Pandas

1 Introdução aos Pandas

Nesta seção do curso, aprenderemos a usar pandas para análise de dados. Você deve enxergar o pandas como uma versão extremamente poderosa do Excel, com muito mais recursos. Nesta seção do curso, você deve passar pelos notebooks nesta ordem:

- Introdução aos Pandas
- Series
- DataFrames
- Dados ausentes
- GroupBy
- Mesclar,Juntar, e Concatenar
- Operações
- Entrada e saída de dados

2 Series

O primeiro tipo de dado que aprenderemos é a Serie. Vamos importar Pandas e explorar tal objeto.

A Serie é muito semelhante a uma matriz NumPy (na verdade, ela é construída em cima do objeto de matriz NumPy). O que diferencia a matriz NumPy de uma Série, é que uma Serie pode ter rótulos de eixos, o que significa que pode ser indexado por um rótulo, em vez de apenas uma localização numérica. Também não precisa manter dados numéricos, ele pode conter qualquer objeto Python arbitrário.

Vamos explorar este conceito através de alguns exemplos:

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

2.0.1 Criando uma Serie

Você pode converter uma lista, numpy array ou dicionário para uma série:

[ ]:
```
labels = ['a','b','c']
minha_lista = [10,20,30]
arr = np.array([10,20,30])
d = {'a':10,'b':20,'c':30}
```

** Usando listas **

[ ]:
```
pd.Series(data=minha_lista)
```

[ ]:
```
0    10
1    20
2    30
dtype: int64
```

[ ]:
```
pd.Series(data=minha_lista,index=labels)
```

[ ]:
```
a    10
b    20
c    30
dtype: int64
```

[ ]:
```
pd.Series(minha_lista,labels)
```

[ ]:
```
a    10
b    20
c    30
dtype: int64
```

** NumPy Arrays **

[ ]:
```
pd.Series(arr)
```

[ ]:
```
0    10
1    20
2    30
dtype: int64
```

[ ]:
```
pd.Series(arr,labels)
```

[ ]:
```
a    10
b    20
c    30
dtype: int64
```

** Dicionários **

[ ]:
```
pd.Series(d)
```

[ ]:
```
a    10
b    20
c    30
dtype: int64
```

2.0.2 Dados nas Series

Uma série de pandas pode conter uma variedade de tipos de objeto:

[ ]:
```
pd.Series(data=labels)
```

[ ]:
```
0    a
1    b
2    c
dtype: object
```

[ ]:
```
# Mesmo funções (embora seja improvável que você use isso)
pd.Series([sum,print,len])
```

[ ]:
```
0      <built-in function sum>
1    <built-in function print>
2      <built-in function len>
dtype: object
```

2.1 Usando um Índice

A chave para usar uma Serie é entender seu índice. O Pandas faz uso desses nomes ou números de índice, permitindo pesquisas rápidas de informações (funciona como uma tabela de hash ou dicionário).

Vamos ver alguns exemplos de como pegar informações de uma Serie. Vamos criar duas Series, ser1 e ser2:

[ ]:
```
ser1 = pd.Series([1,2,3,4],index = ['EUA', 'Alemanha','USSR', 'Japão'])
```

[ ]:
```
ser1
```

[ ]:
```
EUA         1
Alemanha    2
USSR        3
Japão       4
dtype: int64
```

[ ]:
```
ser2 = pd.Series([1,2,5,4],index = ['EUA', 'Alemanha','Italia', 'Japão'])
```

[ ]:
```
ser2
```

[ ]:
```
EUA         1
Alemanha    2
Italia      5
Japão       4
dtype: int64
```

[ ]:
```
ser1['EUA']
```

[ ]:
```
1
```

[ ]:
```
type(ser1)
```

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

[ ]:
```
type(ser1['EUA'])
```

[ ]:
```
numpy.int64
```

As operações também são feitas com base no índice:

[ ]:
```
ser1
```

[ ]:
```
EUA         1
Alemanha    2
USSR        3
Japão       4
dtype: int64
```

[ ]:
```
ser2
```

[ ]:
```
EUA         1
Alemanha    2
Italia      5
Japão       4
dtype: int64
```

[ ]:
```
ser1 + ser2
```

[ ]:
```
Alemanha    4.0
EUA         2.0
Italia      NaN
Japão       8.0
USSR        NaN
dtype: float64
```

Vamos parar aqui por enquanto e passar para a DataFrames, que expandirá o conceito da Serie!

---

3 DataFrame - Criação e Fatiamento

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

[ ]:
```
np.random.seed(101)
```

```
[ ]: df = pd.DataFrame(np.random.randn(5,4), index='A B C D E'.split(), columns='W X␣
     ↪Y Z'.split())
```

```
[ ]: df
```

```
[ ]:           W         X         Y         Z
     A  2.706850  0.628133  0.907969  0.503826
     B  0.651118 -0.319318 -0.848077  0.605965
     C -2.018168  0.740122  0.528813 -0.589001
     D  0.188695 -0.758872 -0.933237  0.955057
     E  0.190794  1.978757  2.605967  0.683509
```

```
[ ]: df['W']
```

```
[ ]: A    2.706850
     B    0.651118
     C   -2.018168
     D    0.188695
     E    0.190794
     Name: W, dtype: float64
```

```
[ ]: type(df)
```

```
[ ]: pandas.core.frame.DataFrame
```

```
[ ]: type(df['W'])
```

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

```
[ ]: type(df)
```

```
[ ]: pandas.core.frame.DataFrame
```

```
[ ]: df[['W', 'Z']]
```

```
[ ]:           W         Z
     A  2.706850  0.503826
     B  0.651118  0.605965
     C -2.018168 -0.589001
     D  0.188695  0.955057
     E  0.190794  0.683509
```

```
[ ]: df.W
```

```
[ ]: A    2.706850
     B    0.651118
     C   -2.018168
```

```
D    0.188695
E    0.190794
Name: W, dtype: float64
```

[ ]:
```
df['new'] = df['W'] + df['X']
```

[ ]:
```
df
```

[ ]:
```
          W         X         Y         Z       new
A  2.706850  0.628133  0.907969  0.503826  3.334983
B  0.651118 -0.319318 -0.848077  0.605965  0.331800
C -2.018168  0.740122  0.528813 -0.589001 -1.278046
D  0.188695 -0.758872 -0.933237  0.955057 -0.570177
E  0.190794  1.978757  2.605967  0.683509  2.169552
```

[ ]:
```
df.drop('new', axis=1)
```

[ ]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

[ ]:
```
df
```

[ ]:
```
          W         X         Y         Z       new
A  2.706850  0.628133  0.907969  0.503826  3.334983
B  0.651118 -0.319318 -0.848077  0.605965  0.331800
C -2.018168  0.740122  0.528813 -0.589001 -1.278046
D  0.188695 -0.758872 -0.933237  0.955057 -0.570177
E  0.190794  1.978757  2.605967  0.683509  2.169552
```

[ ]:
```
df.drop('new', axis=1, inplace=True)
```

[ ]:
```
df
```

[ ]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

[ ]:
```
df.loc['A']
```

```
W    2.706850
X    0.628133
Y    0.907969
Z    0.503826
Name: A, dtype: float64
```

```
df.loc['A', 'W']
```

```
2.706849839399938
```

```
type(df.loc['A', 'W'])
```

```
numpy.float64
```

```
df.loc[['A', 'B', 'C'], ['X', 'Y']]
```

```
          X         Y
A  0.628133  0.907969
B -0.319318 -0.848077
C  0.740122  0.528813
```

```
df.iloc[2:4, 0:2]
```

```
          W         X
C -2.018168  0.740122
D  0.188695 -0.758872
```

```
df.iloc[1:9, 5:]
```

```
Empty DataFrame
Columns: []
Index: [B, C, D, E]
```

4 DataFrame - Seleção Condicional

```
bol = df != 0
```

```
bol
```

```
      W     X     Y     Z
A  True  True  True  True
B  True  True  True  True
C  True  True  True  True
D  True  True  True  True
E  True  True  True  True
```

[ ]:
```
df[bol]
```

[ ]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

[ ]:
```
df
```

[ ]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

[ ]:
```
df[df['W']>0]
```

[ ]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

[ ]:
```
df[df['X']>0]
```

[ ]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
C -2.018168  0.740122  0.528813 -0.589001
E  0.190794  1.978757  2.605967  0.683509
```

[ ]:
```
df[df['X']>0]['Y']
```

[ ]:
```
A    0.907969
C    0.528813
E    2.605967
Name: Y, dtype: float64
```

[ ]:
```
bol = df['X']>0
df2=df[bol]
df2['Y']
```

[ ]:
```
A    0.907969
C    0.528813
E    2.605967
```

```
Name: Y, dtype: float64
```

[ ]:
```
df[(df['W']>0) & (df['Y']>1)]
```

[ ]:
```
          W         X         Y         Z
E  0.190794  1.978757  2.605967  0.683509
```

[ ]:
```
df[(df['W']>0) & (df['Y']>1)]
```

[ ]:
```
          W         X         Y         Z
E  0.190794  1.978757  2.605967  0.683509
```

[ ]:
```
df[(df['W']>0) | (df['Y']>1)]
```

[ ]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

[ ]:
```
df.reset_index()
```

[ ]:
```
  index         W         X         Y         Z
0     A  2.706850  0.628133  0.907969  0.503826
1     B  0.651118 -0.319318 -0.848077  0.605965
2     C -2.018168  0.740122  0.528813 -0.589001
3     D  0.188695 -0.758872 -0.933237  0.955057
4     E  0.190794  1.978757  2.605967  0.683509
```

[ ]:
```
df
```

[ ]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

[ ]:
```
df.reset_index(inplace=True)
```

[ ]:
```
df
```

[ ]:
```
  index         W         X         Y         Z
0     A  2.706850  0.628133  0.907969  0.503826
1     B  0.651118 -0.319318 -0.848077  0.605965
2     C -2.018168  0.740122  0.528813 -0.589001
3     D  0.188695 -0.758872 -0.933237  0.955057
```

```
4     E  0.190794  1.978757  2.605967  0.683509
```

[ ]:
```
col = 'RS RJ SP AM SC'.split()
```

[ ]:
```
col
```

[ ]:
```
['RS', 'RJ', 'SP', 'AM', 'SC']
```

[ ]:
```
df['Estado'] = col
```

[ ]:
```
df
```

[ ]:
```
  index         W         X         Y         Z Estado
0     A  2.706850  0.628133  0.907969  0.503826     RS
1     B  0.651118 -0.319318 -0.848077  0.605965     RJ
2     C -2.018168  0.740122  0.528813 -0.589001     SP
3     D  0.188695 -0.758872 -0.933237  0.955057     AM
4     E  0.190794  1.978757  2.605967  0.683509     SC
```

[ ]:
```
df.set_index('Estado')
```

[ ]:
```
        index         W         X         Y         Z
Estado
RS          A  2.706850  0.628133  0.907969  0.503826
RJ          B  0.651118 -0.319318 -0.848077  0.605965
SP          C -2.018168  0.740122  0.528813 -0.589001
AM          D  0.188695 -0.758872 -0.933237  0.955057
SC          E  0.190794  1.978757  2.605967  0.683509
```

[ ]:
```
df
```

[ ]:
```
  index         W         X         Y         Z Estado
0     A  2.706850  0.628133  0.907969  0.503826     RS
1     B  0.651118 -0.319318 -0.848077  0.605965     RJ
2     C -2.018168  0.740122  0.528813 -0.589001     SP
3     D  0.188695 -0.758872 -0.933237  0.955057     AM
4     E  0.190794  1.978757  2.605967  0.683509     SC
```

[ ]:
```
df.set_index('Estado', inplace=True)
```

[ ]:
```
df
```

[ ]:
```
        index         W         X         Y         Z
Estado
RS          A  2.706850  0.628133  0.907969  0.503826
RJ          B  0.651118 -0.319318 -0.848077  0.605965
SP          C -2.018168  0.740122  0.528813 -0.589001
```

```
AM         D  0.188695 -0.758872 -0.933237  0.955057
SC         E  0.190794  1.978757  2.605967  0.683509
```

5 DataFrame - Indice Multinível

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

[ ]:
```
outside = ['G1', 'G1', 'G1', 'G2', 'G2', 'G2']
inside = [1, 2, 3, 1, 2 , 3]
hier_index= list(zip(outside,inside))
hier_index = pd.MultiIndex.from_tuples(hier_index)
```

[ ]:
```
outside
```

[ ]:
```
['G1', 'G1', 'G1', 'G2', 'G2', 'G2']
```

[ ]:
```
inside
```

[ ]:
```
[1, 2, 3, 1, 2, 3]
```

[ ]:
```
hier_index
```

[ ]:
```
MultiIndex([('G1', 1),
            ('G1', 2),
            ('G1', 3),
            ('G2', 1),
            ('G2', 2),
            ('G2', 3)],
           )
```

[ ]:
```
df = pd.DataFrame(np.random.randn(6,2), index=hier_index, columns=['A', 'B'])
```

[ ]:
```
df
```

[ ]:
```
              A         B
G1 1  0.302665  1.693723
   2 -1.706086 -1.159119
   3 -0.134841  0.390528
G2 1  0.166905  0.184502
   2  0.807706  0.072960
   3  0.638787  0.329646
```

[ ]:
```
df.loc['G1']
```

```
          A         B
1  0.302665  1.693723
2 -1.706086 -1.159119
3 -0.134841  0.390528
```

```
type(df.loc['G1'])
```

```
pandas.core.frame.DataFrame
```

SERIES

```
df.loc['G1'].loc[1]
```

```
A    0.302665
B    1.693723
Name: 1, dtype: float64
```

```
type(df.loc['G1'].loc[1])
```

```
pandas.core.series.Series
```

```
df.index.names
```

```
FrozenList([None, None])
```

```
df.index.names = ['Grupo', 'Numero']
```

```
df
```

```
                     A         B
Grupo Numero
G1    1       0.302665  1.693723
      2      -1.706086 -1.159119
      3      -0.134841  0.390528
G2    1       0.166905  0.184502
      2       0.807706  0.072960
      3       0.638787  0.329646
```

Cross-section (Seção cruzada) Com ele consigo pegar elementos do nivel interno sem a necessidade de passar parametros do nivél interno

```
df.xs('G1')
```

```
               A         B
Numero
1       0.302665  1.693723
2      -1.706086 -1.159119
3      -0.134841  0.390528
```

[ ]:
```
df
```

[ ]:
```
                     A         B
Grupo Numero
G1    1       0.302665  1.693723
      2      -1.706086 -1.159119
      3      -0.134841  0.390528
G2    1       0.166905  0.184502
      2       0.807706  0.072960
      3       0.638787  0.329646
```

[ ]:
```
df.xs(2, level='Numero')
```

[ ]:
```
              A         B
Grupo
G1    -1.706086 -1.159119
G2     0.807706  0.072960
```

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

[ ]:
```
df = pd.DataFrame({'A':[1,2,np.nan],
                   'B':[5,np.nan,np.nan],
                   'C':[1,2,3]})
```

[ ]:
```
df
```

[ ]:
```
     A    B  C
0  1.0  5.0  1
1  2.0  NaN  2
2  NaN  NaN  3
```

[ ]:
```
df.dropna()
```

[ ]:
```
     A    B  C
0  1.0  5.0  1
```

[ ]:
```
df.dropna(axis=1)
```

[ ]:
```
   C
0  1
1  2
2  3
```

[ ]:
```
df.dropna(thresh=2)
```

[ ]:
```
     A    B  C
0  1.0  5.0  1
1  2.0  NaN  2
```

[ ]:
```
df.fillna(value='Conteúdo')
```

[ ]:
```
           A         B  C
0          1         5  1
1          2  Conteúdo  2
2  Conteúdo  Conteúdo  3
```

[ ]:
```
df['A'].fillna(value=df['A'].mean())
```

[ ]:
```
0    1.0
1    2.0
2    1.5
Name: A, dtype: float64
```

6 Tratamento de Dados ausentes

Pandas tem funcionalidades para tratamento de dados ausentes

Vamos mostrar alguns métodos convenientes para lidar com Missing Data em pandas:

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

Vamos criar um dicionário de listas, com valores faltantes.

[ ]:
```
df = pd.DataFrame({'A':[1,2,np.nan],
                   'B':[5,np.nan,np.nan],
                   'C':[1,2,3]})
```

Crio um Dataframe com a lista em que temos dados faltantes.

Veja que quando criamos as listas, informamos que temos elementos nan do tipo numpy.

[ ]:
```
df
```

[ ]:
```
     A    B  C
0  1.0  5.0  1
1  2.0  NaN  2
2  NaN  NaN  3
```

Primeira forma é usando o método dropna, que exclui os valores faltantes e alguns valores existentes.

Isso acontece pois o dropna() faz por padrão referencia ao axis=0 ou seja ao eixo zero, assim ele exclui valores NaN nas linhas 1 e 2.

Se passamos como parametro do método dropna() o eixo 1 ele exclui valores das colunas que tem NaN assim nesse caso ele exckui os valores das colunas A e B

[ ]:
```
df.dropna()
```

[ ]:
```
     A    B  C
0  1.0  5.0  1
```

[ ]:
```
df.dropna(axis=1)
```

[ ]:
```
   C
0  1
1  2
2  3
```

[ ]:
```
df.dropna(axis=1)
```

[ ]:
```
   C
0  1
1  2
2  3
```

6.1 Tresh

Ele só exclui as linhas que tem a quantidade de elementos faltantes indicadas em thresh.

Nesse caso ele usa como eixo padrão o eixo 0 das linhas

[ ]:
```
df.dropna(thresh=2)
```

[ ]:
```
     A    B  C
0  1.0  5.0  1
1  2.0  NaN  2
```

[ ]:
```
df.dropna(axis =1, thresh=2)
```

[ ]:
```
     A  C
0  1.0  1
1  2.0  2
2  NaN  3
```

Nos métodos dropna() e thresh() para realizar a mudança no DataFrame, usamos o inplace=True.

Mas não vamos faze isso agora.

6.2 Método Fillna

Ele substitui os valores indicados em fillna no DataFrame

[ ]:
```
df.fillna(value='Conteúdo')
```

[ ]:
```
           A          B  C
0          1          5  1
1          2  Conteúdo  2
2  Conteúdo  Conteúdo  3
```

Se quisr substituir os valores que tem NaN na coluna A pela média da coluna podemos usar a média dos valores de um dataframe formado pela coluna A, calcular a média e substituir com fillna. Como podemos observar abaixo:

[ ]:
```
df['A'].fillna(value=df['A'].mean())
```

[ ]:
```
0    1.0
1    2.0
2    1.5
Name: A, dtype: float64
```

O método não é alterado só altera se colocarmos inplace=True

Como o método method pegamos o ultimo valor diferente de NaN e colocamos em seu lugar,

[ ]:
```
df.fillna(method='ffill')
```

[ ]:
```
     A    B  C
0  1.0  5.0  1
1  2.0  5.0  2
2  2.0  5.0  3
```

7 Groupby

O método groupby permite agrupar linhas de dados em conjunto e chamar funções agregadas

[ ]:
```
import pandas as pd
# Cria um DataFrame
data = {'Empresa':['GOOG','GOOG','MSFT','MSFT','FB','FB'],
       'Nome':['Sam','Charlie','Amy','Vanessa','Carl','Sarah'],
       'Venda':[200,120,340,124,243,350]}
```

[ ]:
```
df = pd.DataFrame(data)
```

[ ]:
```
df
```

[ ]:
```
  Empresa     Nome  Venda
0    GOOG      Sam    200
1    GOOG  Charlie    120
2    MSFT      Amy    340
3    MSFT  Vanessa    124
4      FB     Carl    243
5      FB    Sarah    350
```

** Agora, você pode usar o método .group by () para agrupar as linhas em conjunto com base em um nome de coluna. Por exemplo, vamos agrupar com base na empresa. Isso criará um objeto DataFrameGroupBy:**

[ ]:
```
df.groupby('Empresa')
```

[ ]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7f3ccc659790>
```

Você pode salvar este objeto como uma nova variável:

[ ]:
```
por_companhia = df.groupby("Empresa")
```

E, em seguida, chamar métodos agregados do objeto:

[ ]:
```
por_companhia.mean()
```

[ ]:
```
         Venda
Empresa
FB       296.5
GOOG     160.0
MSFT     232.0
```

[ ]:
```
df.groupby('Empresa').mean()
```

[ ]:
```
         Venda
Empresa
FB       296.5
GOOG     160.0
MSFT     232.0
```

Mais exemplos de métodos agregados:

[ ]:
```
por_companhia.std()
```

[ ]:
```
              Venda
Empresa
FB        75.660426
GOOG      56.568542
MSFT     152.735065
```

[ ]:
```
por_companhia.min()
```

[ ]:
```
            Nome  Venda
Empresa
FB          Carl    243
GOOG     Charlie    120
MSFT         Amy    124
```

[ ]:
```
por_companhia.max()
```

[ ]:
```
            Nome  Venda
Empresa
FB         Sarah    350
GOOG         Sam    200
MSFT     Vanessa    340
```

[ ]:
```
por_companhia.count()
```

[ ]:
```
         Nome  Venda
Empresa
FB          2      2
GOOG        2      2
MSFT        2      2
```

[ ]:
```
por_companhia.describe()
```

[ ]:
```
        Venda
        count   mean         std    min     25%    50%     75%    max
Empresa
FB        2.0  296.5   75.660426  243.0  269.75  296.5  323.25  350.0
GOOG      2.0  160.0   56.568542  120.0  140.00  160.0  180.00  200.0
MSFT      2.0  232.0  152.735065  124.0  178.00  232.0  286.00  340.0
```

[ ]:
```
por_companhia.describe().transpose()
```

[ ]:
```
Empresa              FB        GOOG        MSFT
Venda count    2.000000    2.000000    2.000000
      mean   296.500000  160.000000  232.000000
      std     75.660426   56.568542  152.735065
      min    243.000000  120.000000  124.000000
      25%    269.750000  140.000000  178.000000
      50%    296.500000  160.000000  232.000000
      75%    323.250000  180.000000  286.000000
      max    350.000000  200.000000  340.000000
```

[ ]:
```
por_companhia.describe().transpose()['GOOG']
```

[ ]:
```
Venda  count      2.000000
       mean     160.000000
       std       56.568542
       min      120.000000
       25%      140.000000
       50%      160.000000
       75%      180.000000
       max      200.000000
Name: GOOG, dtype: float64
```

8 Mesclar, Juntar, e Concatenar

Existem três maneiras principais de combinar os DataFrames: mesclando, juntando e concatenando (merge, join e concat). Nesta palestra, discutiremos esses 3 métodos com exemplos.

8.0.1 Exemplos de DataFrames

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

[ ]:
```
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']},
                    index=[0, 1, 2, 3])
```

[ ]:
```
df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                    'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7']},
                     index=[4, 5, 6, 7])
```

[ ]:
```
df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
                    'B': ['B8', 'B9', 'B10', 'B11'],
                    'C': ['C8', 'C9', 'C10', 'C11'],
                    'D': ['D8', 'D9', 'D10', 'D11']},
                    index=[8, 9, 10, 11])
```

[ ]:
```
df1
```

[ ]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

[ ]:
```
df2
```

[ ]:

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

[ ]:
```
df3
```

[ ]:

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

8.1 Concatenação

Concatenação basicamente cola DataFrames. Tenha em mente que as dimensões devem corresponder ao longo do eixo que você está concatenando. Você pode usar ** pd.concat ** e passar uma lista de DataFrames para concatenar juntos:

[ ]:
```
pd.concat([df1,df2,df3])
```

[ ]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

[ ]:
```
pd.concat([df1,df2,df3],axis=1)
```

[ ]:

	A	B	C	D	A	B	C	D	A	B	C	D
0	A0	B0	C0	D0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	A2	B2	C2	D2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	A3	B3	C3	D3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	A4	B4	C4	D4	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	A5	B5	C5	D5	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	A6	B6	C6	D6	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	A7	B7	C7	D7	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A8	B8	C8	D8
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A9	B9	C9	D9
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A10	B10	C10	D10
11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A11	B11	C11	D11

8.2 Outros DataFrames

[ ]:
```
esquerda = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                         'A': ['A0', 'A1', 'A2', 'A3'],
                         'B': ['B0', 'B1', 'B2', 'B3']})

direita = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                            'C': ['C0', 'C1', 'C2', 'C3'],
                            'D': ['D0', 'D1', 'D2', 'D3']})
```

[ ]:
```
esquerda
```

[ ]:
```
  key   A   B
0  K0  A0  B0
1  K1  A1  B1
2  K2  A2  B2
3  K3  A3  B3
```

[ ]:
```
direita
```

[ ]:
```
  key   C   D
0  K0  C0  D0
1  K1  C1  D1
2  K2  C2  D2
3  K3  C3  D3
```

8.3 Mesclar

A função ** mesclar ** permite que você mescle os quadros de dados juntos usando uma lógica semelhante à mesclagem de tabelas SQL juntas. Por exemplo:

[ ]:
```
pd.merge(esquerda,direita,how='inner',on='key')
```

[ ]:
```
  key   A   B   C   D
0  K0  A0  B0  C0  D0
1  K1  A1  B1  C1  D1
2  K2  A2  B2  C2  D2
3  K3  A3  B3  C3  D3
```

Ou para mostrar um exemplo mais complicado:

[ ]:
```
esquerda = pd.DataFrame({'key1': ['K0', 'K0', 'K1', 'K2'],
                         'key2': ['K0', 'K1', 'K0', 'K1'],
                            'A': ['A0', 'A1', 'A2', 'A3'],
                            'B': ['B0', 'B1', 'B2', 'B3']})

direita = pd.DataFrame({'key1': ['K0', 'K1', 'K1', 'K2'],
                                'key2': ['K0', 'K0', 'K0', 'K0'],
```

```
                                    'C': ['C0', 'C1', 'C2', 'C3'],
                                    'D': ['D0', 'D1', 'D2', 'D3']})
```

[ ]:
```
esquerda
```

[ ]:
```
  key1 key2   A   B
0   K0   K0  A0  B0
1   K0   K1  A1  B1
2   K1   K0  A2  B2
3   K2   K1  A3  B3
```

[ ]:
```
direita
```

[ ]:
```
  key1 key2   C   D
0   K0   K0  C0  D0
1   K1   K0  C1  D1
2   K1   K0  C2  D2
3   K2   K0  C3  D3
```

[ ]:
```
pd.merge(esquerda, direita, on=['key1', 'key2'])
```

[ ]:
```
  key1 key2   A   B   C   D
0   K0   K0  A0  B0  C0  D0
1   K1   K0  A2  B2  C1  D1
2   K1   K0  A2  B2  C2  D2
```

[ ]:
```
esquerda
```

[ ]:
```
  key1 key2   A   B
0   K0   K0  A0  B0
1   K0   K1  A1  B1
2   K1   K0  A2  B2
3   K2   K1  A3  B3
```

[ ]:
```
direita
```

[ ]:
```
  key1 key2   C   D
0   K0   K0  C0  D0
1   K1   K0  C1  D1
2   K1   K0  C2  D2
3   K2   K0  C3  D3
```

[ ]:
```
pd.merge(esquerda, direita, how='outer', on=['key1', 'key2'])
```

[ ]:
```
  key1 key2    A    B    C    D
0   K0   K0   A0   B0   C0   D0
1   K0   K1   A1   B1  NaN  NaN
```

```
2   K1   K0   A2   B2   C1   D1
3   K1   K0   A2   B2   C2   D2
4   K2   K1   A3   B3  NaN  NaN
5   K2   K0  NaN  NaN   C3   D3
```

[ ]:
```
esquerda
```

[ ]:
```
  key1 key2   A   B
0   K0   K0  A0  B0
1   K0   K1  A1  B1
2   K1   K0  A2  B2
3   K2   K1  A3  B3
```

[ ]:
```
direita
```

[ ]:
```
  key1 key2   C   D
0   K0   K0  C0  D0
1   K1   K0  C1  D1
2   K1   K0  C2  D2
3   K2   K0  C3  D3
```

[ ]:
```
pd.merge(esquerda, direita, how='right', on=['key1', 'key2'])
```

[ ]:
```
  key1 key2    A    B   C   D
0   K0   K0   A0   B0  C0  D0
1   K1   K0   A2   B2  C1  D1
2   K1   K0   A2   B2  C2  D2
3   K2   K0  NaN  NaN  C3  D3
```

[ ]:
```
direita
```

[ ]:
```
  key1 key2   C   D
0   K0   K0  C0  D0
1   K1   K0  C1  D1
2   K1   K0  C2  D2
3   K2   K0  C3  D3
```

[ ]:
```
esquerda
```

[ ]:
```
  key1 key2   A   B
0   K0   K0  A0  B0
1   K0   K1  A1  B1
2   K1   K0  A2  B2
3   K2   K1  A3  B3
```

[ ]:
```
pd.merge(esquerda, direita, how='left', on=['key1', 'key2'])
```

[ ]:
```
  key1 key2   A   B    C    D
0   K0   K0  A0  B0   C0   D0
1   K0   K1  A1  B1  NaN  NaN
2   K1   K0  A2  B2   C1   D1
3   K1   K0  A2  B2   C2   D2
4   K2   K1  A3  B3  NaN  NaN
```

8.4 Juntar

Juntar é um método conveniente para combinar as colunas de dois DataFrames indexados potencialmente diferentes em um único resultado DataFrame.

[ ]:
```
esquerda = pd.DataFrame({'A': ['A0', 'A1', 'A2'],
                         'B': ['B0', 'B1', 'B2']},
                         index=['K0', 'K1', 'K2'])

direita = pd.DataFrame({'C': ['C0', 'C2', 'C3'],
                        'D': ['D0', 'D2', 'D3']},
                         index=['K0', 'K2', 'K3'])
```

[ ]:
```
esquerda
```

[ ]:
```
     A   B
K0  A0  B0
K1  A1  B1
K2  A2  B2
```

[ ]:
```
direita
```

[ ]:
```
     C   D
K0  C0  D0
K2  C2  D2
K3  C3  D3
```

[ ]:
```
esquerda.join(direita)
```

[ ]:
```
     A   B    C    D
K0  A0  B0   C0   D0
K1  A1  B1  NaN  NaN
K2  A2  B2   C2   D2
```

[ ]:
```
esquerda.join(direita, how='outer')
```

[ ]:
```
      A    B    C    D
K0   A0   B0   C0   D0
K1   A1   B1  NaN  NaN
K2   A2   B2   C2   D2
```

```
K3  NaN  NaN   C3   D3
```

9 Operações

Há muitas operações com pandas que serão realmente úteis para você, mas não se enquadram em nenhuma categoria distinta. Vamos mostrar aqui nesta aula:

[ ]:
```
import pandas as pd
df = pd.DataFrame({'col1':[1,2,3,4],'col2':[444,555,666,444],'col3':
↪['abc','def','ghi','xyz']})
df.head()
```

[ ]:
```
   col1  col2 col3
0     1   444  abc
1     2   555  def
2     3   666  ghi
3     4   444  xyz
```

9.0.1 Informação sobre valores exclusivos

[ ]:
```
df['col2'].unique()
```

[ ]:
```
array([444, 555, 666])
```

[ ]:
```
df['col2'].nunique()
```

[ ]:
```
3
```

[ ]:
```
df['col1'].value_counts()
```

[ ]:
```
4    1
3    1
2    1
1    1
Name: col1, dtype: int64
```

9.0.2 Selecionando dados

[ ]:
```
# Selecione do DataFrame usando critérios de várias colunas
newdf = df[(df['col1']>2) & (df['col2']==444)]
```

[ ]:
```
newdf
```

[ ]:
```
   col1  col2 col3
3     4   444  xyz
```

9.0.3 Aplicando funções

[ ]:
```
def times2(x):
    return x*2
```

[ ]:
```
df['col1'].apply(times2)
```

[ ]:
```
0    2
1    4
2    6
3    8
Name: col1, dtype: int64
```

[ ]:
```
df
```

[ ]:
```
   col1  col2 col3
0     1   444  abc
1     2   555  def
2     3   666  ghi
3     4   444  xyz
```

[ ]:
```
df['col3'].apply(len)
```

[ ]:
```
0    3
1    3
2    3
3    3
Name: col3, dtype: int64
```

[ ]:
```
df['col2'].sum()
```

[ ]:
```
2109
```

** Removendo colunas permanentemente **

[ ]:
```
del df['col1']
```

[ ]:
```
df
```

[ ]:
```
   col2 col3
0   444  abc
1   555  def
2   666  ghi
3   444  xyz
```

** Obter nomes de coluna e índice: **

[ ]:
```
df.columns
```

[ ]:
```
Index(['col2', 'col3'], dtype='object')
```

[ ]:
```
df.index
```

[ ]:
```
RangeIndex(start=0, stop=4, step=1)
```

** Ordenando um DataFrame **

[ ]:
```
df
```

[ ]:
```
   col2 col3
0   444  abc
1   555  def
2   666  ghi
3   444  xyz
```

[ ]:
```
df.sort_values(by='col2') #inplace=False por padrão
```

[ ]:
```
   col2 col3
0   444  abc
3   444  xyz
1   555  def
2   666  ghi
```

[ ]:
```
df
```

[ ]:
```
   col2 col3
0   444  abc
1   555  def
2   666  ghi
3   444  xyz
```

** Encontre Valores Nulos ou Verifique Valores Nulos **

[ ]:
```
df.isnull()
```

[ ]:
```
    col2   col3
0  False  False
1  False  False
2  False  False
3  False  False
```

[ ]:
```
# Deleta linhas com valores NaN
df.dropna()
```

[ ]:
```
   col2 col3
0   444  abc
1   555  def
```

```
2   666  ghi
3   444  xyz
```

** Preenchendo os valores de NaN com outra coisa: **

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

[ ]:
```
df = pd.DataFrame({'col1':[1,2,3,np.nan],
                   'col2':[np.nan,555,666,444],
                   'col3':['abc','def','ghi','xyz']})
df.head()
```

[ ]:
```
   col1   col2 col3
0   1.0    NaN  abc
1   2.0  555.0  def
2   3.0  666.0  ghi
3   NaN  444.0  xyz
```

[ ]:
```
df.fillna('Preencher')
```

[ ]:
```
         col1       col2 col3
0           1  Preencher  abc
1           2        555  def
2           3        666  ghi
3  Preencher        444  xyz
```

[ ]:
```
data = {'A':['foo','foo','foo','bar','bar','bar'],
     'B':['one','one','two','two','one','one'],
       'C':['x','y','x','y','x','y'],
       'D':[1,3,2,5,4,1]}

df = pd.DataFrame(data)
```

[ ]:
```
df
```

[ ]:
```
     A    B  C  D
0  foo  one  x  1
1  foo  one  y  3
2  foo  two  x  2
3  bar  two  y  5
4  bar  one  x  4
5  bar  one  y  1
```

[ ]:
```
df.pivot_table(values='D',index=['A', 'B'],columns=['C'])
```

[ ]:
```
C          x    y
A   B
```

```
bar one  4.0  1.0
    two  NaN  5.0
foo one  1.0  3.0
    two  2.0  NaN
```

10 Entrada e saída de dados

Este notebook conterá nossas referências sobre entrada e saída de dados. O pandas pode ler uma variedade de tipos de arquivos usando seus métodos pd.read_. Vejamos os tipos de dados mais comuns:

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

10.1 CSV

10.1.1 CSV Input

[ ]:
```
df = pd.read_csv('exemplo')
df
```

[ ]:
```
    a   b   c   d
0   0   1   2   3
1   4   5   6   7
2   8   9  10  11
3  12  13  14  15
```

10.1.2 Saída de dados tipo CSV

[ ]:
```
df.to_csv('exemplo.csv',index=False)
```

10.2 Excel

Pandas podem ler e escrever arquivos do Excel, tenha em mente, isso só importa dados. Não fórmulas nem imagens, lembrando que imagens ou macros podem bugar o método.

10.2.1 Entrada via Excel

[ ]:
```
pd.read_excel('Exemplo_Excel.xlsx')
```

[ ]:
```
   Unnamed: 0   a   b   c   d
0           0   0   1   2   3
1           1   4   5   6   7
2           2   8   9  10  11
3           3  12  13  14  15
```

10.2.2 Saída via Excel

[ ]:
```
df.to_excel('Exemplo_Excel.xlsx')
```

10.3 HTML

Você pode precisar instalar htmllib5, lxml e BeautifulSoup4. No seu terminal / prompt de comando, execute:

```
conda install lxml
conda install html5lib
conda install BeautifulSoup4
```

Em seguida, reinicie o Jupyter Notebook. (Ou use instalação de pip se não estiver usando a Distribuição de Anaconda) Pandas podem ler guias de tabelas fora de html. Por exemplo:

10.3.1 Entrada HTML

A função Pandas read_html irá ler tabelas fora de uma página da Web e retornar uma lista de objetos DataFrame:

[ ]:
```
df = pd.read_csv('https://www.fdic.gov/resources/resolutions/bank-failures/
↪failed-bank-list/banklist.csv')
```

[ ]:
```
df
```

[ ]:
```
                            Bank Name  … Closing Date
0                   Almena State Bank  …    23-Oct-20
1           First City Bank of Florida  …    16-Oct-20
2                 The First State Bank  …     3-Apr-20
3                   Ericson State Bank  …    14-Feb-20
4     City National Bank of New Jersey  …     1-Nov-19
..                                   …  …          …
558                 Superior Bank, FSB  …    27-Jul-01
559               Malta National Bank  …     3-May-01
560    First Alliance Bank & Trust Co.  …     2-Feb-01
561  National State Bank of Metropolis  …    14-Dec-00
562                   Bank of Honolulu  …    13-Oct-00

[563 rows x 6 columns]
```