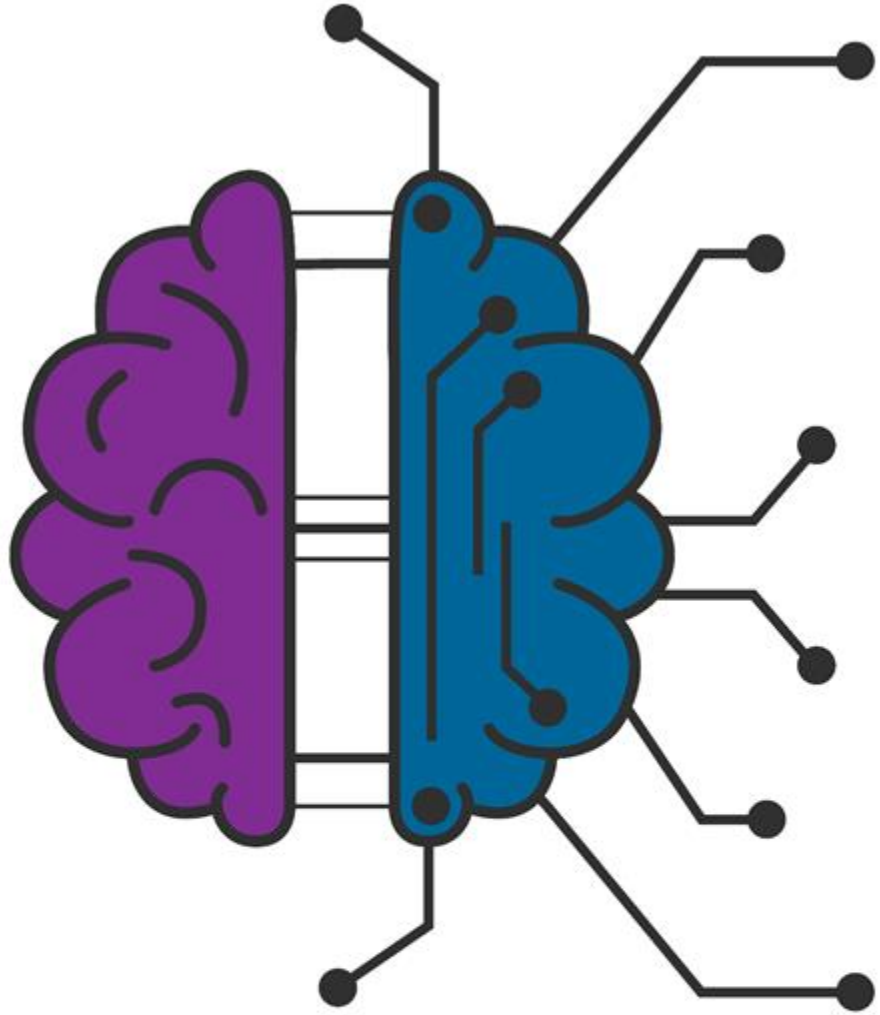


ARTIEEEFICIAIS

IEEE UFBA

WIE UFBA

TECH IEEE

Introdução à Machine Learning com Python

Dos primeiros passos até o desenvolvimento avançado em Machine Learning

Carine Gottschall

Lucas Alves

Conteúdo Programático

1. Gestão de pacotes e ambientes em Python
   I. Anaconda
   II. Jupyter Notebook
   III. Google Colab
2. Pacotes essenciais ao desenvolvimento de RNA com Python
   I. Numpy
   II. Pandas
   III. Tratamento de Dados
3. Machine Learning
   I. Regressão Linear
   II. Classificação
   III. Clustering (K-means)

Pacotes essenciais ao desenvolvimento de RNA com Python: Pandas

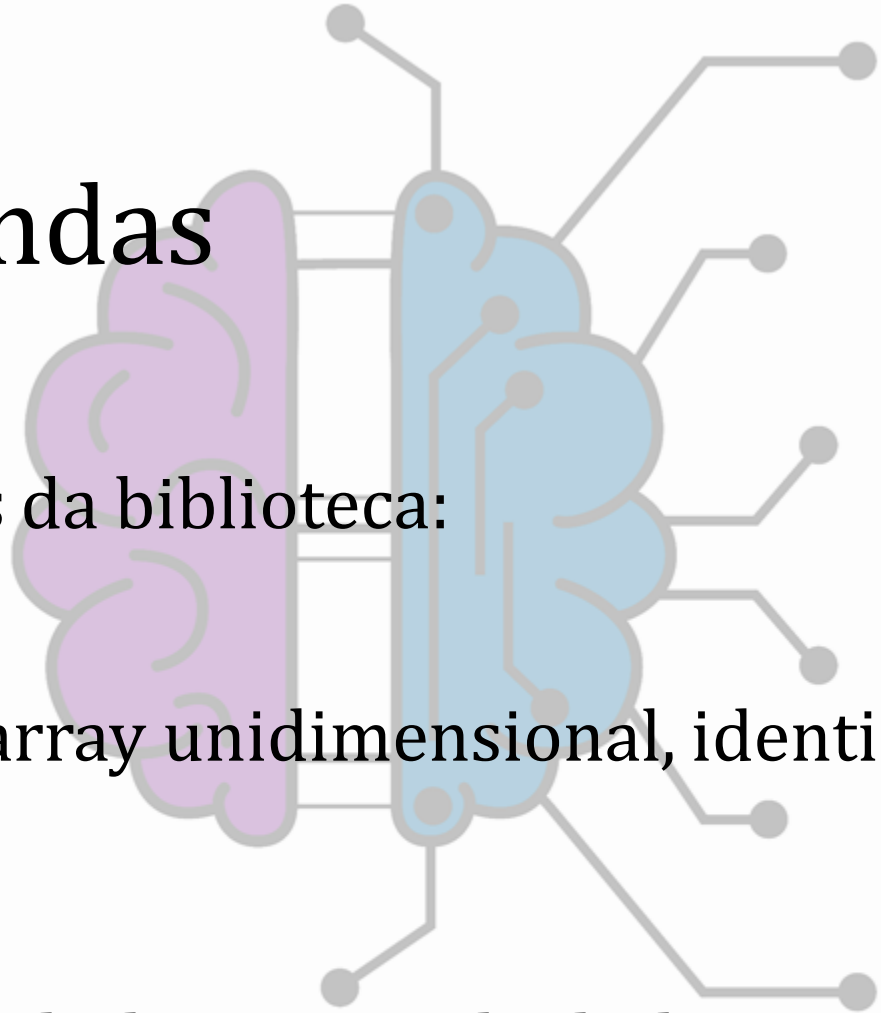
O que é o Pandas?

O Pandas é biblioteca que fornece estrutura de dados rápidas, flexíveis para facilitar o trabalho com dados “relacionais” e/ou rotulados. É a principal ferramenta para manipulação e análise de dados na computação científica em Python.

Porque utilizar Pandas?

- Mutabilidade de tamanho: colunas podem ser inseridas e excluídas do DataFrame.
- Funcionalidades flexíveis para agregação, combinação e transformação dos dados.
- Selecionamento de dados inteligente, com uma indexação sofisticada para utilização em grandes conjuntos de dados.
- Manuseio fácil de dados ausentes.

Biblioteca Pandas

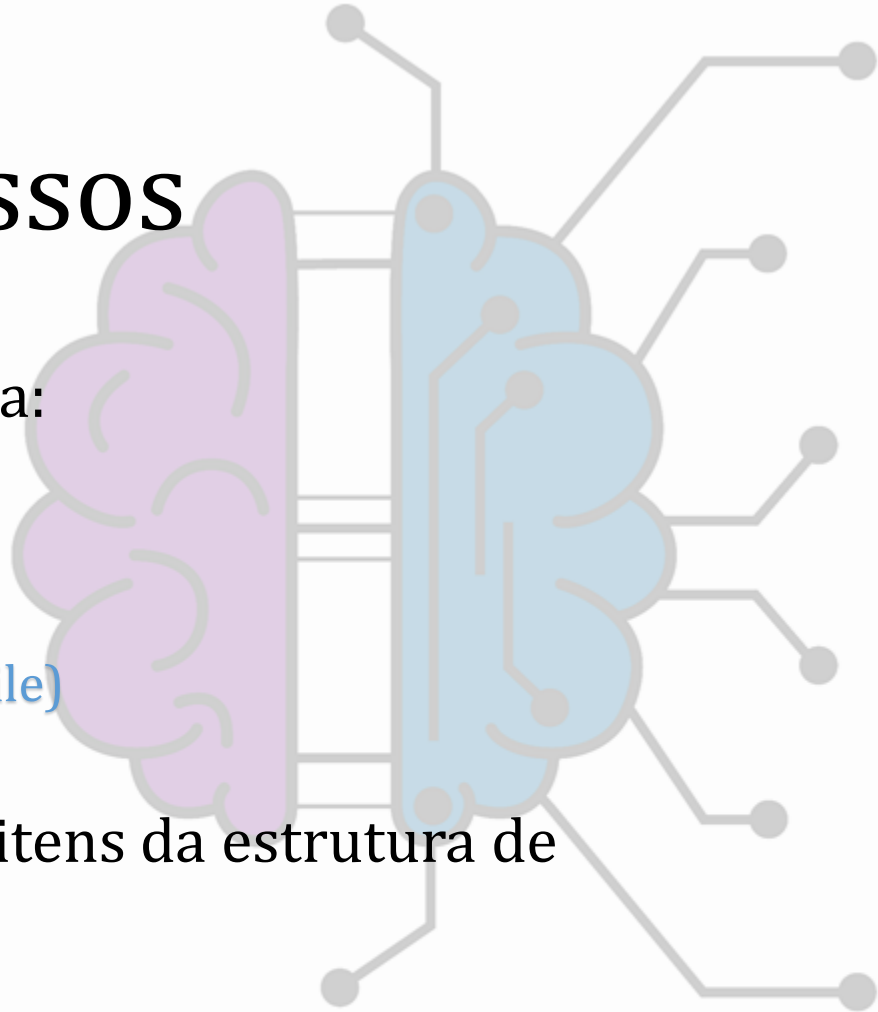


Principais Estruturas da biblioteca:

Series: Similar a um array unidimensional, identificação homogênea.

DataFrame: Estrutura bidimensional tabular potencialmente heterogênea.

Primeiros passos



- Importanto a biblioteca:

  ```
  import pandas as pd
  ```

- Leitura de arquivos:

  ```
  Data = pd.read_csv(pathfile)
  ```

- Verificar os primeiros itens da estrutura de dados

  ```
  Data.head(n_itens)
  ```

- Gerar um relatório estatístico da estrutura de dados

  ```
  Data.describe()
  ```

Criação - Series

- Criar uma serie

  Notas = pd.Series([3,8,7,5,10])

- Criar serie com rótulos:

  Notas = pd.Series([3,8,7,5,10],
  index=[“Lucas”,”Carine”,”Isaac”,”Man”,”???”])

- Acessando elemento

  Notas[0] # 3

  Notas[“Lucas”] # 3

- Verificar Atributos da serie:

  Notas.values

  Notas.index

```
Notas = pd.Series([3, 8, 7, 5, 10])
Notas
```

```
0     3
1     8
2     7
3     5
4    10
dtype: int64
```

```
Notas = pd.Series([3,8,7,5,10],
        index=["Lucas","Carine","Isaac","Man","???"])
Notas
```

```
Lucas      3
Carine     8
Isaac      7
Man        5
???       10
dtype: int64
```

```
Notas.values
```

```
array([ 3,  8,  7,  5, 10], dtype=int64)
```

```
Notas.index
```

```
Index(['Lucas', 'Carine', 'Isaac', 'Man', '???'], dtype='object')
```

Criação - DataFrame

- Criar um DataFrame

```
df = pd.DataFrame({'Aluno':
                   ["Lucas","Carine","Isaac","Man","???"],
                  'NotasNumpy': [3,8,7,5,10],
                  'NotasPandas': [8,9,10,9,8]})
```

- Acessando Coluna

```
df["NotasPandas"]
df.NotasPandas
```

- Ordenar tabela com base em coluna

```
df.sort_values(by="NotasPandas")
```

	Aluno	NotasNumpy	NotasPandas
0	Lucas	3	8
1	Carine	8	9
2	Isaac	7	10
3	Man	5	9
4	???	10	8

```
df.NotasPandas
```

```
0     8
1     9
2    10
3     9
4     8
Name: NotasPandas, dtype: int64
```

	Aluno	NotasNumpy	NotasPandas
0	Lucas	3	8
4	???	10	8
1	Carine	8	9
3	Man	5	9
2	Isaac	7	10

DataFrame - Indexação

	Aluno	NotasNumpy	NotasPandas
0	Lucas	3	8
1	Carine	8	9
2	Isaac	7	10
3	Man	5	9
4	???	10	8

- Localizar linha por índice

  df.iloc[2]

```
df.iloc[2]
```

```
Aluno          Isaac
NotasNumpy         7
NotasPandas       10
Name: 2, dtype: object
```

- Localizar linha por rótulo ou função booleana

  df.loc[df.Alunos == 'Man']

```
df.loc[df.Aluno =='Man']
```

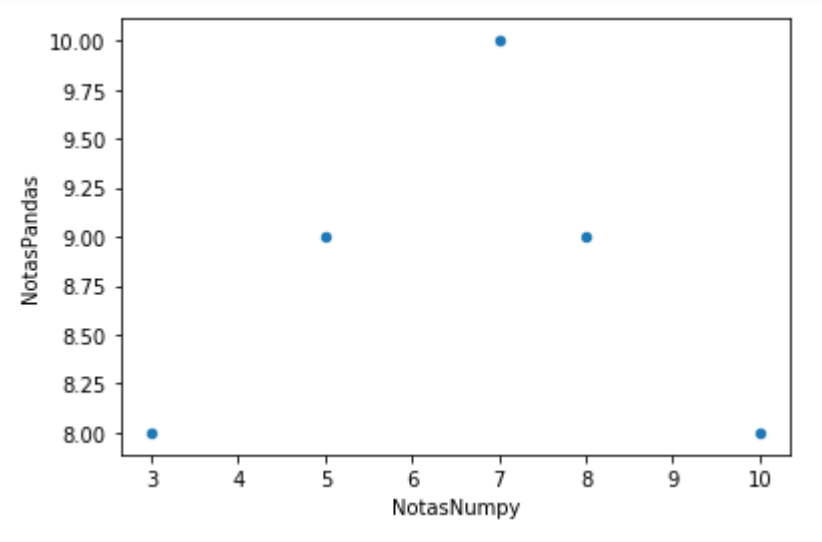
	Aluno	NotasNumpy	NotasPandas
3	Man	5	9

DataFrame - Visualização

	Aluno	NotasNumpy	NotasPandas
0	Lucas	3	8
1	Carine	8	9
2	Isaac	7	10
3	Man	5	9
4	???	10	8

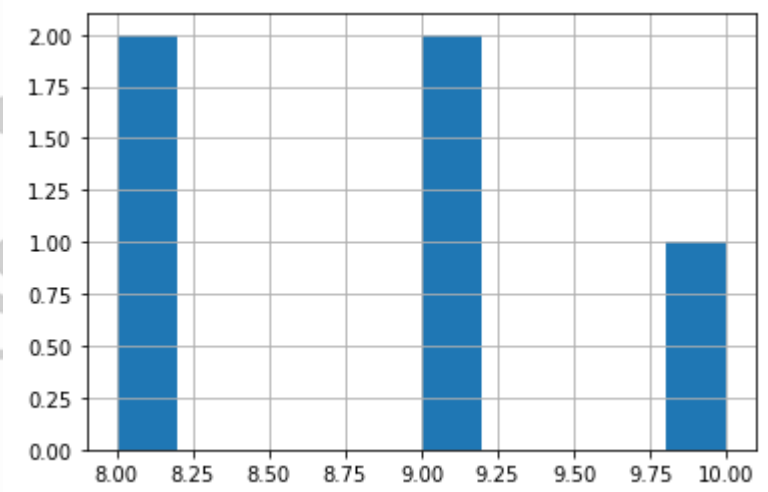
- Plotar gráfico Notas Numpy x Notas Pandas

  df.plot.scatter ('NotasNumpy','NotasPandas')



- Plotar histograma Notas Pandas

  df.NotasPandas.hist()




Hora da prática
ARTIEEEFICIAIS

Python For Data Science *Cheat Sheet*

Pandas Basics

Learn Python for Data Science Interactively at www.DataCamp.com

Pandas

The **Pandas** library is built on NumPy and provides easy-to-use **data structures** and **data analysis** tools for the Python programming language.

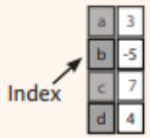
Use the following import convention:

```
>>> import pandas as pd
```

Pandas Data Structures

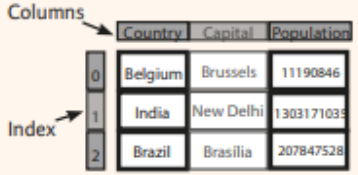
Series

A **one-dimensional** labeled array capable of holding any data type




```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

DataFrame



A **two-dimensional** labeled data structure with columns of potentially different types

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasilia'],
            'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

Asking For Help

```
>>> help(pd.Series.loc)
```

Selection

Also see NumPy Arrays

Getting

```
>>> s['b']
 -5
```
Get one element

```
>>> df[1:]
   Country     Capital  Population
1   India   New Delhi  1303171035
2  Brazil    Brasilia   207847528
```
Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0],[0]]
'Belgium'
```
```
>>> df.iat([0],[0])
'Belgium'
```
Select single value by row & column

By Label

```
>>> df.loc[[0], ['Country']]
'Belgium'
```
```
>>> df.at([0], ['Country'])
'Belgium'
```
Select single value by row & column labels

By Label/Position

```
>>> df.ix[2]
Country      Brazil
Capital    Brasilia
Population  207847528
```
Select single row of subset of rows

```
>>> df.ix[:,'Capital']
0     Brussels
1    New Delhi
2     Brasilia
```
Select a single column of subset of columns

```
>>> df.ix[1,'Capital']
'New Delhi'
```
Select rows and columns

Boolean Indexing

```
>>> s[~(s > 1)]
```
Series `s` where value is not >1

```
>>> s[(s < -1)  |  (s > 2)]
```
`s` where value is <-1 or >2

```
>>> df[df['Population']>1200000000]
```
Use filter to adjust DataFrame

Setting

```
>>> s['a'] = 6
```
Set index `a` of Series `s` to 6

Dropping

```
>>> s.drop(['a', 'c'])
```
Drop values from rows (axis=0)

```
>>> df.drop('Country', axis=1)
```
Drop values from columns(axis=1)

Sort & Rank

```
>>> df.sort_index()
```
Sort by labels along an axis

```
>>> df.sort_values(by='Country')
```
Sort by the values along an axis

```
>>> df.rank()
```
Assign ranks to entries

Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape
```
(rows,columns)

```
>>> df.index
```
Describe index

```
>>> df.columns
```
Describe DataFrame columns

```
>>> df.info()
```
Info on DataFrame

```
>>> df.count()
```
Number of non-NA values

Summary

```
>>> df.sum()
```
Sum of values

```
>>> df.cumsum()
```
Cummulative sum of values

```
>>> df.min()/df.max()
```
Minimum/maximum values

```
>>> df.idxmin()/df.idxmax()
```
Minimum/Maximum index value

```
>>> df.describe()
```
Summary statistics

```
>>> df.mean()
```
Mean of values

```
>>> df.median()
```
Median of values

Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f)
```
Apply function

```
>>> df.applymap(f)
```
Apply function element-wise

Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
a    10.0
b     NaN
c     5.0
d     7.0
```

Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
a    10.0
b    -5.0
c     5.0
d     7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> df.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

Read multiple sheets from the same file

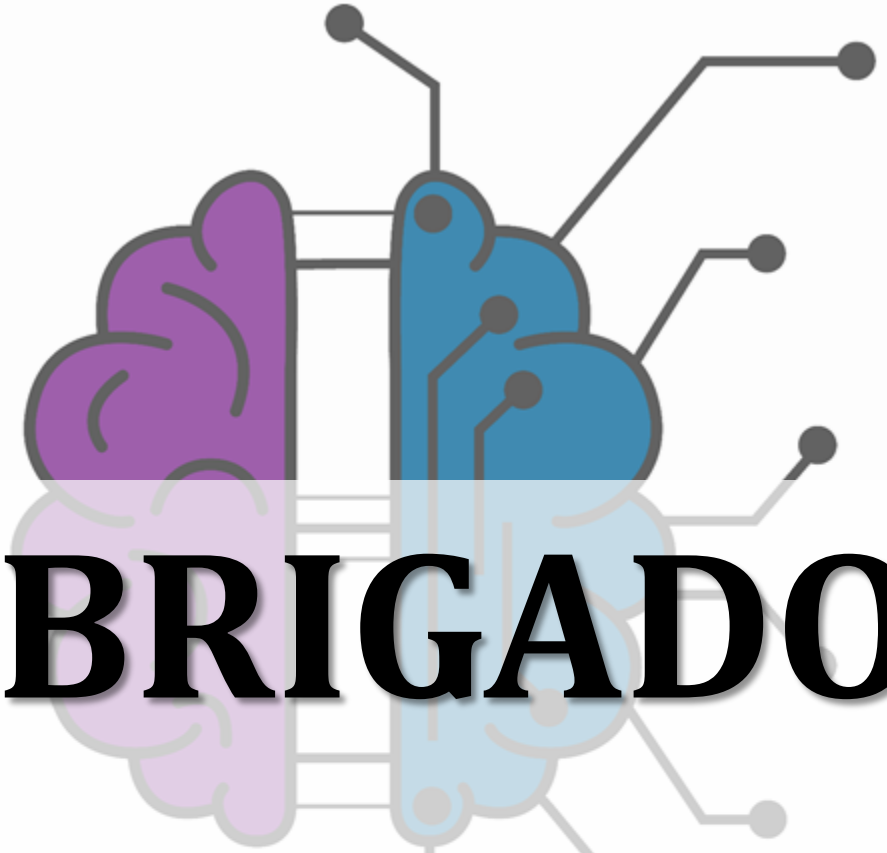
```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

`read_sql()` is a convenience wrapper around `read_sql_table()` and `read_sql_query()`

```
>>> df.to_sql('myDf', engine)
```



OBRIGADO!

Repositório GitHub:
https://github.com/Skyzenho/ArtIEEEficiais

ARTIEEEFICIAIS