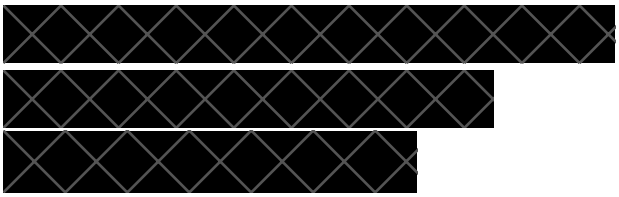
B.Sc(Hons) Sem V
Practical File :Data Analysis and Visualization



(Note: Any platform for Python can be used for lab exercises)

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]
Ans:

```
dict1={"Boys":[72,68,70,69,74],"Girls":[63,65,69,62,61]}

list1=[]
x=(len(dict1["Boys"]))
for i in range (x):
  list1.append({"Boys": dict1["Boys"][i],"Girls": dict1["Girls"][i]})
print(list1)
```

```
[{'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.
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.
Answer:
#a:

```
import numpy as np
array2d = np.random.randint(100, size = (3, 4))
array2d
```

```
array([[91, 10, 18, 61],
       [53, 43, 52, 85],
       [76, 14, 35, 94]])
```

```
#Mean
np.mean(array2d, axis = 1)
```

```
array([45.  , 58.25, 54.75])
```

```
#Standard Deviation
np.std(array2d, axis = 1)
```

```
array([32.8861673 , 15.92757044, 31.79131171])
```

```
#Variance
np.var(array2d, axis = 1)
```

```
array([1081.5   ,  253.6875, 1010.6875])
```

#b

```
b = [56, 48, 22, 41, 78, 91, 24, 46, 8, 33]
sorted_indices = np.argsort(b)
print(sorted_indices)
```

```
[8 2 6 9 3 7 1 0 4 5]
```

#c

```
m = int(input('Enter the value of m: '))
n = int(input('Enter the value of n: '))
arr2d = np.random.randint(100, size = (m, n))
print(arr2d)
```

```
[[ 9 15 65 64]
 [68 26 24  8]
 [58 42 60 19]]
```

```
#Shape
print(arr2d.shape)
```

```
(3, 4)
```

```
#Dimension
print(arr2d.ndim)
```

```
 2
```

```
#Data Type
print(arr2d.dtype)
```

```
int64
```

```
#Reshaping into n*m array
```

```
[[ 9 15 65]
 [64 68 26]
 [24  8 58]
 [42 60 19]]
```

#d

```
arr_2 = np.array([[0, 2, 3], [4, 1, 0], [0, 0, 2], [np.nan, 3, np.nan]])
print(arr_2)
```

```
[[ 0.  2.  3.]
 [ 4.  1.  0.]
 [ 0.  0.  2.]
 [nan  3. nan]]
```

Indices of elements which are zero

```
indices_zero = np.argwhere(arr_2 == 0)
print(indices_zero)
```

```
[[0 0]
 [1 2]
 [2 0]
 [2 1]]
```

Indices of elements which are NaN

```
indices_nan = np.argwhere(np.isnan(arr_2))
print(indices_nan)
```

```
 [[3 0]
  [3 2]]
```

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
Answer:

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

	A	B	C
0	1	11	34
1	29	35	23
2	32	31	1
3	21	11	14
4	22	26	1
5	15	15	42
6	9	8	17
7	9	40	8
8	1	43	26
9	16	9	46
10	15	47	46
11	0	1	2
12	17	1	41
13	10	4	4
14	20	33	3
15	8	12	9
16	38	33	48
17	37	39	10
18	2	13	11
19	18	5	31
20	31	22	15

```
### Replace 10% of the values by null values whose index positions are generated using random function.
rows = len(frame)
cols = len(frame.columns)
```

```
no_of_elements_to_replace = int(rows * cols * 0.1)
no_of_elements_to_replace
```

```
15
```

```
while no_of_elements_to_replace != 0:
    i = np.random.randint(rows)
    j = np.random.randint(cols)

    if frame.iloc[i, j] != np.nan:
        frame.iat[i, j] = np.nan
        no_of_elements_to_replace -= 1
frame
```

	A	B	C
0	1.0	11.0	34.0
1	29.0	35.0	23.0
2	32.0	31.0	1.0
3	21.0	11.0	14.0
4	NaN	26.0	1.0
5	15.0	15.0	42.0
6	9.0	8.0	17.0
7	9.0	40.0	8.0
8	1.0	43.0	26.0
9	16.0	9.0	46.0
10	15.0	47.0	46.0
11	0.0	1.0	2.0
12	17.0	1.0	41.0
13	10.0	4.0	4.0
14	20.0	33.0	NaN
15	8.0	NaN	9.0
16	38.0	33.0	48.0
17	37.0	39.0	10.0
18	2.0	13.0	11.0
19	18.0	5.0	31.0
20	31.0	22.0	15.0
21	3.0	9.0	24.0
22	NaN	37.0	3.0
23	42.0	34.0	37.0
24	NaN	9.0	6.0
25	26.0	NaN	43.0

#a

```
no_of_missing_values = frame.isnull().sum().sum()
no_of_missing_values
```

#b

```
frame.dropna(axis=1, how='any', thresh=rows-5)
```

	B	C
0	11.0	34.0
1	35.0	23.0
2	31.0	1.0
3	11.0	14.0
4	26.0	1.0
5	15.0	42.0
6	8.0	17.0
7	40.0	8.0
8	43.0	26.0
9	9.0	46.0
10	47.0	46.0
11	1.0	2.0
12	1.0	41.0
13	4.0	4.0
14	33.0	NaN
15	NaN	9.0
16	33.0	48.0
17	39.0	10.0
18	13.0	11.0
19	5.0	31.0
20	22.0	15.0
21	9.0	24.0
22	37.0	3.0
23	34.0	37.0

#c

```
row_to_drop = frame.sum(axis=1).idxmax()
row_to_drop
frame.drop(row_to_drop)
```

	A	B	C
0	1.0	11.0	34.0
1	29.0	35.0	23.0
2	32.0	31.0	1.0
3	21.0	11.0	14.0
4	NaN	26.0	1.0
5	15.0	15.0	42.0
6	9.0	8.0	17.0
7	9.0	40.0	8.0
8	1.0	43.0	26.0
9	16.0	9.0	46.0
10	15.0	47.0	46.0
11	0.0	1.0	2.0
12	17.0	1.0	41.0
13	10.0	4.0	4.0
14	20.0	33.0	NaN
15	8.0	NaN	9.0
16	38.0	33.0	48.0
17	37.0	39.0	10.0
18	2.0	13.0	11.0
19	18.0	5.0	31.0
20	31.0	22.0	15.0
21	3.0	9.0	24.0

#d

```
frame.sort_values(by=frame.columns[0])
```

	A	B	C
11	0.0	1.0	2.0
0	1.0	11.0	34.0
32	1.0	42.0	10.0
8	1.0	43.0	26.0
18	2.0	13.0	11.0
41	3.0	5.0	NaN
47	3.0	41.0	13.0
21	3.0	9.0	24.0
15	8.0	NaN	9.0
6	9.0	8.0	17.0
7	9.0	40.0	8.0
36	10.0	34.0	17.0
13	10.0	4.0	4.0
34	12.0	14.0	3.0
38	13.0	2.0	2.0
45	15.0	44.0	49.0
10	15.0	47.0	46.0
5	15.0	15.0	42.0
9	16.0	9.0	46.0
48	16.0	NaN	NaN
12	17.0	1.0	41.0
19	18.0	5.0	31.0
40	19.0	1.0	31.0
14	20.0	33.0	NaN
3	21.0	11.0	14.0
43	22.0	2.0	NaN
33	22.0	31.0	14.0
31	22.0	NaN	12.0

#e

```
frame.drop_duplicates(subset=frame.columns[0], keep='first')
```

	A	B	C
0	1.0	11.0	34.0
1	29.0	35.0	23.0
2	32.0	31.0	1.0
3	21.0	11.0	14.0
4	NaN	26.0	1.0
5	15.0	15.0	42.0
6	9.0	8.0	17.0
9	16.0	9.0	46.0
11	0.0	1.0	2.0
12	17.0	1.0	41.0
13	10.0	4.0	4.0
14	20.0	33.0	NaN
15	8.0	NaN	9.0
16	38.0	33.0	48.0
17	37.0	39.0	10.0
18	2.0	13.0	11.0
19	18.0	5.0	31.0
20	31.0	22.0	15.0
21	3.0	9.0	24.0
23	42.0	34.0	37.0
25	26.0	NaN	43.0
27	49.0	49.0	29.0
31	22.0	NaN	12.0
34	12.0	14.0	3.0
37	28.0	13.0	42.0
38	13.0	2.0	2.0

#f

```
correlation = frame['A'].corr(frame['B'])
correlation
```

```
0.18831749778248819
```

```
covariance = frame['B'].cov(frame['C'])
covariance
```

```
19.534843205574916
```

#h

```
pd.cut(frame['B'], 5)
```

```
0       (9.8, 19.6]
1      (29.4, 39.2]
2      (29.4, 39.2]
3       (9.8, 19.6]
4      (19.6, 29.4]
5       (9.8, 19.6]
6     (-0.049, 9.8]
7      (39.2, 49.0]
8      (39.2, 49.0]
9     (-0.049, 9.8]
10     (39.2, 49.0]
11    (-0.049, 9.8]
12    (-0.049, 9.8]
13    (-0.049, 9.8]
14     (29.4, 39.2]
15              NaN
16     (29.4, 39.2]
17     (29.4, 39.2]
18      (9.8, 19.6]
19    (-0.049, 9.8]
20     (19.6, 29.4]
21    (-0.049, 9.8]
22     (29.4, 39.2]
23     (29.4, 39.2]
24    (-0.049, 9.8]
...
47     (39.2, 49.0]
48              NaN
49    (-0.049, 9.8]
Name: B, dtype: category
Categories (5, interval[float64, right]): [(-0.049, 9.8] < (9.8, 19.6] < (19.6, 29.4] < (29.4, 39.2] < (39.2, 49.0]]
```

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:

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.

Answer:

```
import pandas as pd
day1 = pd.read_excel("file1.xlsx")
day2 = pd.read_excel("file2.xlsx")
```

```
df1 = pd.DataFrame(day1)
df2 = pd.DataFrame(day2)

# a
print(pd.merge(df1,df2,on ="Name",how="inner"))


# b
print("\n")
# print(pd.merge(df1,df2,on ="Name",how="outer"))
names1 = list(df1["Name"])
names2 = list(df2["Name"])

def Union(lst1, lst2):
    final_list = list(set(lst1) | set(lst2))
    return final_list

print(Union(names1, names2))

# c
frame_combined = pd.concat([day1, day2], ignore_index=True)
print(frame_combined.count())


# d
both_days = pd.merge(df1,df2,how='outer',on=['Name','Duration']).copy()

both_days.fillna(value='-',inplace=True)

both_days.set_index(['Name','Duration'])
print(pd.DataFrame(both_days.set_index(['Name','Duration'])).describe() )
```

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.

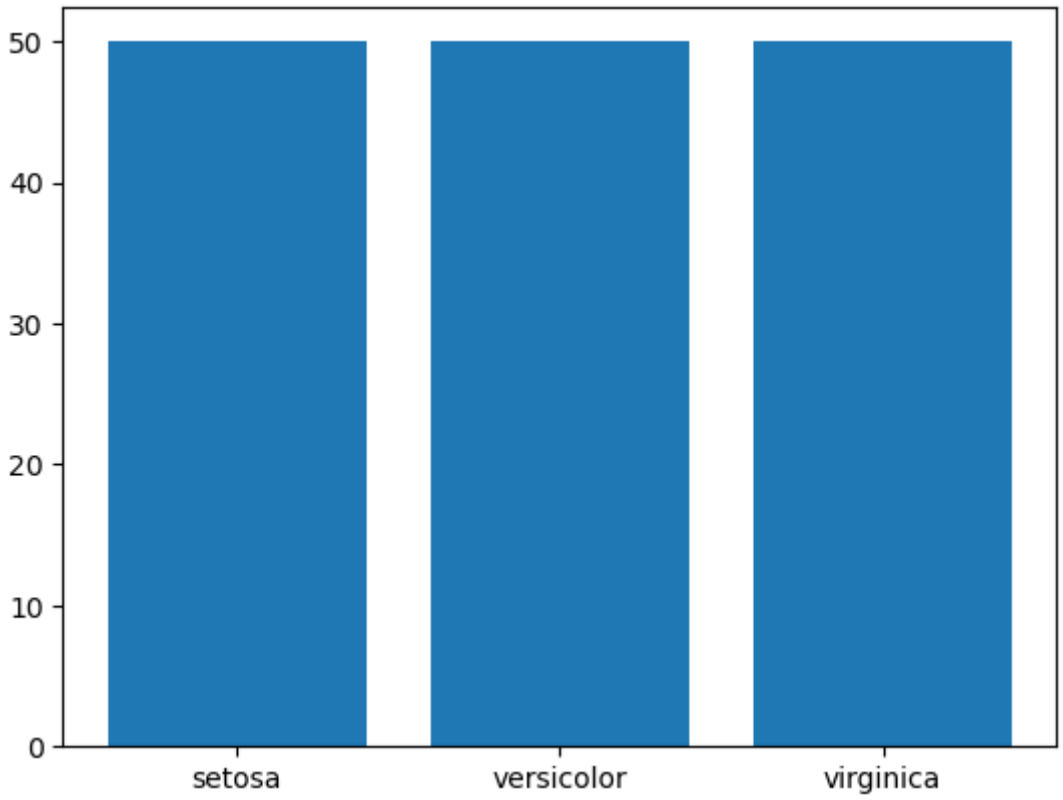
b. Draw a scatter plot for Petal width vs sepal width.

c. Plot density distribution for feature petal length.

d. Use a pair plot to show pairwise bivariaimport pandas as pd

Answer:

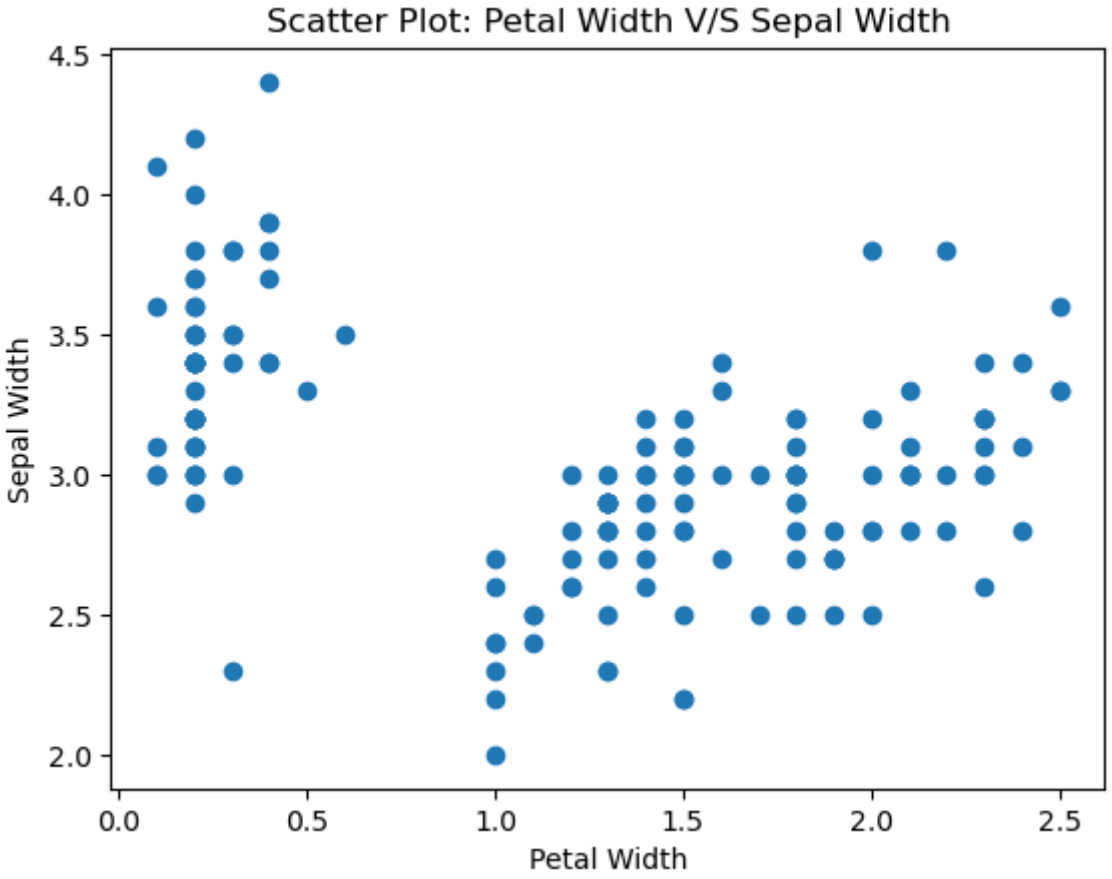
```
import seaborn as sns
import matplotlib.pyplot as pltte distribution in the lris.
from sklearn.datasets import load_iris
data = load_iris()
#a
plt.bar(data.target_names, [50, 50, 50])
```




```
#b
frame = pd.DataFrame(data.data, columns=data.feature_names)
frame.head()
```

	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

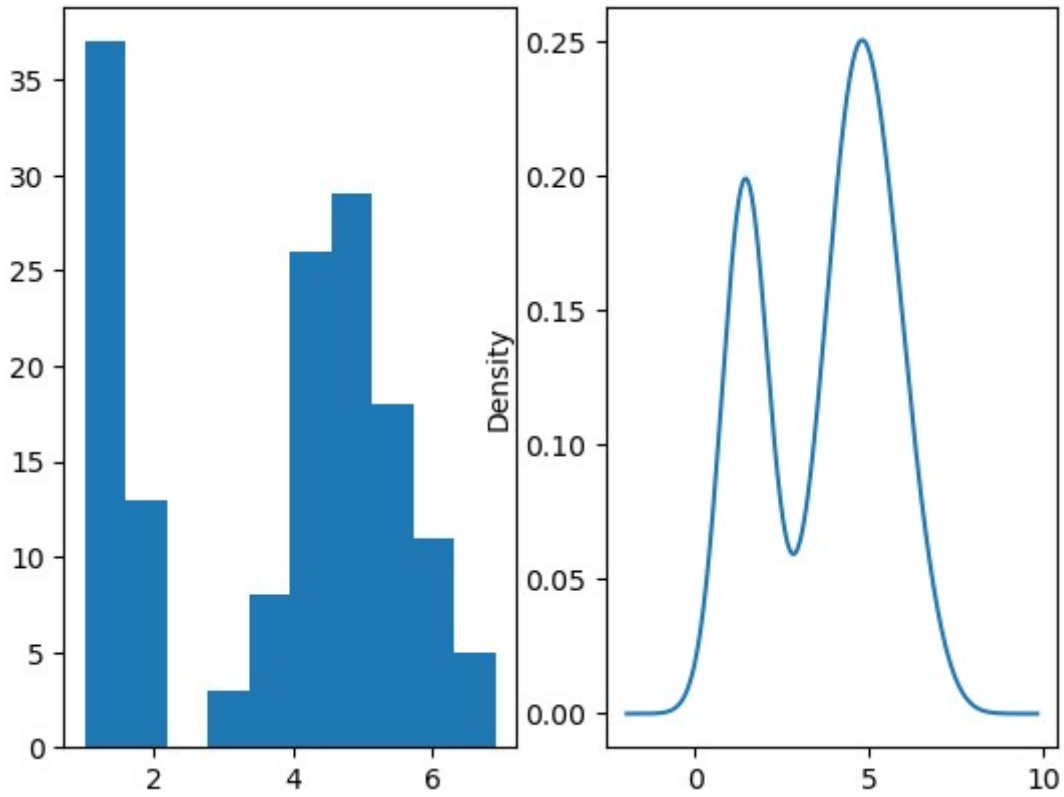
```
sepalWidth = data.feature_names[1]
petalWidth = data.feature_names[3]
plt.scatter(frame[petalWidth], frame[sepalWidth])
plt.title('Scatter Plot: Petal Width V/S Sepal Width')
plt.xlabel('Petal Width')
plt.ylabel('Sepal Width')
plt.show()
```



#c

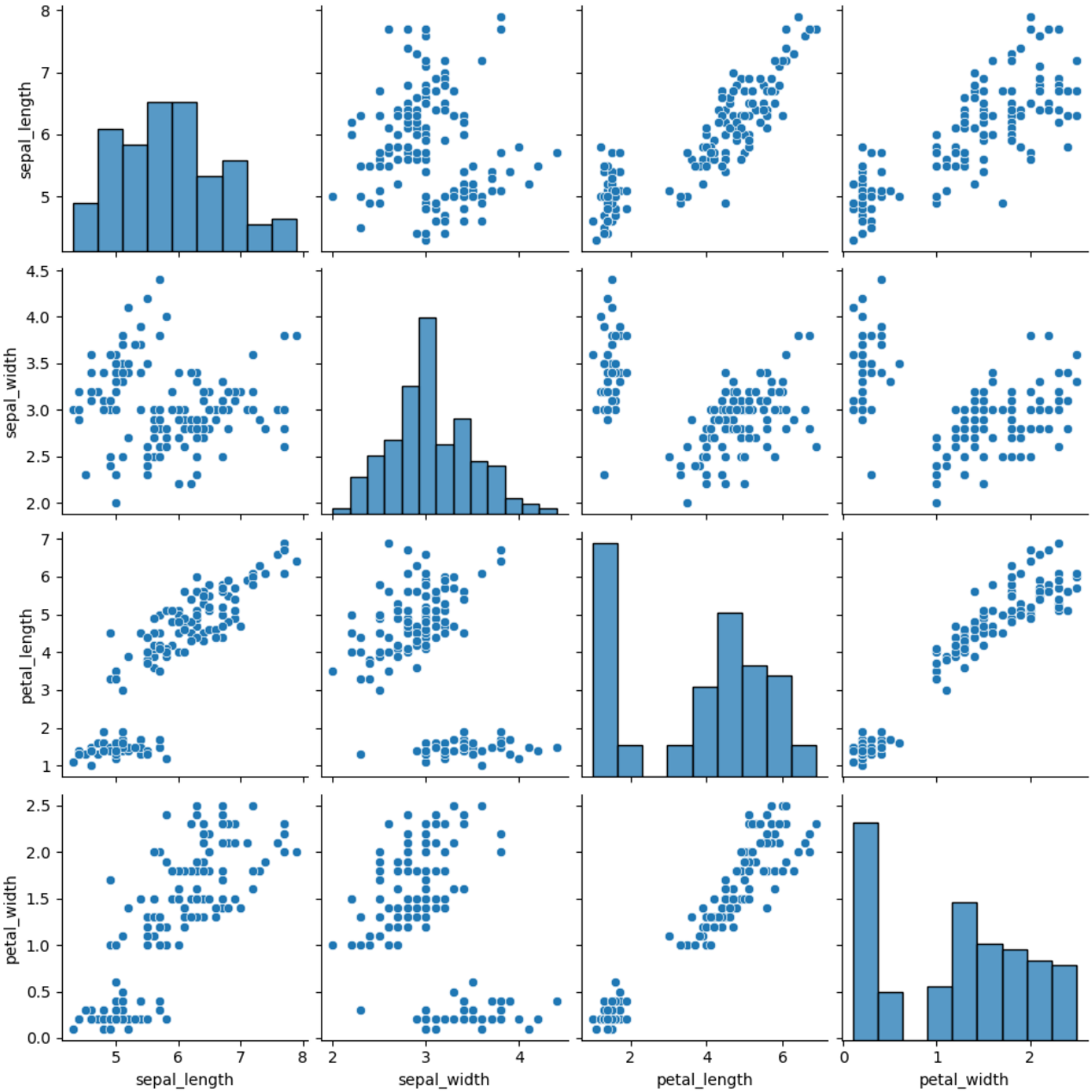
```
petalLength = data.feature_names[2]
fig = plt.figure()
histGraph = fig.add_subplot(1, 2, 1)
```

```
plt.hist(frame[petalLength])
densityGraph = fig.add_subplot(1, 2, 2)
frame[petalLength].plot.density()
```



#d

```
data = sns.load_dataset('iris')
sns.pairplot(data)
```



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.

```
import pandas as pd

import numpy as np

frame = pd.read_csv('climate.csv')

frame.to_csv('weatherReport.csv')
```

dataset = frame.drop_duplicates(subset=['Date Time']).reset_index(drop=True)

dataset

	Date Time	p (mbar)	T (degC)	Tpot (K)	Tdew (degC)	rh (%)	VPmax (mbar)	VPact (mbar)	VPdef (mbar)	sh (g/kg)	H2OC (mmol/mol)	rho (g/m**3)	wv (m/s)	max. wv (m/s)	wd (deg)
0	01.01.2009 00:10:00	996.52	-8.02	265.40	-8.90	93.30	3.33	3.11	0.22	1.94	3.12	1307.75	1.03	1.75	152.3
1	01.01.2009 00:20:00	996.57	-8.41	265.01	-9.28	93.40	3.23	3.02	0.21	1.89	3.03	1309.80	0.72	1.50	136.1
2	01.01.2009 00:30:00	996.53	-8.51	264.91	-9.31	93.90	3.21	3.01	0.20	1.88	3.02	1310.24	0.19	0.63	171.6
3	01.01.2009 00:40:00	996.51	-8.31	265.12	-9.07	94.20	3.26	3.07	0.19	1.92	3.08	1309.19	0.34	0.50	198.0
4	01.01.2009 00:50:00	996.51	-8.27	265.15	-9.04	94.10	3.27	3.08	0.19	1.92	3.09	1309.00	0.32	0.63	214.3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
420219	31.12.2016 23:20:00	1000.07	-4.05	269.10	-8.13	73.10	4.52	3.30	1.22	2.06	3.30	1292.98	0.67	1.52	240.0
420220	31.12.2016 23:30:00	999.93	-3.35	269.81	-8.06	69.71	4.77	3.32	1.44	2.07	3.32	1289.44	1.14	1.92	234.3
420221	31.12.2016 23:40:00	999.82	-3.16	270.01	-8.21	67.91	4.84	3.28	1.55	2.05	3.28	1288.39	1.08	2.00	215.2
420222	31.12.2016 23:50:00	999.81	-4.23	268.94	-8.53	71.80	4.46	3.20	1.26	1.99	3.20	1293.56	1.49	2.16	225.8
420223	01.01.2017 00:00:00	999.82	-4.82	268.36	-8.42	75.70	4.27	3.23	1.04	2.01	3.23	1296.38	1.23	1.96	184.9

420224 rows × 15 columns

#a

dataset['T (degC)'].groupby(dataset['p (mbar)']).mean()

```
p (mbar)
913.60     25.110
914.10     25.330
917.40     25.255
918.30     25.560
918.50     25.080
           ...
1015.26     3.480
1015.28     3.540
1015.29     3.485
1015.30     3.600
1015.35     3.640
Name: T (degC), Length: 6117, dtype: float64
```

#b

rows_to_drop = np.random.choice(dataset.index, int(dataset.shape[0]*25/100), replace=False)

frame = dataset.drop(rows_to_drop).copy()

frame

	Date Time	p (mbar)	T (degC)	Tpot (K)	Tdew (degC)	rh (%)	VPmax (mbar)	VPact (mbar)	VPdef (mbar)	sh (g/kg)	H2OC (mmol/mol)	rho (g/m**3)	wv (m/s)	max. wv (m/s)	wd (deg)
0	01.01.2009 00:10:00	996.52	-8.02	265.40	-8.90	93.30	3.33	3.11	0.22	1.94	3.12	1307.75	1.03	1.75	152.3
3	01.01.2009 00:40:00	996.51	-8.31	265.12	-9.07	94.20	3.26	3.07	0.19	1.92	3.08	1309.19	0.34	0.50	198.0
4	01.01.2009 00:50:00	996.51	-8.27	265.15	-9.04	94.10	3.27	3.08	0.19	1.92	3.09	1309.00	0.32	0.63	214.3
5	01.01.2009 01:00:00	996.50	-8.05	265.38	-8.78	94.40	3.33	3.14	0.19	1.96	3.15	1307.86	0.21	0.63	192.7
6	01.01.2009 01:10:00	996.50	-7.62	265.81	-8.30	94.80	3.44	3.26	0.18	2.04	3.27	1305.68	0.18	0.63	166.5
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
420217	31.12.2016 23:00:00	1000.21	-3.76	269.39	-7.95	72.50	4.62	3.35	1.27	2.09	3.35	1291.71	0.89	1.30	223.7
420218	31.12.2016 23:10:00	1000.11	-3.93	269.23	-8.09	72.60	4.56	3.31	1.25	2.06	3.31	1292.41	0.56	1.00	202.6
420220	31.12.2016 23:30:00	999.93	-3.35	269.81	-8.06	69.71	4.77	3.32	1.44	2.07	3.32	1289.44	1.14	1.92	234.3
420221	31.12.2016 23:40:00	999.82	-3.16	270.01	-8.21	67.91	4.84	3.28	1.55	2.05	3.28	1288.39	1.08	2.00	215.2
420223	01.01.2017 00:00:00	999.82	-4.82	268.36	-8.42	75.70	4.27	3.23	1.04	2.01	3.23	1296.38	1.23	1.96	184.9

time_series = pd.date_range(frame['Date Time'].min(), frame['Date Time'].max(), freq='10T').strftime('%d.%m.%Y %H:%M:%S')

frame = frame.set_index('Date Time').reindex(time_series, fill_value=0.0).rename_axis('Date Time').reset_index()

frame

	Date Time	p (mbar)	T (degC)	Tpot (K)	Tdew (degC)	rh (%)	VPmax (mbar)	VPact (mbar)	VPdef (mbar)	sh (g/kg)	H2OC (mmol/mol)	rho (g/m**3)	wv (m/s)	max. wv (m/s)	wd (deg)
0	01.01.2009 00:10:00	996.52	-8.02	265.40	-8.90	93.30	3.33	3.11	0.22	1.94	3.12	1307.75	1.03	1.75	152.3
1	01.01.2009 00:20:00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.0
2	01.01.2009 00:30:00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.0
3	01.01.2009 00:40:00	996.51	-8.31	265.12	-9.07	94.20	3.26	3.07	0.19	1.92	3.08	1309.19	0.34	0.50	198.0
4	01.01.2009 00:50:00	996.51	-8.27	265.15	-9.04	94.10	3.27	3.08	0.19	1.92	3.09	1309.00	0.32	0.63	214.3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
420761	31.12.2016 23:00:00	1000.21	-3.76	269.39	-7.95	72.50	4.62	3.35	1.27	2.09	3.35	1291.71	0.89	1.30	223.7
420762	31.12.2016 23:10:00	1000.11	-3.93	269.23	-8.09	72.60	4.56	3.31	1.25	2.06	3.31	1292.41	0.56	1.00	202.6
420763	31.12.2016 23:20:00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.0
420764	31.12.2016 23:30:00	999.93	-3.35	269.81	-8.06	69.71	4.77	3.32	1.44	2.07	3.32	1289.44	1.14	1.92	234.3
420765	31.12.2016 23:40:00	999.82	-3.16	270.01	-8.21	67.91	4.84	3.28	1.55	2.05	3.28	1288.39	1.08	2.00	215.2

#c

frame = dataset.copy()

frame.head③

	Date Time	p (mbar)	T (degC)	Tpot (K)	Tdew (degC)	rh (%)	VPmax (mbar)	VPact (mbar)	VPdef (mbar)	sh (g/kg)	H2OC (mmol/mol)	rho (g/m**3)	wv (m/s)	max. wv (m/s)	wd (deg)
0	01.01.2009 00:10:00	996.52	-8.02	265.40	-8.90	93.3	3.33	3.11	0.22	1.94	3.12	1307.75	1.03	1.75	152.3
1	01.01.2009 00:20:00	996.57	-8.41	265.01	-9.28	93.4	3.23	3.02	0.21	1.89	3.03	1309.80	0.72	1.50	136.1
2	01.01.2009 00:30:00	996.53	-8.51	264.91	-9.31	93.9	3.21	3.01	0.20	1.88	3.02	1310.24	0.19	0.63	171.6

frame['Date Time'].dtype

```
dtype('O')
```

frame['Date Time'] = pd.to_datetime(frame['Date Time'])

frame['Date Time'].dtype

#d

frame.insert(1, 'Year', pd.DatetimeIndex(frame['Date Time']).year)

frame.insert(1, 'Month', pd.DatetimeIndex(frame['Date Time']).month_name())

frame.head③

	Date Time	Month	Year	p (mbar)	T (degC)	Tpot (K)	Tdew (degC)	rh (%)	VPmax (mbar)	VPact (mbar)	VPdef (mbar)	sh (g/kg)	H2OC (mmol/mol)	rho (g/m**3)	wv (m/s)	max. wv (m/s)	wd (deg)
0	2009-01-01 00:10:00	January	2009	996.52	-8.02	265.40	-8.90	93.3	3.33	3.11	0.22	1.94	3.12	1307.75	1.03	1.75	152.3
1	2009-01-01 00:20:00	January	2009	996.57	-8.41	265.01	-9.28	93.4	3.23	3.02	0.21	1.89	3.03	1309.80	0.72	1.50	136.1
2	2009-01-01 00:30:00	January	2009	996.53	-8.51	264.91	-9.31	93.9	3.21	3.01	0.20	1.88	3.02	1310.24	0.19	0.63	171.6

aggregate_frame = frame.groupby(['Year', 'Month']).agg({'T (degC)': 'mean'})

