


Truques Pandas

Como processar Big Data com pandas? Aprenda sobre funções como "between", "reindex" e muito mais.

1. `between()`

pandas.Series.between - pandas 1.2.0 documentation

 <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.between.html>

Digamos que temos um DataFrame com preços e gostaríamos de filtrar os preços entre 2 e 4.

```
df = pd.DataFrame({'price': [1.99, 3, 5, 0.5, 3.5, 5.5, 3.9]})
```

Com `entre` a função, você pode reduzir este filtro:

```
df[(df.price >= 2) & (df.price <= 4)]
```

Para isso:

```
df[df.price.between(2, 4)]
```


	price
1	3.0
4	3.5
6	3.9

Pode não parecer muito, mas esses parênteses são irritantes ao escrever muitos filtros. O filtro com `entre` as funções também é mais legível.

2. `reindex()`

pandas.DataFrame.reindex - pandas 1.2.0 documentation

Conform Series/DataFrame to new index with optional filling logic. Places NA/NaN in locations having no value in the previous index. A new object is produced unless the new index is equivalent to the current one and . Parameters keywords

 <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.reindex.html>

Ao especificar a ordem dos tamanhos na função reindex torna uma tabela de resultados mais fácil de interpretar.

A função reindex está em conformidade com uma série ou um DataFrame a um novo índice.

Vamos adicionar tamanhos de camisetas ao nosso Dataframe. O objetivo da análise é calcular o preço médio de cada tamanho:

```
df = pd.DataFrame({'price': [1.99, 3, 5],
                  'size': ['medium', 'large', 'small']})

df_avg = df.groupby('size').price.mean()

df_avg
```

```
size
large    3.00
medium   1.99
small    5.00
Name: price, dtype: float64
```

Os tamanhos têm uma ordem aleatória na tabela acima. Deve ser encomendado: **pequeno, médio, grande**. Aí vem a função de reindex para o resgate:


```
df_avg.reindex(['small', 'medium', 'large'])
```

```
size
small    5.00
medium   1.99
large    3.00
Name: price, dtype: float64
```

3.describe()

pandas.DataFrame.describe - pandas 1.2.0 documentation

Generate descriptive statistics. Descriptive statistics include those that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding values. Analyzes both numeric and object series, as well as column sets

 <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.describe.html>

describe() a função é uma ferramenta essencial ao trabalhar na Análise de Dados Exploratórios. Ele mostra estatísticas de resumo básicas para todas as colunas em um DataFrame.

```
df.price.describe()
```

```
count    3.000000
mean     3.330000
std      1.531894
min      1.990000
25%      2.495000
50%      3.000000
75%      4.000000
max      5.000000
Name: price, dtype: float64
```

E se quisermos calcular 10 quantiles em vez de 3?

```
df.price.describe(percentiles=np.arange(0, 1, 0.1))
```

```
count    3.000000
mean     3.330000
std      1.531894
min      1.990000
0%       1.990000
10%      2.192000
20%      2.394000
30%      2.596000
40%      2.798000
50%      3.000000
60%      3.400000
70%      3.800000
80%      4.200000
90%      4.600000
max      5.000000
Name: price, dtype: float64
```


Descrever função requer argumento percentis. Podemos especificar o número de percentis com a função de arange do NumPy para evitar digitar cada percentil à mão.

Esse recurso torna-se realmente útil quando combinado com o grupo por função:

```
df.groupby('size').describe(percentiles=np.arange(0, 1, 0.1))
```

4. contains()

pandas.Series.str.contains - pandas 1.2.0 documentation

Test if pattern or regex is contained within a string of a Series or Index. Return boolean Series or Index based on whether a given pattern or regex is contained within a string of a Series or Index. Parameters Character sequence or  <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.str.contains.html>

Nosso conjunto de dados de camisetas tem 3 tamanhos. Digamos que gostaríamos de **filtrar tamanhos pequenos e médios**. Uma maneira complicada de filtrar é:

```
df[(df['size'] == 'small') | (df['size'] == 'medium')]
```

Isso é ruim porque geralmente combinamos com outros filtros, o que torna a expressão ilegível. Existe uma maneira melhor?

As colunas de cordas pandas têm um acessório "str", que implementa muitas funções que simplificam a manipulação de cordas.

Uma delas é a função "contains", que suporta pesquisa com expressões regulares.

```
df[df['size'].str.contains('small|medium')]
```

O filtro com função "contains" é mais legível, mais fácil de estender e combinar com outros filtros.

5. usecols

Ao fazer uma análise, você geralmente não precisa de todas as linhas ou todas as colunas no conjunto de dados.

Em um caso, você não precisa de todas as linhas, você pode ler o conjunto de dados em pedaços e filtrar linhas desnecessárias para reduzir o uso da memória:

```
iter_csv = pd.read_csv('dataset.csv', iterator=True, chunksize=1000)

df = pd.concat([chunk[chunk['field'] > constant] for chunk in iter_csv])
```


Ler um conjunto de dados em pedaços é mais lento do que ler tudo uma vez. Eu recomendaria usar esta abordagem apenas com conjuntos de dados maiores do que a memória.

Em um caso, você não precisa de todas as colunas, **você pode especificar colunas necessárias com argumento "usecols"** ao ler um conjunto de dados:

```
df = pd.read_csv('file.csv', usecols=['col1', 'col2'])
```

6. Como recuperar um valor de coluna específico de um Dataframe

pandas.DataFrame.loc - pandas 1.2.0 documentation

Access a group of rows and columns by label(s) or a boolean array. is primarily label based, but may also be used with a boolean array. Allowed inputs are: A single label, e.g. or , (note that is interpreted as a label of the index, and  <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.loc.html>

```
import pandas as pd
df = pd.DataFrame({"col1": ["A0", "A1", "A0", "A2", "A0", "A1"]})
```

	col1
0	A0
1	A1
2	A0
3	A2
4	A0
5	A1

Existem duas maneiras de recuperar valores de coluna de um DataFrame:

- com base na indexação de localização de inteiros (iloc),
- com base na indexação do rótulo (loc).

Para recuperar o valor da primeira coluna para a coluna col1 (indexação de localização de inteiros):

```
df.iloc[0].col1  
# output: 'A0'
```

Para recuperar o valor da última coluna:

```
df.iloc[-1].col1  
# output: 'A1'
```

Você também pode **recuperar um valor de coluna específica com o rótulo**. Por padrão, pandas DataFrame tem índices de 0 a n, onde n é o número de linhas em um DataFrame.

```
df.loc[1, "col1"]  
# output: 'A1'
```

7. Como alterar o valor de uma coluna em um Dataframe

Da mesma forma que o exemplo acima, você pode alterar o valor de uma coluna com funções `loc` e `iloc`. Qual você usa depende se você tiver a posição inteiro da linha (`iloc`) ou rótulo da linha (`loc`).

Para **alterar o valor da primeira coluna** com base na posição de inteiro:

```
df.iloc[0].col1 = "First"
```

Para alterar o valor da última coluna com base na posição de inteiro:

```
df.iloc[-1].col1 = "Last"
```

Para alterar o valor da coluna por rótulo:

```
df.loc[2, "col1"] = "Index"
```

Aplicando essas alterações, obremos o seguinte Dataframe:

	col1
0	First
1	A1
2	Index
3	A2
4	A0
5	Last

8. A maneira adequada de atribuir uma nova coluna a um DataFrame **`assign()`**

Digamos que gostaríamos de adicionar uma lista de inteiros como uma coluna ao nosso DataFrame.

```
new_column = list(range(len(df)))
new_column

# output: [0, 1, 2, 3, 4, 5]
```

Como NÃO adicionar uma coluna (não funciona):

```
df.new_column1 = new_column
```

Também **NÃO é uma maneira adequada de adicionar uma coluna**, mas funciona. Você pode receber um aviso de cópia.

```
df["new_column2"] = new_column
```

Uma **boa maneira de adicionar uma nova coluna**:

```
df.loc[:, 'new_column3'] = new_column
```

A **maneira correta de adicionar uma nova coluna**:

```
df = df.assign(new_column4=new_column)
```

atribuir função também toma um índice como um argumento com o qual podemos definir a ordem das linhas na coluna.

Você pode ser tentado a adicionar o argumento `inplace=True` à função `atribuir` para adicionar uma nova coluna no lugar, mas não funcionará.

A função `assign` também adicionará a coluna `in place`.

```
# This will add two columns
df.assign(new_column4=new_column, inplace=True)
```

Este é, na verdade, outro **benefício da função `assing()`**. Você **pode adicionar várias colunas ao mesmo tempo**.

Veja a documentação dos pandas, para saber mais sobre a função de atribuição `assign()`

9. Recuperar uma série ou um DataFrame

Ao usar suportes quadrados com pandas DataFrame, ele retorna:

- uma série se você usar suportes quadrados únicos
- um Dataframe se você usar suportes quadrados duplos.

Como recuperar uma série de um Dataframe?

```
df["col1"]
```

```
0    A0
1    A1
2    A0
3    A2
4    A0
5    A1
Name: col1, dtype: object
```

```
type(df["col1"])

# pandas.core.series.Series
```

Como recuperar um DataFrame (com um subconjunto de colunas) a partir de um Dataframe?


```
df[["col1"]]
```

	col1
0	A0
1	A1
2	A0
3	A2
4	A0
5	A1

```
type(df[["col1"]])
# pandas.core.frame.DataFrame
```

10.to_frame() transformar Séries em DataFrame

pandas.Series.to_frame - pandas 1.2.0 documentation

 https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.to_frame.html

Ao realizar um agrupamento com agregação, pandas retorna uma Série.

Qual é a maneira mais simples de converter uma série de volta a um Dataframe?

```
df = pd.DataFrame(
    {
        "col1": ["A0", "A1", "A0", "A2", "A0", "A1"],
        "col2": [1, 2, 3, 4, 5, 6],
    }
)
df_group = df.groupby("col1").col2.sum()
df_group
```

```
col1
A0    9
A1    8
A2    4
Name: col2, dtype: int64
```

`df_group` é uma série.

Série tem uma função `to_frame`, que pode encobrir uma série de volta ao `DataFrame`:

```
df_group.to_frame()
```

	col2
col1	
A0	9
A1	8
A2	4

Veja a documentação dos pandas, para saber mais sobre `to_frame()` função.