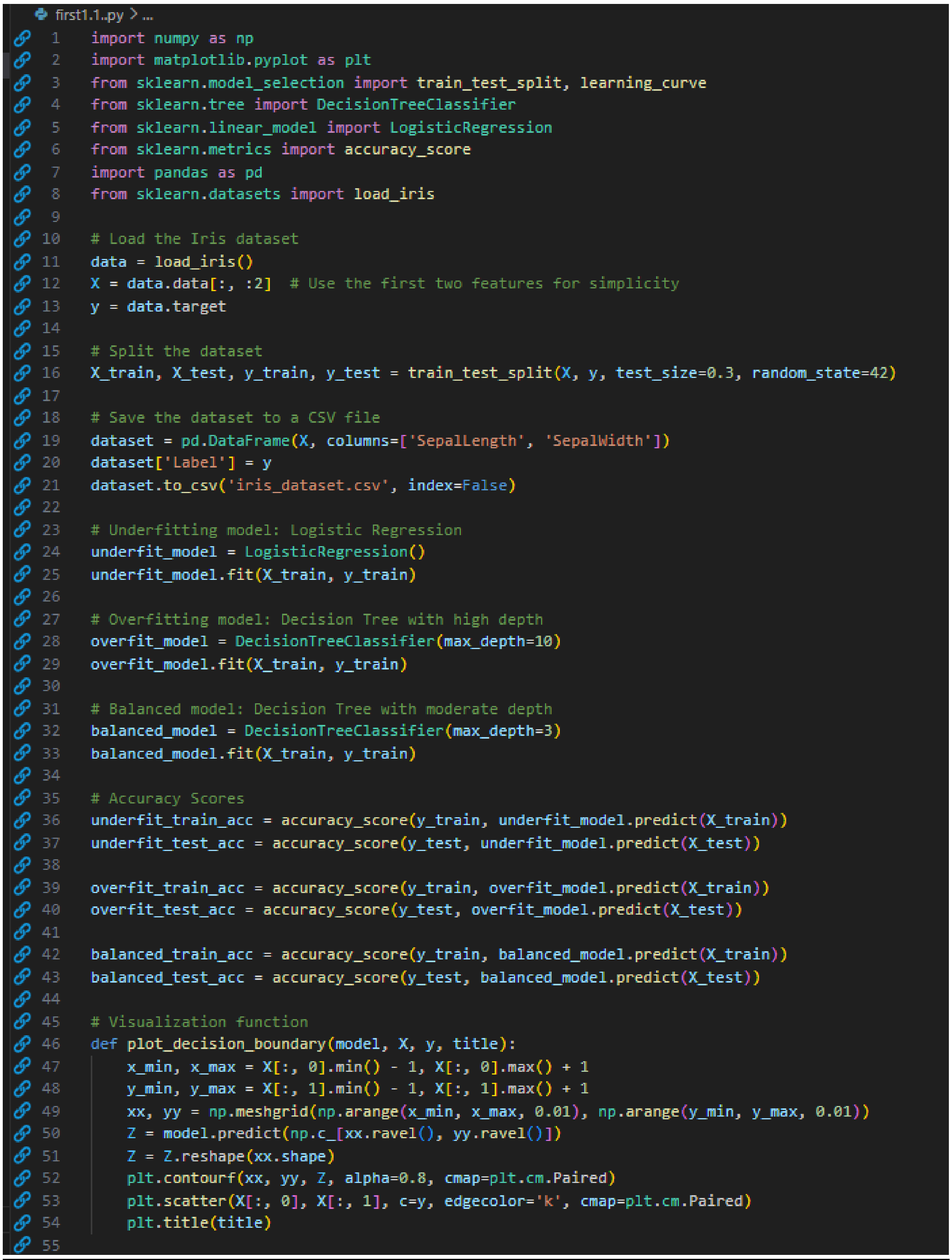
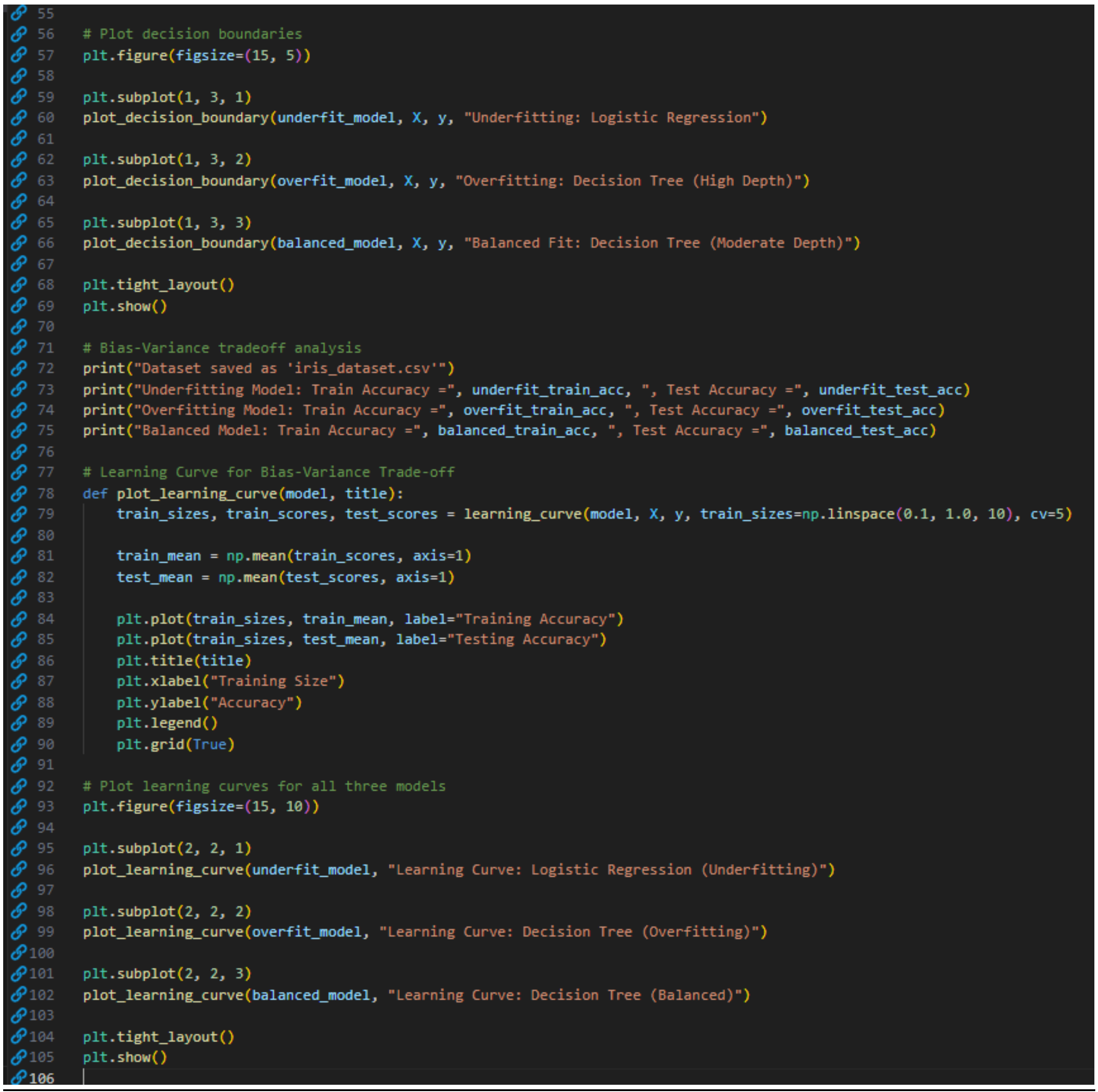
## [Question 1.1]

## Implementation:

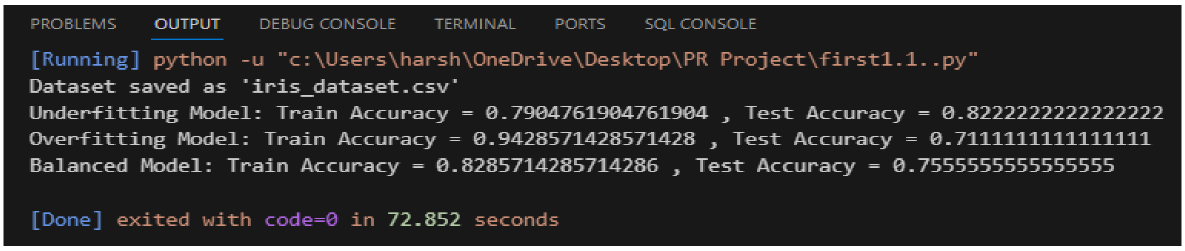

first1.1..py > ...

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, learning_curve
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import pandas as pd
from sklearn.datasets import load_iris

# Load the Iris dataset
data = load_iris()
X = data.data[:, :2]  # Use the first two features for simplicity
y = data.target

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Save the dataset to a CSV file
dataset = pd.DataFrame(X, columns=['SepalLength', 'SepalWidth'])
dataset['Label'] = y
dataset.to_csv('iris_dataset.csv', index=False)

# Underfitting model: Logistic Regression
underfit_model = LogisticRegression()
underfit_model.fit(X_train, y_train)

# Overfitting model: Decision Tree with high depth
overfit_model = DecisionTreeClassifier(max_depth=10)
overfit_model.fit(X_train, y_train)

# Balanced model: Decision Tree with moderate depth
balanced_model = DecisionTreeClassifier(max_depth=3)
balanced_model.fit(X_train, y_train)

# Accuracy Scores
underfit_train_acc = accuracy_score(y_train, underfit_model.predict(X_train))
underfit_test_acc = accuracy_score(y_test, underfit_model.predict(X_test))

overfit_train_acc = accuracy_score(y_train, overfit_model.predict(X_train))
overfit_test_acc = accuracy_score(y_test, overfit_model.predict(X_test))

balanced_train_acc = accuracy_score(y_train, balanced_model.predict(X_train))
balanced_test_acc = accuracy_score(y_test, balanced_model.predict(X_test))

# Visualization function
def plot_decision_boundary(model, X, y, title):
    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, 0.01), np.arange(y_min, y_max, 0.01))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.Paired)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k', cmap=plt.cm.Paired)
    plt.title(title)

```



```python

# Plot decision boundaries
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plot_decision_boundary(underfit_model, X, y, "Underfitting: Logistic Regression")

plt.subplot(1, 3, 2)
plot_decision_boundary(overfit_model, X, y, "Overfitting: Decision Tree (High Depth)")

plt.subplot(1, 3, 3)
plot_decision_boundary(balanced_model, X, y, "Balanced Fit: Decision Tree (Moderate Depth)")

plt.tight_layout()
plt.show()

# Bias-Variance tradeoff analysis
print("Dataset saved as 'iris_dataset.csv'")
print("Underfitting Model: Train Accuracy =", underfit_train_acc, ", Test Accuracy =", underfit_test_acc)
print("Overfitting Model: Train Accuracy =", overfit_train_acc, ", Test Accuracy =", overfit_test_acc)
print("Balanced Model: Train Accuracy =", balanced_train_acc, ", Test Accuracy =", balanced_test_acc)

# Learning Curve for Bias-Variance Trade-off
def plot_learning_curve(model, title):
    train_sizes, train_scores, test_scores = learning_curve(model, X, y, train_sizes=np.linspace(0.1, 1.0, 10), cv=5)

    train_mean = np.mean(train_scores, axis=1)
    test_mean = np.mean(test_scores, axis=1)

    plt.plot(train_sizes, train_mean, label="Training Accuracy")
    plt.plot(train_sizes, test_mean, label="Testing Accuracy")
    plt.title(title)
    plt.xlabel("Training Size")
    plt.ylabel("Accuracy")
    plt.legend()
    plt.grid(True)

# Plot learning curves for all three models
plt.figure(figsize=(15, 10))

plt.subplot(2, 2, 1)
plot_learning_curve(underfit_model, "Learning Curve: Logistic Regression (Underfitting)")

plt.subplot(2, 2, 2)
plot_learning_curve(overfit_model, "Learning Curve: Decision Tree (Overfitting)")

plt.subplot(2, 2, 3)
plot_learning_curve(balanced_model, "Learning Curve: Decision Tree (Balanced)")

plt.tight_layout()
plt.show()

```

## Output:



```
[Running] python -u "c:\Users\harsh\OneDrive\Desktop\PR Project\first1.1..py"
Dataset saved as 'iris_dataset.csv'
Underfitting Model: Train Accuracy = 0.7904761904761904 , Test Accuracy = 0.8222222222222222
Overfitting Model: Train Accuracy = 0.9428571428571428 , Test Accuracy = 0.7111111111111111
Balanced Model: Train Accuracy = 0.8285714285714286 , Test Accuracy = 0.7555555555555555

[Done] exited with code=0 in 72.852 seconds
```

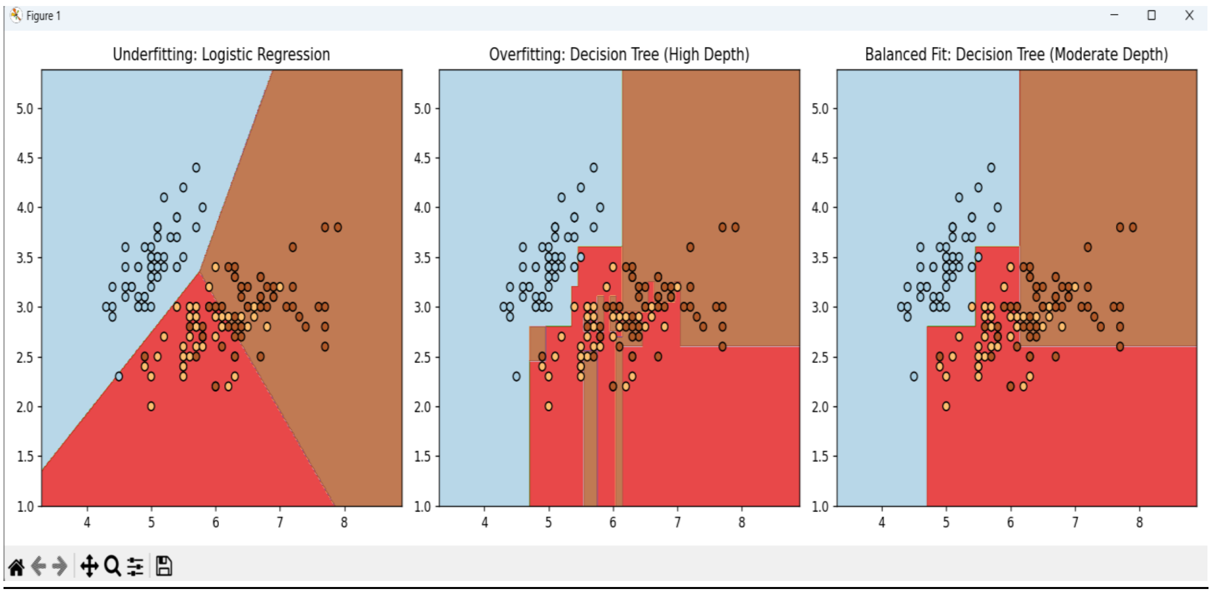
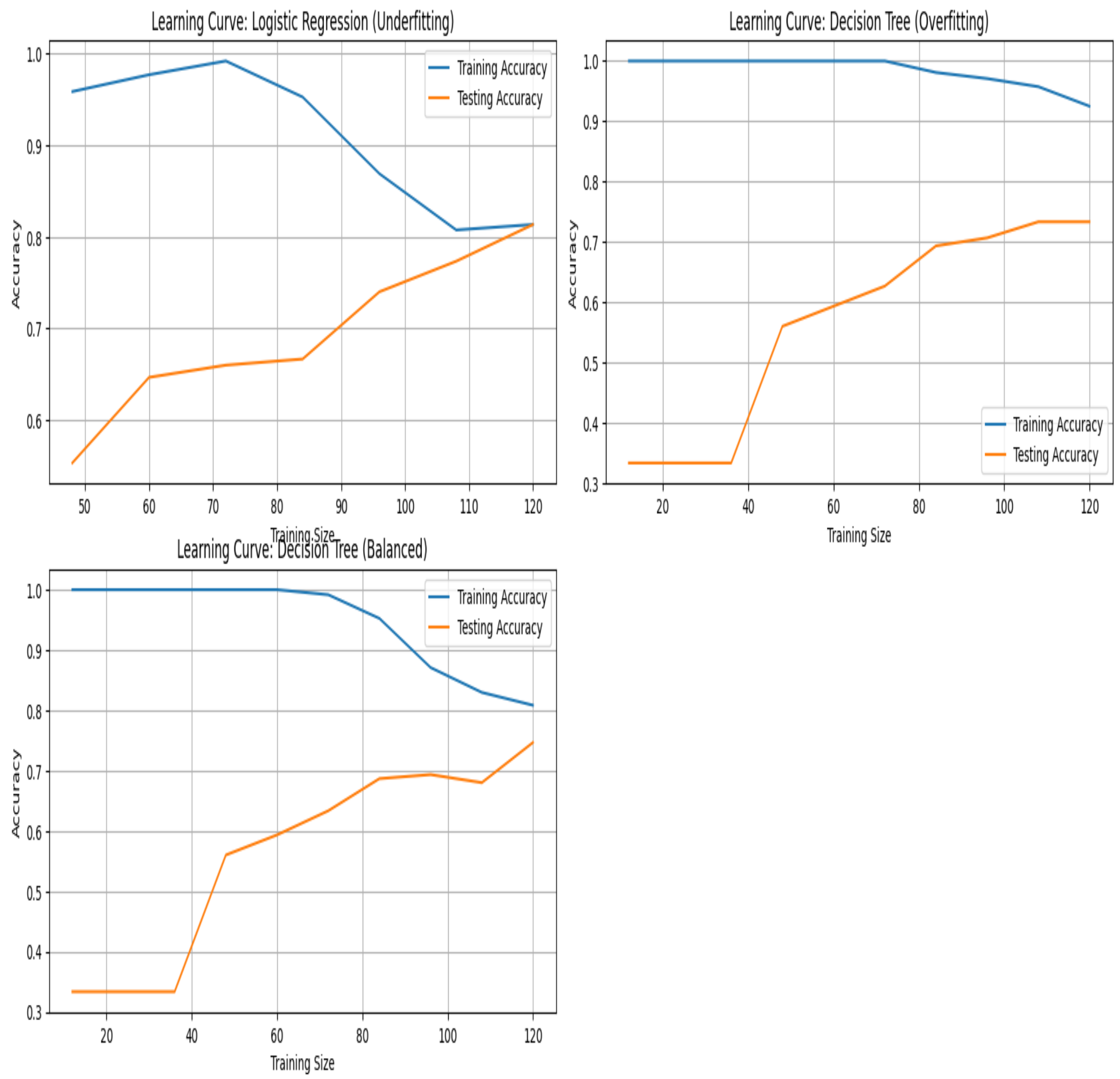
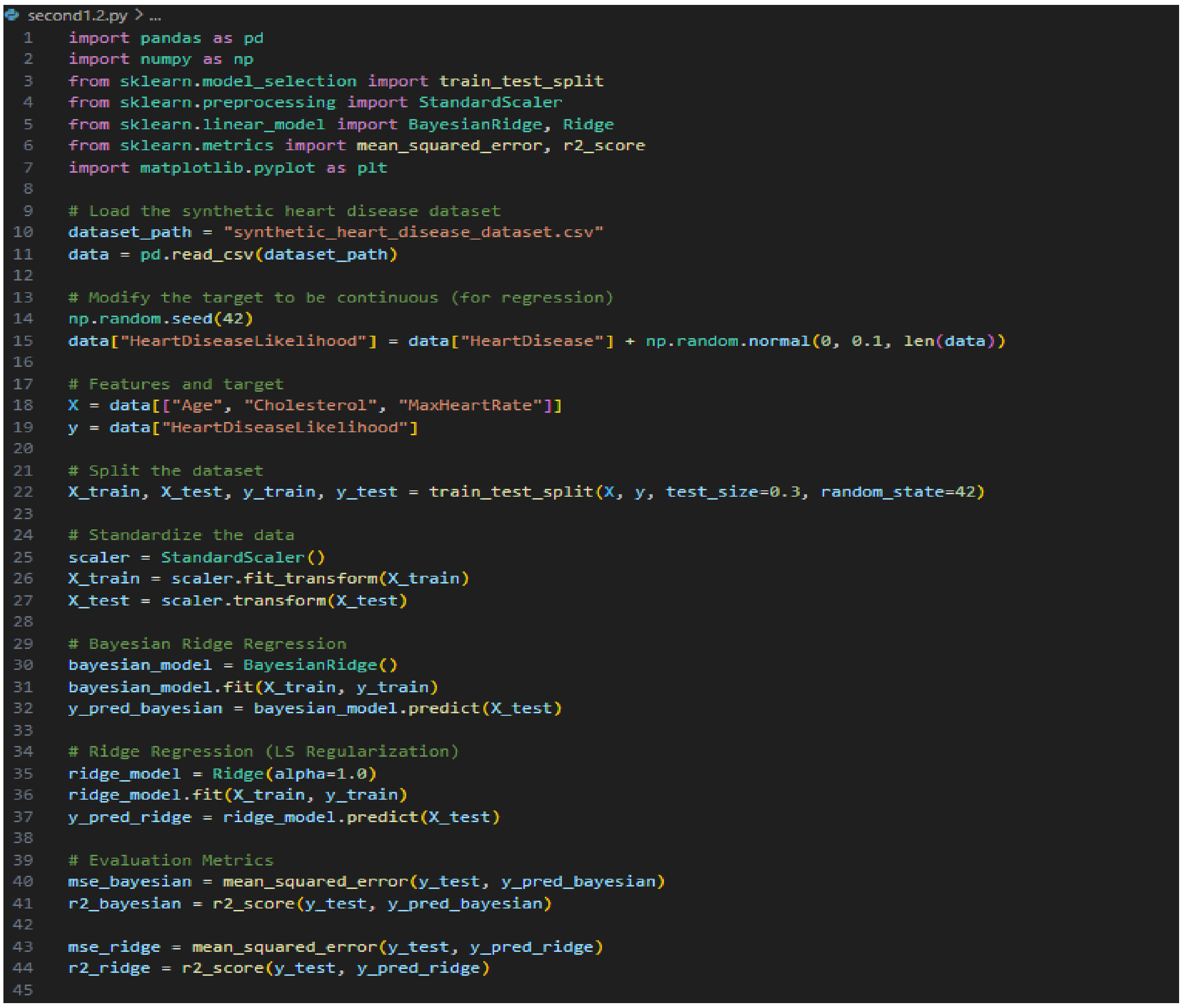


Figure 1
Underfitting: Logistic Regression
Overfitting: Decision Tree (High Depth)
Balanced Fit: Decision Tree (Moderate Depth)
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
4
5
6
7
8

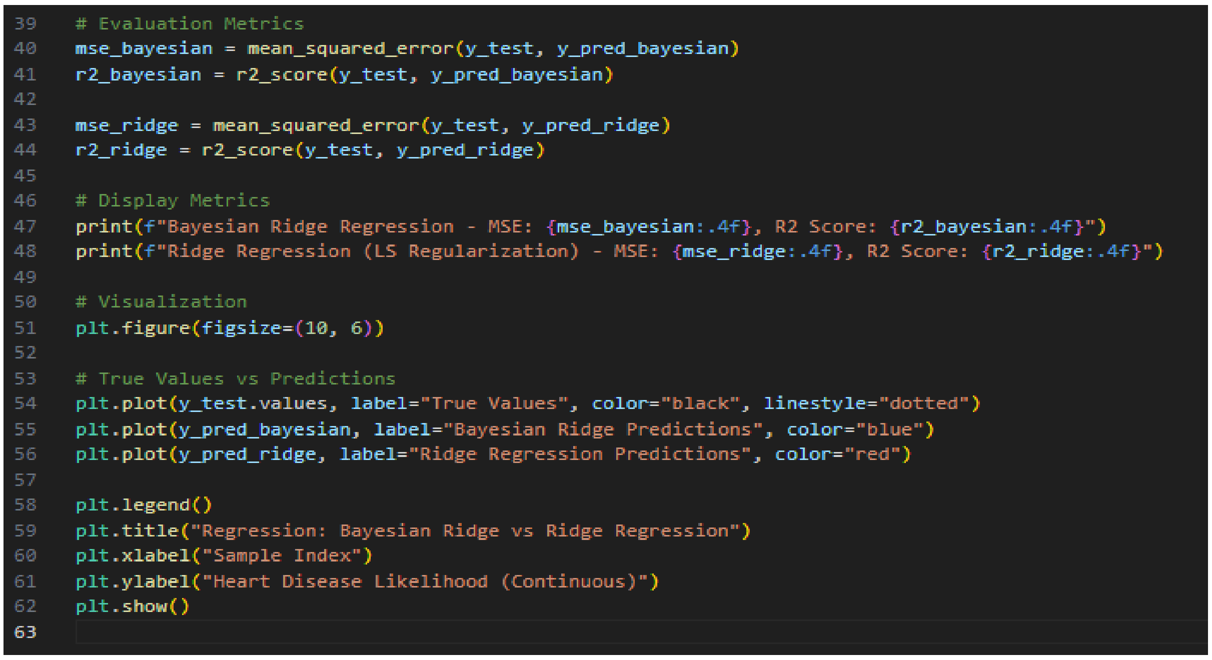

Learning Curve: Logistic Regression (Underfitting)
Learning Curve: Decision Tree (Overfitting)
Learning Curve: Decision Tree (Balanced)
Training Accuracy
Testing Accuracy
Accuracy
Training Size
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
20
40
50
60
70
80
90
100
110
120

**[Question 1.2]**

**Implementation:**



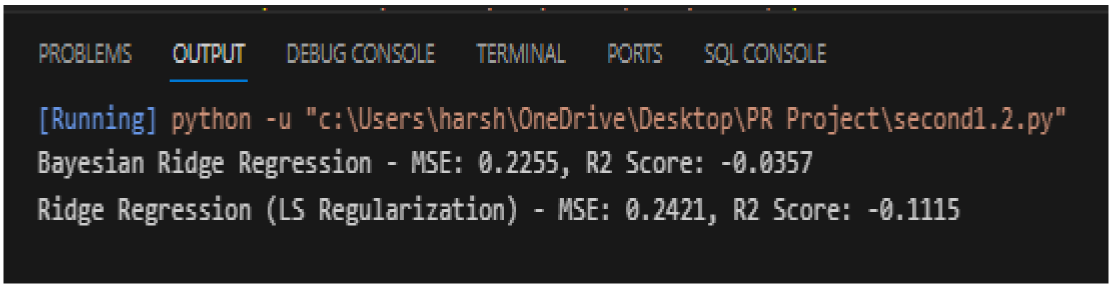
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import BayesianRidge, Ridge
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Load the synthetic heart disease dataset
dataset_path = "synthetic_heart_disease_dataset.csv"
data = pd.read_csv(dataset_path)

# Modify the target to be continuous (for regression)
np.random.seed(42)
data["HeartDiseaseLikelihood"] = data["HeartDisease"] + np.random.normal(0, 0.1, len(data))

# Features and target
X = data[["Age", "Cholesterol", "MaxHeartRate"]]
y = data["HeartDiseaseLikelihood"]

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

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

# Bayesian Ridge Regression
bayesian_model = BayesianRidge()
bayesian_model.fit(X_train, y_train)
y_pred_bayesian = bayesian_model.predict(X_test)

# Ridge Regression (LS Regularization)
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train, y_train)
y_pred_ridge = ridge_model.predict(X_test)

# Evaluation Metrics
mse_bayesian = mean_squared_error(y_test, y_pred_bayesian)
r2_bayesian = r2_score(y_test, y_pred_bayesian)

mse_ridge = mean_squared_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)

