Practical 1

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

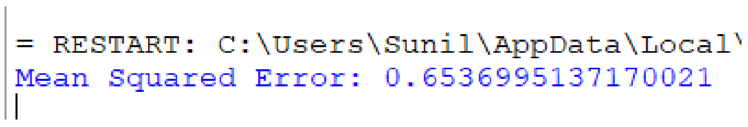
Aim: Design a simple machine learning model to train the training instances and test the same using Python.

Description: In Machine Learning Support Vector Machine is also called as the support vector network are supervised learning models that are associated with learning algorithms that analyze data that are used for classification & regression analysis.

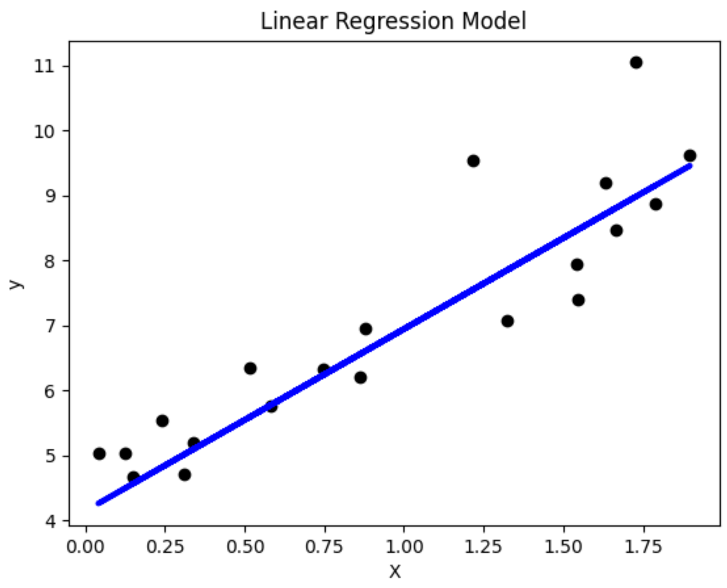
Code

```
# Import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np
# Generate some synthetic data for demonstration
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Create a linear regression model
model = LinearRegression()
# Train the model on the training data
model.fit(X_train, y_train)
# Make predictions on the test data
y_pred = model.predict(X_test)
# Evaluate the model performance
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
# Visualize the results (optional)
import matplotlib.pyplot as plt
plt.scatter(X_test, y_test, color='black')
plt.plot(X_test, y_pred, color='blue', linewidth=3)
plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear Regression Model')
plt.show()
```

Output:



```
= RESTART: C:\Users\Sunil\AppData\Local\
Mean Squared Error: 0.6536995137170021
```



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.

Aim: 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:

Algorithms:

- ☆ Initialize h to the meet specific hypothesis is H
- ☆ For each positive training instances x
  For each attribute constraint a in h
  If the constraint a is satisfied by x , then do nothing
  Else replace a in h by next more general constraint that is satisfied by x
- ☆ Output hypothesis h

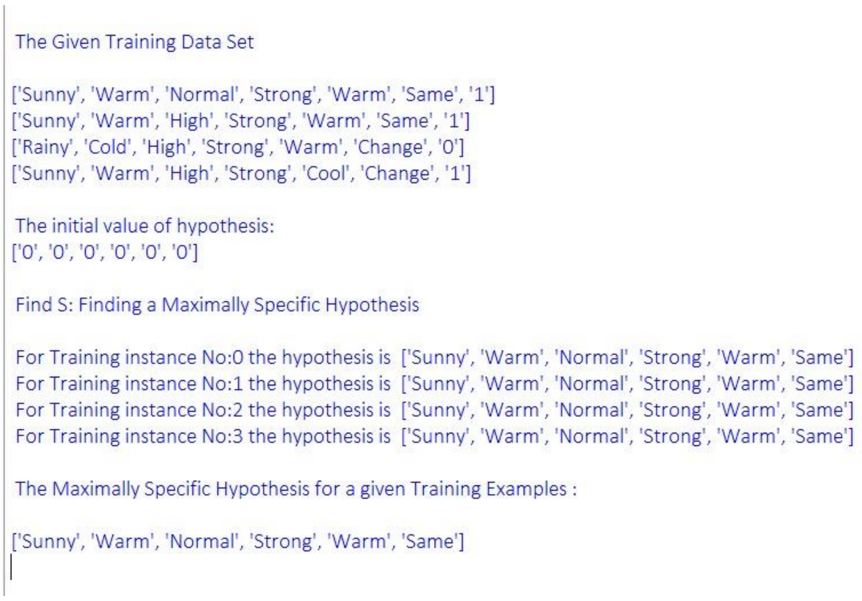
Dataset Description:

- ☆ The unique dataset speaks about how many origin are available within specific size as well as it is available with rates.
- ☆ The origin one of the commonly cited drawbacks of the find-S algorithms is that hypothesis h
- ☆ Im dataset has mention origin about manufacture & its types.

Code:

```
import csv
num_attributes = 6
a = []
print("\n The Given Training Data Set \n")
with open('ENJOYSPORT.csv', 'r') as csvfile:
   reader = csv.reader(csvfile)
   for row in reader:
      a.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] = a[0][j];
print("\n Find S: Finding a Maximally Specific Hypothesis\n")
for i in range(0,len(a)):
    if a[i][num_attributes]=='yes':
        for j in range(0,num_attributes):
            if a[i][j]!=hypothesis[j]:
                hypothesis[j]='?'
            else :
                hypothesis[j]= a[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)
```



```
 The Given Training Data Set

['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', '1']
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', '1']
['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', '0']
['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', '1']

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

 Find S: Finding a Maximally Specific Hypothesis

 For Training instance No:0 the hypothesis is  ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
 For Training instance No:1 the hypothesis is  ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
 For Training instance No:2 the hypothesis is  ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
 For Training instance No:3 the hypothesis is  ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']

 The Maximally Specific Hypothesis for a given Training Examples :

['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
```

Practical 2

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

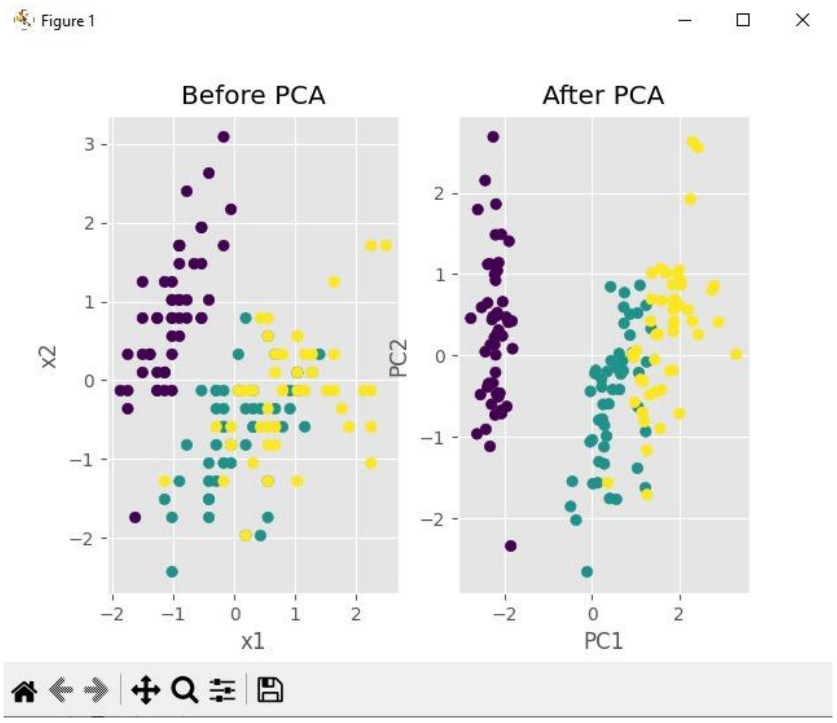
Aim: Perform Data Loading, Feature selection (Principal Component analysis) and Feature Scoring and Ranking.

Description: PCA is a statistical procedure that uses on orthogonal transformation that convert a set of correlated variable to a set of uncorrelated variable.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
import pandas as pd
from sklearn.preprocessing import StandardScaler
plt.style.use('ggplot')
# Load the data
iris = datasets.load_iris()
X = iris.data
y = iris.target
# Z-score the features
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)
# The PCA model
pca = PCA(n_components=2) # estimate only 2 PCs
X_new = pca.fit_transform(X) # project the original data into the PCA space
fig, axes = plt.subplots(1,2)
axes[0].scatter(X[:,0], X[:,1], c=y)
axes[0].set_xlabel('x1')
axes[0].set_ylabel('x2')
axes[0].set_title('Before PCA')
axes[1].scatter(X_new[:,0], X_new[:,1], c=y)
axes[1].set_xlabel('PC1')
axes[1].set_ylabel('PC2')
axes[1].set_title('After PCA')
plt.show()
```

Output:



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.

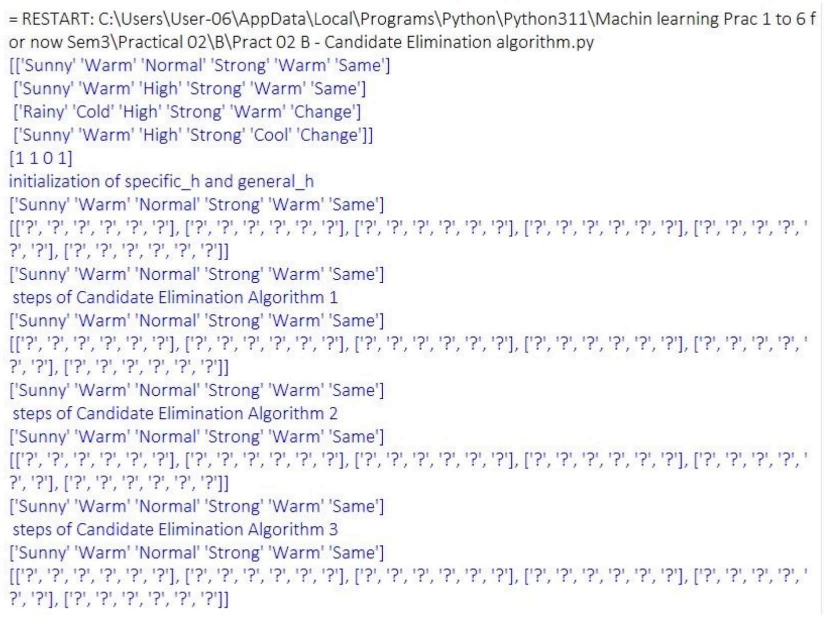
Aim: For a given set of training data examples stored in .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

Description: The Candidate-Elimination algorithm is a concept learning algorithm that is used for learning a hypothesis space from training examples. It maintains a set of hypotheses that are consistent with the observed examples and eliminates hypotheses that are inconsistent. The algorithm incrementally updates the set of generalizations and specializations based on positive and negative examples.

Code:

```
import numpy as np
import pandas as pd
data = pd.DataFrame(data=pd.read_csv('ENJOYSPORT.csv'))
concepts = np.array(data.iloc[:,0:-1])
print(concepts)
target = np.array(data.iloc[:,-1])
print(target)
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("initialization of specific_h and general_h")
    print(specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]
    print(general_h)
    for i, h in enumerate(concepts):
        if target[i] == "yes":
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'
                print(specific_h)
        print(specific_h)
        if target[i] == "no":
            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(" steps of Candidate Elimination Algorithm",i+1)
        print(specific_h)
        print(general_h)
    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:



```
= RESTART: C:\Users\User-06\AppData\Local\Programs\Python\Python311\Machin learning Prac 1 to 6 f
or now Sem3\Practical 02\B\Pract 02 B - Candidate Elimination algorithm.py
[['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]
[1 1 0 1]
initialization of specific_h and general_h
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '
?', '?'], ['?', '?', '?', '?', '?', '?']]
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 steps of Candidate Elimination Algorithm 1
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '
?', '?'], ['?', '?', '?', '?', '?', '?']]
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 steps of Candidate Elimination Algorithm 2
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '
?', '?'], ['?', '?', '?', '?', '?', '?']]
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 steps of Candidate Elimination Algorithm 3
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '
?', '?'], ['?', '?', '?', '?', '?', '?']]
```

Practical 3

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

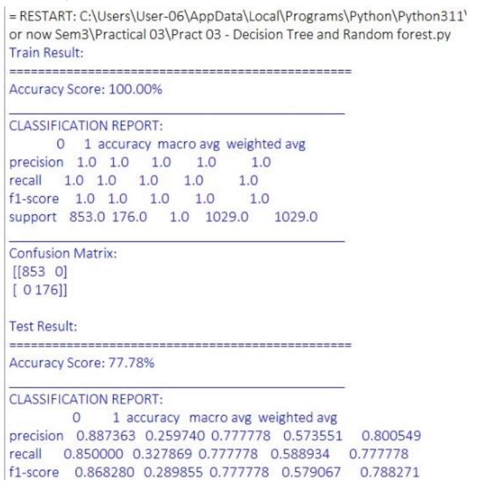
Aim: Write a program to implement Decision Tree and Random forest with Prediction, Test Score and Confusion Matrix.

