QUESTION 1

Name : Ömer Polat

No : 201401057

Subject : Comparative Analysis of Decision Tree Classifier with Other Machine Learning Algorithms on Public Datasets

Dataset Name : Red Wine quality dataset.

Methods : Decision Tree, Exploratory Data Analysis (EDA), Class Imbalance Handling (SMOTE - Synthetic Minority Oversampling Technique), Feature Extraction, Standardization, Hyperparameter Tuning (GridSearchCV), Random Forest Model Training and Comparison, Performance Metrics Evaluation (Confusion Matrix, Accuracy, etc.)

What is a Decision Tree?

The decision tree is used in machine learning for both classification and regression models. The aim of this algorithm is to predict the target variable by splitting the data into branches. The decision tree branches the data according to certain operations on the data. Starting from the root of the tree, it divides each branch according to certain characteristics. This splitting process continues until the last leaf node. At the last leaf node, it predicts the target variable. There are some metrics used in the splitting process here. The most well-known division operations are the information gain and the gini index. At each node, the feature that provides the best split is selected. new branches are created according to the selected feature. This process continues until all data is classified in leaf nodes or until the stopping criterion is reached. The advantages of the decision tree are its easily understandable structure, fast training, flexibility and natural elimination of feature selection. Decision trees are very useful in explaining complex relationships in a simple way.

Dataset

I chose for the DecisionTree algorithm is the Red Wine quality dataset. I chose this dataset because it is simple and clean for classification modeling. The target variable we aim to find in the dataset is quality. Here I will predict the target variables with machine learning. There is class imbalance in the dataset. For this, I will oversample using the SMOTE technique. There is not much on the preprocessing side, but I will preprocess for target grouping. I will group the grouping into bad, average and good. I will also do standardization. I will do Exploratory Data Analysis to make feature selection. I will remove the class imbalance while dividing the dataset into training and test set. Then I will first train the model without adjusting the hyperparameters and see the results. Then I will change the hyperparameters and adjust the appropriate hyperparameters. By adjusting the hyperparameters, I think to avoid overfitting. Finally, I will show and compare the accuracy values. Below you can see what the variables in the dataset are and their description.

Fixed Acidity: There are acids in wine. These are fixed or non-volatile acids. Fixed-acidity is the total amount of acids consisting of organic acids such as malic and lactic. It determines the general characteristics of the wine.

• Volatile Acidity: is the amount of acetic acid in the wine. It refers to the amount of shortchain organic acids such as acetic acid in wine. If the level is high, the wine may develop a vinegar-like flavor.

• Citric Acid: It is present in small amounts in wine and adds freshness and flavor. It is a weak organic acid found naturally in citrus fruits.

• Residual Sugar: is the amount of unfermented sugar remaining in the wine after fermentation. It is used to determine the sweetness of the wine. It is measured in grams per liter.

• Chlorides: It is the amount of salt in the wine. The extraction of more chloride during winemaking is due to the removal of ions from the skins during fermentation.

• Free Sulfur Dioxide: It is reactive sulfur dioxide that exhibits germicidal and antioxidant properties in wine.

• Total Sulfur Dioxide: The total amount of free and bound sulfur dioxide in wine. It is undetectable at low levels, but becomes noticeable as the amount of free SO2 increases.

• Density: The density of wine depends on the ratio of sugar and alcohol and is usually close to the density of water. It is measured with a hydrometer and refers to the specific gravity, which is the ratio of liquid to pure water.

• pH: It defines whether the wine is acidic or basic on a scale of 0-14. Most wines have a pH between 3-4. pH values higher than 3.65 can increase the risk of microbial spoilage.

• Sulphates: An additive found naturally in wine at low levels, which can be added to protect against bacteria and yeasts. Contributes to sulfur dioxide levels.

• Alcohol: The percentage of the alcohol content of the wine.

• Quality: It is an output variable and is rated with a score between 3 and 8 based on sensory data.

Dataset Link :

https://www.kaggle.com/datasets/uciml/red-wine-quality-cortez-et-al-2009

Required libraries

The image below shows the libraries I use. These libraries are necessary for data processing, data visualization, using the machine learning model and using performance metrics.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score,␣
 ↪confusion_matrix
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import GridSearchCV
```

Data Load

I uploaded the dataset I downloaded via Kaggle to the folder where I work. Then I uploaded it to my code from local.

```
wine_data = pd.read_csv('winequality-red.csv')
```

Data Information

The image below provides information about the general structure and content of the wine dataset. the first step towards understanding the dataset has been taken. This type of analysis allows us to understand the size of the dataset, the columns it contains and the general characteristics of the data. It can provide a basic idea to analyze how variables in the dataset can affect wine quality. The information obtained at this stage is used to determine which variables are important in the modeling process, to identify potential problems in the data set and to plan steps such as data cleaning or pre-processing.

3.4 General information about the dataset

```
wine_data = pd.read_csv('winequality-red.csv')
```

```
print(wine_data.head())
```

```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
```



```
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality
0      9.4        5
1      9.8        5
2      9.8        5
3      9.8        6
4      9.4        5
```

```
print(wine_data.shape)
```

```
(1599, 12)
```

```
print(wine_data.describe())
```

```
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count    1599.000000       1599.000000  1599.000000     1599.000000
mean        8.319637          0.527821     0.270976        2.538806
std         1.741096          0.179060     0.194801        1.409928
min         4.600000          0.120000     0.000000        0.900000
25%         7.100000          0.390000     0.090000        1.900000
50%         7.900000          0.520000     0.260000        2.200000
75%         9.200000          0.640000     0.420000        2.600000
max        15.900000          1.580000     1.000000       15.500000

         chlorides  free sulfur dioxide  total sulfur dioxide      density  \
count  1599.000000          1599.000000           1599.000000  1599.000000
mean      0.087467            15.874922             46.467792     0.996747
std       0.047065            10.460157             32.895324     0.001887
min       0.012000             1.000000              6.000000     0.990070
25%       0.070000             7.000000             22.000000     0.995600
50%       0.079000            14.000000             38.000000     0.996750
75%       0.090000            21.000000             62.000000     0.997835
max       0.611000            72.000000            289.000000     1.003690

                pH    sulphates      alcohol      quality
count  1599.000000  1599.000000  1599.000000  1599.000000
mean      3.311113     0.658149    10.422983     5.636023
std       0.154386     0.169507     1.065668     0.807569
min       2.740000     0.330000     8.400000     3.000000
25%       3.210000     0.550000     9.500000     5.000000
50%       3.310000     0.620000    10.200000     6.000000
75%       3.400000     0.730000    11.100000     6.000000
max       4.010000     2.000000    14.900000     8.000000
```

The image below shows the missing values. As can be seen in this output, there is no missing value. Since there are no missing values here, there is no need for any preprocessing.

```
missing_values = wine_data.isnull().sum()
print(missing_values)
```

```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

I will train the model before preprocessing the dataset so that we can see the effect after preprocessing more clearly.

```
X = wine_data.drop(columns=['quality'])
y = wine_data['quality']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=21, stratify=y)

dt_model = DecisionTreeClassifier(random_state=21)

dt_model.fit(X_train, y_train)

y_pred = dt_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Model Accuracy:", accuracy)
print("\nClassification Report:\n", report)
```

```
Model Accuracy: 0.61875

Classification Report:
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.00      0.00      0.00        11
           5       0.69      0.69      0.69       136
           6       0.63      0.63      0.63       128
```



```
           7       0.55      0.55      0.55        40
           8       0.17      0.33      0.22         3

    accuracy                           0.62       320
   macro avg       0.34      0.37      0.35       320
weighted avg       0.62      0.62      0.62       320
```

I divided the dataset into 80 percent training data and 20 percent test data. I did not adjust any hyperparameters. When I trained the model in this way, the accuracy rate was 60 percent. The other performance measures do not look very good as can be seen.

In addition, as can be seen, the model tries to predict 6 values in the quality variable. For this, I will first perform a grouping process.

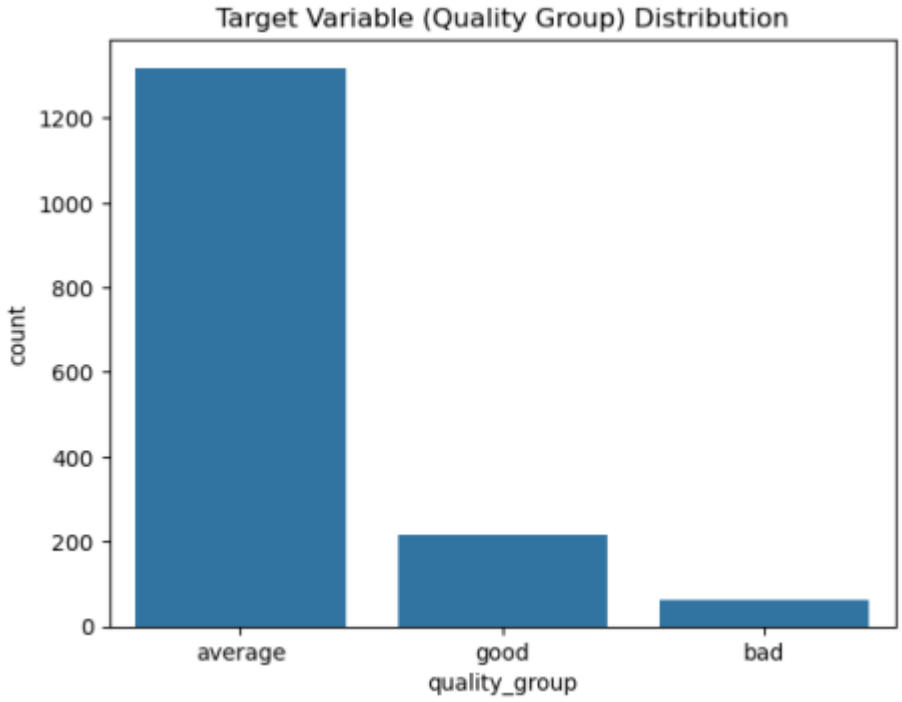
```
def quality_grouping(quality):
    if quality <= 4:
        return 'bad'
    elif quality <= 6:
        return 'average'
    else:
        return 'good'
```

```
wine_data['quality_group'] = wine_data['quality'].apply(quality_grouping)
```

In the next step, we will perform exploratory data analysis. I have avoided unnecessary visualizations here. I used visualizations for feature extraction and for a better understanding of the dataset.

First, let's see the distribution of the groupings we have made.

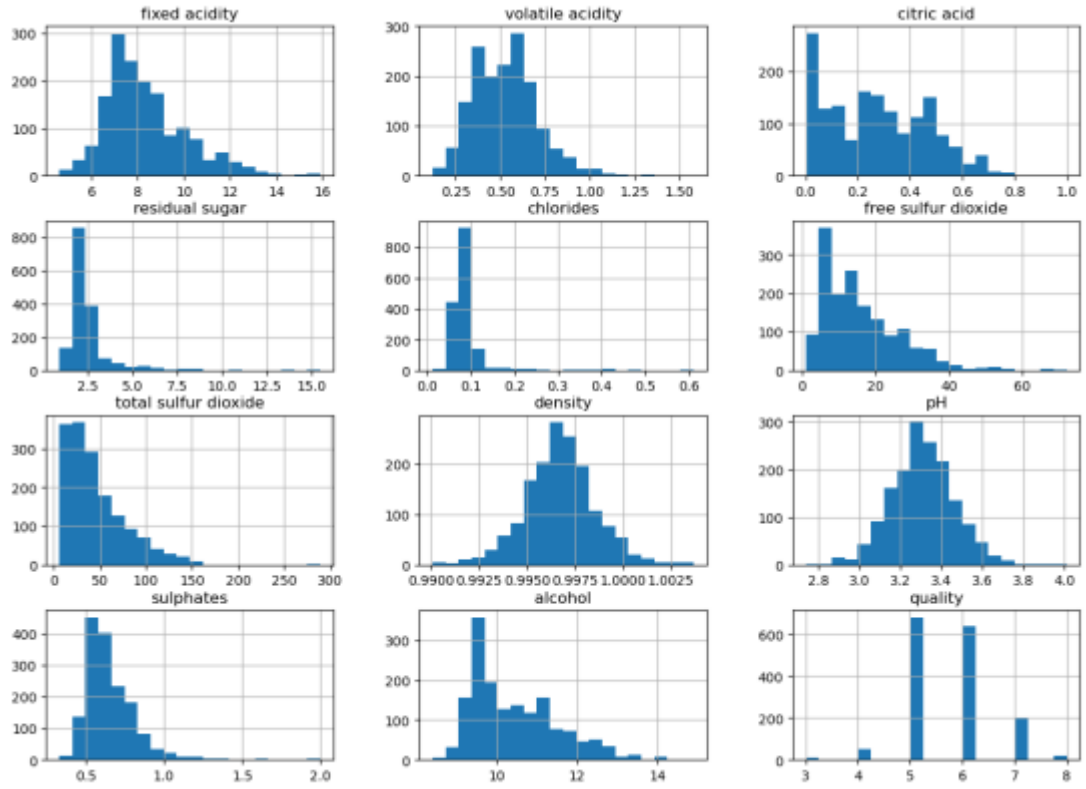
```
sns.countplot(data=wine_data, x='quality_group')
plt.title("Target Variable (Quality Group) Distribution")
plt.show()
```



As can be seen, there is an unbalanced distribution of classes in the group distribution. In the following stages, I will perform operations to overcome the class imbalance.

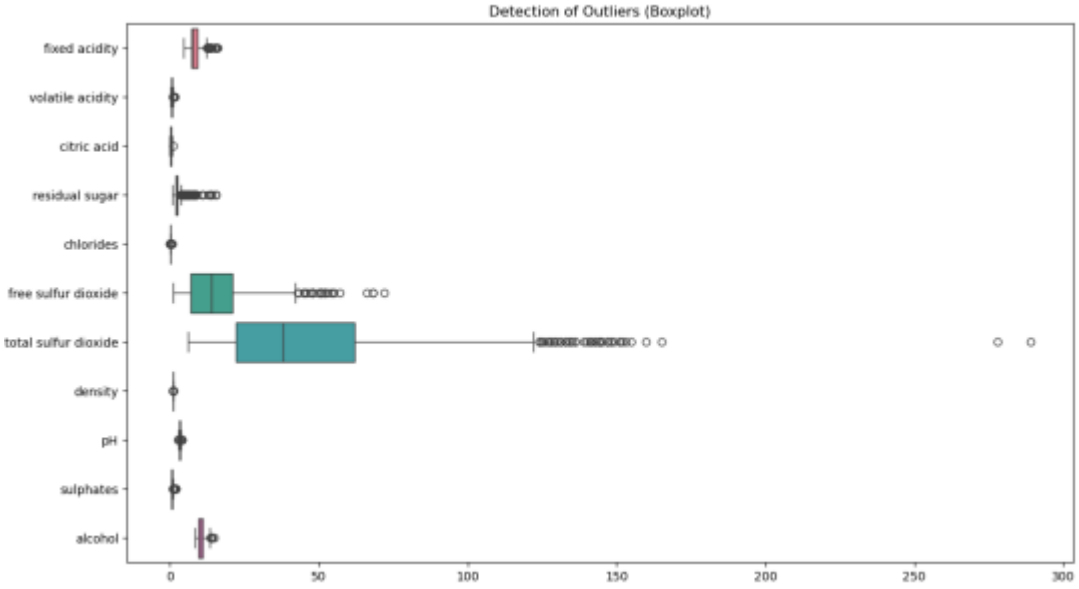
I will perform a visualization about the Distribution of Numerical Properties.

```
wine_data.hist(bins=20, figsize=(14, 10))
plt.suptitle("Distribution of Numerical Characteristics")
plt.show()
```



The table also shows the values of the numerical values. The value ranges can be understood by looking at the table. Since we cannot understand the outliers from the table, we will now draw a boxplot.

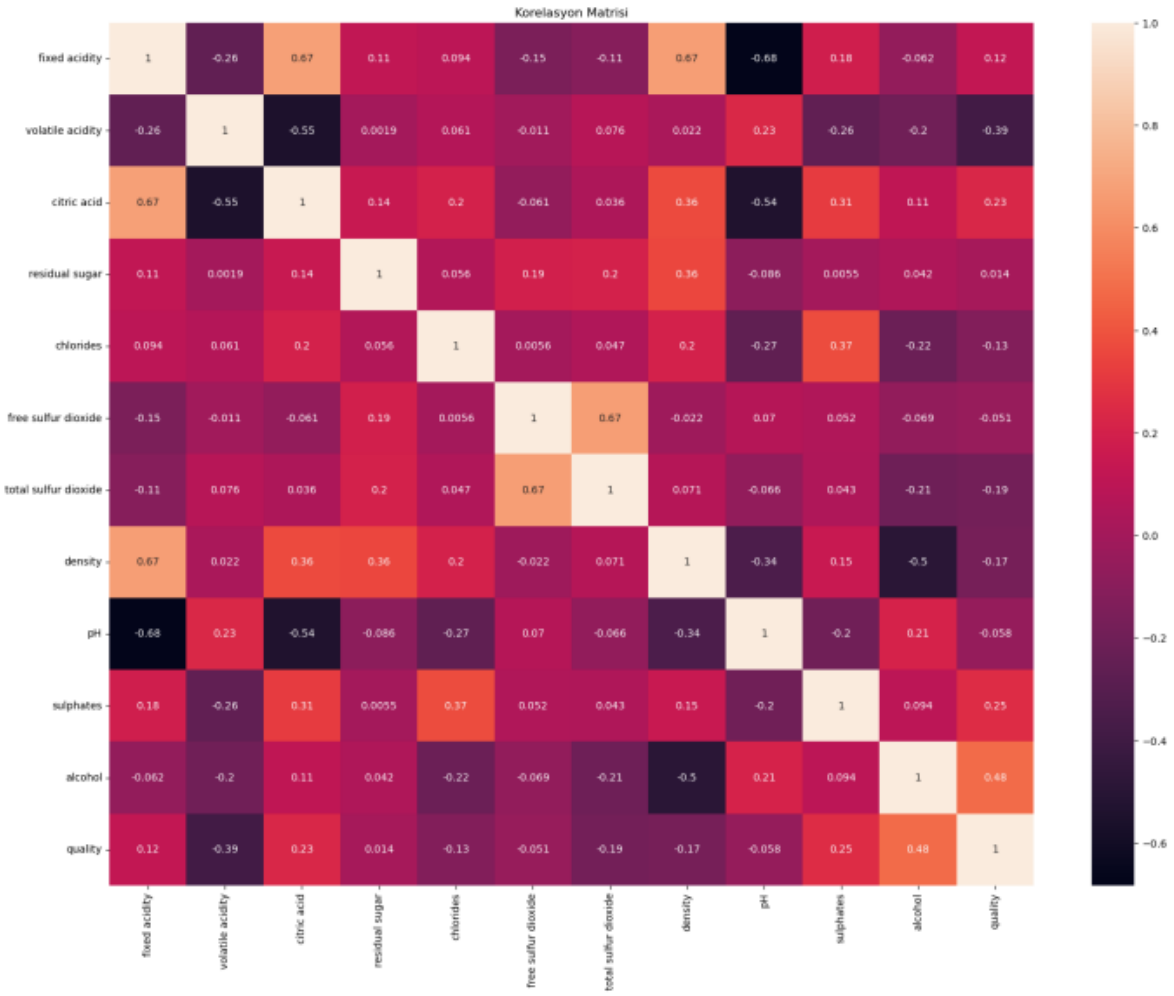
```
plt.figure(figsize=(14, 8))
sns.boxplot(data=wine_data.drop(columns=['quality', 'quality_group']),␣
 ↪orient='h')
plt.title("Detection of Outliers (Boxplot)")
plt.show()
```



As can be seen from the boxplot, outliers are observed in each variable. I will not take any action on these outliers. Due to the small size of my dataset, the data I will remove may have a negative impact on model training. In addition, the fact that I have an unbalanced class distribution is also a reason why I will not take action on outliers.

Now we will create a correlation matrix. The reason for creating this matrix is to show the relationship between variables. I want to select the features needed for feature extraction that identify weak relationships between variables.

```
# We must select numeric columns by removing the 'quality_group' column
numeric_data = wine_data.drop(columns=['quality_group'])

corr_matrix = numeric_data.corr()
plt.figure(figsize=(20,15))
sns.heatmap(corr_matrix, annot=True)
plt.title("Korelasyon Matrisi")
plt.show()
```



In the correlation matrix, the weak correlations were the ones we would pay attention to. Here, I want to identify the weak correlations with the quality variable. Our threshold for identifying weak correlations will be 0.1. We have 3 variables below the threshold. These are ‘residual sugar’, ‘free sulfur dioxide’, ‘pH’ variables that show a weak relationship with the quality variable. I will let you see them textually below. Then I will assign the variables to be extracted to a variable and then extract these variables.

```
cor_target = abs(corr_matrix["quality"])
relevant_features = cor_target[cor_target>0.1]
print("Those with strong relationships:")
print(relevant_features)
```

```
Those with strong relationships:
fixed acidity           0.124052
volatile acidity        0.390558
citric acid             0.226373
chlorides               0.128907
```



```
total sulfur dioxide    0.185100
density                 0.174919
sulphates               0.251397
alcohol                 0.476166
quality                 1.000000
Name: quality, dtype: float64
```

```
to_drop = cor_target[cor_target < 0.1].index
print("Features to be removed, threshold 0.1")
print(to_drop)
```

Now I just want to see how it improves after grouping and then we will look at the situation after feature extraction, standardization and oversampling.

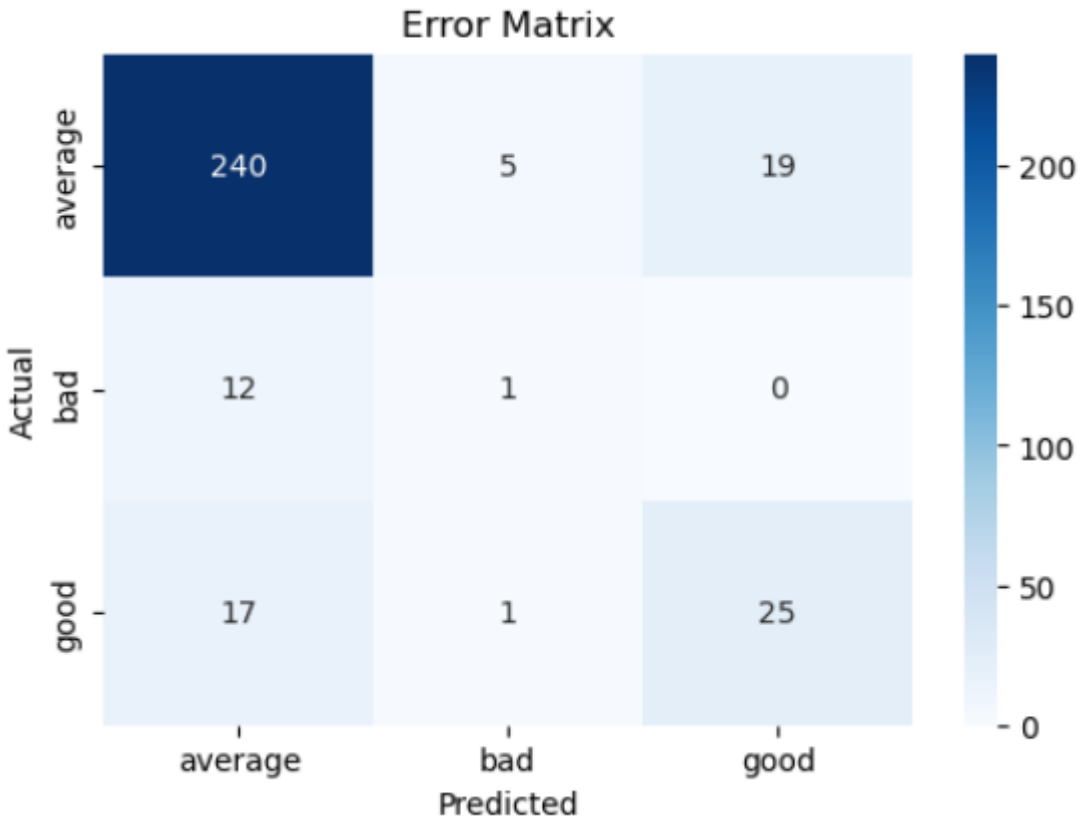
```
X = wine_data.drop(columns=['quality', 'quality_group'])
y = wine_data['quality_group']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=21, stratify=y)

dt_model = DecisionTreeClassifier(random_state=21, class_weight='balanced')

dt_model.fit(X_train, y_train)

y_pred = dt_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Model Accuracy:", accuracy)
print("\nClassification Report:\n", report)
```

```
Model Accuracy: 0.83125

Classification Report:
               precision    recall  f1-score   support

     average       0.89      0.91      0.90       264
         bad       0.14      0.08      0.10        13
        good       0.57      0.58      0.57        43

    accuracy                           0.83       320
   macro avg       0.53      0.52      0.53       320
weighted avg       0.82      0.83      0.82       320
```

As you can see, the accuracy rate has increased. But it only seems to check the average ones better. It can also predict the good value at an acceptable level. But for the bad value, the model has very bad results. You can see this in the performance metrics. In addition, I will use a confusion matrix for better understanding.

```
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',␣
↪xticklabels=['average', 'bad', 'good'], yticklabels=['average', 'bad',␣
↪'good'])
plt.title('Error Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
[[240   5  19]
 [ 12   1   0]
 [ 17   1  25]]
```



As seen in the confusion matrix, we can say that the model does not have good predictive ability for other values except for the mean values. This may be due to unbalanced class distribution, feature extraction, standardization and hyperparameterization. In the next step, we will first perform feature extraction.

```
filtered_data = wine_data.drop(columns=to_drop)

print(f"Extracted Features: {list(to_drop)}")
print(f"New Data Set Sizes: {filtered_data.shape}")
print(filtered_data.info())
```

```
Extracted Features: ['residual sugar', 'free sulfur dioxide', 'pH']
New Data Set Sizes: (1599, 10)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   chlorides             1599 non-null   float64
 4   total sulfur dioxide  1599 non-null   float64
 5   density               1599 non-null   float64
 6   sulphates             1599 non-null   float64
 7   alcohol               1599 non-null   float64
 8   quality               1599 non-null   int64  
 9   quality_group         1599 non-null   object 
dtypes: float64(8), int64(1), object(1)
memory usage: 125.1+ KB
None
```

Now we are going to divide the data set. I divided it into 80 percent training data and 20 percent test data. Before that, I divided it in different ratios, that's how I determined the optimal ratio.

```
X = filtered_data.drop(columns=['quality', 'quality_group'])
y = filtered_data['quality_group']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=21, stratify=y)
```

Now I will perform a standardization process, which will allow us to equalize the scales of the features in the dataset and perform a balanced analysis. The standardized data helps the model learn faster and more accurately.

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

I mentioned earlier that the dataset has an unbalanced class distribution. For this I will use SMOTE, an oversample technique. SMOTE adds synthetic data samples to minority classes. This reduces class imbalance. It has a positive impact on model performance. It also prevents overfitting by increasing generalization ability. But by itself it is not very effective to prevent overfitting.

```
smote = SMOTE(random_state=21)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled,␣
 ↪y_train)
print(X_train_resampled)
```

```
[[ 0.15681567 -0.86821608  1.94009371 …  0.75957296 -0.46752912
   0.54663988]
 [-0.86997907  0.34382322 -1.09074041 … -0.28600853 -0.41063381
   0.35761591]
 [-0.01431679  1.03248191 -0.62840978 …  0.03244268  0.61348184
  -0.87103989]
 …
 [ 1.52168003 -1.11334135  1.25624968 … -0.54456808  0.9277072
   0.4746752 ]
 [ 3.82539467  0.43626472  2.48429069 …  3.01948918  0.20545918
   0.65885558]
 [ 0.13894039 -0.9611377   0.69855417 … -0.70101552  0.88218966
   0.84952659]]
```

```
print("Original Trainer Size:", X_train.shape, y_train.shape)
print("SMOTE Post Trainer Size:", X_train_resampled.shape, y_train_resampled.
 ↪shape)
```

```
Original Trainer Size: (1279, 8) (1279,)
SMOTE Post Trainer Size: (3165, 8) (3165,)
```

Now we will see the results by training the model after all the operations we have done.

```
dt_model = DecisionTreeClassifier(random_state=21, class_weight='balanced')

dt_model.fit(X_train_resampled, y_train_resampled)

y_pred = dt_model.predict(X_test_scaled)
```

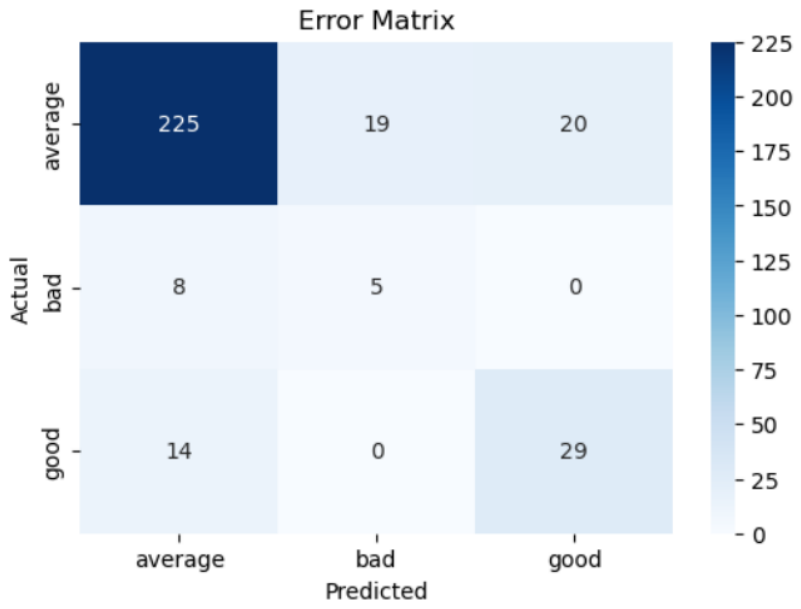


```
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Model Accuracy:", accuracy)
print("\nClassification Report:\n", report)
```

```
Model Accuracy: 0.809375

Classification Report:
               precision    recall  f1-score   support

      average       0.91      0.85      0.88       264
          bad       0.21      0.38      0.27        13
         good       0.59      0.67      0.63        43

     accuracy                           0.81       320
    macro avg       0.57      0.64      0.59       320
weighted avg       0.84      0.81      0.82       320
```

After training the model, the accuracy decreased, but this is not important. Because when we look at the other performance measures, there is an improvement for good and bad value predictions. This shows that our preprocessing, feature extraction and oversample technique for unbalanced class distribution had a positive effect. Now we will understand this better by creating the confusion matrix again.

```
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',␣
 ↪xticklabels=['average', 'bad', 'good'], yticklabels=['average', 'bad',␣
 ↪'good'])
plt.title('Error Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



As can be seen on the confusion matrix, there is a positive improvement in the model.

In the next step we will perform hyperparameter tuning. I will use the gridSearchCV search method to find the best combination of hyperparameters. With hyperparameter tuning I want to improve the accuracy, performance and generalizability of the model. Below I will explain what the hyperparameters are and what they do.

• criterion: This parameter refers to the criterion used to determine how to split the decision tree at each node. It offers two options: gini and entropy

• **max_depth:* determines the maximum depth of the tree. That is, it limits the maximum number of steps of the tree structure from the root node to the leaf node.*

• **min_samples_split:* The minimum number of samples required for a node to be split.*

• **min_samples_leaf:* The minimum number of instances that should be present in a leaf node.*

• **max_features:* It refers to the maximum number of features to be evaluated during the split at each node.

Annotation: The following warnings do not affect the gridSearchCV method. It only gives warnings due to the variable names in the columns.

```
params = {
    "criterion": ['gini', 'entropy'],
```



```
    "max_depth": np.arange(3, 20),
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [2, 4],
    "max_features": [0.5, 0.75, 1]
}

for cv in range(3, 10):
    cv_grid = GridSearchCV(estimator=dt_model, param_grid=params, cv=cv,␣
 ↪scoring='accuracy', n_jobs=-1)
    cv_grid.fit(X_train_resampled, y_train_resampled)

    print("%d fold score: %3.2f" % (cv, cv_grid.score(X_test, y_test)))
    print("Best parameters: ", cv_grid.best_params_)
```

```
C:\Users\Omer\anaconda3\Lib\site-packages\sklearn\base.py:486: UserWarning: X
has feature names, but DecisionTreeClassifier was fitted without feature names
  warnings.warn(

3 fold score: 0.76
Best parameters:  {'criterion': 'entropy', 'max_depth': 12, 'max_features': 0.5,
'min_samples_leaf': 2, 'min_samples_split': 5}

C:\Users\Omer\anaconda3\Lib\site-packages\sklearn\base.py:486: UserWarning: X
has feature names, but DecisionTreeClassifier was fitted without feature names
  warnings.warn(

4 fold score: 0.82
Best parameters:  {'criterion': 'entropy', 'max_depth': 13, 'max_features': 0.5,
'min_samples_leaf': 2, 'min_samples_split': 5}

C:\Users\Omer\anaconda3\Lib\site-packages\sklearn\base.py:486: UserWarning: X
has feature names, but DecisionTreeClassifier was fitted without feature names
  warnings.warn(

5 fold score: 0.82
Best parameters:  {'criterion': 'entropy', 'max_depth': 12, 'max_features':
0.75, 'min_samples_leaf': 2, 'min_samples_split': 5}

C:\Users\Omer\anaconda3\Lib\site-packages\sklearn\base.py:486: UserWarning: X
has feature names, but DecisionTreeClassifier was fitted without feature names
  warnings.warn(

6 fold score: 0.13
Best parameters:  {'criterion': 'entropy', 'max_depth': 12, 'max_features': 0.5,
'min_samples_leaf': 2, 'min_samples_split': 2}

C:\Users\Omer\anaconda3\Lib\site-packages\sklearn\base.py:486: UserWarning: X
has feature names, but DecisionTreeClassifier was fitted without feature names
  warnings.warn(

7 fold score: 0.82
Best parameters:  {'criterion': 'entropy', 'max_depth': 13, 'max_features': 0.5,
```

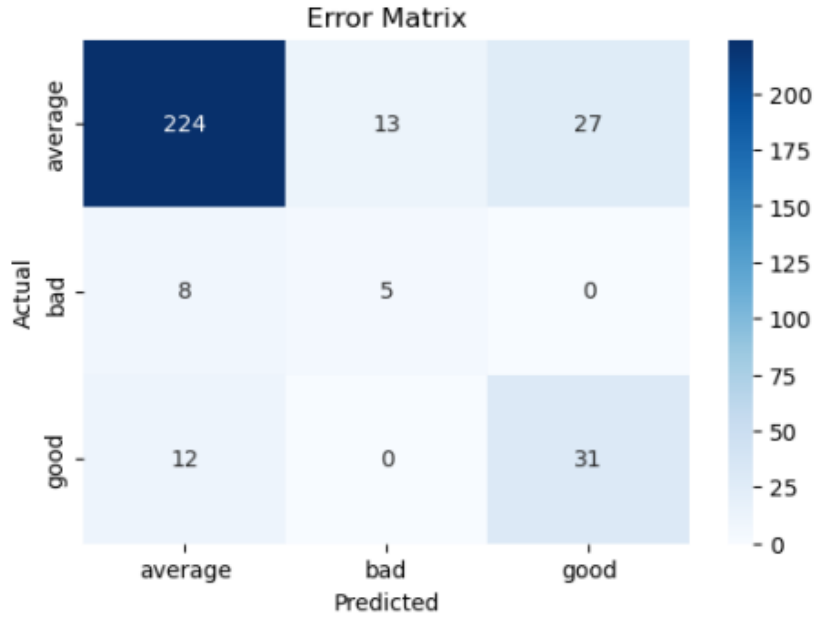
```
'min_samples_leaf': 2, 'min_samples_split': 5}

C:\Users\Omer\anaconda3\Lib\site-packages\sklearn\base.py:486: UserWarning: X
has feature names, but DecisionTreeClassifier was fitted without feature names
  warnings.warn(

8 fold score: 0.82
Best parameters:  {'criterion': 'entropy', 'max_depth': 18, 'max_features':
0.75, 'min_samples_leaf': 2, 'min_samples_split': 2}
9 fold score: 0.82
Best parameters:  {'criterion': 'entropy', 'max_depth': 19, 'max_features':
0.75, 'min_samples_leaf': 2, 'min_samples_split': 5}

C:\Users\Omer\anaconda3\Lib\site-packages\sklearn\base.py:486: UserWarning: X
has feature names, but DecisionTreeClassifier was fitted without feature names
  warnings.warn(
```

As seen above, we can see the hyperparameter combination that gives the best result. Using the above combination, we will train the model again by adjusting the hyperparameter settings below and see the results.

```python
best_model = DecisionTreeClassifier(
    criterion='gini',
    max_depth=13,
    max_features=0.5,
    min_samples_leaf=2,
    min_samples_split=2,
    class_weight="balanced",
    random_state=21
)

best_model.fit(X_train_resampled, y_train_resampled)

y_pred = best_model.predict(X_test_scaled)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.8125

Classification Report:
              precision    recall  f1-score   support

     average       0.92      0.85      0.88       264
         bad       0.28      0.38      0.32        13
        good       0.53      0.72      0.61        43

    accuracy                           0.81       320
   macro avg       0.58      0.65      0.61       320
weighted avg       0.84      0.81      0.82       320
```

The accuracy in predicting the bad value after hyperparameter tuning has moved slightly in a positive direction. We can see this by looking at the performance criteria, and we can also see it in the confusin matrix below.

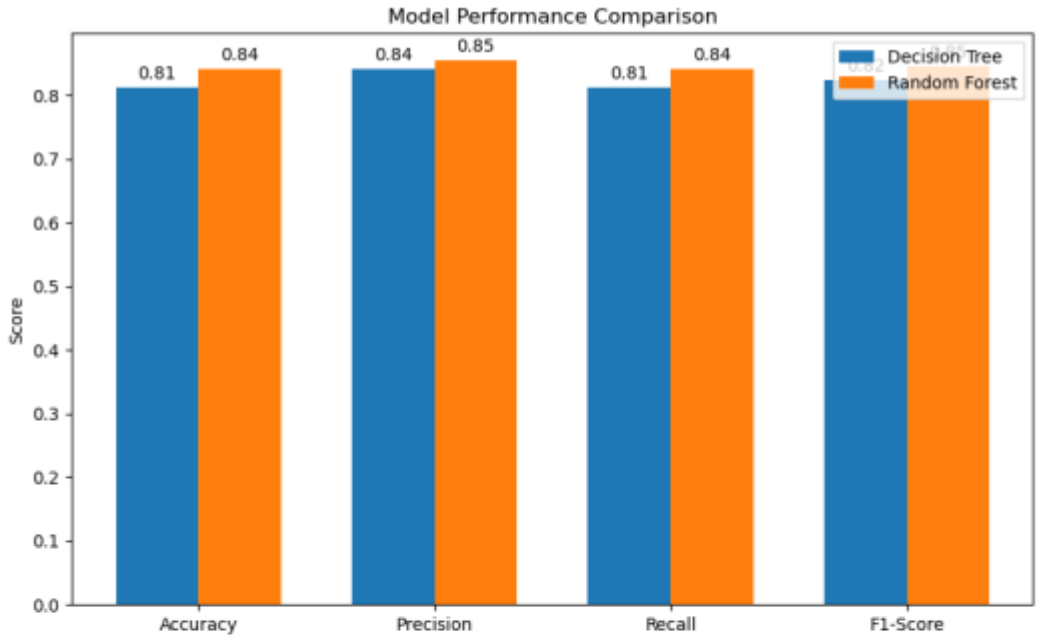


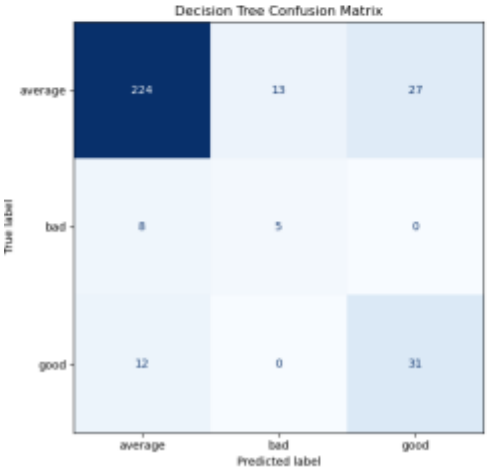
Now we will train the random forest model and compare it with the decision tree.

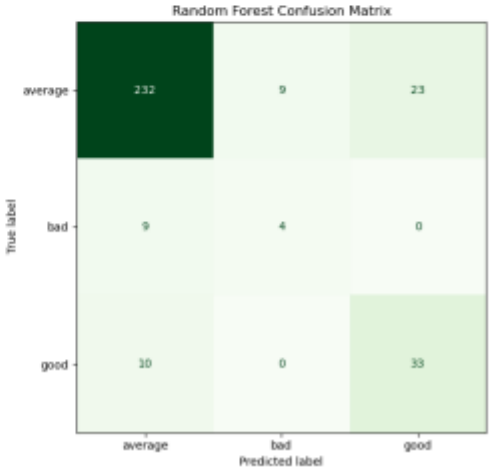
```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(
    n_estimators=100,
    max_depth=16,
    max_features=0.75,
    random_state=21
)
rf_model.fit(X_train_resampled, y_train_resampled)
y_pred_rf = rf_model.predict(X_test_scaled)
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print("\nClassification Report:\n", classification_report(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 0.840625

Classification Report:
              precision    recall  f1-score   support

     average       0.92      0.88      0.90       264
         bad       0.31      0.31      0.31        13
        good       0.59      0.77      0.67        43

    accuracy                           0.84       320
   macro avg       0.61      0.65      0.63       320
weighted avg       0.85      0.84      0.85       320
```

Below you can see a visualization of the comparison between decision tree and random forest. Afterwards we will make a comparison and a general comment.







As seen in the bar graph above, the random forest model has slightly better performance measures than the decision tree model. The optimal hyperparameter values for random forest are not selected here. When we look at the confusion matrix, it is seen that the random forest model is better in predicting the mean and good value. both models have very poor performance in predicting the bad value. The main reason for this is the unbalanced class distribution. Even

though the necessary techniques for class distribution were applied, a more balanced data would always show better performance values.

References

• https://gokerguner.medium.com/machine-learning-2-korelasyon-matrisi-%C3%B6zellikse%C3%A7imi-s%C4%B1n%C4%B1flar%C4%B1n-dengesizli%C4%9Fi-karara%C4%9Fa%C3%A7lar%C4%B1-af993bd8ea66

• https://kardelennerdem.medium.com/dengesiz-veriler-i%C3%A7in-synthetic-minorty-oversampling-tecnique-smote-y%C3%B6ntemleri-25e3c1ac3d06

• https://github.com/kserdem/SMOTE-Algorithms-for-ImblanceData/blob/main/SMOTE_Algorithms_for_Imblance_Data.ipynb

•https://katalog.marmara.edu.tr/veriler/cokluortam/cokluortam/E/C/C/B/A/63cfc0ad978d9.pdf

• https://medium.com/deep-learning-turkiye/karar-a%C4%9Fa%C3%A7lar%C4%B1-makine-%C3%B6%C4%9Frenmesi-serisi-3-a03f3ff00ba5

• https://yigitsener.medium.com/karar-a%C4%9Fa%C3%A7lar%C4%B1-decision-treekavramlar%C4%B1-ve-t%C3%BCrleri-93156d28992a

QUESTION 2

Name : Ömer Polat

No : 201401057

Subject : Segmentation with K-Means

Dataset : insurance

Methods : K-Means Clustering, Exploratory Data Analysis (EDA), Correlation Matrix Analysis, Feature Extraction, Missing Value Handling, Textual Data Processing, Data Standardization, Optimal K Determination (Elbow Method and Silhouette Score), Cluster Training with K-Means, Cluster Comparison

What is K-Means Clustering Algorithm?

K-means clustering algorithm is one of the unsupervised learning methods. That is, it does not try to predict a specific target variable. It tries to divide it into a certain number of k clusters. It groups each cluster according to the similarities of the data. This grouping is centered around a center point. The algorithm initially randomly determines k center points. The data are grouped according to their distance from the centroids. this process continues until it is optimized. K-means usually calculates the distance using the Euclidean distance. The algorithm is often used to understand relationships between groups of data in areas such as customer segmentation and image processing.

Dataset

This dataset is a detailed dataset that provides information about footballers in the fifa 23 game. It holds information on players' general characteristics, physical attributes, position and contract information. This data is suitable for comparing players with each other and grouping them according to their characteristics. It is suitable for K-means because players can be segmented according to certain categories.

There are 29 columns in this dataset. There are 17660 records in total. There are different data types in the dataset. A preliminary look at the dataset shows that it requires a lot of preprocessing.

Columns :

ID : unique identification number assigned to the player

Name : player's name

Age : player's age

Photo : URL link for the player's profile photo

Nationality : country of the player

Flag : the flag url of the player's country

Overall : player's overall skill score

Potential : the player's maximum ability score

Club : player's club

Club logo : the logo url of the player's club

Value: Market value of the player

Wage: Player's weekly salary

Special: Player's total special ability points

Preferred Foot: Player's dominant foot (right or left)

International Reputation: The player's international recognition rating (1-5)

Weak Foot: Ability level of the player's weak foot (1-5)

Skill Moves: The player's skill moves (value between 1-5)

Work Rate: Ratio of the player's offensive and defensive work rate

Body Type: Player's body type (e.g. “Normal,” “Lean,” “Stocky”)

Real Face: Whether the player has real face modeling in the game

Position: Player's actual playing position

Joined: Date the player joined the club

Loaned From: Club of origin for loaned players

Contract Valid Until: Player's contract expiration date

Height: Player's height

Weight: Player's weight

Release Clause: Player's release clause

Kit Number: Player's shirt number

Best Overall Rating: The highest overall skill rating achieved by the player in his career

Dataset Link :

https://www.kaggle.com/datasets/bryanb/fifa-player-stats-database/data

Required libraries

The image below shows the libraries I use. These libraries are necessary for data processing, data visualization, using the machine learning model and using performance metrics.

```
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 re
import seaborn as sns
```

Data Load

I uploaded the dataset I downloaded via Kaggle to the folder where I work. Then I uploaded it to my code from local.

```
file_path = 'FIFA23_official_data.csv'
fifa_data = pd.read_csv(file_path)
```

In the images below we will only take a preliminary look at the dataset.

```
print(fifa_data.head())
```

```
   ID            Name  Age  \
0  209658     L. Goretzka   27
1  212198  Bruno Fernandes   27
2  224334        M. Acuña   30
3  192985     K. De Bruyne   31
4  224232      N. Barella   25

                                  Photo Nationality  \
0  https://cdn.sofifa.net/players/209/658/23_60.png     Germany
1  https://cdn.sofifa.net/players/212/198/23_60.png    Portugal
2  https://cdn.sofifa.net/players/224/334/23_60.png   Argentina
3  https://cdn.sofifa.net/players/192/985/23_60.png     Belgium
4  https://cdn.sofifa.net/players/224/232/23_60.png       Italy

                           Flag  Overall  Potential             Club  \
0  https://cdn.sofifa.net/flags/de.png       87         88  FC Bayern München
1  https://cdn.sofifa.net/flags/pt.png       86         87  Manchester United
2  https://cdn.sofifa.net/flags/ar.png       85         85        Sevilla FC
3  https://cdn.sofifa.net/flags/be.png       91         91    Manchester City
4  https://cdn.sofifa.net/flags/it.png       86         89              Inter

                                Club Logo  … Real Face  \
0  https://cdn.sofifa.net/teams/21/30.png  …       Yes
1  https://cdn.sofifa.net/teams/11/30.png  …       Yes
2  https://cdn.sofifa.net/teams/481/30.png  …        No
3  https://cdn.sofifa.net/teams/10/30.png  …       Yes
4  https://cdn.sofifa.net/teams/44/30.png  …       Yes

                                  Position        Joined Loaned From  \
0  <span class="pos pos28">SUB   Jul 1, 2018          NaN
1  <span class="pos pos15">LCM  Jan 30, 2020          NaN
2    <span class="pos pos7">LB  Sep 14, 2020          NaN
3  <span class="pos pos13">RCM  Aug 30, 2015          NaN
4  <span class="pos pos13">RCM   Sep 1, 2020          NaN

   Contract Valid Until  Height  Weight Release Clause  Kit Number  \
0                  2026   189cm    82kg          €157M         8.0
1                  2026   179cm    69kg          €155M         8.0
2                  2024   172cm    69kg          €97.7M        19.0
3                  2025   181cm    70kg         €198.9M        17.0
4                  2026   172cm    68kg         €154.4M        23.0

   Best Overall Rating
0                  NaN
1                  NaN
2                  NaN
3                  NaN
4                  NaN

[5 rows x 29 columns]
```

The code in the image below is the code I used to perform Exploratory Data Analysis (EDA) on the dataset. The first section tries to summarize the basic information of the dataset such as column names, dimensions, number of missing values and data types of the columns in a dictionary format. In the second part, numeric columns and categorical columns in the dataset are automatically identified and classified into separate lists. In the last section, basic statistical summaries of the numeric columns are summarized. These codes were used to understand the overall structure of the dataset and to quickly obtain basic information to prepare for data analysis or modeling processes.

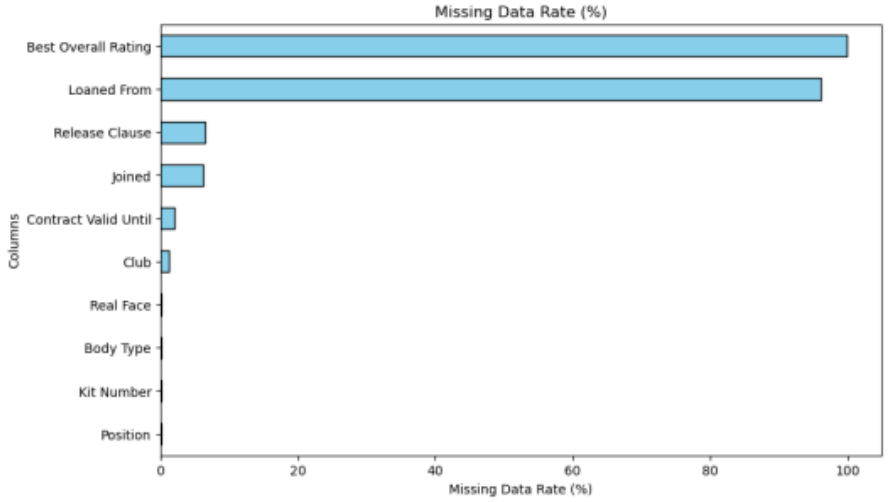
```
eda_summary = {
    "Columns": fifa_data.columns.tolist(),
    "Shape": fifa_data.shape,
    "Missing Values": fifa_data.isnull().sum().to_dict(),
    "Data Types": fifa_data.dtypes.to_dict()
}
```

```
numeric_columns = fifa_data.select_dtypes(include=["float64", "int64"]).columns.
↪tolist()
categorical_columns = fifa_data.select_dtypes(include=["object"]).columns.
↪tolist()
```

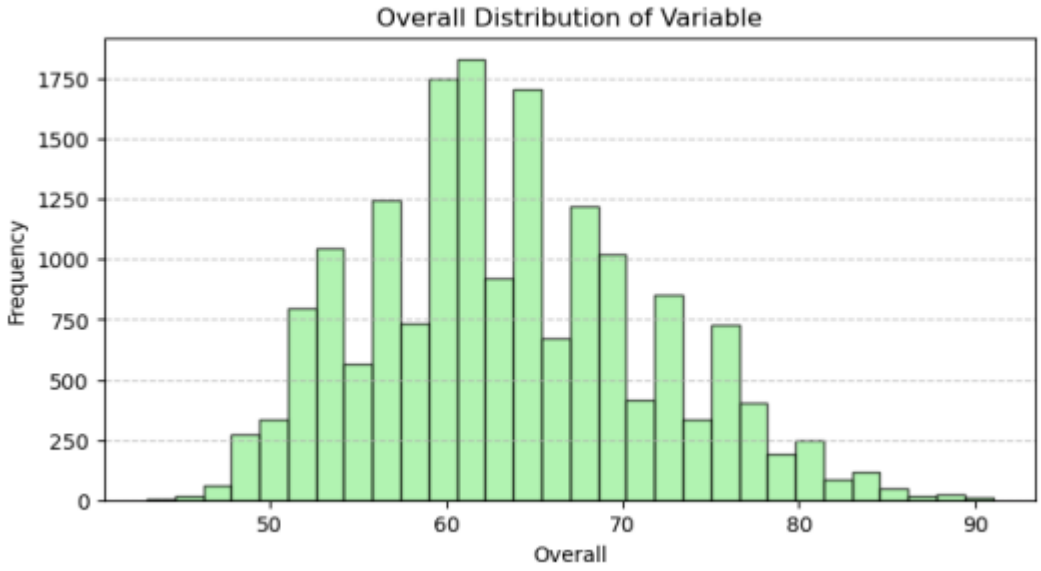
```
eda_summary["Numeric Columns"] = numeric_columns
eda_summary["Categorical Columns"] = categorical_columns
```

```
numeric_summary = fifa_data[numeric_columns].describe()
```

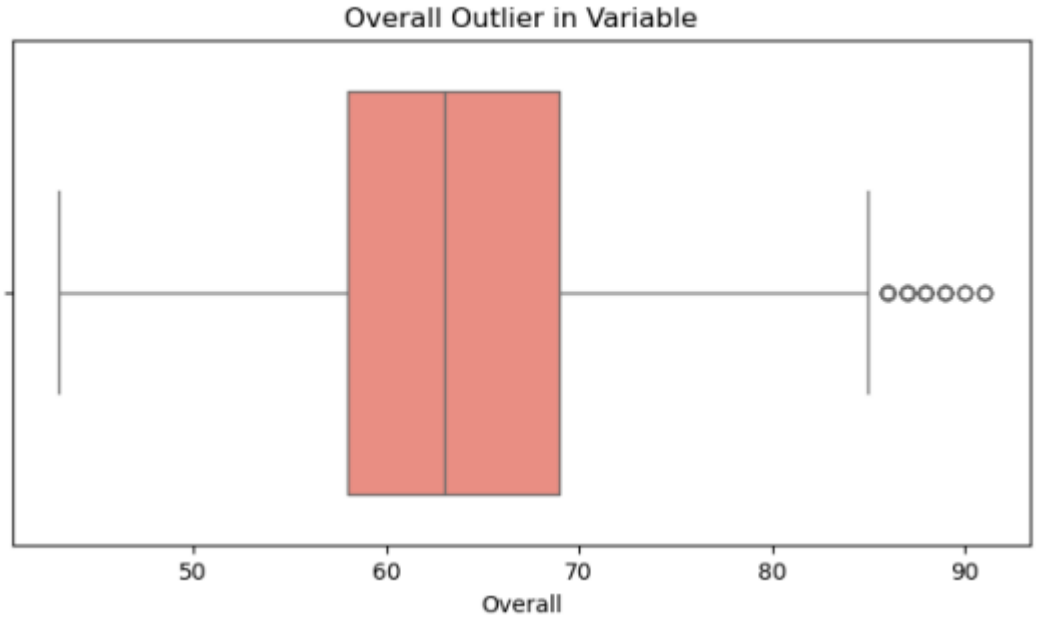
In the image below, the rates of missing values in the data set are visualized. There are quite a lot of missing values in the variables “best overall rating” and “loaned from”. We need to perform preprocessing related to this data. As we can understand from this output, the most logical move is to delete both variables. It will have a positive impact on the model training process.



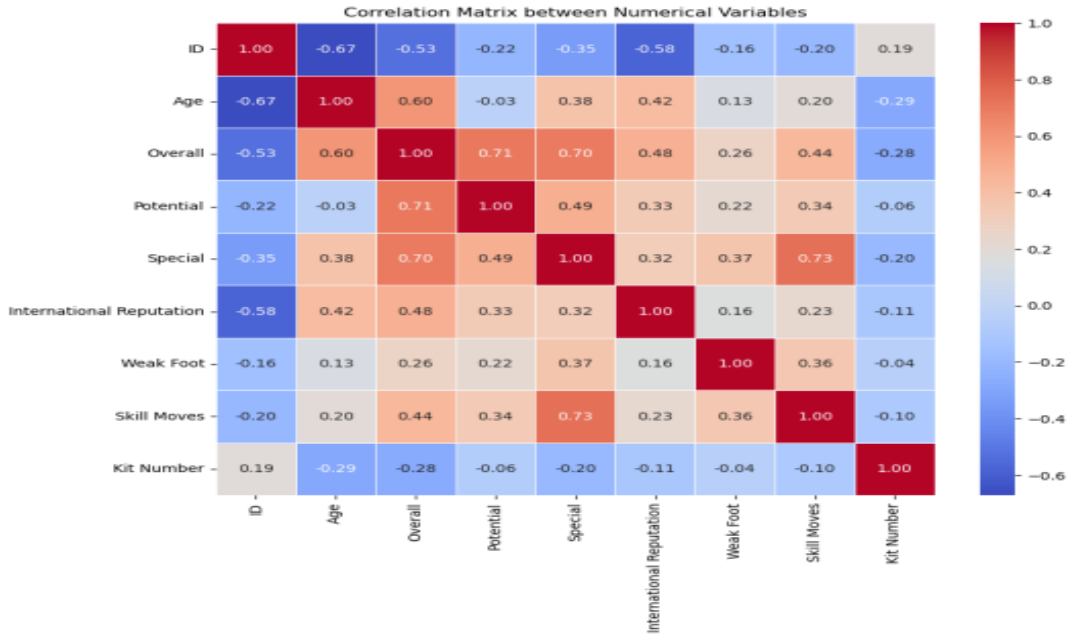
In the image below, only the distribution of one variable is shown. the distribution of each variable will not be shown separately. I will explain the distribution in a single image in order to understand its logic. the output below looks like a normal distribution. Values appear in the range of 60-70. Considering that the average value will be around 60, it can be interpreted that there may be a few outliers in the overall variable in the data set, albeit very rarely. It is also possible to say that there is no class imbalance.



In another image, we can check the accuracy of our interpretation above. As we did above, we will not make a separate representation for each variable. In the image below, it is seen that the overall variable has outlier values. Here, the average values are again collected between 60-70. Outliers are around 90. We will not take action on these outliers because I do not want the meaning of the dataset to be lost.



In the image below, the correlation matrix is visualized to show the relationship between the variables in the data set. The values in this matrix measure the relationship between variables. They take values between -1 and 1. Positive values indicate a positive relationship between variables, while negative values allow us to understand that there is a negative relationship. If the value is close to 0, it means that there is no relationship between them. As can be seen in the figure, there is a strong positive relationship between overall and potential variables. There is also a strong relationship between special and skill moves. There are also positive correlations between the other values, but there are also some with no significant correlation. This correlation matrix is a critical step in deciding which attributes we should choose in the next step.



In the next step, we will start the preprocessing steps. first we will start feature extraction. Here, many features are extracted both by utilizing the correlation matrix and by instinctive decisions. These extracted features can both negatively affect the training process of the model and increase the cost of the model.

```
columns_to_drop = [
    "Photo", "Flag", "Club Logo", "ID", "Name", "Best Overall Rating",
    "Loaned From", "Joined", "Real Face", "Kit Number", "Nationality", "Club",
    "Contract Valid Until", "Height", "Weight", "Weak Foot", "Position",␣
 ↪"Special", "Work Rate", "Body Type", "Preferred Foot",
    "Release Clause"
]
fifa_data_cleaned = fifa_data.drop(columns=columns_to_drop, axis=1)
```

The bottom image shows the process of filling in the missing data. First, categorical and numerical values are determined. While categorical variables are filled with mode, numerical

values are filled with mean values. With this method, missing data is filled and the missing data effect on the model is reduced.

```
categorical_columns_cleaned = fifa_data_cleaned.
 ↪select_dtypes(include=["object"]).columns.tolist()
for col in categorical_columns_cleaned:
    fifa_data_cleaned[col].fillna(fifa_data_cleaned[col].mode()[0],␣
 ↪inplace=True)
```

```
numeric_columns_cleaned = fifa_data_cleaned.select_dtypes(include=["float64",␣
 ↪"int64"]).columns.tolist()
for col in numeric_columns_cleaned:
    fifa_data_cleaned[col].fillna(fifa_data_cleaned[col].mean(), inplace=True)
```

In the image below, it is aimed to extract the textual expressions in the columns. Here, firstly, the textual expressions in the Value and Wage columns have been removed. These two variables are the only ones that have textual expressions left in the final data set. In this way, we have ended the risk of making errors during model training and visualizations.

```
def convert_currency_safe(value):
    if isinstance(value, str):
        value = value.replace("€", "").replace("K", "e3").replace("M", "e6")
        try:
            return float(eval(value))
        except:
            return None
    return value
```

```
fifa_data_cleaned['Value'] = fifa_data_cleaned['Value'].
 ↪apply(convert_currency_safe)
fifa_data_cleaned['Wage'] = fifa_data_cleaned['Wage'].
 ↪apply(convert_currency_safe)
```

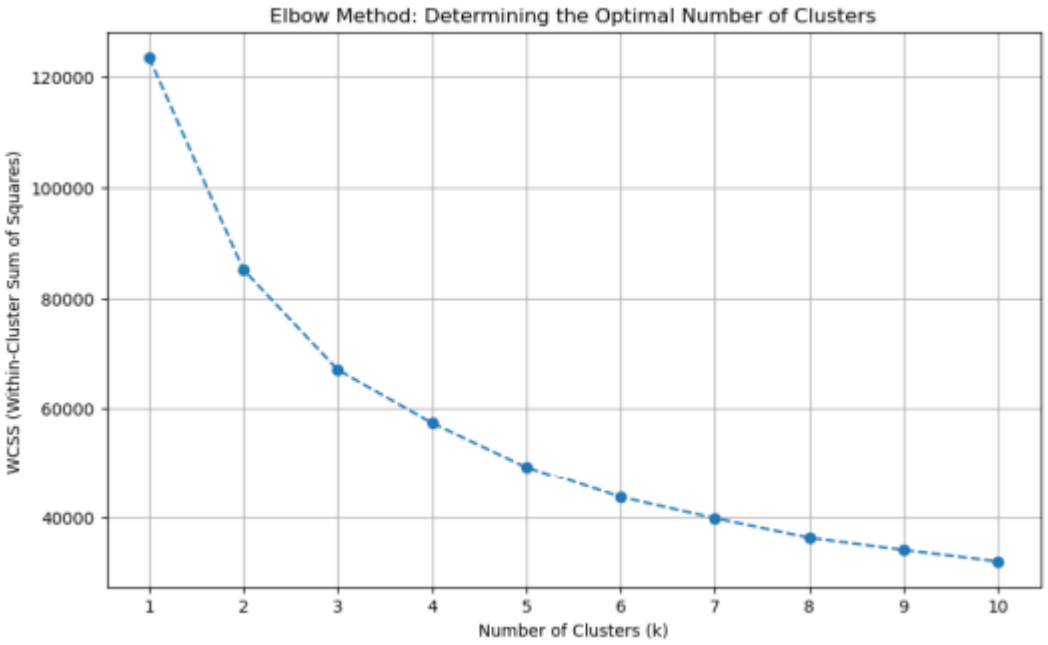
In the code below, if there are missing values in the value and wage columns, they are intended to be filled in again.

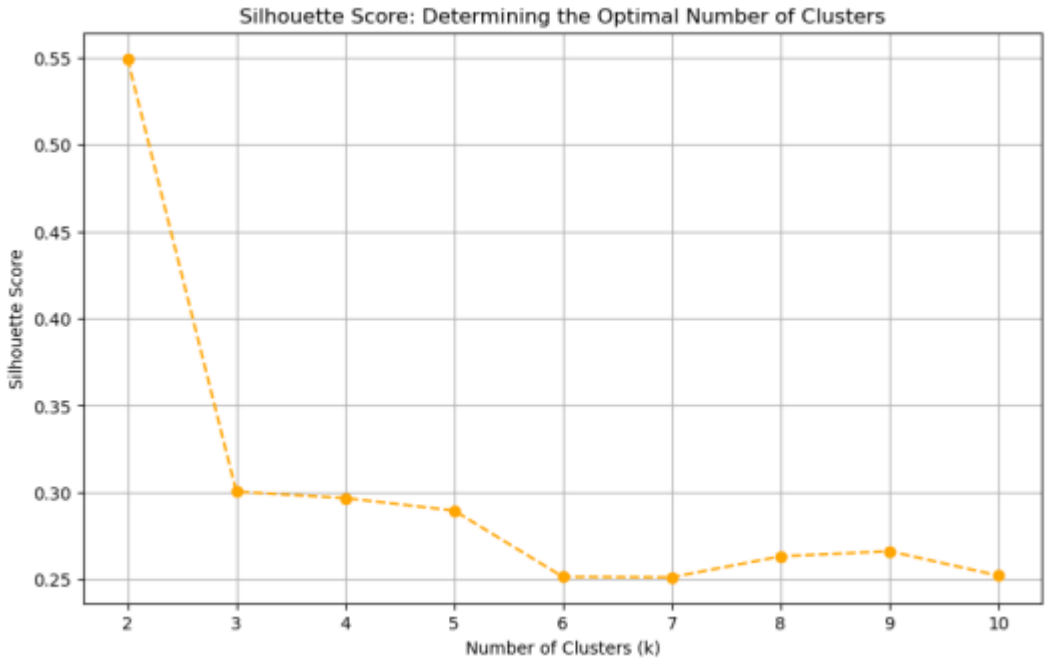
```
fifa_data_cleaned['Value'].fillna(fifa_data_cleaned['Value'].mean(),␣
 ↪inplace=True)
fifa_data_cleaned['Wage'].fillna(fifa_data_cleaned['Wage'].mean(), inplace=True)
```

The lower code ensures that numeric columns with different scales are of the same standard. This kind of standardization is necessary in modeling or analysis phases, especially in distance-based algorithms. This optimizes the impact of different sized variables on the model.

```
scaler = StandardScaler()
scaled_data = scaler.fit_transform(fifa_data_cleaned.
 ↪select_dtypes(include=["float64", "int64"]))

scaled_columns = fifa_data_cleaned.select_dtypes(include=["float64", "int64"]).
 ↪columns.tolist()
fifa_data_cleaned[scaled_columns] = scaled_data
```

In this step, it is necessary to determine the optimal number k before using the K-means algorithm. We will use two methods for this: the elbow method and the silhouette score. first we will see the output of the elbow method. in this output, the place where the sudden drop and break occurs is 2 and 3. To verify this process, we will take help from silhouette score. in the output of silhouette score, the number k is determined as 2, which yields a value of 0.55. Although the silhouette score is not very good here, it gave a good result. After interpreting the output of the two methods, the optimal number of k was determined as 2.

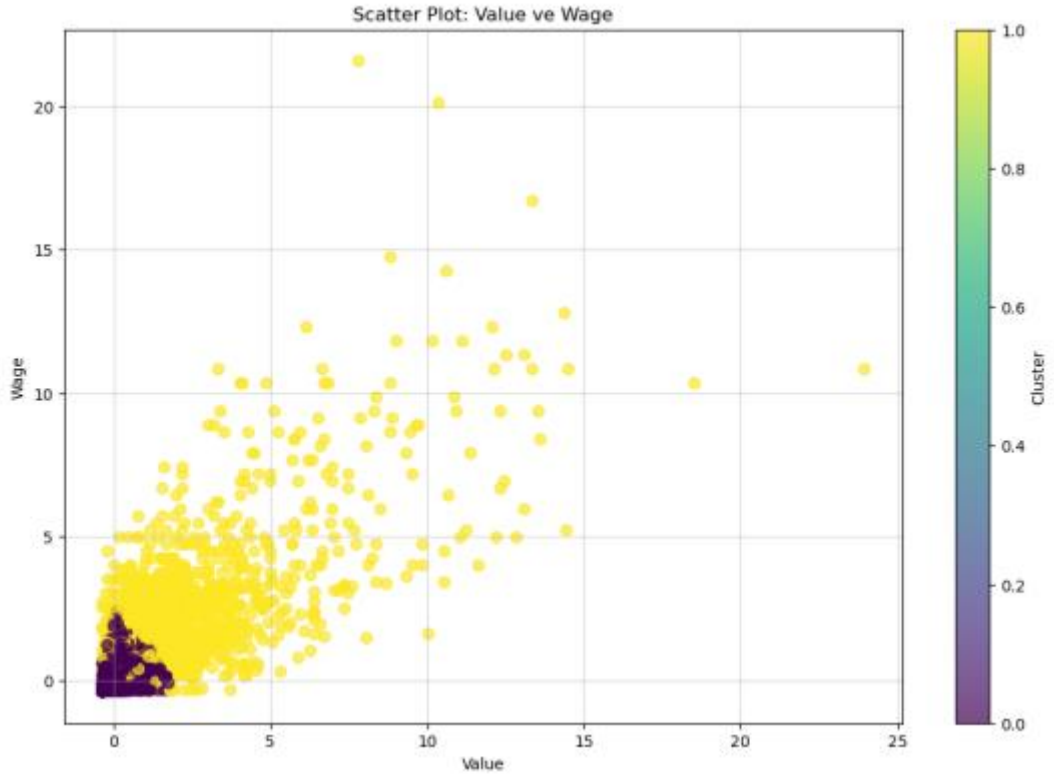




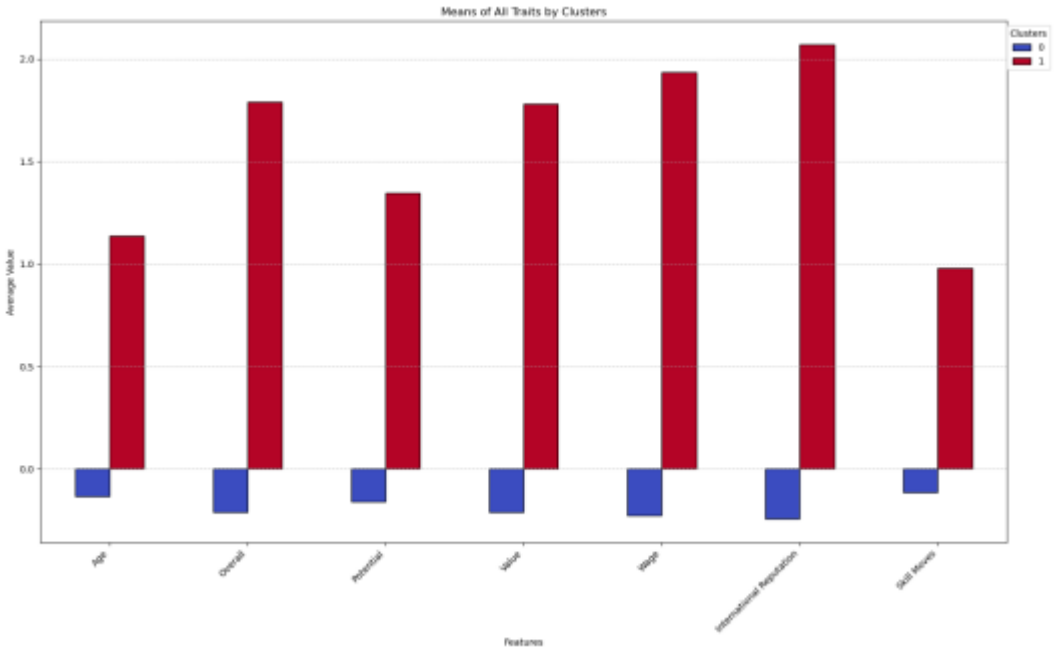
The lower code trains the K-Means algorithm with the optimal number of clusters determined by the Elbow Method and silhouette score. The model divides the dataset into 2 clusters and adds the cluster to which each data point belongs as a new column called Cluster. This process allows the data with similar characteristics in the dataset to be grouped and segmented.

```
optimal_k = 2
kmeans_model = KMeans(n_clusters=optimal_k, random_state=42, n_init=10,␣
 ↪max_iter=300)
kmeans_model.fit(fifa_data_final)

fifa_data_final['Cluster'] = kmeans_model.labels_
```

The bottom image shows the scatter plot of the clusters. The variables used are value and wage. The data is colored according to the cluster label. Players with low value and wage appear in one cluster and players with high wage and high value appear in the other cluster. This graph shows that players are segmented according to their financial characteristics.



In another graph, the average values of all the features in the clusters are compared separately. The red color represents the cluster with higher averages and the blue color represents the cluster with lower averages. As can be clearly seen in this graph, the blue cluster has lower averages and all characteristics are lower. This graph clearly shows that players are segmented according to different skill and salary levels.



Conclusion

In this assignment, I used the FIFA 23 dataset and applied the K-Means algorithm to segment players according to their characteristics. First, I had a detailed knowledge about the dataset. I preprocessed the missing values and textual expressions in the dataset and standardized the numerical data. I determined the optimal number of clusters according to the Elbow Method and Silhouette Score. I decided to create 2 clusters according to the output of these two methods. after training the model, I interpreted the outputs. we saw that the clusters were grouped according to the players' skills and salaries. As a result of the homework, I learned the logic of the clustering algorithm and how to prepare the model by going through a path and what to pay more attention to in clustering processes.

QUESTION 3

Name : Ömer Polat

No : 201401057

Subject : Training and Comparison of Linear Regression and Random Forest Models

Dataset : insurance

Methods : Linear Regression, Label Encoding, Exploratory Data Analysis (EDA), Outlier Detection, Feature Engineering, Data Scaling, Data Splitting, Random Forest Regressor, Hyperparameter Tuning (GridSearchCV), Performance Metrics Evaluation (MSE, R^2 Score), Model Comparison

What is linear regression?

Linear regression is used in statistics and machine learning. It is a prediction model. The main goal is to predict an output. It is suitable for datasets whose output is a continuous variable. The working principle is quite simple. Based on the given dataset, it creates a linear function of the target variable using independent variables. The model will perform well if there is a linear relationship between the data. In other words, its predictions will be reliable. The opposite is true for non-linear relationships.

Dataset

The data set I use is about insurance premiums of individuals. Since the target variable is a continuous variable, it is very suitable for use in a linear regression model. The data set consists of 1338 rows in total. There are 7 columns. There are no missing values in the dataset. I will explain the properties of the variables below. The dataset contains both numeric and categorical values. Our target variable here will be “charges”. We will try to predict this target variable using the model.

Independent variables;

age : The age of the person, consisting of an integer number.

sex : The gender of the person, specified as “male” or “female” in the dataset.

bmi : body mass index calculated according to the person's height and weight. it is a continuous variable. It consists of decimal numbers.

children : the number of children the person has. consists of an integer number.

smoker : indicates whether the person smokes or not. This is also categorized here. It is indicated as “yes” or “no”.

region : the region where the person lives. There are four different regions. “northeast“,”'northwest”, ‘southeast’, and ‘southwest’.

Dependent variable;

charges : the total amount the person paid for insurance. It is a continuous variable. It is our target variable. It is represented by decimal numbers.

Dataset Link :

https://www.kaggle.com/datasets/willianoliveiragibin/healthcare-insurance

Required libraries

The image below shows the libraries I use. These libraries are necessary for data processing, data visualization, using the machine learning model and using performance metrics.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.metrics import mean_squared_error, r2_score
```

Data Load

I uploaded the dataset I downloaded via Kaggle to the folder where I work. Then I uploaded it to my code from local.

```
data = pd.read_csv('insurance.csv')
```

Label Encoders

Our data set included categorical variables. These were textual data. When training the model, we need to use numeric expressions instead of textual expressions. For this we need to convert textual expressions into numeric expressions. First, we need to do this before the exploratory data analysis. Here we used label encoders to convert them to numeric values. We recognized categorical variables and converted them to numeric expressions. For example, it updated the gender value to male -> 1 female -> 0. We did the same for the other two categorical variables.

```
label_encoders = {}
categorical_columns = ['sex', 'smoker', 'region']
for col in categorical_columns:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le
```

Exploratory Data Analysis (EDA)

Exploratory data analysis is a data analysis method used to understand and summarize a data set and discover relationships. The main goal is to visualize the main features of the data set, make statistical summaries and understand the structure of the data in general or in detail. Below I will share the analysis we did step by step. In this analysis process, we will perform operations such as understanding the data set, finding missing values, seeing data types, seeing the relationship between variables. As a result of these analyzes, preprocessing, adding or removing features will be very helpful in understanding relationships.

First we will see the first five values in the dataset. these values will be sufficient for a general overview. We can get answers to our questions such as data types, the values they take, how many columns there are, etc.

```
: print("\n### First 5 Rows of Data ###")
  print(data.head())
```

```
### First 5 Rows of Data ###
   age  sex     bmi  children  smoker  region      charges
0   19    0  27.900         0       1       3  16884.92400
1   18    1  33.770         1       0       2   1725.55230
2   28    1  33.000         3       0       2   4449.46200
3   33    1  22.705         0       0       1  21984.47061
4   32    1  28.880         0       0       1   3866.85520
```

In the image below, we have an overview of the data set. In addition to the image above, how many values are in the dataset, what are the data types, etc.

```
print("\n### Data Information ###")
print(data.info())
```

```
### Data Information ###
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1338 non-null   int64
 1   sex       1338 non-null   int32
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64
 4   smoker    1338 non-null   int32
 5   region    1338 non-null   int32
 6   charges   1338 non-null   float64
dtypes: float64(2), int32(3), int64(2)
memory usage: 57.6 KB
None
```

In the image below, we can view the basic statistical properties of the dataset. this output provides statistical measurements of each numeric variable in the dataset. By interpreting such statistical properties we can understand the distribution of the data set. The summary here is an important output to understand the overall structure of the dataset and to understand potential problems for the values.

```
print("\n### Statistical Summary ###")
print(data.describe())
```

```
### Statistical Summary ###
                age          sex          bmi     children       smoker  \
count  1338.000000  1338.000000  1338.000000  1338.000000  1338.000000
mean     39.207025     0.505232    30.663397     1.094918     0.204783
std      14.049960     0.500160     6.098187     1.205493     0.403694
min      18.000000     0.000000    15.960000     0.000000     0.000000
25%      27.000000     0.000000    26.296250     0.000000     0.000000
50%      39.000000     1.000000    30.400000     1.000000     0.000000
75%      51.000000     1.000000    34.693750     2.000000     0.000000
max      64.000000     1.000000    53.130000     5.000000     1.000000

            region       charges
count  1338.000000   1338.000000
mean      1.515695  13270.422265
std       1.104885  12110.011237
min       0.000000   1121.873900
25%       1.000000   4740.287150
50%       2.000000   9382.033000
75%       2.000000  16639.912515
max       3.000000  63770.428010
```

In the image below, we check whether there is missing data in the data set. It is important to check for missing values because if there is a missing value in the data set, it may affect the model result. There are no missing values in the dataset we use. Therefore, we can see from the output below that we do not need to do any preprocessing about the missing value.

```
print("\n### Missing Value Analysis ###")
missing_values = data.isnull().sum()
print(missing_values)
```

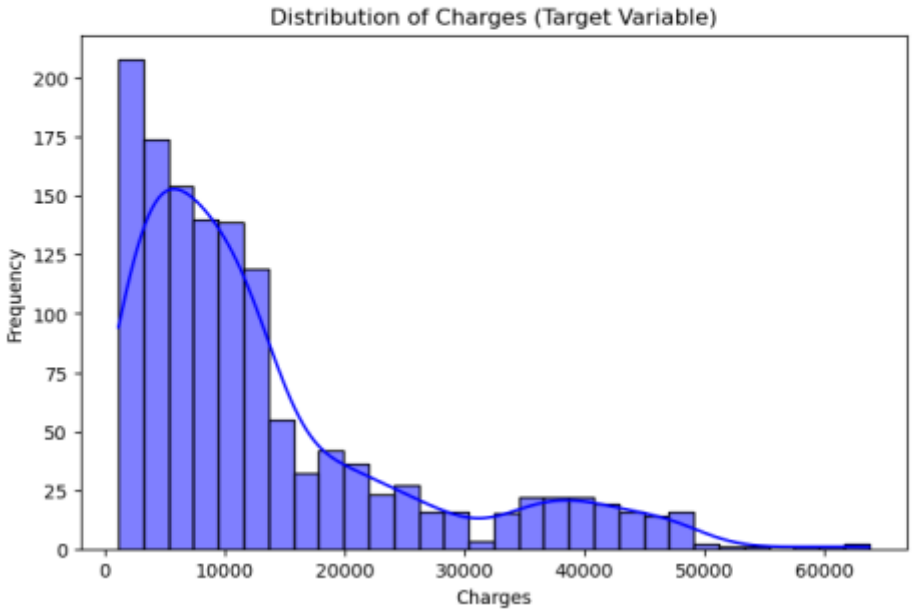
```
### Missing Value Analysis ###
age         0
sex         0
```



```
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

The image below shows the distribution of the independent variable “charges” in the data set. This image gives us a general information about insurance costs. The majority of insurance costs appear to be at low levels, but there are also high values for some individuals. This analysis is important for understanding the distribution of the target variable and for preprocessing steps in the modeling process. As can be seen from this visualization, outliers are likely to occur. For outliers, we will make a visualization again and decide whether to perform the preprocessing step or not.

```
plt.figure(figsize=(8, 5))
sns.histplot(data['charges'], kde=True, color="blue")
plt.title("Distribution of Charges (Target Variable)")
plt.xlabel("Charges")
plt.ylabel("Frequency")
plt.show()
```

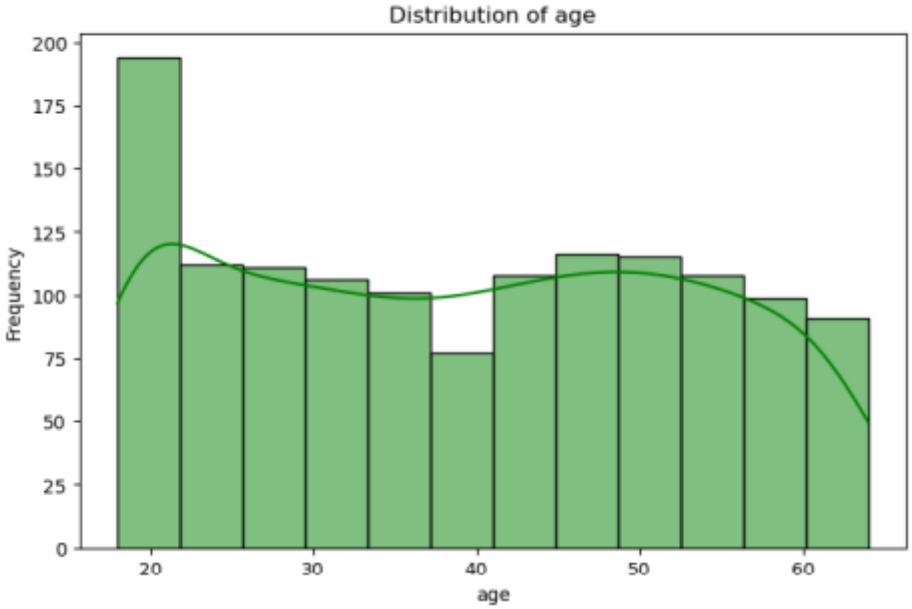


The following figure shows the distributions and frequencies of the numerical variables in the data set. This analysis is important to understand the general structure, distribution and possible irregularities of numerical variables.

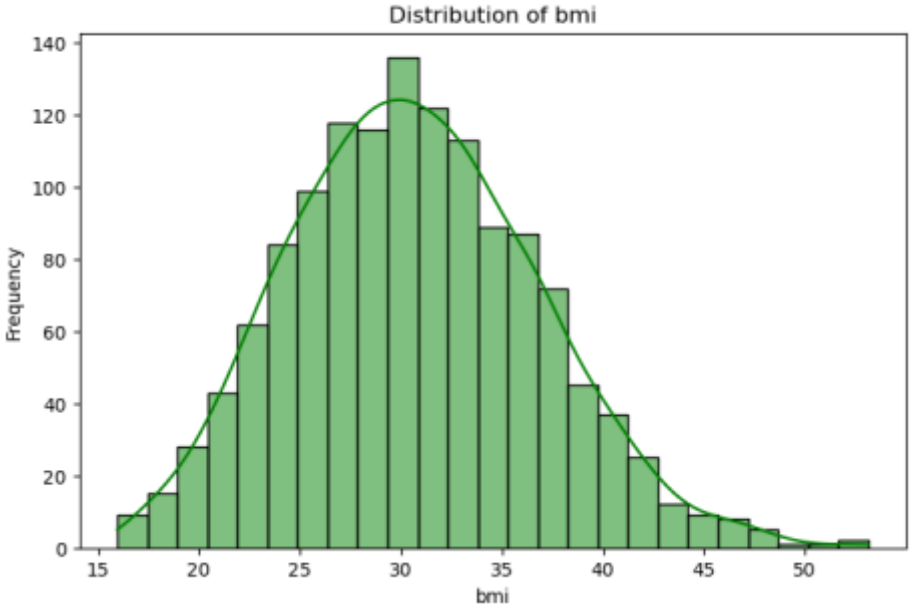
```
numerical_columns = data.select_dtypes(include=['int64', 'float64']).columns
print("\n### Numerical Columns Analysis ###")
for col in numerical_columns:
    if col != 'charges':  # Hedef değişken ayrı analiz edildi
        plt.figure(figsize=(8, 5))
        sns.histplot(data[col], kde=True, color="green")
        plt.title(f"Distribution of {col}")
        plt.xlabel(col)
        plt.ylabel("Frequency")
```

As can be seen in the image below, although the age variable is evenly distributed, there is a concentration between the ages of 20-30. it seems that there are more young people in the data set.

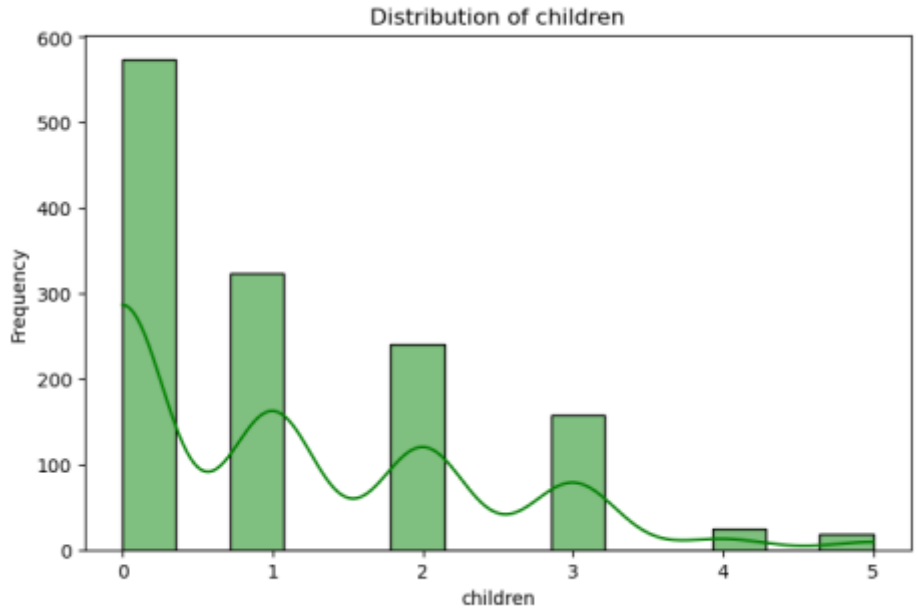
```
### Numerical Columns Analysis ###
```



The bottom image shows that the bmi variable has a normal distribution. it is generally concentrated between 25-35. As a result of this normal distribution, we can interpret that there are no outliers.

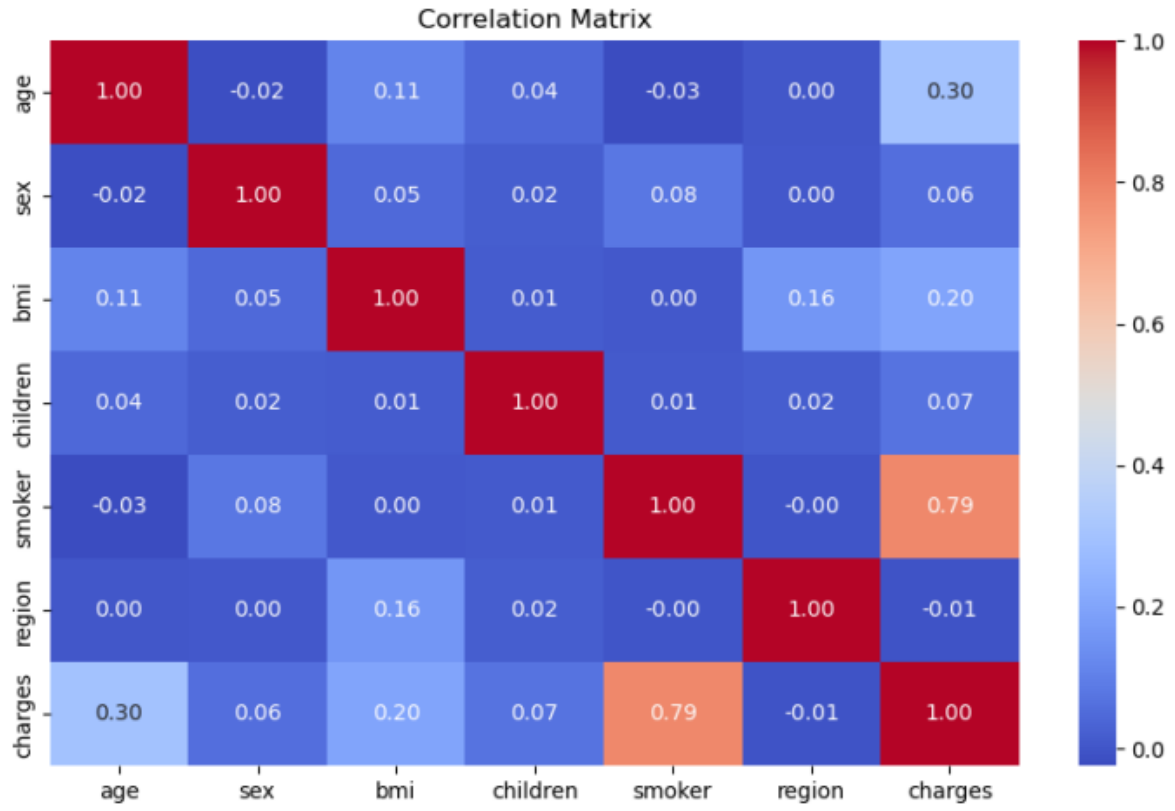


The other image shows the distribution of the number of children. Here, it is seen that most of the people in the dataset have 0 and 1 child.



In the image below, the correlation matrix is visualized to show the relationship between the variables in the data set. The values in this matrix measure the relationship between variables. They take values between -1 and 1. Positive values indicate a positive relationship between variables, while negative values allow us to understand that there is a negative relationship. If the value is close to 0, it means that there is no relationship between them. Here we can see that “smoker” and “bmi” have a strong positive relationship on the target variable, which leads to higher costs for smokers. Or people with a high body index also have higher costs. The variable “age” also has a positive relationship. the other two variables do not have a significant relationship with the target variable. By looking at the correlation matrix here we can comment on adding or removing features. in this assignment we will add features. The variable “smoker” has a strong relationship with the target variable. based on this information we will add a new feature using the variable “smoker”. we will do this in the next steps.

```
plt.figure(figsize=(10, 6))
sns.heatmap(data.corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```

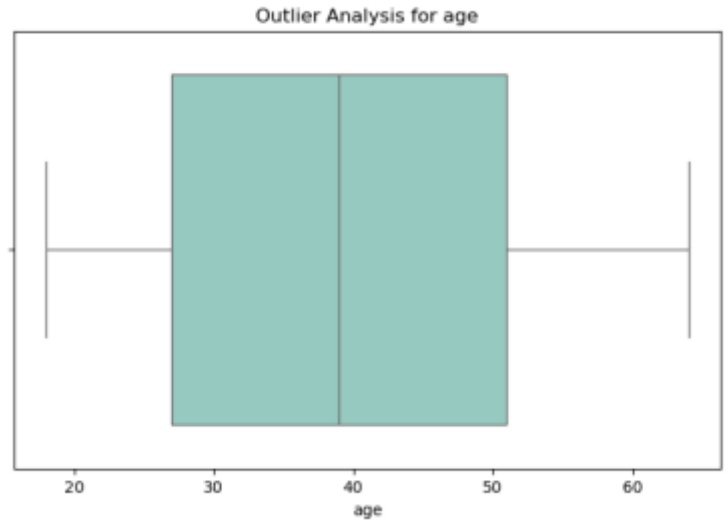


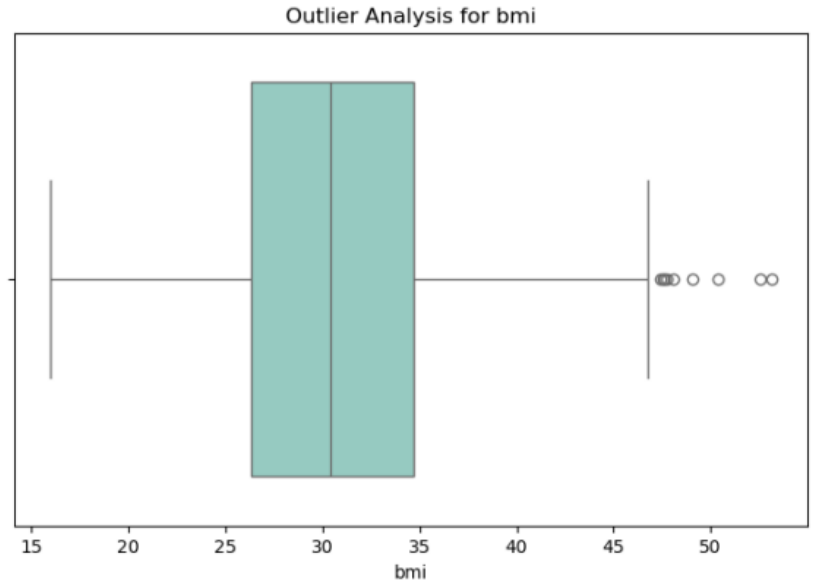
In the image below, we will aim to find outliers in the data set. Outliers have an important place in the data set. Outliers can have a negative impact on the training process of the model, so it is critical to find and preprocess these values. We will perform this analysis in the code below. Then, in the other images, we will look at whether the numeric variables contain outliers. Although every numeric variable except age and children contain outliers, no action will be taken due to the small size of the dataset. Also, there is no need to do anything about the outliers in the target variable, as we have already seen that the data set is concentrated in small costs when we examined its distribution. Since cleaning or processing the outliers in the "charges" variable will have a direct impact on the model, there is no need to process the outliers.

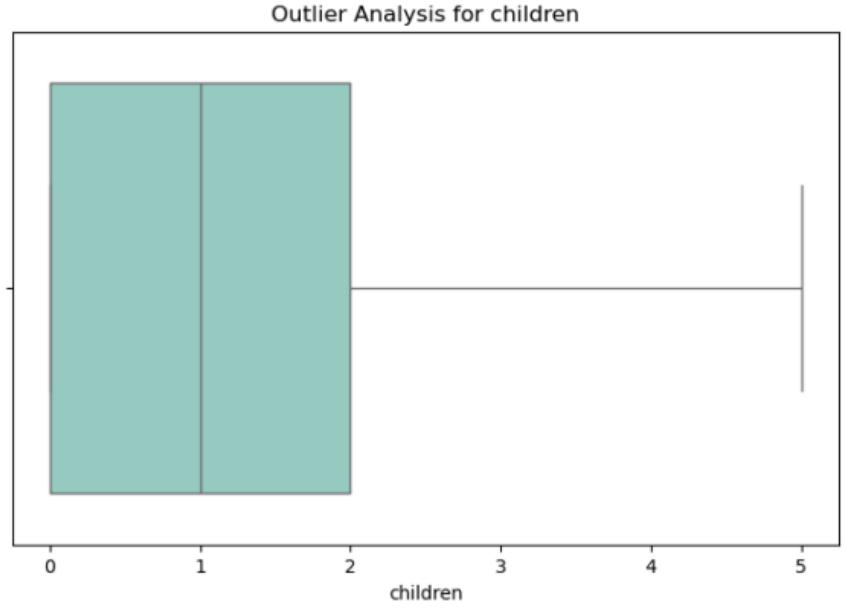
```
print("\n### Outlier Analysis ###")
for col in numerical_columns:
    plt.figure(figsize=(8, 5))
    sns.boxplot(x=data[col], palette="Set3")
    plt.title(f"Outlier Analysis for {col}")
```

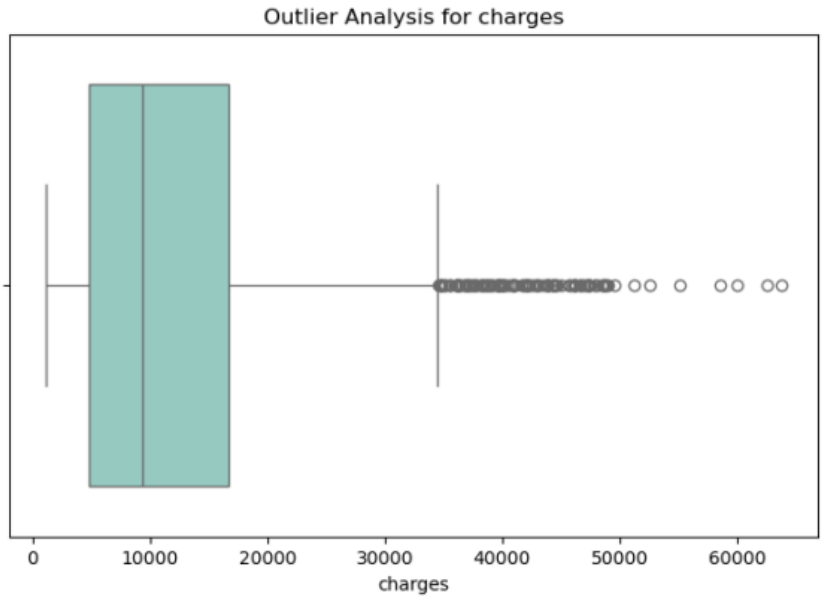


```
    plt.xlabel(col)
    plt.show()
```


Outlier Analysis for age
20
30
40
50
60
age


Outlier Analysis for bmi
15
20
25
30
35
40
45
50
bmi


Outlier Analysis for children
0
1
2
3
4
5
children



The image below shows the separation of independent and dependent variables, the scaling of the data and the division of the data into training and test sets in order to prepare the data set for machine learning models. The target variable, charges, is separated as the dependent variable, while the other features are set as independent variables. The scaling process helps the model to learn more accurately by ensuring that different variables are on the same scale. The data was divided into training and test sets to evaluate the performance of the model. Different values for the test rate (20% and 30%) were used to test the generalization ability of the model. These steps were taken to help the model understand the data better and make more accurate predictions.

```
X = data.drop(columns=['charges'])
y = data['charges']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
↪random_state=42)

X_train_size3, X_test_size3, y_train_size3, y_test_size3 =␣
↪train_test_split(X_scaled, y, test_size=0.3, random_state=42)

X_train_size3_rs21, X_test_size3_rs21, y_train_size3_rs21, y_test_size3_rs21 =␣
↪train_test_split(X_scaled, y, test_size=0.3, random_state=21)
```

The figure below shows how the model is trained and tested using different data split ratios and randomness conditions to evaluate the performance of the linear regression model. The performance of the model is measured by Mean Squared Error (MSE), which measures the difference between the prediction results and the actual values, and R^2 Score, which measures how well the model explains the data. Different test rates and random_state are used to compare how the model works in different situations. This analysis is an important step to

assess the generalizability and accuracy of the model. Thus, we aim to determine the most appropriate data splitting strategy and improve the performance of the model.

```
# Linear Regression
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
y_pred_linear = linear_model.predict(X_test)
mse_linear = mean_squared_error(y_test, y_pred_linear)
```



```
r2_linear = r2_score(y_test, y_pred_linear)
```

```
# Linear Regression 3
linear_model = LinearRegression()
linear_model.fit(X_train_size3, y_train_size3)
y_pred_linear_size3 = linear_model.predict(X_test_size3)
mse_linear_size3 = mean_squared_error(y_test_size3, y_pred_linear_size3)
r2_linear_size3 = r2_score(y_test_size3, y_pred_linear_size3)
```

```
# Linear Regression Random State 21
linear_model = LinearRegression()
linear_model.fit(X_train_size3_rs21, y_train_size3_rs21)
y_pred_linear_size3_rs21 = linear_model.predict(X_test_size3_rs21)
mse_linear_size3_rs21 = mean_squared_error(y_test_size3_rs21,␣
 ↪y_pred_linear_size3_rs21)
r2_linear_size3_rs21 = r2_score(y_test_size3_rs21, y_pred_linear_size3_rs21)
```

In the figure below, a new feature has been added to the dataset to improve the performance of the linear regression model. Based on the correlation matrix above, a new feature called smoker_charges_ratio was created by multiplying the smoker and charges variables. this variable aims to better express the impact of smoking on insurance costs. After the new feature was added, the data was scaled, divided into training and test sets and the model was retrained. The performance of the model was measured again with Mean Squared Error

(MSE) and R^2 Score. This step was done to enable the model to better learn the relationship between smoking and costs and aimed to improve the prediction accuracy.

```
data['smoker_charges_ratio'] = data['smoker'] * data['charges']
```

```
# Linear Regression (With new feature)
X_with_ratio = data.drop(columns=['charges'])
y = data['charges']
X_scaled_with_ratio = scaler.fit_transform(X_with_ratio)

X_train_ratio, X_test_ratio, y_train_ratio, y_test_ratio = train_test_split(
    X_scaled_with_ratio, y, test_size=0.2, random_state=42
)

linear_model_with_ratio = LinearRegression()
linear_model_with_ratio.fit(X_train_ratio, y_train_ratio)
y_pred_linear_ratio = linear_model_with_ratio.predict(X_test_ratio)

mse_linear_ratio = mean_squared_error(y_test_ratio, y_pred_linear_ratio)
r2_linear_ratio = r2_score(y_test_ratio, y_pred_linear_ratio)
```

In the image below, hyperparameter optimization is performed for the Random Forest Regressor model. Parameters such as n_estimators, max_depth, min_samples_split and min_samples_leaf were tested with different combinations for better performance of the model. Hyperparameter optimization was performed using GridSearchCV and the best parameters were selected. With the newly added smoker_charges_ratio feature, the model better learns the relationship between smoking and insurance costs. The performance of the model was re-evaluated with Mean Squared Error (MSE) and R^2 Score, aiming to improve the accuracy and generalizability of the optimized model, an important step towards more reliable and accurate predictions.

```
rf_params = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```



```
X_with_ratio = data.drop(columns=['charges'])
y = data['charges']
X_scaled_with_ratio = scaler.fit_transform(X_with_ratio)

X_train_ratio, X_test_ratio, y_train_ratio, y_test_ratio = train_test_split(
    X_scaled_with_ratio, y, test_size=0.2, random_state=42
)
```

```
rf_grid_with_ratio = GridSearchCV(RandomForestRegressor(random_state=42),␣
 ↪rf_params, scoring='r2', cv=5)
rf_grid_with_ratio.fit(X_train_ratio, y_train_ratio)

best_rf_with_ratio = rf_grid_with_ratio.best_estimator_
best_rf_params_with_ratio = rf_grid_with_ratio.best_params_
y_pred_rf_with_ratio = best_rf_with_ratio.predict(X_test_ratio)

mse_rf_with_ratio = mean_squared_error(y_test_ratio, y_pred_rf_with_ratio)
r2_rf_with_ratio = r2_score(y_test_ratio, y_pred_rf_with_ratio)

print(f"\n### Best Random Forest with Smoker Charges Ratio Parameters ###")
print(best_rf_params_with_ratio)
print(f"MSE (Best Random Forest with Smoker Charges Ratio):␣
 ↪{mse_rf_with_ratio}")
print(f"R² Score (Best Random Forest with Smoker Charges Ratio):␣
 ↪{r2_rf_with_ratio}")
```

The figure below is created to compare the performance of different models. The performance of the models is evaluated with Mean Squared Error (MSE) and R^2 Score metrics. The Best Random Forest with Smoker Charges Ratio model performed the best with the lowest MSE value (1.49746e+07) and the highest R^2 score (0.909454). This result reveals that the Random Forest model can predict insurance costs more accurately with the smoker_charges_ratio feature. On the other hand, Linear Regression models performed poorer with higher MSE values and lower R^2 scores. This comparison shows the importance of proper feature engineering and model selection in improving prediction accuracy.

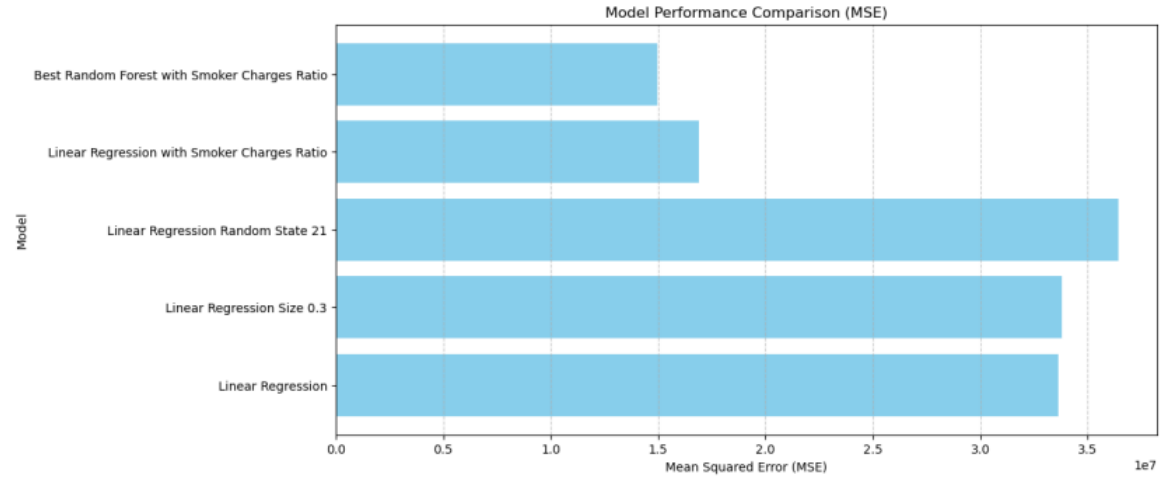
```
results = {
    "Model": ["Linear Regression", "Linear Regression Size 0.3", "Linear␣
 ↪Regression Random State 21",
              "Linear Regression with Smoker Charges Ratio", "Best Random␣
 ↪Forest with Smoker Charges Ratio"],
    "MSE": [mse_linear, mse_linear_size3, mse_linear_size3_rs21,␣
 ↪mse_linear_ratio, mse_rf_with_ratio],
    "R2": [r2_linear, r2_linear_size3, r2_linear_size3_rs21, r2_linear_ratio,␣
 ↪r2_rf_with_ratio]
}

results_df = pd.DataFrame(results)
print("\n### Model Performance Comparison###")
```

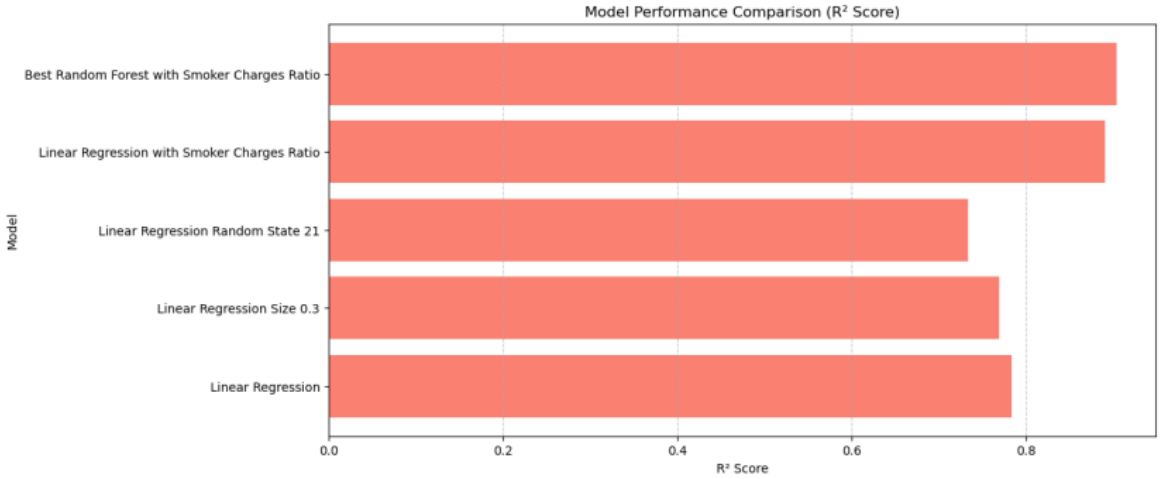


```
print(results_df)
```

```
### Model Performance Comparison###
                                    Model           MSE        R2
0                         Linear Regression  3.363521e+07  0.783346
1                Linear Regression Size 0.3  3.380547e+07  0.769442
2            Linear Regression Random State 21  3.640549e+07  0.734002
3  Linear Regression with Smoker Charges Ratio  1.690800e+07  0.891091
4  Best Random Forest with Smoker Charges Ratio  1.497465e+07  0.903544
```

The figure below visualizes the Mean Squared Error (MSE) and R^2 Score metrics in bar charts to compare the performance of different models. The MSE graph shows the error values, while the R^2 graph shows how well the models explain the data. These visualizations clearly illustrate the performance differences between the models, providing a quick comparison to choose the best model.





Conclusion

In this assignment, I analyzed and evaluated different models for predicting insurance costs. The analysis and visualizations we did to understand the data were instrumental in identifying the relationships of variables and potential problems in the data. We aimed to improve the performance of the models by adding a new feature, smoker_charges_ratio. In particular, we significantly improved the prediction accuracy by optimizing the hyperparameters in the Random Forest model. According to the results, the Random Forest model was the most successful model with the lowest error (MSE) and the highest accuracy (R^2) score. In this process, we observed the impact of proper feature engineering and model selection on prediction accuracy. The Random Forest model's ability to learn complex relationships better allowed us to achieve the most effective results in this assignment.