NAME: JANANI J
ROLL.NO: 220701097

SUBJECT : POAI-Observation

DEPT. & SEC: CSE - 'B'

S.No	DATE	NAME OF THE EXPERIMENT	SIGNATURE
1.		N-Queens problem	
2.		Depth first Search	
3.		Water Jug using DFS	
4.		A* Algorithm.	
5.		Minimax Algorithm.	
6.		Prolog	
7.		Artificial Neural Networks	
8.		Artificial Neural Networks - organisation	
9.		Decision Tree classification.	
10.		Introduction to Prolog.	
11.		Unification and Resolution	
12.		Fuzzy Logic - Image processing	
13.		Importance of clustering - kMeans	
14.		Importance of Decision Tree classification.	

EXP. NO: 01

DATE :

N-QUEENS PROBLEM.

AIM:

To solve the N-Queen Problem where the goal is to place n-queens on a n x n chessboard such that no two queens attack each other.

ALGORITHM:

step 1: Start

step 2: Create a n x n chessboard with all cells set to 0, representing no queens placed.

step 3: Ensure no queen is in the same row, upper diagonal, or lower diagonal for a given position.

step 4: Try placing a queen in each row of the current column if it is safe using isSafe().

step 5: Move to the next column if placing a queen works, else backtrack by removing queen.

step 6: If queen are placed in all columns return sucess.

step 7: Display the board

step 8: If no solution exists, print "Solution does not exist."

PROGRAM:

```
def isSafe(board, row, col, n):
    for i in range(col):
        if board[row][i] == 1:
            return false;
    for i, j in zip(range(row, -1, -1), range
                    (col, -1, -1)):
        if board[i][j] == 1:
            return false;
    for i, j in zip(range(row, 1, -1), range(col, -1, -1)
        if board[i][j] == 1:
            return false
    return true
def solveNQUtil(board, col, n):
    if col >= n:
        return true
    for i in range(n):
        if isSafe(board, i, col, n):
            board[i][col] = 1
        if solveNQUtil(board, col+1, n) == true:
            return True
        board[i][col] = 0
    return False.
def solveNQ(n):
    board = [[0]*n for _ in range(n)]
    if solveNQUtil(board, 0, n) == False:
        print("Solution doest not exist.")
        return
    return False
```

```
for i in board:
    print(i)
return True
n = int(input("enter n value:"))
solve_NQ(n)
```

OUTPUT:

```
Enter n value : 5
[1,0,0,0,0]
[0,0,0,1,0]
[0,1,0,0,0]
[0,0,0,0,1]
[0,0,1,0,0]
```

RESULT:

Thus the program is successfully executed and output is verified.

EXP.NO: 02 DEPTH FIRST SEARCH

DATE:

AIM:

To implement Depth First Search (DFS) to traverse a graph and explore all vertices by visiting as far along each branch as possible before backtracking.

ALGORITHM:

step 1: Start

step 2: Initialize an empty stack and a list to keep track of visited nodes.

step 3: Push the starting node onto stack & mark visite

step 4: While the stack is not empty, repeat step 5 to step 7.

step 5: Pop the top node from the stack.

step 6: Print or process the popped node.

step 7: For each adjacent unvisited neighbour of the popped node.

step 8: Mark the neighbour as visited.

step 9: Push the unvisited neighbour onto the stack.

step 10: Repeat until all reachable node are visited.

step 11: Stop.

PROGRAM:

```
def dfs(graph, start):
    stack = [start]
    visited = set()

    while stack:
        node = stack.pop()
        if node not in visited:
            print(node, end = " ")
            visited.add(node)

            for neighbor in graph[node]:
                if neighbor not in visited:
                    stack.append(neighbor)

graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}
Print("DFS Traversal starting from node 'A':")
dfs(graph, 'A')
```

OUTPUT:

DFS Traversal starting from node 'A':

ACFBED

RESULT:

Thus the program is successfully executed and output is verified.

EXP.NO: 03

DATE:

WATER JUG PROBLEM: USING DEPTH FIRST SEARCH

AIM:

To determine if a specific amount of water can be measured using two jugs of different capacities via Depth-First Search (DFS).

ALGORITHM:

Step 1: Start with two empty jugs and a target volume.

Step 2: Represent each state as a tuple (amount in jug1, amount in jug2)

Step 3: Implement DFS to explore all states

Step 4: Use a set to track visited states.

Step 5: Possible Generate new states by filling, emptying or transferring water between the jugs.

Step 6: Return True if the target volume is reached.

Step 7: Apply DFS recursively to each new state.

Step 8: Explore all ~~possible~~ states by backtracking.

Step 9: Stop when all states are explored or the target is found.

Step 10: Output whether the target is achievable.

PROGRAM:

```
def water_jug_dfs(jug1_capacity, jug2_capacity, target):
    def dfs(state):
        if state in visited:
            return false
        visited.add(state)
        jug1, jug2 = state
        if jug1 == target or jug2 == target:
            return True

        actions = [(jug1_capacity, jug2),
                   (jug1, jug2_capacity),
                   (0, jug2),
                   (jug1, 0),
                   (min(jug1_capacity, jug1 + jug2),
                   jug2 - (min(jug1_capacity, jug1 + jug2)
                                -jug1)),
                   (jug1-(min(jug2_capacity, jug1 + jug2)-
                   jug2), min(jug2_capacity, jug1 + jug2))]
        return any(dfs(new_state) for new_state in
                                    actions)
    visited = set()
    return dfs((0,0))

jug1_capacity = 4
jug2_capacity = 3
target = 2
Print(water_jug_dfs(jug1_capacity, jug2_capacity, target))
```

OUTPUT:

True

RESULT:

Thus the program is successfully executed and output is verified.

EXP.NO:4

DATE:

A* SEARCH

AIM:

To find the shortest path from a start node to a goal node using A* search algorithm.

ALGORITHM:

step 1: Create open and closed sets; start with the initial node.

step 2: Add the start node to the open set with an initial cost of 0.

step 3: Remove the node with the lowest 'f' value (cost + heuristic) from the open set.

step 4: If the current node is the goal node, reconstruct the path.

step 5: For each neighbor, calculate 'g', 'h', 'f' values.

step 6: If the neighbour is not in the open set or a lower cost path is found, update costs and parent.

step 7: Add the neighbour to the open set if it is not already in the closed set.

step 8: Repeat until the open set is empty or the goal is found.

PROGRAM:

```
import heapq
def a_star(start, goal, h, neighbors):
    open_set = []
    heapq.heappush(open_set, (0+h(start), 0,
                              start))
    came_from = {}
    g_score = {start: 0}
    f_score = {start: h(start)}

    while open_set:
        _, current_g, current = heapq.heappop(open_set)
        if current == goal:
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(start)
            return path[::-1]
        for neighbor in neighbors(current):
            tentative_g = g_score[current] + 1
        if neighbor not in g_score or tentative_g <
                                    g_score[neighbor]:
            came_from[neighbor] = current
            g_score[neighbor] = tentative_g
            f_score[neighbor] = tentative_g + h(neighbor)
            if neighbor not in [i[2] for i in open_set]:
                heapq.heappush(open_set,
                    (f_score[neighbor], tentative_g, neighbor))
    return None
```

```
def heuristic(node):
    goal_position = (5,5)
    return abs(node[0] - goal_position[0]) +
           abs(node[1] - goal_position[1])

def neighbors(node):
    x, y = node
    return [(x+1, y), (x-1, y), (x, y+1), (x, y-1)]

start = (0,0)
goal = (5,5)
path = a_star(start, goal, heuristic, neighbors)
print(path)
```

OUTPUT:

```
[(0,0), (1,0), (2,0), (3,0), (4,0), (5,0), (5,1),
(5,2), (5,3), (5,4), (5,5)]
```

RESULT:

Thus the program is successfully executed and output is verified.

5. MIN MAX
6. PROLOG
7. FUZZY LOGIC

EX.NO: 05

DATE :

MINIMAX ALGORITHM

AIM:

To implement MINIMAX Algorithm problem using Python.

SOURCE CODE:

```
from math import inf as infinity
from random import choice
import platform
import time
from os import system
HUMAN = -1
COMP = +1
board = [ [0,0,0],
          [0,0,0],
          [0,0,0] ]
def evaluate(state):
    if wins(state,COMP):
        score = +1
    elif wins(state, HUMAN):
        score = -1
    else
        score = 0
return score

def wins(state,player):
    win_state = [ [state[0][0], state[0][1], state[0][2]],
                  [state[1][0], state[1][1], state[1][2]],
                  [state[2][0], state[2][1], state[2][2]],
                  [state[0][0], state[1][0], state[2][0]],
                  [state[0][1], state[1][1], state[2][1]],
                  [state[0][2], state[1][2], state[2][2]],
                  [state[0][0], state[1][1], state[2][2]],
                  [state[2][0], state[1][1], state[2][2]],
                ]
```

```
    if [player, player, player] in win_state:
        return True
    else:
        return False

def game_over(state):
    return wins(state, HUMAN) or wins(state, COMP

def empty_cells(state):
    cells = []
    for x, row in enumerate(state):
        for y, cell in enumerate(row):
            if cell == 0:
                cells.append([x, y])
    return cells

def valid_move(x, y):
    if [x, y] in empty_cells(board):
        return True
    else
        return False

def set_value(x, y, player):
    if valid_move(x, y):
        board[x][y] = player
        return True,
    else
        return False.

    return best
def clean():

def render(state, c_choice, h_choice):
    chars = {
        -1: h_choice,
        +1: c_choice,
        0: ' '
    }
```

```
str_line = '-'*9
print('\n' + str_line)
for row in state:
    for cell in row:
        symbol = chars[cell]
        print(f'| {symbol} |', end='')
    print('\n' + str_line)

def ai_turn(c_choice, h_choice):
    depth = len(empty_cells(board))
    if depth == 0 or game_over(board):
        return

    clean()
    print(f'Computer turn [{c_choice}]')
    render(board, c_choice, h_choice)

    if depth == 9:
        x = choice([0,1,2])
        y = choice([0,1,2])
    else:
        move = minimax(board, depth, COMP)
        x, y = move[0], move[1]

    set_move(x, y, COMP)
    time.sleep(1)

def main():
    clean()
    h_choice = '' # X or O
    c_choice = '' # X or O
    first = '' # if human is the first

    # Human chooses X or O to play
    while h_choice != 'O' and h_choice != 'X':
        try:
            print('')
            h_choice = input('Choose X or O\n').upper()
```

```
except (EOFError, KeyboardInterrupt):

if wins(board, HUMAN):
    clear()
    print('Human turn [{h_choice}]')
    render(board, c_choice, h_choice)
    print('YOU WIN!')
elif wins(board, COMP):
    clear()
    print('Computer turn [{c_choice}]')
    render(board, c_choice, h_choice)
    print('YOU LOSE!')
else:
    clear()
    render(board, c_choice, h_choice)
    print('DRAW')
    exit()
if __name__ == '__main__':
    main()
```

OUTPUT:

```
Choose X or O
Chosen: X
First to start [y/n]: y
Human turn [X]
 .  .  .
 |  |  |
 .  .  .
 |  |  |
 .  .  .

Computer turn [O]
 .  .  .
 |  |  |
 .  .  .
 |  |  |
 .  .  .
```

Human turn [x]

```
|  O  ||     ||     |
|  X  ||     ||     |
|     ||     ||     |
```

Computer turn [O]

```
|  O  ||  X  ||     |
|  X  ||     ||     |
|     ||     ||     |
```

Human turn

```
|  O  ||  X  ||  X  |
|  X  ||  O  ||     |
|     ||     ||     |
```

Computer turn [O]

```
|  O  ||  X  ||  X  |
|  X  ||  O  ||     |
|     ||     ||  O  |
```

YOU LOSE!

RESULT:

Thus the program for MINIMAX Algorithm is successfully executed and output is verified.

EXP.NO : 06 PROLOG - FAMILY TREE

DATE :

AIM:

To develop a family tree program using PROLOG with all possible facts, rules and queries.

PROGRAM:

```
/* FACTS : : */
male (peter)
male (john)
male (chris)
male (kevin)

female (betty)
female (jeny)
female (lisa)
female (helen).

parentOf (chris, peter).
parentOf (chris, betty)
parentOf (helen, peter)
parentOf (helen, betty)
parentOf (kevin, chris)
parentOf (jeny, john)
parentOf (jeny, helen)

/* RULES : : */
father (x,y) :- male(y),
parentOf (x,y)
```

```
mother(x,y):- female(y),
parentOf(x,y)
grandfather(x,y):- male(y),
parentOf(x,z),
parentOf(z,y).

sister(x,y):- female(y),
father(x,z),
father(y,w),
z==w.
```

OUTPUT:

```
male(peter)
true
father(chris,peter
true
father(chris,betty
false
grandfather(jeny,
false
brother(chris,helen)
false
```

```
?- father('chris', 'peter').
true.
?- mother('helen', 'peter').
false.
?- grandfather('kevin', 'peter').
true.
?- grandmother('jeny', 'helen').
false.
?- brother('peter', 'betty').
false.
?- sister('betty', 'peter').
false.
```

father(x, y)

x = chris, y = peter, x = helen, y = peter, x = kevin, y = chris

mother(x,y)

x = chris, y = betty, x = helen, y = betty, x = jerry, y = helen.

grandmother(x,y)

x = kevin, y = betty, x = jeny, y = betty.

grandfather(x,y)

x = kevin, y = peter, x = jeny, y = peter.

RESULT:

Thus the prolog program for prolog is successfully executed and output is verified.

EXP. NO: 07

DATE : INTRODUCTION TO PROLOG

AIM:

To learn basic PROLOG terminologies and write basic programs.

PROGRAM:

KB1:

```
woman (mia)
woman (jody)
woman (yolanda)
playsAirGuitar (jody)
party.
Query 1: ?- woman (mia).
Query 2: ?- playsAirGuitar (mia).
Query 3: ?- party
Query 4: ?- concert
```

OUTPUT:

```
?- woman (mia).
true
?- playsAirGuitar (mia)
false
?- party
true.
?- concert
ERROR: Unknown procedure:
```

KB2:

```
happy (yolanda).
listens2music (mia)
Listens2music (yolanda) :- happy (yolanda)
playsAirGuitar (mia) :- listens2 music (mia).
playsAirGuitar (yolanda) :- listens2 music (yolan
```

OUTPUT:

?- playsAirGuitar(mia)

true

?- playsAirGuitar(yolanda)

true

?-

KB3:

likes(dan, sally)

likes(sally, dan)

likes(john, britney)

married(X,Y) :- likes(X,Y), likes(Y,X).

friends(X,Y) :- likes(X,Y); likes(Y,X).

OUTPUT:

?- likes(dan, X)

X = sally

?- married(dan, sally)

true

?- married(john, britney)

false.

KB4:

food(burger)

food(sandwich)

food(pizza)

lunch(sandwich)

dinner(pizza)

meal(X) :- food(X)

OUTPUT:

?- food(pizza)

true

?- meal(X), lunch(X)

X = sandwich.

?- dinner(sandwich)

false

?-

KB5:

owns(jack, car(bmw)).
owns(john, car(chevy)).
owns(olivia, car(civic))
owns(jane, car(chevy))
sedan(car(bmw))
sedan(car(civic))
truck(car(chevy))

OUTPUT:

?-
owns(john, X)
X = car(chevy)
?- owns(john, _)
true
?- owns(who, car(chevy))
who = john

```
?- woman(mia).
true.
?- playsAirGuitar(mia).
false.
?- party.
true.
?- concert.
ERROR: Unknown procedure: concert
KB2 Plays Air Guitar: {'mia', 'yolanda'}
KB3 Married (Dan, Sally): True
KB3 Friends (John, Brittney): True
KB4 Is Burger Food?: True
KB4 Is Sandwich Food?: True
KB4 Is Pizza Food?: True
KB5 Jack owns a sedan: True
KB5 John owns a truck: True
KB5 Olivia owns a sedan: True
```

RESULT:

Thus the program for Introduction to Prolog is successfully executed and output is verified.

EX.NO : 08
DATE :

UNIFILATION AND RESOLUTION

AIM:

To execute programs based on Unification and Resolution. Deduction in prolog is based on the Unification and Instantiation. Matching terms are unified and variables get instantiated.

Example 1: In the below prolog program, unification and instantiation take place after querying.

Facts:

likes(john, jane)
likes(jane, john)

Query:

?- likes(john, x).

Answer: x = jane .

Example 2: At the prolog query prompt, when you write below query.

?- owns(x, car(bmw)) = owns(y, car(c))

You will get Answer : x = y, c = bmw.

SOURCE CODE:

```
enjoy :- sunny, warm.
strawberry_picking :- warm, pleasant.
notstrawberry_picking :- raining.
wet :- raining
warm.
raining
sunny.
```

OUTPUT:

```
?- enjoy.
true.
?- strawberry_picking.
false.
?- notstrawberry_picking.
true.
?- wet.
true.
```

RESULT:

Thus, the program for Unification and Resolution is successfully executed and output is verified.

EX.NO: 09

DATE:

FUZZY LOGIC - IMAGE PROCESSING

OVERVIEW:

An edge is a boundary between two uniform regions. You can detect an edge by comparing the intensity of neighbouring pixels. However, because uniform regions are not visply defined, small intensity differences between two neighbouring pixels do not always represent an edge.

The fuzzy logic approach for image processing allows you to use membership functions to define the degree to which a pixel belongs to an edge or a uniform region.

IMPLEMENTATION:

=> Import RGB Image and Convert to Grayscale -

Import the Image.

```
Irgb = imread('peppers.png');
```

=> Convert Irgb to grayscale.

```
Igray = rgb2gray(Irgb);
figure
imagesc(Igray, 'CDataMapping', 'scaled')
colormap('gray')
title('Input Image in Grayscale')
I = im2double(Igray)
```

=> Obtain Image Gradient

```
Gx = [-1, 1];
Gy = Gx';
Ix = conv2(I, Gx, 'same');
Iy = conv2(I, Gy, 'same');
```

=> Plot the image Gradients

```
figure
image (Ix, 'CDataMapping', 'scaled')
colormap ('graymap')
title ('Ix').
```

=> Define Fuzzy Interference System (FIS) for Edge Detection.

```
edgeFIS = mamfis ('Name', 'edgeDetection');
edgeFIS = addInput [edgeFIS, [-1,1], 'Name', 'Ix');
edgeFIS = addInput [edgeFIS, [-1,1], 'Name', 'Iy');
```

=> Specify the triangular membership functions, white and black, for Iout.

```
wa = 0.1;
wb = 1;
wc = 1;
ba = 0;
bb = 0;
bc = 0.7;
edgeFIS = addMF [edgeFIS, 'Iout', 'trimf', [wa wb wc],
                  'Name', 'white');
```

=> Evaluate FIS

Evaluate the output of the edge detector for each row of pixels in I using corresponding rows of Ix and Iy as inputs.

level

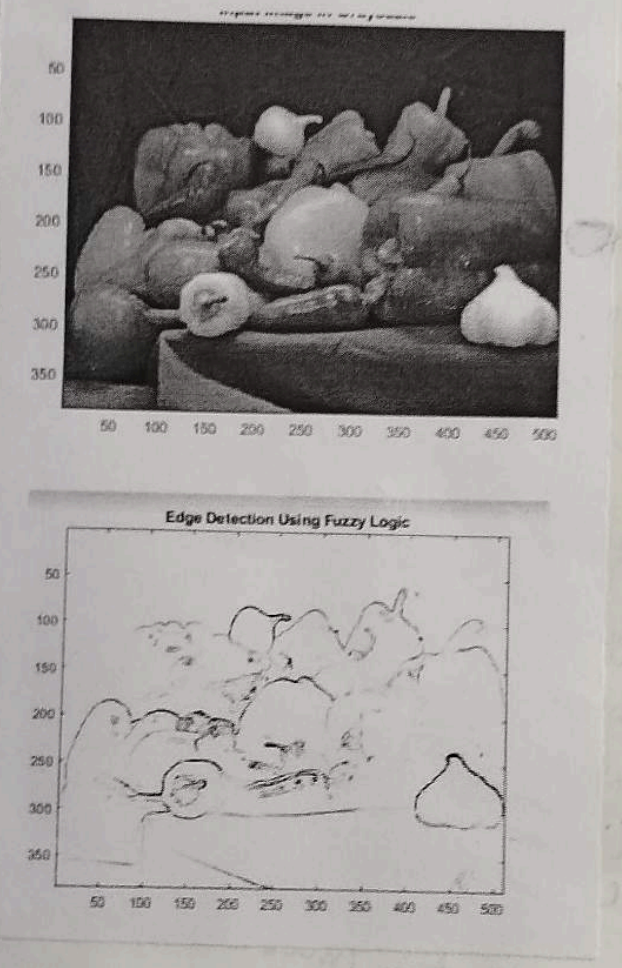
=> Plot the Results.

Plot the original grayscale image.

```
figure
image (I, 'CDataMapping', 'scaled')
colormap ('gray')
title ('Original Grayscale Image')
```

Plot the detected Images.

```
figure
imagesc(level,'CDataMapping','scaled')
colormap('gray')
title('Edge Detection Using Fuzzy Logic')
```



RESULT:

Thus the implementation of Fuzzy logic for Image Processing was successfully implemented.

EX.NO:10

DATE : IMPLEMENTING ARTIFICIAL NEURAL NETWORKS FOR AN APPLICATION USING PYTHON. CLASSIFICATION.

AIM:

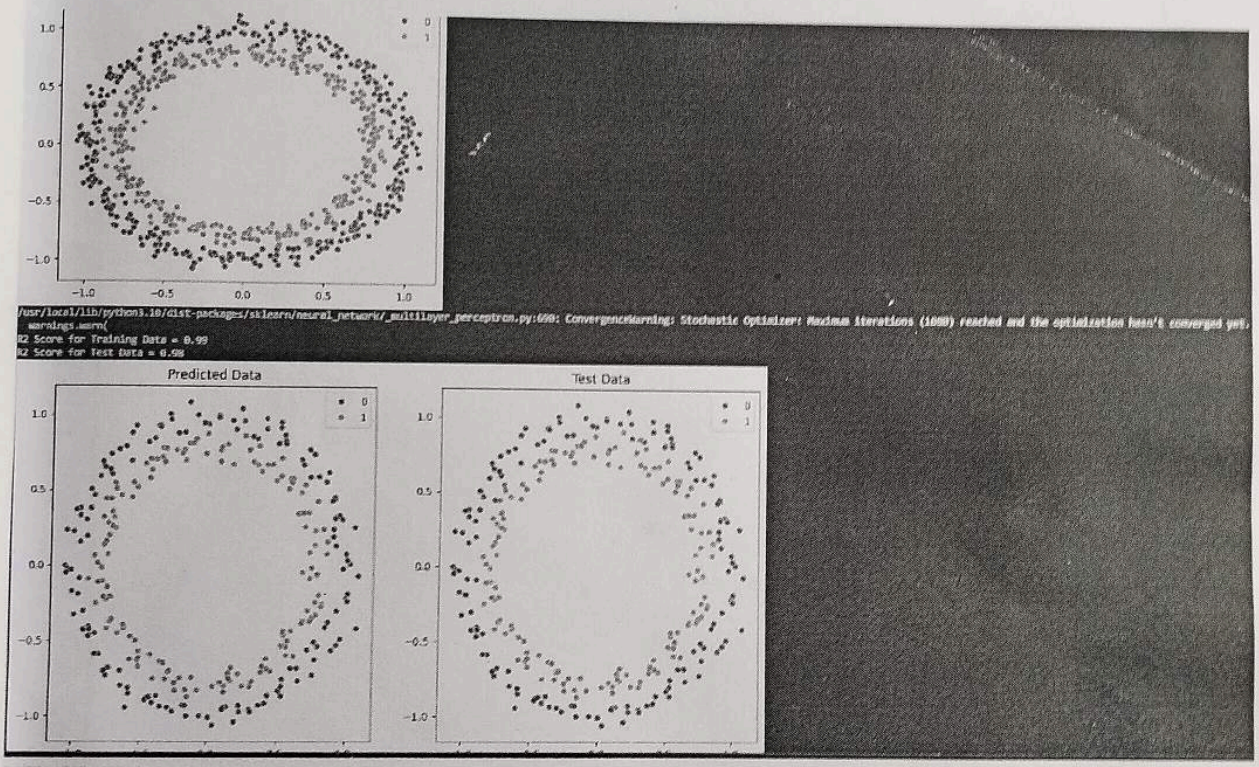
To implementing artificial neural networks for an application in classification using python.

SOURCE CODE:

```
sklearn.model_selection import train_test_split
from sklearn.datasets import make_circles
import from sklearn.neural_network
import MLPClassifier from numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline.

X_train, y_train = make_circles (n_samples=700, noise=0.05)
X_test, y_test = make_circles (n_samples=300, noise=0.05)
sns.scatterplot (x_train[:,0], x_train [:,1], hue=y_train)
plt.title ("Train Data")
plt.show()

clf = MLPclassification (max_iter=1000)
clf.fit (x_train, y_train)
print (f"R2 score for Training Data")

print ("R2 Score for Test Data = {clf.score {x_train, y_train)}")

y_pred = clf.predict (x_test)
fig, ax = plt.subplots (1,2)
sns.scatterplot (x_test[:,0], x_test [:,1], hue=y_pred
                 ax=ax[0])
```

```
ax[1].title.set_text("Predicted Data")
sns.scatterplot(x_test[:,0], x_test[:,1], hue=y_test,
                                          ax=ax[1])
ax[0].title.set_text("Test Data");
plt.show()
```

OUTPUT :



RESULT :

Thus the program for Implementing Artificial Neural Networks for an Application using Python - classification is successfully executed and output is verified.

EX.NO: 4

DATE :

IMPLEMENTING ARTIFICIAL NEURAL NETWORK FOR AN APPLICATION USING PYTHON - REGRESSION.

AIM:

To implementing artificial neural networks for an application in Regression using python.

SOURCE CODE:

```
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import train_test_split
from sklearn.datasets.datasets import make_regression

x,y = make_regression (n_samples = 1000, noise=0.05,
                       n_features = 100)
x.shape, y.shape = ((100,100), (1000,))
x_train, x_test, y_train, y_test = train_test_split
                    (x, y, test_size =0.2, shuffle =True,
                     random_state =42)
clf = MLPRegressor(max_iter =1000)
clf.fit (x_train, y_train)
print (f"R2 Score for Training Data ={clf.score (x_train,
                                    y_train)}")
print(f"R2 Score for Test Data ={clf.score (x_test,
                                y_test)}")
```

OUTPUT:

```
Shapes of X and y: (1000, 100) (1000,)
R2 Score for Training Data = 1.00
R2 Score for Test Data = 0.97
```

RESULT:

Thus, the program was successfully executed and output was verified.

EX.NO: 12

DATE:

DECISION TREE CLASSIFICATION

AIM:

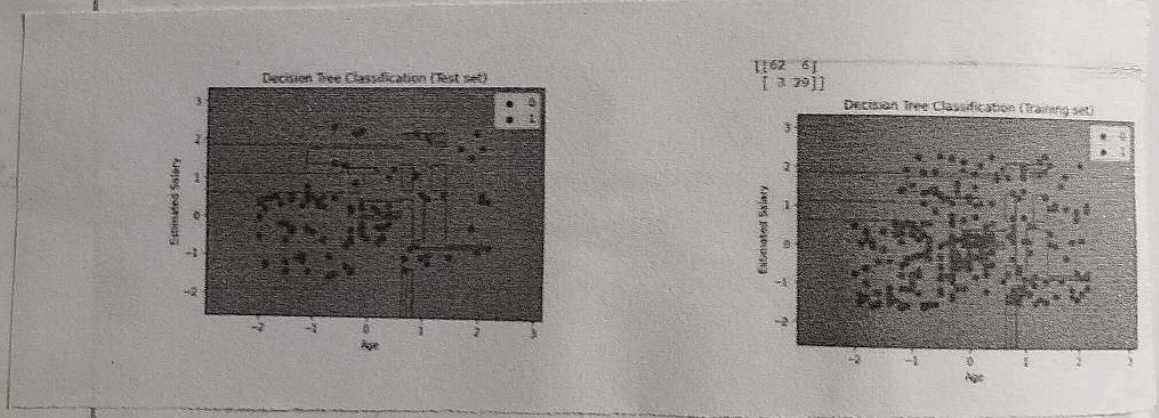
To classify the Social Network dataset using Decision tree analysis.

SOURCE CODE:

```
from google.colab import drive
drive.mount("/content/gdrive")

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
dataset = pd.read_csv('/content/gdrive/My Drive/
                       Social_Network_Ads.csv')

X = dataset.iloc[:,[2,3]].values
y = dataset.iloc[:,-1].value
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
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(x_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
```

```
x1, x2 = np.meshgrid (np.arange (start = x_set[:,0].min() - 1, stop = x_set[:,0].max() + 1, step = 0.01), np.arange (start = x_set[:,1].min() - 1, stop = x_set[:,1].max() + 1, step = 0.01))
for i, j in enumerate (np.unique (y_set)):
    plt.scatter (x_set[y_set == j, 0], x_set[y_set == j, 1],
    = ListedColormap ('red', 'green')) (i), label = j)
plt.title ('Decision Tree Classification (Training set)')
plt.xlabel ('Age')
plt.xlabel ('Purchase')
plt.legend()
plt.show().
```

OUTPUT:



RESULT:

Thus, the program for Decision Tree classification was executed successfully and output is verified.

EX.NO: 13
DATE :

IMPLEMENTATION OF DECISION TREE CLASSIFICATION TECHNIQUES

AIM:

To implement a decision tree classification technique for gender classification using python.

SOURCE CODE:

```
from sklearn import tree
clf = tree.DecisionTreeClassifier()

X = [[181,80,91], [182,90,92], [184,100,94], [185,300,94],
     [186,400,95], [189,600,97], [192,900,100]]

Y = ['male', 'male', 'female', 'male', 'female', 'female']
clf = clf.fit(X, Y)

predictionf = clf.predict([[181,80,91]])
predictionm = clf.predict([[184,100,94]])

print(predictionf)
print(predictionm)
```

OUTPUT:

```
['male']
['female']
```

RESULT:

Thus, the program for DecisionTree Classification technique was successfully executed and output is verified.

EX.NO: 14 IMPLEMENTATION OF CLUSTERING
DATE : TECHNIQUES K-MEANS

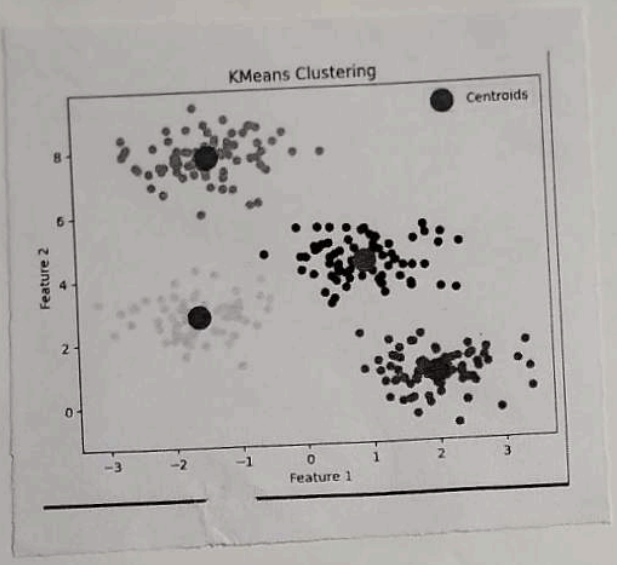
AIM:

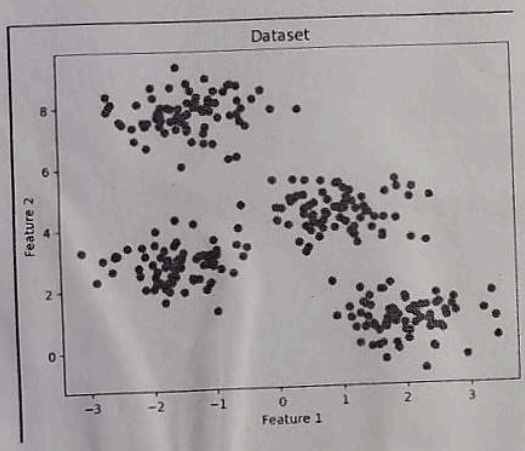
To Implement a k-Means clustering technique using python language.

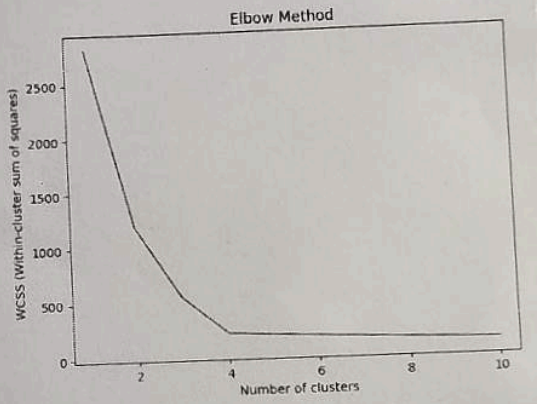
SOURCECODE:

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

x,y = make_blobs (n_samples = 300,
    centers = 4, cluster_std = 0.60, random_state = 0)
plt.scatter (X[:,0], X[:,1])
wcss = []
for i in range (1, 11):
    kmeans = kMeans
    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()

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:







COMPLETED

RESULT:

Thus the program for clustering techniques - KMeans is successfully executed and output is verified.