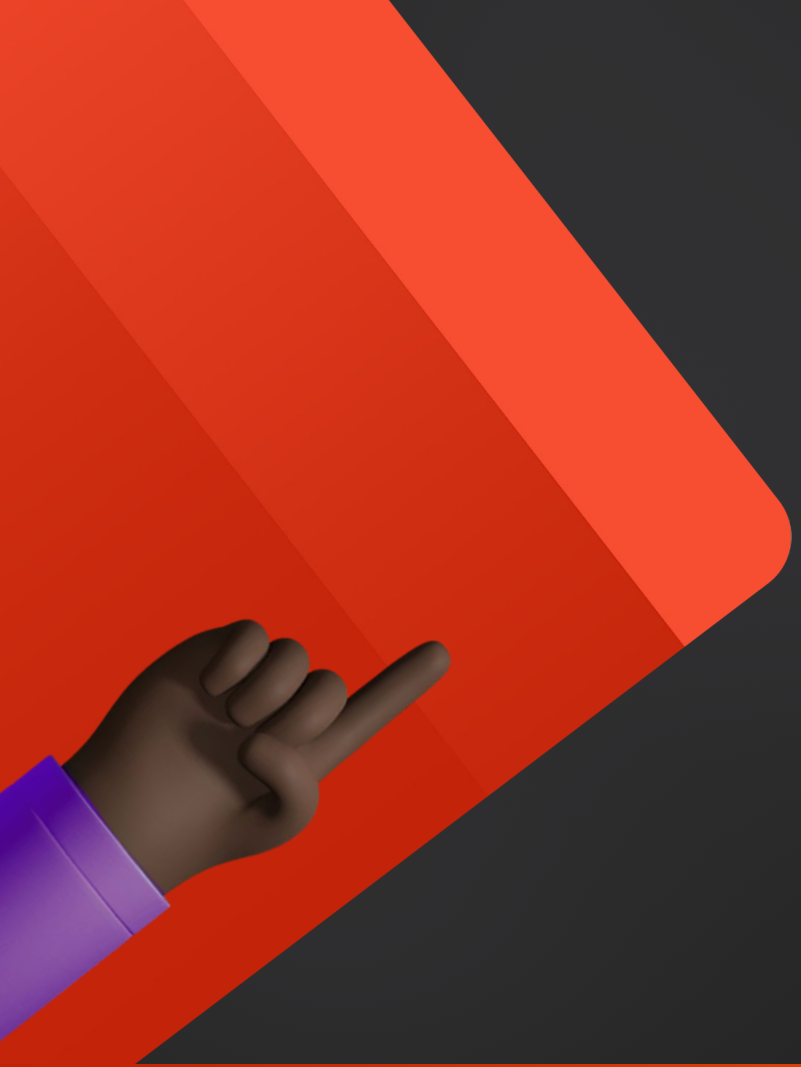
DESTRAVE O CONHECIMENTO

/mesttra



Nasser Boan

Funções Personalizadas

Funções Personalizadas

Uma função é um bloco de código que só é executado quando chamado.

Cada função tem um identificador e assim que definido estará sempre associado ao bloco de código.

O ato de definir uma função não executa o seu código. Para criar um função basta utilizarmos a palavra reservada 'def'.

```
[127]  1  def minha_primeira_funcao():
       2    print('O início de um sonho!')
       3    for _ in range(5):
       4      print('.')
       5    print('Deu tudo certo!')
```

```
[128]  1  minha_primeira_funcao()
```

```
O início de um sonho!
.
.
.
.
.
Deu tudo certo!
```

```
[130]  1  def hello_world():
       2    print('Hello World!')
       3
       4  hello_world()
```

```
Hello World!
```

Funções Personalizadas

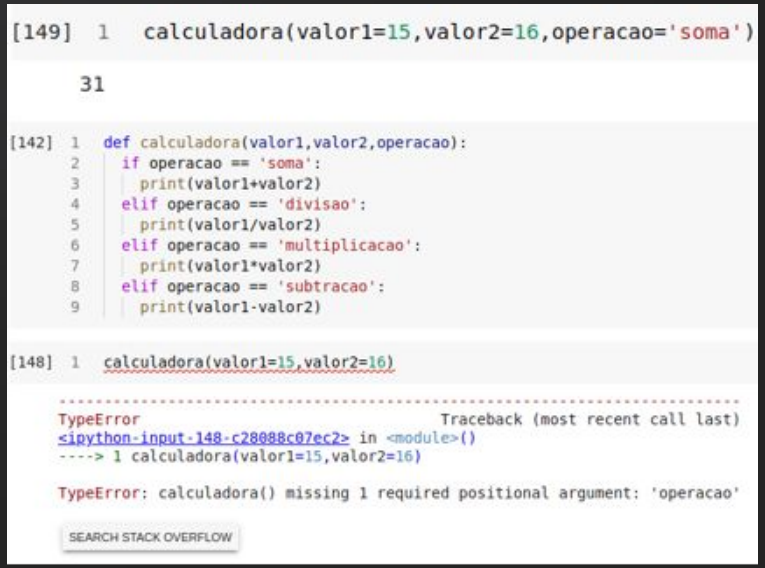
Podemos criar funções que recebem parâmetros. Parâmetros são variáveis definidas no momento de criação da função que serão utilizadas no escopo da função.

```
[133]  1  def hello_person(nome):
       2    print(f'Hello {nome}!')

[147]  1  hello_person(nome = 'Maria')

       Hello Maria!
```

Funções Personalizadas

Podemos criar funções com vários parâmetros. Os valores passados aos parâmetros no momento de executarmos uma função são chamados de argumentos.

