

# pandas

Funcionalidades essenciais  
Aritmética e alinhamento de dados

Vamos ver um pouco sobre o comportamento da aritmética entre objetos com índices diferentes.

Quando realizamos uma operação matemática sobre dois objetos cujo par de índices não é igual, o resultado será a união dos pares de índices.

Para quem conhece bancos de dados, isso se assemelha a uma operação de junção externa (*outer join*) automática nos rótulos dos índices.

## Funcionalidades essenciais - Aritmética e alinhamento de dados

O alinhamento de dados interno introduz valores indicativos de ausência nos locais dos rótulos que não se sobrepõem. Esses valores serão propagados para outros cálculos aritméticos.

```
In [18]: import pandas as pd

In [19]: serie1 = pd.Series([18.7, 4.8, 6.6, -2.5], index=['um', 'dois', 'tres', 'quatro'])

In [21]: serie1
Out[21]: um      18.7
         dois    4.8
         tres    6.6
         quatro -2.5
         dtype: float64

In [4]: serie2 = pd.Series([77.1, 3.5, -4.87, 5.1], index=['um', 'cinco', 'seis', 'quatro'])

In [22]: serie2
Out[22]: um      77.10
         cinco   3.50
         seis   -4.87
         quatro  5.10
         dtype: float64
```

```
In [5]: serie1 + serie2
Out[5]: cinco      NaN
         dois      NaN
         quatro    2.6
         seis      NaN
         tres      NaN
         um       95.8
         dtype: float64
```

# pandas

## Funcionalidades essenciais - Aritmética e alinhamento de dados

No caso do *DataFrame*, o alinhamento é feito tanto nas linhas quanto nas colunas.

```
In [45]: import numpy as np
```

```
In [46]: dataframe1 = pd.DataFrame(np.arange(16.).reshape((4,4)), columns=list('abcd'), index=["ES", "RJ", "SP", "MG"])
```

```
In [47]: dataframe1
```

Out[47]:

	a	b	c	d
ES	0.0	1.0	2.0	3.0
RJ	4.0	5.0	6.0	7.0
SP	8.0	9.0	10.0	11.0
MG	12.0	13.0	14.0	15.0

```
In [48]: dataframe2 = pd.DataFrame(np.arange(9.).reshape((3,3)), columns=list('abc'), index=["ES", "RJ", "SP"])
```

```
In [49]: dataframe2
```

Out[49]:

	a	b	c
ES	0.0	1.0	2.0
RJ	3.0	4.0	5.0
SP	6.0	7.0	8.0

# pandas

## Funcionalidades essenciais - Aritmética e alinhamento de dados

Somar os dados dos *DataFrames* resulta em um *DataFrame* cujos índice e colunas são as uniões dos dados de cada *DataFrame*.

A coluna d não é comum aos dois *DataFrames*, por isso é exibida com dados ausentes.

A linha MG também não é comum aos dois *DataFrames*.

```
In [50]: dataframe1 + dataframe2
```

```
Out[50]:
```

	a	b	c	d
ES	0.0	2.0	4.0	NaN
MG	NaN	NaN	NaN	NaN
RJ	7.0	9.0	11.0	NaN
SP	14.0	16.0	18.0	NaN

# pandas

## Funcionalidades essenciais - Aritmética e alinhamento de dados

Se somarmos os objetos *DataFrame* sem rótulos para colunas ou linhas em comum, o resultado conterá somente nulos:

```
In [7]: dataframe1 = pd.DataFrame({'ES': [1, 2]})

In [8]: dataframe1
Out[8]:
```

	ES
0	1
1	2

```
In [9]: dataframe2 = pd.DataFrame({'RJ': [3, 4]})

In [10]: dataframe2
Out[10]:
```

	RJ
0	3
1	4

```
In [11]: dataframe1 - dataframe2
Out[11]:
```

	ES	RJ
0	NaN	NaN
1	NaN	NaN

### Métodos aritméticos com valores para preenchimento

Em vez de usar o operador de adição, podemos usar o método ***add*** para somar os *dataframes*. Veja o exemplo a seguir:

```
In [51]: dataframe1 = pd.DataFrame(np.arange(16.).reshape((4,4)), columns=list('abcd'), index=["ES", "RJ", "SP", "MG"])
```

```
In [52]: dataframe2 = pd.DataFrame(np.arange(9.).reshape((3,3)), columns=list('abc'), index=["ES", "RJ", "SP"])
```

```
In [53]: dataframe3 = dataframe1.add(dataframe2)
```

```
In [54]: dataframe3
```

Out[54]:

	a	b	c	d
ES	0.0	2.0	4.0	NaN
MG	NaN	NaN	NaN	NaN
RJ	7.0	9.0	11.0	NaN
SP	14.0	16.0	18.0	NaN

## Funcionalidades essenciais - Aritmética e alinhamento de dados

O resultado foi o mesmo obtido quando utilizamos a soma com o operador "+", porém, a vantagem do método *add*, é que podemos usar o parâmetro ***fill\_value*** para preencher os dados com um valor padrão, caso seus índices sejam diferentes. Veja abaixo, que, quando preenchemos com zero, a soma foi realizada corretamente, evitando assim os *NaN*:

```
In [55]: dataframe3 = dataframe1.add(dataframe2, fill_value=0)
```

```
In [56]: dataframe3
```

```
Out[56]:
```

	a	b	c	d
ES	0.0	2.0	4.0	3.0
MG	12.0	13.0	14.0	15.0
RJ	7.0	9.0	11.0	7.0
SP	14.0	16.0	18.0	11.0



Veja uma tabela com os métodos aritméticos:

Método	Descrição
add, radd	Métodos para adição (+)
sub, rsub	Métodos para subtração (-)
div, rdiv	Métodos para divisão (/)
floordiv, rfloordiv	Métodos para divisão inteira (//)
mul, rmul	Métodos para multiplicação (*)
pow, rpow	Métodos para exponenciação (**)

Os métodos que começam com a letra "r" invertem os argumentos.

Os métodos que começam com a letra "r" invertem os argumentos.

```
In [57]: df1 = pd.DataFrame([1, 2, 3])
```

```
In [58]: df2 = pd.DataFrame([4, 5, 6])
```

```
In [61]: df3 = df1.div(df2)
```

```
In [62]: df3
```

```
Out[62]:
```

	0
0	0.25
1	0.40
2	0.50

```
In [63]: df4 = df1.rdiv(df2)
```

```
In [64]: df4
```

```
Out[64]:
```

	0
0	4.0
1	2.5
2	2.0

### Operações entre DataFrames e Series

A aritmética entre *DataFrame* e *Series* de dimensões diferentes segue o mesmo conceito dos *arrays NumPy*. Considere por exemplo a diferença entre um *array* bidimensional e uma de suas linhas:

```
In [66]: arr = np.arange(12.).reshape((3,4))
```

```
In [67]: arr
```

```
Out[67]: array([[ 0.,  1.,  2.,  3.],  
               [ 4.,  5.,  6.,  7.],  
               [ 8.,  9., 10., 11.]])
```

```
In [68]: arr[0]
```

```
Out[68]: array([0., 1., 2., 3.])
```

```
In [69]: arr - arr[0]
```

```
Out[69]: array([[0., 0., 0., 0.],  
               [4., 4., 4., 4.],  
               [8., 8., 8., 8.]])
```

Quando subtraímos `arr[0]` de `arr`, a subtração é realizada uma vez para cada linha (isso é chamado de broadcasting).

# pandas

## Funcionalidades essenciais - Aritmética e alinhamento de dados

As operações entre um *DataFrame* e uma *Series* são semelhantes:

```
In [110]: frame = pd.DataFrame(np.arange(12.).reshape((4,3)), columns=list('bde'), index=["ES", "RJ", "SP", "MG"])
In [111]: frame
Out[111]:
```

	b	d	e
ES	0.0	1.0	2.0
RJ	3.0	4.0	5.0
SP	6.0	7.0	8.0
MG	9.0	10.0	11.0

```
In [112]: series = pd.Series([0., 1., 2.], name="ES", index=["b", "d", "e"])
          series
Out[112]: b    0.0
          d    1.0
          e    2.0
          Name: ES, dtype: float64
```

## Funcionalidades essenciais - Aritmética e alinhamento de dados

Por padrão, a aritmética entre *DataFrame* e *Series* realiza a correspondência entre o índice da *Series* e as colunas do *DataFrame*, fazendo *broadcasting* pelas linhas:

```
In [98]: subtracao = frame - series  
subtracao
```

Out[98]:

	b	d	e
ES	0.0	0.0	0.0
RJ	3.0	3.0	3.0
SP	6.0	6.0	6.0
MG	9.0	9.0	9.0

Estas foram as subtrações realizadas (linha a linha):

0.0 - 0.0 = 0.0	1.0 - 1.0 = 0.0	2.0 - 2.0 = 0.0
3.0 - 0.0 = 3.0	4.0 - 1.0 = 3.0	5.0 - 2.0 = 3.0
6.0 - 0.0 = 6.0	7.0 - 1.0 = 6.0	8.0 - 2.0 = 6.0
9.0 - 0.0 = 9.0	10.0 - 1.0 = 9.0	11.0 - 2.0 = 9.0

## Funcionalidades essenciais - Aritmética e alinhamento de dados

Se o valor de um índice não for encontrado nas colunas do *DataFrame* nem no índice de *Series*, os objetos serão reindexados para formar a união:

```
In [113]: frame
```

```
Out[113]:
```

	b	d	e
ES	0.0	1.0	2.0
RJ	3.0	4.0	5.0
SP	6.0	7.0	8.0
MG	9.0	10.0	11.0

```
In [114]: series2 = pd.Series(range(3), index=["b", "e", "f"])
series2
```

```
Out[114]: b      0
          e      1
          f      2
          dtype: int64
```

```
In [116]: soma = frame + series2
soma
```

```
Out[116]:
```

	b	d	e	f
ES	0.0	NaN	3.0	NaN
RJ	3.0	NaN	6.0	NaN
SP	6.0	NaN	9.0	NaN
MG	9.0	NaN	12.0	NaN

## Funcionalidades essenciais - Aritmética e alinhamento de dados

Se quisermos fazer *broadcast* pelas colunas, fazendo correspondências nas linhas, teremos que usar um dos métodos aritméticos. Veja um exemplo de subtração:

```
In [117]: frame
```

```
Out[117]:
```

	b	d	e
ES	0.0	1.0	2.0
RJ	3.0	4.0	5.0
SP	6.0	7.0	8.0
MG	9.0	10.0	11.0

```
In [118]: series3 = frame['d']  
series3
```

```
Out[118]: ES      1.0  
          RJ      4.0  
          SP      7.0  
          MG     10.0  
          Name: d, dtype: float64
```

```
In [120]: sub = frame.sub(series3, axis='index')  
sub
```

```
Out[120]:
```

	b	d	e
ES	-1.0	0.0	1.0
RJ	-1.0	0.0	1.0
SP	-1.0	0.0	1.0
MG	-1.0	0.0	1.0

Estas foram as subtrações realizadas (coluna a coluna):

0.0 - 1.0 = -1.0	1.0 - 1.0 = 0.0	2.0 - 1.0 = 1.0
3.0 - 4.0 = -1.0	4.0 - 4.0 = 0.0	5.0 - 4.0 = 1.0
6.0 - 7.0 = -1.0	7.0 - 7.0 = 0.0	8.0 - 7.0 = 1.0
9.0 - 10.0 = -1.0	10.0 - 10.0 = 0.0	11.0 - 10.0 = 1.0

# FIM