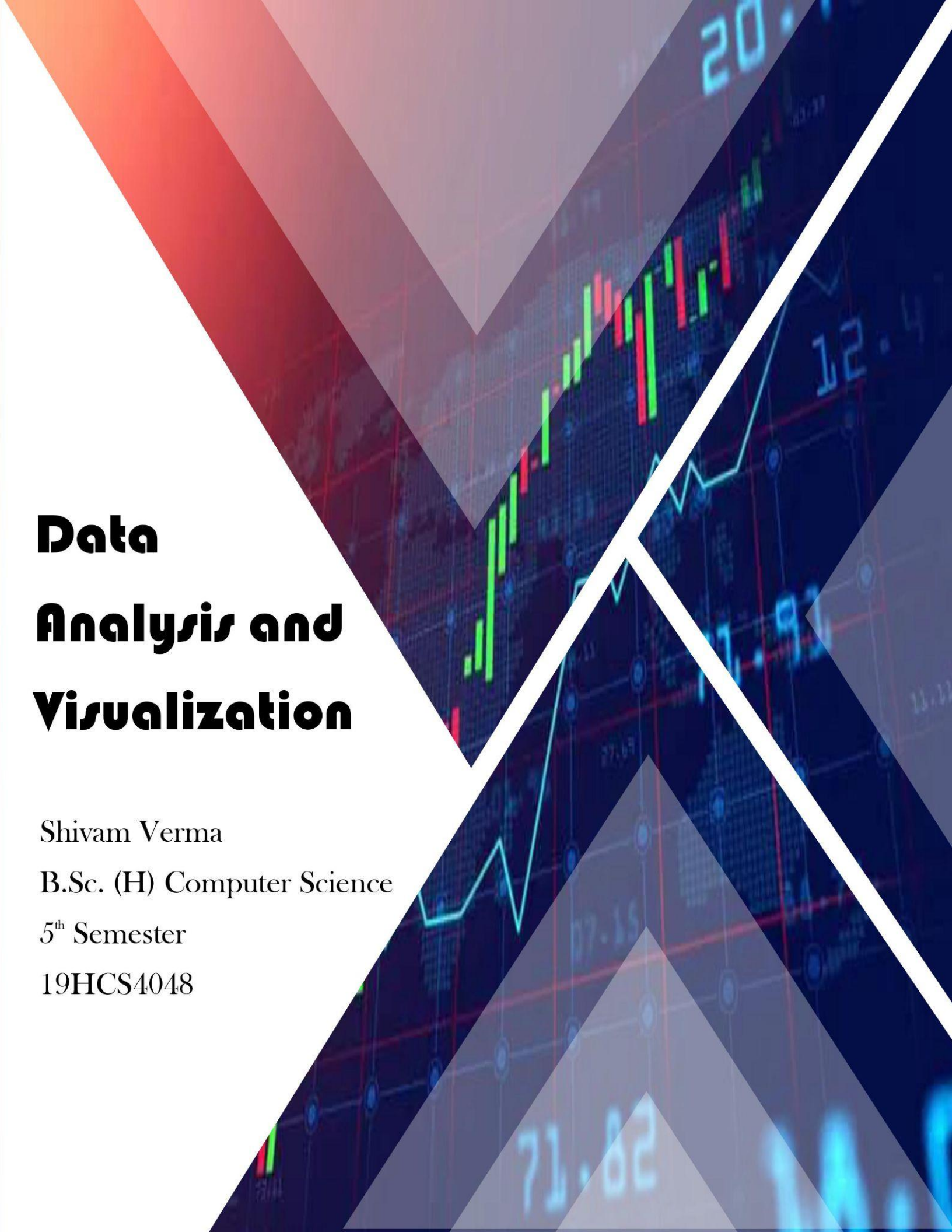


Data Analysis and Visualization

Shivam Verma
B.Sc. (H) Computer Science
5th Semester
19HCS4048

Question 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}]

CODE

```
height = {'Boys': [72, 68, 70, 69, 74], 'Girls': [63, 65, 69, 62, 61]}
result = [dict(zip(height.keys(), i)) for i in zip(*height.values())]
result
```

