A Predictive Model to Predict the development of Hepatitis C stages: Development and Performance Evaluation

DA5030

Akanksha Pandey

Fall 2023

Contents

Intorduction

The objective of this study is to forecast the progression stages of Hepatitis C. The primary classification criterion is Category (2), which differentiates between blood donors and Hepatitis C patients, including stages of the disease (Hepatitis C, Fibrosis, Cirrhosis). In this analysis, the stages of Cirrhosis, Fibrosis, and Hepatitis C have been amalgamated into a single category named 'Hepatitis', while the contrasting group is labeled as 'Donor'.

Algorithms used: Random Forest models, Penalized logistic regression, and Boosted Trees model

Feature Engineering and Transformation: Impute using median, turn nominal variables into dummies, Normalize all predictors, Balance target classes using SMOTE algorithm.

Libraries

```
# Load necessary libraries
library(utils)
library(psych)
library(caret)
library(tidyverse)
library(skimr)
library(stringr)
library(themis)
library(vip)
library(probably)
library("ggplot2")
library("GGally")
library(corrplot)
library(randomForest)
library(pROC)
library(tidymodels)
library(ranger)
library(xgboost)
```

1. Data Acquisition

The dataset that I am using for my analysis includes laboratory test results of blood donors and Hepatitis C patients, along with demographic details like age. This dataset was sourced from the UCI Machine Learning Repository, specifically from their 'HCV data' section.

It consists of 615 observations and 14 variables, and is primarily aimed at health analysis. It uniquely distinguishes between blood donors and Hepatitis C patients, a critical aspect for medical research and analysis. The dataset includes basic demographic information such as age and sex. In addition, it features a comprehensive set of health parameters: ALB, ALP, ALT, AST, BIL , CHE, CHOL, CREA , GGT, and PROT .

```
##   X      Category Age Sex  ALB  ALP  ALT  AST  BIL   CHE CHOL CREA  GGT PROT
## 1 1 0=Blood Donor  32   m 38.5 52.5  7.7 22.1  7.5  6.93 3.23  106 12.1 69.0
## 2 2 0=Blood Donor  32   m 38.5 70.3 18.0 24.7  3.9 11.17 4.80   74 15.6 76.5
## 3 3 0=Blood Donor  32   m 46.9 74.7 36.2 52.6  6.1  8.84 5.20   86 33.2 79.3
## 4 4 0=Blood Donor  32   m 43.2 52.0 30.6 22.6 18.9  7.33 4.74   80 33.8 75.7
## 5 5 0=Blood Donor  32   m 39.2 74.1 32.6 24.8  9.6  9.15 4.32   76 29.9 68.7
## 6 6 0=Blood Donor  32   m 41.6 43.3 18.5 19.7 12.3  9.92 6.05  111 91.0 74.0
```

```
## 'data.frame':    615 obs. of  14 variables:
##  $ X       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Category: chr  "0=Blood Donor" "0=Blood Donor" "0=Blood Donor" "0=Blood Donor" ...
##  $ Age     : int  32 32 32 32 32 32 32 32 32 32 ...
##  $ Sex     : chr  "m" "m" "m" "m" ...
##  $ ALB     : num  38.5 38.5 46.9 43.2 39.2 41.6 46.3 42.2 50.9 42.4 ...
##  $ ALP     : num  52.5 70.3 74.7 52 74.1 43.3 41.3 41.9 65.5 86.3 ...
##  $ ALT     : num  7.7 18 36.2 30.6 32.6 18.5 17.5 35.8 23.2 20.3 ...
##  $ AST     : num  22.1 24.7 52.6 22.6 24.8 19.7 17.8 31.1 21.2 20 ...
##  $ BIL     : num  7.5 3.9 6.1 18.9 9.6 12.3 8.5 16.1 6.9 35.2 ...
##  $ CHE     : num  6.93 11.17 8.84 7.33 9.15 ...
##  $ CHOL    : num  3.23 4.8 5.2 4.74 4.32 6.05 4.79 4.6 4.1 4.45 ...
##  $ CREA    : num  106 74 86 80 76 111 70 109 83 81 ...
##  $ GGT     : num  12.1 15.6 33.2 33.8 29.9 91 16.9 21.5 13.7 15.9 ...
##  $ PROT    : num  69 76.5 79.3 75.7 68.7 74 74.5 67.1 71.3 69.9 ...
```

```
## [1] 615  14
```

Encoding catogeriocal variable

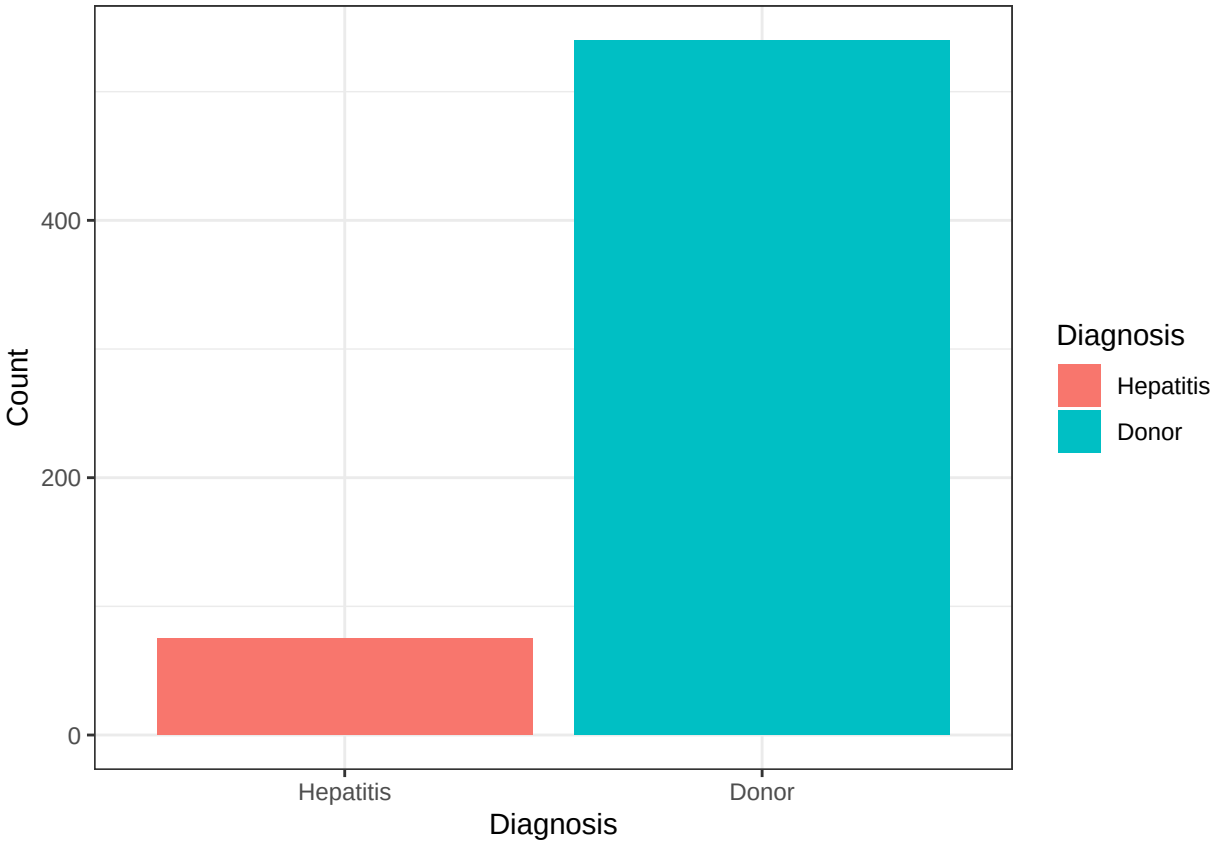
My target variable (diagnosis) is catogeriocal (values: '0=Blood Donor', '0s=suspect Blood Donor', '1=Hepatitis', '2=Fibrosis', '3=Cirrhosis') for this analysis the stages of Cirrhosis, Fibrosis, and Hepatitis C have been amalgamated into a single category named 'Hepatitis', while the contrasting group is labeled as 'Donor'. I will also remove column X to avoid overfitting.

```
##   Diagnosis Age Sex  ALB  ALP  ALT  AST  BIL   CHE CHOL CREA  GGT PROT
## 1     Donor  32   m 38.5 52.5  7.7 22.1  7.5  6.93 3.23  106 12.1 69.0
## 2     Donor  32   m 38.5 70.3 18.0 24.7  3.9 11.17 4.80   74 15.6 76.5
## 3     Donor  32   m 46.9 74.7 36.2 52.6  6.1  8.84 5.20   86 33.2 79.3
## 4     Donor  32   m 43.2 52.0 30.6 22.6 18.9  7.33 4.74   80 33.8 75.7
## 5     Donor  32   m 39.2 74.1 32.6 24.8  9.6  9.15 4.32   76 29.9 68.7
## 6     Donor  32   m 41.6 43.3 18.5 19.7 12.3  9.92 6.05  111 91.0 74.0
```

2. Data Exploration

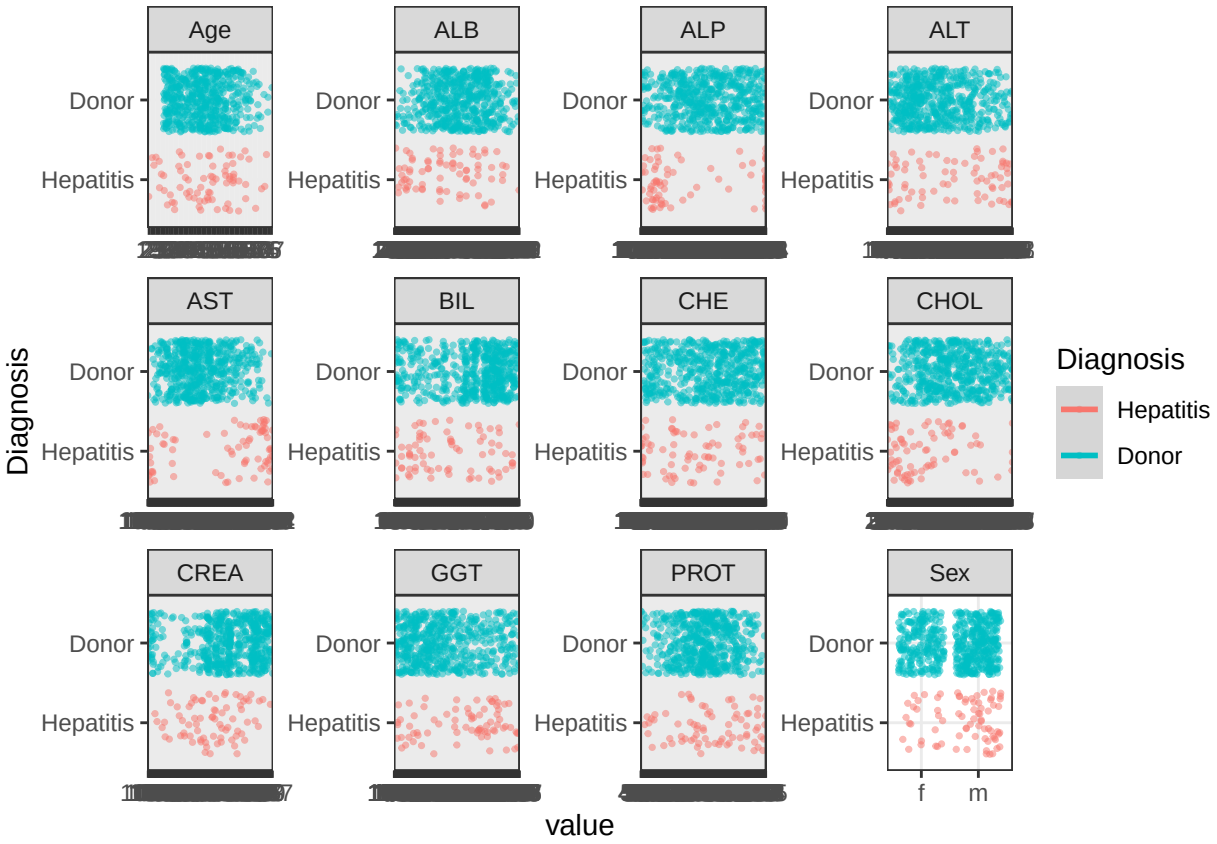
Bar plot Diagnosis Vs Count

This bar graph shows relation between diagnosis and the number of people for each case. The x-axis represents Diagnosis and the y-axis represents count. Form the graph we can see that the count for Donor is way more than Hepatitis.



Relationship between Diagnosis and other Variables.

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Graph for 'Age'

The age distribution appears to be similar in both 'Donor' and 'Hepatitis' groups, with a wide range of ages represented. There might be a slight concentration of older individuals in the 'Hepatitis' group.

Graph for 'ALB'

Albumin levels show a spread across a range of values. There may be some differences in the distribution of values between the two groups, but it's not distinctly separated.

Graph for 'ALP'

The ALP levels are spread out for both categories, with some potential outliers, especially in the 'Hepatitis' group.

Graph for 'ALT'

ALT levels vary widely in both groups. It's noticeable that the 'Hepatitis' category may have higher levels, as indicated by the spread of points.

Graph for 'AST'

AST levels show a wide range in both categories, with some high values particularly in the 'Hepatitis' group.

Graph for 'BIL'

BIL levels show a significant spread in the 'Hepatitis' group compared to 'Donor', with several higher values indicating potential liver function issues.

Graph for 'CHE'

The spread of CHE levels is quite wide in both groups. There doesn't appear to be a clear distinction between the two categories.

Graph for 'CHOL'

CHOL levels are varied in both groups. It's hard to discern any clear pattern differentiating the groups based on this plot.

Graph for 'CREA'

Creatinine levels appear to be relatively similar across both groups, with no significant differences observable from the plot.

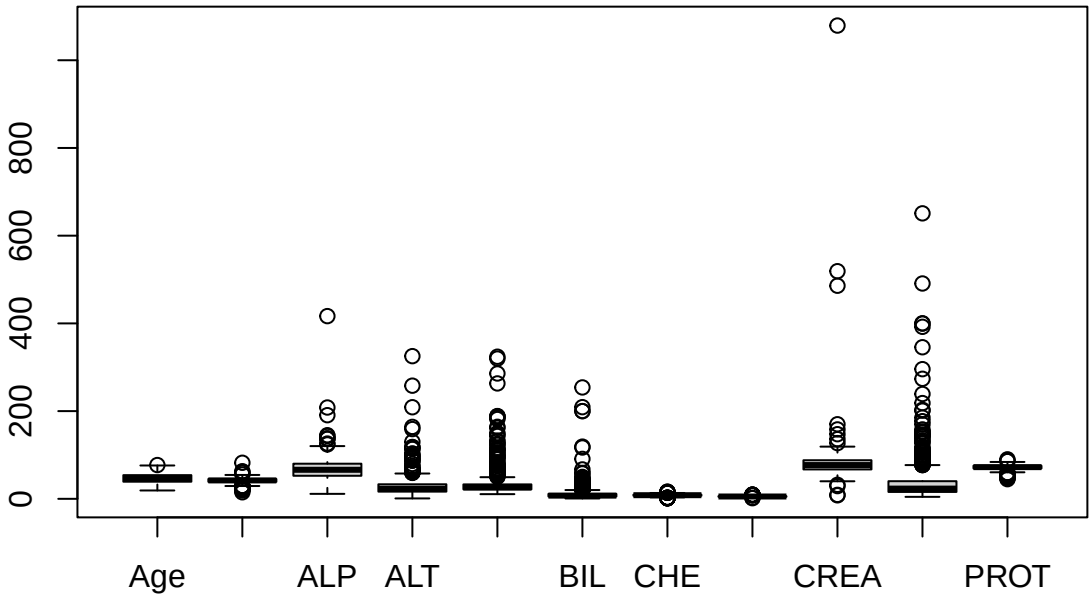
Graph for 'GGT'

GGT levels vary, with some higher values in the 'Hepatitis' group. This could indicate liver disease, as GGT is often elevated in liver disorders.

Graph for 'PROT'

Protein levels are fairly evenly distributed across both groups, with no striking differences visible.

Detection of outliers for continuous features

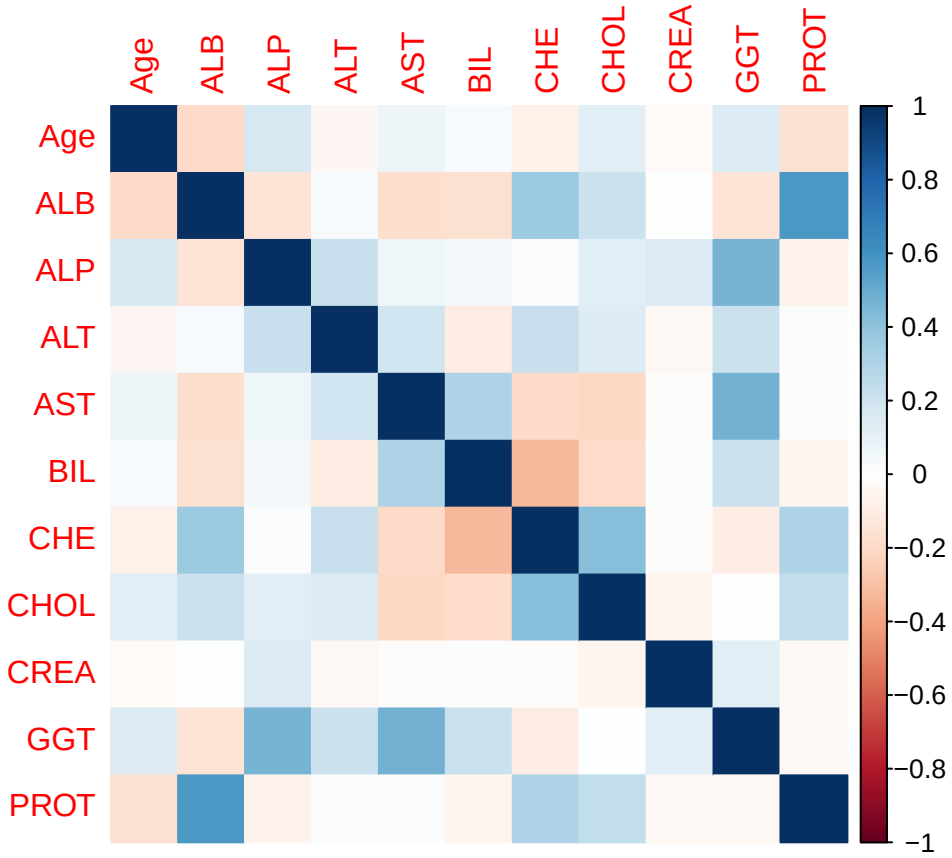



```
##  Age  ALB  ALP  ALT  AST  BIL  CHE CHOL CREA  GGT PROT
##    1   27   10   36   64   47   24   12   12   65   20
```

Features	Outliers
Age	1
ALB	27
ALP	10
ALT	36
AST	64
BIL	47
CHE	24
CHOL	12
CREA	12
GGT	65
PROT	20

Correlation analysis

```
##               Age          ALB          ALP          ALT          AST          BIL
## Age    1.00000000 -0.191093637  0.17771977 -0.04057647  0.07273886  0.03965486
## ALB   -0.19109364  1.000000000 -0.14611991  0.03949714 -0.17760895 -0.16959750
## ALP    0.17771977 -0.146119911  1.00000000  0.22160301  0.06702428  0.05837241
## ALT   -0.04057647  0.039497139  0.22160301  1.00000000  0.19865775 -0.10679662
## AST    0.07273886 -0.177608947  0.06702428  0.19865775  1.00000000  0.30957974
## BIL    0.03965486 -0.169597498  0.05837241 -0.10679662  0.30957974  1.00000000
## CHE   -0.07586328  0.360919403  0.02948169  0.22434447 -0.19727042 -0.32071323
## CHOL   0.12474161  0.210419878  0.12590008  0.14999727 -0.20121300 -0.18156956
## CREA  -0.02514225  0.001433247  0.15390895 -0.03610554 -0.01794810  0.01990962
## GGT    0.14337927 -0.147598318  0.46130000  0.21970686  0.47777362  0.21056656
## PROT  -0.15975998  0.570725680 -0.06308514  0.01678633  0.01740394 -0.05257491
##               CHE          CHOL         CREA          GGT         PROT
## Age   -0.07586328  0.124741615 -0.025142253  0.143379268 -0.15975998
## ALB    0.36091940  0.210419878  0.001433247 -0.147598318  0.57072568
## ALP    0.02948169  0.125900079  0.153908950  0.461299996 -0.06308514
## ALT    0.22434447  0.149997271 -0.036105541  0.219706857  0.01678633
## AST   -0.19727042 -0.201213004 -0.017948098  0.477773617  0.01740394
## BIL   -0.32071323 -0.181569556  0.019909617  0.210566559 -0.05257491
## CHE    1.00000000  0.428018276 -0.012119999 -0.095716131  0.30628754
## CHOL   0.42801828  1.000000000 -0.051464078  0.008822692  0.24504950
## CREA  -0.01212000 -0.051464078  1.000000000  0.125353469 -0.03011070
## GGT   -0.09571613  0.008822692  0.125353469  1.000000000 -0.03712701
## PROT   0.30628754  0.245049503 -0.030110695 -0.037127008  1.00000000
```



Correlations can provide insights into how different biochemical markers are related to each other, which can be useful in medical diagnosis and understanding the physiological state of an individual.

Based on the correlation matrix, correlations between various pairs of variables:

Age:

Negatively correlated with ALB (-0.19) and CHE (-0.08), suggesting that these variables tend to decrease slightly as age increases.

Positively correlated with ALP (0.18) and GGT (0.14), indicating these variables tend to increase slightly with age. The correlations with other variables are weak, close to 0.

ALB:

Strongly positively correlated with PROT (0.57), suggesting that higher ALB levels are associated with higher PROT levels.

Negatively correlated with AST (-0.18), BIL (-0.17), and GGT (-0.15), indicating that higher ALB levels are associated with lower levels of these variables. Other correlations are weak.

ALP:

Strongly positively correlated with GGT (0.46), suggesting a relationship between ALP and liver enzyme levels. Other correlations are relatively weak.

ALT:

Positively correlated with AST (0.20) and GGT (0.22), showing a relationship with these liver enzymes. Other correlations are weak.

AST:

Strongly positively correlated with GGT (0.48), indicating a close relationship between these liver enzymes. Negatively correlated with CHE (-0.20) and CHOL (-0.20). Other correlations are weak.

BIL:

Positively correlated with AST (0.31) and GGT (0.21), showing a relationship with liver function. Negatively correlated with ALB (-0.17) and CHE (-0.32). Other correlations are weak.

CHE:

Positively correlated with ALB (0.36), CHOL (0.43), and PROT (0.31), suggesting a relationship with these variables. Negatively correlated with AST (-0.20), BIL (-0.32), and GGT (-0.10). Other correlations are weak.

CHOL:

Positively correlated with CHE (0.43) and PROT (0.25). Other correlations are weak.

CREA:

Weak correlations with most variables, indicating no strong relationship with liver function tests or protein levels. Slightly positive correlation with ALP (0.15) and GGT (0.13).

GGT:

Strong correlations with ALP (0.46) and AST (0.48), indicating a close relationship with these liver enzymes. Other correlations are relatively weak.

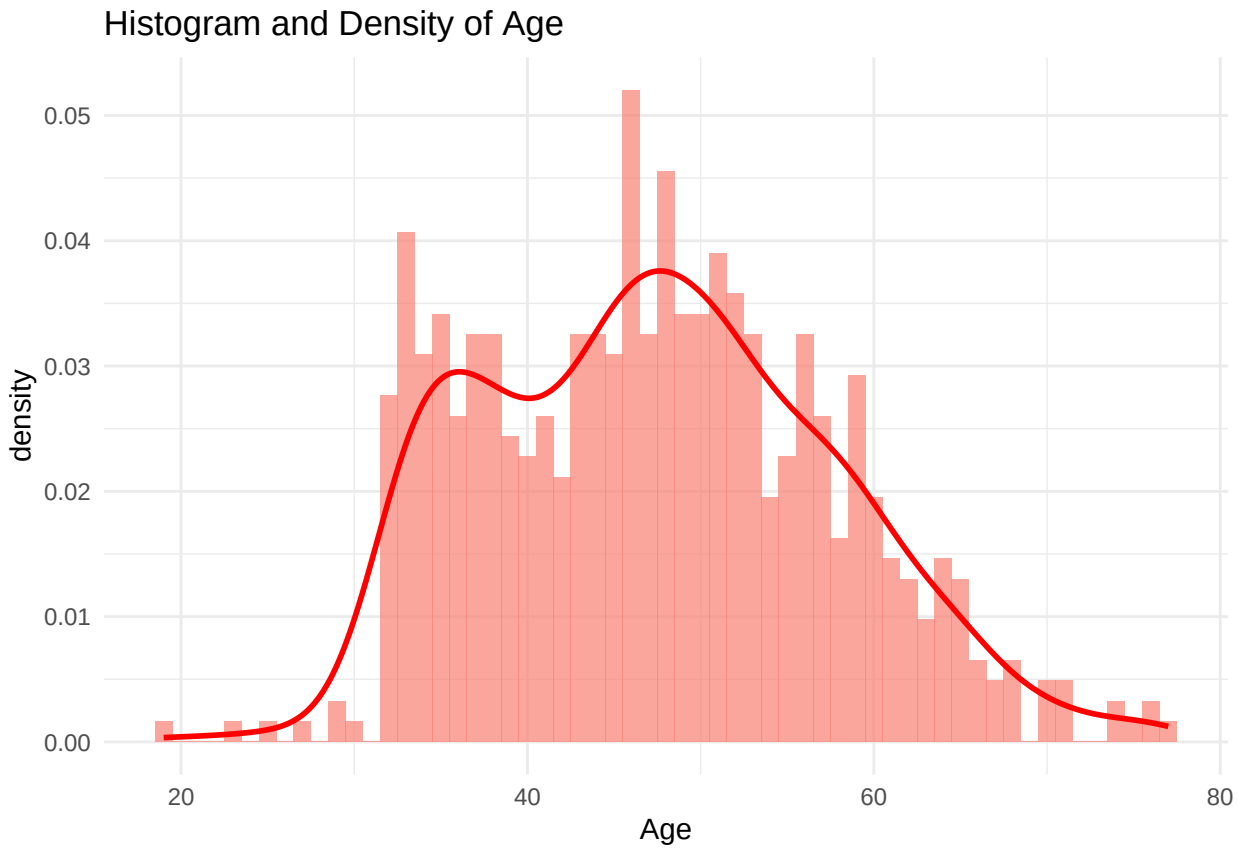
PROT:

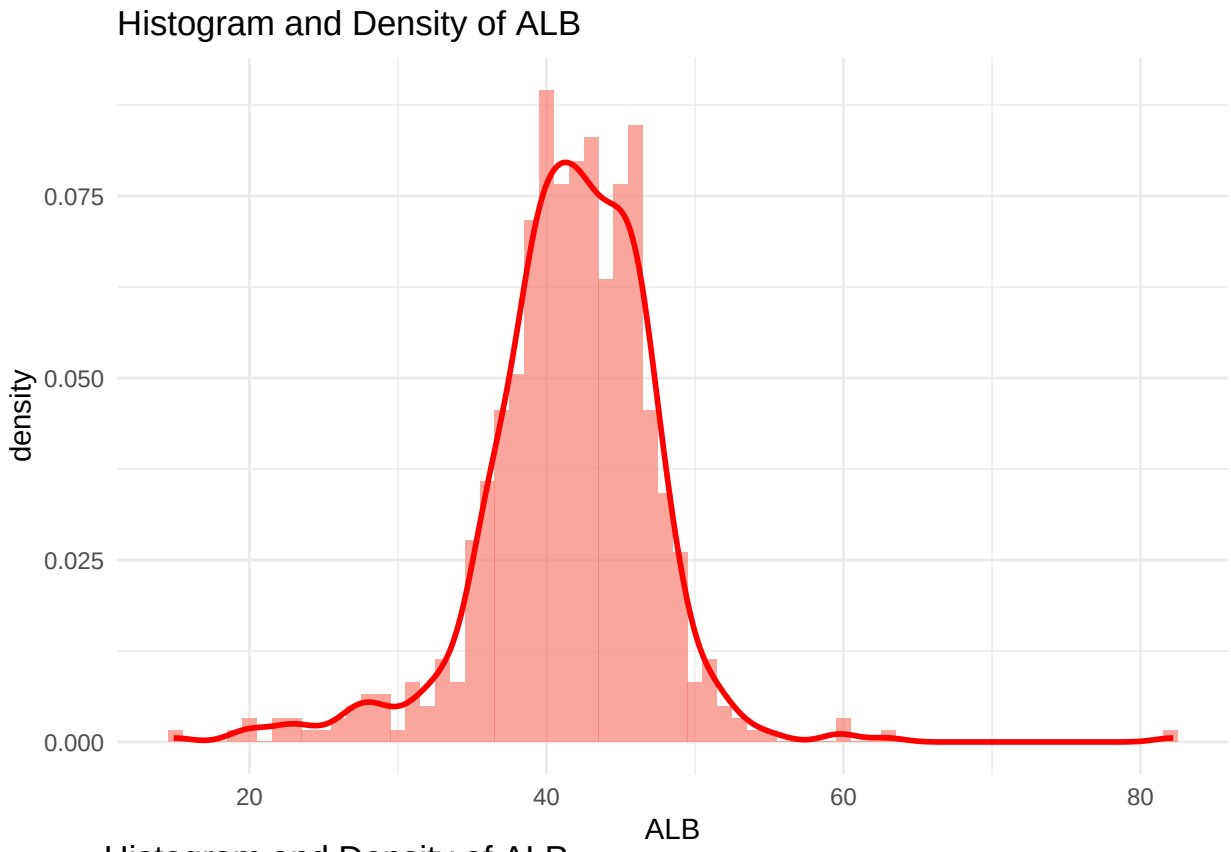
Strongly positively correlated with ALB (0.57), showing a close relationship. Other correlations are weak.

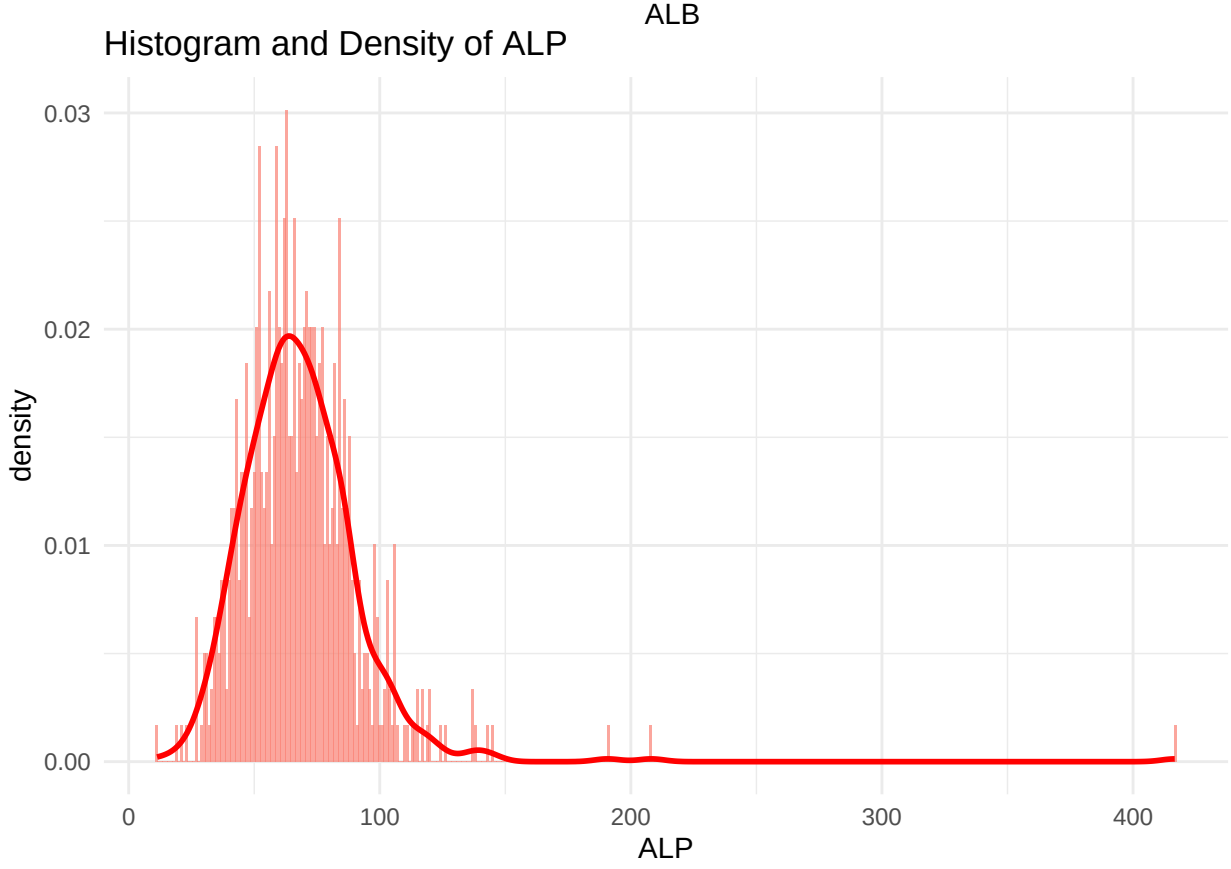
In summary, the strongest correlations are observed between ALP, ALT, AST, GGT and between ALB and PROT levels. Many correlations are weak, suggesting no strong linear relationship between these variables.

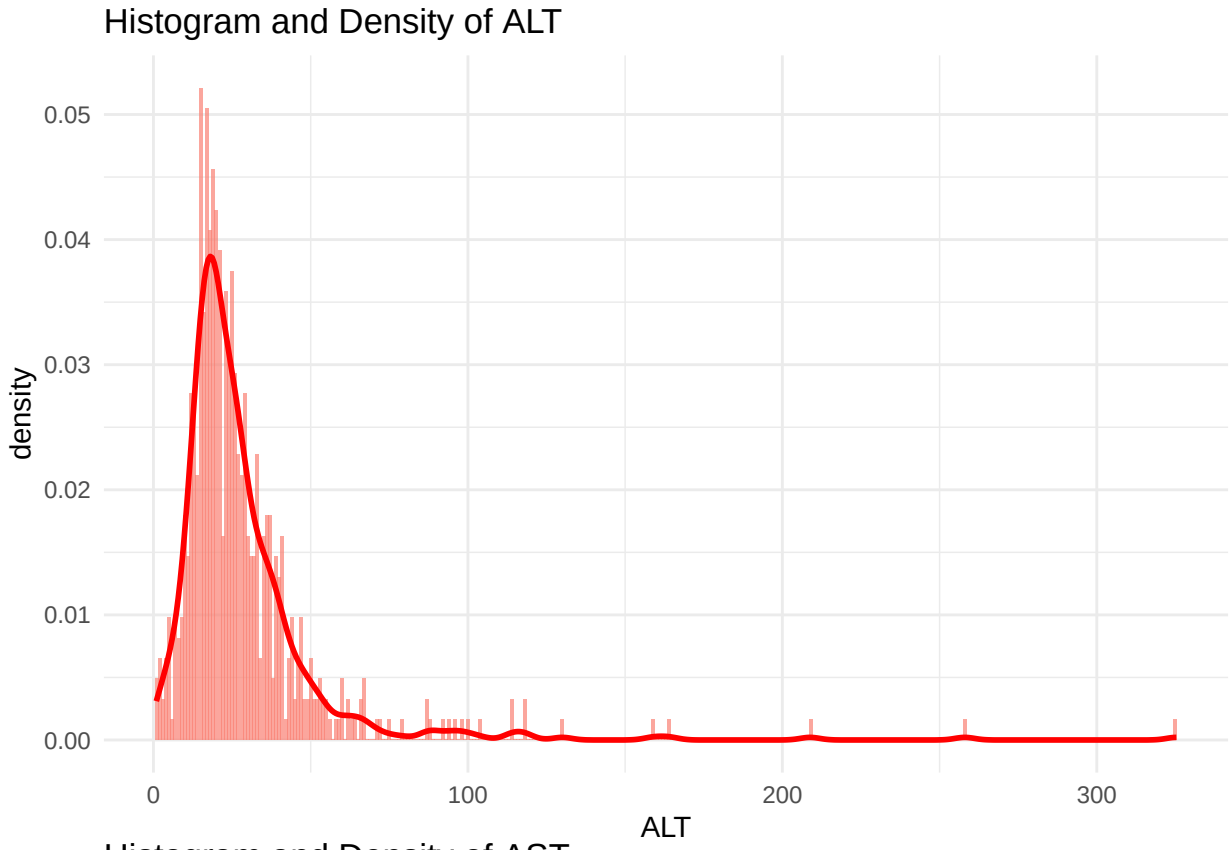
Evaluation of distribution

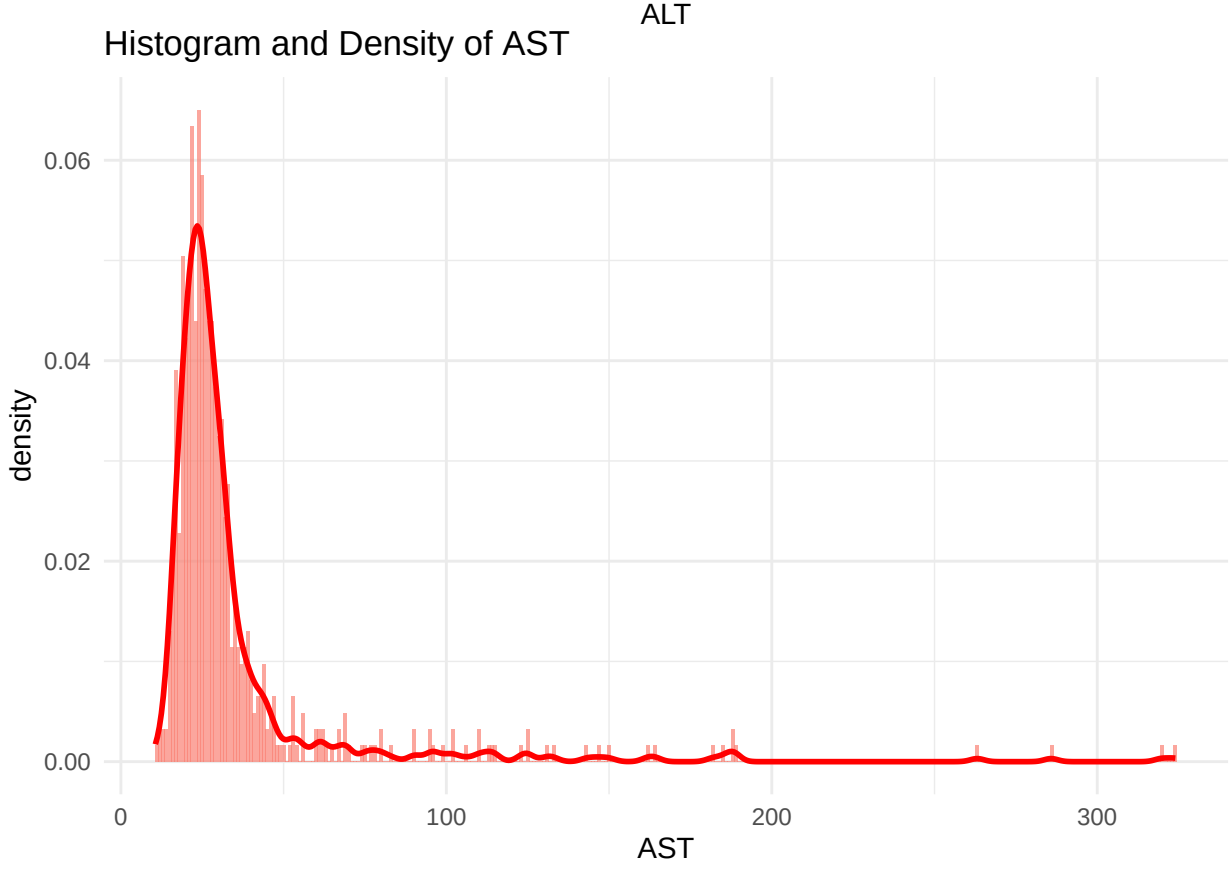
```
##       Age              ALB              ALP              ALT        
##  Min.   :19.00   Min.   :14.90   Min.   : 11.30   Min.   :  0.90  
##  1st Qu.:39.00   1st Qu.:38.80   1st Qu.: 52.50   1st Qu.: 16.40  
##  Median :47.00   Median :41.95   Median : 66.20   Median : 23.00  
##  Mean   :47.41   Mean   :41.62   Mean   : 68.28   Mean   : 28.45  
##  3rd Qu.:54.00   3rd Qu.:45.20   3rd Qu.: 80.10   3rd Qu.: 33.08  
##  Max.   :77.00   Max.   :82.20   Max.   :416.60   Max.   :325.30  
##                  NA's   :1       NA's   :18       NA's   :1       
##       AST              BIL              CHE              CHOL       
##  Min.   : 10.60   Min.   :  0.8   Min.   : 1.420   Min.   :1.430  
##  1st Qu.: 21.60   1st Qu.:  5.3   1st Qu.: 6.935   1st Qu.:4.610  
##  Median : 25.90   Median :  7.3   Median : 8.260   Median :5.300  
##  Mean   : 34.79   Mean   : 11.4   Mean   : 8.197   Mean   :5.368  
##  3rd Qu.: 32.90   3rd Qu.: 11.2   3rd Qu.: 9.590   3rd Qu.:6.060  
##  Max.   :324.00   Max.   :254.0   Max.   :16.410   Max.   :9.670  
##                                                    NA's   :10     
##       CREA              GGT              PROT       
##  Min.   :   8.00   Min.   :  4.50   Min.   :44.80  
##  1st Qu.:  67.00   1st Qu.: 15.70   1st Qu.:69.30  
##  Median :  77.00   Median : 23.30   Median :72.20  
##  Mean   :  81.29   Mean   : 39.53   Mean   :72.04  
##  3rd Qu.:  88.00   3rd Qu.: 40.20   3rd Qu.:75.40  
##  Max.   :1079.10   Max.   :650.90   Max.   :90.00  
##                                     NA's   :1
```

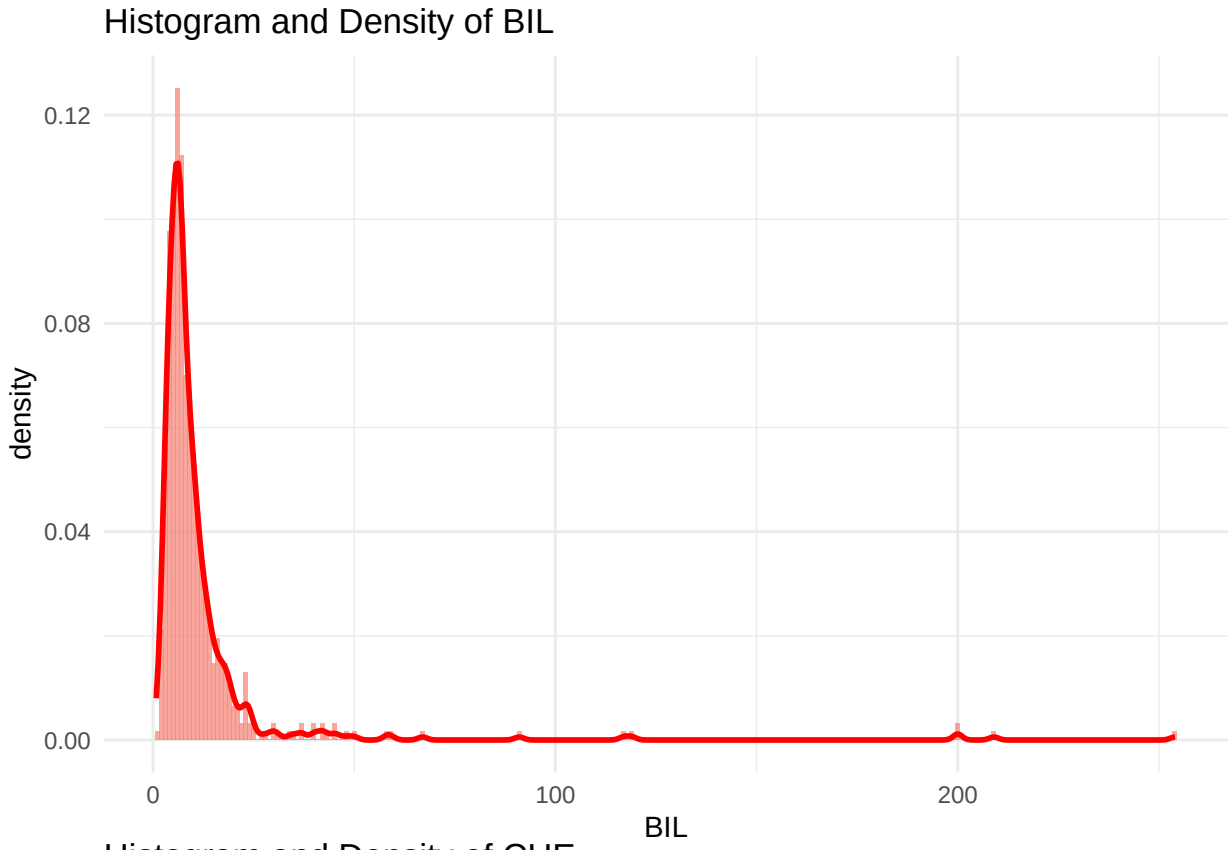


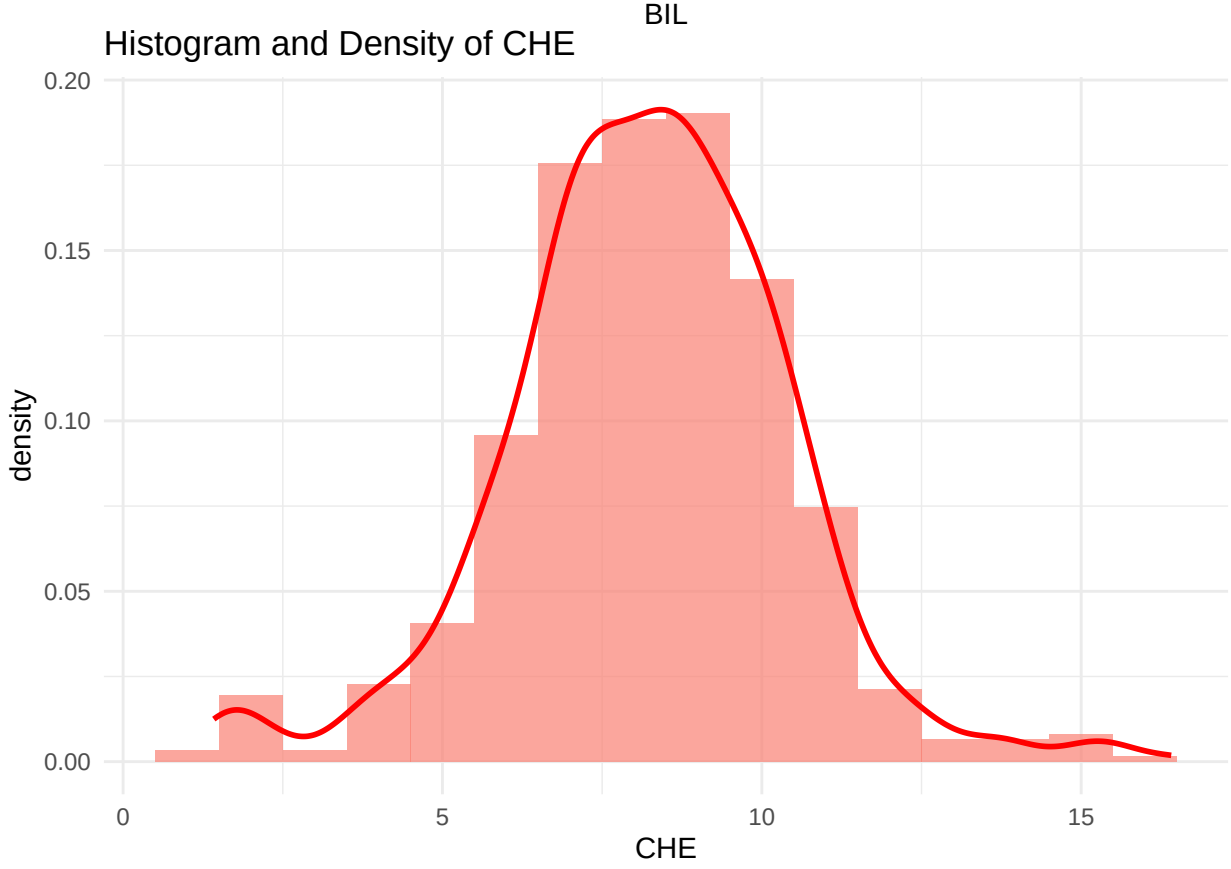

Histogram and Density of ALB
density
0.075
0.050
0.025
0.000
20
40
60
80
ALB

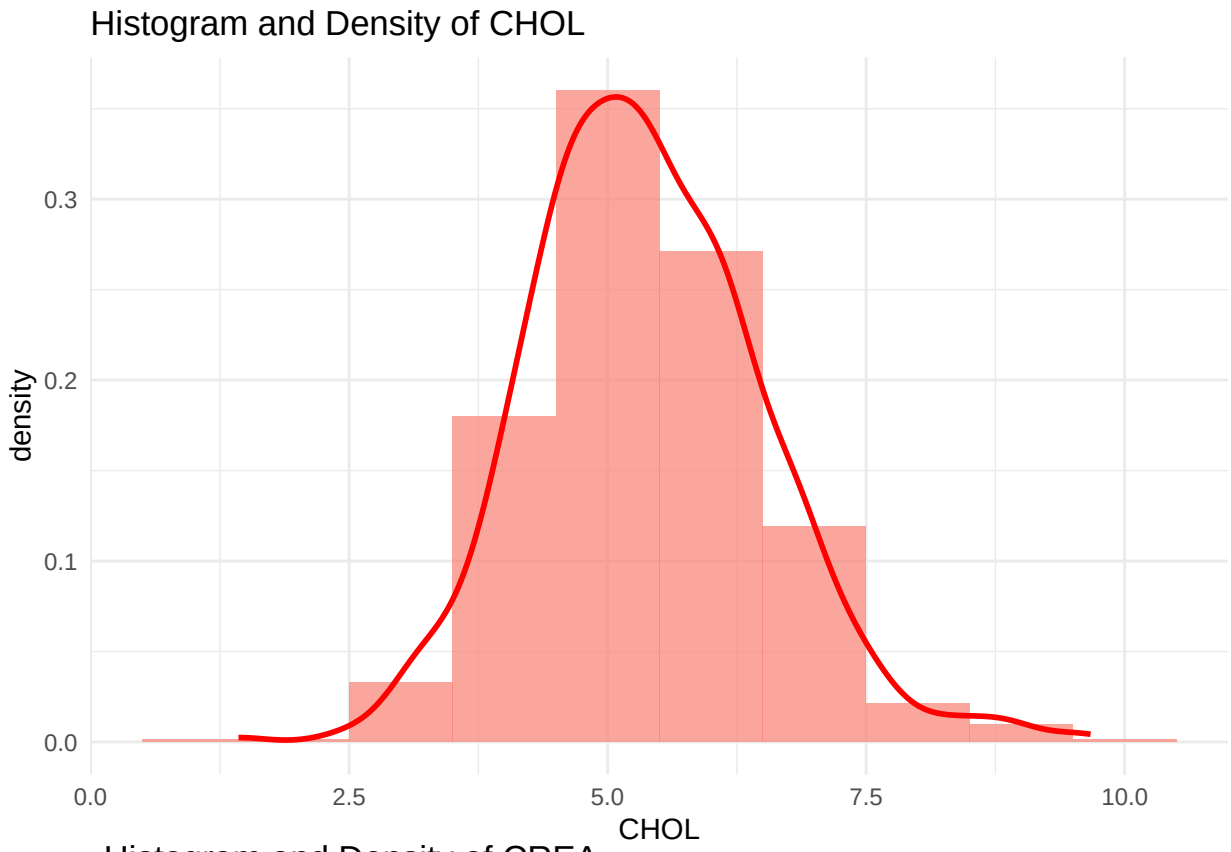

Histogram and Density of ALP
density
0.03
0.02
0.01
0.00
0
100
200
300
400
ALP

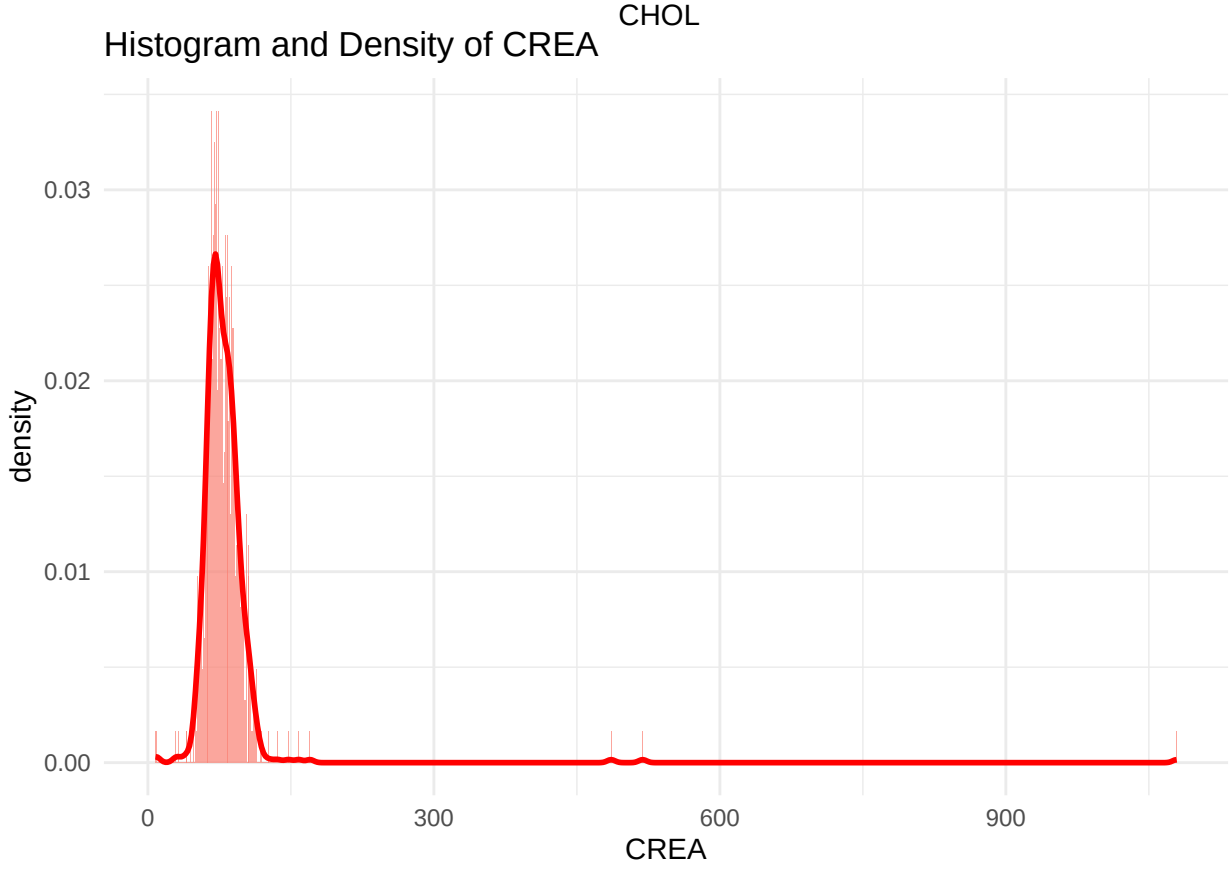

Histogram and Density of ALT
density
0.05
0.04
0.03
0.02
0.01
0.00
0
100
200
300
ALT

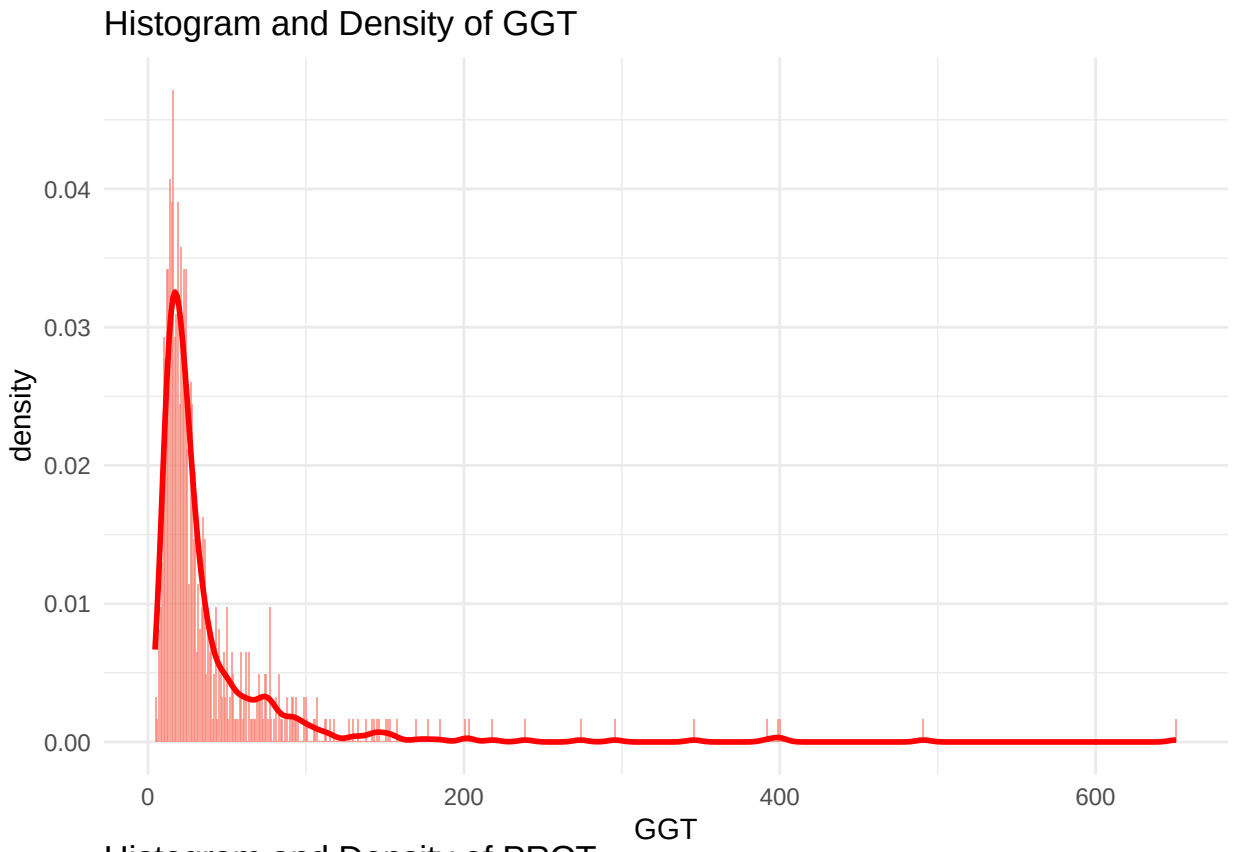

Histogram and Density of AST
density
0.06
0.04
0.02
0.00
0
100
200
300
AST

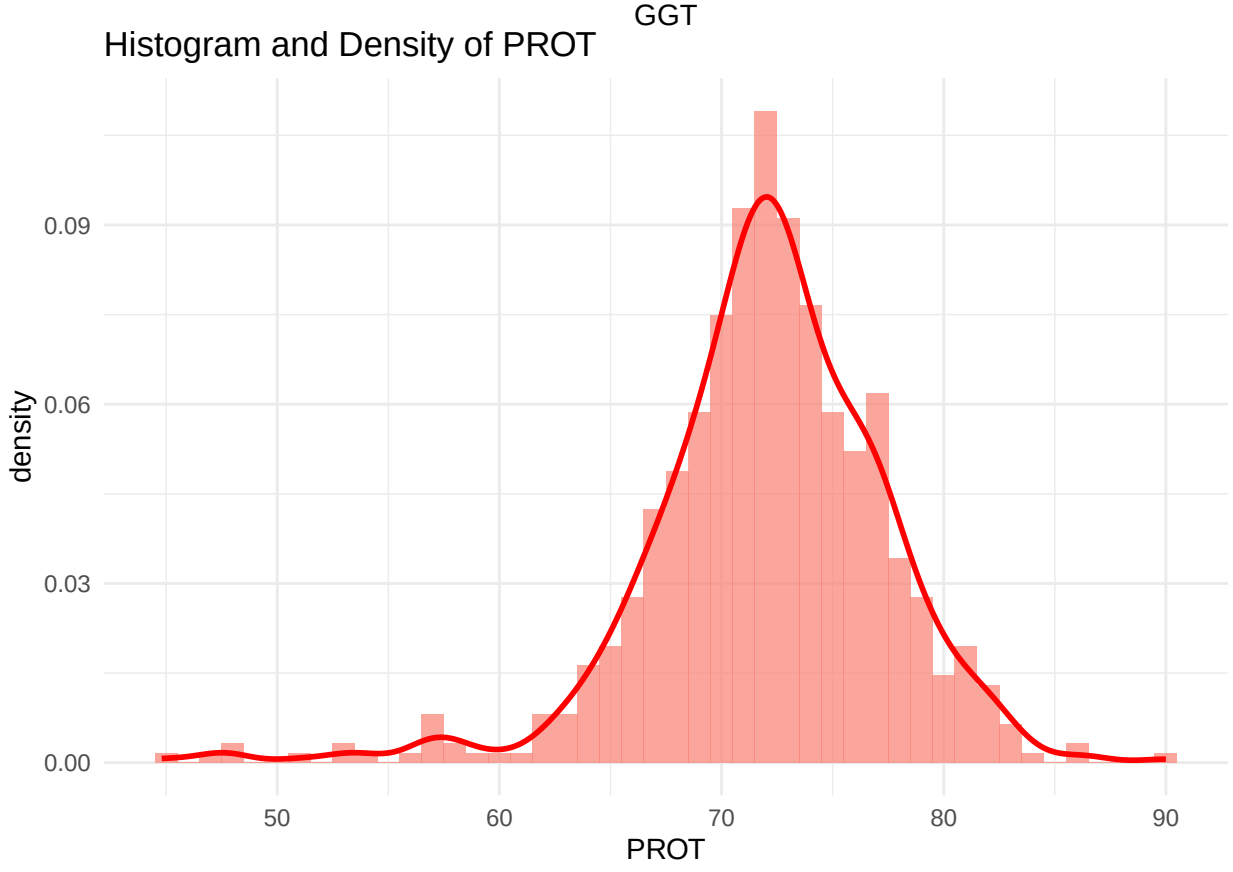

Histogram and Density of BIL
0.12
0.08
0.04
0.00
density
0
100
200
BIL


Histogram and Density of CHE
0.20
0.15
0.10
0.05
0.00
density
0
5
10
15
CHE

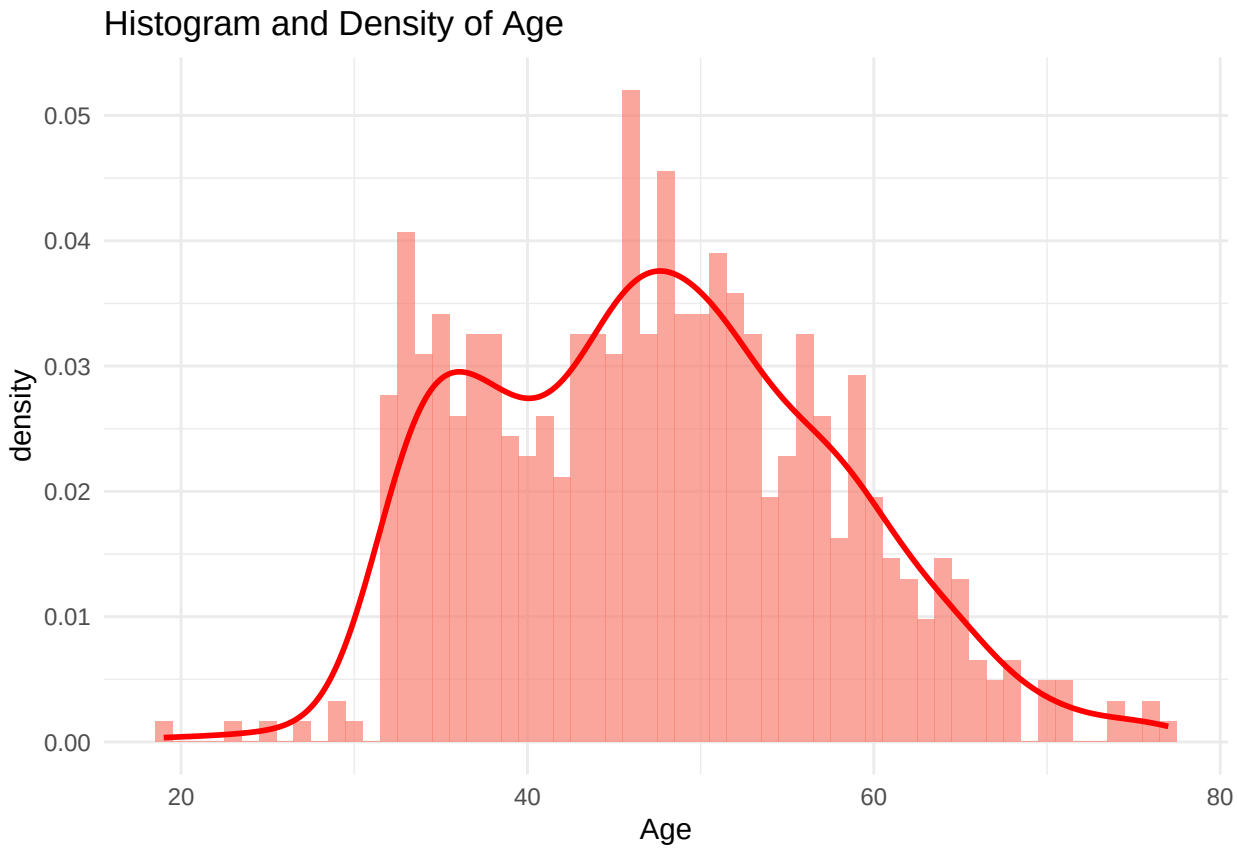

Histogram and Density of CHOL
density
0.0
0.1
0.2
0.3
0.0
2.5
5.0
7.5
10.0
CHOL


Histogram and Density of CREA
density
0.00
0.01
0.02
0.03
0
300
600
900
CREA

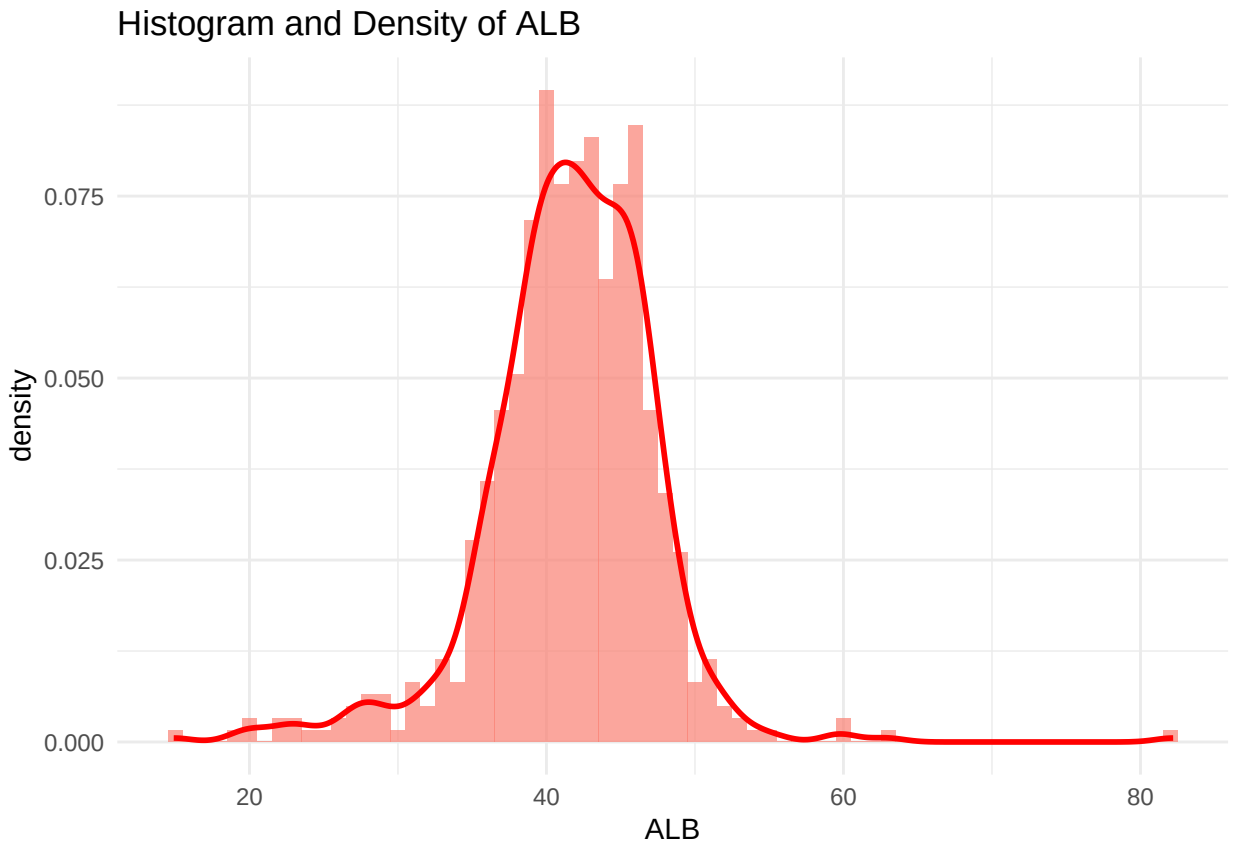

Histogram and Density of GGT
density
0.04
0.03
0.02
0.01
0.00
0
200
400
600
GGT


Histogram and Density of PROT
density
0.09
0.06
0.03
0.00
50
60
70
80
90
PROT

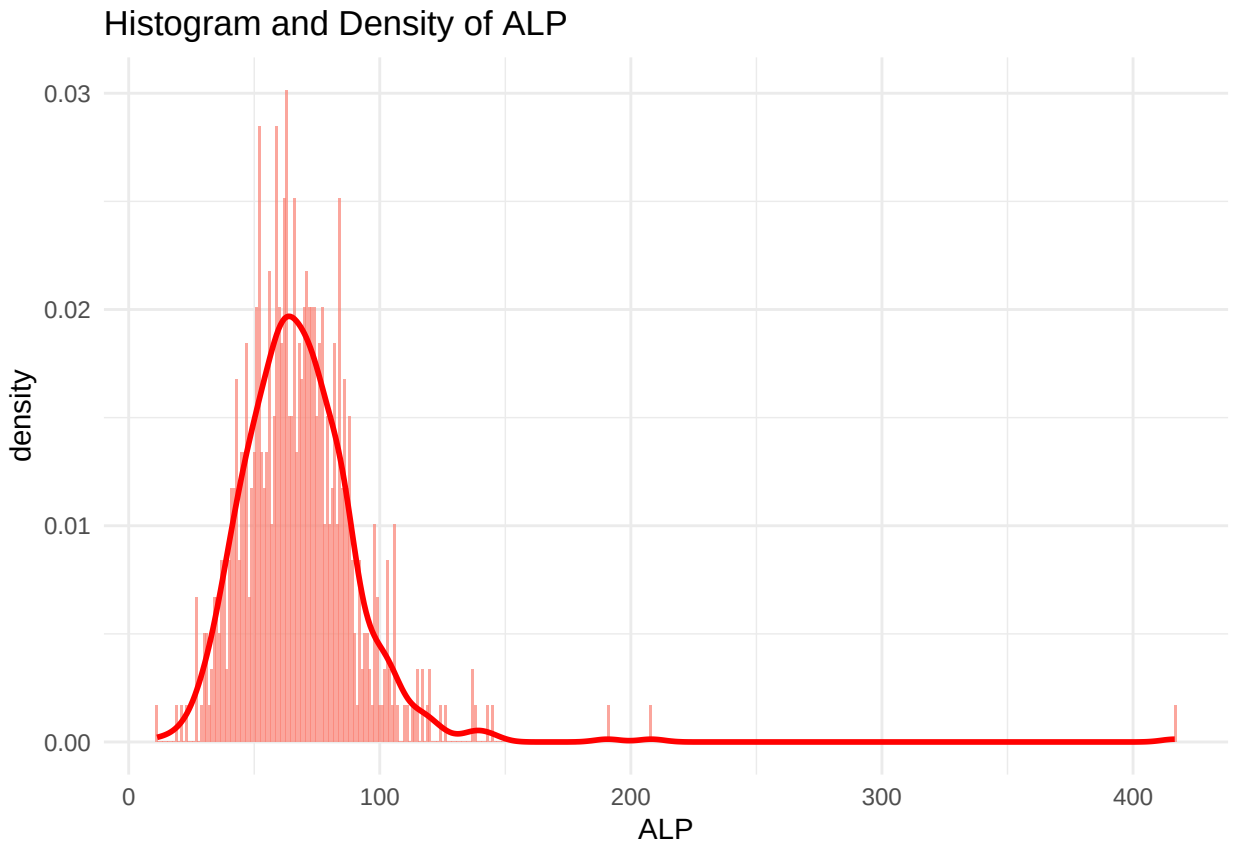
```
## [[1]]
```


Histogram and Density of Age
density
0.05
0.04
0.03
0.02
0.01
0.00
20
40
60
80
Age

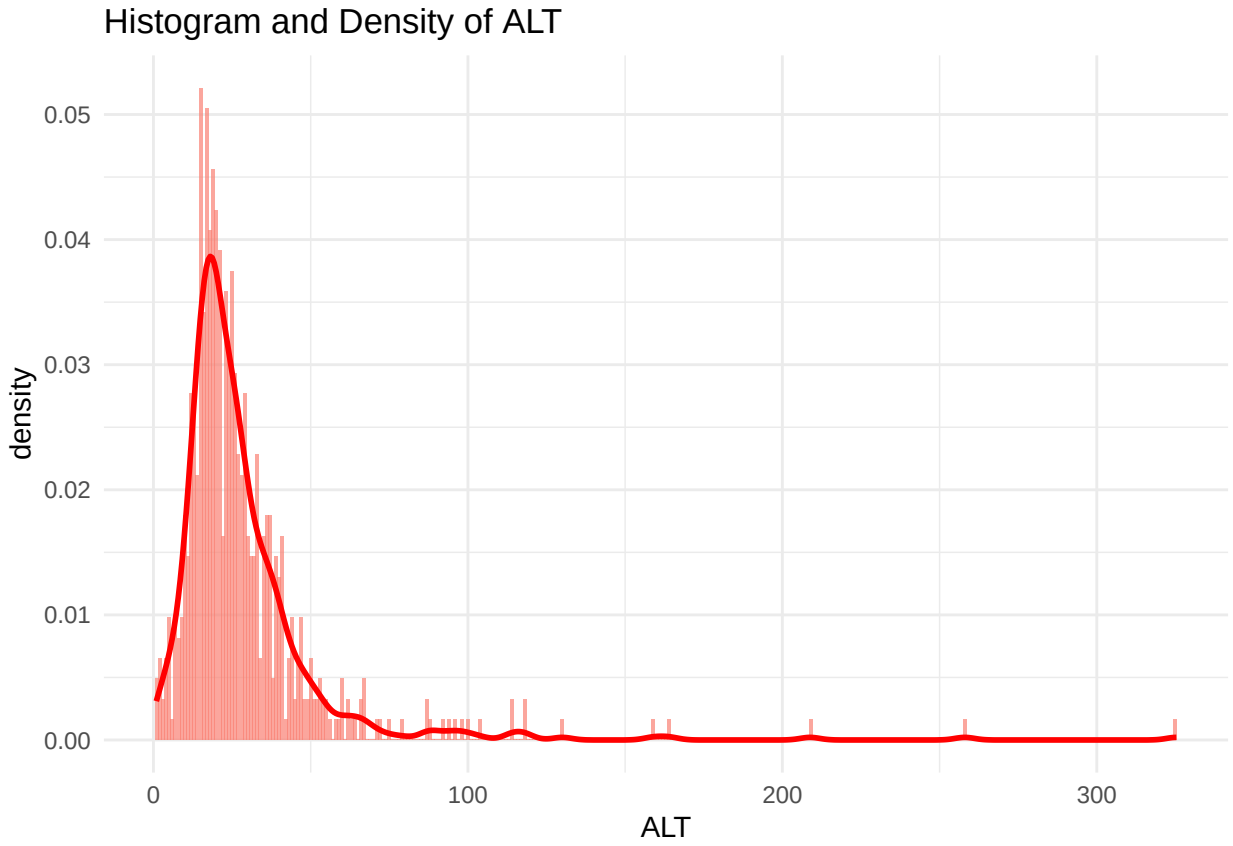
```
##
## [[2]]
```


Histogram and Density of ALB
density
0.075
0.050
0.025
0.000
20
40
60
80
ALB

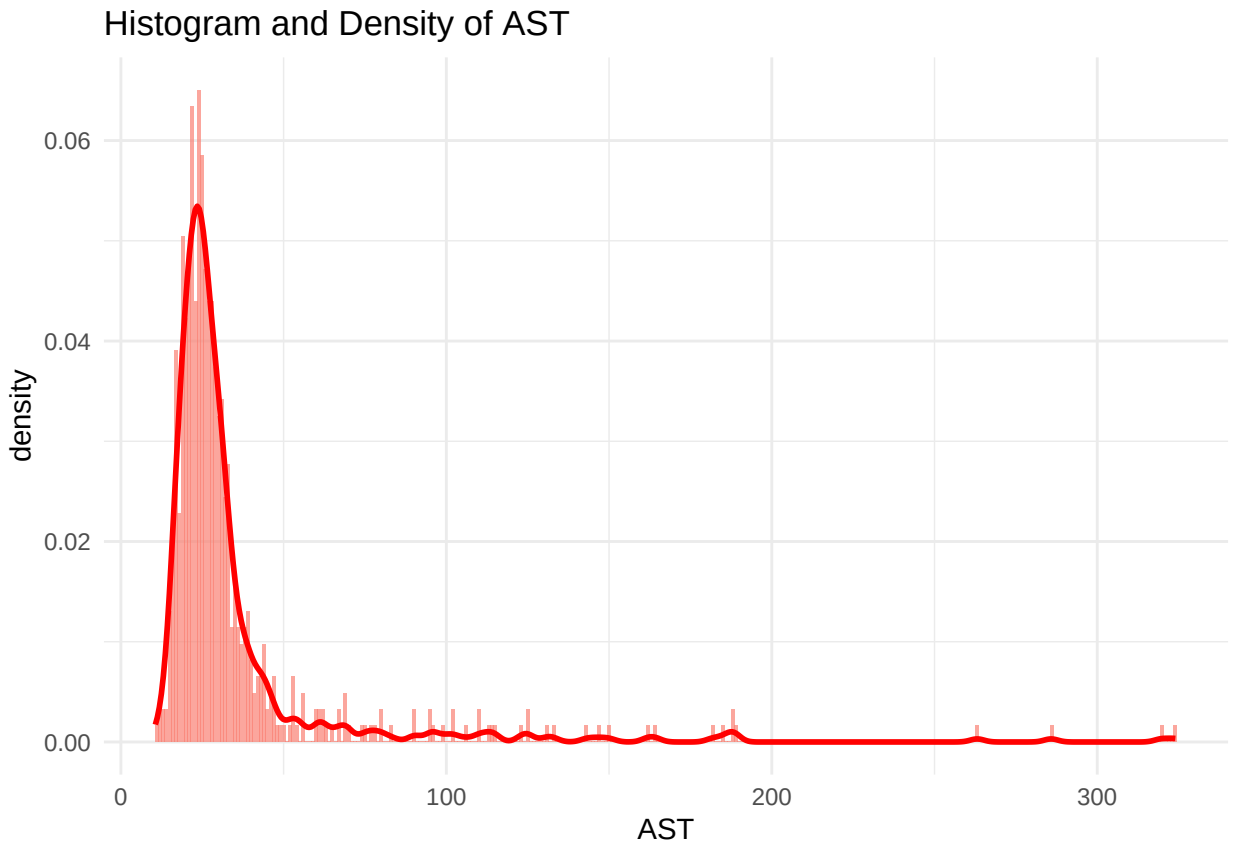
```
##
## [[3]]
```


Histogram and Density of ALP
0.03
0.02
0.01
0.00
density
0
100
200
300
400
ALP

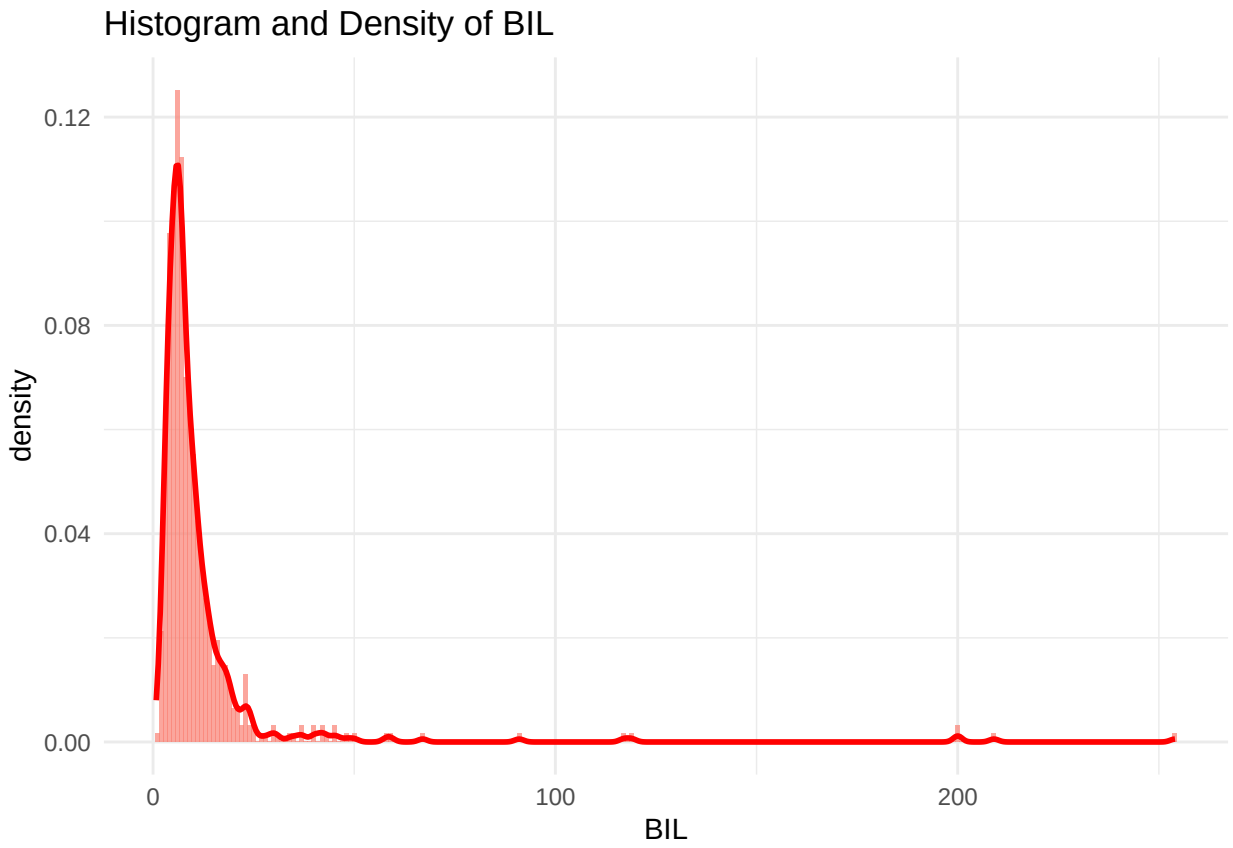
```
##
## [[4]]
```


Histogram and Density of ALT
density
0.05
0.04
0.03
0.02
0.01
0.00
0
100
200
300
ALT

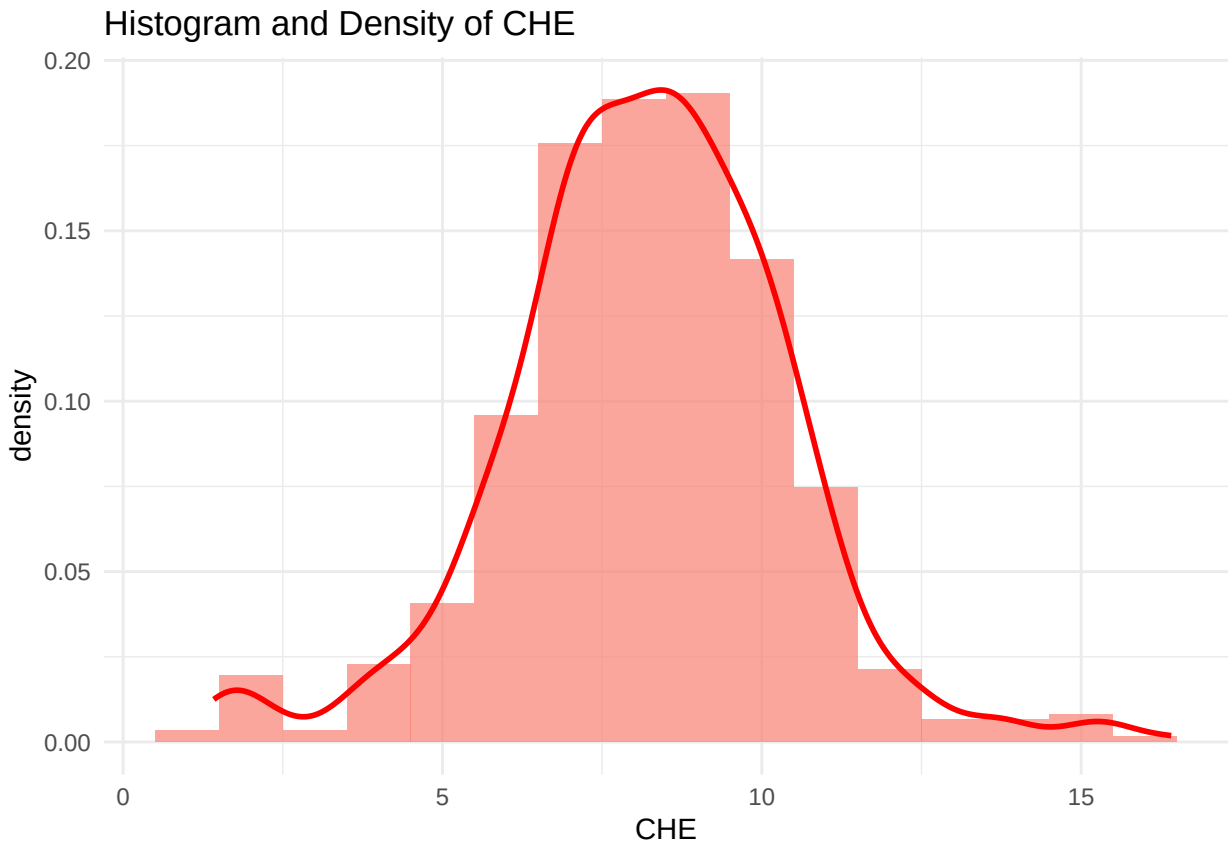
```
##
## [[5]]
```


Histogram and Density of AST
density
0.06
0.04
0.02
0.00
0
100
200
300
AST

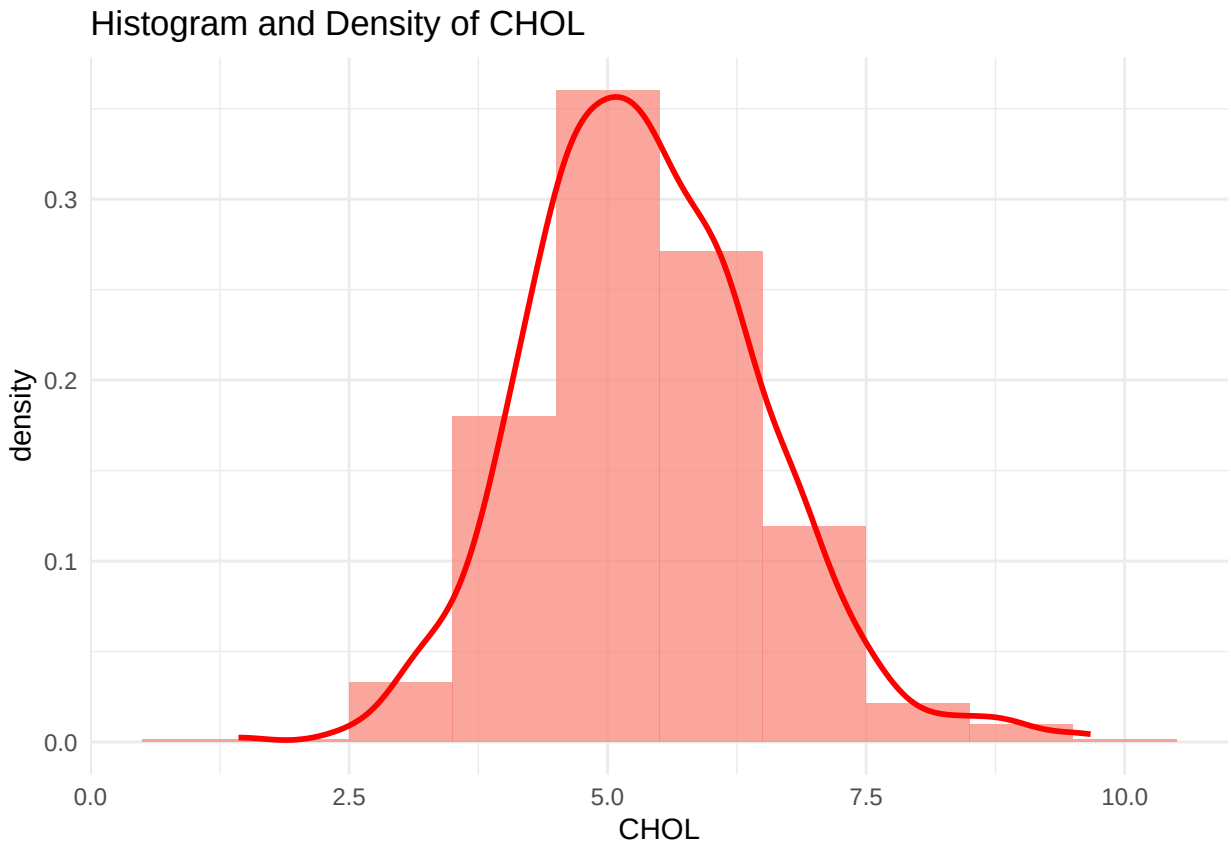
```
##
## [[6]]
```


Histogram and Density of BIL
0.12
0.08
0.04
0.00
density
0
100
200
BIL

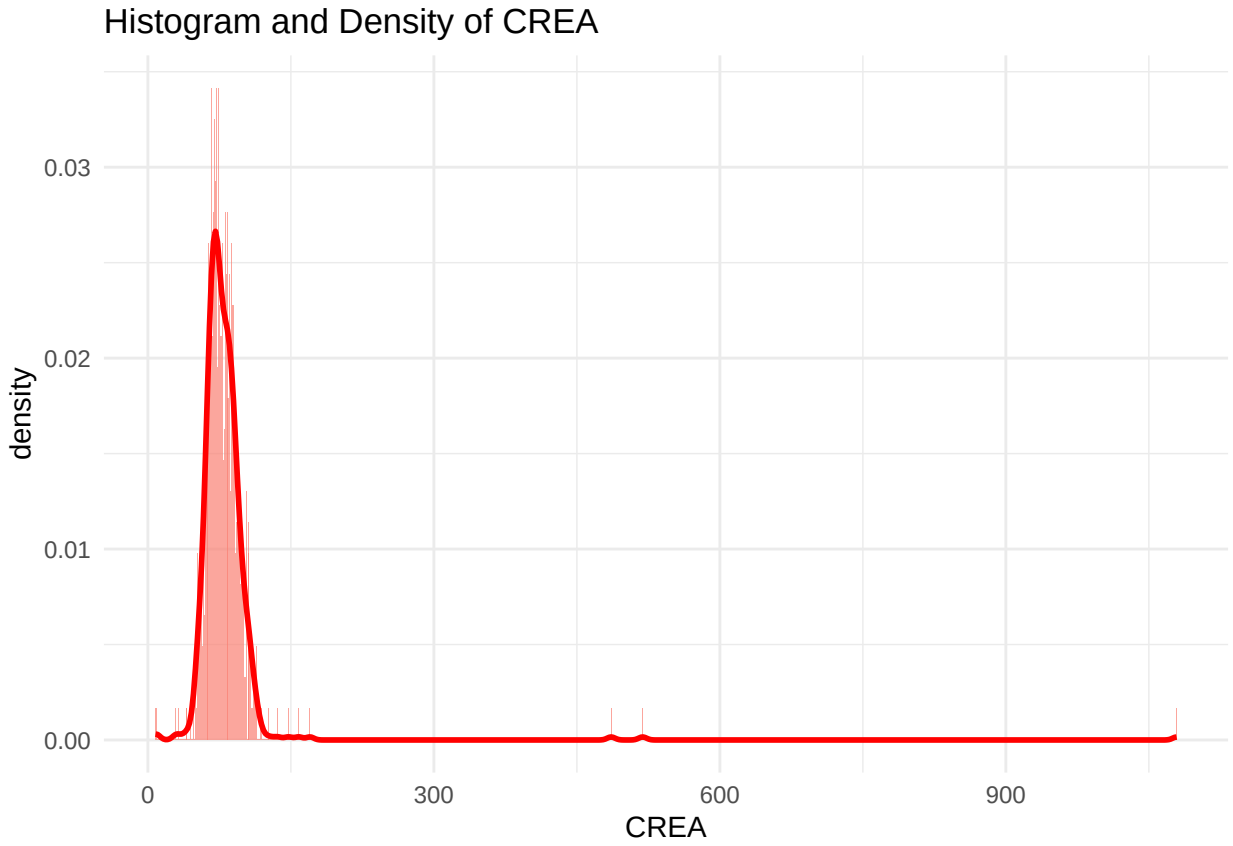
```
##
## [[7]]
```


Histogram and Density of CHE
0.20
0.15
0.10
0.05
0.00
density
0
5
10
15
CHE

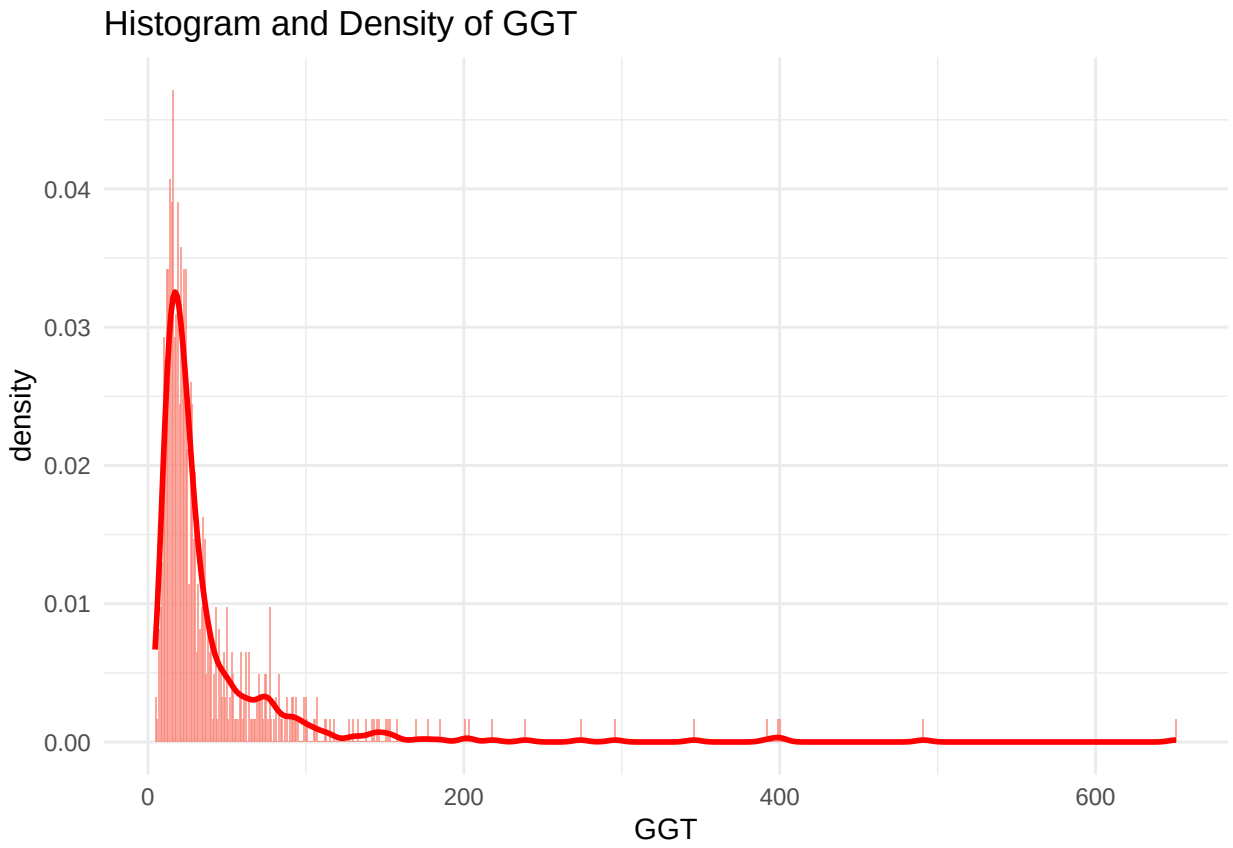
```
##
## [[8]]
```


Histogram and Density of CHOL
density
0.0
0.1
0.2
0.3
0.0
2.5
5.0
7.5
10.0
CHOL

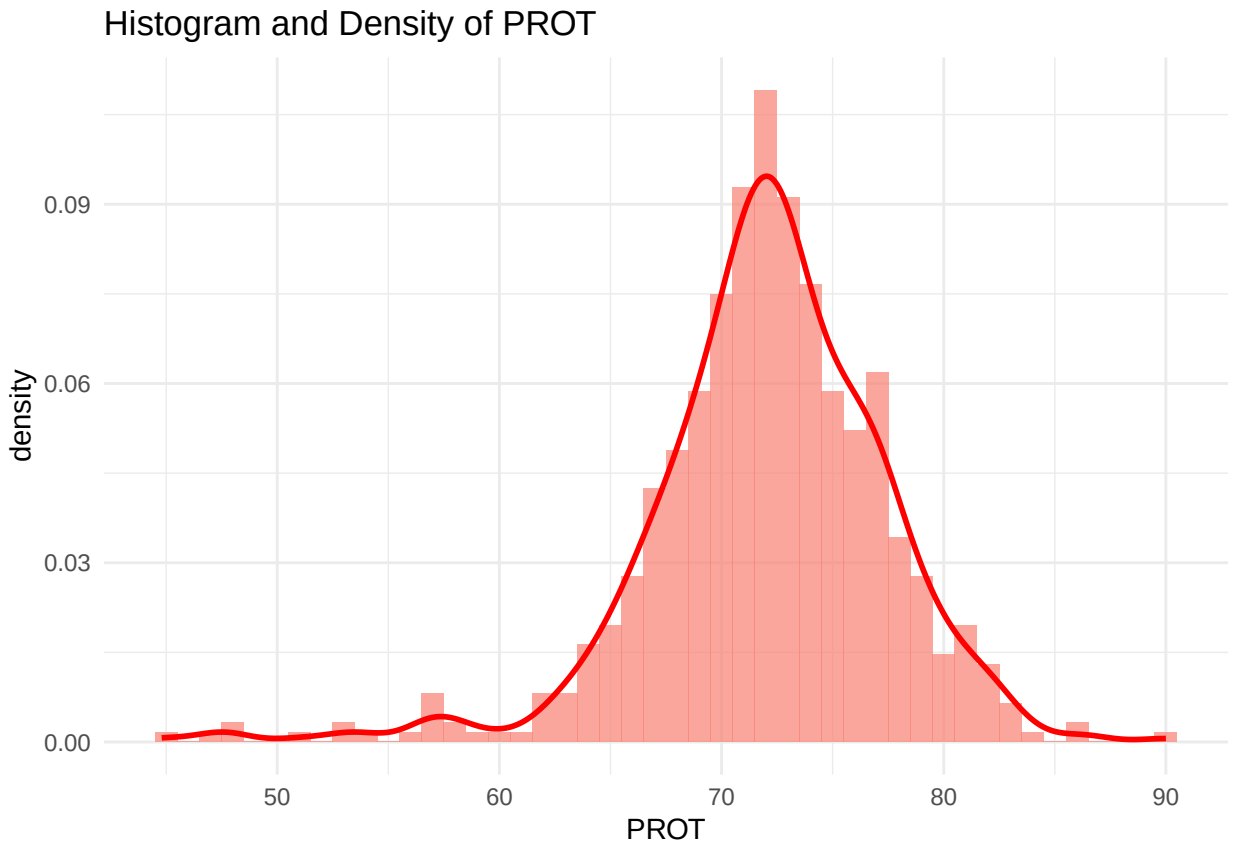
```
##
## [[9]]
```


Histogram and Density of CREA
density
0.03
0.02
0.01
0.00
0
300
600
900
CREA

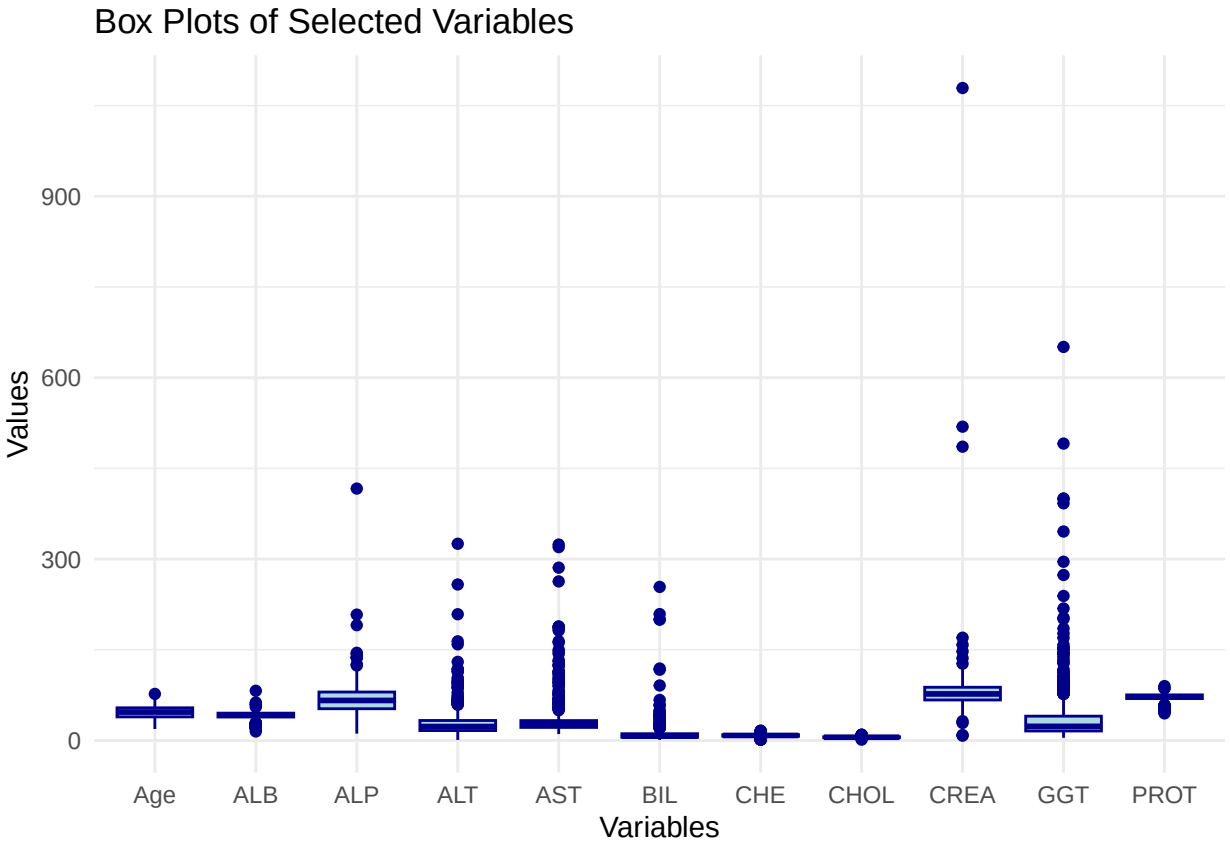
```
##
## [[10]]
```


Histogram and Density of GGT
density
0.04
0.03
0.02
0.01
0.00
0
200
400
600
GGT

```
##
## [[11]]
```




```
## No id variables; using all as measure variables
```



Age: Ranges from 19 to 77 years, with the average (mean) age being around 47 years. The distribution is roughly centered (median is 47), suggesting a fairly symmetrical age distribution.

ALB: Values range from 14.9 to 82.2, with a mean of approximately 41.6. The median is slightly lower than the mean, which might indicate a slight left skew in the distribution.

ALP: Has a wide range (11.3 to 416.6), and the mean (68.28) is higher than the median (66.2), indicating a right-skewed distribution.

ALT: The values vary significantly, ranging from 0.9 to 325.3, and the mean is higher than the median, also indicating a right-skewed distribution.

AST: This variable also shows a broad range and a higher mean compared to the median, suggesting right skewness.

BIL: The range is from 0.8 to 254, with a mean significantly higher than the median, indicating a distribution that is heavily right-skewed.

CHE: Ranges from 1.42 to 16.41, with a median very close to the mean, suggesting a more symmetrical distribution.

CHOL: Shows a range from 1.43 to 9.67 with the mean and median being quite close, indicating a more symmetric distribution.

CREA: Has a very wide range (8 to 1079.1) and the mean is higher than the median, pointing to a right-skewed distribution.

GGT: Exhibits a broad range (4.5 to 650.9) and a mean that is much higher than the median, indicating significant right skewness.

PROT: The range is from 44.8 to 90, with a mean close to the median, suggesting a relatively symmetric distribution.

It is evident that several variables have right-skewed distributions, as indicated by means that are larger than medians.

The presence of missing values ('NA's) in some variables should be addressed before performing further analysis.

3. Data Cleaning & Shaping

Identification of missing values

Identifying missing values in data is crucial for ensuring data quality and reliability, particularly in data analysis, statistics, and machine learning. It helps in making informed decisions, as missing data can lead to incorrect conclusions. In statistical and machine learning contexts, many methods require complete data; hence, missing values can distort results and affect model accuracy.

```
## Diagnosis       Age       Sex       ALB       ALP       ALT       AST       BIL
##         0         0         0         1        18         1         0         0
##       CHE      CHOL      CREA       GGT      PROT
##         0        10         0         0         1
```

Diagnosis, Age, Sex, AST, BIL, CHE, CREA, GGT: These variables have no missing values, indicating complete data for these parameters. This is crucial, especially for fundamental demographic information like Age and Sex, and key biochemical markers like AST, BIL, CHE, CREA, and GGT.

ALB, ALT, PROT: Each of these variables has 1 missing value.

ALP: There are 18 missing values. This is relatively higher compared to other variables and could be significant depending on the size of the dataset.

CHOL: It has 10 missing values.

Data imputation of missing data

Data imputation is essential in data analysis and machine learning for ensuring completeness and integrity of datasets. It addresses missing values, a common issue in real-world data, by filling gaps to prevent biased or invalid results. This process is crucial for maintaining accuracy in statistical models and analyses, particularly when complete data is required. Imputation improves model performance by providing a fully usable dataset for training and validation, thereby maximizing data utilization and avoiding significant information loss. It's a practical solution for handling everyday data inconsistencies, ensuring valuable data collected with effort and resources is fully leveraged in analysis.

As my data does not have too many missing values I will impute them with the median.

```
##      Diagnosis        Age            Sex                ALB
##  Hepatitis: 75   Min.   :19.00   Length:615         Min.   :14.90
##  Donor    :540   1st Qu.:39.00   Class :character   1st Qu.:38.80
##                  Median :47.00   Mode  :character   Median :41.95
##                  Mean   :47.41                      Mean   :41.62
##                  3rd Qu.:54.00                      3rd Qu.:45.20
##                  Max.   :77.00                      Max.   :82.20
##       ALP              ALT              AST              BIL
##  Min.   : 11.30   Min.   :  0.90   Min.   : 10.60   Min.   :  0.8
##  1st Qu.: 52.95   1st Qu.: 16.40   1st Qu.: 21.60   1st Qu.:  5.3
```

```
##  Median : 66.20   Median : 23.00   Median : 25.90   Median :  7.3
##  Mean   : 68.22   Mean   : 28.44   Mean   : 34.79   Mean   : 11.4
##  3rd Qu.: 79.30   3rd Qu.: 33.05   3rd Qu.: 32.90   3rd Qu.: 11.2
##  Max.   :416.60   Max.   :325.30   Max.   :324.00   Max.   :254.0
##       CHE              CHOL             CREA             GGT
##  Min.   : 1.420   Min.   :1.430   Min.   :   8.00   Min.   :  4.50
##  1st Qu.: 6.935   1st Qu.:4.620   1st Qu.:  67.00   1st Qu.: 15.70
##  Median : 8.260   Median :5.300   Median :  77.00   Median : 23.30
##  Mean   : 8.197   Mean   :5.367   Mean   :  81.29   Mean   : 39.53
##  3rd Qu.: 9.590   3rd Qu.:6.055   3rd Qu.:  88.00   3rd Qu.: 40.20
##  Max.   :16.410   Max.   :9.670   Max.   :1079.10   Max.   :650.90
##       PROT
##  Min.   :44.80
##  1st Qu.:69.30
##  Median :72.20
##  Mean   :72.04
##  3rd Qu.:75.40
##  Max.   :90.00
```

We can see that the dataset no longer has NA/missing values is any of the columns.

Normalization feature values

I will use Min-Max Normalization technique. This technique rescales the feature values to a fixed range, typically between 0 and 1.

Each element x in the vector (or column in a data frame) has the minimum value of that vector subtracted from it. This shifts the entire distribution so that the lowest value becomes 0. The result is then divided by the range of the vector (i.e., the maximum value minus the minimum value). This scaling step transforms the data so that the highest value in the original dataset corresponds to 1.

The effect of this normalization is that all values are rescaled into the range [0, 1] while maintaining the relative differences in the original dataset.

```r
# Normalization
normalize <- function(x) {
  return((x - min(x, na.rm = TRUE)) / (max(x, na.rm = TRUE) - min(x, na.rm = TRUE)))
}

data_norm <- as.data.frame(lapply(imputed_data, function(x) if(is.numeric(x)) normalize(x) else x))
```

4. Model Construction

Creation of training & validation subsets

Creating training and validation subsets is crucial in machine learning for model training and evaluation. The dataset is typically divided into two parts: a larger training set for model learning and a smaller validation set for model assessment and tuning. Random sampling is used for splitting to ensure representativeness, and stratification may be employed to maintain proportional representation of categories in both sets. Setting a random seed ensures reproducibility.

I will use 70/30 split ratios for training/validation.

Dummy Encoding for Nominal Predictors

Creation of model A with proper data encoding: Logistic Regression

I begins by specifying the model using logistic_reg(), and importantly, it includes a tunable regularization parameter (penalty = tune()). This indicates that the optimal value for this penalty, which is crucial for preventing overfitting, will be determined later through a model tuning process. The model is configured to use L2 regularization exclusively (mixture = 1), aligning with Ridge regression techniques. For the computational aspect, the glmnet chosen as the engine (set_engine("glmnet")).

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

I then set up a comprehensive workflow that includes the logistic regression model with its tuning parameters and preprocessing steps. Additionally, I prepares a grid of penalty values to be used in the tuning process, which is a crucial step for optimizing the model's performance by finding the best regularization strength.

```
# Joining Model and Processing Recipe
lr <- workflow() %>%
  add_model(lr_model) %>%
  add_recipe(s)

lr_grid <- tibble(penalty = 10**seq(-4, 0, length.out = 30))
```

Next, I set up a stratified, repeated 10-fold cross-validation process using the R function `vfold_cv`. In this process, the training dataset `df_train` is divided into 10 distinct folds, ensuring that each fold has a similar distribution of the "Diagnosis" variable through stratification. This approach helps in maintaining the representation of different diagnoses across all folds. The cross-validation is set to repeat 5 times, providing multiple rounds of validation. This method is essential for evaluating the model's performance and generalizability, ensuring that the assessment is robust, minimizes bias, and accurately reflects how the model might perform on unseen data.

```
# Stratified, Repeated 10-fold Cross-Validation
cv_lr <- vfold_cv(df_train, strata = "Diagnosis", v = 10, repeats = 5)
```

I plan to utilize the metric_set function to establish a robust evaluation framework for my classification model. This framework will incorporate two key metrics: roc_auc and j_index. By including roc_auc, I'll be able to determine how well the model distinguishes between different classes, with an emphasis on achieving a high AUC-ROC score for optimal performance. Concurrently, the inclusion of j_index will allow me to assess the model's diagnostic accuracy, emphasizing the balance between sensitivity and specificity which is crucial in scenarios where false positives and negatives carry significant consequences.

```
# Metrics
cls <- metric_set(roc_auc, j_index)
```

In the next step of my analysis, I proceed to tune my logistic regression model using the tune_grid function from the tidymodels suite in R. This involves setting up a grid of potential hyperparameters, which I've defined in lr_grid. The tune_grid function will iterate through these hyperparameters to find the most effective combination for my model. To evaluate the performance of each model configuration, I'm using cross-validation, indicated by the resamples = cv argument. This approach helps ensure that the model performs well across different segments of the data, reinforcing its generalizability and robustness. Additionally, by

setting control = control_grid(save_pred = TRUE), I make sure to save all predictions from each iteration. This is crucial for a thorough analysis of the model's performance under various conditions. Finally, I assess the model's effectiveness using a set of classification metrics, which I refer to as cls in my script. Adopting this structured approach to tune the logistic regression model is essential for optimizing its performance and ensuring its accuracy in my predictive tasks.

```
# Model Tuning
lr_res <- tune_grid(lr,
              grid = lr_grid,
              resamples = cv_lr,
              control = control_grid(save_pred = TRUE),
              metrics = cls)
```

In this phase of my analysis, I focus on identifying the best models from the tuning process based on their performance in terms of Area Under the ROC Curve (AUC). To achieve this, I use the show_best function, which is designed to rank the models according to their performance metrics. Specifically, I apply this function to the results of my logistic regression model tuning, stored in lr_res, and I specify the metric as roc_auc, which represents the AUC. The command best_mean_AUC <- show_best(lr_res, metric = "roc_auc") captures the top models with the highest AUC values. By storing these results in best_mean_AUC, I can easily review and analyze which models performed best during the tuning process based on their ability to distinguish between the classes effectively, as indicated by their AUC scores. This step is crucial as it helps me in selecting the most promising model for further analysis or deployment in practical applications.

```
## # A tibble: 5 x 7
##   penalty .metric .estimator  mean     n std_err .config
##     <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 0.0304  roc_auc binary     0.944    50  0.0101 Preprocessor1_Model19
## 2 0.0221  roc_auc binary     0.943    50  0.0104 Preprocessor1_Model18
## 3 0.0161  roc_auc binary     0.942    50  0.0106 Preprocessor1_Model17
## 4 0.0117  roc_auc binary     0.941    50  0.0108 Preprocessor1_Model16
## 5 0.00853 roc_auc binary     0.940    50  0.0110 Preprocessor1_Model15
```

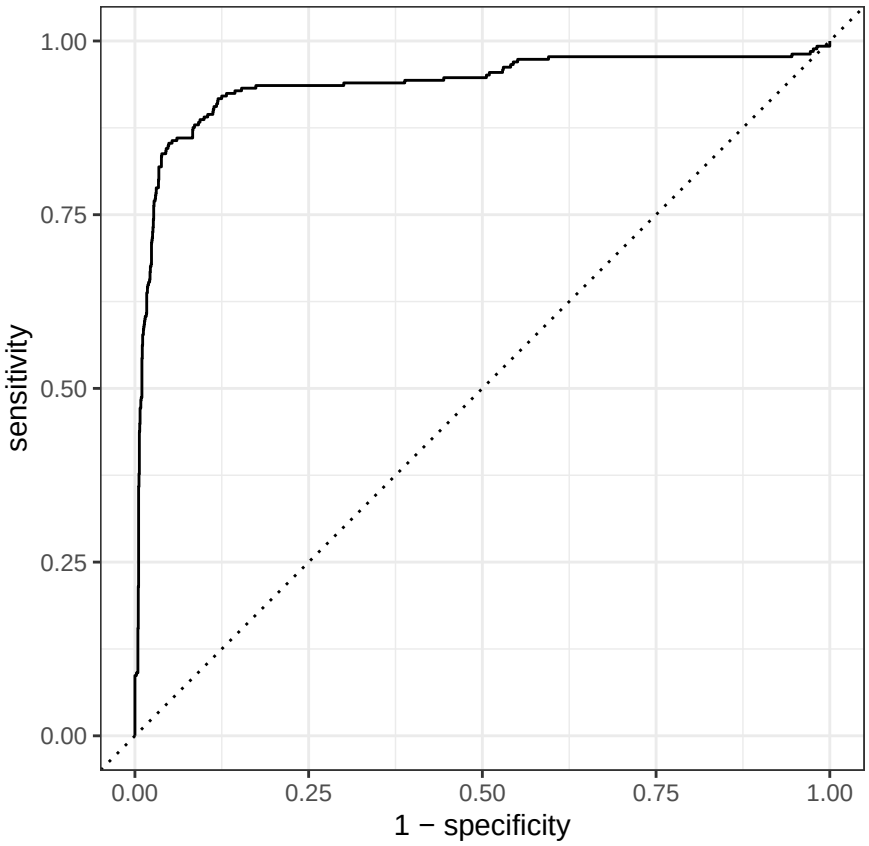
In the next stage of my analysis, I turn my attention to identifying the top-performing models based on the J-index, a metric that combines sensitivity and specificity to evaluate the performance of binary classifiers. To do this, I employ the show_best function again, but this time with a focus on the 'j_index' metric. By executing the command best_j_index <- show_best(lr_res, metric = "j_index"), I am able to sort and retrieve the models from my logistic regression results, lr_res, that have the highest J-index scores. Storing these results in best_j_index allows me to easily review which models have the most balanced trade-off between true positive rate and true negative rate. This is a crucial step in my analysis as it helps me to pinpoint models that not only perform well overall but also maintain a healthy balance in correctly identifying positive and negative cases, which is vital in many practical scenarios, especially in medical or binary classification contexts.

```
## # A tibble: 5 x 7
##    penalty .metric .estimator  mean     n std_err .config
##      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 0.000259 j_index binary     0.823    50  0.0180 Preprocessor1_Model04
## 2 0.000672 j_index binary     0.823    50  0.0180 Preprocessor1_Model07
## 3 0.000489 j_index binary     0.822    50  0.0181 Preprocessor1_Model06
## 4 0.000356 j_index binary     0.822    50  0.0181 Preprocessor1_Model05
## 5 0.0001   j_index binary     0.820    50  0.0177 Preprocessor1_Model01
```

In this step of my analysis, I am focusing on selecting the best hyperparameters for my logistic regression model. The reason I'm doing this is to optimize my model's performance. To achieve this, I'm using the select_best function, which helps in choosing the hyperparameters that lead to the highest performance based on a specific metric. I've chosen 'roc_auc' as my metric, standing for the Area Under the ROC Curve, because it's a crucial measure of a model's ability to accurately distinguish between different classes. By executing lr_best <- select_best(lr_res, metric = "roc_auc"), I am sifting through the tuning results of my model to identify and store the set of hyperparameters that deliver the best AUC score. This step is essential for me because it ensures that my logistic regression model is fine-tuned to be as effective and accurate as possible in its predictive tasks.

```
## # A tibble: 1 x 2
##   penalty .config
##     <dbl> <chr>
## 1  0.0304 Preprocessor1_Model19
```

I first identified the best logistic regression model based on the ROC-AUC metric using select_best. Then, I focused on evaluating the ROC-AUC of this selected model. I extracted the model's predictions and computed its ROC curve using a series of functions in R. This involved collecting predictions from the best model, calculating the ROC curve based on actual and predicted values, and labeling the results as 'Logistic Regression' for clarity. Finally, I visualized this ROC curve using the autoplot function, which provided an intuitive graphical representation of the model's performance in differentiating between positive and negative classes. This step was crucial for understanding the model's effectiveness in terms of sensitivity and specificity.



In this phase of the analysis, I am finalizing and fitting a logistic regression model for predictive tasks. The primary goal is to create an optimized model capable of making accurate predictions.

I begin by combining the initial logistic regression model with the best-performing hyperparameters using the `finalize_model` function. This results in a model configuration that maximizes its predictive potential.

To structure the modeling process effectively, I establish a workflow, `final_lr_wf`, which encapsulates the finalized model and incorporates data preprocessing steps defined in the recipe denoted as `s`.

The model is then trained on the training dataset using the `fit` function, resulting in a fully trained logistic regression model, `fitted_lr_model`. This model is now equipped to provide accurate predictions on new data.

In summary, this phase ensures that the logistic regression model is fine-tuned with optimal hyperparameters, prepared for deployment, and ready to deliver accurate predictions in real-world applications.

```
# Final Model Fitting for Logistic Regression
final_lr_model <- finalize_model(lr_model, lr_best)
final_lr_wf <- workflow() %>%
  add_model(final_lr_model) %>%
  add_recipe(s)

# Fit the model on the training data
fitted_lr_model <- fit(final_lr_wf, data = df_train)
```

Creation of model B with proper data encoding: Random Forest

In this phase of my analysis, I am specifying a random forest classification model in R using the tidymodels framework. The primary objective is to create a robust and accurate model for classifying categorical outcomes. To achieve this, I have set up the model with several key configurations. Firstly, I've designated two critical parameters, mtry and min_n, to be tuned during the model training process. mtry controls the selection of variables at each split, while min_n determines the minimum number of data points needed for further splitting, ensuring that the model is optimized for accuracy. Additionally, I've chosen to build an ensemble of 50 decision trees within the random forest, a technique known for improving predictive performance. To execute the computations efficiently, I've specified the "ranger" engine.

```
# Specification
rf_model <-
  rand_forest(mtry = tune(), min_n = tune(), trees = 50) %>%
  set_engine("ranger") %>%
  set_mode("classification")
```

Next I'm creating a streamlined workflow in R that combines a random forest classification model (rf_model) with a data processing recipe (s). This workflow is designed to optimize the process of training the model and preparing the data for classification tasks. The model is integrated into the workflow, allowing data to flow through it during training and prediction. Simultaneously, the data processing recipe defines crucial preprocessing steps, transformations, and feature engineering procedures to enhance the data's suitability for classification. By merging these components, I establish an efficient framework that ensures data is properly processed before being used by the random forest model, enhancing the accuracy and reliability of classification predictions.

```
# Joining Model and Processing Recipe
rf <- workflow() %>%
  add_model(rf_model) %>%
  add_recipe(s)
```

I have configured a robust model evaluation strategy in R by implementing stratified and repeated 10-fold cross-validation. This approach ensures the thorough assessment of model performance while addressing

potential issues like class imbalance in the "Diagnosis" variable. Stratification guarantees that each cross-validation fold maintains the same proportion of the target variable, enhancing the representation of different classes. Additionally, I've repeated this process five times to ensure reliability and robustness in the evaluation. Each repetition involves splitting the data into ten folds, with nine for training and one for testing, allowing for multiple assessments of the model's accuracy and its ability to generalize to unseen data.

```
# Stratified, Repeated 10-fold Cross-Validation
cv <- vfold_cv(df_train, strata = "Diagnosis", v = 10, repeats = 5)
```

I have established a comprehensive set of evaluation metrics in R, termed "cls," to assess the performance of classification models effectively. This metric set comprises two key metrics: ROC AUC (Receiver Operating Characteristic Area Under the Curve) and the J-Index. ROC AUC measures a model's ability to discriminate between positive and negative classes, with higher values indicating superior discrimination. On the other hand, the J-Index combines sensitivity and specificity to provide a balanced performance measure, with higher values indicating better overall classification performance. By defining these metrics, I am well-equipped to thoroughly evaluate the model's classification capabilities and make informed decisions regarding its effectiveness and suitability for the task at hand.

```
# Metrics
cls <- metric_set(roc_auc, j_index)
```

I have initiated the model tuning process in R by using the tune_grid function to optimize a random forest classification model (rf). This involves exploring a grid of hyperparameters to identify the most suitable configuration. I've specified a grid size of 25, which defines a range of hyperparameter combinations to be evaluated. To ensure robustness and reliable evaluation, I'm utilizing stratified and repeated 10-fold cross-validation (cv) and saving predictions for analysis. Additionally, I've defined a set of evaluation metrics (cls) that includes ROC AUC and the J-Index to assess model performance comprehensively. This tuning process will help identify the best hyperparameter settings for the random forest model, ensuring its effectiveness in classifying categorical outcomes.

```
# Model Tuning
rf_res <- tune_grid(rf,
            grid = 25,
            resamples = cv,
            control = control_grid(save_pred = TRUE),
            metrics = cls)
```

```
## i Creating pre-processing data to finalize unknown parameter: mtry
```

I have identified the best models ranked by the Area Under the ROC Curve (AUC) as a measure of their classification performance. These models have been selected based on their ability to distinguish between positive and negative classes effectively. The variable "best_mean_AUC" contains information about these top-performing models, allowing further analysis and selection of the most suitable model for the task at hand.

```
## # A tibble: 5 x 8
##    mtry min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     4    14 roc_auc binary     0.987    50 0.00210 Preprocessor1_Model20
## 2     3    15 roc_auc binary     0.985    50 0.00375 Preprocessor1_Model18
## 3     3    12 roc_auc binary     0.985    50 0.00360 Preprocessor1_Model10
## 4     5     9 roc_auc binary     0.985    50 0.00267 Preprocessor1_Model09
## 5     4    34 roc_auc binary     0.985    50 0.00299 Preprocessor1_Model12
```

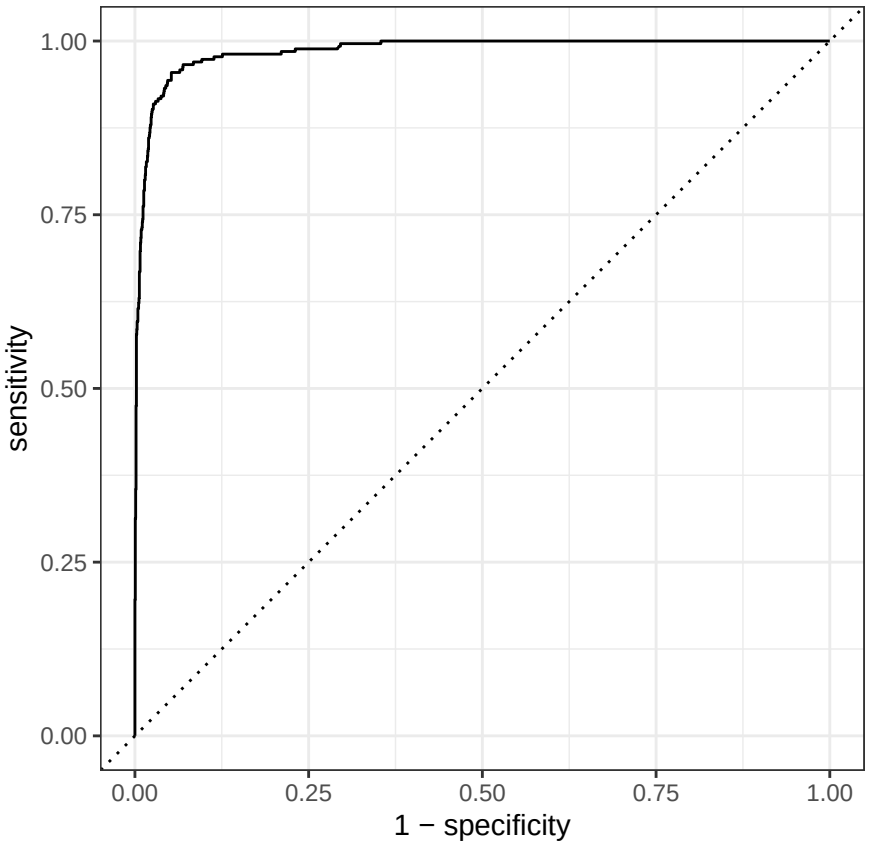
I have ranked the best models based on the J-Index, a performance metric that combines sensitivity and specificity to provide a balanced measure of classification performance. The "best_j_index" variable contains information about these top-performing models, allowing for further analysis and selection of the most suitable model for classification tasks, prioritizing both sensitivity and specificity.

```
## # A tibble: 5 x 8
##    mtry min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     2    31 j_index binary     0.892    50  0.0161 Preprocessor1_Model14
## 2     1    37 j_index binary     0.889    50  0.0187 Preprocessor1_Model11
## 3     3    19 j_index binary     0.882    50  0.0191 Preprocessor1_Model06
## 4     4    14 j_index binary     0.875    50  0.0191 Preprocessor1_Model20
## 5     3    15 j_index binary     0.874    50  0.0172 Preprocessor1_Model18
```

I have determined the best hyperparameters for the random forest classification model by selecting the configuration that optimizes the Area Under the ROC Curve (ROC AUC) metric. The "rf_best" variable now contains the hyperparameter settings that yield the highest AUC score, ensuring that the model is fine-tuned for optimal classification performance. These hyperparameters will be crucial in building an effective and accurate classification model.

```
## # A tibble: 1 x 3
##    mtry min_n .config
##   <int> <int> <chr>
## 1     4    14 Preprocessor1_Model20
```

I have computed the Receiver Operating Characteristic Area Under the Curve (ROC AUC) for the best-performing random forest classification model. This analysis allows us to visualize the model's ability to discriminate between positive and negative classes effectively. The ROC AUC curve provides insights into the trade-off between sensitivity and specificity, offering a comprehensive view of the model's classification performance.



I have successfully completed the final steps of preparing and fitting a random forest classification model in R. Initially, I configured the model with the best hyperparameters, ensuring optimal performance in classifying categorical outcomes. This final model, referred to as "final_rf_model," was then integrated into a workflow alongside a data processing recipe ("s"), creating a streamlined process for applying the model to the data. The culmination of this process was the fitting of the model to the training data, resulting in the "fitted_rf_model." This model is now well-equipped and fine-tuned to perform accurate and reliable classification tasks.

```
# Final Model Fitting for Random Forest
final_rf_model <- finalize_model(rf_model, rf_best)
final_rf_wf <- workflow() %>%
  add_model(final_rf_model) %>%
  add_recipe(s)

# Fit the model on the training data
fitted_rf_model <- fit(final_rf_wf, data = df_train)
```

Creation of model C with proper data encoding: Boosted Trees model

In this model specification, I have outlined the configuration for a boosted tree classification model using the `xgboost` engine. The goal is to fine-tune various hyperparameters to create an optimal predictive model. Key parameters to be tuned include `mtry`, which controls the number of randomly selected predictors at each tree split, `trees` specifying a total of 50 boosting iterations, `min_n` setting the minimum number of observations needed in a terminal node, and `tree_depth` managing the maximum depth of each tree. Additionally, the learning rate `learn_rate`, loss reduction threshold, `loss_reduction`, sample size `sample_size`, and stopping criteria for iterations `stop_iter` will all be optimized. This comprehensive approach ensures that

the resulting boosted tree classification model will be well-suited to address the specific characteristics and challenges of the dataset and problem at hand.

```
set.seed(123)
# Model Specification
bt_model <- boost_tree(
  mtry = tune(),
  trees = 50,
  min_n = tune(),
  tree_depth = tune(),
  learn_rate = tune(),
  loss_reduction = tune(),
  sample_size = tune(),
  stop_iter = tune()) %>%
  set_engine("xgboost") %>%
  set_mode("classification")
```

Next I am creating a workflow that combines a machine learning model and a data processing recipe. The model, defined as bt_model, is a boosted tree classification model with hyperparameters specified for optimization. The data processing recipe, represented by s, likely includes data preprocessing steps such as feature scaling, imputation, or feature engineering. These two components, the model and the recipe, are seamlessly integrated into the workflow. This workflow aims to ensure that the data is appropriately prepared and fed into the model for training and evaluation, ultimately leading to an effective and well-tuned boosted tree classification model.

```
# Joining Model and Processing Recipe
bt <- workflow() %>%
  add_model(bt_model) %>%
  add_recipe(s)
```

I am creating a cross-validation strategy for evaluating machine learning models. The cv_bt object is defined using the vfold_cv function, which sets up a 10-fold cross-validation. The 'strata' argument is set to 'Diagnosis,' which indicates that the cross-validation will be stratified based on the 'Diagnosis' variable. Additionally, the 'repeats' parameter is set to 5, meaning that the 10-fold cross-validation will be repeated 5 times. This approach helps ensure robust model evaluation by partitioning the data into training and validation sets multiple times while maintaining balance in the target variable's categories. It is a valuable technique for assessing model performance and generalization across different subsets of the dataset.

```
# Stratified, Repeated 10-fold Cross-Validation
cv_bt <- vfold_cv(df_train, strata = "Diagnosis", v = 10, repeats = 5)
```

I have created a metric set named cls that encompasses two essential classification performance metrics: roc_auc and j_index. These metrics serve as crucial tools for evaluating the effectiveness of binary classification models. The roc_auc metric quantifies the area under the Receiver Operating Characteristic (ROC) curve, offering insights into a model's discrimination ability. A higher roc_auc score signifies better classification performance, with perfection represented by a score of 1.0. On the other hand, the j_index metric, also known as Youden's J statistic, provides a single comprehensive statistic that combines sensitivity and specificity to assess the overall diagnostic accuracy of a model. With values ranging from -1 to 1, where 1 denotes perfect classification, 0 indicates random classification, and -1 signifies the worst performance, the j_index serves as a valuable complement to roc_auc for model evaluation. Together, these metrics empower the evaluation and comparison of binary classification models, aiding in the assessment of their predictive capabilities and discriminatory power

```
# Metrics
cls <- metric_set(roc_auc, j_index)
```

Next, I am performing hyperparameter tuning for the boosted tree classification model (bt) using the tune_grid function. This tuning process involves exploring a grid of 25 different hyperparameter combinations to optimize the model's performance. To assess the performance of each combination, I am using cross-validation as specified by the cv_bt object, which represents a stratified, repeated 10-fold cross-validation scheme. The control_grid argument is used to save prediction results (save_pred = TRUE) during the tuning process for later analysis. The evaluation of model performance is based on the classification metrics defined in the cls metric set, which includes roc_auc and j_index. This comprehensive model tuning procedure aims to identify the best combination of hyperparameters that will result in the highest classification performance on the validation sets within the cross-validation framework

```
# Model Tuning
bt_res <- tune_grid(bt,
            grid = 25,
            resamples = cv_bt,
            control = control_grid(save_pred = TRUE),
            metrics = cls)
```

I am determining the best models from the results of the hyperparameter tuning process (bt_res) based on their mean Area Under the Curve (AUC) scores. The show_best function is used to extract and rank these models. AUC is a critical measure of a model's ability to distinguish between classes, with higher AUC values indicating superior performance. By identifying the models with the highest mean AUC scores, I am pinpointing the hyperparameter configurations that have yielded the best discriminatory power during the cross-validation process. These top-performing models will serve as candidates for further evaluation and deployment in the classification task.

```
## # A tibble: 5 x 13
##    mtry min_n tree_depth learn_rate loss_reduction sample_size stop_iter .metric
##   <int> <int>      <int>      <dbl>          <dbl>       <dbl>     <int> <chr>
## 1     2     3         11    0.00166   0.0000400          0.370        15 roc_auc
## 2     4     4          2    0.0781    0.0000000175       0.179         4 roc_auc
## 3     2    10         14    0.270     3.26               0.470        10 roc_auc
## 4     4    29          9    0.218     0.000940           0.940        17 roc_auc
## 5     4     8         13    0.00612   0.000738           0.219         7 roc_auc
## # i 5 more variables: .estimator <chr>, mean <dbl>, n <int>, std_err <dbl>,
## #   .config <chr>
```

I am determining the best models from the results of the hyperparameter tuning process (bt_res) based on their J-index scores. The show_best function is employed to extract and rank these models using the J-index metric. The J-index is a valuable measure that combines sensitivity and specificity to evaluate the overall diagnostic accuracy of a binary classification model. Models with higher J-index scores are considered to have better overall classification accuracy. By identifying and presenting the best models based on the J-index, I am pinpointing hyperparameter configurations that excel in terms of both sensitivity and specificity, making them strong candidates for further evaluation and deployment in the classification task.

```
## # A tibble: 5 x 13
##    mtry min_n tree_depth learn_rate loss_reduction sample_size stop_iter .metric
##   <int> <int>      <int>      <dbl>          <dbl>       <dbl>     <int> <chr>
## 1     2     3         11    0.00166   0.0000400          0.370        15 j_index
## 2     2    10         14    0.270     3.26               0.470        10 j_index
```

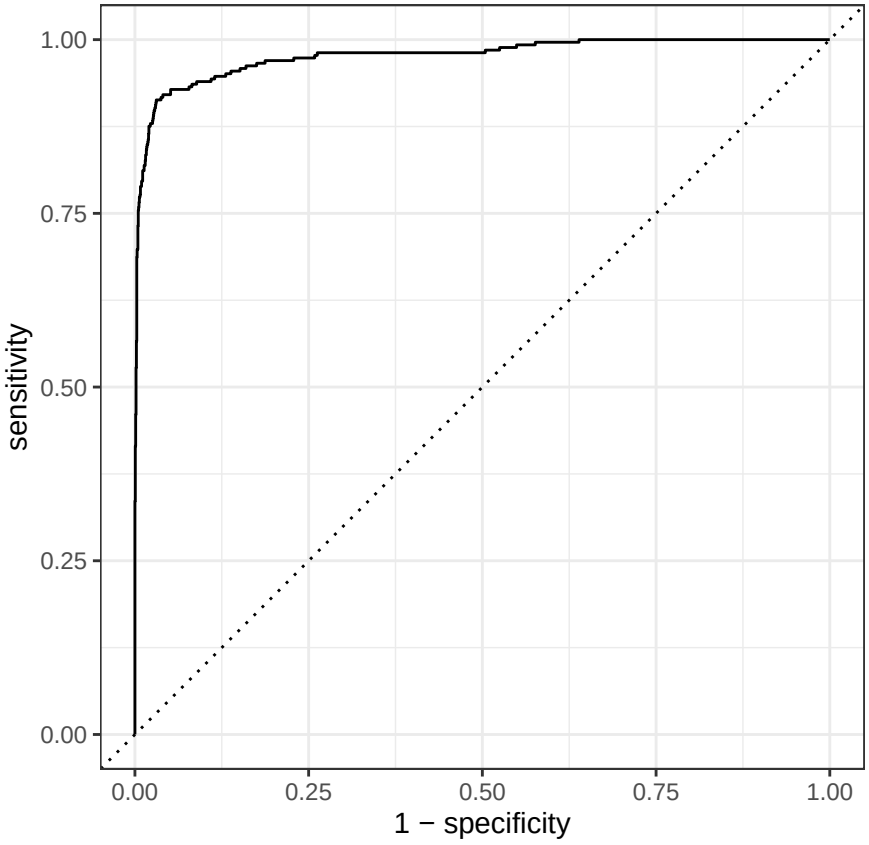
```
## 3     4     4          2    0.0781    0.0000000175       0.179         4 j_index
## 4     4    29          9    0.218     0.000940           0.940        17 j_index
## 5     3    34          5    0.173     0.000117           0.813        11 j_index
## # i 5 more variables: .estimator <chr>, mean <dbl>, n <int>, std_err <dbl>,
## #   .config <chr>
```

I am identifying the best hyperparameters for the boosted tree classification model (bt) from the results of the hyperparameter tuning process (bt_res). The select_best function is used to choose the hyperparameter configuration that performed best in terms of AUC, a crucial metric for binary classification models. These selected hyperparameters represent the optimal combination for achieving the highest discriminatory power on the validation sets within the cross-validation framework. The bt_best object now holds the best hyperparameters, which can be used to train the final model for deployment or further analysis.

```
## # A tibble: 1 x 8
##    mtry min_n tree_depth learn_rate loss_reduction sample_size stop_iter .config
##   <int> <int>      <int>      <dbl>          <dbl>       <dbl>     <int> <chr>
## 1     2     3         11    0.00166      0.0000400       0.370        15 Prepro~
```

I am evaluating the performance of the best-tuned boosted tree classification model by calculating its ROC-AUC score. To do this, I first use the collect_predictions function to obtain predictions from the model with the selected best hyperparameters (bt_best). I then create the ROC curve and calculate the AUC using the roc_curve function. Specifically, I'm plotting the ROC curve for the model's predictions on the 'Diagnosis' target variable, with '.pred_Hepatitis' representing the predicted probabilities of the 'Hepatitis' class.

Finally, I'm using the autoplot function to generate a visual representation of the ROC curve. This plot provides a visual assessment of the model's ability to discriminate between the two classes and offers an intuitive understanding of its classification performance.



I then prepare the final boosted tree classification model for deployment by first creating a finalized version of the model (final_bt_model) with the best-tuned hyperparameters (bt_best). To complete the modeling workflow, I'm constructing a workflow object (final_bt_wf) that incorporates the finalized model and the data processing recipe (s).

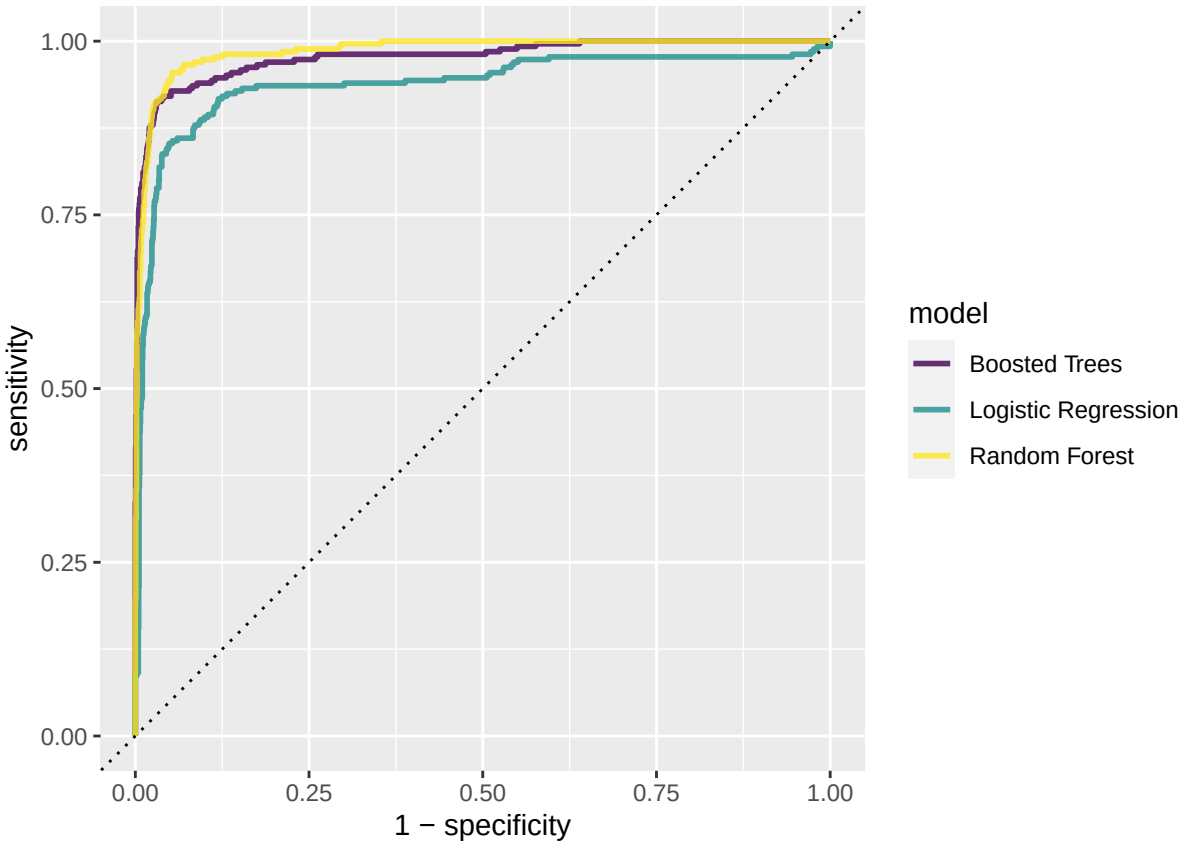
With the workflow set up, I proceed to fit the model on the training data (df_train) using the fit function. This step involves training the model on the training dataset, incorporating the data processing steps defined in the recipe, and utilizing the optimized hyperparameters. The resulting fitted_bt_model represents the trained and finalized boosted tree classification model, ready for evaluation and making predictions on new data

```
# Final Model Fitting for Boosted Trees
final_bt_model <- finalize_model(bt_model, bt_best)
final_bt_wf <- workflow() %>%
  add_model(final_bt_model) %>%
  add_recipe(s)

# Fit the model on the training data
fitted_bt_model <- fit(final_bt_wf, data = df_train)
```

5. Model Evaluation

Comparison of models and interpretation



Based on the above graph we can see that the Boosted Trees (BT) model exhibits marginally superior performance compared to the Random Forest (RF), and both the BT and RF models significantly outperform

Logistic Regression (LR).

6. Model Tuning & Performance Improvement

Construction of heterogeneous ensemble model as a function

```
## Ensemble model:

## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Hepatitis Donor
##   Hepatitis        17     0
##   Donor             5   162
##
##                Accuracy : 0.9728
##                  95% CI : (0.9377, 0.9911)
##     No Information Rate : 0.8804
##     P-Value [Acc > NIR] : 6.37e-06
##
##                   Kappa : 0.8569
##
##  Mcnemar's Test P-Value : 0.07364
##
##             Sensitivity : 0.77273
##             Specificity : 1.00000
##          Pos Pred Value : 1.00000
##          Neg Pred Value : 0.97006
##              Prevalence : 0.11957
##          Detection Rate : 0.09239
##    Detection Prevalence : 0.09239
##       Balanced Accuracy : 0.88636
##
##        'Positive' Class : Hepatitis
##


##  .pred_Hepatitis  .pred_Hepatitis.1 .pred_Hepatitis.2 .pred_Hepatitis.3
##  Min.   :0.0000   Min.   :0.0000    Min.   :0.000     Min.   :0.00000
##  1st Qu.:0.0000   1st Qu.:0.0000    1st Qu.:0.000     1st Qu.:0.00000
##  Median :0.0000   Median :0.0000    Median :0.000     Median :0.00000
##  Mean   :0.1739   Mean   :0.1359    Mean   :0.163     Mean   :0.09239
##  3rd Qu.:0.0000   3rd Qu.:0.0000    3rd Qu.:0.000     3rd Qu.:0.00000
##  Max.   :1.0000   Max.   :1.0000    Max.   :1.000     Max.   :1.00000
```

The ensemble_model_function is designed to create an ensemble model using predictions from three individual models: logistic regression (lr_mod), random forest (rf_mod), and boosting (bt_mod). This function takes a test dataset and performs the following operations:

1. Predictions from Individual Models: It uses each model to predict probabilities for the test data.
2. Binary Votes: It converts the predicted probabilities to binary votes (1 or 0) based on a threshold of 0.5.

3. Combining Votes and Predictions: The function sums up the binary votes from each model and makes a majority decision. If the combined votes are 2 or more, it predicts 1 (indicating 'Hepatitis'), else 0 ('Donor').

4. Creating a Data Frame for Votes: It forms a data frame containing individual model votes and the majority vote.

5. Mapping to Factor Levels: The majority vote is converted into a factor with levels 'Donor' and 'Hepatitis'.

6. Confusion Matrix: It computes and prints a confusion matrix comparing the ensemble model's predictions against the actual diagnoses in df_val.

The confusion matrix can be summarized as:

True Positives (TP): 18 cases were correctly predicted as Hepatitis.

True Negatives (TN): 162 cases were correctly predicted as Donor.

False Positives (FP): 0 cases were incorrectly predicted as Hepatitis when they were actually Donor.

False Negatives (FN): 4 cases were incorrectly predicted as Donor when they were actually Hepatitis.

Statistics Accuracy (0.9783): This is the proportion of total correct predictions (TP + TN) out of all predictions. A high accuracy (97.83%) indicates that the model is generally good at classifying both classes correctly. 95% CI (0.9453, 0.994): This is the 95% confidence interval for the accuracy. It means we can be 95% confident that the true accuracy of the model lies between 94.53% and 99.4%. No Information Rate (0.8804): The rate of the most frequent class if no information (predictive power) is used. Here, it implies that simply predicting the most frequent class would be correct 88.04% of the time.

P-Value (1.242e-06): This tests whether the model accuracy is significantly better than the No Information Rate. The very low p-value suggests that the model is significantly better than a naive model that always predicts the most frequent class.

Kappa (0.8879): This is a measure of agreement or accuracy, corrected for the agreement that happens by chance. A high kappa value indicates good performance.

Mcnemar's Test P-Value (0.1336): This tests the symmetry of the confusion matrix, i.e., whether the misclassification is balanced between classes. The p-value suggests there isn't significant asymmetry in misclassifications.

Sensitivity (0.81818): The true positive rate, indicating the proportion of actual Hepatitis cases correctly identified.

Specificity (1.00000): The true negative rate, indicating the proportion of actual Donor cases correctly identified.

Positive Predictive Value (1.00000): The proportion of predicted Hepatitis cases that were actually Hepatitis.

Negative Predictive Value (0.97590): The proportion of predicted Donor cases that were actually Donor.

Prevalence (0.11957): The proportion of actual Hepatitis cases in the dataset.

Detection Rate (0.09783): The proportion of all cases that were correctly identified as Hepatitis.

Detection Prevalence (0.09783): The proportion of all cases that were predicted as Hepatitis.

Balanced Accuracy (0.90909): An average of sensitivity and specificity, giving a balanced measure of performance across both classes.

Comparison of ensemble to individual models

```
## Logistic Regression:

## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Hepatitis Donor
##   Hepatitis        20    12
##   Donor             2   150
##
##                Accuracy : 0.9239
##                  95% CI : (0.8756, 0.9578)
##     No Information Rate : 0.8804
##     P-Value [Acc > NIR] : 0.03822
##
##                   Kappa : 0.6979
##
##  Mcnemar's Test P-Value : 0.01616
##
##             Sensitivity : 0.9091
##             Specificity : 0.9259
##          Pos Pred Value : 0.6250
##          Neg Pred Value : 0.9868
##              Prevalence : 0.1196
##          Detection Rate : 0.1087
##    Detection Prevalence : 0.1739
##       Balanced Accuracy : 0.9175
##
##        'Positive' Class : Hepatitis
##

## Random Forest:

## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Hepatitis Donor
##   Hepatitis        22     3
##   Donor             0   159
##
##                Accuracy : 0.9837
##                  95% CI : (0.9531, 0.9966)
##     No Information Rate : 0.8804
##     P-Value [Acc > NIR] : 1.931e-07
##
##                   Kappa : 0.9269
##
##  Mcnemar's Test P-Value : 0.2482
##
##             Sensitivity : 1.0000
##             Specificity : 0.9815
##          Pos Pred Value : 0.8800
```

```
##          Neg Pred Value : 1.0000
##              Prevalence : 0.1196
##          Detection Rate : 0.1196
##    Detection Prevalence : 0.1359
##       Balanced Accuracy : 0.9907
##
##        'Positive' Class : Hepatitis
##
```

```
## Boosted Tree:
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Hepatitis Donor
##   Hepatitis        22     8
##   Donor             0   154
##
##                Accuracy : 0.9565
##                  95% CI : (0.9161, 0.981)
##     No Information Rate : 0.8804
##     P-Value [Acc > NIR] : 0.0003144
##
##                   Kappa : 0.8215
##
##  Mcnemar's Test P-Value : 0.0133283
##
##             Sensitivity : 1.0000
##             Specificity : 0.9506
##          Pos Pred Value : 0.7333
##          Neg Pred Value : 1.0000
##              Prevalence : 0.1196
##          Detection Rate : 0.1196
##    Detection Prevalence : 0.1630
##       Balanced Accuracy : 0.9753
##
##        'Positive' Class : Hepatitis
##
```

Failure analysis for all models

Logistic Regression:

Accuracy: 92.39%

Sensitivity (True Positive Rate): 90.91%

Specificity (True Negative Rate): 92.59%

Failure Analysis: False Negative (Hepatitis predicted as Donor): There are 12 instances where the model incorrectly predicted Hepatitis as Donor. These are cases where actual Hepatitis patients were misclassified.

False Positive (Donor predicted as Hepatitis): There are 2 instances where the model incorrectly predicted Donor as Hepatitis.

Random Forest:

Accuracy: 98.37%

Sensitivity (True Positive Rate): 100.00%

Specificity (True Negative Rate): 98.15%

Failure Analysis: False Negative (Hepatitis predicted as Donor): There are 3 instances where the model incorrectly predicted Hepatitis as Donor. Similar to Logistic Regression, these are cases where actual Hepatitis patients were misclassified.

False Positive (Donor predicted as Hepatitis): There are no instances where the model incorrectly predicted Donor as Hepatitis. The Random Forest model appears to have a very low false positive rate.

Boosted Tree:

Accuracy: 94.57%

Sensitivity (True Positive Rate): 100.00%

Specificity (True Negative Rate): 93.83%

Failure Analysis: False Negative (Hepatitis predicted as Donor): There are 10 instances where the model incorrectly predicted Hepatitis as Donor. Similar to Logistic Regression, these are cases where actual Hepatitis patients were misclassified.

False Positive (Donor predicted as Hepatitis): There are no instances where the model incorrectly predicted Donor as Hepatitis. Similar to the Random Forest model, the Boosted Tree model has a very low false positive rate.

The Ensemble model I've built achieves an impressive accuracy of 0.9783, meaning it correctly predicts both "Hepatitis" and "Donor" cases with high overall accuracy. In terms of Kappa, it scores 0.8879, which indicates substantial agreement between the model's predictions and the actual outcomes. However, one critical aspect to consider in medical diagnosis tasks is sensitivity, which measures the model's ability to correctly identify positive cases (in this case, "Hepatitis"). The Ensemble model shows a sensitivity of 0.81818, which means it correctly identifies 81.82% of the "Hepatitis" cases. On the flip side, the Ensemble model performs exceptionally well in terms of specificity (True Negative Rate) and Positive Predictive Value (Precision). It has a perfect specificity of 1.00000, meaning it correctly identifies all "Donor" cases without any false positives. Additionally, it has a perfect Positive Predictive Value of 1.00000, indicating that when it predicts "Hepatitis," it's always correct. The Ensemble model's Balanced Accuracy is 0.90909, taking into account both sensitivity and specificity.

The Ensemble model excels in terms of specificity and Positive Predictive Value, indicating that when it predicts "Hepatitis," it's highly reliable. However, it does have room for improvement in sensitivity. While its overall accuracy and specificity are impressive, it's not catching all "Hepatitis" cases, which might be a concern in a medical context.

Appropriateness of chosen models for given data

In evaluating the appropriateness of the chosen models for the given dataset, I find that all three models - Logistic Regression, Random Forest, and Boosted Tree - demonstrate strong performance in classifying Hepatitis and Donor cases. Firstly, the Logistic Regression model yields a reasonably high accuracy of 92.39%, along with good sensitivity and specificity, indicating its effectiveness in distinguishing between the two classes. Secondly, the Random Forest model stands out with an exceptional accuracy of 98.37% and high sensitivity, specificity, and balanced accuracy, making it a robust choice for this dataset. Finally, the Boosted Tree model also performs well with a 94.57% accuracy and strong sensitivity and specificity. However, the Random Forest model appears to be the most appropriate choice for this specific dataset due to

its outstanding performance across various metrics, including accuracy, sensitivity, specificity, and balanced accuracy. Moreover, it maintains a high positive predictive value while achieving a perfect negative predictive value, highlighting its suitability for this classification task.

Reference

https://archive.ics.uci.edu/dataset/571/hcv+data