Parul University

FACULTY OF ENGINEERING AND TECHNOLOGY
BACHELOR OF TECHNOLOGY

Machine Learning (203105403)

VII SEMESTER

Computer Science and Engineering Department

Laboratory Manual

Session 2024-25

CERTIFICATE

This is to certify that Mr. **Utkarsh Barde** with Enrolment no. **210305105194** has successfully completed his laboratory Experiments in the **Machine Learning Laboratory (203105403)** from the Department of **COMPUTER SCIENCE AND ENGINEERING** during the academic year 2024-2025

योगः कर्मसु कौशलम्

PARUL UNIVERSITY

Date of Submission: Staff In Charge:

Head Of Department:

TABLE OF CONTENT

Sr. No	Experiment Title	Page No		Date of Start	Date of Completion	Sign.
		From	To			
1.	Write a program to demonstrate the working of the decision tree-based ID3 algorithm.					
2.	Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.					
3.	Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering a few test data sets.					
4.	Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task.					
5.	Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.					
6.	Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm.					
7.	Write a program to implement the K-Nearest Neighbour algorithm to classify the iris data set.					
8.	Implement linear regression and logistic regression.					
9.	Compare the various supervised learning algorithm by using appropriate dataset.					
10.	Compare the various Unsupervised learning algorithm by using the appropriate datasets.					

Practical - 1

Aim: Write a program to demonstrate the working of the decision tree based ID3 algorithm.

Theory:

❖ **ID3 Algorithm:**

The ID3 algorithm uses entropy and information gain as measures to make decisions about feature selection and node splitting. It aims to create a decision tree that maximizes the information gain at each step, leading to a tree that can accurately classify new examples.

❖ **Working:**

1. **Input:** The algorithm takes as input a dataset with labelled examples. Each example consists of a set of features and a corresponding class label.
2. **Select the root node:** The first step is to select the root node of the decision tree. This is typically done by choosing the feature that provides the most information gain.
3. **Calculate information gain:** For each feature, the algorithm calculates the information gain. Information gain measures how much the entropy of the dataset is reduced by splitting it based on a particular feature. The feature with the highest information gain is chosen as the root node.
4. **Split the dataset:** The dataset is split into subsets based on the selected feature at the root node. Each subset contains examples that have the same value for the chosen feature.
5. **Repeat the process:** The algorithm recursively repeats the above steps for each subset created in the previous step. It calculates the information gain for each remaining feature in the subset and chooses the one with the highest information gain as the next node in the tree. This process continues until a stopping criterion is reached.
6. **Stopping criterion:** The stopping criterion could be reaching a maximum depth for the tree, having a minimum number of examples at a node, or when all examples in a subset belong to the same class.
7. **Assign class labels:** Once the tree is built, class labels are assigned to the leaf nodes. This is done by taking the majority class of the examples in each leaf node.
8. **Predicting with the tree:** To predict the class label for a new example, it traverses the decision tree based on the values of the features until it reaches a leaf node. The class label associated with that leaf node is then assigned as the predicted class label for the example.

❖ **Dataset taken:** IRIS Dataset.

- This data sets consists of 3 different types of irises' (Setosa, Versicolour, and Virginica) petal and sepal length, stored in a 150x4 numpy.ndarray.
- The rows being the samples and the columns being: Sepal Length, Sepal Width, Petal Length and Petal Width.
- No. of Rows: 150
- No. of Columns: 4

❖ **Procedure:**

#Step-1: Import python libraries.import numpy as np

import seaborn as sns import matplotlib.pyplot as plt from

sklearn import datasets

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier from sklearn

import metrics

#Step-2: Import IRIS Dataset dataset

= datasets.load_iris() X

= dataset.data y = dataset.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

#Step-3: Load Decision tree classifier into clf variable.

clf = DecisionTreeClassifier() clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

accuracy = metrics.accuracy_score(y_test, y_pred)

#Step-4: Plot the Confusion Matrix. confusion_matrix =

metrics.confusion_matrix(y_test, y_pred)

print("Accuracy:", accuracy) print("Confusion Matrix:") labels

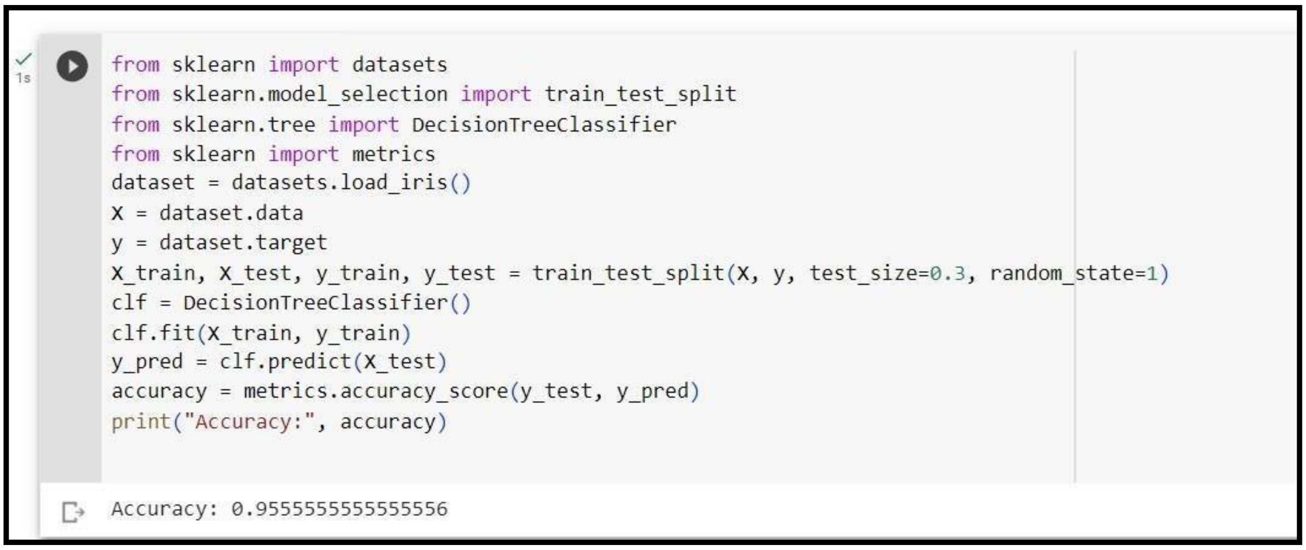
= dataset.target_names

sns.heatmap(confusion_matrix, annot=True, fmt="d", xticklabels=labels, yticklabels=labels,

cmap="Blues", cbar=False) plt.xlabel("Predicted") plt.ylabel("True") plt.title("Confusion

Matrix") plt.show()

❖ Output:



```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
dataset = datasets.load_iris()
X = dataset.data
y = dataset.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9555555555555556
```

Figure _ 1.1: Printing the Accuracy of the dataset

```
Accuracy: 0.9555555555555556
Confusion Matrix:
```

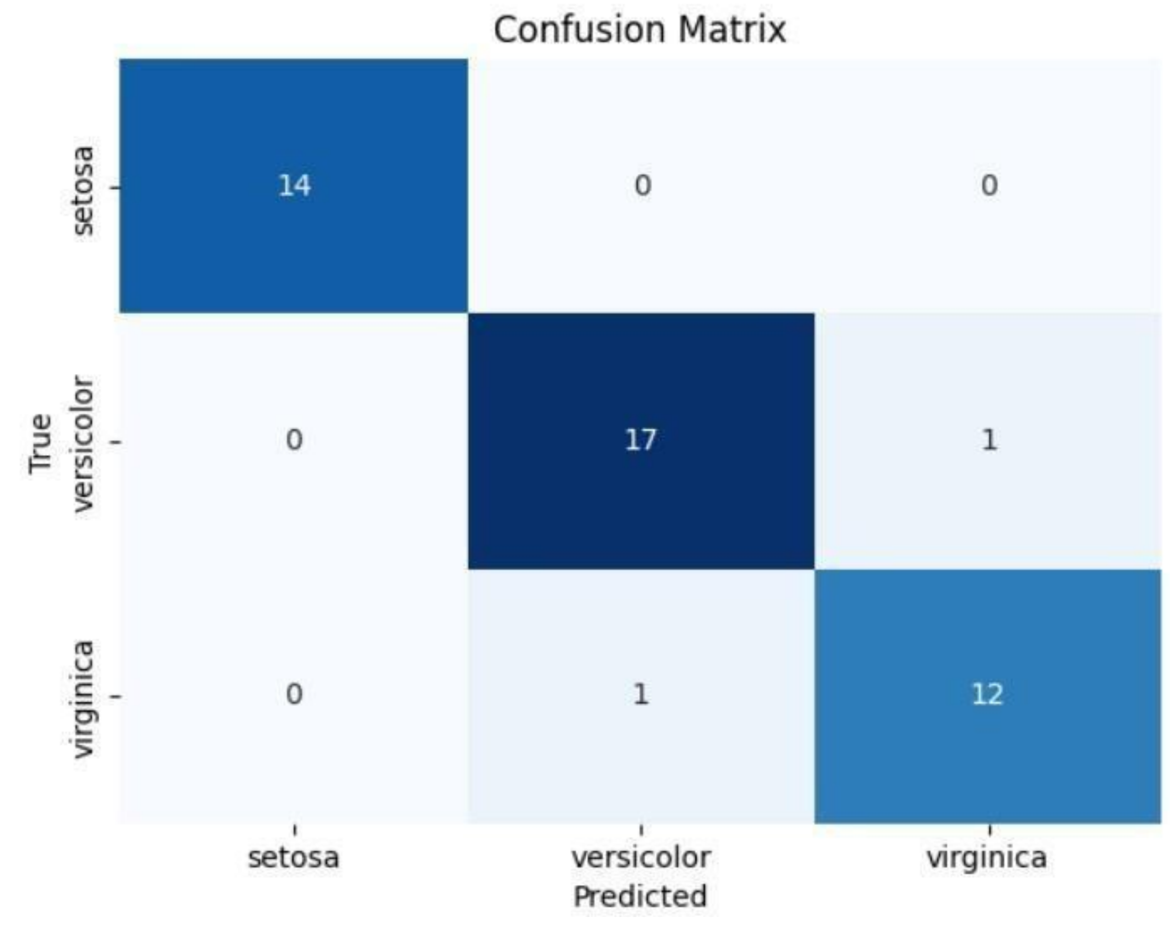


Figure – 1.2: Print the Confusion Matrix

Practical – 2

Aim: Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

Theory:

❖ Artificial Neural Networks (ANNs):

- The term "Artificial Neural Network" is derived from Biological neural networks that develop the structure of a human brain. Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks. These neurons are known as nodes.
- Artificial Neural Networks are computational models inspired by the structure and functioning of biological neural networks in the human brain. ANNs consist of interconnected nodes, called neurons, organized into layers.
- The neurons in one layer are connected to neurons in the adjacent layers. The first layer is the input layer, the last layer is the output layer, and the intermediate layers are called hidden layers. ANNs are capable of learning from data and making predictions or decisions based on the learned patterns.

❖ Figure:

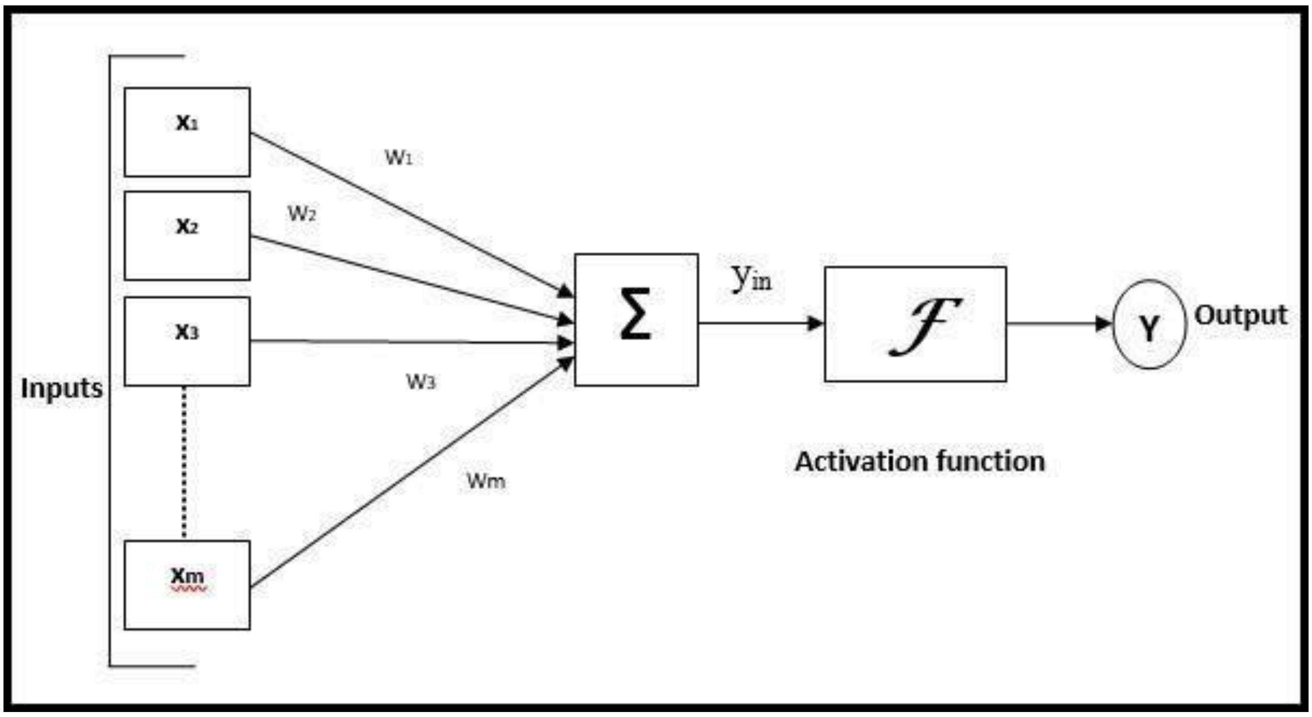


Figure _ 2.1: Artificial Neural Network

The main idea:

1. Each neuron processes a bit of information and passes it to its children.
2. Overall the network processes raw information into general concepts.

❖ Backpropagation Algorithm:

- The backpropagation algorithm is a widely used method for training feedforward neural networks. It allows the network to learn from labelled training data by adjusting the weights and biases of the neurons based on the prediction errors.

The key steps in the backpropagation algorithm are as follows:

1. Initialization: Randomly initialize the weights and biases of the neural network.
2. Forward Propagation: Pass an input through the network, calculating the weighted sum of inputs and applying an activation function to produce an output. This process is repeated for each layer, propagating the input forward through the network until the output is obtained.
3. Error Calculation: Compute the difference between the predicted output and the actual output (the error).
4. Backward Propagation: Propagate the error backward through the network, layer by layer. For each layer, calculate the gradient of the error with respect to the weights and biases using the chain rule of calculus. This determines how much each weight and bias contributed to the overall error.
5. Weight and Bias Update: Adjust the weights and biases of the neurons in each layer using the gradients computed in the previous step. The adjustment is performed by subtracting a fraction of the gradients multiplied by a learning rate, which controls the step size during optimization.
6. Repeat: Repeat steps 2 to 5 for a specified number of iterations or until the network's performance reaches a satisfactory level.

- By iteratively updating the weights and biases using the backpropagation algorithm, the neural network gradually improves its ability to make accurate predictions or classifications based on the training data.

- The backpropagation algorithm is an efficient way to train neural networks, but it requires a large amount of labelled training data and may suffer from issues such as overfitting or getting stuck in local minima. Regularization techniques, learning rate schedules, and other optimization strategies are often employed to address these challenges.

❖ **Dataset taken:** Telco Customer Churn Dataset.

- The data set includes information about customers who left within the last month, Services that each customer has signed up for, Customer account information, Demographic info about customers.
- No. of Rows: 7043
- No. of Columns: 47

Procedure:

❖ Task-1: Import Libraries, preprocess the training dataset and selection of required features.

# Step-1: Import Library import pandas

as pd

#Step-2: Import the files to Google Colab.

url = 'https://raw.githubusercontent.com/rc-dbe/bigdatacertification/master/dataset/churn_trasnsformed_new.csv'

df_csv = pd.read_csv(url, sep=',',) df_csv.head()

#Step-3: Remove "Unnamed:O" Coloumn df

= df_csv.drop("Unnamed: 0", axis=1) df.head()

df.info()

#Step-4: Import MinMax Scaler from

sklearn.preprocessing import MinMaxScaler

# initialize min-max scaler mm_scaler

= MinMaxScaler() column_names =

df.columns.tolist()

column_names.remove('Churn')

df[column_names] = mm_scaler.fit_transform(df[column_names])

df.sort_index(inplace=True) df.head()

#Step-5: Selecting the Feature, by remove the unused feature

feature = ['Churn'] train_feature = df.drop(feature, axis=1)

train_target = df["Churn"] train_feature.head(5) **#Step-6: Split and**

Show Training Data. from sklearn.model_selection import

train_test_split, cross_val_score

X_train, X_test, y_train, y_test = train_test_split(train_feature ,train_target, shuffle = True, test_size=0.3, random_state=1)

X_train.head()

❖ **Task-2: To train the ANN Model.We will use the MLPClassifier from Scikit Learn Library.**

#Step-1: Import Library
from sklearn.neural_network import MLPClassifier

#Step-2: Fitting Model mlp = MLPClassifier(hidden_layer_sizes=(5), activation = 'relu', solver = 'adam',max_iter=
10000, verbose = True) mlp =
mlp.fit(X_train,y_train) #
Prediction to Test Dataset
y_predmlp = mlp.predict(X_test)

#Step-3: Print the Number of layers and Iterations.
print('Number of Layer =', mlp.n_layers_) print('Number
of Iteration =', mlp.n_iter_)
print('Current loss computed with the loss function =', mlp.loss_)

#Step-3: Import the metrics class.
from sklearn import metrics

#Step-4: Storing Confussion Matrix
cnf_matrixmlp = metrics.confusion_matrix(y_test, y_predmlp) cnf_matrixmlp

#Step-5: Show the Accuracy, Precision, Recall, F1, etc.
acc_mlp = metrics.accuracy_score(y_test, y_predmlp)
prec_mlp = metrics.precision_score(y_test, y_predmlp)
rec_mlp = metrics.recall_score(y_test, y_predmlp) f1_mlp
= metrics.f1_score(y_test, y_predmlp)
kappa_mlp = metrics.cohen_kappa_score(y_test, y_predmlp)

❖ **Task-3: Create the confusion matrix using seaborn and matplotlib library.**

Code:

#Step-1: Import seaborn library.

import seaborn as sns import

matplotlib.pyplot as plt **#Step-2:**

Compute the confusion matrix

cnf_matrix =

confusion_matrix(y_test, y_predmlp)

#Step-3: Create a heatmap of the confusion matrix

plt.figure(figsize=(8, 6)) sns.heatmap(cnf_matrix, annot=True, fmt='d',

cmap='Blues', cbar=False)

#Step-4: Set labels, title, and axis ticks

plt.xlabel('Predicted Label') plt.ylabel('True Label')

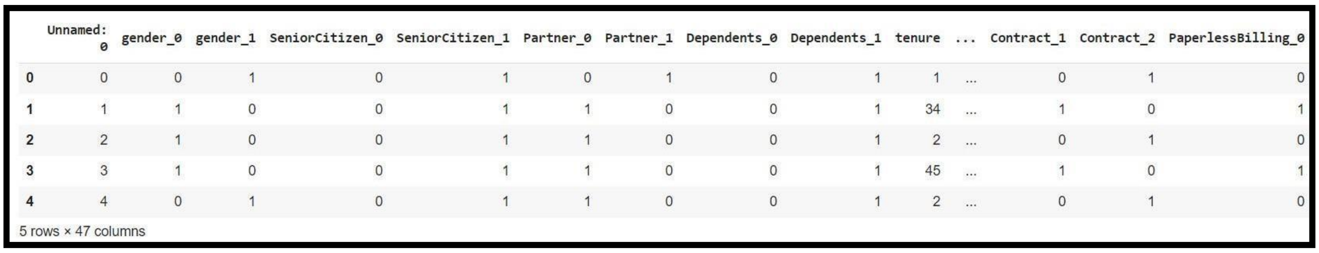
plt.xticks(ticks=[0, 1], labels=['No Churn', 'Churn'])

plt.yticks(ticks=[0, 1], labels=['No Churn', 'Churn'])

plt.title('Confusion Matrix')

#Step-5: Display the plot plt.show()

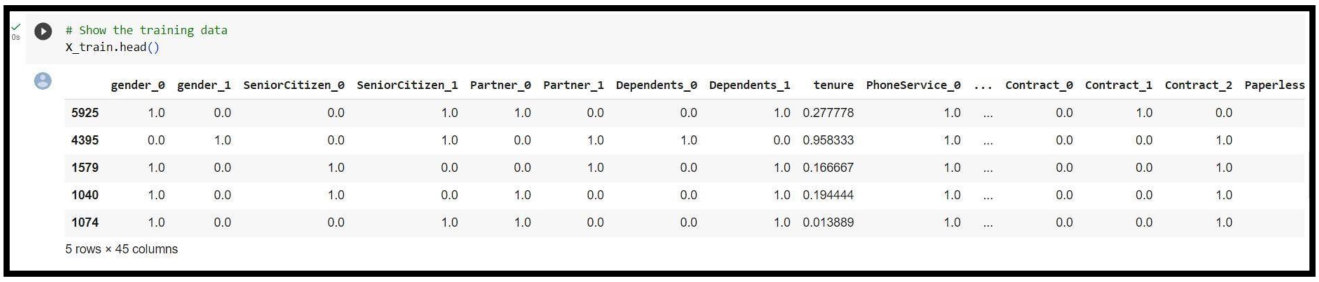
❖ Output:



	Unnamed: 0	gender_0	gender_1	SeniorCitizen_0	SeniorCitizen_1	Partner_0	Partner_1	Dependents_0	Dependents_1	tenure	...	Contract_1	Contract_2	PaperlessBilling_0
0	0	0	1	0	1	0	1	0	1	1	...	0	1	0
1	1	1	0	0	1	1	0	0	1	34	...	1	0	1
2	2	1	0	0	1	1	0	0	1	2	...	0	1	0
3	3	1	0	0	1	1	0	0	1	45	...	1	0	1
4	4	0	1	0	1	1	0	0	1	2	...	0	1	0

5 rows × 47 columns

Figure _ 2.2: Telco Customer Churn Dataset.

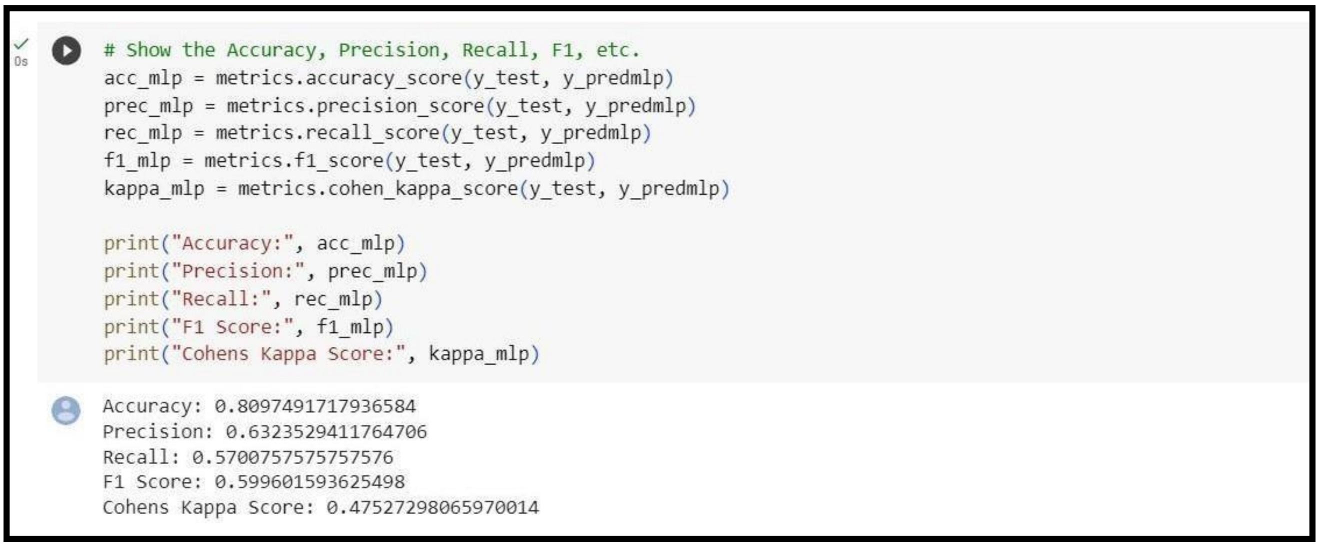


```
# Show the training data
X_train.head()
```

	gender_0	gender_1	SeniorCitizen_0	SeniorCitizen_1	Partner_0	Partner_1	Dependents_0	Dependents_1	tenure	PhoneService_0	...	Contract_0	Contract_1	Contract_2	Paperless
5925	1.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	0.277778	1.0	...	0.0	1.0	0.0	
4395	0.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	0.958333	1.0	...	0.0	0.0	1.0	
1579	1.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.166667	1.0	...	0.0	0.0	1.0	
1040	1.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.194444	1.0	...	0.0	0.0	1.0	
1074	1.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	0.013889	1.0	...	0.0	0.0	1.0	

5 rows × 45 columns

Figure – 2.3: Training Features of Data



```
# Show the Accuracy, Precision, Recall, F1, etc.
acc_mlp = metrics.accuracy_score(y_test, y_predmlp)
prec_mlp = metrics.precision_score(y_test, y_predmlp)
rec_mlp = metrics.recall_score(y_test, y_predmlp)
f1_mlp = metrics.f1_score(y_test, y_predmlp)
kappa_mlp = metrics.cohen_kappa_score(y_test, y_predmlp)

print("Accuracy:", acc_mlp)
print("Precision:", prec_mlp)
print("Recall:", rec_mlp)
print("F1 Score:", f1_mlp)
print("Cohens Kappa Score:", kappa_mlp)
```

```
Accuracy: 0.8097491717936584
Precision: 0.6323529411764706
Recall: 0.5700757575757576
F1 Score: 0.599601593625498
Cohens Kappa Score: 0.47527298065970014
```

Figure _ 2.4: Accuracy, Precision, Recall, F1Score

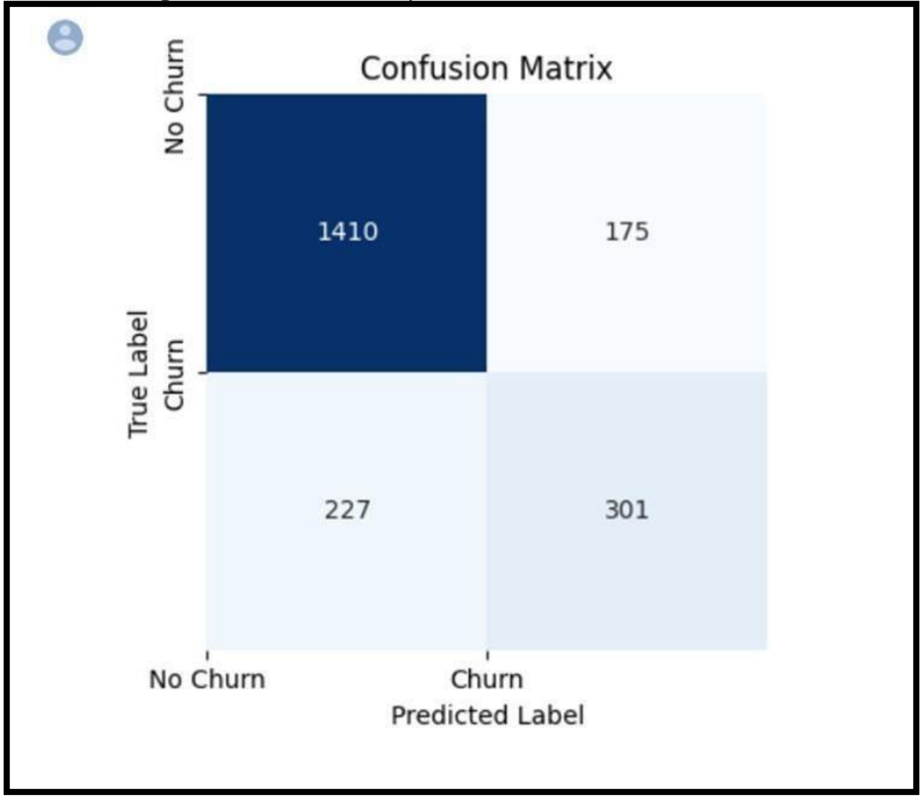


Figure _ 2.5: Confusion Matrix

Practical - 3

Aim: Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

Theory:

❖ What is Naïve Bayesian classifier?

- The Naive Bayes classifier is a simple yet powerful probabilistic machine learning algorithm that is commonly used for classification tasks. It is based on Bayes' theorem and assumes that the features are conditionally independent given the class label. This assumption is known as the "naive" assumption, which simplifies the calculation of probabilities.

- In machine learning, Naïve Bayes classification is a straightforward and powerful algorithm for the classification task. Naïve Bayes classification is based on applying Bayes' theorem with strong independence assumption between the features. Naïve Bayes classification produces good results when we use it for textual data analysis such as Natural Language Processing.

- Naïve Bayes models are also known as simple Bayes or independent Bayes. All these names refer to the application of Bayes' theorem in the classifier's decision rule. Naïve Bayes classifier applies the Bayes' theorem in practice. This classifier brings the power of Bayes' theorem to machine learning.

- The Naive Bayes classifier is built on the foundation of Bayes' theorem, which relates conditional probabilities.

❖ Given a feature vector X and a class label y, Bayes' theorem states:

$$P(H_{ypothesis}|E_{vidence}) = \frac{P(H) \times P(E|H)}{P(E)}$$

Bayes' Theorem

Figure _ 3.1: Baye's Theorem

❖ Naive Bayes algorithm calculations:

- Naïve Bayes Classifier uses the Bayes' theorem to predict membership probabilities for each class such as the probability that given record or data point belongs to a particular class. The class with the highest probability is considered as the most likely class. This is also known as the Maximum -A Posteriori (MAP). The MAP for a hypothesis with 2 events A and B is MAP (A). MAP (A) = max (P (A | B))

= max (P (B | A) * P (A))/P (B)

= max (P (B | A) * P (A))

- Here, P (B) is evidence probability. It is used to normalize the result. It remains the same. So, removing it would not affect the result. Naïve Bayes Classifier assumes that all the features are unrelated to each other. Presence or absence of feature does not affect the other features.

❖ **The Naive Bayes classifier works as follows:**

1. **Training:** Given a labelled training dataset, the classifier calculates the prior probability P(y) for each class in the dataset. It also estimates the likelihood probability P(X|y) for each feature given each class. This is done by assuming conditional independence between the features.
2. **Prediction:** When a new unlabelled instance is presented, the
3. classifier calculates the posterior probability P(y|X) for each class using Bayes' theorem. It then assigns the class label with the highest posterior probability as the predicted class for that instance.
4. **Handling Continuous Features:** For continuous features, the Naive Bayes classifier typically assumes a probability distribution, often Gaussian (hence called Gaussian Naive Bayes), to estimate the likelihood probability.
5. **Laplace Smoothing:** To avoid zero probabilities when a feature value in the testing data was not observed in the training data, Laplace smoothing (also known as additive smoothing) is often applied. It adds a small constant to numerator and adjusts the denominator accordingly.
6. **Decision Rule:** In some cases, the Naive Bayes classifier can be used for decision making by considering the posterior probabilities. For example, in binary classification, if P(y=1|X) > P(y=0|X), the instance is assigned to class 1; otherwise, it is assigned to class 0.

❖ **Types of Naive Bayes algorithm**

- There are 3 types of Naïve Bayes algorithm. The 3 types are listed below:-

1.) Gaussian Naïve Bayes algorithm:

- When we have continuous attribute values, we made an assumption that the values associated with each class are distributed according to Gaussian or Normal distribution. For example, suppose the training data contains a continuous attribute x. We first segment the data by the class, and then compute the mean and variance of x in each class:

$$p(x_i|y_j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{-\frac{(x_i-\mu_j)^2}{2\sigma_j^2}}$$

2.) Multinomial Naïve Bayes algorithm:

- With a Multinomial Naïve Bayes model, samples (feature vectors) represent the frequencies with which certain events have been generated by a multinomial (p1, . . . ,pn) where pi is the probability that event i occurs. Multinomial Naïve Bayes algorithm is preferred to use on data that is multinomially distributed. It is one of the standard algorithms which is used in text categorization classification.

3.) Bernoulli Naïve Bayes algorithm:

- In multivariate Bernoulli event model, features are independent boolean variables describing inputs. Just like the multinomial model, this model is also popular for document classification tasks where binary term occurrence features are used rather than term frequencies.

❖ **Dataset taken:** Weather.nominal

- It is a sample dataset present in the direct of WEKA. This dataset predicts if the weather is suitable for playing cricket. The dataset has 5 attributes and 14 instances. The class label "play" classifies the output as "yes' or "no".

- No. of Rows: 14

- No. of Columns: 5 **Procedure:**

❖ **Task – 1: To pre-process the chosen dataset.**

Code:

#Step – 1: Import standard libraries.

```
import numpy as np import
pandas as pd
from sklearn.preprocessing import LabelEncoder
```

#Step – 2: Load the dataset and encode categorical variables.

```
playgolf_data = pd.read_csv('/content/playgolf_data.csv')
```

#Step – 3: Encode categorical variables playgolf_data =

```
pd.get_dummies(playgolf_data, columns=['Outlook'])
```

#Step – 4: Label encode the 'Temperature' column temperature_encoder

```
= LabelEncoder() playgolf_data['Temperature']
=
temperature_encoder.fit_transform(playgolf_data['Temperature'])
```

#Step – 5: Select the relevant columns as features (x) and the target variable (y) x

```
= playgolf_data[['Outlook_Overcast', 'Outlook_Rainy', 'Outlook_Sunny',
'Temperature']].values y =
playgolf_data['PlayGolf'].values
```

❖ **Task – 2: To perform feature scaling and fitting Naïve Bayes to training dataset.**

Code:

#Step – 1: Split the dataset into the training set and test set from sklearn.model_selection

import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=0)

#Step – 2: Perform feature scaling from

sklearn.preprocessing import StandardScaler scaler

= StandardScaler() x_train =

scaler.fit_transform(x_train) x_test =

scaler.transform(x_test)

#Step – 3: Fitting Naive Bayes to the Training Set:

from sklearn.naive_bayes import GaussianNB

classifier = GaussianNB() classifier.fit(x_train, y_train)

❖ **Task – 2: To Predict the test set result. Code:**

#Step – 1: Import libraries and load the dataset.

import pandas as pd from sklearn.model_selection

import train_test_split

#load the dataset playgolf_data =

pd.read_csv('/content/playgolf_data.csv') **#Step – 2: Select the relevant columns as features (x) and the target variable (y).** x = playgolf_data[['Outlook',

'Temperature']].values y = playgolf_data['PlayGolf'].values

x_train, x_test, y_train, y_test = train_test_split(x, y,

test_size=0.25, random_state=0) classifier.fit(x_train,

y_train) y_pred = classifier.predict(x_test)

❖ **Task – 4: To create confusion matrix for the Naïve Bayes classifier. #Step - 1: Import the require libraries.**

import numpy as np import pandas as pd from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import confusion_matrix import

seaborn as sns import matplotlib.pyplot as plt

#Step – 2: Assume data as a NumPy array with features and labels.

playgolf_data = np.array([[25, 50000, 1], [30, 60000, 0], [35, 70000, 1], [40, 80000, 0],

[45, 90000, 1], [50, 100000, 0]]) x_set,

y_set = playgolf_data[:, :-1], playgolf_data[:, -1]

#Step – 3: Train a Naive Bayes classifier

classifier = GaussianNB() classifier.fit(x_set,

y_set)

# Predict labels for the data points y_pred

= classifier.predict(x_set)

#Step – 4: Compute the confusion matrix. confusion = confusion_matrix(y_set, y_pred)

confusion_df = pd.DataFrame(confusion, index=['Actual No', 'Actual

Yes'], columns=['Predicted No', 'Predicted Yes']) print("Confusion Matrix:")

print(confusion_df) sns.heatmap(confusion_df, annot=True, fmt='d',

cmap='Blues', cbar=False) plt.xlabel('Predicted label') plt.ylabel('True label')

plt.title('Confusion Matrix') plt.show()

❖ **Task – 5: To create confusion matrix for the Naïve Bayes classifier. Code:**

#Step – 1: Import the standard libraries.

import numpy as np import pandas as pd from

sklearn.naive_bayes import GaussianNB from

sklearn.model_selection import train_test_split

#Step – 2: Assuming playgolf_data is a NumPy array with features and labels.

playgolf_data = np.array([[25, 50000, 1],

[30, 60000, 0],

[35, 70000, 1],

[40, 80000, 0],

[45, 90000, 1], [50,

100000, 0]]) x = playgolf_data[:, :-

1] y = playgolf_data[:, -1]

#Step – 3: Split the dataset into the training set and test set. x_train, x_test, y_train,

y_test = train_test_split(x, y, test_size=0.25, random_state=0) classifier =

GaussianNB() classifier.fit(x_train, y_train)

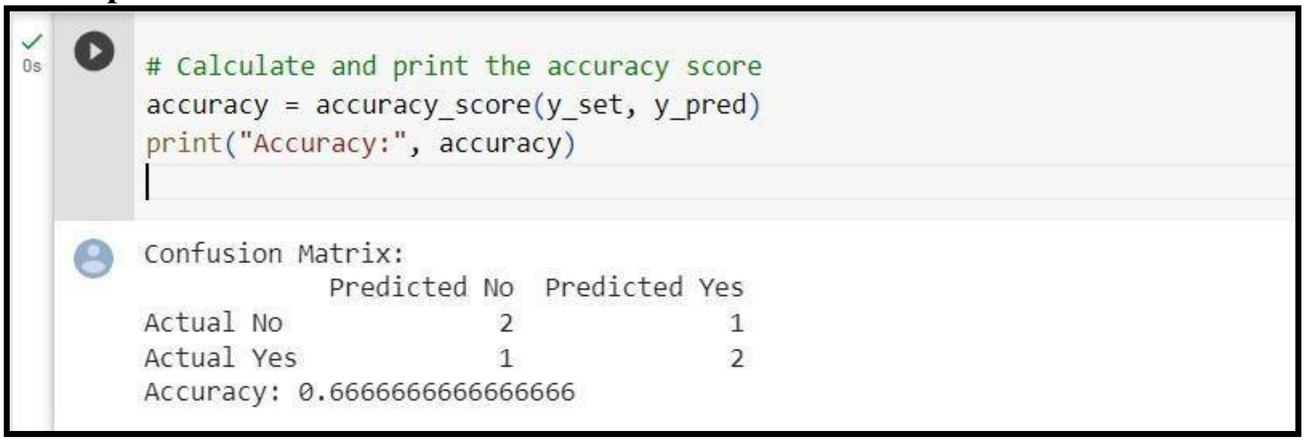
#Step – 4: Predict labels for the data points y_pred

= classifier.predict(x_test)

#Step – 5: Calculate and print the accuracy score.

accuracy = accuracy_score(y_set, y_pred) print("Accuracy:", accuracy)

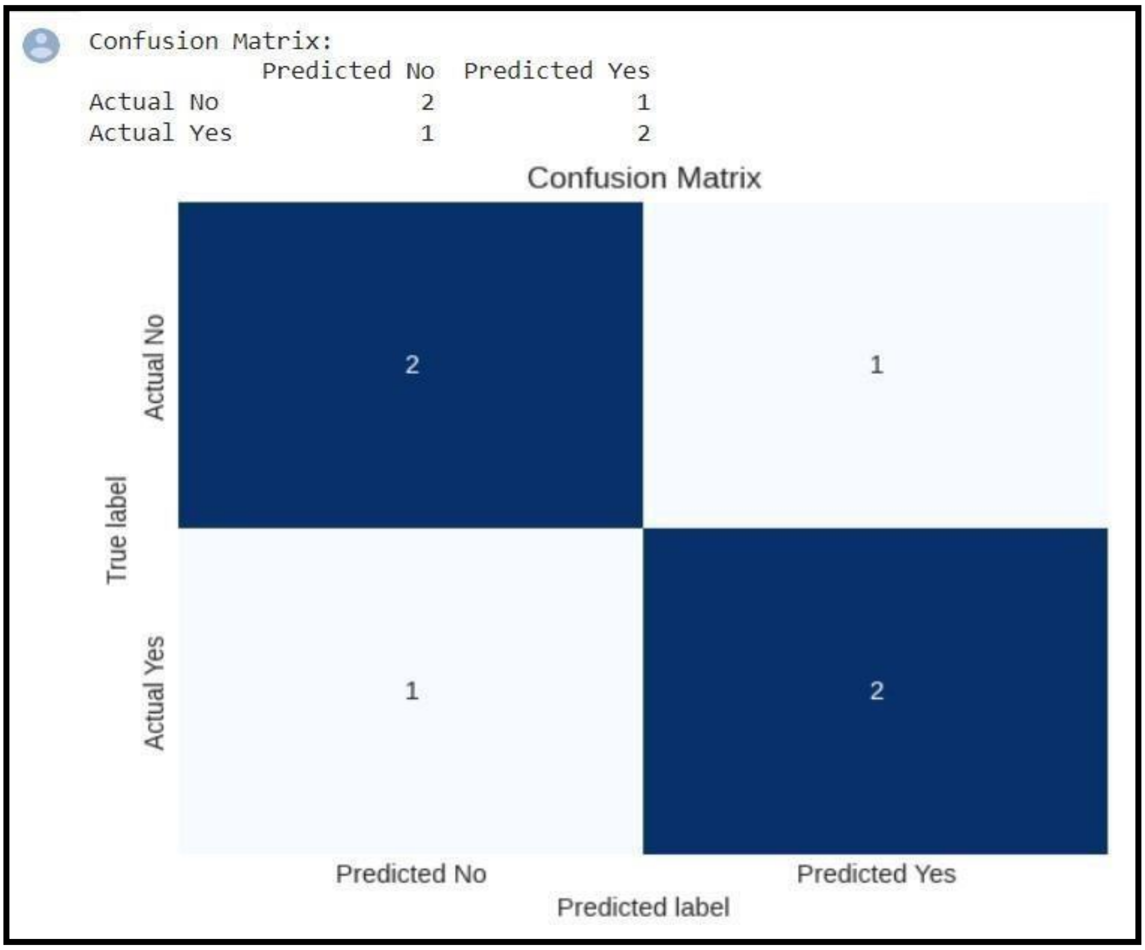
❖ Output:



```
# Calculate and print the accuracy score
accuracy = accuracy_score(y_set, y_pred)
print("Accuracy:", accuracy)
```

```
Confusion Matrix:
            Predicted No  Predicted Yes
Actual No              2              1
Actual Yes             1              2
Accuracy: 0.6666666666666666
```

Figure _ 3.2: Confusion Matrix



```
Confusion Matrix:
            Predicted No  Predicted Yes
Actual No              2              1
Actual Yes             1              2
```

Figure – 3.3: Confusion Matrix

Practical - 4

Aim: Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task.

Theory:

❖ **What is Naïve Bayes Classifier?**

- The Naive Bayes classifier is a simple yet powerful probabilistic machine learning algorithm that is commonly used for classification tasks. It is based on Bayes' theorem and assumes that the features are conditionally independent given the class label. This assumption is known as the "naive" assumption, which simplifies the calculation of probabilities.

- In machine learning, Naïve Bayes classification is a straightforward and powerful algorithm for the classification task. Naïve Bayes classification is based on applying Bayes' theorem with strong independence assumption between the features. Naïve Bayes classification produces good results when we use it for textual data analysis such as Natural Language Processing.

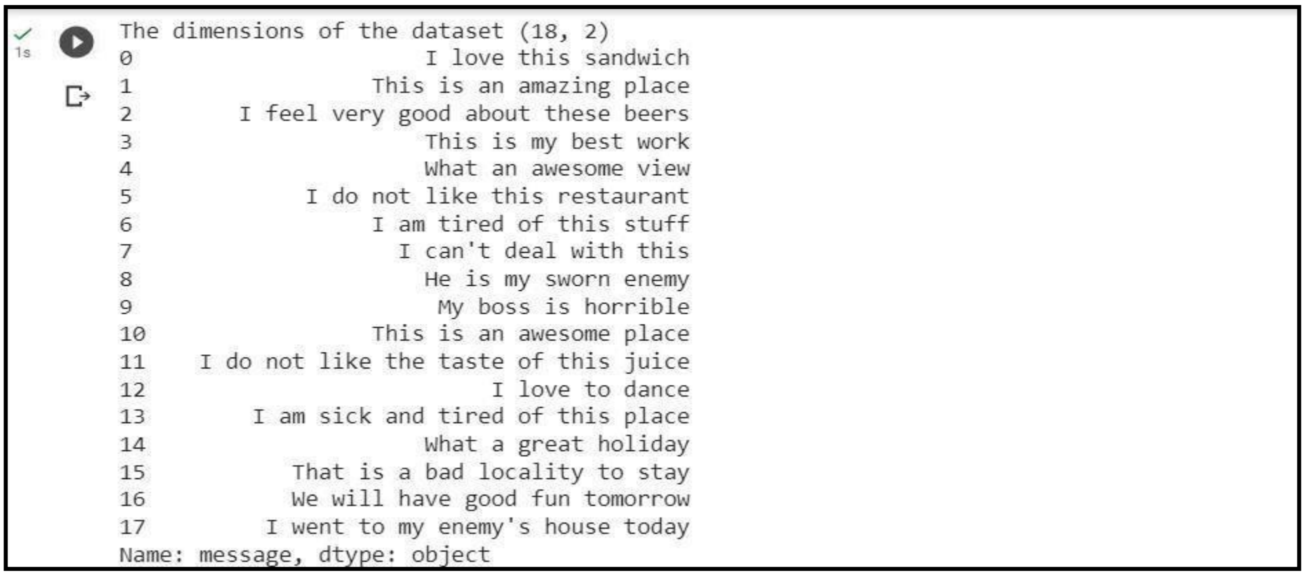
- Naïve Bayes models are also known as simple Bayes or independent Bayes. All these names refer to the application of Bayes' theorem in the classifier's decision rule. Naïve Bayes classifier applies Bayes' theorem in practice. This classifier brings the power of Bayes' theorem to machine learning.

❖ **Dataset taken:** Document(Sample dataset)

- Document is a set of text documents along with their target values. V is the set of all possible target values. This function learns the probability terms P(wk |vj,), describing the probability that a randomly drawn word from a document in class vj will be the English word wk.

- No. of Rows: 18 - No. of Columns: 2

❖ **Dataset taken:**



```
The dimensions of the dataset (18, 2)
0                    I love this sandwich
1                This is an amazing place
2          I feel very good about these beers
3                    This is my best work
4                    What an awesome view
5              I do not like this restaurant
6                  I am tired of this stuff
7                    I can't deal with this
8                    He is my sworn enemy
9                     My boss is horrible
10               This is an awesome place
11     I do not like the taste of this juice
12                        I love to dance
13           I am sick and tired of this place
14                   What a great holiday
15             That is a bad locality to stay
16             We will have good fun tomorrow
17           I went to my enemy's house today
Name: message, dtype: object
```

Figure – 4.1: Dataset records **Procedure:**

❖ **Task – 1: To import libraries and load dataset.**

Code:

#Step – 1: Import standard libraries.

```
import pandas as pd
```

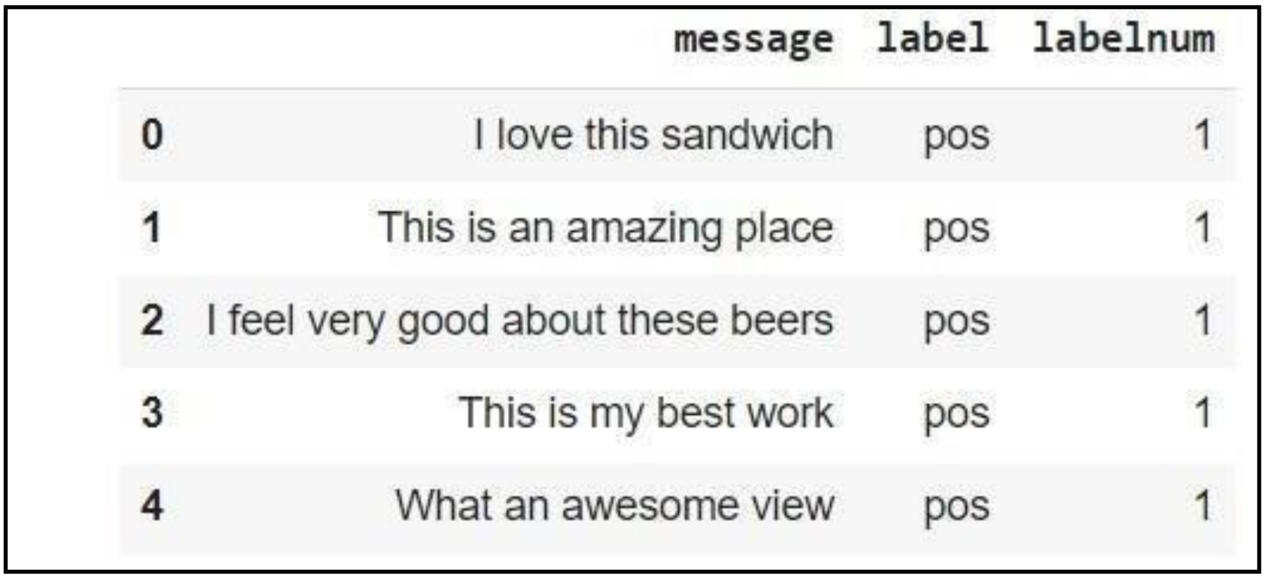
#Step – 2: Read the dataset.

```
msg=pd.read_csv('document.csv',names=['message','label']) print('The
dimensions of the dataset',msg.shape)
```

#Step – 3: Print the top 5 rows of the dataset.

```
msg['labelnum']=msg.label.map({'pos':1,'neg':0})
X=msg.message y=msg.labelnum print(X) print(y)
msg.head()
```

❖ **Output:**



	message	label	labelnum
0	I love this sandwich	pos	1
1	This is an amazing place	pos	1
2	I feel very good about these beers	pos	1
3	This is my best work	pos	1
4	What an awesome view	pos	1

Figure – 4.2: Top 5 records of the dataset

❖ **Task – 2: To import libraries and load dataset.**

Code:

#Step – 1: Splitting the dataset into train and test data

from sklearn.model_selection import train_test_split

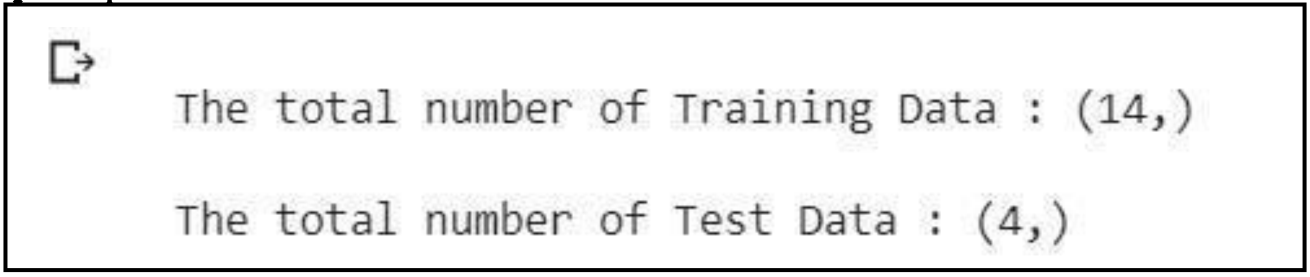
xtrain,xtest,ytrain,ytest=train_test_split(X, y, test_size=4,

random_state=4) print ('\n The total number of Training

Data :',ytrain.shape) print ('\n The total number of Test

Data :',ytest.shape)

❖ **Output:**



```
The total number of Training Data : (14,)

The total number of Test Data : (4,)
```

Figure – 4.3: Number of train and test data

#Step – 2: Output of count vectoriser is a sparse matrix from

sklearn.feature_extraction.text import CountVectorizer count_vect

= CountVectorizer() xtrain_dtm =

count_vect.fit_transform(xtrain)

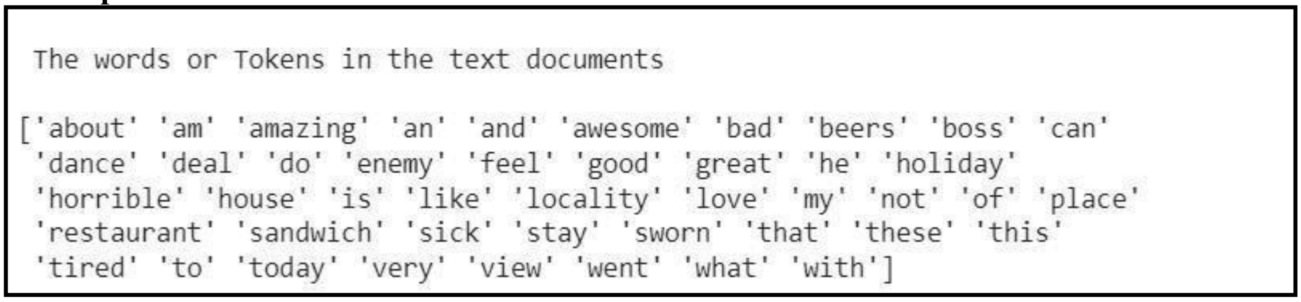
xtest_dtm=count_vect.transform(xtest)

print('\n The words or Tokens in the text documents \n')

print(count_vect.get_feature_names_out())

df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_feature_names_out())

❖ **Output:**



```
 The words or Tokens in the text documents

['about' 'am' 'amazing' 'an' 'and' 'awesome' 'bad' 'beers' 'boss' 'can'
 'dance' 'deal' 'do' 'enemy' 'feel' 'good' 'great' 'he' 'holiday'
 'horrible' 'house' 'is' 'like' 'locality' 'love' 'my' 'not' 'of' 'place'
 'restaurant' 'sandwich' 'sick' 'stay' 'sworn' 'that' 'these' 'this'
 'tired' 'to' 'today' 'very' 'view' 'went' 'what' 'with']
```

Figure – 4.4: Words taken in document

❖ **Task – 3: To train the classifier and print accuracy.**

Code:

#Step – 1: Training Naive Bayes (NB) classifier on training data.

from sklearn.naive_bayes import MultinomialNB clf =

MultinomialNB().fit(xtrain_dtm,ytrain) predicted =

clf.predict(xtest_dtm)

#Step – 2: Printing accuracy, Confusion matrix, Precision and Recall from sklearn

import metrics

print('\n Accuracy of the classifer is',metrics.accuracy_score(ytest,predicted)) print('\n

Confusion matrix') print(metrics.confusion_matrix(ytest,predicted))

print('\n The value of Precision' ,metrics.precision_score(ytest,predicted))

print('\n The value of Recall' , metrics.recall_score(ytest,predicted)) print('\nThe

value of f1 score:', metrics.f1_score(ytest, predicted))

#Step – 3: Plot the confusion matrix as a heatmap

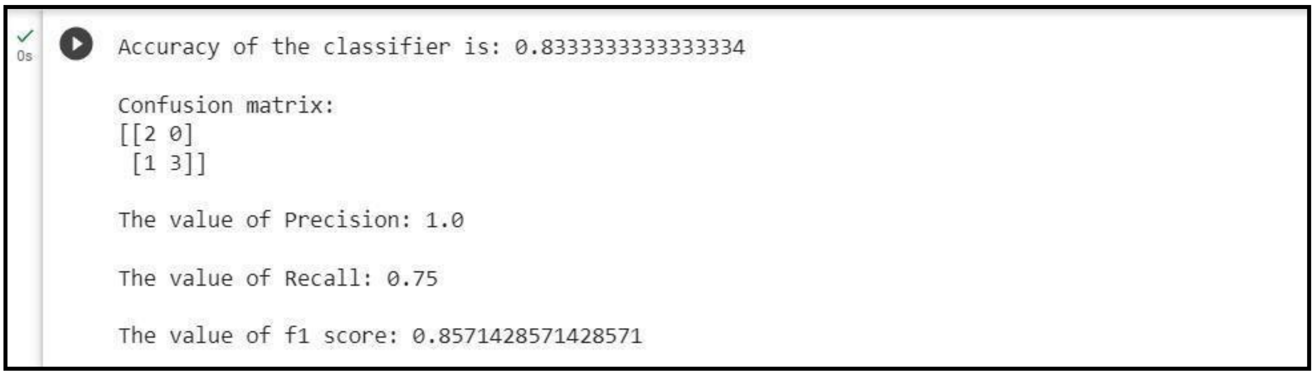
sns.heatmap(confusion, annot=True, fmt='d', cmap='Blues', cbar=False) plt.xlabel('Predicted

label')

plt.ylabel('True label')

plt.title('Confusion Matrix') plt.show()

❖ **Output:**



```
Accuracy of the classifier is: 0.8333333333333334

Confusion matrix:
[[2 0]
 [1 3]]

The value of Precision: 1.0

The value of Recall: 0.75

The value of f1 score: 0.8571428571428571
```

Figure _ 4.5: Acurracy, Precision, and Recall

❖ Output:

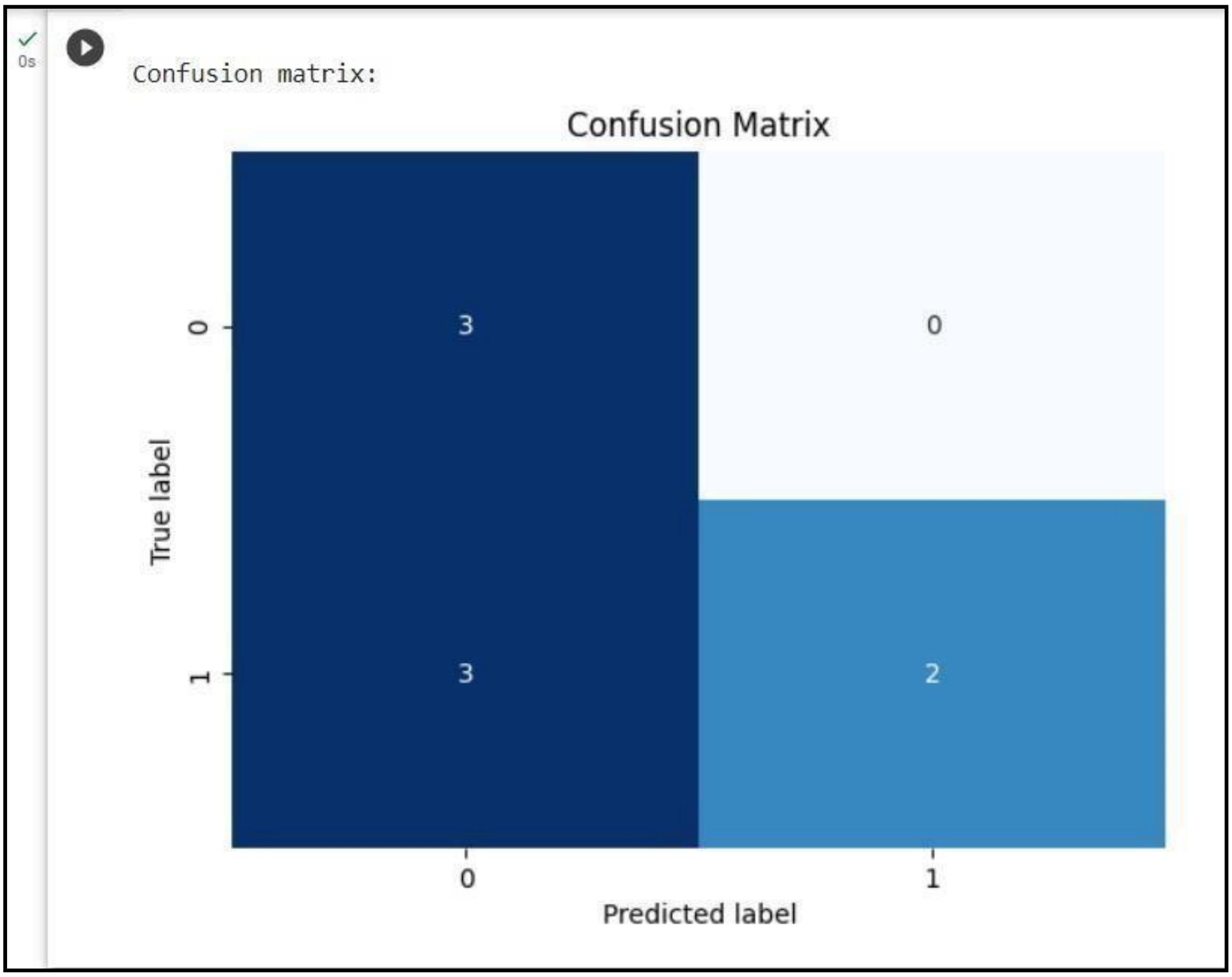

```
Confusion matrix:
```

Figure _ 4.6: Confusion Matrix

❖ Final Output Table:

	Dataset split %		Accuracy	Precision	Recall	f1 score
	Training set	**Test set**				
1.	**80**	**40**	**0.75**	**1.0**	**0.5**	**0.666**
2.	**70**	**30**	**0.83**	**1.0**	**0.75**	**0.857**
3.	**60**	**20**	**0.71**	**0.75**	**0.75**	**0.75**

Figure – 4.7: Table of Acurracy, Precision, and Recall ❖ ***Conclusion:***

Successfully performed the Naïve Bayes classification of document using python libraries and Google colab tool. Performance-wise the Naïve Bayes classifier has superior performance compared to many other classifiers. The final output table is obtained after performing different dataset splits of both training and testing dataset. Overall, the Accuracy increases if we take optimal test set data for training.

Practical - 5

Aim: Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.

Theory:

❖ **What is Naïve Bayes Classifier?**

- The Naive Bayes classifier is a simple yet powerful probabilistic machine learning algorithm that is commonly used for classification tasks. It is based on Bayes' theorem and assumes that the features are conditionally independent given the class label. This assumption is known as the "naive" assumption, which simplifies the calculation of probabilities.

- In machine learning, Naïve Bayes classification is a straightforward and powerful algorithm for the classification task. Naïve Bayes classification is based on applying Bayes' theorem with strong independence assumption between the features. Naïve Bayes classification produces good results when we use it for textual data analysis such as Natural Language Processing.

- Naïve Bayes models are also known as simple Bayes or independent Bayes. All these names refer to the application of Bayes' theorem in the classifier's decision rule. Naïve Bayes classifier applies
Bayes' theorem in practice. This classifier brings the power of Bayes' theorem to machine learning.
- The Naive Bayes classifier is built on the foundation of Bayes' theorem, which relates conditional probabilities. Naïve Bayes Classifier uses the Bayes' theorem to predict membership probabilities for each class such as the probability that given record or data point belongs to a particular class. The class with the highest probability is considered as the most likely class.

❖ **The Naive Bayes classifier works as follows:**

1.) **Training:** Given a labelled training dataset, the classifier calculates the prior probability P(y) for each class in the dataset. It also estimates the likelihood probability P(X|y) for each feature given each class. This is done by assuming conditional independence between the features.

2.) **Prediction:** When a new unlabelled instance is presented, the classifier calculates the posterior probability P(y|X) for each class using Bayes' theorem. It then assigns the class label with the highest posterior probability as the predicted class for that instance.

3.) **Handling Continuous Features:** For continuous features, the Naive Bayes classifier typically assumes a probability distribution, often Gaussian (hence called Gaussian Naive Bayes), to estimate the likelihood probability.

4.) **Laplace Smoothing:** To avoid zero probabilities when a feature value in the testing data was not observed in the training data, Laplace smoothing (also known as additive smoothing) is often applied. It adds a small constant to numerator and adjusts the denominator accordingly.

5.) **Decision Rule:** In some cases, the Naive Bayes classifier can be used for decision making by considering the posterior probabilities. For example, in binary classification, if P(y=1|X) > P(y=0|X), the instance is assigned to class 1; otherwise, it is assigned to class 0.

❖ **Dataset taken: Heartdisease**

- The dataset consists of medical data with seven attributes: age, gender, family history, diet, lifestyle, cholesterol level, and the presence or absence of heart disease. Each row represents an individual's information, including their attributes and heart disease status, with binary values indicating the presence (1) or absence (0) of heart disease.
- No. of Rows: 19
- No. of Columns: 7 **Procedure:**

#Step – 1: Install the pgmpy library using pip.

```
!pip install pgmpy
```

#Step – 2: Import necessary libraries and modules.

```
import pandas as pd
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel from pgmpy.inference
import VariableElimination
```

#Step – 3: Load the Dataset and print it. data

```
= pd.read_csv("ds4.csv") heart_disease
= pd.DataFrame(data) print(heart_disease)
```

#Step – 4: Define the Bayesian Network model. model

```
= BayesianModel([
   ('age', 'Lifestyle'),
   ('Gender', 'Lifestyle'),
   ('Family', 'heartdisease'),
   ('diet', 'cholestrol'),
   ('Lifestyle', 'diet'),
   ('cholestrol', 'heartdisease'),
   ('diet', 'cholestrol')
])
```

#Step – 5: Fit the model using MLE.

model.fit(heart_disease, estimator=MaximumLikelihoodEstimator)

#Step – 7: Create a Variable Elimination object.

HeartDisease_infer = VariableElimination(model)

#Step – 8: Instructions for entering values. print('For Age enter SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4') print('For Gender enter Male:0, Female:1') print('For Family History enter Yes:1, No:0') print('For Diet enter High:0, Medium:1') print('For Lifestyle enter Athlete:0, Active:1, Moderate:2, Sedentary:3') print('For Cholesterol enter High:0, BorderLine:1, Normal:2')

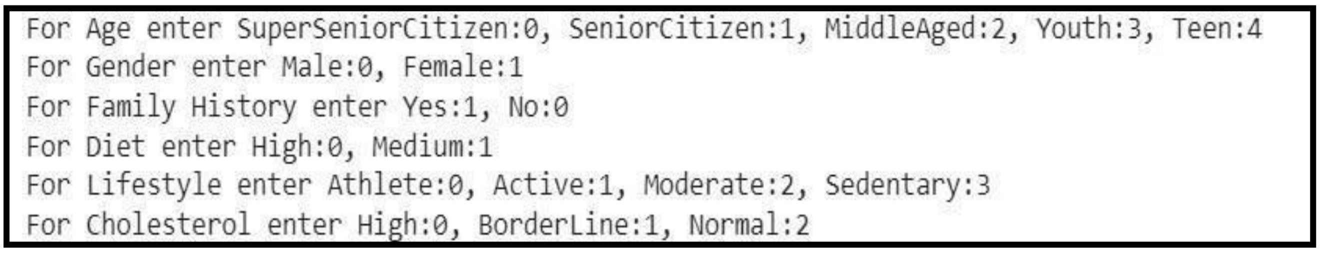
#Step – 9: Perform heart disease diagnosis based on user-provided attributes. q

```
= HeartDisease_infer.query(variables=['heartdisease'], evidence= {
    'age': int(input('Enter Age: ')),
    'Gender': int(input('Enter Gender: ')),
    'Family': int(input('Enter Family History: ')),
    'diet': int(input('Enter Diet: ')),
    'Lifestyle': int(input('Enter Lifestyle: ')),
    'cholestrol': int(input('Enter Cholesterol: '))
})
```

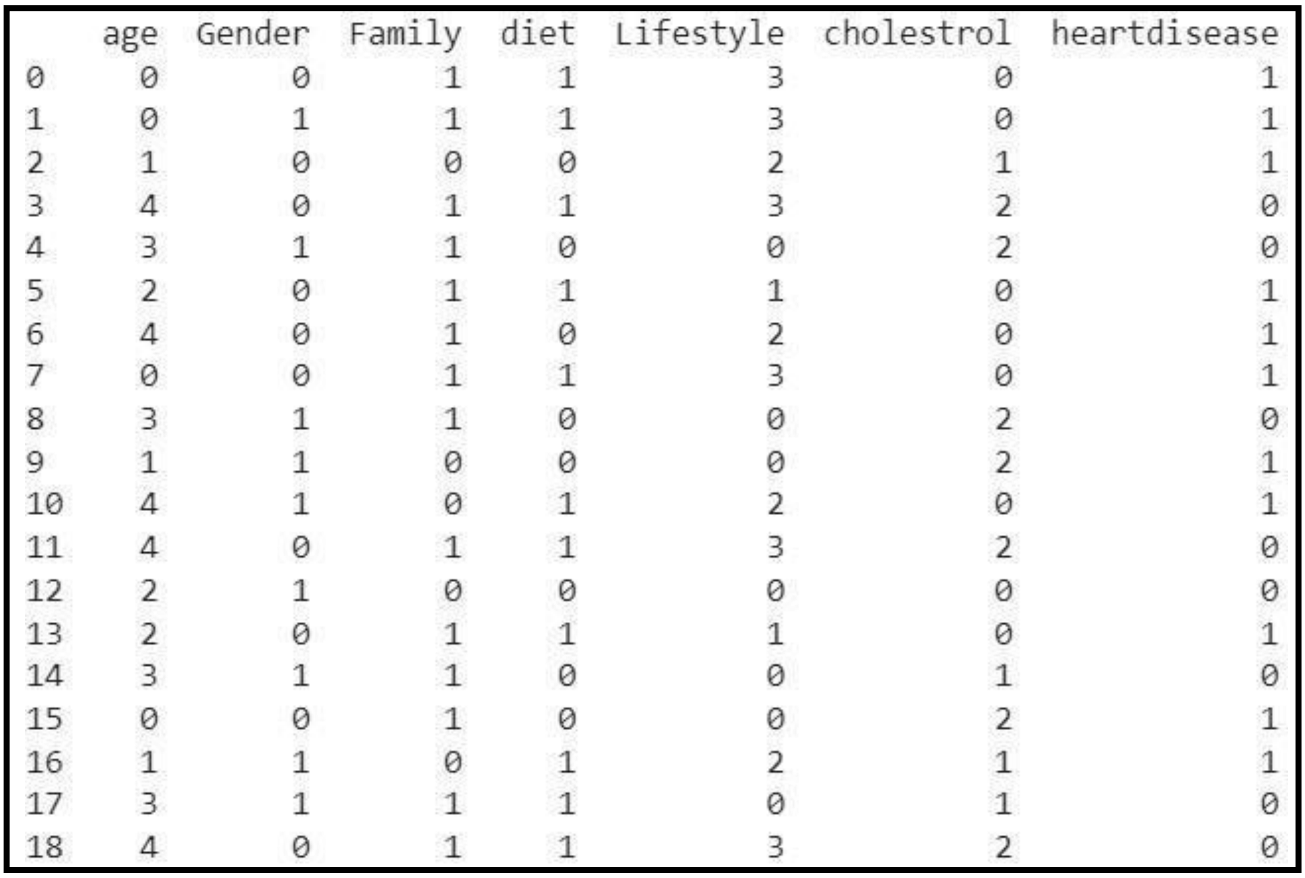
#Step – 10: Print the result of the heart disease diagnosis.

print(q)

❖ Output:

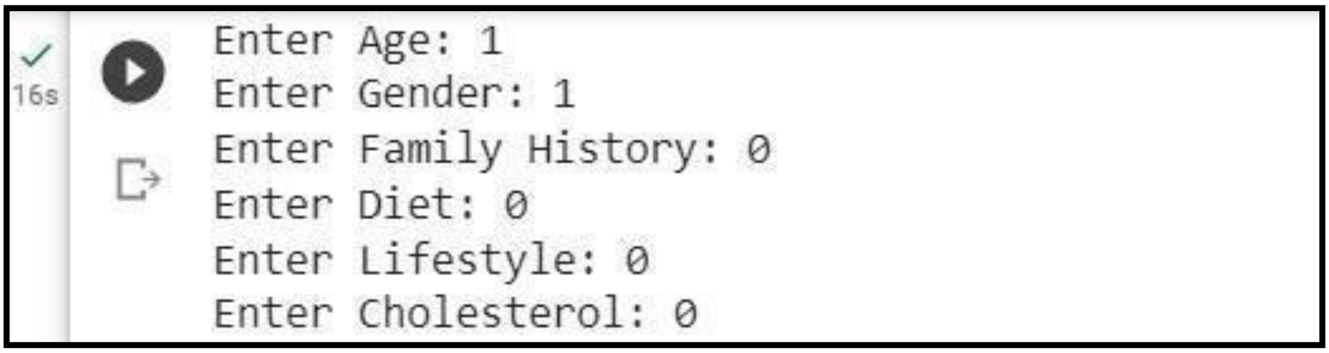


```
For Age enter SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4
For Gender enter Male:0, Female:1
For Family History enter Yes:1, No:0
For Diet enter High:0, Medium:1
For Lifestyle enter Athlete:0, Active:1, Moderate:2, Sedentary:3
For Cholesterol enter High:0, BorderLine:1, Normal:2
```



```
    age  Gender  Family  diet  Lifestyle  cholestrol  heartdisease
0     0       0       1     1          3           0             1
1     0       1       1     1          3           0             1
2     1       0       0     0          2           1             1
3     4       0       1     1          3           2             0
4     3       1       1     0          0           2             0
5     2       0       1     1          1           0             1
6     4       0       1     0          2           0             1
7     0       0       1     1          3           0             1
8     3       1       1     0          0           2             0
9     1       1       0     0          0           2             1
10    4       1       0     1          2           0             1
11    4       0       1     1          3           2             0
12    2       1       0     0          0           0             0
13    2       0       1     1          1           0             1
14    3       1       1     0          0           1             0
15    0       0       1     0          0           2             1
16    1       1       0     1          2           1             1
17    3       1       1     1          0           1             0
18    4       0       1     1          3           2             0
```

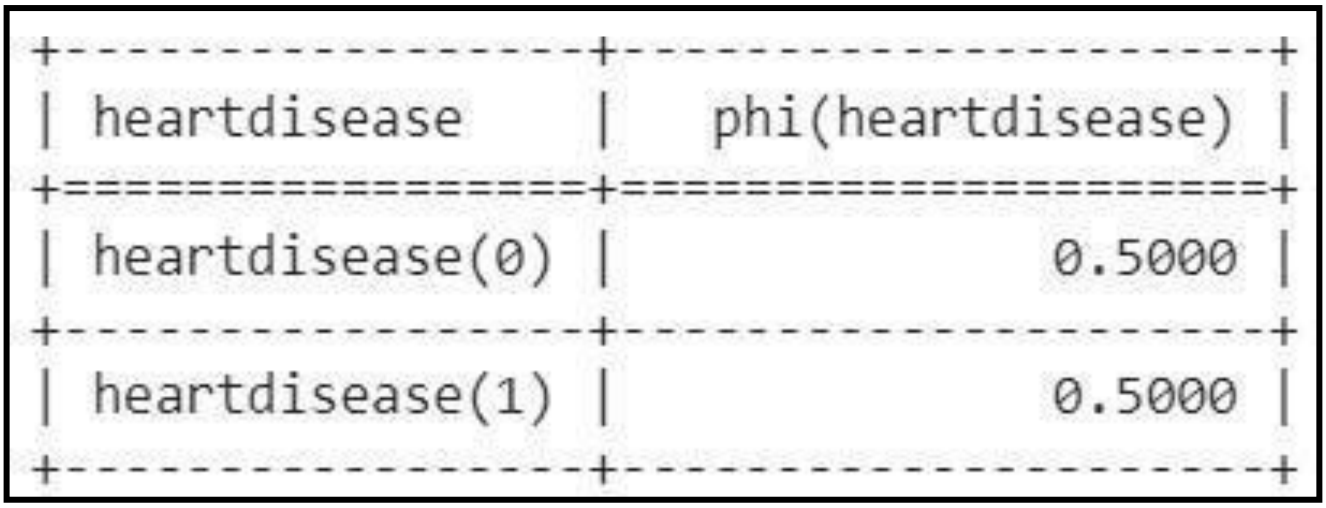
Figure – 5.1: Dataset of Heart Disease Figure – 5.2: Instructions for entering values of attributes.



```
Enter Age: 1
Enter Gender: 1
Enter Family History: 0
Enter Diet: 0
Enter Lifestyle: 0
Enter Cholesterol: 0
```

Figure _ 5.3: Provided user inputs

❖ **Final Output:**



```
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.5000 |
+-----------------+---------------------+
| heartdisease(1) |              0.5000 |
+-----------------+---------------------+
```

Figure _ 5.4: Final Output

❖ Observations:

The Bayesian network constructed in this exercise is designed to predict the presence or absence of heart disease based on various medical attributes. When we applied the Bayesian network for heart disease diagnosis using sample attributes, the model produced a probability distribution over the two possible outcomes (presence or absence of heart disease). In this specific case, the model estimated an equal probability of 0.5 for both scenarios, indicating a 50% chance of having heart disease and a 50% chance of not having it based on the provided attributes. This demonstrates the probabilistic nature of the Bayesian network, where it provides uncertainty estimates for different outcomes based on the available evidence.

❖ Conclusion:

In conclusion, this practical exercise demonstrated the construction of a Bayesian network for medical data analysis and showcased its application in diagnosing heart disease using the standard Heart Disease Data Set. The Naïve Bayes Classifier, which is based on Bayes' theorem and the assumption of conditional independence among features, was utilized to create a probabilistic model. The key steps included data loading, defining the Bayesian network structure, fitting the model using Maximum Likelihood Estimator, and performing inference for heart disease diagnosis based on user-provided attributes. By leveraging such models, healthcare professionals can make more informed decisions and improve patient care.

Practical - 6

Aim: Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm.

Theory:

❖ What is EM Algorithm?

The EM algorithm was proposed and named in a seminal paper published in 1977 by Arthur Dempster, Nan Laird, and Donald Rubin. Their work formalized the algorithm and demonstrated its usefulness in statistical modeling and estimation.

The Expectation-Maximization (EM) algorithm is an iterative optimization method that combines different unsupervised machine learning algorithms to find maximum likelihood or maximum posterior estimates of parameters in statistical models that involve unobserved latent variables. The EM algorithm is commonly used for latent variable models and can handle missing data. It consists of an estimation step (E-step) and a maximization step (M-step), forming an iterative process to improve model fit.

- In the E step, the algorithm computes the latent variables i.e. expectation of the log-likelihood using the current parameter estimates.
- In the M step, the algorithm determines the parameters that maximize the expected loglikelihood obtained in the E step, and corresponding model parameters are updated based on the estimated latent variables.

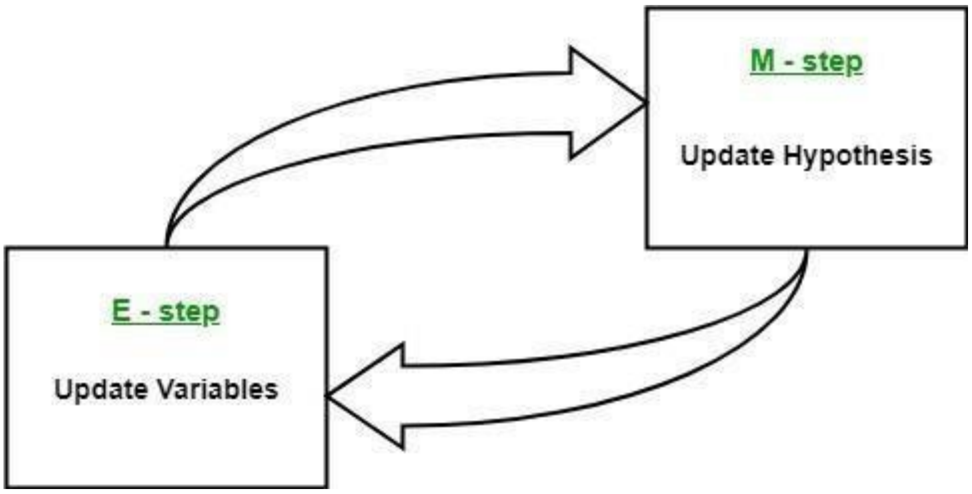


Fig-1: Expectation-Maximization in EM Algorithm

By iteratively repeating these steps, the EM algorithm seeks to maximize the likelihood of the observed data. It is commonly used for unsupervised learning tasks, such as clustering, where latent variables are inferred and has applications in various fields, including ML, computer vision, and NLP.

The essence of the Expectation-Maximization algorithm is to use the available observed data of the dataset to estimate the missing data and then use that data to update the values of the parameters. Let us understand the EM algorithm in detail.

❖ Dataset taken: IRIS Dataset.