OUTPUT

Question 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 an n x 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.**

a.
```
import numpy as np
array2d = np.random.randint(100, size = (3, 4))
array2d
np.mean(array2d, axis = 1)
np.std(array2d, axis = 1)
np.var(array2d, axis = 1)
```

OUTPUT

```
In [9]: array2d
Out[9]: array([[79,  7, 48, 76],
               [20, 14, 53,  4],
               [67, 51, 19, 70]])
```

Mean

```
In [14]: np.mean(array2d, axis = 1)
Out[14]: array([52.5 , 22.75, 51.75])
```

Standard Deviation

```
In [15]: np.std(array2d, axis = 1)
Out[15]: array([28.91798748, 18.37627547, 20.24073862])
```

Variance

```
In [16]: np.var(array2d, axis = 1)
Out[16]: array([836.25  , 337.6875, 409.6875])
```

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

OUTPUT

```
In [13]: b = [56, 48, 22, 41, 78, 91, 24, 46, 8, 33]

In [19]: 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)
print(arr2d.shape)
print(arr2d.ndim)
print(arr2d.dtype)
arr_1 = np.reshape(arr2d, (n, m))
print(arr_1)

OUTPUT

```
In [20]: m = int(input('Enter the value of m: '))
         n = int(input('Enter the value of n: '))

         Enter the value of m: 4
         Enter the value of n: 5

In [26]: arr2d = np.random.randint(100, size = (m, n))
         print(arr2d)

         [[85 47  2 80 81]
          [69 51  5 47 54]
          [42 55 38 65 60]
          [24 56 95 23 70]]
```

Shape

```
In [27]: print(arr2d.shape)
```

```
(4, 5)
```

Dimension

```
In [30]: print(arr2d.ndim)
```

```
2
```

Data Type

```
In [28]: print(arr2d.dtype)
```

```
int32
```

Reshape into n X m array

```
In [29]: arr_1 = np.reshape(arr2d, (n, m))
         print(arr_1)
```

```
[[85 47  2 80]
 [81 69 51  5]
 [47 54 42 55]
 [38 65 60 24]
 [56 95 23 70]]
```

d. arr_2 = np.array([[0, 2, 3], [4, 1, 0], [0, 0, 2], [np.nan, 3, np.nan]])
print(arr_2)
indices_zero = np.argwhere(arr_2 == 0)
print(indices_zero)
indices_non_zero = np.argwhere(arr_2 != 0)
print(indices_non_zero)
indices_nan = np.argwhere(np.isnan(arr_2))
print(indices_nan)

OUTPUT

Indices of elements which are zero

In [12]:
```
indices_zero = np.argwhere(arr_2 == 0)
print(indices_zero)
```

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

Indices of elements which are non-zero

In [13]:
```
indices_non_zero = np.argwhere(arr_2 != 0)
print(indices_non_zero)
```

```
[[0 1]
 [0 2]
 [1 0]
 [1 1]
 [2 2]
 [3 0]
 [3 1]
 [3 2]]
```

Indices of elements which are NaN

In [14]:
```
indices_nan = np.argwhere(np.isnan(arr_2))
print(indices_nan)
```

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

Question 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. **Discretize the second column and create 5 bins.**

```
import pandas as pd
import numpy as np
frame = pd.DataFrame(np.random.randint(0, 50, size=(50, 3)),
columns=list('ABC'))
rows = len(frame)
cols = len(frame.columns)
no_of_elements_to_replace = int(rows * cols * 0.1)
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
```

OUTPUT

```
In [6]: frame
```

Out[6]:

	A	B	C
0	41.0	13.0	30.0
1	34.0	NaN	47.0
2	7.0	42.0	8.0
3	2.0	1.0	17.0
4	26.0	4.0	16.0
5	38.0	42.0	35.0
6	6.0	24.0	43.0
7	9.0	46.0	8.0
8	33.0	NaN	7.0
9	48.0	49.0	0.0
10	NaN	3.0	29.0
11	14.0	18.0	13.0
12	18.0	20.0	23.0
13	5.0	20.0	24.0
14	14.0	47.0	29.0
15	0.0	27.0	37.0
16	44.0	40.0	49.0
17	NaN	34.0	NaN
18	NaN	24.0	32.0
19	10.0	42.0	32.0
20	46.0	38.0	30.0
21	25.0	39.0	48.0
27	15.0	NaN	25.0
28	45.0	45.0	9.0
29	47.0	35.0	8.0
30	13.0	37.0	35.0
31	37.0	4.0	29.0
32	7.0	NaN	10.0
33	30.0	18.0	NaN
34	43.0	32.0	33.0
35	NaN	29.0	12.0
36	NaN	1.0	NaN
37	41.0	3.0	35.0
38	8.0	28.0	43.0
39	16.0	22.0	13.0
40	1.0	10.0	NaN
41	17.0	41.0	22.0
42	39.0	15.0	42.0
43	38.0	25.0	12.0
44	0.0	5.0	10.0
45	39.0	8.0	31.0
46	28.0	4.0	8.0
47	NaN	9.0	29.0
48	41.0	2.0	18.0
49	NaN	21.0	2.0

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

OUTPUT

```
In [7]: no_of_missing_values = frame.isnull().sum().sum()
        no_of_missing_values
Out[7]: 15
```

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

OUTPUT

```
In [8]: frame.dropna(axis=1, how='any', thresh=rows-5)
```

Out[8]:

	B	C
0	13.0	30.0
1	NaN	47.0
2	42.0	8.0
3	1.0	17.0
4	4.0	16.0
5	42.0	35.0
6	24.0	43.0
7	46.0	8.0
8	NaN	7.0
9	49.0	0.0
10	3.0	29.0
11	18.0	13.0
12	20.0	23.0
13	20.0	24.0
14	47.0	29.0
15	27.0	37.0
16	40.0	49.0
17	34.0	NaN
18	24.0	32.0
19	42.0	32.0
20	38.0	30.0

26	21.0	2.0
27	NaN	25.0
28	45.0	9.0
29	35.0	8.0
30	37.0	35.0
31	4.0	29.0
32	NaN	10.0
33	18.0	NaN
34	32.0	33.0
35	29.0	12.0
36	1.0	NaN
37	3.0	35.0
38	28.0	43.0
39	22.0	13.0
40	10.0	NaN
41	41.0	22.0
42	15.0	42.0
43	25.0	12.0
44	5.0	10.0
45	8.0	31.0
46	4.0	8.0
47	9.0	29.0
48	2.0	18.0
49	21.0	2.0

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

OUTPUT

```
In [15]: frame.drop(row_to_drop)
```

Out[15]:

	A	B	C
0	41.0	13.0	30.0
1	34.0	NaN	47.0
2	7.0	42.0	8.0
3	2.0	1.0	17.0
4	26.0	4.0	16.0
5	38.0	42.0	35.0
6	6.0	24.0	43.0
7	9.0	46.0	8.0
8	33.0	NaN	7.0
9	48.0	49.0	0.0
10	NaN	3.0	29.0
11	14.0	18.0	13.0
12	18.0	20.0	23.0
13	5.0	20.0	24.0
14	14.0	47.0	29.0
15	0.0	27.0	37.0
17	NaN	34.0	NaN
18	NaN	24.0	32.0
19	10.0	42.0	32.0
20	46.0	38.0	30.0
21	25.0	39.0	48.0
22	42.0	6.0	6.0

25	13.0	2.0	0.0
26	21.0	21.0	2.0
27	15.0	NaN	25.0
28	45.0	45.0	9.0
29	47.0	35.0	8.0
30	13.0	37.0	35.0
31	37.0	4.0	29.0
32	7.0	NaN	10.0
33	30.0	18.0	NaN
34	43.0	32.0	33.0
35	NaN	29.0	12.0
36	NaN	1.0	NaN
37	41.0	3.0	35.0
38	8.0	28.0	43.0
39	16.0	22.0	13.0
40	1.0	10.0	NaN
41	17.0	41.0	22.0
42	39.0	15.0	42.0
43	38.0	25.0	12.0
44	0.0	5.0	10.0
45	39.0	8.0	31.0
46	28.0	4.0	8.0
47	NaN	9.0	29.0
48	41.0	2.0	18.0
49	NaN	21.0	2.0

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

OUTPUT

```
In [16]: frame.sort_values(by=frame.columns[0])
```

Out[16]:

	A	B	C
44	0.0	5.0	10.0
15	0.0	27.0	37.0
40	1.0	10.0	NaN
3	2.0	1.0	17.0
13	5.0	20.0	24.0
6	6.0	24.0	43.0
32	7.0	NaN	10.0
2	7.0	42.0	8.0
38	8.0	28.0	43.0
7	9.0	46.0	8.0
19	10.0	42.0	32.0
30	13.0	37.0	35.0
25	13.0	2.0	0.0
14	14.0	47.0	29.0
11	14.0	18.0	13.0
27	15.0	NaN	25.0
39	16.0	22.0	13.0
41	17.0	41.0	22.0
12	18.0	20.0	23.0
26	21.0	21.0	2.0
24	25.0	18.0	23.0

8	33.0	NaN	7.0
1	34.0	NaN	47.0
31	37.0	4.0	29.0
43	38.0	25.0	12.0
5	38.0	42.0	35.0
42	39.0	15.0	42.0
45	39.0	8.0	31.0
0	41.0	13.0	30.0
37	41.0	3.0	35.0
48	41.0	2.0	18.0
22	42.0	6.0	6.0
34	43.0	32.0	33.0
16	44.0	40.0	49.0
28	45.0	45.0	9.0
20	46.0	38.0	30.0
29	47.0	35.0	8.0
9	48.0	49.0	0.0
10	NaN	3.0	29.0
17	NaN	34.0	NaN
18	NaN	24.0	32.0
35	NaN	29.0	12.0
36	NaN	1.0	NaN
47	NaN	9.0	29.0
49	NaN	21.0	2.0

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

OUTPUT

```
In [18]: frame.drop_duplicates(subset=frame.columns[0], keep='first')
```

Out[18]:

	A	B	C
0	41.0	13.0	30.0
1	34.0	NaN	47.0
2	7.0	42.0	8.0
3	2.0	1.0	17.0
4	26.0	4.0	16.0
5	38.0	42.0	35.0
6	6.0	24.0	43.0
7	9.0	46.0	8.0
8	33.0	NaN	7.0
9	48.0	49.0	0.0
10	NaN	3.0	29.0
11	14.0	18.0	13.0
12	18.0	20.0	23.0
13	5.0	20.0	24.0
15	0.0	27.0	37.0
16	44.0	40.0	49.0
19	10.0	42.0	32.0
20	46.0	38.0	30.0
21	25.0	39.0	48.0
22	42.0	6.0	6.0
25	13.0	2.0	0.0

25	13.0	2.0	0.0
26	21.0	21.0	2.0
27	15.0	NaN	25.0
28	45.0	45.0	9.0
29	47.0	35.0	8.0
31	37.0	4.0	29.0
33	30.0	18.0	NaN
34	43.0	32.0	33.0
38	8.0	28.0	43.0
39	16.0	22.0	13.0
40	1.0	10.0	NaN
41	17.0	41.0	22.0
42	39.0	15.0	42.0
46	28.0	4.0	8.0

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

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

OUTPUT

```
In [20]: correlation = frame['A'].corr(frame['B'])
         correlation

Out[20]: 0.0538133562721363

In [21]: covariance = frame['B'].cov(frame['C'])
         covariance

Out[21]: 18.845528455284544
```

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

OUTPUT

```
In [22]: pd.cut(frame['B'], 5)

Out[22]: 0      (10.6, 20.2]
         1               NaN
         2      (39.4, 49.0]
         3     (0.952, 10.6]
         4     (0.952, 10.6]
         5      (39.4, 49.0]
         6      (20.2, 29.8]
         7      (39.4, 49.0]
         8               NaN
         9      (39.4, 49.0]
         10    (0.952, 10.6]
         11     (10.6, 20.2]
         12     (10.6, 20.2]
         13     (10.6, 20.2]
         14     (39.4, 49.0]
         15     (20.2, 29.8]
         16     (39.4, 49.0]
         17     (29.8, 39.4]
         18     (20.2, 29.8]
         19     (39.4, 49.0]
         20     (29.8, 39.4]
         21     (29.8, 39.4]
         22    (0.952, 10.6]
         23    (0.952, 10.6]
         24     (10.6, 20.2]
         25    (0.952, 10.6]
         26     (20.2, 29.8]
         27              NaN
         28     (39.4, 49.0]
         29     (29.8, 39.4]
         30     (29.8, 39.4]
         31    (0.952, 10.6]
         32              NaN
         33     (10.6, 20.2]
         34     (29.8, 39.4]
```

```
33     (10.6, 20.2]
34     (29.8, 39.4]
35     (20.2, 29.8]
36    (0.952, 10.6]
37    (0.952, 10.6]
38     (20.2, 29.8]
39     (20.2, 29.8]
40    (0.952, 10.6]
41     (39.4, 49.0]
42     (10.6, 20.2]
43     (20.2, 29.8]
44    (0.952, 10.6]
45    (0.952, 10.6]
46    (0.952, 10.6]
47    (0.952, 10.6]
48    (0.952, 10.6]
49     (20.2, 29.8]
Name: B, dtype: category
Categories (5, interval[float64, right]): [(0.952, 10.6] < (10.6, 20.2] < (20.2, 29.8] < (29.8, 39.4] < (39.4, 49.0]]
```

Question 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.**

```
import pandas as pd
first_workshop = pd.read_excel('Attendence-1.xlsx')
second_workshop = pd.read_excel('Attendence-2.xlsx')
first_frame = pd.DataFrame(first_workshop)
second_frame = pd.DataFrame(second_workshop)
```

a. `pd.merge(first_frame, second_frame, on='Name', how='inner')`

OUTPUT

```
In [4]: pd.merge(first_frame, second_frame, on='Name', how='inner')
```

Out[4]:

	Name	Time Of Joining_x	Duration_x	Time Of Joining_y	Duration_y
0	green	10:10:00	40.0	10:10:00	40
1	mr fantastic	10:20:00	50.0	10:20:00	50
2	invisible women	10:30:00	50.0	10:30:00	50
3	ant man	10:40:00	30.0	10:40:00	30

b. pd.merge(first_frame, second_frame, on='Name', how='outer')

OUTPUT

```
In [5]: pd.merge(first_frame, second_frame, on='Name', how='outer')
```

Out[5]:

	Name	Time Of Joining_x	Duration_x	Time Of Joining_y	Duration_y
0	james	10:00:00	30.0	NaN	NaN
1	green	10:10:00	40.0	10:10:00	40.0
2	mr fantastic	10:20:00	50.0	10:20:00	50.0
3	invisible women	10:30:00	50.0	10:30:00	50.0
4	ant man	10:40:00	30.0	10:40:00	30.0
5	susan	NaN	NaN	10:00:00	20.0

c. total_records = pd.concat({'Day 1': first_frame, 'Day 2': second_frame}, axis=0)
total_records

len(total_records)

OUTPUT

```
In [7]: total_records = pd.concat({'Day 1': first_frame, 'Day 2': second_frame}, axis=0)
```

```
In [8]: total_records
```

Out[8]:

		Name	Time Of Joining	Duration
Day 1	0	james	10:00:00	30.0
	1	green	10:10:00	40.0
	2	mr fantastic	10:20:00	50.0
	3	invisible women	10:30:00	50.0
	4	ant man	10:40:00	30.0
Day 2	0	susan	10:00:00	20.0
	1	green	10:10:00	40.0
	2	mr fantastic	10:20:00	50.0
	3	invisible women	10:30:00	50.0
	4	ant man	10:40:00	30.0

```
In [9]: len(total_records)
```

Out[9]: 10

Question 5

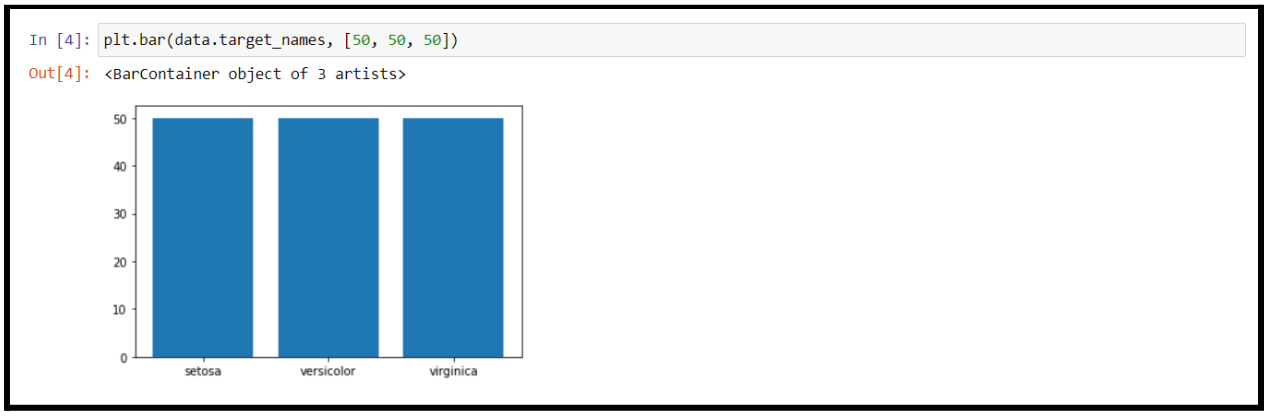
Taking Iris data, plot the following with proper legend and axis labels: (Download IRIS data from: https://archive.ics.uci.edu/ml/datasets/iris or import it from sklearn.datasets)

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 Dataset.**

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
data = load_iris()
```

a. plt.bar(data.target_names, [50, 50, 50])

OUTPUT



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

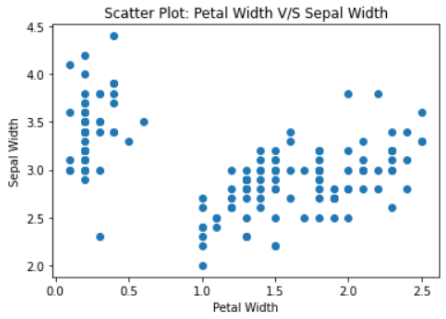
OUTPUT

```
In [5]: frame = pd.DataFrame(data.data, columns=data.feature_names)
        frame.head()
```

Out[5]:

	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

```
In [6]: sepalWidth = data.feature_names[1]
        petalWidth = data.feature_names[3]

In [7]: 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()
```

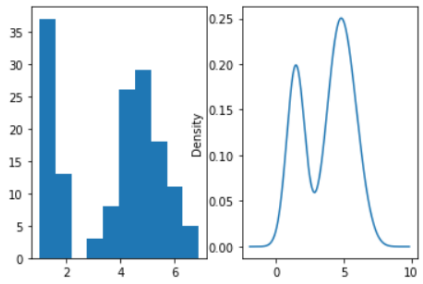


c.
```
petalLength = data.feature_names[2]
fig = plt.figure()
histGraph = fig.add_subplot(1, 2, 1)
plt.hist(frame[petalLength])
densityGraph = fig.add_subplot(1, 2, 2)
```

frame[petalLength].plot.density()

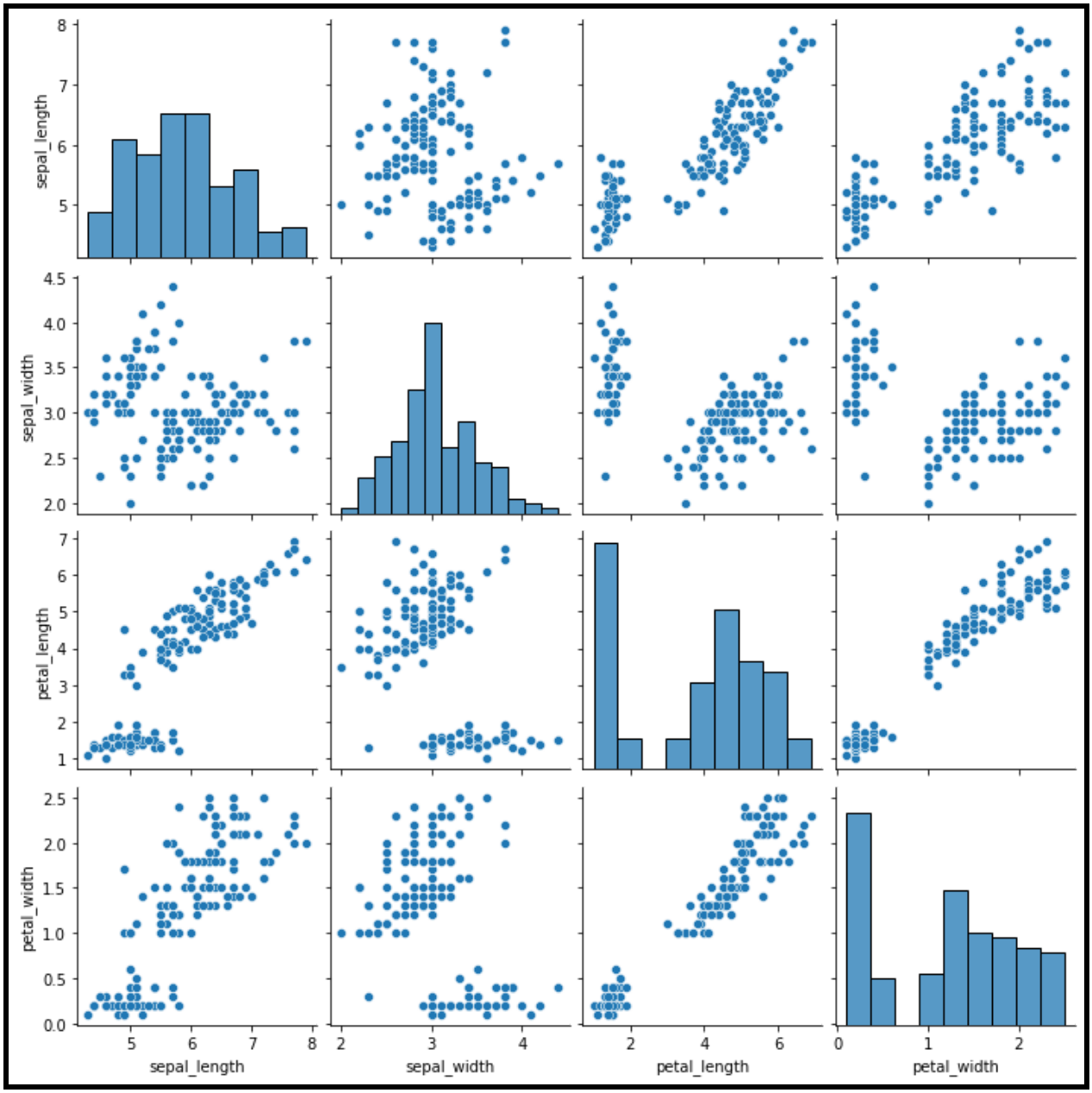
OUTPUT

```
In [8]: petalLength = data.feature_names[2]

In [9]: 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()

Out[9]: <AxesSubplot:ylabel='Density'>
```



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

OUTPUT


sepal_length
sepal_width
petal_length
petal_width
sepal_length
sepal_width
petal_length
petal_width

Question 6

Consider any sales training/ weather forecasting dataset.

a. **Compute the 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 dates.**
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')
dataset = frame.drop_duplicates(subset=['DateTime']).reset_index(drop=True)
dataset
```

OUTPUT

Out[3]:

	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()

OUTPUT

```
In [4]: dataset['T (degC)'].groupby(dataset['p (mbar)']).mean()

Out[4]: 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()

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

OUTPUT

Out[6]:

	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

420766 rows × 15 columns

c. frame = dataset.copy()
frame.head(3)
frame['Date Time'].dtype
frame['Date Time'] = pd.to_datetime(frame['Date Time'])
frame['Date Time'].dtype

OUTPUT

In [7]:
```
frame = dataset.copy()
frame.head(3)
```

Out[7]:

	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

In [8]:
```
frame['Date Time'].dtype
```

Out[8]:
```
dtype('O')
```

In [10]:
```
frame['Date Time'] = pd.to_datetime(frame['Date Time'])
frame['Date Time'].dtype
```

Out[10]:
```
dtype('<M8[ns]')
```

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

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

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)

OUTPUT

Out[13]:

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
2011	August	14.763459
	June	14.404655
	July	13.769617
	May	13.033112
	September	12.054720
	April	10.747826
	October	7.792991
	March	7.348674
	December	5.805069
	November	5.533285
	January	3.352500
	February	2.567361

```
          September    13.695811
          May          12.735692
          October      11.090311
          April        10.943500
          March         9.270172
          February      7.704593
          November      7.203225
          December      6.266496
          January       3.790381
2015      July         16.807265
          August       16.360347
          June         14.078870
          September    12.551669
          May          12.326225
          April        10.518282
          December      8.778217
          October       8.563967
          November      7.890299
          March         7.482659
          February      5.494115
          January       4.893638
2016      August       16.424330
          July         15.952912
          June         14.785060
          September    13.607231
          May          12.514861
          October       9.242924
          April         8.056669
          March         6.732556
          November      6.582278
          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
```

OUTPUT

```
In [15]: bins = 5

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

         result
```

Out[15]:

T (degC)	(-23.07, -10.952]	(-10.952, 1.106]	(1.106, 13.164]	(13.164, 25.222]	(25.222, 37.28]
p (mbar)					
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

Question 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.**

```
import pandas as pd
import numpy as np
frame = pd.read_excel('students.xlsx')
frame.head()
```

OUTPUT

```
In [3]: frame.head()
```

Out[3]:

	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

a. pd.get_dummies(frame, columns=['Gender', 'Pass_Division'])

OUTPUT

```
In [4]: pd.get_dummies(frame, columns=['Gender', 'Pass_Division'])
```

Out[4]:

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

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

frame = frame.sort_values('Birth_Month', ascending=False)
frame
```

OUTPUT

Out[5]:

	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

Question 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.**

```
import numpy as np
import pandas as pd
income = pd.read_excel('Monthly-Income.xlsx')
frame = pd.DataFrame(income)
frame
```

OUTPUT

```
In [3]: frame = pd.DataFrame(income)
        frame
```

Out[3]:

	Name	Gender	MonthlyIncome (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	Vats	Male	71900

a.
```
grouped = frame.groupby('Name')
grouped.agg('sum')
```

OUTPUT

```
In [4]: grouped = frame.groupby('Name')
        grouped.agg('sum')
```

Out[4]:

Name	MonthlyIncome (Rs.)
Kumar	253530
Shah	281400
Vats	335050

b. grouped = frame.groupby('Name')
grouped.agg('max')

OUTPUT

```
In [5]: grouped = frame.groupby('Name')
        grouped.agg('max')
```

Out[5]:

Name	Gender	MonthlyIncome (Rs.)
Kumar	Male	103000
Shah	Male	114000
Vats	Male	155000

c. frame[frame['MonthlyIncome (Rs.)'] > 60000.00]

OUTPUT

```
In [6]: frame[frame['MonthlyIncome (Rs.)'] > 60000.00]
```

Out[6]:

	Name	Gender	MonthlyIncome (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	Vats	Male	71900

d. frame[(frame['Name'] == 'Shah') & (frame['Gender'] == 'Female')].mean()

OUTPUT

```
In [7]: frame[(frame['Name'] == 'Shah') & (frame['Gender'] == 'Female')].mean()

<ipython-input-7-8726a28f5e5b>:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid columns before calling the reduction.
  frame[(frame['Name'] == 'Shah') & (frame['Gender'] == 'Female')].mean()

Out[7]: MonthlyIncome (Rs.)    112400.0
dtype: float64
```