FORECASTING HEART DISEASE RISKS

Seif Kungulio

October 25, 2025

Contents

This page was deliberately left blank

1 Business Understanding

1.1 Introduction

Cardiovascular diseases remain one of the leading causes of death worldwide, imposing significant clinical and financial burdens on individuals and organizations alike. Early identification of people at risk of developing heart disease is therefore a priority not only in healthcare but also in sectors such as insurance and public health policy. For insurance companies in particular, understanding the likelihood that a policyholder will develop heart disease is crucial for assessing health risks, pricing premiums, and designing preventive wellness programs that promote healthier lifestyles among clients.

This project applies predictive analytics and machine learning techniques to the Heart Disease dataset from the UCI Machine Learning Repository. The dataset contains a diverse range of patient attributes—including demographic, physiological, and clinical indicators—that are known to influence cardiovascular health. By systematically analyzing these variables, the project aims to uncover patterns and risk factors associated with the onset of heart disease.

The broader business objective is to build a data-driven decision support system that can forecast an individual's probability of developing heart disease. Such a model enables insurers to perform risk stratification, design personalized premium plans, and support preventive health initiatives that lower long-term costs. Through this analytical framework, the project demonstrates how data science can transform raw medical data into actionable business intelligence, fostering both economic efficiency and social impact.

1.2 Problem statement

Problem statement

To develop models for an insurance company using the Heart Disease dataset from the UCI Machine Learning Repository. The goal is to predict the likelihood of a person developing heart disease, which would help the insurance company estimate health risks and adjust premiums accordingly.

2 Data Understanding

The dataset contains various features related to patients' health and demographic information. We will explore the dataset to understand its structure and relationships between variables.

2.1 Data description

The Heart Disease dataset from the UCI Machine Learning Repository contains 303 instances and 14 attributes. These attributes include both numerical and categorical variables related to patients' health metrics and demographic information. The target variable indicates the presence or absence of heart disease. These attributes are:

1. `age`: Age of the patient (numeric)
2. `sex`: Gender of the patient (1 = male, 0 = female)
3. `cp`: Chest pain type (categorical: 1-4)
4. `trestbps`: Resting blood pressure (numeric)
5. `chol`: Serum cholesterol (numeric)
6. `fbs`: Fasting blood sugar (1 = true, 0 = false)
7. `restecg`: Resting electrocardiographic results (categorical)
8. `thalach`: Maximum heart rate achieved (numeric)
9. `exang`: Exercise-induced angina (1 = yes, 0 = no)
10. `oldpeak`: ST depression induced by exercise (numeric)
11. `slope`: The slope of the peak exercise ST segment (categorical)
12. `ca`: Number of major vessels (0-3, numeric)
13. `thal`: Thalassemia (categorical: 1 = normal, 2 = fixed defect, 3 = reversible defect)
14. `target`: Heart disease (1 = disease, 0 = no disease)

2.2 Data dictionary

The dataset contains 14 key attributes that are either numerical or categorical.

Attribute	Type	Description	Constraints/ Rules
`age`	Numerical	The age of the patient in years	Range: 29-77 (based on dataset statistics)
`sex`	Categorical	The gender of the patient	Values: 1 = Male, 0 = Female
`cp`	Categorical	Type of chest pain experienced by the patient	Values: 1 = Typical angina, 2 = Atypical angina, 3 = Non-anginal pain, 4 = Asymptomatic
`trestbps`	Numerical	Resting blood pressure of the patient, measured in mmHg	Range: Typically, between 94 and 200 mmHg
`chol`	Numerical	Serum cholesterol level in mg/dl	Range: Typically, between 126 and 564 mg/dl
`fbs`	Categorical	Fasting blood sugar level > 120 mg/dl	Values: 1 = True, 0 = False

Attribute	Type	Description	Constraints/ Rules
`restecg`	Categorical	Results of the patient's resting electrocardiogram	Values: 0 = Normal, 1 = ST-T wave abnormality, 2 = Probable or definite left ventricular hypertrophy
`thalach`	Numerical	Maximum heart rate achieved during a stress test	Range: Typically, between 71 and 202 bpm
`exang`	Categorical	Whether the patient experiences exercise-induced angina	Values: 1 = Yes, 0 = No
`oldpeak`	Numerical	ST depression induced by exercise relative to rest (an ECG measure)	Range: 0.0 to 6.2 (higher values indicate more severe abnormalities)
`slope`	Categorical	Slope of the peak exercise ST segment	Values: 1 = Upsloping, 2 = Flat, 3 = Downsloping
`ca`	Numerical	Number of major vessels colored by fluoroscopy	Range: 0-3
`thal`	Categorical	Blood disorder variable related to thalassemia	Values: 3 = Normal, 6 = Fixed defect, 7 = Reversible defect
`target`	Categorical	Diagnosis of heart disease	Values: 0 = No heart disease, 1 = Presence of heart disease

2.3 Initial observations

- The dataset contains a mix of numerical and categorical variables.
- Some variables may require preprocessing, such as handling missing values and encoding categorical variables.
- Missing Values: Some fields like ca and thal may have missing values or unknown entries ('?').
- Data Types: Some categorical variables are encoded numerically and will need to be interpreted correctly during analysis.
- Class Imbalance: Preliminary checks suggest the dataset is relatively balanced between presence and absence of disease, but this will be verified.
- Outliers: Numerical fields such as chol (cholesterol) and trestbps (blood pressure) may have outliers that need to be detected and considered in analysis.

3 Data Preparation

3.1 Data loading

Load the dataset from the UCI website to memory

```
# Load the dataset
url <- "https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.cleveland.data

# Read the dataset into a dataframe
Heart.df <- read.csv(text = getURL(url), header = FALSE, na.strings = "?")
```

Rename the columns into a meaningful column names

```
colnames(Heart.df) <- c("age", "sex", "cp", "trestbps", "chol", "fbs", "restecg",
                        "thalach", "exang", "oldpeak", "slope", "ca", "thal", "target")
```

Display dimensions of the dataset

```
dim(Heart.df)
```

```
## [1] 303  14
```

Display the first six rows of the dataset

```
head(Heart.df)
```

```
##   age sex cp trestbps chol fbs restecg thalach exang oldpeak slope ca thal
## 1  63   1  1      145  233   1       2     150     0     2.3     3  0    6
## 2  67   1  4      160  286   0       2     108     1     1.5     2  3    3
## 3  67   1  4      120  229   0       2     129     1     2.6     2  2    7
## 4  37   1  3      130  250   0       0     187     0     3.5     3  0    3
## 5  41   0  2      130  204   0       2     172     0     1.4     1  0    3
## 6  56   1  2      120  236   0       0     178     0     0.8     1  0    3
##   target
## 1      0
## 2      2
## 3      1
## 4      0
## 5      0
## 6      0
```

Display the structure of the dataframe

```
glimpse(Heart.df)
```

```
## Rows: 303
## Columns: 14
## $ age      <dbl> 63, 67, 67, 37, 41, 56, 62, 57, 63, 53, 57, 56, 56, 44, 52, 5~
## $ sex      <dbl> 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1~
```

```
## $ cp       <dbl> 1, 4, 4, 3, 2, 2, 4, 4, 4, 4, 4, 2, 3, 2, 3, 3, 2, 4, 3, 2, 1~
## $ trestbps <dbl> 145, 160, 120, 130, 130, 120, 140, 120, 130, 140, 140, 140, 1~
## $ chol     <dbl> 233, 286, 229, 250, 204, 236, 268, 354, 254, 203, 192, 294, 2~
## $ fbs      <dbl> 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0~
## $ restecg  <dbl> 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 2~
## $ thalach  <dbl> 150, 108, 129, 187, 172, 178, 160, 163, 147, 155, 148, 153, 1~
## $ exang    <dbl> 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1~
## $ oldpeak  <dbl> 2.3, 1.5, 2.6, 3.5, 1.4, 0.8, 3.6, 0.6, 1.4, 3.1, 0.4, 1.3, 0~
## $ slope    <dbl> 3, 2, 2, 3, 1, 1, 3, 1, 2, 3, 2, 2, 2, 1, 1, 1, 3, 1, 1, 1, 2~
## $ ca       <dbl> 0, 3, 2, 0, 0, 0, 2, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0~
## $ thal     <dbl> 6, 3, 7, 3, 3, 3, 3, 3, 7, 7, 6, 3, 6, 7, 7, 3, 7, 3, 3, 3, 3~
## $ target   <int> 0, 2, 1, 0, 0, 0, 3, 0, 2, 1, 0, 0, 2, 0, 0, 0, 1, 0, 0, 0, 0~
```

Display the statistical summary of the dataframe

```
summary(Heart.df)
```

```
##       age             sex              cp           trestbps
##  Min.   :29.00   Min.   :0.0000   Min.   :1.000   Min.   : 94.0
##  1st Qu.:48.00   1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:120.0
##  Median :56.00   Median :1.0000   Median :3.000   Median :130.0
##  Mean   :54.44   Mean   :0.6799   Mean   :3.158   Mean   :131.7
##  3rd Qu.:61.00   3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:140.0
##  Max.   :77.00   Max.   :1.0000   Max.   :4.000   Max.   :200.0
##
##       chol            fbs            restecg          thalach
##  Min.   :126.0   Min.   :0.0000   Min.   :0.0000   Min.   : 71.0
##  1st Qu.:211.0   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:133.5
##  Median :241.0   Median :0.0000   Median :1.0000   Median :153.0
##  Mean   :246.7   Mean   :0.1485   Mean   :0.9901   Mean   :149.6
##  3rd Qu.:275.0   3rd Qu.:0.0000   3rd Qu.:2.0000   3rd Qu.:166.0
##  Max.   :564.0   Max.   :1.0000   Max.   :2.0000   Max.   :202.0
##
##      exang           oldpeak         slope             ca
##  Min.   :0.0000   Min.   :0.00   Min.   :1.000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.00   1st Qu.:1.000   1st Qu.:0.0000
##  Median :0.0000   Median :0.80   Median :2.000   Median :0.0000
##  Mean   :0.3267   Mean   :1.04   Mean   :1.601   Mean   :0.6722
##  3rd Qu.:1.0000   3rd Qu.:1.60   3rd Qu.:2.000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :6.20   Max.   :3.000   Max.   :3.0000
##                                                  NA's   :4
##       thal           target
##  Min.   :3.000   Min.   :0.0000
##  1st Qu.:3.000   1st Qu.:0.0000
##  Median :3.000   Median :0.0000
##  Mean   :4.734   Mean   :0.9373
##  3rd Qu.:7.000   3rd Qu.:2.0000
##  Max.   :7.000   Max.   :4.0000
##  NA's   :2
```

3.2 Data preprocessing

We will preprocess the data by handling missing values, encoding categorical variables, and scaling numerical features.

Convert binary variables to (0, 1)

According to the data dictionary, the following attributes should be binary variables: `sex`, `fbs`, `exang`, and `target`. But, some shows to have values besides 0's and 1's. Let's convert binary variables to (0, 1)

```
Heart.df$target <- ifelse(Heart.df$target > 0, 1, 0)
Heart.df$sex <- ifelse(Heart.df$sex > 0, 1, 0)
Heart.df$fbs <- ifelse(Heart.df$fbs > 0, 1, 0)
Heart.df$exang <- ifelse(Heart.df$exang > 0, 1, 0)
```

Handle missing values

Handle missing values in `ca` and `thal` variables using mean/mode imputation.

```
Heart.df$ca[is.na(Heart.df$ca)] <- median(Heart.df$ca, na.rm = TRUE)
Heart.df$ca[Heart.df$ca == "?"] <- median(Heart.df$ca, na.rm = TRUE)
Heart.df$thal[is.na(Heart.df$thal)] <- median(Heart.df$thal, na.rm = TRUE)
Heart.df$ca[Heart.df$thal == "?"] <- median(Heart.df$thal, na.rm = TRUE)
```

Check for missing values if still exist

```
sapply(Heart.df, function(x) sum(is.na(x)))
```

```
##      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
```

Handle duplicate entries

Check for duplicate entries and print them if they exist.

```
dupes <- Heart.df[duplicated(Heart.df) | duplicated(Heart.df, fromLast = TRUE), ]
print(dupes)
```

```
##  [1] age      sex      cp       trestbps chol     fbs      restecg  thalach
##  [9] exang    oldpeak  slope    ca       thal     target
## <0 rows> (or 0-length row.names)
```

Convert categorical variables to factor.

Define a list of categorical columns with their levels and labels

```
categorical_columns <- list(
  sex = list(levels = c(0, 1), labels = c("Female", "Male")),
  cp = list(levels = c(1, 2, 3, 4), labels = c("Typical Angina",
                                               "Atypical Angina", "Non-Angina",
```

```
                                                "Asymptomatic")),
  fbs = list(levels = c(0, 1), labels = c("False", "True")),
  restecg = list(levels = c(0, 1, 2), labels = c("Normal", "Wave-abnormality", "Probable")),
  exang = list(levels = c(0, 1), labels = c("No", "Yes")),
  slope = list(levels = c(1, 2, 3), labels = c("Upsloping", "Flat",
                                                "Downsloping")),
  thal = list(levels = c(3, 6, 7), labels = c("Normal", "Fixed Defect", "Reversible")),
  target = list(levels = c(1, 0), labels = c("Yes", "No"))
)
```

Apply the factor transformation using a for-loop.

```
for (col in names(categorical_columns)) {
  Heart.df[[col]] <- factor(Heart.df[[col]],
                            levels = categorical_columns[[col]]$levels,
                            labels = categorical_columns[[col]]$labels)
}
```

Handle outliers in numerical variables

Apply multiple filters to identify and handle outliers in numerical variables.

```
Heart.df <- Heart.df[Heart.df$age > 40 &
                       Heart.df$trestbps < 170 &
                       Heart.df$chol < 340 &
                       Heart.df$chol > 150 &
                       Heart.df$thalach > 115 &
                       Heart.df$oldpeak < 2.4, ]
```

This section of the analysis performs a crucial data-cleaning step aimed at refining the quality of the dataset before modeling and visualization. The filtering operation applies a set of logical conditions to remove extreme or biologically implausible values from key continuous health indicators such as age, blood pressure, cholesterol, heart rate, and ST depression. By doing so, it ensures that the dataset reflects realistic patient characteristics and minimizes the influence of outliers that could distort statistical interpretation or predictive accuracy.

The first filter, `Heart.df$age > 40`, narrows the focus to patients over 40 years of age. This decision is grounded in clinical reasoning—heart disease is relatively uncommon in younger individuals, and including them could introduce noise rather than insight into cardiovascular risk patterns. The next condition, `Heart.df$trestbps < 170`, restricts resting blood pressure to physiologically typical values, removing excessively high readings that may result from measurement error or rare hypertensive crises.

Similarly, cholesterol values are filtered using two constraints: `Heart.df$chol < 340` and `Heart.df$chol > 150`. This dual boundary ensures that cholesterol readings fall within a realistic clinical range, excluding both unusually low and excessively high values. Extremely high cholesterol levels (above 340 mg/dl) could be outliers due to lab errors or rare genetic conditions, while very low levels (below 150 mg/dl) are equally atypical for this patient population.

The condition `Heart.df$thalach > 115` retains only those patients whose maximum heart rate achieved during exercise falls within a normal performance range. Extremely low thalach values often suggest incomplete stress tests or data entry errors, which could bias the interpretation of cardiovascular efficiency. Finally, `Heart.df$oldpeak < 2.4` removes extreme ST depression values. In clinical terms, oldpeak measures the degree of ST segment depression during exercise,

and values beyond 2.4 are uncommon and may represent atypical cardiac events that do not align with general population patterns in the dataset.

Overall, these filters collectively enhance the integrity of the data and the reliability of subsequent analysis. By trimming implausible extremes, the dataset becomes more homogeneous, improving the clarity of boxplots, histograms, and scatterplots generated during exploratory data analysis. Moreover, this targeted filtering supports more stable and interpretable model outcomes by preventing a few extreme observations from disproportionately influencing trends or coefficients. The result is a dataset that better represents realistic health profiles, ultimately strengthening the credibility of insights drawn from the heart disease risk modeling process.

3.3 Helper functions

Function to create Box plots

```
HeartDiseaseBoxplot <- function(var1, var2) {
  ggplot(Heart.df, aes(x = .data[[var1]],
                       y = .data[[var2]],
                       fill = .data[[var1]])) +
    geom_boxplot() +
    labs(title = paste("Boxplot of", var2, "by", var1),
         x = var1, y = var2, fill = "Heart Disease")
}
```

Function to create Bar plots

```
HeartDiseaseBar <- function(var) {
  ggplot(Heart.df, aes(x = .data[[var]], fill = target)) +
    geom_bar(position = "dodge") +
    labs(title = paste("Distribution of Heart Disease by", var),
         x = var, fill = "Heart Disease")
}
```

Function to create Histograms

```
HeartDiseaseHist <- function(var1) {
  ggplot(Heart.df, aes(x = .data[[var1]], fill = target)) +
    geom_histogram(bins = 15) +
    labs(title = paste("Distribution of", var1),
         x = var1, fill = "Heart Disease")
}
```

Function to create Scatter plots

```
HeartDiseaseScatter <- function(point1, point2){
  ggplot(Heart.df, aes(x = .data[[point1]],
                       y = .data[[point2]],
                       color = target)) +
    geom_point(size = 2) +
    geom_smooth(method = "lm", se = FALSE, color = "blue", formula = y ~ x) +
    labs(title = paste("Scatterplot of", point1, "by", point2),
       x = point1, y = point2, color = "Heart Disease")
}
```

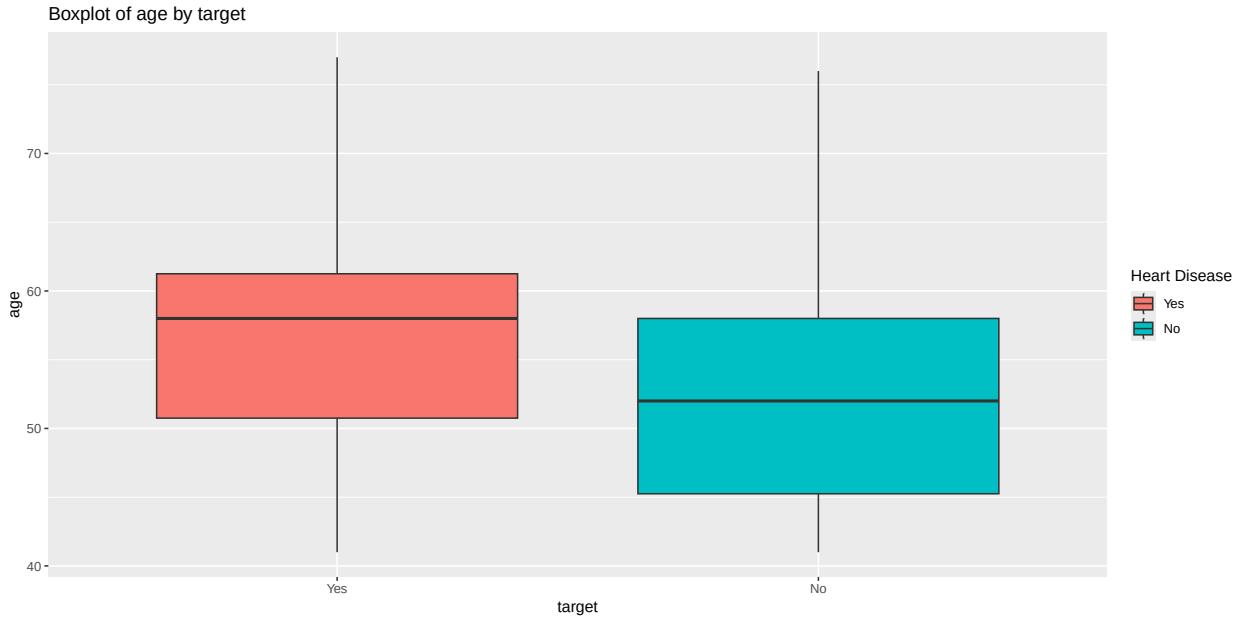
3.4 Exploratory data analysis

3.4.1 Boxplots for numerical variables

I used boxplots to visually examine the distribution of key continuous health indicators — such as age, resting blood pressure (trestbps), cholesterol (chol), maximum heart rate (thalach), and ST depression (oldpeak) — across the binary target variable (Heart Disease: Yes / No). Boxplots were chosen because they efficiently highlight differences in central tendency (median), variability (IQR), and the presence of potential outliers between patients with and without heart disease.

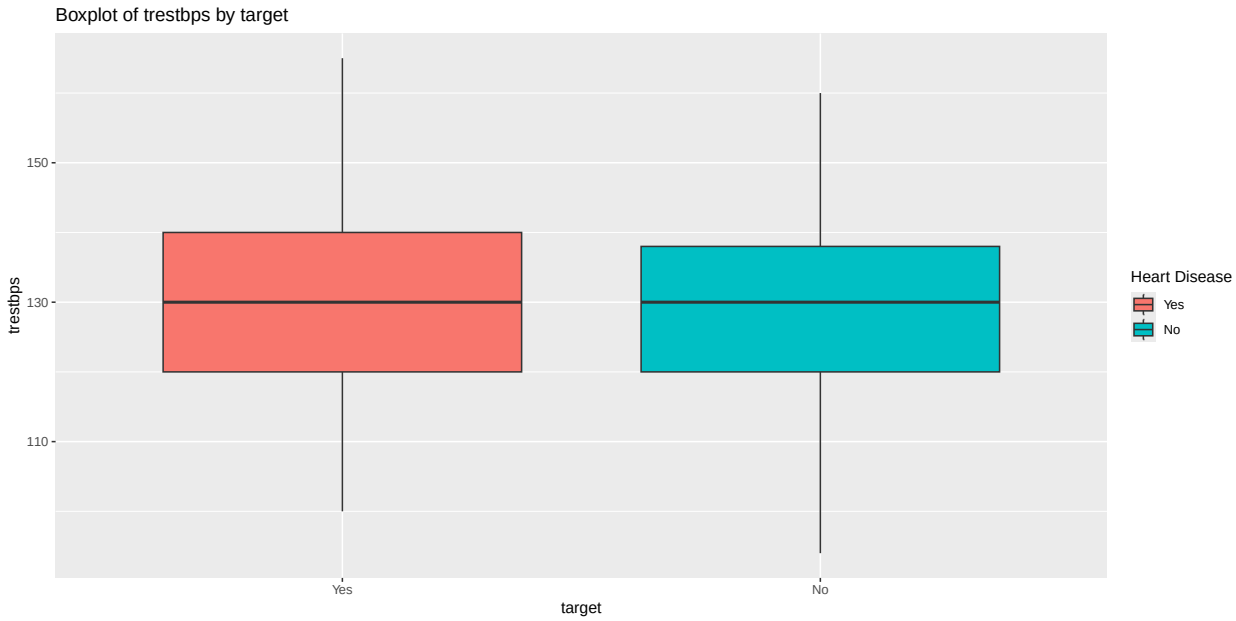
Boxplot of Age by Heart Disease

```
HeartDiseaseBoxplot("target", "age")
```



