Can We Reliably Predict Heart Disease in Individuals: A Regression Analysis?

Nikhil Kesani and Bharath Chandra Pulijala

```r
df <- read.csv("D:\\Heart Disease\\6.1 heart-disease.csv")
```

```r
tail(df)
```

```
##     age sex cp trestbps chol fbs restecg thalach exang oldpeak slope ca thal
## 298  59   1  0      164  176   1       0      90     0     1.0     1  2    1
## 299  57   0  0      140  241   0       1     123     1     0.2     1  0    3
## 300  45   1  3      110  264   0       1     132     0     1.2     1  0    3
## 301  68   1  0      144  193   1       1     141     0     3.4     1  2    3
## 302  57   1  0      130  131   0       1     115     1     1.2     1  1    3
## 303  57   0  1      130  236   0       0     174     0     0.0     1  1    2
##     target
## 298      0
## 299      0
## 300      0
## 301      0
## 302      0
## 303      0
```

```r
head(df)
```

```
##   age sex cp trestbps chol fbs restecg thalach exang oldpeak slope ca thal
## 1  63   1  3      145  233   1       0     150     0     2.3     0  0    1
## 2  37   1  2      130  250   0       1     187     0     3.5     0  0    2
## 3  41   0  1      130  204   0       0     172     0     1.4     2  0    2
## 4  56   1  1      120  236   0       1     178     0     0.8     2  0    2
## 5  57   0  0      120  354   0       1     163     1     0.6     2  0    2
## 6  57   1  0      140  192   0       1     148     0     0.4     1  0    1
##   target
## 1      1
## 2      1
## 3      1
## 4      1
## 5      1
## 6      1
```

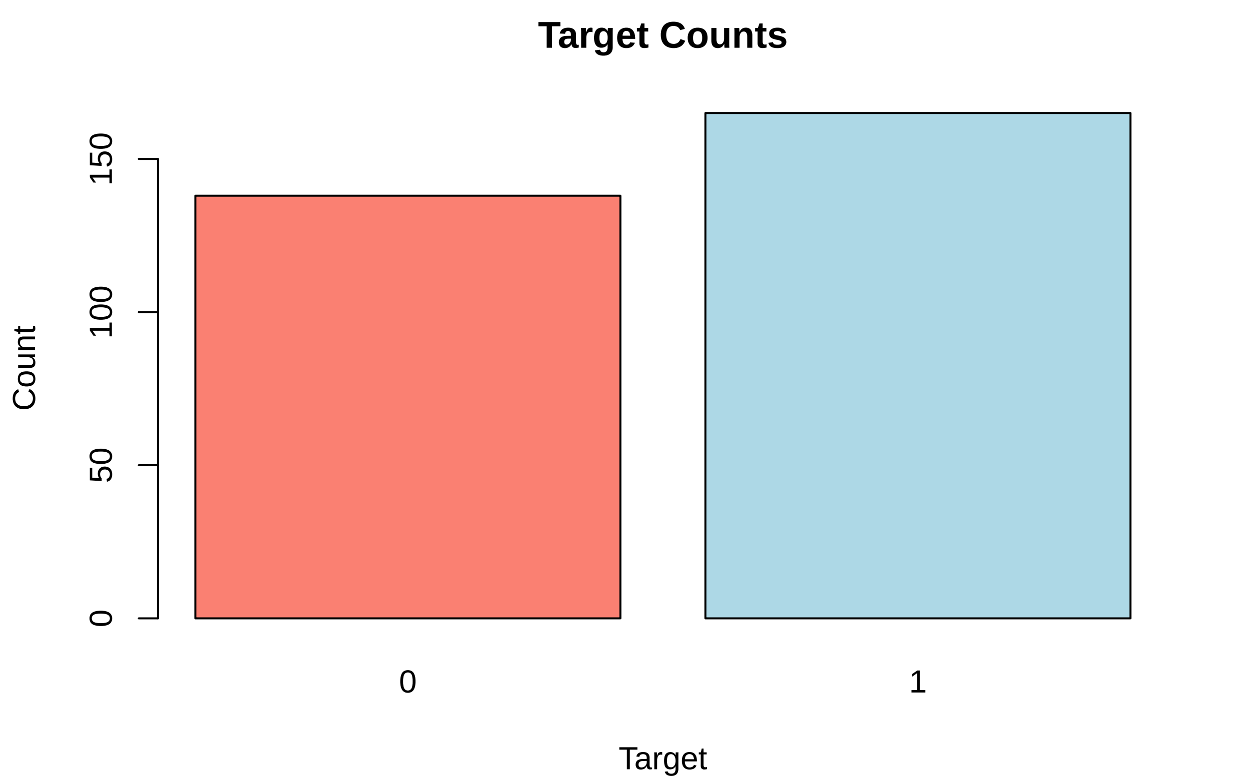
Description of Attributes

• Age—age of patient in years, sex—(1 = male; 0 = female). • Cp—chest pain type. • Trestbps—resting blood pressure (in mm Hg on admission to the hospital). The normal range is 120/80 (if you have a normal blood pressure reading, it is fine, but if it is a little higher than it should be, you should try to lower it. Make healthy changes to your lifestyle). • Chol—serum cholesterol shows the amount of triglycerides present. Triglycerides are another lipid that can be measured in the blood. It should be less than 170 mg/dL (may differ in different Labs). • Fbs—fasting blood sugar larger than 120 mg/dl (1 true). Less than 100 mg/dL (5.6 mmol/L) is normal, and 100 to 125 mg/dL (5.6 to 6.9 mmol/L) is considered prediabetes. • Restecg—resting electrocardiographic results. • Thalach—maximum heart rate achieved. The maximum heart rate is 220 minus your age. • Exang—exercise-induced angina (1 yes). Angina is a type of chest pain caused by reduced blood flow to the heart. Angina is a symptom of coronary artery disease. • Oldpeak—ST depression induced by exercise relative to rest. • Slope—the slope of the peak exercise ST segment. • Ca—number of major vessels (0–3) colored by fluoroscopy. • Thal—no explanation provided, but probably thalassemia (3 normal; 6 fixed defects; 7 reversible defects). • Target (T)—no disease = 0 and disease = 1

```r
table(df$target)
```

```
##
##   0   1
## 138 165
```

```r
target_counts <- table(df$target)

# Define colors for the bars
colors <- c("salmon", "lightblue")

# Plot the bar plot
barplot(target_counts, col = colors, main = "Target Counts", xlab = "Target", ylab = "Count")
```




```r
str(df)
```

```
## 'data.frame':    303 obs. of  14 variables:
##  $ age     : int  63 37 41 56 57 57 56 44 52 57 ...
##  $ sex     : int  1 1 0 1 0 1 0 1 1 1 ...
```

```
##  $ cp      : int  3 2 1 1 0 0 1 1 2 2 ...
##  $ trestbps: int  145 130 130 120 120 140 140 120 172 150 ...
##  $ chol    : int  233 250 204 236 354 192 294 263 199 168 ...
##  $ fbs     : int  1 0 0 0 0 0 0 0 1 0 ...
##  $ restecg : int  0 1 0 1 1 1 0 1 1 1 ...
##  $ thalach : int  150 187 172 178 163 148 153 173 162 174 ...
##  $ exang   : int  0 0 0 0 1 0 0 0 0 0 ...
##  $ oldpeak : num  2.3 3.5 1.4 0.8 0.6 0.4 1.3 0 0.5 1.6 ...
##  $ slope   : int  0 0 2 2 2 1 1 2 2 2 ...
##  $ ca      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ thal    : int  1 2 2 2 2 1 2 3 3 2 ...
##  $ target  : int  1 1 1 1 1 1 1 1 1 1 ...
```

```r
# Assuming df is your dataframe
na_counts <- colSums(is.na(df))
print(na_counts)
```

```
##      age      sex       cp trestbps     chol      fbs  restecg  thalach
##        0        0        0        0        0        0        0        0
##    exang  oldpeak    slope       ca     thal   target
##        0        0        0        0        0        0
```

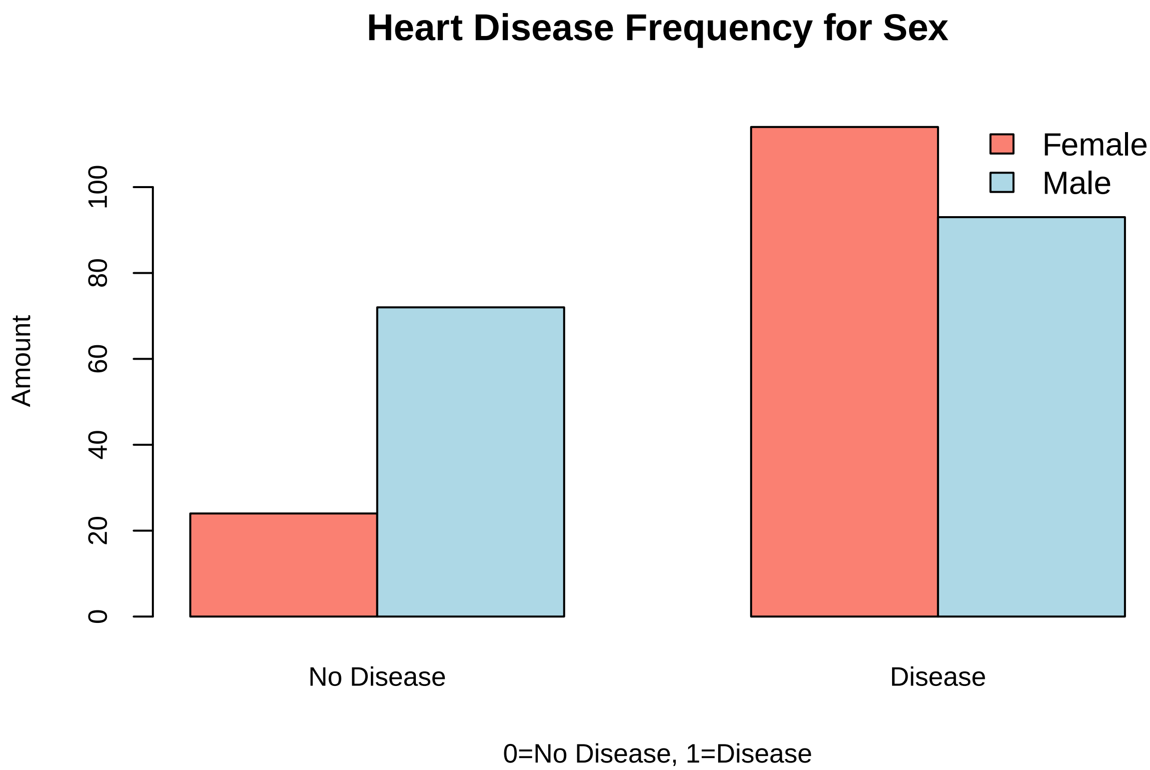
```r
table(df$sex)
```

```
##
##   0   1
##  96 207
```

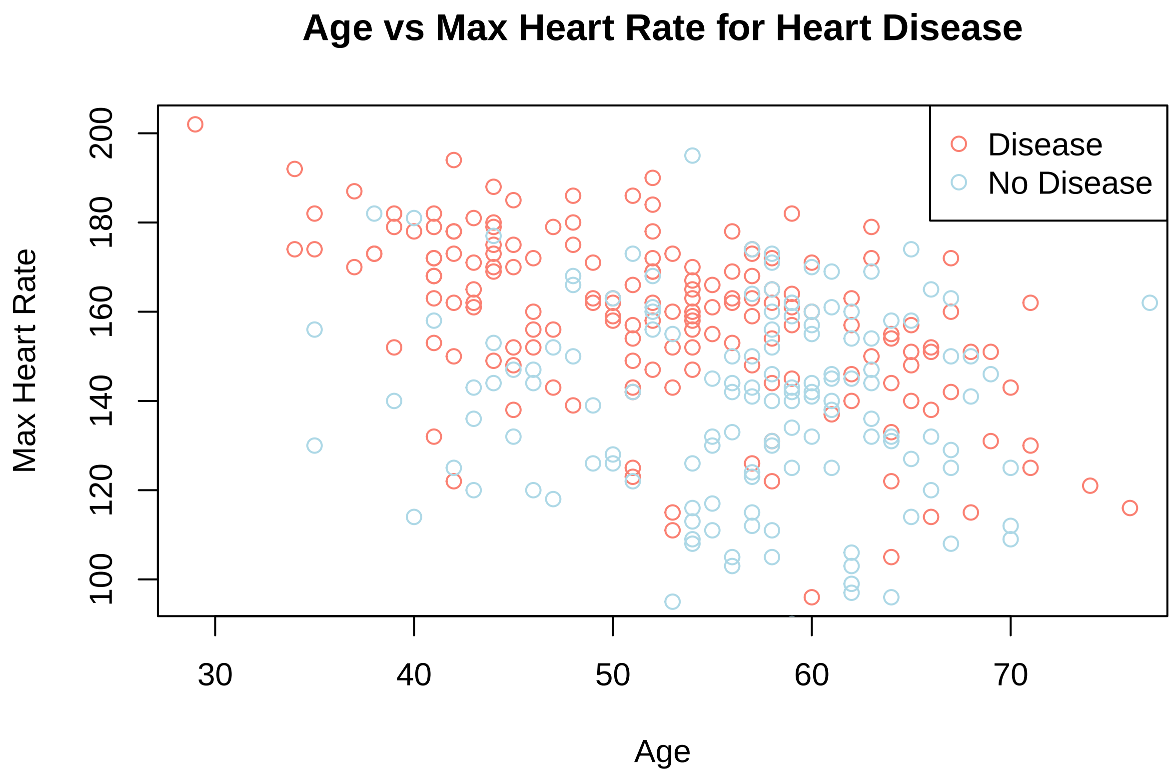
```r
table(df$target, df$sex)
```

```
##
##        0   1
##   0   24 114
##   1   72  93
```