Description:

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
#%matplotlib inline
sns.set_style("whitegrid")
plt.style.use("fivethirtyeight")
df = pd.read_csv("WA_Fn-UseC_-HR-Employee-Attrition.csv")
df.head()
sns.countplot(x='Attrition', data=df)
df.drop(['EmployeeCount', 'EmployeeNumber', 'Over18', 'StandardHours'],
axis="columns", inplace=True)
categorical_col = []
for column in df.columns:
   if df[column].dtype == object and len(df[column].unique()) <= 50:
      categorical_col.append(column)
df['Attrition'] = df.Attrition.astype("category").cat.codes
categorical_col.remove('Attrition')
from sklearn.preprocessing import LabelEncoder
label = LabelEncoder()
for column in categorical_col:
   df[column] = label.fit_transform(df[column])
   from sklearn.model_selection import train_test_split
X = df.drop('Attrition', axis=1)
y = df.Attrition
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
def print_score(clf, X_train, y_train, X_test, y_test, train=True):
   if train:
      pred = clf.predict(X_train)
      clf_report = pd.DataFrame(classification_report(y_train, pred,
output_dict=True))
      print("Train
Result:\n================================================")
      print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
      print("_______________________________________________")
      print(f"CLASSIFICATION REPORT:\n{clf_report}")
      print("_______________________________________________")
      print(f"Confusion Matrix: \n {confusion_matrix(y_train, pred)}\n")
   elif train==False:
      pred = clf.predict(X_test)
      clf_report = pd.DataFrame(classification_report(y_test, pred,
output_dict=True))
      print("Test
Result:\n================================================")
      print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
      print("_______________________________________________")
      print(f"CLASSIFICATION REPORT:\n{clf_report}")
      print("_______________________________________________")
      print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n")
from sklearn.tree import DecisionTreeClassifier
tree_clf = DecisionTreeClassifier(random_state=42)
tree_clf.fit(X_train, y_train)
print_score(tree_clf, X_train, y_train, X_test, y_test, train=True)
print_score(tree_clf, X_train, y_train, X_test, y_test, train=False)
```

Output:



```
= RESTART: C:\Users\User-06\AppData\Local\Programs\Python\Python311\
or now Sem3\Practical 03\Pract 03 - Decision Tree and Random forest.py
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
CLASSIFICATION REPORT:
           0     1  accuracy  macro avg  weighted avg
precision   1.0   1.0      1.0        1.0           1.0
recall      1.0   1.0      1.0        1.0           1.0
f1-score    1.0   1.0      1.0        1.0           1.0
support   853.0 176.0      1.0     1029.0        1029.0
_______________________________________________
Confusion Matrix:
 [[853   0]
 [  0 176]]

Test Result:
================================================
Accuracy Score: 77.78%
_______________________________________________
CLASSIFICATION REPORT:
                  0         1  accuracy   macro avg  weighted avg
precision  0.887363  0.259740  0.777778   0.573551      0.800549
recall     0.850000  0.327869  0.777778   0.588934      0.777778
f1-score   0.868280  0.289855  0.777778   0.579067      0.788271
```

Practical 4

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

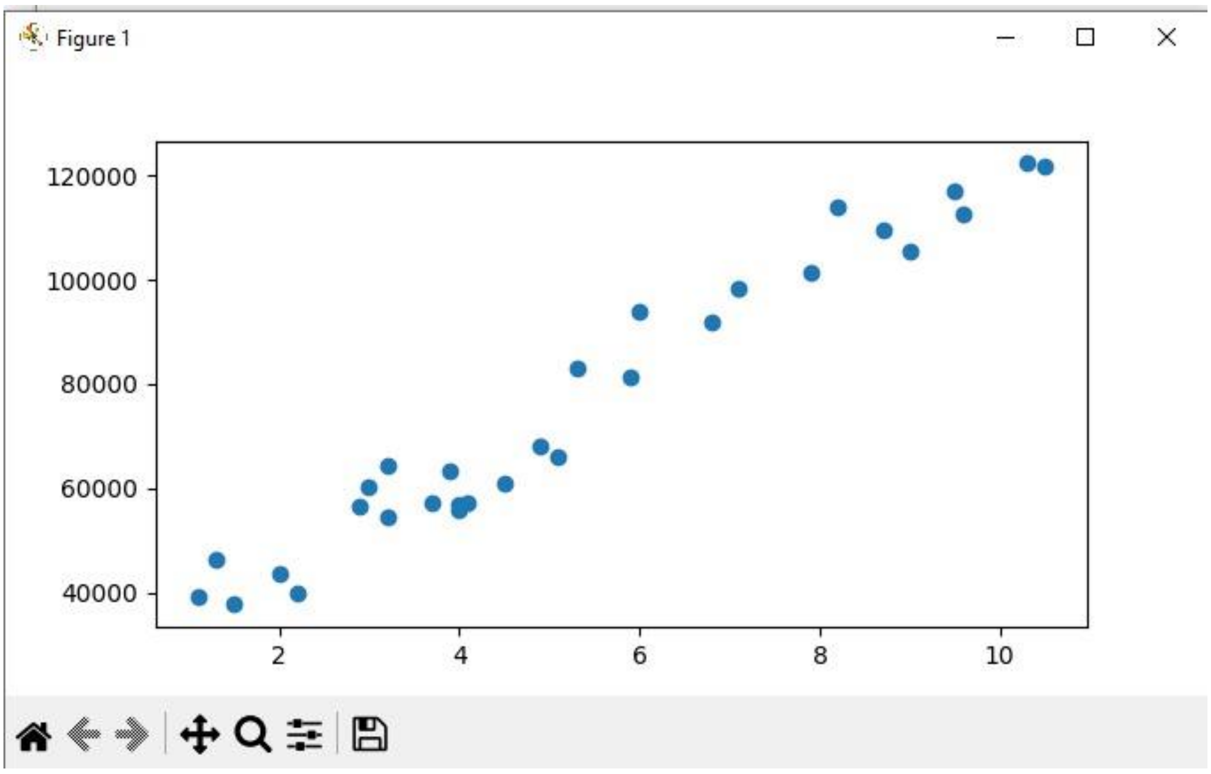
Aim: For a given set of training data examples stored in a .CSV file implement Least Square Regression algorithm. (Use Univariate dataset)

Description: The Least Square method is a technique commonly used in regression analysis. It is a mathematical method used to find the best fit line that represent the relationship between an independent & dependent variable.

Code:

```
# Making imports
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('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) # actual
# plt.scatter(X, Y_pred, color='red')
plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)], color='red') # predicted
plt.show()
```

Output:



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

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

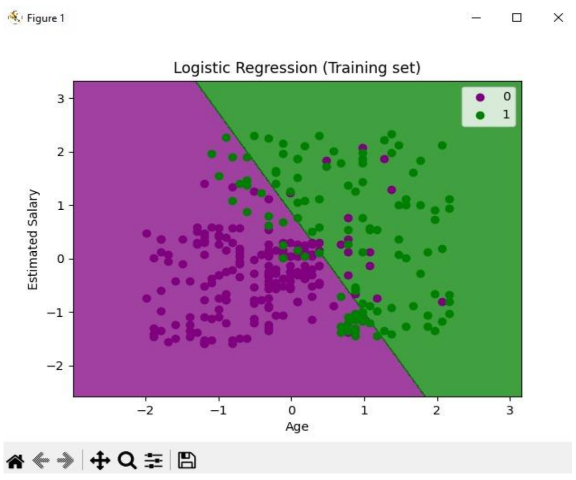
Description: Logistic learning comes under the supervised learning technique. It is a classification algorithm that is used to predict the discrete values

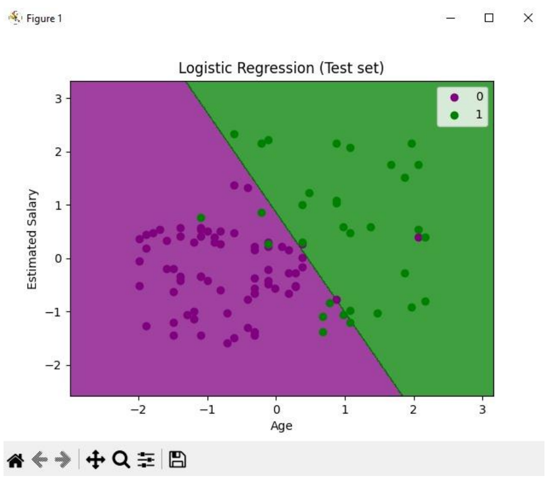
Code:

```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
#importing datasets
data_set= pd.read_csv('user_data.csv')
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25,
random_state=0)
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
from sklearn.linear_model import LogisticRegression
classifier= LogisticRegression(random_state=0)
classifier.fit(x_train, y_train)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                intercept_scaling=1, l1_ratio=None, max_iter=100,
                multi_class='warn', n_jobs=None, penalty='l2',
                random_state=0, solver='warn', tol=0.0001, verbose=0,
                warm_start=False)
y_pred= classifier.predict(x_test)
from sklearn.metrics import confusion_matrix
#cm= confusion_matrix()
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() +
1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
```

```
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
   mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
      c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Logistic Regression (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() +
1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
   mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
      c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Logistic Regression (Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output:





Practical 5

A. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Aim: Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

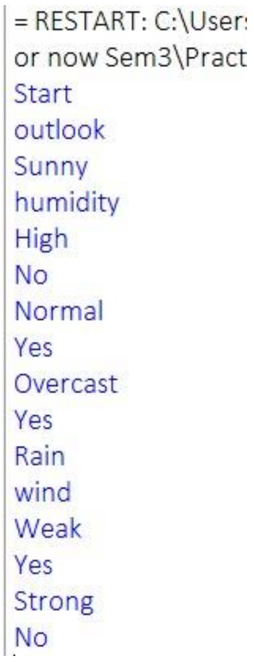
Description: It is a classification algorithm that follows a approach by selecting the best attribute that yields maximum Information Gain (IG) or minimum entropy(H).

Code:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math
import copy
dataset = pd.read_csv('Tennis.csv')
X = dataset.iloc[:, 1:].values
# print(X)
attribute = ['outlook', 'temp', 'humidity', 'wind']
class Node(object):
    def init(self):
        self.value = None
        self.decision = None
        self.childs = None
def findEntropy(data, rows):
    yes = 0
    no = 0
    ans = -1
    idx = len(data[0]) - 1
    entropy = 0
    for i in rows:
        if data[i][idx] == 'Yes':
            yes = yes + 1
        else:
            no = no + 1
    x = yes/(yes+no)
    y = no/(yes+no)
    if x != 0 and y != 0:
        entropy = -1 * (x*math.log2(x) + y*math.log2(y))
    if x == 1:
        ans = 1
    if y == 1:
        ans = 0
    return entropy, ans
def findMaxGain(data, rows, columns):
    maxGain = 0
    retidx = -1
    entropy, ans = findEntropy(data, rows)
    if entropy == 0:
        """if ans == 1:
            print("Yes")
        else:
            print("No")"""
        return maxGain, retidx, ans
    for j in columns:
        mydict = {}
        idx = j
        for i in rows:
            key = data[i][idx]
            if key not in mydict:
                mydict[key] = 1
            else:
                mydict[key] = mydict[key] + 1
        gain = entropy
        # print(mydict)
        for key in mydict:
            yes = 0
            no = 0
            for k in rows:
                if data[k][j] == key:
                    if data[k][-1] == 'Yes':
                        yes = yes + 1
                    else:
```

```
                    no = no + 1
                # print(yes, no)
                x = yes/(yes+no)
                y = no/(yes+no)
                # print(x, y)
                if x != 0 and y != 0:
                    gain += (mydict[key] * (x*math.log2(x) + y*math.log2(y)))/14
            # print(gain)
            if gain > maxGain:
                # print("hello")
                maxGain = gain
                retidx = j
    return maxGain, retidx, ans
def buildTree(data, rows, columns):
    maxGain, idx, ans = findMaxGain(X, rows, columns)
    root = Node()
    root.childs = []
    # print(maxGain
    #
    # )
    if maxGain == 0:
        if ans == 1:
            root.value = 'Yes'
        else:
            root.value = 'No'
        return root
    root.value = attribute[idx]
    mydict = {}
    for i in rows:
        key = data[i][idx]
        if key not in mydict:
            mydict[key] = 1
        else:
            mydict[key] += 1
    newcolumns = copy.deepcopy(columns)
    newcolumns.remove(idx)
    for key in mydict:
        newrows = []
        for i in rows:
            if data[i][idx] == key:
                newrows.append(i)
        # print(newrows)
        temp = buildTree(data, newrows, newcolumns)
        temp.decision = key
        root.childs.append(temp)
    return root
def traverse(root):
    print(root.decision)
    print(root.value)
    n = len(root.childs)
    if n > 0:
        for i in range(0, n):
            traverse(root.childs[i])
def calculate():
    rows = [i for i in range(0, 14)]
    columns = [i for i in range(0, 4)]
    root = buildTree(X, rows, columns)
    root.decision = 'Start'
    traverse(root)
calculate()
```

Output:



```
= RESTART: C:\User:
or now Sem3\Pract
Start
outlook
Sunny
humidity
High
No
Normal
Yes
Overcast
Yes
Rain
wind
Weak
Yes
Strong
No
```

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

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

Description: The Iris dataset is a popular dataset in machine learning and consists of measurements of sepal length, sepal width, petal length, and petal width for three different species of iris flowers.

Code:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
import numpy as np
import matplotlib.pyplot as plt
irisData = load_iris()
# Create feature and target arrays
X = irisData.data
y = irisData.target
# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size = 0.2, random_state=42)
neighbors = np.arange(1, 9)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))
# Loop over K values
for i, k in enumerate(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    # Compute training and test data accuracy
    train_accuracy[i] = knn.score(X_train, y_train)
    test_accuracy[i] = knn.score(X_test, y_test)
# Generate plot
plt.plot(neighbors, test_accuracy, label = 'Testing dataset Accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training dataset Accuracy')
plt.legend()
plt.xlabel('n_neighbors')
plt.ylabel('Accuracy')
plt.show()
```

Output:

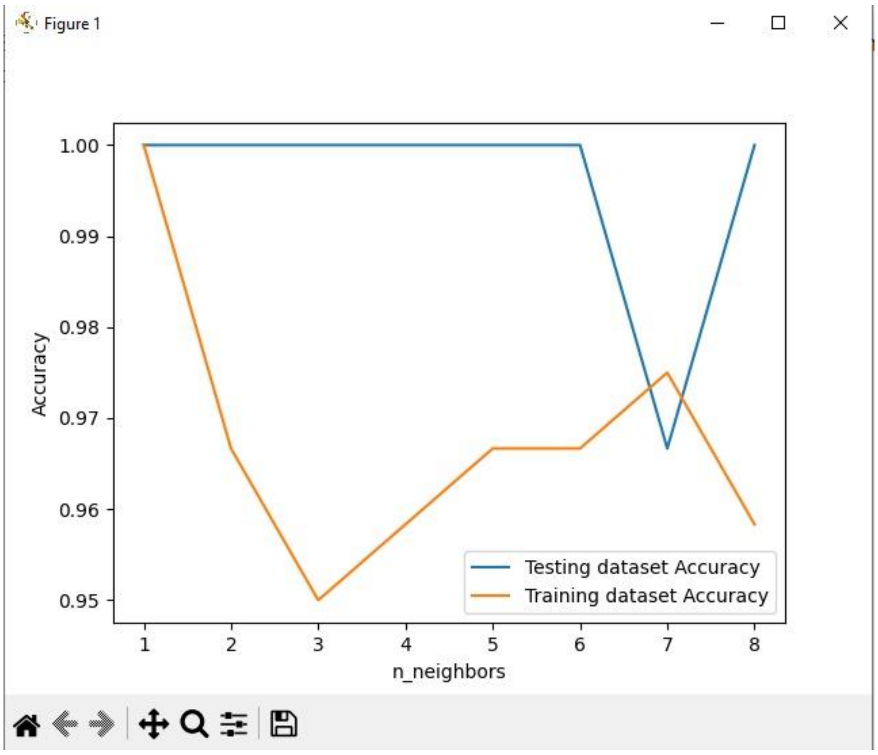


Figure 1
1.00
0.99
0.98
0.97
0.96
0.95
Accuracy
1
2
3
4
5
6
7
8
n_neighbors
Testing dataset Accuracy
Training dataset Accuracy

Practical 6

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

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

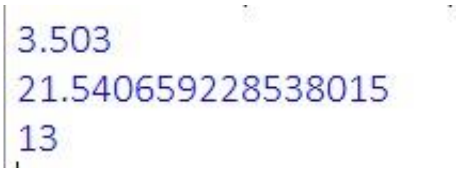
Description: The minkowski_distance function is defined to calculate the Minkowski distance with a specified power parameter (p). The knn_predict function is modified to accept a distance_method parameter, allowing the choice between Euclidean, Manhattan, and Minkowski distances.The program then iterates through each distance method, makes predictions on the testing data, calculates the accuracy, and displays the confusion matrix for each distance method.

