# Heart Diseases

In this notebook we are Covering the DataMining course project.

## 1-Problem[2]

The problem of heart disease prediction is framed as a data mining challenge to identify risk factors and predict the likelihood of heart disease using medical data. Early detection of heart disease is crucial for improving healthcare outcomes by enabling timely and targeted interventions. This aligns with the broader goal of reducing mortality rates and enhancing the quality of life for patients.

The dataset used in this study is highly appropriate as it contains medically relevant features such as patient demographics, vital signs, and clinical test results, which are essential for predictive modeling. Solving this problem contributes to real-world solutions, including informing healthcare policy-making, developing preventive strategies, and personalizing medical treatments.

## 2-Data Mining Task[4,5]

In our project, we will study and analyze patients' data that will help us well in identifying possible factors and risks that lead to heart diseases and help many people to take precautions by predicting the possibility of having a heart disease.

In our project we will use two data mining tasks to help us predict the possibility of having heart diseases :

- Classification:

-Classification is essential for binary outcomes, such as predicting whether a patient is at risk of heart disease. It enables the development of a predictive model that can assist in identifying high-risk individuals based on specific medical attributes.

- Evaluation metrics for classification include:

-Accuracy: Measures the overall correctness of the model.

-Confusion Matrix: Provides insight into the model's performance across true positives, true negatives, false positives, and false negatives.

- Clustering:

-Clustering is important for grouping patients based on similar characteristics, enabling personalized treatments and targeted healthcare strategies.

-This task identifies patterns within the dataset, such as clusters of patients with similar risk profiles, which can assist in healthcare resource allocation.

- Evaluation metrics for clustering include:

-Silhouette Score: Measures how well each data point fits within its cluster.

-Within-Cluster Sum of Squares (WCSS): Assesses the compactness of clusters.

These tasks work together to predict individual patient risks and identify population-level patterns, bridging the gap between data insights and actionable healthcare outcomes.

- The class attribute in this task is the target, which indicates the presence or absence of heart disease.

The dataset used in this study : https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset

Number of objects : 1026
Number of attributes : 13

Attributes:

1- age

2- sex

3- chest pain type (4 values)

4- resting blood pressure

5- serum cholestoral in mg/dl

6- fasting blood sugar > 120 mg/dl

7- resting electrocardiographic results (values 0,1,2)

8- maximum heart rate achieved

9- exercise induced angina

10- oldpeak = ST depression induced by exercise relative to rest

11- the slope of the peak exercise ST segment

12- number of major vessels (0-3) colored by flourosopy

13- thal: 0 = normal; 1 = fixed defect; 2 = reversable defect

## 3-Data

We sourced our data set from Kaggle (https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset?resource=download) It contains heart-related attributes and their impact on heart disease risk.

```
import pandas as pd
df = 'heart.csv'
heart_data = pd.read_csv(df)

print("---------------------------------------------------")
print(f"Number of object: {heart_data.shape[0]}")
print(f"Number of attributes: {heart_data.shape[1]}")
print("---------------------------------------------------")
print("Column Names:")
print(heart_data.columns.tolist())
print("---------------------------------------------------")
print("Data Types of Each Column:")
print(heart_data.dtypes)
print("---------------------------------------------------")
print("\nMissing Values in Each Column:")
print(heart_data.isnull().sum())
```

```
---------------------------------------------------
Number of object: 1025
Number of attributes: 14
---------------------------------------------------
Column Names:
['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target
---------------------------------------------------
Data Types of Each Column:
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
---------------------------------------------------

Missing Values in Each Column:
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

Based on the provided results, the dataset contains 1,025 objects (rows) and 14 attributes (columns). All columns are numeric, with most having an integer (int64) data type, except for oldpeak, which is a floating-point number (float64). Additionally, there are no missing values in any of the columns, as all attributes show a count of zero missing entries.The target variable ('target') indicating the presence or absence of heart disease.

Key attributes include:

'age': Patient's age.

'sex': Biological sex (1 = male, 0 = female).

'cp': Type of chest pain experienced.

'trestbps': Resting blood pressure (mmHg).

'chol': Serum cholesterol (mg/dl).

'thalach': Maximum heart rate achieved.

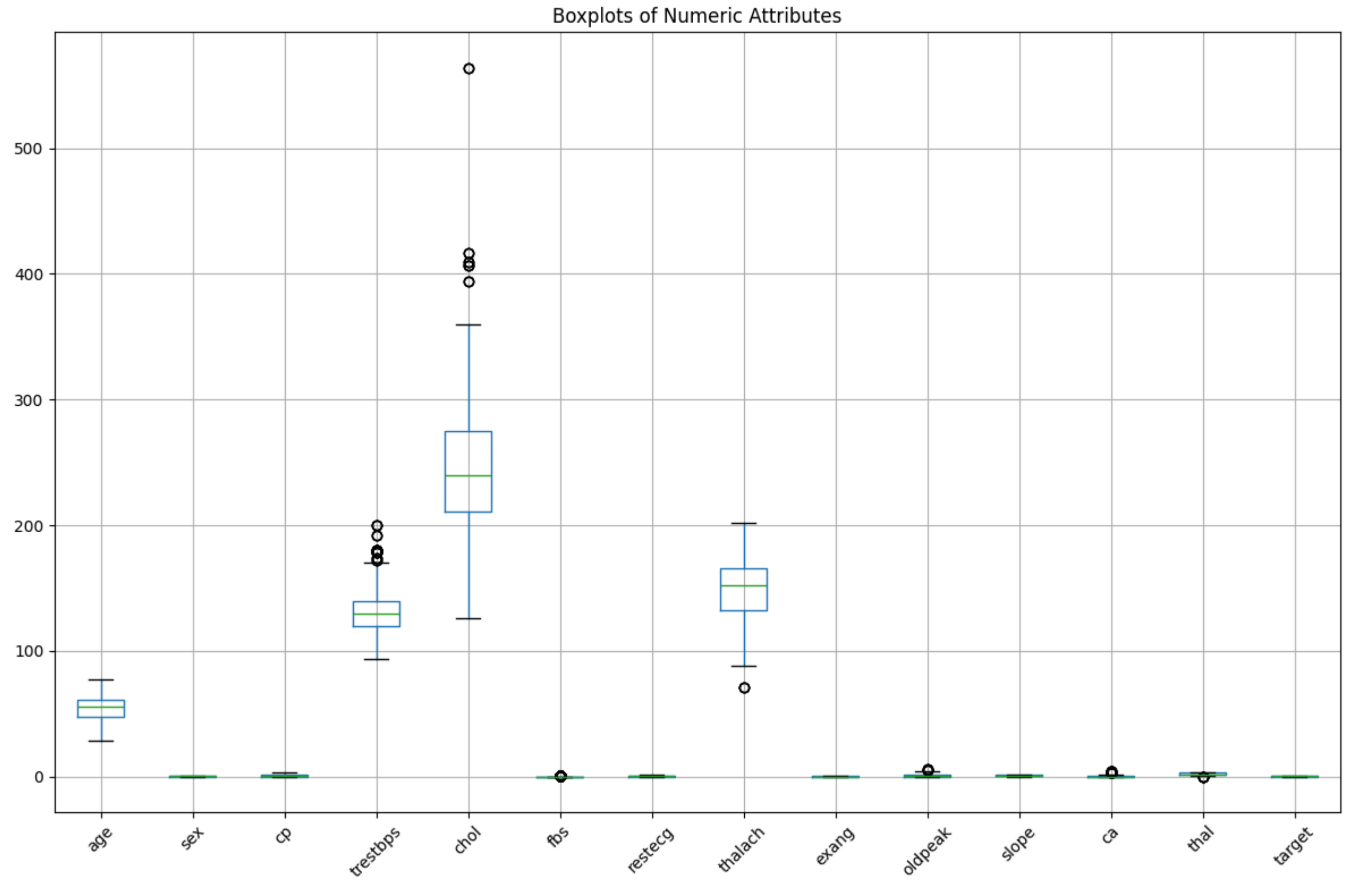
'oldpeak': ST depression induced by exercise.

'ca': Number of major vessels.

'thal': Type of thalassemia (e.g., normal, fixed defect, reversible defect).

All attributes are clean, numeric, and ready for analysis.

```
import numpy as np
import matplotlib.pyplot as plt

#  ive-number summary for the numeric attributes
numeric_columns = heart_data.select_dtypes(include=[np.number]).columns
five_number_summary = heart_data[numeric_columns].describe(percentiles=[0.25, 0.5, 0.75]).T
outliers = {}
for col in numeric_columns:
    Q1 = five_number_summary.loc[col, '25%']
    Q3 = five_number_summary.loc[col, '75%']
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers[col] = heart_data[(heart_data[col] < lower_bound) | (heart_data[col] > upper_bound)][col]


plt.figure(figsize=(12, 8))
heart_data[numeric_columns].boxplot()
plt.title('Boxplots of Numeric Attributes')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



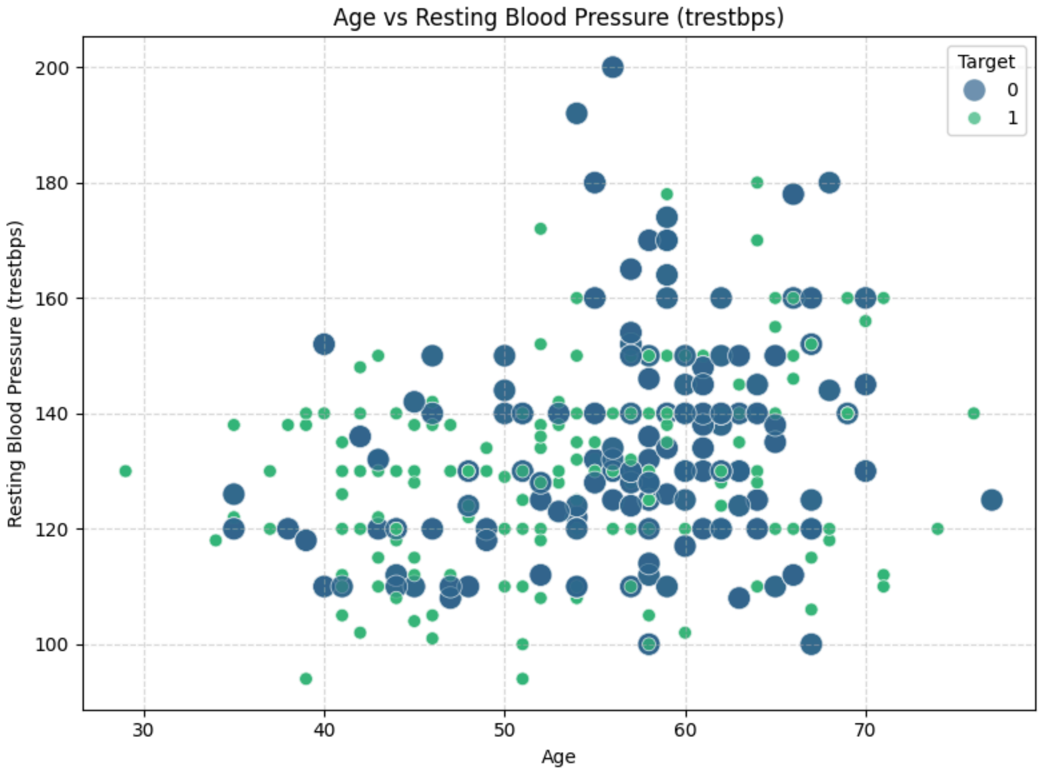

Analysis of Numeric Attributes A five-number summary of the dataset's numeric attributes (minimum, first quartile, median, third quartile, and maximum) provides insights into the central tendency and spread of the data. Outlier detection using the IQR method identified notable anomalies in attributes such as 'trestbps' (resting blood pressure), 'chol' (serum cholesterol), 'fbs' (fasting blood sugar), and 'ca' (number of major vessels).

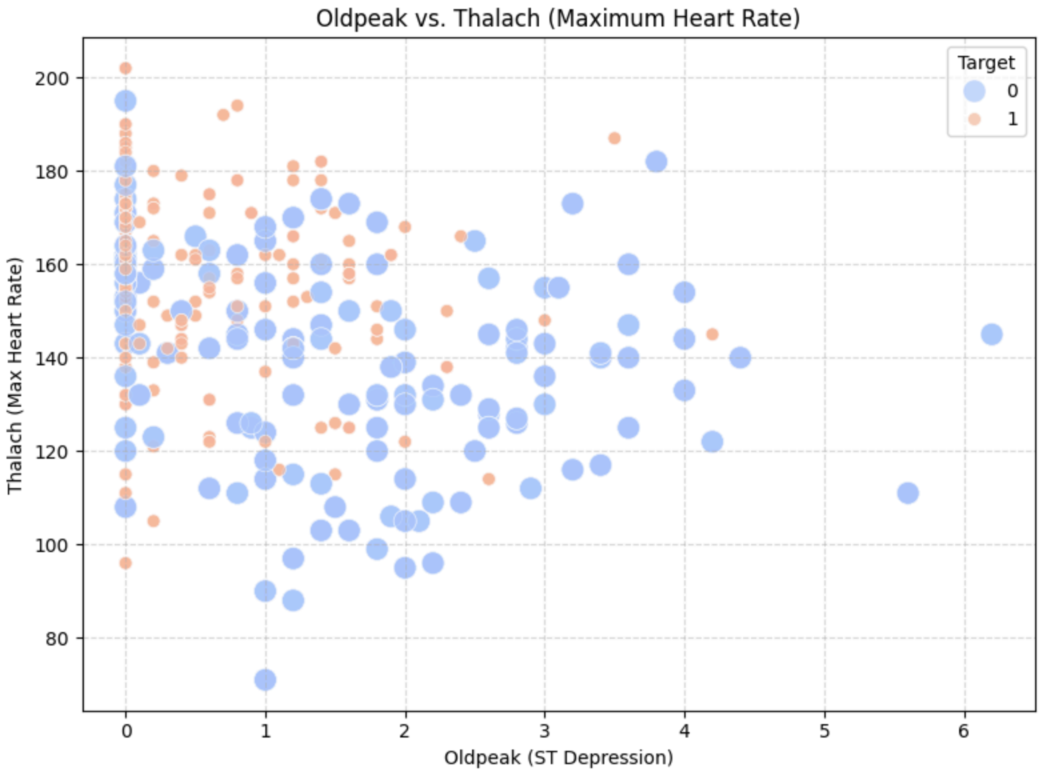
Boxplots visually confirmed the presence of these outliers, which highlight extreme values that may hold medical significance. For example, outliers in 'chol' and 'trestbps' may represent critical medical conditions, such as severe hypertension or hypercholesterolemia. While these extreme values could potentially impact model performance, retaining them is crucial as they provide important insights into rare but high-risk cases. To mitigate their influence on model training, normalization or robust scaling can be applied to ensure these values do not dominate feature scales.

Outlier Handling The boxplots revealed outliers in attributes like 'chol' (serum cholesterol) and 'trestbps' (resting blood pressure). These outliers were retained in the dataset to preserve medically significant cases, which could contribute valuable information about extreme risk factors for heart disease. However, preprocessing methods such as normalization will be employed to minimize their impact on the overall model performance while ensuring the model can learn from both typical and extreme cases.

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


plt.figure(figsize=(8, 6))
sns.scatterplot(
    x=heart_data['age'],
    y=heart_data['trestbps'],
    hue=heart_data['target'],
    palette='viridis',
    size=heart_data['target'],
    sizes=(50, 150),
    alpha=0.7
)

plt.title('Age vs Resting Blood Pressure (trestbps)')
plt.xlabel('Age')
plt.ylabel('Resting Blood Pressure (trestbps)')
plt.legend(title='Target', loc='upper right')
```

```
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.savefig('scatter_plot_age_vs_trestbps.png')
    plt.show()


    plt.figure(figsize=(8, 6))
    sns.scatterplot(
        x=heart_data['oldpeak'],
        y=heart_data['thalach'],
        hue=heart_data['target'],
        palette='coolwarm',
        size=heart_data['target'],
        sizes=(50, 150),
        alpha=0.7
    )

    plt.title('Oldpeak vs. Thalach (Maximum Heart Rate)')
    plt.xlabel('Oldpeak (ST Depression)')
    plt.ylabel('Thalach (Max Heart Rate)')
    plt.legend(title='Target', loc='upper right')
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.savefig('scatter_plot_oldpeak_vs_thalach.png')
    plt.show()
```




Scatter Plot 1: Age vs. Resting Blood Pressure (trestbps) This scatter plot illustrates the relationship between patients' ages and their resting blood pressure (trestbps). Each point represents a patient, color-coded by the target variable: blue indicates no heart disease (target = 0), and green indicates heart disease (target = 1).

Resting blood pressure values predominantly range between 120–140 mmHg, with a few higher outliers exceeding 180 mmHg. Patients with heart disease (target = 1) exhibit a broader spread in resting blood pressure values but tend to cluster in the lower range compared to those without heart disease. There is no strong linear correlation between age and resting blood pressure, as the points are widely dispersed.

Preprocessing Implications:

Outlier Handling: The plot identifies extreme resting blood pressure values (e.g., >180 mmHg). These should be normalized to prevent them from skewing model training. Feature Scaling: Since 'age' and 'trestbps' have different ranges, scaling is required to ensure both features contribute equally during modeling.

Scatter Plot 2: Oldpeak vs. Thalach (Maximum Heart Rate) This scatter plot visualizes the relationship between exercise-induced ST depression ('oldpeak') and maximum heart rate achieved ('thalach'). Points are color-coded by the target variable: blue represents no heart disease (target =

0), and orange represents heart disease (target = 1).

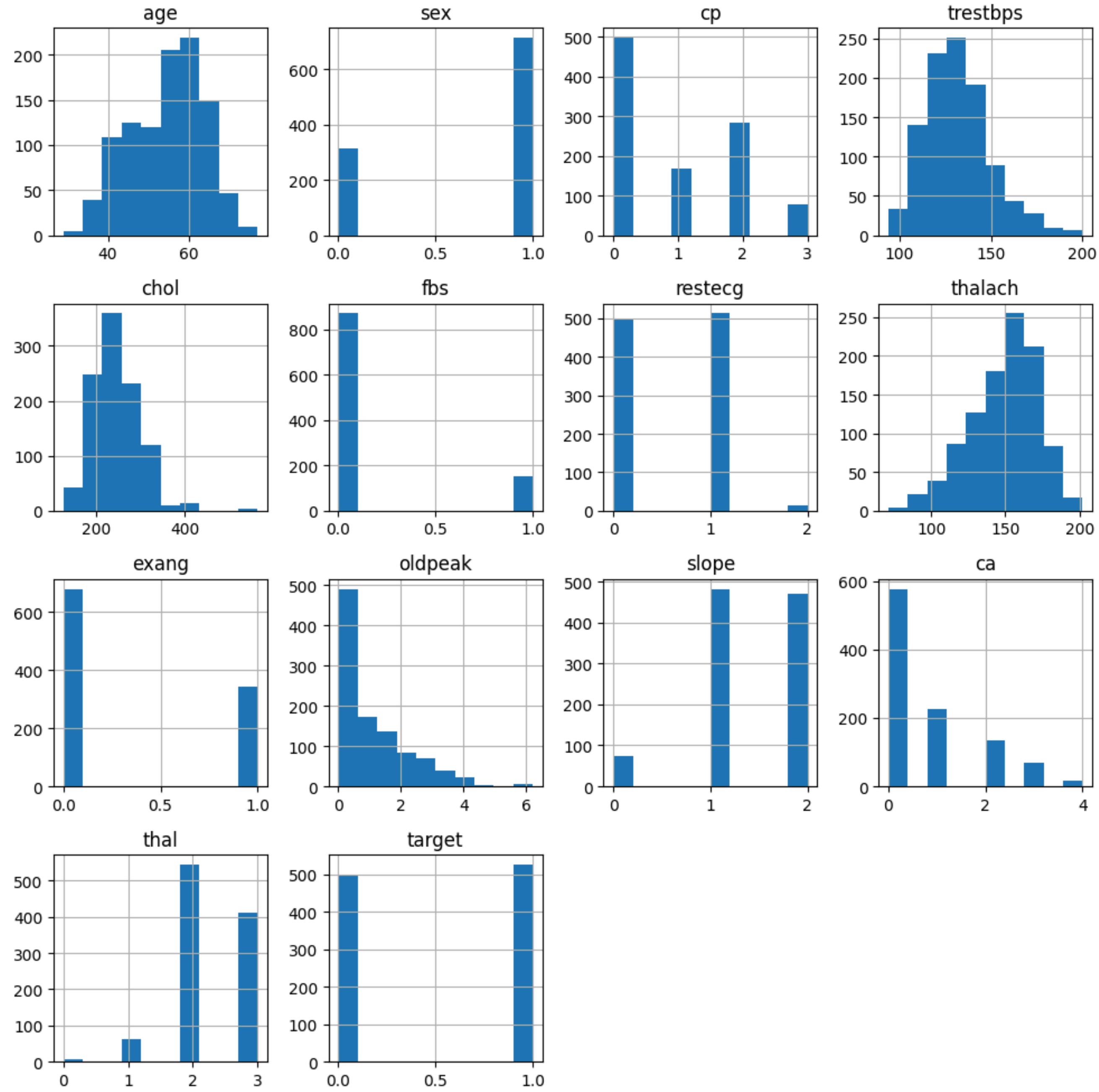
Individuals with heart disease (target = 1) tend to have higher 'oldpeak' values (indicating greater exercise-induced stress) and lower 'thalach' values (indicating reduced heart capacity under stress). Patients without heart disease (target = 0) cluster in regions with lower 'oldpeak' values and higher 'thalach' values. The plot reveals distinct clusters based on the target variable, but there is no strong linear correlation between 'oldpeak' and 'thalach'.

Preprocessing Implications:

Skewed Distribution: The 'oldpeak' feature has a right-skewed distribution with most values concentrated near zero. Feature Scaling: As 'thalach' and 'oldpeak' are measured on different scales, standardization or normalization is necessary to align their contributions during training.

```
print("Histogram to show distribution for all columns")
heart_data.hist(figsize=(10, 10))
plt.tight_layout()
plt.show()
```

```
Histogram to show distribution for all columns
```




From what we observed from the histograms:

Age: The distribution is fairly normal, with most individuals between 40 and 60 years of age. There is slight skewness toward older ages.

Sex: A binary variable, with a higher count for one category (likely males).

Chest Pain Type (cp): Most individuals fall in the 0 (asymptomatic) category, with fewer in higher categories, indicating lower chest pain severity.

Resting Blood Pressure (trestbps): A roughly normal distribution centered around 120–140 mmHg, with a few higher values suggesting outliers.

Serum Cholesterol (chol): The distribution is right-skewed, with most values between 200 and 300 mg/dL, along with some very high outliers.

Fasting Blood Sugar (fbs): A binary variable, with a significant majority in one category (likely 0, indicating fasting blood sugar below 120 mg/dL).

Maximum Heart Rate Achieved (thalach): This has a normal distribution, with most values between 120 and 160 beats per minute.

Exercise-Induced Angina (exang): Another binary variable, with a higher count for one category (likely 0, indicating no angina).

ST Depression (oldpeak): A right-skewed distribution, with most values near 0, indicating little or no depression. Outliers are present in higher values.

Slope: A categorical variable concentrated in two categories (1 and 2), representing different slopes of the ST segment during exercise.

Number of Major Vessels Colored by Fluoroscopy (ca): The distribution shows that most individuals have 0–1 major vessels affected, with fewer in higher categories.

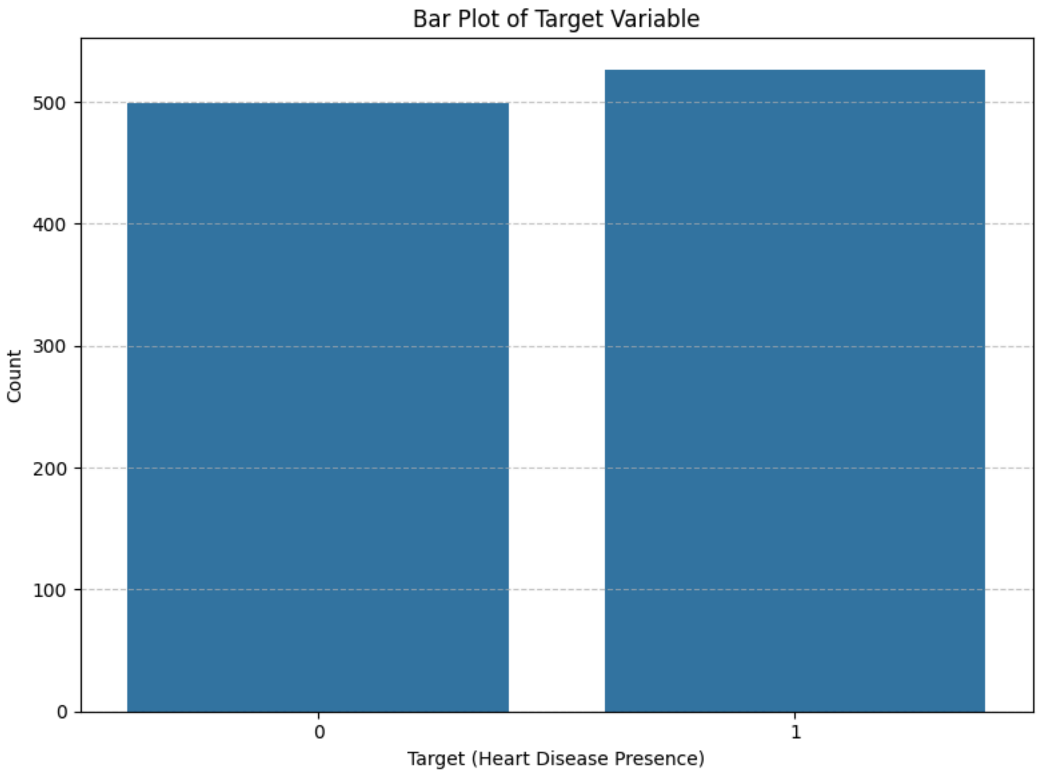
Thalassemia (thal): The majority fall into two categories, likely representing normal and fixed defects.

Target (target): A balanced distribution across the two classes (0 = no heart disease, 1 = heart disease).

Preprocessing Implications These observations indicate the need for:

Outlier Handling: For features like 'trestbps', 'chol', and 'oldpeak', where extreme values may impact model performance. Transformations: To handle skewed distributions, especially for 'chol' and 'oldpeak', which may benefit from logarithmic or Box-Cox transformations. Scaling: For numeric features with varying ranges, such as 'age', 'thalach', and 'chol', to ensure equal contribution to model performance. Encoding: For categorical features like 'cp', 'slope', and 'thal', which require one-hot encoding to properly represent their non-ordinal nature in the dataset.

```
plt.figure(figsize=(8, 6))
sns.countplot(x=heart_data['target'])
plt.title('Bar Plot of Target Variable')
plt.xlabel('Target (Heart Disease Presence)')
plt.ylabel('Count')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```




Bar Plot of the Target Variable The bar plot shows the distribution of the target variable (target), which indicates the presence (1) or absence (0) of heart disease in the dataset. Both classes have nearly equal frequencies.

The balanced distribution between the two classes ensures that the dataset is not biased toward one outcome. This balance is advantageous for training machine learning models, as it minimizes the risk of the model favoring one class over the other. Preprocessing Implications:

Since the target variable is balanced, there is no need for additional preprocessing steps.

## -General information about the dataSet:

the following code show's if the dataset have null attributes.

```
import pandas as pd
df = pd.read_csv('heart.csv')
print(df.isnull().values.any())
```

```
False
```

```
import pandas as pd
df = pd.read_csv('heart.csv')
print(df)
```

```
       age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0       52    1   0       125   212    0        1      168      0      1.0   
1       53    1   0       140   203    1        0      155      1      3.1   
2       70    1   0       145   174    0        1      125      1      2.6   
3       61    1   0       148   203    0        1      161      0      0.0   
4       62    0   0       138   294    1        1      106      0      1.9   
...    ...  ...  ..       ...   ...  ...      ...      ...    ...      ...   
1020    59    1   1       140   221    0        1      164      1      0.0   
1021    60    1   0       125   258    0        0      141      1      2.8   
1022    47    1   0       110   275    0        0      118      1      1.0   
1023    50    0   0       110   254    0        0      159      0      0.0   
1024    54    1   0       120   188    0        1      113      0      1.4   

      slope  ca  thal  target  
0         2   2     3       0  
1         0   0     3       0  
2         0   0     3       0  
3         2   1     3       0  
4         1   3     2       0  
...     ...  ..   ...     ...  
1020      2   0     2       1  
1021      1   1     3       0  
1022      1   1     2       0  
1023      2   0     2       1  
1024      1   1     3       0  

[1025 rows x 14 columns]
```

as we can see from the previous code, the data set contains 1025 rows and 14 coulmns.

```
import pandas as pd
df = pd.read_csv('heart.csv')
print(df.describe())
```

```
               age          sex           cp     trestbps        chol  \
count  1025.000000  1025.000000  1025.000000  1025.000000  1025.00000   
mean     54.434146     0.695610     0.942439   131.611707   246.00000   
std       9.072290     0.460373     1.029641    17.516718    51.59251   
min      29.000000     0.000000     0.000000    94.000000   126.00000   
25%      48.000000     0.000000     0.000000   120.000000   211.00000   
50%      56.000000     1.000000     1.000000   130.000000   240.00000   
75%      61.000000     1.000000     2.000000   140.000000   275.00000   
max      77.000000     1.000000     3.000000   200.000000   564.00000   

               fbs      restecg      thalach        exang      oldpeak  \
count  1025.000000  1025.000000  1025.000000  1025.000000  1025.000000   
mean      0.149268     0.529756   149.114146     0.336585     1.071512   
std       0.356527     0.527878    23.005724     0.472772     1.175053   
min       0.000000     0.000000    71.000000     0.000000     0.000000   
25%       0.000000     0.000000   132.000000     0.000000     0.000000   
50%       0.000000     1.000000   152.000000     0.000000     0.800000   
75%       0.000000     1.000000   166.000000     1.000000     1.800000   
max       1.000000     2.000000   202.000000     1.000000     6.200000   

             slope           ca         thal       target  
count  1025.000000  1025.000000  1025.000000  1025.000000  
mean      1.385366     0.754146     2.323902     0.513171  
std       0.617755     1.030798     0.620660     0.500070  
min       0.000000     0.000000     0.000000     0.000000  
25%       1.000000     0.000000     2.000000     0.000000  
50%       1.000000     0.000000     2.000000     1.000000  
75%       2.000000     1.000000     3.000000     1.000000  
max       2.000000     4.000000     3.000000     1.000000  
```

the following code show's the dataset attributes along with data types.

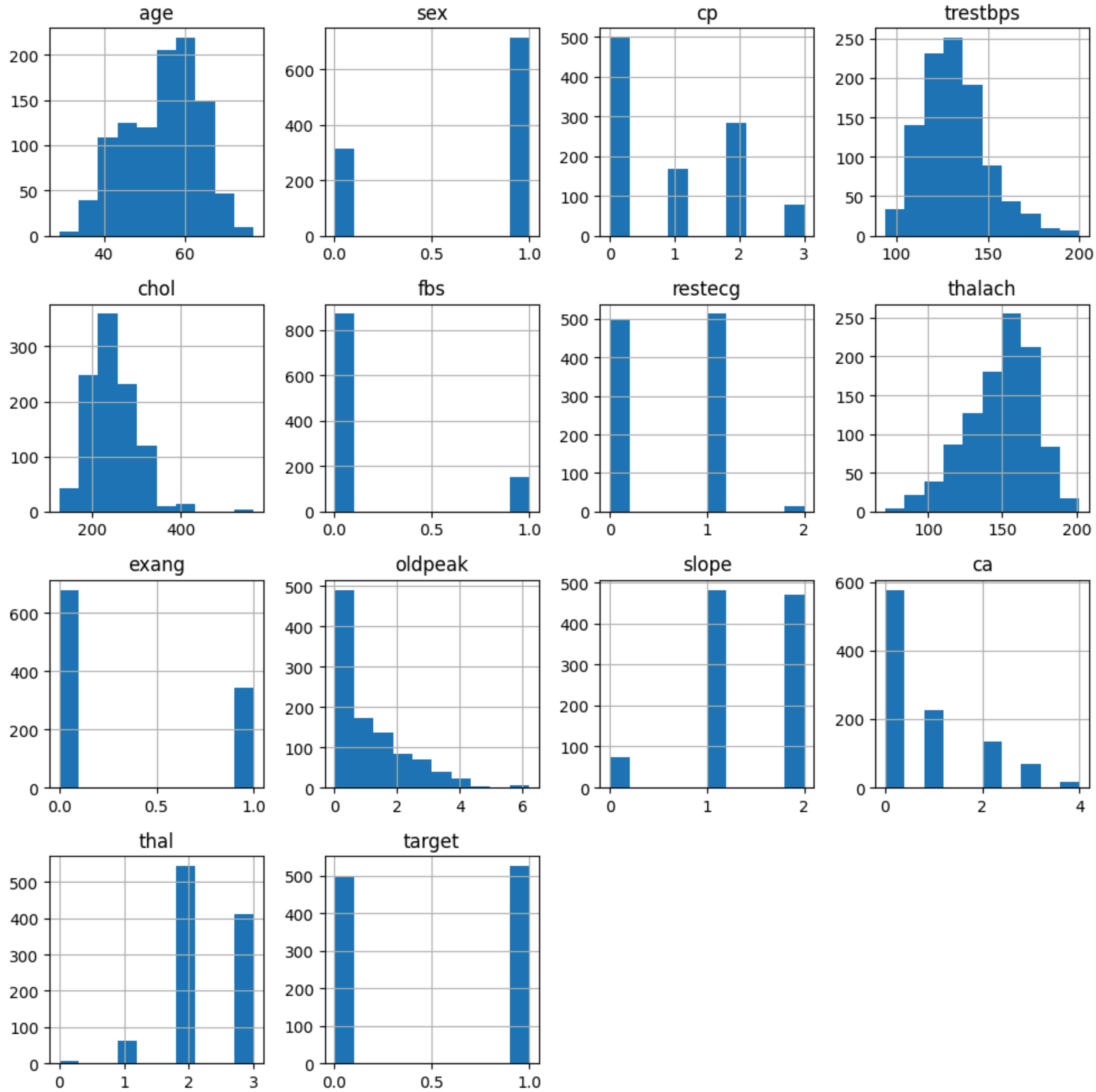
```
import pandas as pd
df = pd.read_csv('heart.csv')
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1025 non-null   int64  
 1   sex       1025 non-null   int64  
 2   cp        1025 non-null   int64  
 3   trestbps  1025 non-null   int64  
 4   chol      1025 non-null   int64  
 5   fbs       1025 non-null   int64  
 6   restecg   1025 non-null   int64  
 7   thalach   1025 non-null   int64  
 8   exang     1025 non-null   int64  
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64  
 11  ca        1025 non-null   int64  
 12  thal      1025 non-null   int64  
 13  target    1025 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
None
```

Here we displayed the distribution of all columns in the data set using histograms.

