UNIVERSITY COLLEGE OF ENGINEERING NAGERCOIL

(A Constituent College of Anna University, Chennai)

Konam, Nagercoil - 629 004, Kanyakumari District.

LABORATORY RECORD

CS3491 ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

(REGULATION 2021)

NAME	: ______________________
REG NO	: ______________________
SEM	: ______________________
YEAR	: ______________________

UNIVERSITY COLLEGE OF ENGINEERING NAGERCOIL

(A Constituent College of Anna University, Chennai)

Konam, Nagercoil - 629 004, Kanyakumari District.

This to certify that this is the bonafide record of work done by Ms/Mr……………………………Register Number ………………………………….

of Sixth semester, Department of Electronics and Communication Engineering of this college in the CS3491 Artificial Intelligence And Machine Learning during 2023-2024 in partial fulfillment of the requirement of the B.E. degree course of the Anna University,Chennai.

Staff-in-charge **Head of the department**

This record is submitted for the University practical Examination held on………………

INTERNAL EXAMINER **EXTERNAL EXAMINER**

INDEX

S.NO.	DATE	EXPERIMENT TITLE	SIGN.
1			

Ex.No:1a	IMPLEMENTATION OF BASIC SEARCH STRATEGIES – BFS
Date :	

AIM:

To implement a python program for Breadth First Search (BFS).

Breadth-First Search

- Breadth-first search (BFS) is a traversing algorithm which starts from a selected node (source or starting node) and explores all of the neighbour nodes at the present depth before moving on to the nodes at the next level of depth.
- It must be ensured that each vertex of the graph is visited exactly once to avoid getting into an infinite loop with cyclic graphs or to prevent visiting a given node multiple times when it can be reached through more than one path.
- Breadth-first search can be implemented using a queue data structure, which follows the first-in-first-out (FIFO) method – i.e., the node that was inserted first will be visited first, and so on.

ALGORITHM:

Step 1: We start the process by considering any random node as the starting vertex.

Step 2: We enqueue (insert) it to the queue and mark it as visited.

Step 3: Then we mark and enqueue all of its unvisited neighbours at the current depth or continue to the next depth level if there is any.

Step 4: The visited vertices are removed from the queue.

Step 5:The process ends when the queue becomes empty.

PROGRAM:

```
graph={
    '5':['3','7'],
    '3':['2','4'],
    '7':['8'],
    '2':[],
    '4':['8'],
    '8':[]
}
visited =[]
queue=[]
def bfs(visited,graph,node):
    visited.append(node)
    queue.append(node)
    while queue:
        m=queue.pop(0)
        print(m, end="")
        for neighbour in graph[m]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)
print ("Following is the Breadth-First Search")
bfs(visited,graph,'5')
```

OUTPUT:

```
Following is the Breadth-First Search
537248
```

RESULT:

Thus the program for breadth-first search was implemented and executed successfully.

Ex.No:1b	**IMPLEMENTATION OF BASIC SEARCH STRATEGIES – DFS**
Date :	

AIM:

To implement a python code for Depth First Search (DFS)

ALGORITHM:

Step: 1 Pick any node. If it is unvisited, mark it as visited and recur on all its adjacent nodes.

Step: 2 Repeat until all the nodes are visited, or the node to be searched is found.

Step: 3 visited is a set that is used to keep track of visited nodes.

Step: 4 The dfs function is called and is passed the visited set, the graph in the form of a dictionary, and A, which is the starting node.

Step: 5 dfs follows the algorithm described above:

1. It first checks if the current node is unvisited - if yes, it is appended in the visited set.
2. Then for each neighbor of the current node, the dfs function is invoked again.
3. The base case is invoked when all the nodes are visited. The function then returns.

PROGRAM

```
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}

visited = set()

def dfs(visited, graph, node):  #function for dfs
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)


print("Following is the Depth-First Search")
dfs(visited, graph, '5')
```

OUTPUT:

```
Following is the Depth-First Search
5
3
2
4
8
7
```

RESULT:

Thus the program for depth-first search was implemented and executed successfully.

Ex.No:2a	**IMPLEMENTATION OF A* SEARCH ALGORITHM**
Date :	

AIM:

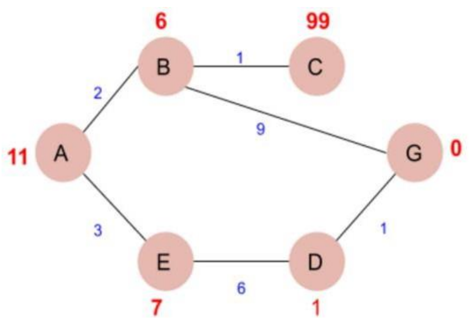
To implement a path finding using A* search algorithm.

A* SEARCH :

- A* search finds the shortest path through a search space to the goal state using the heuristic function.
- This technique finds minimal cost solutions and is directed to a goal state called A* search.
- The A* algorithm also finds the lowest-cost path between the start and goal state, where changing from one state to another requires some cost.

STEPS FOR SOLVING A* SEARCH

- Given the graph, find the cost-effective path from A to G. That is A is the source node and G is the goal node.



- Now from A, we can go to point B or E, so we compute f(x) for each of them,

$$A \rightarrow B = g(B) + h(B) = 2 + 6 = 8$$

$$A \rightarrow E = g(E) + h(E) = 3 + 7 = 10$$

- Since the cost for A → B is less, we move forward with this path and compute the f(x) for the children nodes of B.
- Now from B, we can go to point C or G, so we compute f(x) for each of them,

A → B → C = (2 + 1) + 99= 102

A → B → G = (2 + 9) + 0 = 11

- Here the path A → B → G has the least cost but it is still more than the cost of A → E, thus we explore this path further.
- Now from E, we can go to point D, so we compute f(x),

A → E → D = (3 + 6) + 1 = 10

- Comparing the cost of A → E → D with all the paths we got so far and as this cost is least of all we move forward with this path.
- Now compute the f(x) for the children of D

A → E → D → G = (3 + 6 + 1) +0 = 10

- Now comparing all the paths that lead us to the goal, we conclude that **A → E → D → G** is the most cost-effective path to get from A to G.

ALGORITHM:

// A* Search Algorithm

Step 1: Place the starting node into OPEN and find its f (n) value.

Step 2: Remove the node from OPEN, having the smallest f (n) value. If it is a goal node then stop and return success.

Step 3: Else remove the node from OPEN, find all its successors.

Step 4: Find the f (n) value of all successors; place them into OPEN and place the removed node into CLOSE.

Step 5: Go to Step-2.

Step 6: Exit.

PROGRAM:

```
def aStarAlgo(start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()
    g = {}               #store distance from starting node
    parents = {}         # parents contains an adjacency map of all nodes
    #distance of starting node from itself is zero
    g[start_node] = 0
    #start_node is root node i.e it has no parent nodes
    #so start_node is set to its own parent node
    parents[start_node] = start_node
    while len(open_set) > 0:
        n = None
        #node with lowest f() is found
        for v in open_set:
            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v
        if n == stop_node or Graph_nodes[n] == None:
            pass
        else:
            for (m, weight) in get_neighbors(n):
                #nodes 'm' not in first and last set are added to first
                #n is set its parent
                if m not in open_set and m not in closed_set:
                    open_set.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight
                #for each node m,compare its distance from start i.e g(m) to the
                #from start through n node
                else:
                    if g[m] > g[n] + weight:
                        #update g(m)
                        g[m] = g[n] + weight
                        #change parent of m to n
                        parents[m] = n
                        #if m in closed set,remove and add to open
                        if m in closed_set:
                            closed_set.remove(m)
                            open_set.add(m)
        if n == None:
            print('Path does not exist!')
            return None

        # if the current node is the stop_node
        # then we begin reconstructin the path from it to the start_node
        if n == stop_node:
            path = []
            while parents[n] != n:
                path.append(n)
                n = parents[n]
            path.append(start_node)
            path.reverse()
            print('Path found: {}'.format(path))
            return path
```

```
        # remove n from the open_list, and add it to closed_list
        # because all of his neighbors were inspected
        open_set.remove(n)
        closed_set.add(n)
    print('Path does not exist!')
    return None
#define fuction to return neighbor and its distance
#from the passed node
def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None
#for simplicity we ll consider heuristic distances given
#and this function returns heuristic distance for all nodes
def heuristic(n):
    H_dist = {
        'A': 11,
        'B': 6,
        'C': 5,
        'D': 7,
        'E': 3,
        'F': 6,
        'G': 5,
        'H': 3,
        'I': 1,
        'J': 0
    }
    return H_dist[n]
#Describe your graph here
Graph_nodes = {
    'A': [('B', 6), ('F', 3)],
    'B': [('A', 6), ('C', 3), ('D', 2)],
    'C': [('B', 3), ('D', 1), ('E', 5)],
    'D': [('B', 2), ('C', 1), ('E', 8)],
    'E': [('C', 5), ('D', 8), ('I', 5), ('J', 5)],
    'F': [('A', 3), ('G', 1), ('H', 7)],
    'G': [('F', 1), ('I', 3)],
    'H': [('F', 7), ('I', 2)],
    'I': [('E', 5), ('G', 3), ('H', 2), ('J', 3)],
}
aStarAlgo('A', 'J')
```

OUTPUT:

```
Path found: ['A', 'F', 'G', 'I', 'J']
```

RESULT:

Thus the program for A* search algorithm for path was implemented and executed successfully.

Ex.No:2b	**IMPLEMENTATION OF MEMORY BOUNDED A* ALGORITHM**
Date :	

AIM:

To implement memory bounded A* search for path finding problem.

Memory bounded A* Search:

- Memory Bounded A* is **a shortest path algorithm based on the A* algorithm**.
- The main advantage is that it uses a bounded memory, while the A* algorithm might need exponential memory. All other characteristics of are inherited from A*.
- This search is an optimal and complete algorithm for finding a least-cost path. Unlike A*, it will not run out of memory, unless the size of the shortest path exceeds the amount of space in available memory.

