Introdução ao Pandas

Séries

1 - Importar o Pandas

In [1]:
```
import pandas as pd
pd.Series?
```

2 - Lista de strings em Série

In [2]:
```
animals = ['Gato', 'Cachorro', 'Rato']
pd.Series(animals)
```

Out[2]:
```
0        Gato
1    Cachorro
2        Rato
dtype: object
```

3 - Lista de inteiros em Série

In [3]:
```
numbers = [1,2, None]
pd.Series(numbers)
```

Out[3]:
```
0    1.0
1    2.0
2    NaN
dtype: float64
```

4 - None vs. NAN

In [4]:
```
import numpy as np

animals = ['Tiger', 'Bear', None]
pd.Series(animals)
```

Out[4]:
```
0    Tiger
1     Bear
2     None
dtype: object
```

In [5]:
```
numbers = [1, 2, None]
pd.Series(numbers)
```

Out[5]:
```
0    1.0
1    2.0
2    NaN
dtype: float64
```

5 - NAN não é None

In [6]:
```
import numpy as np
np.nan == None
```

Out[6]:
```
False
```

6 - NAN == NAN é falso

In [7]:
```
np.nan == np.nan
```

Out[7]:
```
False
```

7 - É NAN?

In [8]:
```
np.isnan(np.nan)
```

Out[8]:
```
True
```

8 - Séries a partir de dicionários

In [9]:
```
sports = {'Archery': 'Bhutan',
          'Golf': 'Scotland',
          'Sumo': 'Japan',
          'Taekwondo': 'South Korea'}
s = pd.Series(sports)
s
```

Out[9]:
```
Archery           Bhutan
Golf            Scotland
Sumo               Japan
Taekwondo    South Korea
dtype: object
```

9 - Objeto do índice

In [10]:
```
s.index
```

Out[10]:
```
Index(['Archery', 'Golf', 'Sumo', 'Taekwondo'], dtype='object')
```

10 - Índice como uma lista explícita para a série

In [11]:
```
s = pd.Series(['Gato', 'Cachorro', 'Rato'], index = ['Vitor', 'Pedro', 'Santiago'])
s
```

Out[11]:
```
Vitor           Gato
Pedro       Cachorro
Santiago        Rato
dtype: object
```

11 - Índice como uma lista explícita para a série criada a partir de um dicionário

In [12]:
```
sports = {'Archery': 'Bhutan',
          'Golf': 'Scotland',
          'Sumo': 'Japan',
          'Taekwondo': 'South Korea'}
s = pd.Series(sports, index=['Golf', 'Sumo', 'Hockey'])
s
```

Out[12]:
```
Golf      Scotland
Sumo         Japan
Hockey         NaN
dtype: object
```

Consultando Séries

12 - Lista dos esportes em nosso índice e uma lista de países como valores. Criamos um dicionário e assim uma série.

In [13]:
```
sports = {'Archery': 'Bhutan',
          'Golf': 'Scotland',
          'Sumo': 'Japan',
          'Taekwondo': 'South Korea'}
s = pd.Series(sports)
s
```

Out[13]:
```
Archery           Bhutan
Golf            Scotland
Sumo               Japan
Taekwondo    South Korea
dtype: object
```

13 - Para ver o quarto país usamos o atributo iloc com o parâmetro 3. Para ver qual país tem o golfe como seu esporte nacional, utilizamos o atributo loc com um parâmetro golfe.

In [14]:
```
s.iloc[3]
```

Out[14]:
```
'South Korea'
```

In [15]:
```
s.loc['Golf']
```

Out[15]:
```
'Scotland'
```

14 - Possibilidade de usar o operador de indexação diretamente na série em si.

In [16]:
```
s[3]
```

Out[16]:
```
'South Korea'
```

In [17]:
```
s['Golf']
```

Out[17]:
```
'Scotland'
```

15 - A opção mais segura é ser explícito e usar os atributos iloc ou loc diretamente. Por exemplo....

In [18]:
```
sports = {99: 'Bhutan',
          100: 'Scotland',
          101: 'Japan',
          102: 'South Korea'}
s = pd.Series(sports)
```

In [26]:
```
# s[0]
```

16 - Rotina que itera sobre todos os itens da série, somando-os para obter um total.

In [27]:
```
s = pd.Series([100.0, 120.0, 101.0, 3.00])
s
```

Out[27]:
```
0    100.0
1    120.0
2    101.0
3      3.0
dtype: float64
```

In [28]:
```
total = 0
for item in s:
    total += item
print(total)
```

```
324.0
```

17 - Vetorização Numpy

In [29]:
```
total = np.sum(s)
print (total)
```

```
324.0
```

18 - Criar uma grande série de números aleatórios.

In [30]:
```
s = pd.Series(np.random.randint(0,1000,10000))
s.head()
```

Out[30]:
```
0    106
1      7
2    780
3    646
4    743
dtype: int64
```

In [31]:
```
len(s)
```

Out[31]:
```
10000
```

19 - "timeit" com nosso código iterativo sem o Numpy.

In [32]:
```
%%timeit -n 100
total = 0
for item in s:
    total += item
```

```
1.63 ms ± 93.9 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

20 - Agora vamos com a vetorização do Numpy.

In [33]:
```
%%timeit -n 100
total = np.sum(s)
```

```
117 µs ± 48 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

21 - Broadcasting e head()

In [34]:
```
s[0:1]+=2
s.head()
```

Out[34]:
```
0    108
1      7
```

```
2    780
3    646
4    743
dtype: int64
```

22 - O Pandas permite a iteração por meio de uma série assim como num dicionário.

In [126…
```
for label, value in s.iteritems():
    s.set_value(label, value+2)
s.head()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/var/folders/01/_r7b02r11p15j0s54gb9x0040000gn/T/ipykernel_2417/812083786.py in <module>
      1 for label, value in s.iteritems():
----> 2     s.set_value(label, value+2)
      3 s.head()

~/opt/anaconda3/envs/ml-impa/lib/python3.8/site-packages/pandas/core/generic.py in __getat
tr__(self, name)
   5485         ):
   5486             return self[name]
-> 5487         return object.__getattribute__(self, name)
   5488
   5489     def __setattr__(self, name: str, value) -> None:

AttributeError: 'Series' object has no attribute 'set_value'
```

23 - Vamos cronometrar as duas abordagens.

In [75]:
```
%%timeit -n 1
s = pd.Series(np.random.randint(0,1000,10000))
for label, value in s.iteritems():
    s.loc[label]= value+2
```

```
550 ms ± 63 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [76]:
```
%%timeit -n 1
s = pd.Series(np.random.randint(0,1000,10000))
s+=2
```

```
The slowest run took 4.09 times longer than the fastest. This could mean that an intermedi
ate result is being cached.
512 µs ± 303 µs per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

24 - Tipos misturados para os valores de dados ou índice de rótulos não são problema para o Pandas.

In [77]:
```
s = pd.Series([1,2,3])
s.loc['Animal'] = "Gato"
s
```

Out[77]:
```
0           1
1           2
2           3
Animal    Gato
dtype: object
```

25 . Valores dos índices não são sempre exclusivos.

In [78]:
```
original_sports = pd.Series({'Archery': 'Bhutan',
                             'Golf': 'Scotland',
```

```
                                'Sumo': 'Japan',
                                'Taekwondo': 'South Korea'})
cricket_loving_countries = pd.Series(['Australia',
                                      'Barbados',
                                      'Pakistan',
                                      'England'],
                                     index=['Cricket',
                                            'Cricket',
                                            'Cricket',
                                            'Cricket'])
all_countries = original_sports.append(cricket_loving_countries)
```

26 - O método append na verdade não muda a série subjacente. Em vez disso, ele retorna uma nova série que é composta das duas juntas.

In [79]:
```
original_sports
```

Out[79]:
```
Archery           Bhutan
Golf            Scotland
Sumo               Japan
Taekwondo    South Korea
dtype: object
```

In [80]:
```
cricket_loving_countries
```

Out[80]:
```
Cricket    Australia
Cricket     Barbados
Cricket     Pakistan
Cricket      England
dtype: object
```

In [81]:
```
all_countries
```

Out[81]:
```
Archery           Bhutan
Golf            Scotland
Sumo               Japan
Taekwondo    South Korea
Cricket        Australia
Cricket         Barbados
Cricket         Pakistan
Cricket          England
dtype: object
```

In [82]:
```
all_countries.loc['Cricket']
```

Out[82]:
```
Cricket    Australia
Cricket     Barbados
Cricket     Pakistan
Cricket      England
dtype: object
```

DataFrame

27 - Vamos colocar esas séries no DataFrame como sendo o primeiro argumento e atribuir aos valores de índices quais foram as lojas onde as compras foram feitas.

In [83]:
```
import pandas as pd
purchase_1 = pd.Series({'Name': 'Chris',
                        'Item Purchased': 'Dog Food',
                        'Cost': 22.50})
```

```
purchase_2 = pd.Series({'Name': 'Kevyn',
                        'Item Purchased': 'Kitty Litter',
                        'Cost': 2.50})
purchase_3 = pd.Series({'Name': 'Vinod',
                        'Item Purchased': 'Bird Seed',
                        'Cost': 5.00})
df = pd.DataFrame([purchase_1, purchase_2, purchase_3], index=['Store 1', 'Store 1', 'Stor

df.head()
```

Out[83]:

	Name	Item Purchased	Cost
Store 1	Chris	Dog Food	22.5
Store 1	Kevyn	Kitty Litter	2.5
Store 2	Vinod	Bird Seed	5.0

28 - Se quisermos selecionar os dados associados à loja 'Store 2', apenas consultamos o atributo "loc" com um único parâmetro.

In [84]:
```
df.loc['Store 2']
```

Out[84]:
```
Name                  Vinod
Item Purchased    Bird Seed
Cost                    5.0
Name: Store 2, dtype: object
```

In [85]:
```
type(df.loc['Store 2'])
```

Out[85]:
```
pandas.core.series.Series
```

29 - Se consultarmos os registros da loja 'Store 1', veremos que Chris e Kevin, ambos compraram na mesma loja de suprimentos.

In [86]:
```
df.loc['Store 1']
```

Out[86]:

	Name	Item Purchased	Cost
Store 1	Chris	Dog Food	22.5
Store 1	Kevyn	Kitty Litter	2.5

30 - Se quisermos apenas listar os custos da 'Store 1', bastaria fornecer dois parâmetros para df.loc, um deles o índice da linha e o outro, o nome da coluna.

In [87]:
```
df.loc['Store 1', 'Cost']
```

Out[87]:
```
Store 1    22.5
Store 1     2.5
Name: Cost, dtype: float64
```

31 - Podemos transpôr o DataFrame, usando o atributo T maiúsculo, ele troca as linhas por colunas e vice-versa. Depois podemos usar o método '.loc' Não é muito elegante!

In [88]:
```
df.T
```

Out[88]:

	Store 1	Store 1	Store 2

	Store 1	Store 1	Store 2
Name	Chris	Kevyn	Vinod
Item Purchased	Dog Food	Kitty Litter	Bird Seed
Cost	22.5	2.5	5.0

In [89]:
```
df.T.loc['Cost']
```

Out[89]:
```
Store 1    22.5
Store 1     2.5
Store 2     5.0
Name: Cost, dtype: object
```

32 - Operador de indexação diretamente no DataFrame para seleção de coluna.

In [90]:
```
df['Cost']
```

Out[90]:
```
Store 1    22.5
Store 1     2.5
Store 2     5.0
Name: Cost, dtype: float64
```

33 - Encadeamento (Cuidado)

In [91]:
```
df.loc['Store 1']['Cost']
```

Out[91]:
```
Store 1    22.5
Store 1     2.5
Name: Cost, dtype: float64
```

34 - Fatiamento no Dataframe.

In [92]:
```
df.loc[:,['Name', 'Cost']]
```

Out[92]:

	Name	Cost
Store 1	Chris	22.5
Store 1	Kevyn	2.5
Store 2	Vinod	5.0

35 - Deletar dados de séries e de DataFrames. A função drop não altera o DataFrame por padrão. Ao invés disso, lhe retorna uma cópia do DataFrame com as linhas informadas removidas. Podemos ver que o nosso DataFrame original ainda está intacto.

In [93]:
```
df.drop('Store 1')
```

Out[93]:

	Name	Item Purchased	Cost
Store 2	Vinod	Bird Seed	5.0

In [94]:
```
df
```

Out[94]:

	Name	Item Purchased	Cost

	Name	Item Purchased	Cost
Store 1	Chris	Dog Food	22.5
Store 1	Kevyn	Kitty Litter	2.5
Store 2	Vinod	Bird Seed	5.0

36 - Fazendo cópias do dataframe.

In [95]:
```
copy_df = df.copy()
copy_df = copy_df.drop('Store 1')
copy_df
```

Out[95]:

	Name	Item Purchased	Cost
Store 2	Vinod	Bird Seed	5.0

37 - O drop tem dois interessantes parâmetros opcionais.

In [96]:
```
df.drop?
```

38 - Outra forma de deletar.

In [97]:
```
del df['Name']
df
```

Out[97]:

	Item Purchased	Cost
Store 1	Dog Food	22.5
Store 1	Kitty Litter	2.5
Store 2	Bird Seed	5.0

39 - Adicionar uma nova coluna ao DataFrame.

In [98]:
```
df['Location'] = None
df
```

Out[98]:

	Item Purchased	Cost	Location
Store 1	Dog Food	22.5	None
Store 1	Kitty Litter	2.5	None
Store 2	Bird Seed	5.0	None

Dataframes - Indexando & Carregando

40 - Podemos criar uma série baseada apenas na categoria preço, usando colchetes.

In [99]:
```
costs = df['Cost']
costs
```

Out[99]:
```
Store 1    22.5
Store 1     2.5
```

```
Store 2     5.0
Name: Cost, dtype: float64
```

41 - Broadcasting

In [100…
```
costs+=2
costs
```

Out[100…
```
Store 1    24.5
Store 1     4.5
Store 2     7.0
Name: Cost, dtype: float64
```

In [101…
```
df
```

Out[101…

	Item Purchased	Cost	Location
Store 1	Dog Food	24.5	None
Store 1	Kitty Litter	4.5	None
Store 2	Bird Seed	7.0	None

42 - Carregando arquivo CSV - dados das olimpíadas.

In [102…
```
df = pd.read_csv('./Data/olympics.csv')
df.head()
```

Out[102…

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	NaN	№ Summer	01 !	02 !	03 !	Total	№ Winter	01 !	02 !	03 !	Total	№ Games	01 !	02 !	03 !	Combined total
1	Afghanistan (AFG)	13	0	0	2	2	0	0	0	0	0	13	0	0	2	2
2	Algeria (ALG)	12	5	2	8	15	3	0	0	0	0	15	5	2	8	15
3	Argentina (ARG)	23	18	24	28	70	18	0	0	0	0	41	18	24	28	70
4	Armenia (ARM)	5	1	2	9	12	6	0	0	0	0	11	1	2	9	12

43 - Ignorar a primeira linha, que era composta por nomes das colunas numéricas.

In [103…
```
df = pd.read_csv('./Data/olympics.csv', index_col = 0, skiprows=1)
df.head()
```

Out[103…

	№ Summer	01 !	02 !	03 !	Total	№ Winter	01 !.1	02 !.1	03 !.1	Total.1	№ Games	01 !.2	02 !.2	03 !.2	Combined total
Afghanistan (AFG)	13	0	0	2	2	0	0	0	0	0	13	0	0	2	2
Algeria (ALG)	12	5	2	8	15	3	0	0	0	0	15	5	2	8	15
Argentina (ARG)	23	18	24	28	70	18	0	0	0	0	41	18	24	28	70
Armenia (ARM)	5	1	2	9	12	6	0	0	0	0	11	1	2	9	12
Australasia (ANZ) [ANZ]	2	3	4	5	12	0	0	0	0	0	2	3	4	5	12

44 - Pandas armazena uma lista de todas as colunas no atributo ".columns".

In [104…
```
df.columns
```

Out[104…
```
Index(['№ Summer', '01 !', '02 !', '03 !', 'Total', '№ Winter', '01 !.1',
       '02 !.1', '03 !.1', 'Total.1', '№ Games', '01 !.2', '02 !.2', '03 !.2',
       'Combined total'],
      dtype='object')
```

45 - "inplace" para true, então o Pandas atualiza este Dataframe diretamente.

In [105…
```
for col in df.columns:
    if col[:2]=='01':
        df.rename(columns={col:'Gold' + col[4:]}, inplace=True)
    if col[:2]=='02':
        df.rename(columns={col:'Silver' + col[4:]}, inplace=True)
    if col[:2]=='03':
        df.rename(columns={col:'Bronze' + col[4:]}, inplace=True)
    if col[:1]=='№':
        df.rename(columns={col:'#' + col[1:]}, inplace=True)

df.head()
```

Out[105…

	# Summer	Gold	Silver	Bronze	Total	# Winter	Gold.1	Silver.1	Bronze.1	Total.1	# Games	C
Afghanistan (AFG)	13	0	0	2	2	0	0	0	0	0	13	
Algeria (ALG)	12	5	2	8	15	3	0	0	0	0	15	
Argentina (ARG)	23	18	24	28	70	18	0	0	0	0	41	
Armenia (ARM)	5	1	2	9	12	6	0	0	0	0	11	
Australasia (ANZ) [ANZ]	2	3	4	5	12	0	0	0	0	0	2	

Consultando DataFrames

46 - Mascaramento Booleano.

In [106…
```
df['Gold'] > 0
```

Out[106…
```
Afghanistan (AFG)                          False
Algeria (ALG)                               True
Argentina (ARG)                             True
Armenia (ARM)                               True
Australasia (ANZ) [ANZ]                     True
                                           ...
Independent Olympic Participants (IOP) [IOP]    False
Zambia (ZAM) [ZAM]                         False
Zimbabwe (ZIM) [ZIM]                        True
Mixed team (ZZX) [ZZX]                      True
Totals                                      True
Name: Gold, Length: 147, dtype: bool
```

47 - Sobrepor esta máscara no data frame.

In [107…
```
only_gold = df.where(df['Gold'] > 0)
only_gold
```

Out[107…

	# Summer	Gold	Silver	Bronze	Total	# Winter	Gold.1	Silver.1	Bronze.1	Total.1	Gam

	# Summer	Gold	Silver	Bronze	Total	# Winter	Gold.1	Silver.1	Bronze.1	Total.1	Gam
Afghanistan (AFG)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
Algeria (ALG)	12.0	5.0	2.0	8.0	15.0	3.0	0.0	0.0	0.0	0.0	1
Argentina (ARG)	23.0	18.0	24.0	28.0	70.0	18.0	0.0	0.0	0.0	0.0	4
Armenia (ARM)	5.0	1.0	2.0	9.0	12.0	6.0	0.0	0.0	0.0	0.0	1
Australasia (ANZ) [ANZ]	2.0	3.0	4.0	5.0	12.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	
Independent Olympic Participants (IOP) [IOP]	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
Zambia (ZAM) [ZAM]	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
Zimbabwe (ZIM) [ZIM]	12.0	3.0	4.0	1.0	8.0	1.0	0.0	0.0	0.0	0.0	1
Mixed team (ZZX) [ZZX]	3.0	8.0	5.0	4.0	17.0	0.0	0.0	0.0	0.0	0.0	
Totals	27.0	4809.0	4775.0	5130.0	14714.0	22.0	959.0	958.0	948.0	2865.0	4

147 rows × 15 columns

48 - Existem 100 países que tiveram medalhas de ouro nos jogos de verão, há 147 países no total.

In [108…
```
only_gold['Gold'].count()
```

Out[108…
```
100
```

In [109…
```
df['Gold'].count()
```

Out[109…
```
147
```

49 - Muitas vezes queremos jogar fora as linhas que não têm nenhum dado. Para isso, usar a função dropna.

In [110…
```
only_gold = only_gold.dropna()
only_gold
```

Out[110…

	# Summer	Gold	Silver	Bronze	Total	# Winter	Gold.1	Silver.1	Bronze.1	Total.1	Gam
Algeria (ALG)	12.0	5.0	2.0	8.0	15.0	3.0	0.0	0.0	0.0	0.0	15
Argentina (ARG)	23.0	18.0	24.0	28.0	70.0	18.0	0.0	0.0	0.0	0.0	41
Armenia (ARM)	5.0	1.0	2.0	9.0	12.0	6.0	0.0	0.0	0.0	0.0	11
Australasia (ANZ) [ANZ]	2.0	3.0	4.0	5.0	12.0	0.0	0.0	0.0	0.0	0.0	2
Australia (AUS) [AUS] [Z]	25.0	139.0	152.0	177.0	468.0	18.0	5.0	3.0	4.0	12.0	43

	# Summer	Gold	Silver	Bronze	Total	# Winter	Gold.1	Silver.1	Bronze.1	Total.1	Game
...	...	...	...	...	...	...	...	...	...	...	
Venezuela (VEN)	17.0	2.0	2.0	8.0	12.0	4.0	0.0	0.0	0.0	0.0	21
Yugoslavia (YUG) [YUG]	16.0	26.0	29.0	28.0	83.0	14.0	0.0	3.0	1.0	4.0	30
Zimbabwe (ZIM) [ZIM]	12.0	3.0	4.0	1.0	8.0	1.0	0.0	0.0	0.0	0.0	13
Mixed team (ZZX) [ZZX]	3.0	8.0	5.0	4.0	17.0	0.0	0.0	0.0	0.0	0.0	3
Totals	27.0	4809.0	4775.0	5130.0	14714.0	22.0	959.0	958.0	948.0	2865.0	49

100 rows × 15 columns

50 - Não precisamos, de fato, usar a função where explicitamente.

In [111…

```
only_gold = df[df['Gold'] > 0]
only_gold
```

Out[111…

	# Summer	Gold	Silver	Bronze	Total	# Winter	Gold.1	Silver.1	Bronze.1	Total.1	# Games
Algeria (ALG)	12	5	2	8	15	3	0	0	0	0	15
Argentina (ARG)	23	18	24	28	70	18	0	0	0	0	41
Armenia (ARM)	5	1	2	9	12	6	0	0	0	0	11
Australasia (ANZ) [ANZ]	2	3	4	5	12	0	0	0	0	0	2
Australia (AUS) [AUS] [Z]	25	139	152	177	468	18	5	3	4	12	43
...	...	...	...	...	...	...	...	...	...	...	...
Venezuela (VEN)	17	2	2	8	12	4	0	0	0	0	21
Yugoslavia (YUG) [YUG]	16	26	29	28	83	14	0	3	1	4	30
Zimbabwe (ZIM) [ZIM]	12	3	4	1	8	1	0	0	0	0	13
Mixed team (ZZX) [ZZX]	3	8	5	4	17	0	0	0	0	0	3
Totals	27	4809	4775	5130	14714	22	959	958	948	2865	49

100 rows × 15 columns

51 - 101 países ganharam uma medalha de ouro em algum momento.

In [112…

```
len(df[(df['Gold'] > 0) | (df['Gold.1']>0)])
```

Out[112…

```
101
```

52 - Há algum país que mais de uma medalha de ouro nos jogos Olímpicos de inverno e nenhuma nos jogos Olímpicos de verão.

In [113…
```
df[(df['Gold.1'] > 0) & (df['Gold'] == 0)]
```

Out[113…

	# Summer	Gold	Silver	Bronze	Total	# Winter	Gold.1	Silver.1	Bronze.1	Total.1	# Games	G
Liechtenstein (LIE)	16	0	0	0	0	18	2	2	5	9	34	

Indexando Dataframes

53 - Digamos que não queremos indexar o DataFrame por países, mas em vez disso, indexar pelo número de medalhas de ouro que foram conquistadas nos jogos de verão.

In [114…
```
df['country'] = df.index
df = df.set_index('Gold')
df.head()
```

Out[114…

Gold	# Summer	Silver	Bronze	Total	# Winter	Gold.1	Silver.1	Bronze.1	Total.1	# Games	Gold.2	Silver.2	Bro
0	13	0	2	2	0	0	0	0	0	13	0	0	
5	12	2	8	15	3	0	0	0	0	15	5	2	
18	23	24	28	70	18	0	0	0	0	41	18	24	
1	5	2	9	12	6	0	0	0	0	11	1	2	
3	2	4	5	12	0	0	0	0	0	2	3	4	

54 - Podemos nos livrar completamente do índice chamando a função reset_index. Ela torna o índice uma coluna e cria um índice default numerado.

In [115…
```
df = df.reset_index()
df.head()
```

Out[115…

	Gold	# Summer	Silver	Bronze	Total	# Winter	Gold.1	Silver.1	Bronze.1	Total.1	# Games	Gold.2	Silver.2
0	0	13	0	2	2	0	0	0	0	0	13	0	0
1	5	12	2	8	15	3	0	0	0	0	15	5	2
2	18	23	24	28	70	18	0	0	0	0	41	18	24
3	1	5	2	9	12	6	0	0	0	0	11	1	2
4	3	2	4	5	12	0	0	0	0	0	2	3	4

55 - Carregar o arquivo CSV com dados Censo dos EUA.

In [116…
```
df = pd.read_csv('./Data/census.csv')
df.head()
```

Out[116…

	SUMLEV	REGION	DIVISION	STATE	COUNTY	STNAME	CTYNAME	CENSUS2010POP	ESTIMATESBASE2C
0	40	3	6	1	0	Alabama	Alabama	4779736	4780

	SUMLEV	REGION	DIVISION	STATE	COUNTY	STNAME	CTYNAME	CENSUS2010POP	ESTIMATESBASE2C
1	50	3	6	1	1	Alabama	Autauga County	54571	545
2	50	3	6	1	3	Alabama	Baldwin County	182265	1822
3	50	3	6	1	5	Alabama	Barbour County	27457	274
4	50	3	6	1	7	Alabama	Bibb County	22915	229

5 rows × 100 columns

56 - Ver uma lista de todos os valores exclusivos de uma determinada coluna.

In [117…
```
df['SUMLEV'].unique()
```

Out[117…
```
array([40, 50])
```

57 - Mantendo dados sobre os condados.

In [118…
```
df=df[df['SUMLEV'] == 50]
df.head()
```

Out[118…

	SUMLEV	REGION	DIVISION	STATE	COUNTY	STNAME	CTYNAME	CENSUS2010POP	ESTIMATESBASE2C
1	50	3	6	1	1	Alabama	Autauga County	54571	545
2	50	3	6	1	3	Alabama	Baldwin County	182265	1822
3	50	3	6	1	5	Alabama	Barbour County	27457	274
4	50	3	6	1	7	Alabama	Bibb County	22915	229
5	50	3	6	1	9	Alabama	Blount County	57322	573

5 rows × 100 columns

58 - Reduzir nossa análise de dados para apenas a estimativa de população total e para o total de nascimentos.

In [119…
```
columns_to_keep = ['STNAME',
                   'CTYNAME',
                   'BIRTHS2010',
                   'BIRTHS2011',
                   'BIRTHS2012',
                   'BIRTHS2013',
                   'BIRTHS2014',
                   'BIRTHS2015',
                   'POPESTIMATE2010',
                   'POPESTIMATE2011',
                   'POPESTIMATE2012',
                   'POPESTIMATE2013',
                   'POPESTIMATE2014',
                   'POPESTIMATE2015']
```

```
df = df[columns_to_keep]
df.head()
```

Out[119…

	STNAME	CTYNAME	BIRTHS2010	BIRTHS2011	BIRTHS2012	BIRTHS2013	BIRTHS2014	BIRTHS2015	PC
1	Alabama	Autauga County	151	636	615	574	623	600	
2	Alabama	Baldwin County	517	2187	2092	2160	2186	2240	
3	Alabama	Barbour County	70	335	300	283	260	269	
4	Alabama	Bibb County	44	266	245	259	247	253	
5	Alabama	Blount County	183	744	710	646	618	603	

59 - Note que temos aqui um índice dual, primeiro o nome do estado e em seguida o nome do condado.

In [120…

```
df = df.set_index(['STNAME', 'CTYNAME'])
df.head()
```

Out[120…

		BIRTHS2010	BIRTHS2011	BIRTHS2012	BIRTHS2013	BIRTHS2014	BIRTHS2015	POPES
STNAME	**CTYNAME**							
Alabama	**Autauga County**	151	636	615	574	623	600	
	Baldwin County	517	2187	2092	2160	2186	2240	
	Barbour County	70	335	300	283	260	269	
	Bibb County	44	266	245	259	247	253	
	Blount County	183	744	710	646	618	603	

60 - Se quisermos ver os resultados da população do Condado de Washtenaw.

In [121…

```
df.loc['Alabama', 'Autauga County']
```

Out[121…

```
BIRTHS2010           151
BIRTHS2011           636
BIRTHS2012           615
BIRTHS2013           574
BIRTHS2014           623
BIRTHS2015           600
POPESTIMATE2010    54660
POPESTIMATE2011    55253
POPESTIMATE2012    55175
POPESTIMATE2013    55038
POPESTIMATE2014    55290
POPESTIMATE2015    55347
Name: (Alabama, Autauga County), dtype: int64
```

61 - Comparar dois condados.

In [122…
```
df.loc[ [('Alabama', 'Autauga County'),
         ('Alabama', 'Baldwin County')] ]
```

Out[122…

STNAME	CTYNAME	BIRTHS2010	BIRTHS2011	BIRTHS2012	BIRTHS2013	BIRTHS2014	BIRTHS2015	POPES
Alabama	**Autauga County**	151	636	615	574	623	600	
	Baldwin County	517	2187	2092	2160	2186	2240	

In [ ]: