Python Fundamentos – Biblioteca Pandas e NumPy – Parte 01

Disciplina: Estatística Aplicada à Engenharia de Software

Prof. Me. Max Gabriel Steiner

Introdução à Biblioteca Pandas e NumPy

- **Biblioteca Pandas:** pode ser usado para ler e manipular dados de arquivos CSV em Python usando a função pd.read_csv() para ler o arquivo e criar um DataFrame para manipulação subsequente.
- As Estruturas de dados no Pandas:
- Series: Uma série unidimensional que pode conter qualquer tipo de dado.
- DataFrame: Uma estrutura de dados bidimensional semelhante a uma tabela ou planilha.
- Index: Uma estrutura que fornece rótulos para as linhas e colunas em um DataFrame.

Introdução à Biblioteca Pandas e NumPy

- **Biblioteca NumPy:** o objetivo principal da biblioteca NumPy em Python é fornecer suporte para arrays multidimensionais e funções matemáticas de alto desempenho para manipulação de dados numéricos.
- Exemplo de criação de uma matriz NumPy e operação:

```
import numpy as np

# Criando uma matriz NumPy

matriz = np.array([[1, 2, 3], [4, 5, 6]])

# Realizando uma operação (por exemplo, multiplicação por 2)

resultado = matriz * 2

print(matriz)

print(resultado)
```

Biblioteca Pandas

- Para aplicar com êxito a biblioteca Pandas é legal que você conheça bem duas estruturas:
- **Series**;
- **DataFrame**;

- É importante lembrar que a plataforma Pandas, assim como o Numpy, não é um módulo “built-in”, ou seja, ele não faz parte da instalação padrão do interpretador Python.
- Por isso, precisamos instalar ele, ou utilizar o Anaconda que neste caso ele já vem instalado automaticamente.

- Para importar o pandas podemos utilizar o formato:
- import pandas as pd
- Podemos utilizar junto também:
- from pandas import Series
- ou então:
- from pandas import DataFrame

- **Series**
- Series é um array (arranjo, estrutura de dados) unidimensional que contém um array de dados e um array de labels (índice).

- Vamos aplicar um **exemplo 1:**
- from pandas import Series
- import pandas as pd
- Obj = Series([4, 7, -5, 3])
- print(Obj)

```
In [4]: from pandas import Series

In [5]: import pandas as pd

In [6]: Obj = Series([4, 7, -5, 3])

In [7]: Obj

Out[7]: 0    4
        1    7
        2   -5
        3    3
        dtype: int64
```

- Aplicando:
- type(Obj)

```
Out[7]: 0    4
        1    7
        2   -5
        3    3
        dtype: int64

In [8]: type(Obj)

Out[8]: pandas.core.series.Series
```

- Com essa aplicação podemos perceber que a variável Obj é um objeto do tipo Series do Pandas

- Aplicando:
- Obj.values

```
In [9]: Obj.values
Out[9]: array([ 4,  7, -5,  3], dtype=int64)
```

- Com essa aplicação o programa nos retorna os valores dessa série de dados, que são os parâmetros inseridos por nós.

- Aplicando:
- Obj.index

```
In [10]: Obj.index
Out[10]: RangeIndex(start=0, stop=4, step=1)
```

- Com essa aplicação podemos ver os índices, os quais são os parâmetros dessa série de dados.
- A saída nos permite avaliar onde os dados: começam, terminam e vão de 1 em 1.

➢ Vamos aplicar um **exemplo 2:** Criando uma série e especificando os índices

```
In [19]: # Criando uma série e especificando os índices:
         Obj2 = Series([4, 7, -5, 3], index = ['a', 'b', 'c', 'd'])
```

➢ Vamos imprimir Obj2

```
In [20]: Obj2

Out[20]: a    4
         b    7
         c   -5
         d    3
         dtype: int64
```

- Agora vamos imprimir os valores e vamos imprimir o índice:

```
In [21]: Obj2.values
Out[21]: array([ 4,  7, -5,  3], dtype=int64)

In [22]: Obj2.index
Out[22]: Index(['a', 'b', 'c', 'd'], dtype='object')
```

- Analisando a saída podemos perceber que temos uma série de números e uma série de índices.

- Vamos agora aplicar algumas alternativas para manipular séries de dados.
- **Exemplo 03:**

```
In [23]: Obj2[Obj2 > 3]
```

- Com essa condição vamos selecionar apenas os elementos maiores que 3 dentro da nossa série, portanto:

```
In [23]: Obj2[Obj2 > 3]

Out[23]: a    4
         b    7
         dtype: int64
```

- **Exemplo 04:**
- Com essa condição vamos trazer apenas o elemento contido em uma posição específica dentro do índice:

```
In [24]: Obj2['b']

Out[24]: 7
```

- Podemos ainda, perguntar ao Python se: ‘d’ existe dentro do Obj2?

```
In [25]: 'd' in Obj2

Out[25]: True
```

- E neste caso ele vai retornar que sim (true), pois dentro sim ‘d’ é um elemento do meu índice.

- Podemos também criar uma série de dados a partir de um dicionário em Python.

```
In [26]: # Criando uma série a partir de um dicionário em Python:
         dict = {'Futebol':5200, 'Tenis':120, 'Natação':698, 'Volleyball':1550}
```

- Vamos agora criar mais uma série de dados, utilizando o método Series:

```
In [27]: # Criando uma série a partir de um dicionário:
         Obj3 = Series(dict)
```

- Vamos então imprimir nossa série de dados Obj3:

```
In [28]: Obj3

Out[28]: Futebol      5200
         Tenis         120
         Natação       698
         Volleyball   1550
         dtype: int64
```

- Podemos também criar uma lista:

```
In [29]: # Criando uma lista
         esportes = ['Futebol', 'Tenis', 'Natação', 'Basketball']
```

- Vamos agora criar mais uma série de dados, utilizando o método Series:

```
In [30]: # Criando uma série e usando uma lista como índice:
         Obj4 = Series(dict, index=esportes)
```

- Vamos então imprimir nossa série de dados Obj4:

```
In [31]: Obj4

Out[31]: Futebol       5200.0
         Tenis          120.0
         Natação        698.0
         Basketball       NaN
         dtype: float64
```

- Agora temos que o meu índice de fato é na verdade a lista de esportes.

- Para perceber mais de perto a diferença obtida, vamos comparar os Obj3 e Obj4:
- A lista utilizada no Obj4 possui Basketball no lugar de Volleyball do Obj3

```
In [29]: # Criando uma Lista
         esportes = ['Futebol', 'Tenis', 'Natação', 'Basketball']
```

```
In [26]: # Criando uma série a partir de um dicionário em Python:
         dict = {'Futebol':5200, 'Tenis':120, 'Natação':698, 'Volleyball':1550}
```

- Portanto, percebemos que o Obj4 colocou Basketball como NaN (vazio), pois não encontrou esse termo da lista contido no dicionário:

```
In [28]: Obj3

Out[28]: Futebol       5200
         Tenis          120
         Natação        698
         Volleyball    1550
         dtype: int64
```

```
In [31]: Obj4

Out[31]: Futebol       5200.0
         Tenis          120.0
         Natação        698.0
         Basketball       NaN
         dtype: float64
```

- Podemos utilizar o comando:

```
In [32]: # Comando para verificar se há valores nulos dentro da série:
         pd.isnull(Obj4)
```

- Vamos obter a saída:

```
Out[32]: Futebol       False
         Tenis         False
         Natação       False
         Basketball     True
         dtype: bool
```

- Existe também o comando contrário:

```
In [33]: # Comando para verificar se NÃO há valores nulos dentro da série:
         pd.notnull(Obj4)
```

```
Out[33]: Futebol       True
         Tenis         True
         Natação       True
         Basketball   False
         dtype: bool
```

- Podemos também utilizar o comando concatenar série de dados (ou seja, uni-las):

```
In [34]: # Concatenando duas séries:
         Obj3 + Obj4

Out[34]: Basketball        NaN
         Futebol       10400.0
         Natação        1396.0
         Tenis           240.0
         Volleyball        NaN
         dtype: float64
```

- Vamos verificar que ele somou os valores que já existiam em comum em cada série.
- Por exemplo, futebol era igual a 5200 em cada série e agora passou a valer o dobro.

- Por fim podemos:
- **Dar nome ao nosso objeto:**

```
In [35]: # Podemos dar nome ao nosso objeto:
         Obj4.name = 'população'
```

- **Dar nome ao índice:**

```
In [38]: # Podemos também dar nome ao índice:
         Obj4.index.name = 'Esporte'
```

- Resultado: nosso índice ficou chamado de Esporte e a nossa série de dados de ficou chamada de população:

```
In [39]: Obj4

Out[39]: Esporte
         Futebol       5200.0
         Tenis          120.0
         Natação        698.0
         Basketball       NaN
         Name: população, dtype: float64
```

- **DataFrame**
- Dataframes representam uma estrutura tabular semelhante a estrutura de uma planilha do Excel, contendo uma coleção de colunas em que cada uma pode ser um diferente tipo de valor (número, string, etc...).

- Os Dataframes possuem index e linhas e esta estrutura é muito semelhante a um Dataframe em R.

- Os dados de um dataframe são armazenados em um ou mais blocos bidimensionais, ao invés de listas, dicionários ou alguma estrutura de array.

- Vamos aplicar um **exemplo 1:**

```
In [40]: from pandas import DataFrame
```

```
In [41]: data = {'Estado': ['Santa Catarina', 'Paraná', 'Goiás', 'Bahia', 'Minas Gerais'],
                 'Ano':[2002, 2003, 2004, 2005, 2006],
                 'População': [1.5, 1.7, 3.6, 2.4, 2.9]}
```

- Criamos um dicionário chamado “data” e dentro dele colocamos as chaves: Estado, Ano e População.
- Cada chave possui uma lista de dados atrelados à ela.
- Vamos utilizar a função DataFrame() com o dicionário “data” que criamos.
- E vamos atribuí-la à variável “frame”.

```
In [42]: frame = DataFrame(data)
```

- Imprimindo a variável ‘frame’ temos:

```
In [43]: frame
```

Out[43]:

	Estado	Ano	População
0	Santa Catarina	2002	1.5
1	Paraná	2003	1.7
2	Goiás	2004	3.6
3	Bahia	2005	2.4
4	Minas Gerais	2006	2.9

- Como podemos ver trata-se de uma estrutura muito parecida com as planilhas do excel.
- Podemos perceber que as chaves contidas no dicionário viraram título das colunas.
- E o índice foi preenchido automaticamente pelo Pandas.

- Verificando o tipo de variável criado:

```
In [44]: type(frame)

Out[44]: pandas.core.frame.DataFrame
```

- Podemos confirmar que trata-se de um DataFrame!

➢ **Exemplo 2:** podemos aplicar algumas mudanças no formato

```
In [45]: DataFrame(data, columns=['Ano', 'Estado', 'População'])
```

Out[45]:

	Ano	Estado	População
0	2002	Santa Catarina	1.5
1	2003	Paraná	1.7
2	2004	Goiás	3.6
3	2005	Bahia	2.4
4	2006	Minas Gerais	2.9

- Vamos criar um frame maior:

```
In [48]: frame2 = DataFrame(data, columns = ['Ano', 'Estado', 'População', 'Débito'],
                            index = ['um','dois', 'três', 'quatro', 'cinco'])
```

```
In [49]: # Imprimindo o DataFrame
         frame2
```

Out[49]:

	Ano	Estado	População	Débito
um	2002	Santa Catarina	1.5	NaN
dois	2003	Paraná	1.7	NaN
três	2004	Goiás	3.6	NaN
quatro	2005	Bahia	2.4	NaN
cinco	2006	Minas Gerais	2.9	NaN

- Podemos perceber que a última coluna “Débito” está vazia, pois não há valores dela no dicionário “Data”.

- É possível também imprimir apenas uma coluna:

```
In [50]: # Imprimindo apenas uma coluna do DataFrame
         frame2['Estado']

Out[50]: um          Santa Catarina
         dois               Paraná
         três                Goiás
         quatro              Bahia
         cinco        Minas Gerais
         Name: Estado, dtype: object
```

- Para resolver o problema do slide anterior, onde a coluna Débito ficou vazia, vamos utilizar agora mais uma biblioteca, a biblioteca NumPy.