MACHINE LEARNING LAB [U21CM5L2]

III. B.E. – CSM – Vth SEMESTER

A.Y : 2024 – 2025

LABORATORY MANUAL

Compiled By :

Mr. Md Naushad Alam

Assistant Professor

LORDS INSTITUTE OF ENGINEERING & TECHNOLOGY

UGC AUTONOMOUS

Estd: 2003 | Approved by AICTE | Affiliated to OU |

Survey No. 32, Himayath Sagar, Golconda Post, Near TSPA Junction, Hyderabad, Telangana – 500091.

DEPARTMENT (CSM) VISION & MISSION:

VISION OF THE DEPARTMENT:

To obtain high quality standards in education by utilizing cutting-edge technologies, build an ecosystem that will contribute meaningfully to society by producing world class Engineers in allied fields of Computer Science Engineering

MISSION OF THE DEPARTMENT:

- **DM1:** To Provide intense training to generate knowledge using state of art concepts and technologies related to Computer Science Engineering
- **DM2:**To bring along scholars and students in an interdisciplinary setting to carry on making substantial advances in expanding the potential of computer science applications across various domains.
- **DM3:** To encourage the formation of academic-industry collaborations and societal outreach projects
- **DM4:** To educate students with modern tools to take part fully and ethically in a varied society while also encouraging lifelong learning.

Note: DM: Department Mission

PROGRAM OUTCOMES

Engineering graduates will be able to:

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 analyze 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 practice.

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.

PROGRAM SPECIFIC OUTCOMES:

At the end of 4 years, Computer Science and Engineering graduates at LIET will be able to:

PSO1: Professional Skills: The ability to research, understand and implement computer programs in the areas related to algorithms, system software, multimedia, web design, big data analytics, and networking for efficient analysis and design of computer-based systems of varying complexity.

PSO2: Problem-Solving Skills: The ability to apply standard practices and strategies in software service management using open-ended programming environments with agility to deliver a quality service for business success.

GENERAL LABORATORY INSTRUCTIONS:

1. Students are advised to come to the laboratory at least 5 minutes before to starting time, those who come after 5 minutes will not be allowed into the lab.

2. Plan your task properly much before to the commencement, come prepared to the lab with the program / experiment details.

3. Student should enter into the laboratory with:

   a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.

   b. Laboratory Record updated up to the last session experiments.

   c. Formal dress code and Identity card.

4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.

5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.

6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.

7. Computer labs are established with sophisticated and high-end branded systems, which should be utilized properly.

8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviours with the staff and systems etc., will attract severe punishment.

9. Students must take the permission of the faculty in case of any urgency to go out. If anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.

10. Students should SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system /seat is kept properly.

CODE OF CONDUCT FOR THE LABORATORY

• All students must observe the dress code while in the laboratory

• Footwear is NOT allowed

• Foods, drinks and smoking are NOT allowed

• All bags must be left at the indicated place

• The lab timetable must be strictly followed

• Be PUNCTUAL for your laboratory session

• All programs must be completed within the given time

• Noise must be kept to a minimum

• Workspace must be kept clean and tidy at all-time

• All students are liable for any damage to system due to their own negligence

• Students are strictly PROHIBITED from taking out any items from the laboratory

• Report immediately to the lab programmer if any damages to equipment

BEFORE LEAVING LAB:

• Arrange all the equipment and chairs properly.

• Turn off / shut down the systems before leaving.

• Please check the laboratory notice board regularly for updates

Lab In – Charge

Course Code	Course Title	Core/ Elective
U21CM5L2	**MACHINE LEARNING LAB**	**CORE**

Prerequisite	Contact Hours per Week				CIE	SEE	Credits
	L	T	D	P			
Python Programming	-	-	3	3	25	50	1.5

Course Objectives:

This course enable students to:

1. To introduce students to the basic concepts of Data Science and techniques of Machine Learning.
2. To develop skills of using recent machine learning software for solving practical problems.
3. To gain experience of doing independent study and research.
4. Be capable of confidently applying common Machine Learning algorithms in practice and implementing their own
5. Be capable of performing experiments in Machine Learning using real-world data.

Course Outcomes:

On completion of this course, the students are able to:

1. The student must be able to design and implement machine learning solutions to classification, regression problems.
2. Understand complexity of Machine Learning algorithms and their limitations
3. Able to evaluate and interpret the results of the algorithms.
4. Implement Decision trees and various algorithms
5. Implement and Analyze various random forest techniques

1. Implement a program to demonstrate the following

   a) Operation of data types in Python.
   b) Different Arithmetic Operations on numbers in Python.
   c) Create, concatenate and print a string and access substring from a given string.
   d) Append, and remove lists in python.
   e) Demonstrate working with tuples in python.
   f) Demonstrate working with dictionaries in python.

2. Using python write a NumPy program to compute the

   a) Expected Value
   b) Mean
   c) Standard deviation
   d) Variance
   e) Covariance
   f) Covariance Matrix of two given arrays.

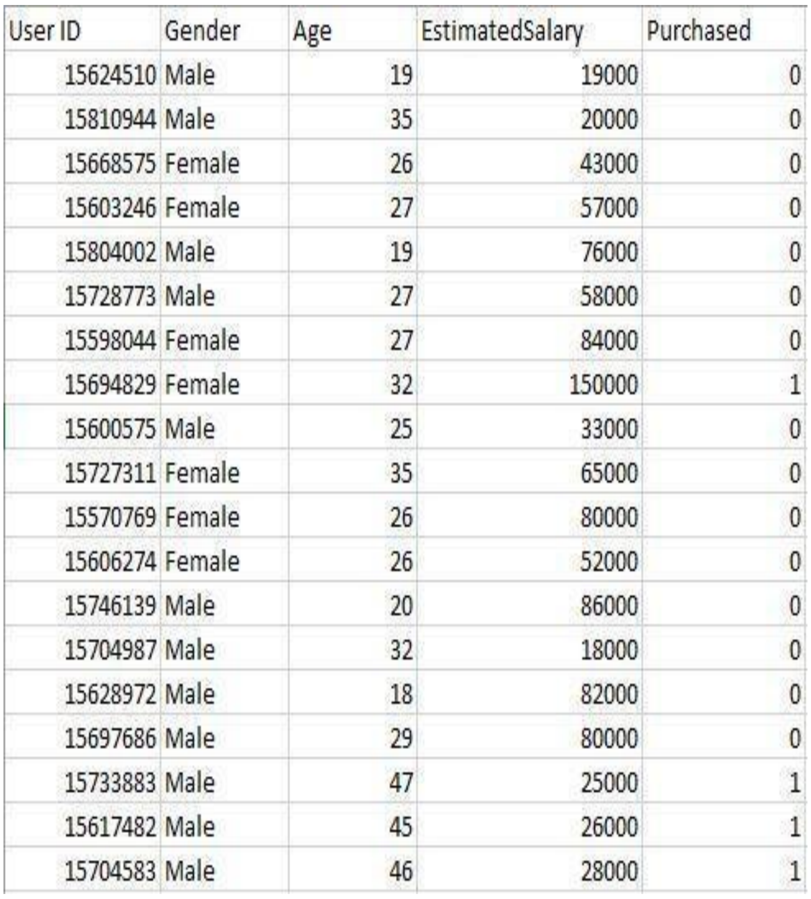
3. For a given set of training data examples stored in a .CSV file, demonstrate Data Preprocessing in Machine learning with the following steps

a) Getting the dataset.
b) Importing libraries.
c) Importing datasets.
d) Finding Missing Data.
e) Encoding Categorical Data.
f) Splitting dataset into training and test set.
g) Feature scaling.

4. Build a linear regression model using python for a particular data set by

a) Splitting Training data and Test data.
b) Evaluate the model (intercept and slope).
c) Visualize the training set and testing set
d) Predicting the test set result
e) Compare actual output values with predicted values

5. The dataset contains information of users from a company's database. It contains information about User ID, Gender, Age, Estimated Salary, and Purchased. Use this dataset for predicting that a user will purchase the company's newly launched product or not by Logistic Regression model.



User ID	Gender	Age	EstimatedSalary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0
15570769	Female	26	80000	0
15606274	Female	26	52000	0
15746139	Male	20	86000	0
15704987	Male	32	18000	0
15628972	Male	18	82000	0
15697686	Male	29	80000	0
15733883	Male	47	25000	1
15617482	Male	45	26000	1
15704583	Male	46	28000	1

6. Implement a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

7. Implement k-nearest neighbour's classification to classify the iris data set using python.

8. Given the following data, which specify classifications for nine combinations of VAR1 and VAR2 predict a classification for a case where VAR1=0.906 and VAR2=0.606, using the result of k-means clustering with 3 means (i.e., 3 centroids)

VAR 1	**VAR2**	**CLASS**
1.713	1.586	0
0.180	1.786	1
0.353	1.240	1
0.940	1.566	0
1.486	0.759	1
1.266	1.106	0
1.540	0.419	1
0.459	1.799	1
0.773	0.186	1

9. Evaluate the metrics for all types of machine learning algorithms using sample data.

10. Implement an algorithm to demonstrate the significance of SVM.

EXPERIMENT 01:

1. Implement a program to demonstrate the following

   a) Operation of data types in Python.
   b) Different Arithmetic Operations on numbers in Python.
   c) Create, concatenate and print a string and access substring from a given string.
   d) Append, and remove lists in python.
   e) Demonstrate working with tuples in python.
   f) Demonstrate working with dictionaries in python.

Aim: To demonstrate basic operations and manipulations using fundamental data types in Python, including numbers, strings, lists, tuples, and dictionaries, as well as performing arithmetic operations and string manipulations.

Objectives:

1. **Understand and operate with various Python data types:**
   - Demonstrate the usage of integers, floats, strings, Booleans, lists, tuples, and dictionaries in Python.
2. **Perform arithmetic operations on numbers:**
   - Showcase basic arithmetic operations like addition, subtraction, multiplication, division, modulus, exponentiation, and integer division.
3. **Create, manipulate, and access strings:**
   - Demonstrate string creation, concatenation, printing, and extracting substrings.
4. **Work with lists:**
   - Show how to append elements to a list and remove elements from a list.
5. **Demonstrate tuple operations:**
   - Illustrate the immutability of tuples by attempting (and failing) to change tuple values, and performing basic tuple operations.
6. **Work with dictionaries:**
   - Show how to create, access, add, and remove key-value pairs in a dictionary.

```python
# a) Operation of data types in Python
x = 10       # int
y = 10.5     # float
z = 3 + 4j   # complex
text = "Hello, Python!"  # string
my_list = [1, 2, 3]      # list
my_tuple = (1, 2, 3)     # tuple
my_dict = {"name": "Alice", "age": 25}  # dict
print("Integer:", x)
print("Float:", y)
print("Complex:", z)
print("String:", text)
print("List:", my_list)
print("Tuple:", my_tuple)
print("Dictionary:", my_dict)
```

<u>OUTPUT:</u>

```
Integer: 10
Float: 10.5
Complex: (3+4j)
String: Hello, Python!
List: [1, 2, 3]
Tuple: (1, 2, 3)
Dictionary: {'name': 'Alice', 'age': 25}
```

```python
# b) Different Arithmetic Operations on numbers in Python

a = 15

b = 4

print("Addition (a + b):", a + b)

print("Subtraction (a - b):", a - b)

print("Multiplication (a * b):", a * b)

print("Division (a / b):", a / b)

print("Floor Division (a // b):", a // b)

print("Modulus (a % b):", a % b)

print("Exponentiation (a ** b):", a ** b)
```

OUTPUT:

```
Addition (a + b): 19

Subtraction (a - b): 11

Multiplication (a * b): 60

Division (a / b): 3.75

Floor Division (a // b): 3

Modulus (a % b): 3

Exponentiation (a ** b): 50625
```

```
# c) Create, concatenate, print a string and access substring
str1 = "Hello"
str2 = "World"
concatenated = str1 + " " + str2  # Concatenation
print("Concatenated String:", concatenated)
print("Substring (first 3 characters):", concatenated[:3])
```

OUTPUT:

```
Concatenated String: Hello World
Substring (first 3 characters): Hel
```

```
# d) Append and remove lists in Python
my_list = [1, 2, 3]
my_list.append(4)  # Append
print("List after append:", my_list)
my_list.remove(2)  # Remove
print("List after removing 2:", my_list)
```

OUTPUT:

```
List after append: [1, 2, 3, 4]
List after removing 2: [1, 3, 4]
```

```
# e) Demonstrate working with tuples in Python

my_tuple = (10, 20, 30)

print("Tuple:", my_tuple)

print("First element of tuple:", my_tuple[0])
```

OUTPUT:

```
Tuple: (10, 20, 30)

First element of tuple: 10
```

```
# f) Demonstrate working with dictionaries in Python

my_dict = {"name": "Alice", "age": 25}

my_dict["city"] = "New York"  # Add new key-value pair

print("Dictionary after adding a key-value pair:", my_dict)

print("Name from dictionary:", my_dict["name"])
```

OUTPUT:

```
Dictionary after adding a key-value pair: {'name': 'Alice', 'age': 25,
'city': 'New York'}

Name from dictionary: Alice
```

EXPERIMENT 02:

2. Using python write a NumPy program to compute the

a) Expected Value
b) Mean
c) Standard deviation
d) Variance
e) Covariance
f) Covariance Matrix of two given arrays.

Aim:

The aim of this program is to compute various statistical measures using Python’s NumPy library. The program will calculate the expected value, mean, standard deviation, variance, covariance, and the covariance matrix of two given arrays.

Objective:

To use NumPy's built-in functions to perform the following statistical calculations on arrays:

1. **Expected Value**: A measure of the central tendency of a random variable.
2. **Mean**: The average of the data points.
3. **Standard Deviation**: Measures the spread of the data points from the mean.
4. **Variance**: Measures how far the data points are spread out.
5. **Covariance**: Shows the relationship between two datasets (how they change together).
6. **Covariance Matrix**: A matrix that shows the covariance between multiple pairs of datasets.

```
# a) Expected Value

import numpy as np

# Define an array

array1 = np.array([1, 2, 3, 4, 5])

# Expected Value (mean for a probability distribution)

expected_value = np.mean(array1)

print("Expected Value of array1:", expected_value)
```

OUTPUT:

```
Expected Value of array1: 3.0
```

```
# b) Mean

import numpy as np

# Define an array

array1 = np.array([1, 2, 3, 4, 5])

# Mean of the array

mean = np.mean(array1)

print("Mean of array1:", mean)
```

OUTPUT:

```
Mean of array1: 3.0
```

```
# c) Standard deviation

import numpy as np

# Define an array

array1 = np.array([1, 2, 3, 4, 5])

# Standard Deviation of the array

std_dev = np.std(array1)

print("Standard Deviation of array1:", std_dev)
```

OUTPUT:

```
Standard Deviation of array1:
1.4142135623730951
```

```
# d) Variance

import numpy as np

# Define an array

array1 = np.array([1, 2, 3, 4, 5])

# Variance of the array

variance = np.var(array1)

print("Variance of array1:", variance)
```

OUTPUT:

```
Variance of array1: 2.0
```

```
# e) Covariance

import numpy as np

# Define two arrays

array1 = np.array([1, 2, 3, 4, 5])

array2 = np.array([5, 4, 3, 2, 1])

# Covariance between the two arrays

covariance = np.cov(array1, array2)[0, 1]  # Extracting the covariance value

print("Covariance between array1 and array2:", covariance)
```

OUTPUT:

```
Covariance between array1 and array2: -2.0
```

```
# f) Covariance Matrix of two given arrays.

import numpy as np

# Define two arrays

array1 = np.array([1, 2, 3, 4, 5])

array2 = np.array([5, 4, 3, 2, 1])

# Covariance Matrix

cov_matrix = np.cov(array1, array2)

print("Covariance Matrix:\n", cov_matrix)
```

OUTPUT:

```
Covariance Matrix:
 [[ 2. -2.]
 [-2.  2.]]
```

EXPERIMENT 03:

3. For a given set of training data examples stored in a .CSV file, demonstrate Data Preprocessing in Machine learning with the following steps

a) Getting the dataset.
b) Importing libraries.
c) Importing datasets.
d) Finding Missing Data.
e) Encoding Categorical Data.
f) Splitting dataset into training and test set.
g) Feature scaling.

Aim:

To demonstrate data pre-processing for a given dataset in Machine Learning using Python. The dataset is stored in a .CSV file, and the pre-processing steps include loading the data, handling missing values, encoding categorical data, splitting the dataset into training and test sets, and applying feature scaling.

Objective:

The objective of this program is to perform the essential pre-processing steps required to prepare the dataset for a machine learning model. The steps include:

1. **Getting the dataset**: Load a CSV dataset into Python.
2. **Importing libraries**: Use relevant libraries for data manipulation and processing.
3. **Importing datasets**: Load the dataset into a Pandas DataFrame.
4. **Finding Missing Data**: Detect and handle missing values in the dataset.
5. **Encoding Categorical Data**: Convert categorical data into numerical values for model compatibility.
6. **Splitting dataset into training and test set**: Divide the dataset into a training set for model training and a test set for evaluation.
7. **Feature Scaling**: Normalize or standardize the features to bring all data points into the same scale.

```
# Step b) Importing Libraries
import pandas as pd  # For handling the dataset
import numpy as np   # For numerical operations
from sklearn.model_selection import train_test_split  # For splitting data
from sklearn.preprocessing import StandardScaler, LabelEncoder,
OneHotEncoder  # For encoding and scaling


# Step a) Getting the Dataset
# Assuming you have a CSV file named 'data.csv'
dataset = pd.read_csv('data.csv')


# Step c) Importing Datasets
print("First 5 rows of the dataset:")
print(dataset.head())  # Display first 5 rows to get an overview


# Step d) Finding Missing Data
print("\nMissing data information:")
print(dataset.isnull().sum())  # Display the count of missing values per
column


# Handling missing data (example: filling with mean for numeric columns)
dataset.fillna(dataset.mean(), inplace=True)
print("\nAfter handling missing data (filling with mean):")
print(dataset.isnull().sum())
```

```
# Step e) Encoding Categorical Data
# Example: Encode a column named 'Category' (if exists)
if 'Category' in dataset.columns:
    labelencoder = LabelEncoder()
    dataset['Category'] = labelencoder.fit_transform(dataset['Category'])
    print("\nAfter encoding categorical data in 'Category':")
    print(dataset.head())

# Step f) Splitting dataset into training and test set
# Assuming that the last column is the dependent variable (target)
X = dataset.iloc[:, :-1].values  # Independent variables (features)
y = dataset.iloc[:, -1].values   # Dependent variable (target)

# Split the dataset: 80% training, 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
print("\nTraining set shape (X_train, y_train):", X_train.shape, y_train.shape)
print("Test set shape (X_test, y_test):", X_test.shape, y_test.shape)

# Step g) Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)  # Fit and transform the training set
X_test = scaler.transform(X_test)        # Transform the test set
print("\nFeature scaling applied to training and test sets.")
```

OUTPUT OF THE DATA PREPROCESSING PROGRAM:

1. Step a) Getting the Dataset: The dataset is loaded from a CSV file named data.csv. This file should be present in the same directory where you are running the code.
2. Step c) Importing Datasets: The first 5 rows of the dataset are displayed to give an overview of the data.

Example:First5rowsofthedataset:

	X1	X2	Category	Y
0	5.1	3.5	A	1
1	4.9	3.0	B	0
2	4.7	3.2	A	1
3	4.6	3.1	B	0
4	5.0	3.6	A	1

3. Step d) Finding Missing Data: The number of missing values per column is displayed. For instance, if there is missing data:

Missing data information:

X1	0
X2	1
Category	0
Y	0

dtype:int64

After handling missing data by filling the numeric columns with the mean value:

After handling missing data (filling with mean):

X1	0
X2	0
Category	0
Y	0

dtype:int64

4. Step e) Encoding Categorical Data: If the dataset contains a categorical column (e.g., Category), the categorical data is encoded into numeric values using LabelEncoder. For example:

	X1	X2	Category	Y
0	5.1	3.5	0	1
1	4.9	3.0	1	0
2	4.7	3.2	0	1
3	4.6	3.1	1	0
4	5.0	3.6	0	1

5. Step f) Splitting the Dataset into Training and Test Set: The dataset is split into 80% training data and 20% test data. The shape of the training and test sets is displayed:

Training set shape (X_train, y_train): (80, 3) (80,)

Test set shape (X_test, y_test): (20, 3) (20,)

6. Step g) Feature Scaling: Feature scaling is applied to the training and test sets using StandardScaler, ensuring that the independent variables are scaled to have a mean of 0 and standard deviation of 1:

Feature scaling applied to training and test sets.

EXPERIMENT 04:

4. Build a linear regression model using python for a particular data set by

a) Splitting Training data and Test data.
b) Evaluate the model (intercept and slope).
c) Visualize the training set and testing set
d) Predicting the test set result
e) Compare actual output values with predicted values

Aim:

The aim of this program is to build a linear regression model using Python on a given dataset. This includes splitting the dataset into training and test sets, evaluating the model parameters (intercept and slope), visualizing the results, predicting values for the test set, and comparing actual and predicted outputs.

Objective:

The objective of this program is to:

1. **Split the Dataset**: Divide the dataset into training and testing subsets.
2. **Evaluate the Model**: Calculate and display the model's intercept and slope.
3. **Visualize Results**: Create plots to visualize the training and testing datasets.
4. **Make Predictions**: Use the trained model to predict outcomes for the test dataset.
5. **Compare Outputs**: Compare the actual values of the test set with the predicted values to assess the model's performance.

```
# Step 1: Import Libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_absolute_error, r2_score


# Step 2: Generate Synthetic Data

np.random.seed(0)  # For reproducibility

X = np.arange(1, 101)  # Independent variable (1 to 100)

Y = 2.5 * X + np.random.normal(0, 10, size=X.shape)  # Dependent variable with
some noise


# Creating a DataFrame

dataset = pd.DataFrame({'X': X, 'Y': Y})


# Step 3: Splitting the Dataset into Training and Test Sets

X = dataset[['X']].values  # Independent variable

y = dataset['Y'].values     # Dependent variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)


# Step 4: Creating the Linear Regression Model

model = LinearRegression()

model.fit(X_train, y_train)  # Fitting the model
```

```
# Step 5: Evaluate the Model (intercept and slope)

intercept = model.intercept_

slope = model.coef_[0]

print("Intercept:", intercept)

print("Slope:", slope)


# Step 6: Visualize the Training Set

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.scatter(X_train, y_train, color='blue', label='Training data')

plt.plot(X_train, model.predict(X_train), color='red', label='Regression line')

plt.title('Training Set')

plt.xlabel('X')

plt.ylabel('Y')

plt.legend()


# Step 7: Visualize the Testing Set

plt.subplot(1, 2, 2)

plt.scatter(X_test, y_test, color='green', label='Testing data')

plt.plot(X_train, model.predict(X_train), color='red', label='Regression line
(training)')

plt.title('Testing Set')

plt.xlabel('X')

plt.ylabel('Y')

plt.legend()

plt.show()
```

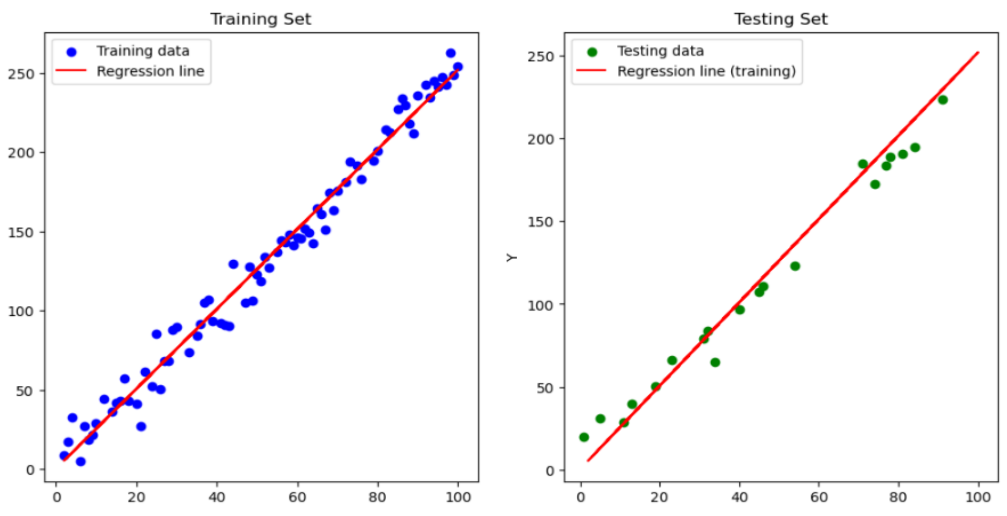
```
# Step 8: Predicting the Test Set Result
y_pred = model.predict(X_test)

# Step 9: Compare Actual Output Values with Predicted Values
results = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print("\nComparison of Actual and Predicted values:")
print(results)

# Additional Metrics
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean Absolute Error:", mae)
print("R-squared:", r2)
```

OUTPUT:

Intercept: 0.3098988330752377

Slope: 2.516245690442155

3 Comparison of Actual vs Predicted Values:



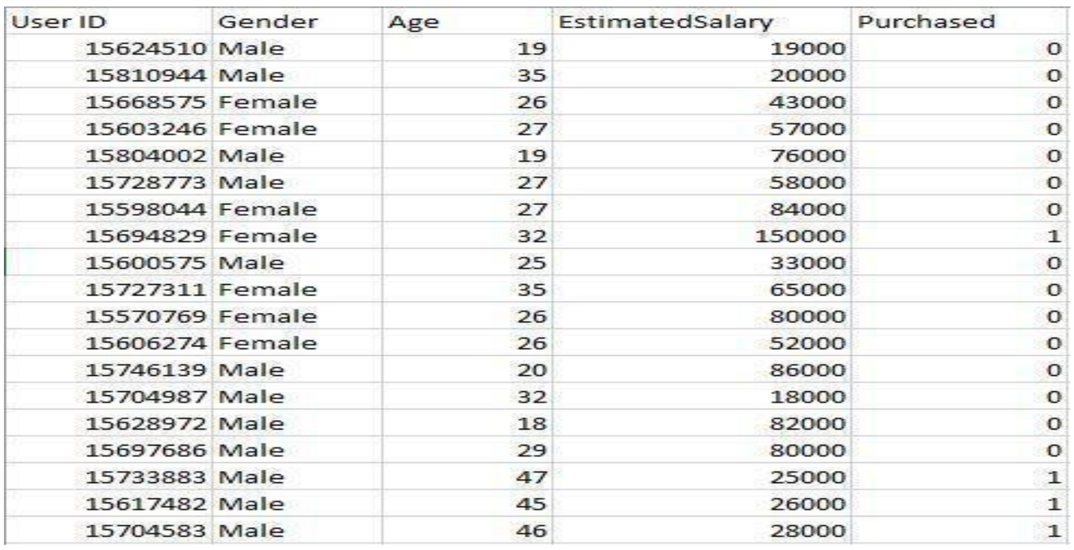
	ACTUAL	PREDICTED
0	194.637563	211.674537
1	123.193678	136.187166
2	184.790906	178.963343
3	110.619257	116.057201
4	107.403478	113.540955

Mean Absolute Error: 9.023956092832567

R-squared: 0.971904686888562

EXPERIMENT 05:

5. The dataset contains information of users from a company's database. It contains information about User ID, Gender, Age, Estimated Salary, and Purchased. Use this dataset for predicting that a user will purchase the company's newly launched product or not by Logistic Regression model.



User ID	Gender	Age	EstimatedSalary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0
15570769	Female	26	80000	0
15606274	Female	26	52000	0
15746139	Male	20	86000	0
15704987	Male	32	18000	0
15628972	Male	18	82000	0
15697686	Male	29	80000	0
15733883	Male	47	25000	1
15617482	Male	45	26000	1
15704583	Male	46	28000	1

Aim:

The aim is to predict whether a user will purchase a newly launched product or not based on their **Age**, **Estimated Salary**, and **Gender** using a **Logistic Regression** model.

Objective:

1. **Data Collection**: Collect user information, including User ID, Gender, Age, Estimated Salary, and Purchase status (whether they purchased the product or not).
2. **Data Preprocessing**:
   - o Encode categorical features (such as Gender) into numerical values.
   - o Normalize or standardize numerical features like Age and Estimated Salary for better model performance.
3. **Model Training**: Train a Logistic Regression model using the features (Age, Gender, and Estimated Salary) to predict the target variable (whether the user purchases the product).
4. **Evaluation**: Evaluate the performance of the model using metrics such as accuracy, confusion matrix, etc., to determine how well the model predicts whether a user will purchase the product.

The objective is to use this predictive model to help the company understand the purchasing behavior of users and make informed marketing and product strategies based on user demographics and income.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score


# Sample dataset (replace with your actual data)
data = {
    'User ID': [15624510, 15810944, 15668575, 15603246, 15804002, 15728773, 15598044, 15694829, 15600575, 15727311, 15570769, 15606274, 15746139, 15704987, 15628972, 15697686, 15733883, 15617482, 15704583],
    'Gender': ['Male', 'Male', 'Female', 'Female', 'Male', 'Male', 'Female', 'Female', 'Male', 'Female', 'Female', 'Female', 'Male', 'Male', 'Male', 'Male', 'Male', 'Male', 'Male'],
    'Age': [19, 35, 26, 27, 19, 27, 27, 32, 25, 35, 26, 26, 20, 32, 18, 29, 47, 45, 46],
    'EstimatedSalary': [19000, 20000, 43000, 57000, 76000, 58000, 84000, 150000, 33000, 65000, 80000, 52000, 86000, 18000,82000, 80000, 25000, 26000, 28000],
    'Purchased': [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1]
}
# Load dataset into a DataFrame
df = pd.DataFrame(data)

# Features (X) and Target (y)
X = df[['Gender', 'Age', 'EstimatedSalary']]
y = df['Purchased']

# Convert categorical data (Gender) into numerical data
le = LabelEncoder()
X['Gender'] = le.fit_transform(X['Gender'])  # Male: 1, Female: 0
```

```
# 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=0)

# Feature scaling

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

# Create Logistic Regression model

classifier = LogisticRegression(random_state=0)

classifier.fit(X_train, y_train)

# Predicting the test set results

y_pred = classifier.predict(X_test)

# Model evaluation

cm = confusion_matrix(y_test, y_pred)

accuracy = accuracy_score(y_test, y_pred)

# Display results

print("Confusion Matrix:")

print(cm)

print(f"Accuracy: {accuracy * 100:.2f}%")
```

OUTPUT:

```
Confusion Matrix:
[[3 0]
 [0 1]]
Accuracy: 100.00%
```

EXPERIMENT 06:

6. Implement a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Aim:

The aim is to implement a **Decision Tree** using the **ID3 algorithm** to classify data, and apply this knowledge to classify a new sample based on the decision tree built from the given dataset.

Objective:

1. **Understand the ID3 Algorithm**:
   - The ID3 algorithm (Iterative Dichotomiser 3) uses **information gain** to select the best attribute at each node in the tree.
   - It recursively partitions the data based on attributes and builds the decision tree until all data is classified.
2. **Dataset Selection**:
   - Choose an appropriate dataset (e.g., with categorical features) for classification, such as predicting whether a person will play a game based on weather conditions (sunny, windy, etc.).
   - Preprocess the data if necessary (e.g., convert any continuous variables to categorical values).
3. **Tree Construction**:
   - Build the decision tree using the ID3 algorithm by calculating **entropy** and **information gain** at each step.
   - Split the data based on the attribute that maximizes information gain.
4. **Classification**:
   - Use the trained decision tree to classify new samples.
   - For any new input, traverse the decision tree to make a prediction based on learned patterns.
5. **Evaluation**:
   - Evaluate the performance of the decision tree using metrics such as accuracy, and validate it with test samples.
   - Visualize the decision tree for better understanding.

ITERATIVE DICHOTOMISER 3 (ID3):

The **ID3 (Iterative Dichotomiser 3)** algorithm is a popular decision tree algorithm used for classification tasks. It was developed by **Ross Quinlan** in 1986 and is based on the concept of **information gain** and **entropy**. The goal of ID3 is to build a decision tree that can classify data by splitting it based on the most informative attributes (features).

Steps of the ID3 Algorithm:

1. **Select the Best Attribute (Feature) to Split**: ID3 selects the attribute that provides the **maximum information gain** to split the dataset. Information gain is a measure of how well an attribute separates the data into classes. To compute information gain, the algorithm uses the concept of **entropy**, which measures the uncertainty or disorder in the data.
2. **Split the Data**: Once the attribute with the highest information gain is selected, the dataset is split into subsets based on the values of that attribute. Each subset should be more "pure" (less uncertain) than the original set.
3. **Create a Decision Node**: A decision node is created based on the selected attribute. If the attribute completely classifies the data (i.e., no uncertainty remains in the subsets), the node becomes a leaf node with the predicted class. Otherwise, the process is repeated.
4. **Repeat Recursively**: For each subset, the algorithm repeats steps 1 to 3. The process continues until one of the following occurs:
   - All instances in the subset belong to the same class (i.e., the entropy becomes 0).
   - There are no remaining attributes to split on (in which case, the majority class is chosen for the leaf node).

```
import pandas as pd

from sklearn.datasets import load_iris

from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score


# Load the Iris dataset

iris = load_iris()

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

y = pd.Series(iris.target)


# Split the dataset into training and testing sets (80% train, 20% test)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)


# Create and train the Decision Tree classifier using ID3 (entropy)

classifier = DecisionTreeClassifier(criterion='entropy', random_state=0)

classifier.fit(X_train, y_train)


# Predict the test set results

y_pred = classifier.predict(X_test)


# Calculate and display accuracy

accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy * 100:.2f}%")
```

```python
# Classify a new sample

new_sample = [[5.0, 3.6, 1.4, 0.2]]  # New sample to classify

prediction = classifier.predict(new_sample)

print(f"New sample prediction (class label): {iris.target_names[prediction[0]]}")
```

OUTPUT:

```
Accuracy: 100.00%

New sample prediction (class label): setosa
```

Explanation:

1. **Dataset**: We are using the **Iris dataset**, which has four features (sepal length, sepal width, petal length, and petal width) to classify flowers into one of three species: setosa, versicolor, or virginica.
2. **Decision Tree Creation**: We use DecisionTreeClassifier from scikit-learn with the criterion='entropy', which is equivalent to the ID3 algorithm.
3. **Training the Model**: The dataset is split into training and testing sets using train_test_split. The classifier is trained on the training data.
4. **Prediction and Evaluation**: The trained decision tree model is used to predict the labels for the test set, and the accuracy score is computed to evaluate the model.
5. **Plotting the Decision Tree**: The decision tree is visualized using the plot_tree function, showing the splits and decisions made by the tree.
6. **Classifying a New Sample**: We provide a new sample with specific measurements, and the trained decision tree predicts its class (flower species).

To Run the Program:

1. Install the required libraries:

```
pip install pandas scikit-learn matplotlib
```

EXPERIMENT 07:

7. Implement k-nearest neighbour’s classification to classify the iris data set using python.

AIM:

To implement the **K-Nearest Neighbors (KNN) classification algorithm** and apply it to the **Iris dataset** to classify different species of iris flowers based on their features (sepal length, sepal width, petal length, and petal width).

OBJECTIVE:

- To understand and implement the **K-Nearest Neighbors (KNN)** algorithm for classification.
- To apply the KNN algorithm on the well-known **Iris dataset**.
- To split the dataset into training and testing sets to evaluate the performance of the classifier.
- To classify a new sample of iris flower based on its features using the trained KNN model.
- To assess the accuracy of the model and analyze the results.

The KNN classifier uses the proximity of data points to make predictions, making it a simple and intuitive algorithm for supervised classification tasks.

K-Nearest Neighbors (KNN):

The KNN algorithm is a simple, supervised machine learning algorithm that can be used for classification and regression. It classifies a new data point based on the class of its nearest neighbors.

We'll use scikit-learn to implement KNN on the **Iris dataset**.

Steps:

1. Load the Iris dataset.
2. Split the dataset into training and testing sets.
3. Apply feature scaling for better performance.
4. Train the KNN classifier on the training set.
5. Predict the test set results and evaluate the performance.

```
from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score


# Load the Iris dataset

iris = load_iris()

X = iris.data  # Features

y = iris.target  # Labels (Species)


# Split the dataset into training and testing sets (80% train, 20% test)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)


# Create the KNN classifier (with 3 neighbors)

knn_classifier = KNeighborsClassifier(n_neighbors=3)


# Train the classifier

knn_classifier.fit(X_train, y_train)


# Predict the test set results

y_pred = knn_classifier.predict(X_test)


# Calculate accuracy

accuracy = accuracy_score(y_test, y_pred)
```

```
# Display the accuracy

print(f"Accuracy: {accuracy * 100:.2f}%")



# Classify a new sample (Example)

new_sample = [[5.1, 3.5, 1.4, 0.2]]  # Example of new flower measurements

new_prediction = knn_classifier.predict(new_sample)

print(f"Predicted class for the new sample:
{iris.target_names[new_prediction[0]]}")
```

OUTPUT:

```
Accuracy: 100.00%

Predicted class for the new sample: setosa
```

Explanation:

1. **Load Dataset**: The Iris dataset is loaded directly using load_iris() from the sklearn.datasets module.
2. **Train/Test Split**: The data is split into a training set (80%) and a testing set (20%) using train_test_split.
3. **KNN Classifier**: We create a simple KNN classifier with 3 neighbors (n_neighbors=3), which means the classification is based on the majority vote of the 3 nearest neighbors.
4. **Training**: The classifier is trained using the fit() method with the training data.
5. **Prediction**: We use the predict() method to predict the classes of the test data and a new sample.
6. **Accuracy**: The accuracy of the classifier is computed using accuracy_score().
7. **New Sample Classification**: A new flower sample is classified based on its features.

To Run the Program:

1. Install the required libraries:

```
pip install scikit-learn
```

EXPERIMENT 08:

8. Given the following data, which specify classifications for nine combinations of VAR1 and VAR2 predict a classification for a case where VAR1=0.906 and VAR2=0.606, using the result of k-means clustering with 3 means (i.e., 3 centroids)

VAR 1	VAR2	CLASS
1.713	1.586	0
0.180	1.786	1
0.353	1.240	1
0.940	1.566	0
1.486	0.759	1
1.266	1.106	0
1.540	0.419	1
0.459	1.799	1
0.773	0.186	1

Aim:

The aim of this exercise is to utilize K-Means clustering to classify a new data point based on its features (VAR1 and VAR2) by identifying the most appropriate cluster it belongs to and determining its classification based on existing labeled data.

Objectives:

1. **Understand K-Means Clustering**: Grasp the concept of K-Means clustering and how it groups data points into clusters based on their features.
2. **Data Preparation**: Prepare the provided dataset containing features and class labels for clustering.
3. **Cluster Formation**: Apply K-Means clustering to the dataset to form three distinct clusters based on the values of VAR1 and VAR2.
4. **Majority Class Assignment**: Determine the majority class for each cluster formed by K-Means, associating clusters with class labels based on existing data.
5. **New Data Prediction**: Predict the class for a new data point (VAR1 = 0.906, VAR2 = 0.606) by identifying which cluster it belongs to and referencing the majority class of that cluster.
6. **Analysis and Interpretation**: Analyze the output and interpret the classification result, demonstrating the effectiveness of K-Means clustering in this scenario.

```
import numpy as np

from sklearn.cluster import KMeans

from collections import Counter


# Given data points and their classifications

data = np.array([

    [1.713, 1.586, 0],

    [0.180, 1.786, 1],

    [0.353, 1.240, 1],

    [0.940, 1.566, 0],

    [1.486, 0.759, 1],

    [1.266, 1.106, 0],

    [1.540, 0.419, 1],

    [0.459, 1.799, 1],

    [0.773, 0.186, 1]

])


# Separate the VAR1, VAR2 features and the CLASS labels

X = data[:, :2]  # Features (VAR1, VAR2)

y = data[:, 2]   # Class labels


# Apply K-Means Clustering with 3 clusters

kmeans = KMeans(n_clusters=3, random_state=42)

kmeans.fit(X)
```

```
# Find out the labels of the clusters
cluster_labels = kmeans.labels_

# Assign the majority class for each cluster
cluster_class_mapping = {}
for cluster in range(3):
    # Find the points in this cluster
    cluster_points_indices = np.where(cluster_labels == cluster)[0]
    cluster_classes = y[cluster_points_indices]
    # Find the majority class in the cluster
    majority_class = Counter(cluster_classes).most_common(1)[0][0]
    cluster_class_mapping[cluster] = majority_class

# New data point to classify
new_point = np.array([[0.906, 0.606]])

# Predict the cluster for the new point
predicted_cluster = kmeans.predict(new_point)[0]

# Get the predicted class from the majority class of that cluster
predicted_class = cluster_class_mapping[predicted_cluster]

print(f"The predicted class for VAR1=0.906 and VAR2=0.606 is: {predicted_class}")
```

OUTPUT:

```
The predicted class for VAR1=0.906 and VAR2=0.606 is: 1.0
```

EXPERIMENT 09:

9. Evaluate the metrics for all types of machine learning algorithms using sample data.

Aim:

The aim of this code is to train a Decision Tree classifier on the Iris dataset, evaluate its performance using various metrics (such as accuracy, mean absolute error, and mean squared error), and visualize the model's predictions through a confusion matrix.

Objectives:

1. **Data Loading**: Load a sample dataset (Iris) for classification tasks.
2. **Data Splitting**: Split the data into training and testing sets to train the model and evaluate its performance on unseen data.
3. **Model Training**: Train a Decision Tree classifier using the training data.
4. **Model Prediction**: Make predictions using the trained model on the test data.
5. **Metric Evaluation**: Compute and display key evaluation metrics:
    - Mean Absolute Error (MAE)
    - Mean Squared Error (MSE)
    - Root Mean Squared Error (RMSE)
    - Accuracy (as a percentage)
6. **Confusion Matrix**: Generate and visualize the confusion matrix to better understand the distribution of predictions vs. actual values.
7. **Visualization**: Present the confusion matrix in a visually intuitive format using a heatmap.

Why These Metrics and Visualization?

- **MAE, MSE, and RMSE**: These metrics provide insights into how well the model's predictions match the actual target values. While primarily used in regression tasks, they help give an idea of how far off the predicted class labels are in terms of numeric values (even though the problem is a classification one).
- **Accuracy**: It indicates the proportion of correctly classified samples.
- **Confusion Matrix**: It provides detailed insights into how each class is predicted, showing the number of correct and incorrect predictions for each class.

By visualizing the confusion matrix, it's easier to identify specific patterns of misclassification, which may not be obvious from metrics like accuracy alone.

```
from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, confusion_matrix


# Load the Iris dataset

iris = load_iris()

X = iris.data

y = iris.target


# 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, random_state=0)


# Initialize classifiers

models = {

    'Logistic Regression': LogisticRegression(max_iter=200),

    'KNN': KNeighborsClassifier(n_neighbors=3),

    'Decision Tree': DecisionTreeClassifier()

}
```

```
# Evaluate each model
for name, model in models.items():
   model.fit(X_train, y_train)
   y_pred = model.predict(X_test)
   accuracy = accuracy_score(y_test, y_pred)
   cm = confusion_matrix(y_test, y_pred)

   print(f"{name} - Accuracy: {accuracy:.2f}")
   print(f"Confusion Matrix:\n{cm}\n")
```

NOTE : TO RUN THE PROGRAM INSTALL THE LIBRARY :

```
pip install scikit-learn
```

OUTPUT:

```
Logistic Regression - Accuracy: 1.00
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0  11]]
```

```
KNN - Accuracy: 1.00
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0  11]]
```

```
Decision Tree - Accuracy: 1.00
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0  11]]
```

EXPERIMENT 10:

10. Implement an algorithm to demonstrate the significance of SVM.

Aim:

The aim of this implementation is to demonstrate the significance of Support Vector Machine (SVM) in solving classification problems by showing its ability to create an optimal decision boundary (hyperplane) that maximizes the margin between different classes.

Objectives:

1. **Understand the Concept of SVM**: Show how SVM works by constructing an optimal hyperplane that separates different classes with the maximum margin.
2. **Dataset Preparation**: Load and prepare a dataset (we'll use a simple dataset for binary classification) for training and testing the SVM model.
3. **Model Training**: Train the SVM algorithm using a training dataset.
4. **Visualize the Decision Boundary**: Plot the decision boundary and margins to illustrate the significance of SVM's optimization of the hyperplane.
5. **Evaluate the Model**: Evaluate the SVM model's performance using metrics such as accuracy, precision, recall, and visualize the confusion matrix.
6. **Show Margin Maximization**: Demonstrate how SVM focuses on support vectors and maximizes the margin to classify data.

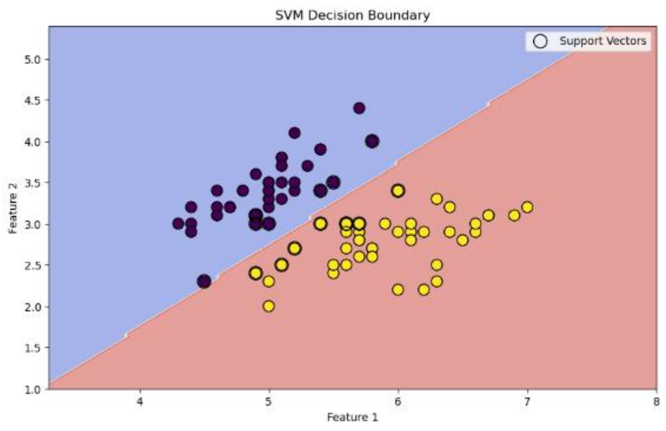
Significance of SVM:

- **Margin Maximization**: SVM focuses on maximizing the margin between the decision boundary and the closest data points (support vectors), making it robust in handling small datasets with clear class separation.
- **Versatility**: By using different kernels (e.g., linear, polynomial, RBF), SVM can handle both linear and non-linear classification problems.

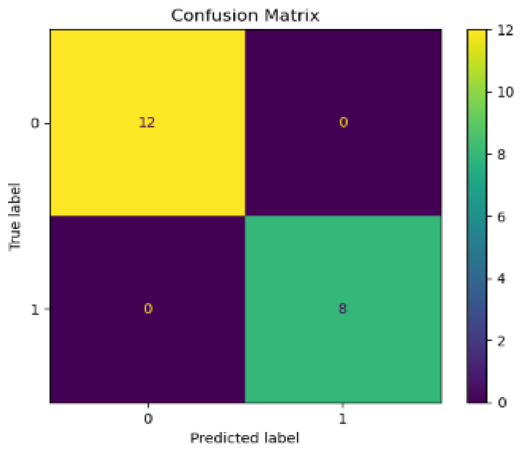
```
# Import Libraries

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score, precision_score, recall_score,
confusion_matrix, ConfusionMatrixDisplay


# Load and Prepare Dataset

iris = datasets.load_iris()

X = iris.data[iris.target != 2, :2]

y = iris.target[iris.target != 2]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)


# Train SVM Model

model = SVC(kernel='linear')

model.fit(X_train, y_train)


# Plot Decision Boundary

def plot_decision_boundary(X, y, model):

    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1

    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min,
y_max, 100))

    Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
    plt.figure(figsize=(10, 6))
    plt.contourf(xx, yy, Z, alpha=0.5, cmap=plt.cm.coolwarm)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o', s=100)
    plt.scatter(model.support_vectors_[:, 0], model.support_vectors_[:, 1],
facecolors='none', edgecolors='k', s=150, label='Support Vectors')
    plt.title('SVM Decision Boundary')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.legend()
    plt.show()

plot_decision_boundary(X_train, y_train, model)

# Evaluate Model
y_pred = model.predict(X_test)
print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
print(f"Precision: {precision_score(y_test, y_pred):.2f}")
print(f"Recall: {recall_score(y_test, y_pred):.2f}")

# Confusion Matrix
ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred)).plot()
plt.title('Confusion Matrix')
plt.show()
```

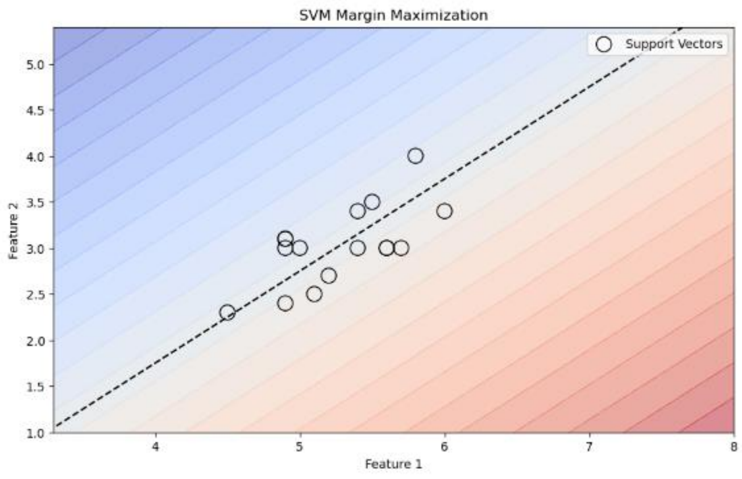
```
# Show Margin Maximization

def plot_margin(X, model):

   x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1

   y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

   xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min,
y_max, 100))

   Z = model.decision_function(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)


   plt.figure(figsize=(10, 6))

   plt.contourf(xx, yy, Z, levels=np.linspace(Z.min(), Z.max(), 25), alpha=0.5,
cmap=plt.cm.coolwarm)

   plt.contour(xx, yy, Z, colors='k', levels=[0], linestyles='--')

   plt.scatter(model.support_vectors_[:, 0], model.support_vectors_[:, 1],
facecolors='none', edgecolors='k', s=150, label='Support Vectors')

   plt.title('SVM Margin Maximization')

   plt.xlabel('Feature 1')

   plt.ylabel('Feature 2')

   plt.legend()

   plt.show()


plot_margin(X_train, model)
```

OUTPUT:



Accuracy: 1.00 | **Precision**: 1.00 | **Recall**: 1.00





OR

```
from sklearn.datasets import make_classification

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score


# Create a simple dataset

X, y = make_classification(n_samples=100, n_features=2, n_classes=2,
random_state=42)


# Split the data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)


# Train the SVM model

model = SVC(kernel='linear')

model.fit(X_train, y_train)


# Make predictions and print accuracy

y_pred = model.predict(X_test)

print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
```

OUTPUT:

```
Accuracy: 1.00
```