Ferramentas de Softwares para uso em ciência de Dados II

OBJETIVO

Nasser Santiago Boan

> arquivologia - UNB

> tecnologia do Petróleo e Gás - UFRJ

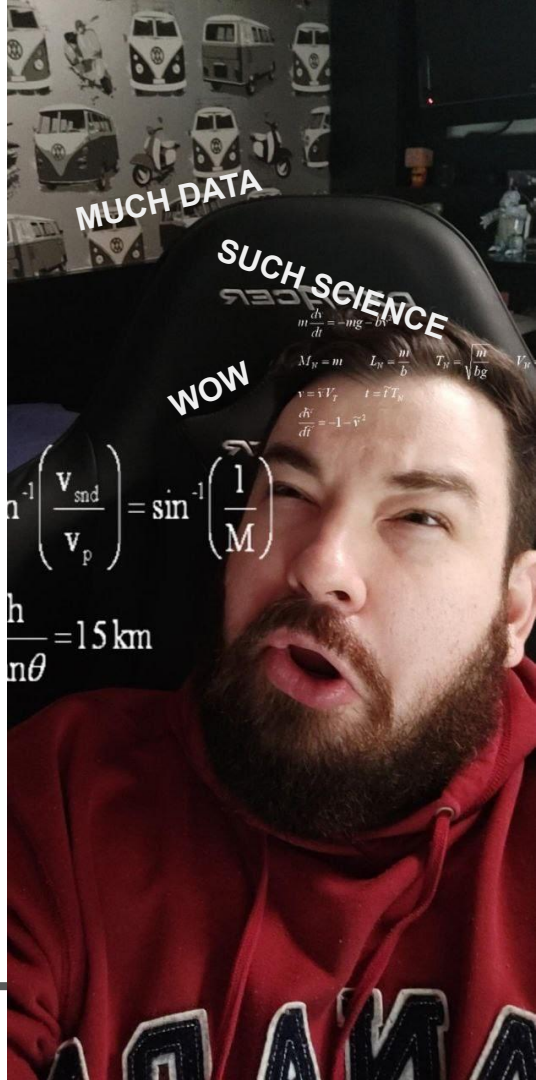
> MBA Processos de Negócio - IBMEC

> 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)

IESB

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

IESB

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)

IESB

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)

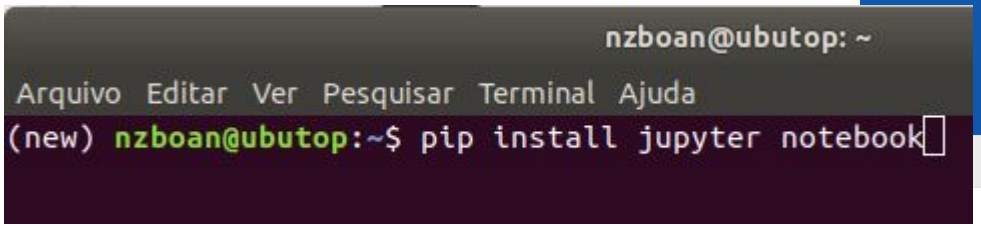
IESB

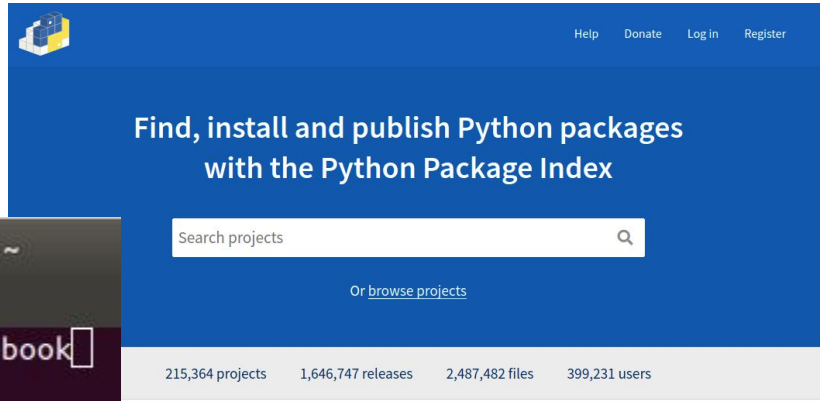
Módulo Básico

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 gerência ambientes 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>
```

IESB

conda e miniconda

O conda vai além e gerência ambientes 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
asn1crypto                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    conda-forge
bzip2                     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
```

IESB

conda e miniconda

O conda vai além e gerência ambientes e dependências

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


IESB

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         125 KB
    certifi-2019.11.28         |           py38_0         153 KB
    ld_impl_linux-64-2.33.1    |       h53a641e_7         568 KB
    mkl-service-2.3.0          |   py38he904b0f_0          62 KB
    mkl_fft-1.0.15             |   py38ha843d7b_0         159 KB
    mkl_random-1.1.0           |   py38h962f231_0         340 KB
    numpy-1.18.1               |   py38h4f9e942_0           5 KB
    numpy-base-1.18.1          |   py38hde5b4d6_1         4.2 MB
    openssl-1.1.1d             |       h7b6447c_3         2.5 MB
    pip-20.0.2                 |           py38_1         1.7 MB
    python-3.8.1               |       h0371630_1        49.5 MB
    scikit-learn-0.22.1        |   py38hd81dba3_0         5.4 MB
    scipy-1.3.2                |   py38h7c811a0_0        14.0 MB
    setuptools-45.1.0          |           py38_0         522 KB
    six-1.14.0                 |           py38_0          27 KB
    sqlite-3.30.1              |       h7b6447c_0         1.1 MB
    wheel-0.34.1               |           py38_0          50 KB
    ------------------------------------------------------------
                                           Total:        80.3 MB
```

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

In [9]: data = load_iris()

In [10]: data.target

Out[10]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```


IESB

conda e miniconda

environment.yml

```yaml
name: stats
dependencies:
  - jupyterlab
  - pandas
  - numpy
  - scikit-learn
```

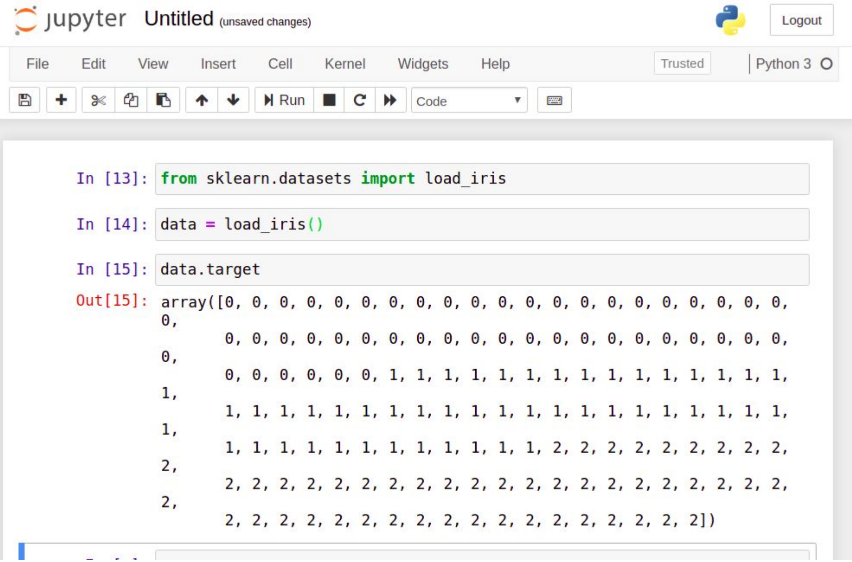
```
(base) nzboan@ubutop:~$ conda env create -f environment.yml
```

```
(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:~$
```

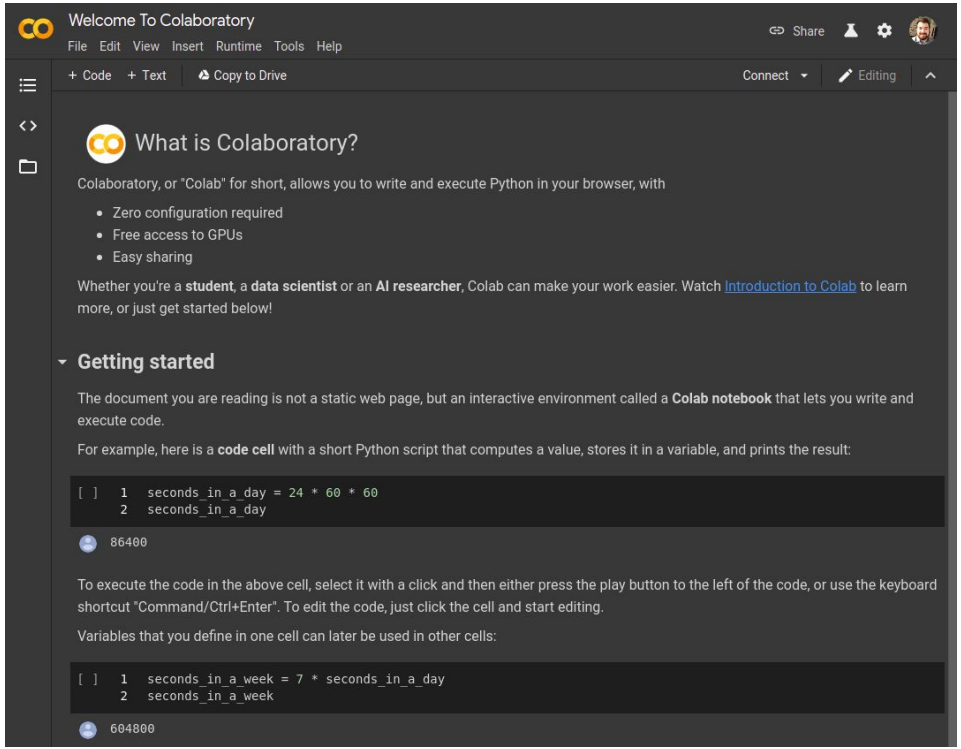

IESB

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



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/

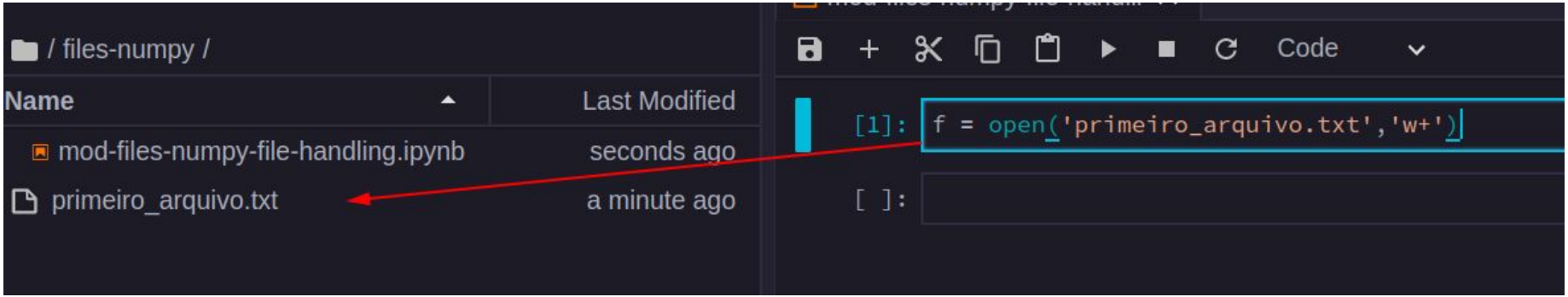
Nivelamento (again...)

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 as grande maioria das necessidades.



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.
```


IESB

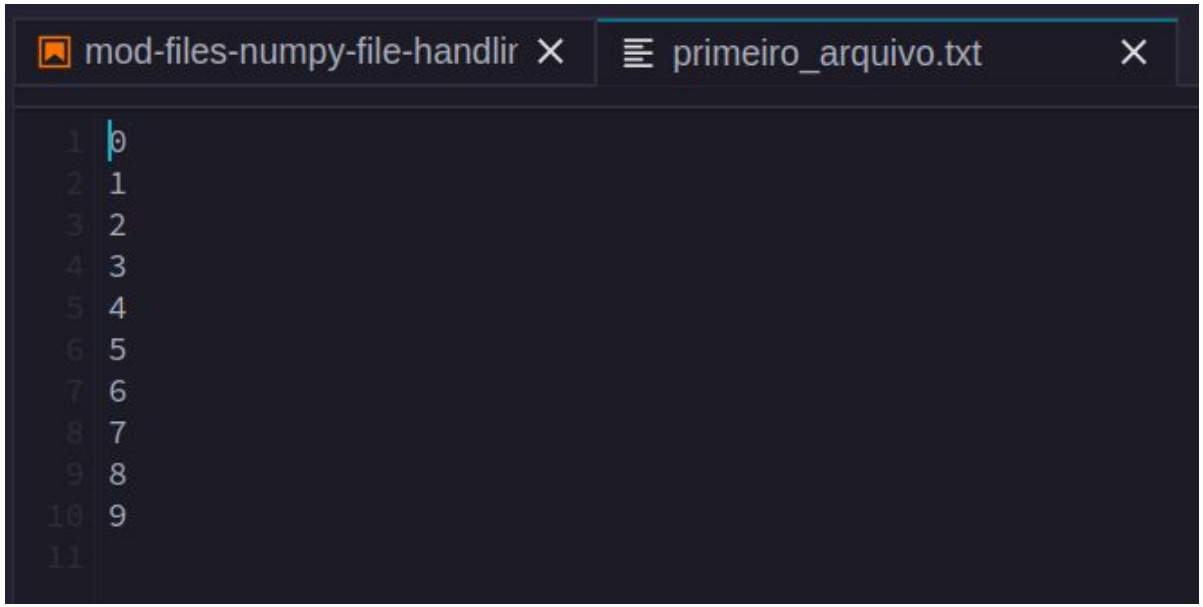
readlines()

```
[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
```


IESB

write()

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



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', 'regio
n']
['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']
```


IESB

csv.reader()

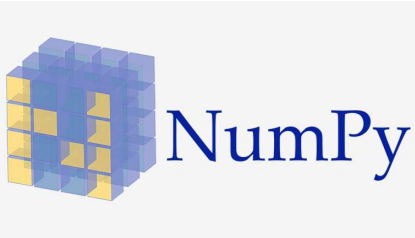
```
[44]: avg_price = []


      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])
```

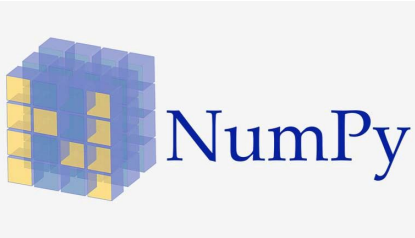
Delimiter: ,

	nome	idade	salario
1	nasser	29	1000
2	joao	20	1500



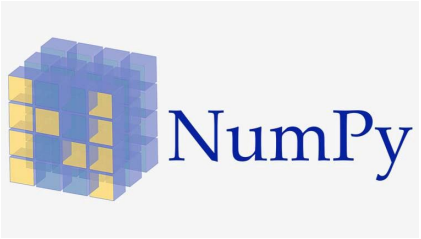

```
[2]: import numpy as np

     a = np.array([[1,2,2],[2,2,2]])
     a

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


IESB



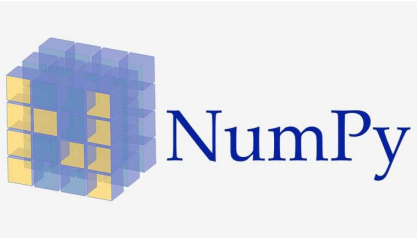

```
[3]: a.ndim

[3]: 2
```

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

[31]: 4
```




```
[38]: a.shape

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

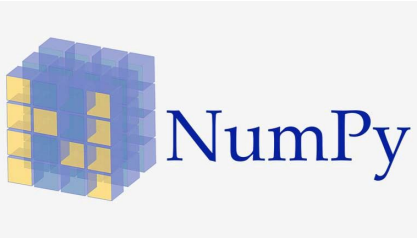
```
[13]: b = np.array([[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15]])
      b

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

[39]: b.shape

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




```
[43]: a.size
[43]: 6
```

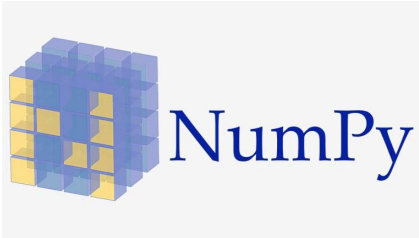
```
[44]: b.size
[44]: 15
```

```
[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]])
```

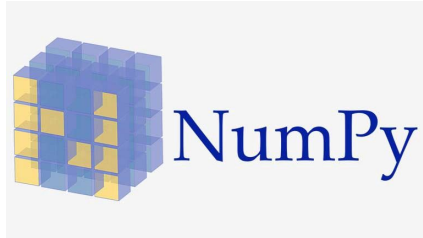
operações




```
[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



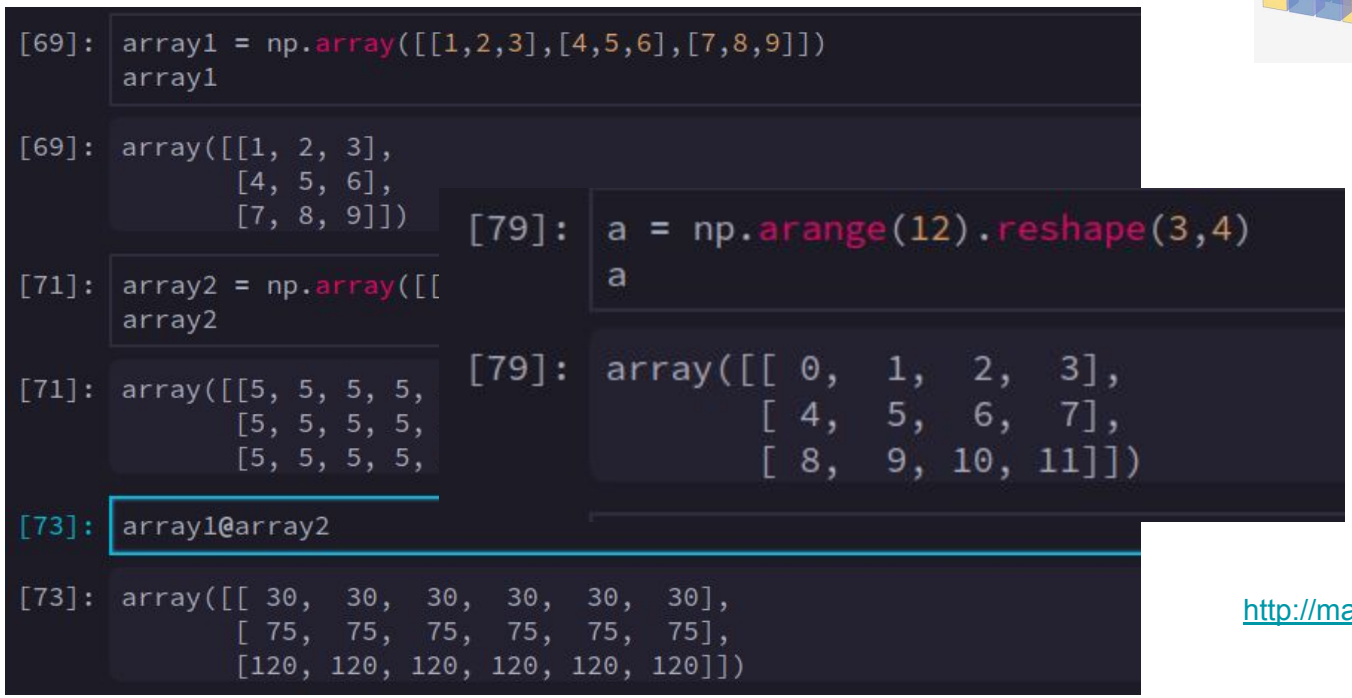

```
[57]: c = np.array([[2,1,1],[1,1,1]])
      c

[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]])
```

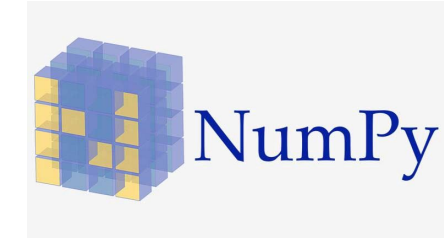

IESB

operações



http://matrixmultiplication.xyz/

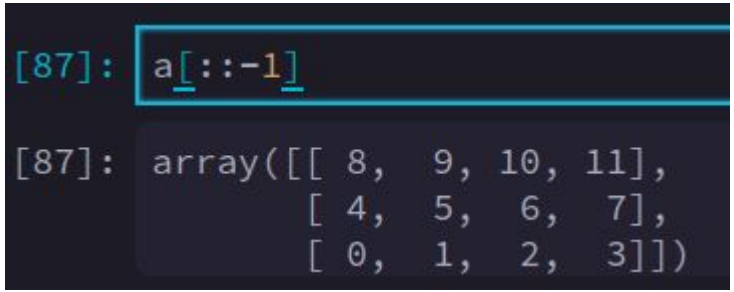
indexing & slicing

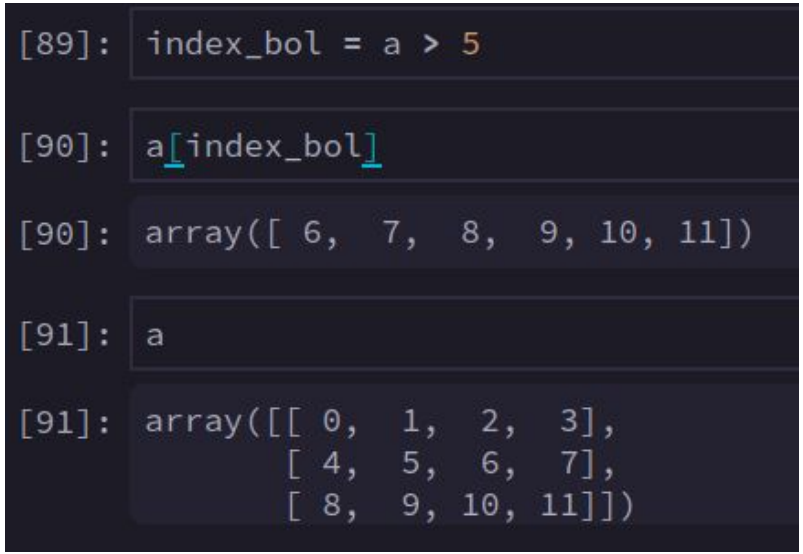



```
[79]: a = np.arange(12).reshape(3,4)
      a

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

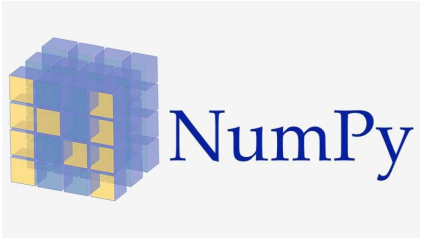
```
[82]: a[1][2]

[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





stacking

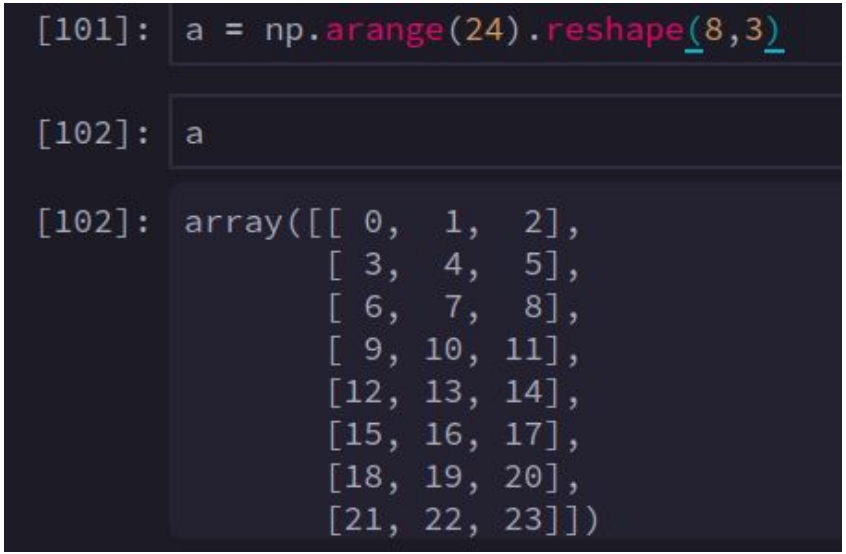



```
a

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

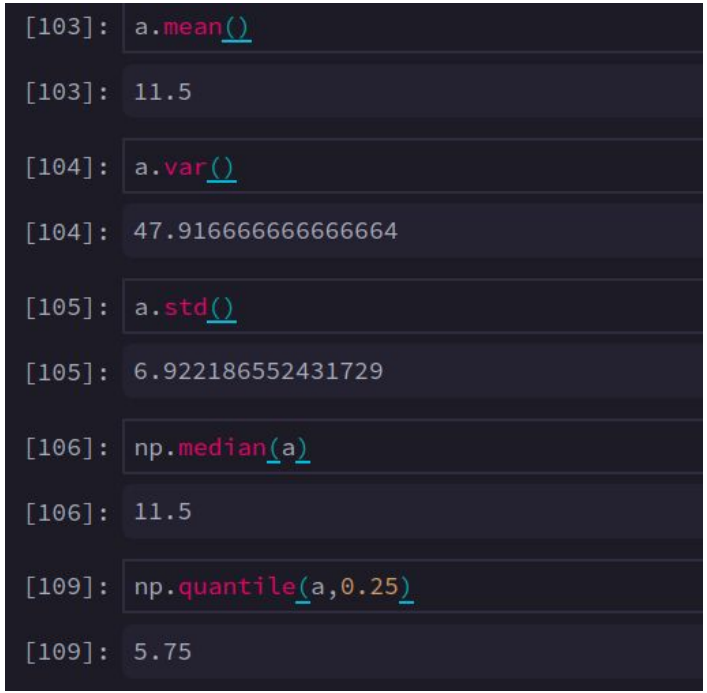
```
[96]: b

[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



funções estatísticas



IESB

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

IESB

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
```

[20]:

	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]

IESB

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, chunksize=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]
```


IESB

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

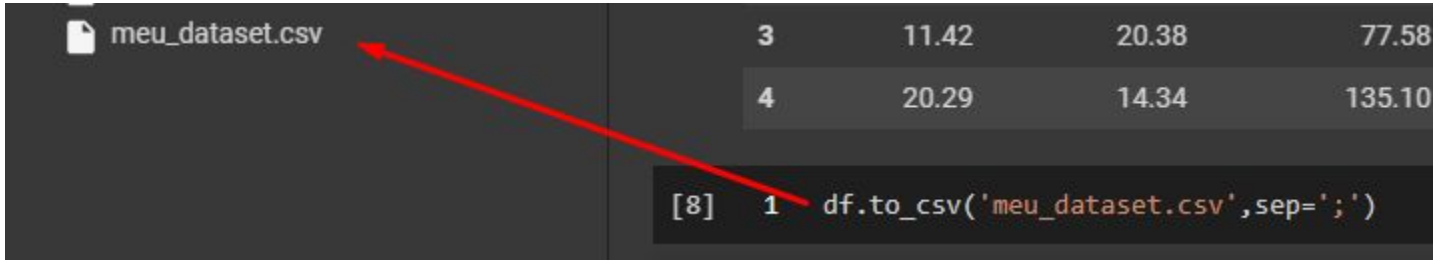
IESB

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()


meu_dataset.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
```


IESB

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()
```

	0 mean_radius	1 mean_texture	2 mean_perimeter	3 mean_area	4 mean_smoothness	5 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>]]

```
[37]: df = pd.read_csv('data/Breast_cancer_data.csv')
      df[['mean_radius','mean_area','diagnosis']]
```

[37]:

	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)]
```

[40]:

	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

IESB

indexing/slicing

condicionais

Operador	Exemplo
& (AND)	df[(df[‘coluna1’] == 0) & (df[‘coluna2’] >= 10)]
| (OR)	df[(df[‘coluna2’] <= 5) | (df[‘coluna2’] >= 10)]

indexing/slicing

.head()

.tail()

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

[41]:

	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

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

[42]:

	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

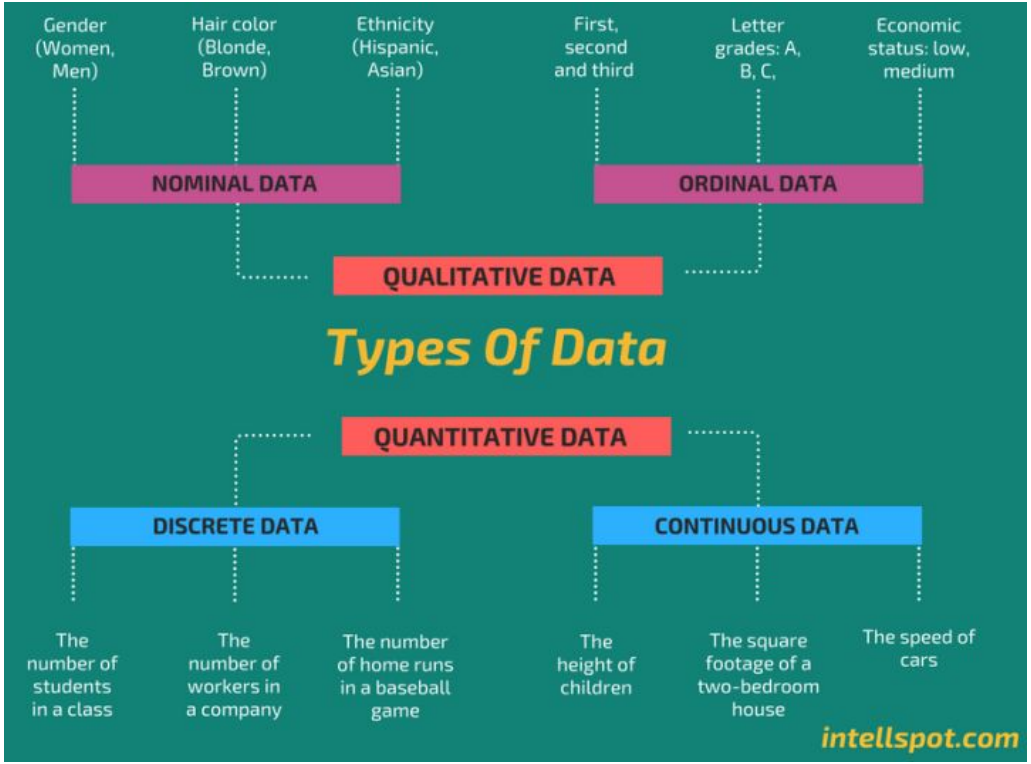
.sample()

```
[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()

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

[46]:

	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_dat

```
[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]
```


IESB

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]
```


IESB

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%		.170000	75.170000	420.300000	0.086370	0.000000
50%			86.240000	551.100000	0.095870	1.000000
75%		00	104.100000	782.700000	0.105300	1.000000
max	28.110000	39.280000	188.500000	2501.000000	0.163400	1.000000

Primeiro Quartil

Segundo Quartil ou Mediana

Terceiro Quartil

IESB

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

IESB

Funções Estatísticas

df.skew()

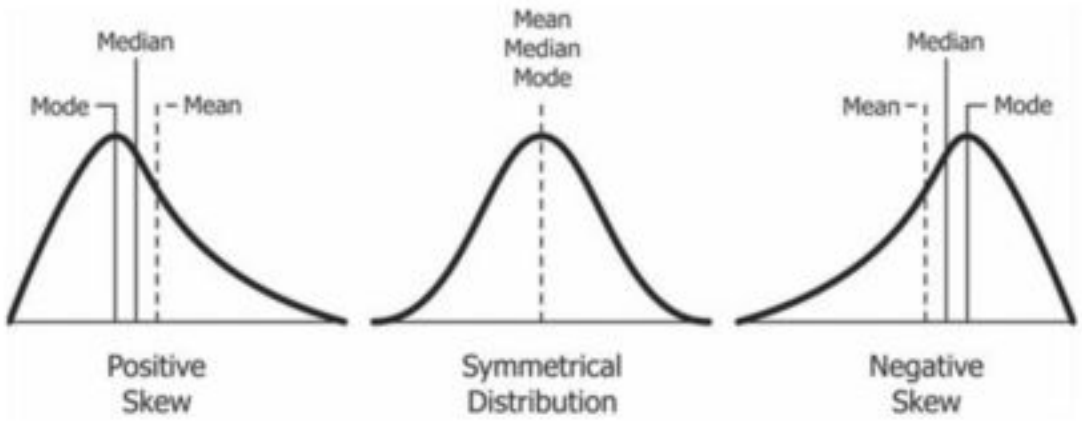
pandas.DataFrame.skew

`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.

IESB

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...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

IESB

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()
```

```
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

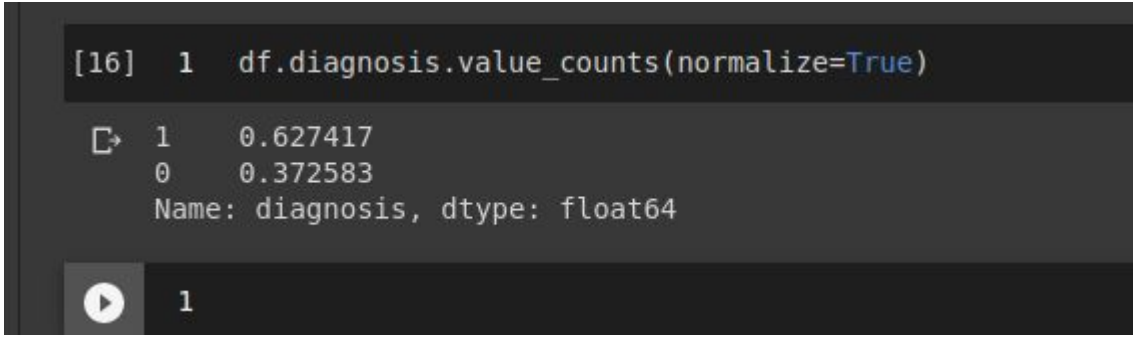
.value_counts()

```
[15]  1  df.diagnosis.value_counts()

      1    357
      0    212
      Name: diagnosis, dtype: int64
```


IESB

Sorting

.value_counts()



IESB

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

IESB

Operações com séries

concatenando strings

```
[17]  1  df = pd.DataFrame({'primeiro_nome':['nasser','maria','joão'],'sobrenome':['boan','silva','carvalho']})
      2  df
```

	primeiro_nome	sobrenome
0	nasser	boan
1	maria	silva
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

IESB

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
```


IESB

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
```

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

```
[25]  1  df.transpose()
```

	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	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()

```
[11]  1  df.nlargest(n=10,columns='mean_area')
```

	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

IESB

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()
```

	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

```
[9]  1  df.sort_values('mean_texture',ascending=True)
```

	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

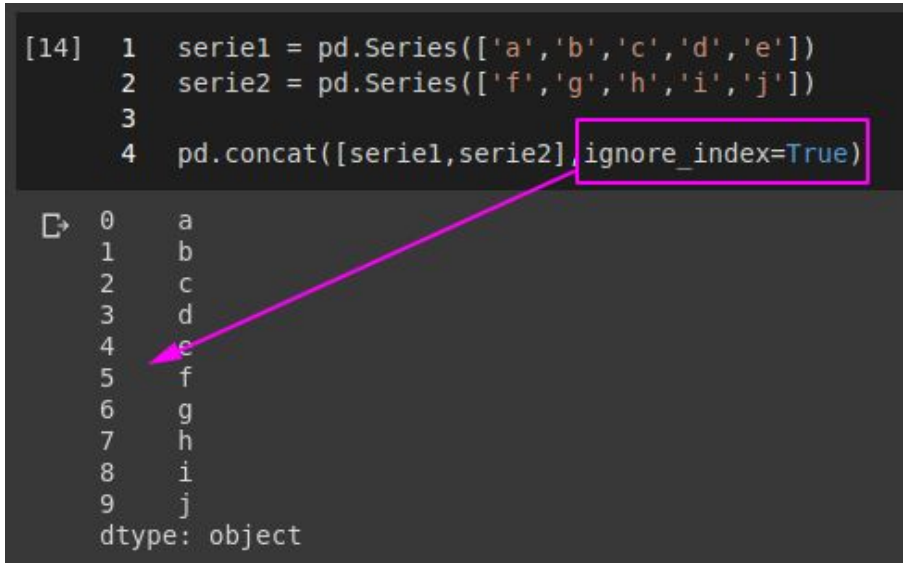
pd.concat()

```
[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()



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

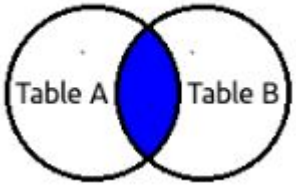
df.merge()

```
[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

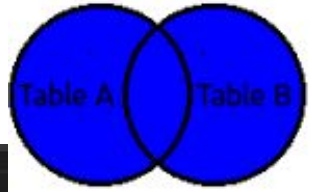
df.merge()

```
[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

IESB

Merge

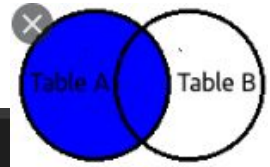
df.merge()

```
[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
      2
      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

IESB

Merge

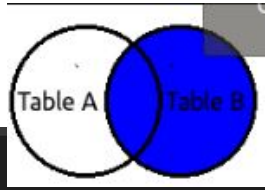
df.merge()

```
[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

IESB

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	Média
01/11/2018	00:05	DF-051 (EPGU) KM 03,4 SENTIDO GUARA II AO ZOOL...	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	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           100
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.

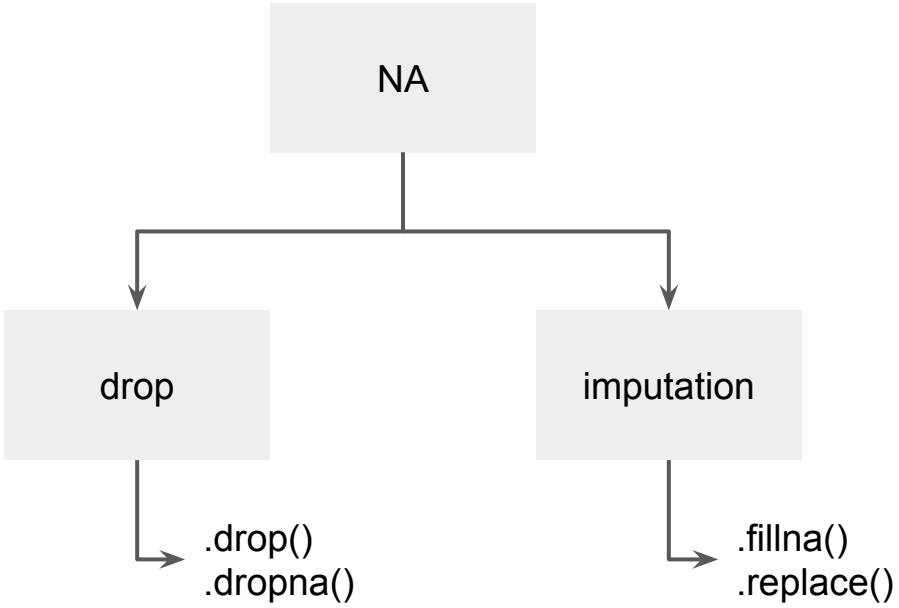
Return a boolean same-sized object indicating if the values are NA. NA values, such as None or `numpy.NaN`, gets mapped to True values. Everything else gets mapped to False values. Characters such as empty strings `''` or `numpy.inf` are not considered NA values (unless you set `pandas.options.mode.use_inf_as_na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

Returns: **DataFrame**

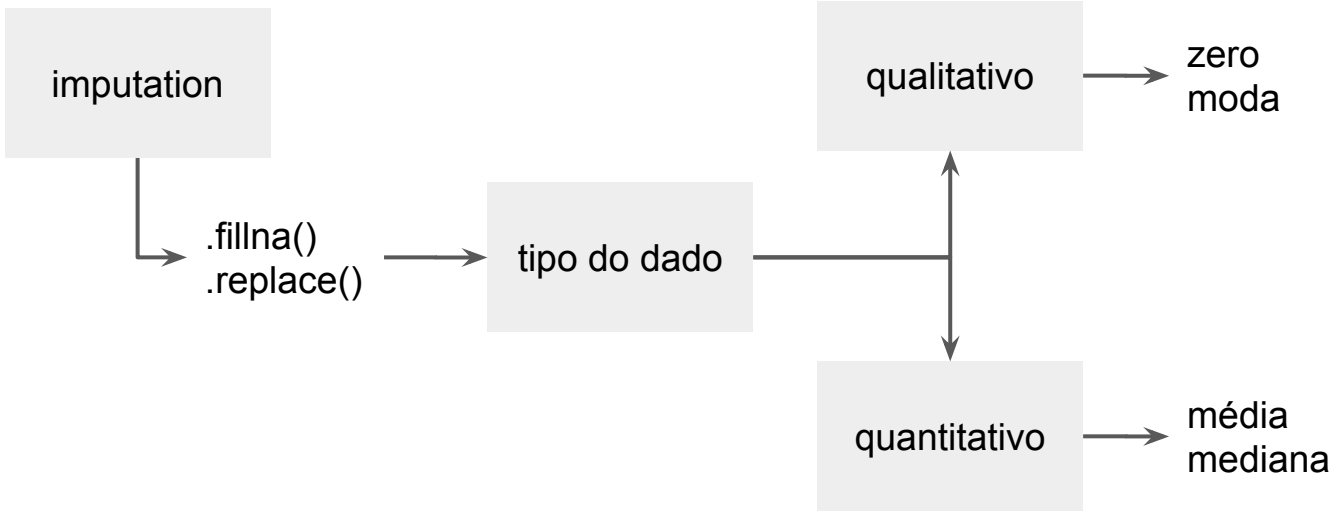
Mask of bool values for each element in DataFrame that indicates whether an element is not an NA value.

IESB

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.

IESB

Valores Nulos

.fillna()

```
1  df[['rooms','bathroom']].isna().sum()
```

```
rooms       11
bathroom    11
dtype: int64
```


IESB

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)
```

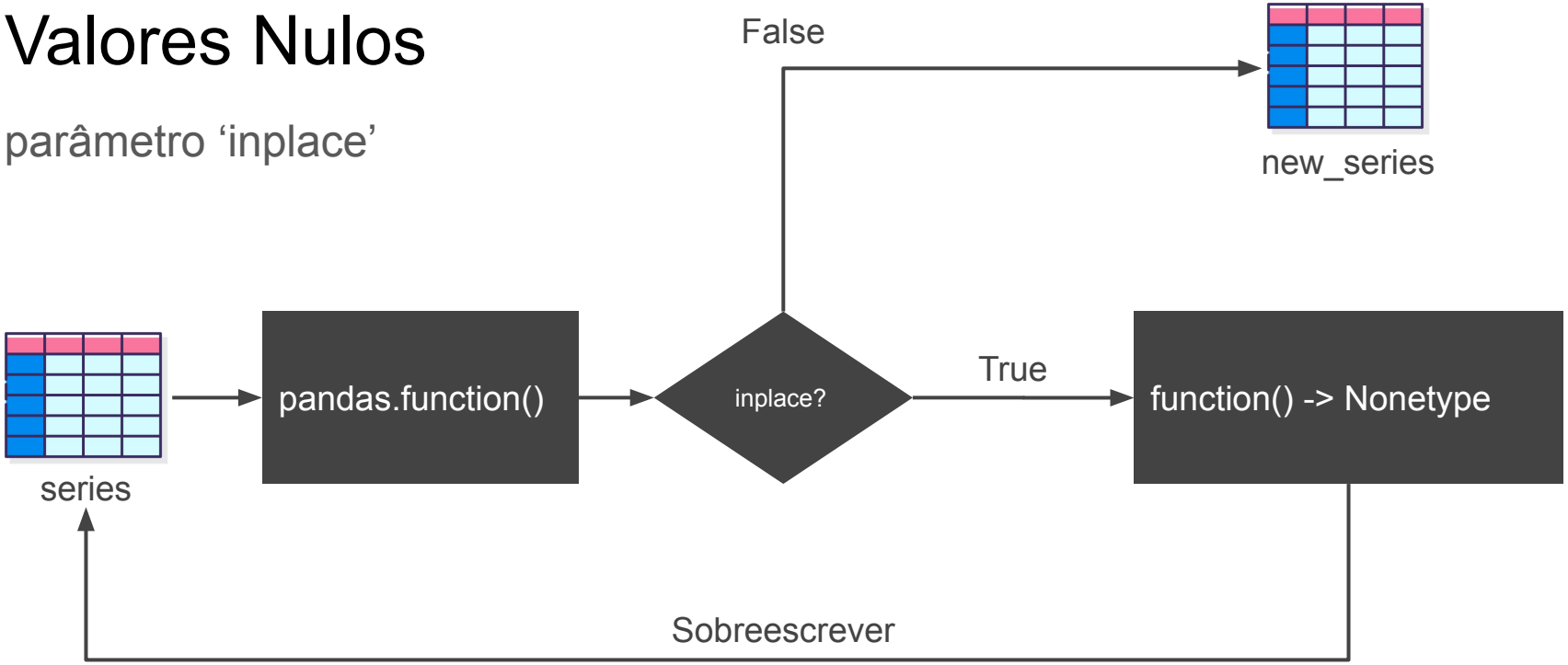
```
[12]  1  ## verificando se as colunas ainda possuem valores nulos
      2  df[['rooms','bathroom']].isna().sum()
```

```
rooms       0
bathroom    0
dtype: int64
```


IESB

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.

IESB

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.

IESB

VAMOS PRATICAR!!

IESB

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.

IESB

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.

IESB

Bônus ...

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

Bônus ...

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

GarageQual	SalePrice
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

IESB

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
```

```
(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

IESB

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'})
```

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

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))
```

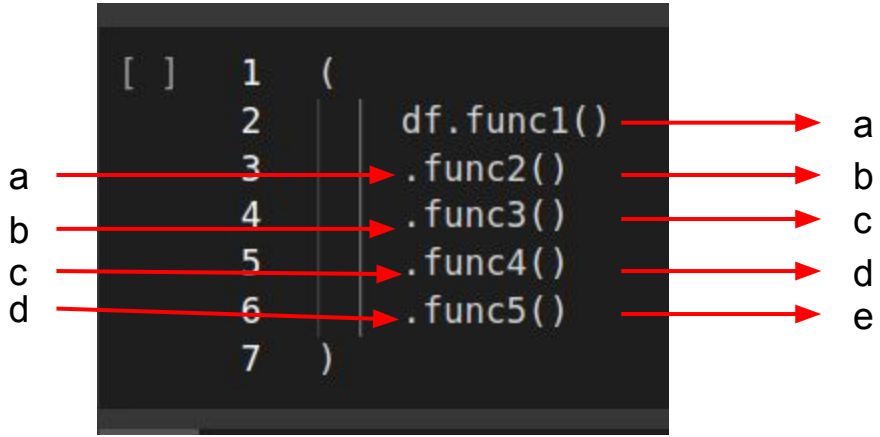
YrSold	valor_total_vendido	média_de_square_feet	valor_total_em_reais	média_em_m2
2006	57320530	10489.458599	292334703.0	974.494481
2007	61214777	10863.686930	312195362.7	1009.261142
2008	53917695	10587.687500	274980244.5	983.620169
2009	60648051	10294.248521	309305060.1	956.359023
2010	31043893	10220.645714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

IESB

METHOD CHAINING!!!

IESB

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

IESB

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>
```


IESB

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  )
```

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

IESB

METHOD CHAINING!!!

```
(
    pd.read_csv('/content/train.csv')
    .loc[:,['YrSold','SalePrice','LotArea']]
    .groupby('YrSold')
    .agg({'SalePrice':'sum','LotArea':'mean'})
    .rename(columns={'SalePrice':'valor_total_vendido','LotArea':'média_de_square_
)
```

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

IESB

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  )
```

YrSold	valor_total_em_reais	média_em_m2
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

IESB

METHOD CHAINING!!!


```
[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              média_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  )
```


IESB

METHOD CHAINING!!!

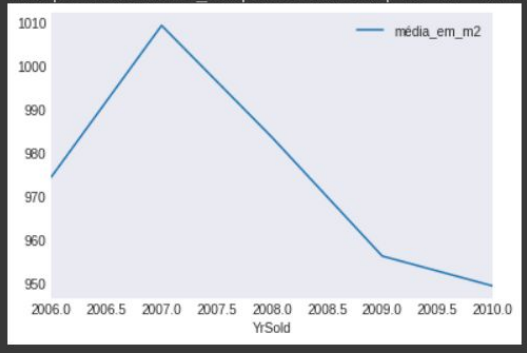
```
[71]  1
      2  def criar_relatorio(path_arquivo):
      3
      4    (
      5        pd.read_csv(path_arquivo)
      6        .loc[:,['YrSold','SalePrice','LotArea']]
      7        .groupby('YrSold')
      8        .agg({'SalePrice':'sum','LotArea':'mean'})
      9        .rename(columns={'SalePrice':'valor_total_vendi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

[72]  1  criar_relatorio('/content/train.csv')
```

IESB

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_squ
      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','valor_total_em_reais
     10      .plot()
     11  )
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbc6ea65c88>
```



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.

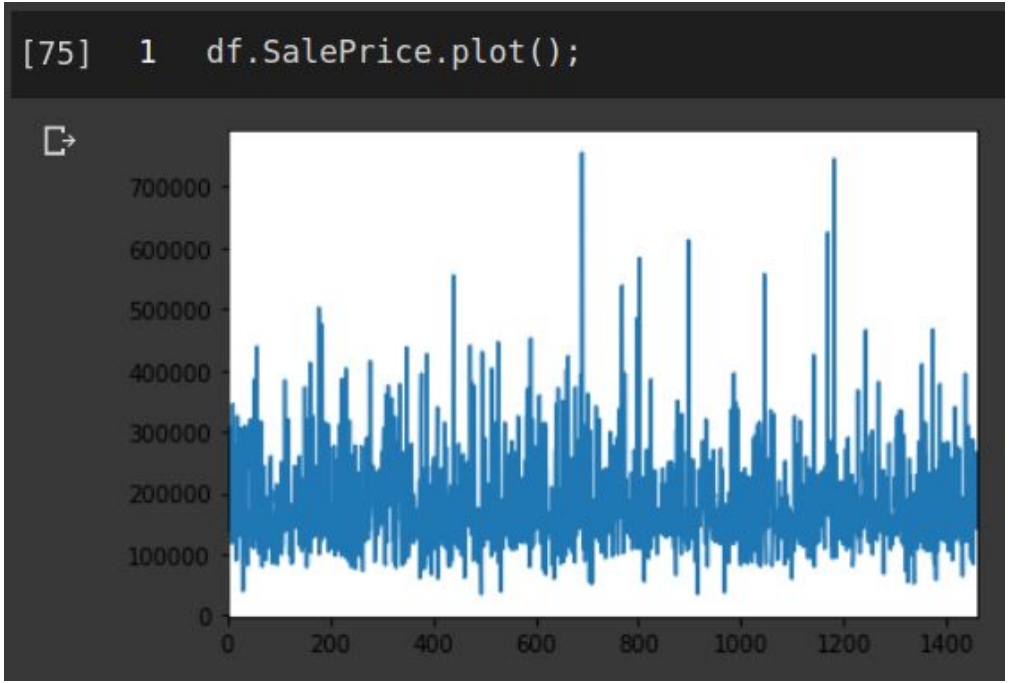
Plotando com pandas

```
[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

```
[75]  1  df.SalePrice.plot();
```

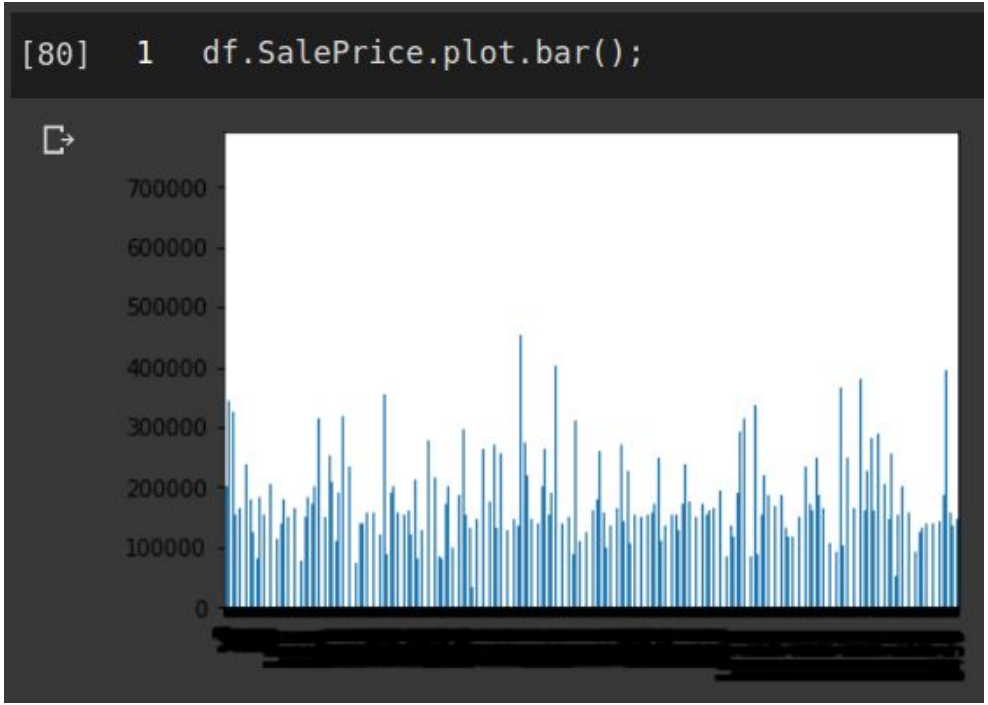


Plotando com pandas

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

Plotando com pandas

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



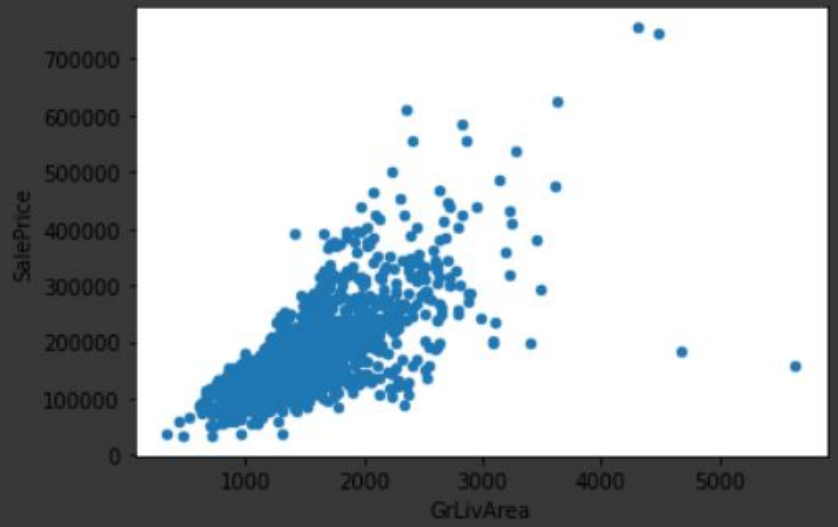
Plotando com pandas

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

IESB

Plotando com pandas

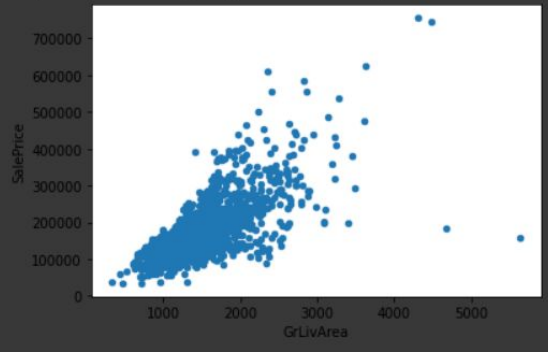
```
[34]  1  df[['GrLivArea','SalePrice']].plot.scatter(x='GrLivArea',y='SalePrice');
```



Plotando com pandas

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

```
matplotlib.axes._subplots.AxesSubplot
```



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:

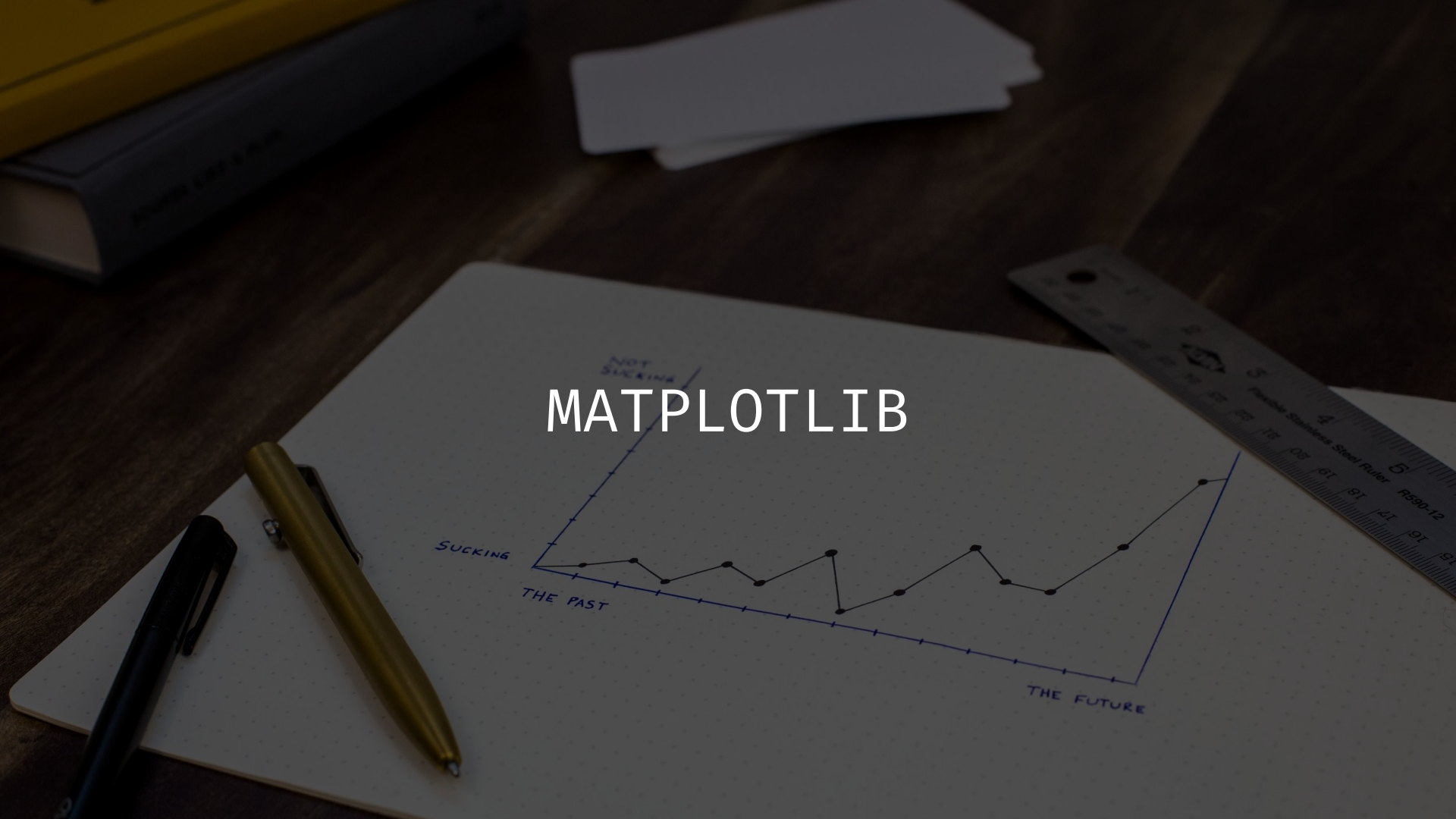
- A single color string referred to by name, RGB or RGBA code, for instance 'red' or '#a98d19'.
- A sequence of color strings referred to by name, RGB or RGBA code, which will be used for each point's color recursively. For instance ['green','yellow'] all points will be filled in green or yellow, alternatively.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

****kwargs**

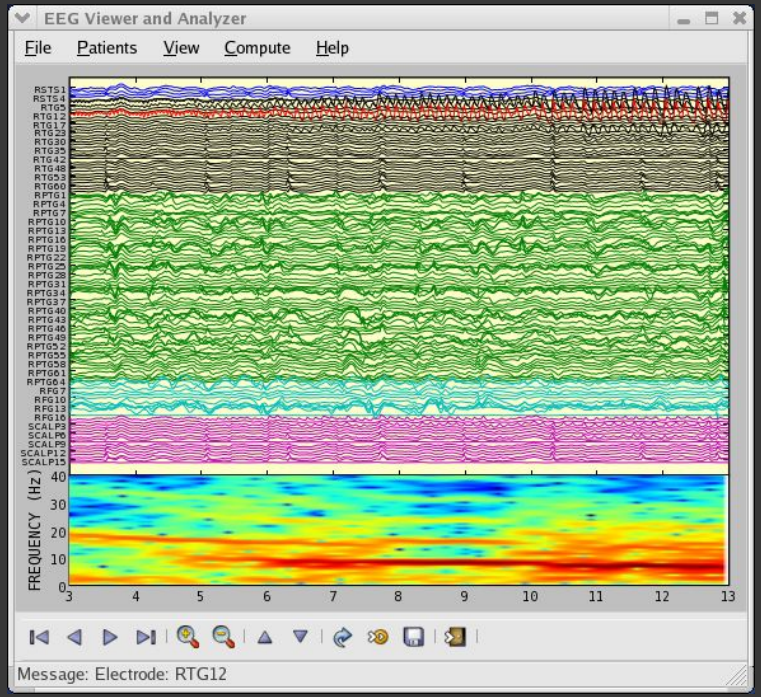
Keyword arguments to pass on to `DataFrame.plot()`.

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

MATPLOTLIB

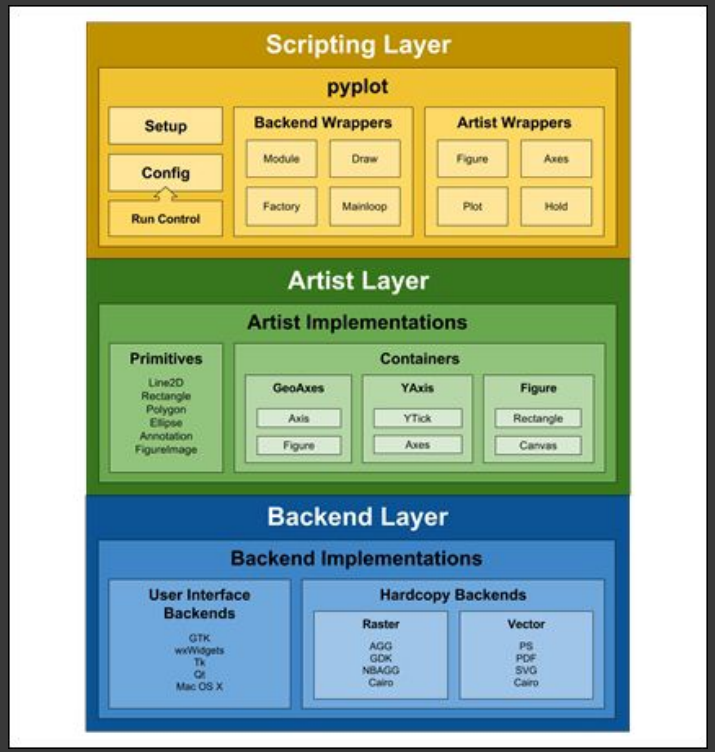


MATPLOTLIB



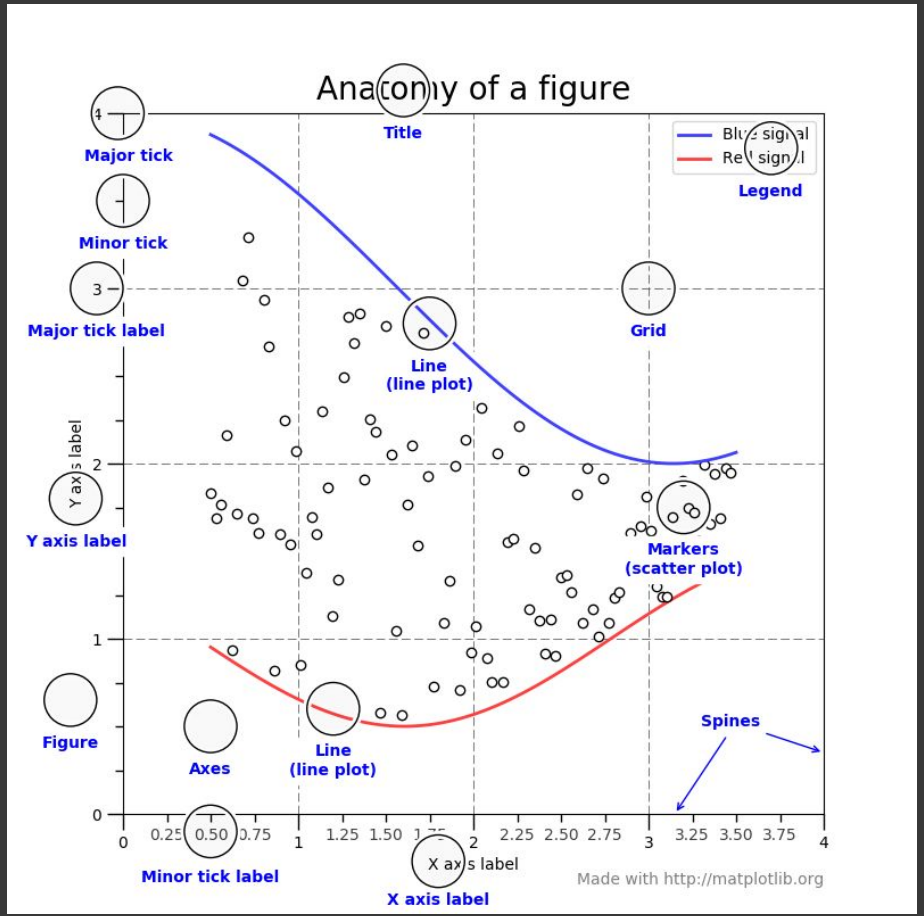
IESB

MATPLOTLIB



IESB

MATPLOTLIB


Anatomy of a figure
Title
Major tick
Minor tick
Major tick label
Legend
Grid
Line
(line plot)
Y axis label
Markers
(scatter plot)
Figure
Axes
Line
(line plot)
Spines
Minor tick label
X axis label
Made with http://matplotlib.org

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.

IESB

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]

IESB

MATPLOTLIB

plt.figure()

```
[2]   1  plt.figure();
```

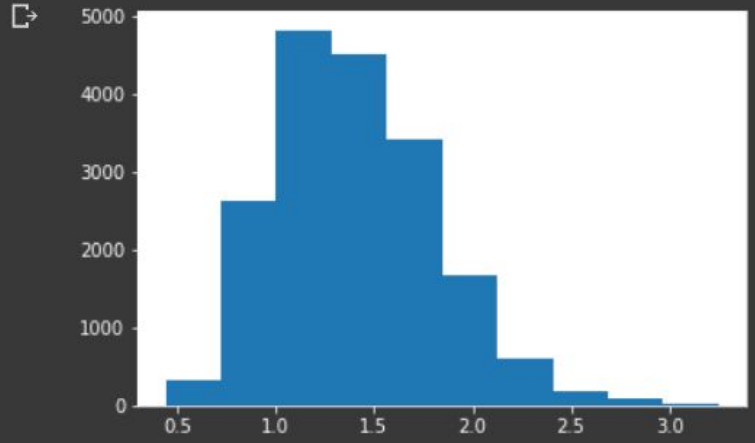
```
<Figure size 432x288 with 0 Axes>
```


IESB

MATPLOTLIB

plt.figure()

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

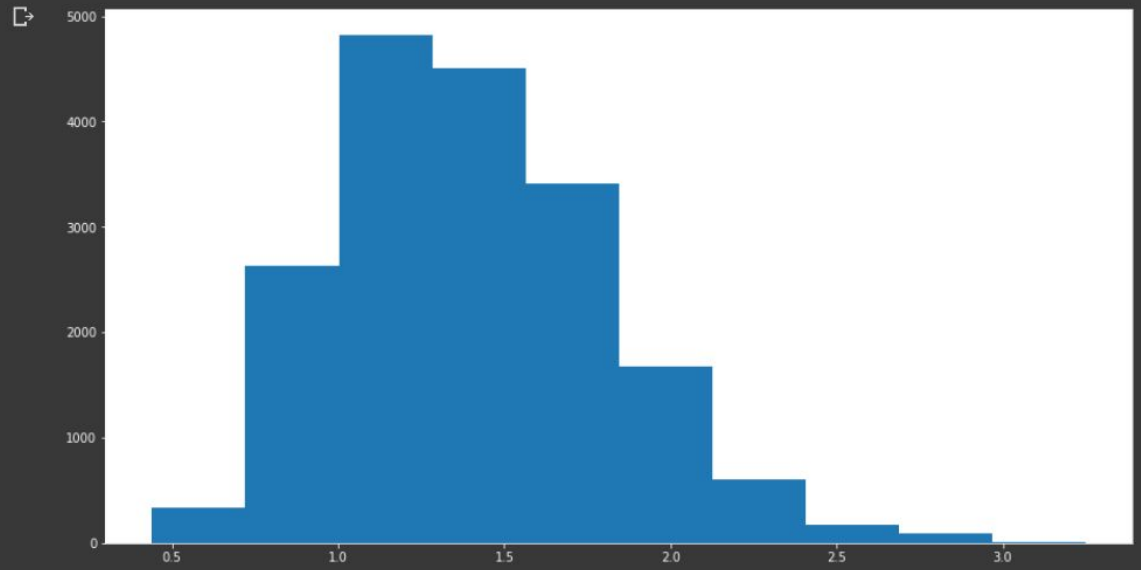


IESB

MATPLOTLIB

plt.figure()

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



IESB

MATPLOTLIB

`plt.bar()`

IESB

MATPLOTLIB

plt.bar()

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

year	AveragePrice
2015	1.375590
2016	1.338640
2017	1.515128
2018	1.347531

IESB

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
```

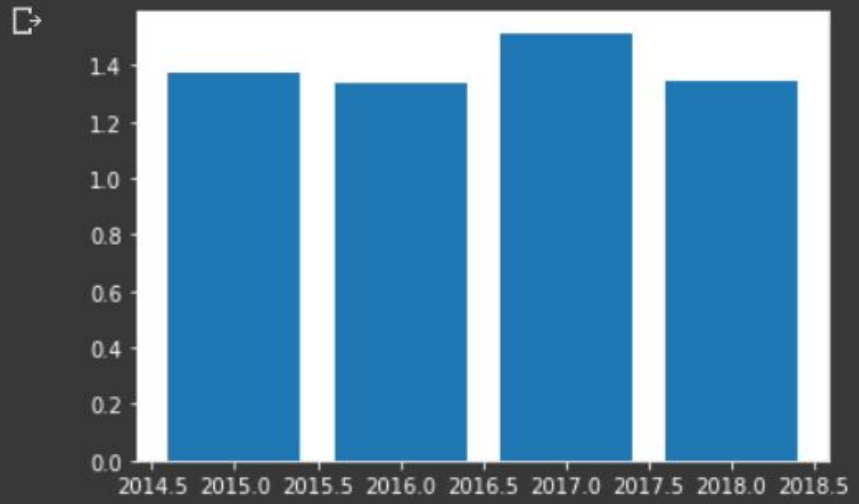
```
[2015, 2016, 2017, 2018]
```


IESB

MATPLOTLIB

`plt.bar()`

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



IESB

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)
```

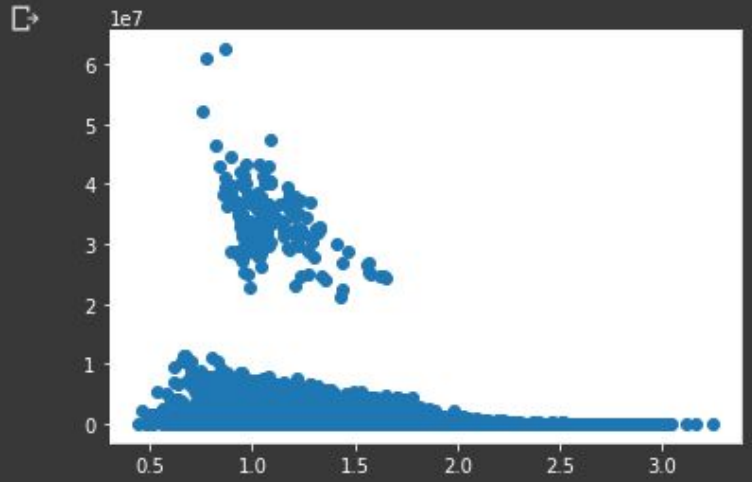
[source]

IESB

MATPLOTLIB

`plt.scatter()`

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



IESB

MATPLOT

`plt.scatter()`

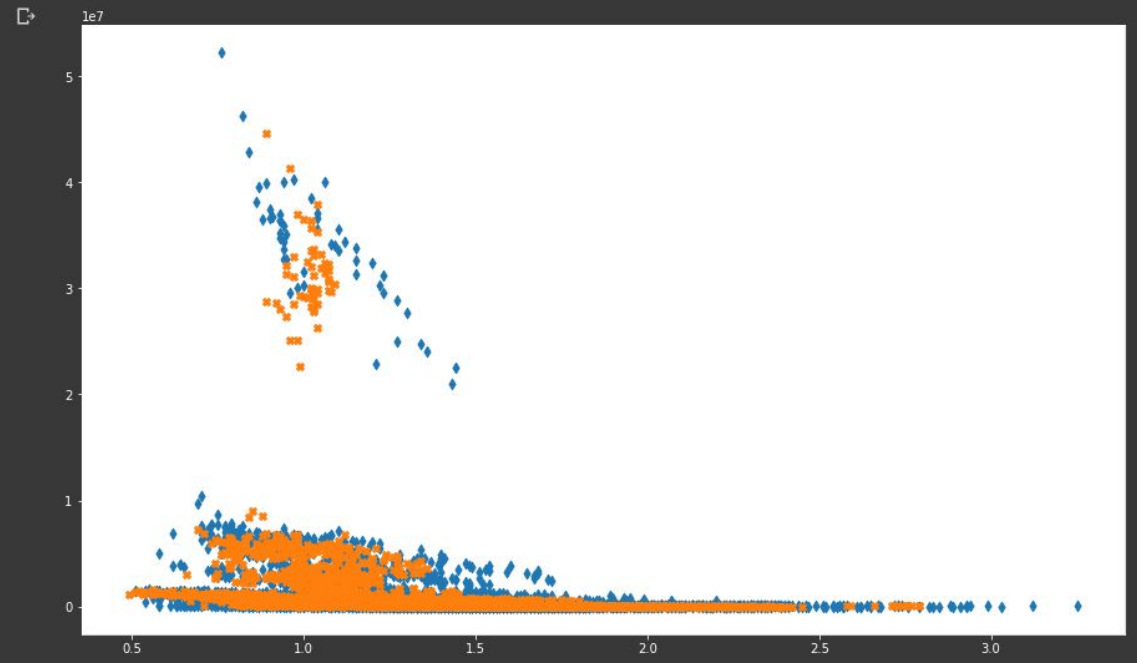
marker	symbol	description
"."	•	point
","	·	pixel
"o"	●	circle
"v"	▼	triangle_down
"^"	▲	triangle_up
"<"	◀	triangle_left
">"	▶	triangle_right
"1"	Y	tri_down
"2"	⅄	tri_up
"3"	≺	tri_left
"4"	≻	tri_right
"8"	●	octagon
"s"	■	square
"p"	⬟	pentagon
"P"	✚	plus (filled)
"*"	★	star
"h"	⬢	hexagon1
"H"	⬣	hexagon2
"+"	+	plus
"x"	×	x
"X"	✖	x (filled)
"D"	◆	diamond
"d"	♦	thin_diamond

IESB

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()
```

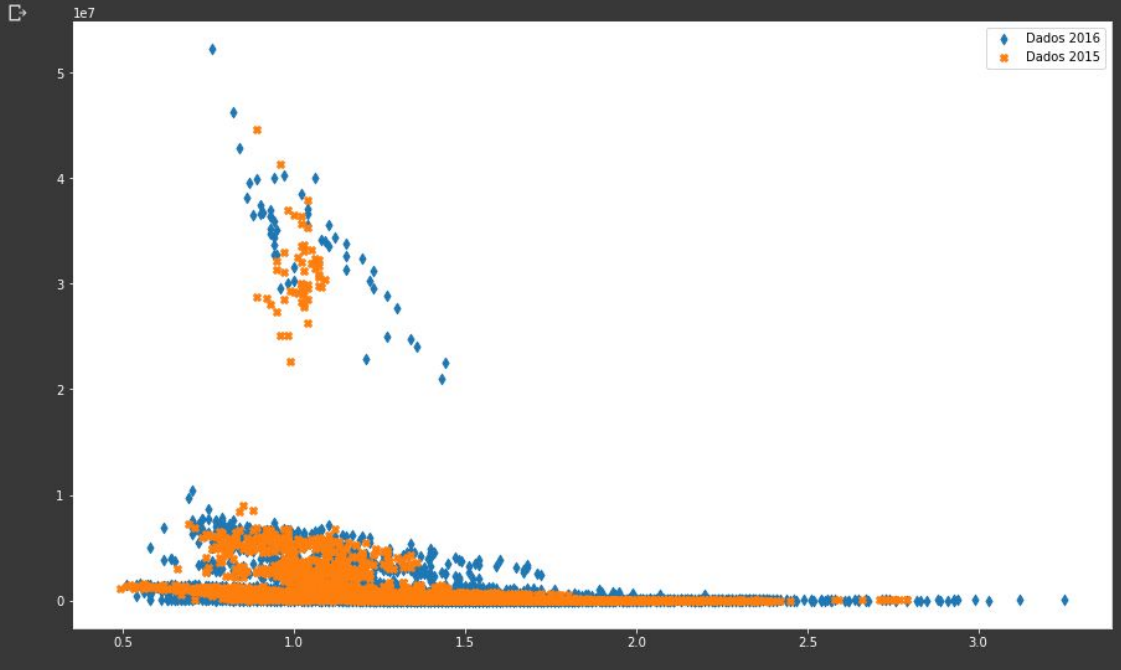


IESB

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()
```



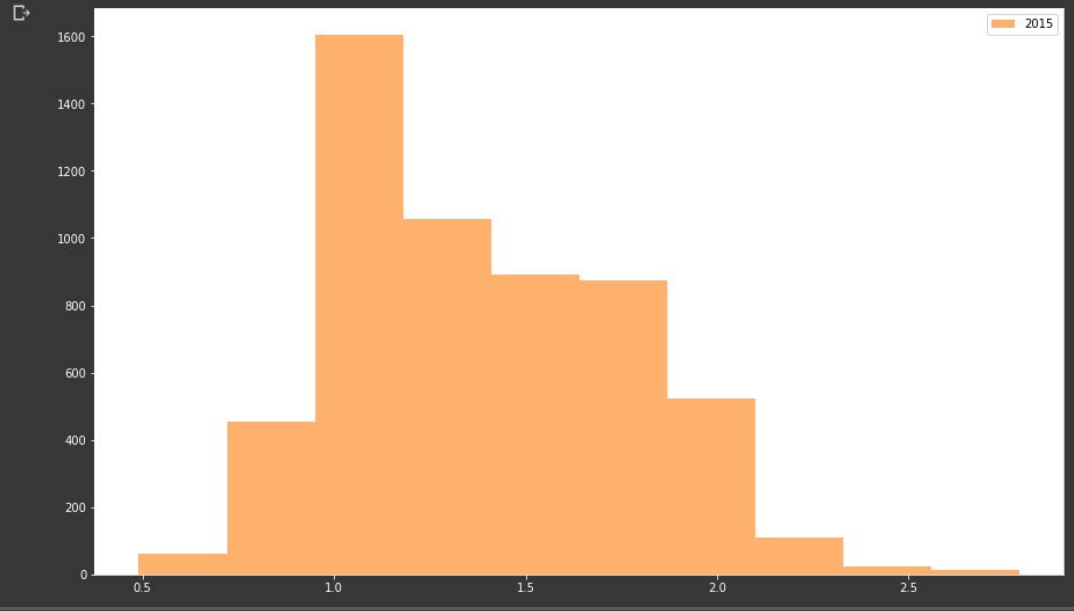
this method.

IESB

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()
```



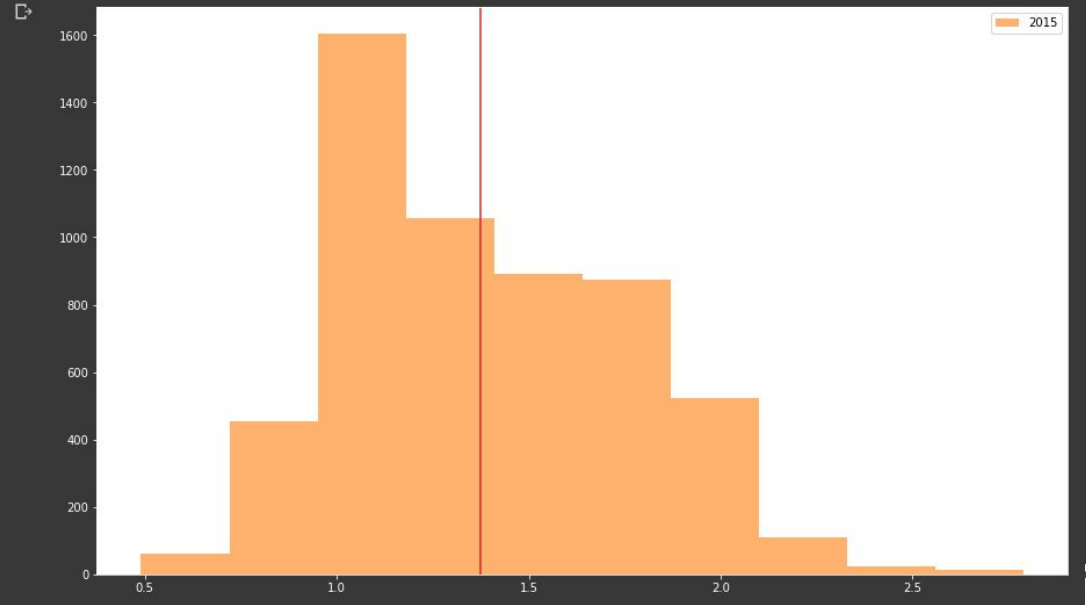
MATPLOTLIB

`plt.axvline()`

MATPLOTLIB

plt.axvline()

```
[68]  1  ## encontrado a media dos valores
      2
      3  plt.figure(figsize=(15,9))
      4  plt.hist(x = 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()`

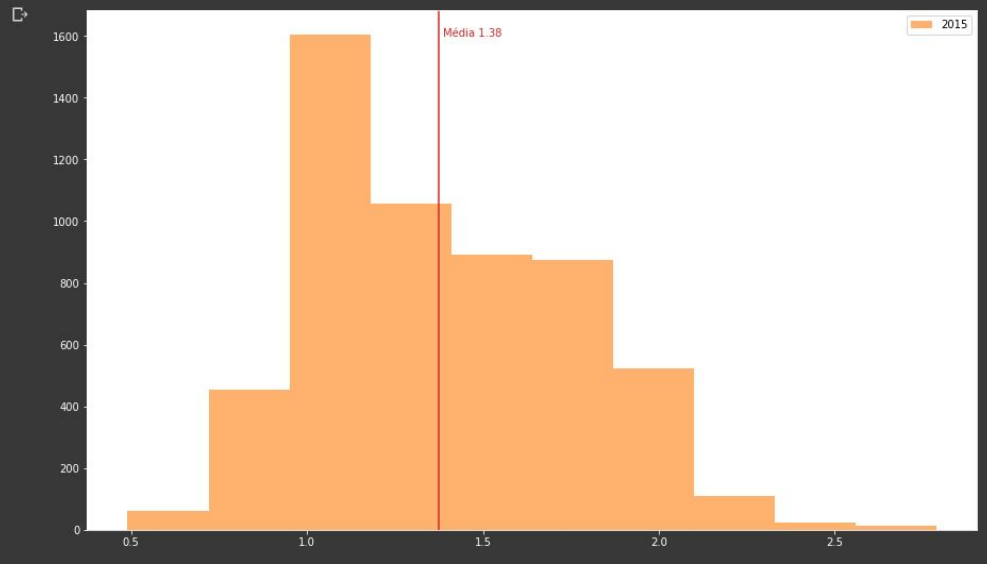
matplotlib.pyplot.text

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

MATPLOTLIB

plt.text()

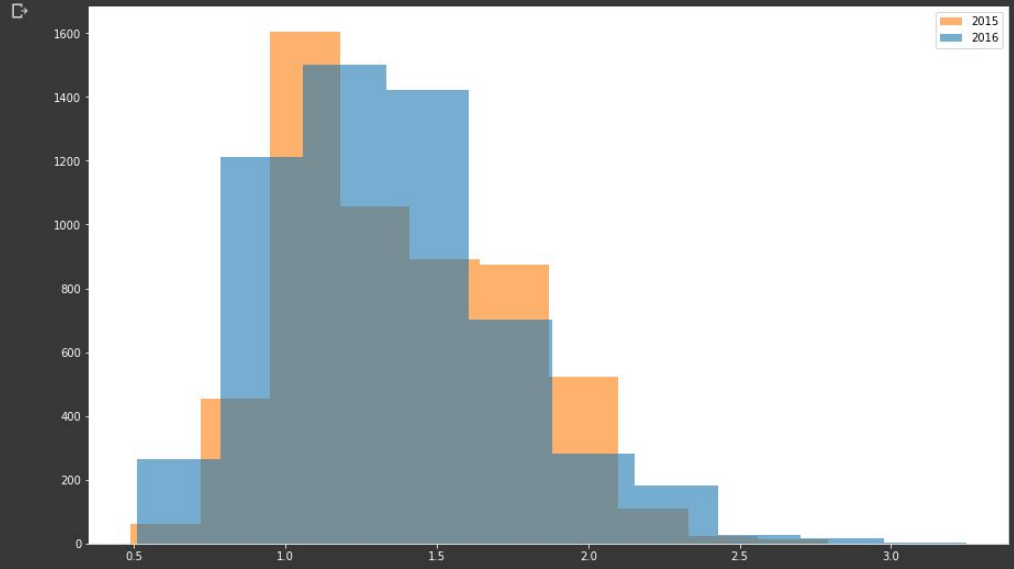
```
[77]  1  ## encontrado a media dos valores
      2
      3  media = df[df['year'] == 2015]['AveragePrice'].mean()
      4
      5  plt.figure(figsize=(15,9))
      6  plt.hist(x = df[df['year'] == 2015]['AveragePrice'],color='tab:orange',alpha=0.6,label='2015')
      7  plt.axvline(x = media ,color= '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

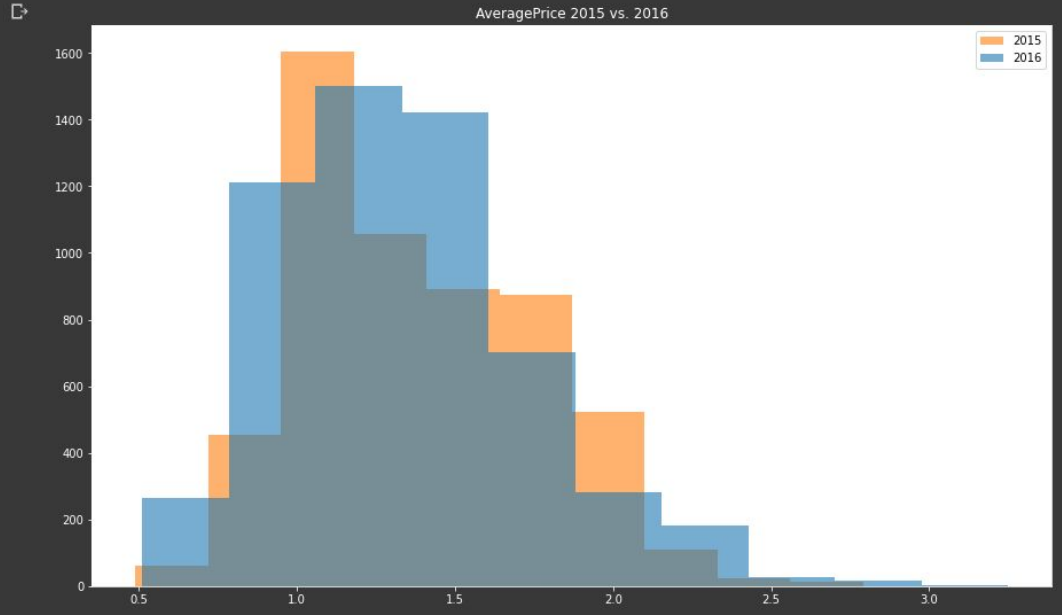
```
matplotlib.pyplot.title(label, fontdict=None, loc='center', pad=None, **kwargs)
```


IESB

MATPLOTLI

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()

```
## gerando os dados necessários para o próximo gráfico

dados = (


  df[['Date','Total Bags','year','region']]
  .query('year == 2015')
  .query("region == 'LasVegas'")
  .drop(['year','region'],axis=1)
  .assign(Date = lambda x: pd.to_datetime(x.Date,infer_datetime_format=True))
  .sort_values('Date')


)
```

	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

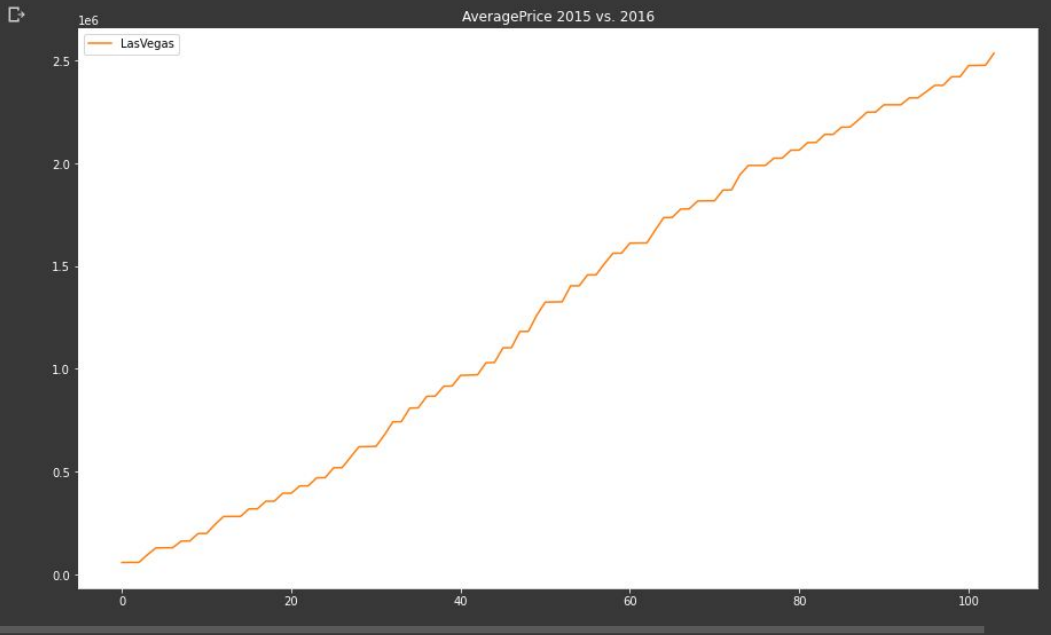
104 rows × 2 columns

IESB

MATPLOTLI

plt.plot()

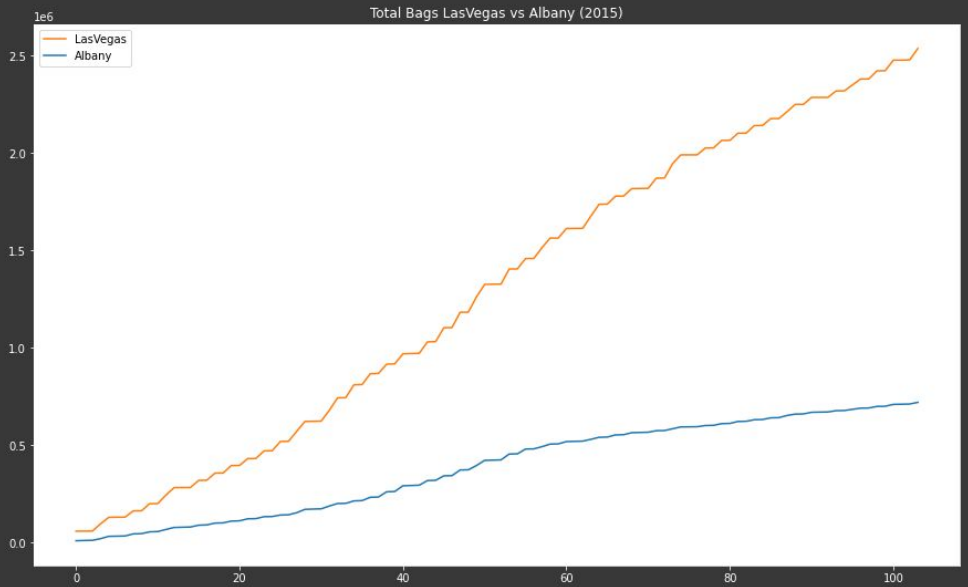
```
[117]  1  ## gerando o gráfico de linhas com a soma cumulativa
       2
       3  soma_cumulativa = dados['Total Bags'].cumsum()
       4
       5  plt.figure(figsize=(15,9))
       6  plt.plot(range(104),soma_cumulativa,color='tab:orange',label='LasVegas')
       7  plt.title('AveragePrice 2015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

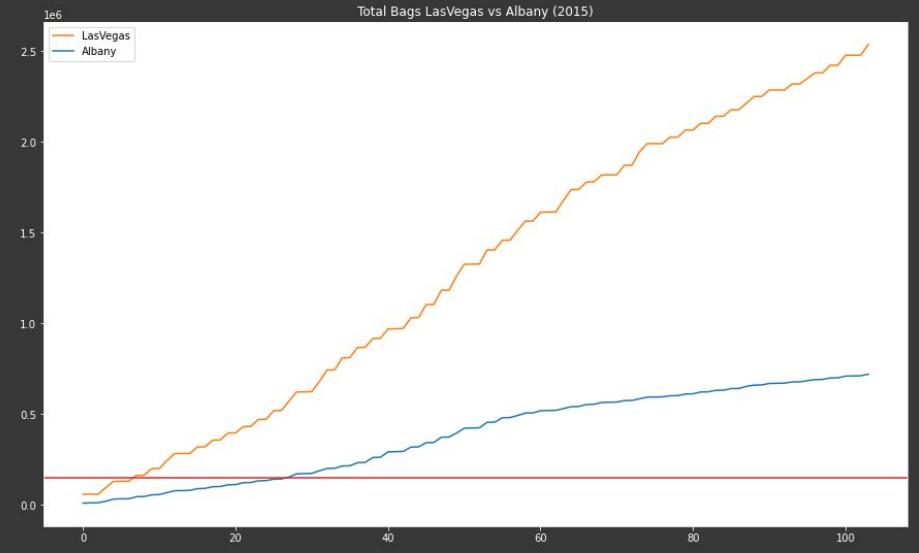
xmax : scalar, optional, default: 1

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

MATPLOTLIB

plt.axhline()

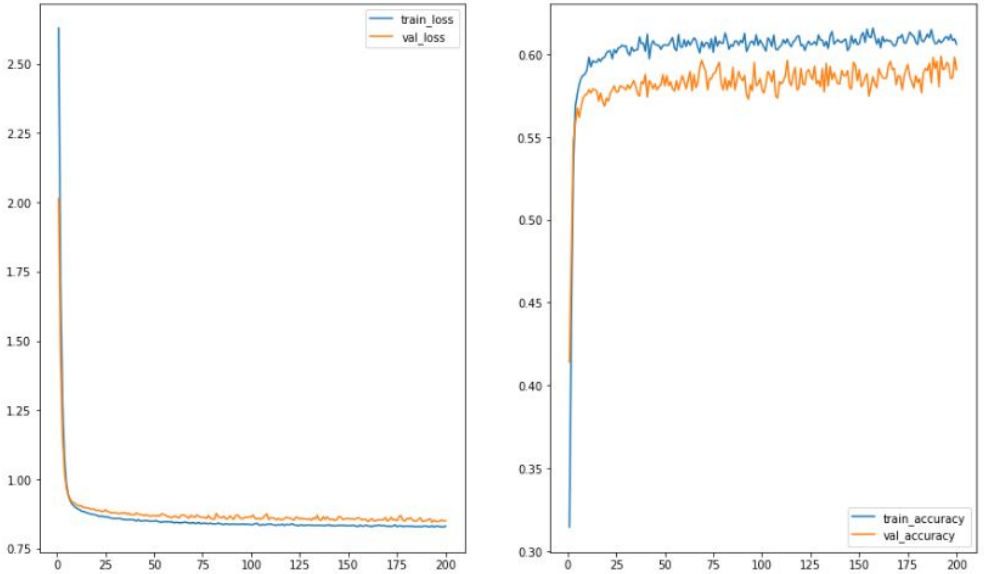
```
## adicionando uma linha horizontal no valor 150000

soma_cumulativa = dados['Total Bags'].cumsum()
soma_cumulativa_albany = dados_albany['Total Bags'].cumsum()

plt.figure(figsize=(15,9))
plt.plot(range(104),soma_cumulativa,color='tab:orange',label='LasVegas')
plt.plot(range(104),soma_cumulativa_albany,color='tab:blue',label='Albany')
plt.title('Total Bags LasVegas vs Albany (2015)',fontdict={'color':'white'})

plt.axhline(y=150000,color='tab:red')

plt.legend()
plt.show()
```



MATPLOTLIB

plt.subplots()

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

         model.generate_loss_curves()
```



IESB

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()

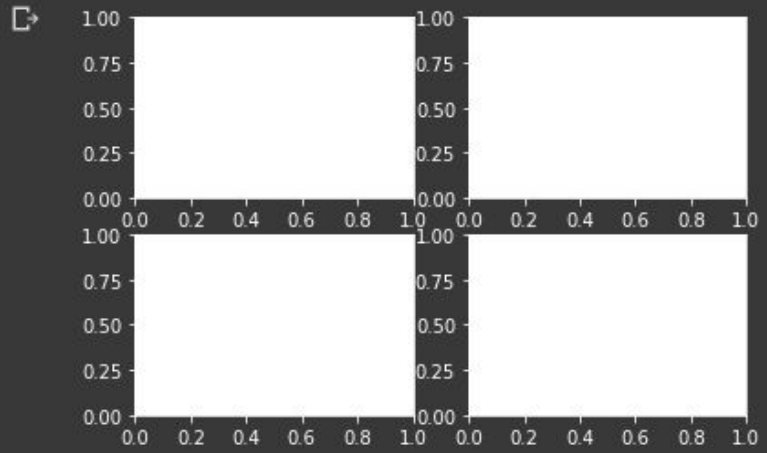
```
[39]  1  ## criando um subplot vazio
      2
      3  plt.subplots()
      4  plt.show()
```

IESB

MATPLOTLIB

plt.subplots()

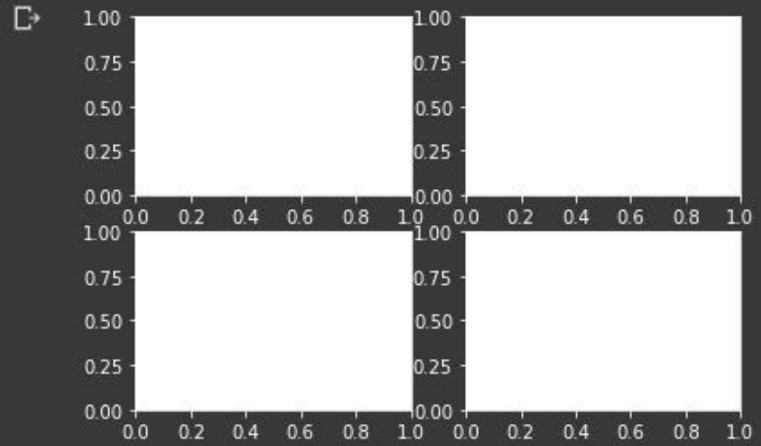
```
[44]  1  ## criando um subplot com 6 axes (2 linhas e 2 colunas)
      2
      3  plt.subplots(2,2)
      4  plt.show()
```



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'>
```


IESB

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'>
```


IESB

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.

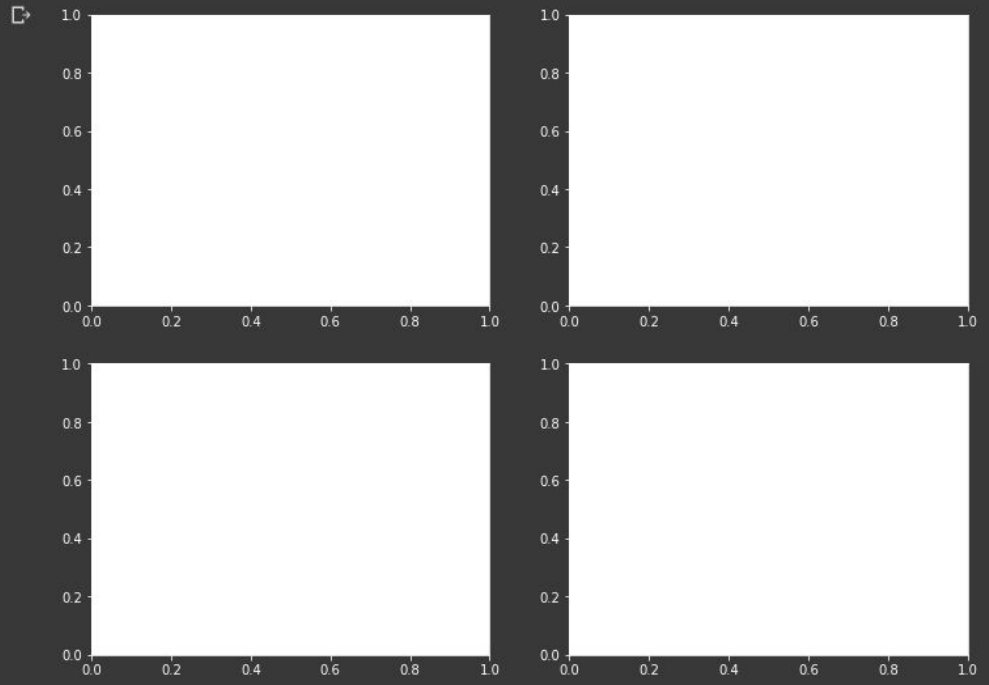
The Figure instance supports callbacks through a *callbacks* attribute which is a CallbackRegistry instance. The events you can connect to are 'dpi_changed', and the callback will be called with func(fig) where fig is the Figure instance.

IESB

MATPLOTLIB

plt.subplots()

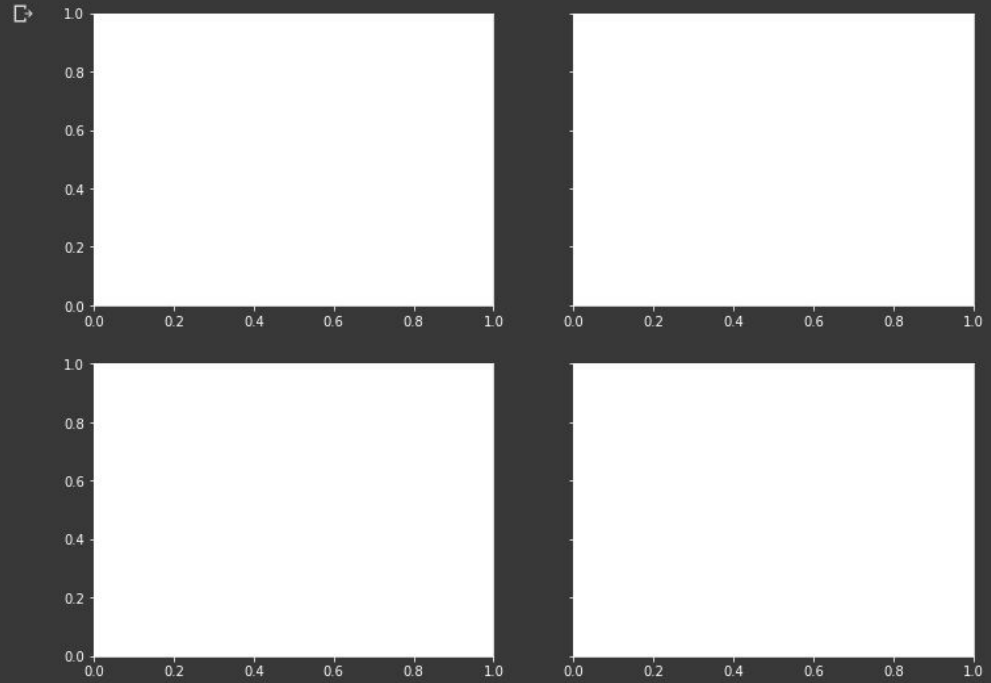
```
[51]  1  ## aumentado o tamanho do subplots
      2
      3  f, ax = plt.subplots(2,2,figsize=(12,9))
      4  plt.show()
```



MATPLOTL

plt.subplots()

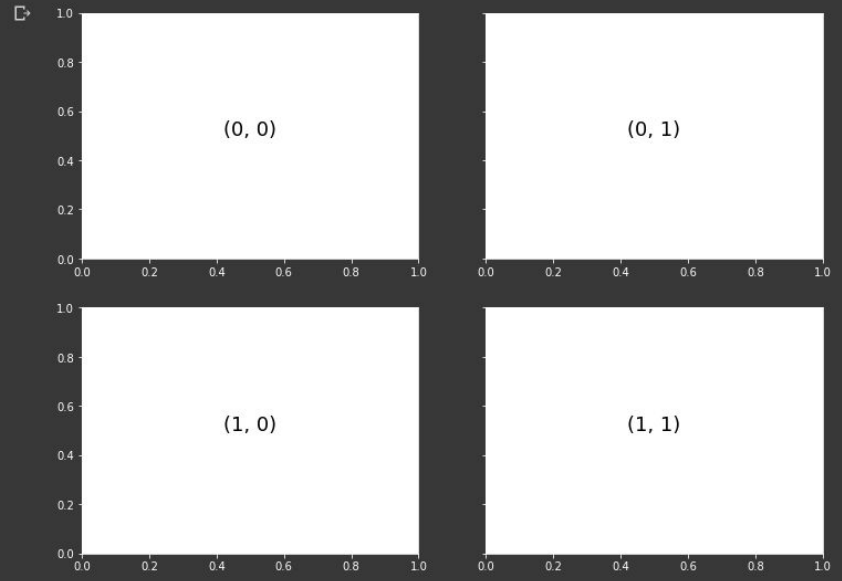
```
[52]  1  ## compartilhando os eixos y
      2
      3  f, ax = plt.subplots(2,2,figsize=(12,9),sharey=True)
      4  plt.show()
```



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
     11  plt.show()
```



MATPLOTI

plt.subplots()

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

	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

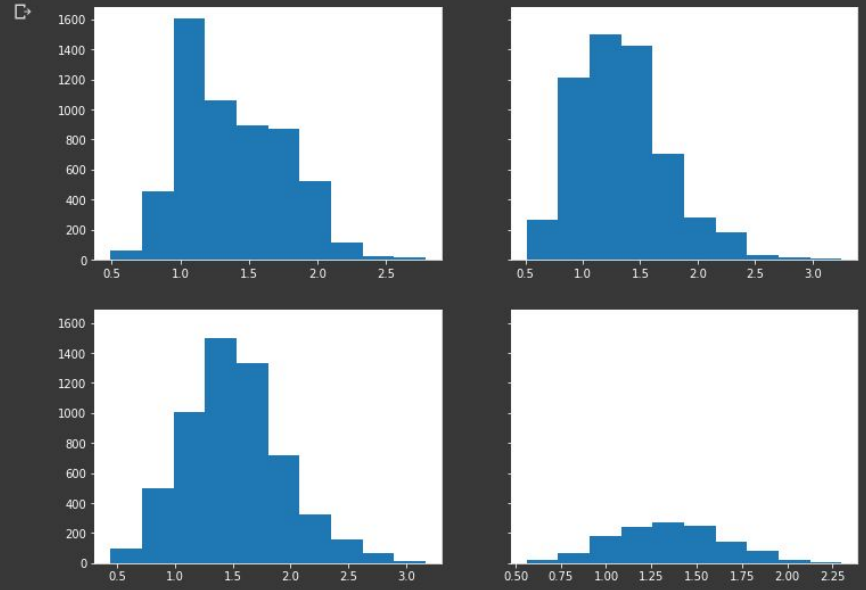
18249 rows × 2 columns

IESB

MATPLOTLIB

plt.subplots()

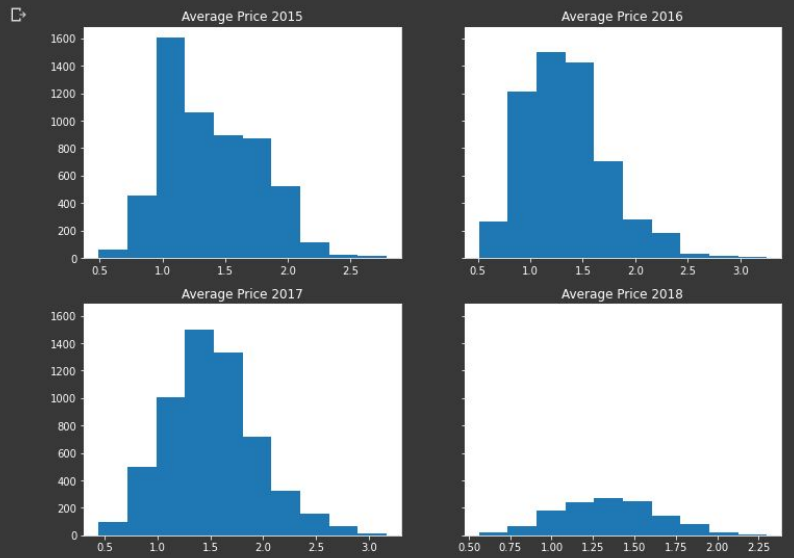
```
[74]  1  ## criando os gráficos em cada um dos eixos
      2
      3  f, ax = plt.subplots(2,2,figsize=(12,9),sharey=True)
      4
      5  ax[0,0].hist(subplot_data.query('year == 2015').AveragePrice)
      6  ax[0,1].hist(subplot_data.query('year == 2016').AveragePrice)
      7  ax[1,0].hist(subplot_data.query('year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

`plt.subplots()`

https://matplotlib.org/api/axes_api.html#plotting

matplotlib.axes

Table of Contents

IESB

MATPLOTLIB

mpimg.imread()

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


IESB

MATPLOTLIB

`mpimg.imread()`

IESB

MATPLOTLIB

mpimg.imread()

```
[81]  1  ## abrindo a imagem
      2
      3  img = mpimg.imread('/content/ceu.jpeg')
      4  img
```

```
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)
```


IESB

MATPL

`plt.imshow()`

```
[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)
```


IESB

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()
```