vpfr75_summative_classification

raj-thakkar

1. Clinical Data Understanding and Preprocessing

1.1 In-depth Clinical EDA

#Loading libraries

```r
# Loading libraries
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.4.3
```

```
## Warning: package 'tidyr' was built under R version 4.4.3
```

```
## Warning: package 'readr' was built under R version 4.4.3
```

```
## Warning: package 'dplyr' was built under R version 4.4.3
```

```
## Warning: package 'forcats' was built under R version 4.4.3
```

```
## Warning: package 'lubridate' was built under R version 4.4.3
```

```
## -- 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 1.9.4     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 error
```

```r
library(dplyr)
library(ggplot2)
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
##
## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
library(corrplot)
```

```
## Warning: package 'corrplot' was built under R version 4.4.3
```

```
## corrplot 0.95 loaded
```

```r
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.4.3
```

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

```r
library(Hmisc)
```

```
## Warning: package 'Hmisc' was built under R version 4.4.3
```

```
##
## Attaching package: 'Hmisc'
##
## The following objects are masked from 'package:dplyr':
##
##     src, summarize
##
## The following objects are masked from 'package:base':
##
##     format.pval, units
```

```r
library(pROC)
```

```
## Warning: package 'pROC' was built under R version 4.4.3
```

```
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
library(knitr)
library(RColorBrewer)

library(glmnet)
```

```
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
##
## Loaded glmnet 4.1-8
```

```r
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.4.3
```

```
## randomForest 4.7-1.2
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:gridExtra':
##
##     combine
##
## The following object is masked from 'package:dplyr':
##
##     combine
##
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(xgboost)
```

```
## Warning: package 'xgboost' was built under R version 4.4.3

##
## Attaching package: 'xgboost'
##
## The following object is masked from 'package:dplyr':
##
##     slice
```

```
library(rpart)
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 4.4.3
```

```
library(ROCR)
```

```
## Warning: package 'ROCR' was built under R version 4.4.3
```

```
# Set a seed for reproducibility
set.seed(200)

# 1.1 In-depth Clinical EDA

# Read the dataset
heart_data <- read.csv("C:/Users/rajth/Desktop/MISCADA/Classification summative/heart_failure.csv", stri

# Display basic information
cat("Dataset dimensions:", dim(heart_data)[1], "rows and",
    dim(heart_data)[2], "columns\n\n")
```

```
## Dataset dimensions: 299 rows and 13 columns
```

```
# Examine the structure of the dataset
str(heart_data)
```

```
## 'data.frame':    299 obs. of  13 variables:
##  $ age                     : num  75 55 65 50 65 90 75 60 65 80 ...
##  $ anaemia                 : int  0 0 0 1 1 1 1 1 0 1 ...
##  $ creatinine_phosphokinase: int  582 7861 146 111 160 47 246 315 157 123 ...
##  $ diabetes                : int  0 0 0 0 1 0 0 1 0 0 ...
##  $ ejection_fraction       : int  20 38 20 20 20 40 15 60 65 35 ...
##  $ high_blood_pressure     : int  1 0 0 0 0 1 0 0 0 1 ...
##  $ platelets               : num  265000 263358 162000 210000 327000 ...
##  $ serum_creatinine        : num  1.9 1.1 1.3 1.9 2.7 2.1 1.2 1.1 1.5 9.4 ...
##  $ serum_sodium            : int  130 136 129 137 116 132 137 131 138 133 ...
##  $ sex                     : int  1 1 1 1 0 1 1 1 0 1 ...
##  $ smoking                 : int  0 0 1 0 0 1 0 1 0 1 ...
##  $ time                    : int  4 6 7 7 8 8 10 10 10 10 ...
##  $ fatal_mi                : int  1 1 1 1 1 1 1 1 1 1 ...
```

```r
# Initial exploration of the dataset
print("First few rows of the dataset:")
```

```
## [1] "First few rows of the dataset:"
```

```r
head(heart_data)
```

```
##   age anaemia creatinine_phosphokinase diabetes ejection_fraction
## 1  75       0                      582        0                20
## 2  55       0                     7861        0                38
## 3  65       0                      146        0                20
## 4  50       1                      111        0                20
## 5  65       1                      160        1                20
## 6  90       1                       47        0                40
##   high_blood_pressure platelets serum_creatinine serum_sodium sex smoking time
## 1                   1    265000              1.9          130   1       0    4
## 2                   0    263358              1.1          136   1       0    6
## 3                   0    162000              1.3          129   1       1    7
## 4                   0    210000              1.9          137   1       0    7
## 5                   0    327000              2.7          116   0       0    8
## 6                   1    204000              2.1          132   1       1    8
##   fatal_mi
## 1        1
## 2        1
## 3        1
## 4        1
## 5        1
## 6        1
```

```r
colnames(heart_data)
```

```
##  [1] "age"                      "anaemia"
##  [3] "creatinine_phosphokinase" "diabetes"
##  [5] "ejection_fraction"        "high_blood_pressure"
##  [7] "platelets"                "serum_creatinine"
##  [9] "serum_sodium"             "sex"
## [11] "smoking"                  "time"
## [13] "fatal_mi"
```

#Summary of heart failure dataset

```r
# Summary statistics of all variables
summary(heart_data)
```

```
##       age           anaemia       creatinine_phosphokinase    diabetes
##  Min.   :40.00   Min.   :0.0000   Min.   :  23.0           Min.   :0.0000
##  1st Qu.:51.00   1st Qu.:0.0000   1st Qu.: 116.5           1st Qu.:0.0000
##  Median :60.00   Median :0.0000   Median : 250.0           Median :0.0000
##  Mean   :60.83   Mean   :0.4314   Mean   : 581.8           Mean   :0.4181
##  3rd Qu.:70.00   3rd Qu.:1.0000   3rd Qu.: 582.0           3rd Qu.:1.0000
##  Max.   :95.00   Max.   :1.0000   Max.   :7861.0           Max.   :1.0000
##  ejection_fraction high_blood_pressure   platelets      serum_creatinine
##  Min.   :14.00     Min.   :0.0000      Min.   : 25100   Min.   :0.500
##  1st Qu.:30.00     1st Qu.:0.0000      1st Qu.:212500   1st Qu.:0.900
##  Median :38.00     Median :0.0000      Median :262000   Median :1.100
##  Mean   :38.08     Mean   :0.3512      Mean   :263358   Mean   :1.394
##  3rd Qu.:45.00     3rd Qu.:1.0000      3rd Qu.:303500   3rd Qu.:1.400
##  Max.   :80.00     Max.   :1.0000      Max.   :850000   Max.   :9.400
##   serum_sodium        sex             smoking             time
##  Min.   :113.0   Min.   :0.0000   Min.   :0.0000   Min.   :  4.0
##  1st Qu.:134.0   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 73.0
##  Median :137.0   Median :1.0000   Median :0.0000   Median :115.0
##  Mean   :136.6   Mean   :0.6488   Mean   :0.3211   Mean   :130.3
##  3rd Qu.:140.0   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:203.0
##  Max.   :148.0   Max.   :1.0000   Max.   :1.0000   Max.   :285.0
##     fatal_mi
##  Min.   :0.0000
##  1st Qu.:0.0000
##  Median :0.0000
##  Mean   :0.3211
##  3rd Qu.:1.0000
##  Max.   :1.0000
```

```r
# Convert binary variables to factors
heart_data <- heart_data %>%
  mutate(
    anaemia = factor(anaemia, levels = c(0, 1),
                     labels = c("No", "Yes")),
    diabetes = factor(diabetes, levels = c(0, 1),
                      labels = c("No", "Yes")),
    high_blood_pressure = factor(high_blood_pressure, levels = c(0, 1),
                                 labels = c("No", "Yes")),
    sex = factor(sex, levels = c(0, 1),
                 labels = c("Female", "Male")),
    smoking = factor(smoking, levels = c(0, 1),
                     labels = c("No", "Yes")),
    fatal_mi = factor(fatal_mi, levels = c(0, 1),
                      labels = c("Survived", "Died"))
  )

# Check for missing values
missing_values <- colSums(is.na(heart_data))
cat("Missing values per column:\n")
```

```
## Missing values per column:
```

```r
print(missing_values)
```

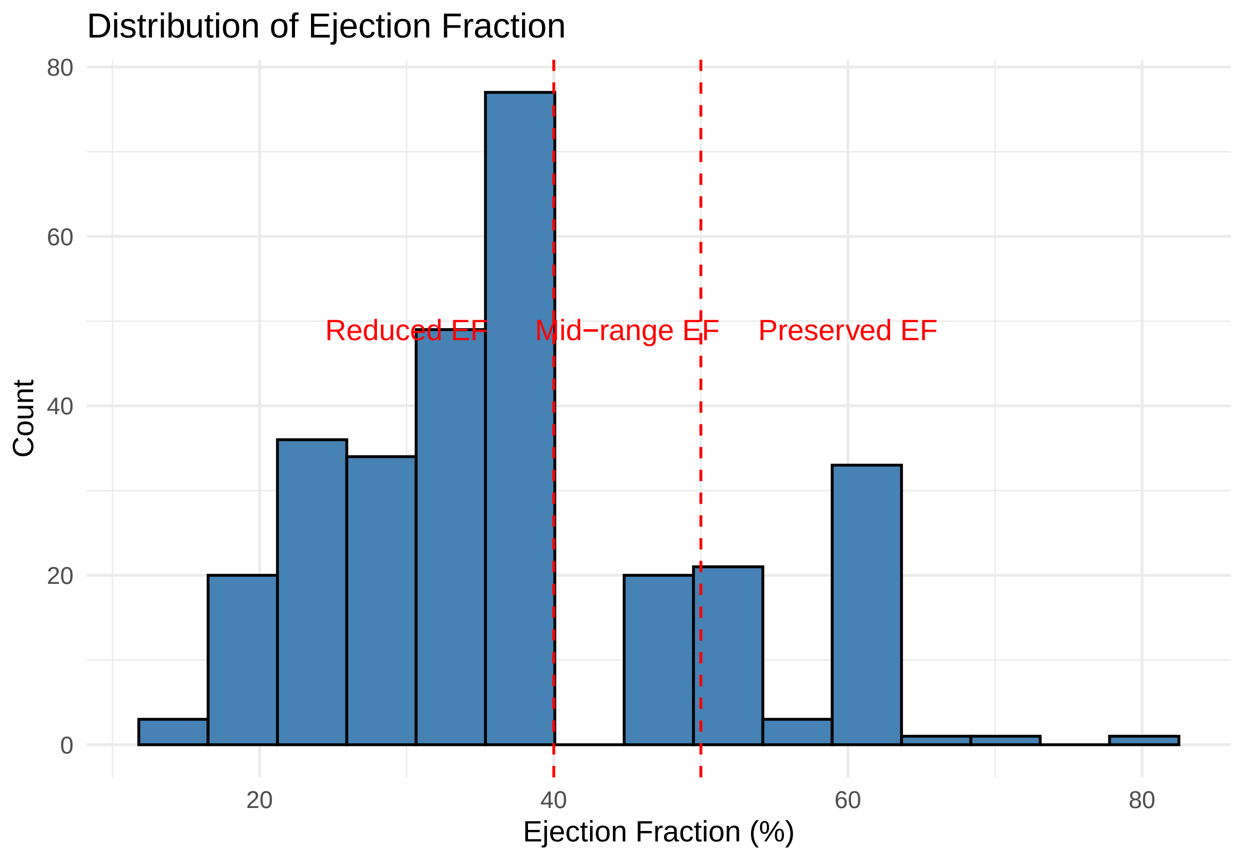
```
##                      age                  anaemia creatinine_phosphokinase
##                        0                        0                        0
##                 diabetes        ejection_fraction      high_blood_pressure
##                        0                        0                        0
##                platelets         serum_creatinine             serum_sodium
##                        0                        0                        0
##                      sex                  smoking                     time
##                        0                        0                        0
##                 fatal_mi
##                        0
```

```
# Calculate percentage of deaths in the dataset
death_rate <- 100 * sum(heart_data$fatal_mi == "Died") / nrow(heart_data)
cat("\nOverall mortality rate:", round(death_rate, 1), "%\n")
```

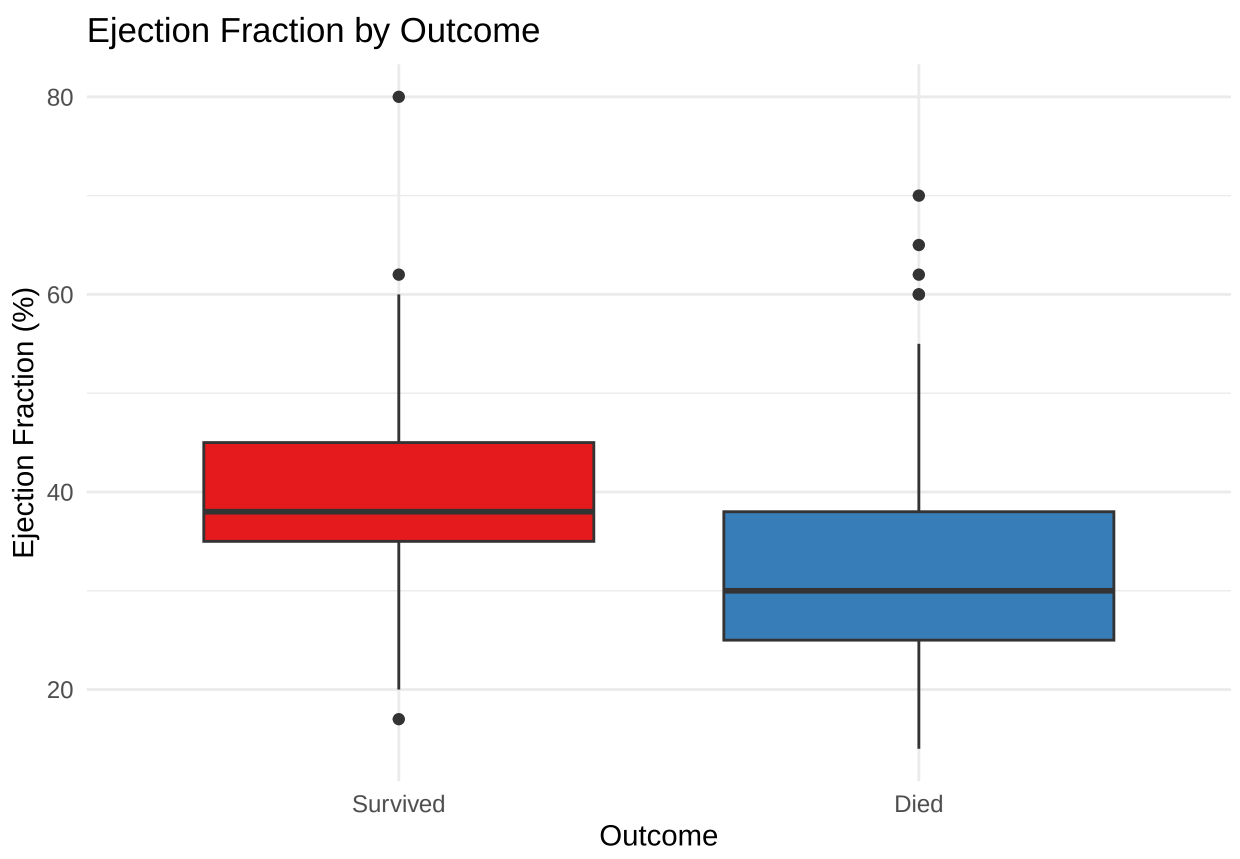
```
##
## Overall mortality rate: 32.1 %
```

a)Analyze distribution of cardiac function markers (ejection_fraction) b)Evaluate renal function indicators (serum_creatinine) c)Examine electrolyte balance (serum_sodium)

```
# 1.1.1 Analyze distribution of cardiac function markers (ejection_fraction)

# Histogram of ejection_fraction
ggplot(heart_data, aes(x = ejection_fraction)) +
  geom_histogram(bins = 15, fill = "steelblue", color = "black") +
  labs(title = "Distribution of Ejection Fraction",
       x = "Ejection Fraction (%)",
       y = "Count") +
  theme_minimal() +
  geom_vline(xintercept = c(40, 50), linetype = "dashed", color = "red") +
  annotate("text", x = 30, y = max(table(heart_data$ejection_fraction)),
           label = "Reduced EF", color = "red") +
  annotate("text", x = 45, y = max(table(heart_data$ejection_fraction)),
           label = "Mid-range EF", color = "red") +
  annotate("text", x = 60, y = max(table(heart_data$ejection_fraction)),
           label = "Preserved EF", color = "red")
```



