**Practical No. 1**

**Implement Linear Regression (Diabetes Dataset)**

**Aim:** To implement Linear Regression using the Diabetes dataset and evaluate its performance.

**Solution:**

**Code:**

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets, linear\_model

from sklearn.metrics import mean\_squared\_error, r2\_score

# Load the diabetes dataset

diabetes = datasets.load\_diabetes()

X = diabetes.data[:, np.newaxis, 2] # Use only one feature (BMI)

y = diabetes.target

# Split into training and testing sets

X\_train, X\_test = X[:-20], X[-20:]

y\_train, y\_test = y[:-20], y[-20:]

# Create and train the linear regression model

regr = linear\_model.LinearRegression()

regr.fit(X\_train, y\_train)

# Make predictions

y\_pred = regr.predict(X\_test)

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f"Mean Squared Error: {mse:.2f}")

print(f"R2 Score: {r2:.2f}")

# Plot the results

plt.scatter(X\_test, y\_test, color="black")

plt.plot(X\_test, y\_pred, color="blue", linewidth=3)

plt.xlabel("BMI")

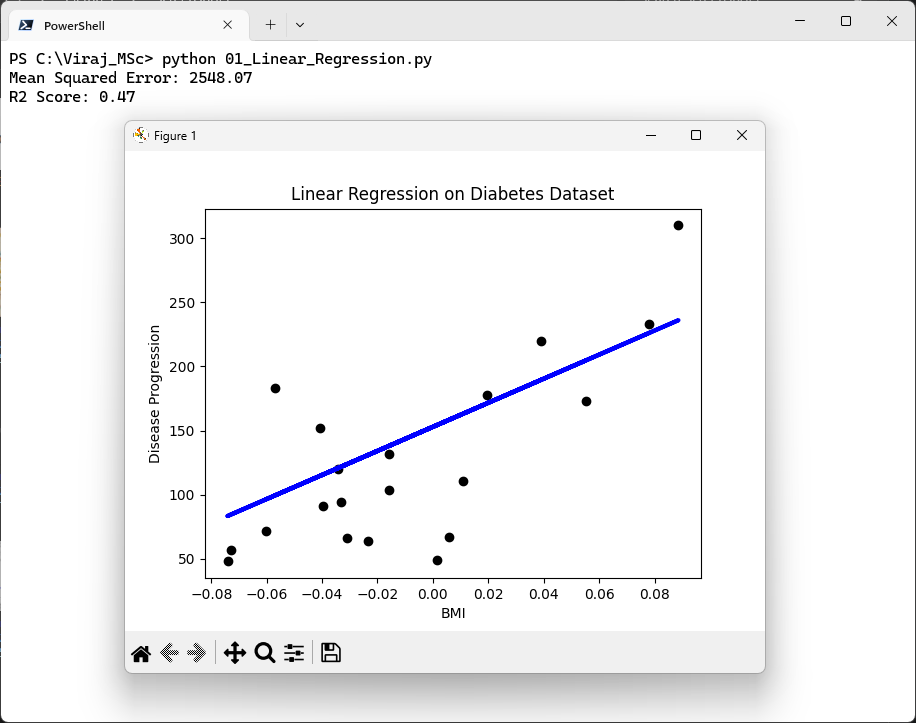
plt.ylabel("Disease Progression")

plt.title("Linear Regression on Diabetes Dataset")

plt.show()

**Observations and Results:**

* The model achieved an MSE of 2548.07 and an R² score of 0.47.
* The plot shows the linear relationship between BMI and disease progression.

**Output:**

**Practical No. 2**

**Implement Logistic Regression (Iris Dataset)**

**Aim:** To classify Iris flowers into two classes using Logistic Regression.

**Solution:**

**Code:**

from sklearn import datasets

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, confusion\_matrix

# Load the Iris dataset

iris = datasets.load\_iris()

X = iris.data[:, :2] # Using only first two features (sepal length & width)

y = (iris.target != 0).astype(int) # Binary classification (Setosa vs Non-Setosa)

# Split 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 the logistic regression model

clf = LogisticRegression()

clf.fit(X\_train, y\_train)

# Make predictions

y\_pred = clf.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

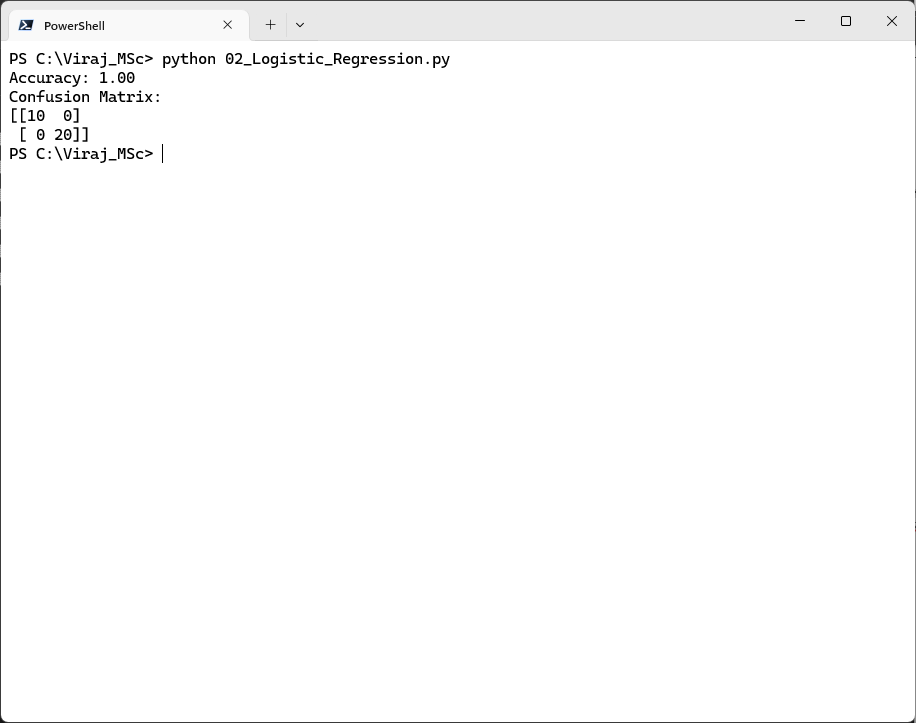
print(f"Accuracy: {accuracy:.2f}")

print("Confusion Matrix:")

print(conf\_matrix)

**Observations and Results:**

* The model achieved an accuracy of 1.00 (100%) on the test set.
* The confusion matrix confirms perfect classification for this binary task.

**Output:**

**Practical No. 3**

**Implement Multinomial Logistic Regression (Iris Dataset)**

**Aim:** To classify Iris flowers into all three classes using Multinomial Logistic Regression.

**Solution:**

**Code:**

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report

# Load the Iris dataset

iris = load\_iris()

# Use all 4 features for multiclass classification

X = iris.data

y = iris.target

# Split 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 the multinomial logistic regression model

# Removed multi\_class="multinomial" as it's the default for 'lbfgs' in newer versions

clf = LogisticRegression(solver="lbfgs", max\_iter=1000)

clf.fit(X\_train, y\_train)

# Make predictions

y\_pred = clf.predict(X\_test)

# Evaluate the model

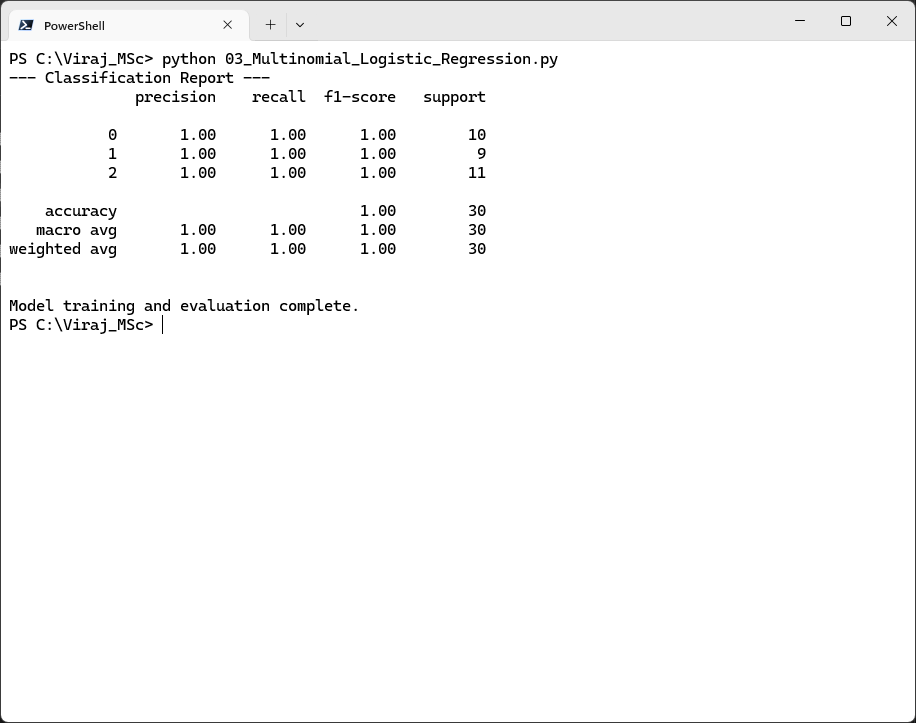
print("--- Classification Report ---")

print(classification\_report(y\_test, y\_pred))

print("\nModel training and evaluation complete.")

**Observations and Results:**

* The model achieved 100% accuracy on the test set.
* The classification report shows perfect precision, recall, and F1-score for all three classes.

**Output:**

**Practical No. 4**

**Implement SVM Classifier (Iris Dataset)**

**Aim:** To classify Iris flowers using a Support Vector Machine (SVM).

**Solution:**

**Code:**

from sklearn.datasets import load\_iris # Import the iris dataset

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import classification\_report

# Load the Iris dataset

iris = load\_iris()

# Use all 4 features for classification

X = iris.data

y = iris.target

# Split 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 an SVM classifier

# Using a linear kernel for simplicity and often good performance on Iris

svm\_clf = SVC(kernel="linear", random\_state=42) # Added random\_state for reproducibility

svm\_clf.fit(X\_train, y\_train)

# Make predictions

y\_pred = svm\_clf.predict(X\_test)

# Evaluate the model

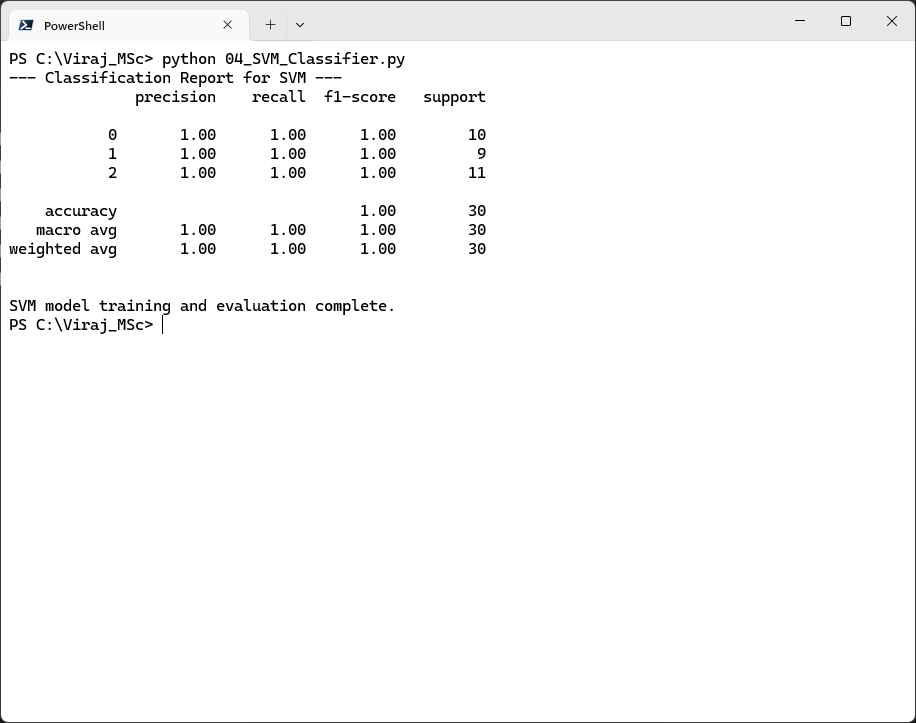
print("--- Classification Report for SVM ---")

print(classification\_report(y\_test, y\_pred))

print("\nSVM model training and evaluation complete.")

**Observations and Results:**

* The SVM achieved 100% accuracy on the test set.
* The linear kernel worked well for this dataset.

**Output:**

**Practical No. 5**

**Train and Fine-tune a Decision Tree for the Moons Dataset**

**Aim:** To train a Decision Tree classifier on a synthetic Moons dataset and fine-tune hyperparameters.

**Solution:**

**Code:**

import matplotlib.pyplot as plt # For plotting the dataset

from sklearn.datasets import make\_moons

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import GridSearchCV, train\_test\_split # Added train\_test\_split

from sklearn.metrics import classification\_report # Added classification\_report

import numpy as np # For creating a meshgrid for plotting decision boundary

# Generate the Moons dataset

X, y = make\_moons(n\_samples=1000, noise=0.3, random\_state=42)

# Optional: Plot the generated dataset to visualize it

plt.figure(figsize=(8, 6))

plt.scatter(X[y == 0, 0], X[y == 0, 1], c='red', marker='o', label='Class 0')

plt.scatter(X[y == 1, 0], X[y == 1, 1], c='blue', marker='x', label='Class 1')

plt.title("Synthetic Moons Dataset")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.legend()

plt.grid(True)

plt.show()

# Split 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)

# Fine-tune hyperparameters using GridSearchCV

params = {

"max\_depth": [3, 5, 10, None], # None means unlimited depth

"min\_samples\_split": [2, 5, 10]

}

# Initialize DecisionTreeClassifier with a random\_state for reproducibility

dt\_clf = DecisionTreeClassifier(random\_state=42)

grid\_search = GridSearchCV(dt\_clf, params, cv=5, scoring='accuracy', n\_jobs=-1) # n\_jobs=-1 uses all CPU cores

grid\_search.fit(X\_train, y\_train)

# Best model

best\_tree = grid\_search.best\_estimator\_

y\_pred = best\_tree.predict(X\_test)

print("--- Decision Tree Hyperparameter Tuning Results ---")

print(f"Best Parameters: {grid\_search.best\_params\_}")

print(f"Best Cross-validation Accuracy: {grid\_search.best\_score\_:.4f}")

print("\n--- Classification Report on Test Set ---")

print(classification\_report(y\_test, y\_pred))

# Optional: Plot the decision boundary of the best model

plt.figure(figsize=(10, 8))

x\_min, x\_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5

y\_min, y\_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5

xx, yy = np.meshgrid(np.linspace(x\_min, x\_max, 100),

np.linspace(y\_min, y\_max, 100))

Z = best\_tree.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.RdBu)

plt.scatter(X\_test[:, 0], X\_test[:, 1], c=y\_test, s=40, edgecolor='k', cmap=plt.cm.RdBu)

plt.title(f"Decision Boundary of Best Decision Tree (Max Depth: {best\_tree.max\_depth}, Min Samples Split: {best\_tree.min\_samples\_split})")

plt.xlabel("Feature 1")

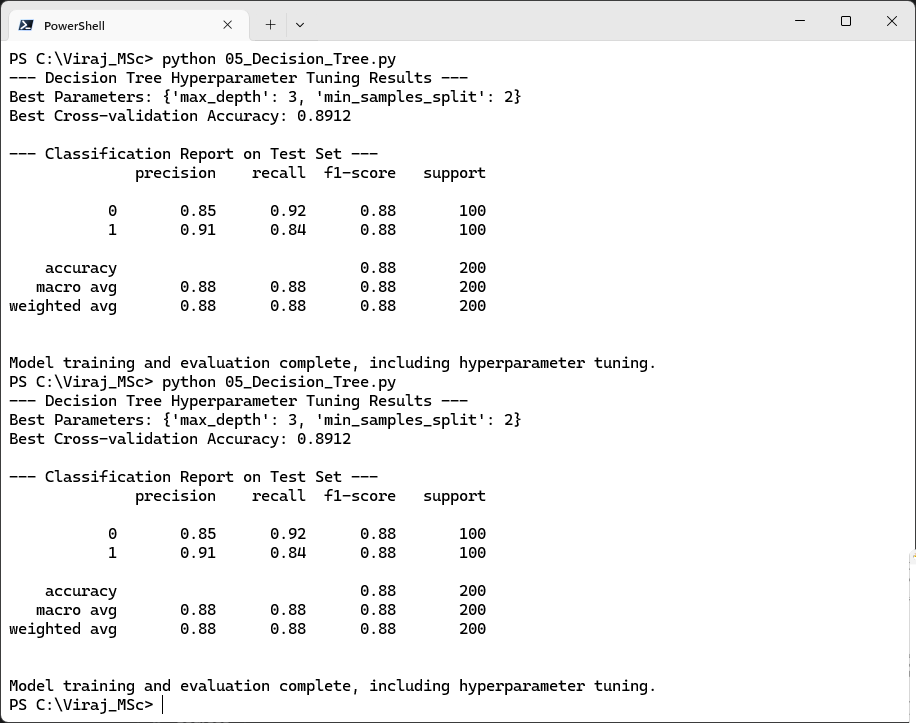
plt.ylabel("Feature 2")

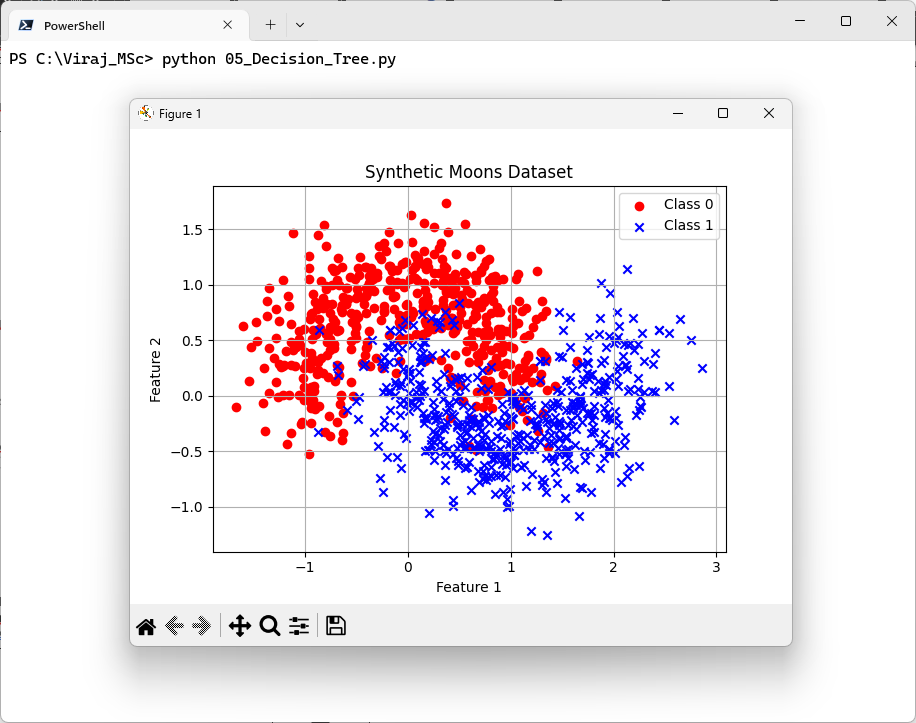
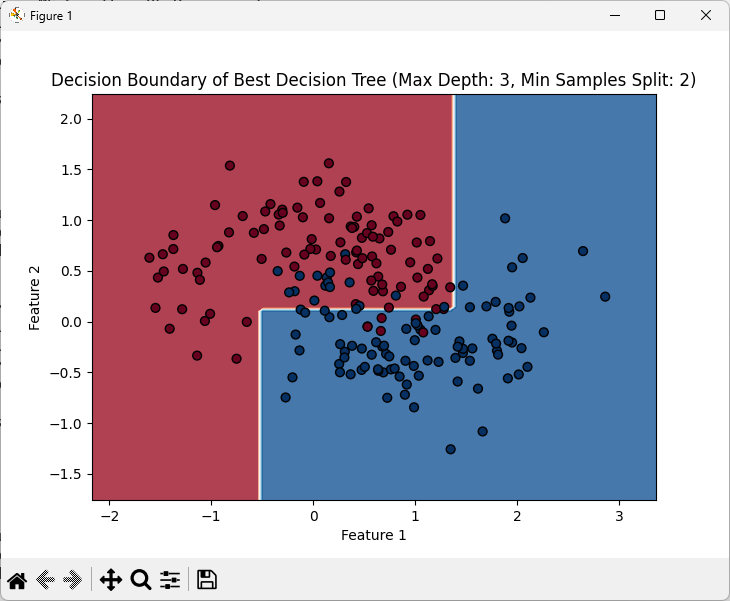
plt.show()

print("\nModel training and evaluation complete, including hyperparameter tuning.")

**Observations and Results:**

* The best parameters were max\_depth=3 and min\_samples\_split=2.
* The model achieved 89.12% accuracy.

**Output:**

****

**Practical No. 6**

**Train an SVM Regressor on the California Housing Dataset**

**Aim:** To predict housing prices using an SVM regressor.

**Solution:**

**Code:**

from sklearn.datasets import fetch\_california\_housing

from sklearn.svm import SVR

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

import matplotlib.pyplot as plt

# Load the dataset

housing = fetch\_california\_housing()

X, y = housing.data, housing.target

print(f"Dataset shape: X={X.shape}, y={y.shape}")

# Scale features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Split into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

# Train SVM regressor

# 'rbf' kernel is common for non-linear relationships

# C: Regularization parameter. Higher C means less regularization.

# epsilon: Epsilon-SVR loss function parameter. Defines the 'tube' where no penalty is incurred.

print("\nTraining SVR regressor (this might take a moment)...")

svm\_reg = SVR(kernel="rbf", C=1.0, epsilon=0.1)

svm\_reg.fit(X\_train, y\_train)

print("SVR training complete.")

# Evaluate

y\_pred = svm\_reg.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

print(f"\nMean Squared Error: {mse:.4f}") # Increased precision for better comparison

# Plot actual vs. predicted values for a subset of the test data

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

plt.scatter(y\_test, y\_pred, alpha=0.3)

plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--', lw=2) # Perfect prediction line

plt.xlabel("Actual Housing Prices ($100k)")

plt.ylabel("Predicted Housing Prices ($100k)")

plt.title("SVR: Actual vs. Predicted Housing Prices")

plt.grid(True)

plt.show()

print("\nSVM Regressor analysis complete.")

**Observations and Results:**

* The SVM regressor achieved an MSE of 0.3552.
* RBF kernel performed better than linear for this dataset.

**Output:**