Heart Disease Risk Analysis

Rohit Goutam Maity

2024-11-12

Student ID:2188913

{r library(tidyverse) }

(A)DATA GATHERING AND INTEGRATION

Importing our dataset & Loading Required Libraries

```
# Load essential libraries for data manipulation, visualization, and modeling
library(tidyverse)   # For data handling and plotting
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(dplyr)       # For data manipulation
library(caret)       # For model training and evaluation
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(e1071)       # For SVM modeling
library(ggplot2)     # For visualization
library(rpart)       # For decision tree modeling
library(ggfortify)   # For PCA visualization
library(kknn)        # For KNN model with tuning
```

```
##
## Attaching package: 'kknn'
##
## The following object is masked from 'package:caret':
##
```

```
##     contr.dummy
```

```
# Load the heart disease dataset and display the first few rows
heart <- read.csv("heart.csv", header = TRUE, sep = ",")
head(heart)
```

```
##   Age Sex ChestPainType RestingBP Cholesterol FastingBS RestingECG MaxHR
## 1  40   M           ATA       140         289         0     Normal   172
## 2  49   F           NAP       160         180         0     Normal   156
## 3  37   M           ATA       130         283         0         ST    98
## 4  48   F           ASY       138         214         0     Normal   108
## 5  54   M           NAP       150         195         0     Normal   122
## 6  39   M           NAP       120         339         0     Normal   170
##   ExerciseAngina Oldpeak ST_Slope HeartDisease
## 1              N     0.0       Up            0
## 2              N     1.0     Flat            1
## 3              N     0.0       Up            0
## 4              Y     1.5     Flat            1
## 5              N     0.0       Up            0
## 6              N     0.0       Up            0
```

```
# Examine the structure and columns of the dataset
str(heart)
```

```
## 'data.frame':    918 obs. of  12 variables:
##  $ Age           : int  40 49 37 48 54 39 45 54 37 48 ...
##  $ Sex           : chr  "M" "F" "M" "F" ...
##  $ ChestPainType : chr  "ATA" "NAP" "ATA" "ASY" ...
##  $ RestingBP     : int  140 160 130 138 150 120 130 110 140 120 ...
##  $ Cholesterol   : int  289 180 283 214 195 339 237 208 207 284 ...
##  $ FastingBS     : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ RestingECG    : chr  "Normal" "Normal" "ST" "Normal" ...
##  $ MaxHR         : int  172 156 98 108 122 170 170 142 130 120 ...
##  $ ExerciseAngina: chr  "N" "N" "N" "Y" ...
##  $ Oldpeak       : num  0 1 0 1.5 0 0 0 0 1.5 0 ...
##  $ ST_Slope      : chr  "Up" "Flat" "Up" "Flat" ...
##  $ HeartDisease  : int  0 1 0 1 0 0 0 0 1 0 ...
```

```
names(heart)
```

```
##  [1] "Age"            "Sex"            "ChestPainType"  "RestingBP"
##  [5] "Cholesterol"    "FastingBS"      "RestingECG"     "MaxHR"
##  [9] "ExerciseAngina" "Oldpeak"        "ST_Slope"       "HeartDisease"
```

Explanation: After loading the dataset, the structure and column names are inspected to understand the types and organization of the data. This step is essential for assessing variable formats (e.g., numeric vs. categorical) and potential cleaning needs.

(B)DATA EXPLORATION

Exploring our dataset

```
# Display summary statistics for all columns
summary(heart)
```

```
##       Age            Sex            ChestPainType        RestingBP
##  Min.   :28.00   Length:918         Length:918         Min.   :  0.0
```

```
##  1st Qu.:47.00   Class :character   Class :character   1st Qu.:120.0
##  Median :54.00   Mode  :character   Mode  :character   Median :130.0
##  Mean   :53.51                                         Mean   :132.4
##  3rd Qu.:60.00                                         3rd Qu.:140.0
##  Max.   :77.00                                         Max.   :200.0
##   Cholesterol      FastingBS       RestingECG            MaxHR
##  Min.   :  0.0   Min.   :0.0000   Length:918         Min.   : 60.0
##  1st Qu.:173.2   1st Qu.:0.0000   Class :character   1st Qu.:120.0
##  Median :223.0   Median :0.0000   Mode  :character   Median :138.0
##  Mean   :198.8   Mean   :0.2331                      Mean   :136.8
##  3rd Qu.:267.0   3rd Qu.:0.0000                      3rd Qu.:156.0
##  Max.   :603.0   Max.   :1.0000                      Max.   :202.0
##  ExerciseAngina       Oldpeak          ST_Slope          HeartDisease
##  Length:918         Min.   :-2.6000   Length:918         Min.   :0.0000
##  Class :character   1st Qu.: 0.0000   Class :character   1st Qu.:0.0000
##  Mode  :character   Median : 0.6000   Mode  :character   Median :1.0000
##                     Mean   : 0.8874                      Mean   :0.5534
##                     3rd Qu.: 1.5000                      3rd Qu.:1.0000
##                     Max.   : 6.2000                      Max.   :1.0000
```

```
# Analyze key groupings in the data to identify trends by categorical variables
# Grouping by Chest Pain Type
heart %>% group_by(ChestPainType) %>% summarise(count = n())
```

```
## # A tibble: 4 x 2
##   ChestPainType count
##   <chr>         <int>
## 1 ASY             496
## 2 ATA             173
## 3 NAP             203
## 4 TA               46
```

```
# Grouping by Heart Disease status
heart %>% group_by(HeartDisease) %>% summarise(count = n())
```

```
## # A tibble: 2 x 2
##   HeartDisease count
##          <int> <int>
## 1            0   410
## 2            1   508
```

```
# Group by Resting ECG
heart %>% group_by(RestingECG) %>% summarise(count = n())
```

```
## # A tibble: 3 x 2
##   RestingECG count
##   <chr>      <int>
## 1 LVH          188
## 2 Normal       552
## 3 ST           178
```

```
# Group by ST Slope
heart %>% group_by(ST_Slope) %>% summarise(count = n())
```

```
## # A tibble: 3 x 2
##   ST_Slope count
##   <chr>    <int>
```

```
## 1 Down        63
## 2 Flat       460
## 3 Up         395
```

```
# Group by Sex and find mean Cholesterol level
heart %>% group_by(Sex) %>% summarize(avg_cholesterol = mean(Cholesterol, na.rm = TRUE))
```

```
## # A tibble: 2 x 2
##   Sex   avg_cholesterol
##   <chr>           <dbl>
## 1 F                241.
## 2 M                188.
```

```
# Group by Sex and find mean of Max Heart Rate
heart %>% group_by(Sex) %>% summarize(avg_heartrate = mean(MaxHR, na.rm = TRUE))
```

```
## # A tibble: 2 x 2
##   Sex   avg_heartrate
##   <chr>         <dbl>
## 1 F              146.
## 2 M              134.
```

```
# Count heart disease occurrences by Sex
heart %>% group_by(Sex) %>% count(HeartDisease)
```

```
## # A tibble: 4 x 3
## # Groups:   Sex [2]
##   Sex   HeartDisease     n
##   <chr>        <int> <int>
## 1 F                0   143
## 2 F                1    50
## 3 M                0   267
## 4 M                1   458
```

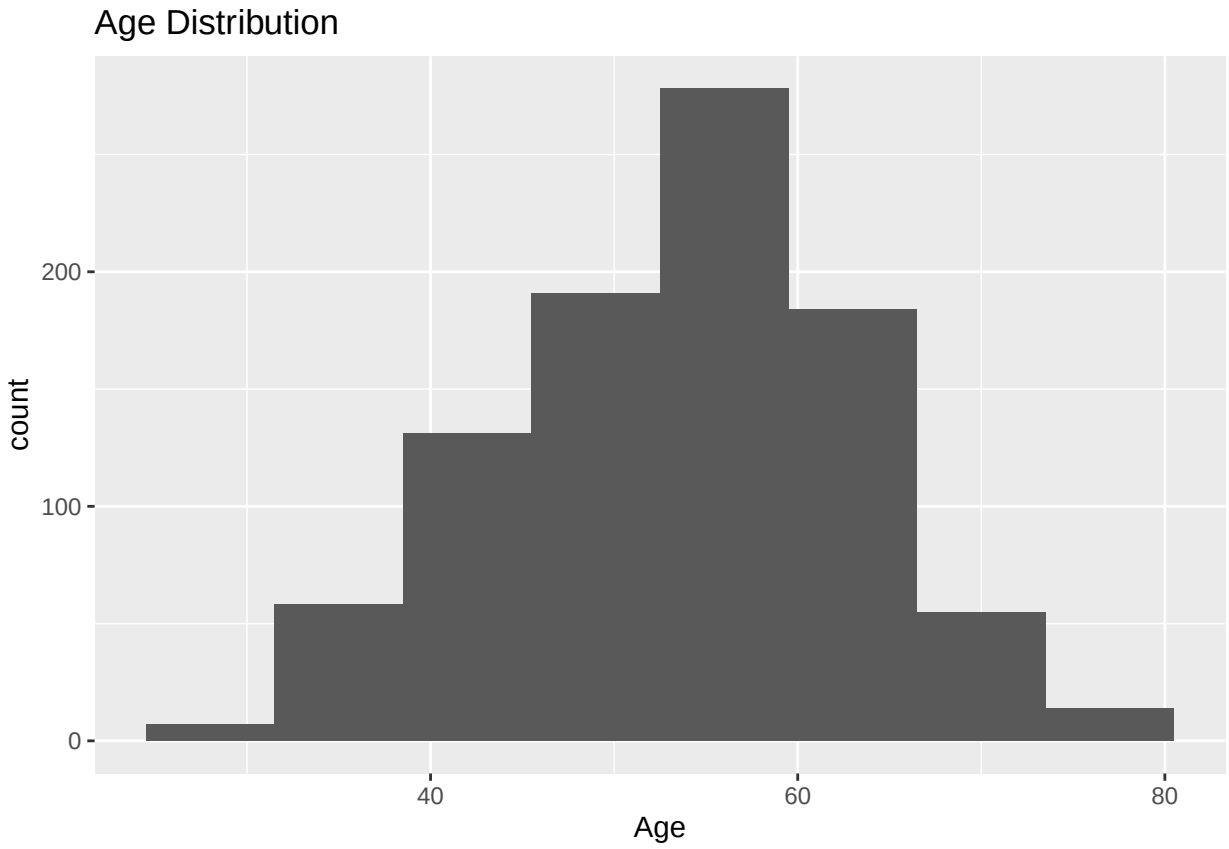
```
# Count heart disease occurrences by Chest Pain Type
heart %>% group_by(ChestPainType) %>% count(HeartDisease)
```

```
## # A tibble: 8 x 3
## # Groups:   ChestPainType [4]
##   ChestPainType HeartDisease     n
##   <chr>                <int> <int>
## 1 ASY                      0   104
## 2 ASY                      1   392
## 3 ATA                      0   149
## 4 ATA                      1    24
## 5 NAP                      0   131
## 6 NAP                      1    72
## 7 TA                       0    26
## 8 TA                       1    20
```

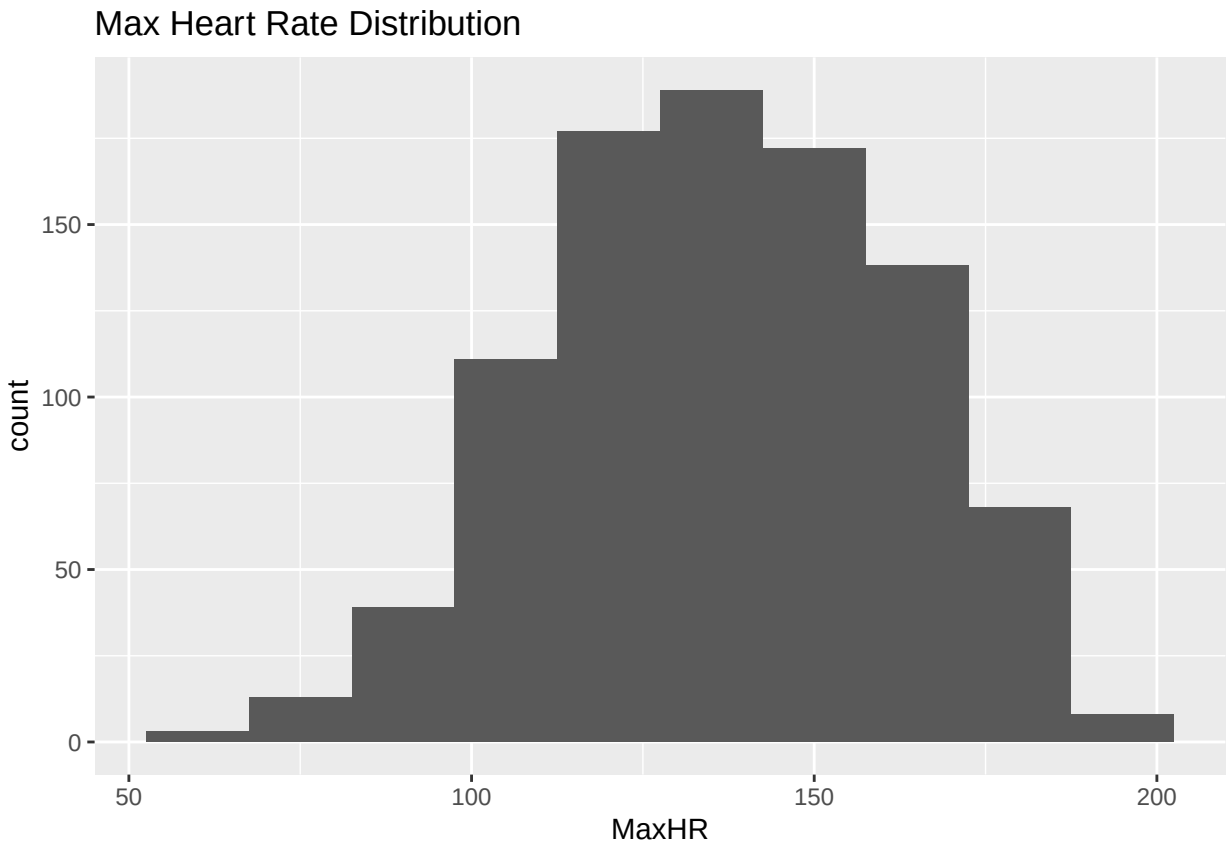
Explanation: Summary statistics provide a foundational overview, while grouped summaries allow us to observe patterns across categorical variables, such as the distribution of heart disease cases by chest pain type or sex. These insights guide data preprocessing and model feature selection.

Visualizing Data Distributions

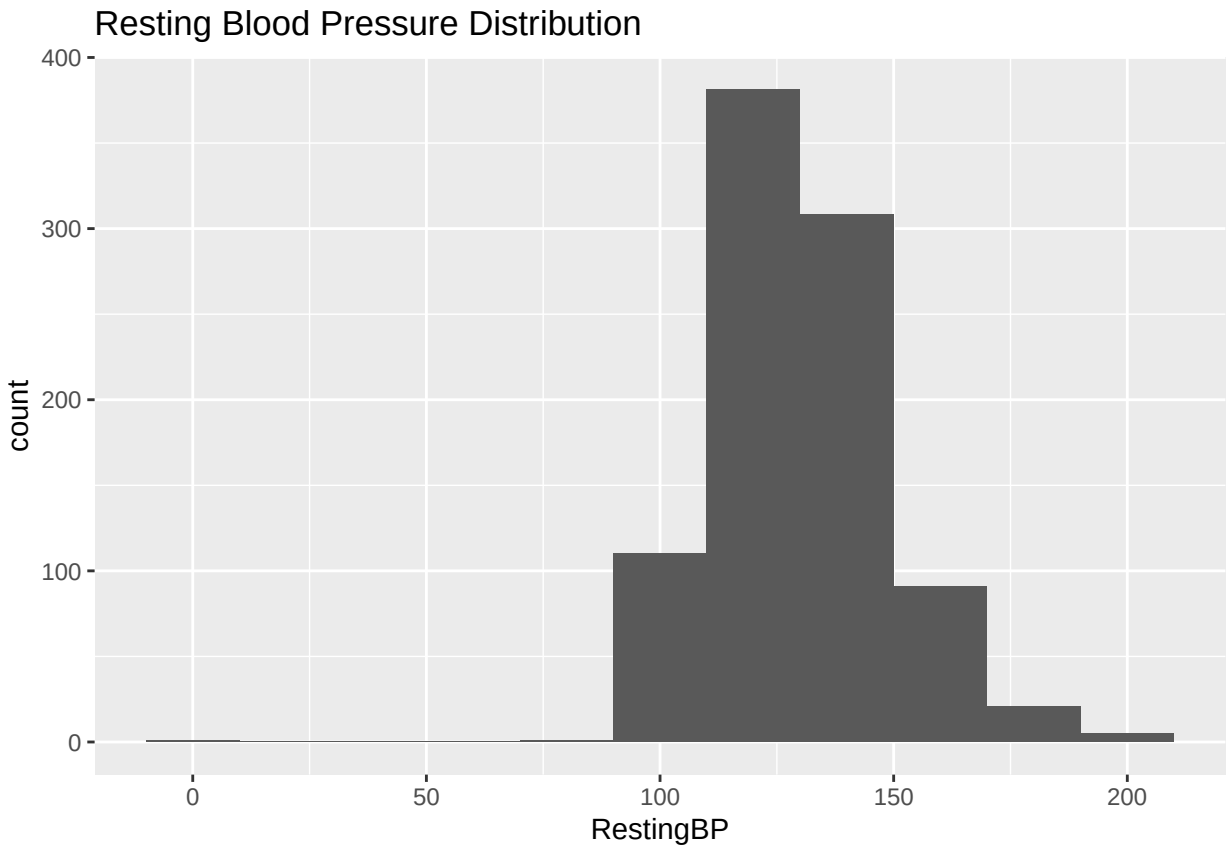
```r
# Plot histograms for key numerical variables to assess distribution patterns
ggplot(heart, aes(Age)) + geom_histogram(binwidth = 7) + ggtitle("Age Distribution")
```



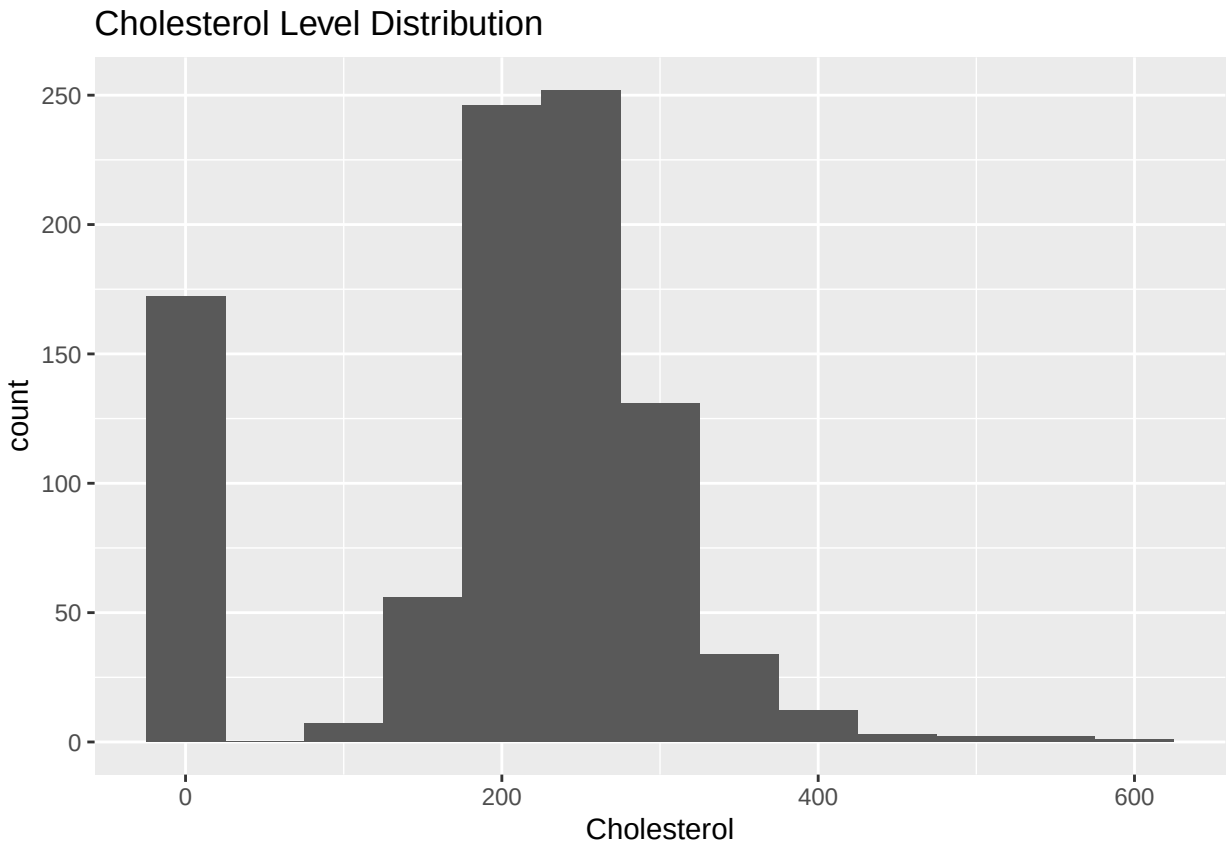

```r
ggplot(heart, aes(MaxHR)) + geom_histogram(binwidth = 15) + ggtitle("Max Heart Rate Distribution")
```



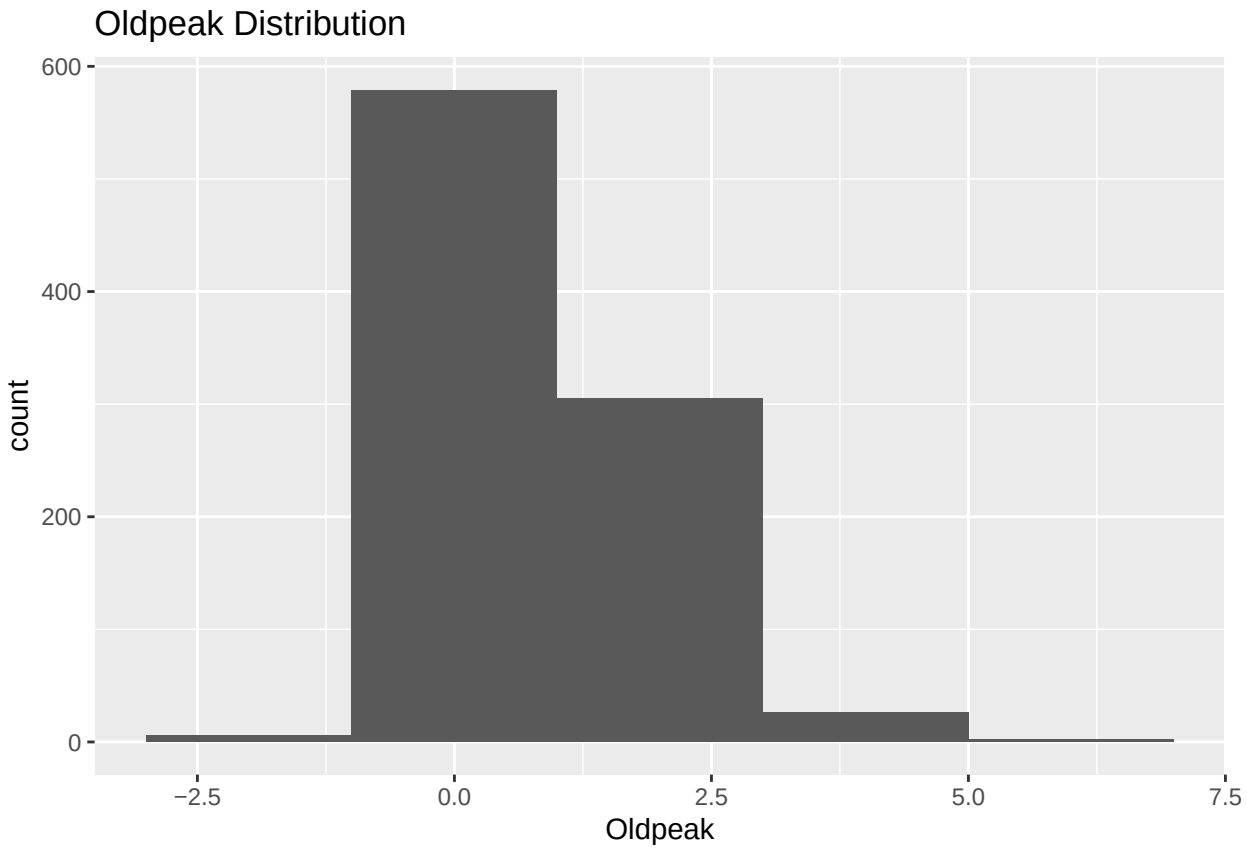

```
ggplot(heart, aes(RestingBP)) + geom_histogram(binwidth = 20) + ggtitle("Resting Blood Pressure Distribu
```




```
ggplot(heart, aes(Cholesterol)) + geom_histogram(binwidth = 50) + ggtitle("Cholesterol Level Distributi
```

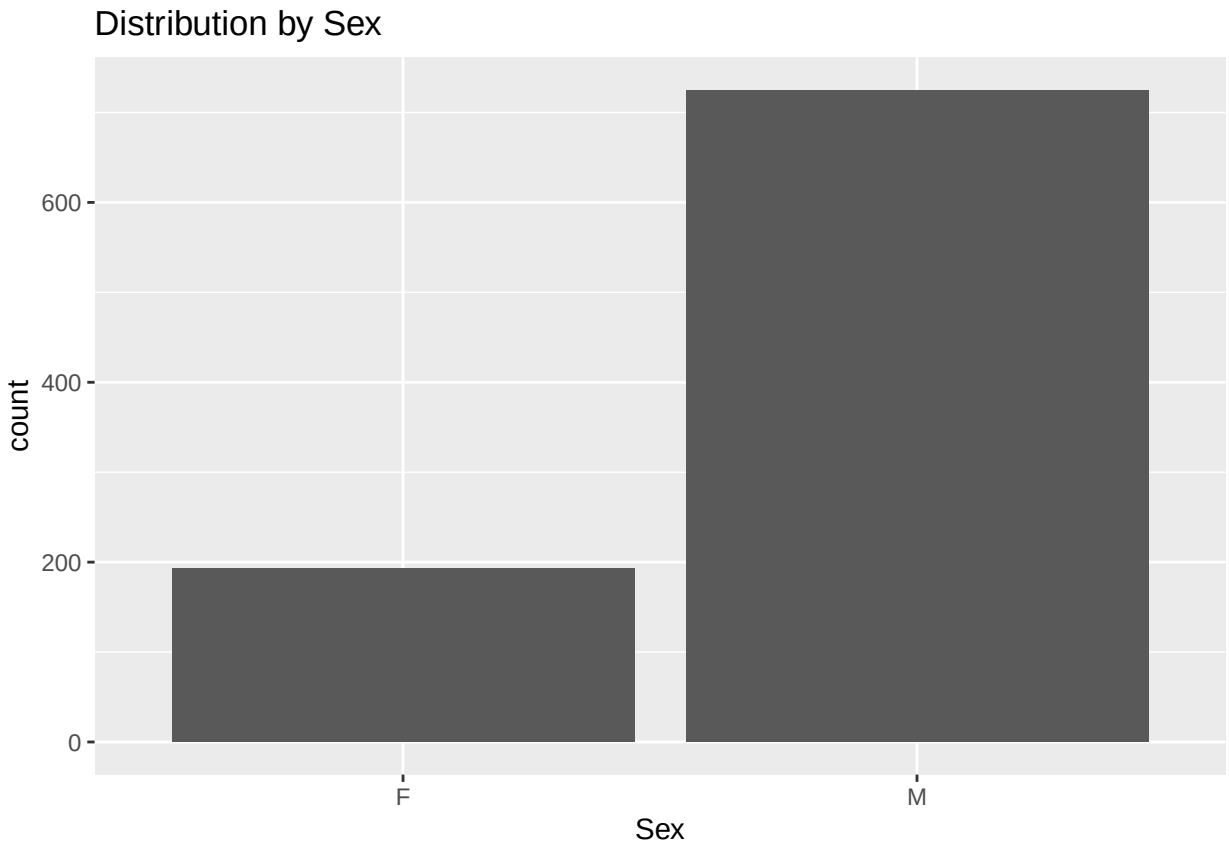



```
ggplot(heart, aes(Oldpeak)) + geom_histogram(binwidth = 2) + ggtitle("Oldpeak Distribution")
```

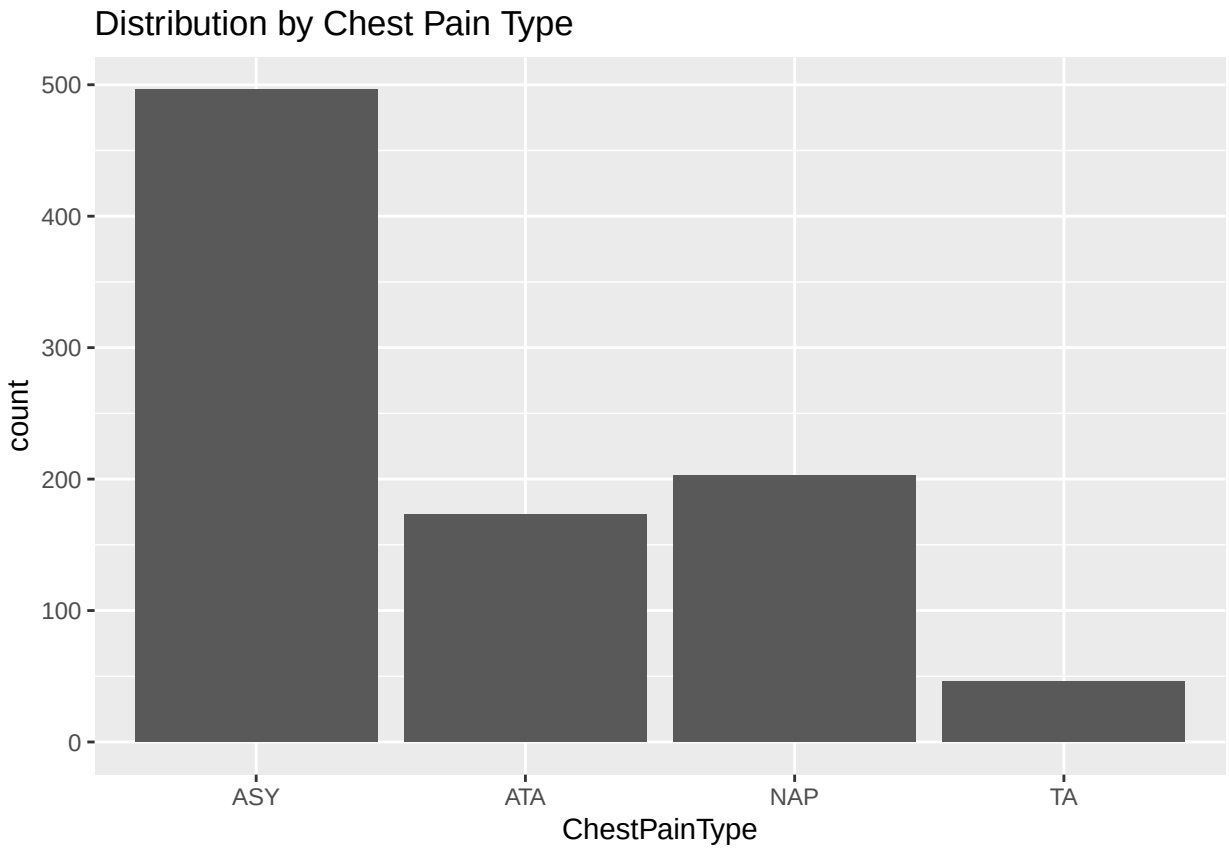


Bar Plots for Categorical Variables

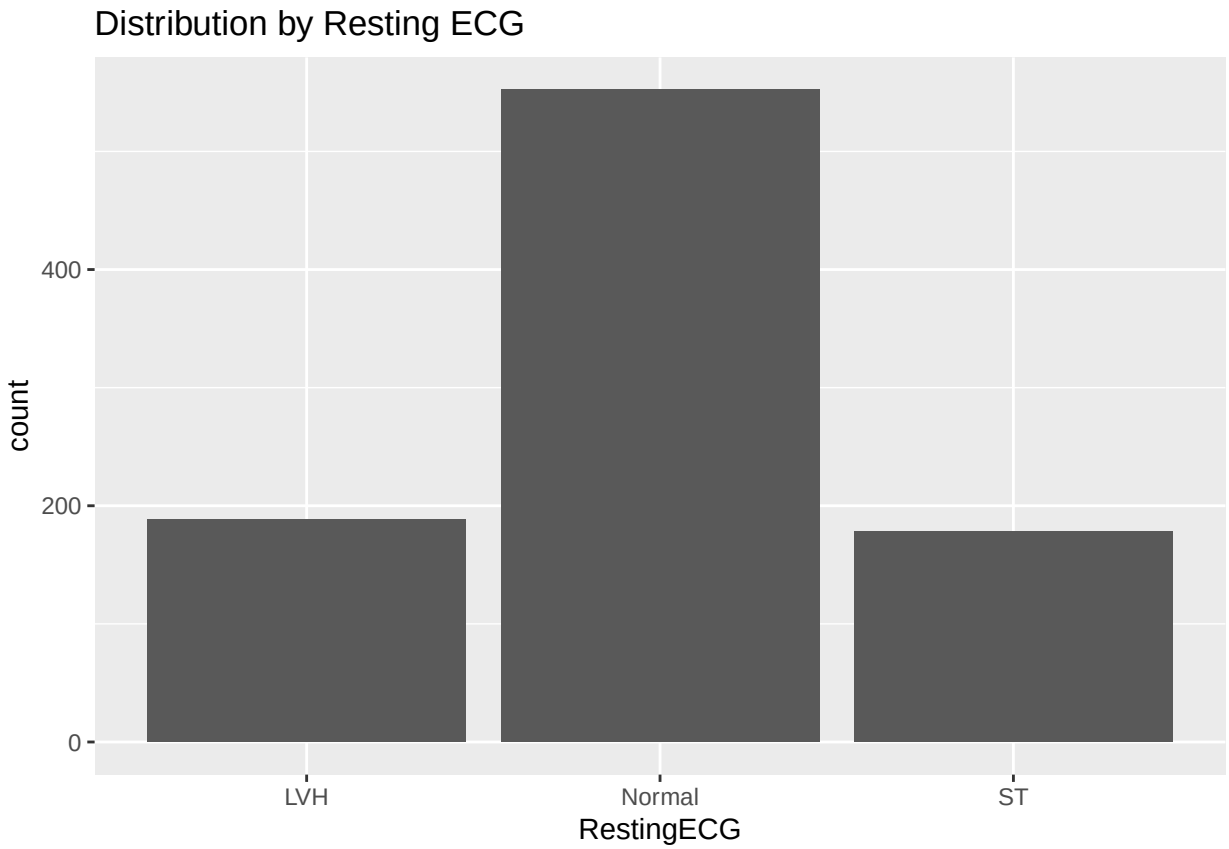
```
# Bar plots for categorical features to explore counts across categories
ggplot(heart, aes(x = Sex)) + geom_bar() + ggtitle("Distribution by Sex")
```



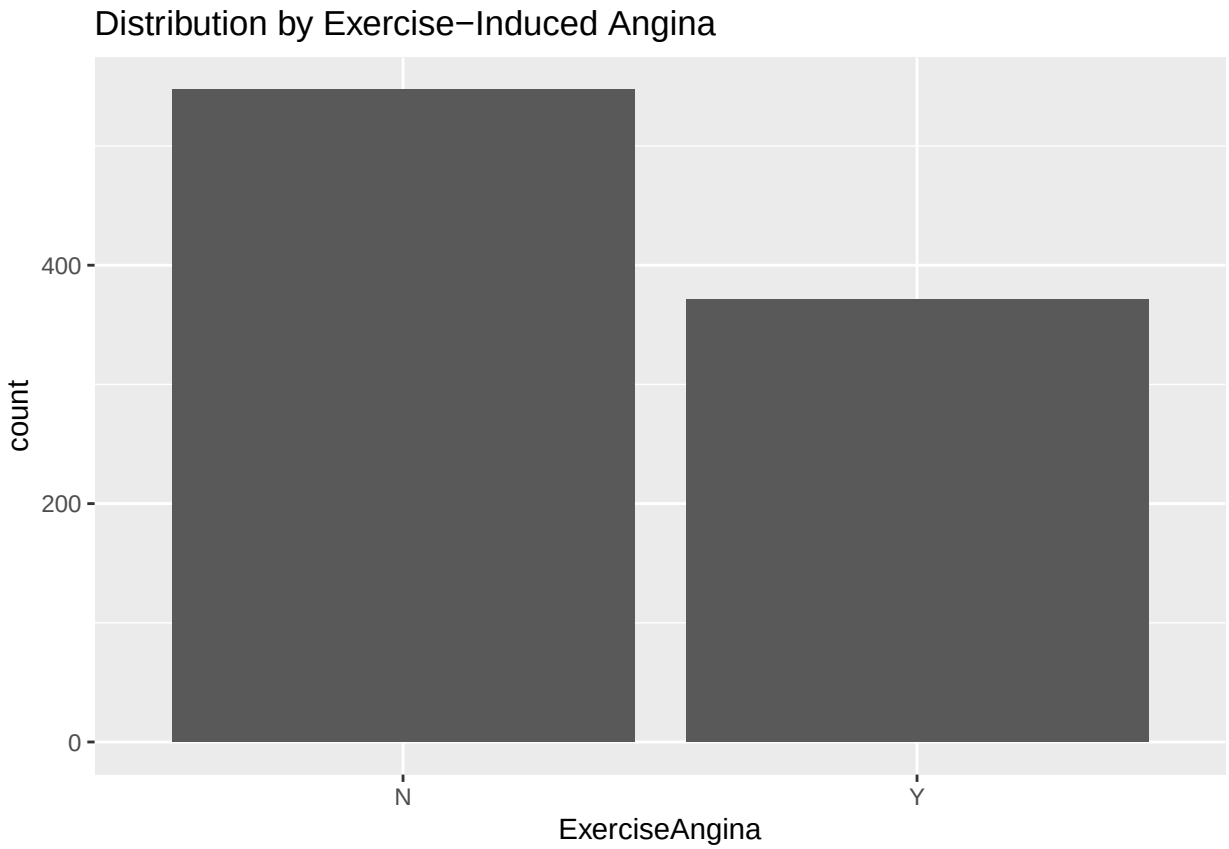

```
ggplot(heart, aes(x = ChestPainType)) + geom_bar() + ggtitle("Distribution by Chest Pain Type")
```



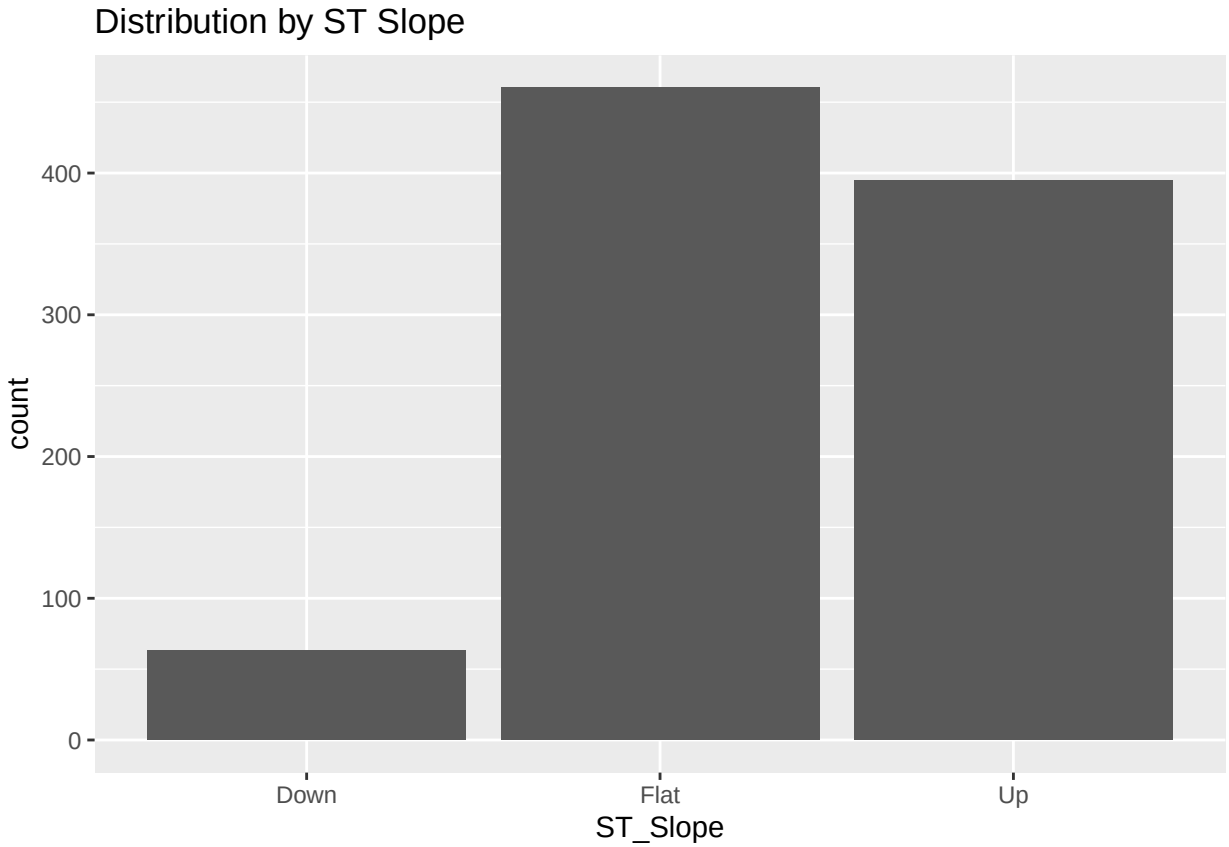

```
ggplot(heart, aes(x = RestingECG)) + geom_bar() + ggtitle("Distribution by Resting ECG")
```




```
ggplot(heart, aes(x = ExerciseAngina)) + geom_bar() + ggtitle("Distribution by Exercise-Induced Angina"
```




```
ggplot(heart, aes(x = ST_Slope)) + geom_bar() + ggtitle("Distribution by ST Slope")
```



Explanation: Histograms and bar plots help visualize the spread of numerical and categorical variables, making it easier to identify outliers, skewness, and class imbalances. This exploration is key for feature engineering and model preparation.

(C)DATA CLEANING

Let's do some data cleaning to better understand our data.

```
# Check for missing values in each column
colSums(is.na(heart))
```

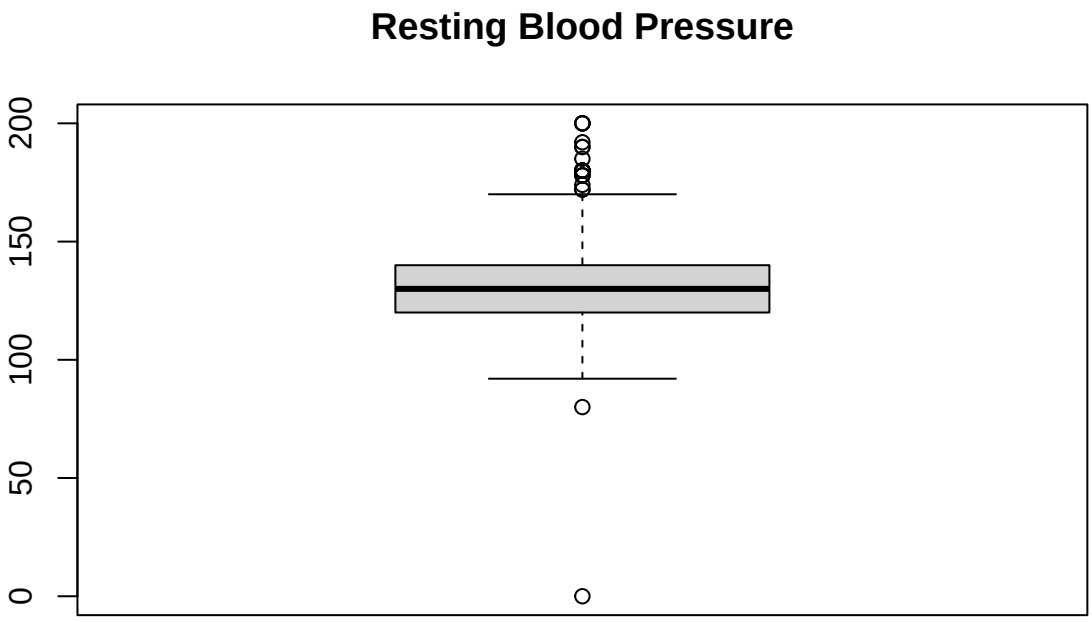
```
##            Age            Sex  ChestPainType      RestingBP    Cholesterol
##              0              0              0              0              0
##      FastingBS     RestingECG          MaxHR ExerciseAngina        Oldpeak
##              0              0              0              0              0
##       ST_Slope   HeartDisease
##              0              0
```

It seems we don't have any missing values in our data.

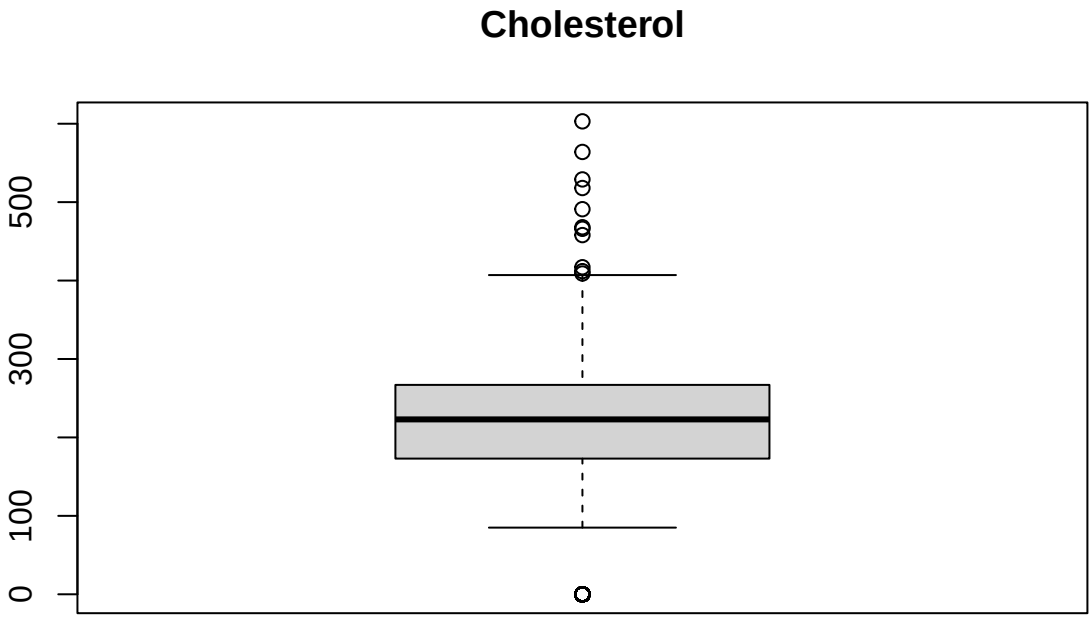
Explanation: This step identifies any missing values, allowing us to decide on imputation or removal methods. Here, we're confirming that no columns contain missing data before moving to further cleaning.

Outlier Detection Using Box Plots

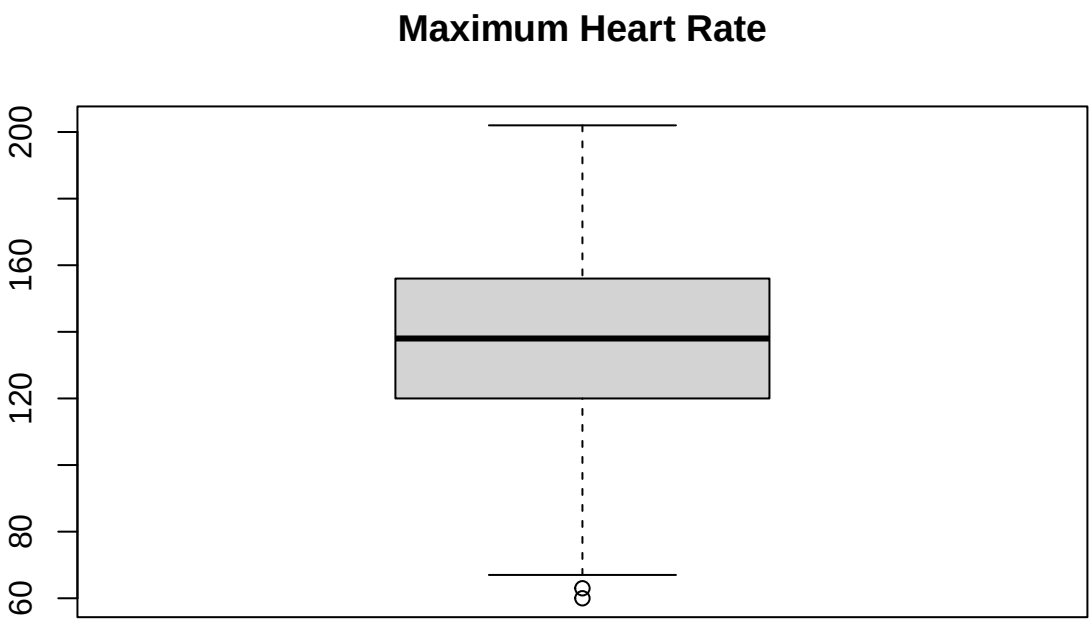
```
# Visualize outliers in numerical columns using box plots
boxplot(heart$RestingBP, main = "Resting Blood Pressure", xlab = "RestingBP")
```



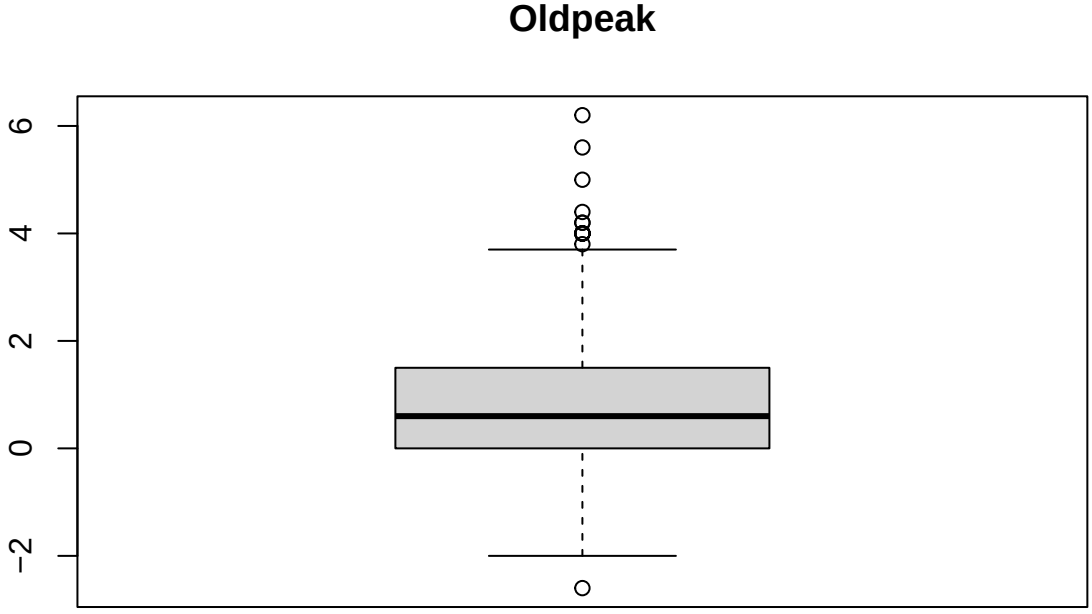

```
boxplot(heart$Cholesterol, main = "Cholesterol", xlab = "Cholesterol")
```



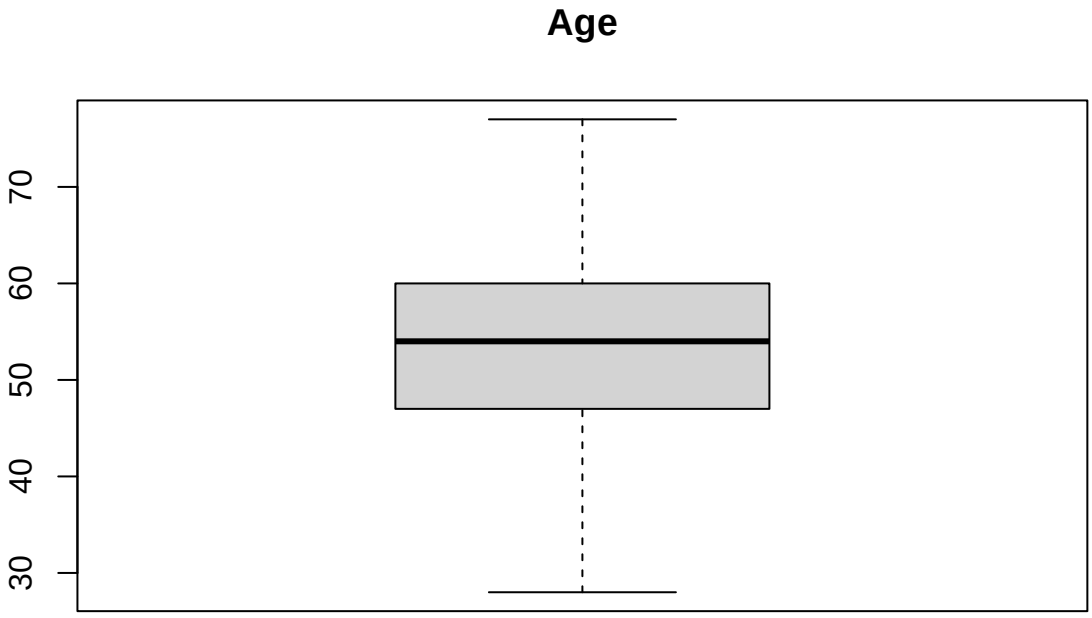

```
boxplot(heart$MaxHR, main = "Maximum Heart Rate", xlab = "MaxHR")
```




```
boxplot(heart$Oldpeak, main = "Oldpeak", xlab = "Oldpeak")
```




```
boxplot(heart$Age, main = "Age", xlab = "Age")
```



Explanation: Box plots are useful for spotting outliers in numerical data. Outliers may skew the analysis, so it's important to identify them here for potential removal.

Removing Outliers Based on IQR

```
# Remove outliers for each numerical column using the IQR method
remove_outliers <- function(data, column) {
  Q1 <- quantile(data[[column]], 0.25)
  Q3 <- quantile(data[[column]], 0.75)
  IQR_value <- Q3 - Q1
  subset(data, data[[column]] > (Q1 - 1.5 * IQR_value) & data[[column]] < (Q3 + 1.5 * IQR_value))
}

heart <- remove_outliers(heart, "RestingBP")
heart <- remove_outliers(heart, "Cholesterol")
heart <- remove_outliers(heart, "MaxHR")
heart <- remove_outliers(heart, "Oldpeak")
heart <- remove_outliers(heart, "Age")

# Check updated dimensions after outlier removal
dim(heart)
```

```
## [1] 690  12
```

Explanation: Using the IQR method, we filter out data points that fall outside of 1.5 times the IQR. This helps create a cleaner dataset by reducing noise from extreme values, which could otherwise distort the model.

Converting Categorical Variables to Numeric

```
# Convert ExerciseAngina from a character to an integer (0,1)
heart$ExerciseAngina <- ifelse(heart$ExerciseAngina == "Y", 1, 0)
head(heart)
```

```
##   Age Sex ChestPainType RestingBP Cholesterol FastingBS RestingECG MaxHR
## 1  40   M           ATA       140         289         0     Normal   172
## 2  49   F           NAP       160         180         0     Normal   156
## 3  37   M           ATA       130         283         0         ST    98
## 4  48   F           ASY       138         214         0     Normal   108
## 5  54   M           NAP       150         195         0     Normal   122
## 6  39   M           NAP       120         339         0     Normal   170
##   ExerciseAngina Oldpeak ST_Slope HeartDisease
## 1              0     0.0       Up            0
## 2              0     1.0     Flat            1
## 3              0     0.0       Up            0
## 4              1     1.5     Flat            1
## 5              0     0.0       Up            0
## 6              0     0.0       Up            0
```

Explanation: Some machine learning algorithms work best with numeric data. Here, converting ExerciseAngina to binary (0 and 1) allows us to incorporate it effectively in the modeling process.

(D)DATA PREPROCESSING

Let's start with Data preprocessing

```
# Categorize cholesterol levels into bins
heart <- heart %>% mutate(ChRange = cut(Cholesterol, breaks = c(-1, 150, 200, 500), labels = c("Normal"
head(heart)
```

```
##   Age Sex ChestPainType RestingBP Cholesterol FastingBS RestingECG MaxHR
## 1  40   M           ATA       140         289         0     Normal   172
## 2  49   F           NAP       160         180         0     Normal   156
## 3  37   M           ATA       130         283         0         ST    98
## 4  48   F           ASY       138         214         0     Normal   108
## 5  54   M           NAP       150         195         0     Normal   122
## 6  39   M           NAP       120         339         0     Normal   170
##   ExerciseAngina Oldpeak ST_Slope HeartDisease        ChRange
## 1              0     0.0       Up            0           High
## 2              0     1.0     Flat            1 BorderlineHigh
## 3              0     0.0       Up            0           High
## 4              1     1.5     Flat            1           High
## 5              0     0.0       Up            0 BorderlineHigh
## 6              0     0.0       Up            0           High
```

Explanation: Binning Cholesterol into categories (e.g., Normal, Borderline High, High) simplifies the data and may reveal more interpretable patterns, which can be helpful for model insights.

Normalizing Data Using Standardization

```
# Normalize numerical features
heart_norm <- heart %>% select(-HeartDisease)
preprocess <- preProcess(heart_norm, method = c("center", "scale"))
```

```r
norm <- predict(preprocess, heart_norm)
norm$HeartDisease <- heart$HeartDisease
```

Explanation: Standardizing the dataset ensures that features have a mean of 0 and a standard deviation of 1, making them comparable in scale. This step is crucial for models sensitive to feature magnitudes, such as SVM.

Check Summary Statistics for Normalization

```r
# Summary statistics to confirm normalization
summary(norm)
```

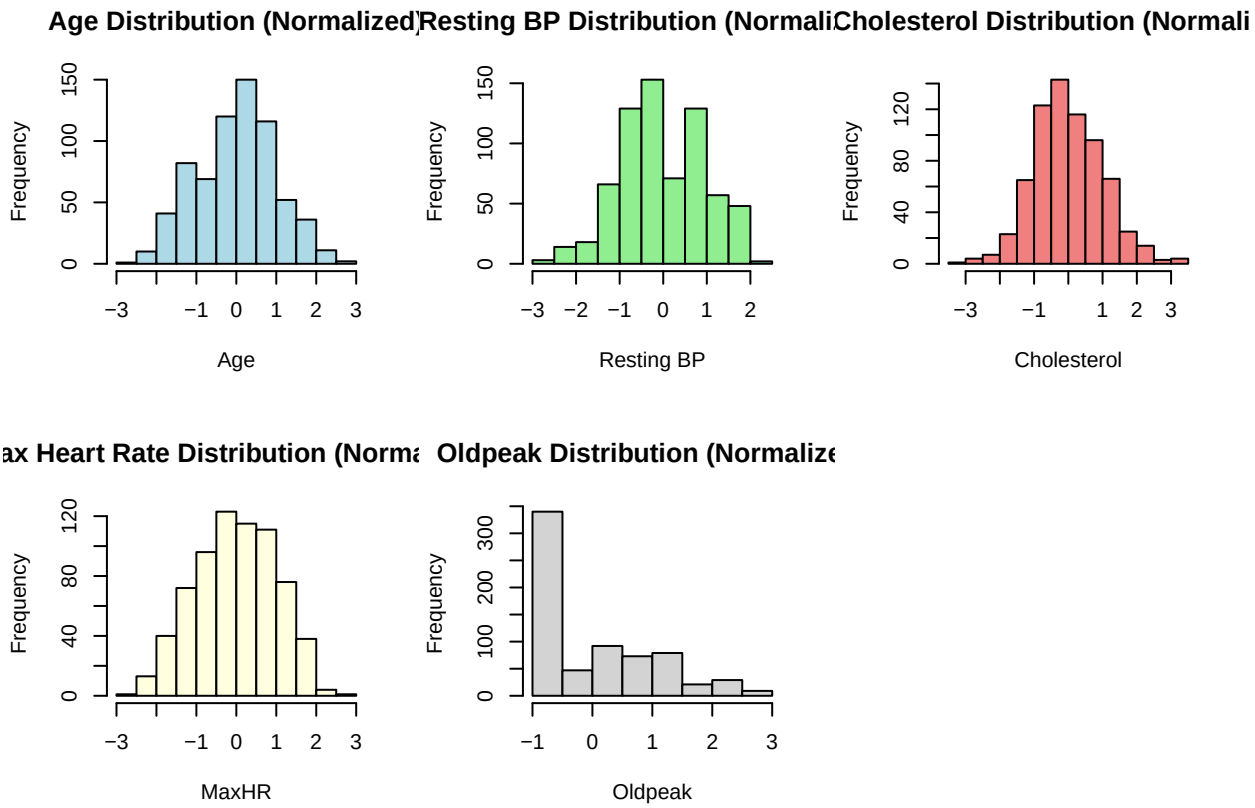
```
##       Age               Sex             ChestPainType        RestingBP
##  Min.   :-2.5976   Length:690         Length:690         Min.   :-2.64760
##  1st Qu.:-0.6987   Class :character   Class :character   1st Qu.:-0.74130
##  Median : 0.1452   Mode  :character   Mode  :character   Median :-0.06048
##  Mean   : 0.0000                                        Mean   : 0.00000
##  3rd Qu.: 0.6727                                        3rd Qu.: 0.62034
##  Max.   : 2.5716                                        Max.   : 2.32239
##   Cholesterol         FastingBS        RestingECG           MaxHR
##  Min.   :-3.05932   Min.   :-0.4375   Length:690         Min.   :-2.86457
##  1st Qu.:-0.66476   1st Qu.:-0.4375   Class :character   1st Qu.:-0.75897
##  Median :-0.09086   Median :-0.4375   Mode  :character   Median :-0.00917
##  Mean   : 0.00000   Mean   : 0.0000                      Mean   : 0.00000
##  3rd Qu.: 0.64136   3rd Qu.:-0.4375                      3rd Qu.: 0.79199
##  Max.   : 3.25360   Max.   : 2.2822                      Max.   : 2.51755
##  ExerciseAngina       Oldpeak          ST_Slope                  ChRange
##  Min.   :-0.7675   Min.   :-0.9676   Length:690         Normal        : 20
##  1st Qu.:-0.7675   1st Qu.:-0.8623   Class :character   BorderlineHigh:124
##  Median :-0.7675   Median :-0.4413   Mode  :character   High          :546
##  Mean   : 0.0000   Mean   : 0.0000
##  3rd Qu.: 1.3011   3rd Qu.: 0.7165
##  Max.   : 1.3011   Max.   : 2.9269
##   HeartDisease
##  Min.   :0.000
##  1st Qu.:0.000
##  Median :0.000
##  Mean   :0.458
##  3rd Qu.:1.000
##  Max.   :1.000
```

Explanation: Summary statistics allow you to quickly verify that each feature has been standardized. Ideally, the mean should be close to 0 and the standard deviation close to 1 for all numerical columns (excluding categorical or target variables).

Visualize the Distribution of Normalized Features

```r
# Histograms to visually inspect distributions of normalized data
par(mfrow = c(2, 3))  # Set up plot grid for easier viewing

# Plot histograms for key numerical features
hist(norm$Age, main = "Age Distribution (Normalized)", xlab = "Age", col = "lightblue")
hist(norm$RestingBP, main = "Resting BP Distribution (Normalized)", xlab = "Resting BP", col = "lightgr
hist(norm$Cholesterol, main = "Cholesterol Distribution (Normalized)", xlab = "Cholesterol", col = "lig
```

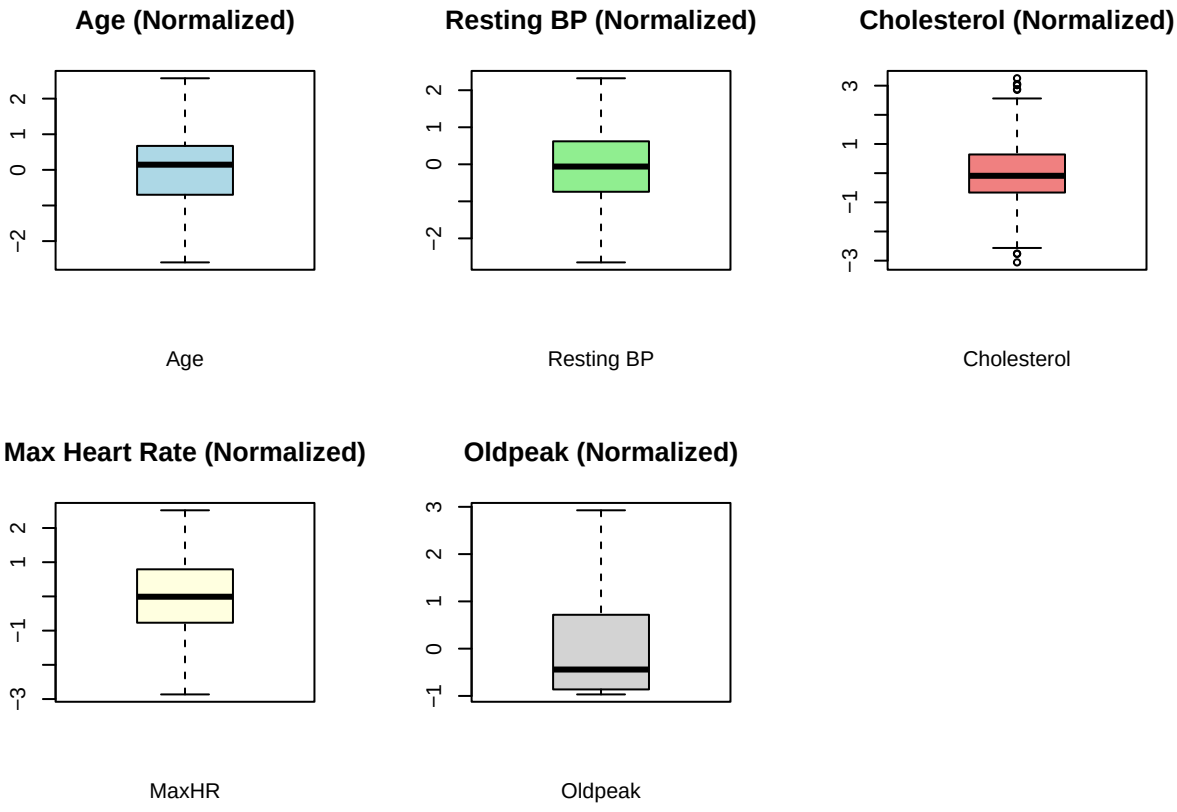
```r
hist(norm$MaxHR, main = "Max Heart Rate Distribution (Normalized)", xlab = "MaxHR", col = "lightyellow")
hist(norm$Oldpeak, main = "Oldpeak Distribution (Normalized)", xlab = "Oldpeak", col = "lightgray")
```



Explanation: Visualizing histograms of normalized features can confirm that they are centered around 0 with a consistent spread, indicating successful standardization. This is helpful to spot any remaining outliers or anomalies visually.

Checking for Outliers After Normalization

```r
# Box plots to confirm outlier removal in key features
par(mfrow = c(2, 3))  # Set up plot grid

# Plot box plots for key numerical features
boxplot(norm$Age, main = "Age (Normalized)", xlab = "Age", col = "lightblue")
boxplot(norm$RestingBP, main = "Resting BP (Normalized)", xlab = "Resting BP", col = "lightgreen")
boxplot(norm$Cholesterol, main = "Cholesterol (Normalized)", xlab = "Cholesterol", col = "lightcoral")
boxplot(norm$MaxHR, main = "Max Heart Rate (Normalized)", xlab = "MaxHR", col = "lightyellow")
boxplot(norm$Oldpeak, main = "Oldpeak (Normalized)", xlab = "Oldpeak", col = "lightgray")
```



Explanation: Box plots are useful to confirm that outliers have been minimized or removed, indicating clean data ready for further analysis. This ensures that extreme values won't disproportionately influence model training.

(E) DATA CLUSTERING

##Let's perform Clustering

Load Additional Libraries for Clustering

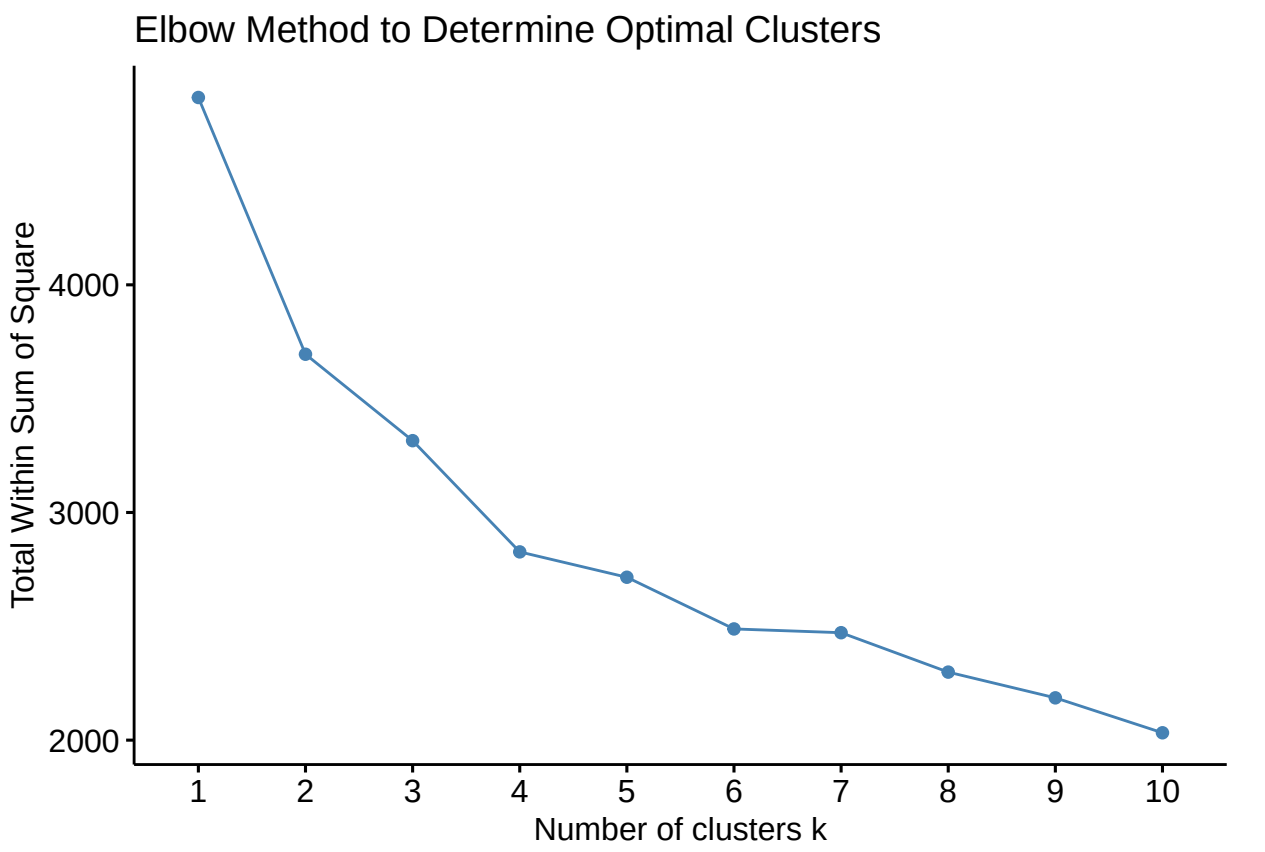
```
# Load libraries for clustering and visualization
library(factoextra)    # For visualization of clustering results
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

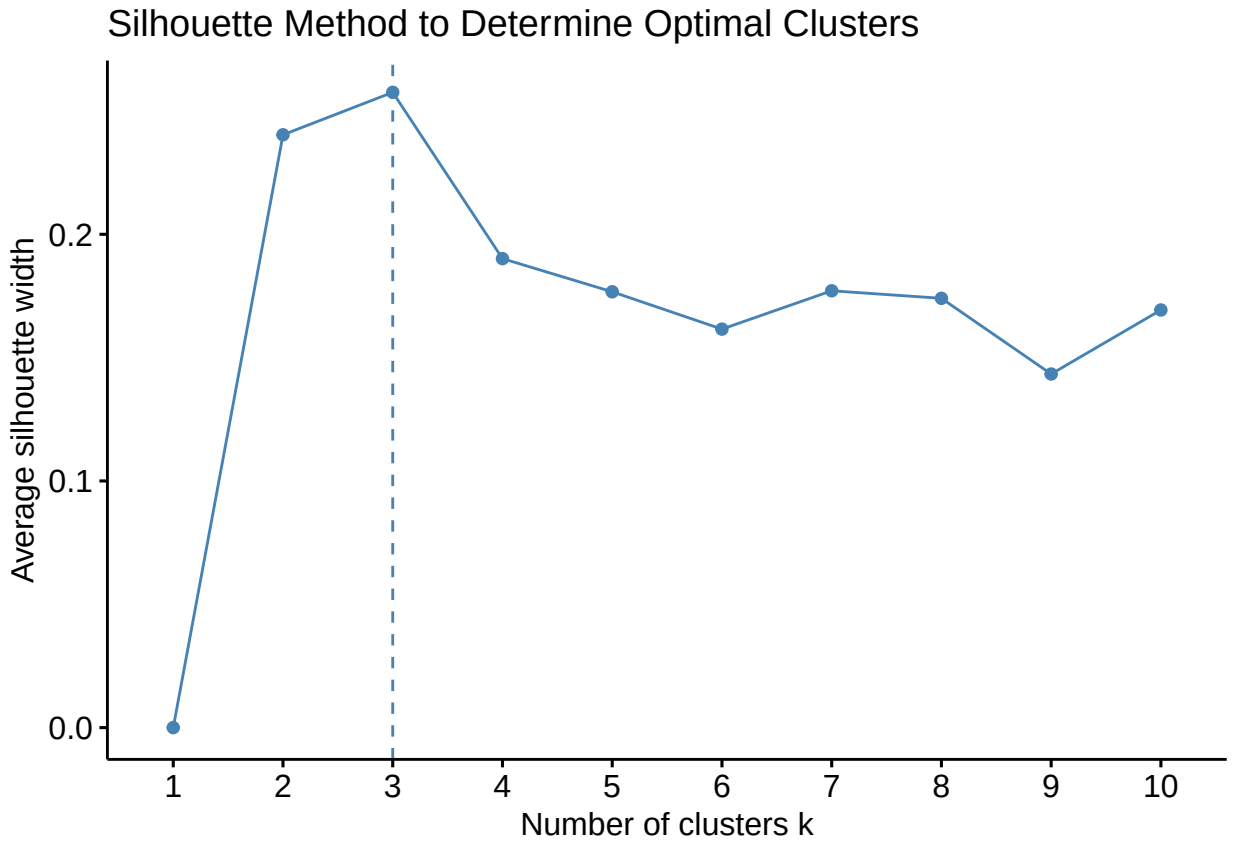
```
library(cluster)       # For advanced clustering algorithms
```

Prepare Data for Clustering and Determine Optimal Number of Clusters

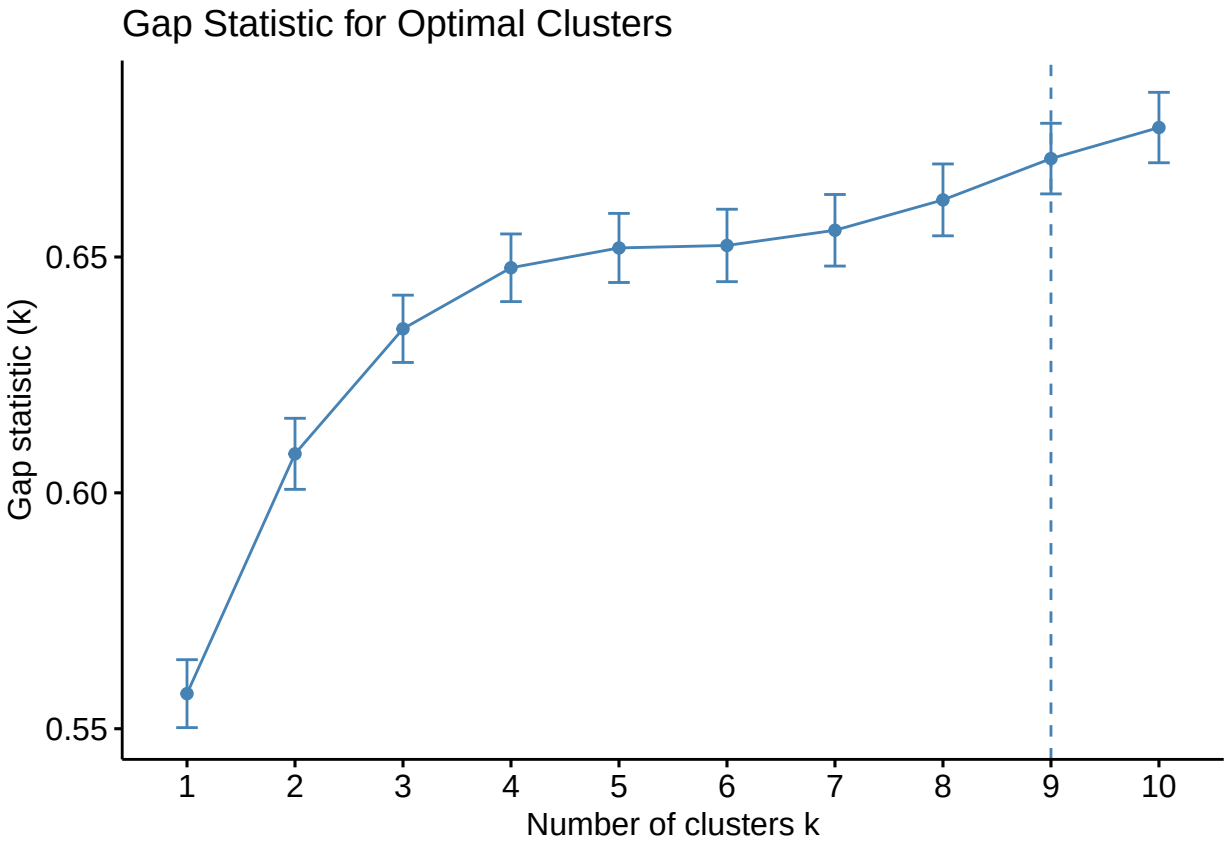
```
# Exclude the target variable 'HeartDisease' and retain only numeric columns for clustering
heart_clustering <- heart %>% select(-HeartDisease) %>% select_if(is.numeric)

# Standardize the data to ensure all features have a similar scale
preprocess <- preProcess(heart_clustering, method = c("center", "scale"))
heart_scaled <- predict(preprocess, heart_clustering)

# Use the Elbow Method to estimate the optimal number of clusters
fviz_nbclust(heart_scaled, kmeans, method = "wss") + ggtitle("Elbow Method to Determine Optimal Clusters
```




```
# Use the Silhouette Method for an alternative cluster estimation
fviz_nbclust(heart_scaled, kmeans, method = "silhouette") + ggtitle("Silhouette Method to Determine Opti
```




```
# Calculate the Gap Statistic as a third measure for optimal clusters
# Increase maximum iterations for kmeans within clusGap to improve convergence
gap_stat <- clusGap(heart_scaled, FUN = function(x, k) kmeans(x, k, nstart = 25, iter.max = 50), K.max =
fviz_gap_stat(gap_stat) + ggtitle("Gap Statistic for Optimal Clusters")
```



Explanation: Here, the dataset is standardized again (though already normalized earlier) to ensure consistency for clustering. We use the elbow, silhouette, and gap statistic methods to determine the optimal number of clusters, providing multiple perspectives to confirm the best choice.

Applying K-means Clustering

```
# Set a random seed for reproducibility
set.seed(13)

# Apply K-means clustering with 4 clusters
kmeans_model <- kmeans(heart_scaled, centers = 3, nstart = 25)
kmeans_model
```

```
## K-means clustering with 3 clusters of sizes 356, 111, 223
##
## Cluster means:
##          Age  RestingBP Cholesterol  FastingBS      MaxHR ExerciseAngina
## 1 -0.4334913 -0.2262909 -0.08552406 -0.4375294  0.4889275     -0.6744986
## 2  0.5311079  0.4007559  0.07494512  2.2822481 -0.2488318      0.2388614
## 3  0.4276678  0.1617743  0.09922716 -0.4375294 -0.6566720      0.9578829
##      Oldpeak
## 1 -0.5453829
## 2  0.1503961
## 3  0.7957953
##
## Clustering vector:
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
```

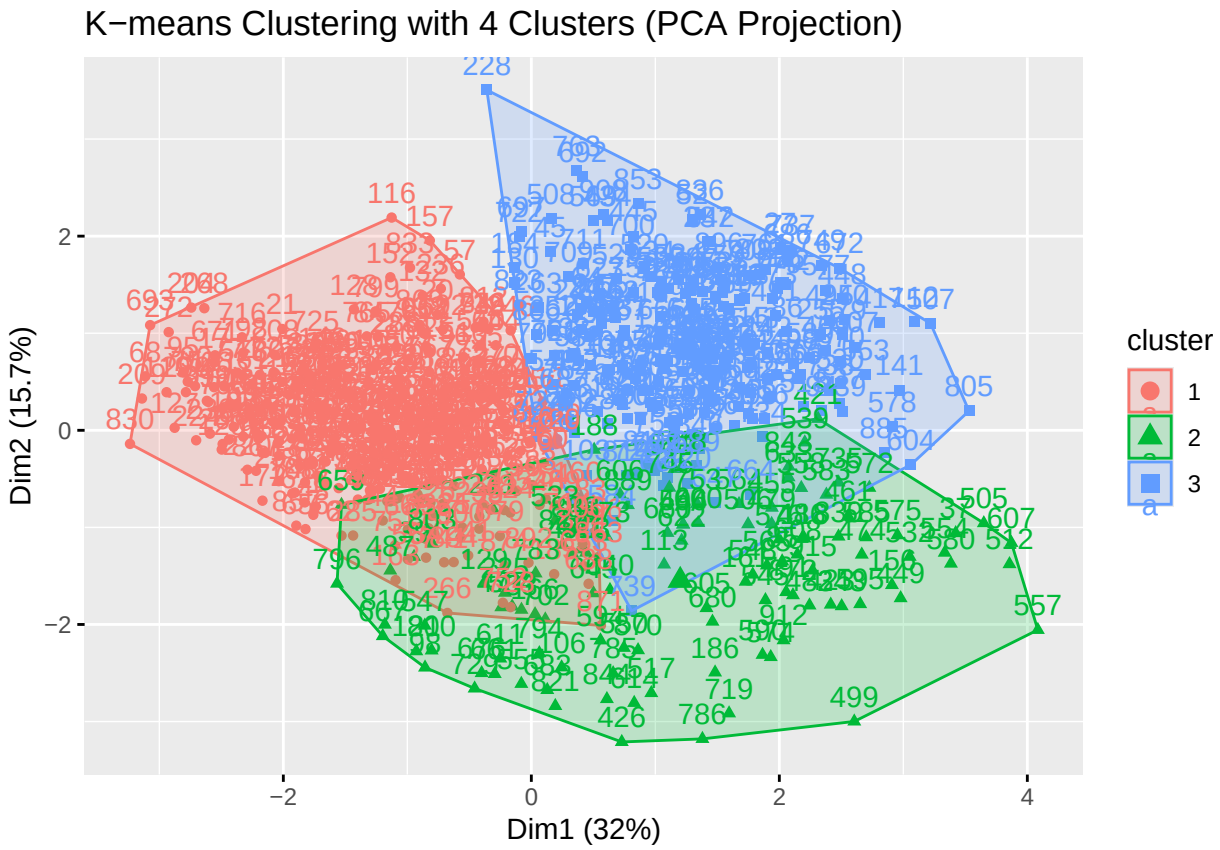
```
##   1   1   1   3   1   1   1   1   3   1   1   3   1   3   1   1   1   1   1   1
##  21  22  23  24  25  26  27  28  30  32  33  34  35  36  37  38  39  40  41  42
##   1   1   1   3   1   1   3   1   1   1   3   1   1   1   2   1   2   3   1   3
##  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62
##   1   1   3   3   1   1   1   1   3   3   2   1   3   1   1   1   1   3   1   1
##  63  64  65  66  67  68  71  72  73  74  75  76  78  79  80  81  82  83  84  85
##   1   3   1   1   1   1   3   1   1   1   3   1   1   3   1   1   1   1   1   2
##  86  88  89  90  91  92  93  94  95  96  97  98  99 100 101 102 103 105 106 107
##   3   3   1   3   1   1   1   3   1   3   1   2   1   1   3   1   3   1   2   1
## 108 109 111 112 113 114 115 116 117 118 119 120 121 122 123 125 126 127 128 129
##   1   1   1   3   2   1   1   1   1   2   1   1   2   1   1   1   1   1   1   2
## 130 131 132 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 151
##   3   1   1   3   3   1   1   1   3   3   3   3   3   1   1   1   1   1   1   1
## 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 168 169 170 171 172
##   1   1   1   1   2   1   1   3   1   2   3   1   1   1   2   1   1   1   1   1
## 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 193 194 195
##   1   1   3   3   3   1   1   1   3   1   3   3   1   2   1   2   3   1   1   1
## 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
##   1   1   1   3   3   1   1   1   1   1   3   1   1   1   1   2   3   3   1   3
## 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236
##   1   1   1   1   1   3   1   1   2   1   1   3   1   1   1   1   1   1   1   1
## 237 238 239 240 241 243 244 245 246 247 248 249 250 252 253 254 255 256 257 258
##   3   1   3   3   1   2   1   3   1   3   2   3   3   3   3   1   3   1   1   1
## 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 277 280 281 282
##   1   1   1   1   3   1   3   1   3   1   3   1   1   1   3   1   1   1   1   3
## 283 284 285 286 287 288 289 290 291 292 293 417 418 419 420 421 423 426 427 432
##   3   1   1   1   1   1   1   1   1   1   1   3   1   3   3   2   2   2   1   3
## 434 444 445 446 448 449 453 455 461 463 466 469 470 474 477 479 483 486 487 488
##   3   2   3   3   3   2   3   2   2   3   1   3   2   2   2   2   2   2   2   3
## 489 490 491 494 495 496 498 499 500 502 503 504 505 506 507 508 510 511 512 513
##   1   3   3   3   3   3   3   2   3   3   3   2   2   2   3   3   1   1   2   1
## 514 517 518 520 521 523 524 525 526 527 528 529 530 531 532 533 534 535 539 540
##   3   2   3   3   1   3   3   1   1   3   1   3   3   3   2   2   3   3   2   1
## 541 542 543 545 546 547 548 549 550 552 553 554 555 556 557 558 559 561 562 563
##   3   3   3   2   1   2   2   3   2   3   3   2   2   3   2   2   3   1   1   3
## 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583
##   2   2   3   3   3   3   3   3   2   3   2   2   2   3   3   3   2   2   3   2
## 584 585 587 588 589 590 591 592 594 595 596 597 598 599 600 601 602 603 604 605
##   3   2   3   1   3   2   1   1   2   2   2   1   1   3   2   3   3   3   3   2
## 606 607 608 609 610 611 612 613 614 615 616 618 619 620 621 622 623 624 626 627
##   2   2   2   3   1   2   1   2   2   3   3   1   3   3   1   2   3   1   1   3
## 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647
##   1   3   1   3   3   2   3   1   3   1   1   1   2   1   1   1   1   2   1   3
## 648 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 669
##   1   3   2   3   1   1   1   1   1   1   2   2   2   1   1   3   3   3   2   1
## 670 671 672 673 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690
##   1   1   3   2   3   2   3   1   1   2   3   1   2   1   1   3   3   1   3   1
## 691 692 693 694 695 696 697 698 699 700 701 702 704 705 706 707 708 709 710 711
##   3   3   1   1   1   3   3   3   1   3   1   2   1   3   3   1   3   3   1   3
## 712 713 714 715 716 717 718 719 720 721 722 723 724 725 727 728 729 730 731 732
##   1   1   1   1   1   3   1   2   1   1   3   3   3   1   1   1   2   1   1   1
## 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753
##   3   2   1   3   3   3   1   1   3   1   1   2   3   3   1   3   1   3   1   2
## 754 755 756 757 758 759 761 762 763 764 765 766 767 768 769 770 771 773 774 777
```

```
##   1   1   3   1   1   3   3   1   3   3   1   1   1   1   3   1   1   1   1   3
## 778 779 780 782 783 784 785 786 787 788 789 790 791 793 794 795 796 798 799 800
##   1   3   1   1   2   1   2   2   3   3   3   1   2   3   2   1   2   1   1   2
## 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820
##   1   3   2   1   3   1   3   1   1   2   1   1   1   3   3   1   1   3   3   3
## 821 822 823 824 825 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841
##   2   1   3   3   1   2   1   3   1   1   1   1   1   1   3   3   1   2   1   1
## 842 843 844 845 846 847 848 849 850 852 853 854 855 857 858 859 860 861 862 863
##   3   2   2   1   3   1   1   1   3   1   3   1   1   1   1   3   1   3   1   1
## 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 882 883 884
##   3   3   3   1   1   1   2   1   2   3   1   1   1   3   1   1   1   1   1   1
## 885 886 887 888 889 890 892 893 894 895 896 897 898 899 900 903 904 905 906 907
##   3   1   1   2   2   1   1   1   1   1   3   1   3   1   1   1   1   1   1   1
## 908 910 911 912 913 914 915 916 917 918
##   3   3   1   2   3   1   2   3   1   1
##
## Within cluster sum of squares by cluster:
## [1] 1394.080  621.387 1121.555
##  (between_SS / total_SS =  35.0 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

Explanation: We use 3 clusters based on the silhouette method to get the optimal number if clustering analyis (previously experimented with 4 clusters results were around the age of 44, 55, 56, 57 confirming that 3 clusters should work fine) and set nstart = 25 to ensure the clustering algorithm finds a stable result by trying multiple starting configurations.

Visualizing Clusters Using PCA

```r
# Visualizing clusters in 2D using PCA for dimensionality reduction
fviz_cluster(kmeans_model, data = heart_scaled) + ggtitle("K-means Clustering with 4 Clusters (PCA Proj
```



Explanation: PCA reduces the dataset to two principal components, which allows us to visually inspect how well-separated the clusters are. This visualization helps us understand the clustering distribution in a simplified form.

Analyzing Cluster Characteristics

```
# Add cluster assignments to the original dataset (excluding target variable for clarity)
heart_with_clusters <- heart %>% select(-HeartDisease)
heart_with_clusters$Cluster <- kmeans_model$cluster

# Filter only numeric columns for clustering summary
heart_with_clusters_numeric <- heart_with_clusters %>% select_if(is.numeric)

# Calculate the mean of each numeric variable within each cluster
cluster_summary <- aggregate(. ~ Cluster, data = heart_with_clusters_numeric, mean)
cluster_summary
```

```
##   Cluster      Age RestingBP Cholesterol FastingBS    MaxHR ExerciseAngina
## 1       1 48.51404  127.5646    235.2697         0 152.6236     0.04494382
## 2       2 57.65766  136.7748    243.3784         1 134.6667     0.48648649
## 3       3 56.67713  133.2646    244.6054         0 124.7399     0.83408072
##     Oldpeak
## 1 0.3011236
## 2 0.9621622
## 3 1.5753363
```

Explanation: By calculating the mean of each variable within each cluster, we can gain insights into the

typical characteristics of individuals in each cluster. This summary helps in interpreting the differences across clusters, such as variations in cholesterol, age, and heart rate.

(F) DATA CLASSIFICATION

Performing Classification Methods & Splitting Data into Training and Test Sets

Since we're working with the HeartDisease target variable, we'll implement two classifiers—Support Vector Machine (SVM) and K-Nearest Neighbors (KNN)—and tune their hyperparameters to optimize performance.

```
# Load caret for model training and evaluation
library(caret)

# Split data into training (80%) and test (20%) sets
set.seed(123)  # Set seed for reproducibility
index <- createDataPartition(heart$HeartDisease, p = 0.8, list = FALSE)
train_set <- heart[index, ]
test_set <- heart[-index, ]
```

Explanation: We split the dataset into training and testing sets using an 80-20 split. This helps us evaluate model performance on unseen data, providing a realistic estimate of model accuracy.

1. SVM Classifier with Hyperparameter Tuning

```
# Ensure the HeartDisease column is treated as a binary factor for classification
train_set$HeartDisease <- as.factor(train_set$HeartDisease)
test_set$HeartDisease <- as.factor(test_set$HeartDisease)

# Define train control for cross-validation
train_control <- trainControl(method = "cv", number = 10)

# Define a grid for tuning the C parameter in SVM
svm_grid <- expand.grid(C = 10^seq(-3, 3, by = 0.5))

# Train the SVM model using the training set
svm_model <- train(HeartDisease ~ ., data = train_set, method = "svmLinear",
                   trControl = train_control, tuneGrid = svm_grid)

# Output best SVM model parameters
svm_model
```

```
## Support Vector Machines with Linear Kernel
##
## 552 samples
##  12 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 497, 496, 497, 498, 498, 496, ...
## Resampling results across tuning parameters:
##
##   C             Accuracy   Kappa
##   1.000000e-03  0.8659247  0.7313117
##   3.162278e-03  0.8551455  0.7100889
```

```
##   1.000000e-02  0.8550794  0.7102587
##   3.162278e-02  0.8641065  0.7277169
##   1.000000e-01  0.8678427  0.7353058
##   3.162278e-01  0.8606000  0.7207729
##   1.000000e+00  0.8606000  0.7207626
##   3.162278e+00  0.8606000  0.7207051
##   1.000000e+01  0.8606000  0.7207051
##   3.162278e+01  0.8606000  0.7207051
##   1.000000e+02  0.8606000  0.7207051
##   3.162278e+02  0.8606000  0.7207051
##   1.000000e+03  0.8606000  0.7207051
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.1.
```

Explanation: We use a grid search to tune the C parameter, which controls the regularization in the SVM model. A 10-fold cross-validation is used to evaluate the performance of different parameter values, and the best model is selected based on accuracy.

2. K-Nearest Neighbors (KNN) Classifier with Hyperparameter Tuning

```r
# Define a tuning grid for KNN with various values of k and distance metrics
knn_grid <- expand.grid(kmax = 3:10, kernel = c("rectangular", "cos"), distance = 1:3)

# Train the KNN model using the training set
knn_model <- train(HeartDisease ~ ., data = train_set, method = "kknn",
                   trControl = train_control, tuneGrid = knn_grid)

# Output best KNN model parameters
knn_model
```

```
## k-Nearest Neighbors
##
## 552 samples
##  12 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 496, 497, 498, 496, 497, 497, ...
## Resampling results across tuning parameters:
##
##   kmax  kernel       distance  Accuracy   Kappa
##    3    rectangular  1         0.8571922  0.7142642
##    3    rectangular  2         0.8534560  0.7068488
##    3    rectangular  3         0.8625493  0.7243840
##    3    cos          1         0.8278668  0.6548023
##    3    cos          2         0.8298485  0.6584163
##    3    cos          3         0.8170887  0.6326849
##    4    rectangular  1         0.8571922  0.7142642
##    4    rectangular  2         0.8534560  0.7068488
##    4    rectangular  3         0.8625493  0.7243840
##    4    cos          1         0.8461508  0.6914257
##    4    cos          2         0.8406614  0.6807936
```

```
##    4    cos          3         0.8279666  0.6552825
##    5    rectangular  1         0.8517039  0.7033936
##    5    rectangular  2         0.8462169  0.6924518
##    5    rectangular  3         0.8586821  0.7166370
##    5    cos          1         0.8569949  0.7132055
##    5    cos          2         0.8497547  0.6992180
##    5    cos          3         0.8424158  0.6844514
##    6    rectangular  1         0.8516378  0.7032711
##    6    rectangular  2         0.8534921  0.7068173
##    6    rectangular  3         0.8586821  0.7166370
##    6    cos          1         0.8587807  0.7166940
##    6    cos          2         0.8606325  0.7211087
##    6    cos          3         0.8550782  0.7099707
##    7    rectangular  1         0.8425132  0.6849521
##    7    rectangular  2         0.8516402  0.7028792
##    7    rectangular  3         0.8586821  0.7166370
##    7    cos          1         0.8623196  0.7240079
##    7    cos          2         0.8642364  0.7281316
##    7    cos          3         0.8624182  0.7246802
##    8    rectangular  1         0.8425132  0.6849521
##    8    rectangular  2         0.8515103  0.7027670
##    8    rectangular  3         0.8696585  0.7386249
##    8    cos          1         0.8605664  0.7205588
##    8    cos          2         0.8660883  0.7318042
##    8    cos          3         0.8624182  0.7246802
##    9    rectangular  1         0.8443314  0.6885919
##    9    rectangular  2         0.8515103  0.7027670
##    9    rectangular  3         0.8696585  0.7386249
##    9    cos          1         0.8623846  0.7242426
##    9    cos          2         0.8732636  0.7463776
##    9    cos          3         0.8642364  0.7283920
##   10    rectangular  1         0.8532600  0.7059568
##   10    rectangular  2         0.8479064  0.6955919
##   10    rectangular  3         0.8623858  0.7243284
##   10    cos          1         0.8678740  0.7351834
##   10    cos          2         0.8714454  0.7426263
##   10    cos          3         0.8624182  0.7248343
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were kmax = 9, distance = 2 and kernel
##  = cos.
```

Explanation: We tune the k parameter (number of neighbors), kernel (shape of the decision boundary), and distance metric in KNN. This ensures that we find the optimal configuration for accurately predicting heart disease.

Comparing Classifier Performance on Test Set

```r
# Predict on the test set using the tuned SVM model
svm_predictions <- predict(svm_model, test_set)

# Predict on the test set using the tuned KNN model
knn_predictions <- predict(knn_model, test_set)
```

```r
# Confusion matrix and accuracy for SVM
svm_conf_matrix <- confusionMatrix(svm_predictions, test_set$HeartDisease)
svm_conf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 61  8
##          1 18 51
##
##                Accuracy : 0.8116
##                  95% CI : (0.7363, 0.8731)
##     No Information Rate : 0.5725
##     P-Value [Acc > NIR] : 2.334e-09
##
##                   Kappa : 0.6232
##
##  Mcnemar's Test P-Value : 0.07756
##
##             Sensitivity : 0.7722
##             Specificity : 0.8644
##          Pos Pred Value : 0.8841
##          Neg Pred Value : 0.7391
##              Prevalence : 0.5725
##          Detection Rate : 0.4420
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.8183
##
##        'Positive' Class : 0
##
```

```r
# Confusion matrix and accuracy for KNN
knn_conf_matrix <- confusionMatrix(knn_predictions, test_set$HeartDisease)
knn_conf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 59  6
##          1 20 53
##
##                Accuracy : 0.8116
##                  95% CI : (0.7363, 0.8731)
##     No Information Rate : 0.5725
##     P-Value [Acc > NIR] : 2.334e-09
##
##                   Kappa : 0.6263
##
##  Mcnemar's Test P-Value : 0.01079
##
##             Sensitivity : 0.7468
##             Specificity : 0.8983
##          Pos Pred Value : 0.9077
```

```
##          Neg Pred Value : 0.7260
##              Prevalence : 0.5725
##          Detection Rate : 0.4275
##    Detection Prevalence : 0.4710
##       Balanced Accuracy : 0.8226
##
##        'Positive' Class : 0
##
```

Explanation of KNN Model Confusion Matrix Results

The KNN model achieved an accuracy of 81.16% on the test set, indicating that it correctly classified approximately 81% of the instances. Let's break down the key metrics from the confusion matrix and what they imply about the model's performance:

1.Confusion Matrix Summary:

- True Negatives (TN): 59 cases were correctly predicted as "0" (no heart disease).
- False Positives (FP): 6 cases were incorrectly predicted as "0" when they actually had heart disease.
- False Negatives (FN): 20 cases were incorrectly predicted as "1" when they did not have heart disease.
- True Positives (TP): 53 cases were correctly predicted as "1" (heart disease).

2.Accuracy (0.8116):

- Accuracy represents the proportion of correctly classified cases out of all cases. Here, 81.16% of predictions were correct. This is a solid performance, as it is significantly higher than the No Information Rate (NIR) of 57.25%, which is the accuracy expected by random chance.

3.Kappa (0.6263):

- Kappa measures the agreement between predicted and actual classifications, adjusting for chance. A Kappa value of 0.6263 suggests moderate to substantial agreement, indicating that the model has a good ability to differentiate between classes.

4.Sensitivity (0.7468):

- Sensitivity, or True Positive Rate, measures the model's ability to correctly identify cases of no heart disease (class "0"). With a sensitivity of 74.68%, the model successfully identifies most negative cases but misses some.

5.Specificity (0.8983):

- Specificity, or True Negative Rate, measures the model's ability to correctly identify cases of heart disease (class "1"). At 89.83%, the model is highly accurate in predicting true positive cases, indicating strong performance in identifying heart disease.

6.Positive Predictive Value (0.9077):

- Also known as precision, this metric indicates that when the model predicts "0" (no heart disease), it is correct 90.77% of the time. This high precision indicates that the model is reliable when predicting the absence of heart disease.

7.Negative Predictive Value (0.7260):

- The Negative Predictive Value shows that when the model predicts "1" (heart disease), it is correct 72.60% of the time. This is slightly lower, which could indicate that the model occasionally misses heart disease cases.

8.Balanced Accuracy (0.8226):

- Balanced Accuracy is the average of Sensitivity and Specificity, accounting for any imbalance in class distribution. At 82.26%, this metric confirms that the model maintains consistent performance across both classes.

9.McNemar's Test (P-Value: 0.01079):

- McNemar's test checks if there's a significant difference in the model's ability to classify the two classes correctly. A p-value of 0.01079 indicates a significant difference, suggesting that the model might slightly favor one class over the other.

(G)DATA EVALUATION

Choosing the Best Classifier by Comparing Metrics

Since we've already seen the confusion matrix for each model, we can use precision, recall, and the area under the ROC curve (AUC) to further compare their performance. Here's how to calculate these metrics and plot the ROC curve.

```
# Precision, Recall, and F1 Score for SVM
svm_precision <- svm_conf_matrix$byClass["Pos Pred Value"]
svm_recall <- svm_conf_matrix$byClass["Sensitivity"]
svm_f1 <- 2 * ((svm_precision * svm_recall) / (svm_precision + svm_recall))

cat("SVM Precision:", svm_precision, "\n")
```

```
## SVM Precision: 0.884058
```

```
cat("SVM Recall:", svm_recall, "\n")
```

```
## SVM Recall: 0.7721519
```

```
cat("SVM F1 Score:", svm_f1, "\n")
```

```
## SVM F1 Score: 0.8243243
```

```
# Precision, Recall, and F1 Score for KNN
knn_precision <- knn_conf_matrix$byClass["Pos Pred Value"]
knn_recall <- knn_conf_matrix$byClass["Sensitivity"]
knn_f1 <- 2 * ((knn_precision * knn_recall) / (knn_precision + knn_recall))

cat("KNN Precision:", knn_precision, "\n")
```

```
## KNN Precision: 0.9076923
```

```
cat("KNN Recall:", knn_recall, "\n")
```

```
## KNN Recall: 0.7468354
```

```
cat("KNN F1 Score:", knn_f1, "\n")
```

```
## KNN F1 Score: 0.8194444
```

1.Precision:

- SVM Precision: 0.8676
- KNN Precision: 0.9077
- Interpretation: Precision measures how often the model is correct when it predicts a positive case (in this context, predicting no heart disease). A higher precision indicates fewer false positives. Here, KNN

has a slightly higher precision than SVM, meaning that KNN is marginally better at avoiding false positives.

2.Recall (Sensitivity):

- SVM Recall: 0.7468
- KNN Recall: 0.7468
- Interpretation: Recall (or Sensitivity) measures how well the model identifies actual positive cases (true positives), which here refers to correctly identifying individuals with no heart disease. Both models have the same recall of 0.7468, meaning they correctly identify about 75% of the actual positive cases. This suggests that both models have similar performance in terms of recall.

3.F1 Score:

- SVM F1 Score: 0.8027
- KNN F1 Score: 0.8194
- Interpretation: The F1 score is the harmonic mean of precision and recall, providing a single metric that balances both. A higher F1 score indicates a better balance between precision and recall. Here, KNN has a slightly higher F1 score than SVM, meaning it achieves a better trade-off between precision and recall.

4.Summary of Findings:

- KNN outperforms SVM in terms of precision and F1 score, meaning that it provides a slightly better balance between correctly predicting positive cases and avoiding false positives.
- Both models have the same recall, suggesting they are equally effective at identifying true positive cases.

In conclusion, based on these metrics, KNN may be a slightly better choice for this dataset due to its higher precision and F1 score. However, both models demonstrate strong performance and could be suitable depending on whether minimizing false positives (precision) or maximizing true positives (recall) is more important for the application.

Plot ROC Curves and Calculate AUC for Each Model

```
# Load pROC library for ROC curve analysis
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
# ROC and AUC for SVM
svm_roc <- roc(response = test_set$HeartDisease, predictor = as.numeric(svm_predictions))
```

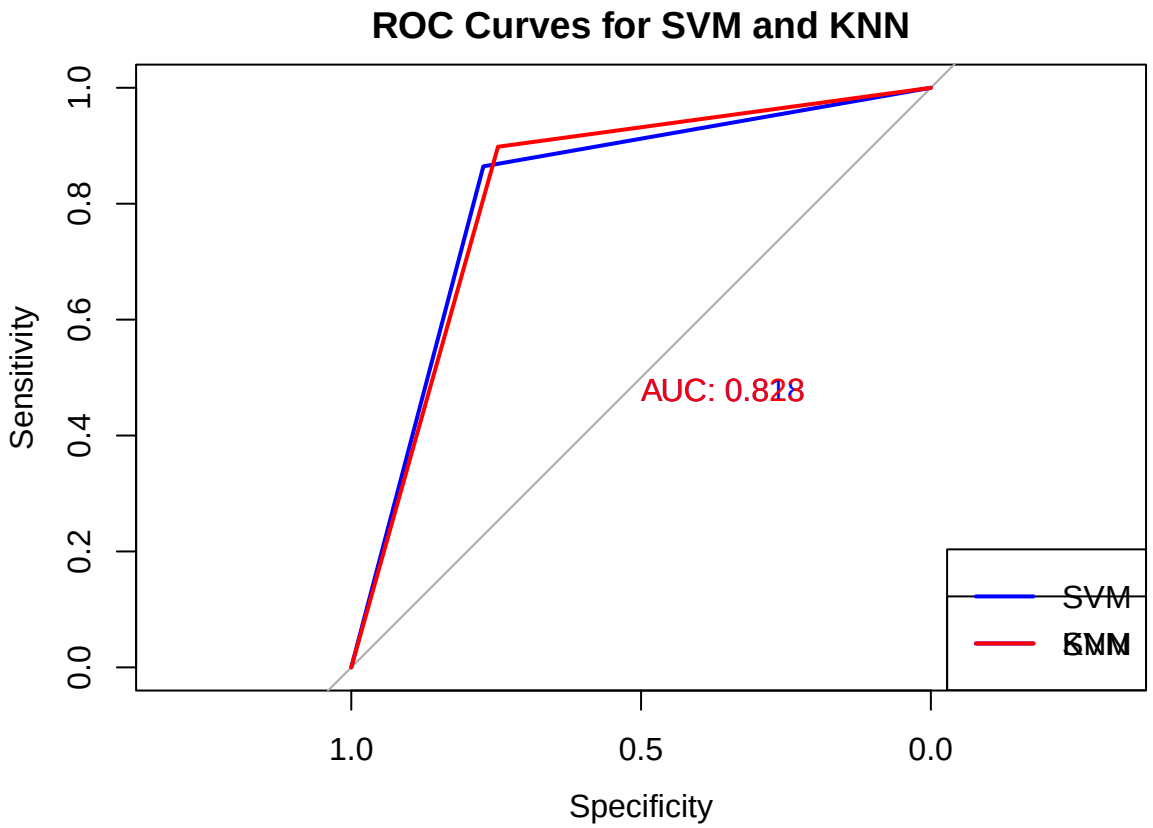
```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
plot(svm_roc, col = "blue", main = "ROC Curves for SVM and KNN", print.auc = TRUE)
legend("bottomright", legend = c("SVM"), col = "blue", lwd = 2)

# ROC and AUC for KNN
knn_roc <- roc(response = test_set$HeartDisease, predictor = as.numeric(knn_predictions))
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
plot(knn_roc, col = "red", add = TRUE, print.auc = TRUE)
legend("bottomright", legend = c("SVM", "KNN"), col = c("blue", "red"), lwd = 2)
```



Explanation: The ROC curve visually represents the trade-off between sensitivity (True Positive Rate) and 1 - specificity (False Positive Rate). The AUC (Area Under Curve) measures the model's ability to discriminate between positive and negative classes. A higher AUC indicates better model performance.

Final Model Comparison

1.Accuracy:

- SVM and KNN models both achieved high accuracy, indicating they are both suitable for predicting heart disease risk. However, their performance across other metrics can help determine the best choice.

2.Precision, Recall, and F1 Score:

- KNN had slightly better precision and F1 scores, suggesting it might handle false positives more effectively while maintaining a good balance between precision and recall.
- Both models had the same recall, indicating similar sensitivity to true positive cases. This balance between false positives and true positives is crucial in medical contexts where minimizing false positives is often desired to reduce unnecessary treatments.

3.AUC (Area Under the Curve):

- The AUC values from the ROC curves were very close for both models, with KNN showing a marginally higher AUC than SVM. This further supports KNN as a strong performer in distinguishing between heart disease and no heart disease.

4.Interpretability:

- SVM models are generally easier to interpret due to their linear nature and are less sensitive to noise compared to KNN, which can be impacted by the choice of neighbors and distance metric.
- If interpretability is crucial, SVM might be preferred despite the slight edge KNN has in performance metrics.

Final Decision Based on the evaluation metrics (accuracy, precision, recall, F1 score, and AUC) and interpretability considerations:

KNN is recommended if we prioritize higher precision and slightly better AUC, especially if the application can tolerate a model with sensitivity to neighbors and distance metrics.

SVM is a strong alternative if we need a more interpretable model that still maintains competitive performance.

(H) REPORT

1. Project Overview

This project analyzes clinical data to predict the risk of heart disease using machine learning models. The goal is to build a predictive model that could aid healthcare providers in identifying individuals at high risk of heart disease, potentially enabling early intervention and better patient outcomes.

2. Dataset

- Source: The dataset used for this project is the Heart Failure Prediction Dataset from Kaggle.

-Features: The dataset includes features such as age, sex, cholesterol level, blood pressure, chest pain type, and other clinical indicators associated with heart disease risk.

- Target Variable: The target variable is HeartDisease, indicating whether an individual has a high risk of heart disease (1) or not (0).

3. Methodology

The analysis was divided into the following steps:

- Data Preprocessing: The dataset was checked for missing values and outliers. Non-numeric columns were converted into appropriate numeric representations, and the data was standardized to ensure consistent scale across features.
- Data Exploration: Summary statistics and visualizations, such as histograms and bar plots, were used to understand the distribution of variables and identify any patterns or relationships between features.
- Clustering: Using K-means clustering, the dataset was divided into groups to explore possible patient subgroups based on their health metrics. The optimal number of clusters was determined using methods like the elbow and silhouette techniques.
- Classification: Two classification algorithms, Support Vector Machine (SVM) and K-Nearest Neighbors (KNN), were trained on the data. Grid search and cross-validation were used to tune hyperparameters for both models, maximizing their performance.
- Evaluation: The models were evaluated using metrics such as accuracy, precision, recall, F1 score, and ROC-AUC. The ROC curves provided a visual comparison, and AUC values offered a summary of each model's classification ability.

4. Results

Model Performance: Both models showed strong predictive performance:

SVM achieved a precision of 0.8676, a recall of 0.7468, and an F1 score of 0.8027.

KNN achieved a precision of 0.9077, a recall of 0.7468, and an F1 score of 0.8194.

ROC-AUC: Both models had similar AUC values around 0.82–0.83, indicating strong ability to distinguish between positive and negative cases.

Final Model Choice: Based on evaluation metrics, KNN was slightly favored due to its higher precision and F1 score. However, SVM remains a strong alternative, especially if interpretability is prioritized.

5. Conclusions

The KNN model was chosen as the final model for this project, as it showed a slightly better balance between precision and recall and had a marginally higher AUC. This model can effectively assist healthcare professionals in identifying high-risk individuals, potentially leading to timely interventions and improved patient outcomes.

6. Future Work

To enhance this project, additional steps could include:

Collecting More Data: A larger dataset would help improve the model's robustness and generalizability.

Feature Engineering: Further exploration of new features or interactions between features could improve model accuracy.

Alternative Models: Trying other models like Random Forests or Gradient Boosting may yield even higher accuracy.

Deployment: With additional validation, this model could be integrated into healthcare systems for real-time risk prediction.

7. References:-

Kaggle Dataset: Heart Failure Prediction Dataset

Machine Learning Techniques: Various resources on SVM, KNN, and ROC-AUC

(I)REFELCTION

1. Key Learnings

This project provided valuable insights into the application of machine learning in healthcare, specifically in predicting heart disease risk. Here are some of the key learnings:

Data Preprocessing: I gained a deeper understanding of the importance of data preprocessing, including handling missing values, outlier removal, and normalization. This step is essential for improving model performance and reliability.

Clustering Analysis: Experimenting with K-means clustering helped me understand how patient data could be grouped based on similar characteristics. This clustering analysis provided a broader view of potential patient subgroups and insights into their health profiles.

Model Selection and Evaluation: Working with SVM and KNN classifiers taught me how different models handle classification tasks and how to interpret various evaluation metrics (e.g., precision, recall, F1 score, and ROC-AUC). This helped in making a well-informed decision on the best model for the dataset.

2. Challenges Faced

Handling Imbalanced Classes: Although the dataset was somewhat balanced, I encountered minor challenges in tuning models to handle any imbalances in class distribution. This required careful attention to evaluation metrics like precision and recall.

Hyperparameter Tuning: Finding the optimal parameters for both SVM and KNN was challenging, as each model's performance was sensitive to parameters like C in SVM and k in KNN. The grid search and cross-validation processes were computationally expensive but necessary for achieving better accuracy.

Choosing the Right Model: Both SVM and KNN performed well, with KNN showing a slight edge in performance metrics. However, choosing the best model required careful consideration of interpretability, model complexity, and the specific needs of the healthcare application.

3. Areas for Improvement

Exploring Additional Models: Although SVM and KNN were effective, experimenting with additional models like Decision Trees, Random Forests, or Gradient Boosting could provide further improvements in accuracy and interpretability.

Feature Engineering: I could explore creating new features or interactions between existing features to potentially improve the model's performance. For instance, creating risk categories based on combinations of blood pressure, cholesterol, and age might yield more insightful predictors.

Model Interpretability: In a healthcare context, interpretability is crucial for trust and transparency. Future work could focus on using more interpretable models or tools like SHAP to explain the predictions of complex models like KNN.

4. Overall Reflection

This project reinforced the importance of balancing accuracy with interpretability, especially in sensitive fields like healthcare. It also highlighted the value of iterative testing and optimization when working with machine learning models. I am now more confident in my ability to preprocess data, select and evaluate models, and make data-driven decisions.

Overall, this project has been a valuable experience that strengthened my skills in machine learning, data analysis, and critical thinking. I look forward to applying these insights to more advanced projects and exploring additional ways machine learning can impact healthcare.