LABORATORY MANUAL

For

DATA MINING LAB

(DM LAB)

Department

Of

Artificial Intelligence and Machine Learning

SRINIVASA

INSTITUTE OF ENGINEERING AND TECHNOLOGY

(UGC – Autonomous Institution) (Approved by AICTE, permanently affiliated to JNTUK, Kakinada, ISO 9001: 2015 certified Institution) (Accredited by NAAC with 'A' Grade; Recognized by UGC under sections 2(f) & 12(B)) NH-216, Amalapuram-Kakinada Highway, Cheyyeru (V), AMALAPURAM -53321

DATA MINING LAB

Course Outcomes:

At the end of the course, the students will be able to

1. apply preprocessing techniques on real world datasets
2. apply apriori algorithm to generate frequent item sets.
3. apply Classification and clustering algorithms on different datasets
4. build a model using linear regression algorithm on any dataset.
5. build a classification model using Decision Tree algorithm on iris dataset

List of Experiments:

1. 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
2. Demonstrate the following data preprocessing tasks using python libraries. a) Dealing with categorical data b) Scaling the features c)
Splitting dataset into Training and Testing Sets
3. Demonstrate the following Similarity and Dissimilarity Measures using python a) Pearson's Correlation b) Cosine Similarity c) Jaccard Similarity d) Euclidean Distance e) Manhattan Distance
4. Build a model using linear regression algorithm on any dataset.
5. Build a classification model using Decision Tree algorithm on iris dataset
6. Apply Naïve Bayes Classification algorithm on any dataset
7. Generate frequent itemset using Apriori Algorithm in python and also generate association rules for any market basket data.
8. Apply K- Means clustering algorithm on any dataset.
9. Apply Hierarchical Clustering algorithm on any dataset.
10. 10.Apply DBSCAN clustering algorithm on any dataset.

II.PROGRAMME OUTCOMES(PO's)

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature, and analyses complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to
the professional engineering practices

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in
societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions. PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

EXPERIMENT 1

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

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

DESCRIPTION:

Loading the dataset

Loading the dataset There 4 Different Ways to Load Data in Python Manual function loadtxt function read_csv function Pandas is a very popular data manipulation library, and it is very commonly used. One of it's very important and mature functions is read_csv() which can read any .csv file very easily and help us manipulate it. **Dealing with missing data What Is a Missing Value?** Missing data is defined as the values or data that is not stored (or no t present) for in the given dataset.

Handling Missing Values

There are 2 primary ways of handling missing values:

1. Deleting the Missing values
2. Imputing the Missing Values **`pandas.isnull()`:**
   - This function is used to detect missing values (including `NaN`, `None`, and `NaT`) in a Series or DataFrame.
   - It returns a boolean mask (a Series or DataFrame of True/False values) indicating whether each element is null.

DATASET

NAME	ROLL NO	DOB	ADDRESS	INCOME	BANK ACCOUNT
Priya	101	31-12-2000	kothapeta	**40**	**YES**
sreya	102	24-07-2000	amalpuram	**50**	**NO**
hansi	103	01-01-2000	yanam	**NAN**	**YES**
anshu	104	03-05-2005	kakinada	**80**	**NO**

PROGRAM:

```
import pandas as pd
df=pd.read_csv(r"data.csv")
print(df)
print("to print col wise of a data on the data frame")
print("***************")
print(df.iloc[:,1:5])
print(df.iloc[:,5:6])
print("to print the specific column null values")

# Check for missing values in the entire DataFrame
df.isnull()

# Check for missing values in a specific column
df['income'].isnull()

# Count  missing values in a column
df['income'].isnull().sum()
```

OUTPUT

```
name    rollno  dob     address income
        bank account
0   k srinithin     30   27-08-004    kothapeta     40         yes
1   p sai teja       50 20-01-2003 pasralapudi     50          no
```

2 k s p ankith 26 15-06-2004 annampalli 30 yes
3 b sri ram 8 26-02-2003 krapa NAN no
4 r.s.m kanta 53 04-04-2003 kakinada 80 yes
to print col wise of a data on the dataframe
****

	rollno	dob	address	income	bank account
0	30	27-08-004	kothapeta	40	yes
1	50	20-01-2003	pasralapudi	50	no
2	26	15-06-2004	annampalli	30	yes
3	8	26-02-2003	krapa	NAN	no
4	53	04-04-2003	kakinada	80	yes

1

EXPERIMENT 2

2. Demonstrate the following data preprocessing tasks using python libraries. a) Dealing with categorical data b) Scaling the features c) Splitting dataset into Training and Testing Sets

AIM:

To Demonstrate the following data preprocessing tasks using python libraries.

a) Dealing with categorical data b) Scaling the features c) Splitting dataset into Training and Testing Sets

DESCRIPTION

DEALING WITH CATEGORICAL DATA:

Categorical data, which represents groups or categories rather than numerical values, poses a unique challenge in machine learning algorithms that often expect numerical inputs. Here are some common techniques to handle categorical data:

1. Encoding Techniques:

- One-Hot Encoding:

    - Creates a new binary column for each category within a feature.
    - A "1" indicates the presence of that category, and "0" indicates its absence.

○ Example: | Color | Red | Green | Blue | |---|---|---|---| | Red | 1 | 0 | 0 | | Green | 0 | 1 | 0 | | Blue | 0 | 0 | 1 | ● Label Encoding:

    - Assigns a unique integer to each category.
    - Preserves the order if the categories have an inherent order (e.g., low, medium, high).

    - Example: | Size | Label | |---|---| | Small | 0 | | Medium | 1 | | Large | 2 |

Splitting a Dataset into Training and Test Sets

Why Split?

- **Model Evaluation:** The primary reason is to evaluate how well your machine learning model generalizes to unseen data.
- **Preventing Overfitting:** By training on one set and testing on another, you can identify if your model is memorizing the training data rather than learning underlying patterns.

Common Approach: Train-Test Split

1. **Divide the Dataset:**
    - **Training Set:** Used to train the machine learning model. Typically, a larger portion of the data (e.g., 70-80%).
    - **Test Set:** Used to evaluate the trained model's performance on unseen data. Typically, a smaller portion (e.g., 20-30%).
2. **Ensure Representativeness:**
    - The split should be random to avoid biases.
    - For imbalanced datasets (where classes have unequal representation), consider stratified sampling to maintain class proportions in both sets.

PROGRAM:

```
import pandas as pd from sklearn.preprocessing import
OneHotEncoder, MinMaxScaler from
sklearn.model_selection import train_test_split

# Sample data
data = {'name': ['Alice', 'Bob', 'Charlie', 'David'],
        'rollno': [1, 2, 3, 4],
        'branch': ['CSE', 'ECE', 'MECH', 'CSE'],
```

```
        'address': ['Addr1', 'Addr2', 'Addr3',
'Addr4']} df = pd.DataFrame(data)

# Dealing with Categorical Data encoder =
OneHotEncoder(sparse_output=False, handle_unknown='ignore')
encoded_data = encoder.fit_transform(df[['branch']]) encoded_df =
pd.DataFrame(encoded_data,
columns=encoder.get_feature_names_out(['branch'])) df =
pd.concat([df, encoded_df], axis=1) df = df.drop(['branch'], axis=1)

# Print categorical data print("Categorical Data (One-Hot
Encoded):\n", df[['name', 'branch_CSE', 'branch_ECE',
'branch_MECH']].head())


# Scaling the
Feature scaler =
MinMaxScaler()
df['rollno_scaled'] = scaler.fit_transform(df[['rollno']])

# Print scaled features print("\nScaled Features:\n",
df[['rollno', 'rollno_scaled']].head())


# Splitting Dataset into Training and Testing Sets
X = df[['rollno_scaled', 'branch_CSE', 'branch_ECE',
'branch_MECH']] y = df['rollno_scaled'] # Assuming
'rollno_scaled' is the target X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2, random_state=42)

# Print train and test datasets
print("\nTraining Data (X_train):\n",
X_train.head()) print("\nTesting Data
(X_test):\n", X_test.head()) print("\nTraining
Target (y_train):\n", y_train.head())
print("\nTesting Target (y_test):\n",
y_test.head())
```

OUTPUT:

Categorical Data (One-Hot Encoded): name branch_CSE branch_ECE branch_MECH

	name	branch_CSE	branch_ECE	branch_MECH
0	Alice	1.0	0.0	0.0
1	Bob	0.0	1.0	0.0
2	Charlie	0.0	0.0	1.0
3	David	1.0	0.0	0.0

Scaled Features:

	rollno	rollno_scaled
0	1	0.000000
1	2	0.333333
2	3	0.666667
3	4	1.000000

Training Data (X_train): rollno_scaled branch_CSE branch_ECE branch_MECH

	rollno_scaled	branch_CSE	branch_ECE	branch_MECH
3	1.000000	1.0	0.0	0.0
0	0.000000	1.0	0.0	0.0
2	0.666667	0.0	0.0	1.0

Testing Data (X_test): rollno_scaled branch_CSE branch_ECE branch_MECH

	rollno_scaled	branch_CSE	branch_ECE	branch_MECH
1	0.333333	0.0	1.0	0.0

Training Target (y_train):

3	1.000000
0	0.000000
2	0.666667

Name: rollno_scaled, dtype: float64

Testing Target (y_test):

1	0.333333

Name: rollno_scaled, dtype: float64

EXPERIMENT 3:

Demonstrate the following Similarity and Dissimilarity Measures using python a) Pearson's Correlation b) Cosine Similarity c) Jaccard Similarity d) Euclidean Distance e) Manhattan Distance

AIM: To Demonstrate the following Similarity and Dissimilarity Measures using python a) Pearson's Correlation b) Cosine Similarity c) Jaccard Similarity d) Euclidean Distance e) Manhattan Distance

Program:

```
import numpy as np
from scipy.stats import pearsonr # Changed 'pearson' to 'pearsonr'
from scipy.spatial.distance import cosine, euclidean, cityblock
def jaccord_similarity(set1,set2):
        intersection = len(set1.intersection(set2))
        union = len(set1) + len(set2) - intersection
        return intersection / union
data1=np.array([1,2,3,4,5])
data2=np.array([2,4,6,8,10])
set1=set([1,2,3])
set2=set([2,3,4,5])
pearson_corr= pearsonr(data1, data2)
print('Pearson Correlation:', pearson_corr)
cos_sim=1-cosine(data1,data2)
print('Cosine Similarity:', cos_sim)
jaccord_sim=jaccord_similarity(set1,set2)
print('Jaccord Similarity:', jaccord_sim)
euclidean_dist=euclidean(data1,data2)
print('Euclidean Distance:', euclidean_dist)
manhattan_dist=cityblock(data1,data2)
print('Manhattan Distance:', manhattan_dist)
```

OUTPUT:

```
Pearson Correlation: PearsonRResult(statistic=1.0, pvalue=0.0)
Cosine Similarity: 1.0
Jaccord Similarity: 0.4
Euclidean Distance: 7.416198487095663
Manhattan Distance: 15
```

Experiment 4

Build a model using linear regression algorithm on any dataset

AIM: Build a model using linear regression algorithm on any dataset

Data set:

House price prediction

index	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renet	street	city	statezip	country
0	2014-05-02 00:00:00	313000.0	3.0	1.5	1340	7912	1.5	0	0	3	1340	0	1955	2005	18810 Densmore Ave N	Shoreline	WA 98133	USA
1	2014-05-02 00:00:00	2384000.0	5.0	2.5	3650	9050	2.0	0	4	5	3370	280	1921	0	709 W Blaine St	Seattle	WA 98119	USA
2	2014-05-02 00:00:00	342000.0	3.0	2.0	1930	11947	1.0	0	0	4	1930	0	1966	0	26206-26214 143rd Ave SE	Kent	WA 98042	USA

Description:

Linear regression is a type of supervised machine learning algorithm that computes the linear relationship between the dependent variable

and one or more independent features by fitting a linear equation to observed data. When there is only one independent feature, it is known as Simple Linear Regression, and when there are more than one feature, it is known as Multiple Linear Regression.

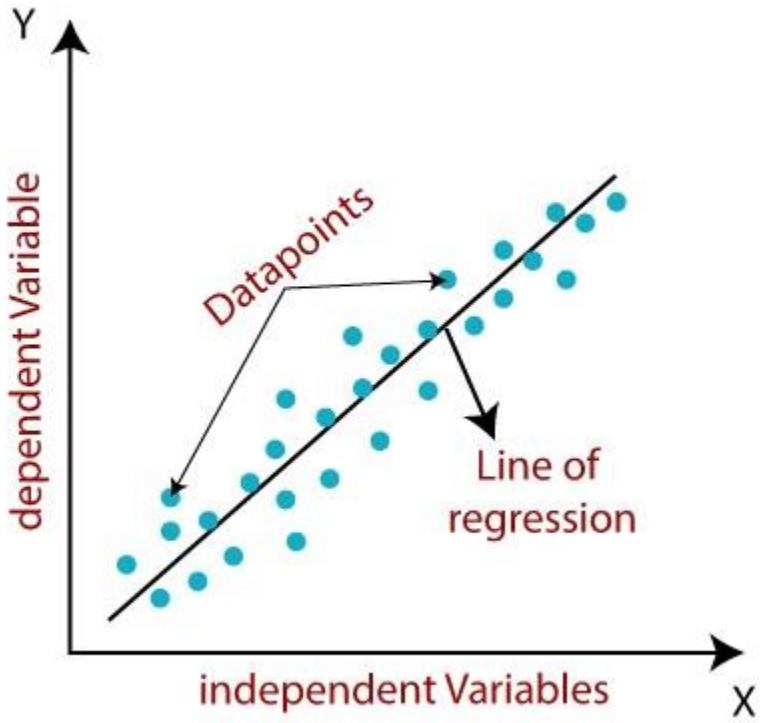
Simple Linear Regression

This is the simplest form of linear regression, and it involves only one independent variable and one dependent variable. The equation for simple linear regression is:

$y=\beta 0+\beta 1X$

where:

- Y is the dependent variable
- X is the independent variable
- β0 is the intercept
- β1 is the slope



PROGRAM:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import
mean_squared_error,r2_score

df=pd.read_csv('data.csv')
df.head(3)

x=df[['bedrooms','bathrooms','sqft_living','sqft_lot','floors','wat
erfront'
,'view','condition']]
y=df['price']

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,ran
dom_st ate=42)

model=LinearRegression()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)

mse=mean_squared_error(y_test,
y_pred)
r2=r2_score(y_test,y_pred)
print(f'Mean Squared Error:
{mse}') print(f'R-squared: {r2}')

new_data=[[3,2,1500,4000,1,0,0,3]]
predicted_price=model.predict(new_data)
print(f'Predicted  Price: {predicted_price[0]}')
```

EXPERIMENT 5

Build a classification model using Decision Tree algorithm on iris dataset

AIM: To Build a classification model using Decision Tree algorithm on iris dataset **DATASET:**

The Iris dataset is loaded using `sklearn.datasets.load_iris`

DESCRIPTION:

Dataset Loading:

- The Iris dataset is loaded using sklearn.datasets.load_iris.

Data Splitting:

- The dataset is split into training and testing subsets using train_test_split.

Model Training:

- A DecisionTreeClassifier is created and trained on the training data.

Evaluation:

- The model's predictions are compared against the true labels using metrics like accuracy and a classification report.

Tree Visualization:

- The tree's decision rules are displayed using export_text.

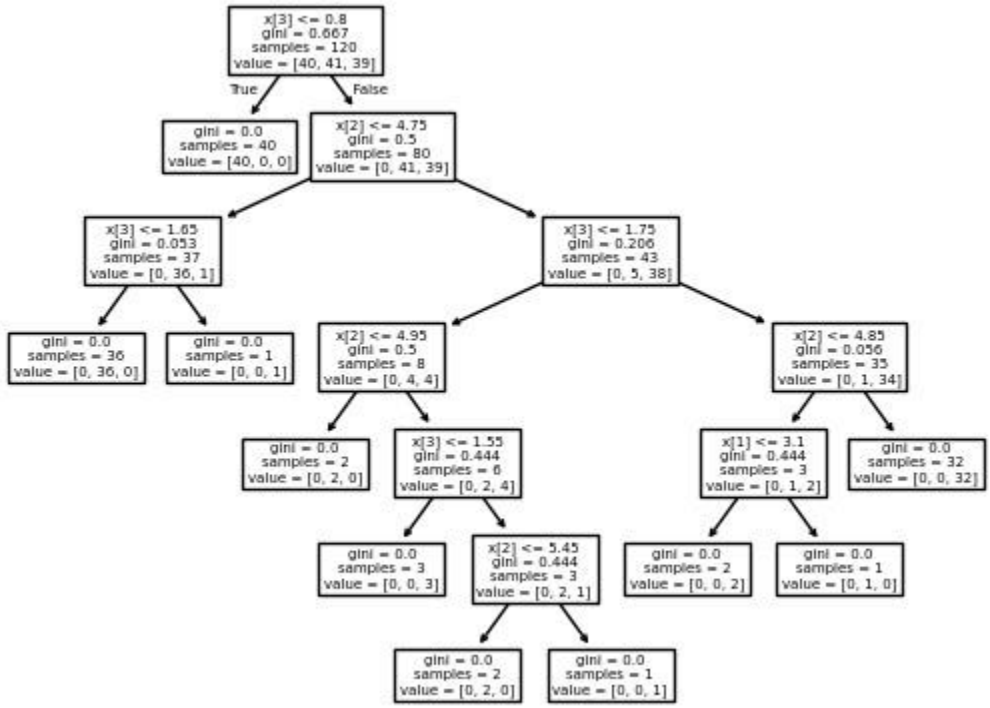
- A Decision Tree is a supervised learning algorithm used for classification and regression tasks. For building a classification model using the Iris dataset, the data is split into training and testing sets. A `DecisionTreeClassifier` is trained on the training set, and its performance is evaluated on the test set using accuracy and classification metrics. The model's decision-making process can be visualized through tree rules, showing how features are used to classify data points into target categories. Decision Trees are

interpretable, making them useful for understanding feature importance and decision paths.

PROGRAM

```
# Import necessary libraries from sklearn.datasets import
load_iris from sklearn.tree import DecisionTreeClassifier,
export_text from sklearn.model_selection import
train_test_split from sklearn.metrics import
accuracy_score, classification_report

# Load the Iris
dataset iris =
load_iris()
X, y = iris.data, iris.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Initialize the Decision Tree classifier
clf =
DecisionTreeClassifier(random_state=
42)

# Train the model
clf.fit(X_train, y_train)

# Make predictions
y_pred =
clf.predict(X_test)

# Evaluate the model accuracy =
accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy *
100:.2f}%")
```

```
print("\nClassification Report:\n", classification_report(y_test, y_pred))

classifier = DecisionTreeClassifier(criterion='gini')  # Changed 'creterion' to 'criterion' classifier.fit(X_train, y_train) y_pred = classifier.predict(X_test) print(classification_report(y_test, y_pred))

from sklearn import tree
tree.plot_tree(classifier)
```



EXPERMENT 6:

Apply Naïve Bayes Classification algorithm on any dataset

Aim: To Apply Naïve Bayes Classification algorithm on any

DATASET**:**

The Iris dataset is loaded using sklearn. datasets. load_iris

DESCRIPTION:

Naive Bayes Classifiers

The **Naïve Bayes Classifier** is a family of probabilistic machine learning algorithms based on applying **Bayes' Theorem** with the assumption of feature independence. Despite its simplicity, it performs well in many applications, especially for text classification and spam filtering.

Bayes' Theorem:

P(A∣B)=P(B∣A)⋅P(A)P(B)P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}P(A∣B)=P(B)P(B∣A)⋅P(A)

- P(A∣B)P(A|B)P(A∣B): Posterior probability (Probability of AAA given BBB).
- P(B∣A)P(B|A)P(B∣A): Likelihood (Probability of BBB given AAA).
- P(A)P(A)P(A): Prior probability (Probability of AAA).
  - P(B)P(B)P(B): Evidence (Probability of BBB).

Naïve Assumption:

Assumes all features are independent, which is rarely true but simplifies computations and works well in practice. Applications of Naive Bayes Classifier

- Spam Email Filtering: Classifies emails as spam or non-spam based on features.
- Text Classification: Used in sentiment analysis, document categorization, and topic classification.
- Medical Diagnosis: Helps in predicting the likelihood of a disease based on symptoms.
- Credit Scoring: Evaluates creditworthiness of individuals for loan approval.
- Weather Prediction: Classifies weather conditions based on various factors.

PROGRAM:

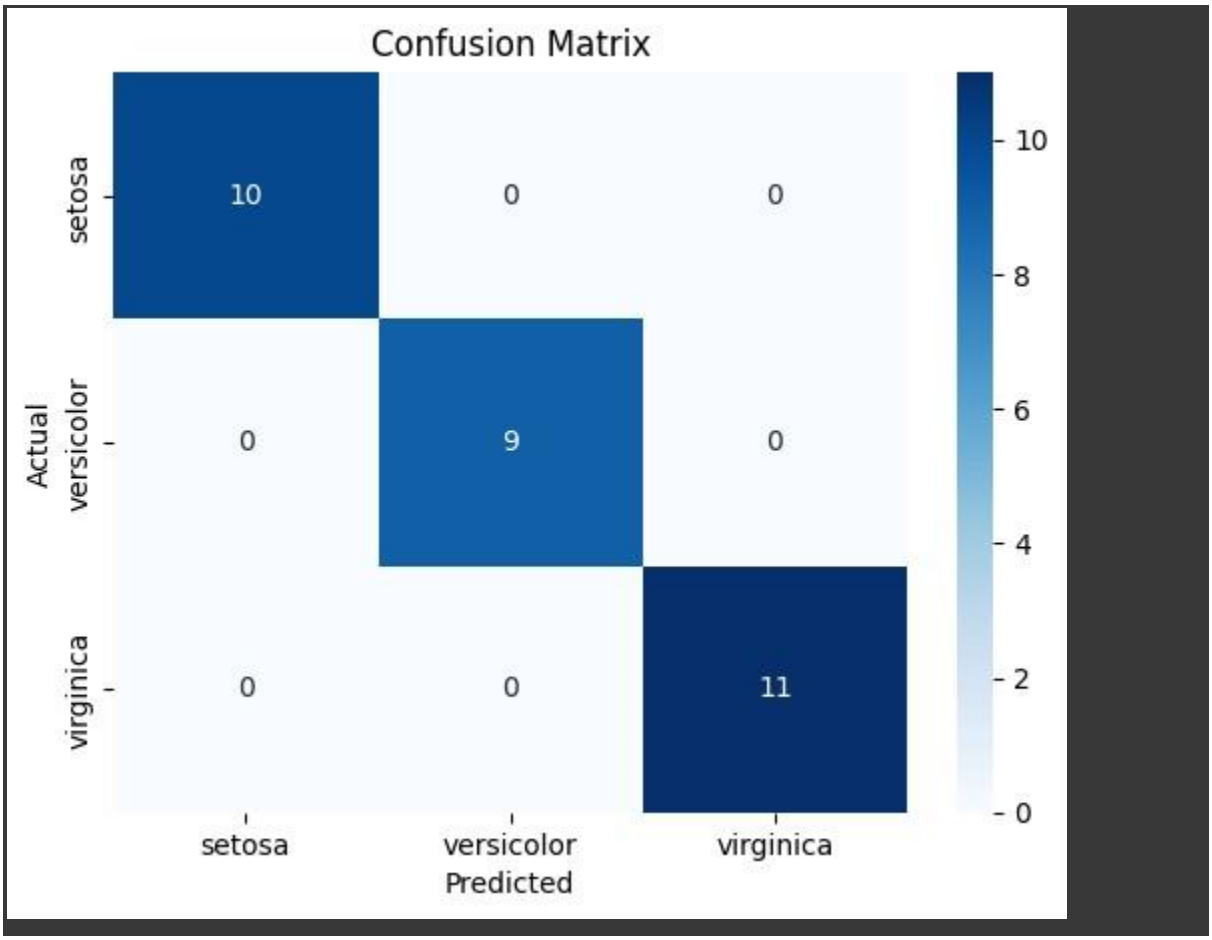
```
from sklearn.datasets import load_iris from
sklearn.model_selection import
train_test_split from sklearn.naive_bayes
import GaussianNB
from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix import pandas as
pd import seaborn as sns import matplotlib.pyplot as plt


# Load the IRIS dataset
iris =
load_iris
() X =
iris.data
y =
iris.targe
t


# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

```
# Initialize Gaussian Naive Bayes
classifier gnb = GaussianNB()

# Train the model
gnb.fit(X_train, y_train)


# Predict on the test set
y_pred =
gnb.predict(X_test)


# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Display classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Confusion Matrix cm =
confusion_matrix(y_test,
y_pred) sns.heatmap(cm,
annot=True, fmt='d',
cmap='Blues',
xticklabels=iris.target_names,
yticklabels=iris.target_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

OUTPUT:

```
Accuracy: 1.00
Classification Report:
              precision    recall f1-score   support
```

0	1.00	1.00	1.00	10
1	1.00	1.00	1.00	9
2	1.00	1.00	1.00	11
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30



EXPERIMENT 7:

Generate frequent itemsets using Apriori Algorithm in python and also generate association rules for any market basket data.

AIM: To Generate frequent itemsets using Apriori Algorithm in python and also generate association rules for any market basket data.

DESCRIPTION:

The Apriori Algorithm is a foundational data mining algorithm used to identify frequent itemsets and generate association rules in transactional datasets. It is widely applied in market basket analysis to find patterns like products frequently bought together**.**We apply an iterative approach or level-wise search where k-frequent itemsets are used to find k+1 itemsets.

Key Features:

1. **Frequent Itemsets**: The algorithm identifies groups of items (itemsets) that appear together frequently in transactions.
2. **Association Rules**: These are rules derived from the frequent itemsets, showing relationships like "If item A is purchased, then item B is likely to be purchased."

Working Principle:

The algorithm uses an iterative, level-wise approach based on the concept of:

- **Support**: The proportion of transactions containing a specific itemset.
- **Confidence**: The likelihood that a transaction containing item A also contains item B.
- **Lift**: Measures the strength of association, comparing confidence to the baseline probability of item B.

Steps:

1. **Generate Candidate Itemsets**: Create itemsets of size 1, then iteratively larger ones.
2. **Prune Infrequent Itemsets**: Remove itemsets that do not meet the minimum support threshold.
3. **Find Frequent Itemsets**: Repeat until no more frequent itemsets are found.
4. **Generate Association Rules**: From the frequent itemsets, derive rules that meet minimum confidence.

Applications:

- **Market Basket Analysis**: Discover product bundling opportunities.
- **Healthcare**: Identify correlations in patient symptoms or treatments.
- **Web Usage Mining**: Understand user navigation patterns on websites.

Limitations:

- Inefficient for large datasets due to exponential growth of candidate itemsets.
- Sensitive to user-defined thresholds for support and confidence.

The Apriori algorithm laid the foundation for more advanced and scalable algorithms like FP-Growth.

PROGRAM:

