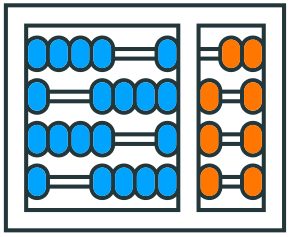

Instituto de Computação

UNIVERSIDADE ESTADUAL DE CAMPINAS

UNICAMP

MC102 – Aula 17

Pandas

Algoritmos e Programação de Computadores

Zanoni Dias

2022

Instituto de Computação

Roteiro

Pandas

DataFrame

Manipulação de Dados

Importando e Exportando Dados

Documentação

Pandas

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:

```
pip install pandas
```

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

Pandas

- 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

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 heterogênios, 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.

Criando um DataFrame

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

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

```
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
```

Modificando os Rótulos de uma 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)
#         0   1
# 0     Ana  21
# 1   Bruno  20
# 2   Carla  22
df.columns = ['Nome', 'Idade']
df.index = ['A', 'B', 'C']
print(df)
#      Nome  Idade
# A     Ana     21
# B   Bruno     20
# 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:

```
import pandas as pd
...
print(df)
#     Nome  Idade
# A    Ana     21
# B  Bruno     20
# C  Carla     22
print(list(df.index))
# ['A', 'B', 'C']
print(list(df.columns))
# ['Nome', 'Idade']
```

Atributos de um DataFrame

- Exemplos:

```
import pandas as pd
...
print(df)
#     Nome  Idade
# A    Ana     21
# B  Bruno     20
# C  Carla     22
print(df.ndim)
# 2
print(df.shape)
# (3, 2)
print(df.size)
# 6
print(df.empty)
# 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:

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

- Exemplo:

```
import pandas as pd
dados = [('Ana', 21), ('Bruno', 20), ('Carla', 22)]
df = pd.DataFrame(data = dados)
print(df)
#         0   1
# 0     Ana  21
# 1   Bruno  20
# 2   Carla  22
print(df[0][0], df[0][1], df[0][2])
# Ana Bruno Carla
```

Indexadores

- 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.

Indexadores

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

```
import pandas as pd
...
print(df)
#      Nome  Idade
# A     Ana     21
# B   Bruno     20
# C   Carla     22
print(df.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.

```python
import pandas as pd
...
print(df)
#      Nome  Idade
# A     Ana     21
# B   Bruno     20
# C   Carla     22
df.at['C', 'Nome']
# 'Carla'
df.at['C', 'Idade']
# 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.

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

Indexadores

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

```
import pandas as pd
...
print(df)
#      Nome  Idade
# A     Ana     21
# B   Bruno     20
# C   Carla     22
print(df.iat[0, 0])
# Ana
print(df.iat[0, 1])
# 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.

```
import pandas as pd
...
print(df)
#     Nome  Idade
# A    Ana     21
# B  Bruno     20
# C  Carla     22
print(df.iat[1, 0])
# Bruno
print(df.iat['B', 'Idade'])
# ValueError: iAt based indexing can only have integer
    indexers
```

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
...
print(df)
#     Nome  Idade
# A    Ana     21
# B  Bruno     20
# C  Carla     22
print(df.loc[['A', 'C']])
#     Nome  Idade
# A    Ana     21
# C  Carla     22
```

Indexadores

- Mais exemplos com o indexador `loc`.

```
import pandas as pd
...
print(df)
#     Nome  Idade
# A    Ana     21
# B  Bruno     20
# C  Carla     22
print(df.loc[[True, False, True]])
#     Nome  Idade
# A    Ana     21
# C  Carla     22
print(df.loc[[True, False, True], 'Nome'])
# A      Ana
# C    Carla
# Name: Nome, dtype: object
```

Indexadores

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

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

Indexadores

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

```
import pandas as pd
...
print(df)
#      Nome  Idade
# A     Ana     21
# B   Bruno     20
# C   Carla     22
print(df.iloc[[1, 2]])
#      Nome  Idade
# B   Bruno     20
# C   Carla     22
```

Indexadores

- Mais exemplos com o indexador `iloc`.

```
import pandas as pd
...
print(df)
#     Nome  Idade
# A    Ana     21
# B  Bruno     20
# C  Carla     22
print(df.iloc[-1])
# Nome      Carla
# Idade        22
# Name: C, dtype: object
print(df.iloc[[0,2],0])
# A      Ana
# C    Carla
# Name: Nome, dtype: object
```

Indexadores

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

```
import pandas as pd
...
print(df)
#     Nome  Idade
# A    Ana     21
# B  Bruno     20
# C  Carla     22
print(df.iloc[['B', 'C']])
# 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:

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

- Associando valores específicos para cada uma das linhas:

```
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:

```
import pandas as pd
...
print(df)
#      Nome  Idade
# A     Ana     21
# B   Bruno     20
# C   Carla     22
df['Sexo'] = 'F'
print(df)
#      Nome  Idade Sexo
# A     Ana     21    F
# B   Bruno     20    F
# 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 lista com os novos valores.
- Caso os rótulos das linhas não sejam compatíveis, o parâmetro `ignore_index` deve ser atribuido 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:

```
import pandas as pd
...
print(df1)
#      Nome  Idade Sexo
# A     Ana     21    F
# B   Bruno     20    M
dados = [ {'Nome': 'Carla', 'Idade': 22, 'Sexo': 'F'},
          {'Nome': 'Daniel', 'Idade': 18, 'Sexo': 'M'} ]

df2 = df1.append(dados, ignore_index = True)
print(df2)
#       Nome  Idade Sexo
# 0      Ana     21    F
# 1    Bruno     20    M
# 2    Carla     22    F
# 3   Daniel     18    M
```

Adicionando e Modificando Linhas de um DataFrame

- Exemplo do método append mantendo os rótulos das linhas:

```
import pandas as pd
...
print(df1)
#     Nome  Idade Sexo
# A    Ana     21    F
# B  Bruno     20    M
dados = [ {'Nome': 'Carla', 'Idade': 22, 'Sexo': 'F'},
          {'Nome': 'Daniel', 'Idade': 18, 'Sexo': 'M'} ]
df2 = pd.DataFrame(dados, index = ['D','E'])
df3 = df1.append(df2, ignore_index = False)
print(df3)
#      Nome  Idade Sexo
# A     Ana     21    F
# B   Bruno     20    M
# C   Carla     22    F
# 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:

```
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.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:

```
import pandas as pd
...
print(df)
#       Nome  Idade Sexo
# A      Ana     21    F
# B    Bruno     20    M
df.loc['B'] = ['Bento', 22, 'M']
df.loc['C'] = ['Carla', 22, 'F']
df.loc['D'] = ['Daniela', 18, 'F']
print(df)
#        Nome  Idade Sexo
# A       Ana     21    F
# B     Bento     22    M
# C     Carla     22    F
# D   Daniela     18    F
```

Adicionando e Modificando Linhas de um DataFrame

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

```
import pandas as pd
...
print(df)
#       Nome  Idade Sexo
# A      Ana     21    F
# B    Bento     22    M
# C    Carla     22    F
# D  Daniela     18    F
df.iloc[1] = ['Bruno', 19, 'M']
df.iloc[3] = ['Daniel', 18, 'M']
print(df)
#      Nome  Idade Sexo
# A     Ana     21    F
# B   Bruno     19    M
# C   Carla     22    F
# 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.

```
df.at[<rótulo>, <rótulo>] = <novo_valor>
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`: determinar se as mudanças desem 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
```

Operadores

- 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`.

```
import pandas as pd
...
print(df)
#       Nome  Idade Sexo
# A      Ana     17    F
# B    Bruno     19    M
# C    Carla     20    F
# D   Daniel     18    M
df['Idade'] += 1
print(df)
#       Nome  Idade Sexo
# A      Ana     18    F
# B    Bruno     20    M
# C    Carla     21    F
# 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.

```
import pandas as pd
...
print(df)
#       Nome  Idade Sexo
# A      Ana     17    F
# B    Bruno     19    M
# C    Carla     20    F
# D   Daniel     18    M
resultado = list(df['Idade'] >= 18)
print(resultado)
# [False, True, True, True]
```

Operadores

- Exemplo de como verificar as pessoas do sexo feminino.

```
import pandas as pd
...
print(df)
#       Nome  Idade Sexo
# A      Ana     17    F
# B    Bruno     19    M
# C    Carla     20    F
# D   Daniel     18    M
resultado = list(df['Sexo'] == 'F')
print(resultado)
# [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.

```
import pandas as pd
...
print(df)
#       Nome  Idade Sexo
# A      Ana     17    F
# B    Bruno     19    M
# C    Carla     20    F
# D   Daniel     18    M
resultado = list(df['Idade'] >= 18)
print(df.loc[resultado])
#       Nome  Idade Sexo
# B    Bruno     19    M
# C    Carla     20    F
# D   Daniel     18    M
```

Seleção de Dados em um DataFrame

- Exemplo de como selecionar do `DataFrame` somente as mulheres.

```
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 mulheres que não atingiram a maioridade penal.

```
import pandas as pd
...
print(df)
#       Nome  Idade Sexo
# A      Ana     17    F
# B    Bruno     19    M
# C    Carla     20    F
# D   Daniel     18    M
resultado = list(df['Sexo'] == 'F')
df = df.loc[resultado]
resultado = list(df['Idade'] < 18)
print(df.loc[resultado])
#    Nome  Idade Sexo
# A  Ana     17    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`: eixo de ordenação vertical (0, padrão) ou horizontal (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.

```
import pandas as pd
...
print(df)
#       Nome  Idade Sexo
# A      Ana     17    F
# B    Bruno     19    M
# C    Carla     20    F
# D   Daniel     18    M
df.sort_values(by = ['Sexo', 'Idade'], inplace = True)
print(df)
#       Nome  Idade Sexo
# A      Ana     17    F
# C    Carla     20    F
# D   Daniel     18    M
# 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`: eixo de ordenação vertical (0, padrão) ou horizontal (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.

```
import pandas as pd
...
print(df)
#      Nome  Idade Sexo
# A     Ana     17    F
# B   Bruno     19    M
# C   Carla     20    F
# D  Daniel     18    M
df.sort_index(axis = 1, inplace = True)
print(df)
#    Idade    Nome Sexo
# A     17     Ana    F
# B     19   Bruno    M
# C     20   Carla    F
# D     18  Daniel    M
```

Ordenando um DataFrame

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

```
import pandas as pd
...
print(df)
#       Nome  Idade Sexo
# A      Ana     17    F
# B    Bruno     19    M
# C    Carla     20    F
# D   Daniel     18    M
df.sort_index(ascending = False, inplace = True)
print(df)
#       Nome  Idade Sexo
# D   Daniel     18    M
# C    Carla     20    F
# B    Bruno     19    M
# A      Ana     17    F
```

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`: realiza a contagem de quantas células da coluna possuem valores disponíveis.
  - `nunique`: conta os valores distintos na coluna.
  - `sum`: retorna a soma dos valores da coluna.
  - `max`: retorna o maior valor da coluna.
  - `min`: retorna o menor valor da coluna.
  - `mean`: computa a média dos valores da coluna.
  - `median`: calcula a mediana dos valores da coluna.

Outros Métodos

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

Métodos Aritméticos

- Exemplo de métodos aritméticos.

```
import pandas as pd
print(df)
#       Nome  Idade Sexo
# A      Ana     17    F
# B    Bruno     19    M
# C    Carla     20    F
# D   Daniel     18    M
print(df.Idade.count())
# 4
print(df.Idade.sum())
# 74
print(df.Idade.max())
# 20
print(df.Idade.min())
# 17
print(df.Idade.mean())
# 18.5
print(df.Idade.median())
# 18.5
```

Operações com Matrizes

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

Importando e Exportando Dados

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).

Exportando Dados

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

```
import pandas as pd
...
print(df)
#       Nome  Idade Sexo
# A      Ana     17    F
# B    Bruno     19    M
# C    Carla     20    F
# D   Daniel     18    M
df.to_csv('dados.csv')
```

Exportando Dados

- 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.

Exportando Dados

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

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

Documentação

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