Code:

```
# import math library
from math import *
from decimal import Decimal
import numpy as np
# calculating manhattan distance between vectors
from math import sqrt
# Function distance between two points
# and calculate distance value to given
# root value(p is root value)
def p_root(value, root):
        root_value = 1 / float(root)
        return round (Decimal(value) **
                        Decimal(root_value), 3)
def minkowski_distance(x, y, p_value):
        # pass the p_root function to calculate
        # all the value of vector parallelly
        return (p_root(sum(pow(abs(a-b), p_value)
                        for a, b in zip(x, y)), p_value))
# Driver Code
vector1 = [0, 2, 3, 4]
vector2 = [2, 4, 3, 7]
p = 3
```

```
print(minkowski_distance(vector1, vector2, p))
#using Formula
# Import NumPy Library
# initializing points in numpy arrays
P1 = np.array((9, 16, 25))
P2 = np.array((1, 4, 9))
# subtracting both the vectors
temp = P1 - P2
# Using Formula
euclid_dist = np.sqrt(np.dot(temp.T, temp))
# printing Euclidean distance
print(euclid_dist)
def manhattan_distance(a, b):
 return sum(abs(e1-e2) for e1, e2 in zip(a,b))
# define data
row1 = [10, 20, 15, 10, 5]
row2 = [12, 24, 18, 8, 7]
# calculate distance
dist = manhattan_distance(row1, row2)
print(dist)
```

Output:



```
3.503
21.540659228538015
13
```

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

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

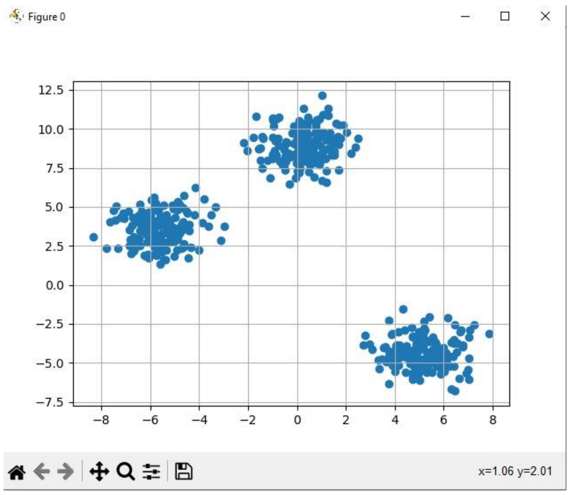
Description: K-means clustering is another simplified algorithms in machine learning. It is categorized into unsupervised learning because here we don’t know the result already. This algorithm is used for vector quantization of the data & has been taken from single processing methodology.

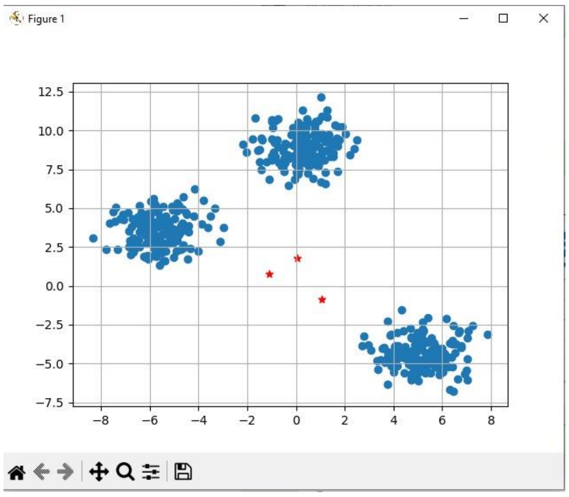
Code:

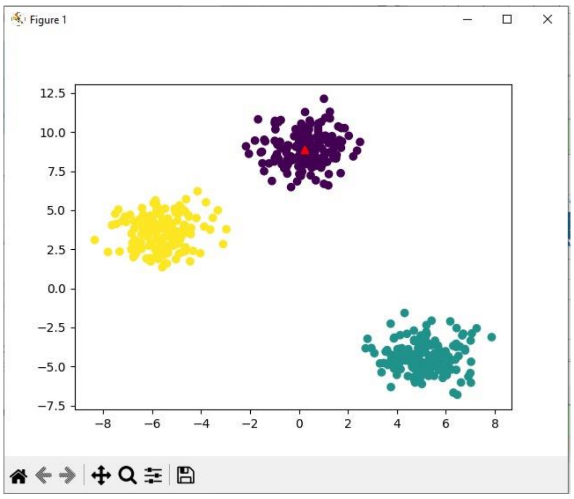
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
X,y = make_blobs(n_samples = 500,n_features = 2,centers = 3,random_state =
23)
fig = plt.figure(0)
plt.grid(True)
plt.scatter(X[:,0],X[:,1])
plt.show()
k = 3
clusters = {}
np.random.seed(23)
for idx in range(k):
        center = 2*(2*np.random.random((X.shape[1],))-1)
        points = []
        cluster = {
                'center' : center,
                'points' : []
        }
        clusters[idx] = cluster
clusters
plt.scatter(X[:,0],X[:,1])
plt.grid(True)
for i in clusters:
        center = clusters[i]['center']
        plt.scatter(center[0],center[1],marker = '*',c = 'red')
plt.show()
def distance(p1,p2):
        return np.sqrt(np.sum((p1-p2)**2))
#Implementing E step
def assign_clusters(X, clusters):
        for idx in range(X.shape[0]):
                dist = []
                curr_x = X[idx]
                for i in range(k):
                        dis = distance(curr_x,clusters[i]['center'])
                        dist.append(dis)
                curr_cluster = np.argmin(dist)
                clusters[curr_cluster]['points'].append(curr_x)
        return clusters
#Implementing the M-Step
def update_clusters(X, clusters):
        for i in range(k):
                points = np.array(clusters[i]['points'])
                if points.shape[0] > 0:
                        new_center = points.mean(axis =0)
                        clusters[i]['center'] = new_center
                        clusters[i]['points'] = []
        return clusters
def pred_cluster(X, clusters):
        pred = []
        for i in range(X.shape[0]):
                dist = []
                for j in range(k):
                        dist.append(distance(X[i],clusters[j]['center']))
                pred.append(np.argmin(dist))
        return pred
clusters = assign_clusters(X,clusters)
clusters = update_clusters(X,clusters)
pred = pred_cluster(X,clusters)
plt.scatter(X[:,0],X[:,1],c = pred)
for i in clusters:
        center = clusters[i]['center']
```

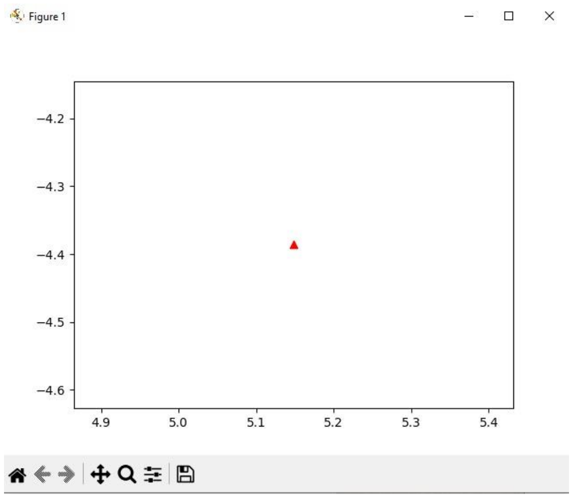
```
plt.scatter(center[0],center[1],marker = '^',c = 'red')
plt.show()
```

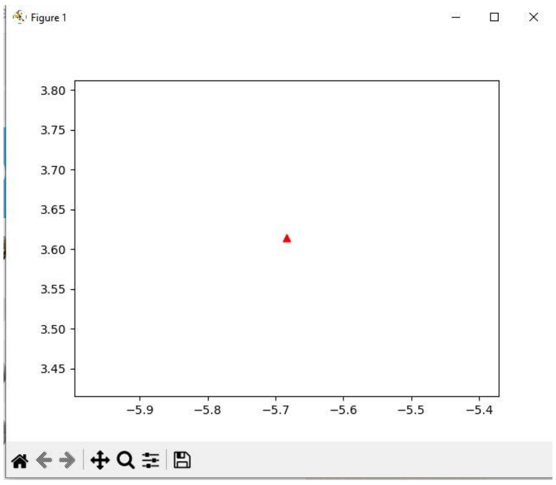
Output:











Practical 7

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

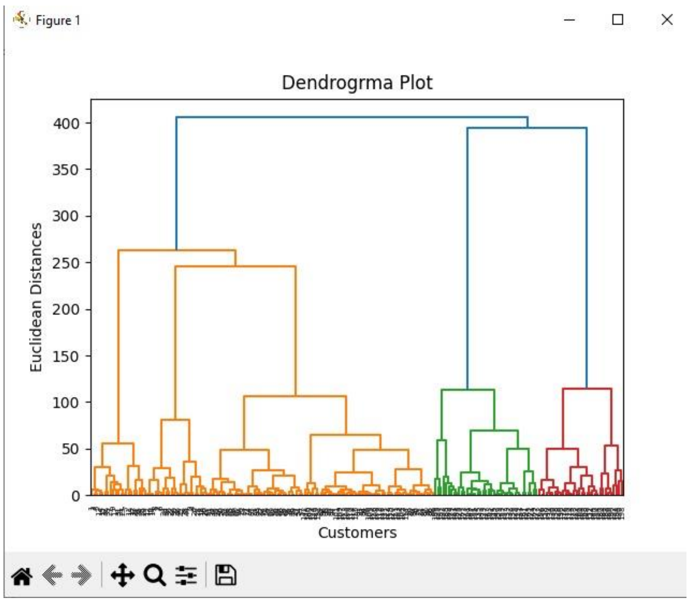
Aim: Implement the classification model using clustering for the following techniques with hierarchical clustering with Prediction, Test Score and Confusion Matrix.

Description: Hierarchical clustering is also known as hierarchical clustering analysis is an algorithm that group similar object into groups called clusters.

Code:

```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
import scipy.cluster.hierarchy as shc
dataset = pd.read_csv('Mall_Customers.csv')
x = dataset.iloc[:, [3, 4]].values
dendro = shc.dendrogram(shc.linkage(x, method="ward"))
mtp.title("Dendrogrma Plot")
mtp.ylabel("Euclidean Distances")
mtp.xlabel("Customers")
mtp.show()
from sklearn.cluster import AgglomerativeClustering
hc= AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
y_pred= hc.fit_predict(x)
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 =
'Cluster2')
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 =
'Cluster5')
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend()
mtp.show()
```

Output:



B. Implement the Rule based method and test the same.

Aim: Implement the Rule based method and test the same.

Description: Rule-based classifier are just another type of classifier which makes the class division depending by using various if-else rules. These rules are easily interpretable & thus these classifiers are generally used to generate descriptive models.

Code:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
# Generate a sample dataset (replace this with your own dataset)
np.random.seed(42)
X = np.random.rand(100, 10)
y = np.random.choice([0, 1], size=100)
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Rule-based classification
def rule_based_classifier(data):
   # Replace this rule with your own logic
   threshold = 0.5
   predictions = (data[:, 0] > threshold).astype(int)
   return predictions
# Apply the rule-based classifier to the training set
y_train_pred = rule_based_classifier(X_train)
# Apply the rule-based classifier to the test set
y_test_pred = rule_based_classifier(X_test)
# Evaluate the model
accuracy_train = accuracy_score(y_train, y_train_pred)
accuracy_test = accuracy_score(y_test, y_test_pred)
conf_matrix_test = confusion_matrix(y_test, y_test_pred)
# Print the results
print("Training Accuracy:", accuracy_train)
print("Test Accuracy:", accuracy_test)
print("Confusion Matrix:")
print(conf_matrix_test)
```

Output:

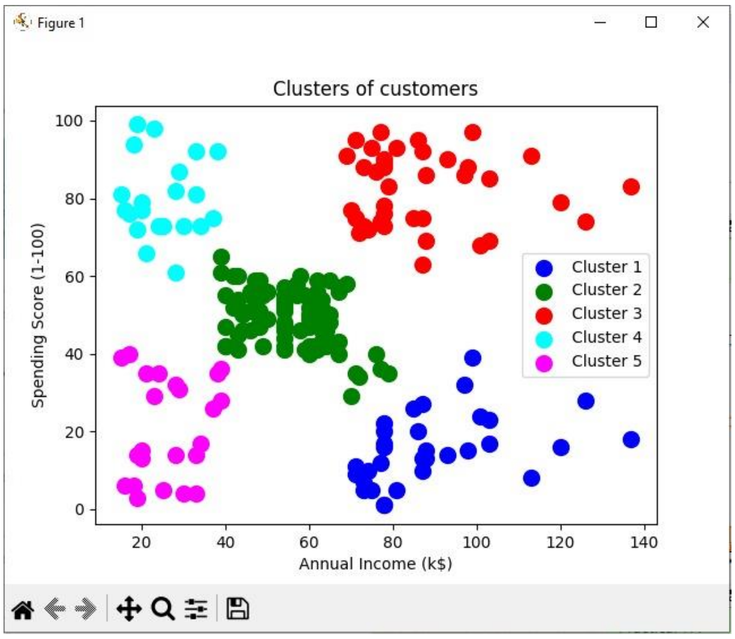


Figure 1
Clusters of customers
Spending Score (1-100)
Annual Income (k$)
0
20
40
60
80
100
20
40
60
80
100
120
140
● Cluster 1
● Cluster 2
● Cluster 3
● Cluster 4
● Cluster 5

Practical 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.

Aim: 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.

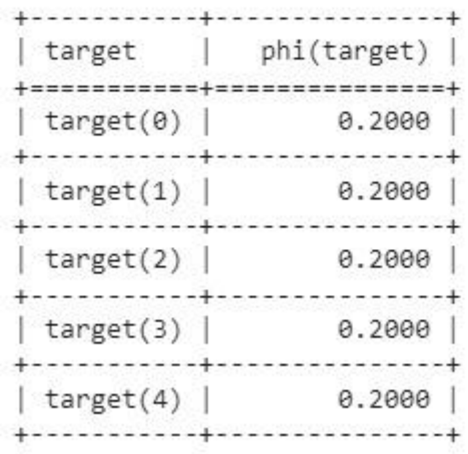
Description: Bayesian network is a graphical representation of different probabilistic relationship among random variable in a particular set.

Code:

```
# from pgmpy.models import BayesianNetwork
# from pgmpy.estimators import MaximumLikelihoodEstimator
# from pgmpy.inference import VariableElimination
# import pandas as pd
# # Load the Heart Disease Data Set
# data =
pd.read_csv('https://github.com/HarshAndroid/network_tools/blob/master/te
st/heart.csv')
# # Considering a subset of relevant columns for constructing the Bayesian
Network
# subset_data = data[['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'target']]
# # Initialize the Bayesian Network model
# model = BayesianNetwork()
# # Define the structure of the Bayesian Network
# model.add_nodes_from(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'target'])
# model.add_edges_from([('age', 'target'), ('sex', 'target'), ('cp', 'target'),
('trestbps', 'target'),
#                     ('chol', 'target'), ('fbs', 'target')])
# # Estimate the parameters of the network using Maximum Likelihood
Estimation
# model.fit(subset_data, estimator=MaximumLikelihoodEstimator)
# # Perform inference using Variable Elimination
# inference = VariableElimination(model)
# # Example query: Given specific conditions, infer the probability of a heart
disease diagnosis
# query_result = inference.query(variables=['target'], evidence={'age': 50, 'sex':
1, 'cp': 2, 'trestbps': 140, 'chol': 260, 'fbs': 1})
```

# print(query_result)

Output:



```
+-----------+---------------+
| target    |   phi(target) |
+===========+===============+
| target(0) |        0.2000 |
+-----------+---------------+
| target(1) |        0.2000 |
+-----------+---------------+
| target(2) |        0.2000 |
+-----------+---------------+
| target(3) |        0.2000 |
+-----------+---------------+
| target(4) |        0.2000 |
+-----------+---------------+
```

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.

Aim: 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: Locally Weighted Regression (LWR) is a non-parametric regression algorithm that aims to fit data points by giving more weight to nearby points and less weight to distant points. The algorithm estimates the value of the target variable based on a weighted combination of the neighboring data points.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
def kernel_function(distance, bandwidth):
   # Gaussian kernel function
   return np.exp(-0.5 * (distance / bandwidth) ** 2)
def locally_weighted_regression(x, X, y, bandwidth):
   m = len(X)
   weights = kernel_function(np.abs(X - x), bandwidth)
   weights_matrix = np.diag(weights)
   X_augmented = np.c_[np.ones(m), X]  # Augmenting X with a column of ones
for bias term
   theta = np.linalg.inv(X_augmented.T @ weights_matrix @ X_augmented) @
(X_augmented.T @ weights_matrix @ y)
   prediction = np.array([1, x]) @ theta
   return prediction
# Generate synthetic dataset
np.random.seed(42)
X = np.linspace(0, 10, 100)
y_true = 2 * X + np.sin(X) + np.random.normal(scale=0.5, size=len(X))
# Apply LWR to predict y for a range of test points
test_points = np.linspace(2, 8, 20)
bandwidth = 0.5
predictions = [locally_weighted_regression(x, X, y_true, bandwidth) for x in
test_points]
# Plot the results
```

```
plt.scatter(X, y_true, label='True data')
plt.plot(test_points, predictions, color='red', label='LWR Predictions')
plt.title('Locally Weighted Regression')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
```

Output:

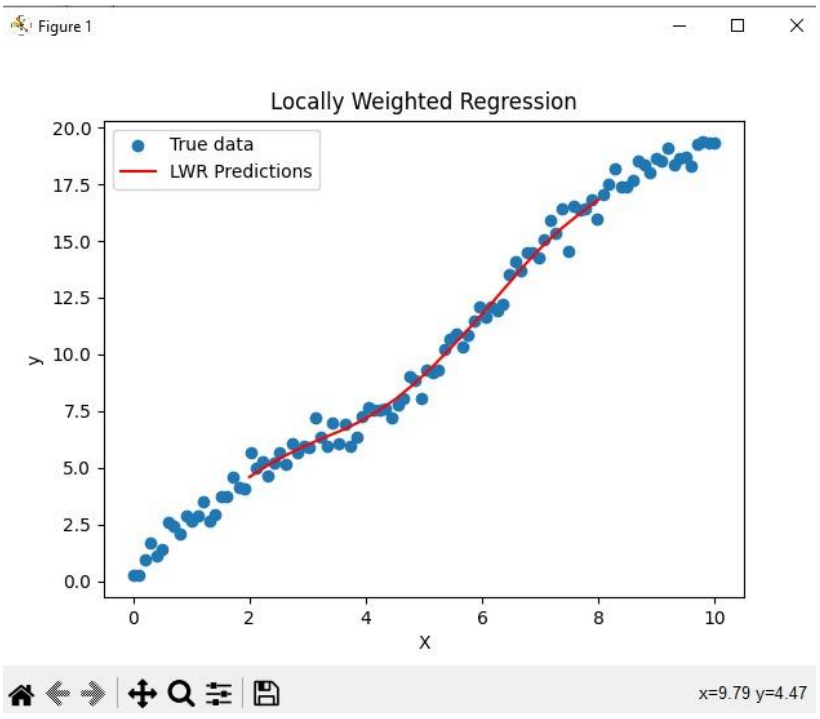


Figure 1
Locally Weighted Regression
True data
LWR Predictions
y
X
x=9.79 y=4.47