Ex No:1 **Create the tic-tac-toe game using the adversarial**

Date : 04.08.2023 **searching algorithm.**

Aim :

To Create the tic tac toe game using any adversarial searching algorithm.

DESCRIPTION:

Tic-tac-toe, also known as nought and Crosses, is a classic two-player strategy board game played on a 3x3 grid. The objective of the game is to be the first player to form a line of three of their symbols, either horizontally, vertically, or diagonally, on the game board.Here are the basic rules of Tic-tac-toe:

Game Setup: The game is played on a 3x3 grid, usually drawn on paper or a game board. One player uses "X" as their symbol, and the other player uses "O" as their symbol.

Starting the Game: The first player, typically using "X," goes first. Players take turns placing their symbols on an empty cell of the grid.

Game Progression: Players continue taking turns, one after the other, until one of the following conditions is met:

Win: If a player successfully places three of their symbols in a row, column, or along a diagonal, they win the game. The game immediately ends, and that player is declared the winner.

Draw: If all cells on the grid are filled, and no player has formed a winning line, the game is a draw, and it ends in a tie.

Player Experience:

Human Player: The human player makes moves by selecting an empty cell on the game board.

AI Opponent: The AI opponent uses the Minimax algorithm to calculate its moves. It strategically blocks the human player's potential wins and seeks opportunities to win itself.

Challenging Gameplay: The AI opponent provides an intelligent challenge, capable of making optimal decisions based on the current game state.

Game Flow:

The game begins with an empty grid, and players take turns.

The human player makes a move by selecting an empty cell.

The game continues until a player wins or the board is full (draw).

End of Game:

If a player wins, the game announces the winner.

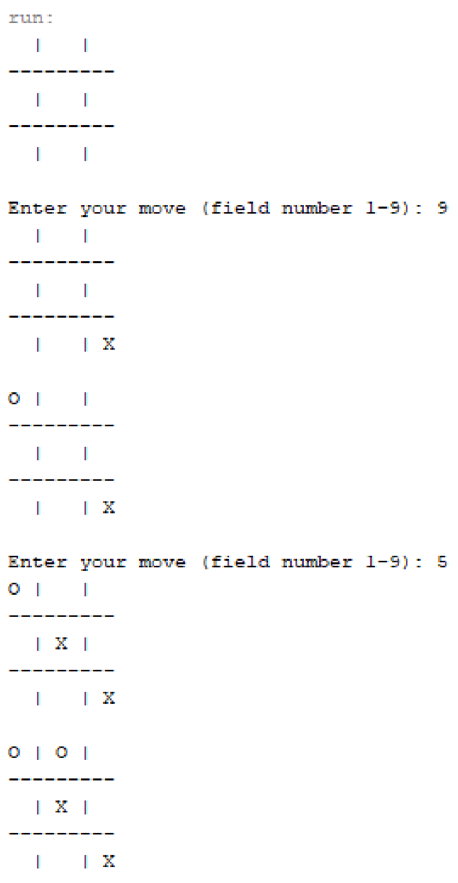
If the game ends in a draw, it declares the game a draw.

Players are given the option to start a new game or exit.

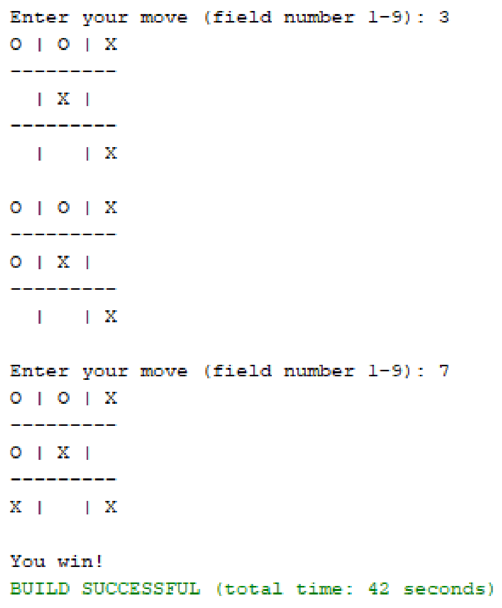
CODE :

```
package ai.ml;
import java.util.Scanner;
public class TicTacToeGame {
    public static void main(String[] args) {
        char[][] board = {
            {' ', ' ', ' '},
            {' ', ' ', ' '},
            {' ', ' ', ' '}
        };
        Scanner scanner = new Scanner(System.in);
        printBoard(board);
                while (true) {
            // Player's turn
            playerMove(board, scanner);
            printBoard(board);
            if (checkWin(board, 'X')) {
                System.out.println("You win!");
                break;
            } else if (isBoardFull(board)) {
                System.out.println("It's a draw!");
                break;
            }
            // AI's turn using Minimax
            aiMove(board);
            printBoard(board);
            if (checkWin(board, 'O')) {
                System.out.println("AI wins!");
                break;
            } else if (isBoardFull(board)) {
                System.out.println("It's a draw!");
                break;
            }
        }
        scanner.close();
    }
    static void printBoard(char[][] board) {
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print(board[i][j]);
                if (j < 2) {
                    System.out.print(" | ");
                }
            }
            System.out.println();
            if (i < 2) {
                System.out.println("---------");
            }
        }
        System.out.println();
    }
        static void playerMove(char[][] board, Scanner scanner) {
        int field;
        do {
            System.out.print("Enter your move (field number 1-9): ");
            field = scanner.nextInt();

            if (field < 1 || field > 9) {
                System.out.println("Invalid input. Please choose a field between 1 and 9.");
            } else if (!isValidMove(board, field)) {
                System.out.println("This field is already occupied. Choose an empty field.");
            }
        } while (field < 1 || field > 9 || !isValidMove(board, field));
        char symbol = 'X';
        makeMove(board, field, symbol);
    }
        static boolean isValidMove(char[][] board, int field) {
        int row = (field - 1) / 3;
        int col = (field - 1) % 3;
        return board[row][col] == ' ';
    }
        static void makeMove(char[][] board, int field, char symbol) {
        int row = (field - 1) / 3;
        int col = (field - 1) % 3;
        board[row][col] = symbol;
    }
    static boolean isBoardFull(char[][] board) {
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (board[i][j] == ' ') {
                    return false;
                }
            }
        }
        return true;
    }
    static boolean checkWin(char[][] board, char symbol) {
        // Check rows, columns, and diagonals for a win
        for (int i = 0; i < 3; i++) {
            if (board[i][0] == symbol && board[i][1] == symbol && board[i][2] == symbol) {
                return true;
            }
            if (board[0][i] == symbol && board[1][i] == symbol && board[2][i] == symbol) {
                return true;
            }
        }
        if (board[0][0] == symbol && board[1][1] == symbol && board[2][2] == symbol) {
            return true;
        }
        if (board[0][2] == symbol && board[1][1] == symbol && board[2][0] == symbol) {
            return true;
        }
        return false;
    }
    static int minimax(char[][] board, int depth, boolean isMaximizing) {
        // Minimax algorithm implementation goes here
        // Return the best score based on the current board state
        return 0;
    }
    static void aiMove(char[][] board) {
        int bestScore = Integer.MIN_VALUE;
        int bestRow = -1;
        int bestCol = -1;
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (board[i][j] == ' ') {
                    board[i][j] = 'O';
                    int score = minimax(board, 0, false);
                    board[i][j] = ' ';

                    if (score > bestScore) {
                        bestScore = score;
                        bestRow = i;
                        bestCol = j;
                    }
                }
            }
        }
        board[bestRow][bestCol] = 'O';
    }
}
```

Output:



```
run:
  |   |
---------
  |   |
---------
  |   |

Enter your move (field number 1-9): 9
  |   |
---------
  |   |
---------
  |   | X

O |   |
---------
  |   |
---------
  |   | X

Enter your move (field number 1-9): 5
O |   |
---------
  | X |
---------
  |   | X

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



```
Enter your move (field number 1-9): 3
O | O | X
---------
  | X |
---------
  |   | X

O | O | X
---------
O | X |
---------
  |   | X

Enter your move (field number 1-9): 7
O | O | X
---------
O | X |
---------
X |   | X

You win!
BUILD SUCCESSFUL (total time: 42 seconds)
```

Observation(20)	
Record(5)	
Total(25)	
Initial	

Result :

Thus the tic tac toe game using any adversarial searching algorithm is written and executed successfully.

Ex No:2 Design the Tower of Hanoi problem using the searching algorithm

Date : 11.08.2023

Aim:

To design the Tower of Hanoi problem using the searching algorithm.

DESCRIPTION:

Objective and Win Condition:

1. The Tower of Hanoi, a classic mathematical puzzle, presents a unique challenge with a clear objective.
2. The goal is to move an entire tower of differently sized disks from one peg to another.
3. To win the puzzle, you must achieve the objective while adhering to specific rules.

Game Setup:

1. The Tower of Hanoi is played with three pegs, labeled as A, B, and C.
2. Disks of varying sizes are stacked in decreasing order on one peg, usually peg A.
3. The challenge is to transfer the entire stack of disks from the source peg to a target peg.

Game Progression:

Players or solvers make moves according to a set of rules that dictate how the disks can be moved.

Only one disk can be moved at a time.

A larger disk cannot be placed on top of a smaller disk.

Player Experience:

The puzzle is typically solved by individuals or computer programs that follow the rules.

Much like the human player in Tic-tac-toe, the solver strategizes to achieve the goal optimally.

Game Flow:

The puzzle starts with all disks on one peg and empty pegs for the destination and auxiliary moves.

Solvers take turns making moves, moving one disk at a time from one peg to another.

The game continues until the entire stack of disks is moved to the target peg.

Strategy and Complexity:

Solving the Tower of Hanoi requires strategic thinking, as each move must be chosen carefully to ensure the optimal outcome.

The complexity of the puzzle increases with the number of disks, requiring more meticulous planning.

Algorithmic Approach:

Just as the AI opponent in Tic-tac-toe uses the Minimax algorithm, solving the Tower of Hanoi optimally can also involve algorithmic approaches.

Recursive techniques are often employed to find the shortest solution, making it a fascinating problem in algorithm design.

Algorithm:

Step 1: Start the program with n disks

Step 2: Move n-1 disks from source to aux

Step 3: Move nth disk from source to dest

Step 4: Move n-1 disks from aux to dest

Step 5: Finally The disks are moved to destination peg

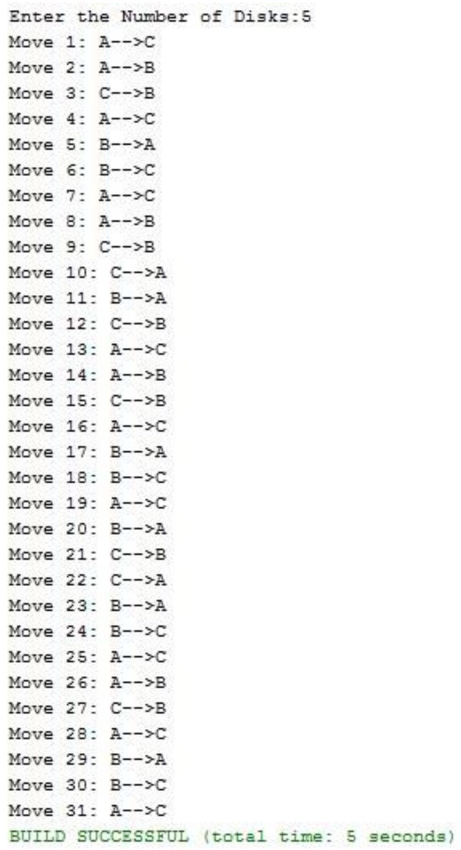
Step 6: Stop

Code:

```
package Exercise_2;

import java.util.Scanner;

public class Towers_of_Hanoi

{

   public static int move= 1;

   public static void main(String[] args)

   {

     String rods[] ={"A","B","C"};

     Scanner obj=new Scanner(System.in);

     System.out.print("Enter the Number of Disks:");

     int disk=obj.nextInt();

     move(disk,rods[0],rods[2],rods[1]);

   }
```

```
    public static void move(int disks,String beg,String end,String aux)
    {
        if(disks!=0)
        {
            move(disks-1,beg,aux,end);
            System.out.println("Move "+move+": "+beg+"-->"+end);
            move++;
            move(disks-1,aux,end,beg);
        }
    }
}
```

Output:



```
Enter the Number of Disks:5
Move 1: A-->C
Move 2: A-->B
Move 3: C-->B
Move 4: A-->C
Move 5: B-->A
Move 6: B-->C
Move 7: A-->C
Move 8: A-->B
Move 9: C-->B
Move 10: C-->A
Move 11: B-->A
Move 12: C-->B
Move 13: A-->C
Move 14: A-->B
Move 15: C-->B
Move 16: A-->C
Move 17: B-->A
Move 18: B-->C
Move 19: A-->C
Move 20: B-->A
Move 21: C-->B
Move 22: C-->A
Move 23: B-->A
Move 24: B-->C
Move 25: A-->C
Move 26: A-->B
Move 27: C-->B
Move 28: A-->C
Move 29: B-->A
Move 30: B-->C
Move 31: A-->C
BUILD SUCCESSFUL (total time: 5 seconds)
```

Observation(20)	
Record(5)	
Total(25)	

Result :

Thus the tower of hanoi problem using searching algorithm has successfully executed and output is verified.

Ex No:3 **Modeling the Greedy Best-First Search Algorithm**

Date : 22.08.2023

Aim:

To design the modelling of greedy best-first searching algorithm.

Theory:

The algorithm works by using a heuristic function to determine which path is the most promising. The heuristic function takes into account the cost of the current path and the estimated cost of the remaining paths. If the cost of the current path is lower than the estimated cost of the remaining paths, then the current path is chosen. This process is repeated until the goal is reached.

Algorithm:

Best First Search:

```
Step 1. Initialize `visited` and `queue`, add `(0, 'A')` to `queue`.
Step 2. Loop until `queue` is empty:
   Step 2.1. Extract highest priority node from `queue`.
   Step 2.2. If node in `visited`, skip.
   Step 2.3. Add node to `visited`.
   Step 2. 4. If node is goal, exit with success.
   Step2.5. Add unvisited neighbors to `queue` with priority.
Step 3. Exit with failure (goal not reachable).
```

A* Search:

```
Step 1. Initialize `visited`, `queue`, `g_scores`.
Step 2. Add `(0, 'A')` to `queue`, set `g_scores['A'] = 0`.
Step 3. Loop until `queue` is empty:
   Step 3 .1. Extract lowest f-value node from `queue`.
   Step 3. 2. If node in `visited`, skip.
   Step3.3. Add node to `visited`.
   Step3.4. If node is goal, exit with success.
```

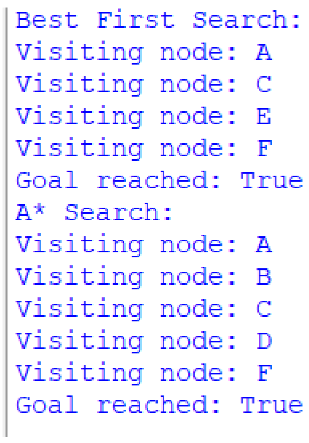
Step3. 5. Update `g_scores`, add unvisited neighbors to `queue` with f-value.

Step 4. Exit with failure (goal not reachable).

Code:

```
import heapq
def best_first_search(graph, start, goal):
    visited = set()
    queue = [(0, start)]  # Queue with (priority, node) tuples
    while queue:
        priority, node = heapq.heappop(queue)
        if node in visited:
            continue
        visited.add(node)
        print(f"Visiting node: {node}")
        if node == goal:
            return True  # Goal reached
        for neighbor, _ in graph[node]:
            if neighbor not in visited:
                heapq.heappush(queue, (heuristic[neighbor], neighbor))
    return False  # Goal not reachable
def a_star_search(graph, start, goal):
    visited = set()
    queue = [(0, start)]  # Queue with (f = g + h, node) tuples
    g_scores = {node: float('inf') for node in graph}
    g_scores[start] = 0
    while queue:
        _, node = heapq.heappop(queue)
        if node in visited:
            continue
        visited.add(node)
        print(f"Visiting node: {node}")
        if node == goal:
            return True  # Goal reached
        for neighbor, edge_cost in graph[node]:
            tentative_g = g_scores[node] + edge_cost
            if tentative_g < g_scores[neighbor]:
                g_scores[neighbor] = tentative_g
                heapq.heappush(queue, (tentative_g + heuristic[neighbor], neighbor))
    return False  # Goal not reachable
# ... (rest of the code remains the same)
graph = {
    'A': [('B', 2), ('C', 5)],
    'B': [('D', 3)],
    'C': [('E', 4)],
    'D': [('F', 1)],
    'E': [('F', 3)],
    'F': []
}
# Example heuristic function
heuristic = {
    'A': 8,
    'B': 6,
    'C': 4,
    'D': 4,
    'E': 2,
    'F': 0
}
start_node = 'A'
goal_node = 'F'
# Using the Best First Search algorithm
print("Best First Search:")
print("Goal reached:", best_first_search(graph, start_node, goal_node))
# Using the A* algorithm
print("A* Search:")
print("Goal reached:", a_star_search(graph, start_node, goal_node))
```

Output:



```
Best First Search:
Visiting node: A
Visiting node: C
Visiting node: E
Visiting node: F
Goal reached: True
A* Search:
Visiting node: A
Visiting node: B
Visiting node: C
Visiting node: D
Visiting node: F
Goal reached: True
```

Observation(20)	
Record(5)	
Total(25)	

Result :

Thus the modelling of greedy best-first searching algorithm has successfully executed and output is verified.

Ex No:4 Create the Environment for Probabilistic Inference Using a Bayesian Network

Date : 30.08.2023

Aim:

To create the environment for the probabilistic inference using a Bayesian network.

Theory:

Bayesian belief network is key computer technology for dealing with probabilistic events and to solve a problem which has uncertainty. We can define a Bayesian network as:

"A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph."

It is also called a **Bayes network, belief network, decision network**, or **Bayesian model**.

Bayesian networks are probabilistic, because these networks are built from a **probability distribution**, and also use probability theory for prediction and anomaly detection.

Real world applications are probabilistic in nature, and to represent the relationship between multiple events, we need a Bayesian network. It can also be used in various tasks including **prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction**, and **decision making under uncertainty**.

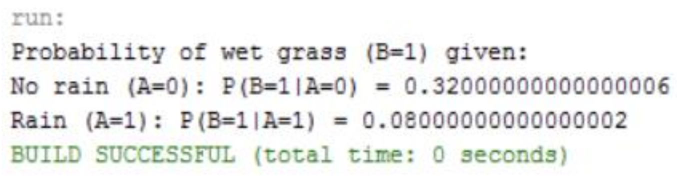
Algorithm:

Step 1: Define Nodes
Step 2: Add Probabilities and Dependencies
Step 3: Calculate Probabilities
Step 4: Display Results
Step 5: Execute the Code

Code:

```
package bayesiannetworkexample;
import java.util.HashMap;
import java.util.Map;
class Node {
    String name;
    Map<String, Double> probabilities = new HashMap<>();
    Map<Node, Double> parentProbabilities = new HashMap<>();
    Node(String name) {
        this.name = name;
    }
    void addProbability(String state, double probability) {
        probabilities.put(state, probability);
    }
    void addParent(Node parent, double conditionalProbability) {
        parentProbabilities.put(parent, conditionalProbability);
    }
    double getProbability(String state, Map<Node, String> evidence) {
        double probability = probabilities.get(state);
        for (Node parent : parentProbabilities.keySet()) {
            String parentState = evidence.get(parent);
            double conditionalProb = parentProbabilities.get(parent);
            probability *= conditionalProb;
        }
        return probability;
    }
}
public class BayesianNetworkExample {
    public static void main(String[] args) {
        // Define the nodes: Rain (A) and Wet Grass (B)
        Node A = new Node("A"); // Rain node
        A.addProbability("0", 0.7); // P(A=0) - No rain
        A.addProbability("1", 0.3); // P(A=1) - Rain
        Node B = new Node("B"); // Wet Grass node
        B.addProbability("0", 0.8); // P(B=0) - Grass not wet
        B.addProbability("1", 0.2); // P(B=1) - Grass wet
        B.addParent(A, 0.6); // P(B=1|A=0) - Grass wet given no rain
        B.addParent(A, 0.4); // P(B=1|A=1) - Grass wet given rain
        // Calculate P(B=1|A=0)
        Map<Node, String> evidenceNoRain = new HashMap<>();
        evidenceNoRain.put(A, "0"); // No rain (A=0)
        double pB1GivenA0 = B.getProbability("0", evidenceNoRain);
        // Calculate P(B=1|A=1)
        Map<Node, String> evidenceRain = new HashMap<>();
        evidenceRain.put(A, "1"); // Rain (A=1)
        double pB1GivenA1 = B.getProbability("1", evidenceRain);
        // Print probabilities in tabular format
        System.out.println("Probability of wet grass (B=1) given:");
        System.out.println("No rain (A=0): P(B=1|A=0) = " + pB1GivenA0);
        System.out.println("Rain (A=1): P(B=1|A=1) = " + pB1GivenA1);
    }
}
```

Output:



```
run:
Probability of wet grass (B=1) given:
No rain (A=0): P(B=1|A=0) = 0.32000000000000006
Rain (A=1): P(B=1|A=1) = 0.08000000000000002
BUILD SUCCESSFUL (total time: 0 seconds)
```

Observation(20)	
Record(5)	
Total(25)	

Result :

Thus the creation of the environment for the probabilistic inference using a Bayesian network has successfully executed and output is verified.

Ex No:5 Design a Program Using a Native Bayesian Classification for a Training Dataset Stored as a CSV File

Date : 07.09.2023

Aim:

To design a program using a native bayesian classification for a training dataset stored as a CSV file.

Theory:

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.
- It is mainly used in *text classification* that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.
- Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

Algorithm:

1. Import Libraries:

- Import necessary libraries including `pandas`, `train_test_split` from `sklearn.model_selection`, `GaussianNB` from `sklearn.naive_bayes`, and `accuracy_score` from `sklearn.metrics`.

2. Load the Dataset:

- Read the CSV dataset into a DataFrame.

3. Prepare Data:

- Select the features (X) and the target variable (y) from the DataFrame.

4. Split Data and Train the Classifier:

- Split the dataset into training and testing sets.

- Create a Gaussian Naive Bayes classifier.

- Train the classifier on the training data.

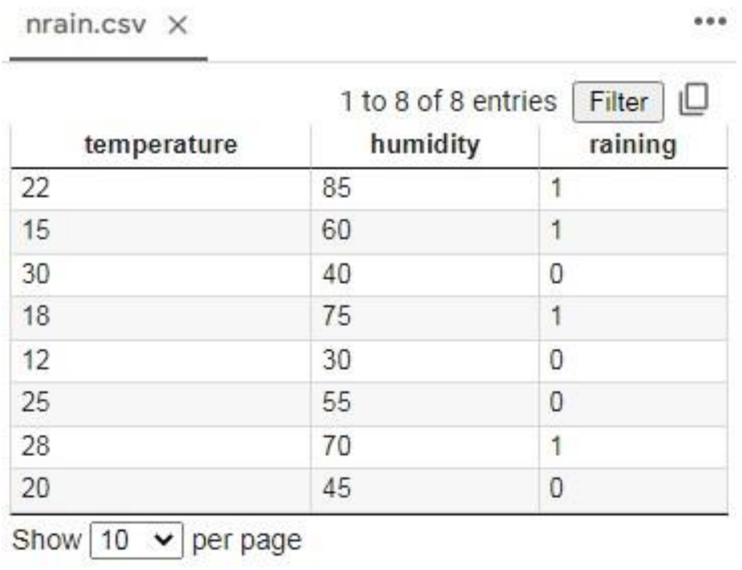
5. Evaluate and Make Predictions:

- Calculate the accuracy of the classifier on the test data using `accuracy_score`.
- Make predictions on new data points using the trained classifier.

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from google.colab import files
uploaded = files.upload()
import io
df = pd.read_csv(io.BytesIO(uploaded['nrain.csv']))
X = df[['temperature', 'humidity']]
y = df['raining']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
classifier = GaussianNB()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
new_data_point = [[21, 68]]
prediction = classifier.predict(new_data_point)
if prediction[0] == 1:
    print("It's predicted to be raining.")
else:
    print("It's predicted to not be raining.")
```

Dataset:

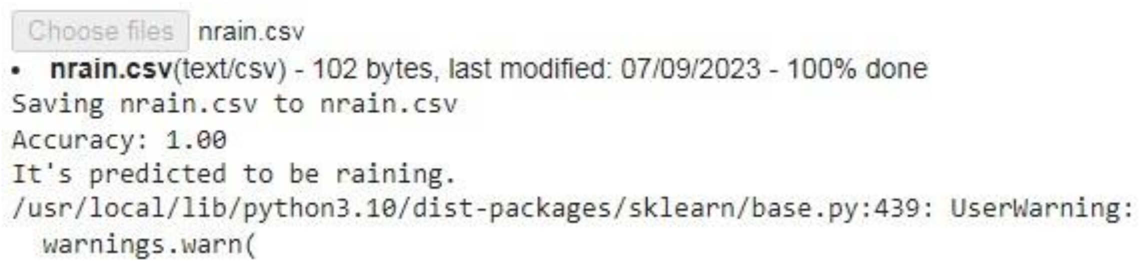

nrain.csv ×

1 to 8 of 8 entries Filter

temperature	humidity	raining
22	85	1
15	60	1
30	40	0
18	75	1
12	30	0
25	55	0
28	70	1
20	45	0

Show 10 per page

Output:


```
Choose files nrain.csv
• nrain.csv(text/csv) - 102 bytes, last modified: 07/09/2023 - 100% done
Saving nrain.csv to nrain.csv
Accuracy: 1.00
It's predicted to be raining.
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning:
  warnings.warn(
```

Observation(20)	
Record(5)	
Total(25)	

Result :

Thus the designing of a program using a native bayesian classification for a training dataset stored as a CSV file have been designed sucessfully.

Ex No:6 **Design a Program in K-nearest neighbour algorithm to**

Date : 14.09.2023 **classify the iris data set**

Aim:

To design a program using the K-nearest neighbour algorithm to classify the iris data set.

Theory:

- ✓ Statistical learning refers to a collection of mathematical and computation tools to understand data.In what is often called supervised learning, the goal is to estimate or predict an output based on one or more inputs.
- ✓ The inputs have many names, like predictors, independent variables, features, and variables being called common.
- ✓ The output or outputs are often called response variables, or dependent variables.If the response is quantitative – say, a number that measures weight or height, we call these problems regression problems.
- ✓ If the response is qualitative– say, yes or no, or blue or green, we call these problems classification problems.This case study deals with one specific approach to classification.
- ✓ The goal is to set up a classifier such that when it's presented with a new observation whose category is not known, it will attempt to assign that observation to a category, or a class, based on the observations for which it does know the true category.
- ✓ This specific method is known as the k-Nearest Neighbors classifier, or kNN for short.Given a positive integer k, say 5, and a new data point, it first identifies those k points in the data that are nearest to the point and classifies the new data point as belonging to the most common class among those k neighbors.

Algorithm:

Step 1: Import Necessary Libraries

Import the required libraries, such as scikit-learn for the KNN classifier.
Step 2: Load and Prepare the Dataset

Load the dataset from a CSV file or any other suitable format.
Separate the dataset into features (X) and labels (y).
Step 3: Initialize the KNN Classifier

Prompt the user to input the value of 'k' for the K-nearest neighbors algorithm.
Step 4: Train the KNN Classifier

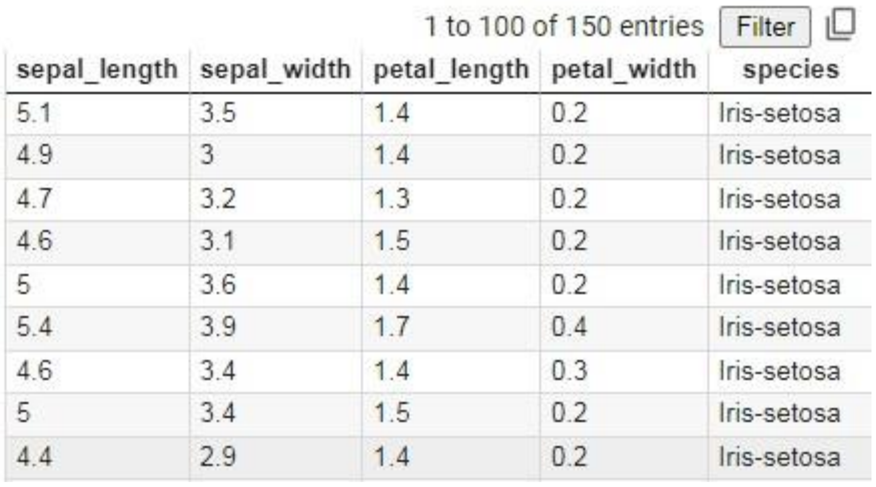
Fit the KNN classifier with the features (X) and labels (y) from the dataset.
Step 5: Make Predictions and Display Results

Prompt the user to input new data for prediction.
Standardize or preprocess the new data using the same preprocessing applied to the dataset.
Use the trained KNN classifier to predict the label for the new data.
Display the predicted label to the user.

Code:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
data = pd.read_csv('/content/IRIS.csv')
X = data.iloc[:, :-1].values  # Features (sepal length, sepal width, petal length, petal width)
y = data.iloc[:, -1].values   # Labels (species)
scaler = StandardScaler()
X = scaler.fit_transform(X)
k = int(input("Enter the value of k for KNN: "))
knn_classifier = KNeighborsClassifier(n_neighbors=k)
knn_classifier.fit(X, y)
new_data = []
for i in range(4):
    value = float(input(f"Enter the value for feature {i + 1}: "))
    new_data.append(value)
new_data = scaler.transform([new_data])
predicted_label = knn_classifier.predict(new_data)
print("Predicted Label:", predicted_label[0])
```

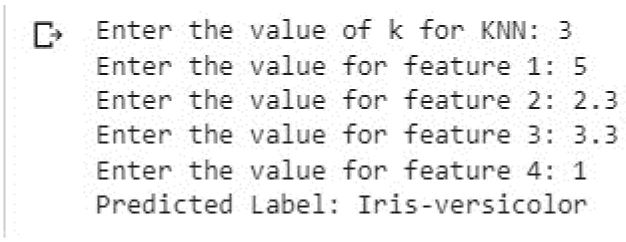
Dataset:



1 to 100 of 150 entries Filter

sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa

Output:



```
Enter the value of k for KNN: 3
Enter the value for feature 1: 5
Enter the value for feature 2: 2.3
Enter the value for feature 3: 3.3
Enter the value for feature 4: 1
Predicted Label: Iris-versicolor
```

Observation(20)	
Record(5)	
Total(25)	

Result :

Thus the design of K-Nearest neighbour algorithm has been created successfully.

Ex No:7 **Model the K-Means Algorithm for Colour Compression**

Date : 04.10.2023

Aim:

To design a program using the K-Means Algorithm for Colour Compression.

Theory:

- ✓ The internet is filled with huge amounts of data in the form of images.
- ✓ People upload millions of pictures every day on social media sites such as Instagram, and Facebook and cloud storage platforms such as google drive, etc.
- ✓ With such large amounts of data, image compression techniques become important to compress the images and reduce storage space.
- ✓ We will look at image compression using the K-means clustering algorithm which is an unsupervised learning algorithm.
- ✓ An image is made up of several intensity values known as Pixels.
- ✓ In a colored image, each pixel is of 3 bytes containing RGB (Red-Blue-Green) values having Red intensity value, then Blue and then Green intensity value for each pixel.

Algorithm:

Step 1: Upload an image

Step 2: Reshape the image

Convert the image to a 2D array of RGB values

Step 3: Prompt user for the number of colors (k) for compression

Step 4: Perform k-means clustering

Step 5: Get the cluster centers and labels

Step 6: Replace pixel values with cluster centers

Step 7: Display the original and compressed images

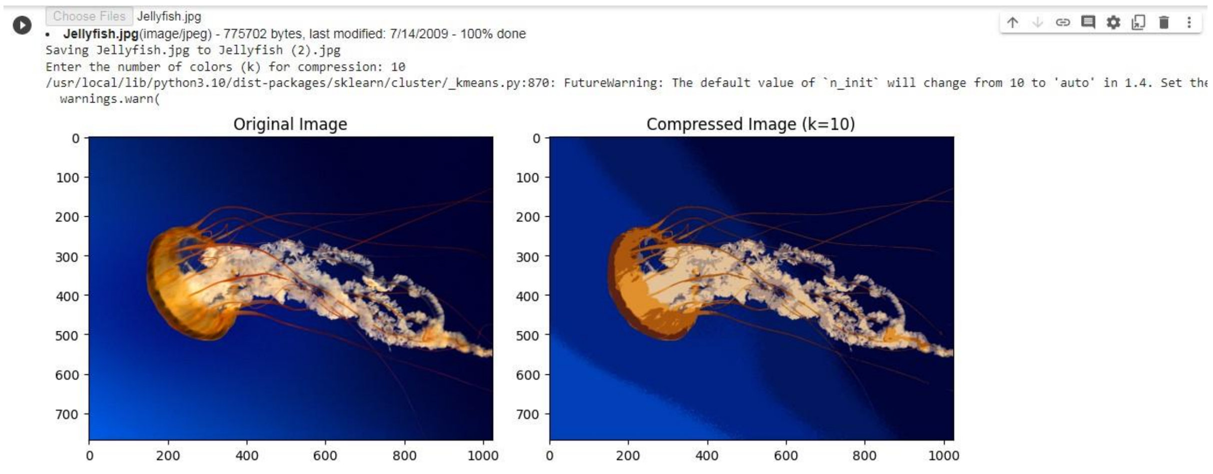
Step 7.1: Original Image with Axes

Step 7.2: Compressed Image without Axes

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from skimage import io
from google.colab import files
uploaded = files.upload()
for filename in uploaded.keys():
    image = io.imread(filename)
image_reshaped = image.reshape(-1, 3)
k = int(input("Enter the number of colors (k) for compression: "))
kmeans = KMeans(n_clusters=k)
kmeans.fit(image_reshaped)
cluster_centers = kmeans.cluster_centers_
labels = kmeans.labels_
compressed_image = cluster_centers[labels].reshape(image.shape)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(image)
plt.axis('on')  # Show axes for the original image
plt.subplot(1, 2, 2)\
plt.title(f"Compressed Image (k={k})")
plt.imshow(compressed_image.astype(np.uint8))
plt.axis('on')  # Hide axes for the compressed image
plt.tight_layout()
plt.show()
```

Output:



```
Choose Files Jellyfish.jpg
•  Jellyfish.jpg(image/jpeg) - 775702 bytes, last modified: 7/14/2009 - 100% done
Saving Jellyfish.jpg to Jellyfish (2).jpg
Enter the number of colors (k) for compression: 10
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the
  warnings.warn(
```

Observation(20)	
Record(5)	
Total(25)	

Result :

Thus the design of K-Nearest neighbour algorithm has been created successfully.

EX NO 8 CREATE LINEAR REGRESSION MODELS USING PYTHON
DATE:

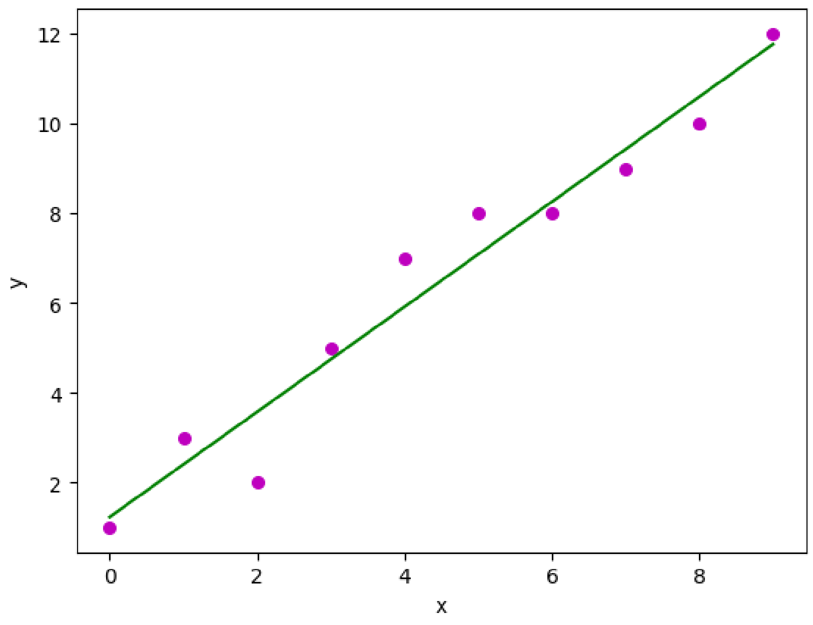
AIM
To design a program using linear regression models using python language.

ALGORITHM:
1. Import the necessary libraries (e.g., **numpy**, **scikit-learn**).
2. Collect and prepare your dataset.
3. Split the data into training and testing sets.
4. Initialize a linear regression model.
5. Train the model on the training data.
6. Use the trained model to make predictions on the testing data.
7. Evaluate the model using metrics like MSE, RMSE, MAE, and R^2.
8. Optionally, create visualizations to interpret model predictions.
9. Use the trained model for making predictions on new data.
10. Conclude your analysis and consider further actions.
11. Deploy the model if needed.
12. Document your code and analysis.

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
```



Observation (20)	
Record (5)	
Total (25)	

RESULT:

Thus, linear regression models are successfully written and output is verified.

EX NO 9 MODEL THE DECISION TREE FOR DATA CLASSIFICATION USING REAL-TIME DATASET
DATE:

AIM

To write a program using decision tree for data classification using real-time datasets.

ALGORITHM

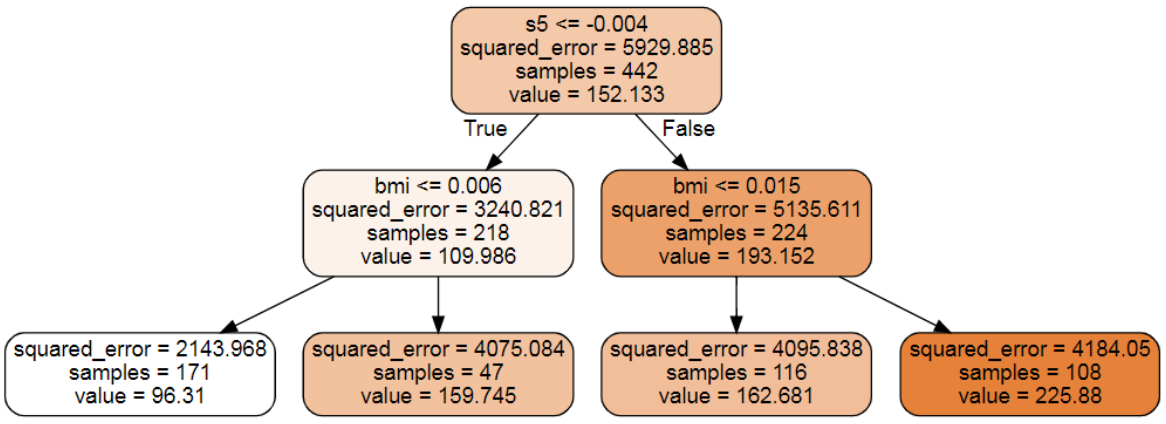
1. The root node of the tree is supposed to be the complete training dataset.
2. Determine the impurity of the data based on each feature present in the dataset. Impurity can be measured using metrics like the Gini index or entropy for classification and Mean squared error, Mean Absolute Error, friedman_mse, or Half Poisson deviance for regression.
3. Then selects the feature that results in the highest information gain or impurity reduction when splitting the data.
4. For each possible value of the selected feature, split the dataset into two subsets (left and right), one where the feature takes on that value, and another where it does not. The split should be designed to create subsets that are as pure as possible with respect to the target variable.
5. Based on the target variable, determine the impurity of each resulting subset.
6. For each subset, repeat steps 2–5 iteratively until a stopping condition is met. For example, the stopping condition could be a maximum tree depth, a minimum number of samples required to make a split or a minimum impurity threshold.
7. Assign the majority class label for classification tasks or the mean value for regression tasks for each terminal node (leaf node) in the tree.

PROGRAM

```
#Import the necessary libraries
from sklearn.datasets import load_diabetes
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import export_graphviz
from graphviz import Source

# Load the dataset
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target

# DecisionTreeRegressor
tree_reg = DecisionTreeRegressor(criterion = 'squared_error',
                                 max_depth=2)

tree_reg.fit(X, y)

# Plot the decision tree graph
```

```
export_graphviz(
    tree_reg,
    out_file="diabetes_tree.dot",
    feature_names=diabetes.feature_names,
    class_names=diabetes.target,
    rounded=True,
    filled=True
 )

with open("diabetes_tree.dot") as f:
    dot_graph = f.read()

Source(dot_graph)
```

OUTPUT



Observation (20)	
Record (5)	
Total (25)	

RESULT

Thus, decision tree for data classification using real-time datasets is done successfully.

Ex No.: 10 **DESIGN DATA CLASSIFICATION FOR REAL-TIME DATASETS USING SUPPORT VECTOR MACHINE**
Date:

Aim:
To design data classification for real-time datasets using Support Vector Machine.

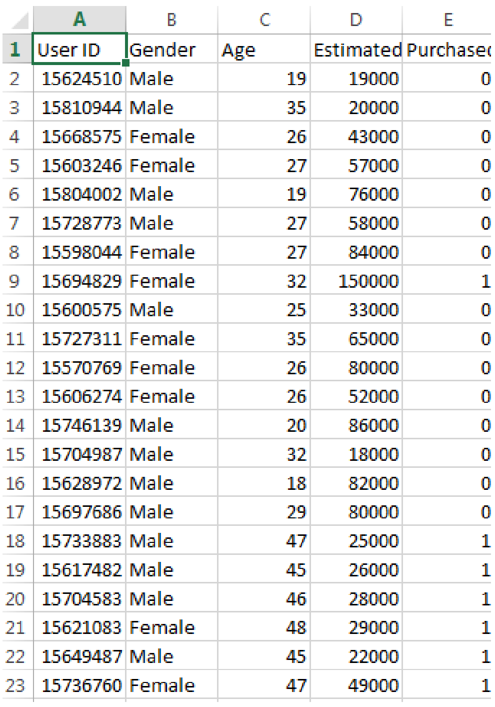
ALGORITHM:
1. Import the necessary libraries (e.g., **numpy**, **scikit-learn** for SVM).
2. Collect and preprocess your dataset.
3. Split the dataset into training and testing sets.
4. Initialize an SVM model, specifying the kernel type (e.g., linear, polynomial, or radial basis function).
5. Train the SVM model on the training data.
6. Fine-tune hyperparameters such as regularization strength (C) and kernel parameters using techniques like cross-validation.
7. Use the trained SVM to make predictions on the testing data.
8. Evaluate the SVM's performance using metrics like accuracy, precision, recall, and F1-score.
9. Optionally, visualize the decision boundaries and support vectors if you have a 2D dataset.
10. Interpret the results and make any necessary adjustments to improve the model's performance.

PROGRAM:
```
# Support Vector Machine
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the datasets
datasets = pd.read_csv('/content/sample_data/Social_Network_Ads.csv')
X = datasets.iloc[:, [2,3]].values
Y = datasets.iloc[:, 4].values
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, Y, test_size = 0.25, random_state = 0)
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_Train = sc_X.fit_transform(X_Train)
X_Test = sc_X.transform(X_Test)
# Fitting the classifier into the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_Train, Y_Train)
```

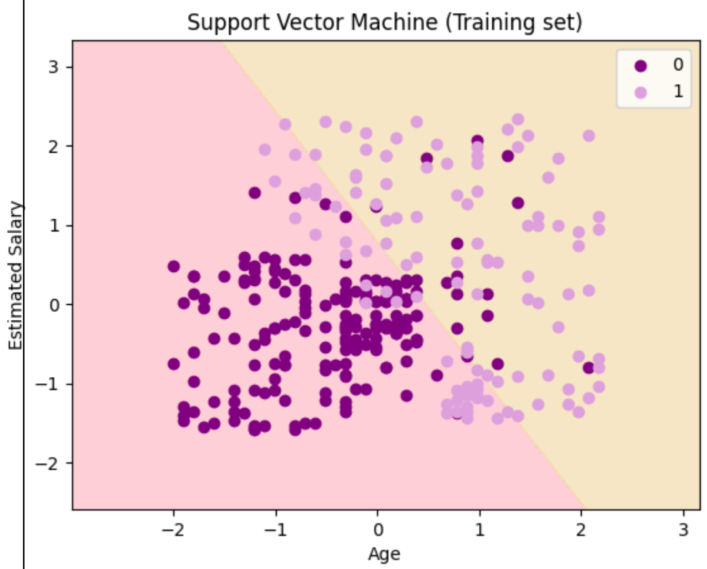
```
# Predicting the test set results
Y_Pred = classifier.predict(X_Test)
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_Test, Y_Pred)
# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_Set, Y_Set = X_Train, Y_Train
X1, X2 = np.meshgrid(np.arange(start = X_Set[:, 0].min() - 1, stop = X_Set[:, 0].max() + 1, step =
0.01),
                     np.arange(start = X_Set[:, 1].min() - 1, stop = X_Set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('pink', 'wheat')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(Y_Set)):
    plt.scatter(X_Set[Y_Set == j, 0], X_Set[Y_Set == j, 1],
                c = ListedColormap(('purple', 'plum'))(i), label = j)
plt.title('Support Vector Machine (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_Set, Y_Set = X_Test, Y_Test
X1, X2 = np.meshgrid(np.arange(start = X_Set[:, 0].min() - 1, stop = X_Set[:, 0].max() + 1, step =
0.01),
                     np.arange(start = X_Set[:, 1].min() - 1, stop = X_Set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('pink', 'wheat')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(Y_Set)):
    plt.scatter(X_Set[Y_Set == j, 0], X_Set[Y_Set == j, 1],
                c = ListedColormap(('purple', 'plum'))(i), label = j)
plt.title('Support Vector Machine (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

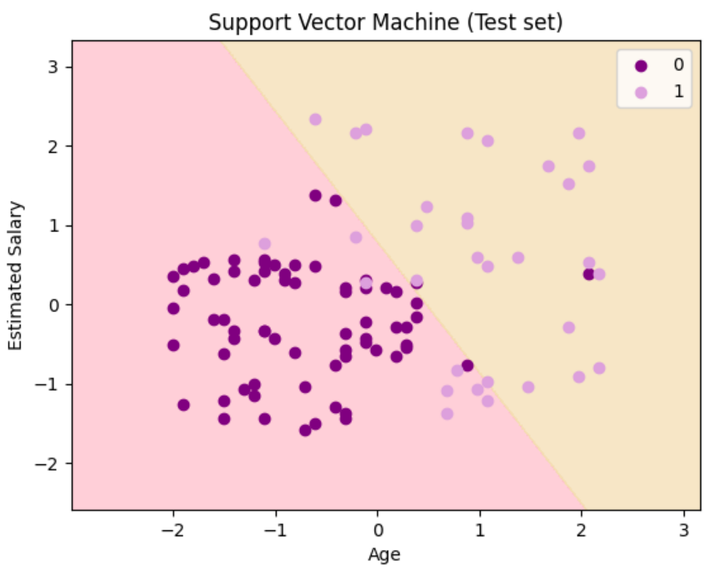
OUTPUT:

SAMPLE DATASETS



	A	B	C	D	E
1	User ID	Gender	Age	Estimated	Purchase
2	15624510	Male	19	19000	0
3	15810944	Male	35	20000	0
4	15668575	Female	26	43000	0
5	15603246	Female	27	57000	0
6	15804002	Male	19	76000	0
7	15728773	Male	27	58000	0
8	15598044	Female	27	84000	0
9	15694829	Female	32	150000	1
10	15600575	Male	25	33000	0
11	15727311	Female	35	65000	0
12	15570769	Female	26	80000	0
13	15606274	Female	26	52000	0
14	15746139	Male	20	86000	0
15	15704987	Male	32	18000	0
16	15628972	Male	18	82000	0
17	15697686	Male	29	80000	0
18	15733883	Male	47	25000	1
19	15617482	Male	45	26000	1
20	15704583	Male	46	28000	1
21	15621083	Female	48	29000	1
22	15649487	Male	45	22000	1
23	15736760	Female	47	49000	1





Observation (20)	
Record (5)	
Total (25)	

RESULT:

Thus, support vector machine is implemented in real-time datasets is successfully done.