```
result = aggregate_frame['T (degC)'].groupby(level=0, group_keys=False)

pd.set_option('display.max_rows()', None)
result.nlargest(12)
```

```
Year  Month
2009  August       15.147069
      July         14.685078
      June         12.519252
      May          11.974877
      September    11.352389
      April        10.443676
      November      8.406447
      October       6.444030
      March         6.177995
      February      4.291989
      January       3.133713
      December      1.187982
2010  July         14.350058
      June         12.574234
      August       12.345121
      September     9.860824
      May           9.687681
      April         8.737472
      March         7.927339
      October       6.510762
      November      4.027065
      February      3.433162
      January       0.421640
      December     -0.531102
...
      February      5.467081
      December      5.239516
      January       4.991823
2017  January      -4.820000
Name: T (degC), dtype: float64
```

#e

```
bins = 5

frame = dataset.groupby(['p (mbar)', pd.cut(dataset['T (degC)'], bins)])
result = frame.size().unstack()
```

result

T (degC) p (mbar)	(-23.07, -10.952]	(-10.952, 1.106]	(1.106, 13.164]	(13.164, 25.222]	(25.222, 37.28]
913.60	0	0	0	1	0
914.10	0	0	0	0	1
917.40	0	0	0	1	1
918.30	0	0	0	0	1
918.50	0	0	0	1	0
942.43	0	0	1	0	0
942.54	0	0	1	0	0
942.58	0	0	1	0	0
942.59	0	0	1	0	0
942.62	0	0	1	0	0
942.65	0	0	2	0	0
942.72	0	0	1	0	0
942.74	0	0	1	0	0
942.95	0	0	1	0	0
942.97	0	0	1	0	0
943.06	0	0	1	0	0
943.11	0	0	1	0	0
943.12	0	0	1	0	0
943.19	0	0	1	0	0
943.22	0	0	1	0	0
943.35	0	0	1	0	0

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

Name	BirthMonth	Gender	Pass Division
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	III
Ritu Agarwal	September	F	I

Akshay Goel	August	M	I
Meeta Kulkarni	July	F	II
PreetiAhuja	Novemb er	F	II
SunilDas Gupta	April	M	III
SonaliSapr e	January	F	I
Rashmi Talwar	June	F	III
Ashish Dubey	May	M	II
Kiran Sharma	February	F	II
Sameer Bansal	October	M	I

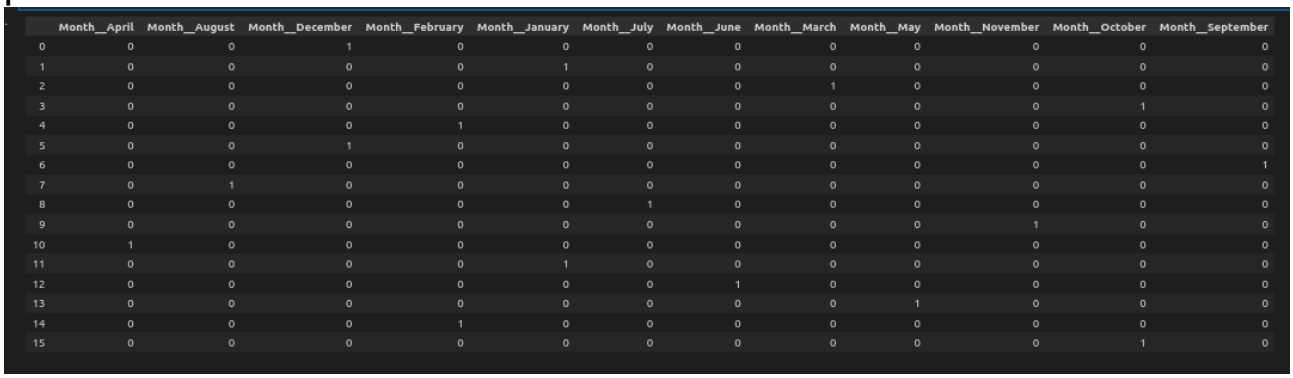
a. Perform one hot encoding of the last two columns of categorical data using the get_dummies() function.
b. Sort this data frame on the "Birth Month" column (i.e. January to December). Hint: Convert Month to Categorical.
Answer:
#a:

```
import numpy as np
import pandas as pd
data=pd.read_csv('question7.csv')
```

data

	Name	BirthMonth	Gender	Pass Division
0	Mudit Chauhan	December	M	Ill
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	Ill
6	Ritu Agarwal	September	F	I
7	Akshay Goel	August	M	I
8	Meeta Kulkarni	July	F	II
9	PreetiAhuja	November	F	II
10	SunilDas Gupta	April	M	Ill
11	SonaliSapre	January	F	I
12	Rashmi Talwar	June	F	Ill
13	Ashish Dubey	May	M	II
14	Kiran Sharma	February	F	II
15	Sameer Bansal	October	M	I

pseudoDF=pd.get_dummies(data.BirthMonth,prefix='Month_')
pseudoDF



	Month_April	Month_August	Month_December	Month_February	Month_January	Month_July	Month_June	Month_March	Month_May	Month_November	Month_October	Month_September
0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	1	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	1	0
4	0	0	0	1	0	0	0	0	0	0	0	0
5	0	0	1	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	1
7	0	1	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	1	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	1	0	0
10	1	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	1	0	0	0	0	0	0	0
12	0	0	0	0	0	0	1	0	0	0	0	0
13	0	0	0	0	0	0	0	0	1	0	0	0
14	0	0	0	1	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	1	0

#b:
dict1={
'Month': ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November','December'],