STEPS FOR MEMORY BOUND SEARCH

Step 1: Works like A* until memory is full

Step 2: When memory is full, drop the leaf node with the highest f-value (the worst leaf), keeping track of that worst value in the parent

Step 3: Complete if any solution is reachable

Step 4: Optimal if any optimal solution is reachable

Step 5: Otherwise, returns the best reachable solution

PROGRAM:

```
def aStarAlgo(start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()
    g = {}              #store distance from starting node
    parents = {}        # parents contains an adjacency map of all nodes
    #distance of starting node from itself is zero
    g[start_node] = 0
    #start_node is root node i.e it has no parent nodes
    #so start_node is set to its own parent node
    parents[start_node] = start_node
    while len(open_set) > 0:
        n = None
        #node with lowest f() is found
        for v in open_set:
            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v
        if n == stop_node or Graph_nodes[n] == None:
            pass
        else:
            for (m, weight) in get_neighbors(n):
                #nodes 'm' not in first and last set are added to first
                #n is set its parent
                if m not in open_set and m not in closed_set:
                    open_set.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight
                #for each node m,compare its distance from start i.e g(m) to the
                #from start through n node
                else:
                    if g[m] > g[n] + weight:
                        #update g(m)
                        g[m] = g[n] + weight
                        #change parent of m to n
                        parents[m] = n
                        #if m in closed set,remove and add to open
                        if m in closed_set:
                            closed_set.remove(m)
                            open_set.add(m)
        if n == None:
            print('Path does not exist!')
            return None

        # if the current node is the stop_node
        # then we begin reconstructin the path from it to the start_node
        if n == stop_node:
            path = []
            while parents[n] != n:
                path.append(n)
                n = parents[n]
            path.append(start_node)
            path.reverse()
            print('Path found: {}'.format(path))
```

```
            return path
        # remove n from the open_list, and add it to closed_list
        # because all of his neighbors were inspected
        open_set.remove(n)
        closed_set.add(n)
    print('Path does not exist!')
    return None

#define fuction to return neighbor and its distance
#from the passed node
def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None

#for simplicity we ll consider heuristic distances given
#and this function returns heuristic distance for all nodes
def heuristic(n):
    H_dist = {
        'A': 11,
        'B': 6,
        'C': 5,
        'D': 7,
        'E': 3,
        'F': 6,
        'G': 5,
        'H': 3,
        'I': 1,
        'J': 0
    }
    return H_dist[n]

#Describe your graph here
Graph_nodes = {
    'A': [('B', 6), ('F', 3)],
    'B': [('A', 6), ('C', 3), ('D', 2)],
    'C': [('B', 3), ('D', 1), ('E', 5)],
    'D': [('B', 2), ('C', 1), ('E', 8)],
    'E': [('C', 5), ('D', 8), ('I', 5), ('J', 5)],
    'F': [('A', 3), ('G', 1), ('H', 7)],
    'G': [('F', 1), ('I', 3)],
    'H': [('F', 7), ('I', 2)],
    'I': [('E', 5), ('G', 3), ('H', 2), ('J', 3)],
}
aStarAlgo('A', 'J')
```

Output:

```
Path found: ['A', 'F', 'G', 'I', 'J']
```

```
#for simplicity we ll consider heuristic distances given
#and this function returns heuristic distance for all nodes
def heuristic(n):
    H_dist = {
        'A': 11,
        'B': 6,
        'C': 99,
        'D': 1,
        'E': 7,
        'G': 0,
    }
    return H_dist[n]
#Describe your graph here
Graph_nodes = {
    'A': [('B', 2), ('E', 3)],
    'B': [('A', 2), ('C', 1), ('G', 9)],
    'C': [('B', 1)],
    'D': [('E', 6), ('G', 1)],
    'E': [('A', 3), ('D', 6)],
    'G': [('B', 9), ('D', 1)]
}
aStarAlgo('A', 'G')
```

Output:

```
Path found: ['A', 'E', 'D', 'G']
```

RESULT:

Thus the program for memory bounded A* search was implemented and executed successfully.

Ex.No:3	**IMPLEMENT NAIVE BAYES MODEL**
Date :	

AIM:

To implement a program for Naïve Bayes model

NAÏVE BAYES CLASSIFIER ALGORITHM

- Naive Bayes is among one of the very simple and powerful algorithms for classification based on Bayes Theorem with an assumption of independence among the predictors.
- The Naive Bayes classifier assumes that the presence of a feature in a class is not related to any other feature.
- Naive Bayes is a classification algorithm for binary and multi-class classification problems.

Bayes Theorem

- Based on prior knowledge of conditions that may be related to an event, Bayes theorem describes the probability of the event
- conditional probability can be found this way
- Assume we have a Hypothesis(*H*) and evidence(*E*),
- According to Bayes theorem, the relationship between the probability of Hypothesis before getting the evidence represented as *P(H)* and the probability of the hypothesis after getting the evidence represented as P(H|E) is:

  - $P(H|E) = P(E|H)*P(H)/P(E)$

STEPS INVOLVE NAÏVE BAYES ALGORITHM

Step 1: Handling Data

Data is loaded from the .csv file and spread into training and tested assets.

Step 2: Summarizing the data

Summarise the properties in the training data set to calculate the probabilities and make predictions.

Step 3: Making a Prediction

A particular prediction is made using a summarise of the data set to make a single prediction

Step 4: Making all the Predictions

Generate prediction given a test data set and a summarise data set.

Step 4: Evaluate Accuracy:

Accuracy of the prediction model for the test data set as a percentage correct out of them all the predictions made.

Step 4: Trying all together

Finally, we tie to all steps together and form our own model of Naive Bayes Classifier.

PROGRAM:

```
import pandas as pd

msg=pd.read_csv('naivetext.csv',names=['message','label'])

print('The dimensions of the dataset',msg.shape)

msg['labelnum']=msg.label.map({'pos':1,'neg':0})

X=msg.message

y=msg.labelnum

print(X)

print(y)


#splitting the dataset into train and test data

from sklearn.model_selection import train_test_split

xtrain,xtest,ytrain,ytest=train_test_split(X,y)

print ('\n The total number of Training Data :',ytrain.shape)

print ('\n The total number of Test Data :',ytest.shape)


#output of count vectoriser is a sparse matrix

from sklearn.feature_extraction.text import CountVectorizer

count_vect = CountVectorizer()
```

```
xtrain_dtm = count_vect.fit_transform(xtrain)

xtest_dtm=count_vect.transform(xtest)

print('\n The words or Tokens in the text documents \n')

print(count_vect.get_feature_names())

df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_feature_names
())


# Training Naive Bayes (NB) classifier on training data.

from sklearn.naive_bayes import MultinomialNB

clf = MultinomialNB().fit(xtrain_dtm,ytrain)

predicted = clf.predict(xtest_dtm)


#printing accuracy, Confusion matrix, Precision and Recall

from sklearn import metrics

print('\n Accuracy of the classifer is',

      metrics.accuracy_score(ytest,predicted))

print('\n Confusion matrix')

print(metrics.confusion_matrix(ytest,predicted))

print('\n The value of Precision' ,

metrics.precision_score(ytest,predicted))

print('\n The value of Recall' ,

metrics.recall_score(ytest,predicted))
```

Output:

```
The dimensions of the dataset (18, 2)
0 I love this sandwich
1 This is an amazing place
2 I feel very good about these beers
```

```
3 This is my best work
4 What an awesome view
5 I do not like this restaurant
6 I am tired of this stuff
7 I can't deal with this
8 He is my sworn enemy
9 My boss is horrible
10 This is an awesome place
11 I do not like the taste of this juice
12 I love to dance
13 I am sick and tired of this place
14 What a great holiday
15 That is a bad locality to stay
16 We will have good fun tomorrow
17 I went to my enemy's house today

Name: message, dtype: object
0 1
1 1
2 1
3 1
4 1
5 0
6 0
7 0
8 0
9 0
10 1
11 0
12 1
13 0
14 1
15 0
16 1
17 0
Name: labelnum, dtype: int64
The total number of Training Data: (13,)
The total number of Test Data: (5,)
The words or Tokens in the text documents
['about', 'am', 'amazing', 'an', 'and', 'awesome', 'beers', 'best', 'can', 'deal', 'do',
'enemy', 'feel',
'fun', 'good', 'great', 'have', 'he', 'holiday', 'house', 'is', 'like', 'love', 'my', 'not', 'of',
'place',
'restaurant', 'sandwich', 'sick', 'sworn', 'these', 'this', 'tired', 'to', 'today',
'tomorrow', 'very',
'view', 'we', 'went', 'what', 'will', 'with', 'work']
Accuracy of the classifier is 0.8
Confusion matrix
[[2 1]
[0 2]]
The value of Precision 0.6666666666666666

The value of Recall 1.0
```

RESULT:

Thus the program for naïve Bayes model was implemented and executed successfully.

Ex.No:4	**Implement Bayesian networks**
Date :	

AIM:

To write a program to construct a Bayesian network to diagnose heart disease.

ALGORITHM:

1. Read Cleveland Heart Disease data.
2. Display the data.
3. Display the Attributes names and datatyes.
4. Create Model- Bayesian Network.
5. Learn CPDs using Maximum Likelihood Estimators
6. Compute the Probability of HeartDisease given restecg.
7. computing the Probability of HeartDisease given cp.

Data set:heart.csv

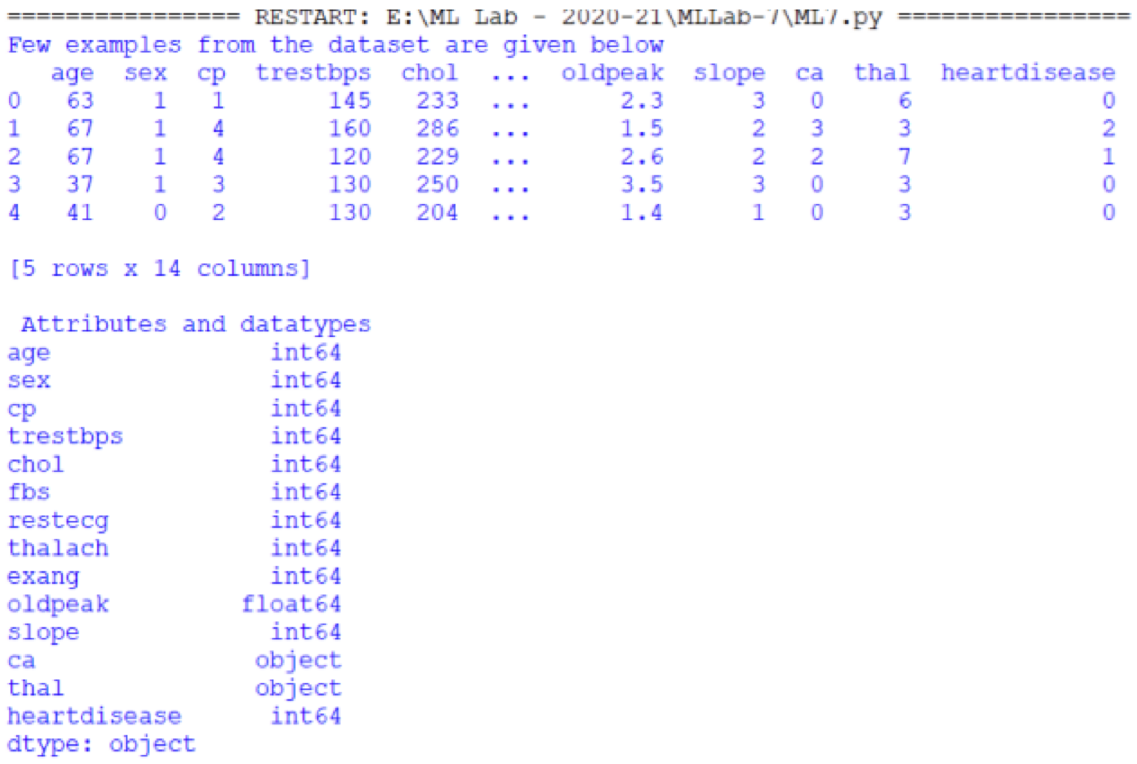
PROGRAM:

```
import numpy as np

import pandas as pd

import csv

from pgmpy.estimators import MaximumLikelihoodEstimator

from pgmpy.models import BayesianModel

from pgmpy.inference import VariableElimination

#read Cleveland Heart Disease data
```

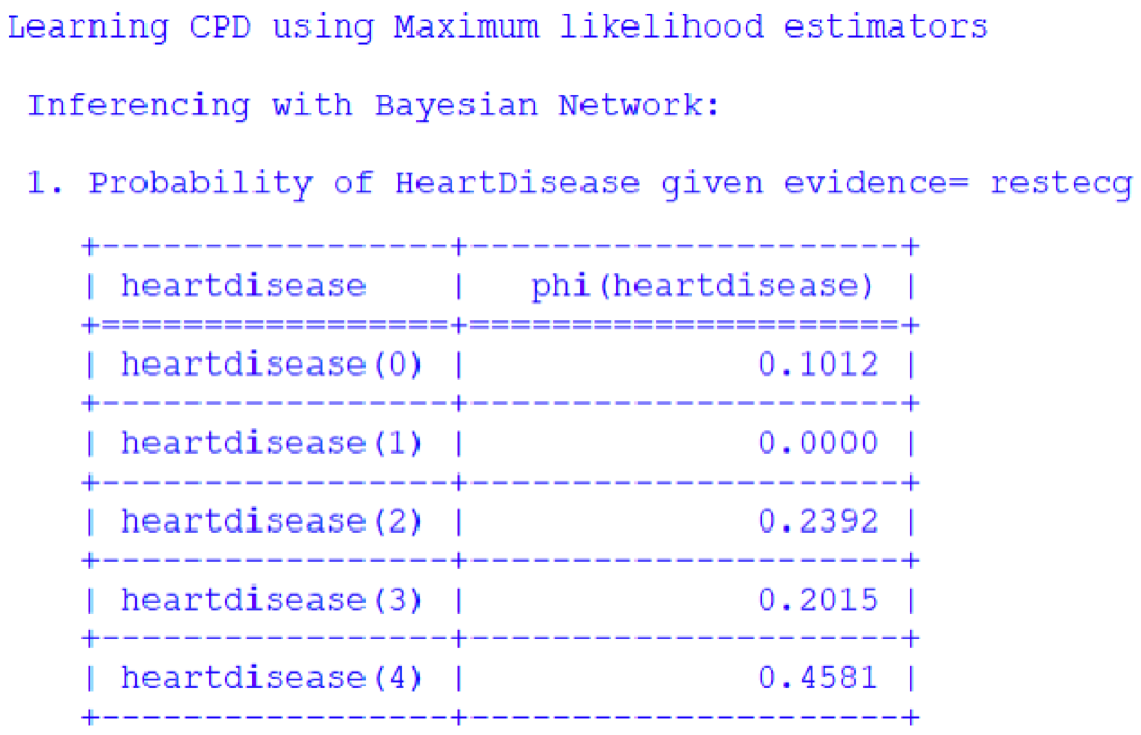
```
heartDisease = pd.read_csv('heart.csv')

heartDisease = heartDisease.replace('?',np.nan)

#display the data

print('Sample instances from the dataset are given below')

print(heartDisease.head())

#display the Attributes names and datatyes

print('\n Attributes and datatypes')

print(heartDisease.dtypes)

#Creat Model- Bayesian Network

model=BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heart
disease'),('cp','heartdisease'),('heartdisease','restecg'),('heartdisease','chol')])

#Learning CPDs using Maximum Likelihood Estimators

print('\n Learning CPD using Maximum likelihood estimators')

model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

# Inferencing with Bayesian Network

print('\n Inferencing with Bayesian Network:')

HeartDiseasetest_infer = VariableElimination(model)

#computing the Probability of HeartDisease given restecg

print('\n 1.Probability of HeartDisease given evidence=restecg :1')

q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evi

dence={'restecg':1})

print(q1)
```

```
#computing the Probability of HeartDisease given cp

print('\n 2.Probability of HeartDisease given evidence= cp:2 ')

q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})

print(q2)
```

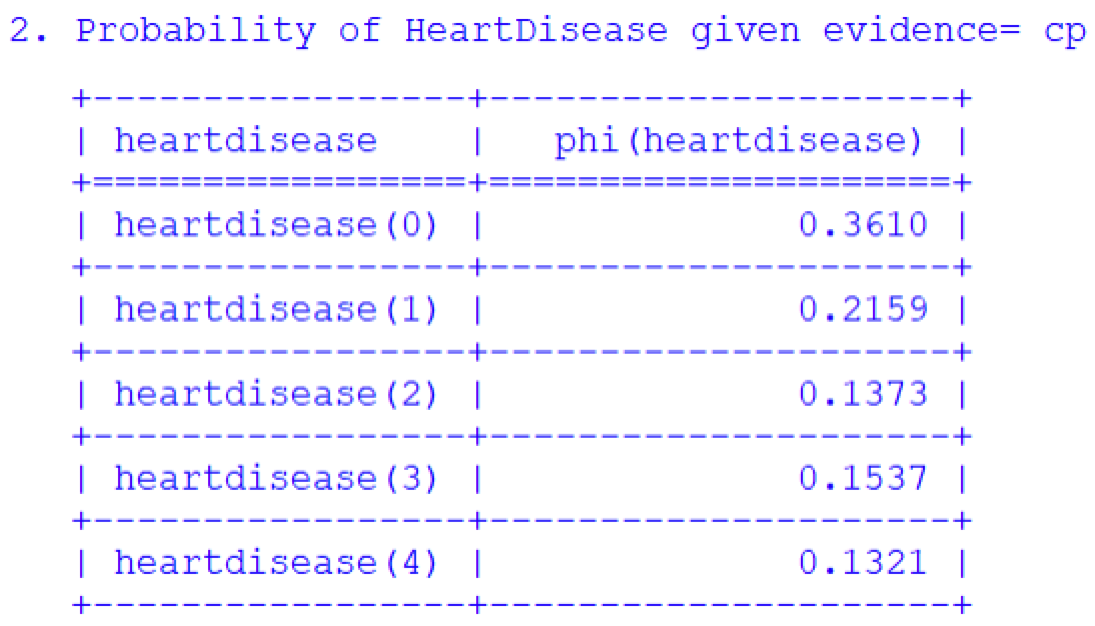
OUTPUT:



```
================ RESTART: E:\ML Lab - 2020-21\MLLab-7\ML7.py ================
Few examples from the dataset are given below
   age  sex  cp  trestbps  chol  ...  oldpeak  slope  ca  thal  heartdisease
0   63    1   1       145   233  ...      2.3      3   0     6             0
1   67    1   4       160   286  ...      1.5      2   3     3             2
2   67    1   4       120   229  ...      2.6      2   2     7             1
3   37    1   3       130   250  ...      3.5      3   0     3             0
4   41    0   2       130   204  ...      1.4      1   0     3             0

[5 rows x 14 columns]

 Attributes and datatypes
age                 int64
sex                 int64
cp                  int64
trestbps            int64
chol                int64
fbs                 int64
restecg             int64
thalach             int64
exang               int64
oldpeak           float64
slope               int64
ca                 object
thal               object
heartdisease        int64
dtype: object
```



```
Learning CPD using Maximum likelihood estimators

 Inferencing with Bayesian Network:

 1. Probability of HeartDisease given evidence= restecg

   +-----------------+----------------------+
   | heartdisease    |   phi(heartdisease)  |
   +=================+======================+
   | heartdisease(0) |               0.1012 |
   +-----------------+----------------------+
   | heartdisease(1) |               0.0000 |
   +-----------------+----------------------+
   | heartdisease(2) |               0.2392 |
   +-----------------+----------------------+
   | heartdisease(3) |               0.2015 |
   +-----------------+----------------------+
   | heartdisease(4) |               0.4581 |
   +-----------------+----------------------+
```



```
2. Probability of HeartDisease given evidence= cp
  +-----------------+---------------------+
  | heartdisease    |   phi(heartdisease) |
  +=================+=====================+
  | heartdisease(0) |              0.3610 |
  +-----------------+---------------------+
  | heartdisease(1) |              0.2159 |
  +-----------------+---------------------+
  | heartdisease(2) |              0.1373 |
  +-----------------+---------------------+
  | heartdisease(3) |              0.1537 |
  +-----------------+---------------------+
  | heartdisease(4) |              0.1321 |
  +-----------------+---------------------+
```

RESULT:

Thus the program to construct a Bayesian network was implemented and executed successfully.

Ex.No:5a	**Implement Regression models (Linear Regression)**
Date :	

AIM:

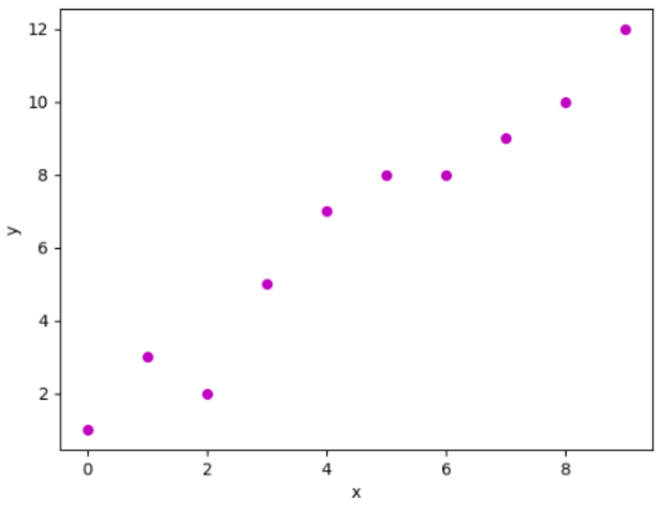
To write a program to implement linear for modeling relationships between a dependent variable with a given set of independent variables.

DEFINITION:

Let us consider a dataset where we have a value of response y for every feature x:

x	0	1	2	3	4	5	6	7	8	9
y	1	3	2	5	7	8	8	9	10	12

A scatter plot of the above dataset looks like:-



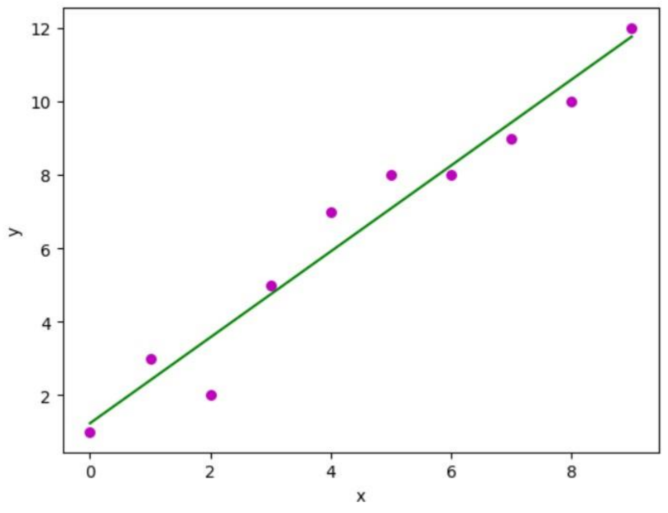
Now, the task is to find a **line that fits best** in the above scatter plot so that we can predict the response for any new feature values. This line is called a **regression line**.

PROGRAM:

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

def estimate_coef(x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x = np.mean(x)
    m_y = np.mean(y)

    # calculating cross-deviation and deviation about x
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x

    # calculating regression coefficients
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x

    return (b_0, b_1)

def plot_regression_line(x, y, b):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color = "m",
               marker = "o", s = 30)

    # predicted response vector
    y_pred = b[0] + b[1]*x

    # plotting the regression line
    plt.plot(x, y_pred, color = "g")

    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')

    # function to show plot
    plt.show()

def main():
    # observations / data
    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])

    # estimating coefficients
    b = estimate_coef(x, y)
    print("Estimated coefficients:\nb_0 = {}  \
          \nb_1 = {}".format(b[0], b[1]))

    # plotting regression line
    plot_regression_line(x, y, b)

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

OUTPUT:

```
Estimated coefficients:
b_0 = 1.2363636363636363
b_1 = 1.1696969696969697
```



RESULT:

Thus the python program to implement linear regression model was implemented and executed successfully.

Ex.No:5b	Implement Regression models (Logistic Regression)
Date :	

AIM:

To implement Logistic Regression using Python

ALGORITHM

- Import all the library function
- Import make_classification from sklearn datasets
- Generate Dataset for Logistic Regression
- Import pyplot from matplotlib
- Classify the Dataset based on the given features.

PROGRAM:

```
from sklearn.datasets import make_classification

from matplotlib import pyplot as plt

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix

import pandas as pd

# Generate and dataset for Logistic Regression

x, y = make_classification(

    n_samples=100,

    n_features=1,

    n_classes=2,

    n_clusters_per_class=1,

    flip_y=0.03,

    n_informative=1,

    n_redundant=0,
    n_repeated=0
)
print(x,y)
```

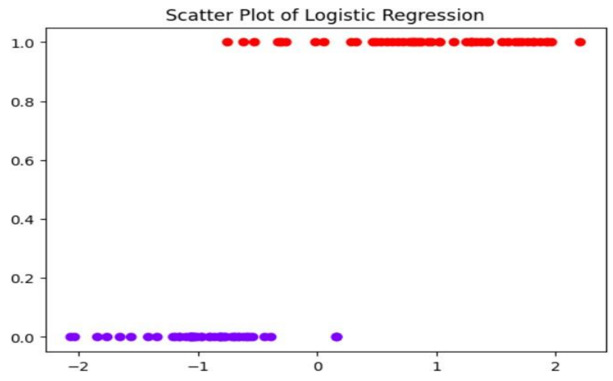
OUTPUT:

```
[[ 0.68072366]
 [-0.806672  ]
 [-0.25986635]
 [-0.96951576]
 [-1.55870949]
 [-0.71107565]
 [ 0.05858082]
 [-2.06472972]
 [-0.61592043]
 [ 1.25423915]
 [ 0.81852686]
 [-1.65141186]
 [-0.5894455 ]
 [ 1.02745431]
 [-0.32508896]
 [-0.53886171]
 [ 1.14821234]
 [ 0.87538478]
 [ 0.95887802]
 [ 1.30514551]
 [-1.02478688]
 [ 0.16563384]
 [ 0.77626036]
 [-1.00622251]
 [-0.55976575]
 [ 1.33550038]
 [ 1.60327317]
 [ 1.82115858]
 [-0.68603388]
 [ 1.8733355 ]
 [-0.52494619]
 [-2.03314002]
 [ 0.47001797]
 [ 1.55400671]
 [-1.34062378]
 [-0.38624537]
```

```
[-1.06339387]
[-1.41465045]
[ 0.58850401]
[ 0.80925135]
[-0.82066568]
[-0.01262654]
[-0.75104194]
[-1.09609801]
[-0.30652093]
[-0.6945338 ]
[-0.90156651]
[-0.96587756]
[ 0.53851931]
[ 0.16533166]
[-1.04609567]
[-1.15065139]
[-0.76739642]
[ 0.83776929]
[ 2.20562241]
[-0.80368921]
[-0.86160904]
[ 0.86032131]
[-0.65752318]
[ 1.81228279]
[-0.81507664]
[ 0.93532773]
[ 1.76874632]
[ 0.32893072]
[ 1.02960085]
[-1.84150254]
[ 0.16156709]
[-1.05944665]
[ 0.28788136]
[-1.05549933]
[ 1.37528673]
[ 1.66369265]
[ 1.71761177]
[ 1.96597594]
[-0.65315492]
[-0.29598263]
[-1.15345006]
[-1.03851861]
[ 1.69109822]
[ 1.92402678]
```

```
 [-0.89593983]
 [-0.58208549]
 [-1.18750595]
 [-1.06231671]
 [-0.79230653]
 [ 1.42147278]
 [ 1.2887393 ]
 [ 1.93706073]
 [-1.03110736]
 [-1.20543711]
 [ 0.79446549]
 [ 1.29599432]
 [ 0.49396915]
 [ 0.63241066]
 [ 0.72416825]
 [-1.76099355]
 [-0.61639759]
 [-0.43854548]
 [ 1.43886371]
 [-0.77167438]] [1 0 1 0 0 0 1 0 1 1 1 0 0 1 1 0 1 1
1 1 0 0 1 0 0 1 1 1 0 1 1 0 1 1 0 0 0
 0 1 1 0 1 1 0 1 0 0 0 1 0 0 0 0 1 1 0 0 1 0 1 0 1 1
1 1 0 0 0 1 0 1 1 1 1
 0 1 0 0 1 1 0 0 0 0 0 1 1 1 0 0 1 1 1 1 1 0 0 0 1 0]
```

# Create a scatter plot
plt.scatter(x, y, c=y, cmap='rainbow')
plt.title('Scatter Plot of Logistic Regression')
plt.show()



# Split the dataset into training and test dataset

x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=1)

```
x_train.shape
```

OUTPUT:

```
(75, 1)
```

```
log_reg=LogisticRegression()
log_reg.fit(x_train, y_train)
y_pred=log_reg.predict(x_test)
confusion_matrix(y_test, y_pred)
```

OUTPUT:

```
array([[12,  0],
       [ 2, 11]], dtype=int64)
```

RESULT:

Thus the python program to implement logistic regression model was implemented and executed successfully.

Ex.No:6a	**Implement Decision Tree**
Date :	

AIM:

To implement Decision Tree using python.

ALGORITHM

- Import Decision tree classifier from sklearn model
- Import train_test_split from sklearn.model
- Import accuracy_score from sklearn.metrics
- Import classification_report from sklearn.metrics
- Read the dataset values from the provided URL
- Print the dataset shape
- Print the dataset observation
- Separate the target variable
- Splitting the dataset into train and test
- Perform training with giniIndex
- Creating the classifier object
- Perform training with entropy
- Create Function to make prediction from the given dataset
- Create Function to calculate accuracy from the given dataset.

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
from sklearn.metrics import classification_report
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 dataset 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

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)*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 = 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:

```
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 python program to implement decision tree was implemented and executed successfully.

Ex.No:6b	Implement Random Forest
Date :	

AIM:

To Implement Random Forest using python.

ALGORITHM:

- Import Load digits from sklearn.datasets
- Import Random forest classifier from sklearn datasets
- Train the given dataset using Random Forest Classifier.
- Obtain the score from the trained dataset

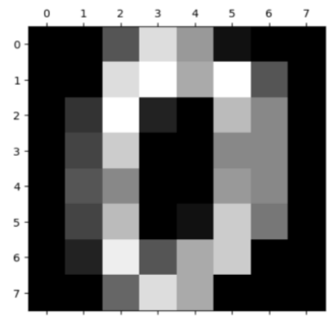
PROGRAM:

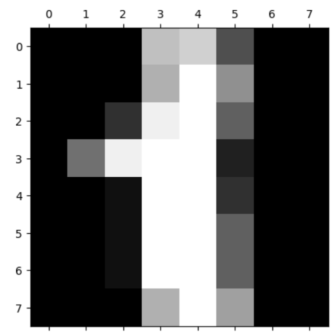
```
import pandas as pd

from sklearn.datasets import load_digits

digits = load_digits()

dir(digits)
```

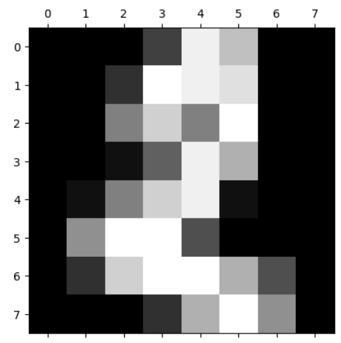
```
Out[2]: ['DESCR', 'data', 'feature_names', 'frame', 'images', 'target', 'target_names']
```

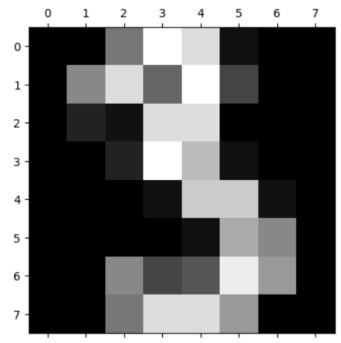
```
%matplotlib inline

import matplotlib.pyplot as plt

plt.gray()

for i in range(4):

    plt.matshow(digits.images[i])
```

```
<Figure size 640x480 with 0 Axes>
```



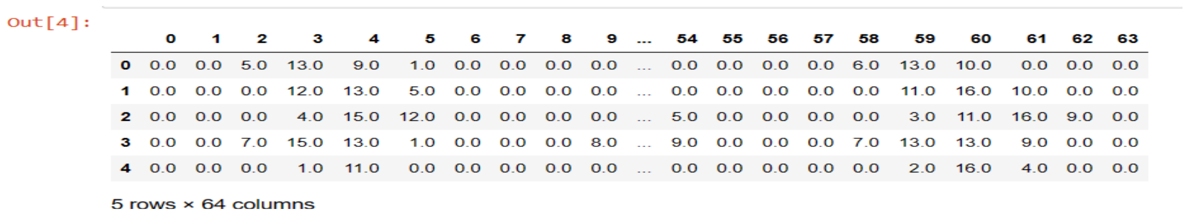






df = pd.DataFrame(digits.data)

df.head()



Out[4]:

	0	1	2	3	4	5	6	7	8	9	...	54	55	56	57	58	59	60	61	62	63
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	6.0	13.0	10.0	0.0	0.0	0.0
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	11.0	16.0	10.0	0.0	0.0
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	5.0	0.0	0.0	0.0	0.0	3.0	11.0	16.0	9.0	0.0
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	9.0	0.0	0.0	0.0	7.0	13.0	13.0	9.0	0.0	0.0
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	2.0	16.0	4.0	0.0	0.0

5 rows × 64 columns

df['target'] = digits.target

df[0:12]

X = df.drop('target',axis='columns')

y = df.target

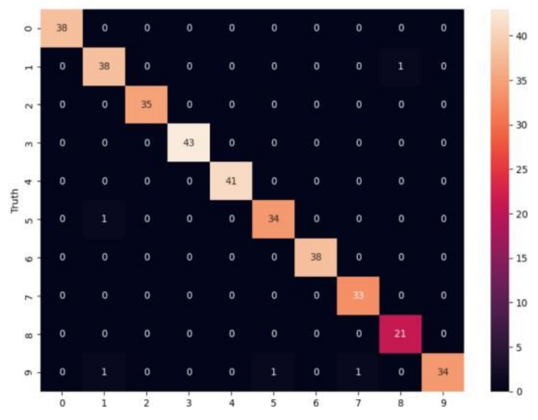
from sklearn.model_selection import train_test_split

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=20)
model.fit(X_train, y_train)
model.score(X_test, y_test)
```

```
Out[7]:  0.9861111111111112
```

```
y_predicted = model.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predicted)
cm
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sn
plt.figure(figsize=(10,7))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
Out[10]: Text(95.72222222222221, 0.5, 'Truth')
```



RESULT:

Thus the python program to implement random forest was implemented and executed successfully.

Ex.No:7	**Implement SVM Model**
Date :	

AIM:

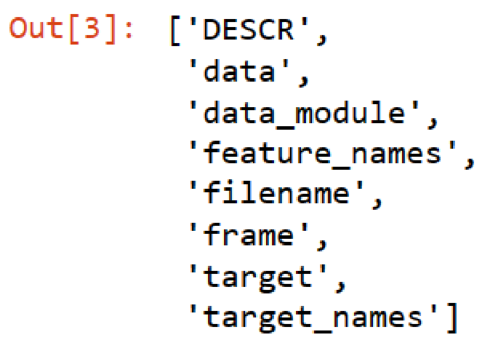
To implement SVM Model using python.

ALGORITHM

- From sklearn datasets import load_iris.
- Display the feature names from load_iris
- Import pyplot from from matplotlib
- Find the sepal length and sepal width from the given dataset
- Find the petal length and petal width from the trained dataset

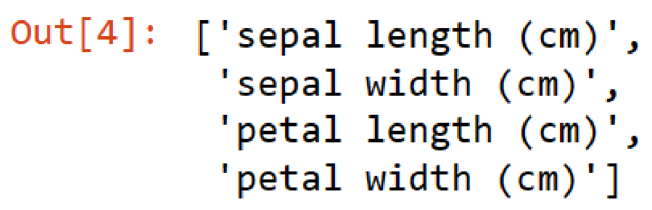
PROGRAM

```
import pandas as pd

from sklearn.datasets import load_iris

iris = load_iris()

dir(iris)
```



```
Out[3]: ['DESCR',
          'data',
          'data_module',
          'feature_names',
          'filename',
          'frame',
          'target',
          'target_names']
```

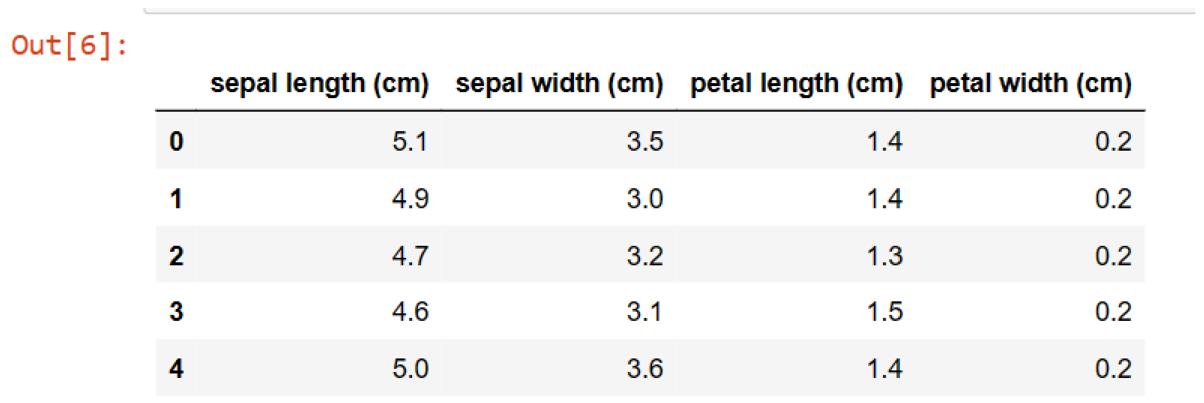
```
iris.feature_names
```



```
Out[4]: ['sepal length (cm)',
         'sepal width (cm)',
         'petal length (cm)',
         'petal width (cm)']
```

df=pd.DataFrame(iris.data, columns=iris.feature_names)

df.head()

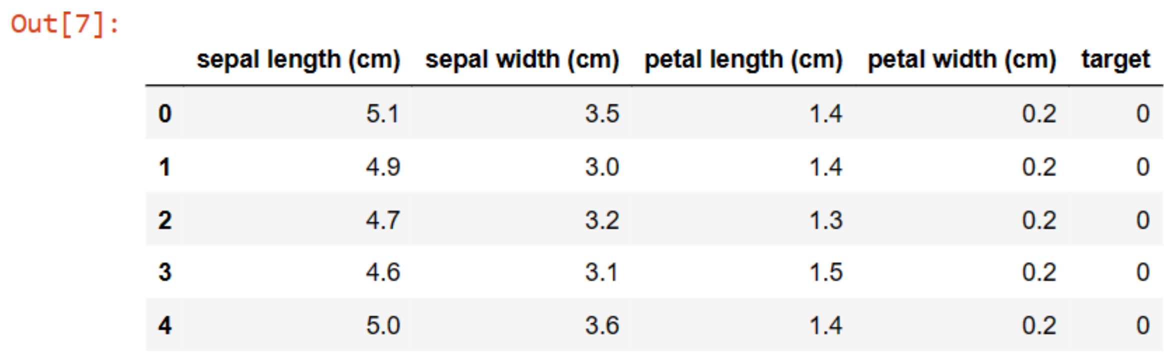


Out[6]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

df['target']=iris.target

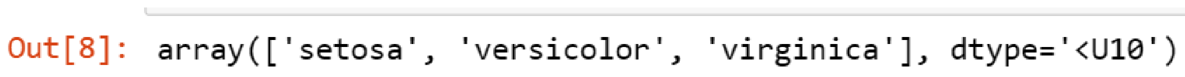
df.head()



Out[7]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

iris.target_names

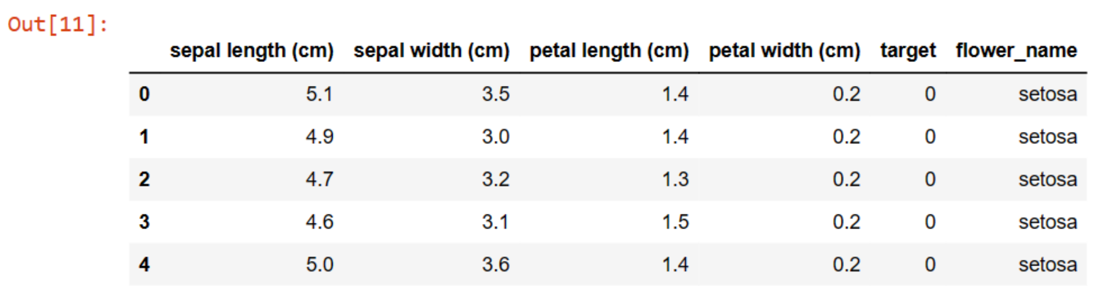


```
Out[8]: array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

df[df.target==2].head

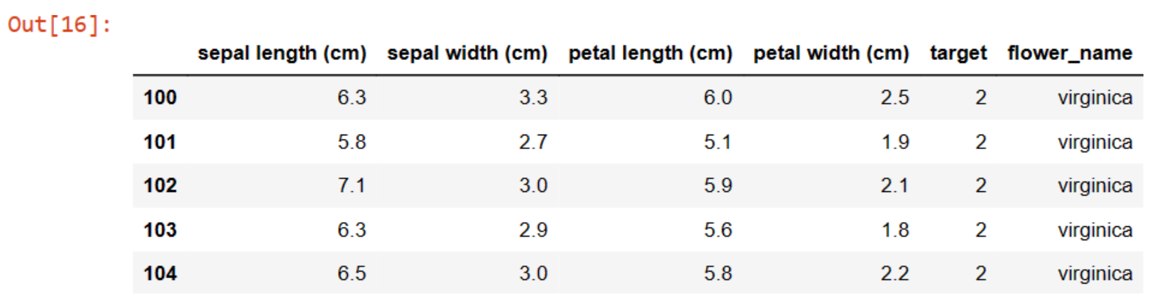
df['flower_name']=df.target.apply(lambda x:iris.target_names[x])

```
df.head()
```


Out[11]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

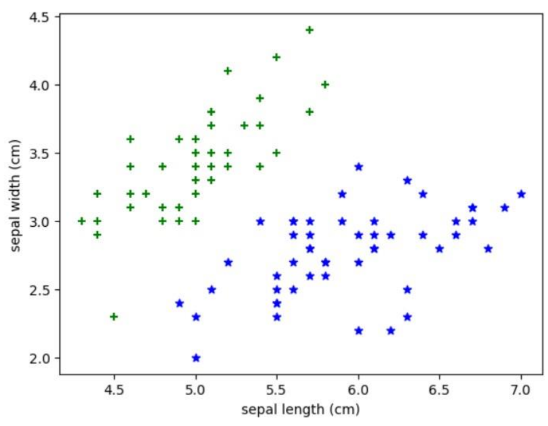
```
from matplotlib import pyplot as plt

%matplotlib inline

df0=df[df.target==0]

df1=df[df.target==1]

df2=df[df.target==2]

df2.head()
```


Out[16]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
100	6.3	3.3	6.0	2.5	2	virginica
101	5.8	2.7	5.1	1.9	2	virginica
102	7.1	3.0	5.9	2.1	2	virginica
103	6.3	2.9	5.6	1.8	2	virginica
104	6.5	3.0	5.8	2.2	2	virginica

```
plt.xlabel('sepal length (cm)')

plt.ylabel('sepal width (cm)')

plt.scatter(df0['sepal length (cm)'],df0['sepal width (cm)'],color='green',marker='+')

plt.scatter(df1['sepal length (cm)'],df1['sepal width (cm)'],color='blue',marker='*')
```

```
Out[21]: <matplotlib.collections.PathCollection at 0x1ca7eafcbb0>
```



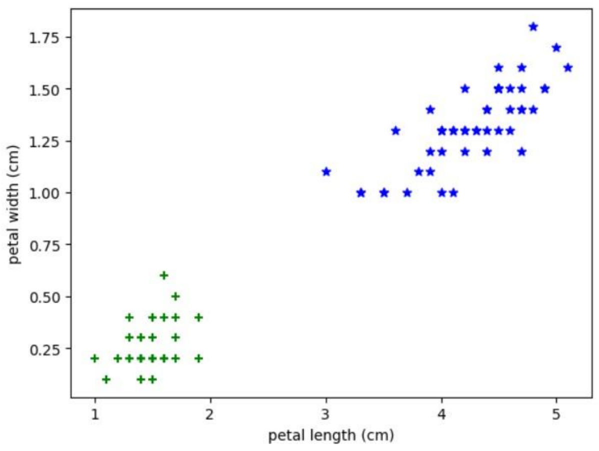
plt.xlabel('petal length (cm)')

plt.ylabel('petal width (cm)')

plt.scatter(df0['petal length (cm)'],df0['petal width (cm)'],color='green',marker='+')

plt.scatter(df1['petal length (cm)'],df1['petal width (cm)'],color='blue',marker='*')

```
Out[22]: <matplotlib.collections.PathCollection at 0x1ca7eb4ff10>
```



from sklearn.model_selection import train_test_split

x = df.drop(['target','flower_name'], axis='columns')

x.head()

```
Out[24]:
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
y=df.target

x_train, x_test, y_train, y_test= train_test_split(x,y,test_size=0.2)

len(x_train)
```

```
Out[29]: 120
```

```
len(x_test)
```

```
Out[30]: 30
```

```
from sklearn.svm import SVC

model = SVC(kernel='linear')

model.fit(x_train, y_train)
```

```
Out[38]:
▾ SVC
SVC(kernel='linear')
```

```
model.score(x_test, y_test)
```

```
Out[39]: 1.0
```

<u>RESULT:</u>

Thus the python program to implement SVM model was implemented and executed successfully.

Ex.No:8	Implement Ensembling Techniques(Bagging)
Date :	

AIM:

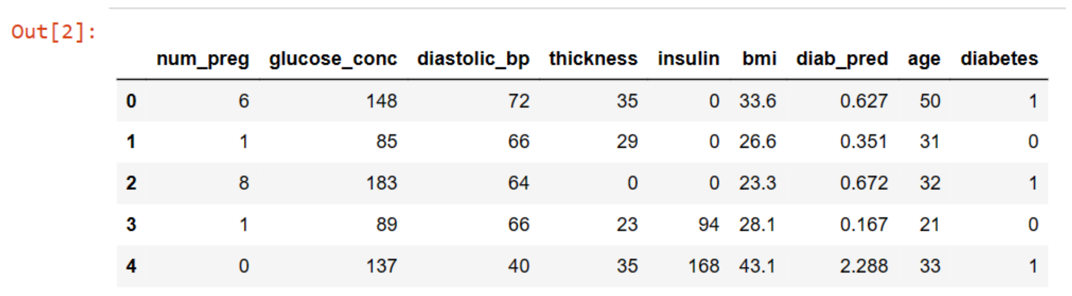
To implement Ensembling Techniques(Bagging) using python.

ALGORITHM

- ➢ Import the panda’s library
- ➢ Read the dataset from the path “C:/python/prima.csv”
- ➢ Display the first five rows from the dataframe using head() function.
- ➢ Returns the number of missing values in the dataset using isnull.sum() function.
- ➢ Preprocess the given dataset using Standard scalar function
- ➢ Find the cross value score using Decision tree classifier.

PROGRAM

```
import pandas as pd

df = pd.read_csv("pima-indians-diabetes.csv")

df.head()
```



Out[2]:

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	diabetes
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
df.isnull().sum()
```

```
Out[3]: num_preg        0
        glucose_conc    0
        diastolic_bp    0
        thickness       0
        insulin         0
        bmi             0
        diab_pred       0
        age             0
        diabetes        0
        dtype: int64
```

df.describe()

Out[4]:

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	diabetes
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

df.diabetes.value_counts()

```
Out[11]: 0    500
         1    268
         Name: diabetes, dtype: int64
```

X = df.drop("diabetes",axis="columns")

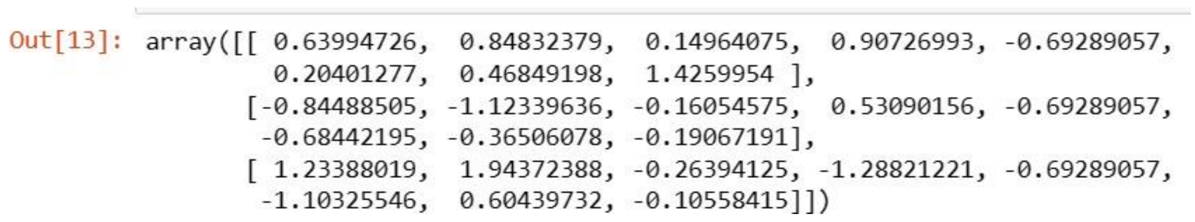
y = df.diabetes

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)

X_scaled[:3]

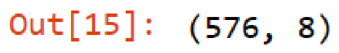


```
Out[13]: array([[ 0.63994726,  0.84832379,  0.14964075,  0.90726993, -0.69289057,
                  0.20401277,  0.46849198,  1.4259954 ],
                [-0.84488505, -1.12339636, -0.16054575,  0.53090156, -0.69289057,
                 -0.68442195, -0.36506078, -0.19067191],
                [ 1.23388019,  1.94372388, -0.26394125, -1.28821221, -0.69289057,
                 -1.10325546,  0.60439732, -0.10558415]])
```

from sklearn.model_selection import train_test_split

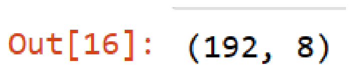
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, stratify=y, random_state=10)

X_train.shape



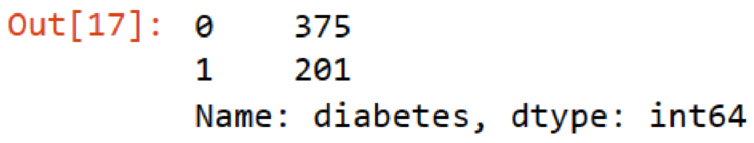
```
Out[15]: (576, 8)
```

X_test.shape



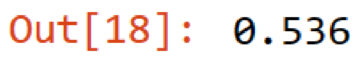
```
Out[16]: (192, 8)
```

y_train.value_counts()



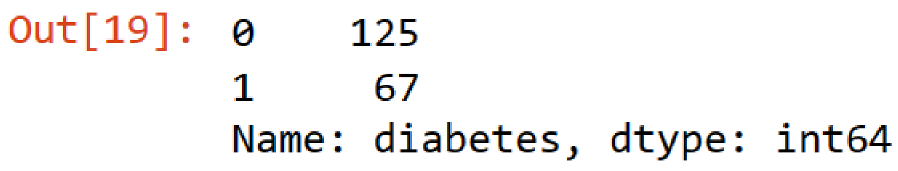
```
Out[17]: 0    375
         1    201
         Name: diabetes, dtype: int64
```

201/375



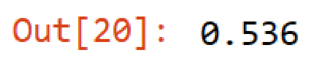
```
Out[18]: 0.536
```

y_test.value_counts()



```
Out[19]: 0    125
         1     67
         Name: diabetes, dtype: int64
```

67/125



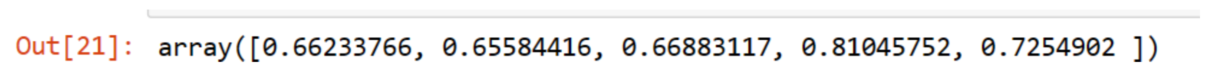
```
Out[20]: 0.536
```

from sklearn.model_selection import cross_val_score

from sklearn.tree import DecisionTreeClassifier

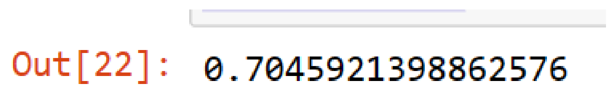
scores = cross_val_score(DecisionTreeClassifier(), X, y, cv=5)

scores



```
Out[21]: array([0.66233766, 0.65584416, 0.66883117, 0.81045752, 0.7254902 ])
```

scores.mean()



```
Out[22]: 0.7045921398862576
```

from sklearn.ensemble import BaggingClassifier

bag_model = BaggingClassifier(

    base_estimator=DecisionTreeClassifier(),

    n_estimators=100,

    max_samples=0.8,

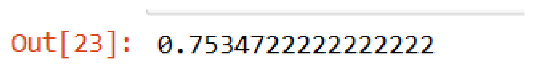
    oob_score=True,

    random_state=0

)

bag_model.fit(X_train, y_train)

bag_model.oob_score_



```
Out[23]: 0.7534722222222222
```

bag_model.score(X_test, y_test)

```
Out[24]:  0.7760416666666666
```

```
bag_model = BaggingClassifier(
    base_estimator=DecisionTreeClassifier(),
    n_estimators=100,
    max_samples=0.8,
    oob_score=True,
    random_state=0
)
scores = cross_val_score(bag_model, X, y, cv=5)
scores
```

```
Out[25]:  array([0.75324675, 0.72727273, 0.74675325, 0.82352941, 0.73856209])
```

```
scores.mean()
```

```
Out[26]:  0.7578728461081402
```

RESULT:

Thus the python program to implement Ensembling Techniques(Bagging)was implemented and executed successfully.

Ex.No: 9	**Implement Clustering Algorithms(KMeans)**
Date :	

AIM:

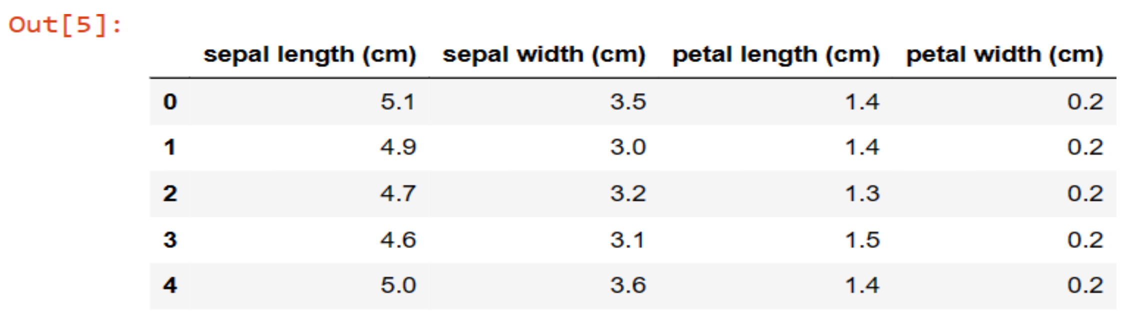
To implement clustering algorithm using python.

ALGORITHM:

- Import MinMaxScaler from sklearn preprocessing
- From sklearn datasets import load_iris
- Display the first five rows of the dataset using head function
- Apply Kmeans to the given dataset and find the septal length, septal width and petal length and petal width

PROGRAM

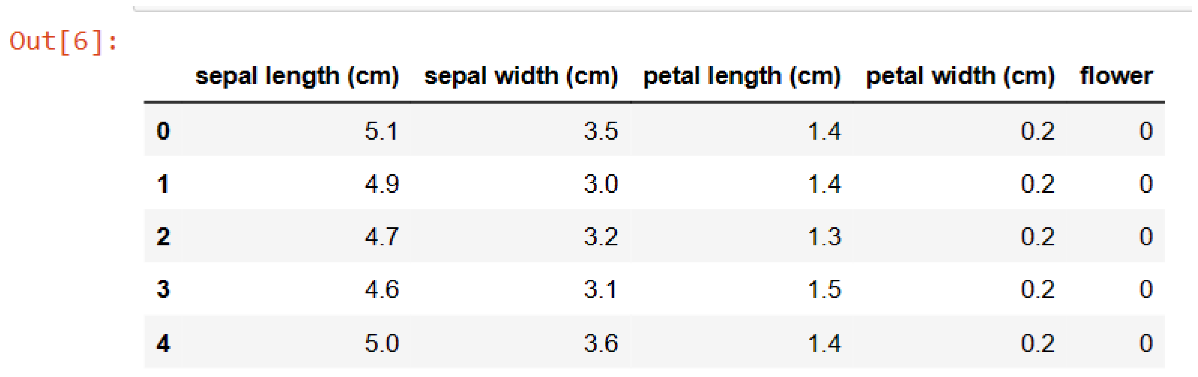
```
from sklearn.cluster import KMeans

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

from matplotlib import pyplot as plt

from sklearn.datasets import load_iris

%matplotlib inline

iris = load_iris()

df = pd.DataFrame(iris.data,columns=iris.feature_names)

df.head()
```



Out[5]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

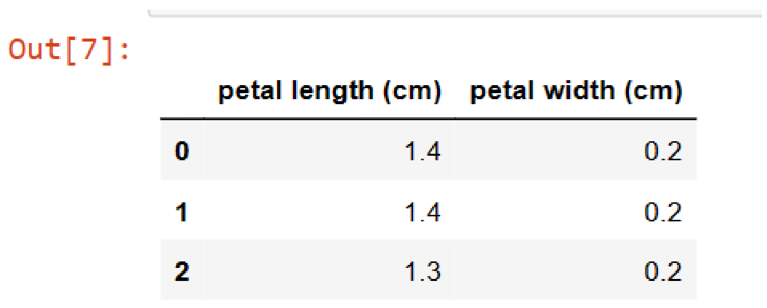
```
df['flower'] = iris.target

df.head()
```



Out[6]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	flower
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
df.drop(['sepal length (cm)', 'sepal width (cm)', 'flower'],axis='columns',inplace=True)

df.head(3)
```



Out[7]:

	petal length (cm)	petal width (cm)
0	1.4	0.2
1	1.4	0.2
2	1.3	0.2

```
km = KMeans(n_clusters=3)

yp = km.fit_predict(df)

yp
```

```
Out[8]: array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
               1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
               1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
               2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2,
               2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

df['cluster'] = yp

df.head(2)

Out[9]:

	petal length (cm)	petal width (cm)	cluster
0	1.4	0.2	1
1	1.4	0.2	1

df.cluster.unique()

```
Out[10]: array([1, 2, 0])
```

df1 = df[df.cluster==0]

df2 = df[df.cluster==1]

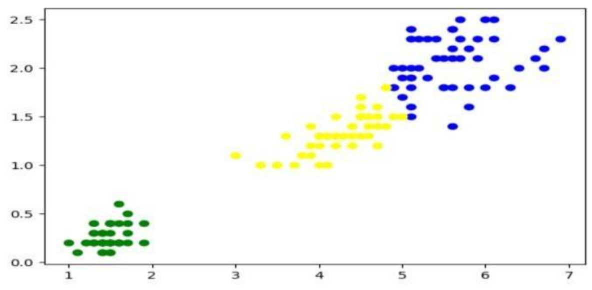
df3 = df[df.cluster==2]

plt.scatter(df1['petal length (cm)'],df1['petal width (cm)'],color='blue')

plt.scatter(df2['petal length (cm)'],df2['petal width (cm)'],color='green')

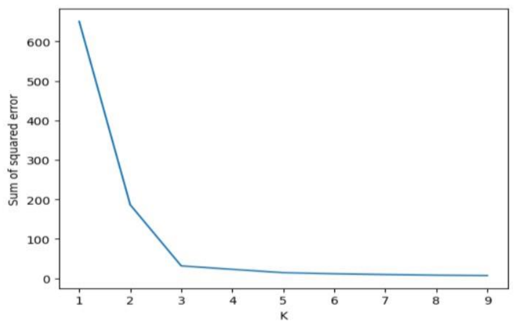
plt.scatter(df3['petal length (cm)'],df3['petal width (cm)'],color='yellow')

```
Out[12]: <matplotlib.collections.PathCollection at 0x1c34f975690>
```



```python
sse = []
k_rng = range(1,10)
for k in k_rng:
    km = KMeans(n_clusters=k)
    km.fit(df)
    sse.append(km.inertia_)
plt.xlabel('K')
plt.ylabel('Sum of squared error')
plt.plot(k_rng,sse)
```

```
Out[13]: [<matplotlib.lines.Line2D at 0x1c34fba9ea0>]
```



RESULT:

Thus the python program to implement clustering algorithm was implemented and executed successfully.

Ex.No: 10	**Implement EM for Bayesian networks**
Date :	

AIM:

To implement EM for Bayesian networks using python

ALGORITHM:

- **Initialize the parameters**: Start by initializing the parameters of the Bayesian network (e.g., probabilities in conditional probability tables).
- **E-step (Expectation)**:
    - Use the current parameter estimates and the observed data (and possibly incomplete data) to estimate the hidden or missing variables using probabilistic inference (like the forward-backward algorithm for hidden Markov models or sum-product algorithm for general Bayesian networks). Compute the expected values of the hidden variables given the current parameter estimates.
- **M-step (Maximization):**
    - Update the parameter estimates based on the expected values obtained in the E-step.
    - This involves maximizing the expected log-likelihood of the complete data with respect to the model parameters.
- **Repeat E-step and M-step:**
    - Alternate between E-step and M-step until convergence (e.g., when the change in parameter estimates falls below a predefined threshold or after a fixed number of iterations).

Program:

```
import numpy as np

# Simulated data generation for the Bayesian network

np.random.seed(42)

# True probabilities for our network

true_prob_A = 0.6

true_prob_B_given_A = np.array([[0.8, 0.2], [0.3, 0.7]])

# Generate observed data

sample_size = 1000

data_A = np.random.choice([0, 1], size=sample_size, p=[1-true_prob_A,
true_prob_A])

data_B = np.zeros(sample_size)

for i in range(sample_size):

    data_B[i] = np.random.choice([0, 1], p=true_prob_B_given_A[data_A[i]])

# EM algorithm for parameter estimation

def expectation_step(data_A, data_B, prob_A, prob_B_given_A):

    # E-step: Expectation

    # Compute the expected values of hidden variables (none here)

    return None

def maximization_step(data_A, data_B, hidden_variables):

    # M-step: Maximization

    # Update parameter estimates based on the observed and hidden data
    # Estimate probability of A
    prob_A = np.mean(data_A)
    # Estimate conditional probability of B given A
    prob_B_given_A = np.zeros((2, 2))
    for a in [0, 1]:
        data_B_given_A = data_B[data_A == a]
        prob_B_given_A[a] = [np.mean(data_B_given_A == 0), np.mean(data_B_given_A == 1)]
    return prob_A, prob_B_given_A
# EM iterations
# Initialize parameters
estimated_prob_A = 0.5
estimated_prob_B_given_A = np.array([[0.5, 0.5], [0.5, 0.5]])
# Perform EM iterations
num_iterations = 10
for i in range(num_iterations):
    hidden_vars = expectation_step(data_A, data_B, estimated_prob_A, estimated_prob_B_given_A)
    estimated_prob_A, estimated_prob_B_given_A = maximization_step(data_A, data_B, hidden_vars)
# Print the estimated parameters
print("Estimated probability of A:", estimated_prob_A)
print("Estimated conditional probability of B given A:")

print(estimated_prob_B_given_A)
```

OUTPUT:

```
Estimated probability of A: 0.579

Estimated conditional probability of B given A:

[[0.7719715 0.2280285]

 [0.2970639 0.7029361]]
```

RESULT:

Thus the python program to implement EM for Bayesian network was implemented and executed successfully.

Ex.No: 11	**Build simple Neural Network**
Date :	

AIM:

To implement simple neural network using python

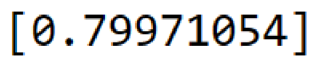
ALGORITHM

- Define the activation function
- Train the input values and obtain the output from the given dataset.
- Test the given dataset from the output obtained from the given dataset
- Obtain the forward and Backward pass from the trained dataset

PROGRAM

```
# importing dependancies

import numpy as np

# The activation function

def activation(x):

    return 1 / (1 + np.exp(-x))

weights = np.random.uniform(-1,1,size = (2, 1))


training_inputs = np.array([[0, 0, 1, 1, 0, 1]]).reshape(3, 2)

training_outputs = np.array([[0, 1, 1]]).reshape(3,1)


for i in range(15000):
    # forward pass

    dot_product = np.dot(training_inputs, weights)

    output = activation(dot_product)

    # backward pass.

    temp2 = -(training_outputs - output) * output * (1 - output)

    adj = np.dot(training_inputs.transpose(), temp2)

    # 0.5 is the learning rate.

    weights = weights - 0.5 * adj


# The testing set

test_input = np.array([1, 0])

test_output = activation(np.dot(test_input, weights))

# OR of 1, 0 is 1

print(test_output)
```

OUTPUT:



```
[0.79971054]
```

RESULT:

Thus the python program to implement neural network was implemented and executed successfully.

Ex.No: 12	**Build Deep Learning Neural Network model**
Date :	

AIM:

To build a deep learning neural network model using python.

ALGORITHM

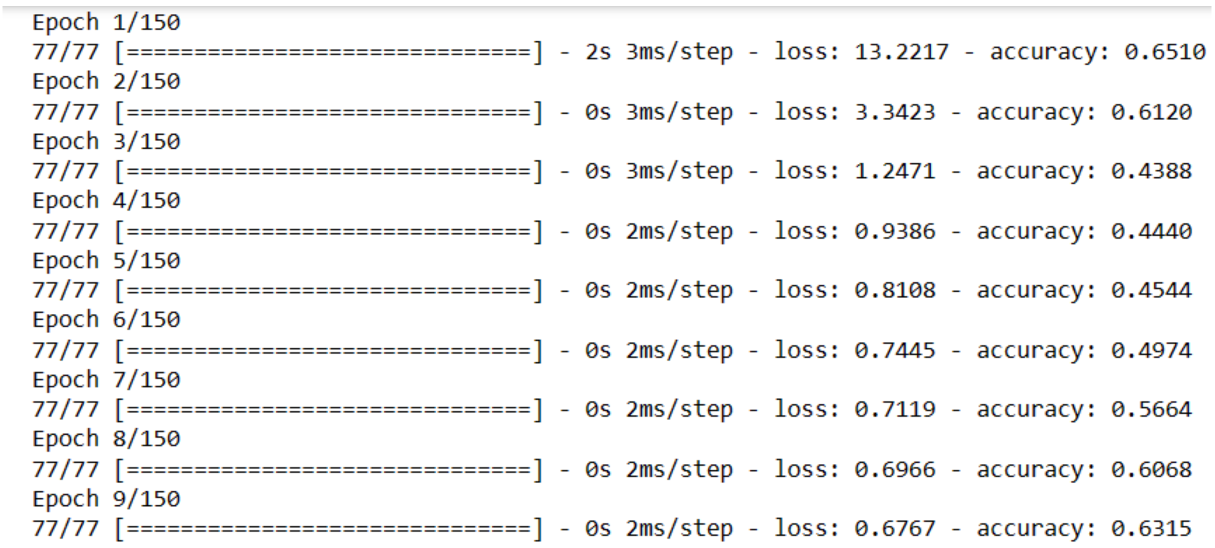
- Load data from the test file using import loadtxt
- Import sequential from tensorflow
- Import Dense from tensor flow
- Load data from from the test file from the path 'C:/python/pima-indians-diabetes.csv', delimiter=','
- Split the given dataset into input and output variables.
- Fit the keras model on the given dataset.

PROGRAM

```
from numpy import loadtxt

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

dataset = loadtxt('C:/python/pima-indians-diabetes.csv', delimiter=',')

# split into input (X) and output (y) variables

X = dataset[:,0:8]

y = dataset[:,8]

# define the keras model
```

```
model = Sequential()

model.add(Dense(12, input_shape=(8,), activation='relu'))

model.add(Dense(8, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

# compile the keras model

model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

# fit the keras model on the dataset

model.fit(X, y, epochs=150, batch_size=10)
```

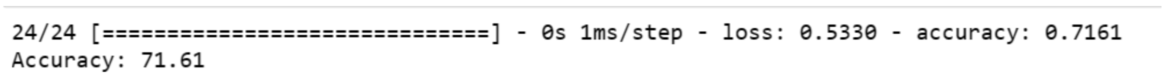
<u>OUTPUT:</u>



```
Epoch 1/150
77/77 [==============================] - 2s 3ms/step - loss: 13.2217 - accuracy: 0.6510
Epoch 2/150
77/77 [==============================] - 0s 3ms/step - loss: 3.3423 - accuracy: 0.6120
Epoch 3/150
77/77 [==============================] - 0s 3ms/step - loss: 1.2471 - accuracy: 0.4388
Epoch 4/150
77/77 [==============================] - 0s 2ms/step - loss: 0.9386 - accuracy: 0.4440
Epoch 5/150
77/77 [==============================] - 0s 2ms/step - loss: 0.8108 - accuracy: 0.4544
Epoch 6/150
77/77 [==============================] - 0s 2ms/step - loss: 0.7445 - accuracy: 0.4974
Epoch 7/150
77/77 [==============================] - 0s 2ms/step - loss: 0.7119 - accuracy: 0.5664
Epoch 8/150
77/77 [==============================] - 0s 2ms/step - loss: 0.6966 - accuracy: 0.6068
Epoch 9/150
77/77 [==============================] - 0s 2ms/step - loss: 0.6767 - accuracy: 0.6315
```

```
# evaluate the keras model

_, accuracy = model.evaluate(X, y)

print('Accuracy: %.2f' % (accuracy*100))
```

<u>OUTPUT:</u>



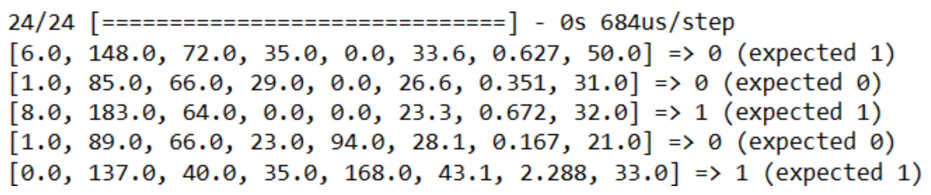
```
24/24 [==============================] - 0s 1ms/step - loss: 0.5330 - accuracy: 0.7161
Accuracy: 71.61
```

```
model.fit(X, y, epochs=150, batch_size=10, verbose=0)

# evaluate the keras model

_, accuracy = model.evaluate(X, y, verbose=0)

# make probability predictions with the model

predictions = model.predict(X)

# round predictions

rounded = [round(x[0]) for x in predictions]

# make class predictions with the model

predictions = (model.predict(X) > 0.5).astype(int)
```

```
24/24 [==============================] - 0s 3ms/step
```

```
from numpy import loadtxt

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# load the dataset

dataset = loadtxt('C:/python/pima-indians-diabetes.csv', delimiter=',')

# split into input (X) and output (y) variables

X = dataset[:,0:8]

y = dataset[:,8]

# define the keras model

model = Sequential()

model.add(Dense(12, input_shape=(8,), activation='relu'))

model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# compile the keras model

model.compile(loss='binary_crossentropy',optimizer='adam',
metrics=['accuracy'])

# fit the keras model on the dataset

model.fit(X, y, epochs=150, batch_size=10, verbose=0)

# make class predictions with the model

predictions = (model.predict(X) > 0.5).astype(int)

# summarize the first 5 cases

for i in range(5):

 print('%s => %d (expected %d)' % (X[i].tolist(), predictions[i], y[i]))
```

OUTPUT:



```
24/24 [==============================] - 0s 684us/step
[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0] => 0 (expected 1)
[1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0] => 0 (expected 0)
[8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0] => 1 (expected 1)
[1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0] => 0 (expected 0)
[0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0] => 1 (expected 1)
```

RESULT:

Thus the python program to build deep learning neural network model was implemented and executed successfully..