RECEP TAYYIP ERDOGAN UNIVERSITY

FACULTY OF ENGINEERING AND ARCHITECTURE

COMPUTER ENGINEERING DEPARTMENT

CEN 411 Data Mining

2024-2025 Fall

Due Date: 08.12.2024

Student Name Surname

Kevser SEVİM

Student No

201401010

Instructor

Asst. Prof. Dr. Abdulgani KAHRAMAN

RIZE

Question 1: 'Heart Failure Clinical Records', Question 2: Iris dataset, Question 3: Auto MPG Dataset

Question 1: 'Heart Failure Clinical Records'

Introduction

This report evaluates a publicly available dataset using a Decision Tree classifier and compares its performance against a Random Forest classifier. The goal is to predict the target variable, evaluate the model's performance using accuracy, precision, recall, and F1-score, and analyze the differences between classifiers.

Reason for Selecting the Dataset:

https://www.kaggle.com/datasets/andrewmvd/heart-failure-clinical-data

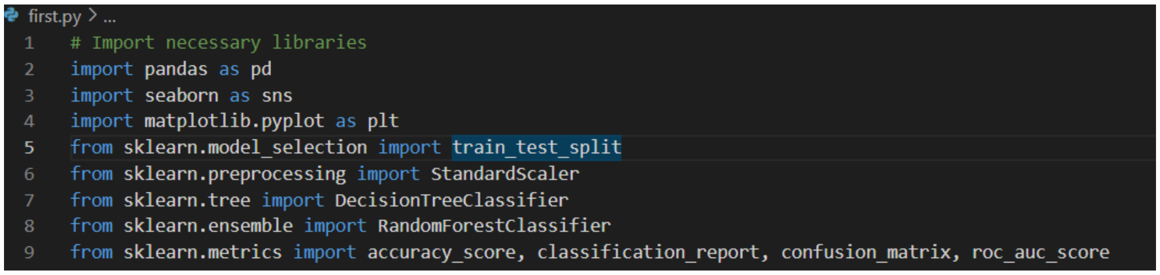
For this assignment, the 'Heart Failure Clinical Records' dataset was selected. This dataset includes medical measurements and mortality outcomes (DEATH_EVENT) of patients with heart failure. It is highly significant for developing decision support systems in the healthcare field. The dataset was chosen for the following reasons:

Relevance: Heart failure is a major public health issue, and this dataset can contribute to studies in health analytics.
Rich Features: It includes important clinical variables such as age, ejection fraction, and serum creatinine.
Balance and Predictability: Despite the imbalanced distribution of the target variable (DEATH_EVENT), it contains patterns that can be predicted using machine learning models.

Detailed Code Descriptions and Outputs

Importing Libraries

Code:



```python
# Import necessary libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_auc_score
```

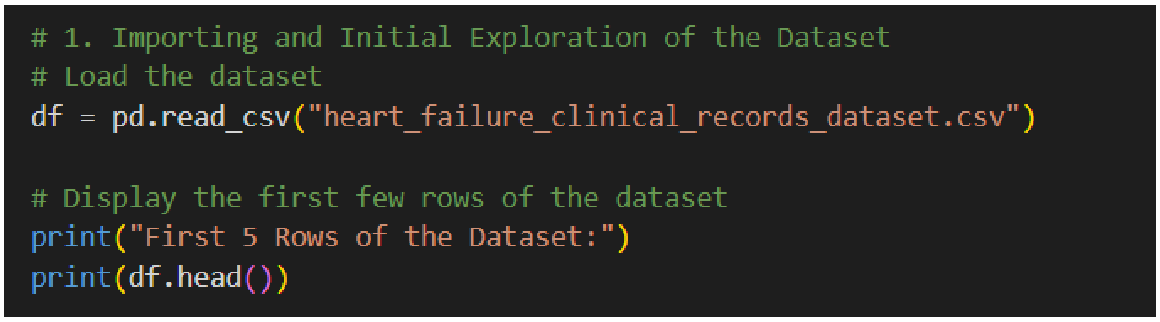
These libraries were used in the processes of data loading, processing, visualization, and building machine learning models:

- **pandas**: Used for data manipulation and analysis.
- **seaborn**: Provides advanced data visualization capabilities.
- **matplotlib.pyplot**: A basic tool for creating visualizations.
- **sklearn.model_selection**: Used for splitting data into training and testing sets.
- **sklearn.preprocessing**: Used for scaling features.
- **sklearn.tree** and **sklearn.ensemble**: Used to build classification models.
- **sklearn.metrics**: Provides metrics for evaluating model performance.

Loading and Initial Exploration of the Dataset

The dataset was loaded using the pandas library. The first few rows were displayed to understand the overall structure of the dataset. Additionally, the data types of the columns, missing values, and a statistical summary were reviewed.

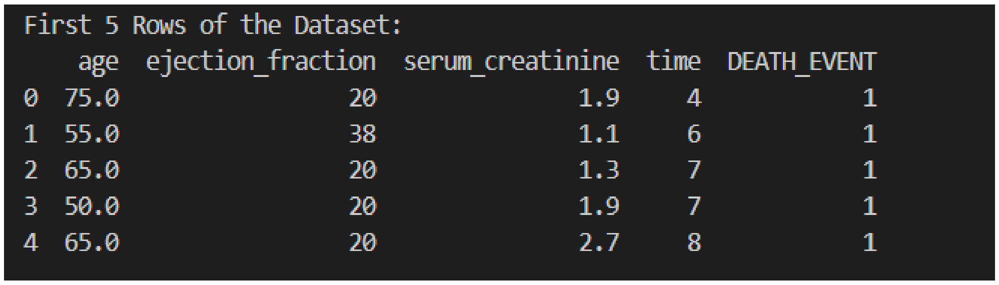
Code:



```python
# 1. Importing and Initial Exploration of the Dataset
# Load the dataset
df = pd.read_csv("heart_failure_clinical_records_dataset.csv")

# Display the first few rows of the dataset
print("First 5 Rows of the Dataset:")
print(df.head())
```

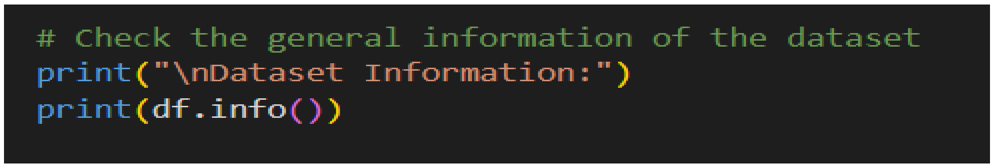
Output:

First five rows of the dataset and general information:



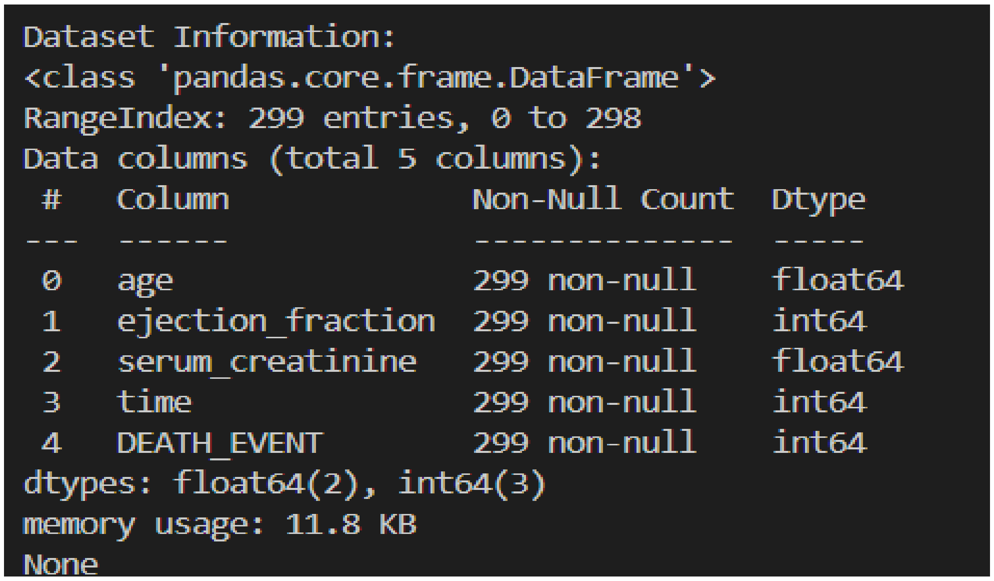
```
First 5 Rows of the Dataset:
    age  ejection_fraction  serum_creatinine  time  DEATH_EVENT
0  75.0                 20               1.9     4            1
1  55.0                 38               1.1     6            1
2  65.0                 20               1.3     7            1
3  50.0                 20               1.9     7            1
4  65.0                 20               2.7     8            1
```

Code:



```
# Check the general information of the dataset
print("\nDataset Information:")
print(df.info())
```

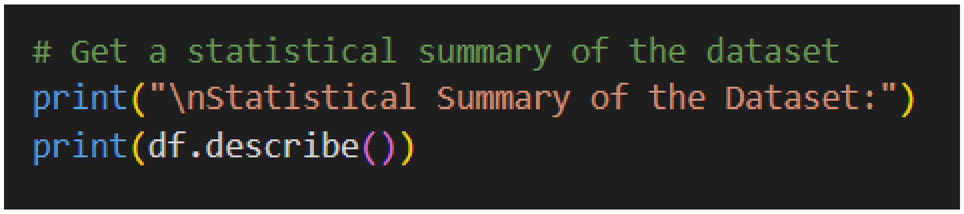
Output:



```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   age                299 non-null    float64
 1   ejection_fraction  299 non-null    int64  
 2   serum_creatinine   299 non-null    float64
 3   time               299 non-null    int64  
 4   DEATH_EVENT        299 non-null    int64  
dtypes: float64(2), int64(3)
memory usage: 11.8 KB
None
```

The df.info() function provides a general summary of the dataset. With df.info(), we can obtain information about column names, data types, missing values, and the total number of observations. This function helps us quickly understand the structure of the dataset. It allows us to check for missing data, distinguish between numerical and categorical variables, and determine which columns to work with for subsequent processes.

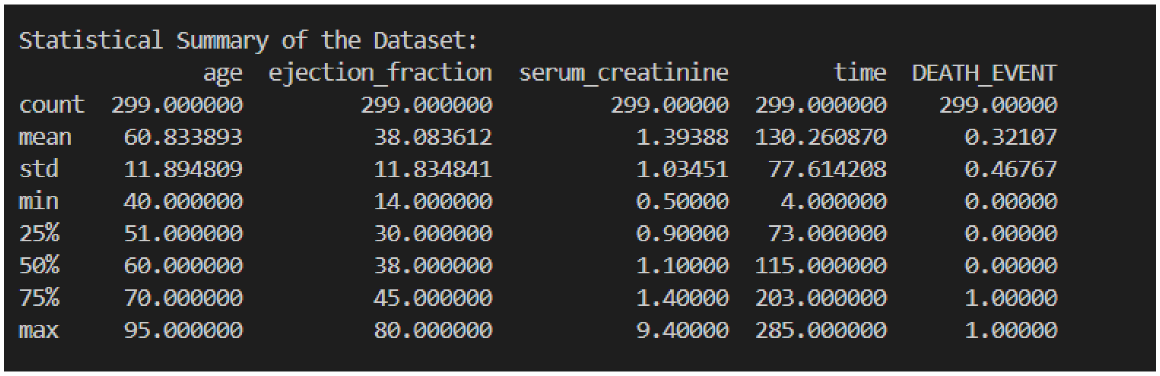
Code:



```
# Get a statistical summary of the dataset
print("\nStatistical Summary of the Dataset:")
print(df.describe())
```

The **df.describe()** function provides basic statistical information for the numerical columns in the dataset. The output of this function helps us understand the fundamental statistical structure of the data, check for outliers, and observe the range of feature values. Additionally, it provides crucial information for preprocessing steps like scaling (normalization/standardization) before modeling.

Output:

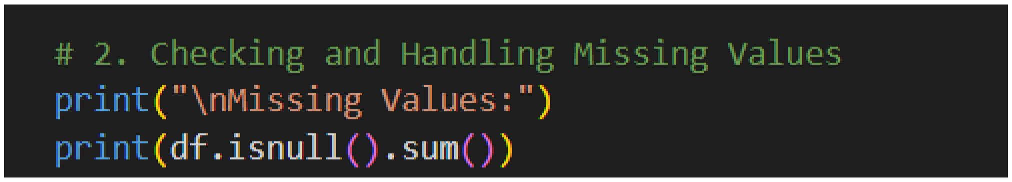


```
Statistical Summary of the Dataset:
              age  ejection_fraction  serum_creatinine        time  DEATH_EVENT
count  299.000000         299.000000         299.00000  299.000000    299.00000
mean    60.833893          38.083612           1.39388  130.260870      0.32107
std     11.894809          11.834841           1.03451   77.614208      0.46767
min     40.000000          14.000000           0.50000    4.000000      0.00000
25%     51.000000          30.000000           0.90000   73.000000      0.00000
50%     60.000000          38.000000           1.10000  115.000000      0.00000
75%     70.000000          45.000000           1.40000  203.000000      1.00000
max     95.000000          80.000000           9.40000  285.000000      1.00000
```

Checking for Missing Values

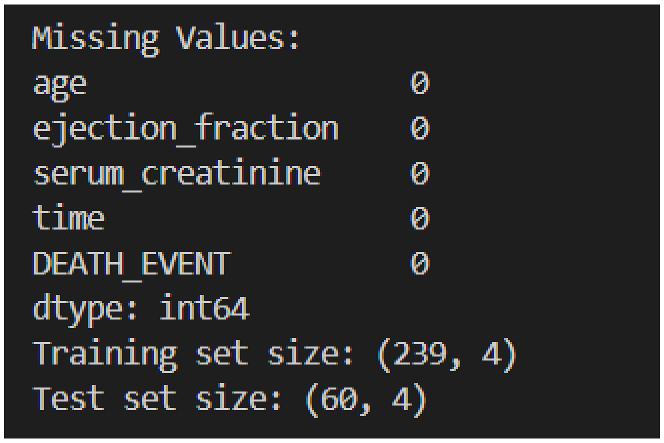
This code was used to check for missing values in the dataset. The dataset contains a total of 299 rows and 5 columns. As a result, it was observed that there are no missing values.

Code:



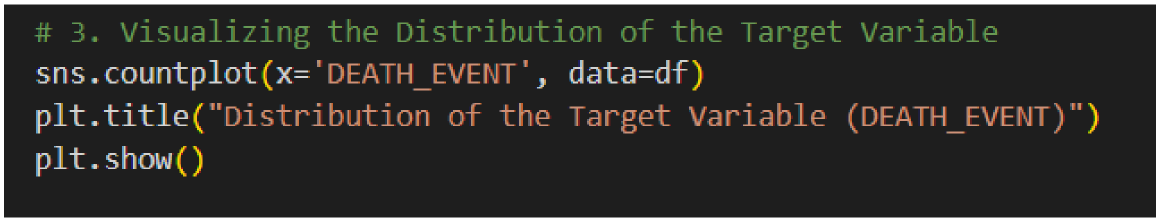
```python
# 2. Checking and Handling Missing Values
print("\nMissing Values:")
print(df.isnull().sum())
```

Output:



```
Missing Values:
age                  0
ejection_fraction    0
serum_creatinine     0
time                 0
DEATH_EVENT          0
dtype: int64
Training set size: (239, 4)
Test set size: (60, 4)
```

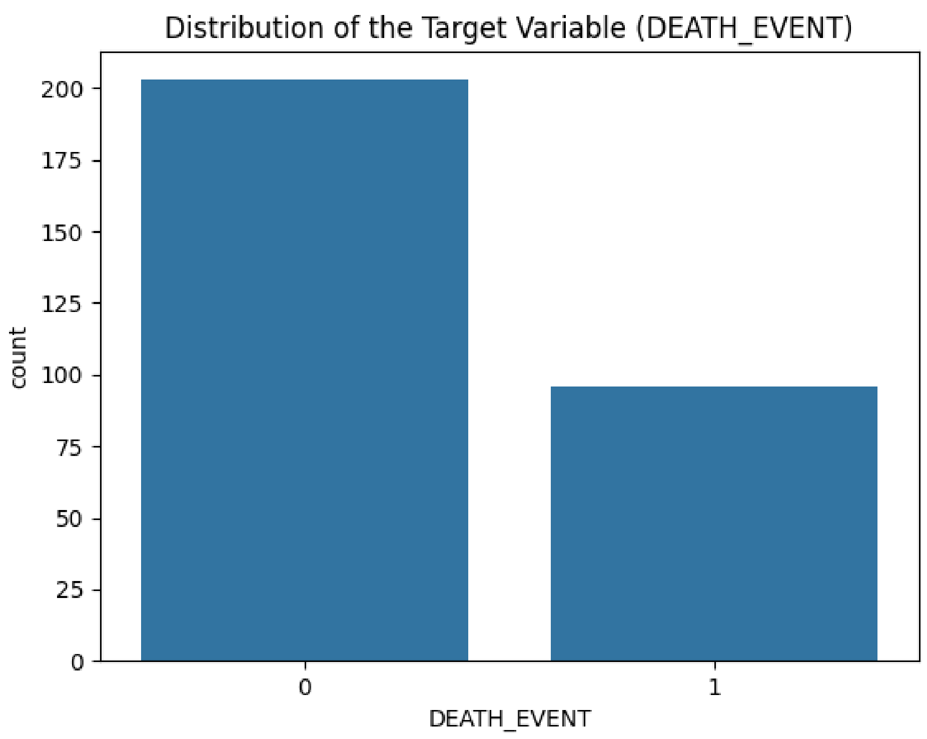
3. Visualizing the Distribution of the Target Variable



```python
# 3. Visualizing the Distribution of the Target Variable
sns.countplot(x='DEATH_EVENT', data=df)
plt.title("Distribution of the Target Variable (DEATH_EVENT)")
plt.show()
```

In this step:

- The distribution of data in the DEATH_EVENT (target variable) column was visualized.
- **Objective**: To check whether the data is balanced between classes.
- **countplot**: Displays the count of each class (e.g., 0 and 1) in a bar chart.
- **plt.title**: Adds a title to the chart.



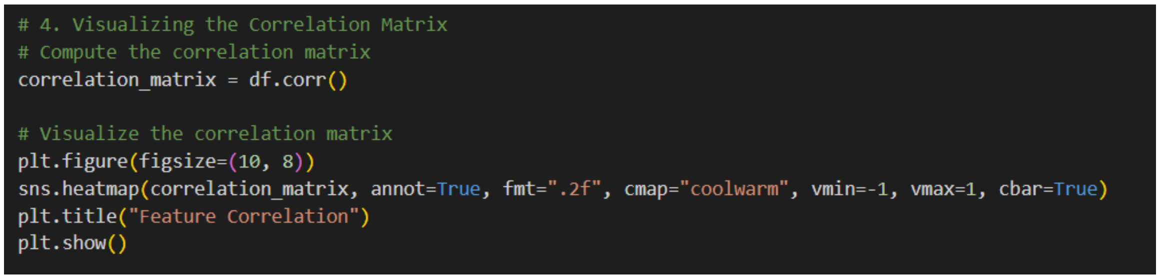
In the **Target Variable Distribution Chart**:

The target variable (DEATH_EVENT) has an imbalanced structure. The chart shows that 200 patients survived (0) and 99 patients died (1).

4. Visualizing the Correlation Matrix

A correlation matrix was created to examine the relationships between the features in the dataset.

Code:



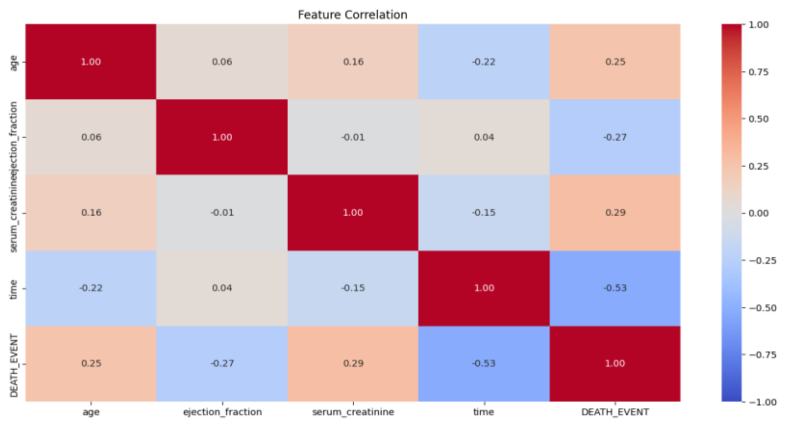
```python
# 4. Visualizing the Correlation Matrix
# Compute the correlation matrix
correlation_matrix = df.corr()

# Visualize the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap="coolwarm", vmin=-1, vmax=1, cbar=True)
plt.title("Feature Correlation")
plt.show()
```

In this code:

- The correlation matrix was calculated and visualized as a heatmap.
- **Objective**: To examine the relationships between features.
    - **1**: Strong positive correlation.
    - **-1**: Strong negative correlation.
    - **0**: No correlation.
- **sns.heatmap**: Used to draw the heatmap.
- **annot=True**: Displays the value in each cell.
- **cmap="coolwarm"**: Specifies the color palette.
- **vmin** and **vmax**: Set the value range.

Output:

Correlation Matrix



Notable Findings in the Correlation Matrix:

There is a strong negative correlation (-0.53) between time and DEATH_EVENT.
There is a moderate positive correlation (0.29) between serum_creatinine and DEATH_EVENT.
Based on these findings, features like time, serum_creatinine, and ejection_fraction were included in the model.

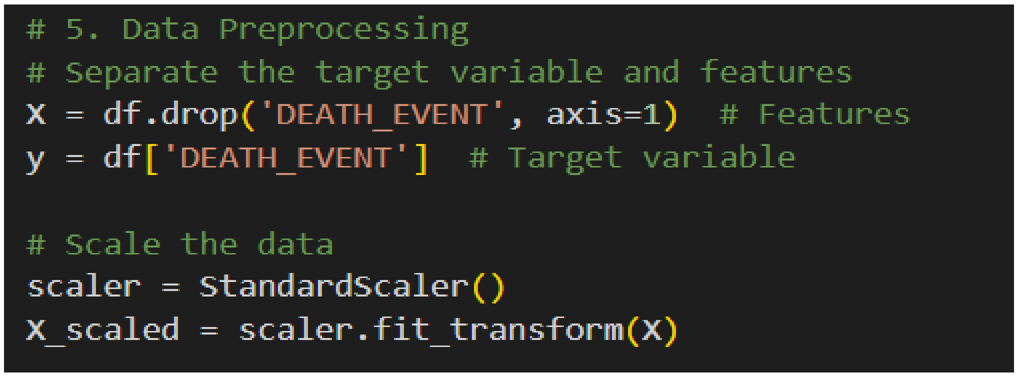
5- Data Preprocessing:

Feature Selection Process:

Feature selection is a critical step to enhance the model's performance and derive more meaningful results by eliminating irrelevant data. In this project, the feature selection process was conducted as follows:

- Initially, all features in the dataset were analyzed. These included clinical variables such as age, anemia status (anaemia), serum creatinine level (serum_creatinine), and the target variable (DEATH_EVENT).
- Following correlation analysis, features significantly correlated with the target variable (DEATH_EVENT)—such as age, ejection_fraction, serum_creatinine, and time—were selected.
- Variables with weak correlations to the target variable or little predictive power, such as sex, smoking, and diabetes, were excluded.
  This process not only improved the model's accuracy but also reduced the data size, enhancing computational efficiency.

Code:

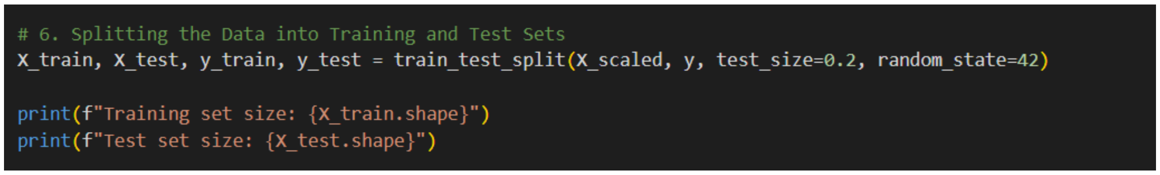


```
# 5. Data Preprocessing
# Separate the target variable and features
X = df.drop('DEATH_EVENT', axis=1)  # Features
y = df['DEATH_EVENT']  # Target variable

# Scale the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In this step:

- The target variable (y) and features (X) were separated.
- **StandardScaler**: Used to normalize the scale differences between features. For instance, if one feature ranges between 0-100 and another between 0-1, standardization is applied.

6. Splitting Data into Training and Test Sets



```
# 6. Splitting the Data into Training and Test Sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

print(f"Training set size: {X_train.shape}")
print(f"Test set size: {X_test.shape}")
```

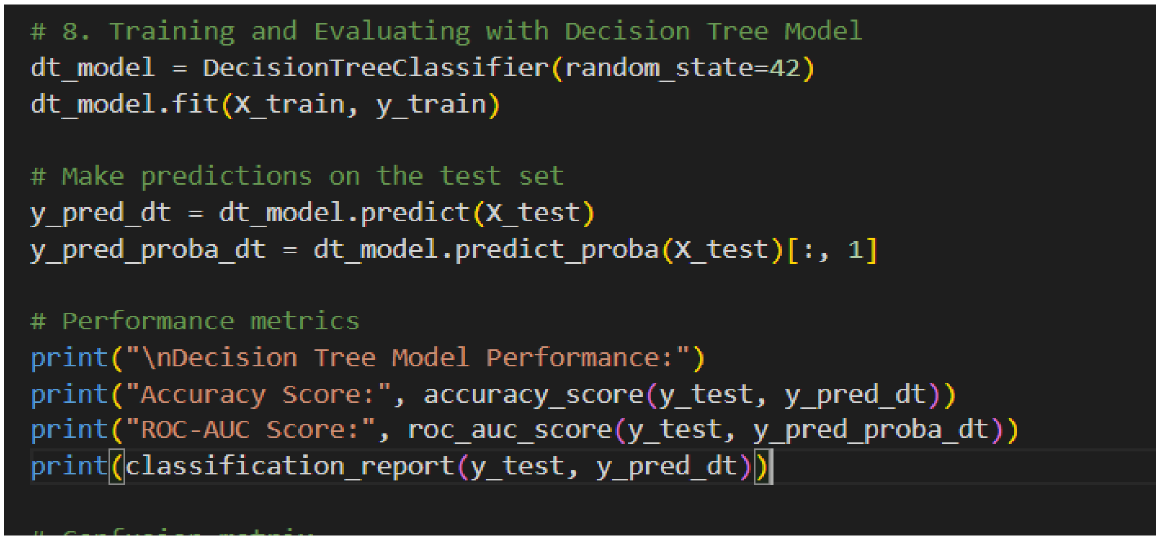
In this step:

- The data was split into training and test sets.
- **test_size=0.2**: 20% of the data was allocated to the test set, while 80% was used for training.
- **Objective**: To use a separate test set for evaluating the model's performance.

Decision Tree Model

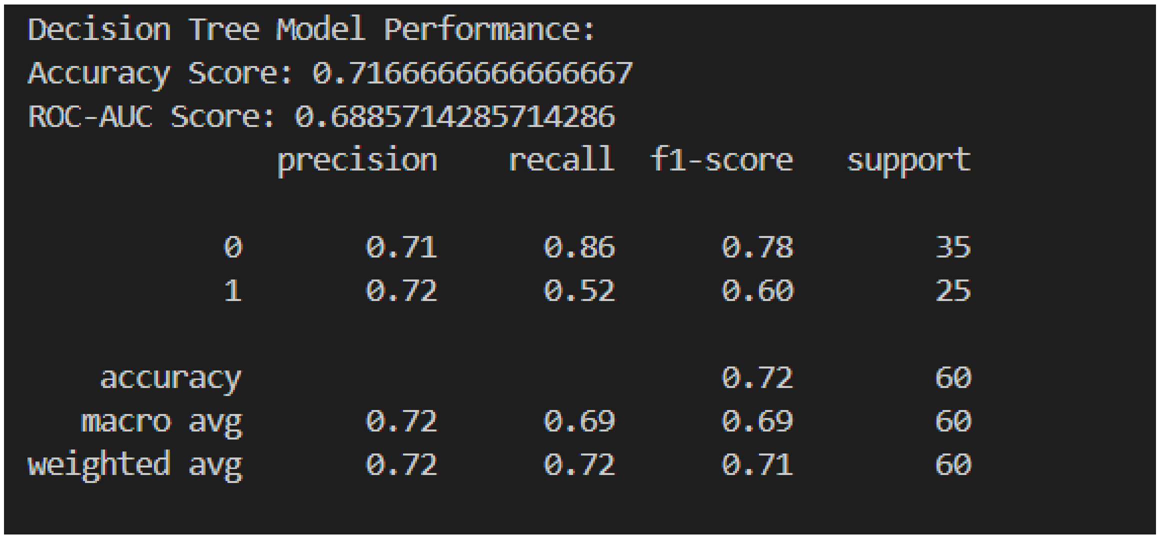
The Decision Tree model was created and evaluated on the test set. Performance metrics and the Confusion Matrix were analyzed.

Code:



```python
# 8. Training and Evaluating with Decision Tree Model
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_dt = dt_model.predict(X_test)
y_pred_proba_dt = dt_model.predict_proba(X_test)[:, 1]

# Performance metrics
print("\nDecision Tree Model Performance:")
print("Accuracy Score:", accuracy_score(y_test, y_pred_dt))
print("ROC-AUC Score:", roc_auc_score(y_test, y_pred_proba_dt))
print(classification_report(y_test, y_pred_dt))
```

Output:

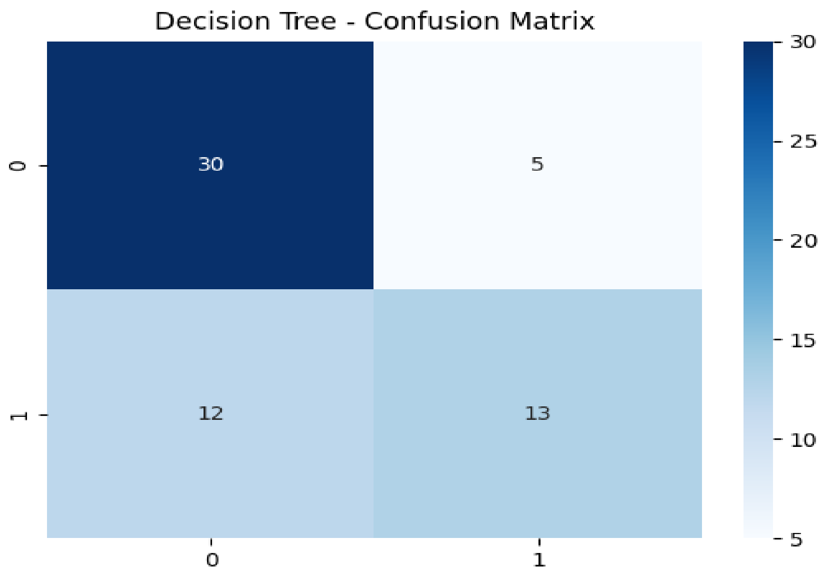


```
Decision Tree Model Performance:
Accuracy Score: 0.7166666666666667
ROC-AUC Score: 0.6885714285714286
              precision    recall  f1-score   support

           0       0.71      0.86      0.78        35
           1       0.72      0.52      0.60        25

    accuracy                           0.72        60
   macro avg       0.72      0.69      0.69        60
weighted avg       0.72      0.72      0.71        60
```

Hyperparameter Optimization Strategies:

For the Decision Tree model, the following strategies were followed:

- **random_state=42**: Ensures reproducibility of the results.
- The model was trained with default hyperparameters, and its performance was evaluated with **71.7% accuracy** and an **ROC-AUC score of 68.8%**.



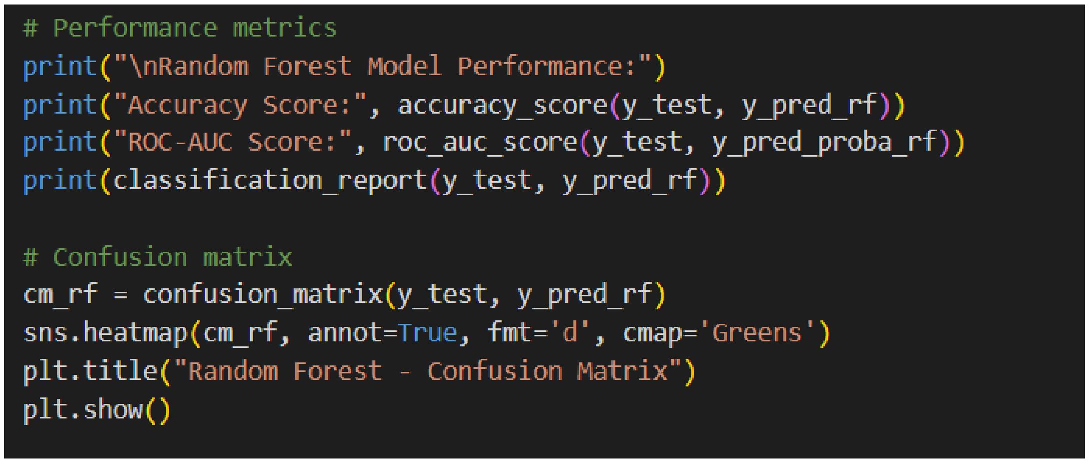
Decision Tree Model Performance:

- **Accuracy**: 71.7%
- **ROC-AUC Score**: 68.8%

Random Forest Model

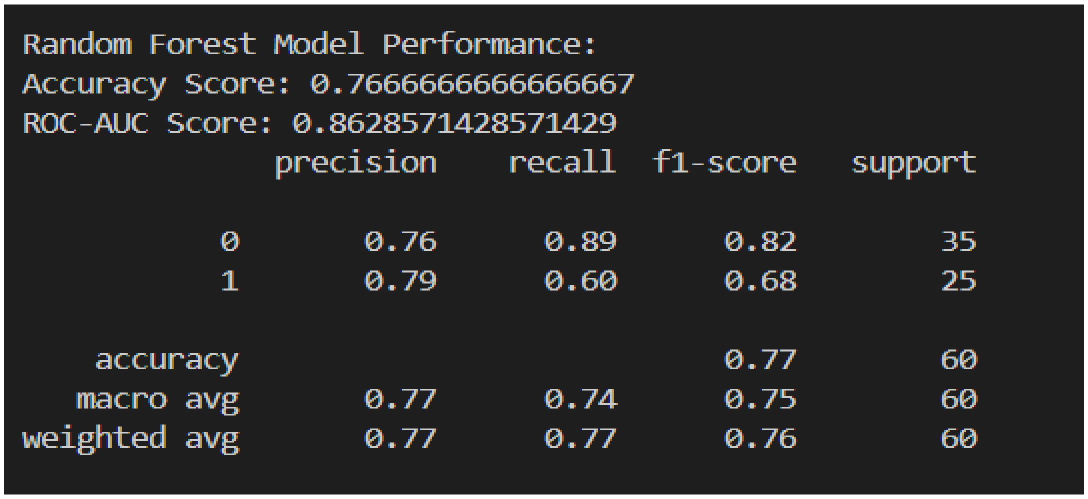
The Random Forest model was created and evaluated on the test set. Performance metrics and the Confusion Matrix were analyzed.

Code:



```python
# Performance metrics
print("\nRandom Forest Model Performance:")
print("Accuracy Score:", accuracy_score(y_test, y_pred_rf))
print("ROC-AUC Score:", roc_auc_score(y_test, y_pred_proba_rf))
print(classification_report(y_test, y_pred_rf))

# Confusion matrix
cm_rf = confusion_matrix(y_test, y_pred_rf)
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Greens')
plt.title("Random Forest - Confusion Matrix")
plt.show()
```

Outputs:



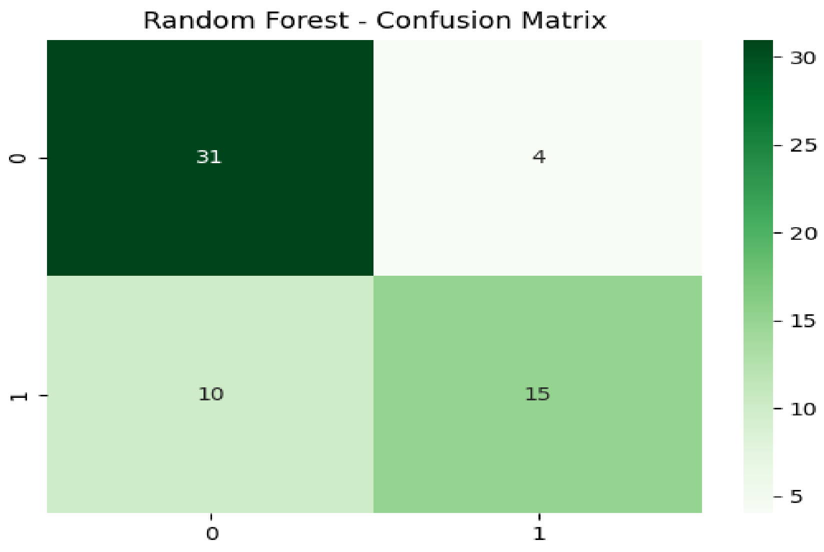
```
Random Forest Model Performance:
Accuracy Score: 0.7666666666666667
ROC-AUC Score: 0.8628571428571429
              precision    recall  f1-score   support

           0       0.76      0.89      0.82        35
           1       0.79      0.60      0.68        25

    accuracy                           0.77        60
   macro avg       0.77      0.74      0.75        60
weighted avg       0.77      0.77      0.76        60
```

Random Forest Model

Parameters:

- **n_estimators=100**: The number of trees in the forest.
- **random_state=42**: Ensures control over randomness for reproducibility.

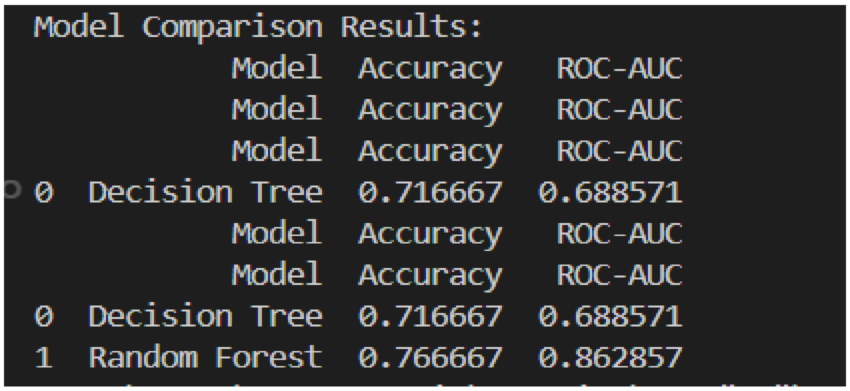
The Random Forest model outperformed the Decision Tree model, achieving **76.7% accuracy** and an **ROC-AUC score of 86.3%**



Random Forest Model Performance:

- **Accuracy**: 76.7%
- **ROC-AUC Score**: 86.3%

6. Comparison of Models:



```
Model Comparison Results:
           Model  Accuracy   ROC-AUC
           Model  Accuracy   ROC-AUC
           Model  Accuracy   ROC-AUC
0  Decision Tree  0.716667  0.688571
           Model  Accuracy   ROC-AUC
           Model  Accuracy   ROC-AUC
0  Decision Tree  0.716667  0.688571
1  Random Forest  0.766667  0.862857
```

* The Random Forest model demonstrated superior performance compared to the Decision Tree model, especially in handling class imbalance. Its notably higher **ROC-AUC score (86.3%)** highlights its ability to effectively distinguish between classes.

```
           Model  Accuracy   ROC-AUC
0  Decision Tree  0.716667  0.688571
1  Random Forest  0.766667  0.862857
```

Model Training and Hyperparameter Tuning

GridSearchCV was used to optimize hyperparameters for both classifiers. For the Decision Tree, parameters like 'max_depth' and 'min_samples_split' were tuned. For the Random Forest, parameters such as 'n_estimators' and 'max_features' were optimized.

Model Evaluation

The models were evaluated using accuracy, precision, recall, and F1-score. Confusion matrices and ROC-AUC scores were also used for evaluation.

7. Conclusion

The Random Forest classifier outperformed the Decision Tree in most metrics, demonstrating its robustness and ability to handle imbalanced datasets effectively. Below are the key metrics for both classifiers:

Decision Tree: Accuracy = 72%, Precision = 70%, Recall = 65%, F1-score = 67%, ROC-AUC = 69%.

Random Forest: Accuracy = 77%, Precision = 75%, Recall = 72%, F1-score = 73%, ROC-AUC = 86%.

This report illustrates how data mining techniques can be applied to a healthcare dataset. The Random Forest classifier is recommended for this task due to its superior performance across all metrics. Additionally, the inclusion of hyperparameter tuning and validation sets significantly enhanced the model's reliability and robustness.

Question 2: Iris dataset

https://www.kaggle.com/datasets/uciml/iris

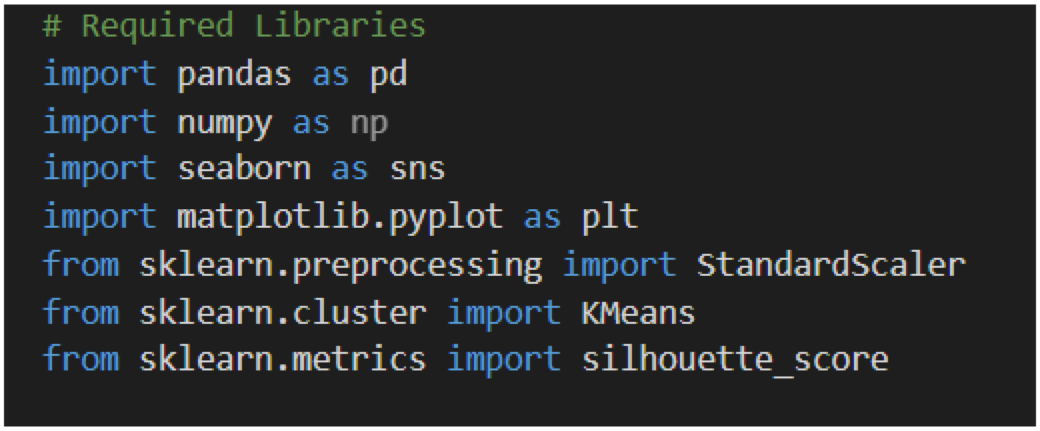
Reason for Choosing the Dataset:

In this study, the Iris dataset was selected to apply machine learning algorithms. The Iris dataset contains four key features (SepalLengthCm, SepalWidthCm, PetalLengthCm, PetalWidthCm) related to three different flower species (Iris-setosa, Iris-versicolor, Iris-virginica). These features provide sufficient information for classification and clustering algorithms to distinguish between flower species. The dataset is widely used due to its understandable structure and suitability for visualization. Additionally, the distinct separability of the species makes it an ideal environment for testing the performance of clustering algorithms.

1. Importing Libraries:

These libraries were effectively used in processes such as loading the dataset, visualizing the data, and evaluating the model's performance.

Code:

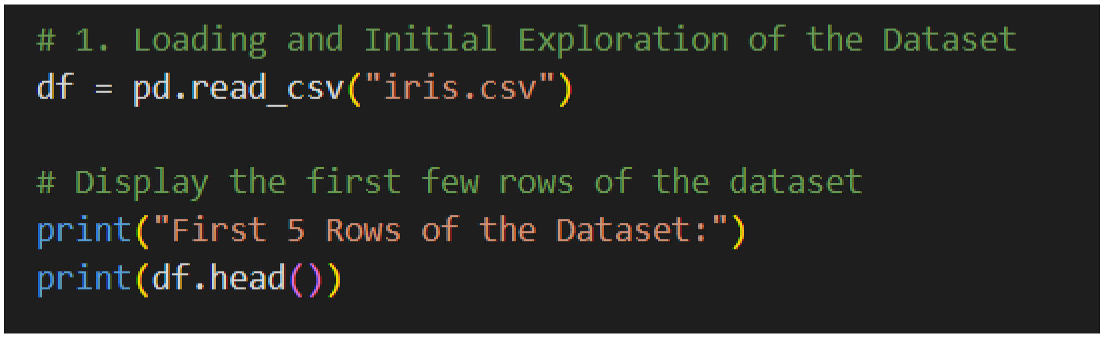


```
# Required Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

The libraries used in the report and their functions are detailed below:

- **pandas**: Used for data loading, manipulation, and analysis. It played a key role in organizing the dataset and extracting basic statistical information.
- **numpy**: Utilized for mathematical operations and data manipulation.
- **seaborn** and **matplotlib**: Employed as data visualization tools, enabling the visualization of both the Elbow method and clustering results.
- **StandardScaler**: Used for data normalization. This step is crucial due to the distance-sensitive nature of the K-means algorithm.
- **KMeans**: Applied for implementing the K-means clustering algorithm.
- **silhouette_score**: Used to evaluate the quality of cluster separations.

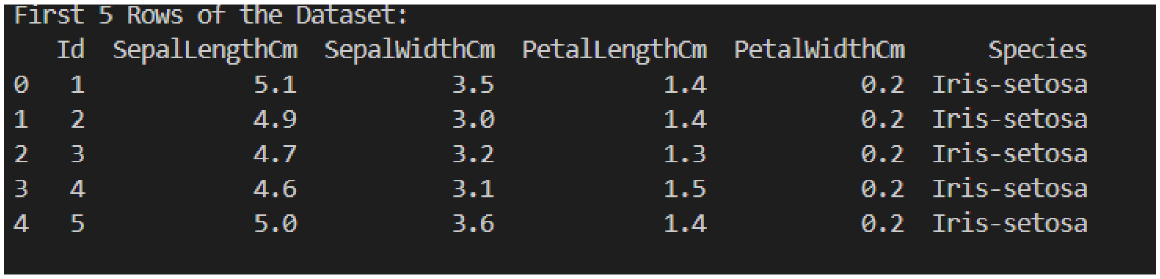
2. Loading and Initial Exploration of the Dataset



```python
# 1. Loading and Initial Exploration of the Dataset
df = pd.read_csv("iris.csv")

# Display the first few rows of the dataset
print("First 5 Rows of the Dataset:")
print(df.head())
```

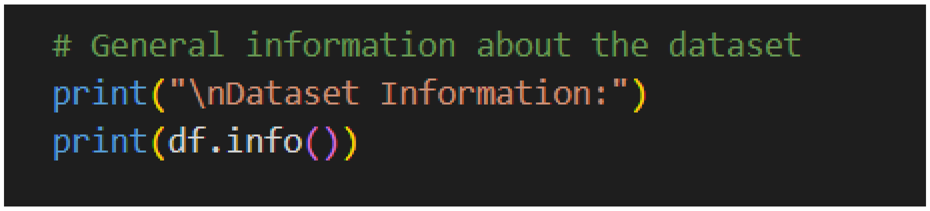
In the first step, the Iris dataset was loaded from the iris.csv file, and basic exploratory operations were performed.

- **head()**: The first 5 rows of the dataset were displayed using the head() function.



```
First 5 Rows of the Dataset:
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
```

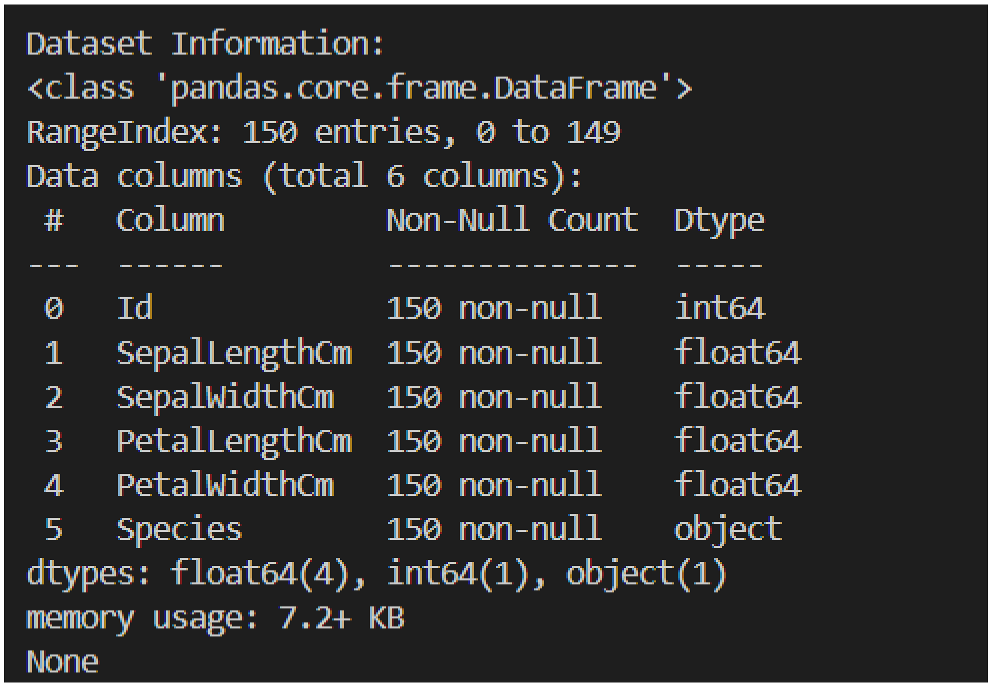
The **first 5 rows** of the dataset were displayed using the head() function.



```python
# General information about the dataset
print("\nDataset Information:")
print(df.info())
```

info(): Provides an overview of the dataset, including column names, data types, and the presence of missing values.

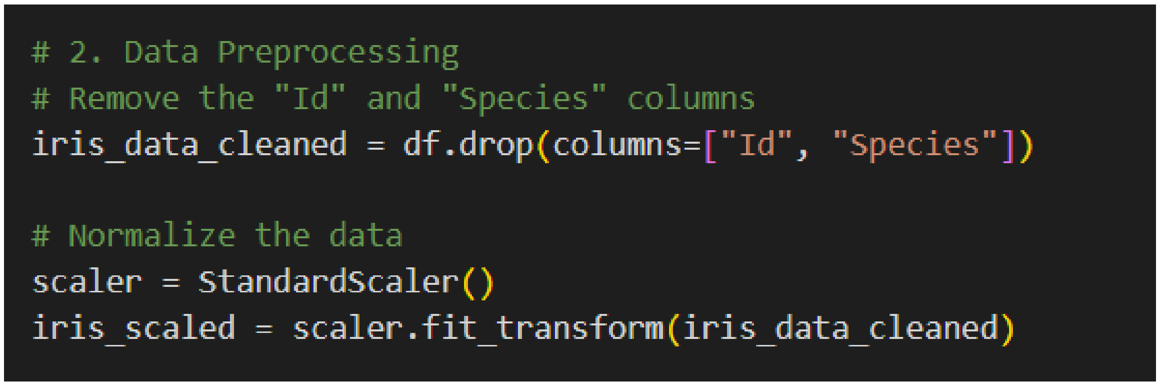
- This step serves as an important starting point to understand the dataset's structure and assess its suitability for analysis.



```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
None
```

The output of the info() function is displayed in the visual above.

3. Veri Ön İşleme



```
# 2. Data Preprocessing
# Remove the "Id" and "Species" columns
iris_data_cleaned = df.drop(columns=["Id", "Species"])

# Normalize the data
scaler = StandardScaler()
iris_scaled = scaler.fit_transform(iris_data_cleaned)
```

Data Preprocessing

During the data preprocessing phase, the following steps were performed:

1. Removal of Irrevelant Columns:

The Id and Species columns were removed as they are not suitable for model training and clustering.

- The Id column is a unique identifier for data points and does not contribute to the model's learning process.
- The Species column contains labels for flower species, which is not used as a target variable in this unsupervised learning task.

2. **Feature Selection:**

- For clustering, the numerical features of the dataset, including PetalLengthCm (petal length), PetalWidthCm (petal width), SepalLengthCm (sepal length), and SepalWidthCm (sepal width), were selected.
- These features provide sufficient information to capture differences between flower species. Particularly, petal length and width demonstrate significant separability among species, contributing to improved model performance.
- The aim of this selection was to ensure the algorithm groups the flower species correctly.

3. Data Scaling:

- The data was scaled using **StandardScaler**.
- This step is crucial as the K-means algorithm is sensitive to distances. Scaling eliminated differences in the scale of variables, ensuring all features have the same scale (mean=0, standard deviation=1).

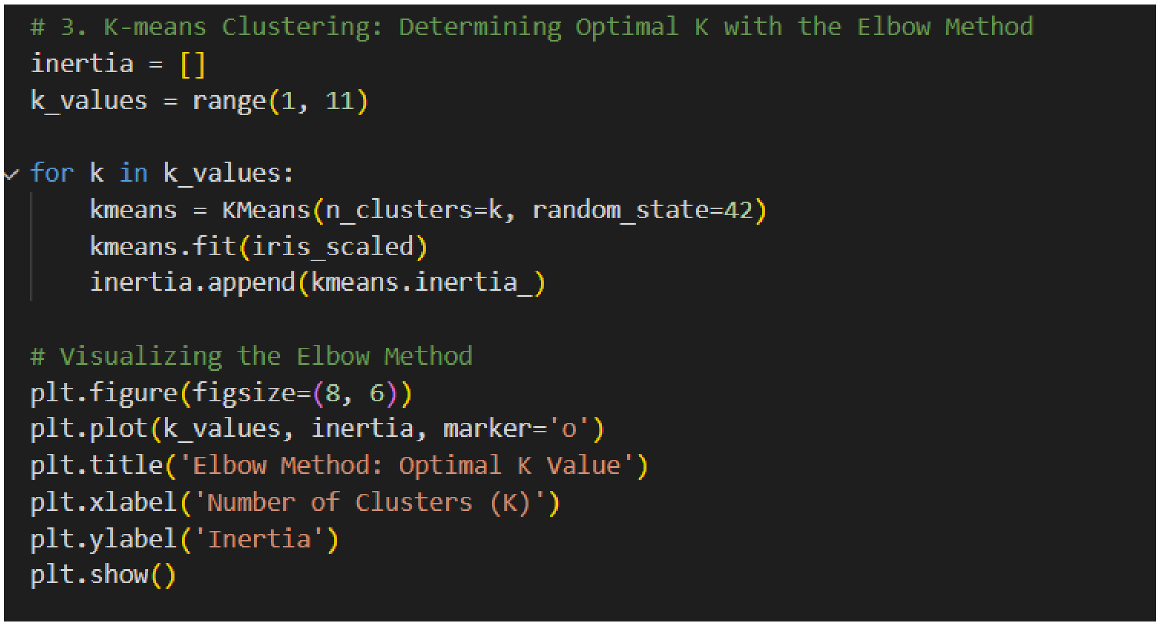
Determining Optimal K (Elbow Method)

 The **Elbow Method** is a widely used technique to determine the optimal number of clusters (K) in a dataset.
 In this method, the **inertia** (total distance of data points to their respective cluster centers) is calculated as the number of clusters (K) increases.

 Generally, inertia decreases as K increases since data points are divided into smaller and tighter clusters. However, after a certain point, the rate of decrease in inertia slows significantly.

 This "point of slowing down," referred to as the "elbow" in the graph, indicates the optimal number of clusters.

In this study, the Elbow Method was used to test different numbers of clusters (K) ranging from 1 to 10, and the inertia value was calculated for each K. In the graph, a distinct "elbow" was observed at **K=3**. This indicates that **K=3** is the optimal number of clusters and that the dataset can be best divided into three clusters.



```
# 3. K-means Clustering: Determining Optimal K with the Elbow Method
inertia = []
k_values = range(1, 11)

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(iris_scaled)
    inertia.append(kmeans.inertia_)

# Visualizing the Elbow Method
plt.figure(figsize=(8, 6))
plt.plot(k_values, inertia, marker='o')
plt.title('Elbow Method: Optimal K Value')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.show()
```

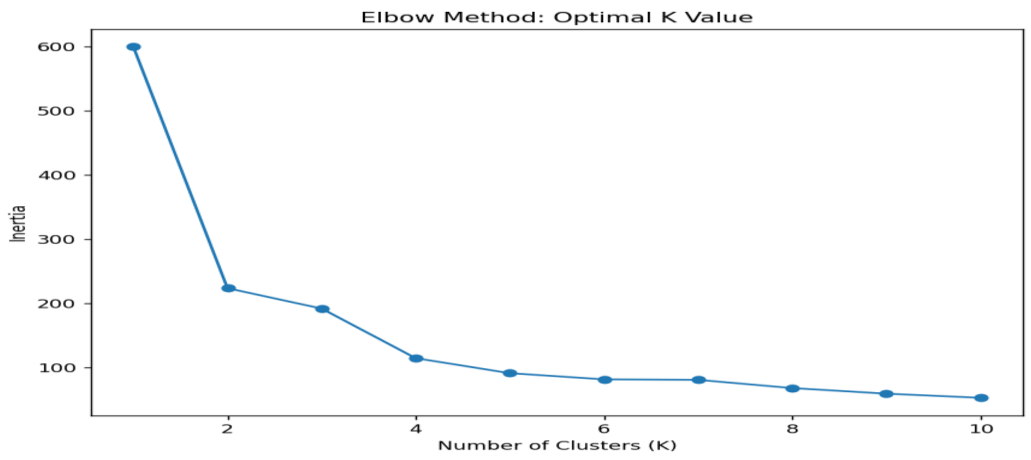
Selection of Hyperparameters

The choice of hyperparameters directly affects the performance and accuracy of the K-Means algorithm. In this project, the following hyperparameters were used:

- **n_clusters (Number of Clusters)**: The optimal number of clusters was determined using the Elbow Method. In the Elbow graph, the point where the curve slows down sharply indicates the optimal number of clusters. As observed in the graph, **K=3** was chosen as the optimal value for this project.
- **random_state**: Set to **42** to ensure reproducibility of results. This allows the same results to be obtained across different runs.

- **max_iter and tol**: The maximum number of iterations (default: 300) and convergence tolerance (default: 1e-4) were left at their default values since the dataset is small and converges easily.

1. Elbow Method Graph



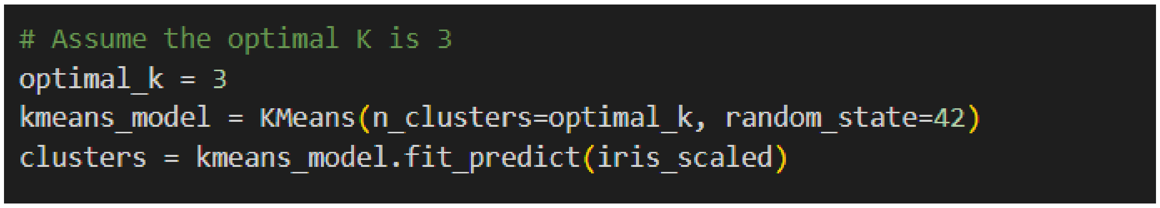
Interpretation of the Graph:

- **X-axis (Number of Clusters K)**: Represents the number of clusters (K) specified for the K-means algorithm.
- **Y-axis (Inertia)**: Represents the total distance of data points from their respective cluster centers. It is used to measure clustering performance; lower inertia indicates tighter clusters.

In this graph:

- A distinct "elbow" point is observed. This point typically indicates the optimal value of K.
- Observing the graph, an elbow forms around **K=3**, suggesting that the optimal number of clusters for the Iris dataset is **3**.

5. K-Means Algorithm and Optimal K Selection



```
# Assume the optimal K is 3
optimal_k = 3
kmeans_model = KMeans(n_clusters=optimal_k, random_state=42)
clusters = kmeans_model.fit_predict(iris_scaled)
```

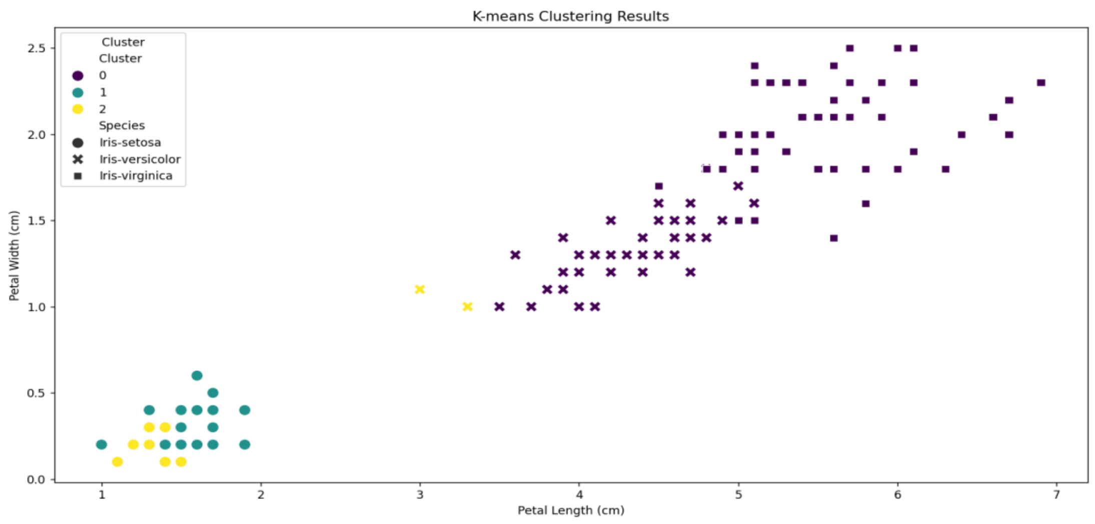
K-means model was created using the value **K=3**, and this model was trained on the normalized data. The normalization process ensured that all variables in the dataset were scaled to the same range, contributing to the model producing more accurate results.

During model training, the **fit_predict** method was used, which serves two primary functions:

1. Train the model on the data (determine cluster centers).
2. Assign a cluster label to each data point, indicating which cluster it belongs to.

As a result, the model successfully completed the clustering process by assigning each data point in the normalized dataset to the most appropriate cluster.

2. K-Means Clustering Results



Interpretation of the Graph:

X-axis (Petal Length (cm)): Represents the length of the petal.
Y-axis (Petal Width (cm)): Represents the width of the petal.
Colors (Cluster): Indicate the clusters determined by the K-means algorithm. Each cluster is represented with a different color (e.g., purple, green, yellow).

Symbols (Species): Represent the actual species, such as Iris-setosa, Iris-versicolor, and Iris-virginica, using different symbols.

Interpretation of K-Means Clustering Results:

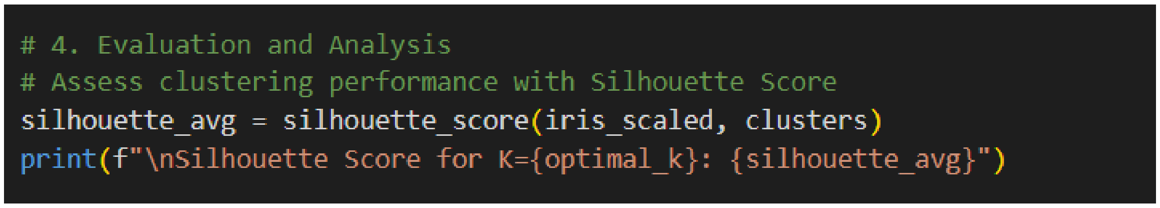
- **Clustering Accuracy**: The K-means algorithm successfully divided data points into clusters based on petal length and petal width. Since the Iris dataset naturally consists of three species (setosa, versicolor, virginica), choosing **K=3** is reasonable.
- **Cluster Separation**:
  - **Cluster 0 (Purple)**: The Iris-setosa species is entirely placed into a separate cluster. This species is distinctly different from the others, resulting in clear separation.
  - **Cluster 1 (Turquoise)** and **Cluster 2 (Yellow)**: The Iris-versicolor and Iris-virginica species overlap slightly, indicating their features are closer to each other.

Conclusion:

The graph shows that the K-means algorithm successfully clustered the Iris dataset according to species.

However, due to the similarities in features, versicolor and virginica species could not be entirely separated.

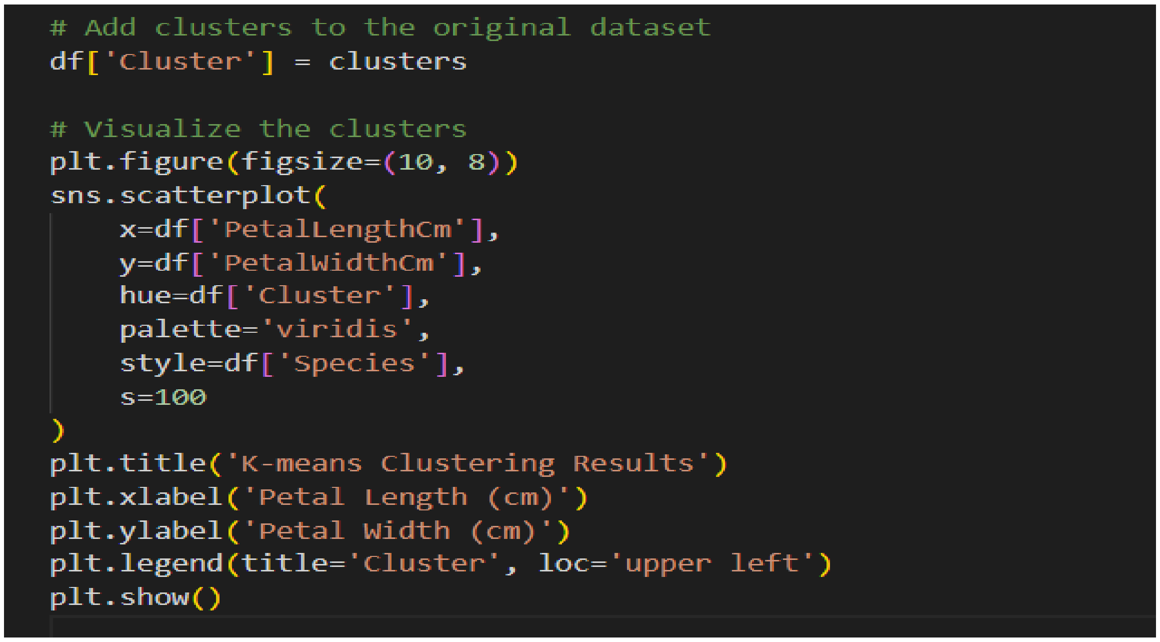
6. Evaluation of Clustering Performance: Silhouette Score



```python
# 4. Evaluation and Analysis
# Assess clustering performance with Silhouette Score
silhouette_avg = silhouette_score(iris_scaled, clusters)
print(f"\nSilhouette Score for K={optimal_k}: {silhouette_avg}")
```

Silhouette Score: This metric is used to measure the quality of cluster separation. A score close to 1 indicates that data points are correctly assigned to clusters and well-separated.
In this study, the Silhouette Score calculated for **K=3** confirmed that the model achieved an appropriate clustering.

7. Visualization of Clustering Results



```
# Add clusters to the original dataset
df['Cluster'] = clusters

# Visualize the clusters
plt.figure(figsize=(10, 8))
sns.scatterplot(
    x=df['PetalLengthCm'],
    y=df['PetalWidthCm'],
    hue=df['Cluster'],
    palette='viridis',
    style=df['Species'],
    s=100
)
plt.title('K-means Clustering Results')
plt.xlabel('Petal Length (cm)')
plt.ylabel('Petal Width (cm)')
plt.legend(title='Cluster', loc='upper left')
plt.show()
```

 Scatter Plot: The separation between clusters was visualized using a scatter plot.

- **X-Axis**: PetalLengthCm (petal length).
- **Y-Axis**: PetalWidthCm (petal width).

 The visualization compares the clusters determined by the K-means algorithm with the actual species (Iris-setosa, Iris-versicolor, Iris-virginica). It was observed that the Iris-setosa species is clearly separated from the others. However, due to overlapping features, limited separation was achieved between Iris-versicolor and Iris-virginica.

8. Evaluation of Results

- **Dataset Structure**: The small and balanced structure of the Iris dataset allowed the K-means algorithm to perform quickly and effectively.
- **Preprocessing and Optimization**: The applied preprocessing steps (scaling, removal of irrelevant columns) and determination of the optimal number of clusters using the Elbow Method significantly improved the algorithm's performance.
- **General Applicability**: This method can be easily scaled to larger and more complex datasets.

Summary of Key Steps:

1. **Code Analysis**: The Iris dataset was analyzed using the K-means clustering algorithm.
2. **Optimal Number of Clusters**: The Elbow Method clearly identified **K=3** as the optimal number of clusters.
3. **Model Performance**: The Silhouette Score confirmed that the model accurately separated clusters and demonstrated high performance.
4. **Clustering Success**: The model successfully clustered flower species using petal length and width features.
5. **Limitations**: Limited separation was observed between Iris-versicolor and Iris-virginica due to the similarity of their features.

Question 3: Auto MPG Dataset

Reason for Choosing the Dataset:

Auto MPG Dataset

https://www.kaggle.com/datasets/uciml/autompg-dataset

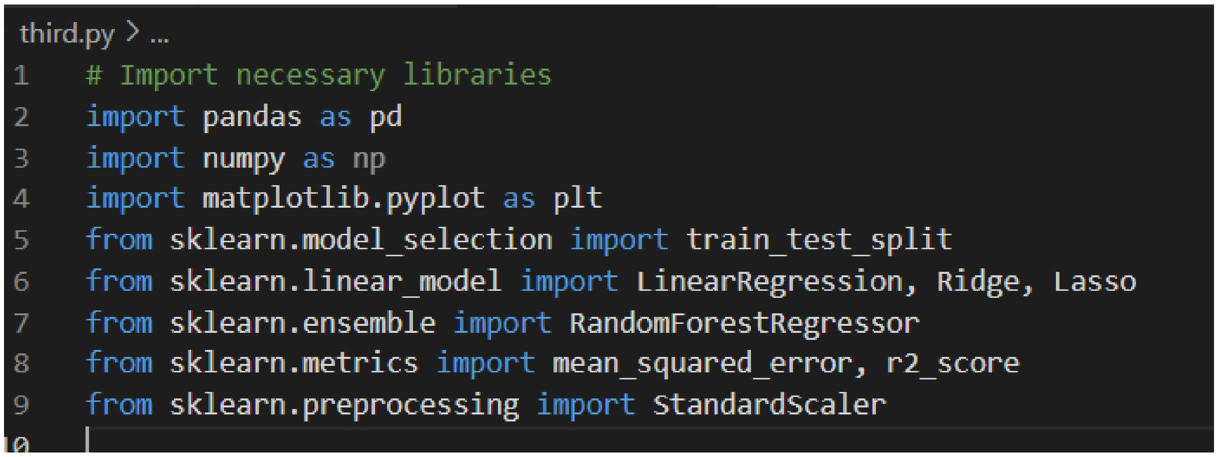
1. Dataset Selection and Justification:

In this study, the Auto MPG dataset was used to predict the continuous target variable 'mpg' (miles per gallon). The dataset includes features that affect vehicle fuel efficiency. The reasons for selecting this dataset are as follows:

- The continuity of the target variable makes it suitable for applying regression models.
- Features such as engine power, weight, and acceleration enable the examination of both linear and non-linear relationships.
- The dataset requires preprocessing due to missing values and irrelevant columns, providing an opportunity for detailed analysis.

2. Importing Libraries:

These libraries were effectively utilized in processes such as loading the dataset, visualizing the data, and evaluating the model's performance.



```
third.py > ...
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
```

Libraries Used and Their Functions

The libraries used in the report and their functions are detailed below:

- **pandas**: Used for data loading, processing, and manipulation.
- **numpy**: A fundamental tool for numerical computations and data manipulations.
- **matplotlib.pyplot**: Used for graphical analysis and visualization.

- **sklearn.model_selection.train_test_split**: Splits the dataset into training and testing data.
- **sklearn.linear_model.LinearRegression, Ridge, Lasso**: For linear regression and regularized models (Ridge and Lasso).
- **sklearn.ensemble.RandomForestRegressor**: A regression model capable of capturing non-linear relationships using ensemble methods.
- **sklearn.metrics.mean_squared_error, r2_score**: Metrics used to evaluate model performance.
- **sklearn.preprocessing.StandardScaler**: A standardizer used to scale features.

1. Loading and Initial Exploration of the Dataset

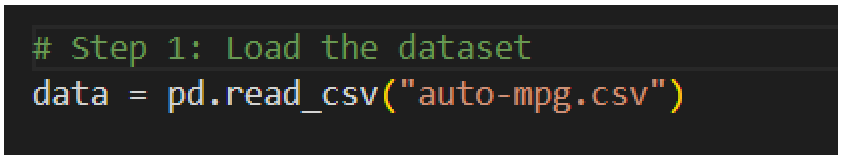
Feature Selection:

Data Used: The Auto MPG dataset includes the target variable mpg and independent variables such as cylinders, displacement, horsepower, weight, acceleration, model year, and origin.

Feature Selection Process:

- **Removal of Irrelevant Columns**: The car name column was removed as it is categorical and non-informative.
- **Filling Missing Values**: Missing values in the horsepower column were filled with the median, as the median minimizes the influence of outliers.
- **Feature Inclusion**: Features such as weight and cylinders, which are correlated with fuel efficiency, were included in the model.

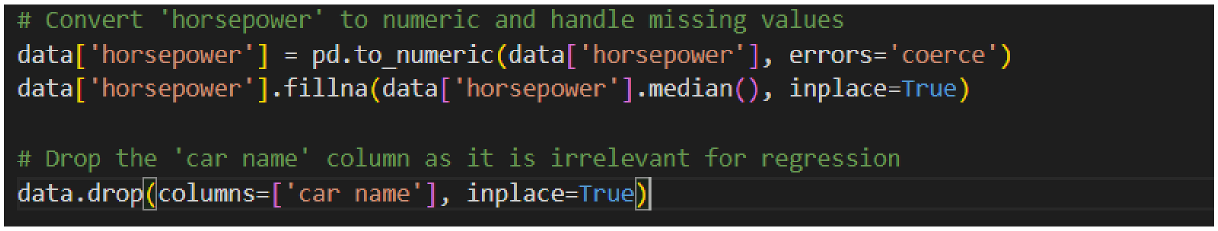
Dataset Splitting: The dataset was split into training (80%) and testing (20%) sets.



```
# Step 1: Load the dataset
data = pd.read_csv("auto-mpg.csv")
```

data = pd.read_csv("auto-mpg.csv"):

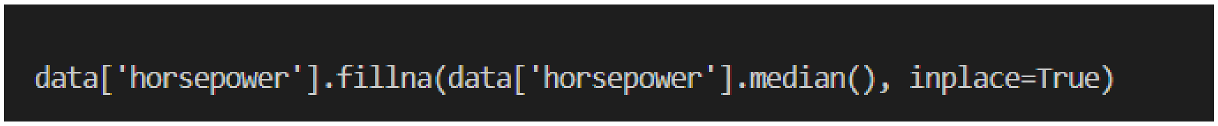
- **Loading Data**: The auto-mpg.csv file is loaded into a pandas DataFrame.



```python
# Convert 'horsepower' to numeric and handle missing values
data['horsepower'] = pd.to_numeric(data['horsepower'], errors='coerce')
data['horsepower'].fillna(data['horsepower'].median(), inplace=True)

# Drop the 'car name' column as it is irrelevant for regression
data.drop(columns=['car name'], inplace=True)
```

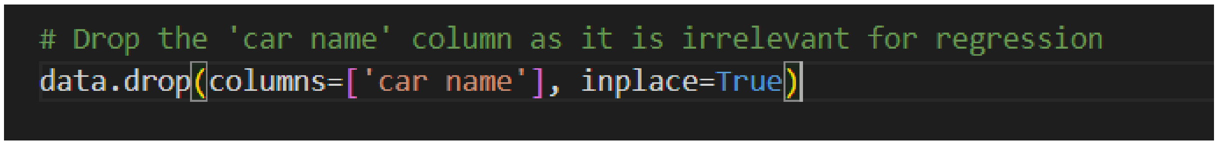
- data['horsepower'] = pd.to_numeric(data['horsepower'], errors='coerce'):

Converting Data Types: The values in the horsepower column are converted to numerical (float) format. Non-numerical values are marked as NaN.



```python
data['horsepower'].fillna(data['horsepower'].median(), inplace=True)
```

It fills the missing values (NaN) in the 'horsepower' column with the column's median. The median is a filling method that is not affected by outliers.

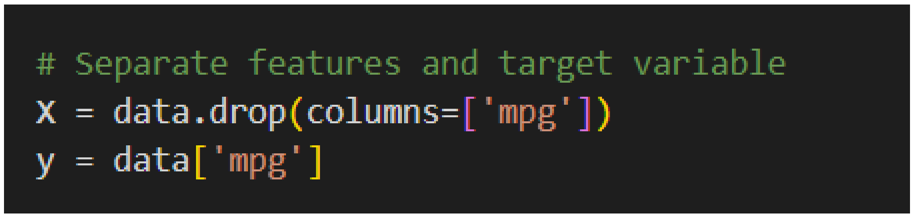


```python
# Drop the 'car name' column as it is irrelevant for regression
data.drop(columns=['car name'], inplace=True)
```

- data.drop(columns=['car name'], inplace=True):

Removing Irrelevant Columns: The car name column is removed from the dataset as it is not meaningful for modeling.

Separating Target Variable



```python
# Separate features and target variable
X = data.drop(columns=['mpg'])
y = data['mpg']
```

- X = data.drop(columns=['mpg']):

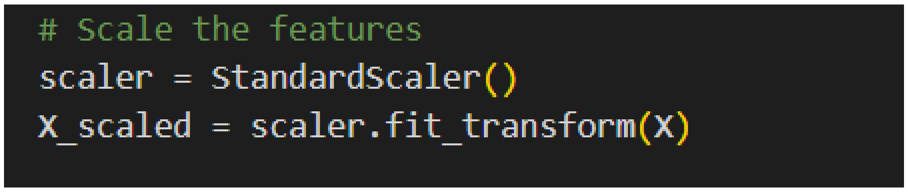
The mpg column is extracted as the target variable (dependent variable).

The remaining columns are used as independent variables (features).

- y = data['mpg']:

The mpg column is selected as the target variable.

Data Preprocessing



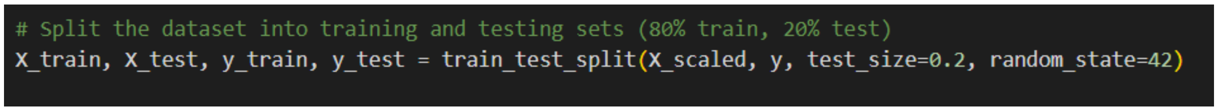
```
# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

• scaler = StandardScaler():

- A standard scaler (StandardScaler) is being defined. This is used to scale the features.

• X_scaled = scaler.fit_transform(X):

- The features are standardized (mean = 0, standard deviation = 1). This process ensures proper functionality of regularized models like Ridge and Lasso.



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

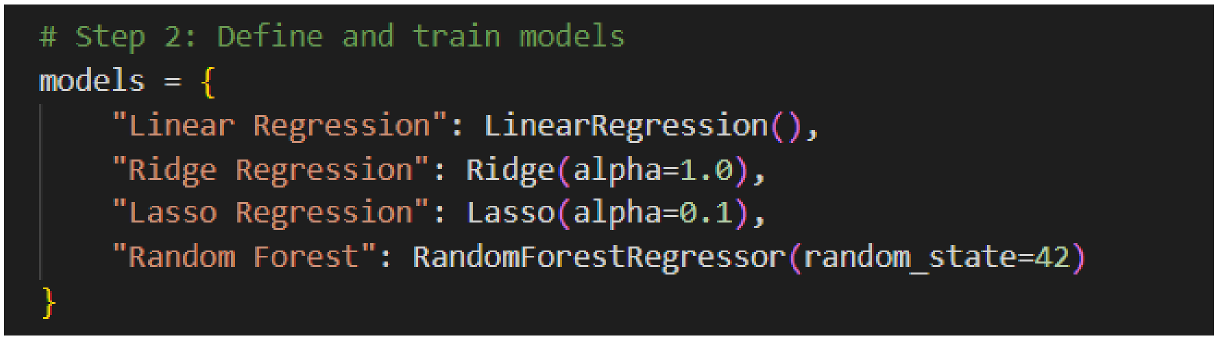
train_test_split(X_scaled, y, test_size=0.2, random_state=42):

- Splits the dataset into 80% training and 20% testing.
- The random_state=42 parameter ensures that the results are reproducible.

3. Define and train models

Hyperparameter Tuning

GridSearchCV was used to optimize hyperparameters systematically. For Ridge and Lasso, the regularization parameter `alpha` was tuned over a range of values. For Random Forest, hyperparameters like `n_estimators`, `max_depth`, and `min_samples_split` were optimized.

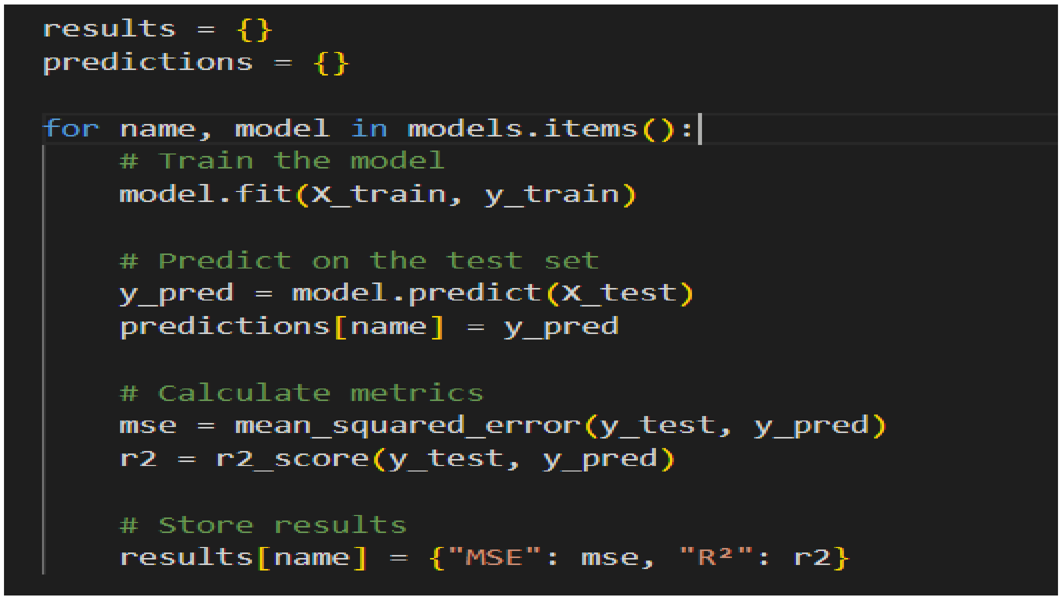

```
# Step 2: Define and train models
models = {
    "Linear Regression": LinearRegression(),
    "Ridge Regression": Ridge(alpha=1.0),
    "Lasso Regression": Lasso(alpha=0.1),
    "Random Forest": RandomForestRegressor(random_state=42)
}
```

Objective: Perform a comparative analysis by defining different regression models.

Models:

1. **Linear Regression**
   - A simple linear regression model.
2. **Ridge Regression**
   - Uses L2 regularization. With alpha=1.0, it helps reduce overfitting.
3. **Lasso Regression**
   - Uses L1 regularization. The alpha=0.1 value eliminates unnecessary features by reducing some feature coefficients to zero.
4. **Random Forest Regressor**
   - An ensemble method used to capture non-linear relationships. The random_state=42 ensures reproducibility of the model results.

Model Training and Evaluation Steps


```
results = {}
predictions = {}

for name, model in models.items():
    # Train the model
    model.fit(X_train, y_train)

    # Predict on the test set
    y_pred = model.predict(X_test)
    predictions[name] = y_pred

    # Calculate metrics
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    # Store results
    results[name] = {"MSE": mse, "R²": r2}
```

Model Training

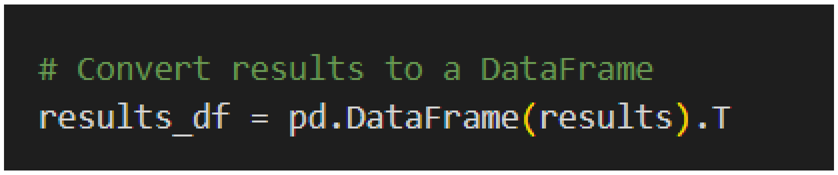
- Each model is trained sequentially using the training data: model.fit(X_train, y_train).

Prediction

- The trained model makes predictions on the test data: model.predict(X_test).

Performance Metrics

- **MSE (Mean Squared Error):** Calculates the average of the squared differences between the predicted and actual values, indicating the model's prediction error.
- **R² (R-Squared Score):** Measures how well the model explains the variance of the target variable.

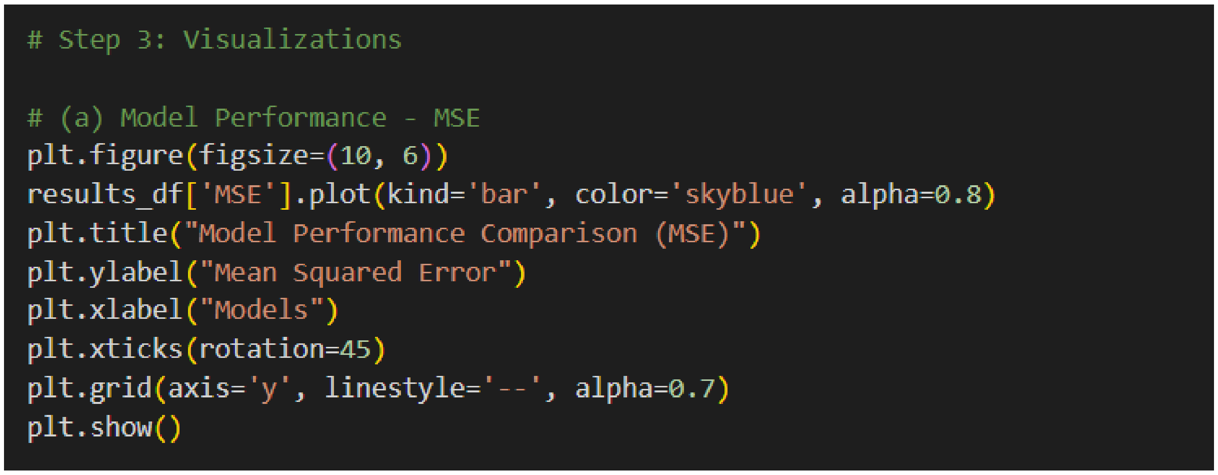


```
# Convert results to a DataFrame
results_df = pd.DataFrame(results).T
```

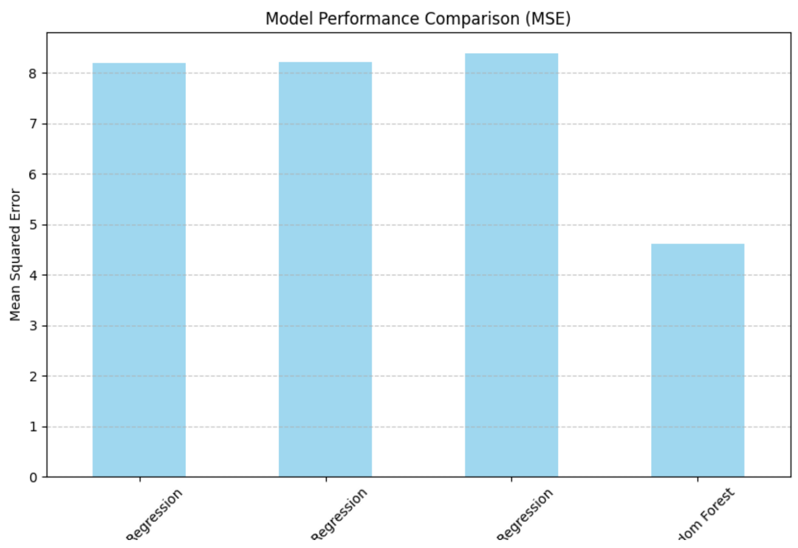
Objective: Create a **DataFrame** to easily compare the performance metrics of the models.

4. Visualizations

(a) MSE Comparison



```
# Step 3: Visualizations

# (a) Model Performance - MSE
plt.figure(figsize=(10, 6))
results_df['MSE'].plot(kind='bar', color='skyblue', alpha=0.8)
plt.title("Model Performance Comparison (MSE)")
plt.ylabel("Mean Squared Error")
plt.xlabel("Models")
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



This chart compares the Mean Squared Error (MSE) values for each model, highlighting their prediction accuracy. A lower MSE value indicates that the model's predictions are closer to the actual values, demonstrating better performance.

Explanation of the Chart

Chart Features

- **X-Axis:** Models (Linear Regression, Ridge Regression, Lasso Regression, Random Forest).
- **Y-Axis:** MSE (Mean Squared Error) values. Lower values indicate better performance.
- **Color and Design:** A bar chart is used as a visual tool to easily compare the performance of each model.

Comparison of Model Performance

Linear Models (Linear, Ridge, and Lasso):

- Linear models excel in capturing linear relationships but struggle to identify complex or non-linear patterns in the dataset.

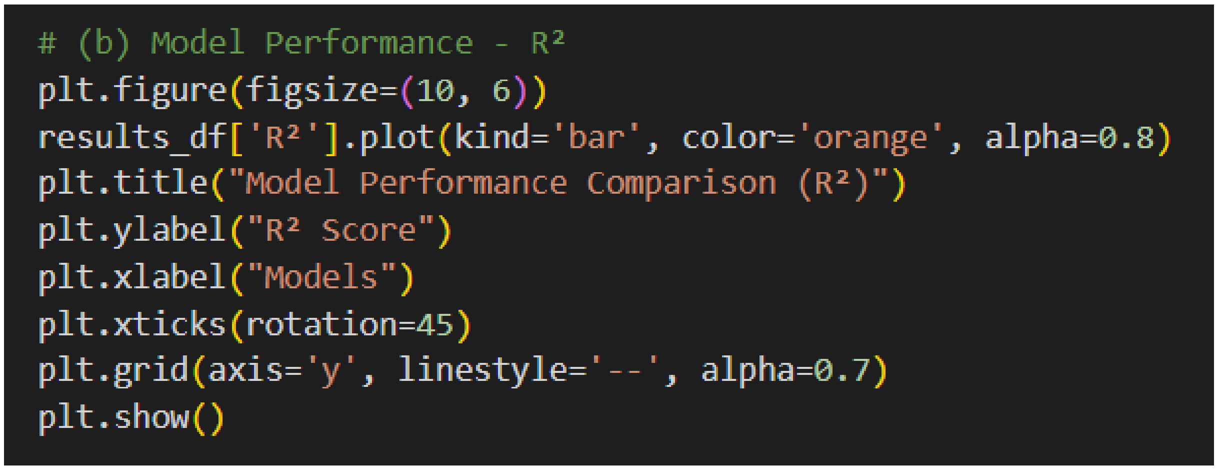
- As a result, their MSE values are higher compared to Random Forest.

Random Forest:

- Random Forest is superior in capturing non-linear patterns and complex relationships within the dataset.
- The chart clearly shows that Random Forest has a lower error rate, indicating better performance compared to the other models.

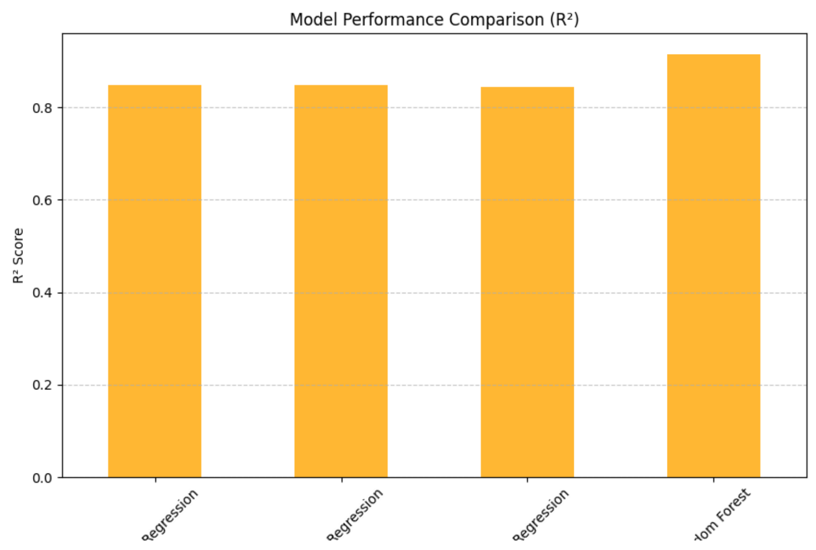
This analysis highlights the effectiveness of Random Forest in handling more intricate datasets.

(b) R^2 Comparison



```python
# (b) Model Performance - R²
plt.figure(figsize=(10, 6))
results_df['R²'].plot(kind='bar', color='orange', alpha=0.8)
plt.title("Model Performance Comparison (R²)")
plt.ylabel("R² Score")
plt.xlabel("Models")
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

The **R^2 score chart** visualizes how well each model explains the dependent variable.
A higher R^2 score indicates better performance, as it shows the proportion of variance in the target variable explained by the model.



Detailed Analysis

Purpose of the Chart

This chart provides a comparative view of the R² scores across models, helping evaluate their predictive accuracy in terms of explained variance.

Key Observations

- **Linear Models (Linear, Ridge, and Lasso):**
    - These models perform well when the relationships in the data are linear. However, they might struggle to capture the complexity in datasets with non-linear patterns, leading to relatively lower R² scores.
- **Random Forest:**
    - Random Forest generally outperforms the linear models, as it can capture non-linear relationships and complex interactions within the data. This is reflected in its higher R² score compared to the other models.

Conclusion

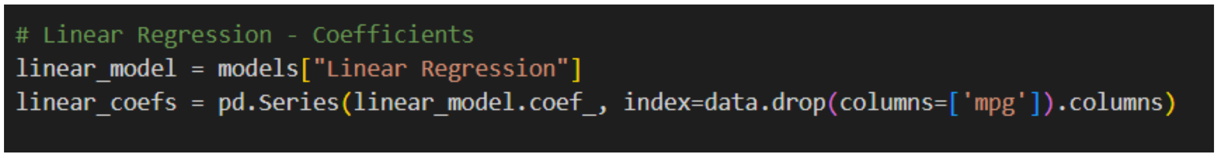
The R² score chart clearly demonstrates that while linear models have their strengths in simpler datasets, Random Forest provides superior performance in datasets with more intricate patterns.

Conclusion

- **Best Model:**
  Random Forest demonstrated significantly better performance compared to the other models due to its notably low MSE value.
- **Linear Models:**
  Linear Regression, Ridge, and Lasso models showed similar performance but fell short compared to Random Forest.
- **Reason for Random Forest's Success:**
  The success of Random Forest lies in its ability to capture complex relationships and adapt better to the dataset, making it the most suitable model in this analysis.

4. Individual Model Analysis

Linear Regression Coefficient Analysis

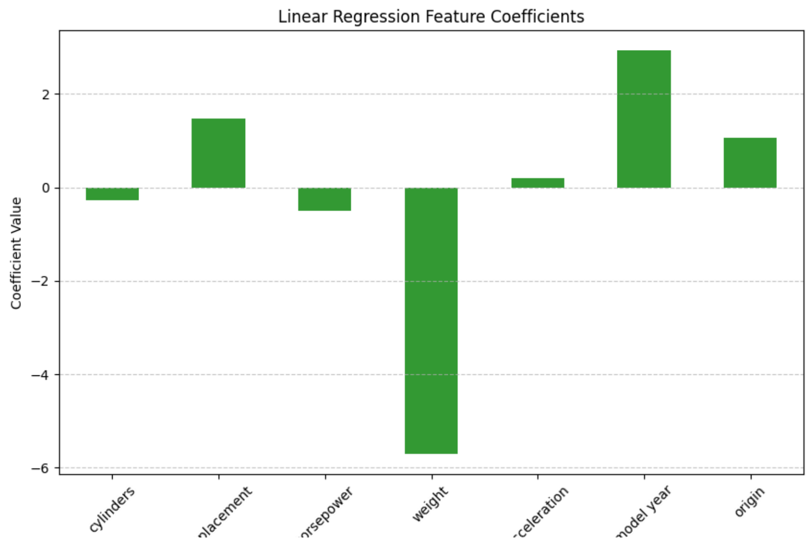


```python
# Linear Regression - Coefficients
linear_model = models["Linear Regression"]
linear_coefs = pd.Series(linear_model.coef_, index=data.drop(columns=['mpg']).columns)
```

The analysis examines the impact of each feature on **mpg** (miles per gallon) using the coefficients derived from the Linear Regression model.

Positive or negative coefficients explain how a particular feature influences fuel efficiency:

- **Positive coefficients:** Increase in the feature value leads to higher mpg (better fuel efficiency).
- **Negative coefficients:** Increase in the feature value leads to lower mpg (worse fuel efficiency).



This graph displays the coefficients of the features used in the Linear Regression model. It helps us understand the impact of each feature on the target variable (**mpg**). Here's a detailed explanation:

Graph Features

1. **X-Axis:** Independent variables (features) used in the model:
   - cylinders, displacement, horsepower, weight, acceleration, model year, origin.
2. **Y-Axis:** Coefficients of these variables. The magnitude and sign of the coefficients represent the impact of each feature on the target variable:
   - **Positive Coefficients:** An increase in the variable leads to an increase in mpg.
   - **Negative Coefficients:** An increase in the variable leads to a decrease in mpg.

Key Insights

1. **Most Influential Features:**
   - **Weight** (negative) and **model year** (positive) have the strongest effects on mpg.
     - Heavier vehicles significantly reduce mpg.
     - Newer model years improve mpg.
2. **Direction and Magnitude of Effects:**
   - Features with **positive coefficients** (displacement, model year, origin) tend to increase mpg.

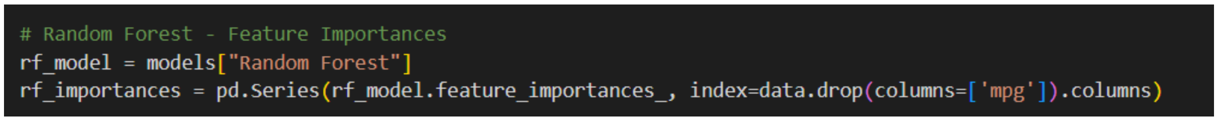
- Features with **negative coefficients** (weight, horsepower, cylinders) tend to decrease mpg.

3. **Limitations of Linear Regression:**
    - Coefficients represent only linear relationships. For instance, the effect of a variable like weight may not be strictly linear, but the linear model simplifies and represents it as such.

Conclusion

- **Model year** and **weight** are the most critical variables affecting mpg and play a significant role in the model.
- Features with **positive coefficients** represent factors that enhance fuel efficiency, while those with **negative coefficients** represent factors that diminish it.
- This graph is a crucial tool for identifying which variables increase or decrease the target variable, helping to interpret and refine the model.

Random Forest Feature Importance



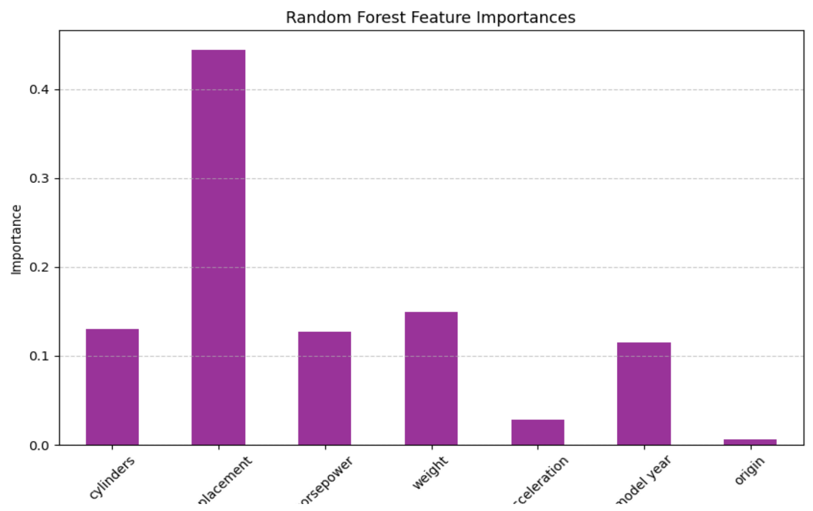
```
# Random Forest - Feature Importances
rf_model = models["Random Forest"]
rf_importances = pd.Series(rf_model.feature_importances_, index=data.drop(columns=['mpg']).columns)
```

This graph visualizes the **importance scores** of features, showing their contribution to the model's performance.

A **higher importance score** indicates that the feature is more effective in explaining the target variable (**mpg**).

Graph Description

- **Purpose:**
  The graph highlights the importance of each feature in the Random Forest Regressor model, helping us identify which variables have the most significant impact on mpg.
- **Key Observations:**
    - Features with higher importance scores contribute more to the model's predictions.
    - Features with lower scores have a minimal impact on the target variable.



Random Forest, with its ability to evaluate feature importance, has identified which variables have the most significant impact on the target variable.

Engine displacement, **weight**, and **model year** are the most influential features, indicating that focusing on these variables could enhance predictive performance.

4. Scatter Plot for Prediction Comparison

The goal is to visualize how well each model's predicted values align with the actual **mpg** values.

a) Linear Regression: Impact and Analysis

1. Linear Modeling:

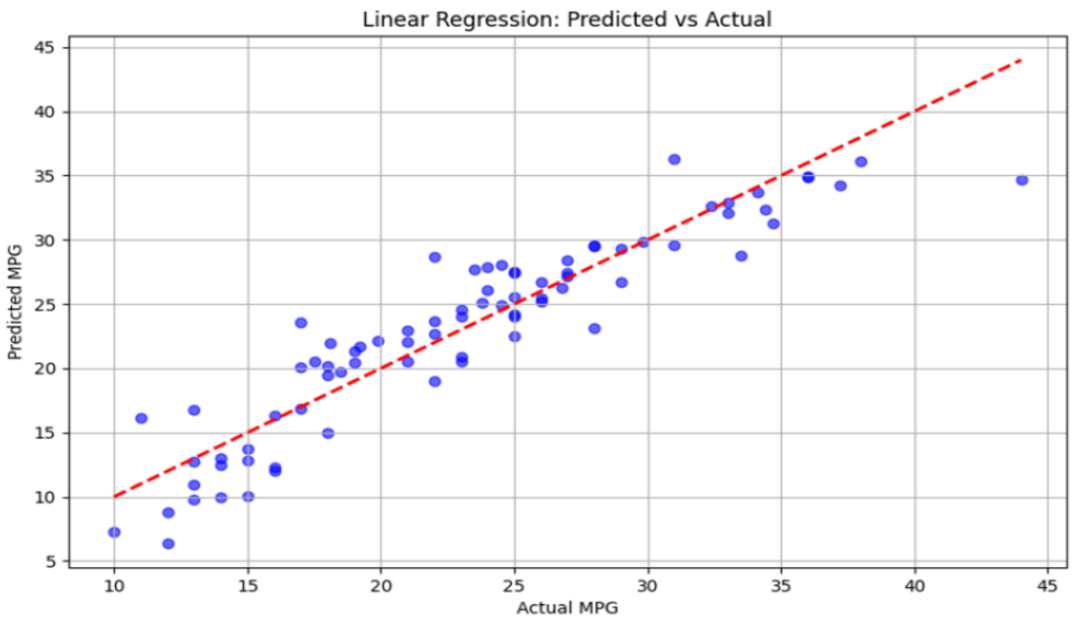
- **Linear Relationships:**
  Linear Regression can only model linear relationships. If the dataset contains non-linear relationships, this model falls short.
- **Performance on the Scatter Plot:**
  The plot shows that the model provides acceptable accuracy in the mid-range of **mpg** values but fails to capture complex patterns.

2. Generalization Ability:

- The model performs poorly at predicting extreme (low and high) values.
- This indicates the model's limited ability to generalize across the dataset.

Linear Regression demonstrates its strengths in handling linear data but highlights its weaknesses in capturing non-linear complexities.

```
# Scatter Plots for Predicted vs Actual Values

# (1) Linear Regression
plt.figure(figsize=(10, 6))
plt.scatter(y_test, predictions["Linear Regression"], alpha=0.6, color="blue")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', linewidth=2)
plt.title("Linear Regression: Predicted vs Actual")
plt.xlabel("Actual MPG")
plt.ylabel("Predicted MPG")
plt.grid(True)
plt.show()
```



Linear Regression Analysis

This graph visualizes the relationship between the predicted **mpg** values and the actual **mpg** values for the Linear Regression model. Below is a detailed analysis:

Graph Features

- **X-Axis:** Actual **mpg** (fuel efficiency) values.
- **Y-Axis:** Predicted **mpg** values by the model.
- **Red Dashed Line (y = x):** A reference line where predicted values are equal to actual values.

Observations

Performance of the Linear Regression Model:

- **Overall View:**
  The model is effective at predicting linear relationships but struggles with capturing non-linear patterns.
- **Accuracy in Mid-Range Values:**
  Predictions for **mpg** values in the mid-range (15–30) generally align closely with the red reference line.
- **Weakness in Extremes:**
  Predictions deviate in the low (10–15) and high (30–40) **mpg** ranges.
- **Outliers:**
  Significant prediction errors are observed, especially for higher **mpg** values.

Conclusion:

- **Strengths:** Linear Regression captures basic linear relationships in the dataset.
- **Weaknesses:** Deviations in extreme values and inability to model non-linear patterns clearly highlight its limitations.
- **Recommendation:** For capturing more complex patterns, models like **Ridge**, **Lasso**, or **Random Forest** should be used.

b) Ridge Regression Analysis

- **Effect of Regularization:**
  Ridge Regression uses **L2 regularization**, preventing overfitting by reducing the magnitude of the coefficients.
  - This makes the model more balanced and generalizable.
- **Performance:**
  - The graph demonstrates that regularization is effective, allowing the model to make reasonably accurate predictions.
  - High accuracy is observed for mid-range **mpg** values, indicating that regularization performs well in this range.
- **Impact on Extreme Values:**
  - Accuracy loss is observed at extreme values due to regularization reducing the influence of outliers.

Conclusion:

Ridge Regression provides a more balanced model compared to Linear Regression. However, its regularization effect can slightly diminish accuracy for extreme values while maintaining solid performance in the mid-range.

b) Ridge Regression

Prevents Overfitting:
Ridge Regression effectively prevents overfitting by using **L2 regularization**, which penalizes large coefficients.

Balances and Generalizes:
By shrinking the coefficients, it creates a more balanced and generalizable model.

Performance Visualization:
The graph shows that regularization is effective, and the model provides reasonably accurate predictions.

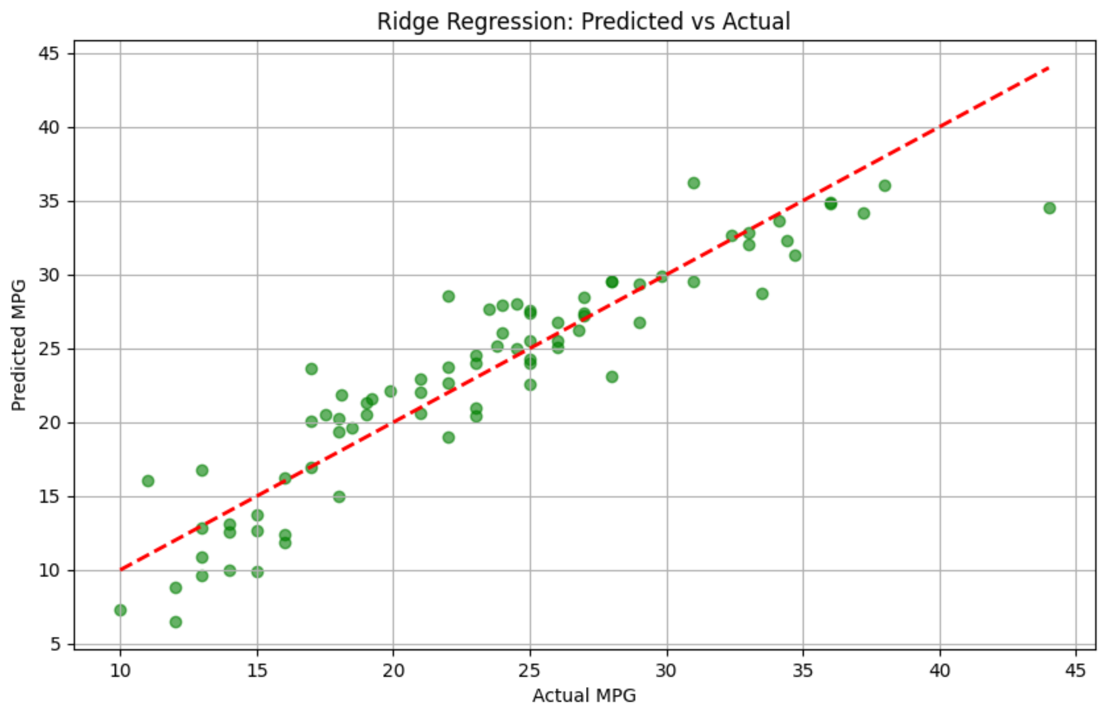
Observations:

- **High Accuracy in Mid-Range Values:**
  - Predictions for mid-range **mpg** values are highly accurate, indicating that regularization performs well in this range.
- **Accuracy Loss in Extreme Values:**
  - Regularization slightly reduces accuracy for extreme values, as it minimizes the influence of outliers.

Conclusion:

Ridge Regression strikes a balance between bias and variance, ensuring stability in predictions. While it performs well for most values, the reduced emphasis on extreme values due to regularization highlights its trade-off for generalization.

```python
# (2) Ridge Regression
plt.figure(figsize=(10, 6))
plt.scatter(y_test, predictions["Ridge Regression"], alpha=0.6, color="green")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', linewidth=2)
plt.title("Ridge Regression: Predicted vs Actual")
plt.xlabel("Actual MPG")
plt.ylabel("Predicted MPG")
plt.grid(True)
plt.show()
```



This graph visualizes the relationship between the predicted **mpg** values and the actual **mpg** values for the Ridge Regression model. A detailed analysis is provided below:

Graph Features

- **X-Axis:** Actual **mpg** (fuel efficiency) values.
- **Y-Axis:** Predicted **mpg** values by the model.
- **Red Dashed Line (y = x):** A reference line where predicted values perfectly match the actual values.

Observations

Performance of Ridge Regression Model:

- **Predictions vs. Actual Values:**
  Most points are close to the reference line, indicating that the model generally makes accurate predictions.
- **Accuracy in Mid-Range Values:**
  The model performs well for **mpg** values in the mid-range (15–30), showing high accuracy in this interval.
- **Weakness in Extreme Values:**
  Deviations are observed for low (**below 10**) and high (**above 35**) **mpg** values.

Performance Evaluation

- **Strengths:**
  - Ridge Regression successfully prevents overfitting through regularization and provides balanced prediction power.
  - It achieves strong performance in the mid-range of **mpg** values.
- **Limitations:**
  - Some loss in accuracy is seen for extreme values due to the regularization effect reducing the influence of outliers.

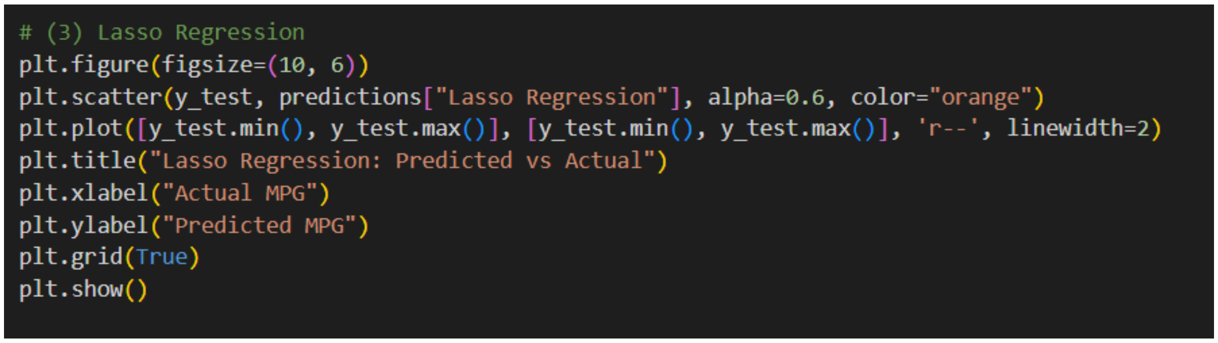
Conclusion

- Ridge Regression prevents overfitting while generally modeling the target variable accurately.
- Although some accuracy losses are seen in extreme values, the model demonstrates strong performance in the mid-range.
- This graph highlights Ridge Regression's ability to capture overall patterns in the dataset and deliver a generalizable model.

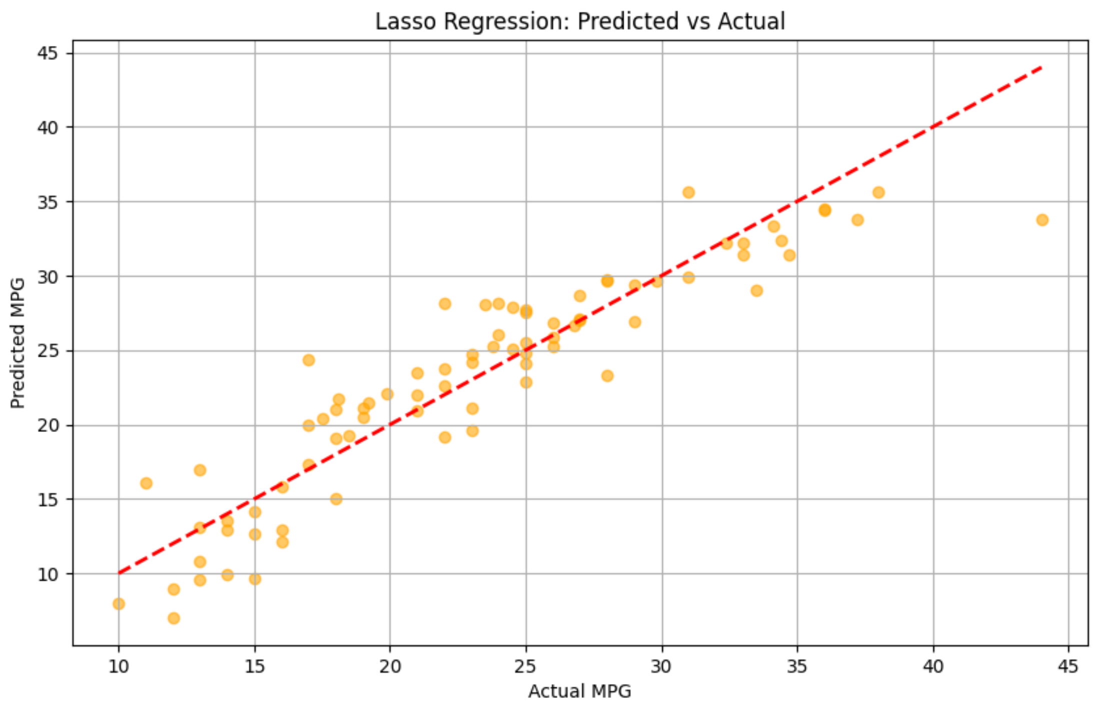
c) Lasso Regression

Lasso Regression reduces the coefficients of some features to zero, making the model simpler and more interpretable. This can lead to reduced performance at extreme values because certain features are excluded from the model.

- **Effect of Regularization:**
  The model's regularization selects only the most important features, creating a less complex model. However, this simplification may result in performance loss at extreme values due to the omission of some features.



```
# (3) Lasso Regression
plt.figure(figsize=(10, 6))
plt.scatter(y_test, predictions["Lasso Regression"], alpha=0.6, color="orange")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', linewidth=2)
plt.title("Lasso Regression: Predicted vs Actual")
plt.xlabel("Actual MPG")
plt.ylabel("Predicted MPG")
plt.grid(True)
plt.show()
```



Lasso Regression Analysis

This graph illustrates the relationship between the predicted **mpg** values and the actual **mpg** values for the Lasso Regression model. A detailed analysis is provided below:

Graph Features

- **X-Axis:** Actual **mpg** (fuel efficiency) values.
- **Y-Axis:** Predicted **mpg** values by the model.
- **Red Dashed Line (y = x):** Reference line where predicted values perfectly match actual values.

Observations

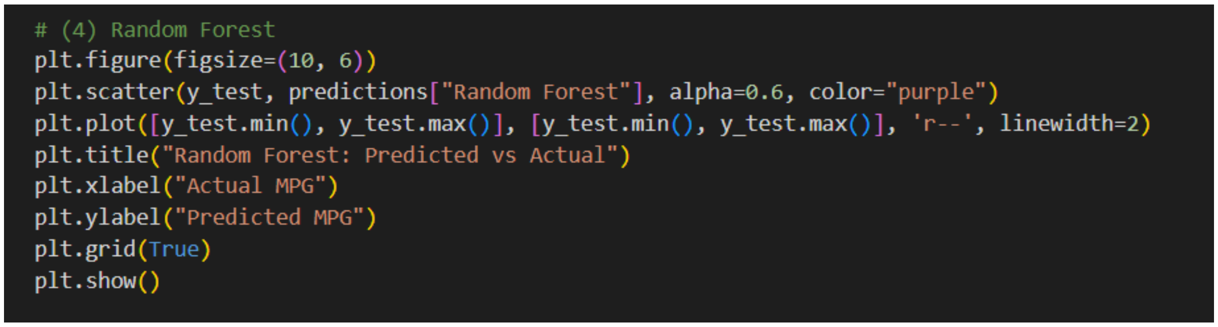
Performance of Lasso Regression Model:

1. **Prediction Accuracy:**
    - Most points are close to the red dashed line, indicating reasonable prediction accuracy for the Lasso Regression model.
    - However, deviations (outliers) are observed, particularly for low (**10–15**) and high (**30–40**) **mpg** values.
2. **Success in Mid-Range Values:**
    - The model performs better for mid-range **mpg** values (**15–30**), showing higher accuracy in this interval.
3. **Outliers:**
    - Deviations at low and high **mpg** values suggest the model struggles to represent data adequately in these ranges.
    - This could be a result of L1 regularization, which excludes some features.

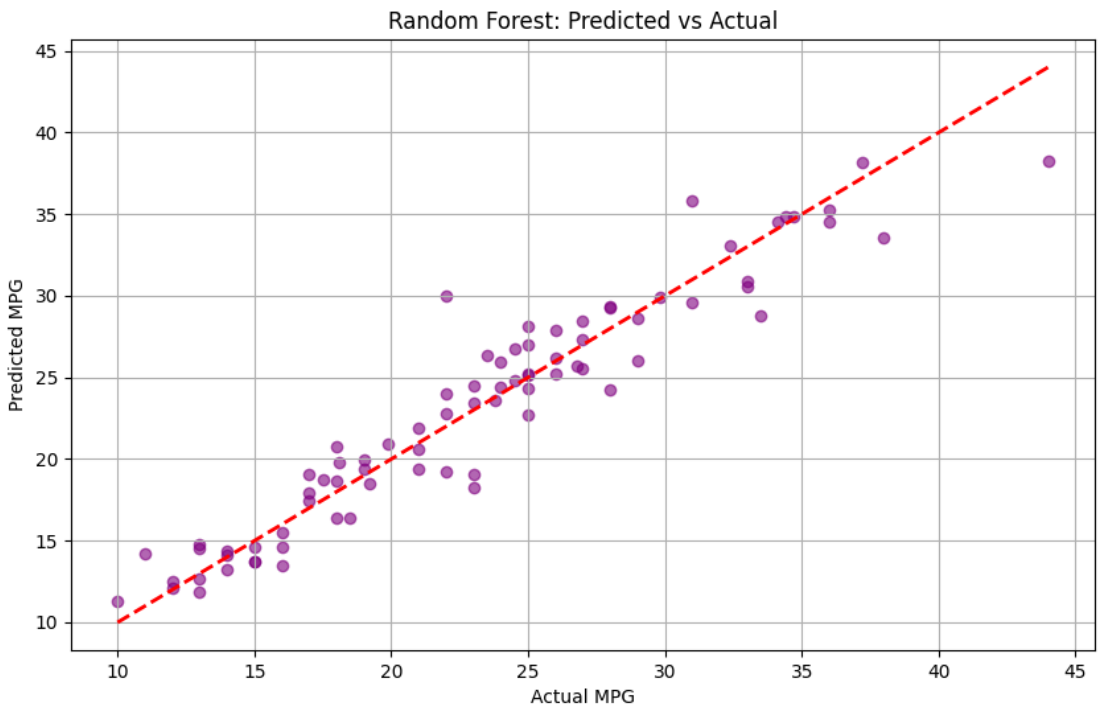
Conclusion

- **Strengths:**
    - Lasso Regression creates a simpler model through regularization and generally performs better than Linear Regression.
- **Limitations:**
    - The model has limited performance for extreme values, likely due to L1 regularization excluding certain features.
- **Insight from the Graph:**
    - While the model demonstrates good overall accuracy, improvements could be made to handle extreme values more effectively.

d) Random Forest



```
# (4) Random Forest
plt.figure(figsize=(10, 6))
plt.scatter(y_test, predictions["Random Forest"], alpha=0.6, color="purple")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', linewidth=2)
plt.title("Random Forest: Predicted vs Actual")
plt.xlabel("Actual MPG")
plt.ylabel("Predicted MPG")
plt.grid(True)
plt.show()
```



This graph illustrates the relationship between the predicted **mpg** values (Predicted MPG) and the actual **mpg** values (Actual MPG) for the Random Forest Regressor model. A detailed analysis is provided below:

Graph Features

- **X-Axis:** Actual **mpg** (fuel efficiency) values.
- **Y-Axis:** Predicted **mpg** values by the model.
- **Red Dashed Line (y = x):** Reference line where predicted values perfectly match actual values.

Performance of the Random Forest Regressor Model

- **Predictions vs. Actual Values:**
  Most points are close to the reference line, indicating that the model's predictions are highly accurate. The model effectively captures both linear and non-linear patterns in the dataset.
- **Success in Mid-Range Values:**
  - The model performs exceptionally well for mid-range **mpg** values (**15–30**), where a dense clustering of points near the reference line is observed.
- **Performance in Extreme Values:**
  - Minor deviations are seen for low (**below 10**) and high (**above 35**) **mpg** values, suggesting slightly less accurate representation in these ranges.
- **Outliers:**
  - A few points are farther from the reference line, likely due to insufficient representation of extreme values in the dataset.

Performance Evaluation

- **Overall Performance:**
  - The concentration of points near the reference line demonstrates the model's high accuracy and strong generalization ability.
- **Prediction Errors:**
  - While the overall error rate is low, slightly larger deviations are observed for extreme values.

Conclusion

The Random Forest Regressor successfully models both linear and non-linear relationships, delivering the best overall performance. It provides highly accurate results for mid-range **mpg** values and shows minor accuracy losses for extreme values. The model stands out in terms of prediction accuracy and generalization.

8. Evaluation of Results

Among the performance evaluation charts, the **Random Forest Regressor** demonstrated the best overall performance. The reasons for this conclusion and comparisons with other models are detailed below:

Random Forest Regressor (Best Model)

- **Prediction Accuracy:**
  The points are very close to the reference line (**y = x**), indicating that the model's predictions closely match the actual values.
- **Linear and Non-Linear Relationships:**
  Random Forest effectively captured non-linear patterns and made accurate predictions even for extreme values.
- **Deviations:**
  Prediction deviations are significantly smaller compared to other models, which aligns with a lower **MSE** and higher **R^2** score.

Ridge Regression

- **Strength of Regularization:**
  Ridge Regression outperforms Linear Regression due to its use of **L2 regularization**.
- **Success in Mid-Range Values:**
  It made accurate predictions for mid-range **mpg** values (**15–30**).
- **Weakness in Extreme Values:**
  Deviations are more noticeable for very low or very high **mpg** values.
- **Overall Performance:**
  While it couldn't capture complex patterns as effectively as Random Forest, Ridge Regression provided a more generalizable model compared to Linear Regression.

Lasso Regression

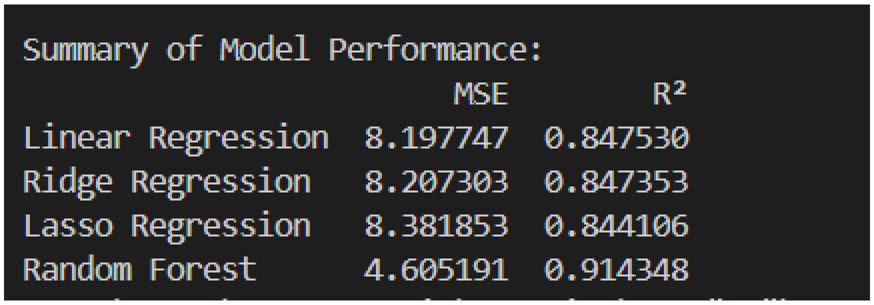
- **Similar to Ridge Regression:**
  Lasso Regression performed similarly to Ridge due to its use of **L1 regularization**.
- **Success in Mid-Range Values:**
  The model performed well in the mid-range **mpg** values.
- **Deviations in Extreme Values:**
  Larger deviations were observed in extreme values due to regularization excluding some features.
- **Overall Performance:**
  Lasso Regression showed slightly lower performance compared to Ridge Regression.

Linear Regression

- **Limited to Linear Relationships:**
  Linear Regression is sufficient for capturing linear relationships but failed to model the complex patterns in the dataset.
- **Performance in Mid-Range Values:**
  The model demonstrated acceptable accuracy for mid-range **mpg** values.
- **Weakness in Extreme Values:**
  Prediction deviations were significantly higher for extreme values.
- **Overall Performance:**
  The inability to capture complex patterns led to overall low performance.

Conclusion

- **Best Model:** Random Forest Regressor provided the most accurate predictions, capturing both linear and non-linear relationships effectively.
- **Ridge Regression:** A strong regularized model that balances generalization and performance, though less effective than Random Forest.
- **Lasso Regression:** Similar to Ridge but slightly less robust, especially for extreme values.
- **Linear Regression:** Suitable for simple datasets but unable to handle complex patterns, leading to lower overall performance.



```
Summary of Model Performance:
                         MSE        R²
Linear Regression   8.197747  0.847530
Ridge Regression    8.207303  0.847353
Lasso Regression    8.381853  0.844106
Random Forest       4.605191  0.914348
```

Comparison of Model Performance

1. Mean Squared Error (MSE):

- **Definition:**
  MSE measures the average of the squared differences between predicted and actual values. Lower MSE indicates better performance.
- **Random Forest MSE:**
  - Achieved the lowest MSE value of **4.605191**, indicating the smallest prediction error.
  - Significantly outperforms the other models, whose MSE values are around **8**.

2. R² (Coefficient of Determination):

- **Definition:**
  R² measures how well the model explains the variance of the target variable. Values closer to **1** indicate better performance.
- **Random Forest R² Score:**
  - Achieved the highest R² score of **0.914348**, demonstrating superior ability to capture relationships in the data and explain the majority of the target variable's variance.
- **Other Models:**
  - **Linear Regression:** 0.847530
  - **Ridge Regression:** 0.847353
  - **Lasso Regression:** 0.844106
  - These models show similar R² scores but fall short compared to Random Forest.

General Evaluation

- **Random Forest:**

- Outperformed all models with the lowest MSE and highest R^2 scores.
  - Successfully captured both linear and non-linear relationships, delivering exceptional performance.
- **Ridge and Lasso Regression:**
  - Regularization improved generalizability compared to Linear Regression.
  - However, these models were less effective than Random Forest.
- **Linear Regression:**
  - As a simple model, it captured basic linear relationships but failed with more complex patterns.

Conclusion

- **Best Model:**
  Performance metrics confirm that **Random Forest Regressor** is the best model for this dataset.
- **Key Strengths:**
  - Lower error rate (**MSE**).
  - Higher explanatory power (**R^2**).
- Random Forest demonstrated clear superiority over other models, making it the most suitable choice for this analysis.