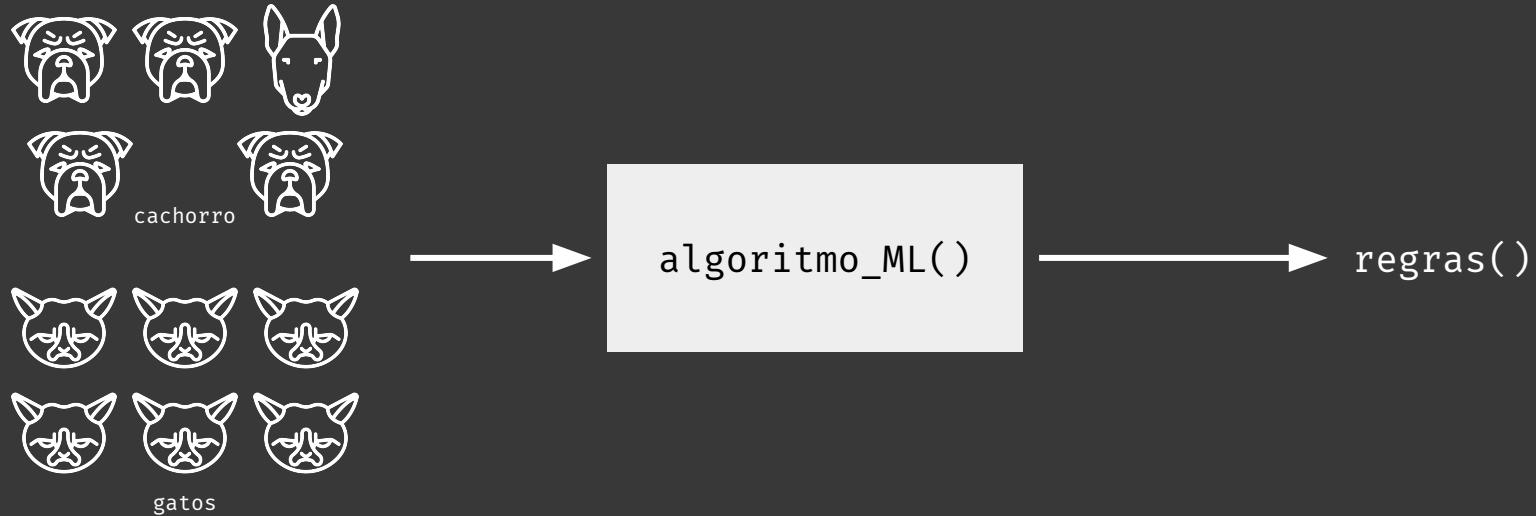
gato

não

cachorro

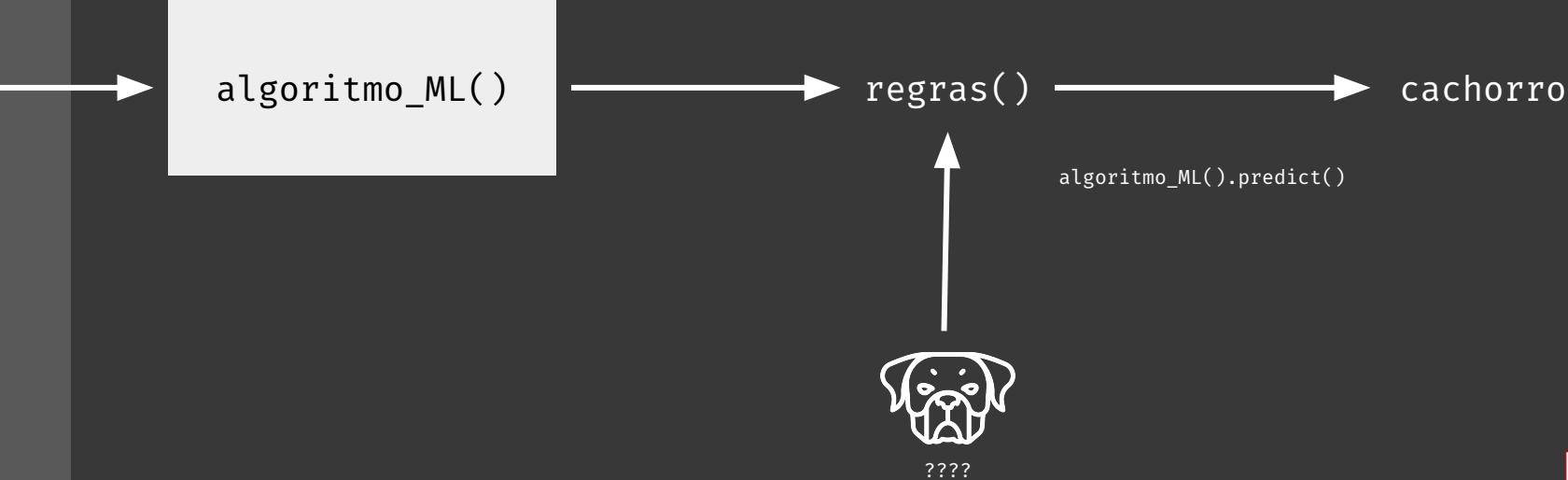
# machine learning

*“Machine learning is the science of getting computers to act without being explicitly programmed.”*



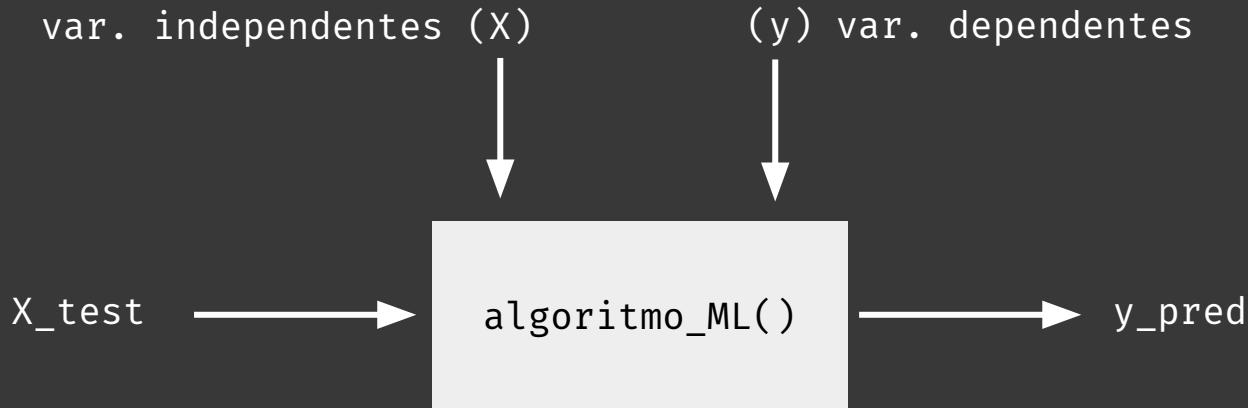
# machine learning

*“Machine learning is the science of getting computers to act without being explicitly programmed.”*



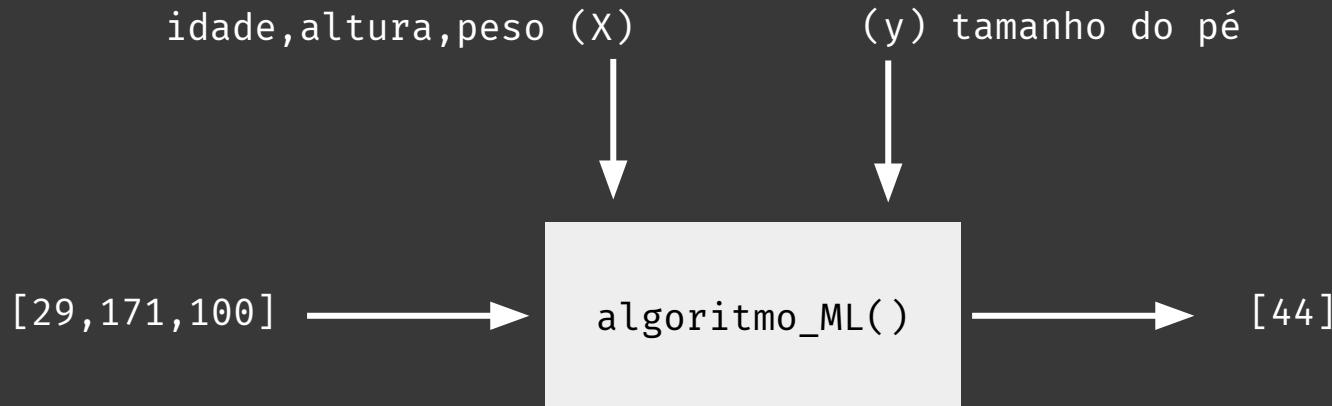
# machine learning

Tipos de aprendizado - Aprendizado Supervisionado



# machine learning

Tipos de aprendizado - Aprendizado Supervisionado



# machine learning

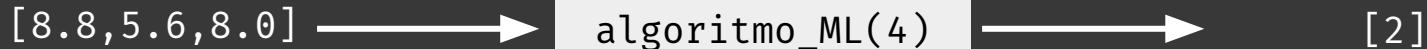
Tipos de aprendizado - Aprendizado Não-Supervisionado



# machine learning

Tipos de aprendizado - Aprendizado Não-Supervisionado

matemática, química, física (X)



# machine learning

Tipos de aprendizado - Aprendizado por Reforço



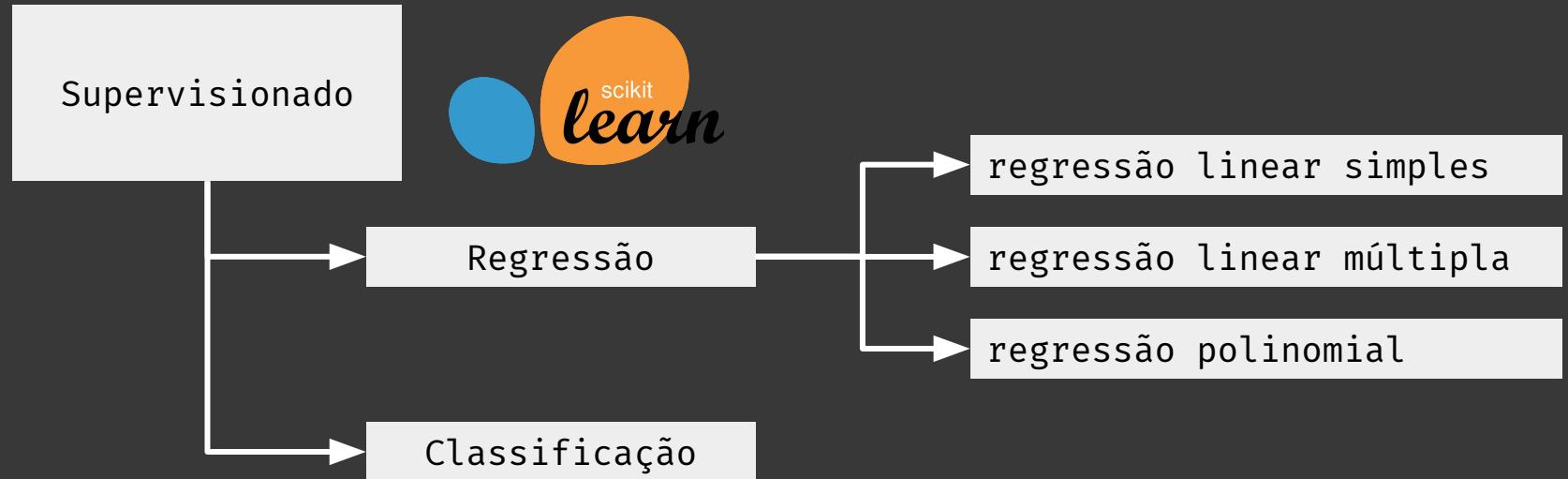
# machine learning

Tipos de aprendizado - Aprendizado por Reforço



# machine learning

Tipos de aprendizado - Nosso escopo



# machine learning

Tipos de aprendizado - Nosso escopo



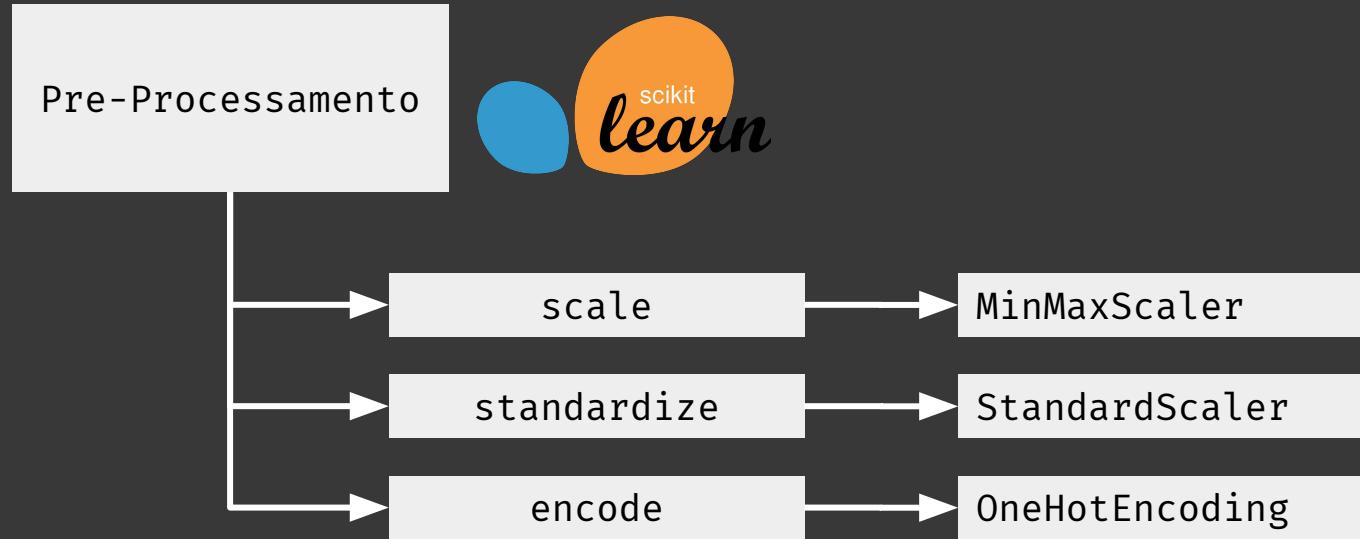
# machine learning

Tipos de aprendizado - Nosso escopo



# machine learning

Conceitos e Utilidades - Nosso escopo



# machine learning

Conceitos e Utilidades - Nosso escopo



# machine learning

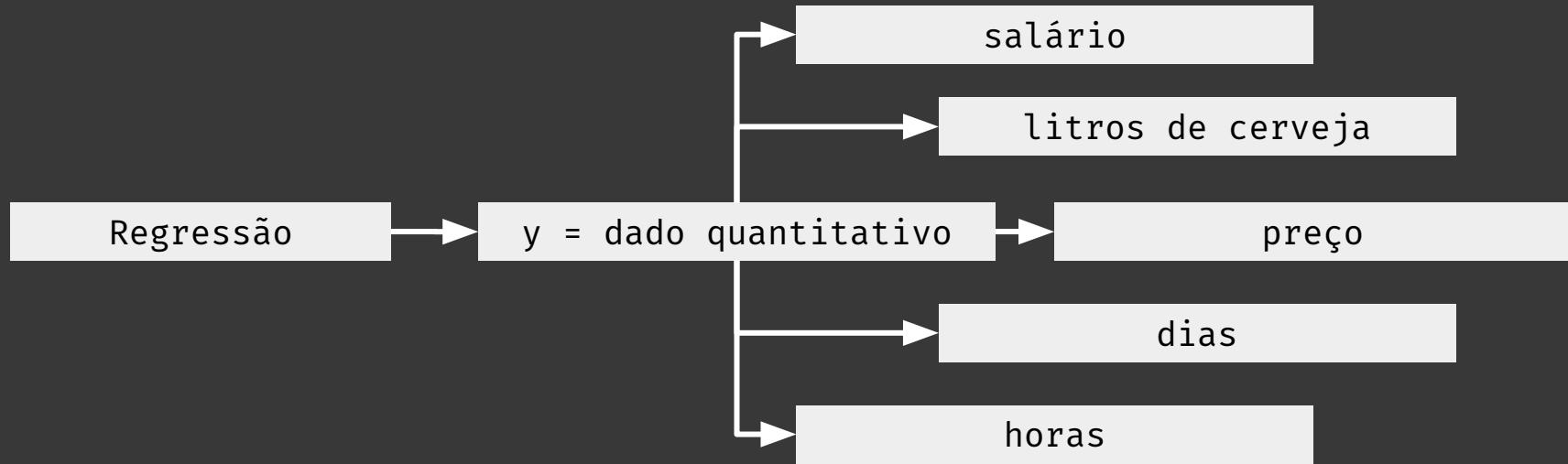
conceitos e Utilidades - Nossa escopo

Seleção de Modelos



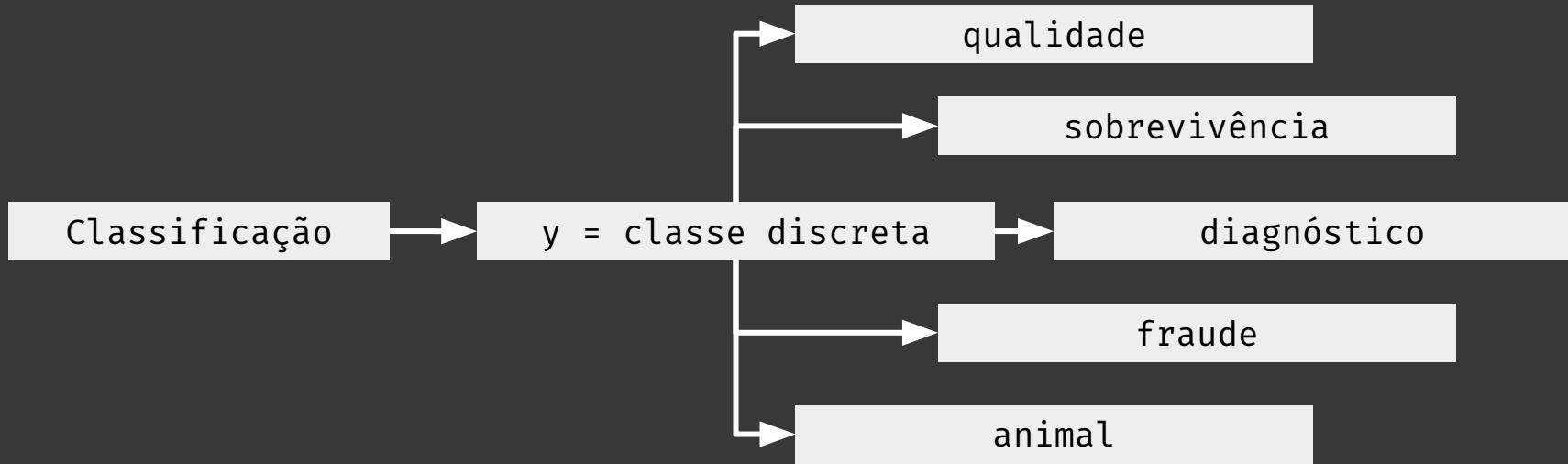
# machine learning

Definição de tarefas (tasks)



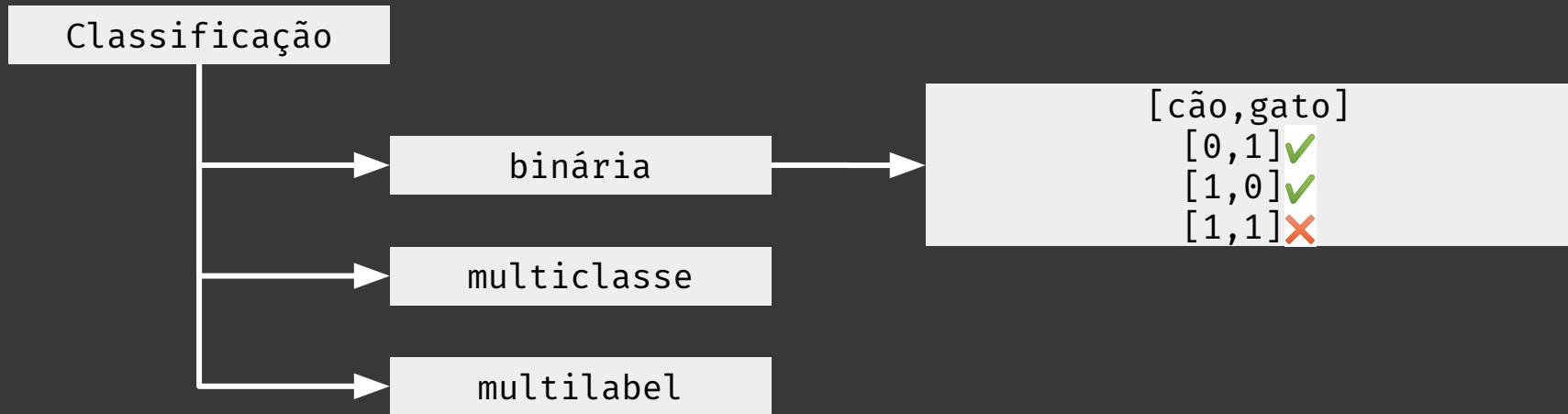
# machine learning

Definição de tarefas (tasks)



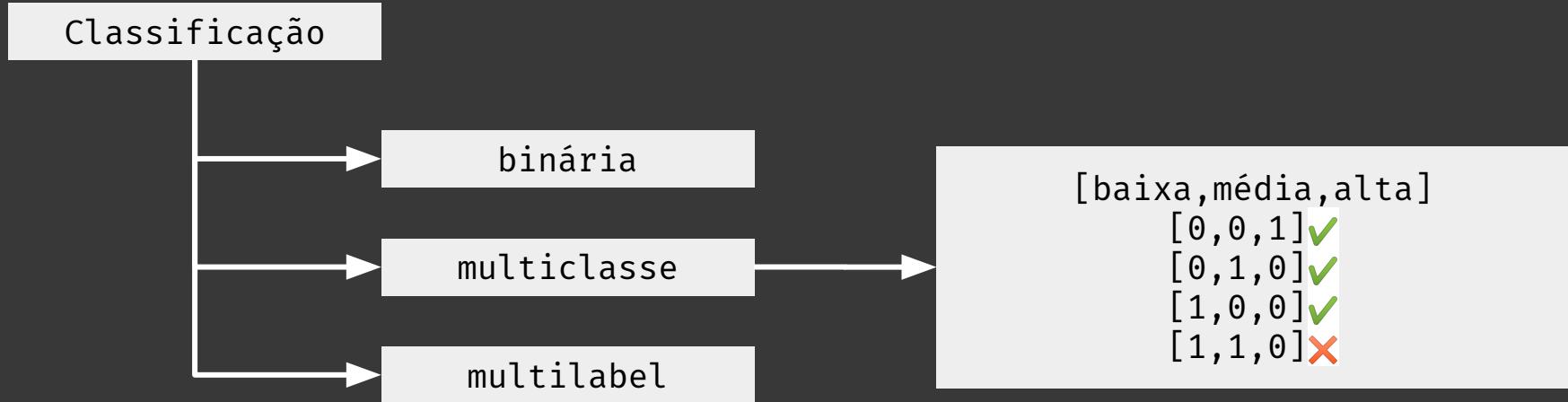
# machine learning

Definição de tarefas (tasks)