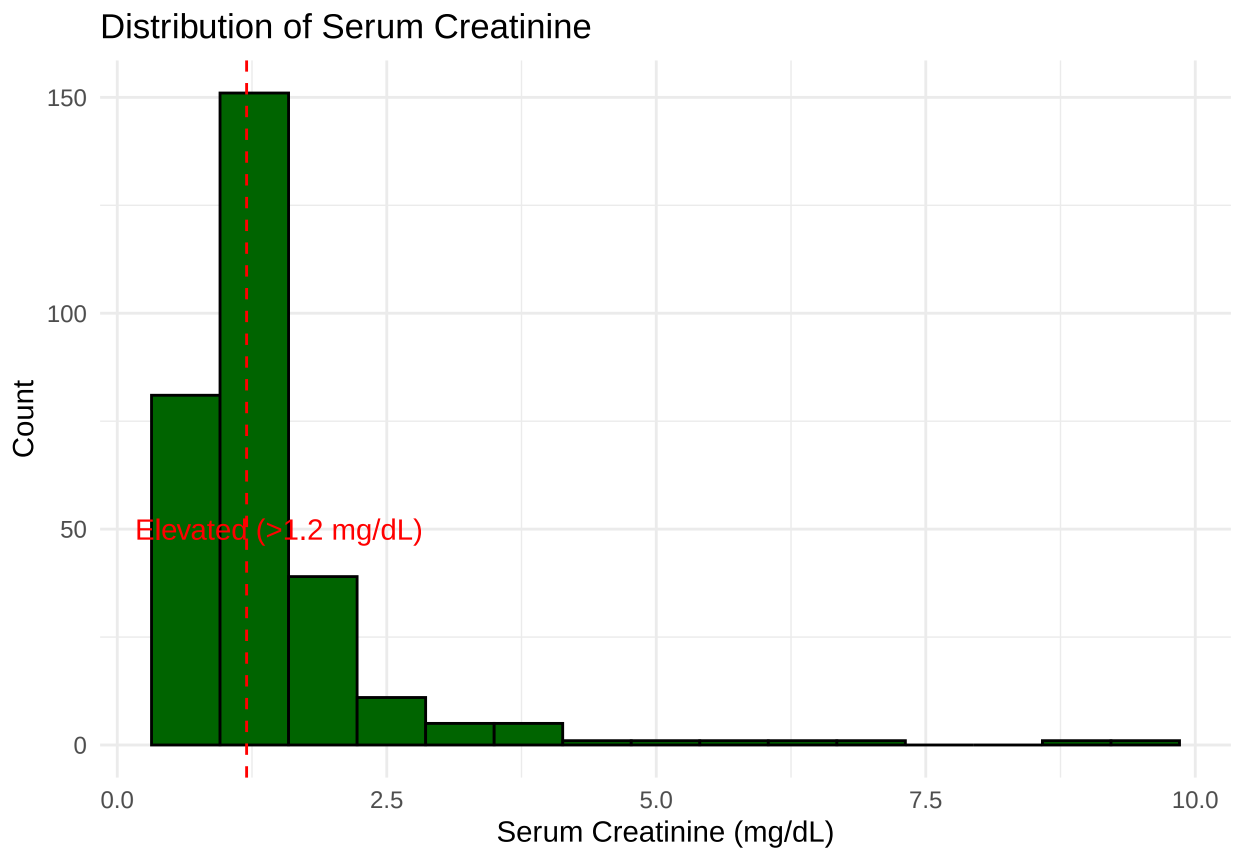
```r
# Box plot of ejection_fraction by mortality
ggplot(heart_data, aes(x = fatal_mi, y = ejection_fraction, fill = fatal_mi)) +
  geom_boxplot() +
  labs(title = "Ejection Fraction by Outcome",
       x = "Outcome",
       y = "Ejection Fraction (%)") +
  scale_fill_brewer(palette = "Set1") +
  theme_minimal() +
  theme(legend.position = "none")
```



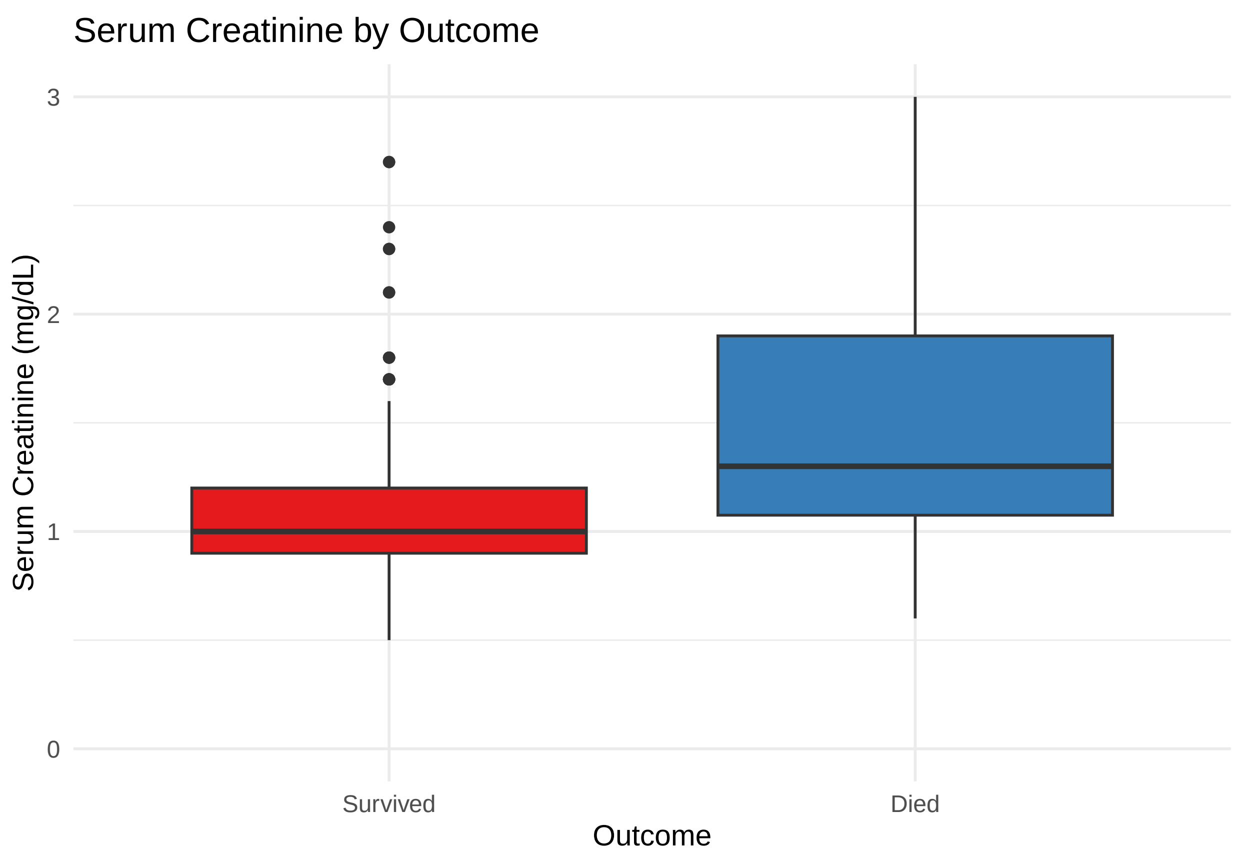
```r
# Calculate statistics for ejection fraction by outcome
ef_stats <- heart_data %>%
  group_by(fatal_mi) %>%
  dplyr::summarize(
    count = n(),
    mean_ef = mean(ejection_fraction, na.rm = TRUE),
    median_ef = median(ejection_fraction, na.rm = TRUE),
    sd_ef = sd(ejection_fraction, na.rm = TRUE),
    min_ef = min(ejection_fraction, na.rm = TRUE),
    max_ef = max(ejection_fraction, na.rm = TRUE)
  )

print(ef_stats)
```

```
## # A tibble: 2 x 7
##   fatal_mi count mean_ef median_ef sd_ef min_ef max_ef
##   <fct>    <int>   <dbl>     <dbl> <dbl>  <int>  <int>
## 1 Survived   203    40.3        38  10.9     17     80
## 2 Died        96    33.5        30  12.5     14     70
```

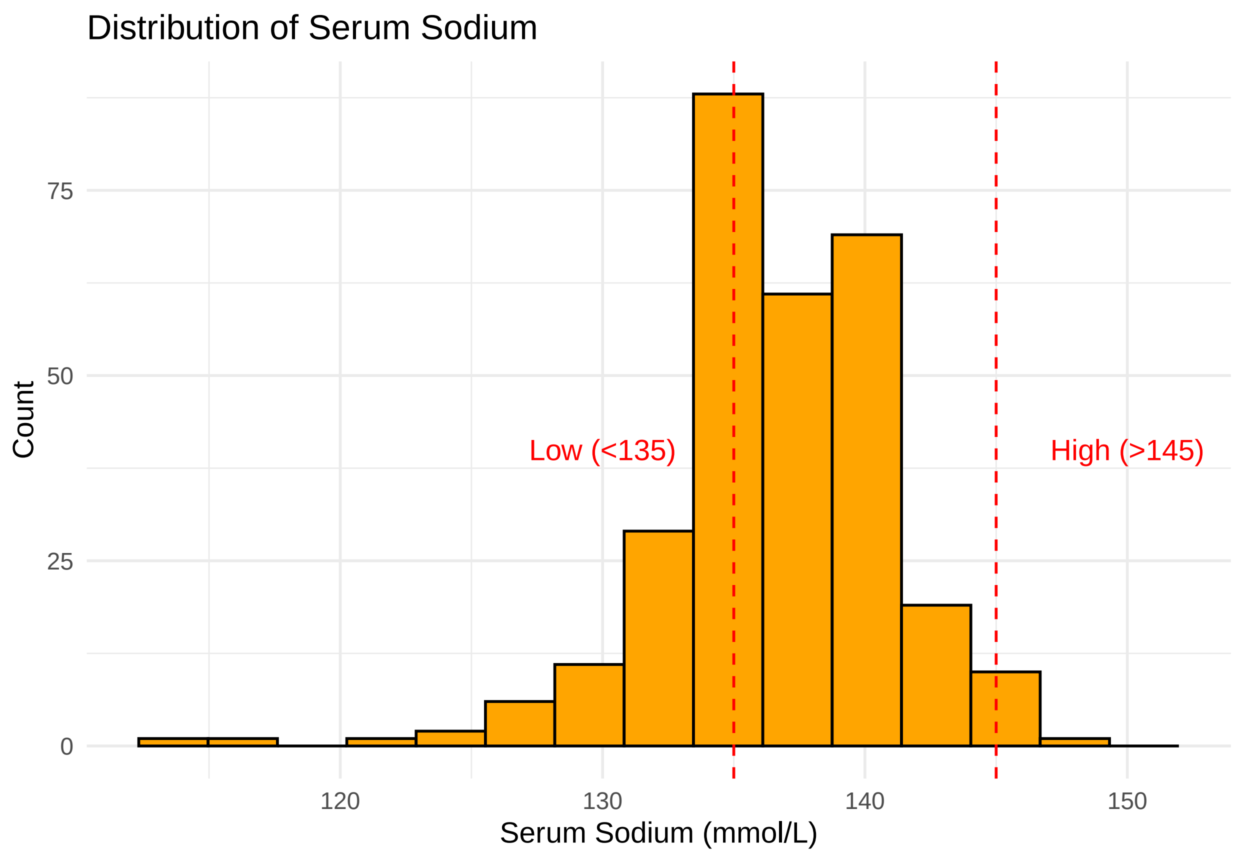
```r
# 1.1.2 Evaluate renal function indicators (serum_creatinine)

# Histogram of serum_creatinine
ggplot(heart_data, aes(x = serum_creatinine)) +
  geom_histogram(bins = 15, fill = "darkgreen", color = "black") +
  labs(title = "Distribution of Serum Creatinine",
       x = "Serum Creatinine (mg/dL)",
       y = "Count") +
  theme_minimal() +
  geom_vline(xintercept = 1.2, linetype = "dashed", color = "red") +
  annotate("text", x = 1.5, y = max(table(round(heart_data$serum_creatinine, 1))),
           label = "Elevated (>1.2 mg/dL)", color = "red")
```



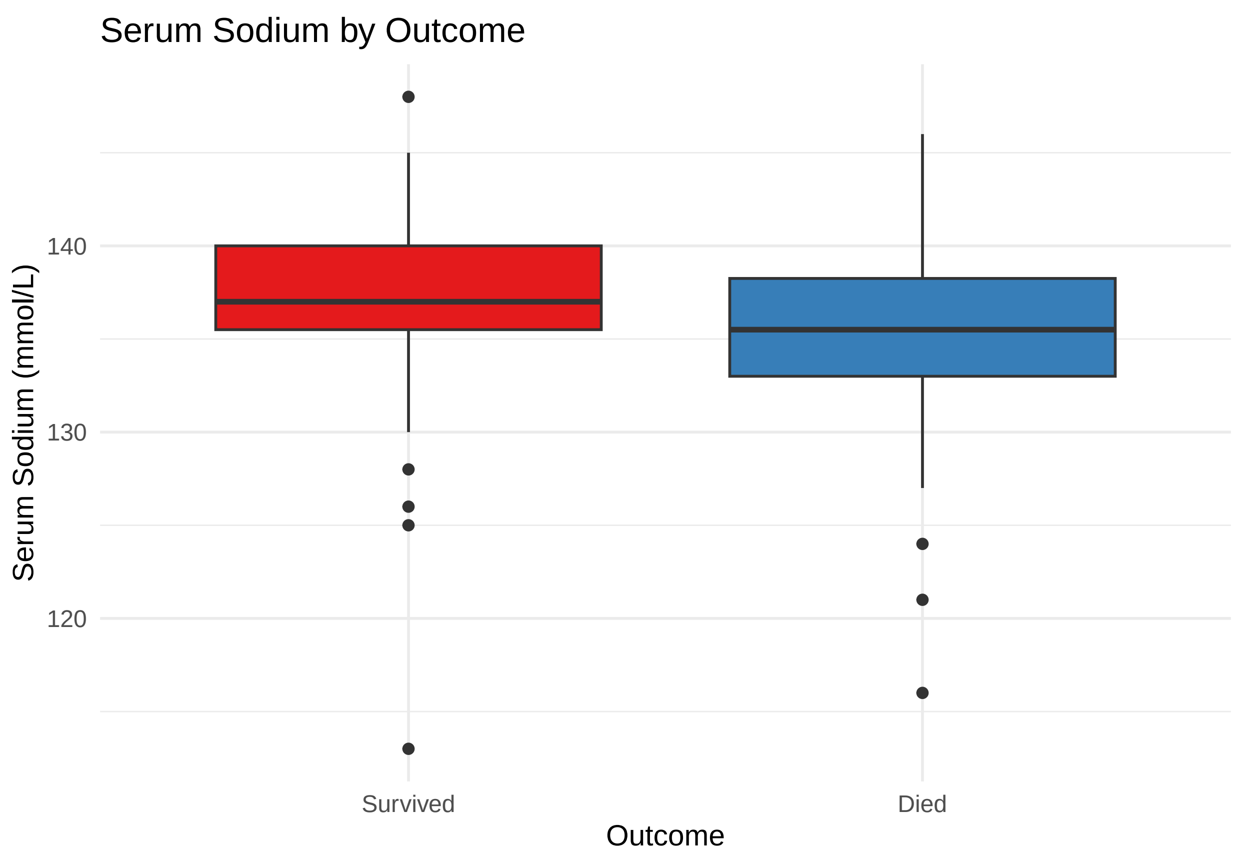
```
# Box plot of serum_creatinine by mortality
ggplot(heart_data, aes(x = fatal_mi, y = serum_creatinine, fill = fatal_mi)) +
  geom_boxplot() +
  labs(title = "Serum Creatinine by Outcome",
       x = "Outcome",
       y = "Serum Creatinine (mg/dL)") +
  scale_fill_brewer(palette = "Set1") +
  theme_minimal() +
  theme(legend.position = "none") +
  coord_cartesian(ylim = c(0, quantile(heart_data$serum_creatinine, 0.95)))  # Limit outliers
```



```r
# Calculate statistics for serum creatinine by outcome
creatinine_stats <- heart_data %>%
  dplyr::group_by(fatal_mi) %>%
  dplyr::summarize(
    count = n(),
    mean_cr = mean(serum_creatinine, na.rm = TRUE),
    median_cr = median(serum_creatinine, na.rm = TRUE),
    sd_cr = sd(serum_creatinine, na.rm = TRUE),
    min_cr = min(serum_creatinine, na.rm = TRUE),
    max_cr = max(serum_creatinine, na.rm = TRUE),
    elevated_cr_percent = 100 * sum(serum_creatinine > 1.2, na.rm = TRUE) / n()
  )

# 1.1.3 Examine electrolyte balance (serum_sodium)

# Histogram of serum_sodium
ggplot(heart_data, aes(x = serum_sodium)) +
  geom_histogram(bins = 15, fill = "orange", color = "black") +
  labs(title = "Distribution of Serum Sodium",
       x = "Serum Sodium (mmol/L)",
       y = "Count") +
  theme_minimal() +
  geom_vline(xintercept = c(135, 145), linetype = "dashed", color = "red") +
  annotate("text", x = 130, y = max(table(heart_data$serum_sodium)),
           label = "Low (<135)", color = "red") +
  annotate("text", x = 150, y = max(table(heart_data$serum_sodium)),
           label = "High (>145)", color = "red")
```



```r
# Box plot of serum_sodium by mortality
ggplot(heart_data, aes(x = fatal_mi, y = serum_sodium, fill = fatal_mi)) +
  geom_boxplot() +
  labs(title = "Serum Sodium by Outcome",
       x = "Outcome",
       y = "Serum Sodium (mmol/L)") +
  scale_fill_brewer(palette = "Set1") +
  theme_minimal() +
  theme(legend.position = "none")
```



```
# Calculate statistics for serum sodium by outcome
sodium_stats <- heart_data %>%
  dplyr::group_by(fatal_mi) %>%
  dplyr::summarize(
    count = n(),
    mean_na = mean(serum_sodium, na.rm = TRUE),
    median_na = median(serum_sodium, na.rm = TRUE),
    sd_na = sd(serum_sodium, na.rm = TRUE),
    min_na = min(serum_sodium, na.rm = TRUE),
    max_na = max(serum_sodium, na.rm = TRUE),
    low_na_percent = 100 * sum(serum_sodium < 135, na.rm = TRUE) / n(),
    high_na_percent = 100 * sum(serum_sodium > 145, na.rm = TRUE) / n()
  )

print(sodium_stats)
```

```
## # A tibble: 2 x 9
##   fatal_mi count mean_na median_na sd_na min_na max_na low_na_percent
##   <fct>    <int>   <dbl>     <dbl> <dbl>  <int>  <int>          <dbl>
## 1 Survived   203    137.       137  3.98    113    148           20.2
## 2 Died        96    135.      136.  5.00    116    146           43.8
## # i 1 more variable: high_na_percent <dbl>
```

```
# 1.1.4 Stratify patients by clinical risk groups

# Define clinical risk groups
heart_data <- heart_data %>%
```

```r
  mutate(
    ef_category = case_when(
      ejection_fraction < 40 ~ "Reduced",
      ejection_fraction >= 40 & ejection_fraction < 50 ~ "Mid-range",
      ejection_fraction >= 50 ~ "Preserved",
      TRUE ~ NA_character_
    ),
    creatinine_category = ifelse(serum_creatinine > 1.2, "Elevated", "Normal"),
    sodium_category = case_when(
      serum_sodium < 135 ~ "Low",
      serum_sodium > 145 ~ "High",
      TRUE ~ "Normal"
    ),
    age_group = case_when(
      age < 55 ~ "<55",
      age >= 55 & age < 65 ~ "55-64",
      age >= 65 & age < 75 ~ "65-74",
      age >= 75 ~ "75+",
      TRUE ~ NA_character_
    )
  )

# Count patients in each clinical risk group
risk_group_counts <- heart_data %>%
  dplyr::group_by(ef_category, creatinine_category, sodium_category) %>%
  dplyr::summarize(
    total_patients = n(),
    deaths = sum(fatal_mi == "Died", na.rm = TRUE),
    mortality_rate = 100 * deaths / total_patients
  ) %>%
  dplyr::arrange(desc(mortality_rate))
```

```
## `summarise()` has grouped output by 'ef_category', 'creatinine_category'. You
## can override using the `.groups` argument.
```

