

CC8210 – NCA210

Programação Avançada I

Prof. Reinaldo A. C. Bianchi
Prof. Isaac Jesus da Silva
Prof. Danilo H. Perico

Aula de Hoje

- Introdução à biblioteca Pandas.
- Referências:
 - Python for Data Analysis, Wes McKinney, O'Reilly Media, 2013.
 - <https://github.com/wesm/pydata-book>
 - <https://pandas.pydata.org>
 - <https://pandas.pydata.org/docs/>
 - [https://pt.wikipedia.org/wiki/Pandas_\(software\)](https://pt.wikipedia.org/wiki/Pandas_(software))
 - Entre outras (explicitas no final da aula)

Introdução à Biblioteca Pandas



The Giant Panda

Giant pandas have black fur on their ears, around their eyes (eye patches), muzzle, legs, and shoulders.



pandas

The Python Pandas

The **Pandas** library is built on NumPy and provides easy-to-use **data structures** and **data analysis** tools for the Python programming language.

O que é a Biblioteca Pandas

- Pandas é uma biblioteca criada para a linguagem Python para a manipulação e análise de dados.
- Em particular, oferece estruturas e operações para manipular tabelas numéricas e séries temporais.
- Usada para grandes quantidades de dados
 - Data Mining e Machine Learning.
- É um software livre.

Pandas é muito popular...

- Fácil uso e aprendizado da biblioteca.
- Porta de entrada para o mundo de *Data Science*.
- **Pandas é uma ótima biblioteca para realizar análises exploratórias dos dados.**
- Fortemente otimizada para desempenho, a biblioteca tem base nas linguagens Cython e C.

Principais características da biblioteca Pandas

- Séries temporais (*time series*):
 - Ferramentas para geração de intervalo de dados e conversão de frequência, estatísticas, entre outras.
- DataFrames (*quadro de dados*):
 - Ferramentas para manipulação de dados, com indexação integrada.
- Ferramentas para ler e escrever dados entre diferentes estruturas de dados e formatos de arquivos.

Principais características da biblioteca Pandas

- Alinhamento de dados e manipulação de dados ausentes.
- Reformatação e pivoteamento de matrizes.
- Divisão (slicing), fancy indexing, e subsetting de grandes conjuntos de dados.
- Facilidades para inserir e retirar colunas em conjuntos de dados.
- Ferramentas para fundir ou juntar conjuntos de dados.
- Filtragem e limpeza de dados.

História

- O desenvolvedor [Wes McKinney](#) começou a desenvolver a Pandas em 2008, enquanto trabalhava na empresa AQR Capital Management.
- A ideia veio quando ele percebeu a necessidade de uma ferramenta flexível e de alto desempenho para realizar análise quantitativa em dados financeiros.
- Antes de deixar a AQR, Wes conseguiu convencer a empresa a liberar a biblioteca como open-source.

Aplicações

- A principal aplicação de pandas é para a análise exploratória de dados.
- Muito usado em aplicações financeiras:
 - Econometria para dados que incluem várias dimensões (indivíduos, empresas, etc) acompanhadas ao longo do tempo.
- Data Science e Machine Learning...

Pandas versus Numpy

- Semelhante ao NumPy, fornece estruturas e ferramentas de análise de dados de alto desempenho e fáceis de usar.
- Ao contrário da biblioteca NumPy, que fornece objetos para arrays multidimensionais, o Pandas fornece um objeto de tabela 2d na memória: o Dataframe.

Instalando e importando Pandas

- Instalando:

- Pandas já vem pre-instalado na maioria dos sistemas
- Caso necessite instalar, use:
- `pip install pandas`

- Importando:

- A biblioteca deve ser importada sempre da seguinte maneira:
- `import pandas as pd`
- É uma regra de convenção que facilita identificar a biblioteca.

Estruturas de dados em Pandas

- **Séries:**
 - É um conjunto de dados escalares.
 - É um array de 1 dimensão.
 - Também pode ser visto como uma coluna de uma tabela
- **DataFrames:**
 - É um conjunto de Series.
 - É uma estrutura de dados de 2 dimensões — colunas e linhas.
 - É uma tabela de dados, semelhante a uma planilha Excel.

10 minutes to pandas

https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html

<https://medium.com/data-hackers/uma-introdução-simples-ao-pandas-1e15eea37fa1>

Series

- Uma Series é como um array unidimensional, uma lista de valores.
- Toda Series possui um índice, o `index`, que dá rótulos a cada elemento da lista.
- O objeto series em pandas é chamado `Series`.

Criando uma Série

- Podemos criar um objeto série usando a função `series()`:

```
import pandas as pd  
  
notas = pd.Series([2, 7, 5, 10, 6])  
  
print(notas)  
print(type(notas))
```

Saída:

```
0      2  
1      7  
2      5  
3     10  
4      6  
dtype: int64  
<class  
'pandas.core.series.Series'>
```

Atributos das séries

- Podemos verificar os atributos da Serie, comecemos pelos valores e o índice, os dois atributos fundamentais nesta estrutura:

```
notas.values
```

```
Out[5]: array([ 2,  7,  5, 10,  6])
```

```
notas.index
```

```
Out[6]: RangeIndex(start=0, stop=5, step=1)
```

Criando uma Série com índice específico

- Como ao criar a Série notas não demos um índice específico o pandas usou os inteiros positivos crescentes como padrão.
- Em muitos casos pode ser conveniente atribuímos um índice diferente do padrão.

Criando uma Série com índice específico

- Supondo que as notas sejam notas de uma turma, poderíamos atribuir nomes ao index:

```
import pandas as pd

notas = pd.Series([2,7,5,10,6], index=["Wilfred",
"Abbie", "Harry", "Julia", "Carrie"])

print(notas)
```

Saída

Wilfred 2

Abbie 7

Harry 5

Julia 10

Carrie 6

dtype: int64

<class 'pandas.core.series.Series'>

Saída

- O index permite acessar os valores pelo seu rótulo:

```
notas["Julia"]
```

```
Out[8]: 10
```

```
notas["Harry"]
```

```
Out[9]: 5
```

Funções estatísticas

- Existe uma grande quantidade de funções para realizar estatísticas sobre os dados.
- Muitas baseadas nas existentes no NumPy.
 - `mean()`
 - Calcula a média dos dados.
 - `std()`
 - Calcula o desvio padrão.
 - `describe()`
 - imprime um resumo estatístico dos dados.

Exemplo

- `print("Média:", notas.mean())`
- `print("Desvio padrão:", notas.std())`
- `print("Descrição:\n", notas.describe())`

Saída

Média: 6.0

Desvio padrão: 2.9154759474226504

Descrição:

count	5.000000
-------	----------

mean	6.000000
------	----------

std	2.915476
-----	----------

min	2.000000
-----	----------

25%	5.000000
-----	----------

50%	6.000000
-----	----------

75%	7.000000
-----	----------

max	10.000000
-----	-----------

dtype: float64

Outras funções...

<u>abs()</u>	Return a Series/DataFrame with absolute numeric value of each element.
<u>argmax()</u> ([axis, skipna])	Return int position of the largest value in the Series.
<u>argmin()</u> ([axis, skipna])	Return int position of the smallest value in the Series.
<u>max()</u> ([axis, skipna, level, numeric_only])	Return the maximum of the values for the requested axis.
<u>min()</u> ([axis, skipna, level, numeric_only])	Return the minimum of the values for the requested axis.
<u>round()</u> ([decimals])	Round each value in a Series to the given number of decimals.

Outras funções...

<code><u>corr</u>(other[, method, min_periods])</code>	Compute correlation with <i>other</i> Series, excluding missing values.
<code><u>cov</u>(other[, min_periods, ddof])</code>	Compute covariance with Series, excluding missing values.
<code><u>mean</u>([axis, skipna, level, numeric_only])</code>	Return the mean of the values for the requested axis.
<code><u>median</u>([axis, skipna, level, numeric_only])</code>	Return the median of the values for the requested axis.
<code><u>quantile</u>([q, interpolation])</code>	Return value at the given quantile.
<code><u>rolling</u>(window[, min_periods, center, ...])</code>	Provide rolling window calculations.

Data Frames

- Um DataFrame, ou um quadro de dados, é uma estrutura bidimensional de dados, como uma planilha.
- É um conjunto de Series.
- Os nomes das colunas podem ser usadas pra acessar seus valores.
- O objeto Data Frame em Pandas é chamado `DataFrame`.

Exemplo: Data Frame

- Abaixo criaremos um DataFrame que possui valores de diferentes tipos, usando um dicionário como entrada dos dados:

```
df = pd.DataFrame({'Aluno' : ["Wilfred", "Abbie",  
"Harry", "Julia", "Carrie"],  
'Faltas' : [3,4,2,1,4],  
'Prova' : [2,7,5,10,6],  
'Seminário': [8.5,7.5,9.0,7.5,8.0]})
```

Exemplo: Data Frame

df

	Aluno	Faltas	Prova	Seminário
0	Wilfred	3	2	8.5
1	Abbie	4	7	7.5
2	Harry	2	5	9.0
3	Julia	1	10	7.5
4	Carrie	4	6	8.0

Exemplo: Data Frame – dtypes

```
df.dtypes
```

```
Aluno          object  
Faltas         int64  
Prova          int64  
Seminário     float64  
dtype: object
```

Exemplo: Data Frame – columns

```
df.columns
```

```
Index(['Aluno', 'Faltas', 'Prova', 'Seminário'], dtype='object')
```


Exemplo: Data Frame – describe()

```
df.describe()
```

	Faltas	Prova	Seminário
count	5.00000	5.000000	5.00000
mean	2.80000	6.000000	8.10000
std	1.30384	2.915476	0.65192
min	1.00000	2.000000	7.50000
25%	2.00000	5.000000	7.50000
50%	3.00000	6.000000	8.00000
75%	4.00000	7.000000	8.50000
max	4.00000	10.000000	9.00000

Ordenação de DataFrames

- Uma facilidade muito importante é a função para ordenação dos dataFrames.
- `sort_values()` :
 - Sort a Series in ascending or descending order by some criterion.
 - `axis{0 or 'index'}` :
 - Axis to direct sorting.
 - Ascending : bool, default True
 - If True, sort values in ascending order, otherwise descending
- Usar o método `sort_values` não modifica o DataFrame original.

Ordenação de DataFrames

```
df.sort_values(by="Seminário")
```

	Aluno	Faltas	Prova	Seminário
1	Abbie	4	7	7.5
3	Julia	1	10	7.5
4	Carrie	4	6	8.0
0	Wilfred	3	2	8.5
2	Harry	2	5	9.0

Acessando ou indexando os DataFrames

- The axis labeling information in pandas objects serves many purposes:
 - Identifies data (i.e. provides *metadata*) using known indicators, important for analysis, visualization, and interactive console display.
 - Enables automatic and explicit data alignment.
 - Allows intuitive getting and setting of subsets of the data set.
- How to slice, dice, and generally get and set subsets of pandas objects?

Acessando ou indexando os DataFrames

- Object selection has had a number of user-requested additions in order to support more explicit location based indexing.
- Pandas supports three types of multi-axis indexing.
 - `.loc` is primarily label based, but may also be used with a boolean array.
 - `.iloc` is primarily integer position based (from 0 to length-1 of the axis), but may also be used with a boolean array.
 - `[]` – para acessar as colunas

Acessando as colunas de um Dataframe

- Os nomes das colunas podem ser usadas pra acessar seus valores:

```
df["Seminário"]
```

```
0      8.5
```

```
1      7.5
```

```
2      9.0
```

```
3      7.5
```

```
4      8.0
```

```
Name: Seminário, dtype: float64
```

Acessando ou indexando os DataFrames

- Para selecionar pelo index ou rótulo usamos o atributo `.loc`:

```
df.loc[3]
```

Aluno	Julia
Faltas	1
Prova	10
Seminário	7.5
Name: 3, dtype: object	

Acessando ou indexando os DataFrames

- Com o `.loc` também podemos fazer buscas no DataFrame:

```
df.loc[df.Faltas > 2]
```

	Aluno	Faltas	Prova	Seminário
0	Wilfred	3	2	8.5
1	Abbie	4	7	7.5
4	Carrie	4	6	8.0

Acessando ou indexando os DataFrames

- Para selecionar pelo index usamos o atributo `.iloc`:

```
df.iloc[3]
```

Aluno	Julia
Faltas	1
Prova	10
Seminário	7.5

Name: 3, dtype: object

Diferença entre o .loc e o .iloc

```
df2 = df.sort_values(by="Seminário")  
df2
```

	Aluno	Faltas	Prova	Seminário
1	Abbie	4	7	7.5
3	Julia	1	10	7.5
4	Carrie	4	6	8.0
0	Wilfred	3	2	8.5
2	Harry	2	5	9.0

Diferença entre o .loc e o .iloc

```
df2.loc[3]
```

Aluno	Julia
Faltas	1
Prova	10
Seminário	7.5

Name: 3, dtype: object

```
df2.iloc[3]
```

Aluno	Wilfred
Faltas	3
Prova	2
Seminário	8.5

Name: 0, dtype: object

Acessando os DataFrames: Boolean Indexing

- Para selecionar de acordo com critérios condicionais, se usa o que se chama de Boolean Indexing.
- Suponha que queiramos selecionar apenas as linhas em que o valor da coluna *Seminário* seja acima de 8.0, podemos realizar esta tarefa passando a condição diretamente como índice...

Acessando os DataFrames: Boolean Indexing

```
df[df["Seminário"] > 8.0]
```

	Aluno	Faltas	Prova	Seminário
0	Wilfred	3	2	8.5
2	Harry	2	5	9.0

Acessando os DataFrames: Boolean Indexing

- Este tipo de indexação também possibilita checar condições de múltiplas colunas.

```
df[(df["Seminário"] > 8.0)&(df["Prova"] > 3)]
```

	Aluno	Faltas	Prova	Seminário
2	Harry	2	5	9.0

Acessando ou indexando os DataFrames

- Para acessar um elemento individual, por uma característica usamos o .loc:

```
df.loc[df['Aluno'] == 'Julia']
```

	Aluno	Faltas	Prova	Seminário
3	Julia	1	10	7.5

Manipulação dos dados...

- Pode-se fazer todo tipo de manipulação para explorar os dados...
- Slicing e Reshaping das matrizes, como se fazia em numpy...

Operações aritméticas em Series

- Somando todos os valores presentes na Series por 2:
 - `s.add(2)`
- Subtraindo 2 de todos os valores:
 - `s.sub(2)`
- Multiplicando todos os valores por 2:
 - `s.mul(2)`
- Dividindo valores por 2:
 - `s.div(2)`

Operações aritméticas em Series. Exemplo:

df2

	Aluno	Faltas	Prova	Seminário
1	Abbie	4	7	7.5
3	Julia	1	10	7.5
4	Carrie	4	6	8.0
0	Wilfred	3	2	8.5
2	Harry	2	5	9.0

```
df2.Faltas = df.Faltas.sub(1)  
df2
```

	Aluno	Faltas	Prova	Seminário
1	Abbie	3	7	7.5
3	Julia	0	10	7.5
4	Carrie	3	6	8.0
0	Wilfred	2	2	8.5
2	Harry	1	5	9.0

Aplicando função do usuário

- É comum queremos aplicar uma função qualquer aos dados, ou à parte deles.
- Para isto o pandas fornece o método `.apply()`.

Aplicando função do usuário

- Por exemplo, para deixar os nomes dos alunos como apenas as suas três primeiras letras...

```
def truncar(aluno):  
    return(aluno[:3])  
df["Aluno"].apply(truncar)
```

```
0    Wil  
1    Abb  
2    Har  
3    Jul  
4    Car  
Name: Aluno, dtype: object
```

Removendo Linhas ou colunas

- Removendo linhas pelo index:
 - `df.drop([0, 1])`
- Removendo colunas utilizando o argumento axis:
 - `df.drop('Aluno', axis=1)`
- Nenhuma destas ações remove a linha ou coluna do DataFrame original...

Entrada e Saída de dados

- Na maioria das vezes, queremos analisar dados que já estão prontos e guardados em arquivos.
- A biblioteca Pandas fornece uma série de funcionalidades de leitura e escrita de dados, para os mais diversos formatos estruturais de dados existentes no mercado...
- EU PESSOALMENTE acredito que esta seja a maior vantagem do Pandas.

Entrada e Saída de dados

- Tipos de dados mais comumente usados:
 - Comma Separated Values (CSV):
 - Arquivo texto com dados separados por vírgulas, com uma entrada por linha.
 - Formato de dados aberto muito usado devido a facilidade de manipulação entre diferentes sistemas.
 - o caracter separador poder ser o ponto-e-vírgula ou outro, pre definido.
 - Excel:
 - Formato proprietário da Microsoft...

Entrada e Saída de dados

- Tipos de dados mais comumente usados:
 - JSON (JavaScript Object Notation)
 - Um modelo simples com a capacidade de estruturar informações de uma forma bem mais compacta do que a conseguida pelo modelo XML, tornando mais rápido o parsing dessas informações.
 - HTML:
 - Formato de arquivo texto no qual os sites de internet são escritos
 - SQL:
 - Formato de banco de dados.
 - Usado com servidores com o MySQL, MS-SQL, Oracle, ...

Leitura de dados em Pandas

- As funções mais usadas em Pandas são:
- `pd.read_csv()`
 - para ler arquivos .csv, formato comum de armazenar dados de tabelas
- `pd.read_excel()`
 - para ler arquivos Excel .xlsx, é necessário instalar uma biblioteca adicional pra esta funcionalidade.
- `pd.read_html()`
 - para ler tabelas diretamente de um website

Escrita de dados em Pandas

- As funções mais usadas em Pandas são:
- `pd.to_csv()`
 - para escrever arquivos .csv, formato comum de armazenar dados de tabelas
- `pd.to_excel()`
 - para escrever arquivos Excel .xlsx, é necessário instalar uma biblioteca adicional pra esta funcionalidade.
- `pd.to_html()`
 - para escrever tabelas diretamente de um website

Leitura e escrita de dados em Pandas

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	Fixed-Width Text File	read_fwf	
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
	MS Excel	read_excel	to_excel
binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	ORC Format	read_orc	
binary	Msgpack	read_msgpack	to_msgpack
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	SPSS	read_spss	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google BigQuery	read_gbq	to_gbq

Exemplo: leitura de dados cvs

- Usa-se a função `read_cvs()`
- O arquivo de dados deve estar na mesma pasta do script Python
 - Caso contrário é necessário passar o caminho completo.
- Outro argumento da função é o `sep`, que por padrão é a vírgula, mas que pode ser definido como outro caractere caso seu dado esteja usando outro separador.

Arquivo dados_apartamentos.csv

```
condominio,quartos,suites,vagas,area,bairro,preco,pm2
350,1,0.0,1.0,21,Botafogo,340000,16190.48
800,1,0.0,1.0,64,Botafogo,770000,12031.25
674,1,0.0,1.0,61,Botafogo,600000,9836.07
700,1,1.0,1.0,70,Botafogo,700000,10000.0
440,1,0.0,1.0,44,Botafogo,515000,11704.55
917,1,1.0,1.0,60,Botafogo,630000,10500.0
850,1,1.0,1.0,65,Botafogo,740000,11384.62
350,1,1.0,1.0,43,Botafogo,570000,13255.81
440,1,1.0,1.0,26,Botafogo,430000,16538.46
510,1,1.0,1.0,42,Botafogo,500000,11904.76
200,1,0.0,1.0,35,Botafogo,500000,14285.71
552,1,1.0,1.0,67,Botafogo,790000,11791.04
495,1,1.0,1.0,54,Botafogo,515000,9537.04
340,1,1.0,1.0,40,Botafogo,410000,10250.0
800,1,1.0,1.0,60,Botafogo,625000,10416.67
530,1,0.0,1.0,40,Botafogo,360000,9000.0
500,1,0.0,1.0,47,Botafogo,670000,14255.32
```

Exemplo: leitura de dados cvs

```
df = pd.read_csv("dados_apartamentos.csv")
```

- O DataFrame tem muitas linhas de dados:
 - Se for muito grande, o Python exibe o início e o final, separados por ...
- Para visualizar sucintamente as primeiras linhas de um DataFrame existe o método `.head()`
- Similarmente o `.tail()` exibe por padrão as últimas 5 linhas do DataFrame.

Saída do df

condominio	quartos	suites	vagas	area	bairro	preco	pm2	
0	350	1	0.0	1.0	21	Botafogo	340000	16190.48
1	800	1	0.0	1.0	64	Botafogo	770000	12031.25
2	674	1	0.0	1.0	61	Botafogo	600000	9836.07
3	700	1	1.0	1.0	70	Botafogo	700000	10000.00
4	440	1	0.0	1.0	44	Botafogo	515000	11704.55
...
1992	1080	3	1.0	1.0	80	Tijuca	680000	8500.00
1993	750	3	0.0	1.0	82	Tijuca	650000	7926.83
1994	700	3	1.0	1.0	100	Tijuca	629900	6299.00
1995	1850	3	1.0	2.0	166	Tijuca	1600000	9638.55
1996	800	3	1.0	1.0	107	Tijuca	540000	5046.73

[1997 rows x 8 columns]

Saida do df.head()

```
df.head()
```

```
Out[32]:
```

	condominio	quartos	suites	vagas	area	bairro	preco	pm2
0	350	1	0.0	1.0	21	Botafogo	340000	16190.48
1	800	1	0.0	1.0	64	Botafogo	770000	12031.25
2	674	1	0.0	1.0	61	Botafogo	600000	9836.07
3	700	1	1.0	1.0	70	Botafogo	700000	10000.00
4	440	1	0.0	1.0	44	Botafogo	515000	11704.55

Saida do df.tail()

```
df.tail()
```

```
Out[34]:
```

	condominio	quartos	suites	vagas	area	bairro	preco	pm2
1992	1080	3	1.0	1.0	80	Tijuca	680000	8500.00
1993	750	3	0.0	1.0	82	Tijuca	650000	7926.83
1994	700	3	1.0	1.0	100	Tijuca	629900	6299.00
1995	1850	3	1.0	2.0	166	Tijuca	1600000	9638.55
1996	800	3	1.0	1.0	107	Tijuca	540000	5046.73

Manipulação dos dados...

- Pode-se fazer todo tipo de manipulação para explorar os dados...
 - `value_counts()` :
 - é usada para obter de uma série a quantidade de valores únicos.
 - `groupby()` :
 - Group DataFrame by a Series of columns.
 - `pivot_table()` :
 - Return reshaped DataFrame organized by given index / column values

```
df["bairro"].value_counts()
```

```
Out[36]:
```

```
Copacabana      346
```

```
Tijuca          341
```

```
Botafogo        307
```

```
Ipanema         281
```

```
Leblon          280
```

```
Grajaú          237
```

```
Gávea           205
```

```
Name: bairro, dtype: int64
```

```
df["bairro"].value_counts(normalize=True)
```

```
Out[37]:
```

```
Copacabana      0.173260
```

```
Tijuca          0.170756
```

```
Botafogo        0.153731
```

```
Ipanema         0.140711
```

```
Leblon          0.140210
```

```
Grajaú          0.118678
```

```
Gávea           0.102654
```

```
Name: bairro, dtype: float64
```

```
df.groupby("bairro").mean()
```

	condominio	quartos	suites	vagas	area	preco	pm2
bairro							
Botafogo	914.475570	2.107492	1.048860	1.159609	83.837134	1.010614e+06	12034.486189
Copacabana	991.861272	2.101156	1.034682	1.080925	101.855491	1.216344e+06	11965.298699
Grajaú	619.940928	2.097046	0.970464	1.130802	79.949367	4.788869e+05	6145.624473
Gávea	985.234146	2.058537	1.029268	1.200000	88.497561	1.454571e+06	16511.582780
Ipanema	1357.120996	2.181495	1.192171	1.220641	100.615658	2.033096e+06	19738.407794
Leblon	1260.010714	2.207143	1.064286	1.164286	91.832143	1.946193e+06	20761.351036
Tijuca	681.175953	2.131965	0.944282	1.143695	81.457478	5.750780e+05	7149.804985

```
[7 rows x 7 columns]
```

```
df.groupby("bairro").mean()["pm2"].sort_values()
```

```
bairro
```

```
Grajaú          6145.624473
```

```
Tijuca          7149.804985
```

```
Copacabana      11965.298699
```

```
Botafogo        12034.486189
```

```
Gávea           16511.582780
```

```
Ipanema         19738.407794
```

```
Leblon          20761.351036
```

```
Name: pm2, dtype: float64
```

```
df.pivot_table(columns = 'bairro')
```

bairro	Botafogo	Copacabana	...	Leblon	Tijuca
area	8.383713e+01	1.018555e+02	...	9.183214e+01	81.457478
condominio	9.144756e+02	9.918613e+02	...	1.260011e+03	681.175953
pm2	1.203449e+04	1.196530e+04	...	2.076135e+04	7149.804985
preco	1.010614e+06	1.216344e+06	...	1.946193e+06	575077.982405
quartos	2.107492e+00	2.101156e+00	...	2.207143e+00	2.131965
suites	1.048860e+00	1.034682e+00	...	1.064286e+00	0.944282
vagas	1.159609e+00	1.080925e+00	...	1.164286e+00	1.143695

```
df.pivot_table(columns = 'quartos')
```

quartos	1	2	3
area	48.027079	8.043059e+01	1.268979e+02
condominio	609.255319	8.809717e+02	1.294975e+03
pm2	13534.898433	1.270310e+04	1.357596e+04
preco	630137.292070	1.034939e+06	1.807086e+06
suites	0.820116	1.093484e+00	1.136951e+00
vagas	0.963250	1.073654e+00	1.352713e+00

Leitura de dados de um servidor

- As funções podem ser usadas para ler dados hospedados em servidores...



Exemplo acesso a servidor

```
nomes =  
pd.read_json("https://servicodados.ibge.gov.br/api/v1/censos/nomes/")
```

Exemplo acesso a servidor

	nome	regiao	freq	rank	sexo
0	MARIA	0	11734129	1	
1	JOSE	0	5754529	2	
2	ANA	0	3089858	3	
3	JOAO	0	2984119	4	
4	ANTONIO	0	2576348	5	
5	FRANCISCO	0	1772197	6	
6	CARLOS	0	1489191	7	
7	PAULO	0	1423262	8	
8	PEDRO	0	1219605	9	
9	LUCAS	0	1127310	10	
10	LUIZ	0	1107792	11	
11	MARCOS	0	1106165	12	
12	LUIS	0	935905	13	
13	GABRIEL	0	932449		

	nome	regiao	freq	rank	sexo
14	RAFAEL	0	821638	15	
15	FRANCISCA	0	725642	16	
16	DANIEL	0	711338	17	
17	MARCELO	0	693215	18	
18	BRUNO	0	668217	19	
19	EDUARDO	0	632664	20	

Exemplo acesso a servidor

```
nomes =  
pd.read_json("https://servicodados.ibge.gov.br/api/v1/censos/nomes/ranking?qtd=200")
```

Exemplo acesso a servidor

	nome	regiao	freq	rank	sexo
0	MARIA	0	11734129	1	
1	JOSE	0	5754529	2	
2	ANA	0	3089858	3	
3	JOAO	0	2984119	4	
4	ANTONIO	0	2576348	5	
...
195	FABIANO	0	159150	196	
196	MILENA	0	159042	197	
197	WESLEY	0	157205	198	
198	DIOGO	0	156119	199	
199	ADILSON	0	155430	200	

200 rows × 5 columns

Saída de dados em arquivos

- A tarefa de salvar seu DataFrame externamente para um formato específico é feita com a mesma simplicidade que a leitura de dados é feita no pandas.
- Pode-se usar, por exemplo, o método `to_csv`, e o arquivo será criado com os dados do DataFrame

Leitura e escrita de dados em Pandas

Format Type	Data Description	Reader	Writer
text	<u>CSV</u>	<u>read_csv</u>	<u>to_csv</u>
text	Fixed-Width Text File	<u>read_fwf</u>	
text	<u>JSON</u>	<u>read_json</u>	<u>to_json</u>
text	<u>HTML</u>	<u>read_html</u>	<u>to_html</u>
text	Local clipboard	<u>read_clipboard</u>	<u>to_clipboard</u>
	<u>MS Excel</u>	<u>read_excel</u>	<u>to_excel</u>
binary	<u>OpenDocument</u>	<u>read_excel</u>	
binary	<u>HDF5 Format</u>	<u>read_hdf</u>	<u>to_hdf</u>
binary	<u>Feather Format</u>	<u>read_feather</u>	<u>to_feather</u>
binary	<u>Parquet Format</u>	<u>read_parquet</u>	<u>to_parquet</u>
binary	<u>ORC Format</u>	<u>read_orc</u>	
binary	<u>Msgpack</u>	<u>read_msgpack</u>	<u>to_msgpack</u>
binary	<u>Stata</u>	<u>read_stata</u>	<u>to_stata</u>
binary	<u>SAS</u>	<u>read_sas</u>	
binary	<u>SPSS</u>	<u>read_spss</u>	
binary	<u>Python Pickle Format</u>	<u>read_pickle</u>	<u>to_pickle</u>
SQL	<u>SQL</u>	<u>read_sql</u>	<u>to_sql</u>
SQL	<u>Google BigQuery</u>	<u>read_gbq</u>	<u>to_gbq</u>

Exemplos de escrita de dados...

```
df.to_csv("novo_arquivo.csv")
```

```
df.to_json("novo_arquivo.json")
```

```
df.to_html("novo_arquivo.html")
```

Conclusão

Conclusão

- Foi realizada uma breve apresentação do Pandas.
- 3 elementos principais:
 - Data Frames
 - Series
 - Leitura de arquivos
- Quando usado junto com o numpy, sklearn e matplotlib, cobrem quase todos as necessidades do cientista de dados moderno...

Conclusão

- Quando usar listas, arrays ou dataframes?
- Regra: use the simplest data structure that still satisfies your needs.
- Ordem de complexidade:
 - Listas e dicionários do Python
 - Arrays do Numpy
 - Séries e DataFrames do Pandas

Conclusão

- Quando usar listas, arrays ou dataframes?
- Se as listas do Python já resolvem seu problema, use-as.
- Se você necessita:
 - Trabalhar com arrays de 2 ou mais dimensões,
 - Realizar muitas operações de cálculo numérico,
 - Usar muitas funções matemáticas,
 - Velocidade de processamento.
- Use NumPy.

Conclusão

- Quando usar listas, arrays ou dataframes?
- Se você necessita:
 - Unir múltiplos conjuntos de dados
 - Reordenar ou fazer reshape dos dados (data Wrangling)
 - Importar e exportar dados de formatos específicos como Excel, SQL, e outras bases de dados.
- Use Pandas.

Fim



Sites usados na aula...

- https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html
- <https://medium.com/tech-grupozap/introdução-a-biblioteca-pandas-89fa8ed4fa38>
- <https://paulovasconcellos.com.br/28-comandos-úteis-de-pandas-que-talvez-você-não-conheça-6ab64beefa93>
- <https://www.alura.com.br/conteudo/pandas-io>