**RECEP TAYYIP ERDOGAN UNIVERSITY**

**FACULTY OF ENGINEERING AND ARCHITECTURE**

**COMPUTER ENGINEERING DEPARTMENT**

# Data Mining

**2024-2025**

**Student Name Surname**

Yıldıray Karasubaşı

**Student No**

201401031

**Topic**

Using and Analyzing Data Mining Methods

**Data Sets Used**

Diabetes Dataset
Mall Customers Dataset
Auto Mpg Dataset

**Applied Methods**

Decision Tree, Random Forest and SVM

K-Means Clustering

Linear Regression and Random Forest Regression

**Instructor**

Doctor Lecturer Abdulgani Kahraman

**RİZE**

**2024**

## 1. Introduction

Data mining is a very important field today, allowing us to analyze data and make inferences based on this analysis. This report covers basic data mining techniques. It examines which techniques and how to use them to extract valuable information from large and complex datasets. The report examines 3 main methods and evaluates their performance on different datasets.

First, classification techniques such as decision trees are used to predict a categorical target variable. In this process, the data is cleaned, models are tested and their accuracy is compared. Then, clustering is performed with the "K-Means" algorithm to identify natural groupings within the data. Finally, linear regression and more advanced regression models are analyzed to predict continuous target variables.

The report details the datasets used, describes the methods step by step, summarizes how the results were measured and presents the findings. Plots are used to display the results and comparisons are made to highlight important points. Overall, this report serves as a guide to understanding both the theory and practice of data mining.

## 2. Datasets Used

The report examines three datasets, each processed using distinct data mining techniques. Details about the datasets and their characteristics are outlined below:

### 2.1. Diabetes Dataset

**Source:** https://www.kaggle.com/datasets/akshaydattatraykhare/diabetes-dataset/data

**Description:** This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether a patient has diabetes, based on certain diagnostic measurements included in the dataset.

**Target Variable(Outcome):**

- Target:
  - 1: Diabetic
  - 0: Not Diabetic

**Purpose:** To predict the target varible using decision tree and other classification algorithms.

**Details:**

- **Total Observations:** 768
- **Number of Columns:** 9
- **Independent Variables:**
  - **Pregnancies:** The number of pregnancies the individual has had.
  - **Glucose:** Plasma glucose concentration after a 2-hour oral glucose tolerance test.
  - **BloodPressure:** Diastolic blood pressure (mm Hg).
  - **SkinThickness:** Triceps skinfold thickness (mm).
  - **Insulin:** 2-hour serum insulin (mu U/ml).
  - **BMI:** Body Mass Index, calculated as weight in kg/(height in m)^2.
  - **DiabetesPedigreeFunction:** A function that scores the likelihood of diabetes based on family history.
  - **Age:** Age of the individual (years).
  - **Outcome:** Binary variable indicating whether the individual has diabetes (1) or not (0).

## 2.2. Mall Customers Dataset

**Source:** https://www.kaggle.com/datasets/simtoor/mall-customers

**Description:** This dataset contains information on customers, including demographic and behavioral attributes. It includes data such as age, gender, annual income, and spending scores, making it suitable for analyzing customer behavior and segmentation.

**Target Variable:** None (Clustering analysis does not include a target variable.)

**Purpose:** To perform customer segmentation using the K-Means algorithm.

**Details:**

- **Total Observations:** 200
- **Number of Columns:** 5
- **Independent Variables:**
  - **CustomerID:** Unique identifier for each customer.
  - **Genre:** Gender of the customer (Male/Female).

    - **Age:** Age of the customer (in years).
    - **Annual Income (k$):** Annual income of the customer (in thousands of dollars).
    - **Spending Score (1-100):** Score assigned to the customer based on their spending behavior and habits (1 = lowest, 100 = highest).

### 2.3. Auto Mpg Dataset

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

**Description:** This dataset provides information about fuel efficiency and car features for various automobiles manufactured between 1970 and 1982.

**Target Variable:**

- **mpg:** Miles per gallon, indicating the fuel efficiency of a car.

**Purpose:** To predictfuel efficiency (mpg) using various car-related features through regression techniques.

**Details:**

- **Total Observations:** 398
- **Number of Columns:** 9
- **Independent Variables:**
    - **Mpg:** Miles per gallon, a measure of the vehicle's fuel efficiency (higher values indicate better fuel efficiency).
    - **Cylinders**: Number of cylinders in the car.
    - **Displacement:** Engine displacement (in cubic inches).
    - **Horsepower:** Engine horsepower.
    - **Weight:** Vehicle weight (in lbs).
    - **Acceleration:** Time taken to accelerate from 0 to 60 mph (in seconds).
    - **Model year:** Year of manufacture (e.g., 70 = 1970).
    - **Origin:** Region of origin (1 = USA, 2 = Europe, 3 = Japan).
    - **Car name:** Name and model of the car (categorical).

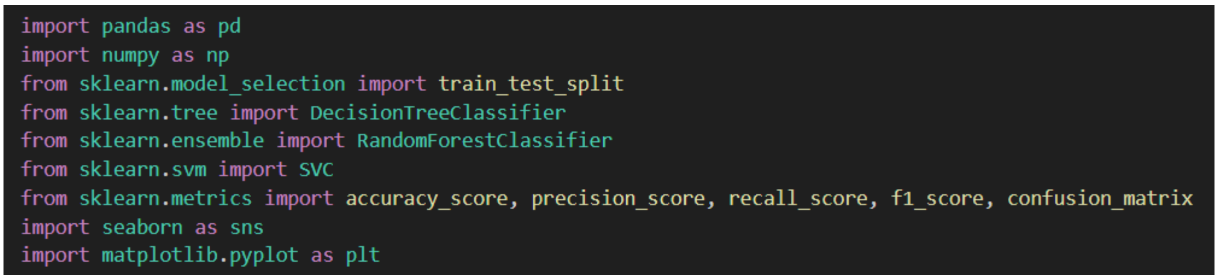
## 3. Question 1: Decision Tree and Comparative Analysis

### 3.1 Introduction

The goal of this project is to use machine learning methods to predict a target variable. Specifially, a decision tree model will be employed to make predictions, and its performance will be assessed. Additionally, the results of the decision tree will be compared with another machine learning method, the Random Forest model, to evaluate its effectiveness.

### 3.2. Importing Necessary Libraries

The necessary libraries which I imported for this task are shown below:



```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

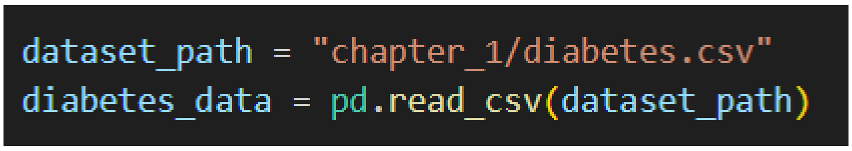
- **pandas(pd):** It is a powerful library used for data analysis and processing in Python. It facilitates operations such as data manipulation, filtering, transformation, statistical analysis and missing data management by providing DataFrame and Series structures, which are structured data types.
- **numpy(np):** It is used for numerical calculations and working with multidimensional arrays. Matrix operations play an important role in linear algebra, random numbers, and scientific calculations.
- **sklearn.model_selection.train_test_split:** This library used for separate data into training and testing data. It is very important to correctly separate data sets for training and testing machine learning models.
- **sklearn.tree.DecisionTreeClassifier:** It is a supervised learning algorithm that classifies data by dividing it according to features and uses a hierarchical tree structure.
- **sklearn.ensemble.RandomForestClassifier:** It is an ensemble learning algorithm that classifies by majority voting using multiple decision trees.

- **sklearn.svm(SVC):** It is used to implement the Support Vector Machines (SVM) algorithm. Can learn more complex decision boundaries.

- **sklearn.metrics:** It is a module that provides metrics and tools to evaluate model performance. The metrics are shown below:
    - **accuracy_score:** It measures the accuracy rate of the model.
    - **precision_score:** It measures the accuracy of the positive predictions.
    - **recall_score:** It measures the rate of the true positive.
    - **f1_score:** It calculates the harmonic mean of Precision and Recall.
    - **confusion_matrix:** It summarizes the difference between actual values and predicted values of the model.
- **seaborn(sns):** It is a library based on Matplotlib used for statistical data visualization in Python. It allows you to create impressive graphics easily.
- **matplotlib.pyplot(plt):** It is a submodule of the Matplotlib library used for data visualization in Python and provides a simple interface for creating graphs and plots.

## 3.3. Data Preprocessing

### 3.3.1. Loading and Preparing the Data

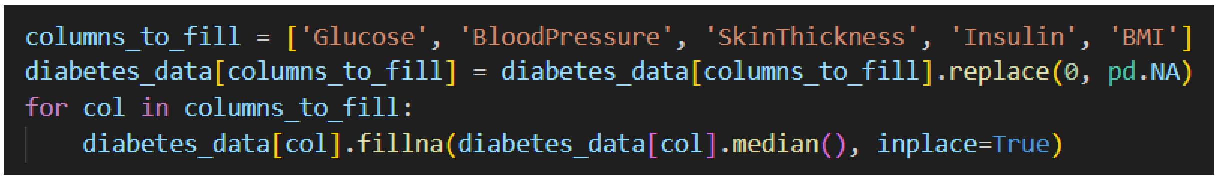


```
dataset_path = "chapter_1/diabetes.csv"
diabetes_data = pd.read_csv(dataset_path)
```

I downloaded the dataset from Kaggle. I then created a variable called "dataset_path" and assigned the file path of the dataset to this variable. Finally, I created a variable called "diabetes_data" and included this data set in my code using “read_csv()” function.

### 3.3.2. Handling missing Values

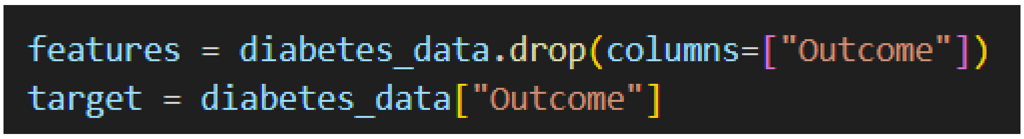
There was missing data in the dataset, so I replaced the missing values with the median values of the columns.



```python
columns_to_fill = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
diabetes_data[columns_to_fill] = diabetes_data[columns_to_fill].replace(0, pd.NA)
for col in columns_to_fill:
    diabetes_data[col].fillna(diabetes_data[col].median(), inplace=True)
```

1) First, I identified the columns with potential missing or invalid values.
2) Then, I looped through each column to address these missing or invalid values.
3) For each column, I replaced zero values with NaN to mark them as missing.
4) After that, I calculated the median value for each column.
5) Finally, I filled the missing values in each column using the median value. This process ensured that all missing or invalid values were replaced, and the dataset became consistent for further analysis.

### 3.3.3. Feature Selection and Engineering

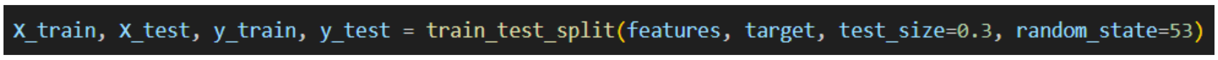


```python
features = diabetes_data.drop(columns=["Outcome"])
target = diabetes_data["Outcome"]
```

1) First, when training the model, I only needed to choose the variable I would use as input. So I got all the columns in my dataset but not the "Outcome" column. I assigned this operation to a variable called "features".
2) I then created a variable called "target" and just selected the "Outcome" column. The reason for this is to determine the variable that the model should predict.

## 3.4. Modeling

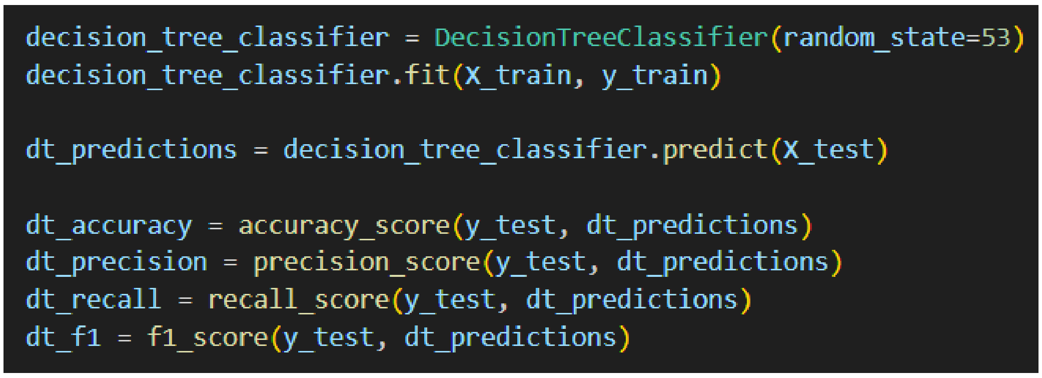
### 3.4.1. Splitting Data



```
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.3, random_state=53)
```

- The dataset divided into:
    - 70% training data for the model to learn.
    - 30% test data for evaluating the model’s performance.
- The splitting has done using the train_test_split function with random_state=53 to ensure reproducibility.

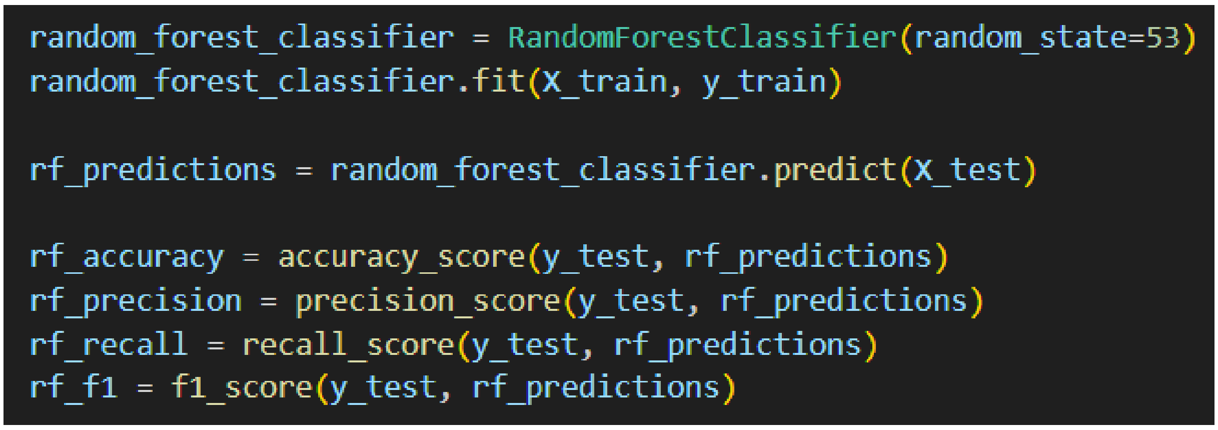
### 3.4.2. Decision Tree Classifier



```
decision_tree_classifier = DecisionTreeClassifier(random_state=53)
decision_tree_classifier.fit(X_train, y_train)

dt_predictions = decision_tree_classifier.predict(X_test)

dt_accuracy = accuracy_score(y_test, dt_predictions)
dt_precision = precision_score(y_test, dt_predictions)
dt_recall = recall_score(y_test, dt_predictions)
dt_f1 = f1_score(y_test, dt_predictions)
```

1) First, I created the decision tree classifier.
2) Then I trained the model. I predicted the input variables of the test data.
3) Finally, I evaluated the performance of the model (accuracy, precision, sensitivity, f1 score).

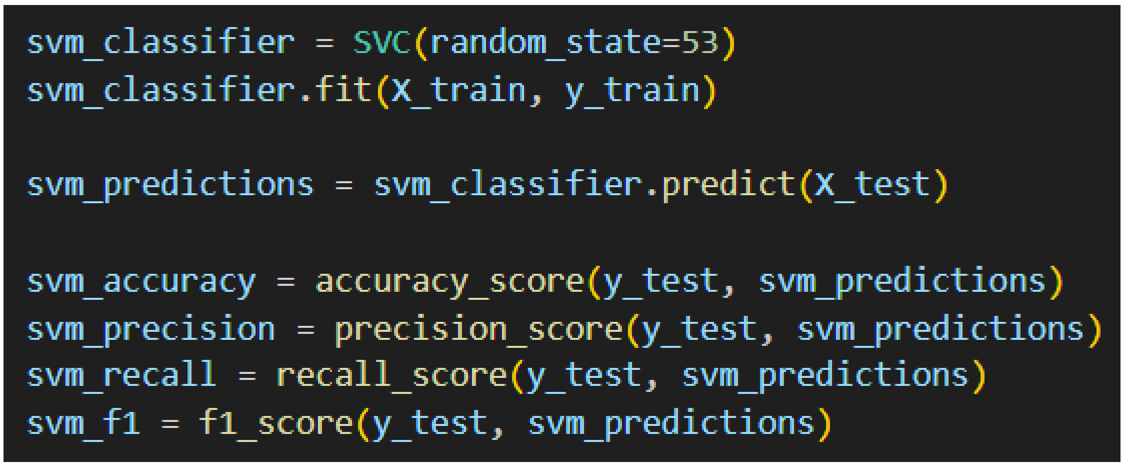
### 3.4.3 Random Forest Classifier



```
random_forest_classifier = RandomForestClassifier(random_state=53)
random_forest_classifier.fit(X_train, y_train)

rf_predictions = random_forest_classifier.predict(X_test)

rf_accuracy = accuracy_score(y_test, rf_predictions)
rf_precision = precision_score(y_test, rf_predictions)
rf_recall = recall_score(y_test, rf_predictions)
rf_f1 = f1_score(y_test, rf_predictions)
```

1) First, I created the random forest classifier.
2) Then I trained the model. I predicted the input variables of the test data.
3) Finally, I evaluated the performance of the model (accuracy, precision, sensitivity, f1 score).

### 3.4.4 Support Vector Machine Classifier

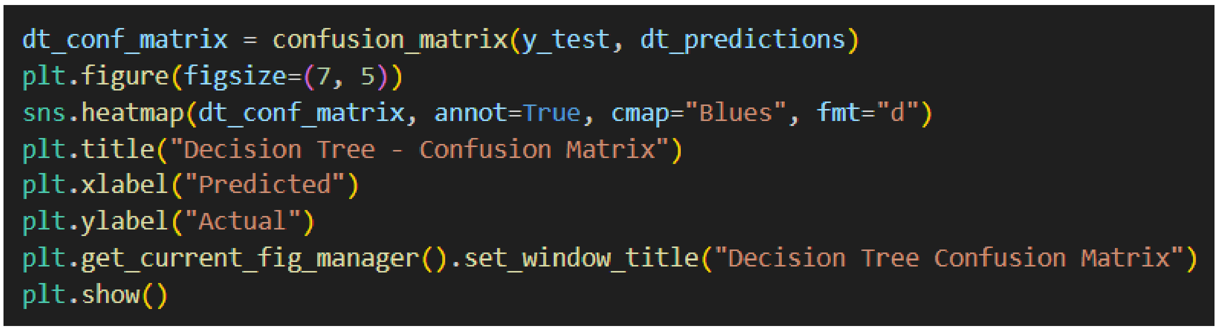


```
svm_classifier = SVC(random_state=53)
svm_classifier.fit(X_train, y_train)

svm_predictions = svm_classifier.predict(X_test)

svm_accuracy = accuracy_score(y_test, svm_predictions)
svm_precision = precision_score(y_test, svm_predictions)
svm_recall = recall_score(y_test, svm_predictions)
svm_f1 = f1_score(y_test, svm_predictions)
```

1) First, I created the SVM classifier.
2) Then, I trained the model and made predictions on the test data.
3) Finally, I evaluated the performance of the model (accuracy, precision, sensitivity, F1 score).

## 3.5. Plotting

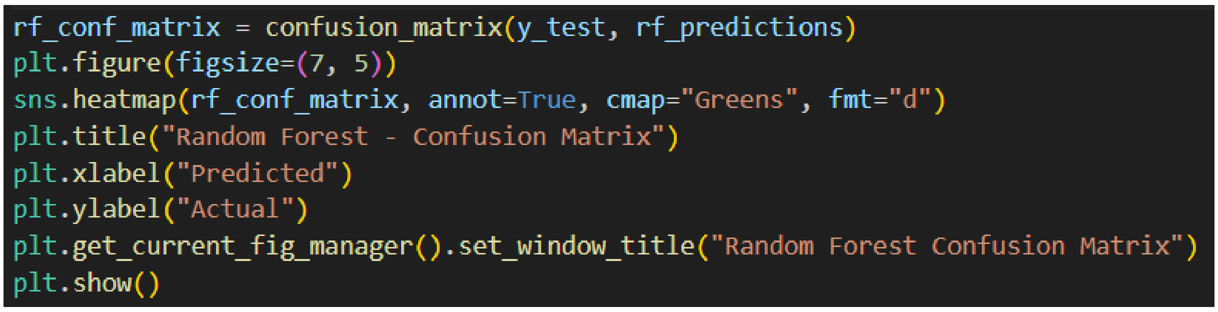
### 3.5.1 Decision Tree Confusion Matrix Visualization



```python
dt_conf_matrix = confusion_matrix(y_test, dt_predictions)
plt.figure(figsize=(7, 5))
sns.heatmap(dt_conf_matrix, annot=True, cmap="Blues", fmt="d")
plt.title("Decision Tree - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.get_current_fig_manager().set_window_title("Decision Tree Confusion Matrix")
plt.show()
```

1) First I calculated the confusion matrix. Using the “confusion_matrix” function, I created a confusion matrix of actual values and predicted values.
2) Then I visualized the matrix and also I colored the graph using the "sns.heatmap()" function.
3) I named the title of the chart and the x and y axes. Also I changed the name of the window.
4) Finally, I plotted the graph with the "show()" function.

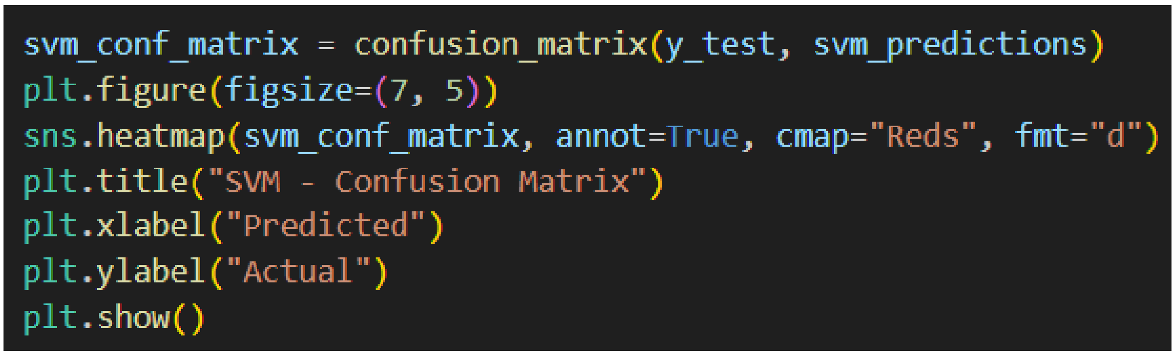
### 3.5.2 Random Forest Confusion Matrix Visualization



```python
rf_conf_matrix = confusion_matrix(y_test, rf_predictions)
plt.figure(figsize=(7, 5))
sns.heatmap(rf_conf_matrix, annot=True, cmap="Greens", fmt="d")
plt.title("Random Forest - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.get_current_fig_manager().set_window_title("Random Forest Confusion Matrix")
plt.show()
```

1) First I calculated the confusion matrix. Using the “confusion_matrix” function, I created a confusion matrix of actual values and predicted values.
2) Then I visualized the matrix and also I colored the graph using the "sns.heatmap()" function.
3) I named the title of the chart and the x and y axes. Also I changed the name of the window.
4) Finally, I plotted the graph with the "show()" function.

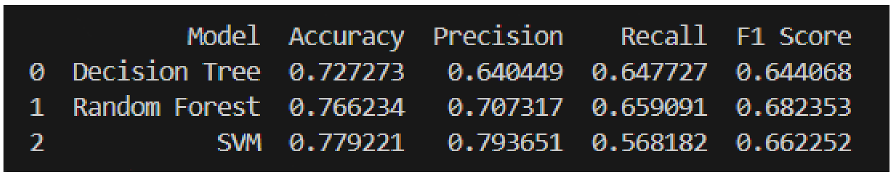
### 3.5.3 Support Vector Machine Confusion Matrix Visualization



```python
svm_conf_matrix = confusion_matrix(y_test, svm_predictions)
plt.figure(figsize=(7, 5))
sns.heatmap(svm_conf_matrix, annot=True, cmap="Reds", fmt="d")
plt.title("SVM - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

1) First I calculated the confusion matrix. Using the “confusion_matrix” function, I created a confusion matrix of actual values and predicted values.
2) Then I visualized the matrix and also I colored the graph using the "sns.heatmap()" function.
3) I named the title of the chart and the x and y axes. Also I changed the name of the window.
4) Finally, I plotted the graph with the "show()" function.

## 3.6. Performance Evaluation



```
           Model  Accuracy  Precision    Recall  F1 Score
0  Decision Tree  0.727273   0.640449  0.647727  0.644068
1  Random Forest  0.766234   0.707317  0.659091  0.682353
2            SVM  0.779221   0.793651  0.568182  0.662252
```

**Performance metrics**

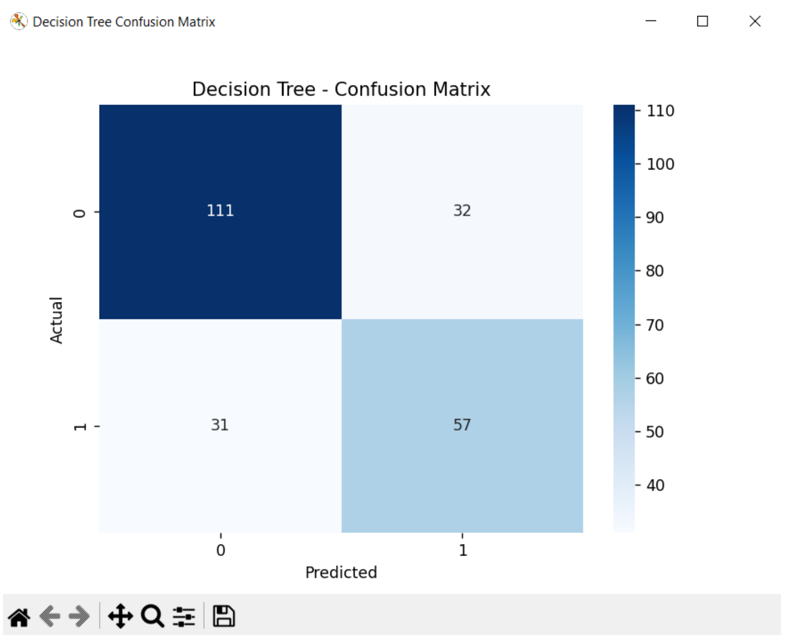
- **Accuracy:** Ratio of correctly predicted samples to total samples.
- **Precision:** The rate at which those predicted as positive are actually positive.
- **Recall:** How many true positive samples were predicted correctly.
- **F1 Score:** Harmonic mean of precision and sensitivity.

### 3.6.1. Decision Tree Performance

Two decision tree models were defined based on their accuracy, precision, recall, and F1 score across two selections. The outcomes for the first selection are detailed below:

| Metric | Value |
|---|---|
| Accuracy | 72.7% |
| Precision | 64.0% |
| Recall | 64.7% |
| F1 Score | 64.4% |

**Confusion Matrix for Decision Tree:**



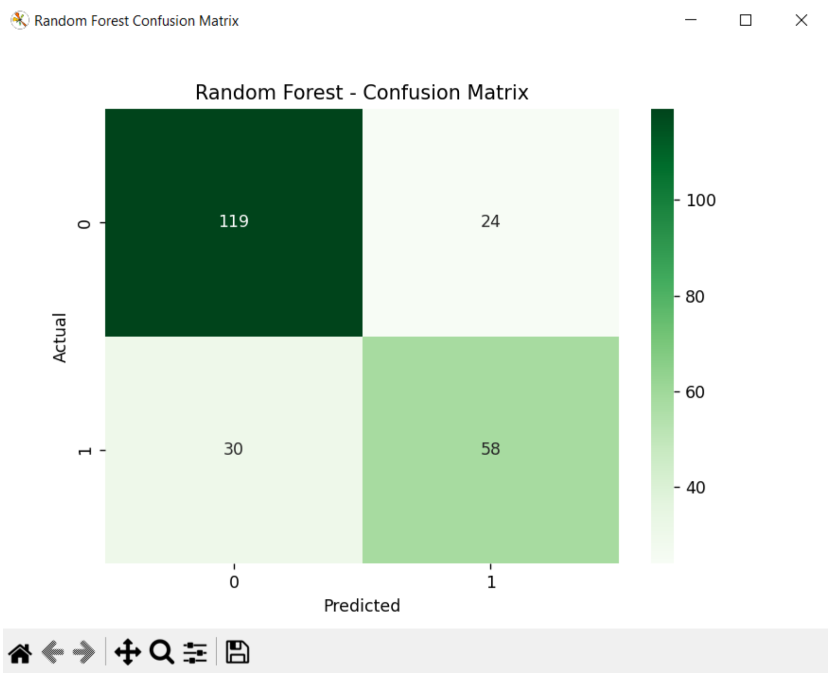
- **True Positives (Correctly predicted heart disease):** 57
- **True Negatives (Correctly predicted no heart disease):** 111
- **False Positives (Wrongly predicted heart disease):** 32
- **False Negatives (Wrongly predicted no heart disease):** 31

### 3.6.2. Random Forest Performance

The random forest classifier was employed to compare its performance against the decision tree model. The performance metrics for the first selection are as follows:

| Metric | Value |
|---|---|
| Accuracy | 76.6% |
| Precision | 70.7% |
| Recall | 65.9% |
| F1 Score | 68.2% |

**Confusion Matrix for Random Forest:**



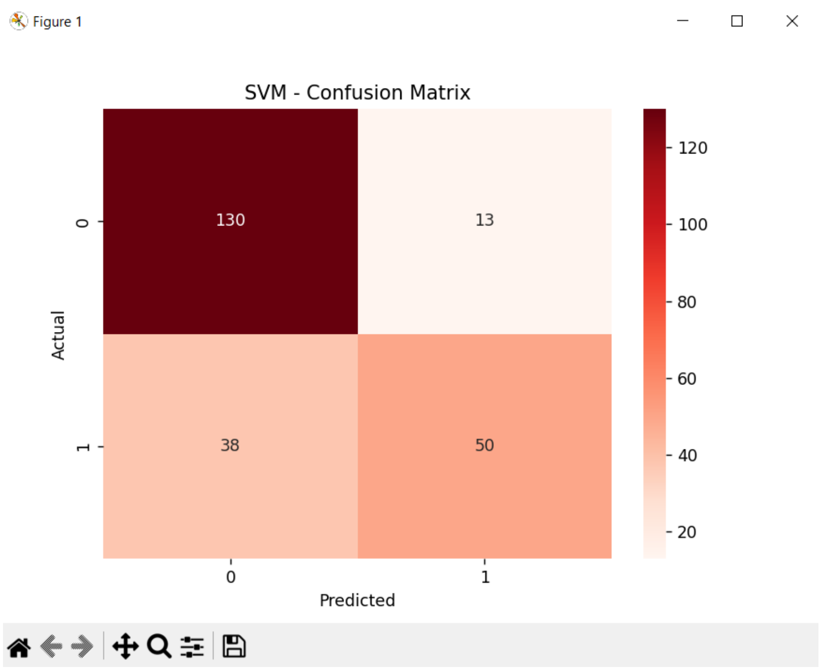
- **True Positives (Correctly predicted heart disease):** 58
- **True Negatives (Correctly predicted no heart disease):** 119
- **False Positives (Wrongly predicted heart disease):** 24
- **False Negatives (Wrongly predicted no heart disease):** 30

### 3.6.3. Support Vector Machine Performance

The performance of two SVM models was analyzed based on accuracy, precision, recall, and F1 score for two selections. The results for the first selection are provided below:

| **Metric** | **Value** |
|---|---|
| Accuracy | 77.9% |
| Precision | 79.3% |
| Recall | 56.8% |
| F1 Score | 66.2% |

**Confusion Matrix for SVM:**



- **True Positives (Correctly predicted heart disease):** 50
- **True Negatives (Correctly predicted no heart disease):** 130
- **False Positives (Wrongly predicted heart disease):** 13
- **False Negatives (Wrongly predicted no heart disease):** 38

## 3.7. Results and Discussion

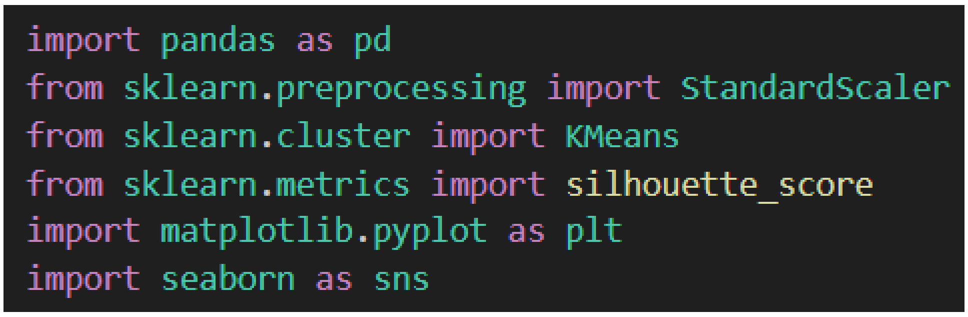
### 3.7.1. Comparison of Model Performance

The performance of all models is summarized in the table below:

| Metric | Decision Tree | Random Forest | SVM |
|---|---|---|---|
| Accuracy | 72.7% | 76.6% | 77.9% |
| Precision | 64.0% | 70.7% | 79.3% |
| Recall | 64.7% | 65.9% | 56.8% |
| F1 Score | 64.4% | 68.2% | 66.2% |

## 4. Question 2: K-Means Clustering

### 4.1 Importing Necessary Libraries

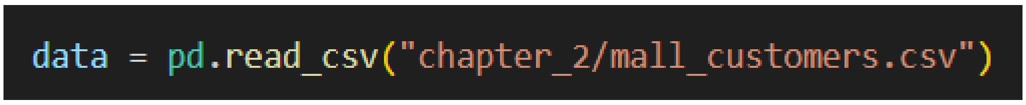


```python
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
import seaborn as sns
```

- **pandas(pd):** It is a powerful library used for data analysis and processing in Python. It facilitates operations such as data manipulation, filtering, transformation, statistical analysis and missing data management by providing DataFrame and Series structures, which are structured data types.
- **sklearn.cluster.KMeans:** It is an unsupervised learning algorithm that divides the data into k clusters.
- **sklearn.preprocessing.StandardScaler:** It is a class that scales data with a mean of 0 and a standard deviation of 1.
- **sklearn.metrics.silhouette_score:** Evaluates cluster quality on a scale of -1 to 1; A higher score indicates better clustering.
- **matplotlib.pyplot(plt):** It is a submodule of the Matplotlib library used for data visualization in Python and provides a simple interface for creating graphs and plots.
- **seaborn(sns):** It is a library based on Matplotlib used for statistical data visualization in Python. It allows you to create impressive graphics easily.

## 4.2. Data Preprocessing

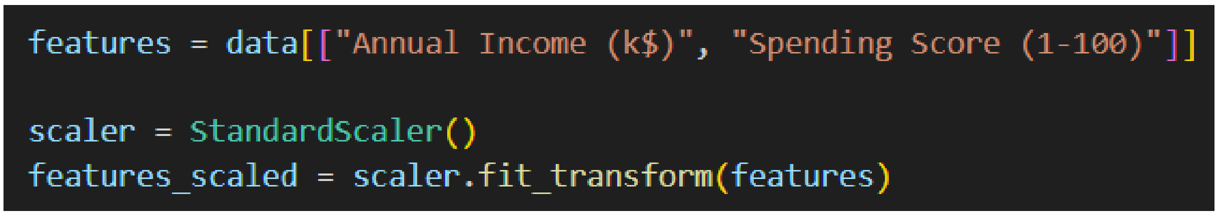
### 4.2.1 Loading and Preparing the Data



```
data = pd.read_csv("chapter_2/mall_customers.csv")
```

I downloaded my dataset called "mall_customers" from Kaggle. Then I included the dataset in my project using the "read_csv()" function.

### 4.2.2. Feature Standardization

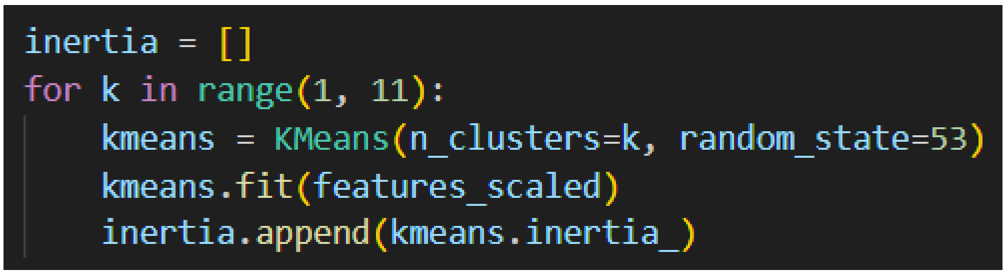


```
features = data[["Annual Income (k$)", "Spending Score (1-100)"]]

scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)
```

1) First, I selected the columns of the annual income and spending score and assigned them to the variable called "features".
2) Then I created a variable called "scaler" and standardized the data.
3) Finally, I created a variable called "features_scaled" and set the mean of the data in the "features" variable to 0 and the standard deviation to 1.

## 4.3. Elbow Method Calculation

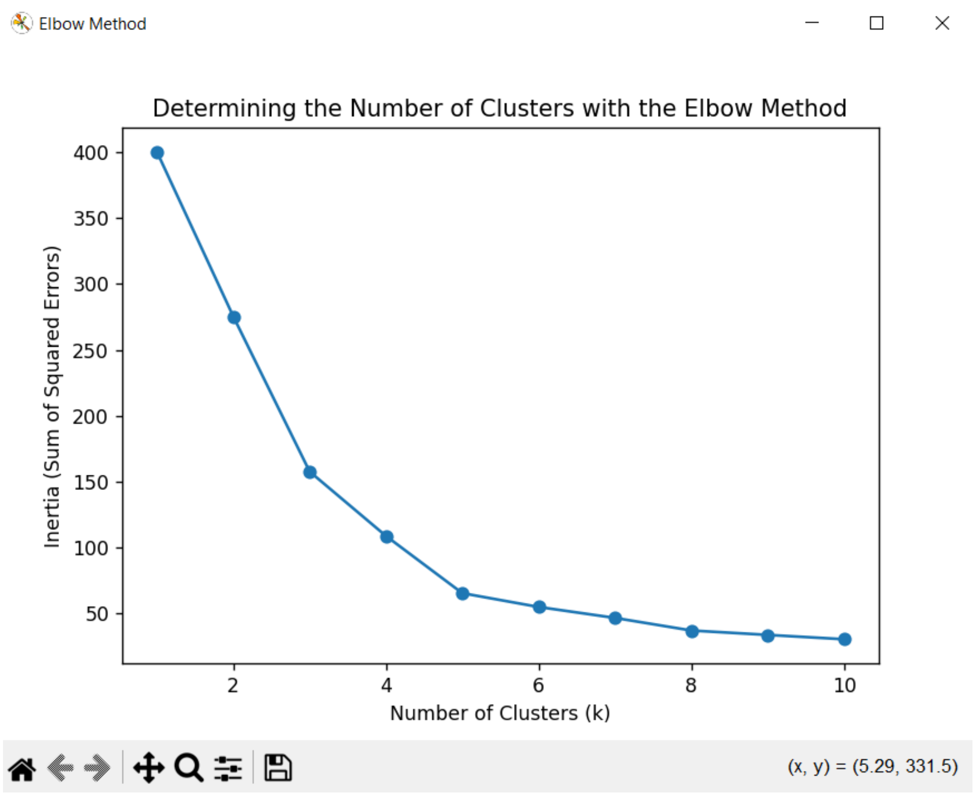
### 4.3.1. Choosing the Number of Clusters(k)



```
inertia = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=53)
    kmeans.fit(features_scaled)
    inertia.append(kmeans.inertia_)
```

In the code, I created a list called inertia and applied a K-Means model for different k values from 1 to 10 to determine the optimal number of clusters with the Elbow Method. I created the Elbow chart by adding the sum of squares of errors calculated for each k to the list and determined the most appropriate number of clusters.

**Elbow Method Graph:**

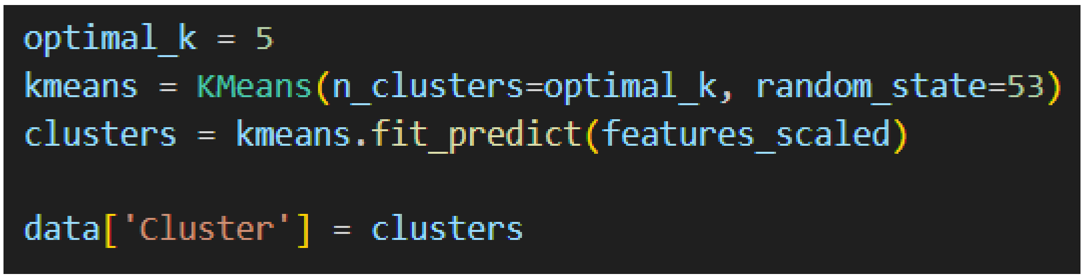


### 4.3.2. Elbow Graph Plotting

```
plt.figure(figsize=(7, 5))
plt.plot(range(1, 11), inertia, marker='o')
plt.title("Determining the Number of Clusters with the Elbow Method")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Inertia (Sum of Squared Errors)")
plt.get_current_fig_manager().set_window_title("Elbow Method")
plt.show()
```

1) First, I determined the size of the graphic I would create.
2) Then, I plotted the elbow chart with the "plt.plot()" function.
3) Then I named the title and x-y axes and also changed the window name to "Elbow Method".
4) Finally, I plotted the graph with the "show()" function.

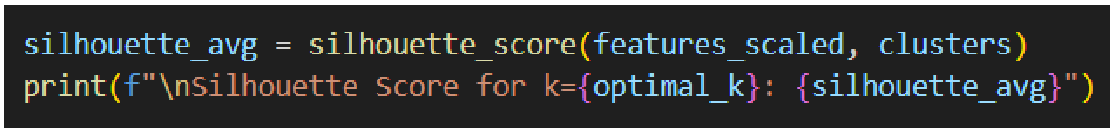
### 4.3.3. Optimal Clustering with K-Means



```python
optimal_k = 5
kmeans = KMeans(n_clusters=optimal_k, random_state=53)
clusters = kmeans.fit_predict(features_scaled)

data['Cluster'] = clusters
```

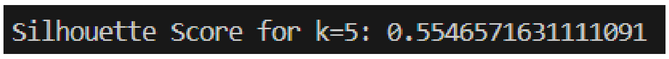
1) First, I determined the optimal number of clusters. (With the Elbow method, I determined that the optimal number of clusters was 5. That's why I assigned it to 5).
2) Then I defined the K-means model.
3) Then I created a variable called "clusters" and made a cluster prediction.
4) Finally, I named these created clusters.

### 4.3.4. Calculating The Silhouette Score and Printing It To The Console



```python
silhouette_avg = silhouette_score(features_scaled, clusters)
print(f"\nSilhouette Score for k={optimal_k}: {silhouette_avg}")
```

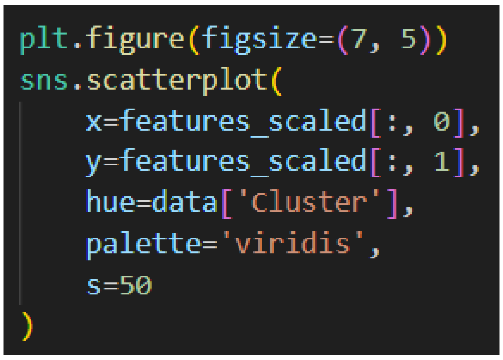
1) First, I calculated the silhouette score using the "silhouette_score()" function.
2) Then, I printed this score to the console with the "print()" function.



```
Silhouette Score for k=5: 0.5546571631111091
```

**- Silhouette Score:** It is a metric used to evaluate the quality of a clustering model. This score measures each data point's fit within its cluster and how far it differs from other clusters.

### 4.3.5. Visualization of K-Means Clustering Results



```python
plt.figure(figsize=(7, 5))
sns.scatterplot(
    x=features_scaled[:, 0],
    y=features_scaled[:, 1],
    hue=data['Cluster'],
    palette='viridis',
    s=50
)
```

1) First, I created the frame of the graphic I would plot and entered its dimensions.
2) Then, I created a scatter plot with the "sns.scatterplot()" function and assigned values to it.

### 4.3.6. Visualization of K-Means Clustering Results

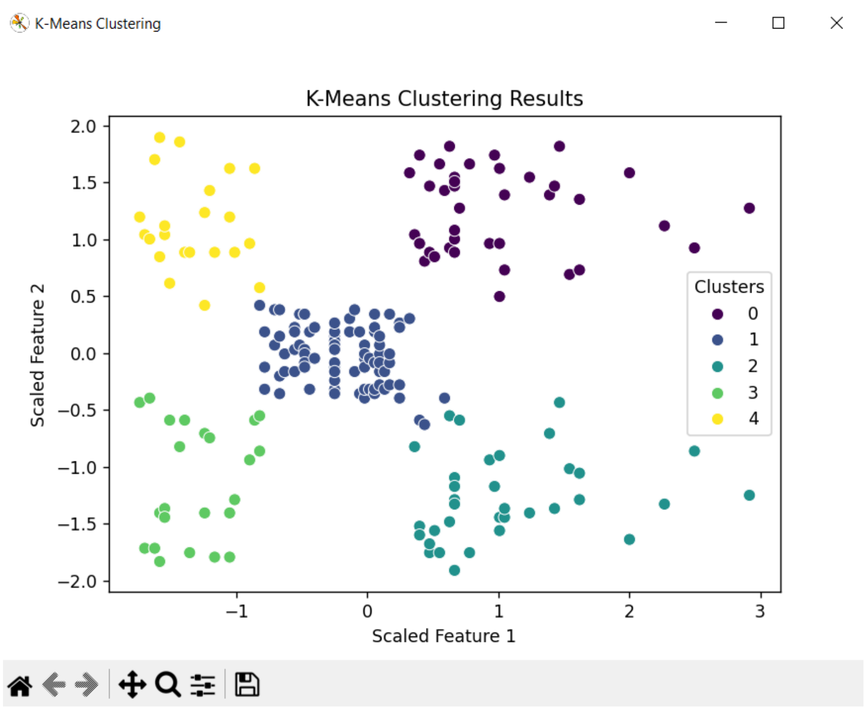
```
plt.title("K-Means Clustering Results")
plt.xlabel("Scaled Feature 1")
plt.ylabel("Scaled Feature 2")
plt.legend(title="Clusters")
plt.get_current_fig_manager().set_window_title("K-Means Clustering")
plt.show()
```

1) First I named the title of the chart and the x-y axes.
2) Then I added a legend to the chart.
3) Then I changed the window name to "K-Means Clustering".
4) Finally, I plotted the graph with the "show()" function.

### 4.3.7. Clustering Results

- The K-Means algorithm was used with k = 5 clusters.
- A scatter plot shows the clusters.
- Each dot is a customer, and each color is a different cluster.

**Cluster Results Scatter Plot:**



## 4.4. Performance Evaluation

**Silhouette Score:** The Silhouette score checks how good the clusters are. A higher score means better clustering.

- Silhouette Score for k = 5 → 0.554

| Metric | Value |
|---|---|
| Silhouette Score | 0.554 |

## 4.5. Results and Discussion

### 4.5.1. Understanding the Clusters

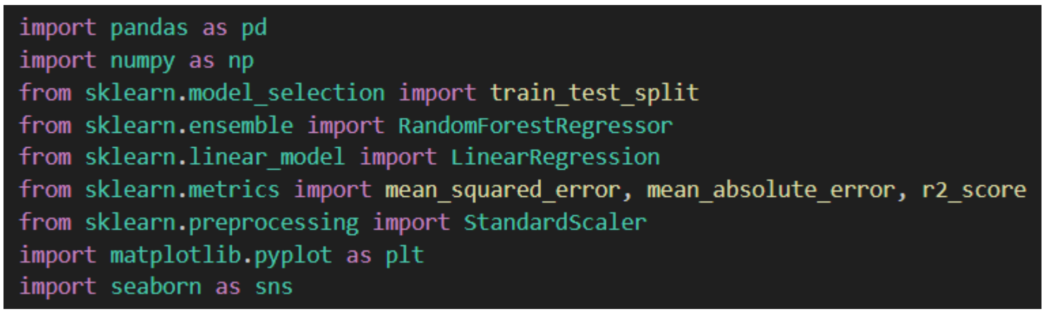
- Cluster 0: Customers with high spending and high income.
- Cluster 1: Customers with moderate spending and average income.
- Cluster 2: Customers with low income and low spending.
- Cluster 3: Customers with high income but low spending.
- Cluster 4: Customers with low income but high spending.

### 4.5.2. Analysis

- The K-Means algorithm worked well with this dataset.
- The Elbow Method and Silhouette score show that k = 5 is the best choice.
- Businesses can utilize these clusters to analyze customer spending habits and develop targeted merketing strategies or personalized services.

## 5. Question 3: Regression Analysis

### 5.1. Importing Necessary Libraries



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

- **pandas(pd):** It is a powerful library used for data analysis and processing in Python. It facilitates operations such as data manipulation, filtering, transformation, statistical analysis and missing data management by providing DataFrame and Series structures, which are structured data types.

- **numpy(np):** It is used for numerical calculations and working with multidimensional arrays. Matrix operations play an important role in linear algebra, random numbers, and scientific calculations.

- **sklearn.model_selection.train_test_split:** It is a library used to create and evaluate machine learning models. It is used to divide the data into training and test sets.

- **sklearn.ensemble.RandomForestRegressor:** A model used to solve regression problems using the random forest algorithm. It makes more accurate predictions by combining the results of multiple decision trees.

- **sklearn.linear_model.LinearRegression:** It is the class used to implement the linear regression model. It is used to model the linear relationship between the dependent variable and independent variables.

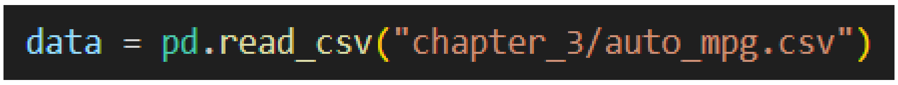
- **sklearn.metrics.mean_sqared_error:** Calculates the average of the squares of the difference between predicted values and actual values. It penalizes larger errors more and is used to measure the accuracy of the model.

- **sklearn.metrics.mean_absolute_error:** Calculates the absolute average of the difference between predicted and actual values. It is used to measure errors directly and is easier to interpret.

- **sklearn.metrics.r2_score:** It measures the explanatory power of the model; It shows how close the predicted values are to the actual values. Values close to 1 indicate that the model performs well.
- **sklearn.preprocessing.StandardScaler:** It is a class that scales data with a mean of 0 and a standard deviation of 1.
- **matplotlib.pyplot(plt):** It is a submodule of the Matplotlib library used for data visualization in Python and provides a simple interface for creating graphs and plots.
- **seaborn(sns):** It is a library based on Matplotlib used for statistical data visualization in Python. It allows you to create impressive graphics easily.

## 5.2. Data Preprocessing

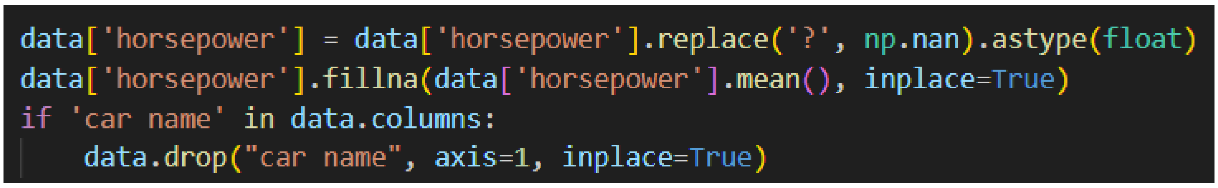
### 5.2.1. Loading and Preparing the Data



```python
data = pd.read_csv("chapter_3/auto_mpg.csv")
```

I downloaded my dataset called "auto_mpg" from Kaggle. Then I included the dataset in my project using the "read_csv()" function.

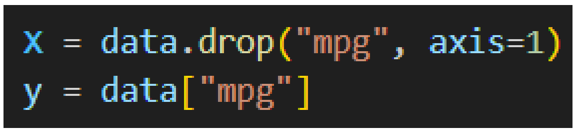
### 5.2.2. Missing Data Management and Column Cleanup



```python
data['horsepower'] = data['horsepower'].replace('?', np.nan).astype(float)
data['horsepower'].fillna(data['horsepower'].mean(), inplace=True)
if 'car name' in data.columns:
    data.drop("car name", axis=1, inplace=True)
```

1) First I fixed the missing data in the "horsepower" column. "?" I set the places to "NAN".
2) Then I filled in the missing values with the average value of the column. After this process, there was no missing data left.
3) Finally, I removed my column named "car name", which was not necessary for analysis.

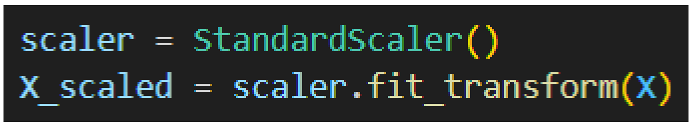
### 5.2.3. Separation of Dependent and Independent Variables



```python
X = data.drop("mpg", axis=1)
y = data["mpg"]
```

1) First, I determined the independent variables in the data set and assigned them to variable x. I removed the "mpg" column from the dataset. As a result of this process, all remaining columns were assigned as independent variables to variable x.
2) Then I determined the dependent variables and assigned them to variable y. I selected the entire "mpg" column and assigned it as the dependent variable. The y variable represents the target variable that is desired to be predicted.

### 5.2.4. Scaling of Independent Variables

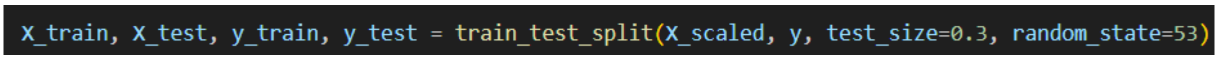


```python
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

1) First, I created a variable called "scaler". I created an object to scale the data using the "StandardScaler()" function and assigned this object to the "scaler" variable (The "StandardScaler()" function is the class that scales the data so that its mean is 0 and its standard deviation is 1).
2) Then I created a variable called "x_scaled". I scaled the data using the "scaler.fit_transform()" function and assigned the scaled values to the "x_scaled" variable.

## 5.3. Modeling

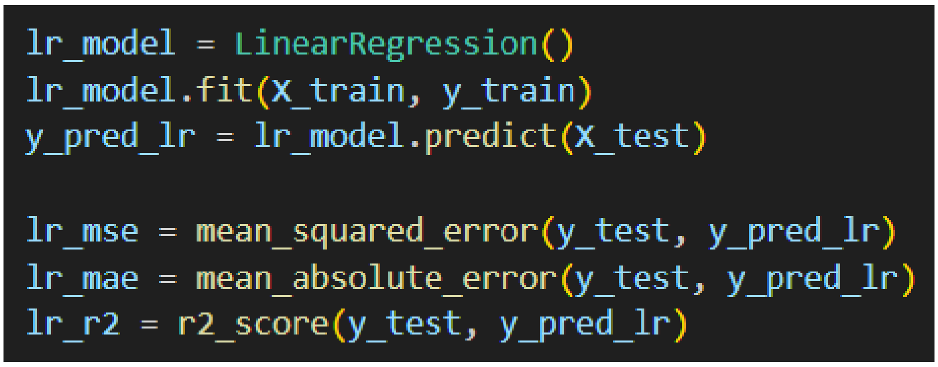
### 5.3.1. Spliting Data



```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=53)
```

- The dataset divided into:
  - 70% training data for the model to learn.
  - 30% test data for evaluating the model's performance.
- The splitting has done using the train_test_split function with random_state=53 to ensure reproducibility.

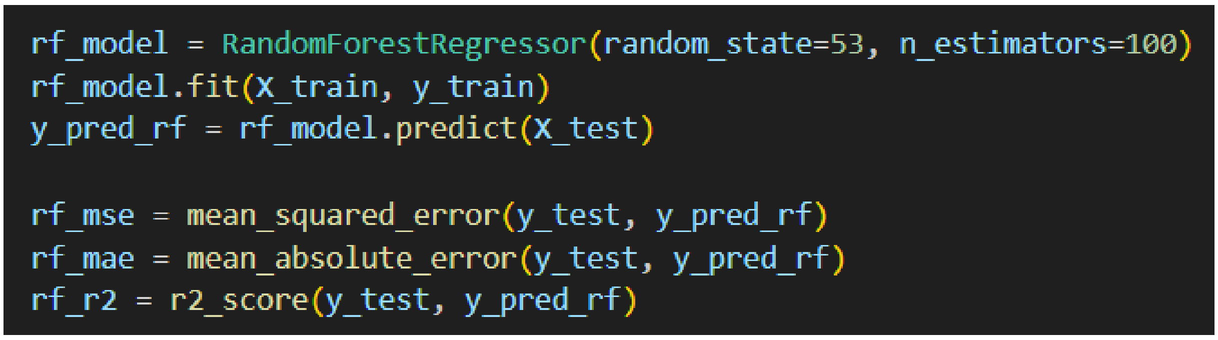
### 5.3.2. Training, Estimation and Performance Evaluation of Linear Regression Model



```
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
y_pred_lr = lr_model.predict(X_test)

lr_mse = mean_squared_error(y_test, y_pred_lr)
lr_mae = mean_absolute_error(y_test, y_pred_lr)
lr_r2 = r2_score(y_test, y_pred_lr)
```

1) First I created a variable called "lr_model". Then, I created a linear regression object using the "LinearRegression()" function and assigned it to the "lr_model" variable.
2) Then I trained the model with the "lr_model.fit()" function.
3) Then, I created a variable named "y_pred_lr" and assigned the values I predicted with the "lr_model.predict()" function to this variable. (y_pred_lr contains the predicted dependent variable values.).
4) Finally, I created variables named "lr_mse", "lr_mae" and "lr_r2" to evaluate model performance. I assigned the resulting values to these variables using the "mean_squared_error()", "mean_absolute_error()" and "r2_score()" functions.

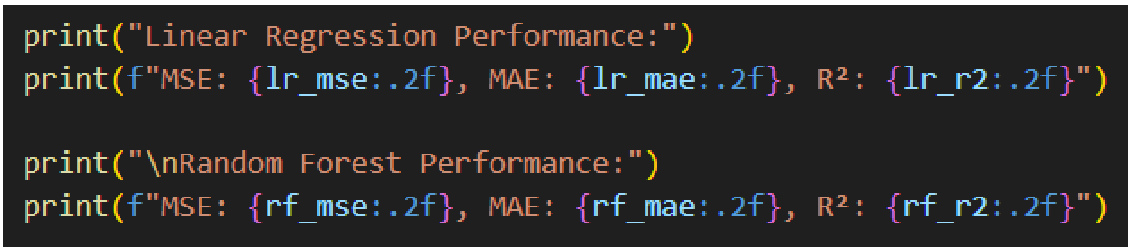
### 5.3.3. Training, Estimation and Performance Evaluation of Random Forest Model



```
rf_model = RandomForestRegressor(random_state=53, n_estimators=100)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

rf_mse = mean_squared_error(y_test, y_pred_rf)
rf_mae = mean_absolute_error(y_test, y_pred_rf)
rf_r2 = r2_score(y_test, y_pred_rf)
```

1) First I created a variable called "rf_model". Then, I created a random forest regression object using the "RandomForestRegression()" function and assigned it to the "rf_model" variable.
2) Then I trained the model with the "rf_model.fit()" function.
3) Then, I created a variable named "y_pred_rf" and assigned the values I predicted with the "rf_model.predict()" function to this variable. (y_pred_rf contains the predicted dependent variable values.).
4) Finally, I created variables named "rf_mse", "rf_mae" and "rf_r2" to evaluate model performance. I assigned the resulting values to these variables using the "mean_squared_error()", "mean_absolute_error()" and "r2_score()" functions.

## 5.4. Performance Evaluation



```
print("Linear Regression Performance:")
print(f"MSE: {lr_mse:.2f}, MAE: {lr_mae:.2f}, R²: {lr_r2:.2f}")

print("\nRandom Forest Performance:")
print(f"MSE: {rf_mse:.2f}, MAE: {rf_mae:.2f}, R²: {rf_r2:.2f}")
```

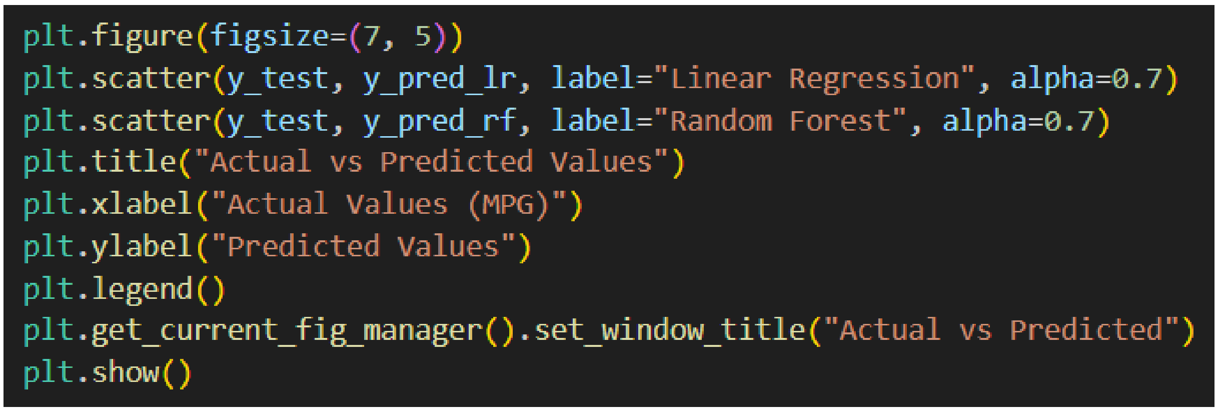
1) First I printed the performance of linear regression. (I used formatting when printing and set it to 2 digits after the comma.).
2) Then I printed the performance of random forest. (I used formatting when printing and set it to 2 digits after the comma.).

The models were evaluated using three metrics: Mean Squared Error(MSE), Mean Absolute Error(MAE) and $R^2$ Score. The table below summarizes the results:

| Metric | Lineer Regression | Random Forest Regression |
|---|---|---|
| MSE | 12.52 | 7.92 |
| MAE | 2.63 | 1.96 |
| $R^2$ Score | 0.82 | 0.88 |

## 5.5. Plotting

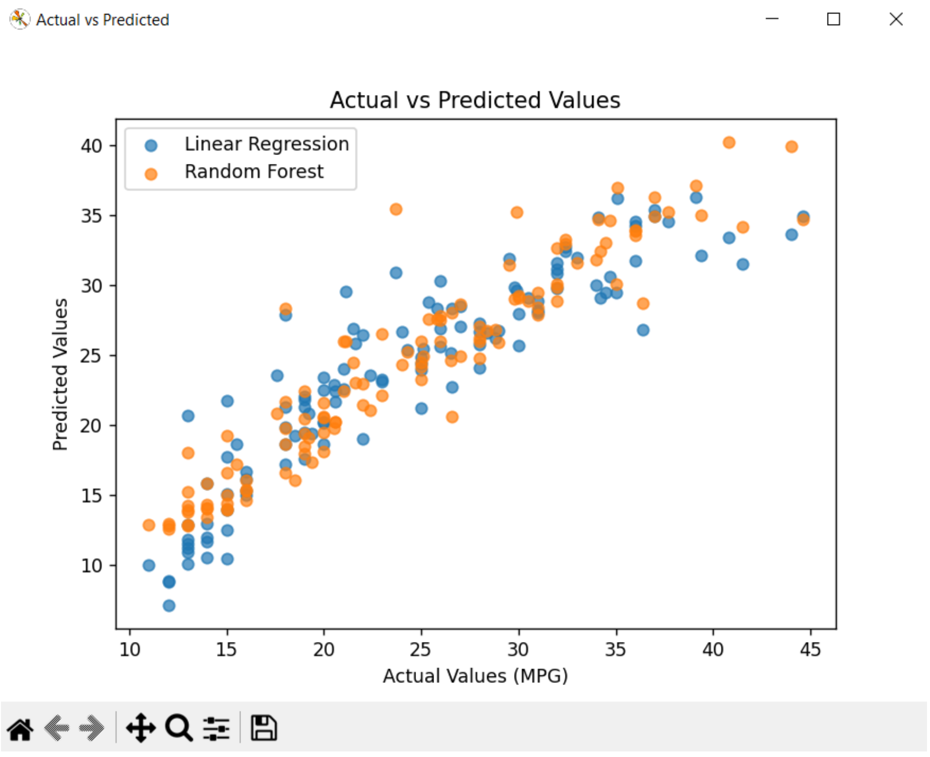
### 5.5.1. Visualization of Actual and Predicted Values



```python
plt.figure(figsize=(7, 5))
plt.scatter(y_test, y_pred_lr, label="Linear Regression", alpha=0.7)
plt.scatter(y_test, y_pred_rf, label="Random Forest", alpha=0.7)
plt.title("Actual vs Predicted Values")
plt.xlabel("Actual Values (MPG)")
plt.ylabel("Predicted Values")
plt.legend()
plt.get_current_fig_manager().set_window_title("Actual vs Predicted")
plt.show()
```

1) First, I set the size of the plot I will create.
2) Then I visualized the linear regression predictions with the "plt.scatter()" function. Again, I visualized the random forest predictions using the same function.
3) Then I named the title and x-y axes.
4) Then, I added a legend to my chart with the "plt.legend()" function. The purpose of this legend is to distinguish between linear regression and random forest predictions.
5) Then I changed the window name to “Actual vs Predicted”.
6) Finally, I plotted the graph with the “show()” function.

**Actual vs Predicted Values Scatter Plot:**



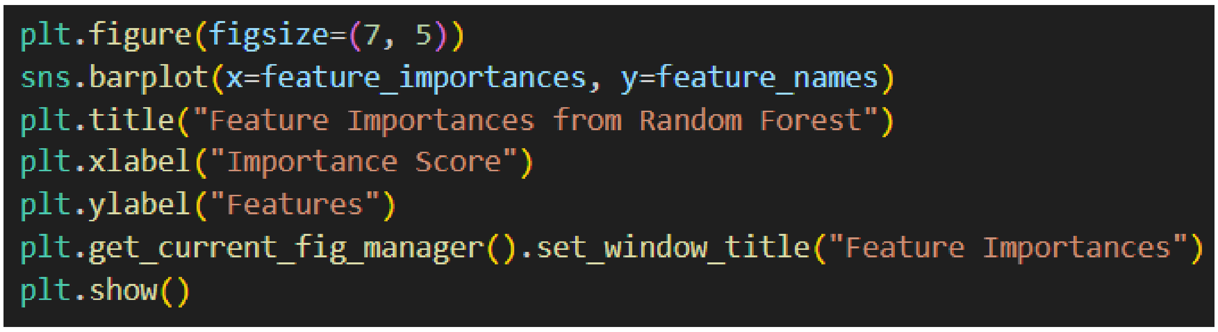
- The Random Forest model performs significantly better than Linear Regression on all metrics.
- Its lower MSE and MAE values indicate higher accuracy.

### 5.5.2. Determining Feature Importance with Random Forest

```
feature_importances = rf_model.feature_importances_
feature_names = data.drop("mpg", axis=1).columns
print(feature_importances)
```

1) First, I created a variable called "feature_importances" and assigned it to this variable by returning the importance level of each feature in the random forest model with "rf_model.feature_importances_".
2) Then I defined a variable called "feature_names" and removed the "mgp" column from the data frame with the "data.drop()" function. After this process, only independent variables (features) remain. I took the names of the columns with ".columns" and assigned them to the variable named "feature_names".
3) Finally I printed the “feature_importances” variable to the console.

### 5.5.3. Visualizing Random Forest Feature Importance

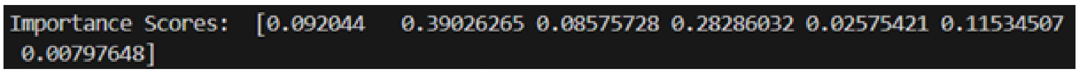


```python
plt.figure(figsize=(7, 5))
sns.barplot(x=feature_importances, y=feature_names)
plt.title("Feature Importances from Random Forest")
plt.xlabel("Importance Score")
plt.ylabel("Features")
plt.get_current_fig_manager().set_window_title("Feature Importances")
plt.show()
```

1) First, I set the size of the plot I will create.
2) Then I created the bar chart with the "sns.barplot()" function. I set it so that the importance of the features is on the x-axis and the names of the features are on the y-axis.
3) Then I named the title and x-y axes.
4) Then I changed the window name to “Feature Importances”.
5) Finally, I plotted the graph with the “show()” function.

## 5.6 Results and Discussion

### 5.6.1. Importance of Features

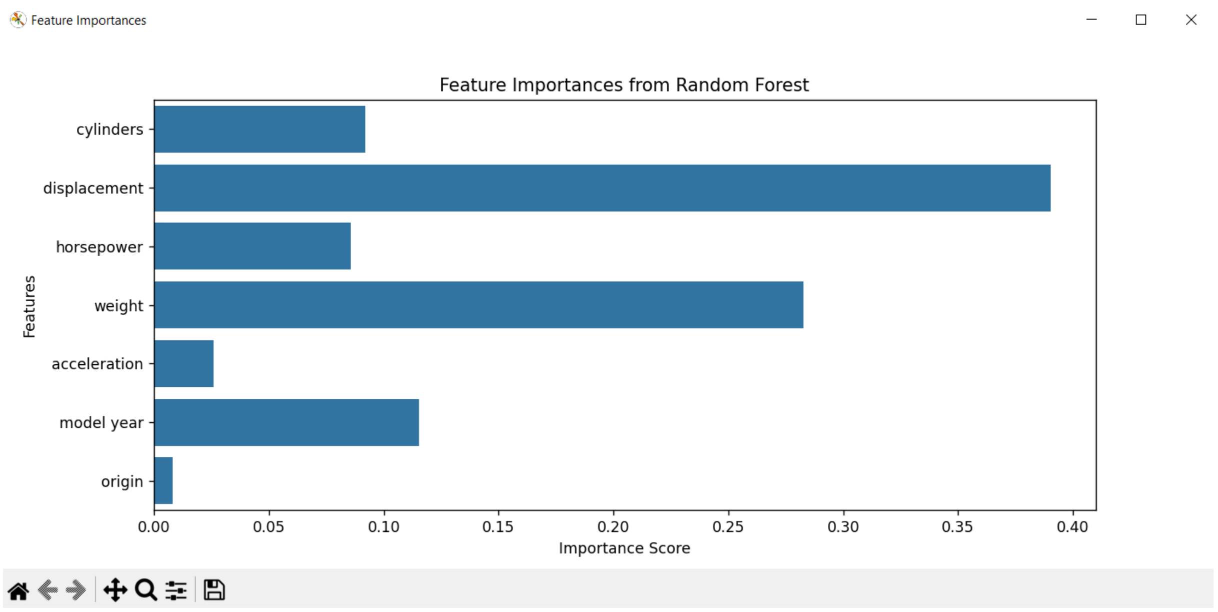


```
Importance Scores:  [0.092044   0.39026265 0.08575728 0.28286032 0.02575421 0.11534507
 0.00797648]
```

The Random Forest model shows the importance of each feature in predicting MPG:

| Feature | Importance Score |
|---|---|
| Cyclinders | 0.09 |
| Displacement | 0.39 |
| Horsepower | 0.08 |
| Weight | 0.28 |
| Acceleration | 0.02 |
| Model Year | 0.11 |
| Origin | 0.00 |

**Feature Importance Bar Chart:**



1) **Displacement (0.39):** The total volume of the engine (engine displacement) seems to be the most important feature for the model. This shows that engine size has a significant impact on fuel efficiency. Larger engines generally consume more fuel, so this feature plays a critical role in mpg estimation.
2) **Weight (0.28):** Vehicle weight is the second most important feature. Weight is a key factor that directly affects a vehicle's fuel efficiency; Heavier vehicles generally have lower mpg ratings.
3) **Model Year (0.11):** The year of production is another important feature. The fact that vehicles have become more efficient with technological developments may explain the effectiveness of this feature.
4) **Cylinders (0.09):** The number of cylinders is of moderate importance in model predictions as it is related to engine performance and fuel consumption.
5) **Horsepower (0.08):** Horsepower refers to engine power and is related to fuel consumption. However, it has a slightly lower importance compared to other features.
6) **Acceleration (0.02):** Acceleration (0-60 mph time) is a feature that has less impact on fuel consumption. This may perhaps be due to the fact that this feature has an indirect relationship with mpg.
7) **Origin (0.00):** The region where the vehicle was produced (USA, Europe, Japan) had little or no impact on the model.

#### 5.6.2. Model Comparison

- **Lineer Regression:** Easier to implement and interpret, but less accurate.
- **Random Forest:** Provides higher accuracy by capturing non-linear relationships in the data.

#### 5.6.3. Conclusion

The Random Forest Regression model is better for this task because it provides more accurate predictions and identifies important features effectively.

# 6. Conclusion

## 6.1. Summary

In this project, we applied three main data analysis methods: classification, clustering and regression. These methods allowed us to extract meaningful insights from different datasets. The key findings are summarized below:

- **Classification:** Using the "diabetes" dataset, we implemented decision tree, random forest, and SVM models to predict whether an individual has diabetes. Among these, SVM achieved the highest accuracy and precision, making it the most reliable model overall. Random forest, while slightly less accurate than SVM, provided a balanced trade-off between precision and recall, outperforming the decision tree in both accuracy and error reduction.
- **Clustering:** The "mall_customers" dataset was used to segment customers based on their spending habits. Using the K-Means algorithm, we grouped customers into three clusters, determined as optimal through the elbow method. This segmentation helps in understanding different customer spending behaviors.
- **Regression:** The "auto_mpg" dataset was employed to predict vehicle fuel efficiency (miles per gallon, MPG). Both linear regression and random forest models were used, with random forest demonstrating superior performance by better capturing the relationships within the data and yielding more precise results.

## 6.2. Analysis

Each method was effective for its respective purpose, offering valuable insights into different types of data. Here's why these methods are significant:

- **Classification:** Using the "diabetes" dataset, we implemented decision tree, random forest, and SVM models to predict diabetes. Among these, SVM achieved the highest accuracy and precision, making it the most reliable model overall. The random forest model, while slightly less accurate than SVM, proved to be highly effective due to its balanced trade-off between precision and recall. This reliability makes it a valuable tool for medical studies, assisting healthcare professionals in making informed decisions about early diagnosis and treatment strategies.
- **Clustering:** The K-Means method provided clear segmentation of customer spending behavior from the "mall_customers" dataset. This is particularly useful for businesses to identify high spenders, moderate spenders, and budget-conscious customers, enabling tailored marketing strategies and better resource allocation.
- **Regression:** The random forest model excelled at predicting vehicle fuel efficiency (MPG) using the "auto_mpg" dataset. Its ability to model complex, non-linear relationships made it superior to simpler approaches like linear regression. This strength is crucial for analyzing intricate data patterns, leading to more accurate predictions in applications like automotive engineering and environmental studies.

## 6.3. Recommendations

1) Use SVM for the most accurate predictions, particularly in applications requiring high precision, such as health-related datasets or complex data like MPG.
2) Apply K-Means clustering to group similar data, helping businesses understand customers better.
3) Clean and preprocess data carefully for better results.
4) Compare methods to find the best fit; random forest often outperforms simpler models.

## 6.4 Final Thoughts

Data analysis provides practical solutions to real-world challenges by finding patterns and enabling informed decisions in various domains.

- **Healthcare:** Using the "diabetes" dataset, we demonstrated how decision tree, random forest, and SVM models can predict diseases like diabetes. SVM, with its superior accuracy and precision, proved to be the most effective for early diagnosis. Random forest also showed great potential, offering a balanced approach that aids in developing better treatment plans.
- **Business:** With the "mall_customers" dataset, clustering techniques like K-Means helped segment customers, enabling businesses to optimize their strategies and improve services.
- **Automotive:** The "auto_mpg" dataset demonstrated how regression methods can predict vehicle fuel efficiency, helping in environmental studies and automotive innovation.

Choosing the right method for the dataset is very important. In this project, SVM gave the best results with high accuracy and precision. Random forest also worked well, offering a good balance of performance. These tools turn data into useful insights, helping make better decisions in many areas.