TABLE OF CONTENT

Practical 1

Program for Tic- Tac- Toe game using Python Programming

Program:

```
class TicTacToe:


   def___init__(self):
      self.board = []


   def create_board(self):
      for i in range(3):
         row = []
         for j in range(3):
            row.append('-')
         self.board.append(row)


   def get_random_first_player(self):
      return random.randint(0, 1)


   def fix_spot(self, row, col, player):
      self.board[row][col] = player


   def is_player_win(self, player):
      win = None
```

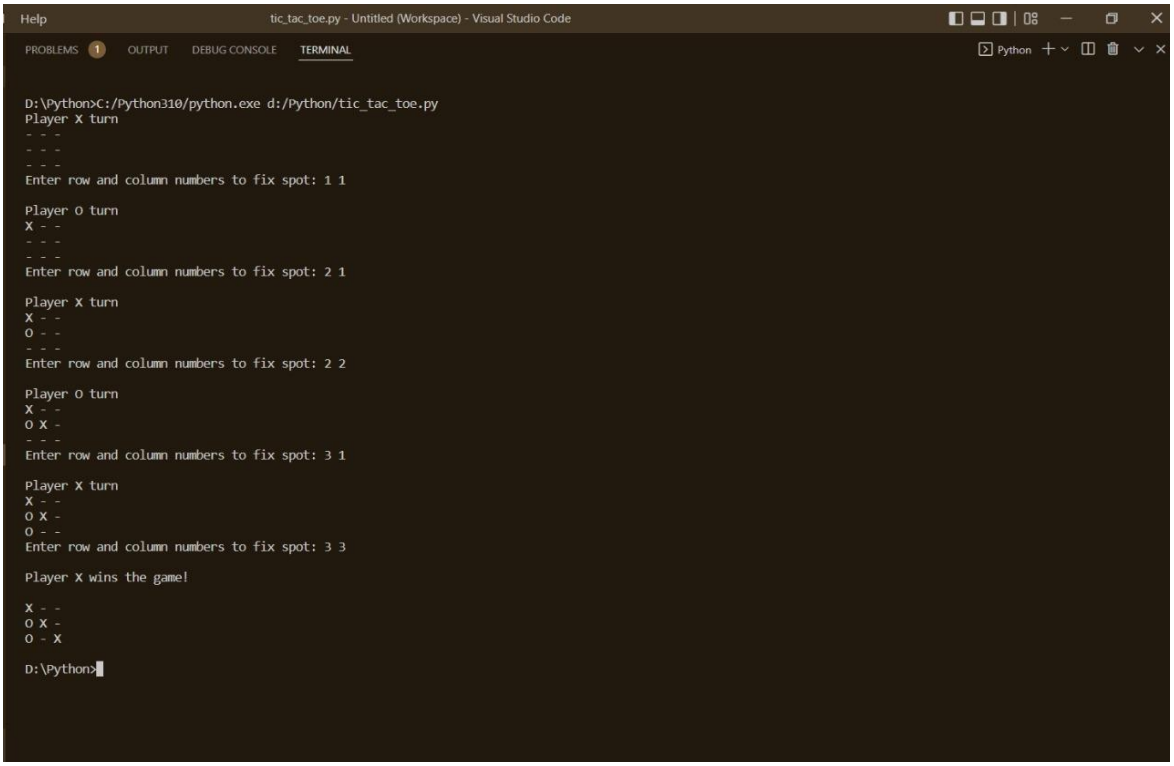
```
n = len(self.board)

# checking rows
for i inrange(n):
    win = True
    for j in range(n):
        if self.board[i][j] != player:
            win = False
            break
    if win:
        return win

# checking columns
for i in range(n):
    win = True
    for j in range(n):
        if self.board[j][i] != player:
            win = False
            break
    if win:
        return win

# checking diagonals
win = True
```

```
    for i in range(n):
        if self.board[i][i] != player:
            win = False
            break
    if win:
        return win

    win = True
    for i in range(n):
        if self.board[i][n - 1 - i] != player:
            win = False
            break
    if win:
        return win
    return False

    for row in self.board:
        for item in row:
            if item == '-':
                return False
    return True

def is_board_filled(self):
    for row in self.board:
        for item in row:
```

```
            if item == '-':
                return False
    return True

def swap_player_turn(self, player):
    return 'X' if player == 'O' else 'O'

def show_board(self):
    for row in self.board:
        for item in row:
            print(item, end=" ")
        print()

def start(self):
    self.create_board()

    player = 'X' if self.get_random_first_player() == 1 else 'O'
    while True:
        print(f"Player {player} turn")

        self.show_board()

        # taking user input
        row, col = list(
            map(int, input("Enter row and column numbers to fix spot: ").split()))
```

```
            print()

            # fixing the spot
            self.fix_spot(row - 1, col - 1, player)

            # checking whether current player is won or not
            if self.is_player_win(player):
                print(f"Player {player} wins the game!")
                break

            # checking whether the game is draw or not
            if self.is_board_filled():
                print("Match Draw!")
                break

            # swapping the turn
            player = self.swap_player_turn(player)

        # showing the final view of board
        print()
        self.show_board()

# starting the game
tic_tac_toe = TicTacToe()
tic_tac_toe.start()
```

