INDEX

8	**A.** Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. **B.** Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.	39	

PRACTICAL NO.1

Practical No – 1A

Design a simple machine learning model to train the training instances and test the same using Python.

Code:

```
# Practical 1 - Linear Regression Model
print("Practical no.1A : Linear Regression \nName: Rohit \t Roll no.:09")
print("---------------------------------------------------")
import random
from sklearn.linear_model import LinearRegression

print("Read the train Data")
print("---------------------------------------------------")
feature_set = []
target_set = []
no_of_rows = 200
limit = 2000
for i in range(0, no_of_rows):
    x = random.randint(0, limit)
    y = random.randint(0, limit)
    z = random.randint(0, limit)
    g = 10 * x + 2 * y + 3 * z
    print("x=", x, "\ty=", y, "\tz=", z, "\tg=", g)
    feature_set.append([x, y, z])
    target_set.append(g)
print("Here the training of model begins. ")
model = LinearRegression()
model.fit(feature_set, target_set)
print("Training of model ends here!")

print("Testing started here")
test_data = [[1, 1, 0]]
print("Test data:", test_data)
prediction = model.predict(test_data)

print("prediction:" + str(prediction) + '\t' + "Coefficient:" +
str(model.coef_))
```

Output:

```
Practical no.1A : Linear Regression
Name: Rohit      Roll no.:09
-------------------------------------------------------
Read the train Data
-------------------------------------------------------
x= 1199     y= 1625     z= 718  g= 17394
x= 1942     y= 1275     z= 102  g= 22276
x= 1307     y= 417  z= 1243     g= 17633
x= 1294     y= 1087     z= 1492     g= 19590
x= 1085     y= 647  z= 1488     g= 16608
x= 476  y= 1948     z= 1836     g= 14164
x= 846  y= 1130     z= 12   g= 10756
x= 1720     y= 239  z= 191  g= 18251
x= 1967     y= 1351     z= 1766     g= 27670
x= 713  y= 1612     z= 1379     g= 14491
x= 326  y= 651  z= 971  g= 7475
x= 1402     y= 466  z= 237  g= 15663
x= 326  y= 1436     z= 973  g= 9051
x= 322  y= 62   z= 1479     g= 7781
x= 1047     y= 198  z= 510  g= 12396
x= 1309     y= 1547     z= 1028     g= 19268
Here the training of model begins.
Training of model ends here!
Testing started here
Test data: [[1, 1, 0]]
prediction:[12.]     Coefficient:[10.  2.  3.]
```

Practical No – 1B

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.

Training data → dataset_1b.csv

Sky	Temp	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

Code:

```
import csv
 num_attributes = 6 dataset = []
print("Name: Rohit \tRoll No: 09")
print("Training data")
 with open(r"data_find_s_1b.csv", "r") as csvfile:
    reader = csv.reader(csvfile)
for row in reader:
        dataset.append(row)

# print(row) # print(dataset) print("\n The
initial value of hypothesis: ") hypothesis =
['0'] * num_attributes print(hypothesis)
 for j in range(0, num_attributes):     hypothesis[j] =
dataset[1][j]     print("\n Find S: Finding a Maximally Specific
Hypothesis \n:")     for i in range(1, len(dataset)):        if
```

```
dataset[i][num_attributes] == 'Yes':             for j in range(0,
num_attributes):                    if dataset[i][j] != hypothesis[j]:
                    hypothesis[j] = '?'
else:
                    hypothesis[j] = dataset[i][j]          print("For
Training instance no:{0} the hypothesis is ".format(i), hypothesis)
    print("\n The Maximally Specific Hypothesis for a given training examples:
\n")     print(hypothesis)
```

Output:

```
Training data

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

 Find S: Finding a Maximally Specific Hypothesis
:
For Training instance no:1 the hypothesis is  ['Sunny', '?', '?', '?', '?', '?']
For Training instance no:2 the hypothesis is  ['Sunny', '?', '?', '?', '?', '?']
For Training instance no:3 the hypothesis is  ['Sunny', '?', '?', '?', '?', '?']
For Training instance no:4 the hypothesis is  ['Sunny', '?', '?', '?', '?', '?']

 The Maximally Specific Hypothesis for a given training examples:

['Sunny', '?', '?', '?', '?', '?']

 Find S: Finding a Maximally Specific Hypothesis
:
PROBLEMS 1    OUTPUT    DEBUG CONSOLE    TERMINAL    JUPYTER

For Training instance no:2 the hypothesis is  ['Sunny', 'Warm', '?', 'Strong', '?', '?']
For Training instance no:3 the hypothesis is  ['Sunny', 'Warm', '?', 'Strong', '?', '?']
For Training instance no:4 the hypothesis is  ['Sunny', 'Warm', '?', 'Strong', '?', '?']

 The Maximally Specific Hypothesis for a given training examples:

['Sunny', 'Warm', '?', 'Strong', '?', '?']

 Find S: Finding a Maximally Specific Hypothesis
:
For Training instance no:1 the hypothesis is  ['Sunny', 'Warm', '?', 'Strong', 'Warm', '?']
For Training instance no:2 the hypothesis is  ['Sunny', 'Warm', '?', 'Strong', 'Warm', '?']
For Training instance no:3 the hypothesis is  ['Sunny', 'Warm', '?', 'Strong', 'Warm', '?']
For Training instance no:4 the hypothesis is  ['Sunny', 'Warm', '?', 'Strong', '?', '?']

 The Maximally Specific Hypothesis for a given training examples:

['Sunny', 'Warm', '?', 'Strong', '?', '?']

 Find S: Finding a Maximally Specific Hypothesis
:
For Training instance no:1 the hypothesis is  ['Sunny', 'Warm', '?', 'Strong', '?', 'Same']
For Training instance no:2 the hypothesis is  ['Sunny', 'Warm', '?', 'Strong', '?', 'Same']
For Training instance no:3 the hypothesis is  ['Sunny', 'Warm', '?', 'Strong', '?', 'Same']
For Training instance no:4 the hypothesis is  ['Sunny', 'Warm', '?', 'Strong', '?', '?']

 The Maximally Specific Hypothesis for a given training examples:

['Sunny', 'Warm', '?', 'Strong', '?', '?']
PS C:\Users\Shalu\Documents\mscit\practical\Machine Learning> data_find_s_1b.csv
```

PRACTICAL NO.2

Practical No – 2A

Perform Data Loading, Feature selection (Principal Component analysis) and Feature Scoring and Ranking.

Code:

```
import pandas as pd from sklearn.model_selection
import train_test_split from sklearn.decomposition
import PCA from sklearn.preprocessing import
StandardScaler
 print("Name: Rohit \tRoll No: 09")


# Reading dataset using pandas url =
'https://archive.ics.uci.edu/ml/machine-
learningdatabases/iris/iris.data' names = ["sepal-length", "sepal-
width", "petal-length", "petal-width",
"class"] dataset = pd.read_csv(url,
names=names)


# Display dataset
print(dataset.head(10))
X = dataset.drop("class", 1)
Y = dataset["class"]


# Splitting the train and test datasets
 x_train, x_test, y_train, y_test = train_test_split(X, Y,
test_size=0.2, random_state=0)
 sc = StandardScaler() x_train =
sc.fit_transform(x_train) x_test =
sc.transform(x_test)


# Display Training and testing data
 print(f"\nDataSet before PCA :\n\nTrain
:\n{x_train}\n\nTest
:\n{x_test}")
```

```
# print(x_train)
# print(x_test)

# Creating PCA
pca = PCA() x_train = pca.fit_transform(x_train)
x_test = pca.transform(x_test)


# Giving a principal feature to model pca =
PCA(n_components=2) x_train =
pca.fit_transform(x_train) x_test =
pca.transform(x_test)
 print(f"\nDataSet After PCA :\n\nTrain :\n{x_train}\n\nTest
:\n{x_test}")
```

Output:

```
   sepal-length  sepal-width  petal-length  petal-width        class
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa
5           5.4          3.9           1.7          0.4  Iris-setosa
6           4.6          3.4           1.4          0.3  Iris-setosa
7           5.0          3.4           1.5          0.2  Iris-setosa
8           4.4          2.9           1.4          0.2  Iris-setosa
9           4.9          3.1           1.5          0.1  Iris-setosa
c:\Users\Shalu\Documents\mscit\practical\Machine Learning\prac_2a.py:15: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' wil
  X = dataset.drop("class", 1)

DataSet before PCA :

Train :
[[ 0.61303014  0.10850105  0.94751783  0.73603967]
 [-0.56776627 -0.12400121  0.38491447  0.34808318]
 [-0.80392556  1.03851009 -1.30289562 -1.3330616 ]
 [ 0.25879121 -0.12400121  0.60995581  0.73603967]
 [ 0.61303014 -0.58900572  1.00377816  1.25331499]
 [-0.80392556 -0.82150798  0.04735245  0.21876435]
```

Practical No – 2B

For a given set of training data examples stored in CSV file, implement and demonstrate the Candidate- Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

Data set :data_2b.csv

sky	airtemp	humidity	wind	water	forcast	enjoysport
sunny	warm	normal	strong	warm	same	yes
sunny	warm	high	strong	warm	same	yes
rainy	cold	high	strong	warm	change	no
sunny	warm	high	strong	cool	change	yes

Code:

```
import numpy as np import pandas as pd
 print("Name: Rohit \tRoll No: 09")
data = pd.read_csv(r'C:\Users\Shalu\Documents\mscit\practical\Machine
Learning\data_find_s_1b.csv') concepts = np.array(data.iloc[:, 0:-1])
print("\nInstances are:\n", concepts) target = np.array(data.iloc[:, -
1]) print("\nTarget Values are: ", target)
 def learn(concepts, target):
    specific_h = concepts[0].copy()     print("\nInitialization of
specific_h and genearal_h")     print("\nSpecific Boundary: ",
specific_h)     general_h = [["?" for i in range(len(specific_h))] for i
in range(len(specific_h))]     print("\nGeneric Boundary: ", general_h)
     for i, h in enumerate(concepts):
        print("\nInstance", i + 1, "is ", h)        if target[i] ==
"Yes":
            print("Instance is Positive ")                for x in
range(len(specific_h)):                if h[x] != specific_h[x]:
                    specific_h[x]          =          '?'
general_h[x][x] = '?'
        if target[i] == "No":
```

```
            print("Instance  is  Negative  ")
for      x      in      range(len(specific_h)):
if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
else:
                    general_h[x][x] = '?'
         print("Specific Bundary after ", i + 1, "Instance is ",
specific_h)             print("Generic Boundary after ", i + 1,
"Instance is ", general_h)          print("\n")
     indices = [i for i, val in enumerate(general_h) if val == ['?',
'?',
'?',  '?',  '?',  '?']]
for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
return specific_h, general_h
 s_final, g_final = learn(concepts,
target)
 print("Final Specific_h: ", s_final, sep="\n")
print("Final General_h: ", g_final, sep="\n")
```

Output:

```
Instances are:
 [['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]

Target Values are:  ['Yes' 'Yes' 'No' 'Yes']

Initialization of specific_h and genearal_h

Specific Boundary:  ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

Instance is Negative
Specific Bundary after  3 Instance is  ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
Generic Boundary after  3 Instance is  [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?
', 'Same']]


Instance 4 is  ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']
Instance is Positive
Specific Bundary after  4 Instance is  ['Sunny' 'Warm' '?' 'Strong' '?' '?']
Generic Boundary after  4 Instance is  [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?
', '?']]


Final Specific_h:
['Sunny' 'Warm' '?' 'Strong' '?' '?']
Final General_h:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
PS C:\Users\Shalu\Documents\mscit\practical\Machine Learning> []
```

PRACTICAL NO.3

Practical No – 3

Write a program to implement Decision Tree and Random forest with Prediction, Test Score and Confusion Matrix.

Code:

```
#   Decision tree import pandas as pd from sklearn.model_selection
import train_test_split from sklearn.tree import
DecisionTreeClassifier

from sklearn.metrics import confusion_matrix, accuracy_score,
classification_report # Read Datasets def read_datasets():

    print("---------------------------------------------------")

    datasets =
pd.read_csv("https://archive.ics.uci.edu/ml/machinelearning-
databases/balance-scale/balance-scale.data",
sep=",", header=None)     print(f"Dataset Length : {len(datasets)}")
print(f"Dataset Shape: {datasets.shape}")     print(f"Datasets :
{datasets.head()}")     return datasets

# Spliting Datasets into train and test def
splitdataset(datasets):

X = datasets.values[:, 1:5]

Y = datasets.values[:, 0]

    X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.3, random_state=100)

    # print(f"x_train:{X_train}\n x_test: {X_test}\n y_train:
{y_train}\n y_test: {y_test}")      return X_train, X_test,
y_train, y_test def train_using_gini(X_train, y_train):

    clf_gini = DecisionTreeClassifier(criterion='gini',
random_state=100, max_depth=3, min_samples_leaf=5)
clf_gini.fit(X_train, y_train)     return clf_gini

def train_using_entropy(X_train, y_train):
```

```
    clf_entropy = DecisionTreeClassifier(criterion='entropy',
random_state=100, max_depth=3, min_samples_leaf=5)
clf_entropy.fit(X_train, y_train)      return clf_entropy def
prediction(X_test, clf_object):

    y_pred = clf_object.predict(X_test)
print(f"Predicted values: {y_pred}")
return y_pred def cal_accuracy(y_test,
y_pred):

    print(f"Confusion Metrix :{confusion_matrix(y_test, y_pred)}")
print(f"Accuracy: {accuracy_score(y_test, y_pred) * 100}")
print(f"Report: {classification_report(y_test, y_pred)}") def
main():

    datasets = read_datasets()
    X_train, X_test, y_train, y_test = splitdataset(datasets)
clf_gini = train_using_gini(X_train, y_train)      clf_entropy
= train_using_entropy(X_train, y_train)      print("Results
using Gini Index: ")      y_pred_gini = prediction(X_test,
clf_gini)      cal_accuracy(y_test, y_pred_gini)
print("Results using Entropy Index: ")      y_pred_entropy =
prediction(X_test, clf_entropy)      cal_accuracy(y_test,
y_pred_entropy if __name__ == "__main__":

    main()
```

Output:

```
--------------------------------------------------
Dataset Length : 625
Dataset Shape: (625, 5)
Datasets :    0  1  2  3  4
0  B  1  1  1  1
1  R  1  1  1  2
2  R  1  1  1  3
3  R  1  1  1  4
4  R  1  1  1  5
Results using Gini Index:
Predicted values: ['R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L'
 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'L' 'R'
 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'R']
Confusion Metrix :[[ 0  6  7]
 [ 0 67 18]
 [ 0 19 71]]
Accuracy: 73.40425531914893
C:\Users\Shalu\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCa
ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parame
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Shalu\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCa
ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parame
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Shalu\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCa
ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parame
  _warn_prf(average, modifier, msg_start, len(result))
Report:               precision    recall  f1-score   support

           B       0.00      0.00      0.00        13
           L       0.73      0.79      0.76        85
           R       0.74      0.79      0.76        90

    accuracy                           0.73       188
   macro avg       0.49      0.53      0.51       188
weighted avg       0.68      0.73      0.71       188
```

```
Results using Entropy Index:
Predicted values: ['R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L'
 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L'
 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'R'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'R']
Confusion Metrix :[[ 0  6  7]
 [ 0 63 22]
 [ 0 20 70]]
Accuracy: 70.74468085106383
C:\Users\Shalu\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\Lo
ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Shalu\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\Lo
ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Shalu\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\Lo
ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` pa
  _warn_prf(average, modifier, msg_start, len(result))
Report:               precision    recall  f1-score   support

           B       0.00      0.00      0.00        13
           L       0.71      0.74      0.72        85
           R       0.71      0.78      0.74        90

    accuracy                           0.71       188
   macro avg       0.47      0.51      0.49       188
weighted avg       0.66      0.71      0.68       188

PS C:\Users\Shalu\Documents\mscit\practical\Machine Learning>
```

PRACTICAL NO.4

Practical No – 4A.

For a given set of training data examples stored in a .CSV file implement Least Square Regression algorithm.

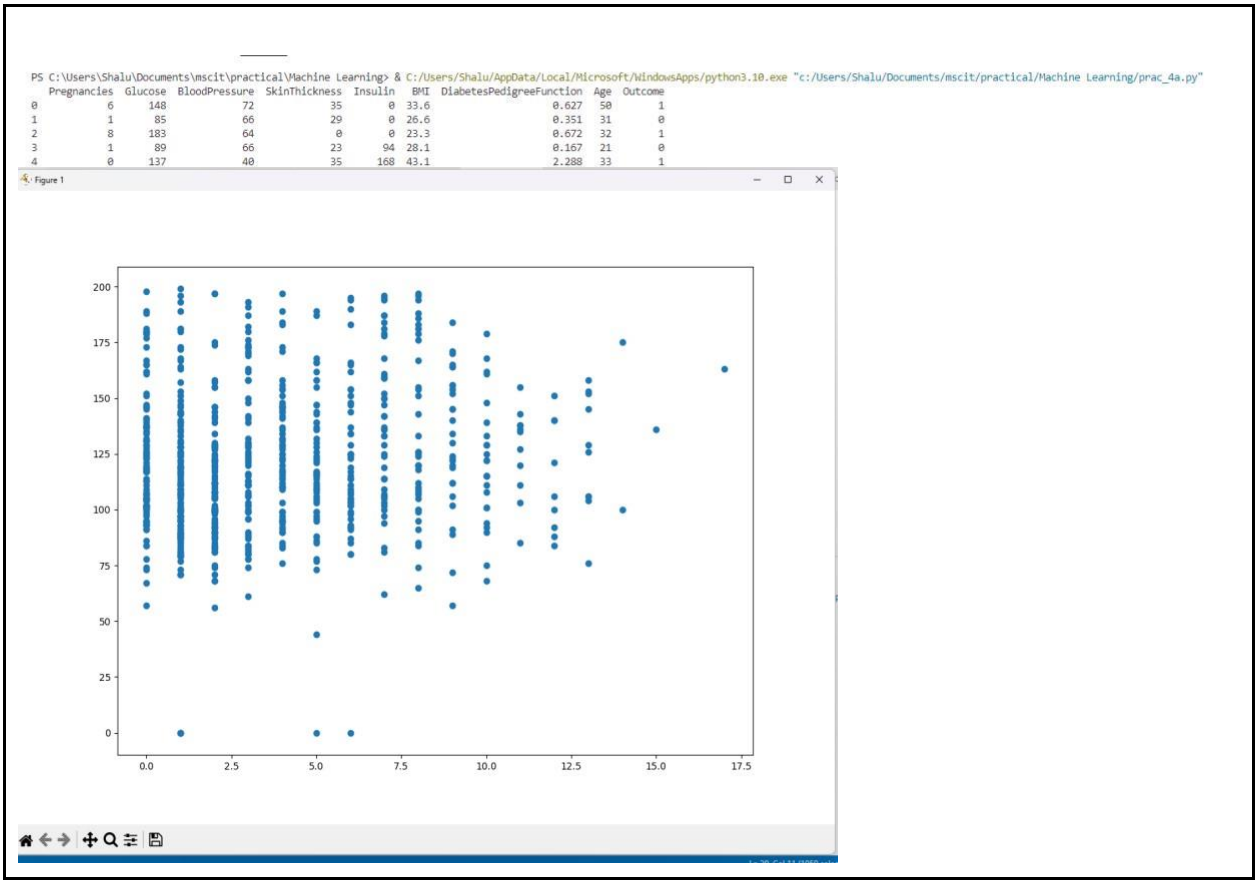
(Use Univariate dataset)

Code:

```
"""
Practical No - 4A.
 For a given set of training data examples stored in a .CSV file
implement Least Square Regression algorithm. (Use Univariate
dataset )
""" import pandas as pd import
numpy as np import
matplotlib.pyplot as plt
# from sklearn.model_selection import train_test_split
# from sklearn.preprocessing import StandardScaler
# from sklearn.linear_model import Lea
# from sklearn.metrics import confusion_matrix, accuracy_score
plt.rcParams['figure.figsize']=(12.0,9.0)
 dataset =
pd.read_csv("https://raw.githubusercontent.com/plotly/datasets/master/di
abetes.csv") print(dataset.head()) print("Name: Rohit, Roll no: 09") x =
dataset.iloc[:,0] y = dataset.iloc[:,1] plt.scatter(x, y) plt.show()
 x_mean = np.mean(x)
y_mean = np.mean(y)  num
= 0 den = 0  for i in
range(len(x)):
    num += (x[i] -x_mean)*(y[i]-y_mean)
    den += (x[i] -x_mean)**2
 m = num/den c =
y_mean - m*x_mean
print(m,c) y_pred =
m*x +c
```

```
plt.scatter(x,y) plt.plot([min(x), max(x)], [min(y_pred),
max(y_pred)], color="red") plt.show()
```

Output:



