Relatório 4 - Principais Bibliotecas e Ferramentas Python para Aprendizado de Máquina

Lucas Scheffer Hundsdorfer

Descrição da atividade

No começo da sessão 3 é ensinado a como instalar e configurar o ambiente de trabalho, o Jupyter Notebook. O Jupyter Notebook é um ambiente de desenvolvimento iterativo com o live code, nele cada desenvolvedor pode dividir o código em partes e trabalhar nelas independentemente. Ele é mais utilizado em Machine Learning criando redes neurais e também por cientistas de dados.

Na sessão 5 já nos é apresentado o Numpy, biblioteca em python de algebra linear, e extremamente eficaz pois seus principais métodos foram escritos em C, sua importação é feita da seguinte maneira:

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

Na aula 20 Criação de vetores e arrays e numpy.random é passado alguns comandos básicos do numpy e suas utilidades:

np.array(), onde eu transformo minha lista em array de numpy:

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

[5]: np.array(minhalista)

[5]: array([1, 2, 3])
```

```
[7]: minhamatriz = [[1,2,3], [4,5,6], [7,8,9]]

[9]: np.array(minhamatriz)

[9]: array([[1, 2, 3],
            [4, 5, 6],
            [7, 8, 9]])
```

np.arrange(começo,fim,passo), como no exemplo abaixo:

```
[13]: np.arange(0,10)

[13]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

[17]: np.arange(0,10,2)

[17]: array([0, 2, 4, 6, 8])
```

np.zeros(), cria um array do tamanho especificado só com zeros:

```
[19]: np.zeros(3)

[19]: array([0., 0., 0.])
```

```
np.ones((3,3))

array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

np.ones(), cria um array de tamanho especificado só com uns:

np.eye(), cria uma matriz entidade com o tamanho especificado:

```
[31]: np.eye(4)

[31]: array([[1., 0., 0., 0.],
             [0., 1., 0., 0.],
             [0., 0., 1., 0.],
             [0., 0., 0., 1.]])
```

np.linspace(começo, fim, quantidade) cria um array com a quantidade especificada e acha os números distribuídos uniformemente:

```
[33]: np.linspace(0,10,2)

[33]: array([ 0., 10.])

[35]: np.linspace(0,10,3)

[35]: array([ 0.,  5., 10.])
```

```
[49]: np.linspace(0,15,4)

[49]: array([ 0.,  5., 10., 15.])
```

np.random.rand() cria um array com valores entre 0 e 1 aleatórios:

```
[51]: np.random.rand(5)

[51]: array([0.80658514, 0.54245829, 0.14868336, 0.33751317, 0.46653766])
```

np.random.randn() cria um array com preenchida com floats aleatórios amostrados de uma distribuição univariada “normal” (Gaussiana):

```
[57]: np.random.randn(4)

[57]: array([ 0.74303431, -1.62845492, -0.96323183, -0.57887424])
```

np.random.randint(menor, maior, quantidade), cria um array com a quantidade, porém com valores variando do menor ao maior aleatórios:

```
[59]: np.random.randint(0,100,10)

[59]: array([37, 44, 64, 66,  6, 75, 99, 83,  2,  1])
```

.reshape(), transforma o array no tamanho que quiser:

```
[71]: arr.reshape((5,5))

[71]: array([[0.11868112, 0.0532342 , 0.66256125, 0.68516834, 0.58217344],
             [0.53141158, 0.65042231, 0.01827175, 0.3654666 , 0.19423083],
             [0.53482347, 0.01974815, 0.37784273, 0.52862588, 0.10076138],
             [0.79618243, 0.72938106, 0.65223364, 0.65560883, 0.72515067],
             [0.77081735, 0.04629129, 0.21692198, 0.72093358, 0.99725091]])
```

.shape, mostra o formato em que o array está:

```
[81]: arr1.shape

[81]: (5, 5)
```

.max(), mostra o maior valor do array, .min(), mostra o menor valor do array:

```
[87]: arr.max()

[87]: 0.9972509100835263
```

```
[89]: arr.min()

[89]: 0.01827174563807976
```

.argmax(), mostra o índice do maior valor do array

```
[93]: arr1.argmax()

[93]: 24
```

Na aula 21, vemos como funciona o indexamento e o fatiamento dos arrays:

```
[7]: arr

[7]: array([ 0,  3,  6,  9, 12, 15, 18, 21, 24, 27])

[9]: arr[4]

[9]: 12
```

Conseguimos também fatiar uma parte do array utilizando os índices:

```
[11]: arr[2:5]

[11]: array([ 6,  9, 12])
```

Conseguimos alterar os valores utilizando os índices:

```
[15]: array([ 6,  9, 12, 15, 18, 21, 24, 27])

[17]: arr[2:] = 100

[19]: arr

[19]: array([  0,   3, 100, 100, 100, 100, 100, 100, 100, 100])
```

Da mesma forma dá para fazer fatiamentos com arrays bidimensionais:

```
arr = np.arange(50).reshape((5,10))
```

```
arr
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44, 45, 46, 47, 48, 49]])
```

```
arr[:3]
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29]])
```

Não dá para criar um array, dar valor de um outro array e alterar o array criado dessa forma, pois o python não faz uma cópia direta:

```
arr2 = arr[:3]
```

```
arr2
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29]])
```

```
arr2[:] = 100
```

```
arr2
```

```
array([[100, 100, 100, 100, 100, 100, 100, 100, 100, 100],
       [100, 100, 100, 100, 100, 100, 100, 100, 100, 100],
       [100, 100, 100, 100, 100, 100, 100, 100, 100, 100]])
```

```
arr
```

```
array([[100, 100, 100, 100, 100, 100, 100, 100, 100, 100],
       [100, 100, 100, 100, 100, 100, 100, 100, 100, 100],
       [100, 100, 100, 100, 100, 100, 100, 100, 100, 100],
       [ 30,  31,  32,  33,  34,  35,  36,  37,  38,  39],
       [ 40,  41,  42,  43,  44,  45,  46,  47,  48,  49]])
```

O arr2 é alterado e altera juntamente o arr, se quiser fazer algo parecido necessita usar .copy como na imagem abaixo:

```
arr2 = arr[:3].copy()
```

```
arr2
```

```
array([[100, 100, 100, 100, 100, 100, 100, 100, 100, 100],
       [100, 100, 100, 100, 100, 100, 100, 100, 100, 100],
```

Dá para fazer fatiamento dos arrays utilizando bool:

```
arr > 50
```

```
array([[ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [ True,  True,  True,  True,  True,  True,  True,  True,  True,
         True],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False]])
```

```
bol = arr > 50
```

```
arr[bol]
```

```
array([100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
       100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
       100, 100, 100, 100])
```

A partir daí são várias formas possíveis para utilizar e usar os arrays, como no exemplo abaixo:

```
[57]: array = np.linspace(0,100,30)

[65]: array.shape

[65]: (30,)

[67]: array = array.reshape(3,10)

[69]: array.shape

[69]: (3, 10)

[71]: array[0:2, 2]

[71]: array([ 6.89655172, 41.37931034])
```

Na aula 22, vemos algumas operações com arrays:

Soma, subtração, multiplicação, divisão do array com o próprio array, importante lembrar que para isso os arrays precisam ter o mesmo tamanho:

```
[3]: import numpy as np

[7]: arr = np.arange(0,16)

[9]: arr

[9]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])

[11]: arr + arr

[11]: array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30])
```

```
[13]: arr - arr

[13]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
[15]: arr * arr

[15]: array([  0,   1,   4,   9,  16,  25,  36,  49,  64,  81, 100, 121, 144,
        169, 196, 225])
```

```
[17]: arr / arr

C:\Users\lshun\AppData\Local\Temp\ipykernel_4960\3001117470.py:1: Runtin
  arr / arr

[17]: array([nan,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
        1.,  1.,  1.])
```

O array também pode ser exponenciado índice por índice:

```
[21]: arr ** 2

[21]: array([  0,   1,   4,   9,  16,  25,  36,  49,  64,  81, 100, 121, 144,
        169, 196, 225])
```

Também é possível efetuar operações com números inteiros:

```
[23]: arr + 100

[23]: array([100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112,
        113, 114, 115])

[25]: arr - 100

[25]: array([-100,  -99,  -98,  -97,  -96,  -95,  -94,  -93,  -92,  -91,  -90,
         -89,  -88,  -87,  -86,  -85])

[27]: arr * 100

[27]: array([   0,  100,  200,  300,  400,  500,  600,  700,  800,  900, 1000,
        1100, 1200, 1300, 1400, 1500])
```

O numpy oferece algumas funções matemáticas como raiz quadrada, desvio padrão, seno entre outros:

```
[33]: np.median(arr)

[33]: 7.5

[35]: np.std(arr)

[35]: 4.6097722286464435

[37]: np.sin(arr)

[37]: array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
              -0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849,
              -0.54402111, -0.99999021, -0.53657292,  0.42016704,  0.99060736,
               0.65028784])
```

Após isso o curso nos apresenta a biblioteca Pandas:

Pandas é uma biblioteca escrita sobre o numpy, permitindo rápida visualização e limpeza de dados, semelhante ao excel, pode trabalhar com vários tipos de dados e possui métodos próprios para visualização.

Na primeira aula de Pandas ele ensina Series que seria algo muito proximo ao dicionário do Python:

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

[3]: labels = ['a', 'b', 'c']

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

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

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

Aqui ele salva um dado que seria 10,20,30 e associa ele com índices próprios, no caso ‘a’, ‘b’, ‘c’.

Também é possível acessar o valor através do seu índice:

```
[21]: series['b']

[21]: 20
```

É possível fazer a soma desses Series, porém acontece de uma maneira diferente, ele identifica os índices iguais e soma os seus valores, como no caso abaixo:

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

[46]: ser1

[46]: EUA               1
      Alemanha          2
      União Soviética   3
      Japão             4
      dtype: int64

 [5]: ser2 = pd.Series([4,5,6,3], index = ['EUA', 'Alemanha','Itália', 'Japão'])

 [7]: ser2

 [7]: EUA         4
      Alemanha    5
      Itália      6
      Japão       3
      dtype: int64

[13]: ser1 + ser2

[13]: Alemanha          7.0
      EUA               5.0
      Itália            NaN
      Japão             7.0
      União Soviética   NaN
      dtype: float64
```

Ele identificou Alemanha como índice nas duas Series e somou os valores que eram correspondentes a 2 e 5, totalizando 7, Itália e União Soviética que tiveram apenas uma aparição somam NaN, que significa que não conseguiu concluir a soma.

DATAFRAME:

DataFrame é o objeto principal da biblioteca Pandas, ele é basicamente um conjunto de Series, eles podem ser analisados quase que em forma de tabela do excel, porém ele só aparece formatado assim em algumas IDES que é o caso do Jupyter:

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

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

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

[11]: df
```

[11]:

	W	X	Y	Z
A	0.302665	1.693723	-1.706086	-1.159119
B	-0.134841	0.390528	0.166905	0.184502
C	0.807706	0.072960	0.638787	0.329646
D	-0.497104	-0.754070	-0.943406	0.484752
E	-0.116773	1.901755	0.238127	1.996652

Aqui ele utiliza dados, colunas e índices, dados sendo randômicos, índice indo de A:E e colunas indo de W:Z.

Conseguimos acessar uma coluna inteira utilizando apenas o nome da coluna:

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

[13]: A    0.302665
      B   -0.134841
      C    0.807706
      D   -0.497104
      E   -0.116773
      Name: W, dtype: float64
```

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

[17]:

	W	Z
A	0.302665	-1.159119
B	-0.134841	0.184502
C	0.807706	0.329646
D	-0.497104	0.484752
E	-0.116773	1.996652

Dá para criar novas colunas utilizando de uma maneira onde você atribui valor para uma coluna que não existe ainda:

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

[59]: df
```

[59]:

	W	X	Y	Z	new
A	0.302665	1.693723	-1.706086	-1.159119	1.996388
B	-0.134841	0.390528	0.166905	0.184502	0.255687
C	0.807706	0.072960	0.638787	0.329646	0.880666
D	-0.497104	-0.754070	-0.943406	0.484752	-1.251174
E	-0.116773	1.901755	0.238127	1.996652	1.784981

Dá para excluir uma coluna utilizando esses comandos:

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

[61]:

	W	X	Y	Z
A	0.302665	1.693723	-1.706086	-1.159119
B	-0.134841	0.390528	0.166905	0.184502
C	0.807706	0.072960	0.638787	0.329646
D	-0.497104	-0.754070	-0.943406	0.484752
E	-0.116773	1.901755	0.238127	1.996652

```
[63]: df
```

[63]:

	W	X	Y	Z	new
A	0.302665	1.693723	-1.706086	-1.159119	1.996388
B	-0.134841	0.390528	0.166905	0.184502	0.255687
C	0.807706	0.072960	0.638787	0.329646	0.880666
D	-0.497104	-0.754070	-0.943406	0.484752	-1.251174
E	-0.116773	1.901755	0.238127	1.996652	1.784981

Porém o DataFrame apenas se mostra alterado, o df continua com a coluna ‘new’.

Para conseguir excluir de fato uma coluna basta adicionar um parâmetro a mais:

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

```
[67]: df
```

[67]:

	W	X	Y	Z
A	0.302665	1.693723	-1.706086	-1.159119
B	-0.134841	0.390528	0.166905	0.184502
C	0.807706	0.072960	0.638787	0.329646
D	-0.497104	-0.754070	-0.943406	0.484752
E	-0.116773	1.901755	0.238127	1.996652

Você consegue acessar os dados utilizando o índice também, dessa forma:

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

[71]: W    0.302665
      X    1.693723
      Y   -1.706086
      Z   -1.159119
      Name: A, dtype: float64
```

uma outra forma de acessar é utilizando o índice e a coluna, assim:

```
[73]: df.loc[['A', 'B'], ['X', 'Z']]
```

[73]:

	X	Z
A	1.693723	-1.159119
B	0.390528	0.184502

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

```
[69]: 0.3026654485851825
```

e a última forma de visualizar os dados é utilizando o formato do numpy, desse jeito:

```
[77]: df.iloc[1:4, 2:]
```

[77]:

	Y	Z
B	0.166905	0.184502
C	0.638787	0.329646
D	-0.943406	0.484752

No DataFrame é possível mostrar apenas os elementos condicionais que você impor, como no caso:

```
import pandas as pd
import numpy as np
```

```
from numpy.random import randn
np.random.seed(101)
```

```
df = pd.DataFrame(randn(5,4), index='A B C D E'.split(), columns='W X Y Z'.split())
```

```
bol = df > 0
```

```
df[bol]
```

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	NaN	NaN	0.605965
C	NaN	0.740122	0.528813	NaN
D	0.188695	NaN	NaN	0.955057
E	0.190794	1.978757	2.605967	0.683509

Só é mostrado valores acima de 0.

Aqui nesse caso ele determina a condição para a coluna ‘W’ todos os elementos serem acima de 0, assim ficando sem a linha ‘C’:

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

[19]:

	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

É possível ir filtrando também da mesma maneira mas imprimir outra coluna com o condicional feito, como no exemplo:

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

```
[21]: A    0.907969
      B   -0.848077
      D   -0.933237
      E    2.605967
      Name: Y, dtype: float64
```

Utilizando os operadores de condição é possível selecionar as linhas utilizando argumentos lógicos dessa forma:

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

[29]:

	W	X	Y	Z
E	0.190794	1.978757	2.605967	0.683509

Está mostrando as linhas que o W é maior que 0 e o Y > 1.

Com um método é possível trocar os índices para se tornarem colunas, e transformando o índice no padrão numpy que vai de 0 até N, da seguinte forma:

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

[33]:

	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

Porém ele não fica alterado permanente, é necessário passar como argumento do método o (inplace = True).

A maneira mais fácil de criar uma coluna é assim:

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

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

[47]: df
```

[47]:

	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

Com um método do Pandas é possível transformar qualquer coluna no índice, porém para conseguir tornar permanente precisa passar o argumento (inplace=True):

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

[49]:

Estado	index	W	X	Y	Z
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

Aula 30, Índices Multinível:

Índices multinível funcionam como se fosse uma hierarquia de índices dentro do dataframe, e isso pode facilitar a maneira em que manipulamos os dados dependendo de como esses índices aparecem.

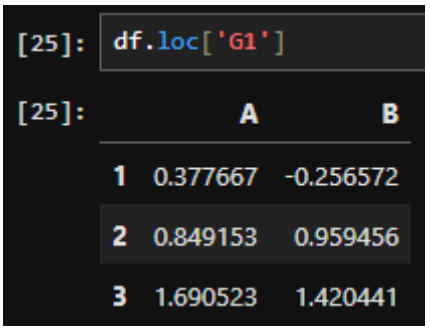
```
[3]: import pandas as pd
     import numpy as np

[7]: 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)

[15]: hier_index = list(zip(outside,inside))

[17]: hier_index

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

[19]: hier_index = pd.MultiIndex.from_tuples(hier_index)

[21]: hier_index

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

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

[15]: df
```

[15]:

		A	B
G1	1	0.377667	-0.256572
	2	0.849153	0.959456
	3	1.690523	1.420441
G2	1	1.874619	-0.601145
	2	0.314028	-0.237207
	3	0.305134	2.074738

Aqui neste código foi criado um MultiIndex, associando 2 listas, e depois disso foi feita a criação do dataframe com dados aleatórios, índice hierarquizado e 2 colunas.

Como nos outros dataFrames é possível acessar os dados com o método loc, aqui não é diferente, mas o resultado de você acessar um dos índices vai te retornar um dataFrame normal com apenas um índice, como no exemplo abaixo:

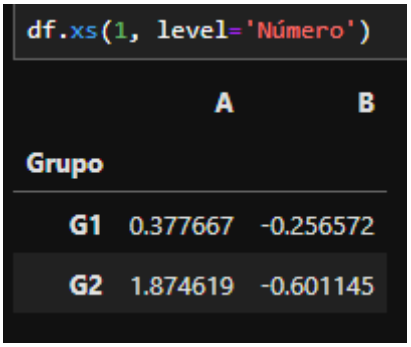


```
[25]: df.loc['G1']

[25]:          A         B
      1  0.377667 -0.256572
      2  0.849153  0.959456
      3  1.690523  1.420441
```

O que é possível também para facilitar o entendimento do DataFrame é nomear cada índice para uma melhor compreensão dos dados e uma manipulação menos complicada, exemplo:

```
[29]: df.index.names = ['Grupo', 'Número']

[31]: df

[31]:                    A         B
      Grupo Número
      G1    1      0.377667 -0.256572
            2      0.849153  0.959456
            3      1.690523  1.420441
      G2    1      1.874619 -0.601145
            2      0.314028 -0.237207
            3      0.305134  2.074738
```

Nos índices de multinível é possível filtrar o grupo interno com um método disponível no panda:



```
df.xs(1, level='Número')

              A         B
Grupo
G1     0.377667 -0.256572
G2     1.874619 -0.601145
```

Aqui ele filtrou os índices 1 do grupo Número e permaneceu o índice maior.

Aula 31 Tratamento de Dados Ausentes:

Nessa aula aprendemos como podemos lidar e tratar com dados que estão faltando nos dataFrames, porém isso também depende de que tipo de dados você está tratando e de onde eles vêm, por isso é interessante saber com o que você está trabalhando.

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

[4]: d = {'A':[1,2,np.nan], 'B':[5,np.nan,np.nan], 'C':[1,2,3]}

[6]: d

[6]: {'A': [1, 2, nan], 'B': [5, nan, nan], 'C': [1, 2, 3]}

[8]: df = pd.DataFrame(d)

[10]: df

[10]:
```

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2
2	NaN	NaN	3

Aqui um dicionário foi criado e passado como parâmetro para o DataFrame com alguns dados faltando.

Uma das formas de lidar com esses dados ausentes é método dropna, porém ele por padrão exclui todas as linhas que tiverem um dado ausente, o que não é muito interessante, mas com o argumento thresh, é possível que você determine um número mínimo de dados ausentes para a exclusão, exemplo:

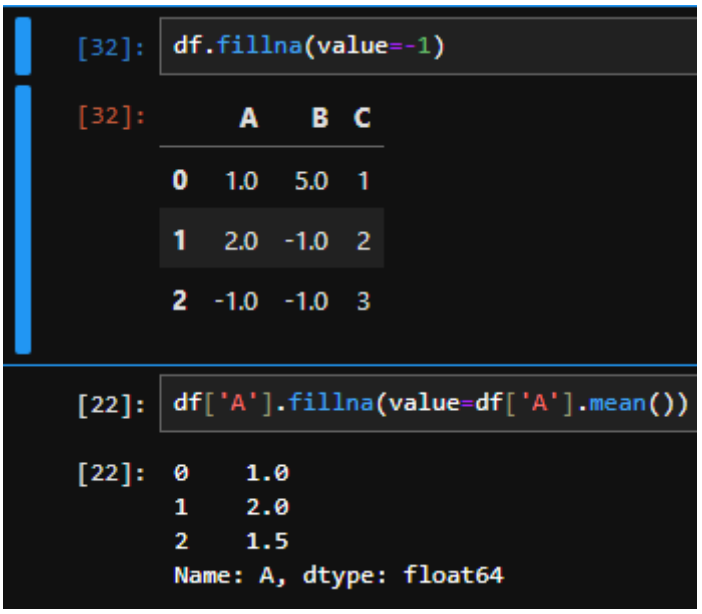
```
[14]: df.dropna()

[14]:
```

	A	B	C
0	1.0	5.0	1

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

[16]:
```

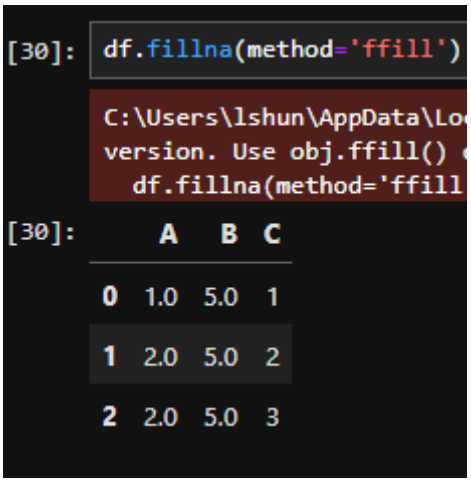
	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2

Existe também um método que filtra todos os dados ausentes e os altera para o valor desejado, tanto como texto quanto inteiros, exemplo



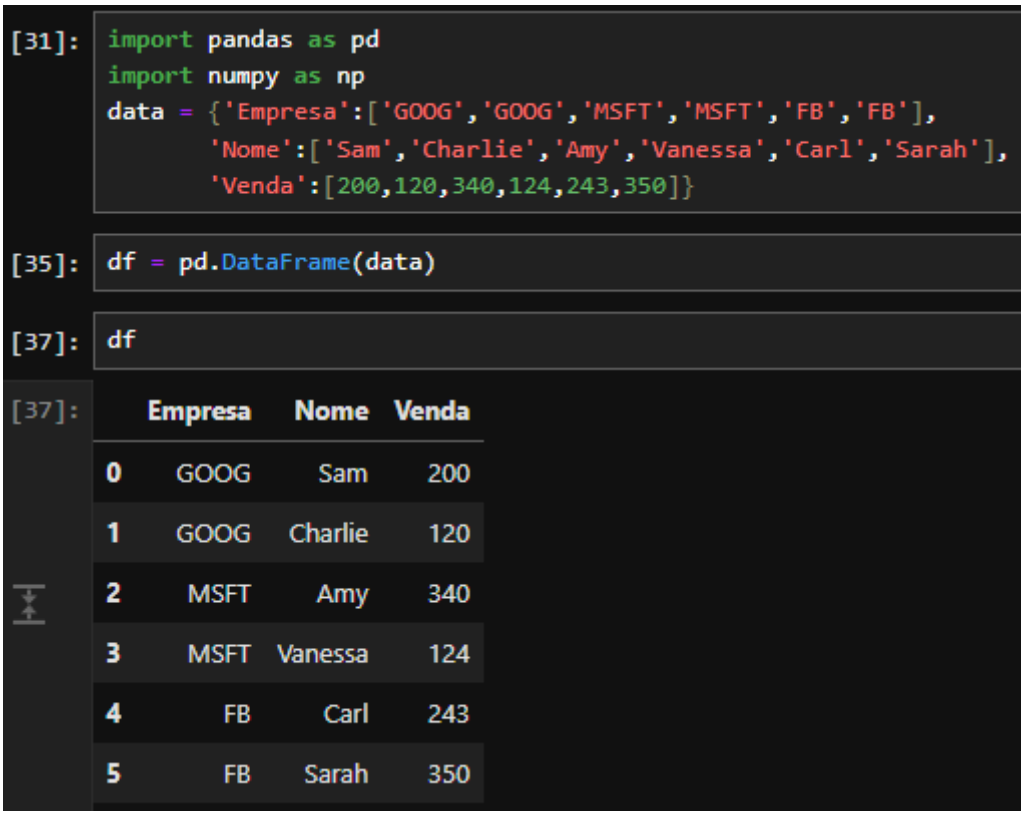
No primeiro comando ele altera todos os valores ausentes para -1, e no segundo ele filtra a coluna e altera os dados ausentes para a média dos dados existentes, como já foi explicado, mas esses comandos só se tornam permanentes se existir um ‘inplace = True’.

Uma das formas de tratar os dados ausentes é repondo eles segundo a última coleta, como no caso ele substitui pelo último dado que apareceu na coluna, exemplo:



Aula 31 GroupBy

O groupBy consiste em agrupar mesmos elementos em uma determinada coluna e realizar uma operação nas demais colunas.



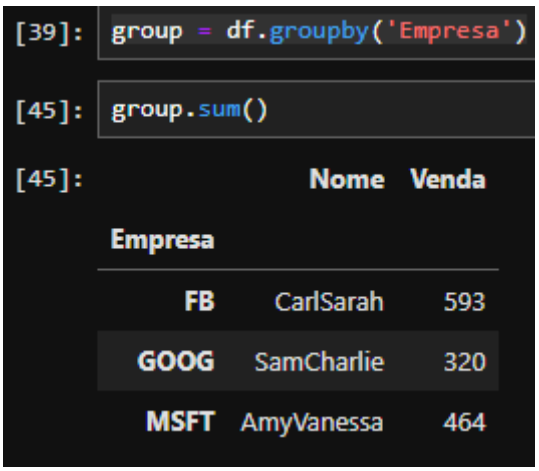
```
[31]: import pandas as pd
      import numpy as np
      data = {'Empresa':['GOOG','GOOG','MSFT','MSFT','FB','FB'],
              'Nome':['Sam','Charlie','Amy','Vanessa','Carl','Sarah'],
              'Venda':[200,120,340,124,243,350]}

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

[37]: df
```

[37]:

	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

Aqui foi apenas a criação do data frame usando o dicionário.

A partir deste dataframe podemos agrupar alguma coluna e fazer algumas operações, exemplo:



```
[39]: group = df.groupby('Empresa')

[45]: group.sum()
```

[45]:

Empresa	Nome	Venda
FB	CarlSarah	593
GOOG	SamCharlie	320
MSFT	AmyVanessa	464

Definimos o agrupamento pela coluna empresa, e ele filtra todas as linhas que contém o mesmo nome de empresa, e faz a soma das demais colunas, o total de vendas que cada empresa teve.

É possível detalhar cada aspecto de vendas também utilizando o .describe(), exemplo:

```
[57]: group.describe()
```

[57]:

	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

Aqui ele mostra vários parâmetros, como contagem de vezes que a empresa aparece, a média das vendas deles, o desvio padrão, as porcentagem,máximo e o mínimo. Como o python entende que aplicar isso na coluna ‘nomes’ que apresenta somente texto não faz sentido, ele apenas ignora e aplica na coluna ‘vendas’.

Tem como fazer a contagem em que cada atributo aparece em cada coluna:

```
[59]: group.count()
```

[59]:

	Nome	Venda
Empresa		
FB	2	2
GOOG	2	2
MSFT	2	2

Aqui mostra que aparece 2 nomes e 2 vendas para cada empresa.

E tem como ser ainda mais específico, exemplo

```
[61]: group = df.groupby('Nome')

[71]: group.sum().loc['Amy']

[71]: Empresa    MSFT
      Venda       340
      Name: Amy, dtype: object

[ ]:
```

No exemplo acima, ele se agrupa pela coluna ‘nome’,e depois disso ele soma as colunas em que o nome ’Amy’ aparece e retorna uma Series.

Aula 33 Concatenar, Juntar e mesclar.

Concatenação de dataframes é basicamente uma junção de dataframes, porém eles tem que ter dimensões correspondentes.

Mesclar é a criação de um dataframe que úne dataframes dado que eles tenham algum elemento em comum.

Juntar é juntar dataframes que potencialmente podem ter alguma diferença entre seus índices.

Concatenação:

Criação dos dataframes:

```
[2]: import pandas as pd

[14]: 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])
```

```
[16]: df1

[16]:
      A   B   C   D
0     A0  B0  C0  D0
1     A1  B1  C1  D1
2     A2  B2  C2  D2
3     A3  B3  C3  D3

[18]: df2

[18]:
      A   B   C   D
4     A4  B4  C4  D4
5     A5  B5  C5  D5
6     A6  B6  C6  D6
7     A7  B7  C7  D7

[20]: df3

[20]:
      A   B   C   D
8     A8  B8  C8  D8
9     A9  B9  C9  D9
10    A10 B10 C10 D10
11    A11 B11 C11 D11
```

A concatenação desses 3 dataframes com o eixo padrão 0

Criará um outro dataframe com as dimensões 12x4.

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

[22]:

	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

Aqui como padrão o eixo do comando é 0, porém se você alterar e colocar 1 ele muda a concatenação, exemplo:

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

[24]:

	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

Acontece assim pois os índices não correspondem da mesma maneira, criando esse dataframe 12x12

Mesclar:

```
[8]: 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']})
```

```
[10]: esquerda
```

[10]:

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

```
[12]: direita
```

[12]:

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

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

[14]:

	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

Aqui ele reúne as tabelas mantendo a coluna key como o elemento em comum entre eles.

Juntar:

```
[26]: 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'])

[28]: esquerda
```

[28]:

	A	B
K0	A0	B0
K1	A1	B1
K2	A2	B2

```
[30]: direita
```

[30]:

	C	D
K0	C0	D0
K2	C2	D2
K3	C3	D3

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

[32]:

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

O que está acontecendo no ‘juntar’ é, colocando os valores do dataframe ‘direita’ sempre que o índice dele for igual ao índice do dataframe ‘esquerda’.

Também é possível fazer assim:

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

[34]:

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2
K3	NaN	NaN	C3	D3

Ele pega todos os índices existentes dos dataframes e os preenche com valores não existentes caso não exista.

Aula 34 - Operações:

Nessa aula é trazido operações dentro da própria biblioteca do pandas.

Método de seleção de dados únicos, ele retorna os valores únicos que não se repetem dentro de uma coluna específica.

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

[1]:

	col1	col2	col3
0	1	444	abc
1	2	555	def
2	3	666	ghi
3	4	444	xyz

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

[3]: array([444, 555, 666], dtype=int64)
```

Da mesma forma dá para obter essa lista utilizando o numpy, exemplo:

```
[5]: import numpy as np

[7]: np.unique(df['col2'])

[7]: array([444, 555, 666], dtype=int64)
```

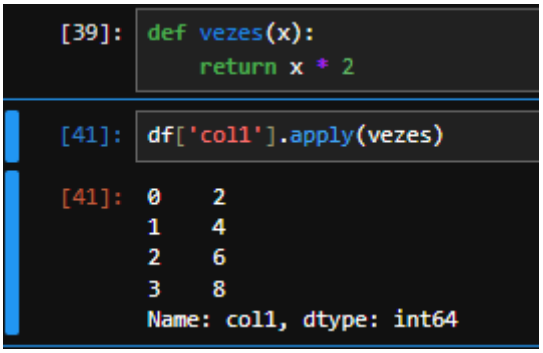
Dá para saber o tamanho da lista retornada usando o método abaixo:

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

[9]: 3
```

E existe outro método dentro do pandas que permite fazer as duas operações de uma vez só, exemplo:

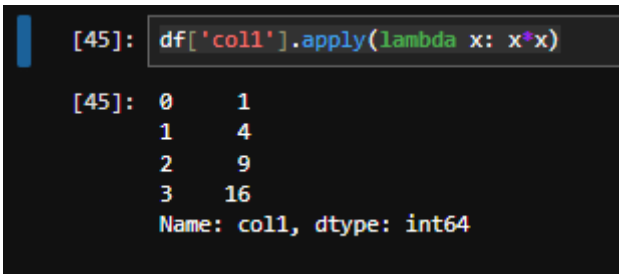
```
[11]: df['col2'].value_counts()

[11]: col2
      444    2
      555    1
      666    1
      Name: count, dtype: int64
```

É possível aplicar funções em dataframes como no seguinte exemplo:



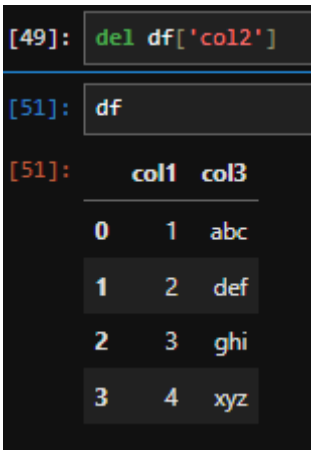
```
[39]: def vezes(x):
          return x * 2

[41]: df['col1'].apply(vezes)

[41]: 0    2
      1    4
      2    6
      3    8
      Name: col1, dtype: int64
```

Da mesma maneira dá para usar o lambda, exemplo:



```
[45]: df['col1'].apply(lambda x: x*x)

[45]: 0     1
      1     4
      2     9
      3    16
      Name: col1, dtype: int64
```

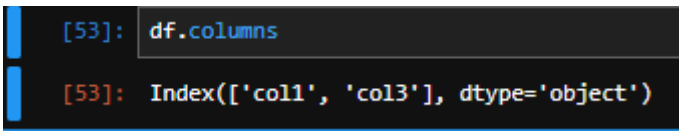
Para excluir colunas de maneira mais rápida, exemplo:



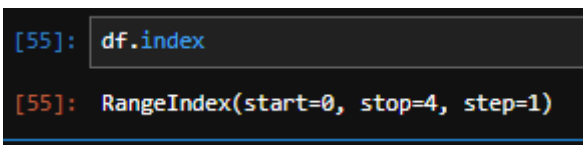
```
[49]: del df['col2']

[51]: df

[51]:
```

	col1	col3
0	1	abc
1	2	def
2	3	ghi
3	4	xyz

Para visualizar quais colunas ou índices estão presentes no dataframe, exemplo:

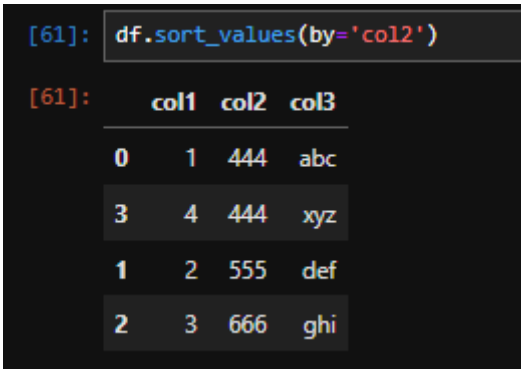


```
[53]: df.columns

[53]: Index(['col1', 'col3'], dtype='object')
```



```
[55]: df.index

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

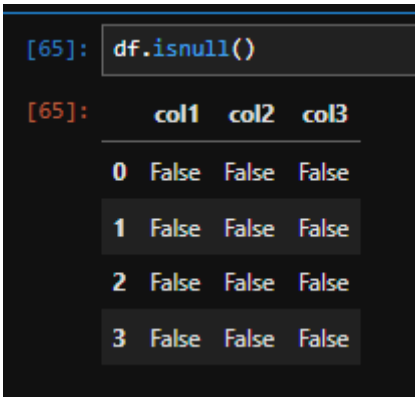
Conseguimos ordenar os valores das colunas, como no exemplo abaixo:



```
[61]: df.sort_values(by='col2')

[61]:
   col1  col2  col3
0     1   444   abc
3     4   444   xyz
1     2   555   def
2     3   666   ghi
```

Novamente para manter essa alteração é necessário passar um (inplace=True).

Outro método é a verificação de elementos nulos dentro do dataframe:



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

[65]:
   col1   col2   col3
0  False  False  False
1  False  False  False
2  False  False  False
3  False  False  False
```

Uma maneira de reorganizar seus dados para uma visualização mais fácil é transformados os valores iguais em uma coluna em índices de multinível, como no exemplo:

```
[67]: 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)
```

```
[69]: df
```

[69]:

	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

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

[71]:

		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

Aqui ele transforma os valores em comum facilitando a visualização da tabela.

Aula 35 - Entrada e Saída de dados.

Nessa aula aprendemos como que o pandas pode ser útil na entrada e na saída de dados facilitando muito.

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

[26]: df = pd.read_csv('exemplo', sep=',')

[28]: df

[28]:
```

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

```
[30]: df = df + 1

[32]: df

[32]:
```

	a	b	c	d
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

```
[34]: df.to_csv("exemplo.csv", sep=';', decimal=',')
```

No exemplo acima, eu importei um arquivo chamado ‘exemplo’ e criei um dataframe com os dados dentro dele, após isso somei um a todos os dados do dataframe e exportei ele como .csv e transformando sua separação em ;.

```
[50]: df = pd.read_html('http://www.fdic.gov/bank/individual/failed/banklist.html')

[54]: type(df)

[54]: list

[58]: df[0]['Bank Name']

[58]: 0       The First National Bank of Lindsay
      1    Republic First Bank dba Republic Bank
      2                         Citizens Bank
      3               Heartland Tri-State Bank
      4                    First Republic Bank
      5                         Signature Bank
      6                    Silicon Valley Bank
      7                      Almena State Bank
      8              First City Bank of Florida
      9                    The First State Bank
      Name: Bank Name, dtype: object
```

No caso acima o pandas permite acessar dados através de um link html, ele retorna uma lista, porém dá para fazer um tratamento de dados com mais facilidade.

Conclusões

A conclusão que eu chego com este card é que, com as bibliotecas pandas e numpy a análise e a manipulação de dados ficam cada vez mais facilitadas se você souber utilizar essas 2 bibliotecas do jeito certo, e acredito que isso seja de suma importância para a área de ciência de dados, aprendizado de máquina na qual este bootcamp é focado.

Referências

https://ebaconline.com.br/blog/jupyter-notebook-o-que-e-e-como-se-usa-seo

https://numpy.org/doc/2.1/index.html