

pandas

Funcionalidades essenciais
Indexação, seleção e filtragem

A indexação de séries funciona como a indexação de arrays NumPy, porém, podemos utilizar os valores de índice da Series em vez de utilizar somente inteiros.

```
In [1]: import pandas as pd

In [4]: obj = pd.Series([0.,1.,2.,3.], index=['a','b','c','d'])

In [5]: obj
Out[5]: a    0.0
       b    1.0
       c    2.0
       d    3.0
       dtype: float64
```

```
In [4]: obj[1]
```

```
Out[4]: 1.0
```

```
In [5]: obj['b']
```

```
Out[5]: 1.0
```

O fatiamento usando rótulos, inclui o final, diferente do fatiamento com número dos índices.

```
In [12]: obj[1:4]
```

```
Out[12]: b    1.0
       c    2.0
       d    3.0
       dtype: float64
```

```
In [14]: obj['b':'d']
```

```
Out[14]: b    1.0
       c    2.0
       d    3.0
       dtype: float64
```

Podemos também retornar as linhas com base em seu valor.

```
In [27]: obj = pd.Series([0.,1.,2.,3.,2.,2.,1.], index=['a','b','c','d','e','f','g'])
```

```
In [25]: obj[obj == 2]
```

```
Out[25]: c    2.0  
         e    2.0  
         f    2.0  
         dtype: float64
```

```
In [26]: obj[obj < 3]
```

```
Out[26]: a    0.0  
         b    1.0  
         c    2.0  
         e    2.0  
         f    2.0  
         g    1.0  
         dtype: float64
```

Podemos atribuir um único valor a um intervalo conforme exemplo abaixo.

```
In [30]: obj['a':'c'] = 8.0
```

```
In [31]: obj
```

```
Out[31]: a      8.0  
         b      8.0  
         c      8.0  
         d      3.0  
         e      2.0  
         f      2.0  
         g      1.0  
         dtype: float64
```

pandas

Funcionalidades essenciais – Indexação, seleção e filtragem

Vamos agora ver a indexação em um DataFrame.

Primeiro vamos criar o nosso DataFrame.

```
import numpy as np
```

```
dados = pd.DataFrame(np.arange(16).reshape((4, 4)),  
                      index=["Vila Velha", "Vitória", "Viana", "Cariacica"],  
                      columns=["um", "dois", "três", "quatro"])
```

`dados`

	um	dois	três	quatro
Vila Velha	0	1	2	3
Vitória	4	5	6	7
Viana	8	9	10	11
Cariacica	12	13	14	15

Funcionalidades essenciais – Indexação, seleção e filtragem

Podemos indexar utilizando fatiamento e a seleção de dados com um array booleano, retornando linhas do DataFrame:

```
dados[:2]
```

	um	dois	três	quatro
Vila Velha	0	1	2	3
Vitória	4	5	6	7

```
dados[dados["um"] < 8]
```

	um	dois	três	quatro
Vila Velha	0	1	2	3
Vitória	4	5	6	7

Funcionalidades essenciais – Indexação, seleção e filtragem

Passar um único elemento ou uma lista para o operador [] seleciona colunas, veja:

```
dados["um"]
```

Vila Velha	0
Vitória	4
Viana	8
Cariacica	12
Name: um, dtype: int32	

```
dados[["um", "dois"]]
```

	um	dois
Vila Velha	0	1
Vitória	4	5
Viana	8	9
Cariacica	12	13

Funcionalidades essenciais – Indexação, seleção e filtragem

Outro caso de uso está na indexação com um DataFrame booleano, como aquele gerado por uma comparação escalar:

```
dados < 8
```

	um	dois	três	quatro
Vila Velha	True	True	True	True
Vitória	True	True	True	True
Viana	False	False	False	False
Cariacica	False	False	False	False

```
dados[dados < 8] = 0
```

```
dados
```

	um	dois	três	quatro
Vila Velha	0	0	0	0
Vitória	0	0	0	0
Viana	8	9	10	11
Cariacica	12	13	14	15

Seleção com loc e iloc

Os operadores loc e iloc permitem selecionar um subconjunto de linhas e colunas de um DataFrame com uma notação semelhante àquela do NumPy, usando rótulos de eixo (loc) ou inteiros (iloc).

Vamos selecionar uma linha com duas colunas de nosso DataFrame:

```
dados.loc['Viana', ['um', 'quatro']]
```

```
um      8  
quatro  11  
Name: Viana, dtype: int32
```

Agora vamos realizar seleções usando inteiros com `iloc`:

```
dados.iloc[2, [0,3]]
```

```
um      8  
quatro  11  
Name: Viana, dtype: int32
```

```
dados.iloc[2]
```

```
um      8  
dois    9  
três   10  
quatro  11  
Name: Viana, dtype: int32
```

```
dados.iloc[[2,3], [0,1,2]]
```

	um	dois	três
Viana	8	9	10
Cariacica	12	13	14

Tanto loc quanto iloc trabalham com fatias, além de rótulos únicos ou listas de rótulos:

```
dados.loc[:, 'Viana', 'um': 'três']
```

	um	dois	três
Vila Velha	0	1	2
Vitória	4	5	6
Viana	8	9	10

```
dados.iloc[:, :3][dados.um >= 8]
```

	um	dois	três
Viana	8	9	10
Cariacica	12	13	14

FIM