

Machine Learning Lab (ECE-350P)

Experiment 1

Aim: Introduction to:

- (A) Jupyter Notebook IDE**
- (B) Spyder IDE**
- (C) NumPy and Pandas**
- (D) Matplotlib and Seaborn, and**
- (E) Scikit Learn**

Theory:

(A) Jupyter Notebook IDE: Jupyter Notebook is an open-source, web-based interactive development environment (IDE) that allows users to create and share documents containing live code, equations, visualizations and explanatory texts, it supports over 40 programming languages, with python being the most widely used.

Features:

- **Interactive Cells:** Code is executed in isolated cells, allowing for incremental testing and debugging.
- **Markdown Support:** Markdown cells enable the inclusion of headings, lists, links, and other formatting options to create structured documentation.
- **Data Visualization:** Direct integration with libraries like Matplotlib, Seaborn, and Plotly allows inline plotting of data.
- **Reproducibility:** Notebooks can be saved in .ipynb format and shared for reproducibility of results.
- **Kernel Support:** Jupyter supports various programming kernels like Python, R, Julia, and more.

Use Cases:

- Data analysis and visualization
- Machine Learning and deep learning
- Educational tutorials and presentations

(B) Spyder IDE: Spyder (Scientific Python Development Environment) is a powerful open-source python IDE designed for scientific computing and data analysis. It is part of the Anaconda distribution and is widely used by data scientists and engineers.

Features:

- **Multi-panel Layout:** Includes a code editor, console, file explorer, and variable explorer.
- **Integrated Debugger:** Allows setting breakpoints and stepping through code.

Machine Learning Lab (ECE-350P)

- **Variable Explorer:** Displays data structures like arrays and dataframes, enabling real-time inspection and modification.
- **IPython Console:** Provides interactive testing and debugging.
- **Integrated Documentation:** Inline help and documentation support improve code efficiency.

Use Cases:

- Scientific Programming
- Data Analysis
- Debugging complex machine learning models

(C) NumPy and Pandas: NumPy (Numerical Python) and Pandas are the two most widely used Python libraries for numerical computing and data manipulation.

NumPy: NumPy provides support for large, multi-dimensional arrays and matrices along with a collection of mathematical functions to operate on these arrays.

Features:

- Efficient handling of arrays and matrices
- Supports element-wise operations
- Offers linear algebra, Fourier transforms, and random number capabilities.

Pandas: Pandas is used for data manipulation and analysis. It introduces two main data structures:

- Series: One dimensional labeled array
- DataFrame: Two-dimensional labeled data structures

(D) Matplotlib and Seaborn: Matplotlib and Seaborn are Python libraries used for data visualization.

Matplotlib: Matplotlib is a low-level library that provides extensive plotting functions such as line plots, scatter plots, bar plots, and histograms.

Seaborn: Seaborn is built on top of Matplotlib and provides a high-level interface for drawing attractive statistical graphics. It integrates closely with Pandas data structures.

(E) Scikit-Learn: Scikit-Learn is one of the most widely used machine learning libraries in Python. It provides simple and efficient tools for data mining and machine learning.

Features:

- Supervised and unsupervised learning algorithms
- Tools for model selection and evaluation
- Preprocessing and data transformation
- Support for regression, classification, clustering, and dimensionality reduction

Installation Instructions:

Step 1: Install Python:

- 1) Go to the official Python website: <https://www.python.org>
- 2) Download the latest version of Python (preferably Python 3.10 or higher).
- 3) During installation, check “Add Python to PATH” and proceed with the installation.
- 4) Verify installation using the command:

```
python --version
```

Step 2: Install Anaconda:

Alternatively, you can install the Anaconda distribution, which includes Jupyter Notebook, Spyder, and common scientific libraries pre-installed.

- 1) Download Anaconda from <https://www.anaconda.com>.
- 2) Install Anaconda following the on screen instructions.
- 3) Verify installation using:

```
conda --version
```

Step 3: Install Jupyter Notebook:

You can install Jupyter Notebook using either pip or conda:

Using pip:

```
pip install notebook
```

Using conda (if Anaconda is installed):

```
conda install -c conda-forge notebook
```

To run Jupyter Notebook:

```
jupyter notebook
```

Step 4: Install Spyder IDE:

You can install Spyder IDE using either pip or conda:

Using pip:

```
pip install spyder
```

Using conda (if Anaconda is installed):

```
conda install -c anaconda spyder
```

To run Spyder IDE:

```
spyder
```

Step 5: Install NpmPy and Pandas:

You can install NumPy and Pandas using pip or conda using either of the following commands:

Using pip:

```
pip install numpy pandas
```

Using conda:

```
conda install numpy pandas
```

Step 6: Install Matplotlib and Seaborn:

You can install Matplotlib and Seaborn using the following commands:

Using pip:

```
pip install matplotlib seaborn
```

Using conda:

```
conda install matplotlib seaborn
```

Step 7: Install Scikit-Learn:

You can install Scikit-Learn using the following commands:

Using pip:

```
pip install scikit-learn
```

Using conda:

```
conda install scikit-learn
```

Machine Learning Lab (ECE-350P)

Experiment 2

Aim: Write a program to demonstrate Simple Linear Regression

Software Used: Visual Studio Code

Code:

```
● ● ●
1 # Importing necessary libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import pandas as pd
5 from sklearn.model_selection import train_test_split
6 from sklearn.linear_model import LinearRegression
7 from sklearn.metrics import mean_squared_error, r2_score
8
9 # Sample dataset (Hours Studied vs Marks Scored)
10 data = {
11     'HoursStudied': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
12     'MarksScored': [35, 40, 45, 50, 55, 65, 70, 75, 80, 90]
13 }
14
15 # Creating a DataFrame
16 df = pd.DataFrame(data)
17
18 # Splitting the data into training and testing sets
19 X = df[['HoursStudied']] # Independent variable
20 y = df['MarksScored'] # Dependent variable
21
22 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
23
24 # Creating and training the model
25 model = LinearRegression()
26 model.fit(X_train, y_train)
27
28 # Making predictions
29 y_pred = model.predict(X_test)
30
31 # Visualizing the training set results
32 plt.scatter(X_train, y_train, color='red', label='Training Data')
33 plt.plot(X_train, model.predict(X_train), color='blue', label='Best Fit Line')
34 plt.title('Marks vs Hours Studied (Training Set)')
35 plt.xlabel('Hours Studied')
36 plt.ylabel('Marks Scored')
37 plt.legend()
38 plt.show()
39
40 # Visualizing the test set results
41 plt.scatter(X_test, y_test, color='red', label='Test Data')
42 plt.plot(X_train, model.predict(X_train), color='blue', label='Best Fit Line')
43 plt.title('Marks vs Hours Studied (Test Set)')
44 plt.xlabel('Hours Studied')
45 plt.ylabel('Marks Scored')
46 plt.legend()
47 plt.show()
48
49 # Model Evaluation
50 print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
51 print("R-squared Score:", r2_score(y_test, y_pred))
52
53 # Coefficients
54 print("Slope (Coefficient):", model.coef_[0])
55 print("Intercept:", model.intercept_)
56
57 # Prediction example
58 hours = float(input("\nEnter hours studied: "))
59 predicted_marks = model.predict([[hours]])
60 print(f"Predicted Marks for {hours} hours of study: {predicted_marks[0]:.2f}")
```

Output:

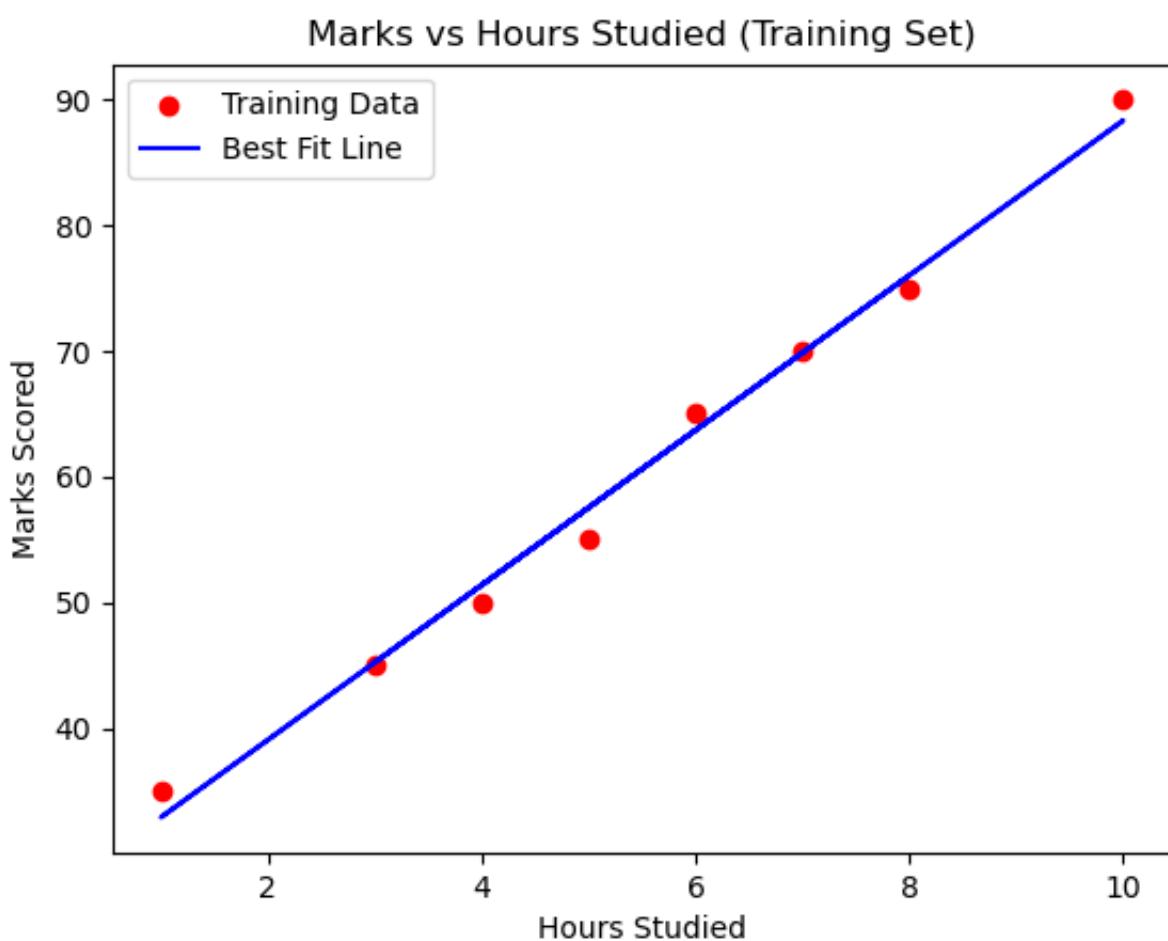
Terminal:

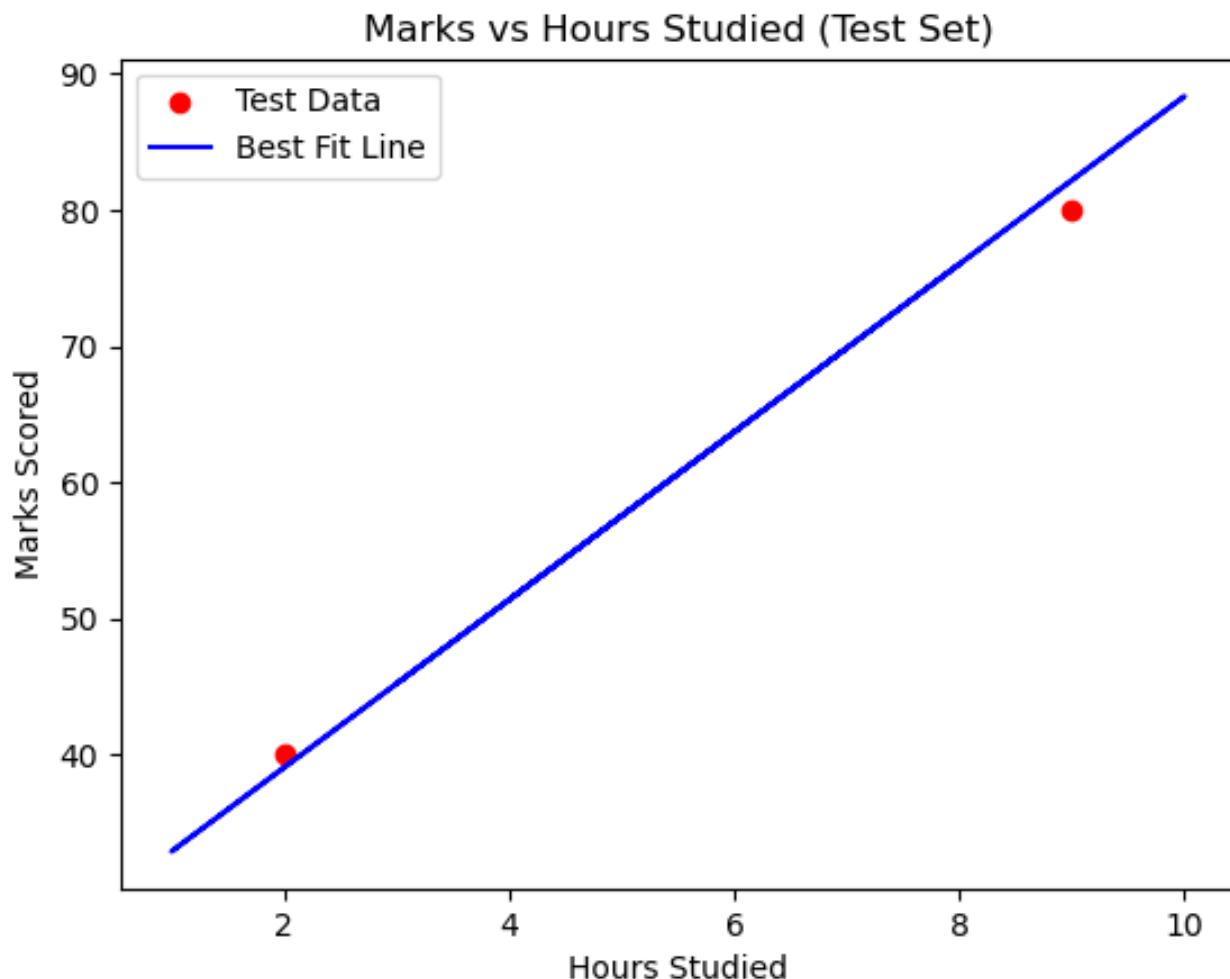
```
python3 -m exp2 #Command to run a python file, here exp2.py  
2025-03-11 20:03:56.104 python3[16900:670047] +[IMKClient subclass]: chose  
IMKClient_Modern  
2025-03-11 20:03:56.104 python3[16900:670047] +[IMKInputSession subclass]:  
chose IMKInputSession_Modern  
2025-03-11 20:04:07.902 python3[16900:670047] The class 'NSSavePanel' overrides  
the method identifier. This method is implemented by class 'NSWindow'  
Mean Squared Error: 2.8658219381688856  
R-squared Score: 0.9928354451545778  
Slope (Coefficient): 6.163793103448278  
Intercept: 26.724137931034477
```

Enter hours studied: 10

Predicted Marks for 10.0 hours of study: 88.36

Graph:





Experiment 3

Machine Learning Lab (ECE-350P)

Aim: Write a program to demonstrate Logistic Regression.

Software Used: Visual Studio Code

Code:

```
● ● ●
1 # Importing necessary libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import pandas as pd
5 from sklearn.model_selection import train_test_split
6 from sklearn.linear_model import LogisticRegression
7 from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
8
9 # Sample dataset (Age, Glucose Level, Diabetes)
10 data = {
11     'Age': [25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80],
12     'GlucoseLevel': [80, 85, 90, 95, 100, 110, 120, 130, 140, 150, 160, 170],
13     'Diabetes': [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1] # 0 = No Diabetes, 1 = Diabetes
14 }
15
16 # Creating a DataFrame
17 df = pd.DataFrame(data)
18
19 # Independent variables (features)
20 X = df[['Age', 'GlucoseLevel']]
21 # Dependent variable (target)
22 y = df['Diabetes']
23
24 # Splitting the data into training and testing sets (80% training, 20% testing)
25 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
26
27 # Creating and training the Logistic Regression model
28 model = LogisticRegression()
29 model.fit(X_train, y_train)
30
31 # Making predictions
32 y_pred = model.predict(X_test)
33
34 # Model Evaluation
35 print("\nModel Accuracy:", accuracy_score(y_test, y_pred))
36 print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
37 print("\nClassification Report:\n", classification_report(y_test, y_pred))
38
39 # Predict for new input
40 age = float(input("\nEnter age: "))
41 glucose = float(input("Enter glucose level: "))
42 prediction = model.predict([[age, glucose]])
43 result = 'Diabetic' if prediction[0] == 1 else 'Non-Diabetic'
44 print(f"\nPrediction for Age = {age} and Glucose Level = {glucose}: {result}")
45
46 # Visualizing the decision boundary
47 plt.figure(figsize=(10, 6))
48 plt.scatter(df['Age'], df['GlucoseLevel'], c=df['Diabetes'], cmap='bwr', edgecolors='k')
49 plt.title('Diabetes Prediction based on Age and Glucose Level')
50 plt.xlabel('Age')
51 plt.ylabel('Glucose Level')
52
53 # Plot decision boundary
54 x_min, x_max = X['Age'].min() - 1, X['Age'].max() + 1
55 y_min, y_max = X['GlucoseLevel'].min() - 1, X['GlucoseLevel'].max() + 1
56 xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.5),
57                       np.arange(y_min, y_max, 0.5))
58 Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
59 Z = Z.reshape(xx.shape)
60 plt.contourf(xx, yy, Z, alpha=0.3, cmap='bwr')
61 plt.colorbar()
62
63 plt.show()
```

Machine Learning Lab (ECE-350P)

Output:

Terminal:

```
python3 -m exp3
Model Accuracy: 1.0
Confusion Matrix:
```

```
[[1 0]
 [0 2]]
```

```
Classification Report:
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	1
1	1.00	1.00	1.00	2

accuracy			1.00	3
macro avg	1.00	1.00	1.00	3
weighted avg	1.00	1.00	1.00	3

```
Enter age: 24
```

```
Enter glucose level: 110
```

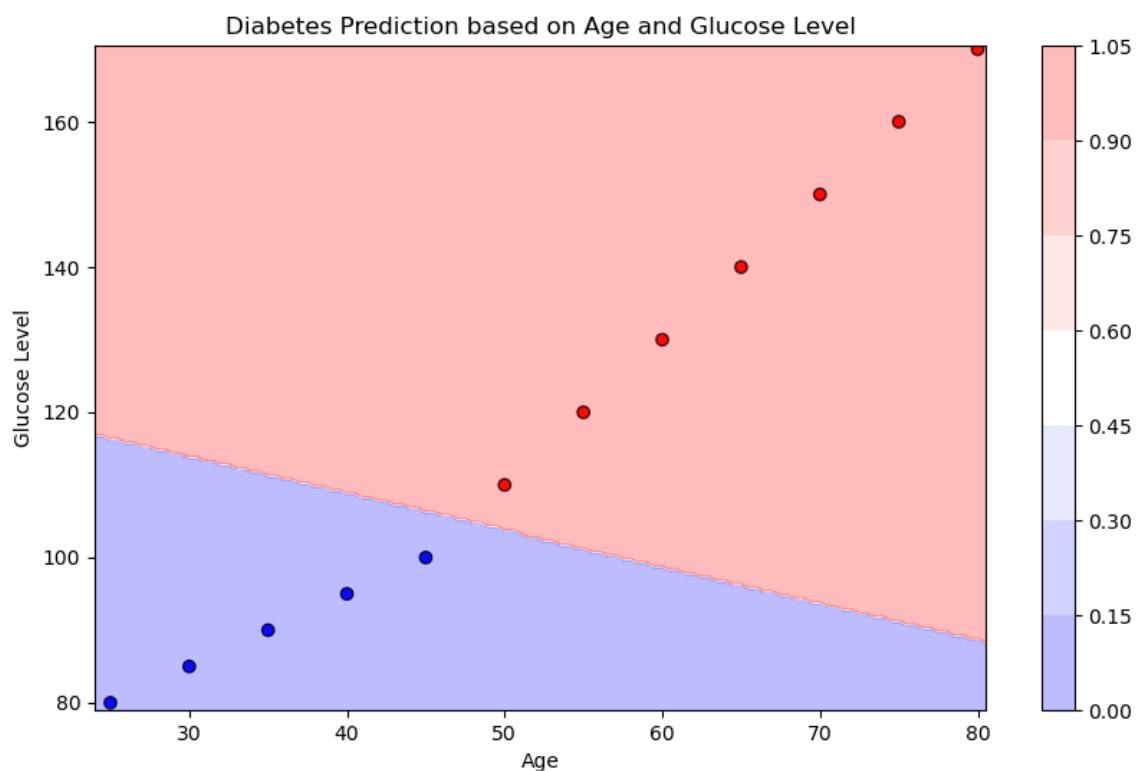
```
Prediction for Age = 24.0 and Glucose Level = 110.0: Non-Diabetic
```

```
2025-03-11 20:20:53.915 python3[17165:681930] +[IMKClient subclass]: chose
```

```
IMKClient_Modern
```

```
2025-03-11 20:20:53.915 python3[17165:681930] +[IMKInputSession subclass]: chose IMKInputSession_Modern
```

Graph:



Machine Learning Lab (ECE-350P)

Experiment 4

Aim: Write a program to demonstrate Decision Tree - ID3 Algorithm

Software Used: Visual Studio Code

Code:

```
● ● ●
1 # Importing necessary libraries
2 import pandas as pd
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import seaborn as sns
6 from sklearn.tree import DecisionTreeClassifier, plot_tree
7 from sklearn.model_selection import train_test_split
8 from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
9 # ----- LOADING DATA -----
10 # Load the IRIS dataset
11 url = 'https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv'
12 df = pd.read_csv(url)
13 # Display first few rows of the dataset
14 print("\nSample Data:\n", df.head())
15 # ----- DATA EXPLORATION -----
16 # Basic information about the data
17 print("\nDataset Info:")
18 print(df.info())
19 # Summary statistics of numeric features
20 print("\nStatistical Summary:")
21 print(df.describe())
22 # Count of each class
23 print("\nClass Distribution:\n", df['species'].value_counts())
24 # ----- VISUALIZATION -----
25 # Pair plot to visualize relationships between features
26 sns.pairplot(df, hue='species')
27 plt.show()
28 # ----- DATA PREPROCESSING -----
29 # Splitting data into features (X) and target (y)
30 X = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
31 y = df['species']
32 # Splitting into training and testing sets (80% training, 20% testing)
33 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
34 # ----- MODEL TRAINING -----
35 # Creating and training the Decision Tree classifier using ID3 algorithm (criterion='entropy')
36 model = DecisionTreeClassifier(criterion='entropy', random_state=42)
37 model.fit(X_train, y_train)
38 # ----- PREDICTION -----
39 # Making predictions
40 y_pred = model.predict(X_test)
41 # ----- MODEL EVALUATION -----
42 # Evaluating the model
43 print("\nModel Accuracy:", accuracy_score(y_test, y_pred))
44 print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
45 print("\nClassification Report:\n", classification_report(y_test, y_pred))
46 # ----- PLOTTING -----
47 # Plotting the Decision Tree
48 plt.figure(figsize=(15, 10))
49 plot_tree(model, feature_names=X.columns, class_names=model.classes_, filled=True)
50 plt.title('Decision Tree using ID3 Algorithm (IRIS Dataset)')
51 plt.show()
52 # ----- TESTING NEW DATA -----
53 # Predict for new input
54 sepal_length = float(input("\nEnter Sepal Length: "))
55 sepal_width = float(input("Enter Sepal Width: "))
56 petal_length = float(input("Enter Petal Length: "))
57 petal_width = float(input("Enter Petal Width: "))
58
59 # Prediction
60 new_prediction = model.predict([[sepal_length, sepal_width, petal_length, petal_width]])
61 print(f"\nPredicted Species: {new_prediction[0]}")
```

Machine Learning Lab (ECE-350P)

Output:

Terminal:

```
python3 -m exp4
```

Sample Data:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Dataset Info:

```
<class 'pandas.core.frame.DataFrame'>
```

RangeIndex: 150 entries, 0 to 149

Data columns (total 5 columns):

#	Column	Non-Null Count	Dtype
0	sepal_length	150 non-null	float64
1	sepal_width	150 non-null	float64
2	petal_length	150 non-null	float64
3	petal_width	150 non-null	float64
4	species	150 non-null	object

```
dtypes: float64(4), object(1)
```

```
memory usage: 6.0+ KB
```

None

Statistical Summary:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Class Distribution:

species	
setosa	50
versicolor	50
virginica	50

```
Name: count, dtype: int64
```

Machine Learning Lab (ECE-350P)

```
2025-03-11 20:49:22.932 python3[17642:703884] +[IMKClient subclass]: chose  
IMKClient_Modern2025-03-11 20:49:22.932 python3[17642:703884] +  
[IMKInputSession subclass]: chose IMKInputSession_Modern
```

Model Accuracy: 1.0

Confusion Matrix:

```
[[10  0  0]  
 [ 0  9  0]  
 [ 0  0 11]]
```

Classification Report:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	1.00	1.00	1.00	9
virginica	1.00	1.00	1.00	11
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Enter Sepal Length: 10

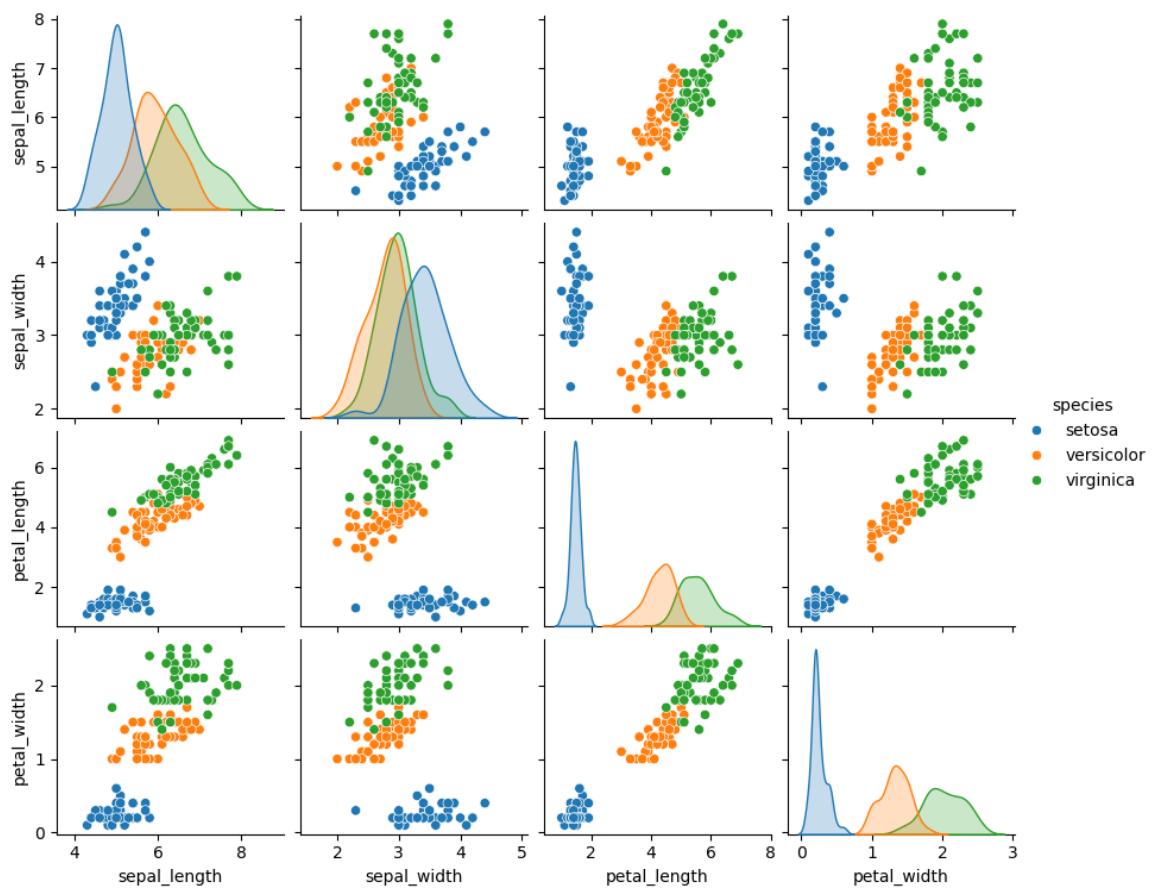
Enter Sepal Width: 2

Enter Petal Length: 5

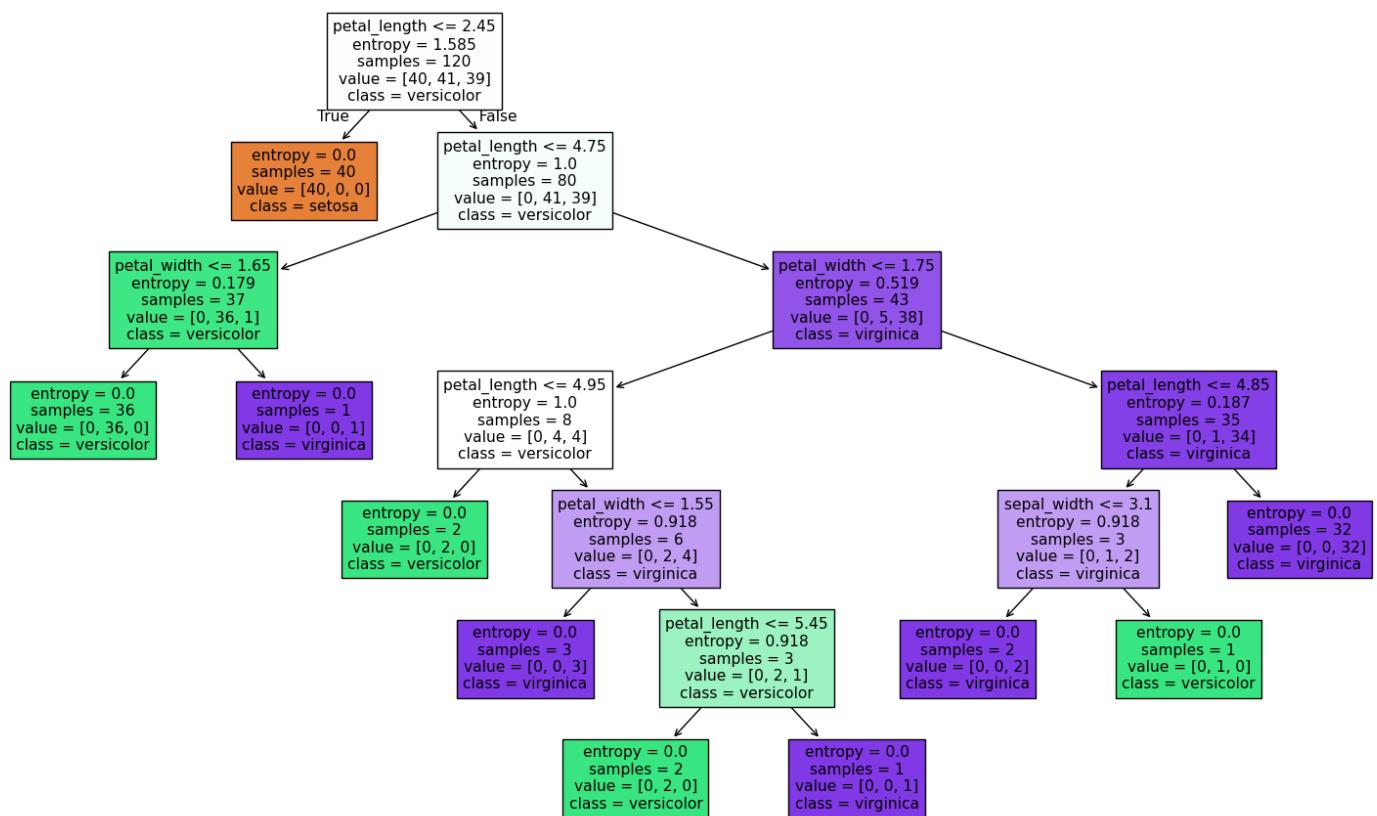
Enter Petal Width: 6

Predicted Species: virginica

Graph:



Decision Tree using ID3 Algorithm (IRIS Dataset)



Machine Learning Lab (ECE-350P)

Experiment 5

Aim: Write a program to demonstrate K-Nearest Neighbors(KNN) for flower classification

Software Used: Visual Studio Code

Code:

```
● ● ●
1 # Importing necessary libraries
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5 import seaborn as sns
6 from sklearn.datasets import load_iris
7 from sklearn.model_selection import train_test_split
8 from sklearn.neighbors import KNeighborsClassifier
9 from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
10 # ----- LOADING DATA -----
11 # Load the IRIS dataset
12 iris = load_iris()
13 df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
14 df['species'] = iris.target
15 df['species'] = df['species'].map({0: 'setosa', 1: 'versicolor', 2: 'virginica'})
16 # Display sample data
17 print("\nSample Data:\n", df.head())
18 # ----- DATA EXPLORATION -----
19 # Basic info about the dataset
20 print("\nDataset Info:")
21 print(df.info())
22 # Count of each class
23 print("\nClass Distribution:\n", df['species'].value_counts())
24 # Pair plot for visualization
25 sns.pairplot(df, hue='species')
26 plt.show()
27 # ----- DATA PREPROCESSING -----
28 # Split data into features (X) and target (y)
29 X = df.iloc[:, :-1]
30 y = df['species']
31 # Split into training and testing sets (80% training, 20% testing)
32 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
33 # ----- MODEL TRAINING -----
34 # Creating and training the KNN classifier (with k=5)
35 k = 5
36 model = KNeighborsClassifier(n_neighbors=k)
37 model.fit(X_train, y_train)
38 # ----- PREDICTION -----
39 # Making predictions
40 y_pred = model.predict(X_test)
41 # ----- MODEL EVALUATION -----
```

Machine Learning Lab (ECE-350P)

```
● ● ●
1 ----- MODEL EVALUATION -----
2 # Accuracy score
3 print("\nModel Accuracy:", accuracy_score(y_test, y_pred))
4 # Confusion matrix
5 print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
6 # Classification report
7 print("\nClassification Report:\n", classification_report(y_test, y_pred))
8 # ----- VISUALIZATION -----
9 # Visualizing the decision boundaries (for sepal length and width)
10 from matplotlib.colors import ListedColormap
11 X_plot = X.iloc[:, [0, 1]].values
12 y_plot = y.map({'setosa': 0, 'versicolor': 1, 'virginica': 2}).values
13 # Fitting the model for plotting
14 knn_plot = KNeighborsClassifier(n_neighbors=k)
15 knn_plot.fit(X_plot, y_plot)
16 # Generating mesh grid
17 x_min, x_max = X_plot[:, 0].min() - 1, X_plot[:, 0].max() + 1
18 y_min, y_max = X_plot[:, 1].min() - 1, X_plot[:, 1].max() + 1
19 xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
20                      np.arange(y_min, y_max, 0.1))
21 # Predict on the mesh grid
22 Z = knn_plot.predict(np.c_[xx.ravel(), yy.ravel()])
23 Z = Z.reshape(xx.shape)
24 # Plot decision boundary
25 plt.figure(figsize=(10, 6))
26 cmap_background = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
27 cmap_points = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
28 plt.contourf(xx, yy, Z, alpha=0.3, cmap=cmap_background)
29 plt.scatter(X_plot[:, 0], X_plot[:, 1], c=y_plot, cmap=cmap_points, edgecolor='k')
30 plt.title(f'KNN Decision Boundaries (k = {k})')
31 plt.xlabel('Sepal Length (cm)')
32 plt.ylabel('Sepal Width (cm)')
33 plt.show()
34 # ----- TESTING NEW DATA -----
35 # Predict for new input
36 sepal_length = float(input("\nEnter Sepal Length: "))
37 sepal_width = float(input("Enter Sepal Width: "))
38 petal_length = float(input("Enter Petal Length: "))
39 petal_width = float(input("Enter Petal Width: "))
40 # Prediction
41 new_prediction = model.predict([[sepal_length, sepal_width, petal_length, petal_width]])
42 print(f"\nPredicted Species: {new_prediction[0]}")
```

Output:**Terminal:**

```
python3 -m exp5
```

Sample Data:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
(cm) species				
0	5.1	3.5		1.4
0.2 setosa				
1	4.9	3.0		1.4
0.2 setosa				
2	4.7	3.2		1.3
0.2 setosa				
3	4.6	3.1		1.5
0.2 setosa				
4	5.0	3.6		1.4
0.2 setosa				

Dataset Info:

```
<class 'pandas.core.frame.DataFrame'>
```

RangeIndex: 150 entries, 0 to 149

Data columns (total 5 columns):

#	Column	Non-Null Count	Dtype
0	sepal length (cm)	150 non-null	float64
1	sepal width (cm)	150 non-null	float64
2	petal length (cm)	150 non-null	float64
3	petal width (cm)	150 non-null	float64
4	species	150 non-null	object

dtypes: float64(4), object(1)

memory usage: 6.0+ KB

None

Class Distribution:

species	
setosa	50
versicolor	50
virginica	50

Name: count, dtype: int64

2025-03-11 22:05:45.111 python3[18466:742049] +[IMKClient subclass]: chose IMKClient_Modern

2025-03-11 22:05:45.111 python3[18466:742049] +[IMKInputSession subclass]: chose IMKInputSession_Modern

Machine Learning Lab (ECE-350P)

```
2025-03-11 22:07:10.977 python3[18466:742049] The class 'NSSavePanel'  
overrides the method identifier. This method is implemented by class  
'NSWindow'
```

Model Accuracy: 1.0

Confusion Matrix:

```
[[10  0  0]  
 [ 0  9  0]  
 [ 0  0 11]]
```

Classification Report:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	1.00	1.00	1.00	9
virginica	1.00	1.00	1.00	11
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Enter Sepal Length: 10

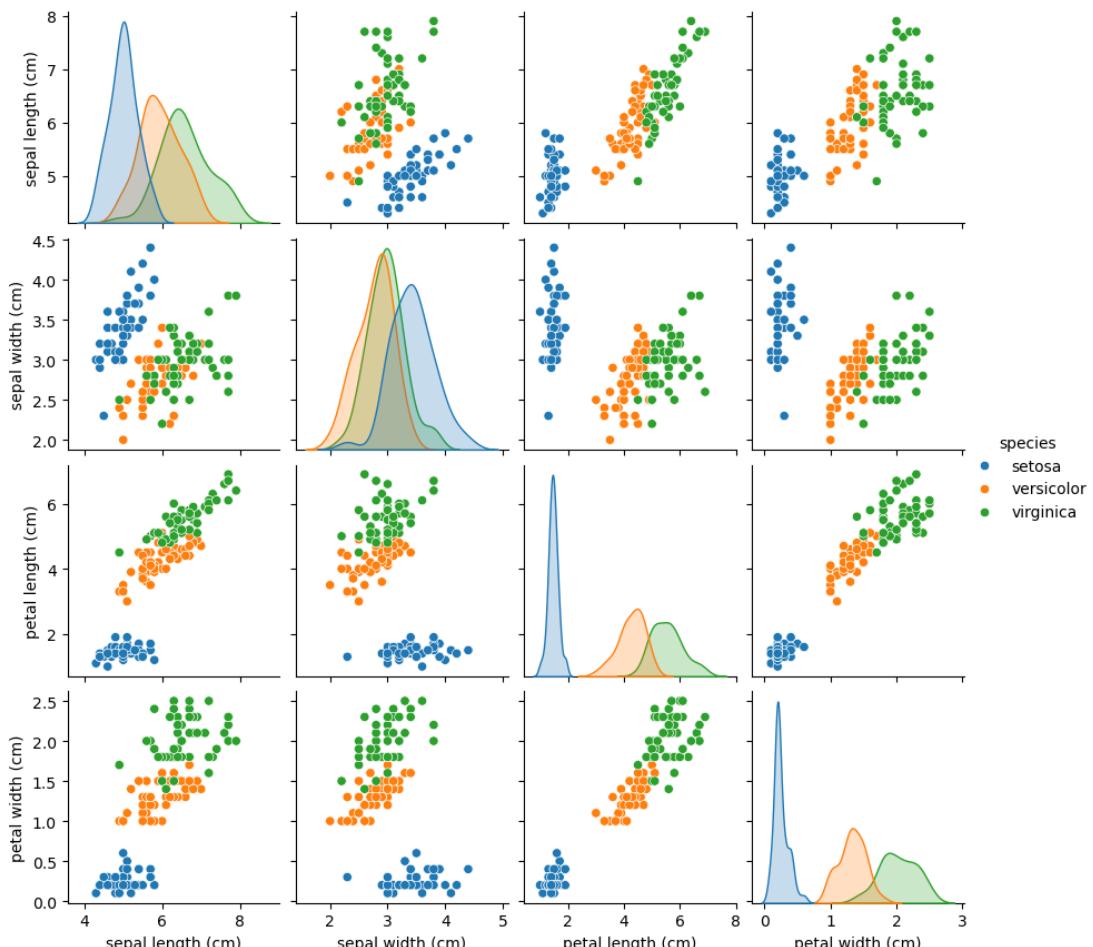
Enter Sepal Width: 20

Enter Petal Length: 5

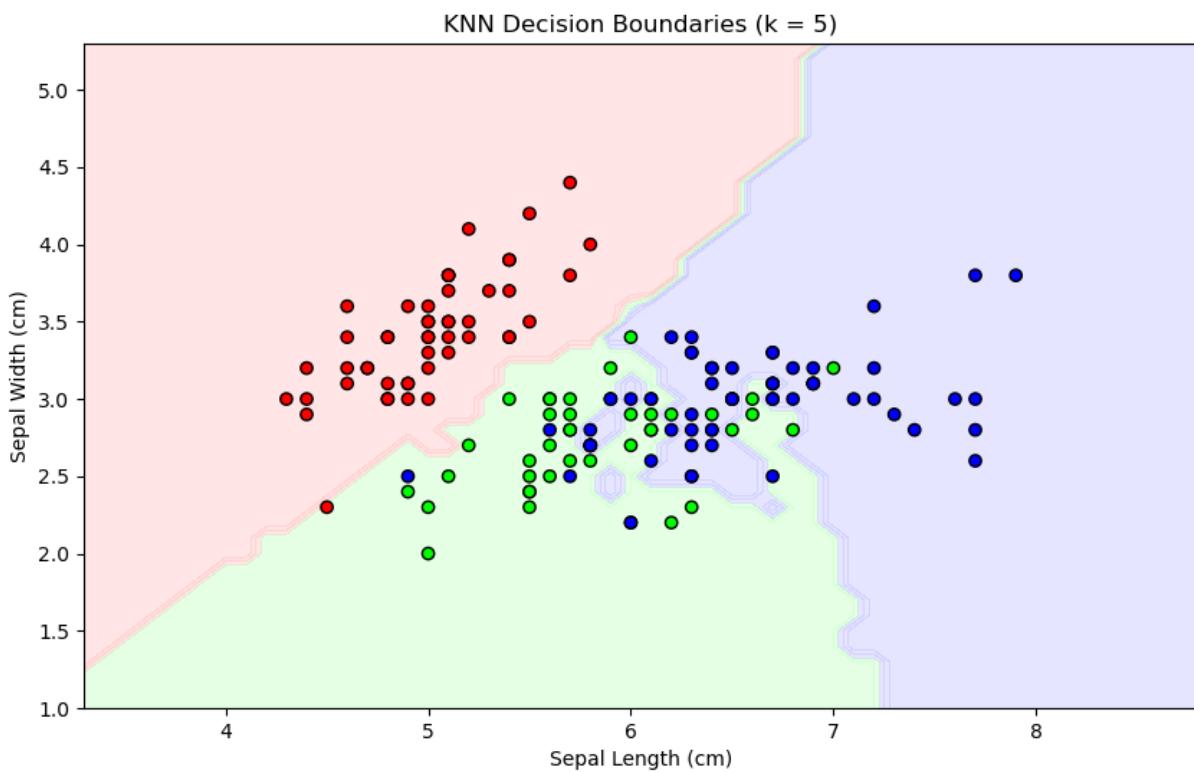
Enter Petal Width: 7

Predicted Species: virginica

Graph:



Machine Learning Lab (ECE-350P)



Machine Learning Lab (ECE-350P)

Experiment 6

Aim: Write a program to demonstrate Naive Bayes Classifier.

Software Used: Visual Studio Code

Code:

```
● ● ●

1 # Importing necessary libraries
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5 import seaborn as sns
6 from sklearn.datasets import load_iris
7 from sklearn.model_selection import train_test_split
8 from sklearn.naive_bayes import GaussianNB
9 from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
10 # ----- LOADING DATA -----
11 # Load the Iris dataset
12 iris = load_iris()
13 df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
14 df['species'] = iris.target
15 df['species'] = df['species'].map({0: 'setosa', 1: 'versicolor', 2: 'virginica'})
16 # Display sample data
17 print("\nSample Data:\n", df.head())
18 # ----- DATA EXPLORATION -----
19 # Basic info about the dataset
20 print("\nDataset Info:")
21 print(df.info())
22 # Count of each class
23 print("\nClass Distribution:\n", df['species'].value_counts())
24 # Pair plot for visualization
25 sns.pairplot(df, hue='species', height=2)
26 plt.show()
27 # ----- DATA PREPROCESSING -----
28 # Split data into features (X) and target (y)
29 X = df.iloc[:, :-1]
30 y = df['species']
31 # Split into training and testing sets (80% t
```

Machine Learning Lab (ECE-350P)

```
● ● ●
1 # ----- MODEL TRAINING -----
2 # Creating and training the Naive Bayes classifier
3 model = GaussianNB()
4 model.fit(X_train, y_train)
5 # ----- PREDICTION -----
6 # Making predictions
7 y_pred = model.predict(X_test)
8 # ----- MODEL EVALUATION -----
9 # Accuracy score
10 print("\nModel Accuracy:", accuracy_score(y_test, y_pred))
11 # Confusion matrix
12 print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
13 # Classification report
14 print("\nClassification Report:\n", classification_report(y_test, y_pred))
15 # ----- TESTING NEW DATA -----
16 # Predict for new input
17 sepal_length = float(input("\nEnter Sepal Length: "))
18 sepal_width = float(input("Enter Sepal Width: "))
19 petal_length = float(input("Enter Petal Length: "))
20 petal_width = float(input("Enter Petal Width: "))
21 # Prediction
22 new_prediction = model.predict([[sepal_length, sepal_width, petal_length, petal_width]])
23 print(f"\nPredicted Species: {new_prediction[0]}")
24 # ----- VISUALIZATION -----
25 # Visualize the relationship between sepal length and width
26 sns.set_style("whitegrid")
27 plt.figure(figsize=(8, 5))
28 sns.scatterplot(x='sepal_length', y='sepal_width', hue='species', data=df, s=100)
29 plt.title('Sepal Length vs Sepal Width')
30 plt.show()
31 # Visualize the relationship between petal length and width
32 plt.figure(figsize=(8, 5))
33 sns.scatterplot(x='petal_length', y='petal_width', hue='species', data=df, s=100)
34 plt.title('Petal Length vs Petal Width')
35 plt.show()
```

Machine Learning Lab (ECE-350P)

Output:

Terminal:

```
python3 -m exp6
```

```
Sample Data:
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
(cm) species				
0	5.1	3.5	1.4	
0.2 setosa				
1	4.9	3.0	1.4	
0.2 setosa				
2	4.7	3.2	1.3	
0.2 setosa				
3	4.6	3.1	1.5	
0.2 setosa				
4	5.0	3.6	1.4	
0.2 setosa				

```
Dataset Info:
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 150 entries, 0 to 149
```

```
Data columns (total 5 columns):
```

#	Column	Non-Null Count	Dtype
0	sepal length (cm)	150 non-null	float64
1	sepal width (cm)	150 non-null	float64
2	petal length (cm)	150 non-null	float64
3	petal width (cm)	150 non-null	float64
4	species	150 non-null	object

```
dtypes: float64(4), object(1)
```

```
memory usage: 6.0+ KB
```

```
None
```

```
Class Distribution:
```

```
  species
setosa      50
versicolor   50
virginica    50
```

```
Name: count, dtype: int64
```

```
2025-03-11 23:23:16.261 python3[19927:801210] +[IMKClient subclass]: chose
IMKClient_Modern
```

```
2025-03-11 23:23:16.261 python3[19927:801210] +[IMKInputSession subclass]: chose
IMKInputSession_Modern
```

```
Model Accuracy: 1.0
```

Machine Learning Lab (ECE-350P)

Confusion Matrix:

```
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

Classification Report:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	1.00	1.00	1.00	9
virginica	1.00	1.00	1.00	11
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Enter Sepal Length: 5.2

Enter Sepal Width: 2.2

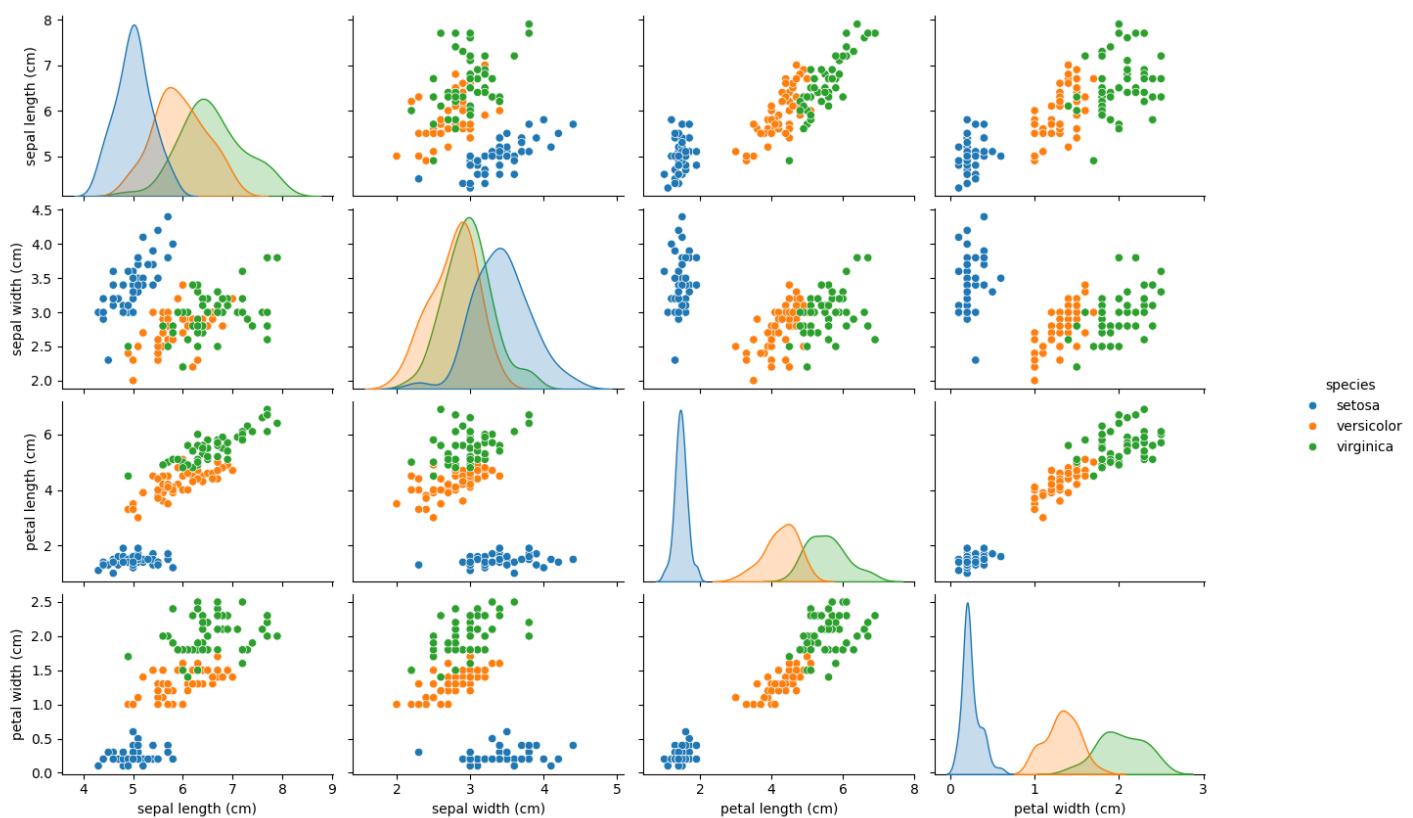
Enter Petal Length: 4

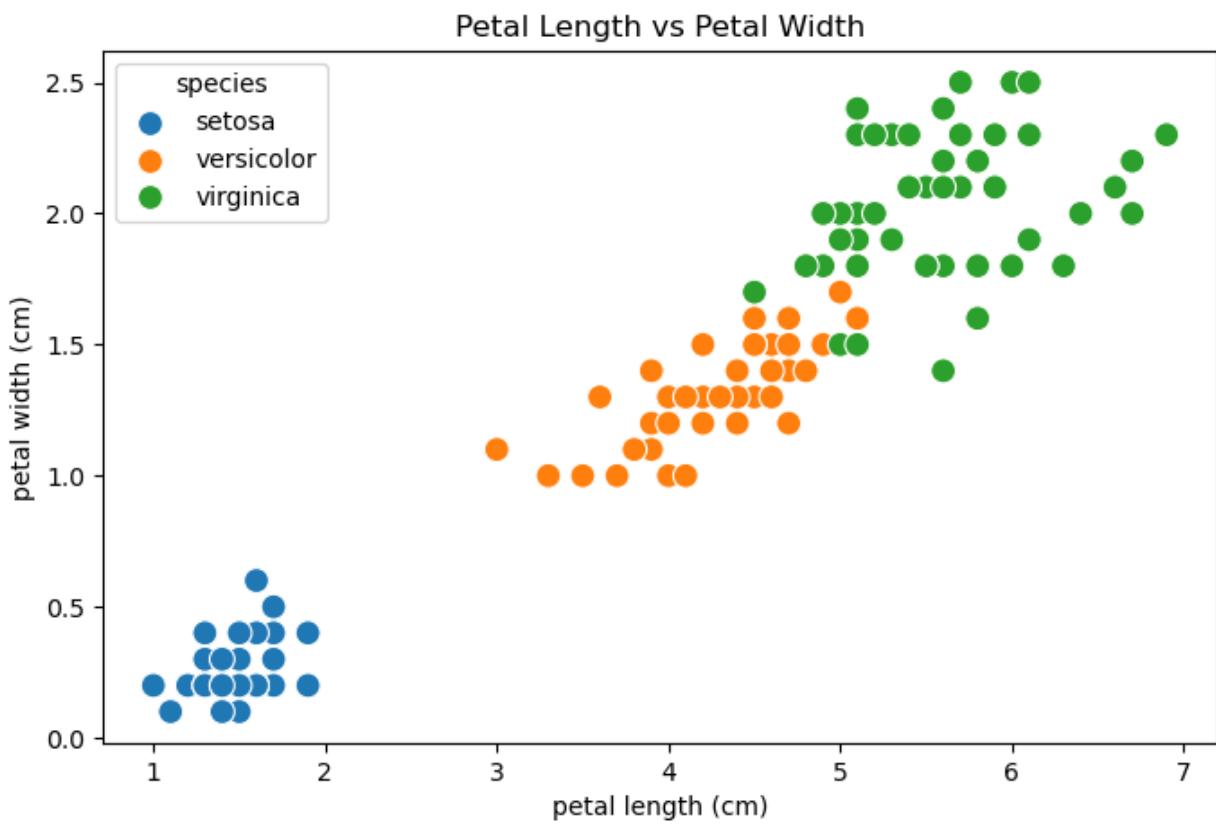
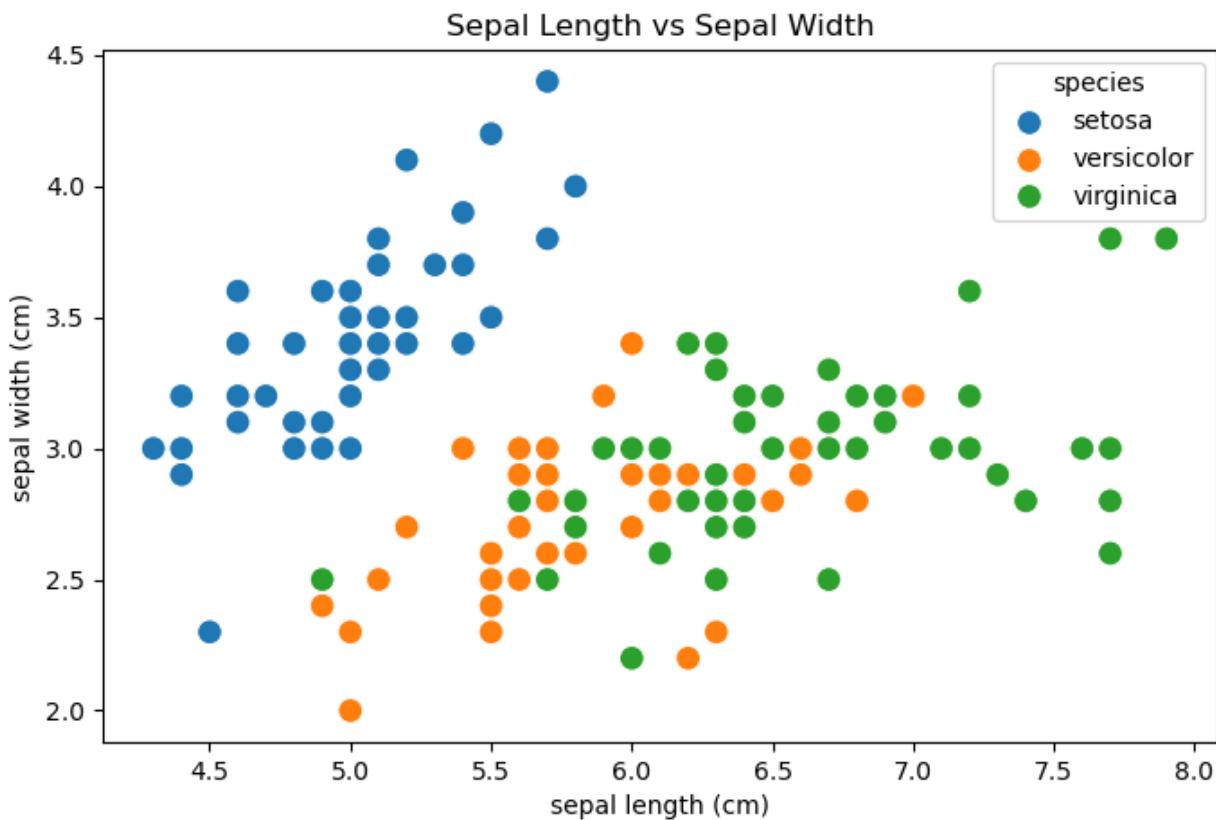
Enter Petal Width: 6.7

Predicted Species: virginica

```
2025-03-11 23:25:54.428 python3[19927:801210] The class 'NSSavePanel' overrides the method identifier. This method is implemented by class 'NSWindow'
```

Graph:





Machine Learning Lab (ECE-350P)

Experiment 7

Aim: Write a program to demonstrate PCA and LDA on the IRIS dataset.

Software Used: Visual Studio Code

Code:

```
● ● ●

1 # Importing necessary libraries
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5 import seaborn as sns
6 from sklearn.datasets import load_iris
7 from sklearn.decomposition import PCA
8 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
9 from sklearn.model_selection import train_test_split
10 from sklearn.naive_bayes import GaussianNB
11 from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
12 # ----- LOADING DATA -----
13 # Load the Iris dataset
14 iris = load_iris()
15 df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
16 df['species'] = iris.target
17 df['species'] = df['species'].map({0: 'setosa', 1: 'versicolor', 2: 'virginica'})
18 # Display sample data
19 print("\nSample Data:\n", df.head())
20 # ----- DATA PREPROCESSING -----
21 # Split data into features (X) and target (y)
22 X = df.iloc[:, :-1]
23 y = df['species']
24 # Split into training and testing sets (80% training, 20% testing)
25 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
26 # ----- PCA -----
27 # Apply PCA to reduce to 2 components
28 pca = PCA(n_components=2)
29 X_pca = pca.fit_transform(X)
30 # Convert PCA results to a DataFrame for visualization
31 pca_df = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
32 pca_df['species'] = y
33 # Plot PCA result
34 plt.figure(figsize=(8, 5))
35 sns.scatterplot(x='PC1', y='PC2', hue='species', data=pca_df, palette='viridis', s=100)
36 plt.title('PCA - Iris Dataset (2D)')
37 plt.show()
```

Machine Learning Lab (ECE-350P)

```
● ● ●
1 # ----- LDA -----
2 # Apply LDA to reduce to 2 components
3 lda = LDA(n_components=2)
4 X_lda = lda.fit_transform(X, y)
5 # Convert LDA results to a DataFrame for visualization
6 lda_df = pd.DataFrame(X_lda, columns=['LD1', 'LD2'])
7 lda_df['species'] = y
8 # Plot LDA result
9 plt.figure(figsize=(8, 5))
10 sns.scatterplot(x='LD1', y='LD2', hue='species', data=lda_df, palette='coolwarm', s=100)
11 plt.title('LDA - Iris Dataset (2D)')
12 plt.show()
13 # ----- MODEL TRAINING (ON PCA) -----
14 # Training the Naive Bayes model on PCA-transformed data
15 X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=42)
16 model_pca = GaussianNB()
17 model_pca.fit(X_train_pca, y_train)
18 # Predict using PCA data
19 y_pred_pca = model_pca.predict(X_test_pca)
20 # Evaluate PCA model
21 print("\nPCA Model Accuracy:", accuracy_score(y_test, y_pred_pca))
22 print("\nConfusion Matrix (PCA):\n", confusion_matrix(y_test, y_pred_pca))
23 print("\nClassification Report (PCA):\n", classification_report(y_test, y_pred_pca))
24 # ----- MODEL TRAINING (ON LDA) -----
25 # Training the Naive Bayes model on LDA-transformed data
26 X_train_lda, X_test_lda, y_train, y_test = train_test_split(X_lda, y, test_size=0.2, random_state=42)
27 model_lda = GaussianNB()
28 model_lda.fit(X_train_lda, y_train)
29 # Predict using LDA data
30 y_pred_lda = model_lda.predict(X_test_lda)
31 # Evaluate LDA model
32 print("\nLDA Model Accuracy:", accuracy_score(y_test, y_pred_lda))
33 print("\nConfusion Matrix (LDA):\n", confusion_matrix(y_test, y_pred_lda))
34 print("\nClassification Report (LDA):\n", classification_report(y_test, y_pred_lda))
```

Output:

```
python3 -m exp7
```

```
Sample Data:
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
(cm)	species			
0	5.1		3.5	1.4
0.2	setosa			
1	4.9		3.0	1.4
0.2	setosa			
2	4.7		3.2	1.3
0.2	setosa			
3	4.6		3.1	1.5
0.2	setosa	5.0	3.6	1.4
0.2	setosa			

Machine Learning Lab (ECE-350P)

```
2025-03-11 23:39:41.330 python3[20146:812343] +[IMKClient subclass]: chose  
IMKClient_Modern [REDACTED]  
2025-03-11 23:39:41.330 python3[20146:812343] +[IMKInputSession subclass]:  
chose IMKInputSession_Modern [REDACTED]  
2025-03-11 23:39:59.409 python3[20146:812343] The class 'NSSavePanel'  
overrides the method identifier. This method is implemented by class  
'NSWindow' [REDACTED]
```

PCA Model Accuracy: 0.9333333333333333

Confusion Matrix (PCA):

```
[[10  0  0]  
 [ 0  8  1]  
 [ 0  1 10]]
```

Classification Report (PCA):

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	0.89	0.89	0.89	9
virginica	0.91	0.91	0.91	11
accuracy			0.93	30
macro avg	0.93	0.93	0.93	30
weighted avg	0.93	0.93	0.93	30

LDA Model Accuracy: 1.0

Confusion Matrix (LDA):

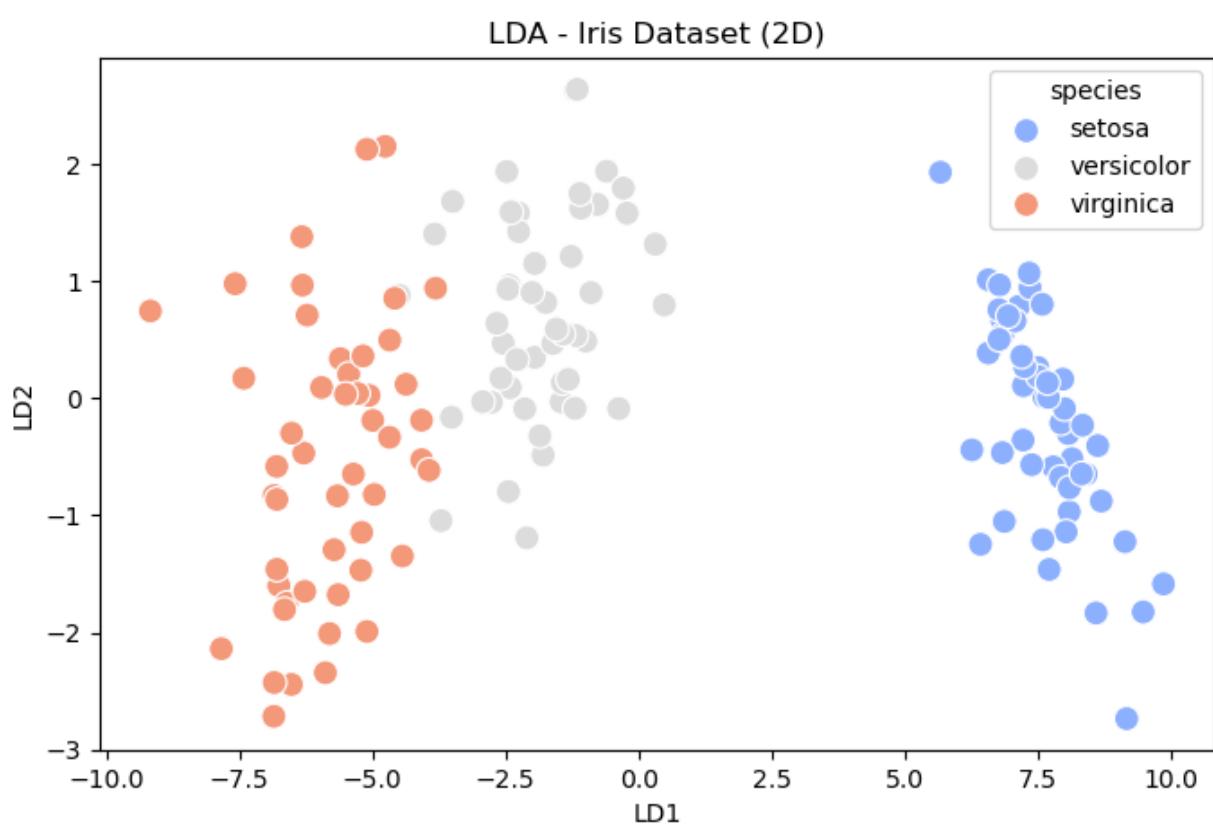
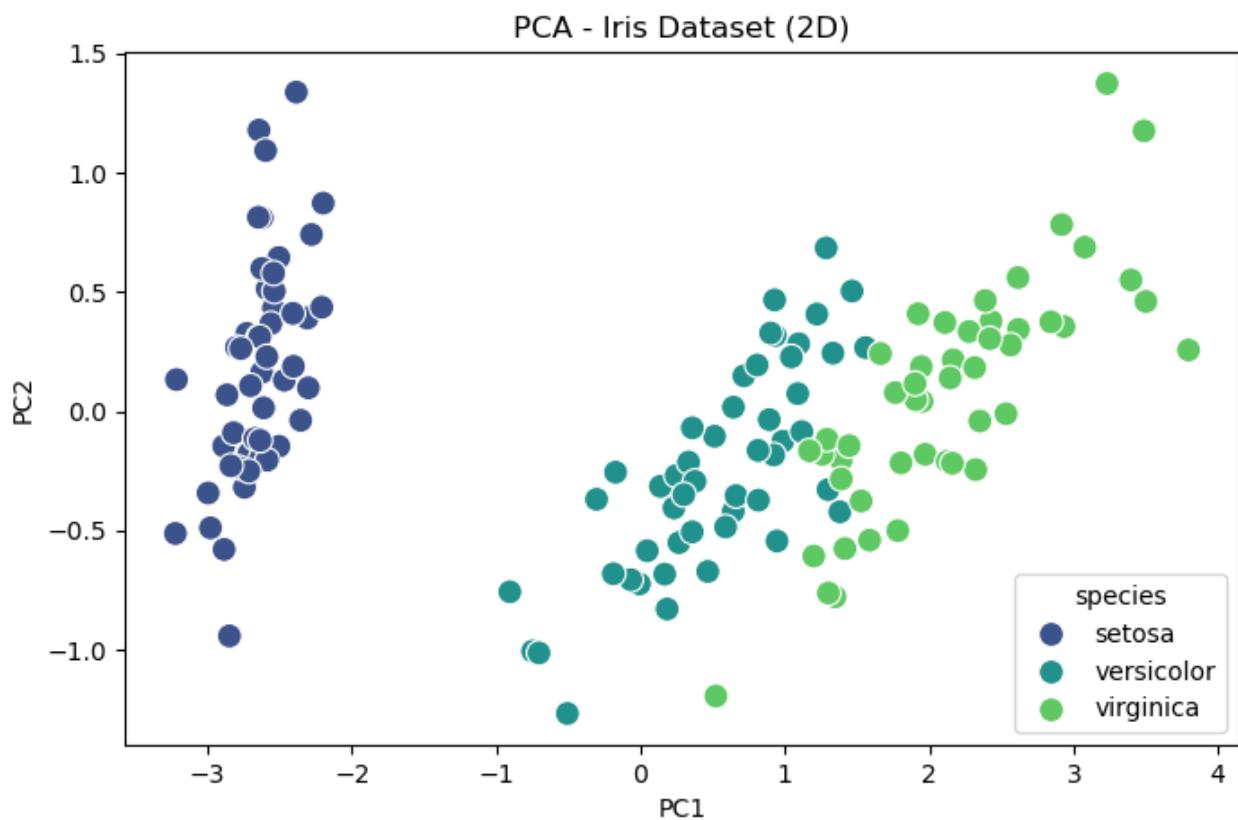
```
[[10  0  0]  
 [ 0  9  0]  
 [ 0  0 11]]
```

Classification Report (LDA):

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	1.00	1.00	1.00	9
virginica	1.00	1.00	1.00	11
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Machine Learning Lab (ECE-350P)

Graph:



Machine Learning Lab (ECE-350P)

Experiment 8

Aim: Write a program to demonstrate DBSCAN (Density-Based Spatial Clustering of Applications with Noise) clustering algorithms.

Software Used: Visual Studio Code

Code:

```
● ● ●
1 # Importing necessary libraries
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5 import seaborn as sns
6 from sklearn.datasets import load_iris
7 from sklearn.preprocessing import StandardScaler
8 from sklearn.cluster import DBSCAN
9 from sklearn.decomposition import PCA
10 # ----- LOADING DATA -----
11 # Load the Iris dataset
12 iris = load_iris()
13 df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
14 # Display sample data
15 print("\nSample Data:\n", df.head())
16 # ----- DATA PREPROCESSING -----
17 # Scale the data for better clustering results
18 scaler = StandardScaler()
19 scaled_data = scaler.fit_transform(df)
20 # ----- APPLYING DBSCAN -----
21 # Apply DBSCAN clustering
22 dbscan = DBSCAN(eps=0.5, min_samples=5)
23 clusters = dbscan.fit_predict(scaled_data)
24 # Add cluster labels to the dataset
25 df['cluster'] = clusters
26 # ----- VISUALIZATION (PCA) -----
27 # Reduce dimensions using PCA for visualization
28 pca = PCA(n_components=2)
29 pca_result = pca.fit_transform(scaled_data)
30 df['PC1'] = pca_result[:, 0]
31 df['PC2'] = pca_result[:, 1]
32 # Plot clusters using PCA components
33 plt.figure(figsize=(8, 5))
34 sns.scatterplot(x='PC1', y='PC2', hue='cluster', data=df, palette='viridis', s=100)
35 plt.title('DBSCAN Clustering on Iris Dataset (PCA Reduced)')
36 plt.show()
37 # ----- EVALUATION -----
38 # Number of clusters and noise points
39 n_clusters = len(set(clusters)) - (1 if -1 in clusters else 0)
40 n_noise = list(clusters).count(-1)
41 print(f"\nNumber of clusters: {n_clusters}")
42 print(f"Number of noise points: {n_noise}")
43 # Display cluster assignments
44 print("\nCluster Assignments:")
45 print(df[['PC1', 'PC2', 'cluster']].head(10))
```

Machine Learning Lab (ECE-350P)

Output:

Terminal:

```
python3 -m exp8
```

Sample Data:

	sepal length (cm) (cm)	sepal width (cm) (cm)	petal length (cm) (cm)	petal width (cm)
0	5.1	3.5	1.4	
1	4.9	3.0	1.4	
2	4.7	3.2	1.3	
3	4.6	3.1	1.5	
4	5.0	3.6	1.4	

```
2025-03-12 00:11:58.344 python3[21002:839157] +[IMKClient subclass]: chose IMKClient_Modern
2025-03-12 00:11:58.344 python3[21002:839157] +[IMKInputSession subclass]: chose IMKInputSession_Modern
2025-03-12 00:12:30.628 python3[21002:839157] The class 'NSSavePanel' overrides the method identifier. This method is implemented by class 'NSWindow'
```

Number of clusters: 2

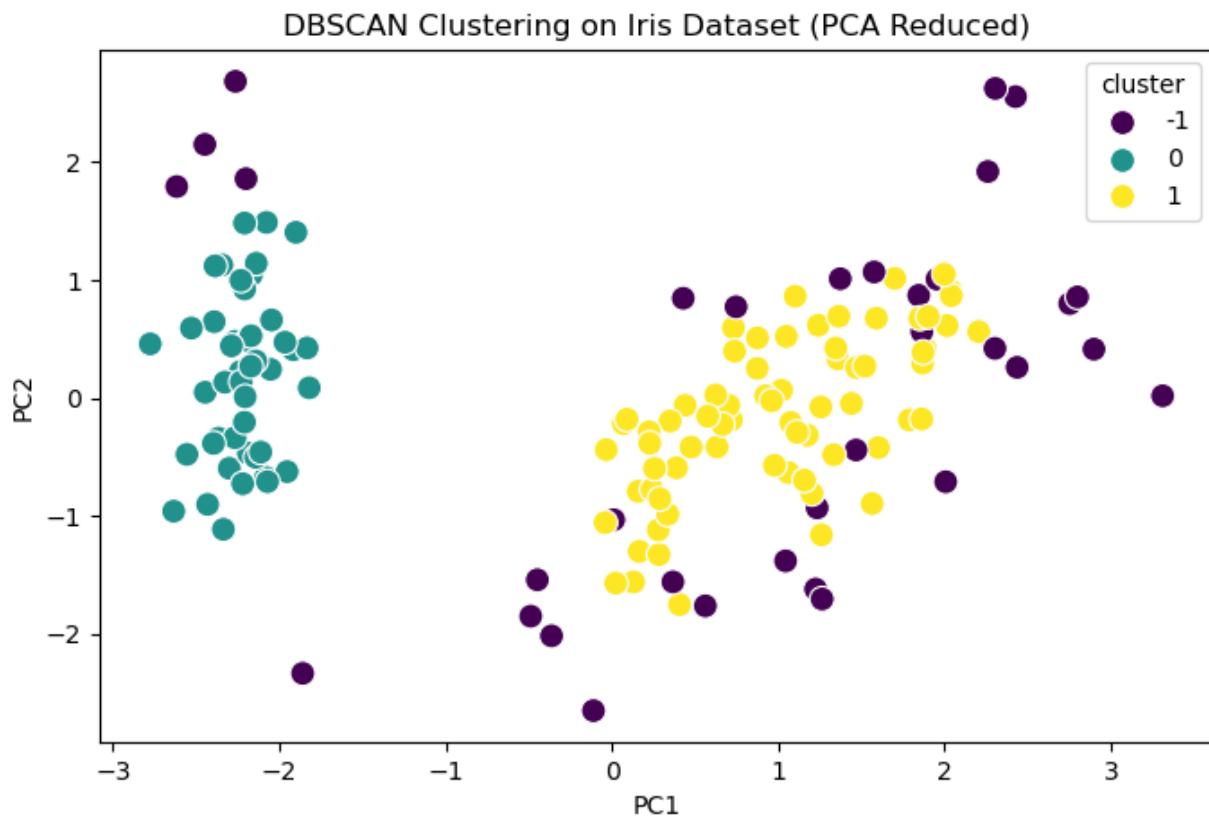
Number of noise points: 34

Cluster Assignments:

	PC1	PC2	cluster
0	-2.264703	0.480027	0
1	-2.080961	-0.674134	0
2	-2.364229	-0.341908	0
3	-2.299384	-0.597395	0
4	-2.389842	0.646835	0
5	-2.075631	1.489178	0
6	-2.444029	0.047644	0
7	-2.232847	0.223148	0
8	-2.334640	-1.115328	0
9	-2.184328	-0.469014	0

Machine Learning Lab (ECE-350P)

Graph:



Experiment 9

Aim: Write a Program to demonstrate K-Medoids clustering algorithm.

Software Used: Visual Studio Code

Code:

```
● ● ●
1 # Importing necessary libraries
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5 import seaborn as sns
6 from sklearn.datasets import load_iris
7 from sklearn.preprocessing import StandardScaler
8 from sklearn_extra.cluster import KMedoids
9 from sklearn.decomposition import PCA
10 # ----- LOADING DATA -----
11 # Load the Iris dataset
12 iris = load_iris()
13 df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
14 # Display sample data
15 print("\nSample Data:\n", df.head())
16 # ----- DATA PREPROCESSING -----
17 # Scale the data for better clustering results
18 scaler = StandardScaler()
19 scaled_data = scaler.fit_transform(df)
20 # ----- APPLYING K-MEDOIDS -----
21 # Apply K-Medoids clustering with 3 clusters (since we have 3 iris species)
22 kmedoids = KMedoids(n_clusters=3, random_state=42)
23 clusters = kmedoids.fit_predict(scaled_data)
24 # Add cluster labels to the dataset
25 df['cluster'] = clusters
```

```

● ● ●
1 # ----- VISUALIZATION (PCA) -----
2 # Reduce dimensions using PCA for visualization
3 pca = PCA(n_components=2)
4 pca_result = pca.fit_transform(scaled_data)
5 df['PC1'] = pca_result[:, 0]
6 df['PC2'] = pca_result[:, 1]
7 # Plot clusters using PCA components
8 plt.figure(figsize=(8, 5))
9 sns.scatterplot(x='PC1', y='PC2', hue='cluster', data=df, palette='viridis', s=100)
10 plt.scatter(pca_result[kmedoids.medoid_indices_, 0],
11               pca_result[kmedoids.medoid_indices_, 1],
12               color='red', label='Medoids', s=150, marker='X')
13 plt.title('K-Medoids Clustering on Iris Dataset (PCA Reduced)')
14 plt.legend()
15 plt.show()
16 # ----- EVALUATION -----
17 # Number of clusters and size of each cluster
18 n_clusters = len(set(clusters))
19 cluster_sizes = pd.Series(clusters).value_counts()
20 print(f"\nNumber of clusters: {n_clusters}")
21 print("\nCluster Sizes:")
22 print(cluster_sizes)
23 # Display cluster assignments
24 print("\nCluster Assignments:")
25 print(df[['PC1', 'PC2', 'cluster']].head(10))

```

Output:

Terminal:

```

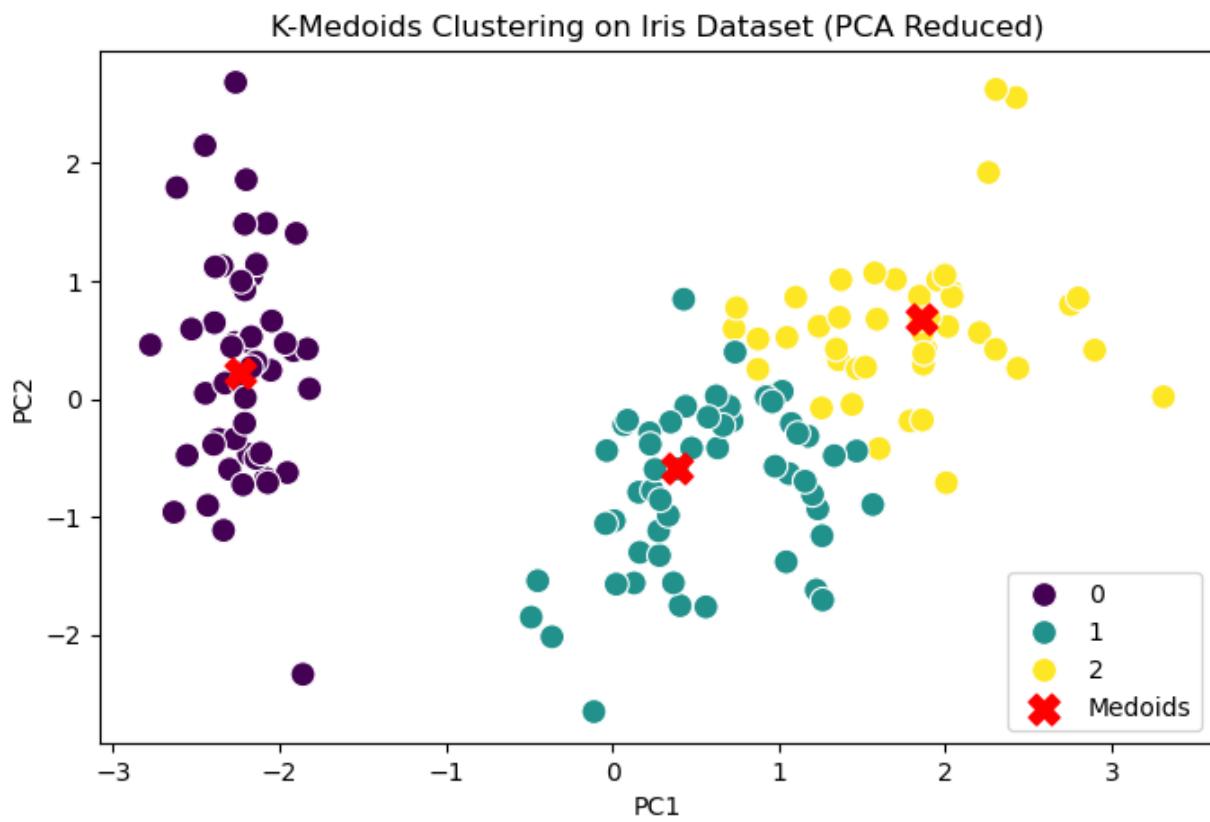
python3 -m exp9
Sample Data:
    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width
(cmall)
0                  5.1                  3.5                 1.4
0.2
1                  4.9                  3.0                 1.4
0.2
2                  4.7                  3.2                 1.3
0.2
3                  4.6                  3.1                 1.5
0.2

```

Machine Learning Lab (ECE-350P)

```
4          5.0          3.6          1.4
0.2 [REDACTED]
2025-03-12 00:25:19.035 python3[21318:850424] +[IMKClient subclass]: chose
IMKClient_Modern [REDACTED]
2025-03-12 00:25:19.035 python3[21318:850424] +[IMKInputSession subclass]:
chose IMKInputSession_Modern [REDACTED]
2025-03-12 00:25:56.752 python3[21318:850424] The class 'NSSavePanel'
overrides the method identifier. This method is implemented by class
'NSWindow' [REDACTED]
Number of clusters: 3
Cluster Sizes:
1    56 [REDACTED]
0    50 [REDACTED]
2    44 [REDACTED]
Name: count, dtype: int64
Cluster Assignments:
   PC1      PC2  cluster
0 -2.264703  0.480027      0
1 -2.080961 -0.674134      0
2 -2.364229 -0.341908      0
3 -2.299384 -0.597395      0
4 -2.389842  0.646835      0
5 -2.075631  1.489178      0
6 -2.444029  0.047644      0
7 -2.232847  0.223148      0
8 -2.334640 -1.115328      0
9 -2.184328 -0.469014      0
```

Graph:



Machine Learning Lab (ECE-350P)

Experiment 10

Aim: Write a program to demonstrate K-Means clustering on Handwritten Dataset.

Software Used: Visual Studio Code

Code:

```
● ● ●
1 # Importing necessary libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 from sklearn.datasets import load_digits
6 from sklearn.preprocessing import StandardScaler
7 from sklearn.cluster import KMeans
8 from sklearn.decomposition import PCA
9 # ----- LOADING DATA -----
10 # Load the Handwritten Digits dataset
11 digits = load_digits()
12 X = digits.data
13 y = digits.target
14 # Display sample data
15 plt.figure(figsize=(8, 4))
16 for i in range(10):
17     plt.subplot(2, 5, i + 1)
18     plt.imshow(digits.images[i], cmap='gray')
19     plt.axis('off')
20 plt.suptitle('Sample Handwritten Digits')
21 plt.show()
22 # ----- DATA PREPROCESSING -----
23 # Scale the data for better clustering performance
24 scaler = StandardScaler()
25 X_scaled = scaler.fit_transform(X)
26 # ----- APPLYING K-MEANS -----
27 # Apply K-Means clustering with 10 clusters (since we have digits 0-9)
28 kmeans = KMeans(n_clusters=10, random_state=42)
29 clusters = kmeans.fit_predict(X_scaled)
```

```
● ● ●
1 # ----- VISUALIZATION (PCA) -----
2 # Reduce dimensions using PCA for visualization
3 pca = PCA(n_components=2)
4 X_pca = pca.fit_transform(X_scaled)
5 # Create a DataFrame for plotting
6 import pandas as pd
7 df = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
8 df['cluster'] = clusters
9 # Plot clusters using PCA components
10 plt.figure(figsize=(8, 5))
11 sns.scatterplot(x='PC1', y='PC2', hue='cluster', data=df, palette='tab10', s=100)
12 plt.title('K-Means Clustering on Handwritten Digits (PCA Reduced)')
13 plt.legend(title='Cluster')
14 plt.show()
15 # ----- EVALUATION -----
16 # Count the number of samples in each cluster
17 cluster_sizes = pd.Series(clusters).value_counts()
18 print("\nNumber of samples in each cluster:")
19 print(cluster_sizes)
20 # Display cluster assignments for the first 10 samples
21 print("\nCluster Assignments:")
22 print(df[['PC1', 'PC2', 'cluster']].head(10))
23 # ----- DISPLAY CLUSTER CENTERS -----
24 # Display cluster centers as images
25 centers = kmeans.cluster_centers_.reshape(10, 8, 8)
26 plt.figure(figsize=(8, 4))
27 for i, center in enumerate(centers):
28     plt.subplot(2, 5, i + 1)
29     plt.imshow(center, cmap='gray')
30     plt.axis('off')
31 plt.suptitle('Cluster Centers as Handwritten Digits')
32 plt.show()
```

Output:

Terminal:

```
python3 -m exp10
2025-03-12 00:42:37.363 python3[21572:863295] +[IMKClient subclass]: chose
IMKClient_Modern
2025-03-12 00:42:37.363 python3[21572:863295] +[IMKInputSession subclass]: chose
IMKInputSession_Modern
```

Machine Learning Lab (ECE-350P)

```
2025-03-12 00:42:53.432 python3[21572:863295] The class 'NSSavePanel'  
overrides the method identifier. This method is implemented by class  
'NSWindow'
```

```
Number of samples in each cluster:
```

```
7      413  
9      259  
2      232  
3      194  
4      180  
5      178  
6      170  
1      139  
0      30  
8      2
```

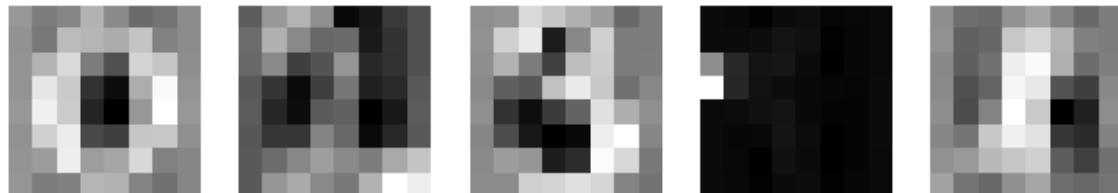
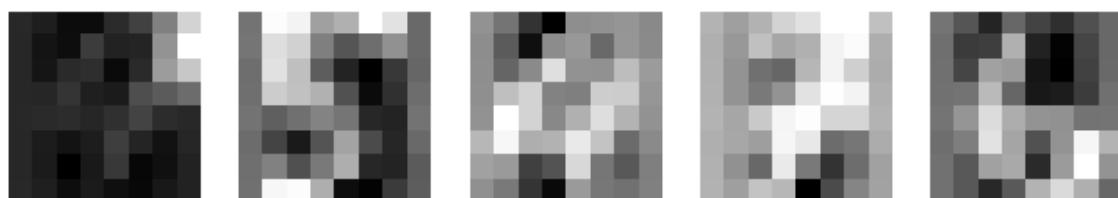
```
Name: count, dtype: int64
```

```
Cluster Assignments:
```

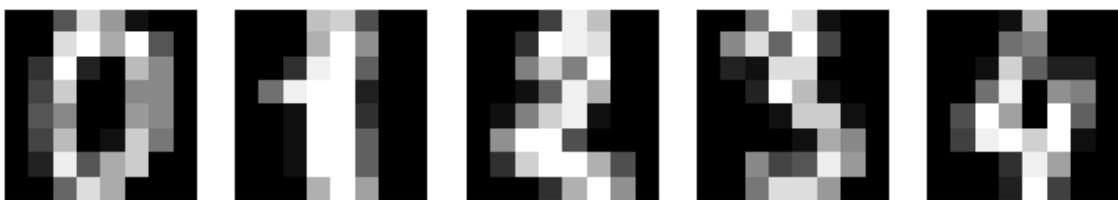
	PC1	PC2	cluster
0	-1.914214	-0.954502	5
1	-0.588980	0.924636	9
2	-1.302039	-0.317189	9
3	3.020770	-0.868772	7
4	-4.528949	-1.093480	2
5	1.301890	-1.148183	7
6	-1.434222	-2.957845	4
7	-0.614240	5.462184	3
8	1.098451	-0.665823	9
9	0.537970	-0.777493	7

Graph:

Cluster Centers as Handwritten Digits



Sample Handwritten Digits



K-Means Clustering on Handwritten Digits (PCA Reduced)

