Ciência de Dados com Python (3)

Aquilo que escuto eu esqueço,
Aquilo que vejo eu lembro,
Aquilo que faço eu aprendo.

Confúcio

”

Let´s code!

1.

Introdução

Pandas para Análise de Dados

Pandas o que é?

- É uma biblioteca de código fonte aberto escrita sobre o Numpy
- Permite visualização rápida e limpeza de dados
- Muito semelhante ao Excel
- Pode trabalhar com diferentes tipos de dados
- Possui métodos próprios de visualização de dados

Para instalar

- Terminal e digite:

  conda install pandas

  pip install pandas

2.
Séries

Pandas para Análise de Dados

Series o que é?

- Semelhante a uma matriz NumPy (construída sobre o objeto de matriz NumPy).

- Diferença é que uma Serie pode ter rótulos de eixos, o que significa que pode ser indexado por um rótulo, em vez de apenas uma localização numérica.

- Também não precisa manter dados numéricos, ele pode conter qualquer objeto Python arbitrário

séries

```
import numpy as np
import pandas as pd
```

```
labels = ['A', 'B', 'C']
lista = [10, 20, 30]
arr = np.array([40, 50, 60])
d = {'a':110, 'b':120, 'c':130}
```

```
series = pd.Series(data = lista, index = labels)
series
```

```
A    10
B    20
C    30
dtype: int64
```

```
series['B']
```

```
20
```

```
pd.Series(lista, labels)
```

```
A    10
B    20
C    30
dtype: int64
```

```
pd.Series(arr, labels)
```

```
A    40
B    50
C    60
dtype: int32
```

dados

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

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

```
ser1 = pd.Series([1,2,3,4], index = ['verde', 'amarelo', 'vermelho', 'azul'])
ser1
```

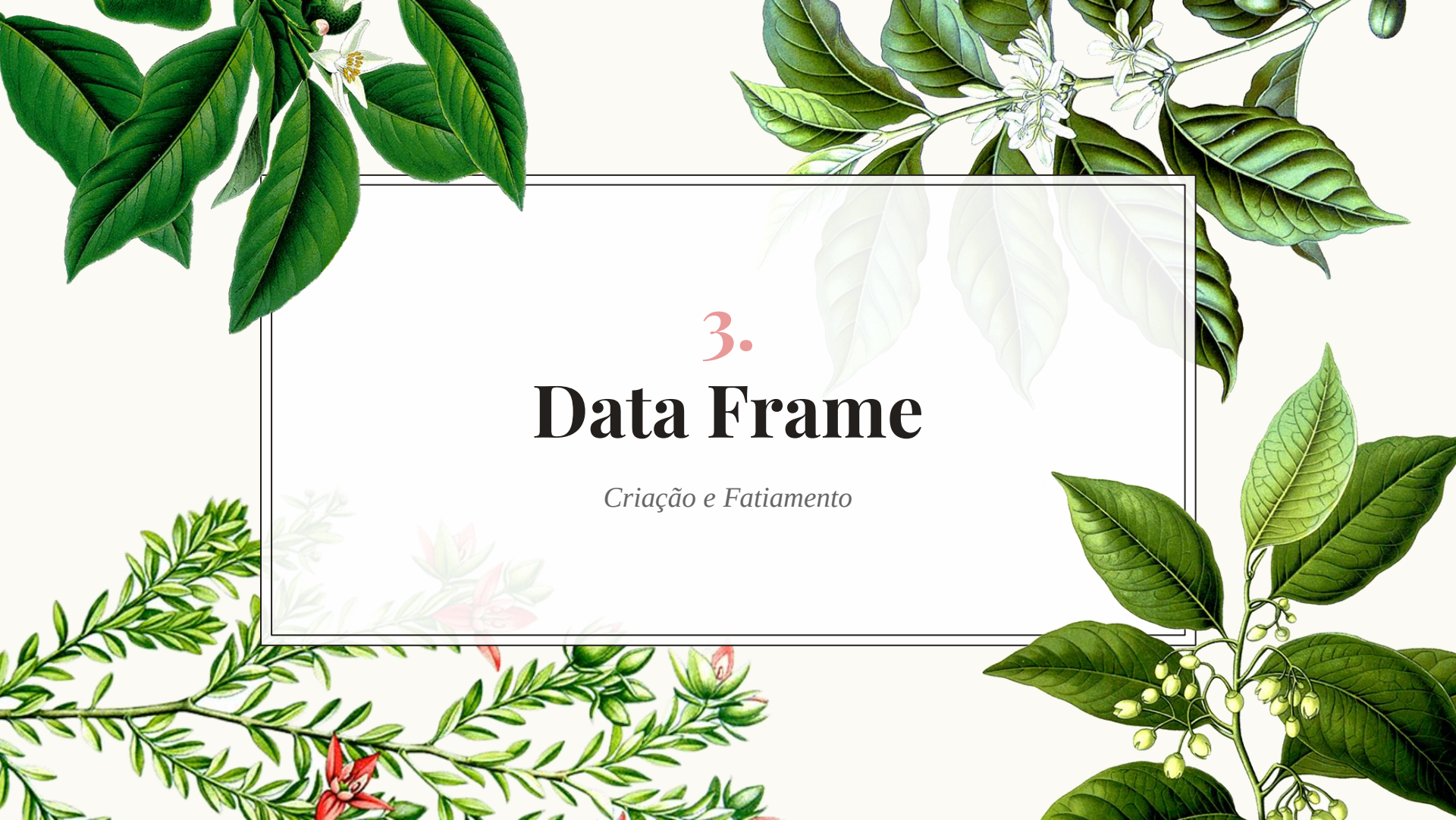
```
verde       1
amarelo     2
vermelho    3
azul        4
dtype: int64
```

```
ser2 = pd.Series([1,2,3,4], index = ['verde', 'amarelo', 'laranja', 'azul'])
ser2
```

```
verde      1
amarelo    2
laranja    3
azul       4
dtype: int64
```

```
ser1 + ser2
```

```
amarelo     4.0
azul        8.0
laranja     NaN
verde       2.0
vermelho    NaN
dtype: float64
```



3.

Data Frame

Criação e Fatiamento

O que é?

- É o elemento mais importante do Pandas
- Diretamente inspirados pela linguagem de programação R.
- Podemos pensar em um DataFrame como um monte de objetos de série juntos para compartilhar o mesmo índice.

criação

```
import pandas as pd
import numpy as np
```

```
#gerar números aleatórios com uma semente fixa
```

```
np.random.seed(101)
```

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

seleção

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

```
A     2.706850
B     0.651118
C    -2.018168
D     0.188695
E     0.190794
Name: W, dtype: float64
```

```
df['X']
```

```
A     0.628133
B    -0.319318
C     0.740122
D    -0.758872
E     1.978757
Name: X, dtype: float64
```

```
# Passando uma lista com nomes das colunas
df[['W','Z']]
```

	W	Z
A	2.706850	0.503826
B	0.651118	0.605965
C	-2.018168	-0.589001
D	0.188695	0.955057
E	0.190794	0.683509

```
# Sintaxe SQL (Não recomendado!)
df.W
```

```
A     2.706850
B     0.651118
C    -2.018168
D     0.188695
E     0.190794
Name: W, dtype: float64
```

Seleção

```
df.loc['A']    # Selecionando linhas: **
```

```
W    2.706850
X    0.628133
Y    0.907969
Z    0.503826
Name: A, dtype: float64
```

```
df.iloc[2]   # selecionando com base na posição
```

```
W   -2.018168
X    0.740122
Y    0.528813
Z   -0.589001
Name: C, dtype: float64
```

```
df.loc['B','Y']    # Selecionando subconjunto de linhas e colunas **
```

```
-0.84807698340363147
```

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

	W	Y
A	2.706850	0.907969
B	0.651118	-0.848077

```
df.iloc[1:3,1:]
```

	X	Y	Z
B	-0.319318	-0.848077	0.605965
C	0.740122	0.528813	-0.589001

4.

Seleção Condicional

DataFrames

Seleção condicional

```
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>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

```
df[df>0]
```

	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]
```

	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']
```

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

```
bol = df['W'] > 0
df2 = df[bol]
df2['Y']
```

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

Seleção condicional

```
df[df['W']>0][['Y','X']]
```

	Y	X
A	0.907969	0.628133
B	-0.848077	-0.319318
D	-0.933237	-0.758872
E	2.605967	1.978757

Para duas condições, você pode usar | e & com parênteses:

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

	W	X	Y	Z
E	0.190794	1.978757	2.605967	0.683509

5.

Inserção / Remoção

DataFrames

Inserção / Remoção

```
df['new'] = df['W'] + df['Y']    # criando colunas
df
```

	W	X	Y	Z	new
A	2.706850	0.628133	0.907969	0.503826	3.614819
B	0.651118	-0.319318	-0.848077	0.605965	-0.196959
C	-2.018168	0.740122	0.528813	-0.589001	-1.489355
D	0.188695	-0.758872	-0.933237	0.955057	-0.744542
E	0.190794	1.978757	2.605967	0.683509	2.796762

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

	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      # Porém, tal exclusão só ocorreu logicamente
```

	W	X	Y	Z	new
A	2.706850	0.628133	0.907969	0.503826	3.614819
B	0.651118	-0.319318	-0.848077	0.605965	-0.196959
C	-2.018168	0.740122	0.528813	-0.589001	-1.489355
D	0.188695	-0.758872	-0.933237	0.955057	-0.744542
E	0.190794	1.978757	2.605967	0.683509	2.796762

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

```
# deletar linha
df.drop('E',axis=0, inplace=True)
df
```

	W	X	Y	Z	new
A	2.706850	0.628133	0.907969	0.503826	3.614819
B	0.651118	-0.319318	-0.848077	0.605965	-0.196959
C	-2.018168	0.740122	0.528813	-0.589001	-1.489355
D	0.188695	-0.758872	-0.933237	0.955057	-0.744542

6.
Índices

DataFrame

Índices

```
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

```
# Redefinir para o padrão 0,1 ... n índice
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

Índices

```
novoind = 'GO SP RJ SC PB'.split()
```

```
df['Estados'] = novoind
df
```

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

```
df.set_index('Estados')
```

Estados	W	X	Y	Z
GO	2.706850	0.628133	0.907969	0.503826
SP	0.651118	-0.319318	-0.848077	0.605965
RJ	-2.018168	0.740122	0.528813	-0.589001
SC	0.188695	-0.758872	-0.933237	0.955057
PB	0.190794	1.978757	2.605967	0.683509

Índices

```
df
```

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

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

	W	X	Y	Z
Estados				
GO	2.706850	0.628133	0.907969	0.503826
SP	0.651118	-0.319318	-0.848077	0.605965
RJ	-2.018168	0.740122	0.528813	-0.589001
SC	0.188695	-0.758872	-0.933237	0.955057
PB	0.190794	1.978757	2.605967	0.683509

7.

Índices Multiníveis

DataFrame

Hierarquia

```
# Níveis de Índice
outside = ['G1','G1','G1','G2','G2','G2']
inside = [1,2,3,1,2,3]
hier_index = list(zip(outside,inside))
hier_index
```

```
[('G1', 1), ('G1', 2), ('G1', 3), ('G2', 1), ('G2', 2), ('G2', 3)]
```

```
hier_index = pd.MultiIndex.from_tuples(hier_index)
hier_index
```

```
MultiIndex([('G1', 1),
            ('G1', 2),
            ('G1', 3),
            ('G2', 1),
            ('G2', 2),
            ('G2', 3)],
           )
```

```
df = pd.DataFrame(np.random.randn(6,2),index=hier_index,columns=['A','B'])
df
```

		A	B
G1	1	2.706850	0.628133
	2	0.907969	0.503826
	3	0.651118	-0.319318
G2	1	-0.848077	0.605965
	2	-2.018168	0.740122
	3	0.528813	-0.589001

Indexação

		A	B
G1	1	2.706850	0.628133
	2	0.907969	0.503826
	3	0.651118	-0.319318
G2	1	-0.848077	0.605965
	2	-2.018168	0.740122
	3	0.528813	-0.589001

```
df['B']
```

```
G1  1    0.628133
    2    0.503826
    3   -0.319318
G2  1    0.605965
    2    0.740122
    3   -0.589001
Name: B, dtype: float64
```

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

	A	B
1	2.706850	0.628133
2	0.907969	0.503826
3	0.651118	-0.319318

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

```
A    2.706850
B    0.628133
Name: 1, dtype: float64
```

Níveis

```
df.index.names
```

```
FrozenList([None, None])
```

```
df.index.names = ['Grupo','Valores']
df
```

Grupo	Valores	A	B
G1	1	2.706850	0.628133
	2	0.907969	0.503826
	3	0.651118	-0.319318
G2	1	-0.848077	0.605965
	2	-2.018168	0.740122
	3	0.528813	-0.589001

```
df.xs('G1')
```

Valores	A	B
1	2.706850	0.628133
2	0.907969	0.503826
3	0.651118	-0.319318

```
df.xs(['G1',1])
```

```
A    2.706850
B    0.628133
Name: (G1, 1), dtype: float64
```

```
df.xs(1,level='Valores')
```

Grupo	A	B
G1	2.706850	0.628133
G2	-0.848077	0.605965

8.

Dados Ausentes

Data Frame

Dados Ausentes

```
df = pd.DataFrame({'A':[1,2,np.nan],
                   'B':[5,np.nan,np.nan],
                   'C':[1,2,3]})
df
```

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

```
df.dropna()
```

	A	B	C
0	1.0	5.0	1

```
df.dropna(axis=1)
```

	C
0	1
1	2
2	3

```
df.dropna(thresh=2)
```

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2

```
df.fillna(value='Conteúdo')
```

	A	B	C
0	1	5	1
1	2	Conteúdo	2
2	Conteúdo	Conteúdo	3

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

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

```
df
```

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

```
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

9.

GroupBy

DataFrame

Criação

```
import pandas as pd
# Cria um DataFrame
dados = {'Empresa':['NSA','NSA','EXTRA','EXTRA','TATICO','TATICO'],
       'Nome':['Júlia','Maria','amanda','Vanessa','Bruna','André'],
       'Venda':[200,120,340,124,243,350]}
df = pd.DataFrame(dados)
df
```

	Empresa	Nome	Venda
0	NSA	Júlia	200
1	NSA	Maria	120
2	EXTRA	amanda	340
3	EXTRA	Vanessa	124
4	TATICO	Bruna	243
5	TATICO	André	350

Métodos

	Empresa	Nome	Venda
0	NSA	Júlia	200
1	NSA	Maria	120
2	EXTRA	amanda	340
3	EXTRA	Vanessa	124
4	TATICO	Bruna	243
5	TATICO	André	350

```
: por_companhia = df.groupby("Empresa")
```

```
: por_companhia.mean()
```

:

Empresa	Venda
EXTRA	232.0
NSA	160.0
TATICO	296.5

```
: por_companhia.std()
```

:

Empresa	Venda
EXTRA	152.735065
NSA	56.568542
TATICO	75.660426

Métodos

```
por_companhia.min()
```

Empresa	Nome	Venda
EXTRA	Vanessa	124
NSA	Júlia	120
TATICO	André	243

```
por_companhia.max()
```

Empresa	Nome	Venda
EXTRA	amanda	340
NSA	Maria	200
TATICO	Bruna	350

```
por_companhia.count()
```

Empresa	Nome	Venda
EXTRA	2	2
NSA	2	2
TATICO	2	2

Métodos

```
por_companhia.describe()
```

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

```
por_companhia.describe().transpose()
```

	Empresa	EXTRA	NSA	TATICO
Venda	count	2.000000	2.000000	2.000000
	mean	232.000000	160.000000	296.500000
	std	152.735065	56.568542	75.660426
	min	124.000000	120.000000	243.000000
	25%	178.000000	140.000000	269.750000
	50%	232.000000	160.000000	296.500000
	75%	286.000000	180.000000	323.250000
	max	340.000000	200.000000	350.000000

Métodos

```
por_companhia.describe().transpose()
```

	Empresa	EXTRA	NSA	TATICO
Venda	count	2.000000	2.000000	2.000000
	mean	232.000000	160.000000	296.500000
	std	152.735065	56.568542	75.660426
	min	124.000000	120.000000	243.000000
	25%	178.000000	140.000000	269.750000
	50%	232.000000	160.000000	296.500000
	75%	286.000000	180.000000	323.250000
	max	340.000000	200.000000	350.000000

```
por_companhia.describe().transpose()['NSA']
```

```
Venda  count      2.000000
       mean     160.000000
       std       56.568542
       min      120.000000
       25%      140.000000
       50%      160.000000
       75%      180.000000
       max      200.000000
Name: NSA, dtype: float64
```

10.

Mesclar, juntar e concatenar

Exemplos

```python
import pandas as pd
```

```python
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']},
                   index=[0, 1, 2, 3])
```

```python
df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                    'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7']},
                     index=[4, 5, 6, 7])
```

```python
df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
                    'B': ['B8', 'B9', 'B10', 'B11'],
                    'C': ['C8', 'C9', 'C10', 'C11'],
                    'D': ['D8', 'D9', 'D10', 'D11']},
                   index=[8, 9, 10, 11])
```


df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df2

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

df3

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Concatenação

```
pd.concat([df1,df2,df3])
```

	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

```
pd.concat([df1,df2,df3],axis=1)
```

	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

```
esquerda = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                         'A': ['A0', 'A1', 'A2', 'A3'],
                         'B': ['B0', 'B1', 'B2', 'B3']})

direita = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                        'C': ['C0', 'C1', 'C2', 'C3'],
                        'D': ['D0', 'D1', 'D2', 'D3']})
```

```
esquerda
```

	A	B	key
0	A0	B0	K0
1	A1	B1	K1
2	A2	B2	K2
3	A3	B3	K3

```
direita
```

	C	D	key
0	C0	D0	K0
1	C1	D1	K1
2	C2	D2	K2
3	C3	D3	K3

```
pd.merge(esquerda,direita,how='inner',on='key')
```

	A	B	key	C	D
0	A0	B0	K0	C0	D0
1	A1	B1	K1	C1	D1
2	A2	B2	K2	C2	D2
3	A3	B3	K3	C3	D3

Mesclar

Outro Exemplo

```
esquerda = pd.DataFrame({'key1': ['K0', 'K0', 'K1', 'K2'],
                         'key2': ['K0', 'K1', 'K0', 'K1'],
                            'A': ['A0', 'A1', 'A2', 'A3'],
                            'B': ['B0', 'B1', 'B2', 'B3']})

direita = pd.DataFrame({'key1': ['K0', 'K1', 'K1', 'K2'],
                                'key2': ['K0', 'K0', 'K0', 'K0'],
                                   'C': ['C0', 'C1', 'C2', 'C3'],
                                   'D': ['D0', 'D1', 'D2', 'D3']})
```

```
esquerda
```

	key1	key2	A	B
0	K0	K0	A0	B0
1	K0	K1	A1	B1
2	K1	K0	A2	B2
3	K2	K1	A3	B3

```
direita
```

	key1	key2	C	D
0	K0	K0	C0	D0
1	K1	K0	C1	D1
2	K1	K0	C2	D2
3	K2	K0	C3	D3

Mesclar

```
pd.merge(esquerda, direita, on=['key1', 'key2'])
```

	A	B	key1	key2	C	D
0	A0	B0	K0	K0	C0	D0
1	A2	B2	K1	K0	C1	D1
2	A2	B2	K1	K0	C2	D2

```
pd.merge(esquerda, direita, how='outer', on=['key1', 'key2'])
```

	A	B	key1	key2	C	D
0	A0	B0	K0	K0	C0	D0
1	A1	B1	K0	K1	NaN	NaN
2	A2	B2	K1	K0	C1	D1
3	A2	B2	K1	K0	C2	D2
4	A3	B3	K2	K1	NaN	NaN
5	NaN	NaN	K2	K0	C3	D3

Juntar

```
esquerda = pd.DataFrame({'A': ['A0', 'A1', 'A2'],
                         'B': ['B0', 'B1', 'B2']},
                         index=['K0', 'K1', 'K2'])

direita = pd.DataFrame({'C': ['C0', 'C2', 'C3'],
                        'D': ['D0', 'D2', 'D3']},
                         index=['K0', 'K2', 'K3'])
```

```
esquerda
```

	A	B
K0	A0	B0
K1	A1	B1
K2	A2	B2

```
direita
```

	C	D
K0	C0	D0
K2	C2	D2
K3	C3	D3

```
esquerda.join(direita)
```

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2

```
esquerda.join(direita, how='outer')
```

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2
K3	NaN	NaN	C3	D3

Valores Exclusivos

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

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

```
df['col2'].unique()
```

```
array([444, 555, 666], dtype=int64)
```

```
df['col2'].nunique()
```

```
3
```

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

```
444    2
555    1
666    1
Name: col2, dtype: int64
```

Selecionando Dados

```
# Selecione do DataFrame usando critérios de várias colunas
newdf = df[(df['col1']>2) & (df['col2']==444)]
```

```
newdf
```

	col1	col2	col3
3	4	444	xyz

Funções

```
def times2(x):
    return x*2
```

```
df['col1'].apply(times2)
```

```
0    2
1    4
2    6
3    8
Name: col1, dtype: int64
```

```
df['col3'].apply(len)
```

```
0    3
1    3
2    3
3    3
Name: col3, dtype: int64
```

```
df['col1'].sum()
```

```
10
```

Outros

** Removendo colunas permanentemente **

```
del df['col1']
```

```
df
```

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

** Obter nomes de coluna e índice: **

```
df.columns
```

```
Index(['col2', 'col3'], dtype='object')
```

```
df.index
```

```
RangeIndex(start=0, stop=4, step=1)
```

Outros

** Ordenando um DataFrame **

```
df
```

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

```
df.sort_values(by='col2') #inplace=False por padrão
```

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

Outros

** Encontre Valores Nulos ou Verifique Valores Nulos **

```
df.isnull()
```

	col2	col3
0	False	False
1	False	False
2	False	False
3	False	False

```
# Deleta linhas com valores NaN
df.dropna()
```

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

** Preenchendo os valores de NaN com outra coisa: **

```
import numpy as np
```

```
df = pd.DataFrame({'col1':[1,2,3,np.nan],
                   'col2':[np.nan,555,666,444],
                   'col3':['abc','def','ghi','xyz']})
df.head()
```

	col1	col2	col3
0	1.0	NaN	abc
1	2.0	555.0	def
2	3.0	666.0	ghi
3	NaN	444.0	xyz

```
df.fillna('Preencher')
```

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

11.

Entrada e Saída

DADOS

CSV

```
a,b,c,d
0,1,2,3
4,5,6,7
8,9,10,11
12,13,14,15
```

```
import numpy as np
import pandas as pd
```

```
df = pd.read_csv('exemplo')
df
```

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

```
df.to_csv('exemplo.csv')
```

Excel

```
dfe = pd.read_excel('Exemplo_Excel.xlsx')
dfe
```

	Nome	Cor	Valor
0	Lucília	Amarelo	52
1	Bruna	Azul	23
2	André	Verde	21

```
dfe.to_excel('Exemplo_saida.xlsx', 'Sheet1','utf-8')
```

Obrigada!

Alguma Pergunta?

professora@lucilia.com.br