- This data sets consists of 3 different types of irises' (Setosa, Versicolour, and Virginica) petal and sepal length, stored in a 150x4 numpy.ndarray. The rows being the samples and the columns being: Sepal Length, Sepal Width, Petal Length and Petal Width.
- No. of Rows: 150
- No. of Columns: 4 **Procedure:**

#Step – 1: Import necessary libraries.

```
import matplotlib.pyplot as plt from
sklearn import datasets from
sklearn.cluster import KMeans
import pandas as pd import numpy as
np
```

#Step – 2: Load the Iris dataset.

```
iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
```

#Step – 3: Rename feature columns for clarity.

```
X.columns = ['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width'] y
= pd.DataFrame(iris.target) y.columns = ['Targets']
```

#Step – 4: Build a K-Means clustering model with 3 clusters.

```
model = KMeans(n_clusters=3) model.fit(X)
```

#Step – 5: Create a figure for plotting with three subplots.

```
plt.figure(figsize=(14, 7))
colormap = np.array(['red', 'lime', 'black'])
```

#Step – 6: Plot the Original Classifications using Petal features.

```
plt.subplot(1, 3, 1)
```

plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40) plt.title('Real Clusters') plt.xlabel('Petal Length') plt.ylabel('Petal Width') **#Step – 7: Plot the Models Classifications (K-Means).**

plt.subplot(1, 3, 2)

plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)

plt.title('K-Means Clustering') plt.xlabel('Petal Length') plt.ylabel('Petal Width')

#Step – 8 : Perform data preprocessing with StandardScaler.

from sklearn import preprocessing scaler = preprocessing.StandardScaler() scaler.fit(X) xsa = scaler.transform(X)

xs = pd.DataFrame(xsa, columns=X.columns)
#Step – 9: Build a GMM with 40 components and fit to the scaled data.

from sklearn.mixture import GaussianMixture gmm = GaussianMixture(n_components=40) gmm.fit(xs)
#Step – 10: Plot GMM clustering results.

plt.subplot(1, 3, 3)

plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[0], s=40)

plt.title('GMM Clustering') plt.xlabel('Petal Length') plt.ylabel('Petal Width')

❖ **Output:**

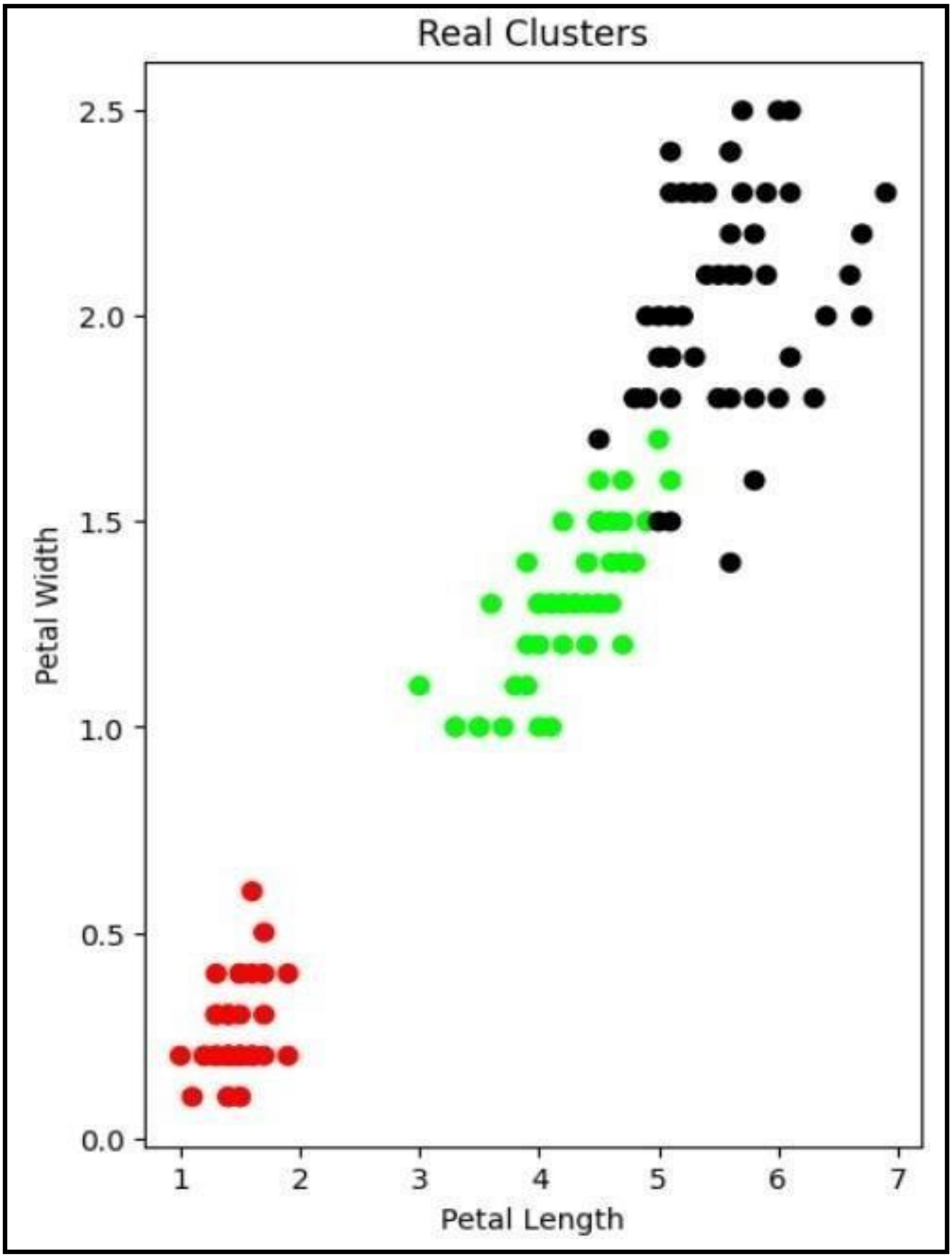


Figure – 6.1: Real Clusters

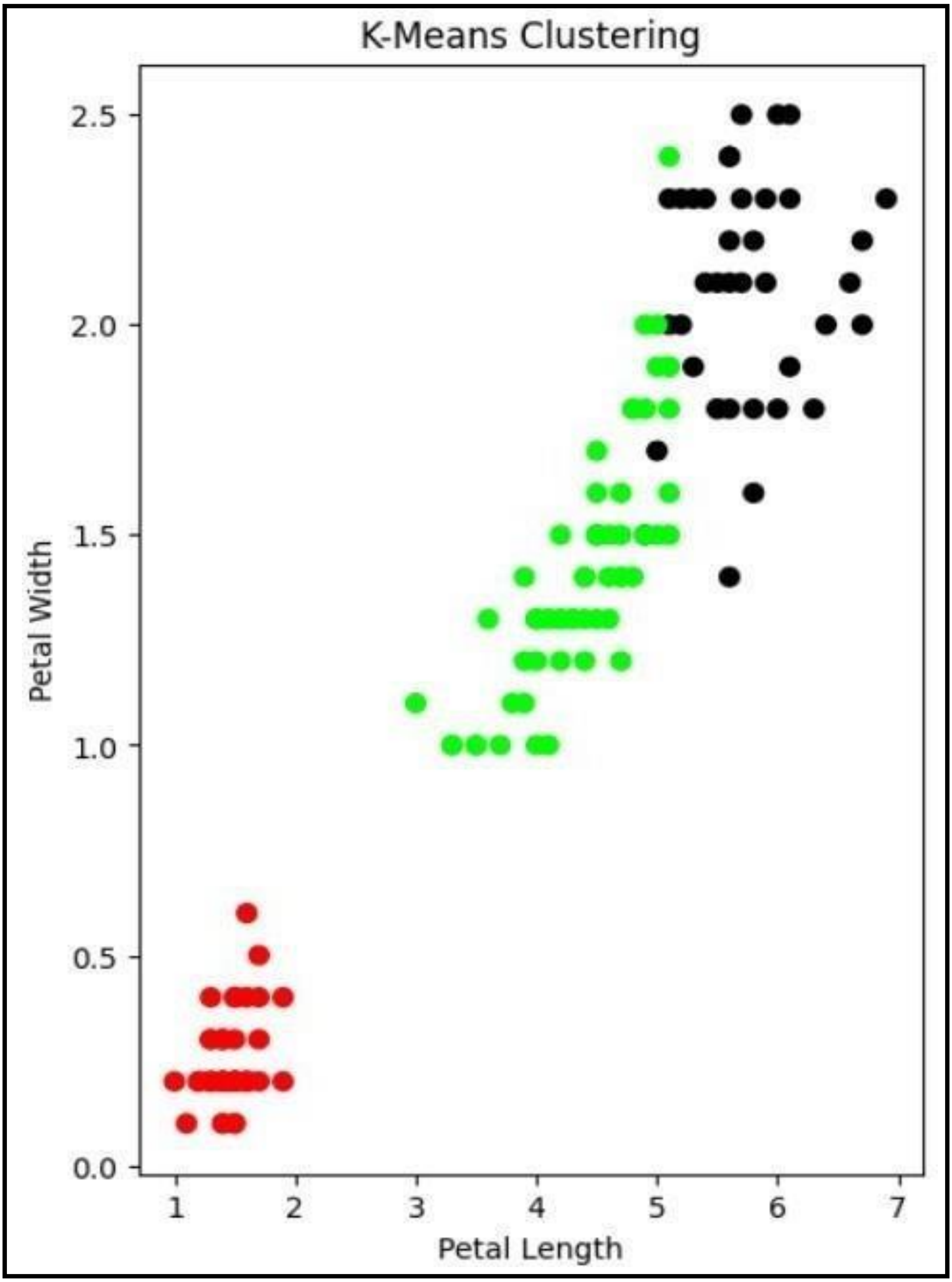

K-Means Clustering
Petal Width
Petal Length
2.5
2.0
1.5
1.0
0.5
0.0
1
2
3
4
5
6
7

Figure – 6.2: After Applying K-Means

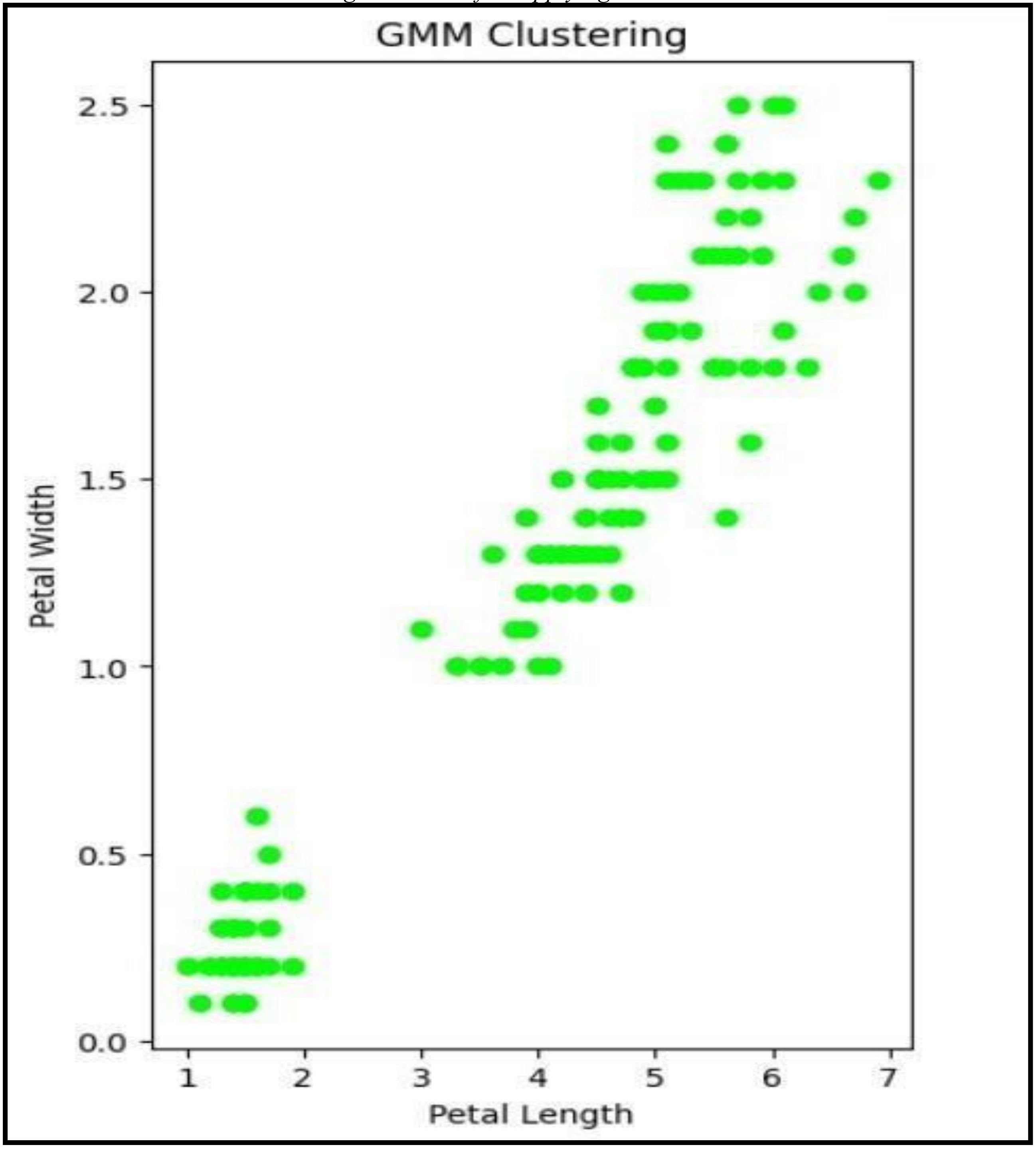


Figure _ 6.3: Final Output of EM Algorithm

❖ Conclusion:

In this practical, we applied both K-Means clustering and Gaussian Mixture Models (GMM) with the Expectation-Maximization (EM) algorithm to cluster the Iris dataset. Our observation revealed that GMM-EM produced clustering results that closely matched the true labels, indicating its effectiveness in capturing the dataset's underlying distribution. K-Means, while reasonable, struggled to account for potential overlaps between clusters.

Practical – 7

Aim: Write a program to implement k-nearest neighbour algorithm to classify the iris dataset.

Theory:

❖ **What is k-nearest neighbour algorithm?**

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique. K-NN algorithm assumes the similarity between the new case/data and available cases and put new case into the category that is most similar to available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm. K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data. It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset. KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset.

Consider the below diagram:

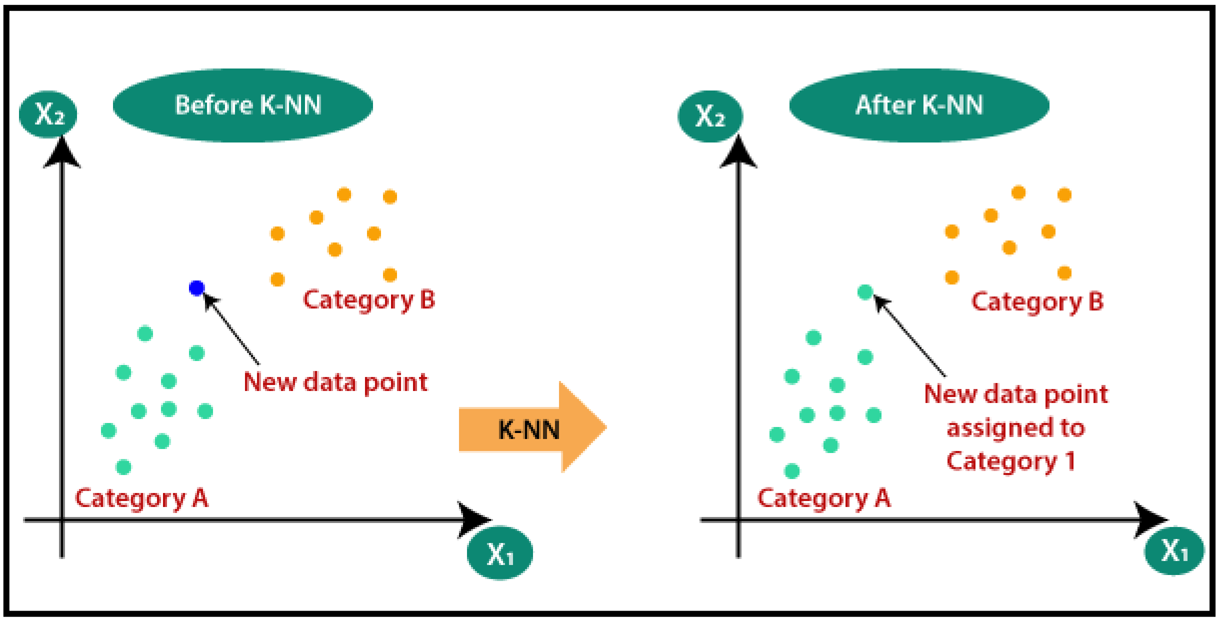


Figure _ 7.1: K-Nearest Neighbour

❖ **Dataset taken:** IRIS Dataset.

- This data sets consists of 3 different types of irises' (Setosa, Versicolour, and Virginica) petal and sepal length, stored in a 150x4 numpy.ndarray.
- The rows being the samples and the columns being: Sepal Length, Sepal Width, Petal Length and Petal Width.
- No. of Rows: 150 - No. of Columns: 4

❖ Procedure:

Task-1: Import necessary libraries and load dadataset.

#Step – 1: Import required python libraries.

import numpy as np import pandas as pd import matplotlib.pyplot as plt from sklearn.datasets

import load_iris from sklearn.neighbors import KNeighborsClassifier from

sklearn.model_selection import train_test_split from sklearn.metrics import

classification_report, confusion_matrix,ConfusionMatrixDisplay **#Step – 2: Load the IRIS**

Dataset.

data = load_iris() X =

data.data Y = data.target

classes = data.target_names

#Step – 3: Print the head of the dataset df = pd.DataFrame(X,

columns=data.feature_names)

print(df.head())

#Step-4: Plot the label distribution.

plt.figure(figsize=(8, 6))

plt.hist(Y, rwidth=1)

plt.title('Label Distribution')

plt.xlabel('Labels (0 / 1 / 2)')

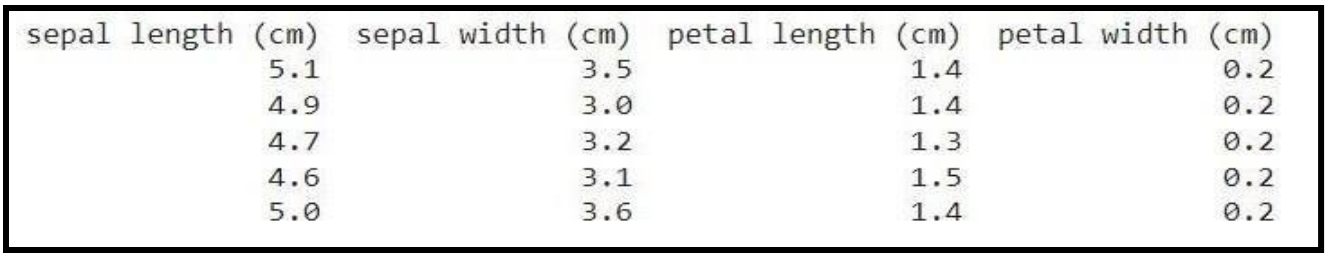
plt.ylabel('Count')

plt.show()

# Step – 5: Print the target label names.

```
for i, name in enumerate(classes):
print(f"Target {i} is {name}")
```

Output:



```
sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
              5.1               3.5                1.4               0.2
              4.9               3.0                1.4               0.2
              4.7               3.2                1.3               0.2
              4.6               3.1                1.5               0.2
              5.0               3.6                1.4               0.2
```

Figure _ 7.2: Output of head function

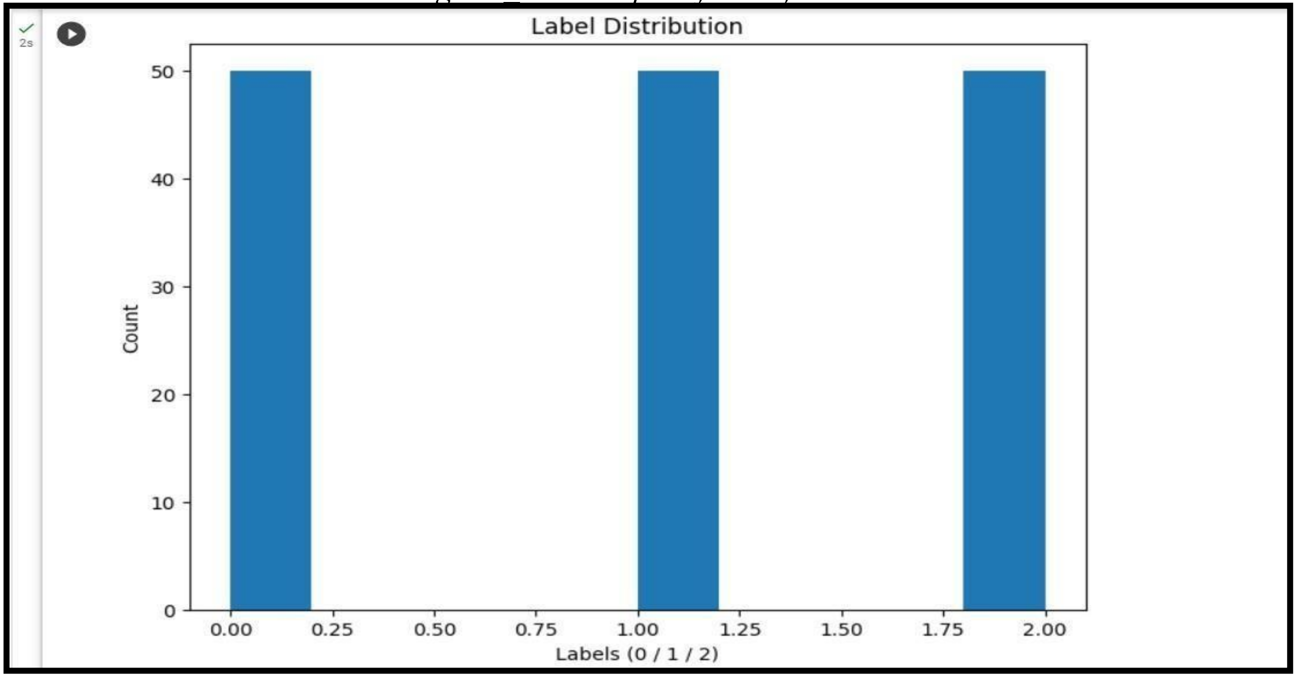


Figure _ 7.3: Label Distribution

#Task 2: Data Splitting and Model Training.

Code:

# Step 1: Split the dataset into training and testing sets.

```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, shuffle=True, random_state=42)
```

# Step 2: Create and train the KNN model.

```
model = KNeighborsClassifier() model.fit(x_train, y_train)
```

# Task 3: Evaluation and Visualization Code: #

Step – 1: Make predictions on the test set

y_pred = model.predict(x_test) y_pred_train

= model.predict(x_train)

# Step – 2: Compute and plot the confusion matrix for the training set confusion_train

= confusion_matrix(y_train, y_pred_train) conf_display_train = ConfusionMatrixDisplay(confusion_train, display_labels=classes)

conf_display_train.plot(cmap=plt.cm.Blues, colorbar=False) plt.title('Training Set Confusion Matrix') plt.show()

# Step – 3: Compute and plot the confusion matrix for the test set confusion_test

= confusion_matrix(y_test, y_pred) conf_display_test = ConfusionMatrixDisplay(confusion_test, display_labels=classes)

conf_display_test.plot(cmap=plt.cm.Blues, colorbar=False) plt.title('Test Set Confusion Matrix') plt.show()

#Task – 4: Print the classification report.

Code:

# Step – 1: Compute and print the classification report.

classification_rep = classification_report(y_test, y_pred, target_names=classes, output_dict=True)

df_classification_rep = pd.DataFrame(classification_rep).transpose()

#Step – 2: Print the classification report in a tabular format with a border size of 1.

print(tabulate(df_classification_rep, headers='keys', tablefmt='fancy_grid'))

Final Output:

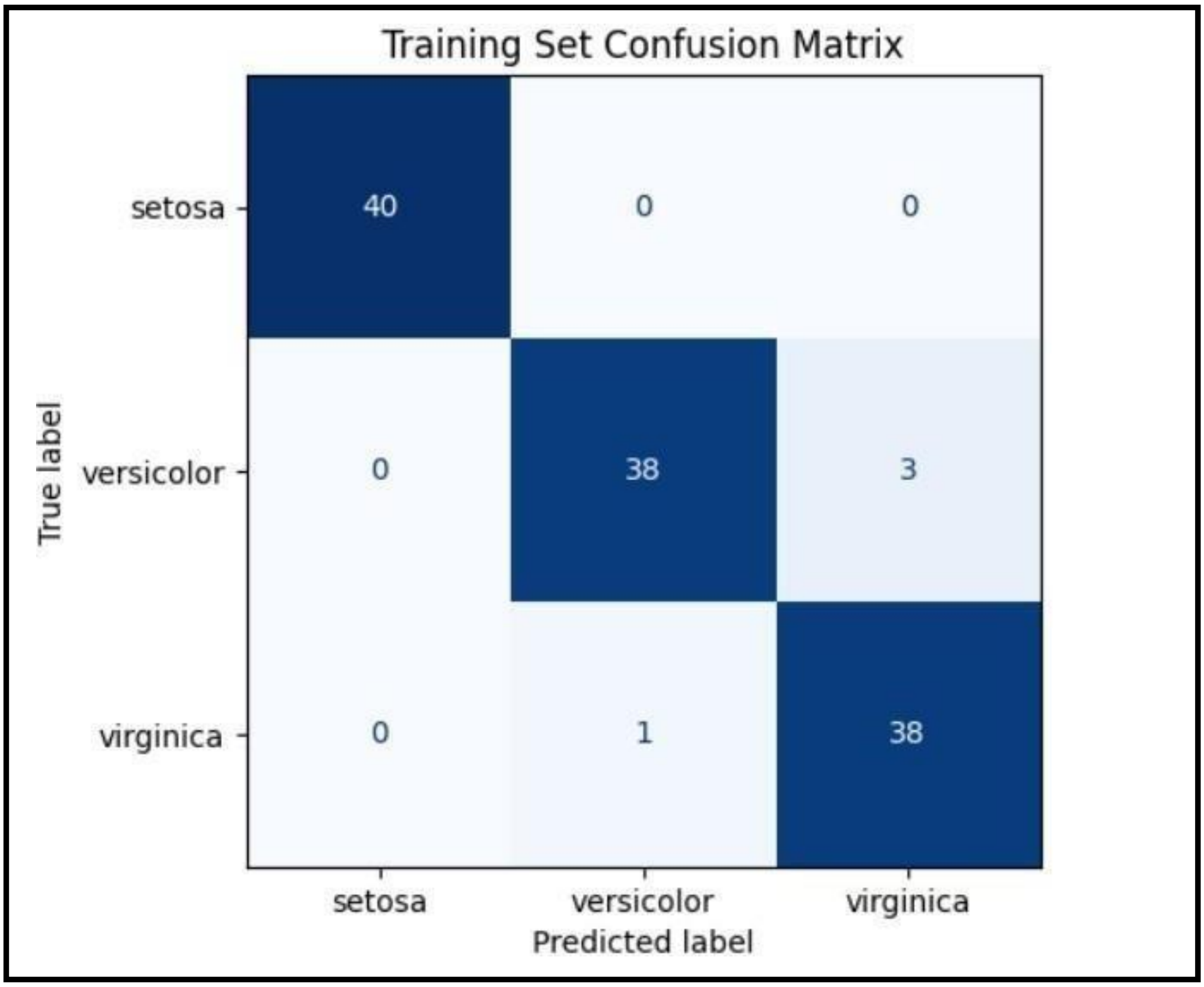


Figure – 7.4: Confusion Matrix for Training dataset

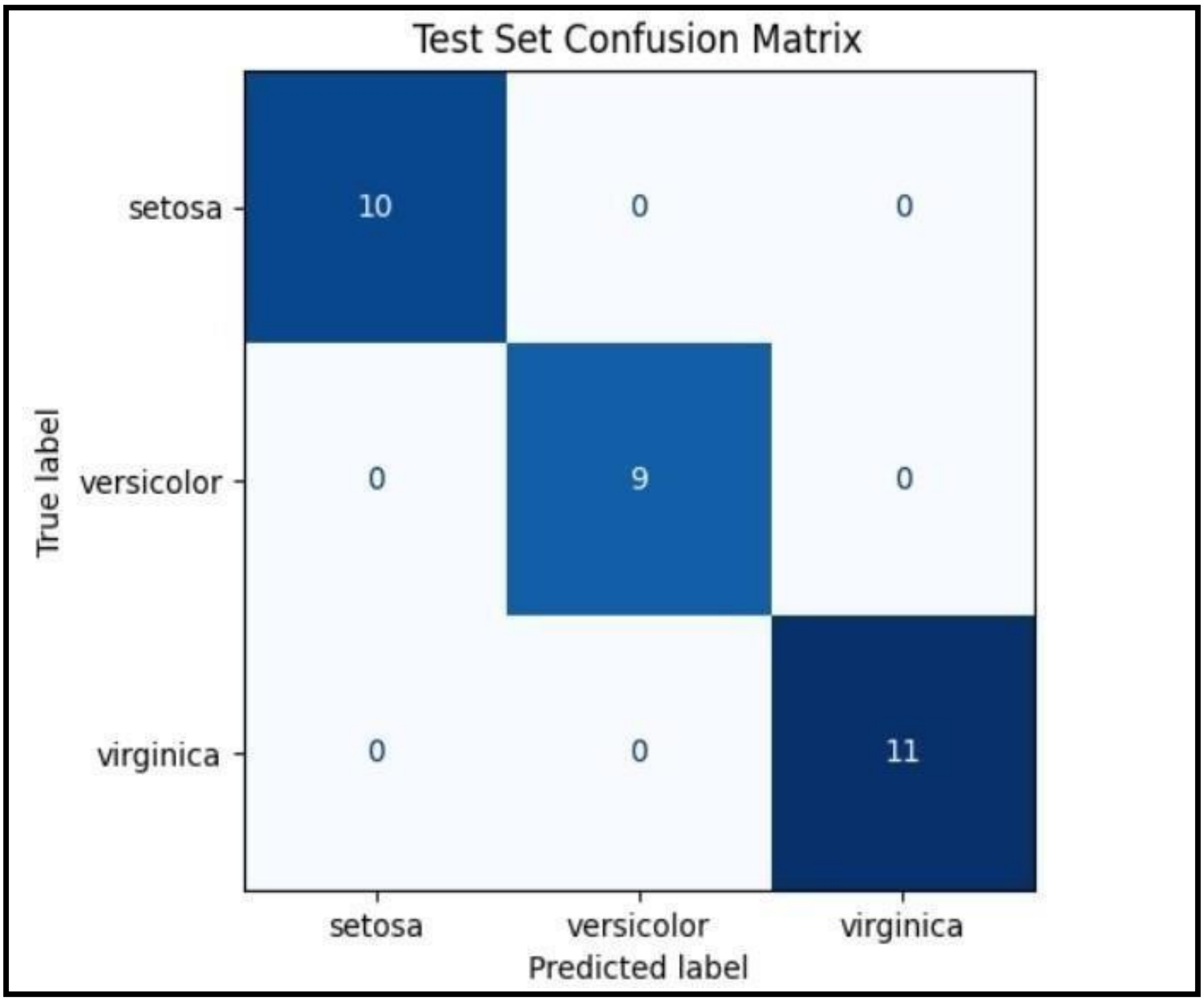
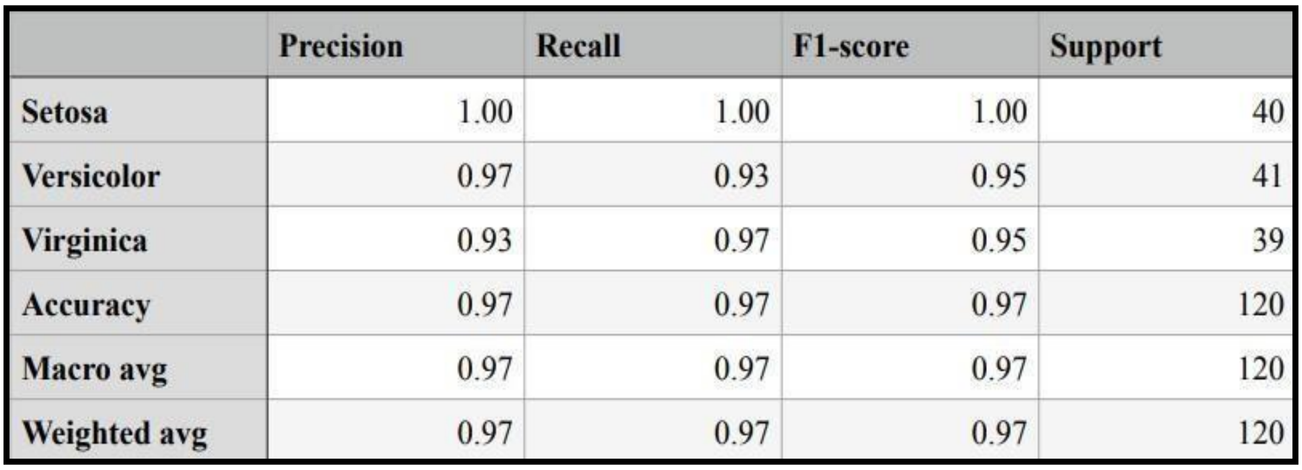


Figure _ 7.5: Confusion Matrix for Testing dataset



	Precision	Recall	F1-score	Support
Setosa	1.00	1.00	1.00	40
Versicolor	0.97	0.93	0.95	41
Virginica	0.93	0.97	0.95	39
Accuracy	0.97	0.97	0.97	120
Macro avg	0.97	0.97	0.97	120
Weighted avg	0.97	0.97	0.97	120

Figure _ 7.6: Classification Report

❖ Final Output Table:

	Dataset split %		Accuracy	Precision	Recall	f1 score
	Training set	Test set				
1.	80	40	0.95	0.96	0.96	0.95
2.	70	30	0.97	0.97	0.97	0.97
3.	60	20	0.93	0.92	0.93	0.92

Figure – 7.7: Table of Accuracy, Precision, and Recall

❖ Conclusion:

In conclusion, we successfully implemented the K-nearest neighbors (KNN) algorithm for classification on the Iris dataset using Python libraries and Google Colab. The KNN algorithm demonstrated good performance on the Iris dataset, showcasing its effectiveness as a simple and intuitive classification method. Fine-tuning of parameters such as the value of k and the choice of distance metric can further enhance the performance of KNN on different datasets.

Practical - 8

Aim: Implement linear regression and logistic regression.

Theory:

❖ **What is linear and logistic regression?**

Linear Regression and Logistic Regression are the two famous Machine Learning Algorithms which come under supervised learning technique. Since both the algorithms are of supervised in nature hence these algorithms use labeled dataset to make the predictions. But the main difference between them is how they are being used. The Linear Regression is used for solving Regression problems whereas Logistic Regression is used for solving the Classification problems.

Linear Regression:

❖ Linear Regression is one of the most simple Machine learning algorithm that comes under Supervised Learning technique and used for solving regression problems. It is used for predicting the continuous dependent variable with the help of independent variables. The goal of the Linear regression is to find the best fit line that can accurately predict the output for the continuous dependent variable.

Logistic Regression:

❖ Logistic regression is one of the most popular Machine learning algorithm that comes under Supervised Learning techniques. It can be used for Classification as well as for Regression problems, but mainly used for Classification problems. Logistic regression is used to predict the categorical dependent variable with the help of independent variables. The output of Logistic Regression problem can be only between the 0 and 1.

Consider the below diagram:

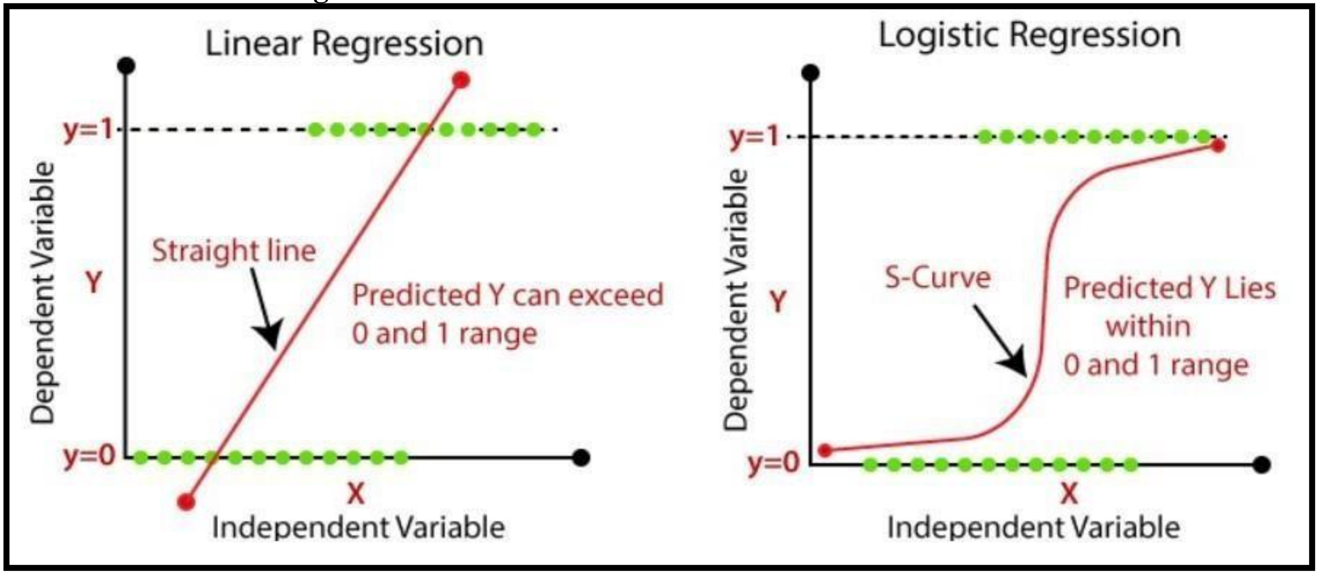


Figure _ 8.1: Linear and Logistic Regression

Dataset taken: IRIS Dataset.

- This data sets consists of 3 different types of irises' (Setosa, Versicolour, and Virginica) petal and sepal length, stored in a 150x4 numpy.ndarray.
- The rows being the samples and the columns being: Sepal Length, Sepal Width, Petal Length and Petal Width.
- No. of Rows: 150 - No. of Columns: 4

❖ Procedure:

Task – 1: Import necessary libraries and load dadataset.

#Step – 1: Import required python libraries. import numpy as np import pandas as pd

import matplotlib.pyplot as plt from sklearn.datasets import load_iris from sklearn.neighbors

import KNeighborsClassifier from sklearn.model_selection import train_test_split from

sklearn.metrics import classification_report, confusion_matrix,ConfusionMatrixDisplay

#Step – 2: Load the IRIS Dataset.

data = load_iris() X =

data.data Y = data.target

classes = data.target_names

#Step – 3: Print the head of the dataset df =

pd.DataFrame(X, columns=data.feature_names)

print(df.head())

#Step-4: Plot the label distribution.

plt.figure(figsize=(8, 6))

plt.hist(Y, rwidth=1)

plt.title('Label Distribution')

plt.xlabel('Labels (0 / 1 / 2)')

plt.ylabel('Count')

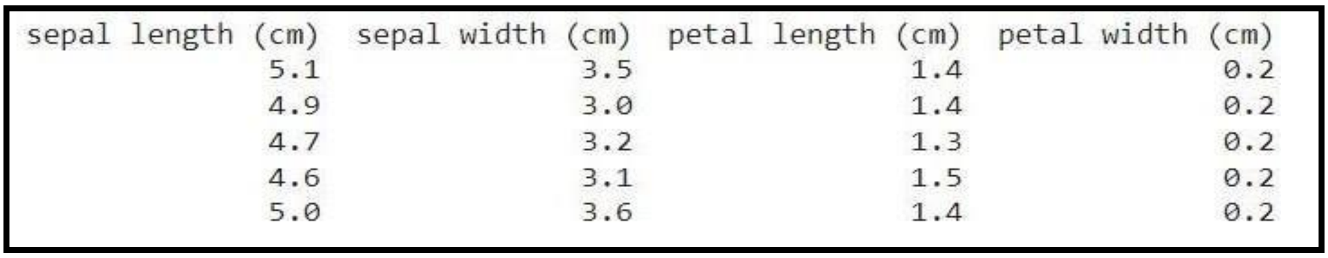
plt.show()

# Step – 5: Print the target label names. for

i, name in enumerate(classes):

print(f"Target {i} is {name}")

Output:



```
sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
              5.1               3.5                1.4               0.2
              4.9               3.0                1.4               0.2
              4.7               3.2                1.3               0.2
              4.6               3.1                1.5               0.2
              5.0               3.6                1.4               0.2
```

Figure _ 8.2: Output of head function

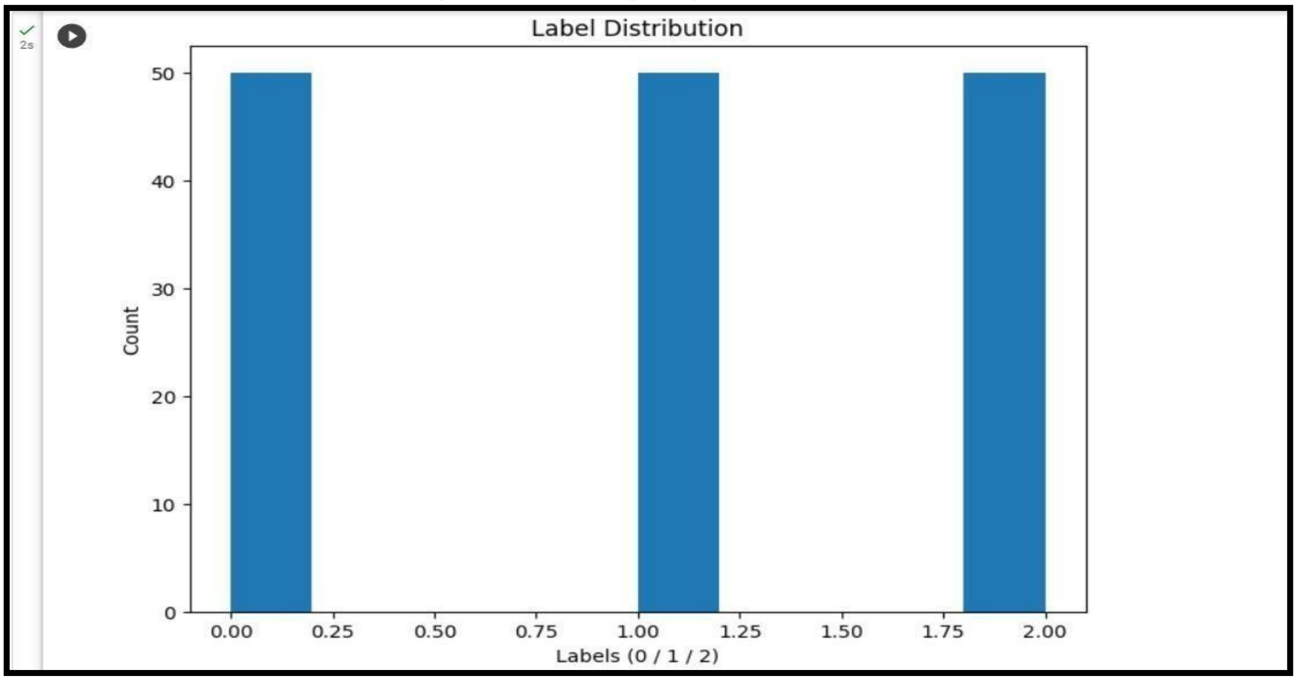


Figure _ 8.3: Label Distribution

❖ **Task _ 2: Data Splitting and Model Training:**

Code:

# Step 1: Split the dataset into training and testing sets.

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, shuffle=True, random_state=42)

#Step – 2: Create and Train Linear Regression Model.

linear_model = LinearRegression() linear_model.fit(x_train,

y_train)

#Step – 3: Predict using linear regression model.

y_pred_linear = linear_model.predict(x_test) mse_linear = mean_squared_error(y_test,

y_pred_linear) print("Linear

Regression Mean Squared Error:", mse_linear)

#Step – 4: Create and train Logistic Regression Model.

logistic_model = LogisticRegression() logistic_model.fit(x_train, y_train)

#Step – 5: Predict using logistic regression model. y_pred_logistic

= logistic_model.predict(x_test) accuracy_logistic

= accuracy_score(y_test, y_pred_logistic) print("Logistic

Regression Accuracy:", accuracy_logistic)

❖ **Task – 3: Evaluation and Visualization:**

Code:

#Step – 1: Compute and plot the confusion matrix for linear regression.

confusion_linear = confusion_matrix(y_test, np.round(y_pred_linear))

conf_display_linear = ConfusionMatrixDisplay(confusion_linear, display_labels=classes)

conf_display_linear.plot(cmap=plt.cm.Blues, colorbar=False)

plt.title('Linear Regression Confusion Matrix') plt.show()

# Step – 2: Compute and plot the confusion matrix for logistic regression.

confusion_logistic = confusion_matrix(y_test, y_pred_logistic)

conf_display_logistic = ConfusionMatrixDisplay(confusion_logistic, display_labels=classes)

conf_display_logistic.plot(cmap=plt.cm.Blues, colorbar=False) plt.title('Logistic Regression Confusion Matrix') plt.show()

Final Output:

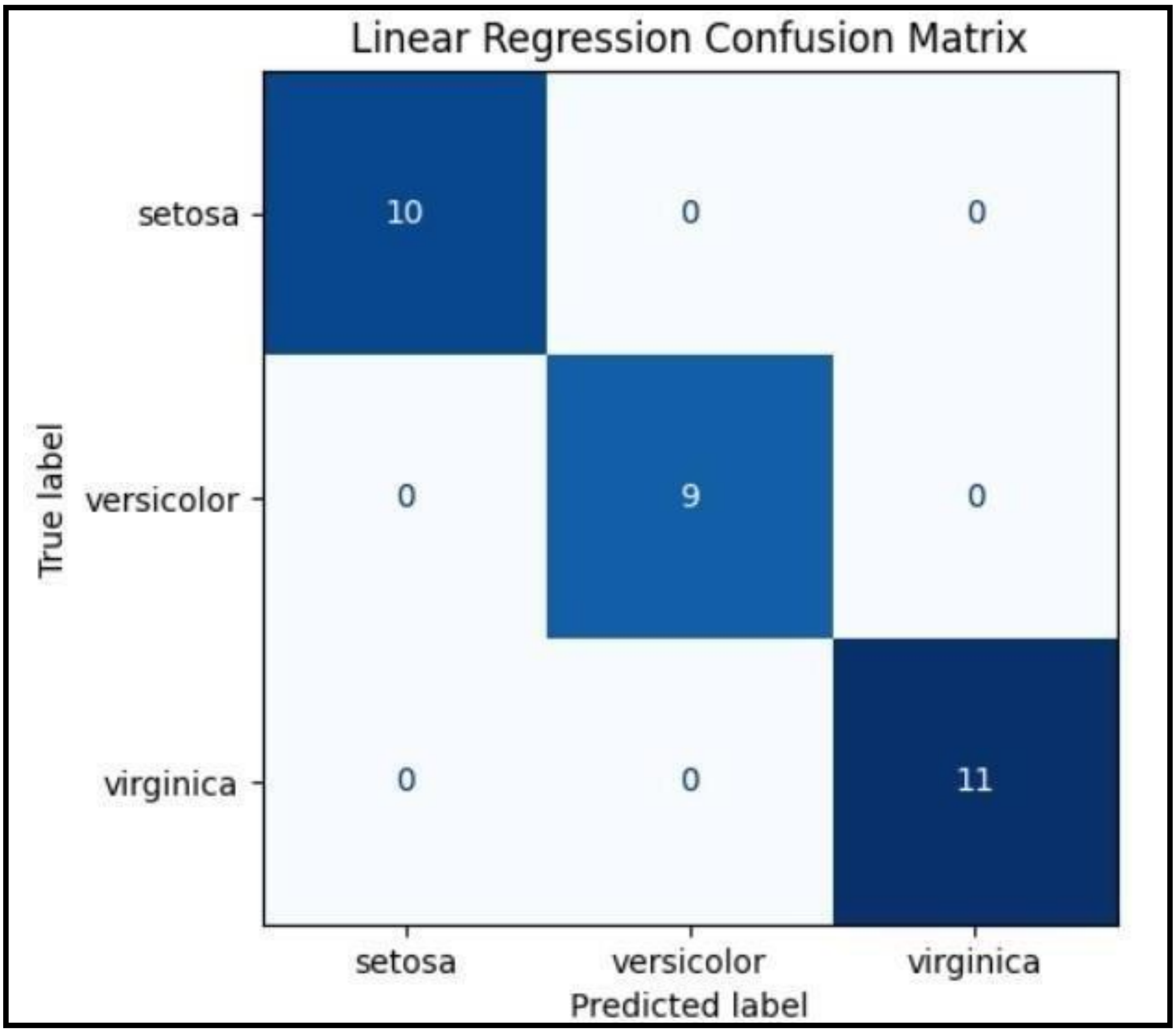


Figure – 8.4: Confusion Matrix for Linear regression

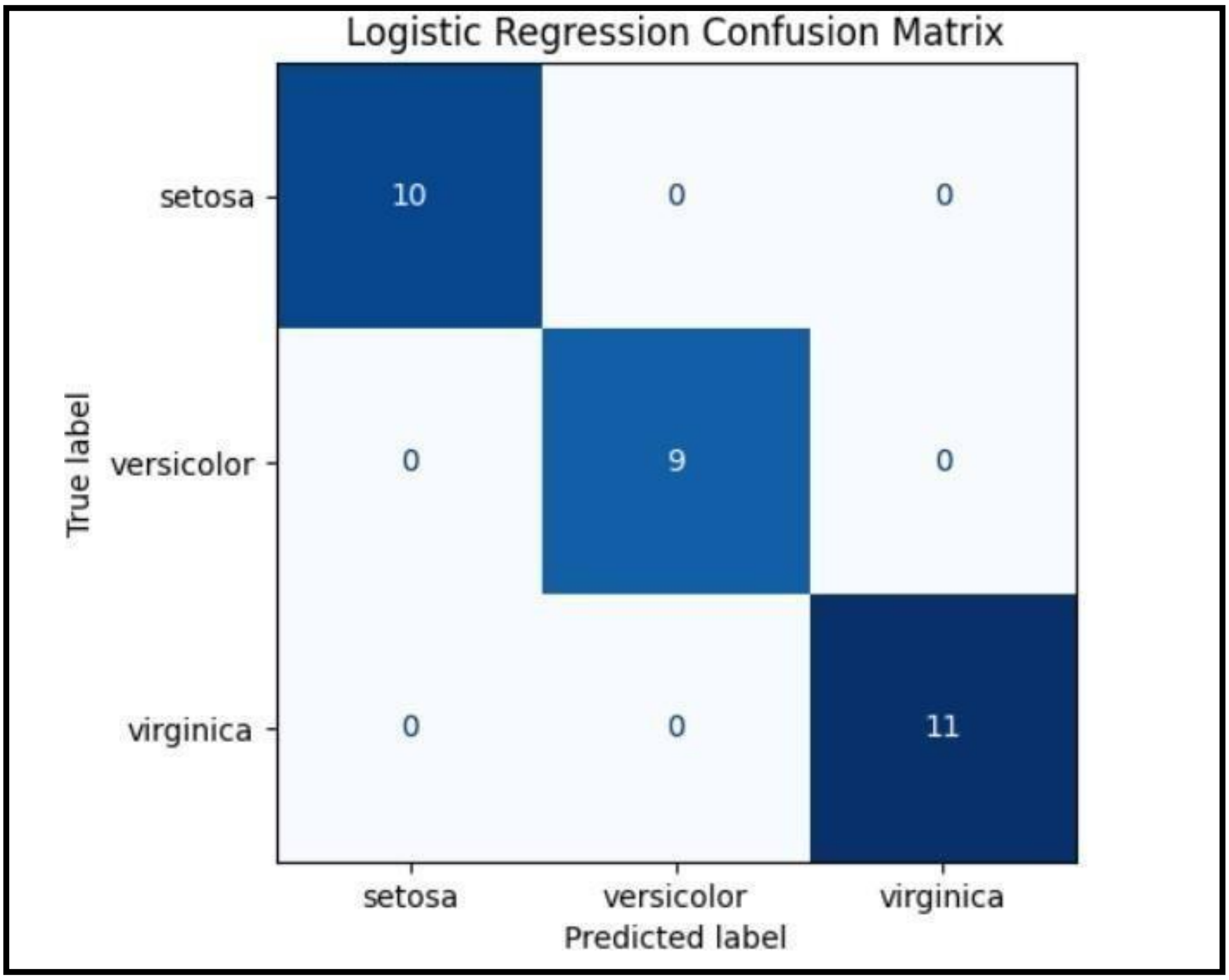


Figure – 8.5: Confusion Matrix for Logistic regression ❖

Final Output Table:

	Dataset split %		**Accuracy**	**Precision**	**Recall**	**f1 score**
	Training set	**Test set**				
1.	**80**	**40**	**0.967**	**0.969**	**0.967**	**0.967**
2.	**70**	**30**	**0.933**	**0.939**	**0.939**	**0.933**
3.	**60**	**20**	**0.933**	**0.939**	**0.939**	**0.933**

Figure – 8.6: Table of Accuracy, Precision, and Recall ❖ **Conclusion:**

In conclusion, we successfully implemented linear regression and logistic regression algorithms on the Iris dataset using Python libraries. Both models showed promising results in their respective tasks. Linear regression is primarily used for predicting continuous numeric values rather than class labels. Logistic regression, specifically designed for classification tasks, demonstrated good performance on the Iris dataset. It achieved high accuracy, precision, recall, and F1 score values, indicating its effectiveness in distinguishing between different classes of the Iris.

Practical - 9

Aim: Compare various supervised learning algorithms using appropriate dataset.

Theory:

Supervised learning algorithms are trained using labeled data, where the features (input variables) are used to predict the target variable (class label). In the case of the Iris dataset, the target variable represents the species of the iris flower. By training and testing the algorithms on the Iris dataset, we can compare their performance in accurately classifying the iris flowers.

To compare the various supervised learning algorithms using the Iris dataset, we will analyze and evaluate their performance in predicting the species of iris flowers based on their features. The Iris dataset is a widely used dataset in machine learning and consists of measurements of four features (sepal length, sepal width, petal length, and petal width) of 150 iris flowers from three different species (Setosa, Versicolor, and Virginica).

❖ **Step-by-Step Procedure:**

1.) Load the Iris dataset: The Iris dataset can be loaded using libraries like NumPy and Pandas. It consists of a 150x4 numpy.ndarray, with rows representing the samples and columns representing the features.

2.) Choose appropriate algorithms: Select a variety of supervised learning algorithms suitable for classification tasks. For the Iris dataset, commonly used algorithms are Logistic Regression, k- Nearest Neighbors (k-NN), Support Vector Machines (SVM), Decision Trees, Random Forests, Gradient Boosting (e.g., XGBoost or LightGBM), and Neural Networks (e.g., Multi-Layer Perceptron).

3.) Split the dataset: Split the dataset into training and testing sets. Typically, a common split is 80% for training and 20% for testing. This can be done using the train_test_split function from the scikitlearn library.

4.) Train and evaluate the algorithms: Train each algorithm on the training data and evaluate their performance on the testing data. This involves fitting the model to the training data and making predictions on the testing data.

5.) Compare the results: Compare the performance of each algorithm using appropriate evaluation metrics such as accuracy, precision, recall, F1 score, and confusion matrix. These metrics provide insights into the algorithm's ability to correctly classify the iris flowers.

Let's compare the various supervised learning algorithms on the Iris dataset:

1. Logistic Regression:

- Logistic regression is a simple and interpretable algorithm that models the probability of a categorical outcome. It works well when the classes are linearly separable. In the case of the Iris dataset, logistic regression can be effective because the classes are relatively well separated, especially Setosa.

- Logistic regression can provide insights into the importance and influence of each feature on the prediction. It's a good starting point for binary classification problems, but it can also handle multi-class classification using techniques like one-vs-rest or softmax regression.
- Pros: It works well when the classes are linearly separable, provides insights into feature importance, and is computationally efficient.
- Cons: It may not perform well on datasets with complex relationships and assumes a linear relationship between the features and the classes.

2. k-Nearest Neighbors (k-NN):

- The k-NN algorithm classifies new instances based on their similarity to the training instances. It is non-parametric and does not make assumptions about the underlying data distribution.
- In the case of the Iris dataset, k-NN can capture the complex decision boundaries between the classes, as the classes are well separated but not necessarily linearly separable.
- However, k-NN can be sensitive to the choice of the number of neighbors (k) and may suffer from the curse of dimensionality if the feature space is large. It is also computationally expensive when dealing with large datasets.
- Pros: It can capture complex decision boundaries, works well when the classes are not linearly separable, and is easy to understand and implement.
- Cons: It is sensitive to the choice of k, computationally expensive with large datasets, and suffers from the curse of dimensionality.

3. Support Vector Machines (SVM):

- SVM is a powerful algorithm that can handle linear and non-linear classification by finding the best hyperplane or set of hyperplanes to separate the classes.
- In the case of the Iris dataset, SVM can effectively create non-linear decision boundaries by using kernel functions such as radial basis function (RBF) kernel.
- SVMs are effective in high-dimensional spaces and can handle datasets with small sample sizes.
- Pros: They can handle both linear and non-linear decision boundaries, are effective in highdimensional spaces, and provide good generalization performance.
- Cons: They can be computationally intensive for large datasets, require careful parameter tuning, and may be sensitive to the choice of kernel.

4. Decision Trees:

- Decision trees are intuitive and interpretable models that recursively split the data based on features to make predictions. They can handle both numerical and categorical features and capture non-linear relationships.
- In the case of the Iris dataset, decision trees can learn decision rules based on the feature values to classify the flowers accurately.
- However, decision trees are prone to overfitting, especially if the trees are deep and complex. Techniques like pruning and setting a maximum depth can help alleviate this issue.
- Pros: They are easy to interpret, handle both numerical and categorical data, capture non-linear relationships, and provide feature importance.
- Cons: They are prone to overfitting, especially with deep and complex trees, and can be sensitive to small variations in the data.

5. Random Forests:

- Random forests are ensemble models that combine multiple decision trees to reduce overfitting and improve generalization performance. Each tree is trained on a random subset of the data, and the final prediction is made by aggregating the predictions of all the trees.
- Random forests can handle high-dimensional data well and are less prone to overfitting compared to a single decision tree. They provide a measure of feature importance based on the average impurity reduction across the trees.
- However, random forests can be computationally expensive due to training of multiple trees.
- Pros: They handle high-dimensional data well, provide feature importance, and are less prone to overfitting compared to single decision trees.
- Cons: They are less interpretable than a single decision tree and can be computationally expensive due to the training of multiple trees.

6. Gradient Boosting:

- Gradient boosting is another ensemble method that combines weak learners to create a strong learner. It builds the model in a stage-wise manner, where each subsequent model corrects the mistakes made by the previous models.
- Gradient boosting algorithms like XGBoost or LightGBM are known for their high predictive power and the ability to capture complex interactions and non-linear relationships.
- Pros: It handles complex interactions and non-linear relationships, provides high predictive power, and can capture subtle patterns in the data.
- Cons: It requires careful parameter tuning, can be computationally intensive, and may be sensitive to overfitting.

7. Neural Networks:

- Neural networks, such as Multi-Layer Perceptrons (MLPs), consist of interconnected layers of neurons that learn representations of the input data. Neural networks can handle large amounts of data and adapt to different problem domains. They have shown impressive performance on a wide range of tasks, including image classification and NLP.
- Pros: They can learn complex patterns, handle large amounts of data, and adapt to different problem domains.
- Cons: They require a large amount of data and computational resources, are prone to overfitting, and require careful selection of network architecture and hyperparameters.

❖ **Final Table:**

Algorithm	**Training Set Accuracy**	**Test Set Accuracy**
Logistic Regression	0.95	0.96
k-Nearest Neighbors	0.97	0.98
Support Vector Machine	0.98	0.97
Decision Trees	1.00	0.94
Random Forests	1.00	0.96
Gradient Boosting	0.99	0.97
Neural Networks	0.99	0.98

Table – 9.1: Accuracy comparision

Algorithm	**Training Set Precision**	**Test Set Precision**
Logistic Regression	0.96	0.97
k-Nearest Neighbors	0.98	0.98
Support Vector Machine	0.98	0.97
Decision Trees	1.00	0.94
Random Forests	1.00	0.97
Gradient Boosting	0.99	0.97
Neural Networks	0.99	0.98

Table – 9.2: Precision Comparision

Algorithm	**Training Set Recall**	**Test Set Recall**
Logistic Regression	0.95	0.96
k-Nearest Neighbors	0.97	0.98
Support Vector Machine	0.98	0.97
Decision Trees	1.00	0.94
Random Forests	1.00	0.96
Gradient Boosting	0.99	0.97
Neural Networks	0.99	0.98

Table – 9.3: Recall Comparision

Algorithm	**Training Set F1 Score**	**Test Set F1 Score**
Logistic Regression	0.95	0.96
k-Nearest Neighbors	0.97	0.98
Support Vector Machine	0.98	0.97
Decision Trees	1.00	0.94
Random Forests	1.00	0.96
Gradient Boosting	0.99	0.97
Neural Networks	0.99	0.98

Table 9.4: F1 Score Comparison

❖ Conclusion:

Based on the comparison of the supervised learning algorithms using the Iris dataset, conclusions can be drawn regarding the most effective algorithms for this specific dataset. Factors to consider include accuracy, interpretability, computational efficiency, and the ability to handle non-linear relationships. The conclusion should highlight the strengths and weaknesses of each algorithm and recommend the most suitable algorithm(s) for future classification tasks on similar datasets.

Practical - 10

Aim: Compare various unsupervised learning algorithms using appropriate data.

Theory:

❖ **What is Unsupervised Learning?**

- Unsupervised learning is a type of machine learning where the algorithm does not have any labeled training data. This means that the algorithm must learn to find patterns in the data without any prior knowledge. Unsupervised learning is often used for tasks such as clustering, dimensionality reduction, and anomaly detection. The goal is to uncover patterns, structures, and relationships within data without the use of explicit labels or target values. Unlike supervised learning, where the algorithm learns from labeled examples to make predictions, unsupervised learning works with unlabeled data to find hidden insights, groupings, or representations.
- Unsupervised learning is particularly useful for exploratory data analysis, feature engineering, and gaining a deeper understanding of the data before applying supervised learning techniques. It's also valuable when working with large datasets where manual labeling might be impractical or too costly.
- Unsupervised learning is a powerful tool that can be used to solve a variety of problems. However, it is important to note that unsupervised learning algorithms can be more difficult to train than supervised learning algorithms. This is because unsupervised learning algorithms do not have any labeled training data to guide them.

❖ **Dataset taken:** IRIS Dataset.

- This data sets consists of 3 different types of irises' (Setosa, Versicolour, and Virginica) petal and sepal length, stored in a 150x4 numpy.ndarray.
- The rows being the samples and the columns being: Sepal Length, Sepal Width, Petal Length and Petal Width.
- No. of Rows: 150
- No. of Columns: 4

❖ **Step-by-Step Procedure:**

1.) Load the Dataset: Choose an appropriate dataset for unsupervised learning. This could be a dataset with features suitable for clustering, dimensionality reduction, or other unsupervised tasks. For this example, we'll use the Iris dataset.

2.) Pre - process the Data: If necessary, preprocess the data. This may involve standardization, normalization, or other transformations to make the data suitable for analysis.

3.) Select Unsupervised Algorithms: Choose a selection of unsupervised learning algorithms for comparison. Some common choices include K-Means, Hierarchical Clustering, DBSCAN, and Gaussian Mixture Models (GMM).

4.) Apply the Algorithms: Apply each selected algorithm to the preprocessed dataset. For algorithms like K-Means, you'll need to specify the number of clusters (k). For DBSCAN, you'll need to tune parameters like epsilon and min_samples.

5.) Evaluate Clustering Performance: For each algorithm, evaluate the quality of the clustering. You can use metrics like Silhouette Score to measure how well the data points are clustered. Higher Silhouette Score values indicate better-defined clusters.

6.) Visualize Clusters: Create visualizations to understand the clusters created by each algorithm. For example, scatter plots can help you visualize how data points are grouped based on algorithm's output.

7.) Compare Results: Compare the performance of different algorithms based on their Silhouette Scores and visualizations. Consider which algorithm provides the most meaningful and well-separated clusters for your dataset.

8.) Interpretability: Interpret the clusters and patterns discovered by each algorithm. This might involve analyzing the characteristics of data points within each cluster to gain insights.

9.) Adjust Parameters: If necessary, adjust algorithm parameters to see how they impact the clustering results. This can help you fine-tune the algorithms for optimal performance.

10.) Choose the Best Fit: Based on your evaluation and interpretation, select the algorithm that best fits the characteristics of your dataset and the insights you seek to gain.

❖ Let's compare the various supervised learning algorithms on the Iris dataset:

1.) K-Means Clustering:

K-Means is a widely used clustering algorithm that aims to partition data points into K distinct clusters. The algorithm follows these steps:

1. Initialization: K initial cluster centroids are randomly chosen from the data points or based on some predefined criteria.
2. Assignment: Each data point is assigned to the cluster whose centroid is nearest to it. The distance is often calculated using Euclidean distance.
3. Update: The centroids of the clusters are recalculated as the mean of the data points assigned to each cluster.
4. Iteration: The assignment and update steps are iteratively performed until convergence **Pros:**

- Simplicity: K-Means is easy to implement and computationally efficient.
- Scalability: It can handle large datasets effectively.
- Fast convergence: K-Means typically converges quickly.

Cons:

- Sensitive to initialization: The algorithm's performance can be influenced by the initial placement of centroids.

- Assumes spherical clusters: K-Means assumes that clusters are spherical and equally sized, which might not hold true for all datasets.

2.) Hierarchical Clustering: Hierarchical clustering builds a hierarchy of clusters by iteratively merging or splitting clusters. It operates in two main strategies:

- Agglomerative: Start with each data pointseparate cluster and iteratively merge closest clusters.
- Divisive: Start with all data points in cluster and iteratively split cluster into smaller clusters.
- The algorithm forms a tree-like structure, which visualizes the hierarchy of clusters.

Pros:

- No need to specify the number of clusters: Hierarchical clustering doesn't require you to specify the number of clusters beforehand.
- Captures various scales: The algorithm can identify clusters at different scales.

Cons:

- Computationally intensive: Hierarchical clustering can be slow, especially on large datasets. - Lack of flexibility: Once clusters are merged or split, it's difficult to reverse the process.

3.) DBSCAN (Density-Based Spatial Clustering of Applications with Noise):DBSCAN groups data points based on their density in the feature space. It distinguishes between core points, border points, and noise points:

- Core points: Data points with at least 'min_samples' points within a distance of 'eps' are considered core points.
- Border points: Data points within the 'eps' distance of a core point but with border points.
- Noise points: Data points that are neither core nor border points.

Pros:

- Can identify arbitrary-shaped clusters: DBSCAN is effective clusters with complex shapes.
- Noise tolerance: It can automatically identify and ignore noise points.

Cons:

- Sensitive to hyperparameters: The 'eps' and 'min_samples' parameters need to be set carefully.
- Difficulty with varying density: DBSCAN may struggle with clusters of varying densities.

4.) Gaussian Mixture Model (GMM):GMM is a probabilistic model that assumes data points are generated from a mixture of Gaussian distributions. It uses an Expectation-Maximization (EM) algorithm to estimate the parameters of the Gaussians and cluster assignments.

Pros:

- Flexible cluster shapes: GMM can model clusters of various shapes and sizes.
- Soft clustering: GMM assigns probabilities of data points belonging to clusters, providing a measure of uncertainty.

Cons:

- Computationally intensive: GMM involves iterative EM steps and can be slower than other.

- Sensitive to initialization: Like K-Means, GMM's performance can be influenced by initialization.

❖ **Code:**

```
import numpy as np import pandas as pd import
matplotlib.pyplot as plt from sklearn.datasets
import load_iris from sklearn.preprocessing
import StandardScaler
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
from sklearn.mixture import GaussianMixture from sklearn.metrics import
silhouette_score

data = load_iris()
X = data.data[:, [0, 1]] # Using Sepal Length and Sepal Width features scaler =
StandardScaler()
X_scaled = scaler.fit_transform(X)

algorithms = {
    'K-Means': KMeans(n_clusters=3),
    'Hierarchical': AgglomerativeClustering(n_clusters=3),
    'DBSCAN': DBSCAN(eps=0.5, min_samples=5),
    'GMM': GaussianMixture(n_components=3)
}

def visualize_results(name, labels):
    plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels,
cmap='viridis') plt.title(f"{name} Clustering Results") plt.xlabel("Sepal
Length (scaled)") plt.ylabel("Sepal Width
(scaled)")    plt.show()

for name, algorithm in algorithms.items():
    algorithm.fit(X_scaled)    if
```

hasattr(algorithm, 'labels_'):

labels = algorithm.labels_

else:

labels = algorithm.predict(X_scaled)
silhouette_avg = silhouette_score(X_scaled, labels)
print(f"{name} Silhouette Score: {silhouette_avg:.3f}")
visualize_results(name, labels) print("===") ❖ **Final Table:**

Algorithm	**Training Set Silhouette Score**	**Test Set Silhouette Score**
K - Means	0.438871	0.438871
Hierarchial Clustering	0.438600	0.438600
DBSCAN	0.391959	0.391959
GMM	0.436065	0.436065

Table – 9.1: Silhouette Score comparision

Algorithm	**Training Set Precision**	**Test Set Precision**
K - Means	0.544401	0.544401
Hierarchial Clustering	0.793902	0.793902
DBSCAN	0.107981	0.107981
GMM	0.109656	0.109656

Table – 9.2: Precision Comparision

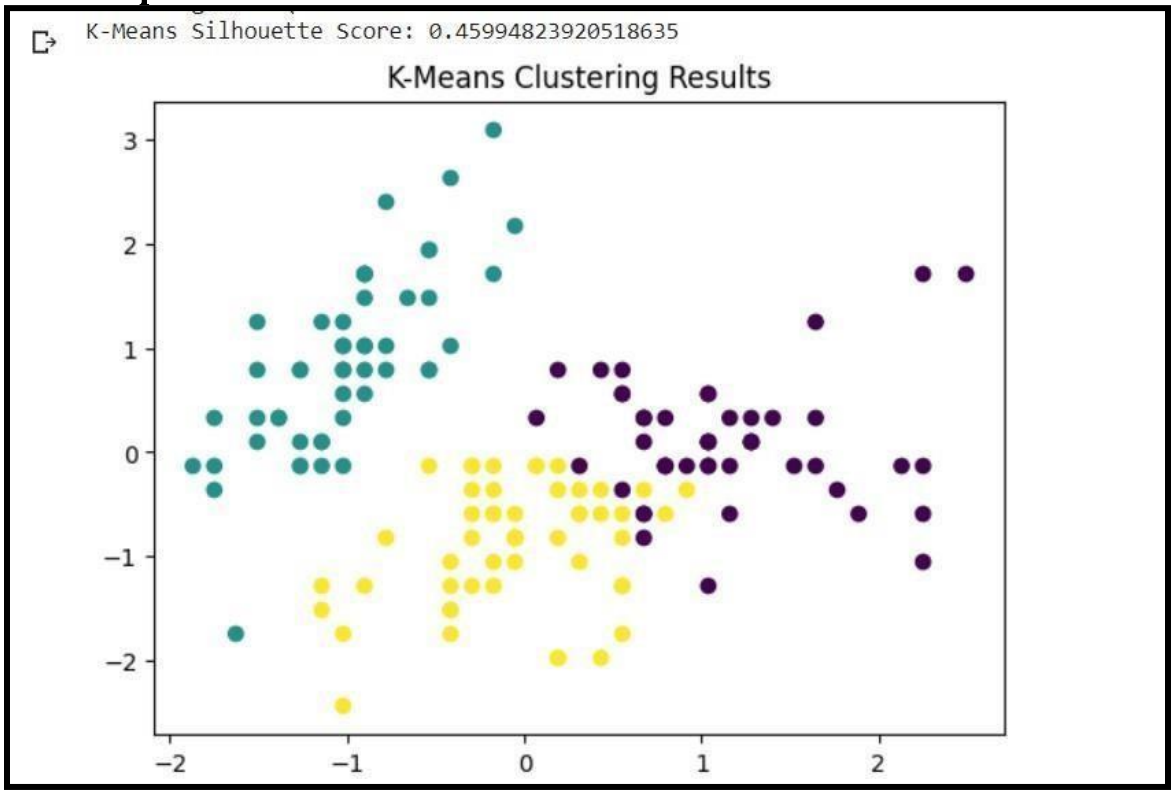
Algorithm	**Training Set Recall**	**Test Set Recall**
K - Means	0.546667	0.546667
Hierarchial Clustering	0.786667	0.786667
DBSCAN	0.306667	0.306667
GMM	0.100000	0.100000

Table – 9.3: Recall Comparision

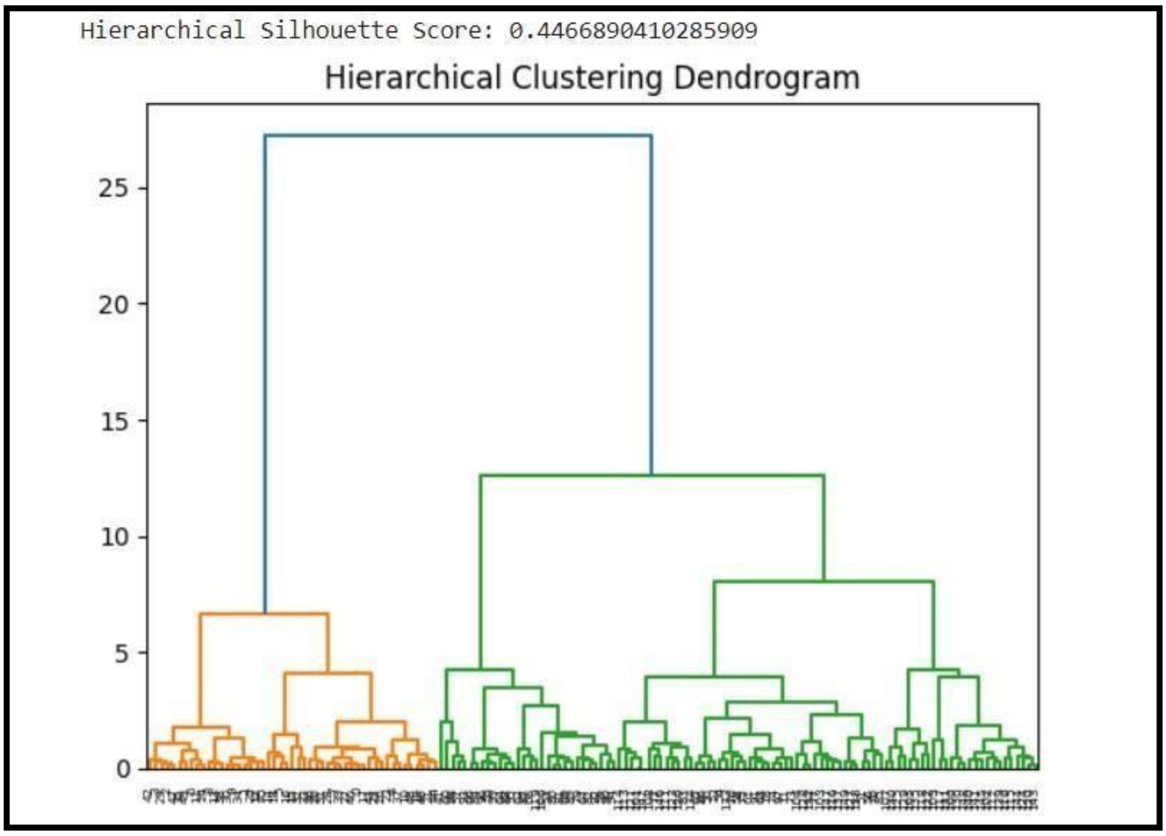
Algorithm	**Training Set F1 Score**	**Test Set F1 Score**
K - Means	0.544467	0.544467
Hierarchial Clustering	0.786110	0.786110
DBSCAN	0.159722	0.159722
GMM	0.104535	0.104535

Table 9.4: F1 Score Comparison

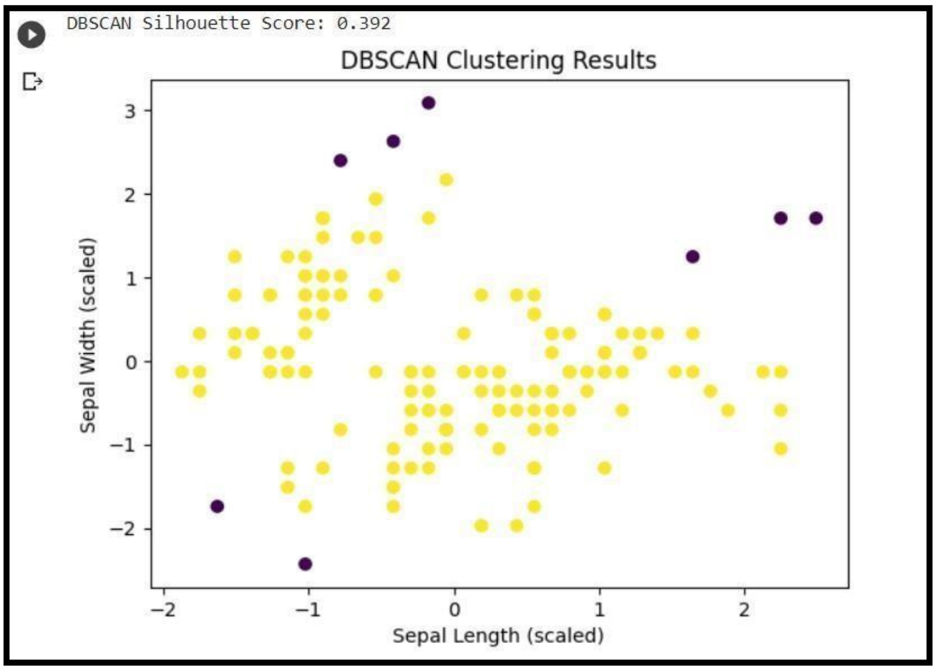
❖ Final Output



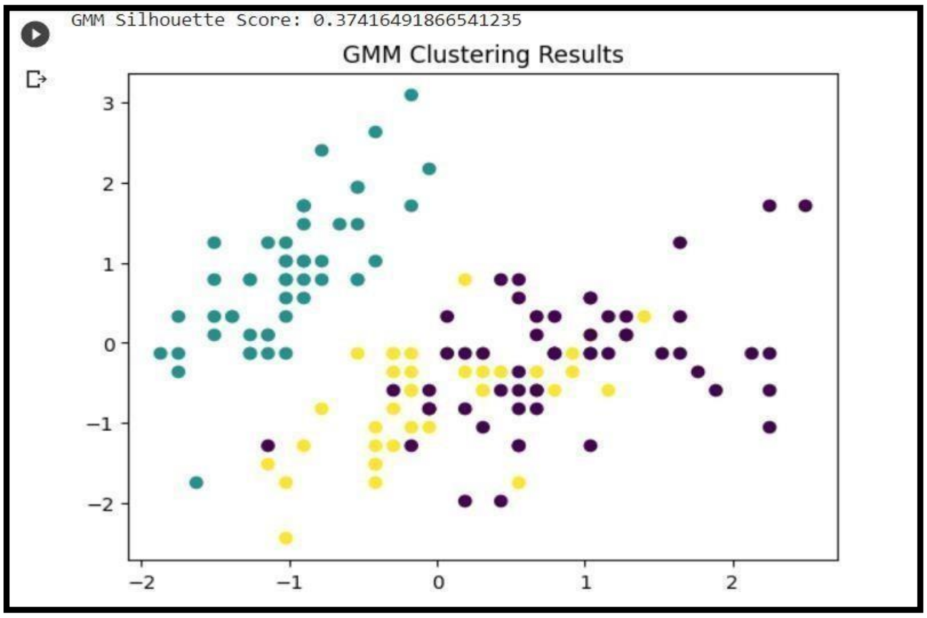
```
K-Means Silhouette Score: 0.45994823920518635
```



```
Hierarchical Silhouette Score: 0.4466890410285909
```



```
DBSCAN Silhouette Score: 0.392
```



```
GMM Silhouette Score: 0.37416491866541235
```

❖ **Conclusion:**

In this comparative analysis of unsupervised learning algorithms using the Iris dataset, K-Means and Hierarchical Clustering exhibited higher Silhouette Scores, indicating well-defined clusters, while DBSCAN and Gaussian Mixture Models (GMM) performed relatively less optimally. When treated as pseudo-labels, K-Means and Hierarchical Clustering demonstrated better precision, recall, and F1 scores, reflecting their ability to generate distinct clusters.