Módulo de Programação Python

Trilha Python - Aula 14: Utilizando Pandas - Introdução

Objetivo: Trabalhar com pacotes e módulos disponíveis em **Python**: **Pandas**: Discutir a importância de obter, carregar e organizar grandes volumes de dados. Apresentar Pandas e suas funcionalidades e características básicas.

Contextualização

Até aqui discutimos sobre a importância de trabalhar com estruturas de dados eficientes para armazenar grandes volumes de dados. Nas aulas anteriores foram apresentados os arrays de tipo fixo implementados na forma de *ndarrays* da **NumPy**.

Ainda que muito eficientes para armazenar e processar dados numéricos, os *ndarrays* apresentam limitações para análise da dados não numéricos.

Imaginem, no exemplo que construímos na aula anterior, que queremos adicionar uma etiqueta ou rótulo a cada aluno com o nome ou o e-mail.

O Pandas, e em particular seus objetos `Series` e `DataFrame` , baseia-se no uso de *ndarrays* de **NumPy** e fornece acesso eficiente a esses tipos de tarefas de "gestão de dados" que ocupam muito do tempo de um cientista de dados.

Vamos abordar então em como utilizar `Series` , `DataFrame` e estruturas relacionadas de forma eficaz.

No ambiente virtual que utilizamos até qui temos os pacotes e módulos para rodar o *jupyter notebook* e **NumPY**. Vamos começar então por instalar **Pandas**

In [1]:
```
#pip list
#pip freeze > requirements.txt
#cat requirements.txt
#pip install pandas
```

In [2]:
```
import numpy as np
import pandas as pd
print("Numpy version: ", np.__version__)
print("Pandas bersion: ", pd.__version__)
```

```
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions
4.2 (Intel(R) SSE4.2) enabled only processors has been deprecated.
Intel oneAPI Math Kernel Library 2025.0 will require Intel(R) Adva
nced Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions
4.2 (Intel(R) SSE4.2) enabled only processors has been deprecated.
Intel oneAPI Math Kernel Library 2025.0 will require Intel(R) Adva
nced Vector Extensions (Intel(R) AVX) instructions.
Numpy version:  1.26.2
Pandas bersion:  2.1.4
```

Os objetos Pandas podem ser considerados versões aprimoradas de matrizes *ndarrays* de **NumPy** nas quais as linhas e colunas são identificadas com rótulos em vez de simples índices inteiros.

Da mesma forma que **NumPy**, _**Pandas** fornece, além das estruturas de dados, uma série de ferramentas, métodos e funcionalidades úteis .

Vamos começar aprestando as estruturas básicas de **Pandas**.

In [3]:
```
from random import uniform
lista = [uniform(4, 10) for _ in range(5)]
for val in lista:
    print(f"{val:.2f}", end=" ")
```

```
9.67 9.53 7.12 7.85 5.02
```

Pandas Series

Uma `Series` **Pandas** é uma matriz unidimensional de dados indexados.

De forma simples um objeto da classe `Series` pode ser criado a partir de uma lista ou de um *ndarray*.

In [4]:
```
dSerie = pd.Series(lista)
dSerie
```

Out[4]:
```
0    9.669556
1    9.528550
2    7.120373
3    7.854567
4    5.024415
dtype: float64
```

In [5]:
```
npArray = np.array(lista)
dSerie = pd.Series(npArray)
dSerie
```

Out[5]:
```
0    9.669556
1    9.528550
2    7.120373
3    7.854567
4    5.024415
dtype: float64
```

Reparem que um objeto `Series` consiste em uma sequência de valores e sua correspondente sequência de índices, que podemos acessar com os atributos `values` e `index` .

In [6]:
```
print(dSerie.values)
print(type(dSerie.values))
```

```
[9.66955565 9.52855031 7.12037284 7.85456688 5.02441452]
<class 'numpy.ndarray'>
```

Já o tributo `index` é um objeto semelhante a um *ndarray*, de tipo `pd.Index` .

In [7]:
```
print(dSerie.index)
print(type(dSerie.index))
```

```
RangeIndex(start=0, stop=5, step=1)
<class 'pandas.core.indexes.range.RangeIndex'>
```

Os elementos de `dSerie` podem ser acessado via indexação.

In [8]:
```
dSerie[0]
```

Out[8]:
```
9.669555648133784
```

In [9]:
```
dSerie[1:3]
```

Out[9]:
```
1    9.528550
2    7.120373
dtype: float64
```

Pode parecer que um objeto da classe `Sreies` é semelhante a um *ndarrays*, podendo usar um o outro. Mas ...

In [10]:
```
print(dSerie.values[-1])
try:
    print(dSerie[-1])
except Exception as e:
    print(e)
```

```
5.024414521667219
-1
```

Entretanto, enquanto o *ndarray* de **NumPy** possui um índice inteiro, definido implicitamente, usado para acessar os valores, os objetos `Series` de **Pandas** possuem um índice definido explicitamente, associado ao conjunto de valores.

Essa definição explícita de índice fornece recursos adicionais como, por exemplo, o fato de que o índice não precisa ser um número inteiro, mas pode consistir em valores de qualquer tipo desejado.

Por exemplo, se desejarmos, podemos usar strings como índice:

In [11]:
```
dSerie = pd.Series(lista, index=['alpha', 'beta', 'gamma', 'delta',
dSerie
```

Out[11]:
```
alpha      9.669556
beta       9.528550
gamma      7.120373
delta      7.854567
epsilon    5.024415
dtype: float64
```

In [12]:
```
print(dSerie.values[-1])
print(dSerie['epsilon'])
```

```
5.024414521667219
5.024414521667219
```

Podemos inclusive usar índices inteiros não contíguos ou não sequenciais.

In [13]:
```
dSerie = pd.Series(lista, index=[99, 87, 65, 43, 21])
dSerie
```

Out[13]:
```
99    9.669556
87    9.528550
65    7.120373
43    7.854567
21    5.024415
dtype: float64
```

In [14]:
```
print(dSerie.values[-1])
print(dSerie[21])
```

```
5.024414521667219
5.024414521667219
```

In [15]:
```
dSerie = pd.Series(lista, index=[0.1, 0.01, 0.001, 0.0001, 0.00001]
dSerie
```

Out[15]:
```
0.10000    9.669556
0.01000    9.528550
0.00100    7.120373
0.00010    7.854567
0.00001    5.024415
dtype: float64
```

Podemos então pensar as `Sreies` **Pandas** como uma forma particular e específica de dicionário **Python**.

- Um dicionário **Python** é uma estrutura que mapeia chaves arbitrárias para um conjunto de valores arbitrários
- Um objeto `Series` é uma estrutura que mapeia chaves de tipo fixo para um conjunto de valores de tipo fixo.

O fato de tratar de chaves e valores tipados é importante: assim como o código compilado, específico de cada tipo, por trás de um *ndarray* **NumPy**, o torna mais eficiente do que uma lista **Python** para determinadas operações, as informações de tipo de um `Series` **Pandas** o tornam muito mais eficiente do que os dicionários **Python** para determinadas operações.

A analogia da série como dicionário pode ficar ainda mais evidente quando constatamos que podemos construir um objeto `Series` diretamente de um dicionário **Python**:

In [16]:
```
popPorEstadoDic = { 'São Paulo': 44411238, 'Minas Gerais':20538718,
                'Bahia':14141626, 'Paraná':11444380, 'Rio Grande do
                'Pernambuco':9058931, 'Ceará':8794957}
popPorEstadoSer = pd.Series(popPorEstadoDic)
popPorEstadoSer
```

Out[16]:
```
São Paulo           44411238
Minas Gerais        20538718
Rio de Janeiro      16054524
Bahia               14141626
Paraná              11444380
Rio Grande do Sul   10882965
Pernambuco           9058931
Ceará                8794957
dtype: int64
```

Nesta construção é criada um objeto `Series` onde o índice é extraído das chaves do dicionário.

Uma vez criado o acesso aos elementos repete a sintaxes típica dos dicionário **Python**.

In [17]:
```
print("A população da Bahia, segundo o IBGE, é de", popPorEstadoSer
```

```
A população da Bahia, segundo o IBGE, é de 14141626 habitantes
```

Por outro lado, diferente de um dicionário, `Series` também suportam operações no estilo array, como *slicing*.

In [18]:
```
print(popPorEstadoSer['Bahia':'Pernambuco'])
```

```
Bahia                14141626
Paraná               11444380
Rio Grande do Sul    10882965
Pernambuco            9058931
dtype: int64
```

Vamos explorar então as formas de ciar `Series` . Já vimos que podemos passar um conjunto de dados, na forma de uma lista ou de um *ndarray*. Neste caso os índices são criados como de forma sequencial como inteiros que correspondem aos índices do *ndarray*.

In [19]:
```
títulos = [12, 8, 8, 7, 6, 4, 4, 4, 3, 3, 2, 2, 2, 1, 1, 1, 1]
brasileirão = pd.Series(títulos)
brasileirão
```

Out[19]:
```
0     12
1      8
2      8
3      7
4      6
5      4
6      4
7      4
8      3
9      3
10     2
11     2
12     2
13     1
14     1
15     1
16     1
dtype: int64
```

Mas podemos acrescentar os índices na forma de uma lista.

In [21]:
```
ras', 'Santos', 'Flamengo', 'Corinthians', 'São Paulo',
ro', 'Fluminense', 'Vasco', 'Internacional', 'Atlético-MG',
', 'Botafogo', 'Grêmio', 'Athletico-PR', 'Coritiba', 'Guarani', 'Spo
```

Quando a lista de índices é fornecida o valor pode ser apenas um escalar. Neste caso o valor é repetido para cada índice.

In [22]:
```
brasileirão = pd.Series(1, index=campBrasileiros)
brasileirão
```

Out[22]:
```
Palmeiras        1
Santos           1
Flamengo         1
Corinthians      1
São Paulo        1
Cruzeiro         1
Fluminense       1
Vasco            1
Internacional    1
Atlético-MG      1
Bahia            1
Botafogo         1
Grêmio           1
Athletico-PR     1
Coritiba         1
Guarani          1
Sport            1
dtype: int64
```

Como já vimos, os dados podem ser fornecidos na forma de um dicionário **Python**.

In [23]:
```
dicBrasileirao = {2:'Santos', 4:'Corinthians', 3:'Flamengo',  1:'Pa
brasileirão = pd.Series(dicBrasileirao)
print(brasileirão)
print(type(brasileirão.values))
```

```
2         Santos
4    Corinthians
3       Flamengo
1      Palmeiras
dtype: object
<class 'numpy.ndarray'>
```

Mesmo quando fornecido um dicionário podemos escolher apenas um subconjunto dos elementos especificando uma lista de iíndices.

In [24]:
```
brasileirão = pd.Series(dicBrasileirao, index=[2,3,4])
print(brasileirão)
```

```
2         Santos
3       Flamengo
4    Corinthians
dtype: object
```

In [26]:
```
'''
Palmeiras   12  1960, 1967, 1967, 1969, 1972, 1973, 1993, 1994, 2016
Santos  8   1961, 1962, 1963, 1964, 1965, 1968, 2002 e 2004
Flamengo    8   1980, 1982, 1983, 1987, 1992, 2009, 2019 e 2020
Corinthians 7   1990, 1998, 1999, 2005, 2011, 2015 e 2017
São Paulo   6   1977, 1986, 1991, 2006, 2007 e 2008
Cruzeiro    4   1993, 1996, 2000, 2003, 2017 e 2018
Fluminense  4   1970, 1984, 2010 e 2012
Vasco   4   1974, 1989, 1997 e 2000
Internacional   3   1975, 1976 e 1979
Atlético-MG 3   1937, 1971 e 2021
Bahia   2   1959 e 1988
Botafogo    2   1968 e 1995
Grêmio  2   1981 e 1996
Athletico-PR    1   2001
Coritiba    1   1985
Guarani 1   1978
Sport   1   1987
'''
```

Out[26]:
```
'\nPalmeiras\t12\t1960, 1967, 1967, 1969, 1972, 1973, 1993, 1994,
2016, 2018, 2022 e 2023\nSantos\t8\t1961, 1962, 1963, 1964, 1965,
1968, 2002 e 2004\nFlamengo\t8\t1980, 1982, 1983, 1987, 1992, 200
9, 2019 e 2020\nCorinthians\t7\t1990, 1998, 1999, 2005, 2011, 2015
e 2017\nSão Paulo\t6\t1977, 1986, 1991, 2006, 2007 e 2008\nCruzeir
o\t4\t1993, 1996, 2000, 2003, 2017 e 2018\nFluminense\t4\t1970, 19
84, 2010 e 2012\nVasco\t4\t1974, 1989, 1997 e 2000\nInternacional\
t3\t1975, 1976 e 1979\nAtlético-MG\t3\t1937, 1971 e 2021\nBahia\t2
\t1959 e 1988\nBotafogo\t2\t1968 e 1995\nGrêmio\t2\t1981 e 1996\nA
thletico-PR\t1\t2001\nCoritiba\t1\t1985\nGuarani\t1\t1978\nSport\t
1\t1987\n'
```

Pandas `DataFrame`

Da mesma forma que os objetos `Series`, o `DataFrame` pode ser pensado como uma generalização de um *ndarray* **NumPy** ou como uma especialização de um dicionário **Python**.

Se um objeto `Series` é análogo a uma matriz unidimensional com índices flexíveis, um `DataFrame` pode ser visto como uma estrutura análoga a uma matriz bidimensional com índices de linha e nomes de colunas flexíveis. Você pode pensar em um `DataFrame` como uma sequência de objetos `Series` alinhados. Aqui, por "alinhado" queremos dizer que eles compartilham o mesmo índice.

In [27]:
```
torcidas= {'Flamengo':46953599, 'Corinthians':30444799, 'Palmeiras'
torcidaSer = pd.Series(torcidas)
torcidaSer
```

Out[27]:
```
Flamengo       46953599
Corinthians    30444799
Palmeiras      20225600
Santos          6646400
dtype: int64
```

In [28]:
```
dicBrasileirao = {'Santos':8, 'Corinthians':7, 'Flamengo':8,  'Palm
brasileirão = pd.Series(dicBrasileirao)
brasileirão
```

Out[28]:
```
Santos          8
Corinthians     7
Flamengo        8
Palmeiras      12
dtype: int64
```

In [29]:
```
times = pd.DataFrame({'Títulos':brasileirão, 'Torcida':torcidaSer})
times
```

Out[29]:

	Títulos	Torcida
Corinthians	7	30444799
Flamengo	8	46953599
Palmeiras	12	20225600
Santos	8	6646400

Assim como o objeto `Series` , o `DataFrame` possui um atributo de índice que dá acesso aos rótulos das linhas.

In [30]:
```
times.index
```

Out[30]:
```
Index(['Corinthians', 'Flamengo', 'Palmeiras', 'Santos'], dtype='o
bject')
```

Além disso, o `DataFrame` possui um atributo `columns` , que é um objeto Index que contém os rótulos das colunas.

In [31]:
```
times.columns
```

Out[31]:
```
Index(['Títulos', 'Torcida'], dtype='object')
```

Desta forma, o `DataFrame` pode ser pensado como uma generalização de um *ndarray* **NumPy** bidimensional, onde tanto as linhas quanto as colunas possuem um índice generalizado para acessar os dados.

O primeiro exemplo mostrou como criar um `DataFrame` a partir de dois `Series`. Também podemos construir um `DataFrame` a partir de uma lista de dicionários.

In [32]:
```
dicBrasileirao = {'Santos':8, 'Corinthians':7, 'Flamengo':8,  'Palm
torcidas= {'Flamengo':46953599, 'Corinthians':30444799, 'Palmeiras'

times = pd.DataFrame([torcidas, dicBrasileirao], index=['Torcida',
times
```

Out[32]:

	Flamengo	Corinthians	Palmeiras	Santos
Torcida	46953599	30444799	20225600	6646400
Títulos	8	7	12	8

No caso de faltarem algumas chaves nos dicionários, o **Pandas** irá preenchê-las com valores **NaN**.

In [33]:
```
pBrasileiros = ['Palmeiras', 'Santos', 'Flamengo', 'Corinthians', 'S
                'Cruzeiro', 'Fluminense', 'Vasco da Gama', 'Internac
                 'Bahia', 'Botafogo', 'Grêmio', 'Athletico-PR', 'Cor
ulos = [12, 8, 8, 7, 6, 4, 4, 4, 3, 3, 2, 2, 2, 1, 1, 1, 1]

 time, titulo in zip(campBrasileiros, títulos):
 dicBrasileirao[time] = titulo

es = ['Flamengo', 'Corinthians', 'São Paulo', 'Palmeiras', 'Vasco da
      'Grêmio', 'Atlético-MG', 'Bahia', 'Internacional', 'Fluminense

cida = [46953599, 30444799, 22225800, 20225600, 13292800, 13078400,
        7718400, 7504000, 7289600, 6646400, 4288000, 4073600]

 time, torc in zip(times, torcida):
 torcidas[time] = torc

es = pd.DataFrame([torcidas, dicBrasileirao], index=['Torcida', 'Tít
es
```

Out[33]:

	Flamengo	Corinthians	Palmeiras	Santos	São Paulo	Vasco da Gama	Cruzeiro	Grêm
Torcida	46953599	30444799	20225600	6646400	22225800	13292800	13078400	98624(
Títulos	8	7	12	8	6	4	4	

Podemos utilizar também dicionários de `Series` .

In [34]:
```
campBrasileiros = ['Palmeiras', 'Santos', 'Flamengo', 'Corinthians'
                   'Cruzeiro', 'Fluminense', 'Vasco da Gama', 'Inte
                    'Bahia', 'Botafogo', 'Grêmio', 'Athletico-PR',
títulos = [12, 8, 8, 7, 6, 4, 4, 4, 3, 3, 2, 2, 2, 1, 1, 1, 1]

for time, titulo in zip(campBrasileiros, títulos):
    dicBrasileirao[time] = titulo

timTitulos = pd.Series(dicBrasileirao)

times = ['Flamengo', 'Corinthians', 'São Paulo', 'Palmeiras', 'Vasc
         'Grêmio', 'Atlético-MG', 'Bahia', 'Internacional', 'Flumin

torcida = [46953599, 30444799, 22225800, 20225600, 13292800, 130784
           7718400, 7504000, 7289600, 6646400, 4288000, 4073600]

for time, torc in zip(times, torcida):
    torcidas[time] = torc

timTorcida = pd.Series(torcidas)

times = pd.DataFrame({'Títulos':timTitulos, 'Torcida':timTorcida})
times
```

Out[34]:

	Títulos	Torcida
Athletico-PR	1	NaN
Atlético-MG	3	9219199.0
Bahia	2	7718400.0
Botafogo	2	4288000.0
Corinthians	7	30444799.0
Coritiba	1	NaN
Cruzeiro	4	13078400.0
Flamengo	8	46953599.0
Fluminense	4	7289600.0
Grêmio	2	9862400.0
Guarani	1	NaN
Internacional	3	7504000.0
Palmeiras	12	20225600.0
Santos	8	6646400.0
Sport	1	4073600.0

São Paulo	6	22225800.0
Vasco da Gama	4	13292800.0

E, pensando num `DataFrame` como um generalização de um *ndarray* **NumPy** bidimensional, podemos construir um `DataFrame` usando uma *ndarray*.

In [35]:
```
títulos = [12, 8, 8, 7]
torcida = [20225600, 6646400, 46953599, 30444799]
tittor = np.array([títulos, torcida])
print(tittor.T)
tittor = pd.DataFrame(tittor.T, columns=['Títulos', 'Torcida'],
                      index=['Palmeiras', 'Santos', 'Flamengo', 'Cori

print(tittor)
```

```
[[      12 20225600]
 [       8  6646400]
 [       8 46953599]
 [       7 30444799]]
             Títulos   Torcida
Palmeiras         12  20225600
Santos             8   6646400
Flamengo           8  46953599
Corinthians        7  30444799
```

In [ ]:

In [ ]:

In [ ]: