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]

```
class_d = {'Boys': [72, 68, 70, 69, 74], 'Girls': [63, 65, 69, 62,
61]}
ans = [{'Boys':class_d['Boys'][i] ,'Girls':class_d['Girls'][i] } for i
in range(5)]
ans
```

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

**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. 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.**

```
# 1
import numpy as np
arr2d = np.random.rand(4,3)
print("Array is :\n",arr2d)
print(np.mean(arr2d,axis=1),np.std(arr2d,axis=1),np.var(arr2d,axis=1))
```

```
Array is :
 [[0.27785979 0.04210814 0.45758546]
 [0.94705286 0.46164633 0.7567715 ]
 [0.43191598 0.78484674 0.15735818]
 [0.92062454 0.39773658 0.27660555]]
[0.25918446 0.72182357 0.4580403  0.53165556] [0.17013118 0.19970127
0.25683631 0.27945284] [0.02894462 0.0398806  0.06596489 0.07809389]
```

```
# 2
B = [56, 48, 22, 41, 78, 91, 24, 46, 8, 33]
sorted = np.argsort(B)
sorted
```

```
array([8, 2, 6, 9, 3, 7, 1, 0, 4, 5], dtype=int64)
```

```
#3
m = int(input("Enter the dimnesion1: "))
n = int(input("Enter the dimnesion2: "))
arr = np.ones((m,n))
print("Original Array:")
print("Shape:", arr.shape)
print("Type:", type(arr))
print("Data Type:", arr.dtype)

arr_reshaped = arr.reshape((n, m))

print("\nReshaped Array:")
print("Shape:", arr_reshaped.shape)
print("Type:", type(arr_reshaped))
print("Data Type:", arr_reshaped.dtype)
```

```
Enter the dimnesion1:  2
Enter the dimnesion2:  3

Original Array:
Shape: (2, 3)
Type: <class 'numpy.ndarray'>
Data Type: float64

Reshaped Array:
Shape: (3, 2)
Type: <class 'numpy.ndarray'>
Data Type: float64
```

```
#4
array = np.array([1,2,3,0,45,0,np.NaN,34,np.NaN,0,90])
indices = np.arange(array.size)
zero_array = np.where(array ==0)
print(zero_array)
NaN_array = np.where(np.isnan(array))
print(NaN_array)
Nonzero_array = np.where((array!=0) & (~np.isnan(array)))
print(Nonzero_array)
```

```
(array([3, 5, 9], dtype=int64),)
(array([6, 8], dtype=int64),)
(array([ 0,  1,  2,  4,  7, 10], dtype=int64),)
```

**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:

a. Identify and count missing values in a dataframe.

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

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

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

e. Remove all duplicates from the first column.

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

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

h. Discretize second column and create 5 bins**

```
import pandas as pd
import numpy as np
np.random.seed(55)
data = pd.DataFrame(np.random.randn(50, 3), columns=['A', 'B', 'C'])
null_indices = np.random.choice(50 * 3, size=int(0.1 * 50 * 3))
data.values.ravel()[null_indices]=np.nan
print(data)
```

```
           A         B         C
0  -1.623731 -0.101784 -1.809791
1   0.262654  0.259953 -0.381086
2  -0.002290  0.341615  0.897572
3  -0.361100  1.656445 -1.189009
4        NaN -2.003439 -0.477873
5   1.368799  0.258169  0.702352
6        NaN  0.722220 -1.382103
7  -0.993455       NaN  0.287980
8   0.297058 -0.187577 -0.368095
9   1.450779  0.219189  0.445122
10 -0.210605 -0.506903  0.400062
11  1.017513  1.210956       NaN
12 -1.355586  0.795226 -1.571194
13 -1.159787 -0.642639 -0.803192
14  0.898589  0.445477  0.628428
15 -0.688902  0.149899  1.264266
16 -1.214419       NaN -0.567562
17 -1.004690  0.197138  0.726756
18  0.275635  0.141798  1.552575
19 -0.728043 -0.087340  1.298419
20       NaN -1.580594  2.426968
21  0.988937  0.736140 -0.821571
22 -0.682896  1.239457  0.300079
23  0.059409  0.866403 -1.453462
24  0.526640 -0.225195  1.130765
25 -0.110284 -0.576672       NaN
26  0.581236 -1.658025  1.639296
```

```
27 -0.057860       NaN -1.773641
28 -0.910098  3.258409  0.126727
29  0.522383 -1.501184 -2.006004
30 -0.824556  1.001420  1.026229
31 -0.952833 -0.579100  0.005545
32 -2.350246  0.193312       NaN
33  0.955015  0.175959       NaN
34  0.186711       NaN -1.223367
35 -0.521896 -0.405500  2.415546
36 -0.690814  0.004434 -0.240718
37 -0.915992  0.348185 -1.383135
38 -0.609456  0.061805  0.324368
39  1.066725  0.903609 -1.820567
40       NaN -0.361377  4.787365
41  1.804814 -0.756736 -0.186132
42  0.440522  2.194025  0.622404
43 -0.496039  0.154343       NaN
44 -0.020723  0.694797  1.001639
45  0.594228 -1.161164 -0.853352
46 -0.169719  0.069562       NaN
47       NaN -0.620683 -0.619584
48 -0.891088  0.170076 -0.915324
49  0.542973  0.138465  0.509312
```

```
#a
missing_values = data.isnull().sum().sum()
print("Missing values count: ",missing_values)
```

```
Missing values count:  15
```

```
#b
data1=data.dropna(axis=1,thresh=len(data)-5)
data1
```

```
           A         B
0  -1.623731 -0.101784
1   0.262654  0.259953
2  -0.002290  0.341615
3  -0.361100  1.656445
4        NaN -2.003439
5   1.368799  0.258169
6        NaN  0.722220
7  -0.993455       NaN
8   0.297058 -0.187577
9   1.450779  0.219189
10 -0.210605 -0.506903
11  1.017513  1.210956
12 -1.355586  0.795226
13 -1.159787 -0.642639
14  0.898589  0.445477
```

```
15 -0.688902  0.149899
16 -1.214419       NaN
17 -1.004690  0.197138
18  0.275635  0.141798
19 -0.728043 -0.087340
20       NaN -1.580594
21  0.988937  0.736140
22 -0.682896  1.239457
23  0.059409  0.866403
24  0.526640 -0.225195
25 -0.110284 -0.576672
26  0.581236 -1.658025
27 -0.057860       NaN
28 -0.910098  3.258409
29  0.522383 -1.501184
30 -0.824556  1.001420
31 -0.952833 -0.579100
32 -2.350246  0.193312
33  0.955015  0.175959
34  0.186711       NaN
35 -0.521896 -0.405500
36 -0.690814  0.004434
37 -0.915992  0.348185
38 -0.609456  0.061805
39  1.066725  0.903609
40       NaN -0.361377
41  1.804814 -0.756736
42  0.440522  2.194025
43 -0.496039  0.154343
44 -0.020723  0.694797
45  0.594228 -1.161164
46 -0.169719  0.069562
47       NaN -0.620683
48 -0.891088  0.170076
49  0.542973  0.138465
```

```
#c
ind=data.sum(axis=1).idxmax()
data4=data.drop(ind)
data4
```

```
           A         B         C
0  -1.623731 -0.101784 -1.809791
1   0.262654  0.259953 -0.381086
2  -0.002290  0.341615  0.897572
3  -0.361100  1.656445 -1.189009
4        NaN -2.003439 -0.477873
5   1.368799  0.258169  0.702352
6        NaN  0.722220 -1.382103
7  -0.993455       NaN  0.287980
```

```
8   0.297058 -0.187577 -0.368095
9   1.450779  0.219189  0.445122
10 -0.210605 -0.506903  0.400062
11  1.017513  1.210956       NaN
12 -1.355586  0.795226 -1.571194
13 -1.159787 -0.642639 -0.803192
14  0.898589  0.445477  0.628428
15 -0.688902  0.149899  1.264266
16 -1.214419       NaN -0.567562
17 -1.004690  0.197138  0.726756
18  0.275635  0.141798  1.552575
19 -0.728043 -0.087340  1.298419
20       NaN -1.580594  2.426968
21  0.988937  0.736140 -0.821571
22 -0.682896  1.239457  0.300079
23  0.059409  0.866403 -1.453462
24  0.526640 -0.225195  1.130765
25 -0.110284 -0.576672       NaN
26  0.581236 -1.658025  1.639296
27 -0.057860       NaN -1.773641
28 -0.910098  3.258409  0.126727
29  0.522383 -1.501184 -2.006004
30 -0.824556  1.001420  1.026229
31 -0.952833 -0.579100  0.005545
32 -2.350246  0.193312       NaN
33  0.955015  0.175959       NaN
34  0.186711       NaN -1.223367
35 -0.521896 -0.405500  2.415546
36 -0.690814  0.004434 -0.240718
37 -0.915992  0.348185 -1.383135
38 -0.609456  0.061805  0.324368
39  1.066725  0.903609 -1.820567
41  1.804814 -0.756736 -0.186132
42  0.440522  2.194025  0.622404
43 -0.496039  0.154343       NaN
44 -0.020723  0.694797  1.001639
45  0.594228 -1.161164 -0.853352
46 -0.169719  0.069562       NaN
47       NaN -0.620683 -0.619584
48 -0.891088  0.170076 -0.915324
49  0.542973  0.138465  0.509312
```

```
# d. Sorting the dataframe based on the first column
data3 = data.sort_values(by='A')
data3
```

```
           A         B         C
32 -2.350246  0.193312       NaN
0  -1.623731 -0.101784 -1.809791
12 -1.355586  0.795226 -1.571194
```

```
16 -1.214419       NaN -0.567562
13 -1.159787 -0.642639 -0.803192
17 -1.004690  0.197138  0.726756
7  -0.993455       NaN  0.287980
31 -0.952833 -0.579100  0.005545
37 -0.915992  0.348185 -1.383135
28 -0.910098  3.258409  0.126727
48 -0.891088  0.170076 -0.915324
30 -0.824556  1.001420  1.026229
19 -0.728043 -0.087340  1.298419
36 -0.690814  0.004434 -0.240718
15 -0.688902  0.149899  1.264266
22 -0.682896  1.239457  0.300079
38 -0.609456  0.061805  0.324368
35 -0.521896 -0.405500  2.415546
43 -0.496039  0.154343       NaN
3  -0.361100  1.656445 -1.189009
10 -0.210605 -0.506903  0.400062
46 -0.169719  0.069562       NaN
25 -0.110284 -0.576672       NaN
27 -0.057860       NaN -1.773641
44 -0.020723  0.694797  1.001639
2  -0.002290  0.341615  0.897572
23  0.059409  0.866403 -1.453462
34  0.186711       NaN -1.223367
1   0.262654  0.259953 -0.381086
18  0.275635  0.141798  1.552575
8   0.297058 -0.187577 -0.368095
42  0.440522  2.194025  0.622404
29  0.522383 -1.501184 -2.006004
24  0.526640 -0.225195  1.130765
49  0.542973  0.138465  0.509312
26  0.581236 -1.658025  1.639296
45  0.594228 -1.161164 -0.853352
14  0.898589  0.445477  0.628428
33  0.955015  0.175959       NaN
21  0.988937  0.736140 -0.821571
11  1.017513  1.210956       NaN
39  1.066725  0.903609 -1.820567
5   1.368799  0.258169  0.702352
9   1.450779  0.219189  0.445122
41  1.804814 -0.756736 -0.186132
4        NaN -2.003439 -0.477873
6        NaN  0.722220 -1.382103
20       NaN -1.580594  2.426968
40       NaN -0.361377  4.787365
47       NaN -0.620683 -0.619584
```

```
# e. Removing duplicates from the first column
data2= data.drop_duplicates(['A'])
data2
```

	A	B	C
0	-1.623731	-0.101784	-1.809791
1	0.262654	0.259953	-0.381086
2	-0.002290	0.341615	0.897572
3	-0.361100	1.656445	-1.189009
4	NaN	-2.003439	-0.477873
5	1.368799	0.258169	0.702352
7	-0.993455	NaN	0.287980
8	0.297058	-0.187577	-0.368095
9	1.450779	0.219189	0.445122
10	-0.210605	-0.506903	0.400062
11	1.017513	1.210956	NaN
12	-1.355586	0.795226	-1.571194
13	-1.159787	-0.642639	-0.803192
14	0.898589	0.445477	0.628428
15	-0.688902	0.149899	1.264266
16	-1.214419	NaN	-0.567562
17	-1.004690	0.197138	0.726756
18	0.275635	0.141798	1.552575
19	-0.728043	-0.087340	1.298419
21	0.988937	0.736140	-0.821571
22	-0.682896	1.239457	0.300079
23	0.059409	0.866403	-1.453462
24	0.526640	-0.225195	1.130765
25	-0.110284	-0.576672	NaN
26	0.581236	-1.658025	1.639296
27	-0.057860	NaN	-1.773641
28	-0.910098	3.258409	0.126727
29	0.522383	-1.501184	-2.006004
30	-0.824556	1.001420	1.026229
31	-0.952833	-0.579100	0.005545
32	-2.350246	0.193312	NaN
33	0.955015	0.175959	NaN
34	0.186711	NaN	-1.223367
35	-0.521896	-0.405500	2.415546
36	-0.690814	0.004434	-0.240718
37	-0.915992	0.348185	-1.383135
38	-0.609456	0.061805	0.324368
39	1.066725	0.903609	-1.820567
41	1.804814	-0.756736	-0.186132
42	0.440522	2.194025	0.622404
43	-0.496039	0.154343	NaN
44	-0.020723	0.694797	1.001639
45	0.594228	-1.161164	-0.853352
46	-0.169719	0.069562	NaN

```
48 -0.891088  0.170076 -0.915324
49  0.542973  0.138465  0.509312
```

```
# f. Finding correlation and covariance
correlation = data['A'].corr(data['B'])
covariance = data['B'].cov(data['C'])
print("Correltaion is: ",correlation,"and Covariance is: ",covariance)
```

```
Correltaion is:  -0.10791218516975758 and Covariance is:  -
0.1877444595687125
```

```
#g
data=data[~(np.abs(data) > 3).any(axis=1)]
data
```

```
           A         B         C
0  -1.623731 -0.101784 -1.809791
1   0.262654  0.259953 -0.381086
2  -0.002290  0.341615  0.897572
3  -0.361100  1.656445 -1.189009
4        NaN -2.003439 -0.477873
5   1.368799  0.258169  0.702352
6        NaN  0.722220 -1.382103
7  -0.993455       NaN  0.287980
8   0.297058 -0.187577 -0.368095
9   1.450779  0.219189  0.445122
10 -0.210605 -0.506903  0.400062
11  1.017513  1.210956       NaN
12 -1.355586  0.795226 -1.571194
13 -1.159787 -0.642639 -0.803192
14  0.898589  0.445477  0.628428
15 -0.688902  0.149899  1.264266
16 -1.214419       NaN -0.567562
17 -1.004690  0.197138  0.726756
18  0.275635  0.141798  1.552575
19 -0.728043 -0.087340  1.298419
20       NaN -1.580594  2.426968
21  0.988937  0.736140 -0.821571
22 -0.682896  1.239457  0.300079
23  0.059409  0.866403 -1.453462
24  0.526640 -0.225195  1.130765
25 -0.110284 -0.576672       NaN
26  0.581236 -1.658025  1.639296
27 -0.057860       NaN -1.773641
29  0.522383 -1.501184 -2.006004
30 -0.824556  1.001420  1.026229
31 -0.952833 -0.579100  0.005545
32 -2.350246  0.193312       NaN
33  0.955015  0.175959       NaN
34  0.186711       NaN -1.223367
```

```
35 -0.521896 -0.405500  2.415546
36 -0.690814  0.004434 -0.240718
37 -0.915992  0.348185 -1.383135
38 -0.609456  0.061805  0.324368
39  1.066725  0.903609 -1.820567
41  1.804814 -0.756736 -0.186132
42  0.440522  2.194025  0.622404
43 -0.496039  0.154343       NaN
44 -0.020723  0.694797  1.001639
45  0.594228 -1.161164 -0.853352
46 -0.169719  0.069562       NaN
47       NaN -0.620683 -0.619584
48 -0.891088  0.170076 -0.915324
49  0.542973  0.138465  0.509312
```

```
# h. Discretizing the second column into 5 bins
data = data.dropna()
# data['B_grps'] = pd.cut(data['B'],5)
data.loc[:, 'B_grps'] = pd.cut(data['B'], 5)
data
```

```
           A         B         C             B_grps
0  -1.623731 -0.101784 -1.809791   (-0.675, 0.309]
1   0.262654  0.259953 -0.381086   (-0.675, 0.309]
2  -0.002290  0.341615  0.897572    (0.309, 1.292]
3  -0.361100  1.656445 -1.189009    (1.292, 2.275]
5   1.368799  0.258169  0.702352   (-0.675, 0.309]
8   0.297058 -0.187577 -0.368095   (-0.675, 0.309]
9   1.450779  0.219189  0.445122   (-0.675, 0.309]
10 -0.210605 -0.506903  0.400062   (-0.675, 0.309]
12 -1.355586  0.795226 -1.571194    (0.309, 1.292]
13 -1.159787 -0.642639 -0.803192   (-0.675, 0.309]
14  0.898589  0.445477  0.628428    (0.309, 1.292]
15 -0.688902  0.149899  1.264266   (-0.675, 0.309]
17 -1.004690  0.197138  0.726756   (-0.675, 0.309]
18  0.275635  0.141798  1.552575   (-0.675, 0.309]
19 -0.728043 -0.087340  1.298419   (-0.675, 0.309]
21  0.988937  0.736140 -0.821571    (0.309, 1.292]
22 -0.682896  1.239457  0.300079    (0.309, 1.292]
23  0.059409  0.866403 -1.453462    (0.309, 1.292]
24  0.526640 -0.225195  1.130765   (-0.675, 0.309]
26  0.581236 -1.658025  1.639296  (-1.663, -0.675]
28 -0.910098  3.258409  0.126727    (2.275, 3.258]
29  0.522383 -1.501184 -2.006004  (-1.663, -0.675]
30 -0.824556  1.001420  1.026229    (0.309, 1.292]
31 -0.952833 -0.579100  0.005545   (-0.675, 0.309]
35 -0.521896 -0.405500  2.415546   (-0.675, 0.309]
36 -0.690814  0.004434 -0.240718   (-0.675, 0.309]
37 -0.915992  0.348185 -1.383135    (0.309, 1.292]
38 -0.609456  0.061805  0.324368   (-0.675, 0.309]
```

```
39  1.066725  0.903609 -1.820567    (0.309, 1.292]
41  1.804814 -0.756736 -0.186132  (-1.663, -0.675]
42  0.440522  2.194025  0.622404    (1.292, 2.275]
44 -0.020723  0.694797  1.001639    (0.309, 1.292]
45  0.594228 -1.161164 -0.853352  (-1.663, -0.675]
48 -0.891088  0.170076 -0.915324   (-0.675, 0.309]
49  0.542973  0.138465  0.509312   (-0.675, 0.309]
```

**4. Consider two excel files having attendance of a workshops 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:

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
df_day1 = pd.read_excel('Book3.xlsx')
df_day2 = pd.read_excel('Book4.xlsx')
df_day1,df_day2
```

```
(    Name  Time of joining  Duration
 0  Alice               11        50
 1    Bob               11        30
 2   Chef               11        40
 3  Carey               12        30
 4  David               10        50
 5  Anita               10        30
 6  Komal               12        40,
          Name  Time of joining  Duration
 0       Alice               10        40
 1       Carey               11        30
 2       David               10        50
 3       Anita               12        30
 4       komal               10        50
 5          Om               12        40
 6  Christoper               11        30)
```

```
#a
both_days_att= pd.merge(df_day1, df_day2, on='Name')
both_days_att
```

```
    Name  Time of joining_x  Duration_x  Time of joining_y  Duration_y
0  Alice                 11          50                 10          40
```

```
1  Carey                  12          30                  11          30
2  David                  10          50                  10          50
3  Anita                  10          30                  12          30
```

```
#b
either_day_att = pd.merge(df_day1, df_day2, on='Name', how='outer')
either_day_att
```

```
         Name  Time of joining_x  Duration_x  Time of joining_y
Duration_y
0       Alice               11.0        50.0               10.0
40.0
1         Bob               11.0        30.0                NaN
NaN
2        Chef               11.0        40.0                NaN
NaN
3       Carey               12.0        30.0               11.0
30.0
4       David               10.0        50.0               10.0
50.0
5       Anita               10.0        30.0               12.0
30.0
6       Komal               12.0        40.0                NaN
NaN
7       komal                NaN         NaN               10.0
50.0
8          Om                NaN         NaN               12.0
40.0
9  Christoper                NaN         NaN               11.0
30.0
```

```
#c
merging_row = pd.concat([df_day1, df_day2], axis=0)
total_records = len(merging_row)
total_records
```

```
14
```

```
#d
multi_merge = pd.merge(df_day1, df_day2, on=['Name', 'Duration'])
stats= multi_merge.groupby(['Name', 'Duration']).describe()
stats
```

```
                Time of joining_x
\
                           count  mean std   min   25%   50%   75%
max
Name  Duration

Anita 30                     1.0  10.0 NaN  10.0  10.0  10.0  10.0
10.0
```

```
Carey 30                      1.0  12.0 NaN  12.0  12.0  12.0  12.0
12.0
David 50                      1.0  10.0 NaN  10.0  10.0  10.0  10.0
10.0

               Time of joining_y

                            count  mean std   min   25%   50%   75%
max
Name  Duration

Anita 30                      1.0  12.0 NaN  12.0  12.0  12.0  12.0
12.0
Carey 30                      1.0  11.0 NaN  11.0  11.0  11.0  11.0
11.0
David 50                      1.0  10.0 NaN  10.0  10.0  10.0  10.0
10.0
```

**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.

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

c. Plot density distribution for feature petal length.

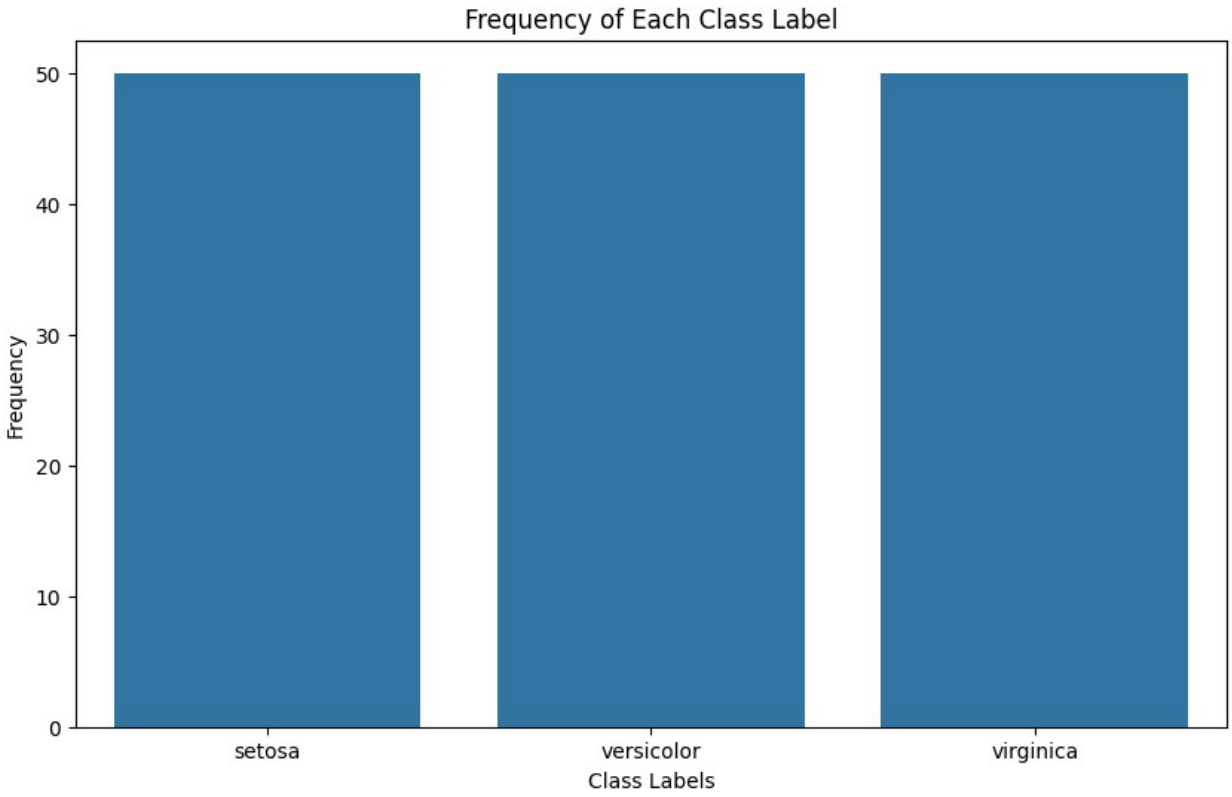
d. Use a pair plot to show pairwise bivariate distribution in the Iris Database**

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
iris = datasets.load_iris()
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['target'] = iris.target
target_names = {i: name for i, name in enumerate(iris.target_names)}
iris_df['target_names'] = iris_df['target'].map(target_names)
iris_df
```

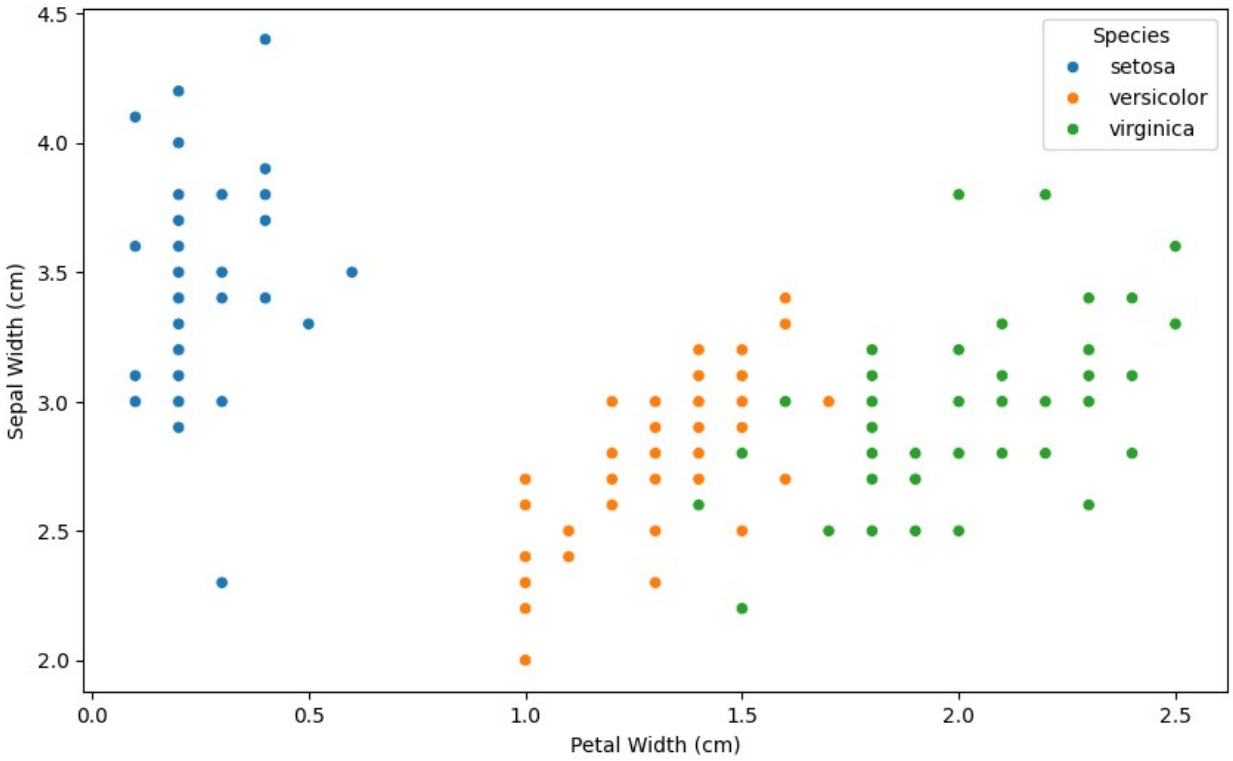
```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal
width (cm)  \
0                  5.1               3.5                1.4
0.2
1                  4.9               3.0                1.4
0.2
2                  4.7               3.2                1.3
0.2
3                  4.6               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

     target target_names
0         0       setosa
1         0       setosa
2         0       setosa
3         0       setosa
4         0       setosa
..      ...          ...
145       2    virginica
146       2    virginica
147       2    virginica
148       2    virginica
149       2    virginica

[150 rows x 6 columns]
```

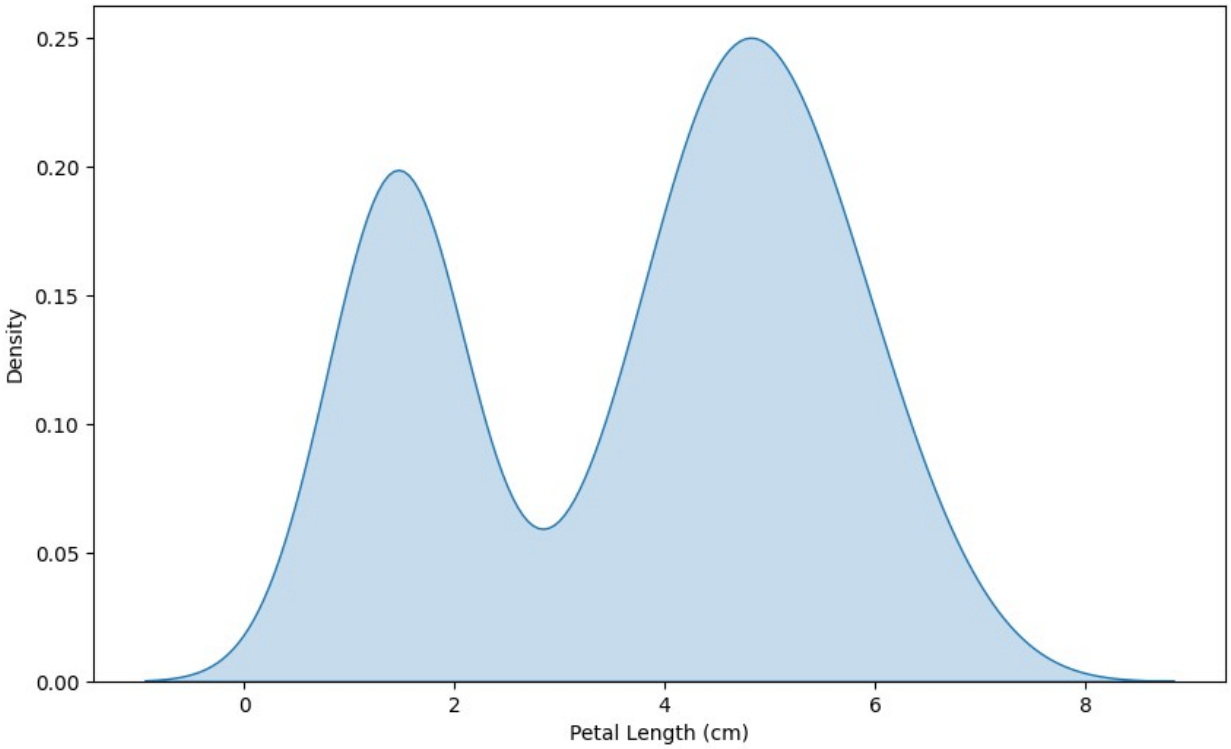
```
#a
plt.figure(figsize=(10, 6))
sns.countplot(x='target_names', data=iris_df)
plt.xlabel('Class Labels')
plt.ylabel('Frequency')
plt.title('Frequency of Each Class Label')
plt.show()
```



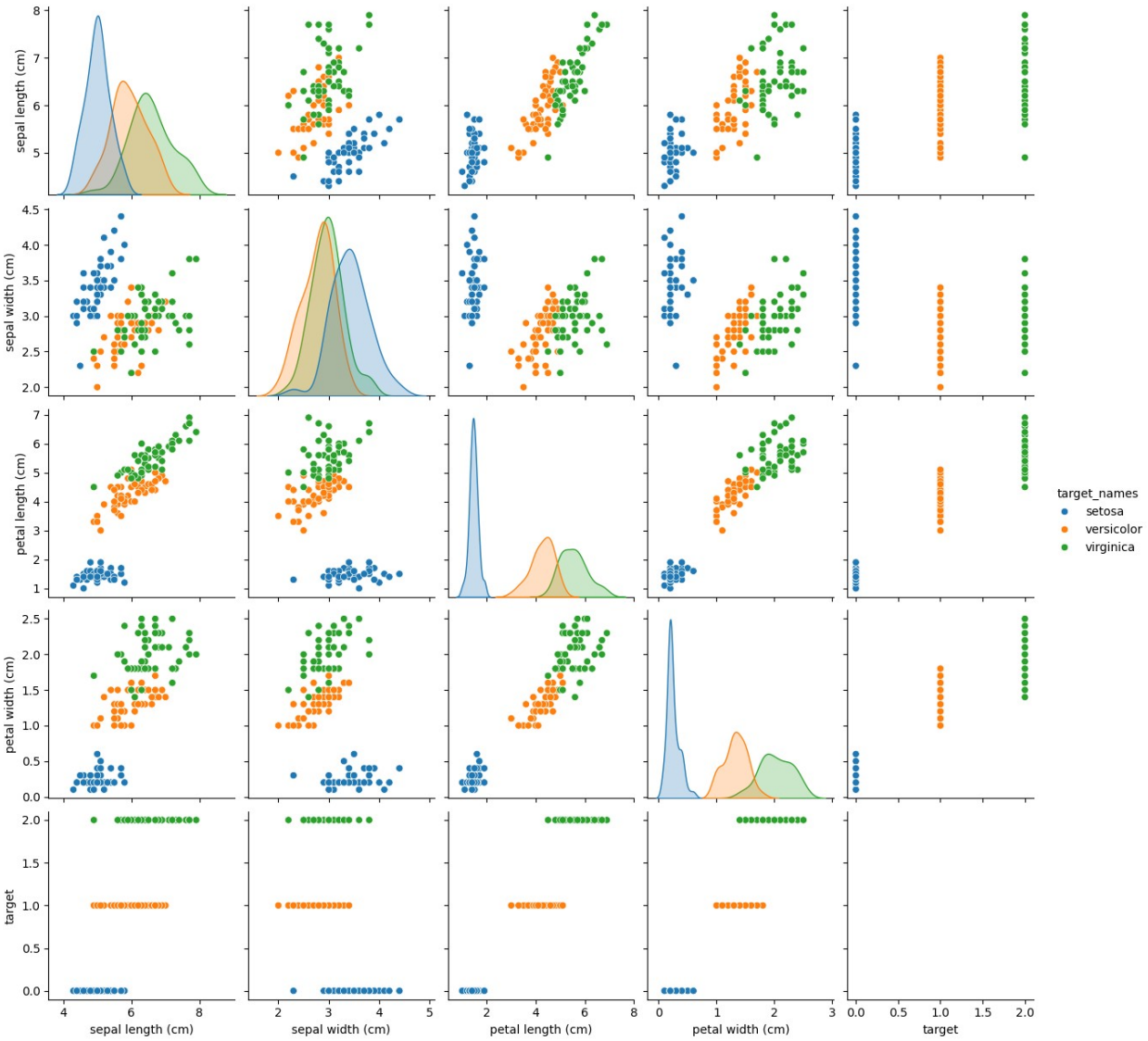

```
#b
plt.figure(figsize=(10, 6))
sns.scatterplot(x='petal width (cm)', y='sepal width (cm)',
hue='target_names', data=iris_df)
plt.xlabel('Petal Width (cm)')
plt.ylabel('Sepal Width (cm)')
plt.legend(title='Species')
plt.show()
```




```
#c
plt.figure(figsize=(10, 6))
sns.kdeplot(iris_df['petal length (cm)'],fill=True)
plt.xlabel('Petal Length (cm)')
plt.ylabel('Density')
plt.show()
```




```
#d
sns.pairplot(iris_df, hue='target_names')
plt.show()
```



**6. Consider any sales training/ weather forecasting dataset

a. Compute mean of a series grouped by another series

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

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

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

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

```
data = {
    'Date': ['2023-01-01', '2023-01-02', '2023-01-04', '2023-01-06',
'2023-01-07'],
```

```
    'Sales': [100, 150, 200, 180, 220],
    'Region': ['North', 'South', 'East', 'West', 'North'],
    'Product': ['A', 'B', 'A', 'C', 'B']
}
# Convert 'Date' column to datetime format
sales_df = pd.DataFrame(data)
sales_df['Date'] = pd.to_datetime(sales_df['Date'])
sales_df
```

```
        Date  Sales Region Product
0 2023-01-01    100  North       A
1 2023-01-02    150  South       B
2 2023-01-04    200   East       A
3 2023-01-06    180   West       C
4 2023-01-07    220  North       B
```

```
#a
mean_sales = sales_df.groupby('Region')['Sales'].mean()
print(mean_sales)
```

```
Region
East     200.0
North    160.0
South    150.0
West     180.0
Name: Sales, dtype: float64
```

```
#b
sales_df1 = sales_df.set_index('Date').asfreq('D', method='ffill')
sales_df1
```

```
            Sales Region Product
Date
2023-01-01    100  North       A
2023-01-02    150  South       B
2023-01-03    150  South       B
2023-01-04    200   East       A
2023-01-05    200   East       A
2023-01-06    180   West       C
2023-01-07    220  North       B
```

```
#c
sales_df['year_month']=['2023-08','2023-09','2023-10','2023-11','2023-
07']
sales_df['year_month'] = pd.to_datetime(sales_df['year_month'])
sales_df
```

```
        Date  Sales Region Product year_month
0 2023-01-01    100  North       A 2023-08-01
1 2023-01-02    150  South       B 2023-09-01
2 2023-01-04    200   East       A 2023-10-01
```

```
3 2023-01-06    180   West       C 2023-11-01
4 2023-01-07    220  North       B 2023-07-01
```

```
#d
sorted = sales_df.groupby(['Region', 'Product'])
['Sales'].sum().sort_values()
print(sorted)
```

```
Region  Product
North   A          100
South   B          150
West    C          180
East    A          200
North   B          220
Name: Sales, dtype: int64
```

```
#e
sales_df['Bins'] = pd.cut(sales_df['Sales'], bins=4)
sales_groups = sales_df.groupby('Bins',observed=False)
for key, group in sales_groups:
    print(key)
    print(group)
```

```
(99.88, 130.0]
        Date  Sales Region Product            Bins
0 2023-01-01    100  North       A  (99.88, 130.0]
(130.0, 160.0]
        Date  Sales Region Product            Bins
1 2023-01-02    150  South       B  (130.0, 160.0]
(160.0, 190.0]
        Date  Sales Region Product            Bins
3 2023-01-06    180   West       C  (160.0, 190.0]
(190.0, 220.0]
        Date  Sales Region Product            Bins
2 2023-01-04    200   East       A  (190.0, 220.0]
4 2023-01-07    220  North       B  (190.0, 220.0]
```

**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.

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

```
data = {
    'Name': ['Mudit Chauhan', 'Seema Chopra', 'Rani Gupta', 'Aditya
Narayan', 'Sanjeev Sahni',
             'Prakash Kumar', 'Ritu Agarwal', 'Akshay Goel', 'Meeta
```

```
Kulkarni', 'Preeti Ahuja',
             'Sunil Das Gupta', 'Sonali Sapre', 'Rashmi Talwar',
'Ashish Dubey', 'Kiran Sharma',
             'Sameer Bansal'],
    'Birth_Month': ['December', 'January', 'March', 'October',
'February', 'December', 'September',
                    'August', 'July', 'November', 'April', 'January',
'June', 'May', 'February',
                    'October'],
    'Gender': ['M', 'F', 'F', 'M', 'M', 'M', 'F', 'M', 'F', 'F', 'M',
'F', 'F', 'M', 'F', 'M'],
    'Pass_Division': ['III', 'II', 'I', 'I', 'II', 'III', 'I', 'I',
'II', 'II', 'III', 'I', 'III',
                      'II', 'II', 'I']
}
df = pd.DataFrame(data)

#a
one_hot = pd.get_dummies(df[['Gender', 'Pass_Division']])
df_encode = pd.concat([df, one_hot], axis=1)
print(df_encode)
```

```
               Name Birth_Month Gender Pass_Division  Gender_F
Gender_M  \
0     Mudit Chauhan    December      M           III     False
True
1      Seema Chopra     January      F            II      True
False
2        Rani Gupta       March      F             I      True
False
3    Aditya Narayan     October      M             I     False
True
4     Sanjeev Sahni    February      M            II     False
True
5     Prakash Kumar    December      M           III     False
True
6      Ritu Agarwal   September      F             I      True
False
7       Akshay Goel      August      M             I     False
True
8    Meeta Kulkarni        July      F            II      True
False
9      Preeti Ahuja    November      F            II      True
False
10  Sunil Das Gupta       April      M           III     False
True
11     Sonali Sapre     January      F             I      True
False
12    Rashmi Talwar        June      F           III      True
False
```

```
13     Ashish Dubey          May      M             II     False
True
14     Kiran Sharma    February       F             II      True
False
15    Sameer Bansal     October       M              I     False
True
```

```
    Pass_Division_I  Pass_Division_II  Pass_Division_III
0             False             False               True
1             False              True              False
2              True             False              False
3              True             False              False
4             False              True              False
5             False             False               True
6              True             False              False
7              True             False              False
8             False              True              False
9             False              True              False
10            False             False               True
11             True             False              False
12            False             False               True
13            False              True              False
14            False              True              False
15             True             False              False
```

```
#b
month_order = ['January', 'February', 'March', 'April', 'May', 'June',
               'July', 'August', 'September', 'October', 'November',
'December']
df['Birth_Month'] = pd.Categorical(df['Birth_Month'],
categories=month_order, ordered=True)
df_sorted = df.sort_values('Birth_Month')
print(df_sorted)
```

```
               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
12    Rashmi Talwar        June      F            III
8    Meeta Kulkarni        July      F             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. 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:

a. Calculate and display familywise gross monthly income.

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

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

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

```
# Creating the DataFrame
data = {
    'Name': ['Shah', 'Vats', 'Vats', 'Kumar', 'Vats', 'Kumar', 'Shah',
'Shah', 'Kumar', 'Vats'],
    'Gender': ['Male', 'Male', 'Female', 'Female', 'Female', 'Male',
'Male', 'Female', 'Female', 'Male'],
    'MonthlyIncome': [114000.00, 65000.00, 43150.00, 69500.00,
155000.00, 103000.00, 55000.00, 112400.00, 81030.00, 71900.00]
}

df = pd.DataFrame(data)

#a
income = df.groupby('Name')['MonthlyIncome'].sum()
income
```

```
Name
Kumar    253530.0
Shah     281400.0
Vats     335050.0
Name: MonthlyIncome, dtype: float64
```

```
#b
member = df.iloc[df.groupby('Name')['MonthlyIncome'].idxmax()]
member
```

```
    Name  Gender  MonthlyIncome
5  Kumar    Male       103000.0
0   Shah    Male       114000.0
4   Vats  Female       155000.0
```

```
#c
high_members = df[df['MonthlyIncome'] > 60000.00]
high_members
```

```
    Name  Gender  MonthlyIncome
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
9   Vats    Male        71900.0
```

```
#d
avg_female_income_shah = df[(df['Name'] == 'Shah') & (df['Gender'] ==
'Female')]['MonthlyIncome'].mean()
print("Average monthly income of female members in Shah family:",
avg_female_income_shah)
```

```
Average monthly income of female members in Shah family: 112400.0
```