Output:


Help
tic_tac_toe.py - Untitled (Workspace) - Visual Studio Code
PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL
Python
D:\Python>C:/Python310/python.exe d:/Python/tic_tac_toe.py
Player X turn
- - -
- - -
- - -
Enter row and column numbers to fix spot: 1 1
Player O turn
X - -
- - -
- - -
Enter row and column numbers to fix spot: 2 1
Player X turn
X - -
O - -
- - -
Enter row and column numbers to fix spot: 2 2
Player O turn
X - -
O X -
- - -
Enter row and column numbers to fix spot: 3 1
Player X turn
X - -
O X -
O - -
Enter row and column numbers to fix spot: 3 3
Player X wins the game!
X - -
O X -
O - X
D:\Python>

Practical 2

Write any program using NumPy using Python

Program:

```
# The standard way to import NumPy:
import numpy as np

# Create a 2-D array, set every second element in
# some rows and find max per row:
x = np.arange(15, dtype=np.int64).reshape(3, 5)
x[1:, ::2] = -99
x
# array([[  0,   1,   2,   3,  4],
#        [-99,   6, -99,   8, -99],
#        [-99, 11, -99, 13, -99]])

x.max(axis=1)

# array([ 4, 8, 13])

# Generate normally distributed random numbers:
rng = np.random.default_rng()
samples = rng.normal(size=2500)
print(samples)
```

Output:

python2.py - D:/MCA/FY/SEM II/AI and ML/python2.py (3.10.2)

File Edit Format Run Options Window Help

```
# The standard way to import NumPy:
import numpy as np

# Create a 2-D array, set every second element in
# some rows and find max per row:

x = np.arange(15, dtype=np.int64).reshape(3, 5)
x[1:, ::2] = -99
x
# array([[  0,   1,   2,   3,   4],
#        [-99,   6, -99,   8, -99],
#        [-99,  11, -99,  13, -99]])

x.max(axis=1)

# array([ 4,  8, 13])

# Generate normally distributed random numbers:
rng = np.random.default_rng()
samples = rng.normal(size=2500)
print(samples)
```

IDLE Shell 3.10.2

File Edit Shell Debug Options Window Help

```
>>> ================ RESTART: D:/MCA/FY/SEM II/AI and ML/python2.py ================
[-1.26693851  0.6465466   0.02690701 ... -0.64648265  0.30617821
  1.74082955]
>>>
>>>
```