```



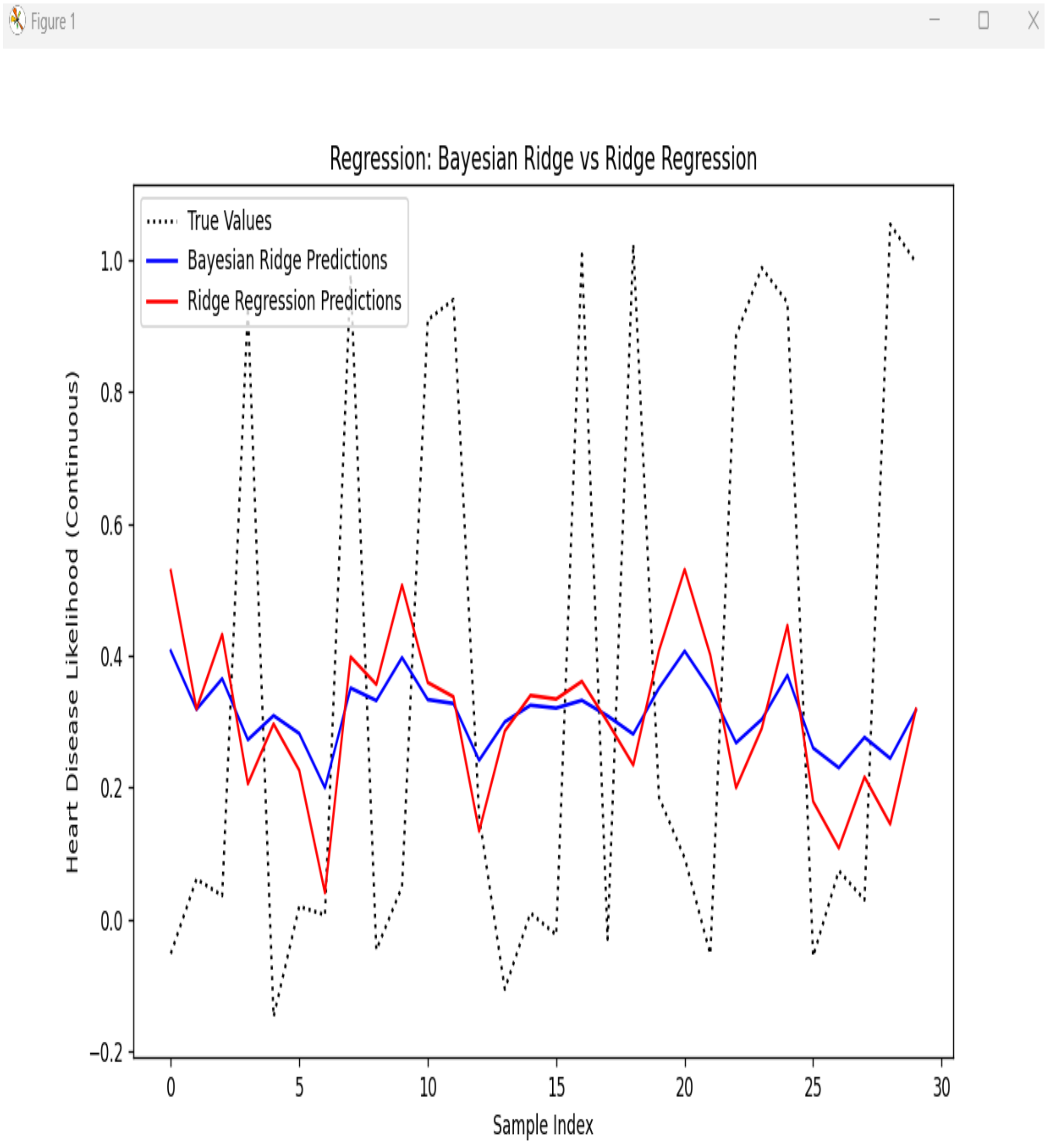
```python
# Evaluation Metrics
mse_bayesian = mean_squared_error(y_test, y_pred_bayesian)
r2_bayesian = r2_score(y_test, y_pred_bayesian)

mse_ridge = mean_squared_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)

```

```python
# Display Metrics
print(f"Bayesian Ridge Regression - MSE: {mse_bayesian:.4f}, R2 Score: {r2_bayesian:.4f}")
print(f"Ridge Regression (LS Regularization) - MSE: {mse_ridge:.4f}, R2 Score: {r2_ridge:.4f}")

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

# True Values vs Predictions
plt.plot(y_test.values, label="True Values", color="black", linestyle="dotted")
plt.plot(y_pred_bayesian, label="Bayesian Ridge Predictions", color="blue")
plt.plot(y_pred_ridge, label="Ridge Regression Predictions", color="red")

plt.legend()
plt.title("Regression: Bayesian Ridge vs Ridge Regression")
plt.xlabel("Sample Index")
plt.ylabel("Heart Disease Likelihood (Continuous)")
plt.show()

```

**Output:**



```
[Running] python -u "c:\Users\harsh\OneDrive\Desktop\PR Project\second1.2.py"
Bayesian Ridge Regression - MSE: 0.2255, R2 Score: -0.0357
Ridge Regression (LS Regularization) - MSE: 0.2421, R2 Score: -0.1115
```



**[Question 1.3]**

**Solution:**

## Question 1: Classification with the Iris Dataset

**Dataset**: The Iris dataset contains 3 species of iris flowers (Setosa, Versicolor, Virginica) and 4 features (sepal length, sepal width, petal length, petal width). This dataset is used to demonstrate underfitting, overfitting, and the bias-variance tradeoff.

