Análise de dados com Pandas 🐼

Prof. Me. Alexandre Henrick

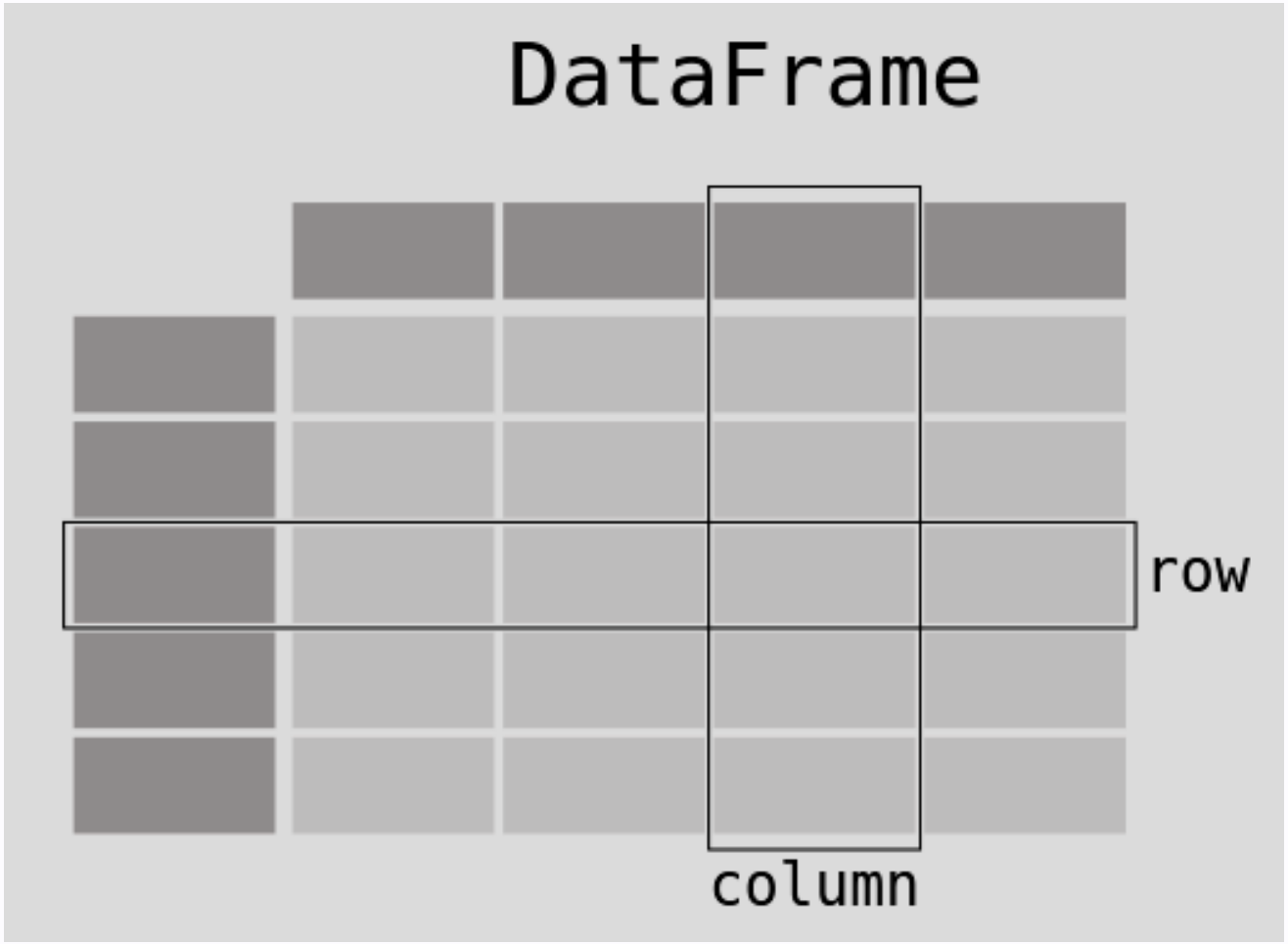
Sistemas de Informação - 8º P

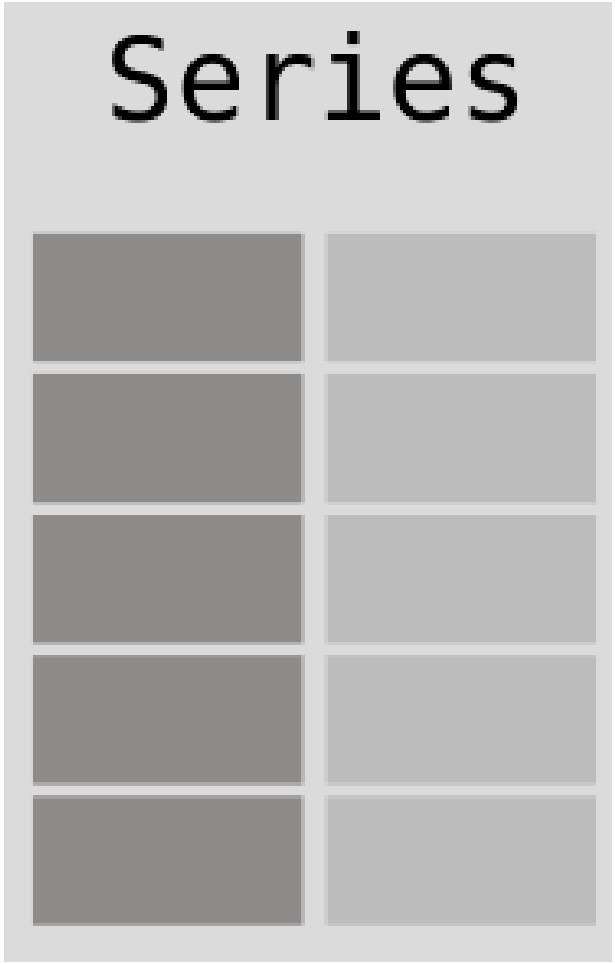
Introdução

- Biblioteca para análise de dados mais popular
- Vastamente utilizada para manipulação, tratamento e transformação de dados estruturados
- Compatível com os formatos mais utilizados (**SQL, CSV, XLS, JSON, PARQUET etc**)
- Documentação

DataFrame

- O Pandas utiliza um **objeto** chamado **DataFrame** para representar dados tabulares
- Esse objeto implementa uma série de métodos para manipularmos esses dados tabulares
- O DataFrame é composto por outro tipo de objeto chamado **Series**
- Cada **coluna** do DataFrame é uma **Series**


DataFrame
row
column


Series

DataFrame

- Podemos construir um DataFrame a partir de um dicionários python

```
import pandas as pd

df = pd.DataFrame(
    {
        "Name": [
            "Fulano da Silva",
            "Ciclano José"
        ],
        "Idade": [35, 44],
        "Profissão": ["Programador", "Analista de Dados"]
    }
)
```

DataFrame

- Podemos selecionar apenas uma **Series** e trabalhar com apenas uma coluna do DF

```
df['Name']
```

- Podemos ainda criar uma Series que não esteja em um DF

```
cidade_natal = pd.Series(["Patos de Minas", "São Paulo"], name="cidade_natal")
```

DataFrame

- A partir de uma Series podemos chamar métodos para extrair informações. Métodos disponíveis

```
# Retorna o maior valor da coluna
df["Idade"].max()
# Número de valores não nulos
df["Idade"].count()
# Estatísticas gerais
df["Idade"].describe()
```

DataFrame

- Podemos ler arquivos e carregá-los dentro de um objeto DataFrame

```
# Método para ler CSV
iris = pd.read_csv("data/iris.csv")
# Método para apresentar os 5 primeiros registros
iris.head()
# Atributos com os tipos de dados de cada coluna
iris.dtypes
# Exporta o DataFrame para uma planilha xlsx
iris.to_excel("iris.xlsx", sheet_name="iris")
```

Dataframe

- Selecionando subconjunto no pandas

```
# Selecionando subconjunto de colunas
sample = iris[['sepal_length', 'selpal_width']]
# Filtrando DataFrame usando uma condição
sample_filter = iris[iris['sepal_length']>5]
#Obetendo apenas a coluna mas com o mesmo filtro
col = iris['sepal_length']>5
# Obtendo média dos valores de uma coluna
iris['sepal_length'].mean()
# iloc é um método onde buscamos uma linha pelo índice
sample_iloc = iris.iloc[3]
# Já o loc nos permite passar linha e coluna, fazendo filtros mais complexos
sample_loc = iris.loc[(iris['sepal_length']>5) & (iris['sepal_width']>3)]
```

DataFrame

- Usando a função Group By

```
carros = pd.read_csv('dados/empresa.csv')

# Obtendo estatística de determinada coluna
print(carros['vendas'].mean())
print(carros['vendas'].std())
# Assim como em listas temos o max e min
print(carros['vendas'].max())
print(carros['vendas'].min())
print(carros['vendas'].describe())

# Usando value_counts()
print(carros['marca'].value_counts(normalize=True))

# Agora usando um agrupamento
# A função groupby do pandas no permite retirar estatísticas de grupos das nossas amostras
print(carros.groupby('marca')) # Apenas o groupby não nos retorna resultado, apenas a indicação do objeto na memória
print(carros.groupby('marca').sum())
print(carros.groupby('data').sum())
print(carros.groupby('ano').sum()) # Soma agrupado por ano
print(carros.groupby(['ano', 'marca']).sum()) # Total agrupado por ano e marca
print(carros.groupby(['ano', 'marca']).mean()) # Total agrupado por ano e marca
print(carros.groupby(['ano']).agg({'vendas': 'mean', 'marca': 'count'})) # Usando a função agg
print(carros.groupby('ano')['vendas'].mean())
```