

# College of Excellence, 2025-9th Rank Autonomous and Affiliated to Bharathiar University

**Accredited with A++ grade by NAAC, An ISO 9001: 2015 Certified Institution Peelamedu, Coimbatore-641004**

**DEPARTMENT OF COMPUTER SCIENCE (PG)**

**DATA MINING TECHNIQUES AND TOOLS LAB (MCS24P1)**

**2025-2026**



# College of Excellence, 2025-9th Rank Autonomous and Affiliated to Bharathiar University

**Accredited with A++ grade by NAAC, An ISO 9001: 2015 Certified Institution Peelamedu, Coimbatore-641004**

**DEPARTMENT OF COMPUTER SCIENCE (PG) DATA MINING TECHNIQUES AND TOOLS LAB (MCS24P1)**

**REGISTER NUMBER:**

Certified that this is a bonafide record work done by of I MSC (Computer Science) during the year 2025-2026.

**FACULTY INCHARGE HEAD OF THE DEPARTMENT**

Submitted for the practical examination held on at PSGR Krishnammal College for Women, Coimbatore.

**INTERNAL EXAMINER EXTERNAL EXAMINER**

# INDEX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.**  **NO.** | **DATE** | **TOPICS** | **PAGE**  **NO** | **SIGN** |
| **PYTHON PROGRAMS** | | | | |
| 1 |  | Linear Regression |  |  |
| 2 |  | Decision Tree |  |  |
| 3 |  | Pearson Correlation, Euclidean  Distance, Manhattan Distance |  |  |
| 4 |  | Hierarchical Clustering |  |  |
| 5 |  | BIRCH Clustering |  |  |
| 6 |  | Text Mining |  |  |
| **R PROGRAMS** | | | | |
| 7 |  | Data Exploration and Visualization |  |  |
| 8 |  | Classification of Support Vector  Machine (SVM) |  |  |
| 9 |  | Logistic Regression |  |  |
| 10 |  | DBSCAN Clustering |  |  |
| **DATA VISUALIZATION USING TABLEAU** | | | | |
| 11 |  | Data Exploration & Visualization |  |  |
| **DATA VISUALIZATION USING KNIME** | | | | |
| 12 |  | Decision Tree Classification |  |  |
| 13 |  | Apriori Algorithm |  |  |
| **APPLICATIONS** | | | | |
| 1 |  | Customer Buying Patterns using  Classification Methods |  |  |
| 2 |  | Credit Card Fraud Detection using  Supervised Algorithm |  |  |
| 3 |  | Breast Cancer Prediction using  Supervised Algorithm |  |  |
| 4 |  | Customer Segmentation using  Clustering |  |  |
| 5 |  | Loan Defaulters Prediction |  |  |
| 6 |  | Classification of Iris Dataset |  |  |

|  |  |
| --- | --- |
| **Ex.No: 1**  **Date:** | **LINEAR REGRESSION** |

**AIM**

To Demonstrate the following data preprocessing tasks using Python libraries.

1. Loading the dataset
2. Identifying the dependent and independent variables
3. Dealing with missing data

# ALGORITHM

**STEP 1:** Start the program.

**STEP 2:** Import necessary libraries (pandas, numpy, matplotlib, sklearn).

**STEP 3:** Load the diabetes dataset using sklearn.datasets.

**STEP 4:** Convert the dataset to a DataFrame and add the target column.

**STEP 5:** Check if any values are missing in the dataset.

**STEP 6:** If missing values exist, fill them with the column mean.

**STEP 7:** Select 'bmi' as the independent variable (X) and 'target' as the dependent variable (y).

**STEP 8:** Split the data into training and testing sets using train\_test\_split().

**STEP 9:** Train a Linear Regression model on the training data.

**STEP 10:** Predict the target values using the test data.

**STEP 11:** Calculate R² score and Mean Squared Error (MSE).

**STEP 12:** Plot the actual vs predicted values.

**STEP 13:** End the program.

# PROGRAM

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import r2\_score, mean\_squared\_error import matplotlib.pyplot as plt

import numpy as np import pandas as pd

from sklearn.datasets import load\_diabetes

from sklearn.model\_selection import train\_test\_split data = load\_diabetes()

df = pd.DataFrame(data.data, columns = data.feature\_names) df['target'] = data.target

if df.isnull().values.any():

df.fillna(df.mean()) X = df[['bmi']]

y = df['target'] print(data.feature\_names)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y) model = LinearRegression() model.fit(X\_train,y\_train)

y\_pred = model.predict(X\_test) r2 = r2\_score(y\_test,y\_pred) print(f"r2 score is {round(r2,3)}")

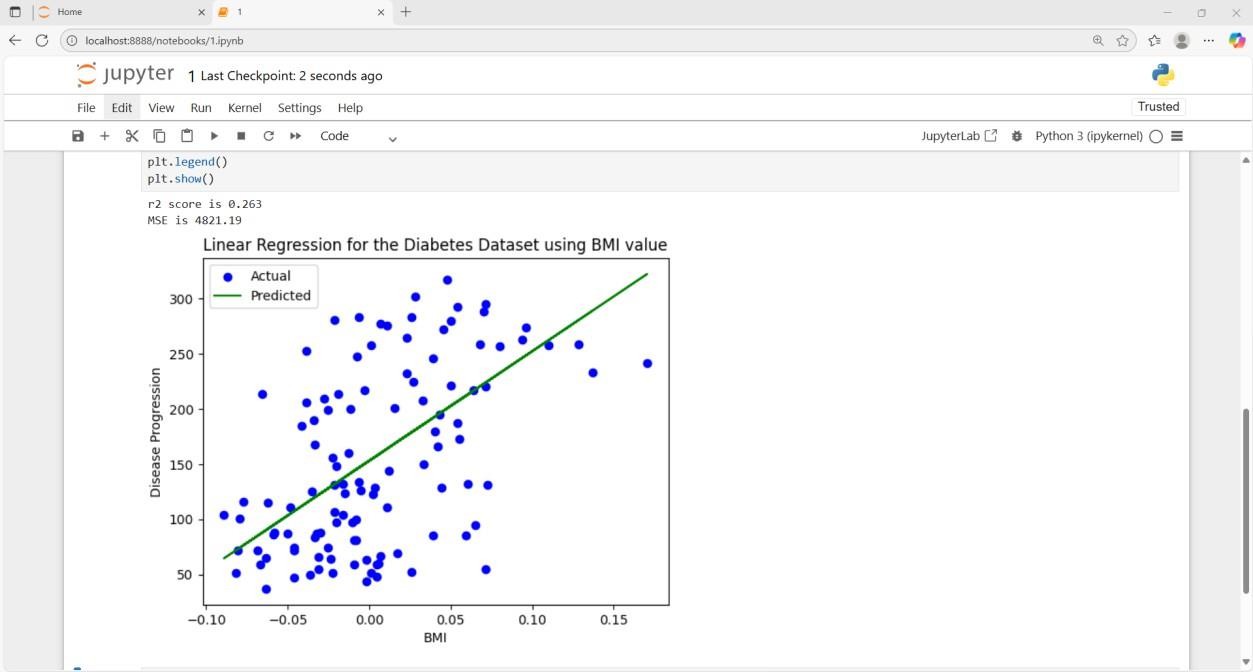
mse = mean\_squared\_error(y\_test,y\_pred) print(f"MSE is {round(mse,3)}")

plt.scatter(X\_test,y\_test,c="blue",label="Actual") # plots the test set using scatter plot plt.plot(X\_test,y\_pred,color = "green", label="Predicted") # plots the predicted line using plot plt.title("Linear Regression for the Diabetes Dataset using BMI value")

plt.xlabel("BMI") plt.ylabel("Disease Progression") plt.legend()

plt.show()

# OUTPUT

r2 score is 0.37 MSE is 4224.086

# RESULT

Thus, the above python program to perform linear regression on the diabetes dataset has been verified and executed successfully.

|  |  |
| --- | --- |
| **Ex.No:2**  **Date:** | **DECISION TREE** |

# AIM

To Demonstrate the following data preprocessing tasks using Python library

1. Dealing with categorical data
2. Scaling the features
3. Splitting dataset into Training and Testing Sets

# ALGORITHM

**STEP 1**: Start the program.

**STEP 2**: Import required libraries (pandas, sklearn, matplotlib). **STEP 3**: Create a dataset with columns: Age, Gender, and Eligibility. **STEP 4**: Encode the 'Gender' column using LabelEncoder.

**STEP 5**: Set 'Age' and 'Gender' as independent variables (X) and 'Eligibility' as the dependent variable (y).

**STEP 6**: Apply StandardScaler to scale the feature values.

**STEP 7**: Split the dataset into training and testing sets using train\_test\_split().

**STEP 8**: Train a Decision Tree Classifier on the training data.

**STEP 9**: Take user input for Age and Gender.

**STEP 10**: Encode and scale the user input using the same encoders and scalers.

**STEP 11**: Predict the eligibility and display the result.

**STEP 12**: End the program.

# PROGRAM

from sklearn.tree import DecisionTreeClassifier, plot\_tree from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, LabelEncoder import matplotlib.pyplot as plt

import pandas as pd import numpy as np data = {

'Age': [12,13,14,15,17,19,20,21,29,45],

'Gender': ["F","F","F","M","M","F","M","M","F","F"],

'Eligibility' : ["No","No","No","No","No","Yes","Yes","Yes","Yes","Yes"]

}

df = pd.DataFrame(data) le = LabelEncoder()

df['Gender'] = le.fit\_transform(df['Gender']) X = df[['Age','Gender']]

y = df['Eligibility']

scaler = StandardScaler() X\_scaled = scaler.fit\_transform(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled,y, test\_size=0.3) model = DecisionTreeClassifier()

model.fit(X\_train,y\_train)

age = float(input("Enter age: ")) gender = input("Enter gender: ")

“””

I commented out the following as this is optional to display the decision tree (based on the ques)

plt.figure(figsize=(10,3)) plot\_tree(model,

feature\_names = ["Age","Gender"],

class\_names = ["No","Yes"]) “””

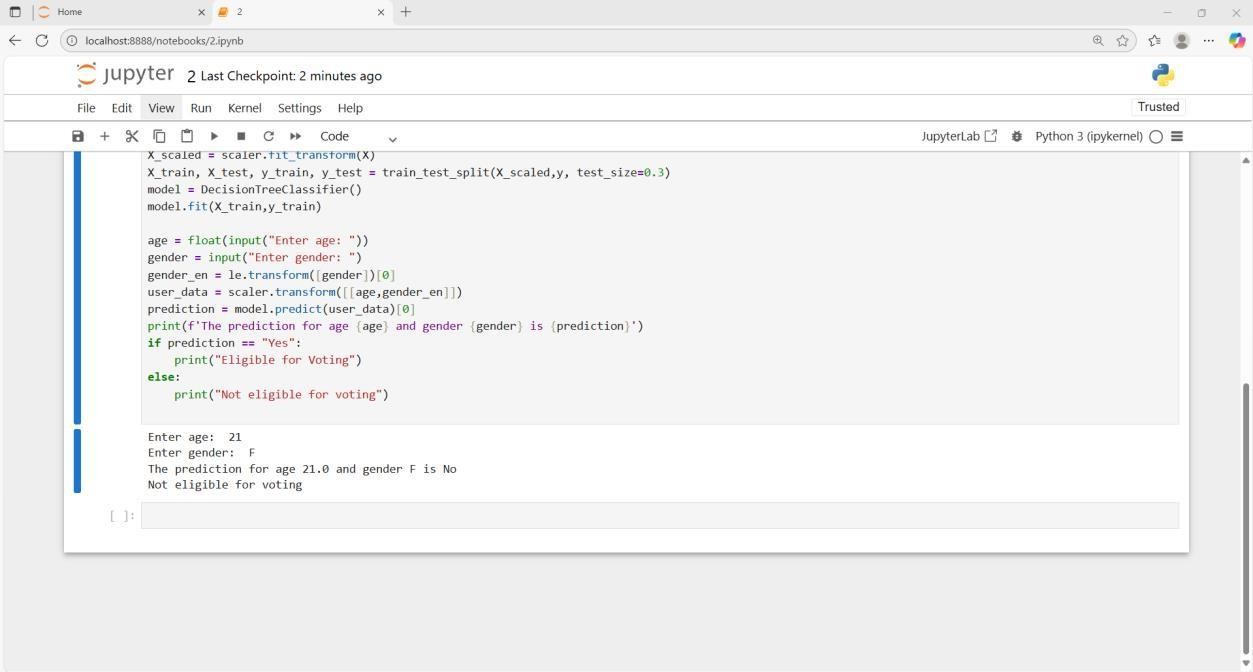
gender\_en = le.transform([gender])[0] user\_data = scaler.transform([[age,gender\_en]]) prediction = model.predict(user\_data)[0]

print(f'The prediction for age {age} and gender {gender} is {prediction}') if prediction == "Yes":

print("Eligible for Voting") else:

print("Not eligible for voting")

# OUTPUT

****

**RESULT**

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

|  |  |
| --- | --- |
| **Ex.No: 3**  **Date:** | **PEARSON’S CORRELATION, EUCLIDEAN DISTANCE, MANHATTAN DISTANCE** |

# AIM

To Demonstrate the following similarity measures in Python:

* 1. Pearson’s Correlation
  2. Euclidean Distance
  3. Manhattan Distance

# ALGORITHM

**STEP 1**: Start the program.

**STEP 2**: Import necessary libraries (pandas, numpy, scipy).

**STEP 3**: Load the dataset containing 'Marks' and 'Hours'.

**STEP 4**: Remove missing values if any.

**STEP 5**: Calculate Pearson’s correlation coefficient between 'Marks' and 'Hours'.

**STEP 6**: Interpret the correlation value.

**STEP 7**: Define two data points as numpy arrays.

**STEP 8**: Compute the Euclidean distance between the two points. **STEP 9**: Compute the Manhattan distance between the two points. **STEP 10**: Display the correlation and distance values.

**STEP 11**: End the program.

# PROGRAM

import pandas as pd

from scipy.stats import pearsonr

df = pd.read\_csv("C:/Users/1MSCCS15/Desktop/study\_hr-marks.csv").dropna() var1 = df['Marks']

var2 = df['Hours']

correlation, p\_value = pearsonr(var1, var2) print(f'Pearson Correlation {round(correlation,3)}') if correlation >= 0.5:

print("The relationship between two variables are strongly related: Positive Correlation") elif correlation < 0.5:

print("The relationship between two variables are not strongly related: Negative Correlation") else:

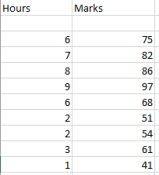
print("No correlation") # The distance measures import numpy as np

import matplotlib.pyplot as plt p1 = np.array([2, 3])

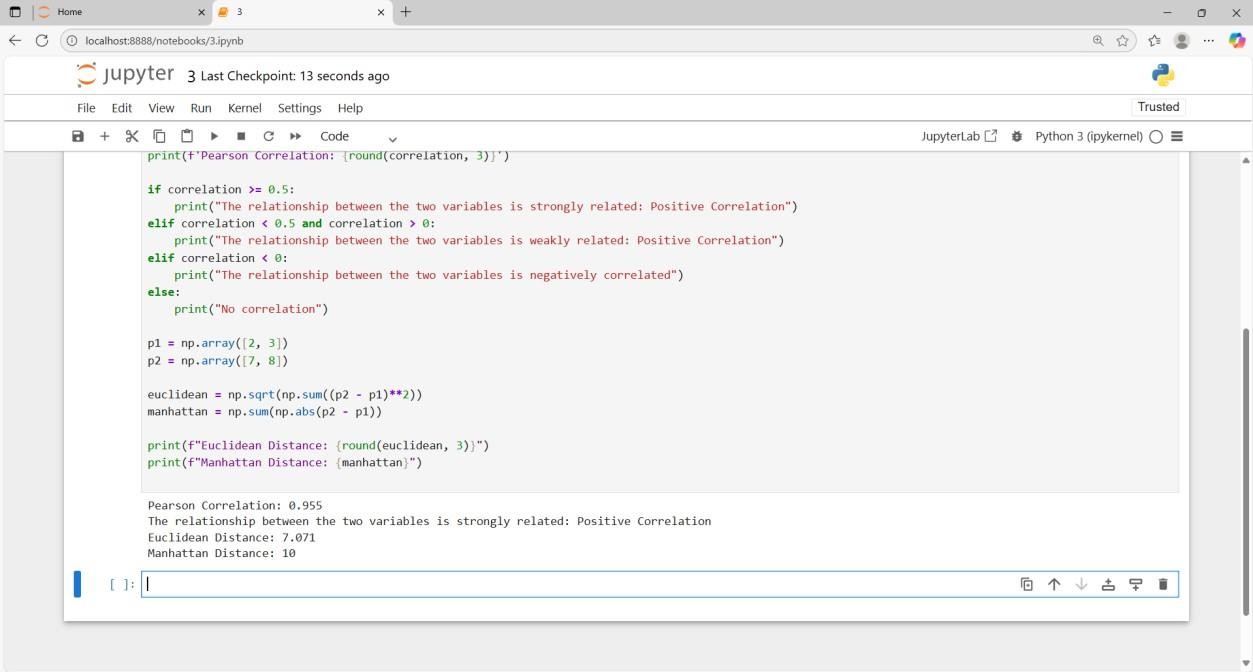
p2 = np.array([7, 8])

euclidean = np.sqrt(np.sum((p2 - p1)\*\*2)) manhattan = np.sum(np.abs(p2 - p1)) print(f"Euclidean Distance: {round(euclidean,3)}") print(f"Manhattan Distance: {manhattan}")

**SAMPLE DATASET**

****

**OUTPUT**

****

**RESULT**

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

|  |  |
| --- | --- |
| **Ex.No: 4**  **Date:** | **HIERARCHICAL CLUSTERING** |

# AIM

To Experiment on Hierarchical Data Clustering algorithms on weather dataset.

# ALGORITHM

**STEP 1**: Start the program.

**STEP 2**: Import necessary libraries (pandas, scipy.cluster, matplotlib).

**STEP 3**: Create a dataset with weather features: Temperature, Humidity, WindSpeed, Pressure.

**STEP 4**: Apply the hierarchical clustering using the 'centroid' linkage method.

**STEP 5**: Generate a linkage matrix using the linkage() function. **STEP 6**: Plot a dendrogram to visualize the clustering hierarchy. **STEP 7**: Display the dendrogram.

**STEP 8**: End the program.

# PROGRAM

import numpy as np import pandas as pd

from scipy.cluster.hierarchy import linkage, dendrogram import matplotlib.pyplot as plt

data = pd.DataFrame({ 'Temperature': [30, 22, 25, 35, 28],

'Humidity': [40, 85, 70, 30, 60],

'WindSpeed': [10, 5, 7, 8, 6],

'Pressure': [1012, 1005, 1008, 1013, 1010]

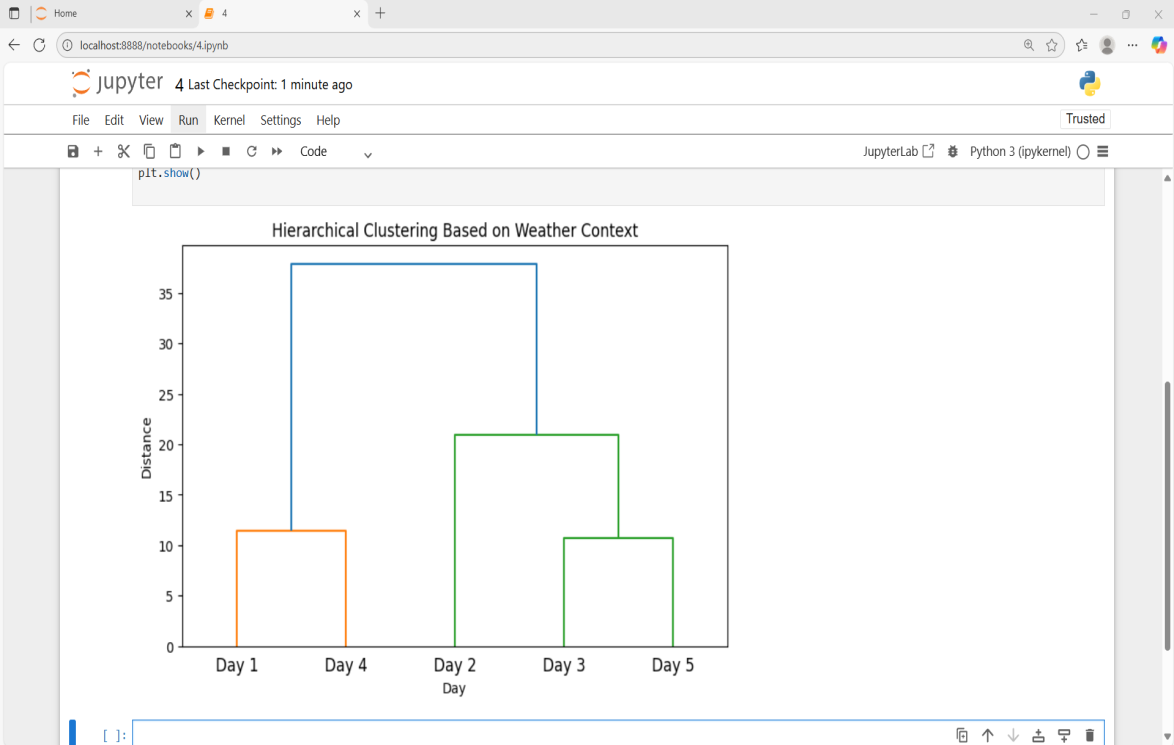
}, index=["Day 1", "Day 2", "Day 3", "Day 4", "Day 5"]) linked = linkage(data, method='centroid', metric="euclidean") plt.figure(figsize=(8, 5))

dendrogram(linked, labels=data.index)

plt.title("Hierarchical Clustering Based on Weather Context") plt.xlabel("Day")

plt.ylabel("Distance") plt.show()

# OUTPUT

****

**RESULT**

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

|  |  |
| --- | --- |
| **Ex.No: 5**  **Date:** | **BIRCH CLUSTERING** |

# AIM

To Write a Python code to perform clustering using the BIRCH algorithm.

# ALGORITHM

**STEP 1**: Start the program.

**STEP 2**: Import required libraries (pandas, numpy, matplotlib, sklearn).

**STEP 3**: Load the temperature dataset from a CSV file.

**STEP 4**: Select 'temperature' and 'humidity' as features.

**STEP 5**: Initialize and apply the BIRCH algorithm with desired number of clusters.

**STEP 6**: Predict the cluster labels for each data point.

**STEP 7**: Add cluster labels to the dataset.

**STEP 8**: Take a new data point and predict which cluster it belongs to.

**STEP 9**: Visualize the clusters along with the new data point using a scatter plot.

**STEP 10**: End the program.

# PROGRAM

import pandas as pd import numpy as np

import matplotlib.pyplot as plt from sklearn.cluster import Birch

data = pd.read\_csv("temperature\_data.csv") df = pd.DataFrame(data)

X = df[["temperature", "humidity"]] birch = Birch(n\_clusters=3) birch.fit(X)

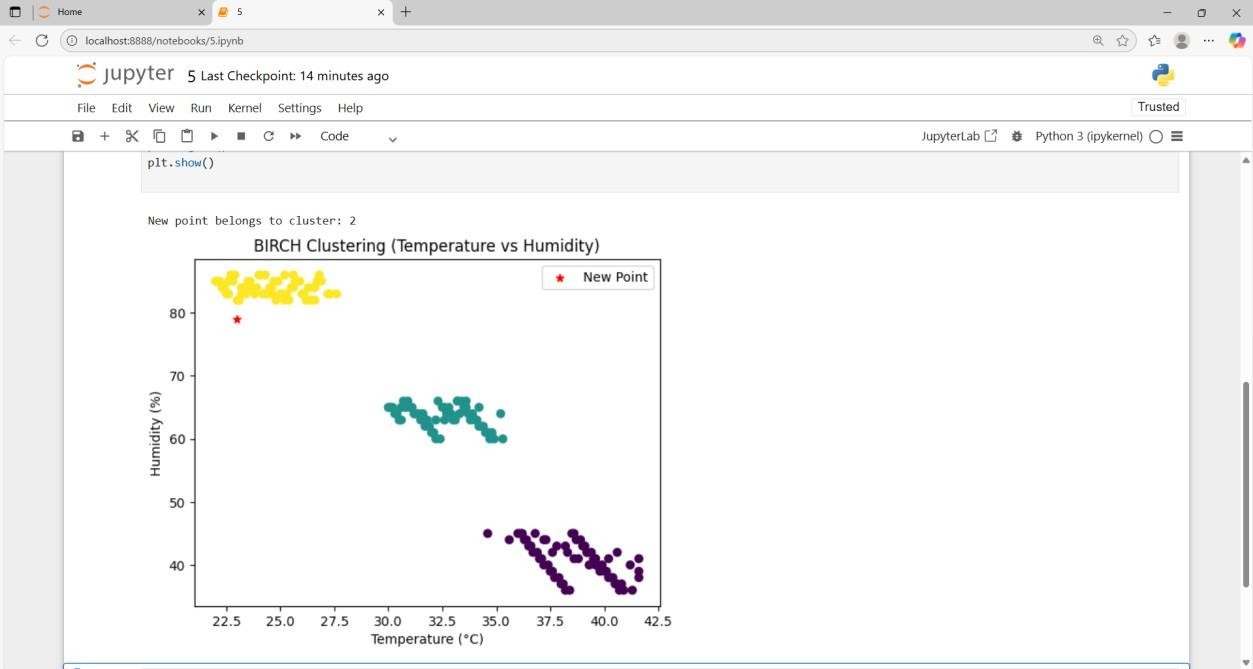
df["cluster"] = birch.predict(X) print("Clustered Data:\n", df) new\_point = np.array([[28, 55]]) new\_cluster = birch.predict(new\_point)

print("\nNew point belongs to cluster:", new\_cluster[0]) plt.scatter(df["temperature"], df["humidity"],c=df["cluster"], cmap="viridis") plt.scatter(new\_point[0,0], new\_point[0,1], c="red", marker="\*",label="New Point") plt.title("BIRCH Clustering (Temperature vs Humidity)")

plt.xlabel("Temperature (°C)") plt.ylabel("Humidity (%)") plt.legend()

plt.show()

# OUTPUT

****

**RESULT**

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

|  |  |
| --- | --- |
| **Ex.No: 6**  **Date:** | **TEXT MINING** |

# AIM

To write a Python code to Implement Text Mining for the corpus data.

# ALGORITHM

**Step 1**: Import the required libraries (nltk, sklearn, pandas).

**Step 2**: Download necessary resources from nltk such as punkt and stopwords.

**Step 3**: Define the text corpus containing multiple documents.

**Step 4**: Create a preprocessing function to lowercase the text, tokenize it, remove stopwords, keep only alphabetic words, and apply stemming.

**Step 5**: Apply the preprocessing function to each document in the corpus to obtain the cleaned corpus.

**Step 6**: Initialize the TfidfVectorizer and transform the cleaned corpus into a TF-IDF matrix. **Step 7**: Combine all words from the cleaned corpus and calculate their frequency distribution using nltk.FreqDist.

**Step 8**: Display the top ten most frequent words with their frequencies.

# PROGRAM

from sklearn.feature\_extraction.text import TfidfVectorizer, ENGLISH\_STOP\_WORDS from nltk.stem import PorterStemmer

import re

from collections import Counter import pandas as pd

corpus = pd.read\_csv("user\_reviews.csv").dropna() corpus = corpus["Translated\_Review"][1:151] stemmer = PorterStemmer()

def preprocess(text):

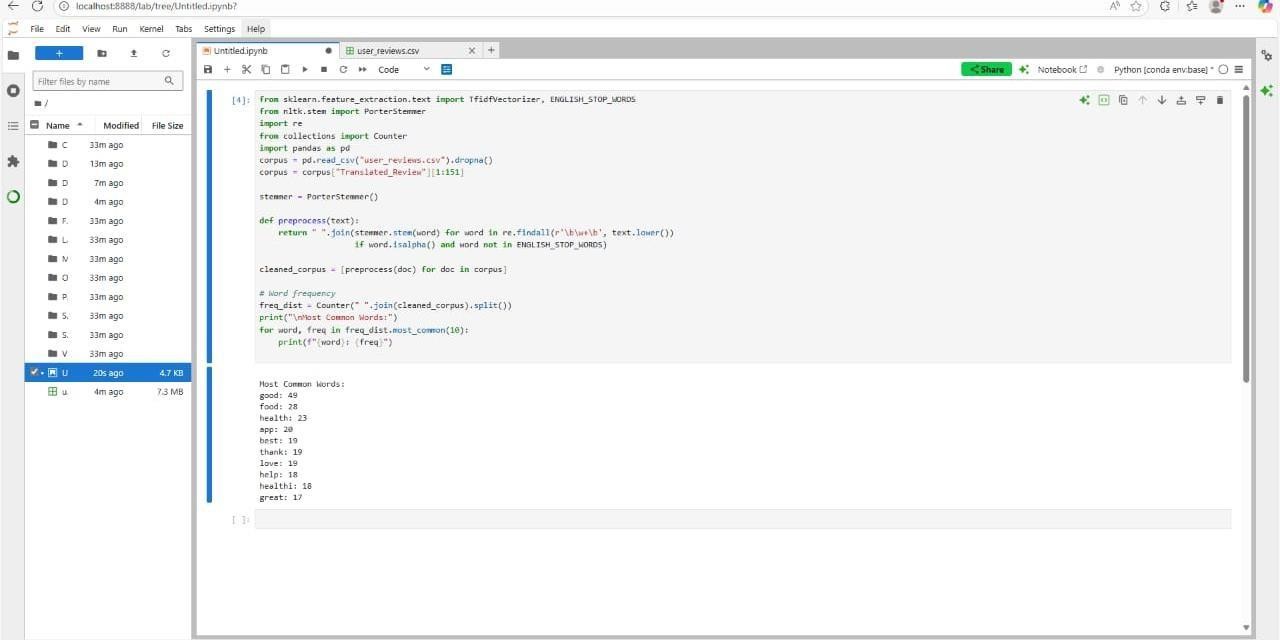
return " ".join(stemmer.stem(word) for word in re.findall(r'\b\w+\b', text.lower()) if word.isalpha() and word not in ENGLISH\_STOP\_WORDS)

cleaned\_corpus = [preprocess(doc) for doc in corpus] # Word frequency

freq\_dist = Counter(" ".join(cleaned\_corpus).split()) print("\nMost Common Words:")

for word, freq in freq\_dist.most\_common(10): print(f"{word}: {freq}")

# OUTPUT

****

**RESULT**

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

|  |  |
| --- | --- |
| **Ex.No: 7**  **Date:** | **DATA EXPLORATION AND VISUALIZATION** |

# AIM

To perform data exploration and visualization of the iris dataset and implement various statistical operations in R.

# ALGORITHM

**STEP 1:** Start the program.

**STEP 2:** Load and view the dataset: load the iris dataset to perform analysis.

**STEP 3:** Check dataset dimensions and structure: get the number of rows and columns and inspect the internal structure.

**STEP 4:** Retrieve column names: get the names of the dataset’s columns.

**STEP 5:** Display summary statistics: Generate summary statistics for all variables

**STEP 6**: Compute central tendencies: calculate the mean, median, and range for Sepal.Length.

**STEP 7:** Covariance calculation: compute the covariance between Sepal.Length and Sepal.Width.

**STEP 8**: Density plot: plot the density distribution of Sepal.Length**.**

**STEP 9**: Visualize Data Relationships: Generate a scatterplot matrix for all numeric variables.

**STEP 10**: Create a pie chart of species: Visualize the proportions of different species using a pie chart.

**STEP 11:** Stop the program.

# PROGRAM

* View(iris)
* dim(iris)

[1] 150 5

* names(iris)

[1] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" "Species"

* structure(iris)

Sepal.Length Sepal.Width Petal.Length Petal.Width Species

1 5.1 3.5 1.4 0.2 setosa

2 4.9 3.0 1.4 0.2 setosa

3 4.7 3.2 1.3 0.2 setosa

4 4.6 3.1 1.5 0.2 setosa

5 5.0 3.6 1.4 0.2 setosa

6 5.4 3.9 1.7 0.4 setosa

7 4.6 3.4 1.4 0.3 setosa

8 5.0 3.4 1.5 0.2 setosa

9 4.4 2.9 1.4 0.2 setosa

10 4.9 3.1 1.5 0.1 setosa

11 5.4 3.7 1.5 0.2 setosa

12 4.8 3.4 1.6 0.2 setosa

1314 4.3 3.0 1.1 0.1 setosa

15 5.8 4.0 1.2 0.2 setosa

16 5.7 4.4 1.5 0.4 setosa

17 5.4 3.9 1.3 0.4 setosa

18 5.1 3.5 1.4 0.3 setosa

19 5.7 3.8 1.7 0.3 setosa

20 5.1 3.8 1.5 0.3 setosa

21 5.4 3.4 1.7 0.2 setosa

22 5.1 3.7 1.5 0.4 setosa

23 4.6 3.6 1.0 0.2 setosa

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 24 | 5.1 | 3.3 | 1.7 | 0.5 | setosa |
| 25 | 4.8 | 3.4 | 1.9 | 0.2 | setosa |
| 26 | 5.0 | 3.0 | 1.6 | 0.2 | setosa |
| 27 | 5.0 | 3.4 | 1.6 | 0.4 | setosa |
| 28 | 5.2 | 3.5 | 1.5 | 0.2 | setosa |
| 29 | 5.2 | 3.4 | 1.4 | 0.2 | setosa |
| 30 | 4.7 | 3.2 | 1.6 | 0.2 | setosa |
| 31 | 4.8 | 3.1 | 1.6 | 0.2 | setosa |
| 32 | 5.4 | 3.4 | 1.5 | 0.4 | setosa |
| 33 | 5.2 | 4.1 | 1.5 | 0.1 | setosa |
| 34 | 5.5 | 4.2 | 1.4 | 0.2 | setosa |
| 35 | 4.9 | 3.1 | 1.5 | 0.2 | setosa |
| 36 | 5.0 | 3.2 | 1.2 | 0.2 | setosa |
| 37 | 5.5 | 3.5 | 1.3 | 0.2 | setosa |
| 38 | 4.9 | 3.6 | 1.4 | 0.1 | setosa |
| 39 | 4.4 | 3.0 | 1.3 | 0.2 | setosa |
| 40 | 5.1 | 3.4 | 1.5 | 0.2 | setosa |
| 41 | 5.0 | 3.5 | 1.3 | 0.3 | setosa |
| 42 | 4.5 | 2.3 | 1.3 | 0.3 | setosa |
| 43 | 4.4 | 3.2 | 1.3 | 0.2 | setosa |
| 44 | 5.0 | 3.5 | 1.6 | 0.6 | setosa |
| 45 | 5.1 | 3.8 | 1.9 | 0.4 | setosa |
| 46 | 4.8 | 3.0 | 1.4 | 0.3 | setosa |
| 47 | 5.1 | 3.8 | 1.6 | 0.2 | setosa |
| 48 | 4.6 | 3.2 | 1.4 | 0.2 | setosa |
| 49 | 5.3 | 3.7 | 1.5 | 0.2 | setosa |
| 50 | 5.0 | 3.3 | 1.4 | 0.2 | setosa |
| 51 | 7.0 | 3.2 | 4.7 | 1.4 | versicolor |
| 52 | 6.4 | 3.2 | 4.5 | 1.5 | versicolor |
| 53 | 6.9 | 3.1 | 4.9 | 1.5 | versicolor |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 54 | 5.5 | 2.3 | 4.0 | 1.3 versicolor |
| 55 | 6.5 | 2.8 | 4.6 | 1.5 versicolor |
| 56 | 5.7 | 2.8 | 4.5 | 1.3 versicolor |
| 57 | 6.3 | 3.3 | 4.7 | 1.6 versicolor |
| 58 | 4.9 | 2.4 | 3.3 | 1.0 versicolor |
| 59 | 6.6 | 2.9 | 4.6 | 1.3 versicolor |
| 60 | 5.2 | 2.7 | 3.9 | 1.4 versicolor |
| 61 | 5.0 | 2.0 | 3.5 | 1.0 versicolor |
| 62 | 5.9 | 3.0 | 4.2 | 1.5 versicolor |
| 63 | 6.0 | 2.2 | 4.0 | 1.0 versicolor |
| 64 | 6.1 | 2.9 | 4.7 | 1.4 versicolor |
| 65 | 5.6 | 2.9 | 3.6 | 1.3 versicolor |
| 66 | 6.7 | 3.1 | 4.4 | 1.4 versicolor |
| 67 | 5.6 | 3.0 | 4.5 | 1.5 versicolor |
| 68 | 5.8 | 2.7 | 4.1 | 1.0 versicolor |
| 69 | 6.2 | 2.2 | 4.5 | 1.5 versicolor |
| 70 | 5.6 | 2.5 | 3.9 | 1.1 versicolor |
| 71 | 5.9 | 3.2 | 4.8 | 1.8 versicolor |
| 72 | 6.1 | 2.8 | 4.0 | 1.3 versicolor |
| 73 | 6.3 | 2.5 | 4.9 | 1.5 versicolor |
| 74 | 6.1 | 2.8 | 4.7 | 1.2 versicolor |
| 75 | 6.4 | 2.9 | 4.3 | 1.3 versicolor |
| 76 | 6.6 | 3.0 | 4.4 | 1.4 versicolor |
| 77 | 6.8 | 2.8 | 4.8 | 1.4 versicolor |
| 78 | 6.7 | 3.0 | 5.0 | 1.7 versicolor |
| 79 | 6.0 | 2.9 | 4.5 | 1.5 versicolor |
| 80 | 5.7 | 2.6 | 3.5 | 1.0 versicolor |
| 81 | 5.5 | 2.4 | 3.8 | 1.1 versicolor |
| 82 | 5.5 | 2.4 | 3.7 | 1.0 versicolor |
| 83 | 5.8 | 2.7 | 3.9 | 1.2 versicolor |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 84 | 6.0 | 2.7 | 5.1 | 1.6 versicolor |
| 85 | 5.4 | 3.0 | 4.5 | 1.5 versicolor |
| 86 | 6.0 | 3.4 | 4.5 | 1.6 versicolor |
| 87 | 6.7 | 3.1 | 4.7 | 1.5 versicolor |
| 88 | 6.3 | 2.3 | 4.4 | 1.3 versicolor |
| 89 | 5.6 | 3.0 | 4.1 | 1.3 versicolor |
| 90 | 5.5 | 2.5 | 4.0 | 1.3 versicolor |
| 91 | 5.5 | 2.6 | 4.4 | 1.2 versicolor |
| 92 | 6.1 | 3.0 | 4.6 | 1.4 versicolor |
| 93 | 5.8 | 2.6 | 4.0 | 1.2 versicolor |
| 94 | 5.0 | 2.3 | 3.3 | 1.0 versicolor |
| 95 | 5.6 | 2.7 | 4.2 | 1.3 versicolor |
| 96 | 5.7 | 3.0 | 4.2 | 1.2 versicolor |
| 97 | 5.7 | 2.9 | 4.2 | 1.3 versicolor |
| 98 | 6.2 | 2.9 | 4.3 | 1.3 versicolor |
| 99 | 5.1 | 2.5 | 3.0 | 1.1 versicolor |
| 100 | 5.7 | 2.8 | 4.1 | 1.3 versicolor |
| 101 | 6.3 | 3.3 | 6.0 | 2.5 virginica |
| 102 | 5.8 | 2.7 | 5.1 | 1.9 virginica |
| 103 | 7.1 | 3.0 | 5.9 | 2.1 virginica |
| 104 | 6.3 | 2.9 | 5.6 | 1.8 virginica |
| 105 | 6.5 | 3.0 | 5.8 | 2.2 virginica |
| 106 | 7.6 | 3.0 | 6.6 | 2.1 virginica |
| 107 | 4.9 | 2.5 | 4.5 | 1.7 virginica |
| 108 | 7.3 | 2.9 | 6.3 | 1.8 virginica |
| 109 | 6.7 | 2.5 | 5.8 | 1.8 virginica |
| 110 | 7.2 | 3.6 | 6.1 | 2.5 virginica |
| 111 | 6.5 | 3.2 | 5.1 | 2.0 virginica |
| 112 | 6.4 | 2.7 | 5.3 | 1.9 virginica |
| 113 | 6.8 | 3.0 | 5.5 | 2.1 virginica |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 114 | 5.7 | 2.5 | 5.0 | 2.0 | virginica |
| 115 | 5.8 | 2.8 | 5.1 | 2.4 | virginica |
| 116 | 6.4 | 3.2 | 5.3 | 2.3 | virginica |
| 117 | 6.5 | 3.0 | 5.5 | 1.8 | virginica |
| 118 | 7.7 | 3.8 | 6.7 | 2.2 | virginica |
| 119 | 7.7 | 2.6 | 6.9 | 2.3 | virginica |
| 120 | 6.0 | 2.2 | 5.0 | 1.5 | virginica |
| 121 | 6.9 | 3.2 | 5.7 | 2.3 | virginica |
| 122 | 5.6 | 2.8 | 4.9 | 2.0 | virginica |
| 123 | 7.7 | 2.8 | 6.7 | 2.0 | virginica |
| 124 | 6.3 | 2.7 | 4.9 | 1.8 | virginica |
| 125 | 6.7 | 3.3 | 5.7 | 2.1 | virginica |
| 126 | 7.2 | 3.2 | 6.0 | 1.8 | virginica |
| 127 | 6.2 | 2.8 | 4.8 | 1.8 | virginica |
| 128 | 6.1 | 3.0 | 4.9 | 1.8 | virginica |
| 129 | 6.4 | 2.8 | 5.6 | 2.1 | virginica |
| 130 | 7.2 | 3.0 | 5.8 | 1.6 | virginica |
| 131 | 7.4 | 2.8 | 6.1 | 1.9 | virginica |
| 132 | 7.9 | 3.8 | 6.4 | 2.0 | virginica |
| 133 | 6.4 | 2.8 | 5.6 | 2.2 | virginica |
| 134 | 6.3 | 2.8 | 5.1 | 1.5 | virginica |
| 135 | 6.1 | 2.6 | 5.6 | 1.4 | virginica |
| 136 | 7.7 | 3.0 | 6.1 | 2.3 | virginica |
| 137 | 6.3 | 3.4 | 5.6 | 2.4 | virginica |
| 138 | 6.4 | 3.1 | 5.5 | 1.8 | virginica |
| 139 | 6.0 | 3.0 | 4.8 | 1.8 | virginica |
| 140 | 6.9 | 3.1 | 5.4 | 2.1 | virginica |
| 141 | 6.7 | 3.1 | 5.6 | 2.4 | virginica |
| 142 | 6.9 | 3.1 | 5.1 | 2.3 | virginica |
| 143 | 5.8 | 2.7 | 5.1 | 1.9 | virginica |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 144 | 6.8 | 3.2 | 5.9 | 2.3 | virginica |
| 145 | 6.7 | 3.3 | 5.7 | 2.5 | virginica |
| 146 | 6.7 | 3.0 | 5.2 | 2.3 | virginica |
| 147 | 6.3 | 2.5 | 5.0 | 1.9 | virginica |
| 148 | 6.5 | 3.0 | 5.2 | 2.0 | virginica |
| 149 | 6.2 | 3.4 | 5.4 | 2.3 | virginica |
| 150 | 5.9 | 3.0 | 5.1 | 1.8 | virginica |
| * str(iris) | |  |  |  |  |

'data.frame': 150 obs. of 5 variables:

$ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...

$ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...

$ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...

$ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...

$ Species : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...

* attributes(iris)

$names

[1] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" "Species"

$class

[1] "data.frame"

$row.names

[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [19] | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 |
| [37] | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 |
| [55] | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 |
| [73] | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 |
| [91] | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 100 101 102 103 104 105 106 107 108 | | | | | | | | | |

[109] 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126

[127] 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144

[145] 145 146 147 148 149 150

* head(iris)

Sepal.Length Sepal.Width Petal.Length Petal.Width Species

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 2 | 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 3 | 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 4 | 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 5 | 5.0 | 3.6 | 1.4 | 0.2 | setosa |
| 6 | 5.4 | 3.9 | 1.7 | 0.4 | setosa |

* tail(iris)

Sepal.Length Sepal.Width Petal.Length Petal.Width Species

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 145 | 6.7 | 3.3 | 5.7 | 2.5 virginica |
| 146 | 6.7 | 3.0 | 5.2 | 2.3 virginica |
| 147 | 6.3 | 2.5 | 5.0 | 1.9 virginica |
| 148 | 6.5 | 3.0 | 5.2 | 2.0 virginica |
| 149 | 6.2 | 3.4 | 5.4 | 2.3 virginica |
| 150 | 5.9 | 3.0 | 5.1 | 1.8 virginica |

* summary(iris)

Sepal.Length Sepal.Width Petal.Length Petal.Width Min. :4.300 Min. :2.000 Min. :1.000 Min. :0.100 1st Qu.:5.100 1st Qu.:2.800 1st Qu.:1.600 1st Qu.:0.300

Median :5.800 Median :3.000 Median :4.350 Median :1.300

Mean :5.843 Mean :3.057 Mean :3.758 Mean :1.199 3rd Qu.:6.400 3rd Qu.:3.300 3rd Qu.:5.100 3rd Qu.:1.800 Max.

:7.900 Max. :4.400 Max. :6.900 Max. :2.500

Species setosa :50 versicolor:50 virginica :50

* mean(iris$Sepal.Length)

[1] 5.843333

* range(iris$Sepal.Length)

[1] 4.3 7.9

* median(iris$Sepal.Length)

[1] 5.8

* cov(iris$Sepal.Length, iris$Sepal.Width)

[1] -0.042434

* plot(density(iris$Sepal.Length))
* hist(iris$Sepal.Length)
* table(iris$Sepal.Length)

4.3 4.4 4.5 4.6 4.7 4.8 4.9 5 5.1 5.2 5.3 5.4 5.5 5.6 5.7 5.8 5.9 6 6.1 6.2

1 3 1 4 2 5 6 10 9 4 1 6 7 6 8 7 3 6 6 4

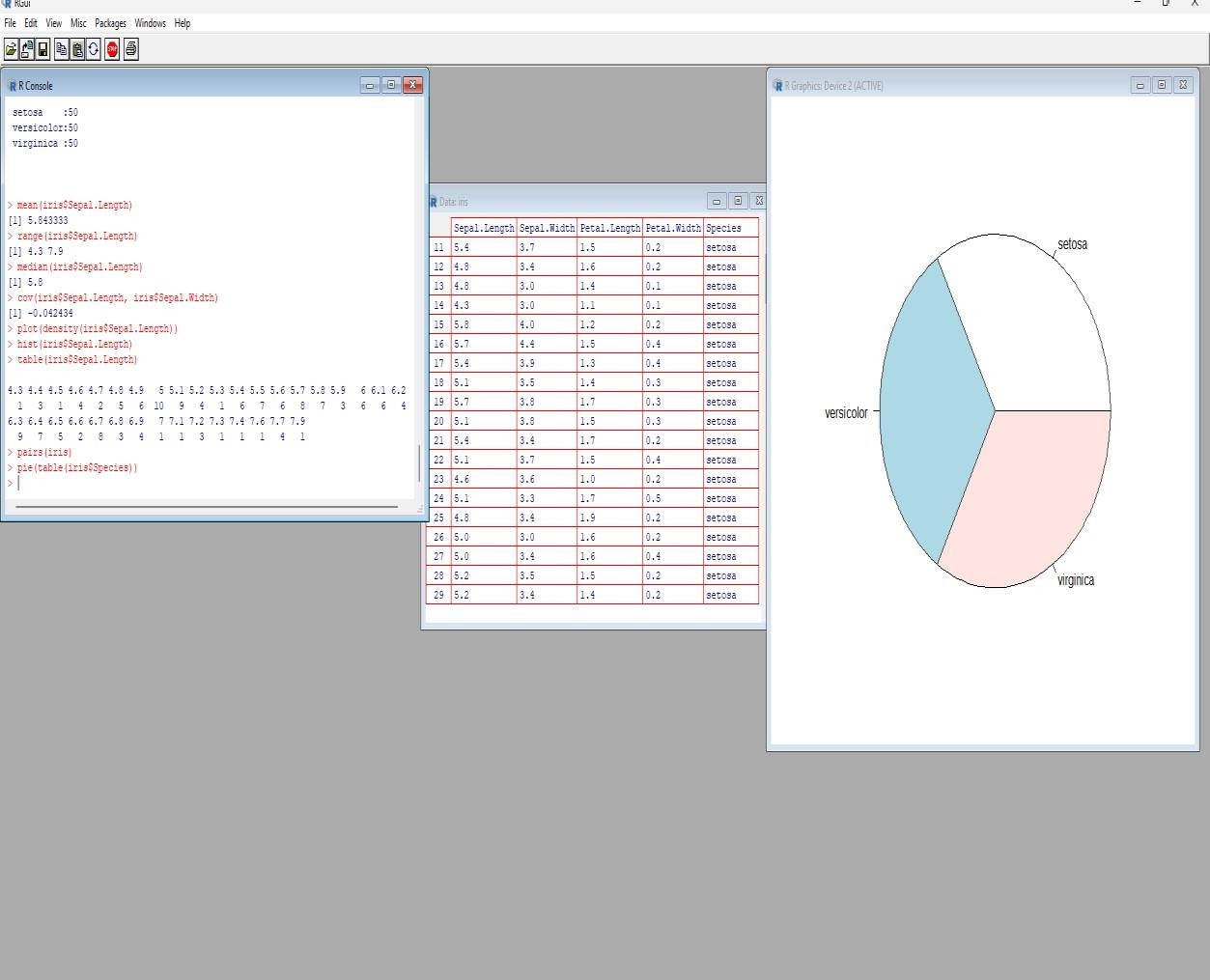
6.3 6.4 6.5 6.6 6.7 6.8 6.9 7 7.1 7.2 7.3 7.4 7.6 7.7 7.9

9 7 5 2 8 3 4 1 1 3 1 1 1 4 1

* pairs(iris)
* pie(table(iris$Species))

# OUTPUT

Flower.csv



# RESULT

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

|  |  |
| --- | --- |
| **Ex.No: 8**  **Date:** | **CLASSIFICATION OF SUPPORT VECTOR MACHINE(SVM)** |

# AIM

To perform classification of the Iris dataset using Support Vector Machine (SVM).

# ALGORITHM

**Step 1:** Start the program.

**Step 2:** Load required libraries: e1071 and caret.

**Step 3:** Load the Iris dataset and set the seed for reproducibility.

**Step 4:** Split the dataset into 70% training and 30% testing.

**Step 5:** Train the SVM model using a linear kernel.

**Step 6:** Predict the species for the test data.

**Step 7:** Generate the confusion matrix.

**Step 8:** Calculate Accuracy, Precision, Recall, and F1-Score.

**Step 9:** Display the results.

**Step 10:** Stop the program.

# PROGRAM

library(e1071) library(caret) data(iris) set.seed(42)

train\_index <- createDataPartition(iris$Species, p = 0.7, list = FALSE) train\_data <- iris[train\_index, ]

test\_data <- iris[-train\_index, ]

svm\_model <- svm(Species ~ ., data = train\_data, kernel = "linear") predictions <- predict(svm\_model, newdata = test\_data) conf\_matrix <- confusionMatrix(predictions, test\_data$Species) accuracy <- conf\_matrix$overall['Accuracy']

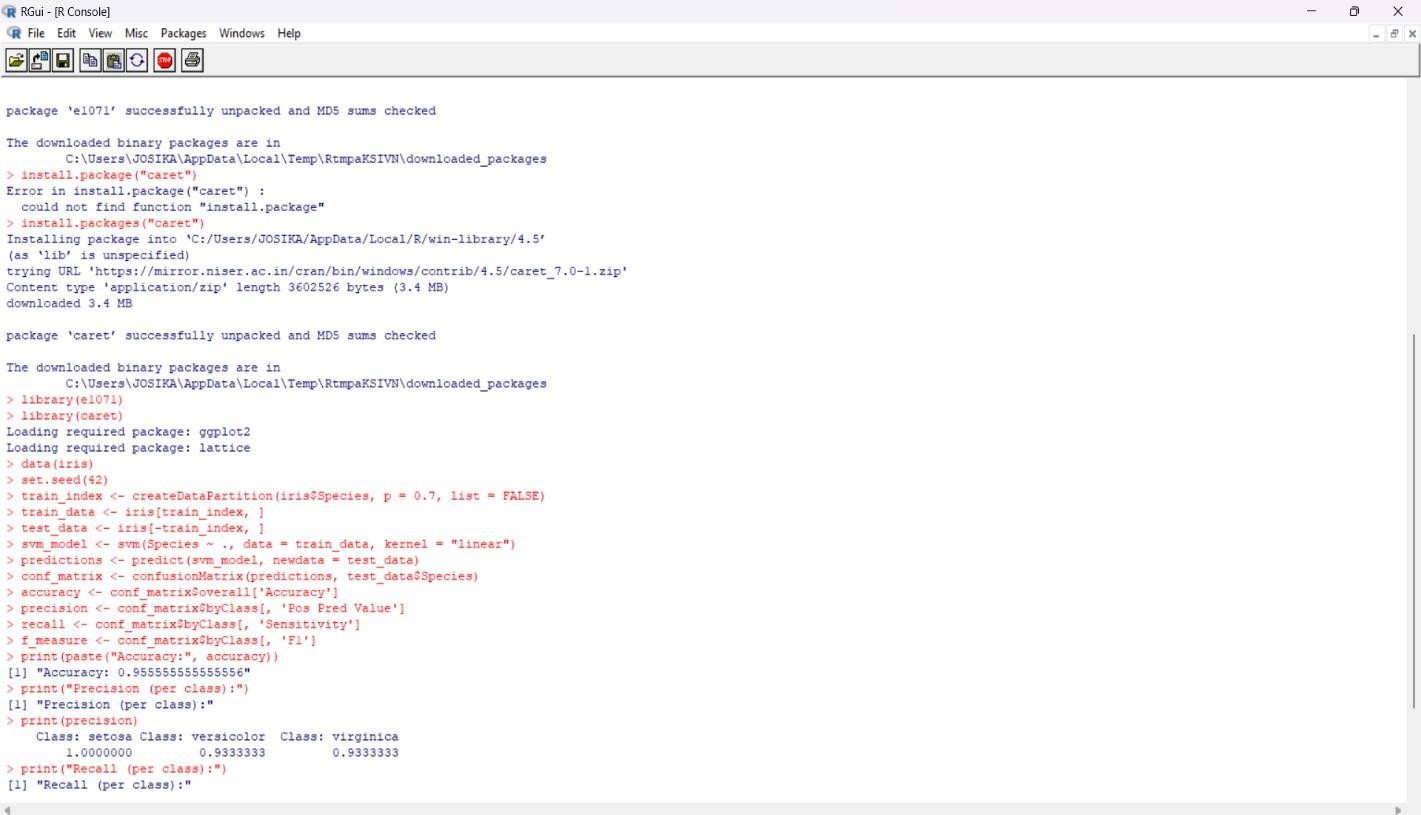
precision <- conf\_matrix$byClass[, 'Pos Pred Value'] recall <- conf\_matrix$byClass[, 'Sensitivity'] f\_measure <- conf\_matrix$byClass[, 'F1'] print(paste("Accuracy:", accuracy))

print("Precision (per class):") print(precision)

print("Recall (per class):") print(recall)

print("F1-Measure (per class):") print(f\_measure)

# OUTPUT

****

**RESULT:**

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

|  |  |
| --- | --- |
| **Ex.No: 9**  **Date:** | **LOGISITIC REGRESSION** |

# AIM

To build a logistic regression model to a dataset using r.

# ALGORITHM

**STEP 1:** Start the program.

**STEP 2:** Start R environment.

**STEP 3:** Install and load required packages (mlbench, ggplot2). **STEP 4:** Import the Breast Cancer dataset from the mlbench library. **STEP 5:** Remove missing or incomplete values using na.omit().

**STEP 6:** Convert the target variable Class into binary values (1 = malignant, 0 = benign). **STEP 7:** Convert predictor variables (e.g., Cl.thickness, Cell.size) into numeric format. **STEP 8:** Specify the logistic regression model using glm() with family = binomial.

**STEP 9:** Train (fit) the model using the dataset.

**STEP 10:** Generate predicted probabilities using the predict() function with type = "response".

**STEP 11:** Add predicted probabilities as a new column in the dataset.

**STEP 12:** Visualize results using ggplot2 with scatter plot and logistic curve.

**STEP 13:** Stop the program.

# PROGRAM

if (!require(mlbench)) install.packages("mlbench") if (!require(ggplot2)) install.packages("ggplot2") library(mlbench)

library(ggplot2)

data("BreastCancer", package = "mlbench") df <- BreastCancer

df <- na.omit(df)

df$Class <- ifelse(df$Class == "malignant", 1, 0) df$Cl.thickness <- as.numeric(as.character(df$Cl.thickness)) df$Cell.size <- as.numeric(as.character(df$Cell.size))

log\_model <- glm(Class ~ Cl.thickness + Cell.size, data = df, family = binomial) df$predicted\_probs <- predict(log\_model, type = "response")

ggplot(df, aes(x = Cl.thickness, y = predicted\_probs)) + geom\_point(aes(color = factor(Class)), size = 2, alpha = 0.7) +

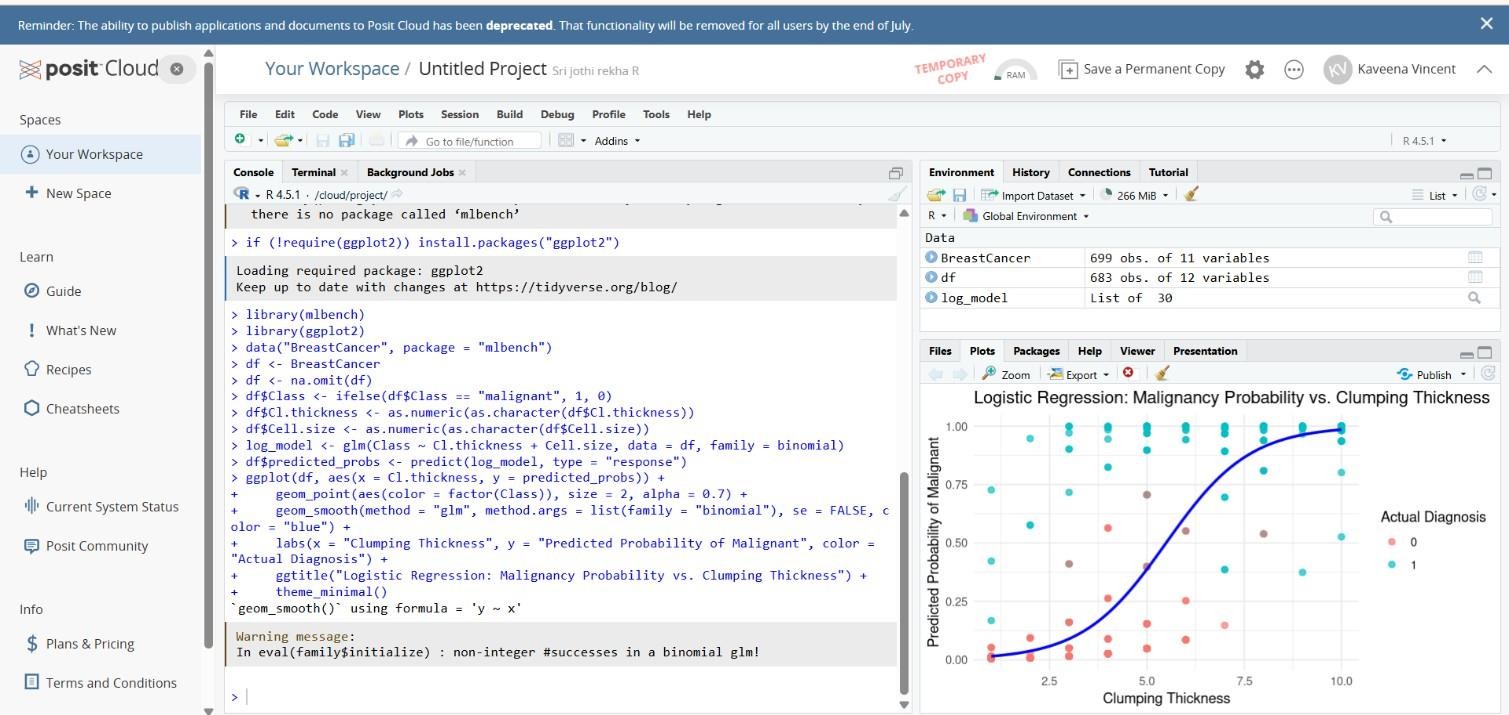
geom\_smooth(method = "glm", method.args = list(family = "binomial"), se = FALSE, color

= "blue") +

labs(x = "Clumping Thickness", y = "Predicted Probability of Malignant", color = "Actual Diagnosis") +

ggtitle("Logistic Regression: Malignancy Probability vs. Clumping Thickness") + theme\_minimal()

# OUTPUT

****

**RESULT**

Thus, the data exploration of the iris dataset is performed and various statistical operations are implemented.

|  |  |
| --- | --- |
| **Ex.No: 10**  **Date:** | **DBSCAN CLUSTERING** |

# AIM

To apply the DBSCAN clustering algorithm to a dataset using R

# ALGORITHM

**STEP 1**: Load the dataset and select only numeric features.

**STEP 2:** Standardize features (DBSCAN is distance-based; scaling matters). **STEP 3:** (Optional) Use a k-NN distance plot to estimate a good epsilon (eps). **STEP 4**: Choose minPts (typical rule of thumb: 2\*dim, e.g., 8–10 for 4 dims). **STEP 5:** Run DBSCAN with chosen eps and minPts.

**STEP 6:** Inspect cluster assignments (noise points have label 0 in dbscan::dbscan).

**STEP 7:** Project data to 2D (PCA) for visualization.

**STEP 8:** Plot clusters and review results; iterate on eps/minPts if needed.

# PROGRAM

install.packages(c("dbscan", "ggplot2", "factoextra")) library(dbscan)

library(ggplot2) library(factoextra) data(iris)

X <- scale(iris[, 1:4]) set.seed(1)

db <- dbscan(X, eps = 0.6, minPts = 5) cat("Cluster counts (0 = noise):\n") print(table(db$cluster))

pc <- prcomp(X)$x[, 1:2] df <- data.frame(

PC1 = pc[, 1],

PC2 = pc[, 2],

cluster = factor(ifelse(db$cluster == 0, "noise", db$cluster))

)

ggplot(df, aes(PC1, PC2, color = cluster)) + geom\_point(size = 2) +

labs(

title = "DBSCAN on iris (PCA view)", x = "PC1",

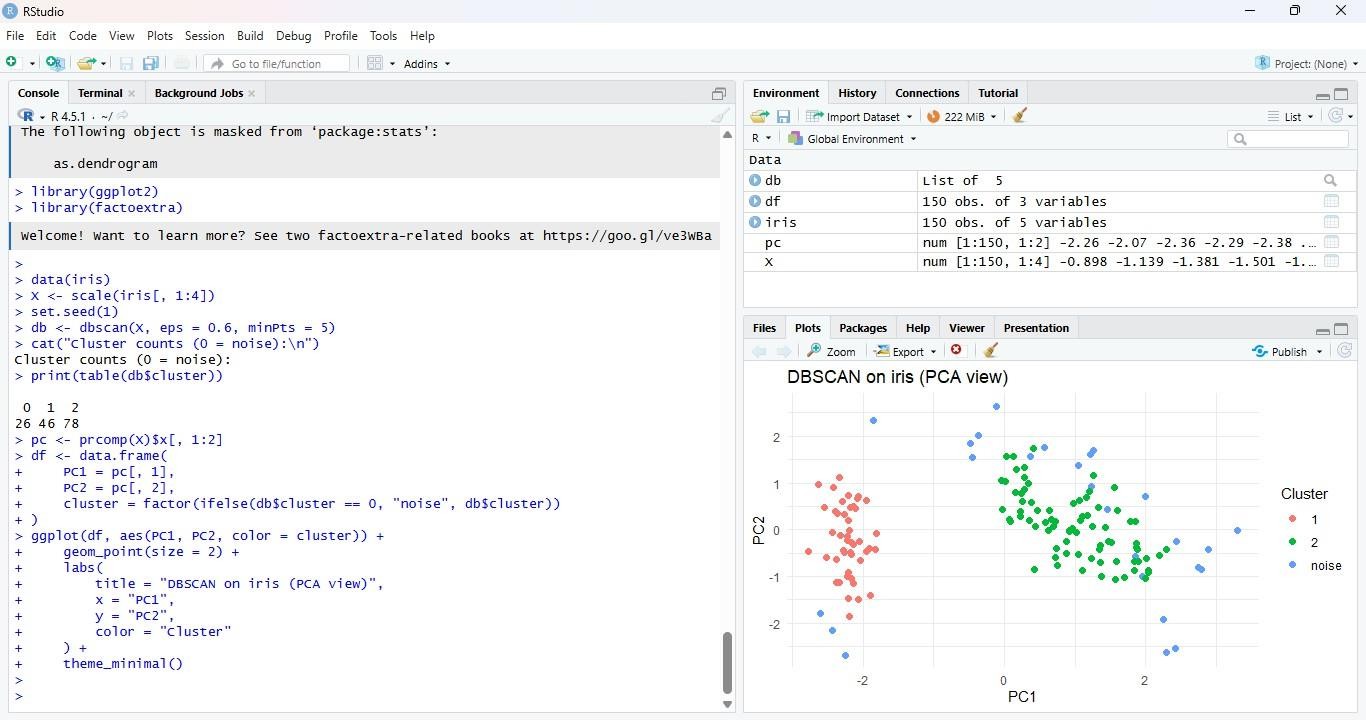
y = "PC2",

color = "Cluster"

) +

theme\_minimal()

# OUTPUT

****

**RESULT**

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

|  |  |
| --- | --- |
| **Ex.No: 11**  **Date:** | **DATA EXPLORATION AND VISUALIZATION** |

# AIM

To Perform Data Exploration and Visualization of the Stock Dataset and Implementation Various Statistical Operations in Tableau

# ALGORITHM

**STEP 1**: Download the dataset

**STEP 2**: Open tableau upload the dataset for use.

**STEP 3**: Open a sheet. (blank) drag the date into column

**STEP 4**: Sheet 2 Drag & drop volume into rows

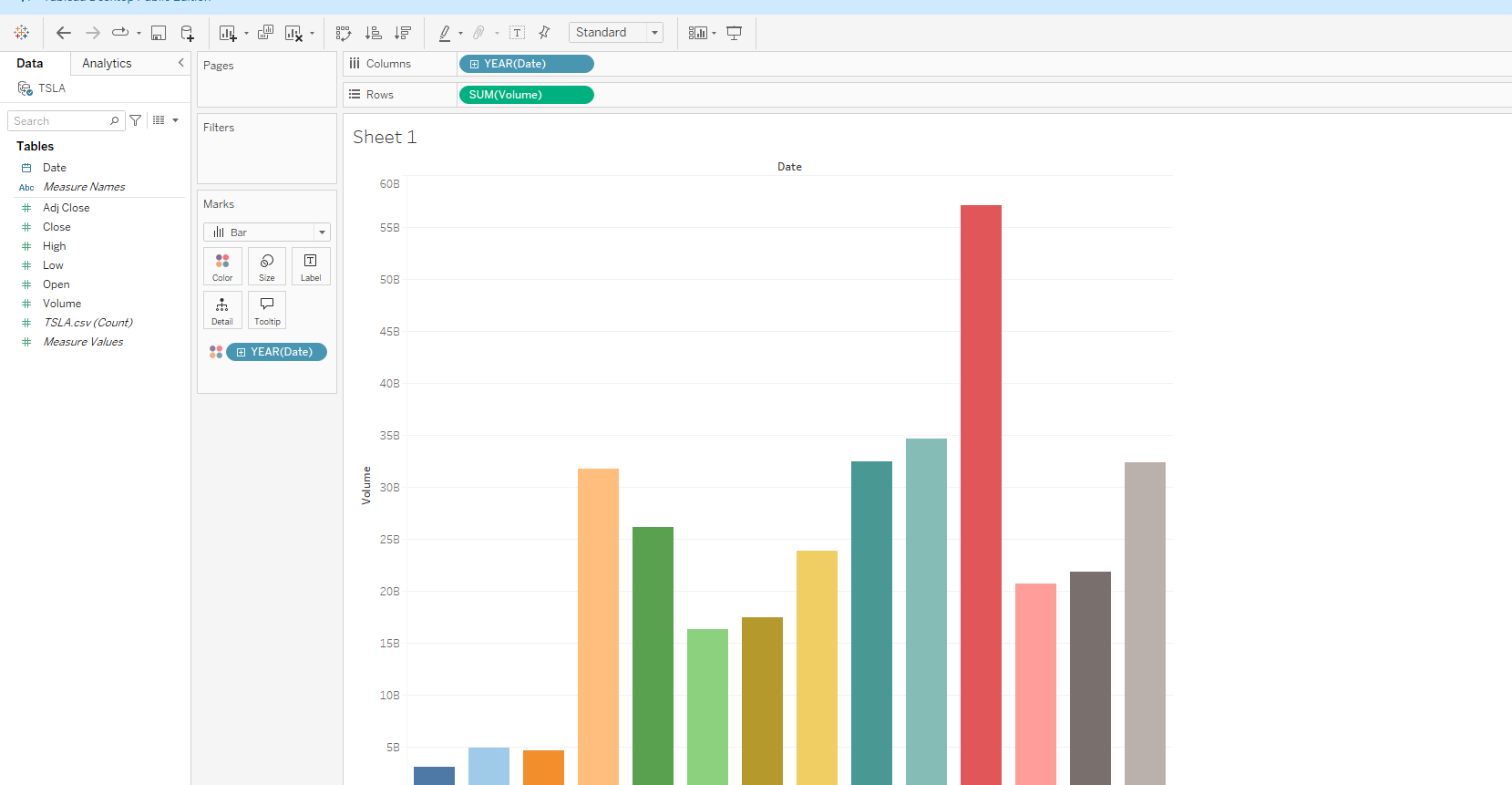
**STEP 5**: Statistical operation moving average in years.Right click on Close.

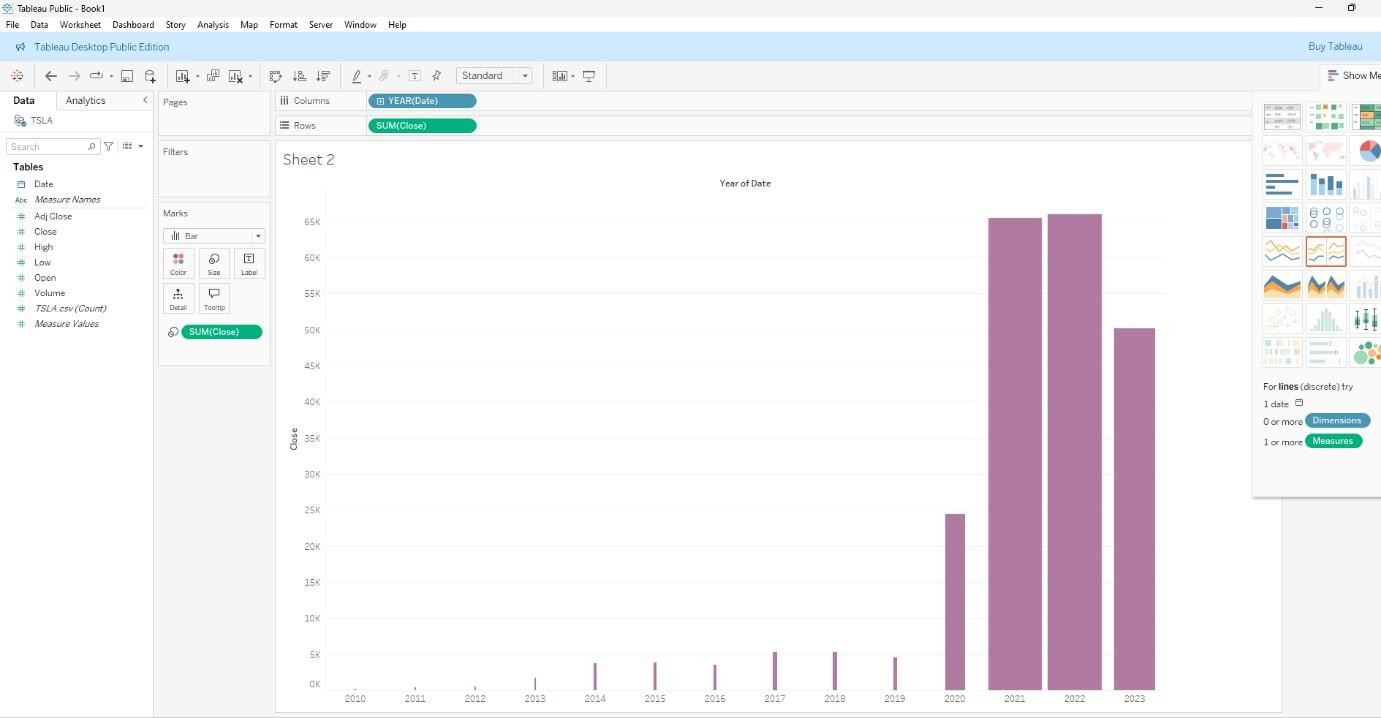
**STEP 6**: Combine into Dashboard.Go to New Dashboard. Drag Sheet 1 , Sheet 2 , and Sheet 3 into the dashboard.

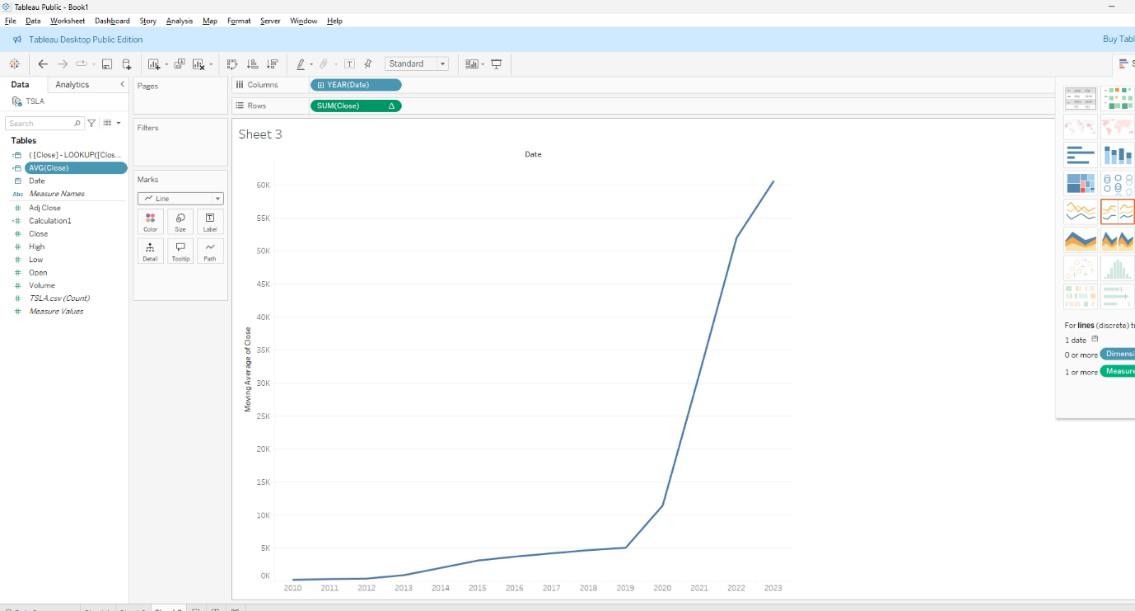
**STEP 7**:Arrange charts side by side or top–bottom.

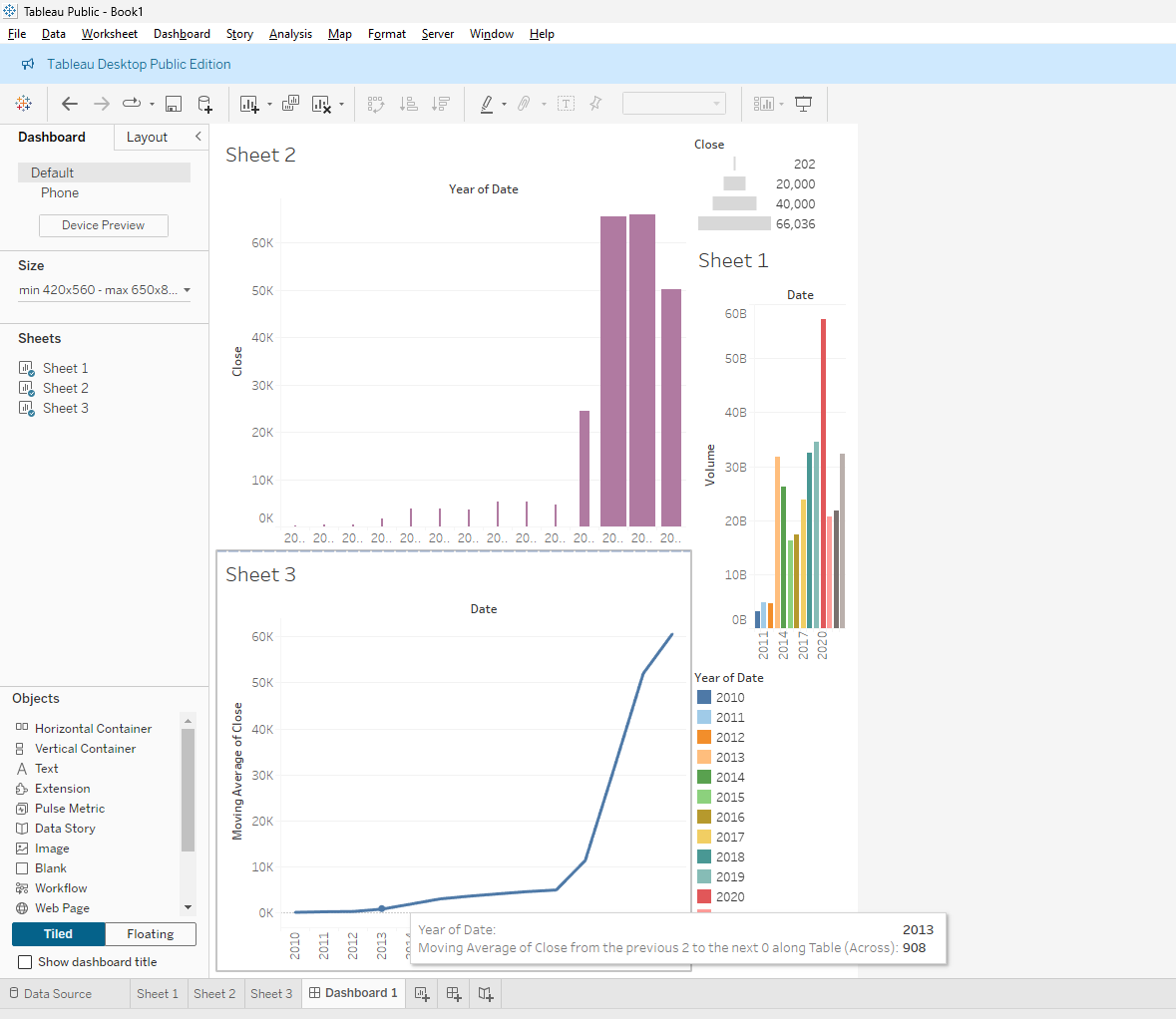
**STEP 8**:Save the workbook or publish to Tableau Public.

# OUTPUT

****

****



R

# RESULT

Thus the above program is executed successfully and the output is verified.

|  |  |
| --- | --- |
| **Ex.No: 12**  **Date:** | **DECISION TREE CLASSIFICATION USING KNIME** |

# AIM

To perform decision tree classification using Knime

# ALGORITHM

**STEP 1:** Start the process

**STEP 2:** Load the Dataset and read the Iris dataset from a CSV file.

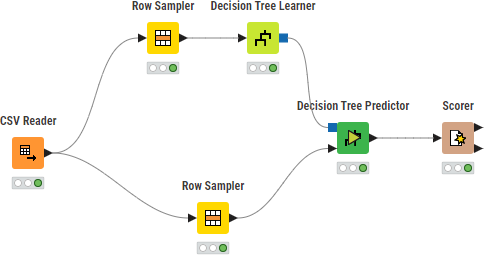
**STEP 3:** Split the Dataset into Training Set (70%) and Testing Set (30%). **STEP 4:** Model should use the training set to train a Decision Tree Classifier. **STEP 5:** Use the trained model to predict labels for the test set.

**STEP 6:** Compute the confusion matrix and accuracy score.

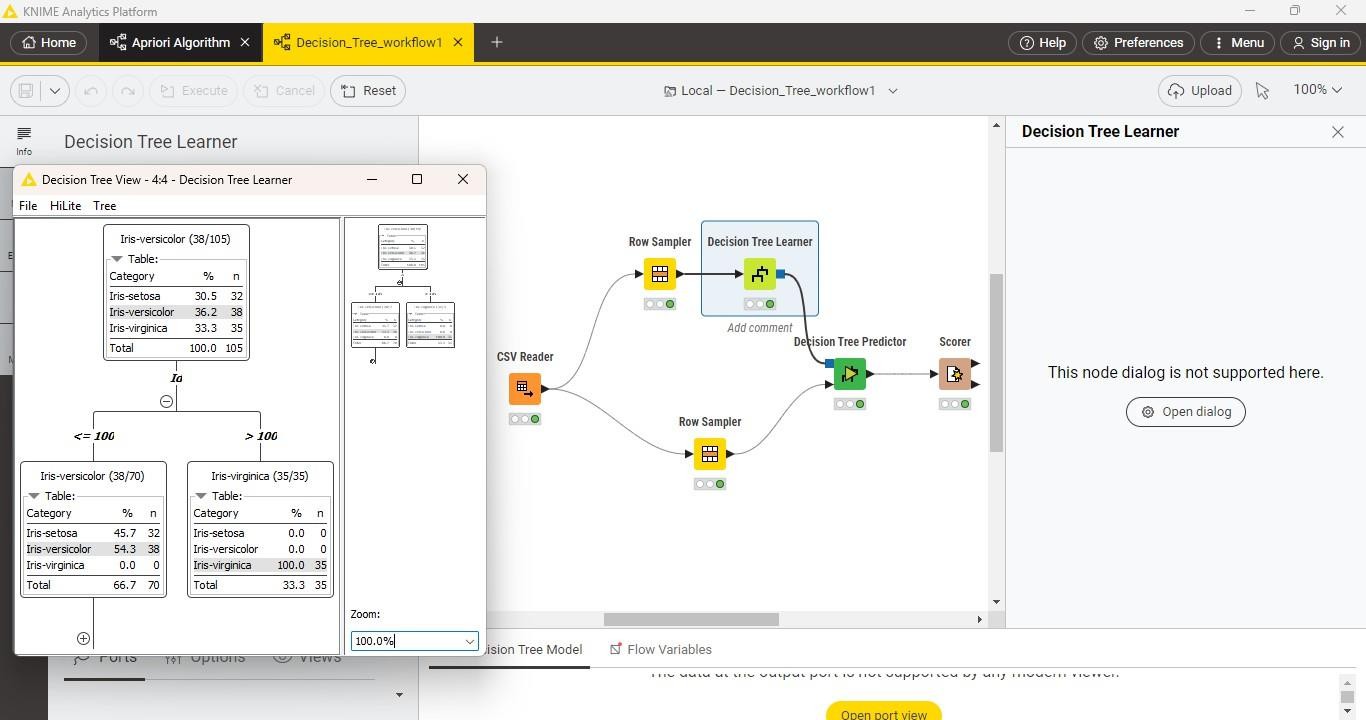
**STEP 7:** Visualize the decision tree diagram

**STEP 8:** Stop the process

**PROGRAM STRUCTURE**

****

**OUTPUT**

****

**RESULT**

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

|  |  |
| --- | --- |
| **Ex.No: 13**  **Date:** | **APRIORI ALGORITHM** |

# AIM

To perform Apriori Algorithm (Frequent Itemset Mining) using Knime.

# ALGORITHM

**STEP 1:** Start KNIME Analytics Platform,

**STEP2:** Load Dataset (ex: transaction dataset)

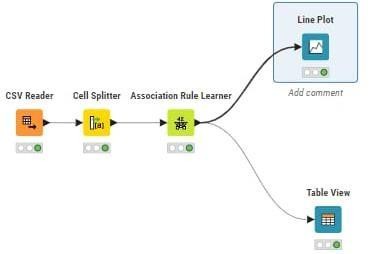
**STEP3:** Preprocess the dataset if it is not already in binary format (0/1).

**STEP 4:** Using the knime application, drag and drop the Association Rule Learner node.

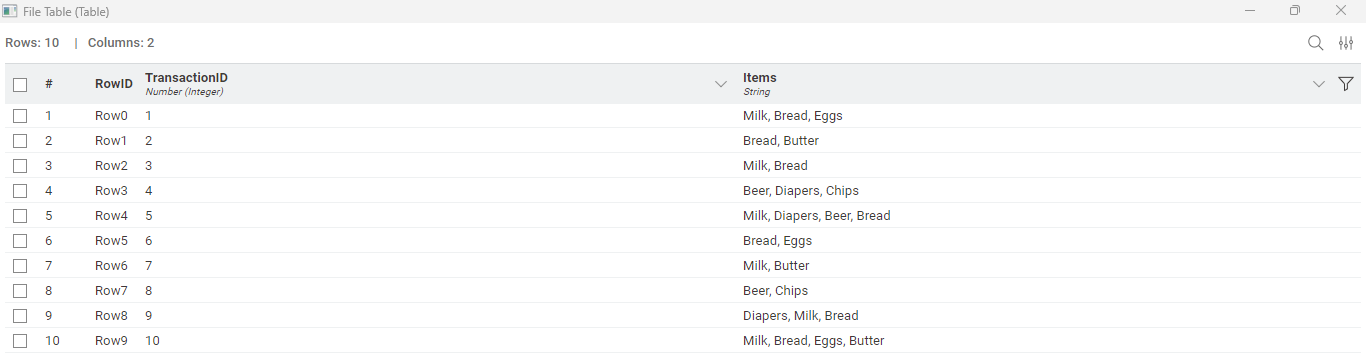
**STEP 5**: Execute Apriori Algorithm

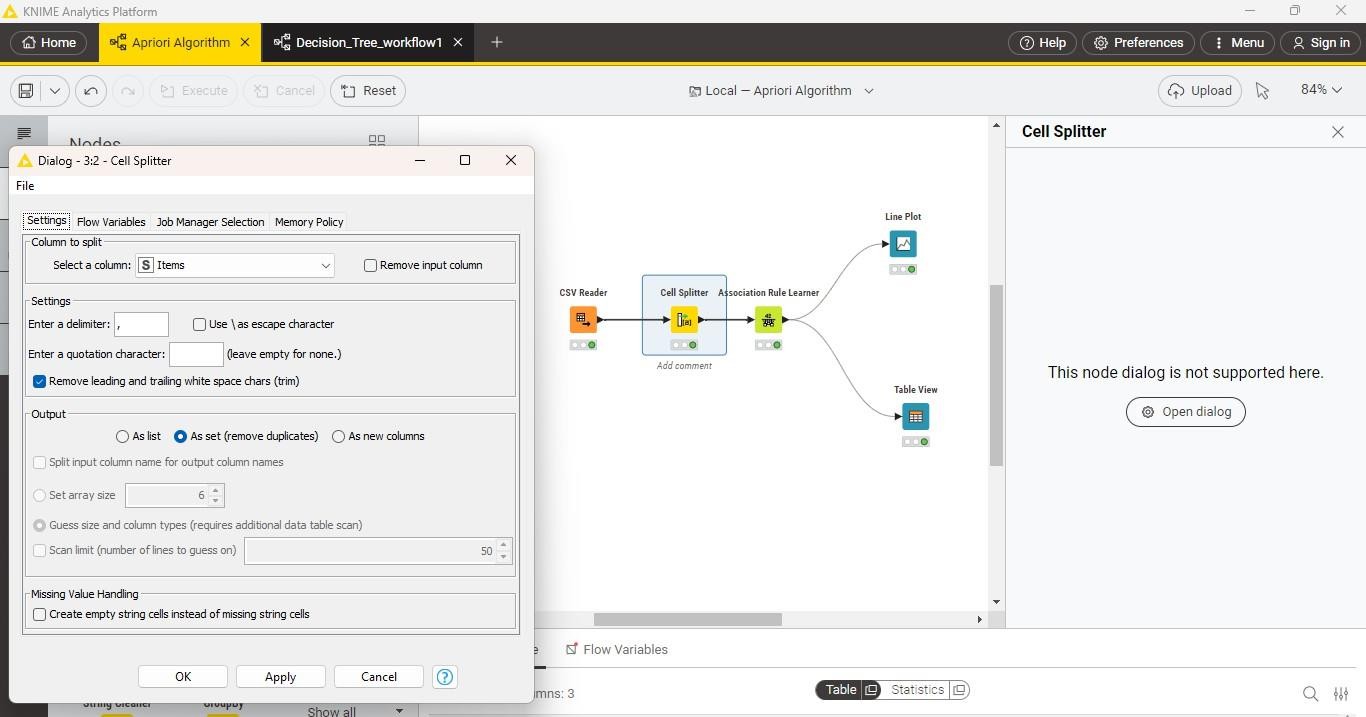
**STEP 6:** Connect the Association Rule Learner to the Association Rule Viewer node. **STEP 7:** Analyse Results by using the Rule Viewer to filter and interpret patterns **STEP 8:** End the program.

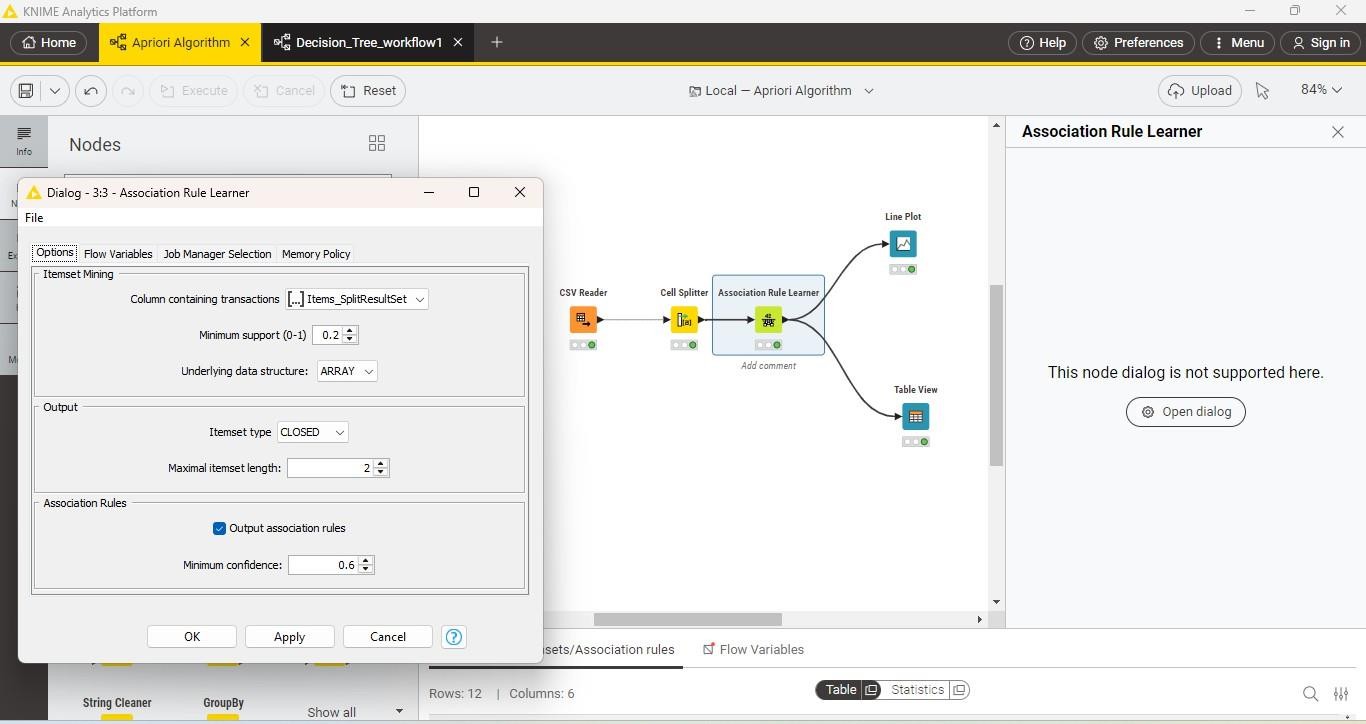
**PROGRAM STRUCTURE**

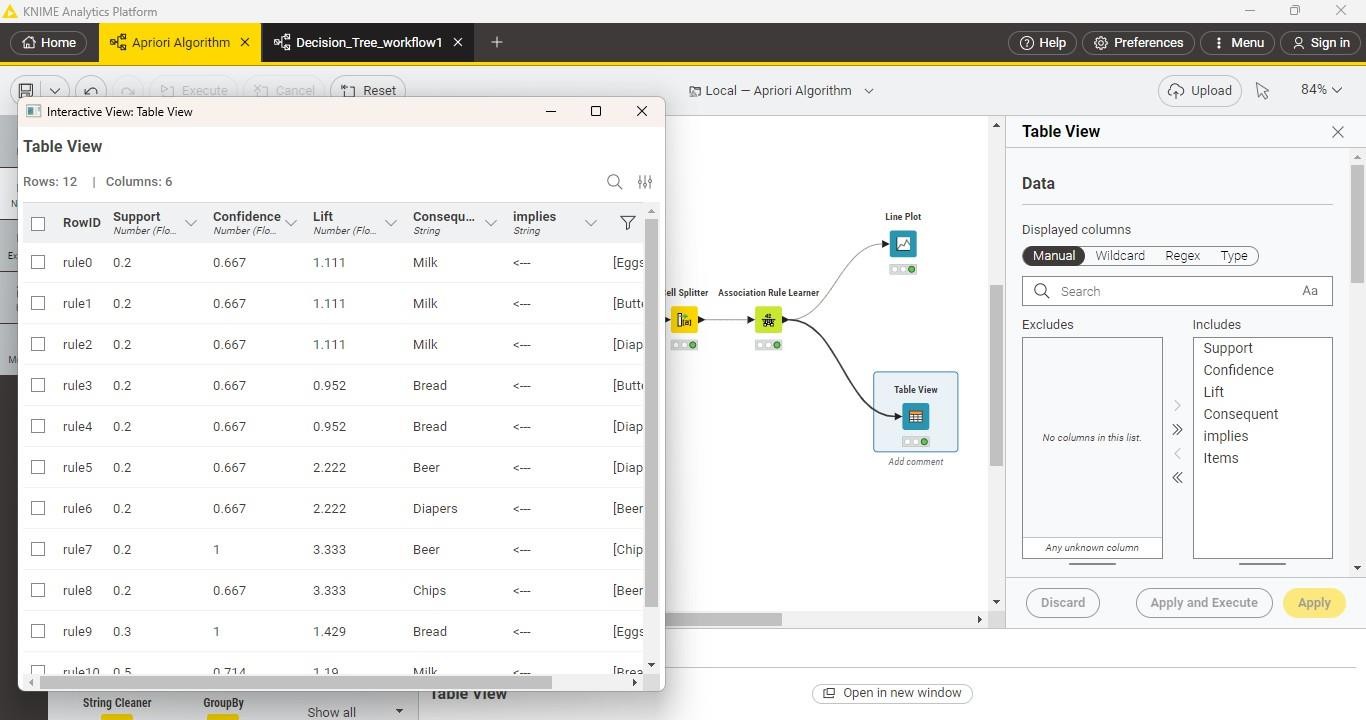
****

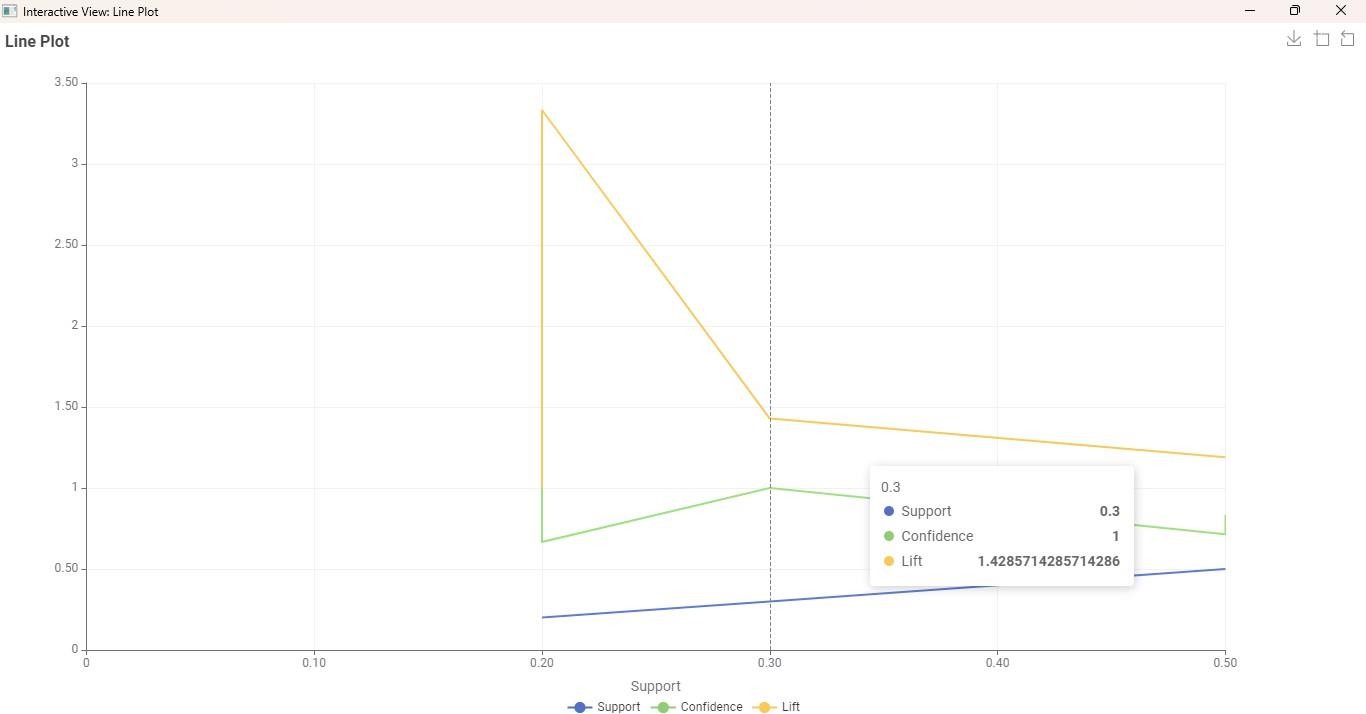
**OUTPUT**

****

****



****



**RESULT**

Thus, The program is executed successfully, and the output is verified

|  |  |
| --- | --- |
| **EX NO:01**  **DATE:** | **CUSTOMER BUYING PATTERN USING CLASSIFICATION**  **METHODS** |

# AIM

To write a python code for customer buying pattern using classification methods.

# ALGORITHM

**STEP 1:** Start the program.

**STEP 2:** Import the python libraries such as pandas,sklearn.model\_selection,sklearn.tree and sklearn.metrics.

**STEP 3:** Load or create the dataset containing features: Age,Annual Income (Target Variable).

**STEP 4:** Separate the dataset into:Features (X),Target (y).Split the dataset into training and testing sets using train\_test\_split.

**STEP 5:** Store the dataset into a pandas DataFrame for easier handling. Create the target variable Purchased with values Yes/No.

**STEP 6:** Create a DecisionTreeClassifier model object.

**STEP 7:** Fit the model using the training data (X\_train, y\_train). Use the trained model to predict values on the testing set (X\_test).Evaluate the Model.Compare predicted values (y\_pred) with actual values (y\_test).

**STEP 8**: Print the accuracy.Print the classification report.

**STEP 9:** Exection the program.

**STEP 10:** End.

# PROGRAM

import pandas as pd

from sklearn.model\_selection import train\_test\_split from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import classification\_report, accuracy\_score # Sample customer data

data = {

'Age': [25, 45, 23, 36, 52, 40, 28, 48],

'AnnualIncome': [25, 70, 30, 60, 80, 50, 40, 65],

'Purchased': ['No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes'] # Target variable

}

df = pd.DataFrame(data)

# Features (X) and Target (y)

X = df[['Age', 'AnnualIncome']] y = df['Purchased']

# Split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Create Decision Tree model model = DecisionTreeClassifier() model.fit(X\_train, y\_train)

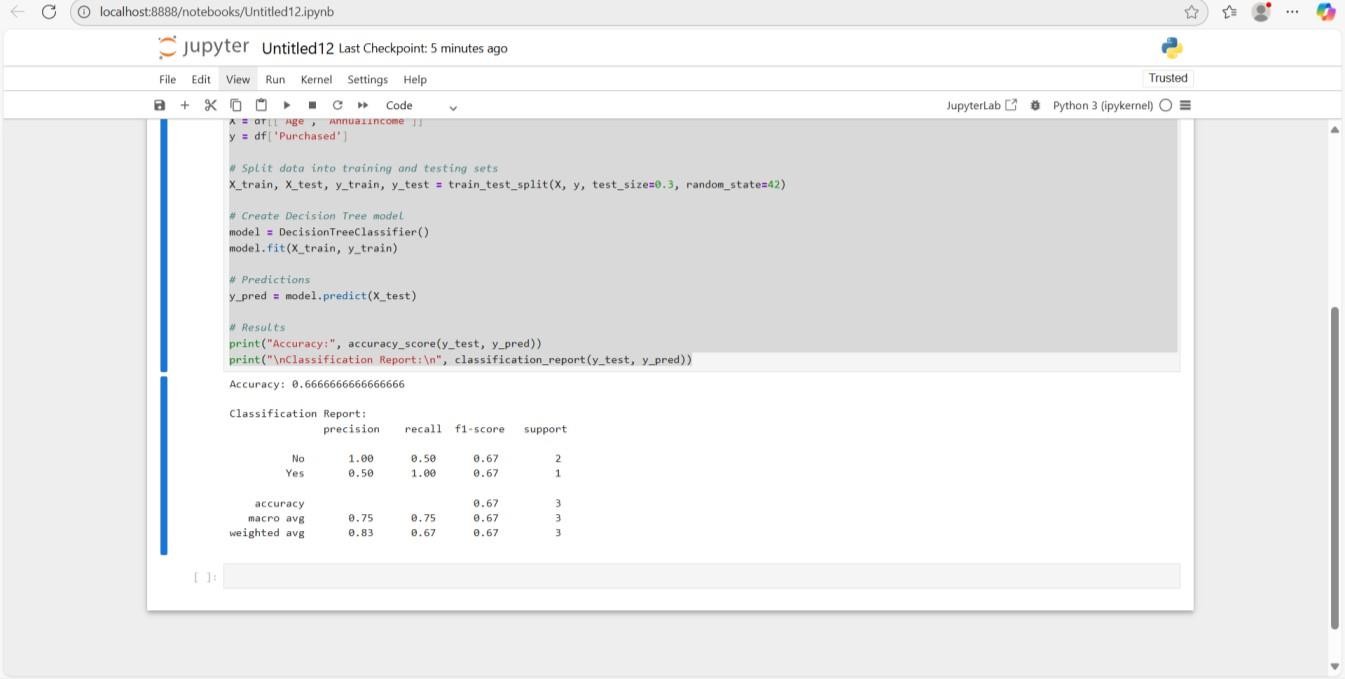
# Predictions

y\_pred = model.predict(X\_test)

# Results

print("Accuracy:", accuracy\_score(y\_test, y\_pred)) print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

# OUTPUT

****

**RESULT**

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

|  |  |
| --- | --- |
| **EX NO:02**  **DATE:** | **CREDIT CARD FRAUD SETECTION USING SUPERVISED**  **ALGORITHM** |

# AIM

The aim of this project is to build a simple machine learning model using logistic regression that can predict whether a financial transaction is fraudulent or not .

# ALGORITHM

**STEP 1**: Import libraries for data processing, splitting, scaling, and modeling.

**STEP 2**: Create a small dataset with transaction details and fraud labels.

**STEP 3**: Separate features and target variable.

**STEP 4**: Split the data into stratified training and testing sets.

**STEP 5**: Scale the feature values using standard scaling.

**STEP 6**: Train a logistic regression model on the scaled training data.

**STEP 7**: Get new transaction input from the user.

# PROGRAM

import pandas as pd

from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn.linear\_model import LogisticRegression data = pd.DataFrame({

'Amount': [100, 250, 300, 50, 500, 450, 120, 80, 700, 60],

'TransactionTime': [5, 4, 8, 2, 20, 18, 7, 3, 22, 1],

'Age': [25, 45, 30, 22, 50, 40, 28, 24, 55, 20],

'Class': [0, 0, 1, 0, 1, 1, 0, 0, 1, 0] # 0 = No Fraud, 1 = Fraud })

X = data.drop('Class', axis=1) y = data['Class']

X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, y, test\_size=0.3, random\_state=42, stratify=y

)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train) X\_test = scaler.transform(X\_test)

model = LogisticRegression(max\_iter=1000) model.fit(X\_train, y\_train)

amount = float(input("Enter Transaction Amount: ")) time = float(input("Enter Transaction Time (0–24 hrs): ")) age = int(input("Enter Customer Age: "))

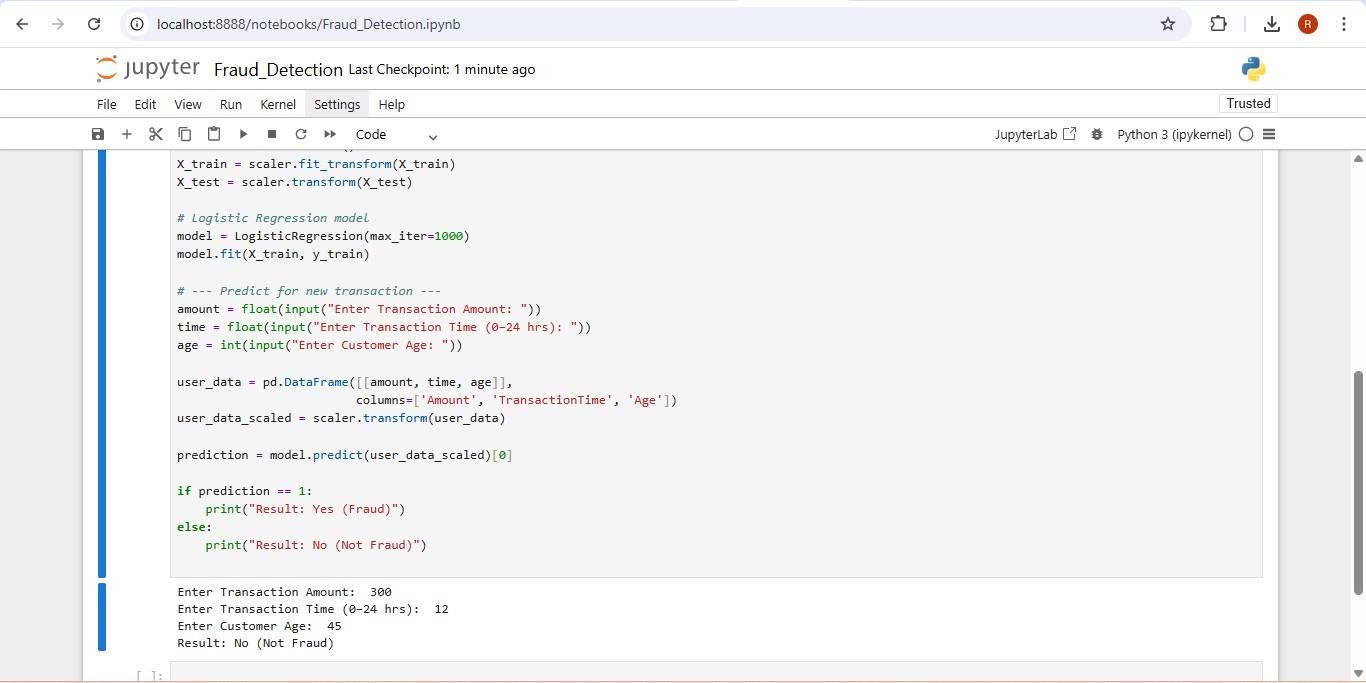
user\_data = pd.DataFrame([[amount, time, age]], columns=['Amount', 'TransactionTime', 'Age'])

user\_data\_scaled = scaler.transform(user\_data) prediction = model.predict(user\_data\_scaled)[0] if prediction == 1:

print("Result: Yes (Fraud)") else:

print("Result: No (Not Fraud)"

# OUTPUT

****

**RESULT**

Thus the program is executed successfully and output is verified.

|  |  |
| --- | --- |
| **EX NO:03**  **DATE:** | **BREAST CANCER PREDICTION USING SUPERVISED**  **ALGORITHM** |

# AIM

To develop a breast cancer prediction model using a supervised machine learning algorithm (Random Forest Classifier) to classify tumors as benign or malignant based on diagnostic features.

# ALGORITHM

**STEP 1:** Import numpy for numerical operations and relevant modules from sklearn for dataset handling, model building, and evaluation.

**STEP 2:** Load the Breast Cancer dataset using load\_breast\_cancer().

**STEP 3:** Separate the dataset into features X and target labels y.

**STEP 4:** Divide the data into training (80%) and testing (20%) sets using train\_test\_split() with a fixed random state for reproducibility.

**STEP 5:** Create a RandomForestClassifier instance with suitable parameters (e.g., random\_state=42).

**STEP 6:** Fit the classifier on the training set (X\_train, y\_train).

**STEP 7:** Predict the tumor class for the test set (X\_test) and Optionally, predict the class for a new, unseen sample.

**STEP 8:** Compute and display the Accuracy score of the test predictions and Classification report including precision, recall, and F1-score.

# PROGRAM

from sklearn.datasets import load\_breast\_cancer from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report import numpy as np

data = load\_breast\_cancer()

X = data.data y = data.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) model = RandomForestClassifier(random\_state=42)

model.fit(X\_train, y\_train) y\_pred = model.predict(X\_test)

print("Test Accuracy:", accuracy\_score(y\_test, y\_pred))

new\_data = np.array([[17.99, 10.38, 122.8, 1001.0, 0.1184, 0.2776, 0.3001, 0.1471, 0.2419,

0.07871, 1.095, 0.9053, 8.589, 153.4, 0.006399, 0.04904, 0.05373, 0.01587, 0.03003,

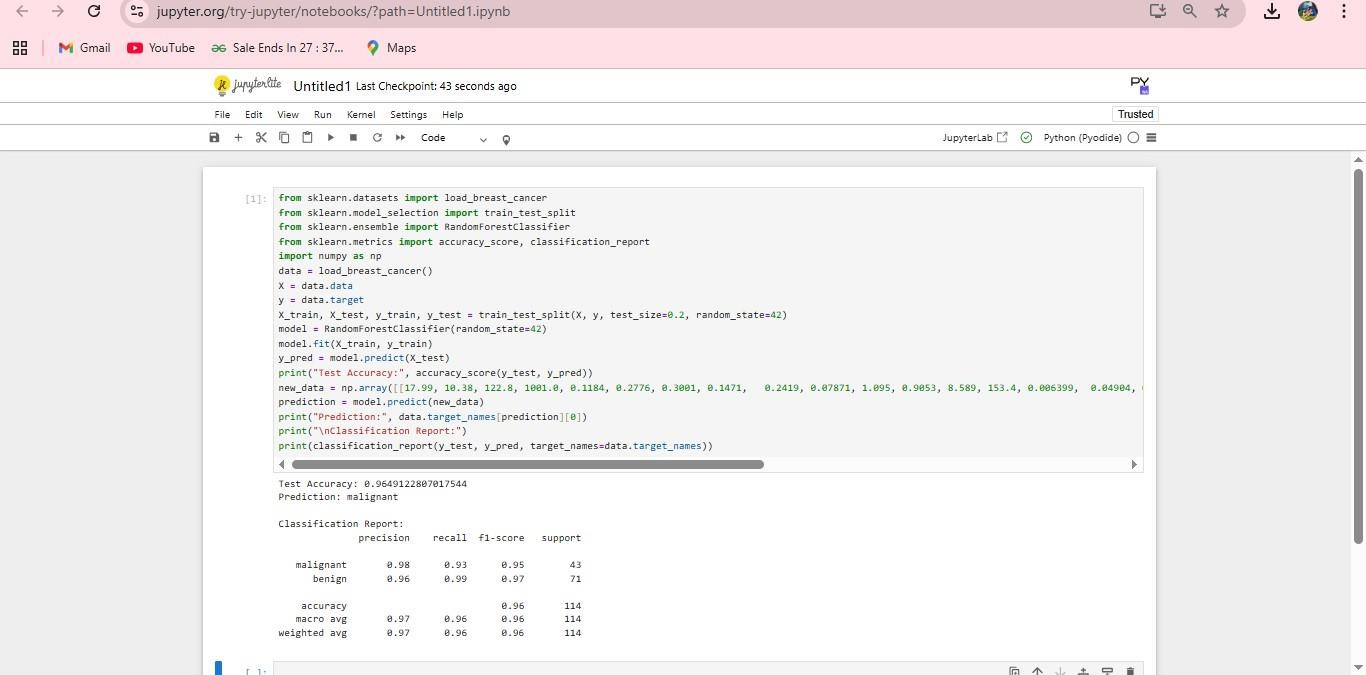
0.006193, 25.38, 17.33, 184.6, 2019.0, 0.1622, 0.6656, 0.7119, 0.2654, 0.4601, 0.1189]])

prediction = model.predict(new\_data) print("Prediction:", data.target\_names[prediction][0]) print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred, target\_names=data.target\_names))

# OUTPUT

Dataset:breast\_cancer dataset



# RESULT

Thus the program is executed successfully and output is verified.

|  |  |
| --- | --- |
| **EX NO:04**  **DATE:** | **CUSTOMER SEGMENTATION USING**  **CLUSTERING** |

# AIM

To write a Python program to segment customers using K-Means clustering based on their Annual Income and Spending Score.

# ALGORITHM

**STEP 1**: Import necessary libraries like pandas for data handling, sklearn for clustering, and matplotlib for visualization.

**STEP 2**: Create the customer data manually using a dictionary and convert it into a pandas DataFrame.

**STEP 3:** Select relevant features (Annual Income, Spending Score) for clustering.

**STEP 4**: Apply K-Means clustering algorithm with a specified number of clusters (n\_clusters=3).

**STEP 5**: Add the predicted cluster labels to the original DataFrame as a new column. **STEP 6**: Visualize the clusters using a scatter plot with different colors for each cluster. **STEP 7**: Label the axes and give a title to the plot.

**STEP 8**: Display the final plot using plt.show().

# PROGRAM

import pandas as pd

from sklearn.cluster import KMeans import matplotlib.pyplot as plt

# Sample data data = {

'CustomerID': [1, 2, 3, 4, 5, 6],

'Age': [25, 45, 23, 36, 52, 40],

'AnnualIncome(k$)': [25, 70, 30, 60, 80, 50],

'SpendingScore(1-100)': [70, 30, 65, 40, 20, 50]

}

# Create DataFrame

df = pd.DataFrame(data)

# Select features for clustering

X = df[['AnnualIncome(k$)', 'SpendingScore(1-100)']]

# Apply KMeans clustering

kmeans = KMeans(n\_clusters=3, random\_state=0) df['Cluster'] = kmeans.fit\_predict(X) # Assign clusters

# Show results print(df)

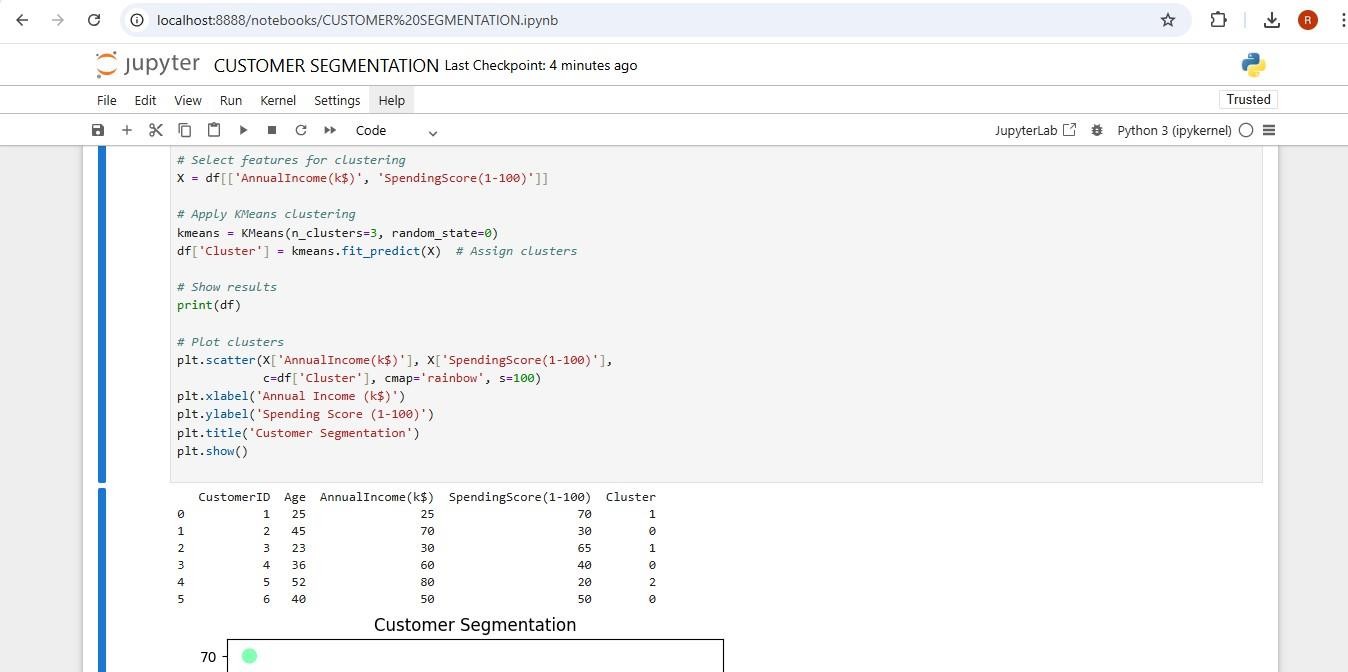
# Plot clusters

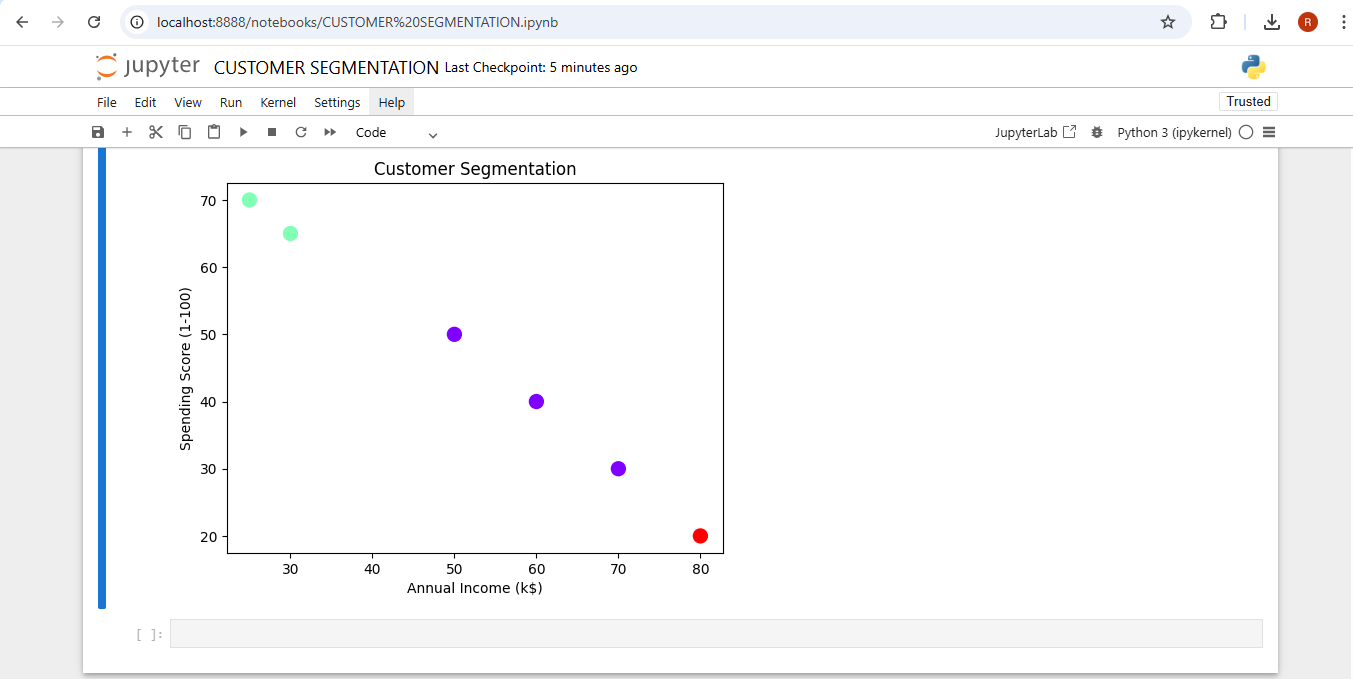
plt.scatter(X['AnnualIncome(k$)'], X['SpendingScore(1-100)'], c=df['Cluster'], cmap='rainbow', s=100)

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)') plt.title('Customer Segmentation') plt.show()

# OUTPUT

****

****

**RESULT**

Thus the program is executed successfully and output is verified.

|  |  |
| --- | --- |
| **EX NO:05**  **DATE:** | **LOAN DEFAULTERS PREDICTION** |

# AIM

To write a python program for loan defaulters prediction using decision tree.

# ALGORITHM

**STEP 1:** Import necessary Python libraries such as pandas, sklearn.model\_selection,sklearn.tree, and sklearn.metrics.

**STEP 2:**Load or create the dataset containing features:Age,Annual Income,Loan Amount,Credit Score,Default Status (Target Variable)

**STEP 3:** Separate the dataset into:Features (X),Target (y).Split the dataset into training and testing sets using train\_test\_split.

**STEP 4:** Initialize the DecisionTreeClassifier.Train the model using the training data (X\_train, y\_train).

**STEP 5:** Use the trained model to predict outcomes on the test set (X\_test).Also make predictions for new applicants.

**STEP 6:**Calculate accuracy using accuracy\_score.Generate a classification report using classification\_report.

**STEP 7:**Accept or define input data for a new loan applicant.

**STEP 8:**Format the input with feature names.

**STEP 9:**Predict whether the applicant is likely to default or not default**.**

# PROGRAM

#import the required libraries import pandas as pd

from sklearn.model\_selection import train\_test\_split from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, classification\_report #Sample loan data

data={'Age':[25,45,35,23,52,40,28,48], 'AnnualIncome':[25000,70000,35000,80000,60000,50000,75000,20000], 'LoanAmount':[5000,20000,8000,3000,25000,10000,7000,18000], 'CreditScore':[650,720,600,580,750,690,640,710],

'Defaulted':['No','No','Yes','Yes','No','Yes','Yes','No']

}

#create dataframe df=pd.DataFrame(data) #features and target

X=df[['Age','AnnualIncome','LoanAmount','CreditScore']] y=df['Defaulted']

#train-test split X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.3,random\_state=42) #mode;

model=DecisionTreeClassifier() model.fit(X\_train,y\_train) #predictions y\_pred=model.predict(X\_test) #Results

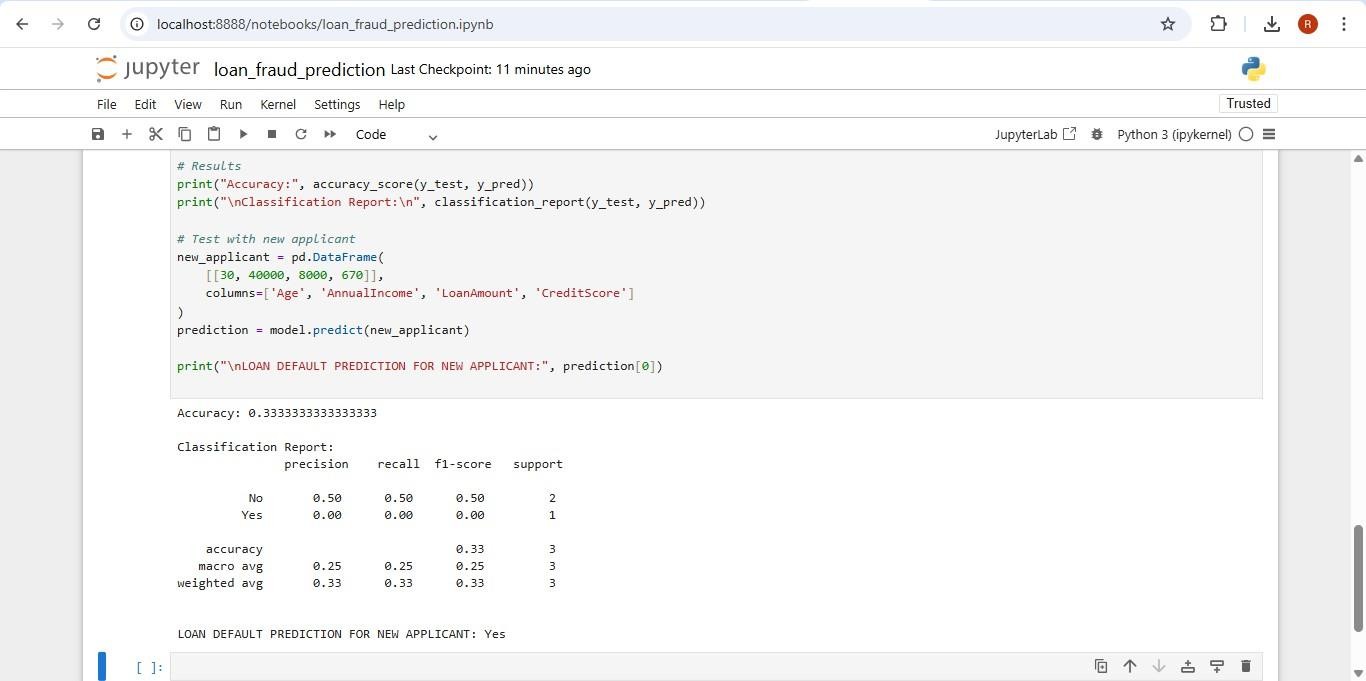
print("Accuracy:",accuracy\_score(y\_test,y\_pred)) print("\nClassification Report:\n",classification\_report(y\_test,y\_pred)) #test with new applicant

new\_applicant = pd.DataFrame([[30, 40000, 8000, 670]], columns=['Age', 'AnnualIncome', 'LoanAmount', 'CreditScore'])

prediction = model.predict(new\_applicant) prediction=model.predict(new\_applicant)

print("\nLOAN DEFAULT PREDICTION FOR NEW APPLICSNT:",prediction[0])

# OUTPUT

****

**RESULT**

Thus, the above python program to perform decision tree on loan defaulters has been verified and executed successfully.

|  |  |
| --- | --- |
| **EX NO:06**  **DATE:** | **CLASSIFICATION OF IRIS DATASET** |

# AIM

To classify the Iris Dataset.

# ALGORITHM

**STEP 1:** Import necessary libraries from scikit-learn for loading data, splitting, modeling, and evaluation.

**STEP 2:** Load the Iris dataset and assign features to X and target labels to y.

**STEP 3:** Split the data into 80% training and 20% testing using train\_test\_split() with a fixed random state.

**STEP 4:** Create a RandomForestClassifier and train it on the training data using .fit().

**STEP 5:** Predict the species for test data using the trained model’s .predict() method.

**STEP 6:** Calculate model accuracy using accuracy\_score() by comparing predicted and actual labels.

**STEP 7:** Display the confusion matrix using confusion\_matrix() to check classification results.

**STEP 8:** Show the classification report using classification\_report() for precision, recall, and F1-score.

# PROGRAM

# Import libraries

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier # Corrected import

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

# Load the Iris dataset iris = load\_iris()

X = iris.data # Features: sepal length, sepal width, petal length, petal width y = iris.target # Target: species (0, 1, 2)

# Split data into training (80%) and testing (20%) sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, y, test\_size=0.2, random\_state=42)

# Create and train the Random Forest model

model = RandomForestClassifier(random\_state=42) model.fit(X\_train, y\_train)

# Make predictions on the test set y\_pred = model.predict(X\_test)

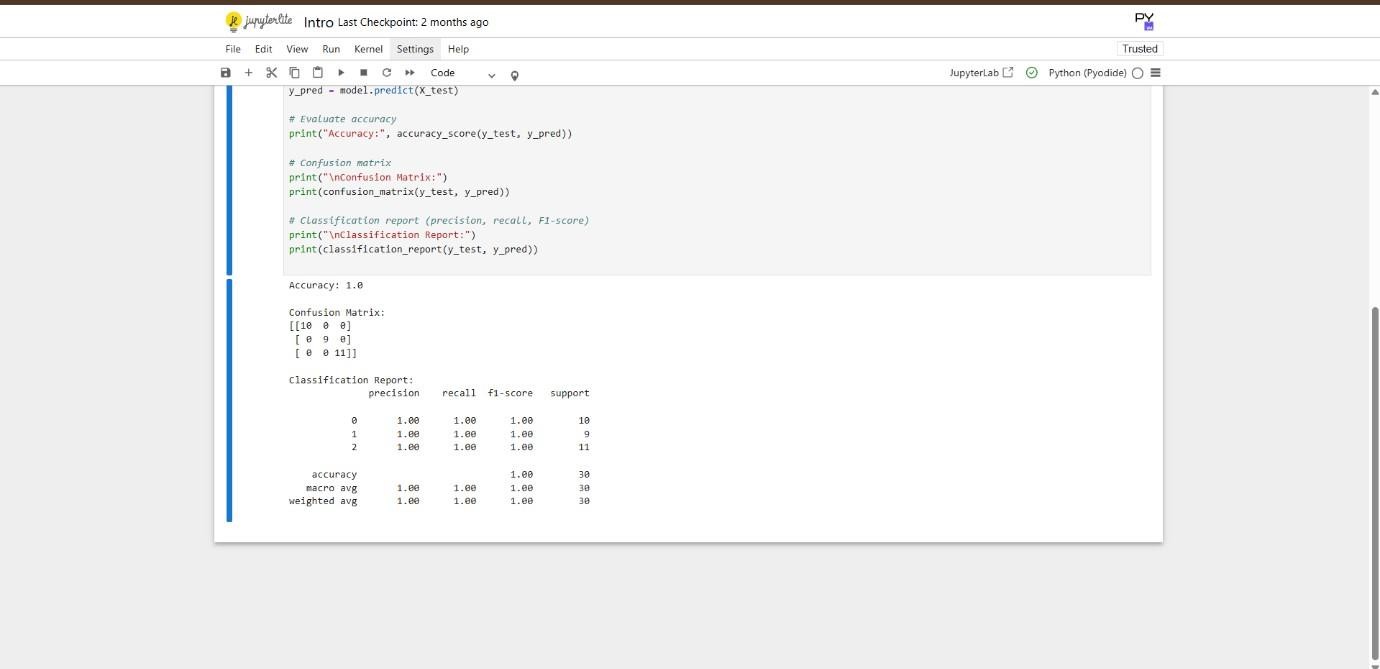
# Evaluate accuracy

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

# Confusion matrix, Classification report (precision, recall, F1-score) print("\nConfusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred)) print("\nClassification Report:") print(classification\_report(y\_test, y\_pred))

# OUTPUT

****

**RESULT**

Thus, the above program is executed successfully and the output is verified.