Numpy Manipulation.ipynb ☆

Arquivo Editar Ver Inserir Ambiente de execução Ferramentas Ajuda Todas as alterações foram salvas

Comentário Compartilhar

+ Código + Texto

RAM Disco

Trabalhando com a biblioteca Numpy

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

[2]
```
#características do dataset
#7 anos e 3 meses de dados
7*12+3
```

```
87
```

[3]
```
#dadas as características acima preciso gerar um array para excluir a primeira coluna de strings
colunas_dataframe = np.arange(1,88,1)
colunas_dataframe
```

```
array([ 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, 50, 51,
       52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
       69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85,
       86, 87])
```

[4]
```
dados = np.loadtxt('apples_ts.csv', delimiter=',', usecols=colunas_dataframe)
dados
```

```
          45.85  ,  49.25  ,  48.14  ,  49.18  ,  49.49  ,  49.83  ,
          49.14  ,  63.21  ,  61.2   ,  60.34  ,  62.2   ,  74.2   ,
          75.32  ,  75.92  ,  72.89  ,  74.85  ,  92.19  ,  99.7   ,
         102.78  ,  92.67  ,  90.63  ,  83.31  ,  75.5   ,  74.6   ,
          76.94  ,  83.04  ,  80.74  ,  80.29  ,  82.82  ,  86.29  ,
          87.41  ,  85.1   ,  78.08  ,  76.06  ,  69.23  ,  69.04  ,
          64.48  ,  65.73  ,  67.2   ,  70.25  ,  82.59  ,  92.12  ,
         100.82  ,  95.23  ,  89.51  ,  82.84  ,  82.3   ,  83.76  ,
          84.72  ,  83.91  ,  99.14  , 101.6   , 113.03  , 119.54  ,
         118.09  , 107.32  ,  82.6   ,  77.4   ,  71.77  ,  71.25  ,
          71.35  ,  73.62  ,  76.2   ,  74.15  ,  75.67  ,  79.2   ,
          80.85  ,  85.33  ,  75.02  ,  77.95  ,  78.98  ,  76.55  ,
          74.89  ,  70.    ,  81.53  ],
        [ 62.55  ,  62.73  ,  63.43  ,  63.83  ,  66.06  ,  69.22  ,
          72.07  ,  69.31  ,  65.18  ,  62.13  ,  64.17  ,  65.5   ,
          66.85  ,  68.38  ,  68.77  ,  69.32  ,  69.77  ,  70.17  ,
          70.76  ,  75.16  ,  74.86  ,  71.47  ,  76.08  ,  82.11  ,
          94.63  ,  98.54  ,  97.84  ,  94.54  ,  97.88  ,  97.73  ,
         100.89  , 104.88  , 104.39  , 101.9   ,  98.99  ,  99.42  ,
         106.37  , 113.89  , 114.78  , 115.6   , 115.89  , 117.66  ,
         118.27  , 114.19  , 101.81  ,  91.62  ,  90.15  ,  90.55  ,
          92.3   ,  91.75  ,  92.    ,  92.57  , 100.28  , 114.33  ,
         122.17  , 117.83  , 112.49  ,  93.7   ,  97.17  ,  95.64  ,
         102.21  , 107.58  , 110.48  , 117.73  , 122.07  , 128.58  ,
         131.12  , 127.08  , 110.99  ,  97.43  ,  92.05  ,  93.21  ,
          98.58  , 100.23  , 102.09  , 101.73  , 107.28  , 115.35  ,
         123.03  , 123.08  , 109.71  ,  97.22  ,  95.75  ,  97.09  ,
          98.18  , 103.1   , 110.6   ],
        [ 48.26  ,  51.01  ,  50.91  ,  53.94  ,  61.27  ,  65.44  ,
          56.51  ,  53.    ,  43.87  ,  42.12  ,  43.98  ,  44.66  ,
          47.73  ,  52.31  ,  56.8   ,  51.15  ,  54.63  ,  55.8   ,
          57.31  ,  53.81  ,  55.5   ,  52.95  ,  53.29  ,  54.86  ,
          71.72  ,  74.36  ,  78.25  ,  70.63  ,  77.06  ,  81.52  ,
          84.12  ,  83.33  ,  66.98  ,  62.04  ,  59.79  ,  66.06  ,
          71.99  ,  76.79  ,  80.98  ,  89.18  ,  92.71  ,  94.85  ,
          99.18  ,  86.33  ,  71.94  ,  67.18  ,  62.98  ,  69.45  ,
          67.56  ,  67.92  ,  68.4   ,  67.97  ,  76.33  ,  95.79  ,
         112.36  ,  82.03  ,  73.83  ,  66.12  ,  63.24  ,  63.98  ,
          69.1   ,  70.04  ,  80.03  ,  87.92  ,  97.07  , 106.52  ,
         108.93  ,  95.17  ,  74.31  ,  62.63  ,  69.92  ,  70.58  ,
          74.63  ,  78.69  ,  81.82  ,  82.36  ,  89.44  , 102.01  ,
         116.12  ,  92.06  ,  82.7   ,  66.62  ,  68.11  ,  73.48  ,
          82.04  ,  81.19  ,  95.05  ],
        [ 71.25  ,  71.35  ,  70.9   ,  71.92  ,  72.91  ,  74.39  ,
          73.1   ,  70.24  ,  69.12  ,  68.98  ,  68.58  ,  69.5   ,
          70.14  ,  70.58  ,  70.78  ,  72.78  ,  72.84  ,  74.74  ,
          76.43  ,  81.2   ,  82.04  ,  79.67  ,  81.25  ,  85.88  ,
          91.35  , 101.32  , 102.45  ,  98.18  , 105.1   , 104.85  ,
         111.48  , 118.51  , 118.92  , 109.87  , 105.22  , 105.45  ,
```

```
        115.61  , 116.61  , 120.39  , 119.2   , 119.92  , 120.57  ,
        123.37  , 121.39  , 118.16  , 108.84  , 100.78  ,  99.92  ,
        100.65  ,  98.92  , 100.31  ,  96.41  , 101.24  , 118.7   ,
        133.29  , 130.31  , 122.4   , 104.98  , 107.54  , 110.74  ,
        110.51  , 111.46  , 113.42  , 118.15  , 114.    , 124.91  ,
        126.06  , 123.3   , 111.61  ,  98.82  ,  97.47  , 103.01  ,
        109.36  , 104.79  , 105.09  , 103.16  , 106.52  , 121.68  ,
        125.32  , 123.41  , 108.48  ,  98.73  ,  96.25  , 100.12  ,
        101.29  , 102.81  , 108.53  ]])
```

[5]
```
#diferença entre Lista e Array em Python
import numpy as np

# cria uma lista
lista = [1, 2, 3, 4, 5]

# transforma a lista em um array Numpy
array = np.array(lista)

print("Lista: ", lista)
print("Array: ", array)
```

```
Lista:  [1, 2, 3, 4, 5]
Array:  [1 2 3 4 5]
```

[6]
```
import time

# cria uma lista com 1000000 elementos
lista = list(range(1000000))

# transforma a lista em um array Numpy
array = np.array(lista)

# começa a cronometrar o tempo para a operação com a lista
start_time = time.time()

# realiza a operação de elevar ao quadrado cada elemento da lista
lista_quadrado = [i**2 for i in lista]

# para o cronômetro
list_time = time.time() - start_time

# começa a cronometrar o tempo para a operação com o array
start_time = time.time()

# realiza a operação de elevar ao quadrado cada elemento do array
array_quadrado = array**2

# para o cronômetro
array_time = time.time() - start_time

print("Tempo da operação com a lista: ", list_time)
print("Tempo da operação com o array: ", array_time)
```

```
Tempo da operação com a lista:  0.320446252822876
Tempo da operação com o array:  0.003691434860229492
```

[7]
```
dados.ndim
```

```
2
```

[8]
```
dados.size
```

```
522
```

[9]
```
dados.shape
```

```
(6, 87)
```

[10]
```
#transposição
dados_transpostos = dados.T
dados_transpostos
```

```
      [  6.2015,  97.29  ,  99.7   ,  97.73  ,  81.52  , 104.85  ],
      [  7.2015,  98.64  , 102.78  , 100.89  ,  84.12  , 111.48  ],
      [  8.2015, 104.26  ,  92.67  , 104.88  ,  83.33  , 118.51  ],
      [  9.2015, 102.63  ,  90.63  , 104.39  ,  66.98  , 118.92  ],
      [ 10.2015,  98.64  ,  83.31  , 101.9   ,  62.04  , 109.87  ],
      [ 11.2015,  97.17  ,  75.5   ,  98.99  ,  59.79  , 105.22  ],
      [ 12.2015,  98.09  ,  74.6   ,  99.42  ,  66.06  , 105.45  ],
      [  1.2016, 103.07  ,  76.94  , 106.37  ,  71.99  , 115.61  ],
      [  2.2016, 110.26  ,  83.04  , 113.89  ,  76.79  , 116.61  ],
      [  3.2016, 110.84  ,  80.74  , 114.78  ,  80.98  , 120.39  ],
      [  4.2016, 112.28  ,  80.29  , 115.6   ,  89.18  , 119.2   ],
      [  5.2016, 111.1   ,  82.82  , 115.89  ,  92.71  , 119.92  ],
      [  6.2016, 110.06  ,  86.29  , 117.66  ,  94.85  , 120.57  ],
      [  7.2016, 113.7   ,  87.41  , 118.27  ,  99.18  , 123.37  ],
      [  8.2016, 112.88  ,  85.1   , 114.19  ,  86.33  , 121.39  ],
      [  9.2016, 102.08  ,  78.08  , 101.81  ,  71.94  , 118.16  ],
      [ 10.2016,  95.54  ,  76.06  ,  91.62  ,  67.18  , 108.84  ],
      [ 11.2016,  91.33  ,  69.23  ,  90.15  ,  62.98  , 100.78  ],
      [ 12.2016,  89.99  ,  69.04  ,  90.55  ,  69.45  ,  99.92  ],
      [  1.2017,  91.44  ,  64.48  ,  92.3   ,  67.56  , 100.65  ],
```

```
[  2.2017,  93.51  ,  65.73  ,  91.75  ,  67.92  ,  98.92  ],
[  3.2017,  93.6   ,  67.2   ,  92.    ,  68.4   , 100.31  ],
[  4.2017,  93.78  ,  70.25  ,  92.57  ,  67.97  ,  96.41  ],
[  5.2017,  98.91  ,  82.59  , 100.28  ,  76.33  , 101.24  ],
[  6.2017, 121.76  ,  92.12  , 114.33  ,  95.79  , 118.7   ],
[  7.2017, 129.6   , 100.82  , 122.17  , 112.36  , 133.29  ],
[  8.2017, 127.9   ,  95.23  , 117.83  ,  82.03  , 130.31  ],
[  9.2017, 114.55  ,  89.51  , 112.49  ,  73.83  , 122.4   ],
[ 10.2017, 101.88  ,  82.84  ,  93.7   ,  66.12  , 104.98  ],
[ 11.2017,  99.09  ,  82.3   ,  97.17  ,  63.24  , 107.54  ],
[ 12.2017, 103.35  ,  83.76  ,  95.64  ,  63.98  , 110.74  ],
[  1.2018, 106.58  ,  84.72  , 102.21  ,  69.1   , 110.51  ],
[  2.2018, 108.    ,  83.91  , 107.58  ,  70.04  , 111.46  ],
[  3.2018, 114.95  ,  99.14  , 110.48  ,  80.03  , 113.42  ],
[  4.2018, 121.17  , 101.6   , 117.73  ,  87.92  , 118.15  ],
[  5.2018, 122.48  , 113.03  , 122.07  ,  97.07  , 114.    ],
[  6.2018, 127.58  , 119.54  , 128.58  , 106.52  , 124.91  ],
[  7.2018, 131.89  , 118.09  , 131.12  , 108.93  , 126.06  ],
[  8.2018, 129.36  , 107.32  , 127.08  ,  95.17  , 123.3   ],
[  9.2018, 104.26  ,  82.6   , 110.99  ,  74.31  , 111.61  ],
[ 10.2018,  93.45  ,  77.4   ,  97.43  ,  62.63  ,  98.82  ],
[ 11.2018,  92.93  ,  71.77  ,  92.05  ,  69.92  ,  97.47  ],
[ 12.2018,  96.15  ,  71.25  ,  93.21  ,  70.58  , 103.01  ],
[  1.2019,  99.1   ,  71.35  ,  98.58  ,  74.63  , 109.36  ],
[  2.2019, 103.    ,  73.62  , 100.23  ,  78.69  , 104.79  ],
[  3.2019, 103.31  ,  76.2   , 102.09  ,  81.82  , 105.09  ],
[  4.2019, 103.01  ,  74.15  , 101.73  ,  82.36  , 103.16  ],
[  5.2019, 107.37  ,  75.67  , 107.28  ,  89.44  , 106.52  ],
[  6.2019, 116.91  ,  79.2   , 115.35  , 102.01  , 121.68  ],
[  7.2019, 125.29  ,  80.85  , 123.03  , 116.12  , 125.32  ],
[  8.2019, 123.94  ,  85.33  , 123.08  ,  92.06  , 123.41  ],
[  9.2019, 113.03  ,  75.02  , 109.71  ,  82.7   , 108.48  ],
[ 10.2019, 102.19  ,  77.95  ,  97.22  ,  66.62  ,  98.73  ],
[ 11.2019,  97.83  ,  78.98  ,  95.75  ,  68.11  ,  96.25  ],
[ 12.2019, 101.07  ,  76.55  ,  97.09  ,  73.48  , 100.12  ],
[  1.202 , 103.44  ,  74.89  ,  98.18  ,  82.04  , 101.29  ],
[  2.202 , 108.23  ,  70.    , 103.1   ,  81.19  , 102.81  ],
[  3.202 , 110.28  ,  81.53  , 110.6   ,  95.05  , 108.53  ]])
```

[11]
```
datas = dados_transpostos[:,0]
datas.shape
```

```
(87,)
```

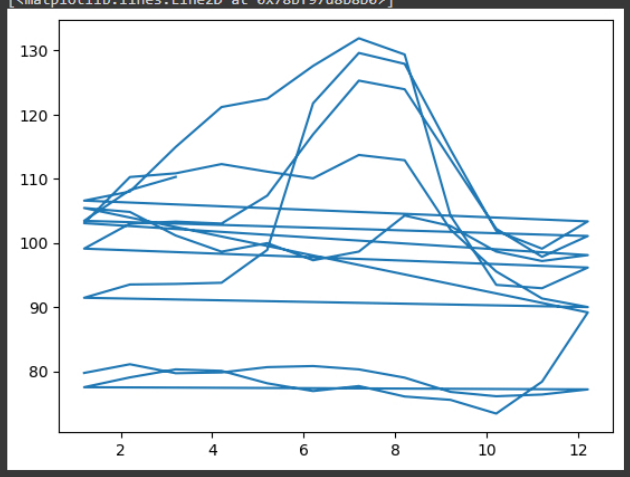
[12]
```
precos = dados_transpostos[:,1:6]
precos.shape
```

```
(87, 5)
```

[13]
```
import matplotlib.pyplot as plt
```

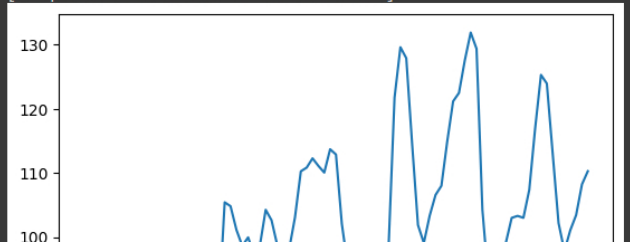
[14]
```
plt.plot(datas, precos[:,0])
```

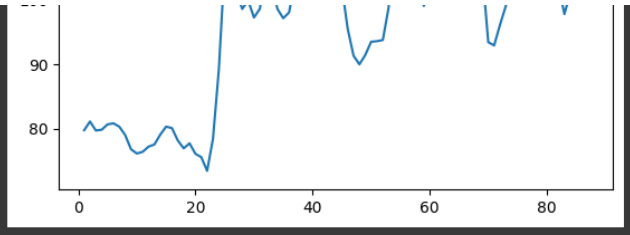
```
[<matplotlib.lines.Line2D at 0x78bf97d8b8b0>]
```



[15]
```
datas = np.arange(1, 88, 1)
plt.plot(datas, precos[:,0])

#eu não estou mais trabalhando com valores de datas na variável "datas"
#simplesmente gerei um array para servir de base de plotagem para o eixo x.
```

```
[<matplotlib.lines.Line2D at 0x78bf94c1d4b0>]
```



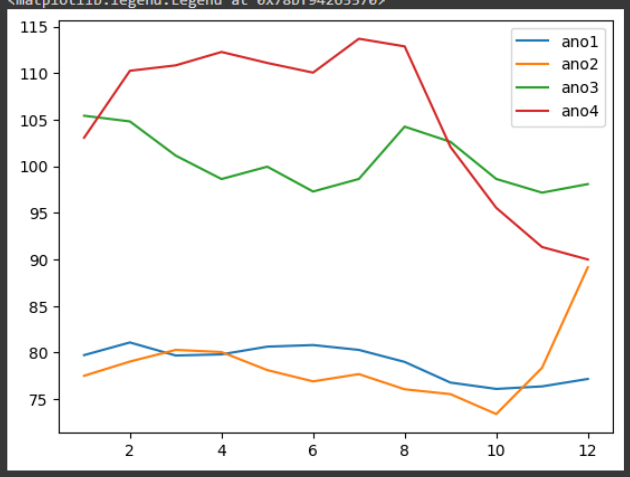



```
[16] Moscow = precos[:,0]
     Kaliningrad = precos[:,1]
     Petersburg = precos[:,2]
     Krasnodar = precos[:,3]
     Ekaterinburg = precos[:,4]
```

```
[17] # segmentando por ano para comparações sazonais
     Moscow_ano1 = Moscow[0:12]
     Moscow_ano2 = Moscow[12:24]
     Moscow_ano3 = Moscow[24:36]
     Moscow_ano4 = Moscow[36:48]
```

```
[18] plt.plot(np.arange(1,13,1), Moscow_ano1)
     plt.plot(np.arange(1,13,1), Moscow_ano2)
     plt.plot(np.arange(1,13,1), Moscow_ano3)
     plt.plot(np.arange(1,13,1), Moscow_ano4)
     plt.legend(['ano1', 'ano2', 'ano3', 'ano4'])
```

```
<matplotlib.legend.Legend at 0x78bf94265570>
```




```
[19] np.array_equal(Moscow_ano3, Moscow_ano4)
```

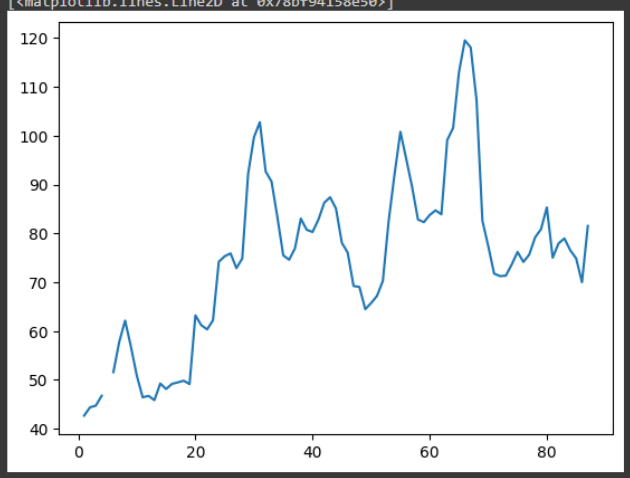
```
False
```

```
[20] np.allclose(Moscow_ano1, Moscow_ano2, 0.01)
```

```
False
```

```
[21] plt.plot(datas, Kaliningrad)
```

```
[<matplotlib.lines.Line2D at 0x78bf94158e50>]
```




```
[22] Kaliningrad
```

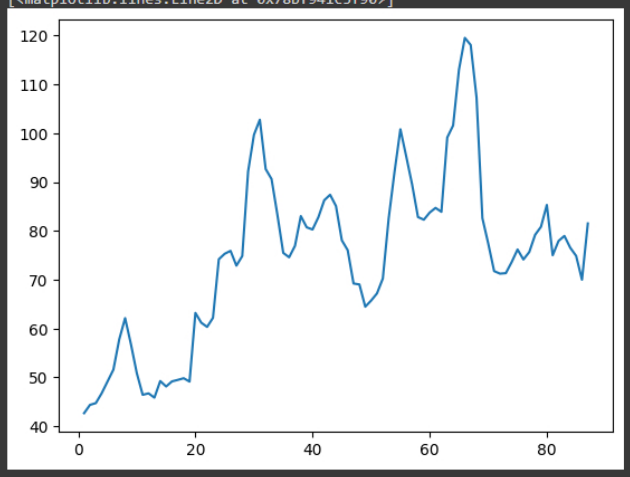
```
array([ 42.67,  44.37,  44.73,  46.73,    nan,  51.59,  57.8 ,  62.14,
        56.76,  50.85,  46.43,  46.73,  45.85,  49.25,  48.14,  49.18,
        49.49,  49.83,  49.14,  63.21,  61.2 ,  60.34,  62.2 ,  74.2 ,
        75.32,  75.92,  72.89,  74.85,  92.19,  99.7 , 102.78,  92.67,
        90.63,  83.31,  75.5 ,  74.6 ,  76.94,  83.04,  80.74,  80.29,
        82.82,  86.29,  87.41,  85.1 ,  78.08,  76.06,  69.23,  69.04,
        64.48,  65.73,  67.2 ,  70.25,  82.59,  92.12, 100.82,  95.23,
        89.51,  82.84,  82.3 ,  83.76,  84.72,  83.91,  99.14, 101.6 ,
       113.03, 119.54, 118.09, 107.32,  82.6 ,  77.4 ,  71.77,  71.25,
        71.35,  73.62,  76.2 ,  74.15,  75.67,  79.2 ,  80.85,  85.33,
        75.02,  77.95,  78.98,  76.55,  74.89,  70.  ,  81.53])
```

[23]
```
sum(np.isnan(Kaliningrad))
```
```
1
```

[24]
```
(Kaliningrad[3]+Kaliningrad[5])/2
```
```
49.17
```

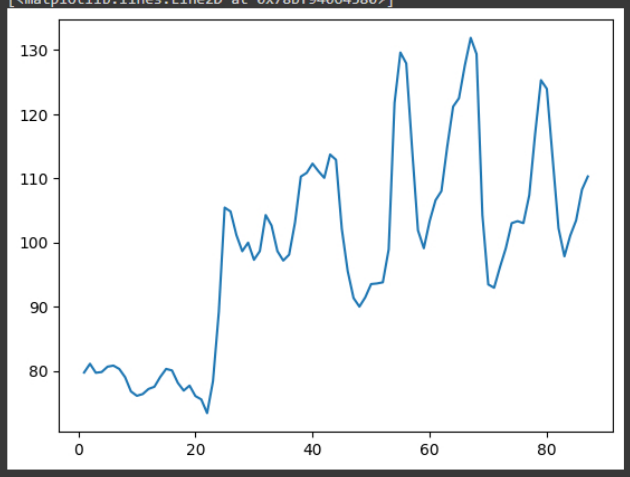
[25]
```
#ou
np.mean([Kaliningrad[3], Kaliningrad[5]])
```
```
49.17
```

[26]
```
#substituindo o NaN
Kaliningrad[4] = np.mean([Kaliningrad[3], Kaliningrad[5]])
```

[27]
```
plt.plot(datas, Kaliningrad)
```
```
[<matplotlib.lines.Line2D at 0x78bf941c5f90>]
```


[28]
```
np.mean(Moscow)
```
```
98.79781609195402
```

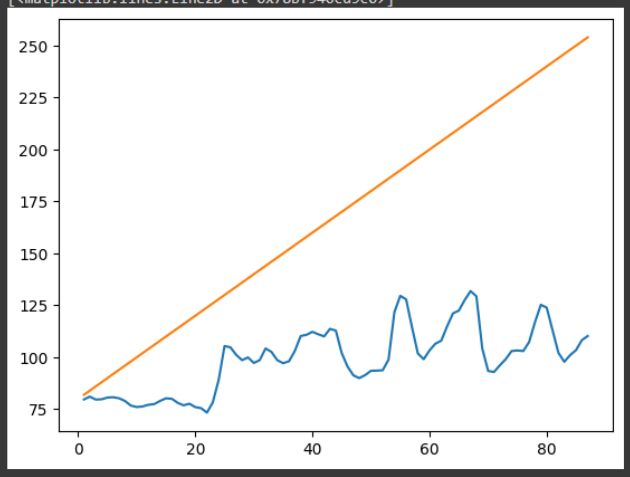
[29]
```
np.mean(Kaliningrad)
```
```
74.5506896551724
```

[30]
```
# ajustando uma reta para os preços
# coeficiente angular: y = ax + b, onde o y é o preço, a é o coeficiente angular e b o coeficiente linear.
plt.plot(datas, Moscow)
```
```
[<matplotlib.lines.Line2D at 0x78bf94064580>]
```


[31]
```
# apenas supondo valores para as variáveis x e y
```

```
x = datas
y = 2*x + 80

plt.plot(datas, Moscow)
plt.plot(x, y)
```

```
[<matplotlib.lines.Line2D at 0x78bf940cd9c0>]
```

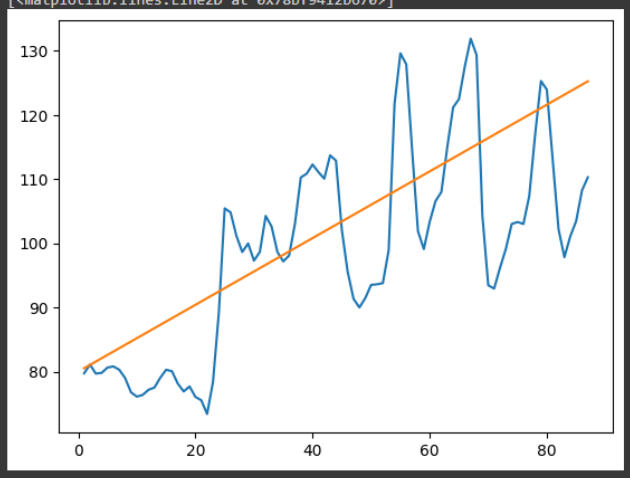



```
[32] # eu preciso ajustar a reta aos meu dados
     # para isso eu subtraio Moscow de y e elevo ao quadrado para eliminar os números negativos
     # após a elevação ao quadrado, eu terei números muito grandes, então é necessário um ajuste com soma.

     np.sqrt(np.sum(np.power(Moscow - y, 2)))
```

```
749.2329171225728
```

```
[33] # refazendo os cálculos com outros valores de reta

     y = 0.52*x + 80
     plt.plot(datas, Moscow)
     plt.plot(x, y)
```

```
[<matplotlib.lines.Line2D at 0x78bf9412b670>]
```




```
[34] np.sqrt(np.sum(np.power(Moscow - y, 2)))
```

```
110.48504740461489
```

```
[35] # refazendo o mesmo cálculo com função do numpy
     np.linalg.norm(Moscow - y)
```

```
110.48504740461489
```

Adendo: Cópia ou Referência de arrays?

```
[36] # Cópias referenciadas: alteração em uma, altera outra
     preco_imoveis = np.array([10000,120000,11000,200000])
     preco_imoveis_sao_paulo = preco_imoveis
```

```
[37] preco_imoveis[0] = 120000
```

```
[38] preco_imoveis_sao_paulo
```

```
array([120000, 120000,  11000, 200000])
```

```
[39] # Cópias não referenciadas
     preco_imoveis = np.array([10000,120000,11000,200000])
     preco_imoveis_sao_paulo = np.copy(preco_imoveis) # <---
     preco_imoveis[0] = 120000
     preco_imoveis_sao_paulo
```

```
array([ 10000, 120000,  11000, 200000])
```

continuando...

```
[40] # trabalhando com os dados reais de preços
     Y = Moscow
     x = datas
     n = np.size(Moscow)
```

```
[41] (x**2).shape
```

```
(87,)
```

```
[42] # escrevendo a fórmula do coeficiente angular
     a = (n*np.sum(x*Y) - np.sum(x)*np.sum(Y)) / (n*np.sum(x**2) - np.sum(x)**2)
```

```
[43] # escrevendo a fórmula do coeficiente linear
     b = np.mean(Y) - a*np.mean(x)
```

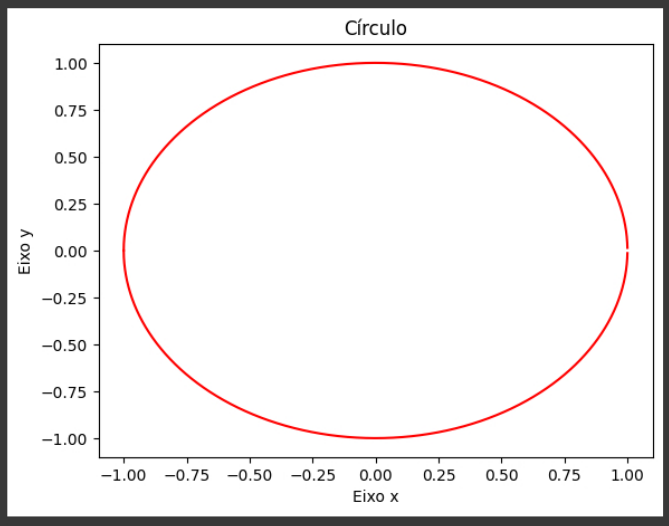
```
[44] y = a*x + b
```

```
[45] np.linalg.norm(Moscow-y)
```

```
101.7969539992751
```

exercício de plotagem de círculo

```
[46] # Gerar uma sequência de valores de x de -1 a 1
     x = np.arange(-1, 1, 0.0001)

     # Implementação da fórmula
     y1 = np.sqrt(1 - x**2)
     y2 = -np.sqrt(1 - x**2)

     # Plotar o gráfico com as duas partes do círculo
     plt.plot(x, y1, 'r')
     plt.plot(x, y2, 'r' )

     # Adicionar o título do gráfico e os rótulos dos eixos x e y
     plt.title("Círculo")
     plt.xlabel("Eixo x")
     plt.ylabel("Eixo y")

     # Exibir o gráfico
     plt.show()
```

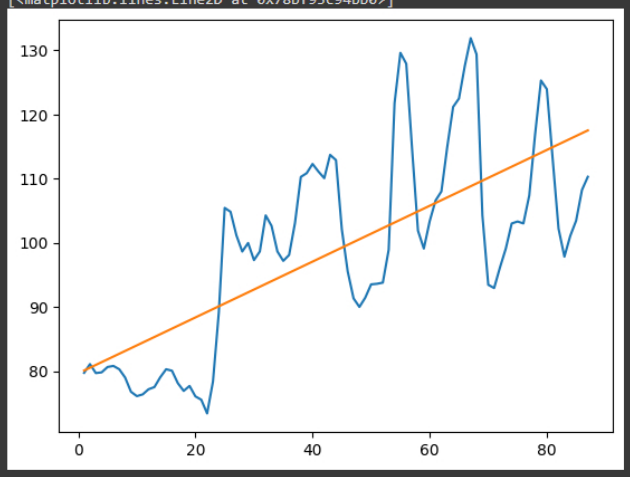


continuando com regressão liner

```
[49] # plotagem da regressão linear calculada corretamente
     Y = Moscow
     x = datas
     n = np.size(Moscow)

     (x**2).shape
```

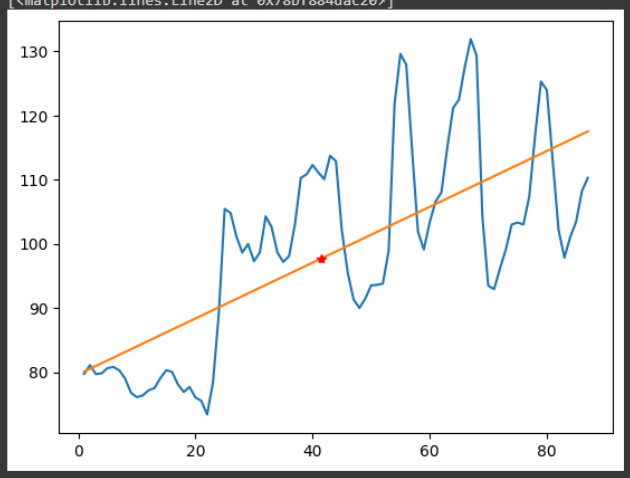
```
a = (n*np.sum(x*Y) - np.sum(x)*np.sum(Y)) / (n*np.sum(x**2) - np.sum(x)**2)

b = np.mean(Y) - a*np.mean(x)

y = a*x + b

plt.plot(datas, Moscow)
plt.plot(x, y)
```

```
[<matplotlib.lines.Line2D at 0x78bf93c94bb0>]
```



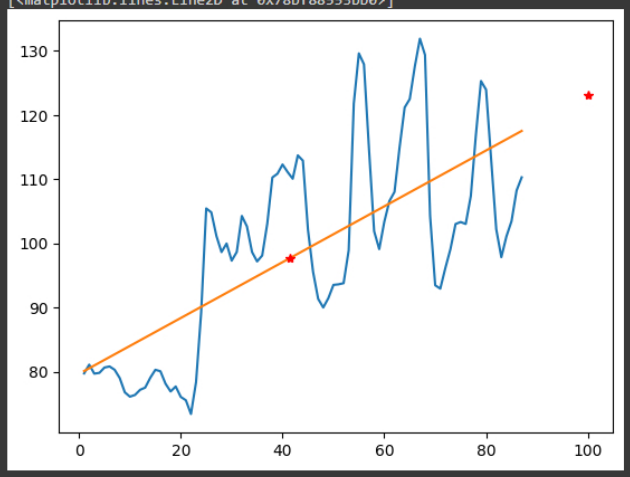
[51]
```
# para plotar apenas um ponto no gráfico
plt.plot(datas, Moscow)
plt.plot(x, y)
plt.plot(41.5,41.5*a+b,'*r')
```

```
[<matplotlib.lines.Line2D at 0x78bf884dac20>]
```



[52]
```
# projetando a reta para previsão de valores
plt.plot(datas, Moscow)
plt.plot(x, y)
plt.plot(41.5,41.5*a+b,'*r')
#altero valores para 100 (máximo 87)
plt.plot(100,100*a+b,'*r')
```

```
[<matplotlib.lines.Line2D at 0x78bf88553bb0>]
```



trabalhando com aleatoriedade

```
[53] # gerando números inteiros aleatórios
     np.random.randint(low=40, high=100, size=100)
```

```
array([55, 52, 62, 80, 98, 96, 84, 91, 90, 58, 63, 83, 90, 72, 51, 63, 77,
       57, 79, 82, 52, 55, 52, 81, 54, 97, 82, 40, 82, 44, 77, 48, 75, 68,
       62, 62, 72, 84, 43, 72, 72, 79, 76, 86, 73, 99, 45, 47, 41, 78, 70,
       53, 51, 44, 90, 58, 59, 80, 87, 90, 86, 93, 44, 56, 49, 94, 41, 78,
       49, 68, 93, 91, 79, 92, 54, 50, 49, 69, 85, 70, 89, 92, 68, 49, 42,
       82, 60, 45, 85, 66, 73, 56, 58, 79, 84, 88, 71, 76, 40, 89])
```

```
[54] # gerando números float aleatórios
     np.random.uniform(low=0.10, high=0.90, size=100)
```

```
array([0.80184047, 0.28105119, 0.63495476, 0.14638628, 0.26925898,
       0.82874877, 0.71296087, 0.20745547, 0.24686278, 0.43508836,
       0.25458102, 0.11836423, 0.30123979, 0.70811736, 0.85461215,
       0.37981964, 0.34363767, 0.89442305, 0.20976168, 0.132793  ,
       0.68627765, 0.28155431, 0.57164003, 0.82233307, 0.63662586,
       0.88361555, 0.1275358 , 0.21503829, 0.43798102, 0.43798976,
       0.65869389, 0.86752676, 0.5623516 , 0.1877705 , 0.45047816,
       0.13017819, 0.46212497, 0.31086822, 0.28443036, 0.35090549,
       0.26999819, 0.8069876 , 0.51539028, 0.33754001, 0.12068728,
       0.47693865, 0.47756444, 0.78028976, 0.66468176, 0.68768448,
       0.88741179, 0.10582072, 0.74690756, 0.25145515, 0.44113542,
       0.68488001, 0.45558659, 0.89373706, 0.69543613, 0.56562004,
       0.71311167, 0.13691203, 0.33958534, 0.42052394, 0.43559358,
       0.21777936, 0.19887186, 0.60000471, 0.19801875, 0.44357298,
       0.62318445, 0.52131239, 0.55098368, 0.87665377, 0.71631875,
       0.47218768, 0.374013  , 0.33008675, 0.10941211, 0.43602839,
       0.12693923, 0.63166349, 0.49670593, 0.49353516, 0.17374747,
       0.8158285 , 0.13581216, 0.27852468, 0.52559458, 0.29711451,
       0.6764442 , 0.45839564, 0.46296011, 0.23522712, 0.89441771,
       0.30057862, 0.28805054, 0.57561912, 0.14947784, 0.85646977])
```

```
[55] # testando coeficiente angular com esses números aleatórios
     coef_angulares = np.random.uniform(low=0.10, high=0.90, size=100)
```

```
[56] for i in range(100):
       print(np.linalg.norm(Moscow - (coef_angulares[i]*x+b)))
```

```
135.42711409580505
117.43401249964366
203.53196228325183
150.2465219585277
124.02717506416148
116.99094953870367
106.40189123489117
141.22452698758542
174.18919891216316
166.5807465821204
180.0419120004132
102.69096628407006
137.40722484664536
149.78000729876882
156.2792804299463
122.97447302040547
108.60076609596442
109.32451223756152
148.645077977689
162.11415266404893
146.80453481493868
154.28840741591068
104.10086590276241
107.14991014644899
170.85680975976726
101.8473372786546
130.7506771111329
161.29663487137086
138.8712054568994
214.77656758718618
214.39275969269042
112.05731790544765
126.7505940237295
138.46146653854564
116.41469892903108
226.8553473843902
106.30570467009734
206.76672753657496
131.3493810071654
102.9284919091172
171.46955632279372
124.56741483049458
145.8665080784717
113.26190824079261
145.70431190004
115.16977350999927
106.45200903790047
103.63107476456527
207.47279650331646
107.93878382436365
240.27676000337613
213.63632679176465
161.25215371799004
164.68851130060915
106.39346726861051
186.87991347005214
```

```
226.04261426590512
174.88522121325357
```

```
[58] # agregando esses valores a um array
     norma = np.array([])

     for i in range(100):
       norma = np.append(norma, np.linalg.norm(Moscow - (coef_angulares[i]*x+b)))

     norma
```

```
array([147.10101138, 101.8165657 , 172.01403888, 147.99170917,
       156.86381528, 223.69121172, 141.07902887, 193.35045219,
       148.04348404, 130.00445928, 143.97411878, 134.59639439,
       136.07395798, 180.3292952 , 179.10964793, 104.17723111,
       135.90188801, 119.46407591, 102.03555571, 167.86744578,
       217.0544629 , 155.04579349, 101.84307135, 159.10522417,
       130.20979449, 173.32242094, 151.97406227, 177.70750684,
       163.17110361, 109.89989274, 102.27580584, 105.4974603 ,
       110.81321053, 103.26271362, 131.33557346, 106.26862509,
       148.63307053, 119.07003563, 143.20540066, 126.19084086,
       129.44787843, 176.62857077, 135.4271141 , 117.4340125 ,
       203.53196228, 150.24652196, 124.02717506, 116.99094954,
       106.40189123, 141.22452699, 174.18919891, 166.58074658,
       180.041912  , 102.69096628, 137.40722485, 149.7800073 ,
       156.27928043, 122.97447302, 108.6007661 , 109.32451224,
       148.64507798, 162.11415266, 146.80453481, 154.28840742,
       104.1008659 , 107.14991015, 170.85680976, 101.84733728,
       130.75067711, 161.29663487, 138.87120546, 214.77656759,
       214.39275969, 112.05731791, 126.75059402, 138.46146654,
       116.41469893, 226.85534738, 106.30570467, 206.76672754,
       131.34938101, 102.92849191, 171.46955632, 124.56741483,
       145.86650808, 113.26190824, 145.7043119 , 115.16977351,
       106.45200904, 103.63107476, 207.4727965 , 107.93878382,
       240.27676   , 213.63632679, 161.25215372, 164.6885113 ,
       106.39346727, 186.87991347, 226.04261427, 174.88522121])
```

```
[59] # testando o coeficiente angular em qualquer posição do array
     coef_angulares[1]
```

```
0.4394309162204504
```

```
[60] #reprodutibilidade - SEED
     #travando a aleatoriedade gerada

     np.random.seed(16)
     np.random.uniform(low=0.10, high=0.90, size=100)
```

```
array([0.27863286, 0.51853067, 0.54056117, 0.13648156, 0.38858307,
       0.27846475, 0.65098093, 0.23098514, 0.15625989, 0.85280869,
       0.5509451 , 0.16239387, 0.67811241, 0.22676174, 0.30022505,
       0.3347898 , 0.65728857, 0.47141127, 0.27204971, 0.45746101,
       0.196703  , 0.83414961, 0.55315706, 0.47761545, 0.35165314,
       0.13457258, 0.70195593, 0.53461524, 0.71375234, 0.73235901,
       0.1577771 , 0.62777267, 0.14263947, 0.64801956, 0.45468077,
       0.47439692, 0.62884747, 0.65197331, 0.31066359, 0.10918315,
       0.71703556, 0.75291386, 0.21171328, 0.5624054 , 0.37253942,
       0.28577239, 0.258915  , 0.31719262, 0.88148221, 0.27234318,
       0.38512593, 0.11471996, 0.59343222, 0.28432431, 0.69388139,
       0.56740017, 0.43608284, 0.66740227, 0.87763642, 0.74803129,
       0.44452414, 0.86618288, 0.13820648, 0.68622544, 0.64053316,
       0.52160413, 0.55358996, 0.6783562 , 0.64608268, 0.73625014,
       0.19027642, 0.25933011, 0.67705538, 0.41509386, 0.37134832,
       0.22275655, 0.18045921, 0.42579769, 0.56633237, 0.69257642,
       0.79628867, 0.87322093, 0.46445188, 0.4203384 , 0.3391517 ,
       0.76268805, 0.82787294, 0.71290934, 0.15465183, 0.58482122,
       0.85137291, 0.18171413, 0.88203161, 0.38535098, 0.62334973,
       0.19822857, 0.86908175, 0.62094514, 0.84934225, 0.82706184])
```

```
[62] # se reproduzir o código sem o seed na sequência, não haverá a trava de aleatoriedade.
     np.random.uniform(low=0.10, high=0.90, size=100)
```

```
array([0.48500983, 0.75450251, 0.54784106, 0.76580928, 0.58895348,
       0.39720625, 0.73190595, 0.56491178, 0.65877136, 0.14779337,
       0.25141208, 0.41854883, 0.68964221, 0.54177352, 0.29256499,
       0.82760739, 0.12016638, 0.33253592, 0.26049265, 0.39180554,
       0.80770835, 0.64889924, 0.2117912 , 0.25004344, 0.13198589,
       0.25282076, 0.24318745, 0.3496539 , 0.60106379, 0.67387554,
       0.20459079, 0.53295927, 0.7685072 , 0.39100571, 0.54835399,
       0.51806898, 0.55469996, 0.82228867, 0.32118238, 0.50094439,
       0.10000356, 0.13371557, 0.35243667, 0.5738667 , 0.12206297,
       0.41098732, 0.7171872 , 0.10437215, 0.76597854, 0.69531581,
       0.68564233, 0.72599325, 0.83940319, 0.56335324, 0.22198688,
       0.67077979, 0.38743239, 0.81998321, 0.77250465, 0.66381904,
       0.20114065, 0.40399719, 0.41808924, 0.78874839, 0.82689583,
       0.4938446 , 0.54418544, 0.13484602, 0.23471086, 0.61582527,
       0.47289357, 0.15476097, 0.18279763, 0.18108401, 0.3607793 ,
       0.57480171, 0.74450307, 0.6925472 , 0.36622843, 0.15841153,
       0.1559766 , 0.22705023, 0.56458643, 0.33814918, 0.60044662,
       0.16047333, 0.61879717, 0.32293671, 0.21905367, 0.86895804,
       0.77228641, 0.19000994, 0.41917446, 0.15702624, 0.8490132 ,
       0.75373905, 0.56289373, 0.84475328, 0.48509605, 0.18019447])
```

```
[65] np.random.seed(84)
     coef_angulares = np.random.uniform(low=0.10, high=0.90, size=100)
     norma = np.array([])
```

```
for i in range(100):
  norma = np.append(norma, np.linalg.norm(Moscow - (coef_angulares[i]*x+b)))

norma
```

```
array([173.90151233, 103.45241986, 121.75367289, 128.97173036,
       238.69504078, 149.16590195, 137.73123242, 118.54419212,
       101.79720981, 149.04689147, 112.80225228, 105.8049351 ,
       230.82444742, 111.69709582, 134.56525603, 110.93209631,
       204.12484401, 196.82117307, 221.88687726, 156.13691054,
       125.51272174, 136.46474504, 165.39506278, 240.68784732,
       104.52254473, 188.75637903, 131.2292927 , 128.94885341,
       234.70773582, 147.01799832, 113.7418054 , 116.94496221,
       109.58257043, 107.09546548, 145.00728041, 182.83818792,
       140.95017242, 106.12371866, 107.0915778 , 132.45242448,
       144.8736358 , 143.54772697, 240.23793834, 113.74604121,
       116.01191029, 152.02977343, 174.72333708, 141.8068161 ,
       101.93467119, 103.01538109, 169.59787792, 128.13083711,
       105.06648948, 101.83421597, 102.53716362, 113.63850533,
       219.96993004, 117.37921502, 182.70217805, 146.69612581,
       178.64742004, 132.25425967, 109.74998636, 186.49825034,
       104.98272763, 133.73791912, 182.82613374, 235.49081018,
       179.79275527, 228.72013359, 184.2584889 , 114.02435937,
       124.95415977, 187.42082714, 105.18421164, 102.1005486 ,
       135.72717949, 107.19212075, 114.07766563, 225.76351056,
       103.96340568, 118.21130391, 109.96158846, 181.54233309,
       114.09828878, 118.61778913, 224.37792949, 153.50523694,
       185.44994519, 117.35423938, 151.19534147, 118.44720795,
       112.26379208, 104.60483872, 123.8587347 , 111.40038324,
       102.05294074, 150.97459646, 108.76787533, 107.97542886])
```

```
[66] #salvando dados
dados = np.column_stack([norma, coef_angulares])
```

```
[67] dados.shape
```

```
(100, 2)
```

```
dados
```

```
[1.16944962e+02, 3.13387503e-01],
[1.09582570e+02, 3.49353446e-01],
[1.07095465e+02, 3.64798675e-01],
[1.45007280e+02, 2.16664750e-01],
[1.82838188e+02, 7.56607337e-01],
[1.40950172e+02, 2.28893954e-01],
[1.06123719e+02, 4.98674696e-01],
[1.07091578e+02, 3.64825163e-01],
[1.32452424e+02, 2.55873812e-01],
[1.44873636e+02, 2.17062053e-01],
[1.43547727e+02, 6.49380794e-01],
[2.40237938e+02, 8.95695282e-01],
[1.13746041e+02, 3.27805663e-01],
[1.16011910e+02, 5.52953548e-01],
[1.52029773e+02, 1.96246038e-01],
[1.74723337e+02, 1.34690357e-01],
[1.41806816e+02, 2.26281568e-01],
[1.01934671e+02, 4.46411433e-01],
[1.03015381e+02, 4.01772345e-01],
[1.69597878e+02, 1.48141097e-01],
[1.28130837e+02, 5.99869975e-01],
[1.05066489e+02, 4.90237320e-01],
[1.01834216e+02, 4.29372898e-01],
[1.02537164e+02, 4.61227883e-01],
[1.13638505e+02, 5.42087557e-01],
[2.19969930e+02, 8.47855493e-01],
[1.17379215e+02, 3.11531216e-01],
[1.82702178e+02, 1.14143458e-01],
[1.46696126e+02, 6.58730132e-01],
[1.78647420e+02, 7.45874224e-01],
[1.32254260e+02, 6.13874257e-01],
[1.09749986e+02, 5.22003253e-01],
[1.86498250e+02, 7.65890838e-01],
[1.04982728e+02, 3.80887466e-01],
[1.33737919e+02, 2.51650906e-01],
[1.82826134e+02, 1.13827615e-01],
[2.35490810e+02, 8.84579966e-01],
[1.79792755e+02, 7.48819153e-01],
[2.28720134e+02, 8.68636214e-01],
[1.84258489e+02, 1.10184840e-01],
[1.14024359e+02, 3.26492005e-01],
[1.24954160e+02, 2.81855776e-01],
[1.87420827e+02, 1.02185763e-01],
[1.05184212e+02, 3.79168955e-01],
[1.02100549e+02, 4.51851906e-01],
[1.35727179e+02, 2.45224309e-01],
[1.07192121e+02, 5.06261248e-01],
[1.14077666e+02, 3.26241845e-01],
[2.25763511e+02, 8.61637911e-01],
[1.03963406e+02, 4.79881898e-01],
[1.18211304e+02, 3.08031447e-01],
[1.09961588e+02, 3.47209860e-01],
[1.81542333e+02, 1.17103490e-01],
[1.14098289e+02, 3.26145185e-01],
[1.18617789e+02, 3.06347578e-01],
[2.24377929e+02, 8.58350258e-01],
[1.53505237e+02, 6.78345791e-01],
[1.85449945e+02, 7.63240042e-01],
[1.17354239e+02, 5.58766835e-01],
```

```
[69] np.savetxt('dados.csv', dados, delimiter=',')
```

0s

```
[ ]
```

Produtos pagos do Colab - Cancelar contratos

0s conclusão: 09:32