GOVERNMENT COLLEGE OF ENGINEERING

(Affiliated to Anna University, Chennai)

THANJAVUR-613402

BONAFIDE CERTIFICATE

This is a bonafide record of the practical work done by ………………………………Register Number………………………………, **SIXTH SEMESTER** B.E.(ECE) in the **CS3491 – ARTIFICIAL INTELLIGENCE & MACHINE LEARNING LABORATORY,** during the academic year **2023 _ 2024** [EVEN].

DATE:

Signature of the staff in-charge **Signature of Head of the Department**

Internal Examiner **External Examiner**

INDEX

Ex.No:	Date	Name of the Experiment	Page. No	Mark (out of 100)	Signature

Ex.No:1(a)	UNINFORMED SEARCH ALGORITHM (BREADTH FIRST SEARCH)
Date:	

AIM:

To implement uninformed search algorithms such as BFS and DFS.

ALGORITHM:

STEP 1: SET STATUS = 1 (ready state) for each node in G

STEP 2: Enqueue the starting node A and set its STATUS = 2 (waiting state)

STEP 3: Repeat Steps 4 and 5 until QUEUE is empty

STEP 4: Dequeue a node N. Process it and set its STATUS = 3 (processed state).

STEP 5: Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state) [END OF LOOP]

STEP 6: EXIT

PROGRAM(BFS):

```
from collections import defaultdictclass
Graph:
    def __init__(self):
        self.graph = defaultdict(list)def
    addEdge(self,u,v):
        self.graph[u].append(v)def
    BFS(self, s):
        visited = [False] * (len(self.graph))
        queue = []
        queue.append(s)
        visited[s] = True
        while queue:
            s = queue.pop(0) print
            (s, end = " ") for i in
            self.graph[s]:
                if visited[i] == False:
```

```
                    queue.append(i)
                    visited[i] = True
g = Graph()
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)
print ("Following is Breadth First Traversal" " (starting from vertex 2)")
g.BFS(2)
```

OUTPUT:

Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1

RESULT:

Thus the uninformed search algorithms such as BFS have been executed successfully and the output got verified.

Ex.No:1(b)	UNINFORMED SEARCH ALGORITHM (DEPTH FIRST SEARCH)
Date:	

AIM:

To implement uninformed search algorithms such as BFS and DFS

ALGORITHM(DFS):

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Push the starting node A on the stack and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until STACK is empty

Step 4: Pop the top node N. Process it and set its STATUS = 3 (processed state)

Step 5: Push on the stack all the neighbors of N that are in the ready state (whoseSTATUS = 1) and set their STATUS = 2 (waiting state) [END OF LOOP]

Step 6: EXIT

PROGRAM(DFS):

```
from collections import defaultdictclass

Graph:

    def __init__(self):

        self.graph = defaultdict(list)def

    addEdge(self, u, v):

        self.graph[u].append(v)def

    DFSUtil(self, v, visited):

        visited.add(v)

        print(v, end=' ')

         for neighbour in self.graph[v]:

            if neighbour not in visited:
                self.DFSUtil(neighbour, visited)

    def DFS(self, v):

        visited = set()

         self.DFSUtil(v, visited)
```

```
if __name__ == "__main__":
g = Graph()
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)
print("Following is DFS from (starting from vertex 2)")
g.DFS(2)
```

OUTPUT:

Following is Depth First Traversal (starting from vertex 2)
2 0 1 3

RESULT:

Thus the uninformed search algorithms such as DFS have been executed successfully and the output got verified.

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

AIM:

To write a python program for implementation of bounded A* algorithm.

ALGORITHM:

STEP 1: Start

STEP 2: Create a class Node to represent a state in the search. It has attributes like the state itself, the parent node, the cost to reach the current state, and aheuristic value.

STEP 3: Implement the astar function that takes the start state, goal state, and agraph representing the connections between states.

STEP 4: Create an empty priority queue to store nodes based on their cost plusheuristic values.

STEP 5: Push the initial node with the start state, no parent, cost of 0, andheuristic of 0 into the priority queue.

STEP 6: Create an empty set to keep track of visited states.

STEP 7: Repeat until the priority queue is empty.

STEP 8: If the current node's state has already been visited, skip to the nextiteration.

STEP 9: Add the current node's state to the visited set.

STEP 10: If no path is found after exhausting all possible nodes, return None.

STEP 11: Define the graph representation, specifying the connections betweenstates and the associated costs.

STEP 12: Define the start and goal states.

STEP 13: Call the astar function with the start, goal, and graph parameters andstore the result.
STEP 14: Print the result.

