Ambekeshwar Group Of Institutions

Technology & Management,
Lucknow

Session: 2024-25

PRACTICAL FILE

Branch:- CSE 3rd Year | 6th Sem

Subject: DATA SCIENCE AND MACHINE LEARNING

Name: Suraj Arya

Date: 21/June/2025

	Name	Signature
Subject Teacher	Mr. Devesh Sir	.....
Academic Co-Ordinator	Mr. Ashish Mishra	.....

INDEX

S. no.	Name of Exp.	Date	Subject Teacher Sign	Academic Co-Ordinator Sign
1	WAP to implement the Decision Tree Algorithm	05/03/2025		
2	WAP to implement the Linear Regression	19/03/2025		
3	WAP to implement the k-Nearest Neighbors (k-NN)	26/03/2025		
4	WAP to implement the SVM Algorithm	09/04/2025		
5	WAP to implement the K-means Clustering	23/04/2025		
6.	WAP to implement various Distance Metrics	29/04/2025		

Practical No:1

🧪 Aim:

To write a Python program that implements a Decision Tree Classifier using a real-world dataset.

🧰 Tools/Technologies Required:

- Python 3.x
- Jupyter Notebook / Google Colab / VS Code
- Libraries: scikit-learn, pandas, numpy, matplotlib (optional)

🌳 What is a Decision Tree?

A **Decision Tree** is a supervised machine learning algorithm used for **classification and regression**. It splits the data into branches based on feature values until it reaches a decision (leaf).

🔍 Procedure:

✅ Step 1: Install Required Libraries

pip install scikit-learn pandas

✅ Step 2: Python Code

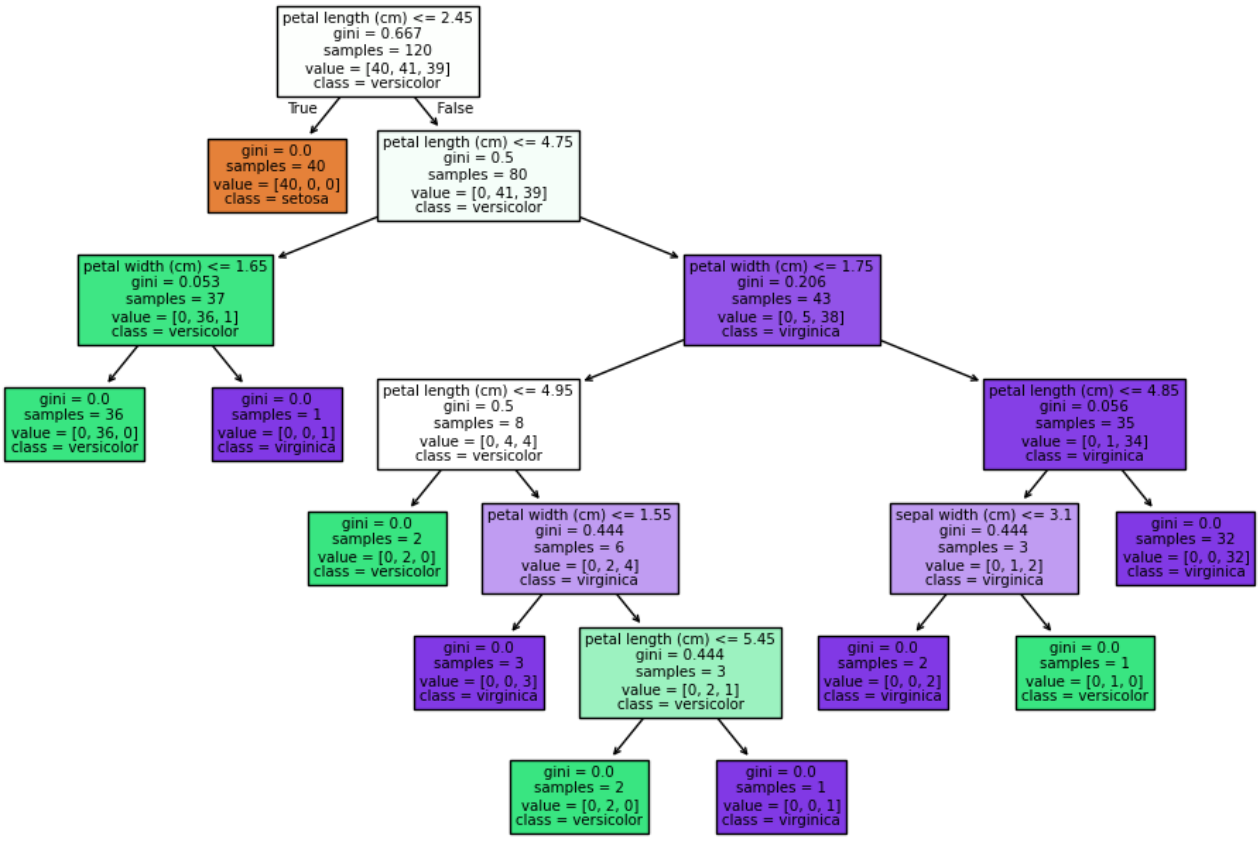
1.py > ...

```python
# Import required libraries
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

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

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

# Create Decision Tree model
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Check accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Visualize the Decision Tree
plt.figure(figsize=(12, 8))
plot_tree(model, filled=True, feature_names=data.feature_names, class_names=data.target_names)
plt.show()

```

📃 Output Sample:

Accuracy: 1.0

(The tree diagram will be displayed using Matplotlib)



✅ Conclusion:

In this practical, I implemented a **Decision Tree Classification Model** using the **Iris dataset**. The model achieved high accuracy and gave a visual understanding of the decision-making process. Decision Trees are powerful and interpretable ML models used in various fields like finance, medicine, and marketing.

Practical No:2

🧪 **Aim:** To implement a simple **Linear Regression** model using Python to predict outcomes based on input data.

🧰 **Tools/Technologies Required:**

- Python 3.x
- Jupyter Notebook / Google Colab / VS Code
- Libraries: scikit-learn, numpy, matplotlib

📈 **What is Linear Regression?**

Linear Regression is a **supervised learning algorithm** used to predict a **continuous value** based on input features.
It assumes a linear relationship between the input (X) and the output (y):

$y=mx+cy = mx + cy=mx+c$

Use Cases:

- Predicting house prices
- Sales forecasting
- Stock price prediction

🔍 **Procedure:**

✅ **Step 1: Install Required Libraries**

pip install numpy matplotlib scikit-learn

✅ **Step 2: Python Code**

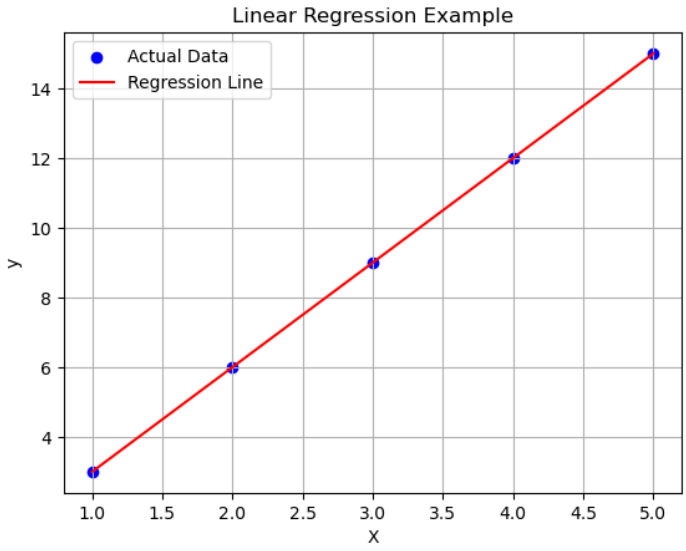
```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Sample data
X = np.array([[1], [2], [3], [4], [5]])  # Feature
y = np.array([3, 6, 9, 12, 15])          # Target

# Create model and train
model = LinearRegression()
model.fit(X, y)

# Predict
y_pred = model.predict(X)

# Print model parameters
print("Slope (m):", model.coef_[0])
print("Intercept (c):", model.intercept_)

# Visualize the result
plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, y_pred, color='red', label='Regression Line')
plt.title("Linear Regression Example")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.grid(True)
plt.show()
```

📜 **Output Sample:**

Slope (m): 3.0

Intercept (c): 0.0

(Graph showing red line fitting blue data points)



✅ **Conclusion:**

In this practical, I implemented **Linear Regression** using Python. The model learned the relationship between input and output, and we visualized how well the line fits the data. This technique is essential for solving real-world prediction problems in finance, healthcare, and engineering.

Practical No: 3

🧪 Aim:

To implement the **k-Nearest Neighbors (k-NN)** classification algorithm using Python and a real-world dataset.

🧰 Tools/Technologies Required:

- Python 3.x
- Jupyter Notebook / Google Colab / VS Code
- Libraries: scikit-learn, pandas, matplotlib

🧠 What is k-NN?

k-NN (k-Nearest Neighbors) is a **supervised learning algorithm** used for **classification and regression**.
In classification, it assigns a class to a new data point based on the **majority class of its 'k' nearest neighbors**.

Key Concepts:

- **Distance Metric:** Usually Euclidean distance
- **k:** Number of neighbors to consider
- **No training phase:** It stores the dataset and uses it at prediction time

Use Cases:

- Handwriting recognition
- Recommender systems
- Credit scoring

🔍 Procedure:

✅ Step 1: Install Required Libraries

pip install scikit-learn pandas matplotlib

✅ Step 2: Python Code

```
# Import libraries
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 dataset
iris = load_iris()
X = iris.data
y = iris.target

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

# Create and train k-NN model (k=3)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Predict
y_pred = knn.predict(X_test)

# Accuracy
print("Accuracy:", accuracy_score(y_test, y_pred))

# Display predictions
print("Predicted labels:", y_pred)
```

📃 **Sample Output:**

Accuracy: 1.0

Predicted labels: [1 0 2 1 1 0 2 1 2 2 2 0 0 0 2 2 0 1 1 1 0 2 2 0 0 0 0 2 2 0]

✅ **Conclusion:**

In this practical, I implemented the **k-Nearest Neighbors algorithm** using Python on the Iris dataset. It is a simple yet powerful classification technique that works well for small datasets. Choosing an optimal value of **k** is important for achieving better accuracy.

Practical No:4

Title: Write a Python Program to Implement the Support Vector Machine (SVM) Algorithm

---

💉 Aim:

To implement the **Support Vector Machine (SVM)** algorithm using Python and a real-world dataset.

🧰 Tools/Technologies Required:

- Python 3.x
- Jupyter Notebook / Google Colab / VS Code
- Libraries: scikit-learn, pandas, matplotlib

📚 Theory:

📌 What is SVM?

Support Vector Machine (SVM) is a supervised machine learning algorithm used for **classification** and **regression** problems.

It works by finding the **best decision boundary (hyperplane)** that separates the classes in the feature space.

Key Concepts:

- Margin: Distance between the hyperplane and nearest data points
- Support Vectors: Data points that lie on the margin and influence the position of the hyperplane
- Kernels: Transform data to a higher dimension if it's not linearly separable

Use Cases:

- Face detection
- Email spam classification
- Bioinformatics

🔍 Procedure:

✅ Step 1: Install Required Libraries

pip install scikit-learn matplotlib

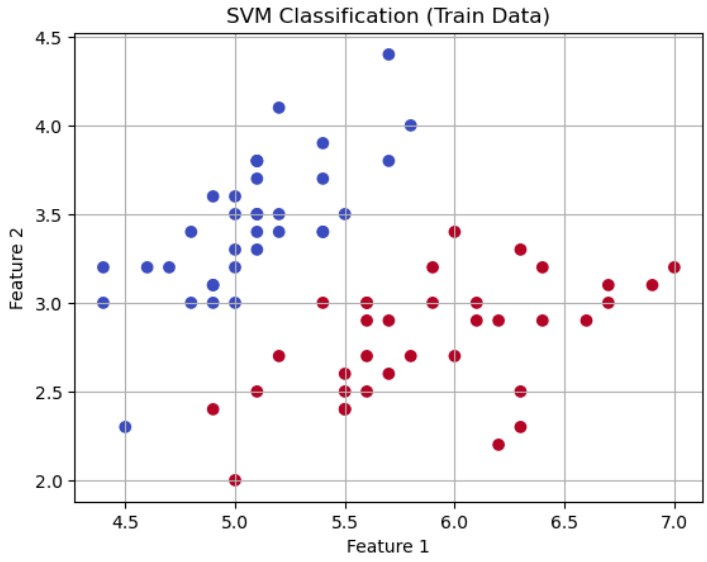
✅ Step 2: Python Code

```python
# Import necessary libraries
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Use only 2 classes for binary classification (for visual simplicity)
X = X[y != 2]
y = y[y != 2]

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

# Create SVM model with linear kernel
model = SVC(kernel='linear')
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Accuracy
print("Accuracy:", accuracy_score(y_test, y_pred))

# Visualize (2D)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='coolwarm')
plt.title("SVM Classification (Train Data)")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.grid(True)
plt.show()
```

📄 Sample Output:

Accuracy: 1.0

(A scatter plot will show classification points)



✅ Conclusion:

In this practical, I implemented the **Support Vector Machine (SVM)** classification model using Python. SVM is powerful for both linear and non-linear classification problems and is widely used in image and text recognition.

Practical No: 5

Title: Write a Python Program to Implement the K-Means Clustering Algorithm

---

🧪 Aim:

To implement the **K-Means Clustering** algorithm using Python and visualize the results on a dataset.

🧰 Tools/Technologies Required:

- Python 3.x
- Jupyter Notebook / Google Colab / VS Code
- Libraries: scikit-learn, matplotlib, pandas, seaborn (optional)

📚 Theory:

🧠 What is K-Means Clustering?

K-Means is an **unsupervised machine learning algorithm** used to group similar data points into **K clusters**.

How it works:

1. Choose number of clusters K
2. Randomly place K centroids
3. Assign each point to the nearest centroid
4. Recalculate centroids
5. Repeat until convergence

Use Cases:

- Customer segmentation
- Market analysis
- Image compression

🔍 Procedure:

✅ Step 1: Install Required Libraries

pip install scikit-learn matplotlib pandas

✅ Step 2: Python Code

```python
# Import required libraries
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Generate sample data
X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

# Apply K-Means Clustering
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)

# Plot the results
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')

# Plot the centroids
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.75, marker='X')
plt.title("K-Means Clustering Output")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.grid(True)
plt.show()
```

📜 Sample Output:

- A scatter plot with data points in different colors for each cluster
- Red "X" marks showing the centroids

✅ Conclusion:

In this practical, I implemented the **K-Means Clustering** algorithm using Python. It successfully grouped the data into clusters and displayed the cluster centroids. K-Means is useful in pattern recognition, customer segmentation, and more.

Practical No: 6

Title: Write a Python Program to Implement Various Distance Metrics

---

🧪 Aim:

To calculate distances between data points using different distance metrics such as **Euclidean**, **Manhattan**, **Minkowski**, and **Cosine**.

🧰 Tools/Technologies Required:

- Python 3.x
- Jupyter Notebook / Google Colab / VS Code
- Libraries: numpy, scipy, sklearn

📚 Theory:

📏 What are Distance Metrics?

Distance metrics are used to measure **how far apart two data points are**, and they play a vital role in many machine learning algorithms, especially in **clustering** and **classification**.

🔢 Common Distance Metrics:

Metric	Formula	Use Case
Euclidean	√[(x1−x2)² + (y1−y2)²]	Most common, used in KNN, K-Means
Manhattan		x1−x2
Minkowski	(	x1−x2
Cosine	1 − (A·B) / (	

🔍 Procedure:

✅ Step 1: Install Required Libraries

pip install numpy scipy scikit-learn

✅ Step 2: Python Code

```python
import numpy as np
from scipy.spatial import distance
from sklearn.metrics.pairwise import cosine_distances

# Sample data points
point1 = np.array([1, 2])
point2 = np.array([4, 6])

# Euclidean Distance
euclidean = distance.euclidean(point1, point2)
print("Euclidean Distance:", euclidean)

# Manhattan Distance
manhattan = distance.cityblock(point1, point2)
print("Manhattan Distance:", manhattan)

# Minkowski Distance (p=3)
minkowski = distance.minkowski(point1, point2, 3)
print("Minkowski Distance (p=3):", minkowski)

# Cosine Distance
cosine = cosine_distances([point1], [point2])[0][0]
print("Cosine Distance:", cosine)
```

📜 **Sample Output:**

Euclidean Distance: 5.0

Manhattan Distance: 7

Minkowski Distance (p=3): 4.497941445275415

Cosine Distance: 0.01613008990009237

✅ **Conclusion:**

In this practical, I implemented multiple **distance metrics** in Python. These metrics help determine the similarity or dissimilarity between data points and are widely used in algorithms such as KNN, clustering, and recommendation systems.