NPR COLLEGE OF ENGINEERING &TECHNOLOGY
NATHAM - 624 401

NAME : ……………………………………………………….

REGISTER NUMBER : ……………………………………………………….

DEPARTMENT : COMPUTER SCIENCE AND ENGINEERING

YEAR & SEM : II &IV

SUBJECT : CS3491 ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

NPR COLLEGE OF ENGINEERING AND TECHNOLOGYNATHAM,

DINDIGUL -624 401

Name:……………………………………………………………………………..

Year:…….………Semester……………………………Branch..……………

University Register No

CERTIFIED that this a Bonafide Record work done by the above

Student in the……………………………………………………………….Laboratory

during the year 20 - 20

Signature of Lab. In-charge ***Signature of Head of the Department***

Submitted for practical examination held on ………………………………………..

Internal Examiner **External Examiner**

CONTENTS

S.No	Date of Experiment	Name of the Experiment	Page No	Date of Submission	Remarks

NPR COLLEGE OF ENGINEERING & TECHNOLOGY, NATHAM

VISION

- To develop students with intellectual curiosity and technical expertise to meet the global needs.

MISSION

- To achieve academic excellence by offering quality technical education using best teachingtechniques.
- To improve Industry – Institute interactions and expose industrial atmosphere.
- To develop interpersonal skills along with value based education in a dynamic learningenvironment.
- To explore solutions for real time problems in the society.

NPR COLLEGE OF ENGINEERING & TECHNOLOGY, NATHAM

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION

- To produce globally competent technical professionals for digitized society.

MISSION

- To establish conducive academic environment by imparting quality education and value addedtraining.
- To encourage students to develop innovative projects to optimally resolve the challenging socialproblems.

PROGRAM EDUCATIONAL OBJECTIVES

Graduates of Computer Science and Engineering Program will be able to:

- Develop into the most knowledgeable professional to pursue higher education and Research or have asuccessful carrier in industries.
- Successfully carry forward domain knowledge in computing and allied areas to solve complex and realworld engineering problems.
- Meet the technological revolution they are continuously upgraded with the technical knowledge.
- Serve the humanity with social responsibility combined with ethics

CS3491 ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING L T P C
3 0 2 4

OBJECTIVES:

The main objectives of this course are to:

- Study about uninformed and Heuristic search techniques.
- Learn techniques for reasoning under uncertainty
- Introduce Machine Learning and supervised learning algorithms
- Study about ensembling and unsupervised learning algorithms
- Learn the basics of deep learning using neural networks

LIST OF EXPERIMENTS:

1. Implementation of Uninformed search algorithms (BFS, DFS)
2. Implementation of Informed search algorithms (A*, memory-bounded A*)
3. Implement naïve Bayes models
4. Implement Bayesian Networks
5. Build Regression models
6. Build decision trees and random forests
7. Build SVM models
8. Implement ensembling techniques
9. Implement clustering algorithms
10. Implement EM for Bayesian networks
11. Build simple NN models
12. Build deep learning NN models

These Programs can be implemented in Python.

TOTAL:30 PERIODS

OUTCOMES:

At the end of this course, the students will be able to:

CO1: Use appropriate search algorithms for problem solving
CO2: Apply reasoning under uncertainty
CO3: Build supervised learning models
CO4: Build ensembling and unsupervised models
CO5: Build deep learning neural network models

CS3491 ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Course Outcomes

After completion of the course, Students are able to learn the listed Course Outcomes.

Cos	Course Code	Course Outcomes	Knowledge Level
CO1	C212.1	Students can be able to **Understand** the basics of Intelligent agents and use appropriate search algorithms to solve the AI based problem	K2
CO2	C212.2	Students will be able to **Learn** the theoretical knowledge about principles of logic-based representation and techniques for reasoning under uncertainty	K1
CO3	C212.3	Students can be able to **Understand** the basics of Machine Learning and ability to understand the Supervised Learning models.	K2
CO4	C212.4	Students will be able to **Understand** the Ensemble Models and Unsupervised Learning models	K2
CO5	C212.5	Students will be able to Describe the basic knowledge of Deep Learning Neural networks and **Demonstrate** the various models to solve the real time complex problems.	K3

List of Experiments with COs, POs and PSOs

Exp.No.	Name of the Experiment	COs	POs	PSOs
1.	Develop a Program to implement Breadth First Search Method.	CO1	PO1,2,3	PSO1,2
2.	Develop a Program to implement Depth First Search Method.	CO1	PO1,2,3	PSO1,2
3.	Develop a Program to implement the A* search algorithm in the Eight puzzle	CO1	PO1,2,3	PSO1,2
4.	Develop a Program to implement Best First Search algorithm	CO1	PO1,2,3	PSO1,2
5.	Develop a Program to Implement Naive Bayes Classification by using the Advertisement clicking dataset and Compute the accuracy of the classifier	CO2	PO1,2,3	PSO1,2
6.	Develop a Program to build a Linear regression Model	CO3	PO1,2,3	PSO1,2
7.	Develop a Program to build a Logistic Regression Model	CO3	PO1,2,3	PSO1,2

8.	Develop a Program to construct a Bayesian network to diagnosis of heart patients using standard Heart Disease Data Set.	CO3	PO1,2,3	PSO1,2
9.	Develop a Program to build Decision tree and classify the Result using the Gini Index,Entrophy	CO3	PO1,2,3	PSO1,2
10.	Develop a Random Forest algorithm by using the data set of Kyphosis patients to predict whether or not patients have the disease	CO3	PO1,2,3	PSO1,2
11.	Develop a Program to build Support Vector Machine by using the Social Network advertisement Dataset and find the accuracy of the given dataset.	CO3	PO1,2,3	PSO1,2
12.	Develop a Program to implement Ensemble Voting classifier technique for IRIS dataset and find the accuracy score of Hard Voting and Soft Voting	CO4	PO1,2,3	PSO1,2
13.	Develop a Program to implement Ensemble Averaging classifier technique and find the accuracy score of averaging classifier	CO4	PO1,2,3	PSO1,2
14.	Develop a Program to build k-means clustering algorithm by generate isotropic Gaussian Clustering blobs and find the optimal number of clusters by using the Elbow Method.	CO4	PO1,2,3	PSO1,2
15.	Develop a Program to build k-nearest neighbor clustering algorithm by using classified dataset and find precision, recall ,F1 score, Support by using different number of clusters.	CO4	PO1,2,3	PSO1,2
16.	Develop a Program to implement Expectation Maximization algorithm	CO4	PO1,2,3	PSO1,2
17.	Develop a program to build a Sequential Neural Network and calculate its accuracy.	CO5	PO1,2,3	PSO1,2
18.	Develop a program to build a Deep Learning NN Model using MNIST Datasets	CO5	PO1,2,3	PSO1,2

Program Outcomes

1. Engineering Knowledge
2. Problem Analysis
3. Design/Development of Solutions
4. Conduct Investigations of Complex Problems
5. Modern Tool Usage
6. The Engineer and Society
7. Environment and Sustainability
8. Ethics
9. Individual and Team Work
10. Communication
11. Project Management and Finance
12. Life-long Learning

Program Specific Outcomes

At the end of the Program students will be able to

- Deal with real time problems by understanding the evolutionary changes in computing, applying standard practices and strategies in software project development using open-ended Programmingenvironments.
- Employ modern computer languages, environments and platforms in creating innovative career pathsby inculcating moral values and ethics.
- Achieve additional expertise through add-on and certificate Programs.

Ex.No: A1 Date:	UNINFORMED SEARCH ALGORITHMS

AIM:

To Develop a Program to implement Breadth First Search Method.

Algorithm:

1. Start at the root node and push it onto the stack.
2. Check for any adjacent nodes of the tree and select one node.
3. Traverse the entire branch of the selected node and push all the nodes into the stack.
4. Upon reaching the end of a branch (no more adjacent nodes) ie nth leaf node, move back by a single step and look for adjacent nodes of the n-1th node.
5. If there are adjacent nodes for the n-1th node, traverse those branches and push nodes onto the stack.

PROGRAM:

```
from queue import Queue

graph = {0: [1, 3], 1: [0, 2, 3], 2: [4, 1, 5], 3: [4, 0, 1], 4: [2, 3, 5], 5: [4, 2], 6: []}
print("The adjacency List representing the graph is:")
print(graph)



def bfs(graph, source):
    Q = Queue()
    visited_vertices = set()
    Q.put(source)
    visited_vertices.update({0})
    while not Q.empty():
        vertex = Q.get()
        print(vertex, end="-->")
        for u in graph[vertex]:
            if u not in visited_vertices:
                Q.put(u)
                visited_vertices.update({u})

print("BFS traversal of graph with source 0 is:")
bfs(graph, 0)
```

OUTPUT:

The adjacency List representing the graph is:

{0: [1, 3], 1: [0, 2, 3], 2: [4, 1, 5], 3: [4, 0, 1], 4: [2, 3, 5], 5: [4, 2], 6: []}

BFS traversal of graph with source 0 is:

0-->1-->3-->2-->4-->5-->

RESULT:

Thus the above Program was successfully executed and the Output was obtained

Ex.No: A2 Date:	UNINFORMED SEARCH ALGORITHMS

AIM:

To Develop a Program to implement Depth First Search Method.

ALGORITHM :

Input: Graph(Adjacency list) and Source vertex

OUTPUT: DFS traversal of graph

Start:

1.Create an empty stack S.

2.Create an empty list to keep record of visited vertices.

3.Insert source vertex into S, mark the source as visited.

4.If S is empty, return. Else goto 5.

5.Take out a vertex v from S.

6.Print the Vertex v.

7.Insert all the unvisited vertices in the adjacency list of v into S and mark them visited.

10.Goto 4.

Stop.

PROGRAM:

```
graph = {0: [1, 3], 1: [0, 2, 3], 2: [4, 1, 5], 3: [4, 0, 1], 4: [2, 3, 5], 5: [4, 2], 6: []}
print("The adjacency List representing the graph is:")
print(graph)

def dfs_explanation(graph, source):
    S = list()
    visited_vertices = list()
    S.append(source)
    visited_vertices.append(source)
    while S:
        vertex = S.pop()
        print("processing vertex {}.".format(vertex))
        for u in graph[vertex]:
            if u not in visited_vertices:
                print("At {}, adding {} to Stack".format(vertex, u))
                S.append(u)
                visited_vertices.append(u)
        print("Visited vertices are:", visited_vertices)
```

```
print("Explanation of DFS traversal of graph with source 0 is:")
dfs_explanation(graph, 0)
```

OUTPUT:

```
The adjacency List representing the graph is:
{0: [1, 3], 1: [0, 2, 3], 2: [4, 1, 5], 3: [4, 0, 1], 4: [2, 3, 5], 5: [4, 2], 6: []}
Explanation of DFS traversal of graph with source 0 is:
processing vertex 0.
At 0, adding 1 to Stack
At 0, adding 3 to Stack
Visited vertices are: [0, 1, 3]
processing vertex 3.
At 3, adding 4 to Stack
Visited vertices are: [0, 1, 3, 4]
processing vertex 4.
At 4, adding 2 to Stack
At 4, adding 5 to Stack
Visited vertices are: [0, 1, 3, 4, 2, 5]
processing vertex 5.
Visited vertices are: [0, 1, 3, 4, 2, 5]
processing vertex 2.
Visited vertices are: [0, 1, 3, 4, 2, 5]
processing vertex 1.
Visited vertices are: [0, 1, 3, 4, 2, 5]
```

RESULT:

Thus the above program was successfully executed and the output was obtained

Ex.No: B1 Date:	INFORMED SEARCH ALGORITHMS

AIM:

To Develop a Program to implement the A* search algorithm in the Eight puzzle

ALGORITHM:

1. The implementation of A* Algorithm involves maintaining two lists- OPEN and CLOSED.
2. OPEN contains those nodes that have been evaluated by the heuristic function but have not been expanded into successors yet.
3. CLOSED contains those nodes that have already been visited.

The algorithm is as follows-

Step-01: Define a OPEN.Initially, OPEN consists solely of a single node, the start node S.

Step-02: If the list is empty, return failure and exit.

Step-03:Remove node n with the smallest value of f(n) from OPEN and move it to list CLOSED.

If node n is a goal state, return success and exit.

Step-04: Expand node n.

Step-05:

- If any successor to n is the goal node, return success and the solution by tracing the path from goal node to S.
- Otherwise, go to Step-06.

Step-06:

For each successor node,

- Apply the evaluation function f to the node.
- If the node has not been in either list, add it to OPEN.

Step-07: Go back to Step-02.

PROGRAM:

```
from copy import deepcopy
import numpy as np
import time

def bestsolution(state):
    bestsol = np.array([], int).reshape(-1, 9)
    count = len(state) - 1
    while count != -1:
        bestsol = np.insert(bestsol, 0, state[count]['puzzle'], 0)
        count = (state[count]['parent'])
```

```
    return bestsol.reshape(-1, 3, 3)



# checks for the uniqueness of the iteration(it).
def all(checkarray):
    set=[]
    for it in set:
        for checkarray in it:
            return 1
        else:
            return 0



# number of misplaced tiles
def misplaced_tiles(puzzle,goal):
    mscost = np.sum(puzzle != goal) - 1
    return mscost if mscost > 0 else 0



def coordinates(puzzle):
    pos = np.array(range(9))
    for p, q in enumerate(puzzle):
        pos[q] = p
    return pos



# start of 8 puzzle evaluvation, using Misplaced tiles heuristics
def evaluvate_misplaced(puzzle, goal):
    steps = np.array([('up', [0, 1, 2], -3),('down', [6, 7, 8],  3),('left', [0, 3, 6], -1),('right', [2,
5, 8],  1)],
                dtype =  [('move',  str, 1),('position', list),('head', int)])

    dtstate = [('puzzle',  list),('parent', int),('gn',  int),('hn',  int)]

    costg = coordinates(goal)
    # initializing the parent, gn and hn, where hn is misplaced_tiles  function call
    parent = -1
    gn = 0
    hn = misplaced_tiles(coordinates(puzzle), costg)
```

```
    state = np.array([(puzzle, parent, gn, hn)], dtstate)

   #priority queues with position as keys and fn as value.
    dtpriority = [('position', int),('fn', int)]

    priority = np.array([(0, hn)], dtpriority)

    while 1:
        priority = np.sort(priority, kind='mergesort', order=['fn', 'position'])
        position, fn = priority[0]
        # sort priority queue using merge sort,the first element is picked for exploring.
        priority = np.delete(priority, 0, 0)
        puzzle, parent, gn, hn = state[position]
        puzzle = np.array(puzzle)

        blank = int(np.where(puzzle == 0)[0])

        gn = gn + 1
        c = 1
        start_time = time.time()
        for s in steps:
            c = c + 1
            if blank not in s['position']:
                openstates = deepcopy(puzzle)
                openstates[blank], openstates[blank + s['head']] = openstates[blank + s['head']],
openstates[blank]

                if ~(np.all(list(state['puzzle']) == openstates, 1)).any():
                    end_time = time.time()
                    if (( end_time - start_time ) > 2):
                        print(" The 8 puzzle is unsolvable \n")
                        break

                    hn = misplaced_tiles(coordinates(openstates), costg)
                    # generate and add new state in the list
                    q = np.array([(openstates, position, gn, hn)], dtstate)
                    state = np.append(state, q, 0)
                    # f(n) is the sum of cost to reach node
                    fn = gn + hn
```

```
            q = np.array([(len(state) - 1, fn)], dtpriority)
            priority = np.append(priority, q, 0)

            if np.array_equal(openstates, goal):
                print(' The 8 puzzle is solvable \n')
                return state, len(priority)

    return state, len(priority)
# initial state
puzzle = []
puzzle.append(2)
puzzle.append(8)
puzzle.append(3)
puzzle.append(1)
puzzle.append(6)
puzzle.append(4)
puzzle.append(7)
puzzle.append(0)
puzzle.append(5)

#goal state
goal = []
goal.append(1)
goal.append(2)
goal.append(3)
goal.append(8)
goal.append(0)
goal.append(4)
goal.append(7)
goal.append(6)
goal.append(5)
state, visited = evaluvate_misplaced(puzzle, goal)
bestpath = bestsolution(state)
print(str(bestpath).replace('[', ' ').replace(']', ''))
totalmoves = len(bestpath) - 1
print('\nSteps to reach goal:',totalmoves)
visit = len(state) - visited
print('Total nodes visited: ',visit, "\n")
```

OUTPUT:

The 8 puzzle is solvable

```
2 8 3
1 6 4
7 0 5

2 8 3
1 0 4
7 6 5

2 0 3
1 8 4
7 6 5

0 2 3
1 8 4
7 6 5

1 2 3
0 8 4
7 6 5

1 2 3
8 0 4
7 6 5
```

Steps to reach goal: 5

Total nodes visited: 6

RESULT:

Thus the above Program was successfully executed and the Output was obtained

Ex.No: B2 Date:	INFORMED SEARCH ALGORITHMS

AIM:

To Develop a program implement Best First Search algorithm

Algorithm:

Step 1 : Create a priorityQueue .

Step 2 : insert ‘start’ in pqueue : pqueue.insert(start)

Step 3 : delete all elements of pqueue one by one.

Step 3.1 : if, the element is goal . Exit.

Step 3.2 : else, traverse neighbours and mark the node examined.

Step 4 : End.

PROGRAM:

```
from queue import PriorityQueue
v = 14
graph = [[] for i in range(v)]

# Function For Implementing Best First Search
# Gives OUTPUT path having lowest cost


def best_first_search(actual_Src, target, n):
    visited = [False] * n
    pq = PriorityQueue()
    pq.put((0, actual_Src))
    visited[actual_Src] = True

    while pq.empty() == False:
        u = pq.get()[1]
        # Displaying the path having lowest cost
        print(u, end=" ")
        if u == target:
            break

        for v, c in graph[u]:
            if visited[v] == False:
                visited[v] = True
                pq.put((c, v))
```

```
    print()

# Function for adding edges to graph


def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))


# The nodes shown in above example(by alphabets) are
# implemented using integers addedge(x,y,cost);
addedge(0, 1, 3)
addedge(0, 2, 6)
addedge(0, 3, 5)
addedge(1, 4, 9)
addedge(1, 5, 8)
addedge(2, 6, 12)
addedge(2, 7, 14)
addedge(3, 8, 7)
addedge(8, 9, 5)
addedge(8, 10, 6)
addedge(9, 11, 1)
addedge(9, 12, 10)
addedge(9, 13, 2)

source = 0
target = 9
best_first_search(source, target, v)
```

OUTPUT: 0 1 3 2 8 9

RESULT:

Thus the above Program was successfully executed and the Output was obtained

Ex.No: C Date:	NAÏVE BAYES MODEL

AIM:

To develop a Program to Implement Naive Bayes Classification in Python by using the Advertisement clicking dataset (about users clicking the ads or not). Compute the accuracy of the classifier, considering few test data sets

The dataset has the following features,

Daily Time Spent on Site — Amount of time spent on the website

Age — User's Age

Area Income — Avg revenue of the Users

Daily Internet Usage — Avg usage of internet daily

Ad Topic Line — Topic text of the advertisement

City — City of the Users

Male — gender of the users(male or female)

Country — Country of the users

Timestamp — Time clicked on the Ad

Clicked on Ad — 0 or 1, 0-not clicked,1-clicked.

Algorithm:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn
#loading the dataset
data=pd.read_csv('C:\\Users\\jenim\\Downloads\\AI ML\\AI ML\\Advertising.csv')
#head of the dataset
data.head()
#describing the data
data.describe()
#drop 'Ad Line Topic','City', 'Country' and Timestamp.
data.drop(['Ad Topic Line','City','Country','Timestamp'],axis= 1,inplace=True)
data.head()
#train test split the data
X = data.iloc[:,0:4].values
Y = data['Clicked on Ad'].values
from sklearn.model_selection import train_test_split
```

```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size= 1/3,random_state= 0)
print('X_train shape:',X_train.shape)
print('X_test shape:',X_test.shape)
print("Y train shape:",Y_train.shape)
print('Y test shape:',X_test.shape)
#feature scaling
from sklearn.preprocessing import StandardScaler
stdscaler= StandardScaler()
X_train=stdscaler.fit_transform(X_train)
X_test=stdscaler.transform(X_test)
#naive bayes model
from sklearn.naive_bayes import GaussianNB
clf=GaussianNB()
clf.fit(X_train,Y_train)
#predicting the RESULT
y_pred=clf.predict(X_test)
print(y_pred)
#confusion matrix
from sklearn.metrics import confusion_matrix,accuracy_score
cm=confusion_matrix(Y_test, y_pred)
print(cm)
#accuracy score of the model
print('Accuracy score :',accuracy_score(Y_test,y_pred))
#plotting the confusion matrix
plt.figure(figsize=(10,5))
plt.title("Confusion matrix")
sns.heatmap(cm,annot=True,fmt='d',cmap='inferno_r')
```

OUTPUT:

data=pd.read_csv('C:\\Users\\jenim\\Downloads\\AI ML\\AI ML\\Advertising.csv')

data.head()

Out[6]:

```
   Daily Time Spent on Site  Age  ...         Timestamp  Clicked on Ad
0                     68.95   35  ...  27-03-2016 00:53              0
1                     80.23   31  ...  04-04-2016 01:39              0
2                     69.47   26  ...  13-03-2016 20:35              0
3                     74.15   29  ...  10-01-2016 02:31              0
4                     68.37   35  ...  03-06-2016 03:36              0
```

[5 rows x 10 columns]

data.drop(['Ad Topic Line','City','Country','Timestamp'],axis= 1,inplace=True)

data.head()
Out[8]:

```
   Daily Time Spent on Site  Age  ...  Male  Clicked on Ad
0                     68.95   35  ...     0              0
1                     80.23   31  ...     1              0
2                     69.47   26  ...     0              0
3                     74.15   29  ...     1              0
4                     68.37   35  ...     0              0

[5 rows x 6 columns]
```

print(y_pred)

```
[0 0 0 1 0 1 1 1 0 1 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 1 0 1 1 0 0
 0 0 0 0 1 1 0 1 0 0 1 0 1 0 0 1 1 0 1 0 1 1 1 0 1 1 1 1 1 0 1 0 1 1 0 1 0
 0 0 1 0 1 0 1 1 0 1 0 1 1 0 1 0 0 1 1 0 1 1 0 1 0 0 1 0 0 1 0 1 0 1 0 0 1
 1 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 1 0 1 1 0 1 1 1 0 1 1 0
 1 1 0 0 0 1 1 1 0 0 0 1 1 0 0 0 1 1 1 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 1 1 0
 1 0 1 1 0 0 1 1 0 0 1 1 1 0 1 0 1 1 0 1 0 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0
 0 1 1 0 1 0 1 1 0 0 1 1 0 1 0 1 1 1 0 1 0 0 1 0 1 0 1 0 1 0 1 1 0 0 0 1 0
 1 1 0 1 0 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 1 1 1 1 1 0 0 0 1 1 0
 1 0 1 1 0 0 1 0 0 0 1 0 1 0 1 1 1 0 0 0 1 0 0 1 0 0 1 1 0 1 1 1 1 1 0 1 1
 0]
```

X_train shape: (666, 4)
X_test shape: (334, 4)
Y train shape: (666,)
Y test shape: (334, 4)

```
[0 0 0 1 0 1 1 1 0 1 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 1 0 1 1 0 0
 0 0 0 0 1 1 0 1 0 0 1 0 1 0 0 1 1 0 1 0 1 1 1 0 1 1 1 1 1 0 1 0 1 1 0 1 0
 0 0 1 0 1 0 1 1 0 1 0 1 1 0 1 0 0 1 1 0 1 1 0 1 0 0 1 0 0 1 0 1 0 1 0 0 1
 1 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 1 0 1 1 0 1 1 1 0 1 1 0
 1 1 0 0 0 1 1 1 0 0 0 1 1 0 0 0 1 1 1 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 1 1 0
 1 0 1 1 0 0 1 1 0 0 1 1 1 0 1 0 1 1 0 1 0 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0
 0 1 1 0 1 0 1 1 0 0 1 1 0 1 0 1 1 1 0 1 0 0 1 0 1 0 1 0 1 0 1 1 0 0 0 1 0
 1 1 0 1 0 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 1 1 1 1 1 0 0 0 1 1 0
 1 0 1 1 0 0 1 0 0 0 1 0 1 0 1 1 1 0 0 0 1 0 0 1 0 0 1 1 0 1 1 1 1 1 0 1 1
```

0]

cm=confusion_matrix(Y_test, y_pred)

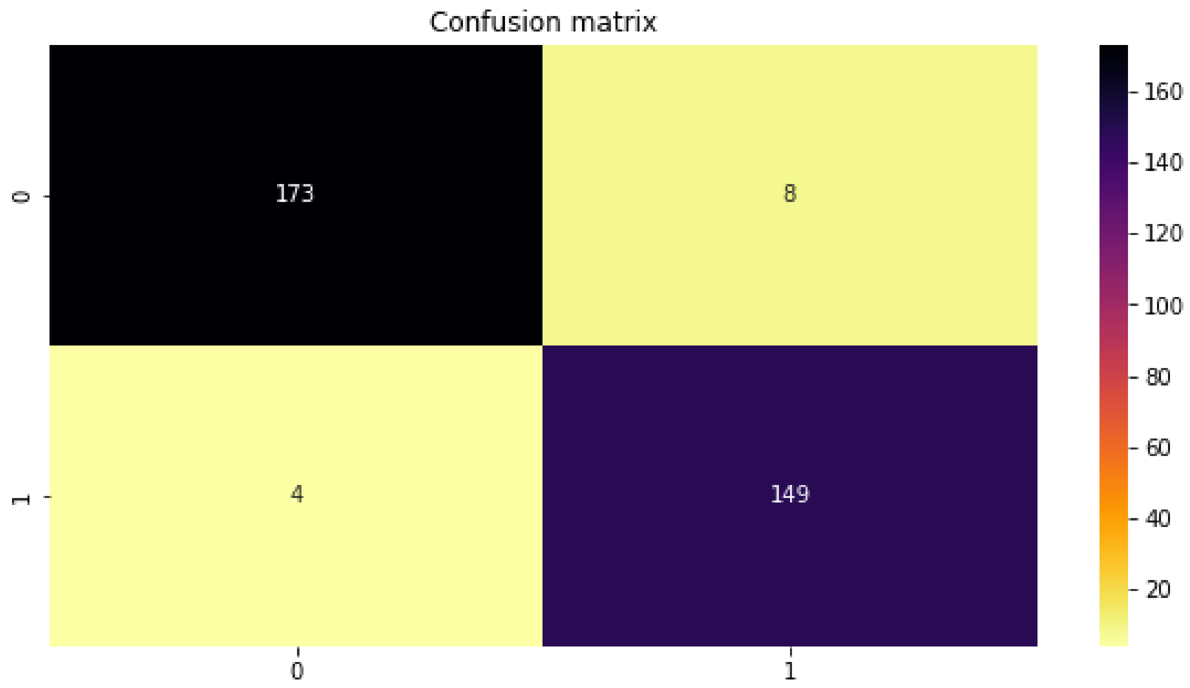
print(cm)

[[173   8]

 [4 149]]

print('Accuracy score :',accuracy_score(Y_test,y_pred))

Accuracy score : 0.9640718562874252



RESULT:

Thus the above Program was successfully executed and the Output was obtained

Ex.No: D1 Date:	**LINEAR REGRESSION MODEL**

AIM:

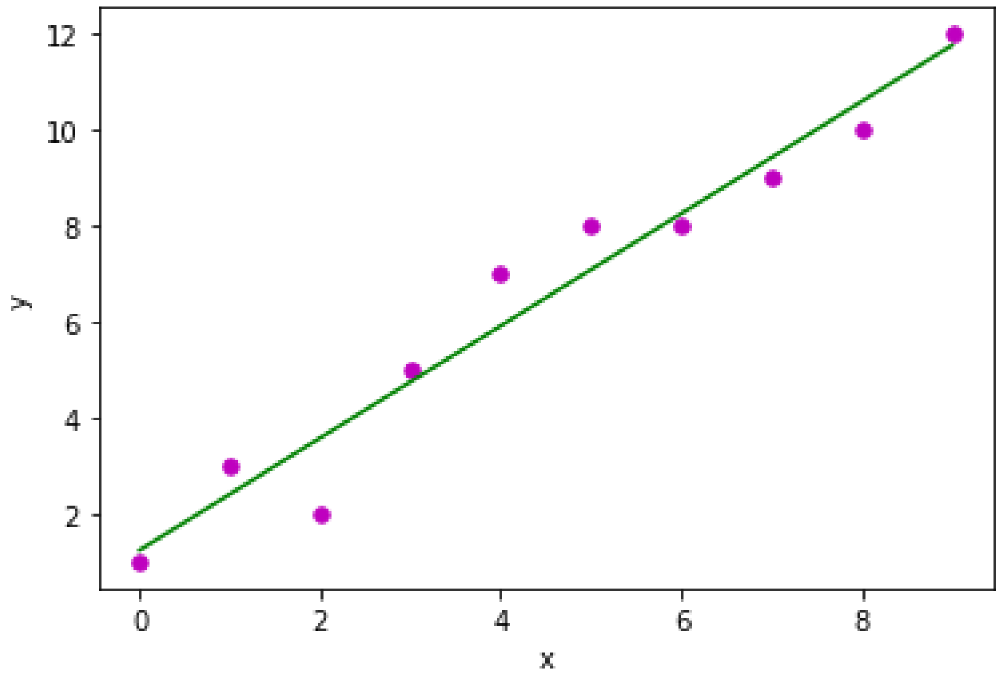
To develop a Program to build a Linear Regression Model

PROGRAM:

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

def estimate_coef(x,y):
    n=np.size(x)
    m_x=np.mean(x)
    m_y=np.mean(y)
    SS_xy=np.sum(x*y)-n*m_y*m_x
    SS_xx=np.sum(x*x)-n*m_x*m_x

    b_1=SS_xy/SS_xx  # corrected typo in the denominator
    b_0=m_y-b_1*m_x
    return (b_0,b_1)

def plot_regression_line(x,y,b):
    plt.scatter(x,y,color="m",marker="o",s=30)
    y_pred=b[0]+b[1]*x
    plt.plot(x,y_pred,color="g")
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()

def main():
    x=np.array([0,1,2,3,4,5,6,7,8,9])
    y=np.array([1,3,2,5,7,8,8,9,10,12])
    b=estimate_coef(x,y)
    print("estimated coefficient :\nb_0={}\nb_1={}".format(b[0],b[1]))
    plot_regression_line(x,y,b)

if __name__=="__main__":
    main()
```

OUTPUT:

estimated coefficient :
b_0=1.2363636363636363
b_1=1.1696969696969697



RESULT:

Thus the above Program was successfully executed and the Output was obtained

Ex.No: D2 **Date:**	**LOGISTIC REGRESSION MODEL**

AIM:

To develop a Program to build a Logistic Regression Model

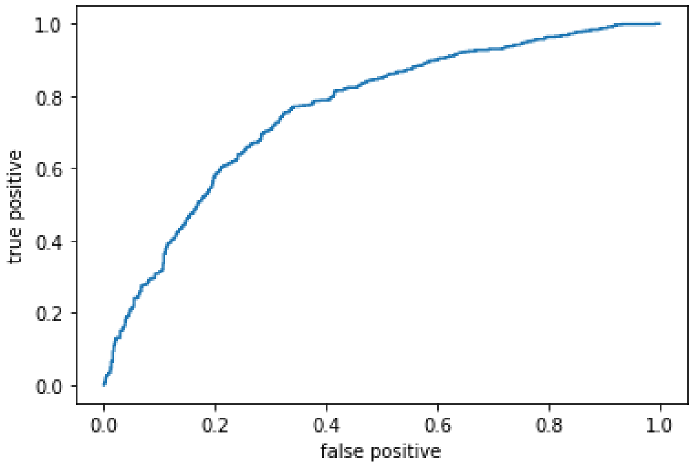
PROGRAM:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
insurance=pd.read_csv('C:\\Users\\CSELAB21\\Downloads\\claimants.csv')
insurance.columns
insurance.drop(['CASENUM'],axis=1,inplace=True)
insurance.columns
insurance.isna().sum()
insurance.iloc[:,:4]
insurance.ATTORNEY.value_counts()
insurance.ATTORNEY.mode()[0]
insurance.CLMSEX.value_counts()
insurance.CLMSEX.mode()[0]
insurance.iloc[:,:4]=insurance.iloc[:,:4].apply(lambda x: x.fillna(x.mode()[0]))
insurance.CLMAGE=insurance.CLMAGE.fillna(insurance.CLMAGE.mean())
insurance.isna().sum()

############Model building###################
import statsmodels.formula.api as smf
insurance_model=smf.logit('ATTORNEY~CLMSEX+CLMINSUR+SEATBELT+C
LMAGE+LOSS',data=insurance).fit()
insurance_model.summary()
insurance_model1=smf.logit('ATTORNEY~CLMSEX+CLMINSUR+CLMAGE+LO
SS',data=insurance).fit()
insurance_model1.summary()
#seatbelt is not significant
insu_pred=insurance_model1.predict(insurance)
insu_pred
insurance['pred_prob']=insu_pred
insurance['att_val']=0
insurance.loc[insu_pred>=0.5,'att_val']=1
insurance.att_val

#############Confusion matrix############
from sklearn.metrics import classification_report
classification_report(insurance.ATTORNEY,insurance.att_val)
confusion_matrix=pd.crosstab(insurance.ATTORNEY,insurance.att_val)
confusion_matrix
accuracy=(436+504)/(436+249+151+504)
accuracy

############ROC curve##################
from sklearn import metrics
fpr,tpr,threshold=metrics.roc_curve(insurance.ATTORNEY,insu_pred)
```

```
plt.plot(fpr,tpr);plt.xlabel('false positive');plt.ylabel('true positive')
roc_auc=metrics.auc(fpr,tpr)#area under curve
roc_auc# -*- coding: utf-8 -*-
```

OUTPUT:

```
Optimization terminated successfully.
         Current function value: 0.609131
         Iterations 7
Optimization terminated successfully.
         Current function value: 0.609777
         Iterations 7
```



RESULT:

Thus the above Program was successfully executed and the Output was obtained

Ex.No: E Date:	IMPLEMENT BAYESIAN NETWORKS

AIM:

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

PROGRAM:

```
import numpy as np
import csv
import pandas as pd
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianNetwork
from pgmpy.inference import VariableElimination

data = pd.read_csv('C:\\Users\\jenim\\Downloads\\AI ML\\AI ML\\ds4.csv')
heart_disease = pd.DataFrame(data)
print(heart_disease)

model = BayesianNetwork([
    ('age', 'Lifestyle'),
    ('Gender', 'Lifestyle'),
    ('Family', 'heartdisease'),
    ('diet', 'cholestrol'),
    ('Lifestyle', 'diet'),
    ('cholestrol', 'heartdisease'),
    ('diet', 'cholestrol')
])

model.fit(heart_disease, estimator=MaximumLikelihoodEstimator)

HeartDisease_infer = VariableElimination(model)

print('For Age enter SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4')
print('For Gender enter Male:0, Female:1')
print('For Family History enter Yes:1, No:0')
print('For Diet enter High:0, Medium:1')
print('for LifeStyle enter Athlete:0, Active:1, Moderate:2, Sedentary:3')
print('for Cholesterol enter High:0, BorderLine:1, Normal:2')
```

```
q = HeartDisease_infer.query(variables=['heartdisease'], evidence={
    'age': int(input('Enter Age: ')),
    'Gender': int(input('Enter Gender: ')),
    'Family': int(input('Enter Family History: ')),
    'diet': int(input('Enter Diet: ')),
    'Lifestyle': int(input('Enter Lifestyle: ')),
    'cholestrol': int(input('Enter Cholestrol: '))
})

print(q)
```

OUTPUT:

```
    age  Gender  Family  diet  Lifestyle  cholestrol  heartdisease
0     0       0       1     1          3           0             1
1     0       1       1     1          3           0             1
2     1       0       0     0          2           1             1
3     4       0       1     1          3           2             0
4     3       1       1     0          0           2             0
5     2       0       1     1          1           0             1
6     4       0       1     0          2           0             1
7     0       0       1     1          3           0             1
8     3       1       1     0          0           2             0
9     1       1       0     0          0           2             1
10    4       1       0     1          2           0             1
11    4       0       1     1          3           2             0
12    2       1       0     0          0           0             0
13    2       0       1     1          1           0             1
14    3       1       1     0          0           1             0
15    0       0       1     0          0           2             1
16    1       1       0     1          2           1             1
17    3       1       1     1          0           1             0
18    4       0       1     1          3           2             0
For Age enter SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4
For Gender enter Male:0, Female:1
For Family History enter Yes:1, No:0
For Diet enter High:0, Medium:1
for LifeStyle enter Athlete:0, Active:1, Moderate:2, Sedentary:3
for Cholesterol enter High:0, BorderLine:1, Normal:2
```

```
Enter Age: 4

Enter Gender: 0

Enter Family History: 1

Enter Diet: 1

Enter Lifestyle: 3

Enter Cholestrol: 2
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.8333 |
+-----------------+---------------------+
| heartdisease(1) |              0.1667 |
+-----------------+---------------------
```

RESULT:

Thus the above Program was successfully executed and the Output was obtained

Ex.No: F1 Date:	DECISION TREES MODEL

AIM:

To develop a Program to build Decision tree and classify the result using the Gini Index,Entrophy

PROGRAM:

```
import numpy as np
import pandas as pd
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, classification_report

# Function importing Dataset
def importdata():
    balance_data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-
databases/balance-scale/balance-scale.data', header=None)
    return balance_data

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

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

# 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 gini index.
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 train_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):
    # Prediction 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)*100)
    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 = train_using_entropy(X_train, X_test, y_train)
    # Testing 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:

```
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' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'R'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'R']
Confusion 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
```

RESULT:

Thus the above Program was successfully executed and the Output was obtained

Ex.No: F2 Date:	RANDOM FOREST MODEL

AIM:

To develop a Random Forest algorithm by using the data set of Kyphosis patients to predict whether or not patients have the disease

PROGRAM:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
#matplotlib inline
# Load the data from the CSV file
raw_data = pd.read_csv('C:\\Users\\jenim\\Downloads\\AI ML\\AI
ML\\kyphosis.csv')
# Explore the data with info() and pairplot()
raw_data.info()
sns.pairplot(raw_data, hue='Kyphosis')

# Split the data into training and test sets
from sklearn.model_selection import train_test_split

# Separate the input features (X) and target variable (y)
X = raw_data.drop('Kyphosis', axis=1)
y = raw_data['Kyphosis']

# Split the data into training and test sets using train_test_split()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Train the decision tree model
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
# Measure the performance of the decision tree model
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_test, predictions))
print(confusion_matrix(y_test, predictions))

# Train the random forest model
from sklearn.ensemble import RandomForestClassifier
random_forest_model = RandomForestClassifier()
random_forest_model.fit(X_train, y_train)
random_forest_predictions = random_forest_model.predict(X_test)

# Measure the performance of the random forest model
print(classification_report(y_test, random_forest_predictions))
print(confusion_matrix(y_test, random_forest_predictions))
```

OUTPUT:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 81 entries, 0 to 80
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Kyphosis  81 non-null     object
 1   Age       81 non-null     int64
 2   Number    81 non-null     int64
 3   Start     81 non-null     int64
dtypes: int64(3), object(1)
memory usage: 2.7+ KB
              precision    recall  f1-score   support
      absent       0.78      0.78      0.78        18
     present       0.43      0.43      0.43         7

    accuracy                           0.68        25
   macro avg       0.60      0.60      0.60        25
weighted avg       0.68      0.68      0.68        25

[[14  4]
 [ 4  3]]
              precision    recall  f1-score   support
      absent       0.73      0.89      0.80        18
     present       0.33      0.14      0.20         7

    accuracy                           0.68        25
   macro avg       0.53      0.52      0.50        25
weighted avg       0.62      0.68      0.63        25

[[16  2]
 [ 6  1]]
```

RESULT:

Thus the above Program was successfully executed and the Output was obtained

Ex.No: G Date:	**SUPPORT VECTOR MACHINE MODEL**

AIM:

To develop a Program to build Support Vector Machine by using the Social Network advertisement Dataset and find the accuracy of the given dataset.

PROGRAM:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder

# read data from csv file
data = pd.read_csv('C:\\Users\\jenim\\Downloads\\AI ML\\AI
ML\\apples_and_oranges.csv')
#print(data)

# splitting data into training and test set
training_set,test_set = train_test_split(data,test_size=0.2,random_state=1)
#print("train:",training_set)
#print("test:",test_set)

# prepare data for applying it to svm
x_train = training_set.iloc[:,0:2].values  # data
y_train = training_set.iloc[:,2].values  # target
x_test = test_set.iloc[:,0:2].values  # data
y_test = test_set.iloc[:,2].values  # target
#print(x_train,y_train)
#print(x_test,y_test)

# fitting the data (train a model)
classifier = SVC(kernel='rbf',random_state=1,C=1,gamma='auto')
classifier.fit(x_train,y_train)

# perform prediction on x_test data
y_pred = classifier.predict(x_test)
#test_set['prediction']=y_pred
#print(y_pred)
```

```
# creating confusion matrix and accuracy calculation
cm = confusion_matrix(y_test,y_pred)
print(cm)
accuracy = float(cm.diagonal().sum())/len(y_test)
print('model accuracy is:',accuracy*100,'%')
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap


data = pd.read_csv('C:\\Users\\jenim\\Downloads\\AI ML\\AI
ML\\apples_and_oranges.csv')
#print(data)
training_set,test_set = train_test_split(data,test_size=0.2,random_state=1)
#print("train:",training_set)
#print("test:",test_set)
x_train = training_set.iloc[:,0:2].values  # data
y_train = training_set.iloc[:,2].values  # target
x_test = test_set.iloc[:,0:2].values  # data
y_test = test_set.iloc[:,2].values  # target

# using labelencoder to convert string target value into no.
lb = LabelEncoder()
y_train = lb.fit_transform(y_train)
#print(y_train)

classifier = SVC(kernel='rbf',random_state=1,C=1,gamma='auto')
classifier.fit(x_train,y_train)

# visualizing the training data after model fitting
plt.figure(figsize=(7,7))
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 =
```

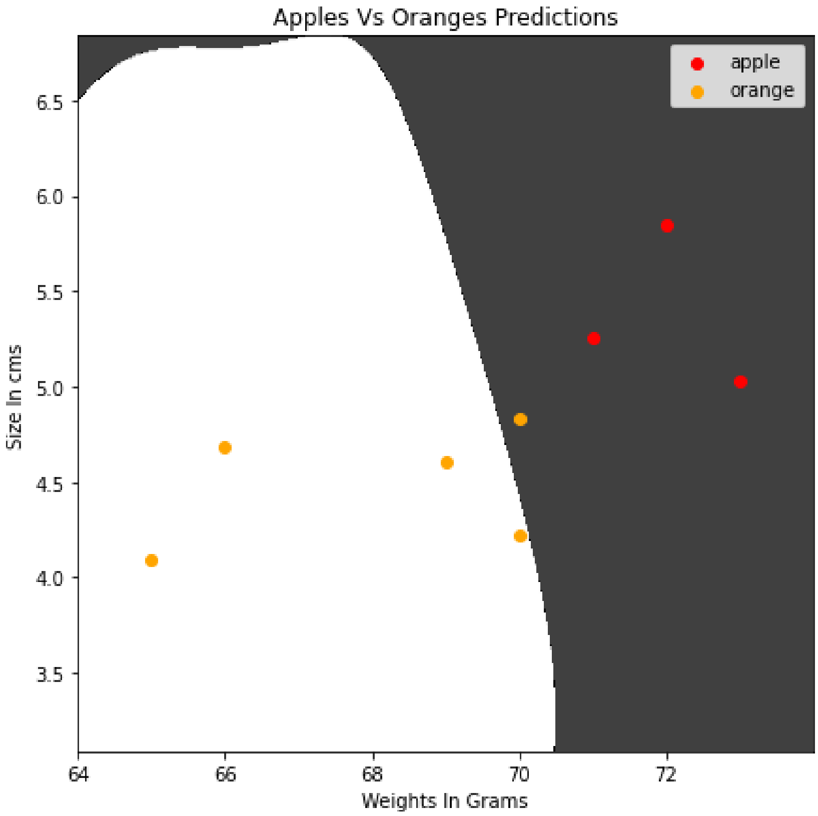
```
0.01),
                 np.arange(start = x_set[:,1].min()-1,stop = x_set[:,1].max()+1,step = 0.01))
plt.contourf(x1,x2,classifier.predict(np.array([x1.ravel(),x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75,cmap =
           ListedColormap(('black','white')))
plt.xlim(x1.min(),x1.max())
plt.ylim(x2.min(),x2.max())
for i,j in enumerate(np.unique(y_set)):
    plt.scatter(x_set[y_set == j,0],
              x_set[y_set == j,1],
              c =ListedColormap(('red','orange'))(i),
              label = j)

plt.title('Apples Vs Oranges')
plt.xlabel('Weights In Grams')
plt.ylabel('Size In cms')
plt.legend()
plt.show()

# visualizing the predictions
plt.figure(figsize=(7,7))
x_set,y_set = x_test,y_test
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))
plt.contourf(x1,x2,classifier.predict(np.array([x1.ravel(),x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75,cmap =
           ListedColormap(('black','white')))
plt.xlim(x1.min(),x1.max())
plt.ylim(x2.min(),x2.max())
for i,j in enumerate(np.unique(y_set)):
    plt.scatter(x_set[y_set == j,0],
              x_set[y_set == j,1],
              c =ListedColormap(('red','orange'))(i),
              label = j)

plt.title('Apples Vs Oranges Predictions')
plt.xlabel('Weights In Grams')
plt.ylabel('Size In cms')
plt.legend()
```

plt.show()

OUTPUT:

[[3 0]
 [1 4]]

model accuracy is: 87.5 %



RESULT:

Thus the above Program was successfully executed and the Output was obtained

Ex.No: H1 Date:	ENSEMBLE TECHNIQUE-VOTING CLASSIFER

AIM:

To develop a Program to implement Ensemble Voting classifier technique for IRIS dataset and find the accuracy score of Hard Voting and Soft Voting

PROGRAM:

```
# importing libraries
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

# loading iris dataset
iris = load_iris()
X = iris.data[:, :4]
Y = iris.target

# train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size = 0.20,random_state = 42)

# group / ensemble of models
estimator = []
estimator.append(('LR',
LogisticRegression(solver ='lbfgs',
multi_class ='multinomial',max_iter = 200)))
estimator.append(('SVC', SVC(gamma ='auto', probability = True)))
estimator.append(('DTC', DecisionTreeClassifier()))

# Voting Classifier with hard voting
vot_hard = VotingClassifier(estimators = estimator, voting ='hard')
vot_hard.fit(X_train, y_train)
y_pred = vot_hard.predict(X_test)

# using accuracy_score metric to predict accuracy
```

```
score = accuracy_score(y_test, y_pred)
print("Hard Voting Score % d" % score)

# Voting Classifier with soft voting
vot_soft = VotingClassifier(estimators = estimator, voting ='soft')
vot_soft.fit(X_train, y_train)
y_pred = vot_soft.predict(X_test)

# using accuracy_score
score = accuracy_score(y_test, y_pred)
print("Soft Voting Score % d" % score)
```

OUTPUT:

Hard Voting Score 1

Soft Voting Score 1

RESULT:

Thus the above Program was successfully executed and the Output was obtained

Ex.No: H2 **Date:**	**ENSEMBLE TECHNIQUE-AVERAGING**

AIM:

To develop a Program to implement Ensemble Averaging classifier technique and find the accuracy score of Averaging Classifier.

PROGRAM:

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import VotingClassifier
# get a list of base models
def get_models():
    models = list()
    models.append(('lr', LogisticRegression()))
    models.append(('cart', DecisionTreeClassifier()))
    models.append(('bayes', GaussianNB()))
    return models
# evaluate each base model
def evaluate_models(models, X_train, X_val, y_train, y_val):
    # fit and evaluate the models
    scores = list()
    for name, model in models:
        # fit the model
        model.fit(X_train, y_train)
        # evaluate the model
        yhat = model.predict(X_val)
        acc = accuracy_score(y_val, yhat)
        # store the performance
        scores.append(acc)
        # report model performance
    return scores
# define dataset
X, y = make_classification(n_samples=10000, n_features=20, n_informative=15,
n_redundant=5, random_state=7)
# split dataset into train and test sets
```

```
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, test_size=0.50, random_state=1)
# split the full train set into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full, test_size=0.33, random_state=1)
# create the base models
models = get_models()
# fit and evaluate each model
scores = evaluate_models(models, X_train, X_val, y_train, y_val)
print(scores)
# create the ensemble
ensemble = VotingClassifier(estimators=models, voting='soft', weights=scores)
# fit the ensemble on the training dataset
ensemble.fit(X_train_full, y_train_full)
# make predictions on test set
yhat = ensemble.predict(X_test)
# evaluate predictions
score = accuracy_score(y_test, yhat)
print('Weighted Avg Accuracy: %.3f' % (score*100))
# evaluate each standalone model
scores = evaluate_models(models, X_train_full, X_test, y_train_full, y_test)
for i in range(len(models)):
	print('>%s: %.3f' % (models[i][0], scores[i]*100))
# evaluate equal weighting
ensemble = VotingClassifier(estimators=models, voting='soft')
ensemble.fit(X_train_full, y_train_full)
yhat = ensemble.predict(X_test)
score = accuracy_score(y_test, yhat)
print('Voting Accuracy: %.3f' % (score*100))
```

OUTPUT:

```
[0.8896969696969697, 0.8642424242424243, 0.8812121212121212]
Weighted Avg Accuracy: 90.800
>lr: 87.800
>cart: 88.180
>bayes: 87.300
Voting Accuracy: 90.58
```

RESULT:

Thus the above Program was successfully executed and the Output was obtained.

Ex.No: I1 **Date:**	**K MEANS CLUSTERING ALGORITHM**

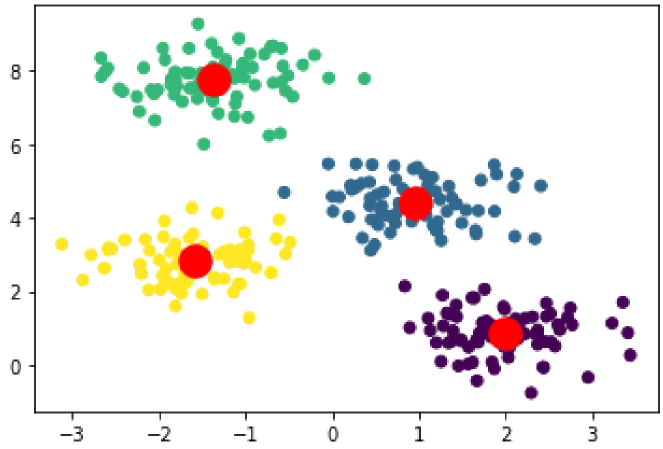
AIM:

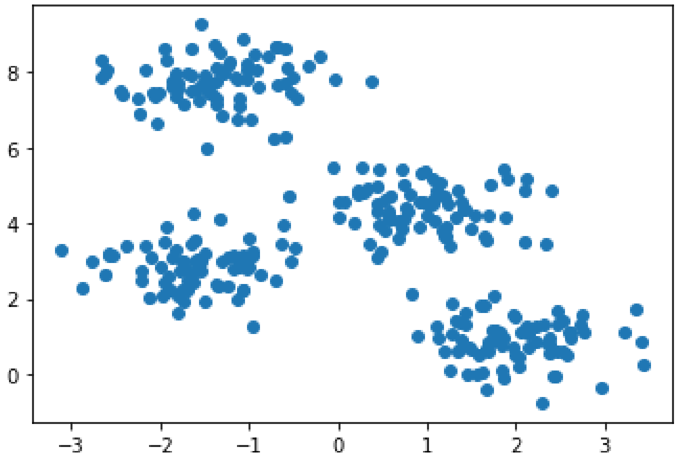
To develop a Program to build k-means clustering algorithm by generate isotropic Gaussian Clustering blobs and find the optimal number of clusters by using the Elbow Method.

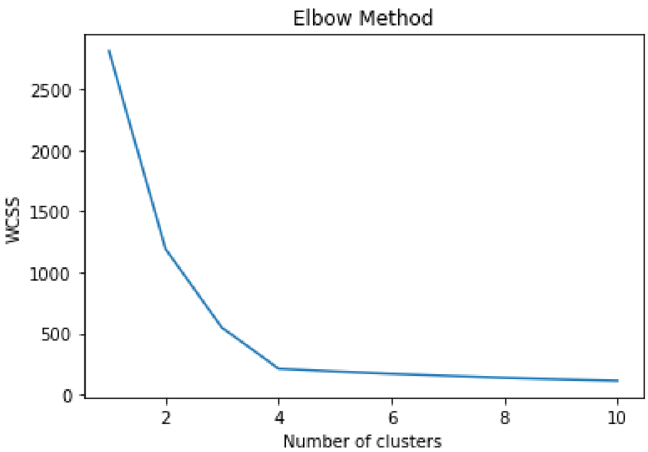
PROGRAM:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
#from sklearn.datasets.samples_generator import make_blobs
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], c=pred_y)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red')
plt.show()
```

OUTPUT:







RESULT:

Thus the above Program was successfully executed and the Output was obtained

Ex.No: I2 **Date:**	**K NEAREST NEIGHBOUR CLUSTERING ALGORITHM**

AIM:

To develop a Program to build k-nearest neighbour clustering algorithm by using classified dataset and find prescion, recall ,f1 score, support by using different number of clusters.

PROGRAM:

```
#K Nearest Neighbors with Python
#Import Libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

#Load the Data
df = pd.read_csv('C://Users//jenim//Downloads//AI ML//AI ML//ensemble//Classified
Data.csv',index_col=0)
df.head()
#Standardize the Variables
#Because the KNN classifier predicts the class of a given test observation
#by identifying the observations that are nearest to it, the scale of the
#variables matters. Any variables that are on a large scale will have a much
#larger effect on the distance between the observations, and hence on the KNN
#classifier, than variables that are on a small scale.
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df.drop('TARGET CLASS',axis=1))
scaled_features = scaler.transform(df.drop('TARGET CLASS',axis=1))
df_feat = pd.DataFrame(scaled_features,columns=df.columns[:-1])
df_feat.head()
#Train-Test Split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(scaled_features,df['TARGET CLASS'],
        test_size=0.30)
## Using KNN
#Remember that we are trying to come up with a model to predict whether someone
#will TARGET CLASS or not. We'll start with k=1.
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

```
knn.fit(X_train,y_train)
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
          pred = knn.predict(X_test)
#Predicting and evavluations
#Let's evaluate our knn model.
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_test,pred))
#chosing a K Value
#Let's go ahead and use the elbow method to pick a good K Value:
error_rate = []
for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed',
marker='o',markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
# FIRST A QUICK COMPARISON TO OUR ORIGINAL K=1
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train,y_train)
pred = knn.predict(X_test)
print('WITH K=1')
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
# NOW WITH K=23
knn = KNeighborsClassifier(n_neighbors=23)
knn.fit(X_train,y_train)
pred = knn.predict(X_test)
print('WITH K=23')
print('\n')
print(confusion_matrix(y_test,pred))
```

print('\n')
print(classification_report(y_test,pred))

OUTPUT:

WITH K=1

[[123 17]
 [12 148]]

	precision	recall	f1-score	support
0	0.91	0.88	0.89	140
1	0.90	0.93	0.91	160
accuracy			0.90	300
macro avg	0.90	0.90	0.90	300
weighted avg	0.90	0.90	0.90	300

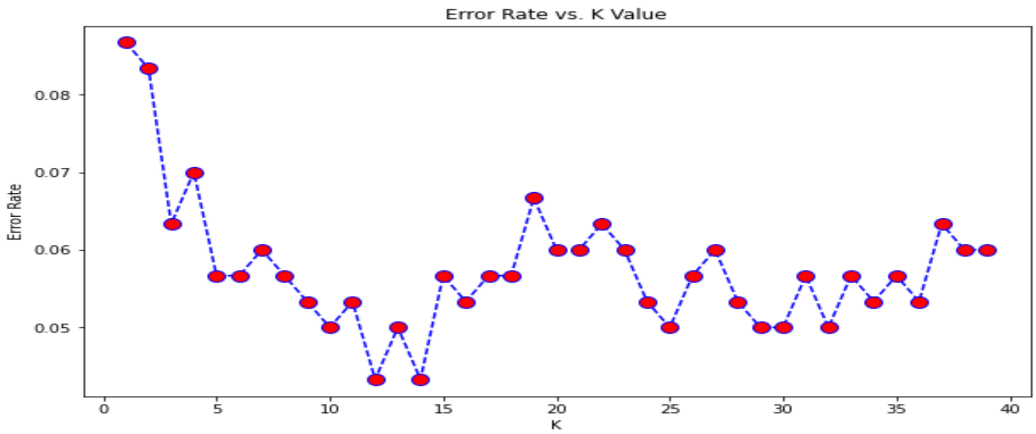
WITH K=23

[[128 12]
 [5 155]]

precision   recall f1-score   support

0   0.96   0.91   0.94   140

1
0.93
0.97
0.95
160



accuracy
0.94
300

macro avg
0.95   0.94   0.94   300

weighted avg   0.94   0.94   0.94   300

RESULT:

Thus the above Program was successfully executed and the Output was obtained

Ex.No: J Date:	**EXPECTATION MAXIMIZATION**

AIM:

To develop a Program to implement Expectation Maximization algorithm

PROGRAM:

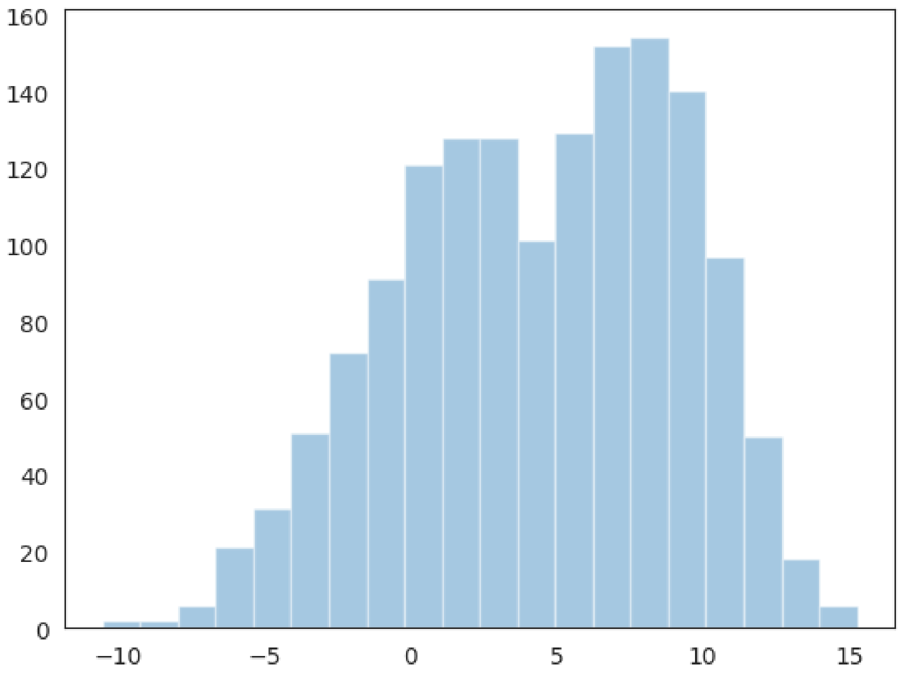
```
# For plotting
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("white")
%matplotlib inline
#for matrix math
import numpy as np
#for normalization + probability density function computation
from scipy import stats
#for data preprocessing
import pandas as pd
from math import sqrt, log, exp, pi
from random import uniform
random_seed=36788765
np.random.seed(random_seed)

Mean1 = 2.0  # Input parameter, mean of first normal probability distribution
Standard_dev1 = 4.0 #@param {type:"number"}
Mean2 = 9.0 # Input parameter, mean of second normal  probability distribution
Standard_dev2 = 2.0 #@param {type:"number"}

# generate data
y1 = np.random.normal(Mean1, Standard_dev1, 1000)
y2 = np.random.normal(Mean2, Standard_dev2, 500)
data=np.append(y1,y2)

# For data visiualisation calculate left and right of the graph
Min_graph = min(data)
Max_graph = max(data)
x = np.linspace(Min_graph, Max_graph, 2000) # to plot the data

print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("1", Mean1, Standard_dev1))
```

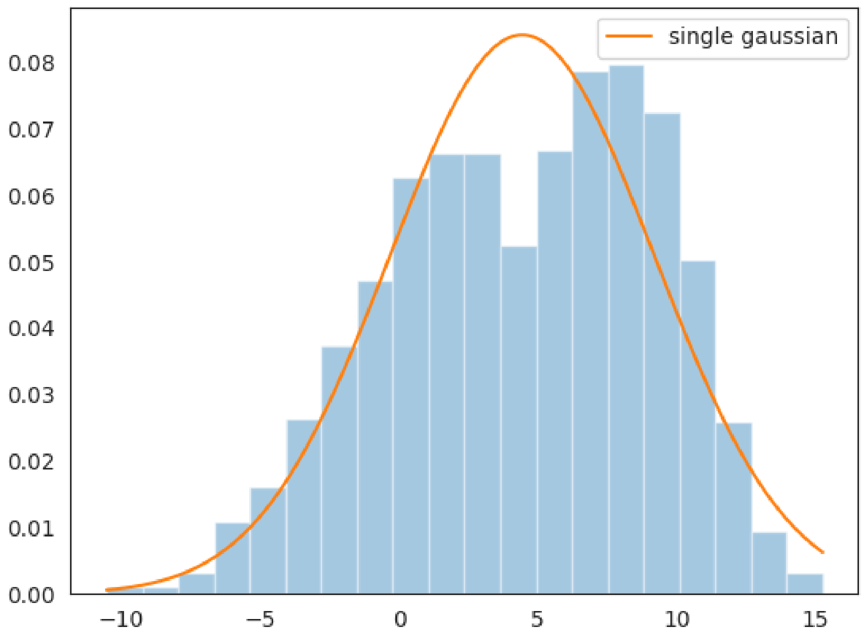
```
print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("2", Mean2, Standard_dev2))
sns.distplot(data, bins=20, kde=False);

class Gaussian:
    "Model univariate Gaussian"
    def __init__(self, mu, sigma):
        #mean and standard deviation
        self.mu = mu
        self.sigma = sigma

    #probability density function
    def pdf(self, datum):
        "Probability of a data point given the current parameters"
        u = (datum - self.mu) / abs(self.sigma)
        y = (1 / (sqrt(2 * pi) * abs(self.sigma))) * exp(-u * u / 2)
        return y

    def __repr__(self):
        return 'Gaussian({0:4.6}, {1:4.6})'.format(self.mu, self.sigma)
#gaussian of best fit
best_single = Gaussian(np.mean(data), np.std(data))
print('Best single Gaussian: μ = {:.2}, σ = {:.2}'.format(best_single.mu,
best_single.sigma))
#fit a single gaussian curve to the data
g_single = stats.norm(best_single.mu, best_single.sigma).pdf(x)
sns.distplot(data, bins=20, kde=False, norm_hist=True);
plt.plot(x, g_single, label='single gaussian');
plt.legend();
```

OUTPUT:

Input Gaussian 1: μ = 2.0, σ = 4.0

Input Gaussian 2: μ = 9.0, σ = 2.0





Best single Gaussian:
$\mu = 4.4$, $\sigma = 4.8$

RESULT:

Thus the above Program was successfully executed and the Output was obtained.

Ex.No: K **Date:**	**SEQUENTAIL NEURAL NETWORK MODEL**

AIM:

To develop a Program to build a sequential Neural Network Model.

PROGRAM:

```
from keras.models import Sequential
from keras.layers import Dense, Activation
import numpy as np
# Use numpy arrays to store inputs (x) and outputs (y):
x = np.array([[0,0], [0,1], [1,0], [1,1]])
y = np.array([[0], [1], [1], [0]])
# Define the network model and its arguments.
# Set the number of neurons/nodes for each layer:
model = Sequential()
model.add(Dense(2, input_shape=(2,)))
model.add(Activation('sigmoid'))
model.add(Dense(1))
model.add(Activation('sigmoid'))
# Compile the model and calculate its accuracy:
model.compile(loss='mean_squared_error', optimizer='sgd', metrics=['accuracy'])
# Print a summary of the Keras model:
model.summary()
```

OUTPUT:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 2)                 6

 activation (Activation)     (None, 2)                 0

 dense_1 (Dense)             (None, 1)                 3

 activation_1 (Activation)   (None, 1)                 0

=================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
_________________________________________________________________
```

RESULT:

Thus the above Program was successfully executed and the Output was obtained.

Ex.No: L **Date:**	**DEEP LEARNING NN MODEL**

AIM:

To develop a Program to build a Deep Learning NN model

PROGRAM:

```
import tensorflow as tf
#load training data and split into train and test sets
mnist = tf.keras.datasets.mnist
(x_train,y_train), (x_test,y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
model = tf.keras.models.Sequential([
                        tf.keras.layers.Flatten(input_shape=(28,28)),
                          tf.keras.layers.Dense(128,activation='relu'),
                          tf.keras.layers.Dropout(0.2),
                          tf.keras.layers.Dense(10)
])
#define loss function variable
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

#define optimizer,loss function and evaluation metric
model.compile(optimizer='adam',
          loss=loss_fn,
          metrics=['accuracy'])
#train the model
model.fit(x_train,y_train,epochs=5)
model.evaluate(x_test,y_test,verbose=2)

probability_model = tf.keras.Sequential([
                              model,
                              tf.keras.layers.Softmax()])
```

OUTPUT:

Epoch 1/5

1875/1875 [==============================] - 7s 3ms/step - loss: 0.2960 - accuracy: 0.9142

Epoch 2/5

1875/1875 [==============================] - 6s 3ms/step - loss: 0.1429 - accuracy: 0.9573

Epoch 3/5

1875/1875 [==============================] - 7s 4ms/step - loss: 0.1040 - accuracy: 0.9679

Epoch 4/5

1875/1875 [==============================] - 7s 4ms/step - loss: 0.0857 - accuracy: 0.9731

Epoch 5/5

1875/1875 [==============================] - 8s 4ms/step - loss: 0.0729 - accuracy: 0.9771

313/313 - 1s - loss: 0.0729 - accuracy: 0.9767 - 615ms/epoch - 2ms/step

EXECUTING AGAIN

Epoch 1/5

1875/1875 [==============================] - 7s 4ms/step - loss: 0.2965 - accuracy: 0.9139

Epoch 2/5

1875/1875 [==============================] - 8s 4ms/step - loss: 0.1430 - accuracy: 0.9572

Epoch 3/5

1875/1875 [==============================] - 8s 4ms/step - loss: 0.1080 - accuracy: 0.9679

Epoch 4/5

1875/1875 [==============================] - 7s 4ms/step - loss: 0.0878 - accuracy: 0.9730

Epoch 5/5

1875/1875 [==============================] - 7s 4ms/step - loss: 0.0740 - accuracy: 0.9764

313/313 - 1s - loss: 0.0721 - accuracy: 0.9789 - 716ms/epoch - 2ms/step

RESULT:

Thus the above Program was successfully executed and the Output was obtained