```
import pandas as pd from
mlxtend.frequent_patterns import apriori from
mlxtend.frequent_patterns import
association_rules from mlxtend.preprocessing
import TransactionEncoder

dataset =[['milk','onion','bread','beans','eggs','yogurt'],
          ['fish','onion','bread','beans','eggs','yogurt'],
          ['milk','apples','beans','eggs'],
          ['milk','sugar','tea','beans','yogurt'],
          ['tea','onion','beans','ice','eggs']]

tr=TransactionEncoder()
tr_arr=tr.fit(dataset).transform(dataset)
df=pd.DataFrame(tr_arr,columns=tr.columns_)
print(df)
```

OUTPUT:

i	apples	beans	bread	eggs	fish	ice	milk	onion	sugar	tea	yogurt
0	false	true	true	true	false	false	true	true	false	false	true
1	false	true	true	true	true	false	false	true	false	false	true
2	true	true	false	true	false	false	true	false	false	false	false
3	false	true	false	false	false	false	true	false	true	true	true
4	false	true	false	true	false	true	false	true	false	true	false

EXPERIMENT 8:

Apply K- Means clustering algorithm on any dataset.

AIM: TO Apply K- Means clustering algorithm on IRIS dataset.

DATASET:

IRIS dataset from sklearn preprocessing datasets

DESCRIPTION:

K-Means Clustering is an Unsupervised Machine Learning algorithm, which groups the unlabeled dataset into different clusters.

K means clustering, assigns data points to one of the K clusters depending on their distance from the center of the clusters. It starts by randomly assigning the clusters centroid in the space. Then each data point assign to one of the cluster based on its distance from centroid of the cluster. After assigning each point to one of the cluster, new cluster centroids are assigned. This process runs iteratively until it finds good cluster. In the analysis we assume that number of cluster is given in advanced and we have to put points in one of the group.

The algorithm works as follows:

1. First, we randomly initialize k points, called means or cluster centroids.
2. We categorize each item to its closest mean, and we update the mean's coordinates, which are the averages of the items categorized in that cluster so far.
3. We repeat the process for a given number of iterations and at the end, we have our clusters.

PROGRAM:

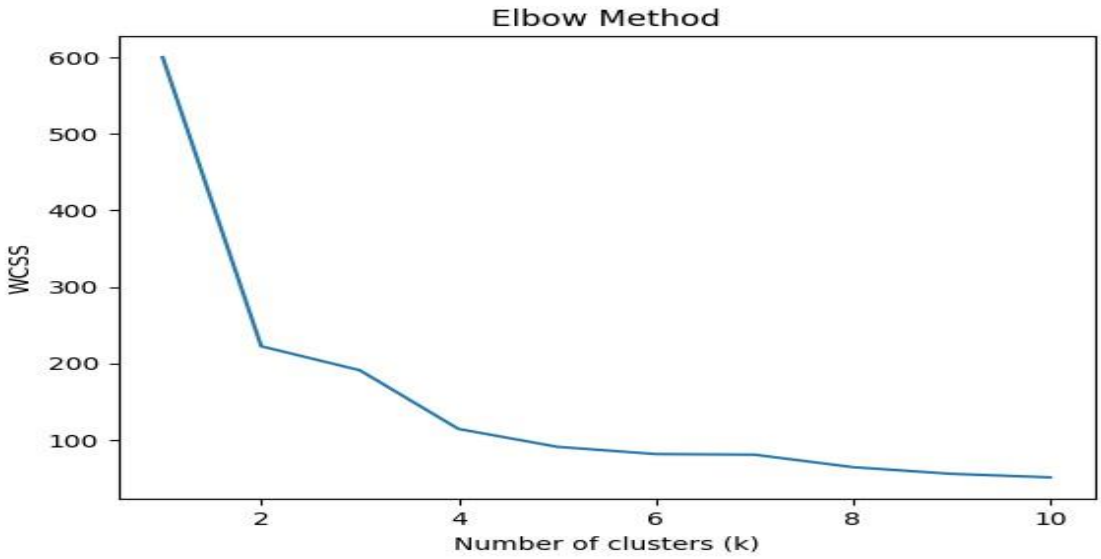
```
import pandas as pd from sklearn.cluster
import KMeans from sklearn.preprocessing
import StandardScaler from
sklearn.datasets import load_iris import
matplotlib.pyplot as plt
# Load the Iris dataset iris = load_iris() data =
pd.DataFrame(data=iris.data, columns=iris.feature_names)

# Select features for clustering features = ['sepal length
(cm)', 'sepal width (cm)', 'petal length (cm)',
'petal width (cm)']
X = data[features]

# Standardize the
data scaler =
StandardScaler()
X_scaled = scaler.fit_transform(X)

# Determine the optimal number of clusters (k) using the Elbow
method wcss = [] for k in range(1, 11): kmeans =
KMeans(n_clusters=k, init='k-means++', random_state=42)
kmeans.fit(X_scaled) wcss.append(kmeans.inertia_)

plt.plot(range(1, 11),
wcss) plt.title('Elbow
Method')
plt.xlabel('Number of
clusters (k)')
plt.ylabel('WCSS')
plt.show()
```

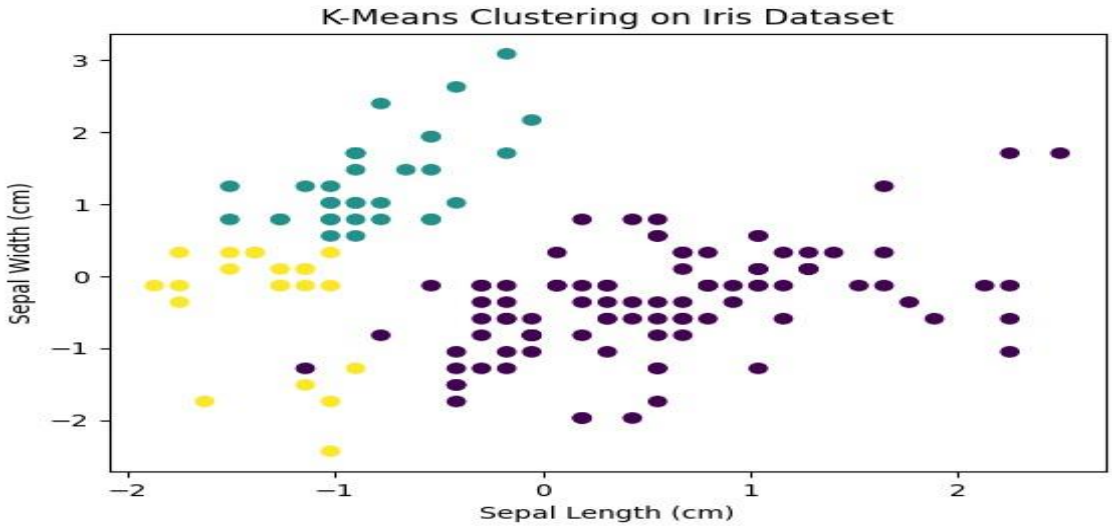
Apply K-Means clustering with k=3 (based on the elbow method) kmeans = KMeans(n_clusters=3, init='k-means++', random_state=42) kmeans.fit(X_scaled)

Assign cluster labels to data points
data['cluster'] = kmeans.labels_

# Visualize the clusters plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=data['cluster'], cmap='viridis') plt.title('K-Means Clustering on Iris Dataset')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.show()

OUTPUT:





EXPERIMENT 9:

Apply Hierarchical Clustering algorithm on any dataset.

AIM : To Apply Hierarchical Clustering algorithm on iris dataset.

DESCRIPTION:

A Hierarchical clustering method works via grouping data into a tree of clusters. Hierarchical clustering begins by treating every data point as a separate cluster. Then, it repeatedly executes the subsequent steps:

1. Identify the 2 clusters which can be closest together, and
2. Merge the 2 maximum comparable clusters. We need to continue these steps until all the clusters are merged together.

In Hierarchical Clustering, the aim is to produce a hierarchical series of nested clusters. A diagram called **Dendrogram**

Dendrogram

A Dendrogram is a tree-like diagram used to visualize the relationship among clusters. More the distance of the vertical lines in the dendrogram, the more the distance between those clusters. The key to interpreting a dendrogram is to concentrate on the height at which any two objects are joined together.

Types of Hierarchical Clustering

Basically, there are two types of hierarchical Clustering:

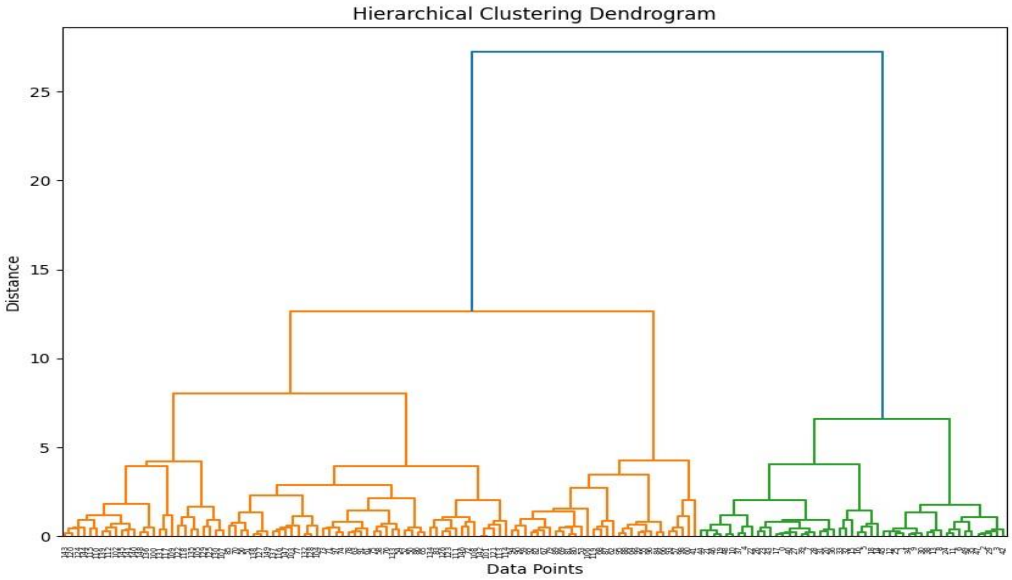
1. Agglomerative Clustering
2. Divisive clustering

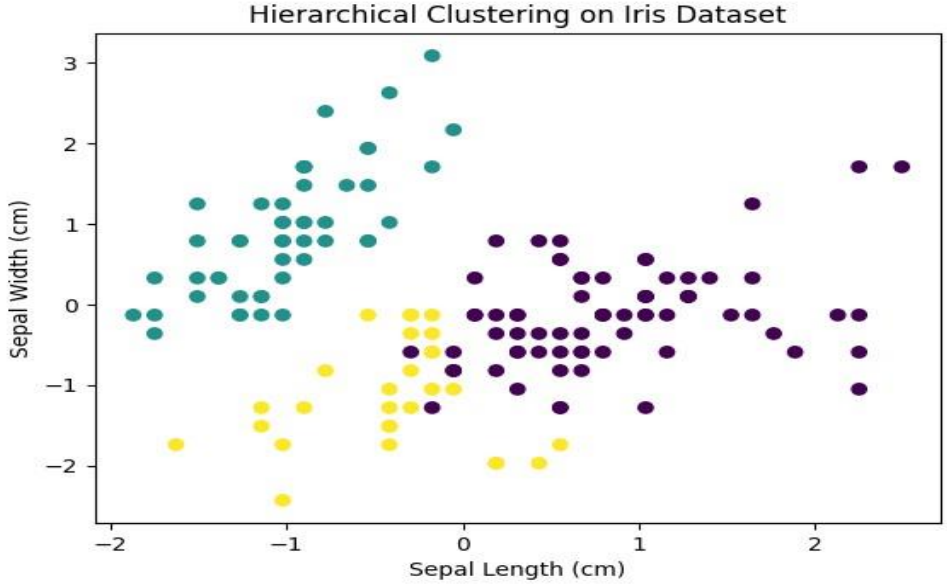
PROGRAM:

```
import pandas as pd from sklearn.cluster import
AgglomerativeClustering from
sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram,
linkage import matplotlib.pyplot as plt

# Load your dataset (replace with your actual data source)
# Here, we'll use the Iris dataset for demonstration
from sklearn.datasets import load_iris iris = load_iris() data
= pd.DataFrame(data=iris.data,
columns=iris.feature_names)

# Select features for clustering features = ['sepal length
(cm)', 'sepal width (cm)', 'petal length (cm)',
'petal width (cm)']
X = data[features]

# Standardize the data (optional, but often
recommended) scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Perform hierarchical clustering with 'ward'
linkage # Perform hierarchical clustering with
'ward' linkage
hc = AgglomerativeClustering(n_clusters=3, linkage='ward') #
Remove affinity='euclidean' hc.fit(X_scaled)

# Get cluster
labels labels =
hc.labels_

# Visualize the dendrogram linked = linkage(X_scaled, 'ward')
# Use 'ward' linkage for minimizing variance within clusters
```

```
plt.figure(figsize=(10, 7)) dendrogram(linked, orientation='top',
distance_sort='descending', show_leaf_counts=True)
plt.title("Hierarchical Clustering Dendrogram") plt.xlabel("Data
Points") plt.ylabel("Distance") plt.show()

# Visualize the clusters (optional) plt.scatter(X_scaled[:, 0],
X_scaled[:, 1], c=labels, cmap='viridis') plt.title("Hierarchical
Clustering on Iris Dataset") plt.xlabel('Sepal Length (cm)')

plt.ylabel('Sepal Width (cm)')
plt.show()
```

OUTPUT:





EXPERIMENT 10:

Apply DBSCAN clustering algorithm on any dataset.

AIM: To .Apply DBSCAN clustering algorithm on iris dataset.

DESCRIPTION:

Density-Based Spatial Clustering Of Applications With Noise (DBSCAN)

Clusters are dense regions in the data space, separated by regions of the lower density of points. The DBSCAN algorithm is based on this intuitive notion of “clusters” and “noise”. The key idea is that for each point of a cluster, the neighborhood of a given radius has to contain at least a minimum number of points.

Parameters Required For DBSCAN Algorithm

1. eps: It defines the neighborhood around a data point i.e. if the distance between two points is lower or equal to ‘eps’ then they are considered neighbors. If the eps value is chosen too small then a large part of the data will be considered as an outlier. If it is chosen very large then the clusters will merge and the majority of the data points will be in the same clusters. One way to find the eps value is based on the k-distance graph.
2. MinPts: Minimum number of neighbors (data points) within eps radius. The larger the dataset, the larger value of MinPts must be chosen. As a general rule, the minimum MinPts can be derived from the number of dimensions D in the dataset as, MinPts >= D+1. The minimum value of MinPts must be chosen at least 3.

In this algorithm, we have 3 types of data points.

Core Point: A point is a core point if it has more than MinPts points within eps.
Border Point: A point which has fewer than MinPts within eps but it is in the neighborhood of a core point.
Noise or outlier: A point which is not a core point or border point.

PROGRAM:

```
import pandas as pd from sklearn.cluster
import DBSCAN from
sklearn.preprocessing import
StandardScaler from sklearn.datasets
import load_iris import matplotlib.pyplot as
plt

# Load the Iris dataset iris = load_iris() data =
pd.DataFrame(data=iris.data, columns=iris.feature_names)

# Select features for clustering features = ['sepal length
(cm)', 'sepal width (cm)', 'petal length (cm)',
'petal width (cm)']
X = data[features]

# Standardize the
data scaler =
StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply DBSCAN clustering dbscan = DBSCAN(eps=0.5,
min_samples=5) # Adjust eps and min_samples as
needed
labels = dbscan.fit_predict(X_scaled)

# Visualize the clusters plt.scatter(X_scaled[:, 0],
X_scaled[:, 1], c=labels, cmap='viridis')
```

```
plt.title('DBSCAN Clustering on Iris Dataset')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal  Width (cm)')
plt.show()
```

OUTPUT: