#867 #DesenvolveDados Seg • Qua • Sex

Técnicas de Programação I

Conteúdo

3 Pandas

1. Pandas

O `pandas` é uma biblioteca muito versátil e simples de se utilizar quando vamos trabalhar com tabelas.
Para entender o funcionamento do `pandas` precisamos entender suas estruturas base, a `Series` e o `DataFrame`.

1.1 Series

As `Series` são basicamente as colunas das tabelas e armazenam suas informações com o `numpy array`. Neste caso, porém, esse array possuirá um índice associado, permitindo o acesso aos conteúdos dessa estrutura por ele.

1.1.1 Criação de uma Series

A partir de uma lista
Vamos ver abaixo, como criar uma Series à partir de uma lista:

```
import pandas as pd

minha_lista = [10, 20, 30]
serie = pd.Series(minha_lista)
```

O conteúdo de `serie` será uma `pd.Series` com os elementos da lista e como não foram definidos índices, os valores serão default, a numeração sequencial iniciada por zero.

A partir de um Array
O processo é igual ao anterior, com a diferença que ao invés de utilizarmos uma lista, utilizaremos um `np.array`.

Criação de uma Series com índice
Nesse caso, iremos passar duas listas, uma contendo os valores e outra contendo os índices:

```
import pandas as pd

labels = ['a', 'b', 'c', 'd']
```

```
valores = [10, 20, 30, 40]
serie = pd.Series(data=valores, index=labels)
```

1.1.2 Acessando elementos da Series

Pelo índice
Uma das maneiras de se acessar valores das nossas Series, é sabendo em qual índice eles se encontram:

```
import pandas as pd

labels = ['a', 'b', 'c', 'd']
valores = [10, 20, 30, 40]
serie = pd.Series(data=valores, index=labels)

serie['c']
```

Nesse exemplo acessamos o elemento `30`, que está associado ao índice `c`.

Utilizando filtros
Podemos aplicar filtros para selecionar apenas os elementos que satisfaçam determinada condição.
No exemplo abaixo, iremos selecionar apenas os elementos que sejam maiores que 15:

```
import pandas as pd

labels = ['a', 'b', 'c', 'd']
valores = [10, 20, 30, 40]
serie = pd.Series(data=valores, index=labels)

serie[serie > 15]
```

Note que `serie > 15` nos retorna uma series com elementos `True` e `False`, caso os elementos da `serie` satisfaçam a condição. Ao utilizar esse comando dentro dos colchetes, `serie[serie > 15]`, estamos selecionado apenas os elementos que satisfazem a condição.

1.1.3 Métodos

O Pandas possui diversos métodos que podem ser utilizados nessa estrutura.
Abaixo estão alguns métodos que essa estrutura de dados possui e facilitam alguns cálculos:

Método	Descrição
`sum`	soma
`mean`	média
`std`	desvio padrão

Método	Descrição
`mode`	moda
`max`	valor máximo
`min`	valor mínimo
`value_counts`	contagem de valores
`describe`	estatísticas básicas

Exemplos de utilização

1) Neste exemplo iremos utilizar o método `sum` para somar os valores da série.

```
import pandas as pd

valores = [1, 1, 2, 3, 5, 8, 13]
fibonacci = pd.Series(valores)

fibonacci.sum()
```

2) Podemos utilizar também os filtros, de maneira a soma apenas os valores maiores que 4.

```
import pandas as pd

valores = [1, 1, 2, 3, 5, 8, 13]
fibonacci = pd.Series(valores)

fibonacci[fibonacci > 4].sum()
```

1.2 DataFrame

O `DataFrame` é a estrutura que se assemelha à tabela. Ela é representada por um dicionário em que a chave é o nome da coluna e os valores são as `Series` (todas com mesmo índice).

1.1.1 Criação de um DataFrame

Existem diversas maneiras de se criar um dataframe, pode ser à partir de listas, dicionários etc. Um dos modos mais comuns é a criação à partir da leitura de um arquivo do formato `.csv`, como veremos à seguir para o caso do dataset `titanic`, muito conhecido por quem trabalha com data science.

Dataframe à partir de um csv

```
import pandas as pd

df_titanic = pd.read_csv('../datasets/titanic.csv')
```

Note que `titanic_df` é um `DataFrame` com os dados do arquivo `titanic.csv` localizados em `../datasets/`. Ou seja, o parâmetro do método `pd.read_csv` é o arquivo (com a localização) que se deseja ler. Existem outros parâmetros, mas não entraremos neles neste momento.

> ***Observação:***
> *Note que podemos também utilizar um arquivo do formato `.xlsx`, natural do excel.*
> *Para tanto, devemos utilizar o método `pd.read_excel`.*

DataFrame à partir de um dicionário
Este é um método muito útil, pois a estrutura do `dicionário` é bem semelhante à de um `DataFrame`. Neste caso, cada **chave** do nosso dicionário se tornará uma coluna e os **valores** (que podem ser na forma de listas, arrays, series...) serão os elementos do DF.

```
import pandas as pd

dicionario = {
    'coluna_A': [1, 2, 3, 4, 5],
    'coluna_B': ['a', 'b', 'c', 'd', 'e'],
    'coluna_C': [0.5, 1.5, 4.5, 6.5, 8.5]
}

df = pd.DataFrame(dicionario)
```

1.1.2 Acessando elementos do DataFrame

Existem diversas maneiras de se acessar valores de um `DataFrame`, veremos a seguir algumas maneiras principais de o fazer.

Selecionando apenas algumas colunas
Esse é o método mais simples, entretanto muito útil, para se selecionar apenas algumas colunas da nossa estrutura.
No exemplo que segue, iremos selecionar apenas as colunas `PassengerId, Name, Sex e Survived`.

```
import pandas as pd

df_titanic = pd.read_csv('../datasets/titanic.csv')
df_titanic[['PassengerId', 'Name', 'Sex', 'Survived']]
```

Acessando pela posição
O método a seguir se chama `iloc`, com ele podemos acessar os elementos do df em questão através das posições das linhas e colunas.
Neste exemplo, iremos selecionar o elemento da linha 2 e coluna 5.

```
import pandas as pd

df_titanic = pd.read_csv('../datasets/titanic.csv')
```

```
df_titanic.iloc[1, 4]
```

Acessando pelos índices De maneira bem semelhante à anterior, podemos acessar os elementos pelos índices e pelos nomes das colunas utilizando o método `loc`. Neste exemplo, iremos selecionar o elemento do índice 4 (linha 5) e coluna 'Name'.

```
import pandas as pd

df_titanic = pd.read_csv('../datasets/titanic.csv')

df_titanic.loc[4, 'Name']
```

Utilizando filtros
De maneira análoga às `Series`, podemos utilizar os filtros também nos `DataFrames`. No exemplo abaixo, iremos selecionar apenas os passageiros que tenham mais que 18 anos:

```
import pandas as pd

df_titanic = pd.read_csv('../datasets/titanic.csv')

df_titanic[df_titanic['Age'] > 18]
```

> *De maneira alternativa, podemos utilizar o método `query` do Pandas, como segue:*

```
df_titanic.query('Age > 19')
```

1.1.3 Tratando os dados

É muito comum num conjunto de dados, seja ele proveniente de um banco dados ou de um arquivo de texto, existirem valores nulos. Para fins de análises/modelos é muito importante identificar a incidência desses valores e tomar alguma atitude, seja a de remover os valores nulos, ou a de substituí-los. Veremos abaixo como os fazer:

Identificando Elementos Nulos por Coluna
Identificar a quantidade de nulos por coluna é muito importante, pois assim podemos identificar qual ação é mais adequada.

```
import pandas as pd

df_titanic = pd.read_csv('../datasets/titanic.csv')

df_titanic.isnull().sum()
```

Removendo os valore Nulos
Para remover os nulos, iremos utilizar o comando `dropna`, como segue:

```
import pandas as pd

df_titanic = pd.read_csv('../datasets/titanic.csv')

df_titanic.dropna()
```

Substituindo Valores Nulos
Como muitas vezes não queremos diminuir o tamanho do nosso conjunto de dados e mesmo assim utilizá-los (muitos modelos não aceitam valores nulos), uma abordagem é substituir esses valores (seja pela média dos valores, pela moda etc.).
Para fazer isso, utilizaremos o método `fillna`, em que o parâmetro passado será o valor de substituição.
Neste exemplo irei substituir os valores por `-1`, mas poderia ser qualquer outro valor, até mesmo uma string.

```
import pandas as pd

df_titanic = pd.read_csv('../datasets/titanic.csv')

df_titanic.fillna(-1)
```

1.1.4 Agrupando valores

Quando estamos fazendo análises no nosso conjunto de dados, é muito útil saber alguns comportamentos dados pela combinação de duas ou mais variáveis.
Para tanto, vamos utilizar o método `groupby` do Pandas.

Utilizando o Groupby
Neste exemplo, iremos analisar a média de idade por sexo.

```
import pandas as pd

df_titanic = pd.read_csv('../datasets/titanic.csv')

df_titanic.groupby('Sex')['Age'].mean()
```

> *Note que nesse exemplo utilizamos a média, mas poderíamos utilizar outras funções de agregação que vimos no item **1.1.3**, os métodos das Series.*

O comando pivot_table
Note que podemos querer cruzar mais informações, como por exemplo a média de idade por sexo e classe.
Para tanto utilizamos o método `pivot_table`, que é semelhante à tabela dinâmica do Excel.

```
import pandas as pd

df_titanic = pd.read_csv('../datasets/titanic.csv')
```

```
pd.pivot_table(df_titanic, values='Age', index='Pclass', columns='Sex',
aggfunc=np.mean)
```

1.1.5 Salvando os DFs em arquivos

É muito útil, depois de se tratar um conjunto de dados, salvar esse DataFrame num arquivo de texto.
Veremos à seguir duas maneiras de se salvar, num `.csv` e num `.xlsx`.

Salvando em csv

```
import pandas as pd

df_titanic = pd.read_csv('../datasets/titanic.csv')


df_titanic.to_csv('arquivo_com_dataframe.csv')
```

Salvando em xlsx

```
import pandas as pd

df_titanic = pd.read_csv('../datasets/titanic.csv')


df_titanic.to_excel('arquivo_com_dataframe.xlsx')
```

‹ Tópico anterior

Próximo Tópico ›