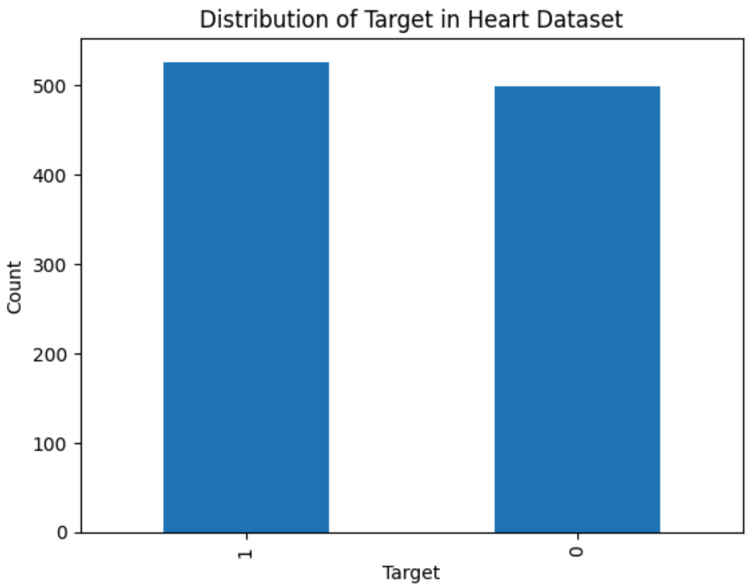
```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv("heart.csv")
print("Histogram to show distribution for all columns")
df.hist(figsize=(10, 10))
plt.tight_layout()
plt.show()
```

```
Histogram to show distribution for all columns
```




We then displayed the distribution of specifically the class label, the target column, usning a plot.

```
import matplotlib.pyplot as plt
import pandas as pd
df = pd.read_csv("heart.csv")
df['target'].value_counts().plot(kind='bar')
plt.title('Distribution of Target in Heart Dataset')
plt.xlabel('Target')
plt.ylabel('Count')
plt.show()
```




Here we displayed a summarization of the statstics as well as the variance for each column in the dataset

```
import pandas as pd
df = pd.read_csv("heart.csv")
print("Statistical Summary of Dataset")
print(df.describe())
print("----------------------------------------")
print("Variance of Dataset")
var_data = df.var()
print(var_data)
```

```
Statistical Summary of Dataset
               age          sex           cp     trestbps        chol  \
count  1025.000000  1025.000000  1025.000000  1025.000000  1025.00000
mean     54.434146     0.695610     0.942439   131.611707   246.00000
std       9.072290     0.460373     1.029641    17.516718    51.59251
min      29.000000     0.000000     0.000000    94.000000   126.00000
25%      48.000000     0.000000     0.000000   120.000000   211.00000
50%      56.000000     1.000000     1.000000   130.000000   240.00000
75%      61.000000     1.000000     2.000000   140.000000   275.00000
max      77.000000     1.000000     3.000000   200.000000   564.00000

               fbs      restecg      thalach        exang      oldpeak  \
count  1025.000000  1025.000000  1025.000000  1025.000000  1025.000000
mean      0.149268     0.529756   149.114146     0.336585     1.071512
std       0.356527     0.527878    23.005724     0.472772     1.175053
min       0.000000     0.000000    71.000000     0.000000     0.000000
25%       0.000000     0.000000   132.000000     0.000000     0.000000
50%       0.000000     1.000000   152.000000     0.000000     0.800000
75%       0.000000     1.000000   166.000000     1.000000     1.800000
max       1.000000     2.000000   202.000000     1.000000     6.200000

             slope           ca         thal       target
count  1025.000000  1025.000000  1025.000000  1025.000000
mean      1.385366     0.754146     2.323902     0.513171
std       0.617755     1.030798     0.620660     0.500070
min       0.000000     0.000000     0.000000     0.000000
25%       1.000000     0.000000     2.000000     0.000000
50%       1.000000     0.000000     2.000000     1.000000
75%       2.000000     1.000000     3.000000     1.000000
max       2.000000     4.000000     3.000000     1.000000
---------------------------------------
Variance of Dataset
age           82.306450
sex            0.211944
cp             1.060160
trestbps     306.835410
chol        2661.787109
fbs            0.127111
restecg        0.278655
thalach      529.263325
exang          0.223514
oldpeak        1.380750
slope          0.381622
ca             1.062544
thal           0.385219
target         0.250071
dtype: float64
```

## 4- Data pre-processing

We chose to apply data preprocessing because it is a critical step to ensure the dataset is clean, consistent, and suitable for machine learning algorithms, ultimately improving model accuracy and reliability. Data cleaning was performed to address missing values and outliers, as these can introduce bias and affect model performance. Transformation techniques like normalization were applied to standardize feature scales, ensuring attributes contribute equally during analysis, particularly in algorithms sensitive to magnitude. Discretization was employed to simplify continuous values into categories, reducing complexity and enhancing algorithm compatibility. Feature selection methods, including correlation analysis, variance thresholding, RFE, and L1 regularization, were used to remove redundant or irrelevant features and retain the most significant ones, improving model efficiency and interpretability. These steps collectively optimized the dataset for modeling by reducing noise, ensuring proper scaling, and enhancing predictive performance.

4.1 Data Cleaning

First we imported and loaded the data set, then displayed basic information and a sample of the data set(the first 5 rows).

```
# Import necessary libraries
import pandas as pd

# Load the dataset
df = pd.read_csv("heart.csv")

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

# Display the first five rows of the dataset
print("\nFirst five rows of the dataset:")
print(df.head())
```

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1025 non-null   int64
 1   sex       1025 non-null   int64
 2   cp        1025 non-null   int64
 3   trestbps  1025 non-null   int64
 4   chol      1025 non-null   int64
 5   fbs       1025 non-null   int64
 6   restecg   1025 non-null   int64
 7   thalach   1025 non-null   int64
 8   exang     1025 non-null   int64
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64
 11  ca        1025 non-null   int64
 12  thal      1025 non-null   int64
 13  target    1025 non-null   int64
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
None

First five rows of the dataset:
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   52    1   0       125   212    0        1      168      0      1.0      2
1   53    1   0       140   203    1        0      155      1      3.1      0
2   70    1   0       145   174    0        1      125      1      2.6      0
3   61    1   0       148   203    0        1      161      0      0.0      2
4   62    0   0       138   294    1        1      106      0      1.9      1

   ca  thal  target
0   2     3       0
1   0     3       0
2   0     3       0
3   1     3       0
4   3     2       0
```

- Checking for Missing Values

Why? Missing values can lead to biased results or errors during analysis

```
import pandas as pd
df = pd.read_csv("heart.csv")
```

```
df = pd.read_csv('heart.csv')
print("Missing values")
print(df.isna().sum())
```

```
Missing values
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

We checked for missing values across all columns and confirmed there were no missing values in our dataset.

- Outlier Detection and Removal

```
* Why? Outliers can distort statistical measures and affect the accuracy of machine learning models.
```

the following code display the outliers using z-score method:

```
from scipy.stats import zscore
import pandas as pd
df = pd.read_csv('heart.csv')
df = df.astype(float)
z_scores = zscore(df)
threshold = 3
outliers = df[(abs(z_scores)>threshold).any(axis=1)]
print(outliers)
```

```
685    1.0  3.0   3.0     0.0
686    1.0  0.0   0.0     0.0
688    0.0  2.0   3.0     0.0
734    1.0  0.0   0.0     0.0
743    1.0  4.0   3.0     1.0
749    1.0  4.0   3.0     1.0
831    1.0  4.0   3.0     1.0
833    0.0  0.0   3.0     0.0
889    1.0  3.0   3.0     0.0
893    1.0  0.0   0.0     0.0
958    2.0  1.0   2.0     1.0
970    2.0  4.0   2.0     1.0
993    1.0  4.0   3.0     0.0
996    1.0  2.0   3.0     0.0
```

Deleting the outliers:

```
from scipy.stats import zscore
import pandas as pd
df = pd.read_csv('heart.csv')
df = df.astype(float)
df_cleaned = df[(abs(z_scores) <= threshold).all(axis=1)]
print("\nDataSet after removing outliers:\n", df_cleaned)
```

```
DataSet after removing outliers:
        age  sex   cp  trestbps   chol  fbs  restecg  thalach  exang  oldpeak  \
0     52.0  1.0  0.0     125.0  212.0  0.0      1.0    168.0    0.0      1.0
1     53.0  1.0  0.0     140.0  203.0  1.0      0.0    155.0    1.0      3.1
2     70.0  1.0  0.0     145.0  174.0  0.0      1.0    125.0    1.0      2.6
3     61.0  1.0  0.0     148.0  203.0  0.0      1.0    161.0    0.0      0.0
4     62.0  0.0  0.0     138.0  294.0  1.0      1.0    106.0    0.0      1.9
...    ...  ...  ...       ...    ...  ...      ...      ...    ...      ...
1020  59.0  1.0  1.0     140.0  221.0  0.0      1.0    164.0    1.0      0.0
1021  60.0  1.0  0.0     125.0  258.0  0.0      0.0    141.0    1.0      2.8
1022  47.0  1.0  0.0     110.0  275.0  0.0      0.0    118.0    1.0      1.0
1023  50.0  0.0  0.0     110.0  254.0  0.0      0.0    159.0    0.0      0.0
1024  54.0  1.0  0.0     120.0  188.0  0.0      1.0    113.0    0.0      1.4

      slope   ca  thal  target
0       2.0  2.0   3.0     0.0
1       0.0  0.0   3.0     0.0
2       0.0  0.0   3.0     0.0
3       2.0  1.0   3.0     0.0
4       1.0  3.0   2.0     0.0
...     ...  ...   ...     ...
1020    2.0  0.0   2.0     1.0
1021    1.0  1.0   3.0     0.0
1022    1.0  1.0   2.0     0.0
1023    2.0  0.0   2.0     1.0
1024    1.0  1.0   3.0     0.0

[969 rows x 14 columns]
```

Using the Z-score method, we identified and removed rows containing outliers in numerical attributes by calculating the Z-scores for each column and comparing them against a threshold of 3, Rows where any value exceeds this threshold are considered outliers. This process ensures that rows with values outside the acceptable range were eliminated. The result is a cleaned dataset with only values within the acceptable range.

```
df_cleaned.to_csv('cleaned_dataset.csv', index=False)
print("Cleaned dataset saved as 'cleaned_dataset.csv'.")
```

```
Cleaned dataset saved as 'cleaned_dataset.csv'.
```

The cleaned dataset was saved as cleaned_dataset.csv. and has [969 rows x 14 columns]

4.2 Data Transformation

- Normalization

- Why? Normalization scales attributes to a uniform range, ensuring no single attribute disproportionately influences the results of algorithms, particularly those sensitive to distances, such as KNN or clustering.

- Techniques Applied:

1. Min-Max Scaling: Scaled [age, chol, oldpeak, thalach, trestbps] attributes to a range of 0-1.
2. Z-Score Normalization: Transformed [age, chol, oldpeak, thalach, trestbps] attributes to have a mean of 0 a
3. Decimal Scaling: Scaled [age, chol, oldpeak, thalach, trestbps] attributes by moving the decimal point, rec

Normalized the DataSet using min-max scaling:

```
import pandas as pd
df = pd.read_csv('cleaned_dataset.csv')
columns_to_normalize = ['age', 'chol', 'oldpeak', 'thalach', 'trestbps']
for column in columns_to_normalize:
    min_value = df[column].min()
    max_value = df[column].max()
    df[column] = (df[column] - min_value) / (max_value - min_value)
print(df)
```

```
          age  sex   cp  trestbps      chol  fbs  restecg   thalach  exang  \
0    0.479167  1.0  0.0  0.360465  0.320896  0.0      1.0  0.701754    0.0
1    0.500000  1.0  0.0  0.534884  0.287313  1.0      0.0  0.587719    1.0
2    0.854167  1.0  0.0  0.593023  0.179104  0.0      1.0  0.324561    1.0
3    0.666667  1.0  0.0  0.627907  0.287313  0.0      1.0  0.640351    0.0
4    0.687500  0.0  0.0  0.511628  0.626866  1.0      1.0  0.157895    0.0
..        ...  ...  ...       ...       ...  ...      ...       ...    ...
964  0.625000  1.0  1.0  0.534884  0.354478  0.0      1.0  0.666667    1.0
965  0.645833  1.0  0.0  0.360465  0.492537  0.0      0.0  0.464912    1.0
966  0.375000  1.0  0.0  0.186047  0.555970  0.0      0.0  0.263158    1.0
967  0.437500  0.0  0.0  0.186047  0.477612  0.0      0.0  0.622807    0.0
968  0.520833  1.0  0.0  0.302326  0.231343  0.0      1.0  0.219298    0.0

      oldpeak  slope   ca  thal  target
0    0.227273    2.0  2.0   3.0     0.0
1    0.704545    0.0  0.0   3.0     0.0
2    0.590909    0.0  0.0   3.0     0.0
3    0.000000    2.0  1.0   3.0     0.0
4    0.431818    1.0  3.0   2.0     0.0
..        ...    ...  ...   ...     ...
964  0.000000    2.0  0.0   2.0     1.0
965  0.636364    1.0  1.0   3.0     0.0
966  0.227273    1.0  1.0   2.0     0.0
967  0.000000    2.0  0.0   2.0     1.0
968  0.318182    1.0  1.0   3.0     0.0

[969 rows x 14 columns]
```

Normalized the DataSet using z-score:

```
import pandas as pd
df = pd.read_csv('cleaned_dataset.csv')
columns_to_normalize = ['age', 'chol', 'oldpeak', 'thalach', 'trestbps']
for column in columns_to_normalize:
    mean = df[column].mean()
    std_dev = df[column].std()
    df[column] = (df[column] - mean) / std_dev
print(df)
```

```
          age  sex   cp  trestbps      chol  fbs  restecg   thalach  exang  \
0   -0.266466  1.0  0.0 -0.353561 -0.703890  0.0      1.0  0.827388    0.0
1   -0.156263  1.0  0.0  0.531864 -0.899008  1.0      0.0  0.251935    1.0
2    1.717186  1.0  0.0  0.827006 -1.527724  0.0      1.0 -1.076034    1.0
3    0.725360  1.0  0.0  1.004091 -0.899008  0.0      1.0  0.517529    0.0
4    0.835563  0.0  0.0  0.413808  1.073857  1.0      1.0 -1.917082    0.0
..        ...  ...  ...       ...       ...  ...      ...       ...    ...
964  0.504954  1.0  1.0  0.531864 -0.508771  0.0      1.0  0.650326    1.0
965  0.615157  1.0  0.0 -0.353561  0.293383  0.0      0.0 -0.367784    1.0
966 -0.817480  1.0  0.0 -1.238986  0.661940  0.0      0.0 -1.385894    1.0
967 -0.486872  0.0  0.0 -1.238986  0.206663  0.0      0.0  0.428998    0.0
968 -0.046060  1.0  0.0 -0.648703 -1.224206  0.0      1.0 -1.607222    0.0

      oldpeak  slope   ca  thal  target
0   -0.031640    2.0  2.0   3.0     0.0
1    1.890266    0.0  0.0   3.0     0.0
2    1.432670    0.0  0.0   3.0     0.0
3   -0.946833    2.0  1.0   3.0     0.0
4    0.792034    1.0  3.0   2.0     0.0
..        ...    ...  ...   ...     ...
964 -0.946833    2.0  0.0   2.0     1.0
965  1.615708    1.0  1.0   3.0     0.0
966 -0.031640    1.0  1.0   2.0     0.0
967 -0.946833    2.0  0.0   2.0     1.0
968  0.334438    1.0  1.0   3.0     0.0

[969 rows x 14 columns]
```

Normalized the DataSet using decimal scaling:

```
import pandas as pd
df = pd.read_csv('cleaned_dataset.csv')
columns_to_normalize = ['age','chol','oldpeak','thalach','trestbps']
for column in columns_to_normalize:
    max_abs_value = df[column].abs().max()
    df[column] = df[column]/(10**len(str(int(max_abs_value ))))
    print(df)
```

```
967    2.0  0.0   2.0     1.0
968    1.0  1.0   3.0     0.0

[969 rows x 14 columns]
      age  sex   cp  trestbps   chol  fbs  restecg  thalach  exang  oldpeak  \
0    0.52  1.0  0.0     125.0  0.212  0.0      1.0    0.168    0.0     0.10
1    0.53  1.0  0.0     140.0  0.203  1.0      0.0    0.155    1.0     0.31
2    0.70  1.0  0.0     145.0  0.174  0.0      1.0    0.125    1.0     0.26
3    0.61  1.0  0.0     148.0  0.203  0.0      1.0    0.161    0.0     0.00
4    0.62  0.0  0.0     138.0  0.294  1.0      1.0    0.106    0.0     0.19
..    ...  ...  ...       ...    ...  ...      ...      ...    ...      ...
964  0.59  1.0  1.0     140.0  0.221  0.0      1.0    0.164    1.0     0.00
965  0.60  1.0  0.0     125.0  0.258  0.0      0.0    0.141    1.0     0.28
966  0.47  1.0  0.0     110.0  0.275  0.0      0.0    0.118    1.0     0.10
967  0.50  0.0  0.0     110.0  0.254  0.0      0.0    0.159    0.0     0.00
968  0.54  1.0  0.0     120.0  0.188  0.0      1.0    0.113    0.0     0.14

     slope   ca  thal  target
0      2.0  2.0   3.0     0.0
1      0.0  0.0   3.0     0.0
2      0.0  0.0   3.0     0.0
3      2.0  1.0   3.0     0.0
4      1.0  3.0   2.0     0.0
..     ...  ...   ...     ...
964    2.0  0.0   2.0     1.0
965    1.0  1.0   3.0     0.0
966    1.0  1.0   2.0     0.0
967    2.0  0.0   2.0     1.0
968    1.0  1.0   3.0     0.0

[969 rows x 14 columns]
      age  sex   cp  trestbps   chol  fbs  restecg  thalach  exang  oldpeak  \
0    0.52  1.0  0.0     0.125  0.212  0.0      1.0    0.168    0.0     0.10
1    0.53  1.0  0.0     0.140  0.203  1.0      0.0    0.155    1.0     0.31
2    0.70  1.0  0.0     0.145  0.174  0.0      1.0    0.125    1.0     0.26
3    0.61  1.0  0.0     0.148  0.203  0.0      1.0    0.161    0.0     0.00
4    0.62  0.0  0.0     0.138  0.294  1.0      1.0    0.106    0.0     0.19
..    ...  ...  ...       ...    ...  ...      ...      ...    ...      ...
964  0.59  1.0  1.0     0.140  0.221  0.0      1.0    0.164    1.0     0.00
965  0.60  1.0  0.0     0.125  0.258  0.0      0.0    0.141    1.0     0.28
966  0.47  1.0  0.0     0.110  0.275  0.0      0.0    0.118    1.0     0.10
967  0.50  0.0  0.0     0.110  0.254  0.0      0.0    0.159    0.0     0.00
968  0.54  1.0  0.0     0.120  0.188  0.0      1.0    0.113    0.0     0.14

     slope   ca  thal  target
0      2.0  2.0   3.0     0.0
1      0.0  0.0   3.0     0.0
2      0.0  0.0   3.0     0.0
3      2.0  1.0   3.0     0.0
4      1.0  3.0   2.0     0.0
..     ...  ...   ...     ...
964    2.0  0.0   2.0     1.0
965    1.0  1.0   3.0     0.0
966    1.0  1.0   2.0     0.0
967    2.0  0.0   2.0     1.0
968    1.0  1.0   3.0     0.0

[969 rows x 14 columns]
```

Min-max scaling is preferred because it scales values between 0 and 1, preserving relationships and improving performance in distance-based algorithms. This ensures equal contribution of all attributes, facilitating data handling and enhancing analysis effectivenes.

- Discretization method:

- Why? Converting continuous values into categorical bins simplifies the analysis and improves performance for certain algorithms.

```
import pandas as pd
data = pd.read_csv('cleaned_dataset.csv')
df = pd.DataFrame(data)
num_bins = 3
discretized_columns = []
for column in df.select_dtypes(include=['float64', 'int64']).columns:
    df['discretized_' + column] = pd.cut(df[column], bins=num_bins, labels=False)
```

```
    discretized_columns.append('discretized_' + column)
print(df[discretized_columns])
```

```
     discretized_age  discretized_sex  discretized_cp  discretized_trestbps  \
0                  1                2               0                     1
1                  1                2               0                     1
2                  2                2               0                     1
3                  1                2               0                     1
4                  2                0               0                     1
..               ...              ...             ...                   ...
964                1                2               0                     1
965                1                2               0                     1
966                1                2               0                     0
967                1                0               0                     0
968                1                2               0                     0

     discretized_chol  discretized_fbs  discretized_restecg  \
0                   0                0                    1
1                   0                2                    0
2                   0                0                    1
3                   0                0                    1
4                   1                2                    1
..                ...              ...                  ...
964                 1                0                    1
965                 1                0                    0
966                 1                0                    0
967                 1                0                    0
968                 0                0                    1

     discretized_thalach  discretized_exang  discretized_oldpeak  \
0                      2                  0                    0
1                      1                  2                    2
2                      0                  2                    1
3                      1                  0                    0
4                      0                  0                    1
..                   ...                ...                  ...
964                    1                  2                    0
965                    1                  2                    1
966                    0                  2                    0
967                    1                  0                    0
968                    0                  0                    0

     discretized_slope  discretized_ca  discretized_thal  discretized_target
0                    2               1                 2                   0
1                    0               0                 2                   0
2                    0               0                 2                   0
3                    2               0                 2                   0
4                    1               2                 1                   0
..                 ...             ...               ...                 ...
964                  2               0                 1                   2
965                  1               0                 2                   0
966                  1               0                 1                   0
967                  2               0                 1                   2
968                  1               0                 2                   0

[969 rows x 14 columns]
```

This method was implemented on numerical columns in the dataset by dividing their range into three bins, with each bin labeled numerically (0, 1, 2). This helps reduce the complexity of numerical features while retaining their relative distribution.

- Correlation Correlation Coefficient

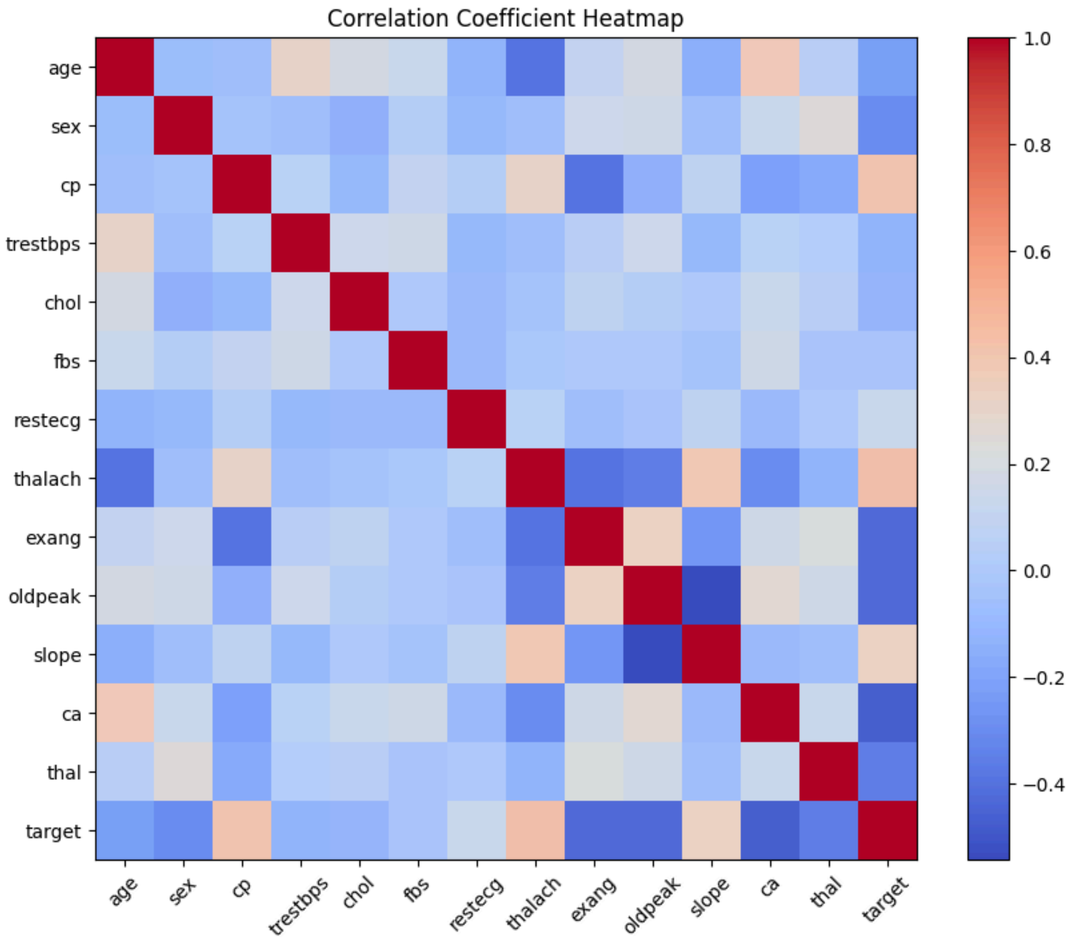
- Why? Highly correlated attributes introduce redundancy, which can hinder model performance.

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('cleaned_dataset.csv')
print("First few rows of the dataset:")
print(df.head())
print("\nDataset Information:")
print(df.info())
print("\nMissing Values:")
print(df.isnull().sum())
numeric_df = df.select_dtypes(include='number')
correlation_matrix = numeric_df.corr()
print("\nCorrelation Matrix:")
print(correlation_matrix)
plt.figure(figsize=(10, 8))
plt.imshow(correlation_matrix, cmap='coolwarm', interpolation='nearest')
plt.colorbar()
plt.xticks(range(len(correlation_matrix.columns)), correlation_matrix.columns, rotation=45)
plt.yticks(range(len(correlation_matrix.columns)), correlation_matrix.columns)
plt.title('Correlation Coefficient Heatmap')
plt.show()
```

```
plt.show()
```

```
First few rows of the dataset:
    age  sex   cp  trestbps   chol  fbs  restecg  thalach  exang  oldpeak  \
0  52.0  1.0  0.0     125.0  212.0  0.0      1.0    168.0    0.0      1.0
1  53.0  1.0  0.0     140.0  203.0  1.0      0.0    155.0    1.0      3.1
2  70.0  1.0  0.0     145.0  174.0  0.0      1.0    125.0    1.0      2.6
3  61.0  1.0  0.0     148.0  203.0  0.0      1.0    161.0    0.0      0.0
4  62.0  0.0  0.0     138.0  294.0  1.0      1.0    106.0    0.0      1.9

   slope   ca  thal  target
0    2.0  2.0   3.0     0.0
1    0.0  0.0   3.0     0.0
2    0.0  0.0   3.0     0.0
3    2.0  1.0   3.0     0.0
4    1.0  3.0   2.0     0.0

Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 969 entries, 0 to 968
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       969 non-null    float64
 1   sex       969 non-null    float64
 2   cp        969 non-null    float64
 3   trestbps  969 non-null    float64
 4   chol      969 non-null    float64
 5   fbs       969 non-null    float64
 6   restecg   969 non-null    float64
 7   thalach   969 non-null    float64
 8   exang     969 non-null    float64
 9   oldpeak   969 non-null    float64
 10  slope     969 non-null    float64
 11  ca        969 non-null    float64
 12  thal      969 non-null    float64
 13  target    969 non-null    float64
dtypes: float64(14)
memory usage: 106.1 KB
None

Missing Values:
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64

Correlation Matrix:
               age       sex        cp  trestbps      chol       fbs  \
age       1.000000 -0.077902 -0.059286  0.297992  0.183075  0.133062
sex      -0.077902  1.000000 -0.047771 -0.057017 -0.138614  0.041548
cp       -0.059286 -0.047771  1.000000  0.061996 -0.097779  0.102924
trestbps  0.297992 -0.057017  0.061996  1.000000  0.142566  0.167518
chol      0.183075 -0.138614 -0.097779  0.142566  1.000000  0.013029
fbs       0.133062  0.041548  0.102924  0.167518  0.013029  1.000000
restecg  -0.120585 -0.107043  0.038883 -0.099957 -0.095590 -0.094705
thalach  -0.387504 -0.057927  0.297609 -0.064813 -0.041939 -0.009113
exang     0.101872  0.145631 -0.390493  0.054314  0.085599  0.013031
oldpeak   0.191232  0.155104 -0.135320  0.144384  0.038335  0.005646
slope    -0.153518 -0.051484  0.090084 -0.097033  0.001309 -0.045372
ca        0.370247  0.130724 -0.216786  0.060339  0.117465  0.156300
thal      0.058987  0.235699 -0.157708  0.023259  0.051000 -0.020786
target   -0.227225 -0.303739  0.408999 -0.114757 -0.112342 -0.023629

           restecg   thalach     exang   oldpeak     slope        ca  \
age      -0.120585 -0.387504  0.101872  0.191232 -0.153518  0.370247
sex      -0.107043 -0.057927  0.145631  0.155104 -0.051484  0.130724
cp        0.038883  0.297609 -0.390493 -0.135320  0.090084 -0.216786
trestbps -0.099957 -0.064813  0.054314  0.144384 -0.097033  0.060339
chol     -0.095590 -0.041939  0.085599  0.038335  0.001309  0.117465
fbs      -0.094705 -0.009113  0.013031  0.005646 -0.045372  0.156300
restecg   1.000000  0.061232 -0.066541 -0.028290  0.081653 -0.095432
thalach   0.061232  1.000000 -0.395719 -0.357793  0.386290 -0.299275
exang    -0.066541 -0.395719  1.000000  0.319344 -0.248610  0.153337
oldpeak  -0.028290 -0.357793  0.319344  1.000000 -0.542464  0.267879
slope     0.081653  0.386290 -0.248610 -0.542464  1.000000 -0.083869
ca       -0.095432 -0.299275  0.153337  0.267879 -0.083869  1.000000
thal      0.004146 -0.120972  0.216089  0.165459 -0.058630  0.126366
target    0.127580  0.429920 -0.429825 -0.431854  0.322791 -0.466639

              thal    target
```

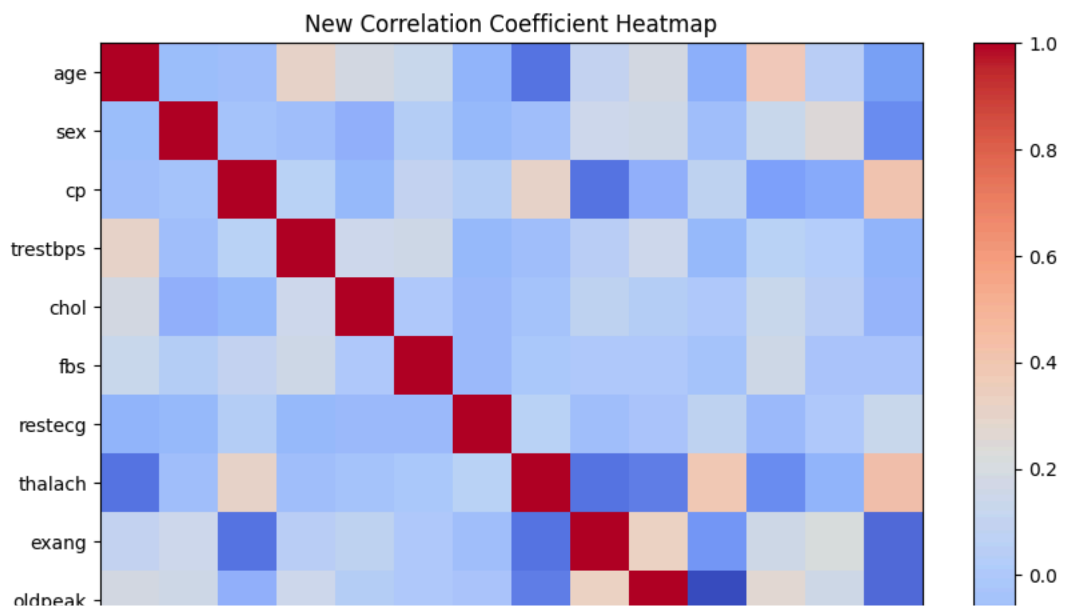
```
age       0.058987 -0.227225
sex       0.235699 -0.303739
cp       -0.157708  0.408999
trestbps  0.023259 -0.114757
chol      0.051000 -0.112342
fbs      -0.020786 -0.023629
restecg   0.004146  0.127580
thalach  -0.120972  0.429920
exang     0.216089 -0.429825
oldpeak   0.165459 -0.431854
slope    -0.058630  0.322791
ca        0.126366 -0.466639
thal      1.000000 -0.352502
target   -0.352502  1.000000
```

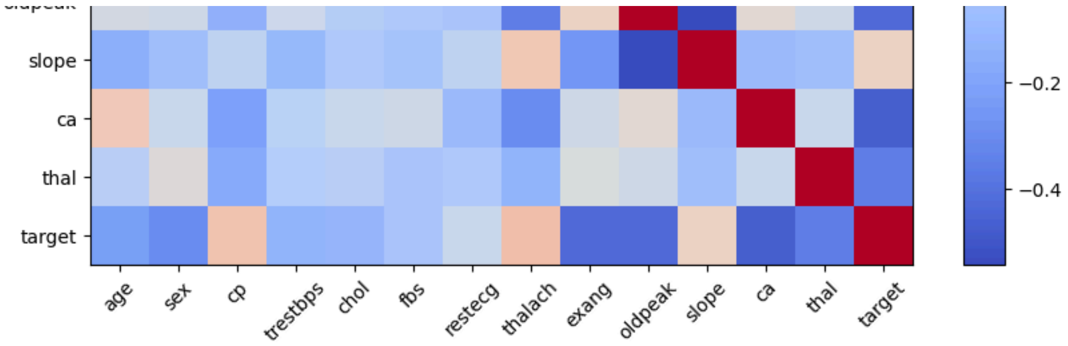


Remove attributes with 0.75 correlation or higher:

```
threshold = 0.75
high_correlation = correlation_matrix[abs(correlation_matrix) > threshold]
columns_to_drop = set()
for i in range(len(high_correlation.columns)):
    for j in range(i):
        if abs(high_correlation.iloc[i, j]) >= threshold:
            colname = high_correlation.columns[i]
            columns_to_drop.add(colname)
df_reduced = df.drop(columns=columns_to_drop)
print("\nDataFrame after removing highly correlated attributes:")
print(df_reduced.head())
new_correlation_matrix = df_reduced.corr()
print("\nNew Correlation Matrix:")
print(new_correlation_matrix)
plt.figure(figsize=(10, 8))
plt.imshow(new_correlation_matrix, cmap='coolwarm', interpolation='nearest')
plt.colorbar()
plt.xticks(range(len(new_correlation_matrix.columns)), new_correlation_matrix.columns, rotation=45)
plt.yticks(range(len(new_correlation_matrix.columns)), new_correlation_matrix.columns)
plt.title('New Correlation Coefficient Heatmap')
plt.show()
```

```
DataFrame after removing highly correlated attributes:
    age  sex   cp  trestbps   chol  fbs  restecg  thalach  exang  oldpeak  \
0  52.0  1.0  0.0     125.0  212.0  0.0      1.0    168.0    0.0      1.0
1  53.0  1.0  0.0     140.0  203.0  1.0      0.0    155.0    1.0      3.1
2  70.0  1.0  0.0     145.0  174.0  0.0      1.0    125.0    1.0      2.6
3  61.0  1.0  0.0     148.0  203.0  0.0      1.0    161.0    0.0      0.0
4  62.0  0.0  0.0     138.0  294.0  1.0      1.0    106.0    0.0      1.9

   slope   ca  thal  target
0    2.0  2.0   3.0     0.0
1    0.0  0.0   3.0     0.0
2    0.0  0.0   3.0     0.0
3    2.0  1.0   3.0     0.0
4    1.0  3.0   2.0     0.0

New Correlation Matrix:
               age       sex        cp  trestbps      chol       fbs  \
age       1.000000 -0.077902 -0.059286  0.297992  0.183075  0.133062
sex      -0.077902  1.000000 -0.047771 -0.057017 -0.138614  0.041548
cp       -0.059286 -0.047771  1.000000  0.061996 -0.097779  0.102924
trestbps  0.297992 -0.057017  0.061996  1.000000  0.142566  0.167518
chol      0.183075 -0.138614 -0.097779  0.142566  1.000000  0.013029
fbs       0.133062  0.041548  0.102924  0.167518  0.013029  1.000000
restecg  -0.120585 -0.107043  0.038883 -0.099957 -0.095590 -0.094705
thalach  -0.387504 -0.057927  0.297609 -0.064813 -0.041939 -0.009113
exang     0.101872  0.145631 -0.390493  0.054314  0.085599  0.013031
oldpeak   0.191232  0.155104 -0.135320  0.144384  0.038335  0.005646
slope    -0.153518 -0.051484  0.090084 -0.097033  0.001309 -0.045372
ca        0.370247  0.130724 -0.216786  0.060339  0.117465  0.156300
thal      0.058987  0.235699 -0.157708  0.023259  0.051000 -0.020786
target   -0.227225 -0.303739  0.408999 -0.114757 -0.112342 -0.023629

           restecg   thalach     exang   oldpeak     slope        ca  \
age      -0.120585 -0.387504  0.101872  0.191232 -0.153518  0.370247
sex      -0.107043 -0.057927  0.145631  0.155104 -0.051484  0.130724
cp        0.038883  0.297609 -0.390493 -0.135320  0.090084 -0.216786
trestbps -0.099957 -0.064813  0.054314  0.144384 -0.097033  0.060339
chol     -0.095590 -0.041939  0.085599  0.038335  0.001309  0.117465
fbs      -0.094705 -0.009113  0.013031  0.005646 -0.045372  0.156300
restecg   1.000000  0.061232 -0.066541 -0.028290  0.081653 -0.095432
thalach   0.061232  1.000000 -0.395719 -0.357793  0.386290 -0.299275
exang    -0.066541 -0.395719  1.000000  0.319344 -0.248610  0.153337
oldpeak  -0.028290 -0.357793  0.319344  1.000000 -0.542464  0.267879
slope     0.081653  0.386290 -0.248610 -0.542464  1.000000 -0.083869
ca       -0.095432 -0.299275  0.153337  0.267879 -0.083869  1.000000
thal      0.004146 -0.120972  0.216089  0.165459 -0.058630  0.126366
target    0.127580  0.429920 -0.429825 -0.431854  0.322791 -0.466639

              thal    target
age       0.058987 -0.227225
sex       0.235699 -0.303739
cp       -0.157708  0.408999
trestbps  0.023259 -0.114757
chol      0.051000 -0.112342
fbs      -0.020786 -0.023629
restecg   0.004146  0.127580
thalach  -0.120972  0.429920
exang     0.216089 -0.429825
oldpeak   0.165459 -0.431854
slope    -0.058630  0.322791
ca        0.126366 -0.466639
thal      1.000000 -0.352502
target   -0.352502  1.000000
```






The correlation coefficient was applied to the numeric attributes of the dataset to identify and remove highly correlated attributes (with a threshold of 0.75), aiming to reduce redundancy, which can affect model performance. However, no attributes had a correlation coefficient exceeding the threshold, so no columns were removed, and the dataset remained unchanged after the process,This means that the dataset does not suffer from significant redundancy among its features.

Save the new dataset:

```
df_cleaned.to_csv('correlated_dataset.csv', index=False)
print("dataset after removing attributes with 0.75 correlation or higher saved as 'correlated_dataset.csv'.")
```

```
dataset after removing attributes with 0.75 correlation or higher saved as 'correlated_dataset.csv'.
```

4.3 Feature Selection

Filter method using Correlation coefficient based feature selection:

```
import pandas as pd
df = pd.read_csv('correlated_dataset.csv')
X = df.drop(columns=['target'])
y = df['target']
correlation = df.corr()['target'].abs()
top_features = correlation.nlargest(3).index
selected_features = top_features[1:]
print("Selected Features:", selected_features.tolist())
```

```
Selected Features: ['ca', 'oldpeak']
```

We used the Correlation method to identify features with the strongest linear relationship with the target variable, aiming to select those most likely to impact the outcome directly. The features selected based on correlation were ['ca', 'oldpeak'], indicating they have the highest absolute correlation values with the target, suggesting they are important for predicting the outcome.

Filter method using variance threshold:

We applied a variance threshold to identify and remove features with very low variance, as these attributes do not contribute significant information to the dataset and may negatively impact model performance.

```
import pandas as pd
df = pd.read_csv('correlated_dataset.csv')
X = df.drop(columns=['target'])
y = df['target']
numeric_X = X.select_dtypes(include='number')
variances = numeric_X.var()
print("Original Variances of all numeric features:")
print(variances)
threshold = 0.2
selected_variances = variances[variances > threshold]
print("\nSelected Features after Variance threshold:")
print(selected_variances)
```

```
Original Variances of all numeric features:
age           82.340628
sex            0.209511
cp             1.071715
trestbps     286.997827
chol        2127.590110
fbs            0.122997
restecg        0.280192
thalach      510.347871
```

```
exang          0.223812
oldpeak        1.193917
slope          0.370080
ca             0.864116
thal           0.351860
dtype: float64

Selected Features after Variance threshold:
age           82.340628
sex            0.209511
cp             1.071715
trestbps     286.997827
chol        2127.590110
restecg        0.280192
thalach      510.347871
exang          0.223812
oldpeak        1.193917
slope          0.370080
ca             0.864116
thal           0.351860
dtype: float64
```

This method was applied to the numerical attributes in the dataset. Attributes with variance below the threshold of 0.2 were removed, as they provide little to no variability and are less likely to influence the target variable. In this case, the attribute fbs was removed due to its variance being below the threshold, while all other features were retained for further analysis.

- Wrapper and Embedded Methods

Wrapper method using recursive feature elimination:

why? applied RFE to identify the most relevant features for predicting the target variable.

```
import pandas as pd
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
df = pd.read_csv('correlated_dataset.csv')
X = df.drop(columns=['target'])
y = df['target']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
model = LogisticRegression(max_iter=1000)
rfe = RFE(estimator=model, n_features_to_select=2)
rfe.fit(X_scaled, y)
selected_features = X.columns[rfe.support_]
print("Selected Features using RFE:", selected_features.tolist())
```

```
Selected Features using RFE: ['oldpeak', 'ca']
```

the dataset was preprocessed by separating the target column (target) from the features. Then, a logistic regression model with a maximum of 1000 iterations was used as the estimator for RFE. RFE was configured to select the top 2 features by iteratively fitting the model and removing the least important attributes based on their contribution to the model's performance. After applying this process, the selected features were oldpeak and ca, while the remaining features were eliminated.

Embedded method using L1 regularization:

why? Penalized non-important features to automatically reduce dimensionality to identify and retain the most important features in my dataset while reducing overfitting and simplifying the model.

```
import pandas as pd
from sklearn.linear_model import Lasso
df = pd.read_csv('correlated_dataset.csv')
X = df.drop(columns=['target'])
y = df['target']
alpha_value = 0.1
model = Lasso(alpha=alpha_value)
model.fit(X, y)
selected_features = X.columns[model.coef_ != 0]
print("Selected Features using L1 Regularization (Lasso):", selected_features.tolist())
```

```
Selected Features using L1 Regularization (Lasso): ['cp', 'trestbps', 'chol', 'thalach', 'oldpeak', 'ca']
```

In using the **RFE** and **Lasso** methods for feature selection, we observed that RFE selected only two features, namely **['oldpeak', 'ca']**, indicating its focus on identifying the most important features to simplify the model. In contrast, Lasso selected a larger set of features **['cp', 'trestbps', 'chol', 'thalach', 'oldpeak', 'ca']**, suggesting it allows for the inclusion of multiple features that may be useful. The difference in results reflects that RFE relies on the overall model performance to determine the features, while Lasso employs regularization techniques to eliminate the influence of less important features. Ultimately, if the goal is to simplify the model, RFE is the more suitable choice, while if the goal is to retain a greater number of potentially useful features, Lasso is the better option. Therefore, both methods can be used together to achieve a balance between accuracy and model complexity

Applying data preprocessing to our dataset ensured the data was clean, consistent, and optimized for analysis. Missing values were confirmed absent, and outliers were removed using Z-scores to enhance reliability. Normalization techniques like Min-Max Scaling and Z-Score ensured consistent feature scaling, critical for algorithms sensitive to magnitudes. Discretization simplified numerical features into bins, reducing complexity. Feature selection reduced redundancy and improved efficiency; for instance, fbs was removed for low variance, while RFE and L1 Regularization identified key features like oldpeak, ca, and cp. These steps improved data quality, reduced noise, and highlighted significant attributes for heart disease prediction.

The Dataset before Preprocessing:

```
import pandas as pd
df = pd.read_csv('heart.csv')
with pd.option_context('display.max_rows',None,'display.max_columns', None):
 print(df)
```

```
967       2  0    2     1
968       0  0    3     0
969       2  0    2     1
970       2  4    2     1
971       2  0    3     1
972       1  0    1     1
973       1  0    2     1
974       2  0    3     1
975       1  0    3     0
976       1  2    2     0
977       2  0    2     1
978       2  1    2     0
979       1  3    3     0
980       1  0    1     1
981       1  0    3     0
982       2  2    2     1
983       1  1    3     1
984       1  0    3     1
985       1  3    3     1
986       1  0    2     0
987       2  1    2     0
988       1  2    3     0
989       2  2    2     1
990       2  0    2     1
991       2  2    3     0
992       2  0    2     1
993       1  4    3     0
994       1  1    3     0
995       2  0    3     1
996       1  2    3     0
997       1  1    3     0
998       1  0    1     0
999       1  2    3     0
1000      1  2    1     0
1001      2  0    2     1
1002      2  1    2     0
1003      2  3    3     1
1004      2  1    2     1
1005      1  1    3     0
1006      2  0    2     1
1007      1  0    3     1
1008      2  0    2     1
1009      2  0    3     0
1010      2  0    3     0
1011      2  0    2     1
1012      0  0    3     0
1013      0  3    1     0
1014      1  0    2     1
1015      1  3    3     0
1016      1  1    2     0
1017      1  2    3     0
1018      2  0    3     0
1019      2  0    2     1
1020      2  0    2     1
1021      1  1    3     0
1022      1  1    2     0
1023      2  0    2     1
1024      1  1    3     0
```

The Dataset after Preprocessing:

```
import pandas as pd
df = pd.read_csv('correlated_dataset.csv')
with pd.option_context('display.max_rows',None,'display.max_columns', None):
 print(df)
```

```
911    1.0  0.0   2.0     1.0
912    1.0  0.0   2.0     1.0
913    1.0  2.0   3.0     0.0
914    2.0  0.0   2.0     1.0
915    0.0  0.0   3.0     0.0
916    2.0  0.0   2.0     1.0
917    2.0  0.0   3.0     1.0
918    1.0  0.0   1.0     1.0
919    1.0  0.0   2.0     1.0
920    2.0  0.0   3.0     1.0
921    1.0  0.0   3.0     0.0
922    1.0  2.0   2.0     0.0
923    2.0  0.0   2.0     1.0
924    2.0  1.0   2.0     0.0
925    1.0  3.0   3.0     0.0
926    1.0  0.0   1.0     1.0
927    1.0  0.0   3.0     0.0
928    2.0  2.0   2.0     1.0
929    1.0  1.0   3.0     1.0
930    1.0  0.0   3.0     1.0
931    1.0  3.0   3.0     1.0
932    1.0  0.0   2.0     0.0
933    2.0  1.0   2.0     0.0
934    1.0  2.0   3.0     0.0
935    2.0  2.0   2.0     1.0
936    2.0  0.0   2.0     1.0
937    2.0  2.0   3.0     0.0
938    2.0  0.0   2.0     1.0
939    1.0  1.0   3.0     0.0
940    2.0  0.0   3.0     1.0
941    1.0  1.0   3.0     0.0
942    1.0  0.0   1.0     0.0
943    1.0  2.0   3.0     0.0
944    1.0  2.0   1.0     0.0
945    2.0  0.0   2.0     1.0
946    2.0  1.0   2.0     0.0
947    2.0  3.0   3.0     1.0
948    2.0  1.0   2.0     1.0
949    1.0  1.0   3.0     0.0
950    2.0  0.0   2.0     1.0
951    1.0  0.0   3.0     1.0
952    2.0  0.0   2.0     1.0
953    2.0  0.0   3.0     0.0
954    2.0  0.0   3.0     0.0
955    2.0  0.0   2.0     1.0
956    0.0  0.0   3.0     0.0
957    0.0  3.0   1.0     0.0
958    1.0  0.0   2.0     1.0
959    1.0  3.0   3.0     0.0
960    1.0  1.0   2.0     0.0
961    1.0  2.0   3.0     0.0
962    2.0  0.0   3.0     0.0
963    2.0  0.0   2.0     1.0
964    2.0  0.0   2.0     1.0
965    1.0  1.0   3.0     0.0
966    1.0  1.0   2.0     0.0
967    2.0  0.0   2.0     1.0
968    1.0  1.0   3.0     0.0
```

## 5-Data Mining Technique

The project employs the following data mining techniques:

- ### Classification: [1,2,3]

  The classification task involves predicting whether a patient has heart disease based on medical attributes. This is a supervised learning approach, where the model is trained on labeled data. The following techniques and steps are used:

- Algorithms:
- Logistic Regression: Suitable for binary classification tasks. It estimates the probability of an instance belonging to a particular class based on a logistic function.

- Decision Trees: A flowchart-like structure where each internal node represents a test on an attribute, each branch represents the test outcome, and each leaf node represents a class label. It is interpretable and effective for smaller datasets.
- Support Vector Machines (SVM): Creates a hyperplane in a multidimensional space to separate classes. Effective for high-dimensional data and when the decision boundary is non-linear.

These algorithms were selected for their interpretability, ability to handle structured data, and effectiveness in binary classification tasks.

- Python Implementation:
- Packages: scikit-learn will be the primary library.

The models were implemented using the scikit-learn library in Python. Parameters were optimized through techniques like cross-validation to ensure the best model performance.

The dataset was split into training and testing subsets, and hyperparameter tuning was performed to enhance model accuracy and robustness.

- Methods:

1. LogisticRegression for logistic regression.
2. DecisionTreeClassifier for decision tree implementation.
3. SVC for support vector classification.

- Evaluation Metrics:
  - Accuracy: Measures the overall correctness of the model.
  - Precision: Evaluates how many predicted positives are true positives.
  - Recall (Sensitivity): Measures how many actual positives are correctly predicted.
  - F1-Score: Balances precision and recall.
- Data Preprocessing:
  - Features were standardized using StandardScaler for algorithms like SVM.
  - Missing values were imputed to ensure data completeness.
  - Feature importance analysis was performed to identify key attributes contributing to predictions.

---

- ## Clustering: [4,5]

  Clustering is an unsupervised learning technique used to group patients with similar characteristics into clusters. This can help identify subgroups within the data and assist in predictive analysis.
- Algorithms:
- K-Means Clustering: Divides the data into k clusters based on feature similarity, minimizing intra-cluster variance.
- DBSCAN (Density-Based Spatial Clustering of Applications with Noise): Groups data points based on density, handling noise and identifying clusters of varying shapes.

K-means clustering was chosen for its efficiency and effectiveness in grouping data points based on similarity. It is particularly well-suited for structured medical datasets with numerical attributes.

- Python Implementation:
- Packages: scikit-learn will be used.

The clustering algorithm was implemented using scikit-learn, and the number of clusters was determined through the elbow method and silhouette analysis.

- Methods:

1. KMeans for K-Means clustering.
2. DBSCAN for density-based clustering.

- Evaluation Metrics:
  - Silhouette Score: Assesses the compactness and separation of clusters.
  - Inertia (WCSS): Measures intra-cluster variance for K-Means.
  - Core Points vs Noise Points: Used to evaluate DBSCAN's handling of noise.
- Data Preprocessing:
  - Features were scaled using StandardScaler to improve clustering performance.
  - Outliers were handled effectively using DBSCAN.

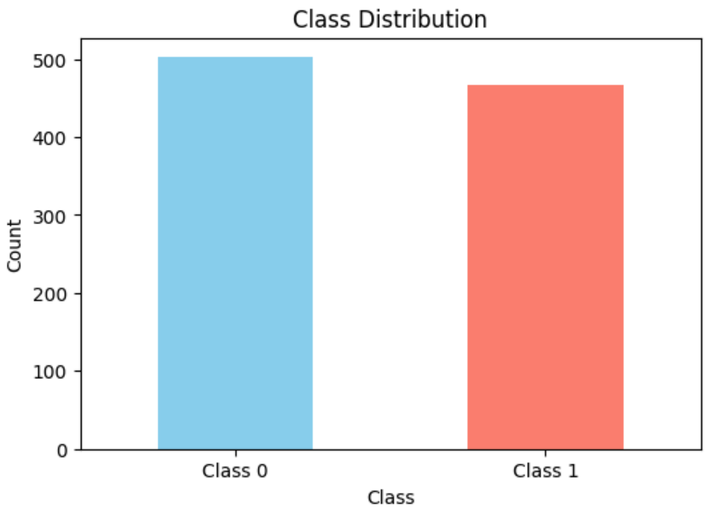
---

By combining these techniques, the study achieves the dual goals of predicting heart disease at the individual level (classification) and uncovering population-level patterns (clustering). This integration supports both preventive and personalized healthcare strategies.

## - Classification

We checked the class distribution of the target variable to ensure the dataset was balanced. This involved plotting the counts of each class and calculating the imbalance ratio. The dataset showed a near-equal distribution of the two classes with an imbalance ratio of 0.93, so no further balancing was needed.

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("correlated_dataset.csv")
X = df.drop(columns=["target"])
y = df["target"]

plt.figure(figsize=(6, 4))
y.value_counts().plot(kind='bar', color=['skyblue', 'salmon'])
plt.title("Class Distribution")
plt.xlabel("Class")
plt.ylabel("Count")
plt.xticks(ticks=[0, 1], labels=['Class 0', 'Class 1'], rotation=0)
plt.show()

class_counts = y.value_counts()
print("Class Counts:\n", class_counts)

imbalance_ratio = min(class_counts) / max(class_counts)
print(f"Imbalance Ratio: {imbalance_ratio:.2f}")

if imbalance_ratio < 0.5:
    print("Dataset is imbalanced.")
else:
    print("Dataset is balanced.")
```




```
Class Counts:
 target
1.0    502
0.0    467
Name: count, dtype: int64
Imbalance Ratio: 0.93
Dataset is balanced.
```

Next we split the dataset into training and testing sets to evaluate the model's performance on unseen data. We will use 3 different ratios (70/30, 80/20, 90/10) to analyze how varying the size of the training data affects model accuracy. This step ensures the model is tested on data it hasn't seen before, providing an unbiased evaluation.

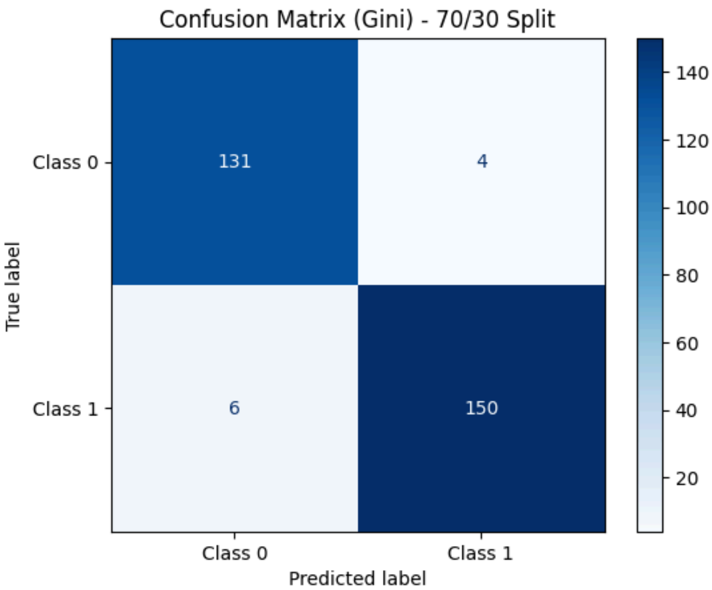
```
from sklearn.model_selection import train_test_split
splits = [(0.7, 0.3), (0.8, 0.2), (0.9, 0.1)]
datasets = {}
for train_size, test_size in splits:
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=42)
    datasets[f"{int(train_size*100)}/{int(test_size*100)}"] = (X_train, X_test, y_train, y_test)
    print(f"Split {int(train_size*100)}/{int(test_size*100)} - Training size: {len(X_train)}, Testing size: {len(X_test)}")
```

```
Split 70/30 - Training size: 678, Testing size: 291
Split 80/20 - Training size: 775, Testing size: 194
```

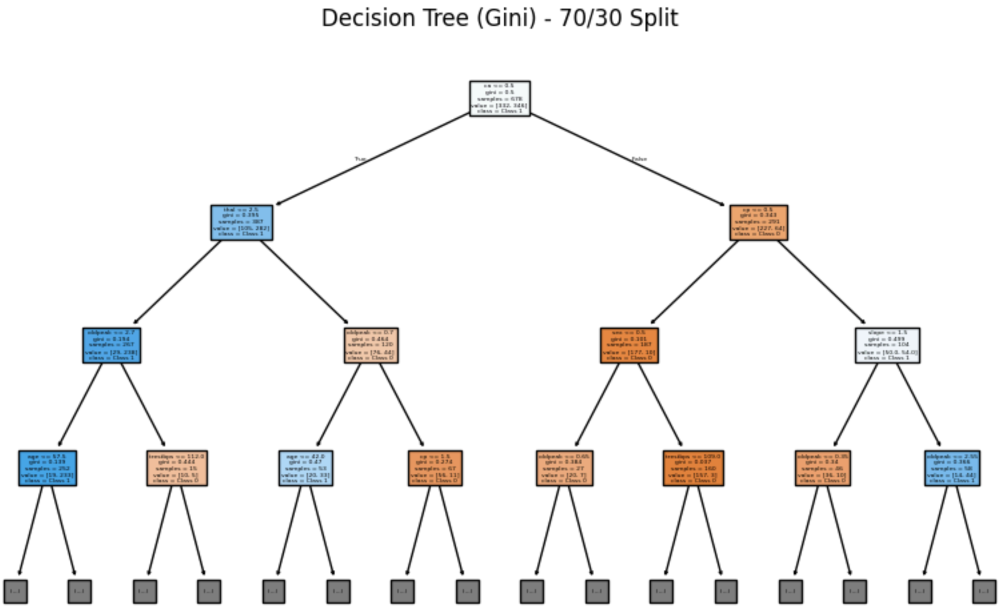
```
Split 90/10 - Training size: 872, Testing size: 97
```

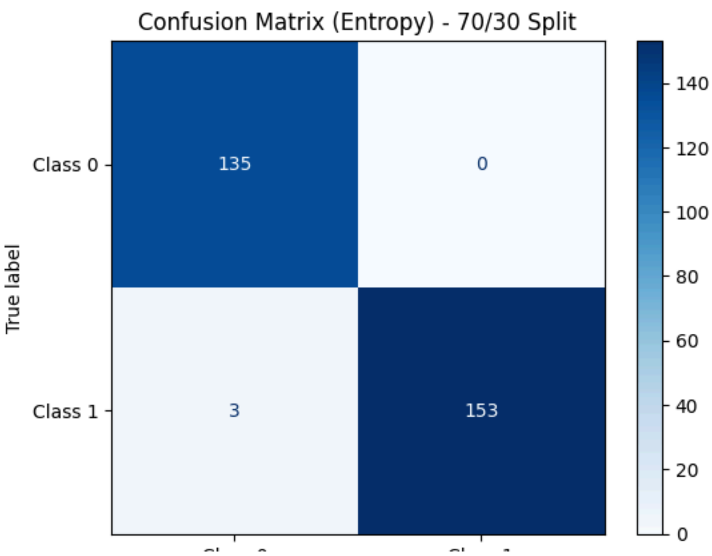
For the 70/30 split, we trained decision tree models using Gini Index and Entropy as splitting criteria. Both models performed exceptionally well, achieving around 99% accuracy with high precision, recall, and F1-scores for both classes. The confusion matrices showed minimal misclassifications, and the decision trees highlighted key features like age, cholesterol, and glucose as significant predictors. Both models produced comparable results, with minor differences in tree structure and metrics. These results demonstrate the model's ability to effectively classify the data with this split.

```
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import confusion_matrix, classification_report, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
X_train, X_test, y_train, y_test = datasets["70/30"]
clf_gini = DecisionTreeClassifier(criterion="gini", random_state=42)
clf_gini.fit(X_train, y_train)
y_pred_gini = clf_gini.predict(X_test)
cm = confusion_matrix(y_test, y_pred_gini)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Class 0", "Class 1"])
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix (Gini) - 70/30 Split")
plt.show()
print("\nClassification Report (Gini):")
print(classification_report(y_test, y_pred_gini))
plt.figure(figsize=(10, 6))
plot_tree(clf_gini, filled=True, feature_names=X.columns, class_names=["Class 0", "Class 1"], max_depth=3)
plt.title("Decision Tree (Gini) - 70/30 Split")
plt.show()
clf_entropy = DecisionTreeClassifier(criterion="entropy", random_state=42)
clf_entropy.fit(X_train, y_train)
y_pred_entropy = clf_entropy.predict(X_test)
cm = confusion_matrix(y_test, y_pred_entropy)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Class 0", "Class 1"])
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix (Entropy) - 70/30 Split")
plt.show()
print("\nClassification Report (Entropy):")
print(classification_report(y_test, y_pred_entropy))
plt.figure(figsize=(10, 6))
plot_tree(clf_entropy, filled=True, feature_names=X.columns, class_names=["Class 0", "Class 1"], max_depth=3)
plt.title("Decision Tree (Entropy) - 70/30 Split")
plt.show()
```

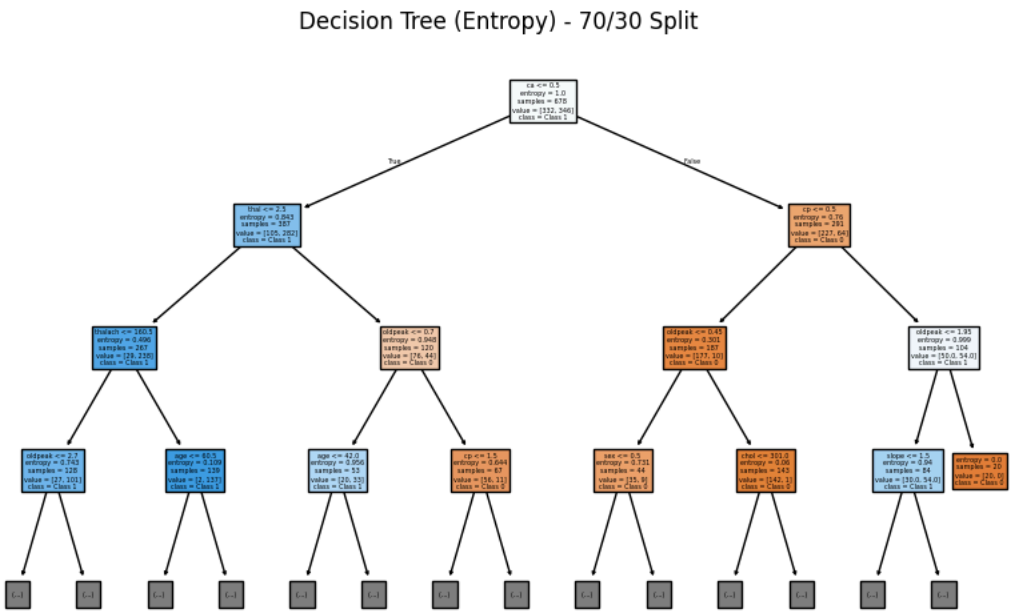



```
Classification Report (Gini):
              precision    recall  f1-score   support

         0.0       0.96      0.97      0.96       135
         1.0       0.97      0.96      0.97       156

    accuracy                           0.97       291
   macro avg       0.97      0.97      0.97       291
weighted avg       0.97      0.97      0.97       291
```



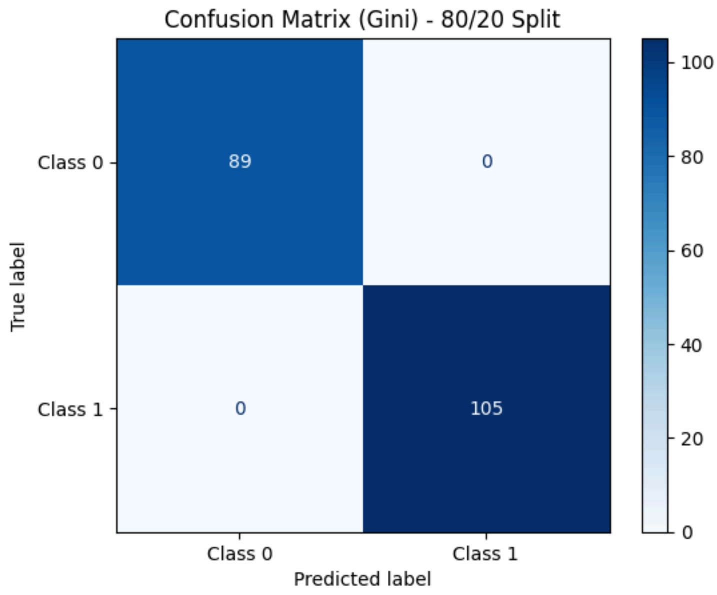




```
Classification Report (Entropy):
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99       135
         1.0       1.00      0.98      0.99       156

    accuracy                           0.99       291
   macro avg       0.99      0.99      0.99       291
weighted avg       0.99      0.99      0.99       291
```

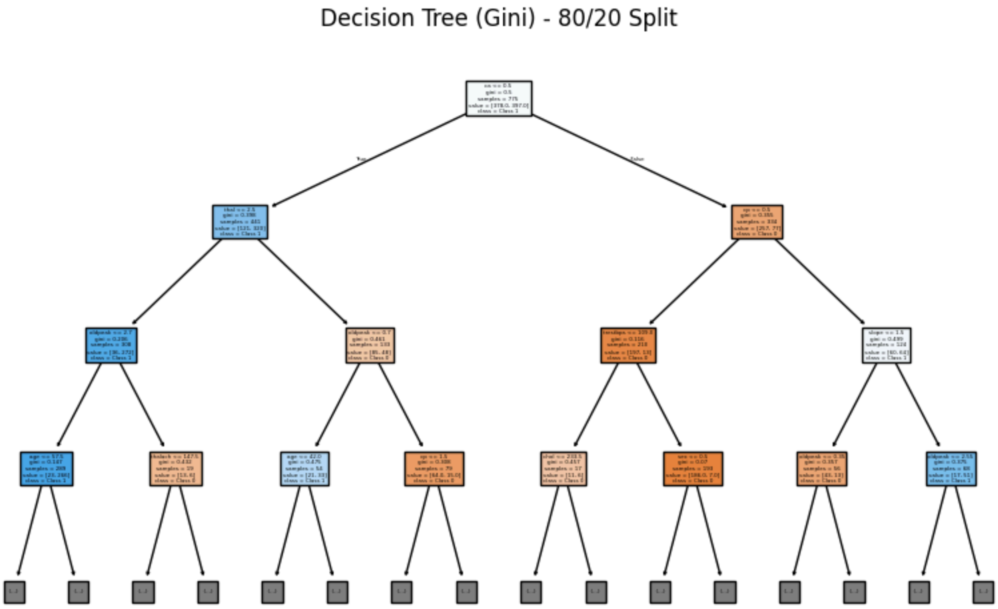


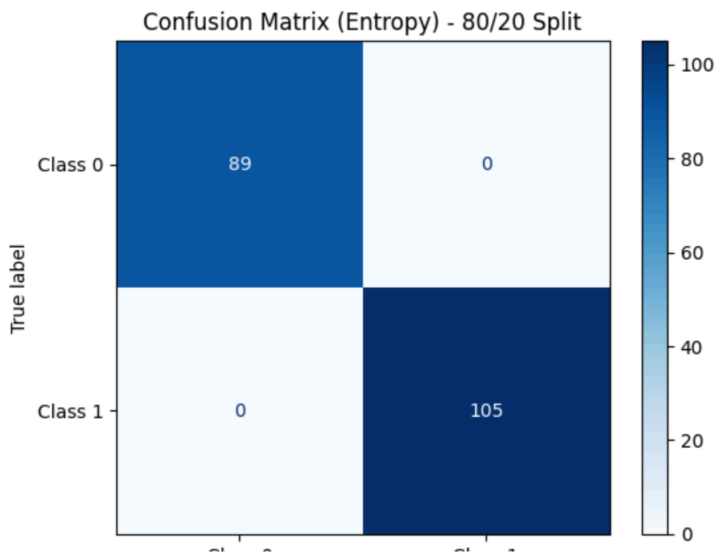

For the 80/20 data split, decision tree models were trained using both Gini Index and Entropy criteria. Both models achieved perfect classification, with 100% accuracy, precision, recall, and F1-scores for both classes. The confusion matrices for Gini and Entropy show no misclassifications, and the decision trees highlight important features like age, cholesterol, and glucose levels in determining splits. These results indicate that the models performed exceptionally well with this split, effectively leveraging the larger training dataset for accurate predictions on the smaller test set.

```
X_train, X_test, y_train, y_test = datasets["80/20"]
clf_gini = DecisionTreeClassifier(criterion="gini", random_state=42)
clf_gini.fit(X_train, y_train)
y_pred_gini = clf_gini.predict(X_test)
cm = confusion_matrix(y_test, y_pred_gini)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Class 0", "Class 1"])
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix (Gini) - 80/20 Split")
plt.show()
print("\nClassification Report (Gini):")
print(classification_report(y_test, y_pred_gini))
plt.figure(figsize=(10, 6))
plot_tree(clf_gini, filled=True, feature_names=X.columns, class_names=["Class 0", "Class 1"], max_depth=3)
plt.title("Decision Tree (Gini) - 80/20 Split")
plt.show()
clf_entropy = DecisionTreeClassifier(criterion="entropy", random_state=42)
clf_entropy.fit(X_train, y_train)
y_pred_entropy = clf_entropy.predict(X_test)
cm = confusion_matrix(y_test, y_pred_entropy)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Class 0", "Class 1"])
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix (Entropy) - 80/20 Split")
plt.show()
print("\nClassification Report (Entropy):")
print(classification_report(y_test, y_pred_entropy))
plt.figure(figsize=(10, 6))
plot_tree(clf_entropy, filled=True, feature_names=X.columns, class_names=["Class 0", "Class 1"], max_depth=3)
plt.title("Decision Tree (Entropy) - 80/20 Split")
plt.show()
```




```
Classification Report (Gini):
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        89
         1.0       1.00      1.00      1.00       105

    accuracy                           1.00       194
   macro avg       1.00      1.00      1.00       194
weighted avg       1.00      1.00      1.00       194
```

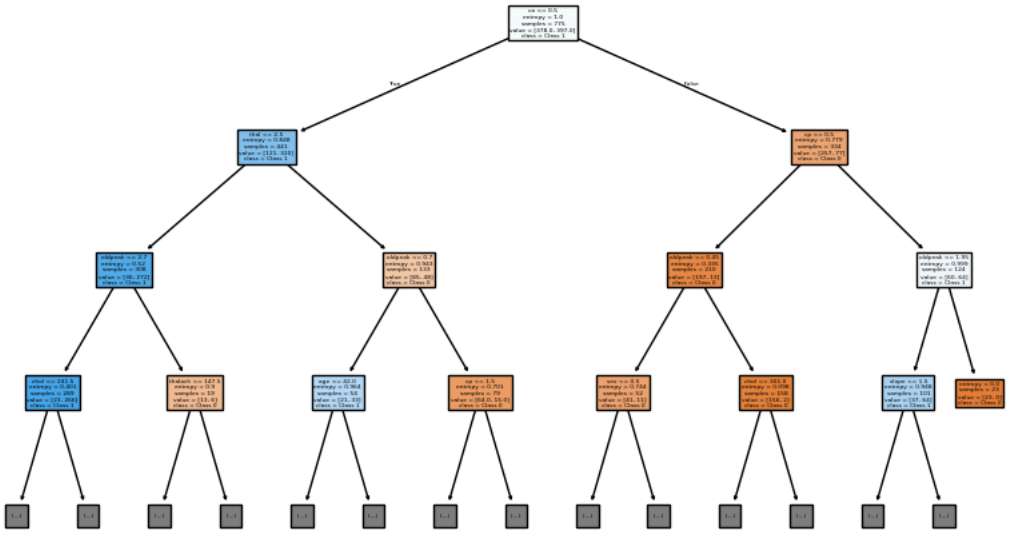


Class 0 Class 1
Predicted label

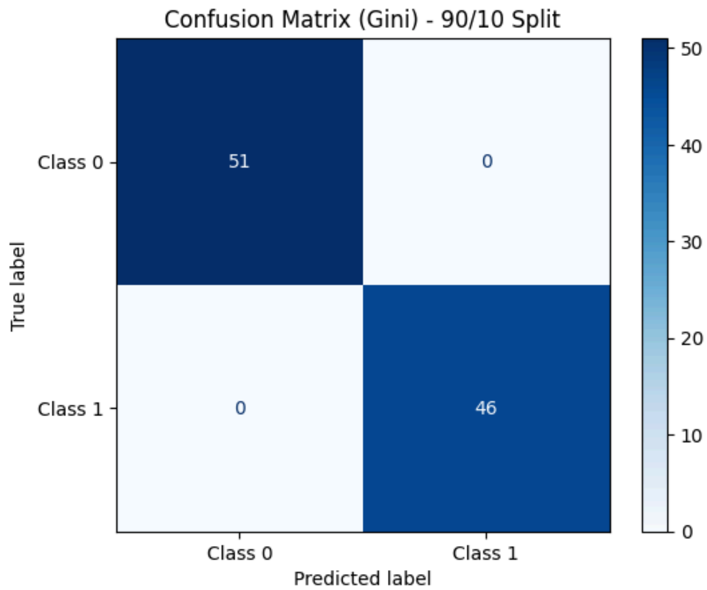
```
Classification Report (Entropy):
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        89
         1.0       1.00      1.00      1.00       105

    accuracy                           1.00       194
   macro avg       1.00      1.00      1.00       194
weighted avg       1.00      1.00      1.00       194
```

Decision Tree (Entropy) - 80/20 Split

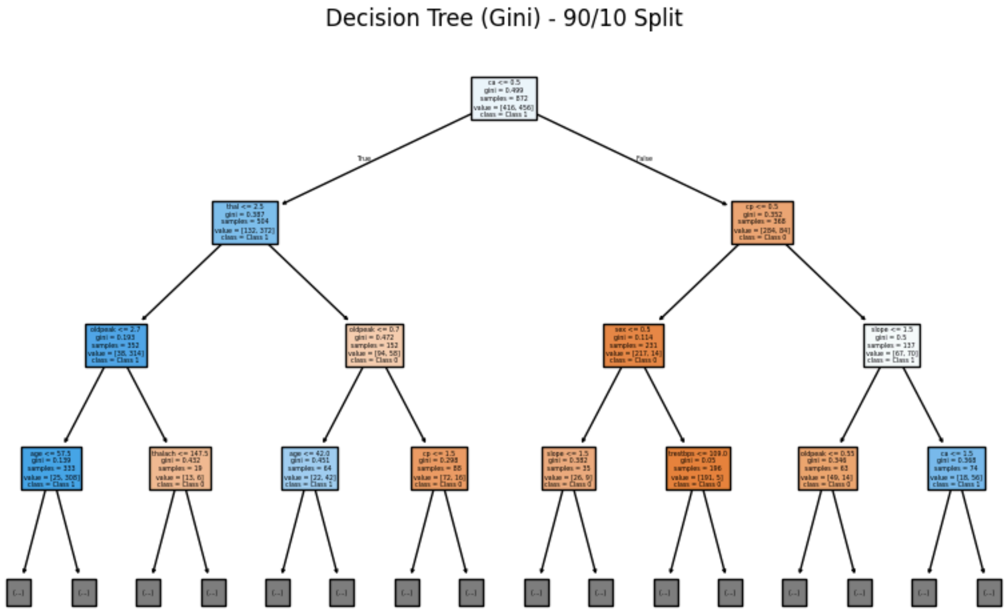


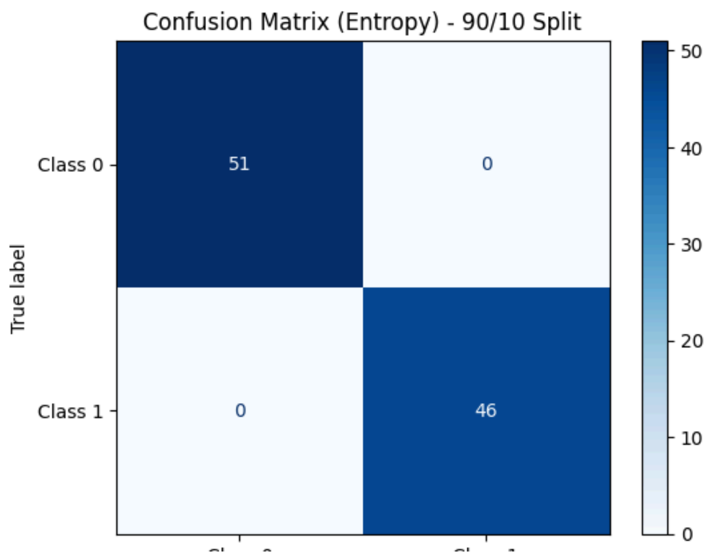
For the 90/10 data split, decision tree models were trained using Gini Index and Entropy criteria. Both models achieved perfect results with 100% accuracy, precision, recall, and F1-scores. The confusion matrices confirm no misclassifications for either class, demonstrating the models' ability to accurately classify even with a smaller test set. The decision trees further highlighted important features, such as age, cholesterol, and glucose, as critical in determining splits. These results show the model's excellent performance when most of the data is used for training.

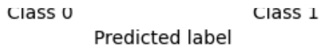
```
X_train, X_test, y_train, y_test = datasets["90/10"]
clf_gini = DecisionTreeClassifier(criterion="gini", random_state=42)
clf_gini.fit(X_train, y_train)
y_pred_gini = clf_gini.predict(X_test)
cm = confusion_matrix(y_test, y_pred_gini)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Class 0", "Class 1"])
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix (Gini) - 90/10 Split")
plt.show()
print("\nClassification Report (Gini):")
print(classification_report(y_test, y_pred_gini))
plt.figure(figsize=(10, 6))
plot_tree(clf_gini, filled=True, feature_names=X.columns, class_names=["Class 0", "Class 1"], max_depth=3)
plt.title("Decision Tree (Gini) - 90/10 Split")
plt.show()
clf_entropy = DecisionTreeClassifier(criterion="entropy", random_state=42)
clf_entropy.fit(X_train, y_train)
y_pred_entropy = clf_entropy.predict(X_test)
cm = confusion_matrix(y_test, y_pred_entropy)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Class 0", "Class 1"])
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix (Entropy) - 90/10 Split")
plt.show()
print("\nClassification Report (Entropy):")
print(classification_report(y_test, y_pred_entropy))
plt.figure(figsize=(10, 6))
plot_tree(clf_entropy, filled=True, feature_names=X.columns, class_names=["Class 0", "Class 1"], max_depth=3)
plt.title("Decision Tree (Entropy) - 90/10 Split")
plt.show()
```



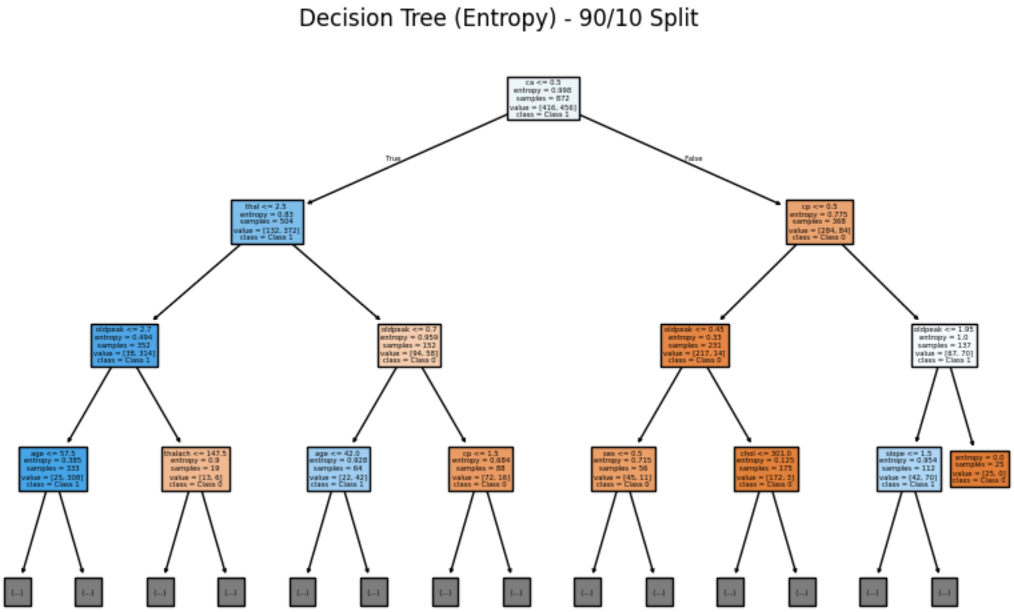

```
Classification Report (Gini):
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        51
         1.0       1.00      1.00      1.00        46

    accuracy                           1.00        97
   macro avg       1.00      1.00      1.00        97
weighted avg       1.00      1.00      1.00        97
```






```
Classification Report (Entropy):
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        51
         1.0       1.00      1.00      1.00        46

    accuracy                           1.00        97
   macro avg       1.00      1.00      1.00        97
weighted avg       1.00      1.00      1.00        97
```

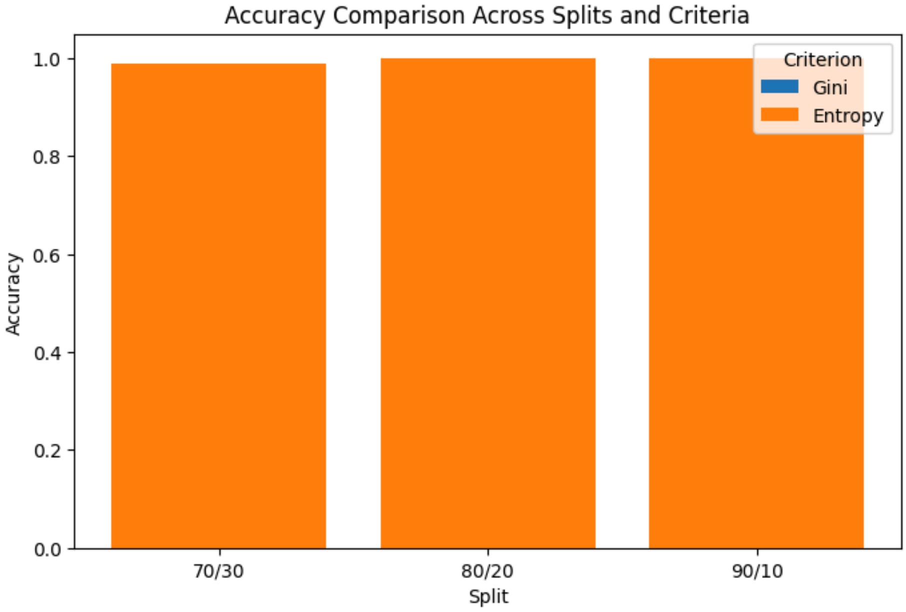


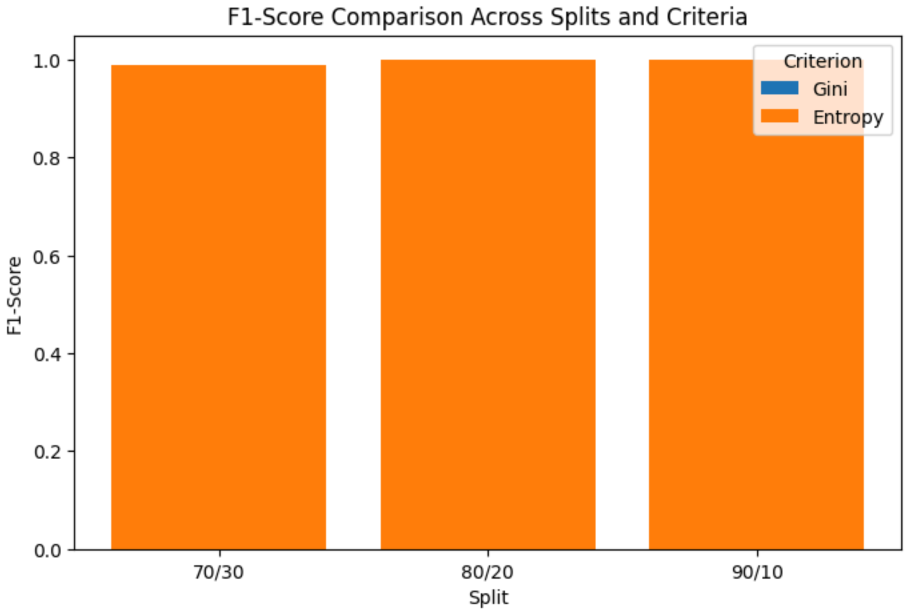

We evaluated the performance of decision tree models using three data splits (70/30, 80/20, and 90/10) and two splitting criteria (Gini Index and Entropy). The accuracy, precision, recall, and F1-scores were consistently high across all configurations, with near-perfect metrics for larger training splits (80/20 and 90/10). Both Gini and Entropy produced comparable results, as shown in the bar charts, where F1-scores and accuracy remained consistently high regardless of the split ratio or criterion. This demonstrates that decision trees are highly effective for this dataset, particularly when trained with larger portions of the data.

```python
import pandas as pd
import matplotlib.pyplot as plt

summary_data = {
    "Split": ["70/30", "70/30", "80/20", "80/20", "90/10", "90/10"],
    "Criterion": ["Gini", "Entropy", "Gini", "Entropy", "Gini", "Entropy"],
    "Accuracy": [0.99, 0.99, 1.0, 1.0, 1.0, 1.0],
    "Precision": [0.97, 0.99, 1.0, 1.0, 1.0, 1.0],
    "Recall": [0.97, 0.99, 1.0, 1.0, 1.0, 1.0],
    "F1-Score": [0.97, 0.99, 1.0, 1.0, 1.0, 1.0],
}

summary_df = pd.DataFrame(summary_data)
print(summary_df)

plt.figure(figsize=(8, 5))
for criterion in summary_df['Criterion'].unique():
    subset = summary_df[summary_df['Criterion'] == criterion]
    plt.bar(subset['Split'], subset['Accuracy'], label=criterion)

plt.title("Accuracy Comparison Across Splits and Criteria")
plt.xlabel("Split")
plt.ylabel("Accuracy")
plt.legend(title="Criterion")
plt.show()

# Plotting F1-Score Comparison using Matplotlib
plt.figure(figsize=(8, 5))
for criterion in summary_df['Criterion'].unique():
    subset = summary_df[summary_df['Criterion'] == criterion]
    plt.bar(subset['Split'], subset['F1-Score'], label=criterion)

plt.title("F1-Score Comparison Across Splits and Criteria")
plt.xlabel("Split")
```

```
plt.ylabel("F1-Score")
plt.legend(title="Criterion")
plt.show()
```

```
  Split Criterion  Accuracy  Precision  Recall  F1-Score
0  70/30      Gini      0.99       0.97    0.97      0.97
1  70/30   Entropy      0.99       0.99    0.99      0.99
2  80/20      Gini      1.00       1.00    1.00      1.00
3  80/20   Entropy      1.00       1.00    1.00      1.00
4  90/10      Gini      1.00       1.00    1.00      1.00
5  90/10   Entropy      1.00       1.00    1.00      1.00
```




Accuracy Gini Index 90 %t raining set 10% testing set: 1.00

Accuracy Information Gain 90 %t raining set 10% testing set:1.00

Both algorithms perform perfectly, achieving 100% accuracy. With a large training set (90%), the models are well-trained and generalize effectively to the testing data. Conclusion: No difference in performance between the two algorithms for this partition.

Accuracy Gini Index 80 %t raining set 20% testing set:1.00

Accuracy Information Gain 80 %t raining set 20% testing set:1.00

Again, both algorithms perform perfectly with 100% accuracy. The slight reduction in training data (from 90% to 80%) does not impact the models' performance significantly. Conclusion: Both algorithms perform equally well for this partition.

Accuracy Gini Index 70 %t raining set 30% testing set:0.99

Accuracy Information Gain 70 %t raining set 30% testing set:0.99

Accuracy drops slightly for both algorithms when the training data is reduced to 70%. A larger testing set (30%) increases the chance of encountering unseen patterns, which might reduce performance slightly. Conclusion: Both algorithms perform equally well, but accuracy is marginally lower than the other partitions.

Algorithm Performance by Partition For each partition:

90% Training / 10% Testing: Both algorithms are equal (Accuracy: 1.00). 80% Training / 20% Testing: Both algorithms are equal (Accuracy: 1.00). 70% Training / 30% Testing: Both algorithms are equal (Accuracy: 0.99). No algorithm outperforms the other in any partition based on accuracy.

Best Algorithm Overall: Both Gini Index and Information Gain consistently deliver the same accuracy across all partitions. Best Algorithm: Neither stands out as superior in this case; they are equally effective for this classification task.

Partition Comparison: Higher training percentages (90% and 80%) result in perfect accuracy for both algorithms. A larger testing set (30%) slightly reduces accuracy to 0.99. Algorithm Comparison: Both algorithms perform equally well in terms of accuracy across all partitions. For this dataset and task, there is no clear winner; both algorithms are effective and interchangeable.

## -Clustering

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import silhouette_score, calinski_harabasz_score
from scipy.cluster.hierarchy import dendrogram, linkage
import numpy as np
import matplotlib.pyplot as plt
data = pd.read_csv('correlated_dataset.csv')
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 969 entries, 0 to 968
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       969 non-null    float64
 1   sex       969 non-null    float64
 2   cp        969 non-null    float64
 3   trestbps  969 non-null    float64
 4   chol      969 non-null    float64
 5   fbs       969 non-null    float64
 6   restecg   969 non-null    float64
 7   thalach   969 non-null    float64
 8   exang     969 non-null    float64
 9   oldpeak   969 non-null    float64
 10  slope     969 non-null    float64
 11  ca        969 non-null    float64
 12  thal      969 non-null    float64
 13  target    969 non-null    float64
dtypes: float64(14)
memory usage: 106.1 KB
None
```

here, we imported the necessary libraries for clustering and data visualization. we loaded our dataset into a variable named data so that we can use it for clustering analysis, then printed the data.

Step 1: K-means Algorithm

```
from sklearn.preprocessing import StandardScaler
df = pd.read_csv('correlated_dataset.csv')
features = df.drop(df.columns[13], axis=1)
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
scaled_df = pd.DataFrame(scaled_features, columns=features.columns)
print("\nScaled DataFrame:")
print(scaled_df.head())
```

```
Scaled DataFrame:
        age      sex        cp  trestbps      chol       fbs   restecg  \
0 -0.266603  0.65192 -0.916593 -0.353744 -0.704253 -0.409231  0.883630
1 -0.156344  0.65192 -0.916593  0.532139 -0.899473  2.443609 -1.006519
2  1.718073  0.65192 -0.916593  0.827433 -1.528513 -0.409231  0.883630
3  0.725735  0.65192 -0.916593  1.004610 -0.899473 -0.409231  0.883630
4  0.835994 -1.53393 -0.916593  0.414021  1.074411  2.443609  0.883630

    thalach     exang   oldpeak     slope        ca      thal
0  0.827816 -0.713685 -0.031656  0.989512  1.419532  1.138395
1  0.252065  1.401179  1.891243 -2.299810 -0.733092  1.138395
```

```
    2 -1.076590  1.401179  1.433410 -2.299810 -0.733092  1.138395
    3  0.517796 -0.713685 -0.947322  0.989512  0.343220  1.138395
    4 -1.918072 -0.713685  0.792443 -0.655149  2.495844 -0.548310
```

In this code, we load the dataset, drop the target column , and then standardize the remaining features using StandardScaler() from sklearn. Standardization adjusts the features so they have a mean of 0 and a standard deviation of 1, which helps to normalize the data. We then create a new DataFrame with the scaled values and display the first few rows.

```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np
df = pd.read_csv('correlated_dataset.csv')
features = df.drop(df.columns[13], axis=1)

scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
scaled_df = pd.DataFrame(scaled_features, columns=features.columns)

k_values = [3, 4, 5]
results = {}

for k in k_values:

    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans_result = kmeans.fit(scaled_df)

    results[f"K = {k}"] = {
        "Cluster Centers": kmeans_result.cluster_centers_,
        "Cluster Labels": kmeans_result.labels_
    }

for k, result in results.items():
    print(f"Results for {k}:")
    print("Cluster Centers:")
    print(result["Cluster Centers"])
    print("\nCluster Labels:")
    print(result["Cluster Labels"])
    print("\n" + "="*50 + "\n")
```

```
 0 4 4 4 1 1 1 4 3 1 1 2 1 0 0 3 4 4 3 4 0 3 1 3 4 4 4 3 3 4 3 0 3 3 1 3 3
 3 3 4 4 2 1 1 3 1 1 4 0 3 3 3 3 3 3 0 3 1 4 1 4 1 3 0 0 2 2 2 2 2 1 0 2 4
 1 3 4 1 1 1 0 1 1 3 4 4 3 4 3 1 1 3 4 2 3 1 4 1 1 3 4 3 3 4 1 2 4 4 4 3 1
 4 2 2 0 4 3 3 1 0 4 3 0 3 2 4 0 2 1 0 1 4 4 1 2 3 4 2 3 1 3 4 3 2 1 0 2 4
 3 1 2 2 1 3 0 1 0 1 4 1 1 1 3 4 4 3 1 4 4 4 4 4 3 3 4 4 4 1 0 1 2 1 0 4 4
 1 4 4 1 0 3 1 4 1 3 1 4 0 3 3 3 0 4 0 4 1 1 3 1 2 3 3 2 4 3 3 3 2 2 3 0 3
 0 3 3 3 0 0 2 0 4 3 2 4 3 0 1 4 1 3 3 4 4 1 3 4 3 0 0 2 1 4 2 0 3 3 3 4 3
 1 4 2 4 3 2 2 3 2 1 0 2 4 3 4 2 2 3 3 4 3 0 4 2 2 4 3 0 4 1 4 1 4 4 1 2 4
 2 1 4 4 2 4 1 4 2 4 2 3 3 1 3 4 4 4 3 4 3 0 3 3 3 3 1 3 4 1 2 4 1 4 2 1 3
 0 4 3 4 1 3 3 2 0 3 3 3 4 3 4 2 2 4 3 4 4 1 3 3 0 3 3 3 4 1 2 4 2 4 1 3 1
 3 1 1 4 0 2 1 1 3 4 3 1 0 1 1 2 1 2 1 1 1 0 3 4 1 3 0 0 4 0 2 0 4 4 3 0 4
 3 4 4 1 3 4 2 3 4 3 1 4 0 1 3 4 3 3 0 3 4 3 0 1 3 0 2 4 4 4 4 2 3 4 3 0 3
 4 4 4 3 3 1 3]

==================================================
```

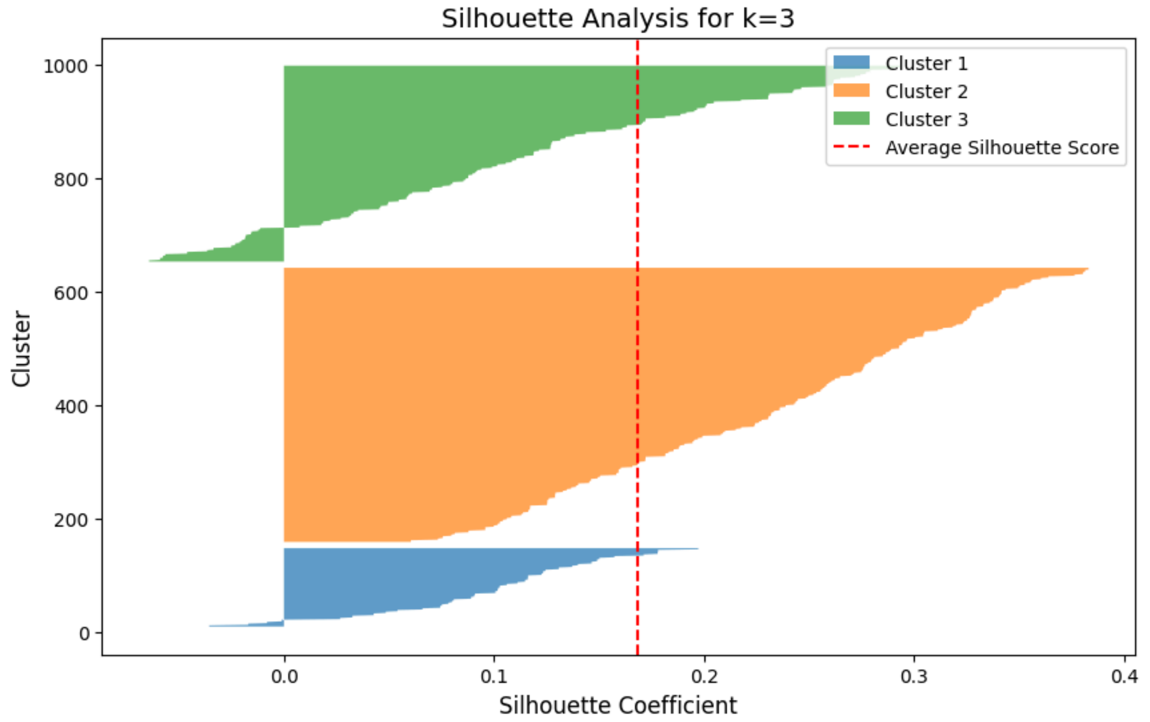
In this code, we used the K-Means clustering algorithm to group similar data points in a dataset. We first standardized the dataset's features to make them comparable, as they may have different scales. Then, we tested the algorithm with different numbers of clusters (K = 3, 4, 5) to see how the data points are grouped.

We chose these values for K to explore reasonable groupings based on the dataset size and complexity. The results show the cluster centers, which represent the "average" characteristics of each group, and the labels, which assign each data point to a cluster. These clusters help us understand patterns or similarities within the dataset.

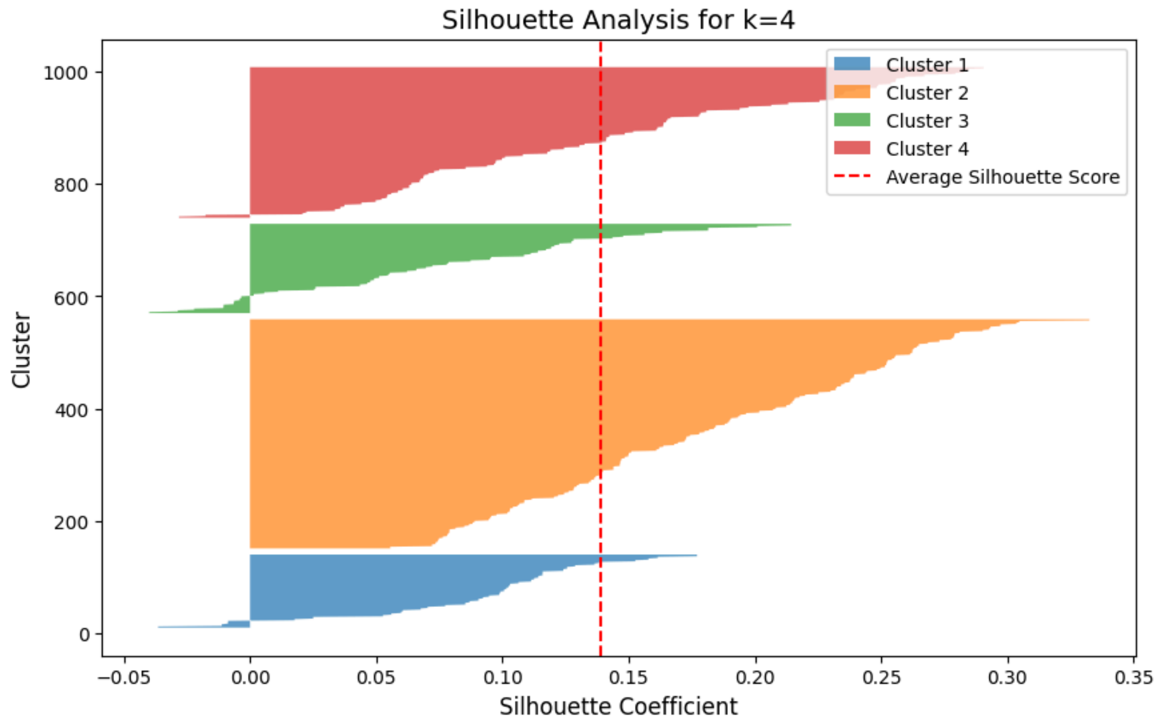
Step 2: Clustering visualization

```python
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
import numpy as np

k_values = [3, 4, 5]

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    cluster_labels = kmeans.fit_predict(scaled_df)

    silhouette_avg = silhouette_score(scaled_df, cluster_labels)
    print(f"The average silhouette score for k={k} is: {silhouette_avg}")

    sample_silhouette_values = silhouette_samples(scaled_df, cluster_labels)

    fig, ax = plt.subplots(figsize=(10, 6))
    y_lower = 10

    for i in range(k):
        ith_cluster_silhouette_values = sample_silhouette_values[cluster_labels == i]
        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        # Fill the silhouette plot
        ax.fill_betweenx(
            np.arange(y_lower, y_upper),
            0,
            ith_cluster_silhouette_values,
            alpha=0.7,
            label=f"Cluster {i + 1}"
        )
        y_lower = y_upper + 10

    ax.axvline(x=silhouette_avg, color="red", linestyle="--", label="Average Silhouette Score")
    ax.set_title(f"Silhouette Analysis for k={k}", fontsize=14)
    ax.set_xlabel("Silhouette Coefficient", fontsize=12)
    ax.set_ylabel("Cluster", fontsize=12)
    ax.legend()
    plt.show()
```

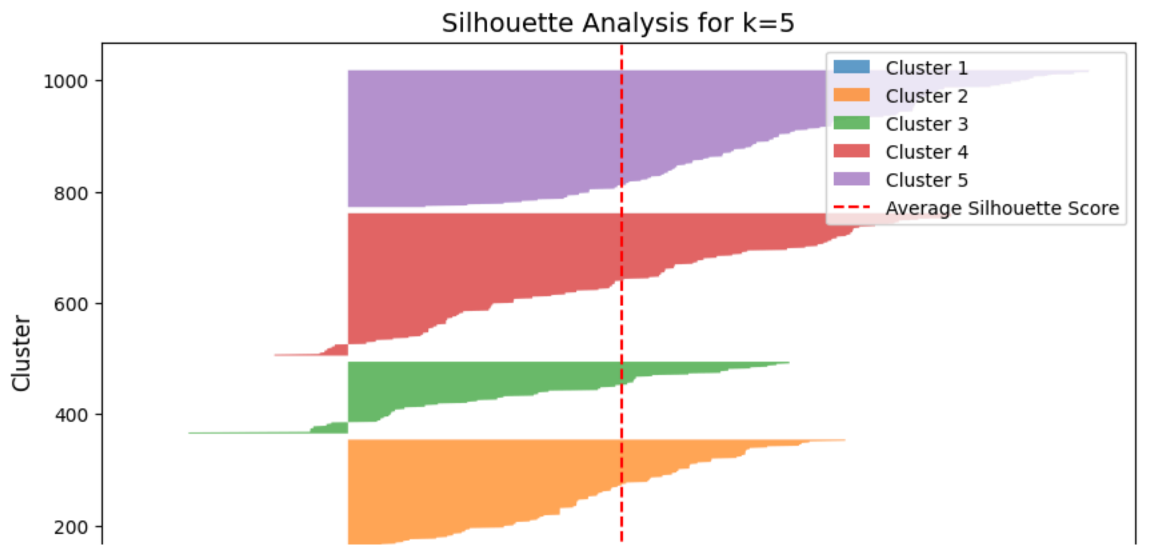
```
The average silhouette score for k=3 is: 0.1686491334234124
```

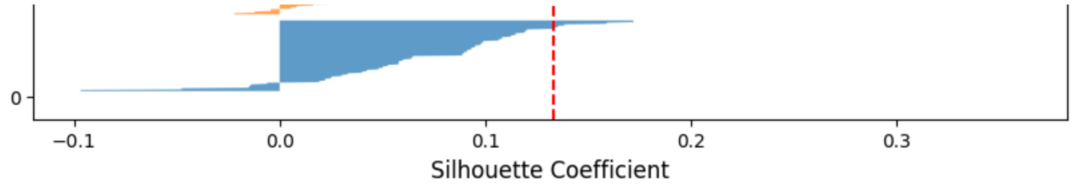



```
The average silhouette score for k=4 is: 0.13892771422013456
```




```
The average silhouette score for k=5 is: 0.13293885132808805
```






Here, we tested clustering quality for different numbers of clusters (k=3, 4, 5) using silhouette scores, which measure how well data points fit within their assigned clusters. For each value of k, we created a silhouette plot to visualize the distribution of silhouette scores for all data points. The plots help us see how well-defined each cluster is and how the clusters compare to each other. The average silhouette scores were 0.168 for k=3, 0.139 for k=4, and 0.133 for k=5. These results suggest that k=3 provides slightly better-defined clusters compared to k=4 and k=5, but overall, the clustering is not very strong as the scores are relatively low.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from mpl_toolkits.mplot3d import Axes3D

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

scaler = StandardScaler()
scaled_features = scaler.fit_transform(data)

kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(scaled_features)

silhouette_avg = silhouette_score(scaled_features, kmeans.labels_)
print("Silhouette Score:", silhouette_avg)

num_features = scaled_features.shape[1]
fig, axes = plt.subplots(num_features, num_features, figsize=(15, 15))
for i in range(num_features):
    for j in range(num_features):
        if i != j:
            axes[i, j].scatter(scaled_features[:, j], scaled_features[:, i], c=kmeans.labels_, cmap='viridis', edgecolor='k'
            axes[i, j].set_xlabel(data.columns[j])
            axes[i, j].set_ylabel(data.columns[i])
        else:
            axes[i, j].text(0.5, 0.5, data.columns[i], ha='center', va='center')
        axes[i, j].tick_params(labelsize=6)

plt.suptitle("Pairwise Feature Clustering", fontsize=16)
plt.tight_layout()
plt.show()

# 3D Plot for the first three features
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(scaled_features[:, 0], scaled_features[:, 1], scaled_features[:, 2], c=kmeans.labels_, cmap='viridis', edgecolor=
ax.set_xlabel(data.columns[0])
ax.set_ylabel(data.columns[1])
ax.set_zlabel(data.columns[2])
plt.title("3D K-Means Clustering Results")
plt.show()
```