



**Instituto de  
Computação**

UNIVERSIDADE ESTADUAL DE CAMPINAS



## MC102 – Aula 17

### Pandas

Algoritmos e Programação de Computadores

---

Zanoni Dias

2023

Instituto de Computação

Pandas

DataFrame

Manipulação de Dados

Importando e Exportando Dados

Documentação

# Pandas

---

- Pandas é uma biblioteca de código aberto que fornece estruturas de dados fáceis de usar para a linguagem de programação Python.
- Além disso, a biblioteca fornece estrutura de dados de alto desempenho e ferramentas de análise de dados.
- Instalação da biblioteca via PyPI:

```
1 pip install pandas
```

- Outras formas de instalação:  
<https://pandas.pydata.org/getpandas.html>

- Para utilizar a biblioteca basta realizar a importação.

```
1 import pandas
```

- 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 consequentemente reduz o tamanho das linhas de código.

```
1 # Forma mais comum de importar a biblioteca com alias  
2 import pandas as pd
```

- Os exemplos de código utilizarão a importação da biblioteca com o alias pd.

# 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.

- 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 existente.
  - `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.



# Criando um DataFrame

- Criando um DataFrame a partir de uma lista de tuplas:

```
1 import pandas as pd
2 nomes = ['Ana', 'Bruno', 'Carla']
3 idades = [21, 20, 22]
4 dados = list(zip(nomes, idades))
5 print(dados)
6 # [('Ana', 21), ('Bruno', 20), ('Carla', 22)]
7 df = pd.DataFrame(data = dados)
8 print(df)
9 #      0    1
10 # 0   Ana  21
11 # 1 Bruno  20
12 # 2 Carla  22
```

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

# Criando um DataFrame

- Criando um DataFrame a partir de um dicionário:

```
1 import pandas as pd
2 dados = {'Nome': ['Ana', 'Bruno', 'Carla'],
3          'Idade': [21, 20, 22]}
4 # {'Nome': ['Ana', 'Bruno', 'Carla'],
5 #  'Idade': [21, 20, 22]}
6 df = pd.DataFrame(data = dados)
7 print(df)
8 #      Nome  Idade
9 # 0    Ana    21
10 # 1 Bruno    20
11 # 2 Carla    22
```

- Note que o DataFrame criado possui as colunas com nomes indicados nas chaves do dicionário.

# Criando um DataFrame com Rótulos Personalizados

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

```
1 import pandas as pd
2 dados = [('Ana', 21), ('Bruno', 20), ('Carla', 22)]
3 colunas = ['Nome', 'Idade']
4 linhas = ['A', 'B', 'C']
5 df = pd.DataFrame(data = dados, columns = colunas,
6                   index = linhas)
7 print(df)
8 #      Nome  Idade
9 # A     Ana    21
10 # B  Bruno    20
11 # C  Carla    22
```

# Modificando os Rótulos de uma DataFrame

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

```
1 import pandas as pd
2 dados = [('Ana', 21), ('Bruno', 20), ('Carla', 22)]
3 df = pd.DataFrame(data = dados)
4 print(df)
5 #      0    1
6 # 0    Ana  21
7 # 1  Bruno  20
8 # 2  Carla  22
9 df.columns = ['Nome', 'Idade']
10 df.index = ['A', 'B', 'C']
11 print(df)
12 #      Nome  Idade
13 # A     Ana    21
14 # B  Bruno    20
15 # C  Carla    22
```

# Atributos de um DataFrame

- Objetos do tipo Dataframe possuem atributos que são bastante úteis:
  - `index`: retorna os rótulos das linhas em formato de lista.
  - `columns`: retorna os rótulos das colunas em formato de lista.
  - `ndim`: retorna o número de dimensões do DataFrame.
  - `shape`: retorna o tamanho de cada uma das dimensões em um formato de tupla.
  - `size`: retorna o número de elementos (células) do DataFrame.
  - `empty`: retorna se o DataFrame está vazio (`True`) ou não (`False`).

# Atributos de um DataFrame

- Exemplos:

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade
5 # A     Ana    21
6 # B  Bruno    20
7 # C  Carla    22
8 print(list(df.index))
9 # ['A', 'B', 'C']
10 print(list(df.columns))
11 # ['Nome', 'Idade']
```

# Atributos de um DataFrame

- Exemplos:

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade
5 # A     Ana    21
6 # B   Bruno    20
7 # C   Carla    22
8 print(df.ndim)
9 # 2
10 print(df.shape)
11 # (3, 2)
12 print(df.size)
13 # 6
14 print(df.empty)
15 # False
```

# Acessando os Dados de um DataFrame

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

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

- Exemplo:

```
1 import pandas as pd
2 dados = [('Ana', 21), ('Bruno', 20), ('Carla', 22)]
3 df = pd.DataFrame(data = dados)
4 print(df)
5 #           0    1
6 # 0      Ana   21
7 # 1  Bruno   20
8 # 2  Carla   22
9 print(df[0][0], df[0][1], df[0][2])
10 # Ana Bruno Carla
```



- Os DataFrames possuem indexadores para seleção de dados.
- Esses indexadores fornecem uma forma fácil e rápida de selecionar um conjunto de dados de um DataFrame.
- Alguns deles são:
  - T: usado para transpor linhas e colunas.
  - at: acessa um único elemento utilizando rótulos.
  - iat: acessa um único elemento utilizando índices.
  - loc: seleção de elementos utilizando rótulos.
  - iloc: seleção de elementos utilizando índices.

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

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade
5 # A     Ana    21
6 # B   Bruno    20
7 # C   Carla    22
8 print(df.T)
9 #           A      B      C
10 # Nome     Ana  Bruno  Carla
11 # Idade     21     20     22
```

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

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade
5 # A     Ana    21
6 # B   Bruno    20
7 # C   Carla    22
8 df.at['C', 'Nome']
9 # 'Carla'
10 df.at['C', 'Idade']
11 # 22
```

# Indexadores

- O indexador `at` opera apenas com os rótulos e não com os índices dos elementos.
- Caso os índices de um elemento sejam fornecidos, ao invés dos seus rótulos, um erro é gerado.

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade
5 # A     Ana    21
6 # B   Bruno    20
7 # C   Carla    22
8 print(df.at['C', 'Nome'])
9 # Carla
10 print(df.at[2, 0])
11 # ValueError: At based indexing on an non-integer index
12 #                can only have non-integer indexers
```

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

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade
5 # A     Ana    21
6 # B  Bruno    20
7 # C   Carla    22
8 print(df.iat[0, 0])
9 # Ana
10 print(df.iat[0, 1])
11 # 21
```

# Indexadores

- O indexador `iat` opera apenas com os índices e não com os rótulos dos elementos.
- Caso os rótulos de um elemento sejam fornecidos, ao invés de seus índices, um erro é gerado.

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade
5 # A     Ana    21
6 # B   Bruno    20
7 # C   Carla    22
8 print(df.iat[1, 0])
9 # Bruno
10 print(df.iat['B', 'Idade'])
11 # ValueError: iAt based indexing can only have integer
12 #           indexers
```

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

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade
5 # A     Ana    21
6 # B   Bruno    20
7 # C   Carla    22
8 print(df.loc[['A', 'C']])
9 #      Nome  Idade
10 # A     Ana    21
11 # C   Carla    22
```

- Mais exemplos com o indexador `loc`.

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade
5 # A     Ana    21
6 # B  Bruno    20
7 # C  Carla    22
8 print(df.loc[[True, False, True]])
9 #      Nome  Idade
10 # A     Ana    21
11 # C  Carla    22
12 print(df.loc[[True, False, True], 'Nome'])
13 # A     Ana
14 # C     Carla
15 # Name: Nome, dtype: object
```



- O indexador `loc` não opera com índices. Um erro é gerado caso índices sejam fornecidos.

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade
5 # A     Ana    21
6 # B  Bruno    20
7 # C  Carla    22
8 print(df.loc[[0,2]])
9 # KeyError: "None of [Int64Index([0, 2], dtype='int64')]
10 #           are in the [index]"
```

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

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade
5 # A   Ana    21
6 # B Bruno    20
7 # C Carla    22
8 print(df.iloc[[1, 2]])
9 #      Nome  Idade
10 # B Bruno    20
11 # C Carla    22
```

- Mais exemplos com o indexador `iloc`.

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade
5 # A     Ana    21
6 # B  Bruno    20
7 # C   Carla    22
8 print(df.iloc[-1])
9 # Nome      Carla
10 # Idade      22
11 # Name: C, dtype: object
12 print(df.iloc[[0,2],0])
13 # A     Ana
14 # C     Carla
15 # Name: Nome, dtype: object
```

- O indexador `iloc` não opera com rótulos. Um erro é gerado caso rótulos sejam fornecidos.

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade
5 # A     Ana    21
6 # B  Bruno    20
7 # C   Carla    22
8 print(df.iloc[['B', 'C']])
9 # ValueError: invalid literal for int() with base 10: 'B'
```

# Manipulação de Dados

---

# Adicionando e Modificando Colunas em um DataFrame

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

```
1 df[<novo rótulo>] = <valor_padrão>
```

- Associando valores específicos para cada uma das linhas:

```
1 df[<novo rótulo>] = [<valor_1>, <valor_2>, ..., <valor_n>]
```

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

# Adicionando e Modificando Colunas em um DataFrame

- Exemplo associando um valor padrão:

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade
5 # A     Ana    21
6 # B  Bruno    20
7 # C  Carla    22
8 df['Sexo'] = 'F'
9 print(df)
10 #      Nome  Idade  Sexo
11 # A     Ana    21     F
12 # B  Bruno    20     F
13 # C  Carla    22     F
```

# Adicionando e Modificando Colunas em um DataFrame

- Exemplo associando valores específicos:

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade Sexo
5 # A     Ana    21    F
6 # B   Bruno    20    F
7 # C   Carla    22    F
8 df['Sexo'] = ['F', 'M', 'F']
9 print(df)
10 # A     Ana    21    F
11 # B   Bruno    20    M
12 # C   Carla    22    F
```



## Adicionando e Modificando Linhas de um DataFrame

- Para adicionar uma ou mais novas linhas ao DataFrame, é possível utilizar o método `append`.
- O método `append` cria um novo DataFrame adicionando no 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.

# Adicionando e Modificando Linhas de um DataFrame

- Exemplo do método `append` ignorando os rótulos das linhas:

```
1 import pandas as pd
2 ...
3 print(df1)
4 #      Nome  Idade Sexo
5 # A     Ana    21    F
6 # B   Bruno    20    M
7 dados = [ {'Nome': 'Carla', 'Idade': 22, 'Sexo': 'F'},
8            {'Nome': 'Daniel', 'Idade': 18, 'Sexo': 'M'} ]
9
10 df2 = df1.append(dados, ignore_index = True)
11 print(df2)
12 #      Nome  Idade Sexo
13 # 0     Ana    21    F
14 # 1   Bruno    20    M
15 # 2   Carla    22    F
16 # 3  Daniel    18    M
```

# Adicionando e Modificando Linhas de um DataFrame

- Exemplo do método `append` mantendo os rótulos das linhas:

```
1 import pandas as pd
2 ...
3 print(df1)
4 #      Nome  Idade Sexo
5 # A     Ana    21    F
6 # B  Bruno    20    M
7 dados = [ {'Nome': 'Carla', 'Idade': 22, 'Sexo': 'F'},
8           {'Nome': 'Daniel', 'Idade': 18, 'Sexo': 'M'} ]
9 df2 = pd.DataFrame(dados, index = ['C', 'D'])
10 df3 = df1.append(df2, ignore_index = False)
11 print(df3)
12 #      Nome  Idade Sexo
13 # A     Ana    21    F
14 # B  Bruno    20    M
15 # C   Carla    22    F
16 # D  Daniel    18    M
```

# Adicionando e Modificando Linhas de um DataFrame

- Os indexadores `loc` e `iloc` também podem ser utilizados para modificar uma linha já existente.
- Para isso, basta atribuir os novos valores desejados ou um valor padrão.
- O indexador `loc` também pode ser utilizado para adicionar uma nova linha no final do `DataFrame` de forma similar.
- Valor padrão para todas as colunas:

```
1 df.loc[<rótulo>] = <valor_padrão>  
2 df.iloc[<linha>] = <valor_padrão>
```

- Valores específicos para cada coluna:

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

# Adicionando e Modificando Linhas de um DataFrame

- Exemplo de utilização do indexador `loc` para inserir e alterar linhas:

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade Sexo
5 # A     Ana    21    F
6 # B   Bruno    20    M
7 df.loc['B'] = ['Bento', 22, 'M']
8 df.loc['C'] = ['Carla', 22, 'F']
9 df.loc['D'] = ['Daniela', 18, 'F']
10 print(df)
11 #      Nome  Idade Sexo
12 # A     Ana    21    F
13 # B   Bento    22    M
14 # C    Carla    22    F
15 # D  Daniela    18    F
```

# Adicionando e Modificando Linhas de um DataFrame

- Exemplo de utilização do indexador `iloc` para alterar linhas:

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade  Sexo
5 # A      Ana    21    F
6 # B      Bento  22    M
7 # C      Carla  22    F
8 # D  Daniela   18    F
9 df.iloc[1] = ['Bruno', 19, 'M']
10 df.iloc[3] = ['Daniel', 18, 'M']
11 print(df)
12 #      Nome  Idade  Sexo
13 # A      Ana    21    F
14 # B      Bruno  19    M
15 # C      Carla  22    F
16 # D      Daniel  18    M
```

# Adicionando e Modificando Linhas de um DataFrame

- De forma semelhante, os indexadores `at` e `iat` também podem ser utilizados para modificar uma célula do `DataFrame`.
- Para isso, basta atribuir um novo valor para a célula desejada.

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

# Adicionando e Modificando Linhas de um DataFrame

- Exemplo de utilização dos indexadores `loc` e `iloc` para alterar células:

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade  Sexo
5 # A     Ana    21    F
6 # B    Bruno    19    M
7 # C    Carla    22    F
8 # D   Daniel    18    M
9 df.at['C', 'Idade'] = 20
10 df.iat[0, 1] = 17
11 print(df)
12 #      Nome  Idade  Sexo
13 # A     Ana    17    F
14 # B    Bruno    19    M
15 # C    Carla    20    F
16 # D   Daniel    18    M
```



# Removendo Linhas e Colunas de um DataFrame

- É possível remover linhas ou colunas de um DataFrame utilizando o método `drop`.
- Alguns dos parâmetros do método `drop` são:
  - `index`: recebe um rótulo ou uma lista de rótulos das linhas que serão removidas.
  - `columns`: recebe um rótulo ou uma lista de rótulos das colunas que serão removidas.
  - `inplace`: determina se as mudanças devem ser aplicadas diretamente no DataFrame ou em uma cópia (valor padrão é `False`).

# Removendo Linhas e Colunas de um DataFrame

- Exemplo de utilização método drop:

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade  Sexo
5 # A     Ana    17    F
6 # B    Bruno   19    M
7 # C    Carla   20    F
8 # D   Daniel   18    M
9 df.drop(index = ['A', 'D'], columns = ['Sexo'],
10         inplace = True)
11 print(df)
12 #      Nome  Idade
13 # B    Bruno   19
14 # C    Carla   20
```

- A biblioteca pandas permite utilizar operadores lógicos e aritméticos em colunas inteiras de um DataFrame.
- Alguns exemplos de operadores:
  - +, +=
  - -, -=
  - \*, \*=
  - /, /=
  - ==, >=, <=, !=, >, <

# Operadores

- Exemplo de como aumentar em 1 ano a idade de todas as pessoas do DataFrame.

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade  Sexo
5 # A      Ana    17    F
6 # B     Bruno    19    M
7 # C     Carla    20    F
8 # D   Daniel    18    M
9 df['Idade'] += 1
10 print(df)
11 #      Nome  Idade  Sexo
12 # A      Ana    18    F
13 # B     Bruno    20    M
14 # C     Carla    21    F
15 # D   Daniel    19    M
```

# Operadores

- Como resultado da aplicação de um operador lógico uma lista de booleanos é obtida representando a resposta para cada linha do DataFrame.
- Exemplo de como verificar as pessoas que já atingiram a maioridade penal.

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade  Sexo
5 # A     Ana    17    F
6 # B   Bruno    19    M
7 # C   Carla    20    F
8 # D  Daniel    18    M
9 resultado = list(df['Idade'] >= 18)
10 print(resultado)
11 # [False, True, True, True]
```

- Exemplo de como verificar as pessoas do sexo feminino.

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade Sexo
5 # A     Ana    17    F
6 # B   Bruno    19    M
7 # C   Carla    20    F
8 # D  Daniel    18    M
9 resultado = list(df['Sexo'] == 'F')
10 print(resultado)
11 # [True, False, True, False]
```

## Seleção de Dados em um DataFrame

- A aplicação de operadores lógicos em colunas juntamente com o indexador `loc` permite a seleção de dados de uma maneira bastante ágil.
- Como visto anteriormente, o resultado da aplicação de operadores lógicos em colunas é uma lista de booleanos representando as linhas que se adequam ao critério de seleção.
- O indexador `loc` permite utilizar como parâmetro uma lista com valores booleanos que representam as linhas que serão selecionadas.

# Seleção de Dados em um DataFrame

- Exemplo de como selecionar do DataFrame as pessoas que já atingiram a maioridade penal.

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade Sexo
5 # A     Ana    17    F
6 # B    Bruno   19    M
7 # C    Carla   20    F
8 # D   Daniel   18    M
9 resultado = list(df['Idade'] >= 18)
10 print(df.loc[resultado])
11 #      Nome  Idade Sexo
12 # B    Bruno   19    M
13 # C    Carla   20    F
14 # D   Daniel   18    M
```



# Seleção de Dados em um DataFrame

- Exemplo de como selecionar do DataFrame somente as pessoas do sexo feminino.

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade Sexo
5 # A     Ana    17    F
6 # B   Bruno    19    M
7 # C   Carla    20    F
8 # D  Daniel    18    M
9 resultado = list(df['Sexo'] == 'F')
10 print(df.loc[resultado])
11 #      Nome  Idade Sexo
12 # A     Ana    17    F
13 # C   Carla    20    F
```

# Seleção de Dados em um DataFrame

- Exemplo de como selecionar do DataFrame somente as pessoas do sexo feminino que atingiram a maioridade penal.

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade Sexo
5 # A     Ana    17    F
6 # B    Bruno    19    M
7 # C    Carla    20    F
8 # D   Daniel    18    M
9 resultado = list(df['Sexo'] == 'F')
10 df = df.loc[resultado]
11 resultado = list(df['Idade'] >= 18)
12 print(df.loc[resultado])
13 #      Nome  Idade Sexo
14 # C    Carla    20    F
```

# Ordenando um DataFrame

- Um DataFrame pode ser ordenado utilizando o método `sort_values`.
- O método `sort_values` possui alguns parâmetros:
  - `by`: string ou lista de strings especificando os rótulos que serão utilizados como chave para a ordenação.
  - `axis`: ordenação de linhas (padrão: 0) ou de colunas (1).
  - `ascending`: ordenação crescente ou decrescente (padrão: True).
  - `kind`: algoritmo de ordenação que será utilizado (padrão: quicksort).
  - `inplace`: define se a ordenação deve ser aplicada diretamente no DataFrame ou em uma cópia (padrão: False).

# Ordenando um DataFrame

- Exemplo de ordenação de um DataFrame.

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade  Sexo
5 # A     Ana    17    F
6 # B    Bruno   19    M
7 # C    Carla   20    F
8 # D   Daniel   18    M
9 df.sort_values(by = 'Idade', ascending = False,
10               inplace = True)
11 print(df)
12 #      Nome  Idade  Sexo
13 # C    Carla   20    F
14 # B    Bruno   19    M
15 # D   Daniel   18    M
16 # A     Ana    17    F
```

# Ordenando um DataFrame

- Exemplo de ordenação com duas chaves.

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade  Sexo
5 # A     Ana    17    F
6 # B    Bruno   19    M
7 # C    Carla   20    F
8 # D   Daniel   18    M
9 df.sort_values(by = ['Sexo', 'Idade'], inplace = True)
10 print(df)
11 #      Nome  Idade  Sexo
12 # A     Ana    17    F
13 # C    Carla   20    F
14 # D   Daniel   18    M
15 # B    Bruno   19    M
```

# Ordenando um DataFrame

- É possível também ordenar um DataFrame pelos seus rótulos utilizando o método `sort_index`.
- O método `sort_index` possui alguns parâmetros:
  - `axis`: ordenação de linhas (padrão: 0) ou de colunas (1).
  - `ascending`: ordenação crescente ou decrescente (padrão: `True`).
  - `kind`: algoritmo de ordenação que será utilizado (padrão: `quicksort`).
  - `inplace`: define se a ordenação deve ser aplicada diretamente no DataFrame ou em uma cópia (padrão: `False`).

# Ordenando um DataFrame

- Exemplo de ordenação de um DataFrame pelos rótulos das colunas.

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade  Sexo
5 # A     Ana    17    F
6 # B    Bruno   19    M
7 # C    Carla   20    F
8 # D   Daniel   18    M
9 df.sort_index(axis = 1, inplace = True)
10 print(df)
11 #      Idade  Nome  Sexo
12 # A     17    Ana    F
13 # B     19   Bruno   M
14 # C     20   Carla   F
15 # D     18  Daniel   M
```

# Ordenando um DataFrame

- Exemplo de ordenação de um DataFrame pelos rótulos das linhas de forma decrescente.

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade  Sexo
5 # A     Ana    17    F
6 # B    Bruno   19    M
7 # C    Carla   20    F
8 # D   Daniel   18    M
9 df.sort_index(ascending = False, inplace = True)
10 print(df)
11 #      Nome  Idade  Sexo
12 # D   Daniel   18    M
13 # C    Carla   20    F
14 # B    Bruno   19    M
15 # A     Ana    17    F
```



- A biblioteca pandas possui vários métodos para realização de cálculos em colunas:
  - `abs`: retorna uma lista com os valores absolutos da coluna.
  - `count`: conta o número de células da coluna que possuem valores disponíveis.
  - `nunique`: conta os valores distintos na coluna.
  - `sum`: calcula a soma dos valores da coluna.
  - `min`: obtém o menor valor da coluna.
  - `max`: obtém o maior valor da coluna.
  - `mean`: calcula a média dos valores da coluna.
  - `median`: obtém a mediana dos valores da coluna.

- `copy`: retorna uma cópia do `DataFrame`.
- `head`: retorna as `n` primeiras linhas do `DataFrame` (padrão: 5).
- `tail`: retorna as `n` últimas linhas do `DataFrame` (padrão: 5).

- Exemplo de métodos aritméticos.

```
1 import pandas as pd
2 print(df)
3      Nome  Idade  Sexo
4  A     Ana    17    F
5  B    Bruno    19    M
6  C    Carla    22    F
7  D  Daniel    18    M
8 print(df.Idade.count())
9 # 4
10 print(df.Idade.sum())
11 # 74
12 print(df.Idade.min(), df.Idade.max())
13 # 17 21
14 print(df.Idade.mean())
15 # 19
16 print(df.Idade.median())
17 # 18.5
```

- A biblioteca pandas possui vários métodos para aplicação em matrizes:
  - `add`: soma os elementos das posições correspondentes das matrizes.
  - `sub`: subtrai os elementos das posições correspondentes das matrizes.
  - `div`: realiza a divisão real entre os elementos das posições correspondentes das matrizes.
  - `mul`: multiplica os elementos das posições correspondentes das matrizes.
  - `eq`: verifica se os elementos das posições correspondentes das matrizes são iguais.
  - `ne`: verifica se os elementos das posições correspondentes das matrizes são diferentes.
  - `dot`: realiza a multiplicação das matrizes.

# Operações com Matrizes

- Exemplo de operações com matrizes.

```
1 import pandas as pd
2 df1 = pd.DataFrame([[19, 23, 34],
3                     [80, 75, 60],
4                     [25, 32, 15]])
5 print(df1)
6 #      0    1    2
7 # 0   19   23   34
8 # 1   80   75   60
9 # 2   25   32   15
10 df2 = pd.DataFrame([[21, 27, 35],
11                    [85, 70, 60],
12                    [25, 50, 15]])
13 print(df2)
14 #      0    1    2
15 # 0   21   27   35
16 # 1   85   70   60
17 # 2   25   50   15
```

# Operações com Matrizes

- Exemplo de operações com matrizes.

```
1 print(df1.add(df2))
2 #      0      1      2
3 # 0    40    50    69
4 # 1   165   145   120
5 # 2    50    82    30
6 print(df1.sub(df2))
7 #      0      1      2
8 # 0  -2   -4   -1
9 # 1  -5    5    0
10 # 2   0  -18    0
11 print(df1.div(df2))
12 #              0              1              2
13 # 0   0.904762   0.851852   0.971429
14 # 1   0.941176   1.071429   1.000000
15 # 2   1.000000   0.640000   1.000000
```

# Operações com Matrizes

- Exemplo de operações com matrizes.

```
1 print(df1.mul(df2))
2 #          0          1          2
3 # 0    399    621   1190
4 # 1   6800   5250   3600
5 # 2    625   1600    225
6 print(df1.eq(df2))
7 #          0          1          2
8 # 0  False  False  False
9 # 1  False  False   True
10 # 2   True  False   True
11 print(df1.ne(df2))
12 #          0          1          2
13 # 0   True   True   True
14 # 1   True   True  False
15 # 2  False   True  False
```

- Exemplo de operações com matrizes.

```
1 print(df1.dot(df2))
2 #      0      1      2
3 # 0  3204  3823  2555
4 # 1  9555 10410  8200
5 # 2  3620  3665  3020
6 print(df2.dot(df1))
7 #      0      1      2
8 # 0  3434  3628  2859
9 # 1  8715  9125  7990
10 # 2  4850  4805  4075
```



- Podemos transformar um objeto do tipo DataFrame em um array da biblioteca NumPy.
- Para isto, devemos utilizar o método `to_numpy`.

# Transformação em NumPy Array

- Exemplo:

```
1 import pandas as pd
2 df = pd.DataFrame({'Idade': [20, 21, 25],
3                     'Altura': [1.75, 1.60, 1.89],
4                     'Peso': [80, 70, 85]},
5                     index = ['Andre', 'Bruna', 'Carlos'])
6 print(df)
7 #           Idade  Altura  Peso
8 # Andre         20    1.75   80
9 # Bruna         21    1.60   70
10 # Carlos        25    1.89   85
11 print(df.to_numpy())
12 # [[20.  1.75  80.]
13 #   [21.  1.60  70.]
14 #   [25.  1.89  85.]
```

- Exemplo:

```
1 import pandas as pd
2 ...
3 print(df.loc[:, 'Idade'].to_numpy())
4 # [20 21 25]
5 print(df.loc[:, 'Altura'].to_numpy())
6 # [1.75 1.60 1.89]
7 print(df.loc[:, 'Peso'].to_numpy())
8 # [80 70 85]
```

# **Importando e Exportando Dados**

---

- A biblioteca pandas fornece uma forma rápida e fácil para exportar os dados de um DataFrame para diferentes formatos.
- Entre os diversos formatos disponíveis, iremos focar no formato CSV (*Comma-Separated Values*, ou *Valores Separados por Vírgulas*).
- Para realizar essa tarefa, temos o método `to_csv`.
- Alguns dos parâmetros desse método são:
  - `path_or_buf`: caminho ou buffer onde o arquivo deve ser salvo.
  - `sep`: caractere separador do arquivo (o padrão é a 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 ou não (padrão: `True`).

- Exemplo de como exportar os dados de um DataFrame para um arquivo CSV.

```
1 import pandas as pd
2 ...
3 print(df)
4 #      Nome  Idade  Sexo
5 # A     Ana    17    F
6 # B    Bruno   19    M
7 # C    Carla   20    F
8 # D   Daniel   18    M
9 df.to_csv('dados.csv')
```

- Para importar um arquivo CSV, a biblioteca `pandas` fornece a função `read_csv`.
- Alguns dos parâmetros desse método são:
  - `filepath_or_buffer`: caminho ou buffer até o arquivo CSV.
  - `sep`: caractere separador do arquivo (o padrão é a vírgula).
  - `names`: lista de rótulos para serem utilizados nas colunas.
  - `header`: linha do arquivo CSV para ser utilizada como rótulos para as colunas.
  - `index_col`: coluna do arquivo CSV para ser utilizada como rótulos para as linhas.

- Exemplo de como inportar os dados de um arquivo CSV para um DataFrame.

```
1 import pandas as pd
2 df = pd.read_csv('dados.csv', index_col = 0, header = 0)
3 print(df)
4 #      Nome  Idade  Sexo
5 # A     Ana    17     F
6 # B    Bruno    19     M
7 # C    Carla    20     F
8 # D   Daniel    18     M
```



# Documentação

---

- A biblioteca pandas fornece uma documentação vasta e detalhada.
- Para mais informações visite:  
<https://pandas.pydata.org/pandas-docs/stable/reference/index.html>
- Documentação sobre DataFrame:  
<https://pandas.pydata.org/pandas-docs/stable/reference/frame.html>