# QUESTION-1 REPORT

**Name:** Doğukan Erzurum

**Number:** 201401023

**Subject:** Analyzing and training Decision Tree and Random Forest models on Diabetes data.

**Datasets:**

- **Name:** Diabets
- **Source:** Kaggle
- **Link:** https://www.kaggle.com/datasets/akshaydattatraykhare/diabetes-dataset/data

**Methods Used:** Data Exporation, Data Cleaning, Feature Analysis, Feature Engineering, Feature Scaling, Data Splitting, Modeling, Visualizition

## 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 which include Accuracy, Precision, Recall, and F1-Score,
    - Analyzing consequences via a Confusion Matrix.

4. **Model Comparison and Interpretation:**
    - Models compared and results interpreted.

## 1.2. Data Exploration and Processing

In this section, we are able to observe the dataset in detail, deal with lacking and outlier values, carry out characteristic selection, and observe statistics preprocessing steps. Our aim is to put together the dataset for modeling and take the essential steps to beautify version performance.

### 1.2.1. Loading and Initial Examination of the Dataset

First, we load the dataset the usage of the pandas library and assessment its simple information. This includes:

```
import pandas as pd

data = pd.read_csv('diabetes.csv')

print("Dataset General Information:")
print(data.info())

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 includes a complete of 768 observations and 9 columns.
- The Outcome column is described because the goal variable, indicating the diabetes popularity of individuals. This variable is classified as:
    - 0: Not diabetic,
    - 1: Diabetic,
- Beginning exam found out no missing values withinside the dataset. Be that as it may, it ended up specified that zero values in a couple of columns may constitute missing information. This trouble has been analyzed as an detail requiring so also investigation.

### 1.2.2. Analysis of Missing Values

Within the dataset, certain columns contain values, which are not naturally conceivable and demonstrate lost information. Tending to these lost values through fitting methods is fundamental for moving forward the model's execution and unwavering quality.

```
print("\nMissing Values Check:")
print(data.isnull().sum())

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

**Output:**

```
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:**

- The lost esteem investigation uncovered no expressly lost sections within the dataset. Be that as it may, a closer see at the factual rundown appeared that the least and 25th percentile values for certain columns are 0. Particularly, the Glucose, BloodPressure, SkinThickness, Affront, and BMI columns incorporate values, which are not organically conceivable and ought to be treated as lost information.
- This highlights the significance of tending to lost information utilizing fitting procedures, because it plays a pivotal part in guaranteeing the precision of the modeling prepare. Earlier to ascription, a careful examination of information

disseminations is fundamental to determine and apply the foremost appropriate ascription strategies.

### 1.2.3. Marking and Filling Missing Values

To suitably address the lost information, the values within the Glucose, BloodPressure, SkinThickness, Affront, and BMI columns were treated as lost and supplanted with NaN. After performing this operation, the number of lost values was calculated as takes after:

```
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)}")

data[columns_with_zeros] = data[columns_with_zeros].replace(0,
pd.NA)

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 investigation recognized a strikingly tall number of lost values within the Affront and SkinThickness columns. To handle this, the lost values were ascribed utilizing the middle values of the individual columns. The median was chosen to play down the impact of exceptions, guaranteeing a more dependable and strong ascription prepare.

Once the ascription handle was completed, the dataset was re-examined, affirming that all lost values had been tended to. The dataset is presently completely arranged for modeling.

```
for column in columns_with_zeros:
    data[column] = data[column].fillna(data[column].median())

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:**

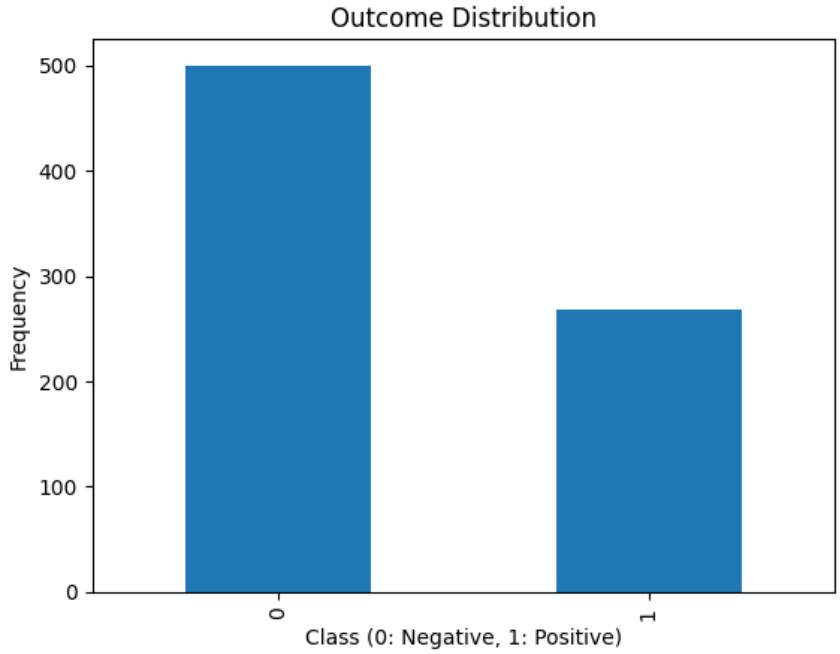
- All missing values have been successfully filled and our dataset is now missing value free.

### 1.2.4. Examination of the Target Variable Distribution

The course conveyance of the Result column, which serves as the target variable, was analyzed to evaluate potential lesson awkwardness within the dataset. The taking after perceptions were made:

```
import matplotlib.pyplot as plt

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's a noticeable class lopsidedness within the dataset. The number of non-diabetic people altogether exceeds the number of diabetic people

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

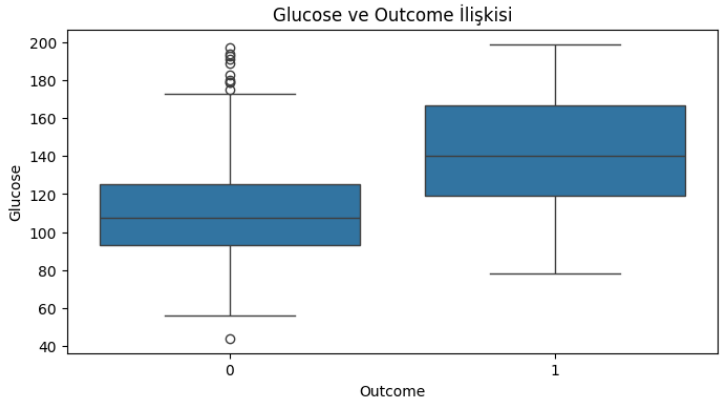
The connections between nonstop highlights within the dataset and the target variable (Result) were analyzed and visualized utilizing boxplots. Underneath are the perceptions and investigations for each highlight:

```
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**:

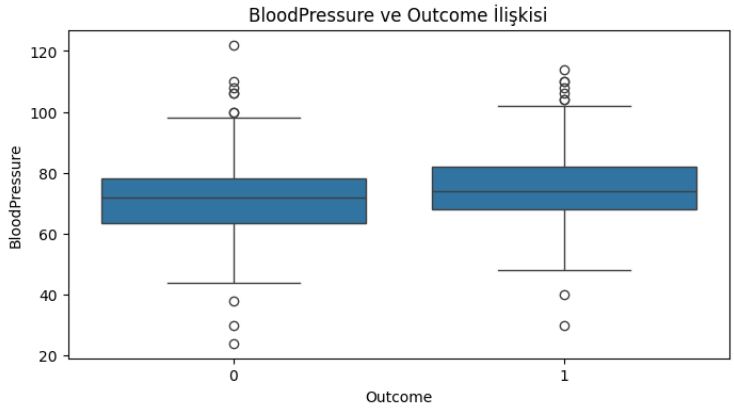
## Glucose and Outcome Relationship:

- The glucose levels of diabetic people (Result = 1) are altogether higher compared to non-diabetic people (Result = 0).
- This shows that glucose levels are a basic highlight for diabetes location.



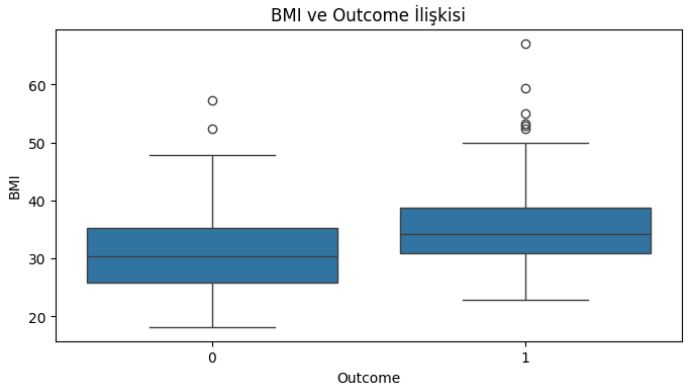

## BloodPressure and Outcome Relationship:

- No noteworthy contrasts were watched in blood weight conveyances between diabetic and non-diabetic people.
- Be that as it may, a few exceptions in blood weight values were distinguished.



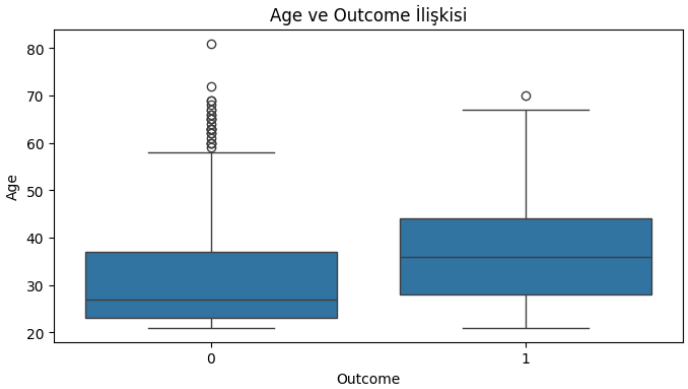
**BMI and Outcome Relationship**:

- Diabetic people (Result = 1) tend to have higher BMI (Body Mass File) values compared to non-diabetic people.
- This recommends that BMI is altogether related with diabetes.




**Age and Outcome Relationship**:

- The normal age of diabetic people is higher than that of non-diabetic people.
- The probability of diabetes increments with age, highlighting age as an critical calculate.




**Insulin and Outcome Relationship**:

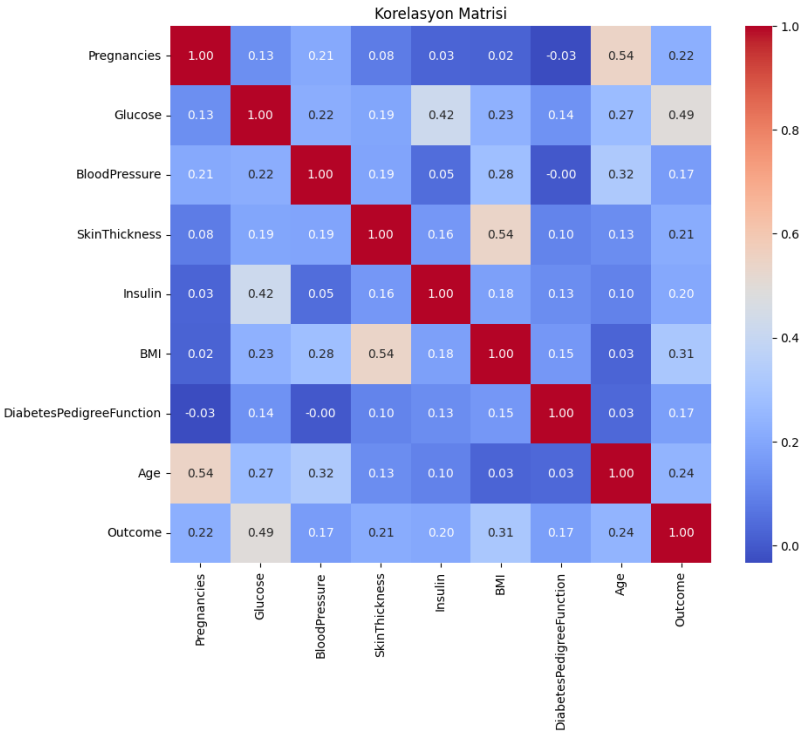
- No clear contrasts in affront levels were watched between diabetic and non-diabetic people.
- In any case, the affront information contains a impressive number of exceptions.

Conclusion:
Highlights such as Glucose, BMI, and Age appear a noteworthy relationship with the target variable and ought to be included within the show. Other highlights require assist examination amid information cleaning and preprocessing steps.

### 1.2.6. Correlation Analysis

To look at the connections between highlights within the dataset, a relationship network was made and visualized. The taking after perceptions were made:

```
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 profoundly related highlights within the dataset. This shows a moo chance of multicollinearity, which might something else contrarily affect the demonstrate.

**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)**.

```
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)

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 improve show execution, modern highlights were included to the dataset. The change connected to the age variable is nitty gritty underneath.

#### 1.2.8.1. Creating Age Groups

**Definition of Age Groups:**
The age variable was isolated into particular interims to make categories such as Youthful, Middle-Aged, Ancient, and Exceptionally Ancient. These bunches were made to way better analyze the affect of age on diabetes.

**Categories:**

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

```
bins_age = [20, 30, 40, 50, 100]
labels_age = ['Young', 'Middle Aged', 'Old', 'Very Old']
data['AgeGroup'] = pd.cut(data['Age'], bins=bins_age,
labels=labels_age)

bins_preg = [-1, 2, 6, 20]
labels_preg = ['Low', 'Medium', 'High']
data['PregnanciesGroup'] = pd.cut(data['Pregnancies'],
bins=bins_preg, labels=labels_preg)

print("\nAge Group Distribution:")
print(data['AgeGroup'].value_counts())

print("\nPregnancy Group Distribution:")
print(data['PregnanciesGroup'].value_counts())
```

**Output:**

```
Yaş Grupları Dağılımı:
Young          417
Middle Aged    157
Old         113
```

```
Very Old      81
Name: AgeGroup, dtype: int64
```

**Conclusion:**
This change permits for a more point by point investigation of the affect of age bunches on the target variable. Such highlight building can improve the model's learning capacity and make strides comes about.

### 1.2.8.2. Creating Pregnancy Groups

**Definition of Pregnancy Groups:**
The number of pregnancies variable was separated into particular interims to form categories such as Moo, Medium, and Tall. These bunches were outlined to superior analyze the affect of the number of pregnancies on diabetes.

**Categories:**

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

```
bins_age = [20, 30, 40, 50, 100]
labels_age = ['Young', 'Middle Aged', 'Old', 'Very Old']
data['AgeGroup'] = pd.cut(data['Age'], bins=bins_age,
labels=labels_age)

bins_preg = [-1, 2, 6, 20]
labels_preg = ['Low', 'Medium', 'High']
data['PregnanciesGroup'] = pd.cut(data['Pregnancies'],
bins=bins_preg, labels=labels_preg)

print("\nAge Group Distribution:")
print(data['AgeGroup'].value_counts())

print("\nPregnancy Group Distribution:")
print(data['PregnanciesGroup'].value_counts())
```

**Output:**

```
Pregnancy Group Distribution:
Low     349
Medium      250
High    169
Name: PregnanciesGroup, dtype: int64
```

**Conclusion:**
This change permits for a more point by point examination of the affect of pregnancy bunches on the target variable. These bunches, beside the AgeGroup highlight, can be included as categorical factors within the show to progress its execution.

### 1.2.9. Feature Scaling

To make strides show execution, ceaseless factors within the dataset were scaled utilizing StandardScaler. This handle standardized the factors and dispensed with issues that may emerge from varying scales amid the show learning handle.

**Process:**

**Scaled Columns:**

- Glucose
- BloodPressure
- SkinThickness
- Insulin
- BMI
- Age

```
from sklearn.preprocessing import StandardScaler

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

scaler = StandardScaler()
data[columns_to_scale] =
scaler.fit_transform(data[columns_to_scale])

print("\nScaled Columns(First 5 Rows):")
print(data[columns_to_scale].head())
```

**Output:**

```
Scaled Columns (First 5 Rows):
    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:**

- Nonstop factors were standardized utilizing StandardScaler.
- As a result, the highlights were scaled to have a cruel of and a standard deviation of 1.
- This scaling disposes of the impacts of varying scales between factors, making a difference the demonstrate create more reliable comes about.

### 1.3. Data Splitting

In this step, the dataset was part into preparing and testing sets to get ready for the modeling handle. The preparing set will be utilized for show learning, whereas the testing set will assess show execution. The part prepare was conducted to preserve the lesson adjust of the target variable.

#### 1.3.1. Splitting Training and Testing Sets

The dataset was part into 80% preparing and 20% testing sets utilizing the train_test_split work. The stratify parameter was utilized to guarantee that the course extents of the target variable, Result, are kept up in both sets.

**Process:**

- **Independent Variables (X):**
  - The Result, AgeGroup, and PregnanciesGroup columns were prohibited to center the show exclusively on ceaseless and center free factors.
- **Dependent Variable (y):**
  - The Result column was assigned as the target variable.

```
from sklearn.model_selection import train_test_split

X = data.drop(['Outcome', 'AgeGroup', 'PregnanciesGroup'], axis=1)
y = data['Outcome']

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

print("Training data size:", X_train.shape)
print("Test data size:", X_test.shape)
```

**Output:**

```
Training data Size: (614, 8)
Test Data Size: (154, 8)
```

**Explanation:**

- **Training Set:** Comprises 614 observations (80% of the data).
- **Testing Set:** Comprises 154 observations (20% of the data).
- **Stratify Parameter:** Guarantees that lesson extents of the target variable are adjusted over both sets.

**Conclusion:**
This step guarantees that the show picks up a generalizable structure amid preparing and is reasonably assessed amid testing.

## 1.4. Modeling and Evaluation

In this part, the Decision Tree model was developed and assessed using the test set. The model's effectiveness was measured through various metrics, including accuracy, precision, recall, and F1-Score. The following outlines the specific procedures and outcomes for the Decision Tree model.

### 1.4.1. Decision Tree Model

The Decision Tree model was created with the sklearn library and trained on the training dataset. Predictions were generated using the test dataset, and performance metrics were assessed.

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

dt_model = DecisionTreeClassifier(random_state=42)

dt_model.fit(X_train, y_train)

y_pred_dt = dt_model.predict(X_test)

# Performans değerlendirme
print("\nDecision Tree Model Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_dt))
print("\nClassification Report:")
print(classification_report(y_test, y_pred_dt))

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")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

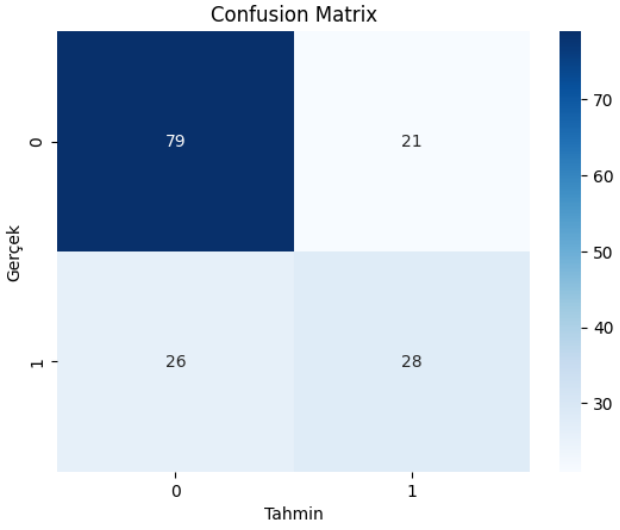
**Output:**

```
Decision Tree Model Performance:
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 The model shows higher accuracy in predicting non-diabetic individuals, achieving 79%, but has difficulty identifying those with diabetes, with only 52% accuracy.
- F1-Score Metrics like the F1-Score suggest that the model's overall performance requires enhancement.
- It appears that class imbalance is impacting the model's effectiveness. To tackle this problem, sampling methods or different modeling approaches could be utilized.

### 1.4.2. Random Forest Model

In this part, the Random Forest model was developed and assessed using the test set. Its effectiveness was measured through various metrics, including accuracy, precision, recall, and F1-Score. The following outlines the specific procedures and outcomes for the Random Forest model. The model was created with the sklearn library and trained on the training set using 100 estimators. Predictions were generated for the test set, and the performance metrics were computed.

```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(random_state=42, n_estimators=100)

rf_model.fit(X_train, y_train)

y_pred_rf = rf_model.predict(X_test)

print("\nRandom Forest Model Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("\nClassification Report:")
print(classification_report(y_test, y_pred_rf))

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("Predicted")
plt.ylabel("Actual")
plt.show()
```

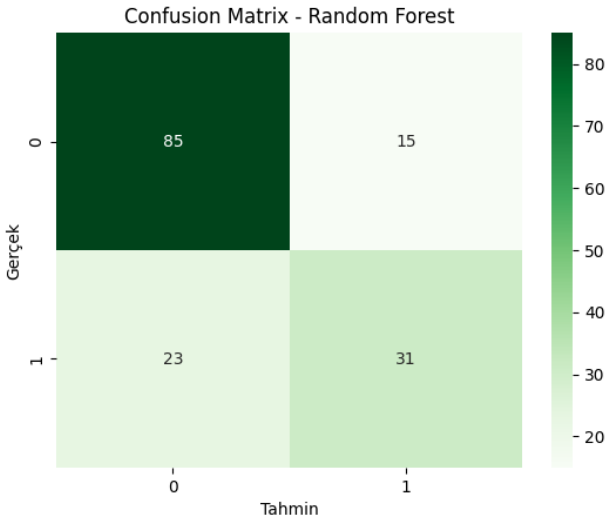
## Output:

```
Random Forest Model Performance:
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 is more effective at identifying non-diabetic individuals, achieving an accuracy of 85%).
- However, it has a lower success rate in detecting diabetic individuals, with only 57% accuracy.
- Overall, the model's accuracy and other performance metrics surpass those of the Decision Tree model.
- Nonetheless, the impact of class imbalance remains evident. Enhancements in performance could be achieved by using class weights or sampling methods.

## 1.4.3. Model Comparison

This part evaluates the performance metrics of the Decision Tree and Random Forest models. The following table presents a summary of their accuracy, precision, recall, and F1-Score for 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 achieved a higher accuracy of 75.32% compared to the Decision Tree model, which had an accuracy of 69.48%.
2. **Class 0 (Non-diabetic):**
    - **Precision:** Both models showed strong performance, with Random Forest achieving a score of 0.79, which was marginally higher than Decision Tree's score of 0.75.
    - **Recall:** Random Forest had a recall rate of 85%, outperforming Decision Tree, which had a recall rate of 79%.
3. **Class 1 (Diabetic):**
    - **Precision:** The Random Forest model (67%) had a greater precision compared to the Decision Tree model (57%).
    - **Recall:** In terms of recall, Random Forest (57%) surpassed Decision Tree (52%).
    - **F1-Score:** Random Forest (62%) attained a superior F1-Score compared to Decision Tree (54%).

**Conclusion:**

- The Random Forest model surpassed the Decision Tree model in every metric.
- It was especially more accurate in identifying individuals with diabetes (Class 1).
- These findings suggest that the Random Forest model is a superior option for this dataset.

### 1.4.4. Strengths and Weaknesses

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

**Name:** Doğukan Erzurum

**Number:** 201401023

**Subject:** Exploring, preprocessing, and clustering Wholesale Customers data using PCA and KMeans.

**Datasets:**

- **Name:** Wholesale Customers Data
- **Source:** UCL Machine
- **Link:** https://archive.ics.uci.edu/dataset/292/wholesale+customers

**Methods Used:** Data Exporation, Data Cleaning, Feature Analysis, Feature Engineering, Feature Scaling, Data Splitting, Modeling, Visualizition, Clustering

## 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 grasp the dataset's structure and create a suitable analysis plan, an initial examination of the dataset was conducted. This involved analyzing the columns, data types, missing values, and overall information regarding the dataset.
2. **Data Preprocessing:**
    - An analysis for missing values was conducted, revealing no missing entries.
    - Outliers were identified and addressed through the IQR method.
    - The entire dataset was standardized using StandardScaler.
3. **Dimensionality Reduction (PCA):**
    - PCA (Principal Component Analysis) was utilized to decrease the dataset's dimensions and enhance the representation of distances between clusters. In this process, the number of components that account for 95% of the overall variance was identified.
4. **Clustering:**
    - The K-means algorithm was utilized.
    - The Elbow method and Silhouette score were used to establish the number of clusters.
    - The outcomes of the clustering were visualized.
5. **Cluster Analysis:**
    - The features of the identified clusters were evaluated.
    - The characteristics of each cluster were analyzed, leading to significant segmentations.
6. **Results and Comments:**
    - The results from the study were analyzed, and conclusions were made regarding customer segmentation.

### 1.1.3 Dataset Introduction and Examination

At this stage, the dataset was imported and preliminary analyses were performed. An examination of the columns, data types, and any missing values in the dataset was carried out. The findings are as follows:

```
import pandas as pd

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

print("First 5 Rows:\n", data.head())

print("\nDataset Information:\n")
data.info()

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**:

Upon reviewing the initial five rows of the dataset, it is evident that the numerical attributes concerning customers are structured effectively. While there isn't a primary key like CustomerID, the data is prepared for immediate analysis. The dataset includes 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.

The dataset information (data.info()) indicates that all columns are of the int64 data type and that there are no missing values. This implies that the dataset appears to be clean at first sight.

The analysis of missing values revealed that there are no gaps in the dataset. This is an important discovery because it streamlines the analysis, particularly for algorithms such as k-means that depend on distance calculations. With no categorical variables needing transformation, we can move straight to scaling the data, which saves both time and effort.

Consequently, the dataset is organized effectively for analyzing customer segments. Nevertheless, in the later phases of this analysis, we will tackle issues like identifying outliers and scaling the data to ready it for clustering.

### 2.1. Correlation and Outlier Analysis

In this phase, fundamental analyses were performed to gain insights into the overall structure of the dataset and the connections between its features. Correlation analysis allows us to assess the strength and direction of relationships among the columns, whereas boxplot analysis provides a visual representation of data distribution and highlights any outliers.

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

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

```
print("\nCorrelation Matrix:\n", data.corr())

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

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

**Explanation**:

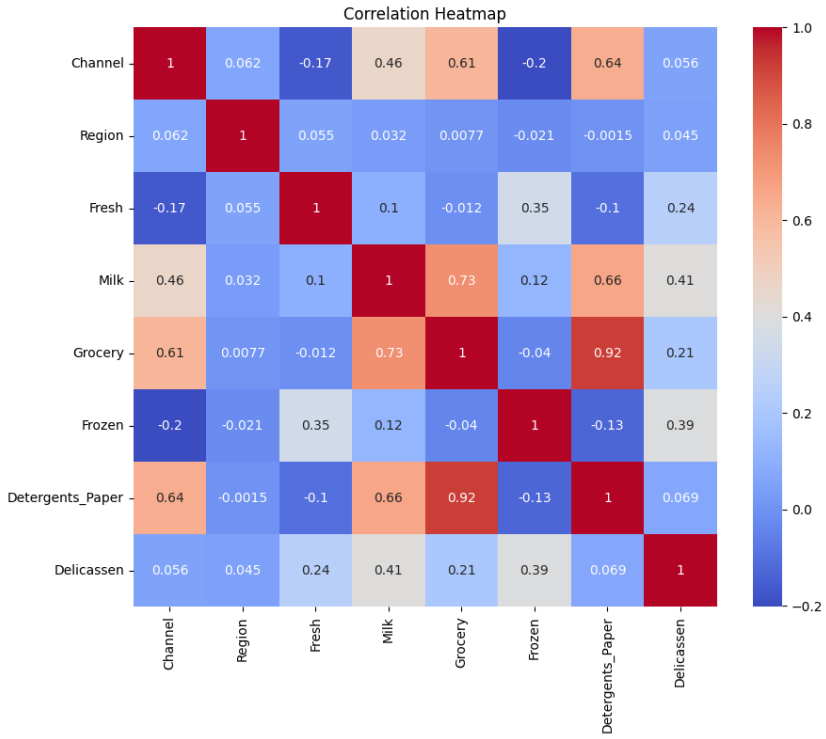
In this code segment, the connections between the columns and the existence of outliers in the dataset were examined using the subsequent steps:

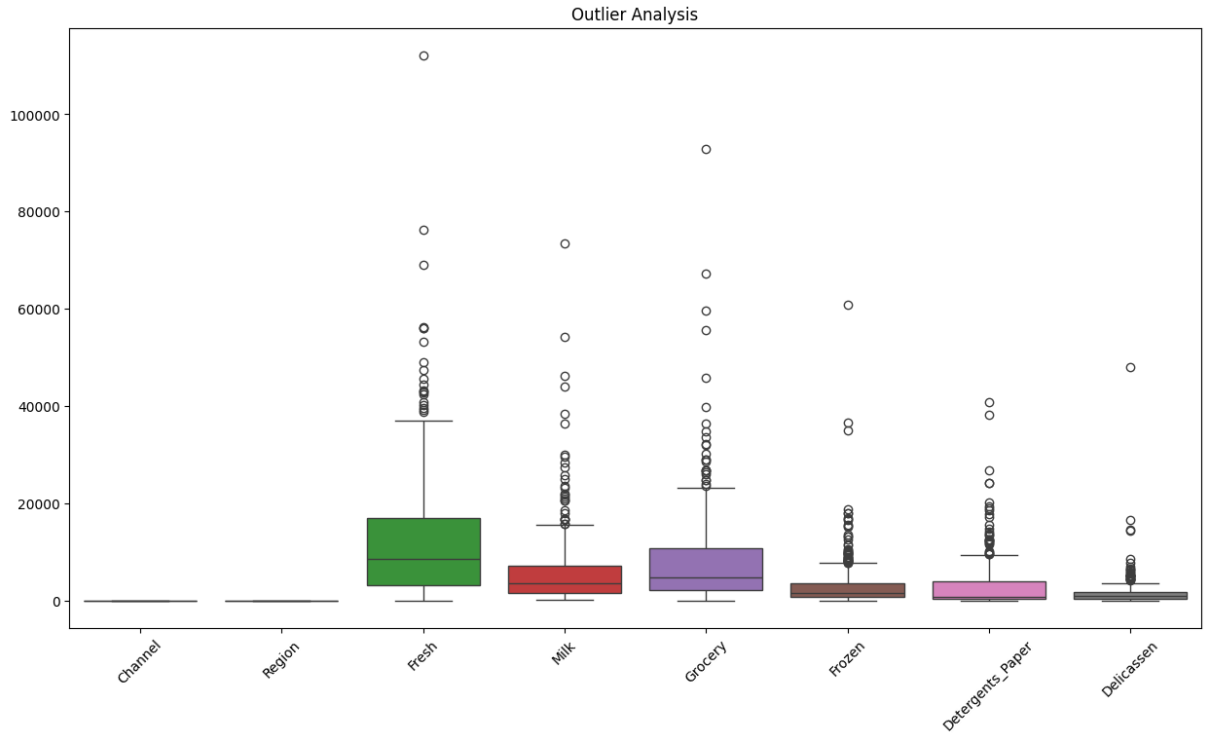
1. **Statistical Summary (describe):**
   - The fundamental statistical characteristics of each column, including mean, median, minimum, maximum, and the 25%-75% quartiles, were displayed.
2. **Correlation Matrix and Heatmap:**
   - Correlation coefficients were computed to assess the relationships between the columns.
   - A heatmap was employed to illustrate the strength of relationships among the columns.
3. **Outlier Analysis (Boxplot):**
   - Additionally, a boxplot was utilized to represent the data distribution and identify potential outliers for each column.
   - This serves as the preliminary visual analysis phase for detecting 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 notable disparity between the highest and lowest values in the output. For instance, in the Fresh column, the highest value is 112151, whereas the median stands at just 8504. This suggests a right-skewed distribution and the likelihood of outliers being present.

- A highly positive correlation of 0.92 was found between Grocery and Detergents_Paper.
- Additionally, a strong correlation of 0.73 exists between Milk and Grocery.
- Overall, weak correlations were noted between Frozen and the other variables, suggesting that Frozen operates more independently.
- The boxplot analysis reveals a considerable number of outliers in the Fresh, Grocery, and Frozen columns, which should be eliminated to avoid complications in the data analysis.

The correlation analysis provided insights into the relationships among the features. Features that exhibit strong correlations, such as Grocery and Detergents_Paper, can be grouped together in the clustering process. Furthermore, the boxplot analysis revealed the existence of outliers in certain columns. These extreme values will be addressed in the next phase to improve the effectiveness of the clustering algorithm.

### 2.2. Detection and Cleaning of Outliers

In data analysis and machine learning, outliers are data points that differ greatly from the overall distribution of the dataset. These anomalies can create significant issues, particularly for distance-based algorithms such as K-means. Outliers can lead to inaccurate calculations of cluster centers, which in turn diminishes the accuracy and dependability of the analysis outcomes.

In this phase, the Interquartile Range (IQR) method was employed to identify outliers. The IQR is the difference between the first quartile (Q1) and the third quartile (Q3) of a dataset. Any data points that fall outside the IQR limits are deemed outliers and are eliminated. This procedure aims to create a more uniform dataset, which enhances the performance of the clustering algorithm.

```
from sklearn.preprocessing import StandardScaler

Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)
IQR = Q3 - Q1

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}")

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

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 eliminating outliers, the dataset's size decreased from 440 to 332.

Outliers frequently occur when there are notable variations in the behaviors of individual customers within the dataset. For example, a customer who spends an unusually large amount in the Fresh category may stand out considerably from the overall customer base. This data cleaning process helps the K-means algorithm form more even clusters. Without this step, outliers could lead to inaccurate outcomes from the algorithm.

This stage is an essential component of the preprocessing procedure. Furthermore, due to the strong correlations among the columns in the dataset, we will tackle these correlations in the subsequent steps through feature selection. For instance, the correlation between Grocery and Detergents_Paper is notably high at 0.92, suggesting that the data in these columns significantly overlaps.

### 3.1. Scaling of Data (PCA)

In data analysis and machine learning, an excessive number of features can make the analysis more complex and hinder the performance of algorithms. To address this issue, dimensionality reduction methods such as PCA (Principal Component Analysis) are utilized. PCA seeks to decrease the number of variables while preserving the crucial information within the dataset.

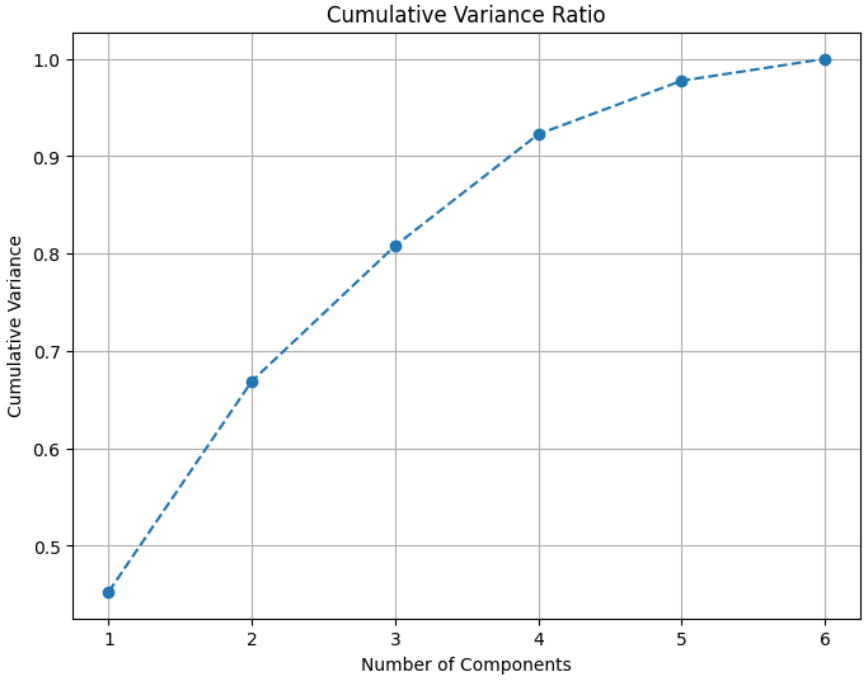
In the PCA process, every new principal component accounts for a segment of the dataset's variance. The objective is to choose a sufficient number of components to capture a significant amount of the overall variance. In this research, we identified the number of components that account for at least 95% of the total variance. This approach is especially effective for removing redundant information from highly correlated features and for developing a more streamlined data representation.

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

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

explained_variance_ratio = pca.explained_variance_ratio_
print("Explained Variance Ratios:\n", explained_variance_ratio)
```

```
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()

n_components = np.argmax(cumulative_variance >= 0.95) + 1
print(f"Number of components explaining 95% variance:
{n_components}")

pca = PCA(n_components=n_components)
pca_data_reduced = pca.fit_transform(scaled_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 as follows: [0.45, 0.21, 0.14, 0.11, 0.05, 0.02]. The cumulative variance graph indicated that the first five components accounted for 95% of the total variance. Following the PCA transformation, the dataset has been condensed to a 5-dimensional format.

Using PCA, we decreased the dataset's original dimensions from 6 to 5. This helps remove excess redundancy while preserving the majority of the essential information in the dataset.

The initial component accounts for approximately 45% of the overall variance, and when combined with the second component, they explain 66% of the variance. This suggests that a significant portion of the key information in the dataset is contained within a limited number of components.

PCA successfully eliminates the unnecessary information present in highly correlated columns, like Grocery and Detergents_Paper. This enhances the efficiency of the K-means algorithm.

The hyperparameter for PCA is the quantity of components chosen. By selecting the number of components that account for 95% of the variance, we made certain that the K-means algorithm functions more effectively with reduced dimensions.

### 4.1. K-Means Clustering and Cluster Analysis

We have now organized our dataset and applied PCA to reduce its dimensions. Next, we will use the K-means algorithm to group the customers into clusters. To find the best number of clusters, we will employ techniques like the Elbow method and Silhouette score. Furthermore, we will examine the resulting clusters and interpret the traits of each one.

The K-means algorithm divides data points into groups by allocating each point to the closest cluster center, which helps make the clusters more uniform. A drawback of this algorithm is that the number of clusters (k) needs to be specified in advance. As a result, choosing the appropriate value for k is essential for the accuracy of the analysis outcomes.

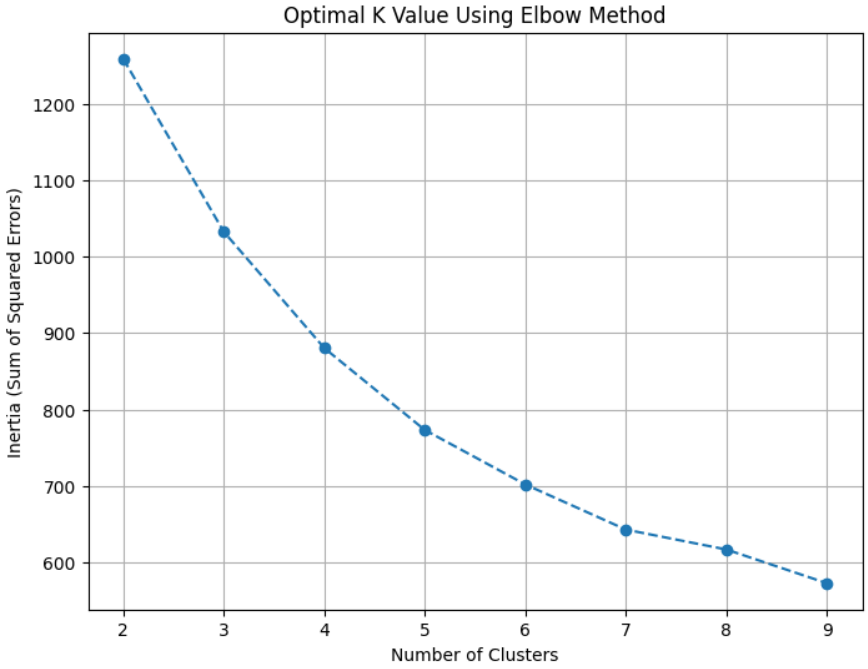
In In this research, the number of clusters was established through the following methods:

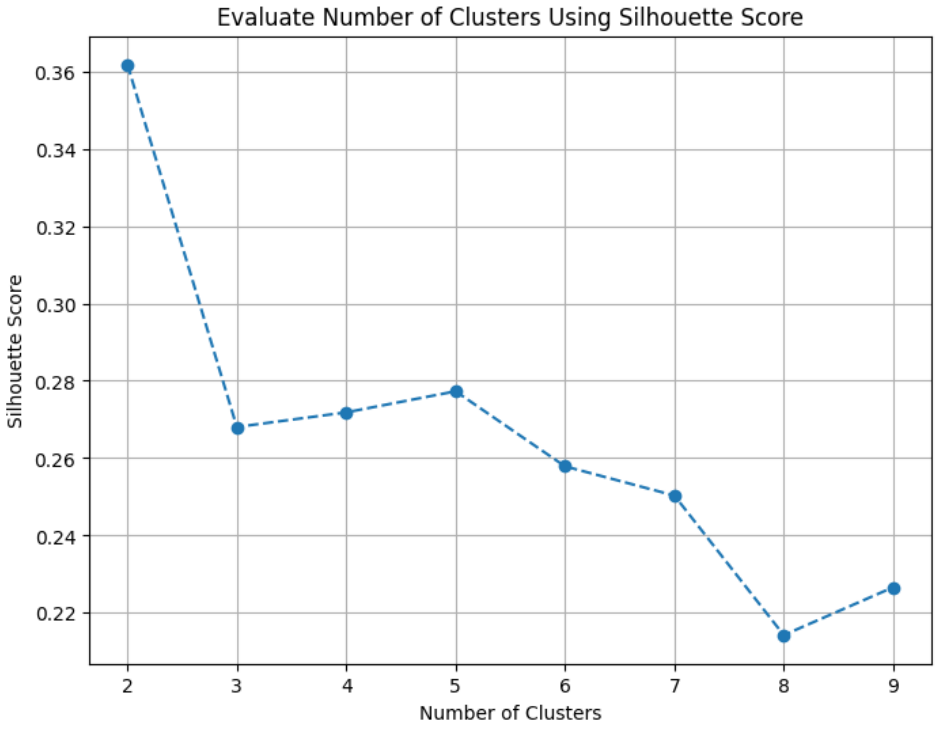
1. **Elbow Method:** We analyzed the decrease in the sum of squared errors (inertia) relative to the number of clusters to find the point where the curve shows an "elbow."
2. **Silhouette Score:** This measure assesses the cohesion within clusters and the separation between them. A high Silhouette score suggests that the clusters are both distinct from one another and cohesive internally.

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

```
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_))

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()

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()

optimal_k = silhouette_scores.index(max(silhouette_scores)) + 2
print(f"Optimal Number of Clusters (Based on Silhouette):
{optimal_k}")

kmeans = KMeans(n_clusters=optimal_k, random_state=42)
kmeans.fit(pca_data_reduced)

print("Cluster Labels:\n", kmeans.labels_)
```

**Output:**


Optimal K Value Using Elbow Method
Inertia (Sum of Squared Errors)
1200
1100
1000
900
800
700
600
2
3
4
5
6
7
8
9
Number of Clusters


Evaluate Number of Clusters Using Silhouette Score
Silhouette Score
0.36
0.34
0.32
0.30
0.28
0.26
0.24
0.22
2
3
4
5
6
7
8
9
Number of Clusters

```
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 1 0 1 1 0 1 1 1 0 1 1 1 1 1
1 1 1
 1 1 1 0 1 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 allocated to the corresponding cluster center.

After analyzing the Silhouette score and the Elbow method, we determined that utilizing 2 clusters is significant. This indicates the presence of two main customer segments within the dataset. Each data point was allocated to the closest cluster center, showcasing the basic operation of the K-means algorithm.

At this stage, the number of clusters (k) is a key hyperparameter in the K-means algorithm. Metrics like the Silhouette score are essential for identifying the best value of k.

### 5.1. Analysis of Cluster Characteristics

At this point, we will conduct a thorough analysis of the clusters generated by the K-means algorithm. Our objective is to gain insights into customer segments by identifying the key characteristics of each cluster.

The K-means algorithm categorizes each data point into the most appropriate cluster, creating separate groups. Nevertheless, a thorough examination of these clusters is necessary to gain

practical insights. By assessing the statistical characteristics of each cluster, we can determine the key features that stand out in each group.

We will specifically investigate if there are notable variations among clusters regarding their spending behaviors and geographic traits. This analysis will aid in identifying the customer segments that each cluster corresponds to.

```
filtered_data = data[~((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5
* IQR))).any(axis=1)].copy()
filtered_data['Cluster'] = kmeans.labels_

cluster_summary = filtered_data.groupby('Cluster').mean()
print("Mean Values of Cluster Features:\n", cluster_summary)
```

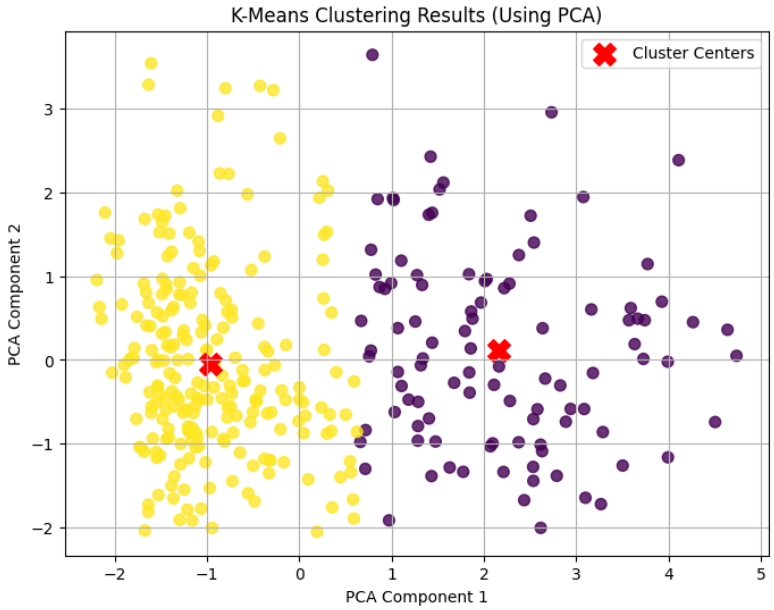
**Outputs:**

```
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 average values for each cluster were determined, and the key traits of each cluster were recognized. Cluster 0 shows increased expenditure on groceries and cleaning supplies, while Cluster 1 displays lower spending on milk but greater investment in fresh items. Cluster 0 (Grocery-Focused Customers) likely represents individual retail shoppers, as their higher spending on grocery and cleaning products suggests they are personal consumers. In contrast, Cluster 1 (Fresh-Product-Focused Customers) appears to belong to the Horeca (hotel/restaurant/café) sector, as their strong preference for fresh products points to a business-oriented demographic. This analysis has enhanced our understanding of the different customer segments. Now, let's visualize this information for clearer insights.

## 6. Results and Comments




In this research, we conducted customer segmentation utilizing the K-means algorithm. We thoroughly executed steps including data preprocessing, dimensionality reduction, and clustering. Consequently, we categorized the customers into two distinct clusters and examined the key characteristics of these groups. The findings offer valuable insights into customer behavior and can aid in formulating various strategies.

Cluster 0 (Individual Consumers): This group of customers prioritizes their everyday necessities, including items like groceries and cleaning supplies.

Cluster 1 (Commercial Consumers): This group of customers tends to invest more in fresh products, suggesting an emphasis on the Horeca (hotel/restaurant/cafe) industry.

PCA successfully eliminated unnecessary information from the dataset, enhancing the efficiency of the K-means algorithm. In particular, it reduced the impact of strongly correlated columns.

Eliminating outliers led to more uniform clusters and enhanced the precision of the analysis outcomes.

Normalizing the data enabled the algorithm to consider all features as equally significant.

Cluster 0: Unique promotions can be arranged for retail merchandise and everyday essential items.

Cluster 1: Supply chain solutions for fresh products can be created for the Horeca industry.

This research illustrated the application of the K-means algorithm for segmenting customers. By meticulously carrying out data preprocessing, reducing dimensionality, and fine-tuning hyperparameters, the algorithm's effectiveness was greatly improved. Two primary customer

segments were recognized, creating opportunities to formulate business strategies specifically designed for these groups.

# QUESTION-3 REPORT

**Name:** Doğukan Erzurum

**Number:** 201401023

**Subject:** Comparing and training Linear Regression, Ridge Regression, Lasso Regression, and Random Forest models on Real Estate Valuation data

**Datasets:**

- **Name:** Real Estate Valudation
- **Source:** UCL Machine
- **Link:** https://archive.ics.uci.edu/dataset/477/real+estate+valuation+data+set

**Methods Used:** Data Exporation, Normalization, Standardization, Outlier Removel, Lineer Regresssion, Lasso Regression, Random Forest Regression

## Question 3

## 1. Introduction

### 1.1. Project Objective

Regression analysis is a supervised learning technique employed to forecast a continuous target variable using explanatory variables. This research focused on predicting house prices per unit area through linear regression and various advanced regression methods. By evaluating the effectiveness of different regression models, we aimed to determine which method offers superior accuracy and reliability for this dataset.

The study utilizes the Real Estate Valuation Dataset sourced from the UCI Machine Learning Repository. This dataset includes details like transaction dates, the age of houses, proximity to the nearest MRT station, the number of convenience stores, and geographic coordinates. These attributes provide essential information for developing regression models aimed at predicting property prices.

The procedures undertaken in this research are outlined below:

1. **Data Preprocessing:** An analysis of missing values was performed, outliers were removed, and all features were standardized for uniformity.
2. **Feature Selection:** The significance of each feature was evaluated to guarantee that only relevant features were incorporated into the model.
3. **Model Training and Evaluation:** Linear regression was developed and evaluated alongside more sophisticated methods such as Ridge Regression, Lasso Regression, and Random Forest Regression.
4. **Hyperparameter Optimization:** For the more sophisticated models, important hyperparameters were adjusted to enhance their effectiveness.
5. **Performance Comparison:** The models were assessed using metrics like Mean Squared Error (MSE) and R-squared ($R^2$), and the outcomes were analyzed.

The findings of this research can offer practical guidance for assessing real estate value and making informed decisions. This report thoroughly outlines each step, complete with relevant code, results, and explanations.

### Dataset and Preprocessing

In this research, we utilized the Real Estate Valuation Dataset, which contains 414 data entries and 8 characteristics. The target variable, house_price_per_unit_area, indicates the price of houses per unit area. The dataset includes the following features:

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 procedure used in this research consists of the following steps:

1. **Dataset Introduction and Examination:**
    - The structure of the dataset was analyzed to create an appropriate regression approach.
    - At this stage, columns, data types, missing values and general statistics of the dataset were reviewed to identify potential issues or needs.
2. **Data Preprocessing:**
    - Missing values were analyzed and it was found that the dataset did not contain any missing entries.
    - Outliers were identified and removed using the interquartile range (IQR) method, specifically targeting distance_to_MRT and house_price_per_unit_area.
    - All attributes were standardized using StandardScaler to maintain data uniformity and prepare for regression analysis.
3. **Feature Selection:**
    - Unnecessary columns (e.g. "No") were removed to optimize the dataset.
    - The features were analyzed to assess their significance in relation to the target variable. A correlation analysis was conducted to reveal the relationships between the features.
4. **Model Training and Evaluation:**
    - The data was divided into two parts: 80% for training and 20% for testing.
    - A linear regression model was developed and evaluated using evaluation metrics such as mean squared error (MSE) and R-squared ($R^2$).
    - Advanced techniques such as ridge regression, lasso regression and random forest regression were used to evaluate their performance compared to each other.
    - Hyperparameters such as alpha in ridge and lasso regression and n_estimators in random forest regression were adjusted to improve performance.
5. **Performance Comparison:**
    - The effectiveness of all regression models was evaluated using test data.
    - The results are visualized and an assessment of the advantages and disadvantages of each model is made.
6. **Results and Comments:**
    - The research results were analyzed in relation to the prediction of real estate prices.
    - The analysis of the dataset and the performance of the models revealed the most successful regression methods for specific data.

### 1.1.3 Dataset Introduction and Examination

At this stage, the dataset was imported and preliminary analyses were performed. Structure, columns, data types and all missing values were assessed to create an appropriate preprocessing and regression strategy.

```
import pandas as pd
```

```
data = pd.read_excel("Real estate valuation data set.xlsx")
print("First 5 Rows:\n", data.head())

print("\nDataset Information:\n")
data.info()

print("\nMissing Values:\n", data.isnull().sum())

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 consists of 414 entries and 8 columns detailing various property characteristics and prices per unit area. The columns include:
    1. **No**: Line numbers of no significant analytical significance will be omitted in future analyses.
    2. **X1 transaction date**: Trade date displayed as a decimal number.
    3. **X2 house age**: The number of years since the house was built.
    4. **X3 distance to the nearest MRT station**: The distance to the nearest metro station is expressed in meters.
    5. **X4 number of convenience stores**: The number of markets within a certain distance of the property.
    6. **X5 latitude** and **X6 longitude**: Location coordinates of the property.
    7. **Y house price of unit area**: The variable of interest is the price of a house per unit area.

isnull() function rationalizes the preprocessing phase by identifying all missing values in the columns, making imputation methods redundant. The dataset is well structured for regression analysis. However, it is important to perform preprocessing activities such as outlier detection and feature scaling to improve the accuracy and reliability of the model. The dataset is clean and ready for preprocessing without missing values or categorical transformations. The next steps will focus on outlier detection and scaling of the data to better prepare it for regression modeling.

**Renaming Columns:**

Column names have been changed to improve the clarity and readability of the dataset. The new names are more meaningful and more accurately reflect the information in each column. This change facilitates future analysis and improves clarity in interpreting the dataset.

```
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)

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:**

Column names have been 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 improved both the readability of the dataset and the analysis workflow by making the column names intuitive and self-explanatory.

### 2. Data Preprocessing

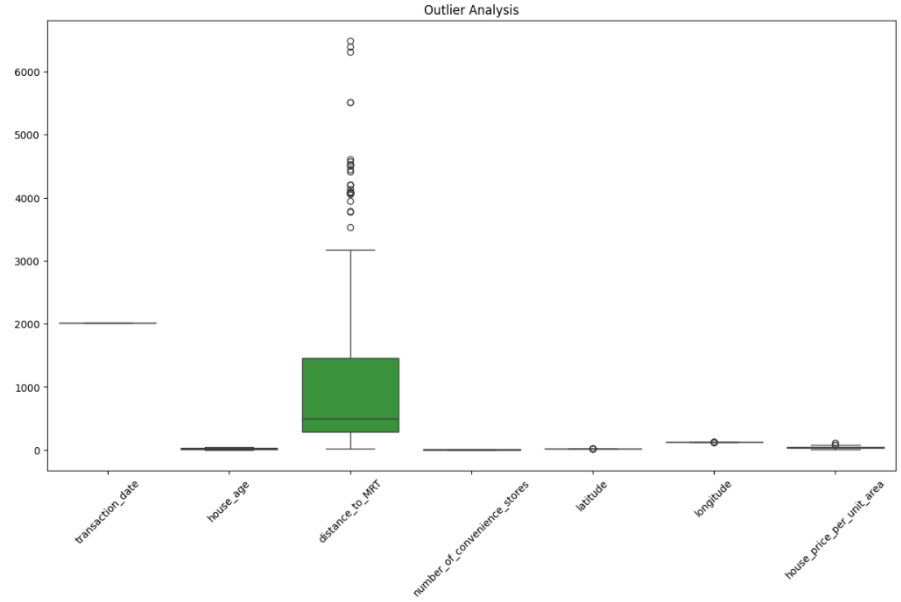
Data preprocessing is a crucial step in any data analysis or machine learning project. It ensures that the dataset is clean, consistent and ready for modeling. In this study, the following preprocessing steps were applied:

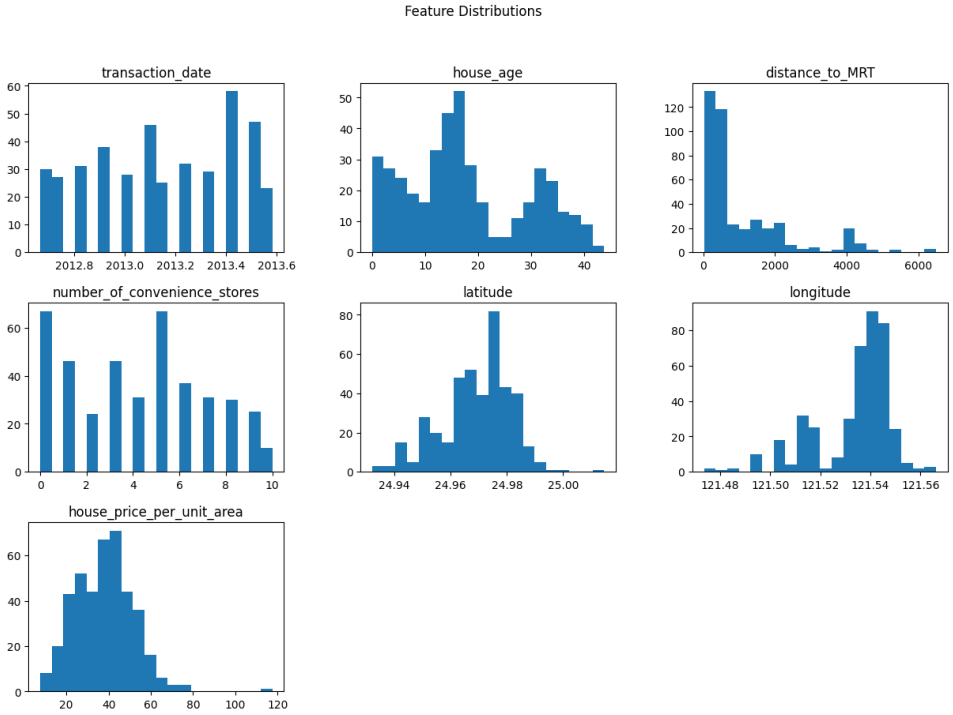
#### 2.1. Outlier Detection and Removal

In order to prepare the dataset for regression analysis, an exploratory data analysis (EDA) was performed to identify outliers and learn about the distribution of features. This process involved two important visualizations: a boxplot to analyze outliers and histograms to assess the distribution of features.

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

data_analysis = data.drop(columns=["No"])

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

data_analysis.hist(bins=20, figsize=(15, 10), grid=False)
plt.suptitle("Feature Distributions")
plt.show()
```

**Output:**


Outlier Analysis
6000
5000
4000
3000
2000
1000
0
transaction_date
house_age
distance_to_MRT
number_of_convenience_stores
latitude
longitude
house_price_per_unit_area


Feature Distributions
transaction_date
house_age
distance_to_MRT
number_of_convenience_stores
latitude
longitude
house_price_per_unit_area

## Comments:

A boxplot was created for each feature to visualize possible outliers. The distance_to_MRT feature showed significant outliers, as can be seen by its long whiskers and outliers. “Other features such as 'house_price_per_unit_area' also showed some outliers, but less noticeable compared to 'distance_to_MRT'.

Histograms were created for all attributes to analyze their distribution. distance_to_MRT attribute is skewed to the right, indicating that most properties are closer to MRT stations and only a few properties are further away. house_price_per_unit_area is almost normally distributed and therefore well suited for regression analysis. “Functions such as 'number of convenient stores' and 'house age' showed discrete or uniform distributions.

### 2.2. Outlier Removal

Outliers in a dataset can skew regression models by amplifying the perceived relationships between features and the target variable. To mitigate this, the interquartile range (IQR) method is applied to filter out outliers from key features like distance_to_MRT and *house_pricehouse_price_per_unit_area.

**Steps:**

1. **Identify Outliers:**
    - First quartile (Q1), third quartile (Q3) and interquartile range (IQR) are calculated for each attribute.
    - Outliers are defined as the following values 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:**
    - To maintain the consistency of the dataset, rows with outliers in distance_to_MRI and house_price_per_unit_area were removed.

```
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)]

data_cleaned = remove_outliers(data, "distance_to_MRT")

data_cleaned = remove_outliers(data_cleaned,
"house_price_per_unit_area")

print("Dataset size after removing outliers:", data_cleaned.shape)
```

**Output:**

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

**Comments:**

After using the IQR method, the data setting size decreased from 414 entries to 373 entries; this shows that about 10% of the data have outliers in the two target functions. Removing these outliers helps to focus on general patterns in the data setting without being affected by anomalies of the regression modes.

### 3. Feature Scaling

Feature scaling is an essential step before performing regression analysis, especially when the dataset includes features measured on different scales. Differences in scale can impact how regression models perform. To address this issue, all numerical features are standardized using the standard scaler method.

**Steps:**

1. Remove unnecessary columns such as “No” that do not contribute to the analysis.
2. Use StandardScaler to transform all numeric features and scale them so that they have a mean of 0 and a standard deviation of 1.
3. Save the scaled data in a new DataFrame for use in later modeling steps.

```
from sklearn.preprocessing import StandardScaler

data_for_scaling = data_cleaned.drop(columns=["No"])
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data_for_scaling)

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 converted all numeric features to a standard scale, resulting in a mean of 0 and a standard deviation of 1. This process prevents a single characteristic from having an excessive influence on the regression models due to different scales. Scaled features such as distance_to_MRT and house_price_per_unit_area are now ready for regression analysis.

### 4.Model Training and Evaluation

Regression models were used to predict the target variable house_price_unit_area. To evaluate the models, the scaled data set was divided into training and test sets. The training set, consisting of 80% of the data, is used to train the models, while the remaining 20% is reserved for testing and evaluation.

```
from sklearn.model_selection import train_test_split

X = scaled_data_df.drop(columns=["house_price_per_unit_area"])
y = scaled_data_df["house_price_per_unit_area"]

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

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

lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

y_pred = lr_model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

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 produced an R-squared value of 0.6543; this shows that around 65% of the fluctuations in property prices per unit area are explained by the model. The mean square error (MSE) is 0.3859 and reflects the mean square difference between predicted values and actual values.

### 4.2. Ridge Regression

Ridge regression is a more complex version of linear regression that uses L2 regularization to minimize overfitting by penalizing large coefficients. This method is particularly useful when multicollinearity is present in the data, as it helps to balance the model by reducing the coefficients of correlated variables.

```
from sklearn.linear_model import Ridge

ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train, y_train)

y_pred_ridge = ridge_model.predict(X_test)

mse_ridge = mean_squared_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)

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 produced an R-squared value of 0.6537, which was slightly lower than that of the linear regression model. The mean squared error (MSE) was 0.3865, nearly identical to the linear regression model. These results indicate that ridge regression does not significantly outperform the base model, likely due to the already simple linear relationships and low multicollinearity in the dataset.

### 4.3. Lasso Regression

Lasso regression is an advanced linear regression technique that applies L1 regularization. Unlike ridge regression, lasso can shrink some coefficients to zero, effectively performing feature selection. This property makes lasso particularly useful when certain features have little to no impact on the target variable.

```
from sklearn.linear_model import Lasso

lasso_model = Lasso(alpha=0.1)
lasso_model.fit(X_train, y_train)

y_pred_lasso = lasso_model.predict(X_test)


mse_lasso = mean_squared_error(y_test, y_pred_lasso)
r2_lasso = r2_score(y_test, y_pred_lasso)

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 produced 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, showing that the prediction accuracy is not as high as the other models. The feature selection capability of lasso did not offer a significant benefit in this case, suggesting that all features played a meaningful role in predicting the outcome. However, there is room for improvement, particularly by fine-tuning the regularization parameter (alpha), which could enhance the model's performance.

### 4.4. Random Forest Regression

Random Forest Regression is a powerful ensemble learning technique that makes predictions by combining multiple decision trees. Unlike linear models, Random Forest can capture non-linear relationships in the data and handle interactions between different features more effectively.

```
from sklearn.ensemble import RandomForestRegressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

y_pred_rf = rf_model.predict(X_test)

mse_rf = mean_squared_error(y_test, y_pred_rf)
```

```
r2_rf = r2_score(y_test, y_pred_rf)

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 gave a much better result than the other models. In this model, we achieved an R^2 value of 0.7744, which indicates that approximately 77% of the variables can be answered by the moedl according to their observed states. In addition, the mean squared error (MSE) value of the model gave a better result than the other models. This result was 0.2518. According to these results, we can see that the model with the most accurate prediction ability is the random forest regression model. Therefore, it is much more important to choose the random forest model to make more accurate predictions.

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 model achieved an MSE of 0.3859 and R2 of 0.6543 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.