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

AIM

To Demonstrate the following data preprocessing tasks using Python libraries.

a) Loading the dataset

b) Identifying the dependent and independent variables

c) 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 R^2 score and Mean Squared Error (MSE).

STEP 12: Plot the actual vs predicted values.

STEP 13: End the program.

PROGRAM

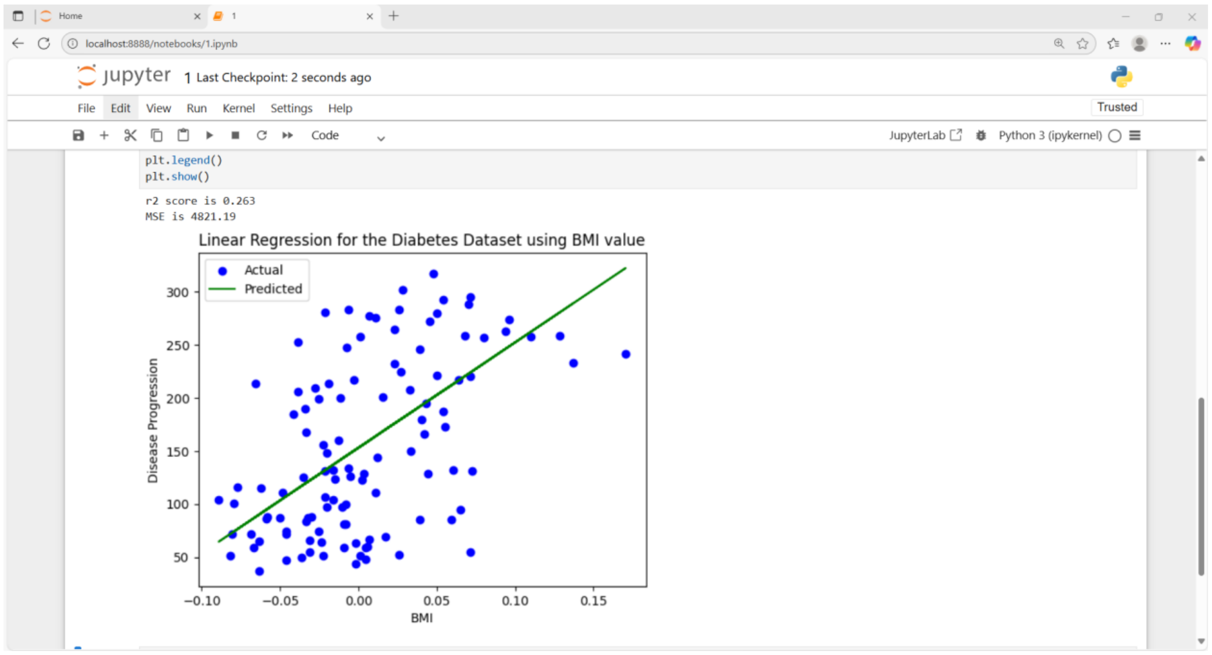
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
data = load_diabetes()
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



```
plt.legend()
plt.show()
```

```
r2 score is 0.263
MSE is 4821.19
```

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

a) Dealing with categorical data

b) Scaling the features

c) 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.

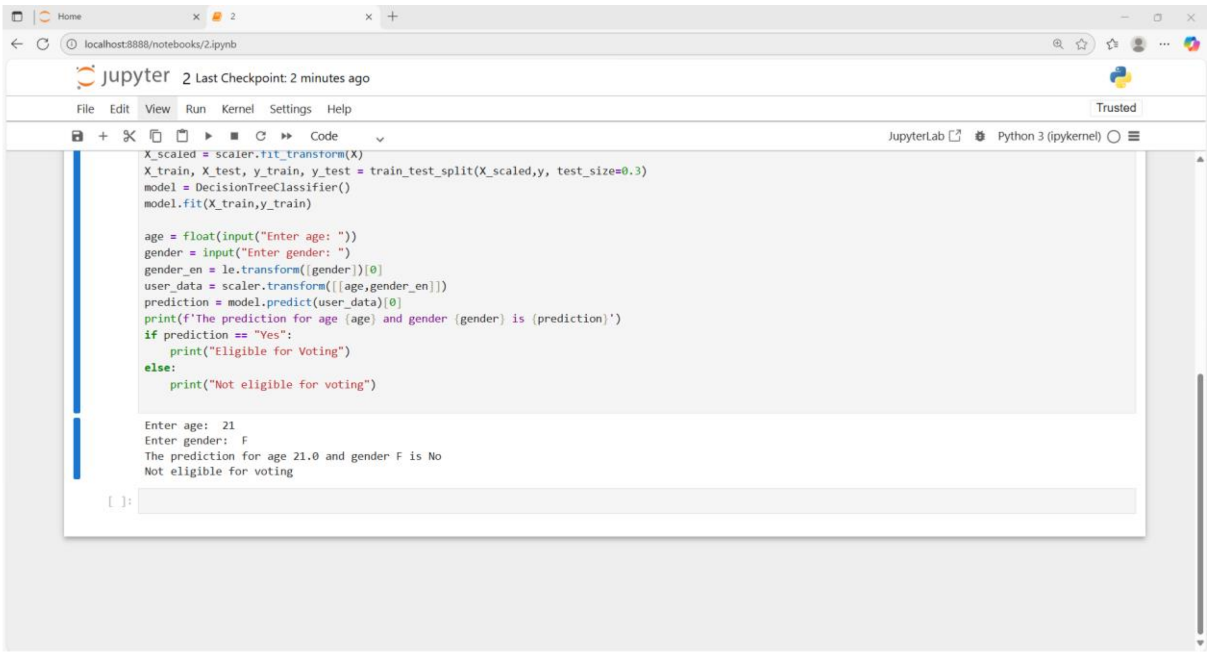
STEP 11: Predict the eligibility and display the result.

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



```
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: ")
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")
```

```
Enter age:  21
Enter gender:  F
The prediction for age 21.0 and gender F is No
Not eligible for voting
```

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:

a. Pearson’s Correlation

b. Euclidean Distance

c. 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.

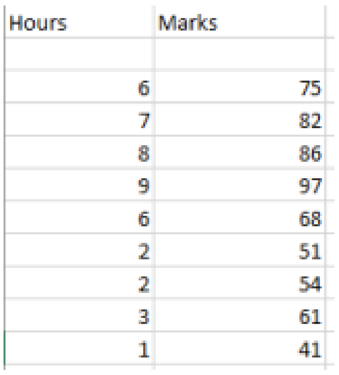
STEP 10: Display the correlation and distance values.

STEP 11: End the program.

PROGRAM

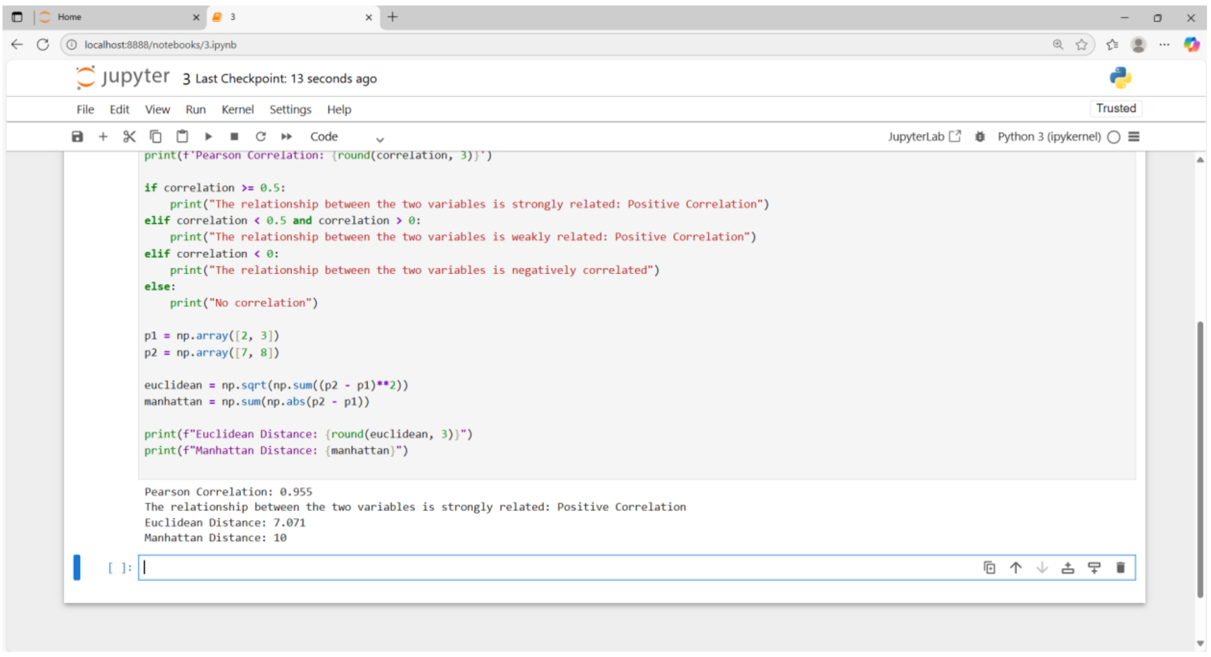
```
import pandas as pd

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



Hours	Marks
6	75
7	82
8	86
9	97
6	68
2	51
2	54
3	61
1	41

OUTPUT


Home
3
localhost:8888/notebooks/3.ipynb
jupyter 3 Last Checkpoint: 13 seconds ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)

```
print(f'Pearson Correlation: {round(correlation, 3)}')

if correlation >= 0.5:
    print("The relationship between the two variables is strongly related: Positive Correlation")
elif correlation < 0.5 and correlation > 0:
    print("The relationship between the two variables is weakly related: Positive Correlation")
elif correlation < 0:
    print("The relationship between the two variables is negatively correlated")
else:
    print("No correlation")

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}")
```

```
Pearson Correlation: 0.955
The relationship between the two variables is strongly related: Positive Correlation
Euclidean Distance: 7.071
Manhattan Distance: 10
```

```
[ ]: |
```

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.

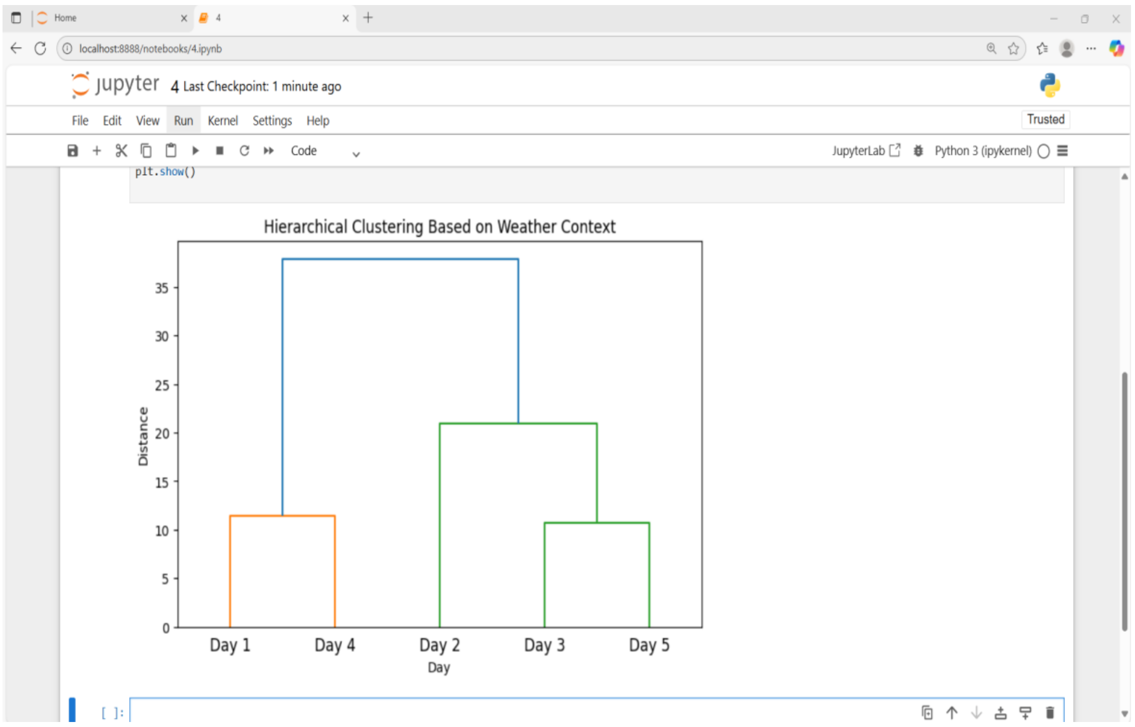
STEP 7: Display the dendrogram.

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



```
plt.show()
```

Hierarchical Clustering Based on Weather Context
Distance
0
5
10
15
20
25
30
35
Day 1
Day 4
Day 2
Day 3
Day 5
Day
localhost:8888/notebooks/4.ipynb
jupyter 4 Last Checkpoint: 1 minute ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
[ ]:

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.

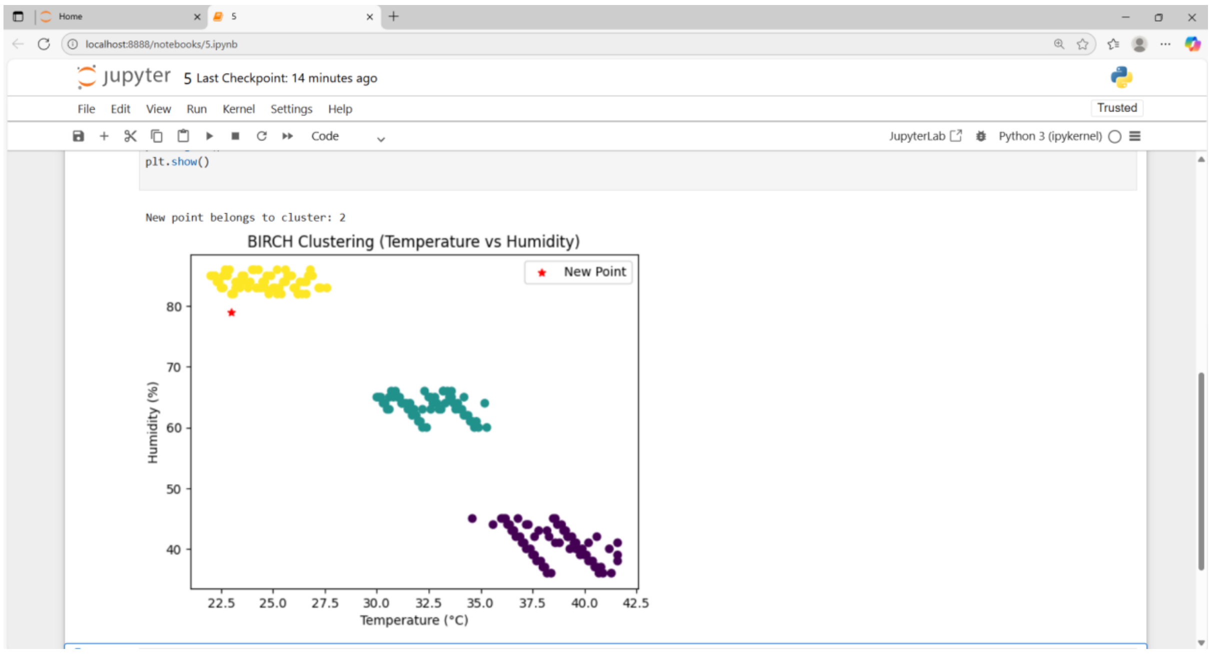
STEP 9: Visualize the clusters along with the new data point using a scatter plot.

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



```
plt.show()
```

```
New point belongs to cluster: 2
```

Home
5
localhost:8888/notebooks/5.ipynb
jupyter 5 Last Checkpoint: 14 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
BIRCH Clustering (Temperature vs Humidity)
New Point
Humidity (%)
Temperature (°C)

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.

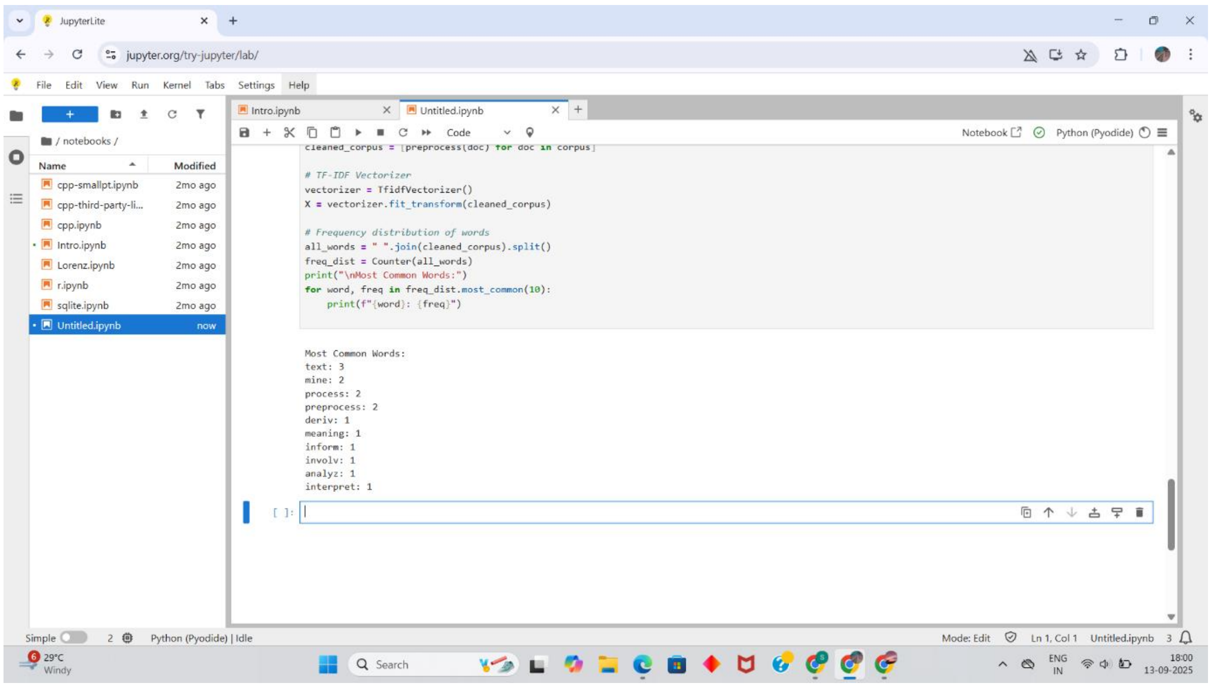
Step 8: Display the top ten most frequent words with their frequencies.

PROGRAM

```
from sklearn.feature_extraction.text import TfidfVectorizer, ENGLISH_STOP_WORDS
from nltk.stem import PorterStemmer
import pandas as pd
import re
from collections import Counter

# Corpus
corpus = [
&quot;Text mining is the process of deriving meaningful information from text.&quot;,
&quot;It involves preprocessing, analyzing, and interpreting textual data.&quot;,
&quot;Machine learning and natural language processing are often used in text
mining.&quot;,
&quot;Stopwords and punctuation need to be removed in preprocessing.&quot;,
&quot;Stemming helps in reducing words to their root form.&quot;
]

# Preprocessing function
def preprocess(text):
# Lowercase, remove punctuation, simple split
tokens = re.findall(r&#39;\b\w+\b&#39;, text.lower())
stop_words = ENGLISH_STOP_WORDS
stemmer = PorterStemmer()
cleaned_tokens = [stemmer.stem(word) for word in tokens if word.isalpha() and word not
in stop_words]
return &quot; &quot;.join(cleaned_tokens)

# Apply preprocessing
cleaned_corpus = [preprocess(doc) for doc in corpus]

# TF-IDF Vectorizer
```

```
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(cleaned_corpus)

# Frequency distribution of words
all_words = &quot; &quot;.join(cleaned_corpus).split()
freq_dist = Counter(all_words)
print(&quot;\nMost Common Words:&quot;)
for word, freq in freq_dist.most_common(10):
print(f&quot;{word}: {freq}&quot;)
```

OUTPUT



```
cleaned_corpus = [preprocess(doc) for doc in corpus]

# TF-IDF Vectorizer
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(cleaned_corpus)

# Frequency distribution of words
all_words = " ".join(cleaned_corpus).split()
freq_dist = Counter(all_words)
print("\nMost Common Words:")
for word, freq in freq_dist.most_common(10):
    print(f"{word}: {freq}")
```

```
Most Common Words:
text: 3
mine: 2
process: 2
preprocess: 2
deriv: 1
meaning: 1
inform: 1
involv: 1
analyz: 1
interpret: 1
```

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.

CODE:

```
 > 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

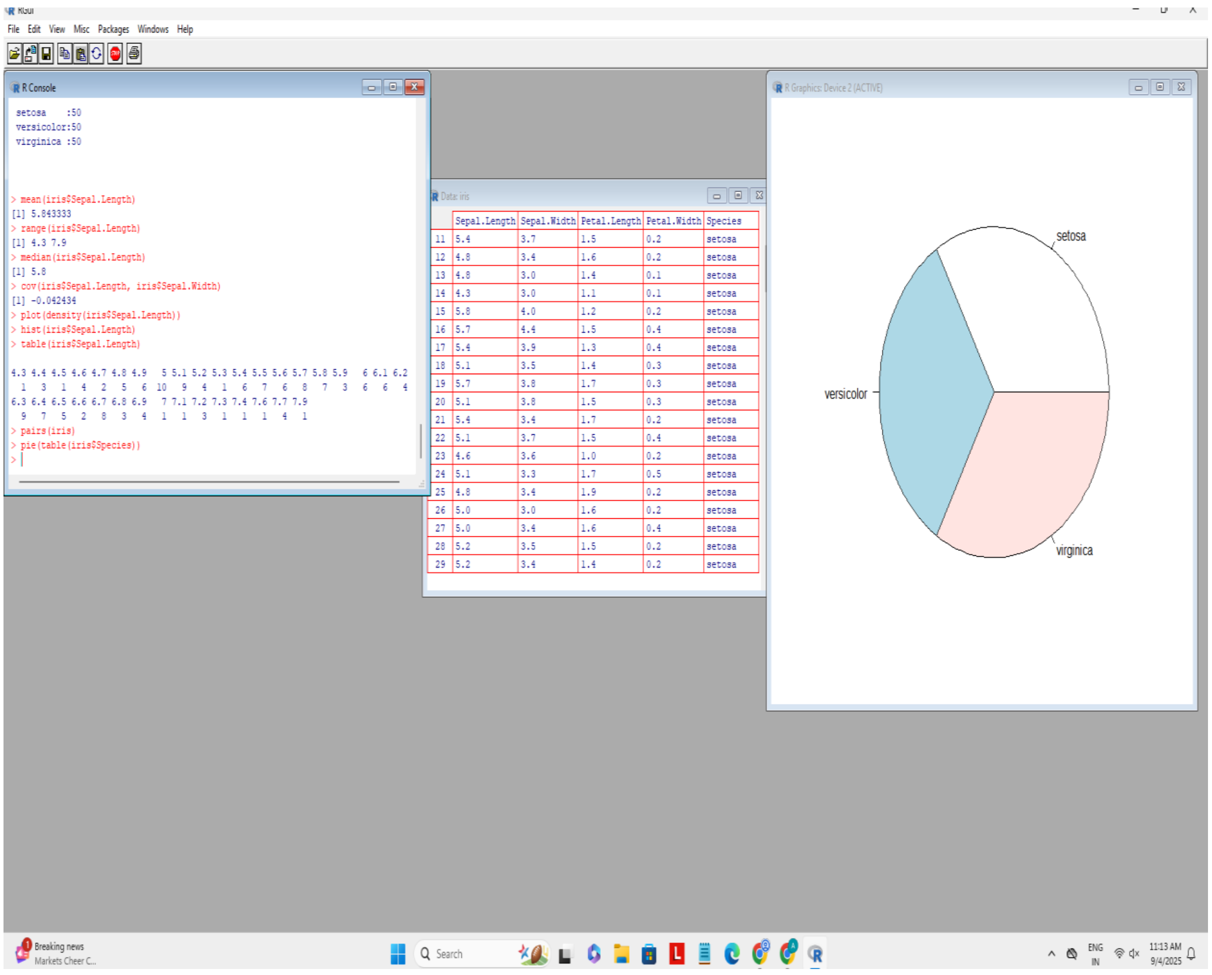
```
144         6.8        3.2         5.9        2.3  virginica
145         6.7        3.3         5.7        2.5  virginica
146         6.7        3.0         5.2        2.3  virginica
147         6.3        2.5         5.0        1.9  virginica
148         6.5        3.0         5.2        2.0  virginica
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


RGui
File Edit View Misc Packages Windows Help
R Console
Data: iris
R Graphics: Device 2 (ACTIVE)
setosa
versicolor
virginica

```
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))
>
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
11	5.4	3.7	1.5	0.2	setosa
12	4.8	3.4	1.6	0.2	setosa
13	4.8	3.0	1.4	0.1	setosa
14	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

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.

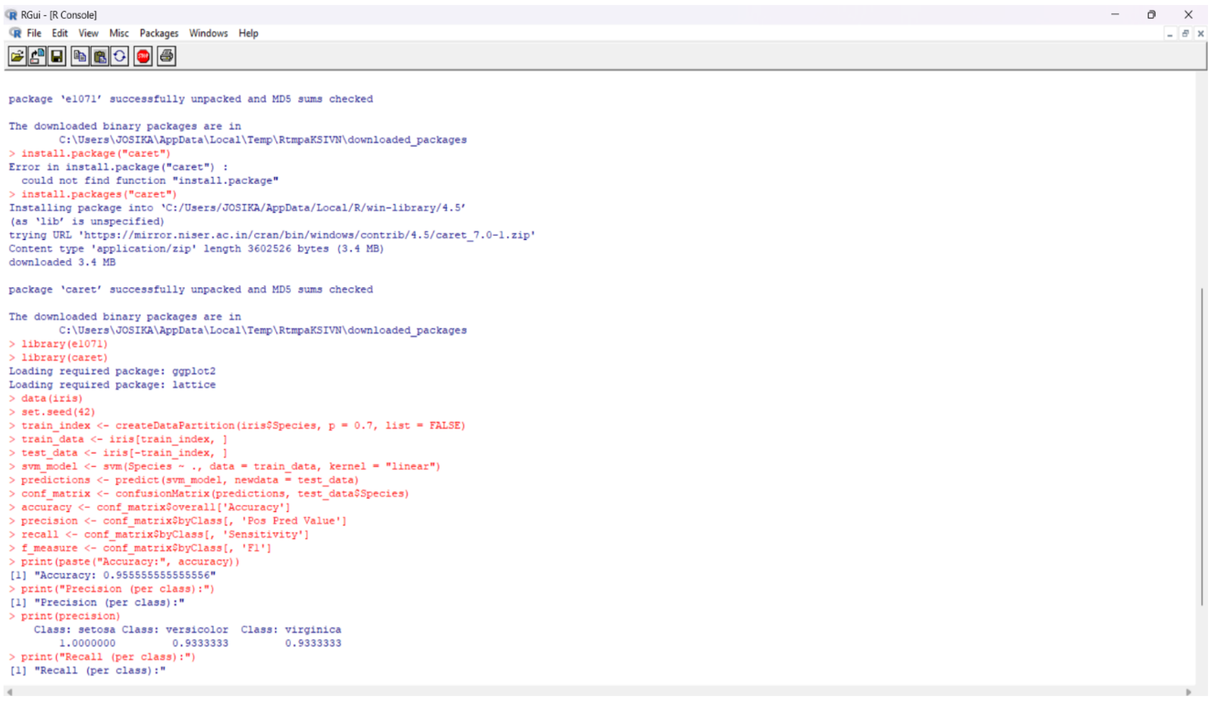
Step 9: Display the results.

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



```
package 'e1071' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\JOSIKA\AppData\Local\Temp\RtmpaKSIVN\downloaded_packages
> install.package("caret")
Error in install.package("caret") :
  could not find function "install.package"
> install.packages("caret")
Installing package into 'C:/Users/JOSIKA/AppData/Local/R/win-library/4.5'
(as 'lib' is unspecified)
trying URL 'https://mirror.niser.ac.in/cran/bin/windows/contrib/4.5/caret_7.0-1.zip'
Content type 'application/zip' length 3602526 bytes (3.4 MB)
downloaded 3.4 MB

package 'caret' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\JOSIKA\AppData\Local\Temp\RtmpaKSIVN\downloaded_packages
> library(e1071)
> library(caret)
Loading required package: ggplot2
Loading required package: lattice
> 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))
[1] "Accuracy: 0.955555555555556"
> print("Precision (per class):")
[1] "Precision (per class):"
> print(precision)
    Class: setosa Class: versicolor  Class: virginica
        1.0000000         0.9333333         0.9333333
> print("Recall (per class):")
[1] "Recall (per class):"
```

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.

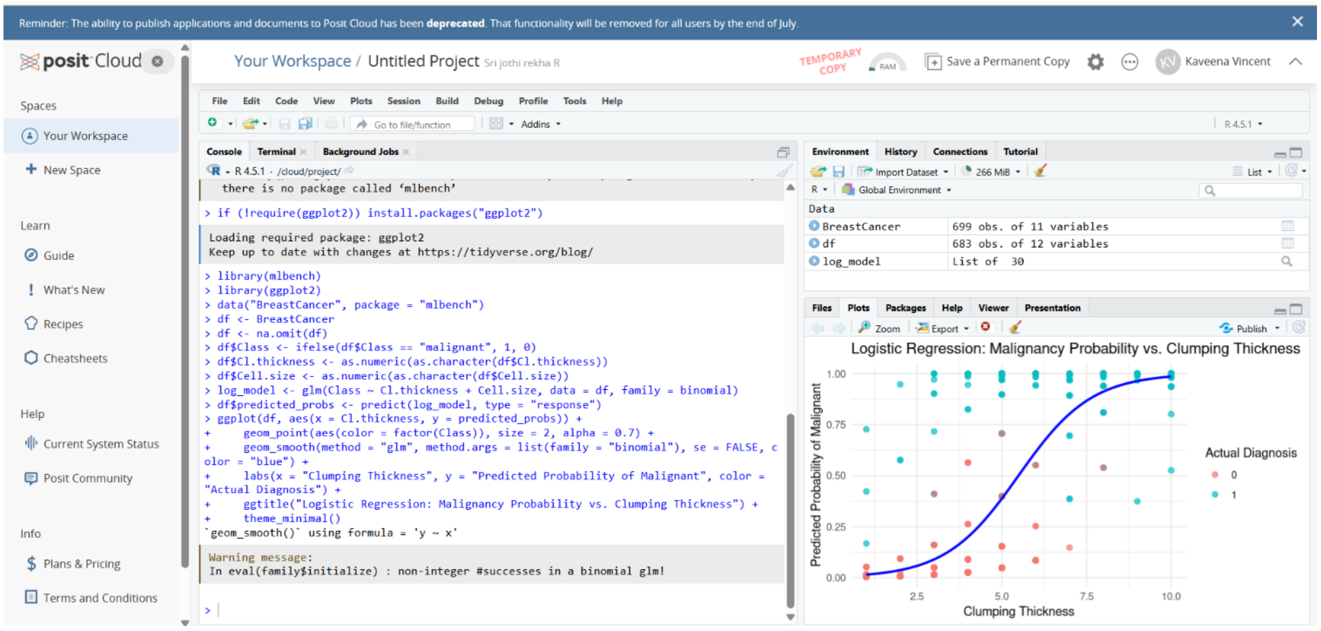
STEP 12: Visualize results using ggplot2 with scatter plot and logistic curve.

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


Reminder: The ability to publish applications and documents to Posit Cloud has been **deprecated**. That functionality will be removed for all users by the end of July.
posit Cloud
Your Workspace / Untitled Project Sri jothi rekha R
TEMPORARY COPY
RAM
Save a Permanent Copy
Kaveena Vincent
Spaces
Your Workspace
New Space
Learn
Guide
What's New
Recipes
Cheatsheets
Help
Current System Status
Posit Community
Info
Plans & Pricing
Terms and Conditions
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
R 4.5.1
Console Terminal Background Jobs
R 4.5.1 · /cloud/project/
Environment History Connections Tutorial
Import Dataset
266 MiB
List
Global Environment
Files Plots Packages Help Viewer Presentation
Zoom
Export
Publish
Logistic Regression: Malignancy Probability vs. Clumping Thickness
Predicted Probability of Malignant
1.00
0.75
0.50
0.25
0.00
2.5
5.0
7.5
10.0
Clumping Thickness
Actual Diagnosis
0
1

```
  there is no package called 'mlbench'

> if (!require(ggplot2)) install.packages("ggplot2")

 Loading required package: ggplot2
 Keep up to date with changes at https://tidyverse.org/blog/

> 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, c
olor = "blue") +
+     labs(x = "Clumping Thickness", y = "Predicted Probability of Malignant", color =
"Actual Diagnosis") +
+     ggtitle("Logistic Regression: Malignancy Probability vs. Clumping Thickness") +
+     theme_minimal()
`geom_smooth()` using formula = 'y ~ x'

 Warning message:
 In eval(family$initialize) : non-integer #successes in a binomial glm!

> 
```

Data	
BreastCancer	699 obs. of 11 variables
df	683 obs. of 12 variables
log_model	List of 30

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).

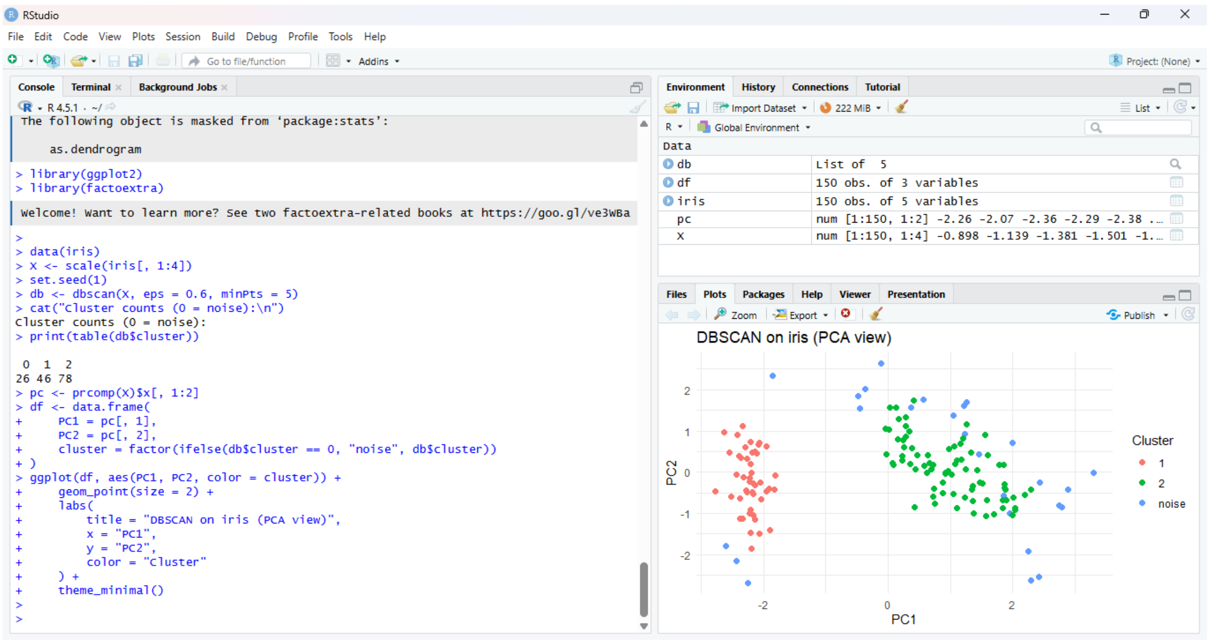
STEP 7: Project data to 2D (PCA) for visualization.

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



```
The following object is masked from ‘package:stats’:

    as.dendrogram

> library(ggplot2)
> library(factoextra)
Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa

>
> data(iris)
> X <- scale(iris[, 1:4])
> set.seed(1)
> db <- dbscan(X, eps = 0.6, minPts = 5)
> cat("Cluster counts (0 = noise):\n")
Cluster counts (0 = noise):
> print(table(db$cluster))

 0  1  2 
26 46 78 
> 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()
>
>
```

```
Data
db      List of  5
df      150 obs. of 3 variables
iris    150 obs. of 5 variables
pc      num [1:150, 1:2] -2.26 -2.07 -2.36 -2.29 -2.38 ...
X       num [1:150, 1:4] -0.898 -1.139 -1.381 -1.501 -1....
```

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.

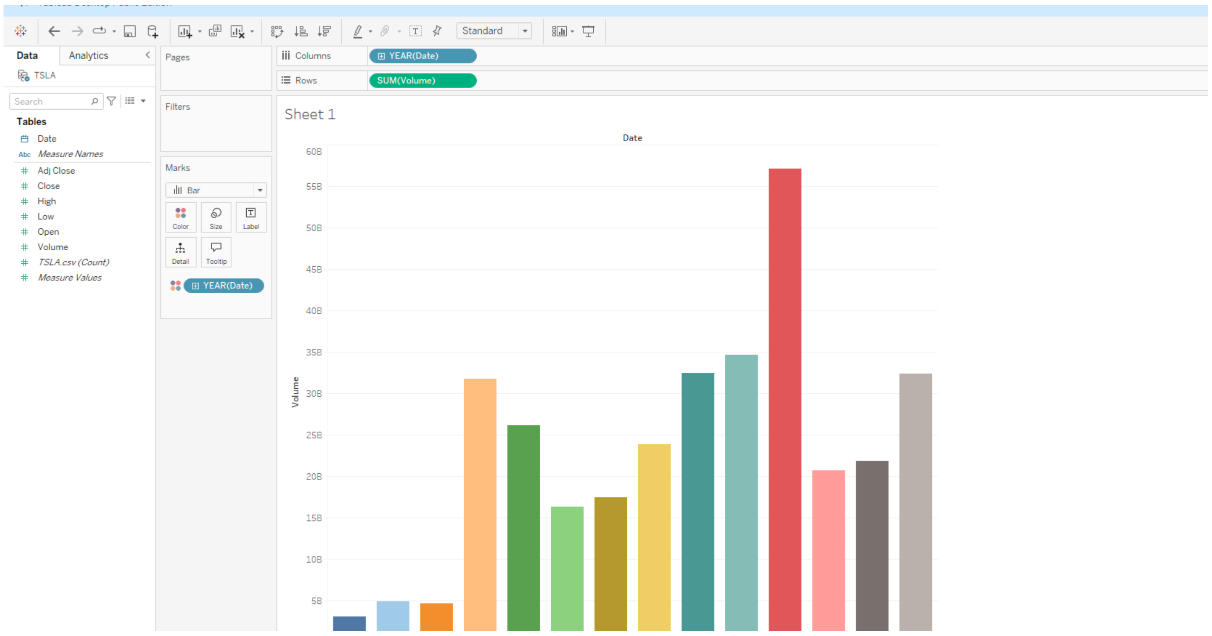
STEP 6: Combine into Dashboard.Go to New Dashboard.

Drag Sheet 1 , Sheet 2 , and Sheet 3 into the dashboard.

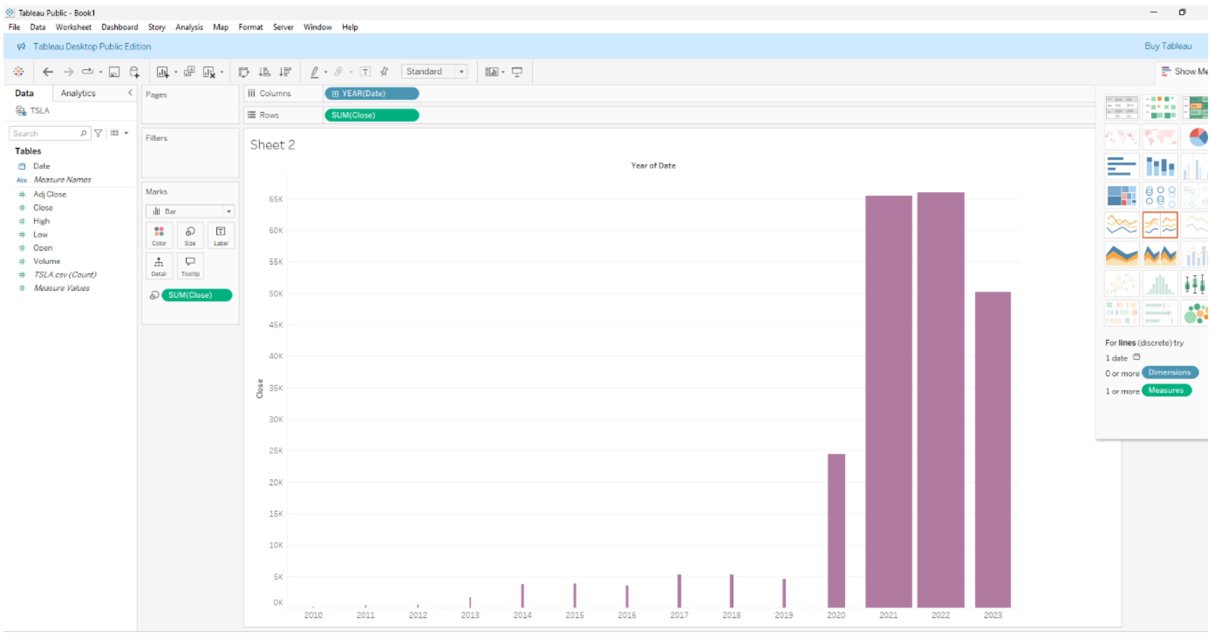
STEP 7:Arrange charts side by side or top–bottom.

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

OUTPUT



Columns: YEAR(Date)
Rows: SUM(Volume)
Sheet 1
Marks: Bar, Color: YEAR(Date)



Columns: YEAR(Date)
Rows: SUM(Close)
Sheet 2
Marks: Bar, SUM(Close)

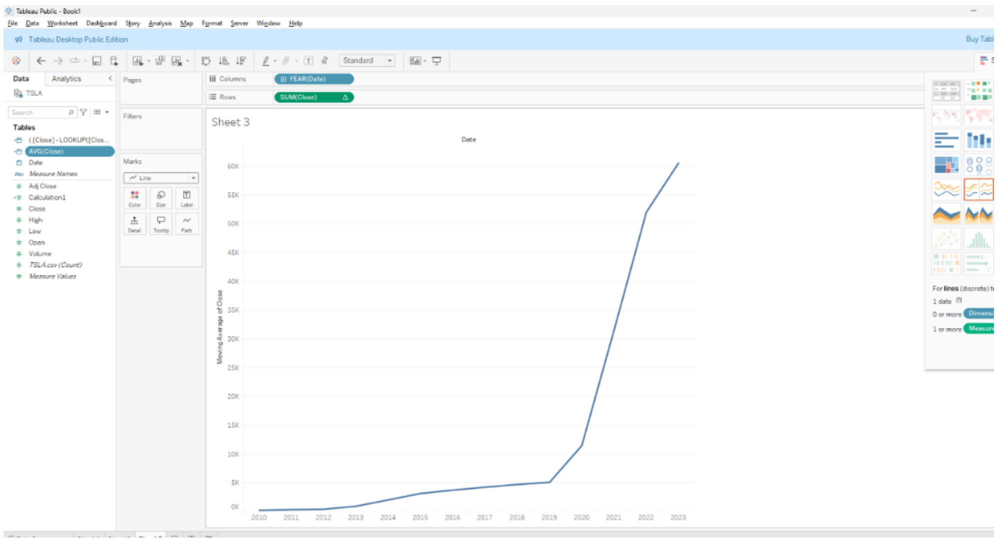
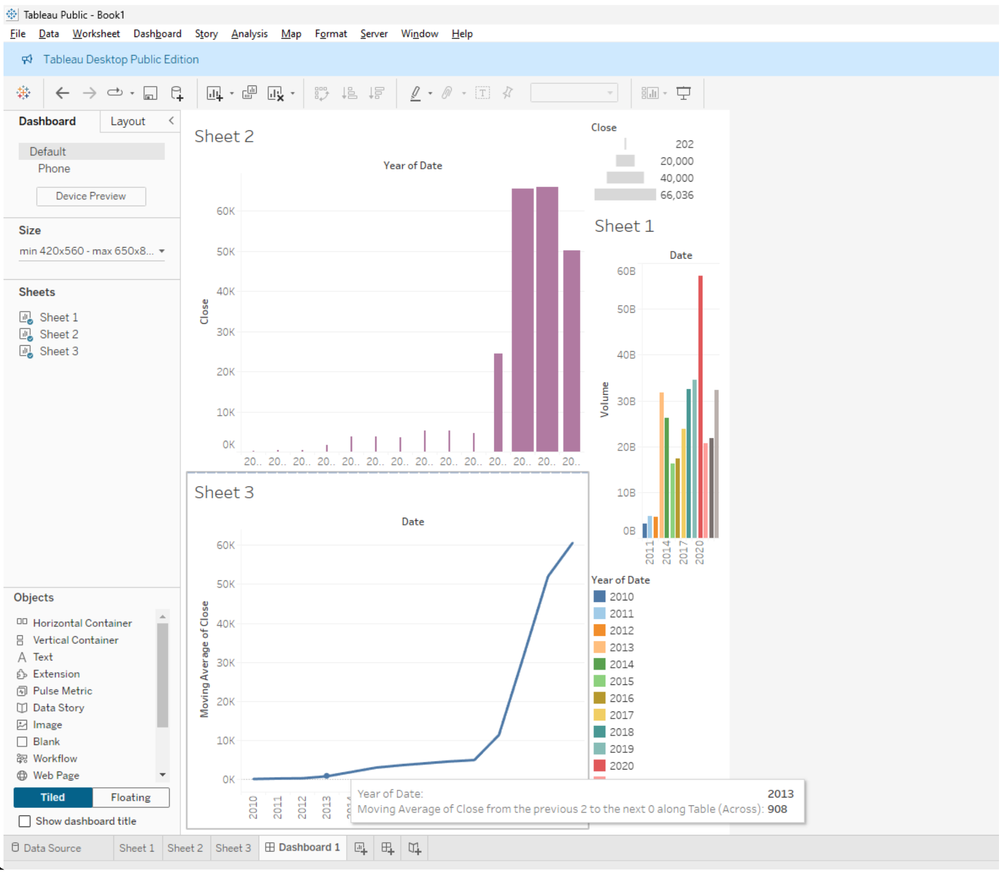


Tableau Public - Book1
File Data Worksheet Dashboard Story Analysis Map Format Server Window Help
Tableau Desktop Public Edition
Columns YEAR(Date)
Rows SUM(Close)
Sheet 3
Date
Moving Average of Close



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.

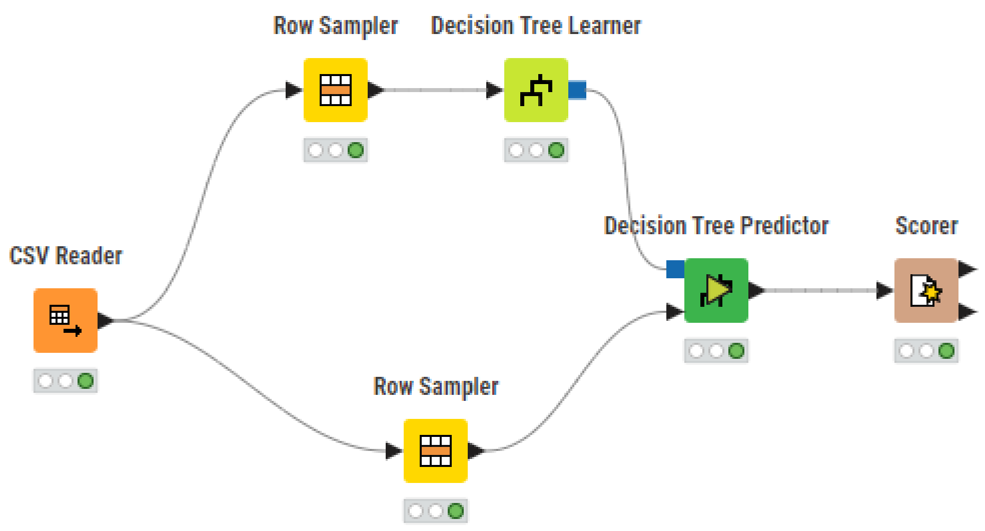
STEP 5: Use the trained model to predict labels for the test set.

STEP 6: Compute the confusion matrix and accuracy score.

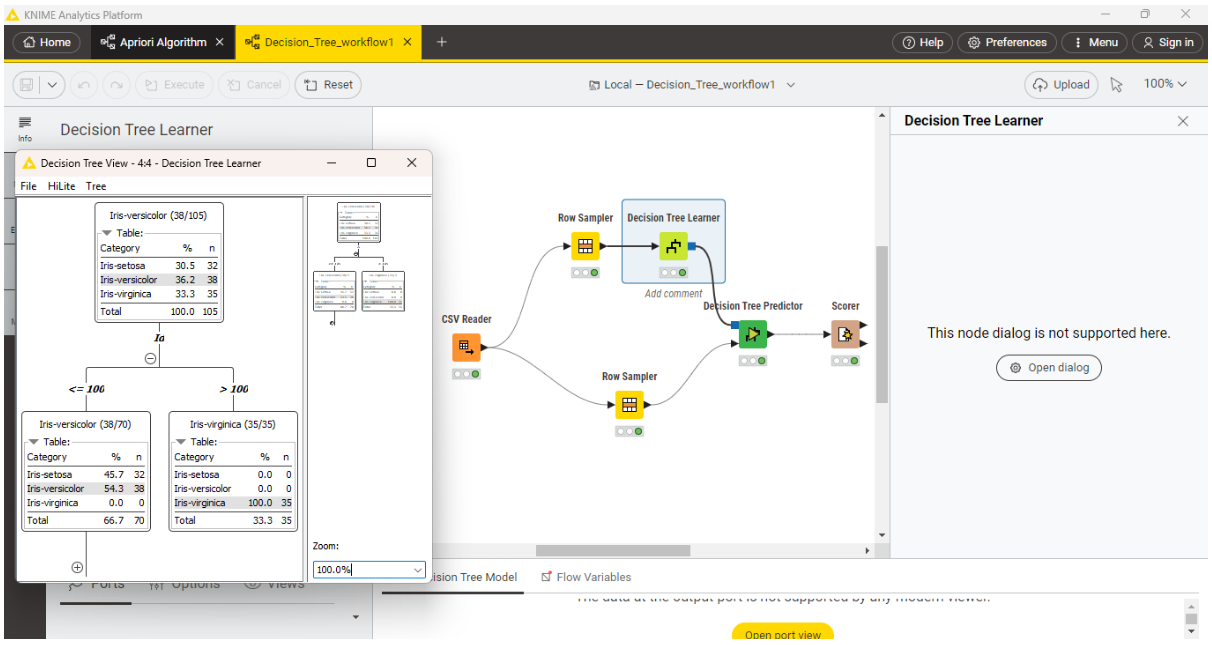
STEP 7: Visualize the decision tree diagram

STEP 8: Stop the process

PROGRAM STRUCTURE



OUTPUT



Decision Tree Learner

Decision Tree View - 4:4 - Decision Tree Learner

Iris-versicolor (38/105)

Table:

Category	%	n
Iris-setosa	30.5	32
Iris-versicolor	36.2	38
Iris-virginica	33.3	35
Total	100.0	105

Id

<= 100

Iris-versicolor (38/70)

Table:

Category	%	n
Iris-setosa	45.7	32
Iris-versicolor	54.3	38
Iris-virginica	0.0	0
Total	66.7	70

> 100

Iris-virginica (35/35)

Table:

Category	%	n
Iris-setosa	0.0	0
Iris-versicolor	0.0	0
Iris-virginica	100.0	35
Total	33.3	35

Decision Tree Learner

This node dialog is not supported here.

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.

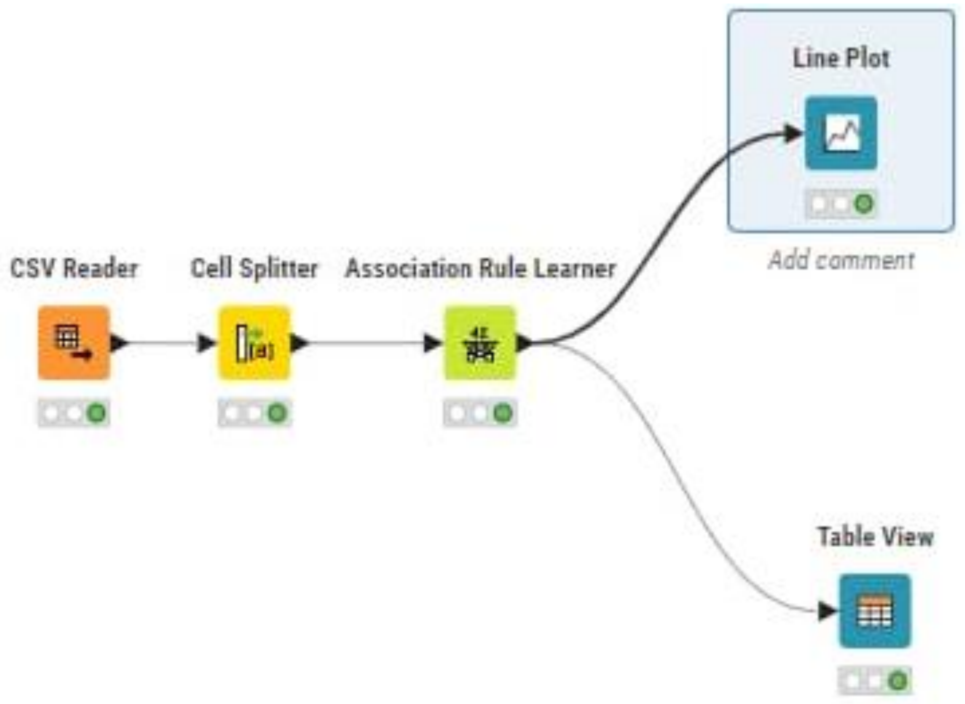
STEP 5: Execute Apriori Algorithm

STEP 6: Connect the Association Rule Learner to the Association Rule Viewer node.

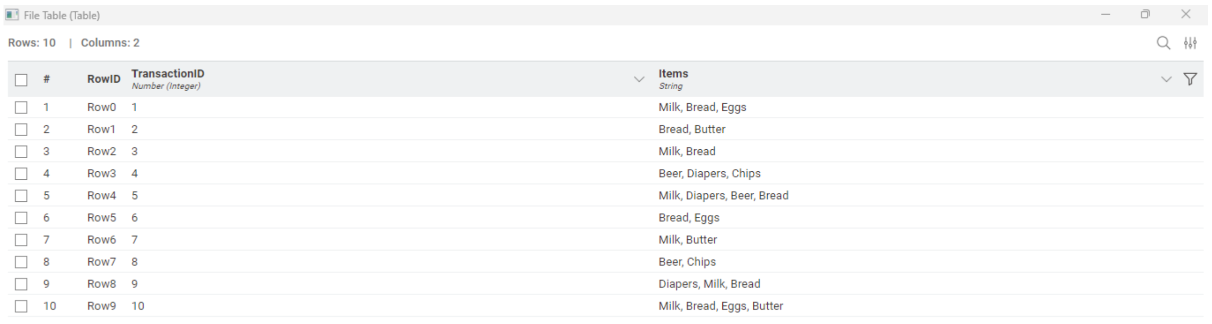
STEP 7: Analyse Results by using the Rule Viewer to filter and interpret patterns

STEP 8: End the program.

PROGRAM STRUCTURE



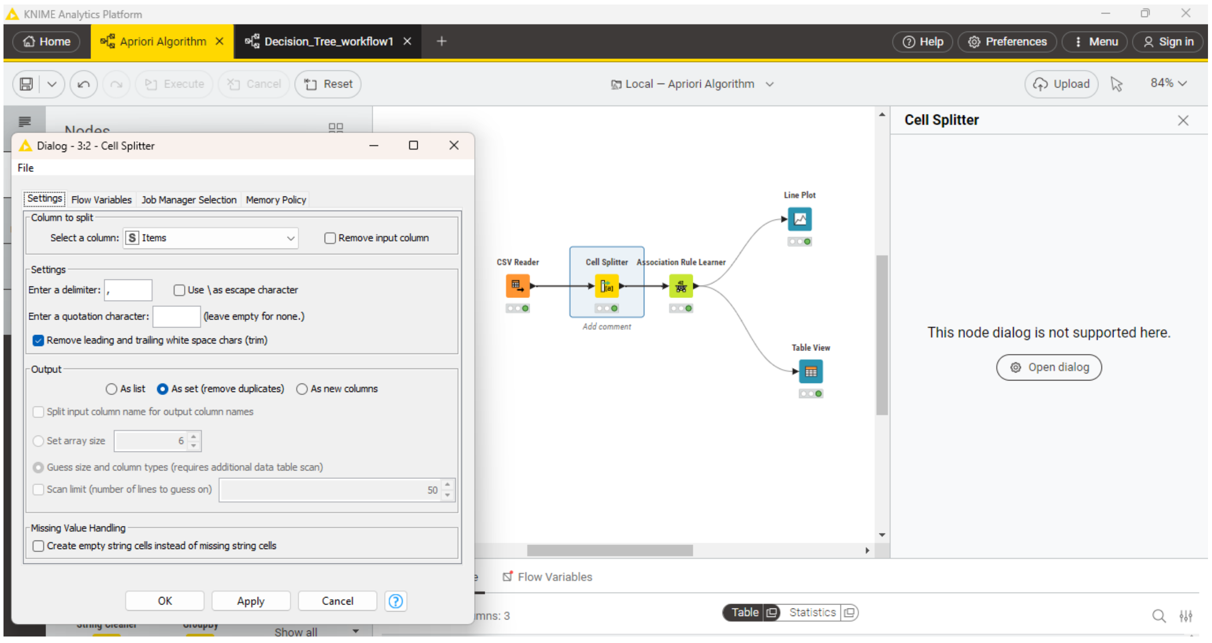
OUTPUT



File Table (Table)

Rows: 10 | Columns: 2

#	RowID	TransactionID *Number (Integer)*	Items *String*
1	Row0	1	Milk, Bread, Eggs
2	Row1	2	Bread, Butter
3	Row2	3	Milk, Bread
4	Row3	4	Beer, Diapers, Chips
5	Row4	5	Milk, Diapers, Beer, Bread
6	Row5	6	Bread, Eggs
7	Row6	7	Milk, Butter
8	Row7	8	Beer, Chips
9	Row8	9	Diapers, Milk, Bread
10	Row9	10	Milk, Bread, Eggs, Butter



Dialog - 3:2 - Cell Splitter

File

Settings | Flow Variables | Job Manager Selection | Memory Policy

Column to split

Select a column: Items   [ ] Remove input column

Settings

Enter a delimiter: ,   [ ] Use \ as escape character

Enter a quotation character: (leave empty for none.)

[✓] Remove leading and trailing white space chars (trim)

Output

As list [✓ As set (remove duplicates)] As new columns

[ ] Split input column name for output column names

Set array size 6

Guess size and column types (requires additional data table scan)

[ ] Scan limit (number of lines to guess on) 50

Missing Value Handling

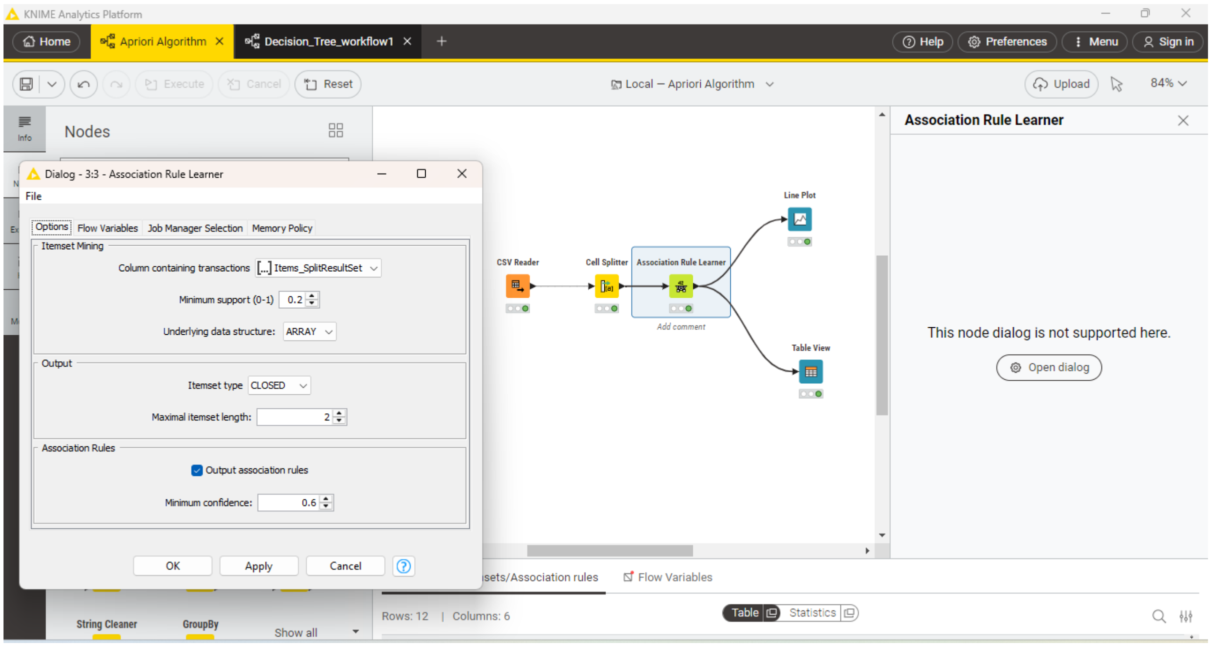
[ ] Create empty string cells instead of missing string cells

OK | Apply | Cancel

Cell Splitter

This node dialog is not supported here.

Open dialog


Dialog - 3:3 - Association Rule Learner

Itemset Mining

Column containing transactions: Items_SplitResultSet

Minimum support (0-1): 0.2

Underlying data structure: ARRAY

Output

Itemset type: CLOSED

Maximal itemset length: 2

Association Rules

[✓] Output association rules

Minimum confidence: 0.6

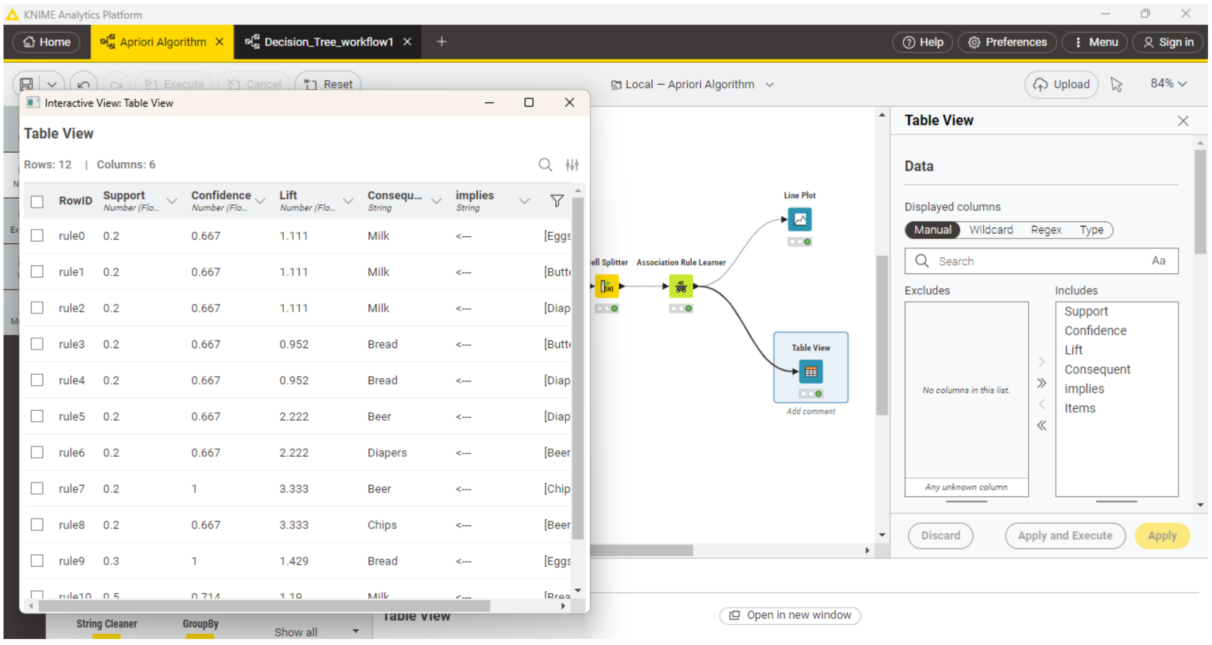

Interactive View: Table View

Table View

Rows: 12 | Columns: 6

RowID	Support	Confidence	Lift	Consequ...	implies	
	Number (Flo...	Number (Flo...	Number (Flo...	String	String	
rule0	0.2	0.667	1.111	Milk	<---	[Eggs
rule1	0.2	0.667	1.111	Milk	<---	[Butt
rule2	0.2	0.667	1.111	Milk	<---	[Diap
rule3	0.2	0.667	0.952	Bread	<---	[Butt
rule4	0.2	0.667	0.952	Bread	<---	[Diap
rule5	0.2	0.667	2.222	Beer	<---	[Diap
rule6	0.2	0.667	2.222	Diapers	<---	[Beer
rule7	0.2	1	3.333	Beer	<---	[Chip
rule8	0.2	0.667	3.333	Chips	<---	[Beer
rule9	0.3	1	1.429	Bread	<---	[Eggs
rule10	0.5	0.714	1.19	Milk	<---	[Brea

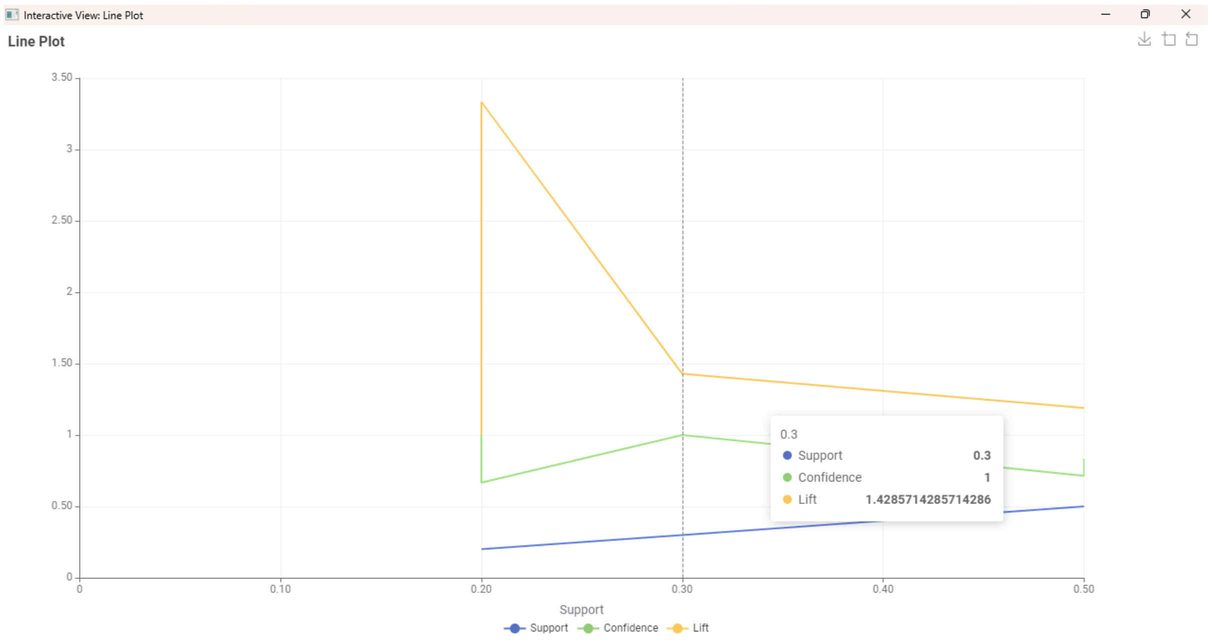
Table View

Data

Displayed columns: Manual

Excludes: No columns in this list.

Includes: Support, Confidence, Lift, Consequent, implies, Items



```
0.3
● Support                    0.3
● Confidence                   1
● Lift          1.4285714285714286
```

RESULT

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