Data Visualization Assignment

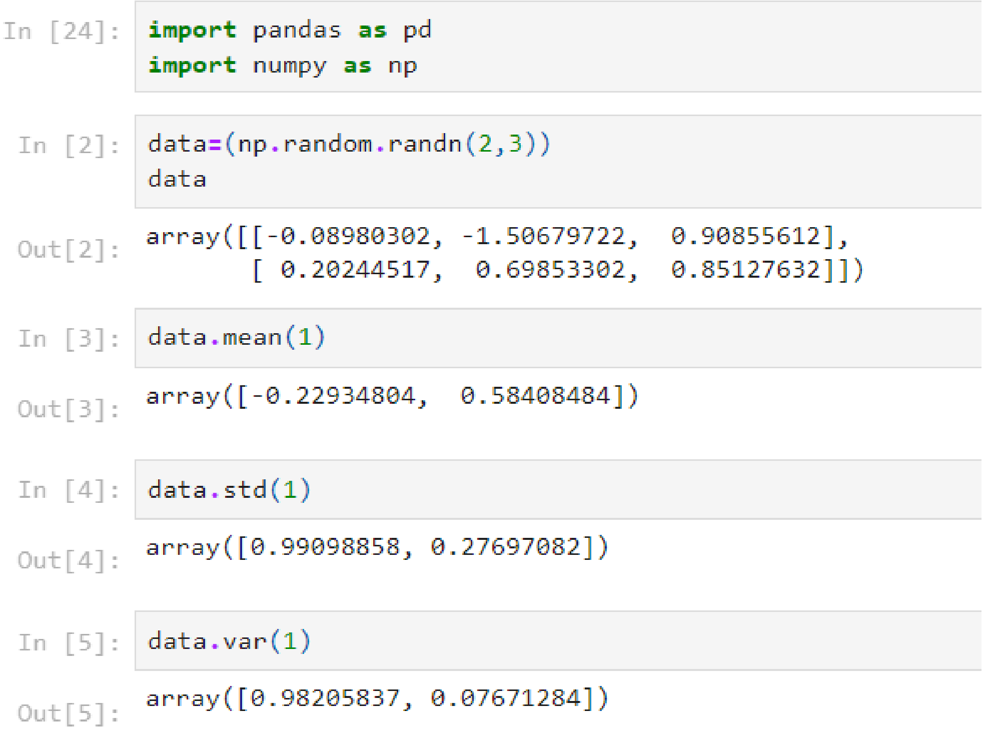
Name: Samruddhi

Roll No: 23ENB015

Semester: 2

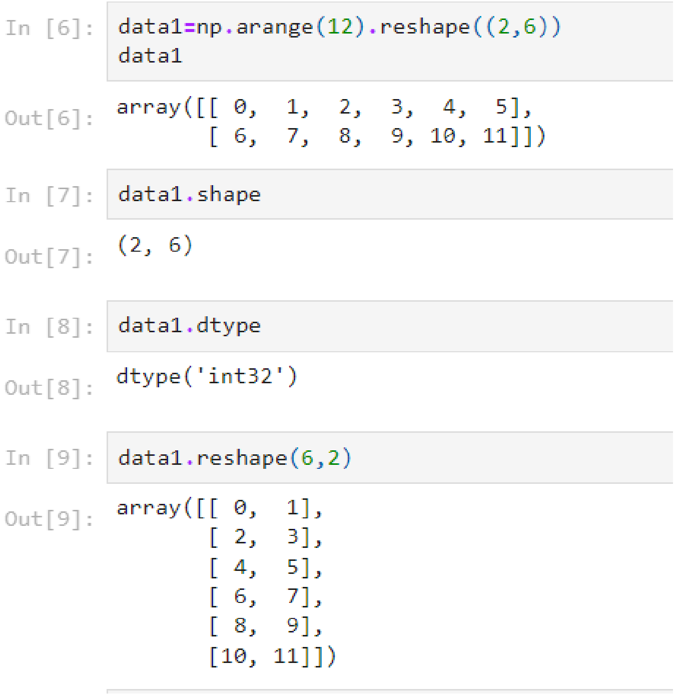
Course: English Hons

1) Write programs in Python using NumPy library to do the following:
i) Compute the mean, standard deviation, and variance of a two dimensional random integer array along the second axis



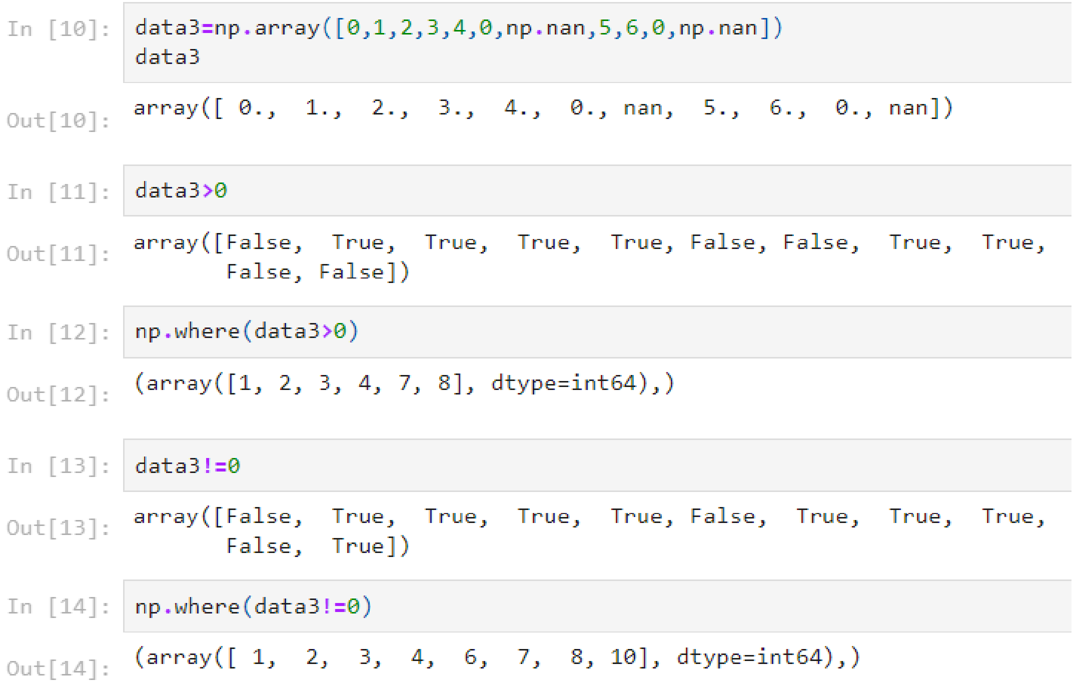
```
In [24]: import pandas as pd
         import numpy as np

In [2]:  data=(np.random.randn(2,3))
         data

Out[2]:  array([[-0.08980302, -1.50679722,  0.90855612],
                [ 0.20244517,  0.69853302,  0.85127632]])

In [3]:  data.mean(1)

Out[3]:  array([-0.22934804,  0.58408484])

In [4]:  data.std(1)

Out[4]:  array([0.99098858, 0.27697082])

In [5]:  data.var(1)

Out[5]:  array([0.98205837, 0.07671284])
```

ii) Create a 2-dimensional array of size m x n integer elements, also print the shape, type and data type of the array and then reshape it into an n x m array, where n and m are user inputs given at the run time



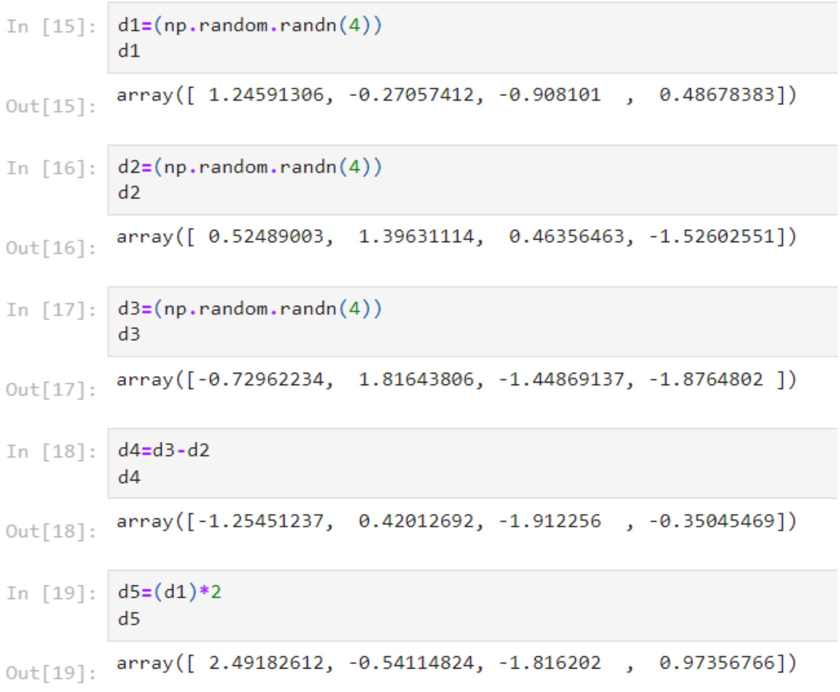
```
In [6]:  data1=np.arange(12).reshape((2,6))
         data1

Out[6]:  array([[ 0,  1,  2,  3,  4,  5],
                [ 6,  7,  8,  9, 10, 11]])

In [7]:  data1.shape

Out[7]:  (2, 6)

In [8]:  data1.dtype

Out[8]:  dtype('int32')

In [9]:  data1.reshape(6,2)

Out[9]:  array([[ 0,  1],
                [ 2,  3],
                [ 4,  5],
                [ 6,  7],
                [ 8,  9],
                [10, 11]])
```

iii) Test whether the elements of a given 1D array are zero, non-zero and NaN. Record the indices of these elements in three separate arrays.



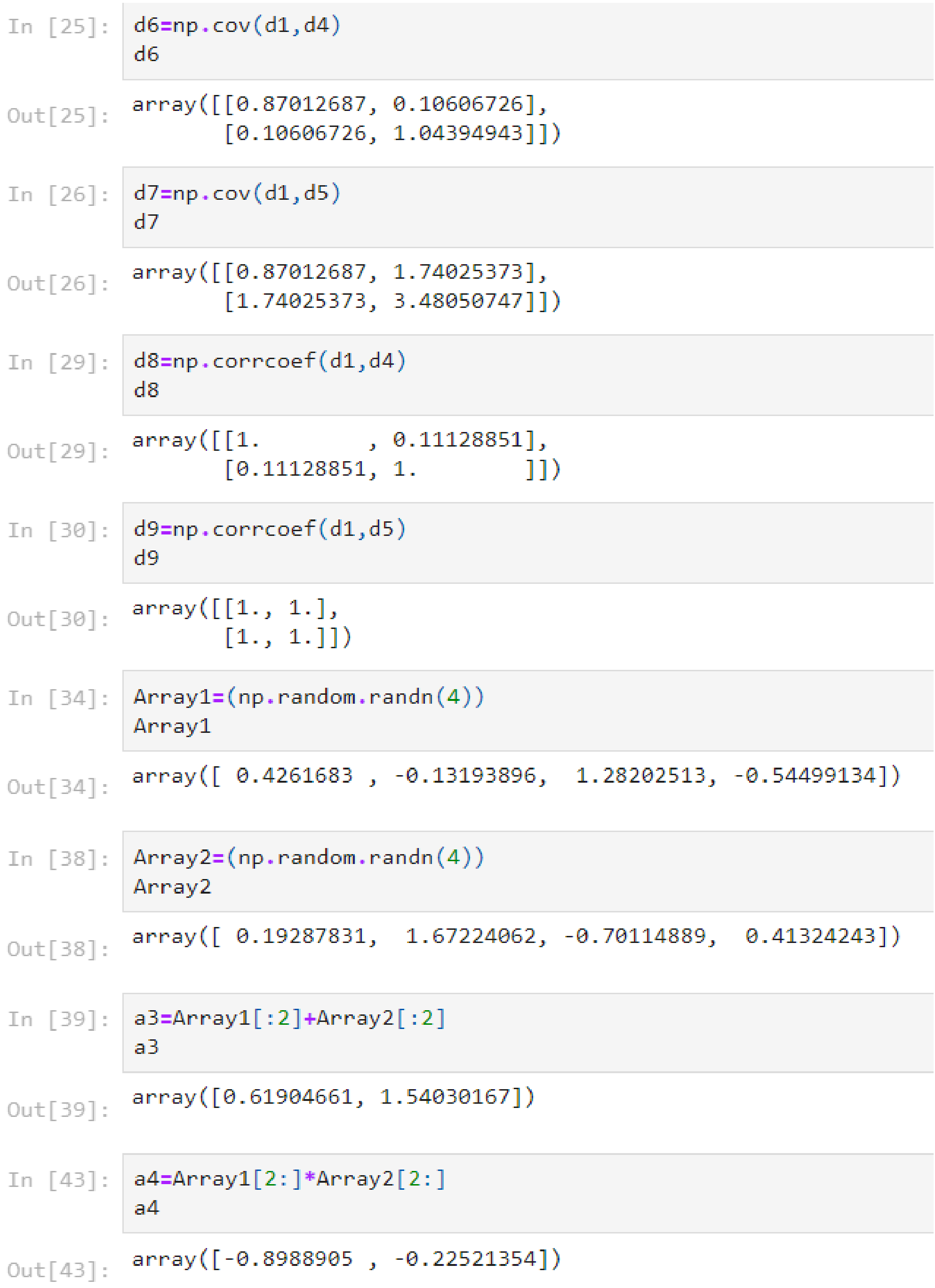
```
In [10]: data3=np.array([0,1,2,3,4,0,np.nan,5,6,0,np.nan])
         data3

Out[10]: array([ 0.,  1.,  2.,  3.,  4.,  0., nan,  5.,  6.,  0., nan])

In [11]: data3>0

Out[11]: array([False,  True,  True,  True,  True, False, False,  True,  True,
                False, False])

In [12]: np.where(data3>0)

Out[12]: (array([1, 2, 3, 4, 7, 8], dtype=int64),)

In [13]: data3!=0

Out[13]: array([False,  True,  True,  True,  True, False,  True,  True,  True,
                False,  True])

In [14]: np.where(data3!=0)

Out[14]: (array([ 1,  2,  3,  4,  6,  7,  8, 10], dtype=int64),)
```

iv) Create three random arrays of the same size: Array1, Array2 and Array3. Subtract Array 2 from Array3 and store in Array4. Create another array Array5 having two times the values in Array1. Find Covariance and Correlation of Array1 with Array4 and Array5 respectively.



```
In [15]: d1=(np.random.randn(4))
         d1

Out[15]: array([ 1.24591306, -0.27057412, -0.908101  ,  0.48678383])

In [16]: d2=(np.random.randn(4))
         d2

Out[16]: array([ 0.52489003,  1.39631114,  0.46356463, -1.52602551])

In [17]: d3=(np.random.randn(4))
         d3

Out[17]: array([-0.72962234,  1.81643806, -1.44869137, -1.8764802 ])

In [18]: d4=d3-d2
         d4

Out[18]: array([-1.25451237,  0.42012692, -1.912256  , -0.35045469])

In [19]: d5=(d1)*2
         d5

Out[19]: array([ 2.49182612, -0.54114824, -1.816202  ,  0.97356766])
```

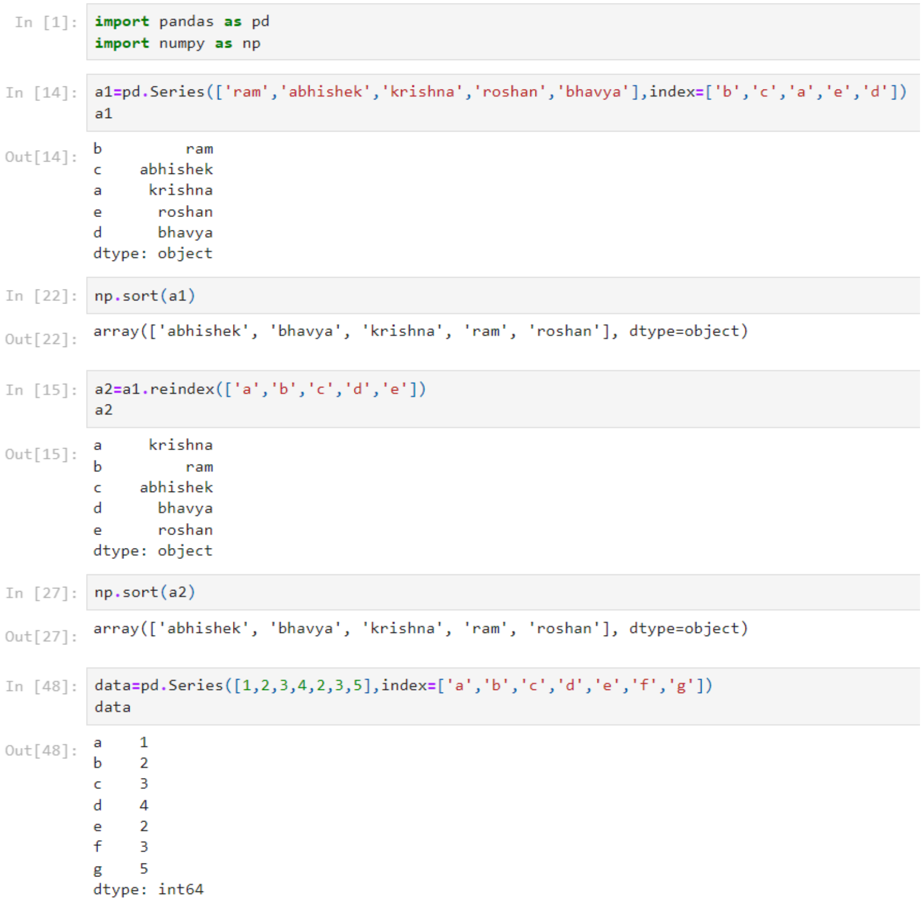
v) Create two random arrays of the same size 10: Array1, and Array2. Find the sum of the first half of both the arrays and product of the second half of both the arrays.



```
In [25]:  d6=np.cov(d1,d4)
          d6

Out[25]:  array([[0.87012687, 0.10606726],
                 [0.10606726, 1.04394943]])

In [26]:  d7=np.cov(d1,d5)
          d7

Out[26]:  array([[0.87012687, 1.74025373],
                 [1.74025373, 3.48050747]])

In [29]:  d8=np.corrcoef(d1,d4)
          d8

Out[29]:  array([[1.        , 0.11128851],
                 [0.11128851, 1.        ]])

In [30]:  d9=np.corrcoef(d1,d5)
          d9

Out[30]:  array([[1., 1.],
                 [1., 1.]])

In [34]:  Array1=(np.random.randn(4))
          Array1

Out[34]:  array([ 0.4261683 , -0.13193896,  1.28202513, -0.54499134])

In [38]:  Array2=(np.random.randn(4))
          Array2

Out[38]:  array([ 0.19287831,  1.67224062, -0.70114889,  0.41324243])

In [39]:  a3=Array1[:2]+Array2[:2]
          a3

Out[39]:  array([0.61904661, 1.54030167])

In [43]:  a4=Array1[2:]*Array2[2:]
          a4

Out[43]:  array([-0.8988905 , -0.22521354])
```

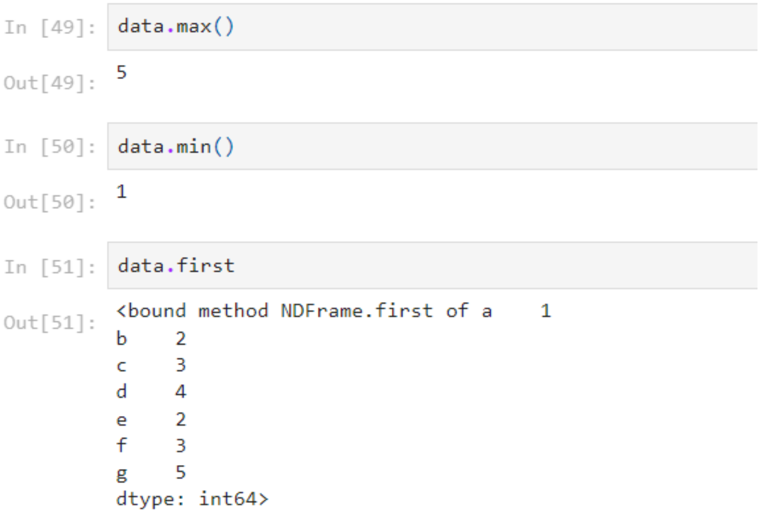
2) Do the following using PANDAS Series:

a) Create a series with 5 elements. Display the series sorted on index and also sorted on values seperately



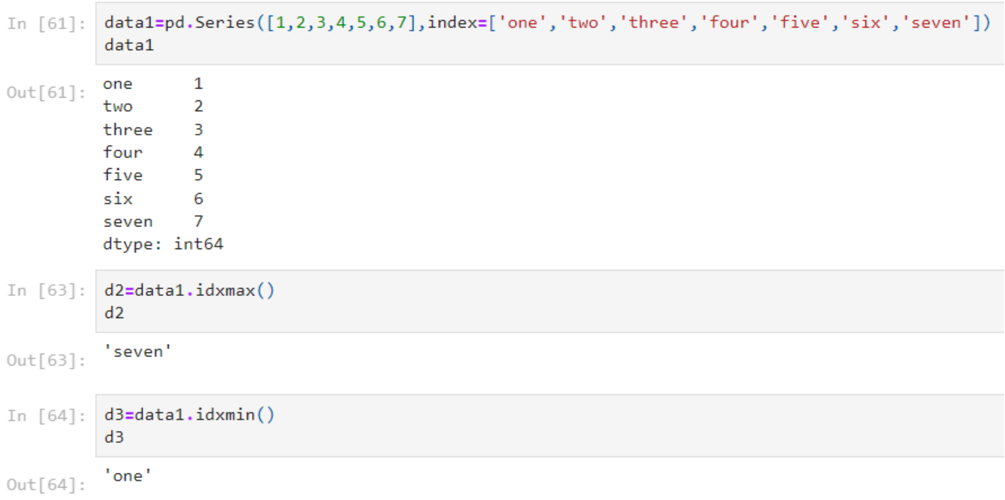
```
In [1]:  import pandas as pd
         import numpy as np

In [14]: a1=pd.Series(['ram','abhishek','krishna','roshan','bhavya'],index=['b','c','a','e','d'])
         a1

Out[14]: b          ram
         c     abhishek
         a      krishna
         e       roshan
         d       bhavya
         dtype: object

In [22]: np.sort(a1)

Out[22]: array(['abhishek', 'bhavya', 'krishna', 'ram', 'roshan'], dtype=object)

In [15]: a2=a1.reindex(['a','b','c','d','e'])
         a2

Out[15]: a      krishna
         b          ram
         c     abhishek
         d       bhavya
         e       roshan
         dtype: object

In [27]: np.sort(a2)

Out[27]: array(['abhishek', 'bhavya', 'krishna', 'ram', 'roshan'], dtype=object)

In [48]: data=pd.Series([1,2,3,4,2,3,5],index=['a','b','c','d','e','f','g'])
         data

Out[48]: a    1
         b    2
         c    3
         d    4
         e    2
         f    3
         g    5
         dtype: int64
```

b) Create a series with N elements with some duplicate values. Find the minimum and maximum ranks assigned to the values using ‘first’ and ‘max’ methods



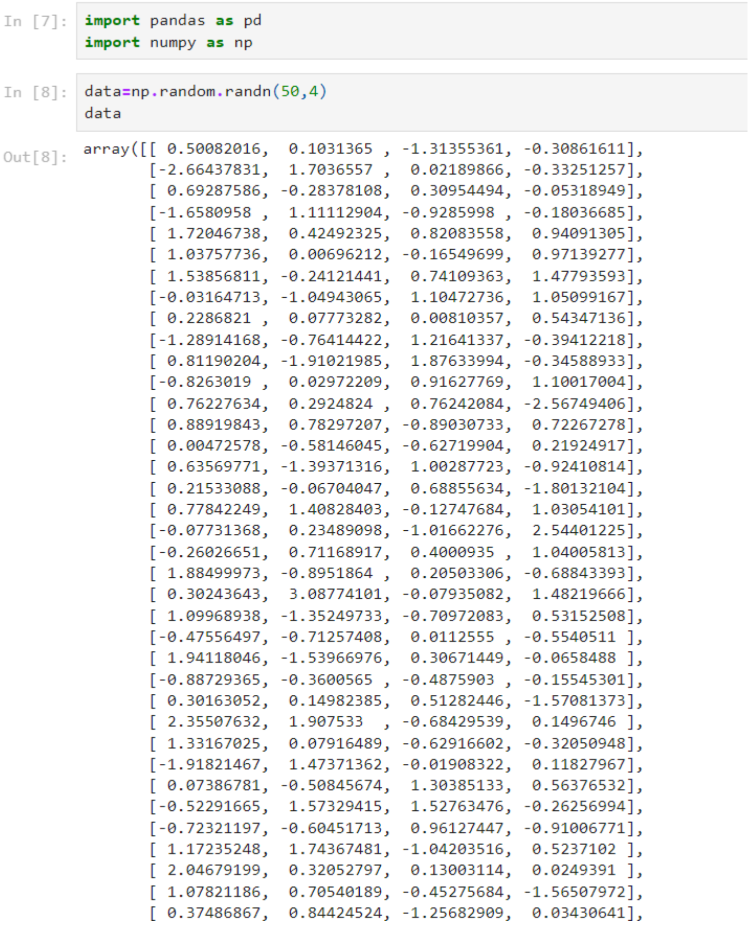
```
In [49]: data.max()

Out[49]: 5

In [50]: data.min()

Out[50]: 1

In [51]: data.first

Out[51]: <bound method NDFrame.first of a    1
         b    2
         c    3
         d    4
         e    2
         f    3
         g    5
         dtype: int64>
```

c) Display the index value of the minimum and maximum element of a Series

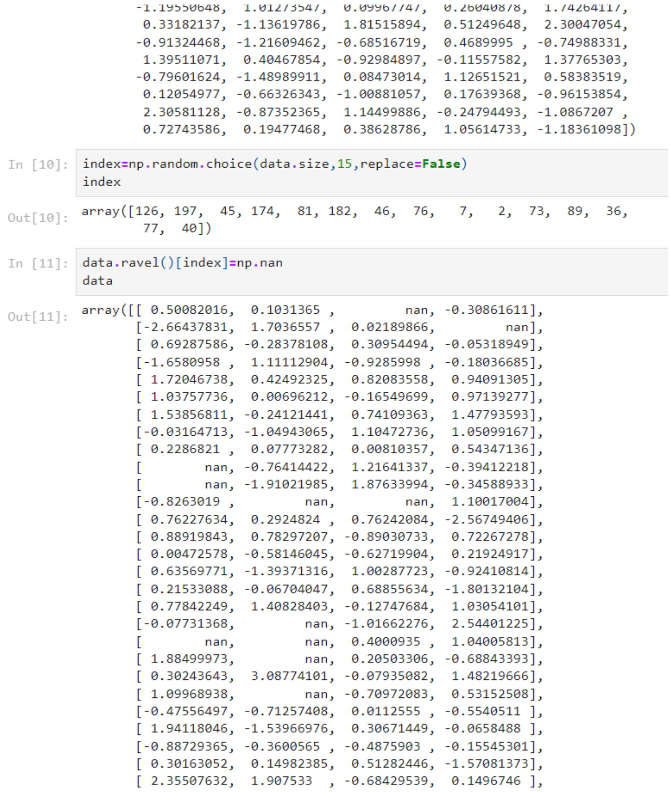


```
In [61]: data1=pd.Series([1,2,3,4,5,6,7],index=['one','two','three','four','five','six','seven'])
         data1

Out[61]: one      1
         two      2
         three    3
         four     4
         five     5
         six      6
         seven    7
         dtype: int64

In [63]: d2=data1.idxmax()
         d2

Out[63]: 'seven'

In [64]: d3=data1.idxmin()
         d3

Out[64]: 'one'
```

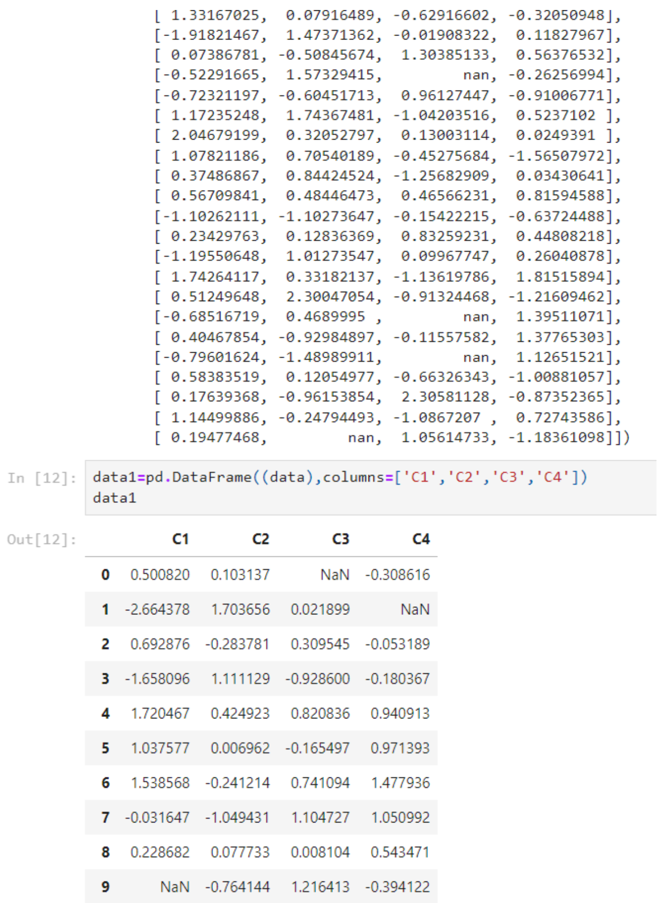
3) Create a data frame having at least 3 columns and 50 rows to store numeric data generated using a random function. Replace 10% of the values by null values whose index positions are generated using random function. Do the following:



```
In [7]: import pandas as pd
        import numpy as np

In [8]: data=np.random.randn(50,4)
        data

Out[8]: array([[ 0.50082016,  0.1031365 , -1.31355361, -0.30861611],
               [-2.66437831,  1.7036557 ,  0.02189866, -0.33251257],
               [ 0.69287586, -0.28378108,  0.30954494, -0.05318949],
               [-1.6580958 ,  1.11112904, -0.9285998 , -0.18036685],
               [ 1.72046738,  0.42492325,  0.82083558,  0.94091305],
               [ 1.03757736,  0.00696212, -0.16549699,  0.97139277],
               [ 1.53856811, -0.24121441,  0.74109363,  1.47793593],
               [-0.03164713, -1.04943065,  1.10472736,  1.05099167],
               [ 0.2286821 ,  0.07773282,  0.00810357,  0.54347136],
               [-1.28914168, -0.76414422,  1.21641337, -0.39412218],
               [ 0.81190204, -1.91021985,  1.87633994, -0.34588933],
               [-0.8263019 ,  0.02972209,  0.91627769,  1.10017004],
               [ 0.76227634,  0.2924824 ,  0.76242084, -2.56749406],
               [ 0.88919843,  0.78297207, -0.89030733,  0.72267278],
               [ 0.00472578, -0.58146045, -0.62719904,  0.21924917],
               [ 0.63569771, -1.39371316,  1.00287723, -0.92410814],
               [ 0.21533088, -0.06704047,  0.68855634, -1.80132104],
               [ 0.77842249,  1.40828403, -0.12747684,  1.03054101],
               [-0.07731368,  0.23489098, -1.01662276,  2.54401225],
               [-0.26026651,  0.71168917,  0.4000935 ,  1.04005813],
               [ 1.88499973, -0.8951864 ,  0.20503306, -0.68843393],
               [ 0.30243643,  3.08774101, -0.07935082,  1.48219666],
               [ 1.09968938, -1.35249733, -0.70972083,  0.53152508],
               [-0.47556497, -0.71257408,  0.0112555 , -0.5540511 ],
               [ 1.94118046, -1.53966976,  0.30671449, -0.0658488 ],
               [-0.88729365, -0.3600565 , -0.4875903 , -0.15545301],
               [ 0.30163052,  0.14982385,  0.51282446, -1.57081373],
               [ 2.35507632,  1.907533  , -0.68429539,  0.1496746 ],
               [ 1.33167025,  0.07916489, -0.62916602, -0.32050948],
               [-1.91821467,  1.47371362, -0.01908322,  0.11827967],
               [ 0.07386781, -0.50845674,  1.30385133,  0.56376532],
               [-0.52291665,  1.57329415,  1.52763476, -0.26256994],
               [-0.72321197, -0.60451713,  0.96127447, -0.91006771],
               [ 1.17235248,  1.74367481, -1.04203516,  0.5237102 ],
               [ 2.04679199,  0.32052797,  0.13003114,  0.0249391 ],
               [ 1.07821186,  0.70540189, -0.45275684, -1.56507972],
               [ 0.37486867,  0.84424524, -1.25682909,  0.03430641],
```



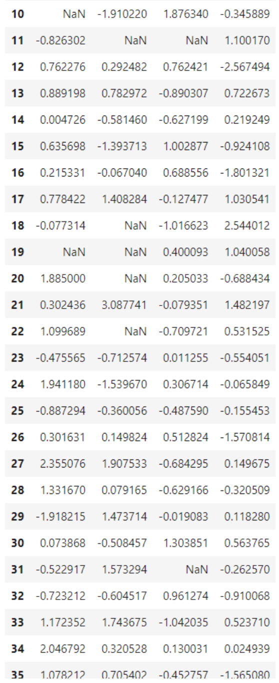
```
        -1.19550648,  1.01273547,  0.09967747,  0.26040878,  1.74264117,
         0.33182137, -1.13619786,  1.81515894,  0.51249648,  2.30047054,
        -0.91324468, -1.21609462, -0.68516719,  0.4689995 , -0.74988331,
         1.39511071,  0.40467854, -0.92984897, -0.11557582,  1.37765303,
        -0.79601624, -1.48989911,  0.08473014,  1.12651521,  0.58383519,
         0.12054977, -0.66326343, -1.00881057,  0.17639368, -0.96153854,
         2.30581128, -0.87352365,  1.14499886, -0.24794493, -1.0867207 ,
         0.72743586,  0.19477468,  0.38628786,  1.05614733, -1.18361098])

In [10]: index=np.random.choice(data.size,15,replace=False)
         index

Out[10]: array([126, 197,  45, 174,  81, 182,  46,  76,   7,   2,  73,  89,  36,
                 77,  40])

In [11]: data.ravel()[index]=np.nan
         data

Out[11]: array([[ 0.50082016,  0.1031365 ,         nan, -0.30861611],
                [-2.66437831,  1.7036557 ,  0.02189866,         nan],
                [ 0.69287586, -0.28378108,  0.30954494, -0.05318949],
                [-1.6580958 ,  1.11112904, -0.9285998 , -0.18036685],
                [ 1.72046738,  0.42492325,  0.82083558,  0.94091305],
                [ 1.03757736,  0.00696212, -0.16549699,  0.97139277],
                [ 1.53856811, -0.24121441,  0.74109363,  1.47793593],
                [-0.03164713, -1.04943065,  1.10472736,  1.05099167],
                [ 0.2286821 ,  0.07773282,  0.00810357,  0.54347136],
                [        nan, -0.76414422,  1.21641337, -0.39412218],
                [        nan, -1.91021985,  1.87633994, -0.34588933],
                [-0.8263019 ,         nan,         nan,  1.10017004],
                [ 0.76227634,  0.2924824 ,  0.76242084, -2.56749406],
                [ 0.88919843,  0.78297207, -0.89030733,  0.72267278],
                [ 0.00472578, -0.58146045, -0.62719904,  0.21924917],
                [ 0.63569771, -1.39371316,  1.00287723, -0.92410814],
                [ 0.21533088, -0.06704047,  0.68855634, -1.80132104],
                [ 0.77842249,  1.40828403, -0.12747684,  1.03054101],
                [-0.07731368,         nan, -1.01662276,  2.54401225],
                [        nan,         nan,  0.4000935 ,  1.04005813],
                [ 1.88499973,         nan,  0.20503306, -0.68843393],
                [ 0.30243643,  3.08774101, -0.07935082,  1.48219666],
                [ 1.09968938,         nan, -0.70972083,  0.53152508],
                [-0.47556497, -0.71257408,  0.0112555 , -0.5540511 ],
                [ 1.94118046, -1.53966976,  0.30671449, -0.0658488 ],
                [-0.88729365, -0.3600565 , -0.4875903 , -0.15545301],
                [ 0.30163052,  0.14982385,  0.51282446, -1.57081373],
                [ 2.35507632,  1.907533  , -0.68429539,  0.1496746 ],
```

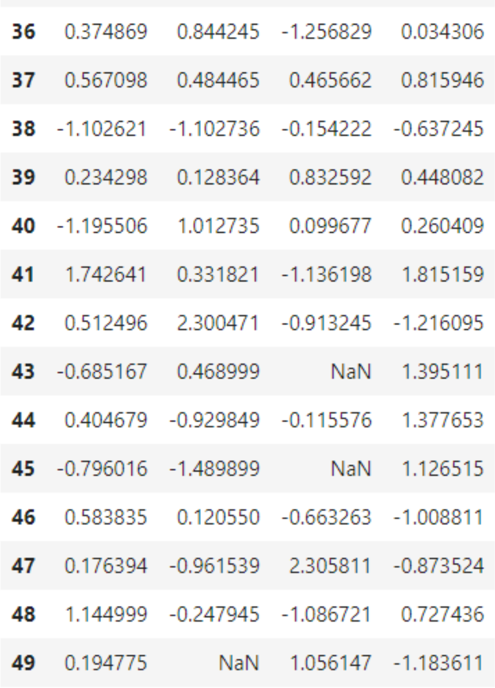


```
       [ 1.33167025,  0.07916489, -0.62916602, -0.32050948],
       [-1.91821467,  1.47371362, -0.01908322,  0.11827967],
       [ 0.07386781, -0.50845674,  1.30385133,  0.56376532],
       [-0.52291665,  1.57329415,         nan, -0.26256994],
       [-0.72321197, -0.60451713,  0.96127447, -0.91006771],
       [ 1.17235248,  1.74367481, -1.04203516,  0.5237102 ],
       [ 2.04679199,  0.32052797,  0.13003114,  0.0249391 ],
       [ 1.07821186,  0.70540189, -0.45275684, -1.56507972],
       [ 0.37486867,  0.84424524, -1.25682909,  0.03430641],
       [ 0.56709841,  0.48446473,  0.46566231,  0.81594588],
       [-1.10262111, -1.10273647, -0.15422215, -0.63724488],
       [ 0.23429763,  0.12836369,  0.83259231,  0.44808218],
       [-1.19550648,  1.01273547,  0.09967747,  0.26040878],
       [ 1.74264117,  0.33182137, -1.13619786,  1.81515894],
       [ 0.51249648,  2.30047054, -0.91324468, -1.21609462],
       [-0.68516719,  0.4689995 ,         nan,  1.39511071],
       [ 0.40467854, -0.92984897, -0.11557582,  1.37765303],
       [-0.79601624, -1.48989911,         nan,  1.12651521],
       [ 0.58383519,  0.12054977, -0.66326343, -1.00881057],
       [ 0.17639368, -0.96153854,  2.30581128, -0.87352365],
       [ 1.14499886, -0.24794493, -1.0867207 ,  0.72743586],
       [ 0.19477468,         nan,  1.05614733, -1.18361098]])
```

```
In [12]: data1=pd.DataFrame((data),columns=['C1','C2','C3','C4'])
         data1
```

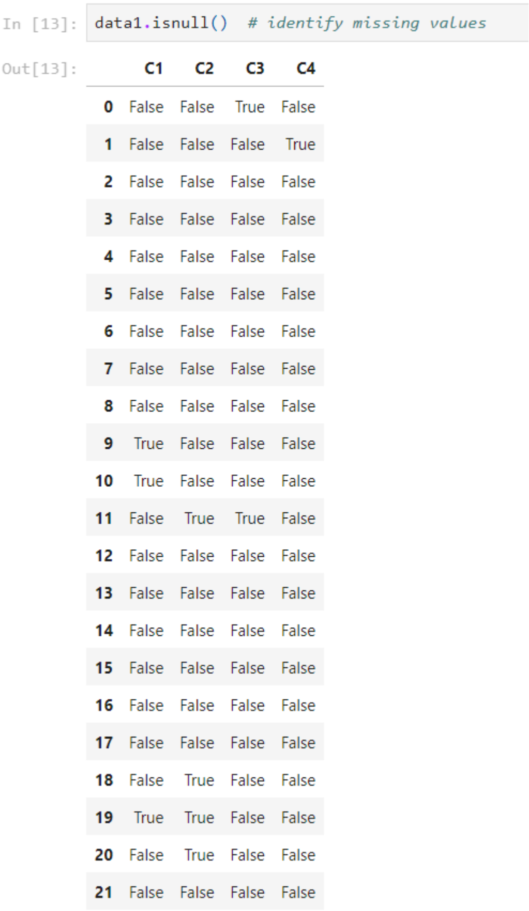
Out[12]:

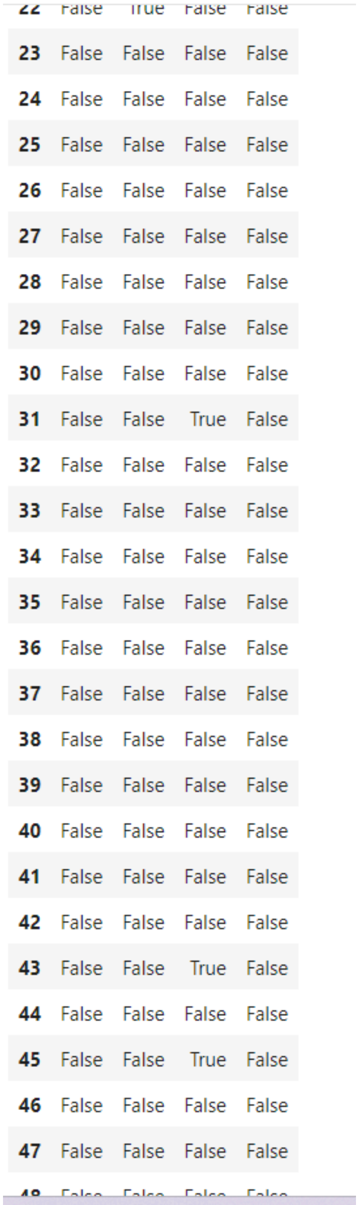




	C1	C2	C3	C4
0	0.500820	0.103137	NaN	-0.308616
1	-2.664378	1.703656	0.021899	NaN
2	0.692876	-0.283781	0.309545	-0.053189
3	-1.658096	1.111129	-0.928600	-0.180367
4	1.720467	0.424923	0.820836	0.940913
5	1.037577	0.006962	-0.165497	0.971393
6	1.538568	-0.241214	0.741094	1.477936
7	-0.031647	-1.049431	1.104727	1.050992
8	0.228682	0.077733	0.008104	0.543471
9	NaN	-0.764144	1.216413	-0.394122
10	NaN	-1.910220	1.876340	-0.345889
11	-0.826302	NaN	NaN	1.100170
12	0.762276	0.292482	0.762421	-2.567494
13	0.889198	0.782972	-0.890307	0.722673
14	0.004726	-0.581460	-0.627199	0.219249
15	0.635698	-1.393713	1.002877	-0.924108
16	0.215331	-0.067040	0.688556	-1.801321
17	0.778422	1.408284	-0.127477	1.030541
18	-0.077314	NaN	-1.016623	2.544012
19	NaN	NaN	0.400093	1.040058
20	1.885000	NaN	0.205033	-0.688434
21	0.302436	3.087741	-0.079351	1.482197
22	1.099689	NaN	-0.709721	0.531525
23	-0.475565	-0.712574	0.011255	-0.554051
24	1.941180	-1.539670	0.306714	-0.065849
25	-0.887294	-0.360056	-0.487590	-0.155453
26	0.301631	0.149824	0.512824	-1.570814
27	2.355076	1.907533	-0.684295	0.149675
28	1.331670	0.079165	-0.629166	-0.320509
29	-1.918215	1.473714	-0.019083	0.118280
30	0.073868	-0.508457	1.303851	0.563765
31	-0.522917	1.573294	NaN	-0.262570
32	-0.723212	-0.604517	0.961274	-0.910068
33	1.172352	1.743675	-1.042035	0.523710
34	2.046792	0.320528	0.130031	0.024939
35	1.078212	0.705402	-0.452757	-1.565080
36	0.374869	0.844245	-1.256829	0.034306
37	0.567098	0.484465	0.465662	0.815946
38	-1.102621	-1.102736	-0.154222	-0.637245
39	0.234298	0.128364	0.832592	0.448082
40	-1.195506	1.012735	0.099677	0.260409
41	1.742641	0.331821	-1.136198	1.815159
42	0.512496	2.300471	-0.913245	-1.216095
43	-0.685167	0.468999	NaN	1.395111
44	0.404679	-0.929849	-0.115576	1.377653
45	-0.796016	-1.489899	NaN	1.126515
46	0.583835	0.120550	-0.663263	-1.008811
47	0.176394	-0.961539	2.305811	-0.873524
48	1.144999	-0.247945	-1.086721	0.727436
49	0.194775	NaN	1.056147	-1.183611

a) Identify and count missing values in a data frame.

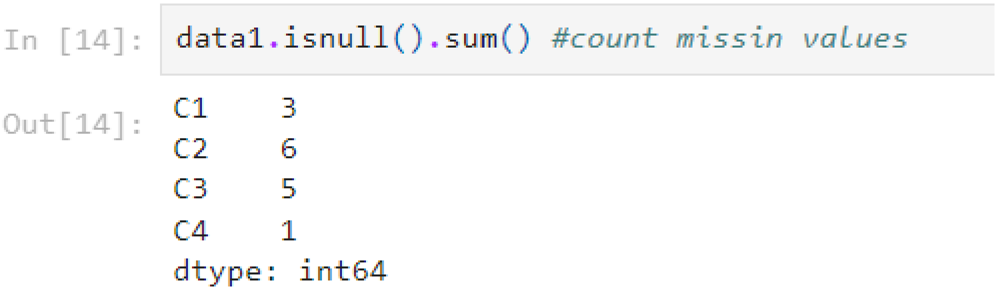




```
In [13]: data1.isnull()  # identify missing values
```

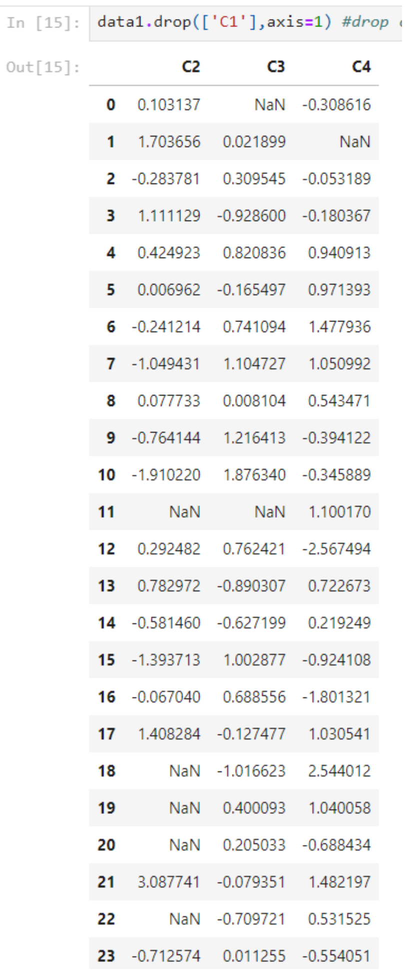
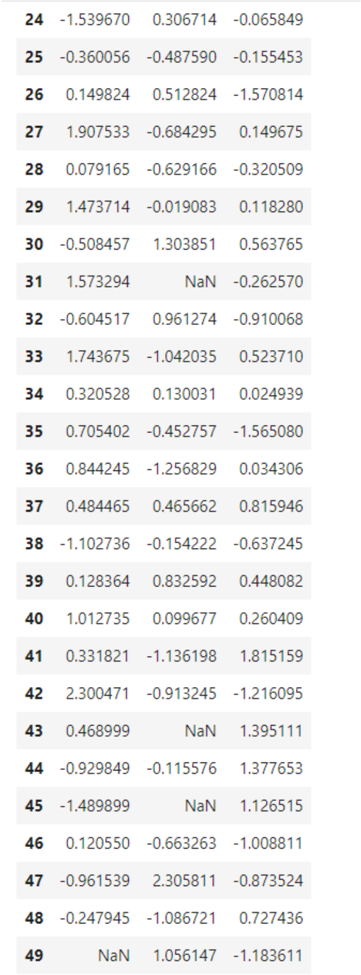
Out[13]:

	C1	C2	C3	C4
0	False	False	True	False
1	False	False	False	True
2	False	False	False	False
3	False	False	False	False
4	False	False	False	False
5	False	False	False	False
6	False	False	False	False
7	False	False	False	False
8	False	False	False	False
9	True	False	False	False
10	True	False	False	False
11	False	True	True	False
12	False	False	False	False
13	False	False	False	False
14	False	False	False	False
15	False	False	False	False
16	False	False	False	False
17	False	False	False	False
18	False	True	False	False
19	True	True	False	False
20	False	True	False	False
21	False	False	False	False
22	False	True	False	False
23	False	False	False	False
24	False	False	False	False
25	False	False	False	False
26	False	False	False	False
27	False	False	False	False
28	False	False	False	False
29	False	False	False	False
30	False	False	False	False
31	False	False	True	False
32	False	False	False	False
33	False	False	False	False
34	False	False	False	False
35	False	False	False	False
36	False	False	False	False
37	False	False	False	False
38	False	False	False	False
39	False	False	False	False
40	False	False	False	False
41	False	False	False	False
42	False	False	False	False
43	False	False	True	False
44	False	False	False	False
45	False	False	True	False
46	False	False	False	False
47	False	False	False	False
48	False	False	False	False



```
In [14]: data1.isnull().sum() #count missin values

Out[14]: C1    3
         C2    6
         C3    5
         C4    1
         dtype: int64
```

b) Drop the column having more than 5 null values.

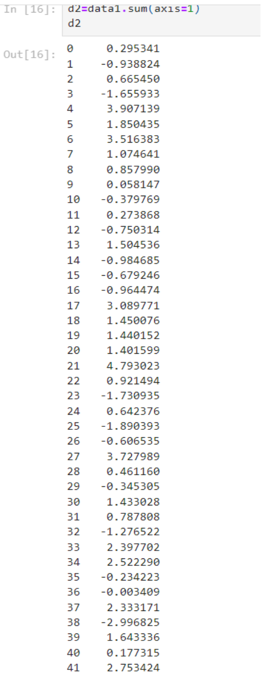
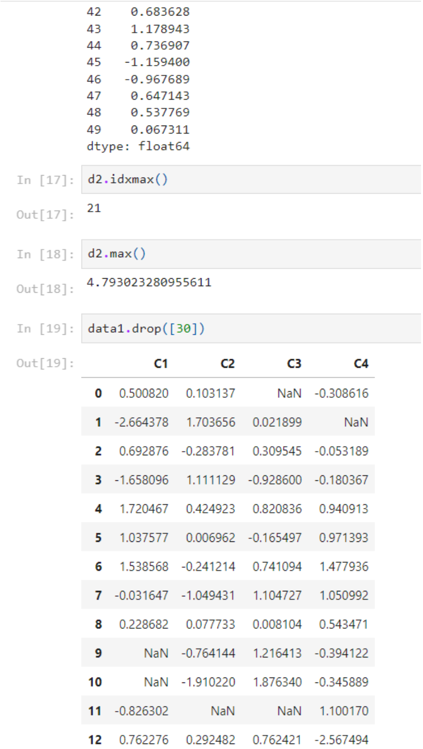



```
In [15]: data1.drop(['C1'],axis=1) #drop
```

Out[15]:

	C2	C3	C4
0	0.103137	NaN	-0.308616
1	1.703656	0.021899	NaN
2	-0.283781	0.309545	-0.053189
3	1.111129	-0.928600	-0.180367
4	0.424923	0.820836	0.940913
5	0.006962	-0.165497	0.971393
6	-0.241214	0.741094	1.477936
7	-1.049431	1.104727	1.050992
8	0.077733	0.008104	0.543471
9	-0.764144	1.216413	-0.394122
10	-1.910220	1.876340	-0.345889
11	NaN	NaN	1.100170
12	0.292482	0.762421	-2.567494
13	0.782972	-0.890307	0.722673
14	-0.581460	-0.627199	0.219249
15	-1.393713	1.002877	-0.924108
16	-0.067040	0.688556	-1.801321
17	1.408284	-0.127477	1.030541
18	NaN	-1.016623	2.544012
19	NaN	0.400093	1.040058
20	NaN	0.205033	-0.688434
21	3.087741	-0.079351	1.482197
22	NaN	-0.709721	0.531525
23	-0.712574	0.011255	-0.554051
24	-1.539670	0.306714	-0.065849
25	-0.360056	-0.487590	-0.155453
26	0.149824	0.512824	-1.570814
27	1.907533	-0.684295	0.149675
28	0.079165	-0.629166	-0.320509
29	1.473714	-0.019083	0.118280
30	-0.508457	1.303851	0.563765
31	1.573294	NaN	-0.262570
32	-0.604517	0.961274	-0.910068
33	1.743675	-1.042035	0.523710
34	0.320528	0.130031	0.024939
35	0.705402	-0.452757	-1.565080
36	0.844245	-1.256829	0.034306
37	0.484465	0.465662	0.815946
38	-1.102736	-0.154222	-0.637245
39	0.128364	0.832592	0.448082
40	1.012735	0.099677	0.260409
41	0.331821	-1.136198	1.815159
42	2.300471	-0.913245	-1.216095
43	0.468999	NaN	1.395111
44	-0.929849	-0.115576	1.377653
45	-1.489899	NaN	1.126515
46	0.120550	-0.663263	-1.008811
47	-0.961539	2.305811	-0.873524
48	-0.247945	-1.086721	0.727436
49	NaN	1.056147	-1.183611

c) Identify the row label having maximum of the sum of all values in a row and drop that row.

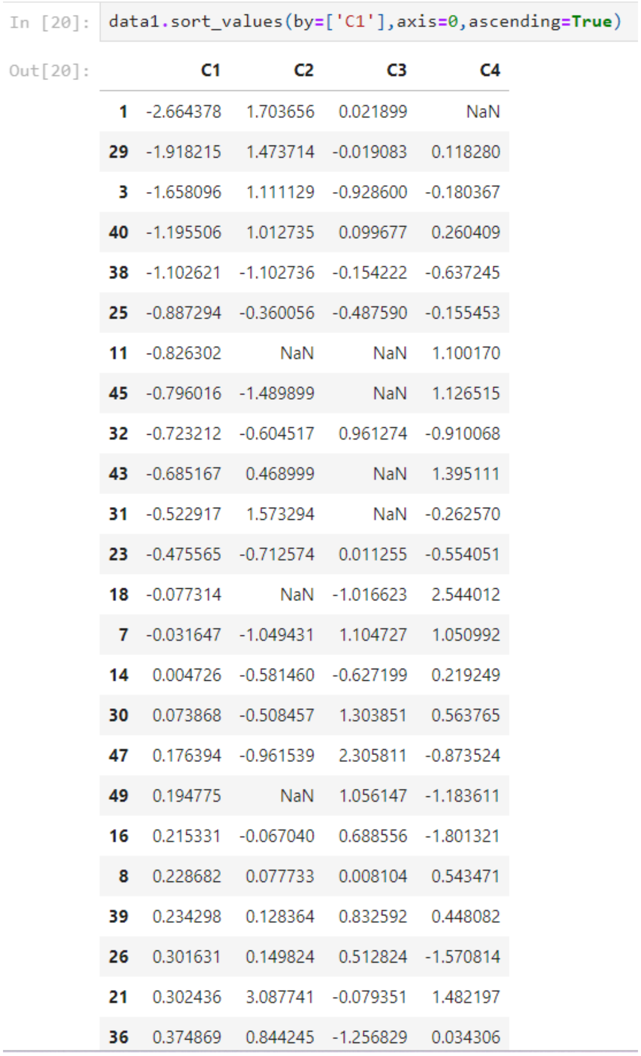



```
In [16]: d2=data1.sum(axis=1)
         d2

Out[16]: 0     0.295341
         1    -0.938824
         2     0.665450
         3    -1.655933
         4     3.907139
         5     1.850435
         6     3.516383
         7     1.074641
         8     0.857990
         9     0.058147
         10   -0.379769
         11    0.273868
         12   -0.750314
         13    1.504536
         14   -0.984685
         15   -0.679246
         16   -0.964474
         17    3.089771
         18    1.450076
         19    1.440152
         20    1.401599
         21    4.793023
         22    0.921494
         23   -1.730935
         24    0.642376
         25   -1.890393
         26   -0.606535
         27    3.727989
         28    0.461160
         29   -0.345305
         30    1.433028
         31    0.787808
         32   -1.276522
         33    2.397702
         34    2.522290
         35   -0.234223
         36   -0.003409
         37    2.333171
         38   -2.996825
         39    1.643336
         40    0.177315
         41    2.753424
         42    0.683628
         43    1.178943
         44    0.736907
         45   -1.159400
         46   -0.967689
         47    0.647143
         48    0.537769
         49    0.067311
         dtype: float64

In [17]: d2.idxmax()

Out[17]: 21

In [18]: d2.max()

Out[18]: 4.793023280955611

In [19]: data1.drop([30])
```

Out[19]:

	C1	C2	C3	C4
0	0.500820	0.103137	NaN	-0.308616
1	-2.664378	1.703656	0.021899	NaN
2	0.692876	-0.283781	0.309545	-0.053189
3	-1.658096	1.111129	-0.928600	-0.180367
4	1.720467	0.424923	0.820836	0.940913
5	1.037577	0.006962	-0.165497	0.971393
6	1.538568	-0.241214	0.741094	1.477936
7	-0.031647	-1.049431	1.104727	1.050992
8	0.228682	0.077733	0.008104	0.543471
9	NaN	-0.764144	1.216413	-0.394122
10	NaN	-1.910220	1.876340	-0.345889
11	-0.826302	NaN	NaN	1.100170
12	0.762276	0.292482	0.762421	-2.567494

d) Sort the data frame on the basis of the first column.

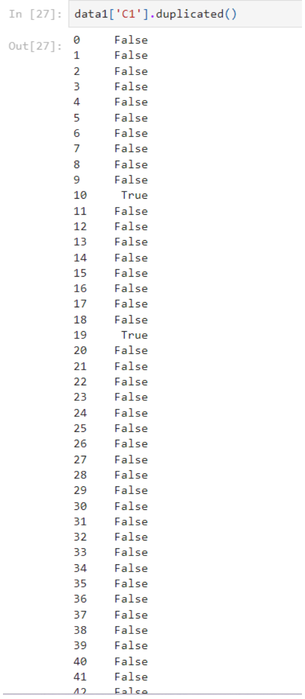


```
In [20]: data1.sort_values(by=['C1'],axis=0,ascending=True)
```

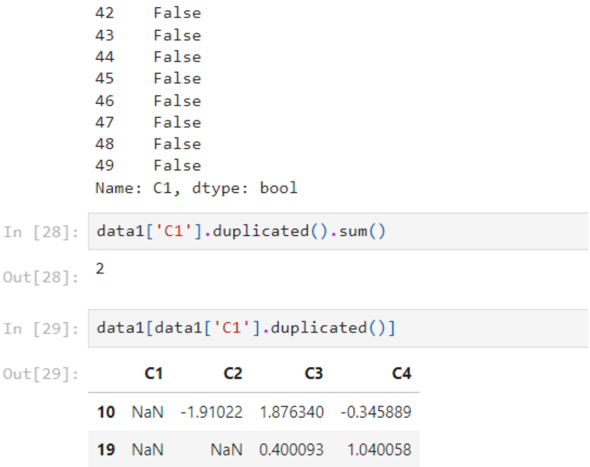
Out[20]:

	C1	C2	C3	C4
1	-2.664378	1.703656	0.021899	NaN
29	-1.918215	1.473714	-0.019083	0.118280
3	-1.658096	1.111129	-0.928600	-0.180367
40	-1.195506	1.012735	0.099677	0.260409
38	-1.102621	-1.102736	-0.154222	-0.637245
25	-0.887294	-0.360056	-0.487590	-0.155453
11	-0.826302	NaN	NaN	1.100170
45	-0.796016	-1.489899	NaN	1.126515
32	-0.723212	-0.604517	0.961274	-0.910068
43	-0.685167	0.468999	NaN	1.395111
31	-0.522917	1.573294	NaN	-0.262570
23	-0.475565	-0.712574	0.011255	-0.554051
18	-0.077314	NaN	-1.016623	2.544012
7	-0.031647	-1.049431	1.104727	1.050992
14	0.004726	-0.581460	-0.627199	0.219249
30	0.073868	-0.508457	1.303851	0.563765
47	0.176394	-0.961539	2.305811	-0.873524
49	0.194775	NaN	1.056147	-1.183611
16	0.215331	-0.067040	0.688556	-1.801321
8	0.228682	0.077733	0.008104	0.543471
39	0.234298	0.128364	0.832592	0.448082
26	0.301631	0.149824	0.512824	-1.570814
21	0.302436	3.087741	-0.079351	1.482197
36	0.374869	0.844245	-1.256829	0.034306

e) Remove all duplicates from the first column.



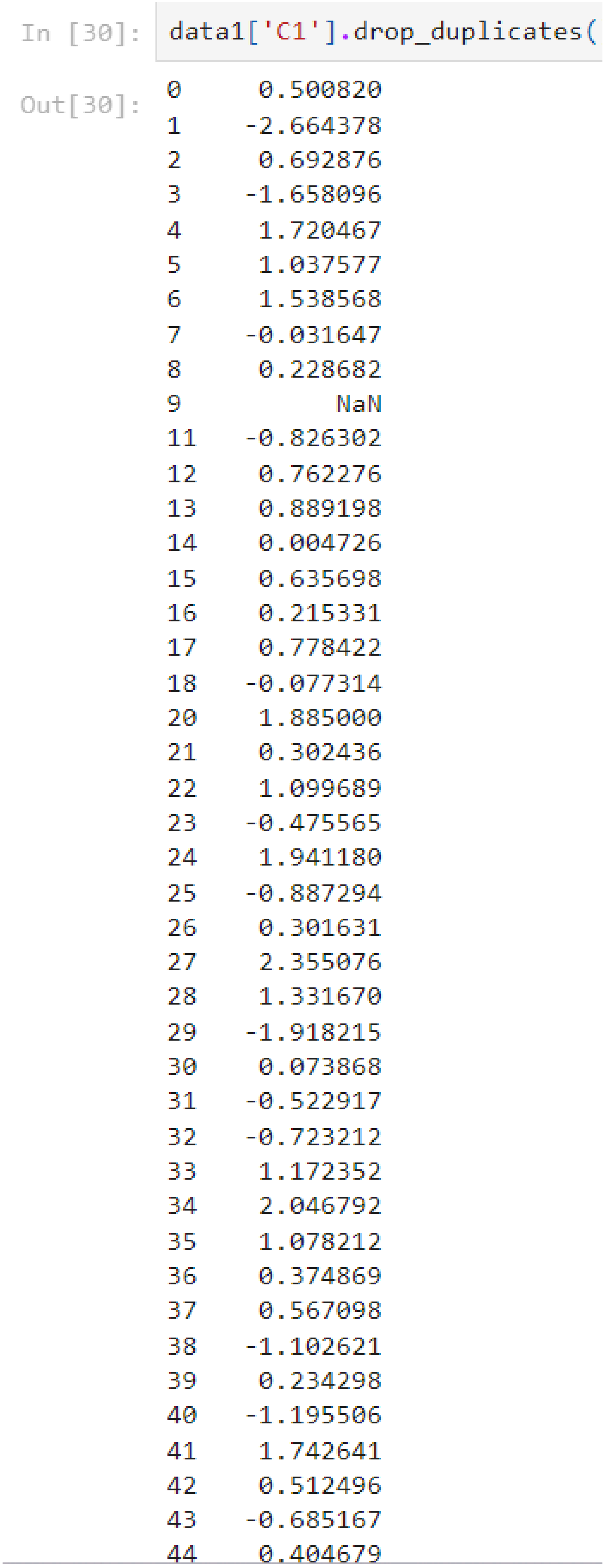
```
In [27]: data1['C1'].duplicated()

Out[27]: 0     False
         1     False
         2     False
         3     False
         4     False
         5     False
         6     False
         7     False
         8     False
         9     False
         10     True
         11    False
         12    False
         13    False
         14    False
         15    False
         16    False
         17    False
         18    False
         19     True
         20    False
         21    False
         22    False
         23    False
         24    False
         25    False
         26    False
         27    False
         28    False
         29    False
         30    False
         31    False
         32    False
         33    False
         34    False
         35    False
         36    False
         37    False
         38    False
         39    False
         40    False
         41    False
         42    False
```



```
42    False
43    False
44    False
45    False
46    False
47    False
48    False
49    False
Name: C1, dtype: bool

In [28]: data1['C1'].duplicated().sum()

Out[28]: 2

In [29]: data1[data1['C1'].duplicated()]
```

Out[29]:

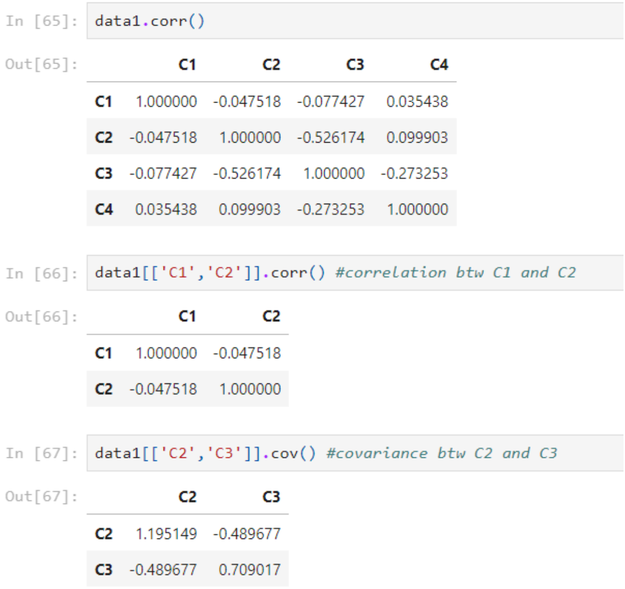
	C1	C2	C3	C4
10	NaN	-1.91022	1.876340	-0.345889
19	NaN	NaN	0.400093	1.040058



```
In [30]:  data1['C1'].drop_duplicates(
```

```
Out[30]:  0      0.500820
          1     -2.664378
          2      0.692876
          3     -1.658096
          4      1.720467
          5      1.037577
          6      1.538568
          7     -0.031647
          8      0.228682
          9           NaN
          11    -0.826302
          12     0.762276
          13     0.889198
          14     0.004726
          15     0.635698
          16     0.215331
          17     0.778422
          18    -0.077314
          20     1.885000
          21     0.302436
          22     1.099689
          23    -0.475565
          24     1.941180
          25    -0.887294
          26     0.301631
          27     2.355076
          28     1.331670
          29    -1.918215
          30     0.073868
          31    -0.522917
          32    -0.723212
          33     1.172352
          34     2.046792
          35     1.078212
          36     0.374869
          37     0.567098
          38    -1.102621
          39     0.234298
          40    -1.195506
          41     1.742641
          42     0.512496
          43    -0.685167
          44     0.404679
```

f) Find the correlation between first and second column and covariance between second and third column.



```
In [65]: data1.corr()
```

Out[65]:

	C1	C2	C3	C4
C1	1.000000	-0.047518	-0.077427	0.035438
C2	-0.047518	1.000000	-0.526174	0.099903
C3	-0.077427	-0.526174	1.000000	-0.273253
C4	0.035438	0.099903	-0.273253	1.000000

```
In [66]: data1[['C1','C2']].corr() #correlation btw C1 and C2
```

Out[66]:

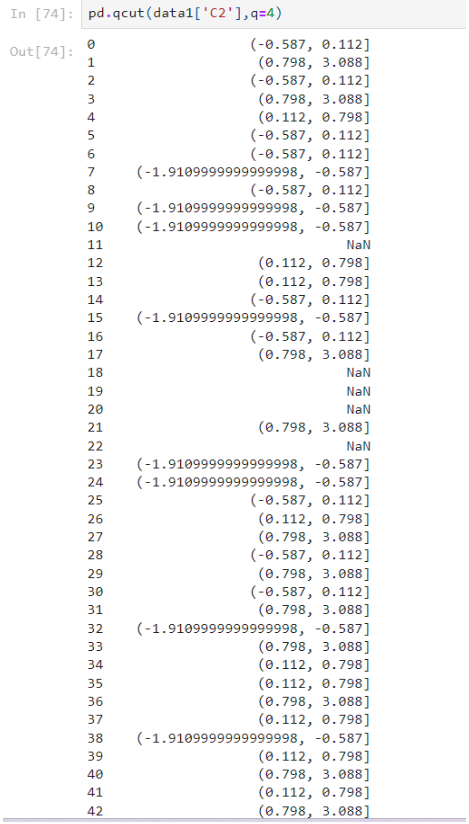
	C1	C2
C1	1.000000	-0.047518
C2	-0.047518	1.000000

```
In [67]: data1[['C2','C3']].cov() #covariance btw C2 and C3
```

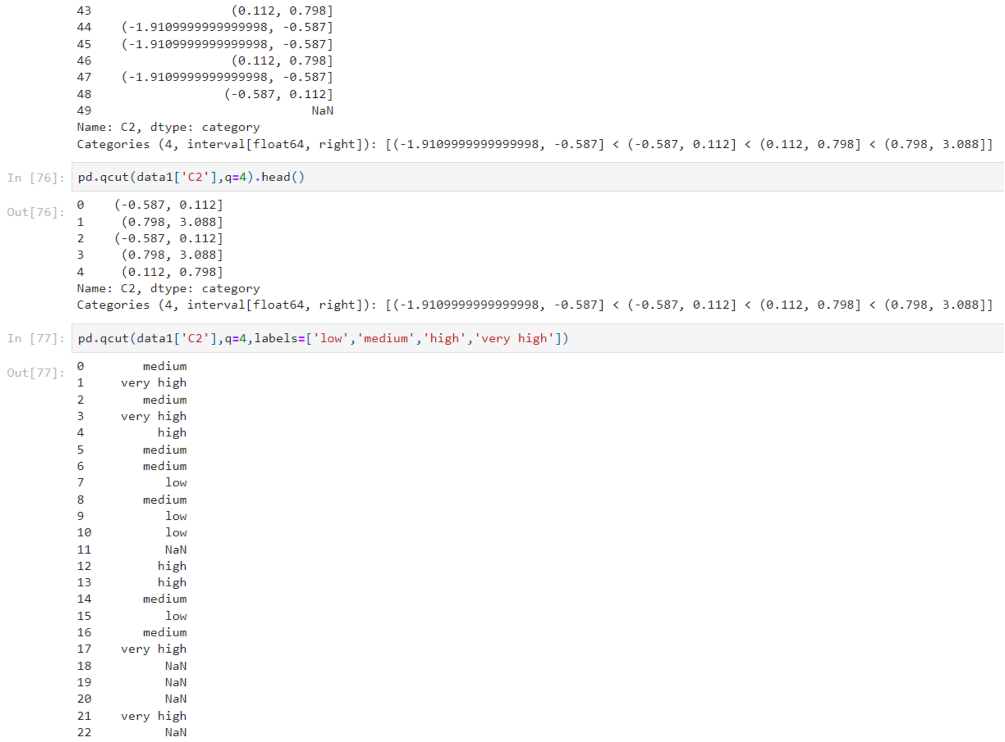
Out[67]:

	C2	C3
C2	1.195149	-0.489677
C3	-0.489677	0.709017

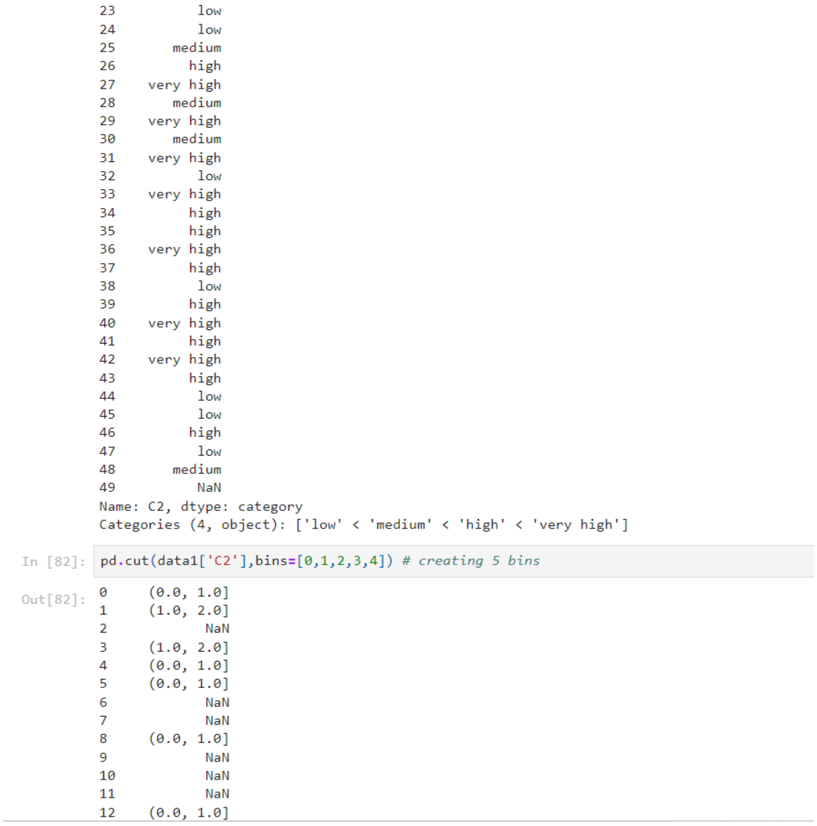
g) Discretize the second column and create 5 bins.



```
In [74]: pd.qcut(data1['C2'],q=4)

Out[74]: 0                      (-0.587, 0.112]
         1                       (0.798, 3.088]
         2                      (-0.587, 0.112]
         3                       (0.798, 3.088]
         4                       (0.112, 0.798]
         5                      (-0.587, 0.112]
         6                      (-0.587, 0.112]
         7     (-1.9109999999999998, -0.587]
         8                      (-0.587, 0.112]
         9     (-1.9109999999999998, -0.587]
         10    (-1.9109999999999998, -0.587]
         11                                NaN
         12                      (0.112, 0.798]
         13                      (0.112, 0.798]
         14                     (-0.587, 0.112]
         15    (-1.9109999999999998, -0.587]
         16                     (-0.587, 0.112]
         17                      (0.798, 3.088]
         18                                NaN
         19                                NaN
         20                                NaN
         21                      (0.798, 3.088]
         22                                NaN
         23    (-1.9109999999999998, -0.587]
         24    (-1.9109999999999998, -0.587]
         25                     (-0.587, 0.112]
         26                      (0.112, 0.798]
         27                      (0.798, 3.088]
         28                     (-0.587, 0.112]
         29                      (0.798, 3.088]
         30                     (-0.587, 0.112]
         31                      (0.798, 3.088]
         32    (-1.9109999999999998, -0.587]
         33                      (0.798, 3.088]
         34                      (0.112, 0.798]
         35                      (0.112, 0.798]
         36                      (0.798, 3.088]
         37                      (0.112, 0.798]
         38    (-1.9109999999999998, -0.587]
         39                      (0.112, 0.798]
         40                      (0.798, 3.088]
         41                      (0.112, 0.798]
         42                      (0.798, 3.088]
```



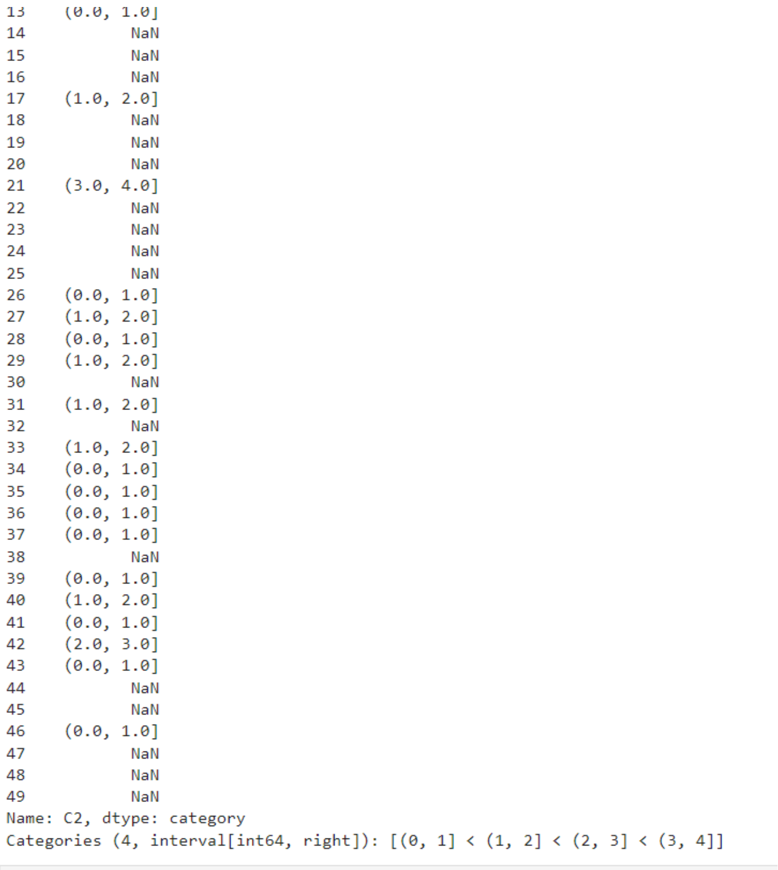
```
         43                      (0.112, 0.798]
         44    (-1.9109999999999998, -0.587]
         45    (-1.9109999999999998, -0.587]
         46                      (0.112, 0.798]
         47    (-1.9109999999999998, -0.587]
         48                     (-0.587, 0.112]
         49                                NaN
         Name: C2, dtype: category
         Categories (4, interval[float64, right]): [(-1.9109999999999998, -0.587] < (-0.587, 0.112] < (0.112, 0.798] < (0.798, 3.088]]

In [76]: pd.qcut(data1['C2'],q=4).head()

Out[76]: 0    (-0.587, 0.112]
         1     (0.798, 3.088]
         2    (-0.587, 0.112]
         3     (0.798, 3.088]
         4     (0.112, 0.798]
         Name: C2, dtype: category
         Categories (4, interval[float64, right]): [(-1.9109999999999998, -0.587] < (-0.587, 0.112] < (0.112, 0.798] < (0.798, 3.088]]

In [77]: pd.qcut(data1['C2'],q=4,labels=['low','medium','high','very high'])

Out[77]: 0        medium
         1     very high
         2        medium
         3     very high
         4          high
         5        medium
         6        medium
         7           low
         8        medium
         9           low
         10          low
         11          NaN
         12         high
         13         high
         14       medium
         15          low
         16       medium
         17    very high
         18          NaN
         19          NaN
         20          NaN
         21    very high
         22          NaN
```



```
23         low
24         low
25      medium
26        high
27   very high
28      medium
29   very high
30      medium
31   very high
32         low
33   very high
34        high
35        high
36   very high
37        high
38         low
39        high
40   very high
41        high
42   very high
43        high
44         low
45         low
46        high
47         low
48      medium
49         NaN
Name: C2, dtype: category
Categories (4, object): ['low' < 'medium' < 'high' < 'very high']
```

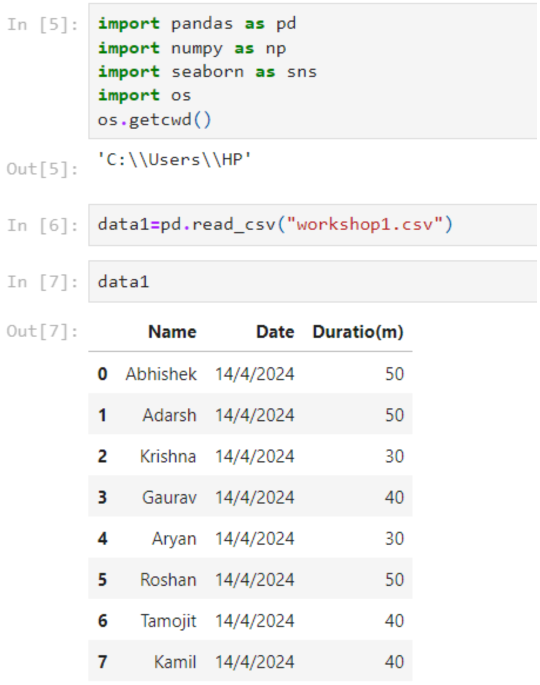
```
In [82]: pd.cut(data1['C2'],bins=[0,1,2,3,4]) # creating 5 bins
```

```
Out[82]: 0     (0.0, 1.0]
         1     (1.0, 2.0]
         2            NaN
         3     (1.0, 2.0]
         4     (0.0, 1.0]
         5     (0.0, 1.0]
         6            NaN
         7            NaN
         8     (0.0, 1.0]
         9            NaN
         10           NaN
         11           NaN
         12    (0.0, 1.0]
```



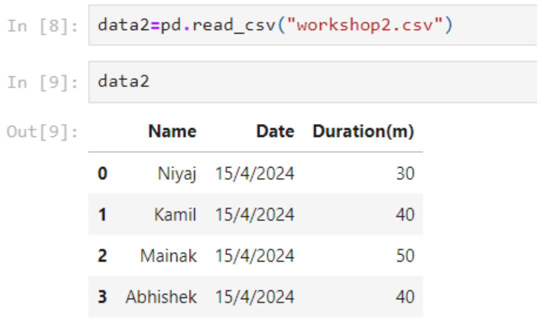
```
13    (0.0, 1.0]
14           NaN
15           NaN
16           NaN
17    (1.0, 2.0]
18           NaN
19           NaN
20           NaN
21    (3.0, 4.0]
22           NaN
23           NaN
24           NaN
25           NaN
26    (0.0, 1.0]
27    (1.0, 2.0]
28    (0.0, 1.0]
29    (1.0, 2.0]
30           NaN
31    (1.0, 2.0]
32           NaN
33    (1.0, 2.0]
34    (0.0, 1.0]
35    (0.0, 1.0]
36    (0.0, 1.0]
37    (0.0, 1.0]
38           NaN
39    (0.0, 1.0]
40    (1.0, 2.0]
41    (0.0, 1.0]
42    (2.0, 3.0]
43    (0.0, 1.0]
44           NaN
45           NaN
46    (0.0, 1.0]
47           NaN
48           NaN
49           NaN
Name: C2, dtype: category
Categories (4, interval[int64, right]): [(0, 1] < (1, 2] < (2, 3] < (3, 4]]
```

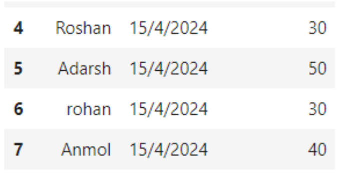
4) Consider two excel files having attendance of two workshos. Each file has three fields ‘Name’, ‘Date, duration (in minutes) where names are unique within a file. Note that duration may take one of three values (30, 40, 50) only. Import the data into two data frames and do the following:



```
In [5]: import pandas as pd
        import numpy as np
        import seaborn as sns
        import os
        os.getcwd()

Out[5]: 'C:\\Users\\HP'

In [6]: data1=pd.read_csv("workshop1.csv")

In [7]: data1

Out[7]:
```

	Name	Date	Duratio(m)
0	Abhishek	14/4/2024	50
1	Adarsh	14/4/2024	50
2	Krishna	14/4/2024	30
3	Gaurav	14/4/2024	40
4	Aryan	14/4/2024	30
5	Roshan	14/4/2024	50
6	Tamojit	14/4/2024	40
7	Kamil	14/4/2024	40

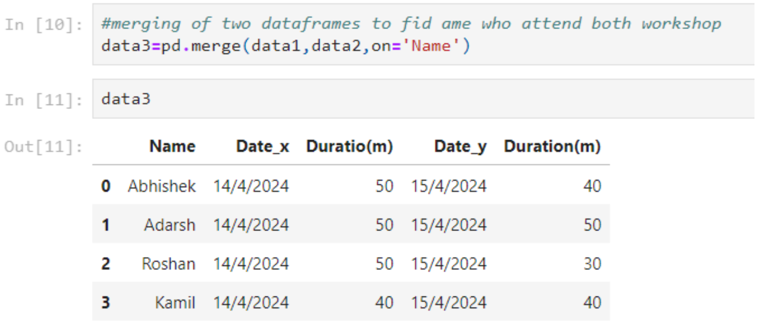




```
In [8]: data2=pd.read_csv("workshop2.csv")

In [9]: data2

Out[9]:
```

	Name	Date	Duration(m)
0	Niyaj	15/4/2024	30
1	Kamil	15/4/2024	40
2	Mainak	15/4/2024	50
3	Abhishek	15/4/2024	40
4	Roshan	15/4/2024	30
5	Adarsh	15/4/2024	50
6	rohan	15/4/2024	30
7	Anmol	15/4/2024	40

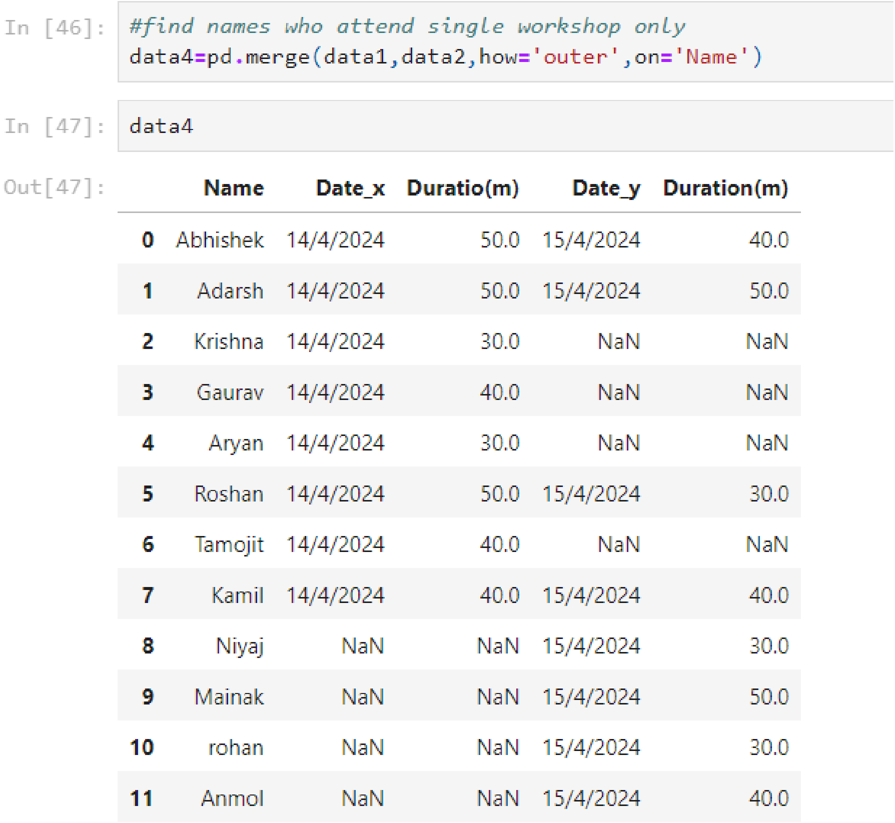
a) Perform merging of the two data frames to find the names of students who had attended both workshops.



```
In [10]: #merging of two dataframes to fid ame who attend both workshop
         data3=pd.merge(data1,data2,on='Name')

In [11]: data3

Out[11]:
```

	Name	Date_x	Duratio(m)	Date_y	Duration(m)
0	Abhishek	14/4/2024	50	15/4/2024	40
1	Adarsh	14/4/2024	50	15/4/2024	50
2	Roshan	14/4/2024	50	15/4/2024	30
3	Kamil	14/4/2024	40	15/4/2024	40

b) Find names of all students who have attended a single workshop only



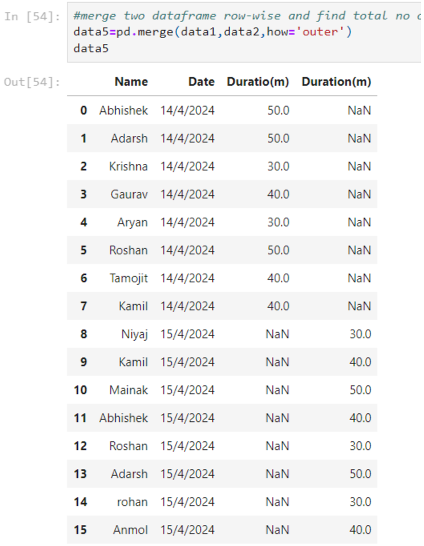
```
In [46]: #find names who attend single workshop only
         data4=pd.merge(data1,data2,how='outer',on='Name')
```

```
In [47]: data4
```

Out[47]:

	Name	Date_x	Duratio(m)	Date_y	Duration(m)
0	Abhishek	14/4/2024	50.0	15/4/2024	40.0
1	Adarsh	14/4/2024	50.0	15/4/2024	50.0
2	Krishna	14/4/2024	30.0	NaN	NaN
3	Gaurav	14/4/2024	40.0	NaN	NaN
4	Aryan	14/4/2024	30.0	NaN	NaN
5	Roshan	14/4/2024	50.0	15/4/2024	30.0
6	Tamojit	14/4/2024	40.0	NaN	NaN
7	Kamil	14/4/2024	40.0	15/4/2024	40.0
8	Niyaj	NaN	NaN	15/4/2024	30.0
9	Mainak	NaN	NaN	15/4/2024	50.0
10	rohan	NaN	NaN	15/4/2024	30.0
11	Anmol	NaN	NaN	15/4/2024	40.0

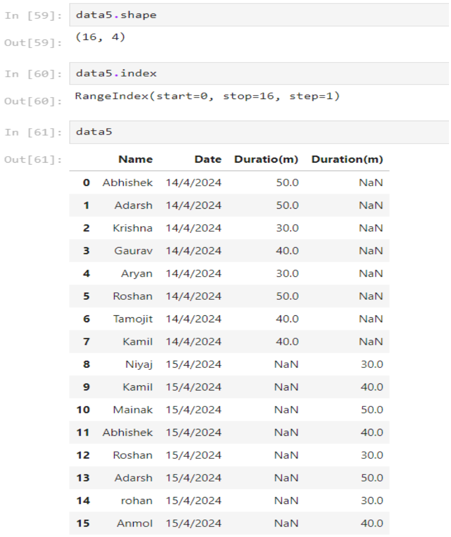
c) Merge two data frames row-wise and find the total number of records in the data frame.



```
In [54]: #merge two dataframe row-wise and find total no c
         data5=pd.merge(data1,data2,how='outer')
         data5
```

Out[54]:

	Name	Date	Duratio(m)	Duration(m)
0	Abhishek	14/4/2024	50.0	NaN
1	Adarsh	14/4/2024	50.0	NaN
2	Krishna	14/4/2024	30.0	NaN
3	Gaurav	14/4/2024	40.0	NaN
4	Aryan	14/4/2024	30.0	NaN
5	Roshan	14/4/2024	50.0	NaN
6	Tamojit	14/4/2024	40.0	NaN
7	Kamil	14/4/2024	40.0	NaN
8	Niyaj	15/4/2024	NaN	30.0
9	Kamil	15/4/2024	NaN	40.0
10	Mainak	15/4/2024	NaN	50.0
11	Abhishek	15/4/2024	NaN	40.0
12	Roshan	15/4/2024	NaN	30.0
13	Adarsh	15/4/2024	NaN	50.0
14	rohan	15/4/2024	NaN	30.0
15	Anmol	15/4/2024	NaN	40.0

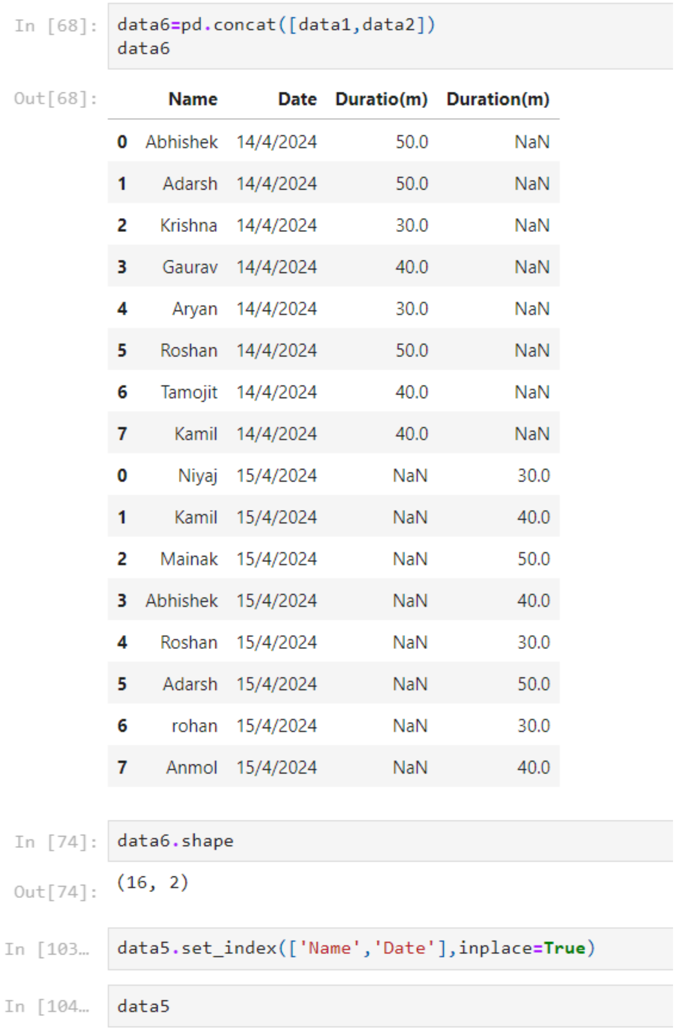


```
In [59]: data5.shape
Out[59]: (16, 4)

In [60]: data5.index
Out[60]: RangeIndex(start=0, stop=16, step=1)

In [61]: data5
```

Out[61]:

	Name	Date	Duratio(m)	Duration(m)
0	Abhishek	14/4/2024	50.0	NaN
1	Adarsh	14/4/2024	50.0	NaN
2	Krishna	14/4/2024	30.0	NaN
3	Gaurav	14/4/2024	40.0	NaN
4	Aryan	14/4/2024	30.0	NaN
5	Roshan	14/4/2024	50.0	NaN
6	Tamojit	14/4/2024	40.0	NaN
7	Kamil	14/4/2024	40.0	NaN
8	Niyaj	15/4/2024	NaN	30.0
9	Kamil	15/4/2024	NaN	40.0
10	Mainak	15/4/2024	NaN	50.0
11	Abhishek	15/4/2024	NaN	40.0
12	Roshan	15/4/2024	NaN	30.0
13	Adarsh	15/4/2024	NaN	50.0
14	rohan	15/4/2024	NaN	30.0
15	Anmol	15/4/2024	NaN	40.0

d) Merge two data frames row-wise and use two columns viz. names and dates as multi-row indexes. Generate descriptive statistics for this hierarchical data frame.

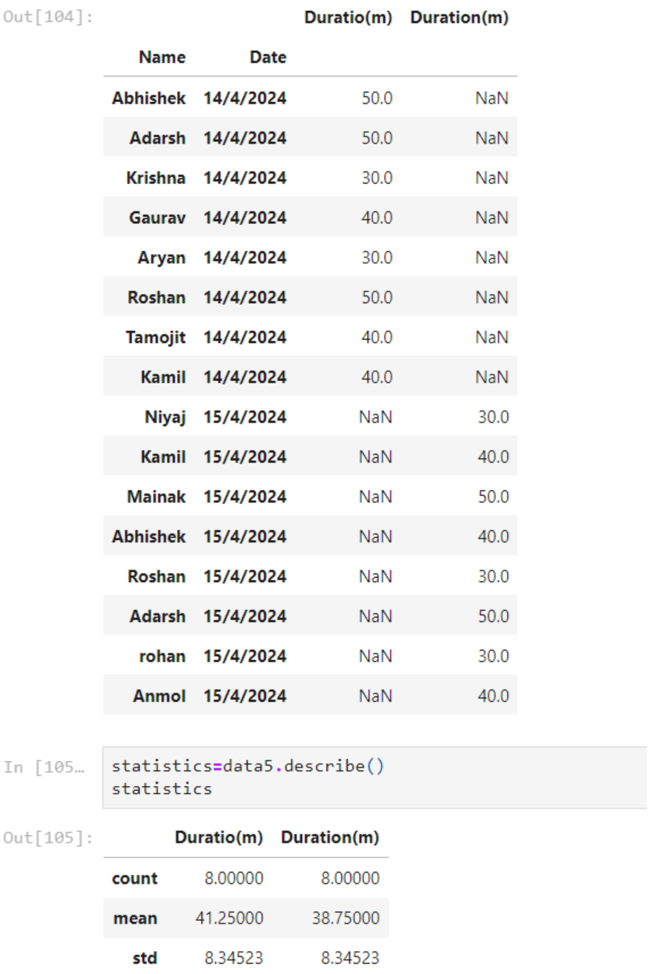


```
In [68]: data6=pd.concat([data1,data2])
         data6
```

Out[68]:

	Name	Date	Duratio(m)	Duration(m)
0	Abhishek	14/4/2024	50.0	NaN
1	Adarsh	14/4/2024	50.0	NaN
2	Krishna	14/4/2024	30.0	NaN
3	Gaurav	14/4/2024	40.0	NaN
4	Aryan	14/4/2024	30.0	NaN
5	Roshan	14/4/2024	50.0	NaN
6	Tamojit	14/4/2024	40.0	NaN
7	Kamil	14/4/2024	40.0	NaN
0	Niyaj	15/4/2024	NaN	30.0
1	Kamil	15/4/2024	NaN	40.0
2	Mainak	15/4/2024	NaN	50.0
3	Abhishek	15/4/2024	NaN	40.0
4	Roshan	15/4/2024	NaN	30.0
5	Adarsh	15/4/2024	NaN	50.0
6	rohan	15/4/2024	NaN	30.0
7	Anmol	15/4/2024	NaN	40.0

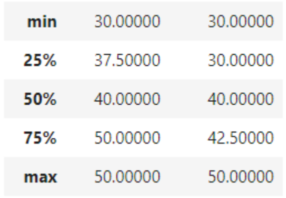
```
In [74]: data6.shape

Out[74]: (16, 2)

In [103… data5.set_index(['Name','Date'],inplace=True)

In [104… data5
```



```
Out[104]:                   Duratio(m)  Duration(m)
          Name       Date
      Abhishek  14/4/2024         50.0          NaN
        Adarsh  14/4/2024         50.0          NaN
       Krishna  14/4/2024         30.0          NaN
        Gaurav  14/4/2024         40.0          NaN
         Aryan  14/4/2024         30.0          NaN
        Roshan  14/4/2024         50.0          NaN
       Tamojit  14/4/2024         40.0          NaN
         Kamil  14/4/2024         40.0          NaN
         Niyaj  15/4/2024          NaN         30.0
         Kamil  15/4/2024          NaN         40.0
        Mainak  15/4/2024          NaN         50.0
      Abhishek  15/4/2024          NaN         40.0
        Roshan  15/4/2024          NaN         30.0
        Adarsh  15/4/2024          NaN         50.0
         rohan  15/4/2024          NaN         30.0
        Anmol   15/4/2024          NaN         40.0
```

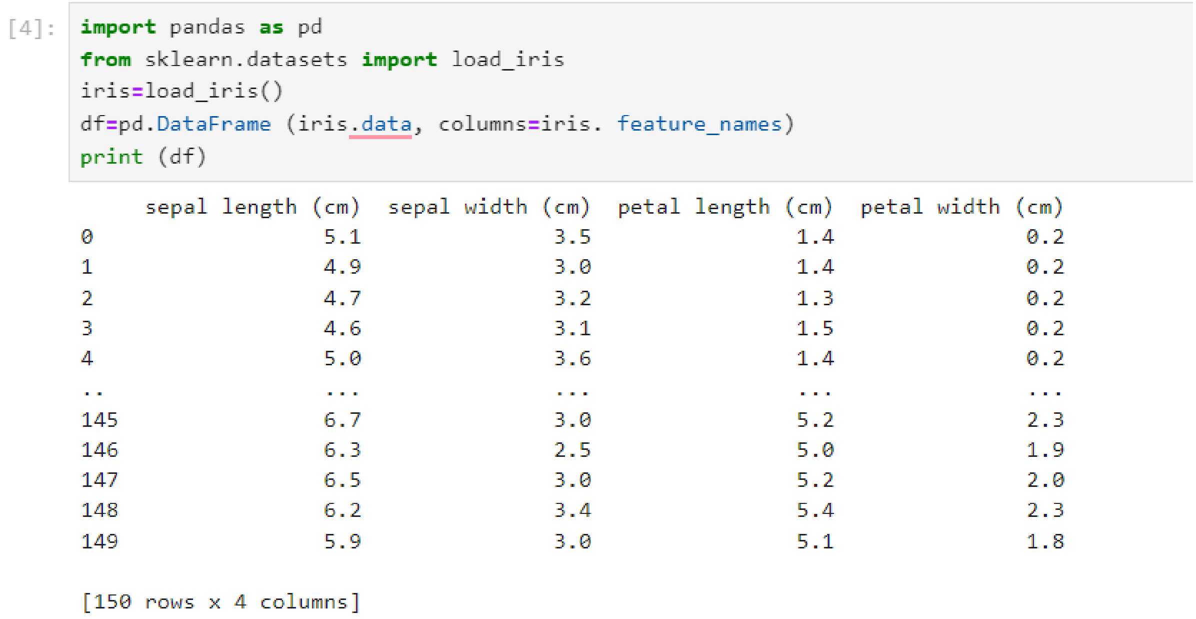
```
In [105…  statistics=data5.describe()
          statistics
```

```
Out[105]:        Duratio(m)  Duration(m)
          count     8.00000      8.00000
          mean     41.25000     38.75000
          std       8.34523      8.34523
```



```
          min      30.00000     30.00000
          25%      37.50000     30.00000
          50%      40.00000     40.00000
          75%      50.00000     42.50000
          max      50.00000     50.00000
```

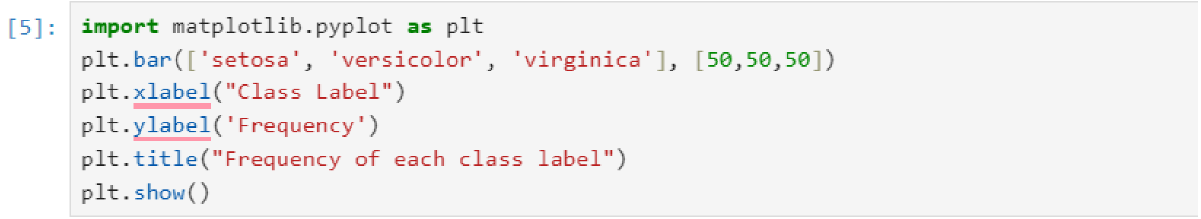
5) Using Iris data, plot the following with proper legend and axis labels: (Download IRIS data from: https://archive.ics.uci.edu/ml/datasets/iris or import it from sklearn datasets)



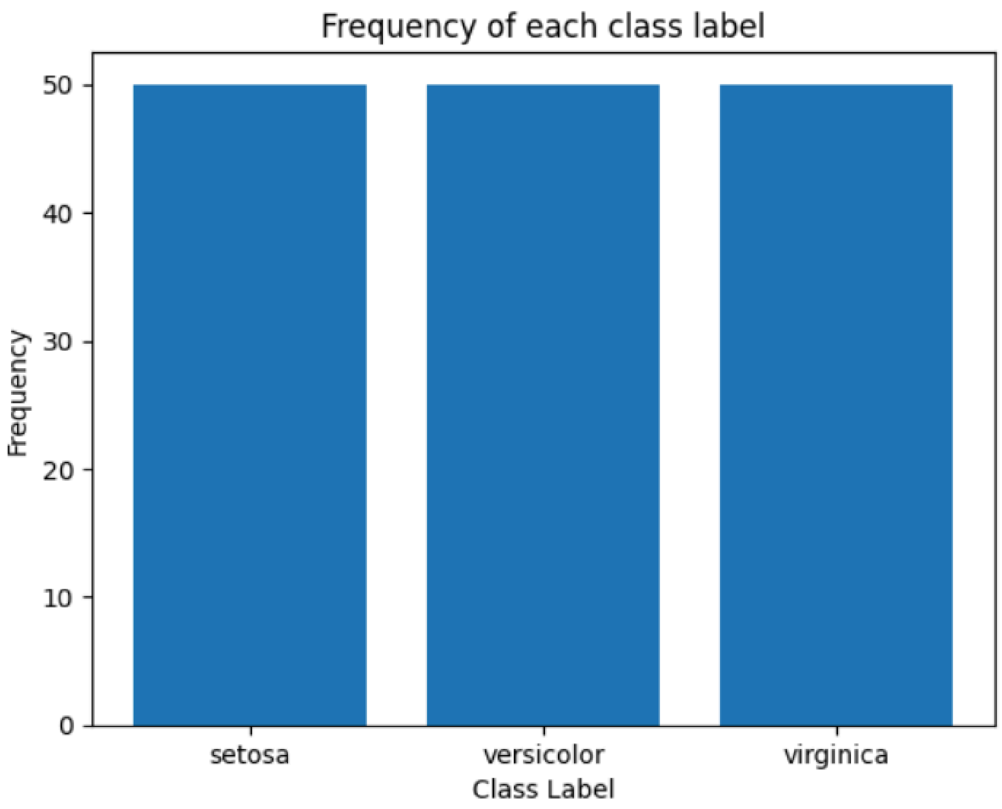
```
[4]: import pandas as pd
     from sklearn.datasets import load_iris
     iris=load_iris()
     df=pd.DataFrame (iris.data, columns=iris. feature_names)
     print (df)
```

```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                  5.1               3.5                1.4               0.2
1                  4.9               3.0                1.4               0.2
2                  4.7               3.2                1.3               0.2
3                  4.6               3.1                1.5               0.2
4                  5.0               3.6                1.4               0.2
..                 ...               ...                ...               ...
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8

[150 rows x 4 columns]
```

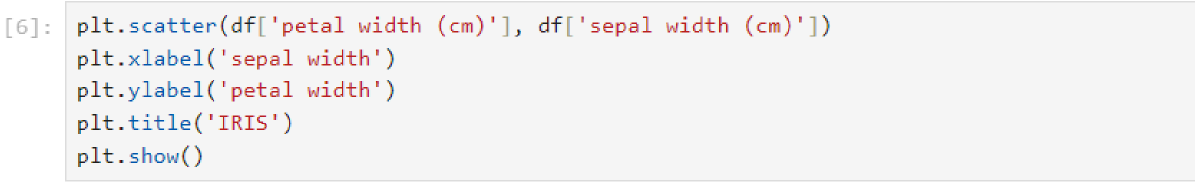
a) Plot bar chart to show the frequency of each class label in the data.



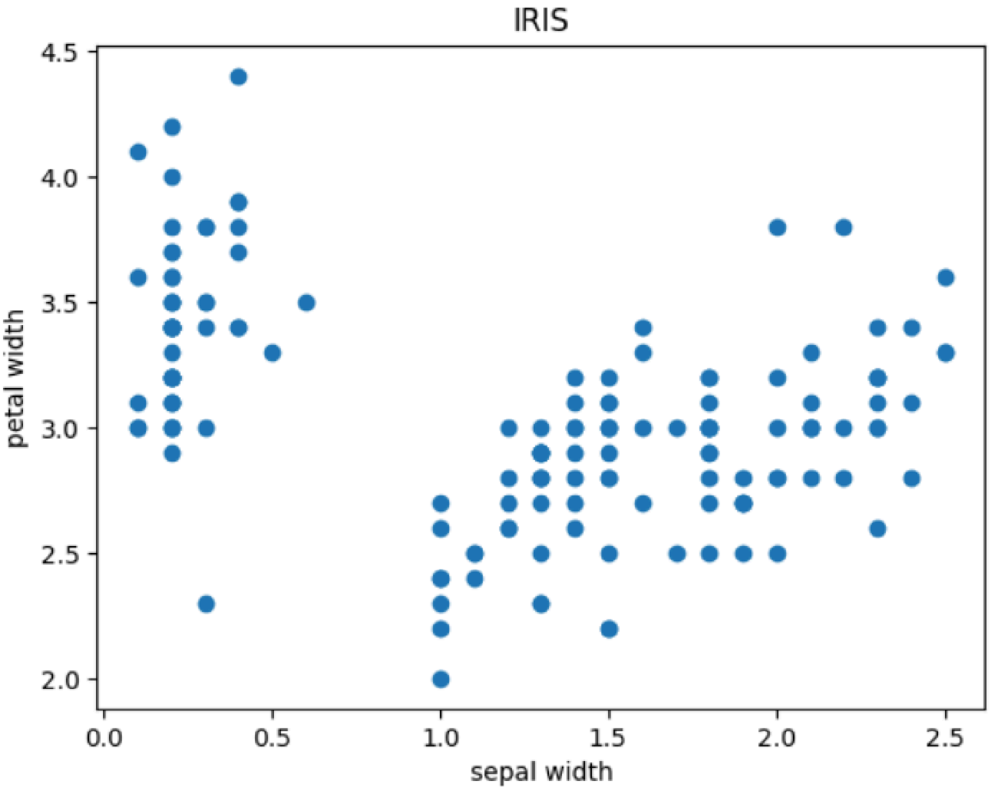
```
[5]: import matplotlib.pyplot as plt
     plt.bar(['setosa', 'versicolor', 'virginica'], [50,50,50])
     plt.xlabel("Class Label")
     plt.ylabel('Frequency')
     plt.title("Frequency of each class label")
     plt.show()
```



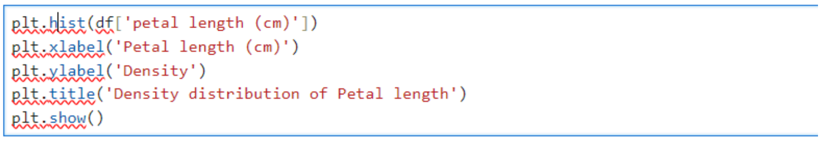
b) Draw a scatter plot for Petal width vs sepal width and fit a regression line



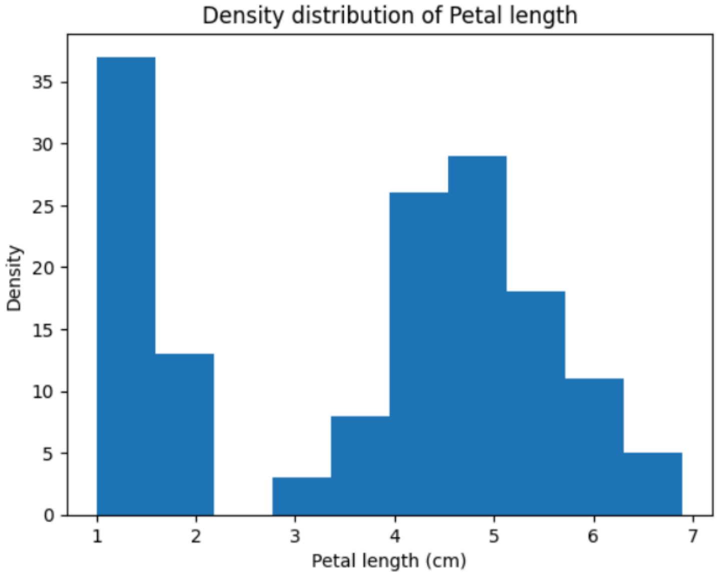
```
[6]: plt.scatter(df['petal width (cm)'], df['sepal width (cm)'])
     plt.xlabel('sepal width')
     plt.ylabel('petal width')
     plt.title('IRIS')
     plt.show()
```



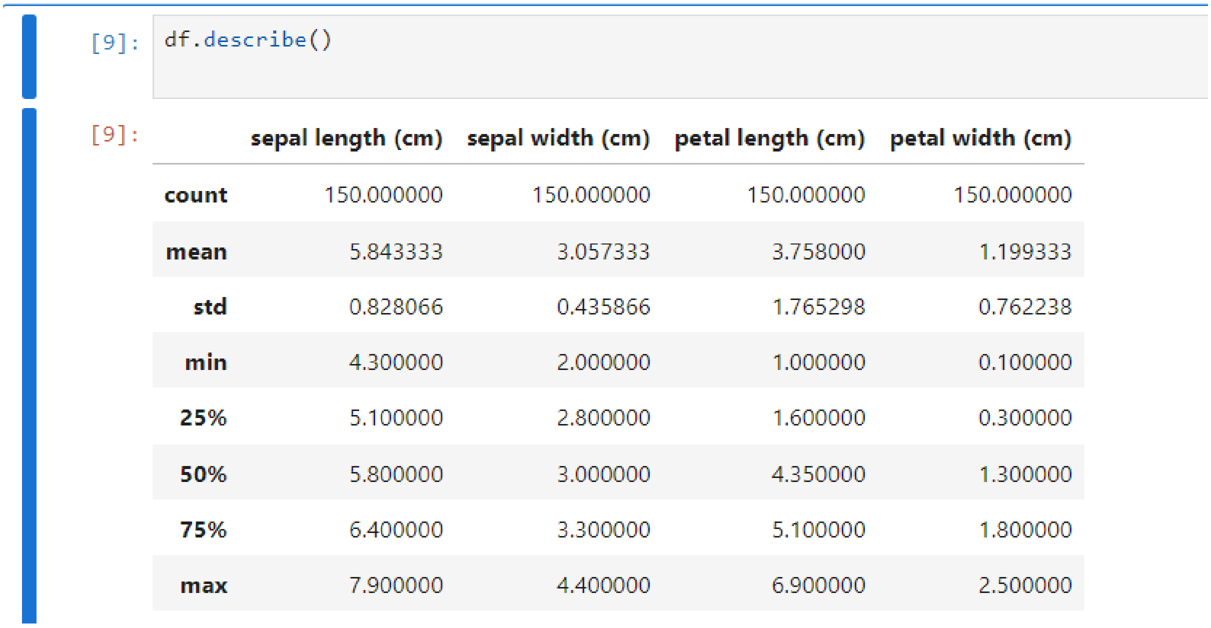
c) Plot density distribution for feature petal length.



```
plt.hist(df['petal length (cm)'])
plt.xlabel('Petal length (cm)')
plt.ylabel('Density')
plt.title('Density distribution of Petal length')
plt.show()
```



d) Compute mean, mode, median, standard deviation, confidence interval and standard error for each feature



```
[9]: df.describe()
```

[9]:

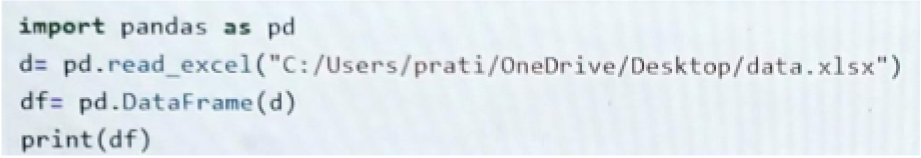
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

6. Consider the following data frame containing a family name, gender of the family member and her/his monthly income in each record.

Name	Gender	MonthlyIncome (Rs.)
Shah	Male	114000.00
Vats	Male	65000.00
Vats	Female	43150.00
Kumar	Female	69500.00
Vats	Female	155000.00
Kumar	Male	103000.00
Shah	Male	55000.00
Shah	Female	112400.00
Kumar	Female	81030.00
Vats	Male	71900.00

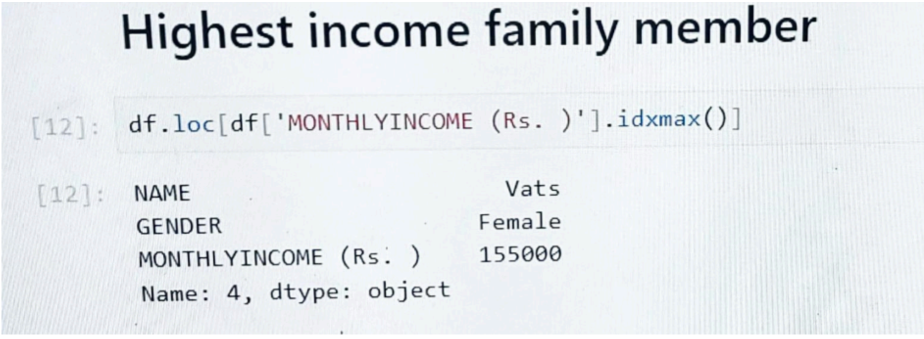
Write a program in Python using Pandas to perform the following:

a. Calculate and display familywise gross monthly income



```
import pandas as pd
d= pd.read_excel("C:/Users/prati/OneDrive/Desktop/data.xlsx")
df= pd.DataFrame(d)
print(df)
```

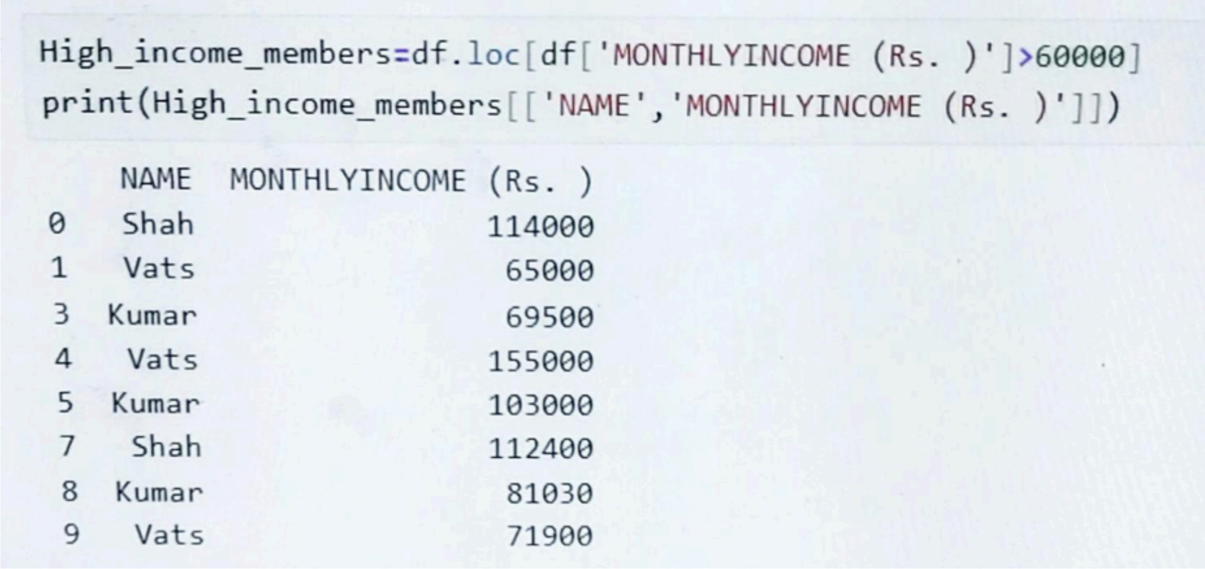
b. Calculate and display the member with the highest monthly income



Highest income family member

```
[12]: df.loc[df['MONTHLYINCOME (Rs. )'].idxmax()]

[12]: NAME                          Vats
      GENDER                      Female
      MONTHLYINCOME (Rs. )        155000
      Name: 4, dtype: object
```

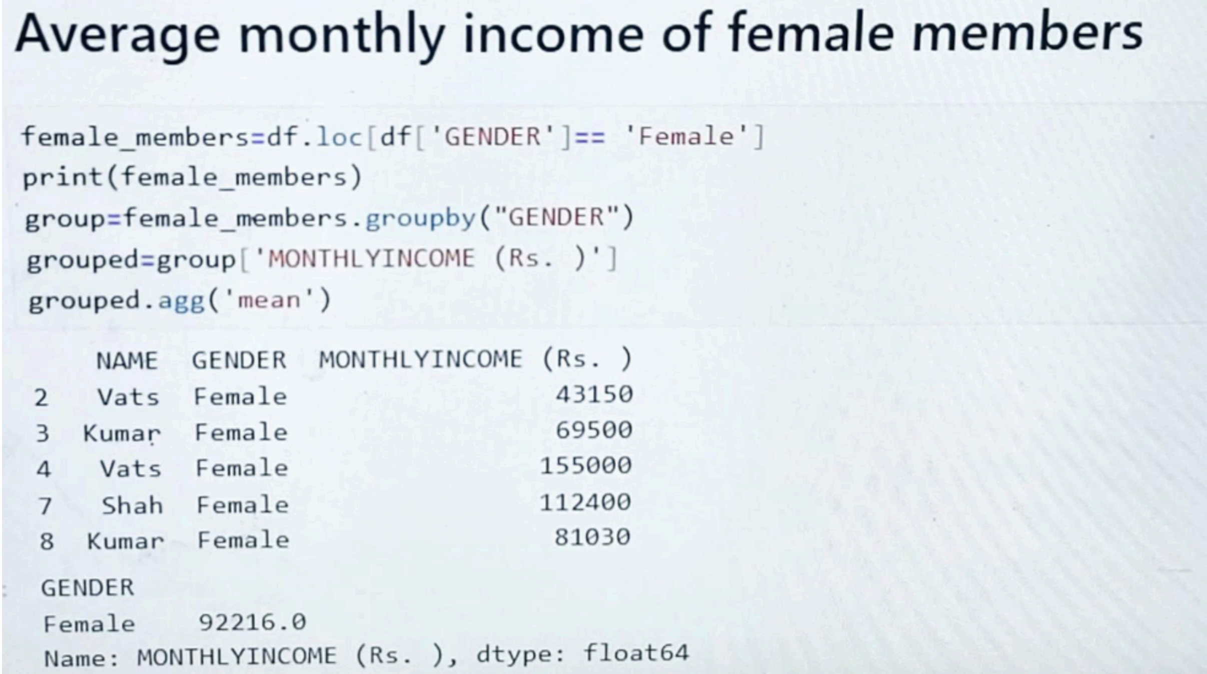
C. Calculate and display monthly income of all members with income greater than Rs. 60000.00



```
High_income_members=df.loc[df['MONTHLYINCOME (Rs. )']>60000]
print(High_income_members[['NAME','MONTHLYINCOME (Rs. )']])
```

```
    NAME  MONTHLYINCOME (Rs. )
0   Shah                114000
1   Vats                 65000
3  Kumar                 69500
4   Vats                155000
5  Kumar                103000
7   Shah                112400
8  Kumar                 81030
9   Vats                 71900
```

D. Calculate and display the average monthly income of the female members.



Average monthly income of female members

```
female_members=df.loc[df['GENDER']== 'Female']
print(female_members)
group=female_members.groupby("GENDER")
grouped=group['MONTHLYINCOME (Rs. )']
grouped.agg('mean')
```

```
    NAME  GENDER  MONTHLYINCOME (Rs. )
2   Vats  Female                 43150
3  Kumar  Female                 69500
4   Vats  Female                155000
7   Shah  Female                112400
8  Kumar  Female                 81030

GENDER
Female    92216.0
Name: MONTHLYINCOME (Rs. ), dtype: float64
```