Guru Nanak College of Arts, Science & Commerce
GTB Nagar, Sion Koliwada, Mumbai – 37

Department of Information Technology

CERTIFICATE

This is to certify that Mr./Ms. **Jameel Shaikh**, Seat No. **3269635** studying in **Master of Science** in **Information Technology Part II (Semester III)** has satisfactorily completed the Practical of PSIT3P3a **Machine Learning** as prescribed by University of Mumbai, during the academic year **2022-23**.

Signature
Subject-In-Charge

Signature
Head of the Department

Signature
External Examiner

College Seal: ____________________

Date: ____________________

INDEX

Sr. No	Practical
1.	A. Design a simple machine learning model to train the training instances and test the same using Python. B. 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.
2.	A. Perform Data Loading, Feature selection (Principal Component analysis) and Feature Scoring and Ranking. B. 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.
3.	Write a program to implement Decision Tree and Random-Forest with Prediction, Test Score and Confusion Matrix.
4.	A. For a given set of training data examples stored in a .CSV file implement Least Square Regression algorithm. (Use Univariate dataset). B. For a given set of training data examples stored in a .CSV file implement Logistic Regression algorithm. (Use Multivariate dataset)
5.	Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set.
6.	A. Implement the different Distance methods (Euclidean, Manhattan Distance, Minkowski Distance) with Prediction, Test Score and Confusion Matrix. B. Implement the classification model using clustering for the following techniques with K means clustering with Prediction, Test Score and Confusion Matrix.
7.	A. Implement the classification model using clustering for the following techniques with hierarchical clustering with Prediction, Test Score and Confusion Matrix.
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.

Practical 01

Aim:

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

Description:

Code:

```python
from operator import imod
import random as r
from numpy import append
from sklearn.linear_model import LinearRegression

print("Name: Jameel Shaikh \t Roll No: 12")
print("Ready the training data")
print("=============================================")

feature_set = []
training_set = []
no_of_rows = 200
limit = 2000

for i in range(0, no_of_rows):
  x = r.randint(0, limit)
  y = r.randint(0, limit)
  z = r.randint(0, limit)
  g = (10 * x) + (2 * y) + (3 * z)

  print("=======================================================")
  print("X = ", x)
  print("Y = ", y)
  print("Z = ", z)
  print("G = ", g)
  print("=======================================================\n")

  feature_set.append([x,y,z])
  training_set.append(g)

print("====== Training of Model Started =======")
model = LinearRegression()
model.fit(feature_set, training_set)
print("====== Training of Model Ended =======\n")

print("====== Testing of Model Started =======")
print("====== Enter the testing data =======")
x = int(input("X = "))
y = int(input("Y = "))
z = int(input("Z = "))
test_data = [[x,y,z]]
prediction = model.predict(test_data)

print("Prediction: " + str(prediction) + "\t" + "Coefficient: " +
str(model.coef_))
```

Output:

```
========================================================
X =  1837
Y =  1864
Z =  658
G =  24072
========================================================

========================================================
X =  1748
Y =  640
Z =  1723
========================================================
X =  868
Y =  362
Z =  1466
G =  13802
========================================================

========================================================
X =  1863
Y =  315
Z =  281
G =  20103
========================================================

====== Training of Model Started =======
====== Training of Model Ended =======

====== Testing of Model Started =======
====== Enter the testing data =======
X = 220
Y = 303
Z = 222
Prediction: [3472.]     Coefficient: [10.  2.  3.]
```

B. **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.**

Description:

Code:

```
import csv

num_attributes = 6

dataset = []

print("============= TRAINING DATASET ================")

with open("D:\\MSc IT\\PART 2\\SEM 3\\ML Practicals\\Practical
2\\training.csv","r") as csvfile:
  readr = csv.reader(csvfile)
  for row in readr:
    dataset.append(row)
    print(row)

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 DATASET ================
['Sky', 'AirTemp', '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']
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']
['Rainy', 'Cold', 'Normal', 'Weak', 'Cool', 'Same', 'No']

 For Training instance No:6 the hypothesis is  ['Sunny', 'Warm', '?', 'Strong', '?', '?']

 Find S: Finding a Maximally Specific Hypothesis

 For Training instance No:6 the hypothesis is  ['Sunny', 'Warm', '?', 'Strong', '?', '?']

 Find S: Finding a Maximally Specific Hypothesis

 For Training instance No:6 the hypothesis is  ['Sunny', 'Warm', '?', 'Strong', '?', '?']

 The Maximally Specific Hypothesis for a given Training Examples :

['Sunny', 'Warm', '?', 'Strong', '?', '?']
PS D:\MSc IT\PART 2\SEM 3\ML Practicals>
```

Practical 02

Aim:

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

Description:

Code:

```
# Library Required For PCA
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Reading Dataset from iris.data which is available in pandas library

url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data"
names = ['sepal-length','sepal-width','petal-length','petal-width','Class']
dataset = pd.read_csv(url, names = names)

# Display iris.data Dataset

print(dataset.head())
x = dataset.drop('Class', 1)
y = dataset['Class']

# Splitting the dataset into the Training set and Test set

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("Dataset before PCA")
print(x_train)
print(x_test)

# Creating PCA object

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("Dataset after PCA")
print(x_train)
```

```python
print(x_test)
```

Output:

```
 [ 1.05608476e+00 -7.58640575e-01]
 [-2.19609621e+00  3.58093278e-01]
 [ 8.40144896e-01 -2.37005702e-02]
 [-2.04173388e-03 -2.28607127e-01]
 [ 6.76377991e-01  3.61953799e-01]
 [ 9.10905621e-01 -1.45303558e+00]
 [-2.06615479e+00  7.42185110e-01]
 [-2.21026982e+00  1.02028205e+00]
 [-2.23288987e+00 -3.82580267e-01]
 [ 1.35062418e+00 -1.17820187e-01]
 [ 6.30991357e-02 -8.12311123e-01]
 [ 2.39539989e+00  2.45039375e+00]
 [-2.43305139e+00 -2.86402494e-01]]
[[ 1.37128269 -0.51422498]
 [ 0.42475842 -1.81423595]
 [-2.41090531  2.26223189]
 [ 2.20457368  0.30506727]
 [-2.23288987 -0.38258027]
 [ 0.58056172 -0.26506855]
 [ 0.17606513 -1.14304791]
 [-2.12654085 -0.6311569 ]
 [-2.19666954  1.57948408]
 [ 0.88546786 -0.62093507]
 [ 0.27702729 -0.21588451]
 [-2.251877    0.23395773]
 [-2.39601679 -1.02161004]
 [ 1.07934479 -0.3804624 ]
 [-2.78023741  0.58014925]
 [-2.13280069  1.23915826]
 [ 0.49199447 -0.19294508]
 [-0.48313446 -2.02003782]
 [-2.07798432  0.32749755]]
PS D:\MSc IT\PART 2\SEM 3\ML Practicals>
```

B. **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.**

Description:

Code:

```
import numpy as np
import pandas as pd

print("Name: Jameel \tRoll No: 12")
data = pd.read_csv(r'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:

```
Name: Jameel    Roll No: 12

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']

Generic Boundary:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 1 is  ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
Instance is Positive
Specific Bundary after  1 Instance is  ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
Generic Boundary after  1 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?
'?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


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 D:\MSc_IT\PART 2\SEM 3\AI_ML\Machine Learning>
```

Practical 03

Aim:

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

Description:

Code:

```
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():
    datasets = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-
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):
```

```python
    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\crusher\AppData\Roaming\Python\Python39\site-packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning: Precision and F-score are ill-define
 being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\crusher\AppData\Roaming\Python\Python39\site-packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning: Precision and F-score are ill-define
 being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\crusher\AppData\Roaming\Python\Python39\site-packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning: Precision and F-score are ill-define
 being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _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
C:\Users\crusher\AppData\Roaming\Python\Python39\site-packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning: Precision and F-score are ill-define
```

Practical 04

Aim:

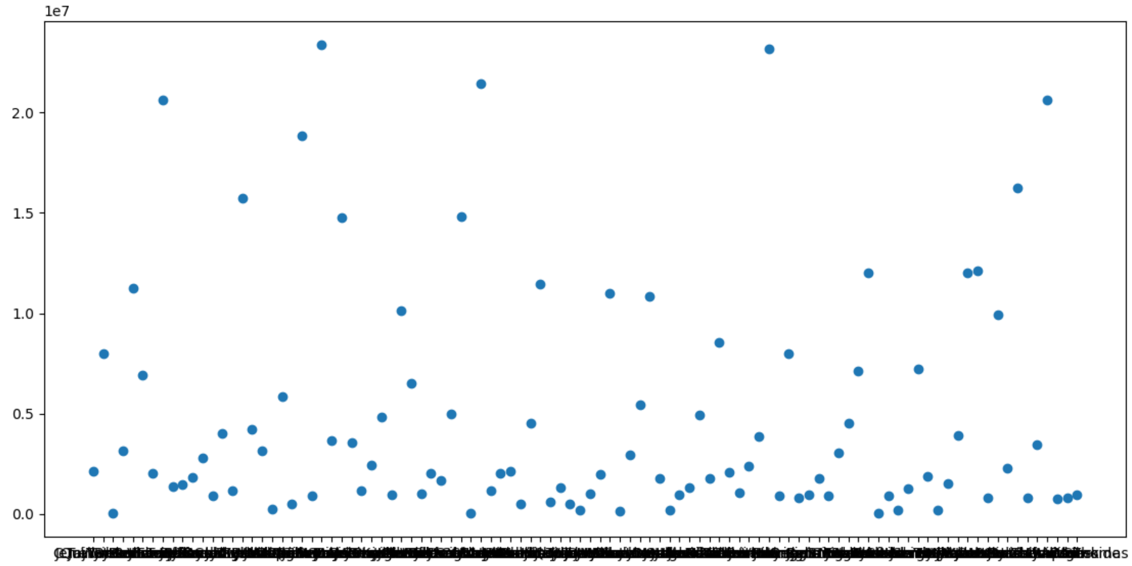
4A. For a given set of training data examples stored in a .CSV file implement Least Square Regression algorithm. (Use Univariate dataset)

Description:

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = (12.0, 9.0)

# Preprocessing Input Data
data = pd.read_csv('sample_salary_data.csv')
X = data.iloc[:, 0]
Y = data.iloc[:, 1]
plt.scatter(X, Y)
plt.show()

#Building the model
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)

#Making predictions
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:



4B. For a given set of training data examples stored in a .CSV file implement Logistic Regression algorithm. (Use Multivariate dataset).

Description:

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: Jameel \tRoll No: 12")

dataset =
pd.read_csv("https://raw.githubusercontent.com/plotly/datasets/master/diabetes.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:

```
 [1]
 [0]
 [0]
 [0]
 [0]
 [1]
 [0]]
[[ 0.90832902  0.91569367  0.44912368  0.52222619  0.3736349   0.37852648
   0.67740401  1.69955804]
 [ 0.03644676 -0.75182191 -0.47230103  0.14814855 -0.69965674 -0.50667229
  -0.07049698 -0.96569189]
  -0.43093119 -0.96569189]
 [-0.54480808  0.03474204 -0.47230103 -0.16358281  0.20327115 -0.30434114
   0.75249448 -0.96569189]
 [ 1.78021128  1.82810786  0.03960159 -1.28581572 -0.69965674  0.36588079
  -0.80037626  0.28364402]
 [-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\crusher\AppData\Roaming\Python\Python39\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 D:\MSc_IT\PART 2\SEM 3\AI_ML\Machine Learning>
```

Practical 05

Aim:

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

Description:

Code:

```
#Importing Libraries
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier

iris=load_iris()
iris.keys()

df=pd.DataFrame(iris['data'])
print(df)
print(iris['target_names'])

iris['feature_names']

X=df
y=iris['target']

#Splitting Dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=42)

#KNN Classifier and Training the model
knn=KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train,y_train)

#Prediction and Accuracy
y_pred=knn.predict(X_test)
cm=confusion_matrix(y_test,y_pred)
print(cm)
print("correct predicition",accuracy_score(y_test,y_pred))
print("worng predicition",(1-accuracy_score(y_test,y_pred)))
```

Output:

```
        0    1    2    3
0    5.1  3.5  1.4  0.2
1    4.9  3.0  1.4  0.2
2    4.7  3.2  1.3  0.2
3    4.6  3.1  1.5  0.2
4    5.0  3.6  1.4  0.2
..   ...  ...  ...  ...
145  6.7  3.0  5.2  2.3
146  6.3  2.5  5.0  1.9
147  6.5  3.0  5.2  2.0
148  6.2  3.4  5.4  2.3
149  5.9  3.0  5.1  1.8

[150 rows x 4 columns]
['setosa' 'versicolor' 'virginica']
[[19  0  0]
 [ 0 15  0]
 [ 0  1 15]]
 correct predicition 0.98
 worng predicition 0.020000000000000018
```

Practical 06

Aim:

6A. Implement the different Distance methods (Euclidean, Manhattan Distance, Minkowski Distance) with Prediction, Test Score and Confusion Matrix.

Description:

Code:

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


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(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])
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)
```

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
```

6B. Implement the classification model using clustering for the following techniques with K means clustering with Prediction, Test Score and Confusion Matrix.

Description:

Code:

```python
# 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('Classified_Data.csv',index_col=0)

# Import standardization functions from scikit-learn


# 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
```

```python
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)
```

Output:

```
              precision    recall  f1-score   support

           0       0.91      0.91      0.91       152
           1       0.91      0.91      0.91       148

    accuracy                           0.91       300
   macro avg       0.91      0.91      0.91       300
weighted avg       0.91      0.91      0.91       300

[[138  14]
 [ 14 134]]
PS D:\MSc_IT\PART 2\SEM 3\Machine Learning>
```

Practical 07

Aim:

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

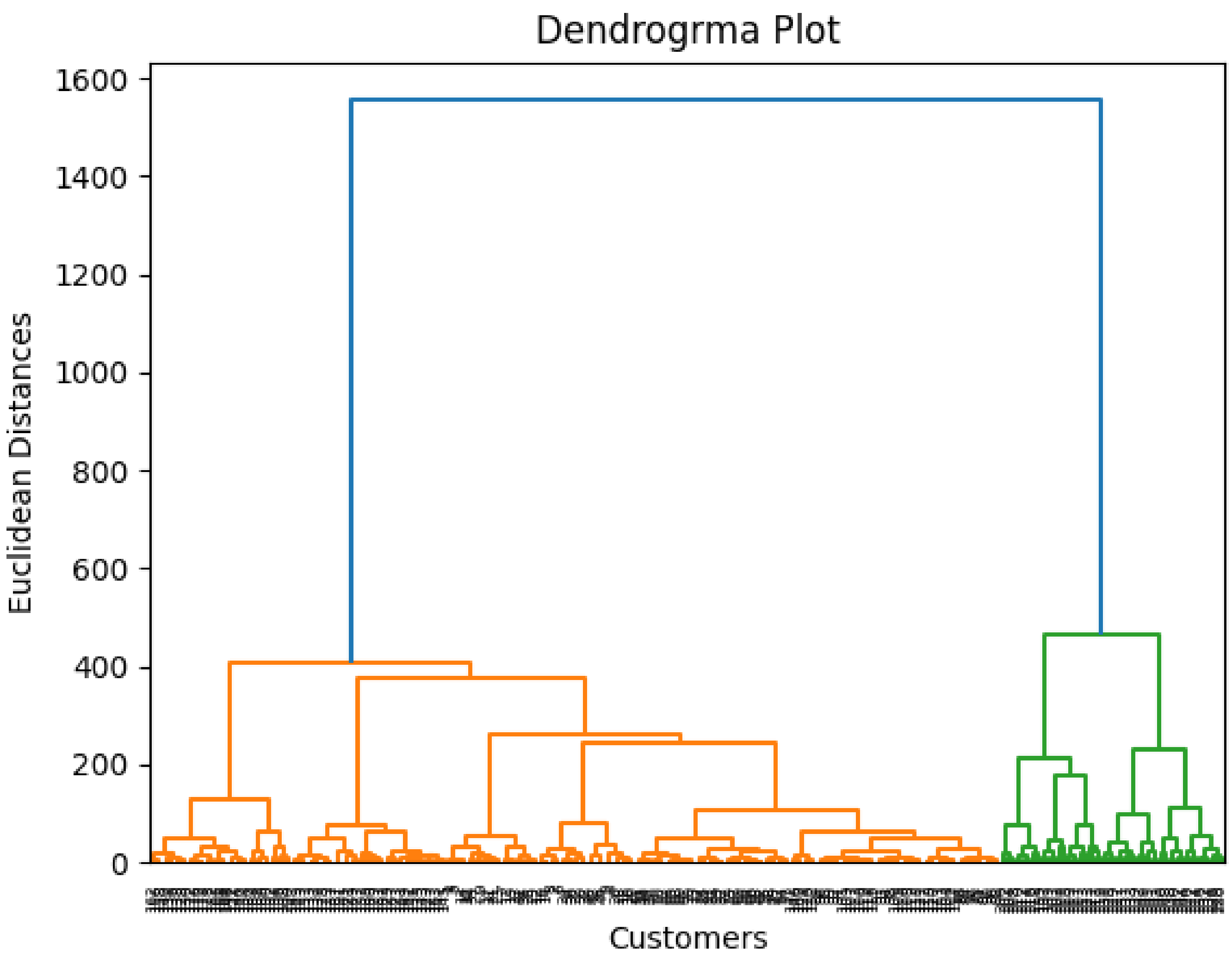
Description:

Code:

```
# 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:



Practical 08

Aim:

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.

Description:

Code:

```
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'),('exang','h
eartdisease'),('cp','heartdisease'),('heartdisease','restecg'),('heartdisease
','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={'restecg
':1})
print(q1)
```

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

Output:

```
 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 |
+-----------------+---------------------+
PS D:\MSc_IT\PART 2\SEM 3\Machine Learning>
```

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.

Description:

Code:

```
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)
```

```python
# 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: