LAB EXERCISE – 1

Descriptive Statistics

Aim of the Experiment

The main aim of this experiment is to explore the given dataset. A sample database is created and is available in the file sample.csv.

Sample Dataset

id	first	last	gender	Marks	selected
1	Leone	Debrick	Female	50	TRUE
2	Romola	Phinnessy	Female	60	FALSE
3	Geri	Prium	Male	65	FALSE
4	Sandy	Doveston	Female	95	FALSE
5	Jacenta	Jansik	Female	31	TRUE
6	Diane-marie	Medhurst	Female	45	TRUE
7	Austen	Pool	Male	45	TRUE
8	Vanya	Teffrey	Male	70	FALSE
9	Giordano	Elloy	Male	36	FALSE
10	Rozele	Fawcett	Female	50	FALSE

The objectives of this experiment are:

1. Explore all the statistical operations of Pandas and given in Listing 1
2. Use Describe command and explore the dataset as given in Listing 2
3. Use Descriptive Statistics for univariate and bivariate data as given in Listing 3

Reference to the Textbook

All the fundamentals are given in Chapter 2 and Appendix 2.

Listing 1

```
import pandas as pd

col_list=["id","first","last","gender","Marks","selected"]

df = pd.read_csv("sample.csv",usecols=col_list)

print(df)


mean1 = df['Marks'].mean()

sum1 = df['Marks'].sum()
```

```
max1 = df['Marks'].max()

min1 = df['Marks'].min()

count1 = df['Marks'].count()

median1 = df['Marks'].median()

sd1 = df['Marks'].std()

var1 = df['Marks'].var()


print('Mean Marks\n' + str(mean1))

print('Sum of the Marks\n' + str(sum1))

print('Maximum of the marks\n' + str(max1))

print('Minimum of the marks\n' + str(min1))

print('Count of the marks\n' + str(count1))

print('Standard deviation of the marks\n' + str(sd1))

print('Variance of the marks\n' + str(var1))

print('End of Summary \n\n\n')
```

Output

```
In [13]: runfile('D:/Test/listing 1.py', wdir='D:/Test')
   id        first       last  gender  Marks  selected
0   1        Leone    Debrick  Female     50      True
1   2       Romola  Phinnessy  Female     60     False
2   3         Geri      Prium    Male     65     False
3   4        Sandy   Doveston  Female     95     False
4   5      Jacenta     Jansik  Female     31      True
5   6  Diane-marie   Medhurst  Female     45      True
6   7       Austen       Pool    Male     45      True
7   8        Vanya    Teffrey    Male     70     False
8   9     Giordano      Elloy    Male     36     False
9  10       Rozele    Fawcett  Female     50     False
Mean Marks
54.7
Sum of the Marks
547
Maximum of the marks
95
Minimum of the marks
31
Count of the marks
10
Standard deviation of the marks
18.666964283341724
Variance of the marks
348.4555555555555
End of Summary
```

Listing 2

```
import pandas as pd
col_list=["id","first","last","gender","Marks","selected"]
df = pd.read_csv("sample.csv",usecols=col_list)
print(df)
print(df.shape)
print(df.head(5))
print(df.describe())
```

Output

```
   id        first       last  gender  Marks  selected
0   1        Leone    Debrick  Female     50      True
1   2       Romola  Phinnessy  Female     60     False
2   3         Geri      Prium    Male     65     False
3   4        Sandy   Doveston  Female     95     False
4   5      Jacenta     Jansik  Female     31      True
5   6  Diane-marie   Medhurst  Female     45      True
6   7       Austen       Pool    Male     45      True
7   8        Vanya    Teffrey    Male     70     False
8   9     Giordano      Elloy    Male     36     False
9  10       Rozele    Fawcett  Female     50     False
(10, 6)
   id    first       last  gender  Marks  selected
0   1    Leone    Debrick  Female     50      True
1   2   Romola  Phinnessy  Female     60     False
2   3     Geri      Prium    Male     65     False
3   4    Sandy   Doveston  Female     95     False
4   5  Jacenta     Jansik  Female     31      True
             id      Marks
count  10.00000  10.000000
mean    5.50000  54.700000
std     3.02765  18.666964
min     1.00000  31.000000
25%     3.25000  45.000000
50%     5.50000  50.000000
75%     7.75000  63.750000
max    10.00000  95.000000
```

Listing 3

```
# Listing 3
# Illustrates the use of univariate, bivariate Statistics
```

```
from scipy import stats
data = [1,2,3,4,5,6,8,8,8,8,9,10,10,10,12,14,18,18,18,22,39,44,55,55,55,55,66,78,79,88]

print("\n Details of the data \n", stats.describe(data))
print("\n The cumulative frequency of the data-n",stats.cumfreq(data))
print("\n The Geometric mean of the data-n",stats.gmean(data))
print("\n The Harmonic mean of the data-n",stats.hmean(data))
print("\n The IQR of the data-n",stats.iqr(data))
print("\n The Zscore of the data-n",stats.zscore(data))
print("\n The skewness of the data-n",stats.skew(data))
print("\n The Kurtosos of the data-n",stats.kurtosis(data))

# Check correlation of the data

data = [1,2,3,6,8,10]
data1 = [2,3,4,5,9,12]

print("\n The Spearman correlation of the data-n",stats.spearmanr(data,data1))
```

Output

```
In [5]: runfile('D:/Test/Lab1-listing three.py', wdir='D:/Test')

 Details of the data
 DescribeResult(nobs=30, minmax=(1, 88), mean=26.933333333333334, variance=715.3057471264367, skewness=0.9591014772240082,
kurtosis=-0.5039593211083249)

 The cumulative frequency of the data-n CumfreqResult(cumcount=array([ 5., 16., 20., 20., 22., 22., 26., 27., 29., 30.]),
lowerlimit=-3.833333333333333, binsize=9.666666666666666, extrapoints=0)

 The Geometric mean of the data-n 15.260661495888634

 The Harmonic mean of the data-n 7.690170827943984

 The IQR of the data-n 44.25

 The Zscore of the data-n [-0.9862207  -0.94819162 -0.91016255 -0.87213347 -0.8341044  -0.79607532
 -0.72001717 -0.72001717 -0.72001717 -0.72001717 -0.68198809 -0.64395902
 -0.64395902 -0.64395902 -0.56790087 -0.49184271 -0.33972641 -0.33972641
 -0.33972641 -0.18761011  0.45888418  0.64902956  1.06734939  1.06734939
  1.06734939  1.06734939  1.48566923  1.94201814  1.98004721  2.3223089 ]

 The skewness of the data-n 0.9591014772240082

 The Kurtosos of the data-n -0.5039593211083249

 The Spearman correlation of the data-n SpearmanrResult(correlation=1.0, pvalue=0.0)
```

LAB EXERCISE – 2

Data Preprocessing

Aim of the Experiment.

The main aim of this experiment is to preprocess the given dataset. The database is created and is available in the file sample.csv.

Sample Dataset

id	first	last	gender	Marks	selected
1	Leone	Debrick	Female	50	TRUE
2	Romola	Phinnessy	Female	60	FALSE
3	Geri	Prium	Male	65	FALSE
4	Sandy	Doveston	Female	95	FALSE
5	Jacenta	Jansik	Female	31	TRUE
6	Diane-marie	Medhurst	Female	45	TRUE
7	Austen	Pool	Male	45	TRUE
8	Vanya	Teffrey	Male	70	FALSE
9	Giordano	Elloy	Male	36	FALSE
10	Rozele	Fawcett	Female	50	FALSE

The objectives of this experiment are

1. Explore Label Encoder
2. Explore Scikit Preprocessing routines like Scaling
3. Explore Scikit Preprocessing routines like Binarizer

Reference to the Textbook and Explanation

All the fundamentals are given in Chapter 2 and Appendix 2.

The variable in the dataset Female and Male can be changed to 0 or 1 using Label Encoder. It is done as given below:

```
df_gender_encode=LabelEncoder()

df.gender=df_gender_encode.fit_transform(df.gender)
```

Scaling can be done as follows:

```
df.Marks = preprocessing.scale(df.Marks)

scaled_df= preprocessing.scale(df.Marks)
```

Scaling removes the mean

Binarization uses threshold and converts values to binary as shown below:

scaled_df_bin = preprocessing.Binarizer(threshold=0.5).transform(newarr)

Duplicates can be removed as follows:

df_duplicates_removed = pd.DataFrame.drop_duplicates(df_duplicated)

The NaN of a column can be removed as shown below:

df['m5']=df['m5'].fillna(0)

This removes all the NaN to zero.

The command,

df=df.dropna(axis=1)

removes all the columns that has NaN.

Listing 1

```
import pandas as pd
col_list=["id","first","last","gender","Marks","selected"]
df = pd.read_csv("sample.csv",usecols=col_list)
print(df)
print("End of Listing\n\n\n")

# Let us convert the in Gender column, make Female as 0 and
# male as 1 using LabelEncoder in scikitlearn method

from sklearn.preprocessing import LabelEncoder
df_gender_encode=LabelEncoder()
df.gender=df_gender_encode.fit_transform(df.gender)
# One can observe that female is coded as 0 and Male as 1
print(df)
print("End of Listing\n\n\n")

# Now one can scale the marks to remove mean
```

```
from sklearn import preprocessing
df.Marks = preprocessing.scale(df.Marks)
scaled_df= preprocessing.scale(df.Marks)
print(df)
print("Scaling of marks is completed\n\n\n\n")


newarr = scaled_df.reshape(-1,1)
scaled_df_bin = preprocessing.Binarizer(threshold=0.5).transform(newarr)
df['Marks']=scaled_df_bin
print(df)
print("Binarizarion of marks is completed\n\n\n\n")
```

Output

```
In [63]: runfile('D:/Test/listing 2.py', wdir='D:/Test')
   id        first       last  gender  Marks  selected
0   1        Leone    Debrick  Female     50      True
1   2       Romola  Phinnessy  Female     60     False
2   3         Geri      Prium    Male     65     False
3   4        Sandy   Doveston  Female     95     False
4   5      Jacenta     Jansik  Female     31      True
5   6  Diane-marie   Medhurst  Female     45      True
6   7       Austen       Pool    Male     45      True
7   8        Vanya    Teffrey    Male     70     False
8   9     Giordano      Elloy    Male     36     False
9  10       Rozele    Fawcett  Female     50     False
End of Listing
```

```
   id        first       last  gender  Marks  selected
0   1        Leone    Debrick       0     50      True
1   2       Romola  Phinnessy       0     60     False
2   3         Geri      Prium       1     65     False
3   4        Sandy   Doveston       0     95     False
4   5      Jacenta     Jansik       0     31      True
5   6  Diane-marie   Medhurst       0     45      True
6   7       Austen       Pool       1     45      True
7   8        Vanya    Teffrey       1     70     False
8   9     Giordano      Elloy       1     36     False
9  10       Rozele    Fawcett       0     50     False
End of Listing
```

```
   id        first       last  gender     Marks  selected
0   1        Leone    Debrick       0 -0.265401      True
1   2       Romola  Phinnessy       0  0.299282     False
2   3         Geri      Prium       1  0.581624     False
3   4        Sandy   Doveston       0  2.275674     False
4   5      Jacenta     Jansik       0 -1.338300      True
5   6  Diane-marie   Medhurst       0 -0.547743      True
6   7       Austen       Pool       1 -0.547743      True
7   8        Vanya    Teffrey       1  0.863966     False
8   9     Giordano      Elloy       1 -1.055958     False
9  10       Rozele    Fawcett       0 -0.265401     False
Scaling of marks is completed
```

```
   id        first       last  gender  Marks  selected
0   1        Leone    Debrick       0    0.0      True
1   2       Romola  Phinnessy       0    0.0     False
2   3         Geri      Prium       1    1.0     False
3   4        Sandy   Doveston       0    1.0     False
4   5      Jacenta     Jansik       0    0.0      True
5   6  Diane-marie   Medhurst       0    0.0      True
6   7       Austen       Pool       1    0.0      True
7   8        Vanya    Teffrey       1    1.0     False
8   9     Giordano      Elloy       1    0.0     False
9  10       Rozele    Fawcett       0    0.0     False
Binarizarion of marks is completed
```

Listing 2

```
import pandas as pd
col_list=["id","first","last","gender","Marks","selected"]
df = pd.read_csv("sample.csv",usecols=col_list)
print(df)
print("End of Listing\n\n\n")

# Let us create duplicate elements in the given dataset
# This is done using the command concate 2 times as given below

df_duplicated = pd.concat([df]*2, ignore_index=True)
print(df_duplicated)

print("Display before duplication\n\n\n\n")

df_duplicates_removed = pd.DataFrame.drop_duplicates(df_duplicated)
print(df_duplicates_removed)

print("Display after duplication\n\n\n\n")
```

Output

```
   id        first       last  gender  Marks  selected
0   1        Leone    Debrick  Female     50      True
1   2       Romola  Phinnessy  Female     60     False
2   3         Geri      Prium    Male     65     False
3   4        Sandy   Doveston  Female     95     False
4   5      Jacenta     Jansik  Female     31      True
5   6  Diane-marie   Medhurst  Female     45      True
6   7       Austen       Pool    Male     45      True
7   8        Vanya    Teffrey    Male     70     False
8   9     Giordano      Elloy    Male     36     False
9  10       Rozele    Fawcett  Female     50     False
End of Listing
```

```
    id        first       last  gender  Marks  selected
0    1        Leone    Debrick  Female     50      True
1    2       Romola  Phinnessy  Female     60     False
2    3         Geri      Prium    Male     65     False
3    4        Sandy   Doveston  Female     95     False
4    5      Jacenta     Jansik  Female     31      True
5    6  Diane-marie   Medhurst  Female     45      True
6    7       Austen       Pool    Male     45      True
7    8        Vanya    Teffrey    Male     70     False
8    9     Giordano      Elloy    Male     36     False
9   10       Rozele    Fawcett  Female     50     False
10   1        Leone    Debrick  Female     50      True
11   2       Romola  Phinnessy  Female     60     False
12   3         Geri      Prium    Male     65     False
13   4        Sandy   Doveston  Female     95     False
14   5      Jacenta     Jansik  Female     31      True
15   6  Diane-marie   Medhurst  Female     45      True
16   7       Austen       Pool    Male     45      True
17   8        Vanya    Teffrey    Male     70     False
18   9     Giordano      Elloy    Male     36     False
19  10       Rozele    Fawcett  Female     50     False
Display before duplication
```

```
   id        first       last  gender  Marks  selected
0   1        Leone    Debrick  Female     50      True
1   2       Romola  Phinnessy  Female     60     False
2   3         Geri      Prium    Male     65     False
3   4        Sandy   Doveston  Female     95     False
4   5      Jacenta     Jansik  Female     31      True
5   6  Diane-marie   Medhurst  Female     45      True
6   7       Austen       Pool    Male     45      True
7   8        Vanya    Teffrey    Male     70     False
8   9     Giordano      Elloy    Male     36     False
9  10       Rozele    Fawcett  Female     50     False
Display after duplication
```

Listing 3

```
import pandas as pd
df = pd.DataFrame({
        'm1':[50,'A',60,'A',80],
        'm2':[60,'A','60','A',80],
        'm3':[50,70,'A','A',60],
        'm4':[60,'A','A','A',60],
        'm5':['A','A','A',10,20]
        })

df = df.apply(pd.to_numeric,errors='coerce')

print(df)

print('Dataframe with NaN\n\n\n')
# Make all the NaN in Mark5 as zero
df['m5']=df['m5'].fillna(0)
print(df)
print('Making m5 NaN as 0 using fillna() function\n\n\n\n')

df1 = df.copy()
df1['m2'].fillna(df1['m2'].mean(),inplace=True)
print(df1)
print('Making m5 NaN as mean using fillna() function\n\n\n\n')

df2 = df.copy()
df1['m3'].fillna(df1['m2'].median(),inplace=True)
print(df2)
print('Making m5 NaN as median using fillna() function\n\n\n\n')
```

```
# Dropping all columns having NaN

df=df.dropna(axis=1)

print(df)

print('Dropping all columns having NaN\n\n\n\n')
```

Output

```
In [104]: runfile('D:/Test/listing 2B.py', wdir='D:/Test')
     m1    m2    m3    m4    m5
0  50.0  60.0  50.0  60.0   NaN
1   NaN   NaN  70.0   NaN   NaN
2  60.0  60.0   NaN   NaN   NaN
3   NaN   NaN   NaN   NaN  10.0
4  80.0  80.0  60.0  60.0  20.0
Dataframe with NaN
```

```
     m1    m2    m3    m4    m5
0  50.0  60.0  50.0  60.0   0.0
1   NaN   NaN  70.0   NaN   0.0
2  60.0  60.0   NaN   NaN   0.0
3   NaN   NaN   NaN   NaN  10.0
4  80.0  80.0  60.0  60.0  20.0
Making m5 NaN as 0 using fillna() function



     m1         m2    m3    m4    m5
0  50.0  60.000000  50.0  60.0   0.0
1   NaN  66.666667  70.0   NaN   0.0
2  60.0  60.000000   NaN   NaN   0.0
3   NaN  66.666667   NaN   NaN  10.0
4  80.0  80.000000  60.0  60.0  20.0
Making m5 NaN as mean using fillna() function
```

```
     m1         m2    m3    m4    m5
0  50.0  60.000000  50.0  60.0   0.0
1   NaN  66.666667  70.0   NaN   0.0
2  60.0  60.000000   NaN   NaN   0.0
3   NaN  66.666667   NaN   NaN  10.0
4  80.0  80.000000  60.0  60.0  20.0
Making m5 NaN as mean using fillna() function



     m1    m2    m3    m4    m5
0  50.0  60.0  50.0  60.0   0.0
1   NaN   NaN  70.0   NaN   0.0
2  60.0  60.0   NaN   NaN   0.0
3   NaN   NaN   NaN   NaN  10.0
4  80.0  80.0  60.0  60.0  20.0
Making m5 NaN as median using fillna() function
```

```
     m5
0   0.0
1   0.0
2   0.0
3  10.0
4  20.0
Dropping all columns having NaN
```

Listing 4

This listing illustrates the use of MinMax scaling and Standard scaling for finding Z-scores.

```
from numpy import asarray
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler


data = asarray([[1,3],[8,5],[6,7],[8,9]])
print("\n Original Data")
print(data)
```

```
scaler1 = MinMaxScaler()

scaler2 = StandardScaler()


scaled1 = scaler1.fit_transform(data)

scaled2 = scaler2.fit_transform(data)


print("\n\nThe output of MinMax Scaling")

print(scaled1)



print("\n\nThe output of Standard scaling as z-score")

print(scaled2)
```

Output

```
In [26]: runfile('D:/Test/Lab2.minmax.py', wdir='D:/Test')

 Original Data
[[1 3]
 [8 5]
 [6 7]
 [8 9]]


The output of MinMax Scaling
[[0.         0.        ]
 [1.         0.33333333]
 [0.71428571 0.66666667]
 [1.         1.        ]]


The output of Standard scaling as z-score
[[-1.66003771 -1.34164079]
 [ 0.78633365 -0.4472136 ]
 [ 0.08737041  0.4472136 ]
 [ 0.78633365  1.34164079]]
```

LAB EXERCISE – 3

Graphics Plots

Aim of the Experiment

To Explore the Univariate and Bivariate Graphs

Textbook reference and Explanations

Chapter 2 and Appendix 2

A csv file is created for this Lab Exercise as shown below and is available in marks.csv file.

	A	B	C	D	E	F	G	H	I
1	id	name	gender	physics	chemistry	maths	biology	language	selected
2	1	Leone	Female	50	60	56	77	80	TRUE
3	2	Romola	Female	60	80	77	99	87	FALSE
4	3	Geri	Male	65	90	44	78	90	FALSE
5	4	Sandy	Female	95	90	44	97	97	FALSE
6	5	Jacenta	Female	31	90	56	98	98	TRUE
7	6	Diane-ma	Female	45	88	58	67	98	TRUE
8	7	Austen	Male	45	78	68	68	99	TRUE
9	8	Vanya	Male	70	79	79	69	28	FALSE
10	9	Giordano	Male	36	82	78	55	27	FALSE
11	10	Rozele	Female	50	89	78	48	27	FALSE
12									

The command df['physics'].hist() plots the histogram of the dataset. The generic form is df[column name].hist(). The command plt.scatter(chemistry1,marks1,alpha=0.5) plots the scatter diagram of chemistry and physics examination. df.plot.box(title="Box and whisker plot of Marks", grid=False) can plot the box and whisker plot. df.plots() and pd.plotting.scatter_matrix() can plot for the entire dataset.

Program Listings

Listing - 1

```
import pandas as pd

from matplotlib import pyplot as plt


col_list=['id','name','gender','physics','chemistry','maths','biology','language','selected']

df = pd.read_csv('marks1.csv',usecols=col_list)

print(df)

# Create a histogram for physics marks


marks1 = df['physics'].values.tolist()
```

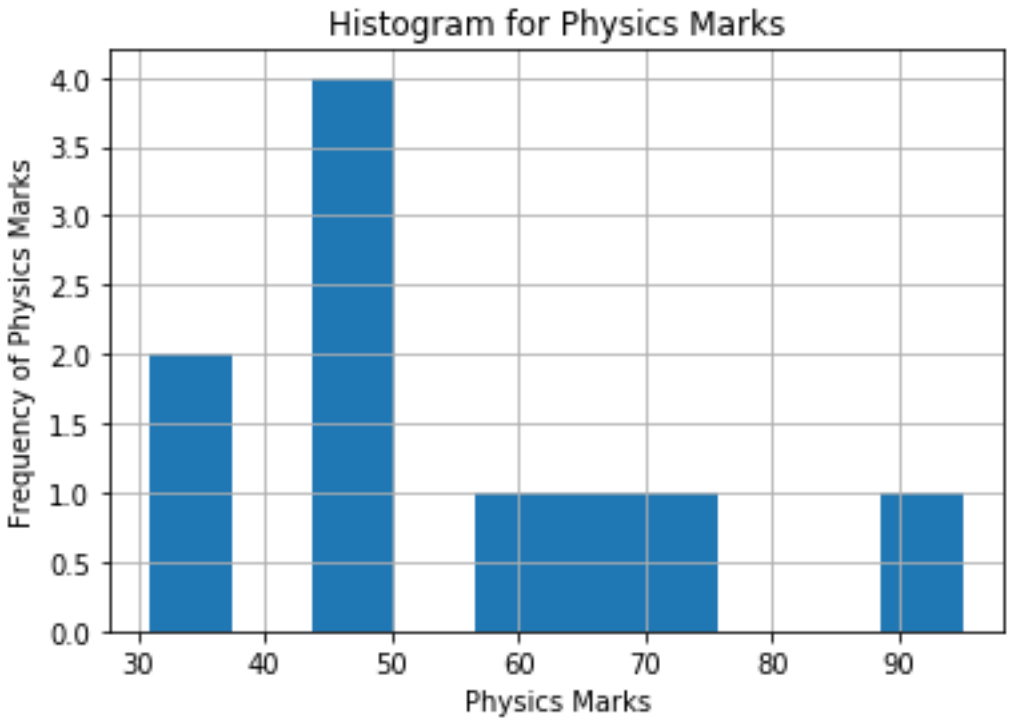
```
marks_scored = df['physics'].hist()
plt.title('Histogram for Physics Marks')
plt.xlabel('Physics Marks')
plt.ylabel('Frequency of Physics Marks')
plt.show()

# Create a scatter plot for Physics Vs Chemistry marks
chemistry1 = df['chemistry'].values.tolist()
print (chemistry1)

plt.scatter(chemistry1,marks1,alpha=0.5)
plt.title('Scatter plot chemistry vs physics')
plt.xlabel('Chemistry')
plt.ylabel('Physics')
plt.show()

# Create a Box plot
df.plot.box(title="Box and whisker plot of Marks", grid=False);
plt.show(block=True);

# Create a pie chart for selected based on Gender

sums = df.selected.groupby(df.gender).sum()
plt.pie(sums,labels=sums.index);
plt.show()


# Multiplots
# Box Plot for the dataset

df.plot(kind='box',subplots=True,layout=(3,3),sharex=False,sharey=False)
```

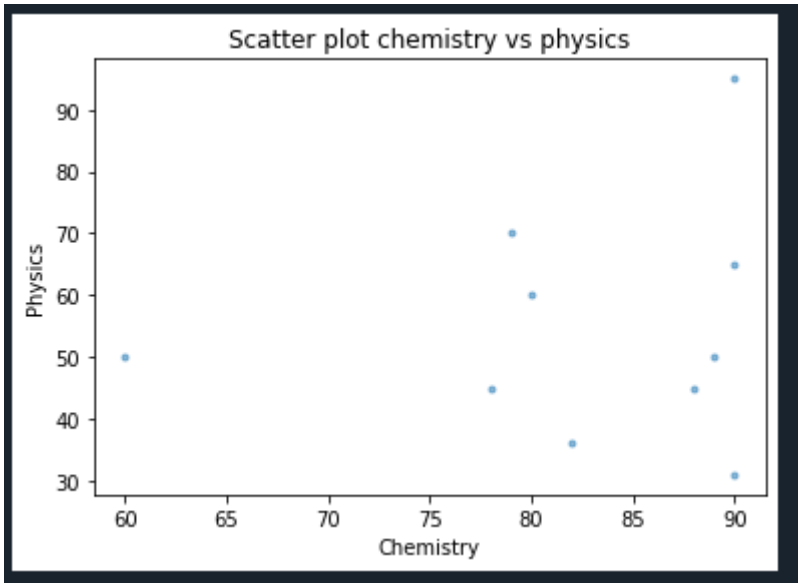
```
plt.show()
```

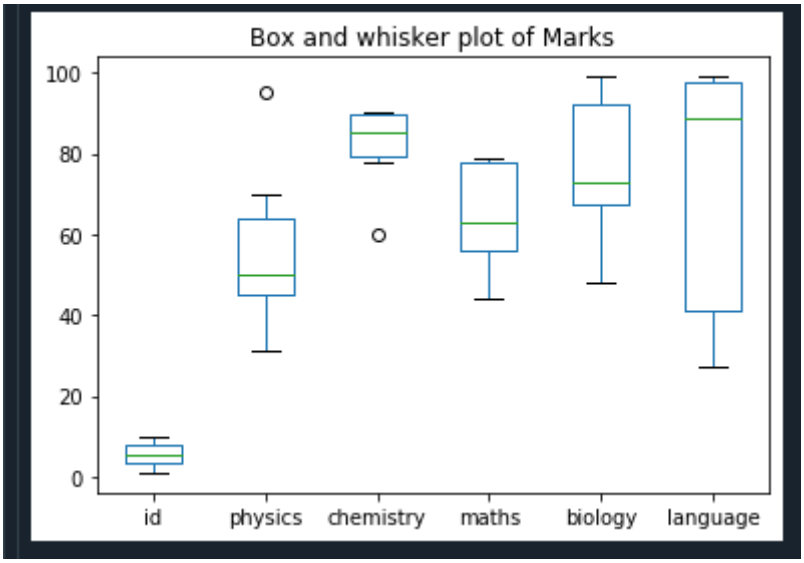
```
# Multiplot
# drop columns id,name,gender,selected

cols=df[['id','name','gender','selected']]
df2 = df.drop(cols, axis=1)
pd.plotting.scatter_matrix(df2, alpha=0.2)
plt.show()
```

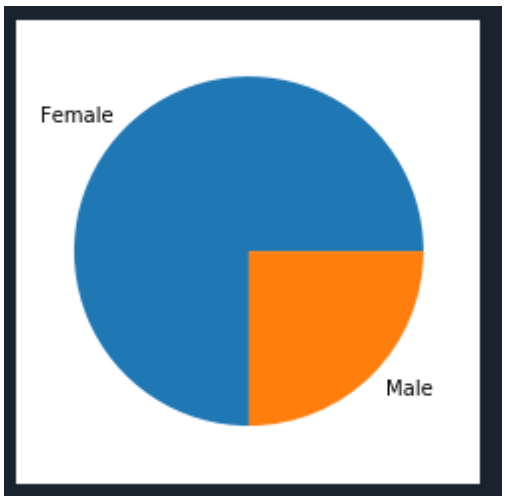
Output

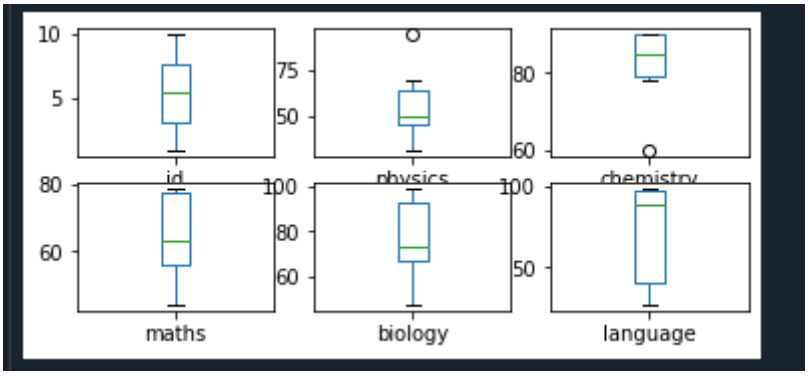
```
In [66]: runfile('D:/Test/lab3-univariate.py', wdir='D:/Test')
   id         name  gender  physics  ...  maths  biology  language  selected
0   1        Leone  Female       50  ...     56       77        80      True
1   2       Romola  Female       60  ...     77       99        87     False
2   3         Geri    Male       65  ...     44       78        90     False
3   4        Sandy  Female       95  ...     44       97        97     False
4   5      Jacenta  Female       31  ...     56       98        98      True
5   6  Diane-marie  Female       45  ...     58       67        98      True
6   7       Austen    Male       45  ...     68       68        99      True
7   8        Vanya    Male       70  ...     79       69        28     False
8   9     Giordano    Male       36  ...     78       55        27     False
9  10       Rozele  Female       50  ...     78       48        27     False
```

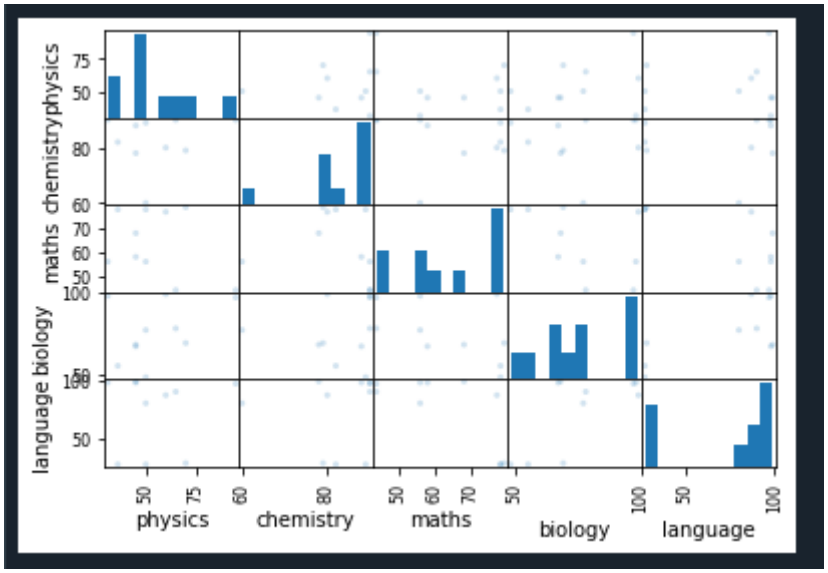












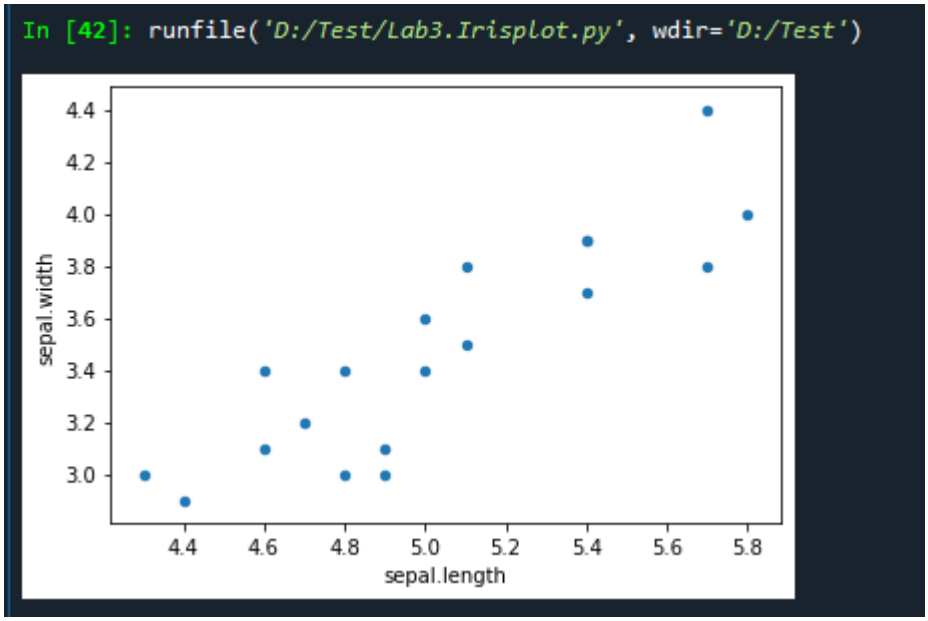
Listing 2

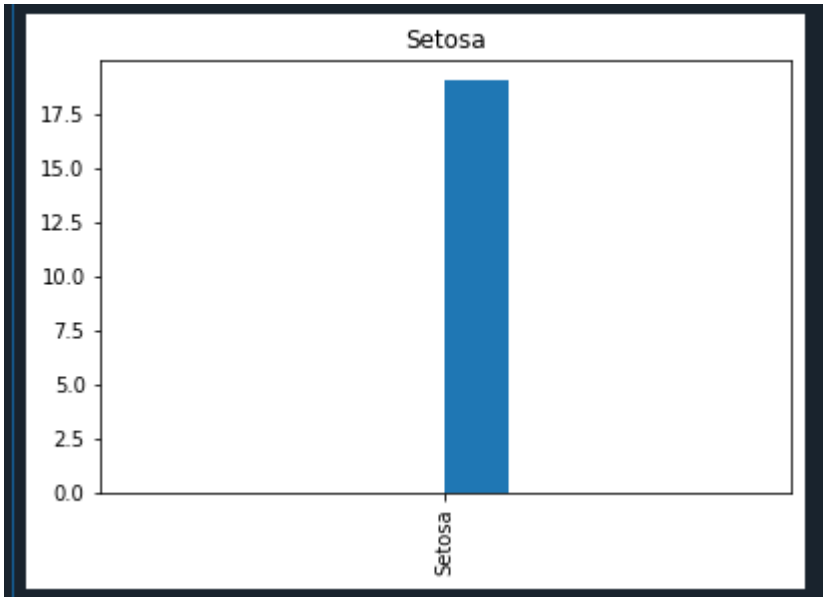
Explain Charts the using Iris dataset.

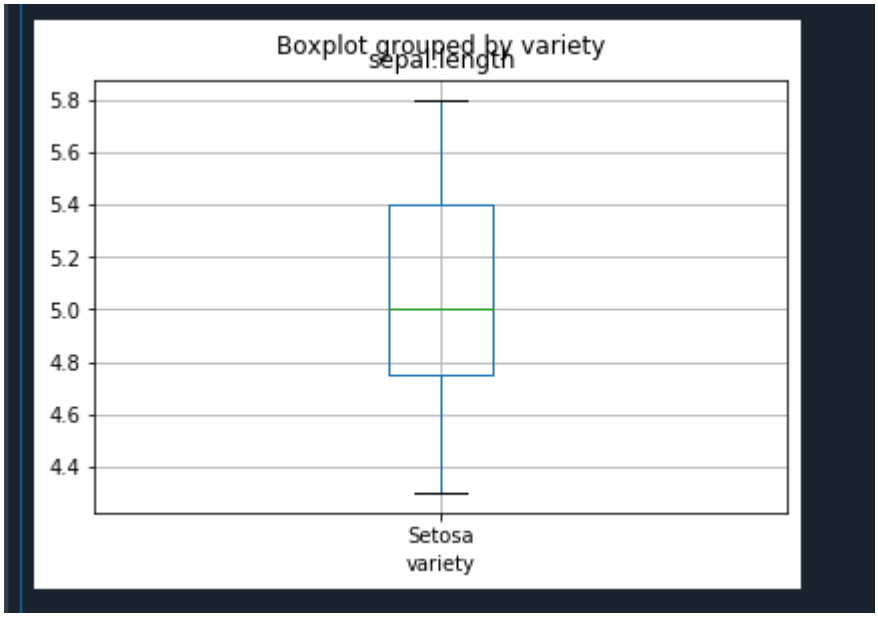
```
import pandas as pd

df = pd.read_csv("iris.csv")
data = df.iloc[1:20]

data.plot.scatter('sepal.length','sepal.width')

data.hist(column='variety',by='variety')
```

data.boxplot(column='sepal.length',by='variety')

Output







LAB EXERCISE – 4

Data Visualization using Seaborn

Aim of the Experiment

To write python program using Seaborn for data visualization. The data visualization is done for both synthetic data as well as for preloaded Iris dataset.

Reference to Textbook and Explanation

Chapter 2 and Appendix 2 for details about Data visualization.