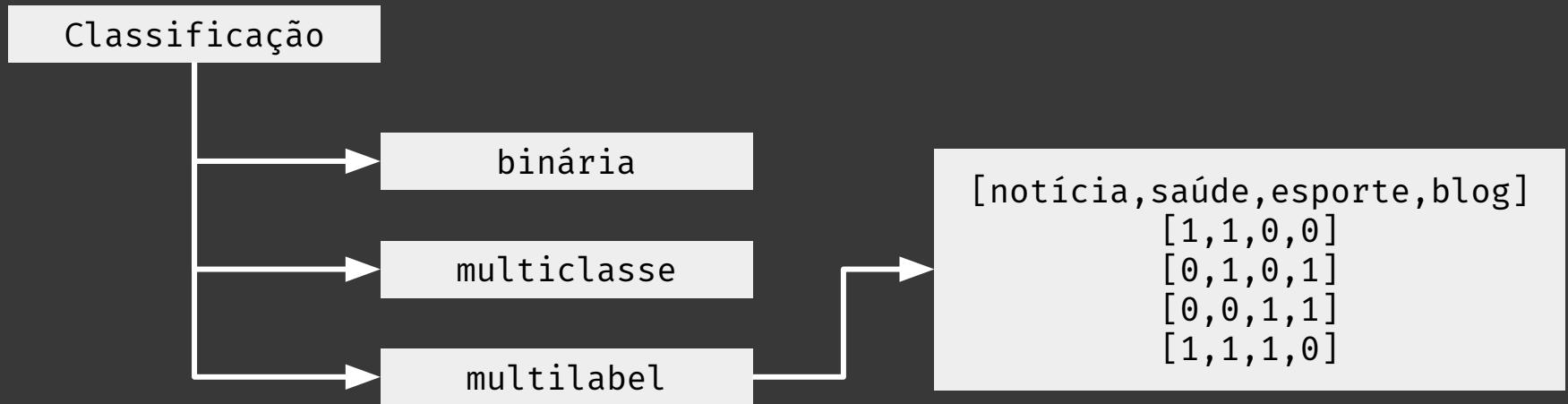
# machine learning

Definição de tarefas (tasks)



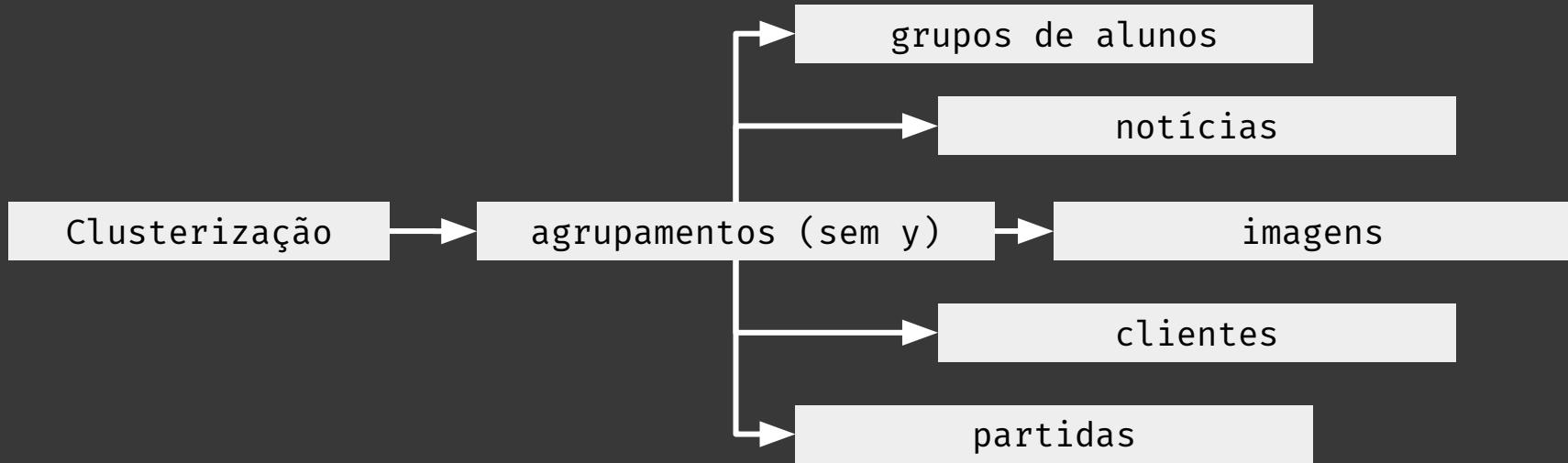
# machine learning

Definição de tarefas (tasks)



# machine learning

Definição de tarefas (tasks)



# machine learning

Conceitos iniciais - treino e teste

shape(100,6)

(25,6)

(25,6)

(25,6)

(25,6)

# machine learning

Conceitos iniciais - treino e teste

shape(100,6)

(25,6)

(25,6)

(25,6)

(25,6)

treino (75%)

teste (25%)

# machine learning

Conceitos iniciais - treino e teste

```
[2] 1 from sklearn.model_selection import train_test_split  
2 import pandas as pd  
  
[3] 1 df = pd.read_csv('/content/regression_dataset01_melbourne.csv',sep=',')  
2 df.head()
```

	Rooms	Distance	Bathroom	Car	Landsize	BuildingArea	Price
0	2	2.5	1.0	0.0	156.0	79.0	1035000.0
1	3	2.5	2.0	0.0	134.0	150.0	1465000.0
2	4	2.5	1.0	2.0	120.0	142.0	1600000.0
3	3	2.5	2.0	0.0	245.0	210.0	1876000.0
4	2	2.5	1.0	2.0	256.0	107.0	1636000.0

# machine learning

Conceitos iniciais - treino e teste

```
[7] 1 ## verificando o tamanho  
2 print(df.shape)
```

```
↳ (9244, 7)
```

```
[8] 1 ## verificando o tamanho de 4 partes iguais  
2 9244 / 4
```

```
↳ 2311.0
```

# machine learning

Conceitos iniciais - treino e teste

## `sklearn.model_selection.train_test_split`

```
sklearn.model_selection.train_test_split(*arrays, **options)
```

[\[source\]](#)

Split arrays or matrices into random train and test subsets

# machine learning

Conceitos iniciais - treino e teste

```
[11] 1 ## definindo X e y
2 X = df.drop('Price',axis=1)
3 y = df.Price
4
5 ## splitando o dataset
6 X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,random_state=42)
```

# machine learning

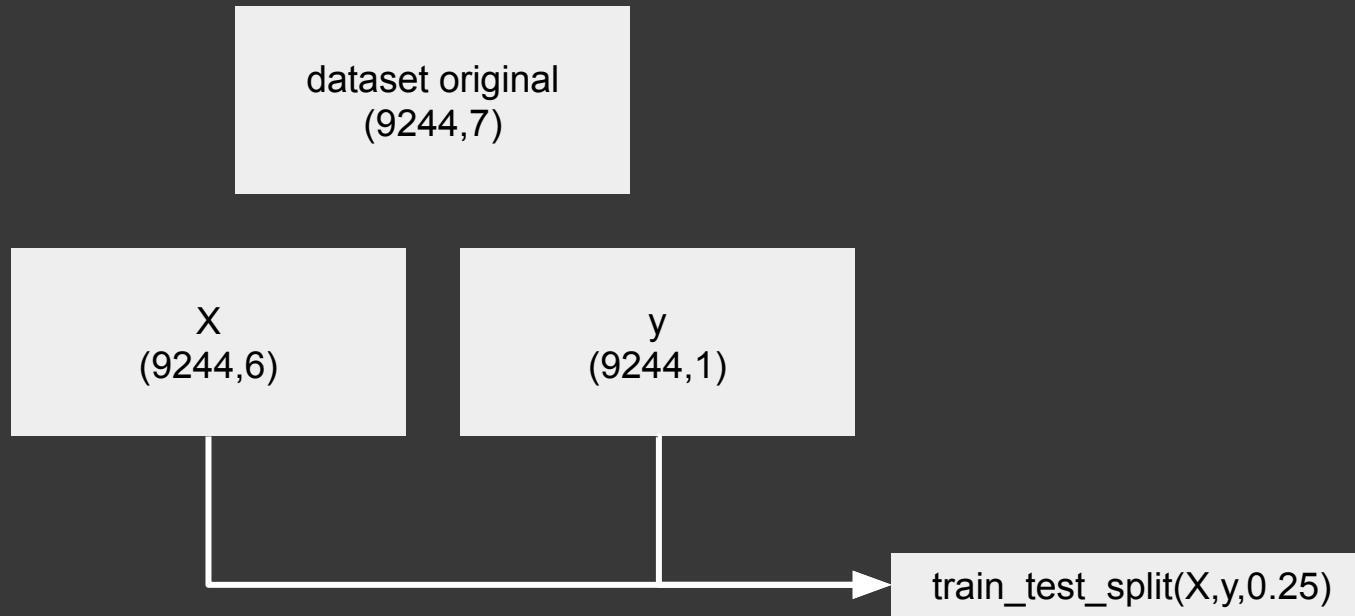
Conceitos iniciais - treino e teste

```
[17] 1 ## verificando o shape dos datasets de treino e teste  
2 print(X_train.shape,y_train.shape)  
3 print(X_test.shape,y_test.shape)
```

```
↳ (6933, 6) (6933,)  
(2311, 6) (2311,)
```

# machine learning

Conceitos iniciais - treino e teste



# machine learning

Conceitos iniciais - treino e teste

original  
4,7)



# machine learning

Conceitos iniciais - treino e teste

```
model.fit(X_train,y_train)
```

X\_train (6933,6)

y\_train (6933,1)

```
model.predict(X_test)
```

X\_test (2311,6)

# machine learning

Conceitos iniciais - treino e teste





Regressão!

# machine learning

Regressão

Posso descobrir o valor de Y com X?

Posso descobrir o tamanho do seu pé (Y) de acordo com a sua altura (X) ?

Por quanto vou vender essa casa?

Qual o limite devo dar para meu cliente?

Quanto de cerveja vai ser bebido hoje no bar?

# machine learning

Regressão Linear Simples

$$y(x) = ax + b$$

# machine learning

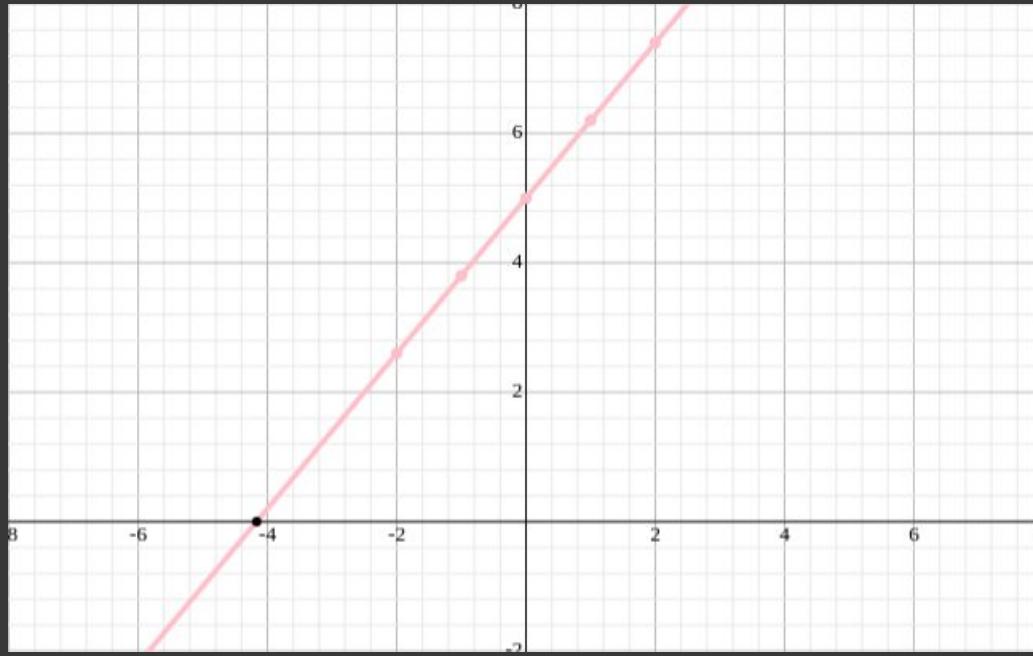
Regressão Linear Simples

$$y(x) = 5 + 1.2 * x$$

x	y
0	5
2	7.4
3	8.6
5	11
10	17

# machine learning

Regressão Linear Simples



# machine learning

Regressão Linear Simples

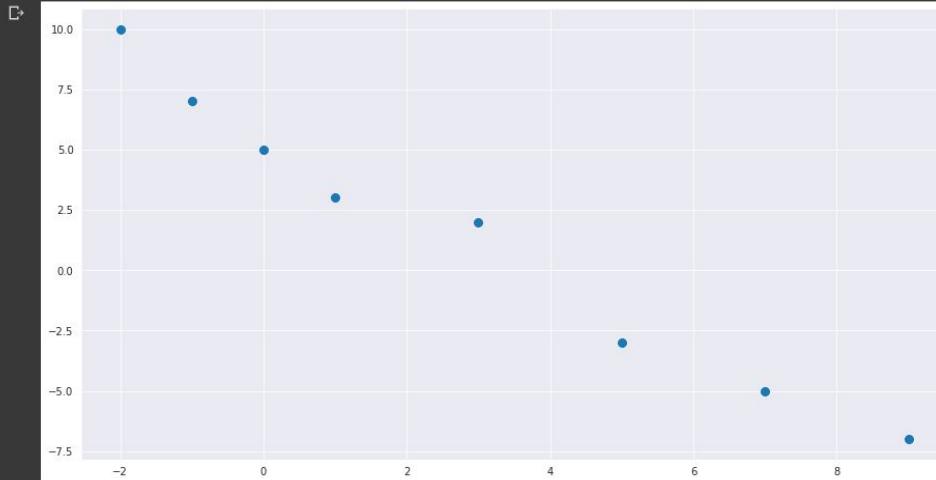
$$y(x) = ?? + ?? * x$$

x	y
-2	10
-1	7
0	5
1	3
3	2
5	-3
7	-5
9	-7

# machine learning

## Regressão Linear Simples

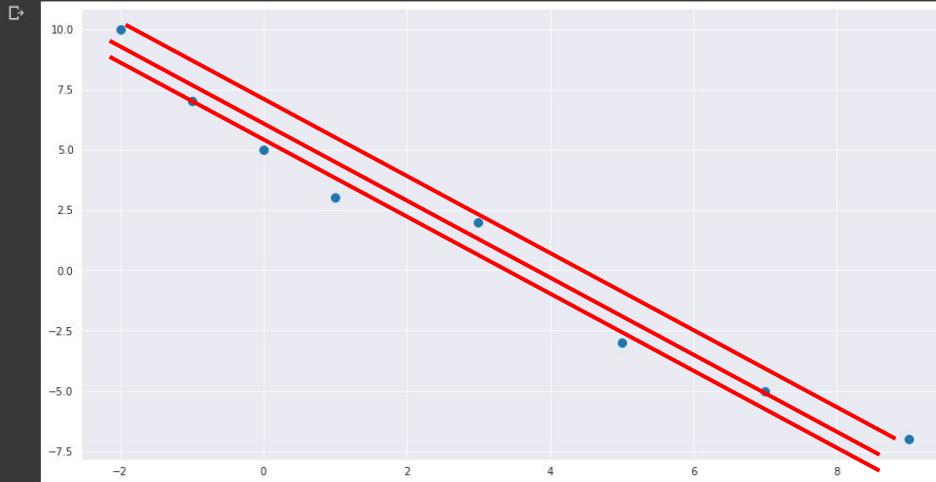
```
[41] 1 x = [-2,-1,0,1,3,5,7,9]
2 y = [10,7,5,3,2,-3,-5,-7]
3
4 plt.figure(figsize=(15,8))
5 sns.scatterplot(x=x,y=y,s=100)
6 plt.show()
```



# machine learning

## Regressão Linear Simples

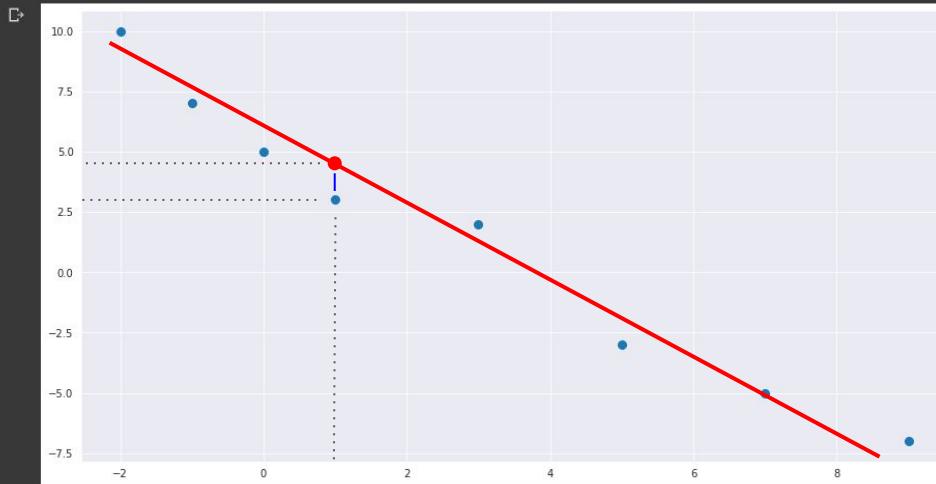
```
[41] 1 x = [-2,-1,0,1,3,5,7,9]
2 y = [10,7,5,3,2,-3,-5,-7]
3
4 plt.figure(figsize=(15,8))
5 sns.scatterplot(x=x,y=y,s=100)
6 plt.show()
```



# machine learning

## Regressão Linear Simples - Resíduo

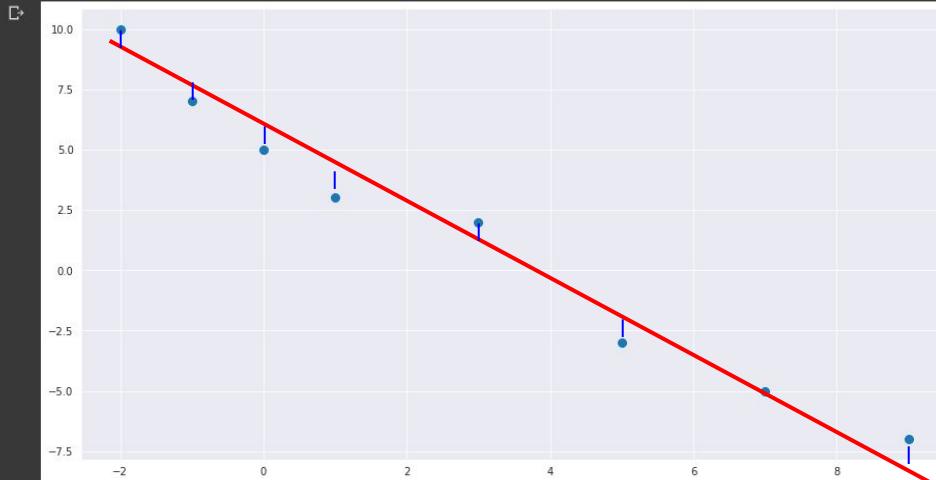
```
[41] 1 x = [-2,-1,0,1,3,5,7,9]
2 y = [10,7,5,3,2,-3,-5,-7]
3
4 plt.figure(figsize=(15,8))
5 sns.scatterplot(x=x,y=y,s=100)
6 plt.show()
```



# machine learning

## Regressão Linear Simples - Resíduo

```
[41] 1 x = [-2,-1,0,1,3,5,7,9]
2 y = [10,7,5,3,2,-3,-5,-7]
3
4 plt.figure(figsize=(15,8))
5 sns.scatterplot(x=x,y=y,s=100)
6 plt.show()
```



# machine learning

Regressão Linear Simples - cost function

```
resíduo = (y_real - y_pred)
```

```
cost_function = Σ(y_real - y_pred)^2
```

```
argmin ( cost_function )
```

# machine learning

Regressão Linear Simples

google sheets

# machine learning

## Regressão Linear Simples

```
[65] 1 ## conferindo o resultado do google sheets
2
3 from sklearn.linear_model import LinearRegression
4
5 x = np.array([-2,-1,0,1,3,5,7,9]).reshape(-1,1)
6 y = [10,7,5,3,2,-3,-5,-7]
7
8 lr = LinearRegression()
9 lr.fit(x,y)
10 lr.coef_, lr.intercept_
```

```
↳ (array([-1.49771689]), 5.6187214611872145)
y(x) = -1.49771689 * x + 5.6187214611872145
```

# machine learning

## Regressão Linear Simples

```
[76] 1 ## utilizando o predict  
2  
3 lr.predict([[1]])  
⇒ array([4.12100457])
```

$$y(x) = -1.49771689 * 1 + 5.6187214611872145$$

# machine learning

## Regressão Linear Simples

```
[77] 1 ## gerando várias predições
      2
      3 for i in range(10):
      4 | print(f'Predizendo para x={i} -- Resultado {lr.predict([[i]])}' )
```

```
↳ Predizendo para x=0 -- Resultado [5.61872146]
Predizendo para x=1 -- Resultado [4.12100457]
Predizendo para x=2 -- Resultado [2.62328767]
Predizendo para x=3 -- Resultado [1.12557078]
Predizendo para x=4 -- Resultado [-0.37214612]
Predizendo para x=5 -- Resultado [-1.86986301]
Predizendo para x=6 -- Resultado [-3.36757991]
Predizendo para x=7 -- Resultado [-4.8652968]
Predizendo para x=8 -- Resultado [-6.3630137]
Predizendo para x=9 -- Resultado [-7.86073059]
```

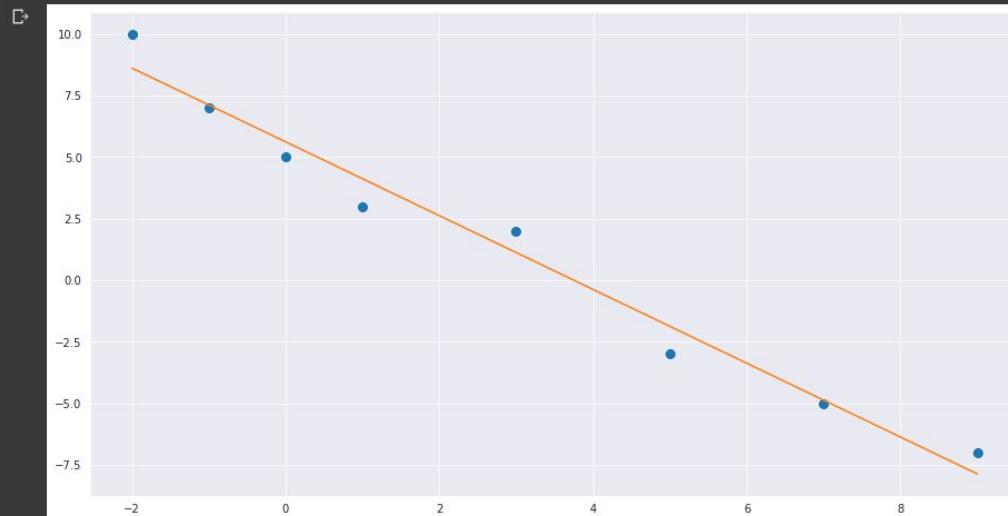
O modelo nunca viu o 4 nem o 8, como sabemos se esses valores estão próximos da realidade?