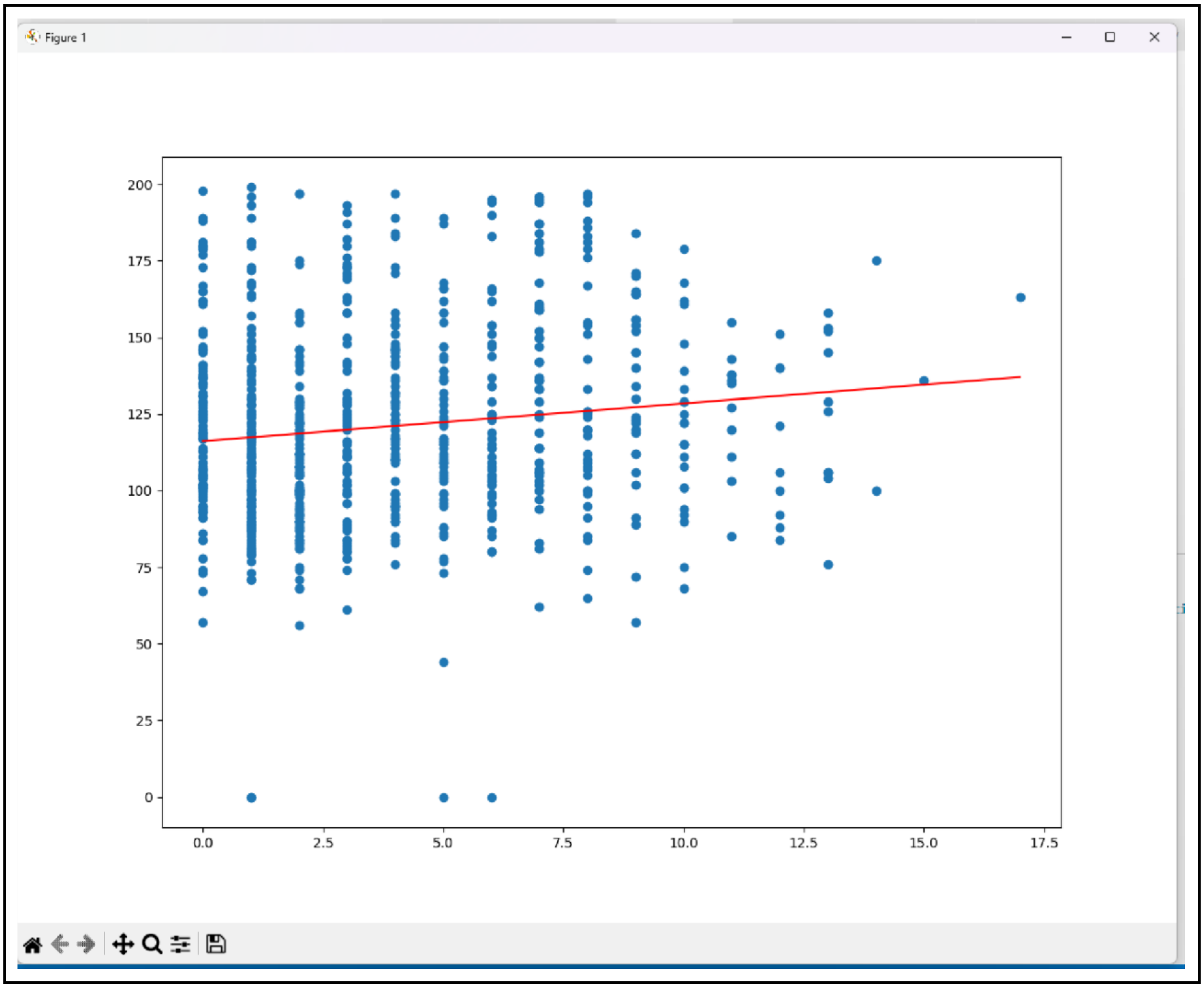


Figure 1
200
175
150
125
100
75
50
25
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

Practical No – 4B

For a given set of training data examples stored in a .CSV file implement Logistic Regression algorithm. (Use Multivariate dataset)

Code:

```
import pandas as pd from sklearn.model_selection import
train_test_split from sklearn.preprocessing import
StandardScaler from sklearn.linear_model import
LogisticRegression from sklearn.metrics import
confusion_matrix, accuracy_score

  print("Name: Rohit \tRoll No: 09")

 dataset =
pd.read_csv("https://raw.githubusercontent.com/plotly/da
tasets/master/di abetes.csv") print(dataset.head()) x =
dataset.iloc[:, [0, 1, 2, 3, 4, 5, 6, 7]].values y =
dataset.iloc[:, [-1]].values print(x) print(y)

 x_train, x_test, y_train, y_test = train_test_split(x,
y, test_size=0.2, random_state=0) sc = StandardScaler()
x_train = sc.fit_transform(x_train) x_test =
sc.transform(x_test)

 print(x_train[0:15, :])

 classifier    =    LogisticRegression()
classifier.fit(x_train, y_train)

 y_pred = classifier.predict(x_test)

 cm   =   confusion_matrix(y_test,   y_pred)
print("Confusion Matrix :\n ", cm)

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

Output:

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  DiabetesPedigreeFunction  Age  Outcome
0            6      148             72             35        0  33.6                     0.627   50        1
1            1       85             66             29        0  26.6                     0.351   31        0
2            8      183             64              0        0  23.3                     0.672   32        1
3            1       89             66             23       94  28.1                     0.167   21        0
4            0      137             40             35      168  43.1                     2.288   33        1
[[  6.    148.     72.    ...  33.6     0.627  50.    ]
 [  1.     85.     66.    ...  26.6     0.351  31.    ]
 [  8.    183.     64.    ...  23.3     0.672  32.    ]
 ...
 [  5.    121.     72.    ...  26.2     0.245  30.    ]
 [  1.    126.     60.    ...  30.1     0.349  47.    ]
 [  1.     93.     70.    ...  30.4     0.315  23.    ]]
[[1]
 [0]
 [1]
 [0]
 [1]
 [0]]
[-1.12606292 -0.05964563 -3.54371676 -1.28581572 -0.69965674  0.02444697
 -0.97758975 -0.79911377]
[ 2.65209354 -1.41253563 -0.47230103 -1.28581572 -0.69965674  0.07502976
 -0.86044863  0.61680026]
[-0.54480808 -0.3113461  -0.47230103 -1.28581572 -0.69965674 -0.75958622
 -0.37085882 -0.88240283]
[-0.25418066  0.12912972 -0.57468156 -1.28581572 -0.69965674 -0.0767186
 -0.94755357 -0.79911377]]
C:\Users\Shalu\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\sklearn\utils\validation.py:1111: DataConversionWarning: A column-vector y was passed when
a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
Confusion Matrix :
  [[98  9]
 [18 29]]
Accuracy : 0.8246753246753247
PS C:\Users\Shalu\Documents\mscit\practical\Machine Learning> 
```

PRACTICAL NO.5

Practical No - 5

Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set.

Code:

```
""" Write a program to implement k-Nearest
Neighbour algorithm to classify the iris data set.
 """ from sklearn.neighbors import KNeighborsClassifier from
sklearn.preprocessing import StandardScaler from sklearn.pipeline
import make_pipeline from sklearn import datasets from
sklearn.model_selection import train_test_split,GridSearchCV


iris=datasets.load_iris()
X=iris.data
Y=iris.target
## Create train and test split X_train,X_test,Y_train,Y_test =
train_test_split(X,Y,test_size=0.3,random_state=42,stratify=Y )
#feature  Scaling  using  StandardScalar
sc= StandardScaler() sc.fit(X_train)
X_train_std= sc.transform(X_train)
X_test_std= sc.transform(X_test)

#Fit the model
knn=KNeighborsClassifier(n_neighbors=5,p=2,weights='uniform',algorithm='
auto') knn.fit(X_train_std,Y_train)


#Evaluate the training and test score print("Name: Rohit \t Roll no:
09") print("Traing Accuracy score: %.3f "
%knn.score(X_train_std,Y_train)) print("Test Accuracy score: %.3f"
%knn.score(X_test_std,Y_test))


iris=datasets.load_iris()
X=iris.data
Y=iris.target
```

```
## Create train and test split X_train,X_test,Y_train,Y_test =
train_test_split(X,Y,test_size=0.3,random_state=42,stratify=Y )

#create a pipeline
pipeline=make_pipeline(StandardScaler(),KNeighborsClassifier())

#Create a parameter
param_grid=[{

    'kneighborsclassifier__n_neighbors':[2,3,4,5,6,7,8,9,10],

    'kneighborsclassifier__p':[1,2],

    'kneighborsclassifier__weights':['uniform','distance'],

'kneighborsclassifier__algorithm':['auto','ball_tree','kd_tree','brute'
}]


#Create a grid search instance
gs=GridSearchCV(pipeline, param_grid=param_grid,
scoring="accuracy",                  refit=True,
cv=10,                     verbose=1,
n_jobs=2)

 gs.fit(X_train,Y_train)


#print best model parameter and score
print("Best Score: %.3f " %gs.best_score_ ," \nBest
Parameters:",gs.best_params_)
```

Output:

```
Traing Accuracy score: 0.981
Test Accuracy score: 0.911
Fitting 10 folds for each of 144 candidates, totalling 1440 fits
Best Score: 0.972
Best Parameters: {'kneighborsclassifier__algorithm': 'auto', 'kneighborsclassifier__n_neighbors': 5, 'kneighborsclassifier__p': 1, 'kneighborsclassifier__weights': 'uniform'}
PS C:\Users\Shalu\Documents\mscit\practical\Machine Learning>
```

PRACTICAL NO.6

Practical No – 6A.

Implement the different Distance methods (Euclidean, Manhattan Distance, Minkowski Distance) with Prediction, Test Score and Confusion Matrix.

Code:

```
# Q. 6A. Implement the different Distance methods (Euclidean, Manhattan
Distance, Minkowski Distance)

# with Prediction, Test Score and Confusion Matrix. from math import

sqrt from sklearn.metrics import confusion_matrix,

classification_report

 print("Practical no.3 : Decision tree \nName: Rohit \t Roll no.:09")

print("----------------------------------------------------")      def

euclidian_distance(a, b):

    return sqrt(sum((e1 - e2) ** 2 for e1, e2 in zip(a, b))) def

manhattan_distance(a, b):

    return sum(abs(e1 - e2) for e1, e2 in zip(a, b))

def minkowski_distance(a, b, p):

    return sum(abs(e1 - e2) ** p for e1, e2 in zip(a, b)) ** (1 / p)

actual = [1, 0, 0, 1, 0, 0, 1, 0, 0, 1] predicted = [1, 0, 0, 1, 0,

0, 0, 1, 0, 0] dist1 = euclidian_distance(actual, predicted) dist2 =

manhattan_distance(actual, predicted) dist3 =

minkowski_distance(actual, predicted, 1)
```

```
print("Confusion_matrix: \n", matrix)

tp, fn, fp, tn = confusion_matrix(actual, predicted, labels=[1,
0]).reshape(-1)

print("Outcome values: \n", tp, fn, fp, tn)

matrix = classification_report(actual, predicted, labels=[1, 0])

print("Classification_report: \n", matrix)
```

```
print(f"Euclidian_dist: {dist1}\nManhattan_dist: {dist2}\nMinkowski_dist
with value 1: {dist3}") dist4 = minkowski_distance(actual, predicted, 2)
print(f"Minkowski_dist with value 2: {dist4}\n")

 matrix = confusion_matrix(actual, predicted, labels=[1, 0])
```

Output:

```
-------------------------------------------------
Euclidian_dist: 1.7320508075688772
Manhattan_dist: 3
Minkowski_dist with value 1: 3.0
Minkowski_dist with value 2: 1.7320508075688772

Confusion_matrix:
 [[2 2]
 [1 5]]
Outcome values:
 2 2 1 5
Classification_report:
              precision    recall  f1-score   support

           1       0.67      0.50      0.57         4
           0       0.71      0.83      0.77         6

    accuracy                           0.70        10
   macro avg       0.69      0.67      0.67        10
weighted avg       0.70      0.70      0.69        10

PS C:\Users\Shalu\Documents\mscit\practical\Machine Learning>
```

Practical No – 6B

Implement the classification model using clustering for the following techniques with K means clustering with Prediction, Test Score and Confusion Matrix.

Code:

```
# Common imports


import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

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


# Import the data set raw_data
= pd.read_csv(

    'https://raw.githubusercontent.com/imhardikj/A-Model-a-
Day/master/KNearest%20Neighbors/Classified%20Data.csv',     index_col=0)


# Standardize the data set
scaler = StandardScaler()

scaler.fit(raw_data.drop('TARGET CLASS', axis=1))
scaled_features = scaler.transform(raw_data.drop('TARGET CLASS',
axis=1))

scaled_data = pd.DataFrame(scaled_features,
columns=raw_data.drop('TARGET CLASS', axis=1).columns)


# Split the data set into training data and test data
```

```
x = scaled_data
y = raw_data['TARGET CLASS']
x_training_data, x_test_data, y_training_data, y_test_data =
train_test_split(x, y, test_size=0.3)
```

```
# Train the model and make predictions model =
KNeighborsClassifier(n_neighbors=1)
model.fit(x_training_data, y_training_data) predictions
= model.predict(x_test_data)


# Performance measurement
print(classification_report(y_test_data, predictions))
print(confusion_matrix(y_test_data, predictions))


# Selecting an optimal K value error_rates = []


# for i in np.arange(1, 101):
new_model = KNeighborsClassifier(n_neighbors=1)
new_model.fit(x_training_data, y_training_data) new_predictions =
new_model.predict(x_test_data)
error_rates.append(np.mean(new_predictions != y_test_data))
plt.figure(figsize=(16, 12)) plt.plot(error_rates) plt.show()
```

Output:

```
              precision    recall  f1-score   support

           0       0.92      0.91      0.92       143
           1       0.92      0.93      0.92       157

    accuracy                           0.92       300
   macro avg       0.92      0.92      0.92       300
weighted avg       0.92      0.92      0.92       300

[[130  13]
 [ 11 146]]
```

PRACTICAL NO.7

Practical No – 7A

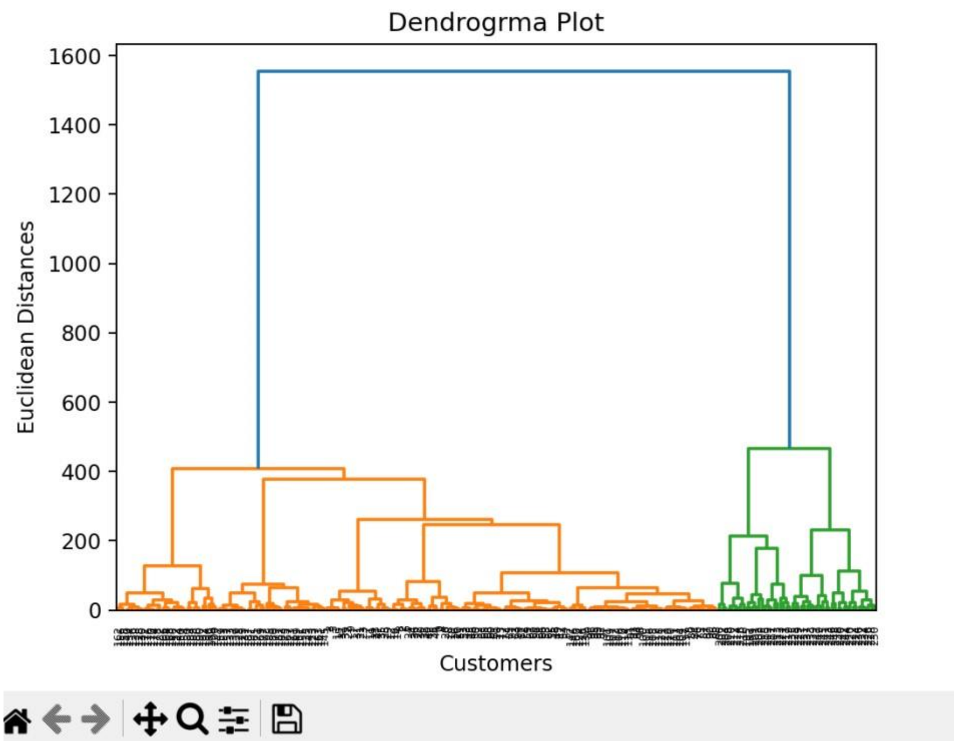
A. Implement the classification model using clustering for the following techniques with hierarchical clustering with Prediction, Test Score and Confusion Matrix

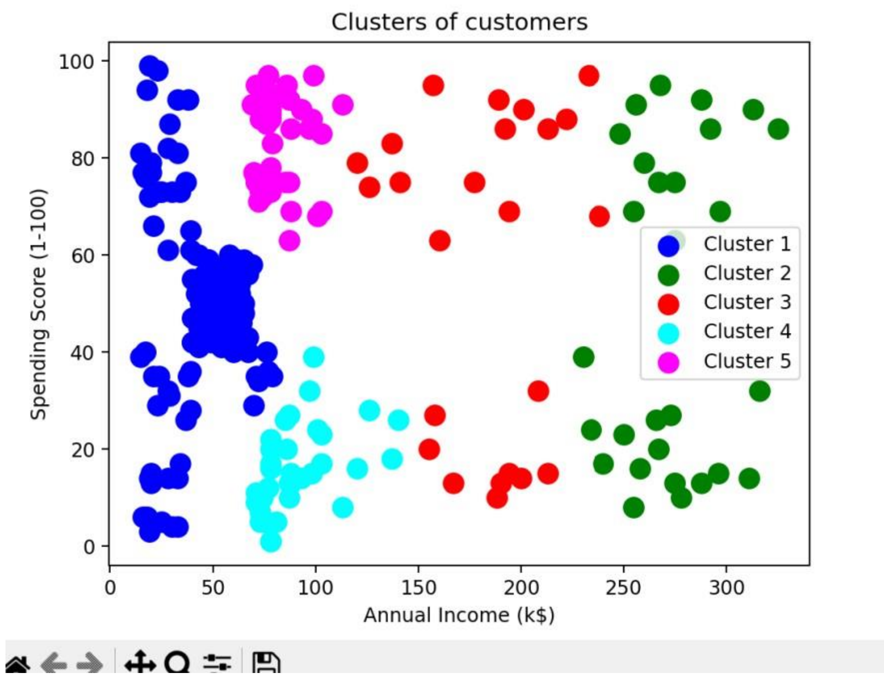
Code:

```
'''
Implement the classification model using clustering for the following
techniques with hierarchical clustering with Prediction,
Test Score and Confusion Matrix
'''

# Importing the libraries import
numpy as nm
import matplotlib.pyplot as mtp import
pandas as pd

# Importing the dataset
dataset = pd.read_csv('7_Mall_Customers.csv') x
= dataset.iloc[:, [3, 4]].values

# Finding the optimal number of clusters using the dendrogram import
scipy.cluster.hierarchy as shc

dendro = shc.dendrogram(shc.linkage(x, method="ward"))
mtp.title("Dendrogrma Plot") mtp.ylabel("Euclidean
Distances") mtp.xlabel("Customers") mtp.show()

# training the hierarchical model on dataset from
sklearn.cluster import AgglomerativeClustering

hc = AgglomerativeClustering(n_clusters=5, affinity='euclidean',
linkage='ward')
y_pred = hc.fit_predict(x)

# visulaizing the clusters
mtp.scatter(x[y_pred == 0, 0], x[y_pred == 0, 1], s=100, c='blue',
label='Cluster 1')
mtp.scatter(x[y_pred == 1, 0], x[y_pred == 1, 1], s=100, c='green',
label='Cluster 2')
mtp.scatter(x[y_pred == 2, 0], x[y_pred == 2, 1], s=100, c='red',
label='Cluster 3')
mtp.scatter(x[y_pred == 3, 0], x[y_pred == 3, 1], s=100, c='cyan',
label='Cluster 4')
mtp.scatter(x[y_pred == 4, 0], x[y_pred == 4, 1], s=100, c='magenta',
label='Cluster 5')
```

```
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend()
mtp.show()
```

Output:




Clusters of customers
Spending Score (1-100)
Annual Income (k$)
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
0
20
40
60
80
100
0
50
100
150
200
250
300

PRACTICAL NO.8

Practical No – 8A

Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.

Code:

```
"""
A. Write a program to construct a Bayesian network considering medical
data.
Use this model to demonstrate the diagnosis of heart patients using
standard Heart Disease Data Set.
"""
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianNetwork from
pgmpy.inference import VariableElimination

heartDisease = pd.read_csv('8a_heart_data.csv')
heartDisease = heartDisease.replace('?',np.nan)

print('Sample instances from the dataset are given below')
print(heartDisease.head())

print('\n Attributes and datatypes')
print(heartDisease.dtypes)

model=
BayesianNetwork([('age','heartdisease'),('gender','heartdisease'),('exan
g','heartdisease'),('cp','heartdisease'),('heartdisease','restecg'),('he
artdisease','chol')])
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)

print('\n 1. Probability of HeartDisease given evidence= restecg')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'re
stecg':1}) print(q1)

print('\n 2. Probability of HeartDisease given evidence= cp ')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp
':2})
```

```
print(q2)
```

Output:

```
Sample instances from the dataset are given below
   age  gender  cp  trestbps  chol  ...  oldpeak  slope  ca  thal  heartdisease
0   63       1   1       145   233  ...      2.3      3   0     6             0
1   67       1   4       160   286  ...      1.5      2   3     3             2
2   67       1   4       120   229  ...      2.6      2   2     7             1
3   37       1   3       130   250  ...      3.5      3   0     3             0
4   41       0   2       130   204  ...      1.4      1   0     3             0

[5 rows x 14 columns]

 Attributes and datatypes
age               int64
gender            int64
cp                int64
trestbps          int64
chol              int64
fbs               int64
restecg           int64
thalach           int64
exang             int64
oldpeak         float64
slope             int64
ca               object
thal             object
heartdisease      int64
dtype: object

 Inferencing with Bayesian Network:

 1. Probability of HeartDisease given evidence= restecg
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.1012 |
+-----------------+---------------------+
| heartdisease(1) |              0.0000 |
+-----------------+---------------------+
| heartdisease(2) |              0.2392 |
+-----------------+---------------------+
| heartdisease(3) |              0.2015 |
+-----------------+---------------------+
| heartdisease(4) |              0.4581 |
+-----------------+---------------------+

 2. Probability of HeartDisease given evidence= cp
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.3610 |
+-----------------+---------------------+
| heartdisease(1) |              0.2159 |
+-----------------+---------------------+
| heartdisease(2) |              0.1373 |
+-----------------+---------------------+
| heartdisease(3) |              0.1537 |
+-----------------+---------------------+
| heartdisease(4) |              0.1321 |
+-----------------+---------------------+
```

Practical No – 8B

Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Code:

```
""" NON Parametric model

 Implement the non-parametric Locally Weighted Regression algorithm in order
to fit data points.
 Select appropriate data set for your experiment and draw graphs.
"""

import matplotlib.pyplot as plt
import pandas as pd import numpy
as np
 def kernel(point, xmat, k):
m, n = np.shape(xmat)     weights
= np.mat(np.eye((m)))     for j in
range(m):
        diff = point - X[j]
        weights[j, j] = np.exp(diff * diff.T / (-2.0 * k ** 2))
return weights
 def localWeight(point, xmat, ymat, k):
    wei = kernel(point, xmat, k)
    W = (X.T * (wei * X)).I * (X.T * (wei * ymat.T))
return W
 def localWeightRegression(xmat, ymat, k):
    m, n = np.shape(xmat)
ypred = np.zeros(m)     for
i in range(m):
        ypred[i] = xmat[i] * localWeight(xmat[i], xmat, ymat, k)
return ypred

# load data points
data = pd.read_csv('8b_resturant_data.csv')
bill = np.array(data.total_bill) tip =
np.array(data.tip)

# preparing and add 1 in bill
mbill = np.mat(bill) mtip =
np.mat(tip)

m = np.shape(mbill)[1] one =
np.mat(np.ones(m)) X =
np.hstack((one.T, mbill.T))

# set k here
ypred = localWeightRegression(X, mtip, 0.5)
SortIndex = X[:, 1].argsort(0) xsort
= X[SortIndex][:, 0]
```

```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1) ax.scatter(bill,
tip, color='green')
```

```
ax.plot(xsort[:, 1], ypred[SortIndex], color='red', linewidth=5) plt.xlabel('Total
bill') plt.ylabel('Tip')
plt.show()
```

Output: