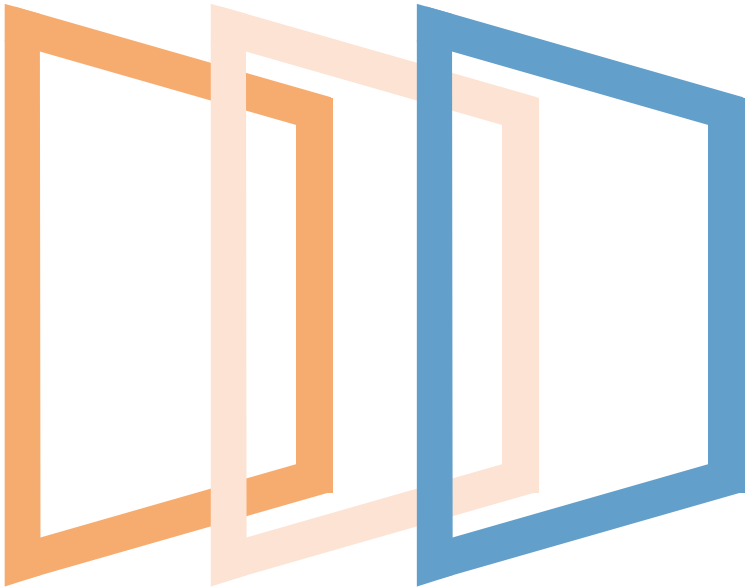
# Python Power: Exploração, Manipulação e Análise de Dados com Numpy e Pandas

Thaís Ratis

Inteligência Artificial Brasil, 10.03.2025



minsait

An Indra company

# Pandas

# Aula 02

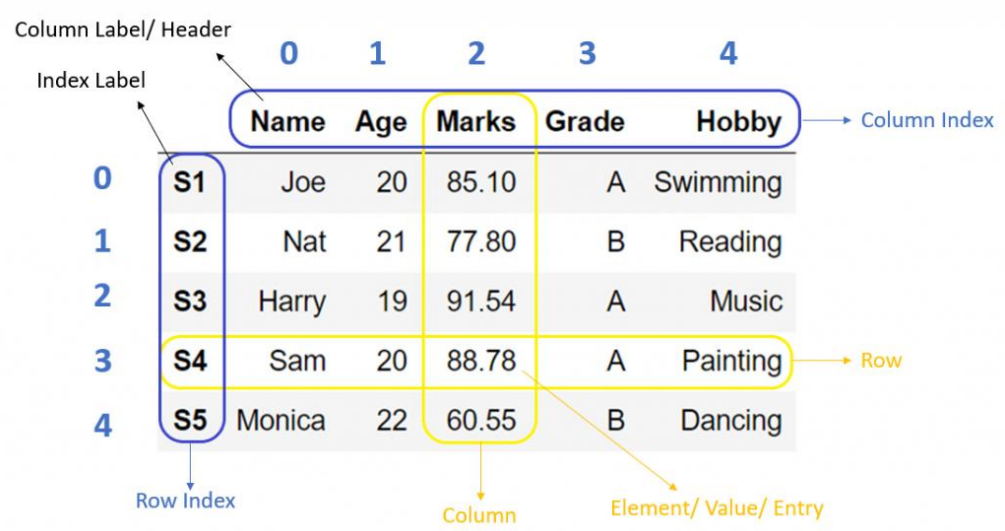
# Numpy

Embora NumPy forneça estruturas e ferramentas fundamentais que facilitam o trabalho com dados, existem algumas limitações para sua utilidade:

