

Pandas

É uma biblioteca escrita sobre o Numpy

Permite rápida visualização e limpeza de dados

Semelhante ao Excel

▼ Instalação:

Conda install pandas - Anaconda

pip install pandas - Vscode e afins

▼ Label:

Label é a construção de algo próximo a uma biblioteca do python base.

Com ele, é possível definir uma informação que tem ligação direta a outra:

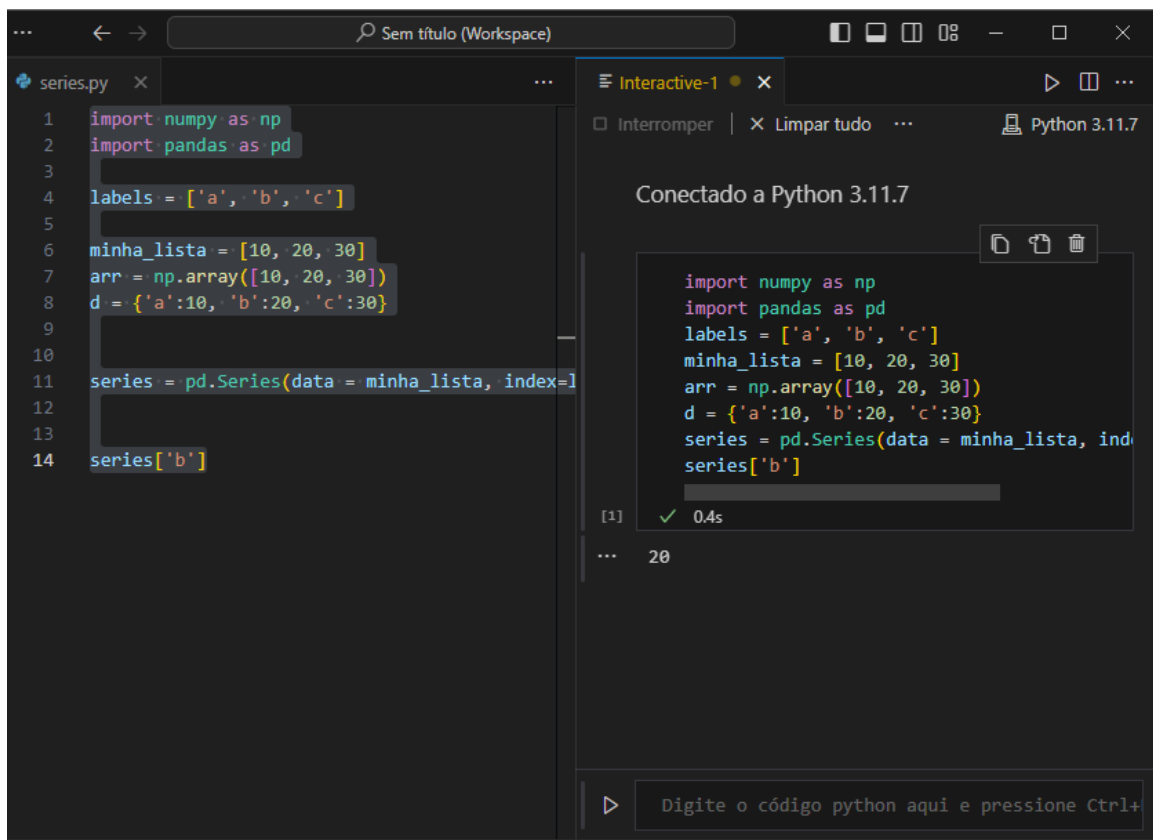
```
import numpy as np
import pandas as pd

labels = ['a', 'b', 'c']

minha_lista = [10, 20, 30]
arr = np.array([10, 20, 30])
d = {'a':10, 'b':20, 'c':30}

series = pd.Series(data = minha_lista, index=labels)

series['b']
```



Uma series não bloqueia para que dados sejam string, funções, contanto que tenha algum valor válido a mesma é possível indicar:

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

The screenshot shows a Jupyter Notebook with two main panels. The left panel is a code editor with a file named 'series.py'. It contains the following Python code:

```
1 import numpy as np
2 import pandas as pd
3
4 labels = ['a', 'b', 'c']
5
6 minha_lista = [10, 20, 30]
7 arr = np.array([10, 20, 30])
8 d = {'a':10, 'b':20, 'c':30}
9
10
11 series = pd.Series(data = minha_lista, index=labels)
12
13
14 series['b']
15
16
17 pd.Series([sum, print, len])
18
```

The right panel is an interactive console titled 'Interactive-1' connected to 'Python 3.11.7'. It shows the execution of the last line of code from the left panel: `pd.Series([sum, print, len])`. The output is a Pandas Series with three elements:

0	<built-in function sum>
1	<built-in function print>
2	<built-in function len>

The dtype is listed as 'object'. At the bottom of the right panel, there is a prompt: 'Digite o código python aqui e pressione Ctrl+'.

Índice será referência aos dados.

Declarando uma série:

```
ser1 = pd.Series([1,2,3,4], index=['EUA', 'ALEMANHA', 'URSS'])
ser1
```

```
1 import numpy as np
2 import pandas as pd
3
4 labels = ['a', 'b', 'c']
5
6 minha_lista = [10, 20, 30]
7 arr = np.array([10, 20, 30])
8 d = {'a':10, 'b':20, 'c':30}
9
10
11 series = pd.Series(data = minha_lista, index=labels)
12
13
14 series['b']
15
16
17 ser1 = pd.Series([1,2,3,4], index=['EUA', 'ALEMANHA', 'URSS', 'JAPÃO'])
18 ser1
19
20 ser2 = pd.Series([1,2,3,4], index=['EUA', 'ALEMANHA', 'URSS', 'JAPÃO'])
21 ser2
22 ser1 + ser2
```

Interromper | ✕ Limpar tudo | Python 3.11.7

✓ ser1 = pd.Series([1,2,3,4], index= ...

✓ ser1 ...

...	EUA	1
	ALEMANHA	2
	URSS	3
	JAPÃO	4
	dtype:	int64

▶ Digite o código python aqui e pressione Ctrl+

Declarando outra série com unica diferença sem ao invés de URSS, ser ITALIA:

```
ser2 = pd.Series([1,2,3,4], index=['EUA', 'ALEMANHA', 'ITALIA'])
ser2
```

The screenshot shows a Jupyter Notebook with a file named 'series.py'. The code in the script is as follows:

```
1 import numpy as np
2 import pandas as pd
3
4 labels = ['a', 'b', 'c']
5
6 minha_lista = [10, 20, 30]
7 arr = np.array([10, 20, 30])
8 d = {'a':10, 'b':20, 'c':30}
9
10
11 series = pd.Series(data = minha_lista, index=labels)
12
13
14 series['b']
15
16
17 ser1 = pd.Series([1,2,3,4], index=['EUA', 'ALEMANHA', 'ITALIA', 'JAPÃO'])
18 ser1
19
20 ser2 = pd.Series([1,2,3,4], index=['EUA', 'ALEMANHA', 'ITALIA', 'JAPÃO'])
21 ser2
22 ser1 + ser2
```

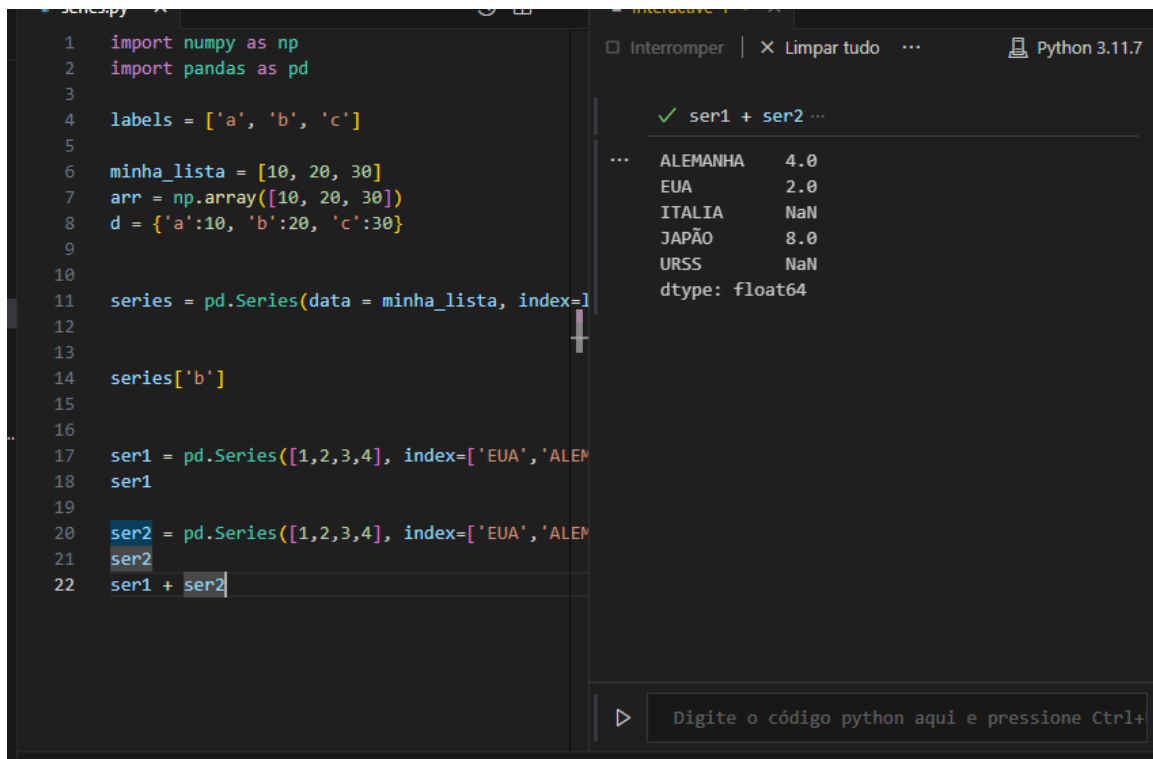
The output of the execution is shown in the right-hand pane. It displays the creation of 'ser2' and its contents:

```
✓ ser2 = pd.Series([1,2,3,4], index=labels)
...
EUA      1
ALEMANHA 2
ITALIA   3
JAPÃO    4
dtype: int64
```

At the bottom of the interface, there is a prompt: "Digite o código python aqui e pressione Ctrl+Enter".

Ao tentar somar os 2, por conta de não haver uma referencia definida aos 2, não existe meio de realizar uma soma:

```
ser1 + ser2
```



```
1 import numpy as np
2 import pandas as pd
3
4 labels = ['a', 'b', 'c']
5
6 minha_lista = [10, 20, 30]
7 arr = np.array([10, 20, 30])
8 d = {'a':10, 'b':20, 'c':30}
9
10 series = pd.Series(data = minha_lista, index=labels)
11
12
13 series['b']
14
15
16
17 ser1 = pd.Series([1,2,3,4], index=['EUA', 'ALEMANHA', 'JAPÃO', 'URSS'])
18 ser1
19
20 ser2 = pd.Series([1,2,3,4], index=['EUA', 'ALEMANHA', 'JAPÃO', 'URSS'])
21 ser2
22 ser1 + ser2
```

ser1 + ser2 ...

...	
ALEMANHA	4.0
EUA	2.0
ITALIA	NaN
JAPÃO	8.0
URSS	NaN
dtype:	float64

▶ Digite o código python aqui e pressione Ctrl+

Retornando assim NaN, por conta da referencia entre os 2 ser invalida.

Ideia do Series, realizar operação através dos indices, caso não haja um indice válido, não será possível realizar.

▼ Dataframes:

Conjunto de series.

▼ Random.seed()

Numeros gerados através do random serão baseados em uma seed padrão a todos os outros computadores

▼ pd.dataframe(1)

Criação de um dataframe

```
pd.dataframe(np.random.randn(5, 4), index='A B C D E').s
```

Puxar uma coluna:

```
df['W']
```

Puxar uma ou mais:

```
df[['W', 'Z']]
```

também é possível utilizando o `.` (semelhante ao sql):

```
df.w
```

Criar uma nova informação:

```
df['new'] = df['W'] + df['X']
```

Dar baixa em uma coluna:

```
df.drop('new', axis=1)
```

Porém isto faz o new ainda permanecer. Para dar baixa de fato, utilizar o `inplace`:

```
df.drop('new', axis=1, inplace = True)
```

para encontrar valores:

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

encontrar uma lista de valores:

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

Encontrando elementos baseados no índice:

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

▼ pd.dataframe(2)

Ao criar um data frame, e realizar uma operação lógica, é possível identificar valores assim como em arrays, diferença é ele não gera um

dataframe redimensionado, gerando uma série de NaN:

```
import numpy as np
import pandas as pd

np.random.seed(101)

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

df

bol = df > 0

df[bol]
```

The screenshot shows a Jupyter Notebook interface with a code editor on the left and an interactive console on the right. The code in the editor is as follows:

```
1 import numpy as np
2 import pandas as pd
3
4
5 np.random.seed(101)
6
7 df = pd.DataFrame(
8     np.random.randn(5, 4), index="A B C D E".split(), columns=["W", "X", "Y", "Z"]
9 )
10
11 df
12
13 bol = df > 0
14
15 df[bol]
```

The interactive console shows the following output:

✓ df > 0 ...

	W	X	Y	Z
A	True	True	True	True
B	True	False	False	True
C	False	True	True	False
D	True	False	False	True
E	True	True	True	True

✓ bol = df > 0 ...

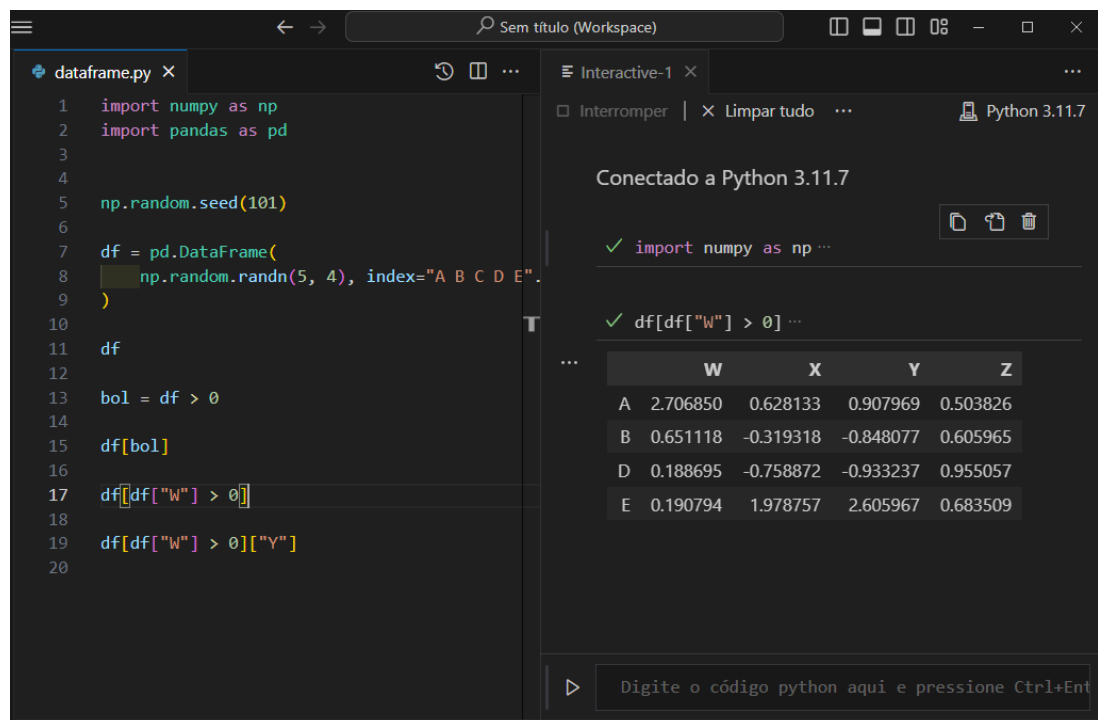
✓ df[bol] ...

	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

At the bottom of the console, there is a prompt: "Digite o código python aqui e pressione Ctrl+En".

É possível também, utilizando uma operação lógica, retornar um exato índice que terá necessidade:

```
df[df['W']>0
# retornar um dataframe, do qual a coluna W do data frame
df[df['W']>0[Y]
# Por fim retornar os valores que sobraram na coluna Y
```



The screenshot shows a Jupyter Notebook interface with a code editor on the left and an interactive console on the right. The code editor contains a Python script that creates a DataFrame with 5 rows (A, B, C, D, E) and 4 columns (W, X, Y, Z). The script then filters the DataFrame to only include rows where the value in column 'W' is greater than 0, and finally returns the values in column 'Y' for those rows.

```
1 import numpy as np
2 import pandas as pd
3
4
5 np.random.seed(101)
6
7 df = pd.DataFrame(
8     np.random.randn(5, 4), index="A B C D E".
9 )
10
11 df
12
13 bol = df > 0
14
15 df[bol]
16
17 df[df["W"] > 0]
18
19 df[df["W"] > 0]["Y"]
20
```

The interactive console shows the output of the code, displaying a table with 5 rows (A, B, C, D, E) and 4 columns (W, X, Y, Z). The values in column 'W' are: A: 2.706850, B: 0.651118, C: -0.319318, D: 0.188695, E: 0.190794. The values in column 'Y' are: A: 0.907969, B: -0.848077, C: -0.933237, D: 0.955057, E: 2.605967.

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

aqui ele retirou linha C, porém retornou valores da B e D que são menores que 0, afinal, condição esta sendo aplicado somente a uma linha na coluna W, utilizando na coluna Y como exemplo:

```
1 import numpy as np
2 import pandas as pd
3
4
5 np.random.seed(101)
6
7 df = pd.DataFrame(
8     np.random.randn(5, 4), index="A B C D E"
9 )
10
11 df
12
13 bol = df > 0
14
15 df[bol]
16
17 df[df["Y"] > 0]
18
19 df[df["W"] > 0]["Y"]
20
```

Conectado a Python 3.11.7

✓ import numpy as np ...

✓ df[df["Y"] > 0] ...

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
C	-2.018168	0.740122	0.528813	-0.589001
E	0.190794	1.978757	2.605967	0.683509

Digite o código python aqui e pressione Ctrl+Ent

Retorno se faz inferior, retirando não os valores da C, mas da B e D.

```
1 import numpy as np
2 import pandas as pd
3
4
5 np.random.seed(101)
6
7 df = pd.DataFrame(
8     np.random.randn(5, 4), index="A B C D E"
9 )
10
11 df
12
13 bol = df > 0
14
15 df[bol]
16
17 df[df["W"] > 0]
18
19 df[df["W"] > 0]["Y"]
20
```

Conectado a Python 3.11.7

✓ import numpy as np ...

✓ df[df["W"] > 0]["Y"] ...

	Y
A	0.907969
B	-0.848077
D	-0.933237
E	2.605967

Name: Y, dtype: float64

Ao utilizar por fim o filtro de índice de coluna, o retorno serão os valores contidos na coluna Y, pegando a definição do dataframe com filtro onde os valores de W são maiores que 0

- Comparação com duas séries:

Quando se faz necessário utilizar um operador lógico, que reverta 2 valores booleanos e seja necessário dar resultado entre os 2 em uma série, é utilizando o and na sua forma "&":

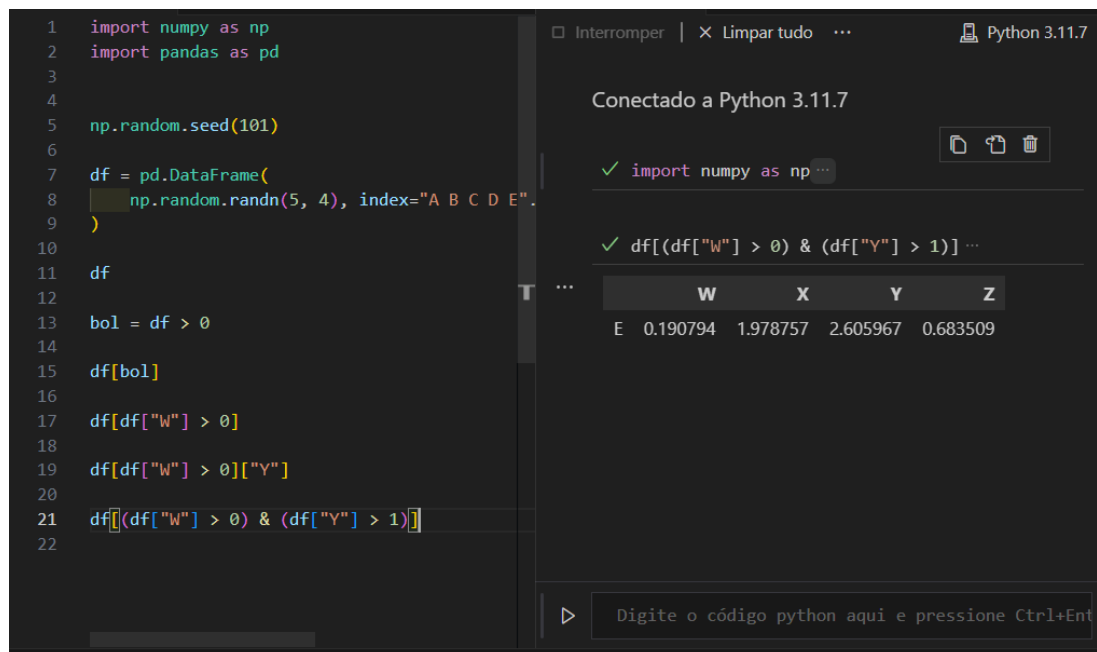
```
df[(df['W']>0) and (df['Y']>1)]
```

Ao tentar realizar do modo acima, ele gerará crítica de "The truth value of a Series is ambiguous. Use a.empty, a.bool(), a.item()..."

Basicamente é por conta do operador and, não poder ser utilizado em série, sendo necessário a utilização do &:

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

Deste modo retorno será o seguinte:



The screenshot shows a Jupyter Notebook interface with a code editor on the left and a console/output area on the right. The code in the editor defines a DataFrame with 5 rows and 4 columns (W, X, Y, Z) and performs a logical operation to filter rows where both W is greater than 0 and Y is greater than 1. The console shows the successful execution of the code and the resulting DataFrame.

```
1 import numpy as np
2 import pandas as pd
3
4
5 np.random.seed(101)
6
7 df = pd.DataFrame(
8     np.random.randn(5, 4), index="A B C D E".
9 )
10
11 df
12
13 bol = df > 0
14
15 df[bol]
16
17 df[df["W"] > 0]
18
19 df[df["W"] > 0]["Y"]
20
21 df[(df["W"] > 0) & (df["Y"] > 1)]
22
```

Conectado a Python 3.11.7

✓ import numpy as np ...

✓ df[(df["W"] > 0) & (df["Y"] > 1)] ...

	W	X	Y	Z
E	0.190794	1.978757	2.605967	0.683509

Digite o código python aqui e pressione Ctrl+Ent

The screenshot shows a Jupyter Notebook with a code editor on the left and an interactive console on the right. The code in the editor creates a DataFrame with 5 rows (A-E) and 4 columns (W, X, Y, Z). The interactive console shows the execution of the code, including the creation of the DataFrame and the application of two filters: `df[df["W"] > 0]` and `df[(df["W"] > 0) & (df["Y"] > 1)]`. The final result is a DataFrame with 5 rows, where the values for columns W, X, Y, and Z are displayed. The values for W and Y are highlighted in red boxes, and the values for X and Z are highlighted in yellow boxes.

	W	X	Y	Z
E	0.190794	1.978757	2.605967	0.683509

Em vermelho os 2 filtros.

Já para operador OU, utilizar |

- `df.reset_index`

Através deste comando, é possível reiniciar o seu dataframe:

```
dataframe.py x
> np.random.seed(101)
6
7 df = pd.DataFrame(
8     np.random.randn(5, 4), index="A B C D E".split(), co
9 )
10
11 df
12
13 bol = df > 0
14
15 df[bol]
16
17 df[df["W"] > 0]
18
19 df[df["W"] > 0]["Y"]
20
21 df[(df["W"] > 0) & (df["Y"] > 1)]
22
23
24 df
25 df.reset_index()
26
```

Interactive-1 x

Interromper | x Limpar tudo | Reiniciar | Python 3.11.7

D	0.188695	NaN	NaN	0.955057
E	0.190794	1.978757	2.605967	0.683509

df ...

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

df.reset_index() ...

	index	W	X	Y	Z
0	A	2.706850	0.628133	0.907969	0.503826
1	B	0.651118	-0.319318	-0.848077	0.605965
2	C	-2.018168	0.740122	0.528813	-0.589001
3	D	0.188695	-0.758872	-0.933237	0.955057
4	E	0.190794	1.978757	2.605967	0.683509

▶ Digite o código python aqui e pressione Ctrl+Enter para exec

Para que comando seja definitivamente executado, adicionar a condição `inplace=True`. Quando é adicionado esta condição, ele definirá por definitivo.

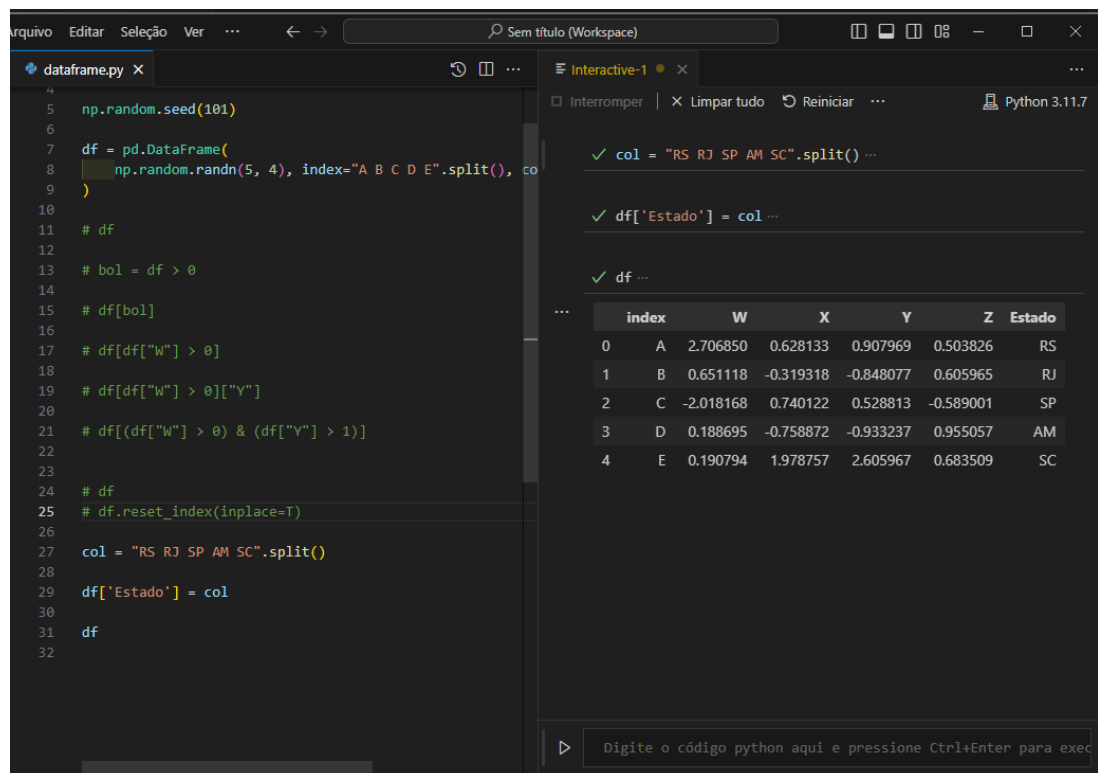
- Adicionar um coluna a um dataframe

Primeiro necessita definir uma variavel, com lista de valores:

```
col = 'RS RJ SP AM SC'.split()
```

Adicionar ao dataframe a nova coluna:

```
df['Estado'] = col
df
```



Deste modo, será acrescido uma nova coluna, na ordem que foi indicado os valores.

Para definir estes valores como o index, utilizamos o seguinte comando:

```
df.set_index('Estado', inplace=True)
```

The screenshot shows a Jupyter Notebook with a file named 'dataframe.py'. The code in the notebook is as follows:

```

4
5 np.random.seed(101)
6
7 df = pd.DataFrame(
8     np.random.randn(5, 4), index="A B C D E".split(), co
9 )
10
11 # df
12
13 # bol = df > 0
14
15 # df[bol]
16
17 # df[df["W"] > 0]
18
19 # df[df["W"] > 0][["Y"]]
20
21 # df[(df["W"] > 0) & (df["Y"] > 1)]
22
23
24 # df
25 # df.reset_index(inplace=True)
26
27 col = "RS RJ SP AM SC".split()
28
29 df["Estado"] = col
30
31 df.set_index("Estado", inplace=True)
32
33 df
34

```

The interactive output shows two views of the DataFrame 'df'. The first view shows the DataFrame with columns 'index', 'W', 'X', 'Y', 'Z', and 'Estado'. The second view shows the DataFrame with 'Estado' as the index and columns 'index', 'W', 'X', 'Y', and 'Z'.

index	W	X	Y	Z	Estado	
0	A	2.706850	0.628133	0.907969	0.503826	RS
1	B	0.651118	-0.319318	-0.848077	0.605965	RJ
2	C	-2.018168	0.740122	0.528813	-0.589001	SP
3	D	0.188695	-0.758872	-0.933237	0.955057	AM
4	E	0.190794	1.978757	2.605967	0.683509	SC

	index	W	X	Y	Z	
Estado	RS	A	2.706850	0.628133	0.907969	0.503826
	RJ	B	0.651118	-0.319318	-0.848077	0.605965
	SP	C	-2.018168	0.740122	0.528813	-0.589001
	AM	D	0.188695	-0.758872	-0.933237	0.955057
	SC	E	0.190794	1.978757	2.605967	0.683509

▼ pd.dataframe(3)

Basicamente o que o primeiro hier_index realiza o processo de juntar o valores de outside com inside, transformando em uma tupla, elemento por elemento

```

outside = ["G1", "G1", "G1", "G2", "G2", "G2"]
inside = [1, 2, 3, 1, 2, 3]
hier_index = list(zip(outside, inside))
hier_index = pd.MultiIndex.from_tuples(hier_index)

```

Já no segundo hier_index ele estará criando um índice de multi nível.

```
1 import numpy as np
2 import pandas as pd
3
4 outside = ["G1", "G1", "G1", "G2", "G2", "G2"]
5 inside = [1, 2, 3, 1, 2, 3]
6 hier_index = list(zip(outside, inside))
7 hier_index = pd.MultiIndex.from_tuples(hier_index)
8
9
10 df = pd.DataFrame(np.random.randn(6, 2), index=hier_index, columns=["A", "B"])
11
12 df
```

Conectado a Python 3.11.7

✓ import numpy as np ...

✓ df = ...

✓ df ...

		A	B
G1	1	1.379219	0.587517
	2	-0.409872	-0.513598
	3	0.831287	0.005237
G2	1	0.846034	-0.118314
	2	0.048949	-0.329986
	3	0.402843	1.235492

▶ Digite o código python aqui e pressi

Basicamente é a criação de "2" tabelas de valores

```
1 import numpy as np
2 import pandas as pd
3
4 outside = ["G1", "G1", "G1", "G2", "G2", "G2"]
5 inside = [1, 2, 3, 1, 2, 3]
6 hier_index = list(zip(outside, inside))
7 hier_index = pd.MultiIndex.from_tuples(hier_index)
8
9
10 df = pd.DataFrame(np.random.randn(6, 2), index=hier_index, columns=["A", "B"])
11
12 df.loc[['G1']]
```

Conectado a Python 3.11.7

✓ import numpy as np ...

✓ df = ...

✓ df ...

		A	B
G1	1	1.379219	0.587517
	2	-0.409872	-0.513598
	3	0.831287	0.005237
G2	1	0.846034	-0.118314
	2	0.048949	-0.329986
	3	0.402843	1.235492

✓ df.loc[['G1']] ...

	A	B
1	1.379219	0.587517
2	-0.409872	-0.513598
3	0.831287	0.005237

Com `df.loc['Valor do índice']` retorna o exato valor do índice em questão, sendo possível utilizar um dentro do outro, especificando mais os dados encontrados:

The screenshot shows a Jupyter Notebook interface. On the left, a code editor displays the following Python code:

```
1 import numpy as np
2 import pandas as pd
3
4 outside = ["G1", "G1", "G1", "G2", "G2", "G2"]
5 inside = [1, 2, 3, 1, 2, 3]
6 hier_index = list(zip(outside, inside))
7 hier_index = pd.MultiIndex.from_tuples(hier_index)
8
9
10 df = pd.DataFrame(np.random.randn(6, 2), index=hier_index, columns=["A", "B"])
11
12 df.loc['G1'].loc[1]
```

A red arrow points from the code `df.loc['G1'].loc[1]` to the interactive view on the right. The interactive view shows the following data:

df ...

		A	B
G1	1	1.379219	0.587517
	2	-0.409872	-0.513598
	3	0.831287	0.005237
G2	1	0.846034	-0.118314
	2	0.048949	-0.329986
	3	0.402843	1.235492

df.loc['G1'] ...

	A	B
1	1.379219	0.587517
2	-0.409872	-0.513598
3	0.831287	0.005237

df.loc['G1'].loc[1] ...

A	1.379219
B	0.587517
Name: 1, dtype: float64	

- Nome de índices

Com comando `df.index.names` é possível identificar os nomes dos índices deste dataframe.

Ao usar também, `df.index.names = ['Nome', 'Nome']` você consegue definir que nome constará neste dataframe

The screenshot shows a Jupyter Notebook with a Python script on the left and its interactive output on the right. The script defines a hierarchical index and a DataFrame. The output shows the state of the DataFrame at different steps, with the current state displaying a table of data.

```
1 import numpy as np
2 import pandas as pd
3
4 outside = ["G1", "G1", "G1", "G2", "G2", "G2"]
5 inside = [1, 2, 3, 1, 2, 3]
6 hier_index = list(zip(outside, inside))
7 hier_index = pd.MultiIndex.from_tuples(hier_index)
8
9
10 df = pd.DataFrame(np.random.randn(6, 2), index=hier_index, columns=["A", "B"])
11
12 df.loc['G1'].loc[1]
13
14 df.index.names = ['Grupo', 'Número']
15 df
```

Interactive-1

df.index.names ...

FrozenList([None, None])

df.index.names = ['Grupo', 'Número'] ...

df ...

		A	B
G1	1	1.379219	0.587517
	2	-0.409872	-0.513598
	3	0.831287	0.005237
G2	1	0.846034	-0.118314
	2	0.048949	-0.329986
	3	0.402843	1.235492

- Retorno por índice

Além do `.loc`, podemos utilizar o `.xs`. A vantagem é poder realizar um filtro de nível interno, sem passar pelo nível externo:

```
df.xs('G1')
# Mas também:
df.xs(1, level='Número')
# Ou
df.xs("G1", level="Grupo")
```

▼ Tratamento de dados ausentes

- `df.dropna()`

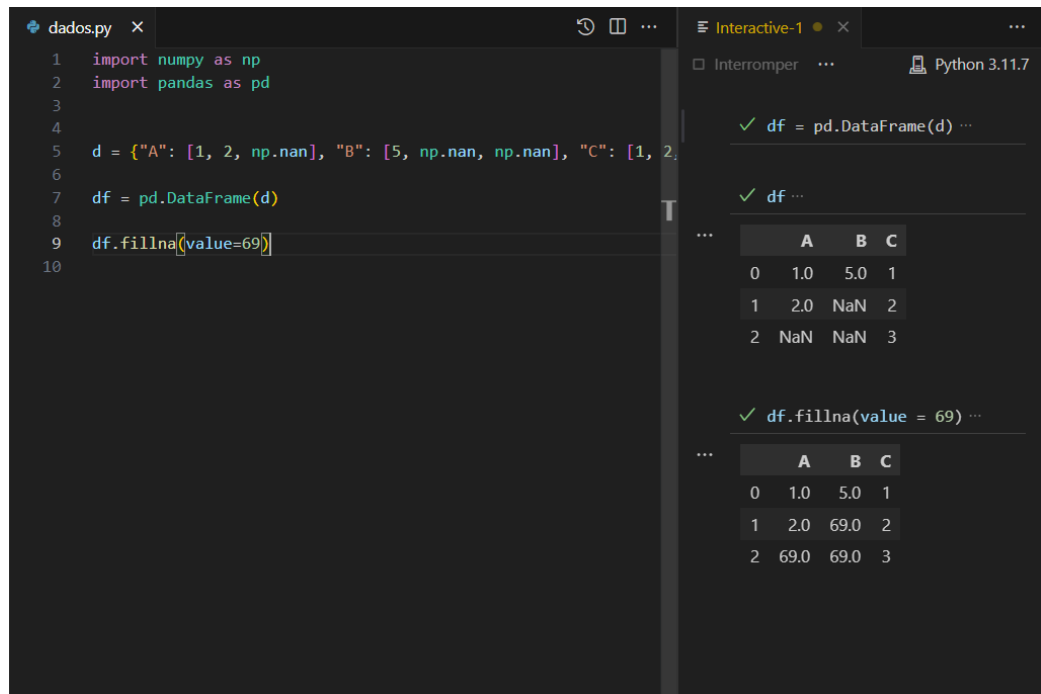
Exclui valores NaN através da exclusão da sua coluna

- `dropna(thresh=2)`

Exclui índice informado caso nas linhas haja exatamente quantidade de itens faltantes

- `df.fillna()`

Preenche os valores NaN com valor que for indicado dentro dele através do atributo 'Value':



The screenshot shows a Jupyter Notebook interface with a code editor on the left and an interactive console on the right. The code in the editor defines a DataFrame with three columns: A, B, and C. Column A has values [1, 2, np.nan], column B has values [5, np.nan, np.nan], and column C has values [1, 2, 3]. The DataFrame is then created and the `fillna()` method is used to replace all NaN values with 69.0.

```
1 import numpy as np
2 import pandas as pd
3
4
5 d = {"A": [1, 2, np.nan], "B": [5, np.nan, np.nan], "C": [1, 2, 3]}
6
7 df = pd.DataFrame(d)
8
9 df.fillna(value=69)
10
```

The interactive console shows the execution of the code. It first displays the DataFrame `df` with the original values, including NaNs. Then, it shows the result of `df.fillna(value = 69)`, where the NaN values have been replaced by 69.0.

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

	A	B	C
0	1.0	5.0	1
1	2.0	69.0	2
2	69.0	69.0	3

Trazendo uma função para valores NaN

The screenshot shows a Jupyter Notebook interface with a code editor on the left and an interactive console on the right. The code editor contains the following Python code:

```
1 import numpy as np
2 import pandas as pd
3
4 d = {"A": [1, 2, np.nan], "B": [5, np.nan, np.nan], "C": [1, 2, 3]}
5 df = pd.DataFrame(d)
6
7 # df.fillna(value=69)
8
9 df['A'].fillna(value=df['A'].sum())
10 df["A"].fillna(value=df["A"].mean())
11 df["A"].fillna(value=df["A"].max())
```

The interactive console shows the output of the code. It displays the initial DataFrame with NaN values, followed by the result of applying the fillna method with the sum, mean, and max of column A. The output shows that the NaN values in column A are replaced by the sum (69.0), mean (2.0), and max (3.0) of the non-NaN values in column A.

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

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2
2	69.0	NaN	3

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

Utilizando o registro 'method', é possível replicar os valores anteriores para os da sequência, como o uso do 'ffill':

```
dados.py x
1 import numpy as np
2 import pandas as pd
3
4
5 d = {"A": [1, 2, np.nan], "B": [5, np.nan, np.nan], "C": [1, 2, 3]}
6
7 df = pd.DataFrame(d)
8
9 # df.fillna(value=69)
10
11 # df['A'].fillna(value=df['A'].sum())
12 # df["A"].fillna(value=df["A"].mean())
13 # df["A"].fillna(value=df["A"].max())
14
15 # ffill = forward fill
16 df.fillna(method='ffill')
17
```

Interactive-1 Python 3.11.7

✓ df.fillna(method='ffill') ...

<ipython-input-12-5c0beae7dc1e: df.fillna(method='ffill')

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

```
df.fillna(method='ffill')
```

▼ GroupBy

```
groupby.py 1 x
1 import pandas as pd
2
3 data = {
4     "Empresa": [
5         "GOOG",
6         "GOOG",
7         "MSFT",
8         "MSFT",
9         "FB",
10        "FB",
11    ],
12    "Nome": ["Sam", "Charlie", "Amy", "Vanessa", "Carl", "Sarah"],
13    "Venda": [200, 120, 340, 124, 144, 290]
14 }
15
16 df = pd.DataFrame(data)
17
18 group = df.groupby('Empresa')
19
20 group
21
22 group.sum()
```

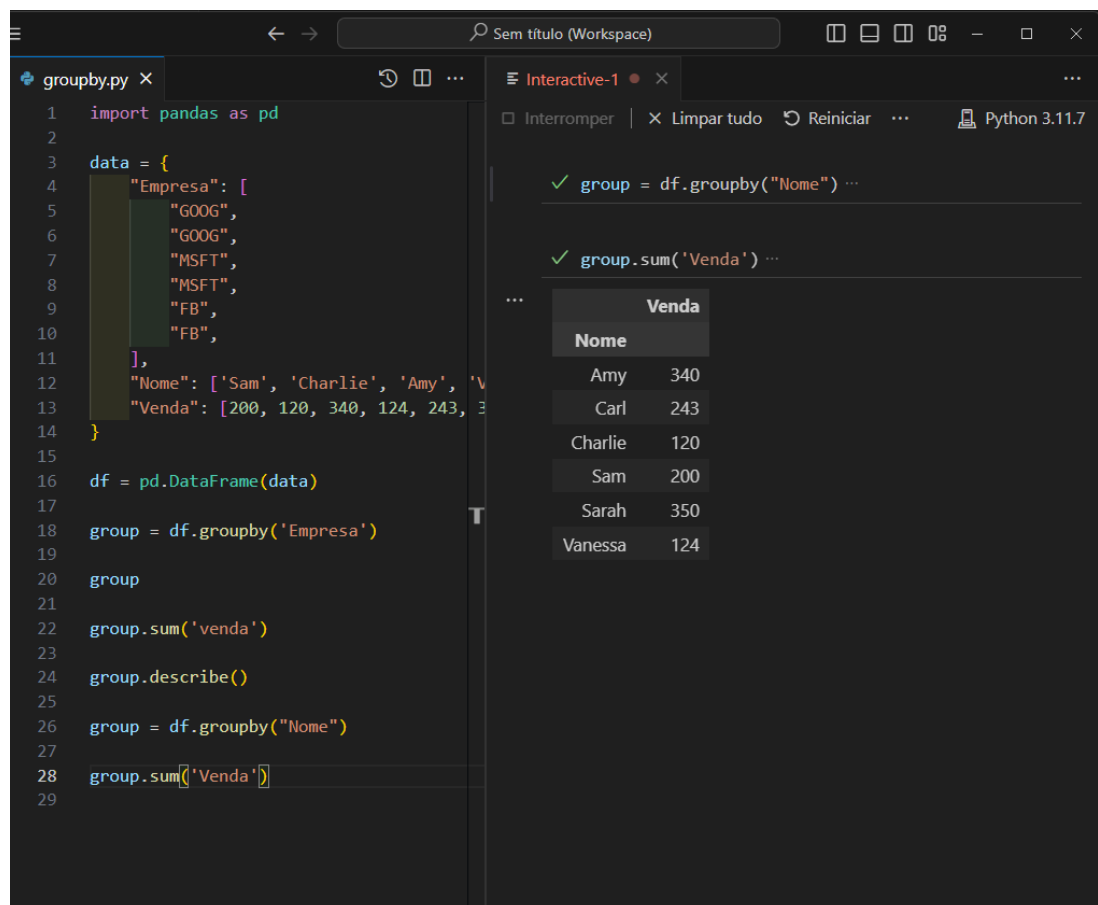
Interactive-1 Python 3.11.7

✓ group.sum() ...

	Nome	Venda
Empresa		
FB	CarlSarah	593
GOOG	SamCharlie	320
MSFT	AmyVanessa	464

Através do mesmo é possível realizar um agrupamento de informações conforme acima.

No caso em questão, foi agrupado as informações por empresa, trazendo assim os somatórios de venda, mas também do campo string. Para resolver, utilizei do atributo `sum()` indicando dentro dele a coluna venda, deste modo, somou devidamente.



```
1 import pandas as pd
2
3 data = {
4     "Empresa": [
5         "GOOG",
6         "GOOG",
7         "MSFT",
8         "MSFT",
9         "FB",
10        "FB",
11    ],
12    "Nome": ['Sam', 'Charlie', 'Amy', 'Vanessa', 'Carl', 'Sarah'],
13    "Venda": [200, 120, 340, 124, 243, 350]
14 }
15
16 df = pd.DataFrame(data)
17
18 group = df.groupby('Empresa')
19
20 group
21
22 group.sum('venda')
23
24 group.describe()
25
26 group = df.groupby("Nome")
27
28 group.sum('Venda')
```

Interactive-1

group = df.groupby("Nome") ...

group.sum('Venda') ...

Nome	Venda
Amy	340
Carl	243
Charlie	120
Sam	200
Sarah	350
Vanessa	124

função `describe` atrelado ao `groupby` trará uma série de valores base para análise base desses dados:

```
groupby.py X
1 import pandas as pd
2
3 data = {
4     "Empresa": [
5         "GOOG",
6         "GOOG",
7         "MSFT",
8         "MSFT",
9         "FB",
10        "FB",
11    ],
12    "Nome": ['Sam', 'Charlie', 'Amy', 'Vanessa', 'Sam', 'Charlie'],
13    "Venda": [200, 120, 340, 124, 243, 350],
14 }
15
16 df = pd.DataFrame(data)
17
18 group = df.groupby("Empresa")
19
20 group
21
22 group.sum('venda')
23
24 group.describe()
25
26 group = df.groupby("Nome")
27
28 group.sum('Venda')
29
```

Interactive-1 X

Interromper | Limpar tudo | Reiniciar | Variáveis | Python 3.11.7

✓ group = df.groupby("Nome") ...

✓ group.sum('Venda') ...

Venda	
Nome	
Amy	340
Carl	243
Charlie	120
Sam	200
Sarah	350
Vanessa	124

✓ group.describe() ...

Venda								
	count	mean	std	min	25%	50%	75%	max
Nome								
Amy	1.0	340.0	NaN	340.0	340.0	340.0	340.0	340.0
Carl	1.0	243.0	NaN	243.0	243.0	243.0	243.0	243.0
Charlie	1.0	120.0	NaN	120.0	120.0	120.0	120.0	120.0
Sam	1.0	200.0	NaN	200.0	200.0	200.0	200.0	200.0
Sarah	1.0	350.0	NaN	350.0	350.0	350.0	350.0	350.0
Vanessa	1.0	124.0	NaN	124.0	124.0	124.0	124.0	124.0

group.describe()

▼ Metodos Mesclar, Juntar e Concatenar

▼ Concatenação:

União de dataframes. Obrigatório que colunas tenham mesmo tamanho:

The screenshot shows a Jupyter Notebook interface with a code editor on the left and an interactive console on the right. The code in the editor defines three DataFrames: df1 (indices 0-3), df2 (indices 4-7), and df3 (indices 8-11). The line `pd.concat([df1, df2, df3])` is highlighted with a red box. The interactive console shows the execution of the code, displaying the concatenated result of df1, df2, and df3. The first table shows the individual DataFrames, and the second table shows the result of concatenating them along axis 1.

```
1 import pandas as pd
2
3 df1 = pd.DataFrame(
4     {
5         "A": ["A0", "A1", "A2", "A3"],
6         "B": ["B0", "B1", "B2", "B3"],
7         "C": ["C0", "C1", "C2", "C3"],
8         "D": ["D0", "D1", "D2", "D3"],
9     }, index=[0, 1, 2, 3])
10
11
12 df2 = pd.DataFrame(
13     {
14         "A": ["A4", "A5", "A6", "A7"],
15         "B": ["B4", "B5", "B6", "B7"],
16         "C": ["C4", "C5", "C6", "C7"],
17         "D": ["D4", "D5", "D6", "D7"],
18     },
19     index=[4, 5, 6, 7],
20 )
21
22 df3 = pd.DataFrame(
23     {
24         "A": ["A8", "A9", "A10", "A11"],
25         "B": ["B8", "B9", "B10", "B11"],
26         "C": ["C8", "C9", "C10", "C11"],
27         "D": ["D8", "D9", "D10", "D11"],
28     },
29     index=[8, 9, 10, 11],
30 )
31
32 pd.concat([df1, df2, df3])
33
34 pd.concat([df1, df2, df3], axis=1)
35
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

	A	B	C	D	A	B	C
0	A0	B0	C0	D0	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN
2	A2	B2	C2	D2	NaN	NaN	NaN
3	A3	B3	C3	D3	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	A4	B4	C4
5	NaN	NaN	NaN	NaN	A5	B5	C5
6	NaN	NaN	NaN	NaN	A6	B6	C6
7	NaN	NaN	NaN	NaN	A7	B7	C7
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN
11	NaN	NaN	NaN	NaN	NaN	NaN	NaN

É possível definir o eixo, para que deste modo sejam 'Criadas' novas colunas com continuação dos valores:


```

1 import pandas as pd
2
3 df1 = pd.DataFrame(
4     {
5         "A": ["A0", "A1", "A2", "A3"],
6         "B": ["B0", "B1", "B2", "B3"],
7         "C": ["C0", "C1", "C2", "C3"],
8         "D": ["D0", "D1", "D2", "D3"],
9     }, index=[0, 1, 2, 3])
10
11
12 df2 = pd.DataFrame(
13     {
14         "A": ["A4", "A5", "A6", "A7"],
15         "B": ["B4", "B5", "B6", "B7"],
16         "C": ["C4", "C5", "C6", "C7"],
17         "D": ["D4", "D5", "D6", "D7"],
18     },
19     index=[4, 5, 6, 7],
20 )
21
22 df3 = pd.DataFrame(
23     {
24         "A": ["A8", "A9", "A10", "A11"],
25         "B": ["B8", "B9", "B10", "B11"],
26         "C": ["C8", "C9", "C10", "C11"],
27         "D": ["D8", "D9", "D10", "D11"],
28     },
29     index=[8, 9, 10, 11],
30 )
31
32 pd.concat([df1, df2, df3])
33
34 pd.concat([df1, df2, df3], axis=1)
35

```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

	A	B	C	D	A	B	C	D	A	B	C	D
0	A0	B0	C0	D0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	A2	B2	C2	D2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	A3	B3	C3	D3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	A4	B4	C4	D4	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	A5	B5	C5	D5	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	A6	B6	C6	D6	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	A7	B7	C7	D7	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A8	B8	C8	D8
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A9	B9	C9	D9
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A10	B10	C10	D10
11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A11	B11	C11	D11

Para isto, usamos o atributo `axis=Indice`, logo, ao concluir o primeiro data frame, ele pula um índice.

▼ Mesclar

Já para valores que não participam da mesma ordenação, utilizamos a mesclagem, idêntica a utilização do `join` no SQL:



▼ Outros

Metodo `.unique`:

Retorna indice e valores da coluna que especificou

```

import pandas as pd
import numpy as np
df = pd.DataFrame(
    {
        "col1": [1, 2, 3, 4],
        "col2": [444, 555, 666, 444],
        "col3": ["abc", "def", "ghi", "xyz"],
    }
)

df['col1']
df['col1'].unique

```

```

1 import pandas as pd
2 import numpy as np
3 df = pd.DataFrame(
4     {
5         "col1": [1, 2, 3, 4],
6         "col2": [444, 555, 666, 444],
7         "col3": ["abc", "def", "ghi", "xyz"],
8     }
9 )
10
11
12 df["col1"]
13 df["col1"].unique()

```

Interactive-1

df["col1"] ...

```

...
0    1
1    2
2    3
3    4
Name: col1, dtype: int64

```

df["col1"].unique() ...

```

...
array([1, 2, 3, 4], dtype=int64)

```

Metodo .nunique

Retorna a contagem de valores, semelhante ao len do numpy:

```

df['col1'].nunique()
len(df['col1'].unique())

```

```
... <--> Sem título (Workspace) [Icons] 0% - [Window] [Close]

outras.py 3 x [Icons] ... Interactive-1 x ...
[Icons] Interromper | x Limpar tudo ... Python 3.11.7

1 import pandas as pd
2 import numpy as np
3 df = pd.DataFrame(
4     {
5         "col1": [1, 2, 3, 4],
6         "col2": [444, 555, 666, 444],
7         "col3": ["abc", "def", "ghi", "xyz"],
8     }
9 )
10
11
12 df["col1"]
13 df['col1'].unique()
14
15
16 df['col1'].nunique()
17 len(df['col1'].unique())
```

```
✓ df['col1'].nunique() ...
... 4

✓ len(df['col1'].unique()) ...
... 4
```

Metodo .value_counts()

Faz contagem de valores por linha,

```
df['col2']
df['col2'].value_counts
```

```
outras.py 2 x [Icons] ... Interactive-1 x ...
[Icons] Interromper | x Limpar tudo ... Python 3.11.7

6     "col2": [444, 555, 666, 444],
7     "col3": ["abc", "def", "ghi", "xyz"],
8 }
9 )
10
11
12 # df["col1"]
13 # df['col1'].unique()
14
15
16 # df['col1'].nunique()
17 # len(df['col1'].unique())
18 df['col2']
19 df['col2'].value_counts
20
```

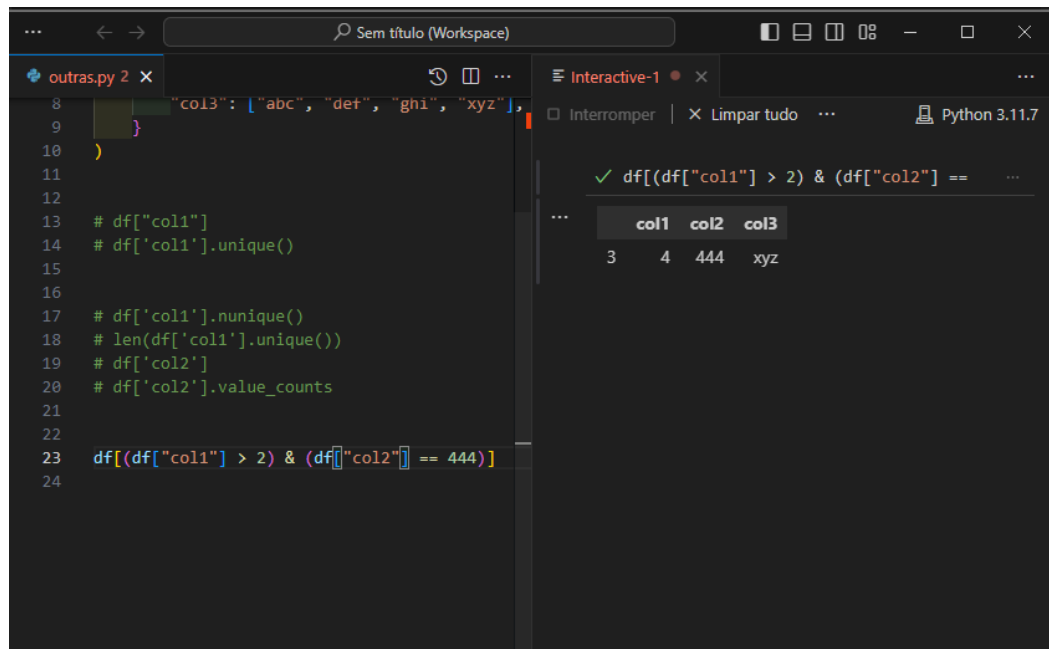
```
✓ df['col2'] ...
...
0    444
1    555
2    666
3    444
Name: col2, dtype: int64

✓ df['col2'].value_counts ...
...
<bound method IndexOpsMixin.value_counts of 0
1    555
2    666
3    444
Name: col2, dtype: int64>
```

Neste caso em questão, como coluna 2 havia valores repetidos foi possível contar 2 para a mesma.

Filtros para subdataframe:

```
df[(df["col1"] > 2) & (df["col2"] == 444)]
```



The screenshot shows a Jupyter Notebook workspace with a file named 'outras.py 2'. The code in the notebook includes several lines for data manipulation and filtering. The final line executed is `df[(df["col1"] > 2) & (df["col2"] == 444)]`. The output of this execution is displayed in the 'Interactive-1' pane, showing a table with columns 'col1', 'col2', and 'col3'. The table contains one row with values 3, 4, and 444, and a label 'xyz' in the 'col3' column.

```
8      "col3": ["abc", "def", "ghi", "xyz"],
9      }
10 )
11
12
13 # df["col1"]
14 # df['col1'].unique()
15
16
17 # df['col1'].nunique()
18 # len(df['col1'].unique())
19 # df['col2']
20 # df['col2'].value_counts
21
22
23 df[(df["col1"] > 2) & (df["col2"] == 444)]
24
```

✓ df[(df["col1"] > 2) & (df["col2"] == 444)]

col1	col2	col3
3	4	444 xyz

Metodo .apply:

Permite aplicar uma função em todos elementos individuais de um dataframe:

```
def vezes2(x):
    return x*2

df['col3'].apply(len)

df['col1'].apply(vezes2)

df["col1"].apply(lambda x: x*x)
```

The screenshot shows a Jupyter Notebook workspace with a file named 'outras.py 2'. The code in the notebook includes comments and function definitions for data analysis. The interactive console on the right shows the execution of three commands: applying 'len' to 'col3', applying a custom function 'vezes2' to 'col1', and applying a lambda function to square the values of 'col1'. Each command's output is displayed as a table with index and value, and the data type is noted as 'int64'.

```
10 )
11
12
13 # df["col1"]
14 # df['col1'].unique()
15
16
17 # df['col1'].nunique()
18 # len(df['col1'].unique())
19 # df['col2']
20 # df['col2'].value_counts
21
22
23 # df[(df["col1"] > 2) & (df["col2"] == 444)]
24
25
26 def vezes2(x):
27     return x * 2
28
29
30 df["col3"].apply(len)
31
32 df["col1"].apply(vezes2)
33
34 df["col1"].apply(lambda x: x * x)
35
```

Interactive-1

df["col3"].apply(len) ...

0	1
1	4
2	9
3	16

Name: col1, dtype: int64

df["col1"].apply(vezes2) ...

0	2
1	4
2	6
3	8

Name: col1, dtype: int64

df["col1"].apply(lambda x: x * x) ...

0	1
1	4
2	9
3	16

Name: col1, dtype: int64

Deletar colunas em um dataframe:

Utilizar função del:

```
del df ['col2']
```

The screenshot shows a Jupyter Notebook workspace with a file named 'outras.py' and an interactive console. The script in 'outras.py' contains the following code:

```
9 }
10 )
11
12 # df["col1"]
13 # df['col1'].unique()
14
15
16 # df['col1'].nunique()
17 # len(df['col1'].unique())
18 # df['col2']
19 # df['col2'].value_counts
20
21
22 # df[(df["col1"] > 2) & (df["col2"] == 444)]
23
24
25
26 # def vezes2(x):
27 #     return x * 2
28
29
30 # df["col3"].apply(len)
31 # df["col1"].apply(vezes2)
32 # df["col1"].apply(lambda x: x * x)
33
34
35
36
37 del df ['col2']
38
39 df
40
```

The interactive console shows the execution of the code, with the following output:

```
✓ del df ['col2'] ...
✓ df ...
```

	col1	col3
0	1	abc
1	2	def
2	3	ghi
3	4	xyz

df.columns

Retorna as colunas do dataframe

The screenshot shows a Jupyter Notebook workspace with a file named 'outras.py' and an interactive console. The script in 'outras.py' contains the following code:

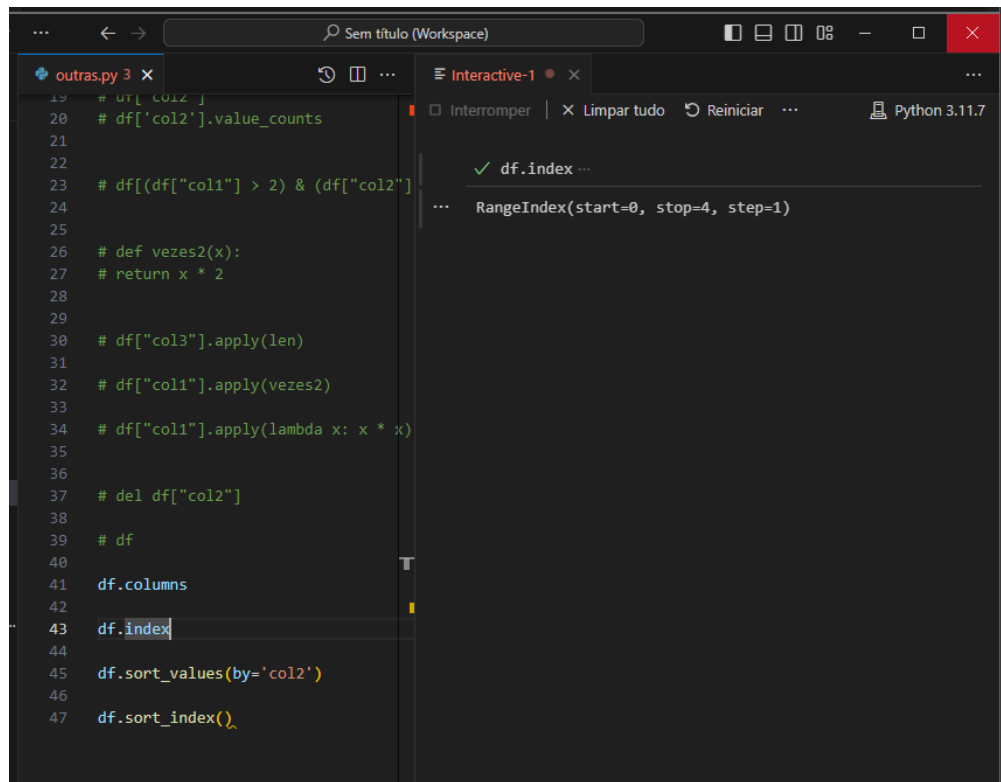
```
19 # df['col2']
20 # df['col2'].value_counts
21
22
23 # df[(df["col1"] > 2) & (df["col2"] == 444)]
24
25
26 # def vezes2(x):
27 #     return x * 2
28
29
30 # df["col3"].apply(len)
31 # df["col1"].apply(vezes2)
32 # df["col1"].apply(lambda x: x * x)
33
34
35
36
37 # del df["col2"]
38
39 # df
40
41 df.columns
42
43 df.index
44
45 df.sort_values(by='col2')
46
47 df.sort_index()
```

The interactive console shows the execution of the code, with the following output:

```
✓ df.columns ...
Index(['col1', 'col2', 'col3'], dtype='object')
```

df.index

Retorna o indice do data frame



The screenshot shows a Jupyter Notebook window with a file named 'outras.py 3'. The code in the notebook includes several pandas operations: dropping a column, getting value counts, filtering rows, applying a function to a column, deleting a column, and sorting values. The current cell shows the command `df.index` being executed. The output in the interactive console shows a green checkmark and the text `df.index ...` followed by `RangeIndex(start=0, stop=4, step=1)`.

```
19 # df['col2']
20 # df['col2'].value_counts
21
22
23 # df[(df["col1"] > 2) & (df["col2"]
24
25
26 # def vezes2(x):
27 # return x * 2
28
29
30 # df["col3"].apply(len)
31
32 # df["col1"].apply(vezes2)
33
34 # df["col1"].apply(lambda x: x * x)
35
36
37 # del df["col2"]
38
39 # df
40
41 df.columns
42
43 df.index
44
45 df.sort_values(by='col2')
46
47 df.sort_index()
```

✓ df.index ...
... RangeIndex(start=0, stop=4, step=1)

df.sort_values

Ordena os valores da coluna indica no dataframe

The screenshot shows a Jupyter Notebook interface with a code editor on the left and an interactive console on the right. The code editor contains the following Python code:

```
19 # df['col2']
20 # df['col2'].value_counts
21
22
23 # df[(df["col1"] > 2) & (df["col2"]
24
25
26 # def vezes2(x):
27 # return x * 2
28
29
30 # df["col3"].apply(len)
31
32 # df["col1"].apply(vezes2)
33
34 # df["col1"].apply(lambda x: x * x)
35
36
37 # del df["col2"]
38
39 # df
40
41 df.columns
42
43 df.index
44
45 df.sort_values(by='col2')
46
47 df.sort_index()
```

The interactive console shows the output of the last executed code cell, which is the result of `df.sort_values(by='col2')`. The output is a DataFrame with the following data:

	col1	col2	col3
0	1	444	abc
3	4	444	xyz
1	2	555	def
2	3	666	ghi

`df.sort_index()`

Ordena através do índice

The screenshot shows a Jupyter Notebook interface with a code editor on the left and an interactive console on the right. The code editor contains the following Python code:

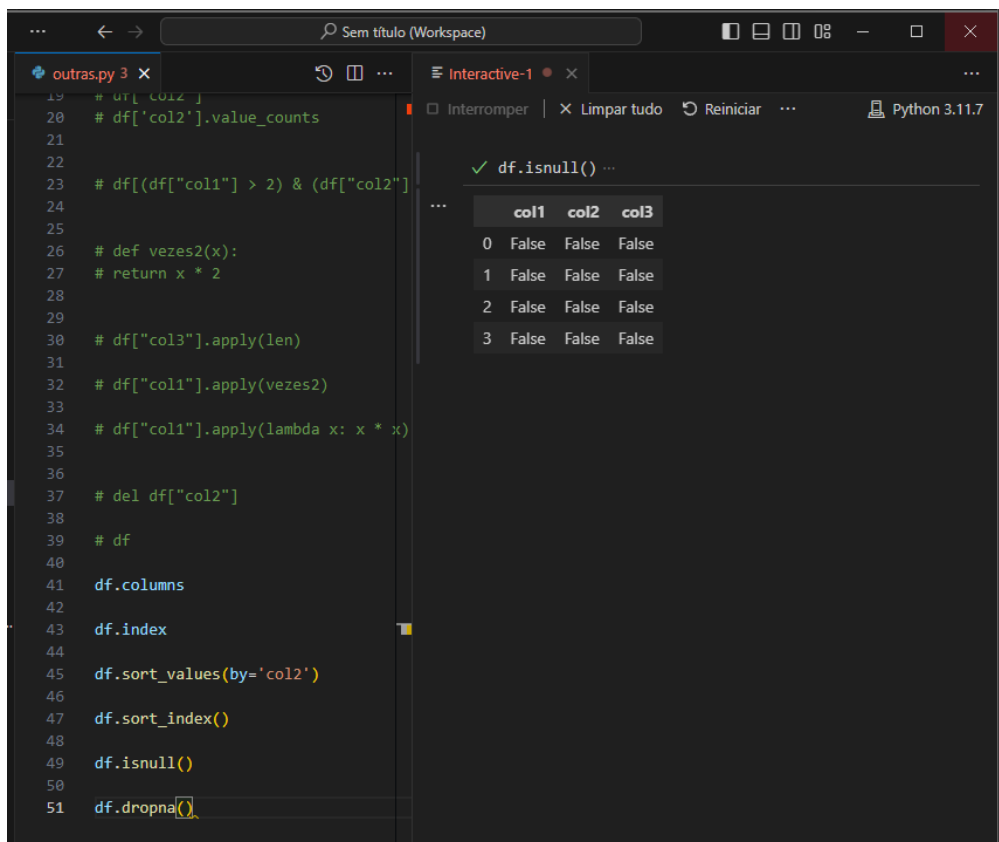
```
19 # df['col2']
20 # df['col2'].value_counts
21
22
23 # df[(df["col1"] > 2) & (df["col2"]
24
25
26 # def vezes2(x):
27 # return x * 2
28
29
30 # df["col3"].apply(len)
31
32 # df["col1"].apply(vezes2)
33
34 # df["col1"].apply(lambda x: x * x)
35
36
37 # del df["col2"]
38
39 # df
40
41 df.columns
42
43 df.index
44
45 df.sort_values(by='col2')
46
47 df.sort_index()
```

The interactive console shows the output of the last executed code cell, which is the result of `df.sort_index()`. The output is a DataFrame with the following data:

	col1	col2	col3
0	1	444	abc
1	2	555	def
2	3	666	ghi
3	4	444	xyz

`df.isnull()`

Retorna todos os valores nulos do dataframe



df.pivot_table

Ajusta um tabela com os parâmetros que você definir:

```
import pandas as pd

data = {
    "A": ["foo", "foo", "foo", "bar", "bar", "bar"]
    "B": ["one", "one", "two", "two", "one", "one"]
    "C": ["x", "y", "x", "y", "x", "y"],
    "D": [1, 3, 2, 5, 4, 1],
}
df = pd.DataFrame(data)

df.pivot_table(values='D', index=['A', 'B'], column
```

The screenshot shows a Jupyter Notebook workspace with a file named 'outros2.py'. The code in the notebook is as follows:

```
1 import pandas as pd
2
3
4 data = {
5     "A": ["foo", "foo", "foo", "bar", "bar", "bar"],
6     "B": ["one", "one", "two", "two", "one", "two"],
7     "C": ["x", "y", "x", "y", "x", "y"],
8     "D": [1, 3, 2, 5, 4, 1],
9 }
10 df = pd.DataFrame(data)
11
12 df.pivot_table(values='D', index=['A', 'B'], c
13
```

The interactive output on the right shows the command `df.pivot_table(values='D', index=['A', 'B'], c` being executed. The resulting pivot table is:

		C	x	y
A	B			
	one	4.0	1.0	
bar	two	NaN	5.0	
	one	1.0	3.0	
foo	two	2.0	NaN	

`df.dropna()`

Deleta os valores nulos

▼ Entrada e Saída

Para dar entrada em um documento, é utilizado comando:

```
Variavel = pd.read_tipoarquivo('Nome_do_arquivo', sep=' '
df = pd.read_csv('exemplo', sep=',')
```

É possível também utilizar o `df = pd.read_html('Link')` para puxar um dataframe da internet, porém não tive bons resultados com os que tentei, acredito que, é possível utilizar somente a alguns casos esse web scraping.

▼ Problemas:

Durante a realização do código, obtive alguns problemas, estes relacionados aos índices. Por algum motivo, eles acabaram se perdendo, na substituição, acredito que pelo comando `df.reset_index:`

The screenshot shows a Jupyter Notebook with a file named 'dataframe.py'. The code in the notebook is as follows:

```

4
5 np.random.seed(101)
6
7 df = pd.DataFrame(
8     np.random.randn(5, 4), index="A B C D E".split(), co
9 )
10
11 # df
12
13 # bol = df > 0
14
15 # df[bol]
16
17 # df[df["W"] > 0]
18
19 # df[df["W"] > 0]["Y"]
20
21 # df[(df["W"] > 0) & (df["Y"] > 1)]
22
23
24 # df
25 df.reset_index(inplace=True)
26
27 col = "RS RJ SP AM SC".split()
28
29 df["Estado"] = col
30
31 df.set_index('Estado', inplace=True)
32
33 df
34

```

The interactive output on the right shows the result of the code execution. It displays the DataFrame 'df' after the `df.reset_index(inplace=True)` operation, which has a single index level. The output is a table with 5 rows and 6 columns:

	Estado	index	W	X	Y	Z
0	RS	A	2.706850	0.628133	0.907969	0.503826
1	RJ	B	0.651118	-0.319318	-0.848077	0.605965
2	SP	C	-2.018168	0.740122	0.528813	-0.589001
3	AM	D	0.188695	-0.758872	-0.933237	0.955057
4	SC	E	0.190794	1.978757	2.605967	0.683509

Below this, the output of `df.reset_index(inplace=True)` is shown, which is a table with 5 rows and 7 columns, including a 'level_0' column:

	level_0	Estado	index	W	X	Y	Z
0	0	RS	A	2.706850	0.628133	0.907969	0.503826
1	1	RJ	B	0.651118	-0.319318	-0.848077	0.605965
2	2	SP	C	-2.018168	0.740122	0.528813	-0.589001
3	3	AM	D	0.188695	-0.758872	-0.933237	0.955057
4	4	SC	E	0.190794	1.978757	2.605967	0.683509

Deste modo ao tentar realizar o `df.set_index`, retorno até trazia os estados como índice principal, mas gerava uma coluna a mais incorretamente.

Durante as aulas, ao utilizar o comando `df.xs` é especificado que para retorno de um valor interno, não precisaria passar valor externo, porém o mesmo só funcionou ao especificar o level dele:

```

import numpy as np
import pandas as pd

outside = ["G1", "G1", "G1", "G2", "G2", "G2"]
inside = [1, 2, 3, 1, 2, 3]
hier_index = list(zip(outside, inside))
hier_index = pd.MultiIndex.from_tuples(hier_index)

df = pd.DataFrame(np.random.randn(6, 2), index=hier_index)

```

```
df.loc['G1'].loc[1]

df.index.names = ['Grupo', 'Número']
df

df.xs('G1')

df.xs('G1').xs(1)

df.xs(1, level='Número')
```

Sem o level, ele retornou o seguinte:

```
df.xs(1) ...

-----
KeyError                                Traceback (most recent call last)
File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-pac
3789 try:
-> 3790     return self._engine.get_loc(casted_key)
3791 except KeyError as err:

File index.pyx:152, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:181, in pandas._libs.index.IndexEngine.get_loc()

File pandas\_libs\hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.PyObjectHashTable.get_item()

File pandas\_libs\hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 1

The above exception was the direct cause of the following exception:

KeyError                                Traceback (most recent call last)
File c:\Users\leofe\Documents\LAMIA\Pandas\dataframe2.py:1
----> 1 df.xs(1)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-pac
4225     index = self.index
...
3800     # InvalidIndexError. Otherwise we fall through and re-raise
3801     # the TypeError.
3802     self._check_indexing_error(key)

KeyError: 1

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```