Relatório do bootcamp - 4

Thassiana C. A. Muller

Jupyter Notebook

Jupyter Notebook é um ambiente interativo que permite criar e compartilhar documentos que possuem código executável, imagens, textos e equações matemáticas. Originalmente, foi desenvolvido para Python mas agora já possui suporte para diversas linguagens de programação através de diferentes kernels. É amplamente utilizado em ciência de dados, aprendizado de máquina e pesquisa científica, pois facilita a integração e visualização do código e documentação em um único arquivo. Seus arquivos são do tipo .ipynb e são suportados em diversas plataformas

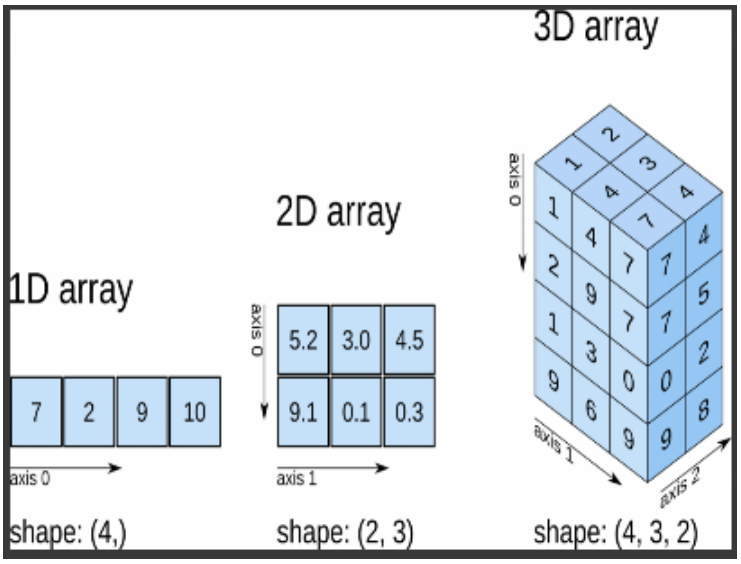
Para meu uso pessoal, gosto de criar esse tipo de arquivo com a alternativa disponibilizada pela Google chamada de Colab, que possui recursos computacionais em nuvem, e a alternativa local no editor de textos da Microsoft Visual Studio Code (VScode) que também possui suporte para .ipynb.

No link a seguir, está um projeto de criação de um filtro passa-baixa digital utilizando um notebook no VScode. https://github.com/ThassiAmorim/FiltroDigitalPassaBaixa

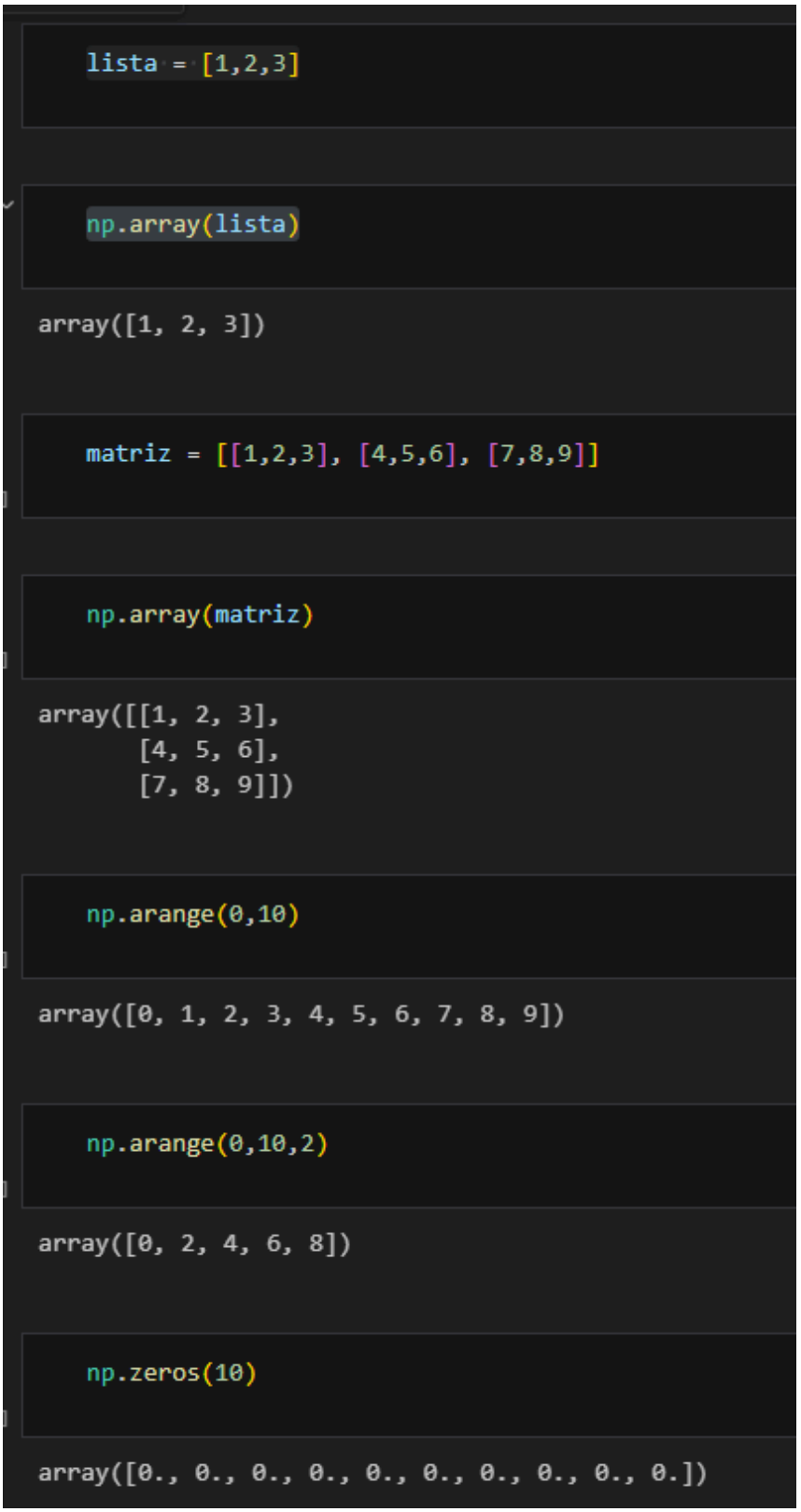
Numpy

NumPy é uma biblioteca em Python, que fornece suporte para arrays e matrizes multidimensionais, além de funções matemáticas para operar sobre esses tipos de dados. É muito eficiente pois aproveita implementações em C para operações de baixo nível, agilizando tarefas que envolvem um grande processamento de dados numéricos. Essa biblioteca também serve como base para muitas outras, como SciPy e Pandas.

Em anexo no card está o notebook completo de estudo do NumPy.



Arrays podem ser criados a partir de listas do Python ou funções de criação como em:



```
lista = [1,2,3]
```

```
np.array(lista)
```

```
array([1, 2, 3])
```

```
matriz = [[1,2,3], [4,5,6], [7,8,9]]
```

```
np.array(matriz)
```

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

```
np.arange(0,10)
```

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

```
np.arange(0,10,2)
```

```
array([0, 2, 4, 6, 8])
```

```
np.zeros(10)
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

Ou gerados aleatoriamente como em:

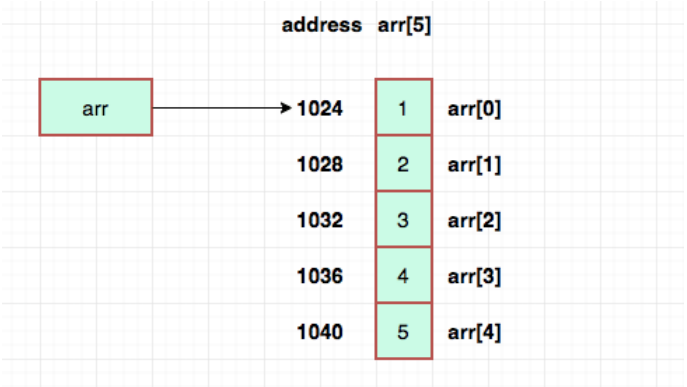
```
np.random.rand(5,4,3)
```

```
array([[[0.47283734, 0.15524364, 0.10052894],
        [0.97196144, 0.73489488, 0.39962538],
        [0.53415081, 0.70597301, 0.11989373],
        [0.48043365, 0.11993069, 0.21956825]],

       [[0.06806987, 0.72574727, 0.17691246],
        [0.65521508, 0.03823606, 0.97140936],
        [0.4102406 , 0.71830984, 0.66654938],
        [0.34624924, 0.27551193, 0.10802752]],

       [[0.38689492, 0.58649002, 0.12544501],
        [0.25177139, 0.73072564, 0.68788274],
        [0.73478659, 0.82563282, 0.80446287],
        [0.22331461, 0.48261186, 0.93471869]],

       [[0.94411469, 0.14214903, 0.61854853],
        [0.09652394, 0.15412652, 0.31412398],
        [0.47472787, 0.56220836, 0.61647782],
        [0.4727591 , 0.94734389, 0.73294256]],

       [[0.5326321 , 0.32073952, 0.4154591 ],
        [0.03148266, 0.6903206 , 0.58394596],
        [0.87683082, 0.61196745, 0.7619619 ],
        [0.84721185, 0.48803856, 0.55739764]]])
```

```
np.random.randn(4) # distribuição normal
```

```
array([ 0.47003825,  0.4067185 ,  0.20641102, -0.64881987])
```

Dessa forma, para a criação de arrays, o NumPy aloca dentro da RAM os valores em endereços sequenciais, igualmente espaçados de acordo com o tipo de dado do array.



Elementos específicos dos arrays podem ser acessados a partir de fatiamentos, isso pode ser feito como a seguir:

```
arr = np.arange(0,30)
arr
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])
```

```
arr[3]
```

```
3
```

```
arr[2:5] # final não inclusivo
```

```
array([2, 3, 4])
```

```
arr[:5]
```

```
array([0, 1, 2, 3, 4])
```

```
arr[5:]
```

```
array([ 5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
       22, 23, 24, 25, 26, 27, 28, 29])
```

```
arr[2:18:4]
```

```
array([ 2,  6, 10, 14])
```

Ou de forma a acessar índices de trás para frente como em

```
arr[-1]
```

✓ 0.0s

```
29
```

```
arr[-1:]
```

✓ 0.0s

```
array([29])
```

```
arr[:-1]
```

✓ 0.0s

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28])
```

```
arr[::-1] # inverte array
```

✓ 0.0s

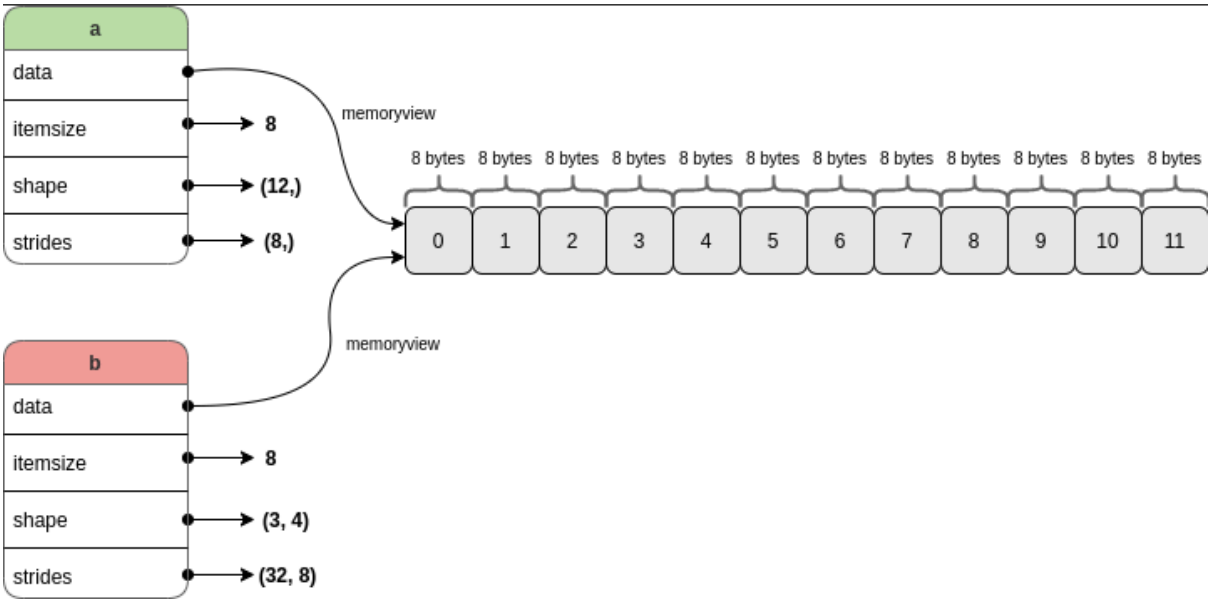
```
array([29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13,
       12, 11, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0])
```

```
arr[-15:-1] = 100
arr
```

✓ 0.0s

```
array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,
        13,  14, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
       100, 100, 100,  29])
```

Como o propósito do NumPy é a eficiência, ao atribuir um array a outro é feito somente uma referência do local da memória armazenado



Portanto, caso um array “a” seja atribuído a um array “b”, qualquer mudança em “b” será refletida em “a”

```
arr2 = arr[:3] # arr2 aponta para o ary[:3], se modificar arr2 modifica arr
arr2
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29]])
```

```
arr
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44, 45, 46, 47, 48, 49]])
```

```
arr2[:] = 100
```

```
arr2
```

```
array([[100, 100, 100, 100, 100, 100, 100, 100, 100, 100],
       [100, 100, 100, 100, 100, 100, 100, 100, 100, 100],
       [100, 100, 100, 100, 100, 100, 100, 100, 100, 100]])
```

```
arr
```

```
array([[100, 100, 100, 100, 100, 100, 100, 100, 100, 100],
       [100, 100, 100, 100, 100, 100, 100, 100, 100, 100],
       [100, 100, 100, 100, 100, 100, 100, 100, 100, 100],
       [ 30,  31,  32,  33,  34,  35,  36,  37,  38,  39],
       [ 40,  41,  42,  43,  44,  45,  46,  47,  48,  49]])
```

Também é possível filtrar os elementos a partir de uma condição booleana:

```
bol = arr > 20
bol
```

✓ 0.0s

```
array([[False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True]])
```

```
arr[bol]
```

✓ 0.0s

```
array([21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
       38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])
```

Operações matemáticas podem ser realizadas entre arrays e escalares contanto que que respeitem as dimensões.

```
arr
```
✓ 0.0s
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```

```
arr + arr
```
✓ 0.0s
```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30])
```

```
np.add(arr, arr)
```
✓ 0.0s
```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30])
```

```
arr - arr
```
✓ 0.0s
```
array([  0,   1,   4,   9,  16,  25,  36,  49,  64,  81, 100, 121, 144,
       169, 196, 225])
```

```
arr * arr
```
✓ 0.0s
```
array([  0,   1,   4,   9,  16,  25,  36,  49,  64,  81, 100, 121, 144,
       169, 196, 225])
```

```
arr/arr # warning que causa um not a number
```
✓ 0.0s
```
C:\Users\thass\AppData\Local\Temp\ipykernel_19928\1862401812.py:1: RuntimeWarning: invalid value encountered in divide
  arr/arr

array([nan,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
        1.,  1.,  1.])
```

```
arr * 2
```
✓ 0.0s
```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30])
```

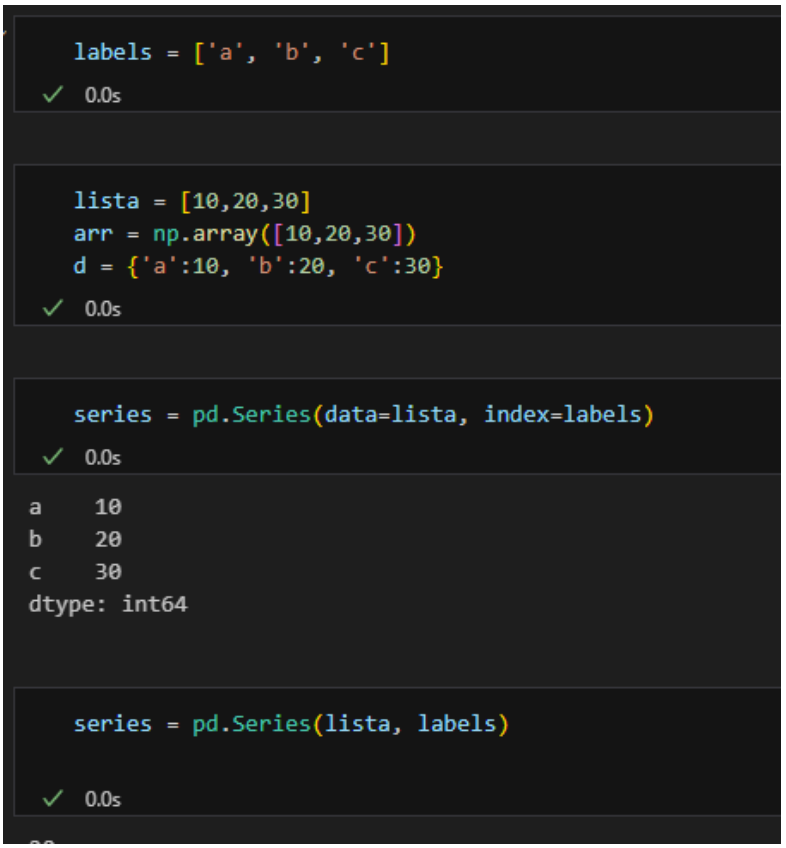
```
arr ** 2
```
✓ 0.0s
```
array([  0,   1,   4,   9,  16,  25,  36,  49,  64,  81, 100, 121, 144,
       169, 196, 225])
```

Pandas

A biblioteca Pandas também é uma ferramenta do Python, foi escrita sobre o NumPy e é amplamente utilizada em ciência de dados e aprendizado de máquina. Ela oferece estruturas de dados como dataframes, que permitem a manipulação e análise de grandes conjuntos de dados de maneira eficiente. Também é possível realizar operações complexas de limpeza, transformação e agregação de dados, além de fornecer uma boa visualização e modelagem de dados de diferentes tipos de dados.
Em anexo no card está o notebook completo de estudo do Pandas.

Series

Series é uma estrutura de dados do tipo chave:valor semelhante a um dicionário. A criação de series pode ser feita como:



Suportam os mais diferentes tipos de dados:

```
pd.Series([sum, print, len])
```

✓ 0.0s

```
0      <built-in function sum>
1    <built-in function print>
2      <built-in function len>
dtype: object
```

E podem ser concatenadas:

```
ser1 = pd.Series(data=[1,2,3,4],
                 index=[ 'Alemanha','EUA', 'Russia', 'China'])

ser2 = pd.Series(data=[1,2,3,4],
                 index=['EUA', 'Alemanha', 'Italia', 'China'])
```

✓ 0.0s

```
ser1 + ser2
```

✓ 0.0s

```
Alemanha    3.0
China       8.0
EUA         3.0
Italia      NaN
Russia      NaN
dtype: float64
```

DataFrames

DataFrames são estruturas de dados semelhantes a uma planilha, possuem índices e colunas para categorizar seus dados, podem ser criados assim:

```
df = pd.DataFrame(data=np.random.randn(5,4),
                  index='A B C D E'.split(),
                  columns='W X Y Z'.split())
```

✓ 0.0s

```
df
```

✓ 0.0s

	W	X	Y	Z
A	0.302665	1.693723	-1.706086	-1.159119
B	-0.134841	0.390528	0.166905	0.184502
C	0.807706	0.072960	0.638787	0.329646
D	-0.497104	-0.754070	-0.943406	0.484752
E	-0.116773	1.901755	0.238127	1.996652

E o método de acesso aos dados pode ser feito como a seguir:

```
df.loc['A','W']
```

✓ 0.0s

```
0.3026654485851825
```

```
df.loc['A']
```

✓ 0.0s

```
W    0.302665
X    1.693723
Y   -1.706086
Z   -1.159119
Name: A, dtype: float64
```

```
df.loc[['A','B'], ['Z']]
```

✓ 0.0s

	Z
A	-1.159119
B	0.184502

```
df.iloc[1:4, 2:]
```

✓ 0.0s

	Y	Z
B	0.166905	0.184502
C	0.638787	0.329646
D	-0.943406	0.484752

Seleções condicionais também podem ser feitas:

```
bol1 = df > 0
```
✓ 0.0s

```
df[bol1]
```
✓ 0.0s

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	NaN	NaN	0.605965
C	NaN	0.740122	0.528813	NaN
D	0.188695	NaN	NaN	0.955057
E	0.190794	1.978757	2.605967	0.683509

```
df[df['W']>0]
```
✓ 0.0s

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

```
df[df['W']>0]['Y']
```
✓ 0.0s

```
A    0.907969
B   -0.848077
D   -0.933237
E    2.605967
Name: Y, dtype: float64
```

Caso queira-se filtrar mais de um critério, utilizar “|” para ou e “&” para and:

```
df[(df['W']>0) & (df['Y']>1)]
```
✓ 0.0s

	W	X	Y	Z
E	0.190794	1.978757	2.605967	0.683509

Alguns métodos para tratamento de nulos são:

```
df
```

✓ 0.0s

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2
2	NaN	NaN	3

```
df.dropna()
```

✓ 0.0s

	A	B	C
0	1.0	5.0	1

```
df.fillna(100)
```

✓ 0.0s

	A	B	C
0	1.0	5.0	1
1	2.0	100.0	2
2	100.0	100.0	3

+ Code

```
df['A'].fillna(value=df['A'].mean())
```

✓ 0.0s

```
0    1.0
1    2.0
2    1.5
Name: A, dtype: float64
```

```
df.ffill()
```

✓ 0.0s

	A	B	C
0	1.0	5.0	1
1	2.0	5.0	2
2	2.0	5.0	3

Para agrupamento dos dados pode-se utilizar:

```
df
```

✓ 0.0s

	Empresa	Nome	Venda
0	Google	Sam	200
1	Google	Charlie	120
2	Microsoft	Amy	340
3	Microsoft	Vanessa	124
4	FaceBook	Carl	243
5	FaceBook	Sarah	350

```
group = df.groupby('Empresa')
```

✓ 0.0s

```
group.sum()
```

✓ 0.0s

Empresa	Nome	Venda
FaceBook	CarlSarah	593
Google	SamCharlie	320
Microsoft	AmyVanessa	464

```
group.describe()
```

✓ 0.0s

	Venda							
	count	mean	std	min	25%	50%	75%	max
Empresa								
FaceBook	2.0	296.5	75.660426	243.0	269.75	296.5	323.25	350.0
Google	2.0	160.0	56.568542	120.0	140.00	160.0	180.00	200.0
Microsoft	2.0	232.0	152.735065	124.0	178.00	232.0	286.00	340.0

Exercícios resolvidos podem ser encontrados em anexo ao card juntamente a esse documento.