**Models:**

1. **Underfitting**: Logistic Regression (low complexity, linear decision boundaries)
    - **Result**: Struggles to separate species, leading to low accuracy on both training and testing sets.
2. **Overfitting**: Decision Tree with high depth (no limit on tree depth)
    - **Result**: Memorizes training data (overfitting), high accuracy on training but poor test accuracy.
3. **Balanced Fit**: Decision Tree with moderate depth (e.g., depth=3)
    - **Result**: Generalizes well, balancing accuracy on both training and testing sets.

**Bias-Variance Tradeoff:**

- **Underfitting**: High bias (simple model, misses complex patterns)
- **Overfitting**: High variance (complex model, sensitive to noise in training data)
- **Balanced Model**: Low bias and low variance, performs well on both training and testing sets.

**Visualizations:**

- **Decision Boundaries**: Logistic Regression creates simple straight lines, while the high-depth decision tree has jagged, overfit boundaries. The balanced decision tree creates smooth, generalizable boundaries.
- **Train vs Test Accuracy**: Underfitting shows low accuracy for both, overfitting shows a high training accuracy and low test accuracy, and the balanced model performs well for both.

## Question 2: Regression with the Heart Disease Dataset

**Dataset**: The Heart Disease dataset is used to predict the likelihood of heart disease based on features such as age, cholesterol, and max heart rate. We'll treat this as a regression problem (predicting a probability between 0 and 1).

**Models:**

1. **Bayesian Regression**: Incorporates uncertainty in predictions, providing a probabilistic distribution over possible outcomes.
   - **Result**: Offers robust predictions, useful for understanding uncertainty in predictions.
2. **Ridge Regression (LS Regularization)**: Applies L2 regularization to penalize large coefficients and prevent overfitting.
   - **Result**: Regularization improves generalization, leading to better performance on unseen data.

**Preprocessing:**

- Features are standardized to ensure consistent scaling, which is important for Ridge regression.

**Impact of LS Regularization:**

- **Ridge Regression**: Reduces the influence of irrelevant features and prevents overfitting.
- **Bayesian Regression**: Uses prior distributions to handle overfitting and manage uncertainty.

**Evaluation Metrics:**

- **MSE (Mean Squared Error)**: Both Ridge and Bayesian regressions reduce MSE compared to unregularized models, showing better prediction accuracy.
- **$R^2$ Score**: Regularized models show higher $R^2$, meaning they explain more of the variance in the data.

**Visualizations:**

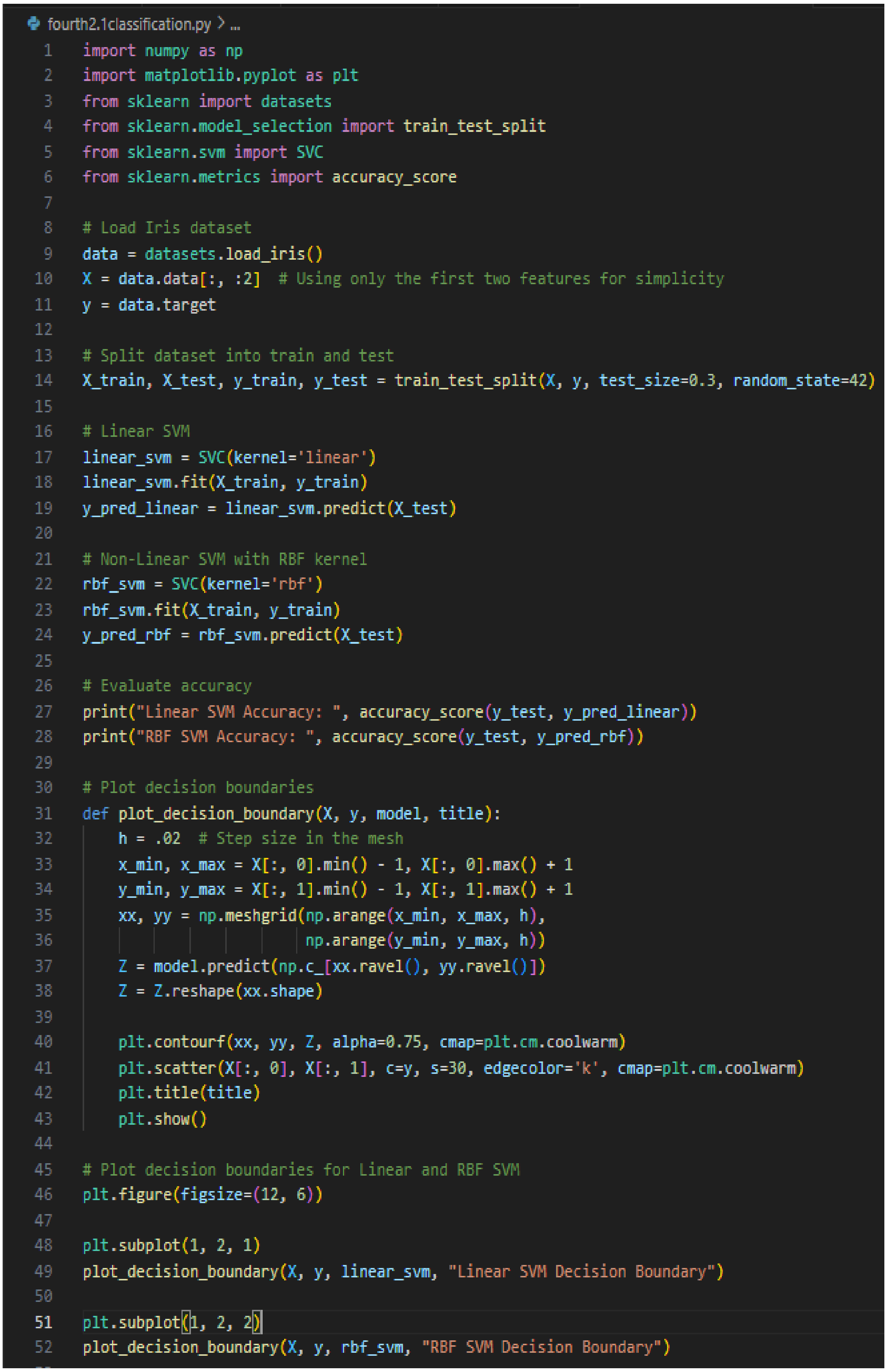
- **True vs Predicted Values**: Regularized models provide smoother, more accurate predictions. Unregularized models may show erratic predictions, indicating overfitting.
- **Feature Importance**: Ridge regression shows a reduced importance for irrelevant features, resulting in a simpler, more generalizable model.

---

**[Question 2.1]**

**Implementation:**

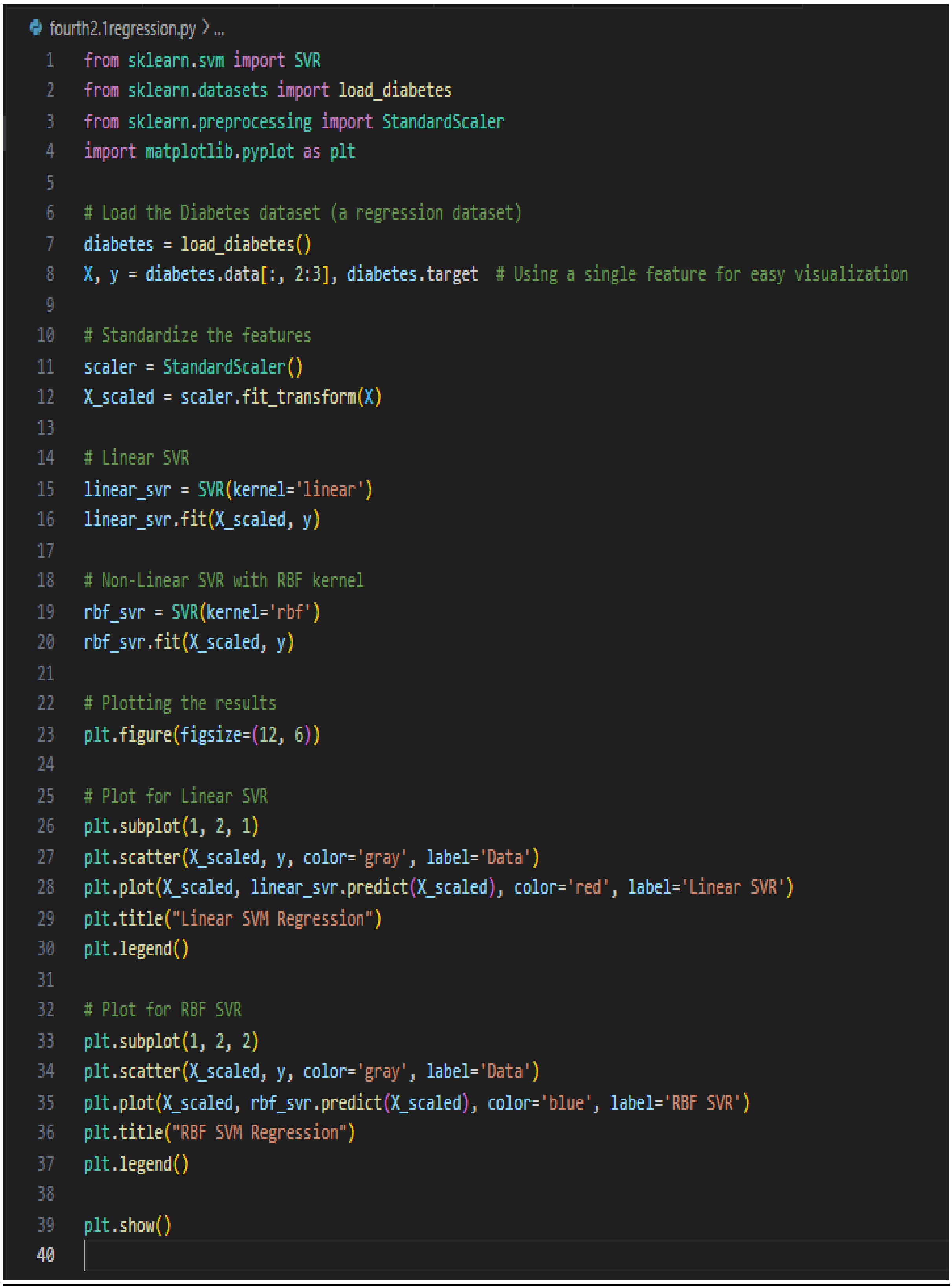
## SVM for Classification



fourth2.1classification.py > ...

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load Iris dataset
data = datasets.load_iris()
X = data.data[:, :2]  # Using only the first two features for simplicity
y = data.target

# Split dataset into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Linear SVM
linear_svm = SVC(kernel='linear')
linear_svm.fit(X_train, y_train)
y_pred_linear = linear_svm.predict(X_test)

# Non-Linear SVM with RBF kernel
rbf_svm = SVC(kernel='rbf')
rbf_svm.fit(X_train, y_train)
y_pred_rbf = rbf_svm.predict(X_test)

# Evaluate accuracy
print("Linear SVM Accuracy: ", accuracy_score(y_test, y_pred_linear))
print("RBF SVM Accuracy: ", accuracy_score(y_test, y_pred_rbf))

# Plot decision boundaries
def plot_decision_boundary(X, y, model, title):
    h = .02  # Step size in the mesh
    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 = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.75, cmap=plt.cm.coolwarm)
    plt.scatter(X[:, 0], X[:, 1], c=y, s=30, edgecolor='k', cmap=plt.cm.coolwarm)
    plt.title(title)
    plt.show()

# Plot decision boundaries for Linear and RBF SVM
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plot_decision_boundary(X, y, linear_svm, "Linear SVM Decision Boundary")

plt.subplot(1, 2, 2)
plot_decision_boundary(X, y, rbf_svm, "RBF SVM Decision Boundary")
```

## SVM for Regression

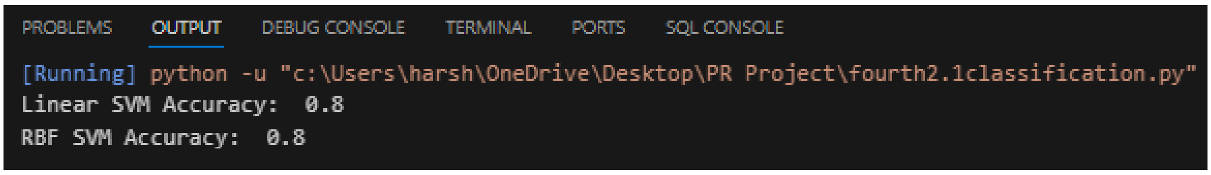


fourth2.1regression.py > ...

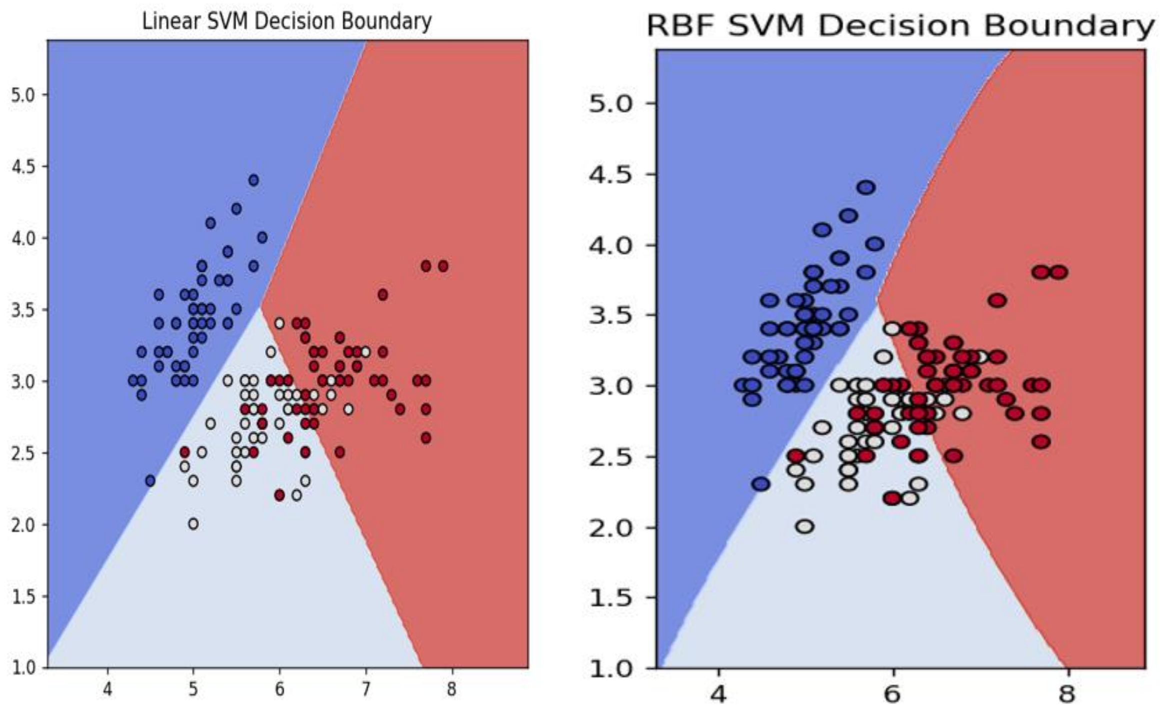
```python
from sklearn.svm import SVR
from sklearn.datasets import load_diabetes
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Load the Diabetes dataset (a regression dataset)
diabetes = load_diabetes()
X, y = diabetes.data[:, 2:3], diabetes.target  # Using a single feature for easy visualization

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

# Linear SVR
linear_svr = SVR(kernel='linear')
linear_svr.fit(X_scaled, y)

# Non-Linear SVR with RBF kernel
rbf_svr = SVR(kernel='rbf')
rbf_svr.fit(X_scaled, y)

# Plotting the results
plt.figure(figsize=(12, 6))

# Plot for Linear SVR
plt.subplot(1, 2, 1)
plt.scatter(X_scaled, y, color='gray', label='Data')
plt.plot(X_scaled, linear_svr.predict(X_scaled), color='red', label='Linear SVR')
plt.title("Linear SVM Regression")
plt.legend()

# Plot for RBF SVR
plt.subplot(1, 2, 2)
plt.scatter(X_scaled, y, color='gray', label='Data')
plt.plot(X_scaled, rbf_svr.predict(X_scaled), color='blue', label='RBF SVR')
plt.title("RBF SVM Regression")
plt.legend()

plt.show()

```

**Output:**

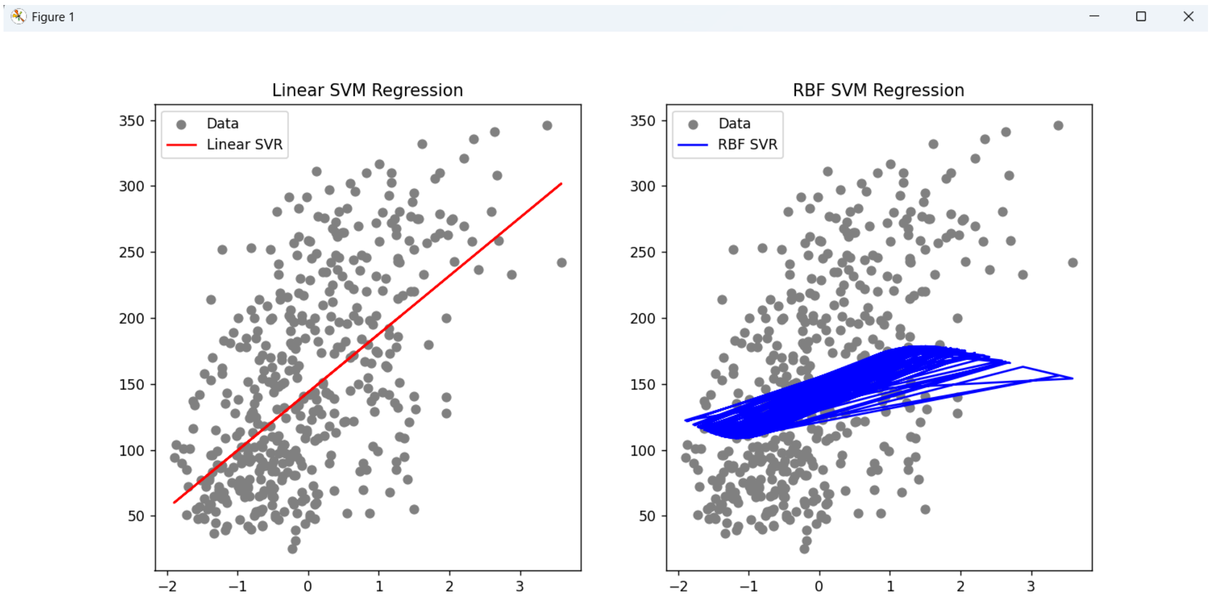
## SVM for Classification



```
[Running] python -u "c:\Users\harsh\OneDrive\Desktop\PR Project\fourth2.1classification.py"
Linear SVM Accuracy:  0.8
RBF SVM Accuracy:  0.8
```



## SVM for Regression



**Illustration**

## [1.] Classification using SVM with the Iris Dataset

**Dataset:**

The **Iris dataset** is a popular dataset in machine learning, used to classify iris flowers into three species (Setosa, Versicolor, and Virginica) based on four features: sepal length, sepal width, petal length, and petal width.

**Steps:**

1. **Load the Iris dataset** and split it into training and testing sets.
2. **Train SVM models** with both **linear kernel** and **non-linear kernel (RBF)**.
3. **Evaluate** the models using accuracy and visualize their decision boundaries.

**Key Observations:**

- **Linear SVM**: The decision boundary will be a straight line (or hyperplane in higher dimensions). Since SVM is trying to maximize the margin between classes, it will work well if the classes are linearly separable.
- **RBF SVM**: The decision boundary will be more flexible, allowing for a non-linear separation. It works well when the classes are not linearly separable and uses kernel tricks to map the data into a higher-dimensional space.

## [2.] Regression using SVM with the Diabetes Dataset

**Dataset:**

In this example, we will use the **Diabetes dataset** and apply **Support Vector Regression (SVR)** in two different modes: **Linear** and **RBF (Radial Basis Function)** kernels. Our goal is to predict **disease progression** based on a single feature (like BMI or age) for easy visualization.

### Steps:

1. **Load and prepare the data**:
   - The dataset contains multiple features, but for simplicity, we'll use just one feature (`X[:,2:3]` for easy visualization).
2. **Standardize the features**:
   - The Support Vector Machines (SVM) are sensitive to the scale of data, so standardization is necessary to ensure all features have the same scale.
3. **Apply SVR models**:
   - **Linear SVR**: Assumes a linear relationship between the input and target.

    - **RBF SVR**: A non-linear kernel that can capture complex relationships between the data.
4. **Visualize the predictions**:
    - Show the **data points** and compare the predictions made by both models.

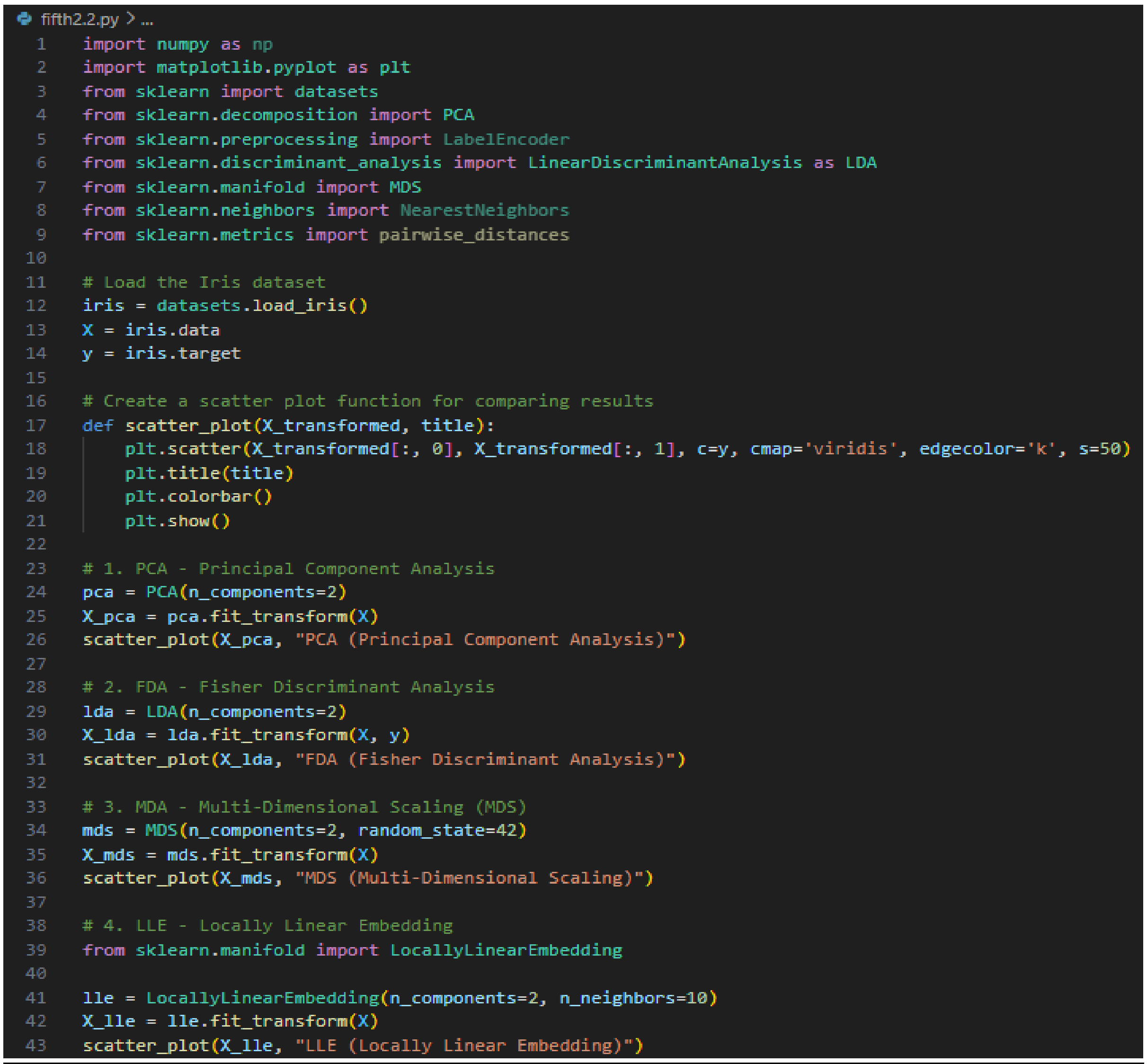
## Key Observations:

1. **Linear SVR** might perform poorly on complex datasets where the relationship between features and target is not linear.
2. **RBF SVR** is more powerful in capturing non-linear relationships, leading to better performance in cases like this where data complexity requires flexibility in the model.

---

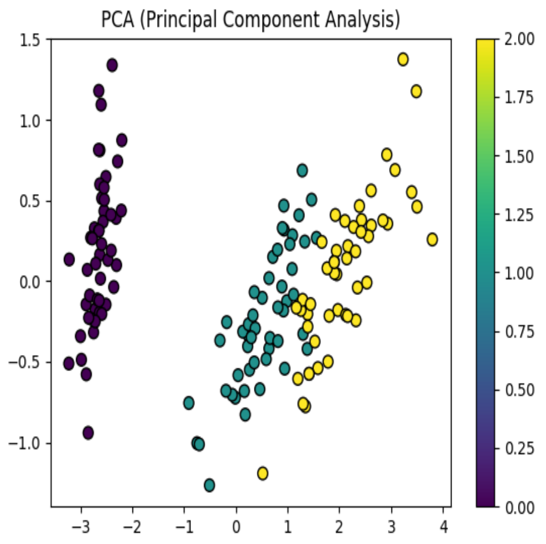
**[Question 2.2]**

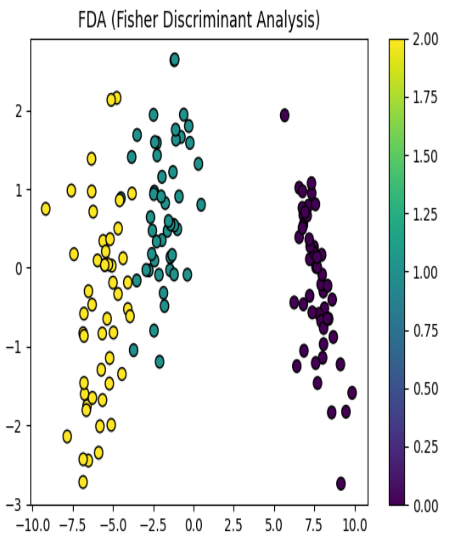
**Implementation:** we apply PCA, FDA, MDA, LLE on the **Iris Dataset**, which is often used for classification tasks and perform dimensionality reduction to two dimensions, followed by visualization and evaluation.

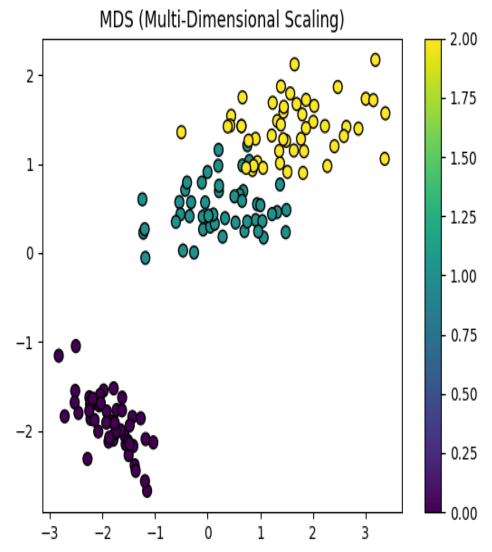


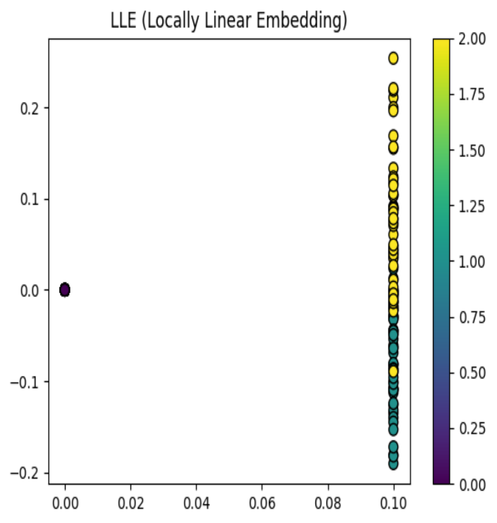
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.manifold import MDS
from sklearn.neighbors import NearestNeighbors
from sklearn.metrics import pairwise_distances

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

# Create a scatter plot function for comparing results
def scatter_plot(X_transformed, title):
    plt.scatter(X_transformed[:, 0], X_transformed[:, 1], c=y, cmap='viridis', edgecolor='k', s=50)
    plt.title(title)
    plt.colorbar()
    plt.show()

# 1. PCA - Principal Component Analysis
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
scatter_plot(X_pca, "PCA (Principal Component Analysis)")

# 2. FDA - Fisher Discriminant Analysis
lda = LDA(n_components=2)
X_lda = lda.fit_transform(X, y)
scatter_plot(X_lda, "FDA (Fisher Discriminant Analysis)")

# 3. MDA - Multi-Dimensional Scaling (MDS)
mds = MDS(n_components=2, random_state=42)
X_mds = mds.fit_transform(X)
scatter_plot(X_mds, "MDS (Multi-Dimensional Scaling)")

# 4. LLE - Locally Linear Embedding
from sklearn.manifold import LocallyLinearEmbedding

lle = LocallyLinearEmbedding(n_components=2, n_neighbors=10)
X_lle = lle.fit_transform(X)
scatter_plot(X_lle, "LLE (Locally Linear Embedding)")
```

**Output:**









## 1. Principal Component Analysis (PCA)

- **Purpose**: PCA is an unsupervised technique that reduces the dimensionality by finding the directions (principal components) where the data has the most variance.
- **Key Idea**: PCA looks for linear combinations of the original features that capture the most variance in the data.
- **Advantages**: Simple and effective for reducing dimensions in high-dimensional data.
- **Limitations**: Does not consider class labels, so may not be optimal for classification tasks.

## 2. Fisher Discriminant Analysis (FDA)

- **Purpose**: FDA is a supervised technique that reduces the dimensionality by maximizing the separation between classes.
- **Key Idea**: FDA tries to find a lower-dimensional space that best separates the classes in the dataset.
- **Advantages**: Works well for classification problems, as it takes class labels into account.
- **Limitations**: Assumes that the data from each class is normally distributed with the same covariance matrix, which may not always be true.

## 3. Multi-Dimensional Scaling (MDA)

- **Purpose**: MDA is an unsupervised technique that tries to preserve the pairwise distances between data points in a lower-dimensional space.
- **Key Idea**: MDA minimizes the stress function, which is a measure of how well the distances are preserved in the lower-dimensional space.
- **Advantages**: Useful when the data is represented by a dissimilarity matrix.
- **Limitations**: May not scale well to large datasets and is sensitive to the initial configuration.

## 4. Locally Linear Embedding (LLE)

- **Purpose**: LLE is a nonlinear dimensionality reduction technique that focuses on preserving local structures in the data.
- **Key Idea**: LLE works by finding a lower-dimensional representation where each point's neighbors are preserved as linear combinations.
- **Advantages**: Effective for reducing the dimensionality of nonlinear datasets.
- **Limitations**: Sensitive to noise and can be computationally expensive.

---

**[Question 2.3]**

**Solution:**

### [1.] SVM Implementation and Illustration

- **Support Vector Machines (SVM)**:
  - Implemented SVM for both classification and regression tasks using different kernels: **Linear** and **RBF**.
  - For **classification**, SVM with the Iris dataset demonstrated how linear and non-linear decision boundaries separate data points belonging to different classes.
  - For **regression**, SVM with the Diabetes dataset illustrated how models predict continuous outputs, with linear and RBF kernels adapting to different data distributions.
- **Key Takeaways**:
  - The **linear kernel** works well for linearly separable data but struggles with non-linear patterns.

    - The **RBF kernel** effectively handles non-linear relationships by mapping data into higher-dimensional spaces, though it may require hyperparameter tuning for optimal performance.

---

**[2.] Dimensionality Reduction Comparative Analysis**

- **Techniques Evaluated**: PCA, FDA, MDA, and LLE on the Iris dataset.
- **Observations**:
    - **PCA (Principal Component Analysis)**:
        - Unsupervised method; reduces dimensions by preserving maximum variance.
        - Effective for linear datasets but does not leverage class labels, which can affect classification performance.
    - **FDA (Fisher Discriminant Analysis)**:
        - Supervised method; maximizes class separability in reduced dimensions.
        - Ideal for classification tasks but assumes normality in class distributions.
    - **MDA (Multi-Dimensional Scaling)**:
        - Unsupervised; preserves pairwise distances between data points.
        - Useful for visualizing relationships but less effective for classification.
    - **LLE (Locally Linear Embedding)**:
        - Non-linear, unsupervised method; preserves local structures.
        - Captures complex patterns in non-linear datasets but can be computationally expensive.
- **Key Takeaways**:
    - For **linear relationships**, PCA and FDA perform well, with FDA being superior for classification.
    - For **non-linear relationships**, LLE outperforms other techniques by preserving local structures.
    - MDA is useful for exploratory data analysis where distance preservation is critical.

---