INDEX

S. No.	Experiment Name	Date	Grade	Signature
1.	Import necessary libraries (scikit-learn, numpy, pandas, matplotlib).Load a dataset suitable for supervised learning.Split the dataset into training and testing sets.	Jan 2025		
2.	Implement a basic gradient descent algorithm for a simple function.Visualize the convergence of the algorithm over iterations.	Feb 2025		
3.	Implement a decision tree classifier using scikit-learn. Tune the hyperparameters of the decision tree.Visualize the constructed decision tree.	Feb 2025		
4.	Implement a Support Vector Machine using scikit-learn. Experiment with different kernel functions.Visualize the decision boundaries.	Feb 2025		
5.	Experiment with different regularization parameters. Implement Logistic Regression for classification tasks.	Feb 2025		
6.	Implement a basic ensemble method (e.g., Random Forest) using scikit-learn. Implement Adaptive Boosting (AdaBoost) and Extreme Gradient Boosting (XGBoost).Compare the performance of individual models with the ensemble.	March 2025		
7.	Implement k-Means clustering and Hierarchical clustering. Implement a density-based clustering algorithm (e.g., DBSCAN).Visualize the clustering results.	March 2025		
8.	Implement k-N). Visualize KNN for a simple classification task and compare its performance with a decision tree model. Analyze the effect of varying K on accuracy.	March 2025		
9.	Train a linear regression model to predict housing prices based on features like area, number of bedrooms, etc. Explore the impact of outliers and feature scaling on model performance.	April 2025		
10.	Implement Principal Component Analysis (PCA) for dimensionality reduction.Visualize the effect of dimensionality reduction using scatter plots.	April 2025		
	Content Beyond Syllabus -			
11.	Implement Features Selection Techniques			

Lab Experiment 1: Split the dataset into training and testing sets (Supervised Learning with scikit-learn)

Aim - To implement supervised learning by loading a dataset, splitting it into training and testing sets, and visualizing its distribution using Python and scikit-learn.

Objective –
Understand supervised learning and its application to real-world datasets.
Load a dataset (e.g., Iris dataset) using scikit-learn.
Preprocess the dataset (handle features and target variables).
Split the dataset into training and testing sets using train_test_split().
Visualize the dataset distribution using matplotlib.

Required Libraries –
numpy , pandas , matplotlib , sklearn.model_selection , sklearn.datasets

Data Description (Iris Dataset) –
The **Iris dataset** is a classic dataset in machine learning used for classification tasks. It contains **150 samples** from **3 different classes** of the iris flower:

- **Setosa (Class 0)**
- **Versicolor (Class 1)**
- **Virginica (Class 2)**

Each sample has **4 features**:
Sepal length (cm)
Sepal width (cm)
Petal length (cm)
Petal width (cm)
The target variable (target) represents the **class of the iris flower** (0, 1, or 2).

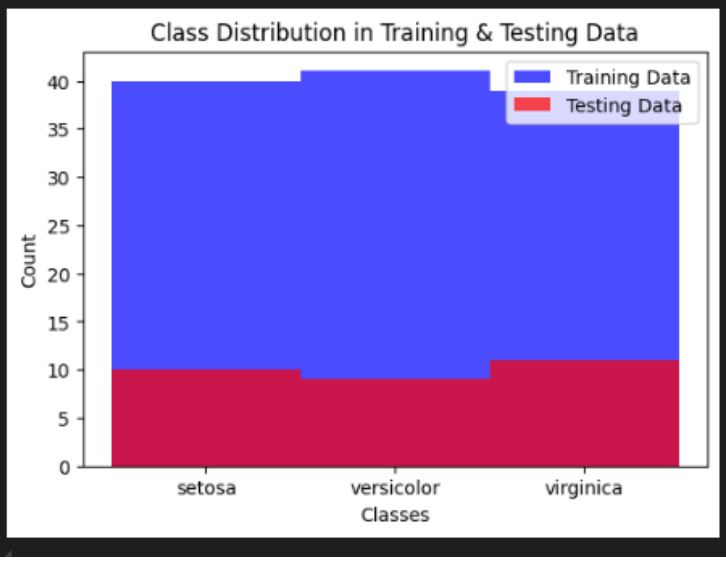
Code –

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

# Display first 5 rows of the dataset
print("First 5 rows of dataset:\n", df.head())
```

```
# Split data into features (X) and target (y)
X = df.drop(columns=['target'])  # Features
y = df['target']  # Target variable (classification labels)

# Split dataset into Training (80%) and Testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Print dataset sizes
print(f"Training set size: {X_train.shape}")
print(f"Testing set size: {X_test.shape}")

# Plot the dataset distribution
plt.figure(figsize=(6, 4))
plt.hist(y_train, bins=np.arange(4)-0.5, alpha=0.7, label="Training Data", color='b')
plt.hist(y_test, bins=np.arange(4)-0.5, alpha=0.7, label="Testing Data", color='r')
plt.xticks([0, 1, 2], iris.target_names)
plt.xlabel("Classes")
plt.ylabel("Count")
plt.legend()
plt.title("Class Distribution in Training & Testing Data")
plt.show()
```

Output –

```
First 5 rows of dataset:
    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

   target
0       0
1       0
2       0
3       0
4       0
Training set size: (120, 4)
Testing set size: (30, 4)
```



Lab Experiment 2: Linear Regression using Gradient Descent, Normal Equation, and Scikit-Learn

Aim - To implement Linear Regression using:

1. Gradient Descent
2. Normal Equation
3. Scikit-Learn's Linear Regression
   And compare the performance of all three methods.

Objective –

- Load the Advertising dataset and extract features.
- Implement Gradient Descent to compute linear regression parameters.
- Use the Normal Equation to find the best-fit parameters.
- Train Linear Regression using Scikit-Learn and compare results.
- Visualize the cost function convergence, actual vs predicted values, and residuals.

Libraries – numpy, Pandas, matplotlib, sklearn.linear_model, sklearn.metrics

Dataset Description (Advertising Dataset) -

The Advertising dataset contains data about how different media channels impact product sales.

Features (Independent Variables):

- TV Advertising Budget ($)
- Radio Advertising Budget ($)
- Newspaper Advertising Budget ($)

Target Variable (Dependent Variable):

- Sales (Units sold)

Code –

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

df = pd.read_csv("advertising.csv")  # Ensure the dataset file is present in the same directory
print(df)

Extract features (TV, Radio, Newspaper) and target variable (Sales)
X = df[['TV', 'Radio', 'Newspaper']].values  # Feature matrix
y = df['Sales'].values.reshape(-1, 1)  # Convert y to a column vector
X = np.c_[np.ones(X.shape[0]), X]  # Adds a column of ones to the first column

# 4Normalize features (excluding the bias term)
X[:, 1:] = (X[:, 1:] - X[:, 1:].mean(axis=0)) / X[:, 1:].std(axis=0)  # Feature Scaling

### ========================= GRADIENT DESCENT ========================= ###
def gradient_descent(X, y, learning_rate=0.01, iterations=1000):
    """
    Performs Gradient Descent to optimize theta values.

    Parameters:
    X (numpy array): Feature matrix (m x n)
    y (numpy array): Target variable (m x 1)
    learning_rate (float): Step size for gradient updates
    iterations (int): Number of iterations

    Returns:
    theta (numpy array): Optimal parameters (n x 1)
    cost_history (list): Cost function values over iterations
    """
    m, n = X.shape  # m = number of samples, n = number of features
    theta = np.zeros((n, 1))  # Initialize parameters (theta) as zeros
    cost_history = []  # Store cost values to visualize convergence

    for _ in range(iterations):
```

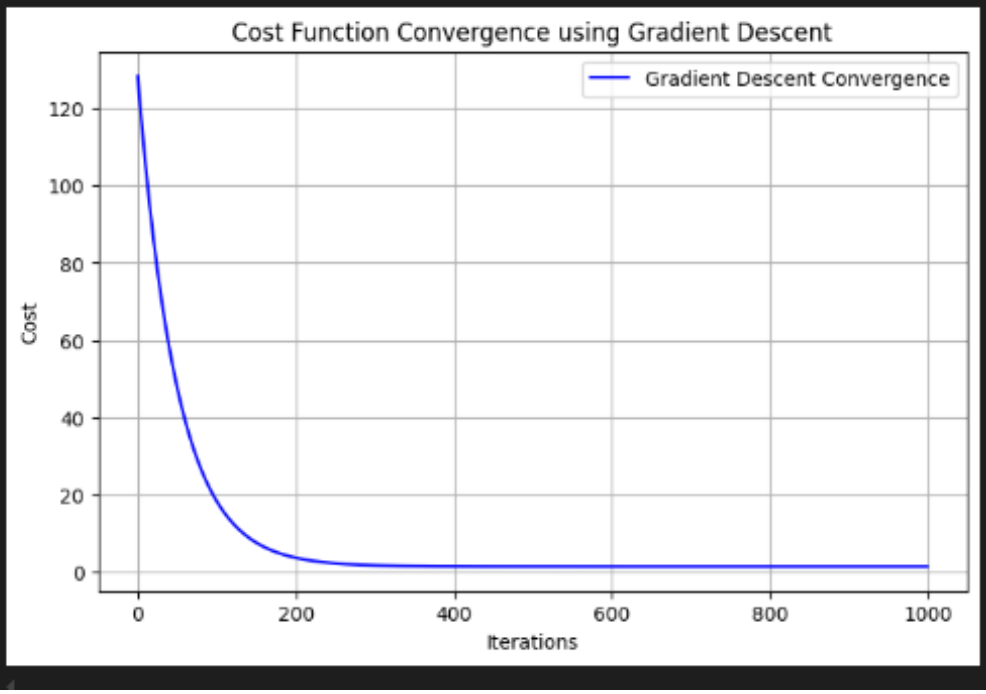
```
        predictions = X.dot(theta)  # Compute predicted values
        errors = predictions - y  # Compute errors
        cost = (1 / (2 * m)) * np.sum(errors**2)  # Compute cost (Mean Squared Error)
        cost_history.append(cost)  # Store cost

        # Compute gradient (partial derivatives of cost function)
        gradient = (1 / m) * X.T.dot(errors)

        # Update theta using gradient descent formula
        theta -= learning_rate * gradient

    return theta, cost_history
# 5️⃣ Apply Gradient Descent
theta_gd, cost_history = gradient_descent(X, y)

print("\nTheta values from Gradient Descent:\n", theta_gd)

### ========================= NORMAL EQUATION ========================= ###
def normal_equation(X, y):
    """
    Computes theta values using the Normal Equation formula:
    θ = (X^T X)^(-1) X^T y

    Parameters:
    X (numpy array): Feature matrix (m x n)
    y (numpy array): Target variable (m x 1)

    Returns:
    theta (numpy array): Optimal parameters (n x 1)
    """
    return np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)

# 7️⃣ Apply Normal Equation
theta_ne = normal_equation(X, y)

print("\nTheta values from Normal Equation:\n", theta_ne)

### ========================= SCIKIT-LEARN LINEAR REGRESSION ========================= ###
# 8️⃣ Train Linear Regression Model using Scikit-Learn
model = LinearRegression()
model.fit(df[['TV', 'Radio', 'Newspaper']], df['Sales'])

# Extract theta values (Intercept + Coefficients)
theta_sklearn = np.vstack(([model.intercept_], model.coef_.reshape(-1, 1)))

print("\nTheta values from Scikit-Learn:\n", theta_sklearn)

### ========================= COMPARE RESULTS ========================= ###
print("\nComparison of Results:")
for i in range(len(theta_gd)):
    print(f"Theta_{i}: Gradient Descent = {theta_gd[i][0]:.4f}, Normal Equation = {theta_ne[i][0]:.4f},
```

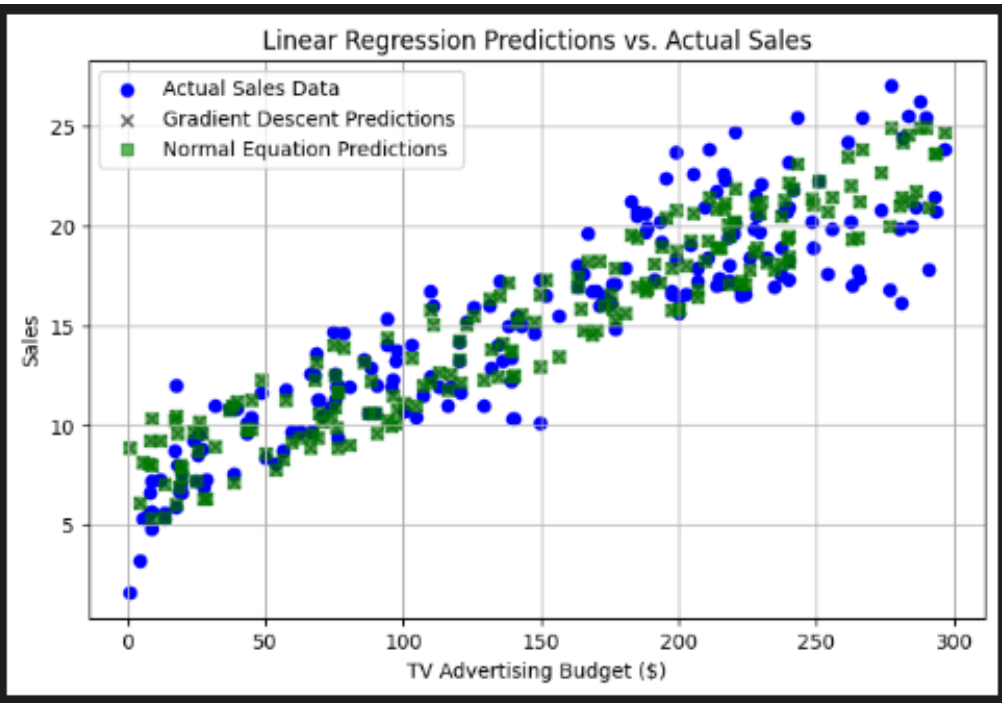
```
Scikit-Learn = {theta_sklearn[i][0]:.4f}")



# Predict values using all three methods
y_pred_gd = X.dot(theta_gd)   # Predictions from Gradient Descent
y_pred_ne = X.dot(theta_ne)   # Predictions from Normal Equation
y_pred_sklearn = model.predict(df[['TV', 'Radio', 'Newspaper']])  # Predictions from Scikit-Learn

#Compute Residual for gradient descent and Normal Equation
residuals_gd = y- y_pred_gd
residuals_ne = y - y_pred_ne

# 1️⃣ Plot Cost Function Convergence
plt.figure(figsize=(8, 5))
plt.plot(range(len(cost_history)), cost_history, label="Gradient Descent Convergence", color="blue")
plt.xlabel("Iterations")
plt.ylabel("Cost")
plt.title("Cost Function Convergence using Gradient Descent")
plt.legend()
plt.grid()
plt.show()

# 2️⃣ Actual vs Predicted Sales Plot
plt.figure(figsize=(8, 5))
plt.scatter(df['TV'], y, color='blue', label="Actual Sales Data")  # Actual data points
plt.scatter(df['TV'], y_pred_gd, color='black', label="Gradient Descent Predictions", alpha=0.6,
marker='x')  # GD Predictions
plt.scatter(df['TV'], y_pred_ne, color='green', label="Normal Equation Predictions", alpha=0.6,
marker='s')  # NE Predictions

# Labels and title - Actual vs Predicted Sales Plot
plt.xlabel("TV Advertising Budget ($)")
plt.ylabel("Sales")
plt.title("Linear Regression Predictions vs. Actual Sales")
plt.legend()
plt.grid()
plt.show()

#Create the residual plot
plt.figure(figsize=(8,5))
plt.scatter(y_pred_gd, residuals_gd, color="yellow", alpha = 0.6, label="Gradient Descent Residuals")
plt.scatter(y_pred_gd, residuals_gd, color="green", alpha = 0.6, label="Normal Equation Residuals")

# Add a horizontal line at y=0
plt.axhline(y=0, color="black", linestyle="--")

# Labels and title - Residual plot "Gradient Descent vs Normal Equation"
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Residual Plot: Gradient Descent vs. Normal Equation")
plt.legend()
plt.grid()
plt.show()
```

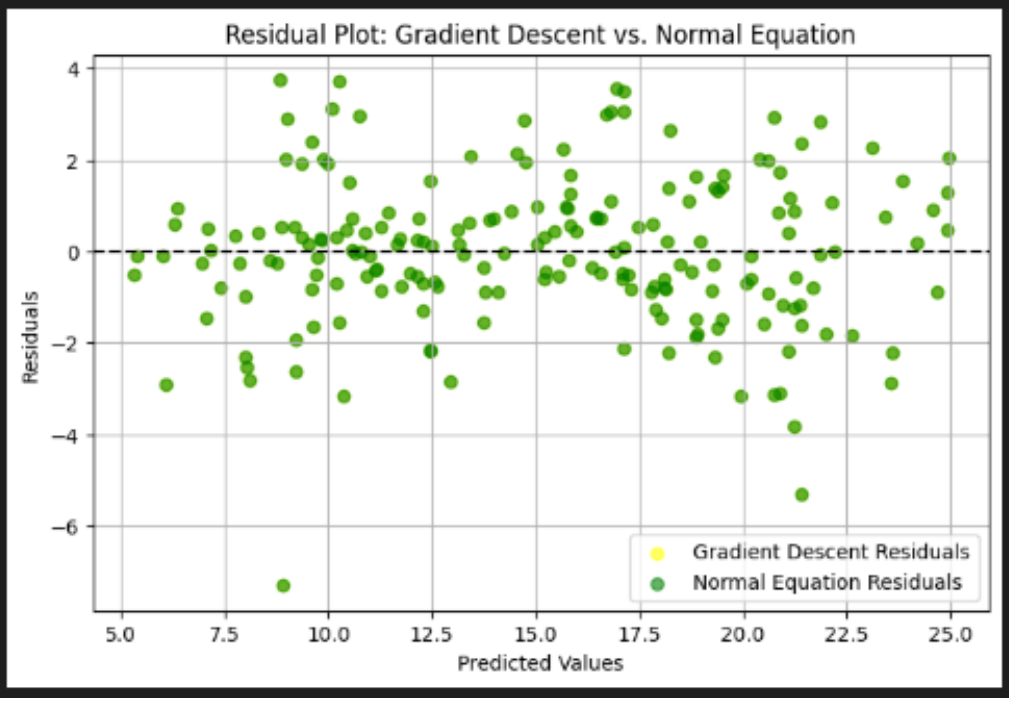
```
# Compute Predictions using Gradient Descent
y_pred_gd = X.dot(theta_gd)

# Compute Predictions using Normal Equation
y_pred_ne = X.dot(theta_ne)

# Compute MSE for both methods
mse_gd = mean_squared_error(y, y_pred_gd)
mse_ne = mean_squared_error(y, y_pred_ne)

# Compute R-squared for both methods
r2_gd = r2_score(y, y_pred_gd)
r2_ne = r2_score(y, y_pred_ne)

# Print Comparison of MSE and R² Score
print("\nPerformance Comparison:")
print(f"{'Method':<20} {'MSE':<10} {'R² Score':<10}")
print("-" * 40)
print(f"{'Gradient Descent':<20} {mse_gd:.4f} {r2_gd:.4f}")
print(f"{'Normal Equation':<20} {mse_ne:.4f} {r2_ne:.4f}")
```

Output –

```
Theta values from Gradient Descent:
 [[1.51298468e+01]
 [4.66247710e+00]
 [1.58345404e+00]
 [8.55119671e-03]]

Theta values from Normal Equation:
 [[1.51305000e+01]
 [4.66270025e+00]
 [1.58465027e+00]
 [7.29186830e-03]]

Theta values from Scikit-Learn:
 [[4.62512408e+00]
 [5.44457803e-02]
 [1.07001228e-01]
 [3.35657922e-04]]

Comparison of Results:
Theta_0: Gradient Descent = 15.1298, Normal Equation = 15.1305, Scikit-Learn = 4.6251
Theta_1: Gradient Descent = 4.6625, Normal Equation = 4.6627, Scikit-Learn = 0.0544
Theta_2: Gradient Descent = 1.5835, Normal Equation = 1.5847, Scikit-Learn = 0.1070
Theta_3: Gradient Descent = 0.0086, Normal Equation = 0.0073, Scikit-Learn = 0.0003
```




Linear Regression Predictions vs. Actual Sales
Actual Sales Data
Gradient Descent Predictions
Normal Equation Predictions
Sales
TV Advertising Budget ($)
0
50
100
150
200
250
300
5
10
15
20
25


Residual Plot: Gradient Descent vs. Normal Equation
Residuals
Predicted Values
Gradient Descent Residuals
Normal Equation Residuals
4
2
0
−2
−4
−6
5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0

Lab Experiment 3: Decision Tree Classifier on Drug200 Dataset

Aim
To implement a Decision Tree Classifier using scikit-learn, tune hyperparameters, and visualize the constructed decision tree on the Drug200 dataset.

Objectives
1. To understand the working of a **Decision Tree Classifier**.
2. To preprocess categorical and numerical data using **Label Encoding**.
3. To split the dataset into **training and testing sets**.
4. To train the **Decision Tree Classifier** and evaluate its performance.
5. To **tune hyperparameters** for better performance.
6. To **visualize the decision tree** for interpretation.

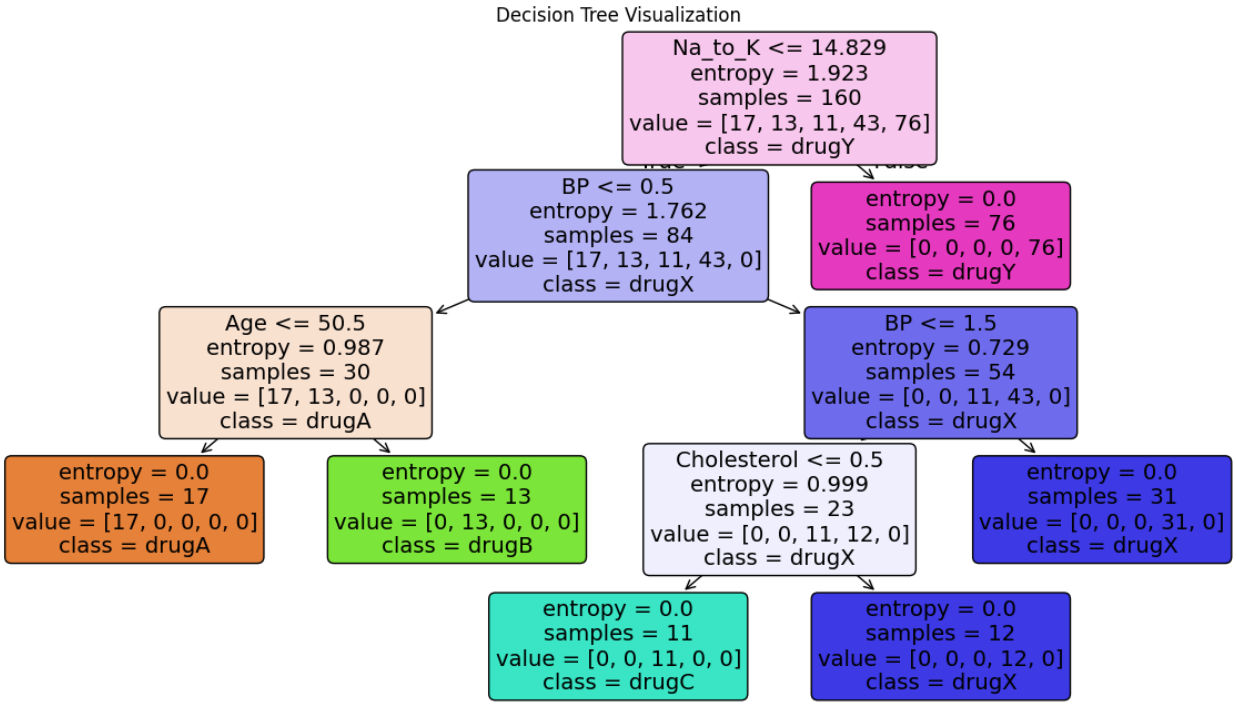
Dataset Description (Drug200.csv)
The dataset contains **200 samples** with the following features:
- **Age** (Numerical)
- **Sex** (Categorical: Male/Female)
- **BP (Blood Pressure)** (Categorical: HIGH, LOW, NORMAL)
- **Cholesterol** (Categorical: HIGH, NORMAL)
- **Na_to_K (Sodium-to-Potassium Ratio)** (Numerical)
- **Drug** (Target variable - 5 different drug types)

Code -
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report

df = pd.read_csv("drug200.csv")
print("Dataset Preview:")
print(df.head())

label_encoders = {}
for column in ['Sex', 'BP', 'Cholesterol', 'Drug']:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le

# Define features and target variable
X = df.drop(columns=['Drug'])  # Independent variables
y = df['Drug']  # Target variable
```

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

# Train a Decision Tree Classifier
clf = DecisionTreeClassifier(criterion='entropy', max_depth=4, random_state=42)
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
print("Classification Report:\n", classification_report(y_test, y_pred))

# Visualizing the Decision Tree
plt.figure(figsize=(15, 8))
plot_tree(clf, feature_names=X.columns, class_names=label_encoders['Drug'].classes_,
filled=True, rounded=True)
plt.title("Decision Tree Visualization")
plt.show()
```

Output -

```
Dataset Preview:
   Age Sex      BP Cholesterol  Na_to_K   Drug
0   23   F    HIGH        HIGH   25.355  drugY
1   47   M     LOW        HIGH   13.093  drugC
2   47   M     LOW        HIGH   10.114  drugC
3   28   F  NORMAL        HIGH    7.798  drugX
4   61   F     LOW        HIGH   18.043  drugY
Accuracy: 1.0000
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         3
           2       1.00      1.00      1.00         5
           3       1.00      1.00      1.00        11
           4       1.00      1.00      1.00        15

    accuracy                           1.00        40
   macro avg       1.00      1.00      1.00        40
weighted avg       1.00      1.00      1.00        40
```


Decision Tree Visualization
Na_to_K <= 14.829
entropy = 1.923
samples = 160
value = [17, 13, 11, 43, 76]
class = drugY
BP <= 0.5
entropy = 1.762
samples = 84
value = [17, 13, 11, 43, 0]
class = drugX
entropy = 0.0
samples = 76
value = [0, 0, 0, 0, 76]
class = drugY
Age <= 50.5
entropy = 0.987
samples = 30
value = [17, 13, 0, 0, 0]
class = drugA
BP <= 1.5
entropy = 0.729
samples = 54
value = [0, 0, 11, 43, 0]
class = drugX
entropy = 0.0
samples = 17
value = [17, 0, 0, 0, 0]
class = drugA
entropy = 0.0
samples = 13
value = [0, 13, 0, 0, 0]
class = drugB
Cholesterol <= 0.5
entropy = 0.999
samples = 23
value = [0, 0, 11, 12, 0]
class = drugX
entropy = 0.0
samples = 31
value = [0, 0, 0, 31, 0]
class = drugX
entropy = 0.0
samples = 11
value = [0, 0, 11, 0, 0]
class = drugC
entropy = 0.0
samples = 12
value = [0, 0, 0, 12, 0]
class = drugX

Lab Experiment 4: Logistic Regression on Pima Diabetes Dataset

Aim
To implement **Logistic Regression** using scikit-learn, evaluate its performance, and visualize the results on the **Pima Diabetes dataset**.

Objectives
1. To understand the **Logistic Regression algorithm** and its application in classification tasks.
2. To preprocess the dataset by handling missing values and **standardizing features**.
3. To **split the dataset** into training and testing sets.
4. To **train and evaluate** the Logistic Regression model.
5. To analyze the model's performance using **accuracy, confusion matrix, classification report, and ROC curve**.

Dataset Description (pima_diabetes.xlsx)
The dataset contains medical diagnostic measurements for **diabetes prediction**. It has the following features:

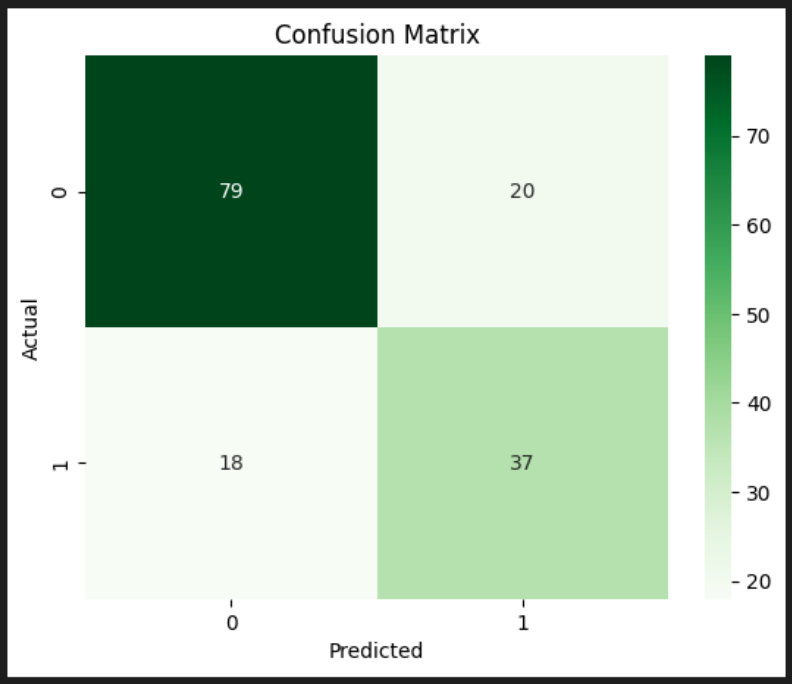
- **Pregnancies**: Number of times pregnant
- **Glucose**: Plasma glucose concentration
- **BloodPressure**: Diastolic blood pressure (mm Hg)
- **SkinThickness**: Triceps skinfold thickness (mm)
- **Insulin**: 2-hour serum insulin (mu U/ml)
- **BMI**: Body Mass Index
- **DiabetesPedigreeFunction**: Genetic predisposition to diabetes
- **Age**: Age of the person
- **Outcome**: Target variable (1 = Diabetic, 0 = Non-Diabetic)

Code –
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the dataset
df = pd.read_excel('pima_diabetes.xlsx')

# Check for missing values
print(df.isnull().sum())

# Separate features and target variable
X = df.drop('Outcome', axis=1)
y = df['Outcome']

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

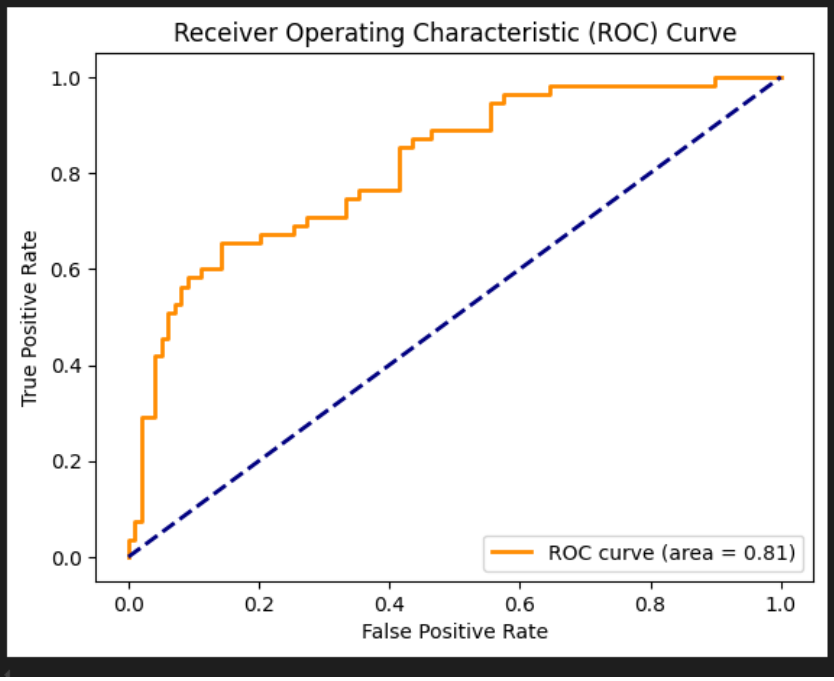
```
# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

from sklearn.linear_model import LogisticRegression

# Initialize the Logistic Regression model
model = LogisticRegression()

# Train the model
model.fit(X_train, y_train)

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_curve, auc
import matplotlib.pyplot as plt
import seaborn as sns

# Predict on the test set
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Greens')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Classification Report
print(classification_report(y_test, y_pred))

# ROC Curve
y_pred_proba = model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

Output –

```
Accuracy: 0.75
```




```
              precision    recall  f1-score   support

           0       0.81      0.80      0.81        99
           1       0.65      0.67      0.66        55

    accuracy                           0.75       154
   macro avg       0.73      0.74      0.73       154
weighted avg       0.76      0.75      0.75       154
```



Lab Experiment 5: Support Vector Machine (SVM) using the Iris dataset

Aim
To implement a **Support Vector Machine (SVM)** using scikit-learn, experiment with different kernel functions, and visualize the decision boundaries.

Objective
- Load and preprocess the **Iris dataset**.
- Train an **SVM classifier** using different kernel functions (linear, rbf, poly).
- Evaluate the model performance using accuracy, confusion matrix, and classification report.
- **Visualize decision boundaries** to compare the impact of different kernels.

Dataset Description: Iris Dataset
The **Iris dataset** is a famous dataset in machine learning and statistics. It consists of **150 samples** of **iris flowers** from three different species:
1. **Setosa**
2. **Versicolor**
3. **Virginica**

Each sample has **four numerical features** that describe the flowers' dimensions.

Dataset Details
- **Total Samples**: 150
- **Classes (Target Variable: Species)**: 3 (Setosa, Versicolor, Virginica)
- **Features (Independent Variables)**: 4
- **Dataset Type**: Supervised Learning (Classification Task)

Features in the Dataset

Column Name	Description	Data Type
Id	Unique identifier for each row	Integer
SepalLengthCm	Length of the sepal (in cm)	Float
SepalWidthCm	Width of the sepal (in cm)	Float
PetalLengthCm	Length of the petal (in cm)	Float
PetalWidthCm	Width of the petal (in cm)	Float
Species	Flower species (Setosa, Versicolor, Virginica)	Categorical

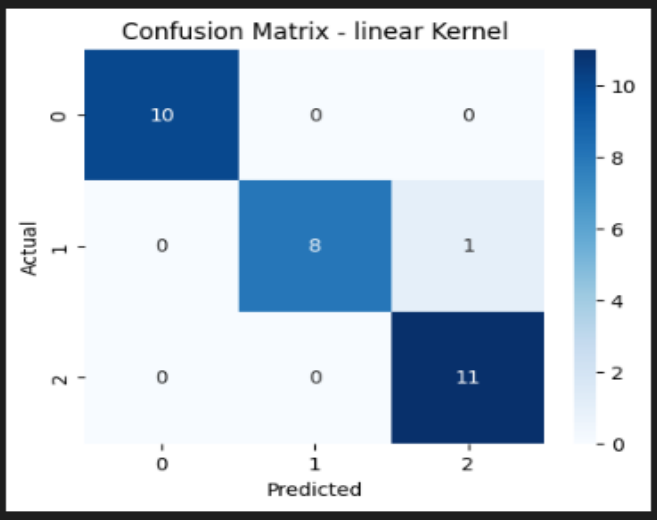
Code –

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

df = pd.read_csv('/mnt/data/Iris.csv')


# Display first 5 rows
print(df.head())

#  Check for missing values
print("\nMissing values in dataset:\n", df.isnull().sum())

#  Encode the categorical target variable (Species)
label_encoder = LabelEncoder()
df['Species'] = label_encoder.fit_transform(df['Species'])

# Split features (X) and target variable (y)
X = df.iloc[:, 1:5].values  # Selecting only numerical columns (ignore 'Id' column)
y = df.iloc[:, 5].values  # Encoded species

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

#  Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

#  Define different SVM classifiers with different kernels
kernels = ['linear', 'rbf', 'poly']
models = {}

for kernel in kernels:
    # Train the SVM model
    model = SVC(kernel=kernel, probability=True, random_state=42)
    model.fit(X_train, y_train)
    models[kernel] = model

    # Make predictions
    y_pred = model.predict(X_test)

    #  Print accuracy
    acc = accuracy_score(y_test, y_pred)
    print(f"\n 🔹 SVM ({kernel} kernel) Accuracy: {acc:.2f}")

    #  Display confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)
```

```
    plt.figure(figsize=(5,4))
    sns.heatmap(conf_matrix, annot=True, cmap="Blues", fmt="d")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title(f"Confusion Matrix - {kernel} Kernel")
    plt.show()

    #  Print classification report
    print(f"\nClassification Report ({kernel} Kernel):\n", classification_report(y_test, y_pred))

#  Decision Boundary Visualization (First 2 features only)
from mlxtend.plotting import plot_decision_regions

# Use only two features for 2D visualization
X_vis = X[:, :2]
X_train_vis, X_test_vis, y_train_vis, y_test_vis = train_test_split(X_vis, y, test_size=0.2,
random_state=42)
X_train_vis = scaler.fit_transform(X_train_vis)
X_test_vis = scaler.transform(X_test_vis)

plt.figure(figsize=(15, 5))

for i, kernel in enumerate(kernels):
    plt.subplot(1, 3, i+1)
    model = SVC(kernel=kernel, random_state=42).fit(X_train_vis, y_train_vis)
    plot_decision_regions(X_train_vis, y_train_vis, clf=model, legend=2)
    plt.title(f"SVM Decision Boundary ({kernel} kernel)")

plt.tight_layout()
plt.show()
```

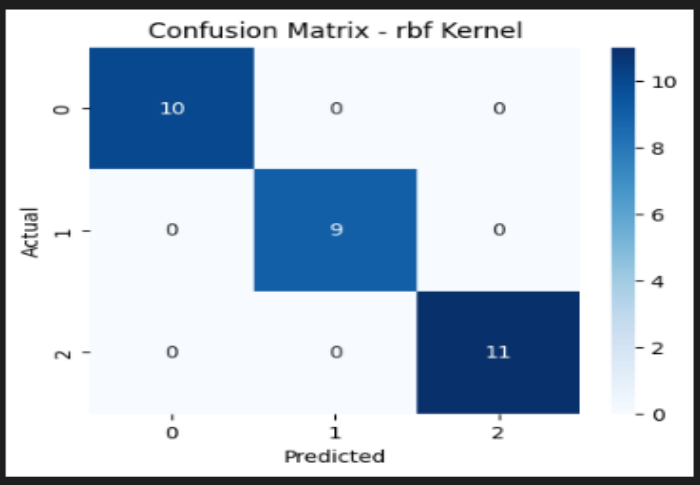
Output –

```
SVM (linear kernel) Accuracy: 0.97
```



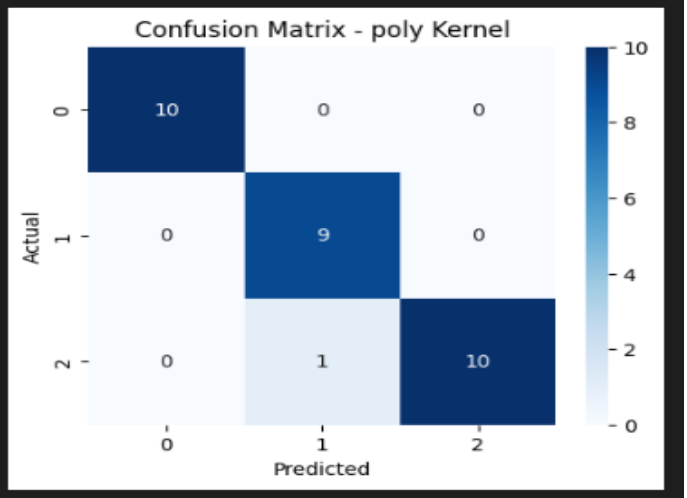

```
Classification Report (linear Kernel):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.89      0.94         9
           2       0.92      1.00      0.96        11

    accuracy                           0.97        30
   macro avg       0.97      0.96      0.97        30
weighted avg       0.97      0.97      0.97        30
```

```
SVM (rbf kernel) Accuracy: 1.00
```



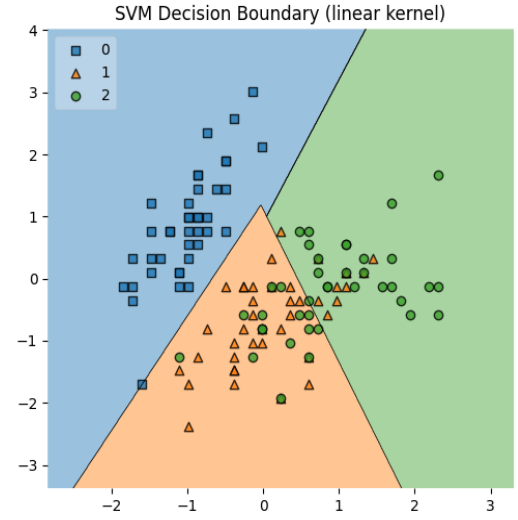

```
Classification Report (rbf Kernel):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       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
```

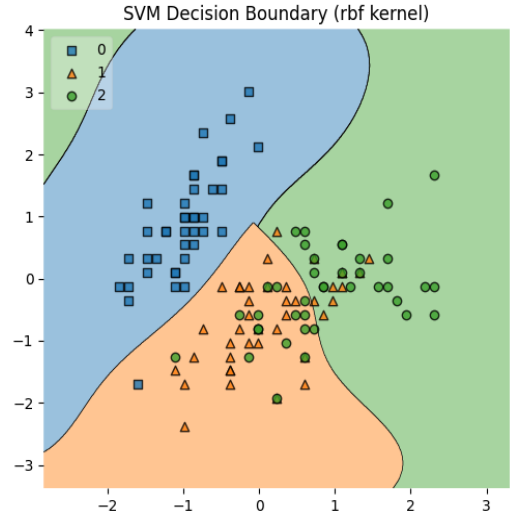
SVM (poly kernel) Accuracy: 0.97

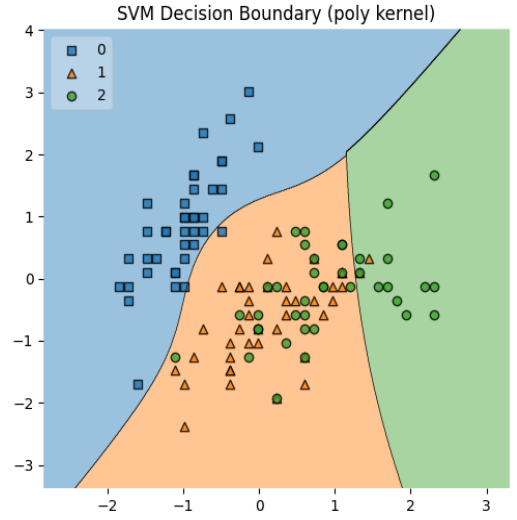


Classification Report (poly Kernel):

	precision	recall	f1-score	support
0	1.00	1.00	1.00	10
1	0.90	1.00	0.95	9
2	1.00	0.91	0.95	11
accuracy			0.97	30
macro avg	0.97	0.97	0.97	30
weighted avg	0.97	0.97	0.97	30







Lab Experiment 6: Implementation and Comparison of Ensemble Learning Techniques (Random Forest, AdaBoost, and XGBoost)

Aim:
To implement ensemble learning methods using scikit-learn, including Random Forest, Adaptive Boosting (AdaBoost), and Extreme Gradient Boosting (XGBoost), and compare their performance with individual classifiers.

Objectives:

- Implement individual classification models such as Logistic Regression and Decision Trees.
- Implement ensemble learning techniques like Random Forest, AdaBoost, and XGBoost.
- Evaluate the performance of each model using accuracy, confusion matrix, classification report, and ROC curves.
- Compare the performance of individual classifiers with ensemble models to analyze their effectiveness.

Dataset Description:
The Heart Disease UCI dataset is used for this experiment. It contains patient health records, including attributes like age, sex, cholesterol levels, blood pressure, and other medical indicators. The target variable (num) indicates the presence (1) or absence (0) of heart disease.

Summary:
In this experiment, multiple machine learning models were trained to predict heart disease. The dataset was preprocessed by handling missing values and categorical features. Various ensemble methods like Random Forest, AdaBoost, and XGBoost were implemented and compared against baseline models like Logistic Regression and Decision Tree. Performance was evaluated using accuracy, confusion matrices, and ROC curves, demonstrating that ensemble techniques generally improve classification performance.

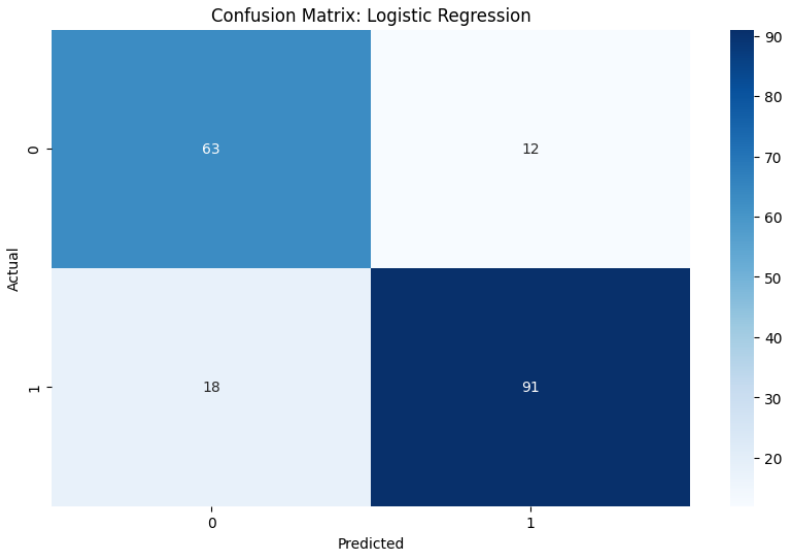
Code –

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_curve, auc

# Load dataset
df = pd.read_csv("heart_disease_uci.csv")

# Drop any unnecessary columns (adjust based on dataset content)
df.drop(columns=['id'], inplace=True, errors='ignore')

# Separate features and target
X = df.drop(columns=['num'])  # Features
y = df['num'].apply(lambda x: 1 if x > 0 else 0)  # Convert target to binary classification

# Convert categorical features to numerical
X = pd.get_dummies(X, drop_first=True)  # One-hot encoding for categorical data

# Handle missing values
imputer = SimpleImputer(strategy="mean")  # Fill missing values with mean
X = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

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

# Standardization
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize models
models = {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(n_estimators=100),
    "AdaBoost": AdaBoostClassifier(n_estimators=100),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='logloss')
}

# Train and evaluate models
results = {}
plt.figure(figsize=(10, 6))
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
```

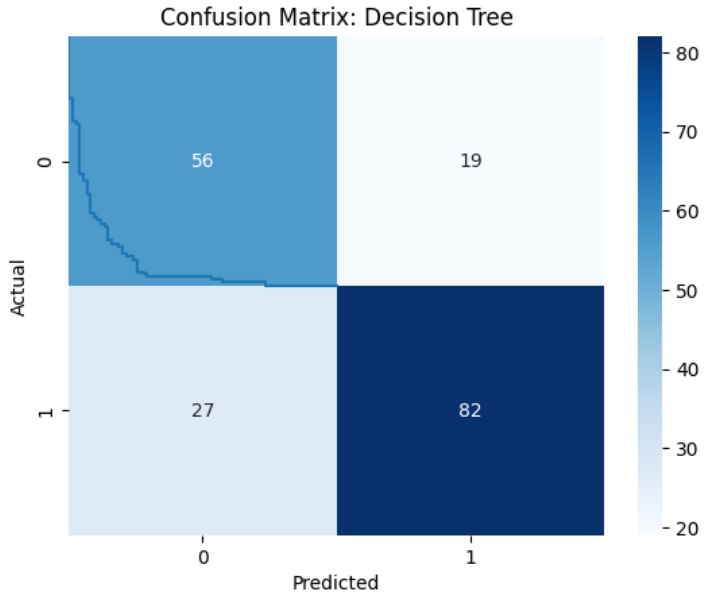
```
    accuracy = accuracy_score(y_test, y_pred)
    results[name] = accuracy
    print(f"\n{name} Accuracy: {accuracy:.4f}")
    print(classification_report(y_test, y_pred))

    # Confusion Matrix
    conf_matrix = confusion_matrix(y_test, y_pred)
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(f'Confusion Matrix: {name}')
    plt.show()

    # ROC Curve
    y_pred_prob = model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_pred_prob)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves for All Models')
plt.legend()
plt.show()

# Final Model Comparison
print("\nFinal Model Comparison:")
for model, acc in results.items():
    print(f"{model}: {acc:.4f}")
```

Output –

```
Logistic Regression Accuracy: 0.8370
              precision    recall  f1-score   support

           0       0.78      0.84      0.81        75
           1       0.88      0.83      0.86       109

    accuracy                           0.84       184
   macro avg       0.83      0.84      0.83       184
weighted avg       0.84      0.84      0.84       184
```

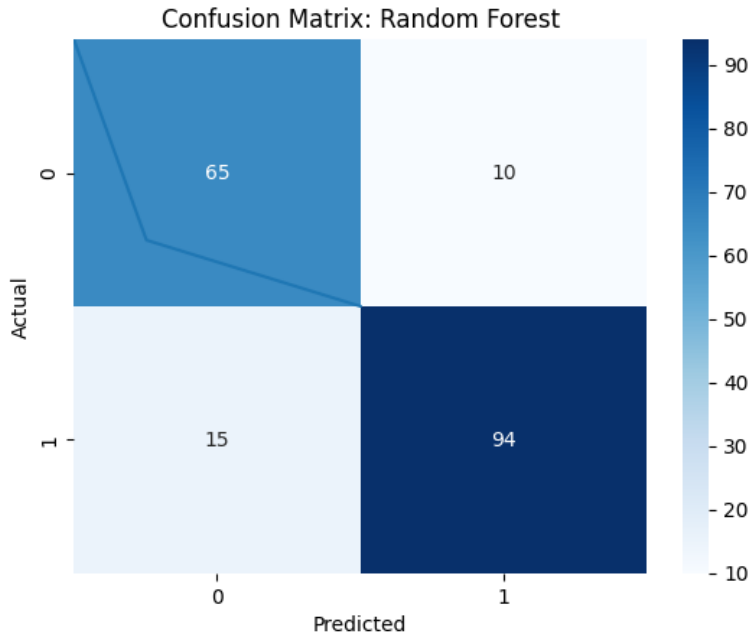



```
Decision Tree Accuracy: 0.7500
              precision    recall  f1-score   support

           0       0.67      0.75      0.71        75
           1       0.81      0.75      0.78       109

    accuracy                           0.75       184
   macro avg       0.74      0.75      0.74       184
weighted avg       0.76      0.75      0.75       184
```



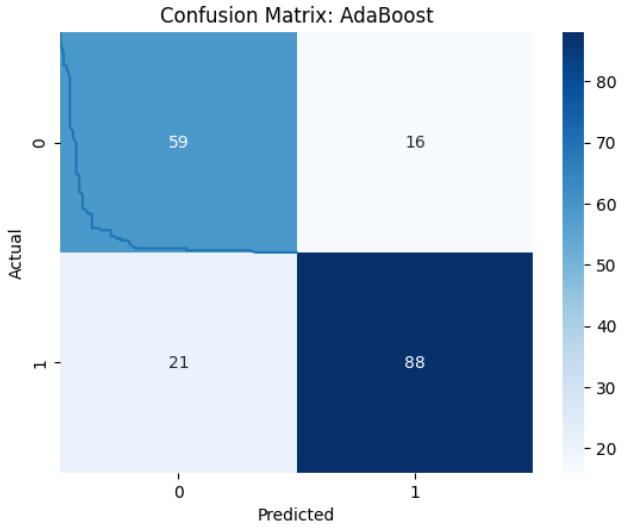
Random Forest Accuracy: 0.8641

	precision	recall	f1-score	support
0	0.81	0.87	0.84	75
1	0.90	0.86	0.88	109
accuracy			0.86	184
macro avg	0.86	0.86	0.86	184
weighted avg	0.87	0.86	0.86	184



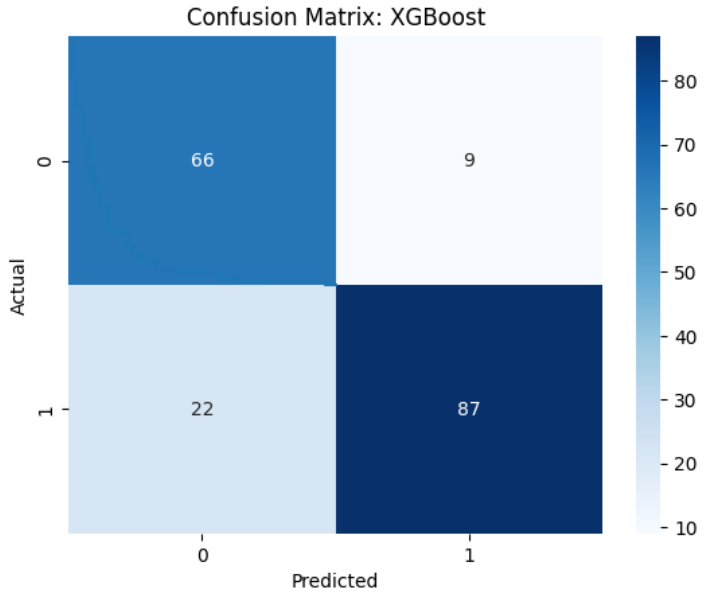
```
c:\Users\Ansh\test\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527:
FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be
removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
```

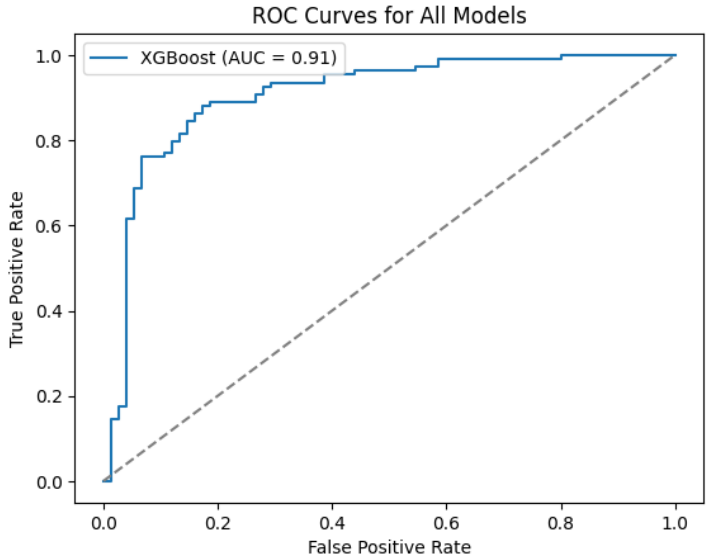
```
AdaBoost Accuracy: 0.7989
              precision    recall  f1-score   support

           0       0.74      0.79      0.76        75
           1       0.85      0.81      0.83       109

    accuracy                           0.80       184
   macro avg       0.79      0.80      0.79       184
weighted avg       0.80      0.80      0.80       184
```




```
c:\Users\Ansh\test\Lib\site-packages\xgboost\training.py:183: UserWarning:
[04:37:59] WARNING: C:\actions-
runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.
  bst.update(dtrain, iteration=i, fobj=obj)
```

```
XGBoost Accuracy: 0.8315
              precision    recall  f1-score   support

           0       0.75      0.88      0.81        75
           1       0.91      0.80      0.85       109

    accuracy                           0.83       184
   macro avg       0.83      0.84      0.83       184
weighted avg       0.84      0.83      0.83       184
```






```
Final Model Comparison:
Logistic Regression: 0.8370
Decision Tree: 0.7500
Random Forest: 0.8641
AdaBoost: 0.7989
XGBoost: 0.8315
```

Experiment 7: Clustering Algorithms Comparison

Aim:

To implement and compare three clustering algorithms — **K-Means**, **Hierarchical (Agglomerative)**, and **Density-Based (DBSCAN)** — and visualize their performance on a non-linearly separable dataset.

Objectives:

1. To understand the working principles of K-Means, Hierarchical, and DBSCAN clustering.
2. To implement these algorithms using scikit-learn.
3. To visualize and interpret the clustering outcomes on complex-shaped datasets.
4. To analyze how different algorithms handle non-spherical clusters and noise.

Dataset Description & Transformation:

- **Dataset Used:** make_moons synthetic dataset from sklearn.datasets.
- **Structure:** 2D dataset with **two interleaving half-moon shapes** (non-linearly separable).
- **Samples:** 300 data points.
- **Noise:** Gaussian noise of 0.15 added to simulate real-world irregularities and test clustering robustness.

```
python
CopyEdit
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=300, noise=0.15, random_state=42)
```

- **Transformations Applied:** No manual feature scaling or dimensionality reduction was required due to the 2D nature of the dataset.

Summary:

In this experiment, we applied three clustering algorithms — **K-Means**, **Hierarchical Clustering**, and **DBSCAN** — to a complex 2D dataset with non-linear patterns.

- **K-Means** performed poorly, as it assumes clusters to be spherical and equally sized.
- **Hierarchical Clustering** handled the data better but still misclassified some regions due to shape assumptions.

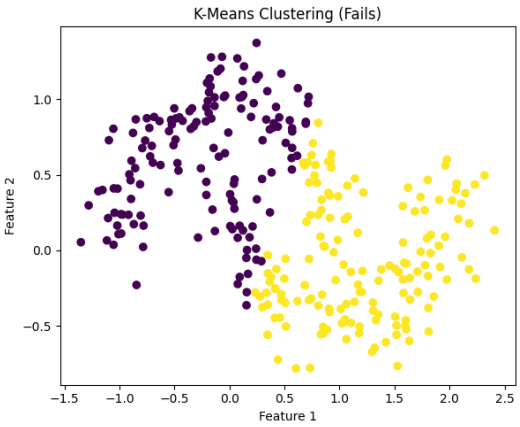
- **DBSCAN**, a density-based clustering technique, performed the best by accurately identifying clusters and treating noise points separately.

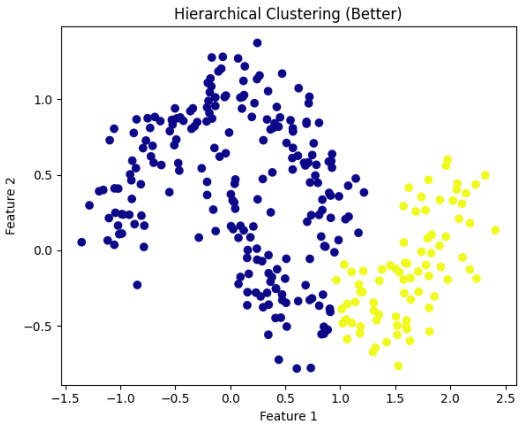
This experiment highlights the importance of **choosing the right clustering technique** based on the shape and nature of data distribution, especially when dealing with real-world, non-linearly separable datasets.

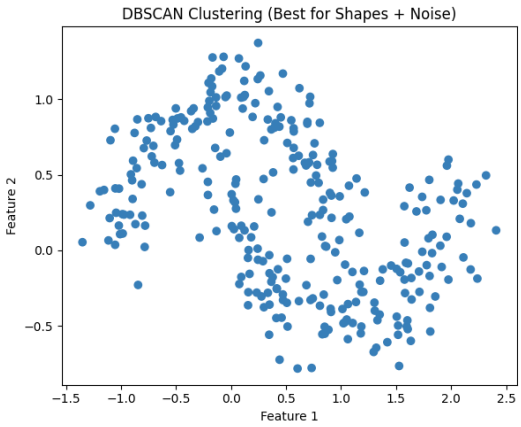
Code –

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
import numpy as np

# Step 1: Generate more challenging moon dataset
X, y = make_moons(n_samples=300, noise=0.15, random_state=42)

# Step 2: Apply K-Means
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans_labels = kmeans.fit_predict(X)

# Step 3: Apply Agglomerative (Hierarchical) Clustering
hierarchical = AgglomerativeClustering(n_clusters=2)
hierarchical_labels = hierarchical.fit_predict(X)

# Step 4: Apply DBSCAN
dbscan = DBSCAN(eps=0.3, min_samples=5)
dbscan_labels = dbscan.fit_predict(X)


# Step 5: Plot the results
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

# K-Means
axes[0].scatter(X[:, 0], X[:, 1], c=kmeans_labels, cmap='viridis')
axes[0].set_title("K-Means Clustering (Fails)")

# Hierarchical
axes[1].scatter(X[:, 0], X[:, 1], c=hierarchical_labels, cmap='plasma')
axes[1].set_title("Hierarchical Clustering (Better)")

# DBSCAN
# Show noise (-1) as black
dbscan_colors = np.array(['#377eb8', '#ff7f00', 'black'])  # Blue, Orange, Black for noise
mapped_colors = [dbscan_colors[label] if label != -1 else 'black' for label in dbscan_labels]
axes[2].scatter(X[:, 0], X[:, 1], c=mapped_colors)
axes[2].set_title("DBSCAN Clustering (Best for Shapes + Noise)")
for ax in axes:
    ax.set_xlabel('Feature 1')
    ax.set_ylabel('Feature 2')
plt.tight_layout()
plt.show()
```

Output –


K-Means Clustering (Fails)
Feature 2
Feature 1
1.0
0.5
0.0
−0.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5


Hierarchical Clustering (Better)
Feature 2
Feature 1
1.0
0.5
0.0
−0.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5


DBSCAN Clustering (Best for Shapes + Noise)
Feature 2
Feature 1
1.0
0.5
0.0
−0.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5

Experiment 8: Classification using k-NN and Decision Tree

Aim

To implement and visualize the performance of the **k-Nearest Neighbors (k-NN)** algorithm on a real-world classification task, compare it with a **Decision Tree classifier**, and analyze how varying the value of `k` affects the accuracy of k-NN.

Objectives

1. Load and preprocess a real-world dataset for binary classification.
2. Train a k-NN classifier and visualize its decision boundaries.
3. Train a Decision Tree classifier for comparison.
4. Evaluate and compare the accuracy of both models.
5. Analyze the impact of different values of `k` on k-NN performance using a line graph.

Dataset Description

Dataset Name: Social Network Ads
Source: Kaggle – Social Network Ads Dataset
Total Records: 400
Features Used:

- `Age` – Age of the user (numerical)
- `EstimatedSalary` – Estimated annual salary of the user (numerical)

Target Variable:

- `Purchased` – Whether the user purchased the product (1 = Yes, 0 = No)

Note: The columns `User ID` and `Gender` were excluded to simplify the analysis and make the data suitable for 2D visualization.

Summary of the Experiment

In this experiment, we performed a classification task using two supervised learning models: **k-Nearest Neighbors (k-NN)** and **Decision Tree**. The dataset chosen included user demographics such as age and salary to predict whether a user purchased a product.

After scaling the features, both models were trained and tested. Their **decision boundaries** were visualized to understand how each model separates the classes in feature space.

- The **k-NN classifier (with k=5)** provided a smooth, flexible boundary and achieved strong classification performance.
- The **Decision Tree** produced blockier decision regions and had a comparable accuracy, though it may be prone to overfitting.

Finally, we analyzed the effect of changing `k` from 1 to 20 on k-NN accuracy. The results showed that:

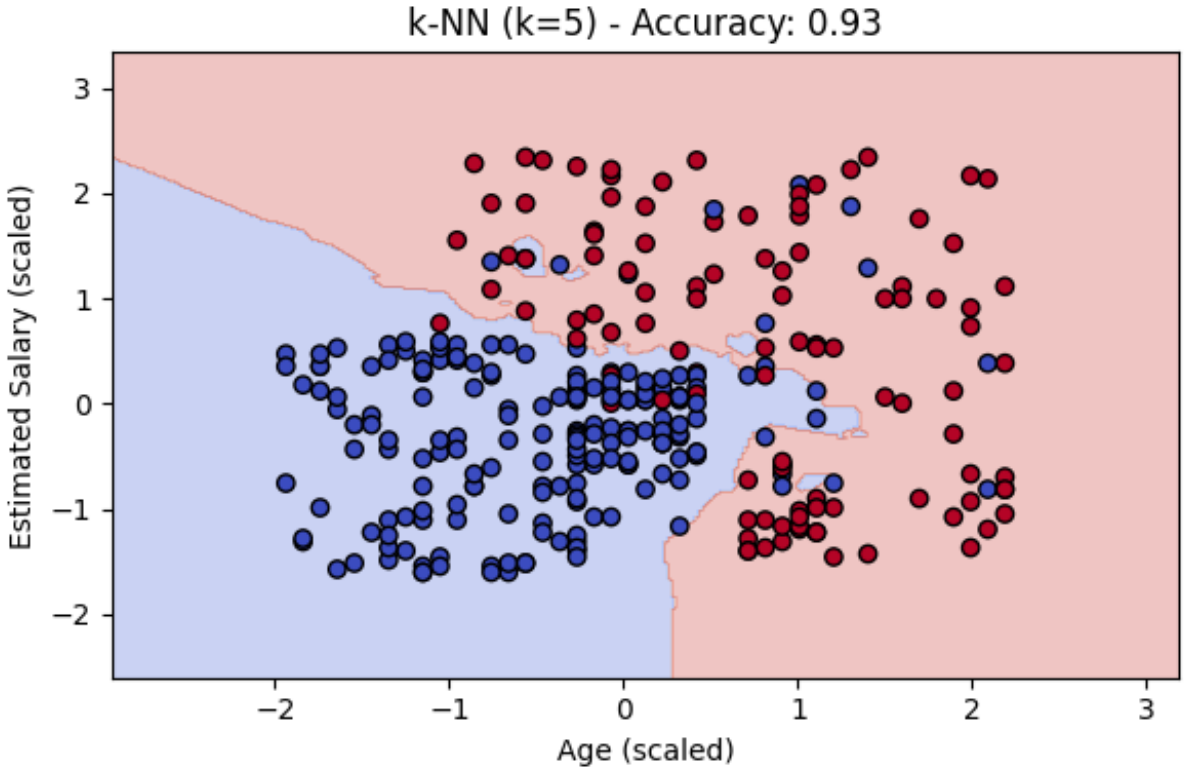
- Low `k` values tend to overfit (high variance),
- High `k` values underfit (high bias),
- And there's a **sweet spot** (usually k=5–10) that offers the best generalization.

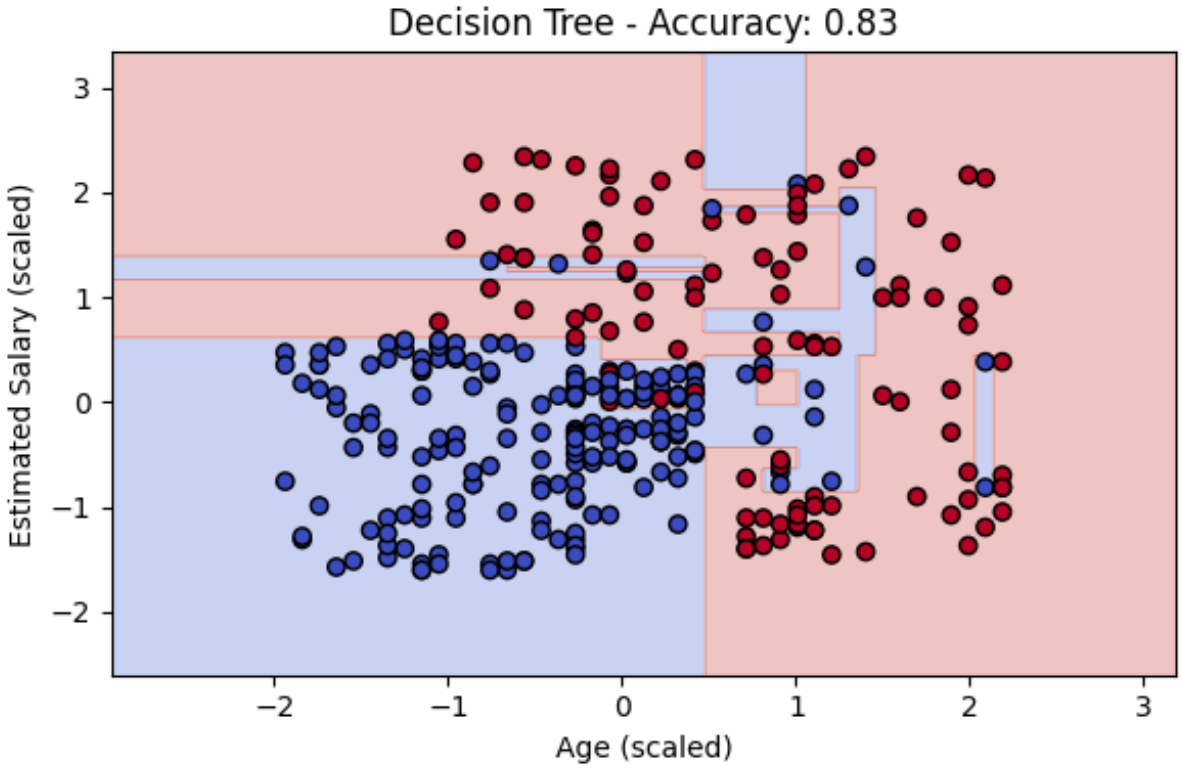
Code –

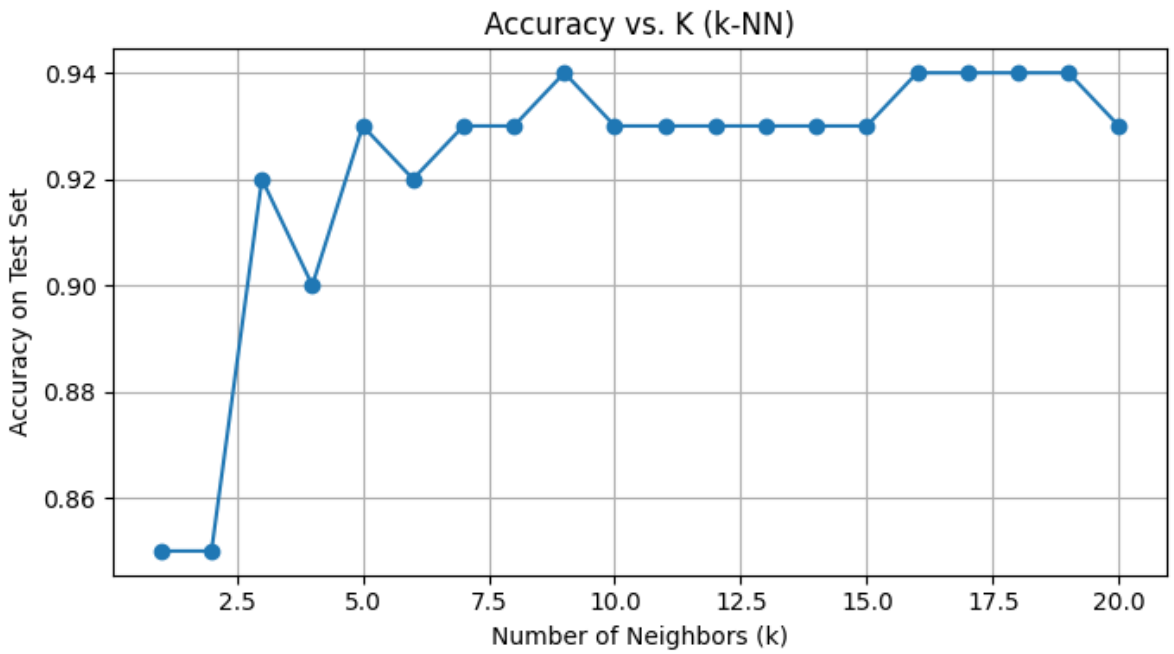
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load dataset
df = pd.read_csv("Social_Network_Ads.csv")

# Preprocessing
df = df.drop(columns=['User ID', 'Gender'])  # Drop unneeded columns
X = df[['Age', 'EstimatedSalary']].values
y = df['Purchased'].values

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train k-NN (k=5)
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)
y_pred_knn = knn.predict(X_test_scaled)
knn_acc = accuracy_score(y_test, y_pred_knn)

# Train Decision Tree
tree = DecisionTreeClassifier(random_state=42)
```

```
tree.fit(X_train_scaled, y_train)
y_pred_tree = tree.predict(X_test_scaled)
tree_acc = accuracy_score(y_test, y_pred_tree)

# Decision boundary plot function
def plot_decision_boundary(clf, X, y, title):
    h = 0.02
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.figure(figsize=(6, 4))
    plt.contourf(xx, yy, Z, alpha=0.3, cmap='coolwarm')
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', edgecolor='k')
    plt.title(title)
    plt.xlabel('Age (scaled)')
    plt.ylabel('Estimated Salary (scaled)')
    plt.tight_layout()
    plt.show()

# Visualize decision boundaries
plot_decision_boundary(knn, X_train_scaled, y_train, f"k-NN (k=5) - Accuracy: {knn_acc:.2f}")
plot_decision_boundary(tree, X_train_scaled, y_train, f"Decision Tree - Accuracy: {tree_acc:.2f}")

# Accuracy vs. K plot
k_values = range(1, 21)
accuracies = []

for k in k_values:
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train_scaled, y_train)
    y_pred_k = model.predict(X_test_scaled)
    accuracies.append(accuracy_score(y_test, y_pred_k))

plt.figure(figsize=(7, 4))
plt.plot(k_values, accuracies, marker='o')
plt.title("Accuracy vs. K (k-NN)")
plt.xlabel("Number of Neighbors (k)")
plt.ylabel("Accuracy on Test Set")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Output –


k-NN (k=5) - Accuracy: 0.93
Estimated Salary (scaled)
Age (scaled)
3
2
1
0
−1
−2
−2
−1
0
1
2
3


Decision Tree - Accuracy: 0.83
Estimated Salary (scaled)
Age (scaled)
3
2
1
0
−1
−2
−2
−1
0
1
2
3


Accuracy vs. K (k-NN)
Accuracy on Test Set
0.94
0.92
0.90
0.88
0.86
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Number of Neighbors (k)

Lab Experiment 9: Linear Regression for Housing Price Prediction

---

Aim

To train a Linear Regression model to predict housing prices based on features such as area, number of rooms, pollution level, etc., and to explore the effects of **outliers** and **feature scaling** on model performance.

Objectives

1. To understand and implement the Linear Regression algorithm.
2. To train a model to predict housing prices using the Boston Housing dataset.
3. To evaluate model performance using MSE and R^2 score.
4. To identify and analyze the impact of **outliers** on the model.
5. To apply **feature scaling** and assess its impact on prediction accuracy.

Dataset Description

- **Dataset Name:** Boston Housing Dataset (HousingData.csv)
- **Records:** 506
- **Attributes:** 13 features + 1 target (MEDV)
- **Target Variable:** MEDV (Median value of owner-occupied homes in $1000s)
- **Missing Values:** Present in several columns, handled by dropping rows with missing data.

Example Features:

- RM: Average number of rooms per dwelling
- LSTAT: % lower status of the population
- CRIM: Crime rate per capita
- NOX: Nitric oxide concentration
- PTRATIO: Pupil-teacher ratio by town

Summary

In this experiment, we built a Linear Regression model to predict housing prices in Boston. We evaluated model performance using metrics like MSE and R^2 under three different scenarios: using raw data, after removing outliers, and after applying feature scaling. We observed that outlier removal dramatically improved model accuracy, while feature scaling had minimal impact on this specific model type.

The experiment highlights the importance of **data cleaning and preprocessing** in building accurate and reliable ML models, especially in real estate or finance where data irregularities are common.

Code –

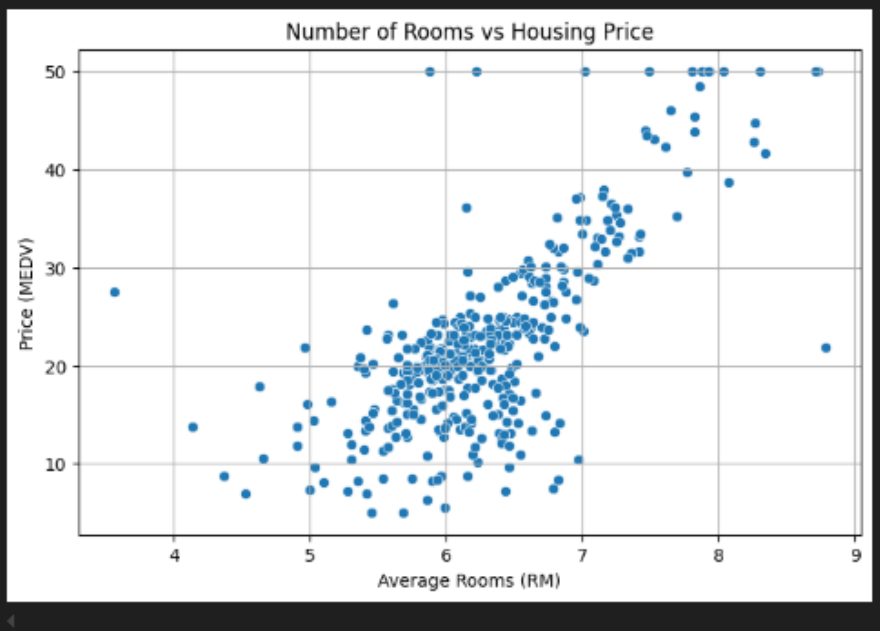
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
import warnings

warnings.filterwarnings("ignore")

# Step 1: Load the dataset
df = pd.read_csv("HousingData.csv")

# Step 2: Show basic info and missing values
print("Original Data Info:")
print(df.info())
print("\nMissing values per column:\n", df.isnull().sum())

# Step 3: Drop missing values
df_clean = df.dropna()
print(f"\nAfter dropping missing values: {df_clean.shape[0]} rows remain.")

# Step 4: Define features and target
X = df_clean.drop(columns=["MEDV"])
y = df_clean["MEDV"]

# Step 5: Visualize feature vs target (e.g., RM vs MEDV)
plt.figure(figsize=(8, 5))
sns.scatterplot(data=df_clean, x="RM", y="MEDV")
plt.title("Number of Rooms vs Housing Price")
plt.xlabel("Average Rooms (RM)")
plt.ylabel("Price (MEDV)")
plt.grid(True)
plt.show()

# Step 6: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 7: Train baseline linear regression model
baseline_model = LinearRegression()
baseline_model.fit(X_train, y_train)
y_pred_baseline = baseline_model.predict(X_test)

# Step 8: Evaluate baseline model
print("\n--- Baseline Model (with outliers) ---")
print("MSE:", mean_squared_error(y_test, y_pred_baseline))
print("R² Score:", r2_score(y_test, y_pred_baseline))

# Plot actual vs predicted
plt.figure(figsize=(6, 4))
sns.scatterplot(x=y_test, y=y_pred_baseline)
```

```
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Baseline Model Predictions")
plt.plot([0, 50], [0, 50], 'r--')
plt.grid(True)
plt.show()

# Step 9: Outlier Removal using IQR method
Q1 = df_clean.quantile(0.25)
Q3 = df_clean.quantile(0.75)
IQR = Q3 - Q1
df_no_outliers = df_clean[~((df_clean < (Q1 - 1.5 * IQR)) | (df_clean > (Q3 + 1.5 *
IQR))).any(axis=1)]
print(f"\nAfter removing outliers: {df_no_outliers.shape[0]} rows remain.")

# Redefine features and target
X_no = df_no_outliers.drop(columns=["MEDV"])
y_no = df_no_outliers["MEDV"]

X_train_no, X_test_no, y_train_no, y_test_no = train_test_split(X_no, y_no, test_size=0.2,
random_state=42)

# Train model on cleaned data
model_no_outliers = LinearRegression()
model_no_outliers.fit(X_train_no, y_train_no)
y_pred_no = model_no_outliers.predict(X_test_no)

# Evaluate
print("\n--- Model After Outlier Removal ---")
print("MSE:", mean_squared_error(y_test_no, y_pred_no))
print("R² Score:", r2_score(y_test_no, y_pred_no))

# Plot
plt.figure(figsize=(6, 4))
sns.scatterplot(x=y_test_no, y=y_pred_no)
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Model After Outlier Removal")
plt.plot([0, 50], [0, 50], 'r--')
plt.grid(True)
plt.show()

# Step 10: Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_no)
X_test_scaled = scaler.transform(X_test_no)

# Train model after scaling
model_scaled = LinearRegression()
model_scaled.fit(X_train_scaled, y_train_no)
y_pred_scaled = model_scaled.predict(X_test_scaled)

# Evaluate
print("\n--- Model After Feature Scaling ---")
print("MSE:", mean_squared_error(y_test_no, y_pred_scaled))
```

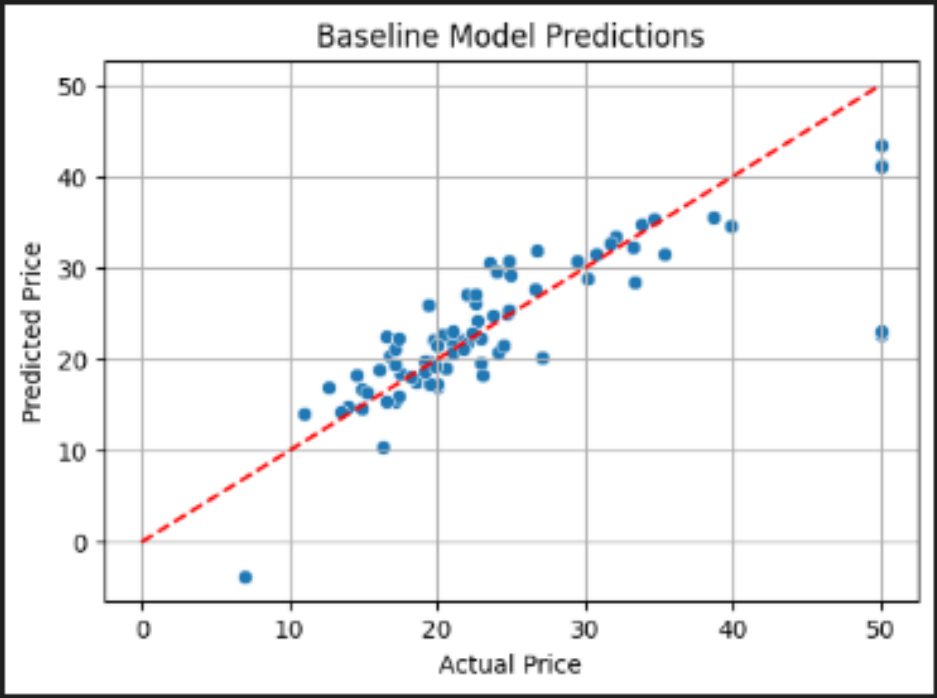
```
print("R² Score:", r2_score(y_test_no, y_pred_scaled))

# Plot
plt.figure(figsize=(6, 4))
sns.scatterplot(x=y_test_no, y=y_pred_scaled)
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Model After Feature Scaling")
plt.plot([0, 50], [0, 50], 'r--')
plt.grid(True)
plt.show()
```

Output –

```
Original Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     486 non-null    float64
 1   ZN       486 non-null    float64
 2   INDUS    486 non-null    float64
 3   CHAS     486 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      486 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64
 9   TAX      506 non-null    int64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    486 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
None

Missing values per column:
...
MEDV       0
dtype: int64

After dropping missing values: 394 rows remain.
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

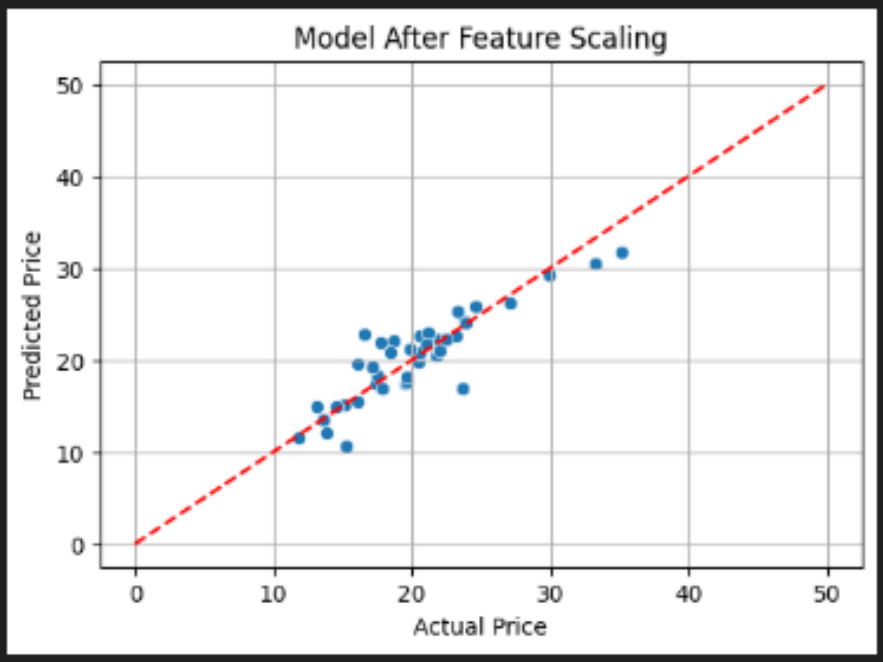


```
--- Baseline Model (with outliers) ---
MSE: 31.454047664950963
R² Score: 0.627084994167318
```




```
After removing outliers: 214 rows remain.

--- Model After Outlier Removal ---
MSE: 5.098328945973743
R² Score: 0.7824596647797841
```

```
--- Model After Feature Scaling ---
MSE: 5.098328945973737
R² Score: 0.7824596647797843
```



Lab Experiment 10: PCA for Dimensionality Reduction

Aim

To implement Principal Component Analysis (PCA) for reducing the dimensionality of a dataset and to visualize its effect using 2D scatter plots.

Objectives

1. Understand the concept and math behind PCA.
2. Reduce high-dimensional data to 2D or 3D while preserving variance.
3. Visualize original and reduced data.
4. Observe how PCA helps in simplifying data without significant information loss.

Dataset Description (Iris)

- **Name:** Iris Flower Dataset
- **Total Samples:** 150
- **Features:** 4 numeric features
  - Sepal length
  - Sepal width
  - Petal length
  - Petal width
- **Target:** Species of iris flower (Setosa, Versicolor, Virginica)
- **Type:** Multiclass classification

Theory

What is PCA?

PCA is a linear transformation technique that reduces the number of features in a dataset by projecting it onto a new set of axes (principal components) which capture the maximum variance.

Key Math:

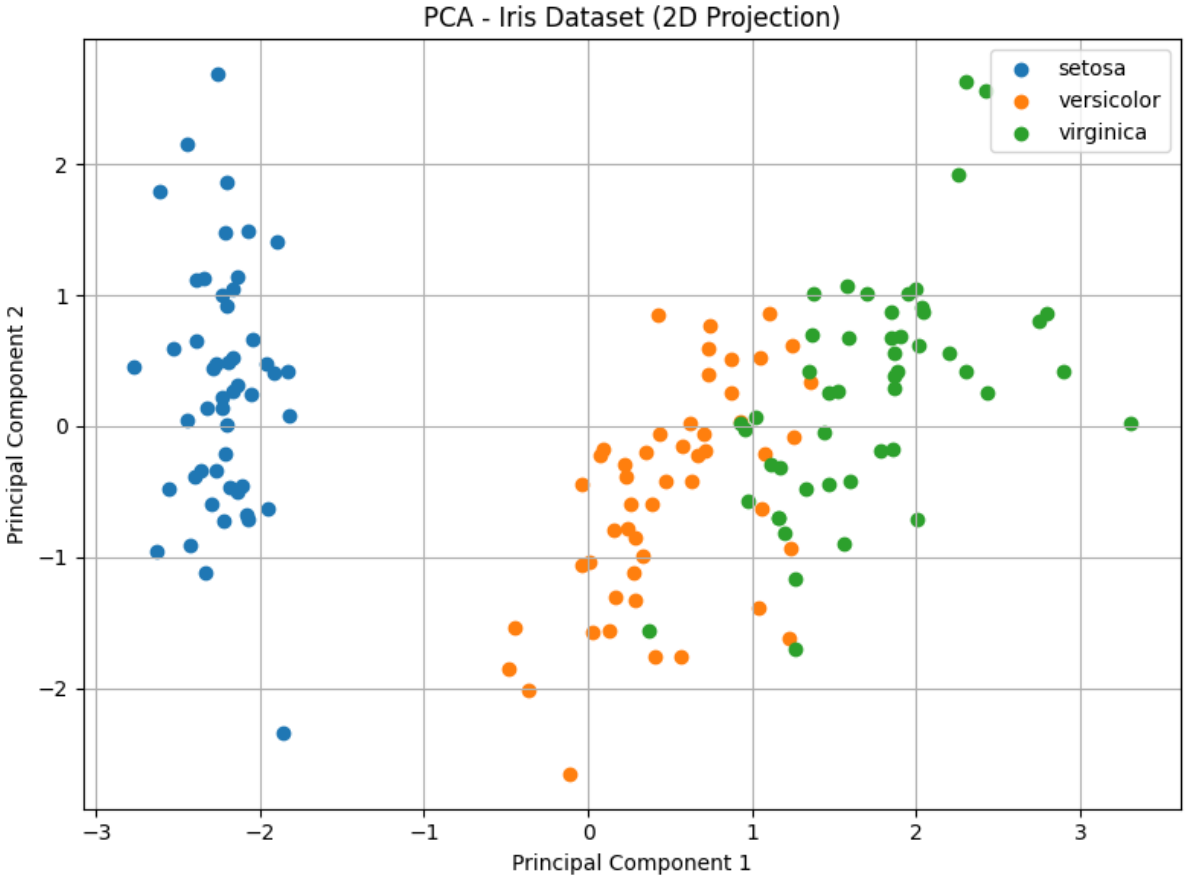
1. Center the data by subtracting the mean.
2. Compute covariance matrix of the data.
3. Compute eigenvalues and eigenvectors.
4. Select top k eigenvectors (with highest eigenvalues).

5. Transform original data to the new space.

Code –

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Step 1: Load and prepare data
iris = load_iris()
X = iris.data
y = iris.target
labels = iris.target_names
features = iris.feature_names

# Step 2: Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Apply PCA
pca = PCA(n_components=2)  # Reduce to 2D
X_pca = pca.fit_transform(X_scaled)

# Step 4: Visualize
plt.figure(figsize=(8, 6))
for i, label in enumerate(np.unique(y)):
    plt.scatter(X_pca[y == label, 0], X_pca[y == label, 1], label=labels[label])

plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA - Iris Dataset (2D Projection)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Step 5: Explained Variance
print("Explained Variance Ratio:", pca.explained_variance_ratio)
```

Output –



Explained Variance Ratio: [0.72962445 0.22850762]
Total Variance Retained: 0.9581320720000166