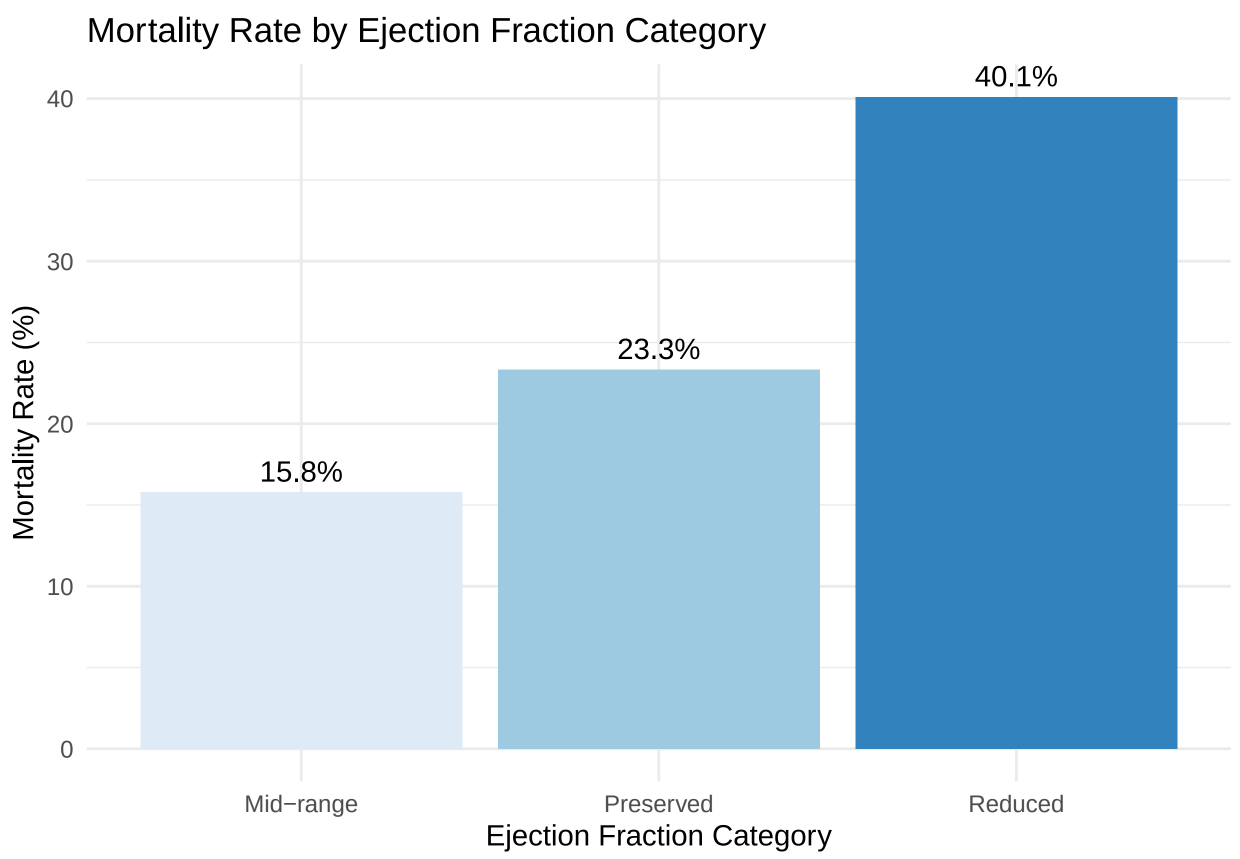
```r
print(risk_group_counts)
```

```
## # A tibble: 14 x 6
## # Groups:   ef_category, creatinine_category [6]
##    ef_category creatinine_category sodium_category total_patients deaths
##    <chr>       <chr>               <chr>                    <int>  <int>
##  1 Preserved   Elevated            High                         1      1
##  2 Mid-range   Elevated            Low                          5      4
##  3 Reduced     Elevated            Low                         35     25
##  4 Preserved   Normal              Low                          8      5
##  5 Reduced     Elevated            Normal                      37     16
##  6 Mid-range   Elevated            Normal                       7      3
##  7 Preserved   Elevated            Normal                      12      4
##  8 Reduced     Normal              Low                         22      7
##  9 Reduced     Normal              Normal                      88     25
## 10 Preserved   Elevated            Low                          4      1
## 11 Preserved   Normal              Normal                      34      3
## 12 Mid-range   Normal              Normal                      36      2
## 13 Mid-range   Normal              Low                          9      0
## 14 Preserved   Normal              High                             1      0
## # i 1 more variable: mortality_rate <dbl>
```

```r
# Visualize mortality rate by ejection fraction category
ef_mortality <- heart_data %>%
  dplyr::group_by(ef_category) %>%
  dplyr::summarize(
    total = n(),
    deaths = sum(fatal_mi == "Died", na.rm = TRUE),
    mortality_rate = 100 * deaths / total
  )

print(ef_mortality)
```

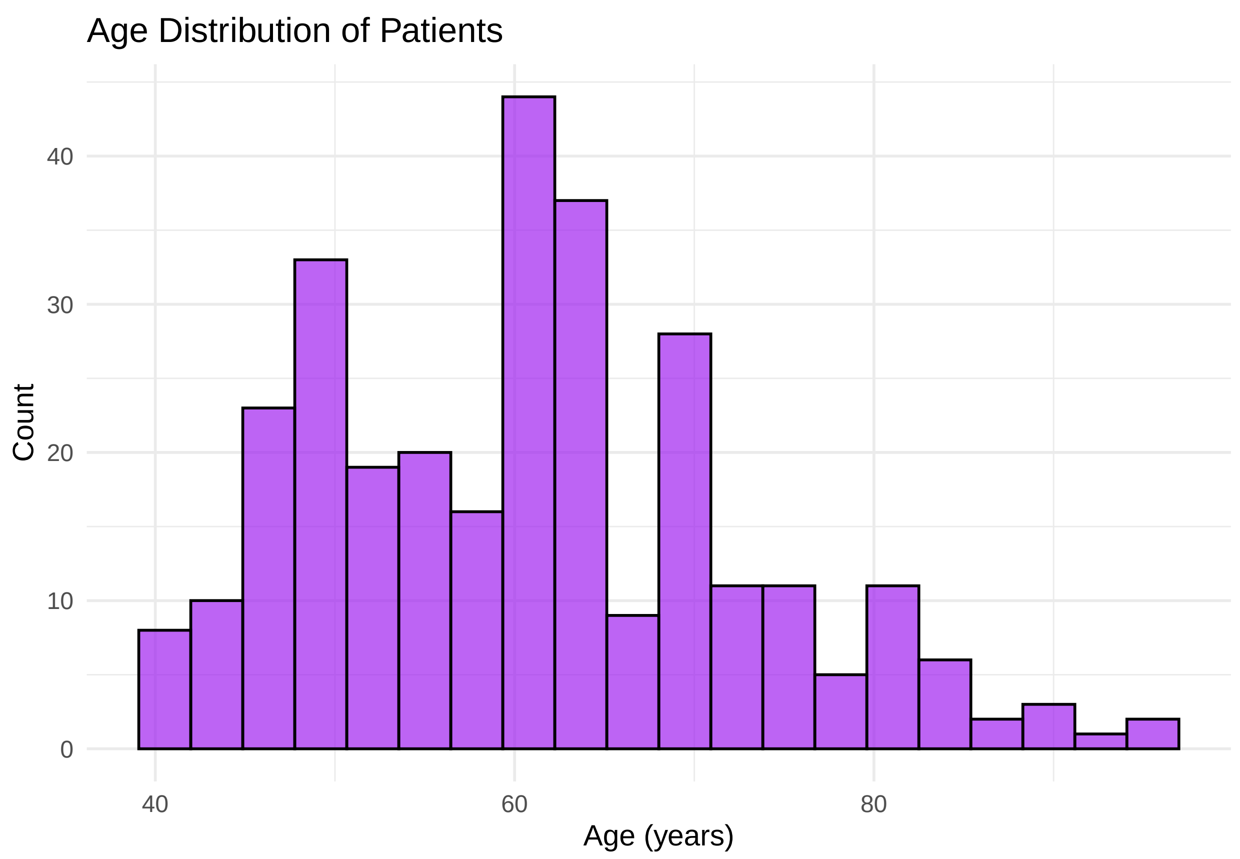
```
## # A tibble: 3 x 4
##   ef_category total deaths mortality_rate
##   <chr>       <int>  <int>          <dbl>
## 1 Mid-range      57      9           15.8
## 2 Preserved      60     14           23.3
## 3 Reduced       182     73           40.1
```

```r
ggplot(ef_mortality, aes(x = ef_category, y = mortality_rate, fill = ef_category)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = paste0(round(mortality_rate, 1), "%")), vjust = -0.5) +
  labs(title = "Mortality Rate by Ejection Fraction Category",
       x = "Ejection Fraction Category",
       y = "Mortality Rate (%)") +
  scale_fill_brewer(palette = "Blues") +
  theme_minimal() +
  theme(legend.position = "none")
```

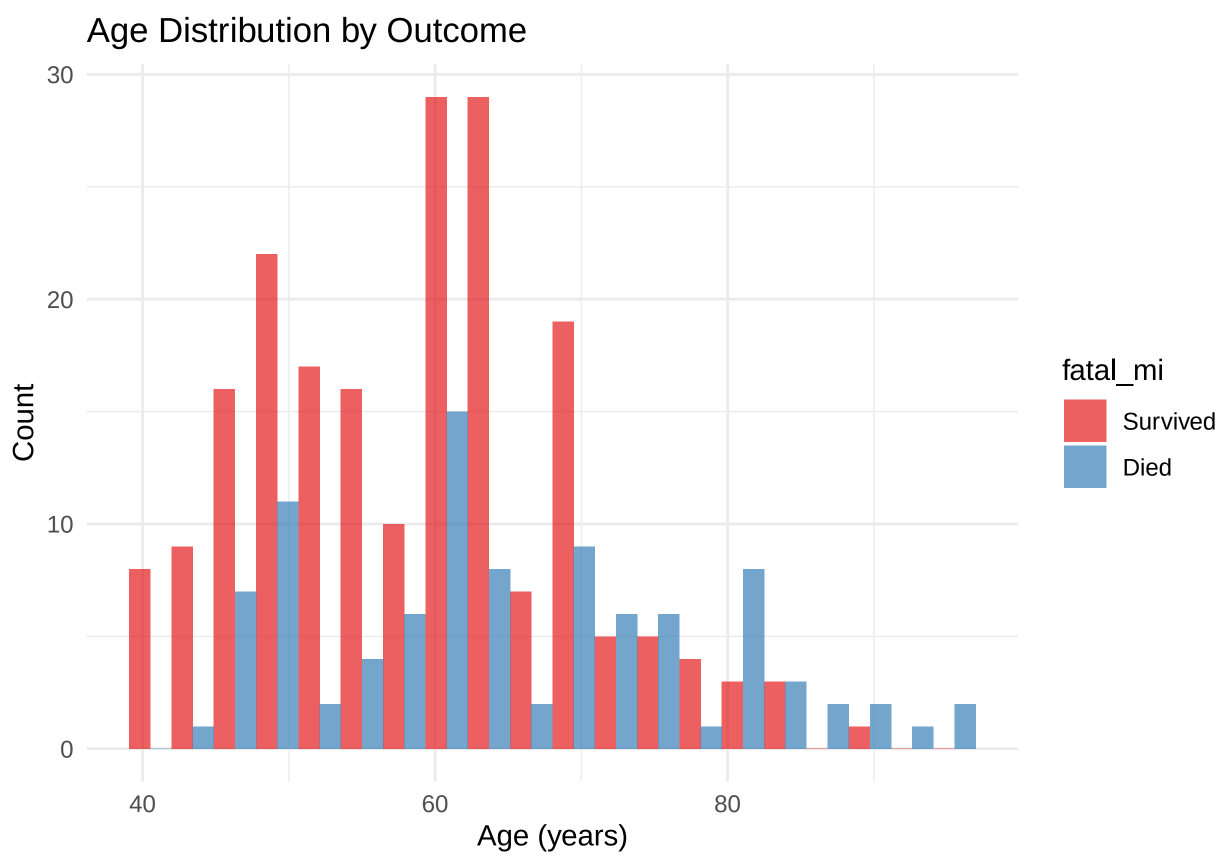


d)Analyze demographic distribution (age, sex) e)Calculate mortality rates across different comorbidities

```
# 1.1.5 Analyze demographic distribution (age, sex) ----------

# Age distribution
ggplot(heart_data, aes(x = age)) +
  geom_histogram(bins = 20, fill = "purple", color = "black", alpha = 0.7) +
  labs(title = "Age Distribution of Patients",
       x = "Age (years)",
       y = "Count") +
  theme_minimal()
```



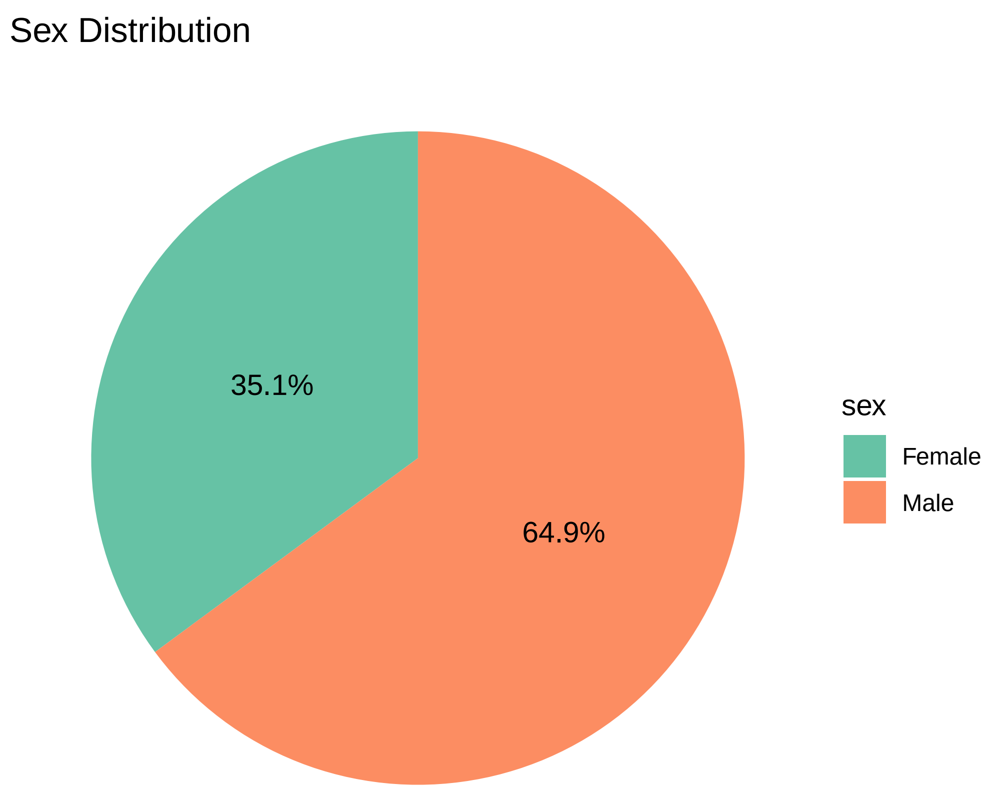
```
# Age distribution by outcome
ggplot(heart_data, aes(x = age, fill = fatal_mi)) +
  geom_histogram(bins = 20, position = "dodge", alpha = 0.7) +
  labs(title = "Age Distribution by Outcome",
       x = "Age (years)",
       y = "Count") +
  scale_fill_brewer(palette = "Set1") +
  theme_minimal()
```



```r
# Sex distribution
sex_counts <- heart_data %>%
  dplyr::group_by(sex) %>%
  dplyr::summarize(
    count = n(),
    percentage = 100 * n() / nrow(heart_data)
  )

print(sex_counts)
```

```
## # A tibble: 2 x 3
##   sex    count percentage
##   <fct>  <int>      <dbl>
## 1 Female   105       35.1
## 2 Male     194       64.9
```

```r
ggplot(sex_counts, aes(x = "", y = percentage, fill = sex)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar(theta = "y") +
  geom_text(aes(label = paste0(round(percentage, 1), "%")),
            position = position_stack(vjust = 0.5)) +
  labs(title = "Sex Distribution",
       x = NULL, y = NULL) +
  scale_fill_brewer(palette = "Set2") +
  theme_void()
```



```
# Mortality rate by sex
mortality_by_sex <- heart_data %>%
  dplyr::group_by(sex, fatal_mi) %>%
  dplyr::summarize(count = n()) %>%
  dplyr::group_by(sex) %>%
  dplyr::mutate(percentage = 100 * count / sum(count)) %>%
  dplyr::filter(fatal_mi == "Died")
```

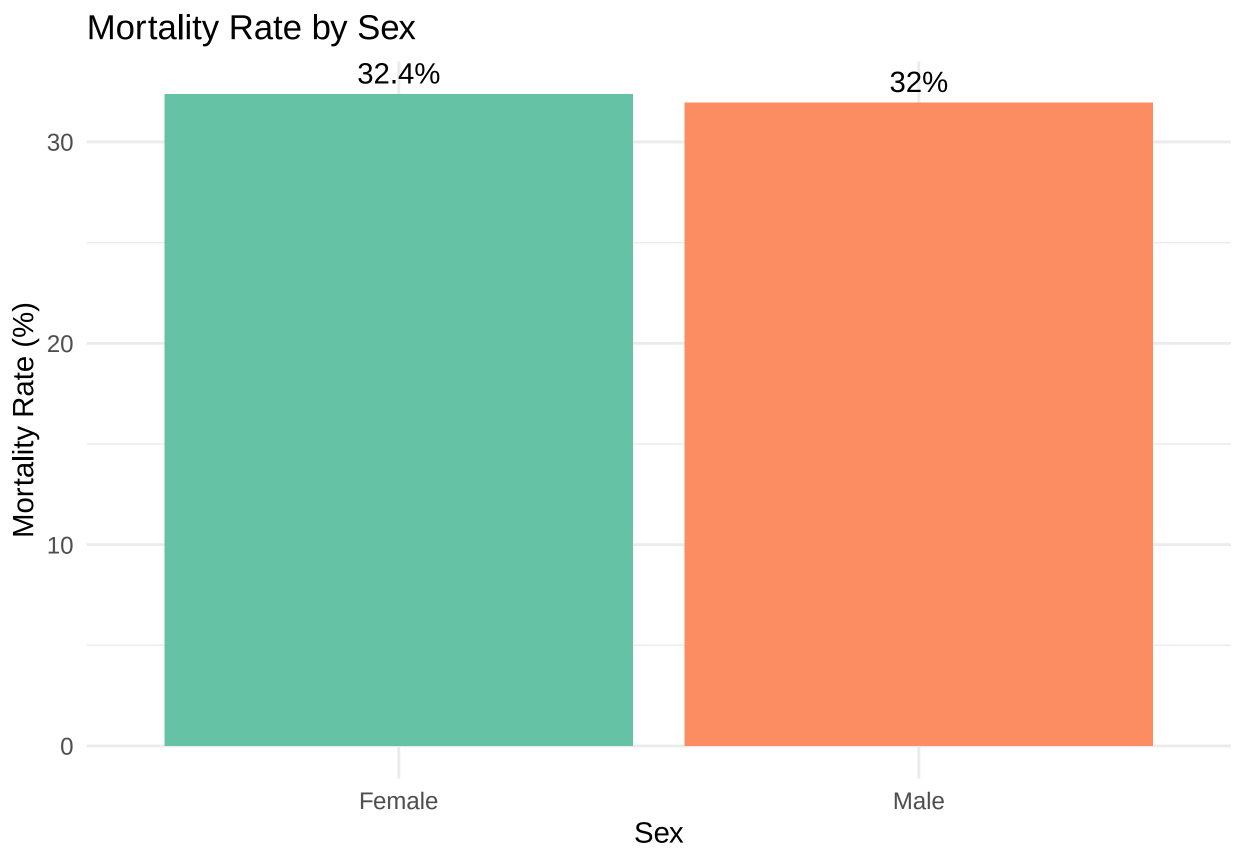
```
## `summarise()` has grouped output by 'sex'. You can override using the `.groups`
## argument.
```

```
print(mortality_by_sex)
```

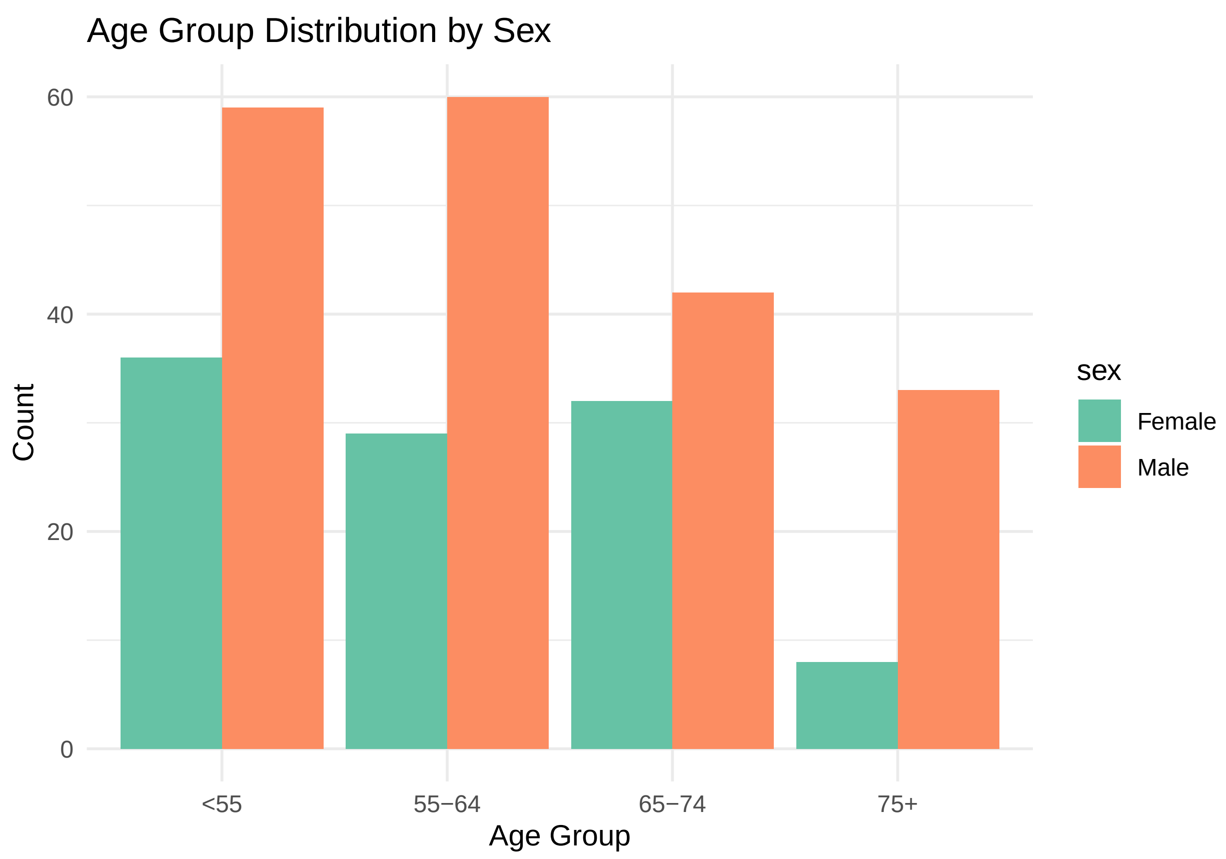
```
## # A tibble: 2 x 4
## # Groups:   sex [2]
##   sex    fatal_mi count percentage
##   <fct>  <fct>    <int>      <dbl>
## 1 Female Died        34       32.4
## 2 Male   Died        62       32.0
```

```
ggplot(mortality_by_sex, aes(x = sex, y = percentage, fill = sex)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = paste0(round(percentage, 1), "%")), vjust = -0.5) +
  labs(title = "Mortality Rate by Sex",
       x = "Sex",
       y = "Mortality Rate (%)") +
  scale_fill_brewer(palette = "Set2") +
  theme_minimal() +
```

```r
  theme(legend.position = "none")
```

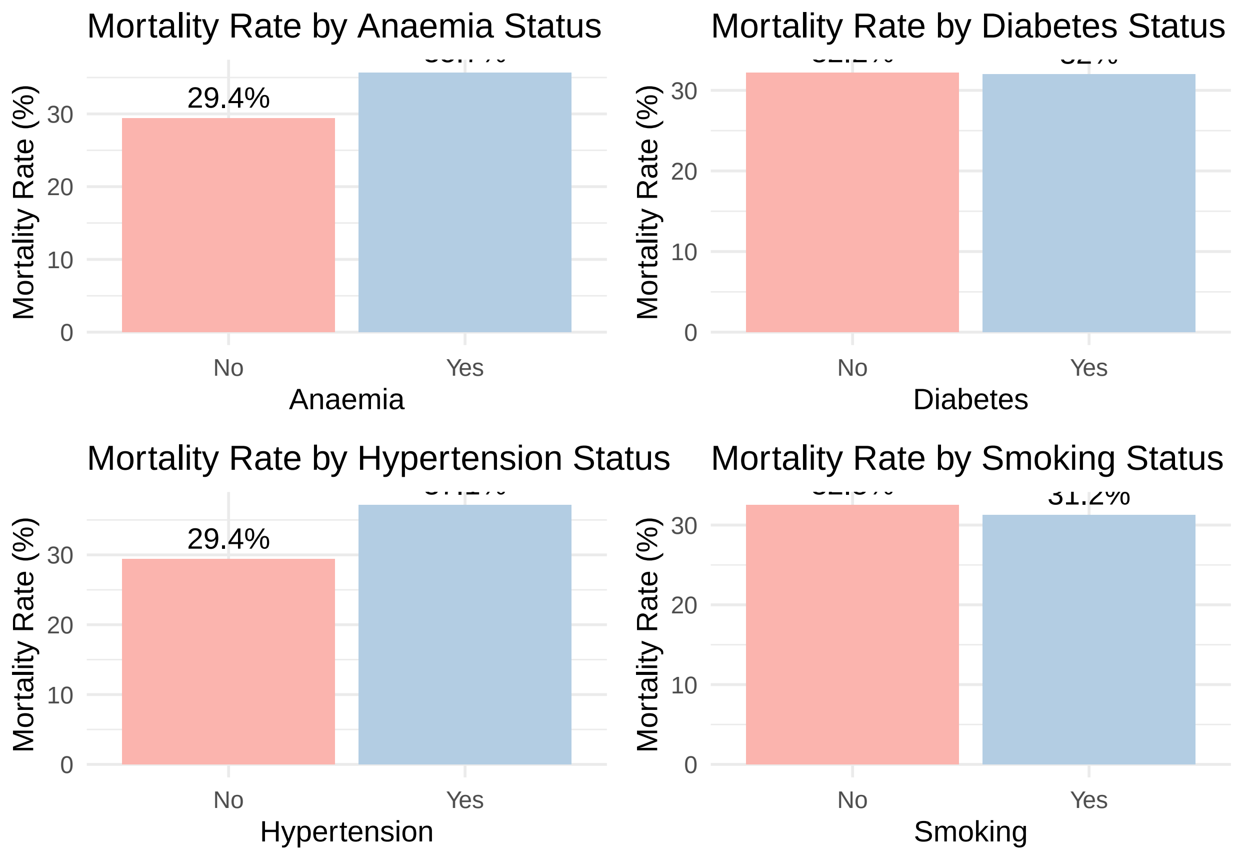


```r
# Age and sex combined analysis
ggplot(heart_data, aes(x = age_group, fill = sex)) +
  geom_bar(position = "dodge") +
  labs(title = "Age Group Distribution by Sex",
       x = "Age Group",
       y = "Count") +
  scale_fill_brewer(palette = "Set2") +
  theme_minimal()
```

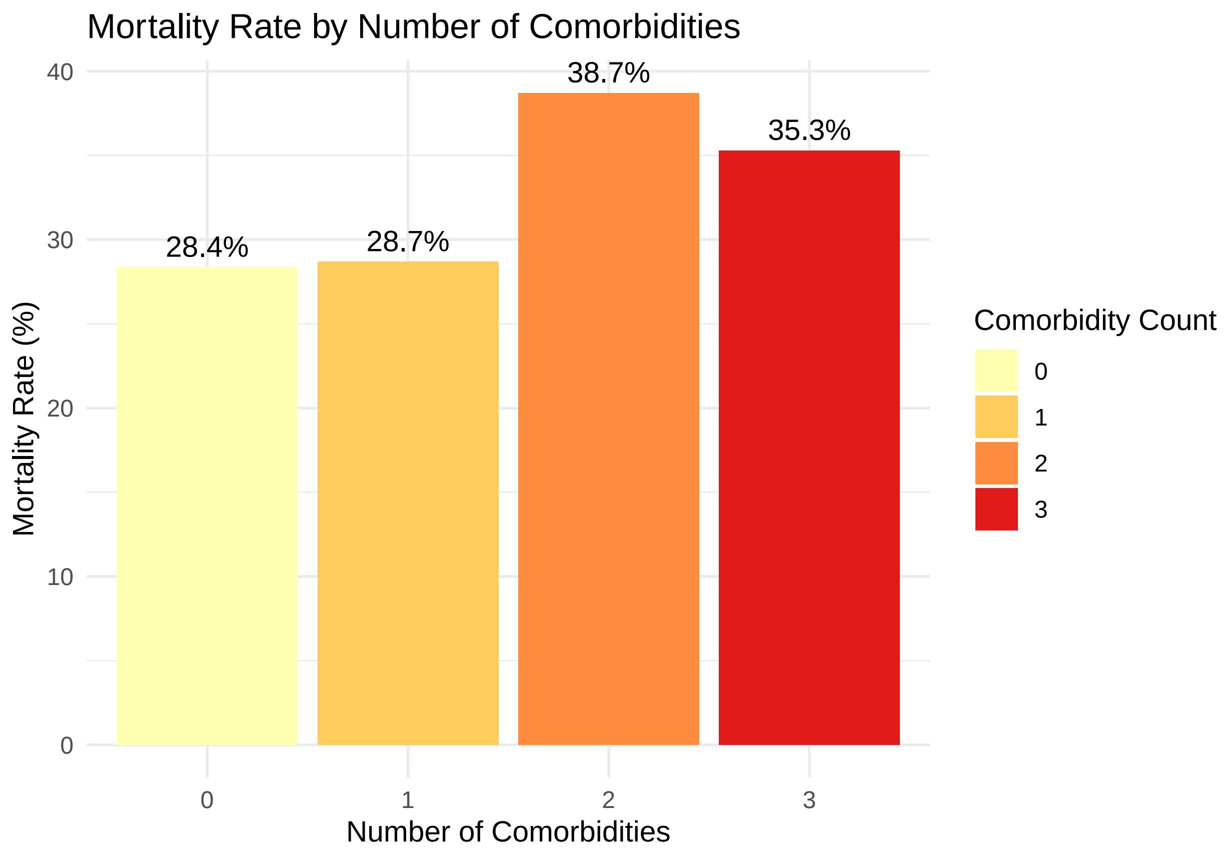


```
# 1.1.6 Calculate mortality rates across different comorbidities

# Create a function to calculate mortality rates for a categorical variable
calculate_mortality_rate <- function(data, var_name) {
  var_sym <- sym(var_name)  # Convert variable name to symbol

  mortality_data <- data %>%
    filter(!is.na(!!var_sym) & !is.na(fatal_mi)) %>%
    group_by(!!var_sym, fatal_mi) %>%
    summarise(count = n(), .groups = "drop") %>%
    group_by(!!var_sym) %>%
    mutate(percentage = 100 * count / sum(count)) %>%
    filter(fatal_mi == "Died")

  return(mortality_data)
}

# Calculate mortality rates for different comorbidities
anaemia_mortality <- calculate_mortality_rate(heart_data, "anaemia")
diabetes_mortality <- calculate_mortality_rate(heart_data, "diabetes")
hypertension_mortality <- calculate_mortality_rate(heart_data, "high_blood_pressure")
smoking_mortality <- calculate_mortality_rate(heart_data, "smoking")

# Create plots for each comorbidity
plot_anaemia <- ggplot(anaemia_mortality,
                       aes(x = anaemia, y = percentage, fill = anaemia)) +
```

```r
  geom_bar(stat = "identity") +
  geom_text(aes(label = paste0(round(percentage, 1), "%")), vjust = -0.5) +
  labs(title = "Mortality Rate by Anaemia Status",
       x = "Anaemia",
       y = "Mortality Rate (%)") +
  scale_fill_brewer(palette = "Pastel1") +
  theme_minimal() +
  theme(legend.position = "none")

plot_diabetes <- ggplot(diabetes_mortality,
                        aes(x = diabetes, y = percentage, fill = diabetes)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = paste0(round(percentage, 1), "%")), vjust = -0.5) +
  labs(title = "Mortality Rate by Diabetes Status",
       x = "Diabetes",
       y = "Mortality Rate (%)") +
  scale_fill_brewer(palette = "Pastel1") +
  theme_minimal() +
  theme(legend.position = "none")

plot_hypertension <- ggplot(hypertension_mortality,
                           aes(x = high_blood_pressure, y = percentage,
                               fill = high_blood_pressure)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = paste0(round(percentage, 1), "%")), vjust = -0.5) +
  labs(title = "Mortality Rate by Hypertension Status",
       x = "Hypertension",
       y = "Mortality Rate (%)") +
  scale_fill_brewer(palette = "Pastel1") +
  theme_minimal() +
  theme(legend.position = "none")

plot_smoking <- ggplot(smoking_mortality,
                      aes(x = smoking, y = percentage, fill = smoking)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = paste0(round(percentage, 1), "%")), vjust = -0.5) +
  labs(title = "Mortality Rate by Smoking Status",
       x = "Smoking",
       y = "Mortality Rate (%)") +
  scale_fill_brewer(palette = "Pastel1") +
  theme_minimal() +
  theme(legend.position = "none")

# Arrange all four plots in a single figure
grid.arrange(plot_anaemia, plot_diabetes, plot_hypertension, plot_smoking,
             ncol = 2)
```



```r
# Multi-comorbidity analysis
heart_data <- heart_data %>%
  mutate(
    comorbidity_count = (anaemia == "Yes") +
                        (diabetes == "Yes") +
                        (high_blood_pressure == "Yes")
  )

# Mortality rate by number of comorbidities
comorbidity_mortality <- heart_data %>%
  filter(!is.na(comorbidity_count) & !is.na(fatal_mi)) %>%
  group_by(comorbidity_count, fatal_mi) %>%
  summarise(count = n(), .groups = "drop") %>%
  group_by(comorbidity_count) %>%
  mutate(percentage = 100 * count / sum(count)) %>%
  filter(fatal_mi == "Died")

# Plot the mortality rate by number of comorbidities
ggplot(comorbidity_mortality,
       aes(x = as.factor(comorbidity_count), y = percentage,
           fill = as.factor(comorbidity_count))) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = paste0(round(percentage, 1), "%")), vjust = -0.5) +
  labs(title = "Mortality Rate by Number of Comorbidities",
       x = "Number of Comorbidities",
       y = "Mortality Rate (%)",
      fill = "Comorbidity Count") +
  scale_fill_brewer(palette = "YlOrRd") +
  theme_minimal()
```



1.2 Medical Domain-Informed Data Preparation

Implement clinical thresholds for continuous variables: Categorize ejection fraction (preserved 50%, mid-range 40-49%, reduced <40%) Flag abnormal sodium levels (<135 or >145 mmol/L) Identify elevated creatinine (>1.2 mg/dL) Create clinically relevant interaction features: Cardiorenal interaction (ejection_fraction × serum_creatinine) Age-comorbidity interactions Diabetes-hypertension combination feature

```
# 1.2 Medical Domain-Informed Data Preparation
# 1.2.1 Implement clinical thresholds for continuous variables



# Create clinical categories based on established thresholds
heart_clinical <- heart_data %>%
  mutate(
    # Ejection fraction categories (clinically established)
    ef_clinical = case_when(
      ejection_fraction < 40 ~ "Reduced",
      ejection_fraction >= 40 & ejection_fraction < 50 ~ "Mid-range",
      ejection_fraction >= 50 ~ "Preserved",
      TRUE ~ NA_character_
    ),
```

```r
    # Sodium level categories
    sodium_clinical = case_when(
      serum_sodium < 135 ~ "Hyponatremia",
      serum_sodium > 145 ~ "Hypernatremia",
      TRUE ~ "Normal"
    ),

    # Creatinine level categories
    creatinine_clinical = case_when(
      serum_creatinine > 1.2 ~ "Elevated",
      TRUE ~ "Normal"
    ),

    # Age categories for cardiovascular risk
    age_clinical = case_when(
      age < 45 ~ "Young",
      age >= 45 & age < 65 ~ "Middle",
      age >= 65 ~ "Elderly",
      TRUE ~ NA_character_
    ),

    # CPK (Creatinine Phosphokinase) categories
    # Normal values vary by sex
    cpk_clinical = case_when(
      (sex == "Male" & creatinine_phosphokinase > 200) |
      (sex == "Female" & creatinine_phosphokinase > 170) ~ "Elevated",
      TRUE ~ "Normal"
    ),

    # Platelets categories
    platelets_clinical = case_when(
      platelets < 150000 ~ "Low",
      platelets > 450000 ~ "High",
      TRUE ~ "Normal"
    )
  )

# Count patients in each clinical category
ef_counts <- table(heart_clinical$ef_clinical)
sodium_counts <- table(heart_clinical$sodium_clinical)
creatinine_counts <- table(heart_clinical$creatinine_clinical)
cpk_counts <- table(heart_clinical$cpk_clinical)
platelets_counts <- table(heart_clinical$platelets_clinical)

# Print counts
cat("Ejection Fraction Categories:\n")
```

```
## Ejection Fraction Categories:
```

```r
print(ef_counts)
```

```
##
## Mid-range Preserved   Reduced
##        57        60       182
```

```
cat("\nSodium Level Categories:\n")
```

```
##
## Sodium Level Categories:
```

```
print(sodium_counts)
```

```
##
## Hypernatremia  Hyponatremia        Normal
##             2            83           214
```

```
cat("\nCreatinine Level Categories:\n")
```

```
##
## Creatinine Level Categories:
```

```
print(creatinine_counts)
```

```
##
## Elevated   Normal
##      101      198
```

```
cat("\nCPK Level Categories:\n")
```

```
##
## CPK Level Categories:
```

```
print(cpk_counts)
```

```
##
## Elevated   Normal
##      171      128
```

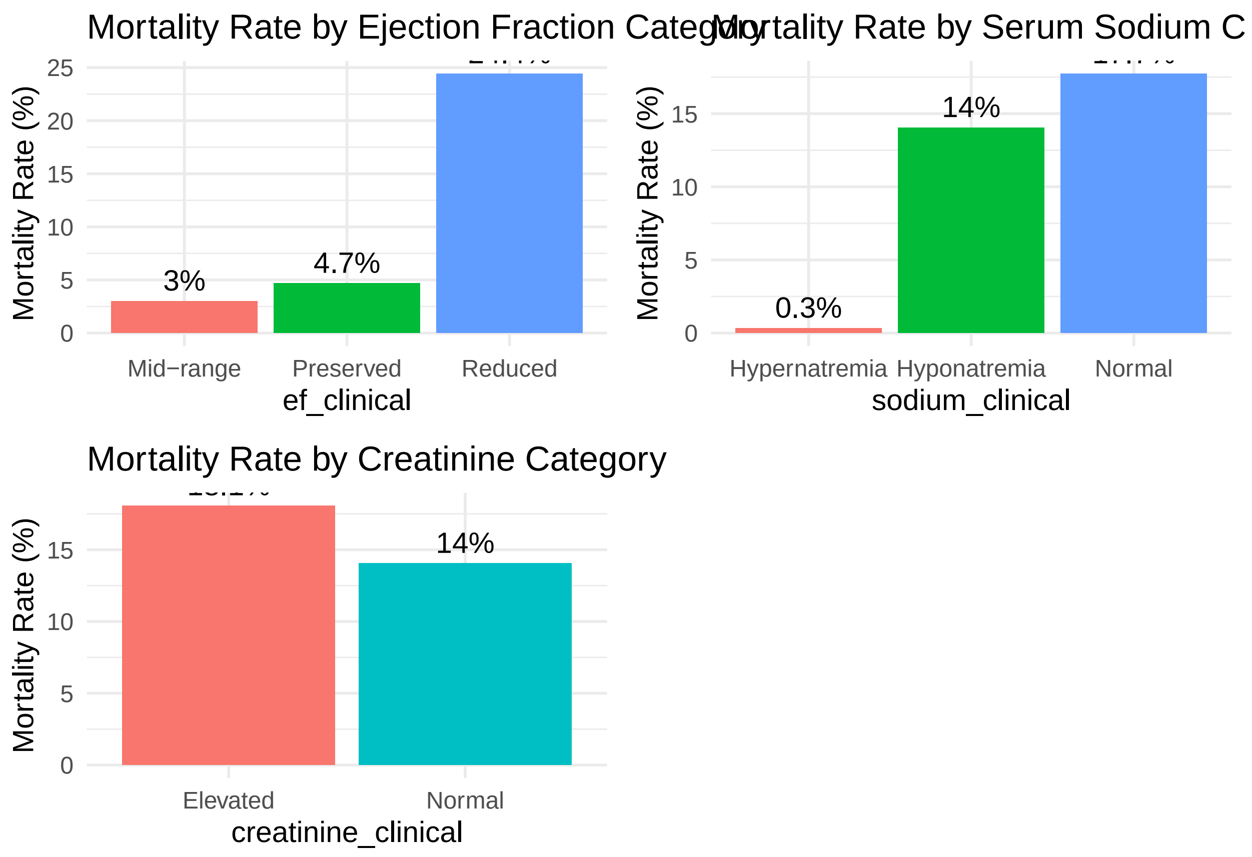
```
cat("\nPlatelets Categories:\n")
```

```
##
## Platelets Categories:
```

```
print(platelets_counts)
```

```
##
##   High    Low Normal
##     13     27    259
```

```
# Visualize distribution of clinical categories with mortality rates
plot_clinical_mortality <- function(data, var_name, title) {
  var_sym <- sym(var_name)

  mortality_data <- data %>%
    filter(!is.na(!!var_sym) & !is.na(fatal_mi)) %>%
    group_by(!!var_sym, fatal_mi) %>%
    summarise(count = n(), .groups = "drop") %>%
    mutate(percentage = 100 * count / sum(count, na.rm = TRUE)) %>%
    filter(fatal_mi == "Died")

  ggplot(mortality_data, aes(x = !!var_sym, y = percentage, fill = as.factor(!!var_sym))) +
    geom_bar(stat = "identity") +
    geom_text(aes(label = paste0(round(percentage, 1), "%")), vjust = -0.5) +
    labs(title = title,
         x = var_name,
         y = "Mortality Rate (%)") +
    theme_minimal() +
    theme(legend.position = "none")
}

# Generate plots
ef_plot <- plot_clinical_mortality(heart_clinical, "ef_clinical",
                                   "Mortality Rate by Ejection Fraction Category")
sodium_plot <- plot_clinical_mortality(heart_clinical, "sodium_clinical",
                                       "Mortality Rate by Serum Sodium Category")
creatinine_plot <- plot_clinical_mortality(heart_clinical, "creatinine_clinical",
                                           "Mortality Rate by Creatinine Category")

grid.arrange(ef_plot, sodium_plot, creatinine_plot, ncol = 2)
```

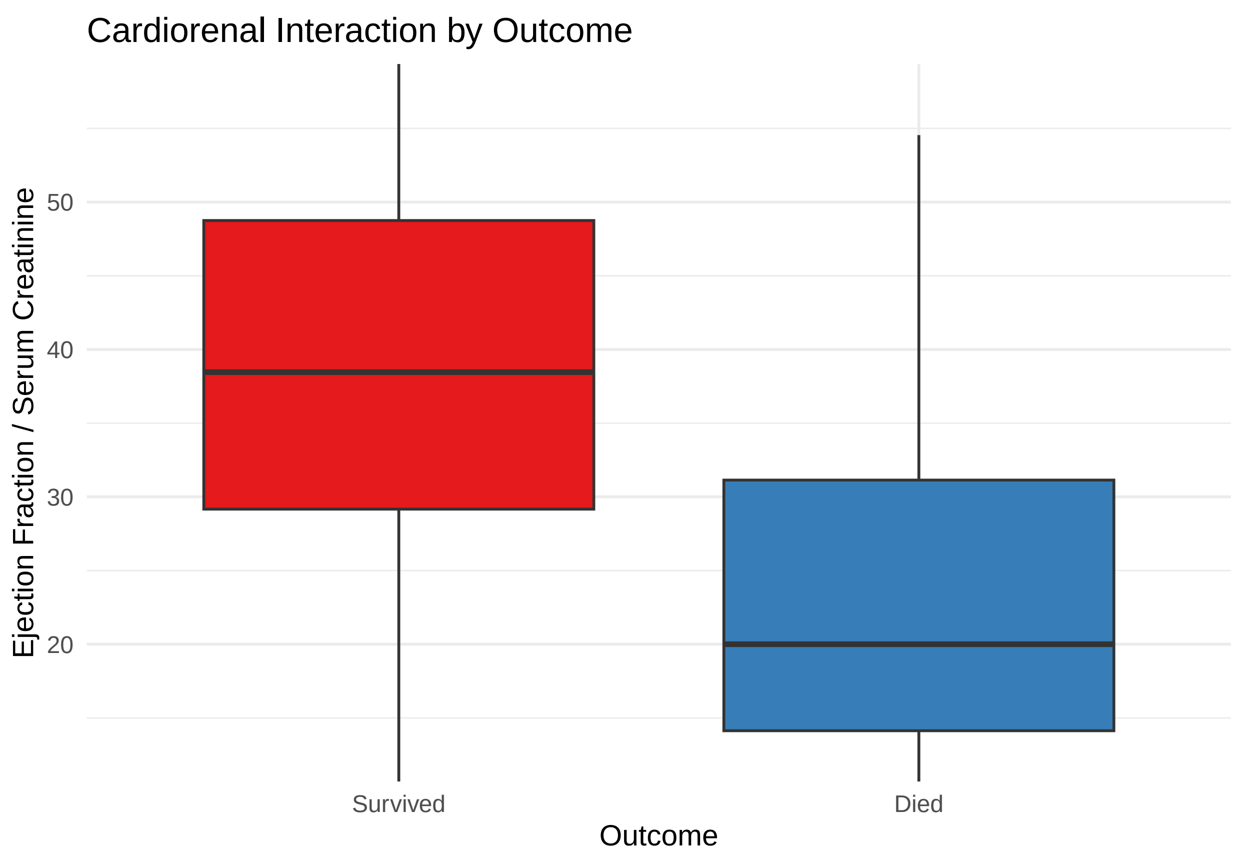


```
# 1.2.2 Create clinically relevant interaction features

# Cardiorenal interaction (ejection_fraction × serum_creatinine)
heart_clinical <- heart_clinical %>%
  mutate(
    # inverse relationship (lower EF and higher creatinine is worse)
    cardiorenal_interaction = ejection_fraction / serum_creatinine,

    # Age-comorbidity interactions
    age_diabetes_interaction = as.numeric(age * (diabetes == "Yes")),
```

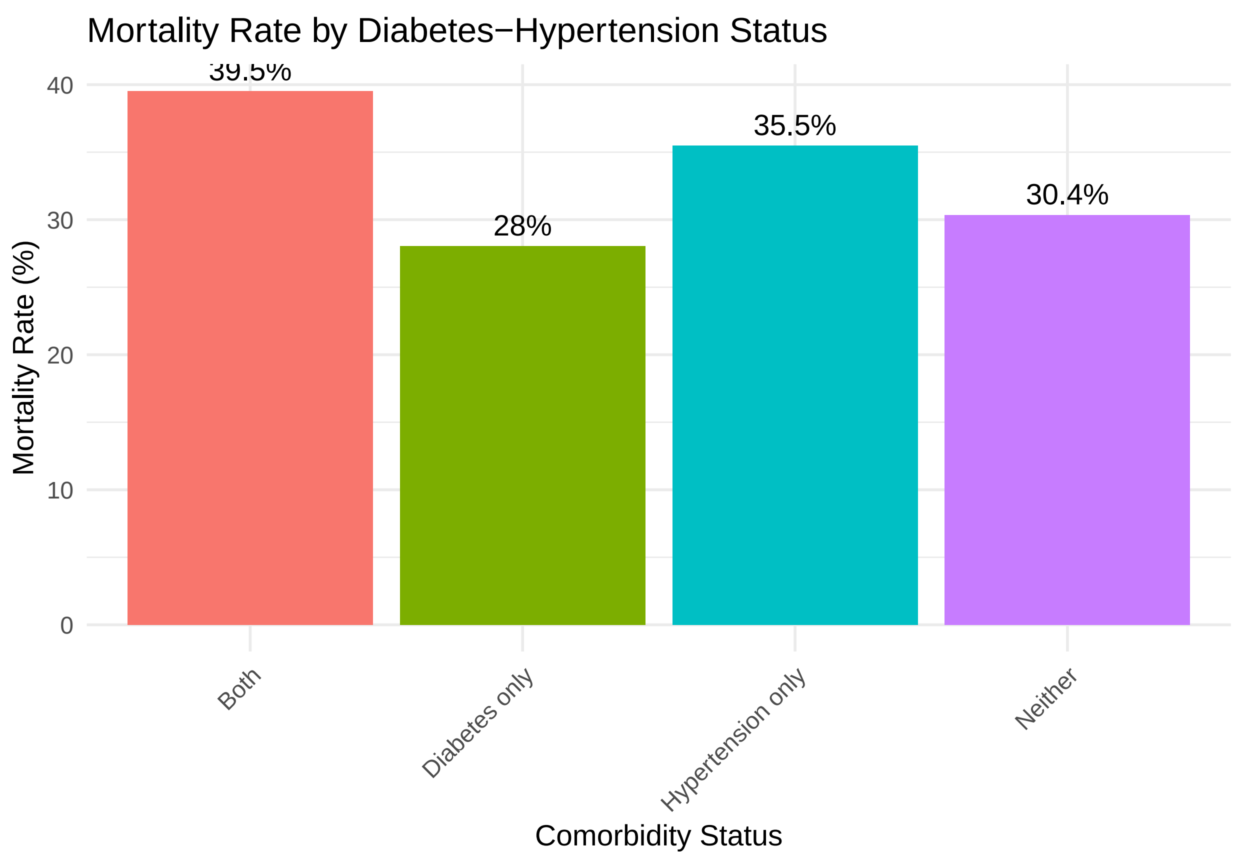
```r
    age_hypertension_interaction = as.numeric(age * (high_blood_pressure == "Yes")),

    # Diabetes-hypertension combination feature
    diabetes_hypertension = case_when(
      diabetes == "Yes" & high_blood_pressure == "Yes" ~ "Both",
      diabetes == "Yes" & high_blood_pressure == "No" ~ "Diabetes only",
      diabetes == "No" & high_blood_pressure == "Yes" ~ "Hypertension only",
      TRUE ~ "Neither"
    ),

    # EF and age interaction (elderly with low EF have worse outcomes)
    ef_age_risk = case_when(
      age >= 65 & ejection_fraction < 40 ~ "High risk",
      age >= 65 | ejection_fraction < 40 ~ "Moderate risk",
      TRUE ~ "Lower risk"
    ),

    # Multiple system involvement
    multisystem_involvement = (anaemia == "Yes") +
                              (creatinine_clinical == "Elevated") +
                              (sodium_clinical != "Normal") +
                              (ef_clinical == "Reduced")
  )

# Examine the cardiorenal interaction feature
summary(heart_clinical$cardiorenal_interaction)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.723  20.833  33.333  34.446  43.750  85.714
```

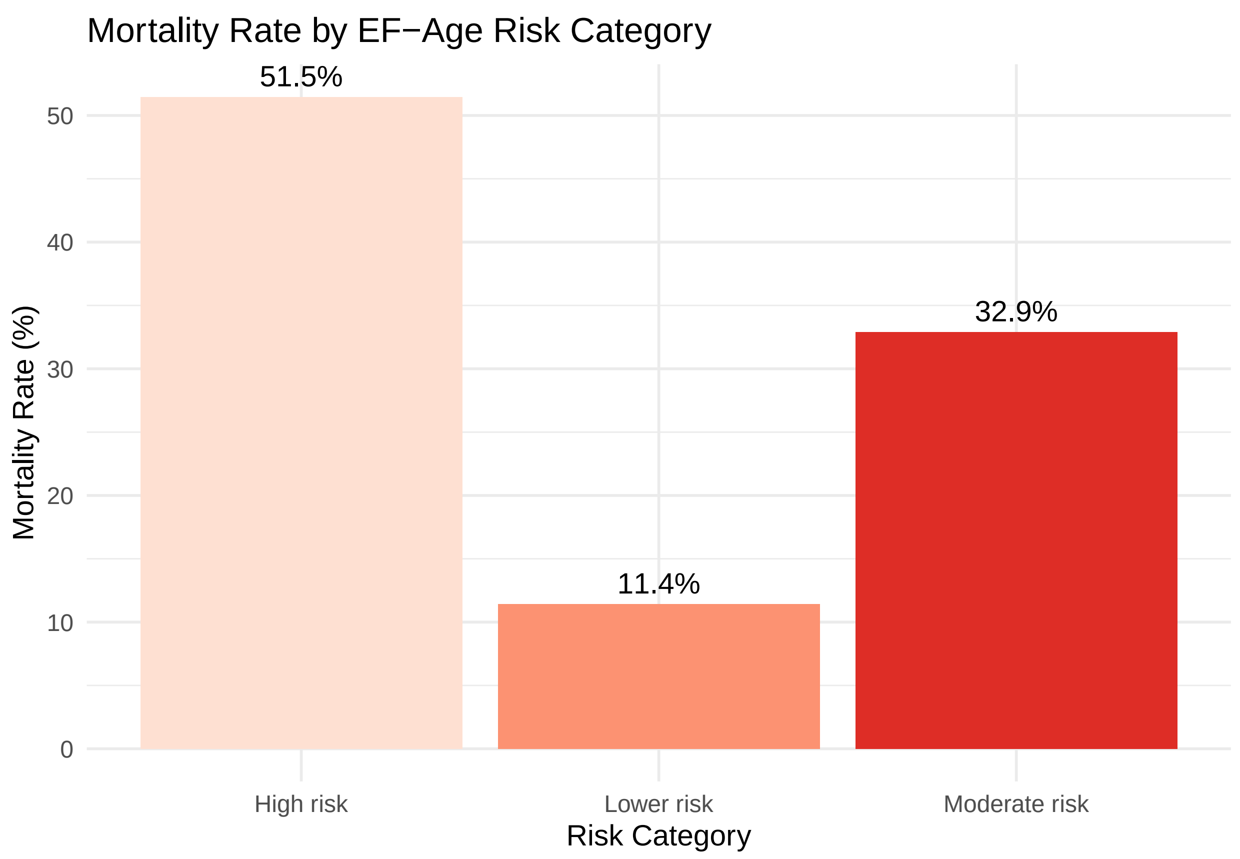
```r
# Visualization cardiorenal interaction by outcome
ggplot(heart_clinical, aes(x = fatal_mi, y = cardiorenal_interaction, fill = fatal_mi)) +
  geom_boxplot() +
  labs(title = "Cardiorenal Interaction by Outcome",
       x = "Outcome",
       y = "Ejection Fraction / Serum Creatinine") +
  scale_fill_brewer(palette = "Set1") +
  theme_minimal() +
  theme(legend.position = "none") +
  coord_cartesian(ylim = quantile(heart_clinical$cardiorenal_interaction, c(0.1, 0.9)))
```



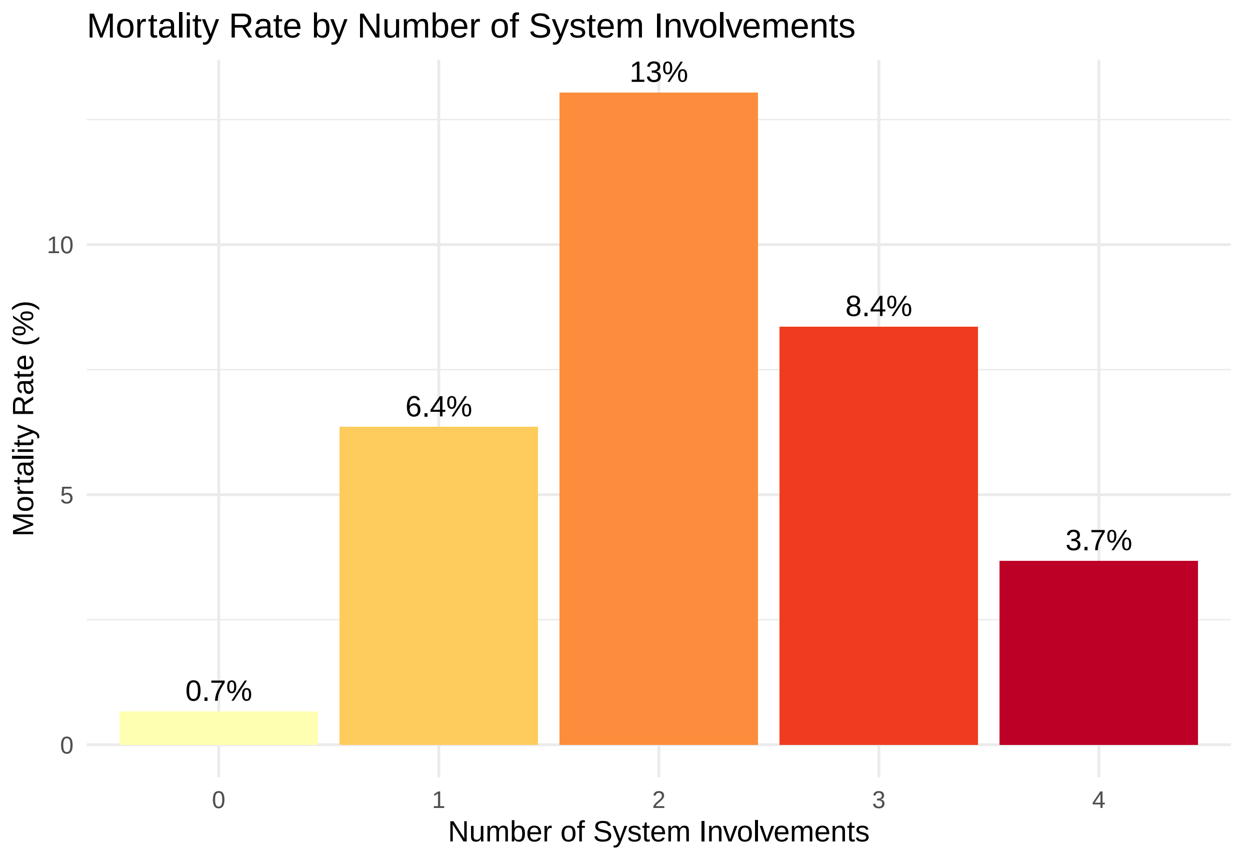
```r
# Mortality rate by diabetes-hypertension combination
diabetes_htn_mortality <- heart_clinical %>%
  group_by(diabetes_hypertension, fatal_mi) %>%
  summarise(count = n(), .groups = "drop") %>%
  group_by(diabetes_hypertension) %>%
  mutate(percentage = 100 * count / sum(count, na.rm = TRUE)) %>%
  filter(fatal_mi == "Died")

ggplot(diabetes_htn_mortality,
       aes(x = diabetes_hypertension, y = percentage, fill = diabetes_hypertension)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = paste0(round(percentage, 1), "%")), vjust = -0.5) +
  labs(title = "Mortality Rate by Diabetes-Hypertension Status",
       x = "Comorbidity Status",
       y = "Mortality Rate (%)") +
  theme_minimal() +
  theme(legend.position = "none", axis.text.x = element_text(angle = 45, hjust = 1))
```



```r
# Mortality rate by EF-age risk category
ef_age_mortality <- heart_clinical %>%
  group_by(ef_age_risk, fatal_mi) %>%
  summarise(count = n(), .groups = "drop") %>%  # Fix grouping issue
  group_by(ef_age_risk) %>%
  mutate(percentage = 100 * count / sum(count, na.rm = TRUE)) %>%
  filter(fatal_mi == "Died")

ggplot(ef_age_mortality,
       aes(x = ef_age_risk, y = percentage, fill = ef_age_risk)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = paste0(round(percentage, 1), "%")), vjust = -0.5) +
  labs(title = "Mortality Rate by EF-Age Risk Category",
       x = "Risk Category",
       y = "Mortality Rate (%)") +
  scale_fill_brewer(palette = "Reds") +
  theme_minimal() +
  theme(legend.position = "none")
```



```r
# Mortality rate by multisystem involvement
multisystem_mortality <- heart_clinical %>%
  filter(!is.na(multisystem_involvement) & !is.na(fatal_mi)) %>%
  group_by(multisystem_involvement, fatal_mi) %>%
  summarise(count = n(), .groups = "drop") %>%
  mutate(percentage = 100 * count / sum(count)) %>%
  filter(fatal_mi == "Died")

ggplot(multisystem_mortality,
       aes(x = as.factor(multisystem_involvement), y = percentage,
           fill = as.factor(multisystem_involvement))) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = paste0(round(percentage, 1), "%")), vjust = -0.5) +
  labs(title = "Mortality Rate by Number of System Involvements",
       x = "Number of System Involvements",
       y = "Mortality Rate (%)",
       fill = "Count") +
  scale_fill_brewer(palette = "YlOrRd") +
  theme_minimal() +
  theme(legend.position = "none")
```



1.2.3 Handle class imbalance with medically appropriate techniques

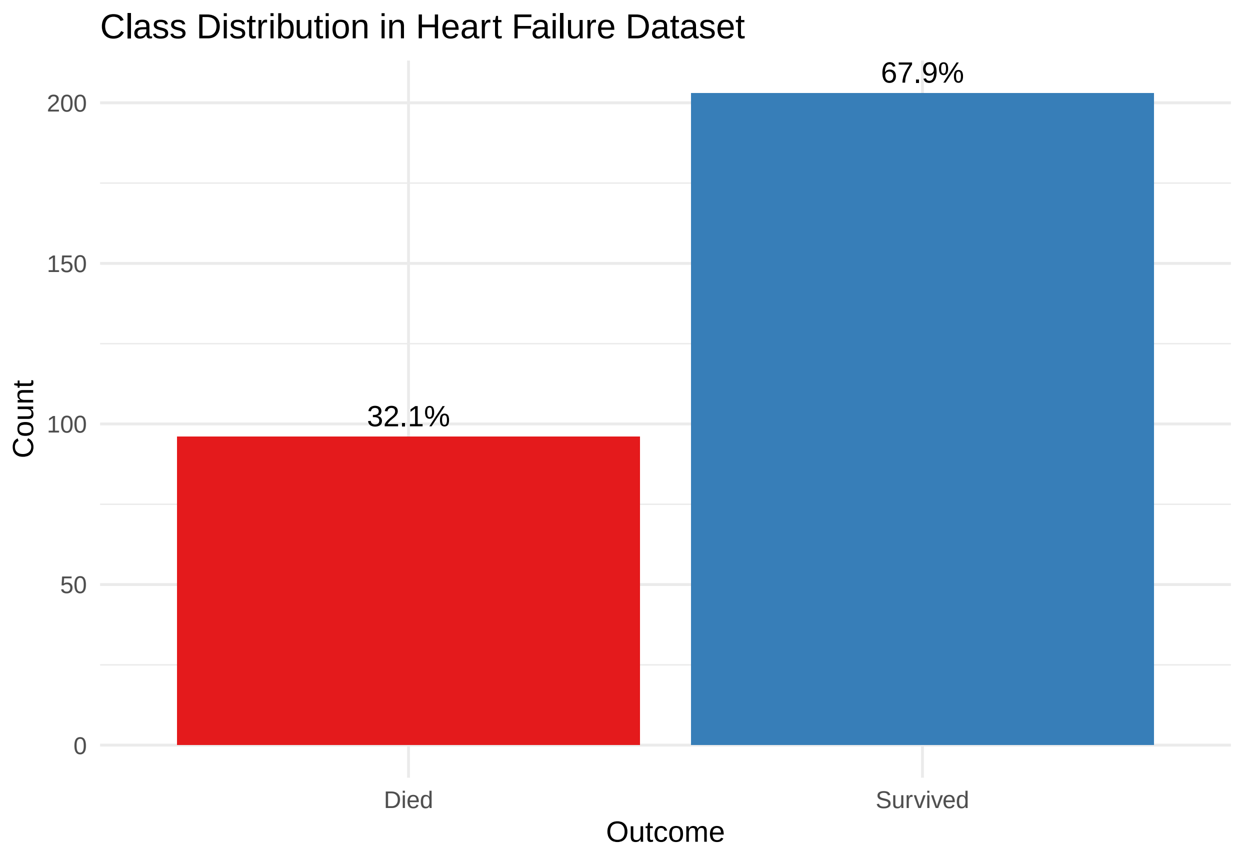
```r
# current class balance
class_balance <- table(heart_clinical$fatal_mi)
print(class_balance)
```

```
##
## Survived     Died
##      203       96
```

```r
class_percentage <- 100 * class_balance / sum(class_balance)
print(class_percentage)
```

```
##
## Survived     Died
## 67.89298 32.10702
```

```r
# Visualize class balance
ggplot(data.frame(Outcome = names(class_balance), Count = as.vector(class_balance)),
       aes(x = Outcome, y = Count, fill = Outcome)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = paste0(round(as.vector(class_percentage), 1), "%")),
            vjust = -0.5) +
  labs(title = "Class Distribution in Heart Failure Dataset",
       x = "Outcome",
       y = "Count") +
  scale_fill_brewer(palette = "Set1") +
  theme_minimal() +
  theme(legend.position = "none")
```



2.1 Evaluation Setup

```r
# 2.1.1 Preparing data for modeling
model_data <- heart_clinical %>%
  # Select variables for modeling
  select(
    # Demographics
    age, sex,

    # Clinical measurements
    ejection_fraction, serum_creatinine, serum_sodium,
    creatinine_phosphokinase, platelets, time,

    # Binary risk factors
    anaemia, diabetes, high_blood_pressure, smoking,

    # Derived clinical categories
    ef_clinical, sodium_clinical, creatinine_clinical,
    # Interaction terms
    cardiorenal_interaction,
    diabetes_hypertension,
    ef_age_risk,
    multisystem_involvement,

    # Target variable
    fatal_mi
  )

# # Check for any missing values
# missing_values <- colSums(is.na(model_data))
# if(sum(missing_values) > 0) {
#   cat("Missing values detected:\n")
#   print(missing_values[missing_values > 0])
# }

# Convert categorical variables to factors
model_data <- model_data %>%
  mutate_if(is.character, as.factor)

# 2.1.2 Implement stratified cross-validation
# Creating a function for stratified CV evaluation
evaluate_model <- function(model_fn, data, target, k = 5) {
  # Create stratified folds
  set.seed(123)
  folds <- createFolds(data[[target]], k = k, list = TRUE, returnTrain = FALSE)

  # Metrics storage
  metrics <- list(
    accuracy = numeric(k),
    sensitivity = numeric(k),
    specificity = numeric(k),
    ppv = numeric(k),
    npv = numeric(k),
    auc = numeric(k)
  )

  # Feature importance storage
  feature_importance <- NULL

  # For each fold
  for(i in 1:k) {
    # Split data
    test_indices <- folds[[i]]
    train_data <- data[-test_indices, ]
    test_data <- data[test_indices, ]

    # Build model
    model_result <- model_fn(train_data, test_data)

    # Store metrics
    metrics$accuracy[i] <- model_result$cm$overall["Accuracy"]
```

```r
    metrics$sensitivity[i] <- model_result$cm$byClass["Sensitivity"]
    metrics$specificity[i] <- model_result$cm$byClass["Specificity"]
    metrics$ppv[i] <- model_result$cm$byClass["Pos Pred Value"]
    metrics$npv[i] <- model_result$cm$byClass["Neg Pred Value"]
    metrics$auc[i] <- model_result$auc

    # Store feature importance
    if(!is.null(model_result$importance)) {
      if(is.null(feature_importance)) {
        feature_importance <- model_result$importance
      } else {
        feature_importance <- feature_importance + model_result$importance
      }
    }
  }

  # Average the metrics
  avg_metrics <- sapply(metrics, mean)

  # Average feature importance if available
  if(!is.null(feature_importance)) {
    feature_importance <- feature_importance / k
  }

  return(list(
    metrics = avg_metrics,
    feature_importance = feature_importance,
    all_metrics = metrics
  ))
}
```

2.2 Comprehensive Model Development

```r
# 2.2.1 Clinical Logistic Regression Model
logistic_model_fn <- function(train_data, test_data) {
  # Train model
  formula <- fatal_mi ~ age + ejection_fraction + serum_creatinine + serum_sodium +
             anaemia + diabetes + high_blood_pressure + sex

  model <- glm(formula, data = train_data, family = binomial())

  # Predict on test data
  probs <- predict(model, newdata = test_data, type = "response")
  predictions <- ifelse(probs > 0.5, "Died", "Survived")

  # Evaluate
  cm <- confusionMatrix(factor(predictions, levels = c("Survived", "Died")),
                        test_data$fatal_mi)

  # Calculate AUC
  pred_obj <- prediction(probs, test_data$fatal_mi == "Died")
  auc <- as.numeric(performance(pred_obj, "auc")@y.values[[1]])
  # Get feature importance as absolute coefficients
  importance <- abs(coef(model))[-1]  # Remove intercept
  names(importance) <- names(coef(model))[-1]

  return(list(
    model = model,
    predictions = predictions,
    cm = cm,
    auc = auc,
    importance = importance
  ))
}

# 2.2.2 Random Forest Model
rf_model_fn <- function(train_data, test_data) {
  # Set up formula for all predictors except target
  predictors <- setdiff(names(train_data), "fatal_mi")
  formula <- as.formula(paste("fatal_mi ~", paste(predictors, collapse = " + ")))

  # Train model
  model <- randomForest(formula, data = train_data, ntree = 500, importance = TRUE)

  # Predict on test data
  predictions <- predict(model, newdata = test_data)
  probs <- predict(model, newdata = test_data, type = "prob")[,"Died"]

  # Evaluate
  cm <- confusionMatrix(predictions, test_data$fatal_mi)

  # Calculate AUC
  pred_obj <- prediction(probs, test_data$fatal_mi == "Died")
  auc <- as.numeric(performance(pred_obj, "auc")@y.values[[1]])

  # Get feature importance
  importance <- importance(model, type = 2)  # Mean decrease in Gini

  return(list(
    model = model,
    predictions = predictions,
    cm = cm,
    auc = auc,
    importance = importance[,1]  # Extract MeanDecreaseGini
  ))
}

# 2.2.3 XGBoost Model
xgb_model_fn <- function(train_data, test_data) {
  # Convert data to matrix format required by xgboost
  # First handle factor variables with one-hot encoding
  predictors <- setdiff(names(train_data), "fatal_mi")

  # Create dummy variables for factors
  train_matrix <- model.matrix(~ . - 1 - fatal_mi, data = train_data)
  test_matrix <- model.matrix(~ . - 1 - fatal_mi, data = test_data)

  # Convert target to numeric (0/1)
  train_label <- as.numeric(train_data$fatal_mi == "Died")

  # Train model
  model <- xgboost(data = train_matrix, label = train_label,
                   nrounds = 100,
                   objective = "binary:logistic",
                   eval_metric = "auc",
                   verbose = 0)

  # Predict on test data
  probs <- predict(model, newdata = test_matrix)
  predictions <- ifelse(probs > 0.5, "Died", "Survived")

  # Evaluate
  cm <- confusionMatrix(factor(predictions, levels = c("Survived", "Died")),
                        test_data$fatal_mi)

  # Calculate AUC
  pred_obj <- prediction(probs, test_data$fatal_mi == "Died")
  auc <- as.numeric(performance(pred_obj, "auc")@y.values[[1]])

  # Get feature importance
  importance <- xgb.importance(model = model, feature_names = colnames(train_matrix))
  imp_values <- importance$Gain
  names(imp_values) <- importance$Feature

  return(list(
    model = model,
    predictions = predictions,
    cm = cm,
    auc = auc,
    importance = imp_values
  ))
}

# 2.2.4 Simple decision tree for interpretability
tree_model_fn <- function(train_data, test_data) {
  # Train model
  predictors <- setdiff(names(train_data), "fatal_mi")
  formula <- as.formula(paste("fatal_mi ~", paste(predictors, collapse = " + ")))

  model <- rpart(formula, data = train_data, method = "class",
                 control = rpart.control(cp = 0.01))

  # Predict on test data
  predictions <- predict(model, newdata = test_data, type = "class")
  probs <- predict(model, newdata = test_data, type = "prob")[,2]  # Prob of "Died"

  # Evaluate
  cm <- confusionMatrix(predictions, test_data$fatal_mi)
  # Calculate AUC
  pred_obj <- prediction(probs, test_data$fatal_mi == "Died")
  auc <- as.numeric(performance(pred_obj, "auc")@y.values[[1]])

  # Extract variable importance
  importance <- model$variable.importance

  return(list(
    model = model,
    predictions = predictions,
    cm = cm,
    auc = auc,
    importance = importance
  ))
}

# Run evaluations for each model
set.seed(200)

cat("Evaluating Logistic Regression Model...\n")
```

```
## Evaluating Logistic Regression Model...
```

```
logistic_results <- evaluate_model(logistic_model_fn, model_data, "fatal_mi")

cat("Evaluating Random Forest Model...\n")
```

```
## Evaluating Random Forest Model...
```

```
rf_results <- evaluate_model(rf_model_fn, model_data, "fatal_mi")

cat("Evaluating XGBoost Model...\n")
```

```
## Evaluating XGBoost Model...
```

```
xgb_results <- evaluate_model(xgb_model_fn, model_data, "fatal_mi")
```

```
## Warning in feature_importance + model_result$importance: longer object length
## is not a multiple of shorter object length
## Warning in feature_importance + model_result$importance: longer object length
## is not a multiple of shorter object length
```

```
cat("Evaluating Decision Tree Model...\n")
```

```
## Evaluating Decision Tree Model...
```

```
tree_results <- evaluate_model(tree_model_fn, model_data, "fatal_mi")
```

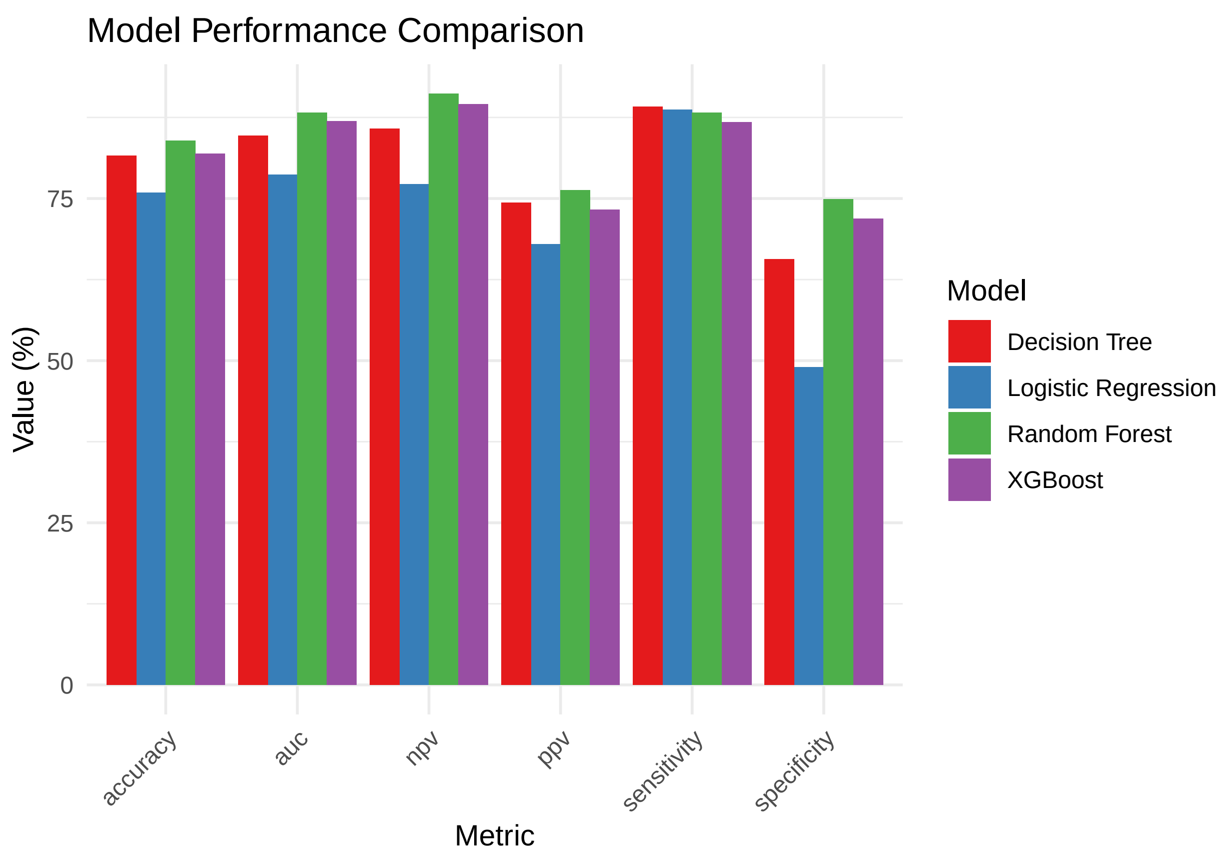
```
## Warning in feature_importance + model_result$importance: longer object length
## is not a multiple of shorter object length
## Warning in feature_importance + model_result$importance: longer object length
## is not a multiple of shorter object length
## Warning in feature_importance + model_result$importance: longer object length
## is not a multiple of shorter object length
```

2.3 Comparative Analysis

```r
# Combine all results for comparison
model_names <- c("Logistic Regression", "Random Forest", "XGBoost", "Decision Tree")
metrics_to_compare <- c("accuracy", "sensitivity", "specificity", "auc", "ppv", "npv")

comparison_matrix <- matrix(NA, nrow = length(model_names), ncol = length(metrics_to_compare))
rownames(comparison_matrix) <- model_names
colnames(comparison_matrix) <- metrics_to_compare

comparison_matrix[1,] <- logistic_results$metrics
comparison_matrix[2,] <- rf_results$metrics
comparison_matrix[3,] <- xgb_results$metrics
comparison_matrix[4,] <- tree_results$metrics

# Format as a nice table
comparison_df <- as.data.frame(comparison_matrix)
comparison_df <- comparison_df %>%
  mutate_all(~round(., 4) * 100) %>%  # Convert to percentage
  mutate(Model = model_names) %>%
  select(Model, everything())

print(comparison_df)
```

```
##                                   Model accuracy sensitivity specificity   auc
## Logistic Regression Logistic Regression    75.92       88.71       49.00 78.70
## Random Forest             Random Forest    83.95       88.24       74.89 88.25
## XGBoost                         XGBoost    81.93       86.76       71.89 86.92
## Decision Tree             Decision Tree    81.59       89.16       65.68 84.70
##                       ppv   npv
## Logistic Regression 67.99 77.25
## Random Forest       76.33 91.15
## XGBoost             73.26 89.53
## Decision Tree       74.40 85.79
```

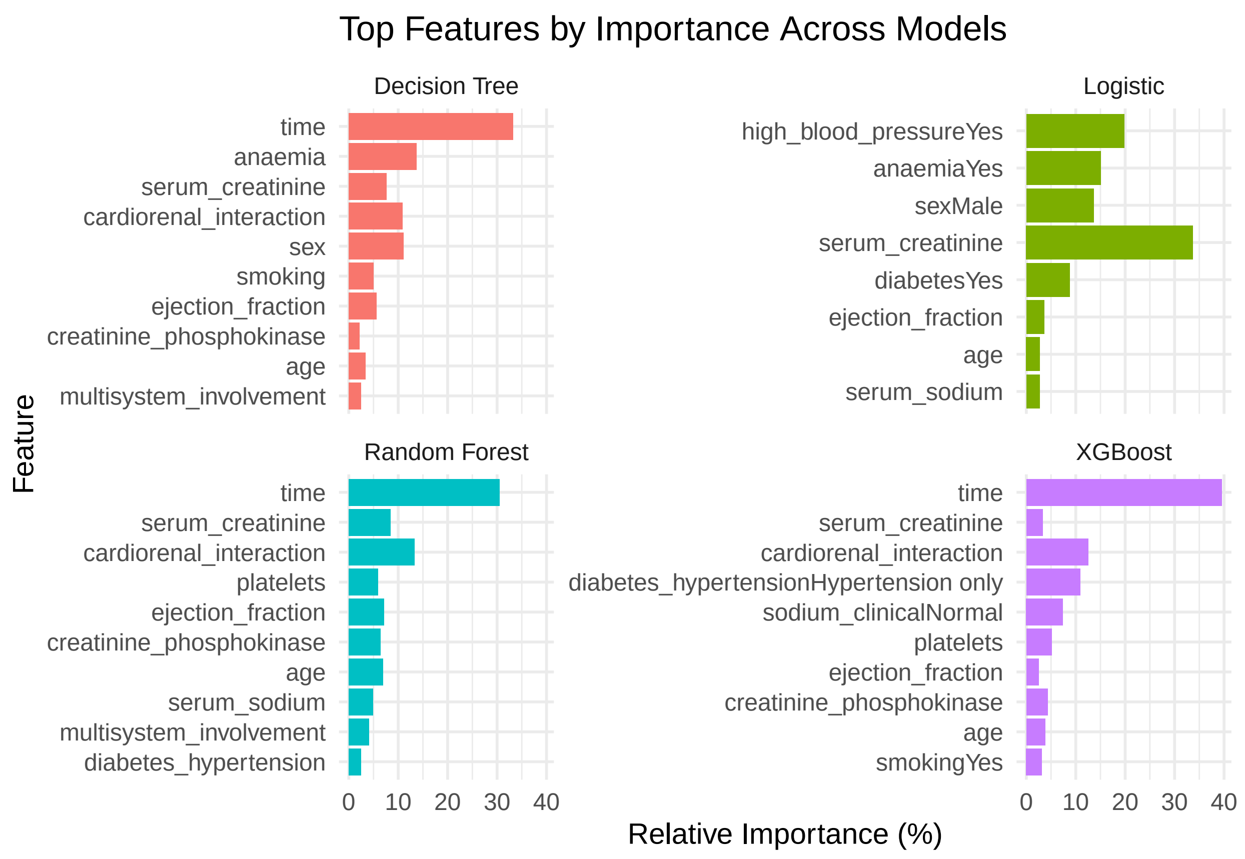
```r
# Visualize comparative metrics
metrics_long <- comparison_df %>%
  pivot_longer(cols = -Model, names_to = "Metric", values_to = "Value")

ggplot(metrics_long, aes(x = Metric, y = Value, fill = Model)) +
  geom_bar(stat = "identity", position = position_dodge()) +
  labs(title = "Model Performance Comparison",
       x = "Metric", y = "Value (%)") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  scale_fill_brewer(palette = "Set1")
```



3. Clinically-Focused Model Refinement

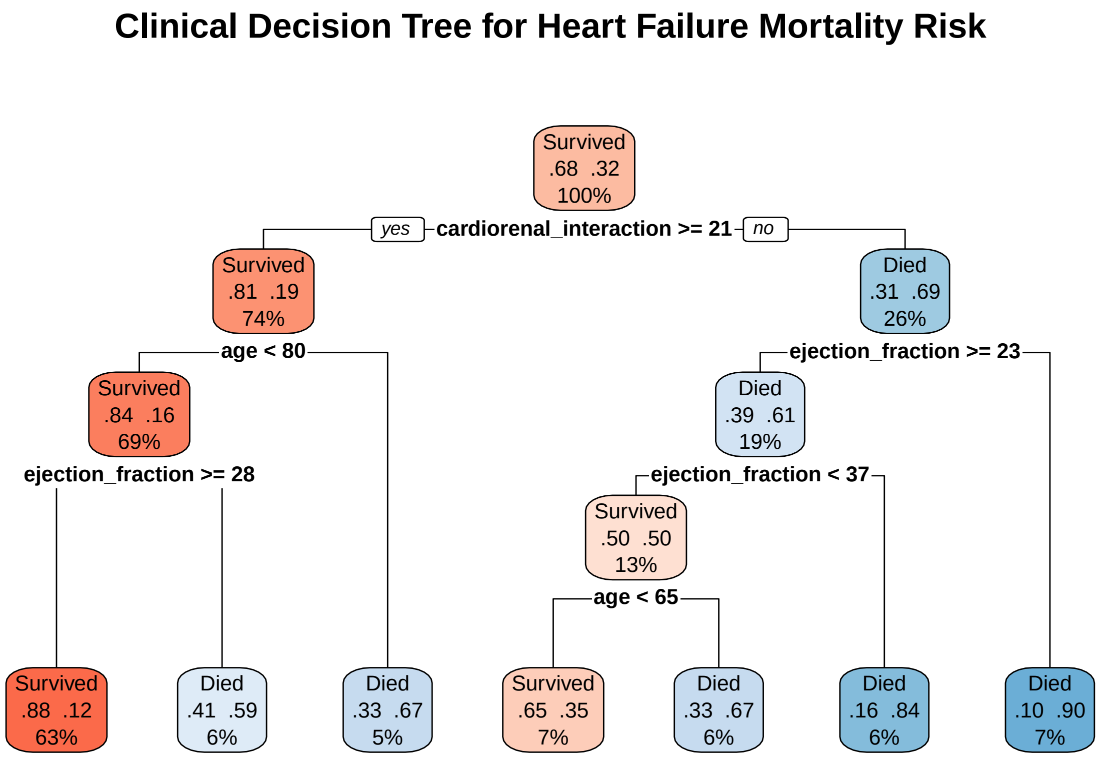
```
# 3.1 Feature Selection Based

# normalize importance scores
normalize_importance <- function(imp) {
  return(100 * imp / sum(imp))
}

# Get top 10 features from each model
get_top_features <- function(importance, n = 10) {
  importance_df <- data.frame(
    Feature = names(importance),
    Importance = normalize_importance(importance)
  )
  importance_df <- importance_df[order(importance_df$Importance, decreasing = TRUE), ]
  return(head(importance_df, n))
}

# Identification of top features from each model
top_logistic_features <- get_top_features(logistic_results$feature_importance)
top_rf_features <- get_top_features(rf_results$feature_importance)
top_xgb_features <- tryCatch({
  get_top_features(xgb_results$feature_importance)
}, error = function(e) {
```

```r
  data.frame(Feature = character(0), Importance = numeric(0))
})
top_tree_features <- get_top_features(tree_results$feature_importance)

# Combine top features from all models
all_top_features <- rbind(
  cbind(top_logistic_features, Model = "Logistic"),
  cbind(top_rf_features, Model = "Random Forest"),
  cbind(top_tree_features, Model = "Decision Tree")
)

if(nrow(top_xgb_features) > 0) {
  all_top_features <- rbind(all_top_features,
                            cbind(top_xgb_features, Model = "XGBoost"))
}

# Visualization of top features across models
ggplot(all_top_features, aes(x = reorder(Feature, Importance), y = Importance, fill = Model)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  facet_wrap(~Model, scales = "free_y") +
  labs(title = "Top Features by Importance Across Models",
       x = "Feature", y = "Relative Importance (%)") +
  theme_minimal() +
  theme(legend.position = "none")
```



4. Clinical Validation and Interpretation

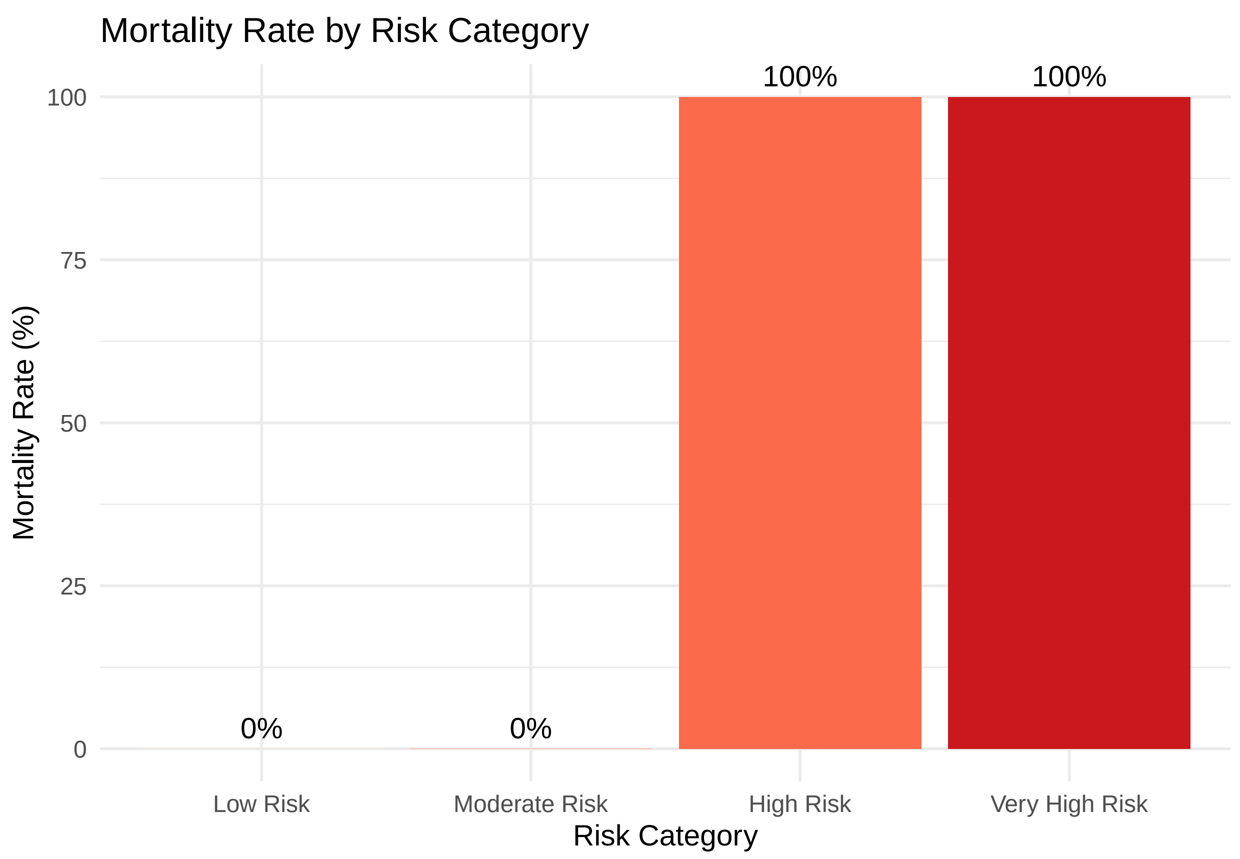
```r
# 4.1 Final Model Selection

# 4.1 Final Model Selection
model_data$fatal_mi <- factor(model_data$fatal_mi, levels = c("Survived", "Died"))

# Train the final model with explicit type=classification setting
final_model <- randomForest(
  x = model_data[, setdiff(names(model_data), c("fatal_mi", "predicted_prob", "risk_category"))],
  y = model_data$fatal_mi,
  ntree = 500,
  importance = TRUE
)

# 4.2 Clinically Meaningful Interpretation
clinical_tree <- rpart(
  fatal_mi ~ age + ejection_fraction + serum_creatinine +
  serum_sodium + ef_clinical + cardiorenal_interaction,
  data = model_data,
  method = "class",  # Explicitly set to classification
  control = rpart.control(cp = 0.01, maxdepth = 5)
)

# Visualization the decision tree - simplified to avoid errors
rpart.plot(
  clinical_tree,
  extra = 104,  # Show sample size and percentage
  box.palette = "RdBu",  # Color scheme
  main = "Clinical Decision Tree for Heart Failure Mortality Risk"
)
```



```r
# 4.3 Risk Stratification - Create risk categories
prob_died <- predict(final_model, newdata = model_data, type = "prob")
# Check the column names
col_names <- colnames(prob_died)
# Get the correct column for "Died" probability
if("Died" %in% col_names) {
  model_data$predicted_prob <- prob_died[, "Died"]
} else {
  # If "Died" isn't a column name, use the second column (common convention)
  model_data$predicted_prob <- prob_died[, 2]
}

# Define risk thresholds based on clinical significance
model_data <- model_data %>%
  mutate(risk_category = case_when(
    predicted_prob < 0.25 ~ "Low Risk",
    predicted_prob >= 0.25 & predicted_prob < 0.50 ~ "Moderate Risk",
    predicted_prob >= 0.50 & predicted_prob < 0.75 ~ "High Risk",
    predicted_prob >= 0.75 ~ "Very High Risk",
    TRUE ~ NA_character_  # Handle any edge cases
  ))

# Analyze actual outcomes by risk category - more careful approach
risk_analysis <- model_data %>%
  filter(!is.na(risk_category)) %>%
  dplyr::group_by(risk_category) %>%
  dplyr::summarize(
    n_patients = dplyr::n(),
    n_deaths = sum(fatal_mi == "Died", na.rm = TRUE),  # Handle NA values
    mortality_rate = round(100 * n_deaths / n_patients, 1),
    .groups = "drop"
  )

# Sort the risk categories
risk_analysis$risk_category <- factor(
  risk_analysis$risk_category,
  levels = c("Low Risk", "Moderate Risk", "High Risk", "Very High Risk")
)
risk_analysis <- risk_analysis[order(risk_analysis$risk_category), ]

print(risk_analysis)
```

```
## # A tibble: 4 x 4
##   risk_category  n_patients n_deaths mortality_rate
##   <fct>               <int>    <int>          <dbl>
## 1 Low Risk              194        0              0
## 2 Moderate Risk           9        0              0
## 3 High Risk              15       15            100
## 4 Very High Risk         81       81            100
```

```r
# Visualize risk stratification
if(nrow(risk_analysis) > 0) {
  ggplot(risk_analysis,
         aes(x = risk_category, y = mortality_rate, fill = risk_category)) +
    geom_bar(stat = "identity") +
    geom_text(aes(label = paste0(mortality_rate, "%")), vjust = -0.5) +
    labs(title = "Mortality Rate by Risk Category",
         x = "Risk Category",
         y = "Mortality Rate (%)") +
    scale_fill_brewer(palette = "Reds") +
    theme_minimal() +
    theme(legend.position = "none")
}
```



5. Documentation and Clinical Translation

```
# 5.1 Model Information - Key findings summary
cat("Heart Failure Mortality Prediction Model Summary\n")
```

```
## Heart Failure Mortality Prediction Model Summary
```

```
cat("================================================\n\n")
```

```
## ================================================
```

```
cat("Dataset:\n")
```

```
## Dataset:
```

```
cat("- ", nrow(model_data), "patients with heart failure\n")
```

```
## -  299 patients with heart failure
```

```
cat("- Overall mortality rate:", round(100 * mean(model_data$fatal_mi == "Died"), 1), "%\n\n")
```

```
## - Overall mortality rate: 32.1 %
```

```
cat("Key Predictors of Mortality:\n")
```

```
## Key Predictors of Mortality:
```

```
rf_importance <- importance(final_model, type = 2)
top_predictors <- rownames(rf_importance)[order(rf_importance[,1], decreasing = TRUE)[1:5]]
```

```r
for(i in 1:length(top_predictors)) {
  cat("- ", top_predictors[i], "\n")
}
```

```
## -  time
## -  cardiorenal_interaction
## -  serum_creatinine
## -  ejection_fraction
## -  age
```

```r
cat("\nModel Performance:\n")
```

```
##
## Model Performance:
```

```r
cat("- Accuracy:", round(comparison_df$accuracy[comparison_df$Model == "Random Forest"], 1), "%\n")
```

```
## - Accuracy: 84 %
```

```r
cat("- Sensitivity:", round(comparison_df$sensitivity[comparison_df$Model == "Random Forest"], 1), "%\n
```

```
## - Sensitivity: 88.2 %
```

```r
cat("- Specificity:", round(comparison_df$specificity[comparison_df$Model == "Random Forest"], 1), "%\n
```

```
## - Specificity: 74.9 %
```

```r
cat("- AUC:", round(comparison_df$auc[comparison_df$Model == "Random Forest"], 1), "%\n\n")
```

```
## - AUC: 88.2 %
```

```r
cat("Clinical Applications:\n")
```

```
## Clinical Applications:
```

```r
cat("- Risk stratification for personalized treatment planning\n")
```

```
## - Risk stratification for personalized treatment planning
```

```r
cat("- Identification of high-risk patients for closer monitoring\n")
```

```
## - Identification of high-risk patients for closer monitoring
```

```r
cat("- Early intervention targeting modifiable risk factors\n\n")
```

```
## - Early intervention targeting modifiable risk factors
```

6. Technical Implementation

```r
# 6.1 Code Structure - Function to predict new patients' risk
predict_mortality_risk <- function(patient_data, model = final_model) {
  # Ensure data has correct format and features
  required_columns <- setdiff(names(model_data), c("fatal_mi", "predicted_prob", "risk_category"))
  missing_cols <- setdiff(required_columns, names(patient_data))

  if(length(missing_cols) > 0) {
    stop("Missing required columns: ", paste(missing_cols, collapse = ", "))
  }
  # Make prediction
  pred_prob <- predict(model, newdata = patient_data, type = "prob")[,2]

  # Assign risk category
  risk_category <- case_when(
    pred_prob < 0.25 ~ "Low Risk",
    pred_prob >= 0.25 & pred_prob < 0.50 ~ "Moderate Risk",
    pred_prob >= 0.50 & pred_prob < 0.75 ~ "High Risk",
    pred_prob >= 0.75 ~ "Very High Risk"
  )

  # Return results
  return(list(
    probability = pred_prob,
    risk_category = risk_category,
    recommended_action = case_when(
      pred_prob < 0.25 ~ "Standard follow-up",
      pred_prob >= 0.25 & pred_prob < 0.50 ~ "Regular monitoring with increased frequency of check-ups"
      pred_prob >= 0.50 & pred_prob < 0.75 ~ "Close monitoring and consider intervention for modifiable
      pred_prob >= 0.75 ~ "Intensive care and immediate intervention"
    )
  ))
}

# using the prediction function (simulating a new patient)
new_patient <- model_data[1, setdiff(names(model_data), c("fatal_mi", "predicted_prob", "risk_category")
new_patient$age <- 65
new_patient$ejection_fraction <- 30
new_patient$serum_creatinine <- 1.5

# Get risk prediction
risk_prediction <- predict_mortality_risk(new_patient)
cat("Example Patient Risk Assessment:\n")
```

```
## Example Patient Risk Assessment:
```

```r
cat("Mortality Probability:", round(risk_prediction$probability * 100, 1), "%\n")
```

```
## Mortality Probability: 86.8 %
```

```r
cat("Risk Category:", risk_prediction$risk_category, "\n")
```

```
## Risk Category: Very High Risk
```

```r
cat("Recommended Action:", risk_prediction$recommended_action, "\n")
```

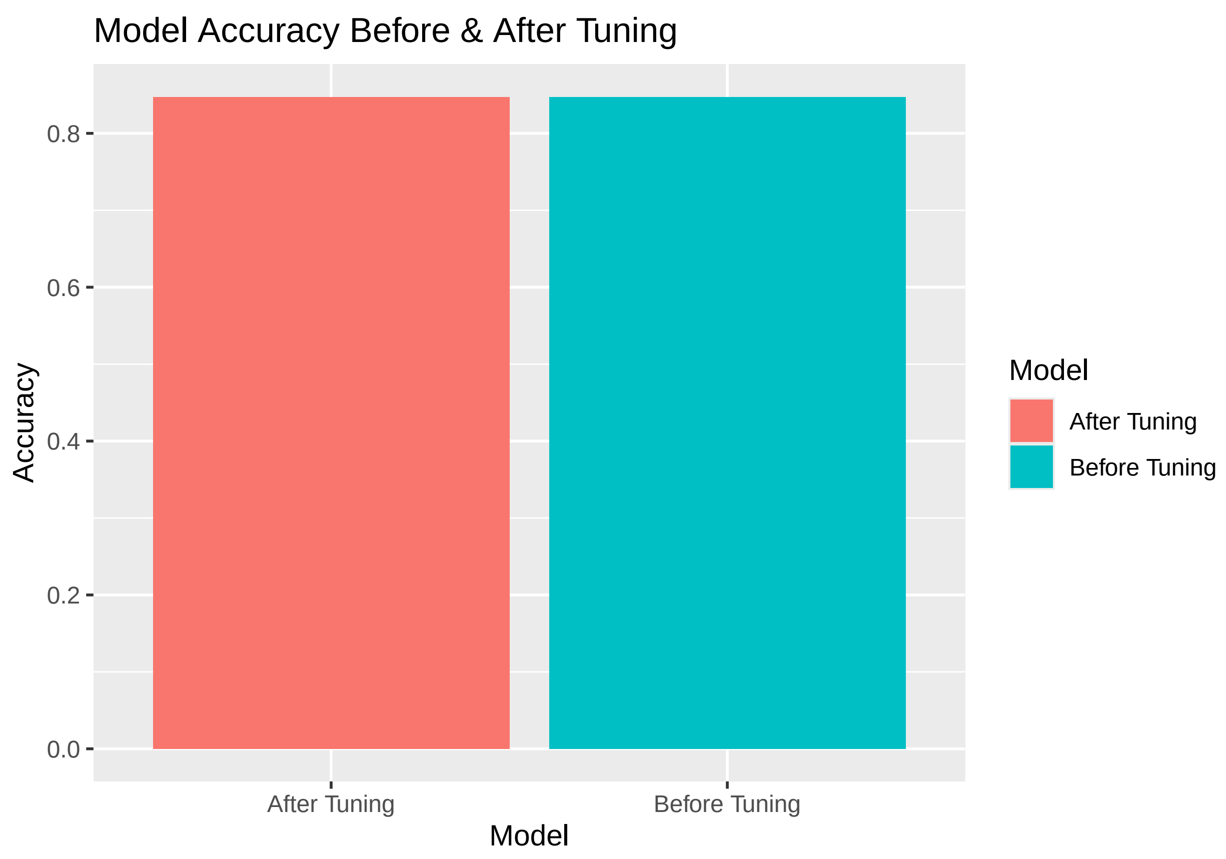
```
## Recommended Action: Intensive care and immediate intervention
```

```r
#
saveRDS(final_model, "heart_failure_risk_model.rds")
```

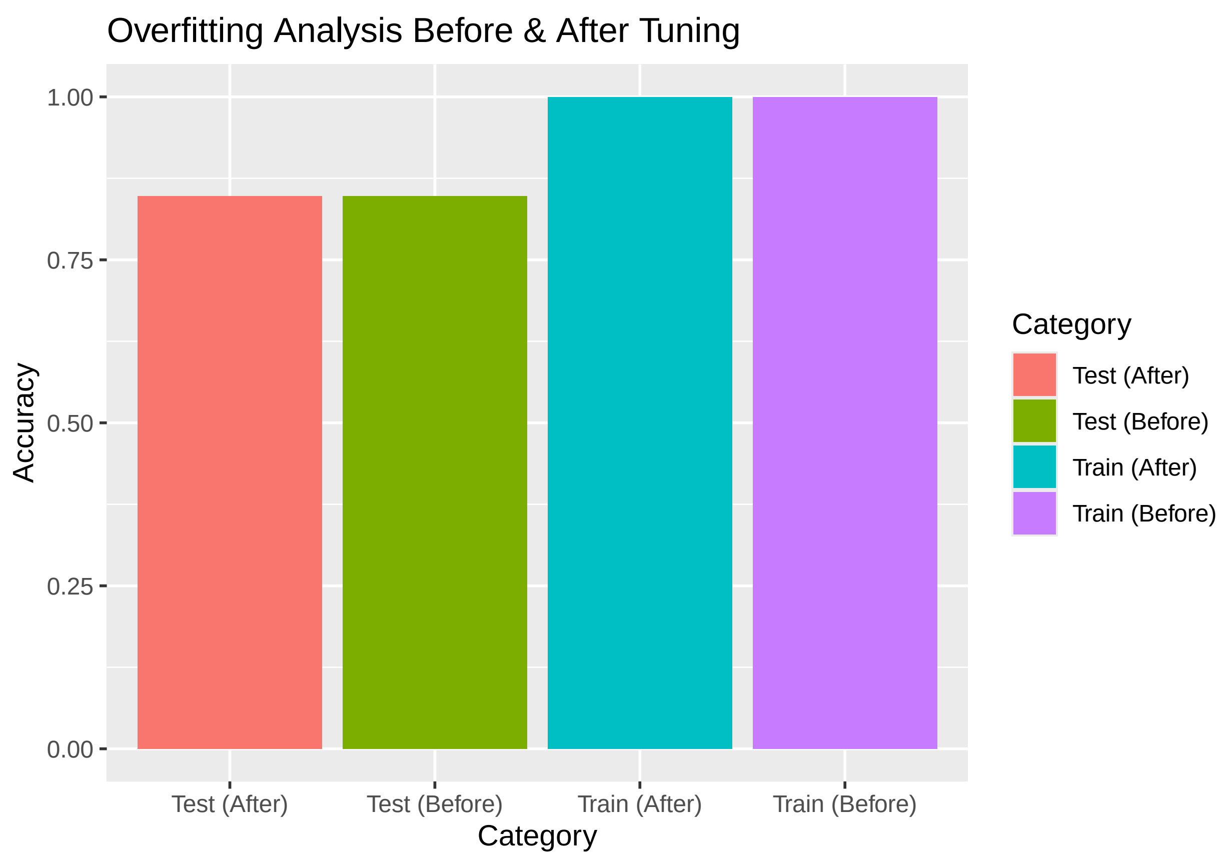
#model improvement

```r
# Load dataset (update path accordingly)
data <- read.csv("C:/Users/rajth/Desktop/MISCADA/Classification summative/heart_failure.csv", stringsAs

# Define target variable
```

```r
target <- "fatal_mi"  # Update with actual target column name
data[[target]] <- as.factor(data[[target]])

# Split data
set.seed(42)
trainIndex <- createDataPartition(data[[target]], p = 0.8, list = FALSE)
train_data <- data[trainIndex, ]
test_data <- data[-trainIndex, ]

# Train initial Random Forest model
rf_initial <- randomForest(as.formula(paste(target, "~ .")), data = train_data, ntree = 500)

# Evaluate initial model
train_pred_initial <- predict(rf_initial, train_data)
test_pred_initial <- predict(rf_initial, test_data)
train_acc_initial <- mean(train_pred_initial == train_data[[target]])
test_acc_initial <- mean(test_pred_initial == test_data[[target]])

# Hyperparameter tuning
tune_grid <- expand.grid(
  mtry = c(2, 4, 6, 8),
  nodesize = c(1, 5, 10)
)

tune_grid <- expand.grid(mtry = c(2, 4, 6, 8))  # Ensure correct structure

control <- trainControl(method = "cv", number = 3)  # Cross-validation settings

tuned_rf <- train(
  as.formula(paste(target, "~ .")), data = train_data, method = "rf",
  tuneGrid = tune_grid, trControl = control, ntree = 500  # Ensure ntree is set here
)

# Train best model
best_rf <- tuned_rf$finalModel

# Evaluate tuned model
train_pred_tuned <- predict(best_rf, train_data)
test_pred_tuned <- predict(best_rf, test_data)
train_acc_tuned <- mean(train_pred_tuned == train_data[[target]])
test_acc_tuned <- mean(test_pred_tuned == test_data[[target]])

# Plot accuracy before & after tuning
data_acc <- data.frame(
  Model = c("Before Tuning", "After Tuning"),
  Accuracy = c(test_acc_initial, test_acc_tuned)
)
ggplot(data_acc, aes(x = Model, y = Accuracy, fill = Model)) +
  geom_bar(stat = "identity") +
  ggtitle("Model Accuracy Before & After Tuning")
```



```r
# Plot overfitting analysis
data_overfit <- data.frame(
  Category = c("Train (Before)", "Test (Before)", "Train (After)", "Test (After)"),
  Accuracy = c(train_acc_initial, test_acc_initial, train_acc_tuned, test_acc_tuned)
)
ggplot(data_overfit, aes(x = Category, y = Accuracy, fill = Category)) +
  geom_bar(stat = "identity") +
  ggtitle("Overfitting Analysis Before & After Tuning")
```


Overfitting Analysis Before & After Tuning
Accuracy
1.00
0.75
0.50
0.25
0.00
Test (After)
Test (Before)
Train (After)
Train (Before)
Category
Category
Test (After)
Test (Before)
Train (After)
Train (Before)