

---

# Filtros em DataFrames

---

# Projeção de DataFrames

**Extração de partes** de um **DataFrame**  
alocados em um novo DataFrame

```
import pandas as pd
import seaborn as sns

iris = sns.load_dataset("iris")

iris2 = iris[['species', 'petal_length', 'petal_width']]

iris2.head()
```

|   | species | petal_length | petal_width |
|---|---------|--------------|-------------|
| 0 | setosa  | 1.4          | 0.2         |
| 1 | setosa  | 1.4          | 0.2         |
| 2 | setosa  | 1.3          | 0.2         |
| 3 | setosa  | 1.5          | 0.2         |
| 4 | setosa  | 1.4          | 0.2         |

# Projeção de DataFrames

É possível utilizar **condições** para ser mais **seletivo**

```
import pandas as pd
import seaborn as sns

iris = sns.load_dataset("iris")


condicao_linha = (iris['species'] == "virginica") &
(iris['petal_length'] > 4.35 )

# Condição para as linhas
iris_novo = iris.loc[condicao_linha]

# Condição para as linhas e seleção de colunas
iris_novo2 = iris.loc[condicao_linha, ['species',
'petal_length', 'petal_width']]
```

# Projeção de DataFrames

É possível utilizar **condições** com o comando **query**



```
iris_novo = iris.query("species ==  
'virginica' & petal_length >4.35 ")
```

# Selecionando Colunas

Podemos selecionar as colunas de acordo com seus tipos com o comando **select\_dtypes()**.

Selecionando apenas colunas numéricas:

```
iris_numerico =  
iris.select_dtypes(include=['float64'])  
  
iris_numerico.head()
```

|   | sepal_length | sepal_width | petal_length | petal_width |
|---|--------------|-------------|--------------|-------------|
| 0 | 5.1          | 3.5         | 1.4          | 0.2         |
| 1 | 4.9          | 3.0         | 1.4          | 0.2         |
| 2 | 4.7          | 3.2         | 1.3          | 0.2         |
| 3 | 4.6          | 3.1         | 1.5          | 0.2         |
| 4 | 5.0          | 3.6         | 1.4          | 0.2         |

# Selecionando Colunas

Podemos também remover as colunas de acordo com seus tipos com o **`select_dtypes()`**.

Excluindo apenas colunas do tipo object:

```
iris_numerico =  
iris.select_dtypes(exclude=['object'])  
  
iris_numerico.head()
```

|   | sepal_length | sepal_width | petal_length | petal_width |
|---|--------------|-------------|--------------|-------------|
| 0 | 5.1          | 3.5         | 1.4          | 0.2         |
| 1 | 4.9          | 3.0         | 1.4          | 0.2         |
| 2 | 4.7          | 3.2         | 1.3          | 0.2         |
| 3 | 4.6          | 3.1         | 1.5          | 0.2         |
| 4 | 5.0          | 3.6         | 1.4          | 0.2         |

# Reiniciando Índices

Observe que os **índices se mantêm**, mesmo após criarmos um novo DataFrame.

|     | sepal_length | sepal_width | petal_length | petal_width | species   |
|-----|--------------|-------------|--------------|-------------|-----------|
| 100 | 6.3          | 3.3         | 6.0          | 2.5         | virginica |
| 101 | 5.8          | 2.7         | 5.1          | 1.9         | virginica |
| 102 | 7.1          | 3.0         | 5.9          | 2.1         | virginica |
| 103 | 6.3          | 2.9         | 5.6          | 1.8         | virginica |
| 104 | 6.5          | 3.0         | 5.8          | 2.2         | virginica |
| 105 | 7.6          | 3.0         | 6.6          | 2.1         | virginica |
| 106 | 4.9          | 2.5         | 4.5          | 1.7         | virginica |
| 107 | 7.3          | 2.9         | 6.3          | 1.8         | virginica |
| 108 | 6.7          | 2.5         | 5.8          | 1.8         | virginica |

# Reiniciando Índices

Observe que os **índices se mantêm**, mesmo após criarmos um novo DataFrame.

```
# Guarda o index antigo como uma coluna
iris_novo.reset_index()

# Não guarda o index antigo
iris_novo.reset_index(drop=True)
```

|   | index | sepal_length | sepal_width | petal_length | petal_width | species   |
|---|-------|--------------|-------------|--------------|-------------|-----------|
| 0 | 100   | 6.3          | 3.3         | 6.0          | 2.5         | virginica |
| 1 | 101   | 5.8          | 2.7         | 5.1          | 1.9         | virginica |
| 2 | 102   | 7.1          | 3.0         | 5.9          | 2.1         | virginica |
| 3 | 103   | 6.3          | 2.9         | 5.6          | 1.8         | virginica |
| 4 | 104   | 6.5          | 3.0         | 5.8          | 2.2         | virginica |



---

# Exercícios

---

cubos  
//academy//

---

# Substituindo e Compreendendo Valores

---

cubos  
//academy//

# Substituição de Valores

É útil para fazer a **manutenção de uma base de forma eficiente**, casos onde será necessário utilizar:

- Bases com **dados faltantes** representados por outros valores como -1 e -999, para NaN ou Null;
- Bases com **valores discrepantes** como "BA" e "Bahia" para se referir a mesma coisa;
- Bases que são precisas com **valores** que precisam ser **padronizados**.

# fillna()

**Substitui valores faltantes por novos valores.** Importante, esse método não sobrescreve o DataFrame.

**Argumento:** Valor que irá **substituir** os valores faltantes

```
ping['bill_length_mm'].count()  
  
#342  
  
#substituindo valores faltantes por zero  
ping['bill_length_mm'] =  
ping['bill_length_mm'].fillna(0)  
  
ping['bill_length_mm'].count()  
#344
```

# replace()

**Substitui valores existentes por novos valores.** Importante, esse método não sobrescreve o DataFrame.

**Argumento 1:** Valor(es) a serem **substituídos**

**Argumento 2:** Valor(es) que irão **substituir**

```
# Um valor
data['combustivel'] =
data['combustivel'].replace('Petrol', 'Gasolina')

# Lista de valores
data['combustivel'] =
data['combustivel'].replace(['Gasolina', 'Diesel',
'CNG'], [1, 2, 3])
```

# apply()

**Utiliza uma função para modificar valores.** Importante, esse método não sobrescreve o DataFrame.

```
# Função
def primeira_palavra (str):
    return str.split()[0]

data['modelo'] = data['modelo'].apply(primeira_palavra)
```

|   | modelo  | ano  | preço  | combustivel |
|---|---------|------|--------|-------------|
| 0 | Maruti  | 2007 | 60000  | Petrol      |
| 1 | Maruti  | 2007 | 135000 | Petrol      |
| 2 | Hyundai | 2012 | 600000 | Diesel      |
| 3 | Datsun  | 2017 | 250000 | Petrol      |
| 4 | Honda   | 2014 | 450000 | Diesel      |
| 5 | Maruti  | 2007 | 140000 | Petrol      |

# rename()

Utilizada para **renomear** colunas

```
iris.columns

#Index(['sepal_length', 'sepal_width',
#       'petal_length', 'petal_width',
#       'species'],
#      dtype='object')

#renomeando colunas
iris = iris.rename(columns =
{"sepal_length": "comprimento_sepala",
"sepal_width": "largura_sepala"})

iris.columns

#Index(['comprimento_sepala',
#       'largura_sepala', 'petal_length',
#       'petal_width',
#       'species'],
#      dtype='object')
```

# value\_counts()

Conta a ocorrência dos valores presentes em uma **série**.

```
iris['species'].value_counts()  
# setosa      50  
# versicolor  50  
# virginica   50  
# Name: species, dtype: int64  
  
carros['combustivel'].value_counts()  
# Gasolina    10  
# Diesel      2  
# CNG         1  
# Name: combustivel, dtype: int64
```



# value\_counts()

○ **atributo normalize** retorna a informação sobre a **frequência relativa** de cada valor

```
carros['combustivel'].value_counts(normalize=True)
# Gasolina      0.769231
# Diesel        0.153846
# CNG           0.076923
# Name: combustivel, dtype: float64
```

# unique()

Retorna uma lista com os **valores únicos** da coluna selecionada



```
iris['species'].unique()  
  
#array(['setosa', 'versicolor',  
       'virginica'], dtype=object)
```

# nunique()

Retorna a **contagem de elementos únicos** da coluna selecionada

```
iris['species'].nunique()
```

```
#3
```

---

# Exercícios

---

cubos  
//academy//



**Helena Guimarães**  
Professora Cubos Academy

cubos  
// academy //

[www.cubos.academy](http://www.cubos.academy)