
Aula 3 - Numpy e Pandas

AGENDA

- Tipo básico NDARRAY
- Criação de um NDARRAY
- Funções básicas
- Operações básicas
- Iteração
- Manipulação de Forma
- Plotagem de Histograma

Biblioteca NumPy

- NumPy é um projeto de código aberto voltado para computação numérica
- Permite a manipulação de Array multidimensionais
- Operações básicas de Álgebra Linear
- Operações estatísticas básicas



Tipo básico: NDARRAY

- A estrutura básica para se trabalhar com NumPy são os arrays
- Os *arrays* possuem dimensão fixa, definida durante sua criação
- Diferentemente das listas em Python os elementos contidos no *array* devem ser do mesmo tipo
- Propriedades básicas de um NDARRAY
 - `ndarray.ndim` - Número de dimensões
 - `ndarray.shape` - Tupla de inteiros que indica o tamanho de cada dimensão
 - `ndarray.size` - Número total de elementos no *array*
 - `ndarray.dtype` - Tipo dos elementos contidos no *array*

Tipo básico: NDARRAY - Exemplo

```
[1] import numpy as np

array = np.array([[5,7,9],[6,8,10]])#criação de um NDARRAY com base em listas
print(array)
```

```
[[ 5  7  9]
 [ 6  8 10]]
```

```
[2] print(array.ndim)
```

```
2
```

```
[3] print(array.shape)
```

```
(2, 3)
```

```
[4] print(array.dtype)
```

```
int64
```

```
[5] print(array.size)
```

```
6
```

Criação de um NDARRAY

- Cria um array de dimensões 4 x 3
preenchido com zeros

```
[17] array_zeros = np.zeros((4,3))  
     print(array_zeros)
```

```
[[0.  0.  0.]  
 [0.  0.  0.]  
 [0.  0.  0.]  
 [0.  0.  0.]
```

Criação de um NDARRAY

- Cria um array de dimensões 4 x 3
preenchido com uns

```
[18] array_ones = np.ones((4,3))  
     print(array_ones)
```

```
[[1.  1.  1.]  
 [1.  1.  1.]  
 [1.  1.  1.]  
 [1.  1.  1.]]
```

Criação de um NDARRAY

- Cria um preenchido com números aleatórios que dependem do estado atual da memória

```
[26] array_empty = np.empty((5,3))  
      print(array_empty)
```

```
[[4.63581274e-310 1.03977794e-312 1.01855798e-312]  
 [9.54898106e-313 1.10343781e-312 1.01855798e-312]  
 [1.23075756e-312 1.01855798e-312 1.03977794e-312]  
 [9.76118064e-313 1.14587773e-312 1.14587773e-312]  
 [1.16709769e-312 4.44659081e-322 0.00000000e+000]]
```


Criação de um NDARRAY - Arrange

- Cria *array* preenchido com números pertencentes à um dado intervalo espaçados de determinado valor

```
[6] array_arange = np.arange( 10, 30, 5 )  
    print(array_arange)
```

```
[10 15 20 25]
```

Funções básicas

- Funções básicas que podem ser aplicadas sobre um *array*
 - ◆ `np.exp(V)`
 - ◆ `np.sqrt(V)`
 - ◆ `np.add(V,U)`

NDARRAY - Indexação

- O *numpy* permite diferentes formas de se indexar um *array*
 - ◆ *ndarray[i]* - retorna o conteúdo da posição *i*
 - ◆ *ndarray[i:j]* - retorna os valores armazenados nas posições de *i* até *j-1*
 - ◆ *ndarray[i,j,k]* - indexação multidimensional
 - ◆ *ndarray[:,1]* - todas as linhas da segunda coluna
 - ◆ *ndarray[1,:]* - todas as colunas da segunda linha
- Os modos de indexação ainda podem ser combinados de forma a obter diferentes resultados

NDARRAY - Iteração

- A iteração sobre um NDARRAY pode ser feita da mesma forma que as listas como apresentado na aula anterior
- Para cada dimensão do *Array* utiliza-se um *for* para chegar até os elementos

```
[3] array = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

```
for linha in array:  
    for coluna in linha:  
        print(coluna)
```

```
1  
2  
3  
4  
5  
6  
7  
8  
9
```

NDARRAY - Iteração utilizando *flat*

- O *Numpy* oferece uma maneira mais simples de se iterar sobre os elementos unitários de um array multidimensional utilizando a propriedade *flat*

```
[4] array = np.array([[1,2,3],[4,5,6],[7,8,9]])  
    for item in array.flat:  
        print(item)
```

```
1  
2  
3  
4  
5  
6  
7  
8  
9
```

NDARRAY - Manipulação de forma

- O Numpy permite que o usuário mude a forma do vetor, aumentando ou diminuindo suas dimensões distribuindo os elementos entre as dimensões
- Por exemplo um vetor com 10 elementos em apenas um dimensão pode ser manipulado de forma que passe a ter duas dimensões 5 e 2
- Neste exemplo temos uma matriz bidimensional onde foi especificado que cada linha deve conter 5 elementos e cada coluna 2 elementos

NDARRAY - Manipulação de forma

- Esta manipulação é feita por meio do método `NDARRAY.reshape()`
- Se uma das dimensões for especificada com `-1` o numpy irá calcular o tamanho desta dimensão de acordo com o valor das demais

```
[8] array = np.array([1,2,3,4,5,6,7,8,9,10])  
    print(array.shape)
```

```
(10,)
```

```
[10] novo_array = array.reshape(5,2)  
    print(novo_array)  
    print("Shape: ", novo_array.shape)
```

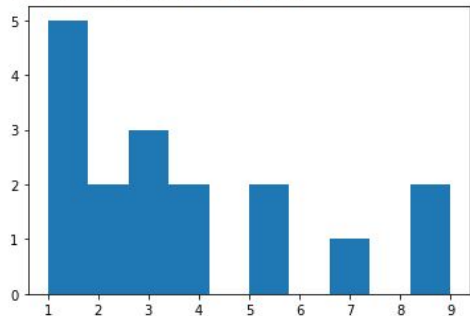
```
[[ 1  2]  
 [ 3  4]  
 [ 5  6]  
 [ 7  8]  
 [ 9 10]]  
Shape: (5, 2)
```

Plotagem de histograma

- Uma forma de visualizar distribuição dos dados dentro de um *array*
- Essa operação pode ser realizada utilizando a biblioteca *matplotlib*

```
[13] import matplotlib.pyplot as plt  
v = np.array([1,1,4,4,3,3,3,1,2,2,5,5,1,1,7,9,9])  
plt.hist(v)
```

```
(array([5., 2., 3., 2., 0., 2., 0., 1., 0., 2.]),  
 array([1. , 1.8, 2.6, 3.4, 4.2, 5. , 5.8, 6.6, 7.4, 8.2, 9. ]),  
 <a list of 10 Patch objects>)
```

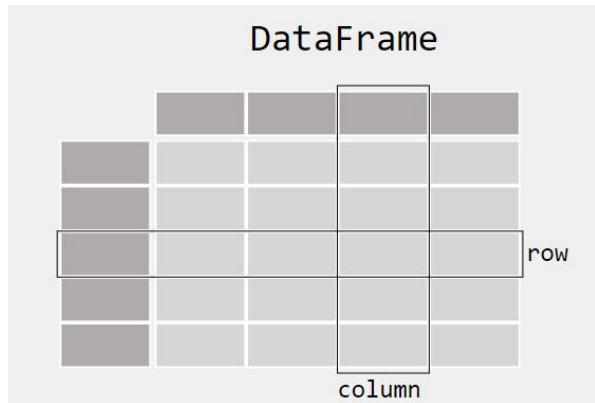


Biblioteca Pandas

- É uma biblioteca open-source criada para Python
- Alta performance, voltada para uso de estruturas de dados e análise

Tipo básico - O DataFrame

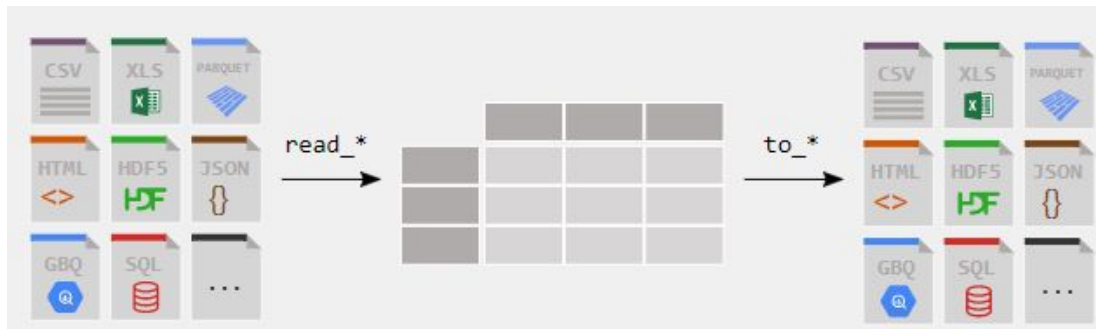
- Tipo básico de dados que usaremos do Pandas é o DataFrame
- Possui formato tabular
- Permite colunas de dados com tipos diferentes de dados
- Pode ser criado com base em diferentes tipos de arquivo como CSV ou XLSX



Fonte: [Getting started — pandas 1.2.4 documentation \(pydata.org\)](https://pandas.pydata.org/pandas-docs/stable/10min.html)

Lendo/Escrevendo dados

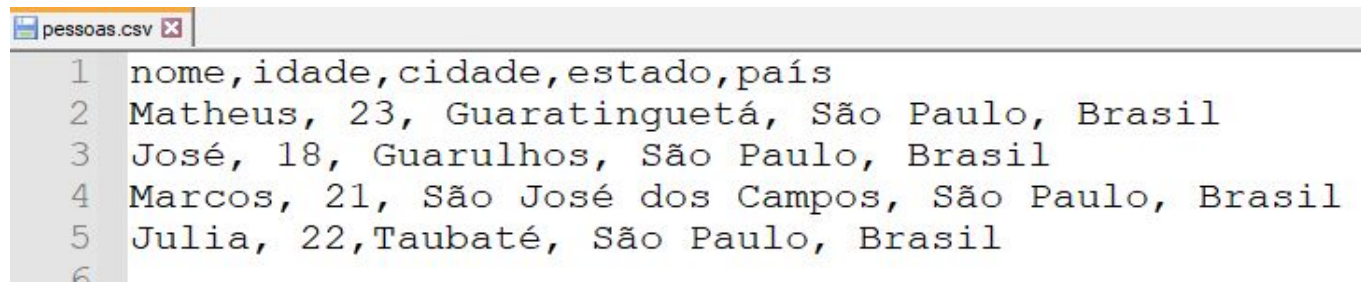
- As funções `read_<formato_entrada>` permitem a leitura de uma grande variedade de dados para um *DataFrame*
- As funções `to_<formato_saida>` permitem a escrita do *DataFrame* em arquivo conforme o formato especificado



Fonte: [Getting started — pandas 1.2.4 documentation \(pydata.org\)](https://pandas.pydata.org/docs/1.2.4/getting_started/)

Exemplo - Leitura de CSV

→ Arquivo de entrada:



```
pessoas.csv x
1 nome,idade,cidade,estado,país
2 Matheus, 23, Guaratinguetá, São Paulo, Brasil
3 José, 18, Guarulhos, São Paulo, Brasil
4 Marcos, 21, São José dos Campos, São Paulo, Brasil
5 Julia, 22, Taubaté, São Paulo, Brasil
6
```

→ DataFrame:

```
[14] import pandas as pd
```

```
data_frame = pd.read_csv('pessoas.csv')
print(data_frame)
```

	nome	idade	cidade	estado	país
0	Matheus	23	Guaratinguetá	São Paulo	Brasil
1	José	18	Guarulhos	São Paulo	Brasil
2	Marcos	21	São José dos Campos	São Paulo	Brasil
3	Julia	22	Taubaté	São Paulo	Brasil

Seleção de dados com *loc*

- O método *loc* presente nos *DataFrames* permite selecionar dados especificando linha e coluna
- Sintaxe: *data_frame.loc[indice]*

```
[18] print(data_frame.loc[3])
```

```
nome      Julia
idade
cidade     Taubaté
estado    São Paulo
país       Brasil
Name: 3, dtype: object
```

```
[22] print(data_frame.loc[[3,1]])
```

	nome	idade	cidade	estado	país
3	Julia		Taubaté	São Paulo	Brasil
1	José	18	Guarulhos	São Paulo	Brasil

Seleção de dados com *loc*

- O método *loc* também permite a indexação por meio de valores booleanos, como pode ser observado no exemplo abaixo.

```
[24] print(data_frame.loc[data_frame['nome'] == 'Matheus'])
```

	nome	idade	cidade	estado	país
0	Matheus	23	Guaratinguetá	São Paulo	Brasil

Seleção de dados com *loc*

- O método *loc* também permite a indexação por meio de valores booleanos, como pode ser observado no exemplo abaixo.

```
[24] print(data_frame.loc[data_frame['nome'] == 'Matheus'])
```

	nome	idade	cidade	estado	país
0	Matheus	23	Guaratinguetá	São Paulo	Brasil

Índice personalizado

- Durante a leitura de um arquivo usando a função `read_<format>` pode-se especificar qual coluna do arquivo de entrada deve ser usada.
- Parâmetro `index_col`: pode-se especificar o nome da coluna ou o índice

```
[42] df = pd.read_csv('pessoas.csv', index_col='nome')  
      print(df)
```

	idade	cidade	estado	país
nome				
Matheus	23	Guaratinguetá	São Paulo	Brasil
José	18	Guarulhos	São Paulo	Brasil
Marcos	21	São José dos Campos	São Paulo	Brasil
Julia		Taubaté	São Paulo	Brasil

```
[43] print(df.loc['Matheus'])
```

```
idade      23  
cidade    Guaratinguetá  
estado     São Paulo  
país        Brasil  
Name: Matheus, dtype: object
```


Operações Básicas - Média

- `df.mean()` - uma lista com o valor médio de cada coluna do DataFrame

```
[47] dates = pd.date_range('1/1/2000', periods=8)
      df = pd.DataFrame(np.random.randn(8, 4), index=dates, columns=['A', 'B', 'C', 'D'])
      print(df)
```

	A	B	C	D
2000-01-01	-1.197954	0.060951	0.610216	0.303173
2000-01-02	0.041010	0.111412	-0.223312	0.841213
2000-01-03	0.085076	-0.469314	0.331651	0.004190
2000-01-04	0.171380	1.037958	1.726166	-0.883824
2000-01-05	-0.621607	-1.183216	1.121509	1.720259
2000-01-06	0.703929	-0.142703	-0.302237	-1.048073
2000-01-07	0.157576	0.063335	0.112254	-1.051250
2000-01-08	-0.019837	0.254561	-0.203744	-0.378953

```
[48] print(df.mean())
```

```
A    -0.085053
B    -0.033377
C     0.396563
D    -0.061658
dtype: float64
```

Operações básicas - *df.apply()*

- `df.apply()` - aplica a função passada por parâmetro sobre os dados

```
[51] def somaUm(x):  
      return x+1  
      df = df.apply(somaUm)  
      print(df)
```

	A	B	C	D
2000-01-01	0.802046	2.060951	2.610216	2.303173
2000-01-02	2.041010	2.111412	1.776688	2.841213
2000-01-03	2.085076	1.530686	2.331651	2.004190
2000-01-04	2.171380	3.037958	3.726166	1.116176
2000-01-05	1.378393	0.816784	3.121509	3.720259
2000-01-06	2.703929	1.857297	1.697763	0.951927
2000-01-07	2.157576	2.063335	2.112254	0.948750
2000-01-08	1.980163	2.254561	1.796256	1.621047

Removendo Colunas e Linhas

- `df.drop()` - remove linhas/colunas de acordo com o parâmetro especificado

```
[54] df.drop(columns=['cidade', 'estado'])
```

	idade	país
nome		
Matheus	23	Brasil
José	18	Brasil
Marcos	21	Brasil
Julia		Brasil

```
[55] df.drop(['Matheus', 'Julia'])
```

	idade	cidade	estado	país
nome				
José	18	Guarulhos	São Paulo	Brasil
Marcos	21	São José dos Campos	São Paulo	Brasil

Iterando sobre o DataFrame

- `df.iterrows()` - permite iterar sobre o *DataFrame* obtendo se o índice e os dados

```
[58] for i, linha in df.iterrows():  
      print('Index: ', i)  
      print('Linha:\n', linha)  
      print('-----')
```

```
Index: Matheus  
Linha:  
  idade          23  
cidade    Guaratinguetá  
estado      São Paulo  
país        Brasil  
Name: Matheus, dtype: object
```

```
-----  
Index: José  
Linha:  
  idade          18  
cidade    Guarulhos  
estado      São Paulo  
país        Brasil  
Name: José, dtype: object
```

Iterando sobre o DataFrame

- `df.iterrows()` - permite iterar sobre o *DataFrame* obtendo se o índice e os dados

```
[58] for i, linha in df.iterrows():  
      print('Index: ', i)  
      print('Linha:\n', linha)  
      print('-----')
```

```
Index: Matheus  
Linha:  
  idade          23  
cidade    Guaratinguetá  
estado      São Paulo  
país        Brasil  
Name: Matheus, dtype: object
```

```
-----  
Index: José  
Linha:  
  idade          18  
cidade    Guarulhos  
estado      São Paulo  
país        Brasil  
Name: José, dtype: object
```

GRATIDÃO!

