

pandas

Funcionalidades essenciais

Ordenação e Classificação

Funcionalidades essenciais – Ordenação


Podemos ordenar nossos conjuntos de dados pelas linhas ou pelas colunas usando o método ***sort_index()***, que devolve um novo objeto ordenado.

```
In [2]: import pandas as pd  
import numpy as np
```

```
In [3]: obj = pd.Series(range(5), index=['b','c','a','d','e'])
```

```
In [4]: obj.sort_index()
```


```
Out[4]: a    2  
b    0  
c    1  
d    3  
e    4  
dtype: int64
```



Para ordenar de forma decrescente, podemos adicionar o parâmetro `ascending=False` informando False.

```
In [10]: obj.sort_index(ascending=False)
```

```
Out[10]: e    4  
d    3  
c    1  
b    0  
a    2  
dtype: int64
```



Vamos agora trabalhar com ordenação em um DataFrame.

```
frame = pd.DataFrame(np.random.randn(5,5), columns=list('baced'),  
                      index=['Brasil', 'Paraguai', 'Uruguai',  
                             'Colômbia', 'Venezuela'])
```

frame

| | b | a | c | e | d |
|-----------|-----------|-----------|-----------|-----------|-----------|
| Brasil | -0.297170 | 0.121786 | -1.236958 | 2.039124 | 1.066947 |
| Paraguai | 0.267801 | 0.952138 | 0.829998 | -0.540277 | 2.316884 |
| Uruguai | 0.271144 | -0.347142 | -1.331291 | 1.139072 | -1.823690 |
| Colômbia | -1.699284 | -0.570392 | -0.451554 | 0.573667 | -0.544994 |
| Venezuela | 1.557113 | 1.063000 | -0.141228 | -1.115673 | 1.872851 |

```
frame.sort_index()
```


| | b | a | c | e | d |
|-----------|-----------|-----------|-----------|-----------|-----------|
| Brasil | -0.297170 | 0.121786 | -1.236958 | 2.039124 | 1.066947 |
| Colômbia | -1.699284 | -0.570392 | -0.451554 | 0.573667 | -0.544994 |
| Paraguai | 0.267801 | 0.952138 | 0.829998 | -0.540277 | 2.316884 |
| Uruguai | 0.271144 | -0.347142 | -1.331291 | 1.139072 | -1.823690 |
| Venezuela | 1.557113 | 1.063000 | -0.141228 | -1.115673 | 1.872851 |

```
frame.sort_index(ascending=False)
```

| | b | a | c | e | d |
|-----------|-----------|-----------|-----------|-----------|-----------|
| Venezuela | 1.557113 | 1.063000 | -0.141228 | -1.115673 | 1.872851 |
| Uruguai | 0.271144 | -0.347142 | -1.331291 | 1.139072 | -1.823690 |
| Paraguai | 0.267801 | 0.952138 | 0.829998 | -0.540277 | 2.316884 |
| Colômbia | -1.699284 | -0.570392 | -0.451554 | 0.573667 | -0.544994 |
| Brasil | -0.297170 | 0.121786 | -1.236958 | 2.039124 | 1.066947 |


Podemos usar o parâmetro `axis` para definir o eixo de ordenação.

```
frame.sort_index(axis=0)
```



| | b | a | c | e | d |
|-----------|-----------|-----------|-----------|-----------|-----------|
| Brasil | -0.297170 | 0.121786 | -1.236958 | 2.039124 | 1.066947 |
| Colômbia | -1.699284 | -0.570392 | -0.451554 | 0.573667 | -0.544994 |
| Paraguai | 0.267801 | 0.952138 | 0.829998 | -0.540277 | 2.316884 |
| Uruguai | 0.271144 | -0.347142 | -1.331291 | 1.139072 | -1.823690 |
| Venezuela | 1.557113 | 1.063000 | -0.141228 | -1.115673 | 1.872851 |

```
frame.sort_index(axis=1)
```



| | a | b | c | d | e |
|-----------|-----------|-----------|-----------|-----------|-----------|
| Brasil | 0.121786 | -0.297170 | -1.236958 | 1.066947 | 2.039124 |
| Paraguai | 0.952138 | 0.267801 | 0.829998 | 2.316884 | -0.540277 |
| Uruguai | -0.347142 | 0.271144 | -1.331291 | -1.823690 | 1.139072 |
| Colômbia | -0.570392 | -1.699284 | -0.451554 | -0.544994 | 0.573667 |
| Venezuela | 1.063000 | 1.557113 | -0.141228 | 1.872851 | -1.115673 |

Funcionalidades essenciais – Ordenação

Para ordenar uma Series de acordo com seus valores, utilizamos o método ***sort_values()***.

```
obj = pd.Series([10, np.nan, 54, 7, -2, np.nan, -4])
```

obj

| | |
|---|------|
| 0 | 10.0 |
| 1 | NaN |
| 2 | 54.0 |
| 3 | 7.0 |
| 4 | -2.0 |
| 5 | NaN |
| 6 | -4.0 |

dtype: float64

```
obj.sort_values()
```

| | |
|---|------|
| 6 | -4.0 |
| 4 | -2.0 |
| 3 | 7.0 |
| 0 | 10.0 |
| 2 | 54.0 |
| 1 | NaN |
| 5 | NaN |

dtype: float64

Funcionalidades essenciais – Ordenação

Quanto ao DataFrame, podemos ordenar usando os dados de uma ou mais colunas, passando o nome das colunas como parâmetro para a opção `by` de `sort_values`. Assim: **`sort_values(by='coluna')`** ou **`sort_values(by=['coluna1','coluna2'])`**.

```
frame = pd.DataFrame(np.random.randn(5,5), columns=list('baced'),  
                      index=['Brasil', 'Paraguai', 'Uruguai',  
                             'Colômbia', 'Venezuela'])
```

frame

| | b | a | c | e | d |
|-----------|-----------|-----------|-----------|-----------|-----------|
| Brasil | -0.426949 | -0.669015 | -0.777855 | 0.228096 | -1.072470 |
| Paraguai | 2.686047 | 1.388839 | 0.165097 | -0.412211 | 1.037880 |
| Uruguai | -0.173377 | -1.078226 | 1.031416 | -2.341744 | 1.095723 |
| Colômbia | -0.342928 | 0.802037 | 0.627536 | -0.877820 | -0.714052 |
| Venezuela | -0.375989 | -1.334716 | -0.232672 | -0.480826 | -0.418742 |

```
frame.sort_values(by='c')
```

| | b | a | c | e | d |
|-----------|-----------|-----------|-----------|-----------|-----------|
| Brasil | -0.426949 | -0.669015 | -0.777855 | 0.228096 | -1.072470 |
| Venezuela | -0.375989 | -1.334716 | -0.232672 | -0.480826 | -0.418742 |
| Paraguai | 2.686047 | 1.388839 | 0.165097 | -0.412211 | 1.037880 |
| Colômbia | -0.342928 | 0.802037 | 0.627536 | -0.877820 | -0.714052 |
| Uruguai | -0.173377 | -1.078226 | 1.031416 | -2.341744 | 1.095723 |

```
frame.sort_values(by=['a','b'])
```

| | b | a | c | e | d |
|-----------|-----------|-----------|-----------|-----------|-----------|
| Venezuela | -0.375989 | -1.334716 | -0.232672 | -0.480826 | -0.418742 |
| Uruguai | -0.173377 | -1.078226 | 1.031416 | -2.341744 | 1.095723 |
| Brasil | -0.426949 | -0.669015 | -0.777855 | 0.228096 | -1.072470 |
| Colômbia | -0.342928 | 0.802037 | 0.627536 | -0.877820 | -0.714052 |
| Paraguai | 2.686047 | 1.388839 | 0.165097 | -0.412211 | 1.037880 |

Funcionalidades essenciais – Classificação

A classificação (rank) atribui posições iniciando em 1, indo até o número de pontos de dados válidos em um array. Por padrão, rank() resolve empates atribuindo a cada grupo a classificação média.

```
obj = pd.Series([7, -5, 7, 4, 2, 0, 4])
```

obj

| | |
|---|----|
| 0 | 7 |
| 1 | -5 |
| 2 | 7 |
| 3 | 4 |
| 4 | 2 |
| 5 | 0 |
| 6 | 4 |

dtype: int64

obj.rank()

| | |
|---|-----|
| 0 | 6.5 |
| 1 | 1.0 |
| 2 | 6.5 |
| 3 | 4.5 |
| 4 | 3.0 |
| 5 | 2.0 |
| 6 | 4.5 |

dtype: float64

primeiro

segundo

```
obj.rank(method='first')
```

| | |
|---|-----|
| 0 | 6.0 |
| 1 | 1.0 |
| 2 | 7.0 |
| 3 | 4.0 |
| 4 | 3.0 |
| 5 | 2.0 |
| 6 | 5.0 |

dtype: float64

Podemos classificar de acordo com a ordem em que os dados são observados. Neste caso, em vez de usar a classificação média 4.5 para as entradas 3 e 6, elas foram definidas com 4 e 5, porque o rótulo 3 antecede ao 6.

Funcionalidades essenciais – Classificação

Podemos classificar de forma decrescente informando `ascending=False`.

```
obj = pd.Series([0, 1, 2, 3, 4, 5, 6])  
  
obj.rank(ascending=False)  
  
0    7.0  
1    6.0  
2    5.0  
3    4.0  
4    3.0  
5    2.0  
6    1.0  
dtype: float64
```

| Método | Descrição |
|---------|---|
| average | Default: Atribui a classificação média para cada entrada no mesmo grupo. |
| min | Utiliza a classificação mínima do grupo todo |
| max | Utiliza a classificação máxima do grupo todo |
| first | Atribui classificações na ordem em que os valores aparecem nos dados |
| dense | Como o 'min', porém as classificações sempre aumentam de 1 entre grupos, em vez do número de elementos iguais em um grupo |

FIM