PROGRAM:

```
import heapq

class Node:

    def __init__(self, state, parent, cost, heuristic):

        self.state = state

        self.parent = parent
```

```
        self.cost = cost
        self.heuristic = heuristic

    def __lt__(self, other):
        return (self.cost + self.heuristic) < (other.cost + other.heuristic)def
astar(start, goal, graph,max_nodes):
    heap = []
    heapq.heappush(heap, (0, Node(start, None, 0, 0)))visited = set()
    node_counter=0
    while heap and node_counter<max_nodes:(cost, current)
        = heapq.heappop(heap)
        if current.state == goal:
            path = []
            while current is not None:
                path.append(current.state)
                current = current.parent
            return path[::-1]
        if current.state in visited:continue
        visited.add(current.state)
        node_counter+=1
        for state, cost in graph[current.state].items():if state not in
            visited:
                heuristic = 0
                heapq.heappush(heap, (cost, Node(state, current, current.cost + cost,heuristic))) return
    None
graph = {
    'A': {'B': 1, 'C': 4},
    'B': {'A': 1, 'C': 2, 'D': 5},
    'C': {'A': 4, 'B': 5, 'D': 1},
    'D': {'B': 5, 'C': 1}

}
```

```
start = 'A'goal
= 'D'
max_nodes=10
result = astar(start, goal, graph,max_nodes)
print(result)
```

OUTPUT:

['A','B','C','D']

RESULT:

Thus the program of bounded A* algorithm implementation has been executed successfully and the output got verified.

Ex.No:2(b)	**IMPLEMENTATION OF INFORMED SEARCH ALGORITHM (MEMORY-BOUNDED A*)**
Date:	

AIM:

To write a python program for implementation of A* algorithm.

ALGORITHM:

STEP 1: Start

STEP 2: Create a class Node to represent a state in the search. It has attributes like the state itself, the parent node, the cost to reach the current state, and aheuristic value.

STEP 3: Implement the astar function that takes the start state, goal state, and a graph representing the connections between states.

STEP 4: Create an empty priority queue to store nodes based on their cost plusheuristic values.

STEP 5: Push the initial node with the start state, no parent, cost of 0, andheuristic of 0 into the priority queue.

STEP 6: Create an empty set to keep track of visited states.

STEP 7: Repeat until the priority queue is empty.

STEP 8: If the current node's state has already been visited, skip to the next iteration.

STEP 9: Add the current node's state to the visited set.

STEP 10: If no path is found after exhausting all possible nodes, return None.

STEP 11: Define the graph representation, specifying the connections between states and the associated costs.

STEP 12: Define the start and goal states.

STEP 13: Call the astar function with the start, goal, and graph parameters and store the result.

STEP 14: Print the result.

PROGRAM:

```
import heapq
class Node:
    def __init__(self, state, parent, cost, heuristic):
        self.state = state
        self.parent = parent
```

```
        self.cost = cost
        self.heuristic = heuristic

    def __lt__(self, other):
        return (self.cost + self.heuristic) < (other.cost + other.heuristic)def
  astar(start, goal, graph):
    heap = []
    heapq.heappush(heap, (0, Node(start, None, 0, 0)))visited = set()
    while heap:
        (cost, current) = heapq.heappop(heap)if
        current.state == goal:
            path = []
            while current is not None:
                path.append(current.state)
                current = current.parent
            # Return reversed pathreturn path[::-1]
     if current.state in visited:continue
     visited.add(current.state)
     for state, cost in graph[current.state].items():if state
         not in visited:
             heuristic = 0 # replace with your heuristic function
             heapq.heappush(heap, (cost, Node(state, current, current.cost + cost,heuristic)))
   return None # No path found
graph = {
   'A': {'B': 1, 'D': 3},
   'B': {'A': 1, 'C': 2, 'D': 4},
   'C': {'B': 2, 'D': 5, 'E': 2},
   'D': {'A': 3, 'B': 4, 'C': 5, 'E': 3},
   'E': {'C': 2, 'D': 3}

}
start = 'A'goal
```

```
= 'E'
result = astar(start, goal, graph)print(result)
```

OUTPUT:

```
['A','B','C','E']
```

RESULT:

Thus the program of A* algorithm implementation has been executed successfully and the output was verified successfully.

Ex.No:3	IMPLEMENT NAÏVE BAYES MODEL
Date:	

AIM:

To write a python program to implement Naïve Bayes model.

ALGORITHM:

STEP 1: Load the libraries: import the required libraries such as pandas, numpy,and sklearn.

STEP 2: Load the data into a pandas dataframe.

STEP 3: Clean and preprocess the data as necessary. For example, you can handle missing values, convert categorical variables into numerical variables, and normalize the data.

STEP 4: Split the data into training and test sets using the train_test_split function from scikit-learn.

STEP 5: Train the Gaussian Naive Bayes model using the training data.

STEP 6: Evaluate the performance of the model using the test data and the accuracy_score function from scikit-learn.

STEP 7: Finally, you can use the trained model to make predictions on new data.

PROGRAM:

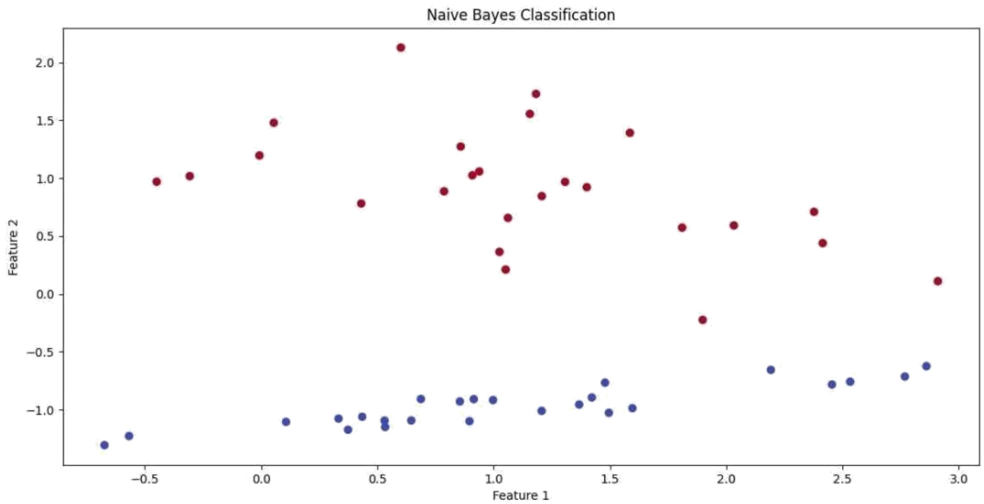
```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_splitfrom

sklearn.datasets import make_classification from

sklearn.naive_bayes import GaussianNB

# Generate random data
x,y=make_classification(n_samples=50,n_features=2,n_informative=2,n_redund
ant=0,n_clusters_per_class=1,random_state=25)

# Split data into training and test sets

x_train,x_test=x[:20],x[20:]

y_train,y_test=y[:20],y[20:]

# Train Naive Bayes classifierclf

= GaussianNB() clf.fit(x_train,

y_train)
```

```
# Predict labels for test datay_pred =
clf.predict(x_test)
# Calculate accuracy of the classifier
accuracy = clf.score(x_test, y_test)
print("Accuracy: ", accuracy)
# Plot the data and decision boundary
x_min, x_max = x[:, 0].min() - 1, x[:, 0].max() + 1 y_min,
y_max = x[:, 1].min() - 1, x[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02), np.arange(y_min, y_max,0.02)) Z =
clf.predict(np.c_[xx.ravel(), yy.ravel()])Z =
Z.reshape(xx.shape)
plt.scatter(x[:, 0], x[:, 1], c=y, cmap=plt.cm.coolwarm)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2') plt.title('Naive
Bayes Classification')plt.show()
```

OUTPUT:

Accuracy:1.0

DIAGRAMATIC PROJECTION



RESULT:

Thus the Python program for implementing Naïve Bayes model was developed and the output was verified successfully

Ex.No:4	IMPLEMENT BAYESIAN NETWORKS
Date:	

AIM:

To construct a Bayesian network, to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.

ALGORITHM:

STEP 1: Start

STEP 2: Read the training dataset T.

STEP 3: Calculate the mean and standard deviation of the predictor variables ineach class.

STEP 4: Repeat calculate the probability of fi using the gauss density equation ineach class. Until the probability of all predictor variables (f1,f2,f3,..,fn)has been calculated.

STEP 5: Calculate the likelihood for each class.

STEP 6: Get the greatest likelihood.

STEP 7: Stop

PROGRAM:

```
import bayespy as bp
import numpy as np
import csv
from colorama import init
from colorama import Fore, Back, Styleinit()
ageEnum = {'SuperSeniorCitizen':0, 'SeniorCitizen':1, 'MiddleAged':2, 'Youth':3,'Teen':4)
genderEnum = {'Male':0, 'Female':1}
familyHistoryEnum = {'Yes':0, 'No':1}
dietEnum = {'High':0, 'Medium':1, 'Low':2}
lifeStyleEnum = {'Athlete': 0, 'Active':1, 'Moderate':2, 'Sedetary':3}cholesterolEnum
= {'High':0, 'BorderLine':1, 'Normal':2} heartDiseaseEnum = {'Yes':0, 'No':1} with
open('heart disease_data.csv') as csvfile:lines =
   csv.reader(csvfile)
   dataset = list(lines)data
```

```
  = []
  for x in dataset:
      data.append([ageEnum[x[0]],genderEnum(x[1]], familyHistoryEnum[x[2]],dietEnum[x [3]],
lifeStyleEnum[x[4]],cholesterolEnum[x[5]],heart DiseaseEnum[x[6]]])
data= np.array(data)
N=len(data)
p_age=bp.nodes.Dirichlet (1.0*np.ones(5)) age =
bp.nodes.Categorical(p_age, plates=(N,))
age.observe(data[:0])
P_gender = bp.nodes.Dirichlet (1.0* np.ones(2)) gender =
bp.nodes.Categorical(p_gender, plates=(N,))
gender.observe(data[:,1])
p_familyhistory = bp.nodes.Dirichlet (1.0*np.ones(2)) familyhistory =
bp.nodes.Categorical(p_familyhistory, plates=(N,))
familyhistory.observe(data[:,2])
p_diet = bp.nodes.Dirichlet(1.0*np.ones(3)) diet =
bp.nodes.Categorical(p_diet, plates=(N,))
diet.observe(data[:,3])
p_lifestyle = bp.nodes.Dirichlet (1.0* np.ones(4)) lifestyle =
bp.nodes.Categorical(p_lifestyle, plates=(N,))
lifestyle.observe(data[:,4])
p_cholesterol = bp.nodes.Dirichlet (1.0* np.ones(3)) cholesterol =
bp.nodes.Categorical(p_cholesterol, plates=(N,))
cholesterol.observe(data[:,5])
p_heartdisease = bp.nodes.Dirichlet(np.ones(2), plates (5, 2, 2, 3, 4, 3))
heartdisease = bp.nodes.MultiMixture([age, gender, familyhistory, diet, lifestyle,cholesterol],
bp.nodes.Categorical, p_heartdisease)
heartdisease.observe(data[:,6])
p_heartdisease.update()
m=0
while m == 0:
print("\n")
```

```
res = bp.nodes.MultiMixture([int(input('Enter Age: ' + str(ageEnum))),
int(input('EnterGender: ' + str(genderEnum))),

      int(input('Enter FamilyHistory: ' + str(familyHistoryEnum))), int(input('Enter
dietEnum: ' + str(dietEnum))),

      int(input('Enter LifeStyle: ' + str(lifeStyleEnum))), int(input('Enter Cholesterol:'
+str(cholesterolEnum)))],

bp.nodes.Categorical,p_heartdisease).get_moments()[0][heartDiseaseEnum['Yes']]

print("Probability(Heart Disease)=" + str(res)) m =

int(input("Enter for Continue:0, Exit :1 "))
```

OUTPUT:

Enter
Age:{‘SuperSeniorCitizen’:0,’SeniorCitizen’:1,’MiddleAged’:2,’Youth’:3,’Teen’:4}1

Enter Gender:{‘Male’:0,’Female’:1}0

Enter FamilyHistory:{‘Yes’:0,’No’:1}0

Enter dietEnum:{‘High’:0,’Medium’:1,’Low’:2}2

Enter LifeStyle:{‘Athlete’:0,’Active’:1,’Moderate’:2,’Sedetary’:3}2Enter

Cholesterol:{‘High’:0,’BorderLine’:1,’Normal’:2}1

Probability(HeartDisease)=0.5

Enter for Continue:0,Exit:1 1

RESULT:

Thus the program to implement a Bayesian networks in the given heart disease dataset have been executed successfully and the output got verified.

Ex.No:5	BUILD REGRESSION MODELS
Date:	

AIM:

To build regression models such as locally weighted linear regression and plot the necessary graphs.

ALGORITHM:

STEP 1: Start

STEP 2: import the packages for the linear regression

STEP 3: Read the Datasets in CSV file

STEP 4: Select the independent and dependent variable X and Y.

STEP 5: Set X, Y labels and title

STEP 6: plot the dots using function ‘scatter()’

STEP 7: Determine the weight matrix using:W X, Xo)=e^(-(X-Xo)^2/2τ^2)

STEP 8: Determine the value of model term parameterβB(Xo)= (X^T W X)^-1 X ^ T W y

STEP 9: Perdition = Xo * β

STEP 10: Stop.

PROGRAM:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as pltimport

statsmodels.api as sm

data = pd.read_csv('Simple linear regression.csv')
data.describe() y =

data["GPA"] x1  =

data["SAT"]

plt.scatter(x1,y)

plt.xlabel("GPA",     fontsize    =   20)
```

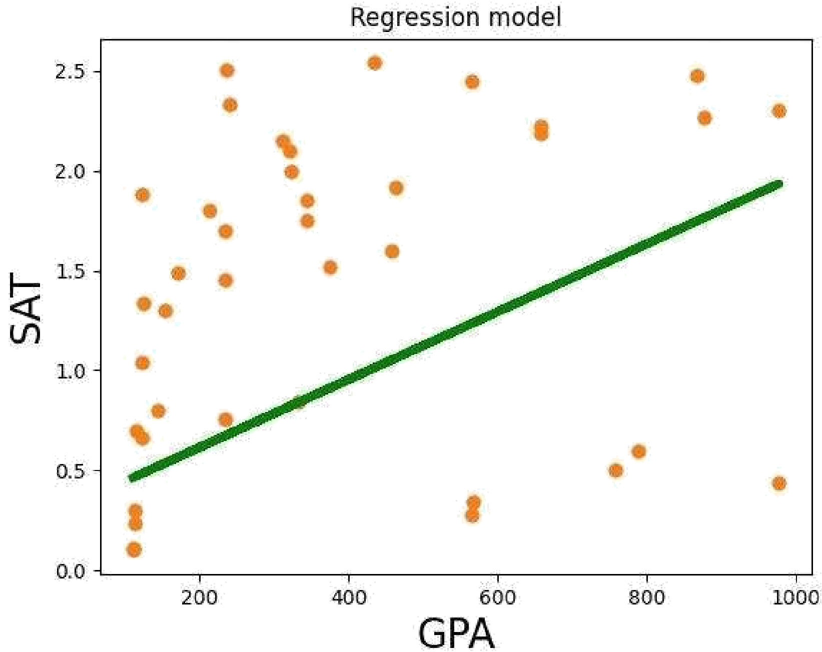
```
plt.ylabel("SAT",     fontsize    =   20)
plt.title("Regression model")
x = sm.add_constant(x1)
results = sm.OLS(y,x).fit()
results.summary()
plt.scatter(x1,y)
yhat = 0.0017*x1 + 0.275
fig = plt.plot(x1,yhat, lw=4, c="green", label = "regression line")plt.xlabel("gpa",
fontsize = 20)
plt.ylabel("sat", fontsize = 20)
plt.show()
```

OUTPUT:

Data set for GPA and SAT

GPA	SAT
0.1	110
0.11	111
0.3	112
0.23	113
0.28	565
0.34	567
0.44	976
0.5	756
0.6	788
0.66	123
0.7	114
……	……
……	
1.04	123
1.3	154
1.34	124

DIAGRAMATIC PROJECTION



RESULT:

Thus the above given program was executed and verified successfully.

Ex.No:6(a)	**BUILD DECISION TREES AND RANDOM FORESTS** (DECISION TREES)
Date:	

AIM:

To implement the concept of decision tree with suitable dataset in python.

ALGORITHM:

STEP 1: Start

STEP 2: Import necessary libraries: numpy, matplotlib, seaborn, pandas,train_test_split, LabelEncoder, DecisionTreeClassifier, plot_tree, and Random Forest Classifier.

STEP 3: Read the data from 'flowers.csv' into a pandas Data Frame.

STEP 4: Extract the features into an array X, and the target variable into an array.

STEP 5: Encode the target variable using the Label Encoder.

STEP 6: Split the data into training and testing sets using train_test_splitfunction.

STEP 7: Create a Decision Tree Classifier object, fit the model to the training data,and visualize the decision tree using plot_tree.

STEP 8: Create a Random Forest Classifier object with 100 estimators, fit themodel to the training data, and visualize the random forest by displaying 6 trees.

STEP 9: Print the accuracy of the decision tree and random forest models using the score method on the test data.

STEP 10: Stop.

PROGRAM:

```
import matplotlib.pyplot as pltimport
pandas as pd
import numpy as np
import scipy as sp
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
```

```
from sklearn.model_selection import train_test_splitfrom
sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score from
sklearn.metrics import classification_report

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


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

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

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

    # Separating the target variableX =
    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: 01234
0B1111
1R1112
2R1113
3R1114
4R1115
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]
[06718]
[0 19 71]]
Accuracy : 73.40425531914893
Report : precision recall f1-score support

	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]
[06322]
[0 20 70]]
Accuracy : 70.74468085106383
Report : precision recall f1-score support

	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 program to implement the concept of decision tree with suitable dataset has been executed successfully.

Ex.No:6(b)	**BUILD DECISION TREES AND RANDOM FORESTS** (RANDOM FORESTS)
Date:	

AIM:

To implement the concept of random forests with suitable dataset in python.

ALGORITHM:

STEP 1: Start

STEP 2: Import necessary libraries: numpy, matplotlib, seaborn, pandas,train_test_split, LabelEncoder, DecisionTreeClassifier, plot_tree, and Random Forest Classifier.

STEP 3: Read the data from 'flowers.csv' into a pandas Data Frame.

STEP 4: Extract the features into an array X, and the target variable into an array.

STEP 5: Encode the target variable using the Label Encoder.

STEP 6: Split the data into training and testing sets using train_test_splitfunction.

STEP 7: Create a DecisionTreeClassifier object, fit the model to the training data,and visualize the decision tree using plot_tree.

STEP 8: Create a RandomForestClassifier object with 100 estimators, fit themodel to the training data, and visualize the random forest by displaying 6 trees.

STEP 9: Print the accuracy of the decision tree and random forest models usingthe score method on the test data.

STEP 10: Stop.

PROGRAM:

```
import matplotlib.pyplot as pltimport
pandas as pd
import numpy as np
import scipy as sp
import numpy as np
import pandas as pd
import numpy as np
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as pltimport
seaborn as sns
df = pd.read_csv("titanic.csv")
df.drop(['Cabin','PassengerId','Name','Ticket'],axis=1,inplace=True) df =
df.fillna(0)
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder() df['Sex']=le.fit_transform(df['Sex'])
df['Embarked']=le.fit_transform(df['Embarked']) # Putting
feature variable to X
X = df.drop('Survived',axis=1)
# Putting response variable to yy =
df['Survived']
# Splitting the data into train and test
from sklearn.model_selection import train_test_split
# Splitting the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7,random_state=42)
Import Random Forest Model
from sklearn.ensemble import RandomForestClassifier#Create
a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)
#Train the model using the training sets y_pred=clf.predict(X_test)clf.fit(X_train,y_train)
# Predicting the test set results Pred
= classifier.predict(X_test) print(Pred)
from sklearn.metrics import classification_report
rand_score=classifier.score(X_test, y_test)
'''rand_score=classifier.accuracy_score(y_test,Pred)'''
classification_report_rf=classification_report(y_test,Pred)
print("Accuracy score:",rand_score)
```

OUTPUT:

[0110111000101011100010111111010000101

111111110110000110001000101100111111 01100010110001001]

Accuracy score: 0.8268156424581006

RESULT:

Thus the program to implement the concept of random forest with suitable dataset has been executed successfully.

Ex.No:7	BUILD SVM MODELS
Date:	

AIM:

To write a Python program to build SVM model.

ALGORITHM:

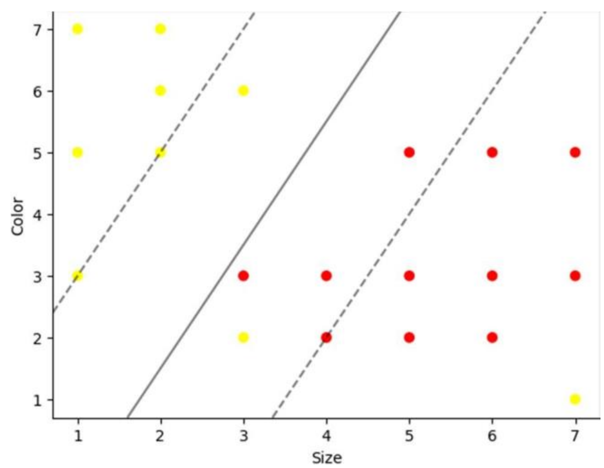
STEP 1: Import the necessary libraries (matplotlib.pyplot, numpy, and svm from sklearn).
STEP 2: Define the features (X) and labels (y) for the fruit dataset.
STEP 3: Create an SVM classifier with a linear kernel using svm.SVC(kernel='linear').
STEP 4: Train the classifier on the fruit data using clf.fit(X, y).
STEP 5: Plot the fruits and decision boundary using plt.scaΣer(X[:, 0], X[:, 1], c=colors), where colors is a list of colors assigned to each fruit based on its label.
STEP 6: Create a meshgrid to evaluate the decision function using np.meshgrid(np.linspace(xlim[0], xlim[1], 100), np.linspace(ylim[0], ylim[1], 100)).
STEP 7: Use the decision function to create a contour plot of the decision boundary and margins using ax.contour(xx, yy, Z, colors='k', levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--']).
STEP 8: Show the plot using plt.show().

PROGRAM:

```
import matplotlib.pyplot as plt
import numpy as np from sklearn
import svm
# Define the fruit features (size and color)
X = np.array([[5, 2], [4, 3], [1, 7], [2, 6], [5, 5], [7, 1], [6, 2], [5, 3], [3, 6], [2, 7], [6, 3], [3, 3], [1, 5],
[7, 3], [6, 5], [2, 5], [3, 2], [7, 5], [1, 3], [4, 2]])
# Define the fruit labels (0=apples, 1=oranges)
y = np.array([0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0])
# Create an SVM classifier with a linear kernel
clf = svm.SVC(kernel='linear')
# Train the classifier on the fruit data clf.fit(X, y)
# Plot the fruits and decision boundary
colors = ['red' if label == 0 else 'yellow' for label in y]
plt.scatter(X[:, 0], X[:, 1], c=colors)
ax = plt.gca()
ax.set_xlabel('Size')
ax.set_ylabel('Color')
 xlim = ax.get_xlim()
ylim = ax.get_ylim()
# Create a meshgrid to evaluate the decision function
xx, yy = np.meshgrid(np.linspace(xlim[0], xlim[1], 100), np.linspace(ylim[0], ylim[1], 100))
```

```
Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
# Plot the decision boundary and margins
ax.contour(xx, yy, Z, colors='k', levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])
plt.show()
```

OUTPUT:



RESULT:

Thus, the Python program to build an SVM model was developed, and the output was successfully verified.

Ex.No:8	**IMPLEMENT ENSEMBLING TECHNIQUES**
Date:	

AIM:

To implement the ensembling technique of Blending with the given Alcohol QCM Dataset.

ALGORIHTM:

STEP 1: Split the training dataset into train, test and validation dataset.
STEP 2: Fit all the base models using train dataset.
STEP 3: Make predictions on validation and test dataset.
STEP 4: These predictions are used as features to build a second level model
STEP 5: This model is used to make predictions on test and meta features.

PROGRAM:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import log_loss
# importing machine learning models for prediction
from sklearn.ensemble
import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
# importing voting classifier
 from sklearn.ensemble import VotingClassifier
# loading train data set in dataframe from train_data.csv file
df = pd.read_csv("train_data.csv")
# geţting target data from the dataframe
target = df["Weekday"]
# geţting train data from the dataframe
train = df.drop("Weekday")
# Spliţng between train data into training and validation dataset
X_train, X_test, y_train, y_test = train_test_split( train, target, test_size=0.20)
# initializing all the model objects with default parameters
model_1 = LogisticRegression()
model_2 = XGBClassifier()
model_3 = RandomForestClassifier()
# Making the final model using voting classifier
final_model = VotingClassifier( estimators=[('lr', model_1), ('xgb', model_2), ('rf', model_3)],
voting='hard')
# training all the model on the train dataset
```

```
final_model.fit(X_train, y_train)
# predicting the output on the test dataset
pred_final = final_model.predict(X_test)
# printing log loss between actual and predicted value
print(log_loss(y_test, pred_final))
```

OUTPUT:

231

RESULT:

Thus the program to implement ensembling technique of Blending with the given Alcohol QCM Dataset have been executed successfully and the output got verified.

Ex.No:9	IMPLEMENT CLUSTERING ALGORITHMS
Date:	

AIM:

To implment k-Nearest Neighbour algorithm to classify the Iris Dataset.

ALGORITHM:

STEP 1: Select the number K of the neighbors
STEP 2: Calculate the Euclidean distance of K number of neighbors
STEP 3: Take the K nearest neighbors as per the calculated Euclidean distance.
STEP 4: Among these k neighbors, count the number of the data points in each Category
STEP 5: Assign the new data points to that category for which the number of the Neighbor is maximum
STEP 6: Our model is ready.

PROGRAM:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import pandas as pd
import numpy as np
from sklearn import datasets
iris=datasets.load_iris()
iris_data=iris.data
iris_labels=iris.target
x_train, x_test, y_train, y_test=(train_test_split(iris_data, iris_labels, test_size=0.20))
classifier=KNeighborsClassifier(n_neighbors=6)
classifier.fit(x_train, y_train)
y_pred=classifier.predict(x_test)
print("accuracy is")
print(classification_report(y_test, y_pred))
```

OUTPUT:

accuracy is

	precision	recall	f1-score	support
0	1.00	1.00.	1.00	8
1	1.00.	0.90.	0.95.	10
2	0.92.	1.00.	0.96.	12
accuracy			0.97	30
macro avg	0.97	0.97	0.97	30
weighted avg	0.97	0.97	0.97	30

RESULT:

Thus the program to implement k-Nearest Neighbour Algorithm for clustering Iris Dataset have been executed successfully and output got verified.

Ex.No:10	IMPLEMENT EM OR BAYESIAN NETWORKS
Date:	

AIM:

To write a program to implement EM for Bayesian Networks.

ALGORITHM:

STEP 1: IMPLEMENT CLUSTERING ALGORITHM Start with a EM Bayesian network using the iris dataset

STEP 2: Frame the iris dataset with the Sepal_Length,Sepal_Width,Petal_length,Petal_width

STEP 3: E-step: Compute the expected sufficient statistics for the latent variables. This involves computing the posterior probability distribution over the latent variables given the observed data and the current parameter estimates. This can be done using the forward-backward algorithm or the belief propagation algorithm.

Compute q(h)=P(H=h|E=e; θ)for each h(probabilistic inference)

STEP 4: M-step: Update the parameter estimates using the expected sufficient statistics computed in step 3. This involves maximizing the likelihood of the data with respect to the parameters of the network, given the expected sufficient statistics.

STEP 5: Repeat steps 3-4 until convergence. Convergence can be measured by monitoring the change in the log-likelihood of the data, or by monitoring the change in the parameter estimates.

PROGRAM:

```
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture
from sklearn.datasets import load_iris
import sklearn.metrics as sm
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
dataset=load_iris()
# print(dataset)
X=pd.DataFrame(dataset.data)
X.columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y=pd.DataFrame(dataset.target)
y.columns=['Targets']
# print(X) plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])
```

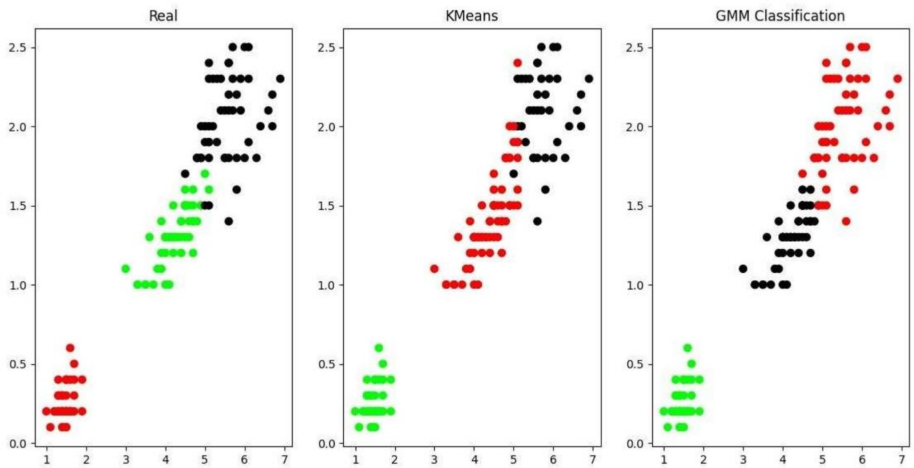
```
# REAL PLOT
plt.subplot(1,3,1)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Real')
# K-PLOT
plt.subplot(1,3,2)
model=KMeans(n_clusters=3)
model.fit(X)
predY=np.choose(model.labels_,[0,1,2]).astype(np.int64)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[predY],s=40)
plt.title('KMeans')
# GMM PLOT
scaler=preprocessing.StandardScaler()
scaler.fit(X)
xsa=scaler.transform(X)
xs=pd.DataFrame(xsa,columns=X.columns)
gmm=GaussianMixture(n_components=3)
gmm.fit(xs)
y_cluster_gmm=gmm.predict(xs)
plt.subplot(1,3,3)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm],s=40)
plt.title('GMM Classification')
plt.show()
```

OUTPUT:

id	Sepal_Length	Sepal_Width	Petal_length	Petal_width	Species
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
---	---	---	---	---	
---	---	---	---	---	
---	---	---	---	---	

7	4.6	3.4	1.4	0.3	Iris-SetoSa
8	5	3.4	1.5	0.2	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
					Iris-seto

DIAGRAMATIC PROJECTION



RESULT:

Thus the python program to implement EM for Bayesian Networks hasbeen executed and verified Successfully.

Ex.No:11	BUILD SIMPLE NN MODELS
Date:	

AIM:

To write a python program to build simple NN models.

ALGORITHM:

STEP 1: Start

STEP 2: Choose the number of layers and neurons in each layer. This depends onthe problem you are trying to solve.

STEP 3: Define the activation function for each layer. Common choices are ReLU,sigmoid, and tanh.

STEP 4: Initialize the weights and biases for each neuron in the network. This canbe done randomly or using a pre-trained model.

STEP 5: Define the loss function and optimizer to be used during training. Theloss function measures how well the model is doing, while the optimizer updates the weights and biases to minimize the loss.

STEP 6: Train the model on the input data using the defined loss function and optimizer. This involves forward propagation to compute the output of the model, and backpropagation to compute the gradients of the loss with respectto the weights and biases. The optimizer then updates the weights and biases based on the gradients.

STEP 7: Evaluate the performance of the model on new data using metrics suchas accuracy, precision, recall, and F1 score.

STEP 8: Stop

PROGRAM:

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
# Define the input and output data
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])
# Define the model
model = Sequential()
model.add(Dense(4, input_dim=2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model to the data
```

```
model.fit(X, y, epochs=2, batch_size=4)
# Evaluate the model on new data
test_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
predictions = model.predict(test_data) print(predictions)
```

OUTPUT:

```
[[0.49927324]
[0.45714095]
[0.5460681]
[0.60924554]]
```

RESULT:

Thus the Python program to build simple NN Models was developed successfully.

Ex.No:12	BUILD DEEP LEARNING NN MODELS
Date:	

AIM:

To implement and build a Convolutional neural network model which predicts the age and gender of a person using the given pre-trained models.

ALGORITHM:

STEP 1: Import the necessary libraries, such as numpy and keras.

STEP 2: Load or generate your dataset. This can be done using numpy or any other data manipulation library.

STEP 3: Preprocess your data by performing any necessary normalization, scaling, or other transformations.

STEP 4: Define your neural network architecture using the Keras Sequential API. Add layers to the model using the add() method, specifying the number of units, activation function, and input dimensions for each layer.

STEP 5: Compile your model using the compile() method. Specify the loss function, optimizer, and any evaluation metrics you want to use.

STEP 6: Train your model using the fit() method. Specify the training data, validation data, batch size, and number of epochs.

STEP 7: Evaluate your model using the evaluate() method. This will give you the loss and accuracy metrics on the test set.

STEP 8: Use your trained model to make predictions on new data using the predict() method.

PROGRAM:

```
# Import necessary libraries
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
# Define the neural network model
model = Sequential()
model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=10, activation='softmax'))
# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
# Generate some random data for training and testing
data = np.random.random((1000, 100))
```

```
labels = np.random.randint(10, size=(1000, 1))
one_hot_labels = keras.utils.to_categorical(labels, num_classes=10)
# Train the model on the data model.fit(data, one_hot_labels, epochs=10, batch_size=32)
# Evaluate the model on a test set
test_data = np.random.random((100, 100))
test_labels = np.random.randint(10, size=(100, 1))
test_one_hot_labels = keras.utils.to_categorical(test_labels, num_classes=10)
loss_and_metrics = model.evaluate(test_data, test_one_hot_labels,batch_size=32)
print("Test loss:", loss_and_metrics[0])
print("Test accuracy:", loss_and_metrics[1])
```

OUTPUT:

gender: Male, conf = 1.000
Age Output: [[2.8247703e-05 8.9249297e-05 3.0017464e-04 8.8183772e-03 9.3055397e-01 5.1735926e-02 7.6946630e-03 7.7927281e-04]]
Age: (25-32), conf = 0.873

RESULT:

Thus the program to implement and build a Convolutional neural network model which predicts the age and gender of a person using the given pre-trained models have been executed successfully and the output got verified.