

Introdução à Python

Altino Dantas

Objetivos

- Conhecer a biblioteca Pandas;
- Aprender recursos básicos da biblioteca.



O que é Pandas



A biblioteca **Pandas** é construída sobre a **NumPy** e **complementa-a** com recursos que são particularmente úteis ao manipular dados, como **indexação rotulada**, índices hierárquicos, **alinhamento de dados** para comparação e mesclagem de conjuntos de dados, manipulação de dados ausentes e muito mais.

Estruturas

- As duas principais estruturas de dados da Pandas são: **Series** e **DataFrame**.
- Embora eles não sejam uma solução universal para todos os problemas, eles fornecem uma base sólida e **fácil** de usar para a maioria dos casos.

```
from pandas import Series, DataFrame  
import pandas as pd
```



Series

- **Serie** é um objeto unidimensional do tipo *array* que contém uma sequência de dados (de qualquer tipo de dados **NumPy**);
- Além de um *array* associado de rótulos de dados, denominado seu índice;
- A **Serie** mais simples é formada apenas por um *array* de dados:

```
obj = Series([4, 7, -5, 3])
```

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

Como não foi especificado um índice para os dados, é atribuído um padrão consistindo dos inteiros 0 a N - 1 (onde N é o comprimento dos dados).



Series

- É fácil obter o *array* de dados bem como os respectivos índices:

```
obj.values
```

```
array([ 4,  7, -5,  3], dtype=int64)
```

```
obj.index
```

```
RangeIndex(start=0, stop=4, step=1)
```



Series

- Muitas vezes será desejável criar uma série com um índice identificando cada ponto de dados:

```
obj2 = Series(data=[4, 7, -5, 3], index=['d', 'b', 'a', 'c'])
```

```
obj2.values
```

```
array([ 4,  7, -5,  3], dtype=int64)
```

```
obj2.index
```

```
Index(['d', 'b', 'a', 'c'], dtype='object')
```



Series

- Assim como em um *array NumPy* regular, você pode usar valores como índice ao selecionar valores únicos ou um conjunto de valores:

```
obj2['a']
```

```
-5
```

```
obj2['d'] = 6  
obj2.values
```

```
array([ 6,  7, -5,  3], dtype=int64)
```

```
obj2[['c', 'a', 'd']]
```

```
c    3
```

```
a   -5
```

```
d    6
```

```
dtype: int64
```



Series

- As operações de *array* **NumPy**, como filtrar com um *array* booleano, multiplicação escalar ou aplicar funções matemáticas, preservarão o link entre valor e índice:

```
In [15]: obj2[obj2 > 0]
```

```
Out[15]:
```

```
d    6
```

```
b    7
```

```
c    3
```

```
In [16]: obj2 * 2
```

```
Out[16]:
```

```
d    12
```

```
b    14
```

```
a   -10
```

```
c     6
```

```
In [17]: np.exp(obj2)
```

```
Out[17]:
```

```
d    403.428793
```

```
b   1096.633158
```

```
a     0.006738
```

```
c    20.085537
```



Series

- Outra maneira de pensar em uma série é como um **dict** ordenado de comprimento **fixo**, pois é um mapeamento de **valores de índice** para **valores de dados**.

```
In [18]: 'b' in obj2  
Out[18]: True
```

```
In [19]: 'e' in obj2  
Out[19]: False
```



Series

- Se você tiver dados contidos em um **dict** de *Python*, você pode criar uma série a partir dele passando o dict para `Series()`;

```
In [20]: sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}
```

```
In [21]: obj3 = Series(sdata)
```

```
In [22]: obj3
```

```
Out[22]:
```

```
Ohio      35000
```

```
Oregon    16000
```



Series

- Neste exemplo, mesmo os dados vindo de um dicionário, é possível especificar os índices;

```
In [27]: sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}  
obj3 = Series(sdata)
```

```
In [28]: states = ['California', 'Ohio', 'Oregon', 'Texas']  
obj4 = Series(sdata, index=states)
```

```
In [29]: obj3
```

```
Out[29]: Ohio      35000  
Texas      71000  
Oregon     16000  
Utah        5000  
dtype: int64
```

```
In [30]: obj4
```

```
Out[30]: California      NaN  
Ohio      35000.0  
Oregon     16000.0  
Texas      71000.0  
dtype: float64
```



Series

- 3 valores encontrados em **sdata** foram colocados nos locais apropriados, mas como nenhum valor para 'Califórnia' foi encontrado, aparece como NaN (Not a Number);
- As funções **isnull** e **notnull** no *pandas* devem ser usadas para detectar dados ausentes:

```
In [26]: pd.isnull(obj4)
```

```
Out[26]:
```

California	True
Ohio	False
Oregon	False
Texas	False

```
In [27]: pd.notnull(obj4)
```

```
Out[27]:
```

California	False
Ohio	True
Oregon	True
Texas	True

ou

```
In [28]: obj4.isnull()
```

```
Out[28]:
```

California	True
Ohio	False
Oregon	False
Texas	False

Series

- Um recurso muito útil é o alinhamento automático, de dados indexados de maneira diferente, em operações aritméticas:

```
In [29]: obj3
Out[29]:
```

Ohio	35000
Oregon	16000
Texas	71000
Utah	5000

```
In [30]: obj4
Out[30]:
```

California	NaN
Ohio	35000
Oregon	16000
Texas	71000

```
In [31]: obj3 + obj4
```

```
Out[31]:
```

California	NaN
Ohio	70000.0
Oregon	32000.0
Texas	142000.0
Utah	NaN

dtype: float64

Series

- Os índices de uma série podem ser alterados por atribuição:

```
In [32]: obj.index = ['Bob', 'Steve', 'Jeff', 'Ryan']
```

```
In [33]: obj
```

```
Out[33]: Bob      4  
         Steve    7  
         Jeff    -5  
         Ryan     3  
         dtype: int64
```



DataFrame

- Um DataFrame representa uma estrutura de dados **tabular**, semelhante a uma planilha;
- Contém uma coleção ordenada de colunas, cada uma das quais pode ser um tipo de dados diferente (numérico, string, booleano, etc.);
- DataFrame possui um índice de **linha** e **coluna**;
- Pode ser pensado como um **dict** de Serie (compartilhando o mesmo índice);



DataFrame

- Existem inúmeras maneiras de construir um DataFrame, embora um dos mais comuns seja de um **dict** de listas de tamanho igual ou de *arrays NumPy*:

```
In [34]: data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],  
                'year': [2000, 2001, 2002, 2001, 2002],  
                'pop': [1.5, 1.7, 3.6, 2.4, 2.9]  
            }  
  
         frame = DataFrame(data)
```

```
In [35]: frame
```



	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9



DataFrame

- Um índice é gerado automaticamente;
- A ordem das colunas pode ser definida:

```
In [36]: data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],  
                'year': [2000, 2001, 2002, 2001, 2002],  
                'pop': [1.5, 1.7, 3.6, 2.4, 2.9]  
            }  
  
frame = DataFrame(data, columns=['year', 'state', 'pop'])
```

```
In [37]: frame
```



	year	state	pop
0	2000	Ohio	1.5
1	2001	Ohio	1.7
2	2002	Ohio	3.6
3	2001	Nevada	2.4
4	2002	Nevada	2.9



DataFrame

- Tal como acontece com **Serie**, se você passar uma coluna que não está contida em **data**, o *DataFrame* resultando aparecerá com valores de NA:

```
In [42]: frame2 = DataFrame(data,  
                             columns=['year', 'state', 'pop', 'debt'],  
                             index=['one', 'two', 'three', 'four', 'five'])
```

```
In [43]: frame2
```

```
Out[43]:
```

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	NaN
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	NaN
five	2002	Nevada	2.9	NaN



DataFrame

- Uma coluna em um *DataFrame* pode ser recuperada como uma série por notação de tipo **dict** ou por **atributo**:

```
In [55]: frame2.state
```

```
Out[55]: one      Ohio  
two      Ohio  
three    Ohio  
four     Nevada  
five     Nevada  
Name: state, dtype: object
```

```
In [56]: frame2['state']
```

```
Out[56]: one      Ohio  
two      Ohio  
three    Ohio  
four     Nevada  
five     Nevada  
Name: state, dtype: object
```



DataFrame

- As linhas também podem ser recuperadas por **posição** ou **nome** por alguns métodos;
- **iloc** para índice numérico;
- **loc** para *label*.

```
In [59]: frame2.iloc[1]
```

```
Out[59]: year      2000  
state      Ohio  
pop        1.5  
debt       NaN  
Name: one, dtype: object
```

```
In [60]: frame2.loc['two']
```

```
Out[60]: year      2001  
state      Ohio  
pop        1.7  
debt       NaN  
Name: two, dtype: object
```



DataFrame

- Colunas podem ser modificadas por atribuição.
- Por exemplo, a coluna “**debt**” vazia pode receber um **array** de valores ou o produto de uma operação sobre outra coluna de dados:

```
In [66]: frame2['debt'] = np.arange(5)  
frame2
```

Out[66]:

	year	state	pop	debt
one	2000	Ohio	1.5	0
two	2001	Ohio	1.7	1
three	2002	Ohio	3.6	2
four	2001	Nevada	2.4	3
five	2002	Nevada	2.9	4

```
In [65]: frame2['debt'] = frame2['pop'] * .5  
frame2
```

Out[65]:

	year	state	pop	debt
one	2000	Ohio	1.5	0.75
two	2001	Ohio	1.7	0.85
three	2002	Ohio	3.6	1.80
four	2001	Nevada	2.4	1.20
five	2002	Nevada	2.9	1.45



DataFrame

- Ao atribuir **listas** ou **arrays** a uma coluna, o comprimento dos dados deve corresponder ao comprimento do **DataFrame**.
- Se você atribuir uma **série**, ela será ajustada exatamente aos índices do **DataFrame**, inserindo **NaN** para onde não houver correspondência.

```
In [68]: val = Series(data=[-1.2, -1.5, -1.7],  
                      index=['two', 'four', 'five'])  
frame2['debt'] = val  
frame2
```

Out[68]:

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	-1.2
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	-1.5
five	2002	Nevada	2.9	-1.7



DataFrame

- Atribuir uma coluna que não existe criará uma nova coluna.
- A palavra-chave **del** excluirá colunas:

```
In [69]: frame2['eastern'] = frame2.state == 'Ohio'  
frame2
```

Out[69]:

	year	state	pop	debt	eastern
one	2000	Ohio	1.5	NaN	True
two	2001	Ohio	1.7	-1.2	True
three	2002	Ohio	3.6	NaN	True
four	2001	Nevada	2.4	-1.5	False
five	2002	Nevada	2.9	-1.7	False

```
In [76]: del frame2['eastern']  
frame2.columns
```

Out[76]: Index(['year', 'state', 'pop', 'debt'], dtype='object')



DataFrame

- Outra forma comum de dados é um **dict** aninhado em formato de **dicts**:
- Se passado para DataFrame, ele interpretará as chaves **dict** externas como as colunas e as chaves internas como os índices de linha

```
In [79]: pop = {'Nevada': {2001: 2.4, 2002: 2.9},  
               'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}  
frame3 = DataFrame(pop)
```

```
In [80]: frame3
```

```
Out[80]:
```

	Nevada	Ohio
2000	NaN	1.5
2001	2.4	1.7
2002	2.9	3.6



Exercício



- Carregue o arquivo WorldCupMatches.csv para um dataframe usando o `.read_csv("")`;
- Monte um outro dataframe de modo a preservar apenas as informações as colunas **Year, Home Team Name, Home Team Goals, Away Team Goals, Away Team Name, Half-time Home Goals e Half-time Away Goals**;
(Dicas: você pode usar colunas diretamente, ou função zip do python ou `pd.copy()`, etc.)
- No segundo dataframe crie uma coluna **Home Team Wins**, com True quando o time da casa tiver sido o vencedor, False, caso contrário;
- Crie mais uma coluna informando a diferença de gols entre as equipes.

Obrigado

altinobasilio@inf.ufg.br
Dúvidas ou sugestões?