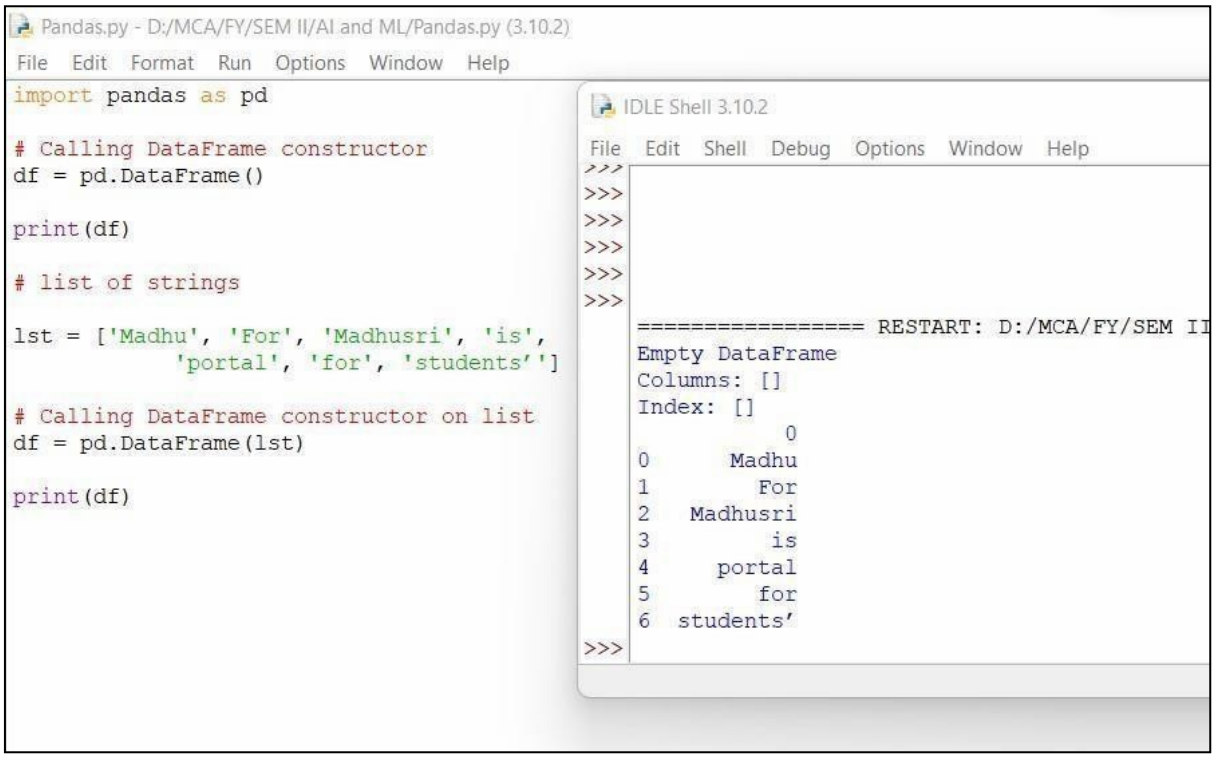
Practical 3

Write Python program using Pandas to illustrateDataframe

Program:

```
import pandas aspd


# Calling DataFrame constructor
df = pd.DataFrame()


print(df)


# list of strings


lst = ['Madhu', 'For', 'Madhusri', 'is',
          'portal', 'for', 'students’']


# Calling DataFrame constructor on list
df = pd.DataFrame(lst)


print(df)
```

Output:


Pandas.py - D:/MCA/FY/SEM II/AI and ML/Pandas.py (3.10.2)
File Edit Format Run Options Window Help
import pandas as pd
# Calling DataFrame constructor
df = pd.DataFrame()
print(df)
# list of strings
lst = ['Madhu', 'For', 'Madhusri', 'is',
'portal', 'for', 'students']
# Calling DataFrame constructor on list
df = pd.DataFrame(lst)
print(df)
IDLE Shell 3.10.2
File Edit Shell Debug Options Window Help
>>>
>>>
>>>
>>>
>>>
========== RESTART: D:/MCA/FY/SEM II
Empty DataFrame
Columns: []
Index: []
0
0 Madhu
1 For
2 Madhusri
3 is
4 portal
5 for
6 students'
>>>

Practical 4

Implementation of K- Means Clustering algorithm inPython

Program:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans


X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)
plt.scatter(X[:,0], X[:,1])
plt.show()


wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10,
random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
```

plt.ylabel('WCSS')

plt.show()

kmeans = KMeans(n_clusters=4, init='k-means++', max_iter=300, n_init=10, random_state=0)

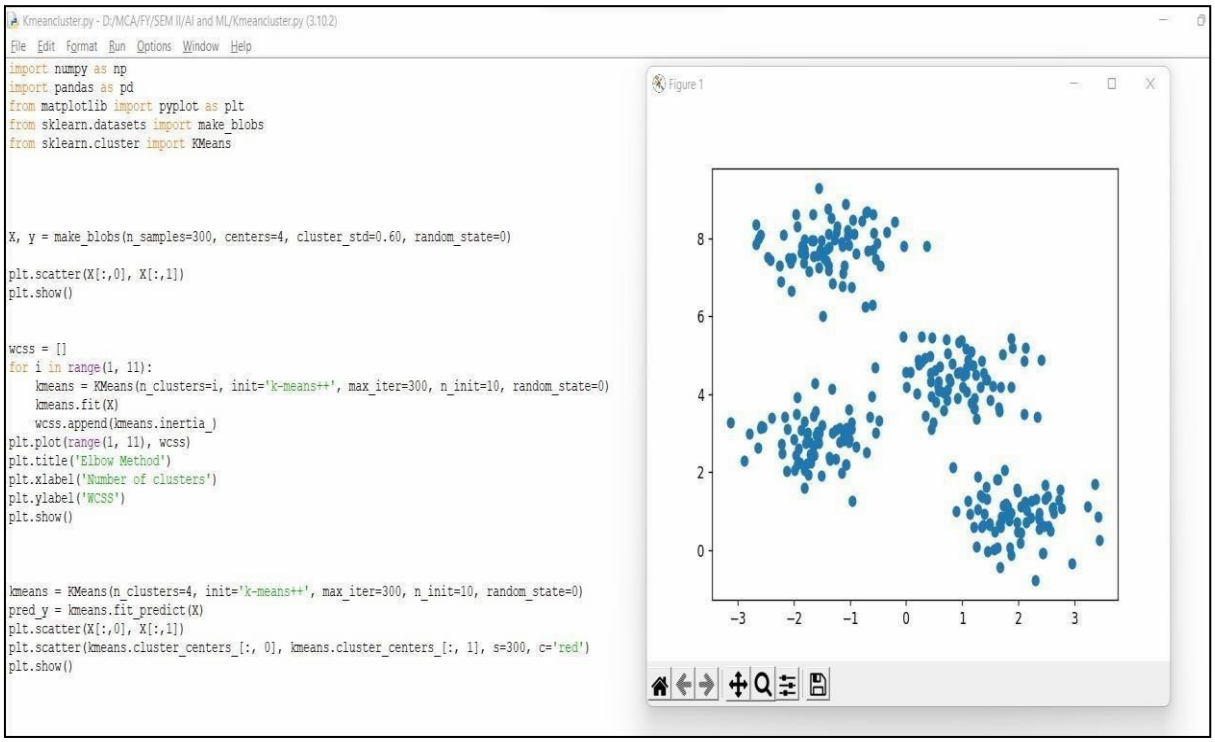
pred_y = kmeans.fit_predict(X)

plt.scatter(X[:,0], X[:,1])

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red')

plt.show()

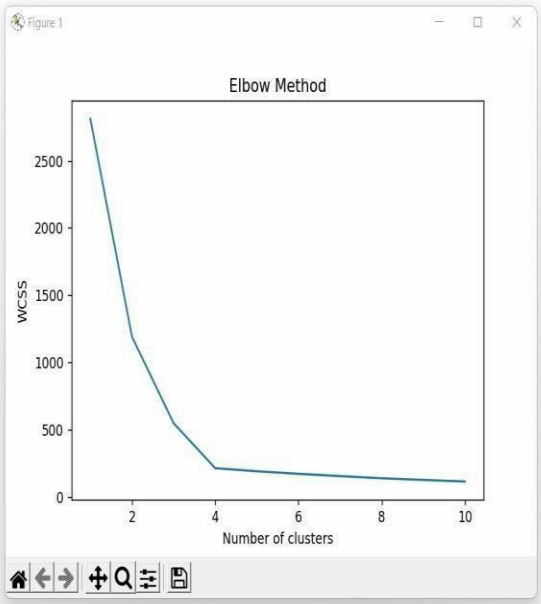
Output:



Kmeancluster.py - D:/MCA/FY/SEM II/AI and ML/Kmeancluster.py (3.10.2)

File Edit Format Run Options Window Help

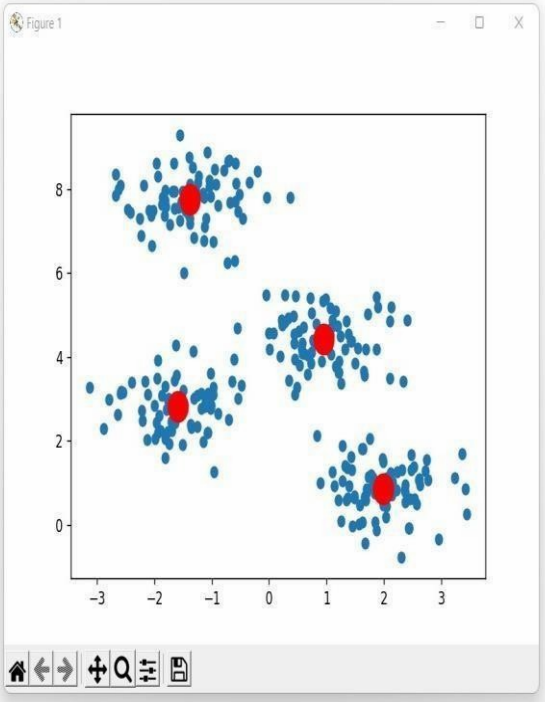
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans



X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

plt.scatter(X[:,0], X[:,1])
plt.show()


wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()



kmeans = KMeans(n_clusters=4, init='k-means++', max_iter=300, n_init=10, random_state=0)
pred_y = kmeans.fit_predict(X)
plt.scatter(X[:,0], X[:,1])
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red')
plt.show()
```



Kmeancluster.py - D:/MCA/FY/SEM II/AI and ML/Kmeancluster.py (3.10.2)

File Edit Format Run Options Window Help

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans



X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

plt.scatter(X[:,0], X[:,1])
plt.show()


wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()



kmeans = KMeans(n_clusters=4, init='k-means++', max_iter=300, n_init=10, random_state=0)
pred_y = kmeans.fit_predict(X)
plt.scatter(X[:,0], X[:,1])
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red')
plt.show()
```



Practical 5

Implementation of Principal Component Analysis (PCA) algorithm in Python

Program:

```
import numpy as nmp
import matplotlib.pyplot asmpltl
import pandas as pnd
from sklearn.model_selection import train_test_split as tts
from sklearn.preprocessing import StandardScaler as SS
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression as LR
from sklearn.metrics import confusion_matrix as CM
from matplotlib.colors import ListedColormap as LCM

