

# CAPTURA E VISUALIZAÇÃO DE DADOS

Diénert Vieira

[dienertalencar@gmail.com](mailto:dienertalencar@gmail.com)

(83) 9 8182-1478

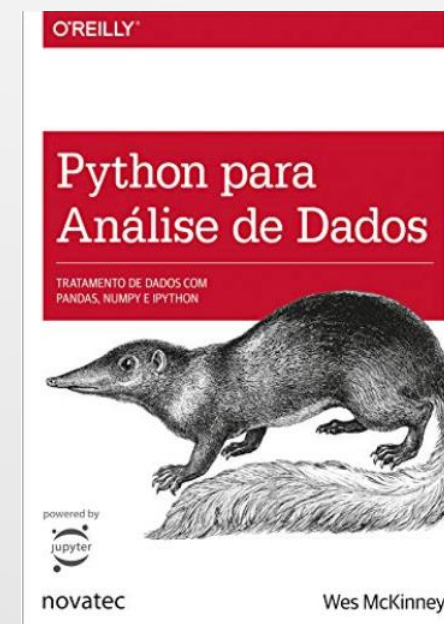
# ANÁLISE E TRANSFORMAÇÃO DE DADOS

► Framework ou biblioteca Python:

► Referência:

► Python para Análise de Dados  
de Wes McKinney (desenvolvedor líder do Pandas)

[Link para versão em português](#)



# PANDAS – PROPÓSITO E OVERVIEW

---

- ▶ Biblioteca ou framework Python com características de análise de dados similares a:
  - ▶ R
  - ▶ Matlab
  - ▶ SAS
- ▶ Construído sobre NumPy, SciPy e, até certo ponto, Matplotlib.
- ▶ Os principais componentes providos pelo pandas são:
  - ▶ Series
  - ▶ Dataframe

# PANDAS – ESTRUTURA DE DADOS: SÉRIE (SERIES)

- Objeto similar a um array (ou lista) de 1 dimensão, contendo dados e rótulos (ou índices)
- Existem várias formas de construir uma série

```
In [4]: 1 s = pd.Series([2, 4, 6, 8])
        2 s
Out[4]: 0    2
        1    4
        2    6
        3    8
        dtype: int64
```

```
In [1]: 1 import pandas as pd
```

```
In [2]: 1 s = pd.Series(list('abcdef'))
        2 s
```

```
Out[2]: 0    a
        1    b
        2    c
        3    d
        4    e
        5    f
        dtype: object
```

# PANDAS – ESTRUTURA DE DADOS: SÉRIE (SERIES)

## ► Trabalhando com índices

► Um índice pode ser especificado:

```
In [5]: 1 s = pd.Series([2, 4, 6, 8])  
        2 s.index = ['f', 'a', 'c', 'e']  
        3 s  
  
Out[5]: f      2  
        a      4  
        c      6  
        e      8  
        dtype: int64
```

► Um único valor pode ser acessado por um índice:

```
In [6]: 1 s['f']  
  
Out[6]: 2
```

► Múltiplos valores podem ser acessados por múltiplos índices:

```
In [7]: 1 s[['c', 'e']]  
  
Out[7]: c      6  
        e      8  
        dtype: int64
```

# PANDAS – ESTRUTURA DE DADOS: SÉRIE (SERIES)

## ► Trabalhando com índices

- Pense em uma série como um dict (dicionário Python) ordenado de tamanho fixo
- Contudo, diferente de um dict, os itens de um índice não precisam ser únicos.

```
In [10]:
```

1	s2 = pd.Series(range(4), index=list('abab'))
2	s2

```
Out[10]:
```

a	0
b	1
a	2
b	3

dtype: int64

```
In [11]:
```

1	s2['a']
---	---------

```
Out[11]:
```

a	0
a	2

dtype: int64

# PANDAS – ESTRUTURA DE DADOS: SÉRIE (SERIES)

## ► Operações

- Filtro
- Operações tipo-NumPy em dados

In [17]:

1	s
---	---

Out[17]:

f	2
a	4
c	6
e	8

dtype: int64

In [18]:

1	s[s > 4]
---	----------

Out[18]:

c	6
e	8

dtype: int64

In [19]:

1	s > 4
---	-------

Out[19]:

f	False
a	False
c	True
e	True

dtype: bool

In [20]:

1	s*2
---	-----

Out[20]:

f	4
a	8
c	12
e	16

dtype: int64

# PANDAS – ESTRUTURA DE DADOS: SÉRIE (SERIES)

## ► Dados Incompletos

- Pandas pode acomodar dados incompletos

```
In [21]: 1 dicionario = {'b': 100, 'c': 150, 'd': 200}
          2 s = pd.Series(dicionario)
          3 s
```

```
Out[21]: b    100
          c    150
          d    200
          dtype: int64
```

```
In [22]: 1 s = pd.Series(dicionario, list('abcd'))
          2 s
```

```
Out[22]: a      NaN
          b    100.0
          c    150.0
          d    200.0
          dtype: float64
```



# PANDAS – ESTRUTURA DE DADOS: SÉRIE (SERIES)

---

- ▶ Dados Incompletos
  - ▶ Pandas pode acomodar dados incompletos

O que acontece se multiplicarmos  $s^2$ ?



# PANDAS – ESTRUTURA DE DADOS: SÉRIE (SERIES)

- ▶ Dados Incompletos
  - ▶ Pandas pode acomodar dados incompletos

```
In [23]: 1 s*2
Out[23]: a      NaN
          b    200.0
          c    300.0
          d    400.0
          dtype: float64
```

# PANDAS – ESTRUTURA DE DADOS: SÉRIE (SERIES)

## ► Alinhamento Automático

- Diferente de um ndarray do Numpy, os dados são alinhados automaticamente

```
In [27]:
```

```
1  
2
```

```
s2 = pd.Series([1, 2, 3], index=['c', 'b', 'a'])  
s2
```

```
Out[27]:
```

```
c    1  
b    2  
a    3
```

```
dtype: int64
```

```
In [29]:
```

```
1
```

```
s
```

```
Out[29]:
```

```
a      NaN  
b    100.0  
c    150.0  
d    200.0
```

```
dtype: float64
```

# PANDAS – ESTRUTURA DE DADOS: SÉRIE (SERIES)

---

## ▶ Alinhamento Automático

- ▶ Diferente de um ndarray do Numpy, os dados são alinhados automaticamente

O que acontece se multiplicarmos  $s*s2$ ?



# PANDAS – ESTRUTURA DE DADOS: SÉRIE (SERIES)

## ► Alinhamento Automático


- Diferente de um ndarray do Numpy, os dados são alinhados automaticamente

```
In [8]:
```

	1	s*s2
Out[8]:	a	NaN
	b	200.0
	c	150.0
	d	NaN
	dtype: float64	

# PANDAS – ESTRUTURA DE DADOS: DATAFRAME

---

- ▶ Estrutura de dados similar a planilhas, contendo uma coleção ordenada de colunas
  - ▶ Possui tanto um índice nas linhas quanto nas colunas
  - ▶ Considere como um dicionário (dict) de séries com um índice compartilhado
- 

# PANDAS – ESTRUTURA DE DADOS: DATAFRAME

- Criação de um Dataframe com um dict de listas de mesmo tamanho:

```
In [3]: 1 data = {'estado': ['PB', 'AL', 'SP', 'SP', 'SP'],  
2          'ano': [2010, 2011, 2008, 2010, 2011],  
3          'pop': [5.5, 4.5, 15.3, 15.6, 15.9]}  
4 frame = pd.DataFrame(data)  
5 frame
```

Out[3]:

	estado	ano	pop
0	PB	2010	5.5
1	AL	2011	4.5
2	SP	2008	15.3
3	SP	2010	15.6
4	SP	2011	15.9

Índice das colunas

Índice compartilhado

Valores das séries

# PANDAS – ESTRUTURA DE DADOS: DATAFRAME

- Criação de um Dataframe com um dict de dicts:

```
In [3]: 1 pop_data = {'PB': {2010: 5.5, 2011: 5.7},  
2          'SP': {2008: 15.5, 2010: 15.6, 2011: 15.9}}  
3 pop = pd.DataFrame(pop_data)  
4 pop
```

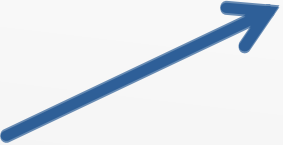
Out[3]:

	PB	SP
2010	5.5	15.6
2011	5.7	15.9
2008	NaN	15.5

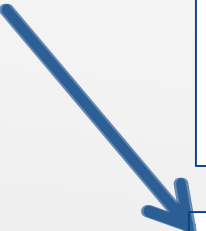


# PANDAS – ESTRUTURA DE DADOS: DATAFRAME


- ▶ Colunas podem ser recuperadas como uma Série (Series)
  - ▶ Usando a notação de um dict
  - ▶ Usando a notação de um atributo
- ▶ Linhas podem ser recuperadas pela posição ou pelo nome (usando o atributo *iloc*)



```
In [12]: 1 frame['estado']  
Out[12]: 0    PB  
          1    AL  
          2    SP  
          3    SP  
          4    SP  
          Name: estado, dtype: object
```



```
In [13]: 1 frame.estado  
Out[13]: 0    PB  
          1    AL  
          2    SP  
          3    SP  
          4    SP  
          Name: estado, dtype: object
```



```
In [15]: 1 frame.iloc[0]  
Out[15]: estado    PB  
          ano      2010  
          pop      5.5  
          Name: 0, dtype: object
```

# PANDAS – ESTRUTURA DE DADOS: DATAFRAME

► Novas colunas podem ser adicionadas por:

- uma atribuição direta
- uma computação

```
In [23]: 1 frame['calc'] = frame['pop'] * 2  
         2 frame
```

Out[23]:

	estado	ano	pop	outra	calc
0	PB	2010	5.5	NaN	11.0
1	AL	2011	4.5	NaN	9.0
2	SP	2008	15.3	NaN	30.6
3	SP	2010	15.6	NaN	31.2
4	SP	2011	15.9	NaN	31.8

```
In [22]:
```

```
1 frame['outra'] = float('NaN')  
2 frame
```

Out[22]:

	estado	ano	pop	outra
0	PB	2010	5.5	NaN
1	AL	2011	4.5	NaN
2	SP	2008	15.3	NaN
3	SP	2010	15.6	NaN
4	SP	2011	15.9	NaN

# PANDAS – ESTRUTURA DE DADOS: DATAFRAME

## ► Reindexação:

- Com a criação de um novo objeto com os dados em conformidade com um novo índice

```
In [27]: 1 obj.reindex(range(5), fill_value='preto')
```

```
Out[27]: 0      azul
         1      preto
         2      lilás
         3      preto
         4      vermelho
         dtype: object
```

```
In [29]: 1 obj
```

```
Out[29]: 0      azul
         2      lilás
         4      vermelho
         dtype: object
```

```
In [25]: 1 obj = pd.Series(['azul', 'lilás', 'vermelho'],
        2                  index=[0, 2, 4])
        3 obj
```

```
Out[25]: 0      azul
         2      lilás
         4      vermelho
         dtype: object
```

```
In [26]: 1 obj.reindex(range(4))
```

```
Out[26]: 0      azul
         1      NaN
         2      lilás
         3      NaN
         dtype: object
```

```
In [28]: 1 obj.reindex(range(5), method='ffill')
```

```
Out[28]: 0      azul
         1      azul
         2      lilás
         3      lilás
         4      vermelho
         dtype: object
```

# PANDAS – ESTRUTURA DE DADOS: DATAFRAME

## ► Somatório e Estatística Descritiva

```
In [7]: 1 pop.describe()
```

```
Out[7]:
```

	PB	SP
count	2.000000	3.000000
mean	5.600000	15.666667
std	0.141421	0.208167
min	5.500000	15.500000
25%	5.550000	15.550000
50%	5.600000	15.600000
75%	5.650000	15.750000
max	5.700000	15.900000

```
In [5]: 1 pop.sum()
```

```
Out[5]: PB      11.2  
        SP      47.0  
        dtype: float64
```

```
In [4]: 1 pop
```

```
Out[4]:
```

	PB	SP
2010	5.5	15.6
2011	5.7	15.9
2008	NaN	15.5

```
In [6]: 1 pop.mean()
```

```
Out[6]: PB      5.600000  
        SP     15.666667  
        dtype: float64
```

# PANDAS – ESTRUTURA DE DADOS: DATAFRAME

## ► Indexação Booleana

```
In [15]: 1 pop['PB'] == 5.5
```

```
Out[15]: 2010    True  
         2011   False  
         2008   False  
         Name: PB, dtype: bool
```

```
In [13]: 1 pop[pop['PB'] == 5.5]
```

```
Out[13]:
```

	PB	SP
2010	5.5	15.6

```
In [4]: 1 pop
```

```
Out[4]:
```

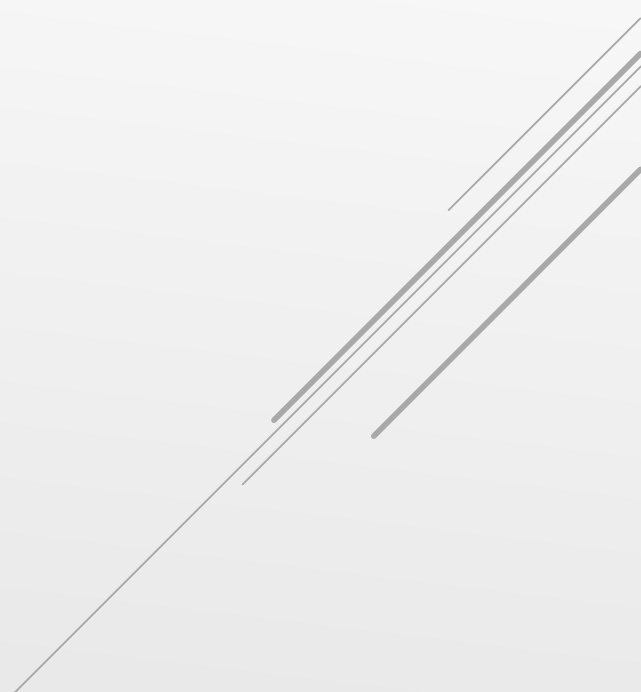
	PB	SP
2010	5.5	15.6
2011	5.7	15.9
2008	NaN	15.5

# PANDAS – CARREGANDO DADOS

---

Pandas suporta várias formas de ligar com carga de dados

- ▶ Dados em arquivo de texto
  - ▶ `pd.read_csv()`
  - ▶ `pd.read_table()`
- ▶ Dados Estruturados (JSON, XML, HTML)
- ▶ Excel (depende dos pacotes xlrd)
- ▶ Banco de dados
  - ▶ Pandas.io.sql modulo (`read_frame`)

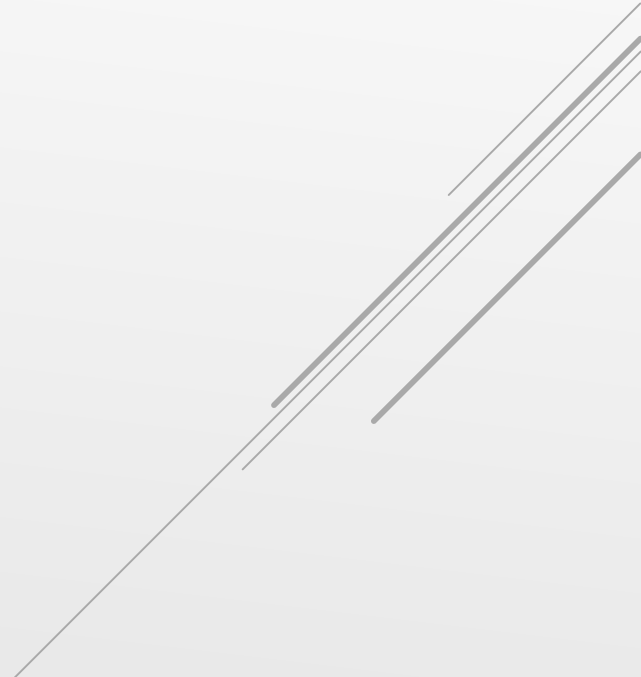


# PANDAS – CARREGANDO DADOS

---

O que mais é possível?

- ▶ Agregação de dados
  - ▶ GroupBy
  - ▶ Pivot Tables
  - ▶ Merge ou Join
- ▶ Séries Temporais
- ▶ Plots de Gráficos



# REFERÊNCIAS

---

- ▶ <https://pt.slideshare.net/AndrewHenshaw1/pandas-22984889>