KALINDI COLLEGE

UNIVERSITY OF DELHI

NAME: Prachi Aggarwal

COURSE: Bsc(H) Computer Science

ROLLNO: 21570015

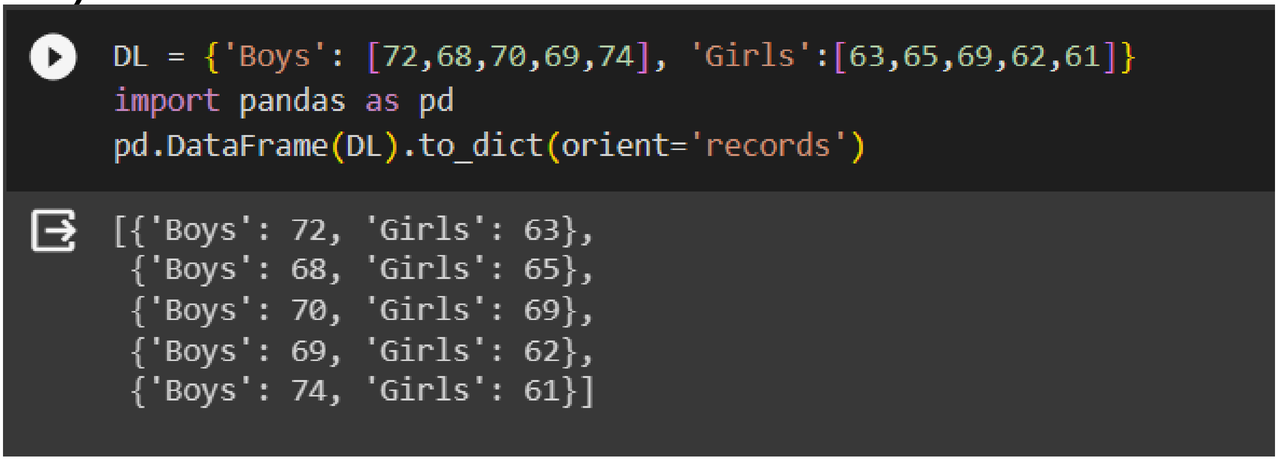
UPC: 32347507

UNIVERSITY RNO:21033570042

SUBJECT: Data Analysis and Visualization

Ques1) Given below is a dictionary having two keys 'Boys' and 'Girls' and having two lists of heights of five Boys and
Five Girls respectively as values associated with these keys
Original dictionary of lists:
{'Boys': [72, 68, 70, 69, 74], 'Girls': [63, 65, 69, 62, 61]}
From the given dictionary of lists create the following list of dictionaries:
[{'Boys': 72, 'Girls': 63}, {'Boys': 68, 'Girls': 65}, {'Boys': 70, 'Girls': 69}, {'Boys': 69, 'Girls': 62}, {'Boys':74, 'Girls':61]

Ans)

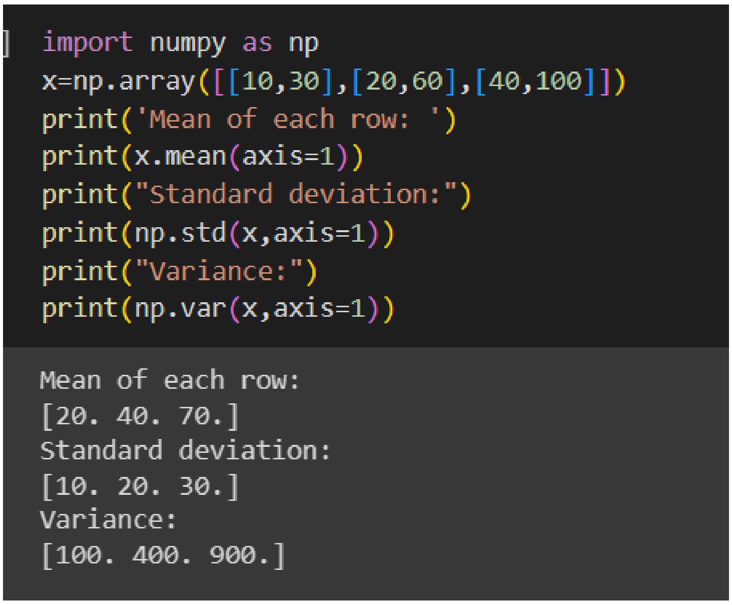


```
DL = {'Boys': [72,68,70,69,74], 'Girls':[63,65,69,62,61]}
import pandas as pd
pd.DataFrame(DL).to_dict(orient='records')
```

```
[{'Boys': 72, 'Girls': 63},
 {'Boys': 68, 'Girls': 65},
 {'Boys': 70, 'Girls': 69},
 {'Boys': 69, 'Girls': 62},
 {'Boys': 74, 'Girls': 61}]
```

Ques2) Write programs in Python using NumPy library to do the following:

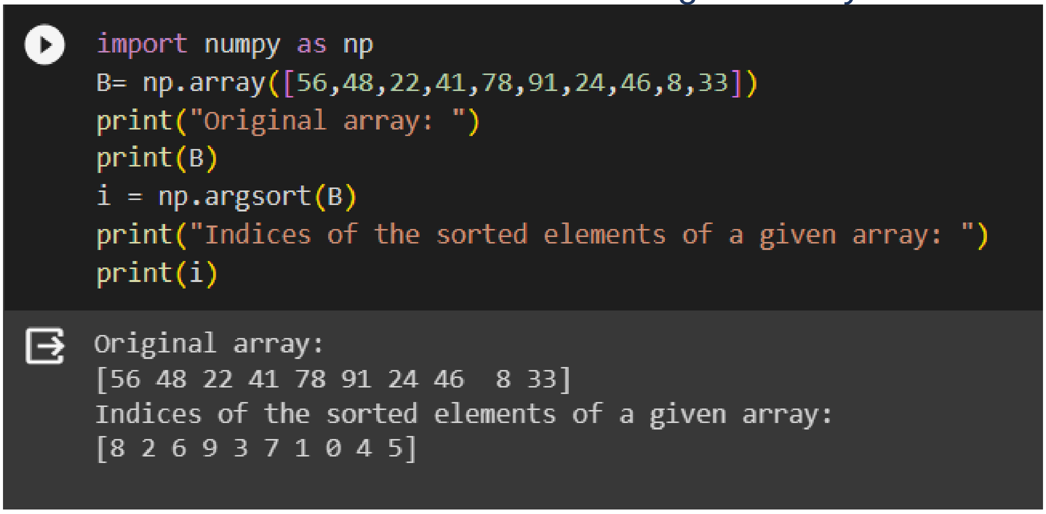
a. Compute the mean, standard deviation, and variance of a two dimensional random integer array along the second axis.



```
import numpy as np
x=np.array([[10,30],[20,60],[40,100]])
print('Mean of each row: ')
print(x.mean(axis=1))
print("Standard deviation:")
print(np.std(x,axis=1))
print("Variance:")
print(np.var(x,axis=1))
```

```
Mean of each row:
[20. 40. 70.]
Standard deviation:
[10. 20. 30.]
Variance:
[100. 400. 900.]
```

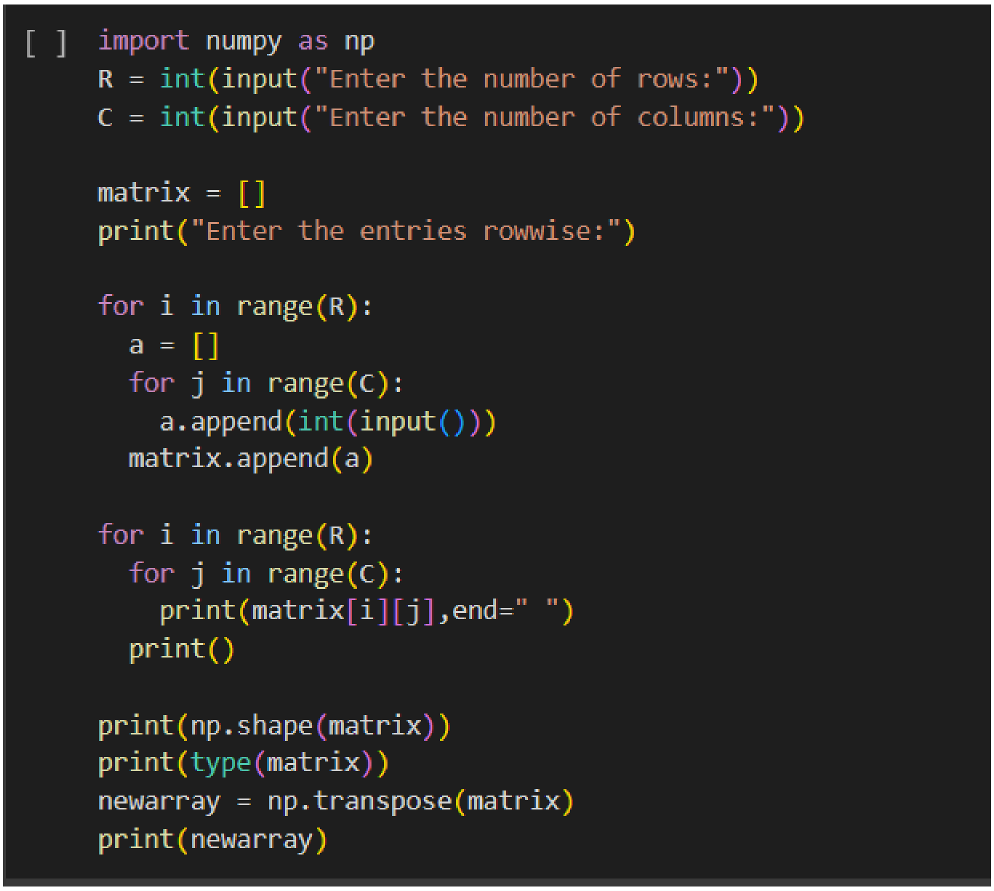
b. Get the indices of the sorted elements of a given array.



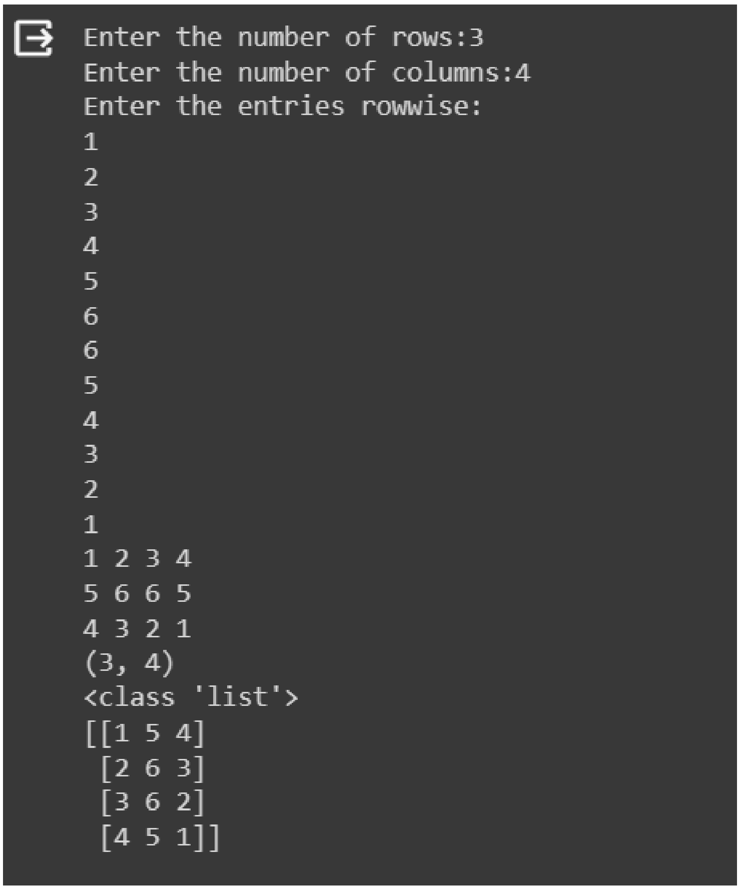
```
import numpy as np
B= np.array([56,48,22,41,78,91,24,46,8,33])
print("Original array: ")
print(B)
i = np.argsort(B)
print("Indices of the sorted elements of a given array: ")
print(i)
```

```
Original array:
[56 48 22 41 78 91 24 46  8 33]
Indices of the sorted elements of a given array:
[8 2 6 9 3 7 1 0 4 5]
```

b. 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 n x m array, n and m are user inputs given at the run time.

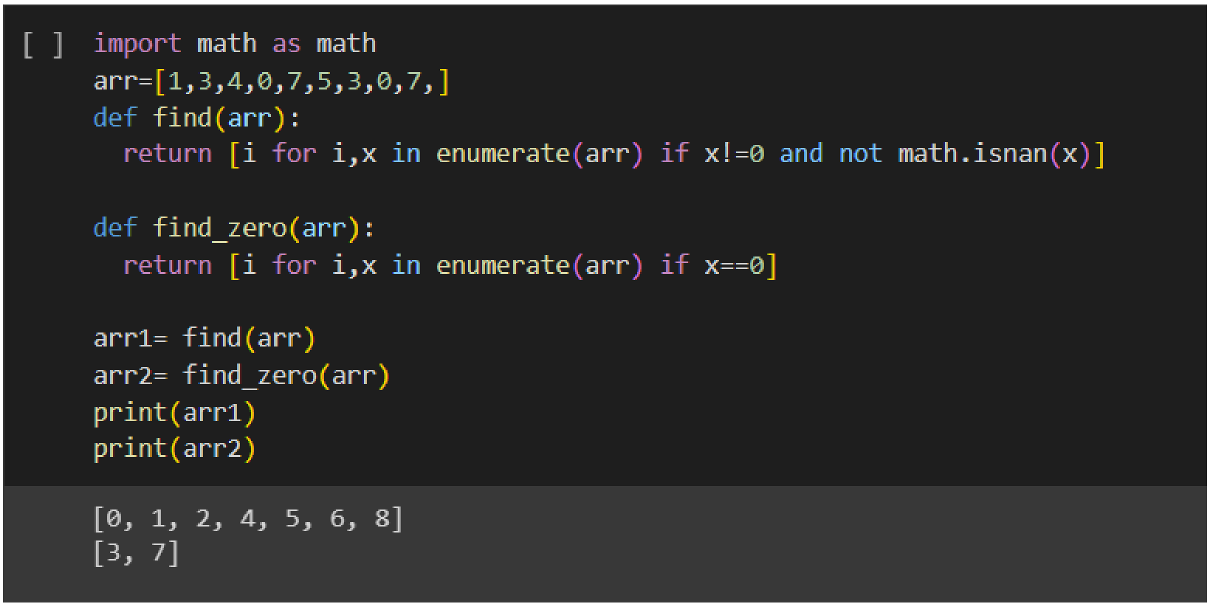


```
[ ] import numpy as np
    R = int(input("Enter the number of rows:"))
    C = int(input("Enter the number of columns:"))

    matrix = []
    print("Enter the entries rowwise:")

    for i in range(R):
      a = []
      for j in range(C):
        a.append(int(input()))
      matrix.append(a)

    for i in range(R):
      for j in range(C):
        print(matrix[i][j],end=" ")
      print()

    print(np.shape(matrix))
    print(type(matrix))
    newarray = np.transpose(matrix)
    print(newarray)
```



```
Enter the number of rows:3
Enter the number of columns:4
Enter the entries rowwise:
1
2
3
4
5
6
6
5
4
3
2
1
1 2 3 4
5 6 6 5
4 3 2 1
(3, 4)
<class 'list'>
[[1 5 4]
 [2 6 3]
 [3 6 2]
 [4 5 1]]
```

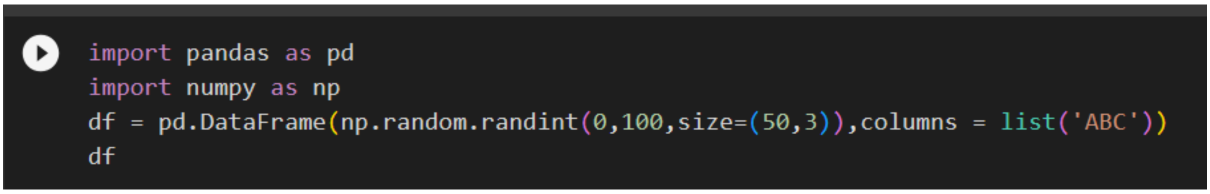
d. Test whether the elements of a given array are zero, non-zero and NaN. Record the indices of these elements in three separate arrays.



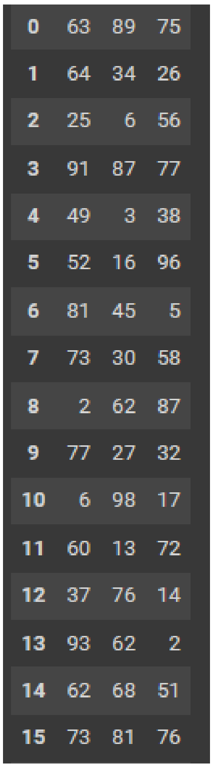
```
import math as math
arr=[1,3,4,0,7,5,3,0,7,]
def find(arr):
  return [i for i,x in enumerate(arr) if x!=0 and not math.isnan(x)]

def find_zero(arr):
  return [i for i,x in enumerate(arr) if x==0]

arr1= find(arr)
arr2= find_zero(arr)
print(arr1)
print(arr2)
```

```
[0, 1, 2, 4, 5, 6, 8]
[3, 7]
```

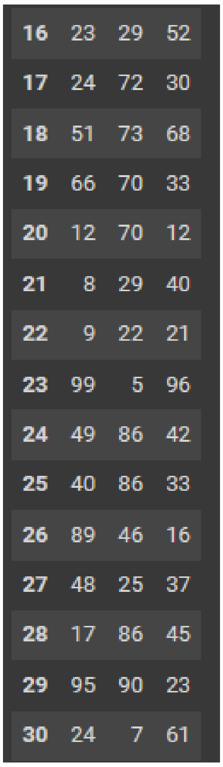
Ques3) Create a dataframe having at least 3 columns and 50 rows to store numeric data generated using a random function. Replace 10% of the values by null values whose index positions are generated using random function. Do the following:



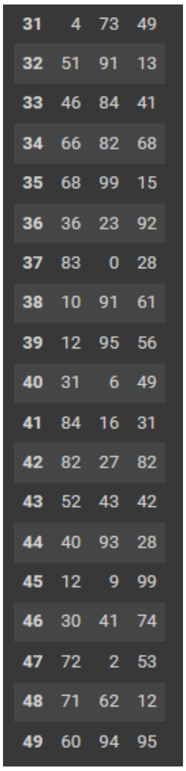
```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.randint(0,100,size=(50,3)),columns = list('ABC'))
df
```



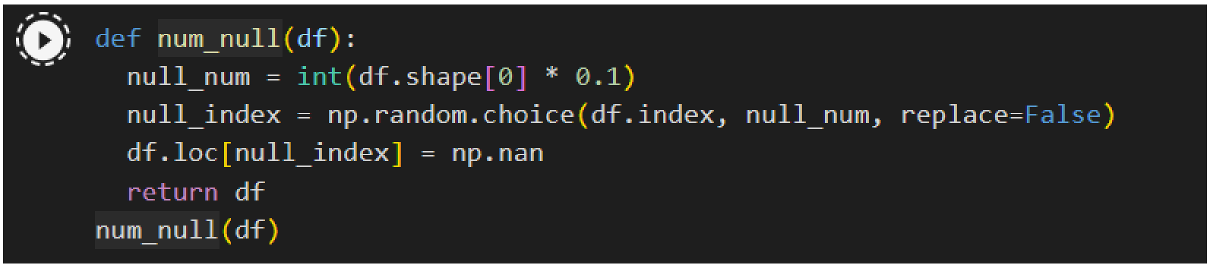
```
 0  63  89  75
 1  64  34  26
 2  25   6  56
 3  91  87  77
 4  49   3  38
 5  52  16  96
 6  81  45   5
 7  73  30  58
 8   2  62  87
 9  77  27  32
10   6  98  17
11  60  13  72
12  37  76  14
13  93  62   2
14  62  68  51
15  73  81  76
```



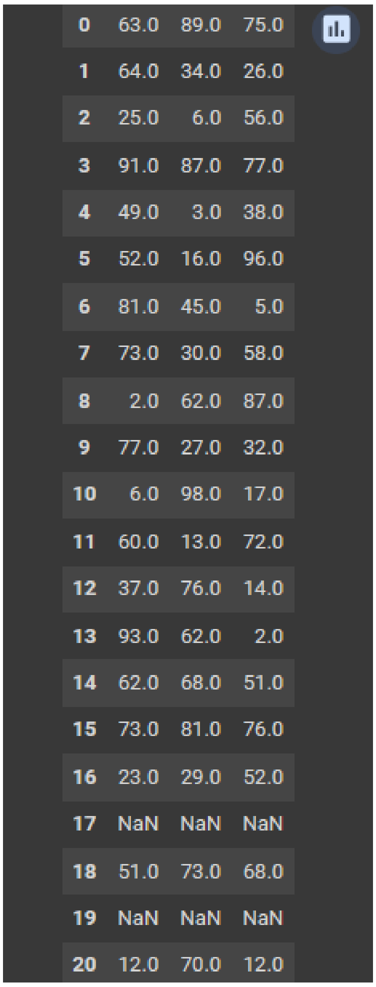
```
16  23  29  52
17  24  72  30
18  51  73  68
19  66  70  33
20  12  70  12
21   8  29  40
22   9  22  21
23  99   5  96
24  49  86  42
25  40  86  33
26  89  46  16
27  48  25  37
28  17  86  45
29  95  90  23
30  24   7  61
```



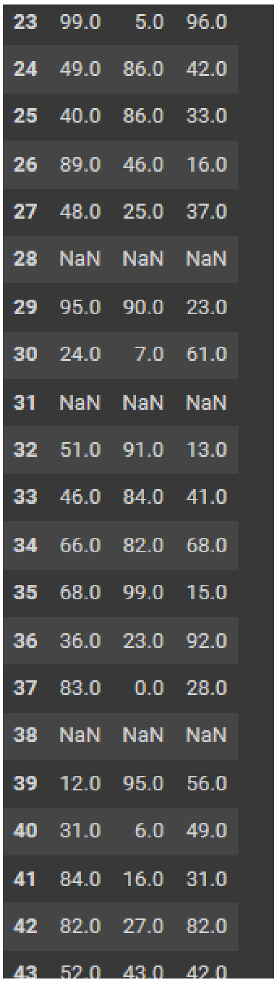
```
31   4  73  49
32  51  91  13
33  46  84  41
34  66  82  68
35  68  99  15
36  36  23  92
37  83   0  28
38  10  91  61
39  12  95  56
40  31   6  49
41  84  16  31
42  82  27  82
43  52  43  42
44  40  93  28
45  12   9  99
46  30  41  74
47  72   2  53
48  71  62  12
49  60  94  95
```



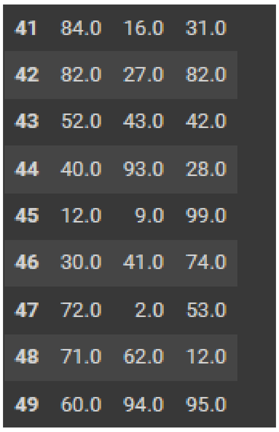
```
def num_null(df):
  null_num = int(df.shape[0] * 0.1)
  null_index = np.random.choice(df.index, null_num, replace=False)
  df.loc[null_index] = np.nan
  return df
num_null(df)
```



0	63.0	89.0	75.0
1	64.0	34.0	26.0
2	25.0	6.0	56.0
3	91.0	87.0	77.0
4	49.0	3.0	38.0
5	52.0	16.0	96.0
6	81.0	45.0	5.0
7	73.0	30.0	58.0
8	2.0	62.0	87.0
9	77.0	27.0	32.0
10	6.0	98.0	17.0
11	60.0	13.0	72.0
12	37.0	76.0	14.0
13	93.0	62.0	2.0
14	62.0	68.0	51.0
15	73.0	81.0	76.0
16	23.0	29.0	52.0
17	NaN	NaN	NaN
18	51.0	73.0	68.0
19	NaN	NaN	NaN
20	12.0	70.0	12.0

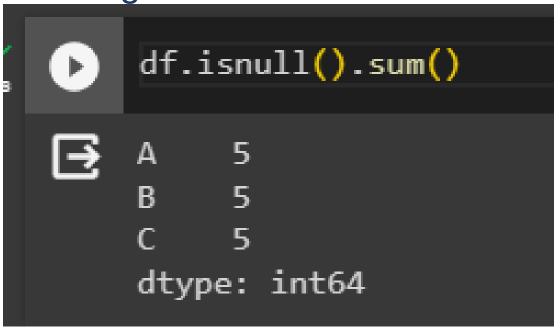


23	99.0	5.0	96.0
24	49.0	86.0	42.0
25	40.0	86.0	33.0
26	89.0	46.0	16.0
27	48.0	25.0	37.0
28	NaN	NaN	NaN
29	95.0	90.0	23.0
30	24.0	7.0	61.0
31	NaN	NaN	NaN
32	51.0	91.0	13.0
33	46.0	84.0	41.0
34	66.0	82.0	68.0
35	68.0	99.0	15.0
36	36.0	23.0	92.0
37	83.0	0.0	28.0
38	NaN	NaN	NaN
39	12.0	95.0	56.0
40	31.0	6.0	49.0
41	84.0	16.0	31.0
42	82.0	27.0	82.0
43	52.0	43.0	42.0



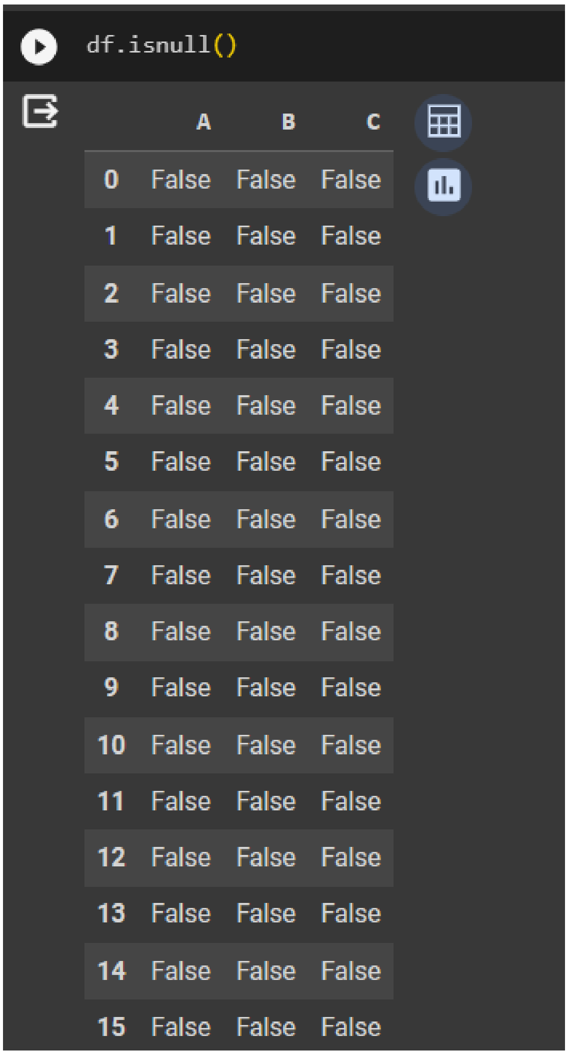
41	84.0	16.0	31.0
42	82.0	27.0	82.0
43	52.0	43.0	42.0
44	40.0	93.0	28.0
45	12.0	9.0	99.0
46	30.0	41.0	74.0
47	72.0	2.0	53.0
48	71.0	62.0	12.0
49	60.0	94.0	95.0

a.Identify and count missing values in a dataframe.



```
df.isnull().sum()
```

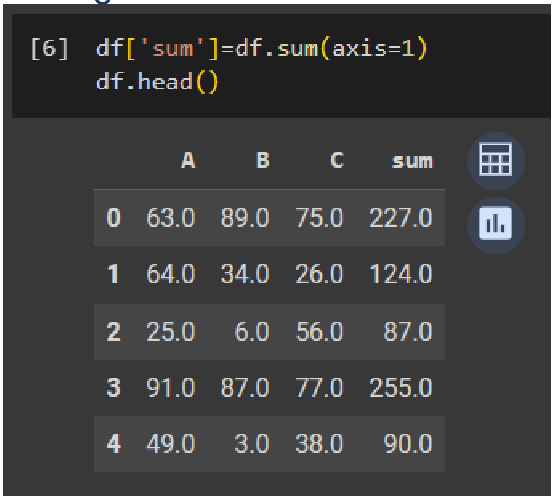
```
A    5
B    5
C    5
dtype: int64
```



```
df.isnull()
```

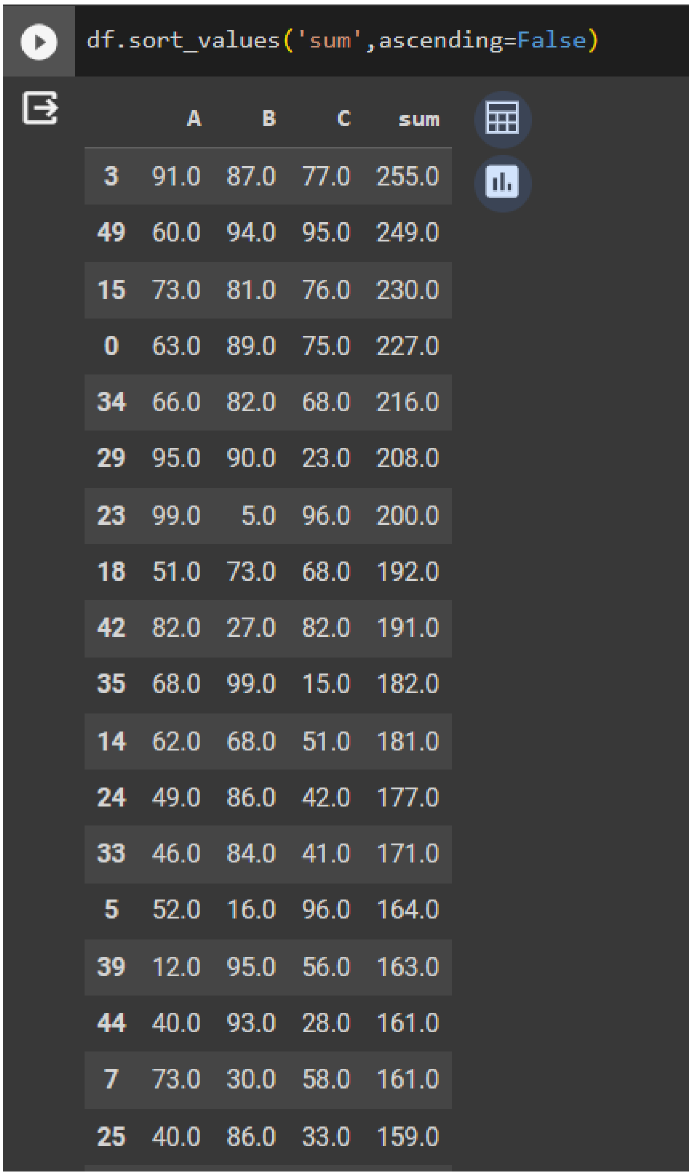
```
        A      B      C
 0  False  False  False
 1  False  False  False
 2  False  False  False
 3  False  False  False
 4  False  False  False
 5  False  False  False
 6  False  False  False
 7  False  False  False
 8  False  False  False
 9  False  False  False
10  False  False  False
11  False  False  False
12  False  False  False
13  False  False  False
14  False  False  False
15  False  False  False
```

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



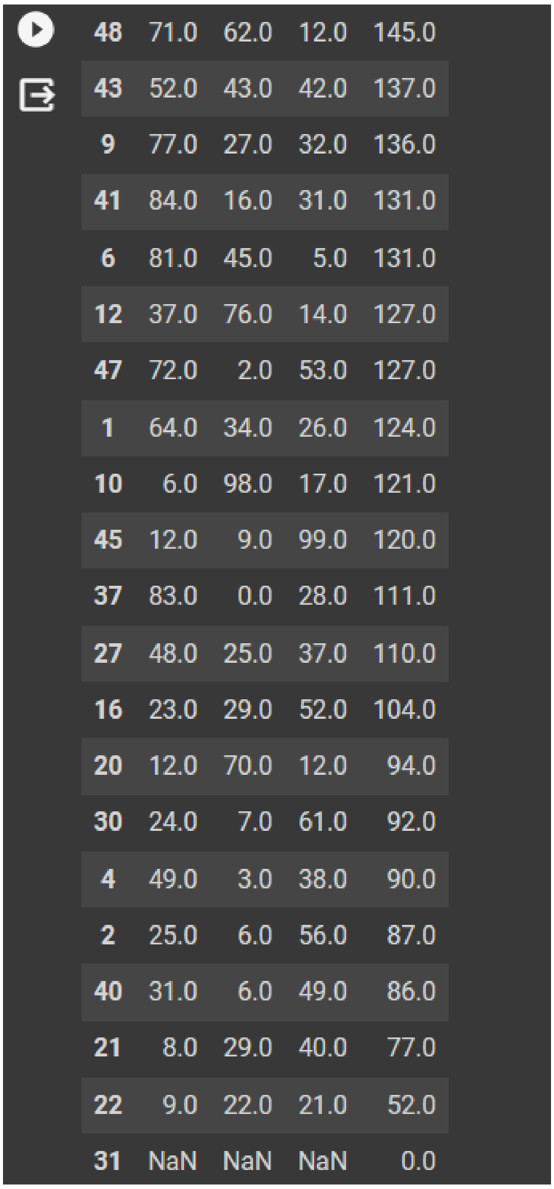
```
[6]  df['sum']=df.sum(axis=1)
     df.head()
```

```
      A     B     C    sum
0  63.0  89.0  75.0  227.0
1  64.0  34.0  26.0  124.0
2  25.0   6.0  56.0   87.0
3  91.0  87.0  77.0  255.0
4  49.0   3.0  38.0   90.0
```

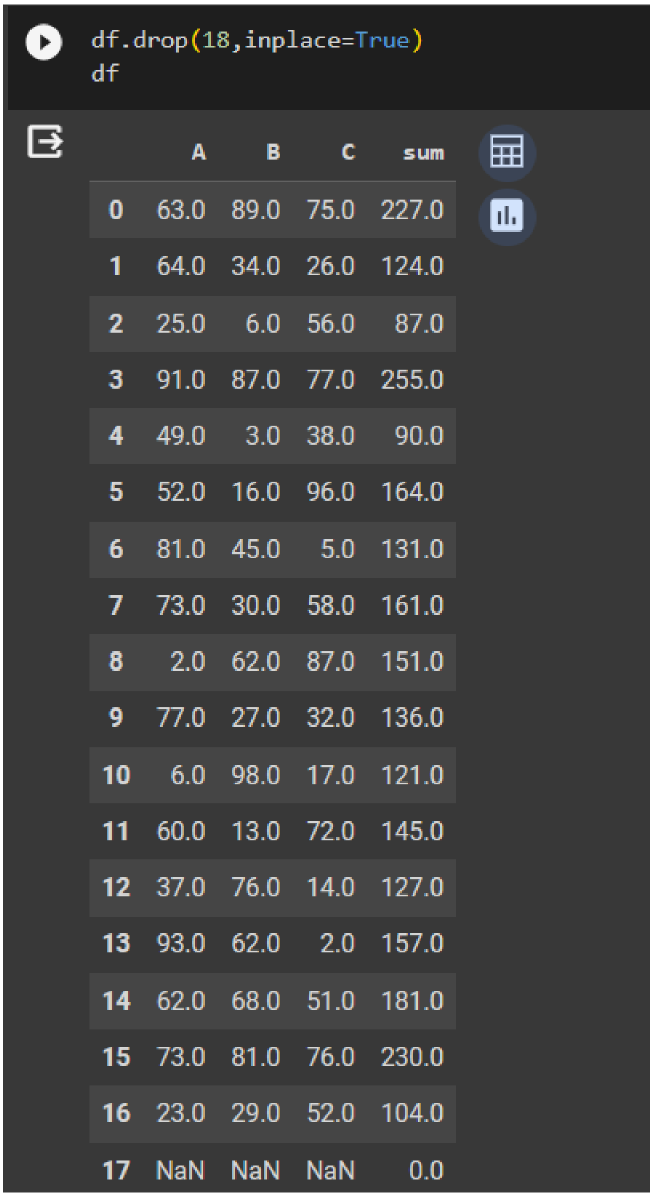


```
df.sort_values('sum',ascending=False)
```

```
       A     B     C    sum
3   91.0  87.0  77.0  255.0
49  60.0  94.0  95.0  249.0
15  73.0  81.0  76.0  230.0
0   63.0  89.0  75.0  227.0
34  66.0  82.0  68.0  216.0
29  95.0  90.0  23.0  208.0
23  99.0   5.0  96.0  200.0
18  51.0  73.0  68.0  192.0
42  82.0  27.0  82.0  191.0
35  68.0  99.0  15.0  182.0
14  62.0  68.0  51.0  181.0
24  49.0  86.0  42.0  177.0
33  46.0  84.0  41.0  171.0
5   52.0  16.0  96.0  164.0
39  12.0  95.0  56.0  163.0
44  40.0  93.0  28.0  161.0
7   73.0  30.0  58.0  161.0
25  40.0  86.0  33.0  159.0
```

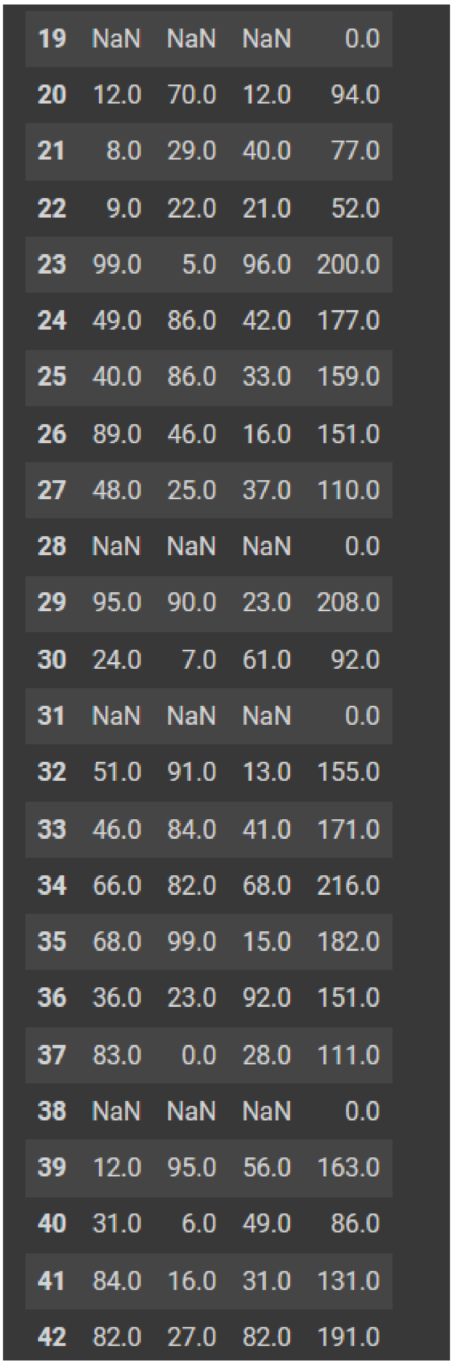


```
48  71.0  62.0  12.0  145.0
43  52.0  43.0  42.0  137.0
 9  77.0  27.0  32.0  136.0
41  84.0  16.0  31.0  131.0
 6  81.0  45.0   5.0  131.0
12  37.0  76.0  14.0  127.0
47  72.0   2.0  53.0  127.0
 1  64.0  34.0  26.0  124.0
10   6.0  98.0  17.0  121.0
45  12.0   9.0  99.0  120.0
37  83.0   0.0  28.0  111.0
27  48.0  25.0  37.0  110.0
16  23.0  29.0  52.0  104.0
20  12.0  70.0  12.0   94.0
30  24.0   7.0  61.0   92.0
 4  49.0   3.0  38.0   90.0
 2  25.0   6.0  56.0   87.0
40  31.0   6.0  49.0   86.0
21   8.0  29.0  40.0   77.0
22   9.0  22.0  21.0   52.0
31   NaN   NaN   NaN    0.0
```



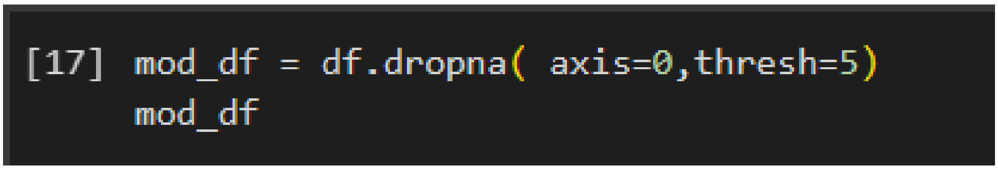
```
df.drop(18,inplace=True)
df
```

```
       A     B     C    sum
 0  63.0  89.0  75.0  227.0
 1  64.0  34.0  26.0  124.0
 2  25.0   6.0  56.0   87.0
 3  91.0  87.0  77.0  255.0
 4  49.0   3.0  38.0   90.0
 5  52.0  16.0  96.0  164.0
 6  81.0  45.0   5.0  131.0
 7  73.0  30.0  58.0  161.0
 8   2.0  62.0  87.0  151.0
 9  77.0  27.0  32.0  136.0
10   6.0  98.0  17.0  121.0
11  60.0  13.0  72.0  145.0
12  37.0  76.0  14.0  127.0
13  93.0  62.0   2.0  157.0
14  62.0  68.0  51.0  181.0
15  73.0  81.0  76.0  230.0
16  23.0  29.0  52.0  104.0
17   NaN   NaN   NaN    0.0
```

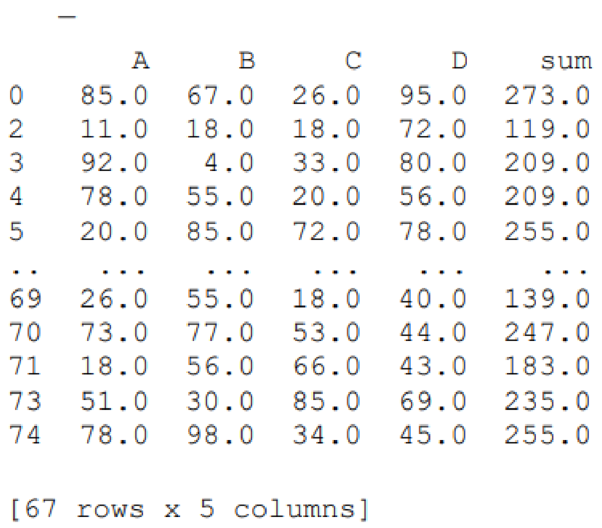


19	NaN	NaN	NaN	0.0
20	12.0	70.0	12.0	94.0
21	8.0	29.0	40.0	77.0
22	9.0	22.0	21.0	52.0
23	99.0	5.0	96.0	200.0
24	49.0	86.0	42.0	177.0
25	40.0	86.0	33.0	159.0
26	89.0	46.0	16.0	151.0
27	48.0	25.0	37.0	110.0
28	NaN	NaN	NaN	0.0
29	95.0	90.0	23.0	208.0
30	24.0	7.0	61.0	92.0
31	NaN	NaN	NaN	0.0
32	51.0	91.0	13.0	155.0
33	46.0	84.0	41.0	171.0
34	66.0	82.0	68.0	216.0
35	68.0	99.0	15.0	182.0
36	36.0	23.0	92.0	151.0
37	83.0	0.0	28.0	111.0
38	NaN	NaN	NaN	0.0
39	12.0	95.0	56.0	163.0
40	31.0	6.0	49.0	86.0
41	84.0	16.0	31.0	131.0
42	82.0	27.0	82.0	191.0

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



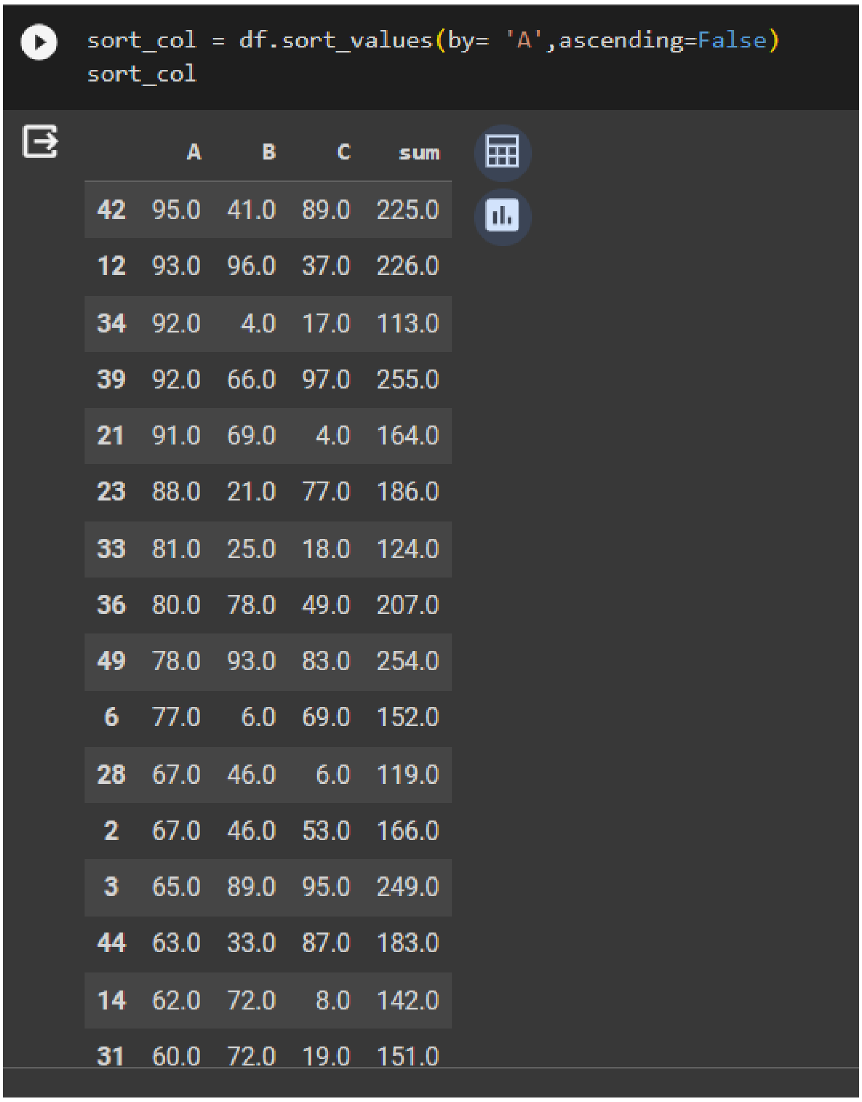
```
[17] mod_df = df.dropna( axis=0,thresh=5)
     mod_df
```



```
   _
        A     B     C     D    sum
0    85.0  67.0  26.0  95.0  273.0
2    11.0  18.0  18.0  72.0  119.0
3    92.0   4.0  33.0  80.0  209.0
4    78.0  55.0  20.0  56.0  209.0
5    20.0  85.0  72.0  78.0  255.0
..    ...   ...   ...   ...    ...
69   26.0  55.0  18.0  40.0  139.0
70   73.0  77.0  53.0  44.0  247.0
71   18.0  56.0  66.0  43.0  183.0
73   51.0  30.0  85.0  69.0  235.0
74   78.0  98.0  34.0  45.0  255.0

[67 rows x 5 columns]
```

c. Sort the dataframe on the basis of the first column.

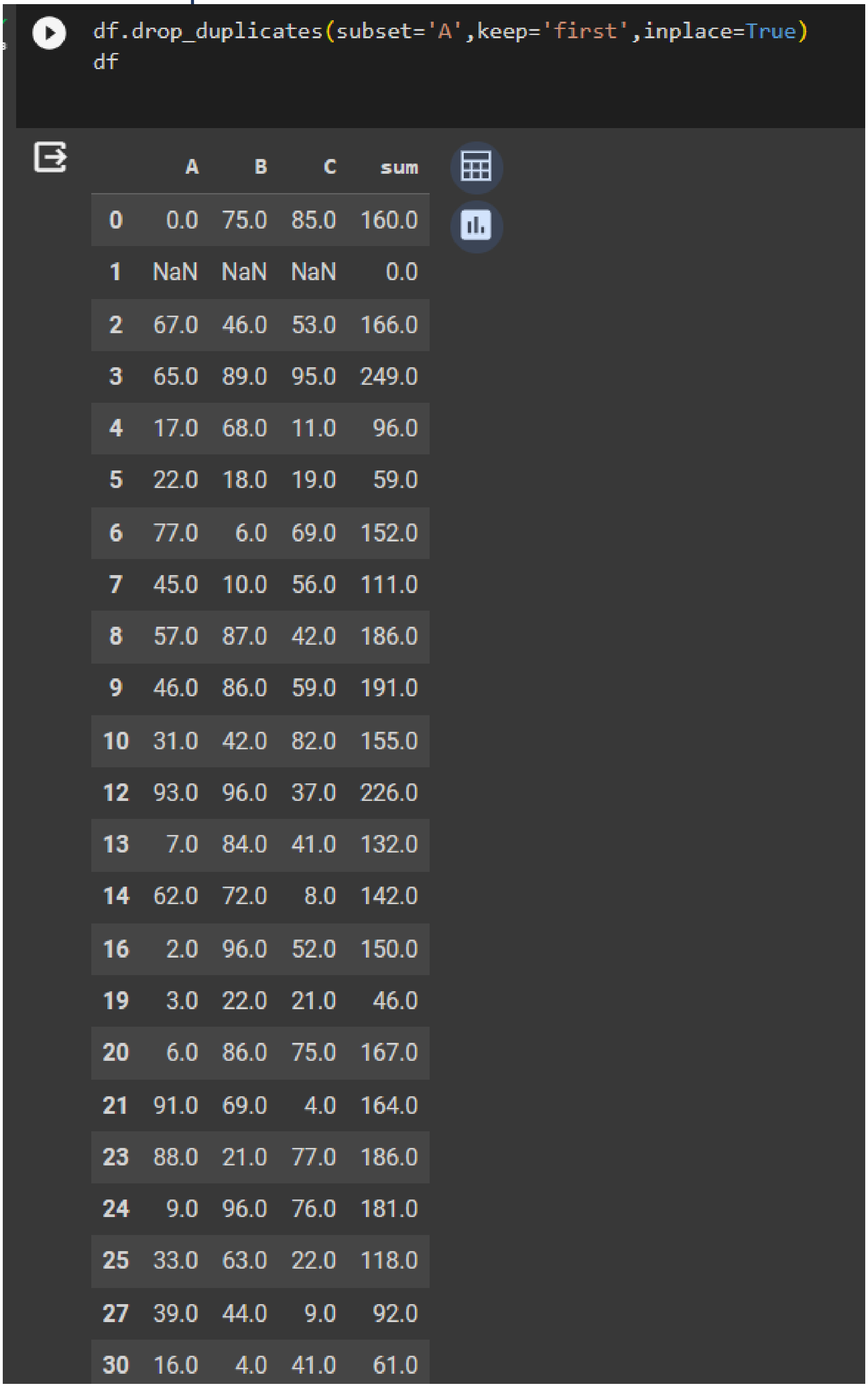
d.



```
sort_col = df.sort_values(by= 'A',ascending=False)
sort_col
```

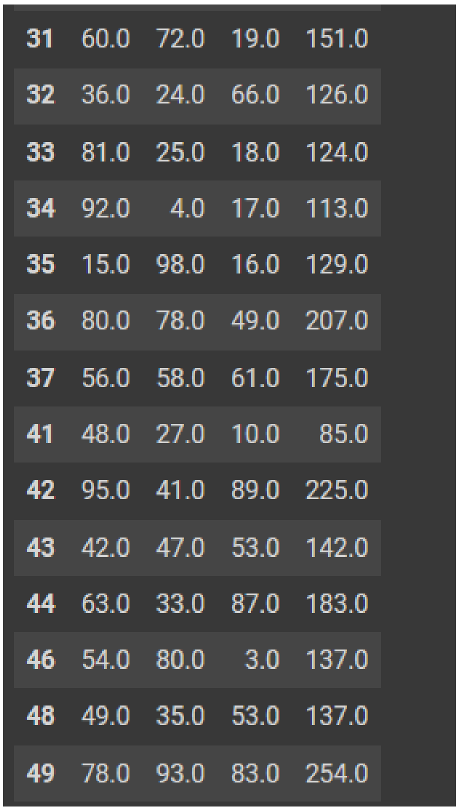
	A	B	C	sum
42	95.0	41.0	89.0	225.0
12	93.0	96.0	37.0	226.0
34	92.0	4.0	17.0	113.0
39	92.0	66.0	97.0	255.0
21	91.0	69.0	4.0	164.0
23	88.0	21.0	77.0	186.0
33	81.0	25.0	18.0	124.0
36	80.0	78.0	49.0	207.0
49	78.0	93.0	83.0	254.0
6	77.0	6.0	69.0	152.0
28	67.0	46.0	6.0	119.0
2	67.0	46.0	53.0	166.0
3	65.0	89.0	95.0	249.0
44	63.0	33.0	87.0	183.0
14	62.0	72.0	8.0	142.0
31	60.0	72.0	19.0	151.0

e. Remove all duplicates from the first column.



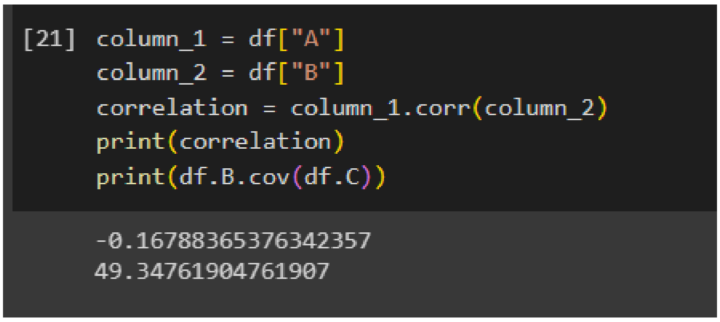
```
df.drop_duplicates(subset='A',keep='first',inplace=True)
df
```

```
       A     B     C    sum
0    0.0  75.0  85.0  160.0
1    NaN   NaN   NaN    0.0
2   67.0  46.0  53.0  166.0
3   65.0  89.0  95.0  249.0
4   17.0  68.0  11.0   96.0
5   22.0  18.0  19.0   59.0
6   77.0   6.0  69.0  152.0
7   45.0  10.0  56.0  111.0
8   57.0  87.0  42.0  186.0
9   46.0  86.0  59.0  191.0
10  31.0  42.0  82.0  155.0
12  93.0  96.0  37.0  226.0
13   7.0  84.0  41.0  132.0
14  62.0  72.0   8.0  142.0
16   2.0  96.0  52.0  150.0
19   3.0  22.0  21.0   46.0
20   6.0  86.0  75.0  167.0
21  91.0  69.0   4.0  164.0
23  88.0  21.0  77.0  186.0
24   9.0  96.0  76.0  181.0
25  33.0  63.0  22.0  118.0
27  39.0  44.0   9.0   92.0
30  16.0   4.0  41.0   61.0
```



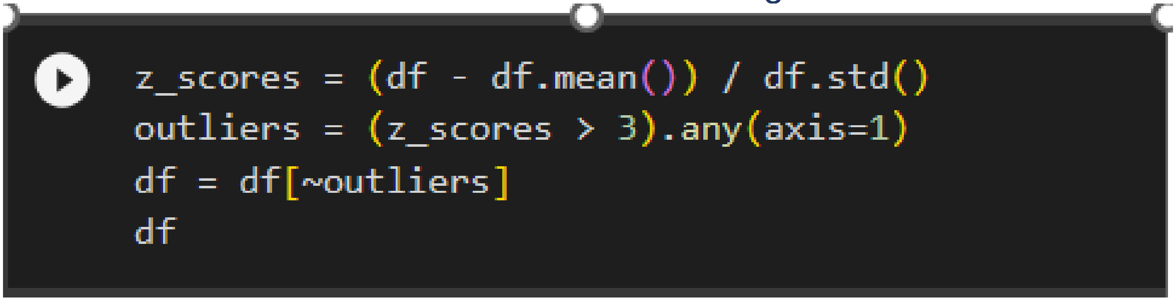
31	60.0	72.0	19.0	151.0
32	36.0	24.0	66.0	126.0
33	81.0	25.0	18.0	124.0
34	92.0	4.0	17.0	113.0
35	15.0	98.0	16.0	129.0
36	80.0	78.0	49.0	207.0
37	56.0	58.0	61.0	175.0
41	48.0	27.0	10.0	85.0
42	95.0	41.0	89.0	225.0
43	42.0	47.0	53.0	142.0
44	63.0	33.0	87.0	183.0
46	54.0	80.0	3.0	137.0
48	49.0	35.0	53.0	137.0
49	78.0	93.0	83.0	254.0

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

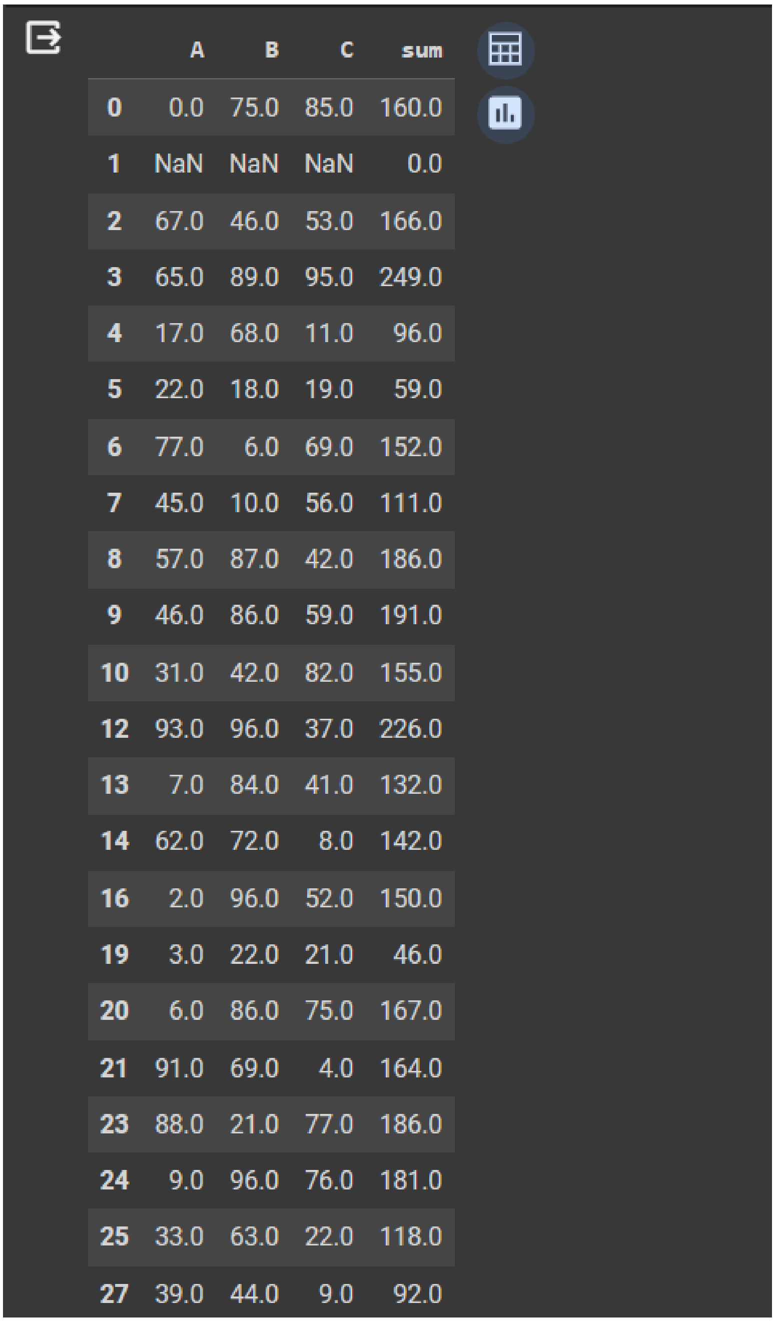


```
[21] column_1 = df["A"]
     column_2 = df["B"]
     correlation = column_1.corr(column_2)
     print(correlation)
     print(df.B.cov(df.C))

     -0.16788365376342357
     49.34761904761907
```

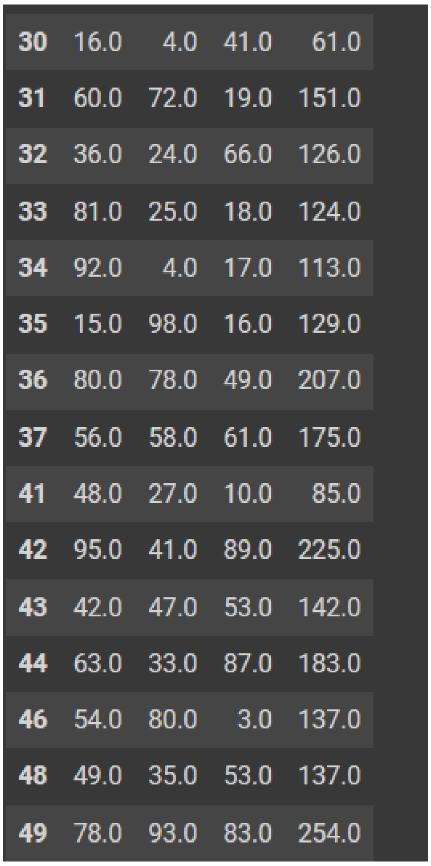
g. Detect the outliers and remove the rows having outliers.



```
z_scores = (df - df.mean()) / df.std()
outliers = (z_scores > 3).any(axis=1)
df = df[~outliers]
df
```

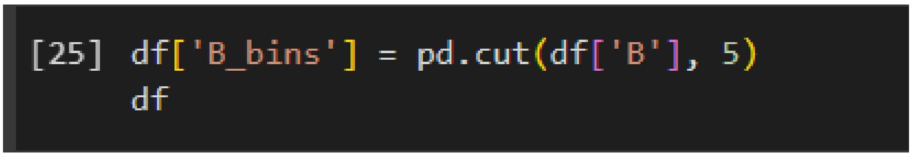


```
       A     B     C    sum
0    0.0  75.0  85.0  160.0
1    NaN   NaN   NaN    0.0
2   67.0  46.0  53.0  166.0
3   65.0  89.0  95.0  249.0
4   17.0  68.0  11.0   96.0
5   22.0  18.0  19.0   59.0
6   77.0   6.0  69.0  152.0
7   45.0  10.0  56.0  111.0
8   57.0  87.0  42.0  186.0
9   46.0  86.0  59.0  191.0
10  31.0  42.0  82.0  155.0
12  93.0  96.0  37.0  226.0
13   7.0  84.0  41.0  132.0
14  62.0  72.0   8.0  142.0
16   2.0  96.0  52.0  150.0
19   3.0  22.0  21.0   46.0
20   6.0  86.0  75.0  167.0
21  91.0  69.0   4.0  164.0
23  88.0  21.0  77.0  186.0
24   9.0  96.0  76.0  181.0
25  33.0  63.0  22.0  118.0
27  39.0  44.0   9.0   92.0
```

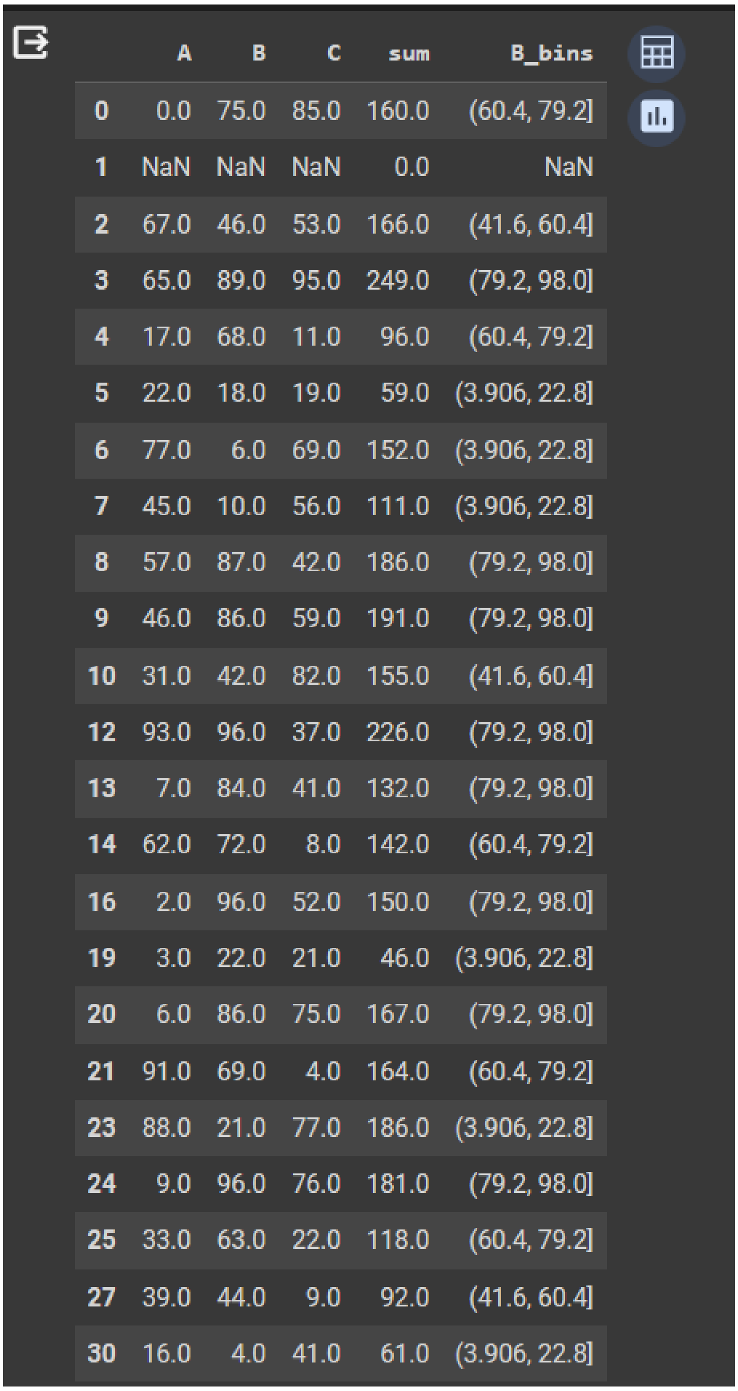


30	16.0	4.0	41.0	61.0
31	60.0	72.0	19.0	151.0
32	36.0	24.0	66.0	126.0
33	81.0	25.0	18.0	124.0
34	92.0	4.0	17.0	113.0
35	15.0	98.0	16.0	129.0
36	80.0	78.0	49.0	207.0
37	56.0	58.0	61.0	175.0
41	48.0	27.0	10.0	85.0
42	95.0	41.0	89.0	225.0
43	42.0	47.0	53.0	142.0
44	63.0	33.0	87.0	183.0
46	54.0	80.0	3.0	137.0
48	49.0	35.0	53.0	137.0
49	78.0	93.0	83.0	254.0

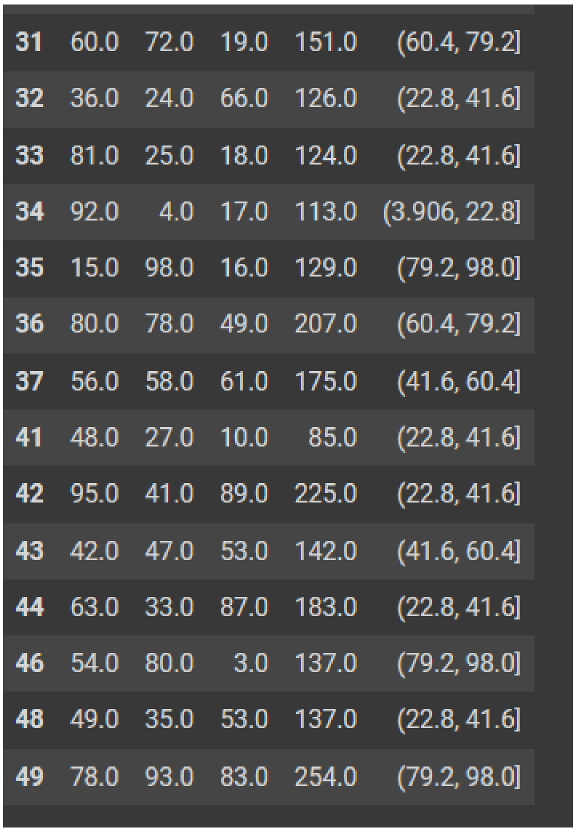
h. Discretize second column and create 5 bins.



```
[25] df['B_bins'] = pd.cut(df['B'], 5)
     df
```



	A	B	C	sum	B_bins
0	0.0	75.0	85.0	160.0	(60.4, 79.2]
1	NaN	NaN	NaN	0.0	NaN
2	67.0	46.0	53.0	166.0	(41.6, 60.4]
3	65.0	89.0	95.0	249.0	(79.2, 98.0]
4	17.0	68.0	11.0	96.0	(60.4, 79.2]
5	22.0	18.0	19.0	59.0	(3.906, 22.8]
6	77.0	6.0	69.0	152.0	(3.906, 22.8]
7	45.0	10.0	56.0	111.0	(3.906, 22.8]
8	57.0	87.0	42.0	186.0	(79.2, 98.0]
9	46.0	86.0	59.0	191.0	(79.2, 98.0]
10	31.0	42.0	82.0	155.0	(41.6, 60.4]
12	93.0	96.0	37.0	226.0	(79.2, 98.0]
13	7.0	84.0	41.0	132.0	(79.2, 98.0]
14	62.0	72.0	8.0	142.0	(60.4, 79.2]
16	2.0	96.0	52.0	150.0	(79.2, 98.0]
19	3.0	22.0	21.0	46.0	(3.906, 22.8]
20	6.0	86.0	75.0	167.0	(79.2, 98.0]
21	91.0	69.0	4.0	164.0	(60.4, 79.2]
23	88.0	21.0	77.0	186.0	(3.906, 22.8]
24	9.0	96.0	76.0	181.0	(79.2, 98.0]
25	33.0	63.0	22.0	118.0	(60.4, 79.2]
27	39.0	44.0	9.0	92.0	(41.6, 60.4]
30	16.0	4.0	41.0	61.0	(3.906, 22.8]



31	60.0	72.0	19.0	151.0	(60.4, 79.2]
32	36.0	24.0	66.0	126.0	(22.8, 41.6]
33	81.0	25.0	18.0	124.0	(22.8, 41.6]
34	92.0	4.0	17.0	113.0	(3.906, 22.8]
35	15.0	98.0	16.0	129.0	(79.2, 98.0]
36	80.0	78.0	49.0	207.0	(60.4, 79.2]
37	56.0	58.0	61.0	175.0	(41.6, 60.4]
41	48.0	27.0	10.0	85.0	(22.8, 41.6]
42	95.0	41.0	89.0	225.0	(22.8, 41.6]
43	42.0	47.0	53.0	142.0	(41.6, 60.4]
44	63.0	33.0	87.0	183.0	(22.8, 41.6]
46	54.0	80.0	3.0	137.0	(79.2, 98.0]
48	49.0	35.0	53.0	137.0	(22.8, 41.6]
49	78.0	93.0	83.0	254.0	(79.2, 98.0]

Ques4) Consider two excel files having attendance of a workshop's participants for two days. Each file has three fields 'Name', 'Time of joining', 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 dataframes and do the following:

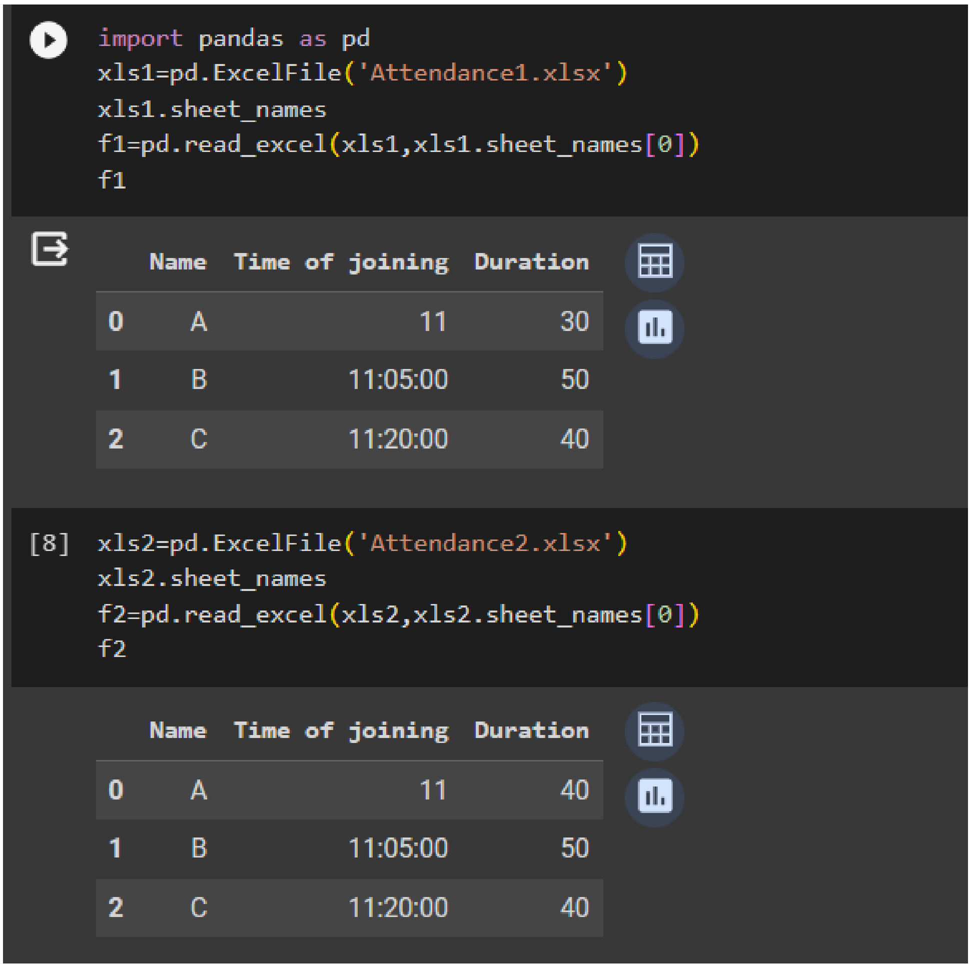
Create Excel files from the dataframes in your provided code:

a. Perform merging of the two dataframes to find the names of students who had attended the workshop on both days.

b. Find names of all students who have attended workshop on either of the days.

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

d. Merge two data frames and use two columns names and duration as multi-row indexes. Generate descriptive statistics for this multi-index.

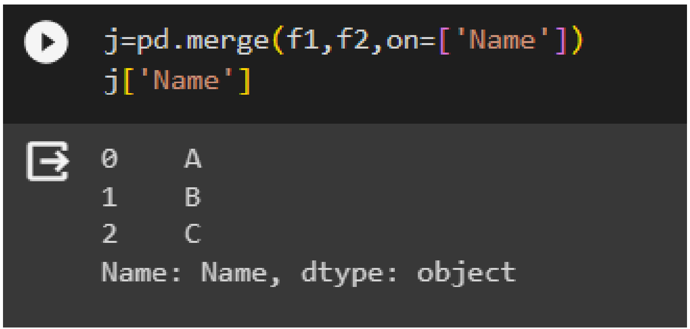


```
import pandas as pd
xls1=pd.ExcelFile('Attendance1.xlsx')
xls1.sheet_names
f1=pd.read_excel(xls1,xls1.sheet_names[0])
f1
```

	Name	Time of joining	Duration
0	A	11	30
1	B	11:05:00	50
2	C	11:20:00	40

```
[8] xls2=pd.ExcelFile('Attendance2.xlsx')
xls2.sheet_names
f2=pd.read_excel(xls2,xls2.sheet_names[0])
f2
```

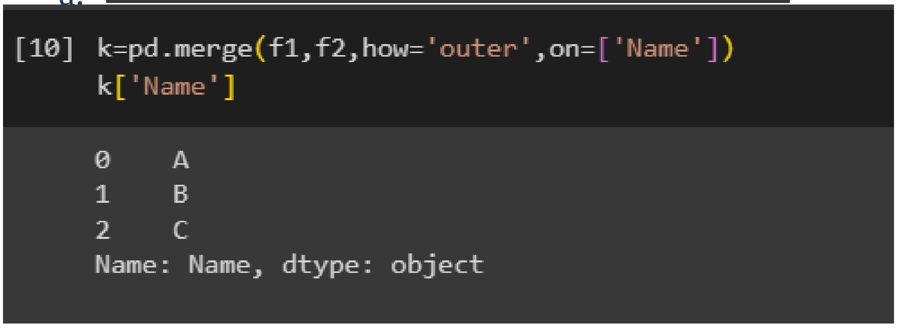
	Name	Time of joining	Duration
0	A	11	40
1	B	11:05:00	50
2	C	11:20:00	40



```
j=pd.merge(f1,f2,on=['Name'])
j['Name']
```

```
0    A
1    B
2    C
Name: Name, dtype: object
```

a.

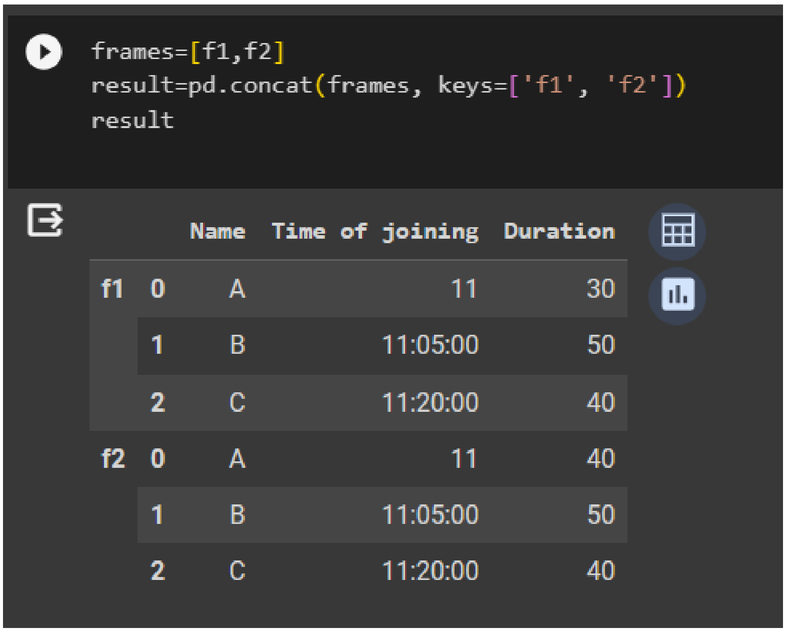


```
[10] k=pd.merge(f1,f2,how='outer',on=['Name'])
k['Name']
```

```
0    A
1    B
2    C
Name: Name, dtype: object
```

b.

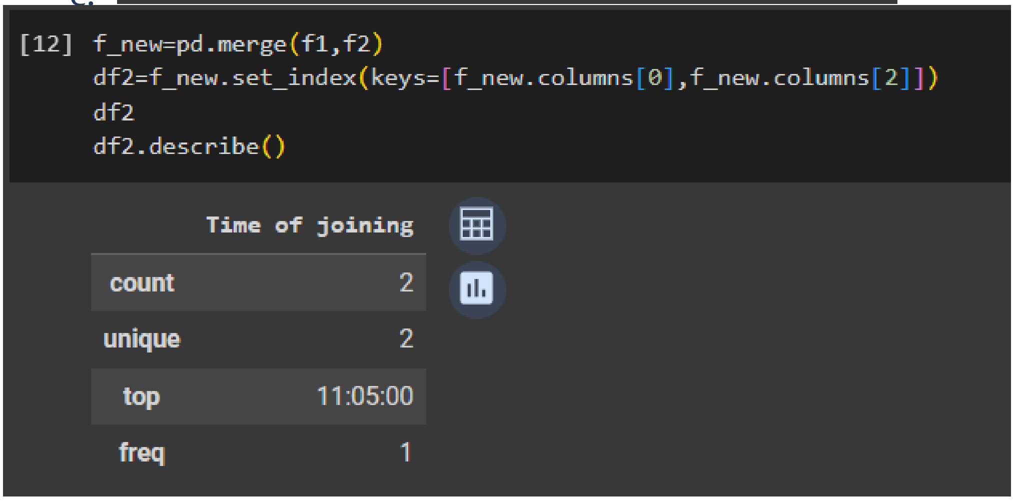
c.



```
frames=[f1,f2]
result=pd.concat(frames, keys=['f1', 'f2'])
result
```

```
             Name  Time of joining  Duration
f1  0        A               11        30
    1        B         11:05:00        50
    2        C         11:20:00        40
f2  0        A               11        40
    1        B         11:05:00        50
    2        C         11:20:00        40
```

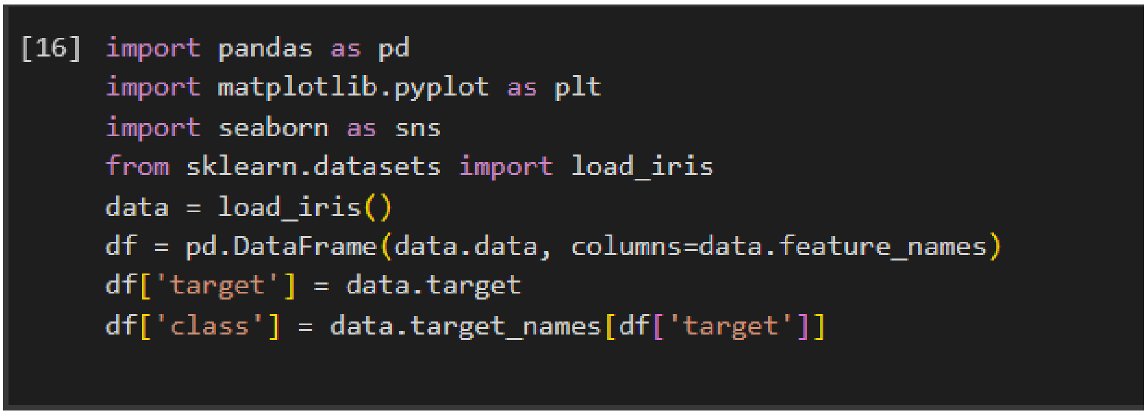
d.



```
[12] f_new=pd.merge(f1,f2)
     df2=f_new.set_index(keys=[f_new.columns[0],f_new.columns[2]])
     df2
     df2.describe()
```

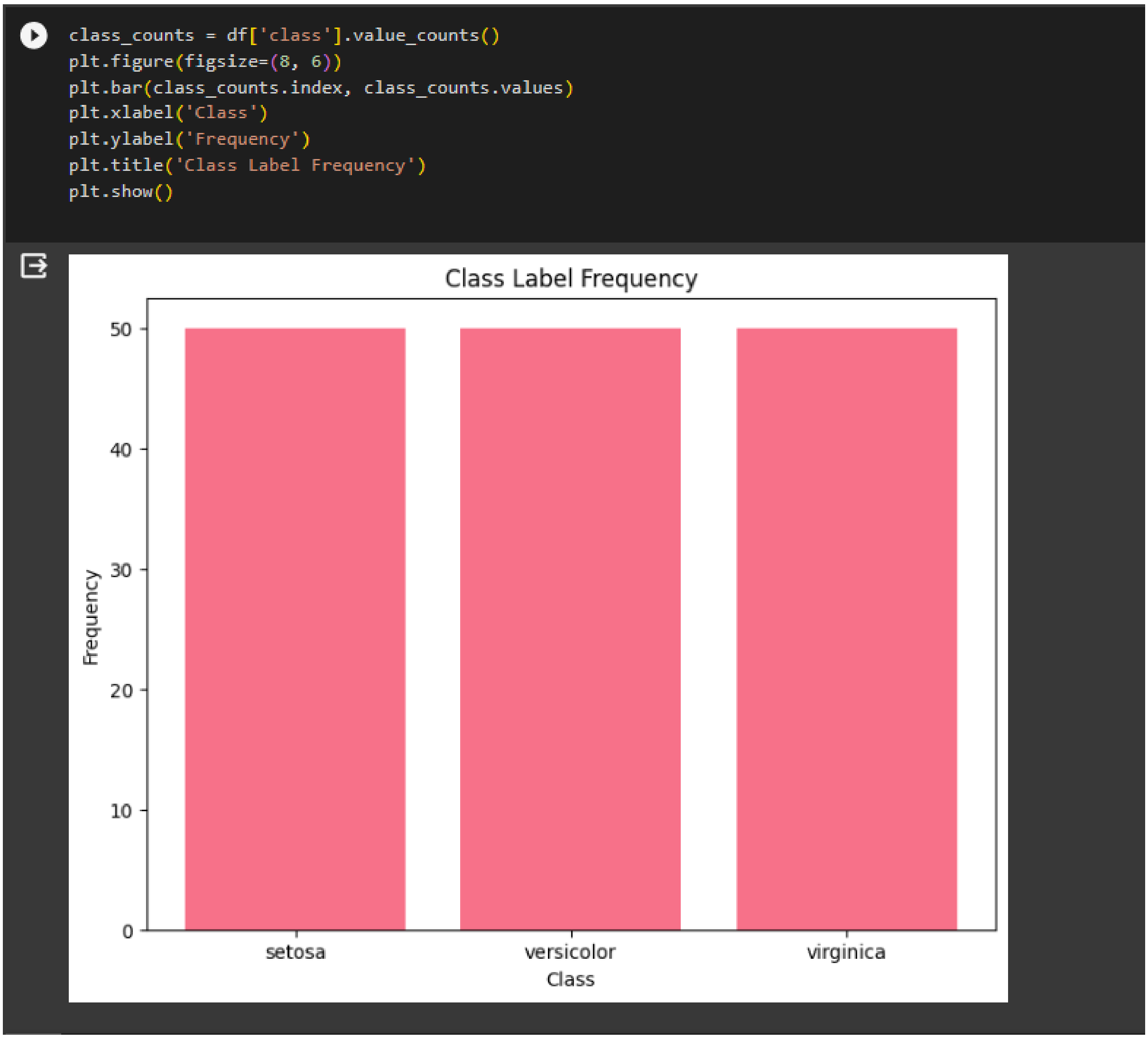
```
        Time of joining
count                 2
unique                2
top            11:05:00
freq                  1
```

Ques5) Taking 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).



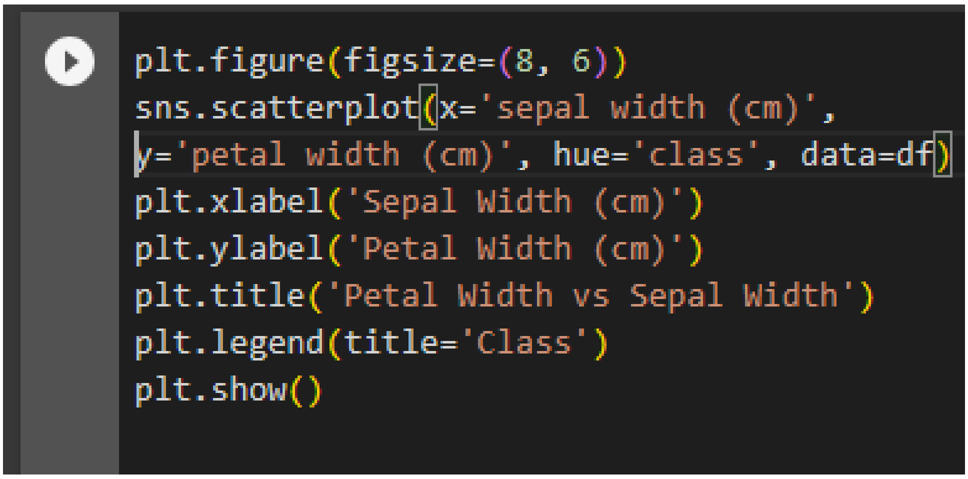
```
[16] import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
     from sklearn.datasets import load_iris
     data = load_iris()
     df = pd.DataFrame(data.data, columns=data.feature_names)
     df['target'] = data.target
     df['class'] = data.target_names[df['target']]
```

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

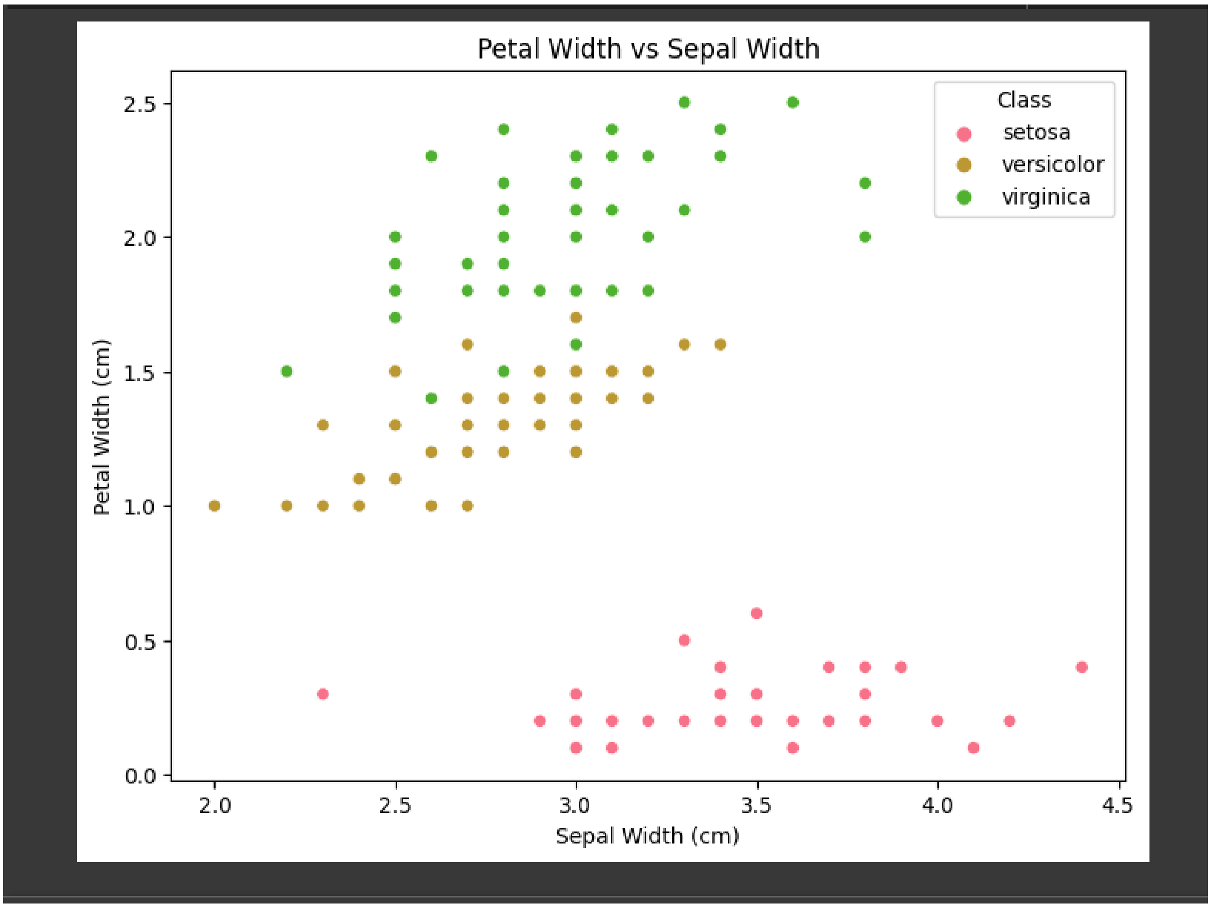


```
class_counts = df['class'].value_counts()
plt.figure(figsize=(8, 6))
plt.bar(class_counts.index, class_counts.values)
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.title('Class Label Frequency')
plt.show()
```

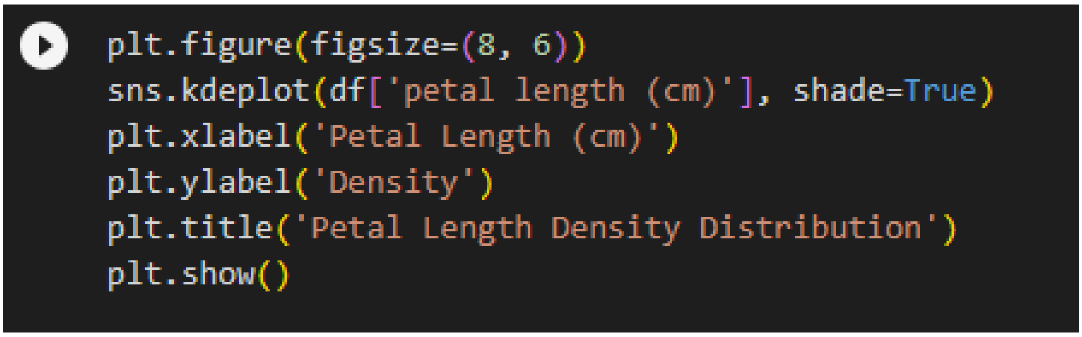
b. Draw a scatter plot for Petal width vs sepal width.



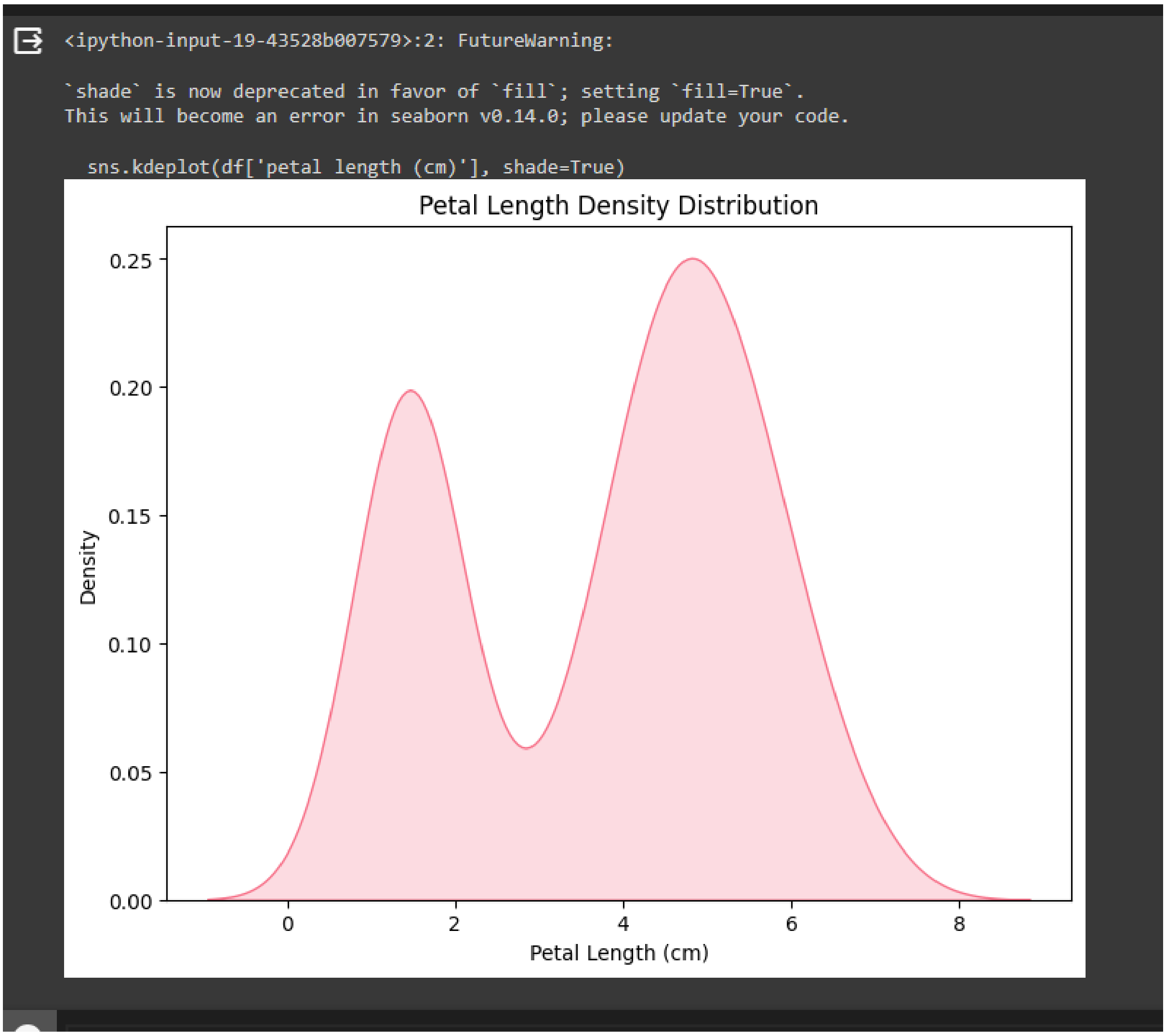
```
plt.figure(figsize=(8, 6))
sns.scatterplot(x='sepal width (cm)',
y='petal width (cm)', hue='class', data=df)
plt.xlabel('Sepal Width (cm)')
plt.ylabel('Petal Width (cm)')
plt.title('Petal Width vs Sepal Width')
plt.legend(title='Class')
plt.show()
```



c. Plot density distribution for feature petal length.

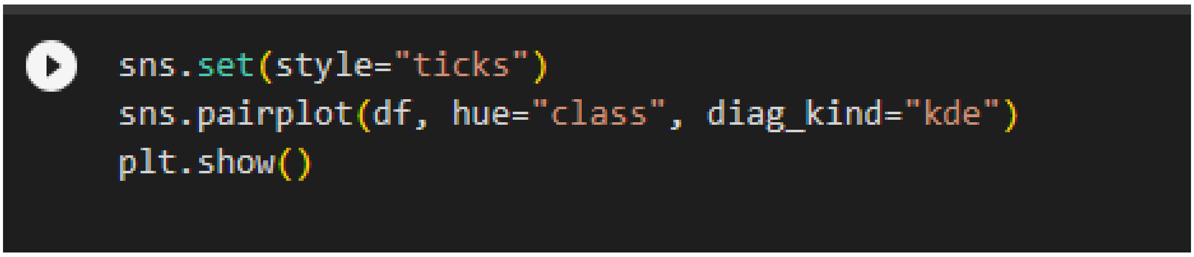


```
plt.figure(figsize=(8, 6))
sns.kdeplot(df['petal length (cm)'], shade=True)
plt.xlabel('Petal Length (cm)')
plt.ylabel('Density')
plt.title('Petal Length Density Distribution')
plt.show()
```

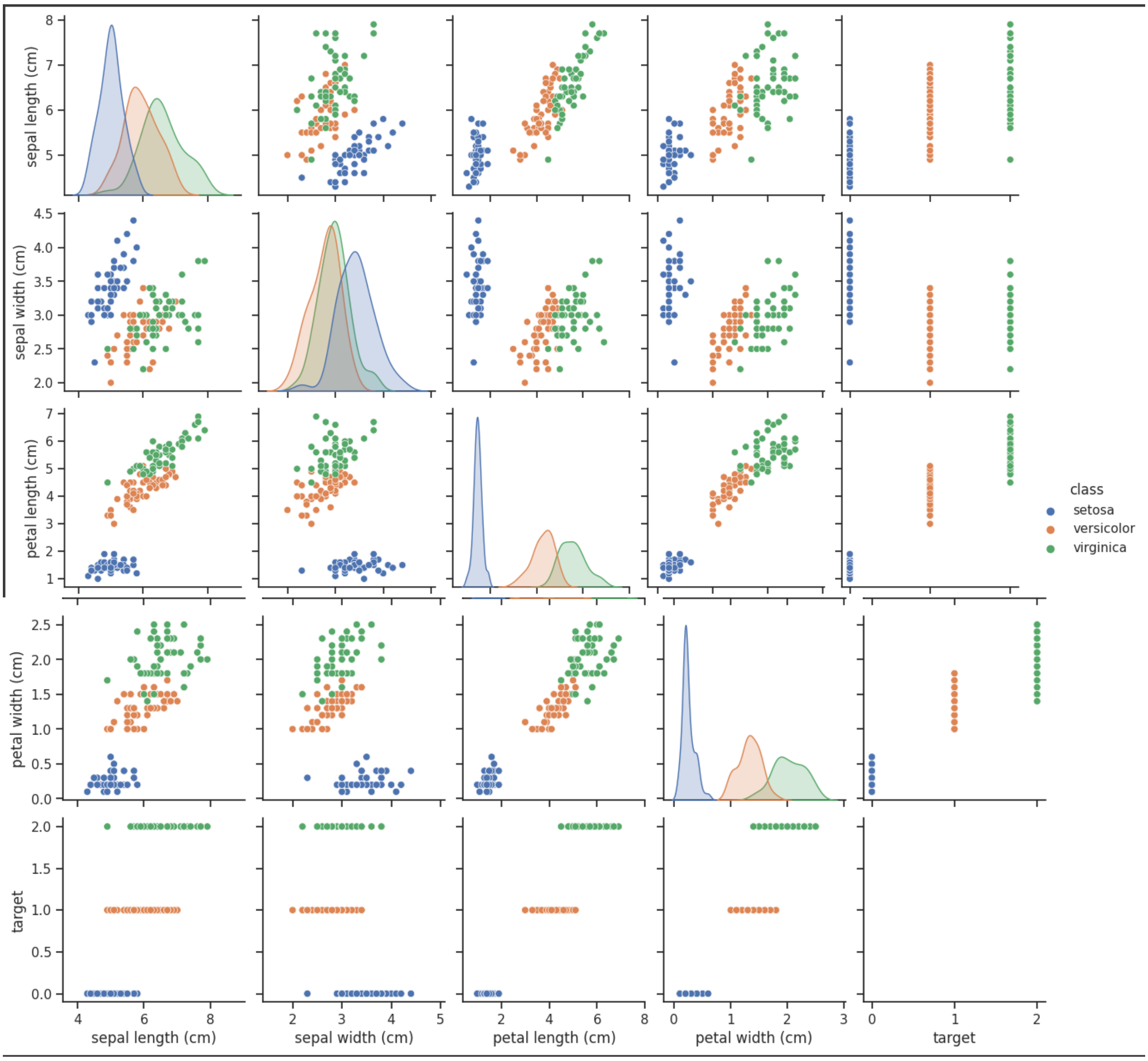


```
<ipython-input-19-43528b007579>:2: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(df['petal length (cm)'], shade=True)
```

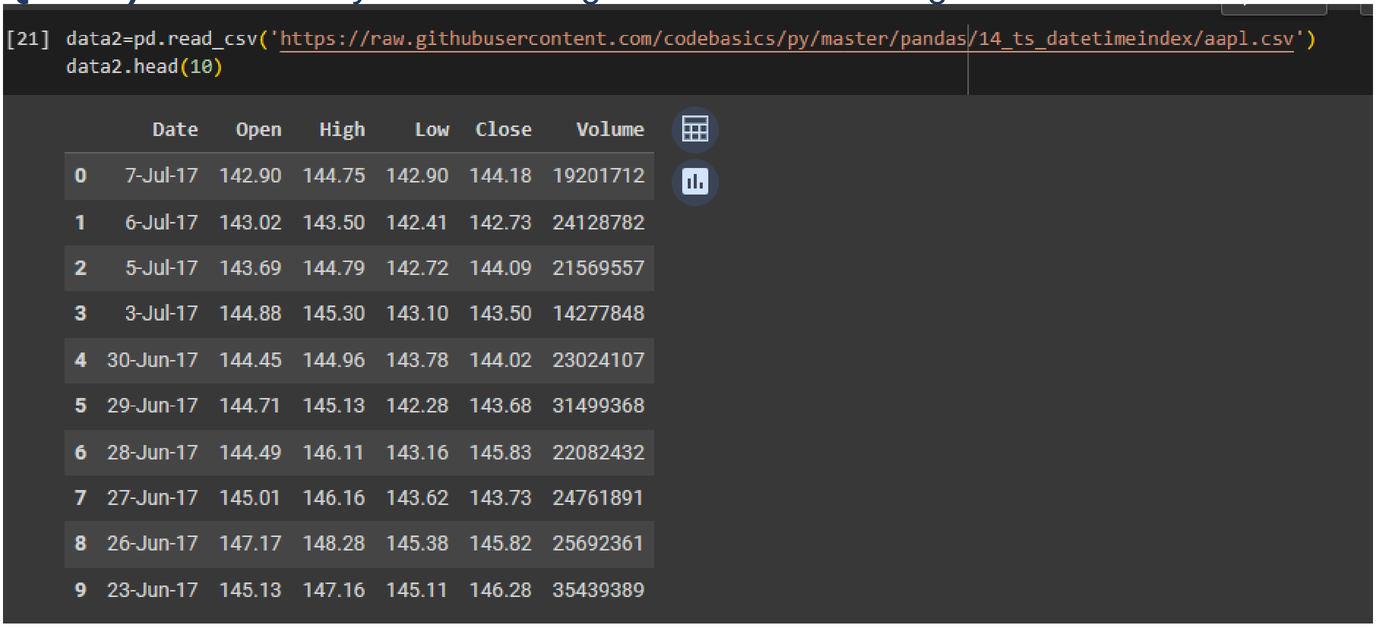
d. Use a pair plot to show pairwise bivariate distribution in the Iris Dataset.



```
sns.set(style="ticks")
sns.pairplot(df, hue="class", diag_kind="kde")
plt.show()
```



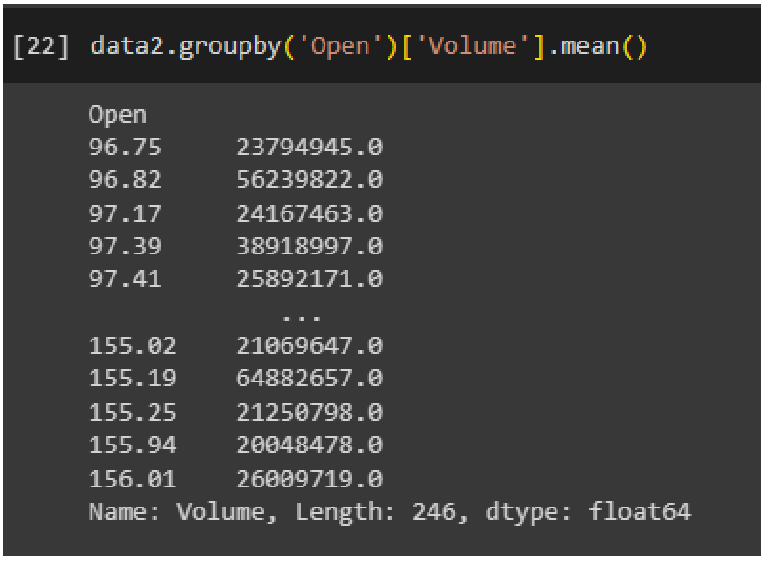
Ques 6) Consider any sales training/ weather forecasting dataset



```
[21] data2=pd.read_csv('https://raw.githubusercontent.com/codebasics/py/master/pandas/14_ts_datetimeindex/aapl.csv')
     data2.head(10)
```

	Date	Open	High	Low	Close	Volume
0	7-Jul-17	142.90	144.75	142.90	144.18	19201712
1	6-Jul-17	143.02	143.50	142.41	142.73	24128782
2	5-Jul-17	143.69	144.79	142.72	144.09	21569557
3	3-Jul-17	144.88	145.30	143.10	143.50	14277848
4	30-Jun-17	144.45	144.96	143.78	144.02	23024107
5	29-Jun-17	144.71	145.13	142.28	143.68	31499368
6	28-Jun-17	144.49	146.11	143.16	145.83	22082432
7	27-Jun-17	145.01	146.16	143.62	143.73	24761891
8	26-Jun-17	147.17	148.28	145.38	145.82	25692361
9	23-Jun-17	145.13	147.16	145.11	146.28	35439389

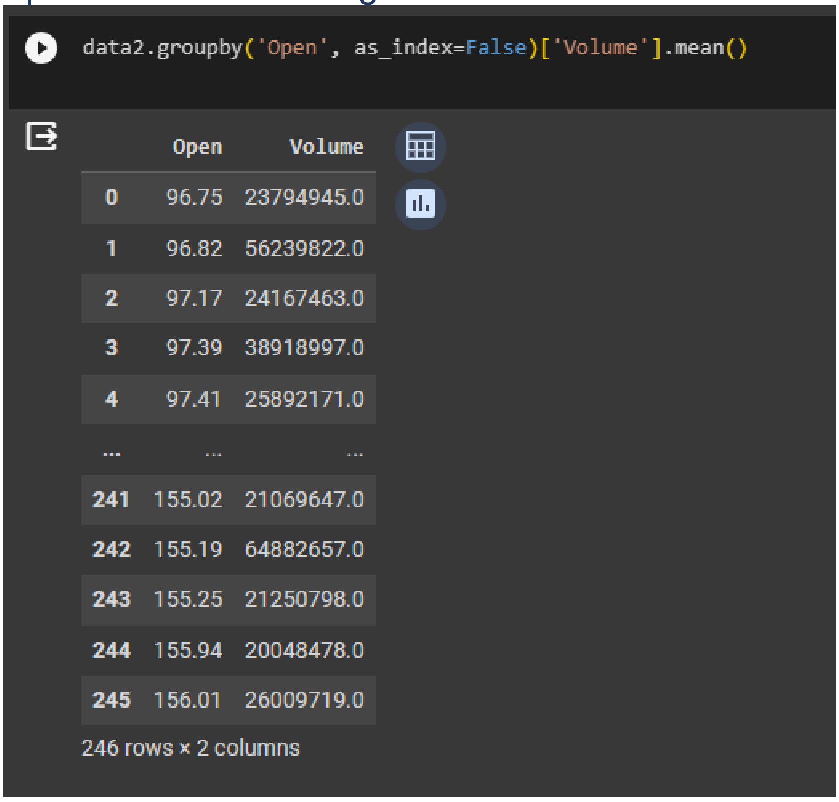
a. Compute mean of a series grouped by another series



```
[22] data2.groupby('Open')['Volume'].mean()
```

```
Open
96.75     23794945.0
96.82     56239822.0
97.17     24167463.0
97.39     38918997.0
97.41     25892171.0
             ...
155.02    21069647.0
155.19    64882657.0
155.25    21250798.0
155.94    20048478.0
156.01    26009719.0
Name: Volume, Length: 246, dtype: float64
```

b. Fill an intermittent time series to replace all missing dates with values of previous non-missing date.

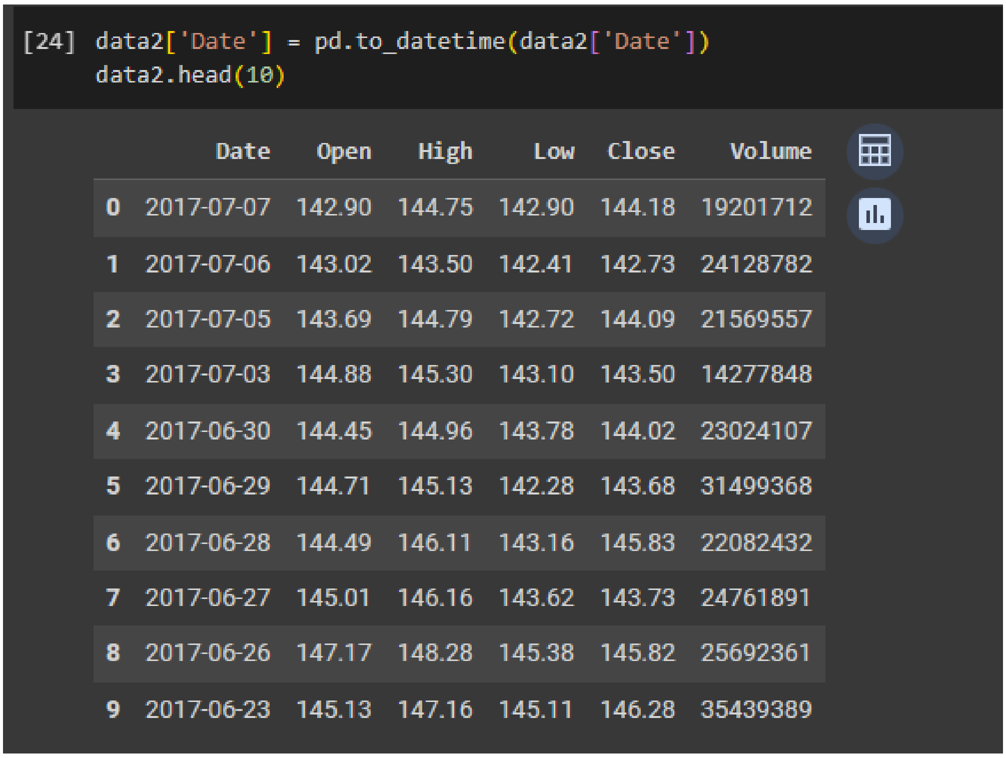


```
data2.groupby('Open', as_index=False)['Volume'].mean()
```

	Open	Volume
0	96.75	23794945.0
1	96.82	56239822.0
2	97.17	24167463.0
3	97.39	38918997.0
4	97.41	25892171.0
...	...	...
241	155.02	21069647.0
242	155.19	64882657.0
243	155.25	21250798.0
244	155.94	20048478.0
245	156.01	26009719.0

246 rows × 2 columns

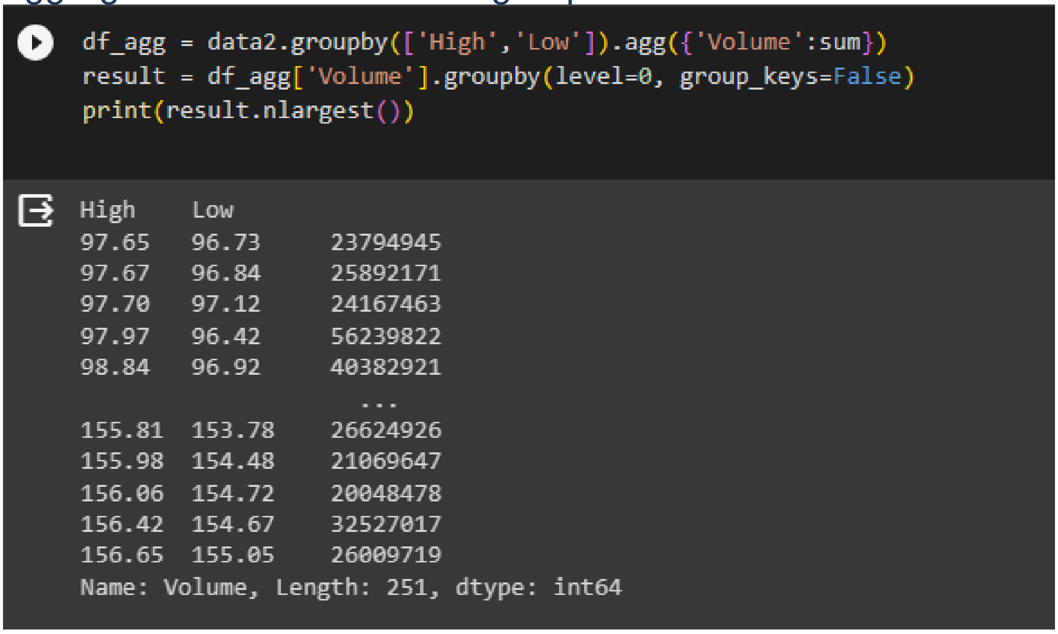
c. Perform appropriate year-month string to dates conversion.



```
[24] data2['Date'] = pd.to_datetime(data2['Date'])
     data2.head(10)
```

	Date	Open	High	Low	Close	Volume
0	2017-07-07	142.90	144.75	142.90	144.18	19201712
1	2017-07-06	143.02	143.50	142.41	142.73	24128782
2	2017-07-05	143.69	144.79	142.72	144.09	21569557
3	2017-07-03	144.88	145.30	143.10	143.50	14277848
4	2017-06-30	144.45	144.96	143.78	144.02	23024107
5	2017-06-29	144.71	145.13	142.28	143.68	31499368
6	2017-06-28	144.49	146.11	143.16	145.83	22082432
7	2017-06-27	145.01	146.16	143.62	143.73	24761891
8	2017-06-26	147.17	148.28	145.38	145.82	25692361
9	2017-06-23	145.13	147.16	145.11	146.28	35439389

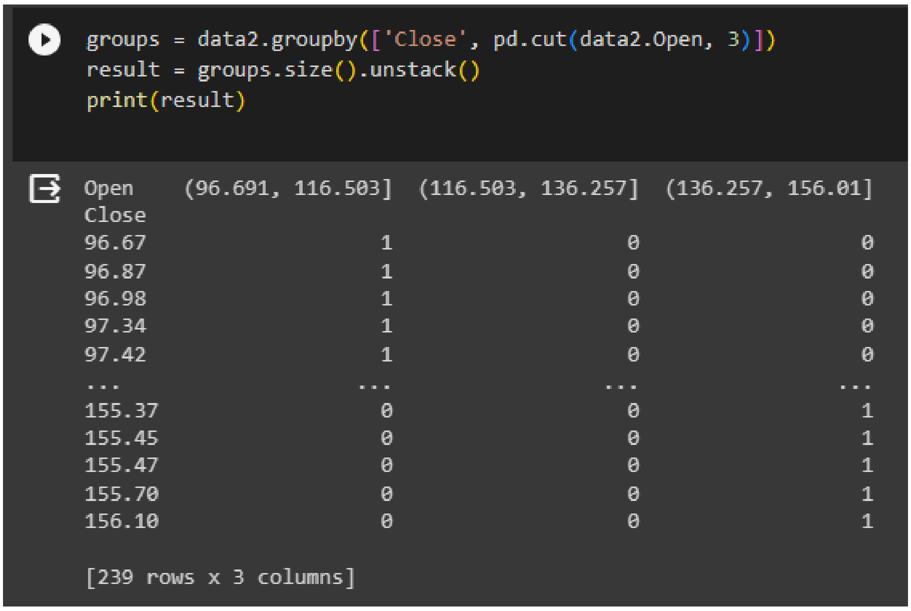
d. Split a dataset to group by two columns and then sort the aggregated results within the groups.



```
df_agg = data2.groupby(['High','Low']).agg({'Volume':sum})
result = df_agg['Volume'].groupby(level=0, group_keys=False)
print(result.nlargest())
```

```
High    Low
97.65   96.73     23794945
97.67   96.84     25892171
97.70   97.12     24167463
97.97   96.42     56239822
98.84   96.92     40382921
                    ...
155.81  153.78    26624926
155.98  154.48    21069647
156.06  154.72    20048478
156.42  154.67    32527017
156.65  155.05    26009719
Name: Volume, Length: 251, dtype: int64
```

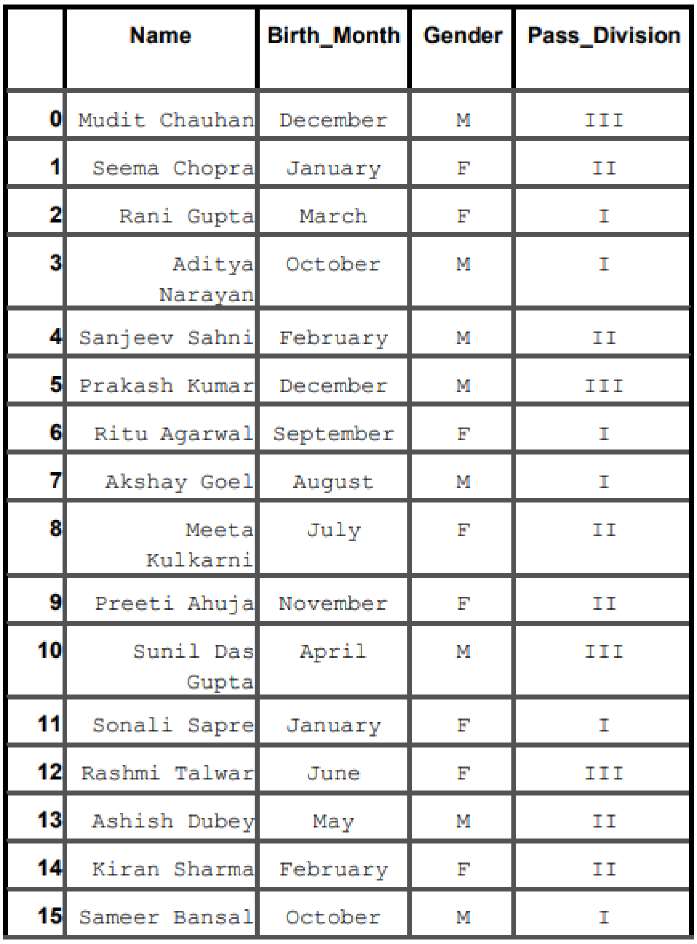
e. Split a given dataframe into groups with bin counts.



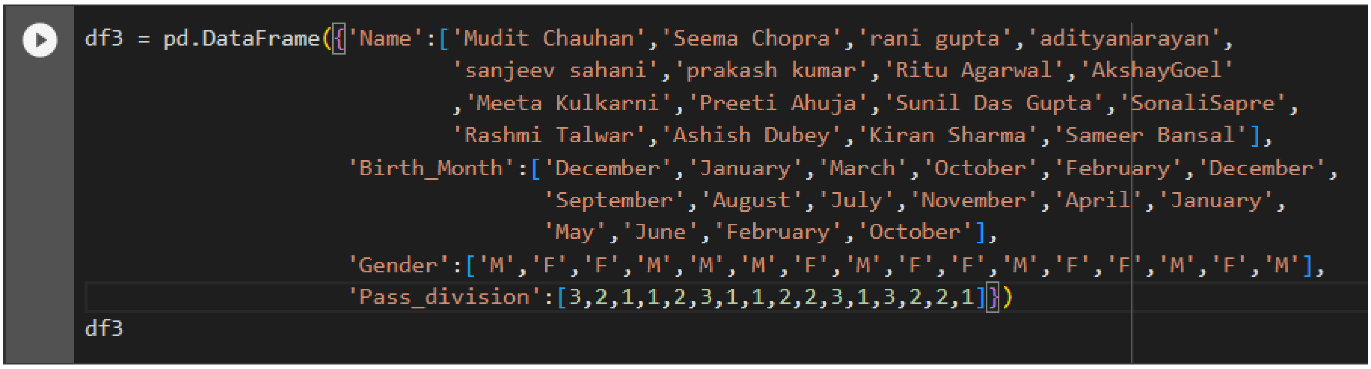
```
groups = data2.groupby(['Close', pd.cut(data2.Open, 3)])
result = groups.size().unstack()
print(result)
```

```
Open    (96.691, 116.503]  (116.503, 136.257]  (136.257, 156.01]
Close
96.67                   1                   0                  0
96.87                   1                   0                  0
96.98                   1                   0                  0
97.34                   1                   0                  0
97.42                   1                   0                  0
...                   ...                 ...                ...
155.37                  0                   0                  1
155.45                  0                   0                  1
155.47                  0                   0                  1
155.70                  0                   0                  1
156.10                  0                   0                  1

[239 rows x 3 columns]
```

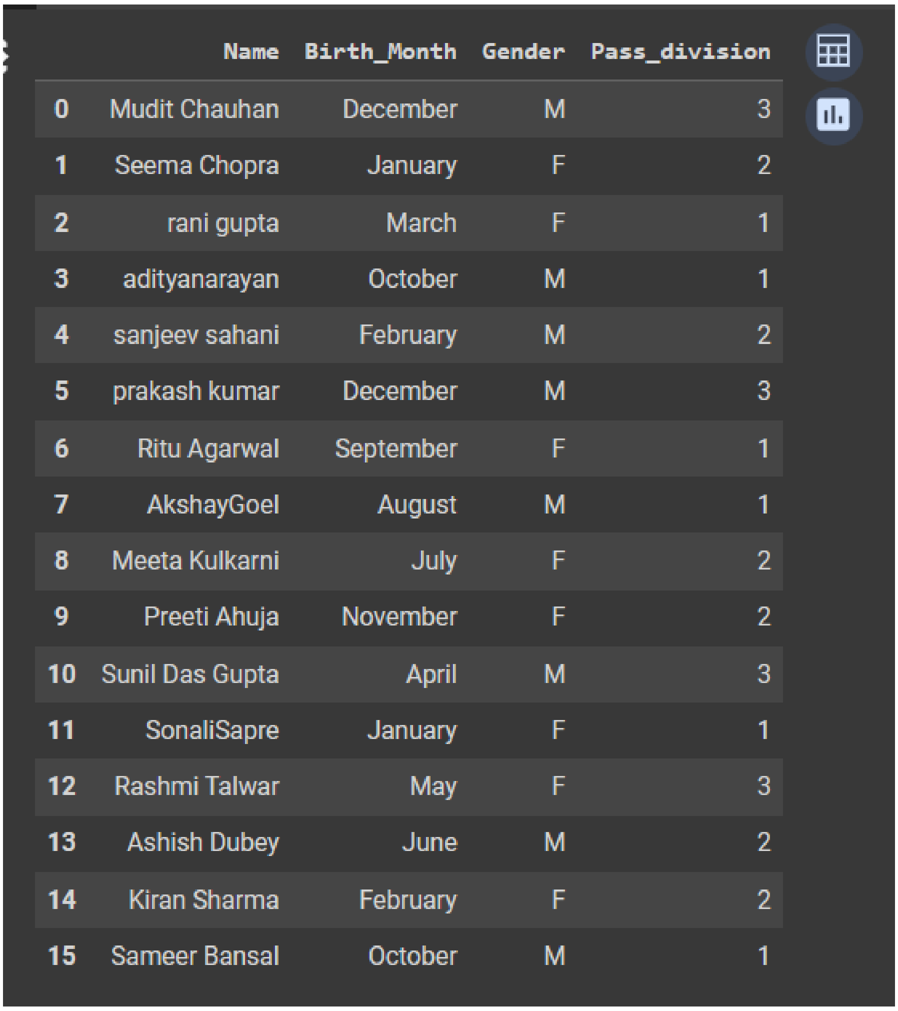
Ques7) Consider a data frame containing data about students i.e. name, gender and passing division:



	Name	Birth_Month	Gender	Pass_Division
0	Mudit Chauhan	December	M	III
1	Seema Chopra	January	F	II
2	Rani Gupta	March	F	I
3	Aditya Narayan	October	M	I
4	Sanjeev Sahni	February	M	II
5	Prakash Kumar	December	M	III
6	Ritu Agarwal	September	F	I
7	Akshay Goel	August	M	I
8	Meeta Kulkarni	July	F	II
9	Preeti Ahuja	November	F	II
10	Sunil Das Gupta	April	M	III
11	Sonali Sapre	January	F	I
12	Rashmi Talwar	June	F	III
13	Ashish Dubey	May	M	II
14	Kiran Sharma	February	F	II
15	Sameer Bansal	October	M	I

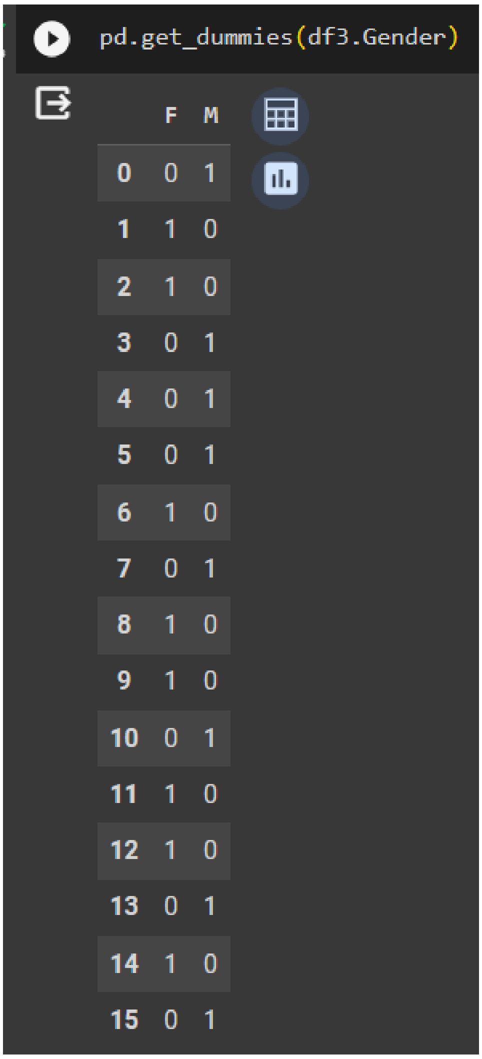


```
df3 = pd.DataFrame({'Name':['Mudit Chauhan','Seema Chopra','rani gupta','adityanarayan',
                            'sanjeev sahani','prakash kumar','Ritu Agarwal','AkshayGoel'
                            ,'Meeta Kulkarni','Preeti Ahuja','Sunil Das Gupta','SonaliSapre',
                            'Rashmi Talwar','Ashish Dubey','Kiran Sharma','Sameer Bansal'],
                    'Birth_Month':['December','January','March','October','February','December',
                                   'September','August','July','November','April','January',
                                   'May','June','February','October'],
                    'Gender':['M','F','F','M','M','M','F','M','F','F','M','F','F','M','F','M'],
                    'Pass_division':[3,2,1,1,2,3,1,1,2,2,3,1,3,2,2,1]})
df3
```



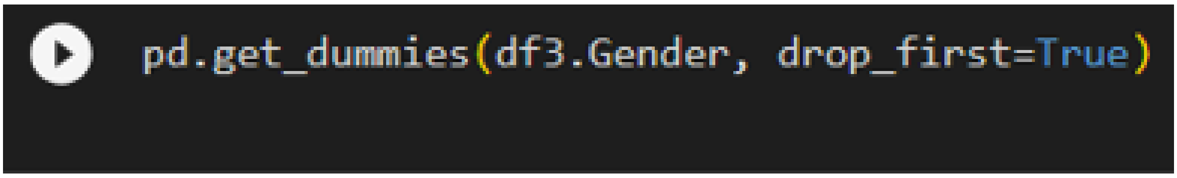
	Name	Birth_Month	Gender	Pass_division
0	Mudit Chauhan	December	M	3
1	Seema Chopra	January	F	2
2	rani gupta	March	F	1
3	adityanarayan	October	M	1
4	sanjeev sahani	February	M	2
5	prakash kumar	December	M	3
6	Ritu Agarwal	September	F	1
7	AkshayGoel	August	M	1
8	Meeta Kulkarni	July	F	2
9	Preeti Ahuja	November	F	2
10	Sunil Das Gupta	April	M	3
11	SonaliSapre	January	F	1
12	Rashmi Talwar	May	F	3
13	Ashish Dubey	June	M	2
14	Kiran Sharma	February	F	2
15	Sameer Bansal	October	M	1

a. Perform one hot encoding of the last two columns of categorical data using the get_dummies() function.

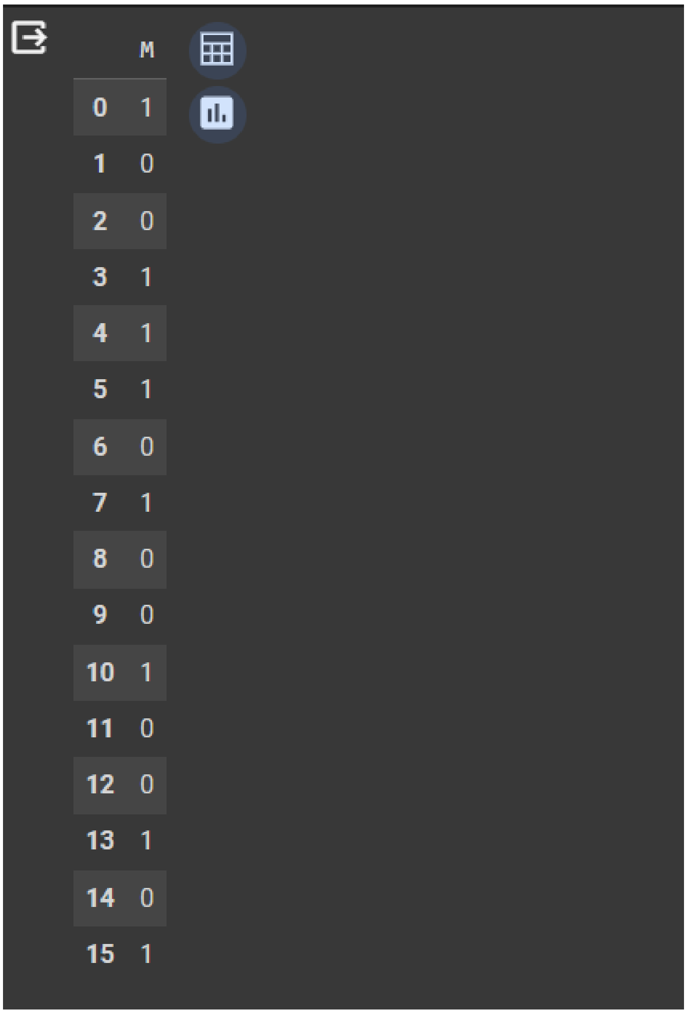


```
pd.get_dummies(df3.Gender)
```

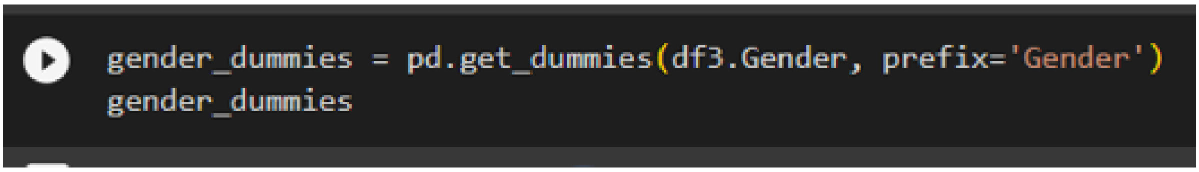
```
    F  M
0   0  1
1   1  0
2   1  0
3   0  1
4   0  1
5   0  1
6   1  0
7   0  1
8   1  0
9   1  0
10  0  1
11  1  0
12  1  0
13  0  1
14  1  0
15  0  1
```



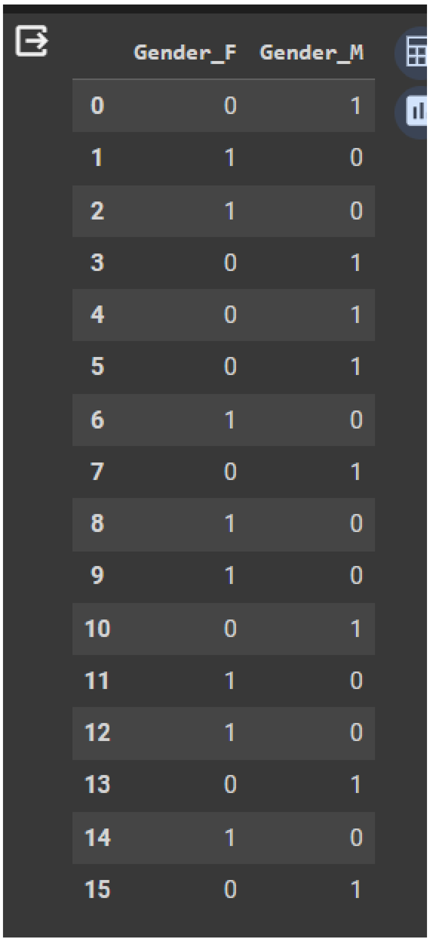
```
pd.get_dummies(df3.Gender, drop_first=True)
```



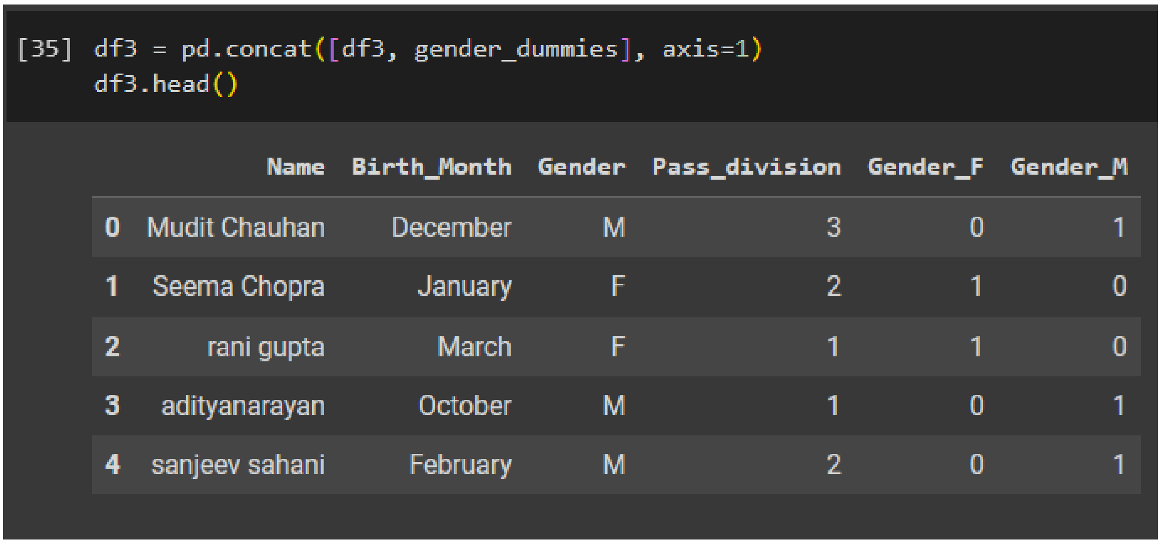
```
    M
0   1
1   0
2   0
3   1
4   1
5   1
6   0
7   1
8   0
9   0
10  1
11  0
12  0
13  1
14  0
15  1
```



```
gender_dummies = pd.get_dummies(df3.Gender, prefix='Gender')
gender_dummies
```

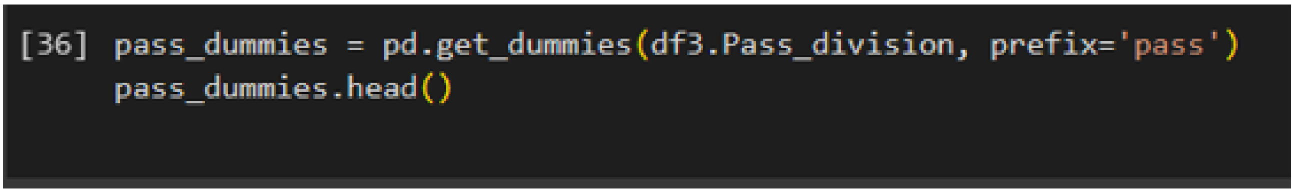


```
    Gender_F  Gender_M
0          0         1
1          1         0
2          1         0
3          0         1
4          0         1
5          0         1
6          1         0
7          0         1
8          1         0
9          1         0
10         0         1
11         1         0
12         1         0
13         0         1
14         1         0
15         0         1
```

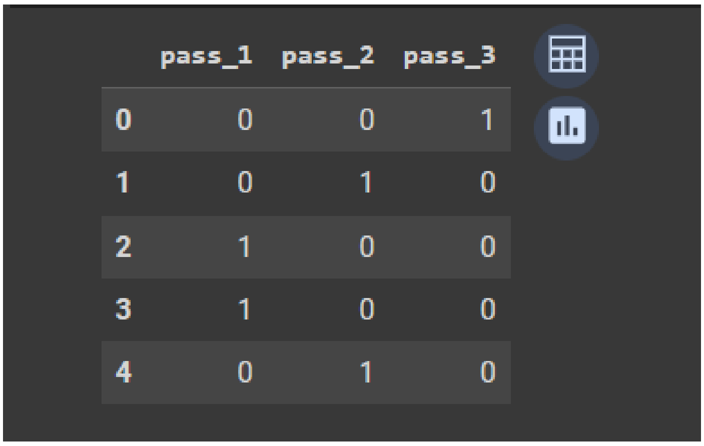


```
[35] df3 = pd.concat([df3, gender_dummies], axis=1)
     df3.head()
```

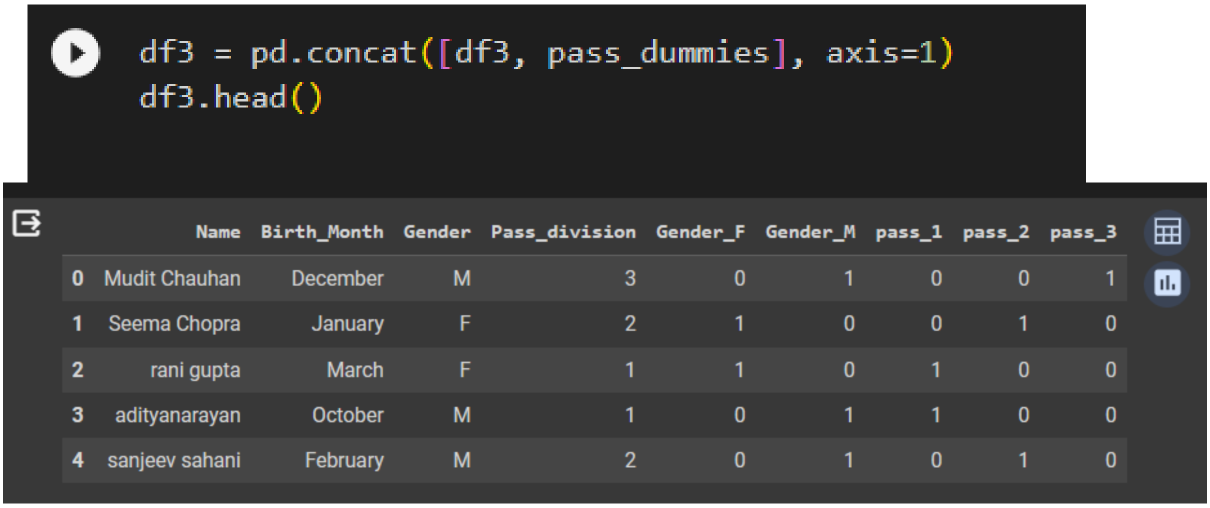
```
             Name  Birth_Month  Gender  Pass_division  Gender_F  Gender_M
0   Mudit Chauhan     December       M              3         0         1
1    Seema Chopra      January       F              2         1         0
2      rani gupta        March       F              1         1         0
3  adityanarayan       October       M              1         0         1
4   sanjeev sahani    February       M              2         0         1
```



```
[36] pass_dummies = pd.get_dummies(df3.Pass_division, prefix='pass')
     pass_dummies.head()
```



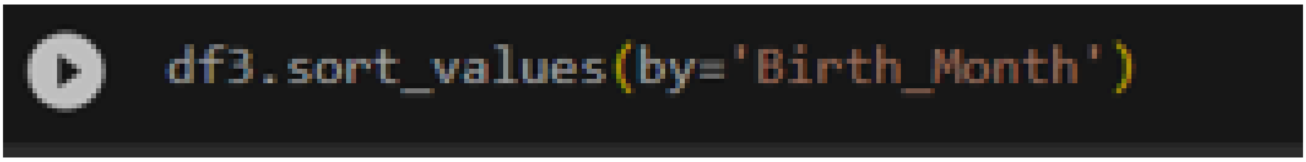
	pass_1	pass_2	pass_3
0	0	0	1
1	0	1	0
2	1	0	0
3	1	0	0
4	0	1	0



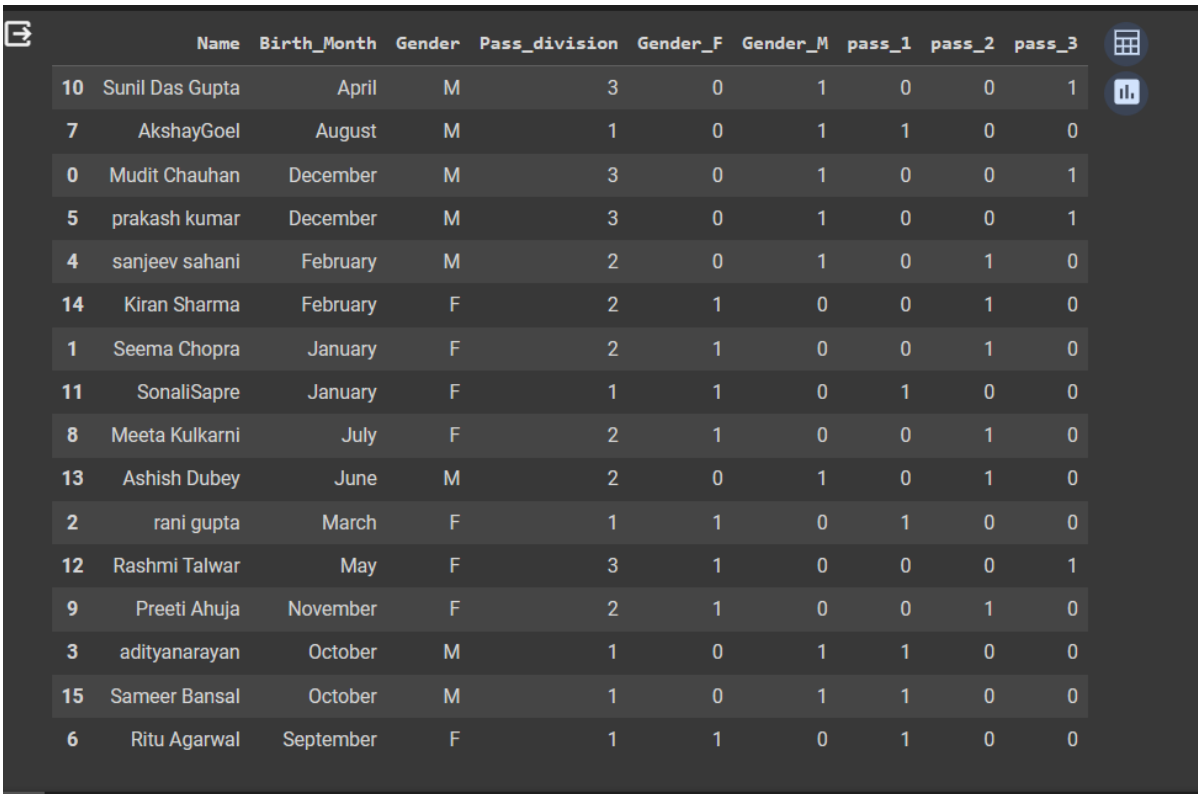
```
df3 = pd.concat([df3, pass_dummies], axis=1)
df3.head()
```

	Name	Birth_Month	Gender	Pass_division	Gender_F	Gender_M	pass_1	pass_2	pass_3
0	Mudit Chauhan	December	M	3	0	1	0	0	1
1	Seema Chopra	January	F	2	1	0	0	1	0
2	rani gupta	March	F	1	1	0	1	0	0
3	adityanarayan	October	M	1	0	1	1	0	0
4	sanjeev sahani	February	M	2	0	1	0	1	0

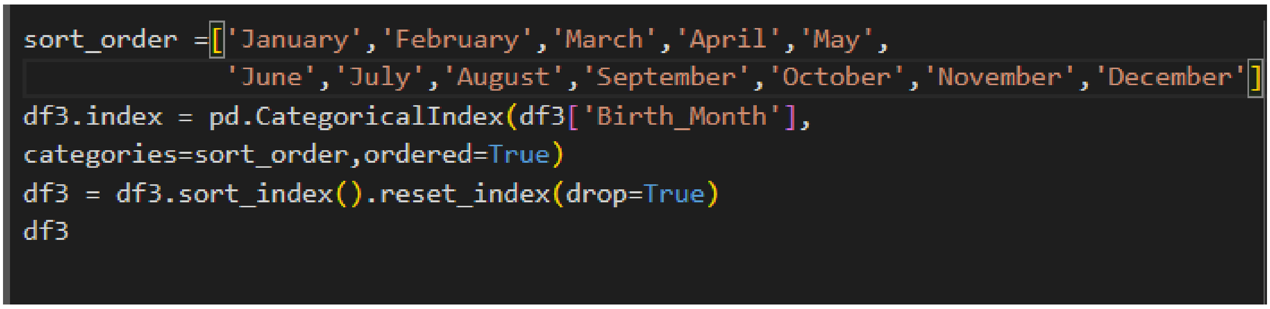
b. Sort this data frame on the “Birth Month” column (i.e. January to December). Hint: Convert Month to Categorical.



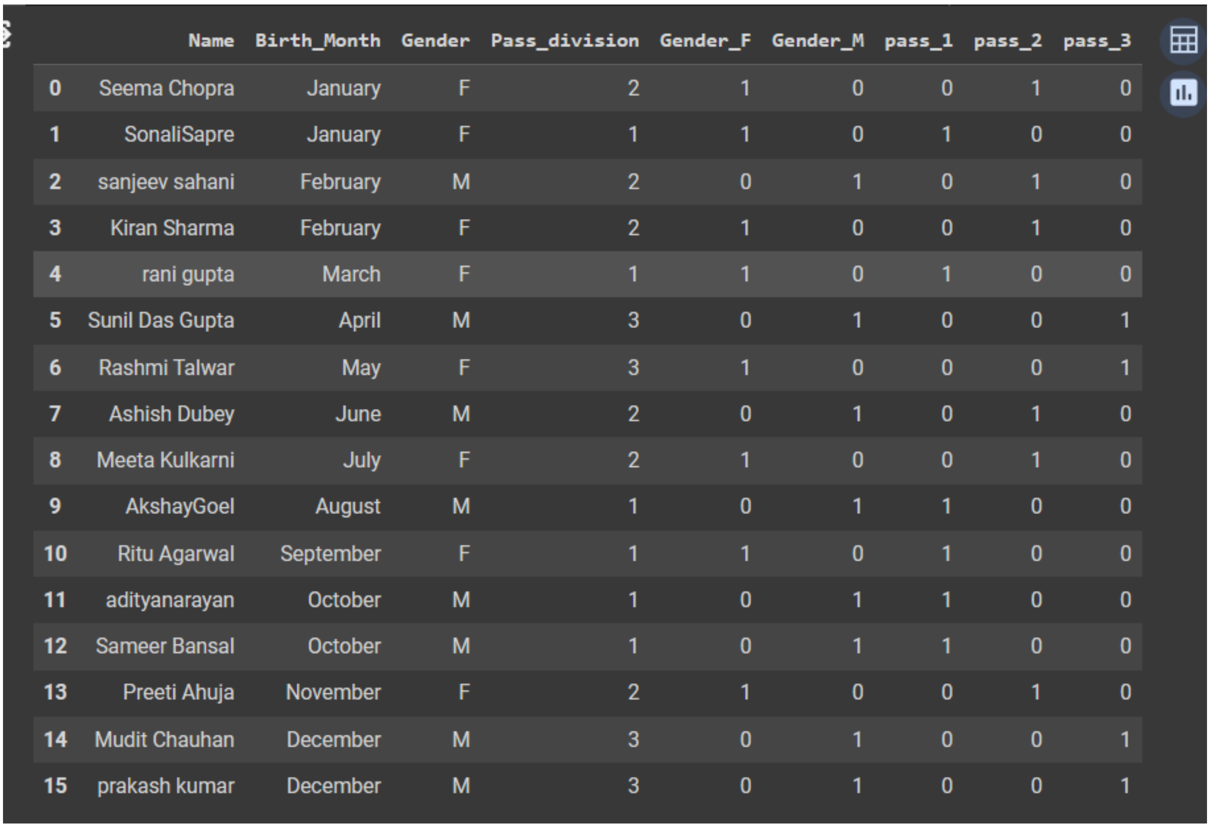
```
df3.sort_values(by='Birth_Month')
```



	Name	Birth_Month	Gender	Pass_division	Gender_F	Gender_M	pass_1	pass_2	pass_3
10	Sunil Das Gupta	April	M	3	0	1	0	0	1
7	AkshayGoel	August	M	1	0	1	1	0	0
0	Mudit Chauhan	December	M	3	0	1	0	0	1
5	prakash kumar	December	M	3	0	1	0	0	1
4	sanjeev sahani	February	M	2	0	1	0	1	0
14	Kiran Sharma	February	F	2	1	0	0	1	0
1	Seema Chopra	January	F	2	1	0	0	1	0
11	SonaliSapre	January	F	1	1	0	1	0	0
8	Meeta Kulkarni	July	F	2	1	0	0	1	0
13	Ashish Dubey	June	M	2	0	1	0	1	0
2	rani gupta	March	F	1	1	0	1	0	0
12	Rashmi Talwar	May	F	3	1	0	0	0	1
9	Preeti Ahuja	November	F	2	1	0	0	1	0
3	adityanarayan	October	M	1	0	1	1	0	0
15	Sameer Bansal	October	M	1	0	1	1	0	0
6	Ritu Agarwal	September	F	1	1	0	1	0	0



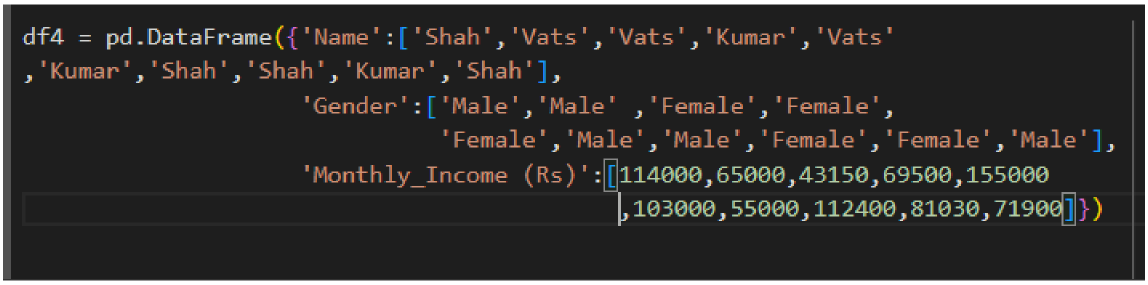
```
sort_order =['January','February','March','April','May',
             'June','July','August','September','October','November','December']
df3.index = pd.CategoricalIndex(df3['Birth_Month'],
categories=sort_order,ordered=True)
df3 = df3.sort_index().reset_index(drop=True)
df3
```



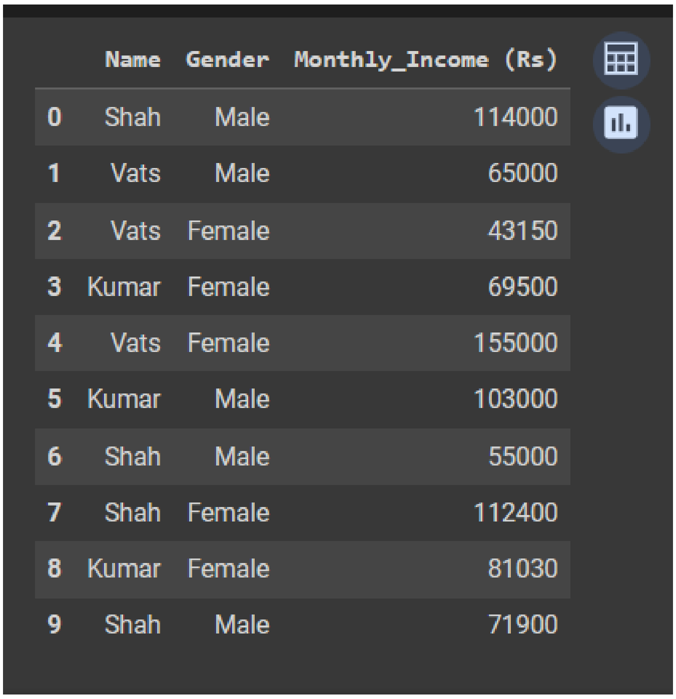
	Name	Birth_Month	Gender	Pass_division	Gender_F	Gender_M	pass_1	pass_2	pass_3
0	Seema Chopra	January	F	2	1	0	0	1	0
1	SonaliSapre	January	F	1	1	0	1	0	0
2	sanjeev sahani	February	M	2	0	1	0	1	0
3	Kiran Sharma	February	F	2	1	0	0	1	0
4	rani gupta	March	F	1	1	0	1	0	0
5	Sunil Das Gupta	April	M	3	0	1	0	0	1
6	Rashmi Talwar	May	F	3	1	0	0	0	1
7	Ashish Dubey	June	M	2	0	1	0	1	0
8	Meeta Kulkarni	July	F	2	1	0	0	1	0
9	AkshayGoel	August	M	1	0	1	1	0	0
10	Ritu Agarwal	September	F	1	1	0	1	0	0
11	adityanarayan	October	M	1	0	1	1	0	0
12	Sameer Bansal	October	M	1	0	1	1	0	0
13	Preeti Ahuja	November	F	2	1	0	0	1	0
14	Mudit Chauhan	December	M	3	0	1	0	0	1
15	prakash kumar	December	M	3	0	1	0	0	1

Ques 8) Consider the following data frame containing a family name, gender of the family member and her/his monthly income in each record.

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

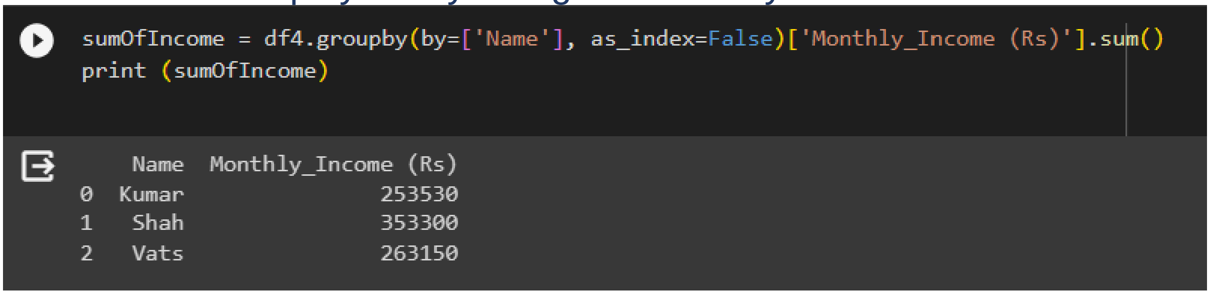


```
df4 = pd.DataFrame({'Name':['Shah','Vats','Vats','Kumar','Vats'
,'Kumar','Shah','Shah','Kumar','Shah'],
                    'Gender':['Male','Male' ,'Female','Female',
                              'Female','Male','Male','Female','Female','Male'],
                    'Monthly_Income (Rs)':[114000,65000,43150,69500,155000
                                          ,103000,55000,112400,81030,71900]})
```



	Name	Gender	Monthly_Income (Rs)
0	Shah	Male	114000
1	Vats	Male	65000
2	Vats	Female	43150
3	Kumar	Female	69500
4	Vats	Female	155000
5	Kumar	Male	103000
6	Shah	Male	55000
7	Shah	Female	112400
8	Kumar	Female	81030
9	Shah	Male	71900

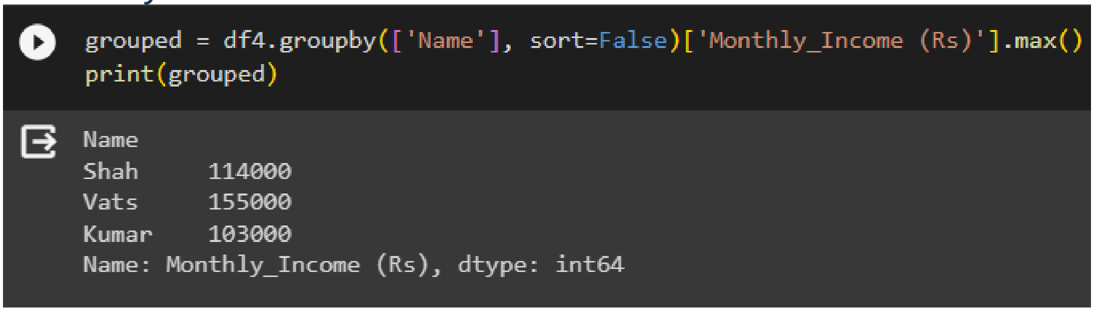
a. Calculate and display familywise gross monthly income.



```
sumOfIncome = df4.groupby(by=['Name'], as_index=False)['Monthly_Income (Rs)'].sum()
print (sumOfIncome)
```

```
    Name  Monthly_Income (Rs)
0  Kumar               253530
1   Shah               353300
2   Vats               263150
```

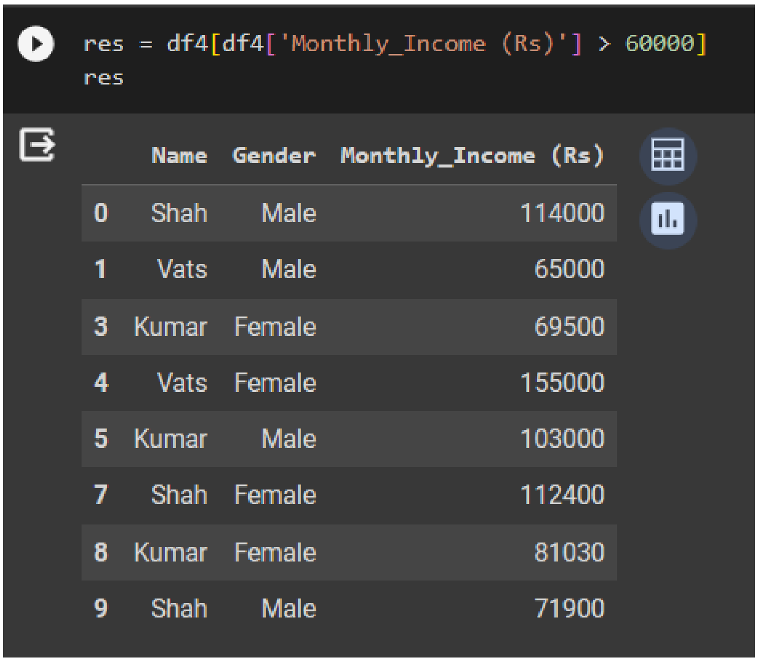
b. Calculate and display the member with the highest monthly income in a family.



```
grouped = df4.groupby(['Name'], sort=False)['Monthly_Income (Rs)'].max()
print(grouped)
```

```
Name
Shah     114000
Vats     155000
Kumar    103000
Name: Monthly_Income (Rs), dtype: int64
```

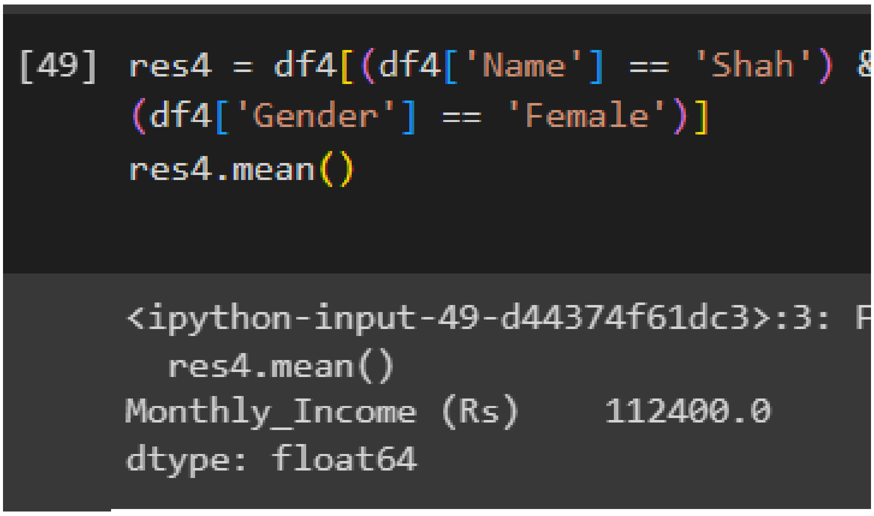
c. Calculate and display monthly income of all members with income greater than Rs. 60000.00.



```
res = df4[df4['Monthly_Income (Rs)'] > 60000]
res
```

	Name	Gender	Monthly_Income (Rs)
0	Shah	Male	114000
1	Vats	Male	65000
3	Kumar	Female	69500
4	Vats	Female	155000
5	Kumar	Male	103000
7	Shah	Female	112400
8	Kumar	Female	81030
9	Shah	Male	71900

d. Calculate and display the average monthly income of the female members in the Shah family.



```
[49] res4 = df4[(df4['Name'] == 'Shah') &
     (df4['Gender'] == 'Female')]
     res4.mean()
```

```
<ipython-input-49-d44374f61dc3>:3: F
  res4.mean()
Monthly_Income (Rs)    112400.0
dtype: float64
```