Se não entregarmos todos os argumentos para preencher os parâmetros a função não será executada.



```
[149]  1  calculadora(valor1=15,valor2=16,operacao='soma')

       31

[142]  1  def calculadora(valor1,valor2,operacao):
       2    if operacao == 'soma':
       3      print(valor1+valor2)
       4    elif operacao == 'divisao':
       5      print(valor1/valor2)
       6    elif operacao == 'multiplicacao':
       7      print(valor1*valor2)
       8    elif operacao == 'subtracao':
       9      print(valor1-valor2)

[148]  1  calculadora(valor1=15,valor2=16)

TypeError                                 Traceback (most recent call last)
<ipython-input-148-c28088c07ec2> in <module>()
----> 1 calculadora(valor1=15,valor2=16)

TypeError: calculadora() missing 1 required positional argument: 'operacao'

SEARCH STACK OVERFLOW
```

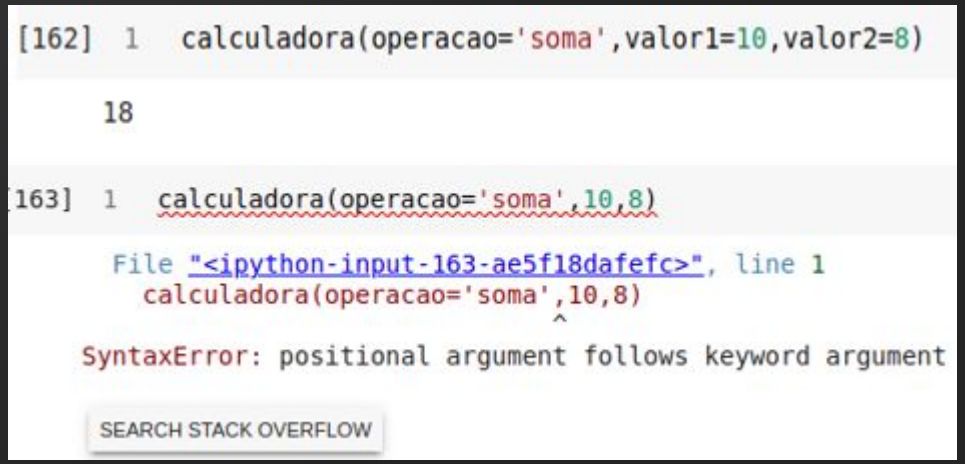
Funções Personalizadas

Podemos passar argumentos para a função de forma posicional, ou seja, o primeiro valor passado para a função ao ser chamada vai ser associada ao primeiro parâmetro e assim por diante.

```
[159]  1  calculadora(15,16,'soma')

       31
```

Funções Personalizadas

ATENÇÃO!!!! Quando usamos o nome do parâmetro ao chamar uma função, não é necessário se ater a ordem dos parâmetros! Ao passar argumentos de forma desordenada, não é possível utilizar a forma posicional logo em seguida.



Funções Personalizadas

É possível determinar valores padrão caso que só serão usados caso o usuário não passe nenhum outro argumento sobre o mesmo parâmetro.

```
[176]  1  def calculadora(valor1, valor2, operacao='soma'):
       2
       3    if operacao == 'soma':
       4      print(valor1+valor2)
       5    elif operacao == 'divisao':
       6      print(valor1/valor2)
       7    elif operacao == 'multiplicacao':
       8      print(valor1*valor2)
       9    elif operacao == 'subtracao':
      10      print(valor1-valor2)
```

```
[177]  1  calculadora(100,10)

       110
```

Funções Personalizadas

O resultado de uma função pode ser guardado em variáveis, porém para que isso ocorra precisamos expressar claramente o que será retornado pela função usando a palavra reservada 'return'. É bastante recomendado criar funções que tenham 'return'.

```
[178]  1  def minha_soma(valor1,valor2):
       2      valor1+valor2
       3
       4  def minha_soma_return(valor1,valor2):
       5      resultado = valor1 + valor2
       6
       7      return resultado
```

Funções Personalizadas

```
[180]  1  x = minha_soma(10,10)
       2  print(x)
       3  print(type(x))
```

```
None
<class 'NoneType'>
```

```
[181]  1  x = minha_soma_return(10,10)
       2  print(x)
       3  print(type(x))
```

```
20
<class 'int'>
```

Funções Personalizadas

Podemos reutilizar funções já montadas por outras pessoas.

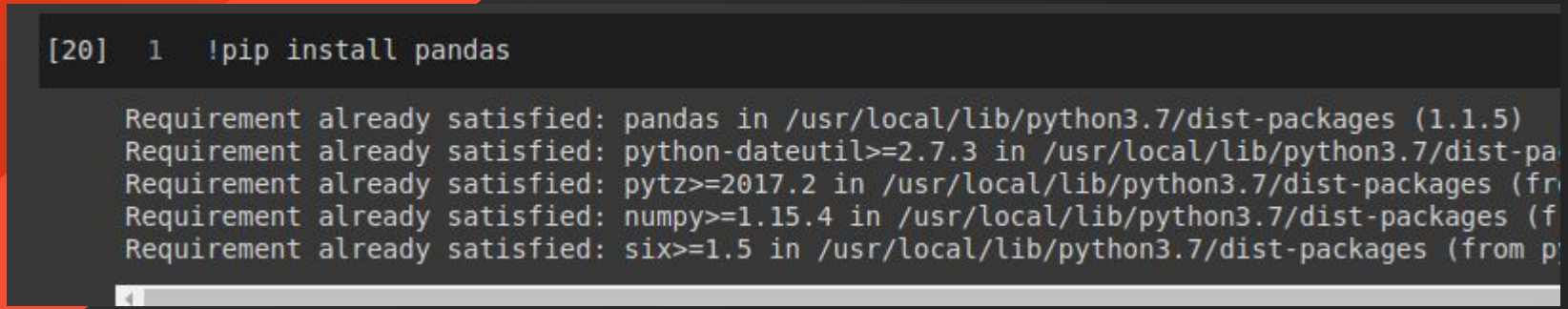
Um conjunto de funções criadas por outras pessoas e que importamos no python são chamados de módulos (ou pacote, ou bibliotecas).

O Python possui até então mais de 291 mil módulos que podemos utilizar.

Funções Personalizadas

Para serem usados os módulos primeiros precisam estar instalados e depois importados.

É possível instalar bibliotecas utilizando o gerenciador de pacotes “pip” da seguinte forma:



Funções Personalizadas

Importando o “pandas”

```
[22]  1  import pandas
      2
      3  pandas.read_csv('/content/sample_data/california_housing_train.csv')
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936
1	-114.47	34.40	19.0	7650.0	1901.0	1129.0	463.0	1.8200
2	-114.56	33.69	17.0	720.0	174.0	333.0	117.0	1.6509

Funções Personalizadas

Ao importar uma biblioteca podemos atribuir um apelido (ou um alias) a ela.

Isso é uma boa prática e existem alguns apelidos que são tão utilizados que já viraram sinônimo da própria biblioteca.

```
[23]  1  import pandas as pd
      2
      3  pd.read_csv('/content/sample_data/california_housing_train.csv')
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936
1	-114.47	34.40	19.0	7650.0	1901.0	1129.0	463.0	1.8200
2	-114.56	33.69	17.0	720.0	174.0	333.0	117.0	1.6509

Funções Personalizadas

Podemos ainda escolher exatamente a função que gostaríamos de importar.

Esse não é um método recomendado, pois causa confusão na leitura do código.

Podemos salvar nossas próprias funções em arquivos .py e importá-los em projetos.

```
[24]  1  from pandas import read_csv
      2
      3  read_csv('/content/sample_data/california_housing_train.csv')
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936

Pacote Pandas

Pacote Pandas

Pandas é um dos principais pacotes utilizados por todo cientista de dados.

Necessário para trabalhos de análise e manipulação de dados de diversas fontes.

Sua principal estrutura de dados, o DataFrame assemelha-se a uma tabela no excel.

A importação do pacote será feita com “import pandas as pd”.

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide
0	7.0	0.27	0.36	20.7	0.045	45.0
1	6.3	0.30	0.34	1.6	0.049	14.0
2	8.1	0.28	0.40	6.9	0.050	30.0
3	7.2	0.23	0.32	8.5	0.058	47.0
4	7.2	0.23	0.32	8.5	0.058	47.0

Data Frame

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]

Data Frame

Um DataFrame é uma estrutura de dados única do Pandas.

Ela é formada por um conjunto de séries.

```
1  ## criando um dataframe
2
3  df = pd.DataFrame(arr, ## dados
4                    columns=['idade','altura','tamanho do pe','salario','cor_do_olho','peso'], ##nome das colunas
5                    index=['nasser','joao','marcelo','maria','joana','raquel']) ## indices
6
7  df
```

	idade	altura	tamanho do pe	salario	cor_do_olho	peso
nasser	0	1	2	3	4	5
joao	6	7	8	9	10	11
marcelo	12	13	14	15	16	17
maria	18	19	20	21	22	23
joana	24	25	26	27	28	29
raquel	30	31	32	33	34	35

A série

A série é um conjunto de dados representado por um array numpy, a direta comparação será com listas.

pandas.Series

class pandas.**Series**(*data=None, index=None, dtype=None, name=None, copy=False, fastpath=False*) [source]

A série

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

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

```
1  ## criando um dataframe
2
3  df = pd.DataFrame(arr, ## dados
4                    columns=['idade','altura','tamanho do pe','salario','cor_do_olho','peso'], ##nome das colunas
5                    index=['nasser','joao','marcelo','maria','joana','raquel']) ## indices
6
7  df
```

	idade	altura	tamanho do pe	salario	cor_do_olho	peso
nasser	0	1	2	3	4	5
joao	6	7	8	9	10	11
marcelo	12	13	14	15	16	17
maria	18	19	20	21	22	23
joana	24	25	26	27	28	29
raquel	30	31	32	33	34	35

A série

A série é um conjunto de dados representado por um array numpy, a direta comparação será com listas.

```
[19]  1  ## acessando a série idade do dataframe acima
      2
      3  df.idade
```

```
nasser      0
joao        6
marcelo    12
maria      18
joana      24
raquel     30
Name: idade, dtype: int64
```

```
[20]  1  df['idade']
```

```
nasser      0
joao        6
marcelo    12
maria      18
joana      24
raquel     30
Name: idade, dtype: int64
```

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]

read_csv()

```
[23]  1  df = pd.read_csv('winequality-white.csv',sep=';')
      2  df.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides
0	7.0	0.27	0.36	20.7	0.045
1	6.3	0.30	0.34	1.6	0.049
2	8.1	0.28	0.40	6.9	0.050
3	7.2	0.23	0.32	8.5	0.058
4	7.2	0.23	0.32	8.5	0.058

.info() e .shape

A série é um conjunto de dados representado por um array numpy, a direta comparação será com listas.

```
[19]  1  ## acessando a série idade do dataframe acima
      2
      3  df.idade
```

```
nasser      0
joao        6
marcelo    12
maria      18
joana      24
raquel     30
Name: idade, dtype: int64
```

```
[20]  1  df['idade']
```

```
nasser      0
joao        6
marcelo    12
maria      18
joana      24
raquel     30
Name: idade, dtype: int64
```

.info() e .shape

pandas.DataFrame.info

`DataFrame.info(verbose=None, buf=None, max_cols=None, memory_usage=None, show_counts=None, null_counts=None)` [source]

Print a concise summary of a DataFrame.

This method prints information about a DataFrame including the index dtype and columns, non-null values and memory usage.

pandas.DataFrame.shape

property `DataFrame.shape`

Return a tuple representing the dimensionality of the DataFrame.

.info() e .shape

```
[50]  1  df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4898 entries, 0 to 4897
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         4898 non-null   float64
 1   volatile acidity      4898 non-null   float64
 2   citric acid           4898 non-null   float64
 3   residual sugar        4898 non-null   float64
 4   chlorides             4898 non-null   float64
 5   free sulfur dioxide   4898 non-null   float64
 6   total sulfur dioxide  4898 non-null   float64
 7   density               4898 non-null   float64
 8   pH                    4898 non-null   float64
 9   sulphates             4898 non-null   float64
 10  alcohol               4898 non-null   float64
 11  quality               4898 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 459.3 KB
```

.info() e .shape

```
[52]  1  ## shape
      2
      3  df.shape
```

```
(4898, 12)
```

4898 linhas
12 colunas

Indexing slicing

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

```
[56]  1  df['alcohol']

0        8.8
1        9.5
2       10.1
3        9.9
4        9.9
       ...
4893    11.2
4894     9.6
4895     9.4
4896    12.8
4897    11.8
Name: alcohol, Length: 4898, dtype: float64
```

Indexing slicing

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

```
[57]  1  df.alcohol
```

```
0        8.8
1        9.5
2       10.1
3        9.9
4        9.9
       ...
4893    11.2
4894     9.6
4895     9.4
4896    12.8
4897    11.8
Name: alcohol, Length: 4898, dtype: float64
```

Indexing slicing

pandas.DataFrame.loc

property `DataFrame.loc`

Access a group of rows and columns by label(s) or a boolean array.

`.loc[]` is primarily label based, but may also be used with a boolean array.

Indexing slicing

Todas as linhas " : "
Coluna com o nome "quality"

```
[60]  1  df.loc[:,'quality']
```

```
0       6
1       6
2       6
3       6
4       6
       ..
4893    6
4894    5
4895    6
4896    7
4897    6
Name: quality, Length: 4898, dtype: int64
```

Indexing slicing

```
6  df_nomes
```

	idade	altura	tamanho do pe	salario	cor_do_olho	peso
nasser	0	1	2	3	4	5
joao	6	7	8	9	10	11
marcelo	12	13	14	15	16	17
maria	18	19	20	21	22	23
joana	24	25	26	27	28	29
raquel	30	31	32	33	34	35

Indexing slicing

ATENÇÃO!!!
.loc é baseado nos nomes das linhas e colunas!!!

Se o nome da linha for um número ele também pode ser utilizado!

```
1  df_nomes.loc['joao','salario']
```

```
9
```

Indexing slicing

ATENÇÃO!!!
.loc é baseado nos nomes das linhas e colunas!!!

Se o nome da linha for um número ele também pode ser utilizado!

```
1  df
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides
0	7.0	0.27	0.36	20.7	0.045
1	6.3	0.30	0.34	1.6	0.049
2	8.1	0.28	0.40	6.9	0.050
3	7.2	0.23	0.32	8.5	0.058
4	7.2	0.23	0.32	8.5	0.058
...	...	...	...	...	...
4893	6.2	0.21	0.29	1.6	0.039
4894	6.6	0.32	0.36	8.0	0.047
4895	6.5	0.24	0.19	1.2	0.041
4896	5.5	0.29	0.30	1.1	0.022
4897	6.0	0.21	0.38	0.8	0.020

Indexing slicing

ATENÇÃO!!!
Apesar de ter usado o número 4 como índice de linha isso não significa que ela, necessariamente é a quarta linha!!!

Isso significa que 4 é o nome (label) dela.

```
1   df.loc[4,'volatile acidity']
```

```
0.23
```

Indexing slicing

pandas.DataFrame.iloc

property `DataFrame.iloc`

Purely integer-location based indexing for selection by position.

`.iloc[]` is primarily integer position based (from `0` to `length-1` of the axis), but may also be used with a boolean array.

Indexing slicing

Todas as linhas " : "
Coluna com o índice 11 ("quality")

```
1  ## usando iloc para encontrar a coluna qualidade
2
3  df.iloc[:,11]
```

```
0       6
1       6
2       6
3       6
4       6
       ..
4893    6
4894    5
4895    6
4896    7
4897    6
Name: quality, Length: 4898, dtype: int64
```

Indexing slicing

```
[71]  1  df_nomes.iloc[1,3]
```

```
9
```

```
6  df_nomes
```

	idade	altura	tamanho do pe	salario	cor_do_olho	peso
nasser	0	1	2	3	4	5
joao	6	7	8	9	10	11
marcelo	12	13	14	15	16	17
maria	18	19	20	21	22	23
joana	24	25	26	27	28	29
raquel	30	31	32	33	34	35

Indexing slicing

```
[76]  1  df_nomes.loc['joao':'joana','altura':'cor_do_olho']
```

	altura	tamanho do pe	salario	cor_do_olho
joao	7	8	9	10
marcelo	13	14	15	16
maria	19	20	21	22
joana	25	26	27	28

Indexing slicing

```
1  df.iloc[50:55,2:6]
```

	citric acid	residual sugar	chlorides	free sulfur dioxide
50	0.31	1.6	0.062	31.0
51	0.29	1.1	0.068	39.0
52	0.33	1.1	0.057	21.0
53	0.35	1.0	0.045	39.0
54	0.59	0.9	0.147	38.0

Indexing slicing

```
1  df.loc[:,['residual sugar','quality']]
```

	residual sugar	quality
0	20.7	6
1	1.6	6
2	6.9	6
3	8.5	6
4	8.5	6
...	...	...
4893	1.6	6
4894	8.0	5
4895	1.2	6
4896	1.1	7
4897	0.8	6

4898 rows × 2 columns

Indexing slicing

```
1  df[['residual sugar','pH','citric acid']]
```

	residual sugar	pH	citric acid
0	20.7	3.00	0.36
1	1.6	3.30	0.34
2	6.9	3.26	0.40
3	8.5	3.19	0.32
4	8.5	3.19	0.32
...	...	...	...
4893	1.6	3.27	0.29
4894	8.0	3.15	0.36
4895	1.2	2.99	0.19
4896	1.1	3.34	0.30
4897	0.8	3.26	0.38

4898 rows × 3 columns

df[[<nome_da_coluna>,<nome_da_coluna>]]

```
df[df.quality == 6]
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6

Indexing slicing

```
1  df[(df.quality == 6 )&(df['residual sugar'] > 20)]
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.270	0.36	20.70	0.045	45.0	170.0	1.00100	3.00	0.45	8.8	6
7	7.0	0.270	0.36	20.70	0.045	45.0	170.0	1.00100	3.00	0.45	8.8	6
444	6.9	0.240	0.36	20.80	0.031	40.0	139.0	0.99750	3.20	0.33	11.0	6
1653	7.9	0.330	0.28	31.60	0.053	35.0	176.0	1.01030	3.15	0.38	8.8	6
1663	7.9	0.330	0.28	31.60	0.053	35.0	176.0	1.01030	3.15	0.38	8.8	6
2781	7.8	0.965	0.60	65.80	0.074	8.0	160.0	1.03898	3.39	0.69	11.7	6
3619	6.8	0.450	0.28	26.05	0.031	27.0	122.0	1.00295	3.06	0.42	10.6	6
3623	6.8	0.450	0.28	26.05	0.031	27.0	122.0	1.00295	3.06	0.42	10.6	6
3730	6.2	0.220	0.20	20.80	0.035	58.0	184.0	1.00022	3.11	0.53	9.0	6
4107	6.8	0.300	0.26	20.30	0.037	45.0	150.0	0.99727	3.04	0.38	12.3	6

Indexing slicing

```
1 df.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

```
[35] 1 df.tail()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
4893	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3.27	0.50	11.2	6
4894	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3.15	0.46	9.6	5
4895	6.5	0.24	0.19	1.2	0.041	30.0	111.0	0.99254	2.99	0.46	9.4	6
4896	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3.34	0.38	12.8	7
4897	6.0	0.21	0.38	0.8	0.020	22.0	98.0	0.98941	3.26	0.32	11.8	6

Indexing slicing

```
1  df.sample(7)
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
895	5.6	0.19	0.39	1.10	0.043	17.0	67.0	0.99180	3.23	0.53	10.3	6
2274	7.3	0.19	0.25	1.40	0.051	41.0	107.0	0.99382	3.53	0.66	10.5	7
2790	7.1	0.20	0.31	6.85	0.053	32.0	211.0	0.99587	3.31	0.59	10.4	6
4645	5.0	0.24	0.34	1.10	0.034	49.0	158.0	0.98774	3.32	0.32	13.1	7
4335	7.3	0.19	0.27	13.90	0.057	45.0	155.0	0.99807	2.94	0.41	8.8	8
3395	6.6	0.26	0.46	7.80	0.047	48.0	186.0	0.99580	3.20	0.54	9.1	5
20	6.2	0.66	0.48	1.20	0.029	29.0	75.0	0.98920	3.33	0.39	12.8	8

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]

to_datetime()

```
new_df = pd.DataFrame([['2021-11-02'],
                       ['2022-03-15'],
                       ['2022-05-16'],
                       ['2021-07-01'],
                       ['2008-02-03'],
                       ['2015-09-03'],
                       ['2013-12-03'],
                       ['2022-06-03'],
                       ['2008-08-03'],
                       ['2002-09-03']])

new_df.columns = ['dt_carga']
```

```
new_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 1 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   dt_carga  10 non-null     object
dtypes: object(1)
memory usage: 208.0+ bytes
```

to_datetime()

```
1   pd.to_datetime(new_df['dt_carga'])
```

```
0   2021-11-02
1   2022-03-15
2   2022-05-16
3   2021-07-01
4   2008-02-03
5   2015-09-03
6   2013-12-03
7   2022-06-03
8   2008-08-03
9   2002-09-03
Name: dt_carga, dtype: datetime64[ns]
```

to_datetime()

```
new_df['dt_carga'] = pd.to_datetime(new_df['dt_carga'])
```

```
1  new_df
```

	dt_carga
0	2021-11-02
1	2022-03-15
2	2022-05-16
3	2021-07-01
4	2008-02-03
5	2015-09-03
6	2013-12-03
7	2022-06-03
8	2008-08-03
9	2002-09-03

to_datetime()

to_datetime()

```
 1   new_df.dt_carga.dt.day

0     2
1    15
2    16
3     1
4     3
5     3
6     3
7     3
8     3
9     3
Name: dt_carga, dtype: int64
```

to_datetime()

pandas.Series.dt.month

`Series.dt.month`

The month as January=1, December=12.

to_datetime()

```
  1   new_df.dt_carga.dt.month
```

```
0    11
1     3
2     5
3     7
4     2
5     9
6    12
7     6
8     8
9     9
Name: dt_carga, dtype: int64
```

to_datetime()

pandas.Series.dt.year

`Series.dt.year`

The year of the datetime.

to_datetime()

```
 1   new_df.dt_carga.dt.year
```

```
0    2021
1    2022
2    2022
3    2021
4    2008
5    2015
6    2013
7    2022
8    2008
9    2002
Name: dt_carga, dtype: int64
```

to_datetime()

pandas.Series.dt.dayofweek

`Series.dt.dayofweek`

The day of the week with Monday=0, Sunday=6.

Return the day of the week. It is assumed the week starts on Monday, which is denoted by 0 and ends on Sunday which is denoted by 6. This method is available on both Series with datetime values (using the *dt* accessor) or DatetimeIndex.

Returns: **Series or Index**

Containing integers indicating the day number.

to_datetime()

```
  1   new_df.dt_carga.dt.dayofweek
0     1
1     1
2     0
3     3
4     6
5     3
6     1
7     4
8     6
9     1
Name: dt_carga, dtype: int64
```

to_datetime()

pandas.Series.dt.time
pandas.Series.dt.timetz
pandas.Series.dt.year
pandas.Series.dt.month
pandas.Series.dt.day
pandas.Series.dt.hour
pandas.Series.dt.minute
pandas.Series.dt.second
pandas.Series.dt.microsecond
pandas.Series.dt.nanosecond
pandas.Series.dt.week

pandas.Series.dt.is_leap_year
pandas.Series.dt.daysinmonth
pandas.Series.dt.days_in_month
pandas.Series.dt.tz
pandas.Series.dt.freq
pandas.Series.dt.to_period
pandas.Series.dt.to_pydatetime
pandas.Series.dt.tz_localize
pandas.Series.dt.tz_convert
pandas.Series.dt.normalize
pandas.Series.dt.strftime
pandas.Series.dt.round
pandas.Series.dt.floor
pandas.Series.dt.ceil
pandas.Series.dt.month_name

pandas.Series.dt.month_name
pandas.Series.dt.day_name
pandas.Series.dt.qyear
pandas.Series.dt.start_time
pandas.Series.dt.end_time
pandas.Series.dt.days
pandas.Series.dt.seconds
pandas.Series.dt.microseconds
pandas.Series.dt.nanoseconds
pandas.Series.dt.components
pandas.Series.dt.to_pytimedelta
pandas.Series.dt.total_seconds

to_datetime()

pandas.DataFrame.describe

`DataFrame.describe(percentiles=None, include=None, exclude=None, datetime_is_numeric=False)` [source]

Generate descriptive statistics.

Descriptive statistics include those that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding `NaN` values.

Analyzes both numeric and object series, as well as `DataFrame` column sets of mixed data types. The output will vary depending on what is provided. Refer to the notes below for more detail.

.describe()

```
1  df
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44	10.1	6

.describe()

```
1  df.describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide
count	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000
mean	6.854788	0.278241	0.334192	6.391415	0.045772	35.308085	138.360657
std	0.843868	0.100795	0.121020	5.072058	0.021848	17.007137	42.498065
min	3.800000	0.080000	0.000000	0.600000	0.009000	2.000000	9.000000
25%	6.300000	0.210000	0.270000	1.700000	0.036000	23.000000	108.000000
50%	6.800000	0.260000	0.320000	5.200000	0.043000	34.000000	134.000000
75%	7.300000	0.320000	0.390000	9.900000	0.050000	46.000000	167.000000
max	14.200000	1.100000	1.660000	65.800000	0.346000	289.000000	440.000000

.describe()

```
1  df.describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density
count	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000
mean	6.854788	0.278241	0.334192	6.391415	0.045772	35.308085	138.360657	0.994027
std	0.843868	0.100795	0.121020	5.072058	0.021848	17.007137	42.498065	0.002991
min	3.800000	0.080000	0.000000	0.600000	0.009000	2.000000	9.000000	0.987110
25%	6.300000	0.210000	0.270000	1.700000	0.036000	23.000000	108.000000	0.991723
50%	6.800000	0.260000	0.320000	5.200000	0.043000	34.000000	134.000000	0.993740
75%	7.300000	0.320000	0.390000	9.900000	0.050000	46.000000	167.000000	0.996100
max	14.200000	1.100000	1.660000	65.800000	0.346000	289.000000	440.000000	1.038980

Primeiro Quartil

Segundo Quartil ou Mediana

Terceiro Quartil

.describe()

```
1  print(df['pH'].count())
2  print(df['pH'].std())
3  print(df['pH'].var())
4  print(df['pH'].quantile(0.25))
5  print(df['pH'].quantile(0.5))
6  print(df['pH'].quantile(0.75))
7  print(df['pH'].quantile(0.95))
```

```
4898
0.1510005996150667
0.02280118108410968
3.09
3.18
3.28
3.46
```

.corr()
correlação

pandas.DataFrame.corr

`DataFrame.corr(self, method='pearson', min_periods=1) → 'DataFrame'` [source]

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

.corr()
correlação

```
1  df[['free sulfur dioxide','chlorides','citric acid']]
```

	free sulfur dioxide	chlorides	citric acid
0	45.0	0.045	0.36
1	14.0	0.049	0.34
2	30.0	0.050	0.40
3	47.0	0.058	0.32
4	47.0	0.058	0.32
...	...	...	...
4893	24.0	0.039	0.29
4894	57.0	0.047	0.36
4895	30.0	0.041	0.19
4896	20.0	0.022	0.30
4897	22.0	0.020	0.38

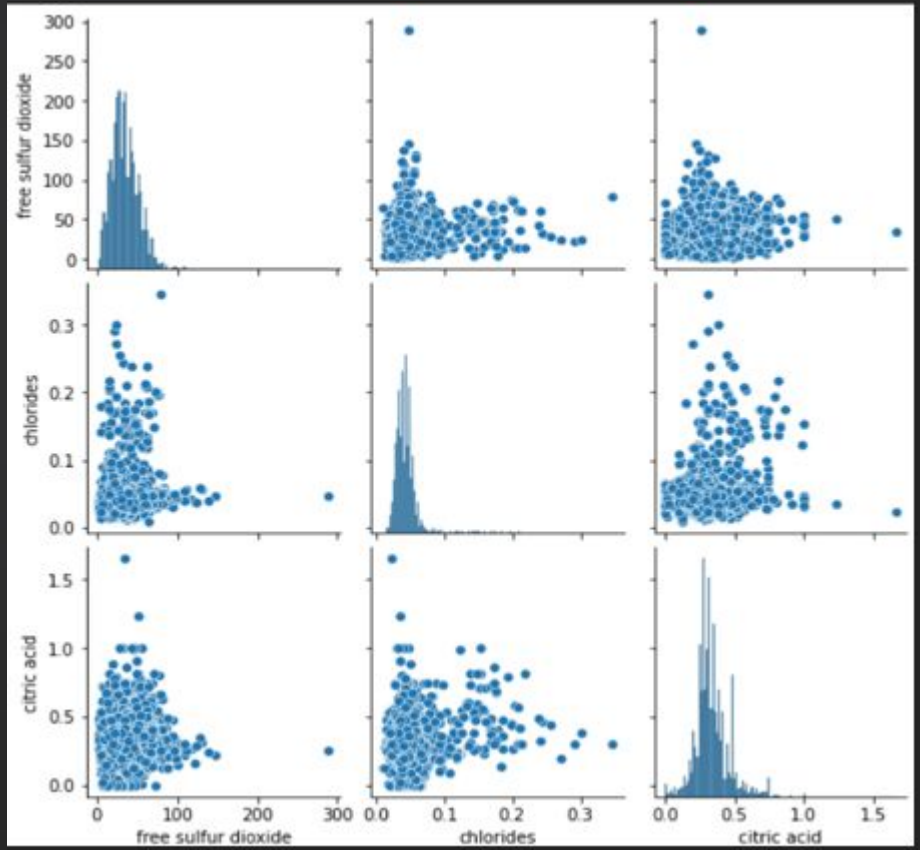
4898 rows × 3 columns

.corr()
correlação

```
1  df[['free sulfur dioxide','chlorides','citric acid']].corr()
```

	free sulfur dioxide	chlorides	citric acid
free sulfur dioxide	1.000000	0.101392	0.094077
chlorides	0.101392	1.000000	0.114364
citric acid	0.094077	0.114364	1.000000

.corr()
correlação



.corr()
correlação

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**

.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.

.mode()

```
1  df.mode()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	6.8	0.28	0.3	1.2	0.044	29.0	111.0	0.992	3.14	0.5	9.4	6

.unique()
.nunique()

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.

.unique()
.nunique()

```
[79]  1  df.quality.unique()
```

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

```
[80]  1  df.quality.nunique()
```

```
7
```

.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.

.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.

.value_counts()

```
1   df.quality.value_counts()
```

```
6    2198
5    1457
7     880
8     175
4     163
3      20
9       5
Name: quality, dtype: int64
```

.value_counts()

```
1   df.quality.value_counts() / df.shape[0]
```

```
6    0.448755
5    0.297468
7    0.179665
8    0.035729
4    0.033279
3    0.004083
9    0.001021
Name: quality, dtype: float64
```

.value_counts()

```
1   df.quality.value_counts(normalize=True)
```

```
6    0.448755
5    0.297468
7    0.179665
8    0.035729
4    0.033279
3    0.004083
9    0.001021
Name: quality, dtype: float64
```

Transformações
.apply()

pandas.DataFrame.apply

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

Apply a function along an axis of the DataFrame.

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

Parameters: **func** : ***function***

Function to apply to each column or row.

Transformações .apply()

```
[61]  1  def binning(value):
      2      if value <= 3:
      3          return 'baixa_qualidade'
      4      elif value > 3 and value <=7:
      5          return 'media_qualidade'
      6      elif value > 7:
      7          return 'alta_qualidade'
```

```
[65]  1  df.quality.apply(binning)
```

```
0       media_qualidade
1       media_qualidade
2       media_qualidade
3       media_qualidade
4       media_qualidade
              ...
4893    media_qualidade
4894    media_qualidade
4895    media_qualidade
4896    media_qualidade
4897    media_qualidade
Name: quality, Length: 4898, dtype: object
```

Transformações .apply()

```
1  def normalizar_qualidade(value,maximo):
2      return value/maximo
3
4  df.quality.apply(normalizar_qualidade,args=(9,))
```

```
0       0.666667
1       0.666667
2       0.666667
3       0.666667
4       0.666667
          ...
4893    0.666667
4894    0.555556
4895    0.666667
4896    0.777778
4897    0.666667
Name: quality, Length: 4898, dtype: float64
```

Transformações .apply()

```
[80]  1  def print_value(value):
      2      print(type(value))
      3
      4  df[['free sulfur dioxide','total sulfur dioxide']].apply(print_value)
```

```
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
free sulfur dioxide     None
total sulfur dioxide    None
dtype: object
```

Obrigada

/mesttra mesttra.com