DS = pnd.read_csv('Wine.csv')

X = DS.iloc[: , 0:13].values
Y = DS.iloc[: , 13].values

X_train, X_test, Y_train, Y_test = tts(X, Y, test_size = 0.2, random_state = 0)

SC = SS()
```

```
X_train = SC.fit_transform(X_train)
X_test = SC.transform(X_test)

PCa = PCA (n_components = 2)

X_train = PCa.fit_transform(X_train)
X_test = PCa.transform(X_test)

explained_variance = PCa.explained_variance_ratio_

classifier_1 = LR (random_state = 0)
classifier_1.fit(X_train, Y_train)

Y_pred = classifier_1.predict(X_test)

c_m = CM (Y_test, Y_pred)

X_set, Y_set = X_train, Y_train
X_1, X_2 = nmp.meshgrid(nmp.arange(start = X_set[:, 0].min() - 1,
                stop = X_set[: , 0].max() + 1, step = 0.01),
                nmp.arange(start = X_set[: , 1].min() - 1,
                stop = X_set[: , 1].max() + 1, step = 0.01))

mpltl.contourf(X_1, X_2, classifier_1.predict(nmp.array([X_1.ravel(),
          X_2.ravel()]).T).reshape(X_1.shape), alpha = 0.75,
```

```
        cmap = LCM (('yellow', 'grey', 'green')))

mpltl.xlim(X_1.min(), X_1.max())
mpltl.ylim(X_2.min(), X_2.max())

for s, t in enumerate(nmp.unique(Y_set)):
   mpltl.scatter(X_set[Y_set == t, 0], X_set[Y_set == t, 1],
            c = LCM (('red', 'green', 'blue'))(s), label = t)

mpltl.title('Logistic Regression for Training set: ')
mpltl.xlabel ('PC_1')
mpltl.ylabel ('PC_2')
mpltl.legend()
mpltl.show()

  X_set, Y_set = X_test, Y_test

X_1, X_2 = nmp.meshgrid(nmp.arange(start = X_set[: , 0].min() - 1,
                stop = X_set[: , 0].max() + 1, step = 0.01),
                nmp.arange(start = X_set[: , 1].min() - 1,
                stop = X_set[: , 1].max() + 1, step = 0.01))

mpltl.contourf(X_1, X_2, classifier_1.predict(nmp.array([X_1.ravel(),
        X_2.ravel()]).T).reshape(X_1.shape), alpha = 0.75,
        cmap = LCM(('pink', 'grey', 'aquamarine')))
```

```
mpltl.xlim(X_1.min(),X_1.max())
mpltl.ylim(X_2.min(), X_2.max())

for s, t in enumerate(nmp.unique(Y_set)):
    mpltl.scatter(X_set[Y_set == t, 0], X_set[Y_set == t, 1],
                c = LCM(('red', 'green', 'blue'))(s), label = t)

mpltl.title('Logistic Regression for Testing set')
mpltl.xlabel ('PC_1')
mpltl.ylabel ('PC_2')
mpltl.legend()
mpltl.show()

# Note: To download the Wine.csv dataset link -
        https://media.geeksforgeeks.org/wp-
        content/uploads/Wine.csv
```

Output:

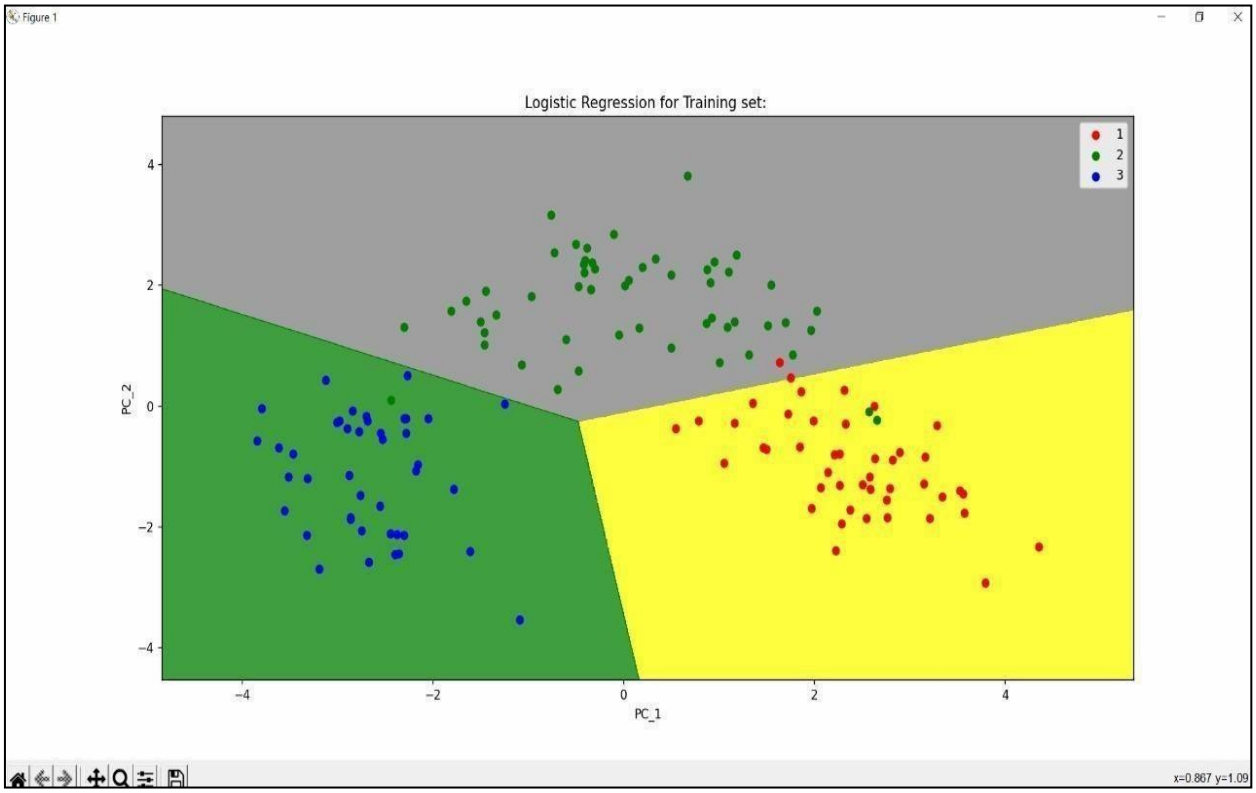


Figure 1
Logistic Regression for Training set:
1
2
3
PC_2
PC_1
x=0.867 y=1.09

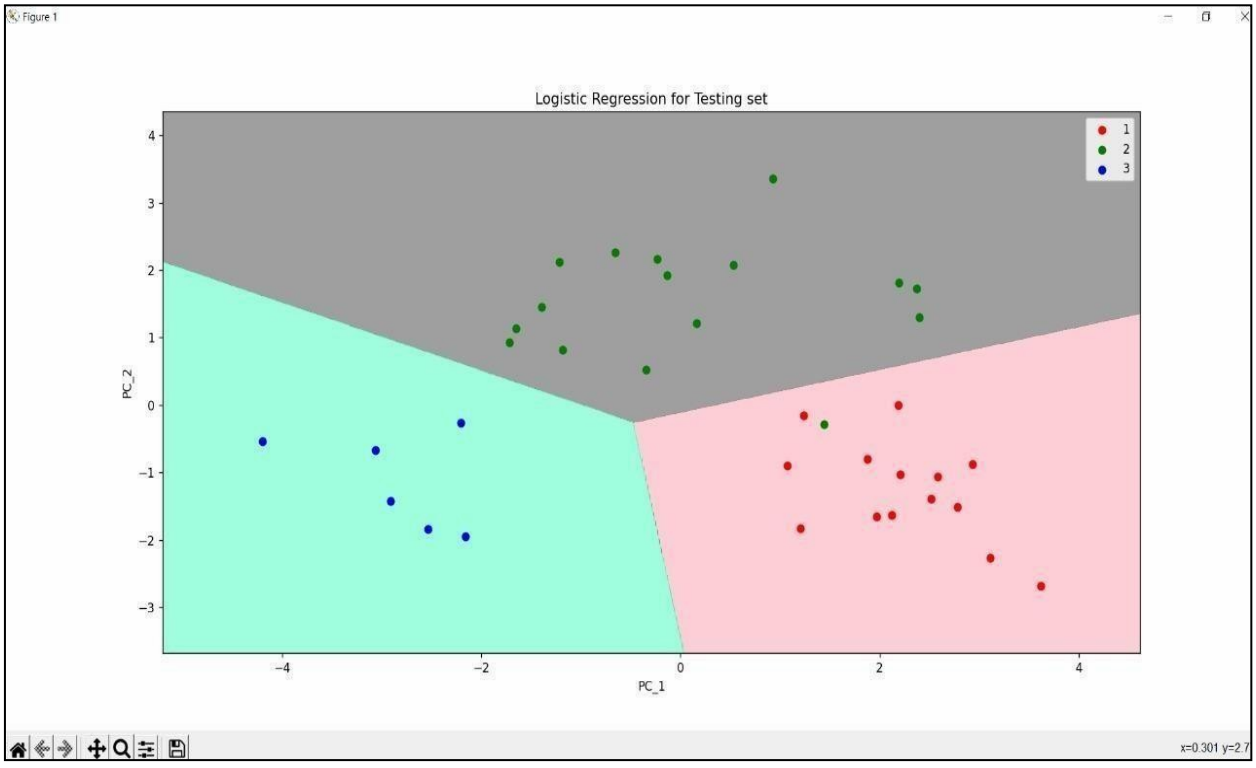


Figure 1
Logistic Regression for Testing set
1
2
3
PC_2
PC_1
x=0.301 y=2.7

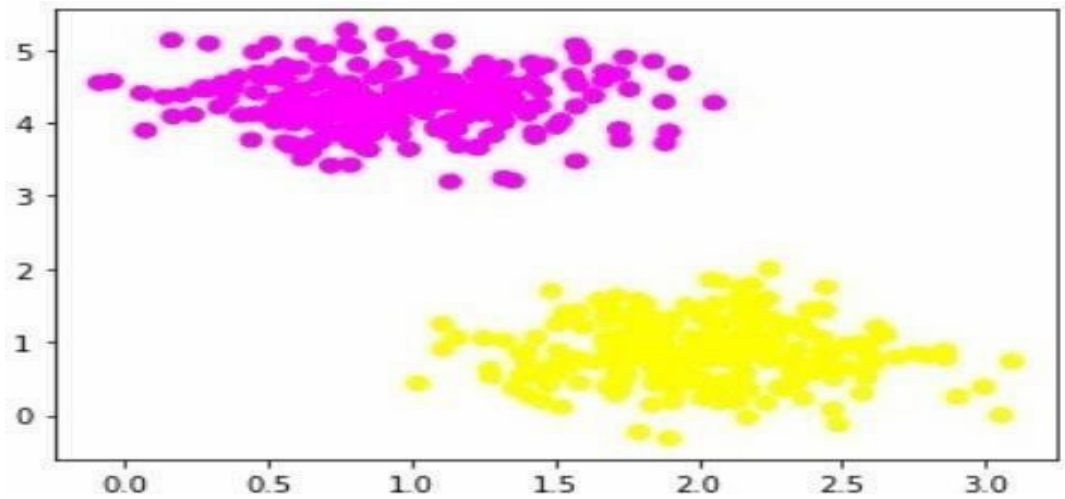
Practical 6

Implement SVM Classifier in Python

Program:

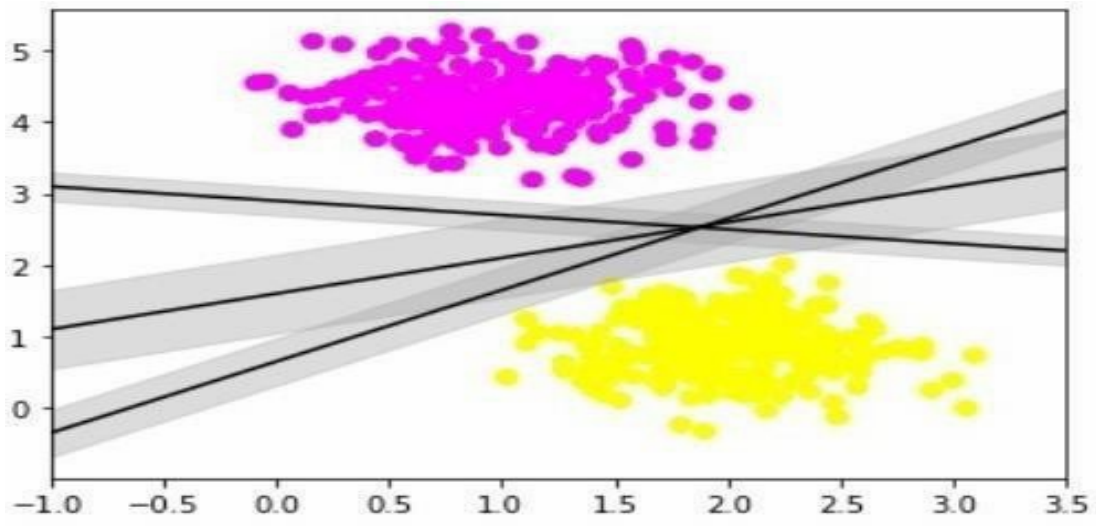
```
# importing scikit learn with make_blobs
from sklearn.datasets import make_blobs

# creating datasets X containing n_samples
# Y containing two classes
X, Y = make_blobs(n_samples=500, centers=2, random_state=0, cluster_std=0.40)
import matplotlib.pyplot as plt
# plotting scatters
plt.scatter(X[:, 0], X[:, 1], c=Y, s=50, cmap='spring');
plt.show()
```

Output:



```
import numpy as np
# creating linspace between -1 to 3.5
xfit = np.linspace(-1, 3.5)
# plotting scatter
plt.scatter(X[:, 0], X[:, 1], c=Y, s=50, cmap='spring')

# plot a line between the different sets of data
for m, b, d in [(1, 0.65, 0.33), (0.5, 1.6, 0.55), (-0.2, 2.9, 0.2)]:
        yfit = m * xfit + b
        plt.plot(xfit, yfit, '-k')
        plt.fill_between(xfit, yfit - d, yfit + d, edgecolor='none',
        color='#AAAAAA', alpha=0.4)

plt.xlim(-1, 3.5);
plt.show()
```

Output:



Practical 7

Program to implement Decision Tree in Python

Program:

```
# Run this program on your local python
# interpreter, provided you haveinstalled
# the required libraries.

# Importing the required packages
import numpy as np
import pandas aspd
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
import warnings
warnings.filterwarnings("ignore")

# Function importing Dataset
def importdata():
        balance_data = pd.read_csv(
'https://archive.ics.uci.edu/ml/machine
-learning-'+ 'databases/balance-scale/balance-
scale.data',
```

```
        sep= ',', header = None)

        # Printing the dataswet shape
        print ("Dataset Length: ", len(balance_data))
        print ("Dataset Shape: ", balance_data.shape)

        # Printing the dataset obseravtions
        print ("Dataset: ",balance_data.head())
        return balance_data

# Function to split the dataset
def splitdataset(balance_data):

        # Separating the target variable
        X = balance_data.values[:, 1:5]
        Y = balance_data.values[:, 0]

        # Splitting the dataset into train and test
        X_train, X_test, y_train, y_test = train_test_split(
        X, Y, test_size = 0.3, random_state = 100)

        return X, Y, X_train, X_test, y_train, y_test

# Function to perform training with giniIndex.
def train_using_gini(X_train, X_test, y_train):
```

```
    # Creating the classifier object
    clf_gini = DecisionTreeClassifier(criterion = "gini",
                random_state = 100,max_depth=3, min_samples_leaf=5)

    # Performing training
    clf_gini.fit(X_train, y_train)
    return clf_gini

# Function to perform training with entropy.
def tarin_using_entropy(X_train, X_test, y_train):

    # Decision tree with entropy
    clf_entropy = DecisionTreeClassifier(
                criterion = "entropy", random_state = 100,
                max_depth = 3, min_samples_leaf = 5)

    # Performing training
    clf_entropy.fit(X_train, y_train)
    return clf_entropy

# Function to make predictions
def prediction(X_test, clf_object):

    # Predicton on test with giniIndex
```

```
    y_pred = clf_object.predict(X_test)
    print("Predicted values:")
    print(y_pred)
    return y_pred

# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):

    print("Confusion Matrix: ",
        confusion_matrix(y_test, y_pred))

    print ("Accuracy : ",
    accuracy_score(y_test,y_pred)*1
    00)

    print("Report : ",
    classification_report(y_test, y_pred))

# Driver code
def main():

    # Building Phase
    data = importdata()
    X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
    clf_gini = train_using_gini(X_train, X_test, y_train)
    clf_entropy = tarin_using_entropy(X_train, X_test, y_train)
```

```
    # Operational Phase
    print("Results Using Gini Index:")

    # Prediction using gini
    y_pred_gini = prediction(X_test, clf_gini)
    cal_accuracy(y_test, y_pred_gini)

    print("Results Using Entropy:")
    # Prediction using entropy
    y_pred_entropy = prediction(X_test, clf_entropy)
    cal_accuracy(y_test, y_pred_entropy)


# Calling main function
if__name__=="__main__":
    main()
```

Output:

```
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.1929 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: D:/MCA/FY/SEM II/AI and ML/decisiontree.py ==========
Dataset Length:  625
Dataset Shape:  (625, 5)
Dataset:     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 Matrix:  [[ 0  6  7]
 [ 0 67 18]
 [ 0 19 71]]
Accuracy :  73.40425531914893
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:
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' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' '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 Matrix:  [[ 0  6  7]
 [ 0 63 22]
 [ 0 20 70]]
Accuracy :  70.74468085106383
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

>>>
```

Ln: 12 Col: 1

Practical 8

To study Bagging Algorithm

Program:

```
# importing utility modules
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# importing machine learning models for prediction
import xgboost as xgb

# importing bagging module
from sklearn.ensemble import BaggingRegressor

# loading train data set in dataframe from train_data.csv file
df = pd.read_csv("train_data.csv")

# getting target data from the dataframe
target = df["target"]

# getting train data from the dataframe
train = df.drop("target")

# Splitting between train data into training and validation dataset
X_train, X_test, y_train, y_test = train_test_split(
```

train, target, test_size=0.20)

initializing the bagging model using XGboost as base model with default parameters

model = BaggingRegressor(base_estimator=xgb.XGBRegressor())

training model

model.fit(X_train, y_train)

predicting the output on the test dataset

pred = model.predict(X_test)

printing the root mean squared error between real value and predicted value

print(mean_squared_error(y_test, pred_final))

Output: