RECEP TAYYIP ERDOGAN UNIVERSITY

FACULTY OF ENGINEERING AND ARCHITECTURE

COMPUTER ENGINEERING DEPARTMENT

DATA MINING

2024-2025

Student Name Surname

Ramazan Serhat Uygun

Student No

201401049

Instructor

Dr. Öğr. Üyesi ABDULGANİ KAHRAMAN

RIZE

QUESTION – 1 REPORT

Name: Ramazan Serhat Uygun

Number: 201401049

Subject: Comparing and training Decision Tree and Random Forest models on Breast Cancer data

Dataset:

- **Name:** Breast Cancer
- **Source:** Kaggle
- **Link:** https://www.kaggle.com/datasets/reihanenamdari/breast-cancer

Methods Used: Normalization, Standardization, Decision Tree, SMOTE, Data Extraction, Random Forest, Data Preprocessing, Hyperparameter Adjustment

1. Objective

The aim of this study is to create a decision tree classification model using the given dataset, evaluate the performance of the model and analyze the effectiveness of the two approaches by comparing them with the random forest method.

Decision trees are an effective way to classify data by dividing it into branches according to certain rules. However, the Random Forest method aims to improve generalizability by using multiple Decisional trees together and avoiding excessive complexity. Therefore, the comparison of the two methods according to accuracy rates, error metrics and other performance criteria is an important part of the study.

The study classifies the target variable using the arguments in the dataset, analyzes the results and evaluates the potential benefits of classification using the random forest method compared to the decision tree approach.

2. Dataset Details

Description: This dataset of breast cancer patients was obtained from the 2017 November update of the SEER Program of the NCI, which provides information on population-based cancer statistics. The dataset involved female patients with infiltrating duct and lobular carcinoma breast cancer (SEER primary cites recode NOS histology codes 8522/3) diagnosed in 2006-2010. Patients with unknown tumor size examined regional LNs, positive regional LNs, and patients whose survival months were less than 1 month were excluded; thus, 4024 patients were ultimately included.

Dataset Structure:

- **Rows:** 4024
- **Columns:** 16
- **Features:**
  1. **Age:** Patient age
  2. **Race:** Patient race
  3. **Marital Status:** Patient's marital status
  4. **T Stage:** Tumor stage
  5. **N Stage:** Lymph node stage
  6. **6th Stage:** Clinical stage
  7. **Differentiate:** Cellular differentiation status
  8. **Grade:** Cancer degree
  9. **A Stage:** Regional stage
  10. **Tumor Size:** Tumor size
  11. **Estrogen Status:** Estrogen receptor status
  12. **Progesterone Status:** Progesterone receptor status
  13. **Regional Node Examined:** Number of lymph nodes examined
  14. **Regional Node Positive:** Number of positive lymph nodes
  15. **Survival Months:** Number of months survived
  16. **Status:** Patient's life situation

2.1. Why Breast Cancer Dataset?

I think this data set I am using is suitable for the decision tree model. The categorical structure of the target variables and the diversity of the independent variables satisfy the basic working logic of the decision tree. In order to use this dataset effectively, I corrected the missing data and eliminated the class imbalance found, then I made it suitable for model training and evaluated it.

3. Required Libraries

For the model training, I included all the libraries necessary for visualization and the necessary functions to work.

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report
from sklearn.preprocessing import LabelEncoder, StandardScaler
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_curve, auc
```

4. Loading the Dataset and Initial Exploration

To examine the data set, I showed the first 5 rows, examined the missing values and performed an analysis by classifying the categorical variable that I identified as the target. To examine the variables in more detail, I also examined the status of some variables depending on the target variable.

```
file_path = "Breast_Cancer.csv"
data = pd.read_csv(file_path)

print("First 5 Rows of the Data Set:")
print("-----------------------------")
print(data.head())
print("-------------------------------------------------------------------------------------------")
print("\nGeneral Information of the Data Set:")
print("---------------------------------------")
print(data.info())
print("-------------------------------------------------------------------------------------------")

print("\nNumber of Missing Values:")
print("---------------------------")
print(data.isnull().sum())
print("-------------------------------------------------------------------------------------------")

print("\nTarget Variable (Status) Class Distribution:")
print("-----------------------------------------------")
print(data['Status'].value_counts())
```

Output:

```
First 5 Rows of the Data Set:
------------------------------
   Age   Race Marital Status T Stage  N Stage 6th Stage  \
0   68  White        Married       T1      N1       IIA
1   50  White        Married       T2      N2      IIIA
2   58  White       Divorced       T3      N3      IIIC
3   58  White        Married       T1      N1       IIA
4   47  White        Married       T2      N1       IIB

              differentiate Grade   A Stage  Tumor Size Estrogen Status  \
0      Poorly differentiated     3  Regional           4        Positive
1  Moderately differentiated     2  Regional          35        Positive
2  Moderately differentiated     2  Regional          63        Positive
3      Poorly differentiated     3  Regional          18        Positive
4      Poorly differentiated     3  Regional          41        Positive

  Progesterone Status  Regional Node Examined  Reginol Node Positive  \
0            Positive                      24                      1
1            Positive                      14                      5
2            Positive                      14                      7
3            Positive                       2                      1
4            Positive                       3                      1

   Survival Months Status
0                60  Alive
1                62  Alive
2                75  Alive
3                84  Alive
4                50  Alive

Target Variable (Status) Class Distribution:
--------------------------------------------------
Status
Alive    3408
Dead      616
Name: count, dtype: int64
```

5. Data Preprocessing Steps

I will start the data preprocessing process by processing the missing data, converting categorical variables and analyzing numerical variables. I will then make the dataset suitable for modeling.

5.1. Processing Incomplete Data

```
print("Missing Value Numbers:")
print("---------------------------")
print(data.isnull().sum())

categorical_columns = ['Marital Status', 'Race', 'Estrogen Status', 'Progesterone Status']
for col in categorical_columns:
    mode_value = data[col].mode()[0]
    data[col] = data[col].fillna(mode_value)

numerical_columns = ['Age', 'Tumor Size', 'Regional Node Examined', 'Reginol Node Positive']
for col in numerical_columns:
    median_value = data[col].median()
    data[col] = data[col].fillna(median_value)
```

Output:

```
Missing Value Numbers:
----------------------------
Age                        0
Race                       0
Marital Status             0
N Stage                    0
6th Stage                  0
differentiate              0
Grade                      0
A Stage                    0
Tumor Size                 0
Estrogen Status            0
Progesterone Status        0
Regional Node Examined     0
Reginol Node Positive      0
Survival Months            0
Status                     0
T Stage                    0
dtype: int64
```

5.2. Transforming Categorical Variables

Categorical data cannot be processed directly by machine learning algorithms. Therefore, I converted categorical data into numerical values.

```
categorical_columns = ['Race', 'Marital Status', 'T Stage', 'N Stage', '6th Stage', 'differentiate', 'Grade', 'A Stage',
'Estrogen Status', 'Progesterone Status', 'Status']

label_encoders = {}
for col in categorical_columns:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le

print("First 5 rows of coded columns:")
print("------------------------------")
print(data[categorical_columns].head())
```

Output:

```
First 5 rows of coded columns:
------------------------------
   Race  Marital Status  T Stage  N Stage  6th Stage  differentiate  Grade  \
0     2               1        0        0          0              1      3
1     2               1        1        1          2              0      2
2     2               0        2        2          4              0      2
3     2               1        0        0          0              1      3
4     2               1        1        0          1              1      3

   A Stage  Estrogen Status  Progesterone Status  Status
0        1                1                    1       0
1        1                1                    1       0
2        1                1                    1       0
3        1                1                    1       0
4        1                1                    1       0
```

5.3. Scaling of Numerical Variables

In this section, I normalize and standardize the dataset in order to better train or use it for the model. In this way, the data takes values between the numbers (0-1), making it more suitable for model training.

```
scaler = StandardScaler()
numerical_columns = data.select_dtypes(include=['int64', 'float64']).columns

scaler = StandardScaler()
data_scaled = data.copy()
data_scaled[numerical_columns] = scaler.fit_transform(data[numerical_columns])

print("First 5 rows of coded columns:")
print("------------------------------")
print(data_scaled.head())
```

Output:

```
First 5 rows of coded columns:
------------------------------
        Age  Race  Marital Status  N Stage  6th Stage  differentiate  Grade  \
0  1.565253     2               1        0          0              1      3
1 -0.443222     2               1        1          2              0      2
2  0.449434     2               0        2          4              0      2
3  0.449434     2               1        0          0              1      3
4 -0.777968     2               1        0          1              1      3

   A Stage  Tumor Size  Estrogen Status  Progesterone Status  \
0        1   -1.253661                1                    1
1        1    0.214345                1                    1
2        1    1.540287                1                    1
3        1   -0.590691                1                    1
4        1    0.498475                1                    1

   Regional Node Examined  Reginol Node Positive  Survival Months  Status  \
0                1.190676              -0.618172        -0.492961       0
1               -0.044095               0.164807        -0.405695       0
2               -0.044095               0.556296         0.161530       0
3               -1.525820              -0.618172         0.554224       0
4               -1.402343              -0.618172        -0.929288       0

   T Stage
0        0
1        1
2        2
3        0
4        1
```

5.4. Splitting a Dataset

To prepare a suitable infrastructure for a machine learning model by dividing the dataset into training and test sets. The training set is used to learn the model, while the test set is used to evaluate the performance of the model. Furthermore, thanks to the stratify parameter, the class distribution of the target variable is preserved in both subsets, which results in a more balanced model training. For this training, I split the dataset into 80% training and 20% testing, to avoid overfitting if the proportion chosen for testing is too high.

```
X = data.drop('Status', axis=1)
y = data['Status']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

print("\nExample 5 Rows from the Training Set (X_train):")
print("-------------------------------------------------")
print(X_train.head())
print("-------------------------------------------------------------------------------")

print("\nTarget 5 Values from Training Set (y_train):")
print("-----------------------------------------------")
print(y_train.head())
print("-------------------------------------------------------------------------------")

print("\nSample 5 Rows from Test Set (X_test):")
print("---------------------------------------")

print(X_test.head())
print("-------------------------------------------------------------------------------")

print("\nTarget 5 Values from Test Set (y_test):")
print("-----------------------------------------")
print(y_test.head())
```

6. Training and Evaluating the Decision Tree Model

I created a Decision Tree model and trained it with the training set. In order to ensure that the model avoids overlearning, the depth was limited with the max_depth parameter. The reason why I did not choose max_depth too much is to avoid overfitting, I kept the depth short to avoid overfitting.

6.1. Model Training

I built the model using DecisionTreeClassifier and trained it with the training set I set.

```
dt_model = DecisionTreeClassifier(max_depth=5, random_state=42)

dt_model.fit(X_train, y_train)

print("Decision Tree Model Trained!")
```

6.2. Evaluating Model Performance

I will compute various evaluation metrics by making predictions of the model on the test set.

```
y_pred = dt_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print("Model Performance:")
print("-------------------")
print(f"- Accuracy: {accuracy:.2f}")
print(f"- Precision: {precision:.2f}")
print(f"- Recall: {recall:.2f}")
print(f"- F1 Score: {f1:.2f}")
print("-------------------------------------------------------------------------------")

print("\nClassification Report:")
print("-------------------------")
print(classification_report(y_test, y_pred))
print("-------------------------------------------------------------------------------")

print("\nConfusion Matrix:")
print("----------------")
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

Output:

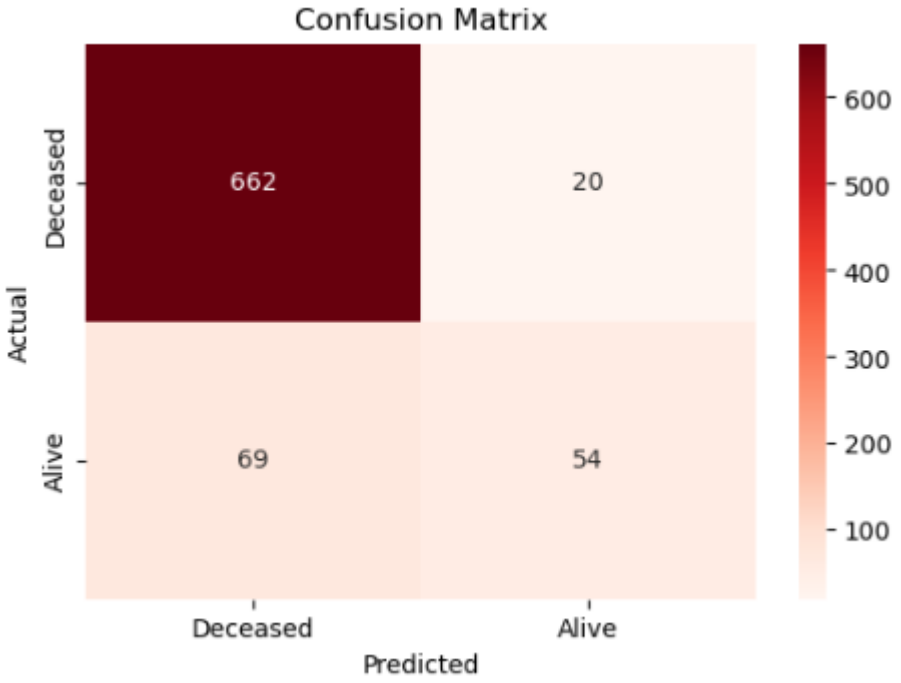
```
Model Performance:
-------------------
- Accuracy: 0.89
- Precision: 0.88
- Recall: 0.89
- F1 Score: 0.88
-------------------------------------------------------------------------------

Classification Report:
-------------------------
              precision    recall  f1-score   support

           0       0.91      0.97      0.94       682
           1       0.73      0.44      0.55       123

    accuracy                           0.89       805
   macro avg       0.82      0.70      0.74       805
weighted avg       0.88      0.89      0.88       805

-------------------------------------------------------------------------------

Confusion Matrix:
----------------
[[662  20]
 [ 69  54]]
```

Description:

Model performance will be measured using various evaluation metrics on the test set. These metrics typically include:

- **Accuracy:** The proportion of samples that the model predicts correctly.
- **Precision:** The proportion of positive values that the model correctly predicts.
- **Recall:** How many true positive values it correctly predicts.
- **F1-Score:** A balanced measure of Precision and Recall.
- **Confusion Matrix:** Shows the relationship between actual and predicted classes.

At this stage, I evaluated the generalization performance of the model by making predictions on the test set.



Description:

This visualization is a confusion matrix that evaluates the performance of a classification model. The image visualizes the relationship between actual and predicted classes divided into four groups.

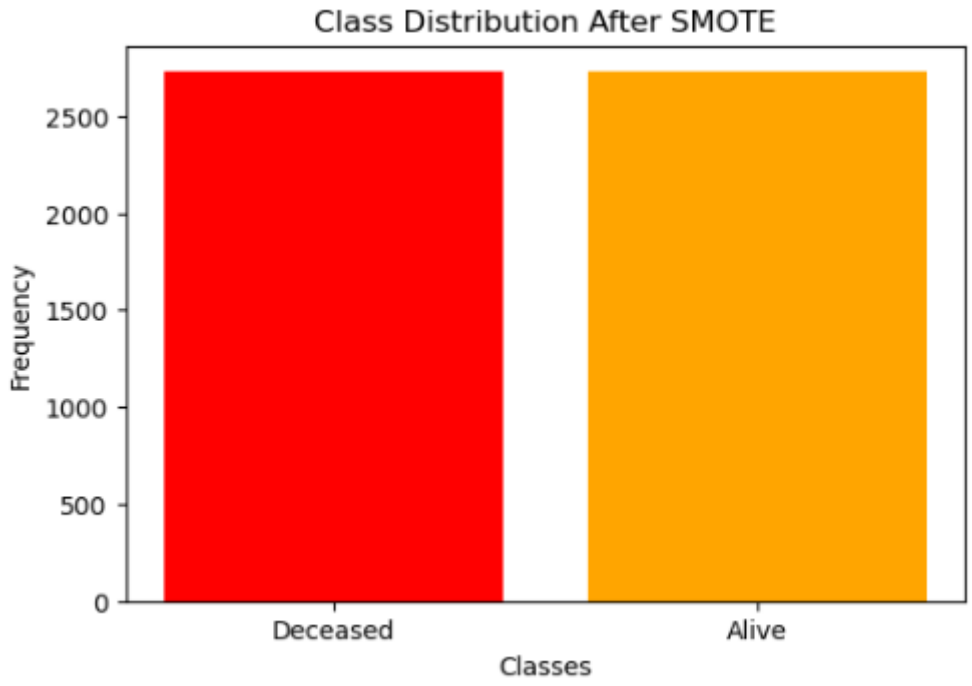
- **Correct Predictions (TN + TP):** In total, the model correctly predicted 657 + 53 = 710 individuals.
- **False Predictions (FP + FN):** In total, the model incorrectly predicted 25 + 70 = 95 individuals.
- **Overall Success:** The correct prediction rate is high, but the number of false negatives (70) is noteworthy. This indicates that the model is less successful in predicting the Alive class.

7. SMOTE Application (for Class Imbalance)

If there is an imbalance between the classes of the target variable, I will increase the minority class using SMOTE (Synthetic Minority Oversampling Technique). This method allows the model to learn the minority class better.

```
print("Class Distribution before SMOTE:")
print("--------------------------------")
print(Counter(y_train))

smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

print("\nClass Distribution after SMOTE:")
print("-------------------------------")
print(Counter(y_train_smote))
```

```
Class Distribution before SMOTE:
--------------------------------
Counter({0: 2726, 1: 493})

Class Distribution after SMOTE:
-------------------------------
Counter({0: 2726, 1: 2726})
```



Description:

This visualization is a bar chart showing the class distribution in the dataset after applying the SMOTE (Synthetic Minority Oversampling Technique) method.

8. Training and Evaluation of the Decision Tree Model (After Applying SMOTE)

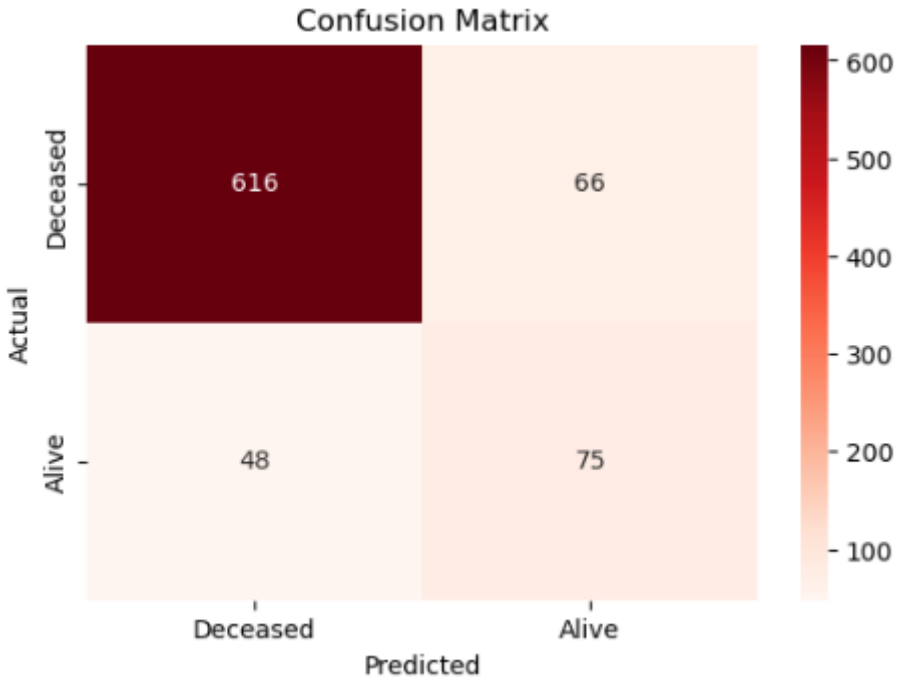
After applying the SMOTE process and removing the imbalance between the data by generating synthetic data, I put it back into the model training process. I check whether there is a change in the model training values due to data imbalance.

```
y_pred = dt_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print("Model Performance:")
print("-------------------")
print(f"- Accuracy: {accuracy:.2f}")
print(f"- Precision: {precision:.2f}")
print(f"- Recall: {recall:.2f}")
print(f"- F1 Score: {f1:.2f}")
print("--------------------------------------------------------------------------------")

print("\nClassification Report:")
print("-------------------------")
print(classification_report(y_test, y_pred))
print("--------------------------------------------------------------------------------")

print("\nConfusion Matrix:")
print("----------------")
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

Output:

```
Model Performance:
-------------------
- Accuracy: 0.86
- Precision: 0.87
- Recall: 0.86
- F1 Score: 0.86
--------------------------------------------------------------------------------

Classification Report:
-------------------------
              precision    recall  f1-score   support

           0       0.93      0.90      0.92       682
           1       0.53      0.61      0.57       123

    accuracy                           0.86       805
   macro avg       0.73      0.76      0.74       805
weighted avg       0.87      0.86      0.86       805

--------------------------------------------------------------------------------

Confusion Matrix:
----------------
[[616  66]
 [ 48  75]]
```

After training the model with Decision Trees after applying only SMOTE and editing with synthetic data, Accuracy, Precision, Recall and F1 Score values decreased. With these decreases, the Error Matrix became more stable.



9. Data Extraction

With the correlation matrix, I perform data extraction to remove the variables with extreme values between each other and bring the values in the model training to a level that I can accept as more stable and accurate.

Extracted Data

From the correlation matrix I examined, I found that the correlations between some of the data had outliers.

Correlation values:

- There is a high correlation of 0.84 between **N Stage** and **Regional Node Positive**.
- There is a 0.81 correlation between **6th Stage** and **T Stage**.
- There is a 0.61 correlation between **Tumor Size** and **T Stage**.
- Low Correlated Variables:
    - **Race**, **Marital Status**, **differentiate** variables showed very low correlations.

10. Perform Splitting for Testing After Extracting Data

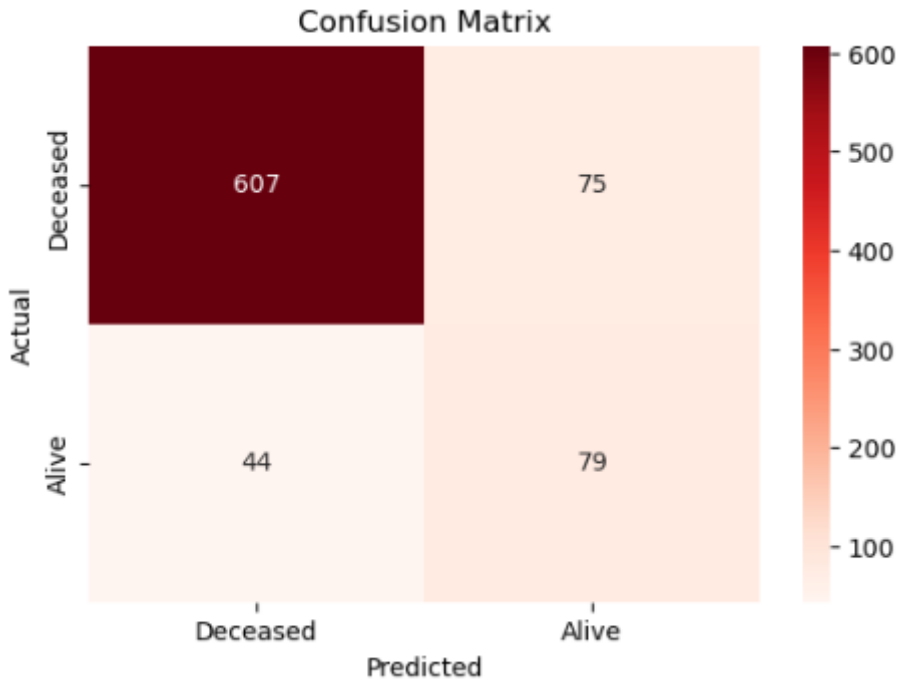
After performing the data extraction process, I train the dataset by setting test and train values again. Then I apply SMOTE to the extracted data and train the model again.

```
Class Distribution before SMOTE:
--------------------------------
Counter({0: 2726, 1: 493})

Class Distribution after SMOTE:
-------------------------------
Counter({0: 2726, 1: 2726})
```

```
Model Performance:
------------------
- Accuracy: 0.85
- Precision: 0.87
- Recall: 0.85
- F1 Score: 0.86
```

```
Classification Report:
-------------------------
              precision    recall  f1-score   support

           0       0.93      0.89      0.91       682
           1       0.51      0.64      0.57       123

    accuracy                           0.85       805
   macro avg       0.72      0.77      0.74       805
weighted avg       0.87      0.85      0.86       805
```

```
Confusion Matrix:
----------------
[[607  75]
 [ 44  79]]
```



Description:

As a result of the removal of the data contrary to the correlation, optimization operations in Accuracy, Precision, Recall and F1 Score values were performed and optimization was provided in a way that would be sufficient in the values in the error matrix.

11. Hyperparameter Tuning

Why do Hyper parameterization?

Hyperparameter tuning is the process of optimizing controllable parameters of the algorithm to improve the performance of the model. Hyperparameters affect how the model works, but they are not learned from the data.

1. Controlling Model Complexity:
    - In algorithms such as decision trees, the max_depth parameter determines how complex the model is. A tree that is too deep can lead to overfitting, while a tree that is too shallow can lead to underfitting.
2. Improving Model Performance:
    - The right combinations of hyperparameters can improve the accuracy, precision and other metrics of the model.
3. Improving Overall Performance:
    - Hyperparameter optimization is performed to avoid models that overfit the training data (overfitting) or miss a general structure (underfitting).

Here I will use Grid Search for hyper parameterization

Grid Search

- For certain hyperparameter values, all combinations are tried.
- It is generally suitable for small data sets.

```
param_grid = {
    'max_depth': [3, 5, 10, 12, 16],
    'min_samples_split': [2, 5, 10, 11, 14],
    'criterion': ['gini', 'entropy']
}

dt_model = DecisionTreeClassifier(random_state=42)

grid_search = GridSearchCV(
    estimator=dt_model,
    param_grid=param_grid,
    scoring='accuracy',
    cv=5,
    verbose=1,
    n_jobs=-1
)

grid_search.fit(X_train_smote, y_train_smote)

results = grid_search.cv_results_

print("\nGrid Search Results:")
print("----------------------")
for mean_score, params in zip(results['mean_test_score'], results['params']):
    print(f"Parameters {params}, Average Accuracy: {mean_score:.4f}")

print("\nBest Parameters:")
print(grid_search.best_params_)

print(f"\nBest Accuracy Score: {grid_search.best_score_:.4f}")
```

Why did I choose these parameters?

The purpose for selecting these parameters:

- **criterion = 'entropy' :**
    - Entropy is based on the calculation of information gain. This criterion splits branches in such a way that more information is gained.
- **max_depth = 16:**
    - The depth of the decision tree is increased to enable more learning.
- **min_samples_split = 2**
    - A minimum of 2 instances are required for a node to branch. This value increases the flexibility of the model and allows it to learn in more detail.

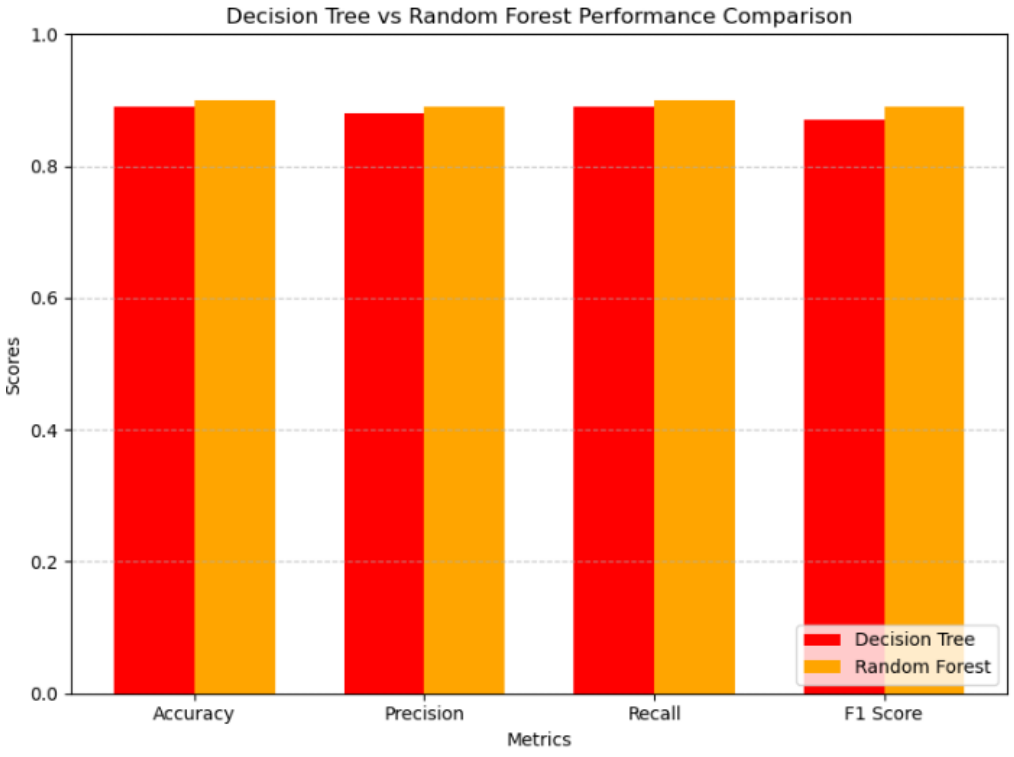
The values I have given for these parameters are experimentally determined in order to obtain the most optimized results.

```
Grid Search Results:
-----------------------
Parameters {'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 2}, Average Accuracy: 0.7913
Parameters {'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 5}, Average Accuracy: 0.7913
Parameters {'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 10}, Average Accuracy: 0.7913
Parameters {'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 11}, Average Accuracy: 0.7913
Parameters {'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 14}, Average Accuracy: 0.7913
Parameters {'criterion': 'gini', 'max_depth': 5, 'min_samples_split': 2}, Average Accuracy: 0.7913
Parameters {'criterion': 'gini', 'max_depth': 5, 'min_samples_split': 5}, Average Accuracy: 0.7913
Parameters {'criterion': 'gini', 'max_depth': 5, 'min_samples_split': 10}, Average Accuracy: 0.7918
Parameters {'criterion': 'gini', 'max_depth': 5, 'min_samples_split': 11}, Average Accuracy: 0.7918
Parameters {'criterion': 'gini', 'max_depth': 5, 'min_samples_split': 14}, Average Accuracy: 0.7918
Parameters {'criterion': 'gini', 'max_depth': 10, 'min_samples_split': 2}, Average Accuracy: 0.8265
Parameters {'criterion': 'gini', 'max_depth': 10, 'min_samples_split': 5}, Average Accuracy: 0.8248
Parameters {'criterion': 'gini', 'max_depth': 10, 'min_samples_split': 10}, Average Accuracy: 0.8248
Parameters {'criterion': 'gini', 'max_depth': 10, 'min_samples_split': 11}, Average Accuracy: 0.8272
Parameters {'criterion': 'gini', 'max_depth': 10, 'min_samples_split': 14}, Average Accuracy: 0.8219
Parameters {'criterion': 'gini', 'max_depth': 12, 'min_samples_split': 2}, Average Accuracy: 0.8412
Parameters {'criterion': 'gini', 'max_depth': 12, 'min_samples_split': 5}, Average Accuracy: 0.8344
Parameters {'criterion': 'gini', 'max_depth': 12, 'min_samples_split': 10}, Average Accuracy: 0.8364
Parameters {'criterion': 'gini', 'max_depth': 12, 'min_samples_split': 11}, Average Accuracy: 0.8377
Parameters {'criterion': 'gini', 'max_depth': 12, 'min_samples_split': 14}, Average Accuracy: 0.8333
Parameters {'criterion': 'gini', 'max_depth': 16, 'min_samples_split': 2}, Average Accuracy: 0.8560
Parameters {'criterion': 'gini', 'max_depth': 16, 'min_samples_split': 5}, Average Accuracy: 0.8459
Parameters {'criterion': 'gini', 'max_depth': 16, 'min_samples_split': 10}, Average Accuracy: 0.8428
Parameters {'criterion': 'gini', 'max_depth': 16, 'min_samples_split': 11}, Average Accuracy: 0.8437
Parameters {'criterion': 'gini', 'max_depth': 16, 'min_samples_split': 14}, Average Accuracy: 0.8406
Parameters {'criterion': 'entropy', 'max_depth': 3, 'min_samples_split': 2}, Average Accuracy: 0.7759
Parameters {'criterion': 'entropy', 'max_depth': 3, 'min_samples_split': 5}, Average Accuracy: 0.7759
Parameters {'criterion': 'entropy', 'max_depth': 3, 'min_samples_split': 10}, Average Accuracy: 0.7759
Parameters {'criterion': 'entropy', 'max_depth': 3, 'min_samples_split': 11}, Average Accuracy: 0.7759
Parameters {'criterion': 'entropy', 'max_depth': 3, 'min_samples_split': 14}, Average Accuracy: 0.7759
Parameters {'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 2}, Average Accuracy: 0.7852
Parameters {'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 5}, Average Accuracy: 0.7852
Parameters {'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 10}, Average Accuracy: 0.7852
Parameters {'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 11}, Average Accuracy: 0.7852
Parameters {'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 14}, Average Accuracy: 0.7852
Parameters {'criterion': 'entropy', 'max_depth': 10, 'min_samples_split': 2}, Average Accuracy: 0.8252
Parameters {'criterion': 'entropy', 'max_depth': 10, 'min_samples_split': 5}, Average Accuracy: 0.8232
Parameters {'criterion': 'entropy', 'max_depth': 10, 'min_samples_split': 10}, Average Accuracy: 0.8225
Parameters {'criterion': 'entropy', 'max_depth': 10, 'min_samples_split': 11}, Average Accuracy: 0.8230
Parameters {'criterion': 'entropy', 'max_depth': 10, 'min_samples_split': 14}, Average Accuracy: 0.8203
Parameters {'criterion': 'entropy', 'max_depth': 12, 'min_samples_split': 2}, Average Accuracy: 0.8373
Parameters {'criterion': 'entropy', 'max_depth': 12, 'min_samples_split': 5}, Average Accuracy: 0.8342
Parameters {'criterion': 'entropy', 'max_depth': 12, 'min_samples_split': 10}, Average Accuracy: 0.8322
Parameters {'criterion': 'entropy', 'max_depth': 12, 'min_samples_split': 11}, Average Accuracy: 0.8320
Parameters {'criterion': 'entropy', 'max_depth': 12, 'min_samples_split': 14}, Average Accuracy: 0.8294
Parameters {'criterion': 'entropy', 'max_depth': 16, 'min_samples_split': 2}, Average Accuracy: 0.8613
Parameters {'criterion': 'entropy', 'max_depth': 16, 'min_samples_split': 5}, Average Accuracy: 0.8531
Parameters {'criterion': 'entropy', 'max_depth': 16, 'min_samples_split': 10}, Average Accuracy: 0.8505
Parameters {'criterion': 'entropy', 'max_depth': 16, 'min_samples_split': 11}, Average Accuracy: 0.8503
Parameters {'criterion': 'entropy', 'max_depth': 16, 'min_samples_split': 14}, Average Accuracy: 0.8443

Best Parameters:
{'criterion': 'entropy', 'max_depth': 16, 'min_samples_split': 2}

Best Accuracy Score: 0.8613
```

```
Decision Tree Replay Model Trained!
Model Performance:
-------------------
- Accuracy: 0.80
- Precision: 0.84
- Recall: 0.80
- F1 Score: 0.82
--------------------------------------------------------------------------------

Classification Report:
-------------------------
              precision    recall  f1-score   support

           0       0.92      0.84      0.88       682
           1       0.40      0.58      0.47       123

    accuracy                           0.80       805
   macro avg       0.66      0.71      0.67       805
weighted avg       0.84      0.80      0.82       805

--------------------------------------------------------------------------------

Confusion Matrix:
----------------
[[574 108]
 [ 52  71]]
```

12. Decision Tree and Random Forest Comparison

After training with both algorithms, it compares performance metrics such as accuracy, precision, recall and F1 score.



Description:

- According to the test results on Decision Tree and Random Forest models, the Random Forest model performed better in accuracy and other performance metrics.
- Both models underperformed on class 1 due to class imbalance.

Decision Tree Performance

- Overall Accuracy: 89%
  - The model showed good accuracy overall.
- Precision: 90%
  - Precision is too high for Class 0.
  - However, the precision for class 1 is 72%, indicating that it does not capture the positive class well enough due to false positive predictions.
- Recall: 97%
  - Recall for Class 0 is close to perfect.
  - However, the recall for class 1 is only 43%, meaning that a minority of the positive instances of the class are hard to catch.
- F1 Score
  - There is a strong score of 94% for Class 0.
  - However, the F1 score for class 1 is 54%, indicating that further improvements are needed on class 1.

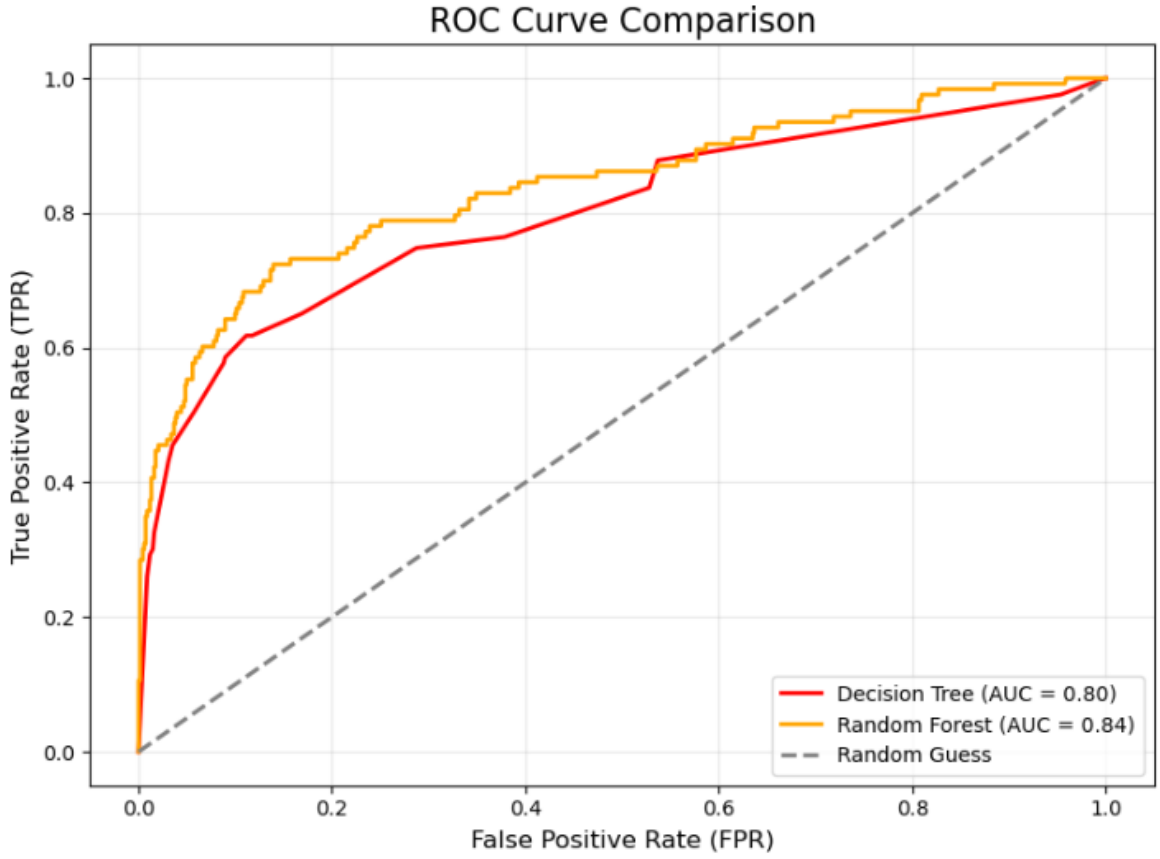
Random Forest Performance

- Overall Accuracy: 90%
  - It showed a 1% higher accuracy than the Decision Tree model.
- Precision: 91%
  - Accuracy for Class 0 is 91%.
  - For class 1, the precision is 79%, indicating that it predicts the positive class more accurately.
- Recall: 98%
  - Recall for class 0 is 98%.
  - Recall for Class 1 is 46%, better than the Decision Tree model but still low.
- F1 Score
  - 94% for class 0 and 58% for class 1, higher than the Decision Tree.

The Decision Tree model focused heavily on the majority class and learned less about the minority class.

13. ROC Curve and AUC Score

The ROC curve and AUC score is a method used to evaluate the performance of classification models



Observations:

- The graph shows the relationship between True Positive Rate (TPR) and False Positive Rate (FPR).
- Decision Tree **(AUC = 0.80)** and Random Forest **(AUC = 0.84)** models are compared.
- The performance of a random prediction model is shown as a gray dashed line (baseline).

Decision Tree

- The ROC curve of the Decision Tree model was lower than that of the Random Forest model.
- **AUC = 0.80** indicates that this model is less successful than Random Forest in separating classes.
- It is observed that the model has difficulties in balancing TPR and FPR and lags behind Random Forest, especially at low FPR levels.

Random Forest

- The curve of the Random Forest model is higher and further away from the baseline compared to the Decision Tree.
- **AUC = 0.84**, indicating that this model is more successful in separating classes.
- The model was able to maintain a high True Positive Rate with a lower False Positive Rate.
- Especially at low FPR values (between 0.0-0.2) Random Forest performs better.

Decision Tree vs. Random Forest

- **Random Forest**
    - It has a higher AUC score than the Decision Tree.
    - This suggests that the model correctly classified more positive examples and made fewer false positive predictions.
- **Decision Tree**
    - As it is a simpler model, it did not generalize as well as Random Forest and its AUC score was lower.

14. Result

In this project, different machine learning algorithms, data preprocessing and hyperparameter optimizations used to improve breast cancer diagnosis prediction were analyzed in detail. Data mining played a critical role in this project and contributed significantly to the process of classifying and analyzing environmental factors. In particular, the use of ensemble methods improved the accuracy and generalizability of breast cancer prediction by leveraging the strengths of different models. Hyperparameter optimization improved the performance of the models, leading to more balanced and accurate predictions. The most important thing I learned during the project is how data preprocessing, model optimization and hyperparameter tuning play a huge role in the success of machine learning models. I also learned that hyperparameters are a very important factor in facilitating model training and preventing overfitting. In this way, I learned that the decision tree in data mining gives more positive results in categorical data, how missing data affects model training, and how data imbalance negatively affects model training.

QUESTION – 2 REPORT

Name: Ramazan Serhat Uygun

Number: 201401049

Subject: Model training with K-means algorithm on data related to students' performance in mathematics course

Dataset:

- **Name:** Students Performance
- **Source:** Kaggle
- **Link:** https://www.kaggle.com/datasets/rabieelkharoua/students-performance-dataset

Methods Used: Normalization, Standardization, Data Exploration, K-Means, Silhouette Score, Elbow Method

1. Objective

The aim of this study is to build a cluster model with a k-means algorithm using the given dataset, to evaluate the performance of the model.

The k-means algorithm is one of the unsupervised learning methods and aims to partition the dataset into subgroups (clusters) with similar characteristics. The algorithm is used to reveal hidden structures in data and to make large and complex datasets more meaningful. K-means creates groups by assigning each data point to the nearest cluster center and calculates the centroids of these groups.

The study analyzes the results by clustering the identified variables using arguments from the dataset.

2. Dataset Details

Description: This dataset focuses on analyzing students' performance in mathematics, reading and writing exams. The dataset is used for educational research, to understand student achievement factors and to identify elements that influence performance.

Dataset Structure:

- **Rows:** 2392
- **Columns:** 15
- **Features:**
  1. **StudentID:** Unique identification number assigned to each student.
  2. **Age:** Age of the student.
  3. **Gender:** Gender of the student.
     a. **1:** Woman
     b. **2:** Man
  4. **Ethnicity:** The ethnic group to which the student belongs.
     a. **0, 1, 2, etc.:** Codes for different groups.
  5. **ParentalEducation:** The highest level of education of one of the student's parents.
     a. **0:** Education level not specified
     b. **1:** High School
     c. **2:** Pre-university
     d. **3:** Bachelor's degree
     e. **4:** Master's degree or higher
  6. **StudyTimeWeekly:** Student's weekly study time (hours).
  7. **Absences:** The number of days the student is absent during the year.
  8. **Tutoring:** Whether the student receives private tutoring.
     a. **1:** Yes
     b. **2:** No
  9. **ParentalSupport:** The level of support the student receives from parents.
     a. **1:** Less
     b. **2:** Medium
     c. **3:** High
  10. **Extracurricular:** Student participation in extra-curricular activities.
     a. **1:** Participates in events
     b. **2:** Does not participate in events

11. **Sports:** Student participation in sports activities.
    a. **1:** Participates in events
    b. **2:** Does not participate in events
12. **Music:** Student participation in music activities.
    a. **1:** Participates in events
    b. **2:** Does not participate in events
13. **Volunteering:** Student participation in volunteering activities.
    a. **1:** Participates in events
    b. **2:** Does not participate in events
14. **GPA:** The student's GPA (a value between 0-4).
15. **GradeClass:** The grade level at which the student is classified according to his/her overall performance.
    a. **1:** Perfect
    b. **2:** Good
    c. **3:** Medium
    d. **4:** Weak

2.1. Why Students Performance Dataset?

I find this data set suitable for the K-means algorithm because it contains numerical and measurable values such as weekly working hours, absences, and grade point average. Since K-means is an effective method for grouping data with similar characteristics, it is possible to determine the academic and behavioral similarities of students with this data set. Thanks to the numerical data, I can clearly calculate the centers of the clusters and assign each student to the nearest cluster. In addition, various dimensions such as duration, absenteeism and success provide a sufficiently diverse and meaningful structure to demonstrate the strength of the algorithm's data analysis and segmentation.

3. Required Libraries

For the model training, I included all the libraries necessary for visualization and the necessary functions to work.

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import silhouette_score
```

4. Loading the Dataset and Initial Exploration

To examine the dataset, I showed the first 5 rows, examined the missing values, and to examine the variables in more detail, I also examined the status of some variables depending on the target variable.

```
file_path = 'Student_performance_data.csv'
data = pd.read_csv(file_path)

print("First 5 Rows of the Data Set:")
print("-----------------------------")
print(data.head())
print("-------------------------------------------------------------------------------------------")

print("\nGeneral Information of the Data Set:")
print("----------------------------------------")
print(data.info())
print("-------------------------------------------------------------------------------------------")

print("\nNumber of Missing Values:")
print("--------------------------")
print(data.isnull().sum())
print("-------------------------------------------------------------------------------------------")
```

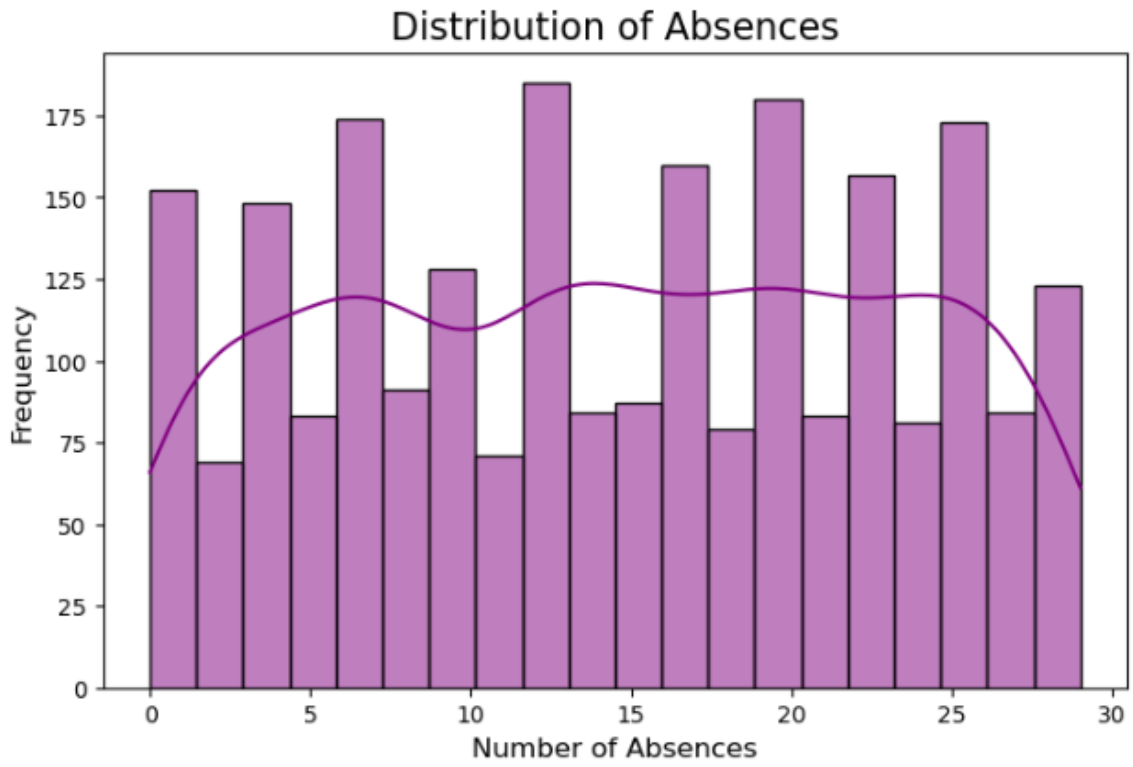
Output:

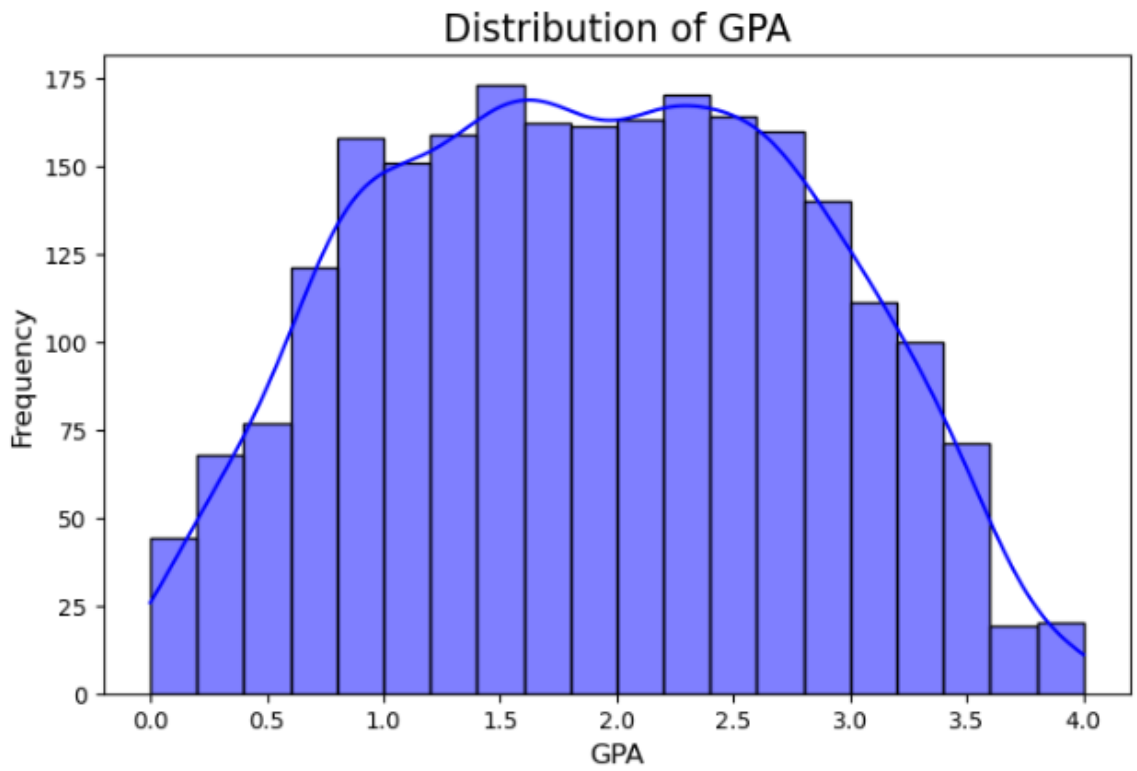
```
First 5 Rows of the Data Set:
------------------------------
   StudentID  Age  Gender  Ethnicity  ParentalEducation  StudyTimeWeekly  \
0       1001   17       1          0                  2        19.833723
1       1002   18       0          0                  1        15.408756
2       1003   15       0          2                  3         4.210570
3       1004   17       1          0                  3        10.028829
4       1005   17       1          0                  2         4.672495

   Absences  Tutoring  ParentalSupport  Extracurricular  Sports  Music  \
0         7         1                2                0       0      1
1         0         0                1                0       0      0
2        26         0                2                0       0      0
3        14         0                3                1       0      0
4        17         1                3                0       0      0

   Volunteering       GPA  GradeClass
0             0  2.929196         2.0
1             0  3.042915         1.0
2             0  0.112602         4.0
3             0  2.054218         3.0
4             0  1.288061         4.0
-------------------------------------------------------------------------------------------
```

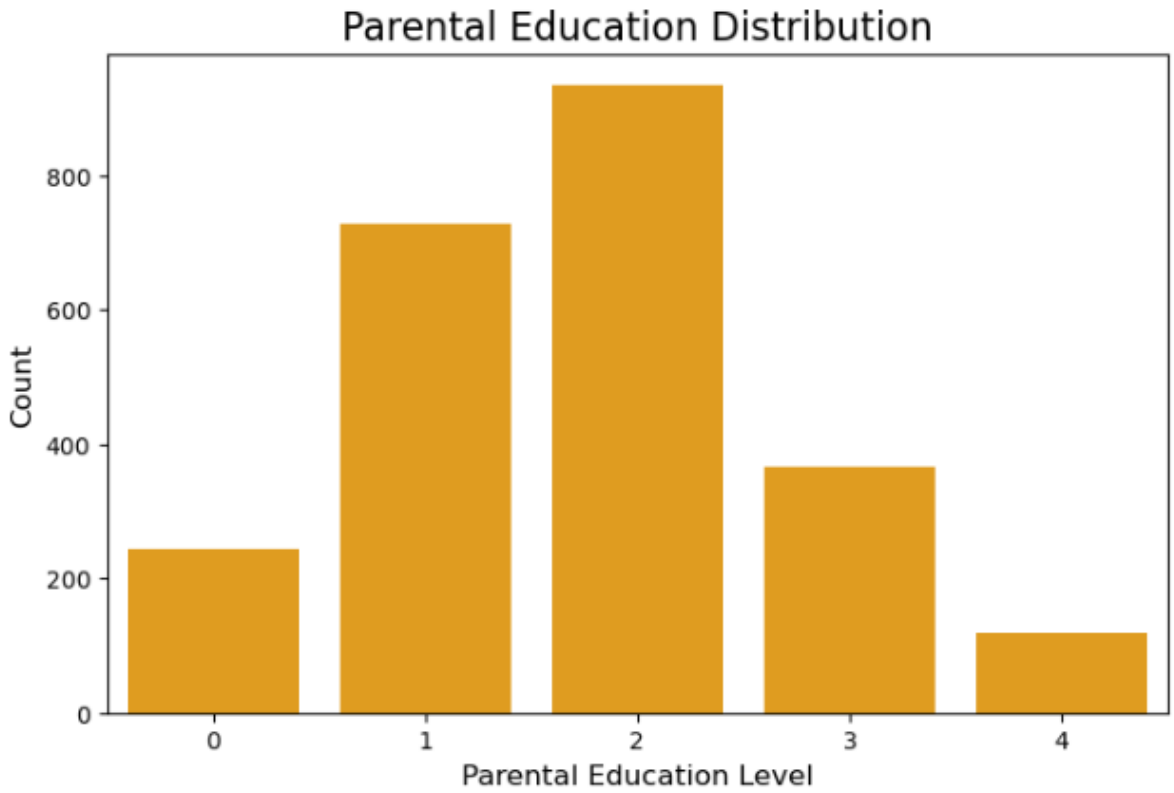
```
General Information of the Data Set:
-------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2392 entries, 0 to 2391
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   StudentID          2392 non-null   int64
 1   Age                2392 non-null   int64
 2   Gender             2392 non-null   int64
 3   Ethnicity          2392 non-null   int64
 4   ParentalEducation  2392 non-null   int64
 5   StudyTimeWeekly    2392 non-null   float64
 6   Absences           2392 non-null   int64
 7   Tutoring           2392 non-null   int64
 8   ParentalSupport    2392 non-null   int64
 9   Extracurricular    2392 non-null   int64
 10  Sports             2392 non-null   int64
 11  Music              2392 non-null   int64
 12  Volunteering       2392 non-null   int64
 13  GPA                2392 non-null   float64
 14  GradeClass         2392 non-null   float64
dtypes: float64(3), int64(12)
memory usage: 280.4 KB
None
-------------------------------------------------------------------------------------

Number of Missing Values:
---------------------------
StudentID            0
Age                  0
Gender               0
Ethnicity            0
ParentalEducation    0
StudyTimeWeekly      0
Absences             0
Tutoring             0
ParentalSupport      0
Extracurricular      0
Sports               0
Music                0
Volunteering         0
GPA                  0
GradeClass           0
dtype: int64
-------------------------------------------------------------------------------------
```

5. Data Visualization

I found it appropriate to visualize some of the variables in order to understand the data set and examine it in depth. Thanks to this visualization, I was able to examine the distribution and frequency of certain variables in terms of data.


Distribution of Absences
Frequency
Number of Absences
0
25
50
75
100
125
150
175
0
5
10
15
20
25
30


Distribution of GPA
Frequency
GPA
0
25
50
75
100
125
150
175
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0



By making a few more data visualizations like the data visualizations above, I had information about the data set before I started training the k-means algorithm model. In this way, I determined the variables I would use for model training.

6. Standardization and Normalization

Since the clustering of numerical data will be used in the K-means algorithm, I am preparing to start model training by selecting 3 numerical values in this selected data set and performing normalization and standardization processes to obtain more consistent balanced data.

Variables I chose:

- **StudyTimeWeekly:** Student's weekly study time (hours).
- **Absences:** The number of days the student is absent during the year.
- **GPA:** The student's GPA (a value between 0-4).

6.1. Standardization

```
columns_to_standardize = ['StudyTimeWeekly', 'Absences', 'GPA']

scaler = StandardScaler()
data_standardized = data.copy()
data_standardized[columns_to_standardize] = scaler.fit_transform(data[columns_to_standardize])

print("First 5 rows of the standardized dataset:")
print("----------------------------------------")
print(data_standardized[columns_to_standardize].head())
```

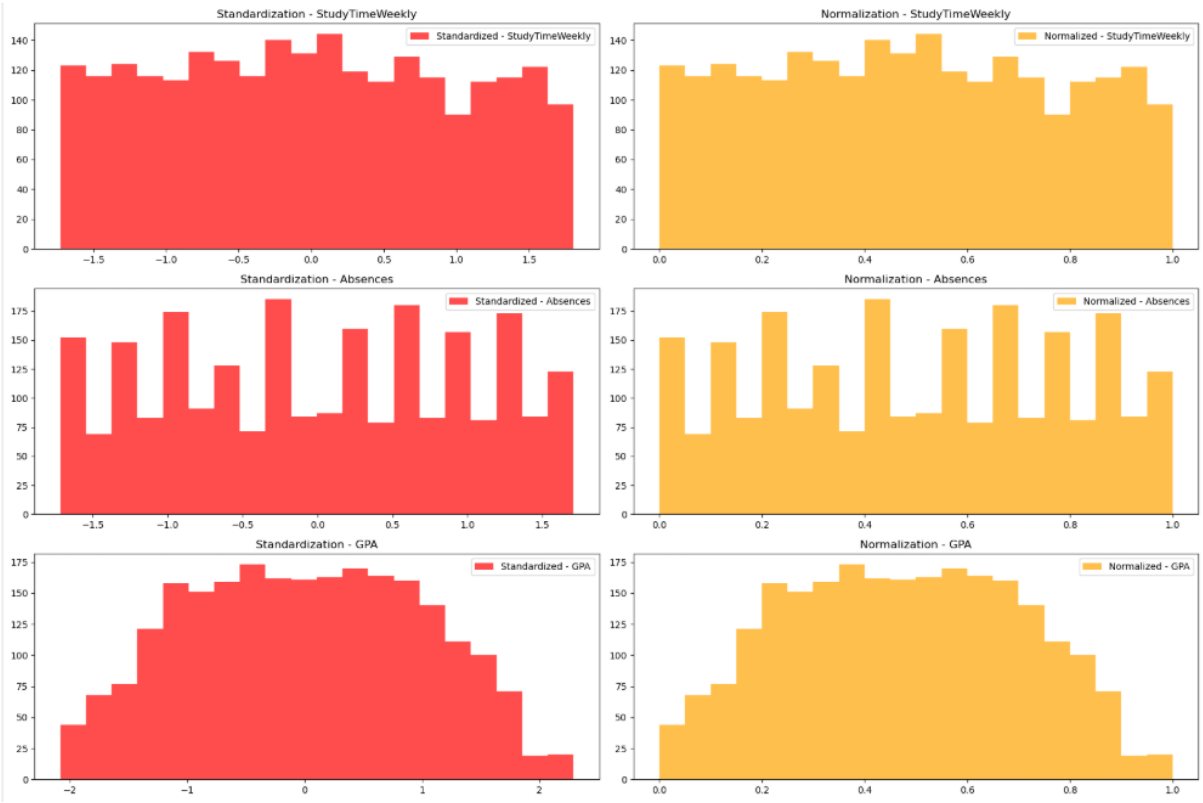
```
First 5 rows of the standardized dataset:
----------------------------------------
   StudyTimeWeekly  Absences       GPA
0         1.780336 -0.890822  1.118086
1         0.997376 -1.717694  1.242374
2        -0.984045  1.353542 -1.960277
3         0.045445 -0.063951  0.161790
4        -0.902311  0.290422 -0.675573
```

6.2. Normalization

```
columns_to_normalize = ['StudyTimeWeekly', 'Absences', 'GPA']

normalizer = MinMaxScaler()
data_normalized = data.copy()
data_normalized[columns_to_normalize] = normalizer.fit_transform(data[columns_to_normalize])

print("First 5 rows of the normalized dataset:")
print("----------------------------------------")
print(data_normalized[columns_to_normalize].head())
```

```
First 5 rows of the normalized dataset:
----------------------------------------
   StudyTimeWeekly  Absences       GPA
0         0.992773  0.241379  0.732299
1         0.771270  0.000000  0.760729
2         0.210718  0.896552  0.028151
3         0.501965  0.482759  0.513555
4         0.233840  0.586207  0.322015
```

After performing these operations, I performed data visualization to examine the status of the variables.



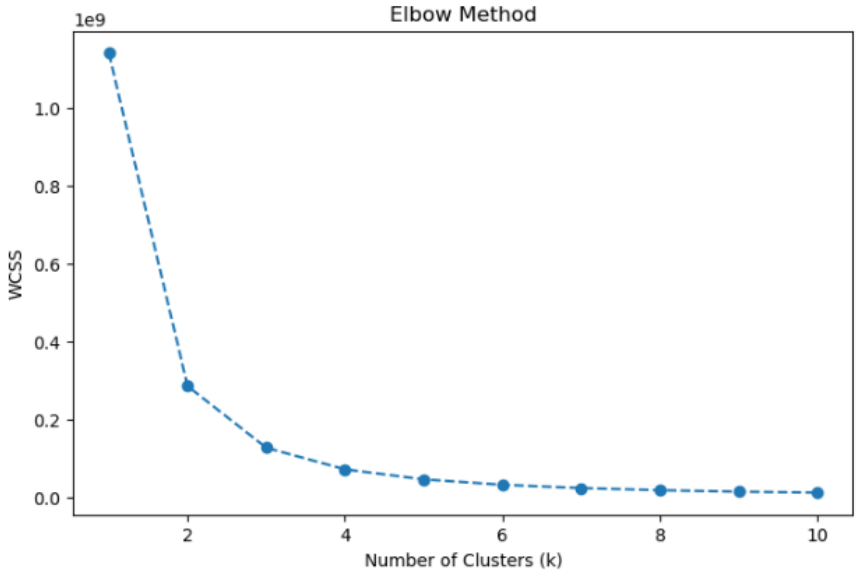
7. Elbow Method and Silhouette Score

In order to apply the k-means algorithm, I need to define a value of k. We use two methods for this. These methods are Elbow Method and Silhouette Score.

7.1. Elbow Method

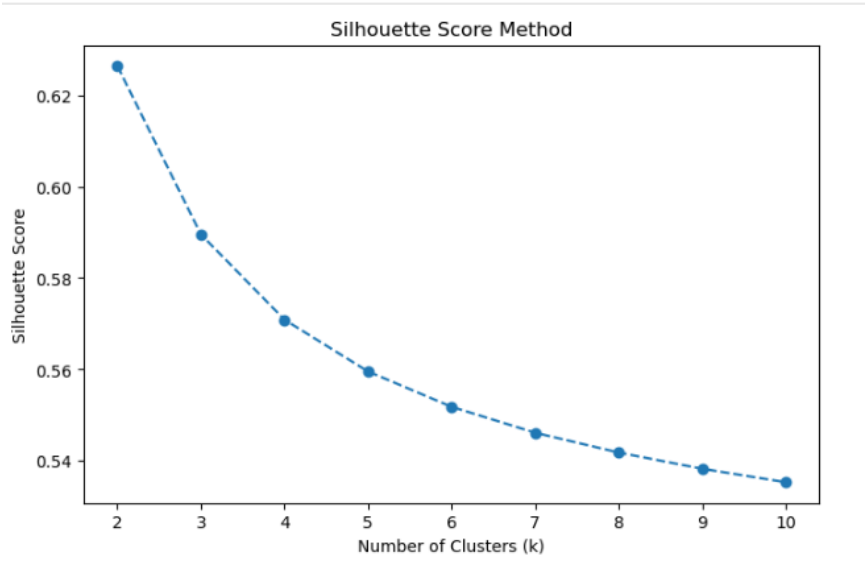
I use the Elbow method to determine the correct number of clusters in the K-means algorithm because the algorithm does not work without knowing the number of clusters (k) at the beginning. The Elbow method allows me to calculate the total intrinsic variance

(inertia) for different values of k and see how this value changes with k. Initially, the variance decreases rapidly as k increases because more clusters represent the data better. But after a certain point (the elbow point), the decrease in variance slows down and the benefit of increasing k decreases. This point provides a critical clue for determining the optimal number of clusters. This way, I avoid unnecessarily over- or under-clustering the dataset and can perform a more efficient analysis.



7.2. Silhouette Score

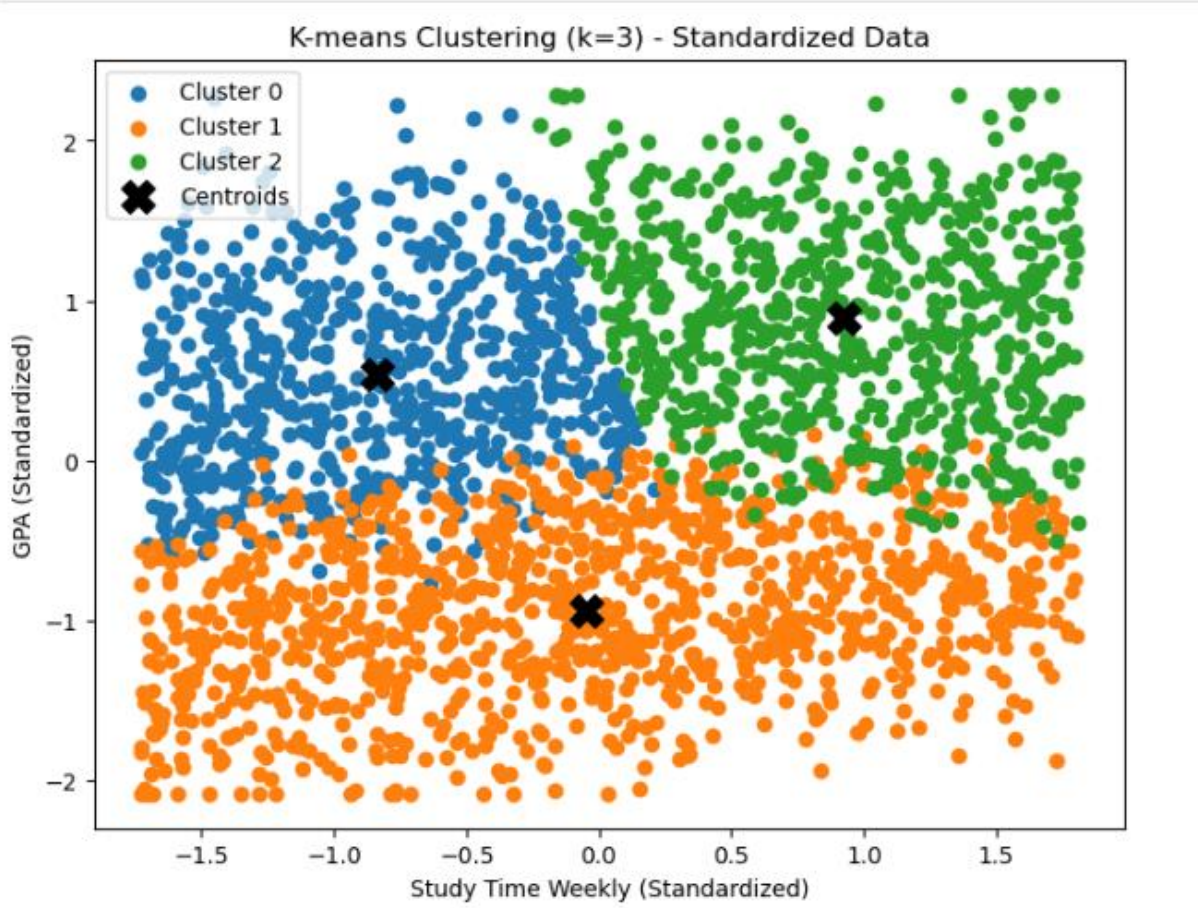
I use the Silhouette score in the same way I use the Elbow Method to evaluate how successful the clusters I created in the K-means algorithm are. This metric generates a score for each data point based on its proximity to its cluster and its distance from other clusters. A score close to 1 indicates that data points are assigned to the correct clusters and are clearly separated from other clusters. If the score is close to 0, I know that there is ambiguity between clusters, and if it is negative, I know that some data points are incorrectly clustered. This metric is a very useful guide for me to both improve the clustering result and to determine the appropriate number of clusters.



Using these methods, I found the value of k that I need to use in the K-means algorithm. I set k to 3 as the value that creates an elbow on the Elbow method and the value that is closest to 1 on the Silhouette Score. In this way, I can create the most optimized clustering process.

8. Applying the K-Means Algorithm

I performed the data clustering process on the k value I determined, and in this way, I achieved the most optimized clustering.



9. Result

In this project, I investigated the effects of weekly working hours, number of absences during the year and students' grade point average on students' performance in mathematics using k-means clustering algorithm. In this project, I learned the working logic of the k-means algorithm, the benefits of normalizing and standardizing the data and training, the methods that should be used to find the k value (Elbow Method and Silhouette Score), and the working logic. Finally, I performed the necessary processing and clustered the determined variables (StudyTimeWeekly, Absences and GPA).

QUESTION – 3 REPORT

Name: Ramazan Serhat Uygun

Number: 201401049

Subject: Linear Regression, Ridge Regression and Lasso Regression Model Training on diamond price data

Dataset:

- **Name:** Diamonds
- **Source:** Kaggle
- **Link:** https://www.kaggle.com/code/karnikakapoor/diamond-price-prediction/notebook

Methods Used: Normalization, Standardization, Data Exploration, Linear Regression, Ridge Regression, Lasso Regression, Hyperparameters (GridSearchCV and RandomizedSearchCV), Random Forest

1. Objective

The aim of this study is to train the model using linear regression, ridge regression and lasso regression algorithms using the specified dataset and to evaluate the performance of the model by comparing it with the random forest algorithm.

Linear, Ridge and Lasso regressions are methods used to model the relationship between dependent and independent variables in data sets. Linear regression, as a basic method, expresses the dependent variable as a linear combination of independent variables and assumes that all variables are used equally in the model. Ridge regression reduces the coefficients by adding a penalty term to reduce the high variance in the model and is robust to the problem of multicollinearity. Lasso regression, on the other hand, obtains a simpler and more interpretable model by both selecting variables and setting the coefficients closer to zero. While Ridge keeps all variables in the model, Lasso reduces the effect of unimportant variables to zero and thus provides better generalization.

The study aims to provide model training based on diamond prices using regressions and the random forest algorithm.

2. Dataset Details

Description: This classic dataset contains the prices and other attributes of almost 54,000 diamonds. There are 10 attributes included in the dataset including the target ie. price.

Dataset Structure:

- **Rows:** 53940
- **Columns:** 10
- **Features:**
  1. **Carat:** Carat weight of the diamond
  2. **Cut:** Describe cut quality of the diamond.
     - **a.** Fair
     - **b.** Good
     - **c.** Very Good
     - **d.** Premium
     - **e.** Ideal
  3. **Color:** Color of the diamond.
     - **a.** G
     - **b.** E
     - **c.** F
     - **d.** H
     - **e.** D
  4. **Clarity:** How obvious inclusions are within the diamond.
     - **a.** SI1
     - **b.** VS2
     - **c.** SI2
     - **d.** VS1
     - **e.** VVS2
  5. **Depth:** The height of a diamond, measured from the cult to the table, divided by its average girdle diameter.

6. **Table:** The width of the diamond's table expressed as a percentage of its average diameter.
7. **Price:** The price of the diamond.
8. **x:** length mm
9. **y:** width mm
10. **z:** depth mm

2.1. Why Diamonds Dataset?

I used the Diamonds dataset because it is rich in different characteristics of diamonds (carat, color, clarity, price, etc.) and provides an ideal sample for testing regression models. For the problem of diamond price forecasting, this dataset allowed me to analyze linear and non-linear relationships between the independent variables (carat, color, clarity, etc.) and the dependent variable (price). At the same time, the fact that this dataset consists of several variables provided a strong basis to compare different regression methods (Linear, Ridge, Lasso) and see which model performs better.

3. Required Libraries

For the model training, I included all the libraries necessary for visualization and the necessary functions to work.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder, MinMaxScaler
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

4. Loading the Dataset and Initial Exploration

To examine the dataset, I showed the first 5 rows, examined the missing values, and to examine the variables in more detail, I also examined the status of some variables depending on the target variable.

```
file_path = 'diamonds.csv'
diamonds = pd.read_csv(file_path)

print(diamonds.info())
print(diamonds.head())
print(diamonds.describe())
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  53940 non-null  int64
 1   carat       53940 non-null  float64
 2   cut         53940 non-null  object
 3   color       53940 non-null  object
 4   clarity     53940 non-null  object
 5   depth       53940 non-null  float64
 6   table       53940 non-null  float64
 7   price       53940 non-null  int64
 8   x           53940 non-null  float64
 9   y           53940 non-null  float64
 10  z           53940 non-null  float64
dtypes: float64(6), int64(2), object(3)
memory usage: 4.5+ MB
None
```

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y
0	1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98
1	2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84
2	3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07
3	4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23
4	5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35

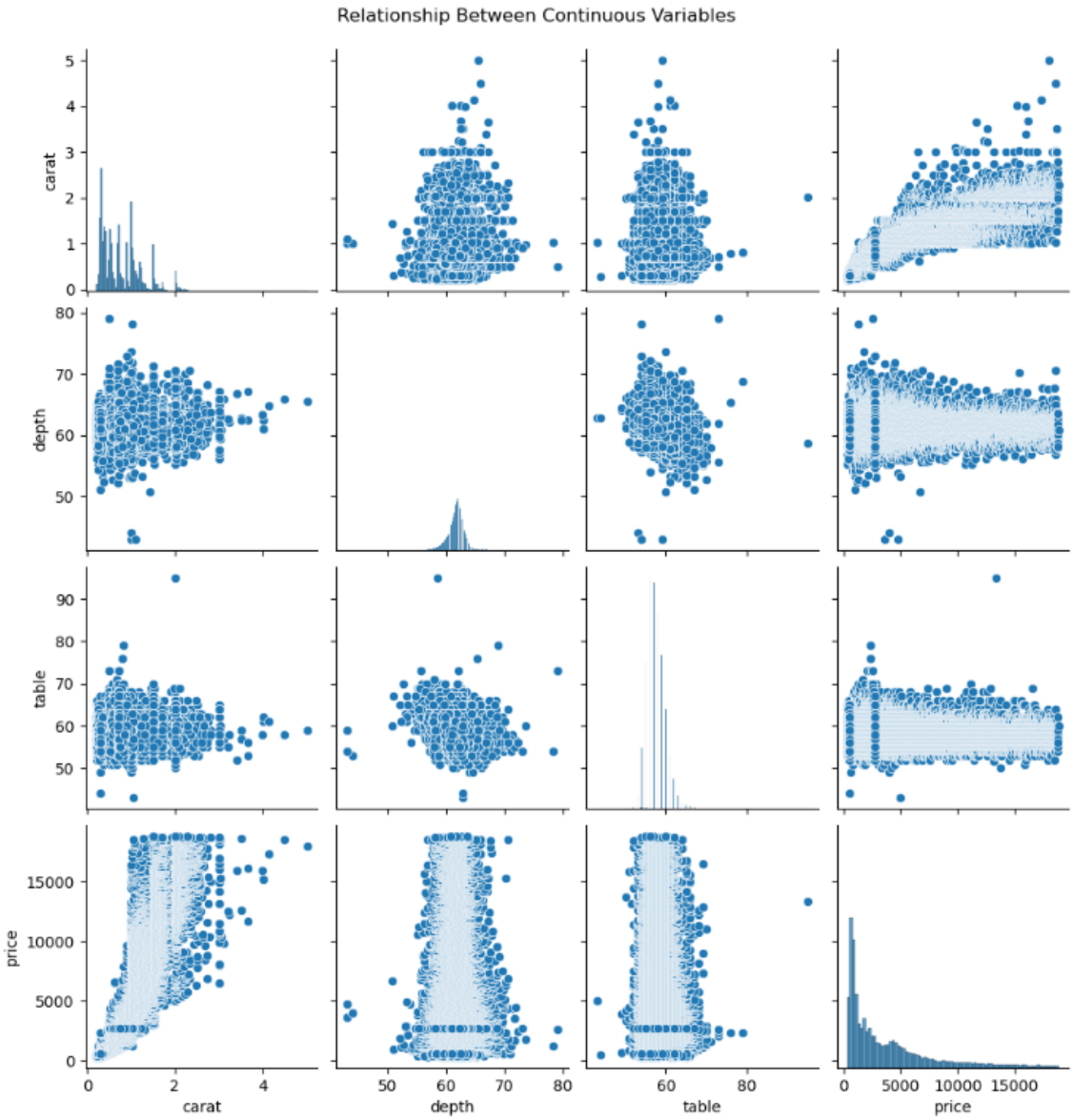
	z
0	2.43
1	2.31
2	2.31
3	2.63
4	2.75

	Unnamed: 0	carat	depth	table	price
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000
mean	26970.500000	0.797940	61.749405	57.457184	3932.799722
std	15571.281097	0.474011	1.432621	2.234491	3989.439738
min	1.000000	0.200000	43.000000	43.000000	326.000000
25%	13485.750000	0.400000	61.000000	56.000000	950.000000
50%	26970.500000	0.700000	61.800000	57.000000	2401.000000
75%	40455.250000	1.040000	62.500000	59.000000	5324.250000
max	53940.000000	5.010000	79.000000	95.000000	18823.000000

	x	y	z
count	53940.000000	53940.000000	53940.000000
mean	5.731157	5.734526	3.538734
std	1.121761	1.142135	0.705699
min	0.000000	0.000000	0.000000
25%	4.710000	4.720000	2.910000
50%	5.700000	5.710000	3.530000
75%	6.540000	6.540000	4.040000
max	10.740000	58.900000	31.800000

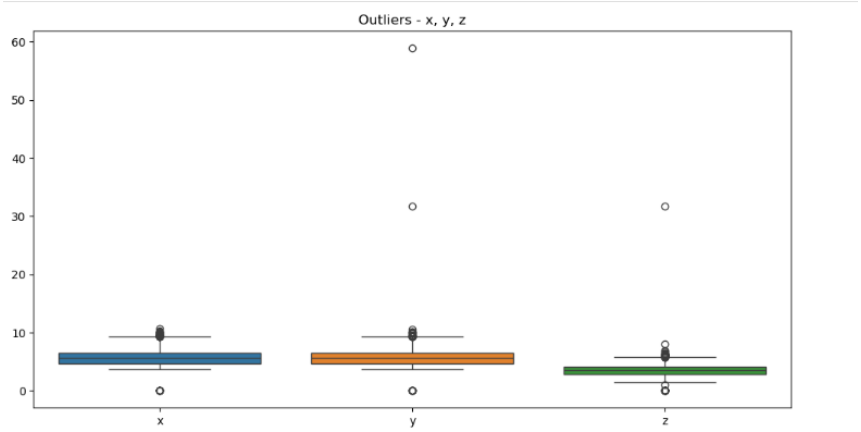
5. Data Visualization

In order to understand the dataset and explore it in depth, I found it appropriate to visualize some variables, which allowed me to examine the distribution of certain variables among each other from a data perspective.



6. Cleaning Outliers

In order to work on the data, we first need to identify outliers and remove those data from the outlier.



After detecting outliers in x, y and z variables, I performed outlier removal.

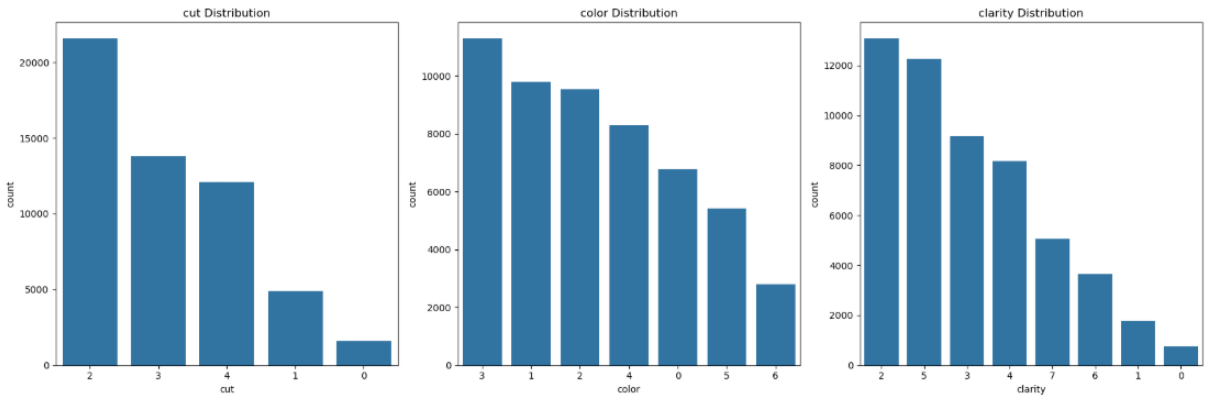
```
diamonds_cleaned = diamonds[(diamonds['x'] > 0) & (diamonds['y'] > 0) & (diamonds['z'] > 0)]
print(diamonds_cleaned.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 53920 entries, 0 to 53939
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  53920 non-null  int64  
 1   carat       53920 non-null  float64
 2   cut         53920 non-null  object 
 3   color       53920 non-null  object 
 4   clarity     53920 non-null  object 
 5   depth       53920 non-null  float64
 6   table       53920 non-null  float64
 7   price       53920 non-null  int64  
 8   x           53920 non-null  float64
 9   y           53920 non-null  float64
 10  z           53920 non-null  float64
dtypes: float64(6), int64(2), object(3)
memory usage: 4.9+ MB
None
```

7. Transformation of Categorical Variables

In order to use categorical data in regression, we need to make it numeric.

```
label_encoders = {}
categorical_columns = ['cut', 'color', 'clarity']

for col in categorical_columns:
    le = LabelEncoder()
    diamonds_cleaned.loc[:, col] = le.fit_transform(diamonds_cleaned[col])  # .loc kullanımı
    label_encoders[col] = le

plt.figure(figsize=(18, 6))
for i, col in enumerate(categorical_columns, 1):
    plt.subplot(1, 3, i)
    sns.countplot(data=diamonds_cleaned, x=col, order=diamonds_cleaned[col].value_counts().index)
    plt.title(f"{col} Distribution")
plt.tight_layout()
plt.show()
```

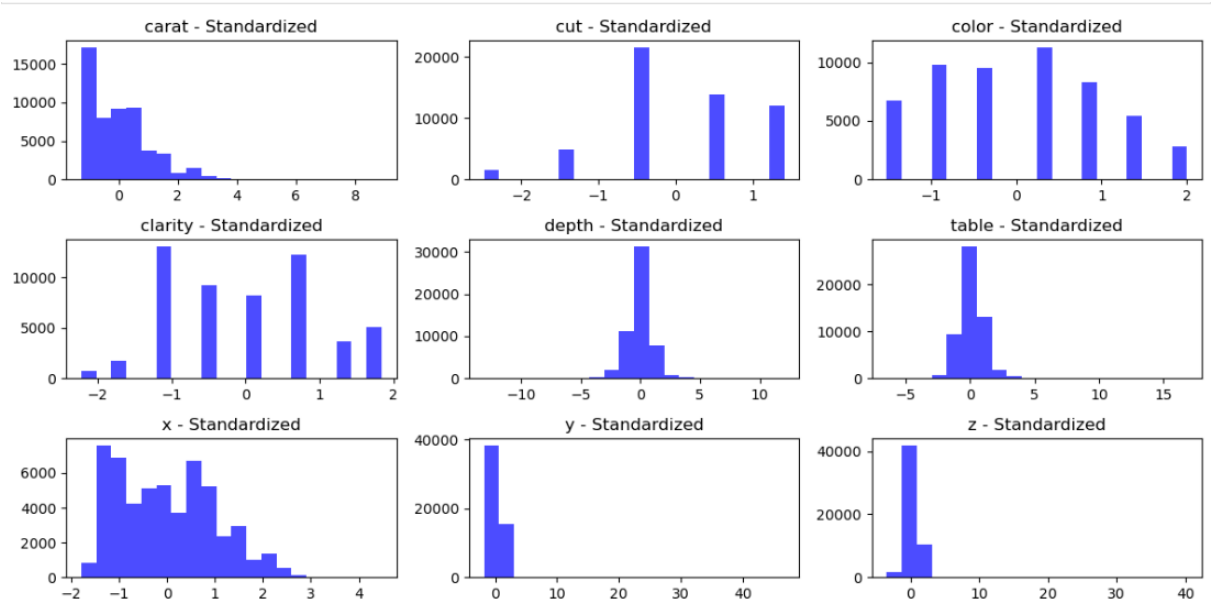


8. Standardization and Normalization

I normalize and standardize the target Price variable and all variables in order to model the regression more optimized.

8.1. Standardization

```
features = diamonds_cleaned.drop(['price', 'Unnamed: 0'], axis=1)
target = diamonds_cleaned['price']

scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)
features_scaled_df = pd.DataFrame(features_scaled, columns=features.columns)

num_columns = len(features.columns)
num_rows = math.ceil(num_columns / 3)

plt.figure(figsize=(12, 6))
for i, col in enumerate(features.columns, 1):
    plt.subplot(num_rows, 3, i)
    plt.hist(features_scaled_df[col], bins=20, color='blue', alpha=0.7)
    plt.title(f"{col} - Standardized")
plt.tight_layout()
plt.show()
```

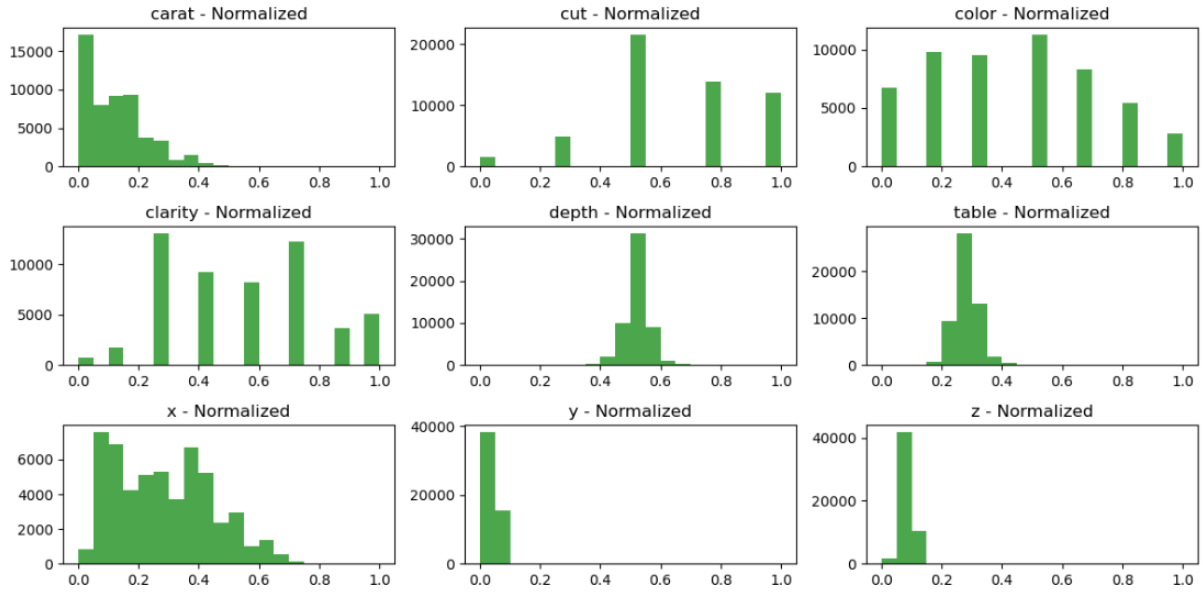


8.2. Normalization

```
scaler = MinMaxScaler()
features_normalized = scaler.fit_transform(features)
features_normalized_df = pd.DataFrame(features_normalized, columns=features.columns)

print(features_normalized_df.head())

plt.figure(figsize=(12, 6))
for i, col in enumerate(features.columns, 1):
    plt.subplot(math.ceil(len(features.columns) / 3), 3, i)
    plt.hist(features_normalized_df[col], bins=20, color='green', alpha=0.7)
    plt.title(f"{col} - Normalized")
plt.tight_layout()
plt.show()
```

	carat	cut	color	clarity	depth	table	x	y	\
0	0.006237	0.50	0.166667	0.428571	0.513889	0.230769	0.031384	0.005433	
1	0.002079	0.75	0.166667	0.285714	0.466667	0.346154	0.022825	0.002898	
2	0.006237	0.25	0.166667	0.571429	0.386111	0.423077	0.045649	0.007063	
3	0.018711	0.75	0.833333	0.714286	0.538889	0.288462	0.067047	0.009960	
4	0.022869	0.25	1.000000	0.428571	0.563889	0.288462	0.087019	0.012133	

	z
0	0.044256
1	0.040351
2	0.040351
3	0.050765
4	0.054670



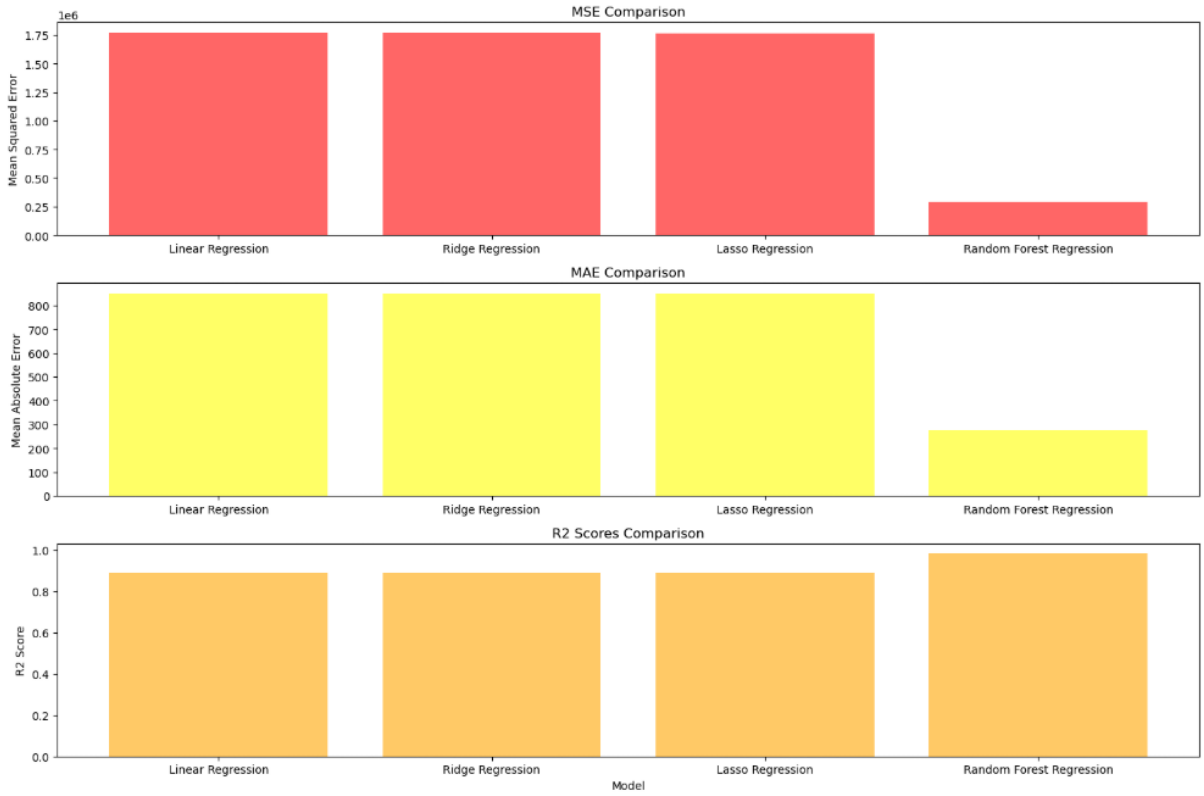
9. Modeling and Performance Comparison

I used the following metrics to evaluate the model performance:

- **R2 Score:** Measures how much the model explains the variance of the target variable.
- **Mean Absolute Error (MAE):** It is the average of the absolute errors between actual values and predicted values.
- **Mean Squared Error (MSE):** The mean squared error between actual and predicted values. Larger errors are penalized more.

```
X_train, X_test, y_train, y_test = train_test_split(features_scaled_df, target, test_size=0.2, random_state=42)

models = {
    "Linear Regression": LinearRegression(),
    "Ridge Regression": Ridge(alpha=1.0),
    "Lasso Regression": Lasso(alpha=0.5, max_iter=5000),
    "Random Forest Regression": RandomForestRegressor(n_estimators=10, random_state=42)
}

results = {"Model": [], "MSE": [], "MAE": [], "R2": []}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    results["Model"].append(name)
    results["MSE"].append(mse)
    results["MAE"].append(mae)
    results["R2"].append(r2)
```

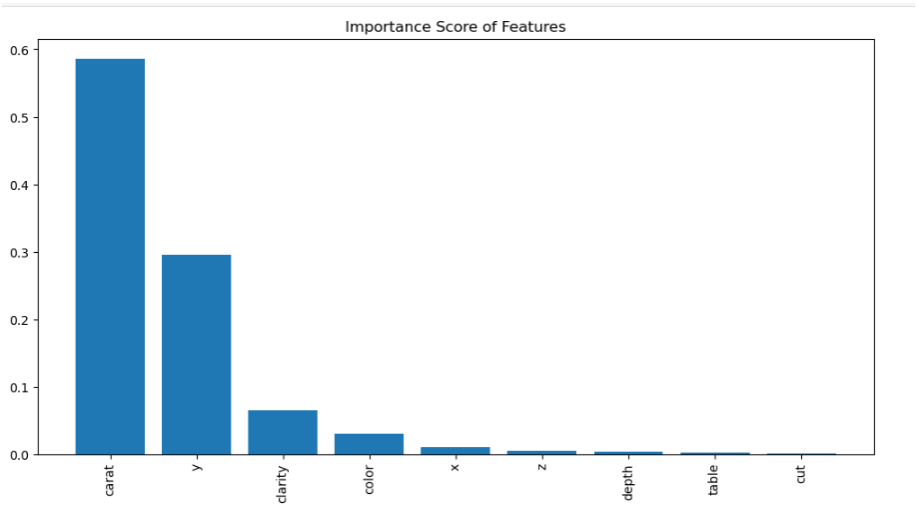
```
plt.figure(figsize=(15, 10))

plt.subplot(3, 1, 1)
plt.bar(results["Model"], results["MSE"], color='red', alpha=0.6)
plt.title("MSE Comparison")
plt.ylabel("Mean Squared Error")

plt.subplot(3, 1, 2)
plt.bar(results["Model"], results["MAE"], color='yellow', alpha=0.6)
plt.title("MAE Comparison")
plt.ylabel("Mean Absolute Error")

plt.subplot(3, 1, 3)
plt.bar(results["Model"], results["R2"], color='orange', alpha=0.6)
plt.title("R2 Scores Comparison")
plt.ylabel("R2 Score")
plt.xlabel("Model")

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



I completed linear regression, Ridge regression, Lasso regression and random forest regression calculations with R2 Score, MAE, MSE metrics and examined the results before the hyperparameters were applied.

10. Importance Score of Features

I calculated the importance score of the attributes because I wanted to understand the impact of the independent variables in the dataset on the dependent variable (price). Since the Diamonds dataset has many attributes such as carat, color, clarity, etc., seeing the contribution of these attributes on the price prediction allowed me to better interpret and optimize the model. By analyzing the importance scores of the attributes, I was able to identify which variables have more impact on price and eliminate the less influential ones to reduce the complexity of the model.



11. Hyperparameter Optimization for All Models

I performed hyperparameter optimization for all models because I wanted to improve the performance of each model and find the best combination of parameters for the dataset. For a problem like price prediction on the Diamonds dataset, the right hyperparameters make the model more effective and reduce overall errors. Hyperparameters, such as the penalty parameter in Ridge and Lasso regressions, increase the generalization ability of the model while reducing the risk of over-learning. In this process, I compared the models by experimenting with different values of hyperparameters and chose the parameters that provided the best results for each.

GridSearchCV for Ridge and Lasso

I used GridSearchCV because I wanted to systematically search for the optimal hyperparameters for each model and compare their performance. This method trains and validates the model for each combination using specified ranges of hyperparameters and selects the best parameters. For the price prediction problem on the Diamonds dataset, I used GridSearchCV to optimize the penalty parameter of models such as Ridge and Lasso, thereby minimizing the risk of overlearning and under learning. Furthermore, this method cross-validated and tested the model on different data partitions, allowing me to better assess its overall performance.

```
param_grid = {'alpha': [0.1, 0.5, 1.0, 5.0, 10.0]}

ridge_model = Ridge()
grid_search_ridge = GridSearchCV(estimator=ridge_model, param_grid=param_grid, scoring='r2', cv=5)
grid_search_ridge.fit(X_train, y_train)

ridge_best_model = grid_search_ridge.best_estimator_
ridge_pred = ridge_best_model.predict(X_test)
ridge_mse = mean_squared_error(y_test, ridge_pred)
ridge_mae = mean_absolute_error(y_test, ridge_pred)

print("Best Ridge parameters:", grid_search_ridge.best_params_)
print("Ridge - Best R2 score:", grid_search_ridge.best_score_)
print(f"Ridge - MSE: {ridge_mse:.2f}")
print(f"Ridge - MAE: {ridge_mae:.2f}")

lasso_model = Lasso(max_iter=5000)
grid_search_lasso = GridSearchCV(estimator=lasso_model, param_grid=param_grid, scoring='r2', cv=5)
grid_search_lasso.fit(X_train, y_train)

lasso_best_model = grid_search_lasso.best_estimator_
lasso_pred = lasso_best_model.predict(X_test)
lasso_mse = mean_squared_error(y_test, lasso_pred)
lasso_mae = mean_absolute_error(y_test, lasso_pred)

print("Best Lasso parameters:", grid_search_lasso.best_params_)
print("Lasso - Best R2 score:", grid_search_lasso.best_score_)
print(f"Lasso - MSE: {lasso_mse:.2f}")
print(f"Lasso - MAE: {lasso_mae:.2f}")
```

```
Best Ridge parameters: {'alpha': 10.0}
Ridge - Best R2 score: 0.8691263120882231
Ridge - MSE: 1770761.04
Ridge - MAE: 852.16
Best Lasso parameters: {'alpha': 5.0}
Lasso - Best R2 score: 0.8830429896665555
Lasso - MSE: 1735591.06
Lasso - MAE: 856.83
```

RandomizedSearchCV for Random Forest

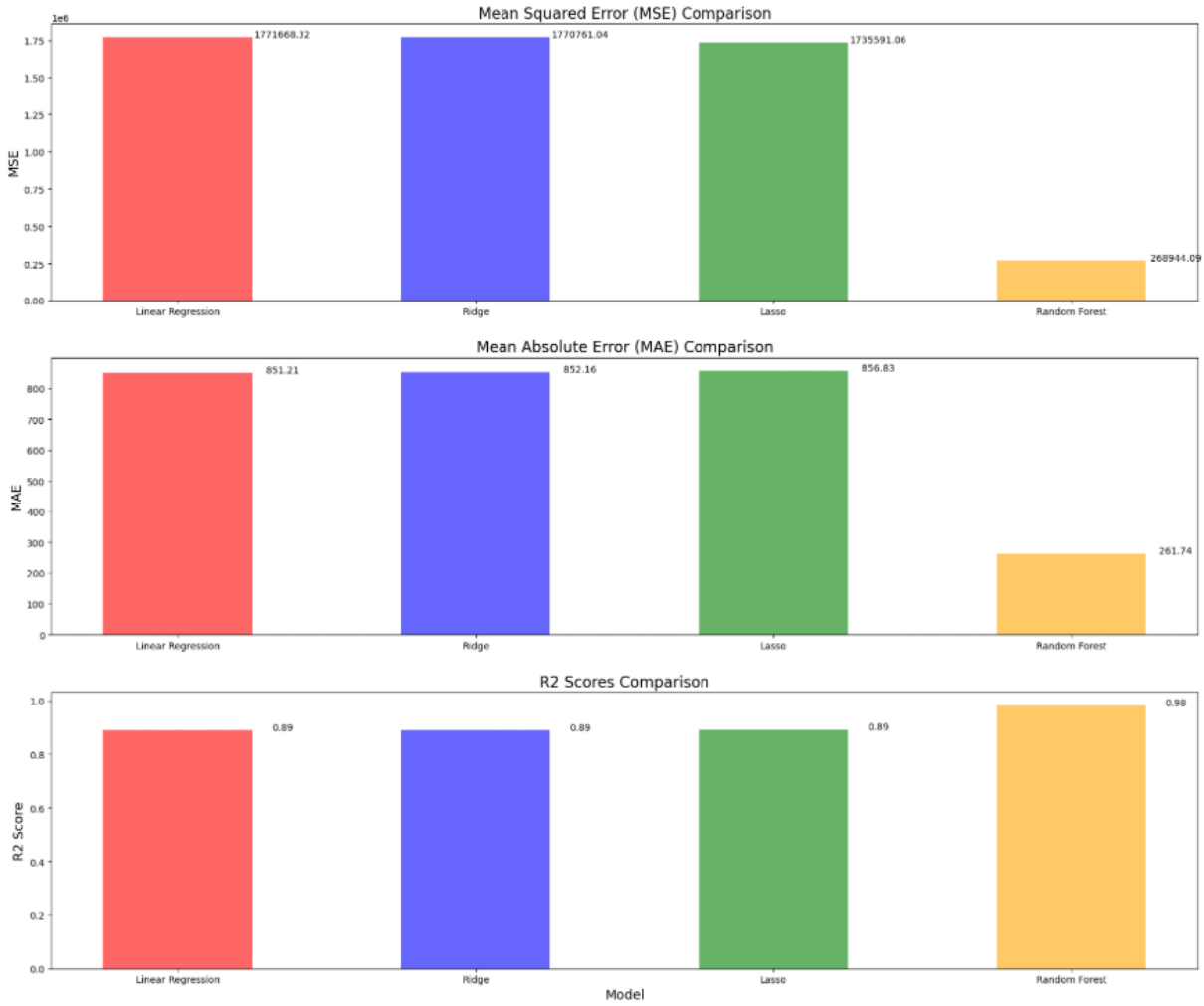
I used RandomizedSearchCV for the Random Forest model because it allows me to perform hyperparameter optimization more quickly and efficiently. Compared to GridSearchCV, it only evaluates a limited number of combinations by randomly sampling within the specified hyperparameter range, which saves time, especially for large parameter ranges. For the price prediction problem on the Diamonds dataset, I aimed to improve the accuracy of the Random Forest model by optimizing its hyperparameters (**n_estimators, max_depth, min_samples_split**). In the process, I was able to quickly scan a wider range and find parameter combinations that provide high performance, reducing the processing time and increasing the generalization capability of the model.

```
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
linear_pred = linear_model.predict(X_test)

ridge_pred = grid_search_ridge.best_estimator_.predict(X_test)
lasso_pred = grid_search_lasso.best_estimator_.predict(X_test)
rf_pred = random_search_rf.best_estimator_.predict(X_test)

optimized_results = {
    "Model": ["Linear Regression", "Ridge", "Lasso", "Random Forest"],
    "Best Parameters": [
        None,
        grid_search_ridge.best_params_,
        grid_search_lasso.best_params_,
        random_search_rf.best_params_
    ],
    "MSE": [
        mean_squared_error(y_test, linear_pred),
        mean_squared_error(y_test, ridge_pred),
        mean_squared_error(y_test, lasso_pred),
        mean_squared_error(y_test, rf_pred)
    ],
    "MAE": [
        mean_absolute_error(y_test, linear_pred),
        mean_absolute_error(y_test, ridge_pred),
        mean_absolute_error(y_test, lasso_pred),
        mean_absolute_error(y_test, rf_pred)
    ],
    "R2 Score": [
        r2_score(y_test, linear_pred),
        r2_score(y_test, ridge_pred),
        r2_score(y_test, lasso_pred),
        r2_score(y_test, rf_pred)
    ]
}

results_df = pd.DataFrame(optimized_results)
print(results_df)
```

```
              Model                                    Best Parameters  \
0  Linear Regression                                               None
1              Ridge                                  {'alpha': 10.0}
2              Lasso                                   {'alpha': 5.0}
3      Random Forest  {'n_estimators': 200, 'min_samples_split': 5, ...

            MSE         MAE  R2 Score
0  1.771668e+06  851.209989  0.889491
1  1.770761e+06  852.160308  0.889548
2  1.735591e+06  856.825580  0.891742
3  2.689441e+05  261.737651  0.983224
```

12. Visualizing Results

This study analyzed the performance of different regression models used to forecast diamond prices. Hyperparameter optimization played an important role in improving the performance of the models. In particular, the Random Forest model showed superior performance in complex data structures.



13. Result

In this study, I used four different regression models (Linear Regression, Ridge Regression, Lasso Regression, and Random Forest Regression) to predict diamond prices and comprehensively analyzed the performance of the models. I used metrics such as R2 Score, Mean Absolute Error (MAE), and Mean Squared Error (MSE) to evaluate the models. For the Ridge and Lasso models, I optimized the hyperparameters with GridSearchCV to observe the impact of regularization power on model performance. By optimizing the Random Forest Regression model with RandomizedSearchCV, I achieved the best performance on complex data structures. I realized that simpler models, such as Linear Regression, may be insufficient for learning complex relationships. In this study, I experienced how important data preprocessing, model selection, hyperparameter optimization, and performance evaluation processes are, and concluded that these processes are the cornerstones of building powerful machine learning models.