

Biblioteca *pandas*

CIÊNCIA DA
COMPUTAÇÃO

ATITUS
EDUCAÇÃO



ATITUS
EDUCAÇÃO



- Aula 22 -
Pensamento Computacional

Prof. Me. Lucas R. C. Pessutto



Pandas

- ✧ Pandas é uma biblioteca de código aberto que fornece estruturas de dados fáceis de usar para a linguagem de programação Python
- ✧ Foi criada em 2008, por Wes McKinney
- ✧ Seu nome é derivado da expressão Panel Data
- ✧ Pandas fornece estrutura de dados de alto desempenho e ferramentas de análise de dados
- ✧ Biblioteca GIGANTE! A documentação do pandas possui mais de 2000 páginas

Pandas - Instalação

- * Instalação da biblioteca via PyPI:

```
>> pip install pandas
```

- * Importando a biblioteca no nosso programa

```
import pandas as pd
```

- * Para evitar a repetição da palavra pandas toda vez em que a biblioteca é referenciada no código, é comum a utilização do alias pd que é uma palavra mais curta e conseqüentemente reduz o tamanho das linhas de código.

- * Outras formas de instalação: https://pandas.pydata.org/getting_started.html

Dataframe

- * Uma das estruturas de dados mais utilizada no pandas é o **DataFrame**.
- * Uma instância do tipo **DataFrame** é um objeto de duas (ou mais) dimensões com as seguintes características:
 - Suas dimensões podem ser modificadas decorrente da modificação dos dados.
 - Seus dados podem ser acessados através de rótulos ao invés de exclusivamente por índices.
 - É possível trabalhar com dados heterogêneos, tanto nas linhas como também nas colunas.

Dataframe

- * A classe DataFrame da biblioteca pandas possui um método construtor com alguns parâmetros:
 - **data**: recebe os dados no formato de lista, dicionário ou até mesmo um DataFrame já existe.
 - **index**: recebe uma string ou uma lista de strings que definem os rótulos das linhas.
 - **columns**: recebe uma string ou uma lista de strings que definem os rótulos das colunas.
 - **dtype**: recebe um tipo de dados com intuito de forçar a conversão do tipo de dados do DataFrame. Por padrão esse parâmetro recebe valor None e os tipos dos dados são inferidos.

Dataframe

- * Criando um dataframe a partir de uma lista:

```
import pandas as pd
```

```
nomes = ["Ana", "Bruno", "Carla"]
```

```
idades = [21, 20, 22]
```

```
dados = list(zip(nomes, idades))
```

```
print(dados)
```

```
df = pd.DataFrame(data = dados)
```

```
print(df)
```

- * Note que o DataFrame cria automaticamente rótulos padrões (índices) para que os dados sejam acessados

```
[('Ana', 21), ('Bruno', 20), ('Carla', 22)]  
      0      1  
0     Ana    21  
1    Bruno    20  
2    Carla    22
```

Dataframe

- * DataFrames permitem a criação de rótulos personalizados para as linhas e para as colunas de forma a facilitar o acesso aos dados.

```
import pandas as pd
```

```
dados = [("Ana", 21), ("Bruno", 20), ("Carla", 22)]
```

```
colunas = ["Nome", "Idade"]
```

```
linhas = ["A", "B", "C"]
```

```
df = pd.DataFrame(data=dados, columns=colunas, index=linhas)
```

```
print(df)
```

| | Nome | Idade |
|---|-------|-------|
| A | Ana | 21 |
| B | Bruno | 20 |
| C | Carla | 22 |

Dataframe

- * Os rótulos de um DataFrame podem ser modificados após sua criação, modificando os atributos `columns` e `index`.

```
import pandas as pd
```

```
dados = [("Ana", 21), ("Bruno", 20), ("Carla", 22)]
```

```
df = pd.DataFrame(data=dados)
```

```
print(df)
```

```
df.columns = ["Nome", "Idade"]
```

```
df.index = ["A", "B", "C"]
```

```
print(df)
```

| | 0 | 1 |
|---|-------|-------|
| 0 | Ana | 21 |
| 1 | Bruno | 20 |
| 2 | Carla | 22 |
| | Nome | Idade |
| A | Ana | 21 |
| B | Bruno | 20 |
| C | Carla | 22 |

Dataframe

- * Objetos da classe DataFrame possuem atributos que são bastante úteis:
 - **index**: lista com os rótulos das linhas
 - **columns**: rótulos das colunas no formato de lista
 - **ndim**: número de dimensões do DataFrame
 - **shape**: tupla com o tamanho de cada dimensão do DataFrame
 - **size**: número de elementos (células) do DataFrame
 - **empty**: indica se o DataFrame está vazio (True) ou não (False)

```
import pandas as pd
```

```
dados = {"Nome": ["Ana", "Bruno", "Carla"],  
         "Idade": [21, 20, 22]}
```

```
df = pd.DataFrame(data = dados)  
print(df)
```

| | Nome | Idade |
|---|-------|-------|
| 0 | Ana | 21 |
| 1 | Bruno | 20 |
| 2 | Carla | 22 |

```
print(list(df.index))
```

```
[0, 1, 2]
```

```
print(list(df.columns))
```

```
['Nome', 'Idade']
```

```
print(df.ndim)
```

```
2
```

```
print(df.shape)
```

```
(3, 2)
```

```
print(df.size)
```

```
6
```

```
print(df.empty)
```

```
False
```

Acesso aos Dados

- * Diferentemente das matrizes, a forma de acessar um dado de um DataFrame por meio de índices é a seguinte:

```
dataframe[<coluna>][<linha>]
```

```
import pandas as pd
```

```
dados = [("Ana", 21), ("Bruno", 20), ("Carla", 22)]
```

```
df = pd.DataFrame(data=dados)
```

```
print(df)
```

```
print(df[0][0], df[0][1], df[0][2])
```

| | 0 | 1 |
|---|-------|----|
| 0 | Ana | 21 |
| 1 | Bruno | 20 |
| 2 | Carla | 22 |

Ana Bruno Carla

Indexadores

- * Os DataFrames possuem indexadores para realizar a seleção dos dados
- * Esses indexadores fornecem uma forma fácil e rápida de selecionar um conjunto de dados
- * Os principais indexadores são:
 - **T**: faz a transposição de linhas e colunas
 - **at**: acessa um único elemento usando rótulo
 - **iat**: acessa um único elemento usando índices
 - **loc**: seleção de elementos usando rótulos
 - **iloc**: seleção de elementos usando índices

Indexadores

- * O indexador **T** retorna um DataFrame onde as linhas do Dataframe original são transformadas em colunas.

```
import pandas as pd
```

```
dados = [("Ana", 21), ("Bruno", 20), ("Carla", 22)]  
colunas = ["Nome", "Idade"]  
linhas = ["A", "B", "C"]
```

```
df = pd.DataFrame(data=dados, columns=colunas, index=linhas)  
print(df)
```

```
print("Indexador T")  
print(df.T)
```

| | Nome | Idade |
|---|-------|-------|
| A | Ana | 21 |
| B | Bruno | 20 |
| C | Carla | 22 |

Indexador T:

| | A | B | C |
|-------|-----|-------|-------|
| Nome | Ana | Bruno | Carla |
| Idade | 21 | 20 | 22 |

Indexadores

- * O indexador `at` acessa um único elemento do DataFrame utilizando o rótulo da linha e da coluna

```
import pandas as pd
```

```
dados = [("Ana", 21), ("Bruno", 20), ("Carla", 22)]  
colunas = ["Nome", "Idade"]  
linhas = ["A", "B", "C"]
```

```
df = pd.DataFrame(data=dados, columns=colunas, index=linhas)  
print(df)
```

```
print("Indexador at:")  
print(df.at["C", "Nome"])  
print(df.at["C", "Idade"])
```

| | Nome | Idade |
|---|-------|-------|
| A | Ana | 21 |
| B | Bruno | 20 |
| C | Carla | 22 |

```
Indexador at:  
Carla  
22
```

Indexadores

- * O indexador `iat` acessa um único elemento do DataFrame utilizando os índices da linha e da coluna.

```
import pandas as pd
```

```
dados = [("Ana", 21), ("Bruno", 20), ("Carla", 22)]  
colunas = ["Nome", "Idade"]  
linhas = ["A", "B", "C"]
```

```
df = pd.DataFrame(data=dados, columns=colunas, index=linhas)  
print(df)
```

```
print("\nIndexador iat:")  
print(df.iat[0, 0])  
print(df.iat[0, 1])
```

```
df.iat[<linha>, <coluna>]
```

| | Nome | Idade |
|---|-------|-------|
| A | Ana | 21 |
| B | Bruno | 20 |
| C | Carla | 22 |

```
Indexador iat:  
Ana  
21
```

Indexadores

- * O indexador `loc` seleciona um conjunto de linhas e colunas através dos rótulos ou por uma lista de valores booleanos

```
import pandas as pd
```

```
dados = [("Ana", 21), ("Bruno", 20), ("Carla", 22)]  
colunas = ["Nome", "Idade"]  
linhas = ["A", "B", "C"]
```

```
df = pd.DataFrame(data=dados, columns=colunas, index=linhas)  
print(df)
```

```
print("\nIndexador loc:")  
print(df.loc[['A', 'C']])
```

| | Nome | Idade |
|---|-------|-------|
| A | Ana | 21 |
| B | Bruno | 20 |
| C | Carla | 22 |

| Indexador | loc: | |
|-----------|-------|-------|
| | Nome | Idade |
| A | Ana | 21 |
| C | Carla | 22 |

Indexadores

* Mais exemplos do indexador `loc`:

```
import pandas as pd
```

```
dados = [("Ana", 21), ("Bruno", 20), ("Carla", 22)]
```

```
colunas = ["Nome", "Idade"]
```

```
linhas = ["A", "B", "C"]
```

```
df = pd.DataFrame(data=dados, columns=colunas, index=linhas)
print(df)
```

```
print(df.loc[[True, False, True]])
```

```
print(df.loc[[True, False, True], 'Nome'])
```

| | Nome | Idade |
|---|-------|-------|
| A | Ana | 21 |
| B | Bruno | 20 |
| C | Carla | 22 |

| | Nome | Idade |
|---|-------|-------|
| A | Ana | 21 |
| C | Carla | 22 |

| | |
|---|-------|
| A | Ana |
| C | Carla |

Indexadores

- * O indexador `iloc` seleciona um conjunto de linhas e colunas baseado unicamente em índices

```
import pandas as pd
```

```
dados = [("Ana", 21), ("Bruno", 20), ("Carla", 22)]  
colunas = ["Nome", "Idade"]  
linhas = ["A", "B", "C"]
```

```
df = pd.DataFrame(data=dados, columns=colunas, index=linhas)  
print(df)
```

```
print("\nIndexador iloc:")  
print(df.iloc[[1, 2]])
```

| | Nome | Idade |
|---|-------|-------|
| A | Ana | 21 |
| B | Bruno | 20 |
| C | Carla | 22 |

| Indexador | iloc: |
|-----------|---------------|
| | Nome Idade |
| B | Bruno 20 |
| C | Carla 22 |

Indexadores

* Mais exemplos do indexador `iloc`:

```
import pandas as pd
```

```
dados = [("Ana", 21), ("Bruno", 20), ("Carla", 22)]
```

```
colunas = ["Nome", "Idade"]
```

```
linhas = ["A", "B", "C"]
```

```
df = pd.DataFrame(data=dados, columns=colunas, index=linhas)
print(df)
```

```
print(df.iloc[[-1]])    # Última linha
```

```
print(df.iloc[[0,2], 0]) # Linhas 0 e 2 e a coluna 0
```

| | Nome | Idade |
|---|-------|-------|
| A | Ana | 21 |
| B | Bruno | 20 |
| C | Carla | 22 |

| | Nome | Idade |
|---|-------|-------|
| C | Carla | 22 |

| | |
|---|-------|
| A | Ana |
| C | Carla |

Inserindo/Modificandos Colunas

* Para adicionar uma nova coluna ao DataFrame basta atribuir ao rótulo da coluna desejada um valor padrão ou uma lista com os valores desejados:

* Valor Padrão:

```
df[<novo_rótulo>] = <valor_padrão>
```

* Lista de Valores:

```
df[<novo_rótulo>] = [<valor_1>, <valor_2>, ..., <valor_n>,]
```

* O mesmo processo pode ser aplicado para modificar uma coluna já existente

```
import pandas as pd
```

```
dados = [("Ana", 21), ("Bruno", 20), ("Carla", 22)]  
colunas = ["Nome", "Idade"]  
linhas = ["A", "B", "C"]
```

```
df = pd.DataFrame(data=dados, columns=colunas,  
print(df)
```

```
# Usando um valor padrão
```

```
df["Ano Nascimento"] = "2000"  
print(df)
```

```
# Usando uma lista de valores
```

```
df["Sexo"] = ["F", "M", "F"]  
print(df)
```

```
# Modificando valores existentes
```

```
df["Ano Nascimento"] = ["1999", "2003", "1992"]  
print(df)
```

| | Nome | Idade |
|---|-------|-------|
| A | Ana | 21 |
| B | Bruno | 20 |
| C | Carla | 22 |

| | Nome | Idade | Ano Nascimento |
|---|-------|-------|----------------|
| A | Ana | 21 | 2000 |
| B | Bruno | 20 | 2000 |
| C | Carla | 22 | 2000 |

| | Nome | Idade | Ano Nascimento | Sexo |
|---|-------|-------|----------------|------|
| A | Ana | 21 | 2000 | F |
| B | Bruno | 20 | 2000 | M |
| C | Carla | 22 | 2000 | F |

| | Nome | Idade | Ano Nascimento | Sexo |
|---|-------|-------|----------------|------|
| A | Ana | 21 | 1999 | F |
| B | Bruno | 20 | 2003 | M |
| C | Carla | 22 | 1992 | F |

Inserindo/Modificandos Linhas

- * Para adicionar uma ou mais linhas ao DataFrame é possível utilizar o método `_append`
- * Esse método cria um novo DataFrame adicionando ao final os novos valores
- * Para isso, o método recebe como parâmetro um outro DataFrame ou uma lista com os novos valores
- * Caso os rótulos das linhas não sejam compatíveis, o parâmetro `ignore_index` deve ser atribuído como `True` para que os rótulos personalizados das linhas sejam ignorados

```
import pandas as pd
```

```
dados = [("Ana", 21), ("Bruno", 20), ("Carla", 22)]
colunas = ["Nome", "Idade"]
linhas = ["A", "B", "C"]
```

```
df = pd.DataFrame(data=dados, columns=colunas, index=linhas)
print(df)
```

```
dados = [{"Nome": "Daniel", "Idade": 18},
          {"Nome": "Mario", "Idade": 51}]
```

```
df1 = df._append(dados, ignore_index=True)
print(df1)
```

Método Depreciado na versão 2.0 do pandas!

| | Nome | Idade |
|---|-------|-------|
| A | Ana | 21 |
| B | Bruno | 20 |
| C | Carla | 22 |

| | Nome | Idade |
|---|--------|-------|
| 0 | Ana | 21 |
| 1 | Bruno | 20 |
| 2 | Carla | 22 |
| 3 | Daniel | 18 |
| 4 | Mario | 51 |

```
import pandas as pd
```

```
dados = [("Ana", 21), ("Bruno", 20), ("Carla", 22)]  
colunas = ["Nome", "Idade"]  
linhas = ["A", "B", "C"]
```

```
df = pd.DataFrame(data=dados, columns=colunas, index=linhas)  
print(df)
```

Usando o método concat para concatenar os dados ao dataframe

```
novos_dados = {"Nome": ["Daniel", "Mario"],  
               "Idade": [18, 51]}  
df2 = pd.DataFrame(data=novos_dados)  
df3 = pd.concat([df, df2], ignore_index=True)  
print(df3)
```

| | Nome | Idade |
|---|-------|-------|
| A | Ana | 21 |
| B | Bruno | 20 |
| C | Carla | 22 |

| | Nome | Idade |
|---|--------|-------|
| 0 | Ana | 21 |
| 1 | Bruno | 20 |
| 2 | Carla | 22 |
| 3 | Daniel | 18 |
| 4 | Mario | 51 |

Inserindo/Modificandos Linhas

- * Os indexadores loc e iloc também podem ser usados para modificar uma linha já existente

- * Para isso, basta atribuir os novos valores desejados ou um valor padrão

- * O indexador iloc também pode ser usado para adicionar uma linha no final do DataFrame de forma similar

- * Valor Padrão:
`df.loc[<rótulo>] = <valor_padrão>`
`df.iloc[<linha>] = <valor_padrão>`

- * Valores específicos para cada coluna:

```
df.loc[<rótulo>] = [<valor_1>, <valor_2>, ..., <valor_n>],  
df.loc[<linha>] = [<valor_1>, <valor_2>, ..., <valor_n>],
```

Inserindo/Modificandos Linhas

```
import pandas as pd
```

```
dados = [("Ana", 21), ("Bruno", 20), ("Carla", 22)]  
colunas = ["Nome", "Idade"]  
linhas = ["A", "B", "C"]
```

```
df = pd.DataFrame(data=dados, columns=colunas, index=linhas)  
print(df)
```

```
df.loc['B'] = ["Bento", 22]  
df.loc['C'] = ["Camila", 31]  
df.loc['D'] = ["Daniela", 18]
```

```
print(df)
```

| | Nome | Idade |
|---|-------|-------|
| A | Ana | 21 |
| B | Bruno | 20 |
| C | Carla | 22 |

| | Nome | Idade |
|---|---------|-------|
| A | Ana | 21 |
| B | Bento | 22 |
| C | Camila | 31 |
| D | Daniela | 18 |

Inserindo/Modificandos Linhas

Modificando dados com o `iloc`:

```
import pandas as pd

dados = [("Ana", 21), ("Bruno", 20), ("Carla", 22)]
colunas = ["Nome", "Idade"]
linhas = ["A", "B", "C"]

df = pd.DataFrame(data=dados, columns=colunas, index=linhas)
print(df)

df.iloc[1] = ["Bruno", 19]
df.iloc[3] = ["Marcela", 21]

print(df)
```

| | Nome | Idade |
|---|-------|-------|
| A | Ana | 21 |
| B | Bruno | 20 |
| C | Carla | 22 |

| | Nome | Idade |
|---|---------|-------|
| A | Ana | 21 |
| B | Bruno | 19 |
| C | Camila | 31 |
| D | Marcela | 21 |

Modificando Células

- * Os indexadores `at` e `iat` também podem ser usados para modificar uma célula do DataFrame
- * Para fazer isso, basta atribuir um novo valor para a célula desejada:

```
df.at[<rótulo>, <rótulo>] = <novo_valor>  
df.iat[<linha>, <coluna>] = <novo_valor>
```

Modificando Células

```
import pandas as pd
```

```
dados = [("Ana", 21), ("Bruno", 20), ("Carla", 22)]
```

```
colunas = ["Nome", "Idade"]
```

```
linhas = ["A", "B", "C"]
```

```
df = pd.DataFrame(data=dados, columns=colunas, index=linhas)
```

```
print(df)
```

```
df.at['C', "Idade"] = 20
```

```
df.iat[0, 1] = 17
```

```
print(df)
```

| | Nome | Idade |
|---|-------|-------|
| A | Ana | 21 |
| B | Bruno | 20 |
| C | Carla | 22 |

| | Nome | Idade |
|---|-------|-------|
| A | Ana | 17 |
| B | Bruno | 20 |
| C | Carla | 20 |

Removendo Linhas e Colunas

- * O método **drop** é utilizado para remover linhas e colunas de um DataFrame.
- * Alguns parâmetros do método **drop** são:
 - **index**: recebe um rótulo ou uma lista de rótulos das linhas que devem ser removidas.
 - **columns**: recebe um rótulo ou uma lista de rótulos das colunas que devem ser removidas.
 - **inplace**: determina se as mudanças devem ser aplicadas diretamente no DataFrame ou em uma cópia (valor padrão é False)

Operações Lógicas e Matemáticas

- * A biblioteca permite utilizar operadores lógicos e aritméticos em colunas inteiras de um DataFrame.

```
import pandas as pd
```

```
dados = [("Ana", 21), ("Bruno", 20), ("Carla", 22)]
```

```
colunas = ["Nome", "Idade"]
```

```
linhas = ["A", "B", "C"]
```

```
df = pd.DataFrame(data=dados, columns=colunas, index=linhas)
```

```
print(df)
```

```
df['Idade'] += 1
```

```
print(df)
```

| | Nome | Idade |
|---|-------|-------|
| A | Ana | 21 |
| B | Bruno | 20 |
| C | Carla | 22 |

| | Nome | Idade |
|---|-------|-------|
| A | Ana | 22 |
| B | Bruno | 21 |
| C | Carla | 23 |

Operações Lógicas e Matemáticas

- * O resultado da aplicação de um operador lógico é uma lista de booleanos com o resultado da operação para cada linha do DataFrame.

```
import pandas as pd
```

```
dados = [("Ana", 21), ("Bruno", 20), ("Carla", 22)]  
colunas = ["Nome", "Idade"]  
linhas = ["A", "B", "C"]
```

```
df = pd.DataFrame(data=dados, columns=colunas, index=linhas)  
print(df)
```

```
maior_idade = list(df["Idade"] > 21)  
print(maior_idade)
```

| | Nome | Idade |
|---|-------|-------|
| A | Ana | 21 |
| B | Bruno | 20 |
| C | Carla | 22 |


```
[False, False, True]
```


Filtrando Dados

- * Podemos combinar o resultado de uma operação lógica com o comando loc para selecionar dados que atendem a uma condição

```
import pandas as pd
```

```
dados = [("Ana", 21, "F"), ("Bruno", 20, "M"), ("Carla", 22, "F")]  
colunas = ["Nome", "Idade", "Sexo"]  
linhas = ["A", "B", "C"]
```

```
df = pd.DataFrame(data=dados, columns=colunas, index=linhas)  
print(df)
```

```
mulheres = list(df['Sexo'] == 'F')  
print(df.loc[mulheres])
```

| | Nome | Idade | Sexo |
|---|-------|-------|------|
| A | Ana | 21 | F |
| B | Bruno | 20 | M |
| C | Carla | 22 | F |

| | Nome | Idade | Sexo |
|---|-------|-------|------|
| A | Ana | 21 | F |
| C | Carla | 22 | F |

Ordenando um DataFrame

- * Podemos ordenar um DataFrame utilizando o método `sort_values`
- * Alguns dos parâmetros do método `sort_values`:
 - `by`: string ou lista de strings especificando os rótulos que serão utilizados como chave para a ordenação
 - `axis`: eixo de ordenação vertical (0, padrão) ou horizontal (1)
 - `ascending`: ordenação crescente ou decrescente (valor padrão é True)
 - `kind`: algoritmo de ordenação que será utilizado (valor padrão é quicksort)
 - `inplace`: determina se as mudanças devem ser aplicadas diretamente no DataFrame ou em uma cópia (valor padrão é False)

Ordenando um DataFrame

```
import pandas as pd

dados = [("Ana", 21, "F"), ("Bruno", 20, "M"), ("Carla", 22, "F"),
         ("Daniel", 18, "M"), ("Mario", 51, "M")]

colunas = ["Nome", "Idade", "Sexo"]

df = pd.DataFrame(data=dados, columns=colunas)
print(df)

df.sort_values(by='Idade', ascending=False, inplace=True)
print(df)
```

| | Nome | Idade | Sexo |
|---|--------|-------|------|
| 0 | Ana | 21 | F |
| 1 | Bruno | 20 | M |
| 2 | Carla | 22 | F |
| 3 | Daniel | 18 | M |
| 4 | Mario | 51 | M |

| | Nome | Idade | Sexo |
|---|--------|-------|------|
| 4 | Mario | 51 | M |
| 2 | Carla | 22 | F |
| 0 | Ana | 21 | F |
| 1 | Bruno | 20 | M |
| 3 | Daniel | 18 | M |

Ordenando um DataFrame

Ordenação com duas chaves:

```
import pandas as pd
```

```
dados = [("Ana", 21, "F"), ("Bruno", 20, "M"), ("Carla", 22, "F"),  
         ("Daniel", 18, "M"), ("Mario", 51, "M")]
```

```
colunas = ["Nome", "Idade", "Sexo"]
```

```
df = pd.DataFrame(data=dados, columns=colunas)  
print(df)
```

```
df.sort_values(by='Idade', ascending=False, inplace=True)  
print(df)
```

```
df.sort_values(by=["Sexo", "Idade"], inplace=True)  
print(df)
```

| | Nome | Idade | Sexo |
|---|--------|-------|------|
| 0 | Ana | 21 | F |
| 1 | Bruno | 20 | M |
| 2 | Carla | 22 | F |
| 3 | Daniel | 18 | M |
| 4 | Mario | 51 | M |

| | Nome | Idade | Sexo |
|---|--------|-------|------|
| 0 | Ana | 21 | F |
| 2 | Carla | 22 | F |
| 3 | Daniel | 18 | M |
| 1 | Bruno | 20 | M |
| 4 | Mario | 51 | M |

Ordenando um DataFrame

- * É possível também ordenar um DataFrame pelos seus rótulos utilizando o método `sort_index`
- * Alguns dos parâmetros do método `sort_index`:
 - `axis`: eixo de ordenação vertical (0, padrão) ou horizontal (1)
 - `ascending`: ordenação crescente ou decrescente (valor padrão é True)
 - `kind`: algoritmo de ordenação que será utilizado (valor padrão é quicksort)
 - `inplace`: determina se as mudanças devem ser aplicadas diretamente no DataFrame ou em uma cópia (valor padrão é False)

Ordenando um DataFrame

```
import pandas as pd
```

```
dados = [("Ana", 21, "F"), ("Bruno", 20, "M"), ("Carla", 22, "F"),  
         ("Daniel", 18, "M"), ("Mario", 51, "M")]
```

```
colunas = ["Nome", "Idade", "Sexo"]
```

```
df = pd.DataFrame(data=dados, columns=colunas)
```

```
print(df)
```

```
# Ordenação pelos rótulos
```

```
df.sort_index(axis=1, inplace=True)
```

```
print(df)
```

| | Nome | Idade | Sexo |
|---|--------|-------|------|
| 0 | Ana | 21 | F |
| 1 | Bruno | 20 | M |
| 2 | Carla | 22 | F |
| 3 | Daniel | 18 | M |
| 4 | Mario | 51 | M |

| | Idade | Nome | Sexo |
|---|-------|--------|------|
| 0 | 21 | Ana | F |
| 2 | 22 | Carla | F |
| 3 | 18 | Daniel | M |
| 1 | 20 | Bruno | M |
| 4 | 51 | Mario | M |

Métodos Aritméticos

- * A biblioteca pandas possui vários métodos para realização de cálculos em colunas:
 - **abs**: retorna uma lista com os valores absolutos da coluna
 - **count**: conta quantas células da coluna possuem valores disponíveis
 - **nunique**: conta o número de valores distintos da coluna
 - **sum**: retorna a soma dos valores de uma coluna
 - **max**: retorna o valor máximo de uma das colunas
 - **min**: retorna o menor valor de uma coluna
 - **mean**: retorna a média aritmética dos valores da coluna
 - **median**: retorna a mediana dos valores da coluna
 - **mode**: retorna a moda dos valores de uma coluna

```
import pandas as pd
```

```
dados = [("Ana", 20, "F"), ("Bruno", 20, "M"), ("Carla", 22, "F"), ("Daniel", 18, "M"),  
("Mario", 51, "M")]
```

```
colunas = ["Nome", "Idade", "Sexo"]
```

```
df = pd.DataFrame(data=dados, columns=colunas)
```

```
print(df)
```

```
print(df.Idade.count())
```

```
print(df.Idade.sum())
```

```
print(df.Idade.max())
```

```
print(df.Idade.min())
```

```
print(df.Idade.mean())
```

```
print(df.Idade.median())
```

```
print(df.Idade.mode())
```

| | Nome | Idade | Sexo |
|------|--------|-------|------|
| 0 | Ana | 20 | F |
| 1 | Bruno | 20 | M |
| 2 | Carla | 22 | F |
| 3 | Daniel | 18 | M |
| 4 | Mario | 51 | M |
| 5 | | | |
| 131 | | | |
| 51 | | | |
| 18 | | | |
| 26.2 | | | |
| 20.0 | | | |
| 0 | 20 | | |

Exportando Dados

- * A biblioteca pandas fornece uma forma rápida e fácil para exportar os dados de um DataFrame para diferentes formatos:
- * Por exemplo, podemos exportar para um arquivo csv usando a função `to_csv`:
- * Essa função possui os seguintes parâmetros:
 - `path_or_buff`: nome do arquivo ou buffer onde o arquivo deve ser salvo
 - `sep`: caractere de separação do arquivo (padrão vírgula)
 - `header`: define se os rótulos das colunas devem ser inseridos no arquivo ou não (padrão True)
 - `index`: define se os rótulos das linhas devem ser inseridos no arquivo (padrão True)

Exportando Dados

```
import pandas as pd
```

```
dados = [("Ana", 20, "F"), ("Bruno", 20, "M"), ("Carla", 22, "F"),  
         ("Daniel", 18, "M"), ("Mario", 51, "M")]
```

```
colunas = ["Nome", "Idade", "Sexo"]
```

```
df = pd.DataFrame(data=dados, columns=colunas)  
print(df)
```

```
df.to_csv('dados.csv')
```

Importando Dados

- * Para importar um arquivo csv o pandas fornece a função `read_csv`:
- * Essa função possui os seguintes parâmetros:
 - `filepath_or_buffer`: nome do arquivo ou buffer do arquivo
 - `sep`: caractere de separação do arquivo (padrão vírgula)
 - `names`: lista de rótulos para serem usados nas colunas
 - `header`: linha do arquivo csv para ser utilizada como rótulo para as colunas
 - `Index_col`: coluna do arquivo csv para ser utilizada como rótulo para as linhas

Importando Dados

```
import pandas as pd

titanic = pd.read_csv('titanic.csv', index_col=0, header=0)
#titanic =
pd.read_csv('https://vincentarelbundock.github.io/Rdatasets/csv/carData/
TitanicSurvival.csv')

print(titanic.head)

print(titanic.tail)

print(titanic.describe())
```

Exercício

1. Crie um DataFrame chamado temperaturas a partir de um dicionário que contém três medições de temperatura para quatro pessoas: “Joe”, “Amanda”, “Max”, “Cindy”
2. Recrie o DataFrame do item anterior, mas nomeando os índices das linhas para “Manhã”, “Tarde” e “Noite”
3. Selecione as temperaturas da “Amanda”
4. Selecione as medidas de temperatura da manhã de todos os pacientes
5. Selecione as temperaturas da manhã e tarde
6. Selecione as temperaturas de “Joe” e “Max”
7. Selecione as temperaturas de “Amanda” e “Cindy” nos períodos da tarde e noite
8. Produza a Transposta dessa tabela
9. Ordene a tabela para que os nomes dos pacientes estejam em ordem alfabética
10. Qual a média de temperatura de Joe?
11. Qual a maior temperatura registrada por Amanda?