The Bar plot for week Vs sales can be done as follows: Here hue is used to specify the colour. Set_title() is used to display the heading as part of the plot.

```
sns.barplot(x = "week", y = "sales", hue = "region", data = df) \
        .set_title('Bar plot for weeks Vs Sales Data')
```

The box plot can be done as follows:

```
sns.boxplot(data = df).set_title('Box Plot for week and sales data')
plt.show()
```

distplots() are used to display distribution plots.

```
sns.distplot(df1['sepal_width'],kde = False).set_title('Distribution plot for sepal_width')
plt.show()
```

Violin plot combines both box and distplot ability.

```
sns.violinplot(data = df).set_title('Violin Plot for week and sales data')
```

Count plots can be used to count the attributes as follows:

```
sns.countplot(x = "region", data = df, palette = "Blues").set_title('Count plot Based on regions')
```

Strip plots can be used to display the data based on certain attributes. Here, it is done as class vs petal length of Iris dataset.

```
sns.stripplot(x = "species", y = "petal_length", data = df1).set_title('Strip plot for petal length and class')
```

Joint plot can be used for bivariate distribution plot as shown below:

```
sns.jointplot(x = 'week',y = 'sales',data = df)
```

lmplot() can be used for creating regression plot of two attributes based on another attribute.

```
g1 = sns.lmplot(x="week", y="sales", hue="region", data=df)
```

Multivariate analysis can be done using pairplots for all the attributes of the dataset.

```
sns.pairplot(df1,hue = 'species',kind = "scatter")
```

Program Listing – 1

```
import seaborn as sns
import pandas as pd
from matplotlib import pyplot as plt

# Create a synthetic dataset

data = {'week': [1,2,3,4,5],
        'sales':[1.2,1.8,2.6,3.2,3.8],
        'region':['south','south','north','north','north']
       }

df = pd.DataFrame(data)

# Read the preloaded dataset Iris using data frame 1 - df1

df1 = sns.load_dataset('iris')

sns.set(color_codes=True)

### Univariate Plots

# Histogram plot using barplot

print("\nHistogram of week vs Sales\n")
sns.barplot(x = "week", y = "sales", hue = "region", data = df) \
          .set_title('Bar plot for weeks Vs Sales Data')
plt.show()
```

```
# Box Plot

print("\nBox Plot of week and sales data\n")
sns.boxplot(data=df)
plt.xlabel("Statistics", size=18)
plt.ylabel("Week and Sales Data", size=18)
plt.show()

# Distplot - Distribution plots

# Display the histogram by making kde flag as falsse
sns.distplot(df1['sepal_width'],kde = False).set_title('Distribution plot for sepal_width')
plt.xlabel("Sepal Width", size=18)
plt.ylabel("Frequency", size=18)
plt.show()

# Display the Distribution plot by making hist flag as false
sns.distplot(df1['sepal_width'],hist = False).set_title('Distribution plot for sepal_width')
plt.xlabel("Sepal Width", size=18)
plt.ylabel("Frequency", size=18)
plt.show()

# Violin Plot

print("\n Violin Plot of week and sales data\n")
sns.violinplot(data = df).set_title('Violin Plot for week and sales data')
plt.xlabel("Statistics", size=18)
plt.ylabel("Region", size=18)
plt.show()

# Count Plot for Iris Data
```

```
sns.countplot(x = "region", data = df, palette = "Blues").set_title('Count plot Based on regions')

plt.xlabel("Statistics", size=18)

plt.ylabel("Region", size=18)

plt.show()
```

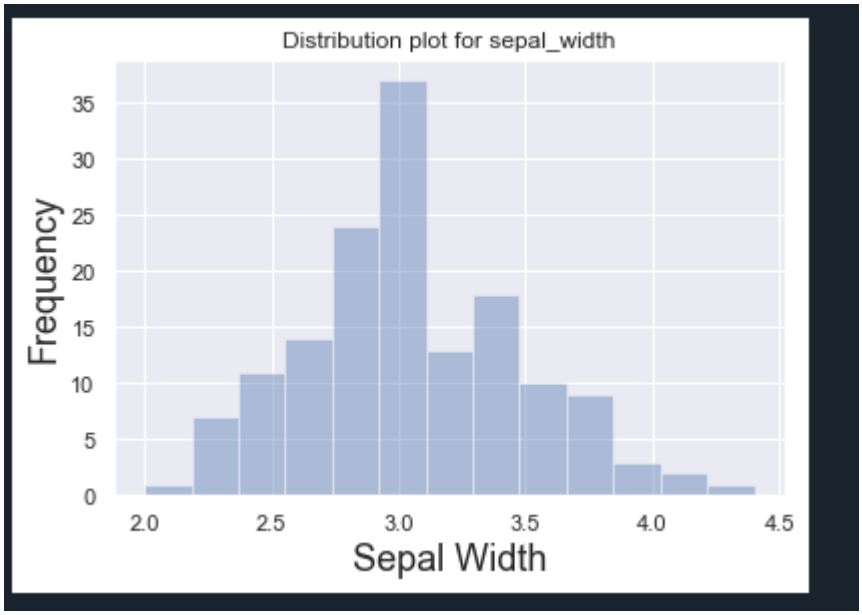
```
# Strip plot for Iris Data

print('\n Strip plot\n')

sns.stripplot(x = "species", y = "petal_length", data = df1).set_title('Strip plot for petal length and class')
```

Output

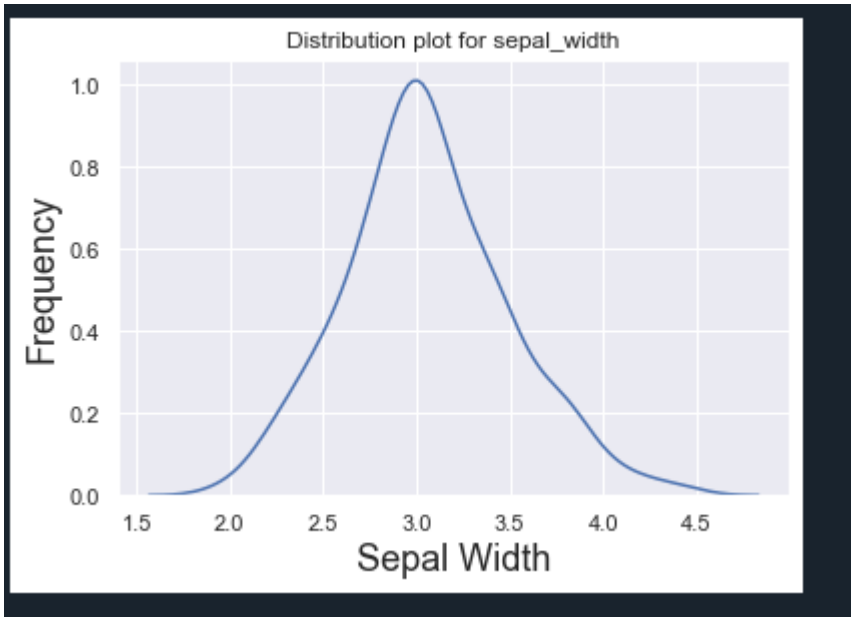
Histogram of week Vs sales

Box Plot for week and sales

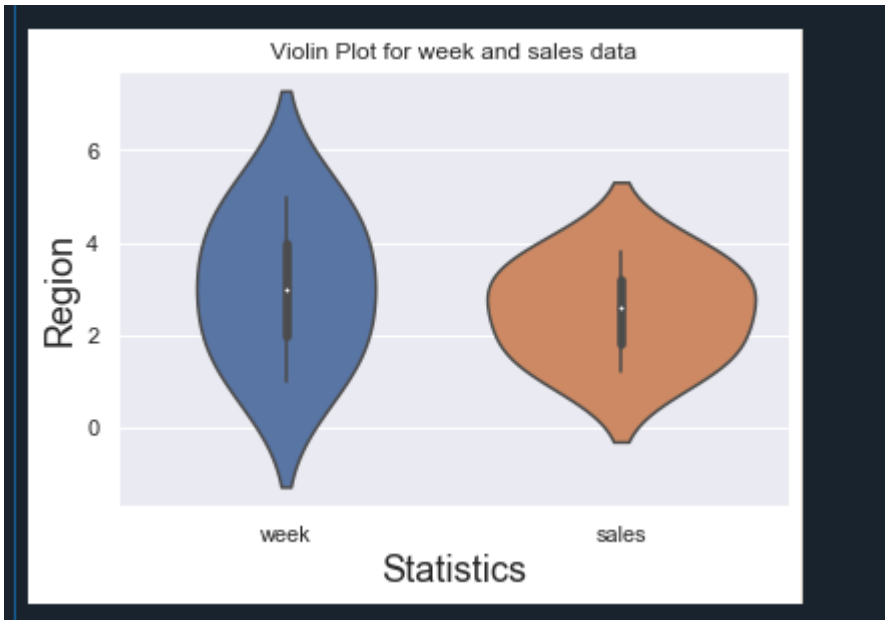
Distribution plot in histograms



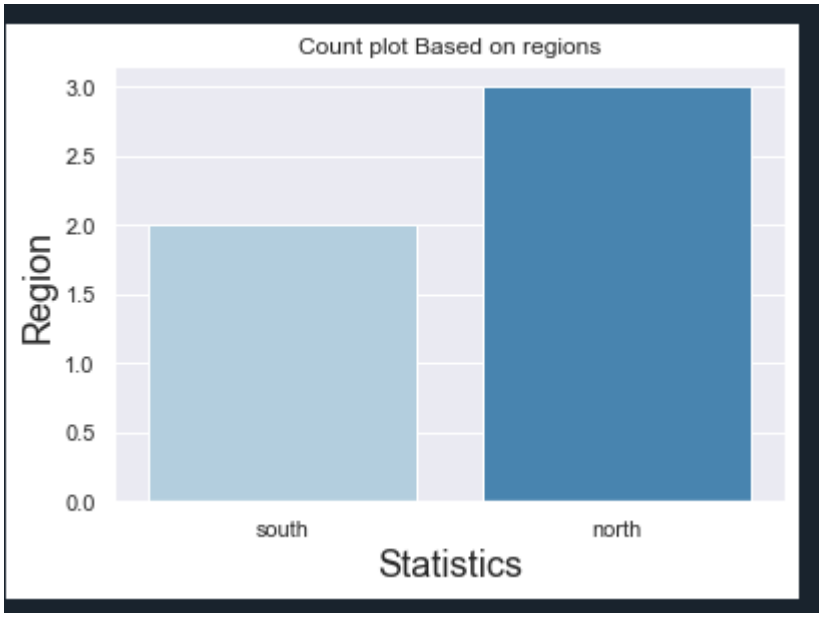
Distribution plot in kernel Distribution function



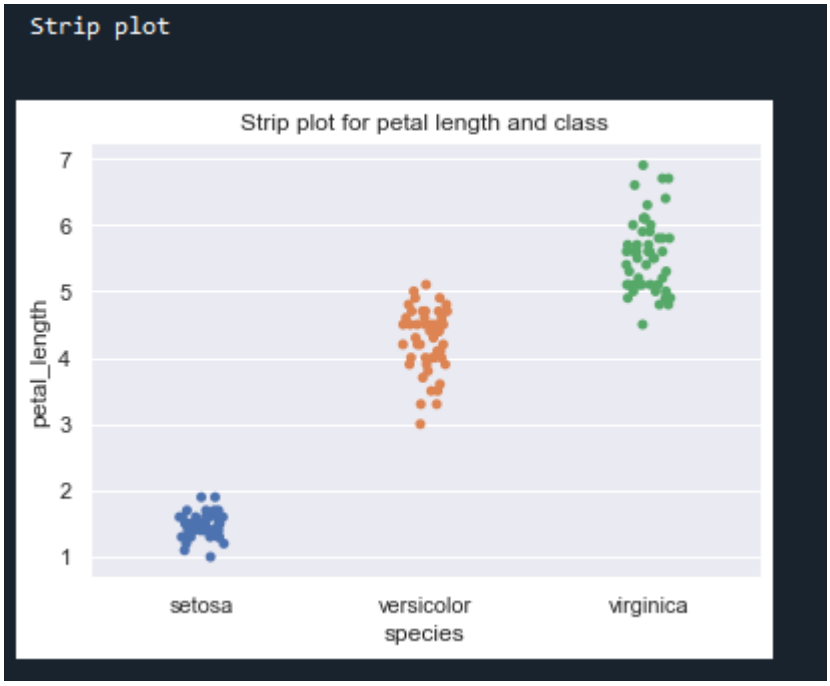
Violin Plot



Count plot of week and sales based on Region



Strip Plot

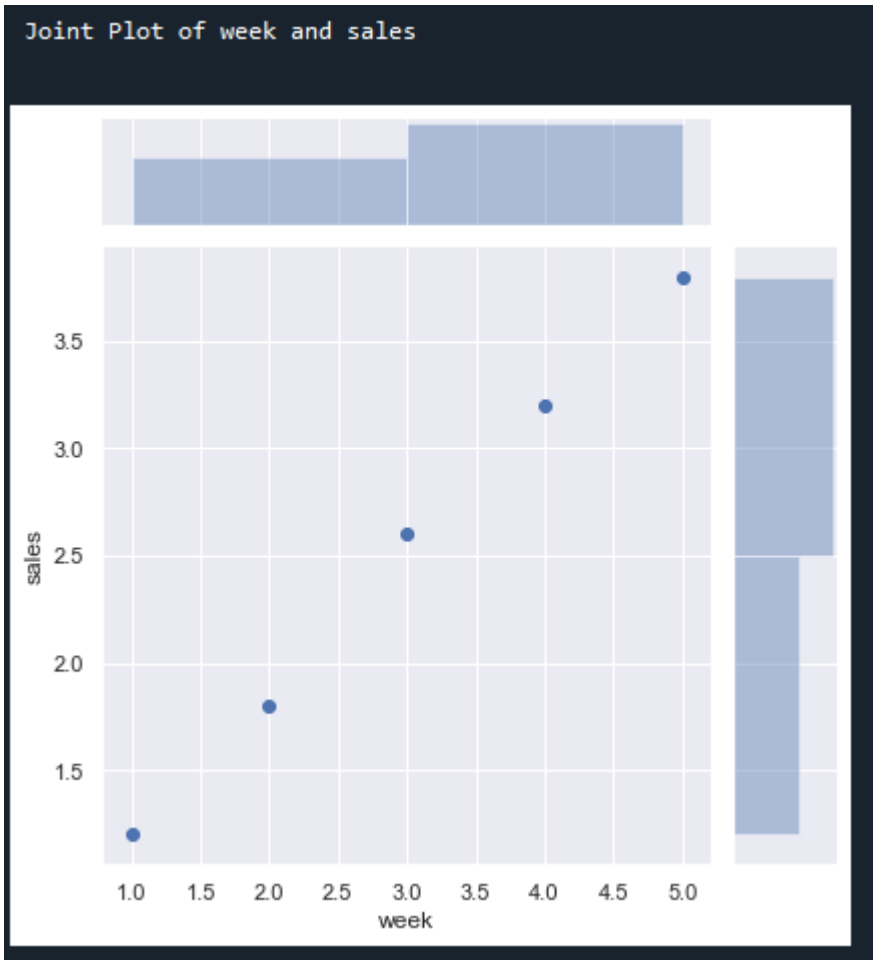


Program Listing 2

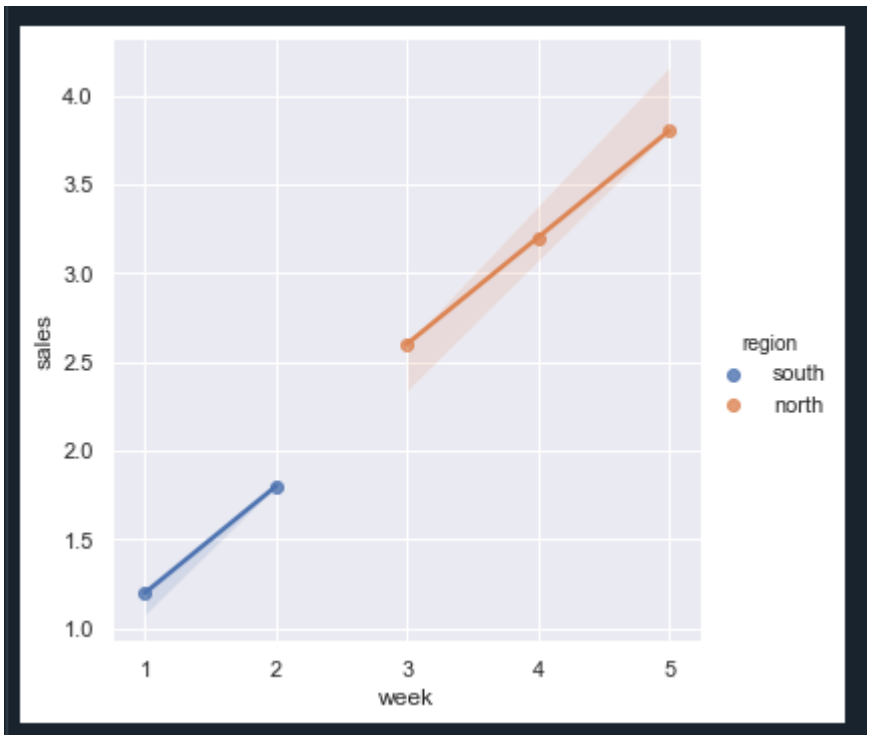
```
import seaborn as sns

import pandas as pd

from matplotlib import pyplot as plt


# Create a synthetic dataset
```

```
data = {'week': [1,2,3,4,5],
        'sales':[1.2,1.8,2.6,3.2,3.8],
        'region':['south','south','north','north','north']
       }

df = pd.DataFrame(data)

# Read the preloaded dataset Iris using data frame 1 - df1

df1 = sns.load_dataset('iris')

sns.set(color_codes=True)

### Bivariate Plots

# Joint Plot
print('\n Joint Plot of week and sales\n')
sns.jointplot(x = 'week',y = 'sales',data = df)
plt.show()

#lmplot

print('\n lmplot of week and sales Based on Region\n\n')
g1 = sns.lmplot(x="week", y="sales", hue="region", data=df)

### Multivariate Plots

# pairplot
sns.pairplot(df1,hue = 'species',kind = "scatter")
plt.show()
```

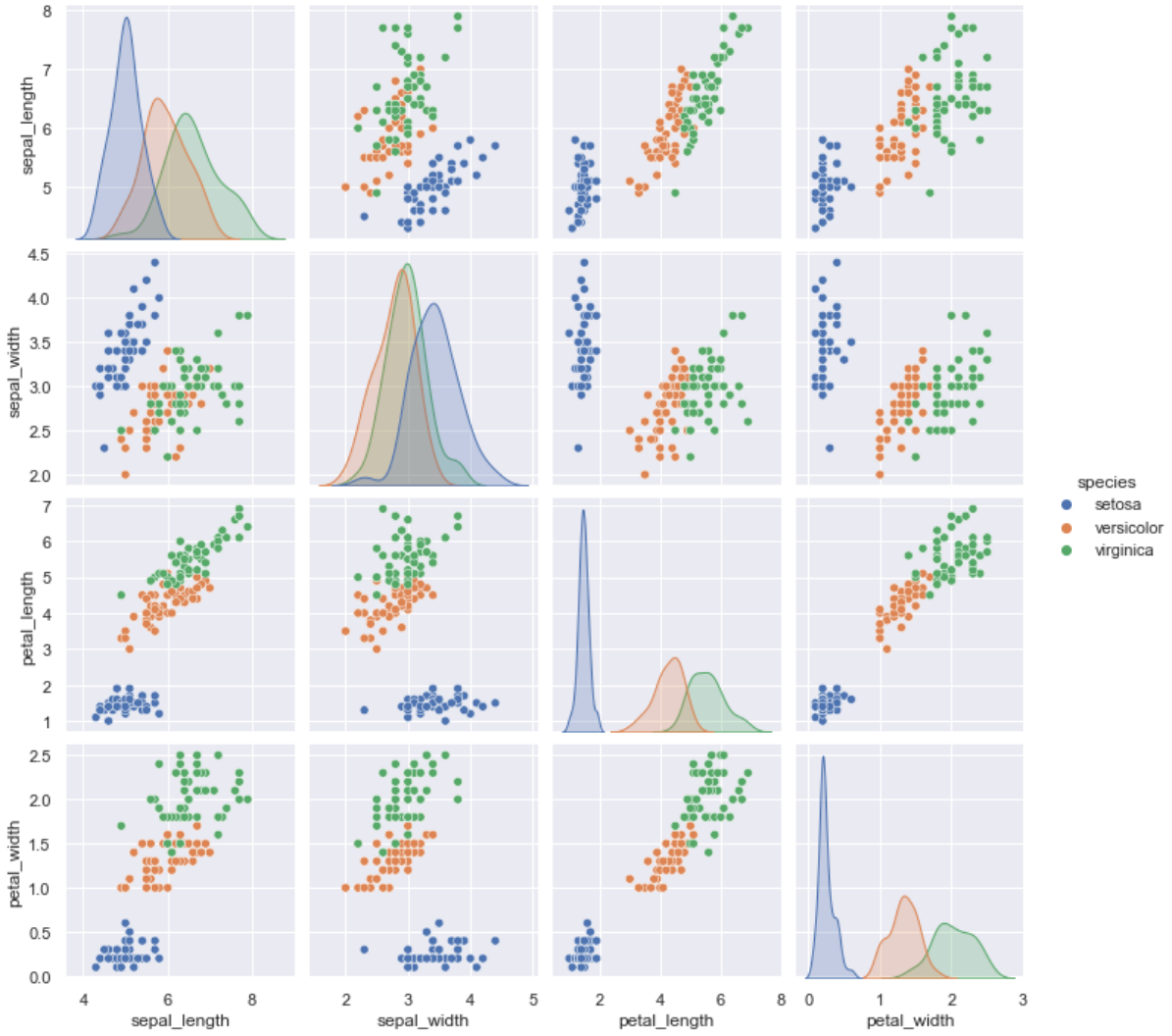
Output



Lmplot of week vs Sales based on region



Pairplot of week and sales



LAB EXERCISE – 5

Statistical Tests Using SCIPY

Aim of the Experiment

To write python program for finding Chi-square test and t-tests using SciPy module

Reference to Textbook and Explanation

Chapter 2 and Appendix 2 for details about Chi-square and t-tests.

The command,

x=chi_2(table) helps to find Chi-square test between the observed and expected values. Refer to the Problem 3.11 of chapter 3, its table is reproduced here for reference.

	Registered	**Not Registered**	**Total**
Boys	35	15	50
Girls	25	25	50
Total	60	40	100

Expectations Table

	Registered	**Not Registered**	**Total**
Boys	$\frac{50X60}{100}=30$	$\frac{50X40}{100}=20$	50

Girls	$\frac{50X60}{100}=30$	$\frac{50X40}{100}=20$	50
Total	60	40	100

Program Listing

```
from scipy.stats import chi2_contingency
from scipy.stats import chi2
from scipy.stats import ttest_ind


# Perform Chisquare Test


# Compute the Observed value


table = [ [35,15],
        [25,25]
        ]
print('\n Obervartion Table')
# Print the observed value
print(table)


#Perform the Chi square test
stat,p,dof,expected = chi2_contingency(table)


# Compute the expected value
print('\n')
print('\n Expectation Table')
```

```
print(expected)

# Set the parameters
prob = 0.90

# Find the critical value
critical = chi2.ppf(prob,dof)
# Print the critical value
print('\n')
print('The critical Value')
print(critical)

print('statistical value')
print(stat)

if abs(stat) >= critical:
   print('Stat value greater or equal to critical:Reject Null Hypotheis')
else:
   print('Stat value is less than critical: Accept Alt Hypothesis')

# Perform t-Tests

x = ([6,3,4,5,5,9,7,8])
y = ([4,4,1,3,5,6,2,7])

print('\n')
print('t-Test completed\n\n')
stat,p= ttest_ind(x,y,equal_var=True)
```

```
print('\n')

print('Statistical value')

print(stat)

print('p value')

print(p)
```

Output

Chi Square Test

```
In [38]: runfile('D:/Test/lab5-stattests.py', wdir='D:/Test')

 Obervartion Table
[[35, 15], [25, 25]]



 Expectation Table
[[30. 20.]
 [30. 20.]]


The critical Value
2.705543454095404
statistical value
3.375
Stat value greater or equal to critical:Reject Null Hypotheis
```

t-test

```
t-Test completed



Statistical value
1.860521018838127
p value
0.08394585608714018
```

LAB EXERCISE – 6

Principal Component Analysis

Aim of the Experiment

To write python program for finding principal component analysis (PCA) for the given problem given in Chapter 2 and to a randomly generated dataset.

Reference to Textbook and Explanation

Chapter 2 and Appendix 2 for details about principal component analysis (PCA).

Consider the dataset

$$\begin{pmatrix} 2 & 1 \\ 6 & 7 \end{pmatrix}$$

Apply PCA and Inverse Transform and Prove that they are similar.

In listing 2, the methods of computing mean matrix, covariance matrix, eigen values and eigen vectors computed are illustrated.

In listing 3, Iris dataset is taken and PCA is applied. It can be verified that after applying PCA, the cross score remains unchanged. That means, all the features of Iris are not important.

Listing 1

```
import numpy as np

from sklearn import decomposition

X = np.array([[2,6],[1,7]])


print("Orginal Matrx X and its Shape")

print(X)

print("Original Shape:",X.shape)

print("Original matrix\n\n")

# Apply Transform for X


pca = decomposition.PCA(n_components=2)

X_pca = pca.fit_transform(X)
```

```
print("Transformed Matrix and its Shape")
print(X_pca)
print("Transformed Shape:",X_pca.shape)
print("Transformed Matrix\n\n")

# Apply Inverse Transform

print("After Inverse Transform")
X_new=pca.inverse_transform(X_pca)
print(X_new)
print("After Inverse Transform\n\n\n")



# Explain variance
print('Explained variance\n')
print(pca.explained_variance_ratio_)
print('completed\n\n')

print('Singular values')
print(pca.singular_values_)
print('completed\n\n')
```

Output

```
Orginal Matrx X and its Shape
[[2 6]
 [1 7]]
Original Shape: (2, 2)
Original matrix


Transformed Matrix and its Shape
[[-7.07106781e-01  1.18606713e-17]
 [ 7.07106781e-01  1.18606713e-17]]
Transformed Shape: (2, 2)
Transformed Matrix


After Inverse Transform
[[2. 6.]
 [1. 7.]]
After Inverse Transform



Explained variance

[1.00000000e+00 2.81351049e-34]
completed


Singular values
[1.00000000e+00 1.67735223e-17]
completed
```

Listing 2

This explains how the eigen values and eigen vectors are calculated.

```
import numpy as np

from numpy.linalg import eig


# define a matrix

X = np.array([[3, 6], [4,7]])


print("Orginal Matrx X and its Shape")

print(X)


print("Original matrix Shape")

print("Original Shape:",X.shape)
```

```
# calculate the mean of each column
M = np.mean(X.T, axis=1)
print("\nMean matrix")
print(M)


# center columns by subtracting column means
C = X - M
print("\nCentre the matrix")
print(C)

# calculate covariance matrix of centered matrix
V = np.cov(C.T)
print("\nCovariance of the matrix\n")
print(V)

# eigendecomposition of covariance matrix
values, vectors = eig(V)
print('\n Eigen vectors')
print(vectors)

print('\n Eigen values')
print(values)
```

Output

```
In [10]: runfile('D:/Test/lab6-detailedpca.py', wdir='D:/Test')
Orginal Matrx X and its Shape
[[3 6]
 [4 7]]
Original matrix Shape
Original Shape: (2, 2)

Mean matrix
[3.5 6.5]

Centre the matrix
[[-0.5 -0.5]
 [ 0.5  0.5]]

Covariance of the matrix

[[0.5 0.5]
 [0.5 0.5]]

 Eigen vectors
[[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]

 Eigen values
[1.00000000e+00 1.11022302e-16]
```

Listing 3

```
import pandas as pd

import numpy as np

from sklearn.model_selection import KFold

from sklearn.model_selection import cross_val_score

from sklearn.neighbors import KNeighborsClassifier

from sklearn import decomposition

import seaborn as sns



df = pd.read_csv("iris.csv")

print(df.head(10))

array = df.values

X = array[:,0:4]

y = array[:,4]
```

```
kfold = KFold(n_splits=10)
model = KNeighborsClassifier(n_neighbors=3)
score = cross_val_score(model,X,y,cv=10)
print('\n\n')
print("Cross score before applying PCA\n")
print(score.mean())

print("Apply PCA now...")

pca = decomposition.PCA(n_components=1)
X_pca = pca.fit_transform(X)
core = cross_val_score(model,X_pca,y,cv=10)
print('\n\n')
print("Cross score After applying PCA\n")
print(score.mean())
```

Output

```
In [21]: runfile('D:/Test/Lab6A-PCA-Iris.py', wdir='D:/Test')
   sepal.length  sepal.width  petal.length  petal.width variety
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
5           5.4          3.9           1.7          0.4  Setosa
6           4.6          3.4           1.4          0.3  Setosa
7           5.0          3.4           1.5          0.2  Setosa
8           4.4          2.9           1.4          0.2  Setosa
9           4.9          3.1           1.5          0.1  Setosa



Cross score before applying PCA

0.9666666666666666
Apply PCA now...



Cross score After applying PCA

0.9666666666666666
```

If more attributes are removed, it leads to more information loss. So optimal replacement of features is required. In the above case, retaining only one component does not result in reduction of scores.

LAB EXERCISE – 7

Find – S Algorithm

1. Aim of the Experiment:

Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file and generate the final specific hypothesis.

2. Reference to Text book for Algorithms:

Refer to Section 3.4.5 in Chapter 3 Basics of Learning Theory to understand the working of the algorithm.

Listing 1:

Sample Dataset Used: Table 3.2 in that chapter.

Table 3.2: Training Dataset

CGPA	Interactiveness	Practical Knowledge	Communication Skills	Logical Thinking	Interest	Job Offer
≥9	Yes	Excellent	Good	Fast	Yes	Yes
≥9	Yes	Good	Good	Fast	Yes	Yes
≥8	No	Good	Good	Fast	No	No
≥9	Yes	Good	Good	Slow	No	Yes

3. Python Program with Explanation:

1. Import the package csv to work with .csv files.

```
import csv
```

2. The attribute values of each attribute in the given dataset are included in the list 'attr_values'.

```
attr_values = [['g9','g8'],['Yes','No'],['Excellent','Good'],['Good','Bad'],
        ['Fast','Slow'],['Yes','No']]
```

3. The number of attributes are counted in the list with the len() function.

```
num_attrs = len(attr_values)
```

4. The most general hypothesis and the most specific hypothesis are printed.

```
print (" \n The most general hypothesis : ['?','?','?','?','?','?']\n")
print ("\n The most specific hypothesis : ['0','0','0','0','0','0']\n")
```

5. Create a dynamic array a[]. It is also called as a list.

```
a = []
```

6. Print the training dataset reading from the .csv file. First open the CSV file as a text file with open() function, which returns a file object. Pass the file object to the reader to read from the CSV file. Read each row and append to the list a[]. Print each row as appended to the list.

```
print("\n Training Dataset \n")
with open('Find-S csv.csv', 'r') as csvFile:
   reader = csv.reader(csvFile)
   for row in reader:
      a.append (row)
      print(row)
```

7. Print the initial value of the hypothesis with '0' for the number of attributes in the dataset.

```
print("\n The initial value of hypothesis: ")
h = ['0'] * num_attrs
print(h)
```

8. Generate the hypothesis 'h' with the first training instance.

```
for j in range(0,num_attrs):
      h[j] = a[0][j];
```

9. Read subsequent training instances and generate hypothesis 'h'.

```
print("\n Find S: Finding Maximally Specific Hypothesis\n")
```

```
        for i in range(0,len(a)):
            if a[i][num_attrs]=='Yes':
                for j in range(0,num_attrs):
                    if a[i][j]!=h[j]:
                        h[j]='?'
                    else :
                        h[j]= a[i][j]
            print(" Training Instance :{0} the hypothesis 'h' is ".format(i),h)
```

10. Print the final maximally specific hypothesis after reading all instances.

```
        print("\n The Final Maximally Specific Hypothesis 'h' is  :\n")
        print(h)
```

4. Complete Program:

```
import csv
attr_values = [['g9','g8'],['Yes','No'],['Excellent','Good'],['Good','Bad'],
        ['Fast','Slow'],['Yes','No']]
num_attrs = len(attr_values)
print (" \n The most general hypothesis : ['?','?','?','?','?','?']\n")
print ("\n The most specific hypothesis : ['0','0','0','0','0','0']\n")
a = []
print("\n Training Dataset \n")
with open('Find-S csv.csv', 'r') as csvFile:
   reader = csv.reader(csvFile)
   for row in reader:
      a.append (row)
```

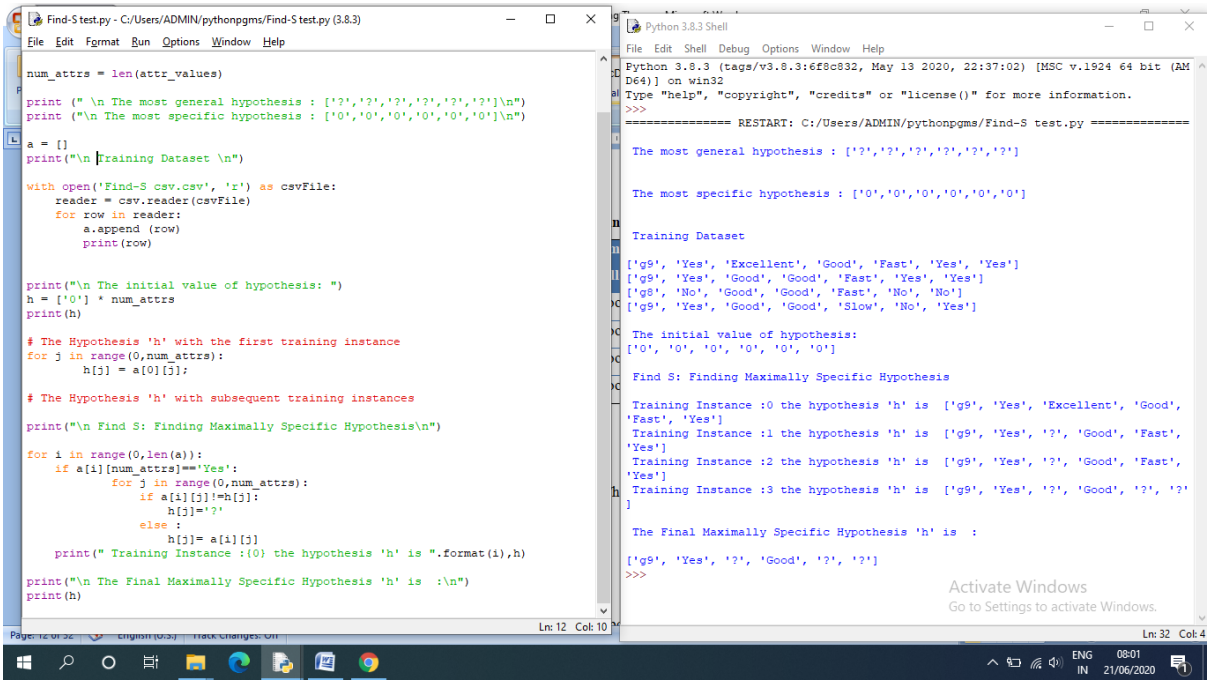
```
        print(row)
print("\n The initial value of hypothesis: ")
h = ['0'] * num_attrs
print(h)
# The Hypothesis 'h' with the first training instance
for j in range(0,num_attrs):
        h[j] = a[0][j];
# The Hypothesis 'h' with subsequent training instances
print("\n Find S: Finding Maximally Specific Hypothesis\n")
for i in range(0,len(a)):
    if a[i][num_attrs]=='Yes':
            for j in range(0,num_attrs):
                if a[i][j]!=h[j]:
                    h[j]='?'
                else :
                    h[j]= a[i][j]
    print(" Training Instance :{0} the hypothesis 'h' is ".format(i),h)
print("\n The Final Maximally Specific Hypothesis 'h' is  :\n")
print(h)
```

Output:

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=============== RESTART: C:/Users/ADMIN/pythonpgms/Find-S test.py ==============

 The most general hypothesis : ['?','?','?','?','?','?']
```

```
The most specific hypothesis : ['0','0','0','0','0','0']


Training Dataset
['g9', 'Yes', 'Excellent', 'Good', 'Fast', 'Yes', 'Yes']
['g9', 'Yes', 'Good', 'Good', 'Fast', 'Yes', 'Yes']
['g8', 'No', 'Good', 'Good', 'Fast', 'No', 'No']
['g9', 'Yes', 'Good', 'Good', 'Slow', 'No', 'Yes']


The initial value of hypothesis:
['0', '0', '0', '0', '0', '0']


Find S: Finding Maximally Specific Hypothesis
Training Instance :0 the hypothesis 'h' is  ['g9', 'Yes', 'Excellent', 'Good', 'Fast', 'Yes']
Training Instance :1 the hypothesis 'h' is  ['g9', 'Yes', '?', 'Good', 'Fast', 'Yes']
Training Instance :2 the hypothesis 'h' is  ['g9', 'Yes', '?', 'Good', 'Fast', 'Yes']
Training Instance :3 the hypothesis 'h' is  ['g9', 'Yes', '?', 'Good', '?', '?']


The Final Maximally Specific Hypothesis 'h' is  :
['g9', 'Yes', '?', 'Good', '?', '?']
>>>
```

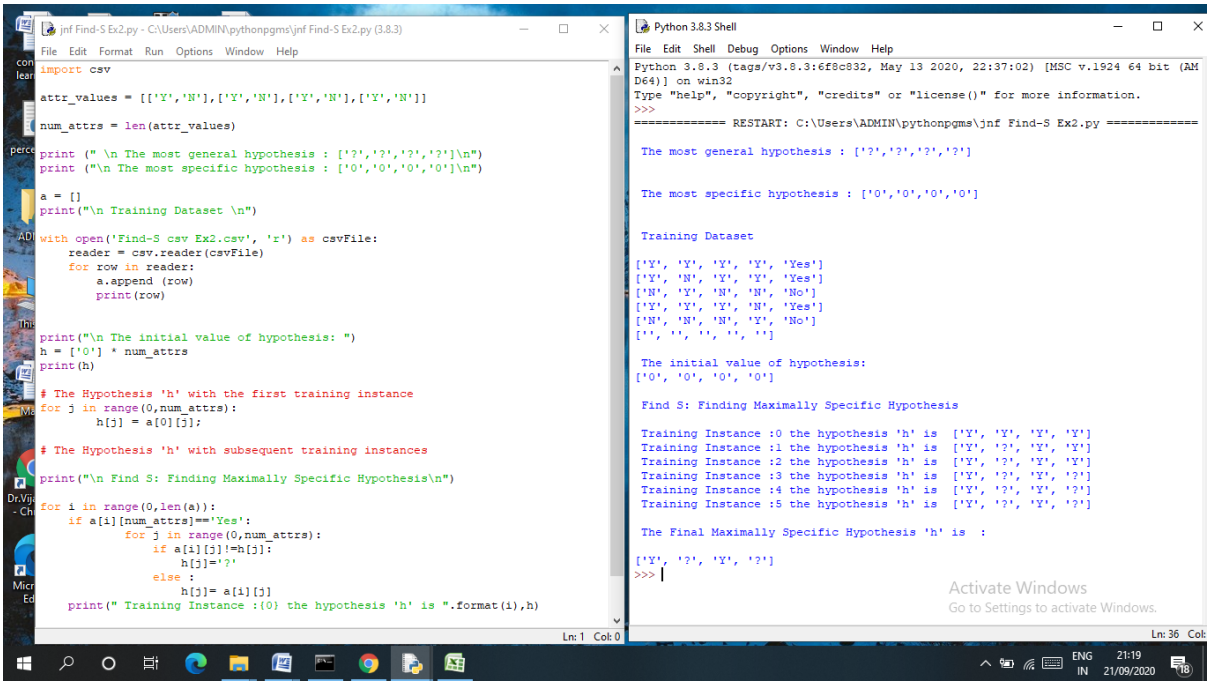
Screenshot of the Output:



Listing 2:

Sample Training Instances

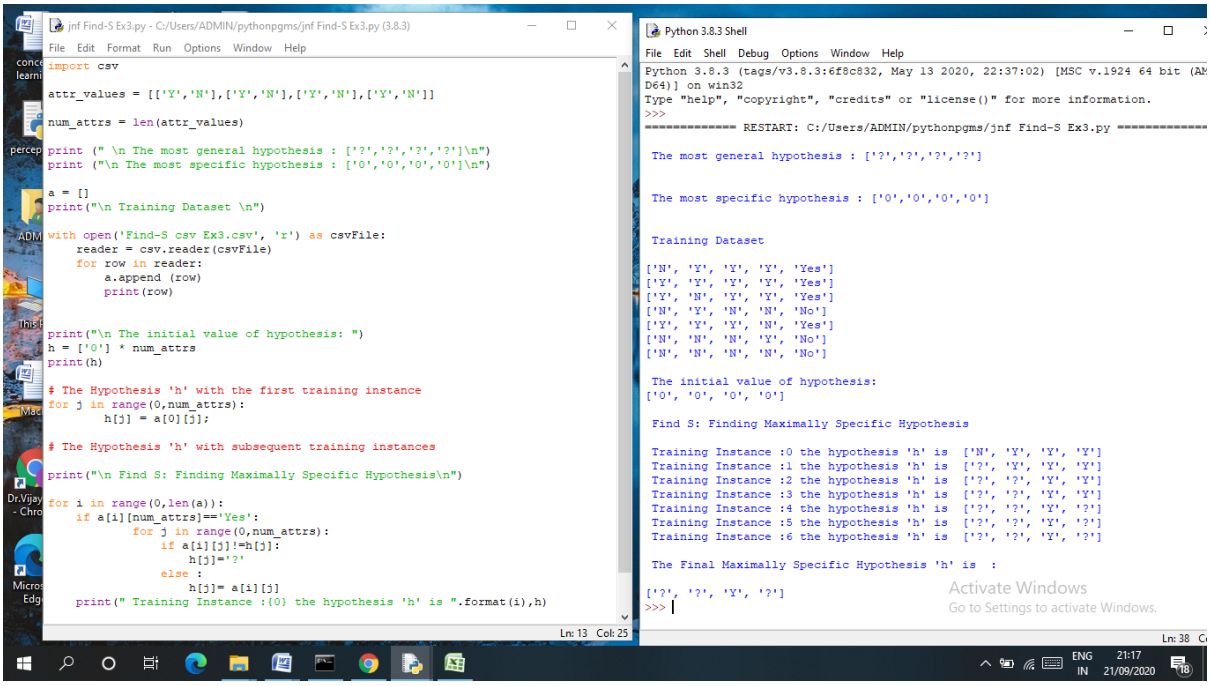
S.No.	Fever	Cough	Throat_Pain	Body_Pain	Covid19
I1.	Y	Y	Y	Y	**Positive**
I2.	Y	N	Y	Y	**Positive**
I3.	N	Y	N	N	**Negative**
I4.	Y	Y	Y	N	**Positive**
I5.	N	N	N	Y	**Negative**



Listing 3:

Table 3.6: Sample Training Instances

S.No.	Fever	Cough	Throat_Pain	Body_Pain	Covid19
I1.	N	Y	Y	Y	**Positive**
I2.	Y	Y	Y	Y	**Positive**
I3.	Y	N	Y	Y	**Positive**
I4.	N	Y	N	N	**Negative**
I5.	Y	Y	Y	N	**Positive**
I6.	N	N	N	Y	**Negative**
I7.	N	N	N	N	**Negative**



LAB EXERCISE –8

k-Nearest Neighbor Algorithm

1. Aim of the Experiment:

Implement and demonstrate k-Nearest Neighbor algorithm. Read the training data from a .CSV file and build the model to classify a test sample. Print both correct and wrong predictions.

2. Reference to Text book for Algorithms:

Refer to Section 4.2 in Chapter 4 Similarity Based Learning to understand the working of the algorithm.

Listing 1:

Sample Dataset Used: Table 4.2 in that chapter.

Table 4.2: Training Dataset T

S.No.	CGPA	Assessment	Project Submitted	Result
1.	9.2	85	8	Pass
2.	8	80	7	Pass
3.	8.5	81	8	Pass
4.	6	45	5	Fail
5.	6.5	50	4	Fail
6.	8.2	72	7	Pass
7.	5.8	38	5	Fail
8.	8.9	91	9	Pass

3. Python Program with Explanation:

1. Import the function train_test_split to split the dataset into training dataset and test dataset.

```
from sklearn.model_selection import train_test_split
```

2. Import KNeighborsClassifier model from sklearn.neighbors.

```
from sklearn.neighbors import KNeighborsClassifier
```

3. Import classification_report and confusion_matrix from sklearn.metrics to measure the quality of predictions.

```
from sklearn.metrics import classification_report, confusion_matrix
```

4. Import StandardScaler to apply scaling transformations.

```
from sklearn.preprocessing import StandardScaler
```

5. Load data from a CSV file into a Pandas DataFrame.

```
dataset = pd.read_csv("knearest csv.csv")
```

6. Print the total number of records and the number of attributes in the dataframe using the function shape().

```
print(dataset.shape)
```

7. Print the first 5 records using the function head().

```
print(dataset.head())
```

8. The function tail(2) will print the last 2 records in the data frame.

```
print(dataset.tail(2))
```

9. Use iloc property to select by position.

Select the columns until (excluding) the last column. Then print it.

```
X = dataset.iloc[:, :-1].values
```

Select the fourth column and then print it.

```
y = dataset.iloc[:, 3].values
print(X)
print(y)
```

10. Split the dataset into training dataset and test dataset by using the function train_test_split().

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

11. Create a new instance of StandardScaler and then fit and transform the scaler to X_train and X_test. Standardize X_train and X_test by computing the mean and standard deviation

by the transform() function on a training set so as to later reapply the same transformation on the testing set.

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

12. Use KNeighborsClassifier model. The number of neighbours 'k' to consider is given through the parameter n_neighbors.

```
classifier = KNeighborsClassifier(n_neighbors=3)
```

13. The model takes as input two arrays: an array X_train, holding the training instances, and an array y_train holding the class labels for the training instances. Then train the classifier using the function fit().

```
classifier.fit(X_train, y_train)
```

14. To make predictions, the predict method of the KNeighborsClassifier class is used.

```
y_pred = classifier.predict(X_test)
```

15. Generate classification report & confusion matrix to measure the quality of predictions.

```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

17. After training, the fitted model can be used to predict a new instance.
The new instance tested is [CGPA:9.1, Assessment:85, Project Submitted:8]

```
print([9.1,85,8.])
predicted = classifier.predict([[9.1,85,8.]])
print(predicted)
```

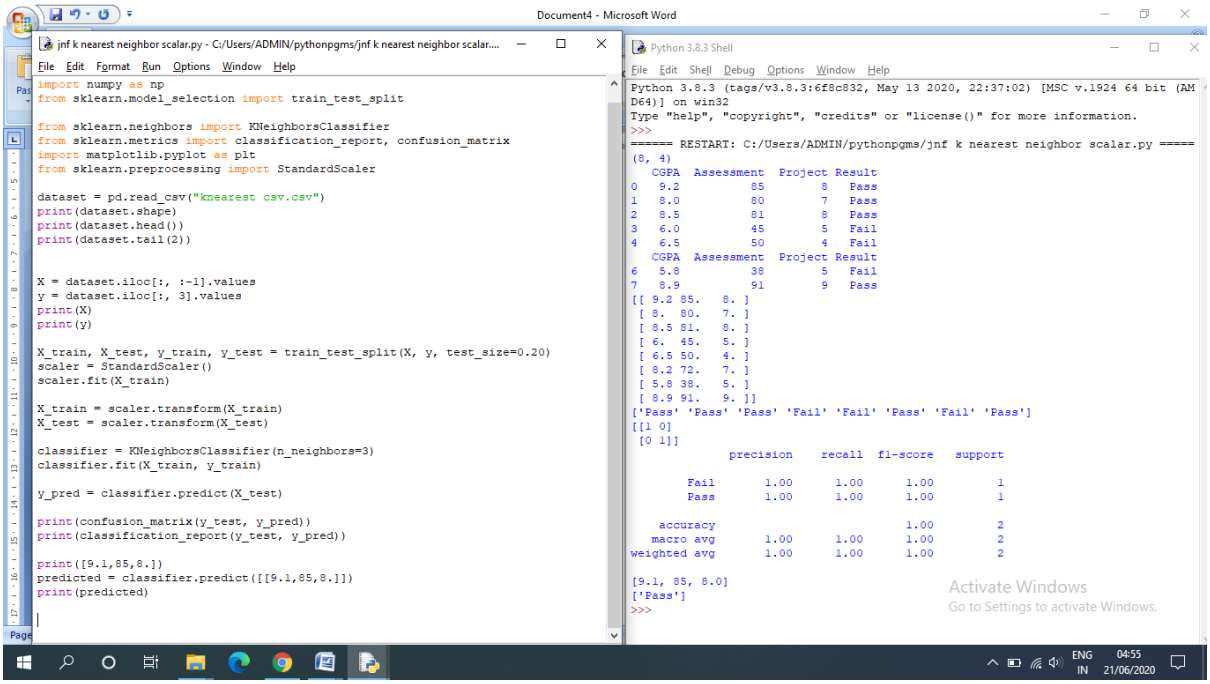
4. Complete Program:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler

dataset = pd.read_csv("knearest csv.csv")
print(dataset.shape)
print(dataset.head())
print(dataset.tail(2))

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 3].values
print(X)
print(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

print([9.1,85,8.])
predicted = classifier.predict([[9.1,85,8.]])
print(predicted)
```

Output:

```
(8, 4)
  CGPA  Assessment  Project Result
0  9.2        85       8   Pass
1  8.0        80       7   Pass
2  8.5        81       8   Pass
3  6.0        45       5   Fail
4  6.5        50       4   Fail
  CGPA  Assessment  Project Result
6  5.8        38       5   Fail
7  8.9        91       9   Pass
[[ 9.2 85.   8. ]
 [ 8.  80.   7. ]
 [ 8.5 81.   8. ]
 [ 6.  45.   5. ]
 [ 6.5 50.   4. ]
 [ 8.2 72.   7. ]
 [ 5.8 38.   5. ]
 [ 8.9 91.   9. ]]
['Pass' 'Pass' 'Pass' 'Fail' 'Fail' 'Pass' 'Fail' 'Pass']
[[1 0]
 [0 1]]
              precision    recall  f1-score   support

      Fail       1.00      1.00      1.00         1
      Pass       1.00      1.00      1.00         1

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2

[9.1, 85, 8.0]
['Pass']
>>>
```

Screenshot of the Output:



Listing 2:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import matplotlib.pyplot as plt

# Load the Iris data set
dataset = pd.read_csv("Iris.csv")
print(dataset.shape)
print(dataset.head())
print(dataset.tail(2))
print(dataset.describe())

#  Split the Iris features into input and output columns
X = dataset.iloc[:, 1:5].values
y = dataset.iloc[:, 5].values
```

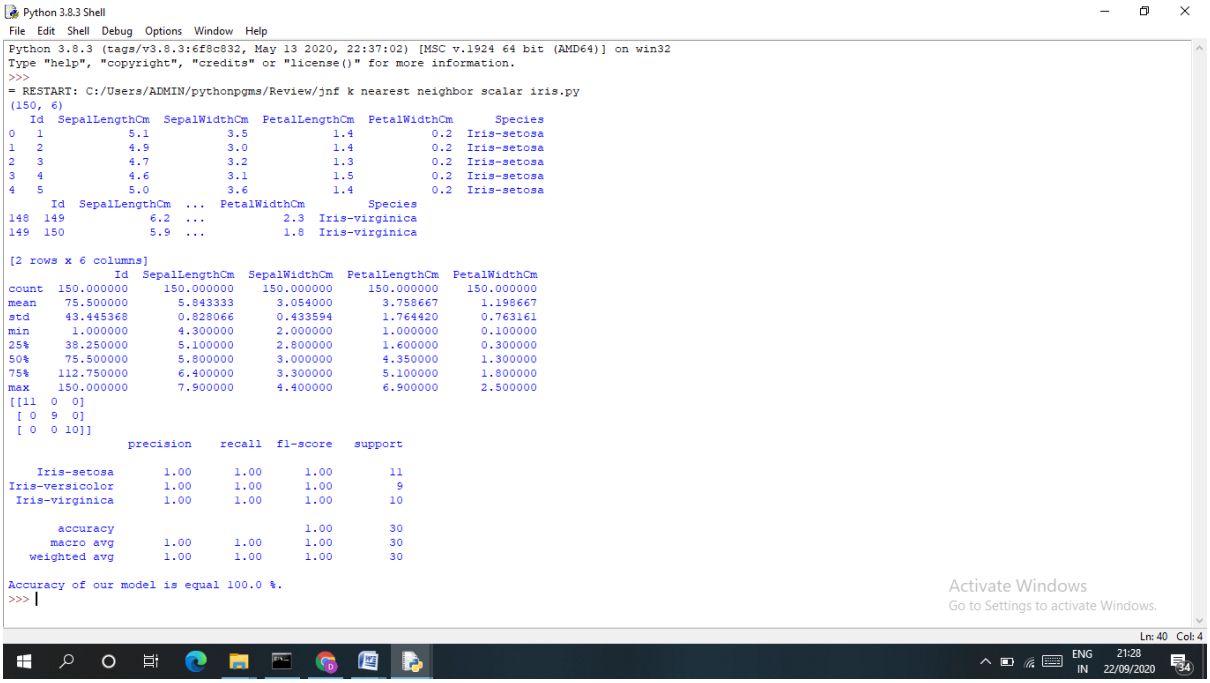
```
# Split the data matrix into train and test dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)

#Train the model using KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

# Evaluate the model and print the confusion matrix and classification report
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Print the accuracy score
accuracy = accuracy_score(y_test, y_pred)*100
print('Accuracy of our model is equal ' + str(round(accuracy, 2)) + ' %.')
```

Output:

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/ADMIN/pythonpgms/Review/jnf k nearest neighbor scalar iris.py
(150, 6)
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
     Id  SepalLengthCm  ...  PetalWidthCm         Species
148  149            6.2  ...           2.3  Iris-virginica
149  150            5.9  ...           1.8  Iris-virginica

[2 rows x 6 columns]
          Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
```

```
count  150.000000    150.000000    150.000000    150.000000    150.000000
mean    75.500000      5.843333      3.054000      3.758667      1.198667
std     43.445368      0.828066      0.433594      1.764420      0.763161
min      1.000000      4.300000      2.000000      1.000000      0.100000
25%     38.250000      5.100000      2.800000      1.600000      0.300000
50%     75.500000      5.800000      3.000000      4.350000      1.300000
75%    112.750000      6.400000      3.300000      5.100000      1.800000
max    150.000000      7.900000      4.400000      6.900000      2.500000
[[11  0  0]
 [ 0  9  0]
 [ 0  0 10]]
              precision    recall  f1-score   support

   Iris-setosa       1.00      1.00      1.00       11
Iris-versicolor       1.00      1.00      1.00        9
 Iris-virginica       1.00      1.00      1.00       10

      accuracy                           1.00       30
     macro avg       1.00      1.00      1.00       30
  weighted avg       1.00      1.00      1.00       30

Accuracy of our model is equal 100.0 %.
>>>
```

Screenshot of the Output:



LAB EXERCISE – 9

LINEAR REGRESSION AND MULTIPLE REGRESSION

Aim of the Experiment

To write Python program for finding linear regression.

Reference to Textbook and Explanation

Chapter 5 and Appendix 2 for details about Linear and multiple Regression.

Consider the dataset given in Chapter 5 of this textbook. It is reproduced here for understanding in the Table below.

First one can write a linear regression for this problem and can verify that the results are same.

Table 5.1: Sample Data

x **(Week)**	**y** **(Sales in Thousands)**
1	1.2
2	1.8
3	2.6
4	3.2
5	3.8

In listing 2, A random dataset is taken, and multiple regression is applied. This experiment will help to understand the concepts of multiple regression.

The command

X,y = make_regression(n_samples = 50,n_features=1,noise=0.1)

Can create a regression dataset with 50 samples and 1 feature. The number of features field can be changed with 2 for multiple regression as

X,y = make_regression(n_samples = 50,n_features=2,noise=0.1)

WARNING – Random dataset is used for Listing 2 and 3. So, the dataset would be generated at every run. As dataset is generated again, the results would vary every time the program is run.

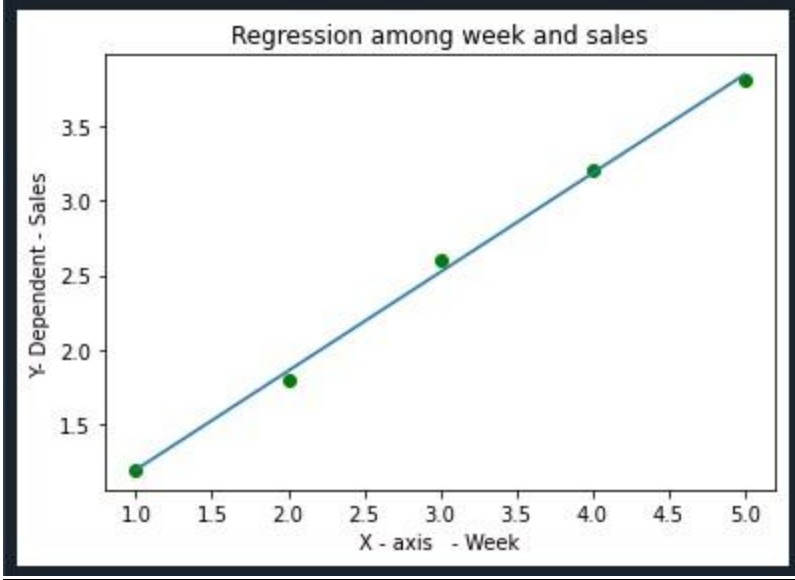
Listing - 1

```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn import linear_model

salesdata = {'week': [1,2,3,4,5],
             'sales': [1.2,1.8,2.6,3.2,3.8]
             }

df = pd.DataFrame(salesdata,columns=['week','sales'])


plt.scatter(df['week'], df['sales'], color='green')
plt.title('Regression among week and sales')
plt.xlabel('X - axis   - Week')
plt.ylabel('Y- Dependent - Sales')


"""

week = df['week'].values.reshape(1,-1)
sales = df['sales'].values.reshape(1,-1)

"""
```

```
X = df[['week']]
y = df['sales']

regr = linear_model.LinearRegression()
regr.fit(X,y)

print('Intercept: \n', regr.intercept_)
print('Coefficients: \n', regr.coef_)

print('\nThe Regression Equation is',regr.coef_,'* X+',regr.intercept_)

# Fit the model for the given data

pred = regr.predict(X)
plt.plot(X,pred)

# Compute Adjusted R squared Error
print("\nAdjusted R Squared for Regression model:",regr.score(X,y))
```

Output

```
Intercept:
 0.5400000000000005
Coefficients:
 [0.66]

The Regression Equation is [0.66] * X+ 0.5400000000000005

Adjusted R Squared for Regression model: 0.9972527472527473
```



Listing 2

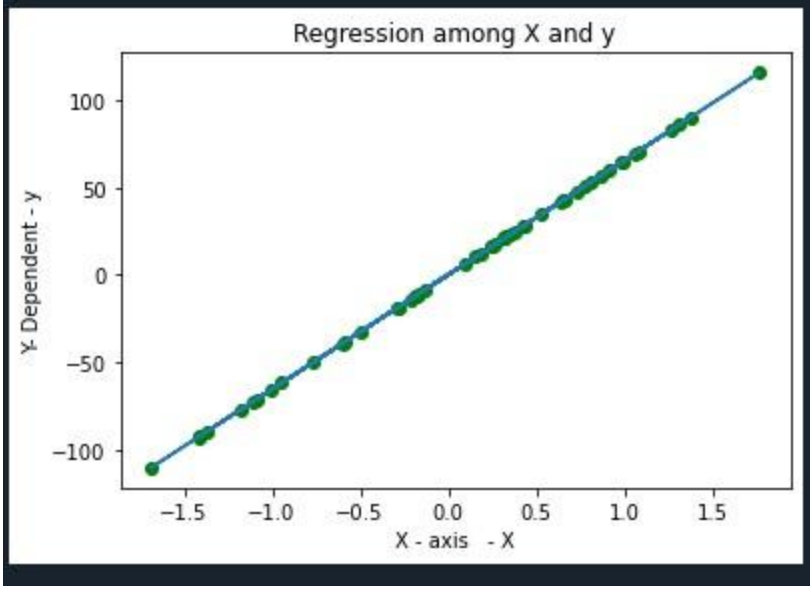
WARNING – Random dataset is used for Listing 2. So, the random dataset would be generated at every run. As dataset is generated again, the results would vary every time the program is run.

```
import matplotlib.pyplot as plt

from sklearn import linear_model

from sklearn.datasets import make_regression


X,y = make_regression(n_samples = 50,n_features=1,noise=0.1)


plt.scatter(X,y,color='green')

plt.title('Regression among X and y')

plt.xlabel('X - axis   - X')

plt.ylabel('Y- Dependent - y')
```

```
regr = linear_model.LinearRegression()
regr.fit(X,y)

print('Intercept: \n', regr.intercept_)
print('Coefficients: \n', regr.coef_)
print('\nThe Regression Equation is',regr.coef_,'* X +',regr.intercept_)

# Fit the model for the given data

pred = regr.predict(X)
plt.plot(X,pred)

# Compute Adjusted R squared Error
print("\nAdjusted R Squared for Regression model:",regr.score(X,y))
```

Output

```
Intercept:
 -0.01950347285833942
Coefficients:
 [65.47285324]

The Regression Equation is [65.47285324] * X + -0.01950347285833942

Adjusted R Squared for Regression model: 0.99999677094372
```



Listing 3

WARNING – Random dataset is used for Listing 3. So, the random dataset would be generated at every run. As dataset is generated again, the results would vary every time the program is run.

Multiple Regression

```
from sklearn import linear_model

from sklearn.datasets import make_regression


print("Multiple regression \n\n")


# Multiple Regression

# Create random dataset with 2 features. Dataset has 50 samples with noise 0.1.


X,y = make_regression(n_samples = 50,n_features=2,noise=0.1)
```

```
regr = linear_model.LinearRegression()

regr.fit(X,y)


print('Intercept: \n', regr.intercept_)

print('Coefficients: \n', regr.coef_)


# Compute Adjusted R squared Error


print("\nAdjusted R Squared for Regression model:",regr.score(X,y))
```

Output

```
Multiple regression


Intercept:
 -0.012331786831634162
Coefficients:
 [53.95803654 36.80928639]

Adjusted R Squared for Regression model: 0.999998052358959
```

LAB EXERCISE – 11

Logistic Regression

Aim of the Experiment

The main aim of this experiment is to explore logistic regression model of scikit-learn. The objectives of this experiment are:

1. Explore random dataset generation for logistic regression.
2. Explore logistic regression model in python for randomly generated dataset

Reference to the Textbook

All the fundamentals are given in Chapter 5 and Appendix 2.

Random dataset for classification model can be as follows:

X, y = make_blobs(n_samples=200, centers=3, n_features=3)

The n_samples and n_features can be changed. This has to be imported using the command,

from sklearn.datasets import make_blobs

Logistic regression model can be created by scikit-learn as

model = LogisticRegression()

The algorithm can be applied to the given data as

model.fit(X_train,y_train)

The predictions of the constructed model can be done as

predicted = model.predict(X_test)

The classification report can be generated as follows:

report_lr = classification_report(y_test,predicted)

This classification report must be imported as

from sklearn.metrics import classification_report

WARNING – Random dataset is used for Listing 1. So, the dataset would be generated at every run. As dataset is generated again, the results would vary every time the program is run.

Listing - 1

```
import pandas as pd
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report


X, y = make_blobs(n_samples=200, centers=3, n_features=3)
df = pd.DataFrame(dict(x=X[:,0], y=X[:,1], label=y))
# Print the sample top five records
print("Top five Records\n\n")
df_top = df.head(5)
print(df_top)
 # Condition the input
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.40,random_state=0)
# Construct the logistic regression model
model = LogisticRegression()
# Fit the model
model.fit(X_train,y_train)
#Prediction for the test sample
predicted = model.predict(X_test)
# Print the classification report
print("\n\nClassification Report")
report_lr = classification_report(y_test,predicted)
print(report_lr)
```

Output

The top five records of 200 samples is displayed as follows:

```
Top five Records


          x         y  label
0  0.624773 -4.846126      1
1  7.688025 -4.351219      0
2  4.140132  6.958442      2
3  5.603908  5.247835      2
4  4.720044  6.363697      2
```

The Classification report generated for this problem is shown as follows:

```
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        26
           1       1.00      1.00      1.00        28
           2       1.00      1.00      1.00        26

    accuracy                           1.00        80
   macro avg       1.00      1.00      1.00        80
weighted avg       1.00      1.00      1.00        80
```

LAB EXERCISE – 12

Decision Tree Classifier – CART

1. Aim of the Experiment:

Implement and demonstrate the working of the decision tree based CART algorithm using a sample data set. Build the decision tree and use this model to classify a test sample.

2. Reference to Text book for Algorithms:

Refer to Section 6.2.3 in Chapter 6 Decision Tree Learning to understand the working of the algorithm.

Listing 1:

Sample Dataset Used: **Table 6.3**

S.No.	CGPA	Interactiveness	Practical Knowledge	Communication Skills	Job Offer
1.	≥9	Yes	Very good	Good	Yes
2.	≥8	No	Good	Moderate	Yes
3.	≥9	No	Average	Poor	No
4.	<8	No	Average	Good	No
5.	≥8	Yes	Good	Moderate	Yes
6.	≥9	Yes	Good	Moderate	Yes
7.	<8	Yes	Good	Poor	No
8.	≥9	No	Very good	Good	Yes
9.	≥8	Yes	Good	Good	Yes
10.	≥8	Yes	Average	Good	Yes

3. Python Program with Explanation:

1. Import the library ‘pandas’ to create a Data frame which is a two-dimensional data structure.

```
import pandas
```

2. Import DecisionTreeClassifier from sklearn.tree.

```
from sklearn.tree import DecisionTreeClassifier
```

3. Import LabelEncoder to normalize labels.

```
from sklearn.preprocessing import LabelEncoder
```

4. Import train_test_split function.

```
from sklearn.model_selection import train_test_split
```

5. Create a list ‘data’ with the sample dataset.

```
data = {'CGPA':['g9','g8','g9','l8','g8','g9','l8','g9','g8','g8'],
        'Inter':['Y','N','N','N','Y','Y','Y','N','Y','Y'],
        'PK':['+++','+','==','==','+','+','+','+++','+','=='],
        'CS':['G','M','P','G','M','M','P','G','G','G'],
        'Job':['Y','Y','N','N','Y','Y','N','Y','Y','Y']}
```

6. Create pandas dataframe “table” using the structure DataFrame with the given dataset ‘data'.

```
table=pandas.DataFrame(data, columns=["CGPA","Inter","PK","CS","Job"])
```

7. Use a value ["CGPA"]=="g9" in the table to select matching row and count the number of columns.

```
table.where(table["CGPA"]=="g9").count()
```

8. Use LabelEncoder() to encode target labels with value between 0 and no_of_classes-1.

```
encoder=LabelEncoder()
```

9. Then transform non-numerical labels to numerical labels.

```
for i in table:
   table[i]=encoder.fit_transform(table[i])
```

10. Use iloc property to select by position.

Select the columns until (excluding) the fourth column.

```
X=table.iloc[:,0:4].values
```

Select the fourth column

```
y=table.iloc[:,4].values
```

11. Split the dataset into training dataset and test dataset by using the function train_test_split(). This function has several parameters, but we pass 3 parameters, data, test_size and random_state.

X, y is the dataset we are selecting to use.

test_size. to specify the size of the testing dataset. It will be set to 0.25 if the training size is set to default.

random_state to perform a random split.

X_train is the features of the training subset

y_train is the class labels of the target feature of the training subset

X_test holds the features of the testing subset

y_test holds the class labels of the target feature of the testing subset

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=2)
```

12. Use DecisionTreeClassifier model. It allows some attributes like criterion, splitter, max_features, max_depth, max_leaf_nodes etc., we will use the attribute criterion which takes a value ‘gini’ to implement a classifier using CART

```
model = DecisionTreeClassifier(criterion='gini')
```

DecisionTreeClassifier takes as input two arrays: an array X_train, holding the training instances, and an array y_train holding the class labels for the training instances.

13. Then train the classifier using the function fit().

```
model.fit(X_train,y_train)
```

14. After training, the fitted model can be used to predict a new instance.

The non-numerical equivalent of the new instance [1,0,0,1] given is [‘g9’, ‘Y’, ‘***’, ‘M’]

```
print([1,0,0,1])
if model.predict([[1,0,0,1]])==1:
   print("Got JOB")
else:
   print("Didnt get JOB")

# The non-numerical equivalent of the new instance [2,0,2,0] given is [‘l8’, ‘Y’, ‘==’, ‘G’]
print([2,0,2,0])
if model.predict([[2,0,2,0]])==1:
   print("Got JOB")
else:
   print("Didnt get JOB")
```

Complete Program:

```
import pandas
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
data = {'CGPA':['g9','g8','g9','l8','g8','g9','l8','g9','g8','g8'],
    'Inter':['Y','N','N','N','Y','Y','Y','N','Y','Y'],
    'PK':['+++','+','==','==','+','+','+','+++','+','=='],
    'CS':['G','M','P','G','M','M','P','G','G','G'],
    'Job':['Y','Y','N','N','Y','Y','N','Y','Y','Y']}

table=pandas.DataFrame(data,columns=["CGPA","Inter","PK","CS","Job"])
table.where(table["CGPA"]=="g9").count()
encoder=LabelEncoder()
```

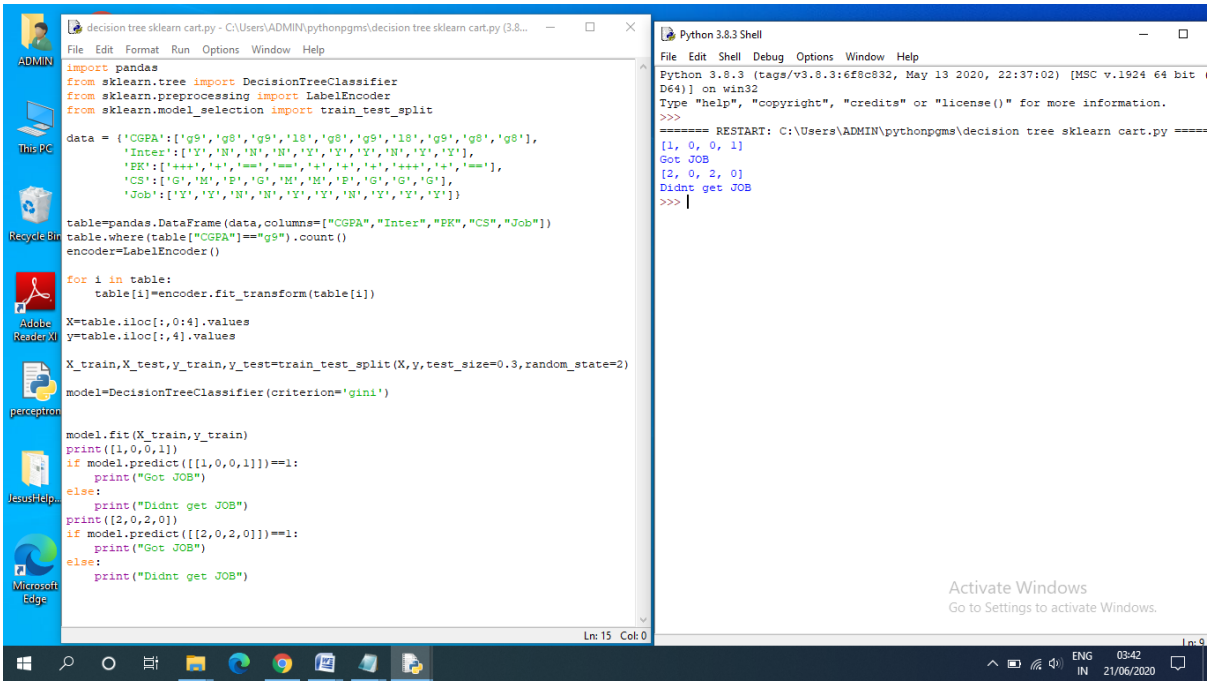
```
for i in table:
    table[i]=encoder.fit_transform(table[i])
X=table.iloc[:,0:4].values
y=table.iloc[:,4].values
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=2)
model=DecisionTreeClassifier(criterion='gini')
model.fit(X_train,y_train)
print([1,0,0,1])
if model.predict([[1,0,0,1]])==1:
    print("Got JOB")
else:
    print("Didnt get JOB")
print([2,0,2,0])
if model.predict([[2,0,2,0]])==1:
    print("Got JOB")
else:
    print("Didnt get JOB")
```

Output:

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======= RESTART: C:\Users\ADMIN\pythonpgms\decision tree sklearn cart.py =======
[1, 0, 0, 1]
Got JOB
[2, 0, 2, 0]
Didnt get JOB
```

>>>

Screenshot of the Output:



Listing 2:

Program Code:

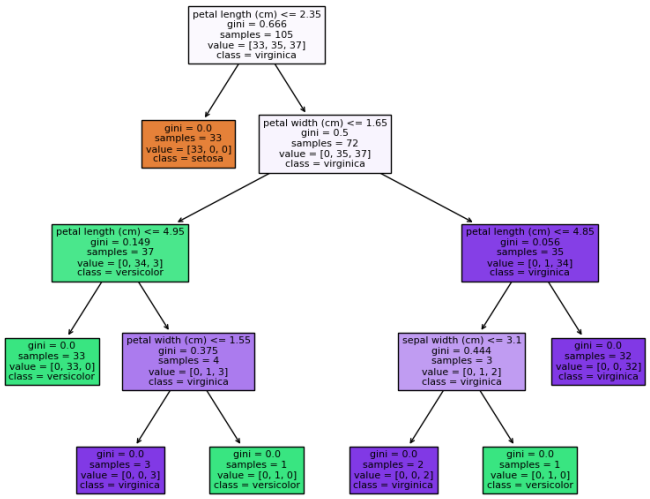
```
from matplotlib import pyplot as plt

from sklearn import datasets

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import train_test_split

from sklearn import metrics

from sklearn.metrics import classification_report, confusion_matrix



# Load the Iris dataset

iris = datasets.load_iris()
```

```
X = iris.data

y = iris.target


# Split the data matrix into train and test dataset
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.30,random_state=2)


# Train the model using DecisionTreeClassifier CART
clf = DecisionTreeClassifier(criterion='gini',random_state=1234)

model = clf.fit(X_train, y_train)

y_pred = model.predict(X_test)


# Evaluating Classification Model Accuracy

print("Accuracy:",metrics.accuracy_score(y_test, y_pred)) # classification rate of 100%,good
accuracy.


#Print Confusion Matrix

print(confusion_matrix(y_test, y_pred))


#Print Classification Report and plot the tree graph

print(classification_report(y_test, y_pred))

fig = plt.figure(figsize=(10,8))

_ = tree.plot_tree(clf,

            feature_names=iris.feature_names,

            class_names=iris.target_names,

            filled=True)

plt.show()
```

```
*Python 3.8.3 Shell*
File  Edit  Shell  Debug  Options  Window  Help
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====== RESTART: C:\Users\ADMIN\pythonpgms\Review\Decision tree Viz CART.py =====
Accuracy: 0.9555555555555556
[[17  0  0]
 [ 0 14  1]
 [ 0  1 12]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        17
           1       0.93      0.93      0.93        15
           2       0.92      0.92      0.92        13

    accuracy                           0.96        45
   macro avg       0.95      0.95      0.95        45
weighted avg       0.96      0.96      0.96        45
```


Ln: 5 Col: 0

Figure 1



x=0.594389 y=0.534558

LAB EXERCISE – 13

Decision Tree Classifier – ID3

1. Aim of the Experiment:

Implement and demonstrate the working of the decision tree based ID3 algorithm using a sample data set. Build the decision tree and use this model to classify a test sample.

2. Reference to Text book for Algorithms:

Refer to Section 6.2.1 in Chapter 6 Decision Tree Learning to understand the working of the algorithm.

Listing 1:

Sample Dataset Used: **Table 6.3**

S.No.	CGPA	Interactiveness	Practical Knowledge	Communication Skills	Job Offer
1.	≥9	Yes	Very good	Good	Yes
2.	≥8	No	Good	Moderate	Yes
3.	≥9	No	Average	Poor	No
4.	<8	No	Average	Good	No
5.	≥8	Yes	Good	Moderate	Yes
6.	≥9	Yes	Good	Moderate	Yes
7.	<8	Yes	Good	Poor	No
8.	≥9	No	Very good	Good	Yes
9.	≥8	Yes	Good	Good	Yes
10.	≥8	Yes	Average	Good	Yes

3. Python Program with Explanation:

1. Import the library ‘pandas’ to create a Data frame which is a two-dimensional data Structure.

```
import pandas
```

2. Import DecisionTreeClassifier from sklearn.tree.

```
from sklearn.tree import DecisionTreeClassifier
```

3. Import LabelEncoder to normalize labels.

```
from sklearn.preprocessing import LabelEncoder
```

4. Import train_test_split function.

```
from sklearn.model_selection import train_test_split
```

5. Import metrics module to implement functions to measure classification performance.

```
from sklearn import metrics
```

6. Import classification_report and confusion_matrix from sklearn.metrics to measure the quality of predictions.

```
from sklearn.metrics import classification_report, confusion_matrix
```

7. Create a list ‘data’ with the sample dataset.

```
data = {'CGPA':['g9','g8','g9','l8','g8','g9','l8','g9','g8','g8'],
        'Inter':['Y','N','N','N','Y','Y','Y','N','Y','Y'],
        'PK':['+++','+','==','==','+','+','+','+++','+','=='],
        'CS':['G','M','P','G','M','M','P','G','G','G'],
        'Job':['Y','Y','N','N','Y','Y','N','Y','Y','Y']}
```

8. Create pandas dataframe “table” using the structure DataFrame with the given dataset ‘data'.

```
table=pandas.DataFrame(data, columns=["CGPA","Inter","PK","CS","Job"])
```

9. Use a value ["CGPA"]=="g9" in the table to select matching row and count the number of columns.

```
table.where(table["CGPA"]=="g9").count()
```

10. Use LabelEncoder() to encode target labels with value between 0 and no_of_classes-1.

```
encoder=LabelEncoder()
```

11. Then transform non-numerical labels to numerical labels.

```
for i in table:
   table[i]=encoder.fit_transform(table[i])
```

12. Use iloc property to select by position.

```
Select the columns until (excluding) the fifth column.
X=table.iloc[:,0:4].values

Select the fifth column
y=table.iloc[:,4].values
```

13. Split the dataset into training dataset and test dataset by using the function train_test_split(). This function has several parameters, but we pass 3 parameters, data, test_size and random_state.
X, y is the dataset we are selecting to use.
test_size to specify the size of the testing dataset. It will be set to 0.25 if the training size is set to default.
random_state to perform a random split.
X_train is the features of the training subset
y_train is the class labels of the target feature of the training subset
X_test holds the features of the testing subset
y_test holds the class labels of the target feature of the testing subset

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=2)
```

14. Use DecisionTreeClassifier model. It allows some attributes like criterion, splitter, max_features, max_depth, max_leaf_nodes etc., we will use the attribute criterion which takes a value ‘entropy’ to implement a classifier using ID3. The attribute value for max_depth is given as 3 to pre prune the tree.

```
model=DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

13. DecisionTreeClassifier model takes as input two arrays: an array X_train, holding the training instances, and an array y_train holding the class labels for the training instances. Then train the classifier using the function fit().

```
model.fit(X_train,y_train)
```

14. To make predictions, the predict method of the DecisionTreeClassifier class is used.

```
y_pred = model.predict(X_test)
```

15. Use sklearn.metrics.accuracy_score() to compute the accuracy by comparing actual test set values and predicted values.

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

16. Generate classification report & confusion matrix to measure the quality of predictions.

```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

17. After training, the fitted model can be used to predict a new instance.

```
# The non-numerical equivalent of the new instance [1,0,0,1] given is ['g9', 'Y', '***', 'M']

print([1,0,0,1])
if model.predict([[1,0,0,1]])==1:
   print("Got JOB")
else:
   print("Didnt get JOB")

# The non-numerical equivalent of the new instance [2,0,2,0] given is ['l8', 'Y', '==', 'G']
print([2,0,2,0])
if model.predict([[2,0,2,0]])==1:
   print("Got JOB")
```

```
        else:
            print("Didnt get JOB")
```

Complete Program:

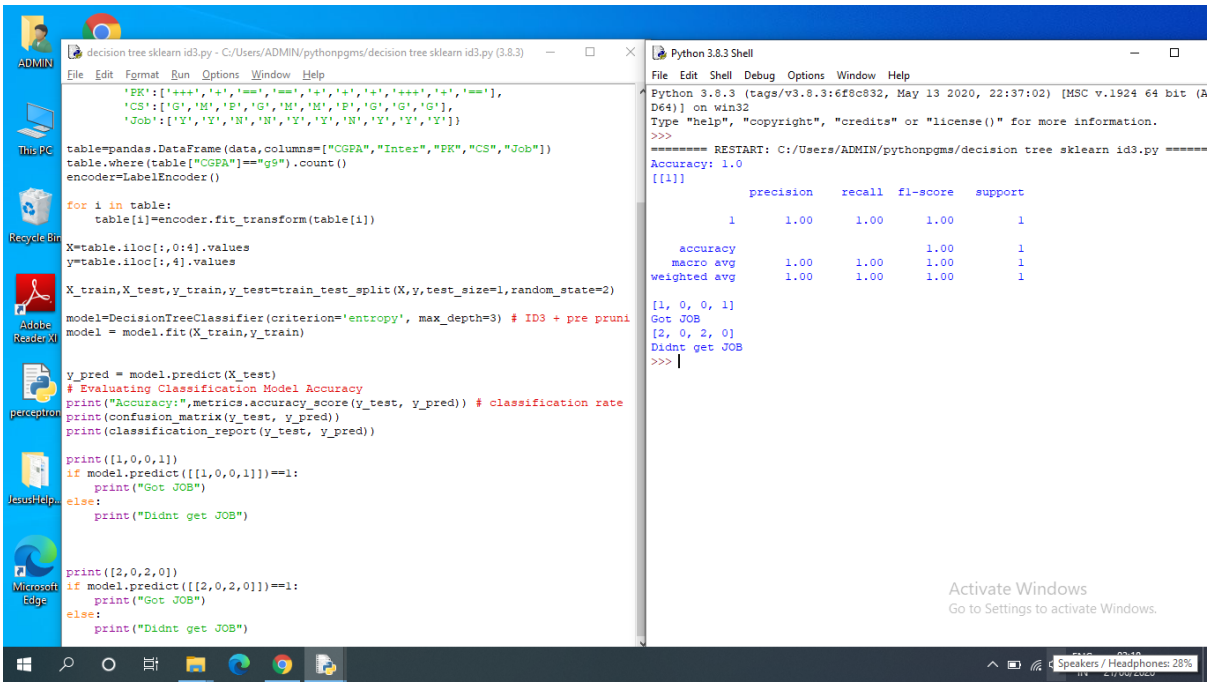
```
import pandas
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix

data = {'CGPA':['g9','g8','g9','l8','g8','g9','l8','g9','g8','g8'],
      'Inter':['Y','N','N','N','Y','Y','Y','N','Y','Y'],
      'PK':['+++','+','==','==','+','+','+','+++','+','=='],
      'CS':['G','M','P','G','M','M','P','G','G','G'],
      'Job':['Y','Y','N','N','Y','Y','N','Y','Y','Y']}

table=pandas.DataFrame(data,columns=["CGPA","Inter","PK","CS","Job"])
table.where(table["CGPA"]=="g9").count()
encoder=LabelEncoder()

for i in table:
   table[i]=encoder.fit_transform(table[i])

X=table.iloc[:,0:4].values
y=table.iloc[:,4].values

 X_train,X_test,y_train,y_test=train_test_split(X,y,test_size= 0.20,random_state=2)
model=DecisionTreeClassifier(criterion='entropy', max_depth=3)
model = model.fit(X_train,y_train)

y_pred = model.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

print([1,0,0,1])
if model.predict([[1,0,0,1]])==1:
   print("Got JOB")
else:
   print("Didn’t get JOB")
 print([2,0,2,0])
if model.predict([[2,0,2,0]])==1:
   print("Got JOB")
else:
   print("Didn’t get JOB")
```

Output:

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======== RESTART: C:\Users\ADMIN\pythonpgms\decision tree sklearn id3.py =======
Accuracy: 1.0
[[1]]
              precision    recall  f1-score   support

           1       1.00      1.00      1.00         1

    accuracy                           1.00         1
   macro avg       1.00      1.00      1.00         1
weighted avg       1.00      1.00      1.00         1

[1, 0, 0, 1]
Got JOB
[2, 0, 2, 0]
Didn’t get JOB
>>>
```

Screen Shot of the Output:

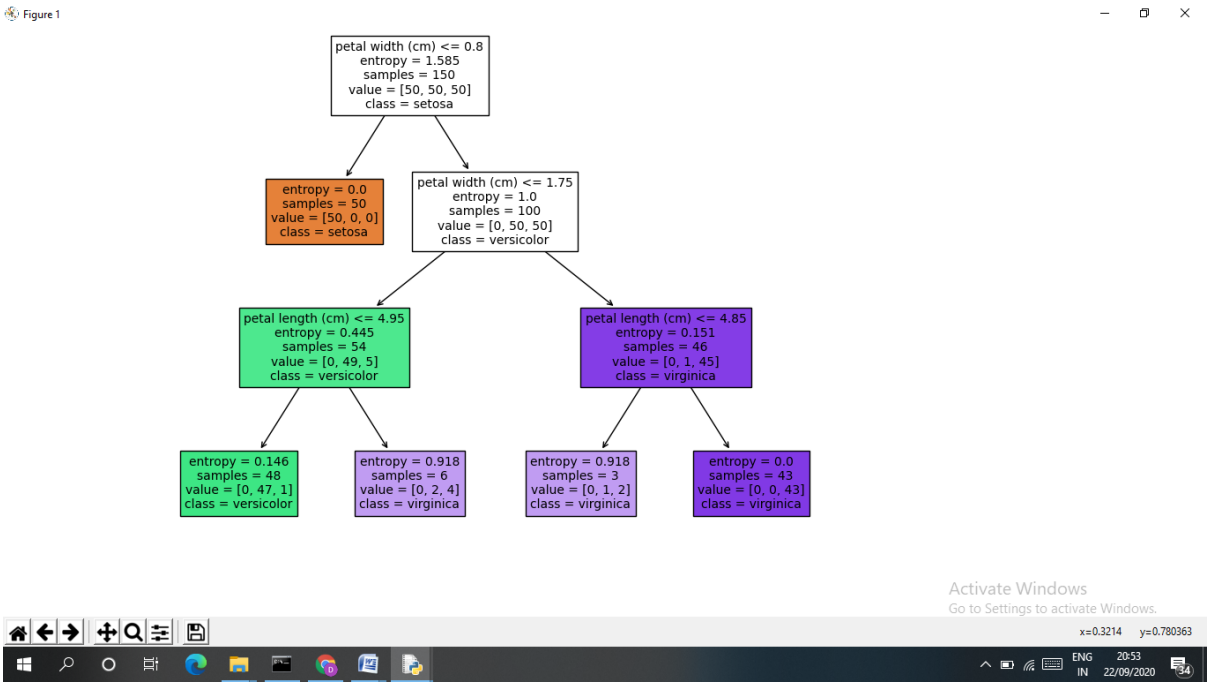


Listing 2:

Program Code:

```
from matplotlib import pyplot as plt
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import graphviz

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Train the model using DecisionTreeClassifier ID3
clf = DecisionTreeClassifier(criterion='entropy', max_depth=3)
model = clf.fit(X, y)

#Visualize the model using tree graph
```

```
fig = plt.figure(figsize=(10,8))
_ = tree.plot_tree(clf,
                   feature_names=iris.feature_names,
                   class_names=iris.target_names,
                   filled=True)
plt.show()
#fig.savefig("decistion_tree.png")
```



LAB EXERCISE – 14

Naive Bayes Classifier

1. Aim of the Experiment:

Implement and demonstrate the working of Naive Bayesian classifier using a sample data set. Build the model to classify a test sample.

2. Reference to Text book for Algorithms:

Refer to Section 8.3.1 in Chapter 8 Bayesian Learning to understand the working of the algorithm.

Listing 1:

Sample Dataset Used: **Table 8.1**

Table 8.1 Training Dataset

S.No.	CGPA	Interactiveness	Practical Knowledge	Communication Skills	Job Offer
1.	≥9	Yes	Very good	Good	Yes
2.	≥8	No	Good	Moderate	Yes
3.	≥9	No	Average	Poor	No
4.	<8	No	Average	Good	No
5.	≥8	Yes	Good	Moderate	Yes
6.	≥9	Yes	Good	Moderate	Yes
7.	<8	Yes	Good	Poor	No
8.	≥9	No	Very good	Good	Yes
9.	≥8	Yes	Good	Good	Yes
10.	≥8	Yes	Average	Good	Yes

3. Python Program with Explanation:

1. Import LabelEncoder to normalize labels.

```
from sklearn.preprocessing import LabelEncoder
```

2. Import train_test_split function.

```
from sklearn.model_selection import train_test_split
```

3. Import Gaussian Classifier from sklearn.naive_bayes.

```
from sklearn.naive_bayes import GaussianNB
```

4. Import preprocessing package to use transformer classes.

```
from sklearn import preprocessing
```

5. Import classification_report and confusion_matrix from sklearn.metrics to measure the quality of predictions.

```
from sklearn.metrics import classification_report, confusion_matrix
```

6. Create lists, CGPA, Inter, PK, CS and Job.

```
CGPA = ['g9','g8','g9','l8','g8','g9','l8','g9','g8','g8']
Inter = ['Y','N','N','N','Y','Y','Y','N','Y','Y']
PK = ['+++','+','==','==','+','+','+','+++','+','==']
CS = ['G','M','P','G','M','M','P','G','G','G']
Job = ['Y','Y','N','N','Y','Y','N','Y','Y','Y']
```

7. Create labelEncoder le to encode labels with value between 0 and no_of_classes-1.

```
le = preprocessing.LabelEncoder()
```

8. Convert non-numeric labels of all features into numbers. Then print and see the encoded labels.

```
CGPA_encoded = le.fit_transform(CGPA)
print("CGPA:", CGPA_encoded)
Inter_encoded = le.fit_transform(Inter)
PK_encoded = le.fit_transform(PK)
CS_encoded = le.fit_transform(CS)
label = le.fit_transform(Job)
print("Inter:",Inter_encoded)
print ("PK:",PK_encoded)
```

```
print ("CS:",CS_encoded)
print("Job:",label)
```

9. Create a list/dynamic array called ‘features’.

```
features = []
```

10. Append all the encoded features to the list created.

```
for i in range(len(CGPA_encoded)):
  features.append([CGPA_encoded[i], Inter_encoded[i], PK_encoded[i],
CS_encoded[i]])
```

11. Split the dataset into training dataset and test dataset by using the function train_test_split().

```
X_train,X_test,y_train,y_test=train_test_split(features,label,test_size=0.30,random_
state=2)
```

12. Create a Gaussian Classifier.

```
model = GaussianNB()
```

13. Train the model using the training sets.

```
model.fit(features, label)
```

14. After training, use the fitted model to predict a new instance.

```
y_pred = model.predict(X_test)
```

15. Generate classification report & confusion matrix to measure the quality of predictions.

```
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

16. Predict Output.

```
# The non-numerical equivalent of the new instance [2, 0, 2, 0] given is [2:‘l8’, 0:‘N’, 2:‘==’, 0:‘G’]
print([2,0,2,0])
if model.predict([[2,0,2,0]])==1:
   print("Predicted Value:Got JOB",predicted )
else:
   print("Didnt get JOB")

   # The non-numerical equivalent of the new instance [0, 1, 0, 1] given is [0:‘g8’, 1:‘Y’, 0:‘+’, 1:‘M’]
print([0,1,0,1])
if model.predict([[0,1,0,1]])==1:
   print("Predicted Value:Got JOB",predicted )
else:
   print("Didnt get JOB")
```

Complete Program:

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import preprocessing
from sklearn.metrics import classification_report, confusion_matrix


CGPA = ['g9','g8','g9','l8','g8','g9','l8','g9','g8','g8']
Inter = ['Y','N','N','N','Y','Y','Y','N','Y','Y']
PK = ['+++','+','==','==','+','+','+','+++','+','==']
CS = ['G','M','P','G','M','M','P','G','G','G']
Job = ['Y','Y','N','N','Y','Y','N','Y','Y','Y']

#creating labelEncoder
le = preprocessing.LabelEncoder()
# Converting string labels into numbers.
```

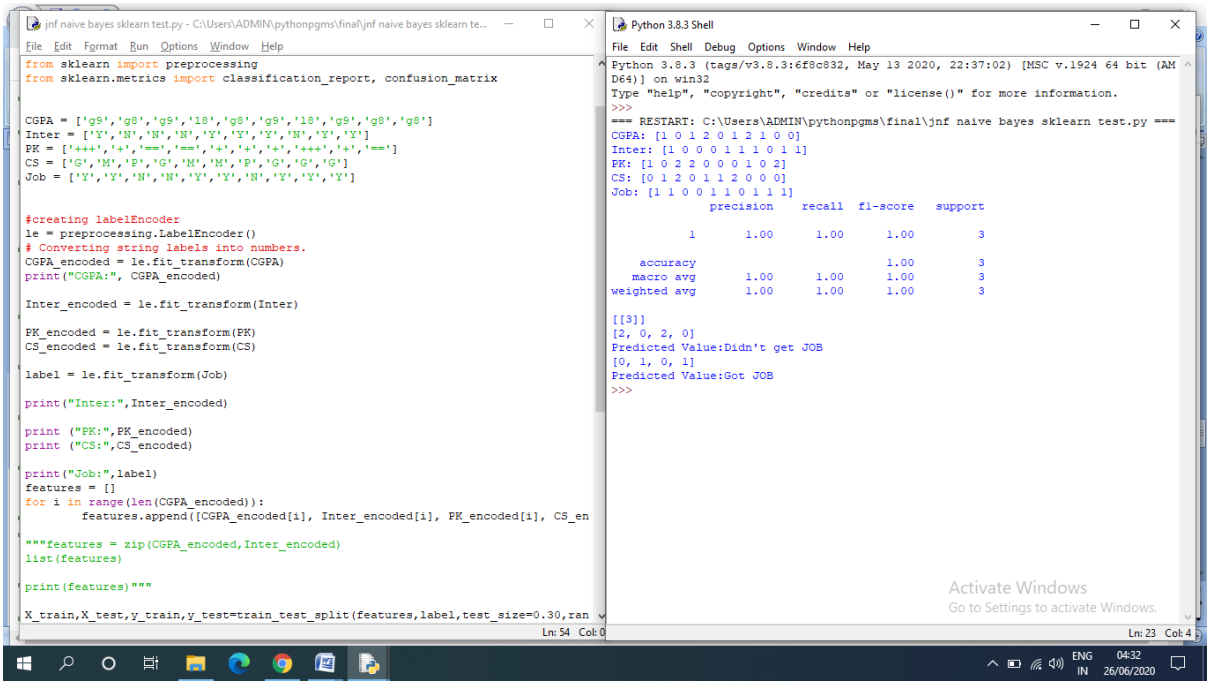
```
CGPA_encoded = le.fit_transform(CGPA)
print("CGPA:", CGPA_encoded)

Inter_encoded = le.fit_transform(Inter)
PK_encoded = le.fit_transform(PK)
CS_encoded = le.fit_transform(CS)
label = le.fit_transform(Job)
print("Inter:",Inter_encoded)
print ("PK:",PK_encoded)
print ("CS:",CS_encoded)
print("Job:",label)

features = []
for i in range(len(CGPA_encoded)):
      features.append([CGPA_encoded[i], Inter_encoded[i], PK_encoded[i], CS_encoded[i]])

X_train,X_test,y_train,y_test=train_test_split(features,label,test_size=0.30,random_state=2)
#Create a Gaussian Classifier
model = GaussianNB()

# Train the model using the training sets
model.fit(features,label)

#Predict Output
y_pred = model.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

print([2,0,2,0])
if model.predict([[2,0,2,0]])==1:
   print("Predicted Value:Got JOB")
else:
   print("Predicted Value:Didn't get JOB")
```

```
print([0,1,0,1])
if model.predict([[0,1,0,1]])==1:
   print("Predicted Value:Got JOB")
else:
   print("Predicted Value:Didn't get JOB")
```

Output:

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:\Users\ADMIN\pythonpgms\final\jnf naive bayes sklearn test.py ===
CGPA: [1 0 1 2 0 1 2 1 0 0]
Inter: [1 0 0 0 1 1 1 0 1 1]
PK: [1 0 2 2 0 0 0 1 0 2]
CS: [0 1 2 0 1 1 2 0 0 0]
Job: [1 1 0 0 1 1 0 1 1 1]
              precision    recall  f1-score   support

           1       1.00      1.00      1.00         3

    accuracy                           1.00         3
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3

[[3]]
[2, 0, 2, 0]
Predicted Value:Didn't get JOB
[0, 1, 0, 1]
Predicted Value:Got JOB
>>>
```

Screenshot of the Output:



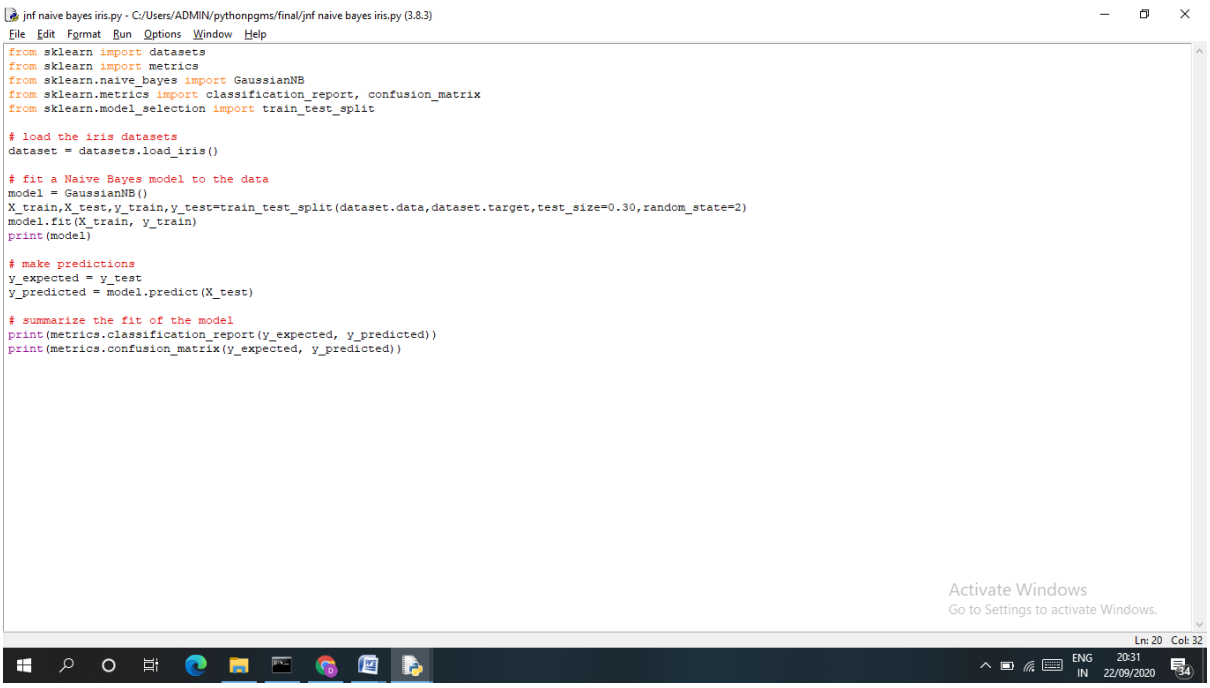
Listing 2:

Program Code:

```
from sklearn import datasets

from sklearn import metrics

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import classification_report, confusion_matrix

from sklearn.model_selection import train_test_split


# Load the Iris dataset

dataset = datasets.load_iris()


# Fit a Naive Bayes model to the data

model = GaussianNB()

X_train,X_test,y_train,y_test=train_test_split(dataset.data,dataset.target,test_size=0.30,random_state=2)

model.fit(X_train, y_train)
```

```
print(model)


# Make predictions
y_expected = y_test
y_predicted = model.predict(X_test)


# Evaluate the model and print the classification report, Confusion Matrix
print(metrics.classification_report(y_expected, y_predicted))
print(metrics.confusion_matrix(y_expected, y_predicted))
```

Screen Shot of the Program:



Output:

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======= RESTART: C:\Users\ADMIN\pythonpgms\Review\jnf naive bayes iris.py ======
```

GaussianNB()

	precision	recall	f1-score	support
0	1.00	1.00	1.00	17
1	1.00	0.93	0.97	15
2	0.93	1.00	0.96	13
accuracy			0.98	45
macro avg	0.98	0.98	0.98	45
weighted avg	0.98	0.98	0.98	45

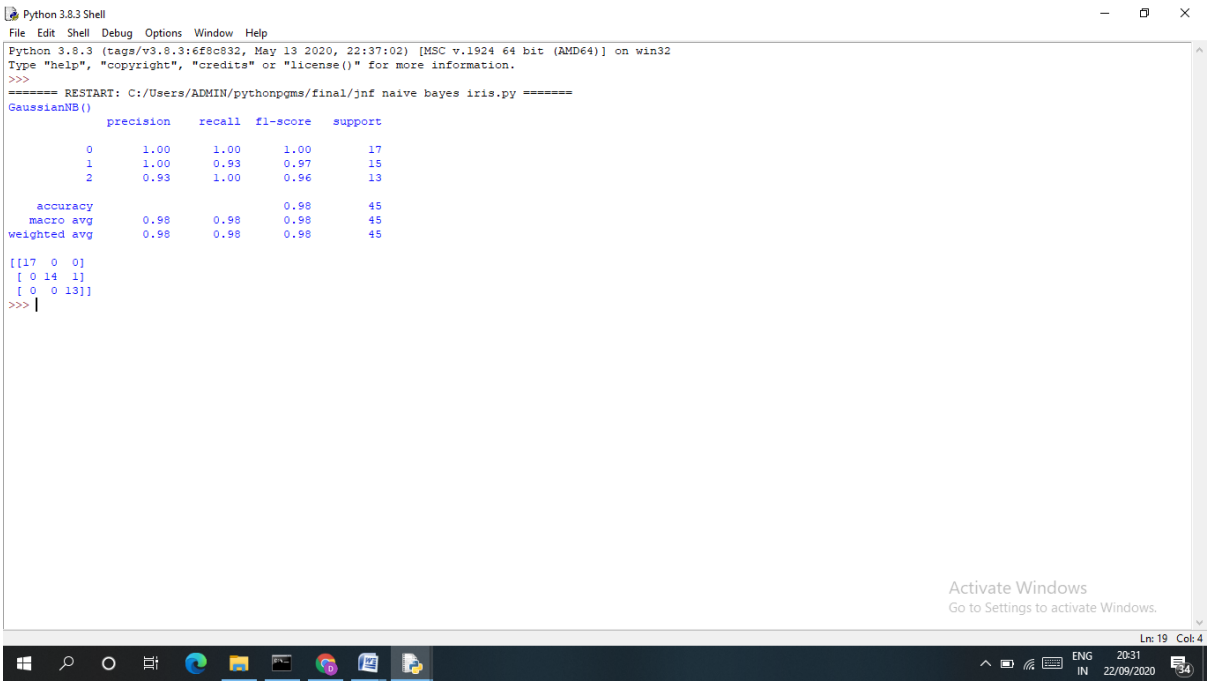
[[17 0 0]

[0 14 1]

[0 0 13]]

>>>

Screen Shot of the Output:



LAB EXERCISE – 15

Hidden Markov Model

1. Aim of the Experiment:

Implement and demonstrate Hidden Markov Model (HMM) to decode the hidden states given a sequence of observation states using Viterbi algorithm.

2. Reference to Text book for Algorithms:

Refer to Section 9.3.2 and 9.4 in Chapter 9 Probabilistic Graphical Models to understand the working of the algorithm.

Listing 1:

Consider the Hidden Markov Model which shows the performance of a student in an exam as 'Good', 'Moderate' and 'Fail' as hidden states in Figure 10.13. The observations we see on a student are 'Happy', 'Blink', 'Fine' and 'Fine'. Based on the observations we see on a student, we want to predict the state of how the student has written the exam. The decoding problem predicts the sequence of hidden states which is predicting the performance of the student in the exam.

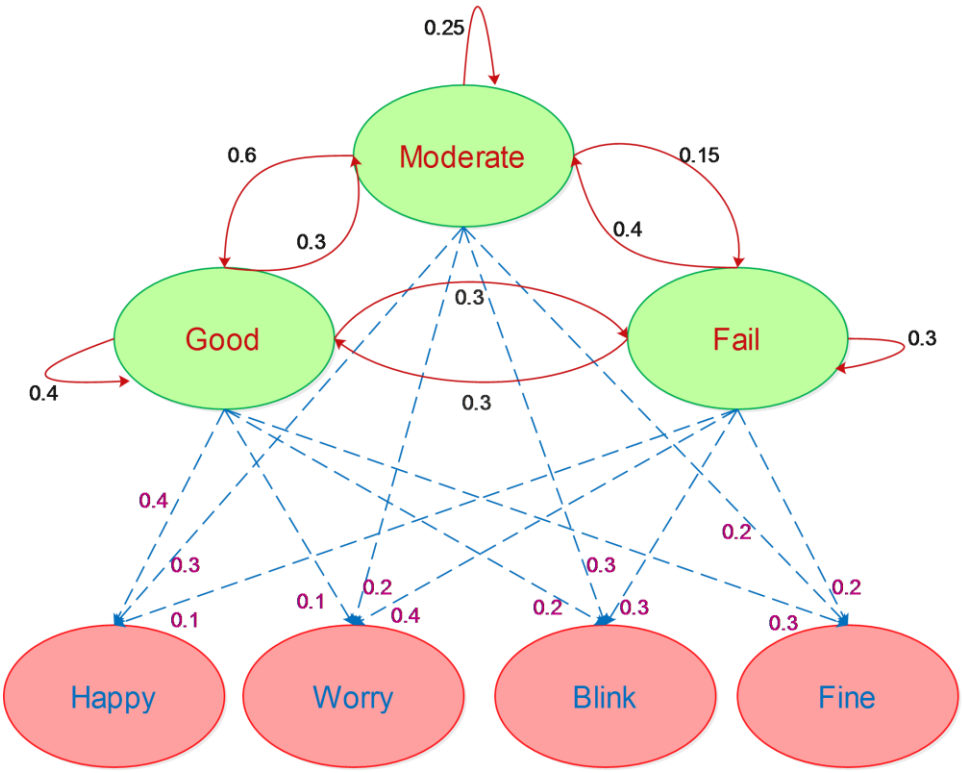


Figure 10.13: Sample Hidden Markov Model for Performance of a Student

Find the hidden states for the given observance sequence O_s = {Happy, Blink, Fine, Fine}.

3. Python Program with Explanation:

1. Initialize the 3 hidden states 'Good', 'Moderate' and 'Fail'.

```
states = ('Good', 'Moderate', 'Fail')
```

2. Initialize the 4 observation states 'Happy', 'Worry', 'Blink' and 'Fine'.

```
observations = ('Happy', 'Worry', 'Blink', 'Fine')
```

3. Set the initial probability of the hidden states.

```
initial_probability = {'Good': 0.5, 'Moderate': 0.25, 'Fail': 0.25}
```

4. Set the transition probability from each hidden state to other hidden states.

```
transition_probability = {
  'Good' : {'Good': 0.4, 'Moderate': 0.3, 'Fail' : 0.3},
  'Moderate' :   {'Good': 0.6, 'Moderate': 0.25, 'Fail' :0.15},
  'Fail' :   {'Good': 0.3, 'Moderate': 0.4, 'Fail' :0.3},
  }
```

5. Set the observation probability or emission probability from hidden states to observation states.

```
observation_probability = {
  'Good' : {'Happy': 0.4, 'Worry': 0.1, 'Blink': 0.2, 'Fine' :0.3 },
  'Moderate'   : {'Happy': 0.3, 'Worry': 0.2, 'Blink': 0.3, 'Fine' : 0.2},
  'Fail' : {'Happy': 0.1, 'Worry': 0.4, 'Blink': 0.3, 'Fine' :0.2 }
  }
```

6. Define the function ‘Viterbi’ to implement the logic of Viterbi algorithm. The function arguments are the observation sequence ‘obs’, the set of hidden states ‘states’, initial probability to hidden states i_pro, transition probability ‘t_pro’ and observation probability ‘o_pro’.

```
def Viterbi(obs, states, i_pro, t_pro, o_pro):
```

Set path to each state as empty.

```
path = { s:[] for s in states}
```

Set the current probability alpha_pro to empty.

```
alpha_pro = {}
```

Step 1: Initialization: Iterate this loop to calculate $\alpha_1(i)$ for each state i = 1 to no of states.

```
for s in states:
    alpha_pro[s] = i_pro[s]*o_pro[s][obs[0]]
    print(alpha_pro)
```

Step 2: Recursion: Iterate this loop to calculate $\alpha_t(j)$ for each observation from the second one in the observation sequence and for each state.

```
for i in range(1, len(obs)):
    last_pro = alpha_pro
    alpha_pro = {}
    for curr_state in states:
```

Calculate maximum probability at each step for each of the observation with each of the state.

```
max_pro, last_sta =
max(((last_pro[last_state]*t_pro[last_state][curr_state]*o_pro[curr_state][obs[i]],
last_state) for last_state in states))
                alpha_pro[curr_state] = max_pro
                print(alpha_pro)
                path[curr_state].append(last_sta)
```

Step 3: Termination: Find the maximum probability path.

```
max_pro = -1
max_path = None
for s in states:
    path[s].append(s)
    if alpha_pro[s] > max_pro:
        max_path = path[s]
        max_pro = alpha_pro[s]
```

```
        return max_path
```

7. Define the main function. Set the observation sequence ‘obs’ and call the Viterbi function defined.

```
    if __name__ == '__main__':
        obs = ['Happy', 'Blink', 'Fine', 'Fine']
        print("Given Observation Sequence\n", obs)
        print ("Maximum Probability Path \n", Viterbi(obs, states, initial_probability,
    transition_probability, observation_probability))
```

Complete Program:

```
states = ('Good', 'Moderate', 'Fail')

observations = ('Happy', 'Worry', 'Blink', 'Fine')

initial_probability = {'Good': 0.5, 'Moderate': 0.25, 'Fail': 0.25}

transition_probability = {
  'Good' : {'Good': 0.4, 'Moderate': 0.3, 'Fail' : 0.3},
  'Moderate' :   {'Good': 0.6, 'Moderate': 0.25, 'Fail' :0.15},
  'Fail' :   {'Good': 0.3, 'Moderate': 0.4, 'Fail' :0.3},
  }

observation_probability = {
  'Good' : {'Happy': 0.4, 'Worry': 0.1, 'Blink': 0.2, 'Fine' :0.3 },
  'Moderate'   : {'Happy': 0.3, 'Worry': 0.2, 'Blink': 0.3, 'Fine' : 0.2},
  'Fail' : {'Happy': 0.1, 'Worry': 0.4, 'Blink': 0.3, 'Fine' :0.2 }
  }

def Viterbi(obs, states, i_pro, t_pro, o_pro):
        path = { s:[] for s in states}
        print(path)
        alpha_pro = {}
        for s in states:
```

```
                alpha_pro[s] = i_pro[s]*o_pro[s][obs[0]]
                print(alpha_pro)
        for i in range(1, len(obs)):
                last_pro = alpha_pro
                alpha_pro = {}
                for curr_state in states:
                        max_pro, last_sta =
max(((last_pro[last_state]*t_pro[last_state][curr_state]*o_pro[curr_state][obs[i]], last_state)
                                for last_state in states))

                        alpha_pro[curr_state] = max_pro
                        print(alpha_pro)
                        path[curr_state].append(last_sta)

        # find the maximum probability path
        max_pro = -1
        max_path = None
        for s in states:
                path[s].append(s)
                if alpha_pro[s] > max_pro:
                        max_path = path[s]
                        max_pro = alpha_pro[s]
                return max_path

if __name__ == '__main__':
        obs = ['Happy', 'Blink', 'Fine', 'Fine']
        print("Given Observation Sequence\n", obs)
        print ("Maximum Probability Path \n", Viterbi(obs, states, initial_probability,
transition_probability, observation_probability))
```

Output:

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

========== RESTART: C:/Users/ADMIN/pythonpgms/final/jnf hmmtest.py ==========

Given Observation Sequence

['Happy', 'Blink', 'Fine', 'Fine']

{'Good': 0.2}

{'Good': 0.2, 'Moderate': 0.075}

{'Good': 0.2, 'Moderate': 0.075, 'Fail': 0.025}

{'Good': 0.016000000000000004}

{'Good': 0.016000000000000004, 'Moderate': 0.018}

{'Good': 0.016000000000000004, 'Moderate': 0.018, 'Fail': 0.018}

{'Good': 0.0032399999999999994}

{'Good': 0.0032399999999999994, 'Moderate': 0.00144}

{'Good': 0.0032399999999999994, 'Moderate': 0.00144, 'Fail': 0.00108}

{'Good': 0.00038879999999999996}

{'Good': 0.00038879999999999996, 'Moderate': 0.00019439999999999995}

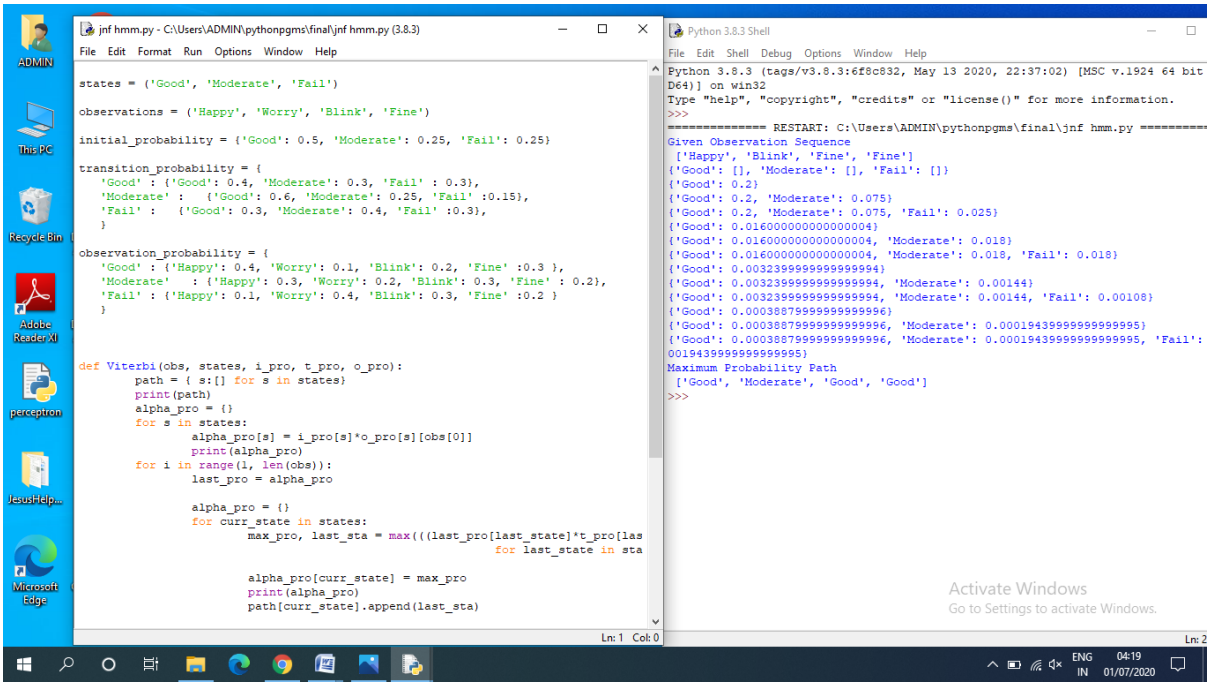
{'Good': 0.00038879999999999996, 'Moderate': 0.00019439999999999995, 'Fail': 0.00019439999999999995}

Maximum Probability Path

['Good', 'Moderate', 'Good', 'Good']

>>>

Screenshot of the Output:



Listing 2:

Refer Chapter 9 Probabilistic Graphical Models: Review Questions

Consider the Hidden Markov Model which shows the behaviour of a student going to school in the next morning as observation states 'Happy', 'Cry' and 'Dull' and hidden states 'Tuition', 'Sleep', ‘Movie’ and ‘Games’. Based on the observations seen on a student, predict the state of the student last evening.

Table 10.16 contains the observation probability, Table 10.17 contains the initial probability and Table 10.18 contains the transition probability.

Table 10.16: Observation Probability

Observation	**Happy**	**Cry**	**Dull**
Tuition	0.5	0.1	0.4
Sleep	0.35	0.45	0.2
Games	0.3	0.3	0.4
Movie	0.2	0.5	0.3

Table 10.17: Initial Probability

Initial Probability π			
0.25	0.25	0.25	0.25

Table 10.18: Transition Probability

Transition probability	**Tuition**	**Sleep**	**Games**	**Movie**
Tuition	0.4	0.3	0.2	0.1
Sleep	0.3	0.35	0.15	0.2
Games	0.6	0.05	0.1	0.25
Movie	0.1	0.5	0.2	0.2

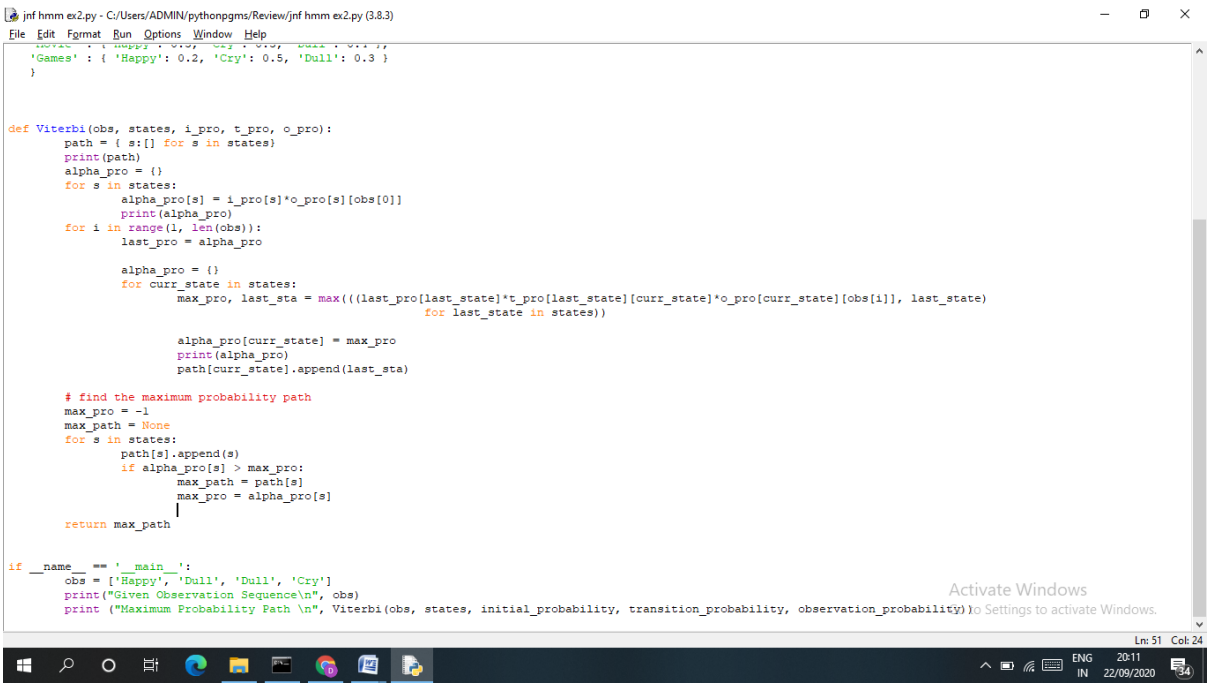
Assume the end probability as given in Table 10.19.

Table 10.19: End Probability

P(End|Tuition)	**P(End|Sleep)**	**P(End|Movie)**	**P(End|Games)**
0.1	0.2	0.1	0.2

Find the hidden states sequence for the given observance sequence O_s = {Happy, Dull, Dull, Cry}.

Screenshot of the Program:



Output:

Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license()" for more information.

>>>

========== RESTART: C:\Users\ADMIN\pythonpgms\Review\jnf hmm ex2.py ==========

Given Observation Sequence

['Happy', 'Dull', 'Dull', 'Cry']

{'Tuition': [], 'Sleep': [], 'Movie': [], 'Games': []}

{'Tuition': 0.125}

{'Tuition': 0.125, 'Sleep': 0.0875}

{'Tuition': 0.125, 'Sleep': 0.0875, 'Movie': 0.075}

{'Tuition': 0.125, 'Sleep': 0.0875, 'Movie': 0.075, 'Games': 0.05}

{'Tuition': 0.020000000000000004}

{'Tuition': 0.020000000000000004, 'Sleep': 0.0075}

{'Tuition': 0.020000000000000004, 'Sleep': 0.0075, 'Movie': 0.010000000000000002}

{'Tuition': 0.020000000000000004, 'Sleep': 0.0075, 'Movie': 0.010000000000000002, 'Games': 0.005625}

{'Tuition': 0.003200000000000001}

{'Tuition': 0.003200000000000001, 'Sleep': 0.0012000000000000003}

{'Tuition': 0.003200000000000001, 'Sleep': 0.0012000000000000003, 'Movie': 0.0016000000000000005}

{'Tuition': 0.003200000000000001, 'Sleep': 0.0012000000000000003, 'Movie': 0.0016000000000000005, 'Games': 0.0007500000000000001}

{'Tuition': 0.00012800000000000005}

{'Tuition': 0.00012800000000000005, 'Sleep': 0.0004320000000000001}

{'Tuition': 0.00012800000000000005, 'Sleep': 0.0004320000000000001, 'Movie': 0.0001920000000000009}

{'Tuition': 0.00012800000000000005, 'Sleep': 0.0004320000000000001, 'Movie': 0.0001920000000000009, 'Games': 0.0002000000000000006}

Maximum Probability Path

['Tuition', 'Tuition', 'Tuition', 'Sleep']

>>>

Screenshot of the Output: