Aula 6B - Manipulação de Dados

Gustavo Oliveira e Andréa Rocha

Departamento de Computação Científica / UFPB

Julho de 2020

1 Manipulação de Dados em *Python*

1.1 A biblioteca *pandas*

1.2 Índices dos valores máximos ou mínimos

- Os métodos **idxmin()** e **idxmax()** retornam o índice cuja entrada fornece o valor mínimo ou máximo da *Serie* ou *DataFrame*.
- Se houverem múltiplas ocorrências de mínimos ou máximos, o método retorna a primeira ocorrência.

[1]:
```
import numpy as np
import pandas as pd
```

[2]:
```
df_exemplo = pd.read_csv('exemplo_data.csv', index_col=0);df_exemplo
```

[2]:
```
            coluna_1  coluna_2
2020-01-01 -0.416092  1.810364
2020-01-02 -0.137970  2.578520
2020-01-03  0.575827  0.060866
2020-01-04 -0.017367  1.299587
2020-01-05  1.384279 -0.381732
2020-01-06  0.549706 -1.308789
2020-01-07 -0.282296 -1.688979
2020-01-08 -0.989730 -0.028121
2020-01-09  0.275582 -0.177659
2020-01-10  0.685132  0.502535
```

[3]:
```
df_exemplo = pd.DataFrame(df_exemplo,␣
↪columns=['coluna_1','coluna_2','coluna_3'])
```

[4]:
```
df_exemplo['coluna_3'] = pd.Series([1,2,3,4,5,6,7,8,np.nan,np.
↪nan],index=df_exemplo.index)
```

[5]:
```
df_exemplo
```

[5]:
```
            coluna_1  coluna_2  coluna_3
2020-01-01 -0.416092  1.810364       1.0
2020-01-02 -0.137970  2.578520       2.0
2020-01-03  0.575827  0.060866       3.0
2020-01-04 -0.017367  1.299587       4.0
2020-01-05  1.384279 -0.381732       5.0
2020-01-06  0.549706 -1.308789       6.0
2020-01-07 -0.282296 -1.688979       7.0
2020-01-08 -0.989730 -0.028121       8.0
2020-01-09  0.275582 -0.177659       NaN
2020-01-10  0.685132  0.502535       NaN
```

[6]:
```
df_exemplo.idxmin()
```

[6]:
```
coluna_1    2020-01-08
coluna_2    2020-01-07
coluna_3    2020-01-01
dtype: object
```

[7]:
```
df_exemplo.idxmax()
```

[7]:
```
coluna_1    2020-01-05
coluna_2    2020-01-02
coluna_3    2020-01-08
dtype: object
```

1.3 Reindexar *DataFrames*

Em *pandas*, o método *reindex* faz o seguinte:

- Reordenar o *DataFrame* de acordo com o conjunto de rótulos inserido como argumento;
- Insere valores faltantes caso um rótulo do novo índice não tenha valor atribuído no conjunto de dados;
- Remove valores correspondentes a rótulos que não estão presentes no novo índice.

Exemplos:

[8]:
```
serie_Idade = pd.Series({'Ana':20, 'João': 19, 'Maria': 21, 'Pedro': 22,␣
↪'Túlio': 20}, name="Idade")
serie_Peso = pd.Series({'Ana':55, 'João': 80, 'Maria': 62, 'Pedro': 67, 'Túlio':
↪ 73}, name="Peso")
serie_Altura = pd.Series({'Ana':162, 'João': 178, 'Maria': 162, 'Pedro': 165,␣
↪'Túlio': 171}, name="Altura")
```

[9]:
```
dicionario_series_exemplo = {'Idade': serie_Idade, 'Peso': serie_Peso, 'Altura':
↪ serie_Altura}
```

[10]:
```
df_dict_series = pd.DataFrame(dicionario_series_exemplo)
```

[11]:
```
df_dict_series
```

[11]:

	Idade	Peso	Altura
Ana	20	55	162
João	19	80	178
Maria	21	62	162
Pedro	22	67	165
Túlio	20	73	171

[12]:
```
df_dict_series.reindex(index=['Victor', 'Túlio', 'Pedro', 'João'],␣
↪columns=['Altura','Peso','IMC'])
```

[12]:

	Altura	Peso	IMC
Victor	NaN	NaN	NaN
Túlio	171.0	73.0	NaN
Pedro	165.0	67.0	NaN
João	178.0	80.0	NaN

1.4 Removendo linhas ou colunas de um *DataFrame*

Para remover linhas ou colunas de um *DataFrame* do *pandas* podemos utilizar o método **drop**:

[13]:
```
df_dict_series.drop(['Ana','Maria'], axis=0)
```

[13]:

	Idade	Peso	Altura
João	19	80	178
Pedro	22	67	165
Túlio	20	73	171

[14]:
```
df_dict_series.drop(['Idade'], axis=1)
```

[14]:

	Peso	Altura
Ana	55	162
João	80	178
Maria	62	162
Pedro	67	165
Túlio	73	171

1.5 Renomear *index* e *columns*

O método **rename** retorna uma cópida na qual o *index* (no caso de *Series* e *DataFrames*) e *columns* (no caso de *DataFrames*) foram renomeados.

O método aceita como entrada um dicionário, uma *Serie* do *pandas* ou uma função.

Exemplo:

[15]:
```
serie_exemplo = pd.Series([1,2,3], index=['a','b','c'])
```

[16]:
```
serie_exemplo
```

[16]:
```
a    1
b    2
c    3
dtype: int64
```

[17]:
```
serie_exemplo.rename({'a':'abacaxi', 'b':'banana', 'c': 'cebola'})
```

[17]:
```
abacaxi    1
banana     2
cebola     3
dtype: int64
```

[18]:
```
df_dict_series
```

[18]:

	Idade	Peso	Altura
Ana	20	55	162
João	19	80	178
Maria	21	62	162
Pedro	22	67	165
Túlio	20	73	171

[19]:
```
df_dict_series.rename(index = {'Ana':'a', 'João':'j', 'Maria':'m', 'Pedro':
↪'p','Túlio':'t'},
                      columns = {'Idade':'I', 'Peso':'P','Altura':'A'})
```

[19]:

	I	P	A
a	20	55	162
j	19	80	178
m	21	62	162
p	22	67	165
t	20	73	171

[20]:
```
indice_novo = pd.Series({'Ana':'a', 'João':'j', 'Maria':'m', 'Pedro':
↪'p','Túlio':'t'})
```

[21]:
```
df_dict_series.rename(index = indice_novo)
```

[21]:

	Idade	Peso	Altura
a	20	55	162
j	19	80	178
m	21	62	162
p	22	67	165
t	20	73	171

[22]:
```
df_dict_series.rename(columns=str.upper) # Aqui utilizando uma função para␣
↪renomear
```

[22]:

	IDADE	PESO	ALTURA
Ana	20	55	162
João	19	80	178
Maria	21	62	162
Pedro	22	67	165
Túlio	20	73	171

1.6 Ordenando *Series* e *DataFrames*

É possível ordenar pelos rótulos do *index* (para tanto é necessário que eles sejam ordenáveis) ou por valores nas colunas.

- O método *sort_index* ordena a *Serie* ou o *DataFrame* pelo *index*;
- O método *sort_values* ordena a *Serie* ou o *DataFrame* pelos valores (escolhendo uma coluna ou mais colunas no caso de *DataFrames*). No caso do *DataFrame* precisa de um argumento *by* indicando qual(is) coluna(s) a ser(em) utilizada(s).

Exemplos:

[23]:
```
serie_desordenada = pd.Series({'Pedro': 22, 'Maria': 21, 'Ana':20, 'Túlio': 20,␣
↪'João': 19,  });serie_desordenada
```

[23]:
```
Pedro    22
Maria    21
Ana      20
Túlio    20
João     19
dtype: int64
```

[24]:
```
serie_desordenada.sort_index()
```

[24]:
```
Ana      20
João     19
Maria    21
Pedro    22
Túlio    20
dtype: int64
```

[25]:
```
df_desordenado = df_dict_series.
↪reindex(index=['Pedro','Maria','Ana','Túlio','João'])
```

[26]:
```
df_desordenado
```

[26]:

	Idade	Peso	Altura
Pedro	22	67	165

```
Maria     21    62     162
Ana       20    55     162
Túlio     20    73     171
João      19    80     178
```

[27]:
```
df_desordenado.sort_index()
```

[27]:
```
        Idade  Peso  Altura
Ana        20    55     162
João       19    80     178
Maria      21    62     162
Pedro      22    67     165
Túlio      20    73     171
```

[28]:
```
serie_desordenada.sort_values()
```

[28]:
```
João     19
Ana      20
Túlio    20
Maria    21
Pedro    22
dtype: int64
```

[29]:
```
df_desordenado.sort_values(by=['Idade'])
```

[29]:
```
        Idade  Peso  Altura
João       19    80     178
Ana        20    55     162
Túlio      20    73     171
Maria      21    62     162
Pedro      22    67     165
```

[30]:
```
df_desordenado.sort_values(by=['Altura','Peso']) # No caso de empate, utiliza a␣
↪coluna 'Peso'
                                                 # para desempatar
```

[30]:
```
        Idade  Peso  Altura
Ana        20    55     162
Maria      21    62     162
Pedro      22    67     165
Túlio      20    73     171
João       19    80     178
```

- Os métodos *sort_index* e *sort_values* admitem o argumento opcional *ascending*, que permite inverter a ordenação:

[31]:
```
df_desordenado.sort_index(ascending=False)
```

[31]:
```
        Idade  Peso  Altura
Túlio      20    73     171
Pedro      22    67     165
Maria      21    62     162
João       19    80     178
Ana        20    55     162
```

[32]:
```
df_desordenado.sort_values(by=['Idade'], ascending=False)
```

[32]:
```
        Idade  Peso  Altura
Pedro      22    67     165
Maria      21    62     162
Ana        20    55     162
Túlio      20    73     171
João       19    80     178
```

1.7 Comparando *Series* e *DataFrames*

Series e *DataFrames* possuem os métodos de comparações lógicas *eq* (igual), *ne* (diferente), *lt* (menor do que), *gt* (maior do que), *le* (menor ou igual), *ge* (maior ou igual), que permitem a utilização dos operadores binários $==$, $!=$, $<$, $>$, $<=$, $>=$, respectivamente.

As comparações são realizadas em cada entrada da *Serie* ou do *DataFrame*.

Observação: Para que esses métodos sejam aplicados todos os objetos presentes nas colunas do *DataFrame* devem possuir este métodos comparáveis com o que está sendo pedido. Por exemplo se um *DataFrame* possui algumas colunas numéricas e outras colunas com strings, ao realizar uma comparação do tipo > 1, teremos um erro, pois o *pandas* tentará realizar comparações entre objetos do tipo *int* e *str*.

Exemplos:

[33]:
```
serie_exemplo
```

[33]:
```
a    1
b    2
c    3
dtype: int64
```

[34]:
```
serie_exemplo == 2
```

[34]:
```
a    False
b     True
c    False
dtype: bool
```

[35]:
```
serie_exemplo > 1
```

[35]:
```
a    False
b     True
c     True
dtype: bool
```

[36]:
```
df_exemplo > 1
```

[36]:

	coluna_1	coluna_2	coluna_3
2020-01-01	False	True	False
2020-01-02	False	True	True
2020-01-03	False	False	True
2020-01-04	False	True	True
2020-01-05	True	False	True
2020-01-06	False	False	True
2020-01-07	False	False	True
2020-01-08	False	False	True
2020-01-09	False	False	False
2020-01-10	False	False	False

Observação: Ao comparar *np.nan*, o resultado tipicamente é falso:

[37]:
```
np.nan == np.nan
```

[37]:
```
False
```

[38]:
```
np.nan > np.nan
```

[38]:
```
False
```

[39]:
```
np.nan >= np.nan
```

[39]:
```
False
```

Só é verdadeiro para indicar que é diferente:

[40]:
```
np.nan != np.nan
```

[40]:
```
True
```

Nesse sentido podemos ter tabelas iguais sem que a comparação usual funcione:

[41]:
```
df_exemplo_2 = df_exemplo.copy() # Este método, como o nome sugere, fornece uma␣
↪cópia do DataFrame
```

[42]:
```
(df_exemplo == df_exemplo_2).all().all()
```

[42]:
```
False
```

O motivo da saída *False* ainda que *df_exemplo_2* seja uma cópia exata do *df_exemplo* é a presença do *np.nan*.

Para comparar neste caso devemos utilizar o método *equals*:

[43]:
```
df_exemplo.equals(df_exemplo_2)
```

[43]:
```
True
```

1.8 Os métodos *any*, *all* e a propriedade *empty*

- O método *any* é aplicado a entradas booleanas (verdadeiras ou falsas) e retorna verdadeiro se existir alguma entrada verdadeira e falsa se todas forem falsas;
- O método *all* é aplicado a entradas booleanas e retorna verdadeiro se todas as entradas forem verdadeiras e falso se houver pelo menos uma entrada falsa.
- A propriedade *empty* retorna verdadeiro se a *Serie* ou o *DataFrame* estiver vazio e falso caso contrário.

Exemplos:

[44]:
```
(df_exemplo > 1).any()
```

[44]:
```
coluna_1    True
coluna_2    True
coluna_3    True
dtype: bool
```

[45]:
```
(serie_exemplo > 1).all()
```

[45]:
```
False
```

[46]:
```
serie_exemplo.empty
```

[46]:
```
False
```

[47]:
```
pd.DataFrame().empty
```

[47]:
```
True
```

1.9 Como selecionar colunas de um *DataFrame*

Para selecionar colunas de um *DataFrame*, basta aplicar o *colchete* a uma lista contendo os nomes das colunas de interesse.

No exemplo abaixo, temos um *DataFrame* contendo as colunas *Idade*, *Peso* e *Altura*. Iremos selecionar *Peso* e *Altura*:

Exemplo:

[48]:
```
df_dict_series[['Peso','Altura']]
```

[48]:
```
        Peso  Altura
Ana       55     162
João      80     178
Maria     62     162
Pedro     67     165
Túlio     73     171
```

Se quisermos selecionar apenas uma coluna, não há a necessidade de inserir uma lista. Basta utilizar o nome da coluna:

[49]:
```
df_dict_series['Peso']
```

[49]:
```
Ana      55
João     80
Maria    62
Pedro    67
Túlio    73
Name: Peso, dtype: int64
```

Se quisermos remover algumas colunas, podemos utilizar o método **drop**:

[50]:
```
df_dict_series.drop(['Peso','Altura'], axis=1)
```

[50]:
```
        Idade
Ana        20
João       19
Maria      21
Pedro      22
Túlio      20
```

1.10 Criando novas colunas a partir das colunas já existentes

Um método eficiente para criarmos novas colunas a partir de colunas já existentes é o **eval**. Neste método podemos utilizar como argumento uma *string* contendo uma expressão matemática envolvendo nomes de colunas do *DataFrame*.

Como exemplo, vamos ver como calcular o IMC no *DataFrame* anterior:

[51]:
```
df_dict_series.eval('Peso/(Altura/100)**2')
```

[51]:
```
Ana      20.957171
João     25.249337
Maria    23.624447
Pedro    24.609734
Túlio    24.964946
dtype: float64
```

Se quisermos obter um *DataFrame* contendo o IMC como uma nova coluna, podemos utilizar o método **assign** (sem modificar o *DataFrame* original):

[52]:
```
df_dict_series.assign(IMC=df_dict_series.eval('Peso/(Altura/100)**2'))
```

[52]:

	Idade	Peso	Altura	IMC
Ana	20	55	162	20.957171
João	19	80	178	25.249337
Maria	21	62	162	23.624447
Pedro	22	67	165	24.609734
Túlio	20	73	171	24.964946

[53]:
```
df_dict_series
```

[53]:

	Idade	Peso	Altura
Ana	20	55	162
João	19	80	178
Maria	21	62	162
Pedro	22	67	165
Túlio	20	73	171

Se quisermos modificar o *DataFrame* para incluir a coluna IMC fazemos:

[54]:
```
df_dict_series['IMC']=round(df_dict_series.eval('Peso/(Altura/100)**2'),2)
```

[55]:
```
df_dict_series
```

[55]:

	Idade	Peso	Altura	IMC
Ana	20	55	162	20.96
João	19	80	178	25.25
Maria	21	62	162	23.62
Pedro	22	67	165	24.61
Túlio	20	73	171	24.96

1.11 Selecionando linhas de um *DataFrame*:

Podemos selecionar linhas de um *DataFrame* de diversas formas diferentes. Veremos agora algumas dessas formas.

Diferentemente da forma de selecionar colunas, para selecionar diretamente linhas de um *DataFrame* devemos utilizar o método **loc** (fornecendo o *index*, isto é, o rótulo da linha) ou o **iloc** (fornecendo a posição da linha):

[56]:
```
dados_covid_PB = pd.read_csv('https://superset.plataformatarget.com.br/superset/
↪explore_json/?form_data=%7B%22slice_id%22%3A1550%7D&csv=true',
                             sep=';', index_col=0)

dados_covid_PB.index = pd.to_datetime(dados_covid_PB.index) # Convertendo o␣
↪index de string para data
dados_covid_PB.loc['2020-07-10'].drop('Letalidade', axis=1)
#Aqui para vermos as informações do dia 10 de Julho de 2020
```

```
#Excluímos a coluna letalidade
#axis=1, indica que estamos removendo a coluna, axis=0, que é o padrão, indica␣
↪a remoção de linhas
```

[56]:
```
            casosAcumulados  casosNovos  descartados  recuperados  \
data
2020-07-10            59118        1504        69567        21481

            obitosAcumulados  obitosNovos
data
2020-07-10              1229           33
```

Podemos colocar um intervalo de datas como argumento (novamente excluindo a coluna letalidade e convertendo para inteiro):

[57]:
```
dados_covid_PB.loc[pd.date_range('2020-06-01',periods=5,freq="D")].
↪drop('Letalidade',axis=1)
```

[57]:
```
            casosAcumulados  casosNovos  descartados  recuperados  \
2020-06-01            13695         533        12068         2637
2020-06-02            14859        1164        13270         2920
2020-06-03            16018        1159        16043         3175
2020-06-04            17579        1561        17516         3633
2020-06-05            18579        1000        18730         3945

            obitosAcumulados  obitosNovos
2020-06-01               370           10
2020-06-02               379            9
2020-06-03               414           35
2020-06-04               438           24
2020-06-05               451           13
```

Podemos colocar uma lista como argumento:

[58]:
```
dados_covid_PB.loc[pd.to_datetime(['2020-06-01','2020-07-01'])]
```

[58]:
```
            casosAcumulados  casosNovos  descartados  recuperados  \
2020-06-01            13695         533        12068         2637
2020-07-01            48175        1218        45395        15359

            obitosAcumulados  obitosNovos  Letalidade
2020-06-01               370           10    0.027017
2020-07-01              1002           25    0.020799
```

Vamos agora olhar os dados da posição 100 (novamente excluindo a coluna letalidade e convertendo para inteiro):

[59]:
```
dados_covid_PB.iloc[100].drop('Letalidade').astype('int')
#Excluímos a linha letalidade (da Serie) e convertemos para inteiro para melhor␣
↪apresentação
```

[59]:
```
casosAcumulados      35
casosNovos            1
descartados         563
recuperados           0
obitosAcumulados      4
obitosNovos           1
Name: 2020-04-05 00:00:00, dtype: int32
```

Podemos colocar uma lista ou um intervalo como argumento:

[60]:
```
dados_covid_PB.iloc[97:100].drop('Letalidade', axis=1).astype('int')
#Aqui foi necessário mudar o eixo para coluna, já que o resultado é um␣
↪DataFrame (com axis=1)
```

[60]:
```
            casosAcumulados  casosNovos  descartados  recuperados  \
data
2020-04-08               55          14          693            0
2020-04-07               41           5          649            0
2020-04-06               36           1          608            0

            obitosAcumulados  obitosNovos
data
2020-04-08                 7            3
2020-04-07                 4            0
2020-04-06                 4            0
```

1.12 Selecionando colunas pelos métodos *loc* e *iloc*

Podemos selecionar colunas utilizando os métodos **loc** e **iloc**:

[61]:
```
dados_covid_PB.loc[:,['casosNovos','obitosNovos']]
```

[61]:
```
            casosNovos  obitosNovos
data
2020-07-14        1354           40
2020-07-13         324           18
2020-07-12         363           34
2020-07-11        1303           21
2020-07-10        1504           33
…                  …             …
2020-03-20           0            0
2020-03-19           0            0
2020-03-18           0            0
2020-03-17           0            0
```

```
2020-03-16           0            0

[121 rows x 2 columns]
```

[62]:
```
dados_covid_PB.iloc[:,4:6]
```

[62]:
```
            obitosAcumulados  obitosNovos
data
2020-07-14              1342           40
2020-07-13              1302           18
2020-07-12              1284           34
2020-07-11              1250           21
2020-07-10              1229           33
…                          …            …
2020-03-20                 0            0
2020-03-19                 0            0
2020-03-18                 0            0
2020-03-17                 0            0
2020-03-16                 0            0

[121 rows x 2 columns]
```

1.13 Selecionando linhas e colunas específicas pelos métodos *loc* e *iloc*:

[63]:
```
dados_covid_PB.loc[pd.
 ↪date_range('2020-07-05','2020-07-10'),['casosNovos','obitosNovos']]
```

[63]:
```
            casosNovos  obitosNovos
2020-07-05         422           17
2020-07-06         423           19
2020-07-07        1651           27
2020-07-08        1542           26
2020-07-09        1270           25
2020-07-10        1504           33
```

[64]:
```
dados_covid_PB.iloc[95:100,4:6]
```

[64]:
```
            obitosAcumulados  obitosNovos
data
2020-04-10                11            0
2020-04-09                11            4
2020-04-08                 7            3
2020-04-07                 4            0
2020-04-06                 4            0
```

Para alterar uma entrada específica é simples. Suponha que o peso de Ana foi medido errado e é, na realidade, 65, então, fazemos:

[65]:
```
df_dict_series.loc['Ana','Peso'] = 65

df_dict_series = df_dict_series.assign(IMC=df_dict_series.eval('Peso/(Altura/
↪100)**2')) # O IMC mudou
```

[66]:
```
df_dict_series
```

[66]:

	Idade	Peso	Altura	IMC
Ana	20	65	162	24.767566
João	19	80	178	25.249337
Maria	21	62	162	23.624447
Pedro	22	67	165	24.609734
Túlio	20	73	171	24.964946

1.13.1 Selecionando linha através de critérios lógicos ou funções:

Vamos selecionar quais os dias em que houve mais de 30 mortes registradas:

[67]:
```
dados_covid_PB.loc[dados_covid_PB['obitosNovos']>30]
```

[67]:

data	casosAcumulados	casosNovos	descartados	recuperados	\
2020-07-14	62462	1354	73028	23027	
2020-07-12	60784	363	71257	22292	
2020-07-10	59118	1504	69567	21481	
2020-07-02	49536	1361	48272	16349	
2020-06-30	46957	1900	43070	14930	
2020-06-27	44242	1410	39353	13756	
2020-06-12	26556	1186	23189	6329	
2020-06-03	16018	1159	16043	3175	

data	obitosAcumulados	obitosNovos	Letalidade
2020-07-14	1342	40	0.021485
2020-07-12	1284	34	0.021124
2020-07-10	1229	33	0.020789
2020-07-02	1044	42	0.021076
2020-06-30	977	46	0.020806
2020-06-27	896	32	0.020252
2020-06-12	610	40	0.022970
2020-06-03	414	35	0.025846

Selecionando os dias com mais de 25 óbitos e mais de 1500 casos novos:

Observação: Note, no exemplo abaixo, que podemos utilizar o nome da coluna como um atributo da série.

[68]:
```
dados_covid_PB.loc[(dados_covid_PB.obitosNovos >25) & (dados_covid_PB.
↪casosNovos>1500)]
```

[68]:

data	casosAcumulados	casosNovos	descartados	recuperados	obitosAcumulados	obitosNovos	Letalidade
2020-07-10	59118	1504	69567	21481	1229	33	0.020789
2020-07-08	56344	1542	67549	19999	1171	26	0.020783
2020-07-07	54802	1651	64933	19373	1145	27	0.020893
2020-06-30	46957	1900	43070	14930	977	46	0.020806
2020-06-09	22452	1501	20650	4671	534	27	0.023784

Vamos inserir uma coluna sobrenome no *df_dict_series*:

[69]:
```
df_dict_series['Sobrenome'] = ['Silva', 'PraDo', 'Sales', 'MachadO', 'Coutinho']
df_dict_series
```

[69]:

	Idade	Peso	Altura	IMC	Sobrenome
Ana	20	65	162	24.767566	Silva
João	19	80	178	25.249337	PraDo
Maria	21	62	162	23.624447	Sales
Pedro	22	67	165	24.609734	MachadO
Túlio	20	73	171	24.964946	Coutinho

Vamos encontrar as linhas cujo sobrenome termina em "do". Para tanto, note que a função (note que convertemos tudo para minúsculo)

```
def verifica_final_do(palavra):
    return palavra.lower()[-2:] == 'do'
```

retorna *True* se o final é "do" e *False* caso contrário.

Agora vamos utilizar essa função para alcançar nosso objetivo:

[70]:
```
df_dict_series['Sobrenome'].map(lambda palavra: palavra.lower()[-2:]=='do')
```

[70]:
```
Ana      False
João      True
Maria    False
Pedro     True
Túlio    False
Name: Sobrenome, dtype: bool
```

[71]:
```
df_dict_series.loc[df_dict_series['Sobrenome'].map(lambda palavra: palavra.
↪lower()[-2:]=='do')]
```

[71]:
```
       Idade  Peso  Altura        IMC Sobrenome
João      19    80     178  25.249337     PraDo
Pedro     22    67     165  24.609734   MachadO
```

Vamos selecionar as linhas do mês 4 (Abril):

[72]:
```
dados_covid_PB.loc[dados_covid_PB.index.month==4].head()
```

[72]:
```
            casosAcumulados  casosNovos  descartados  recuperados  \
data
2020-04-30              926         112         1695          156
2020-04-29              814         115         1616          152
2020-04-28              699          66         1531          149
2020-04-27              633          90         1482          119
2020-04-26              543          44         1421          119

            obitosAcumulados  obitosNovos  Letalidade
data
2020-04-30                67            5    0.072354
2020-04-29                62            4    0.076167
2020-04-28                58            5    0.082976
2020-04-27                53            3    0.083728
2020-04-26                50            1    0.092100
```

1.14 Selecionando linhas com o método *query*

No mesmo espírito do método *eval*, ao utilizarmos o método query podemos criar expressões lógicas a partir de nomes das colunas do *DataFrame*.

Assim, podemos reescrever o código

```
dados_covid_PB.loc[(dados_covid_PB.obitosNovos>25) & (dados_covid_PB.casosNovos>1500)]
```

como

[73]:
```
dados_covid_PB.query('obitosNovos>25 and casosNovos>1500')
```

[73]:
```
            casosAcumulados  casosNovos  descartados  recuperados  \
data
2020-07-10            59118        1504        69567        21481
2020-07-08            56344        1542        67549        19999
2020-07-07            54802        1651        64933        19373
2020-06-30            46957        1900        43070        14930
2020-06-09            22452        1501        20650         4671

            obitosAcumulados  obitosNovos  Letalidade
```

```
data
2020-07-10              1229           33    0.020789
2020-07-08              1171           26    0.020783
2020-07-07              1145           27    0.020893
2020-06-30               977           46    0.020806
2020-06-09               534           27    0.023784
```

1.15 Agregando informações de linhas ou colunas

Para agregar informações (por exemplo somar, tomar médias, etc) de linhas ou colunas podemos utilizar alguns métodos específicos já existentes em *DataFrames* e *Series*, como *sum*, *mean*, *cumsum*, etc, como também podemos utilizar o método *aggregate* ou equivalentemente *agg*:

[74]:
```
dados_covid_PB.agg(lambda vetor: np.sum(vetor))[['casosNovos','obitosNovos']].
↪astype('int')
```

[74]:
```
casosNovos     62462
obitosNovos     1342
dtype: int32
```

[75]:
```
dados_covid_PB.head()
```

[75]:
```
            casosAcumulados  casosNovos  descartados  recuperados  \
data
2020-07-14            62462        1354        73028        23027
2020-07-13            61108         324        71609        22468
2020-07-12            60784         363        71257        22292
2020-07-11            60421        1303        70966        22116
2020-07-10            59118        1504        69567        21481

            obitosAcumulados  obitosNovos  Letalidade
data
2020-07-14              1342           40    0.021485
2020-07-13              1302           18    0.021307
2020-07-12              1284           34    0.021124
2020-07-11              1250           21    0.020688
2020-07-10              1229           33    0.020789
```

Isto também pode ser obtido utilizando o método *sum* de *DataFrames* e *Series*:

[76]:
```
dados_covid_PB[['casosNovos','obitosNovos']].sum()
```

[76]:
```
casosNovos     62462
obitosNovos     1342
dtype: int64
```

Podemos recriar a coluna obitosAcumulados com o método *cumsum*:

[77]:
```
dados_covid_PB.obitosNovos.sort_index().cumsum()
```

[77]:
```
data
2020-03-16       0
2020-03-17       0
2020-03-18       0
2020-03-19       0
2020-03-20       0
                ...
2020-07-10    1229
2020-07-11    1250
2020-07-12    1284
2020-07-13    1302
2020-07-14    1342
Name: obitosNovos, Length: 121, dtype: int64
```

1.16 Selecionando entradas distintas

Para selecionar entradas distintas utilizamos o método *drop_duplicate*. Aqui, para exemplificar, vamos utilizar o banco de dados oficial de covid do Brasil:

[78]:
```
covid_BR = pd.read_excel('HIST_PAINEL_COVIDBR_12jul2020.xlsx')
```

[79]:
```
covid_BR.estado.drop_duplicates().dropna().array
```

[79]:
```
<PandasArray>
['RO', 'AC', 'AM', 'RR', 'PA', 'AP', 'TO', 'MA', 'PI', 'CE', 'RN', 'PB', 'PE',
 'AL', 'SE', 'BA', 'MG', 'ES', 'RJ', 'SP', 'PR', 'SC', 'RS', 'MS', 'MT', 'GO',
 'DF']
Length: 27, dtype: object
```

1.17 Agrupando dados por valores em colunas e agregando os resultados

Vamos determinar uma coluna para agrupar. No caso, iremos considerar o *DataFrame* **covid_BR**, vamos selecionar os estados *PB*, *PE*, *RJ*, *SP* e vamos realizar alguns cálculos com eles, agrupando os resultados por estados.

[80]:
```
covid_BR.query('estado in ["PB", "PE", "RJ", "SP"]')
```

[80]:
```
          regiao estado     municipio  coduf    codmun  codRegiaoSaude  \
1668    Nordeste     PB           NaN     25       NaN             NaN
1669    Nordeste     PB           NaN     25       NaN             NaN
1670    Nordeste     PB           NaN     25       NaN             NaN
1671    Nordeste     PB           NaN     25       NaN             NaN
1672    Nordeste     PB           NaN     25       NaN             NaN
...          ...    ...           ...    ...       ...             ...
424007   Sudeste     SP  Estiva Gerbi     35  355730.0         35141.0
424008   Sudeste     SP  Estiva Gerbi     35  355730.0         35141.0
```

```
424009   Sudeste     SP  Estiva Gerbi     35  355730.0         35141.0
424010   Sudeste     SP  Estiva Gerbi     35  355730.0         35141.0
424011   Sudeste     SP  Estiva Gerbi     35  355730.0         35141.0

       nomeRegiaoSaude       data  semanaEpi populacaoTCU2019  casosAcumulado  \
1668               NaN 2020-02-25          9          4018127               0
1669               NaN 2020-02-26          9          4018127               0
1670               NaN 2020-02-27          9          4018127               0
1671               NaN 2020-02-28          9          4018127               0
1672               NaN 2020-02-29          9          4018127               0
…                  …          …          …               …              …
424007   BAIXA MOGIANA 2020-07-08         28            11304              48
424008   BAIXA MOGIANA 2020-07-09         28            11304              52
424009   BAIXA MOGIANA 2020-07-10         28            11304              53
424010   BAIXA MOGIANA 2020-07-11         28            11304              53
424011   BAIXA MOGIANA 2020-07-12         29            11304              56

        casosNovos  obitosAcumulado  obitosNovos  Recuperadosnovos  \
1668             0                0            0               NaN
1669             0                0            0               NaN
1670             0                0            0               NaN
1671             0                0            0               NaN
1672             0                0            0               NaN
…              …                …          …                 …
424007           2                2            0               NaN
424008           4                2            0               NaN
424009           1                2            0               NaN
424010           0                2            0               NaN
424011           3                2            0               NaN

        emAcompanhamentoNovos  interior/metropolitana
1668                      NaN                     NaN
1669                      NaN                     NaN
1670                      NaN                     NaN
1671                      NaN                     NaN
1672                      NaN                     NaN
…                         …                       …
424007                    NaN                     0.0
424008                    NaN                     0.0
424009                    NaN                     0.0
424010                    NaN                     0.0
424011                    NaN                     0.0

[123244 rows x 17 columns]
```

Dando uma inspecionada neste conjunto de dados, observamos que os dados para o estado são apresentados com o valor *NaN* para **codmun** e quando **codmun** possui um valor diferente de

NaN, o resultado é apenas para o município do código em questão.

Como estamos interessados nos valores por estado, vamos selecionar apenas os dados com munícipio *NaN*:

[81]:
```
covid_estados = covid_BR.query('estado in ["PB", "PE", "RJ", "SP"]')
covid_apenas_estados = covid_estados.loc[covid_estados['codmun'].isna()]
```

Vamos agora apenas selecionar as colunas de interesse. Para tanto, vejamos os nomes das colunas:

[82]:
```
covid_apenas_estados.columns
```

[82]:
```
Index(['regiao', 'estado', 'municipio', 'coduf', 'codmun', 'codRegiaoSaude',
       'nomeRegiaoSaude', 'data', 'semanaEpi', 'populacaoTCU2019',
       'casosAcumulado', 'casosNovos', 'obitosAcumulado', 'obitosNovos',
       'Recuperadosnovos', 'emAcompanhamentoNovos', 'interior/metropolitana'],
      dtype='object')
```

[83]:
```
covid_apenas_estados = covid_apenas_estados[['estado', 'data', 'casosNovos',␣
↪'obitosNovos']]
```

[84]:
```
covid_apenas_estados
```

[84]:

	estado	data	casosNovos	obitosNovos
1668	PB	2020-02-25	0	0
1669	PB	2020-02-26	0	0
1670	PB	2020-02-27	0	0
1671	PB	2020-02-28	0	0
1672	PB	2020-02-29	0	0
…	…	…	…	…
2914	SP	2020-07-08	8657	313
2915	SP	2020-07-09	8350	330
2916	SP	2020-07-10	9395	324
2917	SP	2020-07-11	7780	260
2918	SP	2020-07-12	5107	146

[556 rows x 4 columns]

A data parece ser o índice natural, já que o índice atual não representa nada. Observe que termos índices repetidos, pois teremos as mesmas datas em estados diferentes.

[85]:
```
covid_apenas_estados = covid_apenas_estados.set_index('data')
```

[86]:
```
covid_apenas_estados
```

[86]:

data	estado	casosNovos	obitosNovos
2020-02-25	PB	0	0
2020-02-26	PB	0	0

```
2020-02-27     PB           0            0
2020-02-28     PB           0            0
2020-02-29     PB           0            0
…              …            …            …
2020-07-08     SP        8657          313
2020-07-09     SP        8350          330
2020-07-10     SP        9395          324
2020-07-11     SP        7780          260
2020-07-12     SP        5107          146

[556 rows x 3 columns]
```

1.18 Agrupando com o método *groupby*

Podemos escolher uma (ou mais colunas, incluindo o índice) para agrupar os dados. Ao agruparmos os dados, receberemos um objeto do tipo DataFrameGroupBy. Para vermos os resultados, devemos agregar os valores:

[87]:
```
covid_estados_agrupado = covid_apenas_estados.groupby('estado')
```

[88]:
```
covid_estados_agrupado.sum().rename({'casosNovos':'Casos Totais', 'obitosNovos':
 ↪'Obitos Totais'},axis=1)
```

[88]:
```
        Casos Totais  Obitos Totais
estado
PB             60784           1284
PE             72470           5595
RJ            129684          11415
SP            371997          17848
```

Podemos agrupar por mais de uma coluna. Vamos fazer dois grupos. *grupo_1* formado por PB e PE e *grupo_2* formado por RJ e SP. Em seguida, vamos agrupar por grupo e por data:

[89]:
```
covid_estados_grupos = covid_apenas_estados.copy()
col_grupos = covid_estados_grupos.estado.map(lambda estado: 'grupo_1' if estado␣
 ↪in ['PB','PE']
                                                 else 'grupo_2')
covid_estados_grupos['grupo'] = col_grupos
```

[90]:
```
covid_estados_grupos
```

[90]:
```
           estado  casosNovos  obitosNovos    grupo
data
2020-02-25     PB           0            0  grupo_1
2020-02-26     PB           0            0  grupo_1
2020-02-27     PB           0            0  grupo_1
2020-02-28     PB           0            0  grupo_1
```

```
2020-02-29     PB           0            0  grupo_1
…           …          …          …       …
2020-07-08     SP        8657          313  grupo_2
2020-07-09     SP        8350          330  grupo_2
2020-07-10     SP        9395          324  grupo_2
2020-07-11     SP        7780          260  grupo_2
2020-07-12     SP        5107          146  grupo_2

[556 rows x 4 columns]
```

Agora vamos agrupar e agregar:

[91]:
```
covid_grupo_agrupado = covid_estados_grupos.groupby(['grupo','data'])
```

[92]:
```
covid_grupo_agrupado.sum()
```

[92]:
```
                    casosNovos  obitosNovos
grupo   data
grupo_1 2020-02-25           0            0
        2020-02-26           0            0
        2020-02-27           0            0
        2020-02-28           0            0
        2020-02-29           0            0
…                          …          …
grupo_2 2020-07-08       10900          402
        2020-07-09       10345          475
        2020-07-10       10514          489
        2020-07-11        8012          386
        2020-07-12        5116          155

[278 rows x 2 columns]
```

1.19 Mesclando *DataFrames* (concatenações e *joins*)

Vamos agora ver algumas formas de juntar dois ou mais *DataFrames* com *index* ou colunas em comum para formar um novo *DataFrame*.

Vamos começar vendo concatenações, que nada mais é do que "colar" dois ou mais *DataFrames*. Podemos concatenar por linhas ou por colunas.

A função que realiza a concatenação é **concat**. Os dois argumentos mais utilizados são a lista de *DataFrames* a serem concatenados e **axis**, onde *axis = 0* indica concatenação por linha (um *DataFrame* "embaixo" do outro) e *axis=1* indica concatenação por coluna (um *DataFrame* ao lado do outro).

Relembre do *DataFrame df_dict_series*:

[93]:
```
df_dict_series
```

[93]:

	Idade	Peso	Altura	IMC	Sobrenome
Ana	20	65	162	24.767566	Silva
João	19	80	178	25.249337	PraDo
Maria	21	62	162	23.624447	Sales
Pedro	22	67	165	24.609734	MachadO
Túlio	20	73	171	24.964946	Coutinho

Vamos criar um novo, com novas pessoas:

[94]:
```
serie_Idade_nova = pd.Series({'Augusto':13, 'André': 17, 'Adriana': 31},␣
↪name="Idade")
serie_Peso_novo = pd.Series({'Augusto':75, 'André': 85, 'Adriana': 68},␣
↪name="Peso")
serie_Altura_nova = pd.Series({'Augusto':189, 'André': 175, 'Adriana': 156},␣
↪name="Altura")
serie_sobrenome = pd.Series({'Augusto':'Castro', 'André':'Castro', 'Adriana':
↪'Castro'}, name='Sobrenome')
dicionario_novo = {'Sobrenome':serie_sobrenome, 'Peso': serie_Peso_novo,
                   'Idade': serie_Idade_nova, 'Altura': serie_Altura_nova}
df_novo = pd.DataFrame(dicionario_novo)
df_novo = df_novo.assign(IMC=df_novo.eval('Peso/(Altura/100)**2'))
```

[95]:
```
df_novo
```

[95]:

	Sobrenome	Peso	Idade	Altura	IMC
Augusto	Castro	75	13	189	20.996053
André	Castro	85	17	175	27.755102
Adriana	Castro	68	31	156	27.942143

Agora vamos concatená-los:

[96]:
```
pd.concat([df_dict_series,df_novo], sort=False) # Utilizamos o argumento sort␣
↪pois os DataFrames não
                                                # estavam alinhados
```

[96]:

	Idade	Peso	Altura	IMC	Sobrenome
Ana	20	65	162	24.767566	Silva
João	19	80	178	25.249337	PraDo
Maria	21	62	162	23.624447	Sales
Pedro	22	67	165	24.609734	MachadO
Túlio	20	73	171	24.964946	Coutinho
Augusto	13	75	189	20.996053	Castro
André	17	85	175	27.755102	Castro
Adriana	31	68	156	27.942143	Castro

1.19.1 Concatenando por coluna

Para exemplificar vamos considerar os dados de COVID da Paraíba, selecionando casos novos e óbitos novos, e vamos obter dos dados do Brasil apenas os casos e óbitos diários do país, e vamos concatená-los por coluna.

[97]:
```
covid_PB_casos_obitos = dados_covid_PB[['casosNovos','obitosNovos']]
```

Vamos tratar os dados do Brasil:

[98]:
```
covid_BR_casos_obitos = covid_BR.query('regiao=="Brasil"')
covid_BR_casos_obitos = covid_BR_casos_obitos.set_index('data')
covid_BR_casos_obitos = covid_BR_casos_obitos[['casosNovos','obitosNovos']].
 ↪rename({
    'casosNovos':'casosBR', 'obitosNovos':'obitosBR'
}, axis=1)
```

[99]:
```
covid_PB_casos_obitos
```

[99]:

data	casosNovos	obitosNovos
2020-07-14	1354	40
2020-07-13	324	18
2020-07-12	363	34
2020-07-11	1303	21
2020-07-10	1504	33
…	…	…
2020-03-20	0	0
2020-03-19	0	0
2020-03-18	0	0
2020-03-17	0	0
2020-03-16	0	0

[121 rows x 2 columns]

[100]:
```
covid_BR_casos_obitos
```

[100]:

data	casosBR	obitosBR
2020-02-25	0	0
2020-02-26	1	0
2020-02-27	0	0
2020-02-28	0	0
2020-02-29	1	0
…	…	…
2020-07-08	44571	1223
2020-07-09	42619	1220
2020-07-10	45048	1214

```
2020-07-11    39023      1071
2020-07-12    24831       631

[139 rows x 2 columns]
```

Vamos agora concatená-los por coluna:

[101]:
```
pd.concat([covid_PB_casos_obitos, covid_BR_casos_obitos], axis=1)
```

[101]:
```
            casosNovos  obitosNovos  casosBR  obitosBR
data
2020-02-25         NaN          NaN      0.0       0.0
2020-02-26         NaN          NaN      1.0       0.0
2020-02-27         NaN          NaN      0.0       0.0
2020-02-28         NaN          NaN      0.0       0.0
2020-02-29         NaN          NaN      1.0       0.0
…                  …            …        …         …
2020-07-10      1504.0         33.0  45048.0    1214.0
2020-07-11      1303.0         21.0  39023.0    1071.0
2020-07-12       363.0         34.0  24831.0     631.0
2020-07-13       324.0         18.0      NaN       NaN
2020-07-14      1354.0         40.0      NaN       NaN

[141 rows x 4 columns]
```

Para um polimento final, vamos substituir os valores *NaN* que ocorreram antes do dia 13 de julho por 0. Para tanto, a forma ideal é utilizando o método **map**:

[ ]:
```
```

[102]:
```
dados_PB_BR = pd.concat([covid_PB_casos_obitos, covid_BR_casos_obitos], axis=1)
dados_PB_BR['casosNovos'] = dados_PB_BR.casosNovos.map(lambda caso: 0 if np.
 ↪isnan(caso) else caso).astype('int')
dados_PB_BR['obitosNovos'] = dados_PB_BR.obitosNovos.map(lambda obito: 0 if np.
 ↪isnan(obito) else obito).astype('int')
dados_PB_BR
```

[102]:
```
            casosNovos  obitosNovos  casosBR  obitosBR
data
2020-02-25           0            0      0.0       0.0
2020-02-26           0            0      1.0       0.0
2020-02-27           0            0      0.0       0.0
2020-02-28           0            0      0.0       0.0
2020-02-29           0            0      1.0       0.0
…                    …            …        …         …
2020-07-10        1504           33  45048.0    1214.0
2020-07-11        1303           21  39023.0    1071.0
```

```
2020-07-12         363           34  24831.0     631.0
2020-07-13         324           18      NaN       NaN
2020-07-14        1354           40      NaN       NaN

[141 rows x 4 columns]
```

1.20 Mesclando *DataFrames* através de *joins*

Para realizar *joins* (bastante comuns em *SQL*), iremos utilizar a função **merge** do *pandas*.

joins tomam duas tabelas, uma tabela à esquerda e uma à direita e retornam uma terceira tabela contendo a união das colunas das duas tabelas.

Existem 4 tipos de *joins*:

- *left join*: Apenas irão aparecer os índices (da linha) que existem na tabela à esquerda;
- *right join*: Apenas irão aparecer os índices (da linha) que existem na tabela à direita;
- *inner join*: Apenas irão aparecer os índices que existem nas duas tabelas;
- *full join* ou *outer join*: irão aparecer todos os índices das duas tabelas.

Para exemplificar vou considerar um cenário que ocorreu comigo neste semestre quando tive que realizar um *join* para um dos cursos.

Vamos considerar dois *DataFrames* (aqui teremos menos linhas e nomes e dados fictícios). O primeiro *DataFrame* consistirá de Nomes de alunos, CPF e matrícula da UFPB e recebe o nome de *nome_cpf_mat*. O segundo *DataFrame* consistirá de Nome, CPF e e-mail e recebe o nome de *nome_cpf_email*.

Nosso objetivo é criar um novo *DataFrame* contendo Nome, CPF, matrícula e e-mail.

Temos ainda a seguinte situação:

No *DataFrame nome_cpf_mat* existem alunos que não estão presentes no *nome_cpf_email*, pois não enviaram esta informação.

No *DataFrame nome_cpf_email* existem alunos que não estão presentes no *nome_cpf_mat* pois estes não são alunos da UFPB.

[103]:
```
nome_cpf_mat = pd.read_csv('nome_cpf_mat.csv')
nome_cpf_email = pd.read_csv('nome_cpf_email.csv')
```

Vamos agora dar uma examinada nos *DataFrames*. Como são bem simples, basta realizar *prints* deles.

[104]:
```
nome_cpf_mat
```

[104]:

	Nome	CPF	Matricula
0	João Paulo	326.475.190-99	8848484
1	Ana Silva	073.101.240-22	8451212
2	Antonio Carlos	830.060.930-03	5151213
3	Debora Santos	472.006.460-40	51848484
4	Rodrigo Gomes	566.712.550-16	1415816

```
5    Edson Jardim  308.226.400-07    9592303
```

[105]:
```
nome_cpf_email
```

[105]:
```
             Nome             CPF                      e-mail
0      João Paulo  326.475.190-99      joao@inventado.com.br
1       Ana Silva  073.101.240-22       ana@inventado.com.br
2  Antonio Carlos  830.060.930-03   antonio@inventado.com.br
3    Saulo Santos  370.981.810-99     saulo@inventado.com.br
4   Paulo Cardoso  250.078.710-95     paulo@inventado.com.br
5    Edson Jardim  308.226.400-07     edson@inventado.com.br
6       Ana Silva  344.246.630-00  anasilva@inventado.com.br
```

Tipicamente é bom possuir *index* únicos. Neste sentido, vamos definir o CPF como *index*:

[106]:
```
nome_cpf_mat = nome_cpf_mat.set_index('CPF')
nome_cpf_email = nome_cpf_email.set_index('CPF')
```

Vamos agora realizar um **LEFT** join com o *DataFrame* **nome_cpf_mat** ficando à esquerda (neste caso, apenas alunos com matrícula irão aparecer):

[107]:
```
pd.merge(nome_cpf_mat, nome_cpf_email, how = 'left', on = ['Nome','CPF'])
```

[107]:
```
                          Nome  Matricula                    e-mail
CPF
326.475.190-99      João Paulo    8848484     joao@inventado.com.br
073.101.240-22       Ana Silva    8451212      ana@inventado.com.br
830.060.930-03  Antonio Carlos    5151213  antonio@inventado.com.br
472.006.460-40   Debora Santos   51848484                       NaN
566.712.550-16   Rodrigo Gomes    1415816                       NaN
308.226.400-07    Edson Jardim    9592303    edson@inventado.com.br
```

Na opção *how* dizemos qual o tipo de *join* que queremos realizar.

Na opção *on* dizemos quais as colunas que existem em comum nos *DataFrames*.

Veja o que aconteceria se informássemos apenas que o *CPF* está presente nos dois *DataFrames*:

[108]:
```
pd.merge(nome_cpf_mat, nome_cpf_email, how = 'left', on = 'CPF')
```

[108]:
```
                        Nome_x  Matricula          Nome_y  \
CPF
326.475.190-99      João Paulo    8848484      João Paulo
073.101.240-22       Ana Silva    8451212       Ana Silva
830.060.930-03  Antonio Carlos    5151213  Antonio Carlos
472.006.460-40   Debora Santos   51848484             NaN
566.712.550-16   Rodrigo Gomes    1415816             NaN
308.226.400-07    Edson Jardim    9592303    Edson Jardim
```

```
                                e-mail
CPF
326.475.190-99     joao@inventado.com.br
073.101.240-22      ana@inventado.com.br
830.060.930-03  antonio@inventado.com.br
472.006.460-40                     NaN
566.712.550-16                     NaN
308.226.400-07    edson@inventado.com.br
```

Observe que os nomes dos alunos que estão na segunda tabela ficam indeterminados na coluna *Nome_y*.

Vamos agora realizar um **RIGHT** join com o *DataFrame* **nome_cpf_mat** ficando à esquerda (neste caso, apenas alunos **com e-mail** irão aparecer):

[109]:
```
pd.merge(nome_cpf_mat, nome_cpf_email, how = 'right', on = ['Nome','CPF'])
```

[109]:
```
                         Nome  Matricula                    e-mail
CPF
326.475.190-99      João Paulo  8848484.0      joao@inventado.com.br
073.101.240-22       Ana Silva  8451212.0       ana@inventado.com.br
830.060.930-03  Antonio Carlos  5151213.0   antonio@inventado.com.br
308.226.400-07    Edson Jardim  9592303.0     edson@inventado.com.br
370.981.810-99    Saulo Santos        NaN     saulo@inventado.com.br
250.078.710-95   Paulo Cardoso        NaN     paulo@inventado.com.br
344.246.630-00       Ana Silva        NaN  anasilva@inventado.com.br
```

Vamos agora realizar um **INNER** join com o *DataFrame* **nome_cpf_mat** ficando à esquerda (neste caso, apenas alunos **COM matrícula E COM e-mail** irão aparecer):

[110]:
```
pd.merge(nome_cpf_mat, nome_cpf_email, how = 'inner', on = ['Nome','CPF'])
```

[110]:
```
                         Nome  Matricula                    e-mail
CPF
326.475.190-99      João Paulo    8848484     joao@inventado.com.br
073.101.240-22       Ana Silva    8451212      ana@inventado.com.br
830.060.930-03  Antonio Carlos    5151213  antonio@inventado.com.br
308.226.400-07    Edson Jardim    9592303    edson@inventado.com.br
```

Por fim, vamos agora realizar um **FULL** (ou outer) join com o *DataFrame* **nome_cpf_mat** ficando à esquerda (neste caso, **TODOS** os alunos irão aparecer):

Observação: No *pandas* o *full join* é chamado de *outer join*.

[111]:
```
pd.merge(nome_cpf_mat, nome_cpf_email, how = 'outer', on = ['Nome','CPF'])
```

[111]:
```
                         Nome   Matricula                    e-mail
CPF
326.475.190-99      João Paulo   8848484.0      joao@inventado.com.br
```

```
073.101.240-22       Ana Silva   8451212.0       ana@inventado.com.br
830.060.930-03  Antonio Carlos   5151213.0   antonio@inventado.com.br
472.006.460-40   Debora Santos  51848484.0                       NaN
566.712.550-16   Rodrigo Gomes   1415816.0                       NaN
308.226.400-07    Edson Jardim   9592303.0     edson@inventado.com.br
370.981.810-99    Saulo Santos         NaN     saulo@inventado.com.br
250.078.710-95   Paulo Cardoso         NaN     paulo@inventado.com.br
344.246.630-00       Ana Silva         NaN  anasilva@inventado.com.br
```

1.21 Os métodos *apply*, *map* e *applymap*

A ideia é relativamente simples. Os três métodos são vetorizados e aplicam uma função ou uma substituição via dicionário de tal forma que: * *apply* é realizado via linha ou coluna em um *DataFrame*; * *map* é aplicado a cada elemento de uma *Serie*; * *applymap* é aplicado a cada elemento de um *DataFrame*.

Já vimos diversos exemplos de uso do map. Vejamos exemplos de *applymap* e *apply*.

Exemplo de *applymap*: Neste exemplo vamos retomar a concatenação entre os dados da Paraíba e do Brasil, porém iremos substituir *todos* os valores de *NaN* por zero.

[112]:
```
dados_PB_BR = pd.concat([covid_PB_casos_obitos, covid_BR_casos_obitos], axis=1)
dados_PB_BR.applymap(lambda valor: 0 if np.isnan(valor) else valor)
```

[112]:
```
            casosNovos  obitosNovos  casosBR  obitosBR
data
2020-02-25         0.0          0.0      0.0       0.0
2020-02-26         0.0          0.0      1.0       0.0
2020-02-27         0.0          0.0      0.0       0.0
2020-02-28         0.0          0.0      0.0       0.0
2020-02-29         0.0          0.0      1.0       0.0
…                  …            …        …         …
2020-07-10      1504.0         33.0  45048.0    1214.0
2020-07-11      1303.0         21.0  39023.0    1071.0
2020-07-12       363.0         34.0  24831.0     631.0
2020-07-13       324.0         18.0      0.0       0.0
2020-07-14      1354.0         40.0      0.0       0.0

[141 rows x 4 columns]
```

Exemplo de *apply*. Vamos utilizar o *apply* para realizar a soma de casos e óbitos de mais uma forma diferente:

[113]:
```
dados_PB_BR.applymap(lambda valor: 0 if np.isnan(valor) else valor).
↪apply(lambda x: np.sum(x)).astype('int')
```

[113]:
```
casosNovos       62462
obitosNovos       1342
casosBR        1864681
```

```
obitosBR         72100
dtype: int32
```

Se quisermos realizar a operação por linhas, basta utilizar o argumento *axis=1*:

[114]:
```
dados_PB_BR.applymap(
    lambda valor: 0 if np.isnan(valor) else valor
).apply(lambda x: (x>0).all(), axis=1).astype('int')
```

[114]:
```
data
2020-02-25    0
2020-02-26    0
2020-02-27    0
2020-02-28    0
2020-02-29    0
              ..
2020-07-10    1
2020-07-11    1
2020-07-12    1
2020-07-13    0
2020-07-14    0
Freq: D, Length: 141, dtype: int32
```