```
'Values':[1,2,3,4,5,6,7,8,9,10,11,12]
}
MonthsName=list(dict1['Month'])
#
ValueList=list(dict1['Values'])
i=0
while i < len(data.index):
   j=0
   while j < 12:
      if(data.BirthMonth[i]==MonthsName[j]):
         data.BirthMonth[i]=ValueList[j]
      j=j+1
   i=i+1
data.BirthMonth
```

```
0     12
1      1
2      3
3     10
4      2
5     12
6      9
7      8
8      7
9     11
10     4
11     1
12     6
13     5
14     2
15    10
Name: BirthMonth, dtype: object
```

```
data.sort_values(by=['BirthMonth','Name'])
```

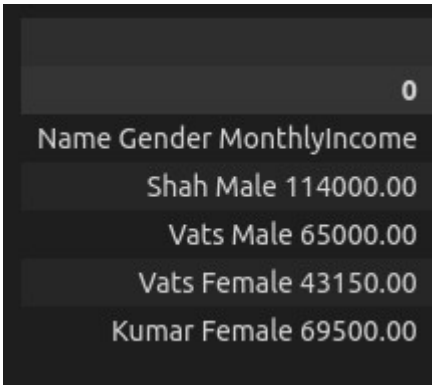
	Name	BirthMonth	Gender	Pass Division
1	Seema Chopra	1	F	II
11	SonaliSapre	1	F	I
14	Kiran Sharma	2	F	II
4	Sanjeev Sahni	2	M	II
2	Rani Gupta	3	F	I
10	SunilDas Gupta	4	M	Ill
13	Ashish Dubey	5	M	II
12	Rashmi Talwar	6	F	Ill
8	Meeta Kulkarni	7	F	II
7	Akshay Goel	8	M	I
6	Ritu Agarwal	9	F	I
3	Aditya Narayan	10	M	I
15	Sameer Bansal	10	M	I
9	PreetiAhuja	11	F	II
0	Mudit Chauhan	12	M	Ill
5	Prakash Kumar	12	M	Ill

8. Consider the following data frame containing a family name, gender of the family member and her/his monthly
income in each record.
Name Gender MonthlyIncome (Rs.)
Shah Male 114000.00
Vats Male 65000.00
Vats Female 43150.00
Kumar Female 69500.00
Vats Female 155000.00
Kumar Male 103000.00
Shah Male 55000.00
Shah Female 112400.00
Kumar Female 81030.00
Vats Male 71900.00
Write a program in Python using Pandas to perform the following:
a. Calculate and display familywise gross monthly income.
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.

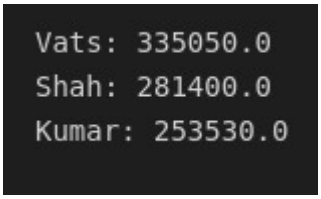
Answer:

```
import pandas as pd
import numpy as np
data =
pd.read_fwf('/home/raj/pyth/income.txt',header=None,index_col=0)
data.head()
```



	0
Name	Gender MonthlyIncome
Shah	Male 114000.00
Vats	Male 65000.00
Vats	Female 43150.00
Kumar	Female 69500.00

```
dict1={'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]
 }
#a
list1=set(dict1["Name"])
list1
for i in list1:
   print(i+":",df.loc[df['Name']==i,'MonthlyIncome'].sum())
```



```
Vats: 335050.0
Shah: 281400.0
Kumar: 253530.0
```

```
#b
listOfNames=list(dict1['Name'])
listOfMonthlyIncome=list(dict1['MonthlyIncome'])
```

```
ShahIncome=[]
KumarIncome=[]
VatsIncome=[]
length=len(listOfNames)
i=0
while i < length:
    if(listOfNames[i]=='Shah'):
        ShahIncome.append(listOfMonthlyIncome[i])
    elif(listOfNames[i]=='Kumar'):
        KumarIncome.append(listOfMonthlyIncome[i])
    elif(listOfNames[i]=='Vats'):
        VatsIncome.append(listOfMonthlyIncome[i])

    i=i+1
ShahIncome=pd.Series(ShahIncome)
ShahIncome.max()
```

```
114000.0
```

```
KumarIncome=pd.Series(KumarIncome)
KumarIncome.max()
```

```
103000.0
```

```
VatsIncome=pd.Series(VatsIncome)
VatsIncome.max()
```

```
155000.0
```

#c

```
df.loc[df['MonthlyIncome']>60000,'MonthlyIncome']
```

```
0    114000.0
1     65000.0
3     69500.0
4    155000.0
5    103000.0
7    112400.0
8     81030.0
9     71900.0
Name: MonthlyIncome, dtype: float64
```

#d

```
listOfGender=list(dict1['Gender'])
ShahIncomeFemale=[]
j=0
while j < length:
   if(listOfNames[j]=='Shah' and listOfGender[j]=='Female'):
      ShahIncomeFemale.append(listOfMonthlyIncome[j])
   j=j+1
ShahIncomeFemale=pd.Series(ShahIncomeFemale)
ShahIncomeFemale.mean()
```

```
112400.0
```