

# *Introdução ao Pandas ("panel data")*



Fonte: <https://pt.wikipedia.org/wiki/Panda-gigante>

<https://pandas.pydata.org/docs/>

Professor: Alex Pereira

?

# Revisão

drop os.listdir plt.imshow n\_estimators  
df\_test warnings.filterwarnings plt.imshow mode epochs ax.set\_title  
sklearn.preprocessing plt.legend train\_test\_split del zip loc class predictions  
filenames sklearn.metrics cmap and .value\_counts text float  
palette pred bins plt.plot random\_state fontsize name enumerate test\_df model.fit feature  
ConvD .join y\_pred .fillna get\_ipython label plt.title model columns layout how plt.xticks shape  
path values rotation or plt.xlabel .sum else test return pd.read\_csv .mean batch\_size of with dataset  
seed np.mean row inplace list figsize from in True str index train\_df params .unique  
padding on features None pd data import df train .format alpha cv .plot the count  
Dropout Dense height img not None pd data import df train .format alpha cv .plot the count  
round shuffle verbose size dict color if os print is for as len fig .values self dtype trace marker  
image warnings plt.ylabel title pandas axis def False range .astype X\_test tqdm .index inputs  
.set\_title np.int ascending sns int np plt.show numpy plt.figure np.array width np.float preds  
mask np.zeros pd.DataFrame lambda matplotlib.pyplot .apply activation elif score .min  
colors kind loss filename X\_train seaborn plt.subplots pd.concat idx y\_test .max  
optimizer target plt.subplot sklearn.model\_selection .run\_line\_magic y\_test .head \_\_init\_\_  
.groupby .map os.path.join .sort\_values sns.distplot df\_train .head  
pd.Series sklearn axes time sns.barplot metrics submission  
sns.countplot hue .count

# *Pandas*

Limpeza e análise de dados  
fácil e rápido em Python

# *Pandas*

- Adota o estilo idiomático de computação baseada em arrays
  - do NumPy
    - ✓ Preferência por processar dados sem loops
- NumPy armazena dados homogêneos
  - Pandas foi projetado para trabalhar com dados tabulares e heterogêneos
- Se tornou open source em 2010
  - conta com mais de 800 colaboradores

# *Series*

- **import pandas as pd**
- É um objeto (semelhante ao Array) de uma dimensão
  - contendo uma sequência de valores, e
  - um array de rótulos
    - ✓ chamado index (índice)

```
In [11]: obj = pd.Series([4, 7, -5, 3])
```

```
In [12]: obj
```

```
Out[12]:
```

```
0      4
```

```
1      7
```

```
2     -5
```

```
3      3
```

```
dtype: int64
```

# *Series*

- Pode-se indicar o seu array de índices

```
In [15]: obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])
```

```
In [16]: obj2
```

```
Out[16]:
```

```
d      4
```

```
b      7
```

```
a     -5
```

```
c      3
```

```
dtype: int64
```

# *Series*

- Use o índice para acessar o valor
  - Pode ser usada em contextos onde você usaria um dicionário

```
In [18]: obj2['a']
```

```
Out[18]: -5
```

```
In [19]: obj2['d'] = 6
```

```
In [20]: obj2[['c', 'a', 'd']]
```

```
Out[20]:
```

```
c      3
```

```
a     -5
```

```
d      6
```

```
dtype: int64
```

# Verificar se um elemento pertence a uma Serie

- Usando o keyword in
  - Pode-se verificar se um elemento pertence ao **índice** de um a Serie

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

```
1 # Verifique se o índice 'c' está presente
2 'c' in obj2
```

True



# Series

- Use o operações semelhantes às operações NumPy
  - filtro com um array de booleanos, multiplicação por escalar e funções matemáticas

```
In [21]: obj2[obj2 > 0]
Out[21]:
d      6
b      7
c      3
dtype: int64
```

```
In [22]: obj2 * 2
Out[22]:
d     12
b     14
a    -10
c      6
dtype: int64
```

```
In [23]: np.exp(obj2)
Out[23]:
d     403.428793
b    1096.633158
a      0.006738
c     20.085537
dtype: float64
```

# Series

- Criando uma Serie com um dicionário
  - mas especificando seu próprio índice separadamente
    - ✓ sdata= {'Ohio': 35000, 'Texas': 71000, 'Oregon': None, 'Utah': None}
    - In [29]: states = ['California', 'Ohio', 'Oregon', 'Texas']
    - In [30]: obj4 = pd.Series(sdata, index=states)
    - In [31]: obj4
    - Out[31]:

California	NaN
Ohio	35000.0
Oregon	16000.0
Texas	71000.0

    - dtype: float64

# Valores Ausentes/Indisponíveis (NA)

- Verifique quais elementos são NA
  - e vice versa

```
In [32]: pd.isnull(obj4)
Out[32]:
California    True
Ohio          False
Oregon        False
Texas         False
dtype: bool
```

```
In [33]: pd.notnull(obj4)
Out[33]:
California    False
Ohio          True
Oregon        True
Texas         True
dtype: bool
```

```
In [34]: obj4.isnull()
Out[34]:
California    True
Ohio          False
Oregon        False
Texas         False
dtype: bool
```

# Aritmética com Series

- Automaticamente alinha os valores pelo índice

```
In [35]: obj3  
Out[35]:  
Ohio      35000  
Oregon    16000  
Texas     71000  
Utah       5000  
dtype: int64
```

```
In [36]: obj4  
Out[36]:  
California      NaN  
Ohio            35000.0  
Oregon          16000.0  
Texas           71000.0  
dtype: float64
```

```
In [37]: obj3 + obj4  
Out[37]:  
California      NaN  
Ohio            70000.0  
Oregon          32000.0  
Texas          142000.0  
Utah            NaN  
dtype: float64
```

# DataFrame

- Representa uma tabela retangular de dados
- Contém uma coleção de colunas ordenadas
  - Onde cada coluna pode conter diferentes tipos de dados
    - ✓ numérico, string, boolean e etc
- Tem uma linha e uma coluna de índices

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],  
        'year': [2000, 2001, 2002, 2001, 2002, 2003],  
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}  
frame = pd.DataFrame(data)
```

```
In [45]: frame  
Out[45]:
```

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

# *Mostre as primeiras linhas de um DataFrame*

- Use o método head

```
In [46]: frame.head()
```

```
Out[46]:
```

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

# *Uma coluna pode ser recuperada como uma Serie*

- A notação `.nome_coluna` não funciona
  - Quando o nome da coluna contém espaços
    - ✓ Acostume-se a usar a notação `['nome_coluna']`

```
In [51]: frame2['state']  
Out[51]:  
one      Ohio  
two      Ohio  
three    Ohio  
four     Nevada  
five     Nevada  
six      Nevada  
Name: state, dtype: object
```

```
In [52]: frame2.year  
Out[52]:  
one      2000  
two      2001  
three    2002  
four     2001  
five     2002  
six      2003  
Name: year, dtype: int64
```

# Colunas podem ser modificadas por atribuição

- Atribuição de um valor único ou de um array numpy

```
In [54]: frame2['debt'] = 16.5
```

```
In [55]: frame2
```

```
Out[55]:
```

	year	state	pop	debt
one	2000	Ohio	1.5	16.5
two	2001	Ohio	1.7	16.5
three	2002	Ohio	3.6	16.5
four	2001	Nevada	2.4	16.5
five	2002	Nevada	2.9	16.5
six	2003	Nevada	3.2	16.5

```
In [56]: frame2['debt'] = np.arange(6.)
```

```
In [57]: frame2
```

```
Out[57]:
```

	year	state	pop	debt
one	2000	Ohio	1.5	0.0
two	2001	Ohio	1.7	1.0
three	2002	Ohio	3.6	2.0
four	2001	Nevada	2.4	3.0
five	2002	Nevada	2.9	4.0
six	2003	Nevada	3.2	5.0



## *Atribuir uma coluna que não existe criará uma nova coluna*

```
In [61]: frame2['eastern'] = frame2.state == 'Ohio'
```

```
In [62]: frame2
```

```
Out[62]:
```

	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
six	2003	Nevada	3.2	NaN	False

*del exclui as colunas (como em um dicionário)*

```
In [63]: del frame2['eastern']
```

```
In [64]: frame2.columns
```

```
Out[64]: Index(['year', 'state', 'pop', 'debt'], dtype='object')
```

## *Criar um DataFrame a partir de um dicionário de dicionários*

```
In [65]: pop = {'Nevada': {2001: 2.4, 2002: 2.9},  
.....:         'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}
```

```
In [66]: frame3 = pd.DataFrame(pop)
```

```
In [67]: frame3
```

```
Out[67]:
```

	Nevada	Ohio
2000	NaN	1.5
2001	2.4	1.7
2002	2.9	3.6

*O atributo values retorna um array 2d com os valores*

```
In [74]: frame3.values  
Out[74]:  
array([[ nan,  1.5],  
       [ 2.4,  1.7],  
       [ 2.9,  3.6]])
```

## *Removendo entradas de uma Serie*

```
In [105]: obj = pd.Series(np.arange(5.), index=['a', 'b', 'c', 'd', 'e'])
```

```
In [106]: obj
```

```
Out[106]:
```

```
a    0.0
```

```
b    1.0
```

```
c    2.0
```

```
d    3.0
```

```
e    4.0
```

```
dtype: float64
```

```
In [107]: new_obj = obj.drop('c')
```

```
In [108]: new_obj
```

```
Out[108]:
```

```
a    0.0
```

```
b    1.0
```

```
d    3.0
```

```
e    4.0
```

```
dtype: float64
```

# Removendo entradas de um DataFrame

- Pode-se remover linhas ou colunas

```
In [110]: data = pd.DataFrame(np.arange(16).reshape((4, 4)),  
.....:                      index=['Ohio', 'Colorado', 'Utah', 'New York'],  
.....:                      columns=['one', 'two', 'three', 'four'])
```

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
In [112]: data.drop(['Colorado', 'Ohio'])
```

Out[112]:

	one	two	three	four
Utah	8	9	10	11
New York	12	13	14	15

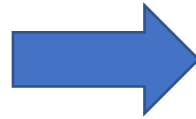
```
In [113]: data.drop('two', axis=1)
```

Out[113]:

	one	three	four
Ohio	0	2	3
Colorado	4	6	7
Utah	8	10	11
New York	12	14	15

# Atribuindo valor num DataFrame

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15



In [134]: data < 5  
Out[134]:

	one	two	three	four
Ohio	True	True	True	True
Colorado	True	False	False	False
Utah	False	False	False	False
New York	False	False	False	False

In [135]: data[data < 5] = 0

In [136]: data  
Out[136]:

	one	two	three	four
Ohio	0	0	0	0
Colorado	0	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

# Filtro e Seleção

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

In [130]: data['two']  
Out[130]:

Ohio	1
Colorado	5
Utah	9
New York	13

Name: two, dtype: int64

In [131]: data[['three', 'one']]  
Out[131]:

	three	one
Ohio	2	0
Colorado	6	4
Utah	10	8
New York	14	12

In [133]: data[data['three'] > 5]  
Out[133]:

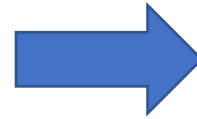
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15



# Seleção com o Operador loc e iloc

- Seleção de linhas ou linhas e colunas

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15



```
In [138]: data.iloc[2, [3, 0, 1]]  
Out[138]:  
four      11  
one        8  
two        9  
Name: Utah, dtype: int64
```



```
In [137]: data.loc['Colorado', ['two', 'three']]  
Out[137]:  
two      5  
three    6  
Name: Colorado, dtype: int64
```

# *Filtro e Seleção - Resumo*

- `.loc` faz filtros
  - de linhas pelo nome das linhas no índice ou por um vetor de booleanos;
  - de linhas e colunas pelo nome das linhas no índice e pelos nomes das colunas; e
  - de linhas e colunas por um vetor de booleanos (True e False).
- `.iloc` faz filtros
  - de linhas e colunas pelo número das linhas no índice e pelo número das colunas.
- `[]`
  - de colunas pelos nomes das colunas; e
  - de linhas por um vetor de booleanos (True e False).

# Filtro e Seleção - Exemplos

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],  
        'year': [2000, 2001, 2002, 2001, 2002, 2003],  
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}  
frame = pd.DataFrame(data, index=['a', 'b', 'c', 'd', 'e', 'f'])
```

```
# Nome(s) de coluna(s)  
print(frame.loc['a'])  
# Nome(s) de linha(s) e nomes de coluna(s)  
print(frame.loc['a', 'pop'])  
# Vetor de booleano e nome de coluna  
print(frame.loc[[True, True, False, False, False, False], 'pop'])  
# Vetor(es) de booleano(s)  
print(frame.loc[[True, True, False, False, False, False], [True, True, False]])  
print(frame.iloc[0:2, 1]) # Índice(s) da(s) linha(s) e coluna(s)  
print(frame['pop']) # Nome(s) da(s) colunas  
print(frame[[True, True, False, False, False, False]]) # Vetor de booleano
```

# Prática no Colab Notebook

- Escolham por onde começar: Teoria, Warmup ou Exercícios;
  - As soluções dos warmups já estão publicadas;
  - As soluções dos exercícios extra serão disponibilizadas no dia seguinte;
- É esperado que não terminem todos os exercícios durante a aula;
  - Façam o restante ao longo da semana.
- Ao final da lista você será capaz executar tarefas relevantes