Pra isso temos que utilizar uma métrica! Em treino e teste!

# machine learning

Regressão Linear Simples

```
[129] 1 ## verificando o modelo montado
2
3 x = [-2,-1,0,1,3,5,7,9]
4 y = [10,7,5,3,2,-3,-5,-7]
5
6 y_pred = [lr.predict([[x]])[0] for x in x]
7
8 plt.figure(figsize=(15,8))
9 sns.scatterplot(x=x,y=y,s=100)
10 sns.lineplot(x=x,y=y_pred,color='tab:orange')
11 plt.show()
```



# machine learning

## Regressão Linear Simples

```
[68] 1 ## separando em treino e teste (0.25)
2
3 x_test = np.array([0,1]).reshape(-1,1)
4 y_test = [5,3]
5
6 x_treino = np.array([-2,-1,3,5,7,9]).reshape(-1,1)
7 y_treino = [10,7,2,-3,-5,-7]
8
9 lr2 = LinearRegression()
10 lr2.fit(x_treino,y_treino)
11 lr2.coef_, lr.intercept_

```

↳ (array([-1.54973822]), 5.6187214611872145)  
y(x) = -1.54973822 \* x + 5.6187214611872145

# machine learning

## Regressão Linear Simples

```
[71] 1 ## utilizando predict  
2  
3 print(y_test)  
4 lr2.predict(x_test)  
  
[5, 3]  
array([6.09075044, 4.54101222])
```

# machine learning

Regressão Linear Simples - Métrica MSE

$$MSE = \frac{1}{n} \sum \left( y - \hat{y} \right)^2$$

The square of the difference  
between actual and  
predicted

# machine learning

Regressão Linear Simples - Métrica MSE

```
[91] 1  ## definido nosso_mean_squared_error
2
3  def nosso_mean_squared_error(y_real,y_pred):
4  |  return ((y_real - y_pred)**2).mean()
```

# machine learning

Regressão Linear Simples - Métrica MSE

```
[93] 1 ## calculando nosso_mean_squared_error  
2  
3 valores_reais_de_y = np.array([5,3])  
4 valores_preditos_de_y = np.array([6.09075044, 4.54101222])  
5  
6 nosso_mean_squared_error(valores_reais_de_y, valores_preditos_de_y)  
  
↳ 1.7822275922747606
```

# machine learning

Regressão Linear Simples - Métrica MSE

```
[94] 1 ## usando a API do sklearn
2
3 from sklearn.metrics import mean_squared_error
4
5 mean_squared_error(y_test,lr2.predict(x_test))
```

```
↳ 1.782227582699063
```

A desvantagem dessa métrica é que ela pode ser bastante afetada por erros grandes já que os eleva ao quadrado ( $\text{erro}^2$ ). Ela também retorna um valor não representativo de Y.

# machine learning

Regressão Linear Simples - Métrica RMSE

$$RMSE = \sqrt{\frac{\sum_{i=1}^N (Predicted_i - Actual_i)^2}{N}}$$

# machine learning

Regressão Linear Simples - Métrica RMSE

```
[95] 1 ## definido nosso_ROOT_mean_squared_error
2
3 def nosso_ROOT_mean_squared_error(y_real,y_pred):
4     return np.sqrt(((y_real - y_pred)**2).mean())
```

# machine learning

Regressão Linear Simples - Métrica RMSE

```
[96] 1 ## calculando nosso_ROOT_mean_squared_error  
2  
3 nosso_ROOT_mean_squared_error(valores_reais_de_y,valores_preditos_de_y)  
[96] 1.3350009708890704
```

# machine learning

Regressão Linear Simples - erro em treino

```
[114] 1 ## vamos tentar calcular o erro de treino  
2  
3 print(nosso_mean_squared_error(y_treino,lr2.predict(x_treino)))  
4 print(nosso_ROOT_mean_squared_error(y_treino,lr2.predict(x_treino)))  
⇒ 0.662012798138453  
0.8136416890366748
```

# machine learning

Regressão Linear Múltipla

$$y(x, z) = ax + bz + c$$

# machine learning

## Regressão Linear Múltipla

```
[166] 1 ## lendo o dataset  
2 df = pd.read_csv('/content/regression_dataset01_melbourne.csv',sep=';')  
3 df.head()
```

	Rooms	Distance	Bathroom	Car	Landsize	BuildingArea	Price
0	2	2.5	1.0	0.0	156.0	79.0	1035000.0
1	3	2.5	2.0	0.0	134.0	150.0	1465000.0
2	4	2.5	1.0	2.0	120.0	142.0	1600000.0
3	3	2.5	2.0	0.0	245.0	210.0	1876000.0
4	2	2.5	1.0	2.0	256.0	107.0	1636000.0

# machine learning

## Regressão Linear Múltipla

```
[174] 1 ## descrevendo o dataset  
2  
3 df.describe()
```

	Rooms	Distance	Bathroom	Car	Landsize	BuildingArea	Price
count	9244.000000	9244.000000	9244.000000	9244.000000	9244.000000	9244.000000	9.244000e+03
mean	3.098118	11.241151	1.652423	1.695370	528.833838	156.994628	1.092329e+06
std	0.964029	6.882570	0.724991	0.975529	1212.965090	480.976260	6.796212e+05
min	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	1.310000e+05
25%	2.000000	6.400000	1.000000	1.000000	210.000000	100.000000	6.410000e+05
50%	3.000000	10.300000	2.000000	2.000000	474.000000	132.000000	9.000000e+05
75%	4.000000	13.900000	2.000000	2.000000	651.000000	181.000000	1.341250e+06
max	12.000000	48.100000	9.000000	10.000000	44500.000000	44515.000000	9.000000e+06

# machine learning

## Regressão Linear Múltipla

```
[175] 1 ## range de Price  
2  
3 df.Price.max() - df.Price.min()
```

↳ 8869000.0

# machine learning

## Regressão Linear Múltipla

```
[167] 1 ## verificando valores nulos  
2 df.isna().sum()
```

```
   □ Rooms          0  
     Distance       0  
     Bathroom       0  
     Car            0  
     Landsize       0  
     BuildingArea   0  
     Price          0  
     dtype: int64
```

# machine learning

## Regressão Linear Múltipla

```
[182] 1 ## verificando as correlações
2 plt.figure(figsize=(15,9))
3 sns.heatmap(df.corr(), annot=True, vmin=-1, vmax=1, cmap='coolwarm')
4 plt.show()
```



# machine learning

## Regressão Linear Múltipla

```
[183] 1 ## splitando o dataset
      2
      3 x = df.drop('Price',axis=1)
      4 y = df.Price
      5
      6 x_train, x_test, y_train, y_test = train_test_split(X,y,test_size=0.25,random_state=42)
      7
      8 print(x_train.shape,y_train.shape)
      9 print(x_test.shape,y_test.shape)
```

```
↳ (6933, 6) (6933,)
     (2311, 6) (2311,)
```

# machine learning

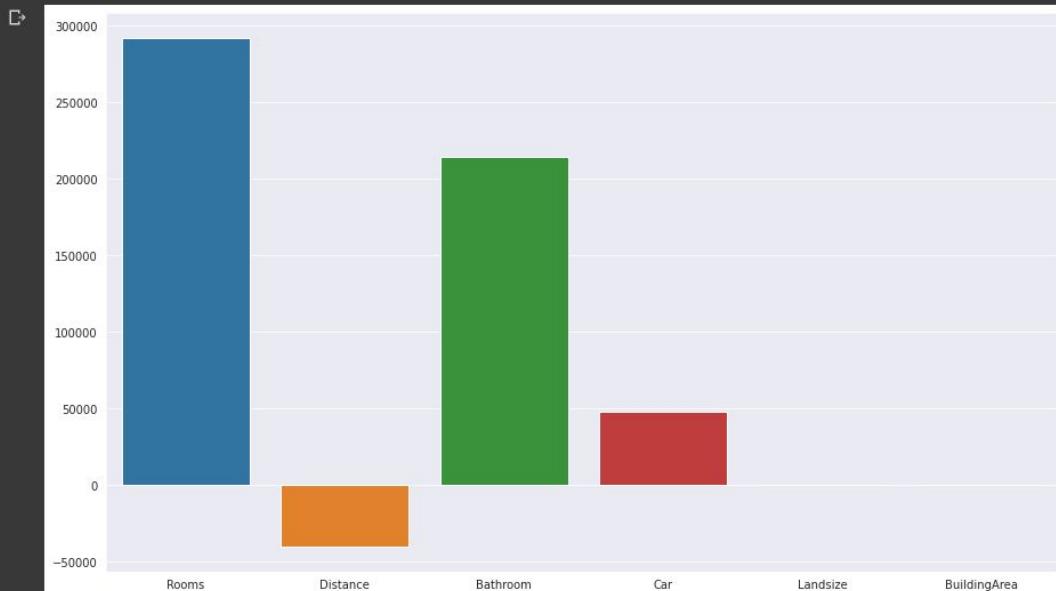
## Regressão Linear Múltipla

```
[170] 1 ## treinando um modelo de regressão linear multipla  
2  
3 lr = LinearRegression()  
4 lr.fit(x_train,y_train)  
5 lr.coef_  
  
[170] array([ 2.91937337e+05, -3.96883189e+04,  2.14182464e+05,  4.81893042e+04,  
[170]          1.75684873e+01,   3.29228029e+01])
```

# machine learning

## Regressão Linear Múltipla

```
[163] 1 ## verificando os pesos  
2  
3 plt.figure(figsize=(15,9))  
4 sns.barplot(x = x_real.columns,y = lr.coef_)  
5 plt.show()
```



# machine learning

## Regressão Linear Múltipla

```
[172] 1 ## MSE do modelo  
2  
3 y_pred = lr.predict(x_test)  
4  
5 mean_squared_error(y_test,y_pred)
```

```
↳ 280083847108.7673
```

```
[173] 1 ## RMSE do modelo  
2  
3 np.sqrt(mean_squared_error(y_test,y_pred))
```

```
↳ 529229.4843532125
```

# machine learning

## Regressão Linear Múltipla

```
[176] 1 ## r2 do modelo  
2  
3 lr.score(x_test,y_test)
```

```
⇒ 0.385704077883528
```

```
[177] 1 ## r2 do modelo com API  
2  
3 from sklearn.metrics import r2_score  
4  
5 r2_score(y_test,y_pred)
```

```
⇒ 0.385704077883528
```

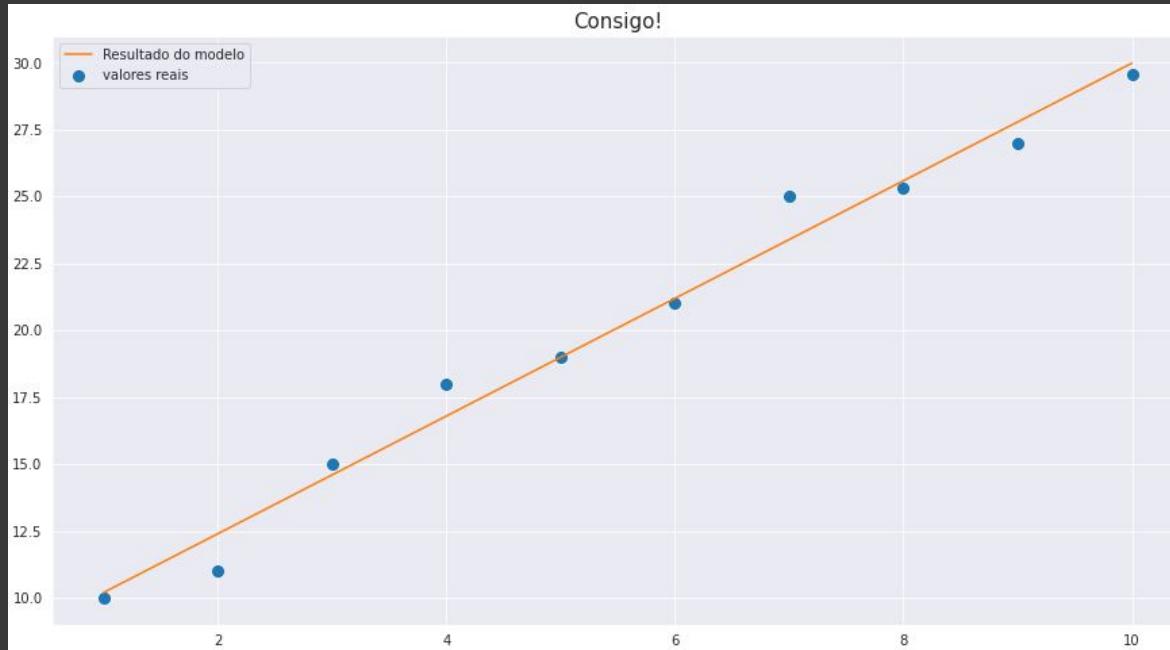
# machine learning

Regressão Linear Múltipla - r2



# machine learning

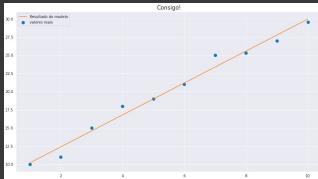
Regressão Linear Múltipla - r2



# machine learning

Regressão Linear Múltipla - r2

SS<sub>reg</sub>



$$r^2 = 1 - \left( \frac{SS_{reg}}{SS_{total}} \right)$$

SS<sub>total</sub>



# machine learning

Regressão Linear Múltipla - r2

```
[242] 1 ## definindo nosso_r2
      2
      3 def nosso_r2(y_true,y_pred):
      4
      5     med = y_true.mean()
      6
      7     ssreg = sum((y_true - y_pred)**2)
      8     sstotal = sum((y_true - med)**2)
      9
     10    return 1-(ssreg/sstotal)
```

# machine learning

Regressão Linear Múltipla - r2

```
[243] 1 ## conferindo o r2 do modelo  
2 nosso_r2(y,y_pred)
```

```
↳ 0.9824589902504507
```

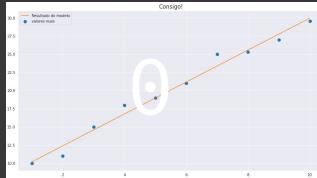
```
[244] 1 ## comparando resultado com a API do sklearn  
2 r2_score(y,y_pred)
```

```
↳ 0.9824589902504507
```

# machine learning

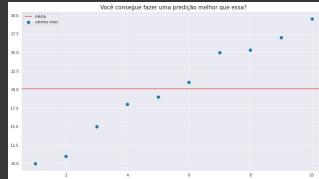
Regressão Linear Múltipla  $r^2$

$SS_{reg}$



$$r^2 = 1 - \left( \frac{SS_{reg}}{SS_{total}} \right) = ??$$

$SS_{total}$



# machine learning

Regressão Linear Múltipla - r2

$$r^2 = 1 - \left( \frac{100}{101} \right) = ??$$

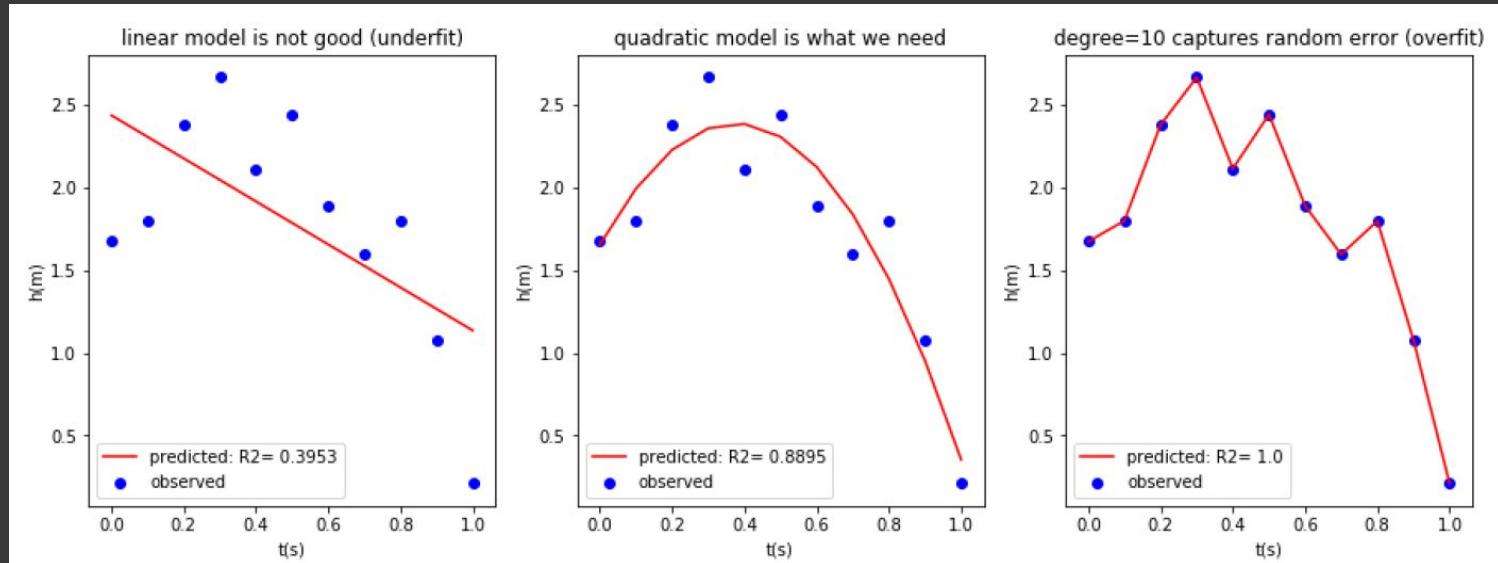
# machine learning

Regressão Linear Múltipla - r2

$$r^2 = 1 - \left( \frac{250}{110} \right) = ??$$

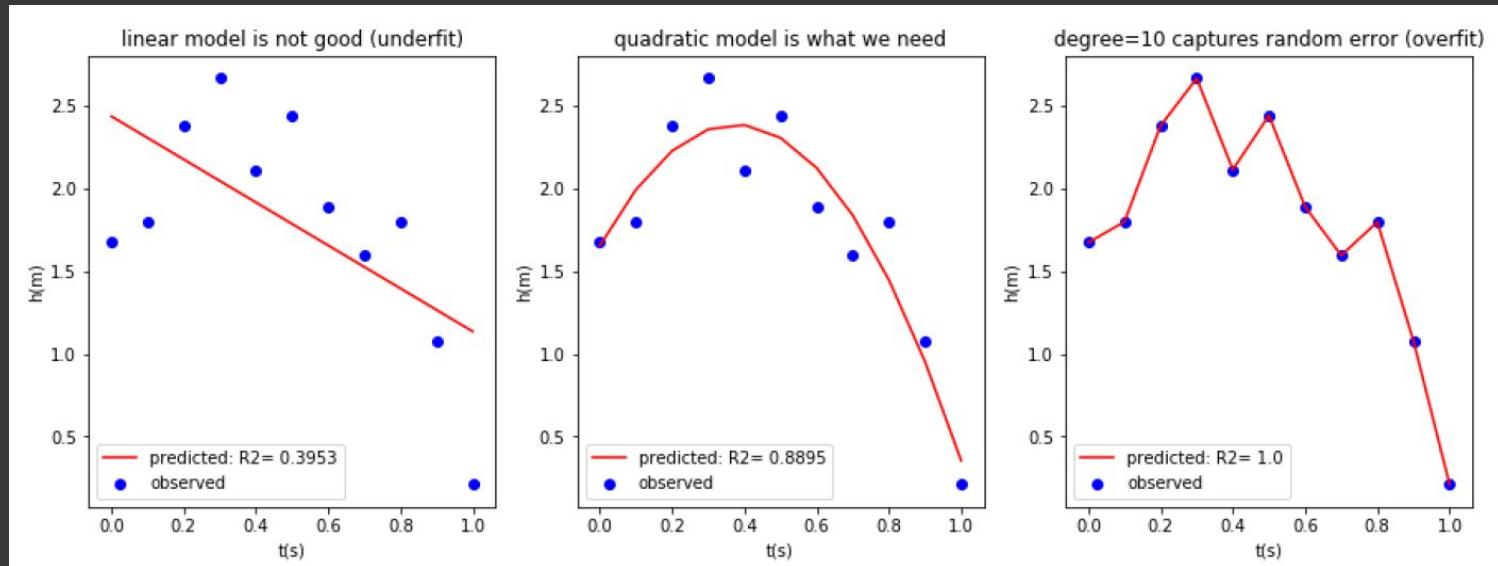
# machine learning

Regressão Linear Múltipla - r2 e under/overfitting



# machine learning

Regressão Linear Múltipla - r2 e under/overfitting



> Aumentar a  
complexidade do modelo

> Regularizar o modelo

# machine learning

Regressão Polinomial - Adicionando Complexidade

$$y(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 \dots$$

# machine learning

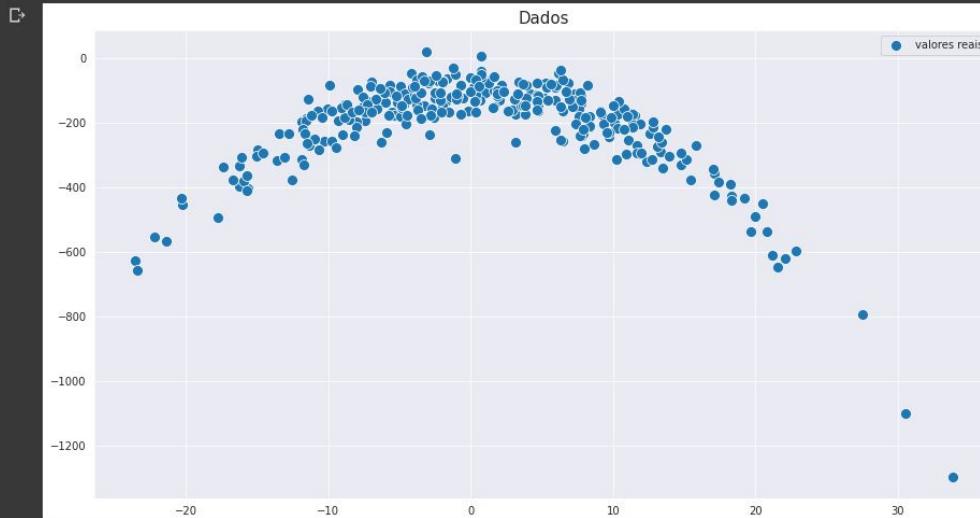
Regressão Polinomial - Adicionando Complexidade

```
[852] 1 ## definindo a função
2
3 def val_model(y_true,y_pred):
4     print('x-x-x PRINTING METRICS FOR THE MODEL x-x-x')
5     print('---'*30)
6     print(f'MEAN SQUARE ERROR      : {mean_squared_error(y_true,y_pred)}')
7     print(f'ROOT MEAN SQUARE ERROR : {np.sqrt(mean_squared_error(y_true,y_pred))}')
8     print(f'R2                  : {r2_score(y_true,y_pred)}')
```

# machine learning

## Regressão Polinomial - Adicionando Complexidade

```
[854] 1 ## plotando os dados
2 x=10*np.random.normal(0,1,300)
3 y=(-x**2)+np.random.normal(-100,50,300)
4 plt.figure(figsize=(15,8))
5 sns.scatterplot(x=x,y=y,s=100,label='valores reais')
6 plt.title('Dados',size=15)
7 plt.legend()
8 plt.show()
```



# machine learning

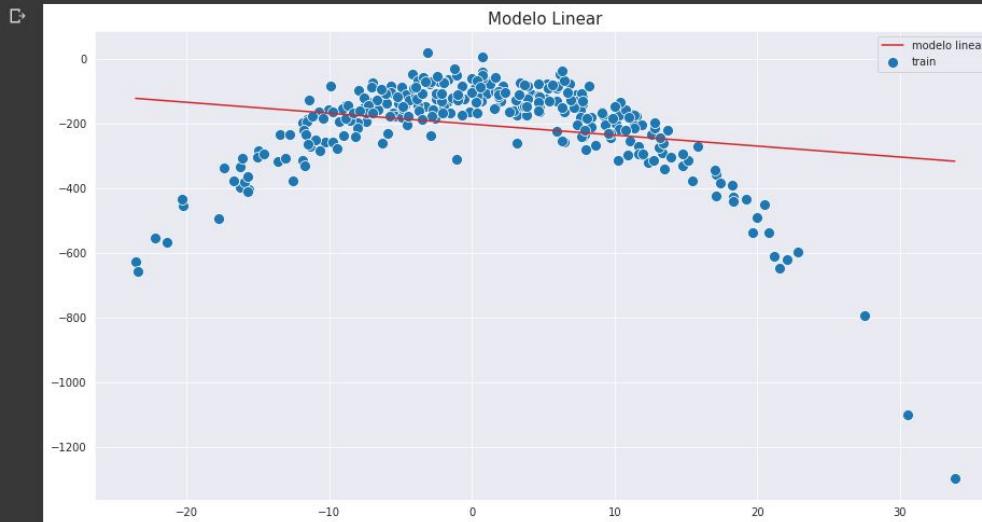
Regressão Polinomial - Adicionando Complexidade

```
[855] 1 ## treinando um modelo linear
      2 lr = LinearRegression()
      3 lr.fit(x.reshape(-1,1),y)
→ LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

# machine learning

## Regressão Polinomial - Adicionando Complexidade

```
[856] 1 ## plotando o modelo linear
2 plt.figure(figsize=(15,8))
3 sns.scatterplot(x=x,y=y,s=100,label='train',color='tab:blue')
4 sns.lineplot(x = x, y = lr.predict(x.reshape(-1,1)),label='modelo linear',color = 'tab:red')
5 plt.title('Modelo Linear',size=15)
6 plt.legend()
7 plt.show()
```



# machine learning

Regressão Polinomial - Adicionando Complexidade

```
[857] 1 ## avaliando o modelo  
2  
3 y_pred = lr.predict(x.reshape(-1,1))  
4  
5 val_model(y,y_pred)  
  
↳ x-x-x PRINTING METRICS FOR THE MODEL x-x-x  
-----  
MEAN SQUARE ERROR      : 21206.77861519947  
ROOT MEAN SQUARE ERROR : 145.62547378532187  
R2                      : 0.05122693633684061
```

# machine learning

Regressão Polinomial - Adicionando Complexidade

## sklearn.preprocessing.PolynomialFeatures

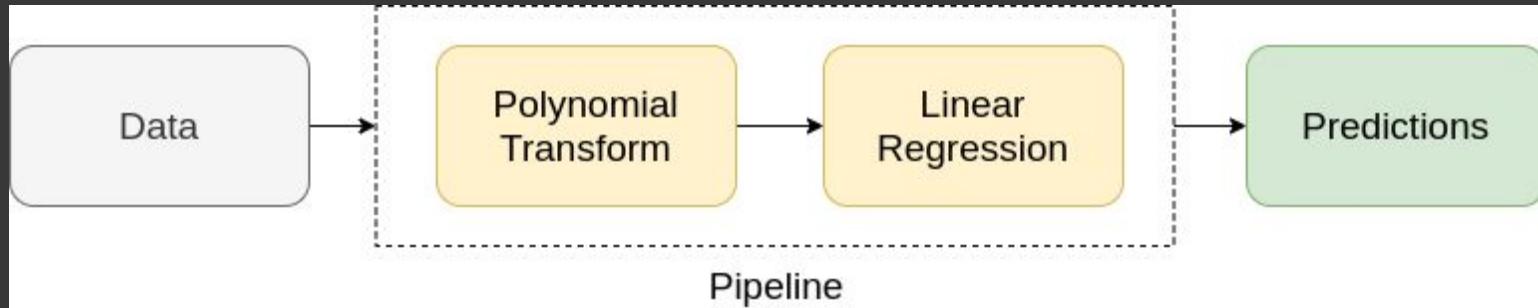
```
class sklearn.preprocessing.PolynomialFeatures(degree=2, interaction_only=False,  
include_bias=True, order='C')
```

[source]

Generate polynomial and interaction features.

# machine learning

Regressão Polinomial - Adicionando Complexidade



# machine learning

Regressão Polinomial - Adicionando Complexidade

```
[858] 1 ## criando as features polinominais  
2  
3 from sklearn.preprocessing import PolynomialFeatures  
4  
5 pol_transformer = PolynomialFeatures(degree=2)  
6 x_poly = pol_transformer.fit_transform(x.reshape(-1,1))
```

```
[859] 1 x_poly
```

```
↳ array([[ 1.00000000e+00, -4.71239143e+00,  2.22066330e+01],  
       [ 1.00000000e+00,  1.12680517e+01,  1.26968989e+02],
```

# machine learning

Regressão Polinomial - PolynomialFeatures(degree=2)

x = [2]

[1, x, x^2]

[1, 2, 4]

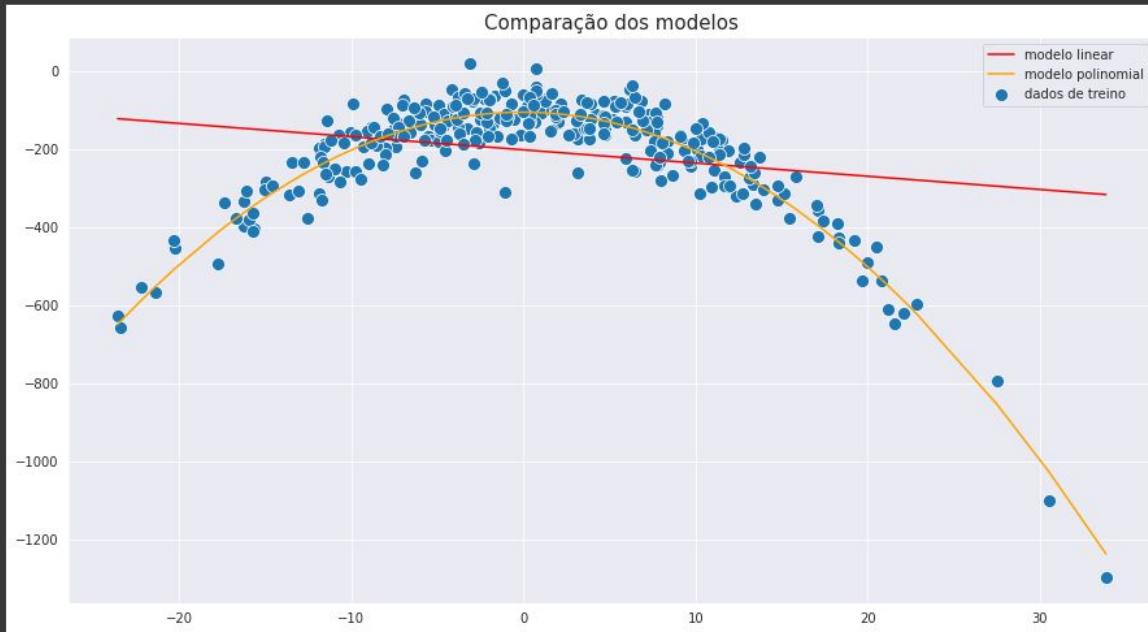
# machine learning

Regressão Polinomial - Adicionando Complexidade

```
[869] 1 ## treinando um modelo com features polinomiais  
2  
3 lr_poly = LinearRegression()  
4 lr_poly.fit(x_poly,y)  
  
↳ LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

# machine learning

## Regressão Polinomial - Adicionando Complexidade



# machine learning

## Regressão Polinomial - Adicionando Complexidade

```
[857] 1 ## avaliando o modelo  
2  
3 y_pred = lr.predict(x.reshape(-1,1))  
4  
5 val_model(y,y_pred)
```

```
↳ x-x-x PRINTING METRICS FOR THE MODEL x-x-x  
-----  
MEAN SQUARE ERROR      : 21206.77861519947  
ROOT MEAN SQUARE ERROR : 145.62547378532187  
R2                      : 0.05122693633684061
```

```
[870] 1 ## avaliando o modelo polinomial  
2  
3 y_pred = lr_poly.predict(x_poly)  
4  
5 val_model(y,y_pred)
```

```
↳ x-x-x PRINTING METRICS FOR THE MODEL x-x-x  
-----  
MEAN SQUARE ERROR      : 2480.232956665437  
ROOT MEAN SQUARE ERROR : 49.80193727823685  
R2                      : 0.8890365074492165
```

# machine learning

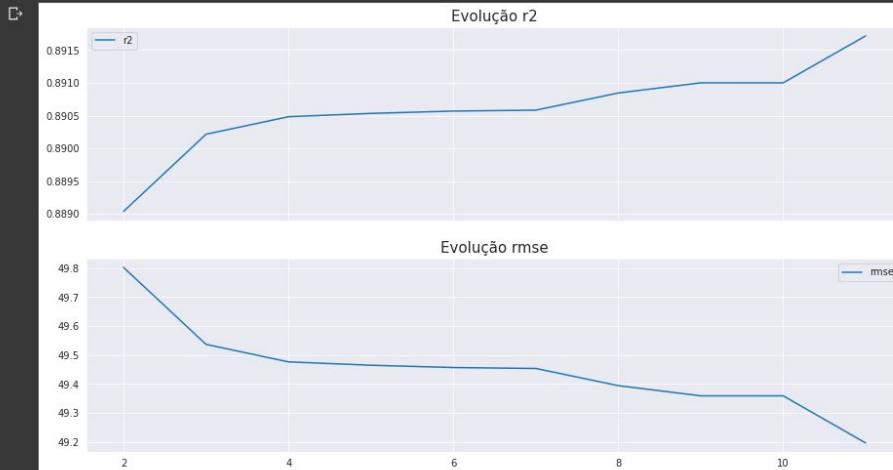
Regressão Polinomial - Adicionando Complexidade

```
[867] 1 ## polynomial regression + pipeline
2
3 from sklearn.pipeline import Pipeline
4
5 val_score = {}
6
7 for i in range(2,12):
8     steps = [('poly',PolynomialFeatures(degree=i)),('model',LinearRegression())]
9     pipe = Pipeline(steps)
10    pipe.fit(x.reshape(-1,1),y)
11    y_pred = pipe.predict(x.reshape(-1,1))
12    val_score[i] = [r2_score(y,y_pred),np.sqrt(mean_squared_error(y,y_pred))]
```

# machine learning

## Regressão Polinomial - Adicionando Complexidade

```
[872] 1 ## plotando
2
3 f, ax = plt.subplots(2,1,figsize=(15,8),sharex=True)
4
5 sns.lineplot(x = list(val_score.keys()), y = [val_score[key][0] for key in val_score.keys()],label='r2',ax=ax[0])
6 sns.lineplot(x = list(val_score.keys()), y = [val_score[key][1] for key in val_score.keys()],label='rmse',ax=ax[1])
7
8 ax[0].set_title('Evolução r2',size=15)
9 ax[1].set_title('Evolução rmse',size=15)
10
11 plt.legend()
12 plt.show()
```



# machine learning

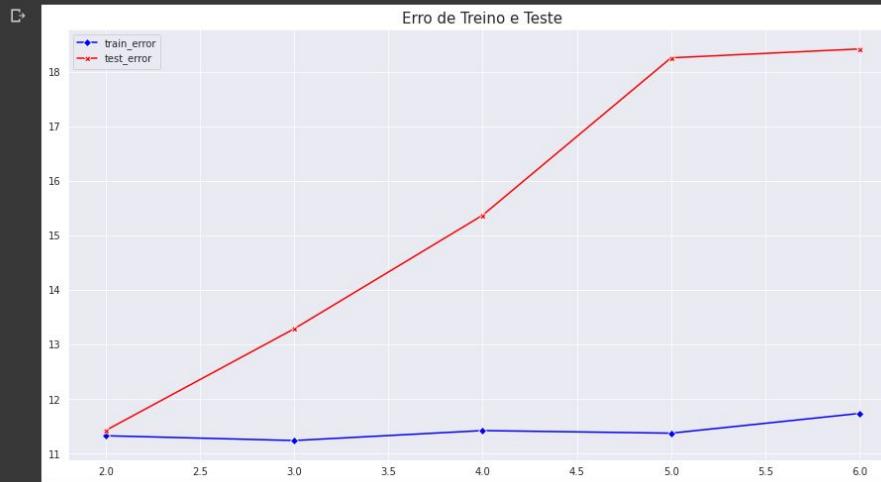
## Regressão Polinomial - Melhorando o modelo

```
[126] 1  train_error = []
2  test_error = []
3
4  for i in range(2,7):
5
6      ## transformando x
7      poly = PolynomialFeatures(i)
8      x_poly = poly.fit_transform(x)
9
10     ## splitando o dataset
11     x_train, x_test, y_train, y_test = train_test_split(x_poly,y,test_size=0.25,random_state=42)
12
13     ## treinando para I
14     regressor = LinearRegression()
15     regressor.fit(x_train,y_train)
16
17     ## avaliando o erro no test
18     y_pred = regressor.predict(x_test)
19     test_error.append(mean_squared_error(y_test,y_pred))
20
21     ## avaliando o erro no treino
22     train_error.append(mean_squared_error(y_train,regressor.predict(x_train)))
```

# machine learning

## Regressão Polinomial - Melhorando o modelo

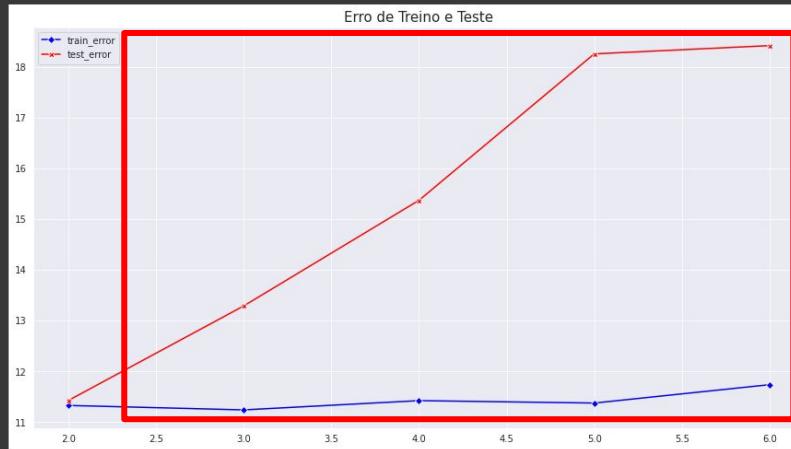
```
[127] 1  plt.figure(figsize=(15,8))
2
3  train_error = np.log10(train_error)
4  test_error = np.log10(test_error)
5
6  sns.lineplot(x=list(range(2,7)),y=train_error,label='train_error',color='b',marker='D')
7  sns.lineplot(x=list(range(2,7)),y=test_error,label='test_error',color='r',marker='X')
8
9  plt.title('Erro de Treino e Teste',size=15)
10 plt.legend()
11 plt.show()
```



# machine learning

Regressão Polinomial - Melhorando o modelo

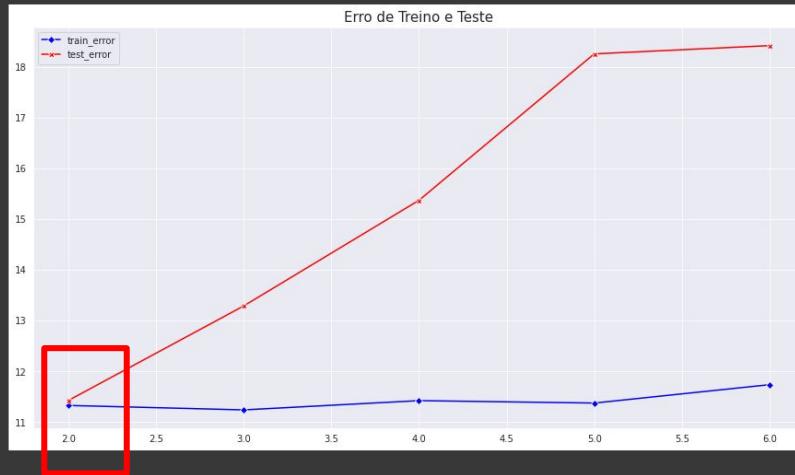
OVERFITTING = ERRO DE TREINO  $\downarrow\downarrow$   
ERRO DE TESTE  $\uparrow\uparrow$



# machine learning

Regressão Polinomial - Melhorando o modelo

GOOD FIT =      ERRO DE TREINO ⇕  
                          ERRO DE TESTE ⇕



# machine learning

## Regressão Polinomial - Melhorando o modelo

$$\text{UNDERFITTING} = \frac{\text{ERRO DE TREINO}}{\text{ERRO DE TESTE}}$$

```
[104] 1 ## RMSE de teste do modelo  
2  
3 np.sqrt(mean_squared_error(y_test,y_pred))
```

⌚ 529229.4843532125

```
[109] 1 ## avaliação no treino  
2  
3 y_pred_train = lr.predict(x_train)  
4 print(mean_squared_error(y_train,y_pred_train))  
5 print(np.sqrt(mean_squared_error(y_train,y_pred_train)))  
6 print(lr.score(x_train,y_train))
```

⌚ 270073031034.8642  
519685.511665338  
0.4176840683710384

# machine learning

KFold Cross Validation



# machine learning

```
KFold = 4  
test_size = 0.25
```

shape(100,6)



# machine learning

Validação Cruzada - cross\_val\_score

## sklearn.model\_selection.cross\_val\_score

```
sklearn.model_selection.cross_val_score(estimator, X, y=None, groups=None, scoring=None,  
cv=None, n_jobs=None, verbose=0, fit_params=None, pre_dispatch='2*n_jobs', error_score=nan)
```

[source]

Evaluate a score by cross-validation

O cross\_val\_score sempre retorna um array com K 'scores' de teste.

# machine learning

Validação Cruzada - cross\_val\_score

```
[239] 1 from sklearn.model_selection import cross_val_score  
  
[240] 1 ## lendo o dataset  
2 df = pd.read_csv('/content/regression_dataset01_melbourne.csv',sep=';')  
3  
4 ## separando x e y  
5 x = df.drop(['Price'],axis=1)  
6 y = df.Price  
  
[241] 1 ## transformando as features  
2  
3 poly = PolynomialFeatures(2)  
4 poly_x = poly.fit_transform(x)
```

# machine learning

Validação Cruzada - cross\_val\_score

```
[242] 1 ## usando o cross_val_score
2
3 modelo = LinearRegression()
4
5 (cross_val_score(modelo,poly_x,y,cv=10,scoring='neg_root_mean_squared_error')*-1)

[5]: array([ 619200.10656456,   682525.52206306,   445777.64908942,
       475269.26617279,   457163.61645115,   464021.39558731,
       474859.59521624,  52034423.72453149,   699837.18128355,
      480217.26898749])
```

Root Mean Squared Error é uma métricas de LOSS (mede o erro) e por isso a implementação em termos de model\_selection é negativa, pois queremos minimizar esse valor.



# machine learning

Validação Cruzada - cross\_val\_score

```
[251] 1 ## testando para vários 'i' em Polynomial features
2
3 data = []
4
5 for i in range(2,7):
6     poly = PolynomialFeatures(i)
7     poly_x = poly.fit_transform(x)
8
9     modelo = LinearRegression()
10    data.append((cross_val_score(modelo,poly_x,y,cv=10,scoring='neg_root_mean_squared_error')*-1).mean())

[253] 1 np.log10(data)
```

```
array([ 6.75460284,  8.46559229, 11.19852639, 13.52732512, 15.34056271])
```

O aumento no erro nunca é considerado bom, por isso podemos utilizar cross\_val\_score para validar rapidamente uma 'hyperparameter' do modelo

# machine learning

## Validação Cruzada - cross\_val\_score

```
[288] 1 ## usando cross_val_score para comparar modelos
2
3 from sklearn.tree import DecisionTreeRegressor
4 from sklearn.dummy import DummyRegressor
5
6 def train_models(x,y,cv):
7
8     error = pd.DataFrame([])
9
10    ## val dummy
11    dummy = DummyRegressor('mean')
12    error['dummy'] = (cross_val_score(dummy,x,y,cv=cv,scoring='neg_root_mean_squared_error')*-1)
13
14    ## val LinearModel
15    linear = LinearRegression()
16    error['linear'] = (cross_val_score(linear,x,y,cv=cv,scoring='neg_root_mean_squared_error')*-1)
17
18    ## val DecisionTree
19    dt_reg = DecisionTreeRegressor(max_depth=5,max_features=4,min_samples_leaf=5)
20    error['tree'] = (cross_val_score(dt_reg,x,y,cv=cv,scoring='neg_root_mean_squared_error')*-1)
21
22    error = error.transpose()
23    error = error.assign(mean = lambda x: x.mean(axis=1))
24
25 return error
```

# machine learning

Validação Cruzada - cross\_val\_score

```
[290] 1 ## usando a função acima com cv=10
2
3 train_models(x,y,10)
```

	0	1	2	3	4	5	6	7	8	9	mean
dummy	784306.612226	698247.389687	661360.050102	676198.254000	653301.300146	650153.677900	629790.335595	6.538235e+05	698214.415453	694662.313112	680005.787171
linear	564942.533584	487885.628974	496423.165099	518130.647738	508954.320359	506795.844679	528893.603905	1.016350e+06	566997.409849	536114.437184	573148.766877
tree	558397.084608	462082.170637	491382.942441	509405.981804	459377.592734	450613.539974	461087.766507	4.983464e+05	446654.535118	475631.871070	481297.984320

# machine learning

Validação Cruzada - cross\_val\_score

Regression	
'explained_variance'	metrics.explained_variance_score
'max_error'	metrics.max_error
'neg_mean_absolute_error'	metrics.mean_absolute_error
'neg_mean_squared_error'	metrics.mean_squared_error
'neg_root_mean_squared_error'	metrics.mean_squared_error
'neg_mean_squared_log_error'	metrics.mean_squared_log_error
'neg_median_absolute_error'	metrics.median_absolute_error
'r2'	metrics.r2_score
'neg_mean_poisson_deviance'	metrics.mean_poisson_deviance
'neg_mean_gamma_deviance'	metrics.mean_gamma_deviance