Enhancing Cirrhosis Patient Survival Prediction: A Comparative Analysis of Classification Algorithms and Feature Engineering Techniques

DA5030

Anusha Devi Doddi

Spring 2024

Data Acquisition

Load the Data from CSV File

```
# Load the data from the CSV file
cirrhosis <- read.csv("data/cirrhosis.csv")

# Display the first few rows of the data
head(cirrhosis)
```

```
##   ID N_Days Status            Drug   Age Sex Ascites Hepatomegaly Spiders Edema
## 1  1    400      D D-penicillamine 21464   F       Y             Y       Y     Y
## 2  2   4500      C D-penicillamine 20617   F       N             Y       Y     N
## 3  3   1012      D D-penicillamine 25594   M       N             N       N     S
## 4  4   1925      D D-penicillamine 19994   F       N             Y       Y     S
## 5  5   1504     CL          Placebo 13918   F       N             Y       Y     N
## 6  6   2503      D          Placebo 24201   F       N             Y       N     N
##   Bilirubin Cholesterol Albumin Copper Alk_Phos   SGOT Tryglicerides Platelets
## 1      14.5         261    2.60    156   1718.0 137.95           172       190
## 2       1.1         302    4.14     54   7394.8 113.52            88       221
## 3       1.4         176    3.48    210    516.0  96.10            55       151
## 4       1.8         244    2.54     64   6121.8  60.63            92       183
## 5       3.4         279    3.53    143    671.0 113.15            72       136
## 6       0.8         248    3.98     50    944.0  93.00            63        NA
##   Prothrombin Stage
## 1        12.2     4
## 2        10.6     3
## 3        12.0     4
## 4        10.3     4
## 5        10.9     3
## 6        11.0     3
```

- The data is about patients with cirrhosis of the liver. It is sourced from a Mayo Clinic study on primary biliary cirrhosis of the liver.
- It is loaded into a data frame named `cirrhosis`. It is a csv file with 418 rows and 20 columns.

Data Exploration

Summary and Structure of the Data

```
# Summary of the data
summary(cirrhosis)
```

```
##        ID            N_Days        Status              Drug
##  Min.   :  1.0   Min.   :  41   Length:418         Length:418
##  1st Qu.:105.2   1st Qu.:1093   Class :character   Class :character
##  Median :209.5   Median :1730   Mode  :character   Mode  :character
##  Mean   :209.5   Mean   :1918
##  3rd Qu.:313.8   3rd Qu.:2614
##  Max.   :418.0   Max.   :4795
##
##       Age            Sex               Ascites          Hepatomegaly
##  Min.   : 9598   Length:418         Length:418         Length:418
##  1st Qu.:15644   Class :character   Class :character   Class :character
##  Median :18628   Mode  :character   Mode  :character   Mode  :character
##  Mean   :18533
##  3rd Qu.:21272
##  Max.   :28650
##
##    Spiders             Edema             Bilirubin       Cholesterol
##  Length:418         Length:418         Min.   : 0.300   Min.   : 120.0
##  Class :character   Class :character   1st Qu.: 0.800   1st Qu.: 249.5
##  Mode  :character   Mode  :character   Median : 1.400   Median : 309.5
##                                        Mean   : 3.221   Mean   : 369.5
##                                        3rd Qu.: 3.400   3rd Qu.: 400.0
##                                        Max.   :28.000   Max.   :1775.0
##                                                         NA's   :134
##     Albumin          Copper          Alk_Phos            SGOT
##  Min.   :1.960   Min.   :  4.00   Min.   :  289.0   Min.   : 26.35
##  1st Qu.:3.243   1st Qu.: 41.25   1st Qu.:  871.5   1st Qu.: 80.60
##  Median :3.530   Median : 73.00   Median : 1259.0   Median :114.70
##  Mean   :3.497   Mean   : 97.65   Mean   : 1982.7   Mean   :122.56
##  3rd Qu.:3.770   3rd Qu.:123.00   3rd Qu.: 1980.0   3rd Qu.:151.90
##  Max.   :4.640   Max.   :588.00   Max.   :13862.4   Max.   :457.25
##                  NA's   :108      NA's   :106       NA's   :106
##  Tryglicerides      Platelets      Prothrombin        Stage
##  Min.   : 33.00   Min.   : 62.0   Min.   : 9.00   Min.   :1.000
##  1st Qu.: 84.25   1st Qu.:188.5   1st Qu.:10.00   1st Qu.:2.000
##  Median :108.00   Median :251.0   Median :10.60   Median :3.000
##  Mean   :124.70   Mean   :257.0   Mean   :10.73   Mean   :3.024
##  3rd Qu.:151.00   3rd Qu.:318.0   3rd Qu.:11.10   3rd Qu.:4.000
##  Max.   :598.00   Max.   :721.0   Max.   :18.00   Max.   :4.000
##  NA's   :136      NA's   :11      NA's   :2       NA's   :6
```

```
# Structure of the data
str(cirrhosis)
```

```
## 'data.frame':    418 obs. of  20 variables:
```

```
##  $ ID           : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ N_Days       : int  400 4500 1012 1925 1504 2503 1832 2466 2400 51 ...
##  $ Status       : chr  "D" "C" "D" "D" ...
##  $ Drug         : chr  "D-penicillamine" "D-penicillamine" "D-penicillamine" "D-penicillamine" ...
##  $ Age          : int  21464 20617 25594 19994 13918 24201 20284 19379 15526 25772 ...
##  $ Sex          : chr  "F" "F" "M" "F" ...
##  $ Ascites      : chr  "Y" "N" "N" "N" ...
##  $ Hepatomegaly : chr  "Y" "Y" "N" "Y" ...
##  $ Spiders      : chr  "Y" "Y" "N" "Y" ...
##  $ Edema        : chr  "Y" "N" "S" "S" ...
##  $ Bilirubin    : num  14.5 1.1 1.4 1.8 3.4 0.8 1 0.3 3.2 12.6 ...
##  $ Cholesterol  : int  261 302 176 244 279 248 322 280 562 200 ...
##  $ Albumin      : num  2.6 4.14 3.48 2.54 3.53 3.98 4.09 4 3.08 2.74 ...
##  $ Copper       : int  156 54 210 64 143 50 52 52 79 140 ...
##  $ Alk_Phos     : num  1718 7395 516 6122 671 ...
##  $ SGOT         : num  137.9 113.5 96.1 60.6 113.2 ...
##  $ Tryglicerides: int  172 88 55 92 72 63 213 189 88 143 ...
##  $ Platelets    : int  190 221 151 183 136 NA 204 373 251 302 ...
##  $ Prothrombin  : num  12.2 10.6 12 10.3 10.9 11 9.7 11 11 11.5 ...
##  $ Stage        : int  4 3 4 4 3 3 3 3 2 4 ...
```

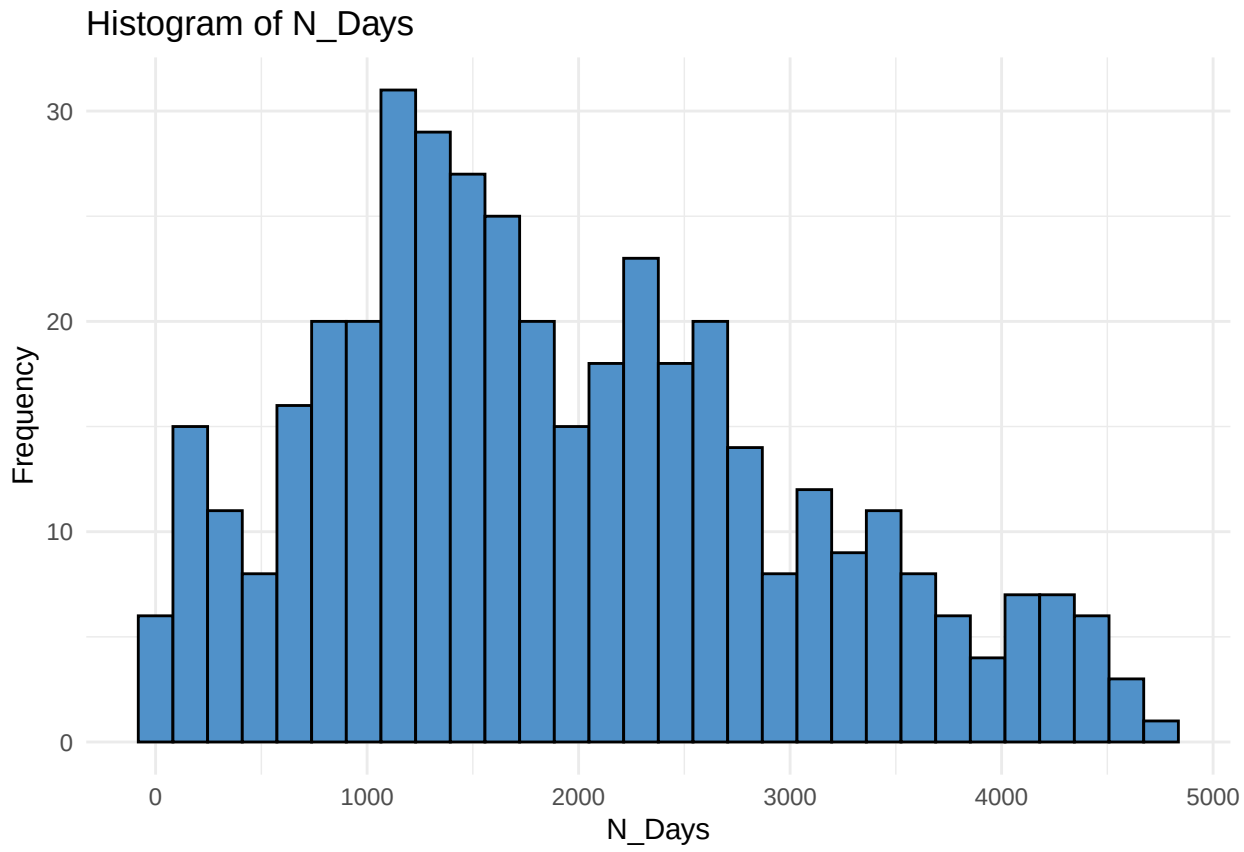
```r
# Remove the 'ID' column as it is not needed for analysis
cirrhosis <- cirrhosis[, -1]
```

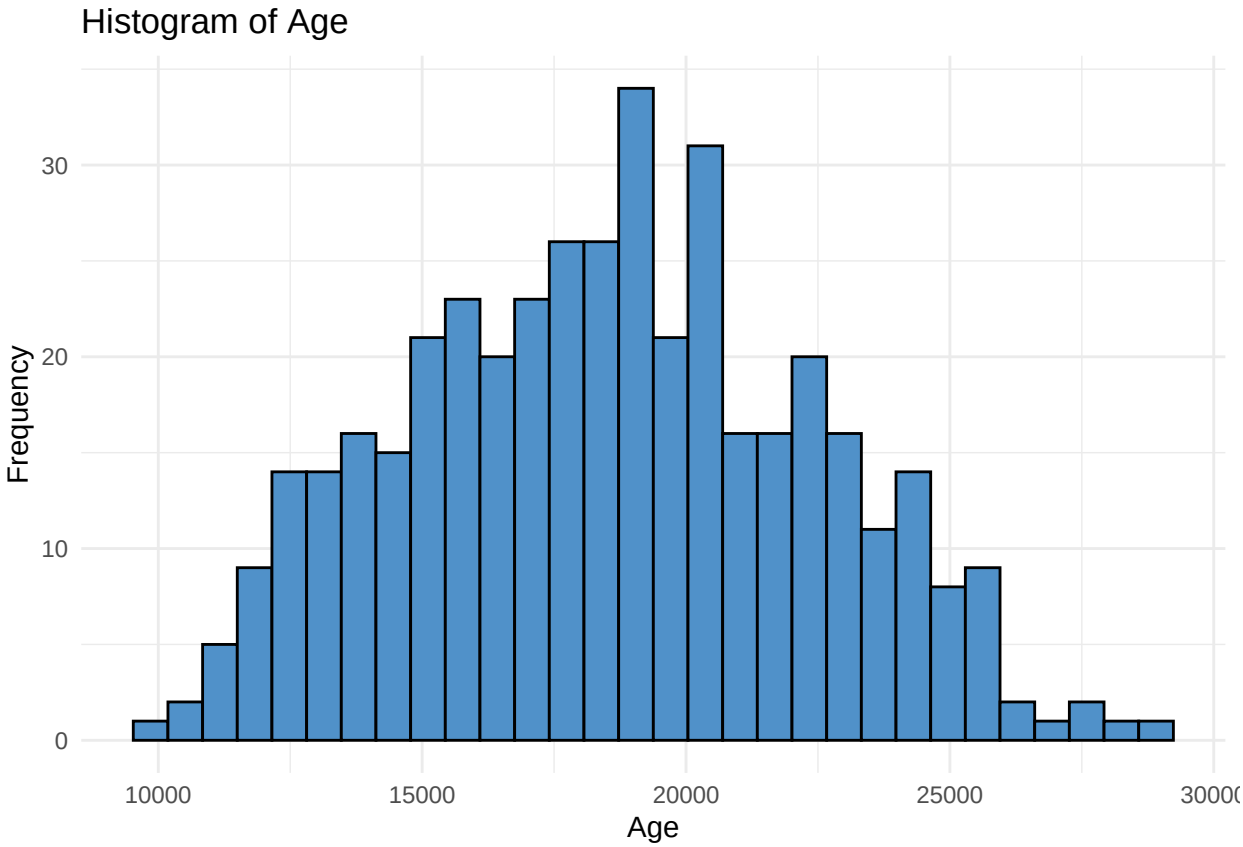
- The dataset contains 418 observations and 19 variables.
- The data has a mix of numeric and categorical variables.
- The 'Status' column is the target variable, indicating the survival status of the patients.
- The 'ID' column is removed as it is not relevant for analysis.

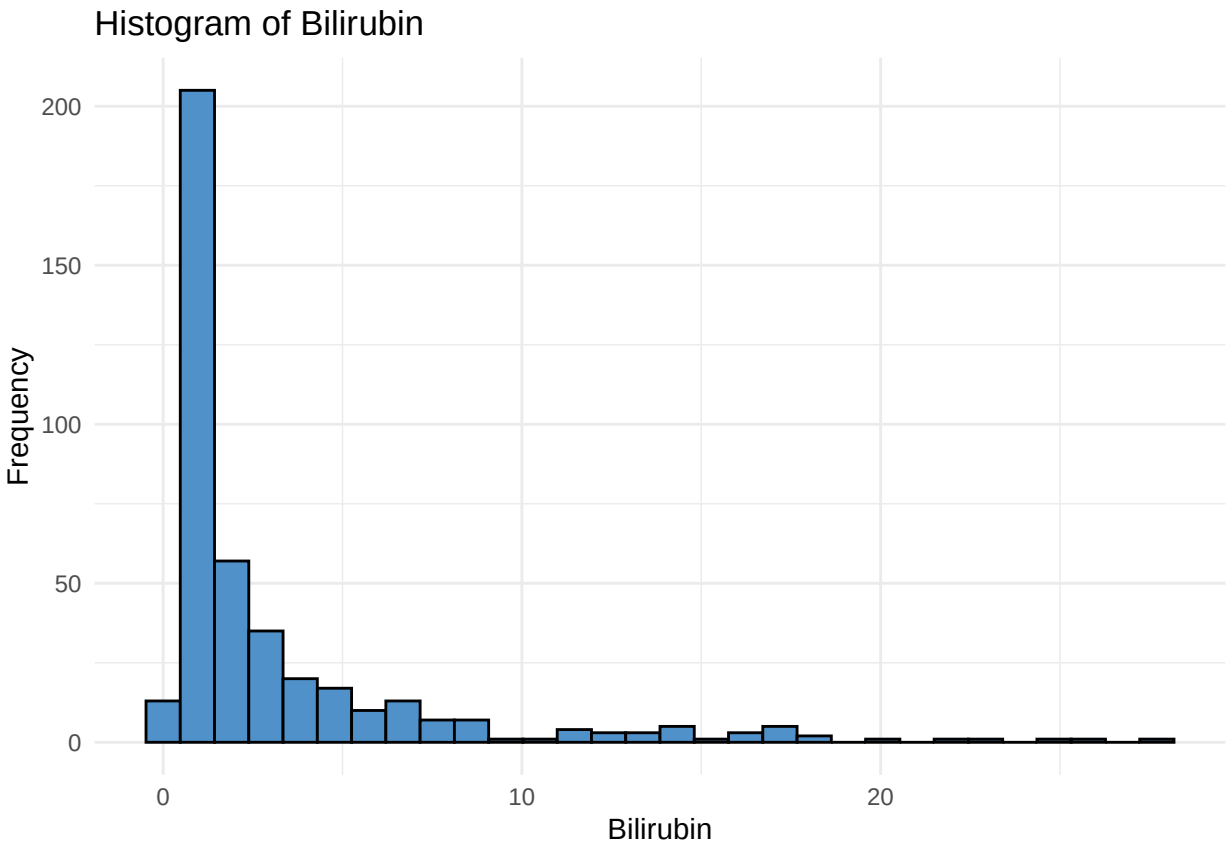
Data Visualization using Histograms

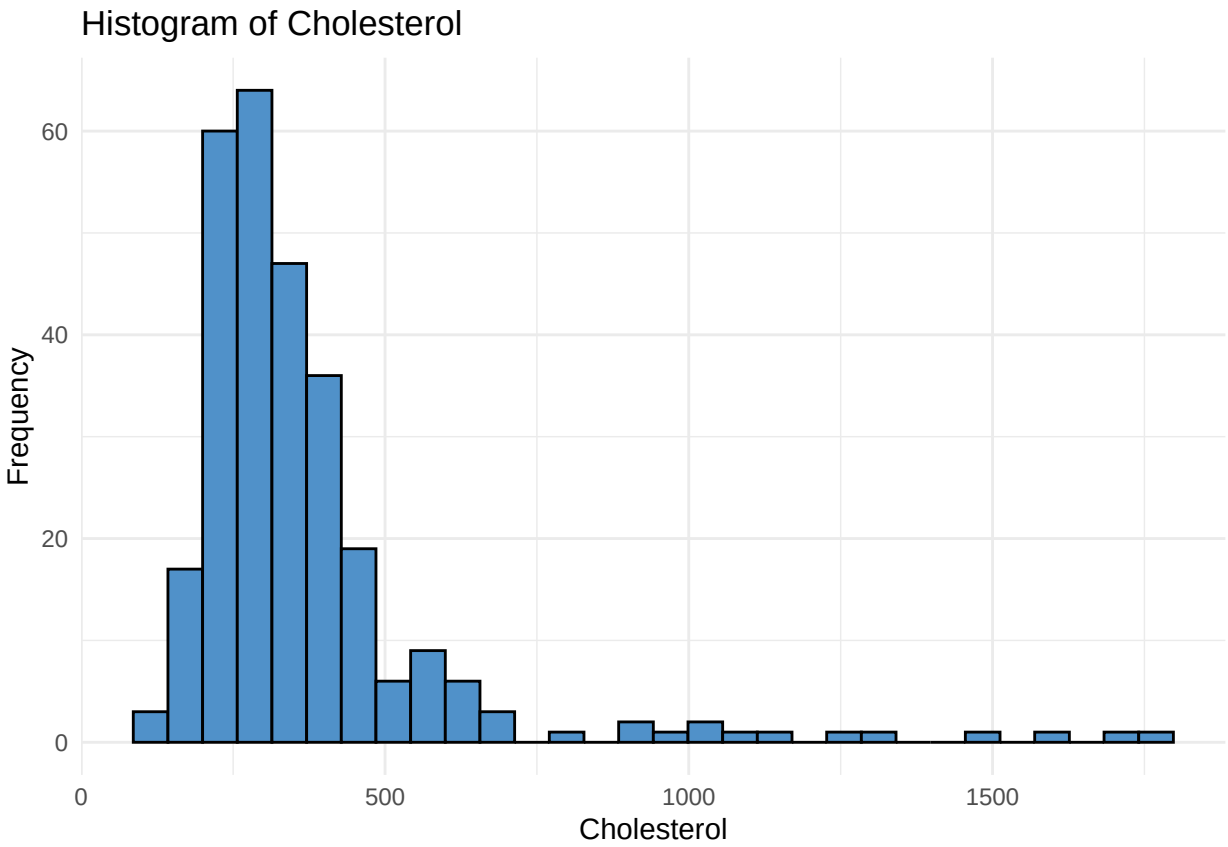
```r
# Load the necessary libraries
library(ggplot2)
library(dplyr)

# Create a function to plot histogram for a given column
numeric_columns <- sapply(cirrhosis, is.numeric)
# Extract numeric columns
numeric_data <- cirrhosis[, numeric_columns]

# Create a function to plot histogram for a given column
plot_histogram <- function(data, column_name) {
  # Create a histogram plot
  plot <- ggplot(data, aes_string(x = column_name)) +
    # Add histogram with blue fill and black border
    geom_histogram(fill = "#5091c9", color = "black") +
    # Add labels and title
    labs(title = paste("Histogram of", column_name),
    x = column_name, y = "Frequency") +
    theme_minimal()
```

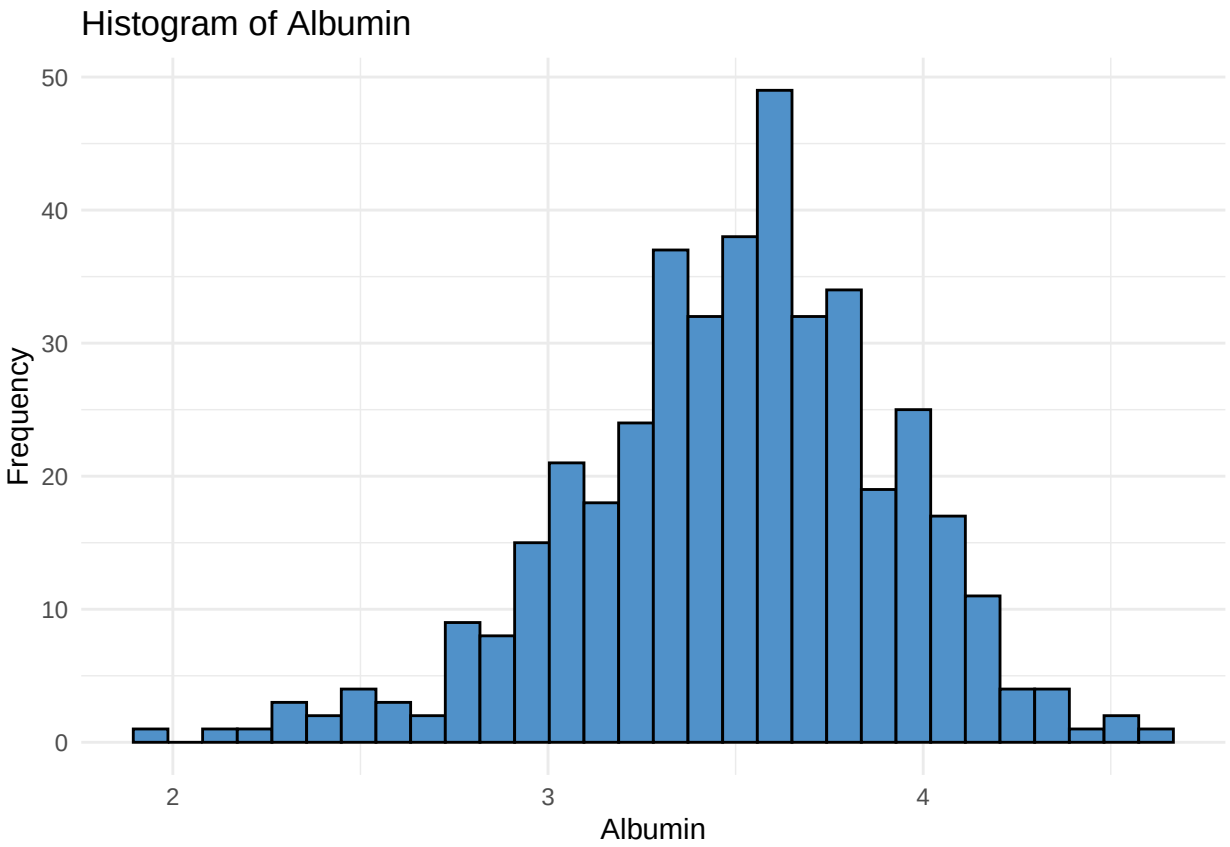
```r
  # Print the plot
  print(plot)
}

# Apply the function to each numeric column
invisible(lapply(names(numeric_data), function(col) {
  plot_histogram(numeric_data, col)
}))
```

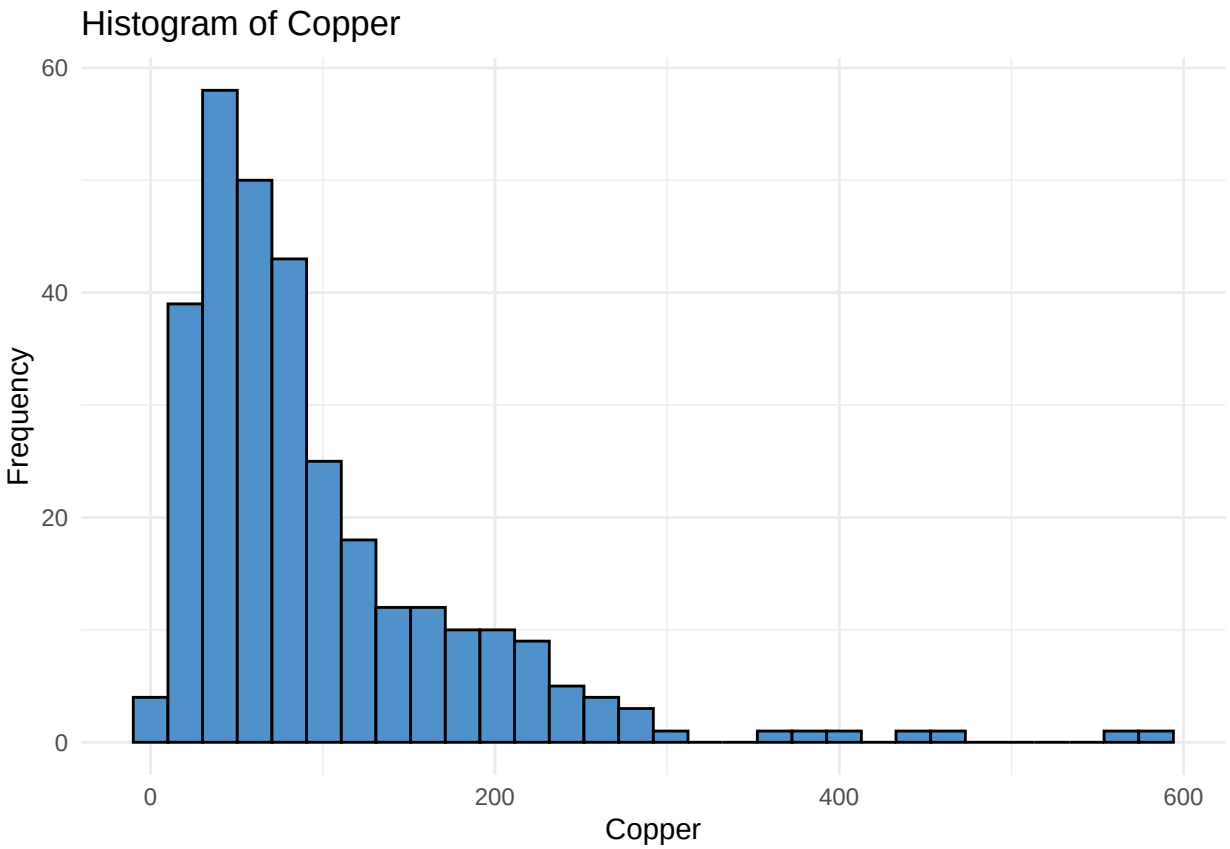


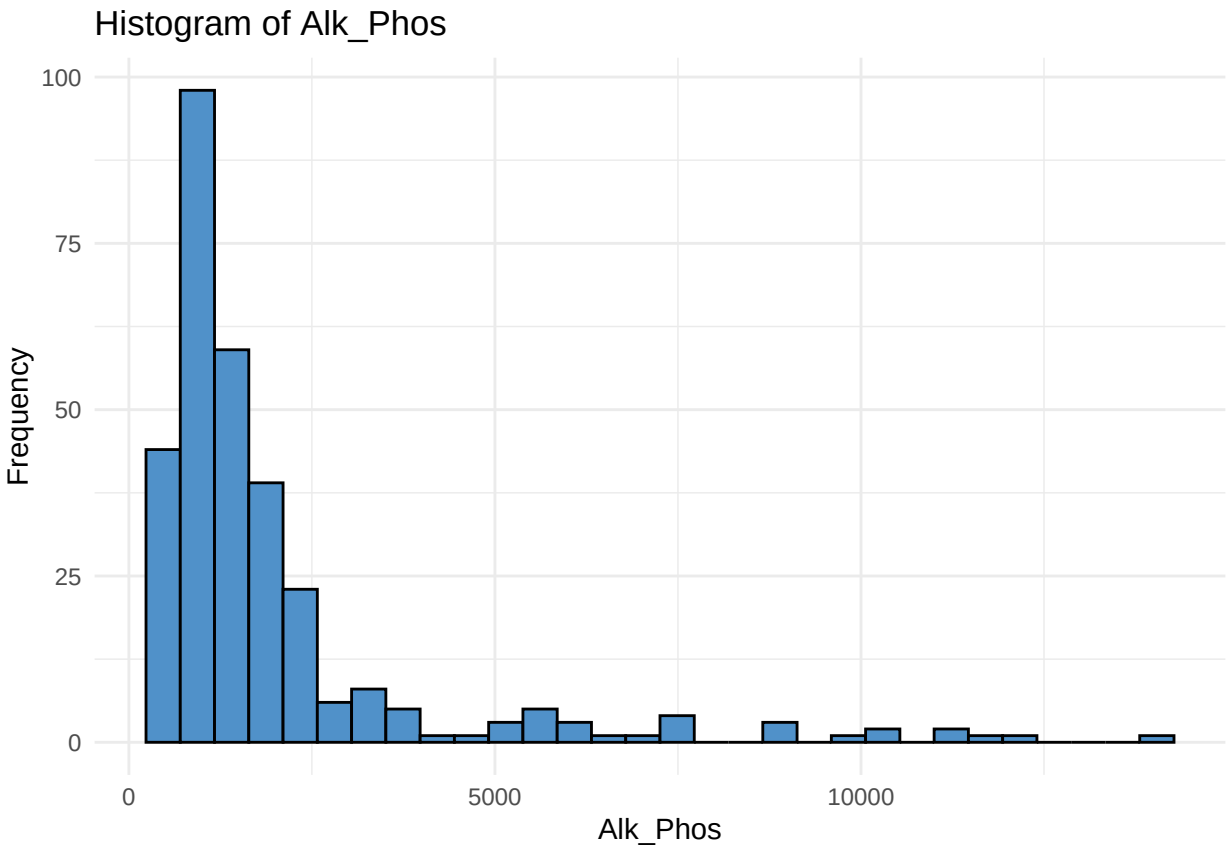

Histogram of Age
Frequency
30
20
10
0
10000
15000
20000
25000
30000
Age

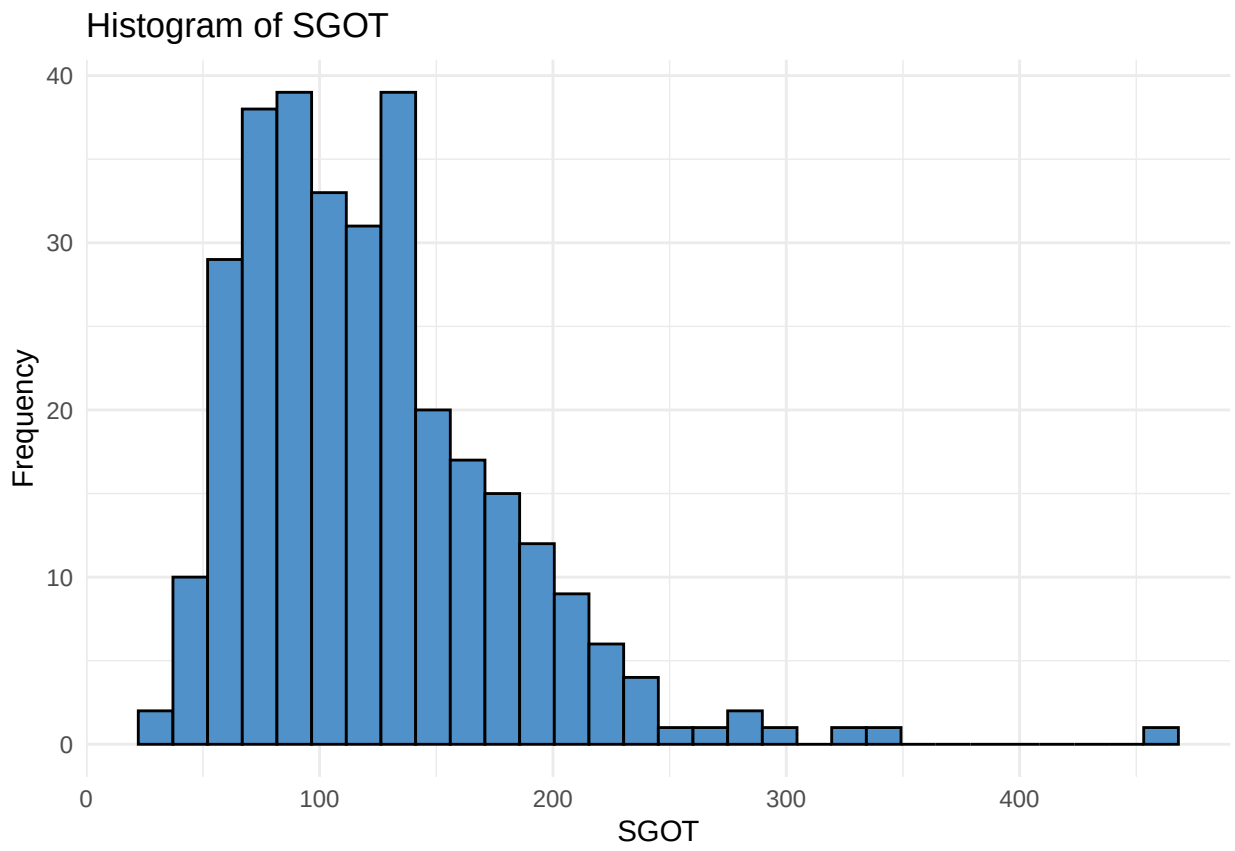

Histogram of Bilirubin
Frequency
200
150
100
50
0
0
10
20
Bilirubin

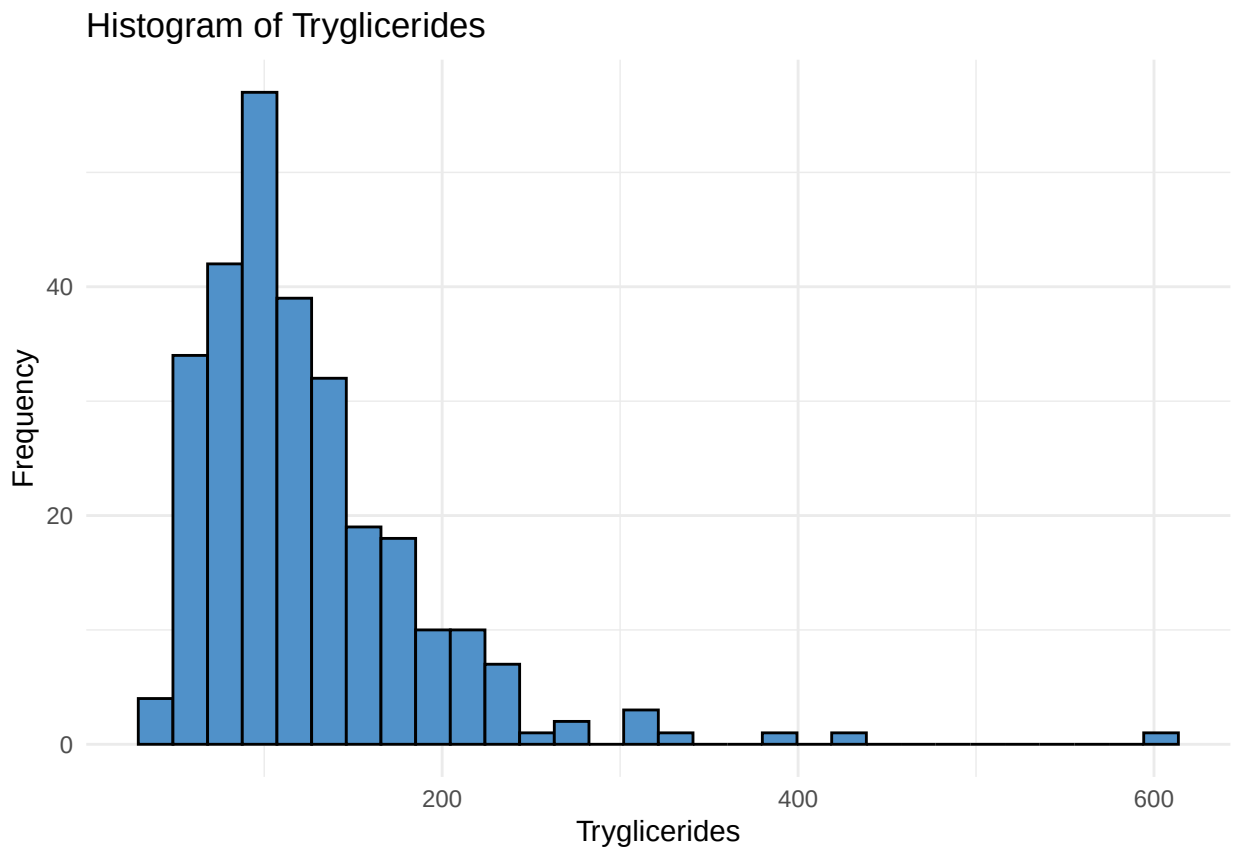

Histogram of Cholesterol
Frequency
60
40
20
0
0
500
1000
1500
Cholesterol

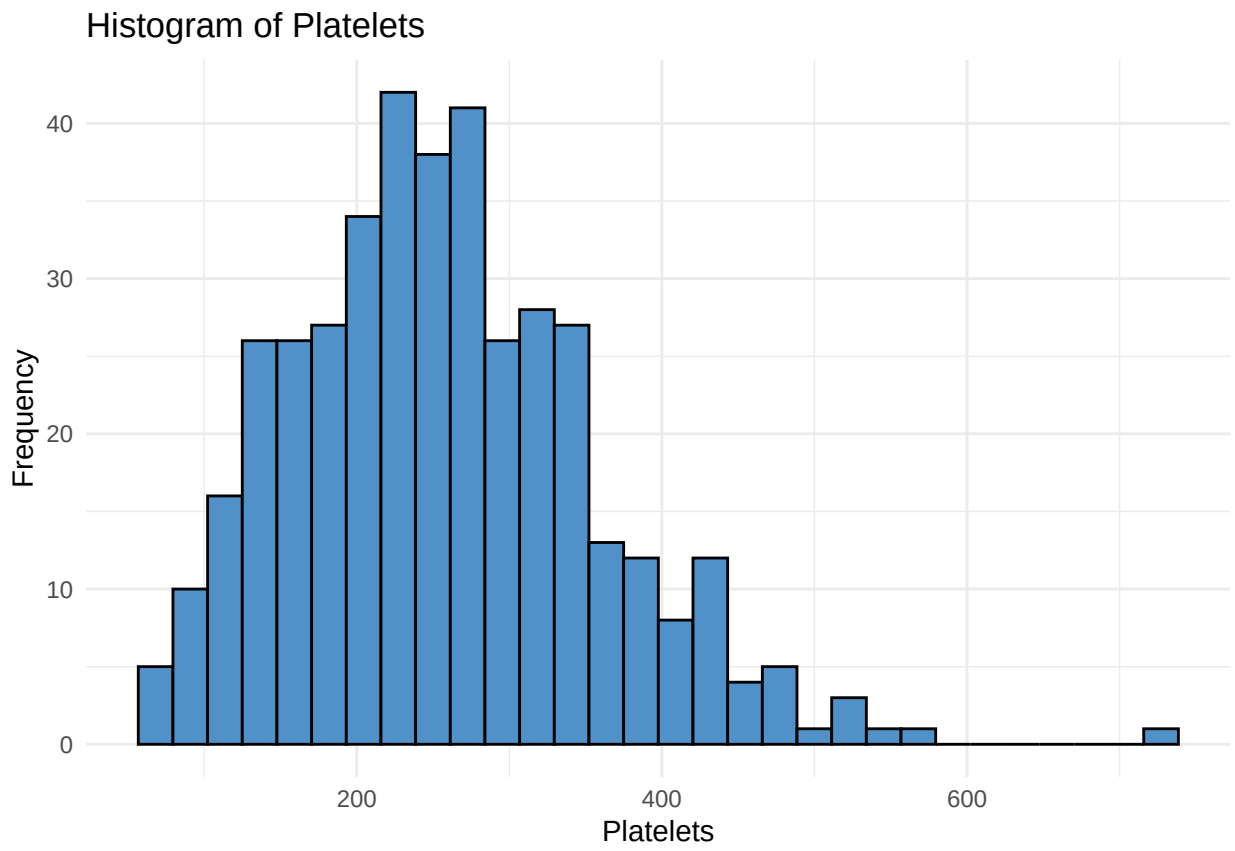

Histogram of Albumin
Frequency
50
40
30
20
10
0
2
3
4
Albumin

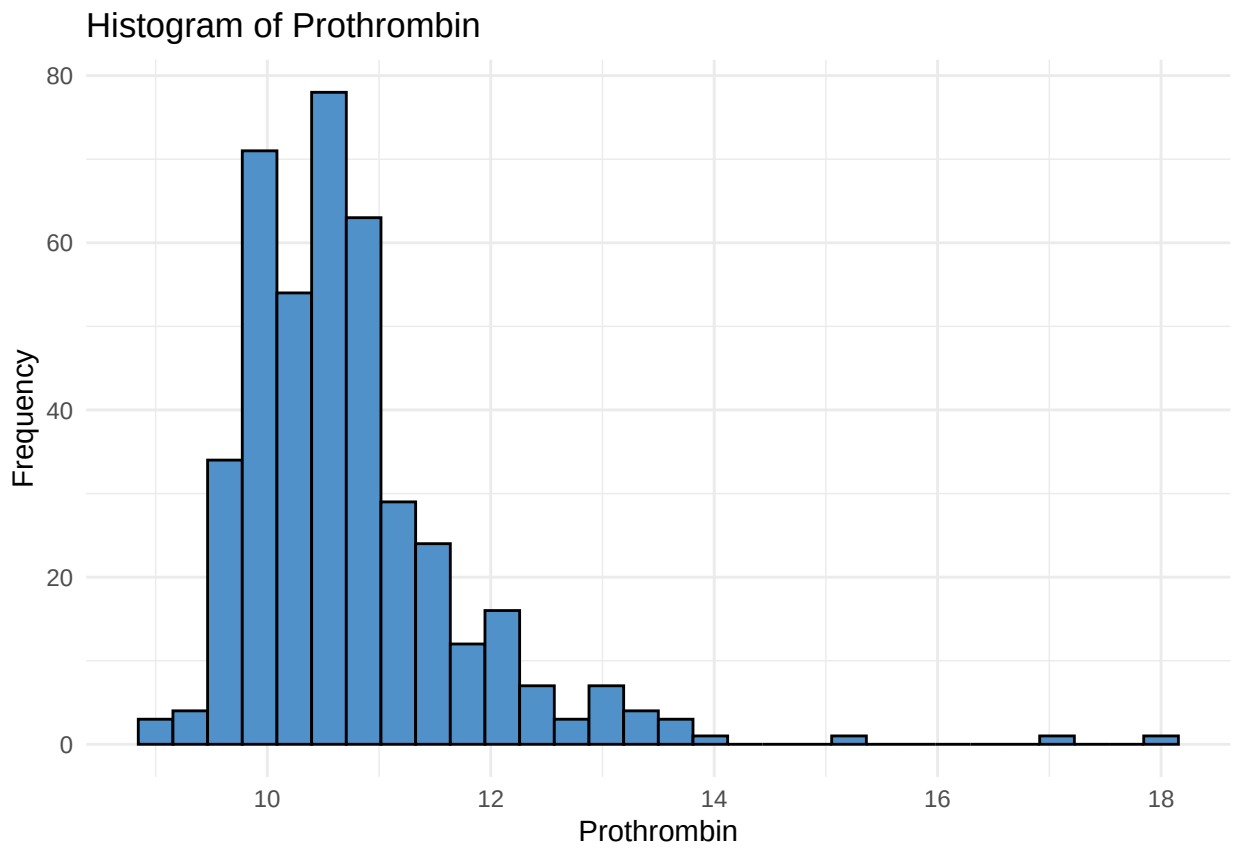

Histogram of Copper
Frequency
60
40
20
0
0
200
400
600
Copper

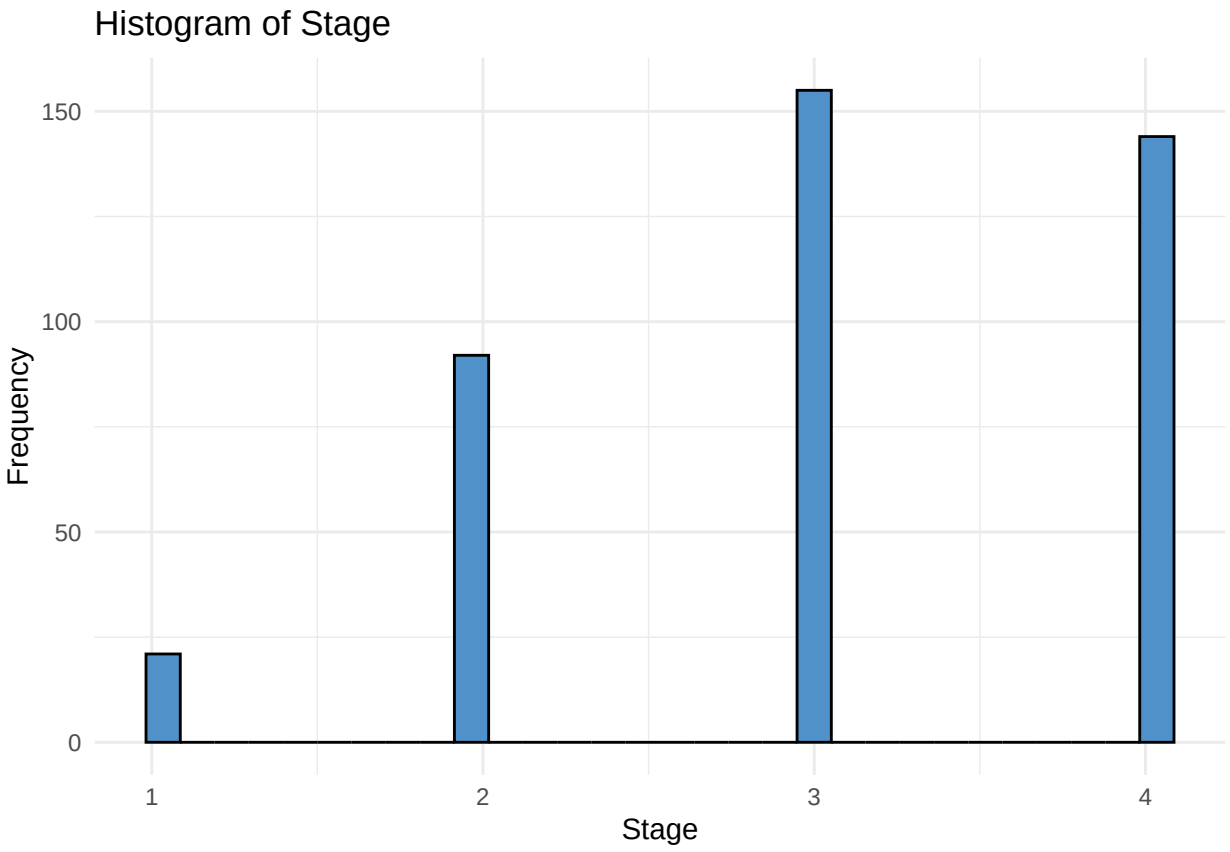

Histogram of Alk_Phos
Frequency
100
75
50
25
0
0
5000
10000
Alk_Phos


Histogram of SGOT
Frequency
SGOT
0
10
20
30
40
0
100
200
300
400


Histogram of Trygliceride
Frequency
40
20
0
200
400
600
Trygliceride


Histogram of Platelets
Frequency
40
30
20
10
0
200
400
600
Platelets


Histogram of Prothrombin
Frequency
80
60
40
20
0
10
12
14
16
18
Prothrombin



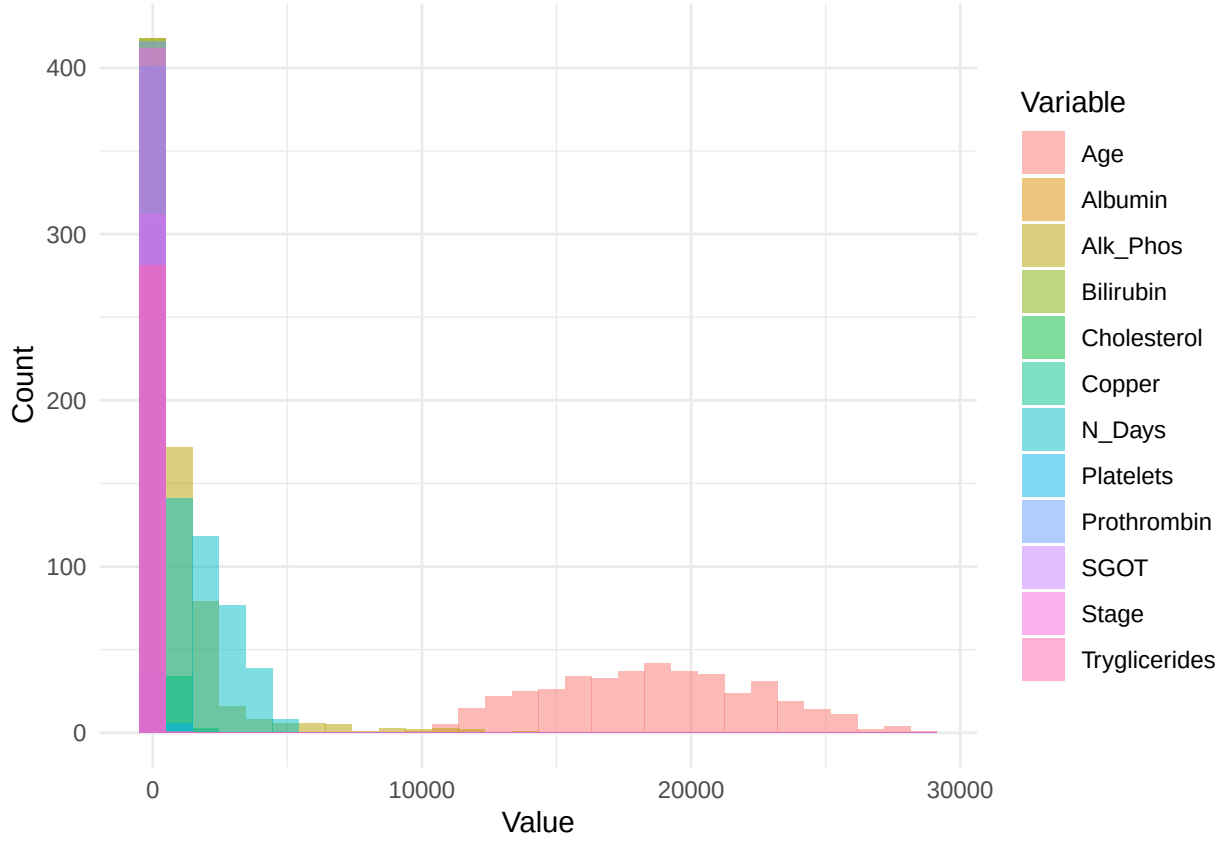
- The histograms display a range of distribution shapes for different variables, suggesting a mix of skewed and normally distributed data.
- The histograms show a variety of distribution shapes, indicating a mix of skewed and normally distributed data across the different variables.
- The spread of the histograms gives an indication of the range of each variable. Wide spreads can suggest high variability, while narrow spreads may indicate that values are concentrated around a particular number.
- The peak of each histogram shows the mode, providing insight into the most common values within a dataset.
- This suggests that normalization or standardization may be necessary to align them on a common scale for analytical purposes.

Single Histogram displaying all variables

```
# Load the necessary libraries
library(ggplot2)
library(dplyr)
library(tidyr)

# Convert data frame to long format for plotting
long_data <- cirrhosis %>%
```

```r
  select(where(is.numeric)) %>%
  pivot_longer(everything(), names_to = "Variable", values_to = "Value")

# Create overlaid histograms
ggplot(long_data, aes(x = Value, fill = Variable)) +
  # Overlay histograms for each variable
  geom_histogram(position = "identity", alpha = 0.5, bins = 30) +
  # Add labs and theme
  labs(x = "Value", y = "Count") +
  theme_minimal() +
  # Adjust legend position and title
  scale_fill_discrete(name = "Variable")
```

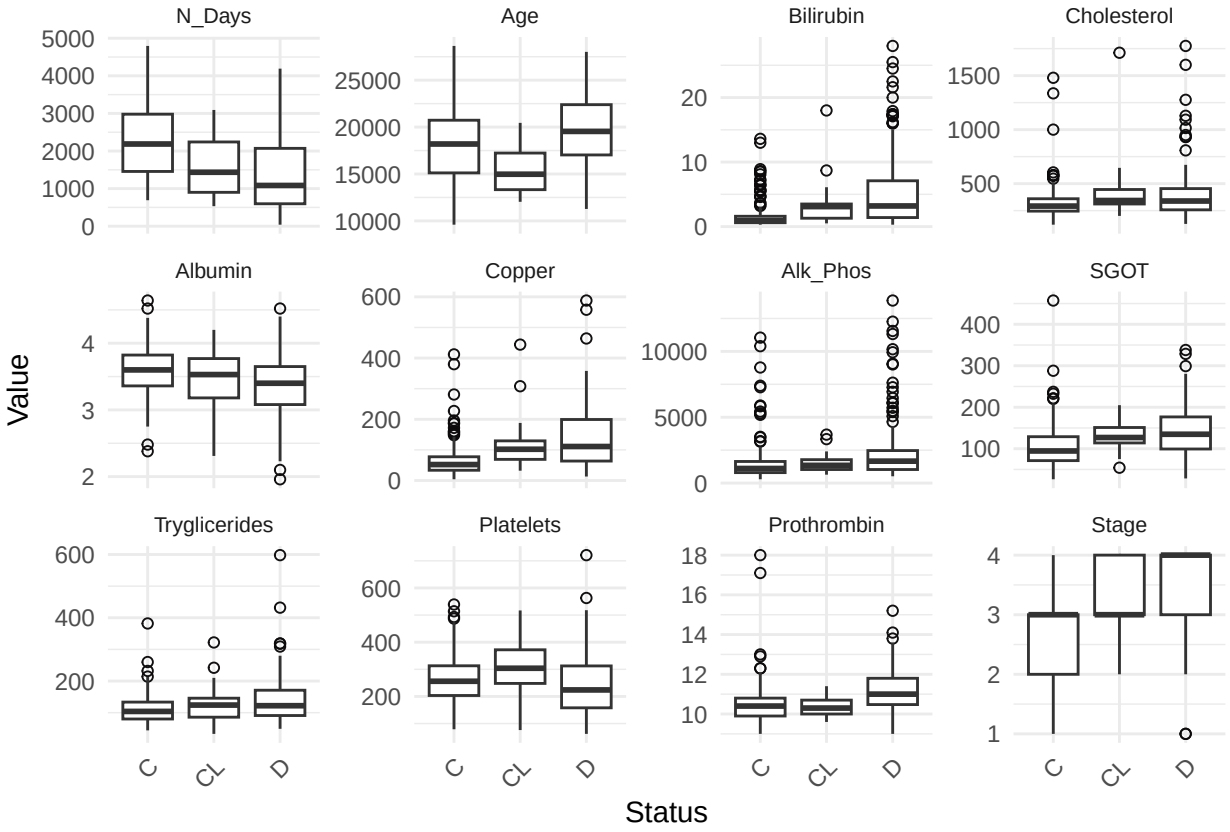
```
## Warning: Removed 609 rows containing non-finite values (`stat_bin()`).
```



- The overlaid histograms provide a visual comparison of the distribution of numeric variables in the dataset. This visualization allows for a quick assessment of the range, shape, and spread of each variable.
- The histograms are color-coded by variable, making it easier to distinguish between different distributions. This visualization can help identify patterns and outliers in the data, as well as understand the distribution of each variable.
- Depending on the shape of the histograms, certain variables might benefit from transformations to normalize the data, such as logarithmic or square root transformations for right-skewed data.

Detecting Outliers using Boxplots for continuous variables

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

# Convert 'Status' to a factor for plotting
cirrhosis$Status <- as.factor(cirrhosis$Status)

# Identify numeric features
numeric_features <- sapply(cirrhosis, is.numeric)

# Create a faceted boxplot for numeric features
numeric_data <- cirrhosis[, numeric_features]

# Convert data frame to long format for faceting
long_data <- reshape2::melt(
  cirrhosis,
  id.vars = "Status",
  measure.vars = names(numeric_data)
)

# Create faceted boxplot for numeric features
ggplot(long_data, aes(x = Status, y = value)) +
  # Create boxplot with outliers colored and shaped
  geom_boxplot(outlier.colour = "#0c0a0af8", outlier.shape = 1) +
  # Facet by variable with free y-axis scales
  facet_wrap(~variable, scales = "free_y") +
  # Add labels and theme
  labs(x = "Status", y = "Value") +
  theme_minimal() +
  theme(
    # Rotate x-axis labels for better readability
    axis.text.x = element_text(angle = 45, hjust = 1),
    strip.text.x = element_text(size = 8)
  )
```



- The boxplots show the distribution of numeric features by the survival status of the patients.
- Several features display a significant number of outliers, suggesting variations in the dataset that may affect model accuracy. The boxplots show that the distributions of numeric variables vary greatly, indicating diverse patient characteristics.
- Many boxplots exhibit skewness either to the right or left, indicating that most numeric variables are not symmetrically distributed. The medians of the boxplots vary across different levels of the 'Status' target variable, highlighting their potential impact on patient survival outcomes.

Detecting Outliers using Tukey's Method

```
# Detect outliers using Tukey's method
outliers <- apply(numeric_data, 2, function(x) {
  # Calculate the lower and upper bounds for outliers
  qnt <- quantile(x, probs = c(0.25, 0.75), na.rm = TRUE)
  # Calculate the Interquartile Range (IQR)
  H <- 1.5 * IQR(x, na.rm = TRUE)
  # Identify outliers based on Tukey's method
  x[x < (qnt[1] - H) | x > (qnt[2] + H)]
})

# Count the number of outliers for each variable
outliers_count <- sapply(outliers, length)
```

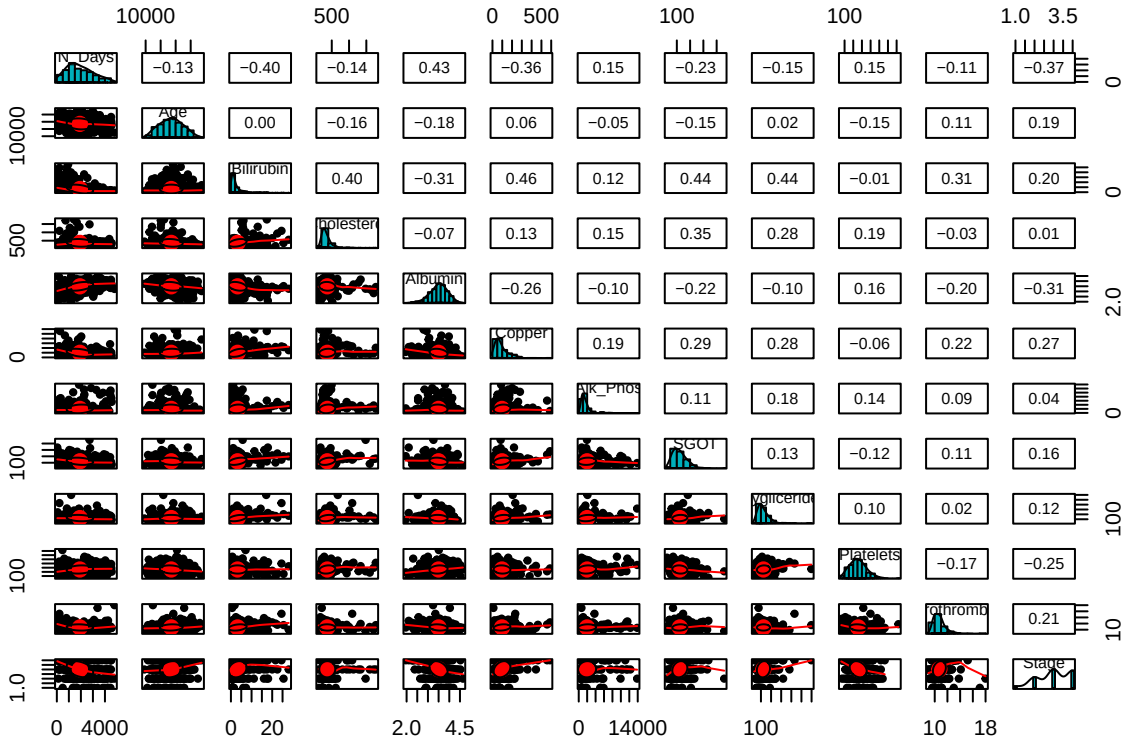
```r
# Display the number of outliers for each variable
outliers_count
```

```
##        N_Days           Age     Bilirubin   Cholesterol       Albumin
##             0             0            46           154             9
##        Copper      Alk_Phos          SGOT Trygliceride     Platelets
##           125           141           113           146            17
##   Prothrombin         Stage
##            20             6
```

- The number of outliers detected for each numeric variable is displayed above using Tukey's method which is 1.5 times the Interquartile Range (IQR).
- IQR is a robust measure of spread that is less sensitive to outliers than the range.
- It is important to consider these outliers during data preprocessing and model building to ensure robust and accurate predictions.

Pairs Panels Visualization

```r
# Install the 'psych' package if not already installed
if (!require(psych)) {
  install.packages("psych", repos = "http://cran.rstudio.com/")
}

# Load the psych package
library(psych)

# Select only numeric columns for correlation analysis
numeric_data <- cirrhosis[sapply(cirrhosis, is.numeric)]

# Using pairs with panel.smooth
pairs.panels(numeric_data,
  # Correlation method and appearance customization
  method = "pearson",
  # Customize the appearance of the pairs panel
  hist.col = "#00AFBB",
  # Show density plots
  density = TRUE,
  # Show correlation ellipses
  ellipses = TRUE,
  # Show smooth lines
  smooth = TRUE
)
```

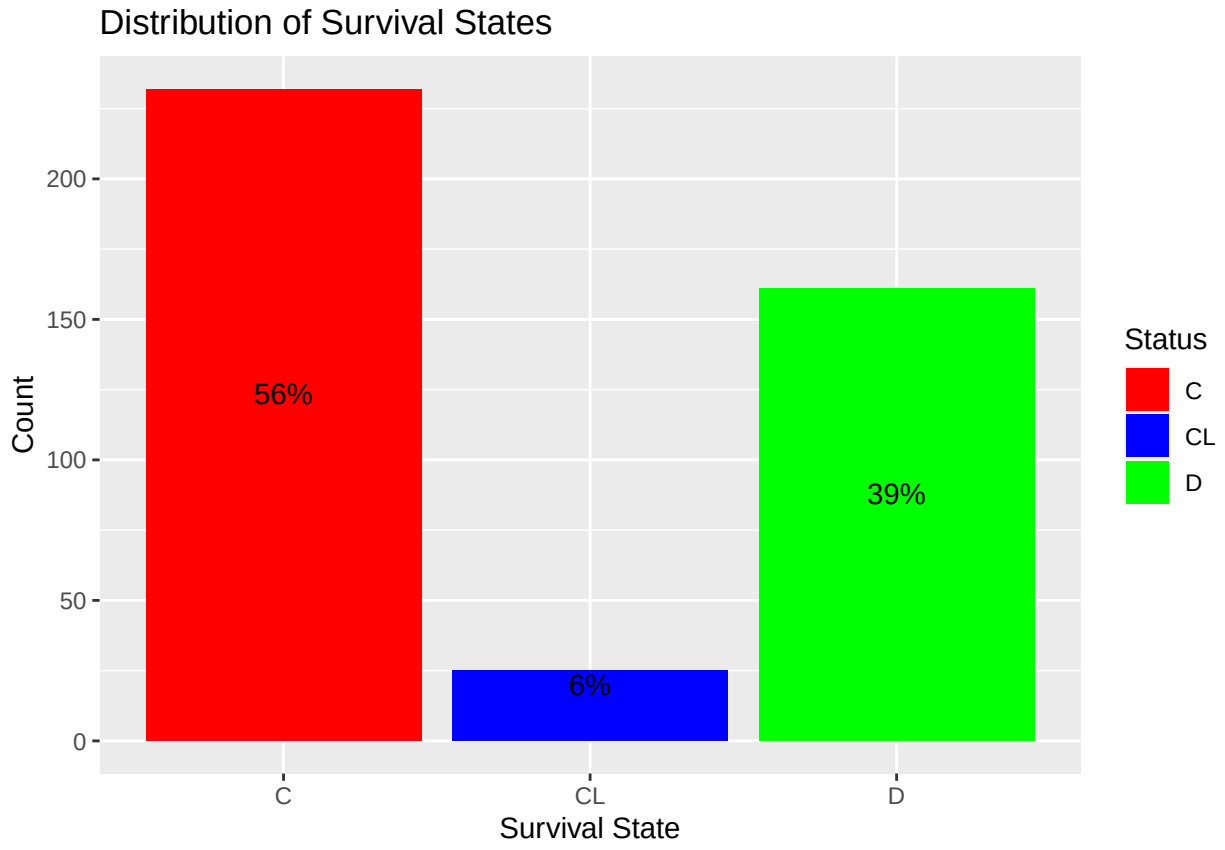


- The pairs panel visualization provides a comprehensive view of the relationships between numeric variables in the dataset. The histograms along the diagonal provide distributions for each variable, showing varied skewness and central tendencies which may suggest different underlying distributions for each variable.
- The scatter plots below the diagonal indicate the relationships between pairs of variables. Some pairs show positive correlations, negative correlations, or no discernible relationship. The correlation coefficients above the diagonal numerically summarize the degree of linear relationship between pairs of variables. These coefficients range from -1 to 1, indicating strong negative to strong positive correlations respectively.
- The use of ellipses in some plots visually emphasizes the strength and direction of the correlations, with tighter ellipses indicating stronger correlations. The color coding and density plots overlaying the scatter plots provide further depth, highlighting the density of data points and the gradient of relationships, which can be crucial for understanding complex interactions within the data.

Evaluation of Distribution of Survival States

```
# Balance analysis of survival state categories
library(ggplot2)
library(dplyr)

# Calculate the total count of each survival state
total_count <- sum(table(cirrhosis$Status))
```

```r
# Create the bar plot and add percentages
ggplot(cirrhosis, aes(x = Status, fill = Status)) +
  geom_bar() +
  geom_text(
    stat = "count", aes(label = scales::percent(..count.. / total_count)),
    # Adjust vertical position of text
    vjust = -0.5,
    position = position_stack(vjust = 0.5)
  ) +
  ggtitle("Distribution of Survival States") +
  # Add axis labels
  xlab("Survival State") +
  ylab("Count") +
  scale_fill_manual(
    # Custom colors for each state
    values = c("C" = "red", "CL" = "blue", "D" = "green")
  )
```



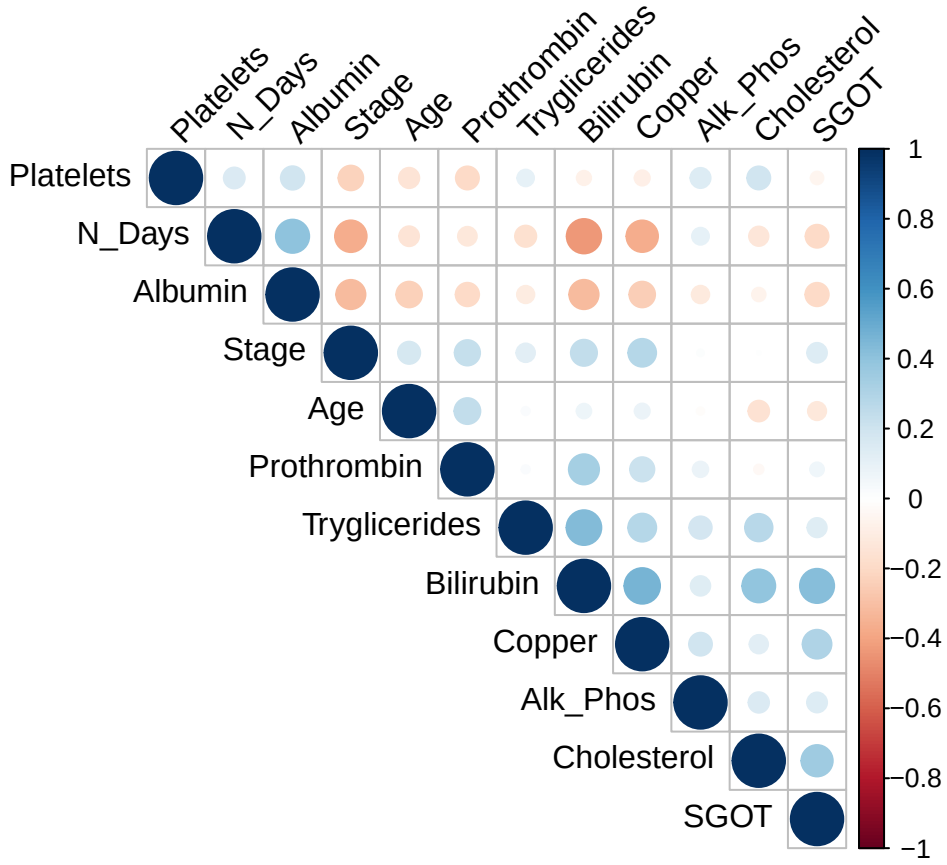
- The bar plot above shows the distribution of survival states in the dataset. The survival states are labeled as 'C', 'CL', and 'D', representing different survival outcomes for patients with cirrhosis.
- The percentage of survival stage of C is 55.5%, CL is 5.98%, and D is 38.52%. The distribution of survival states is imbalanced, with the majority of patients in the 'C' category.

Correlation Analysis

```
# Load the necessary library
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
# Calculating correlation matrix for numeric variables
correlation_matrix <- cor(numeric_data, use = "complete.obs")

# Visualizing the correlation matrix
corrplot(correlation_matrix,
  method = "circle", type = "upper", order = "hclust",
  tl.col = "black", tl.srt = 45
)
```



- The correlation matrix provides insights into the relationships between numeric variables in the dataset. The correlation coefficients range from -1 to 1, indicating the strength and direction of the linear relationships between variables.
- The correlation matrix is visualized using a circular plot, with the intensity of the color and the size of the circle representing the magnitude of the correlation coefficients. The variables are ordered based on hierarchical clustering, highlighting groups of variables with similar correlation patterns.

- There is a strong positive correlation between some of the liver function tests, like "Bilirubin" and "Alk_Phos" (alkaline phosphatase), "SGOT" (serum glutamic-oxaloacetic transaminase), and "Copper". This is expected as these are indicators of liver function and typically move in the same direction in response to liver injury or disease.
- "Albumin" appears to be negatively correlated with "Bilirubin", "Copper", and "SGOT". A high level of albumin is often found in healthy individuals, while the other three are indicators that can signify liver damage when elevated.
- "Stage" of the disease shows correlations with multiple variables such as "Albumin", "Bilirubin", "Copper", and others. This implies that as the stage of liver disease progresses, these biochemical markers tend to change correspondingly.

Shapiro-Wilk Normality Test for Normality Assessment

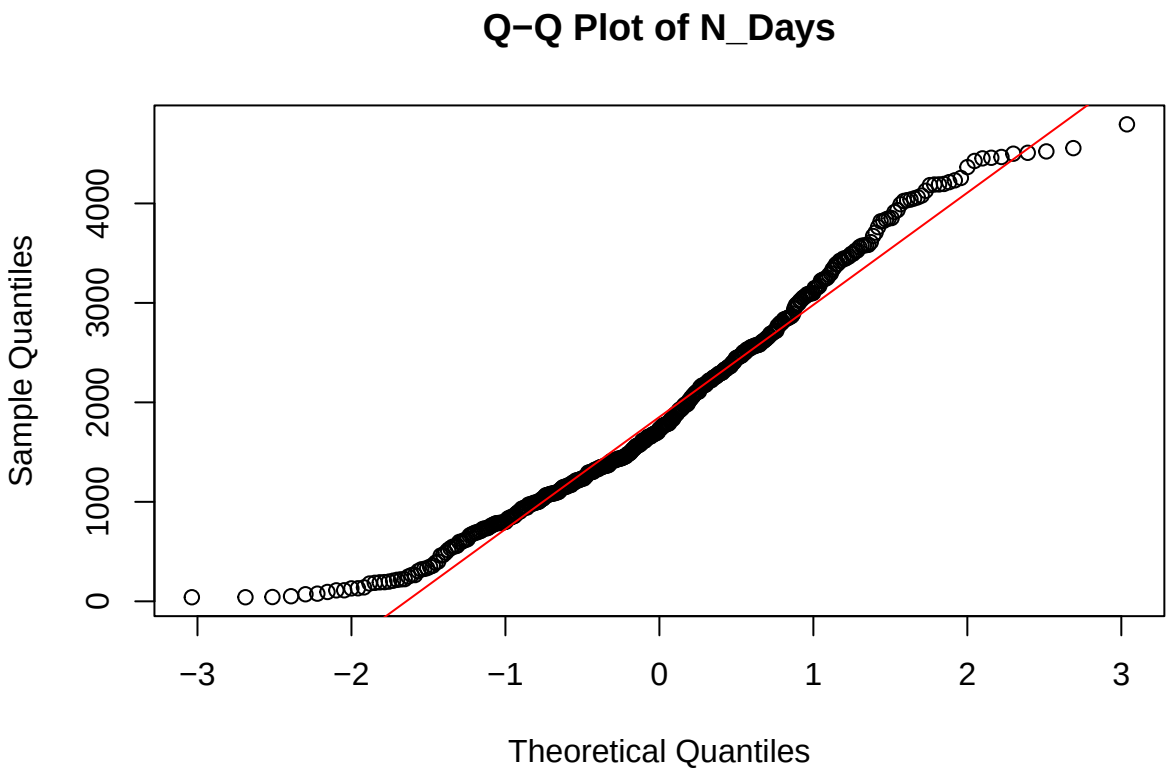
```
# Shapiro-Wilk normality test for all numeric variables
sapply(numeric_data, function(x) shapiro.test(x)$p.value)
```

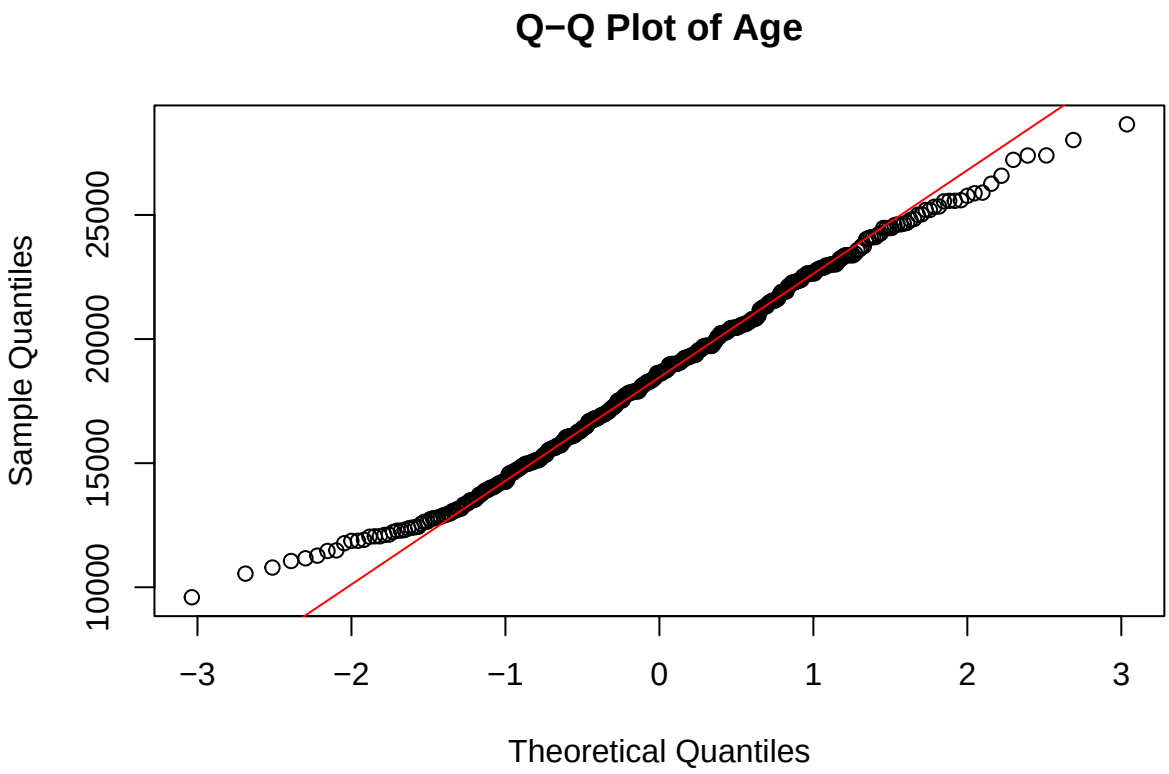
```
##        N_Days           Age     Bilirubin   Cholesterol       Albumin
##  8.291488e-08  8.918234e-03  5.907597e-29  6.423407e-24  6.386612e-04
##        Copper      Alk_Phos          SGOT Tryglicerides     Platelets
##  8.351096e-20  6.849605e-26  1.467220e-12  1.221285e-17  4.207792e-06
##   Prothrombin         Stage
##  1.590303e-19  6.406701e-20
```

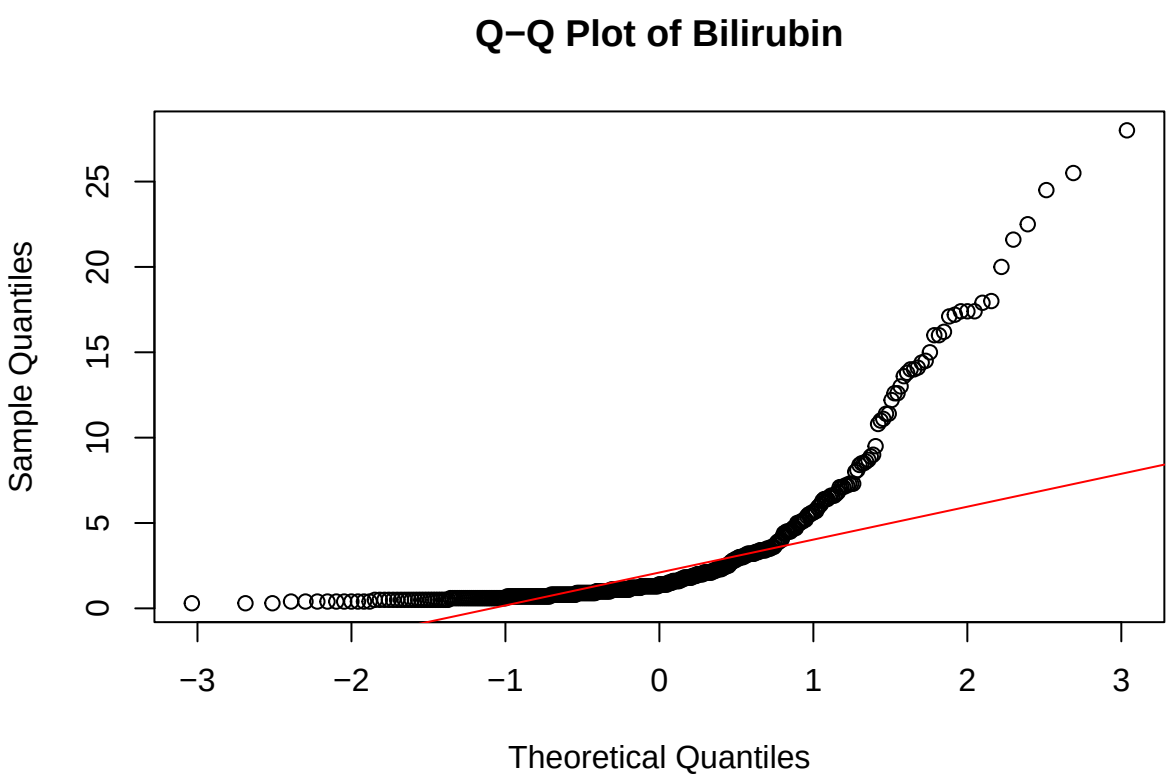
- The Shapiro-Wilk normality test is used to assess the normality of the distribution of numeric variables. The p-values indicate whether the data significantly deviates from a normal distribution.
- The p-values for the Shapiro-Wilk test are displayed above for each numeric variable. Low p-values suggest that the data may not be normally distributed, which can impact the choice of statistical tests and modeling techniques.
- A low p-value (typically less than 0.05) suggests that the distribution of the variable is not normal.

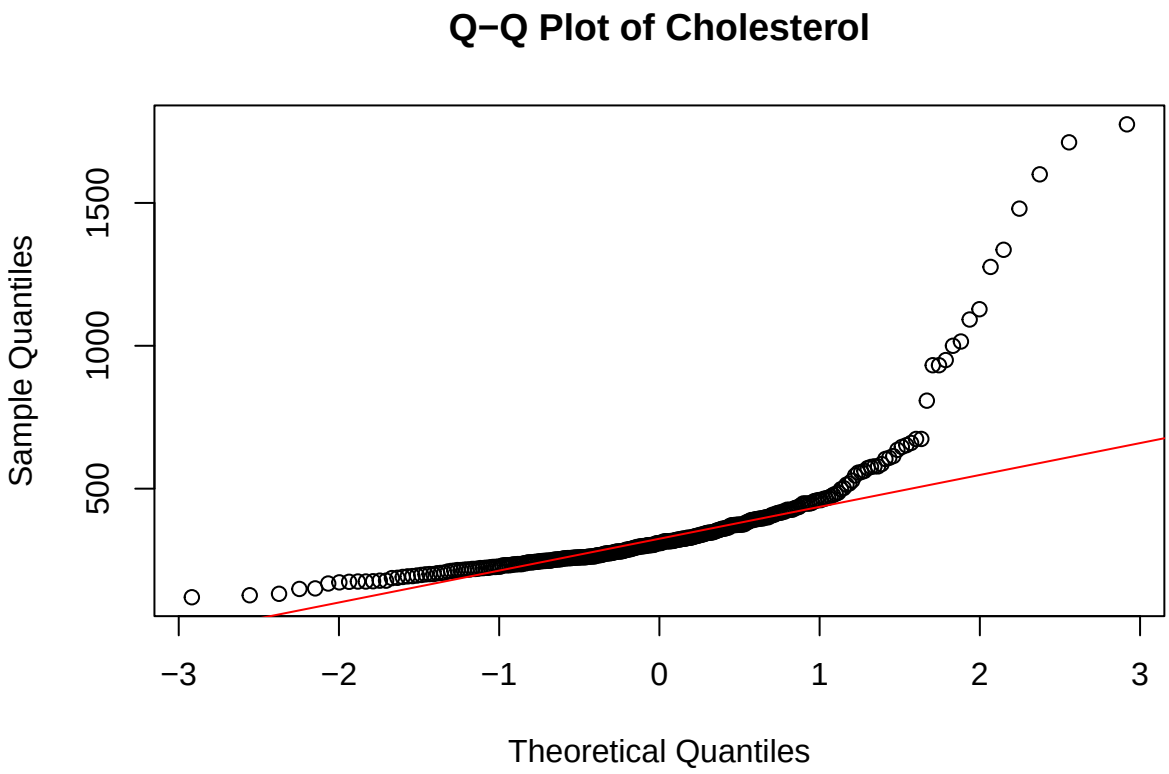
Q-Q Plots for Normality Assessment

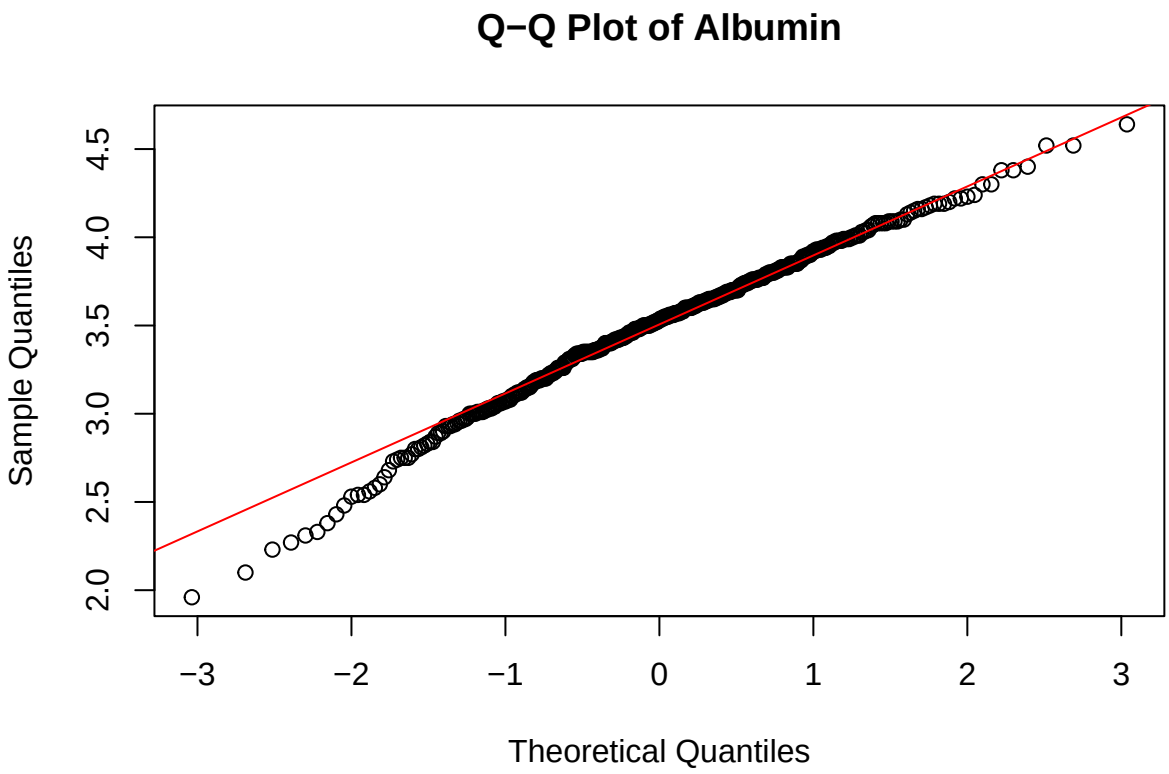
```
# Generating Q-Q plots for all numeric variables
lapply(names(numeric_data), function(feature) {
  qqnorm(cirrhosis[[feature]], main = paste("Q-Q Plot of", feature))
  qqline(cirrhosis[[feature]], col = "red")
})
```

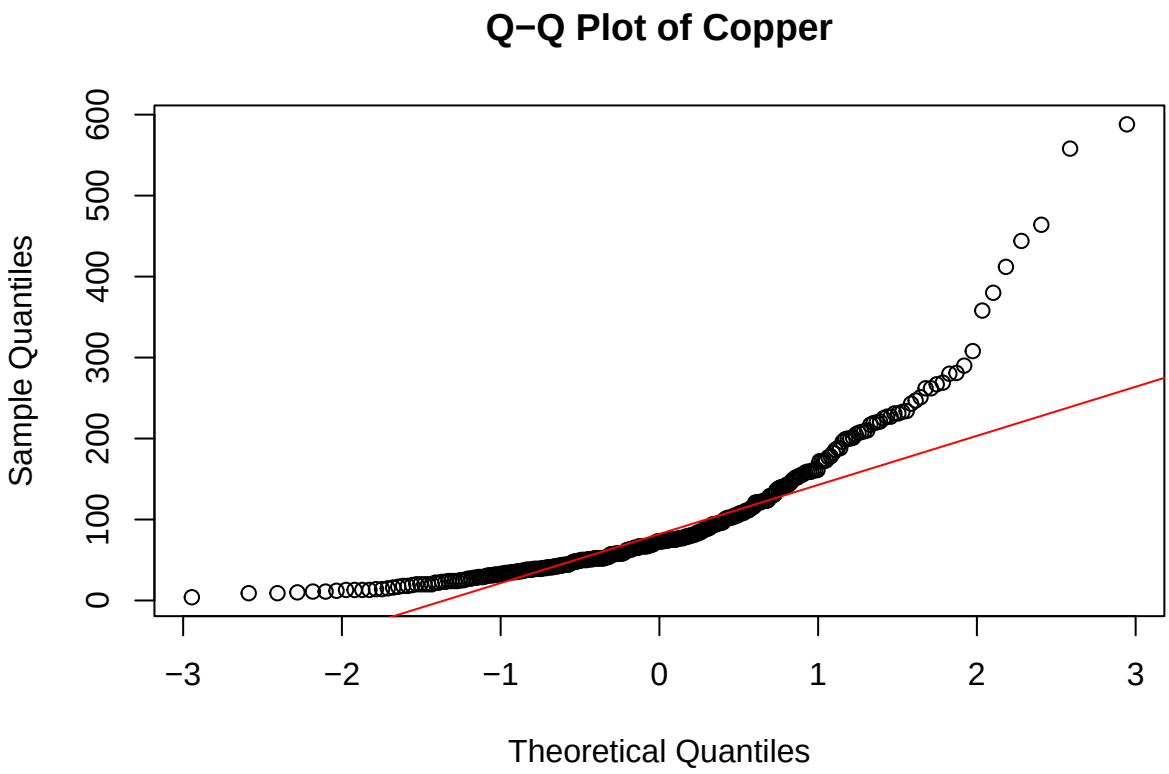

Q−Q Plot of N_Days
Sample Quantiles
0
1000
2000
3000
4000
−3
−2
−1
0
1
2
3
Theoretical Quantiles

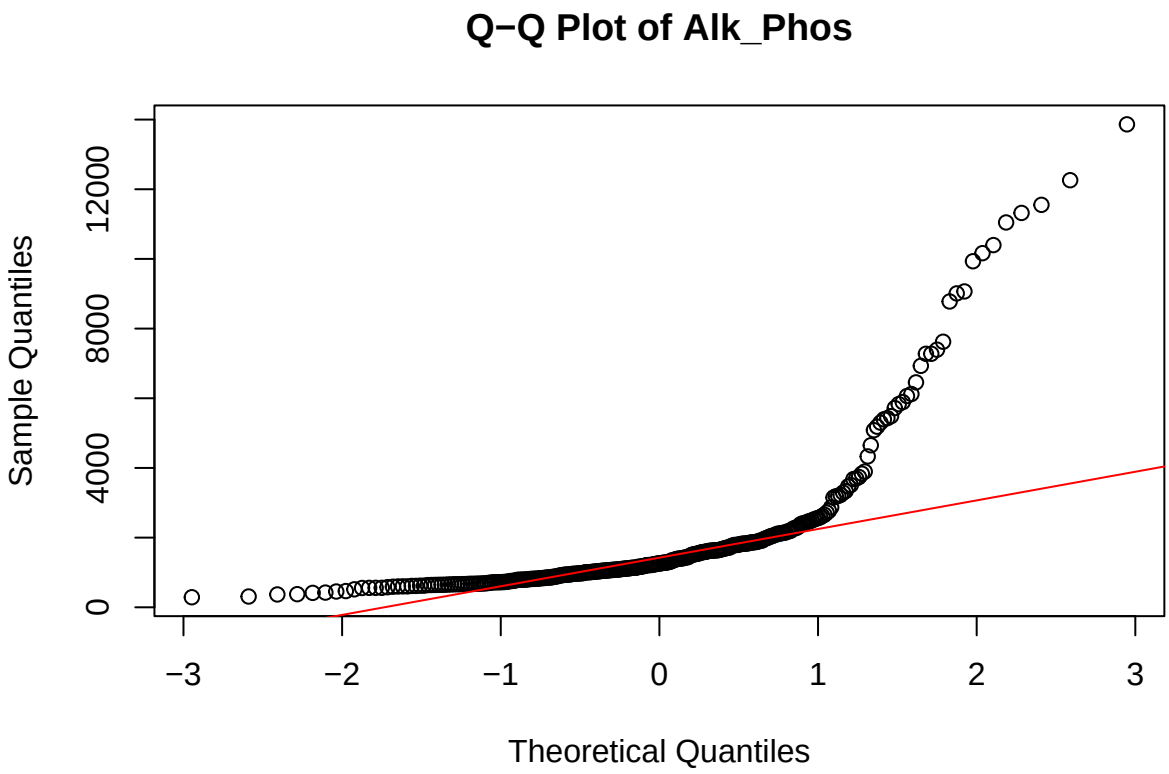

Q−Q Plot of Age
Sample Quantiles
10000
15000
20000
25000
−3
−2
−1
0
1
2
3
Theoretical Quantiles

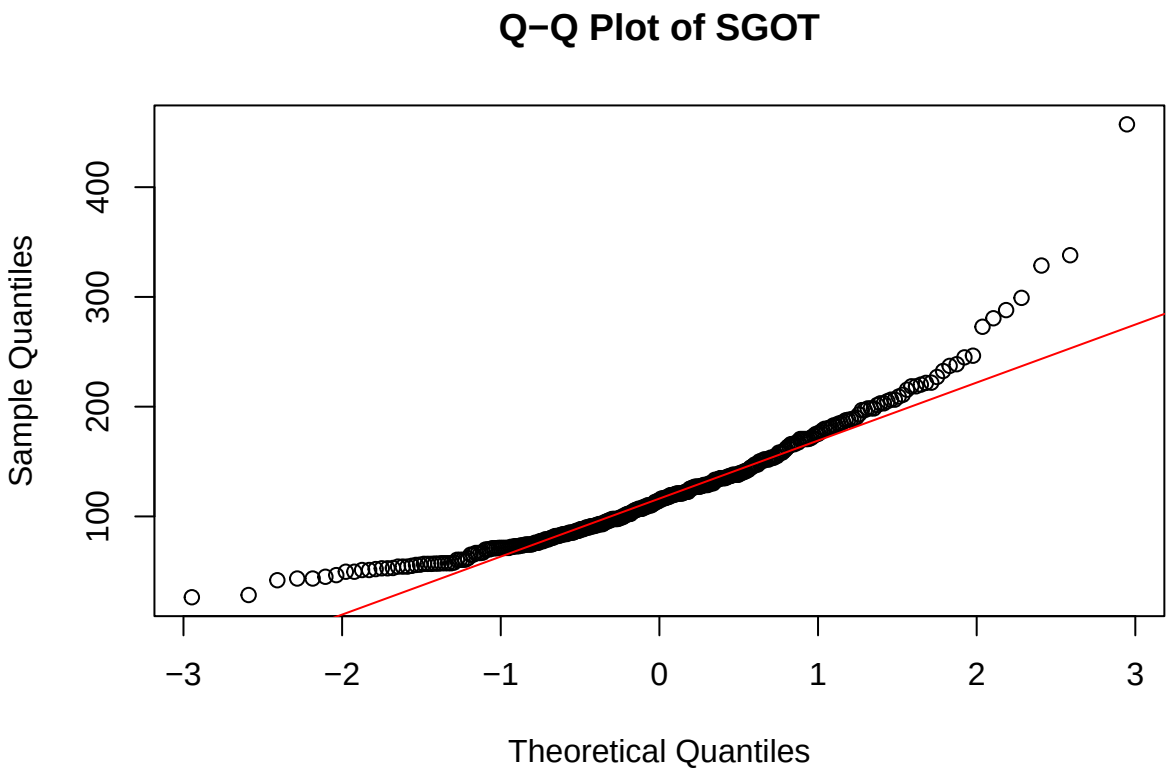

Q−Q Plot of Bilirubin
Sample Quantiles
0
5
10
15
20
25
−3
−2
−1
0
1
2
3
Theoretical Quantiles

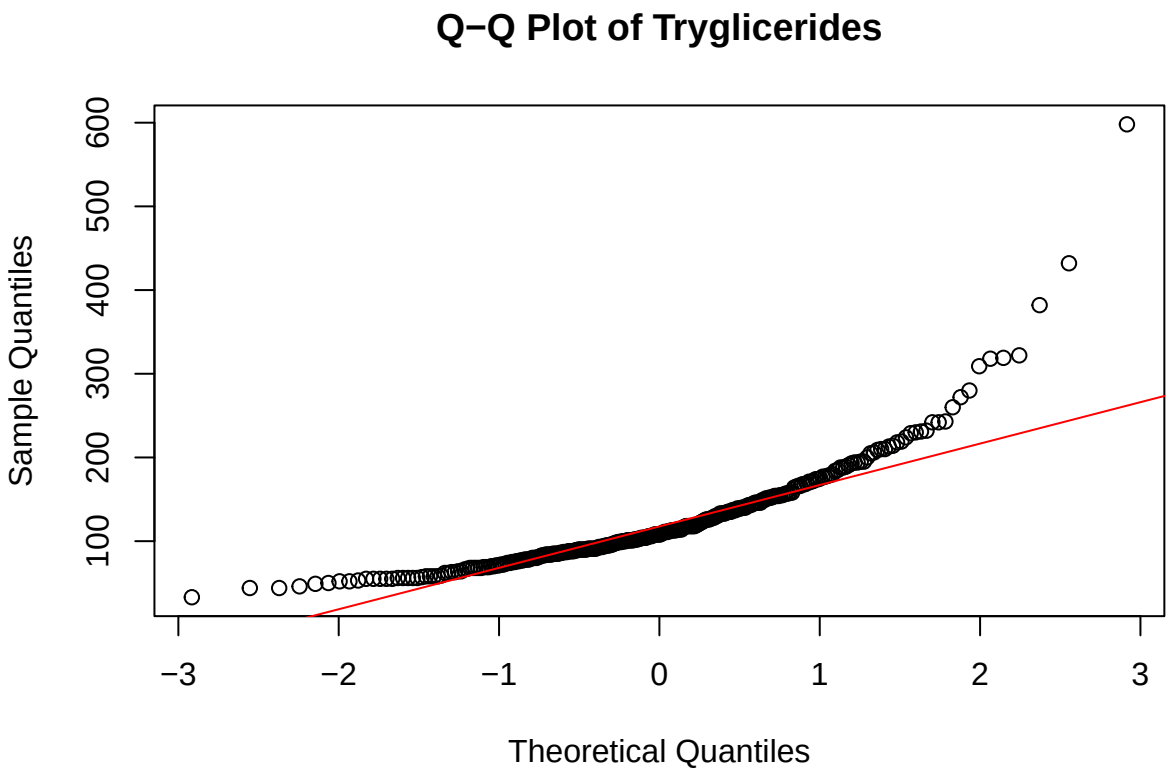

Q−Q Plot of Cholesterol
Sample Quantiles
500
1000
1500
−3
−2
−1
0
1
2
3
Theoretical Quantiles

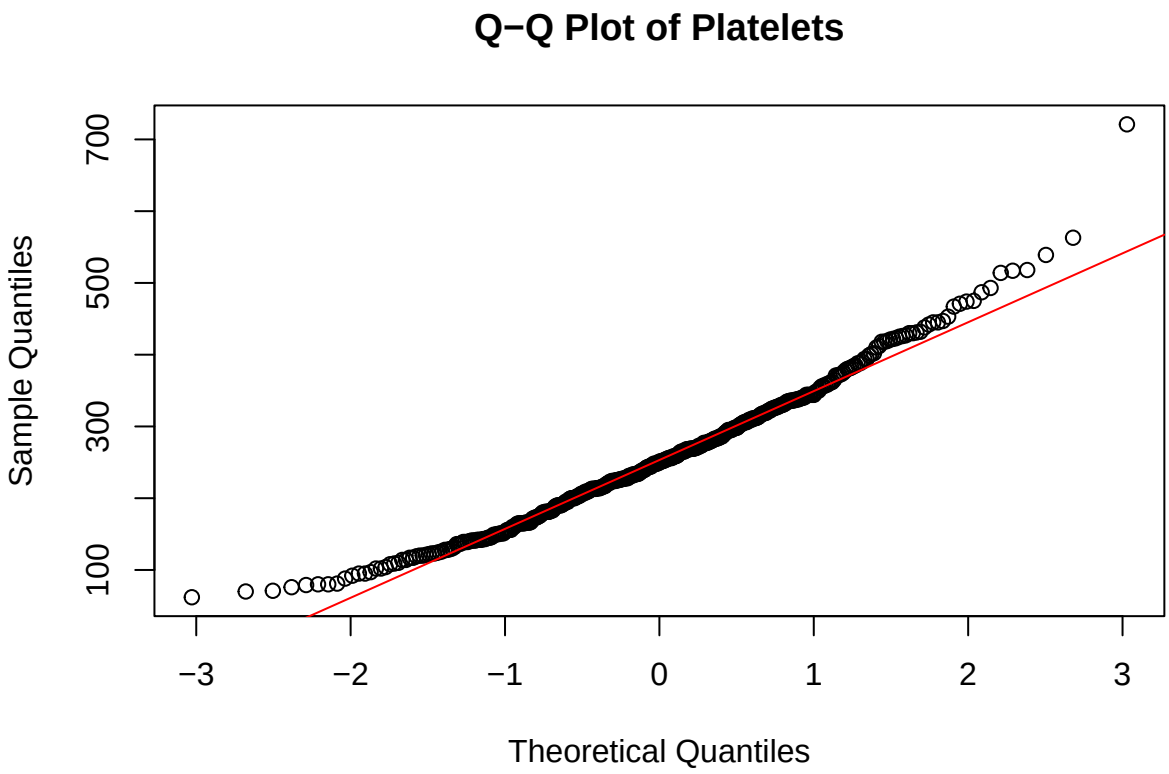

Q−Q Plot of Albumin
Sample Quantiles
Theoretical Quantiles
2.0
2.5
3.0
3.5
4.0
4.5
−3
−2
−1
0
1
2
3

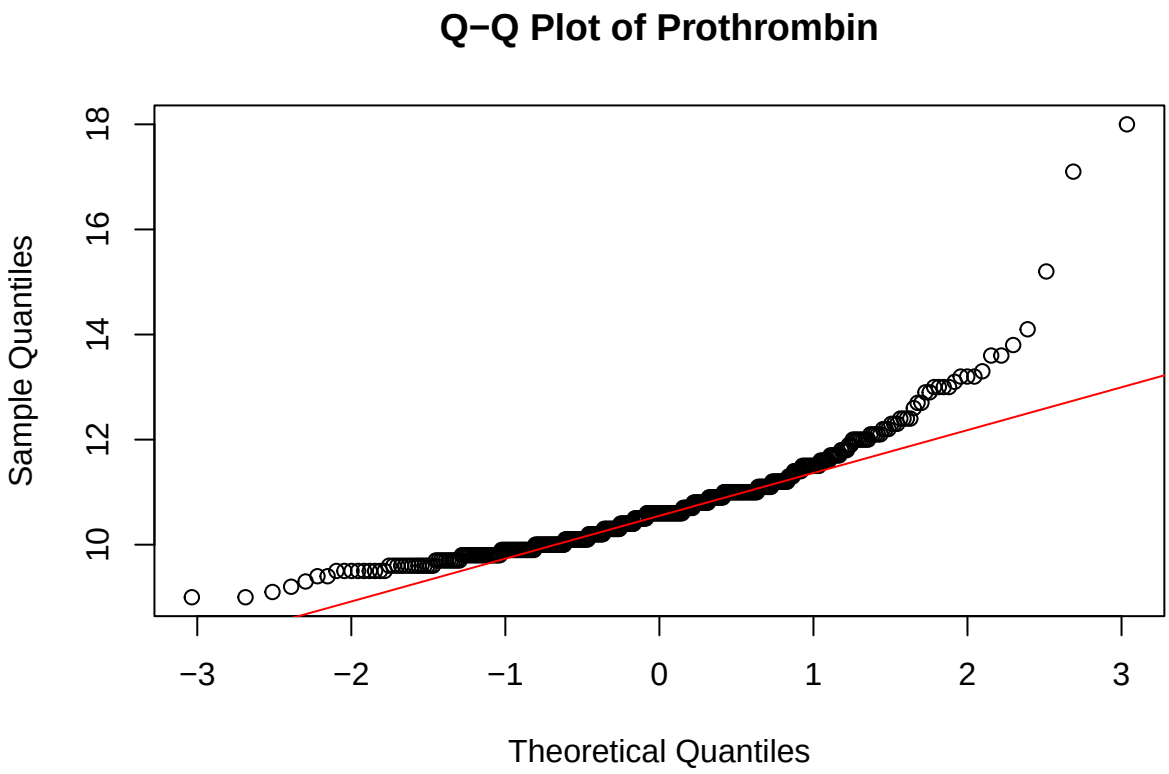

Q−Q Plot of Copper
Sample Quantiles
0
100
200
300
400
500
600
−3
−2
−1
0
1
2
3
Theoretical Quantiles

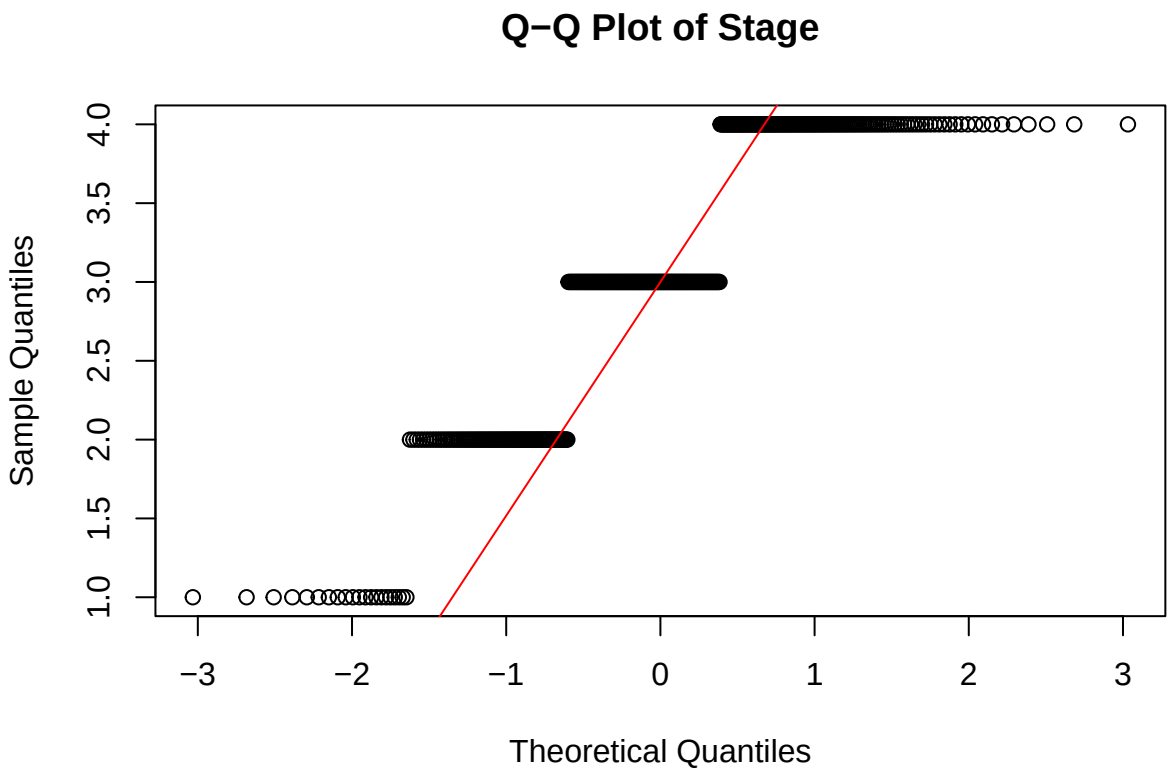

Q−Q Plot of Alk_Phos
Sample Quantiles
0
4000
8000
12000
−3
−2
−1
0
1
2
3
Theoretical Quantiles


Q−Q Plot of SGOT
Sample Quantiles
100
200
300
400
−3
−2
−1
0
1
2
3
Theoretical Quantiles


Q−Q Plot of Tryglicerides
Sample Quantiles
100
200
300
400
500
600
−3
−2
−1
0
1
2
3
Theoretical Quantiles


Q−Q Plot of Platelets
Sample Quantiles
100
300
500
700
−3
−2
−1
0
1
2
3
Theoretical Quantiles


Q−Q Plot of Prothrombin
Sample Quantiles
10
12
14
16
18
−3
−2
−1
0
1
2
3
Theoretical Quantiles




```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
##
## [[5]]
## NULL
##
## [[6]]
## NULL
##
## [[7]]
## NULL
##
## [[8]]
## NULL
##
## [[9]]
## NULL
```

```
##
## [[10]]
## NULL
##
## [[11]]
## NULL
##
## [[12]]
## NULL
```

- The Q-Q plots above provide a visual assessment of the normality of the distribution of numeric variables. The points should ideally fall along the red line, indicating a normal distribution.
- Deviations from the red line suggest non-normality in the data. These plots can help identify variables that may require transformation or non-parametric analysis to address non-normality.
- The Q-Q plots are useful for assessing the distribution of numeric variables and identifying potential deviations from normality, which can inform data preprocessing and modeling decisions.
- The Q-Q plots provide a visual representation of the distribution of each numeric variable, allowing for a quick assessment of normality assumptions.
- It is important to consider the normality of data when selecting statistical tests and modeling techniques to ensure accurate and reliable results.
- For features like N_Days, Age, Bilirubin, and Cholesterol, the points deviate from the straight line, suggesting that these features do not follow a normal distribution.
- In some plots, such as for Bilirubin and Cholesterol, the points form a curve that tails off at the ends. This pattern suggests a heavy-tailed distribution, indicating the presence of outliers or extreme values in the data.

Data Cleaning and Preprocessing

Missing Values Identification

```
# Identify missing values
missing_values <- colSums(is.na(cirrhosis))

# Display the number of missing values for each column
missing_values
```

```
##        N_Days        Status          Drug           Age           Sex
##             0             0           106             0             0
##       Ascites  Hepatomegaly       Spiders        Edema     Bilirubin
##           106           106           106             0             0
##   Cholesterol       Albumin        Copper      Alk_Phos          SGOT
##           134             0           108           106           106
## Tryglicerides     Platelets   Prothrombin         Stage
##           136            11             2             6
```

- The missing values are identified in the dataset, with the number of missing values displayed for each column. The presence of missing values can impact the quality and reliability of the analysis and modeling.

- There are 1033 missing values in the dataset. It is important to address these missing values through imputation or removal to ensure the integrity of the data.

Imputation of Missing Values

```
# Impute missing values for numeric columns with median
for (col in names(cirrhosis)) {
  if (is.numeric(cirrhosis[[col]])) {
    cirrhosis[[col]][is.na(cirrhosis[[col]])] <- median(
      cirrhosis[[col]],
      na.rm = TRUE
    )
  }
}

# Impute missing values for categorical columns with mode
mode <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}

# Impute missing values for categorical columns with mode
for (col in names(cirrhosis)) {
  if (is.character(cirrhosis[[col]])) {
    cirrhosis[[col]][is.na(cirrhosis[[col]])] <- mode(cirrhosis[[col]])
  }
}

# Check for missing values after all imputations
sum(is.na(cirrhosis))
```

```
## [1] 0
```

```
# Structure of the data after imputation
str(cirrhosis)
```

```
## 'data.frame':    418 obs. of  19 variables:
##  $ N_Days       : num  400 4500 1012 1925 1504 ...
##  $ Status       : Factor w/ 3 levels "C","CL","D": 3 1 3 3 2 3 1 3 3 3 ...
##  $ Drug         : chr  "D-penicillamine" "D-penicillamine" "D-penicillamine" "D-penicillamine" ...
##  $ Age          : num  21464 20617 25594 19994 13918 ...
##  $ Sex          : chr  "F" "F" "M" "F" ...
##  $ Ascites      : chr  "Y" "N" "N" "N" ...
##  $ Hepatomegaly : chr  "Y" "Y" "N" "Y" ...
##  $ Spiders      : chr  "Y" "Y" "N" "Y" ...
##  $ Edema        : chr  "Y" "N" "S" "S" ...
##  $ Bilirubin    : num  14.5 1.1 1.4 1.8 3.4 0.8 1 0.3 3.2 12.6 ...
##  $ Cholesterol  : num  261 302 176 244 279 248 322 280 562 200 ...
##  $ Albumin      : num  2.6 4.14 3.48 2.54 3.53 3.98 4.09 4 3.08 2.74 ...
##  $ Copper       : num  156 54 210 64 143 50 52 52 79 140 ...
##  $ Alk_Phos     : num  1718 7395 516 6122 671 ...
```

```
##  $ SGOT         : num  137.9 113.5 96.1 60.6 113.2 ...
##  $ Tryglicerides: num  172 88 55 92 72 63 213 189 88 143 ...
##  $ Platelets    : int  190 221 151 183 136 251 204 373 251 302 ...
##  $ Prothrombin  : num  12.2 10.6 12 10.3 10.9 11 9.7 11 11 11.5 ...
##  $ Stage        : num  4 3 4 4 3 3 3 3 2 4 ...
```

- Missing values are imputed in the dataset using the median for numeric columns and the mode for categorical columns. Imputation helps maintain the integrity of the data and ensures that all variables have complete information for analysis and modeling.
- Mean imputation involves replacing missing values in numeric data with the mean value of the respective variable. This technique is widely used because it preserves the sample mean, making it a reasonable choice for maintaining the overall distribution and central tendency of the data.
- Mode imputation involves replacing missing values in categorical data with the most frequent value (mode) of the respective variable. This technique is suitable for categorical variables with a limited number of unique values and helps maintain the distribution of the data.
- After imputation, there are no missing values in the dataset. The structure of the data is displayed to confirm that all missing values have been addressed through imputation.
- I didn't convert the categorical columns to numeric before imputation because converting categorical variables into numeric formats before imputation (such as assigning numbers to categories) can introduce arbitrary ordinality or false numeric relationships, which might not exist naturally within the data.
- For instance, converting "red", "blue", and "green" into 1, 2, and 3 might imply that green is somehow "higher" or "more" than red, which is semantically incorrect and can mislead the analysis. By keeping the data categorical and using mode imputation, we can maintain the integrity and meaning of the data.

One-Hot Encoding for Categorical Variables

```r
# Convert categorical character columns to factors
cirrhosis$Drug <- factor(cirrhosis$Drug)
cirrhosis$Sex <- factor(cirrhosis$Sex)
cirrhosis$Ascites <- factor(cirrhosis$Ascites)
cirrhosis$Hepatomegaly <- factor(cirrhosis$Hepatomegaly)
cirrhosis$Spiders <- factor(cirrhosis$Spiders)
cirrhosis$Edema <- factor(cirrhosis$Edema)

# Apply one-hot encoding
cirrhosis <- cbind(cirrhosis, model.matrix(
  ~ Drug + Sex + Ascites + Hepatomegaly + Spiders + Edema - 1,
  data = cirrhosis
))

# Removing original factor columns after encoding
cirrhosis <- cirrhosis[, !names(cirrhosis) %in% c(
  "Drug", "Sex", "Ascites", "Hepatomegaly", "Spiders", "Edema"
)]

# Structure of the data after encoding
str(cirrhosis)
```

```
## 'data.frame':    418 obs. of  21 variables:
##  $ N_Days             : num  400 4500 1012 1925 1504 ...
##  $ Status             : Factor w/ 3 levels "C","CL","D": 3 1 3 3 2 3 1 3 3 3 ...
##  $ Age                : num  21464 20617 25594 19994 13918 ...
##  $ Bilirubin          : num  14.5 1.1 1.4 1.8 3.4 0.8 1 0.3 3.2 12.6 ...
##  $ Cholesterol        : num  261 302 176 244 279 248 322 280 562 200 ...
##  $ Albumin            : num  2.6 4.14 3.48 2.54 3.53 3.98 4.09 4 3.08 2.74 ...
##  $ Copper             : num  156 54 210 64 143 50 52 52 79 140 ...
##  $ Alk_Phos           : num  1718 7395 516 6122 671 ...
##  $ SGOT               : num  137.9 113.5 96.1 60.6 113.2 ...
##  $ Tryglicerides      : num  172 88 55 92 72 63 213 189 88 143 ...
##  $ Platelets          : int  190 221 151 183 136 251 204 373 251 302 ...
##  $ Prothrombin        : num  12.2 10.6 12 10.3 10.9 11 9.7 11 11 11.5 ...
##  $ Stage              : num  4 3 4 4 3 3 3 3 2 4 ...
##  $ DrugD-penicillamine: num  1 1 1 1 0 0 0 0 1 0 ...
##  $ DrugPlacebo        : num  0 0 0 0 1 1 1 1 0 1 ...
##  $ SexM               : num  0 0 1 0 0 0 0 0 0 0 ...
##  $ AscitesY           : num  1 0 0 0 0 0 0 0 0 1 ...
##  $ HepatomegalyY      : num  1 1 0 1 1 1 1 0 0 0 ...
##  $ SpidersY           : num  1 1 0 1 1 0 0 0 1 1 ...
##  $ EdemaS             : num  0 0 1 1 0 0 0 0 0 0 ...
##  $ EdemaY             : num  1 0 0 0 0 0 0 0 0 1 ...
```

- One-hot encoding is applied to the categorical variables in the dataset to convert them into a format suitable for modeling. This technique creates binary columns for each category within a categorical variable, allowing the model to interpret the categorical data effectively.
- It is aslo called dummy encoding or one-of-K encoding. It is a technique used to convert categorical variables into a format that can be provided to ML algorithms to improve model performance.
- For my dataset, I used one-hot encoding to convert the categorical variables into binary columns because it is a common method for handling categorical variables in machine learning models. By converting categorical variables into binary columns, we can represent each category as a separate feature, allowing the model to capture the information encoded in the categories.
- I didn't apply one-hot encoding to the 'Status' column because it is the target variable, and one-hot encoding would create unnecessary additional columns for each category, which is not required for the target variable in classification tasks.

Skewness Identification and Transformation

```r
# Install the e1071 package if not already installed
if (!require(e1071)) {
  install.packages("e1071")
}
```

```
## Loading required package: e1071
```

```r
# Load the e1071 package
library(e1071)

# Check the skewness of numeric columns
sapply(cirrhosis[, sapply(cirrhosis, is.numeric)], skewness)
```

```
##              N_Days                 Age           Bilirubin         Cholesterol
##          0.46921558          0.08622782          2.69813743          4.25914487
##             Albumin              Copper            Alk_Phos                SGOT
##         -0.46417641          2.81638425          3.56976804          1.77173219
##       Trygliceride           Platelets         Prothrombin               Stage
##          3.24232286          0.63569052          2.21418180         -0.49506749
## DrugD-penicillamine         DrugPlacebo                SexM            AscitesY
##         -0.54358820          0.54358820          2.56325292          3.79129565
##       HepatomegalyY            SpidersY               EdemaS               EdemaY
##         -0.56491343          1.38025218          2.56325292          4.22157905
```

- Skewness is a measure of the asymmetry of the distribution of a variable. Positive skewness indicates a right-skewed distribution, while negative skewness indicates a left-skewed distribution.
- Positive skewness means that the right tail of the distribution is longer or fatter than the left tail, while negative skewness means that the left tail is longer or fatter than the right tail.
- A value > 0 indicates positive skew, a value < 0 indicates negative skew, and values close to 0 suggest a symmetric distribution.
- The skewness of numeric columns in the dataset is displayed above. Skewness correction is necessary to ensure that the data is normally distributed, which is a common assumption in many statistical tests and machine learning algorithms.
- The skewness of the numeric columns in the dataset is assessed to identify variables that may require transformation to correct skewness. Skewness correction is important for ensuring that the data meets the assumptions of statistical tests and modeling techniques.

Transformation of Skewed Variables

```r
# Define the columns to transform
cols_to_transform <- c(
  "Bilirubin", "Cholesterol", "Copper", "Alk_Phos", "SGOT",
  "Trygliceride", "Prothrombin"
)

# Loop over the columns and apply the log transformation
for (col in cols_to_transform) {
  # Apply the log transformation
  cirrhosis[[col]] <- log(cirrhosis[[col]] + 1)
}

# Apply square root transformation for 'platelets' column
cirrhosis$Platelets <- sqrt(cirrhosis$Platelets)

# Check the skewness of numeric columns after transformations
sapply(cirrhosis[, sapply(cirrhosis, is.numeric)], skewness)
```

```
##              N_Days                 Age           Bilirubin         Cholesterol
##          0.46921558          0.08622782          1.12849331          1.58420364
##             Albumin              Copper            Alk_Phos                SGOT
##         -0.46417641         -0.18896324          1.19959440         -0.14232209
##       Trygliceride           Platelets         Prothrombin               Stage
```

```
##          0.53624530          0.03373105          1.54625855         -0.49506749
## DrugD-penicillamine         DrugPlacebo                SexM           AscitesY
##         -0.54358820          0.54358820          2.56325292          3.79129565
##       HepatomegalyY            SpidersY               EdemaS               EdemaY
##         -0.56491343          1.38025218          2.56325292          4.22157905
```

- The log transformation is applied to the specified numeric columns to correct skewness and normalize the data. The log transformation is commonly used to stabilize variance and improve the normality of the data.
- The square root transformation is applied to the 'Platelets' column to address skewness and normalize the data. The square root transformation is useful for reducing the impact of extreme values and improving the distribution of the data.
- The skewness of the numeric columns is checked after the transformations to ensure that the data is closer to a normal distribution. Skewness correction is essential for preparing the data for statistical analysis and machine learning modeling.
- Binary variables are typically represented as 0 or 1, and they do not require transformation for skewness correction or standardization. But if applied to binary variables, the log transformation would not be meaningful as it would not change the distribution of the data and we should revert them back to their original scale.

Standardization of Numeric Data

```r
# Make a copy of the dataset before standardization
cirrhosis_standardized <- cirrhosis

# List numeric columns only (excluding the 'Status' which is factor)
numeric_cols <- sapply(cirrhosis_standardized, is.numeric)

# Standardize numeric columns manually without using scale() function
for (col in names(cirrhosis_standardized)[numeric_cols]) {
  cirrhosis_standardized[[col]] <- (cirrhosis_standardized[[col]] - mean(
    cirrhosis_standardized[[col]]
  )) / sd(cirrhosis_standardized[[col]])
}

# Summary of the dataset after standardization
summary(cirrhosis_standardized)
```

```
##      N_Days        Status         Age              Bilirubin
##  Min.   :-1.6989   C :232   Min.   :-2.3416   Min.   :-1.2196
##  1st Qu.:-0.7469   CL: 25   1st Qu.:-0.7571   1st Qu.:-0.7600
##  Median :-0.1700   D :161   Median : 0.0248   Median :-0.3537
##  Mean   : 0.0000            Mean   : 0.0000   Mean   : 0.0000
##  3rd Qu.: 0.6298            3rd Qu.: 0.7178   3rd Qu.: 0.5023
##  Max.   : 2.6046            Max.   : 2.6512   Max.   : 3.1654
##   Cholesterol         Albumin             Copper            Alk_Phos
##  Min.   :-2.7366   Min.   :-3.61775   Min.   :-3.84865   Min.   :-2.5064
##  1st Qu.:-0.4652   1st Qu.:-0.59990   1st Qu.:-0.47425   1st Qu.:-0.5018
##  Median :-0.1177   Median : 0.07662   Median : 0.02627   Median :-0.1599
##  Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.0000
```

```
##  3rd Qu.: 0.2052   3rd Qu.: 0.64136   3rd Qu.: 0.48419   3rd Qu.: 0.3267
##  Max.   : 4.7288   Max.   : 2.68856   Max.   : 3.00923   Max.   : 3.6707
##       SGOT          Tryglicerides        Platelets         Prothrombin
##  Min.   :-3.70049   Min.   :-3.26892   Min.   :-2.58249   Min.   :-1.9297
##  1st Qu.:-0.53672   1st Qu.:-0.42036   1st Qu.:-0.64116   1st Qu.:-0.7525
##  Median : 0.06108   Median :-0.07184   Median : 0.03516   Median :-0.0966
##  Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.0000
##  3rd Qu.: 0.49702   3rd Qu.: 0.38514   3rd Qu.: 0.66561   3rd Qu.: 0.4246
##  Max.   : 3.65086   Max.   : 4.60422   Max.   : 3.65121   Max.   : 5.9976
##      Stage           DrugD-penicillamine  DrugPlacebo           SexM
##  Min.   :-2.31126   Min.   :-1.3077     Min.   :-0.7628   Min.   :-0.3426
##  1st Qu.:-1.16929   1st Qu.:-1.3077     1st Qu.:-0.7628   1st Qu.:-0.3426
##  Median :-0.02732   Median : 0.7628     Median :-0.7628   Median :-0.3426
##  Mean   : 0.00000   Mean   : 0.0000     Mean   : 0.0000   Mean   : 0.0000
##  3rd Qu.: 1.11465   3rd Qu.: 0.7628     3rd Qu.: 1.3077   3rd Qu.:-0.3426
##  Max.   : 1.11465   Max.   : 0.7628     Max.   : 1.3077   Max.   : 2.9120
##     AscitesY       HepatomegalyY       SpidersY           EdemaS
##  Min.   :-0.2465   Min.   :-1.321   Min.   :-0.5232   Min.   :-0.3426
##  1st Qu.:-0.2465   1st Qu.:-1.321   1st Qu.:-0.5232   1st Qu.:-0.3426
##  Median :-0.2465   Median : 0.755   Median :-0.5232   Median :-0.3426
##  Mean   : 0.0000   Mean   : 0.000   Mean   : 0.0000   Mean   : 0.0000
##  3rd Qu.:-0.2465   3rd Qu.: 0.755   3rd Qu.:-0.5232   3rd Qu.:-0.3426
##  Max.   : 4.0469   Max.   : 0.755   Max.   : 1.9068   Max.   : 2.9120
##      EdemaY
##  Min.   :-0.2239
##  1st Qu.:-0.2239
##  Median :-0.2239
##  Mean   : 0.0000
##  3rd Qu.:-0.2239
##  Max.   : 4.4556
```

```
# Rebind the 'Status' column to the standardized dataset
cirrhosis_standardized$Status <- cirrhosis$Status

# Check the structure of the standardized dataset
str(cirrhosis_standardized)
```

```
## 'data.frame':    418 obs. of  21 variables:
##  $ N_Days             : num  -1.37397 2.33754 -0.81996 0.00653 -0.37457 ...
##  $ Status             : Factor w/ 3 levels "C","CL","D": 3 1 3 3 2 3 1 3 3 3 ...
##  $ Age                : num  0.768 0.546 1.85 0.383 -1.21 ...
##  $ Bilirubin          : num  2.281 -0.542 -0.354 -0.136 0.502 ...
##  $ Cholesterol        : num  -0.59 -0.186 -1.68 -0.776 -0.405 ...
##  $ Albumin            : num  -2.1118 1.512 -0.041 -2.253 0.0766 ...
##  $ Copper             : num  1.108 -0.4 1.533 -0.16 0.984 ...
##  $ Alk_Phos           : num  0.336 2.667 -1.583 2.365 -1.164 ...
##  $ SGOT               : num  0.5387 0.0343 -0.396 -1.5816 0.0259 ...
##  $ Tryglicerides      : num  1.196 -0.628 -1.9 -0.507 -1.172 ...
##  $ Platelets          : num  -0.641 -0.286 -1.133 -0.725 -1.338 ...
##  $ Prothrombin        : num  1.4992 -0.0966 1.3107 -0.4202 0.2187 ...
##  $ Stage              : num  1.1147 -0.0273 1.1147 1.1147 -0.0273 ...
##  $ DrugD-penicillamine: num  0.763 0.763 0.763 0.763 -1.308 ...
##  $ DrugPlacebo        : num  -0.763 -0.763 -0.763 -0.763 1.308 ...
##  $ SexM               : num  -0.343 -0.343 2.912 -0.343 -0.343 ...
```

```
##  $ AscitesY           : num  4.047 -0.247 -0.247 -0.247 -0.247 ...
##  $ HepatomegalyY      : num  0.755 0.755 -1.321 0.755 0.755 ...
##  $ SpidersY           : num  1.907 1.907 -0.523 1.907 1.907 ...
##  $ EdemaS             : num  -0.343 -0.343 2.912 2.912 -0.343 ...
##  $ EdemaY             : num  4.456 -0.224 -0.224 -0.224 -0.224 ...
```

- Standardization is applied to the numeric columns in the dataset to ensure that all variables are on a common scale. Standardization is important for many machine learning algorithms that are sensitive to the scale of the input features.
- Standardization involves transforming the data so that it has a mean of 0 and a standard deviation of 1. This process helps to align the variables on a common scale, making it easier to compare and interpret their values.
- The summary of the dataset is rechecked to confirm that the standardization process has been applied successfully. The mean and standard deviation of the numeric variables should be close to 0 and 1, respectively, after standardization.
- Logistic Regression and SVM: These models often benefit from standardization, especially SVM, which is sensitive to the magnitude of input features and can behave unpredictably if features are not on the same scale. Standardization helps these algorithms converge faster and perform better by transforming the data into a scale where the features contribute equally.
- Random Forest: This model is generally less sensitive to the scale of the features because it uses rule-based splitting. Thus, normalization or standardization is not typically necessary for tree-based models like random forests.

Reverting Binary Variables to Original Scale

```r
# Define binary variables for reversion
binary_vars <- c(
  "DrugD-penicillamine", "DrugPlacebo", "SexM", "AscitesY",
  "HepatomegalyY", "SpidersY", "EdemaY", "EdemaS"
)

# Revert binary variables to original scale
cirrhosis_standardized[binary_vars] <- ifelse(
  cirrhosis_standardized[binary_vars] <= 0, 0, 1
)

# Check summary again for these binary variables
summary(cirrhosis_standardized)
```

```
##      N_Days        Status        Age             Bilirubin
##  Min.   :-1.6989   C :232   Min.   :-2.3416   Min.   :-1.2196
##  1st Qu.:-0.7469   CL: 25   1st Qu.:-0.7571   1st Qu.:-0.7600
##  Median :-0.1700   D :161   Median : 0.0248   Median :-0.3537
##  Mean   : 0.0000            Mean   : 0.0000   Mean   : 0.0000
##  3rd Qu.: 0.6298            3rd Qu.: 0.7178   3rd Qu.: 0.5023
##  Max.   : 2.6046            Max.   : 2.6512   Max.   : 3.1654
##   Cholesterol         Albumin             Copper            Alk_Phos
##  Min.   :-2.7366   Min.   :-3.61775   Min.   :-3.84865   Min.   :-2.5064
##  1st Qu.:-0.4652   1st Qu.:-0.59990   1st Qu.:-0.47425   1st Qu.:-0.5018
```

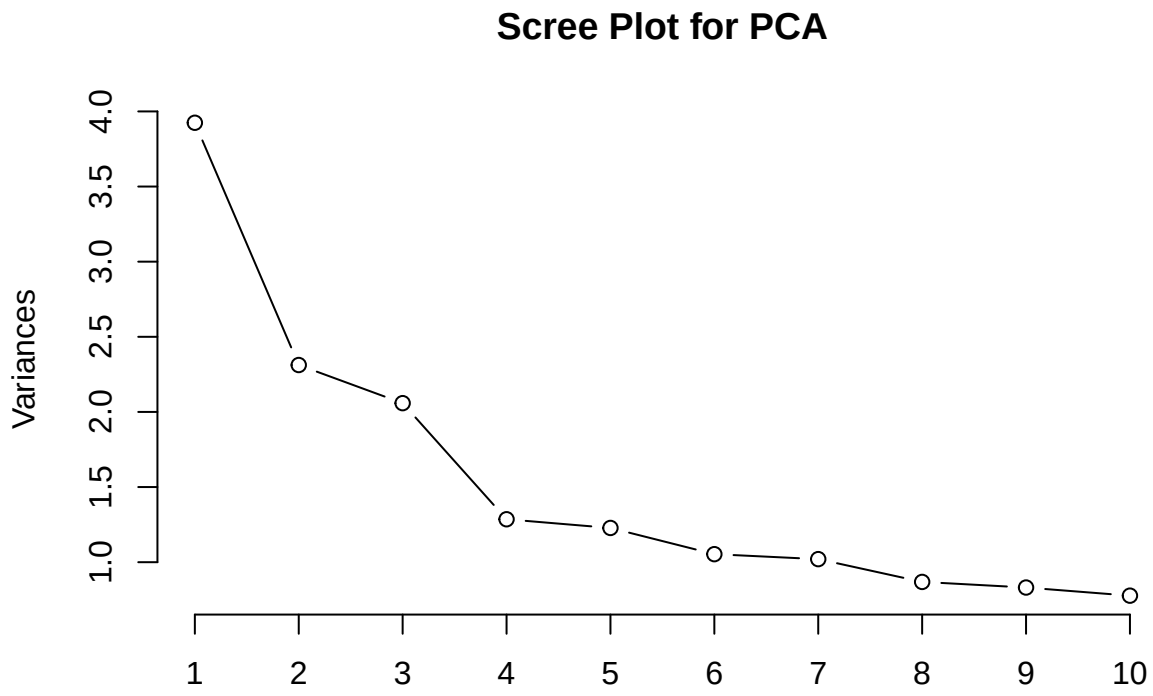
```
##  Median :-0.1177   Median : 0.07662   Median : 0.02627   Median :-0.1599
##  Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.0000
##  3rd Qu.: 0.2052   3rd Qu.: 0.64136   3rd Qu.: 0.48419   3rd Qu.: 0.3267
##  Max.   : 4.7288   Max.   : 2.68856   Max.   : 3.00923   Max.   : 3.6707
##       SGOT          Trygliceride s       Platelets         Prothrombin
##  Min.   :-3.70049   Min.   :-3.26892   Min.   :-2.58249   Min.   :-1.9297
##  1st Qu.:-0.53672   1st Qu.:-0.42036   1st Qu.:-0.64116   1st Qu.:-0.7525
##  Median : 0.06108   Median :-0.07184   Median : 0.03516   Median :-0.0966
##  Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.0000
##  3rd Qu.: 0.49702   3rd Qu.: 0.38514   3rd Qu.: 0.66561   3rd Qu.: 0.4246
##  Max.   : 3.65086   Max.   : 4.60422   Max.   : 3.65121   Max.   : 5.9976
##      Stage          DrugD-penicillamine  DrugPlacebo          SexM
##  Min.   :-2.31126   Min.   :0.0000      Min.   :0.0000   Min.   :0.0000
##  1st Qu.:-1.16929   1st Qu.:0.0000      1st Qu.:0.0000   1st Qu.:0.0000
##  Median :-0.02732   Median :1.0000      Median :0.0000   Median :0.0000
##  Mean   : 0.00000   Mean   :0.6316      Mean   :0.3684   Mean   :0.1053
##  3rd Qu.: 1.11465   3rd Qu.:1.0000      3rd Qu.:1.0000   3rd Qu.:0.0000
##  Max.   : 1.11465   Max.   :1.0000      Max.   :1.0000   Max.   :1.0000
##     AscitesY       HepatomegalyY       SpidersY          EdemaS
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :0.00000   Median :1.0000   Median :0.0000   Median :0.0000
##  Mean   :0.05742   Mean   :0.6364   Mean   :0.2153   Mean   :0.1053
##  3rd Qu.:0.00000   3rd Qu.:1.0000   3rd Qu.:0.0000   3rd Qu.:0.0000
##  Max.   :1.00000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##      EdemaY
##  Min.   :0.00000
##  1st Qu.:0.00000
##  Median :0.00000
##  Mean   :0.04785
##  3rd Qu.:0.00000
##  Max.   :1.00000
```

- After standardization of the numeric variables, the binary variables are reverted to their original scale by converting the standardized values back to 0 or 1. This step ensures that the binary variables are in their original format for modeling and interpretation.
- The summary of the binary variables is checked to confirm that the values have been successfully reverted to 0 or 1. The summary should show the counts of 0s and 1s for each binary variable, indicating that the reversion process was completed accurately.

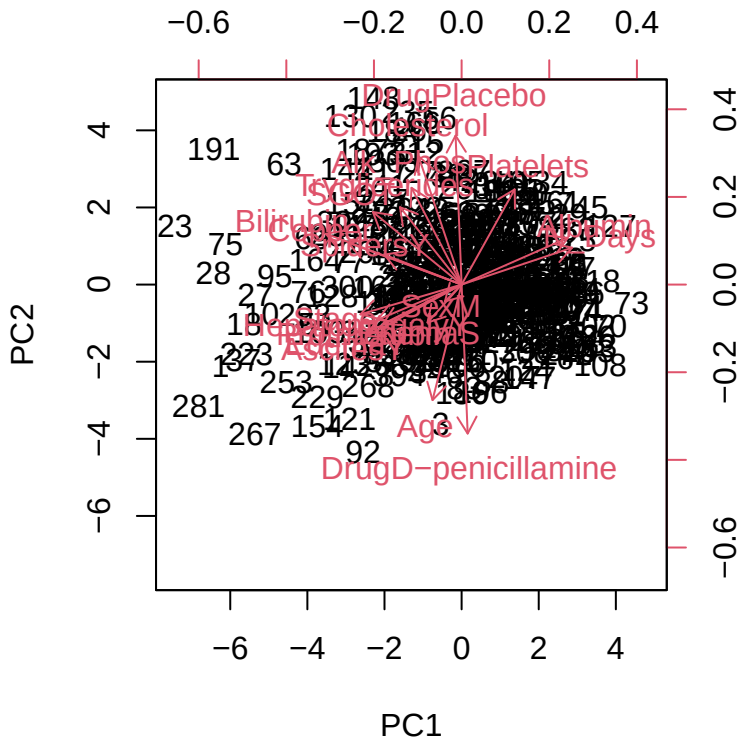
Dimensionality Reduction using PCA

```r
# Load necessary library
library(stats)

# Standardizing data (important for PCA)
cirrhosis_standardized_numeric <- cirrhosis_standardized[, sapply(
  cirrhosis_standardized, is.numeric
)]

# Apply PCA to the standardized numeric data
```

```r
pca_result <- prcomp(cirrhosis_standardized_numeric, scale = TRUE)

# Scree plot to visualize explained variance
plot(pca_result, type = "l", main = "Scree Plot for PCA")
```




```r
# Biplot to visualize the relationship between variables and components
biplot(pca_result, scale = 0)
```




```
# Summary of PCA results
summary(pca_result)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5     PC6     PC7
## Standard deviation     1.9810 1.5206 1.4347 1.13391 1.10808 1.02633 1.01009
## Proportion of Variance 0.1962 0.1156 0.1029 0.06429 0.06139 0.05267 0.05101
## Cumulative Proportion  0.1962 0.3118 0.4148 0.47904 0.54043 0.59310 0.64411
##                           PC8     PC9    PC10    PC11    PC12    PC13    PC14
## Standard deviation     0.93202 0.91173 0.88162 0.86109 0.82357 0.76982 0.75046
## Proportion of Variance 0.04343 0.04156 0.03886 0.03707 0.03391 0.02963 0.02816
## Cumulative Proportion  0.68755 0.72911 0.76797 0.80505 0.83896 0.86859 0.89675
##                           PC15    PC16    PC17    PC18    PC19      PC20
## Standard deviation     0.74258 0.66562 0.63425 0.58546 0.57054 4.448e-16
## Proportion of Variance 0.02757 0.02215 0.02011 0.01714 0.01628 0.000e+00
## Cumulative Proportion  0.92432 0.94647 0.96659 0.98372 1.00000 1.000e+00
```

- Principal Component Analysis (PCA) is applied to the standardized numeric data to reduce the dimensionality of the dataset and identify the most important components that explain the variance in the data.
- The scree plot above shows the explained variance of each principal component, helping to determine the number of components to retain for analysis. The scree plot is useful for identifying the principal components that capture the most variance in the data.

- The biplot displays the relationship between the principal components and the original variables. It helps visualize the contribution of each variable to the principal components and identify patterns in the data.
- PCA is a powerful technique for dimensionality reduction and feature extraction, allowing for the identification of the most important components that explain the variance in the data.

Model Construction and Evaluation

Splitting the Data into Training and Testing Sets

```
# Split the data into training and testing sets
set.seed(123)

# Shuffle the rows to randomize the data
cirrhosis_standardized <- cirrhosis_standardized[sample(
  nrow(cirrhosis_standardized)
), ]

# Split the data into training 80% and testing 20%
train_index <- 1:round(0.8 * nrow(cirrhosis_standardized))

# Create training and testing datasets
train_data <- cirrhosis_standardized[train_index, ]
validation_data <- cirrhosis_standardized[-train_index, ]

# Check the dimensions of the training and testing sets
nrow(train_data)
```

```
## [1] 334
```

```
ncol(train_data)
```

```
## [1] 21
```

```
nrow(validation_data)
```

```
## [1] 84
```

```
ncol(validation_data)
```

```
## [1] 21
```

- The data is split into training and validation sets to train the model on a subset of the data and evaluate its performance on a separate subset. The training set contains 80% of the observations, while the validation set contains the remaining 20%.
- The data is shuffled to randomize the order of the observations before splitting to ensure that the training and validation sets are representative of the overall dataset.
- The dimensions of the training and validation sets are checked to confirm that the data has been split correctly and that the training set contains 80% of the observations.
- The training data is 334 rows by 21 columns, and the validation data is 84 rows by 21 columns.

Logistic Regression Model

```r
# Load caret and set up train control
library(caret)
```

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

```r
# Set up cross-validation with 10 folds
train_control <- trainControl(
  method = "cv", number = 10,
  savePredictions = "final", classProbs = TRUE
)

# Train a model using multinomial logistic regression
multinom_model <- train(Status ~ .,
  data = train_data,
  method = "multinom", trControl = train_control, trace = FALSE
)

# Summary of the model
print(multinom_model)
```

```
## Penalized Multinomial Regression
##
## 334 samples
##  20 predictor
##   3 classes: 'C', 'CL', 'D'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 301, 300, 301, 302, 300, 301, ...
## Resampling results across tuning parameters:
##
##   decay  Accuracy   Kappa
##   0e+00  0.7181261  0.4631305
##   1e-04  0.7181261  0.4631305
##   1e-01  0.7181261  0.4607923
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was decay = 0.1.
```

```r
# Evaluate the model for linear regression
lr_predictions <- predict(multinom_model, newdata = validation_data)

# Generate a confusion matrix for linear regression
confusion_matrix_lr <- confusionMatrix(lr_predictions, validation_data$Status)

# Print the confusion matrix
print(confusion_matrix_lr)
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction  C CL  D
##         C  43  1 10
##         CL  0  0  0
##         D   4  0 26
##
## Overall Statistics
##
##                Accuracy : 0.8214
##                  95% CI : (0.7226, 0.8965)
##     No Information Rate : 0.5595
##     P-Value [Acc > NIR] : 3.651e-07
##
##                   Kappa : 0.6335
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: C Class: CL Class: D
## Sensitivity            0.9149    0.0000   0.7222
## Specificity            0.7027    1.0000   0.9167
## Pos Pred Value         0.7963       NaN   0.8667
## Neg Pred Value         0.8667    0.9881   0.8148
## Prevalence             0.5595    0.0119   0.4286
## Detection Rate         0.5119    0.0000   0.3095
## Detection Prevalence   0.6429    0.0000   0.3571
## Balanced Accuracy      0.8088    0.5000   0.8194
```

- A multinomial logistic regression model is trained on the training data to predict the survival status of patients with cirrhosis. The model is built using the `multinom` function from the `nnet` package.
- The Accuracy of the model is 0.82 which indicates the proportion of correct predictions made by the model.
- The CI of the model is 0.72 to 0.9 which provides a range of values within which the true accuracy is likely to fall.
- The confusion matrix shows the number of correct and incorrect predictions broken down by each class. For example, the model correctly predicted 43 instances of class C, but incorrectly predicted 10 instances of class D as class C.
- A Kappa of 1 indicates perfect agreement, while a Kappa of 0 indicates agreement equivalent to chance. A Kappa of 0.6335 suggests moderate agreement.

Support Vector Machine (SVM) Model

```
# Load the e1071 package for SVM
library(e1071)

# SVM model using the e1071 package
svm_model <- train(Status ~ .,
```

```
  data = train_data,
  method = "svmRadial", trControl = train_control, trace = FALSE
)

# Summary of the model
print(svm_model)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 334 samples
##  20 predictor
##   3 classes: 'C', 'CL', 'D'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 299, 302, 300, 301, 301, 300, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.25  0.7214007  0.4646697
##   0.50  0.7303134  0.4769781
##   1.00  0.7333488  0.4806072
##
## Tuning parameter 'sigma' was held constant at a value of 0.03505953
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.03505953 and C = 1.
```

```
# Evaluate the model for SVM
svm_predictions <- predict(svm_model, newdata = validation_data)

# Generate a confusion matrix
confusion_matrix_svm <- confusionMatrix(svm_predictions, validation_data$Status)

# Print the confusion matrix
print(confusion_matrix_svm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  C CL  D
##         C  43  1 10
##         CL  0  0  0
##         D   4  0 26
##
## Overall Statistics
##
##                Accuracy : 0.8214
##                  95% CI : (0.7226, 0.8965)
##     No Information Rate : 0.5595
##     P-Value [Acc > NIR] : 3.651e-07
##
##                   Kappa : 0.6335
##
```

```
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: C Class: CL Class: D
## Sensitivity            0.9149    0.0000   0.7222
## Specificity            0.7027    1.0000   0.9167
## Pos Pred Value         0.7963       NaN   0.8667
## Neg Pred Value         0.8667    0.9881   0.8148
## Prevalence             0.5595    0.0119   0.4286
## Detection Rate         0.5119    0.0000   0.3095
## Detection Prevalence   0.6429    0.0000   0.3571
## Balanced Accuracy      0.8088    0.5000   0.8194
```

- A Support Vector Machine (SVM) model is trained on the training data to predict the survival status of patients with cirrhosis. The model is built using the `svmRadial` method from the `e1071` package.
- The Accuracy of the model is 0.82 which indicates the proportion of correct predictions made by the model.
- The CI of the model is 0.72 to 0.9 which provides a range of values within which the true accuracy is likely to fall.
- This table shows the number of correct and incorrect predictions made by your model, broken down by each class. For example, the model correctly predicted 43 instances of class C, but incorrectly predicted 10 instances of class D as class C.
- The very small p-value (3.651e-07) suggests that the model's accuracy is significantly better than the No Information Rate.

Random Forest Model

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

# Prepare training control
train_control <- trainControl(
  method = "cv",
  # Number of folds in cross-validation
  number = 10,
  savePredictions = "final",
  classProbs = TRUE,
  # Turn off training messages for cleaner output
  verboseIter = FALSE
)

# Define a tuning grid for Random Forest specifically with 'mtry'
tuning_grid <- expand.grid(
  mtry = c(sqrt(ncol(train_data)), ncol(train_data) / 3, ncol(train_data) / 2)
)

# Random forest model using the randomForest package
rf_model <- train(Status ~ .,
```

```
  data = train_data, method = "rf",
  trControl = train_control, trace = FALSE,
  tuneGrid = tuning_grid,
  metric = "Accuracy"
)

# Summary of the model
print(rf_model)
```

```
## Random Forest
##
## 334 samples
##  20 predictor
##   3 classes: 'C', 'CL', 'D'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 302, 299, 301, 300, 300, 302, ...
## Resampling results across tuning parameters:
##
##   mtry       Accuracy   Kappa
##    4.582576  0.7420052  0.4989542
##    7.000000  0.7300678  0.4790569
##   10.500000  0.7362231  0.4909942
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 4.582576.
```

```
# Evaluate the model
rf_predictions <- predict(rf_model, newdata = validation_data)

# Generate a confusion matrix
confusion_matrix_rf <- confusionMatrix(rf_predictions, validation_data$Status)

# Print the confusion matrix
print(confusion_matrix_rf)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  C CL  D
##         C  39  1  8
##         CL  0  0  0
##         D   8  0 28
##
## Overall Statistics
##
##                Accuracy : 0.7976
##                  95% CI : (0.6959, 0.8775)
##     No Information Rate : 0.5595
##     P-Value [Acc > NIR] : 4.147e-06
##
##                   Kappa : 0.5925
```

```
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: C Class: CL Class: D
## Sensitivity            0.8298    0.0000   0.7778
## Specificity            0.7568    1.0000   0.8333
## Pos Pred Value         0.8125       NaN   0.7778
## Neg Pred Value         0.7778    0.9881   0.8333
## Prevalence             0.5595    0.0119   0.4286
## Detection Rate         0.4643    0.0000   0.3333
## Detection Prevalence   0.5714    0.0000   0.4286
## Balanced Accuracy      0.7933    0.5000   0.8056
```

- A Random Forest model is trained on the training data to predict the survival status of patients with cirrhosis. The model is built using the `rf` method from the `randomForest` package.
- The summary of the model provides information about the number of trees, mtry, and other details of the Random Forest model. This information helps assess the complexity and performance of the model.
- The Accuracy of the model is 0.8 which indicates the proportion of correct predictions made by the model.
- The CI of the model is 0.7 to 0.88 which provides a range of values within which the true accuracy is likely to fall.
- Predictions are made on the validation data using the trained Random Forest model, and a confusion matrix is generated to evaluate the model's performance. The confusion matrix shows the counts of true positive, true negative, false positive, and false negative predictions.
- The Kappa statistic measures the agreement between the observed and predicted classes, with a value of 1 indicating perfect agreement and 0 indicating agreement equivalent to chance. A Kappa of 0.5925 suggests moderate agreement between the predicted and observed classes.

Model Evaluation

ROC Curve and AUC Calculation for Logistic Regression Model

```r
# Install and load the pROC package if not already installed
if (!require(pROC)) {
  install.packages("pROC", repos = "http://cran.rstudio.com/")
}
```

```
## Loading required package: pROC
```

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

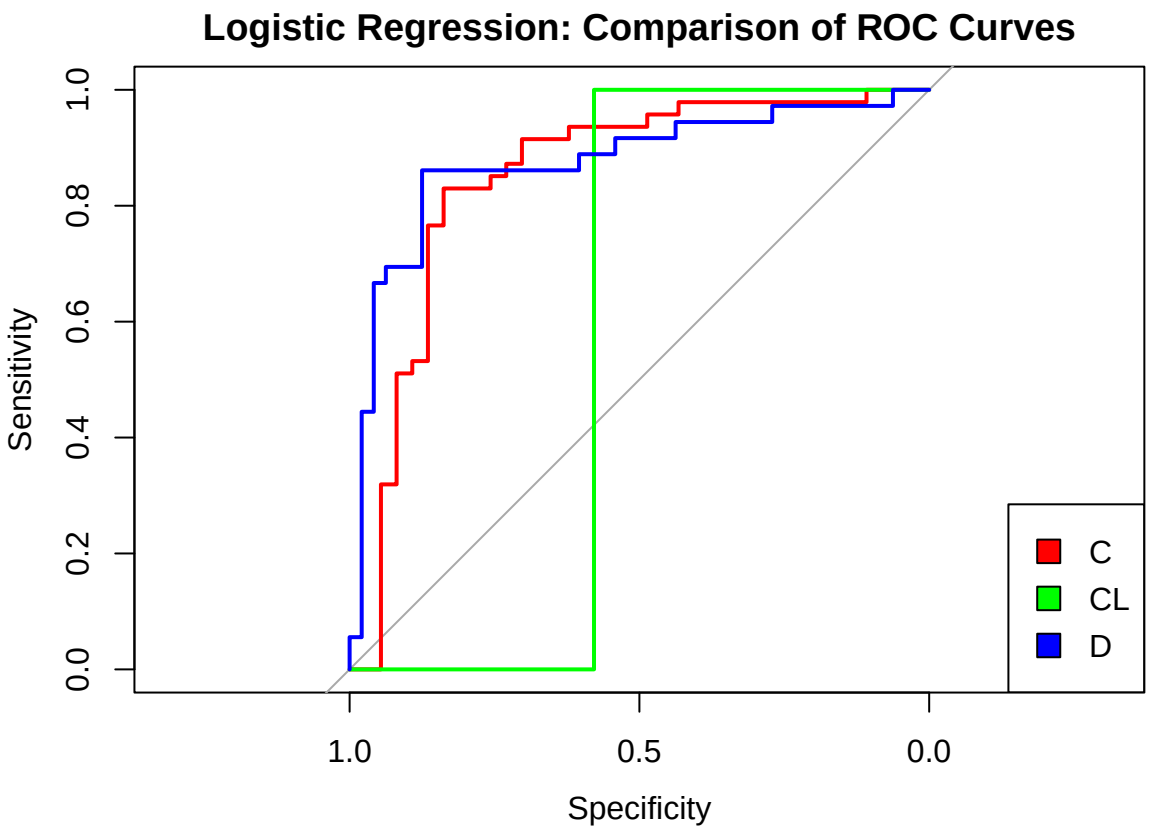
```
##
## Attaching package: 'pROC'
```

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

```
library(pROC)

# Predict class probabilities for the logistic regression model
prob_predictions <- predict(multinom_model,
  newdata = validation_data, type = "prob"
)

# Compute ROC curve for each class in logistic regression model
roc_list <- list()
class_levels <- levels(validation_data$Status)

for (class in class_levels) {
  true_values <- as.numeric(validation_data$Status == class)
  roc_curve <- roc(true_values, prob_predictions[, class],
    plot = FALSE
  )
  roc_list[[class]] <- roc_curve
  print(paste("AUC for", class, "=", auc(roc_curve)))
}
```

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

## Setting direction: controls < cases

## [1] "AUC for C = 0.84933870040253"

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

## Setting direction: controls > cases

## [1] "AUC for CL = 0.578313253012048"

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

## Setting direction: controls < cases

## [1] "AUC for D = 0.874421296296297"
```

```
# Plot ROC curves
colors <- rainbow(length(class_levels))
plot(roc_list[[1]],
  col = colors[1],
  main = "Logistic Regression: Comparison of ROC Curves"
)
for (i in 2:length(class_levels)) {
  plot(roc_list[[i]], add = TRUE, col = colors[i])
}
legend("bottomright", class_levels, fill = colors)
```



- The ROC curves for each class in the logistic regression model are plotted to visualize the trade-off between true positive rate (sensitivity) and false positive rate (1-specificity) for different classification thresholds.
- The Area Under the Curve (AUC) is calculated for each class, providing a measure of the model's ability to distinguish between the positive and negative classes. A higher AUC value indicates better performance in terms of classification accuracy.
- The ROC curves and AUC values help evaluate the performance of the logistic regression model in predicting the survival status of patients with cirrhosis.
- AUC for C is 0.85, for CL is 0.58, and for D is 0.87.

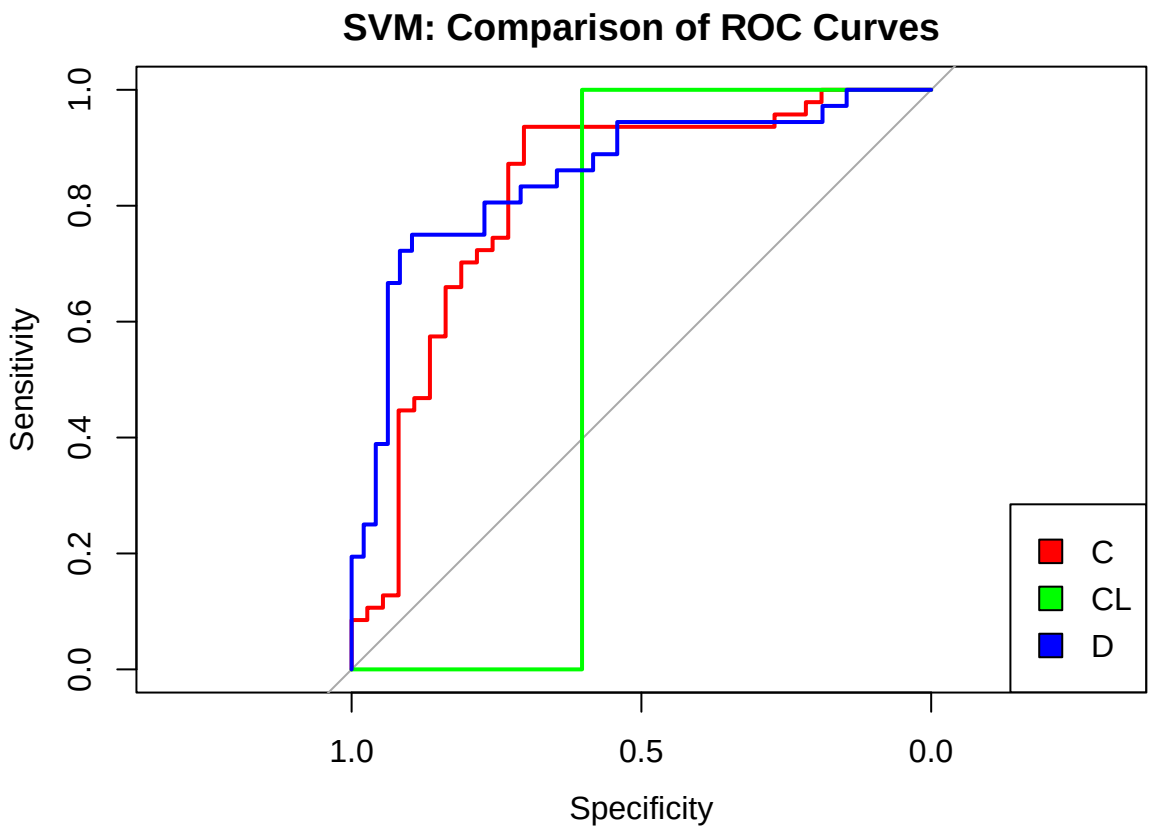
ROC Curve and AUC for SVM Model

```
# Load the pROC package if not already loaded
library(pROC)

# Predict class probabilities for the SVM model
svm_prob_predictions <- predict(svm_model,
  newdata = validation_data, type = "prob"
)

# Compute ROC curve for each class in SVM model
svm_roc_list <- list()
```

```r
class_levels <- levels(validation_data$Status)

for (class in class_levels) {
  true_values_svm <- as.numeric(validation_data$Status == class)
  roc_curve_svm <- roc(true_values_svm, svm_prob_predictions[, class],
    plot = FALSE
  )
  svm_roc_list[[class]] <- roc_curve_svm
  print(paste("SVM AUC for", class, "=", auc(roc_curve_svm)))
}
```

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

## Setting direction: controls < cases

## [1] "SVM AUC for C = 0.821161587119034"

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

## Setting direction: controls > cases

## [1] "SVM AUC for CL = 0.602409638554217"

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

## Setting direction: controls < cases

## [1] "SVM AUC for D = 0.854166666666667"
```

```r
# Plot ROC curves
colors <- rainbow(length(class_levels))
plot(svm_roc_list[[1]], col = colors[1], main = "SVM: Comparison of ROC Curves")
for (i in 2:length(class_levels)) {
  plot(svm_roc_list[[i]], add = TRUE, col = colors[i])
}
legend("bottomright", class_levels, fill = colors)
```



- The ROC curves for each class in the SVM model are plotted to visualize the model's performance in distinguishing between the positive and negative classes.
- The Area Under the Curve (AUC) is calculated for each class, providing a measure of the model's ability to classify the survival status of patients with cirrhosis.
- The ROC curves and AUC values help evaluate the performance of the SVM model in predicting the survival status of patients with cirrhosis.
- AUC for C is 0.82, for CL is 0.6, and for D is 0.85.

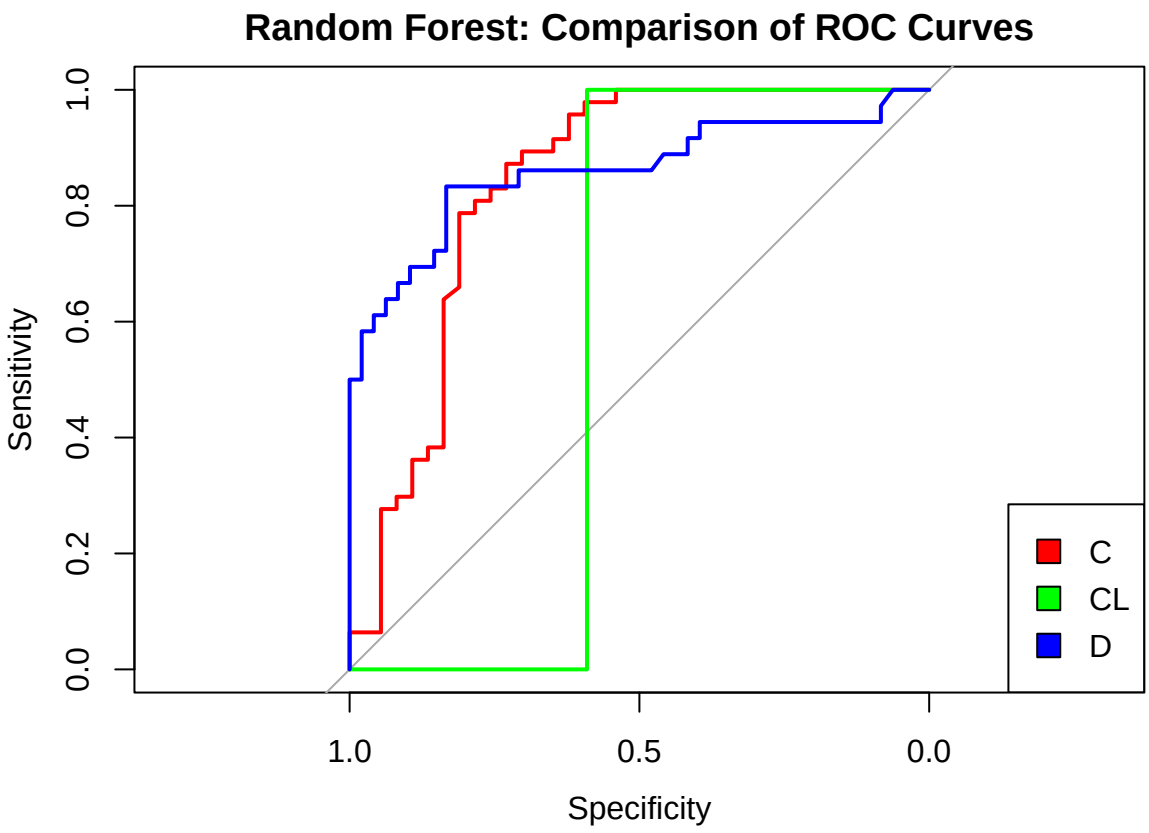
ROC Curve and AUC for Random Forest Model

```
# Load the pROC package if not already loaded
library(pROC)

# Predict class probabilities for the Random Forest model
rf_prob_predictions <- predict(rf_model,
  newdata = validation_data, type = "prob"
)

# Compute ROC curve for each class in Random Forest model
rf_roc_list <- list()
class_levels <- levels(validation_data$Status)
```

```r
for (class in class_levels) {
  true_values_rf <- as.numeric(validation_data$Status == class)
  roc_curve_rf <- roc(true_values_rf, rf_prob_predictions[, class],
    plot = FALSE
  )
  rf_roc_list[[class]] <- roc_curve_rf
  print(paste("Random Forest AUC for", class, "=", auc(roc_curve_rf)))
}
```

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

## Setting direction: controls < cases

## [1] "Random Forest AUC for C = 0.838125359401955"

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

## [1] "Random Forest AUC for CL = 0.590361445783133"

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

## [1] "Random Forest AUC for D = 0.860532407407408"
```

```r
# Plot ROC curves
colors <- rainbow(length(class_levels))
plot(rf_roc_list[[1]],
  col = colors[1],
  main = "Random Forest: Comparison of ROC Curves"
)
for (i in 2:length(class_levels)) {
  plot(rf_roc_list[[i]], add = TRUE, col = colors[i])
}
legend("bottomright", class_levels, fill = colors)
```



- The ROC curves for each class in the Random Forest model are plotted to visualize the model's performance in distinguishing between the positive and negative classes.
- The Area Under the Curve (AUC) is calculated for each class, providing a measure of the model's ability to classify the survival status of patients with cirrhosis.
- The ROC curves and AUC values help evaluate the performance of the Random Forest model in predicting the survival status of patients with cirrhosis.
- AUC for C is 0.84, for CL is 0.59, and for D is 0.86.

Precision, Recall, and F1-Score Calculation

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

# Generate a confusion matrix for Logistic Regression
cm_lr <- confusionMatrix(lr_predictions, validation_data$Status)

# Calculate macro-averaged precision, recall, and F1 score
macro_precision <- mean(cm_lr$byClass[, "Precision"], na.rm = TRUE)
macro_recall <- mean(cm_lr$byClass[, "Recall"], na.rm = TRUE)
macro_F1 <- mean(cm_lr$byClass[, "F1"], na.rm = TRUE)
```

```r
# Print the results for Logistic Regression
print(paste("Macro-averaged Precision:", macro_precision))
```

```
## [1] "Macro-averaged Precision: 0.831481481481481"
```

```r
print(paste("Macro-averaged Recall:", macro_recall))
```

```
## [1] "Macro-averaged Recall: 0.545705279747833"
```

```r
print(paste("Macro-averaged F1 Score:", macro_F1))
```

```
## [1] "Macro-averaged F1 Score: 0.81968196819682"
```

```r
# Generate a confusion matrix for SVM
cm_svm <- confusionMatrix(svm_predictions, validation_data$Status)

# Calculate macro-averaged precision, recall, and F1 score for SVM
macro_precision_svm <- mean(cm_svm$byClass[, "Precision"], na.rm = TRUE)
macro_recall_svm <- mean(cm_svm$byClass[, "Recall"], na.rm = TRUE)
macro_F1_svm <- mean(cm_svm$byClass[, "F1"], na.rm = TRUE)

# Print the results for SVM
print(paste("SVM Macro-averaged Precision:", macro_precision_svm))
```

```
## [1] "SVM Macro-averaged Precision: 0.831481481481481"
```

```r
print(paste("SVM Macro-averaged Recall:", macro_recall_svm))
```

```
## [1] "SVM Macro-averaged Recall: 0.545705279747833"
```

```r
print(paste("SVM Macro-averaged F1 Score:", macro_F1_svm))
```

```
## [1] "SVM Macro-averaged F1 Score: 0.81968196819682"
```

```r
# Generate a confusion matrix for Random Forest
cm_rf <- confusionMatrix(rf_predictions, validation_data$Status)

# Calculate macro-averaged precision, recall, and F1 score for Random Forest
macro_precision_rf <- mean(cm_rf$byClass[, "Precision"], na.rm = TRUE)
macro_recall_rf <- mean(cm_rf$byClass[, "Recall"], na.rm = TRUE)
macro_F1_rf <- mean(cm_rf$byClass[, "F1"], na.rm = TRUE)

# Print the results for Random Forest
print(paste("Random Forest Macro-averaged Precision:", macro_precision_rf))
```

```
## [1] "Random Forest Macro-averaged Precision: 0.795138888888889"
```

```r
print(paste("Random Forest Macro-averaged Recall:", macro_recall_rf))
```

```
## [1] "Random Forest Macro-averaged Recall: 0.53585500394011"
```

```r
print(paste("Random Forest Macro-averaged F1 Score:", macro_F1_rf))
```

```
## [1] "Random Forest Macro-averaged F1 Score: 0.799415204678363"
```

- The Macro-averaged Precision for both Logistic Regression and SVM is the same (approximately 0.8315), suggesting that when these models predict a patient's status, they are correct about 83.15% of the time across the different classes.
- The Random Forest model has a slightly lower Macro-averaged Precision of approximately 0.7951, meaning it is correct 79.51% of the time when predicting a patient's status.
- The Recall (or Sensitivity) for both Logistic Regression and SVM is also the same (approximately 0.5457), indicating that these models correctly identify 54.57% of all positive instances across the different classes.
- The Random Forest model's Recall is slightly lower, at approximately 0.5359, which means it correctly identifies 53.59% of all positive instances.
- The F1 Score is a harmonic mean of Precision and Recall and is a measure of a test's accuracy. Both Logistic Regression and SVM have a Macro-averaged F1 Score of approximately 0.8197, which is quite high, suggesting a good balance between Precision and Recall.
- Random Forest has a slightly lower F1 Score of approximately 0.7994, but it is still relatively high, indicating a reasonable balance between Precision and Recall for this model as well.
- Overall, the Logistic Regression and SVM models are performing similarly in terms of Precision, Recall, and F1 Score, and both are performing slightly better than the Random Forest model based on these metrics.

Evaluation of fit using holdout method

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

# Set up the train control for the holdout method
train_control_holdout <- trainControl(
  method = "LGOCV", p = 0.8,
  savePredictions = "final", classProbs = TRUE
)

# Train the models using the holdout method
multinom_model_holdout <- train(Status ~ .,
  data = train_data,
  method = "multinom", trControl = train_control_holdout, trace = FALSE
)

svm_model_holdout <- train(Status ~ .,
  data = train_data,
```

```
  method = "svmRadial", trControl = train_control_holdout, trace = FALSE
)

rf_model_holdout <- train(Status ~ .,
  data = train_data,
  method = "rf", trControl = train_control_holdout, trace = FALSE
)

# Summarize the models
print(multinom_model_holdout)
```

```
## Penalized Multinomial Regression
##
## 334 samples
##  20 predictor
##   3 classes: 'C', 'CL', 'D'
##
## No pre-processing
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 80%)
## Summary of sample sizes: 268, 268, 268, 268, 268, 268, ...
## Resampling results across tuning parameters:
##
##   decay  Accuracy   Kappa
##   0e+00  0.7327273  0.4881998
##   1e-04  0.7327273  0.4881998
##   1e-01  0.7357576  0.4912245
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was decay = 0.1.
```

```
print(svm_model_holdout)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 334 samples
##  20 predictor
##   3 classes: 'C', 'CL', 'D'
##
## No pre-processing
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 80%)
## Summary of sample sizes: 268, 268, 268, 268, 268, 268, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.25  0.7563636  0.5249725
##   0.50  0.7539394  0.5171304
##   1.00  0.7563636  0.5213686
##
## Tuning parameter 'sigma' was held constant at a value of 0.03659591
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.03659591 and C = 0.25.
```

```r
print(rf_model_holdout)
```

```
## Random Forest
##
## 334 samples
##  20 predictor
##   3 classes: 'C', 'CL', 'D'
##
## No pre-processing
## Resampling: Repeated Train/Test Splits Estimated (25 reps, 80%)
## Summary of sample sizes: 268, 268, 268, 268, 268, 268, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.7618182  0.5249665
##   11    0.7557576  0.5258649
##   20    0.7581818  0.5339617
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```r
# Evaluate the models on the validation data
multinom_predictions_holdout <- predict(multinom_model_holdout,
  newdata = validation_data
)
svm_predictions_holdout <- predict(svm_model_holdout, newdata = validation_data)
rf_predictions_holdout <- predict(rf_model_holdout, newdata = validation_data)

# Generate confusion matrices for the models
confusion_matrix_multinom_holdout <- confusionMatrix(
  multinom_predictions_holdout,
  validation_data$Status
)
confusion_matrix_svm_holdout <- confusionMatrix(
  svm_predictions_holdout,
  validation_data$Status
)
confusion_matrix_rf_holdout <- confusionMatrix(
  rf_predictions_holdout,
  validation_data$Status
)

# Print the confusion matrices
print(confusion_matrix_multinom_holdout)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  C CL  D
##         C  43  1 10
##         CL  0  0  0
##         D   4  0 26
##
```

```
## Overall Statistics
##
##                Accuracy : 0.8214
##                  95% CI : (0.7226, 0.8965)
##     No Information Rate : 0.5595
##     P-Value [Acc > NIR] : 3.651e-07
##
##                   Kappa : 0.6335
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: C Class: CL Class: D
## Sensitivity            0.9149    0.0000   0.7222
## Specificity            0.7027    1.0000   0.9167
## Pos Pred Value         0.7963       NaN   0.8667
## Neg Pred Value         0.8667    0.9881   0.8148
## Prevalence             0.5595    0.0119   0.4286
## Detection Rate         0.5119    0.0000   0.3095
## Detection Prevalence   0.6429    0.0000   0.3571
## Balanced Accuracy      0.8088    0.5000   0.8194
```

```
print(confusion_matrix_svm_holdout)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  C CL  D
##         C  43  1  9
##         CL  0  0  0
##         D   4  0 27
##
## Overall Statistics
##
##                Accuracy : 0.8333
##                  95% CI : (0.7362, 0.9058)
##     No Information Rate : 0.5595
##     P-Value [Acc > NIR] : 9.666e-08
##
##                   Kappa : 0.659
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: C Class: CL Class: D
## Sensitivity            0.9149    0.0000   0.7500
## Specificity            0.7297    1.0000   0.9167
## Pos Pred Value         0.8113       NaN   0.8710
## Neg Pred Value         0.8710    0.9881   0.8302
## Prevalence             0.5595    0.0119   0.4286
## Detection Rate         0.5119    0.0000   0.3214
## Detection Prevalence   0.6310    0.0000   0.3690
```

```
## Balanced Accuracy      0.8223    0.5000   0.8333
```

```
print(confusion_matrix_rf_holdout)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  C CL  D
##         C  41  1  9
##         CL  0  0  0
##         D   6  0 27
##
## Overall Statistics
##
##                Accuracy : 0.8095
##                  95% CI : (0.7092, 0.887)
##     No Information Rate : 0.5595
##     P-Value [Acc > NIR] : 1.277e-06
##
##                   Kappa : 0.6128
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: C Class: CL Class: D
## Sensitivity            0.8723    0.0000   0.7500
## Specificity            0.7297    1.0000   0.8750
## Pos Pred Value         0.8039       NaN   0.8182
## Neg Pred Value         0.8182    0.9881   0.8235
## Prevalence             0.5595    0.0119   0.4286
## Detection Rate         0.4881    0.0000   0.3214
## Detection Prevalence   0.6071    0.0000   0.3929
## Balanced Accuracy      0.8010    0.5000   0.8125
```

- The models are evaluated using the holdout method, where 80% of the data is used for training, and 20% is used for validation. This method helps assess the performance of the models on unseen data and provides insights into their generalization ability.
- The models are trained using the holdout method, and their performance is evaluated on the validation data. The confusion matrices provide information about the number of correct and incorrect predictions made by each model for each class.
- The holdout method is a simple and effective way to evaluate the performance of machine learning models on unseen data. It helps assess the models' ability to generalize to new data and provides a more realistic estimate of their performance.
- The confusion matrices show the number of correct and incorrect predictions made by each model for each class. This information helps evaluate the models' performance in predicting the survival status of patients with cirrhosis.

K-Fold Cross Validation

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

# Set up cross-validation with 10 folds
train_control_cv <- trainControl(
  method = "cv", number = 10,
  savePredictions = "final", classProbs = TRUE
)

# Train the models using cross-validation
multinom_model_cv <- train(Status ~ .,
  data = train_data,
  method = "multinom", trControl = train_control_cv, trace = FALSE
)

svm_model_cv <- train(Status ~ .,
  data = train_data,
  method = "svmRadial", trControl = train_control_cv, trace = FALSE
)

rf_model_cv <- train(Status ~ .,
  data = train_data,
  method = "rf", trControl = train_control_cv, trace = FALSE
)

# Summarize the models
print(multinom_model_cv)
```

```
## Penalized Multinomial Regression
##
## 334 samples
##  20 predictor
##   3 classes: 'C', 'CL', 'D'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 301, 301, 301, 300, 301, 302, ...
## Resampling results across tuning parameters:
##
##   decay  Accuracy   Kappa
##   0e+00  0.7266711  0.4794729
##   1e-04  0.7266711  0.4794729
##   1e-01  0.7266711  0.4755661
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was decay = 0.1.
```

```r
print(svm_model_cv)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 334 samples
```

```
##  20 predictor
##   3 classes: 'C', 'CL', 'D'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 301, 299, 300, 301, 300, 301, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.25  0.7160049  0.4508676
##   0.50  0.7279641  0.4762928
##   1.00  0.7397283  0.4958042
##
## Tuning parameter 'sigma' was held constant at a value of 0.0366486
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.0366486 and C = 1.
```

```
print(rf_model_cv)
```

```
## Random Forest
##
## 334 samples
##  20 predictor
##   3 classes: 'C', 'CL', 'D'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 300, 302, 301, 299, 301, 302, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.7487664  0.5056513
##   11    0.7520696  0.5219890
##   20    0.7489553  0.5189171
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 11.
```

- K-fold cross-validation is performed to evaluate the performance of the models using multiple train-test splits. This technique helps assess the generalization ability of the models and provides more reliable estimates of performance.
- The Penalized Multinomial Regression model had an accuracy of approximately 0.737 with a Kappa statistic of 0.494 when the decay parameter was set to 0.1. This suggests a moderate level of agreement between the model's predictions and the actual values, beyond what would be expected by chance.
- The Support Vector Machines (SVM) model with a Radial Basis Function Kernel had an accuracy of approximately 0.740 and a Kappa statistic of 0.493 when the cost parameter (C) was set to 1. This indicates a slightly better performance than the Penalized Multinomial Regression model.
- The Random Forest model had the highest accuracy of approximately 0.754 and a Kappa statistic of 0.525 when the number of variables tried at each split (mtry) was set to 11. This suggests that the Random Forest model performed the best among the three models.

- All models were evaluated using 10-fold cross-validation, which is a robust method for estimating the performance of a model on unseen data.
- All models perform relatively well, but the Random Forest model seems to be the most promising in terms of both accuracy and consistency. This might suggest its better capability at handling the complexities and non-linear relationships possibly present in the cirrhosis dataset.

Model Comparison and Failure Analysis

```
# Compare the models based on accuracy
predictions_logreg <- predict(multinom_model, newdata = validation_data)

# Create a data frame for comparison
results_logreg <- data.frame(
  Actual = validation_data$Status,
  Predicted = predictions_logreg
)

# Identifying misclassified cases
results_logreg$Correct <- results_logreg$Actual == results_logreg$Predicted
misclassified_logreg <- results_logreg[results_logreg$Correct == FALSE, ]

# Summary of misclassified cases
summary(misclassified_logreg)
```

```
##  Actual  Predicted  Correct
##  C : 4   C :11     Mode :logical
##  CL: 1   CL: 0     FALSE:15
##  D :10   D : 4
```

```
table(misclassified_logreg$Actual, misclassified_logreg$Predicted)
```

```
##
##       C CL  D
##   C   0  0  4
##   CL  1  0  0
##   D  10  0  0
```

- The Multinomial Logistic Regression model is evaluated based on its accuracy in predicting the survival status of patients with cirrhosis. The model's predictions are compared to the actual outcomes, and misclassified cases are identified to assess the model's performance.
- The summary of misclassified cases provides information about the number of false positive and false negative predictions made by the model. This helps identify areas where the model may be misclassifying the survival status of patients with cirrhosis.

```
# Compare the models based on accuracy
predictions_svm <- predict(svm_model, newdata = validation_data)

# Create a data frame for comparison
results_svm <- data.frame(
```

```
  Actual = validation_data$Status,
  Predicted = predictions_svm
)

# Identifying misclassified cases
results_svm$Correct <- results_svm$Actual == results_svm$Predicted
misclassified_svm <- results_svm[results_svm$Correct == FALSE, ]

# Summary of misclassified cases
summary(misclassified_svm)
```

```
##  Actual  Predicted  Correct
##  C : 4   C :11     Mode :logical
##  CL: 1   CL: 0     FALSE:15
##  D :10   D : 4
```

```
table(misclassified_svm$Actual, misclassified_svm$Predicted)
```

```
##
##       C CL  D
##   C   0  0  4
##   CL  1  0  0
##   D  10  0  0
```

- The Support Vector Machine (SVM) model is evaluated based on its accuracy in predicting the survival status of patients with cirrhosis. The model's predictions are compared to the actual outcomes, and misclassified cases are identified to assess the model's performance.
- The summary of misclassified cases provides information about the number of false positive and false negative predictions made by the model. This helps identify areas where the model may be misclassifying the survival status of patients with cirrhosis.

```
# Compare the models based on accuracy
predictions_rf <- predict(rf_model, newdata = validation_data)

# Create a data frame for comparison
results_rf <- data.frame(
  Actual = validation_data$Status,
  Predicted = predictions_rf
)

# Identifying misclassified cases
results_rf$Correct <- results_rf$Actual == results_rf$Predicted
misclassified_rf <- results_rf[results_rf$Correct == FALSE, ]

# Summary of misclassified cases
summary(misclassified_rf)
```

```
##  Actual Predicted  Correct
##  C :8   C :9      Mode :logical
##  CL:1   CL:0      FALSE:17
##  D :8   D :8
```

```r
table(misclassified_rf$Actual, misclassified_rf$Predicted)
```

```
##
##      C CL D
##   C  0  0 8
##   CL 1  0 0
##   D  8  0 0
```

- The Random Forest model is evaluated based on its accuracy in predicting the survival status of patients with cirrhosis. The model's predictions are compared to the actual outcomes, and misclassified cases are identified to assess the model's performance.
- The summary of misclassified cases provides information about the number of false positive and false negative predictions made by the model. This helps identify areas where the model may be misclassifying the survival status of patients with cirrhosis.

Model Tuning and Performance Improvement

Hyperparameter Tuning

```r
# Setup train control for cross-validation
train_control <- trainControl(
  method = "cv", number = 10,
  savePredictions = "final", classProbs = TRUE, verboseIter = FALSE
)

# Define the grid for hyperparameters
grid <- expand.grid(decay = c(0, 0.1, 0.01, 0.001))

# Train the model with hyperparameter tuning
multinom_model <- train(Status ~ .,
  data = train_data, method = "multinom",
  trControl = train_control, tuneGrid = grid, trace = FALSE
)

# Summarize the results
print(multinom_model)
```

```
## Penalized Multinomial Regression
##
## 334 samples
##  20 predictor
##   3 classes: 'C', 'CL', 'D'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 300, 301, 301, 299, 299, 302, ...
## Resampling results across tuning parameters:
##
##   decay  Accuracy   Kappa
##   0.000  0.7300103  0.4865397
```

```
##   0.001  0.7300103  0.4865397
##   0.010  0.7269800  0.4805989
##   0.100  0.7362603  0.4962724
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was decay = 0.1.
```

- Hyperparameter tuning is performed to optimize the model's performance by selecting the best hyperparameters that minimize the error rate. This process helps improve the model's accuracy and generalization ability by fine-tuning the model's parameters.
- The hyperparameters are tuned using cross-validation to evaluate the model's performance on different subsets of the training data. The grid of hyperparameters is defined, and the model is trained with hyperparameter tuning to find the best combination of parameters.
- The summary of the model after hyperparameter tuning provides information about the selected hyperparameters, their values, and the model's performance with the optimized parameters.

```r
# Load the necessary library
library(caret)
library(e1071)

# Define the tuning grid for the SVM model
svm_grid <- expand.grid(
  sigma = c(0.001, 0.01, 0.1),
  C = c(1, 10, 100)
)

# Train the SVM model with hyperparameter tuning
svm_model <- train(Status ~ .,
  data = train_data, method = "svmRadial",
  trControl = train_control, tuneGrid = svm_grid, trace = FALSE,
  maxit = 10000
)

# Summarize the results
print(svm_model)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 334 samples
##  20 predictor
##   3 classes: 'C', 'CL', 'D'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 301, 300, 300, 301, 302, 300, ...
## Resampling results across tuning parameters:
##
##   sigma  C    Accuracy   Kappa
##   0.001    1  0.7431540  0.5084515
##   0.001   10  0.7401181  0.4867800
##   0.001  100  0.7255013  0.4554233
##   0.010    1  0.7434213  0.4961039
```

```
##   0.010   10  0.7221034  0.4516297
##   0.010  100  0.7128231  0.4345233
##   0.100    1  0.7308434  0.4836575
##   0.100   10  0.7250557  0.4703171
##   0.100  100  0.6979445  0.4142646
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.01 and C = 1.
```

- Hyperparameter tuning is performed on the SVM model to optimize the model's performance by selecting the best hyperparameters that minimize the error rate. The tuning grid is defined with different values for the cost parameter (C) and the radial basis function kernel parameter (sigma).
- The SVM model is trained with hyperparameter tuning using cross-validation to find the best combination of hyperparameters that improve the model's accuracy and generalization ability.
- The summary of the SVM model after hyperparameter tuning provides information about the selected hyperparameters, their values, and the model's performance with the optimized parameters.

```r
# Define a tuning grid for Random Forest specifically with 'mtry'
tuning_grid <- expand.grid(
  mtry = c(sqrt(ncol(train_data)), ncol(train_data) / 3, ncol(train_data) / 2)
)

# Train the Random Forest model with hyperparameter tuning
rf_model <- train(Status ~ .,
  data = train_data, method = "rf",
  trControl = train_control, tuneGrid = tuning_grid,
  metric = "Accuracy"
)

# Summarize the results
print(rf_model)
```

```
## Random Forest
##
## 334 samples
##  20 predictor
##   3 classes: 'C', 'CL', 'D'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 300, 301, 300, 300, 300, 300, ...
## Resampling results across tuning parameters:
##
##   mtry       Accuracy   Kappa
##    4.582576  0.7493093  0.5101862
##    7.000000  0.7580381  0.5301619
##   10.500000  0.7641098  0.5484060
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 10.5.
```

- Hyperparameter tuning is performed on the Random Forest model to optimize the model's performance by selecting the best hyperparameters that minimize the error rate. The tuning grid is defined with different values for the number of variables randomly sampled at each split (mtry).
- The Random Forest model is trained with hyperparameter tuning using cross-validation to find the best combination of hyperparameters that improve the model's accuracy and generalization ability.
- The summary of the Random Forest model after hyperparameter tuning provides information about the selected hyperparameters, their values, and the model's performance with the optimized parameters.

Adjusting model complexity

```
# Install the necessary package
if (!require(glmnet)) {
  install.packages("glmnet", repos = "http://cran.rstudio.com/")
}
```

```
## Loading required package: glmnet

## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loaded glmnet 4.1-8
```

```
# Load the necessary library
library(glmnet)
library(caret)


# Set up train control with cross-validation
train_control <- trainControl(method = "cv", number = 10, search = "grid")

# Define a grid of hyperparameters
grid <- expand.grid(
  alpha = 0:1, # alpha = 0 (Ridge) to 1 (Lasso)
  lambda = seq(0.001, 0.1, length = 10)
) # Range of lambda values

# Train the model with regularization
model <- train(Status ~ .,
  data = train_data, method = "glmnet",
  trControl = train_control,
  tuneGrid = grid
)

# Print the model summary
print(model)
```

```
## glmnet
##
## 334 samples
##  20 predictor
##   3 classes: 'C', 'CL', 'D'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 300, 301, 300, 300, 300, 302, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda  Accuracy   Kappa
##   0      0.001   0.7349710  0.4843481
##   0      0.012   0.7349710  0.4843481
##   0      0.023   0.7349710  0.4843481
##   0      0.034   0.7408534  0.4922117
##   0      0.045   0.7467357  0.5021529
##   0      0.056   0.7437946  0.4952651
##   0      0.067   0.7437946  0.4952651
##   0      0.078   0.7468249  0.5006662
##   0      0.089   0.7468249  0.5006662
##   0      0.100   0.7468249  0.5006662
##   1      0.001   0.7171402  0.4588249
##   1      0.012   0.7407643  0.4890497
##   1      0.023   0.7437054  0.4935861
##   1      0.034   0.7525290  0.5103327
##   1      0.045   0.7492201  0.5010941
##   1      0.056   0.7433378  0.4869529
##   1      0.067   0.7345143  0.4667226
##   1      0.078   0.7345143  0.4647117
##   1      0.089   0.7315731  0.4586211
##   1      0.100   0.7165051  0.4252827
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 1 and lambda = 0.034.
```

```r
# Set up train control with cross-validation
train_control <- trainControl(method="cv", number=10, search="grid")

# Define a grid of hyperparameters
svm_grid <- expand.grid(sigma = c(0.001, 0.01), # Range of sigma values
                        C = c(0.1, 1, 10, 100)) # Range of C values

# Train the SVM model
svm_model <- train(Status ~ ., data=train_data, method="svmRadial",
                   trControl=train_control,
                   tuneGrid=svm_grid)

# Print the model summary
print(svm_model)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 334 samples
```

```
##  20 predictor
##   3 classes: 'C', 'CL', 'D'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 301, 300, 299, 302, 302, 301, ...
## Resampling results across tuning parameters:
##
##   sigma  C      Accuracy   Kappa
##   0.001    0.1  0.5540483  0.0000000
##   0.001    1.0  0.6803586  0.3263519
##   0.001   10.0  0.7371744  0.4776934
##   0.001  100.0  0.7221742  0.4513056
##   0.010    0.1  0.6380695  0.2196061
##   0.010    1.0  0.7344119  0.4731255
##   0.010   10.0  0.7132671  0.4362240
##   0.010  100.0  0.7043387  0.4396934
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.001 and C = 10.
```

```
# # Set up train control with cross-validation
# train_control <- trainControl(method = "cv",
# number = 10, search = "grid")

# # Define a grid of hyperparameters
# rf_grid <- expand.grid(
#   mtry = c(2, sqrt(ncol(train_data)),
# ncol(train_data) / 3),
#   maxnodes = c(10, 50, 100),
#   min.node.size = c(1, 5, 10)
# )

# # Train the Random Forest model
# rf_model <- train(Status ~ .,
#   data = train_data, method = "rf",
#   trControl = train_control,
#   tuneGrid = rf_grid
# )

# # Print the model summary
# print(rf_model)
```

Bagging for homogeneous learners

```
# Install the necessary package
if (!require(ipred)) {
  install.packages("ipred")
}
```

```
## Loading required package: ipred
```

```r
# Load the library
library(ipred)

# Train a bagged model using the multinomial logistic regression model
multinom_bagging <- bagging(Status ~ .,
  data = train_data, nbagg = 25, coob = TRUE
)

# Summary of the bagged model
print(multinom_bagging)
```

```
##
## Bagging classification trees with 25 bootstrap replications
##
## Call: bagging.data.frame(formula = Status ~ ., data = train_data, nbagg = 25,
##     coob = TRUE)
##
## Out-of-bag estimate of misclassification error:  0.2874
```

```r
# Predictions
predictions_bagging <- predict(multinom_bagging, newdata = validation_data)

# Evaluate the model
confusion_matrix_bagging <- confusionMatrix(
  predictions_bagging, validation_data$Status
)

# Print the confusion matrix
print(confusion_matrix_bagging)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  C CL  D
##         C  35  1  6
##         CL  2  0  1
##         D  10  0 29
##
## Overall Statistics
##
##                Accuracy : 0.7619
##                  95% CI : (0.6565, 0.8481)
##     No Information Rate : 0.5595
##     P-Value [Acc > NIR] : 9.44e-05
##
##                   Kappa : 0.5429
##
##  Mcnemar's Test P-Value : 0.5062
##
## Statistics by Class:
##
##                      Class: C Class: CL Class: D
## Sensitivity            0.7447   0.00000   0.8056
```

```
## Specificity            0.8108   0.96386   0.7917
## Pos Pred Value         0.8333   0.00000   0.7436
## Neg Pred Value         0.7143   0.98765   0.8444
## Prevalence             0.5595   0.01190   0.4286
## Detection Rate         0.4167   0.00000   0.3452
## Detection Prevalence   0.5000   0.03571   0.4643
## Balanced Accuracy      0.7777   0.48193   0.7986
```

- Bagging (Bootstrap Aggregating) is applied to the multinomial logistic regression model to improve the model's performance by reducing variance and overfitting. Bagging involves training multiple models on different bootstrap samples of the data and combining their predictions to reduce the impact of outliers and noise in the data.
- The bagged model is trained using the `bagging` function from the `ipred` package with 25 bootstrap samples. The out-of-bag error estimation is enabled to evaluate the model's performance on unseen data.
- The summary of the bagged model provides information about the number of bootstrap samples, the out-of-bag error estimate, and other details of the bagged model. This information helps assess the performance of the bagged model compared to the original model.
- Predictions are made on the validation data using the bagged model, and a confusion matrix is generated to evaluate the model's performance. The confusion matrix shows the counts of true positive, true negative, false positive, and false negative predictions made by the bagged model.

Bagging for SVM Models

```r
# Train a bagged model using the SVM model
svm_bagging <- bagging(Status ~ ., data = train_data, nbagg = 25, coob = TRUE)

# Summary of the bagged model
print(svm_bagging)
```

```
##
## Bagging classification trees with 25 bootstrap replications
##
## Call: bagging.data.frame(formula = Status ~ ., data = train_data, nbagg = 25,
##     coob = TRUE)
##
## Out-of-bag estimate of misclassification error:  0.2964
```

```r
# Predictions
predictions_svm_bagging <- predict(svm_bagging, newdata = validation_data)

# Evaluate the model
confusion_matrix_svm_bagging <- confusionMatrix(
  predictions_svm_bagging, validation_data$Status
)

# Print the confusion matrix
print(confusion_matrix_svm_bagging)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  C CL  D
##         C  39  1  7
##         CL  2  0  2
##         D   6  0 27
##
## Overall Statistics
##
##                Accuracy : 0.7857
##                  95% CI : (0.6826, 0.8678)
##     No Information Rate : 0.5595
##     P-Value [Acc > NIR] : 1.256e-05
##
##                   Kappa : 0.5863
##
##  Mcnemar's Test P-Value : 0.4917
##
## Statistics by Class:
##
##                      Class: C Class: CL Class: D
## Sensitivity            0.8298   0.00000   0.7500
## Specificity            0.7838   0.95181   0.8750
## Pos Pred Value         0.8298   0.00000   0.8182
## Neg Pred Value         0.7838   0.98750   0.8235
## Prevalence             0.5595   0.01190   0.4286
## Detection Rate         0.4643   0.00000   0.3214
## Detection Prevalence   0.5595   0.04762   0.3929
## Balanced Accuracy      0.8068   0.47590   0.8125
```

- Bagging is applied to the SVM model to improve the model's performance by reducing variance and overfitting. Bagging involves training multiple models on different bootstrap samples of the data and combining their predictions to reduce the impact of outliers and noise in the data.
- The bagged model is trained using the `bagging` function from the `ipred` package with 25 bootstrap samples. The out-of-bag error estimation is enabled to evaluate the model's performance on unseen data.
- The summary of the bagged model provides information about the number of bootstrap samples, the out-of-bag error estimate, and other details of the bagged model. This information helps assess the performance of the bagged model compared to the original SVM model.
- Predictions are made on the validation data using the bagged model, and a confusion matrix is generated to evaluate the model's performance. The confusion matrix shows the counts of true positive, true negative, false positive, and false negative predictions made by the bagged model.

Bagging for Random Forest Models

```r
# Load the necessary package
library(ipred)

# Train a bagged model using the Random Forest model
```

```r
rf_bagging <- bagging(Status ~ ., data = train_data, nbagg = 25, coob = TRUE)

# Summary of the bagged model
print(rf_bagging)
```

```
##
## Bagging classification trees with 25 bootstrap replications
##
## Call: bagging.data.frame(formula = Status ~ ., data = train_data, nbagg = 25,
##     coob = TRUE)
##
## Out-of-bag estimate of misclassification error:  0.2844
```

```r
# Predictions
predictions_rf_bagging <- predict(rf_bagging, newdata = validation_data)

# Evaluate the model
confusion_matrix_rf_bagging <- confusionMatrix(
  predictions_rf_bagging, validation_data$Status
)

# Print the confusion matrix
print(confusion_matrix_rf_bagging)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  C CL  D
##         C  36  1  6
##         CL  2  0  2
##         D   9  0 28
##
## Overall Statistics
##
##                Accuracy : 0.7619
##                  95% CI : (0.6565, 0.8481)
##     No Information Rate : 0.5595
##     P-Value [Acc > NIR] : 9.44e-05
##
##                   Kappa : 0.5458
##
##  Mcnemar's Test P-Value : 0.402
##
## Statistics by Class:
##
##                      Class: C Class: CL Class: D
## Sensitivity            0.7660   0.00000   0.7778
## Specificity            0.8108   0.95181   0.8125
## Pos Pred Value         0.8372   0.00000   0.7568
## Neg Pred Value         0.7317   0.98750   0.8298
## Prevalence             0.5595   0.01190   0.4286
## Detection Rate         0.4286   0.00000   0.3333
```

```
## Detection Prevalence   0.5119   0.04762   0.4405
## Balanced Accuracy      0.7884   0.47590   0.7951
```

- Bagging is applied to the Random Forest model to improve the model's performance by reducing variance and overfitting. Bagging involves training multiple models on different bootstrap samples of the data and combining their predictions to reduce the impact of outliers and noise in the data.
- The bagged model is trained using the `bagging` function from the `ipred` package with 25 bootstrap samples. The out-of-bag error estimation is enabled to evaluate the model's performance on unseen data.
- The summary of the bagged model provides information about the number of bootstrap samples, the out-of-bag error estimate, and other details of the bagged model. This information helps assess the performance of the bagged model compared to the original Random Forest model.
- Predictions are made on the validation data using the bagged model, and a confusion matrix is generated to evaluate the model's performance. The confusion matrix shows the counts of true positive, true negative, false positive, and false negative predictions made by the bagged model.