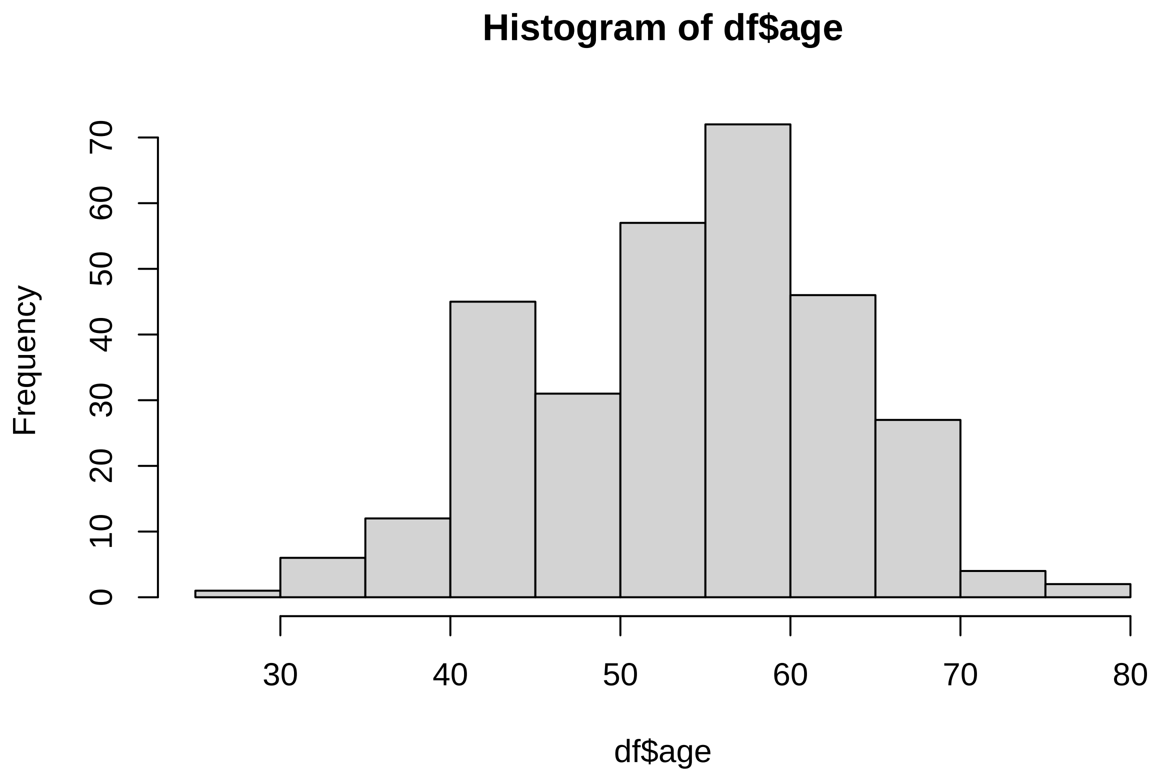
```r
# Create contingency table
cross_tab <- table(df$target, df$sex)

# Plot the contingency table
barplot(cross_tab, beside = TRUE, col = c("salmon", "lightblue"),
          main = "Heart Disease Frequency for Sex",
        xlab = "0=No Disease, 1=Disease",
        ylab = "Amount",
        legend.text = c("Female", "Male"),
        args.legend = list(x = "topright", bty = "n"),
        names.arg = c("No Disease", "Disease"),
        ylim = c(0, max(cross_tab) + 5),
        cex.axis = 0.8,
        cex.names = 0.8,
        cex.lab = 0.8)
```



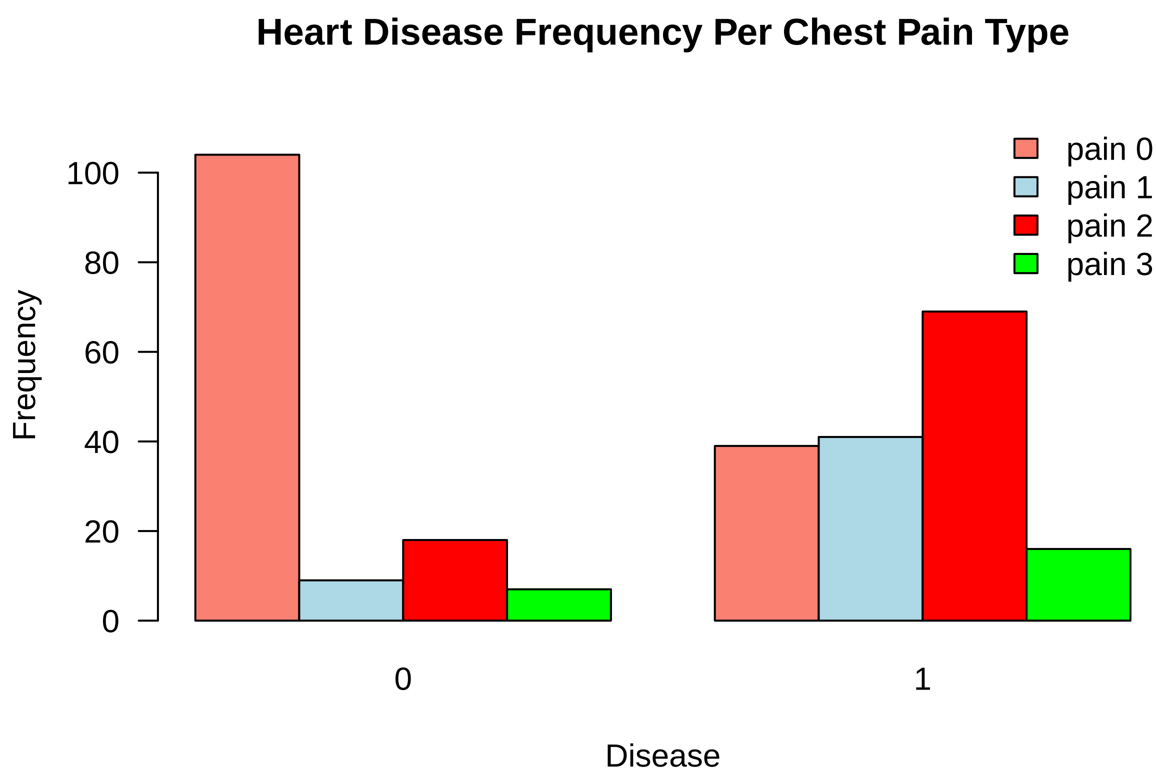

```r
# Set the size of the plot
options(repr.plot.width=10, repr.plot.height=6)

# Scatter plot with positive example
plot(df$age[df$target == 1], df$thalach[df$target == 1], col = 'salmon',
     xlab = 'Age', ylab = 'Max Heart Rate')

# Add points for negative example
points(df$age[df$target == 0], df$thalach[df$target == 0], col = 'lightblue')

# Add title and legend
title('Age vs Max Heart Rate for Heart Disease')
legend('topright', legend = c('Disease', 'No Disease'), col = c('salmon', 'lightblue'), pch = 1)
```


Age vs Max Heart Rate for Heart Disease
Disease
No Disease
Max Heart Rate
100
120
140
160
180
200
30
40
50
60
70
Age

```r
hist(df$age)
```




```
# Assuming df$cp is the column representing chest pain type and df$target is the target variable

# Create the contingency table
result <- table(df$cp, df$target)

# Define colors
colors <-  c("salmon", "lightblue","red","green")

# Plot the bar plot
barplot(result, beside = TRUE, col = colors, main = "Heart Disease Frequency Per Chest Pain Type",
        xlab = "Disease", ylab = "Frequency", legend.text = c("pain 0","pain 1","pain 2","pain 3"),
        args.legend = list(x = "topright", bty = "n"), ylim = c(0, max(result) + 10), las = 1)
```




```r
correlation_matrix <- cor(df)
correlation_matrix
```

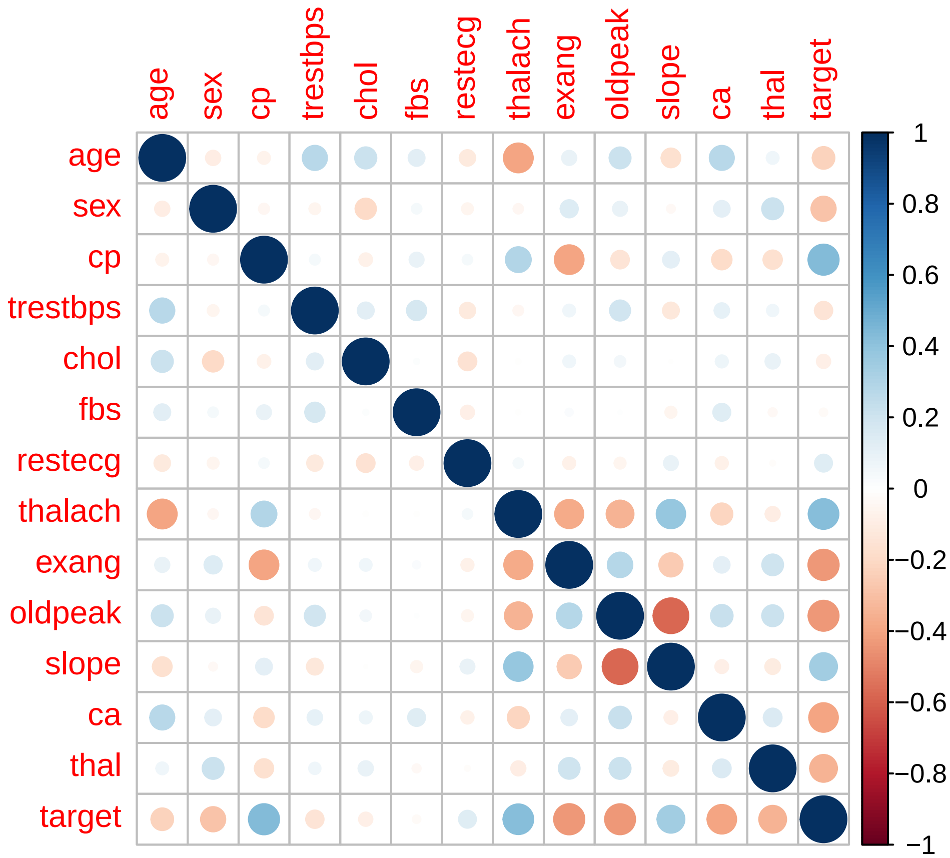
```
##                  age         sex          cp    trestbps         chol
## age       1.00000000 -0.09844660 -0.06865302  0.27935091  0.213677957
## sex      -0.09844660  1.00000000 -0.04935288 -0.05676882 -0.197912174
## cp       -0.06865302 -0.04935288  1.00000000  0.04760776 -0.076904391
## trestbps  0.27935091 -0.05676882  0.04760776  1.00000000  0.123174207
## chol      0.21367796 -0.19791217 -0.07690439  0.12317421  1.000000000
## fbs       0.12130765  0.04503179  0.09444403  0.17753054  0.013293602
## restecg  -0.11621090 -0.05819627  0.04442059 -0.11410279 -0.151040078
## thalach  -0.39852194 -0.04401991  0.29576212 -0.04669773 -0.009939839
## exang     0.09680083  0.14166381 -0.39428027  0.06761612  0.067022783
## oldpeak   0.21001257  0.09609288 -0.14923016  0.19321647  0.053951920
## slope    -0.16881424 -0.03071057  0.11971659 -0.12147458 -0.004037770
## ca        0.27632624  0.11826141 -0.18105303  0.10138899  0.070510925
## thal      0.06800138  0.21004110 -0.16173557  0.06220989  0.098802993
## target   -0.22543872 -0.28093658  0.43379826 -0.14493113 -0.085239105
##                   fbs     restecg      thalach       exang      oldpeak
## age       0.121307648 -0.11621090 -0.398521938  0.09680083  0.210012567
## sex       0.045031789 -0.05819627 -0.044019908  0.14166381  0.096092877
## cp        0.094444035  0.04442059  0.295762125 -0.39428027 -0.149230158
## trestbps  0.177530542 -0.11410279 -0.046697728  0.06761612  0.193216472
## chol      0.013293602 -0.15104008 -0.009939839  0.06702278  0.053951920
## fbs       1.000000000 -0.08418905 -0.008567107  0.02566515  0.005747223
## restecg  -0.084189054  1.00000000  0.044123444 -0.07073286 -0.058770226
```

```
## thalach  -0.008567107  0.04412344  1.000000000 -0.37881209 -0.344186948
## exang     0.025665147 -0.07073286 -0.378812094  1.00000000  0.288222808
## oldpeak   0.005747223 -0.05877023 -0.344186948  0.28822281  1.000000000
## slope    -0.059894178  0.09304482  0.386784410 -0.25774837 -0.577536817
## ca        0.137979327 -0.07204243 -0.213176928  0.11573938  0.222682322
## thal     -0.032019339 -0.01198140 -0.096439132  0.20675379  0.210244126
## target   -0.028045760  0.13722950  0.421740934 -0.43675708 -0.430696002
##                slope          ca        thal      target
## age      -0.16881424  0.27632624  0.06800138 -0.22543872
## sex      -0.03071057  0.11826141  0.21004110 -0.28093658
## cp        0.11971659 -0.18105303 -0.16173557  0.43379826
## trestbps -0.12147458  0.10138899  0.06220989 -0.14493113
## chol     -0.00403777  0.07051093  0.09880299 -0.08523911
## fbs      -0.05989418  0.13797933 -0.03201934 -0.02804576
## restecg   0.09304482 -0.07204243 -0.01198140  0.13722950
## thalach   0.38678441 -0.21317693 -0.09643913  0.42174093
## exang    -0.25774837  0.11573938  0.20675379 -0.43675708
## oldpeak  -0.57753682  0.22268232  0.21024413 -0.43069600
## slope     1.00000000 -0.08015521 -0.10476379  0.34587708
## ca       -0.08015521  1.00000000  0.15183213 -0.39172399
## thal     -0.10476379  0.15183213  1.00000000 -0.34402927
## target    0.34587708 -0.39172399 -0.34402927  1.00000000
```

```r
# Load the corrplot package
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```r
# Compute the correlation matrix
corr_matrix <- cor(df)


corrplot(corr_matrix, method = "circle")
```



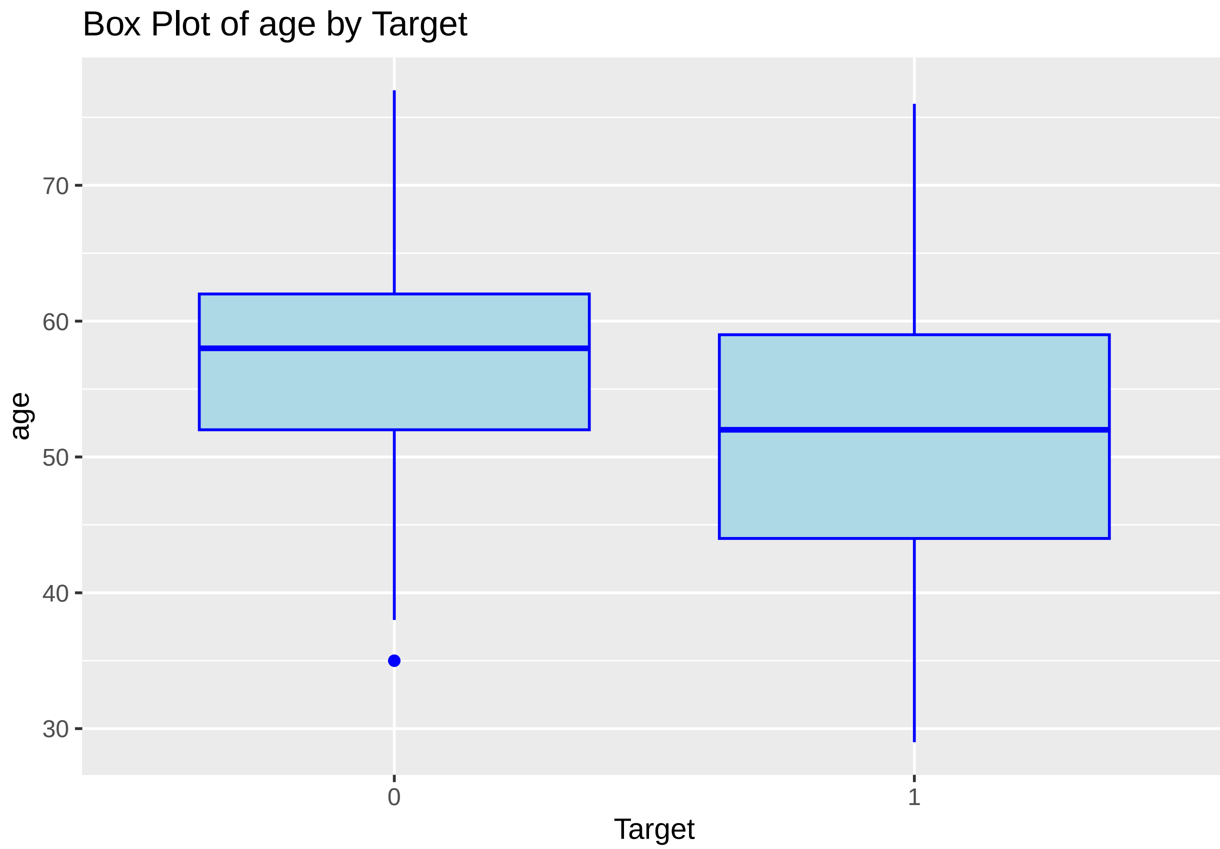

```r
# Load necessary libraries
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

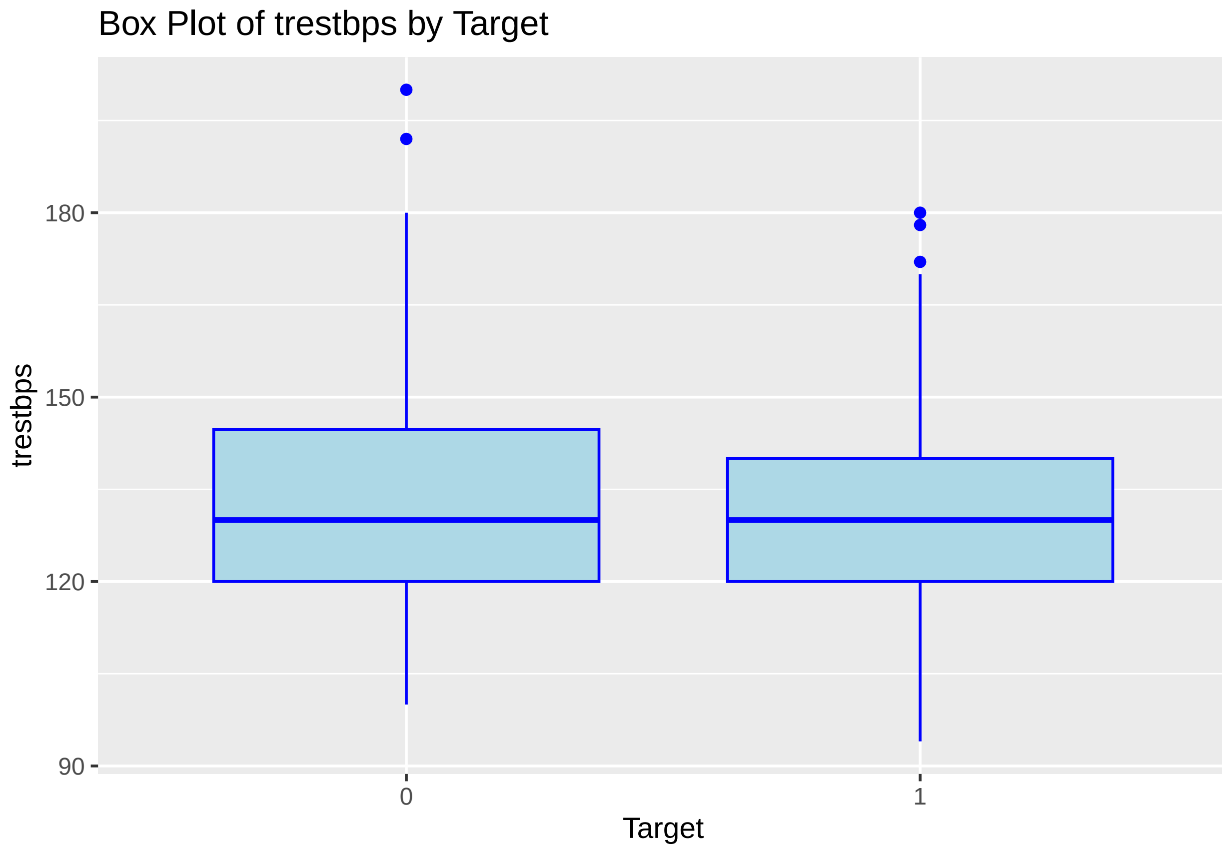
```r
# Box plots for numerical variables
numerical_vars <- c("age", "trestbps", "chol", "thalach", "oldpeak")

# Create box plots for each numerical variable
numerical_plots <- lapply(numerical_vars, function(var) {
  ggplot(data = df, aes(x = as.factor(target), y = !!sym(var))) +
    geom_boxplot(fill = "lightblue", color = "blue") +
    labs(title = paste("Box Plot of", var, "by Target"),
         x = "Target", y = var)
})


# Bar plots for categorical variables
categorical_vars <- c("sex", "cp", "fbs", "restecg", "exang", "slope", "ca", "thal")

# Create bar plots for each categorical variable
categorical_plots <- lapply(categorical_vars, function(var) {
  ggplot(data = df, aes(x = as.factor(target), y = ..count.., fill = as.factor(!!sym(var)))) +
    geom_bar(position = "dodge", color = "black") +
    labs(title = paste("Bar Plot of", var, "by Target"),
         x = "Target", y = "Count", fill = var)
})
```

```r
# Print box plots for numerical variables
print(numerical_plots)
```

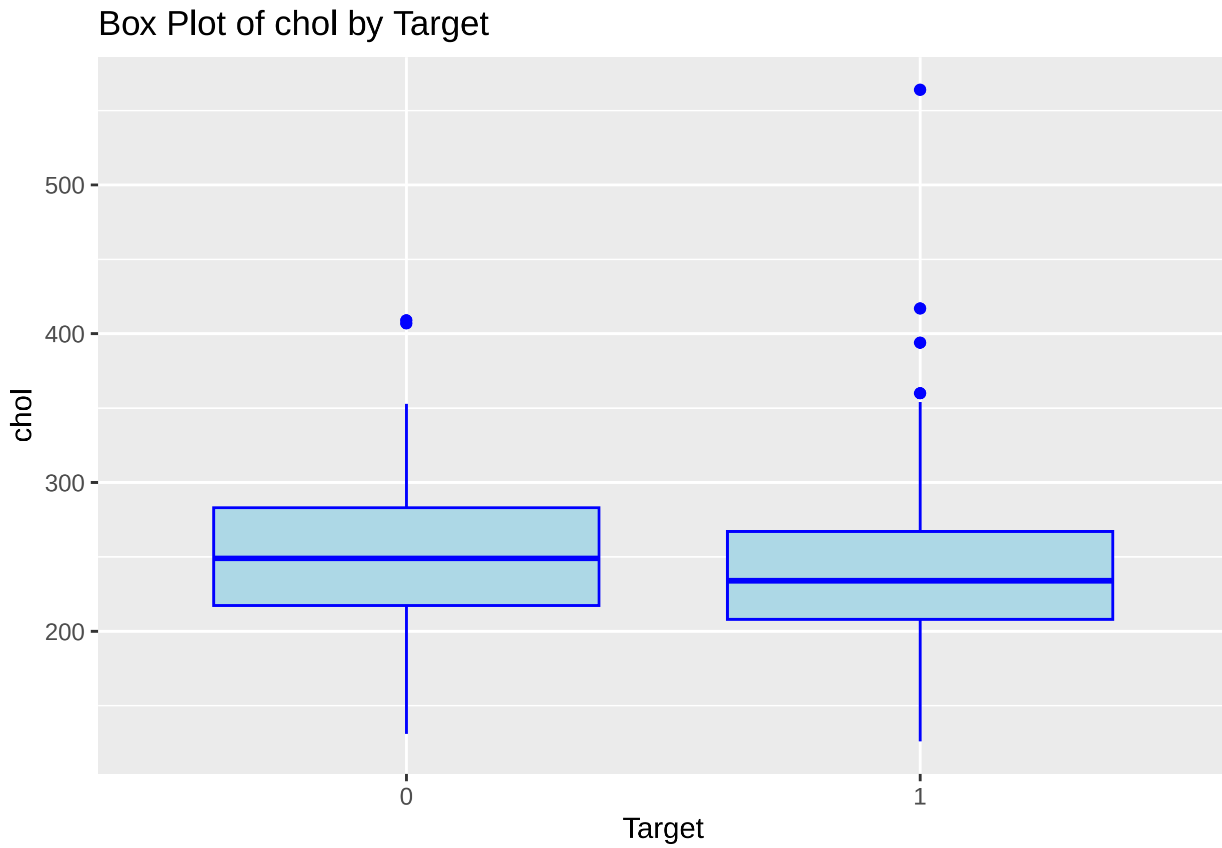
```
## [[1]]
```



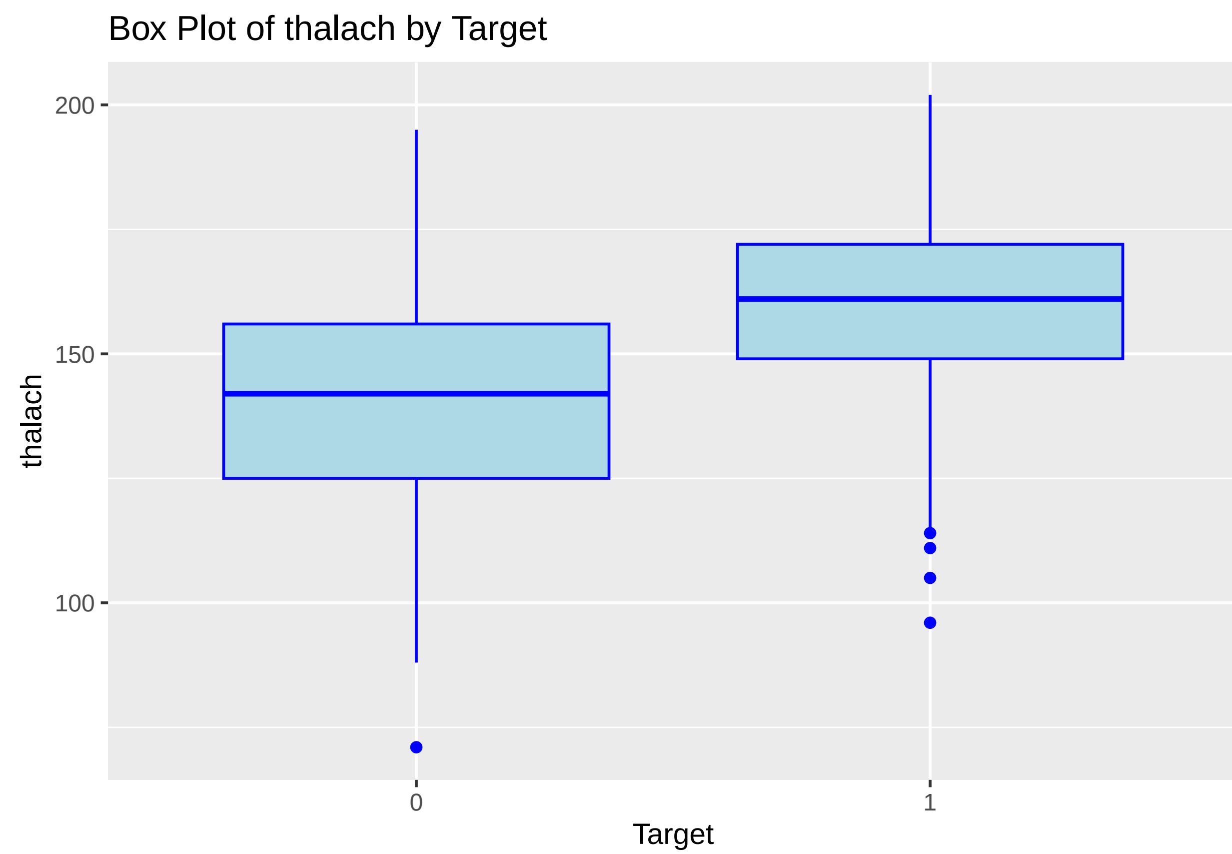

```
##
## [[2]]
```


Box Plot of trestbps by Target
180
150
120
90
trestbps
0
1
Target

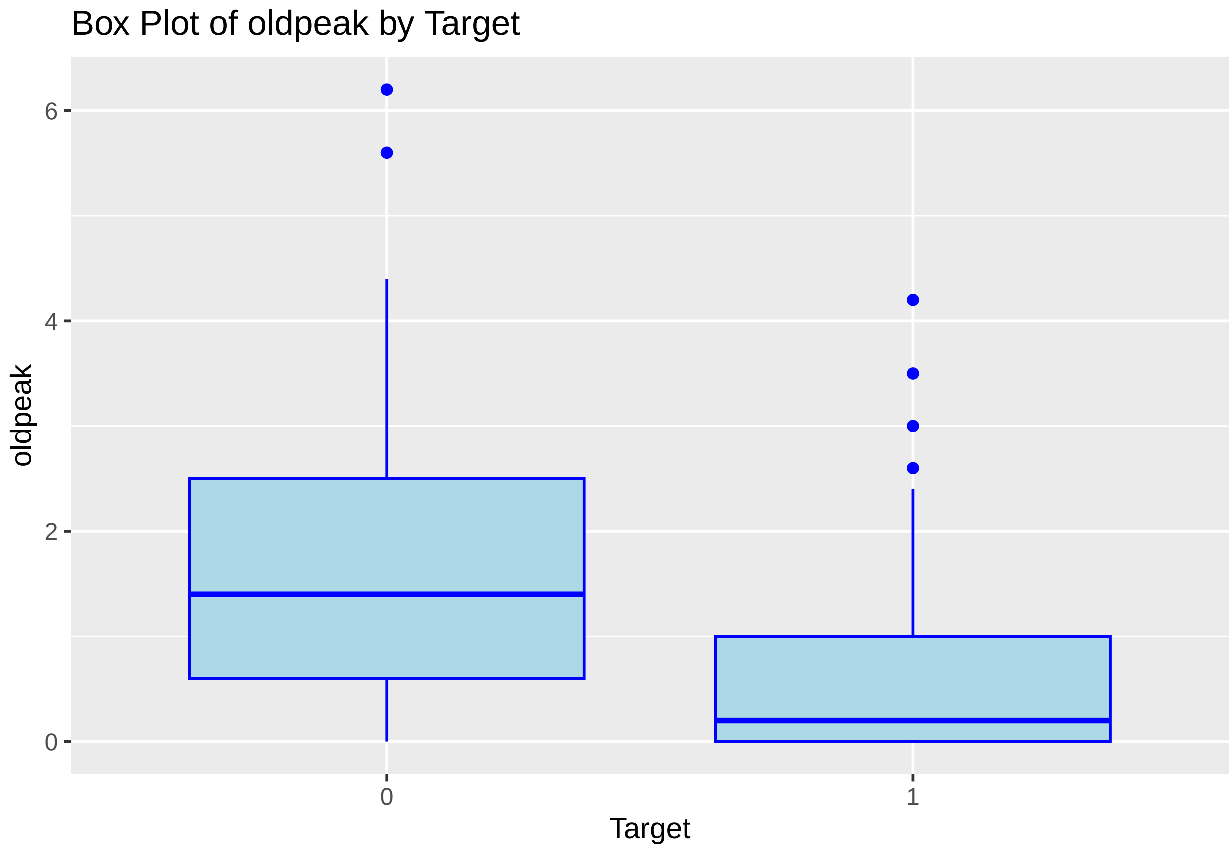
```
##
## [[3]]
```




```
##
## [[4]]
```



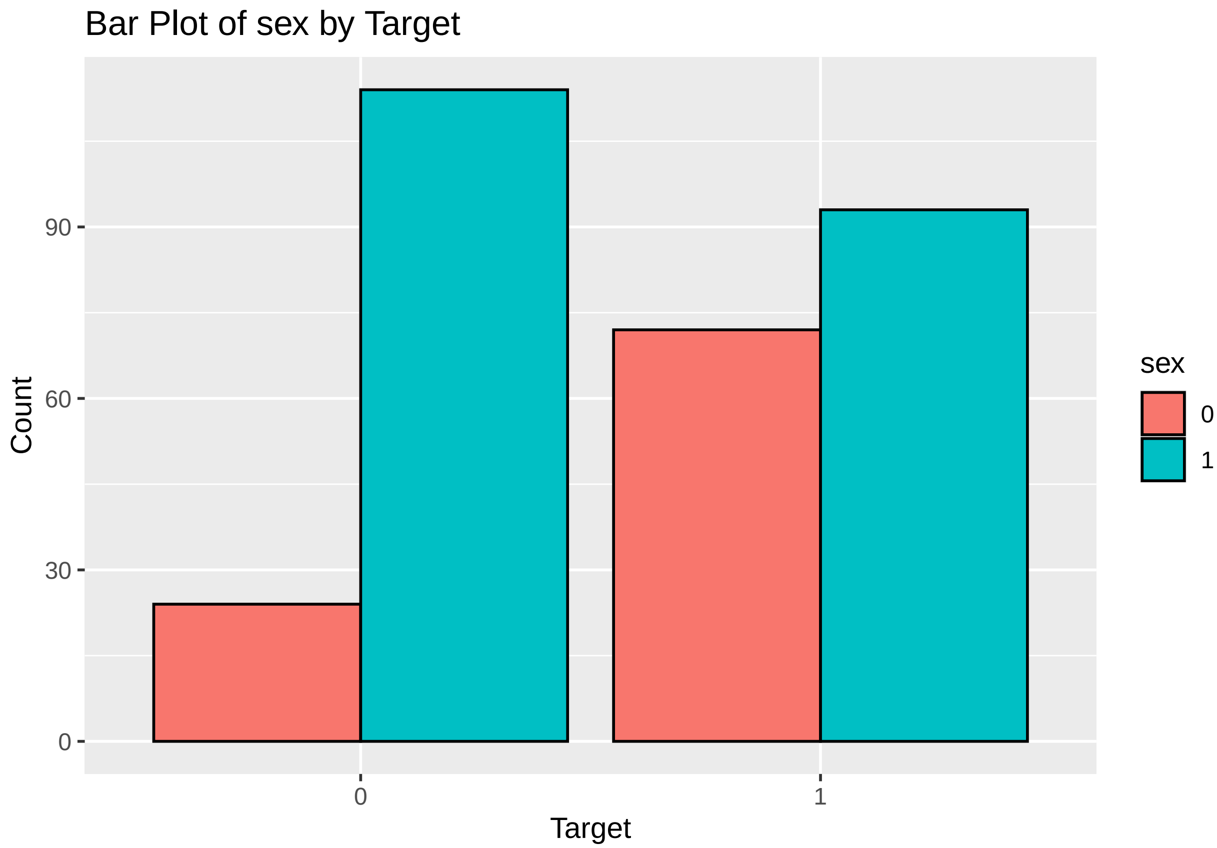

```
##
## [[5]]
```



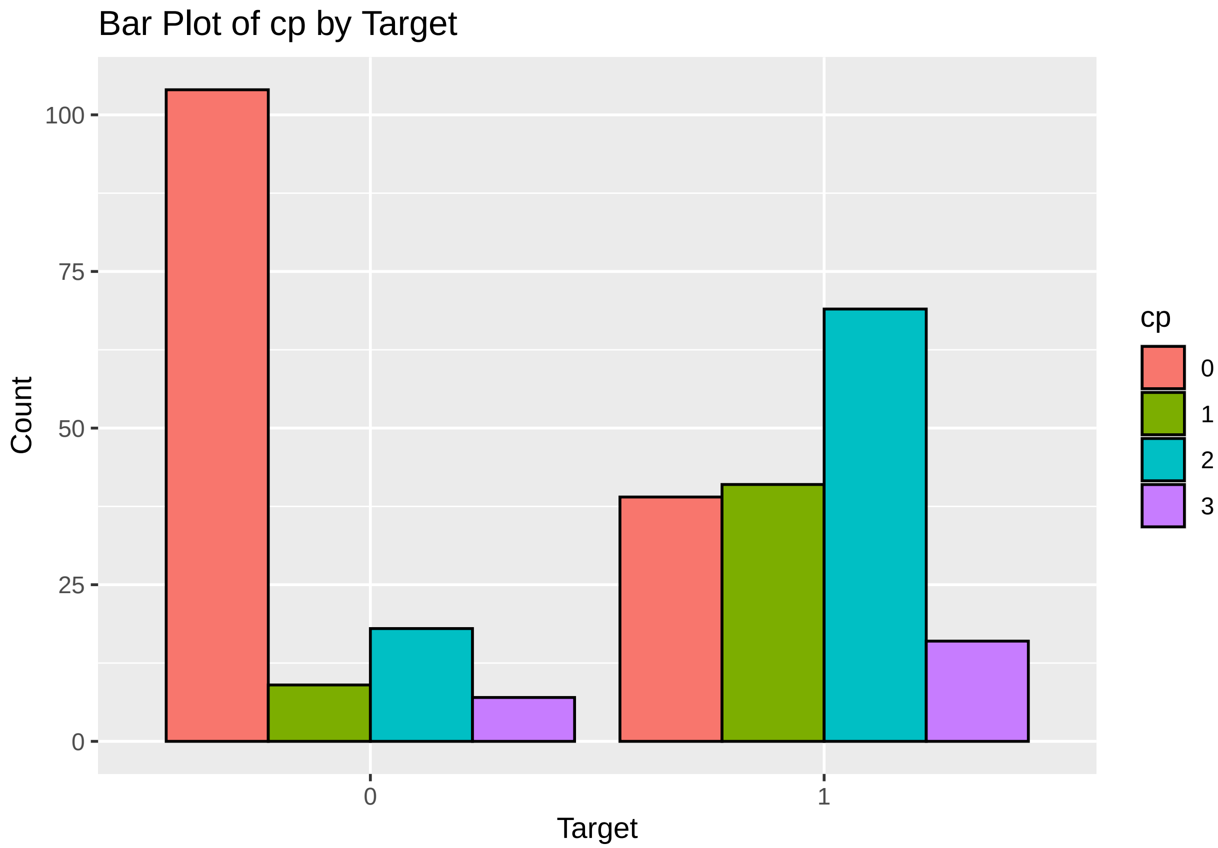

```r
# Print bar plots for categorical variables
print(categorical_plots)
```

```
## [[1]]
```

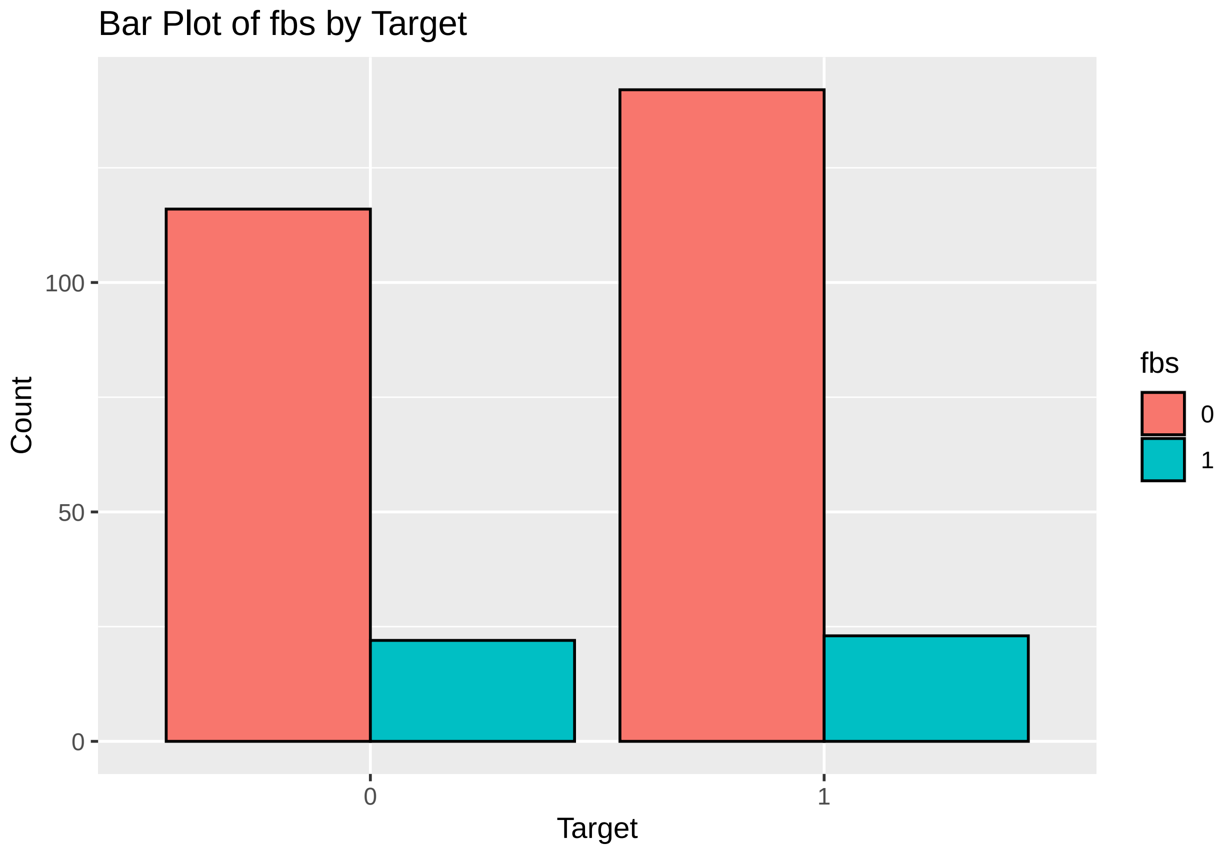
```
## Warning: The dot-dot notation (`..count..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(count)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



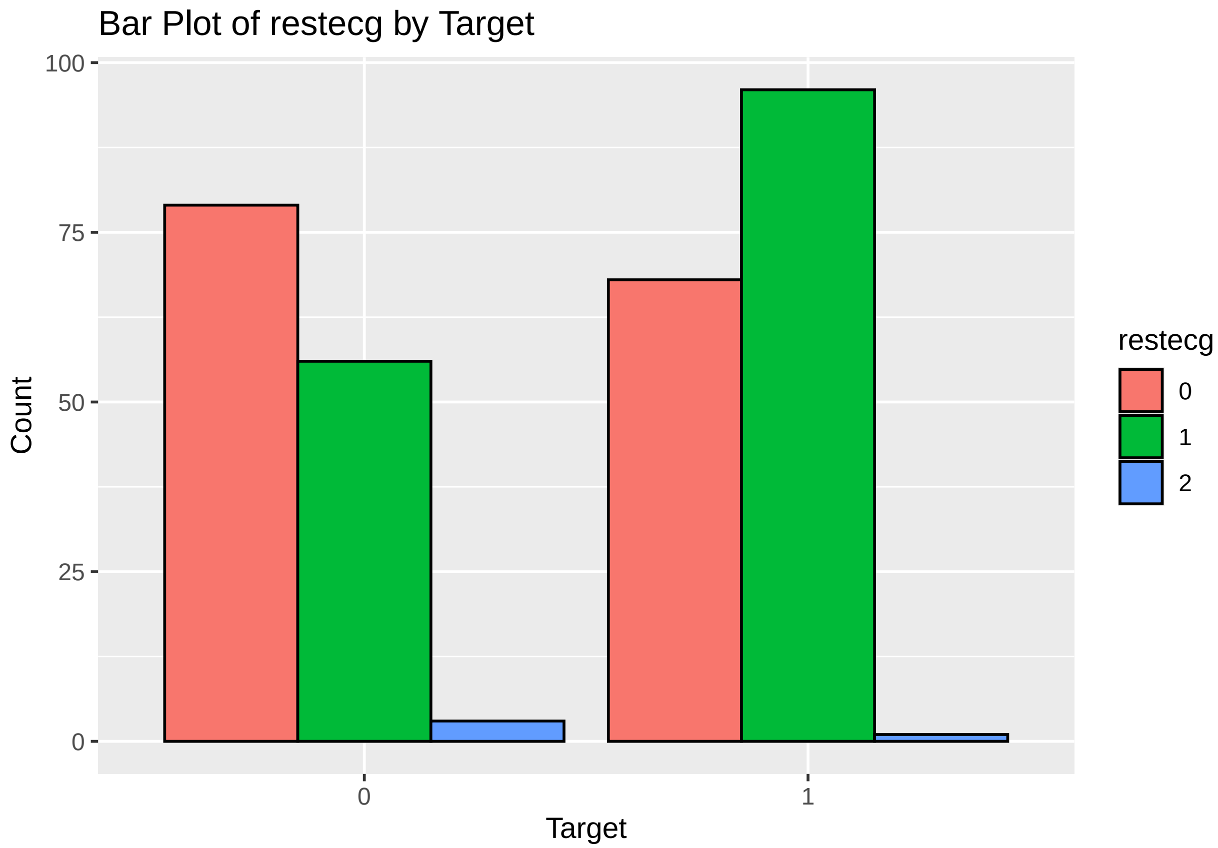

```
##
## [[2]]
```



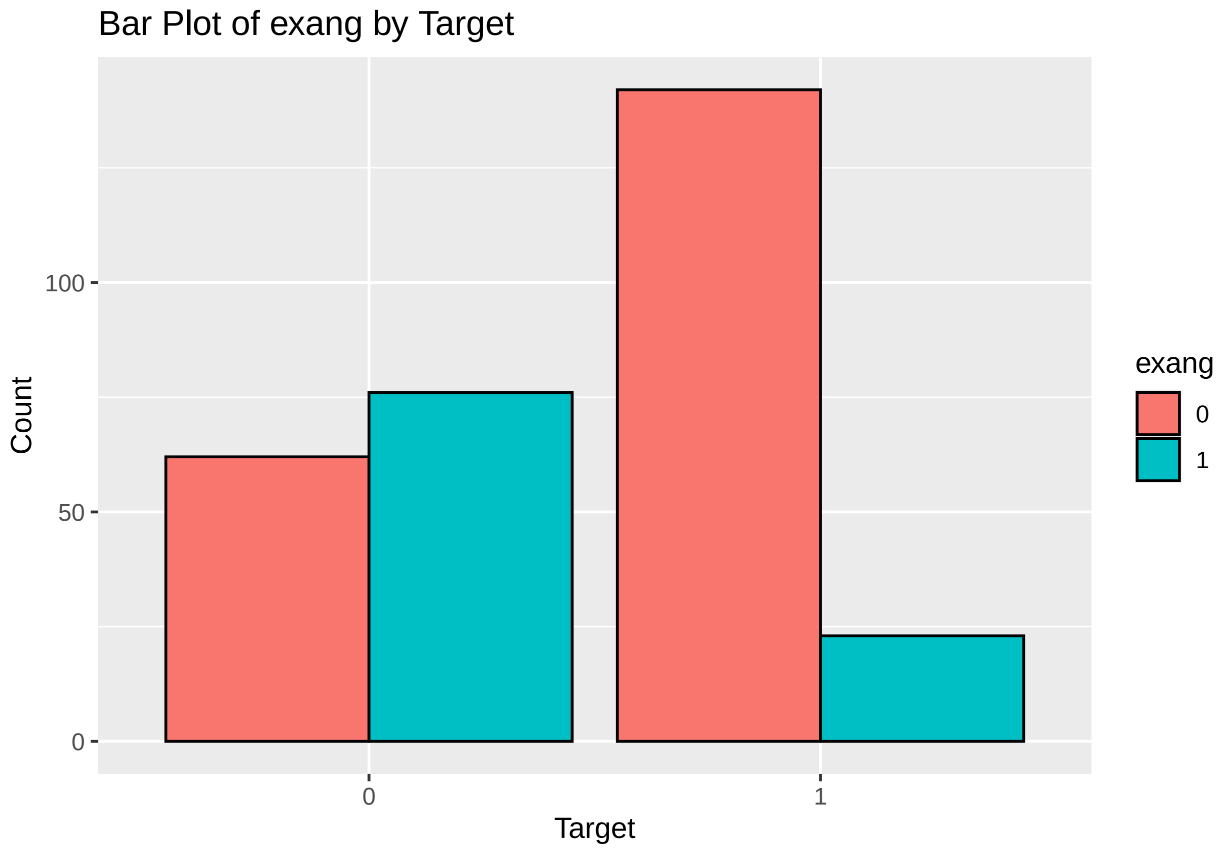

```
##
## [[3]]
```



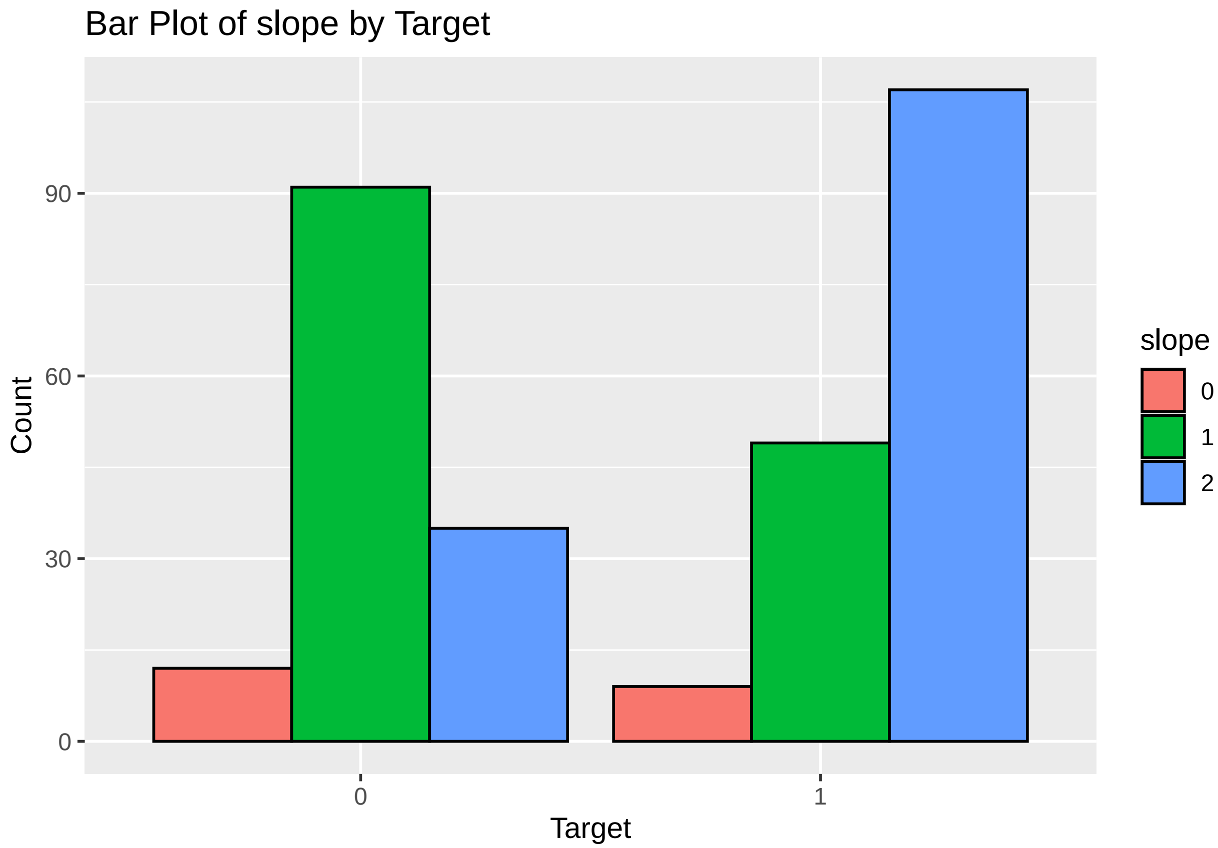

```
##
## [[4]]
```


Bar Plot of restecg by Target
100
75
50
25
0
Count
0
1
Target
restecg
0
1
2

```
##
## [[5]]
```

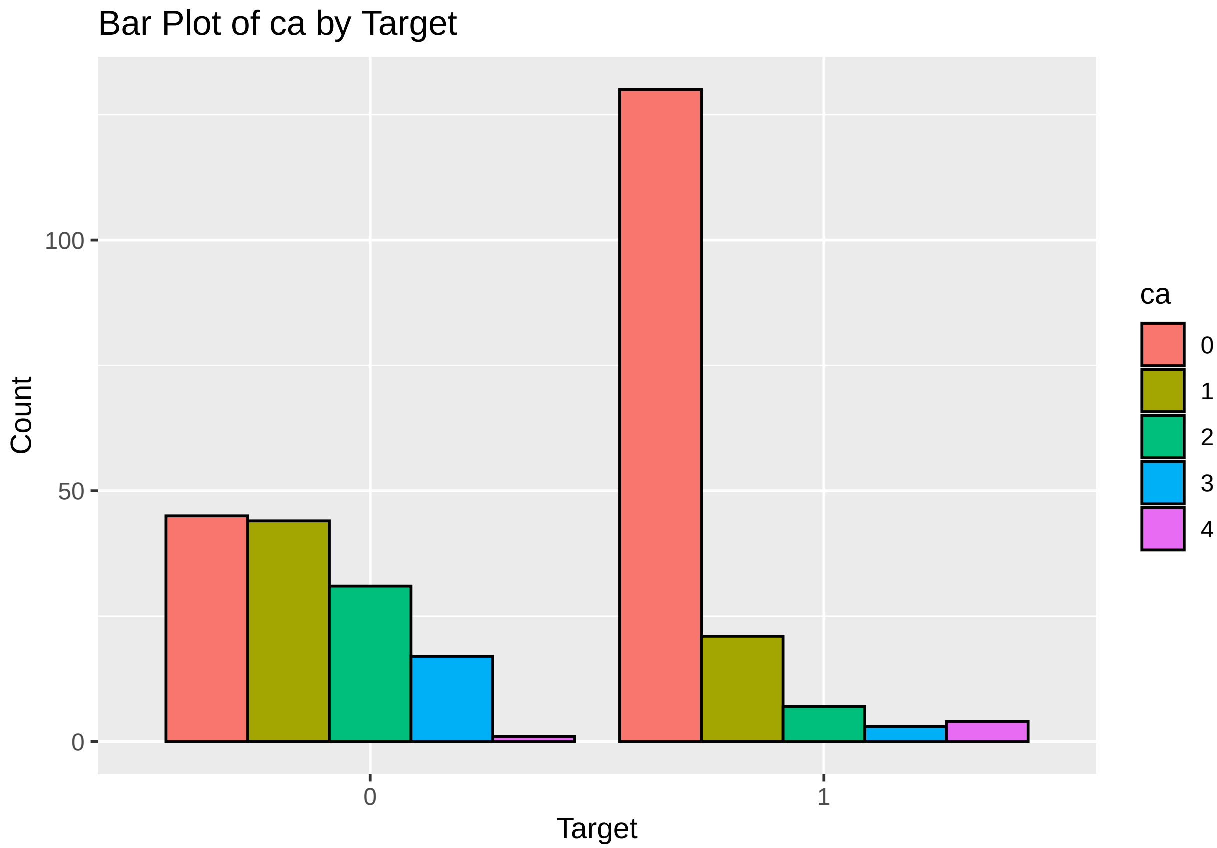



```
##
## [[6]]
```

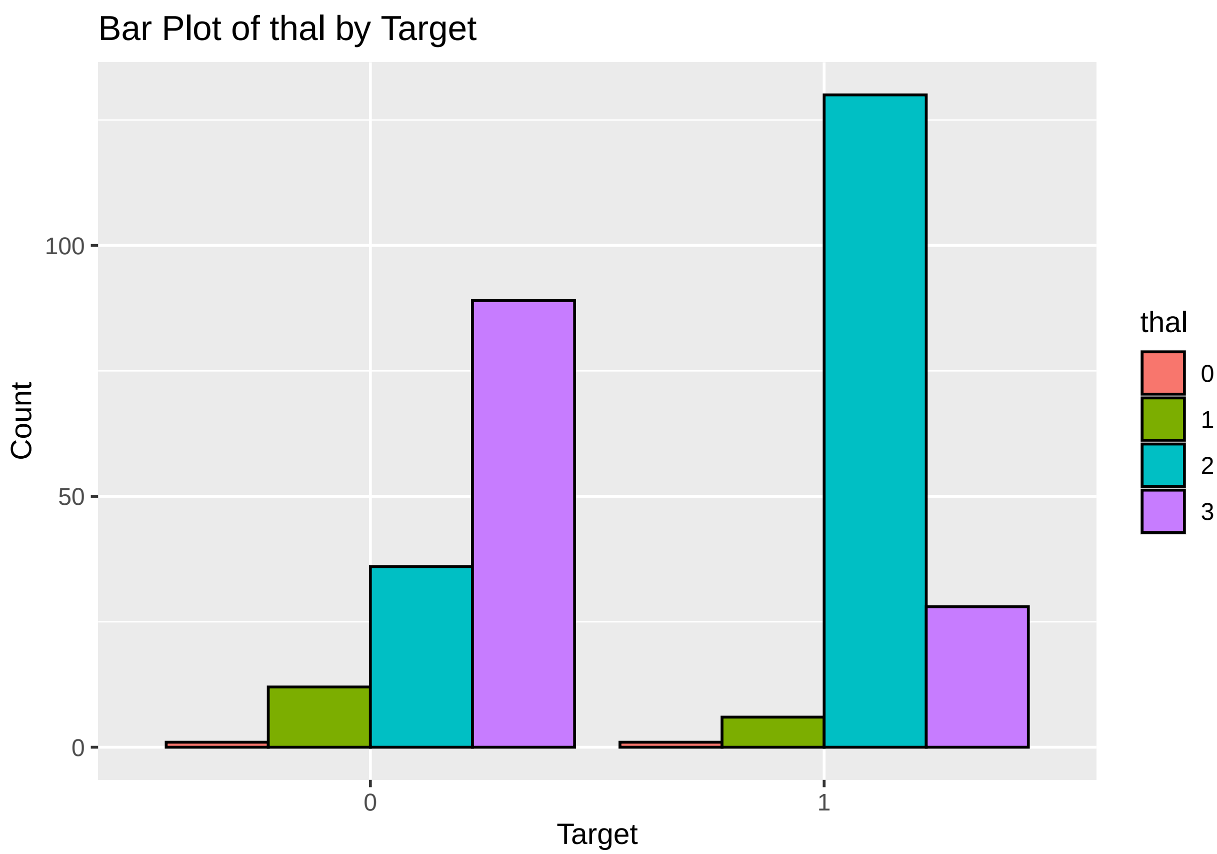


Bar Plot of slope by Target
Count
90
60
30
0
0
1
Target
slope
0
1
2


```
##
## [[7]]
```




```
##
## [[8]]
```

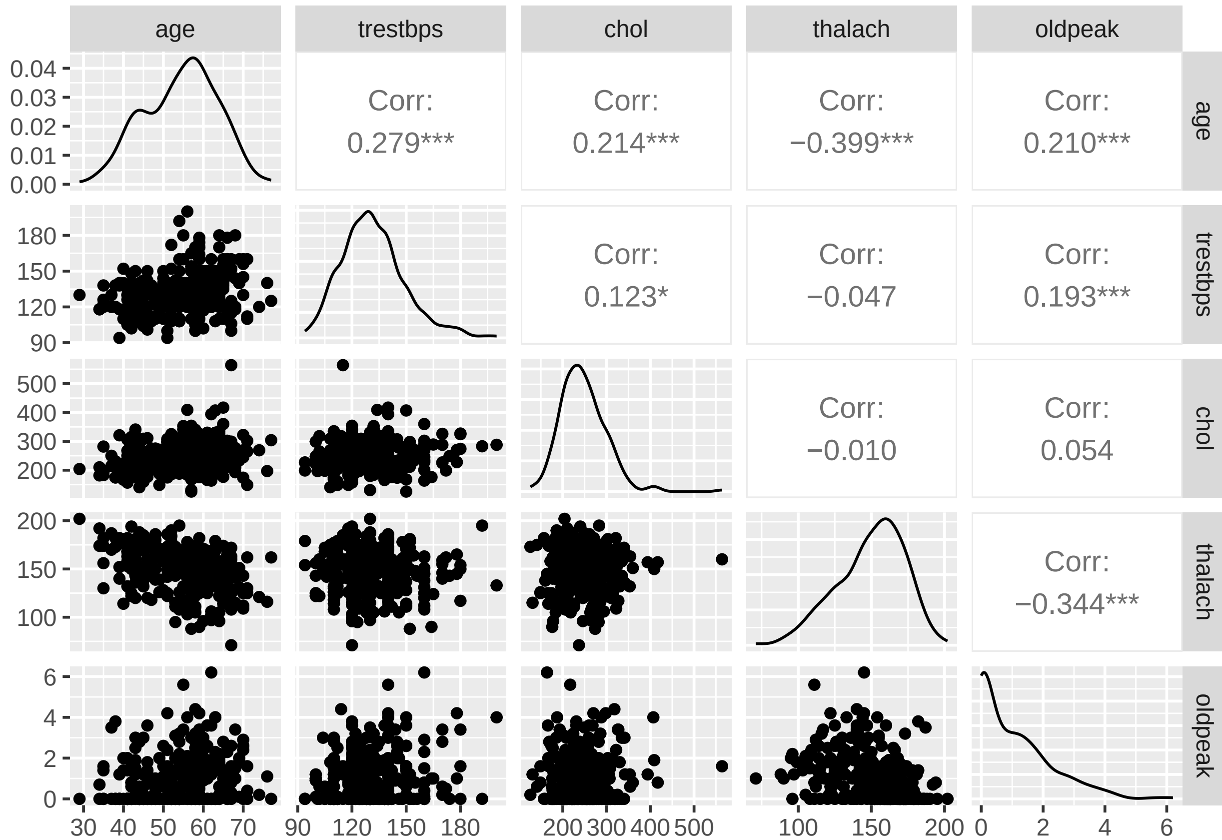



```r
library(ggplot2)
library(GGally)
```

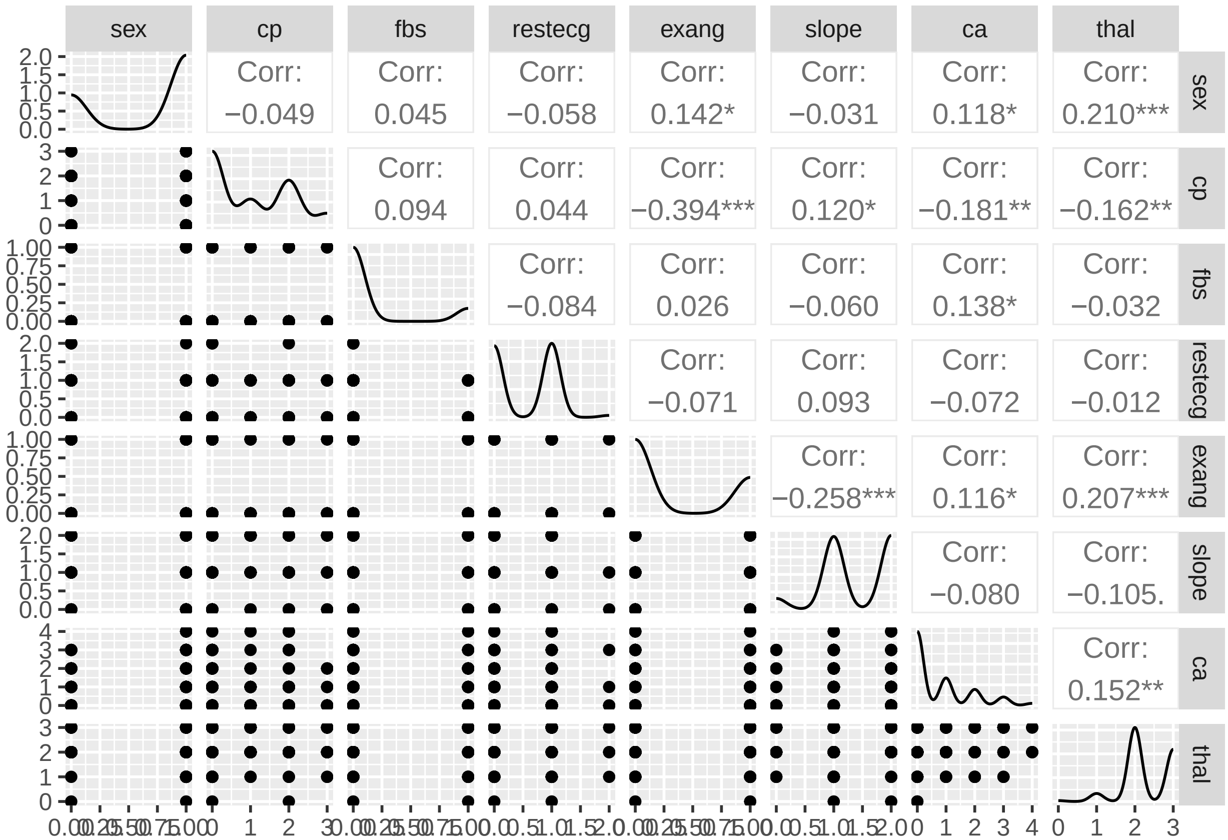
```
## Warning: package 'GGally' was built under R version 4.3.3
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```r
# Plot ggpairs for numerical variables
ggpairs(df[, numerical_vars])
```




```r
# Plot ggpairs for categorical variables
ggpairs(df[, categorical_vars])
```




```r
x <- df[, -which(names(df) == "target")]
y <- df$target

# Set the seed for reproducibility
set.seed(42)

# Split the data into training and testing sets
indices <- sample(1:nrow(x), size = 0.8 * nrow(x), replace = FALSE)
x_train <- x[indices, ]
x_test <- x[-indices, ]
y_train <- y[indices]
y_test <- y[-indices]

y_train <- as.factor(y_train)
y_test <- as.factor(y_test)

data_train <- data.frame(x_train, y_train)
```

```r
# Define different formulas with interaction terms
formula1 <- as.formula("y_train ~ (age + sex +trestbps + chol + fbs + restecg +cp+thalach + exang + oldp
formula2 <- as.formula("y_train ~ .+ cp :fbs +cp:thalach")

# Fit logistic models with interaction terms
logistic_model_interaction1 <- glm(formula1, family = binomial(link = "logit"), data = data_train)
logistic_model_interaction2 <- glm(formula2, family = binomial(link = "logit"), data = data_train)
logistic_model_interaction3 <-step(logistic_model_interaction2,direction = "backward",trace=FALSE)
```

```r
# Print summaries of the models
summary(logistic_model_interaction1)
```

```
##
## Call:
## glm(formula = formula1, family = binomial(link = "logit"), data = data_train)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.883372   3.034433   1.280 0.200626
## age         -0.003245   0.027399  -0.118 0.905732
## sex         -1.812443   0.534734  -3.389 0.000700 ***
## trestbps    -0.031847   0.012891  -2.470 0.013496 *
## chol        -0.002763   0.004180  -0.661 0.508544
## fbs          0.135574   0.596690   0.227 0.820260
## restecg      1.022961   0.421785   2.425 0.015295 *
## cp           0.980034   0.224118   4.373 1.23e-05 ***
## thalach      0.026335   0.012908   2.040 0.041332 *
## exang       -0.763286   0.483756  -1.578 0.114604
## oldpeak     -0.788357   0.277158  -2.844 0.004449 **
## slope        0.375117   0.413027   0.908 0.363765
## ca          -0.752483   0.220871  -3.407 0.000657 ***
## thal        -0.787237   0.352247  -2.235 0.025424 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 332.68  on 241  degrees of freedom
## Residual deviance: 159.48  on 228  degrees of freedom
## AIC: 187.48
##
## Number of Fisher Scoring iterations: 6
```

```r
summary(logistic_model_interaction2)
```

```
##
## Call:
## glm(formula = formula2, family = binomial(link = "logit"), data = data_train)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  5.359577   3.276496   1.636 0.101889
## age          0.002157   0.028137   0.077 0.938883
## sex         -1.783722   0.532228  -3.351 0.000804 ***
## cp          -0.947007   1.531140  -0.618 0.536247
## trestbps    -0.032469   0.012873  -2.522 0.011661 *
## chol        -0.002657   0.004250  -0.625 0.531859
## fbs          0.052230   1.075846   0.049 0.961279
## restecg      1.002210   0.422906   2.370 0.017797 *
## thalach      0.014955   0.015297   0.978 0.328263
## exang       -0.746356   0.487113  -1.532 0.125472
## oldpeak     -0.800581   0.280762  -2.851 0.004352 **
## slope        0.442906   0.422540   1.048 0.294547
```

```
## ca          -0.783300   0.235011  -3.333 0.000859 ***
## thal        -0.832043   0.370974  -2.243 0.024906 *
## cp:fbs       0.036272   0.605150   0.060 0.952204
## cp:thalach   0.012853   0.010245   1.255 0.209625
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 332.68  on 241  degrees of freedom
## Residual deviance: 157.88  on 226  degrees of freedom
## AIC: 189.88
##
## Number of Fisher Scoring iterations: 6
```

```r
summary(logistic_model_interaction3)
```

```
##
## Call:
## glm(formula = y_train ~ sex + cp + trestbps + restecg + thalach +
##     exang + oldpeak + ca + thal, family = binomial(link = "logit"),
##     data = data_train)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.44565    2.38474   1.445 0.148494
## sex         -1.64962    0.48825  -3.379 0.000729 ***
## cp           0.97470    0.21906   4.450 8.61e-06 ***
## trestbps    -0.03260    0.01251  -2.606 0.009168 **
## restecg      1.11153    0.41246   2.695 0.007042 **
## thalach      0.02787    0.01163   2.396 0.016581 *
## exang       -0.79229    0.47770  -1.659 0.097204 .
## oldpeak     -0.92764    0.24229  -3.829 0.000129 ***
## ca          -0.72263    0.21005  -3.440 0.000581 ***
## thal        -0.80591    0.34635  -2.327 0.019972 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 332.68  on 241  degrees of freedom
## Residual deviance: 160.72  on 232  degrees of freedom
## AIC: 180.72
##
## Number of Fisher Scoring iterations: 6
```

Backward Selection Model has lowest AIC VAlue so it chosen as final model.

```r
logistic_model <- glm(formula = y_train ~ sex + cp + trestbps + restecg + thalach +
    exang + oldpeak + ca + thal, family = binomial(link = "logit"),
    data = data_train)
summary_logistic_model <- summary(logistic_model)

# Extract coefficients from the summary
coefficients <- summary_logistic_model$coefficients
```

```r
# Calculate the odds ratio (OR) by exponentiating the coefficients
odds_ratios <- exp(coefficients[, "Estimate"])

# Display the odds ratios
print(odds_ratios)
```

```
## (Intercept)         sex          cp    trestbps     restecg     thalach 
##  31.3636230   0.1921237   2.6503794   0.9679212   3.0389922   1.0282658 
##       exang     oldpeak          ca        thal 
##   0.4528046   0.3954842   0.4854715   0.4466811
```

```r
# Load necessary libraries
library(ROCR)
```

```
## Warning: package 'ROCR' was built under R version 4.3.3
```

```r
predictions <- predict(logistic_model, newdata = data.frame(x_test, y_test), type = "response")

# Create prediction object for ROC curve
prediction_obj <- prediction(predictions, y_test)

library(pROC)
```

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

## 
## Attaching package: 'pROC'

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

```r
roc_obj <- roc(y_test, predictions)
```

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

## Setting direction: controls < cases
```

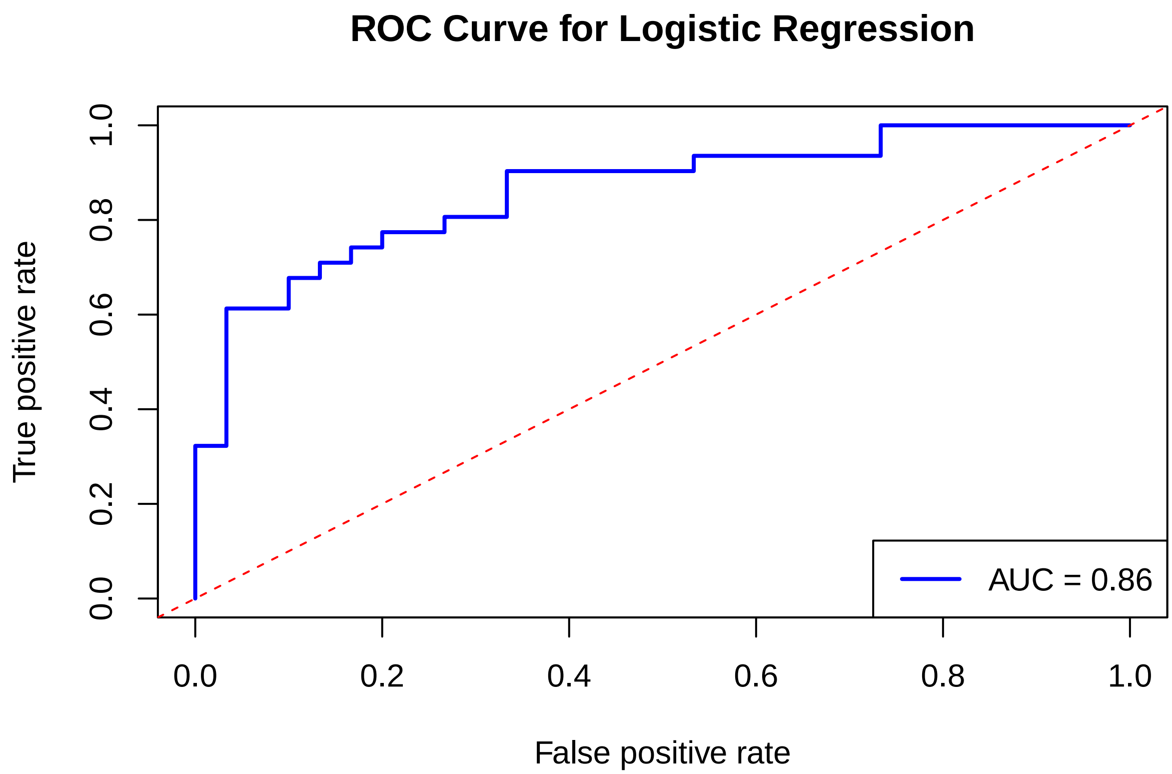
```r
# Find the threshold maximizing Youden's J statistic
optimal_idx <- which.max(roc_obj$sensitivities + roc_obj$specificities - 1)
optimal_threshold <- roc_obj$thresholds[optimal_idx]

# Print the optimal threshold value
cat("Optimal Threshold Value:", optimal_threshold, "\n")
```

```
## Optimal Threshold Value: 0.8400636
```

```r
# Create performance object
performance_obj <- performance(prediction_obj, "tpr", "fpr")

# Plot ROC curve
plot(performance_obj, main = "ROC Curve for Logistic Regression", col = "blue", lwd = 2)

# Calculate AUC
auc_value <- performance(prediction_obj, "auc")@y.values[[1]]

# Print AUC value
cat("AUC value:", auc_value, "\n")
```

```
## AUC value: 0.8623656
```

```r
# Add diagonal reference line
abline(a = 0, b = 1, lty = 2, col = "red")

# Add legend
legend("bottomright", legend = paste("AUC =", round(auc_value, 2)), col = "blue", lwd = 2)
```




```r
# Assuming 'new_data' is the DataFrame containing future data
predictions <- predict(logistic_model, newdata = data.frame(x_test, y_test), type = "response")

predicted_labels <- ifelse(predictions > 0.84, 1, 0)
predicted_labels
```

```
##   7   9  19  21  23  39  44  46  48  50  52  59  64  67  68  77  96 102 105 106
##   0   0   0   0   1   1   0   1   1   1   0   1   1   1   1   0   0   0   1   1
## 117 119 123 132 134 137 145 148 151 163 164 167 169 172 174 176 178 180 204 207
##   0   1   1   1   1   1   1   1   0   1   0   0   0   0   0   0   1   0   0   0
## 209 211 213 214 217 222 233 239 240 242 243 244 253 260 273 274 278 282 284 289
##   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## 301
##   0
```

```r
# Assuming 'y_test' contains the actual labels for the testing data

# Calculate accuracy
accuracy <- mean(predicted_labels == y_test)
```

```r
# Print the accuracy
cat("Logistic Regression Accuracy:", accuracy)
```

```
## Logistic Regression Accuracy: 0.7868852
```

```r
# Define your formula
#formula <- as.formula("y_train ~ (age + sex + cp + trestbps + chol + fbs + restecg + thalach + exang +

# Create the model matrix for back sel.
modM <- with(data_train, model.matrix(~ sex + cp + trestbps + restecg + thalach +
    exang + oldpeak + ca + thal))


# Display the first few rows of the model matrix
head(modM)
```

```
##   (Intercept) sex cp trestbps restecg thalach exang oldpeak ca thal
## 1           1   0  2      128       0     115     0     0.0  0    0
## 2           1   1  3      170       0     155     0     0.6  0    3
## 3           1   1  0      140       0     186     1     0.0  0    2
## 4           1   1  0      120       1     130     1     1.6  0    3
## 5           1   1  1      156       0     143     0     0.0  0    2
## 6           1   1  0      138       0     182     0     0.0  0    2
```

```r
library(class)
library(randomForest)
```

```
## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
# Fit random forest model
rf_model <- randomForest(x_train, y_train, method = 'rf')
rf_model
```

```
##
## Call:
##  randomForest(x = x_train, y = y_train, method = "rf")
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 16.12%
## Confusion matrix:
##    0   1 class.error
## 0 85  23    0.212963
## 1 16 118    0.119403
```

```r
# Evaluate the Random Forest model on the test set
predictions_rf <- predict(rf_model, newdata = x_test)
```

```r
accuracy_rf <- sum(predictions_rf == y_test) / length(y_test)
cat("Random Forest Accuracy:", accuracy_rf, "\n")
```

```
## Random Forest Accuracy: 0.8032787
```

```r
knn_model <- knn(train = x_train, test = x_test, cl = y_train, k = 5)
knn_model
```

```
##  [1] 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 0 0 1 1 1 1 1 1 1 0 0 0 1 1 1 0 0 0 1 0 0 0
## [39] 0 0 1 1 1 0 0 1 0 0 1 1 0 0 0 1 0 1 0 1 1 0 0
## Levels: 0 1
```

```r
# Evaluate the KNN model
accuracy_knn <- sum(knn_model == y_test) / length(y_test)
cat("KNN Accuracy:", accuracy_knn, "\n")
```

```
## KNN Accuracy: 0.6885246
```

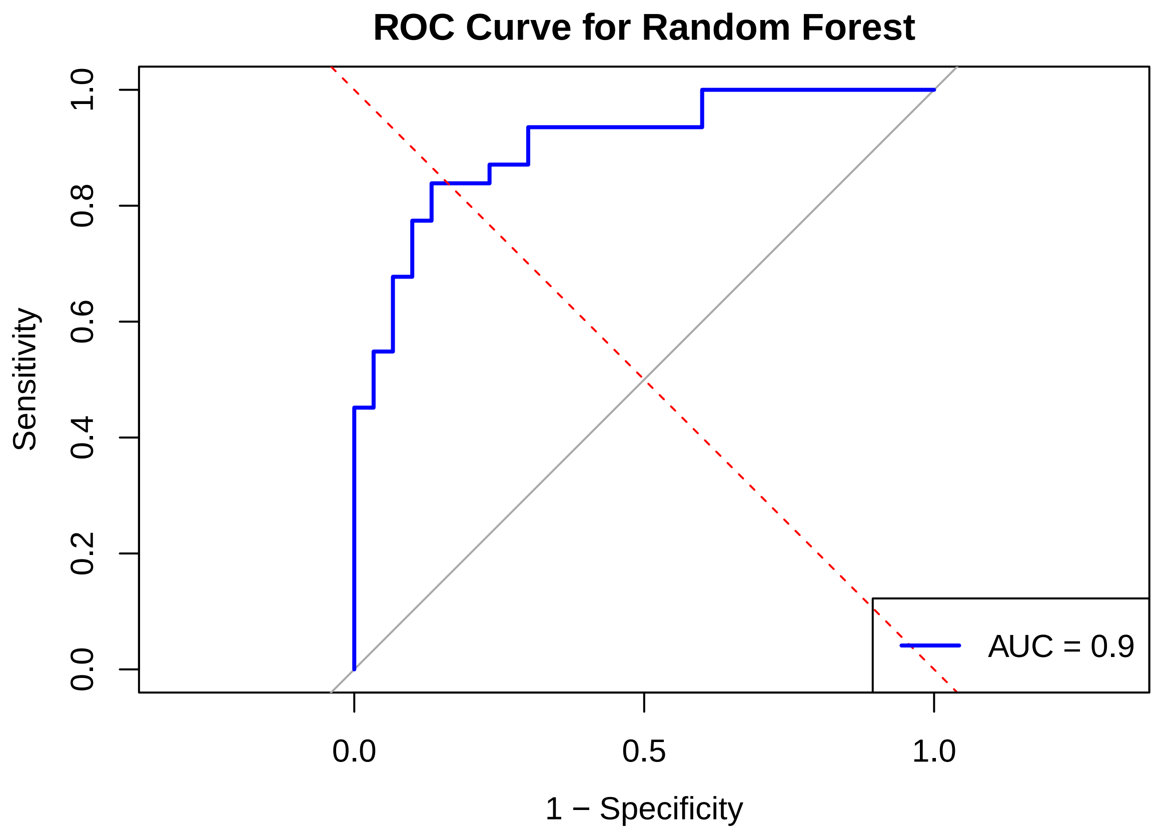
```r
# Get predicted probabilities for positive class from the Random Forest model
rf_probabilities <- predict(rf_model, newdata = x_test, type = "prob")[, 2]

# Create ROC object
roc_obj_rf <- roc(y_test, rf_probabilities)
```

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

## Setting direction: controls < cases
```

```r
# Plot ROC curve
plot(roc_obj_rf, main = "ROC Curve for Random Forest", col = "blue", lwd = 2, legacy.axes = TRUE)

# Add diagonal reference line
abline(a = 0, b = 1, lty = 2, col = "red")

# Calculate AUC
auc_rf <- auc(roc_obj_rf)

# Print AUC value
cat("AUC for Random Forest:", auc_rf, "\n")
```

```
## AUC for Random Forest: 0.9043011
```

```r
# Add legend
legend("bottomright", legend = paste("AUC =", round(auc_rf, 2)), col = "blue", lwd = 2)
```



Cross-validation and hyperparameter tuning

```
# Load necessary libraries
library(caret)
```

```
## Loading required package: lattice
```

```
## Warning: package 'lattice' was built under R version 4.3.3
```

```
# Set up cross-validation
set.seed(123) # For reproducibility
cv <- trainControl(method = "cv", number = 10)

# Define the formula for logistic regression
formula <- as.formula("y_train ~ sex + cp + trestbps + restecg + thalach + exang + oldpeak + ca + thal")

# Fit logistic regression model with cross-validation
logistic_model_cv <- train(formula, data = data_train,
                           method = "glm", family = binomial(link = "logit"),
                           trControl = cv)
logistic_accuracy <- logistic_model_cv$results$Accuracy
print(logistic_model_cv)
```

```
## Generalized Linear Model
##
## 242 samples
##   9 predictor
##   2 classes: '0', '1'
##
```

```
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 217, 218, 217, 218, 217, 219, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8474565  0.6878835
```

```r
# Fit k-nearest neighbors model with cross-validation
knn_model <- train(x = x_train, y = y_train, method = "knn",
                   trControl = cv, tuneLength = 10)
knn_accuracy <- knn_model$results$Accuracy


# Fit random forest model with cross-validation
rf_model <- train(x = x_train, y = y_train, method = "rf",
                  trControl = cv, tuneLength = 10)
rf_accuracy <- rf_model$results$Accuracy
print(rf_model)
```

```
## Random Forest
##
## 242 samples
##  13 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 217, 217, 218, 218, 217, 218, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.8346087  0.6611251
##    3    0.8266087  0.6462583
##    4    0.8306087  0.6549436
##    5    0.8182754  0.6309988
##    6    0.8267899  0.6468062
##    8    0.8267899  0.6488042
##    9    0.8186232  0.6318270
##   10    0.8184420  0.6317161
##   11    0.8146087  0.6234660
##   13    0.8187899  0.6307657
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```r
# Print accuracies
cat("Logistic Regression Accuracy (CV):", logistic_accuracy, "\n")
```

```
## Logistic Regression Accuracy (CV): 0.8474565
```

```r
cat("KNN Accuracy (CV):", knn_accuracy, "\n")
```

```
## KNN Accuracy (CV): 0.6355362 0.6558696 0.6637029 0.6762029 0.6430217 0.6513406 0.6475072 0.6642029 0
```

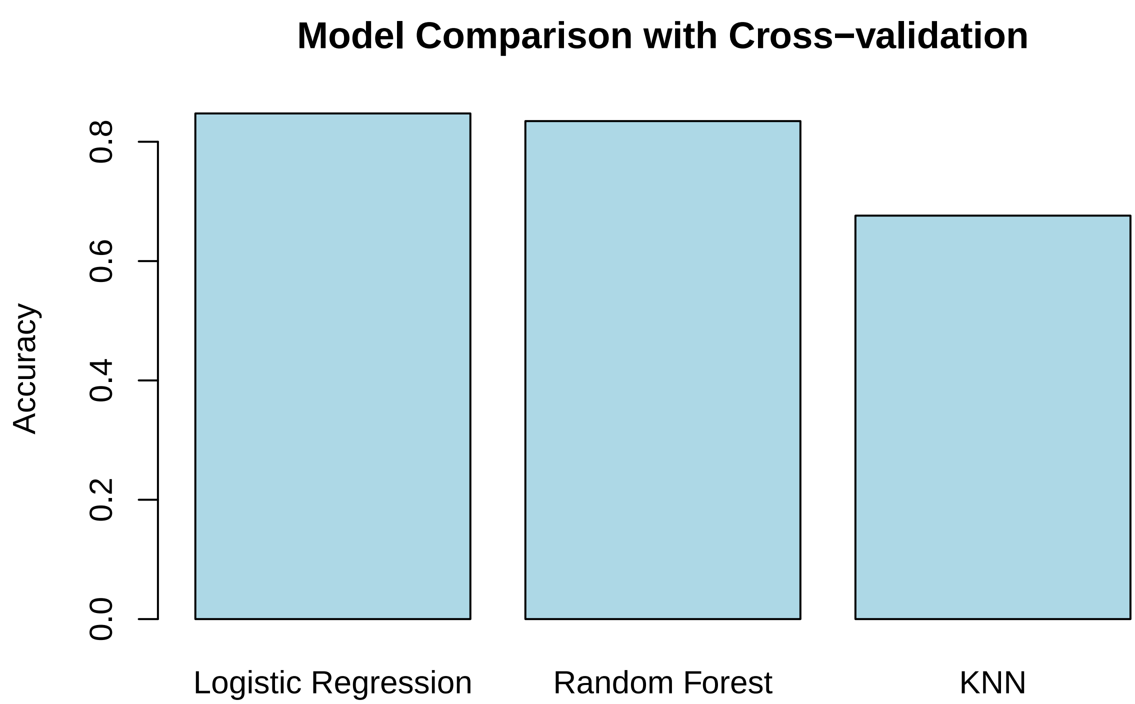
```r
cat("Random Forest Accuracy (CV):", rf_accuracy, "\n")
```

```
## Random Forest Accuracy (CV): 0.8346087 0.8266087 0.8306087 0.8182754 0.8267899 0.8267899 0.8186232 0
```

```r
logistic_accuracy <- max(logistic_model_cv$results$Accuracy)
knn_accuracy <- max(knn_model$results$Accuracy)
rf_accuracy <- max(rf_model$results$Accuracy)

# Compare models
model_compare_cv <- data.frame(
  Model = c("Logistic Regression", "Random Forest","KNN"),
  Accuracy = c(logistic_accuracy, rf_accuracy,knn_accuracy)
)

# Print model accuracies
print(model_compare_cv)
```

```
##                 Model  Accuracy
## 1 Logistic Regression 0.8474565
## 2       Random Forest 0.8346087
## 3                 KNN 0.6762029
```

```r
# Plot the bar graph
barplot(model_compare_cv$Accuracy, names.arg = model_compare_cv$Model, col = "lightblue",
        main = "Model Comparison with Cross-validation", ylab = "Accuracy")
```



Cross-validation is a technique used to estimate the performance of a model on unseen data. It involves splitting the training data into multiple subsets, training the model on a subset, and evaluating it on the remaining data. The reported accuracy after cross-validation (0.84) represents an estimate of how well the model is expected to perform on new, unseen data.

The accuracy obtained after cross-validation (0.84) is likely a more reliable estimate of the model's performance on unseen data. Cross-validation helps to reduce the risk of overfitting by providing a more robust evaluation of the model's generalization ability.

```r
# Load necessary libraries
library(caret)

# Create empty vectors to store metrics
logistic_metrics <- c(NA, NA, NA, NA)
knn_metrics <- c(NA, NA, NA, NA)
rf_metrics <- c(NA, NA, NA, NA)

# Calculate metrics for Logistic Regression
tryCatch({

  logistic_predicted_labels <- ifelse(predictions > 0.84, 1, 0)

  # Confusion matrix for logistic regression
  logistic_conf_matrix <- confusionMatrix(factor(logistic_predicted_labels), factor(y_test))

  # Extract precision, recall, and F1-score for logistic regression
  logistic_metrics[1] <- logistic_conf_matrix$byClass["Pos Pred Value"]
  logistic_metrics[2] <- logistic_conf_matrix$byClass["Sensitivity"]
  logistic_metrics[3] <- logistic_conf_matrix$byClass["F1"]

})

# Calculate metrics for KNN
tryCatch({
  # Predictions on the test data for KNN model
  knn_predictions <- knn(train = x_train, test = x_test, cl = y_train, k = 5)

  # Confusion matrix for KNN
  knn_conf_matrix <- confusionMatrix(factor(knn_predictions), factor(y_test))

  # Extract precision, recall, and F1-score for KNN
  knn_metrics[1] <- knn_conf_matrix$byClass["Pos Pred Value"]
  knn_metrics[2] <- knn_conf_matrix$byClass["Sensitivity"]
  knn_metrics[3] <- knn_conf_matrix$byClass["F1"]

})

# Calculate metrics for Random Forest
tryCatch({
  # Predictions on the test data for Random Forest model
  rf_predictions <- predict(rf_model, newdata = x_test)

  # Confusion matrix for Random Forest
  rf_conf_matrix <- confusionMatrix(factor(rf_predictions), factor(y_test))

  # Extract precision, recall, and F1-score for Random Forest
  rf_metrics[1] <- rf_conf_matrix$byClass["Pos Pred Value"]
  rf_metrics[2] <- rf_conf_matrix$byClass["Sensitivity"]
  rf_metrics[3] <- rf_conf_matrix$byClass["F1"]
```

```r
})

# Create a data frame to store the metrics
metrics_df <- data.frame(
  Model = c("Logistic Regression", "KNN", "Random Forest"),
  Precision = c(logistic_metrics[1], knn_metrics[1], rf_metrics[1]),
  Recall = c(logistic_metrics[2], knn_metrics[2], rf_metrics[2]),
  F1_score = c(logistic_metrics[3], knn_metrics[3], rf_metrics[3])
)
# Print confusion matrix for Logistic Regression
print("Confusion Matrix for Logistic Regression:")
```

```
## [1] "Confusion Matrix for Logistic Regression:"
```

```r
print(logistic_conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 29 12
##          1  1 19
##
##                Accuracy : 0.7869
##                  95% CI : (0.6632, 0.8814)
##     No Information Rate : 0.5082
##     P-Value [Acc > NIR] : 6.823e-06
##
##                   Kappa : 0.5762
##
##  Mcnemar's Test P-Value : 0.005546
##
##             Sensitivity : 0.9667
##             Specificity : 0.6129
##          Pos Pred Value : 0.7073
##          Neg Pred Value : 0.9500
##              Prevalence : 0.4918
##          Detection Rate : 0.4754
##    Detection Prevalence : 0.6721
##       Balanced Accuracy : 0.7898
##
##        'Positive' Class : 0
##
```

```r
# Print confusion matrix for Random Forest
print("Confusion Matrix for Random Forest:")
```

```
## [1] "Confusion Matrix for Random Forest:"
```

```r
print(rf_conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
```

```
##          0 21  3
##          1  9 28
##
##                Accuracy : 0.8033
##                  95% CI : (0.6816, 0.894)
##     No Information Rate : 0.5082
##     P-Value [Acc > NIR] : 1.809e-06
##
##                   Kappa : 0.6052
##
##  Mcnemar's Test P-Value : 0.1489
##
##             Sensitivity : 0.7000
##             Specificity : 0.9032
##          Pos Pred Value : 0.8750
##          Neg Pred Value : 0.7568
##              Prevalence : 0.4918
##          Detection Rate : 0.3443
##    Detection Prevalence : 0.3934
##       Balanced Accuracy : 0.8016
##
##        'Positive' Class : 0
##
```

```r
# Print confusion matrix for KNN
print("Confusion Matrix for KNN:")
```

```
## [1] "Confusion Matrix for KNN:"
```

```r
print(knn_conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 19  8
##          1 11 23
##
##                Accuracy : 0.6885
##                  95% CI : (0.5571, 0.801)
##     No Information Rate : 0.5082
##     P-Value [Acc > NIR] : 0.003287
##
##                   Kappa : 0.3759
##
##  Mcnemar's Test P-Value : 0.646355
##
##             Sensitivity : 0.6333
##             Specificity : 0.7419
##          Pos Pred Value : 0.7037
##          Neg Pred Value : 0.6765
##              Prevalence : 0.4918
##          Detection Rate : 0.3115
##    Detection Prevalence : 0.4426
##       Balanced Accuracy : 0.6876
```

```
##
##        'Positive' Class : 0
##
```

```r
# Print the metrics data frame
print(metrics_df)
```

```
##                 Model Precision    Recall  F1_score
## 1 Logistic Regression 0.7073171 0.9666667 0.8169014
## 2                 KNN 0.7037037 0.6333333 0.6666667
## 3       Random Forest 0.8750000 0.7000000 0.7777778
```

In medical applications, high sensitivity is often crucial, especially when the goal is to avoid missing any positive cases of heart disease. Missing a positive case could mean failing to diagnose and treat someone who has heart disease.

Logistic Regression offers the highest sensitivity (0.9667) in your results, meaning it is most effective at correctly identifying individuals with heart disease.Sensitivity, also known as recall or true positive rate, is a measure of a model's ability to correctly identify positive instances in a dataset. In the context of predicting heart disease, sensitivity measures the proportion of actual heart disease cases that the model correctly identifies as having heart disease.

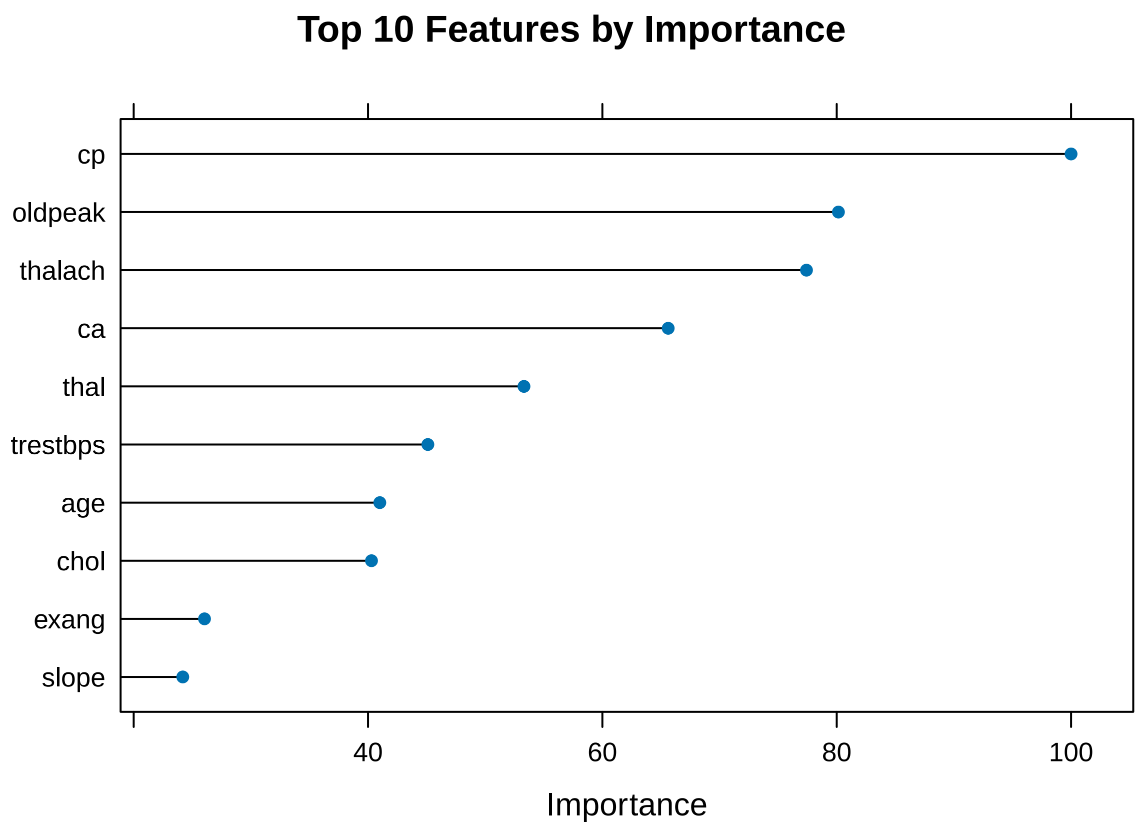
While the Random Forest model may have higher overall performance based on AUC, specificity, and balanced accuracy, but we are right to consider Logistic Regression if we focus is on maximizing sensitivity to minimize false negatives.

A high sensitivity value indicates that the model is good at identifying actual cases of heart disease, which is important in medical diagnosis to ensure that as many individuals with the condition as possible are correctly diagnosed and can receive appropriate treatment.

```r
# Load necessary libraries
library(caret)

# Fit random forest model
rf_model <- train(x = x_train, y = y_train, method = "rf", trControl = cv, tuneLength = 10)

# Get feature importance
rf_feature_importance <- varImp(rf_model)

# Print feature importance
print(rf_feature_importance)
```

```
## rf variable importance
##
##          Overall
## cp        100.00
## oldpeak    80.15
## thalach    77.42
## ca         65.62
## thal       53.31
## trestbps   45.11
## age        41.01
## chol       40.30
## exang      26.05
## slope      24.19
## sex        15.72
## restecg    10.23
## fbs         0.00
```

```r
# Visualize feature importance
plot(rf_feature_importance, top = 10, main = "Top 10 Features by Importance")
```



The RF model considers "cp" (Chest Pain Type) as the most important feature for predicting heart disease, followed by "thalach" (Maximum Heart Rate Achieved) and "oldpeak" (ST Depression Induced by Exercise Relative to Rest). These findings provide insights into which features are most influential in the model's predictions.

#Feature importance

```r
# Define your logistic regression model
model <- glm(y_train ~ sex + cp + trestbps + restecg + thalach +
    exang + oldpeak + ca + thal, family = binomial(link = "logit"), data = data_train)

# Fit the logistic regression model
fit <- model

# Extract coefficients from the fitted model
coefficients <- coef(fit)
coefficients
```

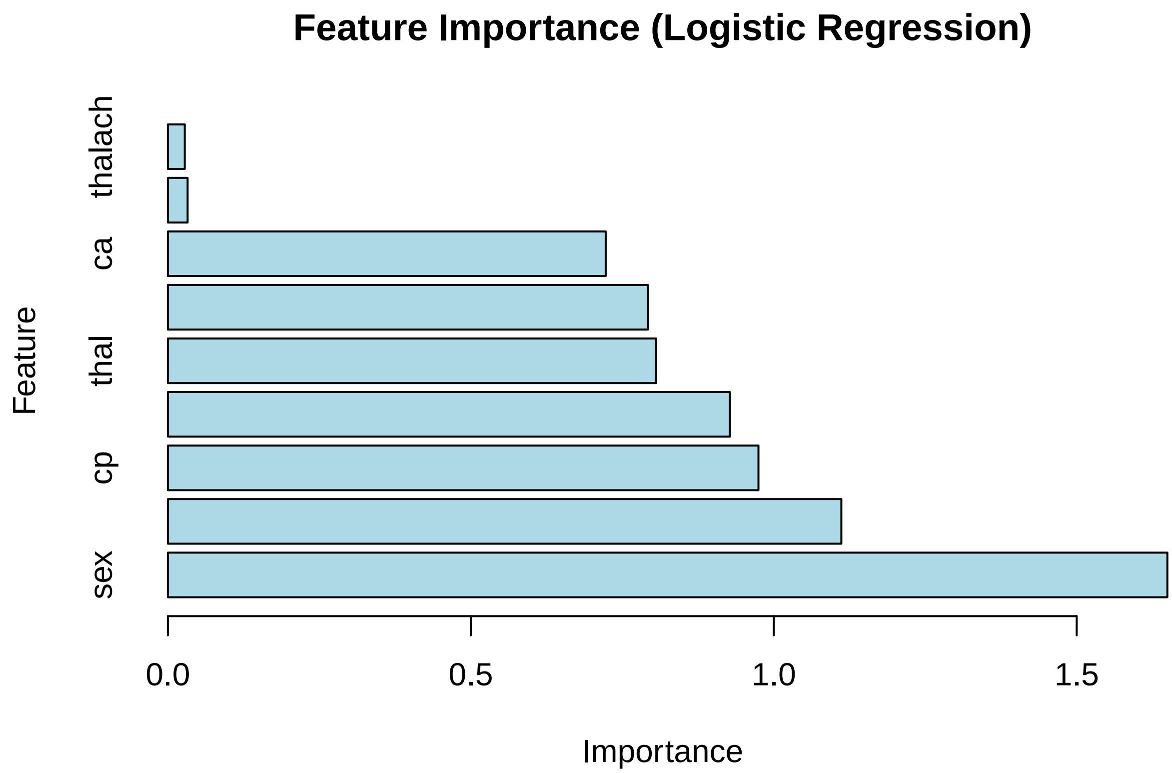
```
## (Intercept)         sex          cp    trestbps     restecg     thalach
##  3.44564872 -1.64961565  0.97470280 -0.03260461  1.11152596  0.02787371
##       exang     oldpeak          ca        thal
## -0.79229469 -0.92764446 -0.72263459 -0.80591028
```

```r
# Create a named vector with feature names as keys and coefficients as values
feature_dict <- as.vector(coefficients[-1])  # Exclude the intercept term
names(feature_dict) <- names(coefficients)[-1]  # Use column names as keys
```

```r
# Assuming you have your feature_dict in R as a named vector
feature_names <- names(feature_dict)
feature_values <- abs(unname(feature_dict))  # Take absolute values for magnitude

# Create a dataframe
feature_df <- data.frame(Feature = feature_names, Importance = feature_values)

# Sort the dataframe by importance
feature_df <- feature_df[order(feature_df$Importance, decreasing = TRUE), ]
feature_df
```

```
##    Feature Importance
## 1      sex 1.64961565
## 4  restecg 1.11152596
## 2       cp 0.97470280
## 7  oldpeak 0.92764446
## 9     thal 0.80591028
## 6    exang 0.79229469
## 8       ca 0.72263459
## 3 trestbps 0.03260461
## 5  thalach 0.02787371
```

```r
# Plot the bar plot
barplot(height = feature_df$Importance,
        names.arg = feature_df$Feature,
        main = "Feature Importance (Logistic Regression)",
        xlab = "Importance",
        ylab = "Feature",
        col = "lightblue",
        horiz = TRUE)
```


Feature Importance (Logistic Regression)
Feature
thalach
ca
thal
cp
sex
0.0
0.5
1.0
1.5
Importance