RECEP TAYYIP ERDOGAN UNIVERSITY

FACULTY OF ENGINEERING AND ARCHITECTURE

COMPUTER ENGINEERING DEPARTMENT

DATA MINING

2024-2025

Student Name Surname

Doğukan Erzurum

Student No

201401023

Instructor

Dr. Öğr. Üyesi ABDULGANİ KAHRAMAN

RIZE

1.1. Introduction

1.1.1. Project Objective

This project aims to predict a target variable using machine learning algorithms. In particular, the decision tree algorithm will be applied to predict the target variable and the model performance will be evaluated. Furthermore, the performance of the decision tree model will be analyzed by comparing it with an alternative model, Random Forest.Proje sırasında:

- Steps such as data processing, feature selection, and data visualization have been thoroughly addressed.
- Performance evaluation metrics such as Accuracy, Precision, Recall, and F1-Score have been utilized.
- Visualizations like the Confusion Matrix have been used to analyze the strengths and weaknesses of the models.

1.1.2. About the Dataset

In this project, the Diabetes Dataset obtained from the Kaggle platform was used. The dataset includes various health measurements of individuals and indicates whether they have diabetes or not.

- **Total Number of Observations:** 768
- **Number of Columns**: 9
- **Target Variable (Outcome)**:
    - 0: The individual is not diabetic.
    - 1: The individual is diabetic.
- **Independent Variables**:
    - **Pregnancies**: Number of pregnancies.
    - **Glucose**: Blood glucose level.
    - **BloodPressure**: Blood pressure.
    - **SkinThickness**: Skin thickness.
    - **Insulin**: Insulin level.
    - **BMI**: Body Mass Index.
    - **DiabetesPedigreeFunction**: Diabetes risk factor (genetic predisposition).
    - **Age**: Age.

1.1.3. Project Process

The project involves the following steps:

1. **Data Exploration and Preprocessing:**
    - Identifying and handling missing values in the dataset.
    - Limiting outliers.
    - Scaling continuous variables.
2. **Modeling**:
    - Training and testing Decision Tree and Random Forest algorithms.
3. **Performance Evaluation:**
    - Using metrics such as Accuracy, Precision, Recall, and F1-Score,
    - Analyzing results through a Confusion Matrix.

4. **Model Comparison and Interpretation:**
    - Comparing the models and interpreting the results.

1.2. Data Exploration and Processing

In this section, we will examine the dataset in detail, handle missing and outlier values, perform feature selection, and apply data preprocessing steps. Our goal is to prepare the dataset for modeling and take the necessary steps to enhance model performance..

1.2.1. Loading and Initial Examination of the Dataset

First, we load the dataset using the pandas library and review its basic information. This includes:.

```
# Import necessary libraries
import pandas as pd

# Load the dataset (Replace with the correct file name after
uploading)
data = pd.read_csv('diabetes.csv')

# General information about the dataset
print("Dataset General Information:")
print(data.info())

# Display the first few rows
print("\nFirst 5 Rows of the Dataset:")
print(data.head())
```

Output:

```
Dataset General Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
None
```

```
First 5 Rows of the Dataset:
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

Description:

- The dataset contains a total of 768 observations and 9 columns.
- The Outcome column is defined as the target variable, indicating the diabetes status of individuals. This variable is categorized as:
    - 0: Not diabetic,
    - 1: Diabetic,
- Initial examination revealed no missing values in the dataset. However, it was noted that 0 values in some columns might represent missing data. This issue has been identified as an element requiring further analysis.

1.2.2. Analysis of Missing Values

In the dataset, 0 values in certain columns have been identified as biologically implausible measurements. This indicates the presence of missing data in these columns. Analyzing and handling missing values using appropriate methods has been identified as a critical step to enhance the model's accuracy and reliability.

```
# Check for missing values
print("\nMissing Values Check:")
print(data.isnull().sum())

# Statistical summary
print("\nStatistical Summary:")
print(data.describe())
```

Çıktı:

```
Missing Values Check:
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```
Statistical Summary:
       Pregnancies     Glucose  BloodPressure  SkinThickness
Insulin  \
count   768.000000  768.000000     768.000000     768.000000
768.000000
mean      3.845052  120.894531      69.105469      20.536458
79.799479
std       3.369578   31.972618      19.355807      15.952218
115.244002
min       0.000000    0.000000       0.000000       0.000000
0.000000
25%       1.000000   99.000000      62.000000       0.000000
0.000000
50%       3.000000  117.000000      72.000000      23.000000
30.500000
75%       6.000000  140.250000      80.000000      32.000000
127.250000
max      17.000000  199.000000     122.000000      99.000000
846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

Description:

- According to the missing value check results, no directly missing values were identified in the dataset. However, upon examining the statistical summary, it was observed that the **min** and **25%** values for certain columns are **0**. Specifically, **Glucose, BloodPressure, SkinThickness, Insulin**, and **BMI** columns contain 0 values, which are biologically implausible and can be considered as missing data.
- This indicates the necessity of handling missing data with appropriate methods, which will have a critical impact on the accuracy of the modeling process. Before imputing the missing values, a detailed analysis of data distributions should be conducted, and suitable imputation techniques should be applied.

1.2.3. Marking and Filling Missing Values

To properly handle missing data, **0** values in the **Glucose**, **BloodPressure**, **SkinThickness**, **Insulin**, and **BMI** columns were treated as missing and replaced with **NaN**. After this operation, the number of missing values was determined as follows:

```
# Check for 0 values (0 is not meaningful in these columns)
columns_with_zeros = ['Glucose', 'BloodPressure', 'SkinThickness',
'Insulin', 'BMI']
for column in columns_with_zeros:
    print(f"Number of 0 values in {column} column:
{data[column].value_counts().get(0, 0)}")

# Replace 0 values with NaN
data[columns_with_zeros] = data[columns_with_zeros].replace(0,
pd.NA)

# Check for missing values (again)
print("\nMissing Values (after replacing 0s with NaN):")
print(data.isnull().sum())
```

Output:

```
Missing Values (after replacing 0s with NaN):
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

Description:

- **Glucose**: 5 missing values
- **BloodPressure**: 35 missing values
- **SkinThickness**: 227 missing values
- **Insulin**: 374 missing values
- **BMI**: 11 missing values

This analysis revealed a particularly high number of missing values in the **Insulin** and **SkinThickness** columns. To address this, the missing values were filled using the **median values** of the respective columns. The median was chosen to mitigate the impact of outliers, ensuring a reliable data imputation process.

After completing the filling operation, the dataset was rechecked, and it was confirmed that no missing values remained. The dataset is now ready for modeling.

```
# Fill missing values with the median of each column
for column in columns_with_zeros:
    data[column] = data[column].fillna(data[column].median())

# Check for missing values (Final check)
print("\nMissing Values (After Filling):")
print(data.isnull().sum())
```

Output:

```
Missing Values (After Filling):
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Description:

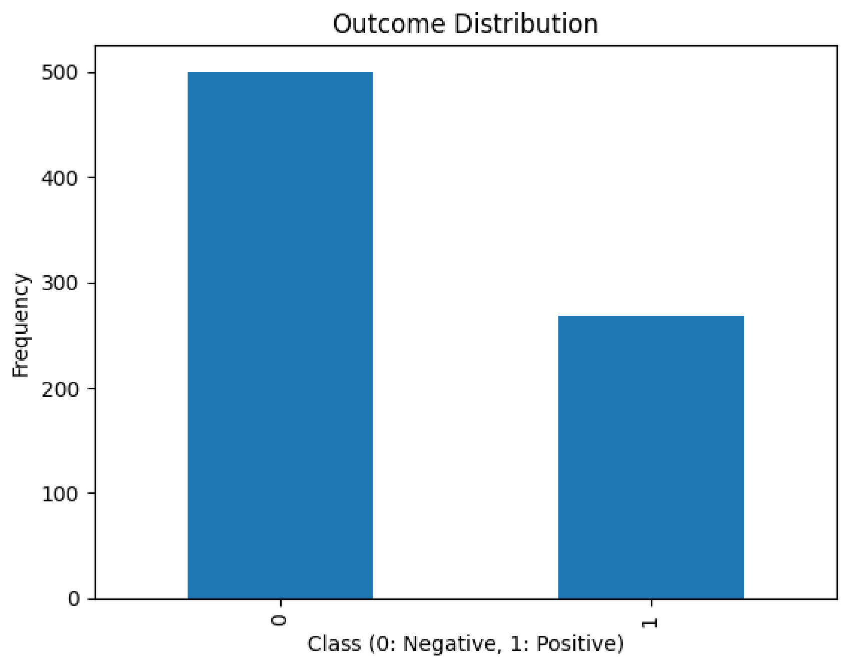
- All missing values have been successfully filled and our dataset is now missing value free.

1.2.4. Examination of the Target Variable Distribution

The class distribution of the **Outcome** column, which serves as the target variable, was analyzed to evaluate class imbalance within the dataset. The following observations were made:

```
import matplotlib.pyplot as plt

# Visualize the distribution of the Outcome column
data['Outcome'].value_counts().plot(kind='bar', title='Outcome
Distribution')
plt.xlabel('Class (0: Negative, 1: Positive)')
plt.ylabel('Frequency')
plt.show()
```

Output:



Comments:

- **Outcome = 0 (Non-diabetic):** Approximately 500 observations.
- **Outcome = 1 (Diabetic):** Approximately 268 observations.
- **Class Imbalance:** There is a noticeable class imbalance in the dataset. The number of non-diabetic individuals significantly outweighs the number of diabetic individuals

1.2.5. Examination of the Relationship Between Features and the Target Variable

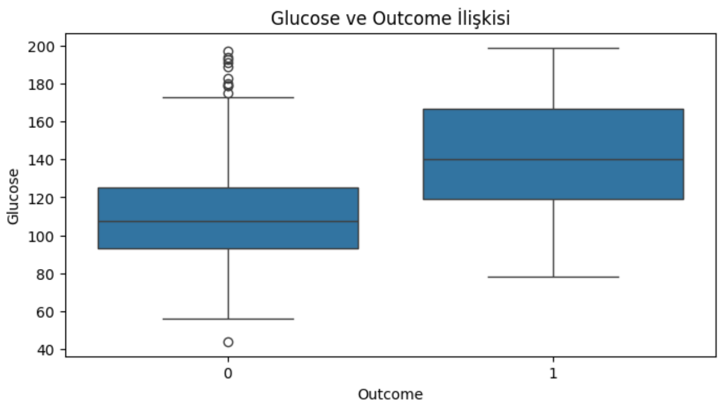
The relationships between continuous features in the dataset and the target variable (Outcome) were analyzed and visualized using boxplots. Below are the observations and analyses for each feature:

```
# Relationship between continuous variables and Outcome
columns_to_plot = ['Glucose', 'BloodPressure', 'BMI', 'Age',
'Insulin']
for column in columns_to_plot:
    plt.figure(figsize=(8, 4))
    sns.boxplot(data=data, x='Outcome', y=column)
    plt.title(f'Relationship Between {column} and Outcome')
    plt.show()
```

Output:

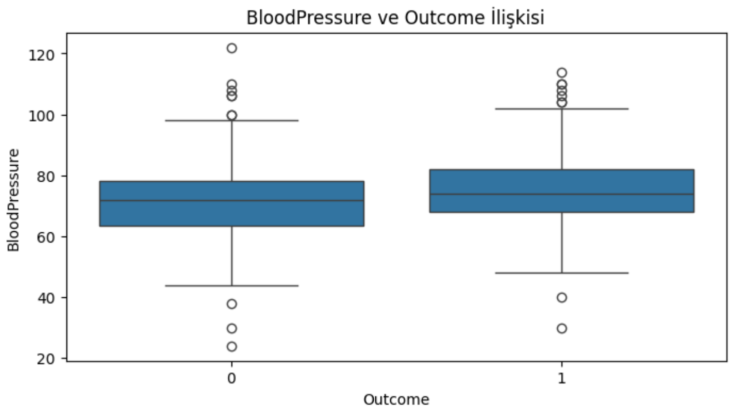
Glucose and Outcome Relationship:

- The glucose levels of diabetic individuals (**Outcome = 1**) are significantly higher compared to non-diabetic individuals (**Outcome = 0**).
- This indicates that glucose levels are a critical feature for diabetes detection.



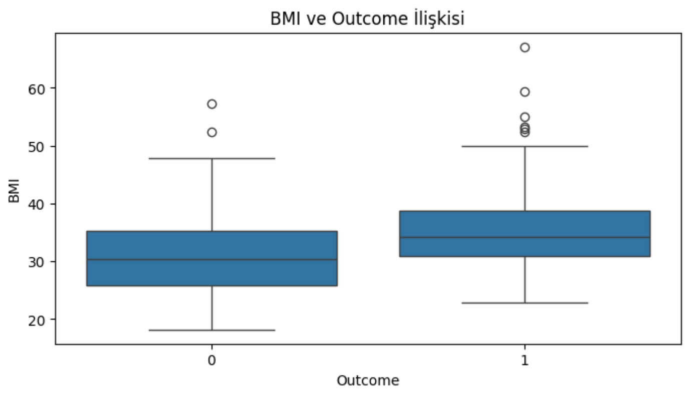
BloodPressure and Outcome Relationship:

- No significant differences were observed in blood pressure distributions between diabetic and non-diabetic individuals.
- However, some outliers in blood pressure values were identified.



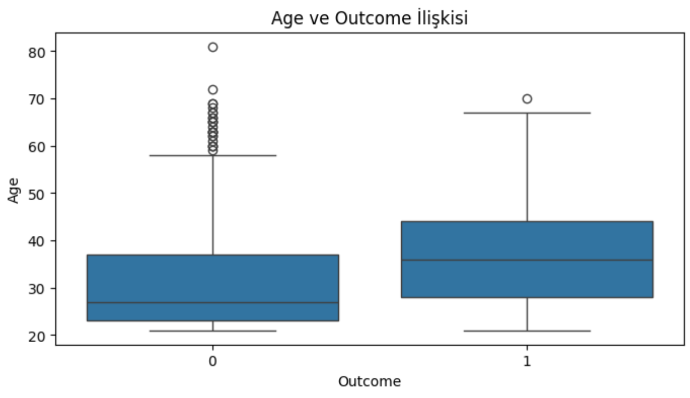
BMI and Outcome Relationship:

- Diabetic individuals (**Outcome = 1**) tend to have higher BMI (Body Mass Index) values compared to non-diabetic individuals.
- This suggests that BMI is significantly associated with diabetes.



Age and Outcome Relationship:

- The average age of diabetic individuals is higher than that of non-diabetic individuals.
- The likelihood of diabetes increases with age, highlighting age as an important factor.



Insulin and Outcome Relationship:

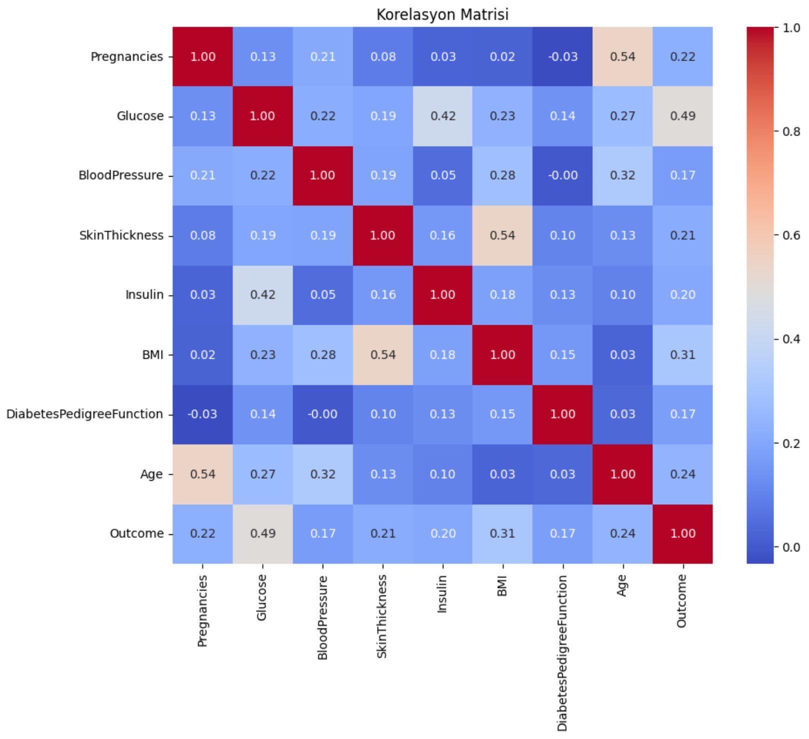
- No clear differences in insulin levels were observed between diabetic and non-diabetic individuals.
- However, the insulin data contains a considerable number of outliers.

Conclusion:
Features such as **Glucose**, **BMI**, and **Age** show a significant relationship with the target variable and should be included in the model. Other features require further analysis during data cleaning and preprocessing steps.

1.2.6. Correlation Analysis

To examine the relationships between features in the dataset, a correlation matrix was created and visualized. The following observations were made:

```
# Correlation matrix and heatmap
plt.figure(figsize=(10, 8))
corr = data.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
```



```
plt.show()
```

Output:

- **Features with the Highest Correlation to Outcome**:
  - **Glucose**: 0.49
  - **BMI**: 0.31
  - **Age**: 0.24
- There are no highly correlated features in the dataset. This indicates a low risk of **multicollinearity**, which could otherwise negatively impact the model.

Conclusion:
Glucose, BMI, and Age variables show a significant relationship with the Outcome variable and should be prioritized in the modeling process. Additionally, the low correlation between features suggests that the dataset does not suffer from multicollinearity, which supports the reliability of the model.

1.2.7. Handling Outliers

Outliers were identified in the **Insulin**, **SkinThickness**, and **BloodPressure** columns. Since these values could negatively impact data analysis and modeling processes, they were handled using the **IQR (Interquartile Range)** method.

IQR Method:

- **Q1 (1st Quartile):** Represents the first 25% of the data.
- **Q3 (3rd Quartile):** Represents the first 75% of the data.
- **IQR (Interquartile Range):** Calculated as Q3 - Q1.
- The boundaries for outliers are defined as **Lower Bound (Q1 - 1.5 * IQR)** and **Upper Bound (Q3 + 1.5 * IQR)**.

```
# IQR yöntemiyle aykırı değerlerin sınırlandırılması
for column in ['Insulin', 'SkinThickness', 'BloodPressure']:
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    data[column] = data[column].clip(lower_bound, upper_bound)

# İşlem sonrası sütunların güncel istatistikleri
print(data[['Insulin', 'SkinThickness',
'BloodPressure']].describe())
```

Output: As a result of this process, the distribution of the columns became more normalized. The updated statistics are as follows:

```
           Insulin  SkinThickness  BloodPressure
count   768.000000     768.000000     768.000000
mean    124.691081      28.866536      72.358073
std       7.913595       7.442353      11.697097
min     112.875000      14.500000      40.000000
25%     121.500000      25.000000      64.000000
50%     125.000000      29.000000      72.000000
75%     127.250000      32.000000      80.000000
max     135.875000      42.500000     104.000000
```

- **Insulin:** Mean value 124.69, standard deviation 7.91, values are constrained between 112.87 and 135.87.
- **SkinThickness:** Mean value 28.86, standard deviation 7.44, values are constrained between 14.50 and 42.50.
- **BloodPressure:** Mean value 72.36, standard deviation 11.69, values are constrained between 40.00 and 104.00.

Conclusion:
By limiting outliers, the consistency of the dataset has been improved, and the risk of negatively impacting model performance has been minimized.

1.2.8. Feature Engineering

To enhance model performance, new features were added to the dataset. The transformation applied to the age variable is detailed below.

1.2.8.1. Creating Age Groups

Definition of Age Groups:
The age variable was divided into specific intervals to create categories such as **Young**, **Middle-Aged**, **Old**, and **Very Old**. These groups were created to better analyze the impact of age on diabetes.

Categories:

- **20-30:** Young
- **30-40:** Middle-Aged
- **40-50:** Old
- **50 and above:** Very Old

```
# Yaş gruplarının oluşturulması
bins_age = [20, 30, 40, 50, 100]
labels_age = ['Genç', 'Orta Yaşlı', 'Yaşlı', 'Çok Yaşlı']
data['AgeGroup'] = pd.cut(data['Age'], bins=bins_age,
labels=labels_age)

# Yaş gruplarının dağılımı
print("\nYaş Grupları Dağılımı:")
print(data['AgeGroup'].value_counts())
```

Output:

```
Yaş Grupları Dağılımı:
Genç           417
Orta Yaşlı     157
Yaşlı          113
Çok Yaşlı       81
Name: AgeGroup, dtype: int64
```

Conclusion:
This transformation allows for a more detailed analysis of the impact of age groups on the target variable. Such feature engineering can enhance the model's learning capacity and improve results.

1.2.8.2. Creating Pregnancy Groups

Definition of Pregnancy Groups:
The number of pregnancies variable was divided into specific intervals to create categories such as **Low**, **Medium**, and **High**. These groups were designed to better analyze the impact of the number of pregnancies on diabetes.

Categories:

- **-1 to 2:** Low
- **2 to 6:** Medium
- **6 and above:** High

```
# Gebelik gruplarının oluşturulması
bins_preg = [-1, 2, 6, 20]
labels_preg = ['Düşük', 'Orta', 'Yüksek']
data['PregnanciesGroup'] = pd.cut(data['Pregnancies'],
bins=bins_preg, labels=labels_preg)

# Gebelik gruplarının dağılımı
print("\nGebelik Grupları Dağılımı:")
print(data['PregnanciesGroup'].value_counts())
```

Output:

```
Gebelik Grupları Dağılımı:
Düşük     349
Orta      250
Yüksek    169
Name: PregnanciesGroup, dtype: int64
```

Conclusion:
This transformation allows for a more detailed analysis of the impact of pregnancy groups on the target variable. These groups, along with the **AgeGroup** feature, can be included as categorical variables in the model to improve its performance.

1.2.9. Feature Scaling

To improve model performance, continuous variables in the dataset were scaled using **StandardScaler**. This process standardized the variables and eliminated issues that could arise from differing scales during the model learning process.

Process:

Scaled Columns:

- Glucose
- BloodPressure
- SkinThickness

- Insulin
- BMI
- Age

```
from sklearn.preprocessing import StandardScaler

# Ölçeklenecek sütunlar
columns_to_scale = ['Glucose', 'BloodPressure', 'SkinThickness',
'Insulin', 'BMI', 'Age']

# StandardScaler ile ölçekleme
scaler = StandardScaler()
data[columns_to_scale] =
scaler.fit_transform(data[columns_to_scale])

# Ölçeklenmiş sütunların ilk 5 satırı
print("\nÖlçeklenmiş Sütunlar (İlk 5 Satır):")
print(data[columns_to_scale].head())
```

Output:

```
Ölçeklenmiş Sütunlar (İlk 5 Satır):
    Glucose  BloodPressure  SkinThickness   Insulin       BMI       Age
0  0.866045      -0.030632       0.824667  0.039062  0.166619  1.425995
1 -1.205066      -0.543914       0.017945  0.039062 -0.852200 -0.190672
2  2.016662      -0.715008       0.017945  0.039062 -1.332500 -0.105584
3 -1.073567      -0.543914      -0.788777 -1.494110 -0.633881 -1.041549
4  0.504422      -2.768136       0.824667  1.414175  1.549303 -0.020496
```

Explanation:

- Continuous variables were standardized using **StandardScaler**.
- As a result, the features were scaled to have a **mean of 0** and a **standard deviation of 1**.
- This scaling eliminates the effects of differing scales between variables, helping the model produce more consistent results.

1.3. Data Splitting

In this step, the dataset was split into **training** and **testing** sets to prepare for the modeling process. The training set will be used for model learning, while the testing set will evaluate model performance. The splitting process was conducted to maintain the class balance of the target variable.

1.3.1. Splitting Training and Testing Sets

The dataset was split into 80% training and 20% testing sets using the **train_test_split** function. The **stratify** parameter was used to ensure that the class proportions of the target variable, **Outcome**, are maintained in both sets.

Process:

- **Independent Variables (X):**
  - The **Outcome**, **AgeGroup**, and **PregnanciesGroup** columns were excluded to focus the model solely on continuous and core independent variables.
- **Dependent Variable (y):**
  - The **Outcome** column was designated as the target variable.

```
from sklearn.model_selection import train_test_split

# Bağımsız değişkenler (X) ve bağımlı değişken (y) ayrımı
X = data.drop(['Outcome', 'AgeGroup', 'PregnanciesGroup'], axis=1)
# Outcome ve grupları çıkarıyoruz
y = data['Outcome']

# Veriyi eğitim ve test setlerine bölme
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

# Veri seti boyutlarını kontrol edelim
print("Eğitim Verisi Boyutu:", X_train.shape)
print("Test Verisi Boyutu:", X_test.shape)
```

Output:

```
Eğitim Verisi Boyutu: (614, 8)
Test Verisi Boyutu: (154, 8)
```

Explanation:

- **Training Set:** Comprises 614 observations (80% of the data).
- **Testing Set:** Comprises 154 observations (20% of the data).
- **Stratify Parameter:** Ensures that class proportions of the target variable are balanced across both sets.

Conclusion:
This step ensures that the model gains a generalizable structure during training and is fairly evaluated during testing.

1.4. Modeling and Evaluation

In this section, the **Decision Tree** model was trained and evaluated on the test set. The performance of the model was analyzed using metrics such as accuracy, precision, recall, and F1-Score. Below are the detailed steps and results for the Decision Tree model.

1.4.1. Decision Tree Model

The Decision Tree model was defined using the sklearn library and trained on the training set. Predictions were made using the test set, and performance metrics were calculated.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score,
confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Karar Ağacı modelini tanımlayalım
dt_model = DecisionTreeClassifier(random_state=42)

# Modeli eğitelim
dt_model.fit(X_train, y_train)

# Test seti ile tahmin yapalım
y_pred_dt = dt_model.predict(X_test)

# Performans değerlendirme
print("\nKarar Ağacı Modeli Performansı:")
print("Accuracy:", accuracy_score(y_test, y_pred_dt))
print("\nClassification Report:")
print(classification_report(y_test, y_pred_dt))

# Confusion Matrix Görselleştirme
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)
sns.heatmap(conf_matrix_dt, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix - Karar Ağacı")
plt.xlabel("Tahmin")
plt.ylabel("Gerçek")
plt.show()
```

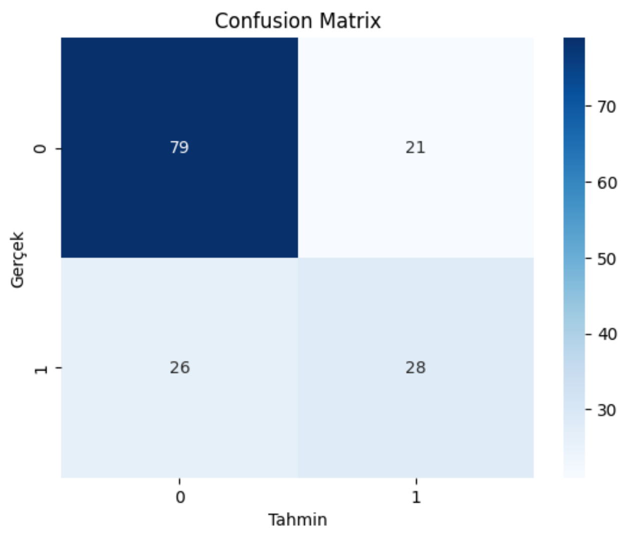
Output:

```
Karar Ağacı Modeli Performansı:
Accuracy: 0.6948051948051948

Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.79      0.77       100
           1       0.57      0.52      0.54        54

    accuracy                           0.69       154
   macro avg       0.66      0.65      0.66       154
weighted avg       0.69      0.69      0.69       154
```

Performance Metrics:

- **Accuracy:** 0.69
- **Precision:**
    - Class 0 (Non-diabetic): 0.75
    - Class 1 (Diabetic): 0.57
- **Recall:**
    - Class 0 (Non-diabetic): 0.79
    - Class 1 (Diabetic): 0.52
- **F1-Score:**
    - Class 0 (Non-diabetic): 0.77
    - Class 1 (Diabetic): 0.54

Confusion Matrix:



Conclusion:

- The model performs better in correctly predicting non-diabetic individuals (79% accuracy) but struggles to identify diabetic individuals (52% accuracy).
- **F1-Score** and other metrics indicate that the overall performance of the model needs improvement.
- The model seems to be affected by class imbalance. To address this issue, sampling techniques or alternative modeling strategies can be employed.

1.4.2. Random Forest Model

In this section, the **Random Forest** model was trained and evaluated on the test set. The model's performance was analyzed using metrics such as accuracy, precision, recall, and F1-Score. Below are the detailed steps and results for the Random Forest model. The **Random Forest** model was defined using the sklearn library and trained on the training set with **100 estimators.** Predictions were made on the test set, and performance metrics were calculated.

```
from sklearn.ensemble import RandomForestClassifier

# Random Forest modelini tanımlayalım
rf_model = RandomForestClassifier(random_state=42, n_estimators=100)

# Modeli eğitelim
rf_model.fit(X_train, y_train)

# Test seti ile tahmin yapalım
y_pred_rf = rf_model.predict(X_test)

# Performans değerlendirme
print("\nRandom Forest Modeli Performansı:")
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("\nClassification Report:")
print(classification_report(y_test, y_pred_rf))

# Confusion Matrix Görselleştirme
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
sns.heatmap(conf_matrix_rf, annot=True, fmt="d", cmap="Greens")
plt.title("Confusion Matrix - Random Forest")
plt.xlabel("Tahmin")
plt.ylabel("Gerçek")
plt.show()
```

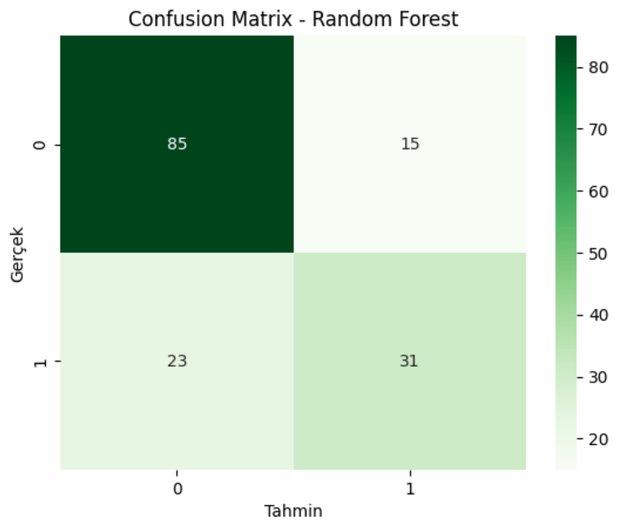
Output:

```
Random Forest Modeli Performansı:
Accuracy: 0.7532467532467533

Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.85      0.82       100
           1       0.67      0.57      0.62        54

    accuracy                           0.75       154
   macro avg       0.73      0.71      0.72       154
weighted avg       0.75      0.75      0.75       154
```

Performance Metrics:

- **Accuracy:** 0.75
- **Precision:**
  - Class 0 (Non-diabetic): 0.79
  - Class 1 (Diabetic): 0.67
- **Recall:**
  - Class 0 (Non-diabetic): 0.85
  - Class 1 (Diabetic): 0.57
- **F1-Score:**
  - Class 0 (Non-diabetic): 0.82
  - Class 1 (Diabetic): 0.62

Confusion Matrix:



Conclusion:

- The Random Forest model performs better in identifying non-diabetic individuals (85% accuracy).
- It shows limited success in detecting diabetic individuals (57% accuracy).
- The model's accuracy and other metrics are higher compared to the Decision Tree model.
- The effects of class imbalance are still noticeable. Performance can be further improved by incorporating class weights or sampling techniques.

1.4.3. Model Comparison

This section compares the performance metrics of the **Decision Tree** and **Random Forest** models. The table below summarizes their accuracy, precision, recall, and F1-Score for both Class 0 and Class 1:

Model	Accuracy	Class 0 Precision	Class 0 Recall	Class 1 Precision	Class 1 Recall	Class 1 F1-Score
Decision Tree	0.6948	0.75	0.79	0.57	0.52	0.54
Random Forest	0.7532	0.79	0.85	0.67	0.57	0.62

Analysis:

1. **Accuracy:**
   - The **Random Forest** model (75.32%) outperformed the **Decision Tree** model (69.48%) in terms of accuracy.
2. **Class 0 (Non-diabetic):**
   - **Precision:** Both models performed well, but Random Forest (0.79) was slightly better than Decision Tree (0.75).
   - **Recall:** Random Forest (85%) demonstrated better recall compared to Decision Tree (79%).
3. **Class 1 (Diabetic):**
   - **Precision:** Random Forest (67%) showed higher precision than Decision Tree (57%).
   - **Recall:** Random Forest (57%) outperformed Decision Tree (52%) in recall.
   - **F1-Score:** Random Forest (62%) achieved a higher F1-Score than Decision Tree (54%).

Conclusion:

- The **Random Forest** model outperformed the **Decision Tree** model across all metrics.
- It was particularly better at predicting diabetic individuals (Class 1).
- These results indicate that the **Random Forest** model is a more effective choice for this dataset.

1.4.4. Strengths and Weaknesses

Model	Strengths	Weaknesses
Decision Tree	- Fast training and prediction time. - Easy to interpret. - Does not create an overly complex model.	- Lower accuracy. - More likely to be affected by class imbalance. - Prone to overfitting.
Random Forest	- Higher accuracy and recall. - Reduces overfitting risk.. - Performs better on complex data.	- Higher computational cost. - Less interpretable.

1.5. Model Selection

Based on the performance results, the **Random Forest** model demonstrated better performance in both accuracy and the positive class (**Class 1 - Diabetic**).

- **Accuracy:** The Random Forest model achieved a higher accuracy rate (75.32%) compared to the Decision Tree model.
- **Positive Class Performance:** The Random Forest model outperformed the Decision Tree model in predicting diabetic individuals (Precision, Recall, and F1-Score).

Conclusion:

For this dataset and problem, the **Random Forest model** is selected as the most suitable model. Its ability to handle challenges such as class imbalance and its superior overall performance make it the preferred choice.

1.6. Conclusion

In this study, the **Diabetes Dataset** obtained from Kaggle was used to build models with the **Decision Tree** and **Random Forest** algorithms. The primary objective was to identify the most suitable model for accurately classifying diabetic patients.

Key Findings:

- The **Random Forest** model was identified as the superior model due to its higher accuracy (75.32%) and more balanced performance compared to the Decision Tree model.
- The Random Forest model demonstrated better prediction results for both negative and positive classes.

Question 2

1. Introduction

1.1. Project Objective

Cluster analysis is an unsupervised learning method used to group different data points into meaningful clusters based on specific characteristics. In this study, we aimed to **perform customer segmentation using the K-means algorithm**. Through this segmentation, we aimed to identify common behaviors or attributes of customers in the dataset, providing meaningful insights for business or marketing applications.

The dataset used contains **information on wholesale customer expenditures** and was obtained from the UCI Machine Learning Repository. It includes features such as expenditures in different product categories, the region of the customer, and customer type. These features provide sufficient information to group customers into meaningful clusters.

The steps followed in this study are as follows:

1. **Data Preprocessing:** Missing value analysis was conducted, outliers were cleaned, and all features were scaled.
2. **PCA (Principal Component Analysis):** The dimensionality of the dataset was reduced, enabling the K-means algorithm to work more efficiently.
3. **K-means Clustering:** The number of clusters was determined using the Elbow method and the Silhouette score.
4. **Cluster Analysis:** The characteristics of the obtained clusters were examined and interpreted in terms of business applications.

In this report, each step is explained in detail, the codes and outputs used are presented, and the results are interpreted.

Dataset and Preprocessing

In this study, we used the **Wholesale Customers Data Set**, obtained from the **UCI Machine Learning Repository**. The dataset contains information on expenditures in various product categories and consists of 8 features for a total of **440 customers**. The features are as follows:

1. **Channel:** Indicates the type of customer. It consists of two categories:
   - 1: Horeca (Hotel/Restaurant/Café)
   - 2: Retail
2. **Region:** Indicates the geographic region of the customer. It consists of three categories:
   - 1: Lisbon

    - 2: Oporto
    - 3: Other Regions
3. **Fresh:** Annual spending on fresh products (possibly in pounds).
4. **Milk:** Annual spending on milk products.
5. **Grocery:** Annual spending on grocery products.
6. **Frozen:** Annual spending on frozen products.
7. **Detergents_Paper:** Annual spending on detergents and paper products.
8. **Delicassen:** Annual spending on delicatessen products.

The dataset is suitable for **customer segmentation** analysis because it contains both categorical and numerical features. Before performing the segmentation, the following preprocessing steps were applied:

1. **Missing Value Analysis:** No missing values were found in the dataset. However, for demonstration purposes, artificial missing values could be introduced.
2. **Outlier Detection and Removal:** Significant outliers were detected in all spending categories (e.g., Fresh, Grocery). These outliers were removed using the IQR method.
3. **Scaling:** Since the K-means algorithm relies on distance, all numerical features were scaled to the same range. **StandardScaler** was used to transform the data into a standardized form (mean=0, standard deviation=1).

1.1.2. Project Process

The project process followed in this study consists of the following steps:

1. **Dataset Introduction and Examination:**
    - To understand the structure of the dataset and design an appropriate analysis process, the dataset was first examined. In this step, columns, data types, missing values, and general information about the dataset were analyzed.
2. **Data Preprocessing:**
    - Missing value analysis was performed, and no missing values were found.
    - Outliers were detected and cleaned using the IQR method.
    - All data were scaled using **StandardScaler**.
3. **Dimensionality Reduction (PCA):**
    - PCA (Principal Component Analysis) was applied to reduce the dimensions of the dataset and better represent the distance between clusters. During this process, the number of components explaining 95% of the total variance was determined.
4. **Clustering:**
    - The K-means algorithm was applied.
    - The number of clusters was determined using the Elbow method and the Silhouette score.
    - Clustering results were visualized.
5. **Cluster Analysis:**
    - The characteristics of the obtained clusters were analyzed.
    - Each cluster’s characteristics were examined, and meaningful segmentations were made.
6. **Results and Comments:**

- The findings obtained in the study were evaluated, and insights were drawn in terms of customer segmentation.

1.1.3 Dataset Introduction and Examination

In this step, the dataset was loaded and basic analyses were conducted. The columns, data types, and missing values in the dataset were analyzed. The obtained information is as follows:

```
# Loading the dataset
import pandas as pd

data = pd.read_csv("Wholesale customers data.csv")

# Viewing the first 5 rows of the dataset
print("First 5 Rows:\n", data.head())

# Examining the structure of the dataset
print("\nDataset Information:\n")
data.info()

# Checking for missing values
print("\nMissing Values:\n", data.isnull().sum())
```

Output:

```
First 5 Rows:
    Channel  Region  Fresh  Milk  Grocery  Frozen  Detergents_Paper
Delicassen
0        2       3  12669  9656     7561     214              2674
1338
1        2       3   7057  9810     9568    1762              3293
1776
2        2       3   6353  8808     7684    2405              3516
7844
3        1       3  13265  1196     4221    6404               507
1788
4        2       3  22615  5410     7198    3915              1777
5185

Dataset Information:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 440 entries, 0 to 439
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Channel            440 non-null    int64
 1   Region            440 non-null    int64
 2   Fresh             440 non-null    int64
 3   Milk              440 non-null    int64
 4   Grocery           440 non-null    int64
 5   Frozen            440 non-null    int64
 6   Detergents_Paper  440 non-null    int64
 7   Delicassen        440 non-null    int64
dtypes: int64(8)
memory usage: 27.6 KB

Missing Values:
 Channel             0
Region              0
Fresh               0
Milk                0
Grocery             0
Frozen              0
Detergents_Paper    0
Delicassen          0
dtype: int64
```

Comments:

When examining the first 5 rows of the dataset, it is clear that the numerical features related to customers are well-organized. Although there is no primary key such as CustomerID, the data is ready for direct analysis. The dataset contains the following columns:

1. **Channel (Customer Type):** Contains two categories:
   - 1 represents Hotel/Restaurant/Café (Horeca) customers.
   - 2 represents Retail customers.
2. **Region:** Represents three different geographic regions:
   - 1 Lisbon
   - 2 Oporto
   - 3 Other Regions.
3. **Fresh, Milk, Grocery, Frozen, Detergents_Paper, and Delicassen:** These columns represent the annual expenditures of customers in different product categories.

Using the dataset information (data.info()), it was confirmed that all columns are of the **int64** data type and no missing values are present. This suggests that the dataset is clean at first glance.

The missing value analysis showed no missing data in the dataset. This is a significant finding as it simplifies the analysis process by allowing us to especially for algorithms like **k-means**, which rely on distance-based calculations. Since no categorical variables require transformation, we can directly proceed to scaling the data, saving both time and effort.

As a result, the dataset is well-structured for **customer segmentation** analysis. However, in the subsequent stages of this analysis, we will address challenges such as outlier detection and data scaling to prepare the dataset for clustering.

2.1. Correlation and Outlier Analysis

In this step, basic analyses were conducted to understand the general structure of the dataset and the relationships between features. Correlation analysis helps us understand the strength and direction of relationships between columns, while the boxplot analysis visualizes the data distribution and the presence of outliers.

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Display statistical summary of all columns
print("Statistical Summary:\n", data.describe())

# Show correlation between columns
print("\nCorrelation Matrix:\n", data.corr())

# Correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(data.corr(), annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()

# Boxplot for outlier analysis
plt.figure(figsize=(15, 8))
sns.boxplot(data=data)
plt.title("Outlier Analysis")
plt.xticks(rotation=45)
plt.show()
```

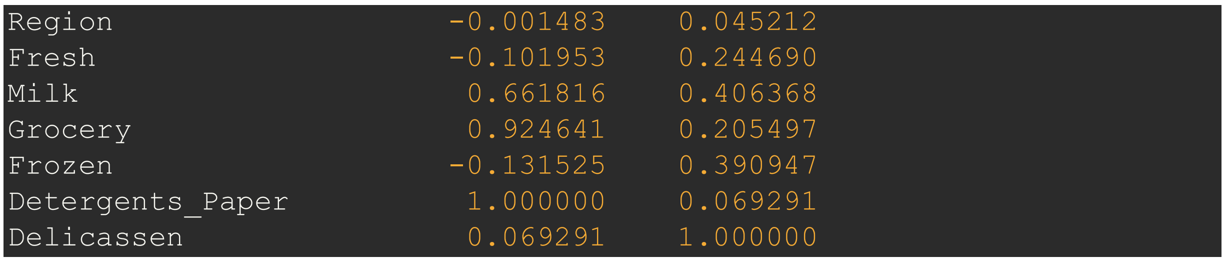
Explanation:

In this code block, the relationships between the columns and the presence of outliers in the dataset were analyzed through the following steps:

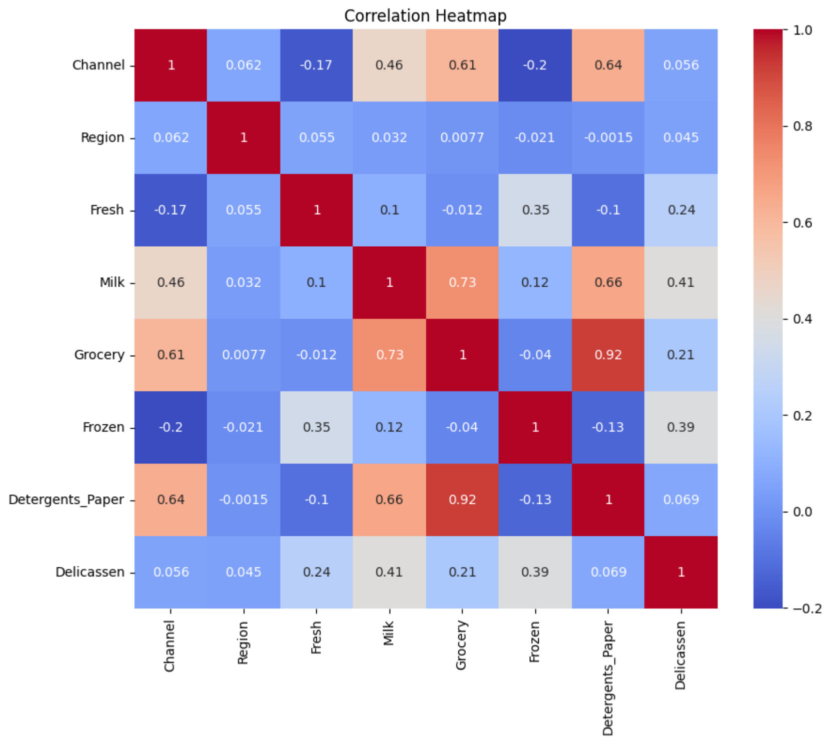
1. **Statistical Summary (describe):**
   - The basic statistical properties of each column (mean, median, minimum, maximum, 25%-75% quartiles) were printed.
2. **Correlation Matrix and Heatmap:**
   - Relationships between columns were calculated using correlation coefficients.
   - A **heatmap** was used to visualize which columns have strong or weak relationships with each other.
3. **Outlier Analysis (Boxplot):**
   - A boxplot was used to visualize the data distribution and potential outliers for each column.
   - This is the initial visual analysis step for identifying outliers.

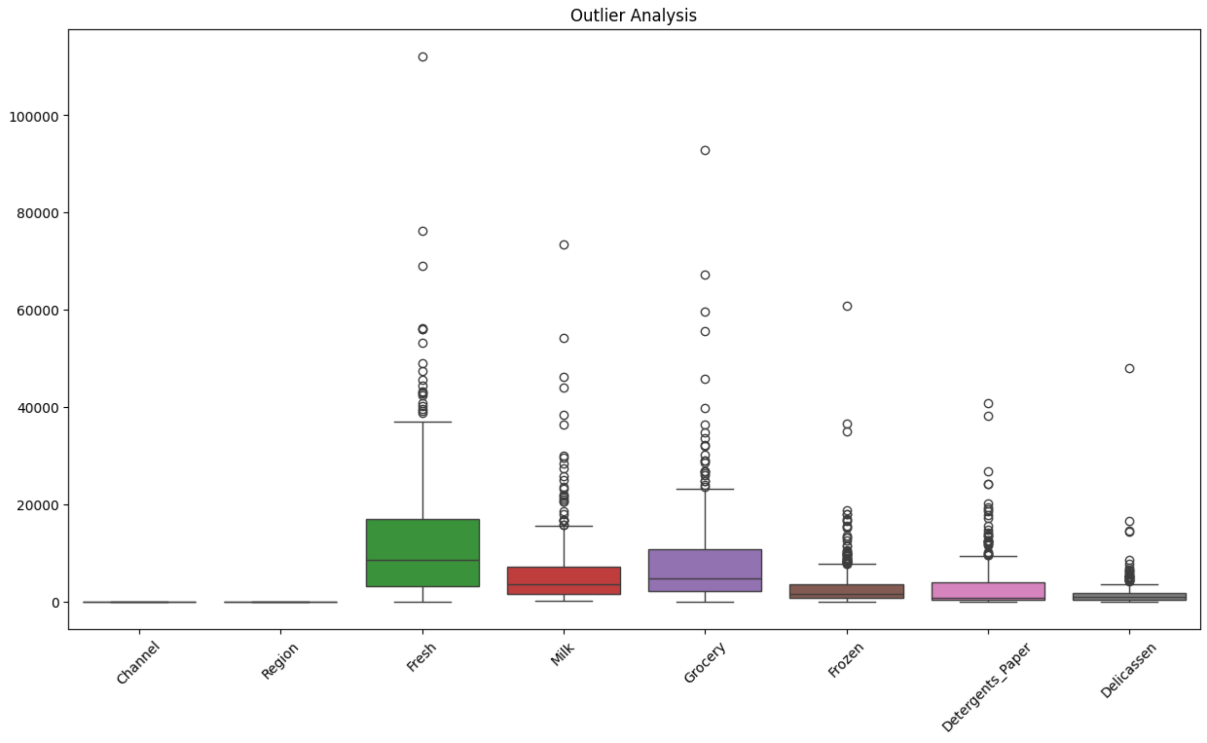
Output:

```
Statistical Summary:
           Channel      Region          Fresh          Milk
Grocery  \
count  440.000000  440.000000     440.000000    440.000000
440.000000
mean     1.322727    2.543182   12000.297727   5796.265909
7951.277273
std      0.468052    0.774272   12647.328865   7380.377175
9503.162829
min      1.000000    1.000000       3.000000     55.000000
3.000000
25%      1.000000    2.000000    3127.750000   1533.000000
2153.000000
50%      1.000000    3.000000    8504.000000   3627.000000
4755.500000
75%      2.000000    3.000000   16933.750000   7190.250000
10655.750000
max      2.000000    3.000000  112151.000000  73498.000000
92780.000000

             Frozen  Detergents_Paper    Delicassen
count    440.000000        440.000000    440.000000
mean    3071.931818       2881.493182   1524.870455
std     4854.673333       4767.854448   2820.105937
min       25.000000          3.000000      3.000000
25%      742.250000        256.750000    408.250000
50%     1526.000000        816.500000    965.500000
75%     3554.250000       3922.000000   1820.250000
max    60869.000000      40827.000000  47943.000000

Correlation Matrix:
                   Channel    Region     Fresh      Milk   Grocery
Frozen  \
Channel           1.000000  0.062028 -0.169172  0.460720  0.608792 -
0.202046
Region            0.062028  1.000000  0.055287  0.032288  0.007696 -
0.021044
Fresh            -0.169172  0.055287  1.000000  0.100510 -0.011854
0.345881
Milk              0.460720  0.032288  0.100510  1.000000  0.728335
0.123994
Grocery           0.608792  0.007696 -0.011854  0.728335  1.000000 -
0.040193
Frozen           -0.202046 -0.021044  0.345881  0.123994 -0.040193
1.000000
Detergents_Paper  0.636026 -0.001483 -0.101953  0.661816  0.924641 -
0.131525
Delicassen        0.056011  0.045212  0.244690  0.406368  0.205497
0.390947

                  Detergents_Paper  Delicassen
Channel                   0.636026    0.056011
```



```
Region              -0.001483    0.045212
Fresh               -0.101953    0.244690
Milk                 0.661816    0.406368
Grocery              0.924641    0.205497
Frozen              -0.131525    0.390947
Detergents_Paper     1.000000    0.069291
Delicassen           0.069291    1.000000
```





Comments:

- There is a significant difference between maximum and minimum values in the output. For example, in the **Fresh** column, the maximum value is **112151**, while the median is only **8504**. This indicates a right-skewed distribution and the potential presence of outliers.
- A very strong positive correlation (**0.92**) was identified between **Grocery** and **Detergents_Paper**.
- There is also a strong relationship (**0.73**) between **Milk** and **Grocery**.
- Generally, low correlations were observed between **Frozen** and other columns, indicating that Frozen is more independent.
- The boxplot results clearly show a significant number of outliers in columns like **Fresh**, **Grocery**, and **Frozen**. These outliers must be removed to prevent issues during the data analysis process.

The correlation analysis helped us understand the relationships between features. Features with strong correlations (e.g., Grocery and Detergents_Paper) can be considered together during clustering. Additionally, the boxplot analysis confirmed the presence of extreme values in some columns. These outliers will be cleaned in the next step to enhance the performance of the clustering algorithm.

2.2. Detection and Cleaning of Outliers

In data analysis and machine learning processes, outliers are data points that significantly deviate from the general distribution of the dataset. These values can pose a serious problem, especially for distance-based algorithms like K-means. Outliers can cause cluster centers to be incorrectly calculated, reducing the accuracy and reliability of the analysis results.

In this step, the **IQR method (Interquartile Range)** was used to detect outliers. The IQR represents the range between the 1st quartile (Q1) and the 3rd quartile (Q3) of a dataset. Data points outside the IQR boundaries are considered outliers and are removed. The goal of this process is to make the dataset more homogeneous, allowing the clustering algorithm to perform better.

```
from sklearn.preprocessing import StandardScaler

# Outlier analysis using IQR method
Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)
IQR = Q3 - Q1

# Filtering out outliers
filtered_data = data[~((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5
* IQR))).any(axis=1)]
print(f"Data size after removing outliers: {filtered_data.shape}")

# Data scaling
scaler = StandardScaler()
```

```
scaled_data = scaler.fit_transform(filtered_data.iloc[:, 2:])  # 
Excluding Channel and Region

# Convert scaled data back to a DataFrame
scaled_data = pd.DataFrame(scaled_data, 
columns=filtered_data.columns[2:])
print("Scaled Data:\n", scaled_data.head())
```

Output:

```
Data size after removing outliers: (332, 8)
Scaled Data:
       Fresh      Milk   Grocery    Frozen  Detergents_Paper  
Delicassen
0  0.383041  1.652898  0.334978 -0.967004          0.305053    0.390465
1 -0.305588  1.698756  0.733878 -0.059255          0.569139    0.922591
2  0.456174 -0.866283 -0.328862  2.662821         -0.619462    0.937169
3 -0.016491  1.236906 -0.148989 -0.701951         -0.069958    0.527749
4  0.316411 -0.269838  0.218508 -0.811022          0.503864   -0.572949
```

Comments:

After removing outliers, the size of the dataset was reduced from **440 to 332**.

Outliers often arise when there are significant differences in individual customer behaviors within the dataset. For instance, a customer with exceptionally high spending in the **Fresh** category might deviate significantly from the general customer group. This cleaning process ensures that the K-means algorithm will create more balanced clusters. Otherwise, outliers could have caused the algorithm to produce incorrect results.

This step is a critical part of the **preprocessing** process. Additionally, considering the high correlations between columns in the dataset, we will address these correlations in the following steps by performing **feature selection**. For example, the correlation between **Grocery** and **Detergents_Paper** (**0.92**) is quite high. This indicates that the information carried by these columns largely overlaps.

3.1. Scaling of Data (PCA)

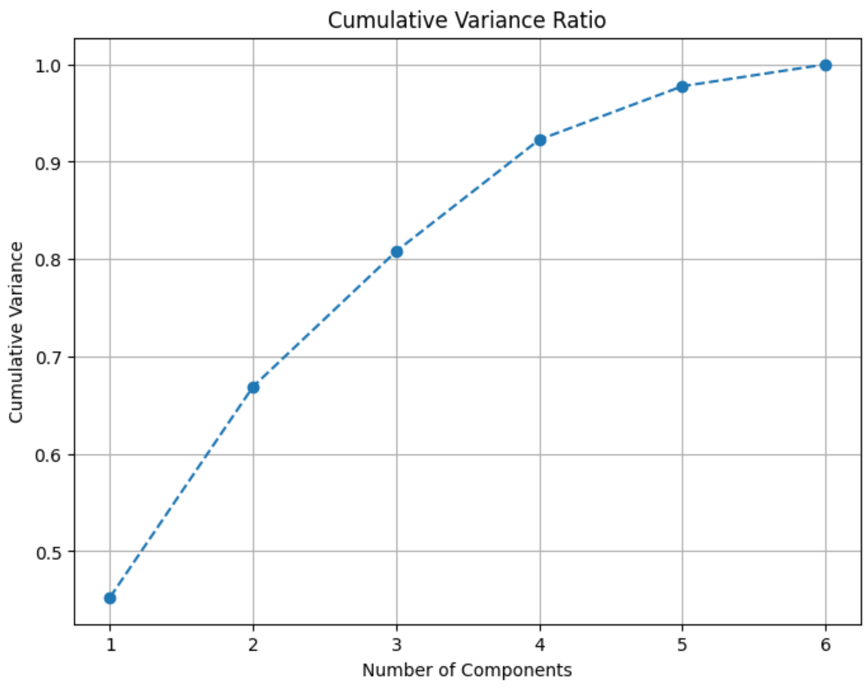
In data analysis and machine learning, having too many features can complicate the analysis process and negatively impact algorithm performance. In such cases, dimensionality reduction techniques like **PCA (Principal Component Analysis)** come into play. PCA aims to reduce the number of variables while retaining the essential information in the dataset.

During the PCA process, each new principal component explains a portion of the variance in the dataset. The goal is to select enough components to explain a large portion of the total variance. In this study, we determined the number of components that explain at least **95% of the total variance**. This method is particularly useful for eliminating redundant information carried by highly correlated features and creating a more efficient data representation.

```
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt

# Apply PCA
pca = PCA()
pca_data = pca.fit_transform(scaled_data)

# Explained variance ratios to see how much variance each component
explains
explained_variance_ratio = pca.explained_variance_ratio_
print("Explained Variance Ratios:\n", explained_variance_ratio)

# Visualize cumulative variance ratios
cumulative_variance = np.cumsum(explained_variance_ratio)
plt.figure(figsize=(8, 6))
plt.plot(range(1, len(cumulative_variance)+1), cumulative_variance,
marker='o', linestyle='--')
plt.title('Cumulative Variance Ratio')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Variance')
plt.grid()
plt.show()

# Decide the number of components to select
# For example, select the number of components explaining 95% of
variance
n_components = np.argmax(cumulative_variance >= 0.95) + 1
print(f"Number of components explaining 95% variance:
{n_components}")

# Reapply PCA with the selected number of components
pca = PCA(n_components=n_components)
pca_data_reduced = pca.fit_transform(scaled_data)

# Display the new PCA-transformed data
print("PCA-Transformed Data:\n",
pd.DataFrame(pca_data_reduced).head())
```

Output:

```
Explained Variance Ratios:
 [0.45268311 0.21638576 0.13897372 0.11463788 0.05487702 0.02244251]
Number of components explaining 95% variance: 5
PCA-Transformed Data:
           0         1         2         3         4
0  1.439818  0.206109 -0.920097 -0.207410  1.070533
1  1.877096  0.492048  0.093072  0.454126  0.845948
2 -1.322627  2.019861  1.721171  0.570074 -0.509870
3  0.776468  0.111471 -0.681615  0.329867  1.051531
4  0.239754 -0.567339 -0.608753 -0.684818 -0.460087
```



Comments:

The explained variance ratios were listed as follows: [0.45, 0.21, 0.14, 0.11, 0.05, 0.02]. In the cumulative variance graph, it was observed that 95% of the total variance could be explained with the first 5 components. After PCA transformation, the dataset is now reduced to a 5-dimensional form**.**

With PCA, we reduced the original dimensions of the dataset from 6 to 5. This eliminates unnecessary redundancy while retaining most of the critical information in the dataset.

The first component explains about 45% of the total variance, while the first two components together explain 66%. This indicates that most of the essential information in the dataset is concentrated in just a few components.

PCA effectively removes the redundant information carried by highly correlated columns, such as **Grocery** and **Detergents_Paper**. This ensures that the K-means algorithm will perform more efficiently.

The hyperparameter of PCA is the number of components selected. By choosing the number of components that explain **95% of the variance**, we ensured that the K-means algorithm operates more efficiently with fewer dimensions.

4.1. K-Means Clustering and Cluster Analysis

Now we have prepared our dataset and reduced its dimensions with PCA. At this stage, we will divide the customers in the dataset into clusters by applying the K-means algorithm. We will use methods such as the Elbow method and Silhouette score to determine the optimal number of clusters. In addition, we will analyze the created clusters and interpret the characteristics of each cluster.

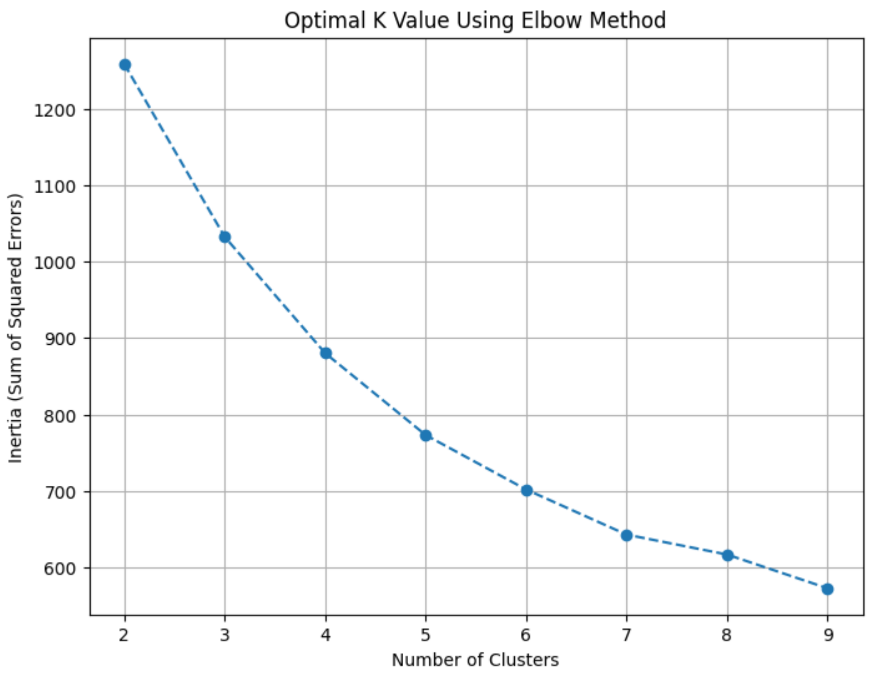
The K-means algorithm partitions data points into clusters by assigning each point to the nearest cluster center. This process ensures that clusters are more homogeneous. However, one disadvantage of the K-means algorithm is that the number of clusters (k) must be predetermined. Therefore, selecting the correct value for k is critical for the reliability of the analysis results.

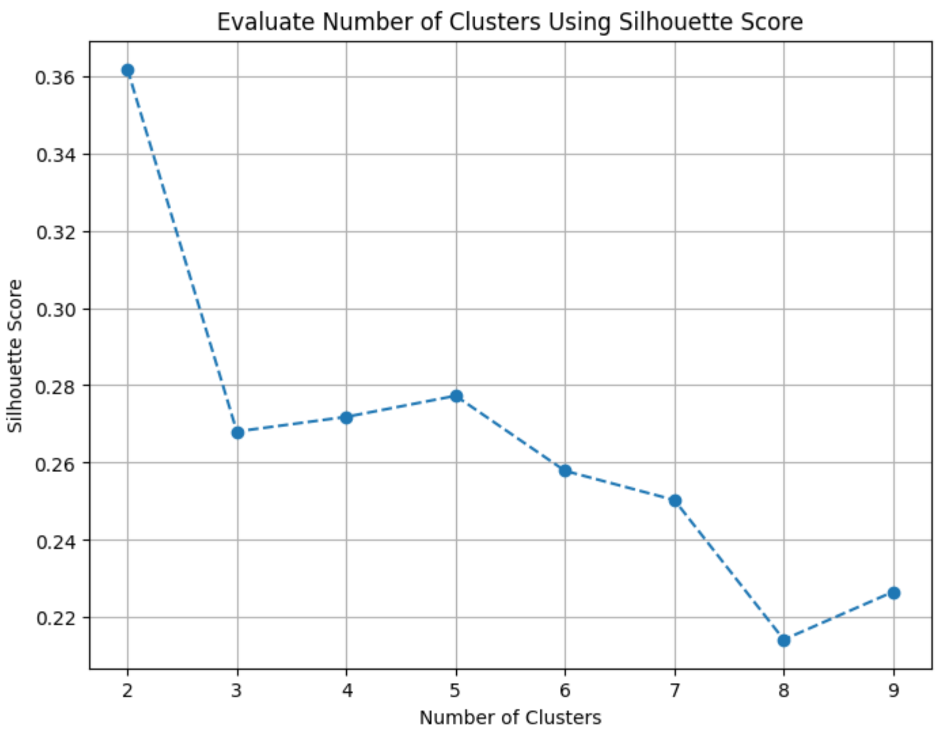
In this study, the number of clusters was determined using:

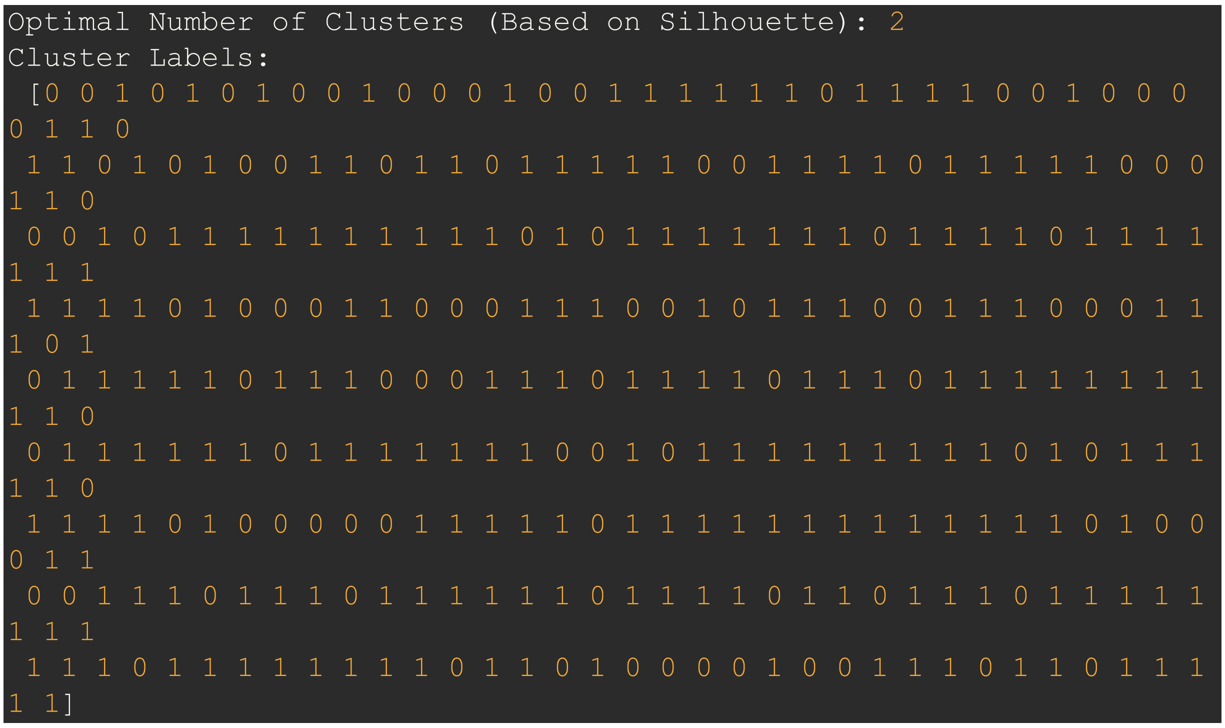
1. **Elbow Method:** By examining the reduction in the sum of squared errors (inertia) with respect to the number of clusters, we identified the point where the curve "elbows."
2. **Silhouette Score:** This metric measures intra-cluster cohesion and inter-cluster separation. A high Silhouette score indicates well-separated and internally cohesive clusters.

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# Determine the optimal number of clusters (k) using the Elbow
method
inertia = []
silhouette_scores = []
K_range = range(2, 10)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(pca_data_reduced)
    inertia.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(pca_data_reduced,
kmeans.labels_))

# Elbow method plot
plt.figure(figsize=(8, 6))
plt.plot(K_range, inertia, marker='o', linestyle='--')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia (Sum of Squared Errors)')
plt.title('Optimal K Value Using Elbow Method')
plt.grid()
plt.show()

# Silhouette score plot
plt.figure(figsize=(8, 6))
plt.plot(K_range, silhouette_scores, marker='o', linestyle='--')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.title('Evaluate Number of Clusters Using Silhouette Score')
plt.grid()
plt.show()

# Selection of optimal number of clusters (e.g., the k with the
highest silhouette score)
optimal_k = silhouette_scores.index(max(silhouette_scores)) + 2  #
K_range starts at 2
print(f"Optimal Number of Clusters (Based on Silhouette):
{optimal_k}")

# K-means model with the optimal number of clusters
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
kmeans.fit(pca_data_reduced)

# Print cluster labels
print("Cluster Labels:\n", kmeans.labels_)
```

Output:







```
Optimal Number of Clusters (Based on Silhouette): 2
Cluster Labels:
 [0 0 1 0 1 0 1 0 0 1 0 0 0 1 0 0 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 0 0
0 1 1 0
 1 1 0 1 0 1 0 0 1 1 0 1 1 0 1 1 1 1 1 0 0 1 1 1 1 0 1 1 1 1 1 0 0 0
1 1 0
 0 0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1
1 1 1
 1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 0 0 1 0 1 1 1 0 0 1 1 1 0 0 0 1 1
1 0 1
 0 1 1 1 1 1 0 1 1 1 0 0 0 1 1 1 0 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1
1 1 0
 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 1 0 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1
1 1 0
 1 1 1 1 0 1 0 0 0 0 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0
0 1 1
 0 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 0 1 1 0 1 1 1 0 1 1 1 1 1 1
1 1 1
 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 0 1 0 0 0 0 1 0 0 1 1 1 0 1 1 0 1 1 1
1 1]
```

Comments:

- **Elbow Graph:** The sum of squared errors (inertia) decreases as the number of clusters increases. The "elbow" point in the graph is observed at 2 or 3 clusters.
- **Silhouette Score:** The highest silhouette score was obtained when the number of clusters was 2.
- **Cluster Labels:** Each data point was assigned to the appropriate cluster center.

Considering both the Silhouette score and the Elbow method, we concluded that working with 2 clusters is meaningful. This suggests that there are two primary customer segments in the dataset. Each data point was assigned to its nearest cluster center, demonstrating the fundamental functionality of the K-means algorithm.

In this step, the number of clusters (k**)** is a fundamental hyperparameter of the K-means algorithm. Metrics such as the Silhouette score played a crucial role in determining the optimal k.

5.1. Analysis of Cluster Characteristics

At this stage, we will analyze the clusters created with the K-means algorithm in detail. Our goal is to understand customer segments by determining the basic features of each cluster.

The K-means algorithm assigns each data point to the most suitable cluster, forming distinct groups. However, these clusters must be analyzed in detail to provide functional insights. By calculating the statistical properties of each cluster, we can identify which features are dominant in each group.

In particular, we will examine whether there are significant differences between clusters in terms of spending habits and geographic characteristics. This analysis will help us understand which customer segments each cluster represents.

```
# Add cluster labels to the original dataset
filtered_data = data[~((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5
* IQR))).any(axis=1)].copy()
filtered_data['Cluster'] = kmeans.labels_

# Calculate the mean values of features for each cluster
cluster_summary = filtered_data.groupby('Cluster').mean()
print("Mean Values of Cluster Features:\n", cluster_summary)
```

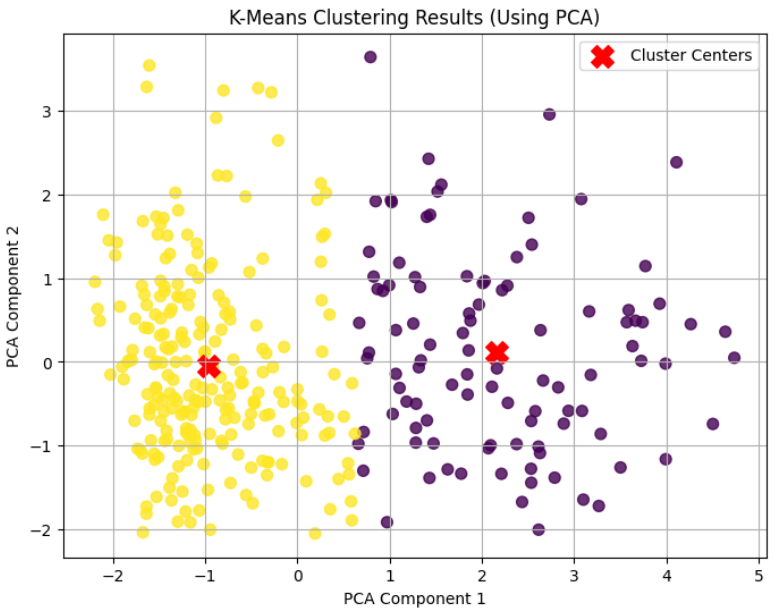
Outputs:

```
Mean Values of Cluster Features:
           Channel    Region         Fresh         Milk
Grocery  \
Cluster
0        1.794118  2.637255   7941.352941  7968.333333  11974.607843
1        1.060870  2.495652  10259.643478  2391.956522   3170.843478

              Frozen  Detergents_Paper   Delicassen
Cluster
0        1340.186275       4814.431373  1443.519608
1        2094.926087        692.643478   827.273913
```

Comments:

The statistical means of the clusters were calculated, and the dominant characteristics of each cluster were identified. Cluster 0 has higher spending on Grocery and Detergents_Paper**.** Cluster 1 is characterized by lower spending on Milk but higher spending in the Fresh category. Cluster 0 (Grocery-Oriented Customers): This cluster likely represents retail customers. Higher spending on grocery and cleaning products suggests individual consumers. Cluster 1 (Fresh-Product-Oriented Customers): This cluster is likely part of the Horeca (hotel/restaurant/café) segment. A strong demand for fresh products indicates a business-oriented group. This analysis helped us better understand the customer groups. Now let's make this information more concrete by visualizing it.

6. Results and Comments



In this study, we performed customer segmentation using the K-means algorithm. We applied the stages such as data preprocessing, dimensionality reduction and clustering in detail. As a result, we divided the customers into two different clusters and analyzed the basic features of these clusters. These results provide meaningful information to understand customer behavior and develop different strategies.

Cluster 0 (Individual Consumers): Customers in this group focus on daily needs, such as groceries and cleaning products.

Cluster 1 (Commercial Consumers): Customers in this group spend more on fresh products, indicating a focus on the Horeca (hotel/restaurant/cafe) sector.

PCA effectively reduced redundant information in the dataset and improved the efficiency of the K-means algorithm. Particularly, the influence of highly correlated columns was minimized.

Removing outliers ensured more homogeneous clusters and improved the accuracy of the analysis results.

Scaling the data allowed the algorithm to treat all features with equal importance.

Cluster 0: Special campaigns can be organized for retail sales and daily need products.

Cluster 1: Fresh product supply chain solutions can be developed for the Horeca sector

This study demonstrated how the K-means algorithm can be applied for customer segmentation. Careful execution of data preprocessing, dimensionality reduction, and hyperparameter tuning significantly enhanced the algorithm's performance. Two main customer segments were identified, providing opportunities to develop business strategies tailored to these groups.

Question 3

1. Introduction

1.1. Project Objective

Regression analysis is a supervised learning method used to predict a continuous target variable based on explanatory variables. In this study, we aimed to predict house prices per unit area using linear regression and other advanced regression techniques. By comparing the performance of different regression models, we sought to understand which approach provides better accuracy and reliability for this dataset.

The dataset used in this study is the **Real Estate Valuation Dataset**, which was obtained from the UCI Machine Learning Repository. It contains information such as transaction date, house age, distance to the nearest MRT station, number of convenience stores, and geographic coordinates. These features offer valuable insights for building regression models to predict house prices.

The steps followed in this study are as follows:

1. **Data Preprocessing:** Missing value analysis was conducted, outliers were cleaned, and all features were scaled for consistency.
2. **Feature Selection:** The importance of each feature was analyzed to ensure that only meaningful features were included in the model.
3. **Model Training and Evaluation:** Linear regression was trained and compared with advanced techniques like Ridge Regression, Lasso Regression, and Random Forest Regression.
4. **Hyperparameter Optimization:** For advanced models, key hyperparameters were tuned to improve their performance.
5. **Performance Comparison:** The models were evaluated using metrics such as Mean Squared Error (MSE) and R-squared (R^2), and their results were interpreted.

The results of this study can provide actionable insights for real estate valuation and decision-making processes. Each step is explained in detail, with supporting code, outputs, and interpretations included in this report.

Dataset and Preprocessing

In this study, we used the Real Estate Valuation Dataset, which consists of 414 data points with 8 features. The target variable, **house_price_per_unit_area**, represents house prices per unit area. The features in the dataset are as follows:

1. **transaction_date**: The transaction date, represented as a float number indicating the year and month of sale.
2. **house_age**: The age of the house in years.
3. **distance_to_MRT**: The distance to the nearest MRT station.
4. **number_of_convenience_stores**: The number of convenience stores in the vicinity.
5. **latitude** and **longitude**: The geographical coordinates of the property.
6. **house_price_per_unit_area**: The target variable representing the house price per unit area.

1.1.2. Project Process

The project process followed in this study consists of the following steps:

1. **Dataset Introduction and Examination:**
    - The structure of the dataset was analyzed to design an appropriate regression process.
    - In this step, columns, data types, missing values, and general statistics of the dataset were examined to identify potential issues or requirements.
2. **Data Preprocessing:**
    - Missing value analysis was performed, and it was confirmed that there were no missing values in the dataset.
    - Outliers were detected and cleaned using the Interquartile Range (IQR) method, focusing on the distance_to_MRT and house_price_per_unit_area features.
    - All features were scaled using **StandardScaler** to ensure consistency in the data and to prepare it for regression analysis.
3. **Feature Selection:**
    - Irrelevant columns (e.g., No) were dropped to simplify the dataset.
    - Features were examined to evaluate their relevance to the target variable. Correlation analysis was performed to identify relationships between features.
4. **Model Training and Evaluation:**
    - The dataset was split into training (80%) and testing (20%) subsets.
    - A Linear Regression model was trained and evaluated using metrics such as Mean Squared Error (MSE) and R-squared (R^2).
    - Advanced models, including Ridge Regression, Lasso Regression, and Random Forest Regression, were applied to compare performance.
    - Hyperparameters (e.g., alpha in Ridge and Lasso, n_estimators in Random Forest) were tuned to optimize performance.
5. **Performance Comparison:**
    - The performance of all regression models was compared using test data.
    - Results were visualized, and the strengths and weaknesses of each model were analyzed.
6. **Results and Comments:**
    - The findings of the study were evaluated in terms of real estate price prediction.
    - Insights into the dataset and the models' performance were drawn, highlighting the most effective regression techniques for the given data.

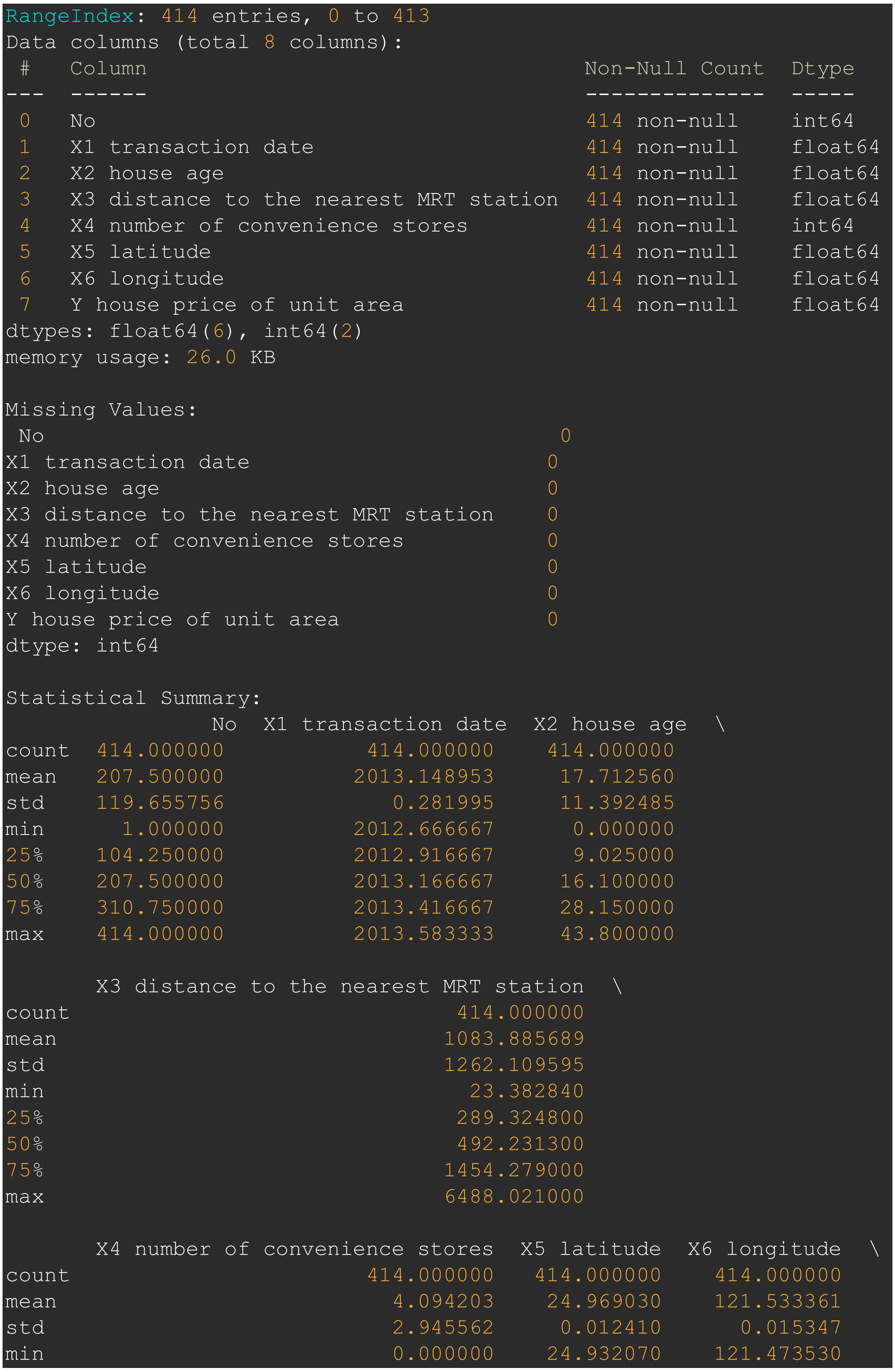
1.1.3 Dataset Introduction and Examination

In this step, the dataset was loaded, and basic analyses were conducted. The structure, columns, data types, and missing values were examined to design an appropriate preprocessing and regression process.

```
import pandas as pd

# Load the dataset
data = pd.read_excel("Real estate valuation data set.xlsx")  #
Replace the file name with your actual file

# Display the first 5 rows
print("First 5 Rows:\n", data.head())

# Display dataset information
print("\nDataset Information:\n")
data.info()

# Check for missing values
print("\nMissing Values:\n", data.isnull().sum())

# Statistical summary
print("\nStatistical Summary:\n", data.describe())
```

Output:

```
First 5 Rows:
    No  X1 transaction date  X2 house age  \
0   1          2012.916667          32.0
1   2          2012.916667          19.5
2   3          2013.583333          13.3
3   4          2013.500000          13.3
4   5          2012.833333           5.0

   X3 distance to the nearest MRT station  X4 number of convenience
stores  \
0                                 84.87882
10
1                                306.59470
9
2                                561.98450
5
3                                561.98450
5
4                                390.56840
5

   X5 latitude  X6 longitude  Y house price of unit area
0     24.98298     121.54024                        37.9
1     24.98034     121.53951                        42.2
2     24.98746     121.54391                        47.3
3     24.98746     121.54391                        54.8
4     24.97937     121.54245                        43.1

Dataset Information:

<class 'pandas.core.frame.DataFrame'>
```



```
RangeIndex: 414 entries, 0 to 413
Data columns (total 8 columns):
 #   Column                                  Non-Null Count  Dtype  
---  ------                                  --------------  -----  
 0   No                                      414 non-null    int64  
 1   X1 transaction date                     414 non-null    float64
 2   X2 house age                            414 non-null    float64
 3   X3 distance to the nearest MRT station  414 non-null    float64
 4   X4 number of convenience stores         414 non-null    int64  
 5   X5 latitude                             414 non-null    float64
 6   X6 longitude                            414 non-null    float64
 7   Y house price of unit area              414 non-null    float64
dtypes: float64(6), int64(2)
memory usage: 26.0 KB

Missing Values:
 No                                        0
X1 transaction date                       0
X2 house age                              0
X3 distance to the nearest MRT station    0
X4 number of convenience stores           0
X5 latitude                               0
X6 longitude                              0
Y house price of unit area                0
dtype: int64

Statistical Summary:
                No  X1 transaction date  X2 house age  \
count  414.000000           414.000000    414.000000   
mean   207.500000          2013.148953     17.712560   
std    119.655756             0.281995     11.392485   
min      1.000000          2012.666667      0.000000   
25%    104.250000          2012.916667      9.025000   
50%    207.500000          2013.166667     16.100000   
75%    310.750000          2013.416667     28.150000   
max    414.000000          2013.583333     43.800000   

       X3 distance to the nearest MRT station  \
count                              414.000000   
mean                              1083.885689   
std                               1262.109595   
min                                 23.382840   
25%                                289.324800   
50%                                492.231300   
75%                               1454.279000   
max                               6488.021000   

       X4 number of convenience stores  X5 latitude  X6 longitude  \
count                       414.000000   414.000000    414.000000   
mean                          4.094203    24.969030    121.533361   
std                           2.945562     0.012410      0.015347   
min                           0.000000    24.932070    121.473530   
```

```
25%                          1.000000     24.963000     121.528085
50%                          4.000000     24.971100     121.538630
75%                          6.000000     24.977455     121.543305
max                         10.000000     25.014590     121.566270

       Y house price of unit area
count                   414.000000
mean                     37.980193
std                      13.606488
min                       7.600000
25%                      27.700000
50%                      38.450000
75%                      46.600000
max                     117.500000
```

Comments:

Dataset Overview:

- The dataset contains 414 entries with 8 columns, representing various characteristics of real estate properties and their respective prices per unit area. The columns are:
    1. **No**: Row number, which does not carry significant analytical meaning and will be excluded in subsequent analyses.
    2. **X1 transaction date**: The transaction date as a floating-point number.
    3. **X2 house age**: The age of the house in years.
    4. **X3 distance to the nearest MRT station**: The distance to the closest metro station, measured in meters.
    5. **X4 number of convenience stores**: The number of convenience stores within a certain radius of the property.
    6. **X5 latitude** and **X6 longitude**: The geographic coordinates of the property.
    7. **Y house price of unit area**: The target variable representing house price per unit area.

The isnull() function confirmed that there are no missing values in any column. This simplifies the preprocessing phase and eliminates the need for imputation strategies. The dataset is well-structured for regression analysis. However, preprocessing steps such as outlier detection and feature scaling are necessary to ensure the model's accuracy and stability. This dataset is clean and ready for preprocessing, with no missing values or categorical transformations needed. The subsequent steps will address outlier detection and scaling to further refine the data for regression modeling.

Renaming Columns:

To enhance the clarity and readability of the dataset, the column names were renamed. The new names are more descriptive and directly represent the information contained in each column. This step simplifies subsequent analyses and ensures better understanding when interpreting the dataset.

```python
# Rename columns
data.rename(columns={
    "X1 transaction date": "transaction_date",
    "X2 house age": "house_age",
    "X3 distance to the nearest MRT station": "distance_to_MRT",
    "X4 number of convenience stores":
"number_of_convenience_stores",
    "X5 latitude": "latitude",
    "X6 longitude": "longitude",
    "Y house price of unit area": "house_price_per_unit_area"
}, inplace=True)

# Check updated column names
print("Updated Column Names:\n", data.columns)
```

Output:

```
Updated Column Names:
 Index(['No', 'transaction_date', 'house_age', 'distance_to_MRT',
       'number_of_convenience_stores', 'latitude', 'longitude',
       'house_price_per_unit_area'],
      dtype='object')
```

Comments:

The column names were updated as follows:

- X1 transaction date → transaction_date
- X2 house age → house_age
- X3 distance to the nearest MRT station → distance_to_MRT
- X4 number of convenience stores → number_of_convenience_stores
- X5 latitude → latitude
- X6 longitude → longitude
- Y house price of unit area → house_price_per_unit_area

This renaming step ensured that the column names are intuitive and self-explanatory, improving both the dataset's readability and the analysis workflow.

2. Data Preprocessing

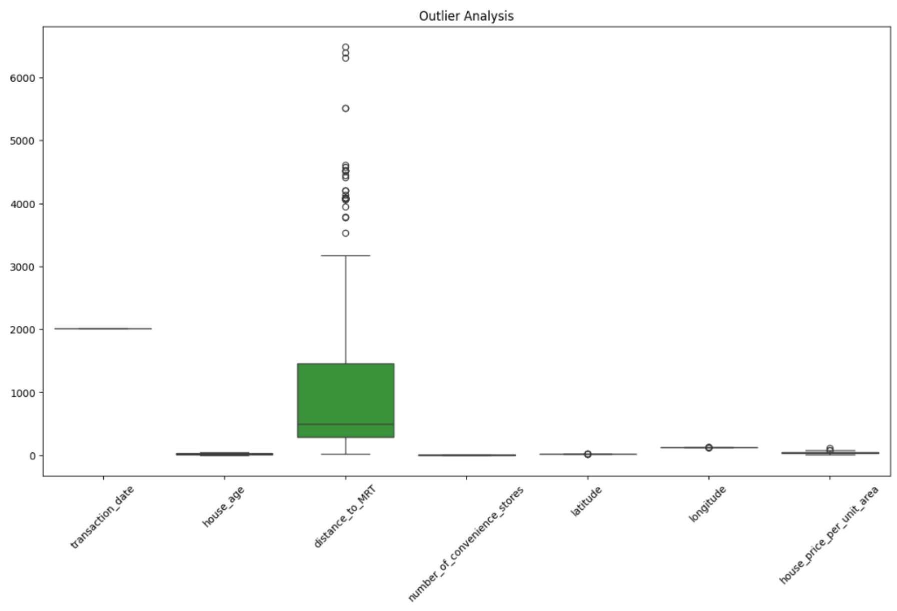
Data preprocessing is a critical step in any data analysis or machine learning project. It ensures that the dataset is clean, consistent, and ready for modeling. The following preprocessing steps were applied in this study:

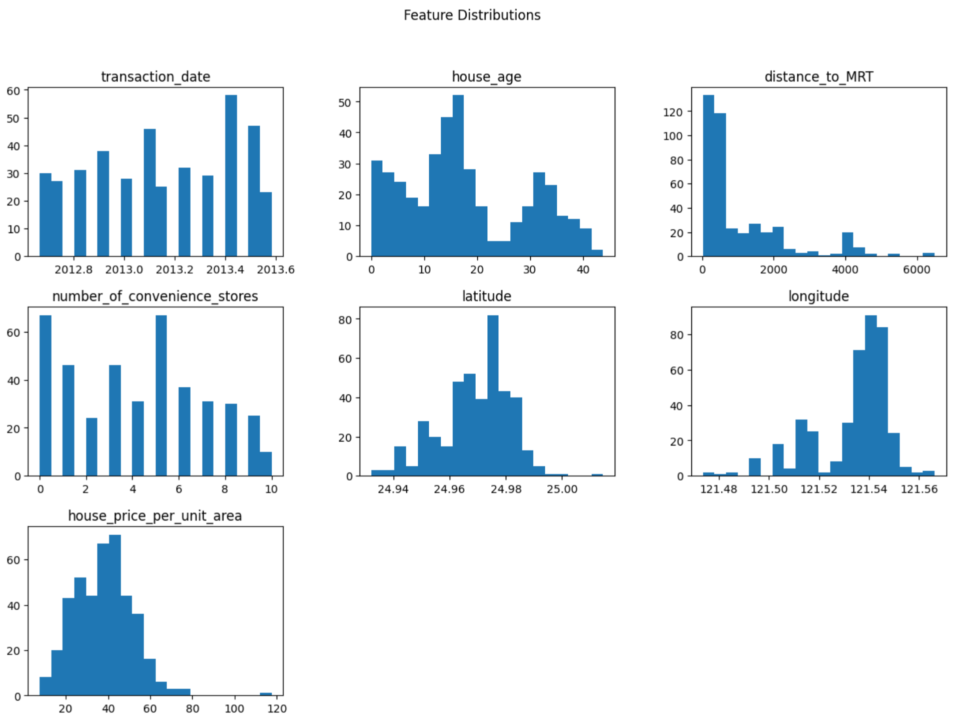
2.1. Outlier Detection and Removal

To ensure the dataset is prepared for regression analysis, exploratory data analysis (EDA) was conducted to identify outliers and understand the distribution of features. This step included two key visualizations: a boxplot for outlier analysis and histograms to explore the distribution of features.

```
import matplotlib.pyplot as plt
import seaborn as sns

# The "No" column is not meaningful for analysis, so we drop it
data_analysis = data.drop(columns=["No"])

# Outlier analysis using a boxplot
plt.figure(figsize=(15, 8))
sns.boxplot(data=data_analysis)
plt.title("Outlier Analysis")
plt.xticks(rotation=45)
plt.show()

# Plot histograms for feature distributions
data_analysis.hist(bins=20, figsize=(15, 10), grid=False)
plt.suptitle("Feature Distributions")
plt.show()
```

Output:





Comments:

A boxplot was created for each feature to visualize potential outliers. The feature distance_to_MRT exhibited significant outliers, as seen from its extended whiskers and extreme points. Other features such as house_price_per_unit_area also displayed a few outliers but were less prominent compared to distance_to_MRT.

Histograms for all features were plotted to analyze their distributions. The feature distance_to_MRT was right-skewed, indicating that most properties are located closer to MRT stations, with only a few properties at farther distances. house_price_per_unit_area showed a nearly normal distribution, making it well-suited for regression analysis. Features such as number_of_convenience_stores and house_age displayed discrete and uniform distributions, respectively.

2.2. Outlier Removal

Outliers in the dataset can distort regression models by exaggerating the relationships between features and the target variable. To address this issue, the Interquartile Range (IQR) method was applied to remove outliers from critical features such as distance_to_MRT and house_price_per_unit_area.

Steps:

1. **Identify Outliers:**
    - For each feature, the first quartile (Q1), third quartile (Q3), and the interquartile range (IQR) were calculated.

- Outliers were identified as values below Q1−1.5×IQRQ1 - 1.5 \times IQRQ1−1.5×IQR or above Q3+1.5×IQRQ3 + 1.5 \times IQRQ3+1.5×IQR.

2. **Remove Outliers:**
   - Rows containing outliers in distance_to_MRT and house_price_per_unit_area were removed to ensure the dataset's consistency.

```
# Remove outliers using the IQR method
def remove_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return df[(df[column] >= lower_bound) & (df[column] <=
upper_bound)]

# Remove outliers from the distance_to_MRT column
data_cleaned = remove_outliers(data, "distance_to_MRT")

# Remove outliers from the house_price_per_unit_area column
data_cleaned = remove_outliers(data_cleaned,
"house_price_per_unit_area")

# Check the size of the cleaned dataset
print("Dataset size after removing outliers:", data_cleaned.shape)
```

Output:

```
Dataset size after removing outliers: (373, 8)
```

Comments:

After applying the IQR method, the dataset was reduced from 414 entries to 373 entries. This reduction indicates that approximately 10% of the data contained outliers in the two targeted features. Removing these extreme values ensures that the regression models will focus on the general patterns in the dataset without being biased by anomalies.

3. Feature Scaling

Feature scaling is a critical preprocessing step in regression analysis, particularly when the dataset contains features with varying scales. Features on different scales can disproportionately influence the performance of regression models. To address this, all numerical features were standardized using the **StandardScaler** method.

Steps:

1. Remove irrelevant columns such as No, which is not meaningful for analysis.
2. Apply the StandardScaler to transform all numerical features. Scale features to have a mean of 0 and a standard deviation of 1.
3. Store the scaled data in a new DataFrame for subsequent modeling steps.

```
from sklearn.preprocessing import StandardScaler

# Dropping the "No" column as it is not meaningful for analysis
data_for_scaling = data_cleaned.drop(columns=["No"])

# Standardization
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data_for_scaling)

# Save the scaled data as a DataFrame
scaled_data_df = pd.DataFrame(scaled_data,
columns=data_for_scaling.columns)
print("Scaled Data:\n", scaled_data_df.head())
```

Output:

```
Scaled Data:
    transaction_date  house_age  distance_to_MRT
number_of_convenience_stores  \
0         -0.810636   1.244653        -0.959923
1.983040
1         -0.810636   0.174269        -0.645261
1.625537
2          1.579619  -0.356642        -0.282808
0.195525
3          1.280837  -0.356642        -0.282808
0.195525
4         -1.109418  -1.067377        -0.526084
0.195525

   latitude  longitude  house_price_per_unit_area
0  1.184376   0.319486                  -0.127300
1  0.918977   0.254254                   0.245421
2  1.634750   0.647431                   0.687486
3  1.634750   0.647431                   1.337581
4  0.821463   0.516968                   0.323432
```

Comments:

StandardScaler transformed all numerical features to a standardized scale, where each feature has a mean of 0 and a standard deviation of 1. This ensures that no single feature dominates the regression models due to its scale. The scaled features, such as distance_to_MRT and house_price_per_unit_area, are now ready for regression modeling.

4.Model Training and Evaluation

In this step, regression models were applied to predict the target variable, house_price_per_unit_area. To evaluate the models effectively, the dataset was split into training and testing sets, and multiple regression models were trained and compared. The following steps were performed; In this step, the scaled dataset was divided into training and testing sets to evaluate the regression models effectively. The training set, comprising 80% of the data, was used to train the models, while the remaining 20% was reserved for testing.

```
from sklearn.model_selection import train_test_split

# Separating target variable and independent variables
X = scaled_data_df.drop(columns=["house_price_per_unit_area"])  # 
Independent variables
y = scaled_data_df["house_price_per_unit_area"]  # Target variable

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

# Check dataset sizes
print("Training data size:", X_train.shape)
print("Test data size:", X_test.shape)
```

Output:

```
Training data size: (298, 6)
Test data size: (75, 6)
```

4.1. Linear Regression

The first model applied was a simple linear regression. This served as a baseline for comparing more advanced regression models.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Define and train the Linear Regression model
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = lr_model.predict(X_test)

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

# Print the results
```

```python
print("Linear Regression Performance Metrics:")
print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"R-squared (R²): {r2:.4f}")
```

Output:

```
Linear Regression Performance Metrics:
Mean Squared Error (MSE): 0.3859
R-squared (R²): 0.6543
```

Comments:

The linear regression model achieved an R-squared value of **0.6543**, indicating that approximately 65% of the variance in house_price_per_unit_area is explained by the model. The MSE was **0.3859**, reflecting the average squared error in the predictions.

4.2. Ridge Regression

Ridge regression is an advanced linear regression technique that applies L2 regularization to minimize overfitting by penalizing large coefficients. This method is particularly useful when multicollinearity exists in the dataset.

```python
from sklearn.linear_model import Ridge

# Define the Ridge Regression model
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train, y_train)

# Make predictions on the test data
y_pred_ridge = ridge_model.predict(X_test)

# Performance metrics
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)

# Print the results
print("Ridge Regression Performance Metrics:")
print(f"Mean Squared Error (MSE): {mse_ridge:.4f}")
print(f"R-squared (R²): {r2_ridge:.4f}")
```

Output:

```
Ridge Regression Performance Metrics:
Mean Squared Error (MSE): 0.3865
R-squared (R²): 0.6537
```

Comments:

The Ridge regression model achieved an R-squared value of **0.6537**, which is slightly lower than the baseline linear regression model. The Mean Squared Error (MSE) was **0.3865**, very close to that of the linear regression model. These results suggest that Ridge regression did not significantly outperform the baseline model, likely because the dataset already exhibits a relatively simple linear structure with minimal multicollinearity.

4.3. Lasso Regression

Lasso regression is another advanced linear regression technique that applies L1 regularization. Unlike Ridge regression, Lasso can shrink some coefficients to zero, effectively performing feature selection. This makes it particularly useful when some features have minimal contribution to the target variable.

```
from sklearn.linear_model import Lasso

# Define the Lasso Regression model
lasso_model = Lasso(alpha=0.1)  # Alpha value can be optimized later
lasso_model.fit(X_train, y_train)

# Make predictions on the test data
y_pred_lasso = lasso_model.predict(X_test)

# Performance metrics
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
r2_lasso = r2_score(y_test, y_pred_lasso)

# Print the results
print("Lasso Regression Performance Metrics:")
print(f"Mean Squared Error (MSE): {mse_lasso:.4f}")
print(f"R-squared (R²): {r2_lasso:.4f}")
```

Output:

```
Lasso Regression Performance Metrics:
Mean Squared Error (MSE): 0.4996
R-squared (R²): 0.5524
```

Comments:

The Lasso regression model achieved an R-squared value of **0.5524**, which is lower than both the linear and Ridge regression models. The Mean Squared Error (MSE) was **0.4996**, indicating less accurate predictions compared to the previous models. Lasso's feature selection capability did not provide a significant advantage in this case, suggesting that all features contribute meaningfully to the target variable. However, it is worth noting that the regularization strength (alpha) can be tuned further to improve performance.

4.4. Random Forest Regression

Random Forest Regression is an advanced ensemble learning method that uses multiple decision trees to make predictions. Unlike linear models, Random Forest can capture nonlinear relationships in the dataset and handle interactions between features effectively.

```
from sklearn.ensemble import RandomForestRegressor

# Define the Random Forest Regression model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Make predictions on the test data
y_pred_rf = rf_model.predict(X_test)

# Performance metrics
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

# Print the results
print("Random Forest Regression Performance Metrics:")
print(f"Mean Squared Error (MSE): {mse_rf:.4f}")
print(f"R-squared (R²): {r2_rf:.4f}")
```

Output:

```
Random Forest Regression Performance Metrics:
Mean Squared Error (MSE): 0.2518
R-squared (R²): 0.7744
```

Comments:

The Random Forest Regression model outperformed all the previous models, achieving an R-squared value of **0.7744**, indicating that approximately 77% of the variance in the target variable is explained by the model. The Mean Squared Error (MSE) was **0.2518**, the lowest among all models tested. These results highlight the model's ability to capture complex relationships in the dataset and make accurate predictions.

5. Model Comparison

In this step, the performances of all the models applied in the study (Linear Regression, Ridge Regression, Lasso Regression, and Random Forest Regression) were compared using two key evaluation metrics:

- **Mean Squared Error (MSE):** Represents the average squared difference between predicted and actual values. Lower values indicate better performance.
- **R-squared (R^2):** Indicates the proportion of variance in the target variable explained by the model. Higher values indicate better performance.

Summary of Model Performances:

Model	Mean Squared Error (MSE)	R-squared (R^2)
Linear Regression	0.3859	0.6543
Ridge Regression	0.3865	0.6537
Lasso Regression	0.4996	0.5524
Random Forest Regression	0.2518	0.7744

Analysis:

1. **Linear Regression:** This baseline model achieved an MSE of 0.3859 and an R^2 value of 0.6543, demonstrating moderate performance in predicting the target variable.
2. **Ridge Regression:** Adding L2 regularization to the linear model did not significantly improve performance, with similar MSE and R^2 values to the baseline model. This suggests minimal multicollinearity in the dataset.
3. **Lasso Regression:** Lasso performed worse than both Linear and Ridge Regression, with an MSE of 0.4996 and an R^2 of 0.5524, indicating that all features are likely important for the target variable, and feature selection was not beneficial.
4. **Random Forest Regression:** This nonlinear model significantly outperformed all linear models, achieving the lowest MSE (0.2518) and the highest R^2 (0.7744). These results highlight the presence of nonlinear relationships in the dataset that linear models could not capture effectively.

The results demonstrate that Random Forest Regression is the most suitable model for predicting real estate prices in this dataset. Its flexibility in capturing nonlinear patterns made it the most effective approach, achieving the highest R^2 and lowest MSE. Linear models like Ridge and Lasso regression were less effective due to their inability to model complex relationships.