Que 1. 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]

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

In [4]:
```
n_list=[]

for i in range(len(dictt['Boys'])):
    n_d={}
    for j in dictt:
        n_d[j]=dictt[j][i]

    n_list.append(n_d)
n_list
```

Out[4]:
```
[{'Boys': 72, 'Girls': 63},
 {'Boys': 68, 'Girls': 65},
 {'Boys': 70, 'Girls': 69},
 {'Boys': 69, 'Girls': 62},
 {'Boys': 74, 'Girls': 61}]
```

Que 2. 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.

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

a. B = [56, 48, 22, 41, 78, 91, 24, 46, 8, 33]

c. 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 nx m array, n and m are user inputs given at the run time.

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.

In [2]:
```
import numpy as np
```

In [40]:
```
arr=np.random.randint(low=1,high=20,size=(4,5))
arr
```

Out[40]:
```
array([[ 3,  2,  8, 13,  6],
       [ 4,  2,  8,  5,  5],
       [17, 19, 19, 19, 12],
       [ 1,  1,  6, 18,  3]])
```

In [50]:
```
x=arr.mean(axis=1)
print("Mean =",x)
```

```
Mean = [ 6.4  4.8 17.2  5.8]
```

In [51]:
```
x=arr.var(axis=1)
print("Variance=",x)
```

```
Variance= [15.44  3.76  7.36 40.56]
```

In [52]:
```
y=arr.std(axis=1)
print("Standard Deviation= ",y)
```

```
Standard Deviation=  [3.92937654 1.93907194 2.71293199 6.36867333]
```

In [67]:
```
B=np.array([56, 48, 22, 41, 78, 91, 24, 46, 8, 33])
B.ndim
```

Out[67]:
```
1
```

In [68]:
```
#np.sort()
#B
```

In [69]:
```
x=B.argsort() #return the sorted output indexs
y=np.sort(B)
print("Sort", y)
print("Sortedindex",x)
```

```
Sort [ 8 22 24 33 41 46 48 56 78 91]
Sortedindex [8 2 6 9 3 7 1 0 4 5]
```

In [75]:
```
n=int(input("Enter array row"))
m=int(input("Enter array col"))
ar=np.random.randint(low=1,high=50,size=(n,m))
ar
```

```
Enter array row3
Enter array col4
```

Out[75]:
```
array([[28, 27, 36, 45],
       [16, 46, 10, 24],
       [15, 39, 25, 14]])
```

In [77]:
```
print("Dim= ",ar.ndim)
```

```
Shape 2
```

In [78]:
```
print("Shape =",ar.shape)
```

```
Shape = (3, 4)
```

In [79]:
```
print("Type= ",ar.dtype)
```

```
Type=  int32
```

In [80]:
```
print("reshape= ",ar.reshape(4,3))
```

```
reshape=  [[28 27 36]
 [45 16 46]
 [10 24 15]
 [39 25 14]]
```

In [37]:
```
aa=np.array([2,0,3,4,1,34,np.nan,np.nan])
aa
```

Out[37]:
```
array([ 2.,  0.,  3.,  4.,  1., 34., nan, nan])
```

In [15]:
```
s=np.array([1,2,5,0,2])
np.all(s)
ind=np.where(s==0)[0]
print("Index in new array = ",ind)
```

```
Index in new array =  [3]
```

In [24]:
```
np.any(s)
ind=np.where(s!=0)[0]
print("Index in new array = ",ind)
```

```
Index in new array =  [0 1 2 4]
```

In [18]:
```
np.isnan(aa)
```

Out[18]:
```
array([False, False, False, False, False,  True,  True])
```

In [22]:
```
index=np.where(np.isnan(aa)==True)[0]
print("Index = ", index)
```

```
Index =  [5 6]
```

Out[22]:
```
numpy.ndarray
```

In [46]:
```
a0=[]
an0=[]
anan=[]

for i in range(aa.size):
    if(aa[i]==0):

        a0.append(i)

    elif(np.isnan(aa[i])):
        anan.append(i)
    else:
        an0.append(i)


print(a0)
```

```
[1]
```

In [47]:
```
ar_zero=np.array(a0)
ar_nonzero=np.array(anan)
ar_nan=np.array(an0)
```

In [48]:
```
ar_zero
```

Out[48]:
```
array([1])
```

In [49]:
```
ar_nonzero
```

Out[49]:
```
array([6, 7])
```

In [50]:
```
ar_nan
```

Out[50]:
```
array([0, 2, 3, 4, 5])
```

QUE 3. 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:

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

In [102…

```
df=pd.DataFrame(np.random.randint(1,100,(50,3)),columns=list('123'))
df
```

Out[102]:

	1	2	3
0	81	11	99
1	92	51	21
2	33	62	20
3	67	54	76
4	7	38	68
5	45	2	52
6	64	8	74
7	22	71	81
8	10	9	49
9	32	25	58
10	44	24	28
11	20	28	74
12	39	75	17
13	78	39	76
14	9	35	83
15	56	25	43
16	42	69	83
17	16	35	82
18	25	8	64
19	30	14	1
20	21	35	74
21	14	84	11
22	76	57	82
23	3	41	1
24	32	92	67
25	26	9	76
26	29	70	75
27	3	11	85
28	35	54	84
29	95	10	37
30	15	51	68
31	29	45	29
32	35	16	80
33	99	46	24
34	45	69	34
35	9	3	5

	1	2	3
36	22	85	14
37	97	44	12
38	10	75	50
39	37	87	40
40	25	9	63
41	30	49	2
42	29	66	39
43	85	71	8
44	61	93	77
45	15	24	52
46	23	33	27
47	26	88	91
48	44	5	32
49	58	90	67

x=df.sample(frac=.10) #Pandas sample() is used to generate a sample random row or column from the function caller data frame. x

In [103…

```
for c in  df.sample(int(df.shape[0]*df.shape[1]*0.10)).index:
    df.loc[c,str(np.random.randint(1,4))]=np.nan
df
```

Out[103]:

	1	2	3
0	81.0	11.0	99.0
1	92.0	51.0	21.0
2	33.0	62.0	20.0
3	67.0	54.0	76.0
4	7.0	38.0	68.0
5	45.0	2.0	52.0
6	NaN	8.0	74.0
7	NaN	71.0	81.0
8	10.0	9.0	49.0
9	NaN	25.0	58.0
10	44.0	24.0	28.0
11	20.0	28.0	74.0
12	39.0	75.0	17.0
13	78.0	39.0	NaN
14	9.0	35.0	83.0
15	56.0	25.0	43.0
16	42.0	69.0	83.0
17	16.0	35.0	82.0
18	25.0	8.0	64.0
19	30.0	14.0	1.0
20	21.0	NaN	74.0
21	14.0	84.0	11.0
22	NaN	57.0	82.0
23	3.0	41.0	NaN
24	32.0	92.0	67.0
25	26.0	9.0	76.0
26	29.0	70.0	75.0
27	3.0	11.0	85.0
28	35.0	54.0	84.0
29	NaN	10.0	37.0
30	15.0	51.0	68.0
31	29.0	45.0	NaN
32	35.0	16.0	80.0
33	99.0	46.0	24.0
34	45.0	69.0	34.0
35	9.0	3.0	5.0

	1	2	3
36	22.0	85.0	NaN
37	97.0	44.0	12.0
38	NaN	75.0	50.0
39	37.0	87.0	40.0
40	25.0	9.0	63.0
41	30.0	49.0	2.0
42	29.0	66.0	39.0
43	85.0	71.0	8.0
44	61.0	NaN	77.0
45	15.0	24.0	52.0
46	23.0	NaN	27.0
47	26.0	88.0	91.0
48	44.0	5.0	NaN
49	NaN	90.0	67.0

a. Identify and count missing values in a dataframe

In [104…
```
df.isnull().sum()
```

Out[104]:
```
1    7
2    3
3    5
dtype: int64
```

b. Drop the column having more than 5 null values

In [105…
```
newdf=df.copy()

for i in df.columns:
    if(newdf[i].isnull().sum()>5):

        del newdf[i]
newdf
```

Out[105]:

	2	3
0	11.0	99.0
1	51.0	21.0
2	62.0	20.0
3	54.0	76.0
4	38.0	68.0
5	2.0	52.0
6	8.0	74.0
7	71.0	81.0
8	9.0	49.0
9	25.0	58.0
10	24.0	28.0
11	28.0	74.0
12	75.0	17.0
13	39.0	NaN
14	35.0	83.0
15	25.0	43.0
16	69.0	83.0
17	35.0	82.0
18	8.0	64.0
19	14.0	1.0
20	NaN	74.0
21	84.0	11.0
22	57.0	82.0
23	41.0	NaN
24	92.0	67.0
25	9.0	76.0
26	70.0	75.0
27	11.0	85.0
28	54.0	84.0
29	10.0	37.0
30	51.0	68.0
31	45.0	NaN
32	16.0	80.0
33	46.0	24.0
34	69.0	34.0
35	3.0	5.0

	2	3
36	85.0	NaN
37	44.0	12.0
38	75.0	50.0
39	87.0	40.0
40	9.0	63.0
41	49.0	2.0
42	66.0	39.0
43	71.0	8.0
44	NaN	77.0
45	24.0	52.0
46	NaN	27.0
47	88.0	91.0
48	5.0	NaN
49	90.0	67.0

In [106…

```
df
```

Out[106]:

	1	2	3
0	81.0	11.0	99.0
1	92.0	51.0	21.0
2	33.0	62.0	20.0
3	67.0	54.0	76.0
4	7.0	38.0	68.0
5	45.0	2.0	52.0
6	NaN	8.0	74.0
7	NaN	71.0	81.0
8	10.0	9.0	49.0
9	NaN	25.0	58.0
10	44.0	24.0	28.0
11	20.0	28.0	74.0
12	39.0	75.0	17.0
13	78.0	39.0	NaN
14	9.0	35.0	83.0
15	56.0	25.0	43.0
16	42.0	69.0	83.0
17	16.0	35.0	82.0
18	25.0	8.0	64.0
19	30.0	14.0	1.0
20	21.0	NaN	74.0
21	14.0	84.0	11.0
22	NaN	57.0	82.0
23	3.0	41.0	NaN
24	32.0	92.0	67.0
25	26.0	9.0	76.0
26	29.0	70.0	75.0
27	3.0	11.0	85.0
28	35.0	54.0	84.0
29	NaN	10.0	37.0
30	15.0	51.0	68.0
31	29.0	45.0	NaN
32	35.0	16.0	80.0
33	99.0	46.0	24.0
34	45.0	69.0	34.0
35	9.0	3.0	5.0

	1	2	3
36	22.0	85.0	NaN
37	97.0	44.0	12.0
38	NaN	75.0	50.0
39	37.0	87.0	40.0
40	25.0	9.0	63.0
41	30.0	49.0	2.0
42	29.0	66.0	39.0
43	85.0	71.0	8.0
44	61.0	NaN	77.0
45	15.0	24.0	52.0
46	23.0	NaN	27.0
47	26.0	88.0	91.0
48	44.0	5.0	NaN
49	NaN	90.0	67.0

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

In [107…

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

Out[107]:

	1	2	3	sum
0	81.0	11.0	99.0	191.0
1	92.0	51.0	21.0	164.0
2	33.0	62.0	20.0	115.0
3	67.0	54.0	76.0	197.0
4	7.0	38.0	68.0	113.0
5	45.0	2.0	52.0	99.0
6	NaN	8.0	74.0	82.0
7	NaN	71.0	81.0	152.0
8	10.0	9.0	49.0	68.0
9	NaN	25.0	58.0	83.0
10	44.0	24.0	28.0	96.0
11	20.0	28.0	74.0	122.0
12	39.0	75.0	17.0	131.0
13	78.0	39.0	NaN	117.0
14	9.0	35.0	83.0	127.0
15	56.0	25.0	43.0	124.0
16	42.0	69.0	83.0	194.0
17	16.0	35.0	82.0	133.0
18	25.0	8.0	64.0	97.0
19	30.0	14.0	1.0	45.0
20	21.0	NaN	74.0	95.0
21	14.0	84.0	11.0	109.0
22	NaN	57.0	82.0	139.0
23	3.0	41.0	NaN	44.0
24	32.0	92.0	67.0	191.0
25	26.0	9.0	76.0	111.0
26	29.0	70.0	75.0	174.0
27	3.0	11.0	85.0	99.0
28	35.0	54.0	84.0	173.0
29	NaN	10.0	37.0	47.0
30	15.0	51.0	68.0	134.0
31	29.0	45.0	NaN	74.0
32	35.0	16.0	80.0	131.0
33	99.0	46.0	24.0	169.0
34	45.0	69.0	34.0	148.0
35	9.0	3.0	5.0	17.0

	1	2	3	sum
36	22.0	85.0	NaN	107.0
37	97.0	44.0	12.0	153.0
38	NaN	75.0	50.0	125.0
39	37.0	87.0	40.0	164.0
40	25.0	9.0	63.0	97.0
41	30.0	49.0	2.0	81.0
42	29.0	66.0	39.0	134.0
43	85.0	71.0	8.0	164.0
44	61.0	NaN	77.0	138.0
45	15.0	24.0	52.0	91.0
46	23.0	NaN	27.0	50.0
47	26.0	88.0	91.0	205.0
48	44.0	5.0	NaN	49.0
49	NaN	90.0	67.0	157.0

In [109…

```
newdf=df.copy()
m=newdf['sum'].idxmax()
print("Maximum index = ",m)
newdf.drop(m)
```

47

Out[109]:

	1	2	3	sum
0	81.0	11.0	99.0	191.0
1	92.0	51.0	21.0	164.0
2	33.0	62.0	20.0	115.0
3	67.0	54.0	76.0	197.0
4	7.0	38.0	68.0	113.0
5	45.0	2.0	52.0	99.0
6	NaN	8.0	74.0	82.0
7	NaN	71.0	81.0	152.0
8	10.0	9.0	49.0	68.0
9	NaN	25.0	58.0	83.0
10	44.0	24.0	28.0	96.0
11	20.0	28.0	74.0	122.0
12	39.0	75.0	17.0	131.0
13	78.0	39.0	NaN	117.0
14	9.0	35.0	83.0	127.0
15	56.0	25.0	43.0	124.0
16	42.0	69.0	83.0	194.0
17	16.0	35.0	82.0	133.0
18	25.0	8.0	64.0	97.0
19	30.0	14.0	1.0	45.0
20	21.0	NaN	74.0	95.0
21	14.0	84.0	11.0	109.0
22	NaN	57.0	82.0	139.0
23	3.0	41.0	NaN	44.0
24	32.0	92.0	67.0	191.0
25	26.0	9.0	76.0	111.0
26	29.0	70.0	75.0	174.0
27	3.0	11.0	85.0	99.0
28	35.0	54.0	84.0	173.0
29	NaN	10.0	37.0	47.0
30	15.0	51.0	68.0	134.0
31	29.0	45.0	NaN	74.0
32	35.0	16.0	80.0	131.0
33	99.0	46.0	24.0	169.0
34	45.0	69.0	34.0	148.0
35	9.0	3.0	5.0	17.0

	1	2	3	sum
36	22.0	85.0	NaN	107.0
37	97.0	44.0	12.0	153.0
38	NaN	75.0	50.0	125.0
39	37.0	87.0	40.0	164.0
40	25.0	9.0	63.0	97.0
41	30.0	49.0	2.0	81.0
42	29.0	66.0	39.0	134.0
43	85.0	71.0	8.0	164.0
44	61.0	NaN	77.0	138.0
45	15.0	24.0	52.0	91.0
46	23.0	NaN	27.0	50.0
48	44.0	5.0	NaN	49.0
49	NaN	90.0	67.0	157.0

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

In [112…

```
sort=df.sort_values('1',ascending=True)  #by=[1] df.columns[0]
sort
```

Out[112]:

	1	2	3	sum
27	3.0	11.0	85.0	99.0
23	3.0	41.0	NaN	44.0
4	7.0	38.0	68.0	113.0
35	9.0	3.0	5.0	17.0
14	9.0	35.0	83.0	127.0
8	10.0	9.0	49.0	68.0
21	14.0	84.0	11.0	109.0
30	15.0	51.0	68.0	134.0
45	15.0	24.0	52.0	91.0
17	16.0	35.0	82.0	133.0
11	20.0	28.0	74.0	122.0
20	21.0	NaN	74.0	95.0
36	22.0	85.0	NaN	107.0
46	23.0	NaN	27.0	50.0
40	25.0	9.0	63.0	97.0
18	25.0	8.0	64.0	97.0
47	26.0	88.0	91.0	205.0
25	26.0	9.0	76.0	111.0
26	29.0	70.0	75.0	174.0
31	29.0	45.0	NaN	74.0
42	29.0	66.0	39.0	134.0
41	30.0	49.0	2.0	81.0
19	30.0	14.0	1.0	45.0
24	32.0	92.0	67.0	191.0
2	33.0	62.0	20.0	115.0
28	35.0	54.0	84.0	173.0
32	35.0	16.0	80.0	131.0
39	37.0	87.0	40.0	164.0
12	39.0	75.0	17.0	131.0
16	42.0	69.0	83.0	194.0
48	44.0	5.0	NaN	49.0
10	44.0	24.0	28.0	96.0
5	45.0	2.0	52.0	99.0
34	45.0	69.0	34.0	148.0
15	56.0	25.0	43.0	124.0
44	61.0	NaN	77.0	138.0

	1	2	3	sum
3	67.0	54.0	76.0	197.0
13	78.0	39.0	NaN	117.0
0	81.0	11.0	99.0	191.0
43	85.0	71.0	8.0	164.0
1	92.0	51.0	21.0	164.0
37	97.0	44.0	12.0	153.0
33	99.0	46.0	24.0	169.0
6	NaN	8.0	74.0	82.0
7	NaN	71.0	81.0	152.0
9	NaN	25.0	58.0	83.0
22	NaN	57.0	82.0	139.0
29	NaN	10.0	37.0	47.0
38	NaN	75.0	50.0	125.0
49	NaN	90.0	67.0	157.0

e. Remove all duplicates from the first column

In [116…

```
dupli=df.drop_duplicates(subset='1') #,keep='first')
dupli
```

Out[116]:

	1	2	3	sum
0	81.0	11.0	99.0	191.0
1	92.0	51.0	21.0	164.0
2	33.0	62.0	20.0	115.0
3	67.0	54.0	76.0	197.0
4	7.0	38.0	68.0	113.0
5	45.0	2.0	52.0	99.0
6	NaN	8.0	74.0	82.0
8	10.0	9.0	49.0	68.0
10	44.0	24.0	28.0	96.0
11	20.0	28.0	74.0	122.0
12	39.0	75.0	17.0	131.0
13	78.0	39.0	NaN	117.0
14	9.0	35.0	83.0	127.0
15	56.0	25.0	43.0	124.0
16	42.0	69.0	83.0	194.0
17	16.0	35.0	82.0	133.0
18	25.0	8.0	64.0	97.0
19	30.0	14.0	1.0	45.0
20	21.0	NaN	74.0	95.0
21	14.0	84.0	11.0	109.0
23	3.0	41.0	NaN	44.0
24	32.0	92.0	67.0	191.0
25	26.0	9.0	76.0	111.0
26	29.0	70.0	75.0	174.0
28	35.0	54.0	84.0	173.0
30	15.0	51.0	68.0	134.0
33	99.0	46.0	24.0	169.0
36	22.0	85.0	NaN	107.0
37	97.0	44.0	12.0	153.0
39	37.0	87.0	40.0	164.0
43	85.0	71.0	8.0	164.0
44	61.0	NaN	77.0	138.0
46	23.0	NaN	27.0	50.0

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

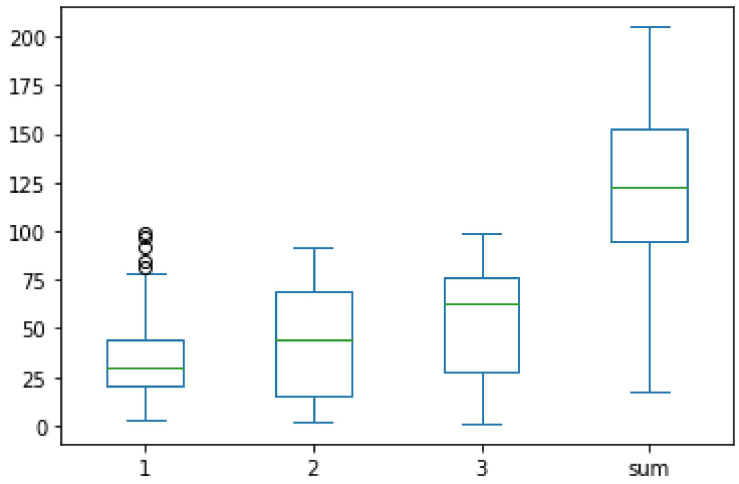
In [117…
```
cor=df['1'].corr(df['2'])
print("CORRELATION between column 1 and 2 : ", cor)
covariance = df['2'].cov(df['3'])
print("COVARIANCE between column 2 and 3 :",covariance)
```

```
CORRELATION between column 1 and 2 :  0.10483572382089804
COVARIANCE between column 2 and 3 : -54.016840882694574
```

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

In [119…
```
df.plot.box()
```

Out[119]:
```
<AxesSubplot:>
```



h.) Discretize second column and create 5 bins

In [124…
```
cats=pd.qcut(df['2'],5)
cats
```

Out[124]:

```
0     (1.999, 11.0]
1      (35.0, 51.0]
2      (51.0, 70.8]
3      (51.0, 70.8]
4      (35.0, 51.0]
5     (1.999, 11.0]
6     (1.999, 11.0]
7      (70.8, 92.0]
8     (1.999, 11.0]
9      (11.0, 35.0]
10     (11.0, 35.0]
11     (11.0, 35.0]
12     (70.8, 92.0]
13     (35.0, 51.0]
14     (11.0, 35.0]
15     (11.0, 35.0]
16     (51.0, 70.8]
17     (11.0, 35.0]
18    (1.999, 11.0]
19     (11.0, 35.0]
20               NaN
21     (70.8, 92.0]
22     (51.0, 70.8]
23     (35.0, 51.0]
24     (70.8, 92.0]
25    (1.999, 11.0]
26     (51.0, 70.8]
27    (1.999, 11.0]
28     (51.0, 70.8]
29    (1.999, 11.0]
30     (35.0, 51.0]
31     (35.0, 51.0]
32     (11.0, 35.0]
33     (35.0, 51.0]
34     (51.0, 70.8]
35    (1.999, 11.0]
36     (70.8, 92.0]
37     (35.0, 51.0]
38     (70.8, 92.0]
39     (70.8, 92.0]
40    (1.999, 11.0]
41     (35.0, 51.0]
42     (51.0, 70.8]
43     (70.8, 92.0]
44               NaN
45     (11.0, 35.0]
46               NaN
47     (70.8, 92.0]
48    (1.999, 11.0]
49     (70.8, 92.0]
Name: 2, dtype: category
Categories (5, interval[float64, right]): [(1.999, 11.0] < (11.0, 35.0] < (35.0, 5
1.0] < (51.0, 70.8] < (70.8, 92.0]]
```

Que 4. 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:

In [126…

```
df1=pd.read_excel('workshop1.xlsx')
df1
```

Out[126]:

	Name	Time of Joining	Duration
0	Shikha	08:40:00	50
1	Deepali	09:00:00	50
2	Purvi	09:15:00	40
3	Seemran	09:30:00	30
4	Riya	10:00:00	40
5	Walsha	10:20:00	30
6	Naman	10:00:00	40
7	Priyanka	10:30:00	50
8	Arvind	10:00:00	50
9	Cheshta	10:40:00	30

In [127…
```
df2=pd.read_excel('workshop2.xlsx')
df2
```

Out[127]:

	Name	Time of Joining	Duration
0	Leena	08:40:00	50
1	Deepali	10:10:00	40
2	Pooja	09:15:00	50
3	Nikita	10:12:00	50
4	Riya	09:40:00	50
5	Akansha	10:20:00	40
6	Neha	09:30:00	30
7	Priyanka	09:50:00	30
8	Tamanna	11:00:00	40
9	Cheshta	09:10:00	40

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

In [133…
```
both=pd.merge(df1,df2,how='inner',on='Name')
both
```

Out[133]:

	Name	Time of Joining_x	Duration_x	Time of Joining_y	Duration_y
0	Deepali	09:00:00	50	10:10:00	40
1	Riya	10:00:00	40	09:40:00	50
2	Priyanka	10:30:00	50	09:50:00	30
3	Cheshta	10:40:00	30	09:10:00	40

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

In [134…
```
either=pd.merge(df1,df2,how='outer',on='Name')
either
```

Out[134]:

	Name	Time of Joining_x	Duration_x	Time of Joining_y	Duration_y
0	Shikha	08:40:00	50.0	NaN	NaN
1	Deepali	09:00:00	50.0	10:10:00	40.0
2	Purvi	09:15:00	40.0	NaN	NaN
3	Seemran	09:30:00	30.0	NaN	NaN
4	Riya	10:00:00	40.0	09:40:00	50.0
5	Walsha	10:20:00	30.0	NaN	NaN
6	Naman	10:00:00	40.0	NaN	NaN
7	Priyanka	10:30:00	50.0	09:50:00	30.0
8	Arvind	10:00:00	50.0	NaN	NaN
9	Cheshta	10:40:00	30.0	09:10:00	40.0
10	Leena	NaN	NaN	08:40:00	50.0
11	Pooja	NaN	NaN	09:15:00	50.0
12	Nikita	NaN	NaN	10:12:00	50.0
13	Akansha	NaN	NaN	10:20:00	40.0
14	Neha	NaN	NaN	09:30:00	30.0
15	Tamanna	NaN	NaN	11:00:00	40.0

c.) Merge two dataframes row-wise and find the total number of records in the dataframe

In [135…
```
either['Name'].count()
```

Out[135]:
```
16
```

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

In [139…
```
d=pd.merge(df1,df2,how='outer',on=['Name','Duration']).copy()
d.set_index(['Name','Duration']).sort_index()

#or  df=pd.concat([df1,df2]).set_index(['Name','Duration']).sort_index()
```

Out[139]:

Name	Duration	Time of Joining_x	Time of Joining_y
Akansha	**40**	NaN	10:20:00
Arvind	**50**	10:00:00	NaN
Cheshta	**30**	10:40:00	NaN
	40	NaN	09:10:00
Deepali	**40**	NaN	10:10:00
	50	09:00:00	NaN
Leena	**50**	NaN	08:40:00
Naman	**40**	10:00:00	NaN
Neha	**30**	NaN	09:30:00
Nikita	**50**	NaN	10:12:00
Pooja	**50**	NaN	09:15:00
Priyanka	**30**	NaN	09:50:00
	50	10:30:00	NaN
Purvi	**40**	09:15:00	NaN
Riya	**40**	10:00:00	NaN
	50	NaN	09:40:00
Seemran	**30**	09:30:00	NaN
Shikha	**50**	08:40:00	NaN
Tamanna	**40**	NaN	11:00:00
Walsha	**30**	10:20:00	NaN

5. 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

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

In [142…
```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
C:\Users\Lenovo\anaconda3\lib\site-packages\scipy\__init__.py:146: UserWarning: A
NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected
version 1.23.1
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [148…
```
iris_df=sns.load_dataset('iris')
iris_df
```

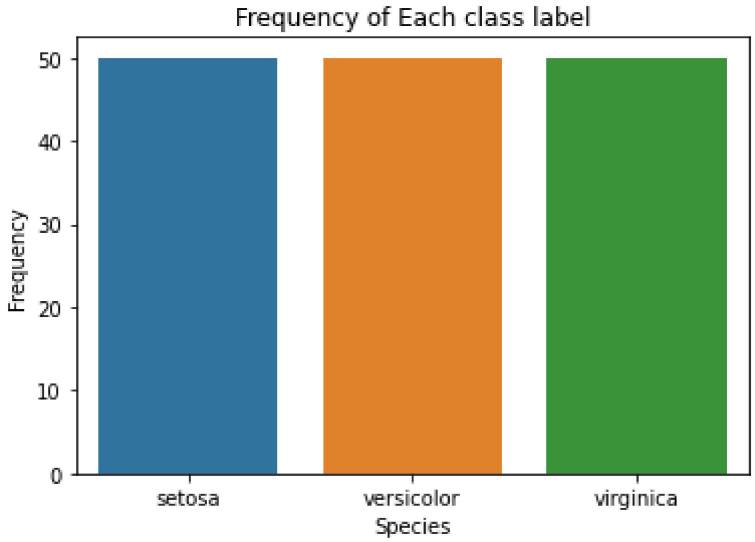
Out[148]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

In [156...

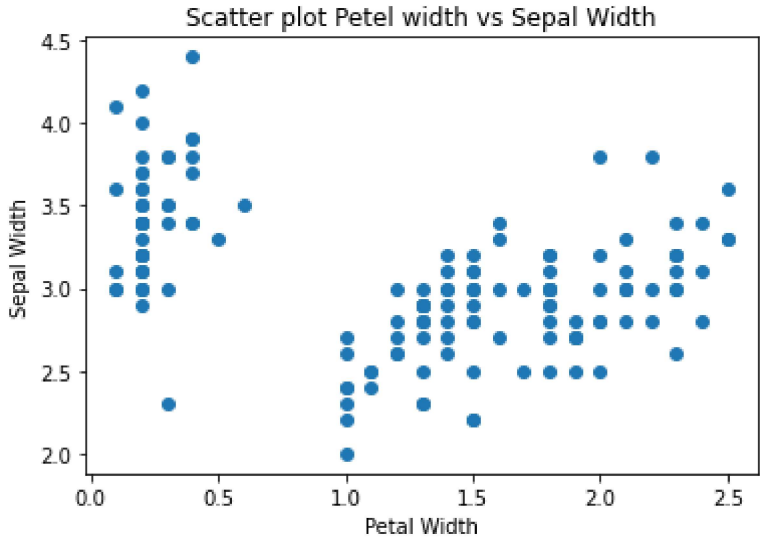
```
sns.countplot(x='species',data=iris_df)
plt.xlabel('Species')
plt.ylabel('Frequency')
plt.title('Frequency of Each class label')
plt.show()
```



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

In [158...

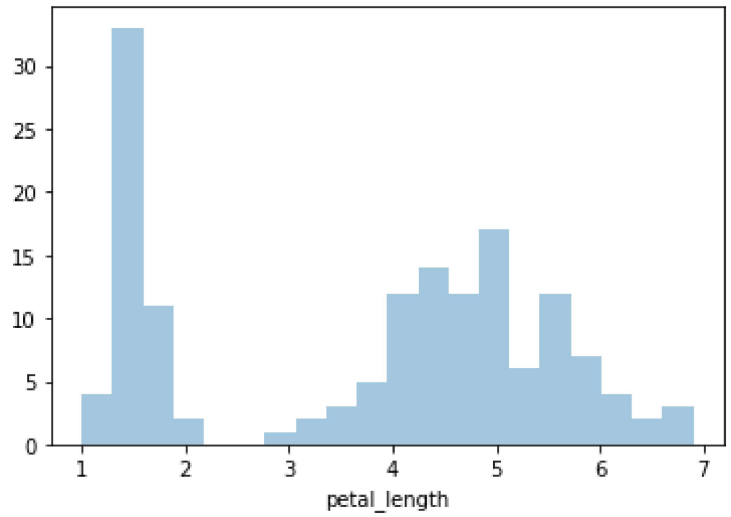
```
plt.scatter(x='petal_width',y='sepal_width',data=iris_df)
plt.xlabel('Petal Width')
plt.ylabel('Sepal Width')
plt.title("Scatter plot Petel width vs Sepal Width")
plt.show()
```



c. Plot density distribution for feature petal length.

In [162…
```
sns.distplot(iris_df['petal_length'],bins=20,kde=False)
```

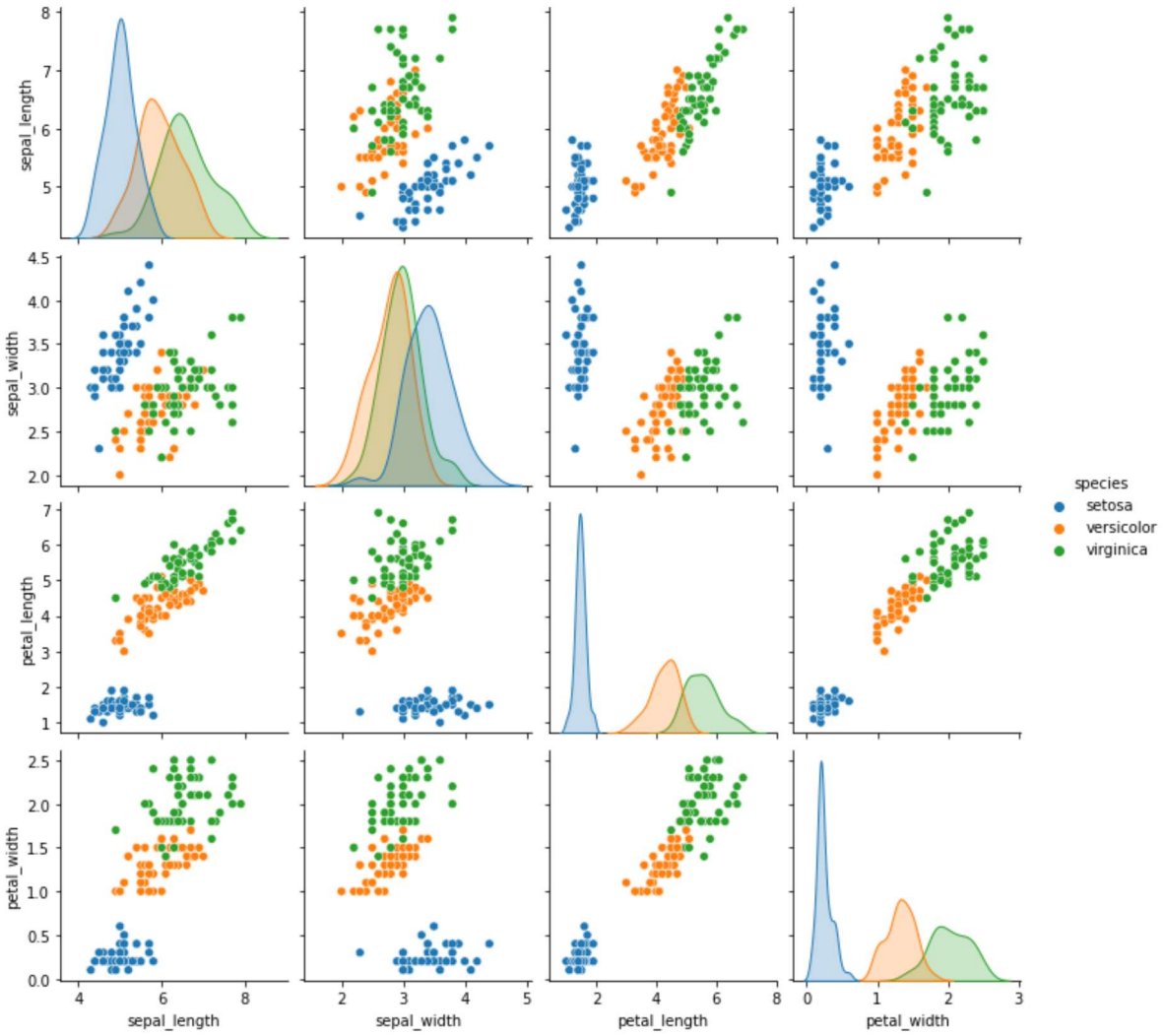
Out[162]:
```
<AxesSubplot:xlabel='petal_length'>
```



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

In [165…
```
sns.pairplot(iris_df,hue='species') #,palette='coolwarm')
```

Out[165]:
```
<seaborn.axisgrid.PairGrid at 0x1620ee30d90>
```



Que 6. Consider any sales training/ weather forecasting dataset

a. Compute mean of a series grouped by another series

In [186…

```
wdf=pd.read_csv("weather.csv")
wdf
```

Out[186]:

	day	city	temperature	windspeed	event
0	01-01-2017	new york	32	6	rain
1	01-02-2017	new york	36	7	sunny
2	01-03-2017	mumbai	28	12	snow
3	01-04-2017	new york	33	7	sunny
4	01-01-2017	mumbai	90	5	sunny
5	01-02-2017	mumbai	85	12	fog
6	01-03-2017	new york	96	15	fog
7	01-04-2017	pune	67	5	rain
8	01-01-2017	pune	45	20	sunny
9	01-02-2017	pune	50	13	cloudy
10	01-03-2017	new york	54	8	cloudy
11	01-04-2017	mumbai	42	10	cloudy

In [187…
```
wdf.groupby('city')['temperature','windspeed'].mean()
```

```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_3844\3933810636.py:1: FutureWarning:
Indexing with multiple keys (implicitly converted to a tuple of keys) will be depr
ecated, use a list instead.
  wdf.groupby('city')['temperature','windspeed'].mean()
```

Out[187]:

city	temperature	windspeed
mumbai	61.25	9.750000
new york	50.20	8.600000
pune	54.00	12.666667

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

In [188…
```
wdf['day'].fillna(method='ffill')
```

Out[188]:
```
0     01-01-2017
1     01-02-2017
2     01-03-2017
3     01-04-2017
4     01-01-2017
5     01-02-2017
6     01-03-2017
7     01-04-2017
8     01-01-2017
9     01-02-2017
10    01-03-2017
11    01-04-2017
Name: day, dtype: object
```

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

In [189…
```
import datetime as dt
```

```
wdf['day']=pd.to_datetime(wdf['day']).dt.strftime('%d-%m-%y')
wdf
```

Out[189]:

	day	city	temperature	windspeed	event
0	01-01-17	new york	32	6	rain
1	02-01-17	new york	36	7	sunny
2	03-01-17	mumbai	28	12	snow
3	04-01-17	new york	33	7	sunny
4	01-01-17	mumbai	90	5	sunny
5	02-01-17	mumbai	85	12	fog
6	03-01-17	new york	96	15	fog
7	04-01-17	pune	67	5	rain
8	01-01-17	pune	45	20	sunny
9	02-01-17	pune	50	13	cloudy
10	03-01-17	new york	54	8	cloudy
11	04-01-17	mumbai	42	10	cloudy

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

In [206…

```
df_agg=wdf.groupby(['event']).agg({'temperature':sum})
result=df_agg['temperature'].groupby(level=0 ,group_keys=False)
print(result.nlargest())
```

```
event
cloudy    146
fog       181
rain       99
snow       28
sunny     204
Name: temperature, dtype: int64
```

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

In [200…

```
e=wdf.groupby(['event',pd.cut(wdf.windspeed,10)])
res=e.size().unstack()
print(res)
```

```
windspeed  (4.985, 6.5]  (6.5, 8.0]  (8.0, 9.5]  (9.5, 11.0]  (11.0, 12.5]  \
event
cloudy                0          1          0           1             0
fog                   0          0          0           0             1
rain                  2          0          0           0             0
snow                  0          0          0           0             1
sunny                 1          2          0           0             0

windspeed  (12.5, 14.0]  (14.0, 15.5]  (15.5, 17.0]  (17.0, 18.5]  \
event
cloudy                1             0             0             0
fog                   0             1             0             0
rain                  0             0             0             0
snow                  0             0             0             0
sunny                 0             0             0             0

windspeed  (18.5, 20.0]
event
cloudy                0
fog                   0
rain                  0
snow                  0
sunny                 1
```

Que 7. Consider a data frame containing data about students i.e. name, gender and passing division:

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

In [209…
```
dataf = [["Mudit Chauhan","December","M","III"], ["Seema Chopra","January","F","II
        ["Rani Gupta","March","F","I"], ["Aditya Narayan","October","M","I"],
        ["Sanjeev Sahni","February","M","II"], ["Prakash Kumar","December","M","II
        ["Ritu Agarwal", "September","F","I"], ["Akshay Goel", "August", "M","I"],
        ["Meeta Kulkarni" ,"July ","F", "II"],  ["Preeti Ahuja ","November", "F",
        ["Sunil Das Gupta ","April", "M", "III"], ["Sonali Sapre ","January","F","
        ["Rashmi Talwar", "June ","F" ,"III"],  ["Ashish Dubey" ,"May" ,"M", "II"]
        ["Kiran Sharma", "February","F", "II"], ["Sameer Bansal", "October","M", "
```

In [210…
```
stu=pd.DataFrame(data=dataf,columns=['Name','Birth_Month','Gender','Pass_Division'
stu
```

Out[210]:

	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

In [213…

```
gender=pd.get_dummies(stu['Gender'],drop_first=True)
gender
```

Out[213]:

	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

In [215…
```
div=pd.get_dummies(stu['Pass_Division'])
div
```

Out[215]:

	I	II	III
0	0	0	1
1	0	1	0
2	1	0	0
3	1	0	0
4	0	1	0
5	0	0	1
6	1	0	0
7	1	0	0
8	0	1	0
9	0	1	0
10	0	0	1
11	1	0	0
12	0	0	1
13	0	1	0
14	0	1	0
15	1	0	0

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

In [221…
```
months = ["January", "February", "March", "April", "May", "June", "July", "August"
          "November", "December"]
stu['Birth_Month']=pd.Categorical(stu['Birth_Month'],categories=months,ordered=True
stu.sort_values(by='Birth_Month',inplace=True)
stu
```

Out[221]:

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

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

a. Calculate and display familywise gross monthly income

In [230…

```
dataf = [["Shah","Male",114000.00],["Vats"," Male",65000.00],["Vats","Female",4315(
        ["Vats", "Female", 155000.00],["Kumar","Male", 103000.00],["Shah", "Male"
        ["Shah", "Female", 112400.00],["Kumar","Female",81030.00],["Vats", "Male",
```

In [246…

```
family=pd.DataFrame(data=dataf,columns=['Name','Gender','Monthly Income'])
family
```

Out[246]:

	Name	Gender	Monthly Income
0	Shah	Male	114000.0
1	Vats	Male	65000.0
2	Vats	Female	43150.0
3	Kumar	Female	69500.0
4	Vats	Female	155000.0
5	Kumar	Male	103000.0
6	Shah	Male	55000.0
7	Shah	Female	112400.0
8	Kumar	Female	81030.0
9	Vats	Male	55000.0

In [236…
```
fm=family.groupby('Name')
fm.sum()
```

Out[236]:

Name	Monthly Income
Kumar	253530.0
Shah	281400.0
Vats	318150.0

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

In [248…
```
fm.max()
```

Out[248]:

Name	Gender	Monthly Income
Kumar	Male	103000.0
Shah	Male	114000.0
Vats	Male	155000.0

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

In [251…
```
familyy=family[family['Monthly Income']>60000]
familyy
```

Out[251]:

	Name	Gender	Monthly Income
0	Shah	Male	114000.0
1	Vats	Male	65000.0
3	Kumar	Female	69500.0
4	Vats	Female	155000.0
5	Kumar	Male	103000.0
7	Shah	Female	112400.0
8	Kumar	Female	81030.0

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

In [258…
```
xf=family[(family["Gender"]=="Female") &(family["Name"]=="Shah")]["Monthly Income"
```

In [262…
```
print("Average monthly Income = ",xf)
```

```
Average monthly Income =  112400.0
```

or

In [263…
```
family[family['Gender']=='Female'].groupby('Name').mean().loc['Shah']
```

Out[263]:
```
Monthly Income    112400.0
Name: Shah, dtype: float64
```

In [ ]:
```
```