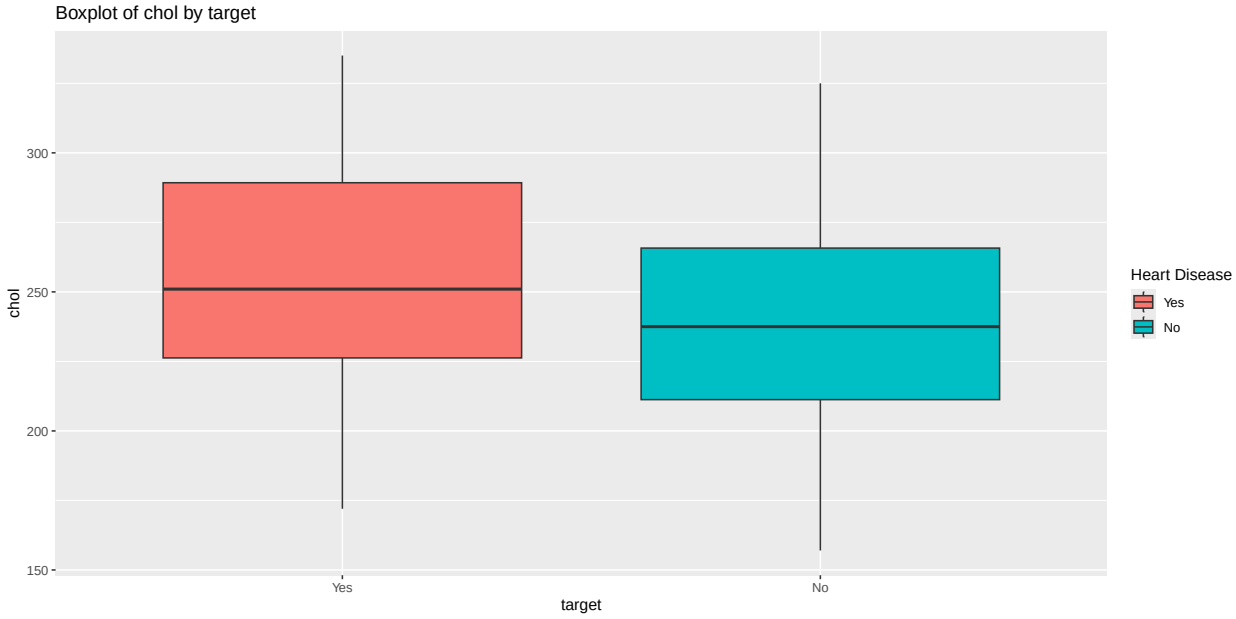
Boxplot of Resting Blood Pressure (trestbps) by Heart Disease

```
HeartDiseaseBoxplot("target", "trestbps")
```



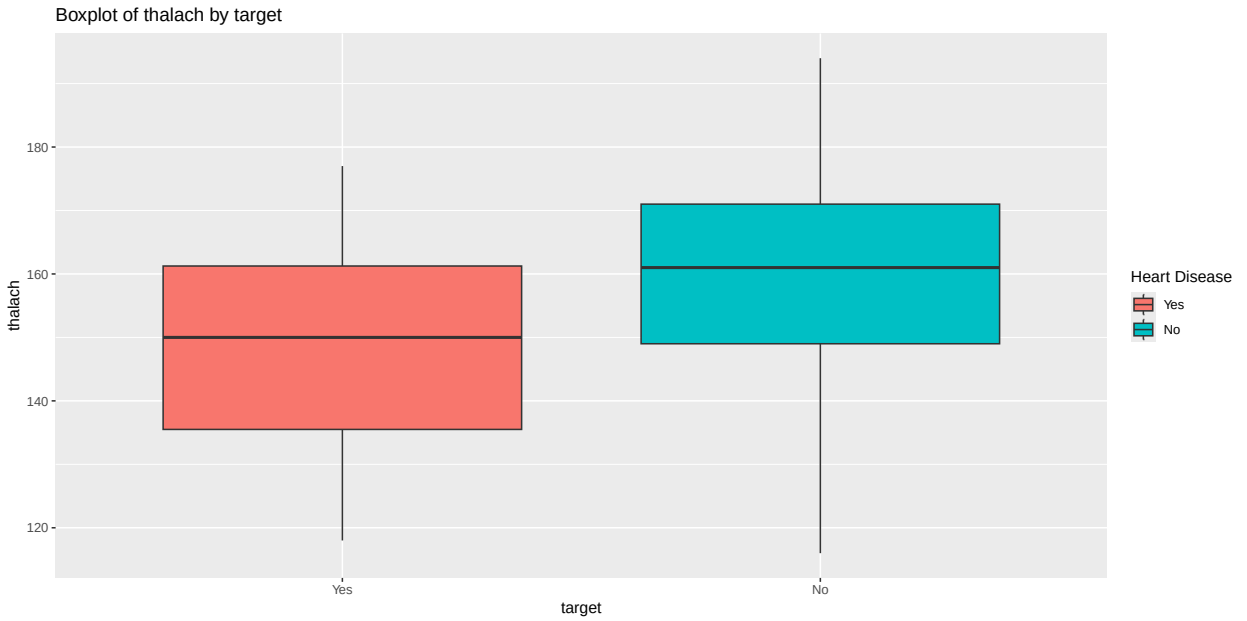
Boxplot of Cholesterol (chol) by Heart Disease

```
HeartDiseaseBoxplot("target", "chol")
```



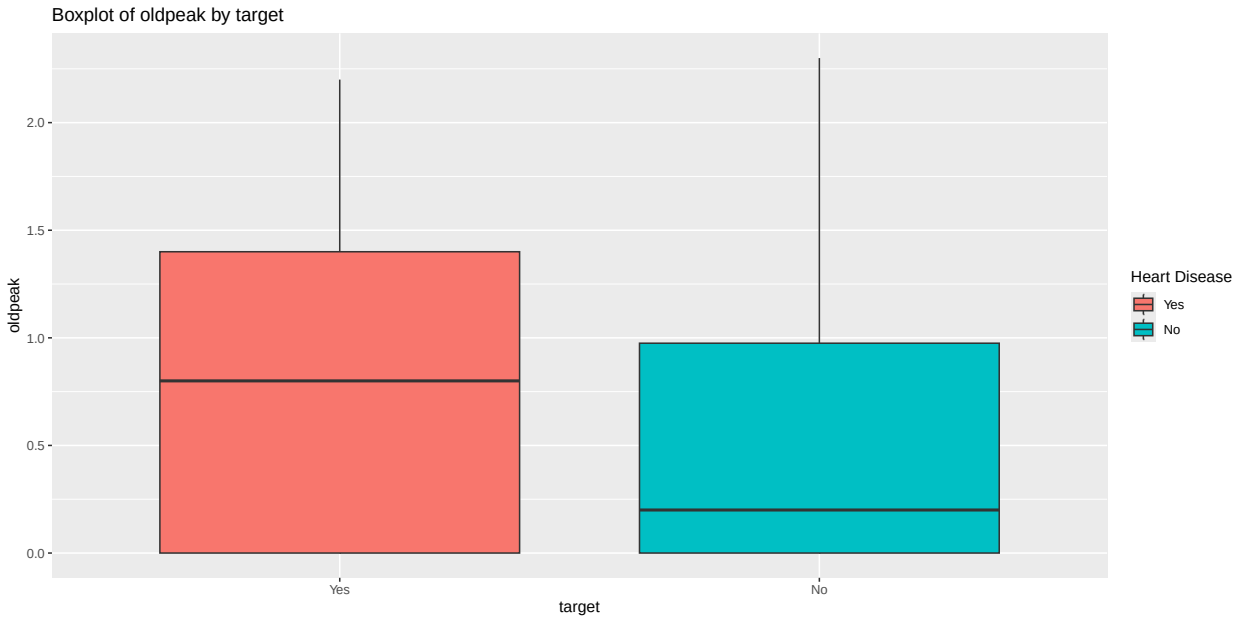
Boxplot of Maximum Heart Rate Achieved (thalach) by Heart Disease

```
HeartDiseaseBoxplot("target", "thalach")
```



Boxplot of ST Depression (oldpeak) by Heart Disease

```
HeartDiseaseBoxplot("target", "oldpeak")
```



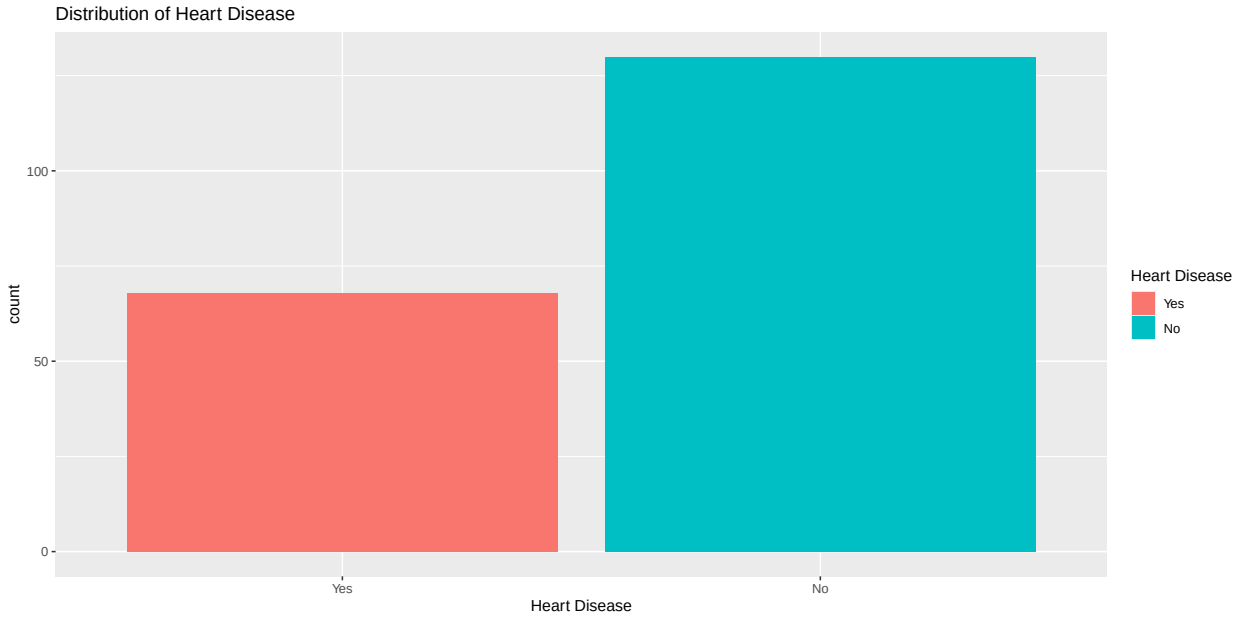
Overall boxplots observations:

3.4.2 Barplots for categorical variables

Heart disease distribution

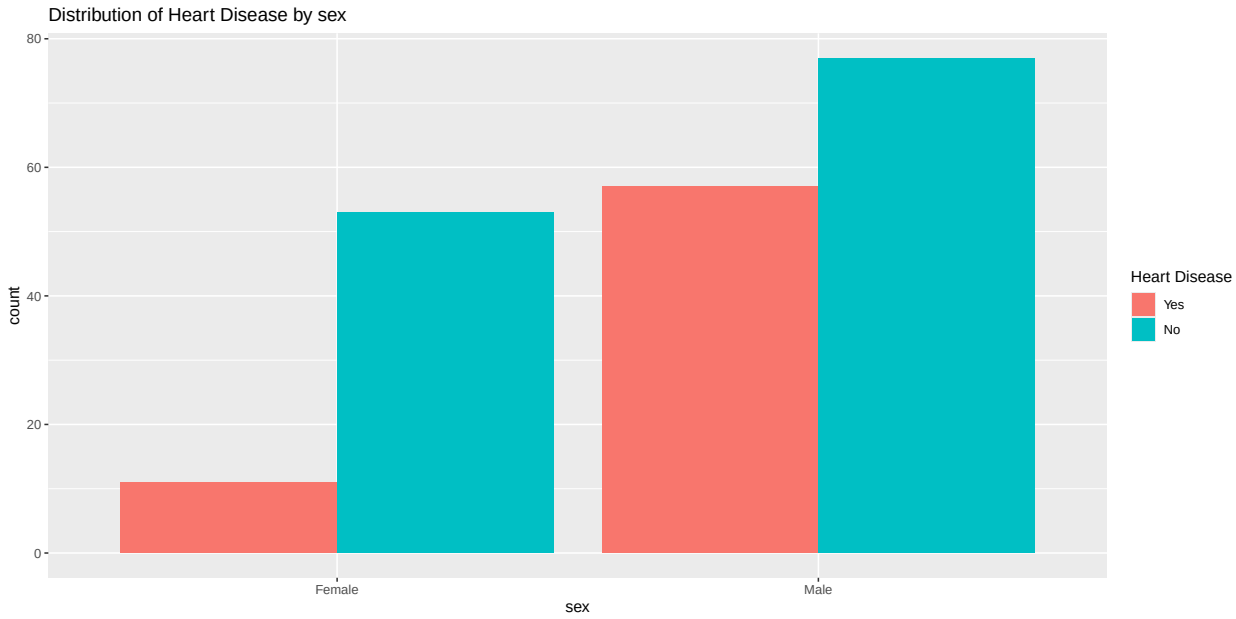
```
ggplot(Heart.df, aes(x=target, fill=target))+
  geom_bar() +
```

```
  ggtitle("Distribution of Heart Disease") +
  labs(x = "Heart Disease", fill = "Heart Disease")
```

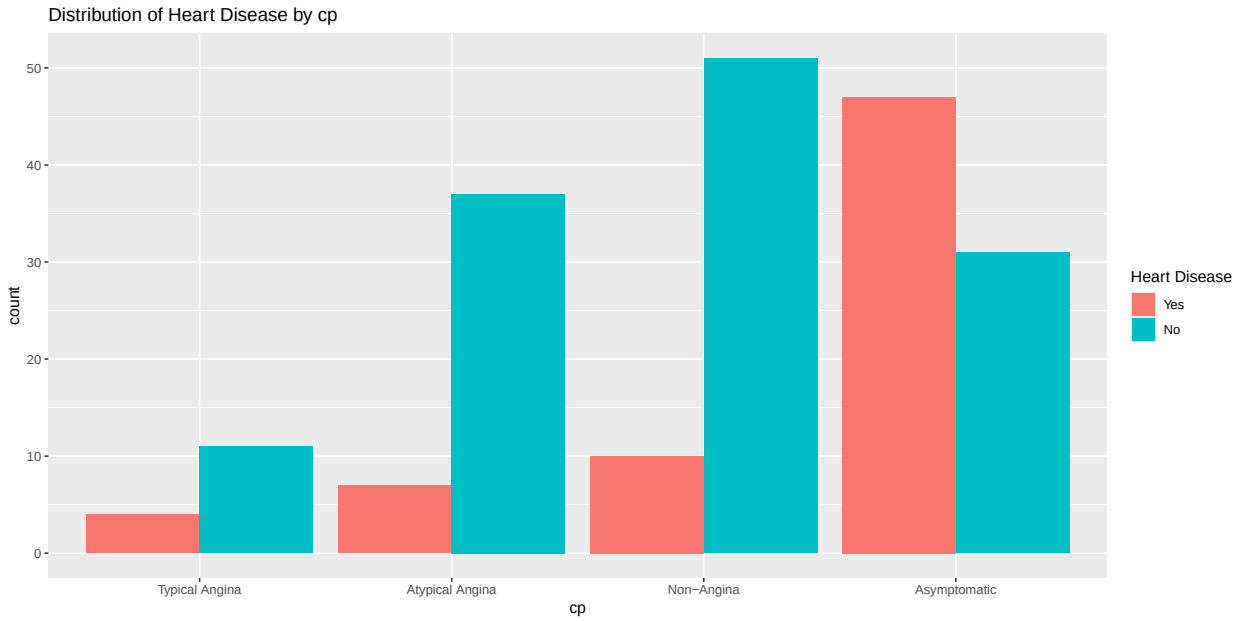


Visualize distribution of categorical variables by heart disease presence.

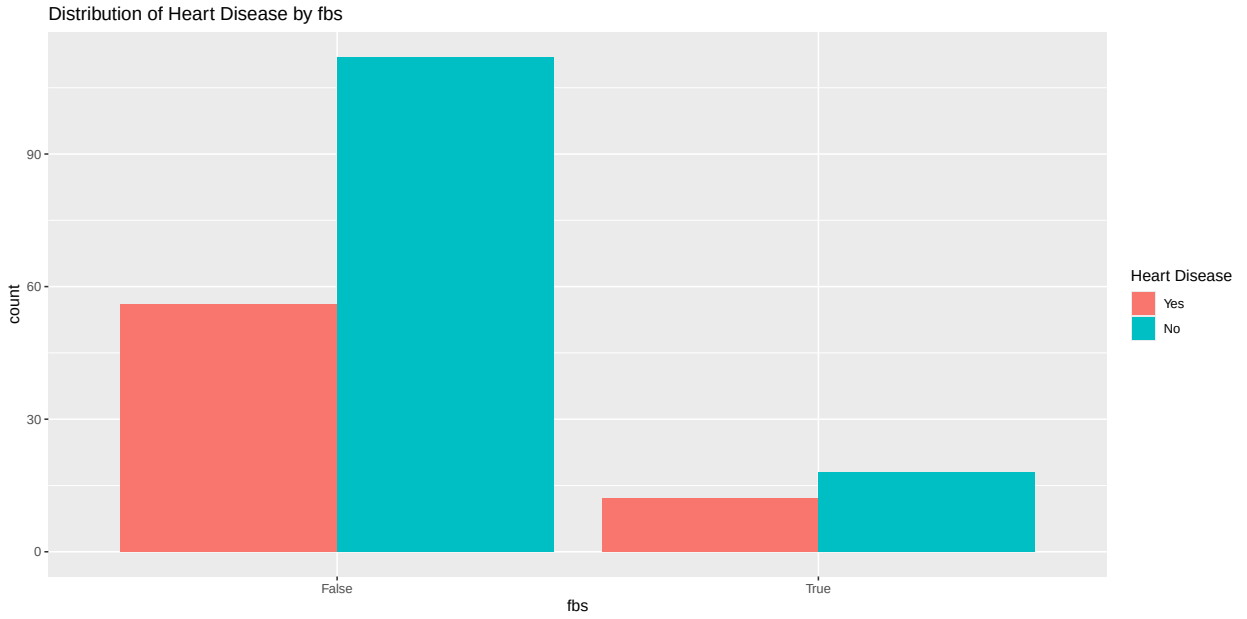
```
HeartDiseaseBar("sex")
```



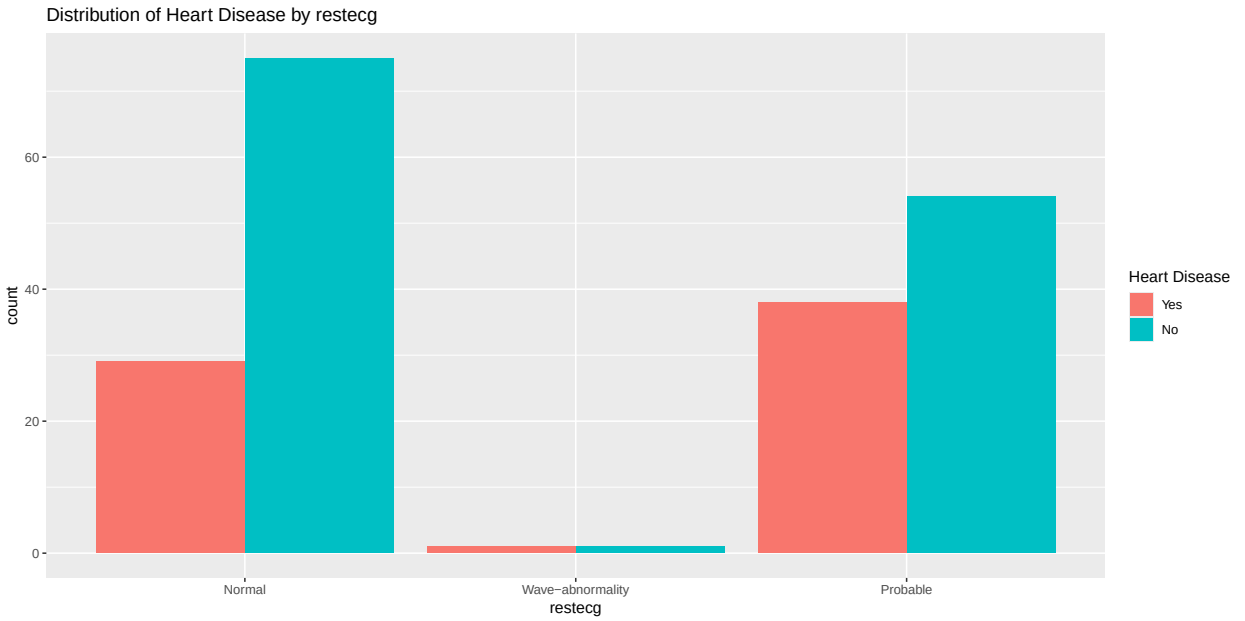

```
HeartDiseaseBar("cp")
```



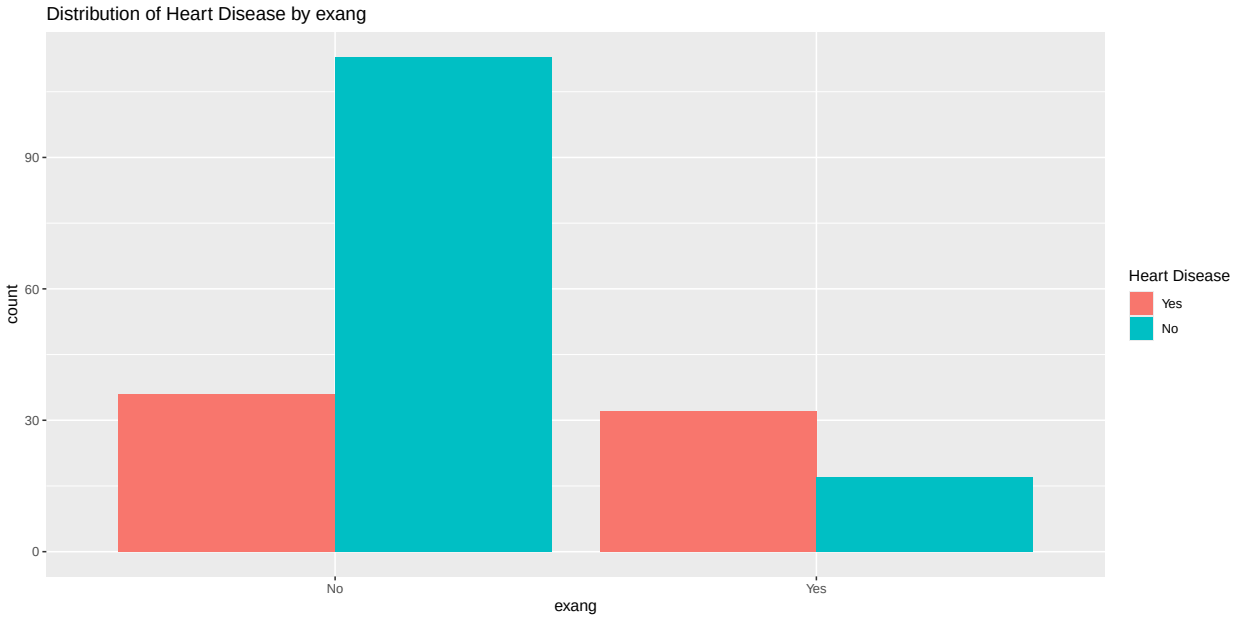

```
HeartDiseaseBar("fbs")
```



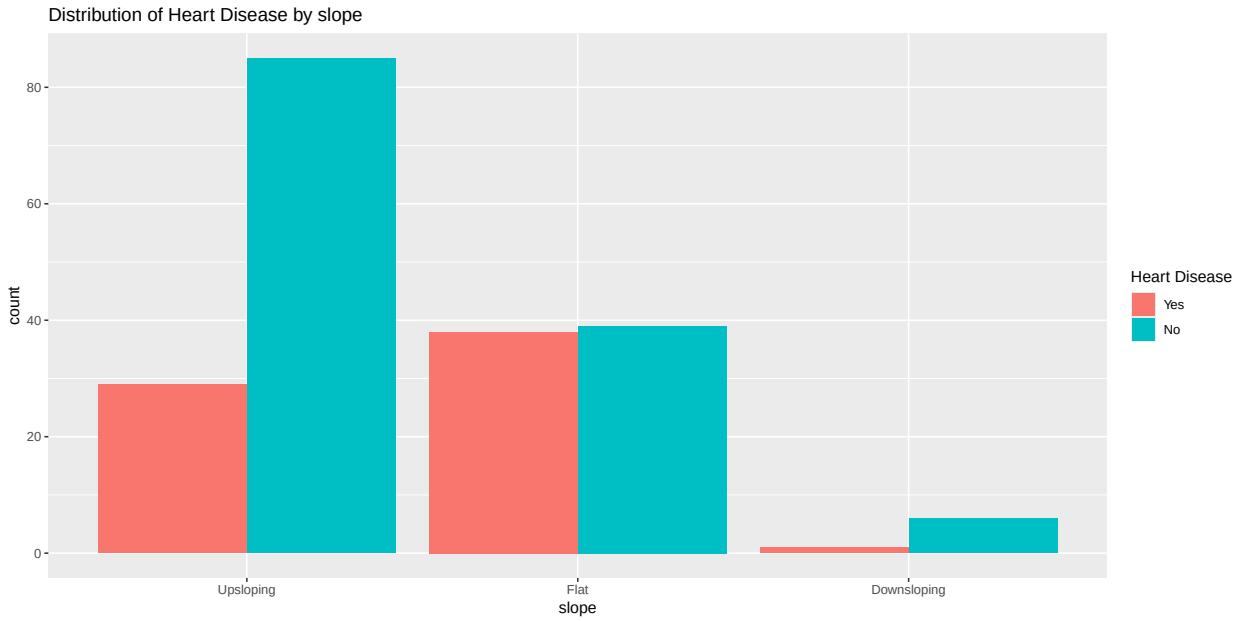

```
HeartDiseaseBar("restecg")
```



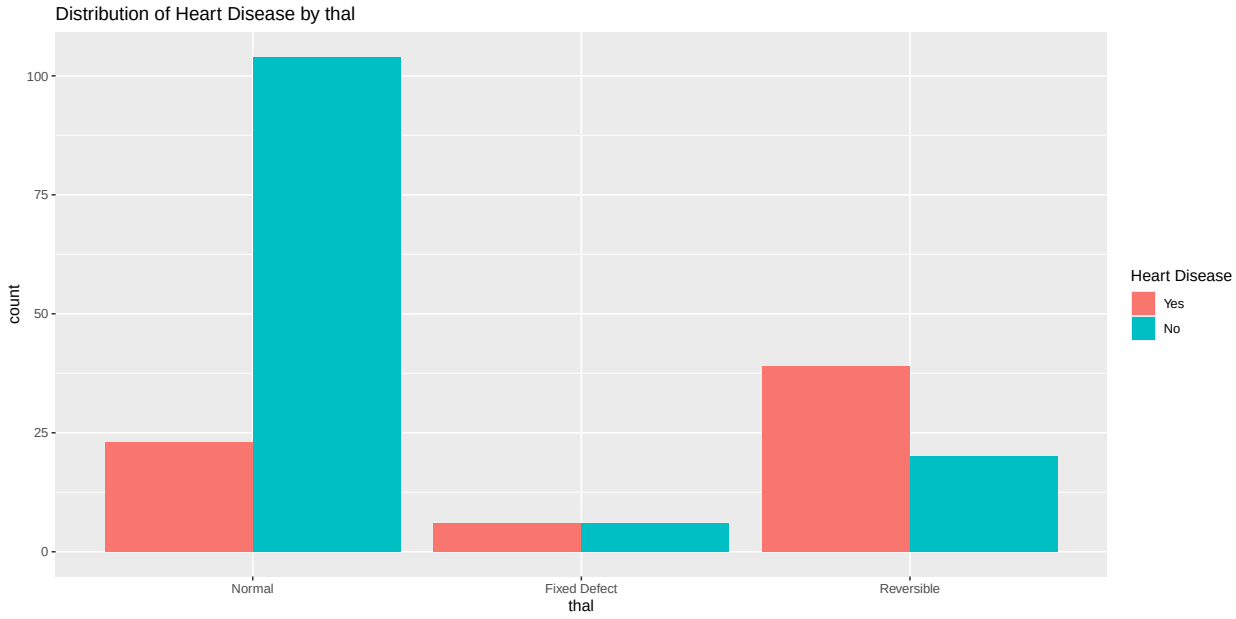

```
HeartDiseaseBar("exang")
```




```
HeartDiseaseBar("slope")
```

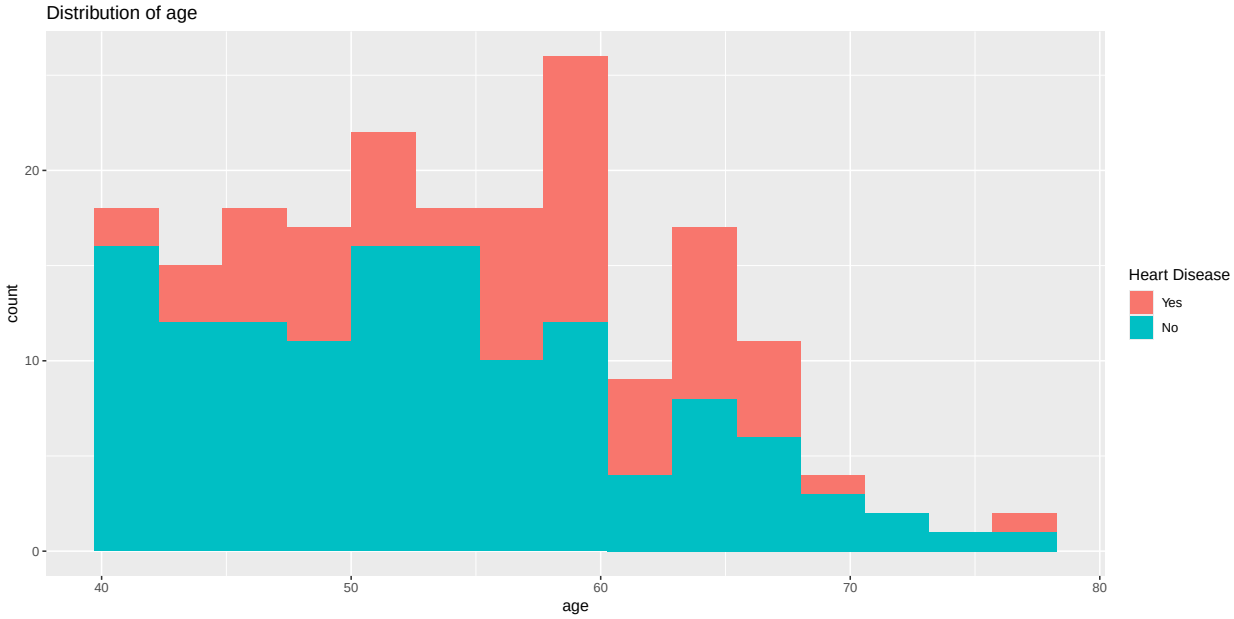



```
HeartDiseaseBar("thal")
```

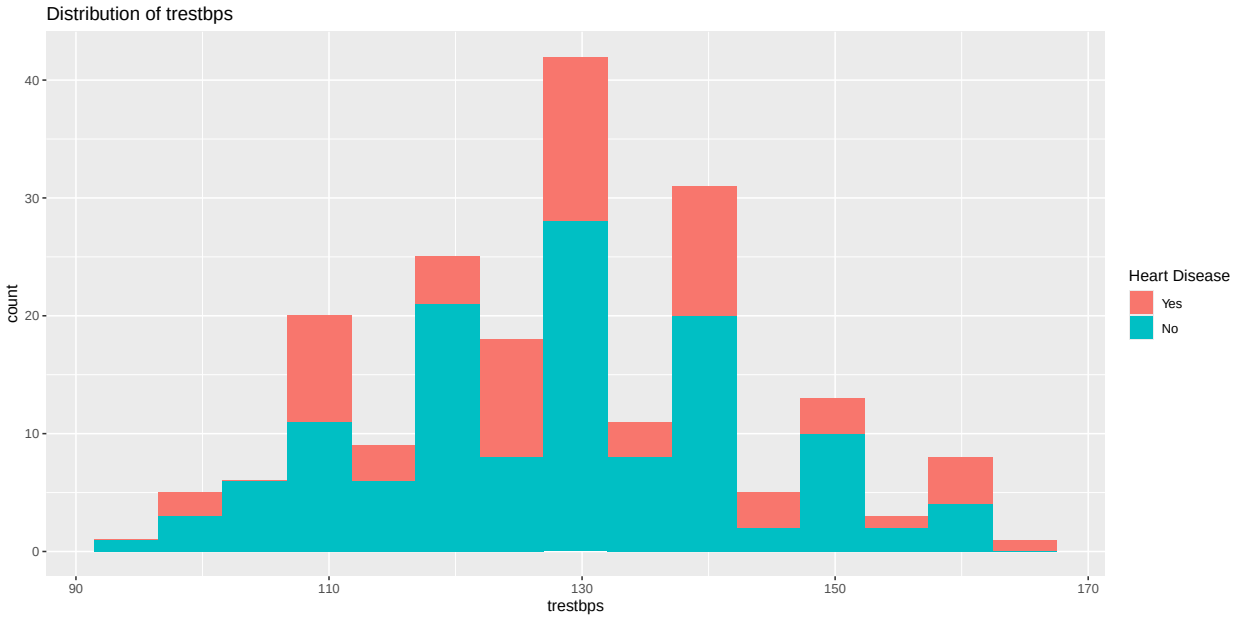


3.4.3 Histograms for Numerical Variables

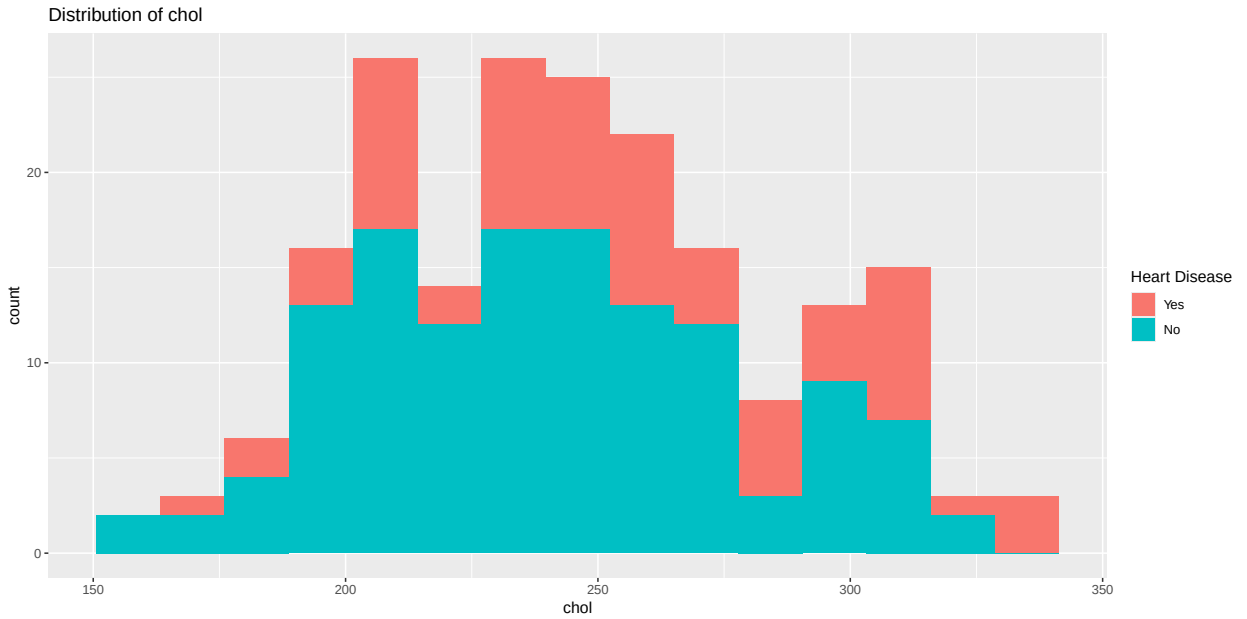
```
HeartDiseaseHist("age")
```



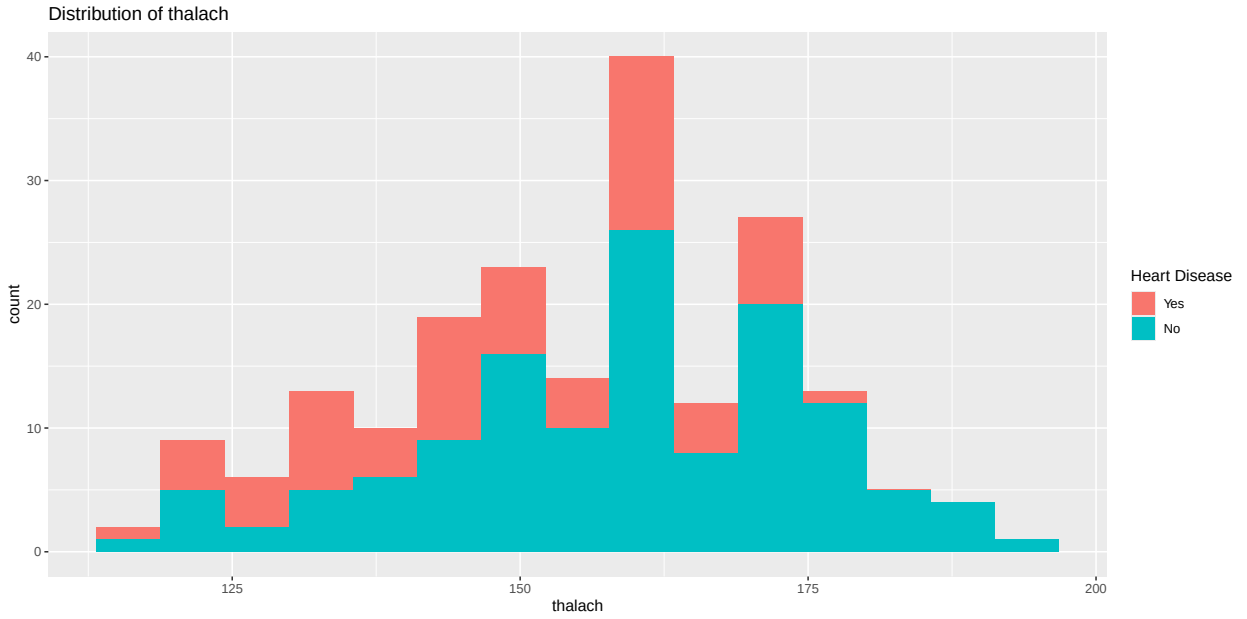

```
HeartDiseaseHist("trestbps")
```



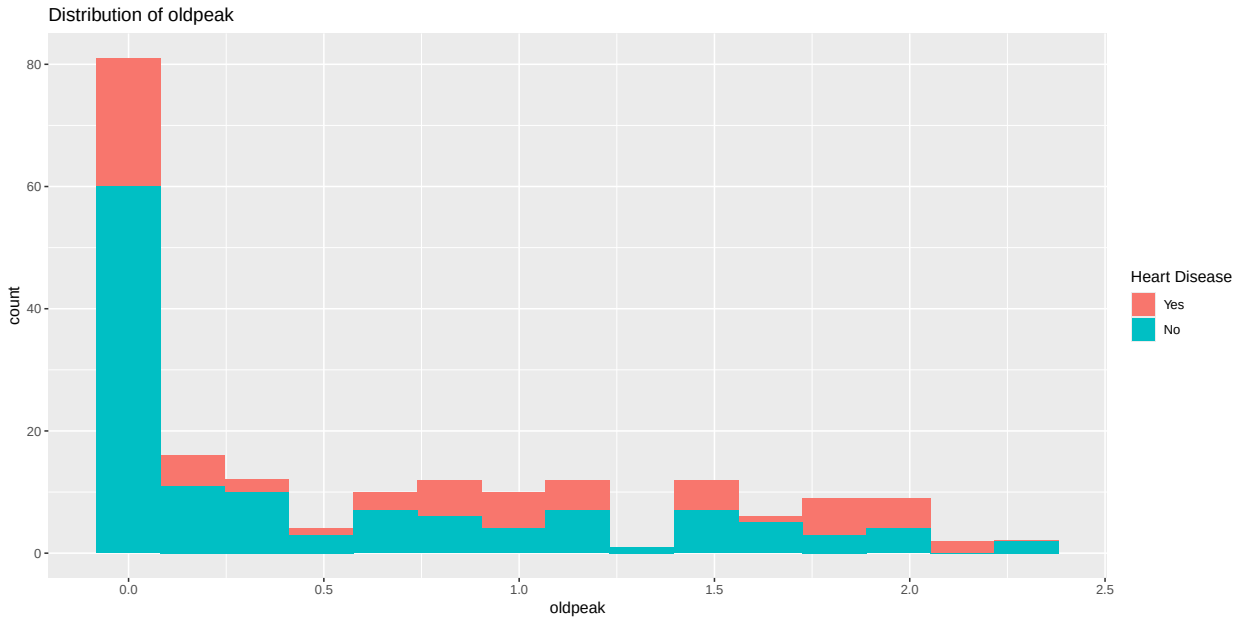

```
HeartDiseaseHist("chol")
```




```
HeartDiseaseHist("thalach")
```

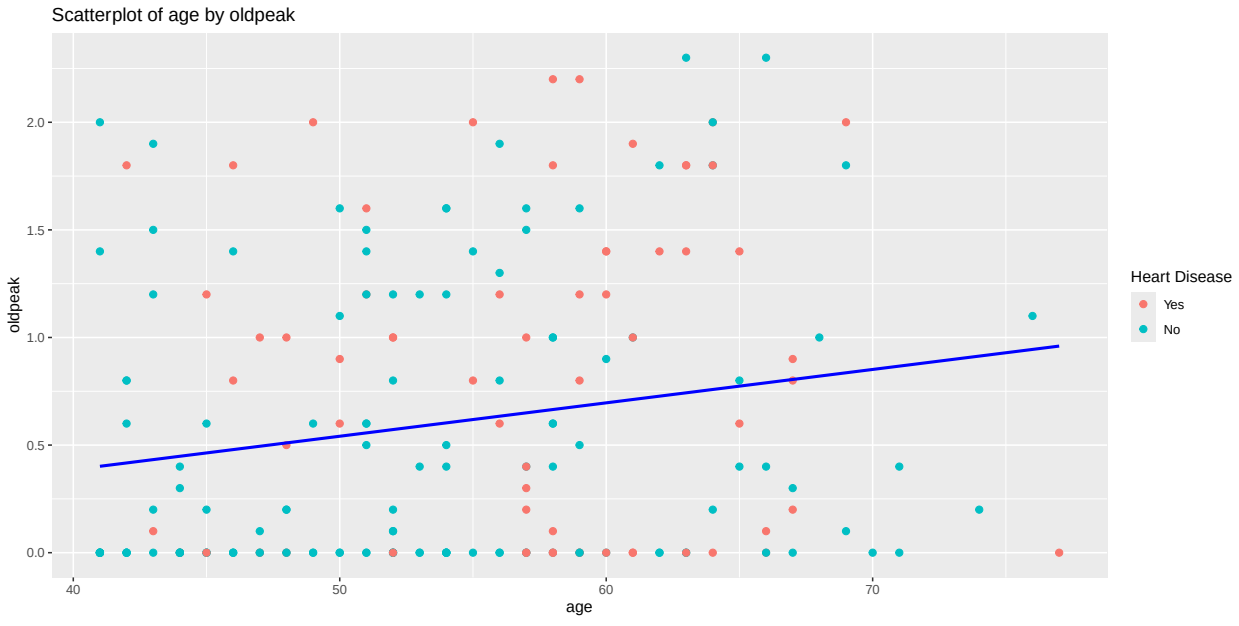



```
HeartDiseaseHist("oldpeak")
```

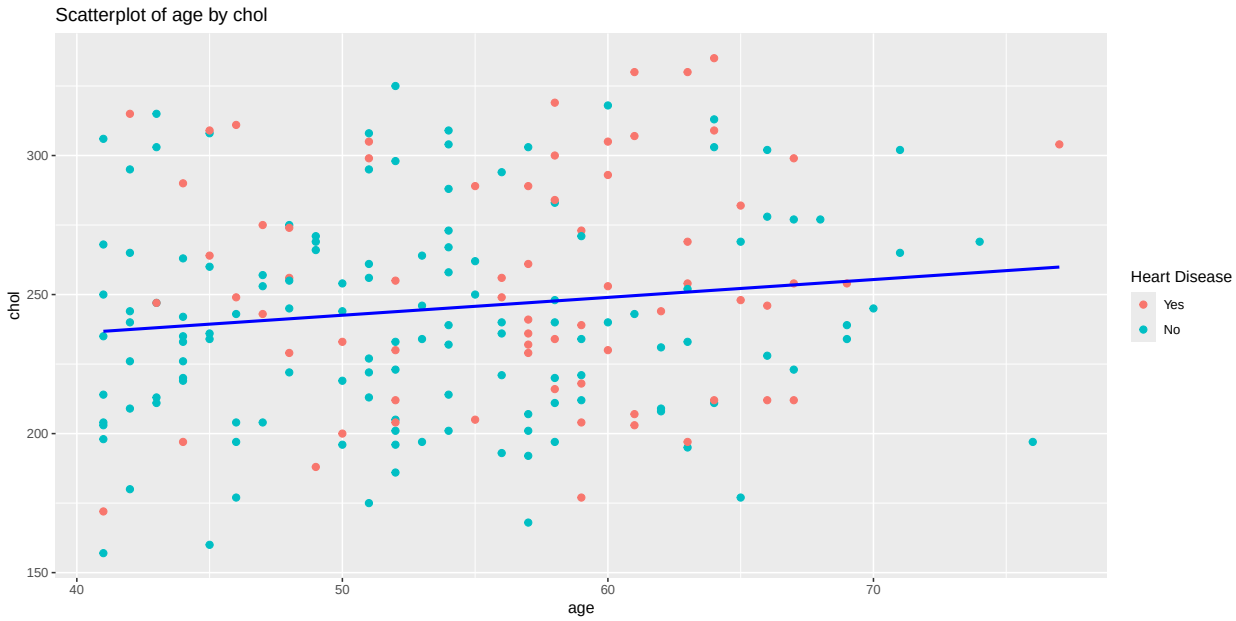


3.4.4 Scatterplots for numerical variables

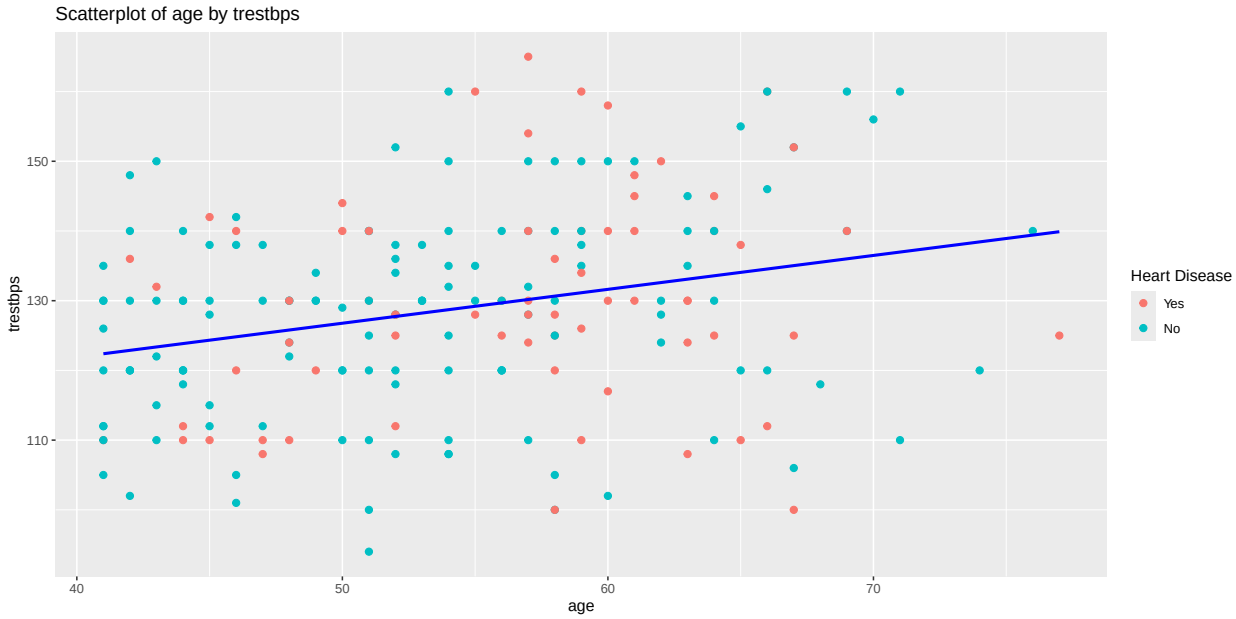
```
HeartDiseaseScatter("age", "oldpeak")
```



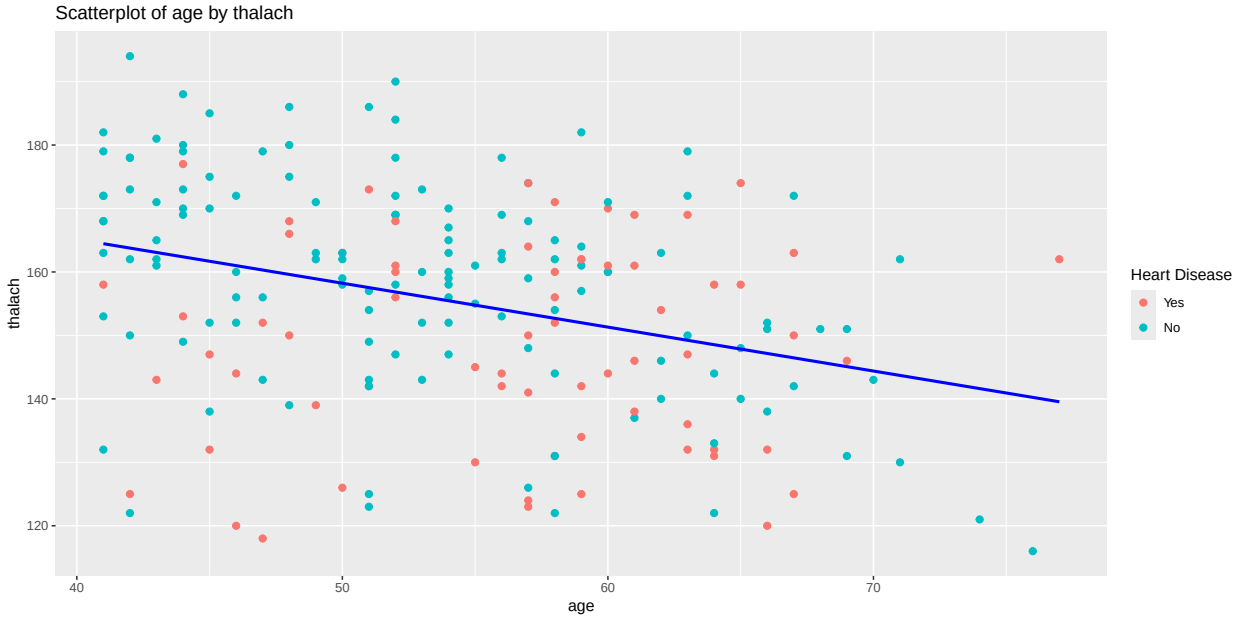

```
HeartDiseaseScatter("age", "chol")
```



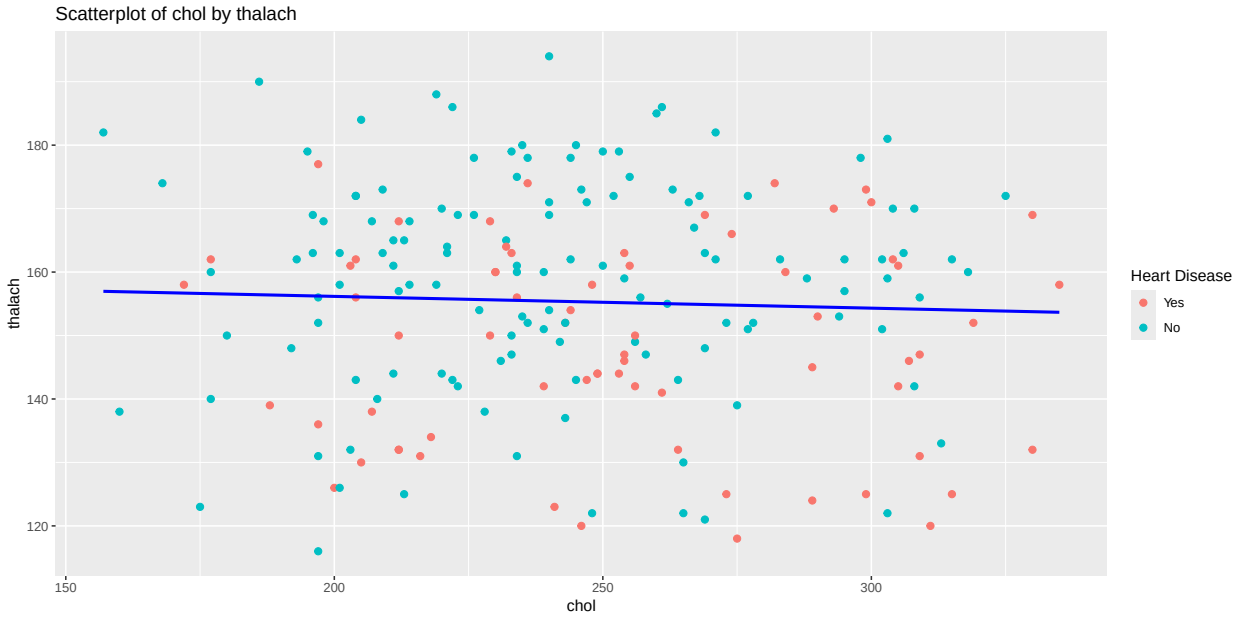

```
HeartDiseaseScatter("age", "trestbps")
```



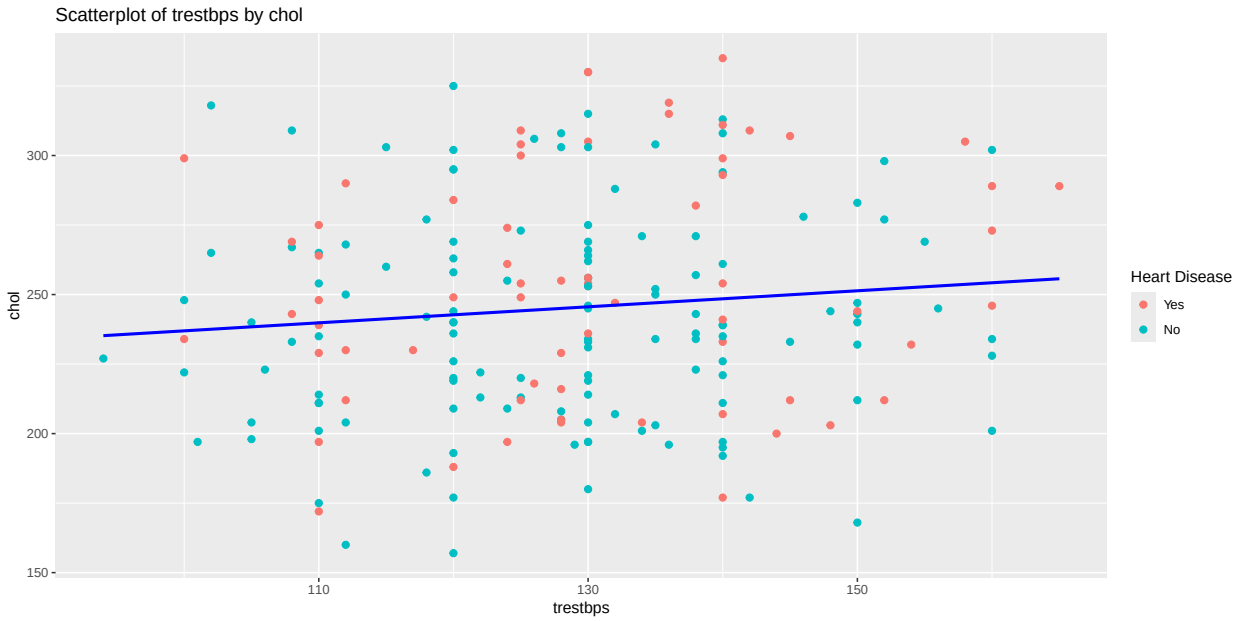

```
HeartDiseaseScatter("age", "thalach")
```



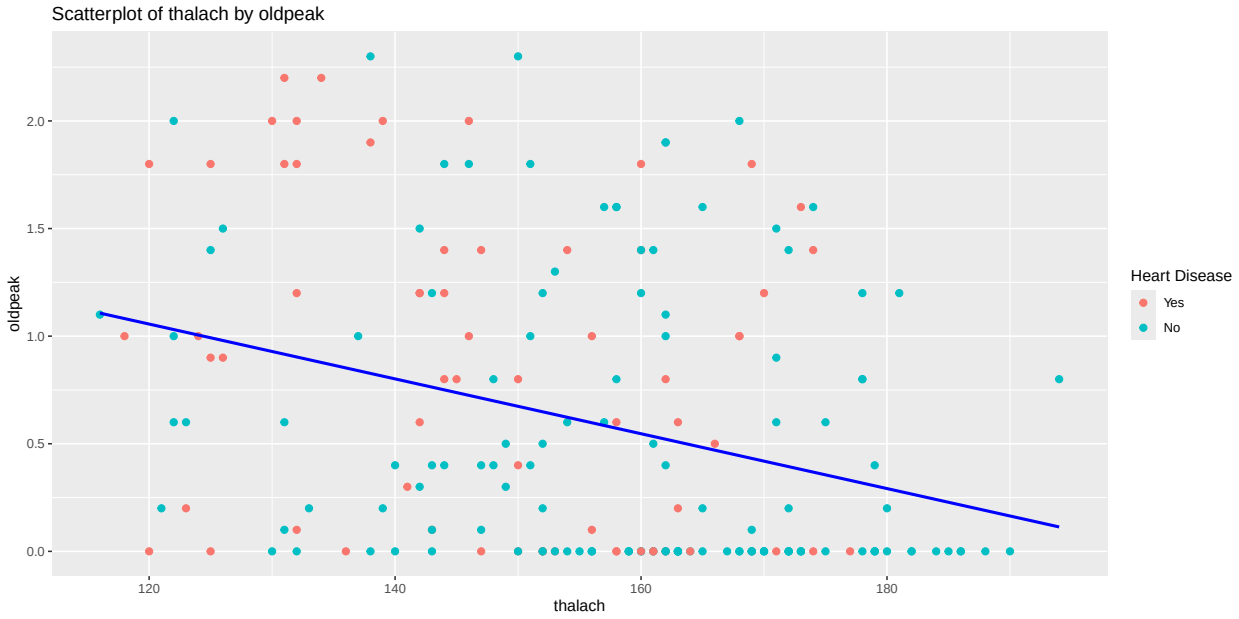

```
HeartDiseaseScatter("chol", "thalach")
```




```
HeartDiseaseScatter("trestbps", "chol")
```




```
HeartDiseaseScatter("thalach", "oldpeak")
```



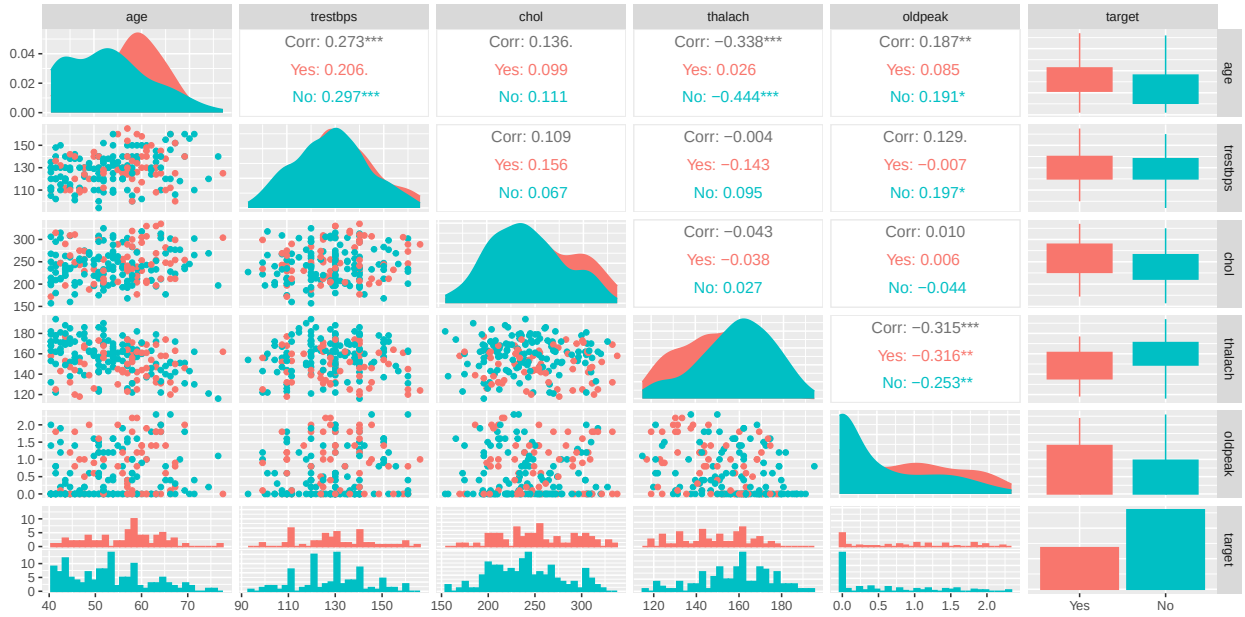
3.4.5 Pairwise correlation plots

Pairwise correlation plot for numerical variables

```
ggpairs(Heart.df[, c("age", "trestbps", "chol",
                     "thalach", "oldpeak", "target")],
        aes(color = target, fill = target))
```

```
## `stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```

```
## `stat_bin()` using `bins = 30`. Pick better value `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```



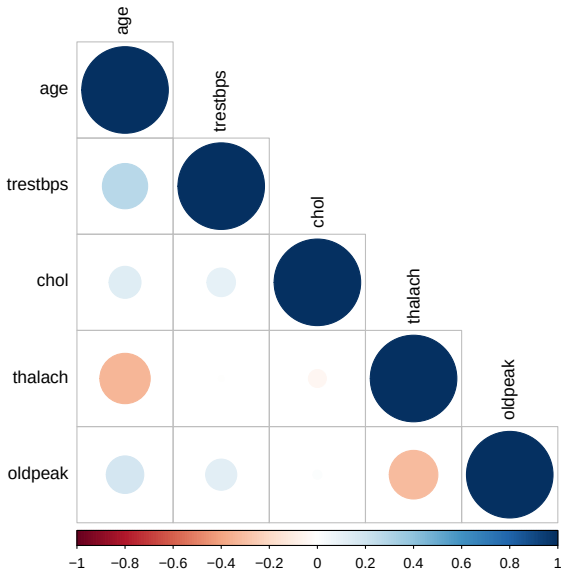
3.4.6 Correlation matrix

Correlation matrix for numerical variables

```r
# Selecting only continuous variables
continuous_vars <- c("age", "trestbps", "chol", "thalach", "oldpeak")
continuous_data <- Heart.df %>% select(all_of(continuous_vars))

# Calculating correlation matrix
correlation_matrix <- cor(continuous_data)

# Plotting the correlation matrix
corrplot(correlation_matrix, method = "circle",
         type = "lower", tl.col = "black")
```



3.4.7 Class imbalance

```
Heart.df %>% count(target) %>% mutate(pct = n/sum(n))
```

```
##   target   n       pct
## 1    Yes  68 0.3434343
## 2     No 130 0.6565657
```

3.4.8 Split & Resampling

```
set.seed(123)
split <- initial_split(Heart.df, prop = 0.8, strata = target)
train <- training(split);  test <- testing(split)

set.seed(123)
folds <- vfold_cv(train, v = 10, strata = target)
```

3.4.9 Feature engineering

```
base_recipe <-
  recipe(target ~ ., data = train) %>%
  # Optional domain transforms
  step_mutate(
    age_band = cut(age, breaks = c(0,40,50,60,70,100),
                   labels = c("<40","40-49","50-59","60-69","70+"))
  ) %>%
```

```
  step_rm(age_band) %>%  # keep linear age first; reintroduce if helpful
  # Handle rare levels (robust to small data)
  step_other(all_nominal_predictors(), threshold = 0.02) %>%
  # Impute
  step_impute_median(all_numeric_predictors()) %>%
  step_impute_mode(all_nominal_predictors()) %>%
  # One-hot encode
  step_dummy(all_nominal_predictors()) %>%
  # Standardize numeric
  step_zv(all_predictors()) %>%
  step_normalize(all_numeric_predictors())
```

4 Modeling

4.1 Logistic Regression

```
logit_spec <-
  logistic_reg(penalty = tune(), mixture = 1) %>%  # LASSO
  set_engine("glmnet")
```

4.2 Random Forest

```
rf_spec <-
  rand_forest(mtry = tune(), min_n = tune(), trees = 1000) %>%
  set_engine("ranger", importance = "permutation", num.threads = parallel::detectCores()) %>%
  set_mode("classification")
```

4.3 XGBoost

```
xgb_spec <-
  boost_tree(
    trees = 1000, tree_depth = tune(), learn_rate = tune(),
    mtry  = tune(), loss_reduction = tune(), min_n = tune()
  ) %>%
  set_engine("xgboost") %>%
  set_mode("classification")
```

4.4 Workflows

```
logit_wf <- workflow() %>% add_model(logit_spec) %>% add_recipe(base_recipe)
rf_wf    <- workflow() %>% add_model(rf_spec)    %>% add_recipe(base_recipe)
xgb_wf   <- workflow() %>% add_model(xgb_spec)   %>% add_recipe(base_recipe)
```

4.4.1 Workflow set for parallel tuning

```
wf_set <- workflow_set(
  preproc = list(baseline = base_recipe),
  models  = list(logit = logit_spec, rf = rf_spec, xgb = xgb_spec),
  cross   = FALSE
)
```

4.4.2 Tuning grids

```
logit_grid <- grid_regular(penalty(range = c(-5, 0)), levels = 20) %>% # 1e-5..1
  mutate(penalty = 10^penalty)
rf_grid    <- grid_latin_hypercube(mtry(range = c(2L, 10L)), min_n(), size = 20)
```

```
## Warning: `grid_latin_hypercube()` was deprecated in dials 1.3.0.
## i Please use `grid_space_filling()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
xgb_grid   <- grid_latin_hypercube(
  tree_depth(), learn_rate(range = c(-3, -0.5)), mtry(range = c(2L,10L)),
  loss_reduction(), min_n(), size = 30
) %>% mutate(learn_rate = 10^learn_rate)

ctrl <- control_grid(save_pred = TRUE, parallel_over = "resamples")
```

```
set.seed(2025)
logit_res <- tune_grid(logit_wf, resamples = folds, grid = logit_grid,
                       metrics = metric_set(roc_auc, pr_auc, brier_class),
                       control = ctrl)
```

```
set.seed(2025)
rf_res    <- tune_grid(rf_wf,    resamples = folds, grid = rf_grid,
                       metrics = metric_set(roc_auc, pr_auc, brier_class),
                       control = ctrl)
```

4.4.3 Model selection

5 Evaluation

6 Deployment

7 Conclusion

8 Session Information

```
sessionInfo()
```

```
## R version 4.5.1 (2025-06-13 ucrt)
## Platform: x86_64-w64-mingw32/x64
## Running under: Windows 11 x64 (build 26100)
##
## Matrix products: default
##   LAPACK version 3.12.1
##
## locale:
## [1] LC_COLLATE=English_United States.utf8
## [2] LC_CTYPE=English_United States.utf8
## [3] LC_MONETARY=English_United States.utf8
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.utf8
##
## time zone: America/Chicago
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] pROC_1.19.0.1        corrplot_0.95        rpart_4.1.24
##  [4] e1071_1.7-16         discrim_1.0.2        probably_1.2.0
##  [7] finetune_1.2.1       GGally_2.4.0         janitor_2.2.1
## [10] themis_1.0.3         vip_0.4.1            yardstick_1.3.2
## [13] workflowsets_1.1.1   workflows_1.3.0      tune_2.0.1
## [16] tailor_0.1.0         rsample_1.3.1        recipes_1.3.1
## [19] parsnip_1.3.3        modeldata_1.5.1      infer_1.0.9
## [22] dials_1.4.2          scales_1.4.0         broom_1.0.10
## [25] tidymodels_1.4.1     formatR_1.14         randomForest_4.7-1.2
## [28] caret_7.0-1          lattice_0.22-7       RCurl_1.98-1.17
## [31] lubridate_1.9.4      forcats_1.0.1        stringr_1.5.2
## [34] dplyr_1.1.4          purrr_1.1.0          readr_2.1.5
## [37] tidyr_1.3.1          tibble_3.3.0         ggplot2_4.0.0
## [40] tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##  [1] bitops_1.0-9         rlang_1.1.6          magrittr_2.0.4
##  [4] snakecase_0.11.1     furrr_0.3.1          compiler_4.5.1
##  [7] mgcv_1.9-3           vctrs_0.6.5          reshape2_1.4.4
## [10] lhs_1.2.0            shape_1.4.6.1        crayon_1.5.3
## [13] pkgconfig_2.0.3      fastmap_1.2.0        backports_1.5.0
## [16] labeling_0.4.3       rmarkdown_2.30       prodlim_2025.04.28
## [19] tzdb_0.5.0           glmnet_4.1-10        xfun_0.53
## [22] parallel_4.5.1       R6_2.6.1             stringi_1.8.7
## [25] RColorBrewer_1.1-3   ranger_0.17.0        parallelly_1.45.1
## [28] Rcpp_1.1.0           iterators_1.0.14     knitr_1.50
## [31] future.apply_1.20.0  Matrix_1.7-4         splines_4.5.1
```

```
## [34] nnet_7.3-20          timechange_0.3.0     tidyselect_1.2.1
## [37] rstudioapi_0.17.1    yaml_2.3.10          timeDate_4051.111
## [40] codetools_0.2-20     listenv_0.9.1        plyr_1.8.9
## [43] withr_3.0.2          S7_0.2.0             evaluate_1.0.5
## [46] future_1.67.0        survival_3.8-3       proxy_0.4-27
## [49] ggstats_0.11.0       pillar_1.11.1        foreach_1.5.2
## [52] stats4_4.5.1         generics_0.1.4       hms_1.1.4
## [55] globals_0.18.0       class_7.3-23         glue_1.8.0
## [58] ROSE_0.0-4           tools_4.5.1          data.table_1.17.8
## [61] ModelMetrics_1.2.2.2 gower_1.0.2          grid_4.5.1
## [64] ipred_0.9-15         nlme_3.1-168         cli_3.6.5
## [67] DiceDesign_1.10      lava_1.8.1           gtable_0.3.6
## [70] GPfit_1.0-9          digest_0.6.37        farver_2.1.2
## [73] htmltools_0.5.8.1    lifecycle_1.0.4      hardhat_1.4.2
## [76] sparsevctrs_0.3.4    MASS_7.3-65
```