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.

- **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 condo 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
```

Experiment 2

Aim: Write a program to demonstrate Simple Linear Regression

Software Used: Visual Studio Code

Code:

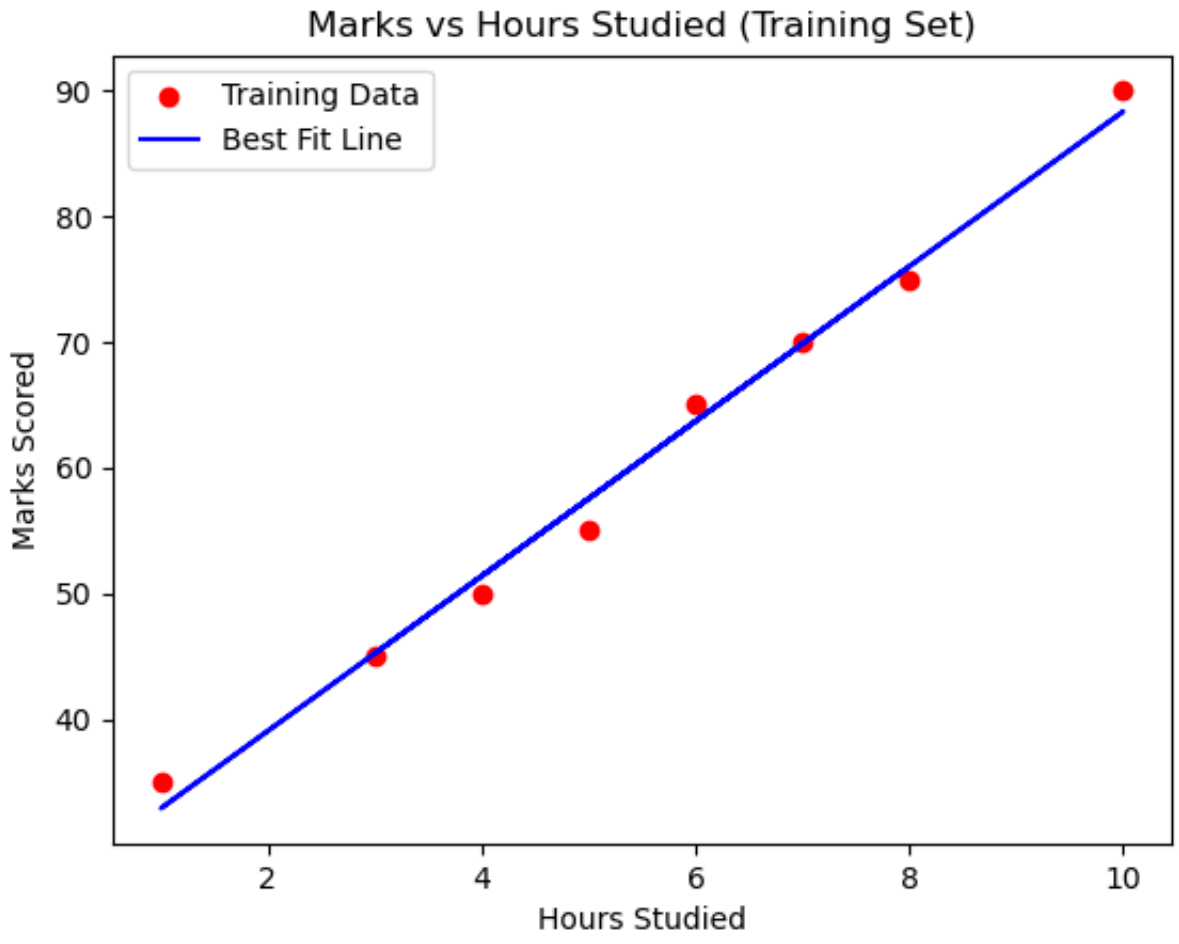
```python
# Importing necessary libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Sample dataset (Hours Studied vs Marks Scored)
data = {
    'HoursStudied': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'MarksScored': [35, 40, 45, 50, 55, 65, 70, 75, 80, 90]
}

# Creating a DataFrame
df = pd.DataFrame(data)

# Splitting the data into training and testing sets
X = df[['HoursStudied']]  # Independent variable
y = df['MarksScored']     # Dependent variable

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

# Creating and training the model
model = LinearRegression()
model.fit(X_train, y_train)

# Making predictions
y_pred = model.predict(X_test)

# Visualizing the training set results
plt.scatter(X_train, y_train, color='red', label='Training Data')
plt.plot(X_train, model.predict(X_train), color='blue', label='Best Fit Line')
plt.title('Marks vs Hours Studied (Training Set)')
plt.xlabel('Hours Studied')
plt.ylabel('Marks Scored')
plt.legend()
plt.show()

# Visualizing the test set results
plt.scatter(X_test, y_test, color='red', label='Test Data')
plt.plot(X_train, model.predict(X_train), color='blue', label='Best Fit Line')
plt.title('Marks vs Hours Studied (Test Set)')
plt.xlabel('Hours Studied')
plt.ylabel('Marks Scored')
plt.legend()
plt.show()

# Model Evaluation
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R-squared Score:", r2_score(y_test, y_pred))

# Coefficients
print("Slope (Coefficient):", model.coef_[0])
print("Intercept:", model.intercept_)

# Prediction example
hours = float(input("\nEnter hours studied: "))
predicted_marks = model.predict([[hours]])
print(f"Predicted Marks for {hours} hours of study: {predicted_marks[0]:.2f}")
```

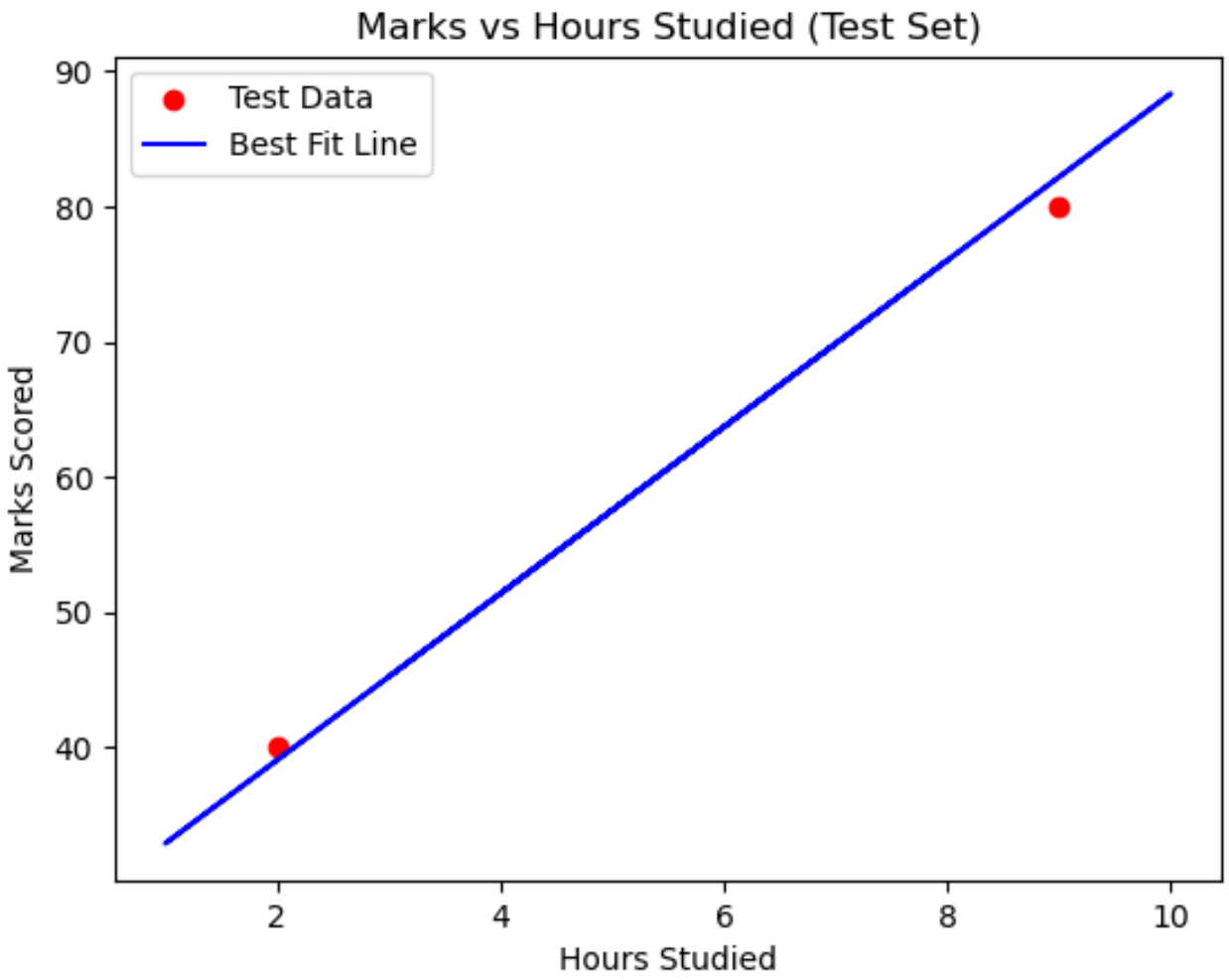
Output:

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:




Marks vs Hours Studied (Test Set)
Test Data
Best Fit Line
Marks Scored
Hours Studied
90
80
70
60
50
40
2
4
6
8
10

Experiment 3

Aim: Write a program to demonstrate Logistic Regression.

Software Used: Visual Studio Code

Code:

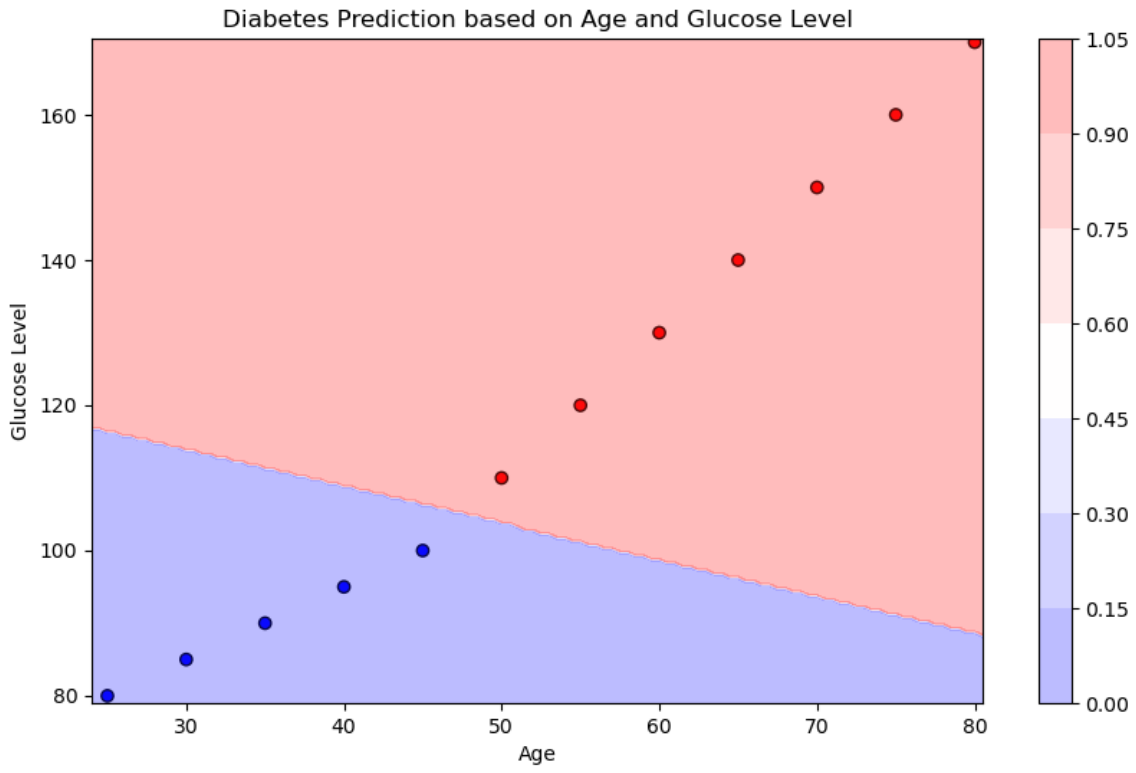
```python
# Importing necessary libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Sample dataset (Age, Glucose Level, Diabetes)
data = {
    'Age': [25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80],
    'GlucoseLevel': [80, 85, 90, 95, 100, 110, 120, 130, 140, 150, 160, 170],
    'Diabetes': [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1]  # 0 = No Diabetes, 1 = Diabetes
}

# Creating a DataFrame
df = pd.DataFrame(data)

# Independent variables (features)
X = df[['Age', 'GlucoseLevel']]
# Dependent variable (target)
y = df['Diabetes']

# Splitting the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Creating and training the Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Making predictions
y_pred = model.predict(X_test)

# Model Evaluation
print("\nModel Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Predict for new input
age = float(input("\nEnter age: "))
glucose = float(input("Enter glucose level: "))
prediction = model.predict([[age, glucose]])
result = 'Diabetic' if prediction[0] == 1 else 'Non-Diabetic'
print(f"\nPrediction for Age = {age} and Glucose Level = {glucose}: {result}")

# Visualizing the decision boundary
plt.figure(figsize=(10, 6))
plt.scatter(df['Age'], df['GlucoseLevel'], c=df['Diabetes'], cmap='bwr', edgecolors='k')
plt.title('Diabetes Prediction based on Age and Glucose Level')
plt.xlabel('Age')
plt.ylabel('Glucose Level')

# Plot decision boundary
x_min, x_max = X['Age'].min() - 1, X['Age'].max() + 1
y_min, y_max = X['GlucoseLevel'].min() - 1, X['GlucoseLevel'].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.5),
                     np.arange(y_min, y_max, 0.5))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.3, cmap='bwr')
plt.colorbar()

plt.show()
```

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:



Experiment 4

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

Software Used: Visual Studio Code

Code:

```python
# Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# ------------------------- LOADING DATA -------------------------
# Load the IRIS dataset
url = 'https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv'
df = pd.read_csv(url)
# Display first few rows of the dataset
print("\nSample Data:\n", df.head())
# ------------------------- DATA EXPLORATION -------------------------
# Basic information about the data
print("\nDataset Info:")
print(df.info())
# Summary statistics of numeric features
print("\nStatistical Summary:")
print(df.describe())
# Count of each class
print("\nClass Distribution:\n", df['species'].value_counts())
# ------------------------- VISUALIZATION -------------------------
# Pair plot to visualize relationships between features
sns.pairplot(df, hue='species')
plt.show()
# ------------------------- DATA PREPROCESSING -------------------------
# Splitting data into features (X) and target (y)
X = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
y = df['species']
# Splitting into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# ------------------------- MODEL TRAINING -------------------------
# Creating and training the Decision Tree classifier using ID3 algorithm (criterion='entropy')
model = DecisionTreeClassifier(criterion='entropy', random_state=42)
model.fit(X_train, y_train)
# ------------------------- PREDICTION -------------------------
# Making predictions
y_pred = model.predict(X_test)
# ------------------------- MODEL EVALUATION -------------------------
# Evaluating the model
print("\nModel Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
# ------------------------- PLOTTING -------------------------
# Plotting the Decision Tree
plt.figure(figsize=(15, 10))
plot_tree(model, feature_names=X.columns, class_names=model.classes_, filled=True)
plt.title('Decision Tree using ID3 Algorithm (IRIS Dataset)')
plt.show()
# ------------------------- TESTING NEW DATA -------------------------
# Predict for new input
sepal_length = float(input("\nEnter Sepal Length: "))
sepal_width = float(input("Enter Sepal Width: "))
petal_length = float(input("Enter Petal Length: "))
petal_width = float(input("Enter Petal Width: "))

# Prediction
new_prediction = model.predict([[sepal_length, sepal_width, petal_length, petal_width]])
print(f"\nPredicted Species: {new_prediction[0]}")
```

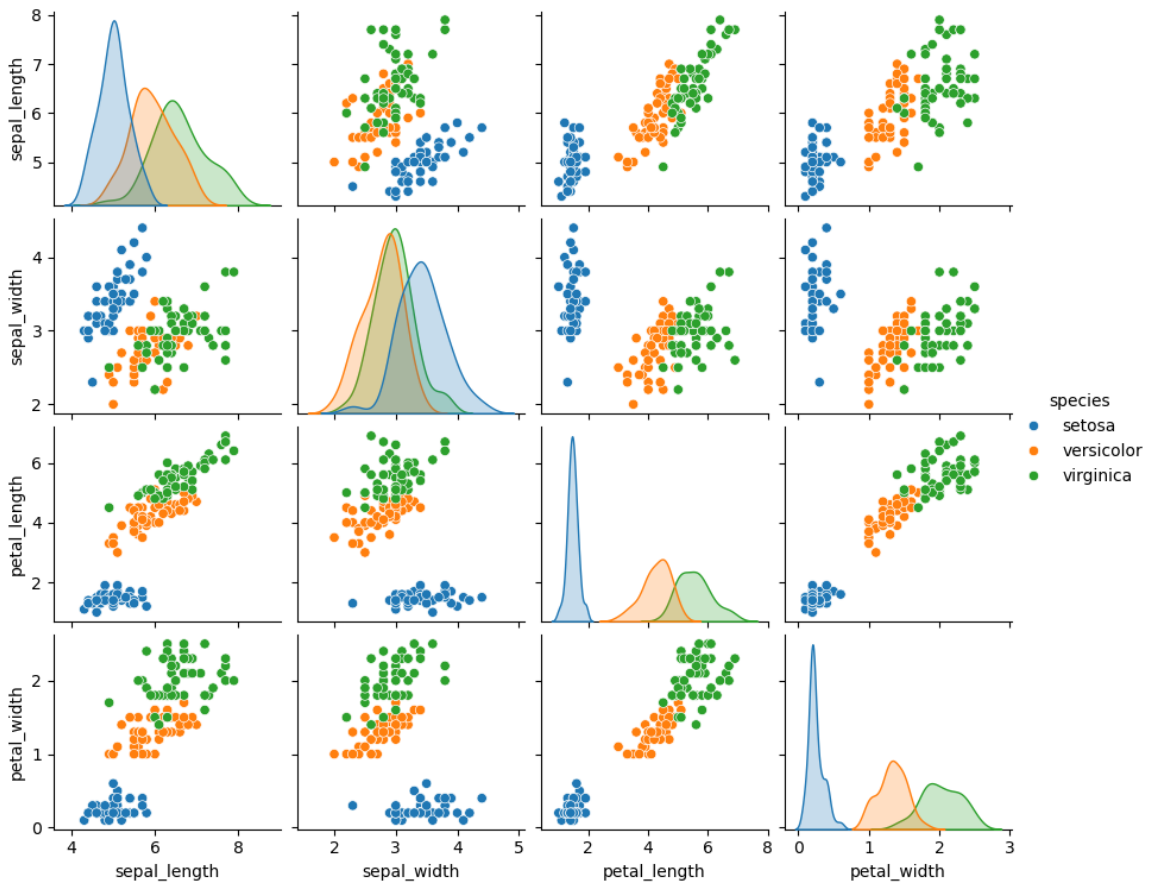
Output:

Terminal:

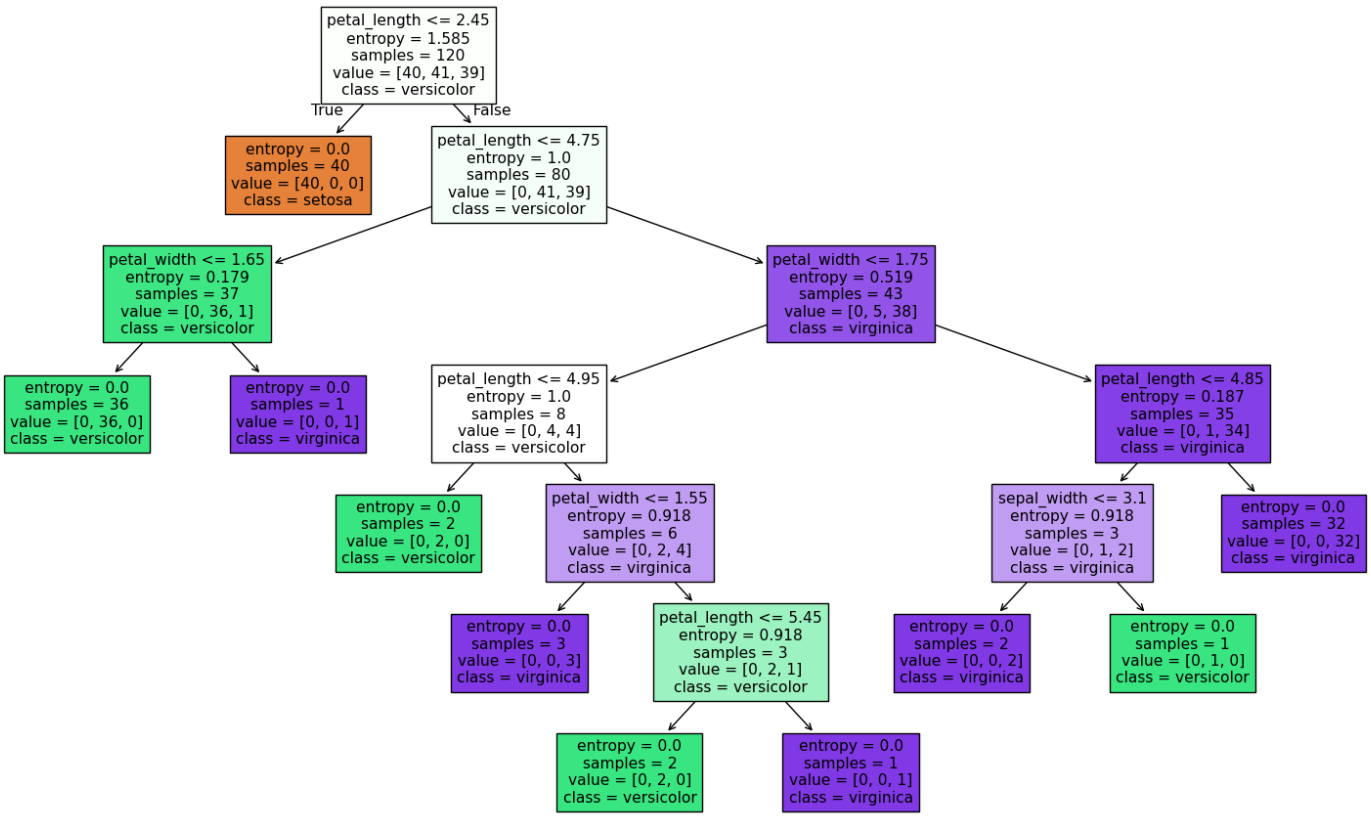
```
python3 -m exp4
Sample Data:
    sepal_length  sepal_width  petal_length  petal_width species
0            5.1          3.5           1.4          0.2  setosa
1            4.9          3.0           1.4          0.2  setosa
2            4.7          3.2           1.3          0.2  setosa
3            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
```

```
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)



Experiment 5

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

Software Used: Visual Studio Code

Code:

```python
# Importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
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, confusion_matrix, classification_report
# ------------------------- LOADING DATA -------------------------
# Load the IRIS dataset
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['species'] = iris.target
df['species'] = df['species'].map({0: 'setosa', 1: 'versicolor', 2: 'virginica'})
# Display sample data
print("\nSample Data:\n", df.head())
# ------------------------- DATA EXPLORATION -------------------------
# Basic info about the dataset
print("\nDataset Info:")
print(df.info())
# Count of each class
print("\nClass Distribution:\n", df['species'].value_counts())
# Pair plot for visualization
sns.pairplot(df, hue='species')
plt.show()
# ------------------------- DATA PREPROCESSING -------------------------
# Split data into features (X) and target (y)
X = df.iloc[:, :-1]
y = df['species']
# Split into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# ------------------------- MODEL TRAINING -------------------------
# Creating and training the KNN classifier (with k=5)
k = 5
model = KNeighborsClassifier(n_neighbors=k)
model.fit(X_train, y_train)
# ------------------------- PREDICTION -------------------------
# Making predictions
y_pred = model.predict(X_test)
# ------------------------- MODEL EVALUATION ------
```

```python
---------- MODEL EVALUATION -------------------------
# Accuracy score
print("\nModel Accuracy:", accuracy_score(y_test, y_pred))
# Confusion matrix
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
# Classification report
print("\nClassification Report:\n", classification_report(y_test, y_pred))
# ------------------------- VISUALIZATION ----------------------------
# Visualizing the decision boundaries (for sepal length and width)
from matplotlib.colors import ListedColormap
X_plot = X.iloc[:, [0, 1]].values
y_plot = y.map({'setosa': 0, 'versicolor': 1, 'virginica': 2}).values
# Fitting the model for plotting
knn_plot = KNeighborsClassifier(n_neighbors=k)
knn_plot.fit(X_plot, y_plot)
# Generating mesh grid
x_min, x_max = X_plot[:, 0].min() - 1, X_plot[:, 0].max() + 1
y_min, y_max = X_plot[:, 1].min() - 1, X_plot[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))
# Predict on the mesh grid
Z = knn_plot.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
# Plot decision boundary
plt.figure(figsize=(10, 6))
cmap_background = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
cmap_points = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
plt.contourf(xx, yy, Z, alpha=0.3, cmap=cmap_background)
plt.scatter(X_plot[:, 0], X_plot[:, 1], c=y_plot, cmap=cmap_points, edgecolor='k')
plt.title(f'KNN Decision Boundaries (k = {k})')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.show()
# ------------------------- TESTING NEW DATA -------------------------
# Predict for new input
sepal_length = float(input("\nEnter Sepal Length: "))
sepal_width = float(input("Enter Sepal Width: "))
petal_length = float(input("Enter Petal Length: "))
petal_width = float(input("Enter Petal Width: "))
# Prediction
new_prediction = model.predict([[sepal_length, sepal_width, petal_length, petal_width]])
print(f"\nPredicted Species: {new_prediction[0]}")
```

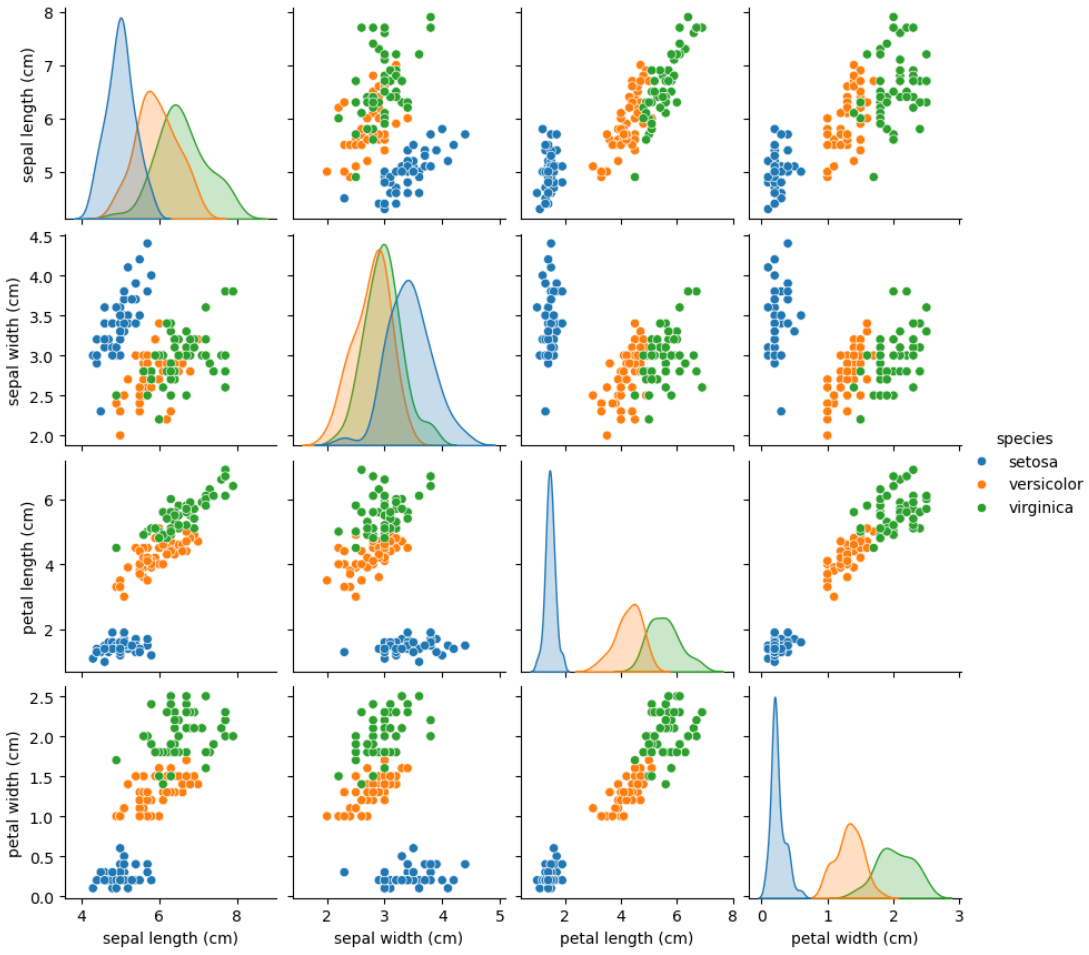
Output:

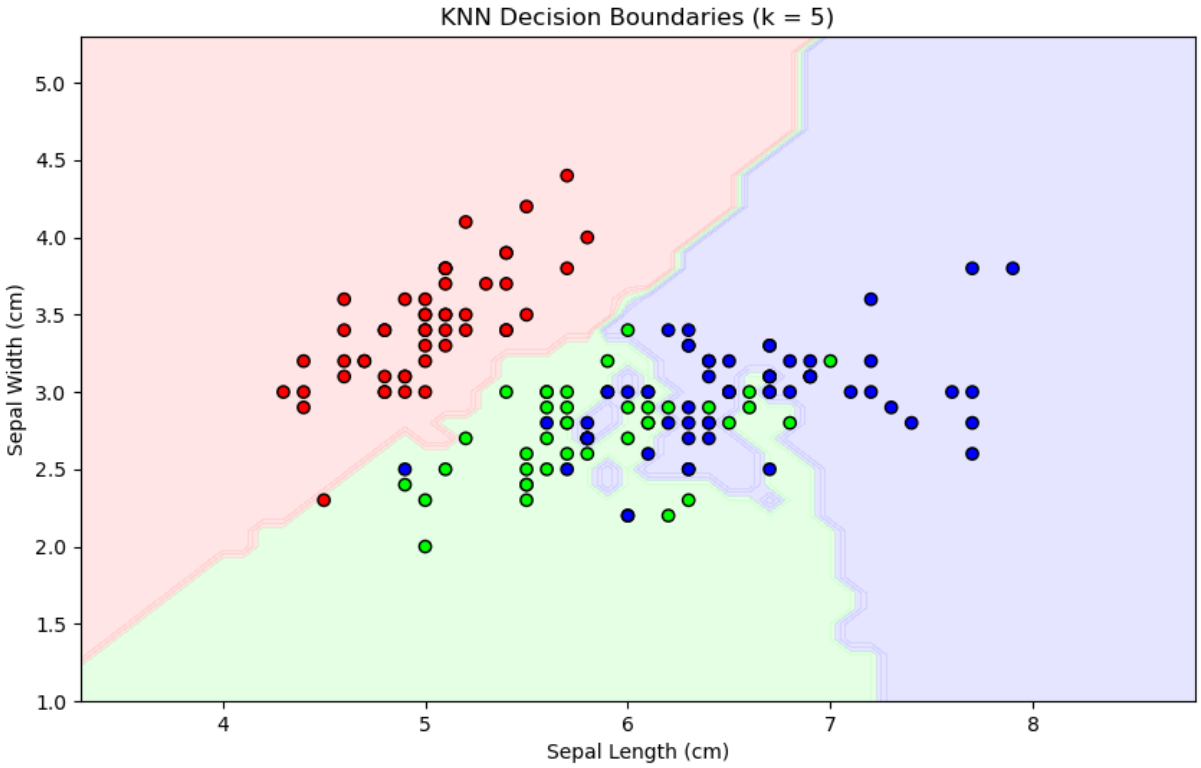
Terminal:

```
python3 -m exp5
Sample Data:
    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width
(cm) 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
```

```
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:




KNN Decision Boundaries (k = 5)
Sepal Width (cm)
Sepal Length (cm)
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8

Experiment 6

Aim: Write a program to demonstrate Naive Bayes Classifier.

Software Used: Visual Studio Code

Code:

```python
# Importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# ------------------------- LOADING DATA -------------------------
# Load the Iris dataset
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['species'] = iris.target
df['species'] = df['species'].map({0: 'setosa', 1: 'versicolor', 2: 'virginica'})
# Display sample data
print("\nSample Data:\n", df.head())
# ------------------------- DATA EXPLORATION -------------------------
# Basic info about the dataset
print("\nDataset Info:")
print(df.info())
# Count of each class
print("\nClass Distribution:\n", df['species'].value_counts())
# Pair plot for visualization
sns.pairplot(df, hue='species', height=2)
plt.show()
# ------------------------- DATA PREPROCESSING -------------------------
# Split data into features (X) and target (y)
X = df.iloc[:, :-1]
y = df['species']
# Split into training and testing sets (80% t
```

```python
# ------------------------- MODEL TRAINING -------------------------
# Creating and training the Naive Bayes classifier
model = GaussianNB()
model.fit(X_train, y_train)
# ------------------------- PREDICTION -------------------------
# Making predictions
y_pred = model.predict(X_test)
# ------------------------- MODEL EVALUATION -------------------------
# Accuracy score
print("\nModel Accuracy:", accuracy_score(y_test, y_pred))
# Confusion matrix
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
# Classification report
print("\nClassification Report:\n", classification_report(y_test, y_pred))
# ------------------------- TESTING NEW DATA -------------------------
# Predict for new input
sepal_length = float(input("\nEnter Sepal Length: "))
sepal_width = float(input("Enter Sepal Width: "))
petal_length = float(input("Enter Petal Length: "))
petal_width = float(input("Enter Petal Width: "))
# Prediction
new_prediction = model.predict([[sepal_length, sepal_width, petal_length, petal_width]])
print(f"\nPredicted Species: {new_prediction[0]}")
# ------------------------- VISUALIZATION -------------------------
# Visualize the relationship between sepal length and width
sns.set_style("whitegrid")
plt.figure(figsize=(8, 5))
sns.scatterplot(x='sepal_length', y='sepal_width', hue='species', data=df, s=100)
plt.title('Sepal Length vs Sepal Width')
plt.show()
# Visualize the relationship between petal length and width
plt.figure(figsize=(8, 5))
sns.scatterplot(x='petal_length', y='petal_width', hue='species', data=df, s=100)
plt.title('Petal Length vs Petal Width')
plt.show()
```

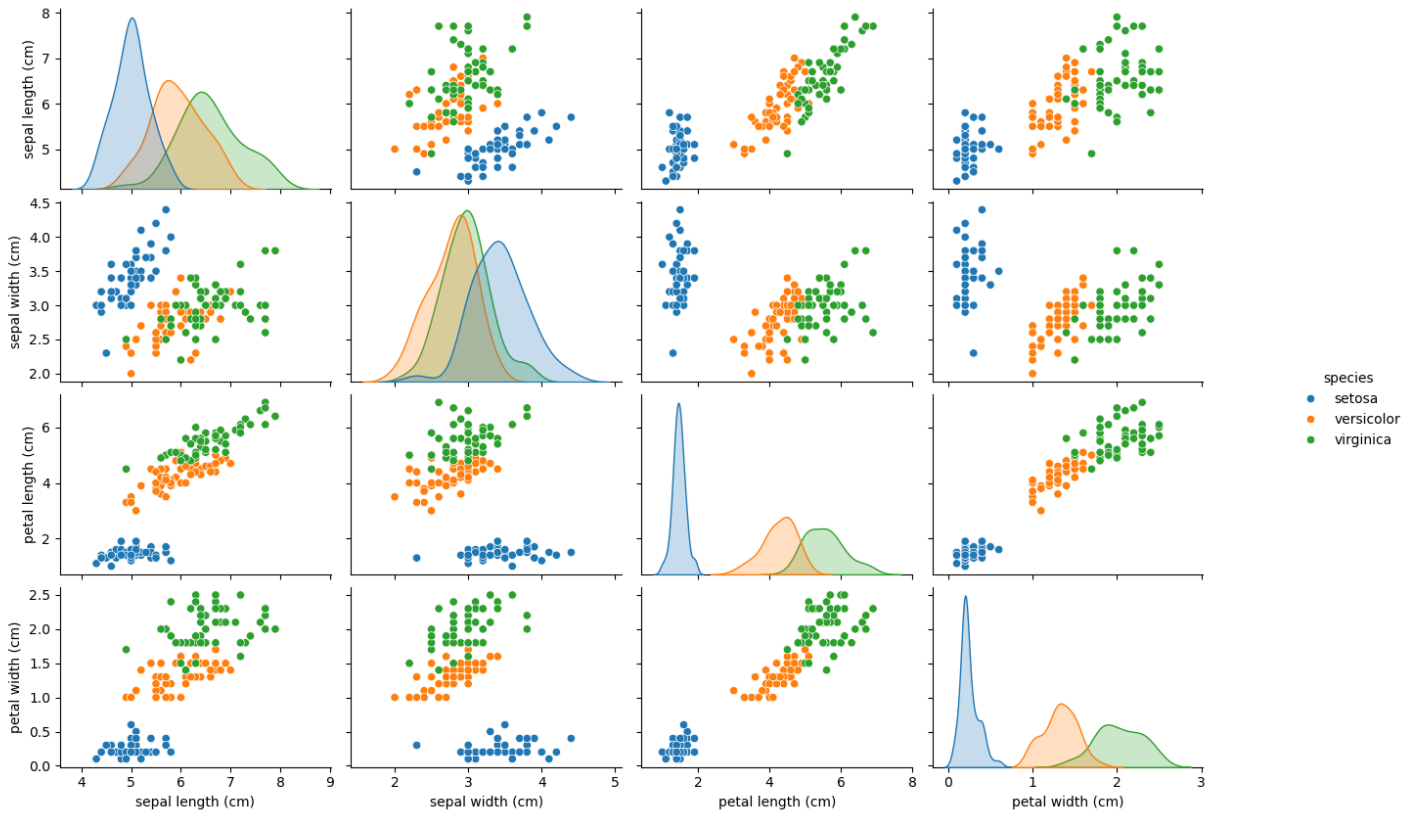
Output:

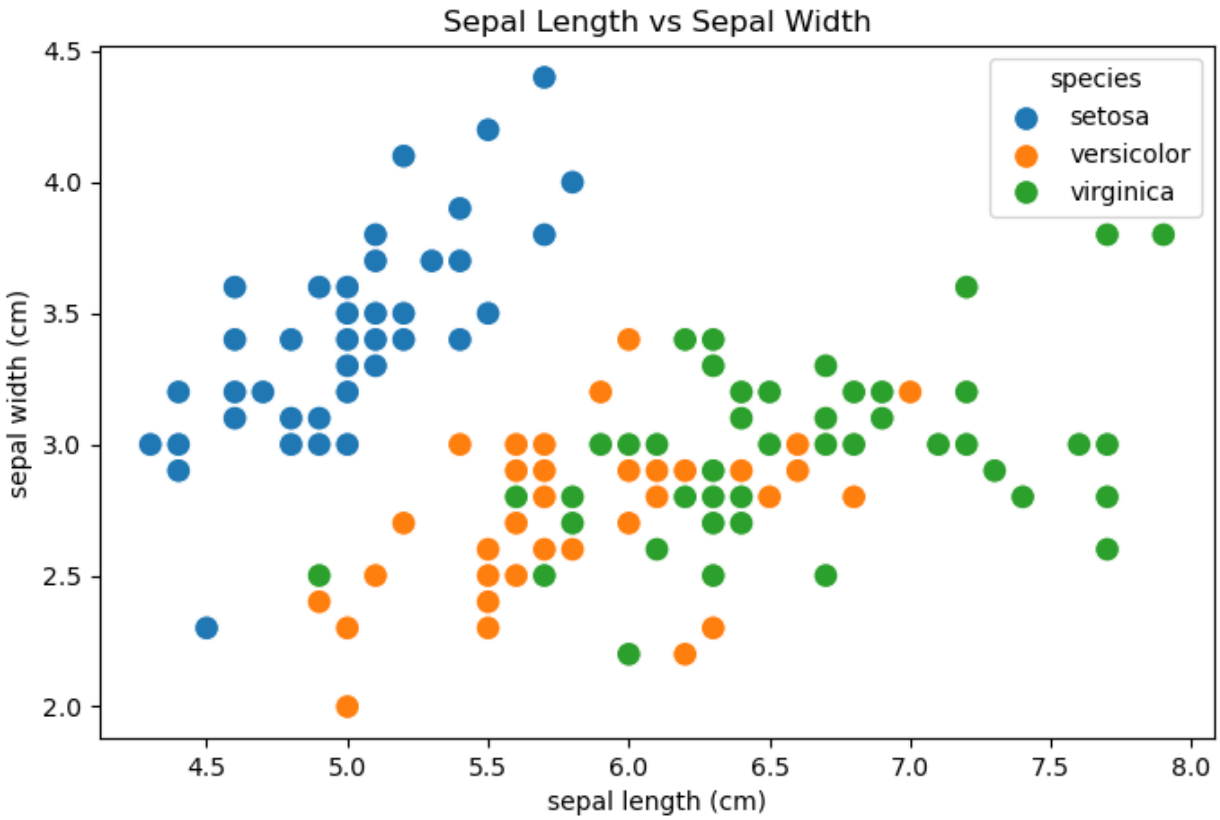
Terminal:

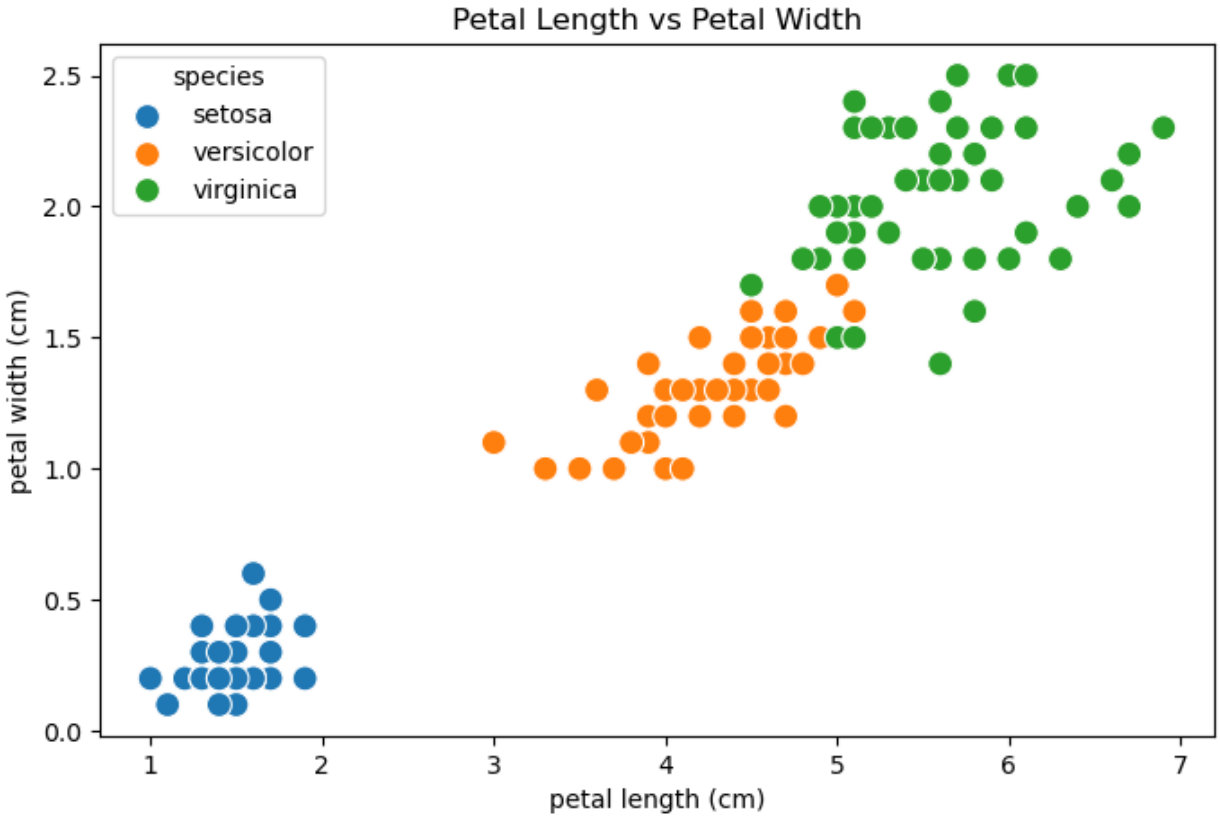
```
python3 -m exp6
Sample Data:
    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width
(cm) species
0                5.1               3.5                1.4
0.2  setosa
1                4.9               3.0                1.4
0.2  setosa
2                4.7               3.2                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
```

```
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:




Sepal Length vs Sepal Width
species
setosa
versicolor
virginica
sepal width (cm)
sepal length (cm)
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0


Petal Length vs Petal Width
species
setosa
versicolor
virginica
petal width (cm)
petal length (cm)
2.5
2.0
1.5
1.0
0.5
0.0
1
2
3
4
5
6
7

Experiment 7

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

Software Used: Visual Studio Code

Code:

```python
# Importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# ------------------------- LOADING DATA -------------------------
# Load the Iris dataset
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['species'] = iris.target
df['species'] = df['species'].map({0: 'setosa', 1: 'versicolor', 2: 'virginica'})
# Display sample data
print("\nSample Data:\n", df.head())
# ------------------------- DATA PREPROCESSING -------------------------
# Split data into features (X) and target (y)
X = df.iloc[:, :-1]
y = df['species']
# Split into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# ------------------------- PCA -------------------------
# Apply PCA to reduce to 2 components
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
# Convert PCA results to a DataFrame for visualization
pca_df = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
pca_df['species'] = y
# Plot PCA result
plt.figure(figsize=(8, 5))
sns.scatterplot(x='PC1', y='PC2', hue='species', data=pca_df, palette='viridis', s=100)
plt.title('PCA - Iris Dataset (2D)')
plt.show()
```

```python
# ------------------------ LDA ------------------------
# Apply LDA to reduce to 2 components
lda = LDA(n_components=2)
X_lda = lda.fit_transform(X, y)
# Convert LDA results to a DataFrame for visualization
lda_df = pd.DataFrame(X_lda, columns=['LD1', 'LD2'])
lda_df['species'] = y
# Plot LDA result
plt.figure(figsize=(8, 5))
sns.scatterplot(x='LD1', y='LD2', hue='species', data=lda_df, palette='coolwarm', s=100)
plt.title('LDA - Iris Dataset (2D)')
plt.show()
# ------------------------ MODEL TRAINING (ON PCA) ------------------------
# Training the Naive Bayes model on PCA-transformed data
X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=42)
model_pca = GaussianNB()
model_pca.fit(X_train_pca, y_train)
# Predict using PCA data
y_pred_pca = model_pca.predict(X_test_pca)
# Evaluate PCA model
print("\nPCA Model Accuracy:", accuracy_score(y_test, y_pred_pca))
print("\nConfusion Matrix (PCA):\n", confusion_matrix(y_test, y_pred_pca))
print("\nClassification Report (PCA):\n", classification_report(y_test, y_pred_pca))
# ------------------------ MODEL TRAINING (ON LDA) ------------------------
# Training the Naive Bayes model on LDA-transformed data
X_train_lda, X_test_lda, y_train, y_test = train_test_split(X_lda, y, test_size=0.2, random_state=42)
model_lda = GaussianNB()
model_lda.fit(X_train_lda, y_train)
# Predict using LDA data
y_pred_lda = model_lda.predict(X_test_lda)
# Evaluate LDA model
print("\nLDA Model Accuracy:", accuracy_score(y_test, y_pred_lda))
print("\nConfusion Matrix (LDA):\n", confusion_matrix(y_test, y_pred_lda))
print("\nClassification Report (LDA):\n", classification_report(y_test, y_pred_lda))
```

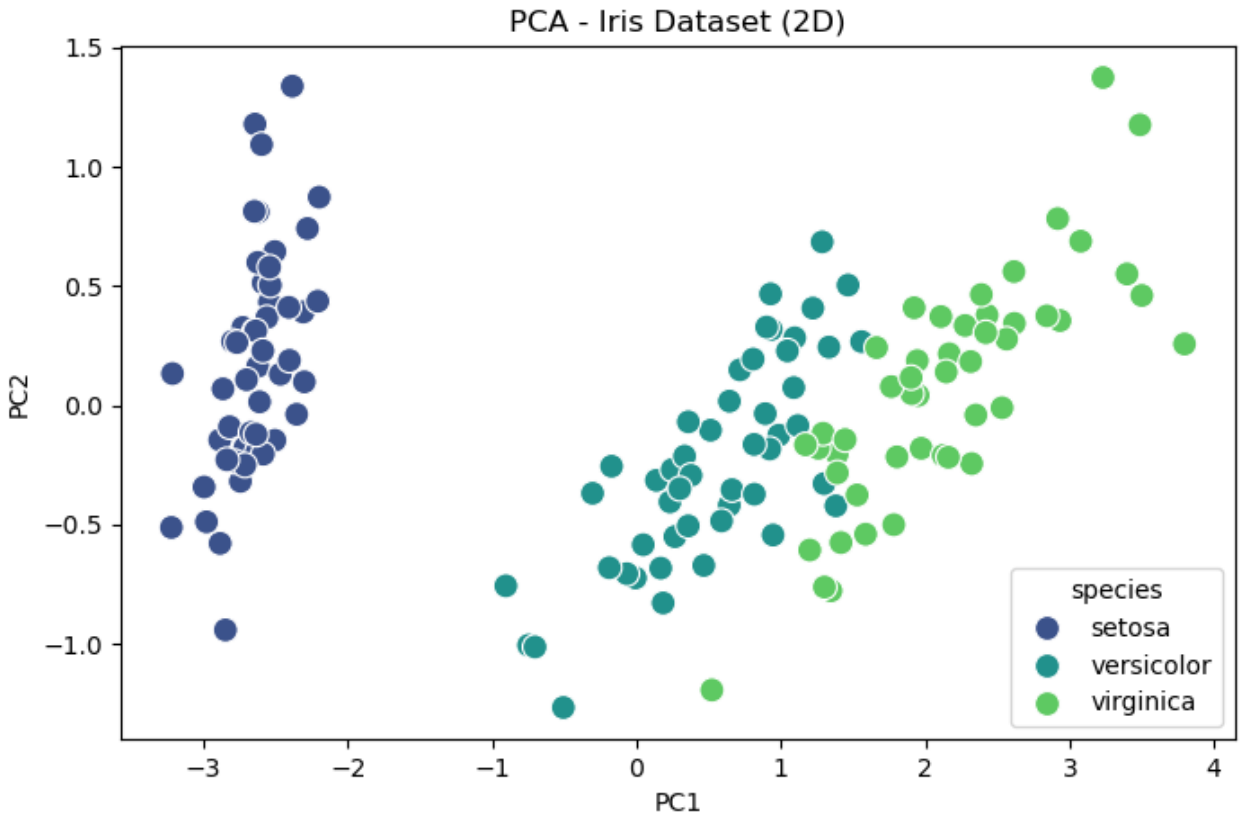
Output:

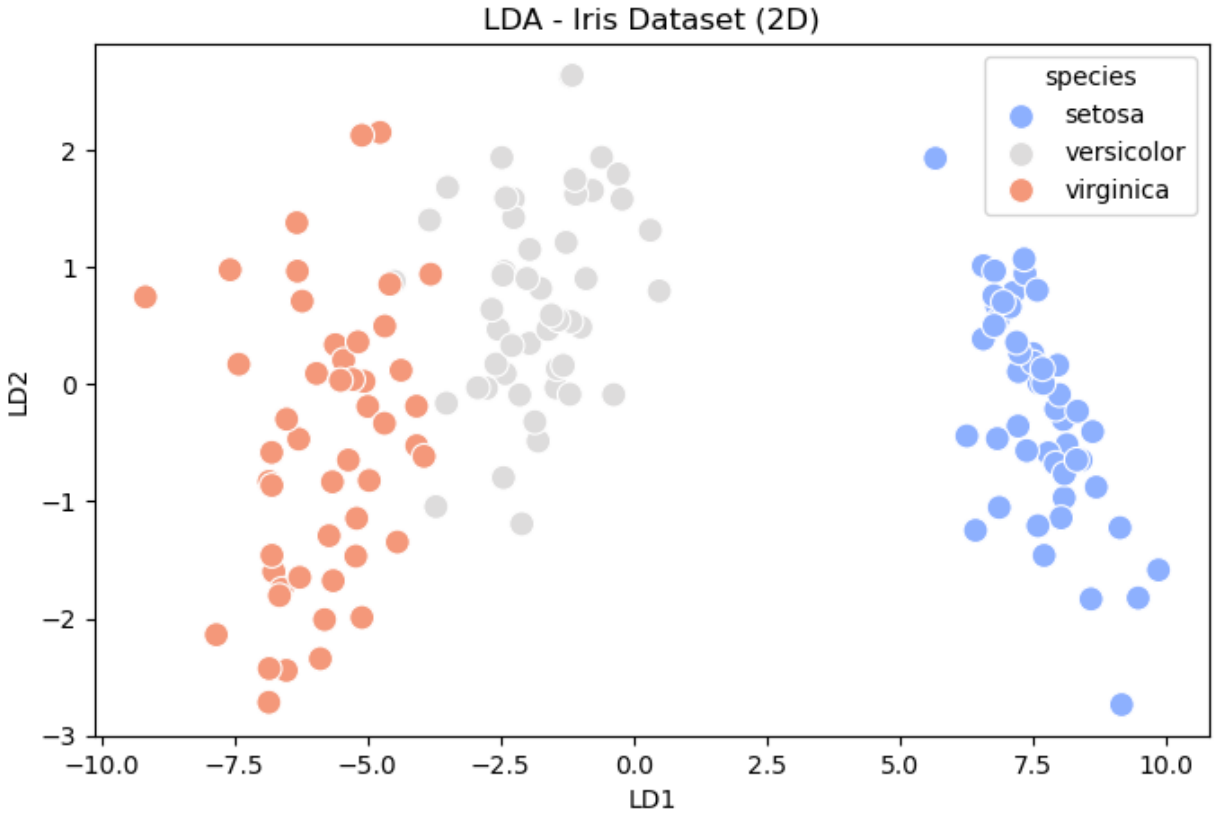
Terminal:

```
python3 -m exp7
Sample Data:
    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width
(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  setosa4               5.0               3.6                1.4
0.2  setosa
2025-03-11 23:39:41.330 python3[20146:812343] +[IMKClient subclass]: chose
IMKClient_Modern
2025-03-11 23:39:41.330 python3[20146:812343] +[IMKInputSession subclass]:
chose IMKInputSession_Modern
2025-03-11 23:39:59.409 python3[20146:812343] The class 'NSSavePanel'
overrides the method identifier.  This method is implemented by class
'NSWindow'
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
```

Graph:


PCA - Iris Dataset (2D)
PC2
PC1
1.5
1.0
0.5
0.0
−0.5
−1.0
−3
−2
−1
0
1
2
3
4
species
setosa
versicolor
virginica


LDA - Iris Dataset (2D)
LD2
LD1
2
1
0
−1
−2
−3
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
species
setosa
versicolor
virginica

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:

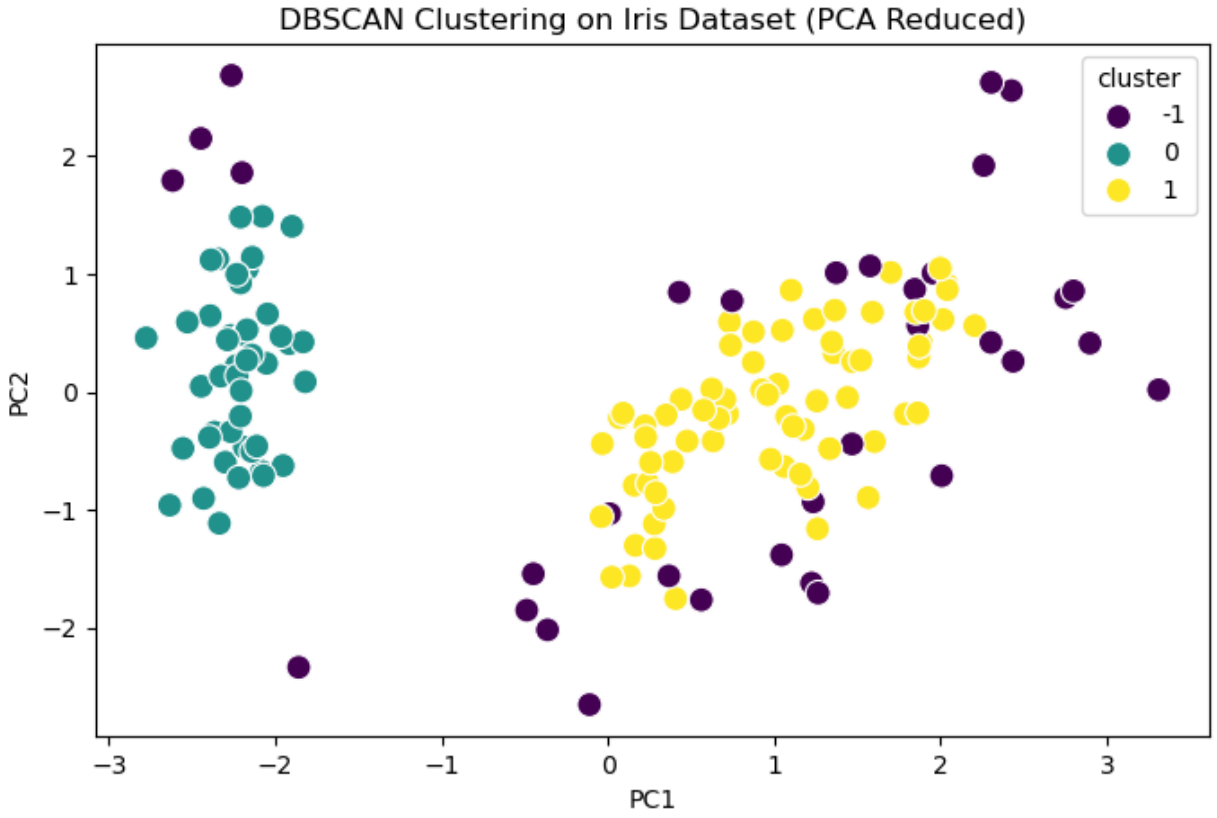
```python
# Importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
from sklearn.decomposition import PCA
# ------------------------- LOADING DATA -------------------------
# Load the Iris dataset
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
# Display sample data
print("\nSample Data:\n", df.head())
# ------------------------- DATA PREPROCESSING -------------------------
# Scale the data for better clustering results
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df)
# ------------------------- APPLYING DBSCAN -------------------------
# Apply DBSCAN clustering
dbscan = DBSCAN(eps=0.5, min_samples=5)
clusters = dbscan.fit_predict(scaled_data)
# Add cluster labels to the dataset
df['cluster'] = clusters
# ------------------------- VISUALIZATION (PCA) -------------------------
# Reduce dimensions using PCA for visualization
pca = PCA(n_components=2)
pca_result = pca.fit_transform(scaled_data)
df['PC1'] = pca_result[:, 0]
df['PC2'] = pca_result[:, 1]
# Plot clusters using PCA components
plt.figure(figsize=(8, 5))
sns.scatterplot(x='PC1', y='PC2', hue='cluster', data=df, palette='viridis', s=100)
plt.title('DBSCAN Clustering on Iris Dataset (PCA Reduced)')
plt.show()
# ------------------------- EVALUATION -------------------------
# Number of clusters and noise points
n_clusters = len(set(clusters)) - (1 if -1 in clusters else 0)
n_noise = list(clusters).count(-1)
print(f"\nNumber of clusters: {n_clusters}")
print(f"Number of noise points: {n_noise}")
# Display cluster assignments
print("\nCluster Assignments:")
print(df[['PC1', 'PC2', 'cluster']].head(10))
```

Output:

Terminal:

```
python3 -m exp8
Sample Data:
    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width
(cm)
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
4                5.0               3.6                1.4
0.2
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
```

Graph:


DBSCAN Clustering on Iris Dataset (PCA Reduced)
cluster
-1
0
1
PC1
PC2
−3
−2
−1
0
1
2
3
2
1
0
−1
−2

Experiment 9

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

Software Used: Visual Studio Code

Code:

```python
# Importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn_extra.cluster import KMedoids
from sklearn.decomposition import PCA
# ------------------------- LOADING DATA -------------------------
# Load the Iris dataset
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
# Display sample data
print("\nSample Data:\n", df.head())
# ------------------------- DATA PREPROCESSING -------------------------
# Scale the data for better clustering results
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df)
# ------------------------- APPLYING K-MEDOIDS -------------------------
# Apply K-Medoids clustering with 3 clusters (since we have 3 iris species)
kmedoids = KMedoids(n_clusters=3, random_state=42)
clusters = kmedoids.fit_predict(scaled_data)
# Add cluster labels to the dataset
df['cluster'] = clusters
```

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

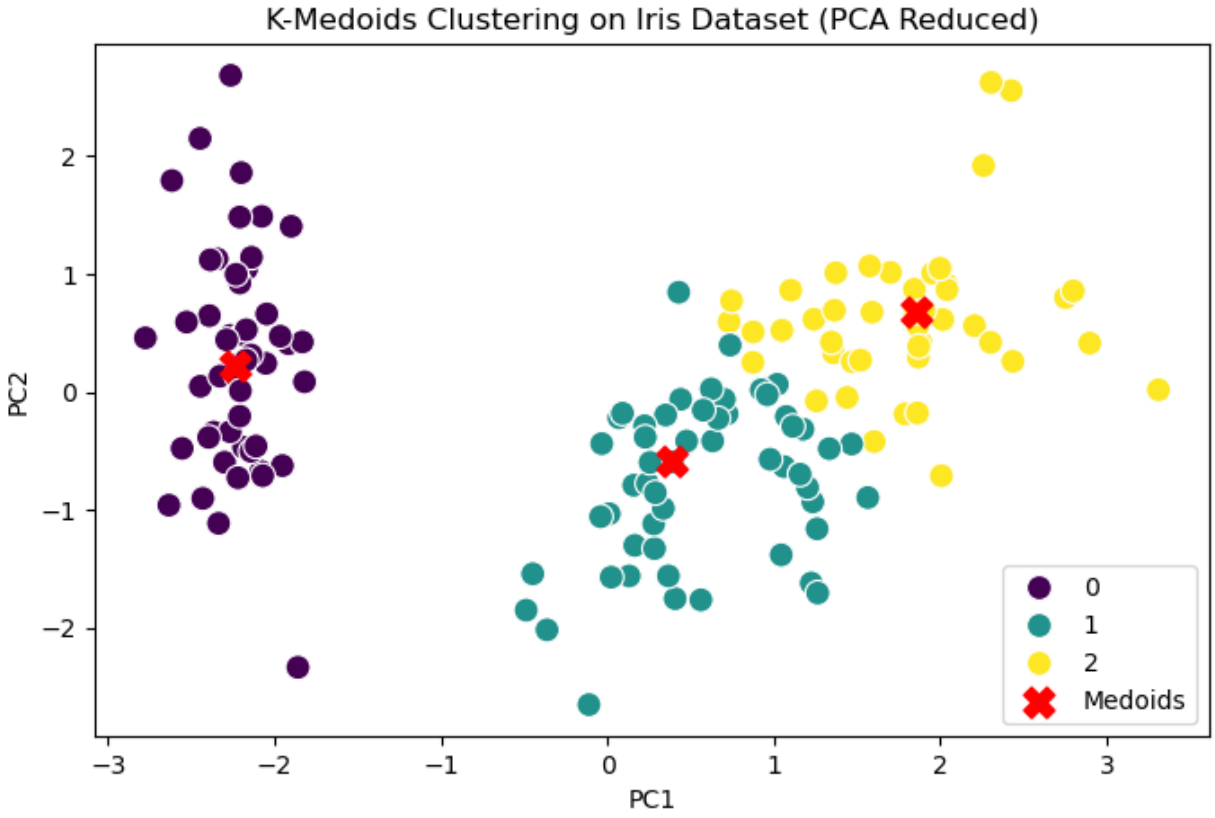
Output:

Terminal:

```
python3 -m exp9
Sample Data:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width
(cm)
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
4                5.0               3.6                1.4
0.2
2025-03-12 00:25:19.035 python3[21318:850424] +[IMKClient subclass]: chose
IMKClient_Modern
2025-03-12 00:25:19.035 python3[21318:850424] +[IMKInputSession subclass]:
chose IMKInputSession_Modern
2025-03-12 00:25:56.752 python3[21318:850424] The class 'NSSavePanel'
overrides the method identifier.  This method is implemented by class
'NSWindow'
Number of clusters: 3
Cluster Sizes:
1    56
0    50
2    44
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:


K-Medoids Clustering on Iris Dataset (PCA Reduced)
PC1
PC2
0
1
2
Medoids

Experiment 10

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

Software Used: Visual Studio Code

Code:

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

```python
# ------------------------ VISUALIZATION (PCA) ------------------------
# Reduce dimensions using PCA for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
# Create a DataFrame for plotting
import pandas as pd
df = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
df['cluster'] = clusters
# Plot clusters using PCA components
plt.figure(figsize=(8, 5))
sns.scatterplot(x='PC1', y='PC2', hue='cluster', data=df, palette='tab10', s=100)
plt.title('K-Means Clustering on Handwritten Digits (PCA Reduced)')
plt.legend(title='Cluster')
plt.show()
# ------------------------ EVALUATION ------------------------
# Count the number of samples in each cluster
cluster_sizes = pd.Series(clusters).value_counts()
print("\nNumber of samples in each cluster:")
print(cluster_sizes)
# Display cluster assignments for the first 10 samples
print("\nCluster Assignments:")
print(df[['PC1', 'PC2', 'cluster']].head(10))
# ------------------------ DISPLAY CLUSTER CENTERS ------------------------
# Display cluster centers as images
centers = kmeans.cluster_centers_.reshape(10, 8, 8)
plt.figure(figsize=(8, 4))
for i, center in enumerate(centers):
    plt.subplot(2, 5, i + 1)
    plt.imshow(center, cmap='gray')
    plt.axis('off')
plt.suptitle('Cluster Centers as Handwritten Digits')
plt.show()
```

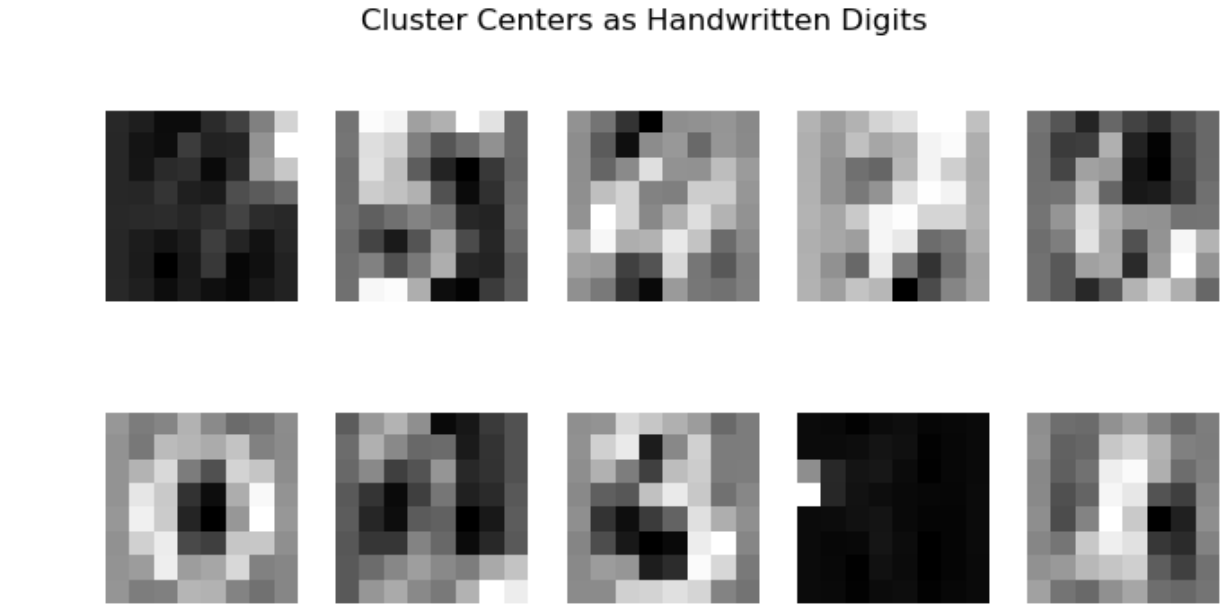
Output:

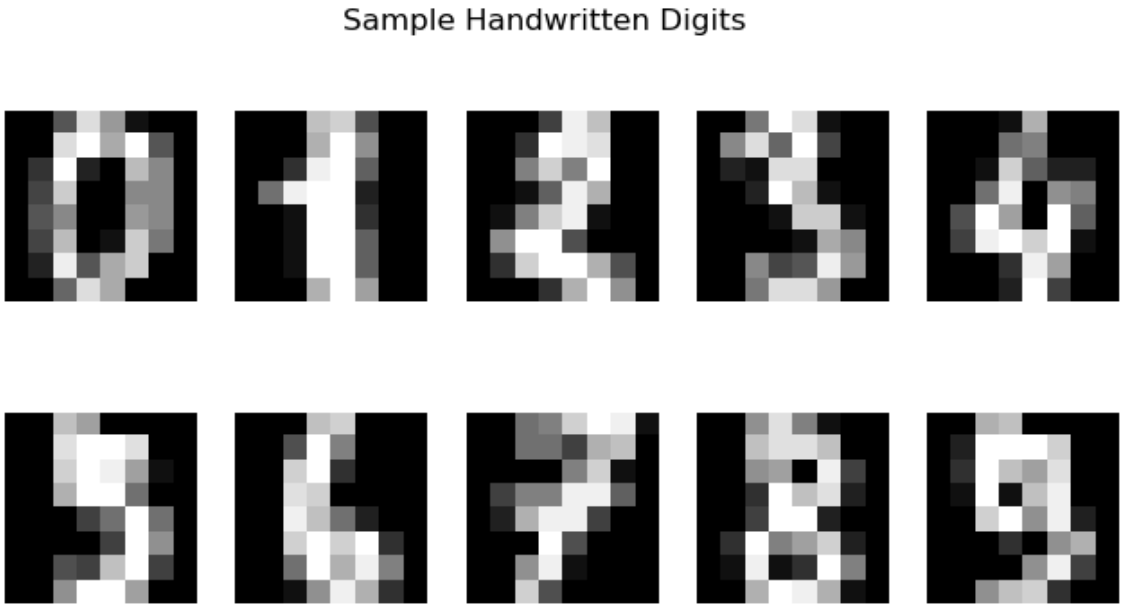
Terminal:

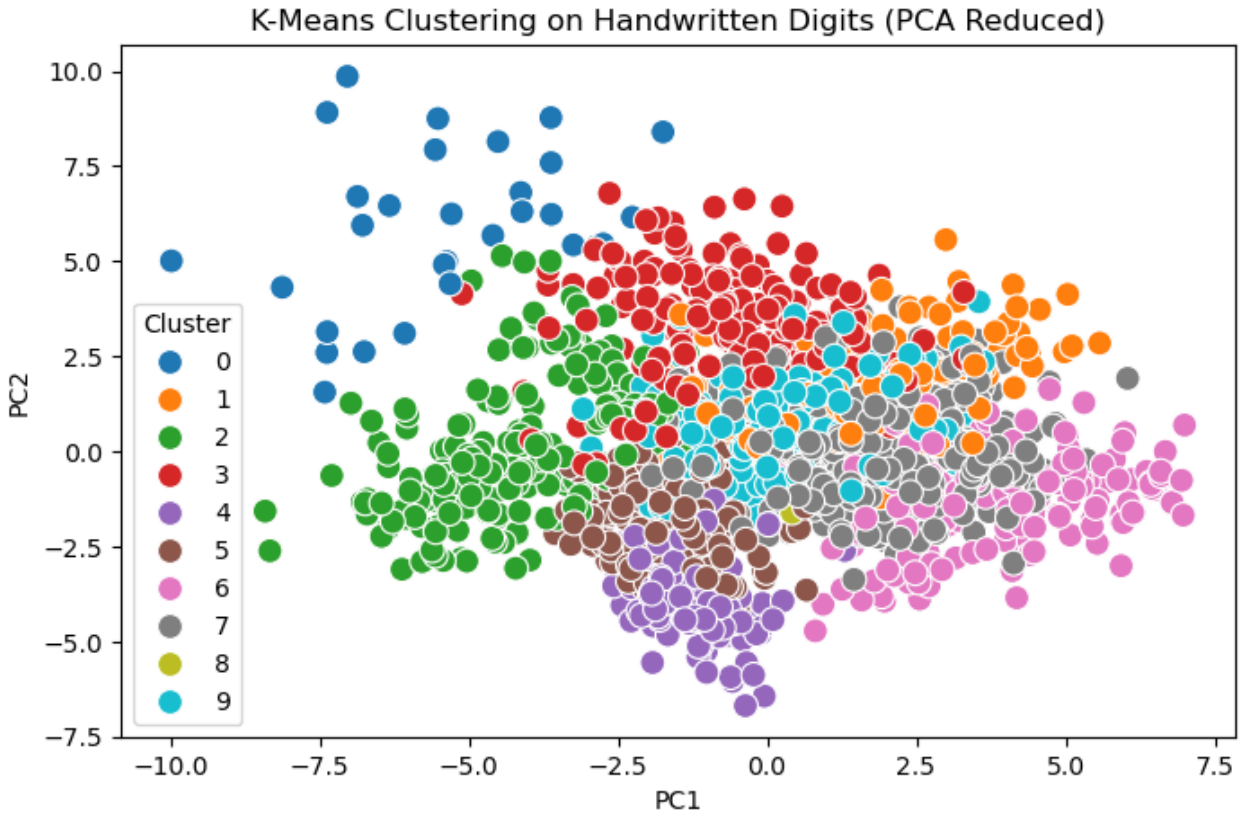
```
python3 -m exp10
2025-03-12 00:42:37.363 python3[21572:863295] +[IMKClient subclass]: chose IMKClient_Modern
2025-03-12 00:42:37.363 python3[21572:863295] +[IMKInputSession subclass]: chose IMKInputSession_Modern
```

```
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:




Sample Handwritten Digits


K-Means Clustering on Handwritten Digits (PCA Reduced)
PC1
PC2
Cluster
0
1
2
3
4
5
6
7
8
9
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5

Machine Learning (ECE-350P)

Lab Record

S.No	Experiment Aim	Date	Sign
1	Introduction to: (a) Jupyter Notebook IDE (b) Spyder IDE (c) Numpy and Pandas (d) Matplotlib and Seaborn (e) Scikit Learn		
2	Write a program to demonstrate Simple Linear Regression		
3	Write a program to demonstrate Logistic Regression		
4	Write a program to demonstrate Decision Tree-ID3 Algorithm		
5	Write a program to demonstrate K-Nearest Neighbors(KNN) for flower classification		
6	Write a program to demonstrate Naive Bayes Classifier		
7	Write a program to demonstrate PCA and LDA on the IRIS dataset		
8	Write a program to demonstrate DBSCAN(Density Based Spatial Clustering of Applications With Noise) clustering algorithm		
9	Write a program to demonstrate K-Medoids clustering algorithm		
10	Write a program to demonstrate K-Mean clustering on handwritten Digits dataset		
11	Write a program to demonstrate Support Vector Machine(SVM) for binary classification using the Breast Cancer Dataset		
12	Write a program to determine the optimal number of clusters using the Elbow method on the Wine Dataset		

Experiment 11

Aim: Write a program to demonstrate Support Vector Machine (SVM) for binary classification using the Breast Cancer dataset.

Software Used: Visual Studio Code

Code:

```python
# ------------------------- IMPORTING NECESSARY LIBRARIES -------------------------
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.decomposition import PCA
from sklearn.metrics import classification_report, confusion_matrix

# ------------------------- LOADING DATA -------------------------
# Load the Breast Cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target
feature_names = data.feature_names
target_names = data.target_names

# Display dataset shape and feature names
print("Dataset shape:", X.shape)
print("Target classes:", target_names)

# ------------------------- DATA PREPROCESSING -------------------------
# Scale the data for better performance
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```python
# Split into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)

# ------------------------- TRAINING THE SVM MODEL -------------------------
# Create and train the Support Vector Classifier
svm_model = SVC(kernel='rbf', random_state=42)
svm_model.fit(x_train, y_train)

# Predict on test set
predictions = svm_model.predict(x_test)

# ------------------------- EVALUATION -------------------------
# Print classification results
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, predictions))

print("\nClassification Report:")
print(classification_report(y_test, predictions, target_names=target_names))

# ------------------------- VISUALIZATION USING PCA -------------------------
# Reduce features to 2 principal components for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Create a DataFrame for plotting
pca_df = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
pca_df['Target'] = y

# Plot PCA-reduced data
plt.figure(figsize=(8, 5))
sns.scatterplot(data=pca_df, x='PC1', y='PC2', hue='Target', palette='Set1', s=100)
plt.title('Breast Cancer Dataset (PCA Reduced)')
plt.legend(title='Class', labels=target_names)
plt.show()
```

Output:

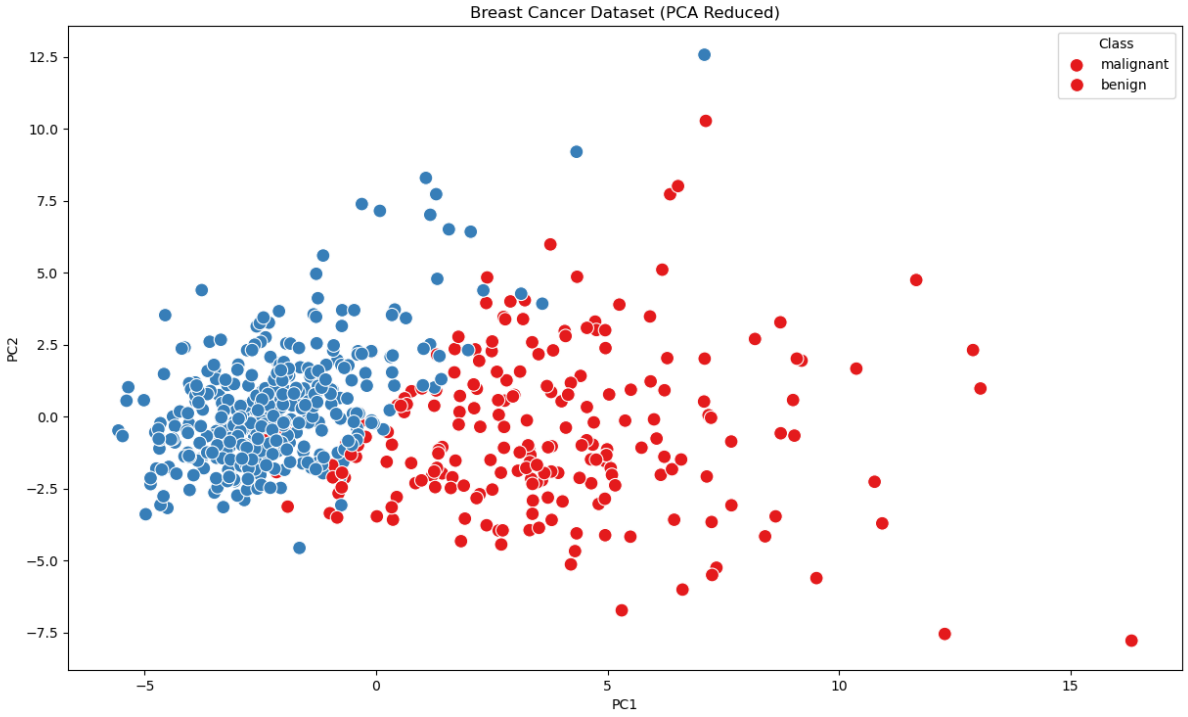
Terminal:

```
python3 -m exp11
Dataset shape: (569, 30)
Target classes: ['malignant' 'benign']
Confusion Matrix:
[[ 61   2]
 [  3 105]]
```

Classification Report:

	precision	recall	f1-score	support
malignant	0.95	0.97	0.96	63
benign	0.98	0.97	0.98	108
accuracy			0.97	171
macro avg	0.97	0.97	0.97	171
weighted avg	0.97	0.97	0.97	171

2025-04-24 01:17:28.441 python3[8113:237083] The class 'NSSavePanel' overrides



the method identifier. This method is implemented by class 'NSWindow'

Graph:

Experiment 12

Aim: Write a program to determine the optimal number of clusters using the Elbow Method on the Wine dataset.

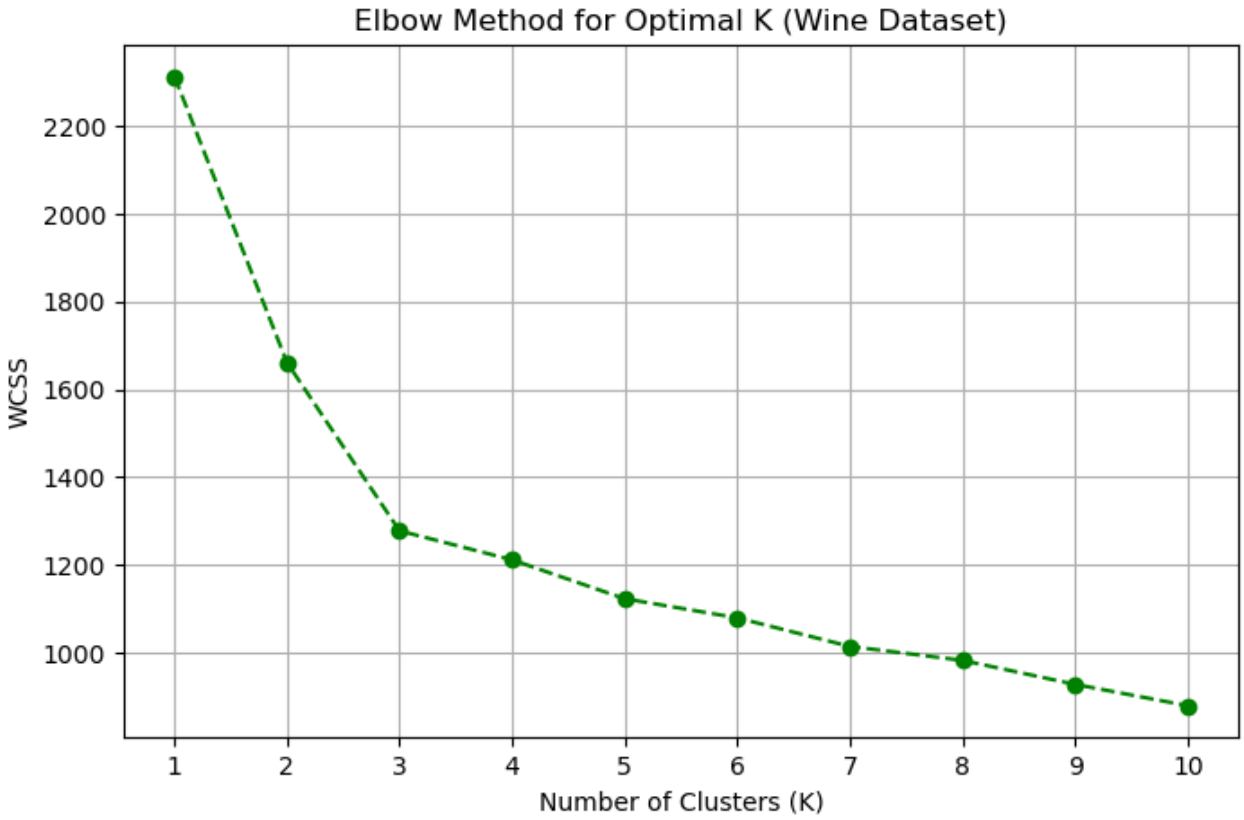
Software Used: Visual Studio Code

Code:

```python
# ------------------------- IMPORTING NECESSARY LIBRARIES -------------------------
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_wine
from sklearn.preprocessing import StandardScaler

# ------------------------- LOADING AND PREPROCESSING DATA -------------------------
# Load the wine dataset (also available on Kaggle)
wine_data = load_wine()
X = wine_data.data
feature_names = wine_data.feature_names

```

```python
# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# ------------------------- ELBOW METHOD TO FIND OPTIMAL K -------------------------
wcss = []

# Test for cluster counts from 1 to 10
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, init='k-means++', random_state=42)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)

# ------------------------- PLOTTING THE ELBOW GRAPH -------------------------
plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), wcss, marker='o', linestyle='--', color='green')
plt.title('Elbow Method for Optimal K (Wine Dataset)')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('WCSS')
plt.xticks(range(1, 11))
plt.grid(True)
plt.show()
```

Output:


Elbow Method for Optimal K (Wine Dataset)
WCSS
2200
2000
1800
1600
1400
1200
1000
1
2
3
4
5
6
7
8
9
10
Number of Clusters (K)