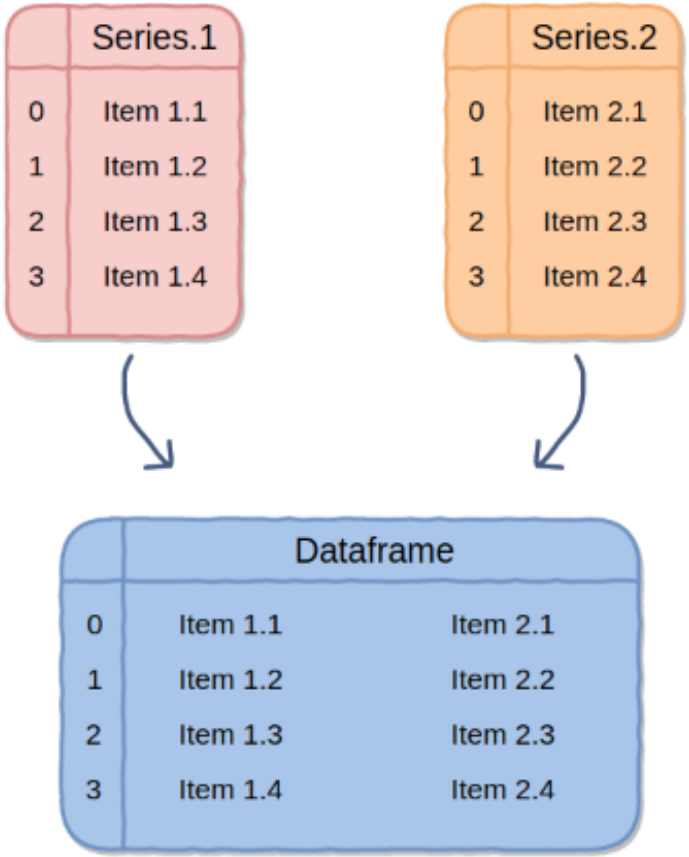
- A falta de suporte para nomes de colunas nos força a enquadrar as questões como arrays;
- O suporte a apenas um tipo de dado, torna difícil trabalhar com dados que contenham números e strings;
- Não existem muitos padrões para análises.

# Pandas

O Pandas não substitui o Numpy, mas o estende. Pandas é uma biblioteca poderosa para análise de dados. Ela oferece estruturas de dados flexíveis, como DataFrames, que permitem a manipulação, limpeza, transformação e análise de dados de forma eficiente. O Pandas também possui recursos para lidar com dados ausentes e integrar-se com outras bibliotecas de análise e visualização.



# Pandas – Tipos de dados




- Numéricos;
- Datas (datetime);
- Object, que o pandas usa para colunas que possuem dados, que não se encaixam em nenhum outro tipo.
  - Também normalmente é usado em dados do tipo *string*.

# Pandas vs Numpy – Accesos de elementos

In [1]:
```
import numpy as np
import pandas as pd

a = np.array([1,2,3,4])
series = pd.Series([1,2,3,4])

print (series.describe(),'\n')  # Função não disponível no numpy
```

```
count    4.000000
mean     2.500000
std      1.290994
min      1.000000
25%      1.750000
50%      2.500000
75%      3.250000
max      4.000000
dtype: float64
```

In [4]:
```
print (a[0],'\n')
print (a[:2],'\n')
```

```
1

[1 2]
```

In [5]:
```
print (series[0],'\n')
print (series[:2],'\n')
```

```
1

0    1
1    2
dtype: int64
```

# Pandas – Dataframes

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.

# Pandas – Dataframes

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
```

Criando um DataFrame a partir de um dicionário:

```
import pandas as pd
dados = {'Nome': ['Ana', 'Bruno', 'Carla'],
         'Idade': [21, 20, 22]}
# {'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
```

```
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
```

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

# Pandas – Dataframes

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).

# Pandas – Dataframes

```
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']
```

```
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
```

# Pandas – Dataframes

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
```

# Pandas vs Numpy - Operações

```
In [13]: print ('mean', a.mean())
         print ('std',  a.std())
         print ('max', a.max())
```

```
mean 2.5
std 1.118033988749895
max 4
```

```
In [14]: print ('mean', series.mean())
         print ('std',  series.std())
         print ('max', series.max())
```

```
mean 2.5
std 1.2909944487358056
max 4
```

**Population standard deviation:**

$$\sigma = \sqrt{\frac{\sum (x_i - \mu)^2}{N}}$$

**Sample standard deviation:**

$$s_x = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n-1}}$$

# Pandas vs Numpy – Desvio Padrão

```
numpy.std(a, axis=None, dtype=None, out=None, ddof=0, keepdims=<no value>, *, where=<no value>)
```
[source]

Means Delta Degrees of Freedom. The divisor used in calculations is N - ddof, where N represents the number of elements. By default ddof is zero.

```
DataFrame.std(axis=0, skipna=True, ddof=1, numeric_only=False, **kwargs)
```

ddofint, default 1. Delta Degrees of Freedom. The divisor used in calculations is N - ddof, where N represents the number of elements.

```
print('ddof = 1')
print ('std',  a.std(ddof=1))
print ('std',  series.std())

print('ddof = 0')
print ('std',  a.std())
print ('std',  series.std(ddof=0))
```

✓ 0.0s

```
ddof = 1
std 1.2909944487358056
std 1.2909944487358056
ddof = 0
std 1.118033988749895
std 1.118033988749895
```

# Pandas - Operações

O Pandas possui diversos métodos que podem ser utilizados nessa estrutura. Abaixo estão alguns métodos que essa estrutura de dados possui e facilitam alguns cálculos:

| Método | Descrição |
|---|---|
| `sum` | soma |
| `mean` | média |
| `median` | mediana |
| `std` | desvio padrão |
| `mode` | moda |
| `max` | valor máximo |
| `min` | valor mínimo |
| `value_counts` | contagem de valores |
| `describe` | estatísticas básicas |
| `abs` | valores absolutos |
| `count` | contagem de células |
| `unique` | valores únicos |
| `nunique` | contagem de valores únicos |

# Pandas vs Numpy - Filtros e operações

```
In [17]: a = pd.Series([1, 2, 3, 4])
         b = pd.Series([1, 2, 1, 2])

         print (a + b)
         print (a * 2)
         print (a >= 3)
         print (a[a >= 3])
```

```
0    2
1    4
2    4
3    6
dtype: int64
0    2
1    4
2    6
3    8
dtype: int64
0    False
1    False
2     True
3     True
dtype: bool
2    3
3    4
dtype: int64
```

```
In [16]: a = np.array([1, 2, 3, 4])
         b = np.array([1, 2, 1, 2])

         print (a + b)
         print (a * 2)
         print (a >= 3)
         print (a[a >= 3])
```

```
[2 4 4 6]
[2 4 6 8]
[False False  True  True]
[3 4]
```

# Pandas - Filtros e operações

```
valores = [1, 1, 2, 3, 5, 8, 13]
fibonacci = pd.Series(valores)

fibonacci.sum()
```

✓ 0.0s

```
33
```

```
valores = [1, 1, 2, 3, 5, 8, 13]
fibonacci = pd.Series(valores)

fibonacci[fibonacci > 4].sum()
```

✓ 0.0s

```
26
```

# Pandas vs Numpy – Accesos de elementos

In [22]:
```
population = pd.Series([1415045928,1354051854,326766748], index = ["China", "India", "US"])

print (population,'\n')
print ('Population of {} is {}'.format(population.index[1], population['India']))
```

```
China    1415045928
India    1354051854
US        326766748
dtype: int64

Population of India is 1354051854
```

In [23]:
```
import numpy as np

population = np.array([1415045928,1354051854,326766748])
index = np.array(["China", "India", "US"])

print (population,'\n')
print ('Population of {} is {}'.format(index[1], population[1]))
```

```
[1415045928 1354051854  326766748]

Population of India is 1354051854
```

# Pandas vs Lista - Filtros

## Pandas

```
In [28]: # Busca por um valor específico
         print(population==1415045928,'\n')

         China     True
         India    False
         US       False
         dtype: bool

In [30]: value = population[population==1415045928]
         value

Out[30]: China    1415045928
         dtype: int64
```

## Lista

```
In [31]: population = [1415045928,1354051854,326766748]

         for i in population:
             if i == 1415045928:
                 value = population[population==1415045928]

         print (value)

         1415045928
```

# Pandas Dataframe – Accesos de elementos

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.
- *loc*: seleção de elementos utilizando rótulos.
- *iloc*: seleção de elementos utilizando índices.

# Pandas Dataframe – Transposta

```
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
```

# Pandas Dataframe – Accesos de elementos com loc

```
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
```

```
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
```

# Pandas Dataframe – Accesos de elementos com iloc

```
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
```

```
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
```

# Pandas Dataframe – Modificando e Adicionando 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:

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>]
```

# Pandas Dataframe – Modificando e Adicionando colunas

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
```

Associando valores específicos:

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

# Pandas Dataframe – Modificando e Adicionando linhas

- 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.

```
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
```

```
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 = ['C','D'])
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
```

# Pandas Dataframe – Modificando e Adicionando linhas

- 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>]
```

# Pandas Dataframe – Modificando e Adicionando linhas

```
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
```

```
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
```

# Map, apply, applymap

O método map() só funciona em séries pandas onde diferentes tipos de operação podem ser aplicados aos itens da série.

Quando você aplica o método map(função) em uma série, a função map() pega cada elemento da série e aplica a função a ele e retorna a série transformada.

```
: from sklearn.datasets import load_iris
data = load_iris()
features = pd.DataFrame(data = data['data'], columns= data['feature_names'])
features.head()
```

:

| | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) |
|---|---|---|---|---|
| 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 |



```
In [59]: def cm_to_mm(cm):
             mm = cm * 10
             return mm

         features['conversion_SL'] = features['sepal length (cm)'].map(cm_to_mm)
         features.head()
```

Out[59]:

| | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) | conversion_SL |
|---|---|---|---|---|---|
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | 51.0 |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | 49.0 |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | 47.0 |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | 46.0 |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | 50.0 |

# Map, apply, applymap

O método apply () funciona em séries pandas e dataframes com uma variedade de funções facilmente aplicadas dependendo do tipo de dados.

Semelhante a map(), quando você usa o método apply() em uma série ou dataframe, a função pega cada elemento da série e aplica a função no elemento, então retorna a série ou dataframe transformados.

```
In [61]: features[['sepal length (cm)', 'sepal width (cm)']].map(cm_to_mm)
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
C:\Users\TALMEI~1\AppData\Local\Temp/ipykernel_17636/3128970834.py in <module>
----> 1 features[['sepal length (cm)', 'sepal width (cm)']].map(cm_to_mm)

~\Anaconda3\lib\site-packages\pandas\core\generic.py in __getattr__(self, name)
   5485         ):
   5486             return self[name]
-> 5487         return object.__getattribute__(self, name)
   5488
   5489     def __setattr__(self, name: str, value) -> None:

AttributeError: 'DataFrame' object has no attribute 'map'
```

```
In [62]: features[['sepal length (cm)',"sepal width (cm)"]].apply(cm_to_mm).head()
```

Out[62]:

| | sepal length (cm) | sepal width (cm) |
|---|---|---|
| 0 | 51.0 | 35.0 |
| 1 | 49.0 | 30.0 |
| 2 | 47.0 | 32.0 |
| 3 | 46.0 | 31.0 |
| 4 | 50.0 | 36.0 |

# Map, apply, applymap

O método applymap() funciona em todo o dataframe do pandas, onde a função de entrada é aplicada a cada elemento individualmente.

```
features
```

| | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) |
|---|---|---|---|---|
| **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 |



```
In [64]: features.applymap(cm_to_mm).head()
```

Out[64]:

| | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) |
|---|---|---|---|---|
| **0** | 51.0 | 35.0 | 14.0 | 2.0 |
| **1** | 49.0 | 30.0 | 14.0 | 2.0 |
| **2** | 47.0 | 32.0 | 13.0 | 2.0 |
| **3** | 46.0 | 31.0 | 15.0 | 2.0 |
| **4** | 50.0 | 36.0 | 14.0 | 2.0 |

# Pandas Dataframe – Removendo linhas e colunas

- 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*).

# Pandas Dataframe – Removendo linhas e colunas

```
print(df)
#       Nome  Idade Sexo
# A      Ana     17    F
# B    Bruno     19    M
# C    Carla     20    F
# D   Daniel     18    M
df.drop(index = ['A', 'D'], columns = ['Sexo'],
        inplace = True)
print(df)
#      Nome  Idade
# B   Bruno     19
# C   Carla     20
```

# Pandas Dataframe – Ordenando

- Um DataFrame pode ser ordenado utilizando o método *sort_values* para ordenar os valores ou *sort_index* para ordenar pelos rótulos.
- Os métodos possuem 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).

# Pandas Dataframe – Ordenando

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

```
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
```

# Pandas Dataframe – Ordenando

Ordenação pelos rótulos das colunas:

```
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
```

Ordenação pelos rótulos das linhas de forma decrescente:

```
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
```

# Pandas Dataframe – Estrutura

*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).

# Pandas Dataframe – Importação

- Para importar um arquivo CSV, a biblioteca pandas fornece a função *read_csv*.
- Alguns dos parâmetros desse método são:

  - *filepath*: caminho 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.

```
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
```

# Pandas Dataframe – Exportação

- 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*: caminho 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*).

```
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')
```

# Comparações

| Característica | Listas | Numpy | Pandas |
|---|---|---|---|
| **Estrutura de Dados** | Lista ordenada de elementos | Arrays N-dimensionais homogêneos | Estruturas de dados tabulares |
| **Tipo dos elementos** | Pode conter tipos diversos | Homogêneo (mesmo tipo de dado) | Pode conter tipos diversos |
| **Performance** | Menos eficiente para cálculos e operações com grandes conjuntos de dados | Altamente otimizado para cálculos e operações vetorizadas | Eficiente para manipulação de dados grandes |
| **Funcionalidades** | Possui funcionalidades básicas para manipulação de elementos | Oferece funções matemáticas e operações vetorizadas | Oferece ferramentas de análise de dados sofisticadas |
| **Indexação** | Acesso aos elementos é realizado por índices | Acesso aos elementos é realizado por índices | Oferece indexação personalizada (por rótulos) além de indexação por índices |
| **Operações matemáticas** | Suporta algumas operações básicas | Oferece uma ampla gama de funções matemáticas e operações vetorizadas | Oferece operações matemáticas e estatísticas |
| **Funções de Agregação** | Limitadas | Oferece funções de agregação poderosas (por exemplo, soma, média, mínimo, máximo) | Oferece funções de agregação avançadas |
| **Manipulação de dados** | Limitada | Possui métodos para filtragem, seleção, transformação e agrupamento de dados | Oferece recursos completos para manipulação e limpeza de dados |
| **Compatibilidade** | Integra-se bem com Python e outras bibliotecas | Compatível com outras bibliotecas científicas e matemáticas | Compatível com outras bibliotecas de análise de dados e visualização |

# JUPYTER NOTEBOOK
# +
# ATIVIDADE PRÁTICA

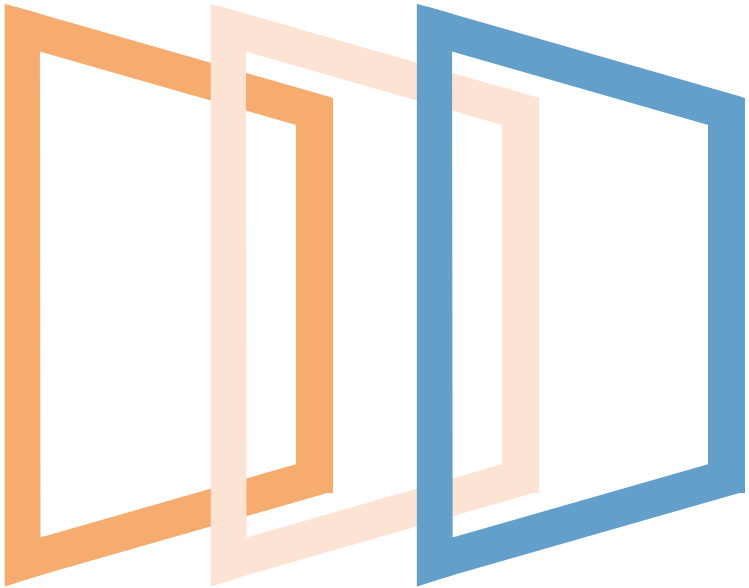
# Referências


- pandas - Python Data Analysis Library (pydata.org)
- Pandas Tutorial (w3schools.com)
- Tutorial Pandas slides (jmsevillam.github.io)
- https://stack-academy.memberkit.com.br/32408-data-science-do-zero
- Pandas - Algoritmos e Programação de Computadores (unicamp.br)

# Python Power: Exploração, Manipulação e Análise de Dados com Numpy e Pandas

Thaís Ratis

Inteligência Artificial Brasil, 10.03.2025



minsait

An Indra company