Hepatisis data analysis

1 Introduction. Data description

1.1 Problem description

We chose data: B) Medical diagnostics: Hepatitis Data Set. http://archive.ics.uci.edu/ml/datasets/Hepatitis (http://archive.ics.uci.edu/ml/datasets/Hepatitis)

The main aim of our project is to analyze the dataset with clinical trial results of people with hepatitis and try to evaluate death risk. Hepatitis is a serious disease, inflammation of the liver from any cause, and it can lead to the death of a person. We want to find better diagnostic methods that should help to determine the risk of death due to hepatitis.

1.2 Data characteristics

Our data set consist of 155 observations and 20 columns. The features are the following:

1. **Class** - a factor at two levels, which we want to predict (1 - patient dies, 2 - patient lives).
2. **Age** - age of the patients in years (from 20 to 80 years).
3. **Sex** - gender of patient, a factor at two levels coded by 1 (male) and 2 (female).
4. **Steroid** - steroid treatment, a factor at two levels coded by 1 (yes) and 2 (no).
5. **Antivirals** - antivirals medication, a factor at two levels 1 (yes) and 2 (no).
6. **Fatigue** - fatigue is a frequent and disabling symptom reported by patients with chronic hepatitis, a factor at two levels 1 (yes) and 2 (no).
7. **Malaise** - malaise one of the symptoms of hepatitis, a factor at two levels 1 (yes) and 2 (no).
8. **Anorexia** - anorexia, loss of appetite, a factor at two levels 1 (yes) and 2 (no).
9. **LiverBig** - the size of liver increased or fatty, a factor at two levels 1 (yes) and 2 (no).
10. **LiverFirm** - the liver is firm, a factor at two levels 1 (yes) and 2 (no).
11. **SpleenPalpable** - splenomegaly is an enlargement of the spleen, a factor at two levels 1 (yes) and 2 (no).
12. **Spiders** - enlarged blood vessels that resemble little spiders,a factor at two levels 1 (yes) and 2 (no).
13. **Ascites** - ascites is the presence of excess fluid in the peritoneal cavity, a factor at two levels 1 (yes) and 2 (no).
14. **Varices** - varicose veins are a medical condition in which superficial veins become enlarged and twisted, a factor at two levels 1 (yes) and 2 (no).
15. **Bilirubin** - bilirubin is a substance made when the body breaks down old red blood cells.
16. **AlkPhosphate** - alkaline phosphatase is an enzyme made in liver cells and bile ducts, a discrete valued feature reveals level alkaline phosphatase measured in IU/L, where UI - international unit. A 2013 research review showed that the normal range for a serum ALP level in healthy adults is 20 to 140 IU/L.
17. **Sgot** - a glutamic-oxaloacetic transaminase (SGOT) test measures the levels of the enzyme AST in the blood to assess liver health. A discrete valued feature measured in units per liter of serum. If the results of your SGOT test are high, that means one of the organs or muscles containing the enzyme could be damaged. The normal range of an SGOT test is generally between 8 and 45 units per liter of serum.
18. **Albumin** - albumin is a family of globular proteins, the most common of which are the serum albumins. Low albumin levels can indicate a disorder of the liver or kidneys.
19. **Protime** - a discrete valued feature. How long it takes blood to form a clot in sec. It shows how bad liver works.
20. **Histology** - histology is the branch of biology that studies the microscopic anatomy of biological tissues. A factor at two levels 1 (yes) and 2 (no).

```
df <- read.table("hepatitis.data", sep = ",")
colnames(df) <- c("Class","Age","Sex","Steroid","Antivirals","Fatigue","Malaise","Anorexia","LiverBig","LiverFirm","SpleenPa
lpable","Spiders","Ascites","Varices","Bilirubin","AlkPhosphate","Sgot","Albumin","Protime","Histology") # nolint
```

Below we present the first rows of our data set. As can be seen, in addition to the fact that we will have to change the encoding from 1-2 to 0-1, we will also have to deal with missing values. In our dataset, they are marked as “?”.

```
head(df[1:10]) %>%
    flextable() %>%
    theme_box() %>%
    autofit()
```

Class	Age	Sex	Steroid	Antivirals	Fatigue	Malaise	Anorexia	LiverBig	LiverFirm
2	30	2	1	2	2	2	2	1	2
2	50	1	1	2	1	2	2	1	2
2	78	1	2	2	1	2	2	2	2
2	31	1	?	1	2	2	2	2	2
2	34	1	2	2	2	2	2	2	2

Class	Age	Sex	Steroid	Antivirals	Fatigue	Malaise	Anorexia	LiverBig	LiverFirm
2	34	1	2	2	2	2	2	2	2

```
head(df[11:20]) %>%
    flextable() %>%
    theme_box() %>%
    autofit()
```

SpleenPalpable	Spiders	Ascites	Varices	Bilirubin	AlkPhosphate	Sgot	Albumin	Protime	Histology
2	2	2	2	1.00	85	18	4.0	?	1
2	2	2	2	0.90	135	42	3.5	?	1
2	2	2	2	0.70	96	32	4.0	?	1
2	2	2	2	0.70	46	52	4.0	80	1
2	2	2	2	1.00	?	200	4.0	?	1
2	2	2	2	0.90	95	28	4.0	75	1

2 Preparing data

2.1 Data types

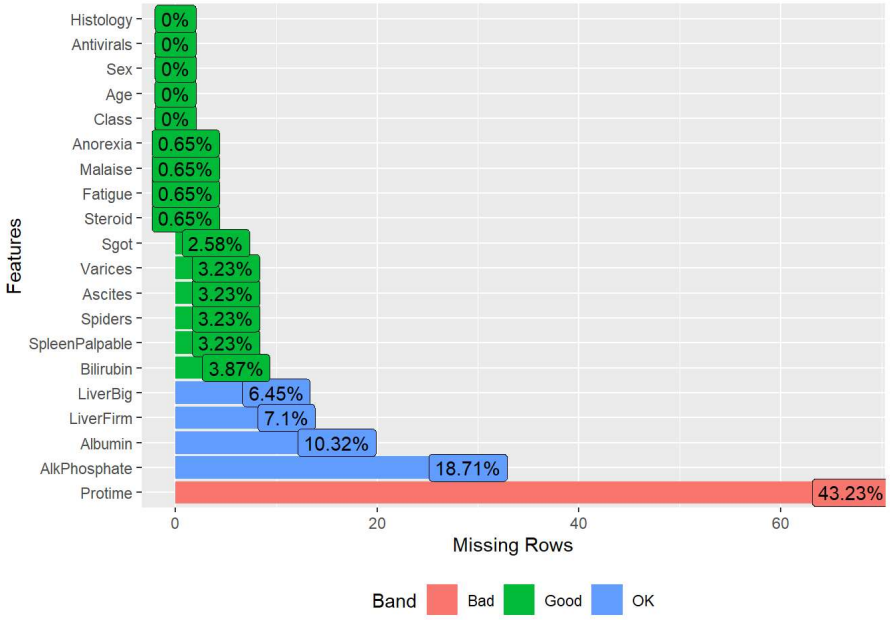
We initially had the wrong data types, so we changed them to the correct ones.

```
df[df == "?"] <- NA

df <- mutate_all(df, function(x) as.numeric(as.character(x)))
categorical <- c(1, 3:14, 20)
df[, categorical] <- replace(df[, categorical], df[, categorical] == 2, 0)
df[, categorical] <-  lapply(df[, categorical], as.factor)
```

2.2 Missing values

There are 5 features without missing values, 10 with a small amount, 4 with an allowed number, and 1 (Protime) with almost half of the missing values.

```
plot_missing(df)
```



In all, we have 75 rows with missing values, which have a total of 167 missing values. So we can not remove these rows.

```
sum(rowSums(is.na(df)) != 0)
```

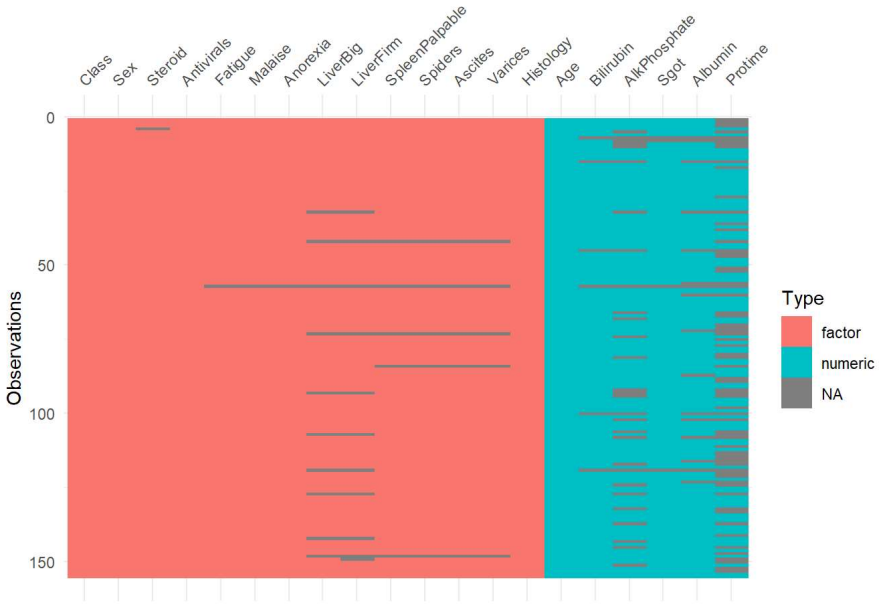
```
## [1] 75
```

```
sum(is.na(df))
```

```
## [1] 167
```

We can see that the majority of variables have missing values only in several attributes, but there exist objects with several non-missing characteristics.

```
vis_dat(df)
```



2.3 Add missing values

We used several methods to impute missing values: knnImpute (df1), bagImpute (df2), medianImpute (df3), and function with different methods for different attributes (predictive mean matching for numeric data and logistic regression imputation for binary data where a factor is with 2 levels)

We need all numerical values to use the first 3 methods.

```
set.seed(123)
df.new <- mutate_all(df, function(x) as.numeric(as.character(x)))
data_transform <- preProcess(df.new, method = "knnImpute")
data_transform2 <- preProcess(df.new, method = "bagImpute")
data_transform3 <- preProcess(df.new, method = "medianImpute")

df1 <- predict(data_transform, df.new)
df2 <- predict(data_transform2, df.new)
df3 <- predict(data_transform3, df.new)
```

Since knnImpute returns normalized data, we need to return to the initial form. Also, we have to round values for intiger and categorical variables.

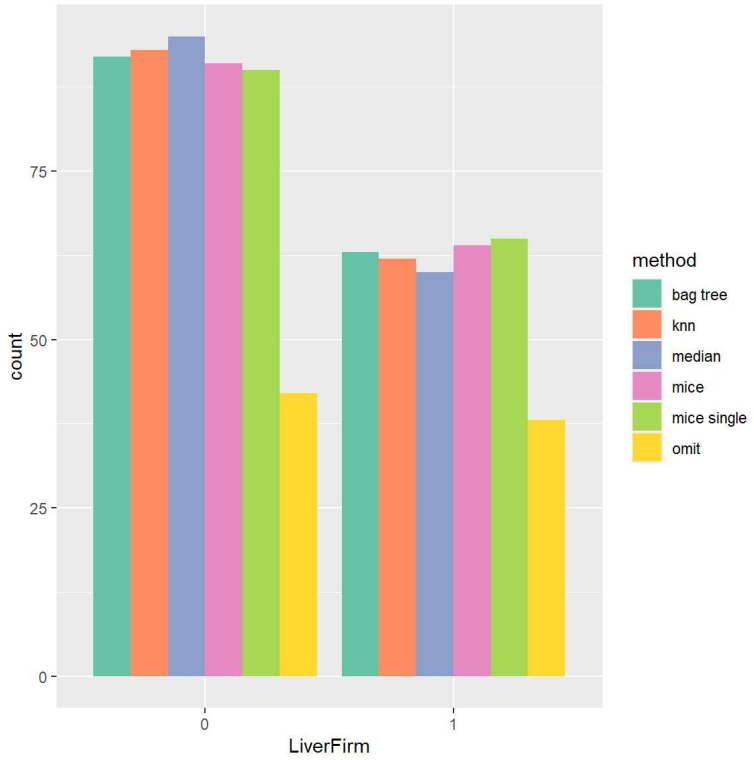
```
df1.standarized <- df1
df2[c(4:14, 16:17, 19)] <- round(df2[c(4:14, 16:17, 19)])
df3[c(4:14, 16:17, 19)] <- round(df3[c(4:14, 16:17, 19)])
df1 <- unstandarize(df1)
```

In this function we impute missing values 1 time (df4), and 5 times in order to get better results (df5)

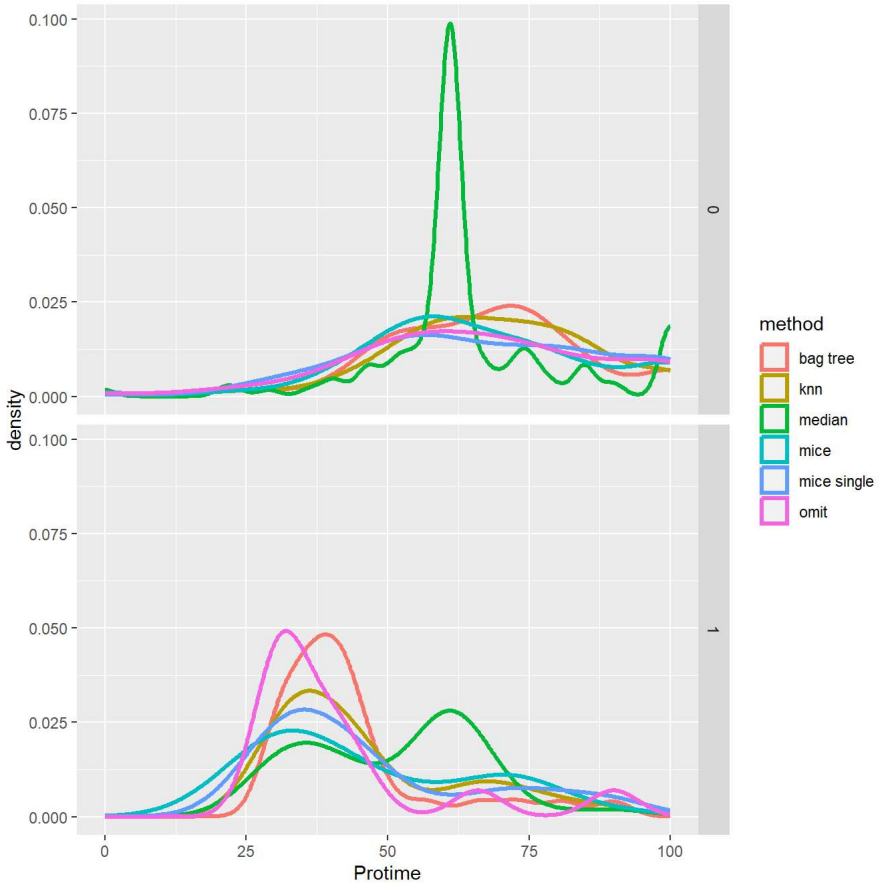
```
methods = c(" ", " ", " ", "logreg", " ", "logreg", "logreg", "logreg", "logreg", "logreg", "logreg", "logreg", "logreg", "l
ogreg", "pmm", "pmm", "pmm", "pmm", "pmm", " ")
imp_single <- mice(df, m = 1, method = methods) # Impute missing values
df4 <- complete(imp_single)         # Store imputed data

imp_multi <- mice(df, method = methods)  # Impute missing values multiple times
df5 <- complete(imp_multi, 1)
```

Now, let's check if the distribution has changed.

```
ggplot(na.omit(df_all), aes(x = LiverFirm, fill = method)) +
  scale_fill_brewer(palette = "Set2") +
  geom_bar(position = "dodge")
```




```
ggplot(na.omit(df_all), aes(x = Protime, color = method)) +
  scale_fill_brewer(palette = "Set2") +
  geom_density(size = 1.1) +
  facet_grid(Class ~ .)
```



The median method works badly for the attribute with many missing values (we have such a feature). We can see that the distributions of both LiverFirm and Protime have changed a lot compared to the original. Median method added all values to the class with bigger number of records for categorical attributes. So we can not use it. For the rest, results are similar, but we can also skip mice single because mice is just an average value of 5 single using of this (so results must be better). So, we will compare three methods of imputing missing values: knn(df1), bag tree(df2), and mice(df5).

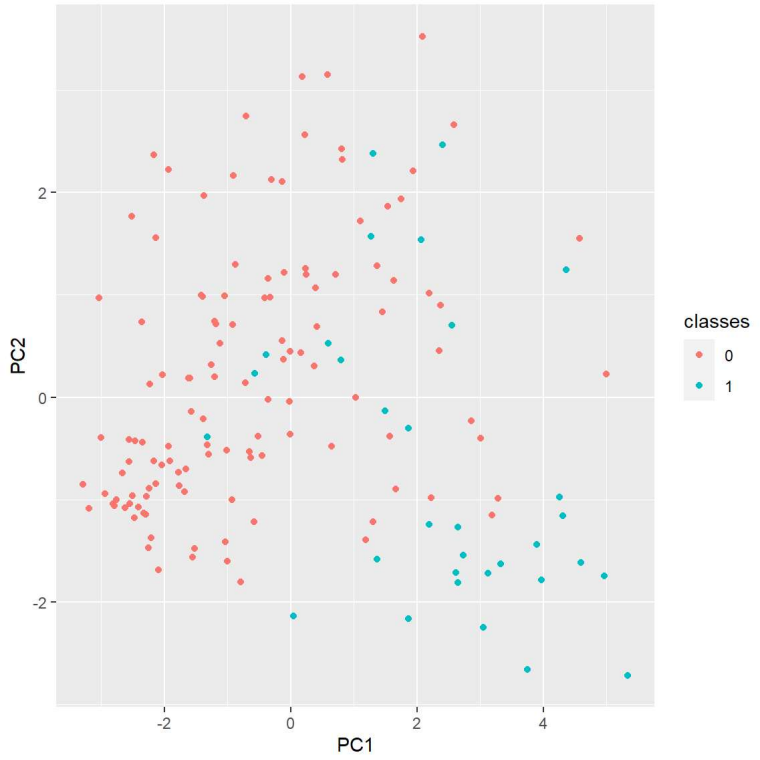
3 EDA

3.1 Data visualization in 2D

We visualize data in 2D using Principal component analysis (PCA).

In a 2D plot, it's difficult to split our data into 2 classes, because they are overlapping, but maybe in more space, the situation is better.

```
df1.num <- to_numeric1(df1)
df1.num.pca <- prcomp(df1.num[,2:20], center=T, scale=T)
pca <- data.frame(PC1 = df1.num.pca$x[,1], PC2 = df1.num.pca$x[,2], classes = as.factor(df1.num$Class))

ggplot(data = pca, aes(x = PC1, y = PC2)) +
    geom_point(aes(color = classes))
```



3.2 Summary

Below we present basic statistics for continuous variables for various types of missing data substitutions. The statics are not significantly different. However, density functions must also be taken into account. They will tell us more about distribution. We have already presnted the density of "Protime" to justify the rejection of this method. Other distributions will be analyzed later in the report.

Method	Age							Bilirubin							AlkPhosphate					
	min	q1	median	mean	q3	max	std	min	q1	median	mean	q3	max	std	min	q1	median	mean	q3	n
bag tree	7	32	39	41.2	50	78	12.57	0.3	0.80	1	1.41	1.5	8	1.19	26	75.00	85	104.55	131.50	
knn	7	32	39	41.2	50	78	12.57	0.3	0.80	1	1.42	1.5	8	1.19	26	75.00	85	103.30	126.00	
mice	7	32	39	41.2	50	78	12.57	0.3	0.75	1	1.41	1.5	8	1.19	26	73.00	85	104.65	126.50	
omit	7	32	39	41.2	50	78	12.57	0.3	0.70	1	1.43	1.5	8	1.21	26	74.25	85	105.33	132.25	

Method	Sgot							Albumin							Protime					
	min	q1	median	mean	q3	max	std	min	q1	median	mean	q3	max	std	min	q1	median	mean	q3	n
bag tree	14	32.5	58	85.73	100.5	648	88.58	2.1	3.5	4	3.82	4.2	6.4	0.62	0	47.5	63	62.15	75.00	
knn	14	32.5	58	85.57	99.0	648	88.55	2.1	3.5	4	3.82	4.2	6.4	0.62	0	50.5	64	63.89	78.50	
mice	14	31.0	55	84.55	99.0	648	88.88	2.1	3.5	4	3.82	4.2	6.4	0.64	0	47.0	60	61.61	74.50	
omit	14	31.5	58	85.89	100.5	648	89.65	2.1	3.4	4	3.82	4.2	6.4	0.65	0	46.0	61	61.85	76.25	

3.3 Barplots

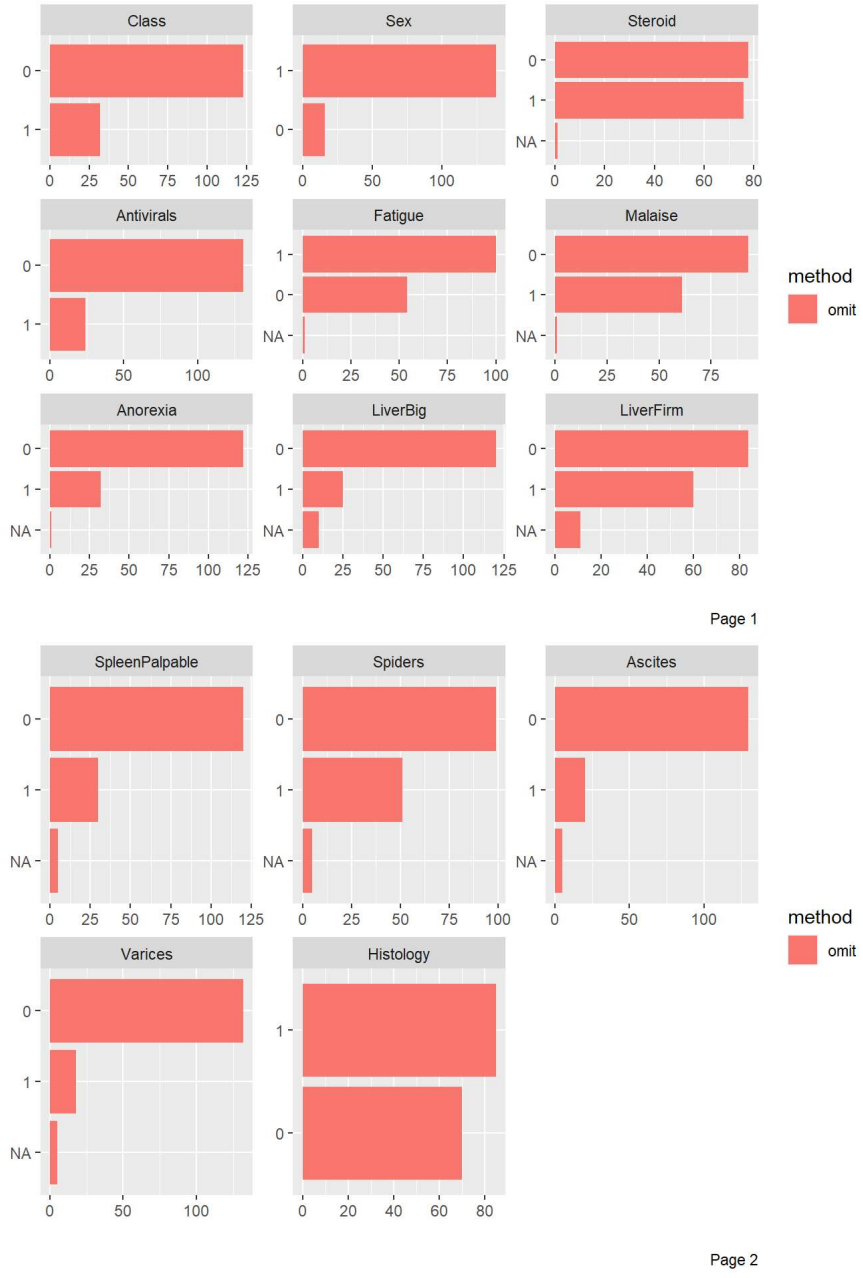
First, we will compare the distribution of binary variables. We will only show the results for the initial data frame, without imputed missing values, because the distributions do not change across different imputation methods. A comparison of the different imputation methods will be available in the supplementary materials.

Insights:

- We have imbalance problem in dependent variable;

- We have imbalance classes in almost all features;
- Also, there are several features where imbalance is not so large: Fatigue, Malaise, LiverFirm, Spiders;
- We have several balanced attributes: Steroid, Histology.

```
plot_bar(df, by = "method", by_position = "dodge")
```

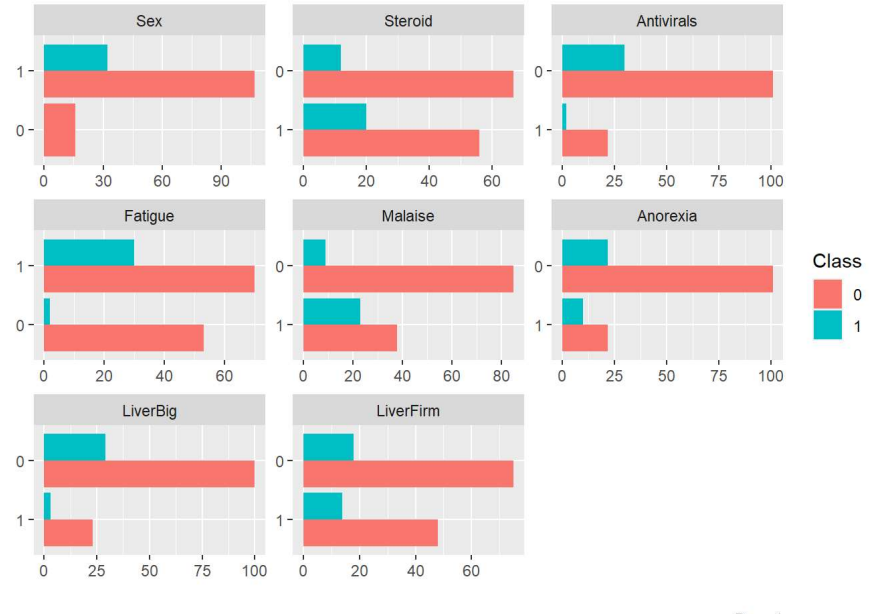


Now we will try to assess the ability to separate objects from different classes by comparing the distribution of the target class for diffrent values of binary attributes. We will use only one data set (with missing values imputed using knn method) as the results for other data sets are similar.

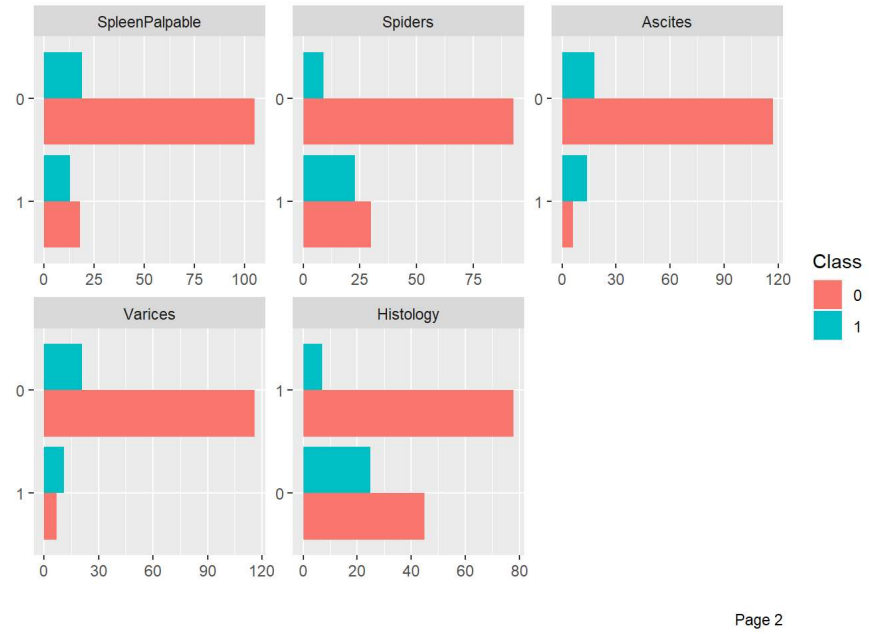
Insights:

- We have more men than women in the dataset. Moreover no women died. This may suggest that men are more likely to die from hepatitis.
- Almost all of the people who died had symptoms of fatigue and did not take antiviral medications.
- Surprisingly, no enlarged liver was found in those who died.
- “Spiders” attribute might be important. The majority of people who died had spiders.

```
df1[, categorical] <-  lapply(df1[, categorical], as.factor)

plot_bar(df1[-21], by = "Class", by_position = "dodge")
```



Page 1



Page 2

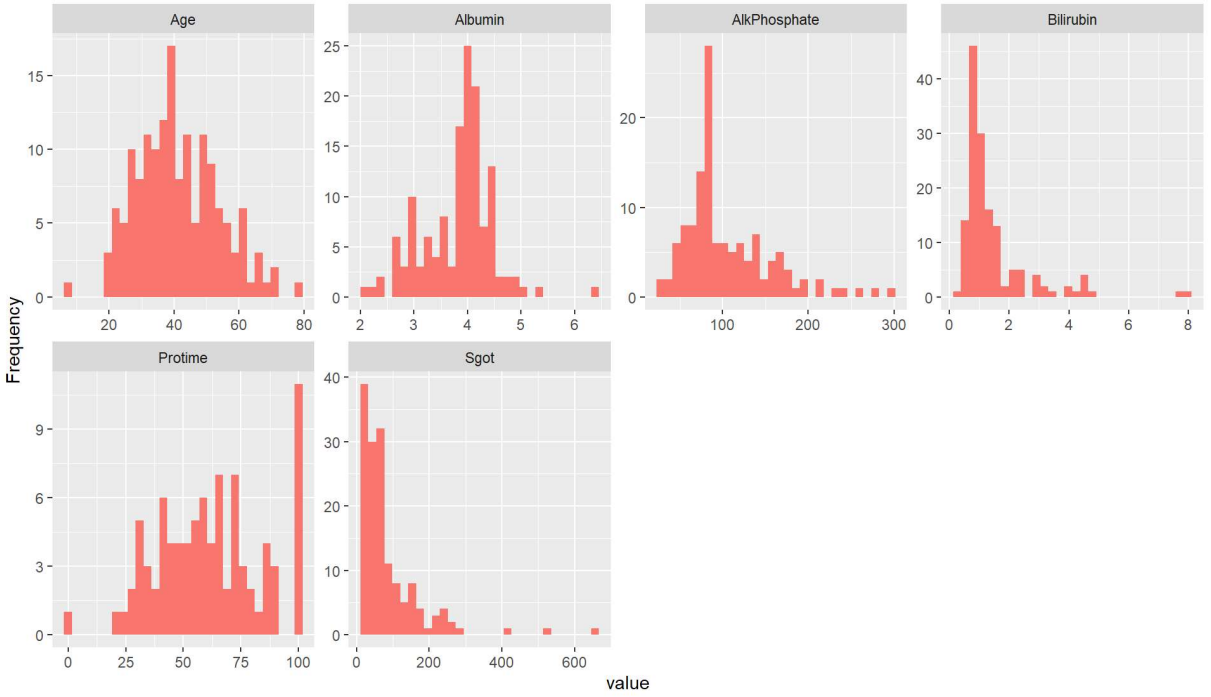
3.4 Histograms

We have only 6 continuous attributes: Age, Bilirubin, AlkPhosphate, Sgot, Albumin, Protime.

Insights:

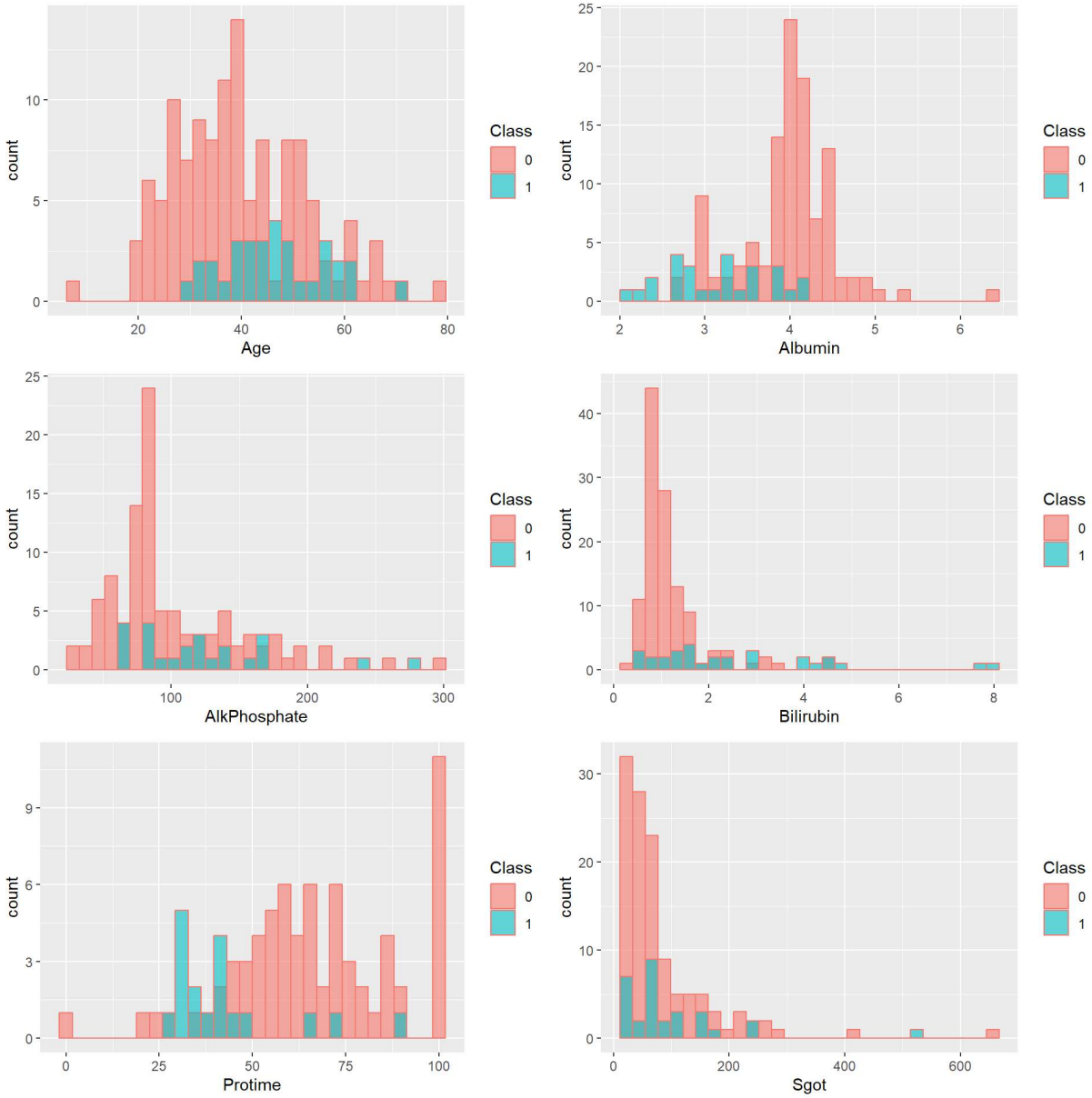
- All have unimodal distribution;
- Albumin has a left-skewed distribution and AlkPhosphate, Bilirubin, Sgot have a right-skewed distribution. Also, Age and Protime have an almost symmetric distribution;
- The most common value for Age is about 40, Albumin - 4, AlkPhosphate - 75, Bilirubin - 0.7, Protime - 100, and Sgot - 0;
- Age looks like Gamma distribution, Albumin - Beta, AlkPhosphate - Log Normal, Bilirubin - Log Normal, Protime - Beta, and Sgot - Exponential.

```
plot_histogram(df, geom_histogram_args = list("fill" = "#f8766d"))
```



Insights:

- There is a big difference in distributions of different classes, because the number of values from class 1 is significantly less;
- But, maybe, it will be difficult to separate objects from different classes, because they are overlapping. Only Albumin and Protime are a bit of difference. There are no values from class 1 with values of Albumin higher than 4.3 and a few values from class 1 with values of Protime higher than 50.

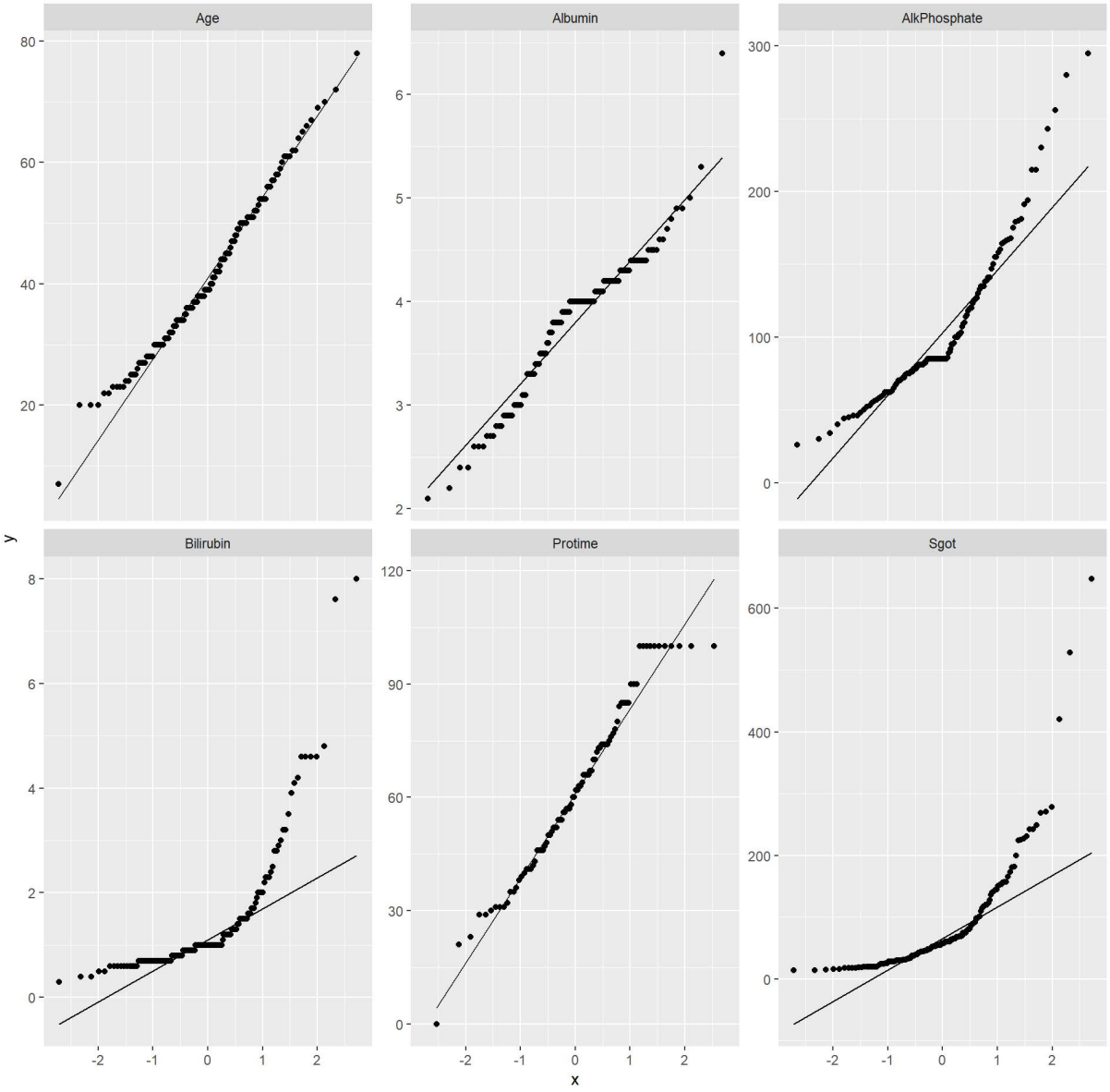


In the additional file, we can see density plots for different methods of imputing missing values. The results are very similar.

3.5 Q-Q plot

We can see that q-q plot for Age, Albumin and Protime look no so bad, maybe this attributes have normal distribution.

```
plot_qq(df)
```



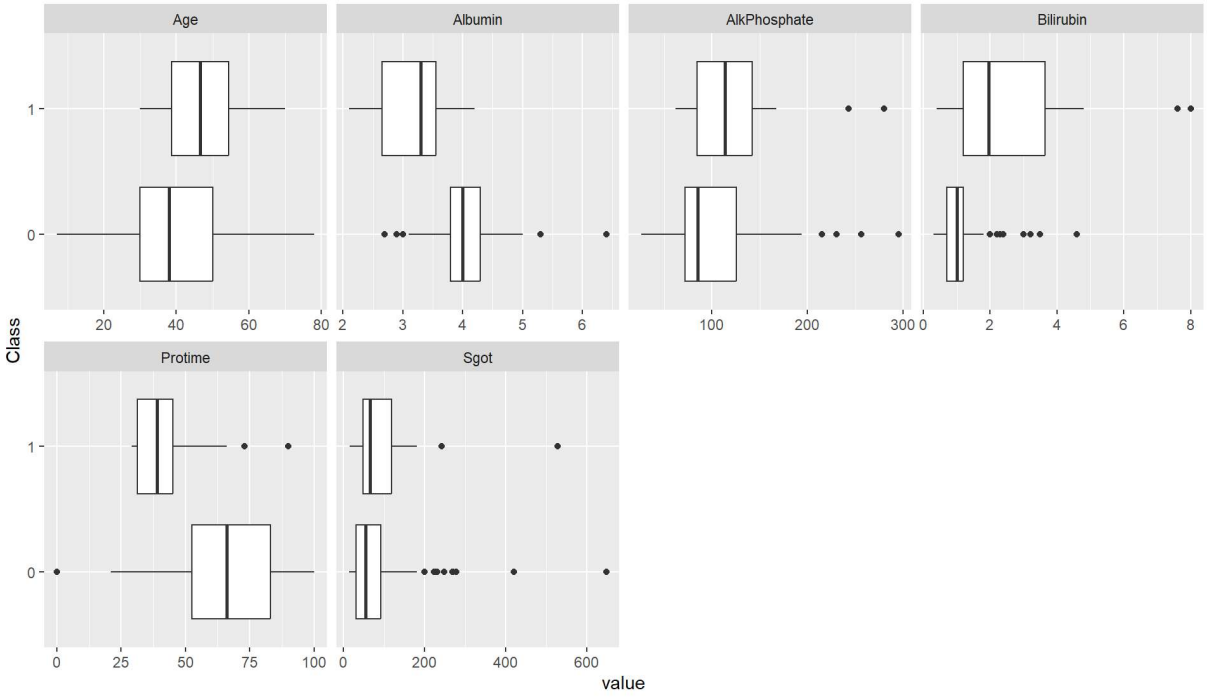
In the additional file, we can see Q=Q plots for different methods of imputing missing values. The results are very similar.

3.6 Boxplots

Insights:

- There are no outliers for Age, 3 for Protime, and a lot for other features.
- It's better to use Albumin, Bilirubin, and Protime to classify because they have larger differences.

```
plot_boxplot(df, by = "Class")
```



In the additional file, we can see box plots for different methods of imputing missing values. The results are very similar.

3.7 Correlation

First we will analyze the Pearson correlation between continuous variables. To calculate correlation, we must use data without missing values, so we use data with imputing missing values using knn. In the additional file, we can see a correlation matrix for other methods of imputing missing values. The results are very similar.

According to the correlation matrix we have low correlation for all pair of features (< 0.41). Furthermore, as can be seen on the scatter plots, combinations of Albumin and Protime with other attributes seem to be good at distinguishing between classes.

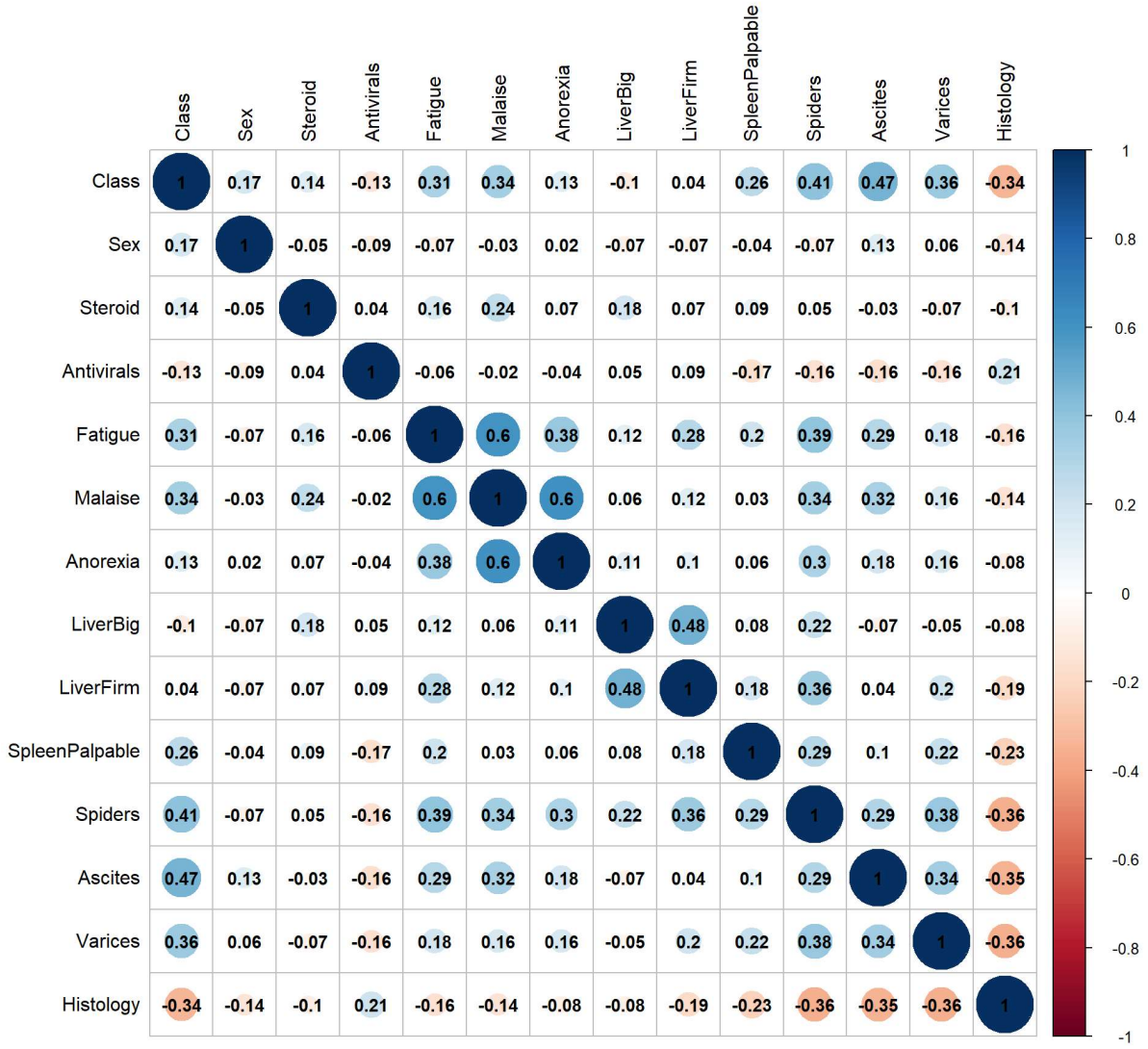
We can also observe two outliers (people with high level of Bilirubin).

```
ggpairs(df1, columns = c(2,15:19), aes(color = Class, alpha = 0.5))
```

When it comes to correlation between binary variables, “Malaise” is correlated to “Fatigue” and “Anorexia”. The target class, on the other hand, is most correlated with “Spiders” and “Ascites”.

```
to_numeric <- function(data) {
  data[,categorical] <- lapply(data[, categorical], as.numeric)
  data[,categorical] <- data[,categorical] - 1
  return(data)
}
```

```
cor_matrix <- cor(to_numeric(df1)[, categorical], method = "pearson")
corrplot(cor_matrix, tl.col = "black", addCoef.col = 1, number.cex = 0.9)
```



3.8 Conclusions

- We need to impute missing values, and use for it 3 different methods;
- Results in EDA for all imputing data are similar;
- We have class imbalance problem;
- Most of categorical attributes also have imbalanced classes;
- We have low correlation for almost all pairs of features (< 0.41);
- Next fields seems to be important: Antivirals, Fatigue, LiverBig, Albumin, Bilirubin, and Protime;
- Next variables seems useless: Steroid, Age, AlkPhosphate, and Sgot.

4 Classification

We have the same values in variable Class in datasets for all imputing methods, so we can use the same inTrain.

```
set.seed(123)
inTrain <- createDataPartition(y=df1$Class, times=1, p=0.75, list=FALSE)

train1 <- df1[inTrain,-21]
test1 <- df1[-inTrain,-21]

train2 <- df2[inTrain,-21]
test2 <- df2[-inTrain,-21]

train3 <- df5[inTrain,-21]
test3 <- df5[-inTrain,-21]
```

4.1 Solving class imbalance problem

To solve class imbalance problem we decided to use Majority Weighted Minority Oversampling Technique. We have the same values in variable Class in datasets for all imputing methods, so we can use the same n_new.

```
# oversampling
n_new <- sum(train1$Class == 0) - sum(train1$Class == 1)
train.num1 <- to_numeric1(train1)
train.num2 <- to_numeric1(train2)
train.num3 <- to_numeric1(train3)

test.num1 <- to_numeric1(test1)
test.num2 <- to_numeric1(test2)
test.num3 <- to_numeric1(test3)

newMWMOTE1 <- mwmote(train.num1, numInstances = n_new)
train.balanced1 <- rbind(train.num1[-21], newMWMOTE1)

newMWMOTE2 <- mwmote(train.num2, numInstances = n_new)
train.balanced2 <- rbind(train.num2[-21], newMWMOTE2)

newMWMOTE3 <- mwmote(train.num3, numInstances = n_new)
train.balanced3 <- rbind(train.num3[-21], newMWMOTE3)
```

Let's check the proportion of the classes in train and test sets. We can see that they are similar. In a balanced set, classes are divided equally.

```
prop.table(table(train1$Class))
```

```
##
##         0         1
## 0.7948718 0.2051282
```

```
prop.table(table(test1$Class))
```

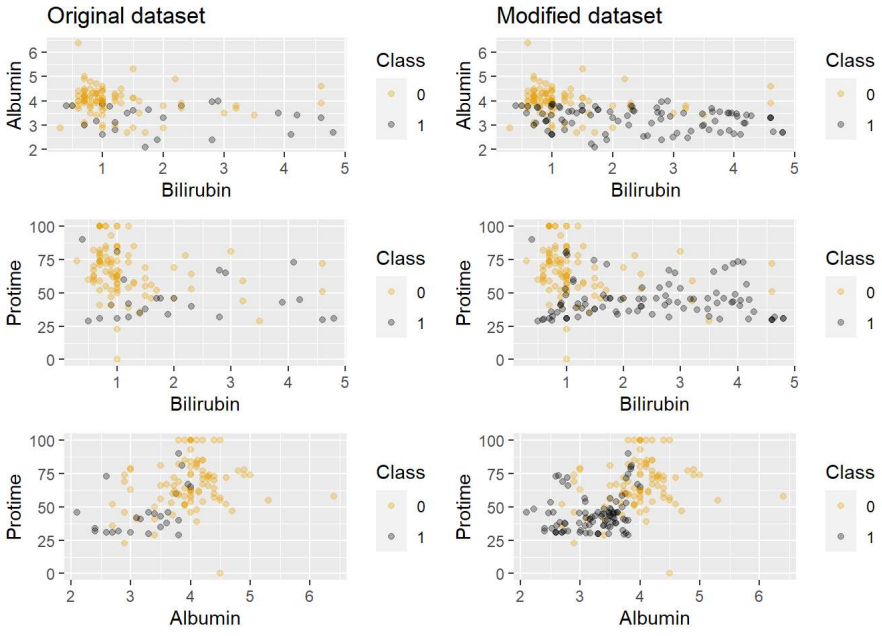
```
##
##         0         1
## 0.7894737 0.2105263
```

```
prop.table(table(train.balanced1$Class))
```

```
##
##   0   1
## 0.5 0.5
```

Below we can check the comparison of values before and after oversampling for three selected variables.

```
train.balanced1.2 <- train.balanced1
train.balanced1.2[, categorical] <-  lapply(train.balanced1.2[, categorical], as.factor)

plotComparison(train1, train.balanced1.2, attrs = c("Bilirubin","Albumin", "Protime"))
```



4.2 Confusion matrix

In our case we have medical data, so we can use:

- Sensitivity/Recall – an estimate of the probability that the test will predict the death, provided that the patient is actually dead.
- Specificity – an estimate of the probability that the test predicts that the patient is alive, provided that he or she is in fact alive.
- Precision – an estimate of the probability that the patient is in fact dead, provided that he or she is predicted to be dead.
- Also, the consequences of a mistaken diagnosis of a disease in a healthy person are usually less serious than if the disease is not diagnosed in a really sick person.
- So, in our case, sensitivity is more important.
- Also, we have an imbalance problem, so Accuracy can show wrong results.

Formulas:

- Sensitivity/Recall = TP / (TP + FN)
- Specificity = TN / (FP + TN)
- Precision = TP / (FP + TP)
- F1 = 2 ∗ precision * recall / (precision + recall)
- BalancedAccuracy = (sensitivity + specificity) / 2
- Accuracy = (TP + TN) / N

4.3 Linear regression

Our first clasification is based linear regression. We fited the model, which took into account all the variables. As can be seen the importatnt variables are the following: Sex, Malaise, Anorexia, SpleenPalpable, Spiders, Bilurubin, Albumin and Protime.

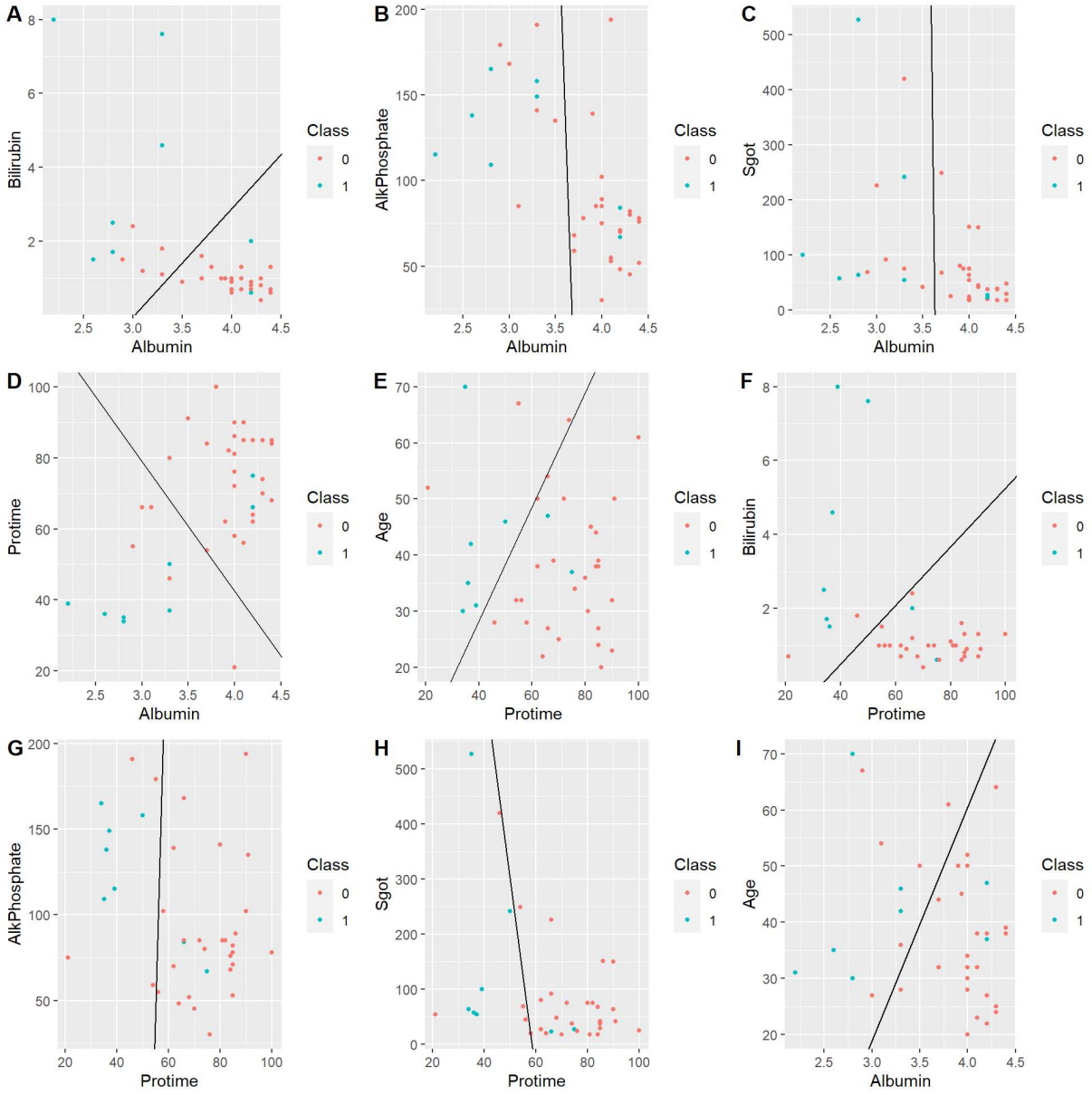
```
set.seed(123)
model.0 <- lm(Class~., data=train.balanced1)
summary(model.0)
```

```
##
## Call:
## lm(formula = Class ~ ., data = train.balanced1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.93020 -0.18083  0.00412  0.15896  1.13392
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)     0.4107282  0.3294763   1.247 0.214297
## Age             0.0032144  0.0023672   1.358 0.176353
## Sex             0.3615349  0.1089433   3.319 0.001112 **
## Steroid         0.0605472  0.0551066   1.099 0.273478
## Antivirals      0.0640862  0.0786034   0.815 0.416063
## Fatigue         0.0753427  0.0806802   0.934 0.351741
## Malaise         0.2914143  0.0754054   3.865 0.000159 ***
## Anorexia       -0.2496392  0.0738821  -3.379 0.000907 ***
## LiverBig       -0.0896927  0.0907706  -0.988 0.324530
## LiverFirm      -0.1096920  0.0682182  -1.608 0.109744
## SpleenPalpable  0.1929769  0.0634972   3.039 0.002757 **
## Spiders         0.1700447  0.0613978   2.770 0.006253 **
## Ascites         0.0986712  0.0931994   1.059 0.291269
## Varices         0.0019499  0.0868240   0.022 0.982109
## Bilirubin       0.0599509  0.0237290   2.526 0.012457 *
## AlkPhosphate   -0.0001686  0.0005735  -0.294 0.769091
## Sgot           -0.0002541  0.0003597  -0.706 0.480951
## Albumin        -0.1647514  0.0566994  -2.906 0.004164 **
## Protime        -0.0035964  0.0015505  -2.320 0.021582 *
## Histology       0.0363281  0.0611034   0.595 0.552964
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3016 on 166 degrees of freedom
## Multiple R-squared:  0.6754, Adjusted R-squared:  0.6382
## F-statistic: 18.18 on 19 and 166 DF,  p-value: < 2.2e-16
```

We then decided to construct the model using important features. We also wanted to chceck if our initial insights were correct and fitted nine models using all combinations of Albumin and Protime with other attributes

```
# two variables
model.1 <- lm(Class~Albumin+Bilirubin, data=train.balanced1)
model.2 <- lm(Class~Albumin+AlkPhosphate, data=train.balanced1)
model.3 <- lm(Class~Albumin+Sgot, data=train.balanced1)
model.4 <- lm(Class~Albumin+Protime, data=train.balanced1)
model.5 <- lm(Class~Protime+Age, data=train.balanced1)
model.6 <- lm(Class~Protime+Bilirubin, data=train.balanced1)
model.7 <- lm(Class~Protime+AlkPhosphate, data=train.balanced1)
model.8 <- lm(Class~Protime+Sgot, data=train.balanced1)
model.9 <- lm(Class~Albumin+Age, data=train.balanced1)

model.10 <- lm(Class~Sex+Malaise+Anorexia+SpleenPalpable+Spiders+Bilirubin+Albumin+Protime, data=train.balanced1)
```

The results for the test set are presented on the scatterplots below. Protime-Bilirubin and Protime-AlkPhosphate pairs seem best for distinguishing classes. We will also compare some metrics to better evaluate the performance of different models.



As can be seen in the table below, we can get the best result if we take into account all important attributes. However, models based on Protime-Bilirubin and Protime-Age pairs also performed well.

metric	All	Important	Albumin.Age	Albumin.Bilirubin	Albumin.AlkPhosphate	Albumin.Sgot
Accuracy	0.84	0.92	0.79	0.82	0.79	0.79
Kappa	0.56	0.75	0.46	0.51	0.46	0.46
Sensitivity	0.75	0.75	0.75	0.75	0.75	0.75
Specificity	0.87	0.97	0.80	0.83	0.80	0.80
F1	0.67	0.80	0.60	0.63	0.60	0.60
Balanced Accuracy	0.81	0.86	0.78	0.79	0.78	0.78
Precision	0.60	0.86	0.50	0.55	0.50	0.50

metric	Albumin.Protime	Protime.Age	Protime.Bilirubin	Protime.AlkPhosphate	Protime.Sgot
Accuracy	0.82	0.87	0.89	0.84	0.84
Kappa	0.51	0.62	0.68	0.56	0.56
Sensitivity	0.75	0.75	0.75	0.75	0.75
Specificity	0.83	0.90	0.93	0.87	0.87
F1	0.63	0.71	0.75	0.67	0.67
Balanced Accuracy	0.79	0.82	0.84	0.81	0.81
Precision	0.55	0.67	0.75	0.60	0.60

The final step will be to compare the three best models for two other imputation methods. It can be easily seen that for the knn imputation method we obtain the best results. For all three models, the accuracy, recall, precision and F-measure are really high.

```
model.6.1 <- lm(Class~Protime+Bilirubin, data=train.balanced1)
model.6.2 <- lm(Class~Protime+Bilirubin, data=train.balanced2)
model.6.3 <- lm(Class~Protime+Bilirubin, data=train.balanced3)

model.5.1 <- lm(Class~Protime+Age, data=train.balanced1)
model.5.2 <- lm(Class~Protime+Age, data=train.balanced2)
model.5.3 <- lm(Class~Protime+Age, data=train.balanced3)

model.10.1 <- lm(Class~Sex+Malaise+Anorexia+SpleenPalpable+Spiders+Bilirubin+Albumin+Protime, data=train.balanced1)
model.10.2 <- lm(Class~Sex+Malaise+Anorexia+SpleenPalpable+Spiders+Bilirubin+Albumin+Protime, data=train.balanced2)
model.10.3 <- lm(Class~Sex+Malaise+Anorexia+SpleenPalpable+Spiders+Bilirubin+Albumin+Protime, data=train.balanced3)
```

metric	Protime.Age.1	Protime.Age.2	Protime.Age.3
Accuracy	0.87	0.87	0.74
Kappa	0.62	0.62	0.27
Sensitivity	0.75	0.75	0.50
Specificity	0.90	0.90	0.80
F1	0.71	0.71	0.44
Balanced Accuracy	0.82	0.82	0.65
Precision	0.67	0.67	0.40

metric	Protime.Bilirubin.1	Protime.Bilirubin.2	Protime.Bilirubin.3
Accuracy	0.89	0.87	0.74
Kappa	0.68	0.62	0.33
Sensitivity	0.75	0.75	0.62
Specificity	0.93	0.90	0.77
F1	0.75	0.71	0.50
Balanced Accuracy	0.84	0.82	0.70
Precision	0.75	0.67	0.42

metric	Important.1	Important.2	Important.3
Accuracy	0.92	0.89	0.89
Kappa	0.75	0.68	0.68
Sensitivity	0.75	0.75	0.75
Specificity	0.97	0.93	0.93
F1	0.80	0.75	0.75
Balanced Accuracy	0.86	0.84	0.84
Precision	0.86	0.75	0.75

To conclude, in the case of linear regression clasification. The best model is the one including: Sex, Malaise, Anorexia, SpleenPalpable, Spiders, Bilurubin, Albumin and Protime features and the best results can be obtained for the knn imputation method.

4.4 Linear discriminant analysis (LDA)

For the binary clasification, where classes distribution is balanced the LDA is equivalent to the linerar regression. Hence this time we will use the inbalanced data set and compare LDA models for the different threshlods calculated based on the cost matrix. We will use the same features as before.

```
model.0 <- lda(Class~., data=train.num1)
model.1 <- lda(Class~Albumin+Bilirubin, data=train.num1)
model.2 <- lda(Class~Albumin+AlkPhosphate, data=train.num1)
model.3 <- lda(Class~Albumin+Sgot, data=train.num1)
model.4 <- lda(Class~Albumin+Protime, data=train.num1)
model.5 <- lda(Class~Protime+Age, data=train.num1)
model.6 <- lda(Class~Protime+Bilirubin, data=train.num1)
model.7 <- lda(Class~Protime+AlkPhosphate, data=train.num1)
model.8 <- lda(Class~Protime+Sgot, data=train.num1)
model.9 <- lda(Class~Albumin+Age, data=train.num1)

model.10 <- lda(Class~Sex+Malaise+Anorexia+SpleenPalpable+Spiders+Bilirubin+Albumin+Protime, data=train.num1)
```

First we will establish equal costs for classifying into incorrect classe. Hence the threshlod will be equal to 0.5. It can be observed that despite the high accuracy, recall for most of the cases is low.

metric	All	Important	Albumin.Age	Albumin.Bilirubin	Albumin.AlkPhosphate	Albumin.Sgot
Accuracy	0.92	0.92	0.82	0.89	0.84	0.84
Kappa	0.75	0.75	0.36	0.68	0.48	0.48
Sensitivity	0.75	0.75	0.38	0.75	0.50	0.50
Specificity	0.97	0.97	0.93	0.93	0.93	0.93
F1	0.80	0.80	0.46	0.75	0.57	0.57
Balanced Accuracy	0.86	0.86	0.65	0.84	0.72	0.72
Precision	0.86	0.86	0.60	0.75	0.67	0.67

metric	Albumin.Protime	Protime.Age	Protime.Bilirubin	Protime.AlkPhosphate	Protime.Sgot
Accuracy	0.89	0.79	0.89	0.79	0.79
Kappa	0.65	0.13	0.61	0.13	0.13
Sensitivity	0.62	0.12	0.50	0.12	0.12
Specificity	0.97	0.97	1.00	0.97	0.97
F1	0.71	0.20	0.67	0.20	0.20
Balanced Accuracy	0.80	0.55	0.75	0.55	0.55
Precision	0.83	0.50	1.00	0.50	0.50

We will now adjust the cost of incorrect classification of dead people to 5, which will change our threshold to 0.167. As can be seen in the table below, we managed to increase recall (an important metric for our clasification) at the expense of precision. Just like in the case of linear regression the best models are Protime-Bilirubin and the one including all important features.

metric	All	Important	Albumin.Age	Albumin.Bilirubin	Albumin.AlkPhosphate	Albumin.Sgot
Accuracy	0.84	0.82	0.74	0.79	0.74	0.74
Kappa	0.56	0.51	0.38	0.46	0.38	0.38
Sensitivity	0.75	0.75	0.75	0.75	0.75	0.75
Specificity	0.87	0.83	0.73	0.80	0.73	0.73
F1	0.67	0.63	0.55	0.60	0.55	0.55
Balanced Accuracy	0.81	0.79	0.74	0.78	0.74	0.74
Precision	0.60	0.55	0.43	0.50	0.43	0.43

metric	Albumin.Protime	Protime.Age	Protime.Bilirubin	Protime.AlkPhosphate	Protime.Sgot
Accuracy	0.79	0.79	0.87	0.76	0.79
Kappa	0.46	0.46	0.65	0.42	0.46
Sensitivity	0.75	0.75	0.88	0.75	0.75
Specificity	0.80	0.80	0.87	0.77	0.80
F1	0.60	0.60	0.74	0.57	0.60
Balanced Accuracy	0.78	0.78	0.87	0.76	0.78
Precision	0.50	0.50	0.64	0.46	0.50

While comparing the results for different imputation methods similar conclusions may be drawn. Protime-Bilirubin for knn imputation method is the best model.

metric	Protime.Bilirubin.1	Protime.Bilirubin.2	Protime.Bilirubin.3
Accuracy	0.87	0.79	0.74
Kappa	0.65	0.46	0.38

metric	Protime.Bilirubin.1	Protime.Bilirubin.2	Protime.Bilirubin.3
Sensitivity	0.88	0.75	0.75
Specificity	0.87	0.80	0.73
F1	0.74	0.60	0.55
Balanced Accuracy	0.87	0.78	0.74
Precision	0.64	0.50	0.43

metric	Important.1	Important.2	Important.3
Accuracy	0.82	0.82	0.79
Kappa	0.51	0.51	0.46
Sensitivity	0.75	0.75	0.75
Specificity	0.83	0.83	0.80
F1	0.63	0.63	0.60
Balanced Accuracy	0.79	0.79	0.78
Precision	0.55	0.55	0.50

4.5 Quadratic discriminant analysis (QDA)

For the QDA we will take into account only numeric variables and we will compare our results with LDA. The best model is the same (Protime-Bilirubin). However the recall and F1 are better for LDA.

On the other hand Protime-Age and Albumin-Bilirubin models improved they performance.

```
set.seed(123)
model.0 <- qda(Class~Age+Bilirubin+AlkPhosphate+Sgot+Albumin+Protime, data=train.num1)
model.1 <- qda(Class~Albumin+Bilirubin, data=train.num1)
model.2 <- qda(Class~Albumin+AlkPhosphate, data=train.num1)
model.3 <- qda(Class~Albumin+Sgot, data=train.num1)
model.4 <- qda(Class~Albumin+Protime, data=train.num1)
model.5 <- qda(Class~Protime+Age, data=train.num1)
model.6 <- qda(Class~Protime+Bilirubin, data=train.num1)
model.7 <- qda(Class~Protime+AlkPhosphate, data=train.num1)
model.8 <- qda(Class~Protime+Sgot, data=train.num1)
model.9 <- qda(Class~Albumin+Age, data=train.num1)
```

metric	All	Albumin.Age	Albumin.Bilirubin	Albumin.AlkPhosphate	Albumin.Sgot
Accuracy	0.84	0.68	0.82	0.76	0.76
Kappa	0.52	0.30	0.51	0.42	0.32
Sensitivity	0.62	0.75	0.75	0.75	0.50
Specificity	0.90	0.67	0.83	0.77	0.83
F1	0.62	0.50	0.63	0.57	0.47
Balanced Accuracy	0.76	0.71	0.79	0.76	0.67
Precision	0.62	0.38	0.55	0.46	0.44

metric	Albumin.Protime	Protime.Age	Protime.Bilirubin	Protime.AlkPhosphate	Protime.Sgot
Accuracy	0.79	0.84	0.87	0.76	0.71
Kappa	0.46	0.60	0.62	0.42	0.23
Sensitivity	0.75	0.88	0.75	0.75	0.50
Specificity	0.80	0.83	0.90	0.77	0.77
F1	0.60	0.70	0.71	0.57	0.42
Balanced Accuracy	0.78	0.85	0.82	0.76	0.63
Precision	0.50	0.58	0.67	0.46	0.36

As can be observed in the tables below, for the first model (Bilirubin-Protime) knn impute is the best. For the other model there aren't any differences between methods. To conclude, once again the Bilirubin and Protime attributes are the most efficient in distinguishing classes. The differences between LDA and QDA depends on features selected to build the model.

metric	Protime.Bilirubin.1	Protime.Bilirubin.2	Protime.Bilirubin.3
Accuracy	0.87	0.84	0.74
Kappa	0.62	0.56	0.33
Sensitivity	0.75	0.75	0.62

metric	Protime.Bilirubin.1	Protime.Bilirubin.2	Protime.Bilirubin.3
Specificity	0.90	0.87	0.77
F1	0.71	0.67	0.50
Balanced Accuracy	0.82	0.81	0.70
Precision	0.67	0.60	0.42

metric	Albumin.Bilirubin.1	Albumin.Bilirubin.2	Albumin.Bilirubin.3
Accuracy	0.82	0.82	0.82
Kappa	0.51	0.51	0.51
Sensitivity	0.75	0.75	0.75
Specificity	0.83	0.83	0.83
F1	0.63	0.63	0.63
Balanced Accuracy	0.79	0.79	0.79
Precision	0.55	0.55	0.55

metric	Albumin.Bilirubin.1	Albumin.Bilirubin.2	Albumin.Bilirubin.3
Accuracy	0.82	0.82	0.82
Kappa	0.51	0.51	0.51
Sensitivity	0.75	0.75	0.75
Specificity	0.83	0.83	0.83
F1	0.63	0.63	0.63
Balanced Accuracy	0.79	0.79	0.79
Precision	0.55	0.55	0.55

4.6 Logistic regression (LR)

4.6.1 Method

We need numeric data. In the beginning we use all attributes for different imputing methods.

```
set.seed(123)
model.logit1 <-  glm(Class~.-Class, data=train.balanced1, family=binomial(link="logit"))
model.logit2 <-  glm(Class~.-Class, data=train.balanced2, family=binomial(link="logit"))
model.logit3 <-  glm(Class~.-Class, data=train.balanced3, family=binomial(link="logit"))
```

Now, we build the logistic regression model without variables that are unnecessary from the summary and useless from EDA.

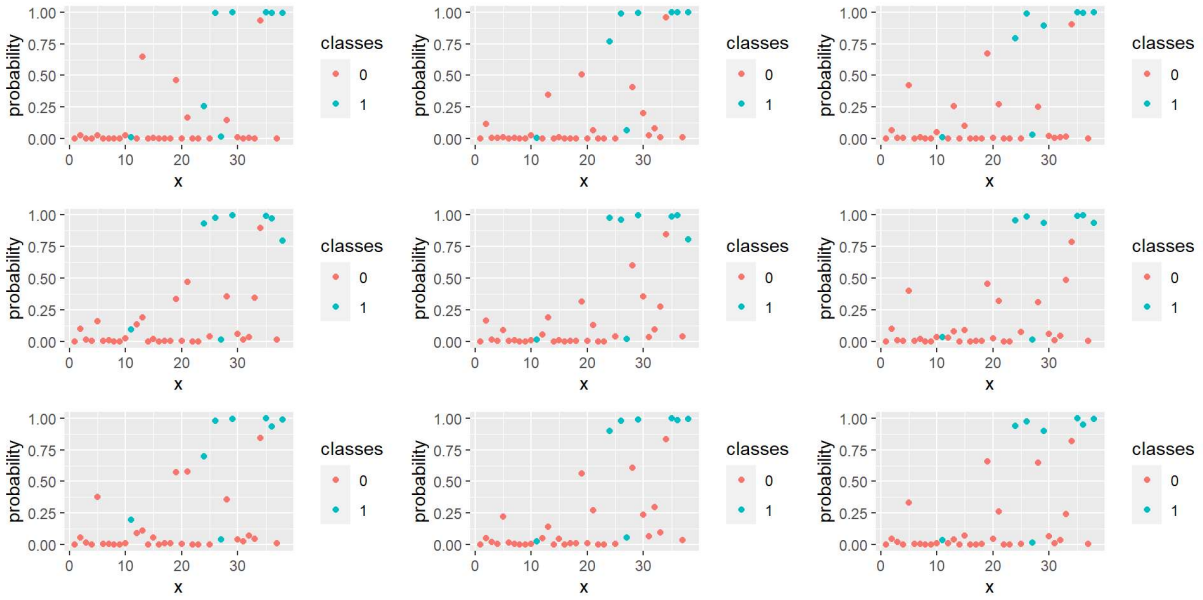
```
set.seed(123)
model.logit4 <-  glm(Class~.-Class-Age-Anorexia-Bilirubin, data=train.balanced1, family=binomial(link="logit"))
model.logit5 <-  glm(Class~.-Class-Age-Anorexia-Bilirubin, data=train.balanced2, family=binomial(link="logit"))
model.logit6 <-  glm(Class~.-Class-Age-Anorexia-Bilirubin, data=train.balanced3, family=binomial(link="logit"))

model.logit7 <-  glm(Class~.-Class-Age-Steroid-AlkPhosphate-Sgot, data=train.balanced1, family=binomial(link="logit"))
model.logit8 <-  glm(Class~.-Class-Age-Steroid-AlkPhosphate-Sgot, data=train.balanced2, family=binomial(link="logit"))
model.logit9 <-  glm(Class~.-Class-Age-Steroid-AlkPhosphate-Sgot, data=train.balanced3, family=binomial(link="logit"))
```

In the additional file, we can see summaries for all logistic regression models. There is a difference between statistics, but not so big.

Now, let's predict the posterior probability i.e. Pr(0|x). Plot results where color - class. Since we have the same splits, we can use the same n, p, colors.

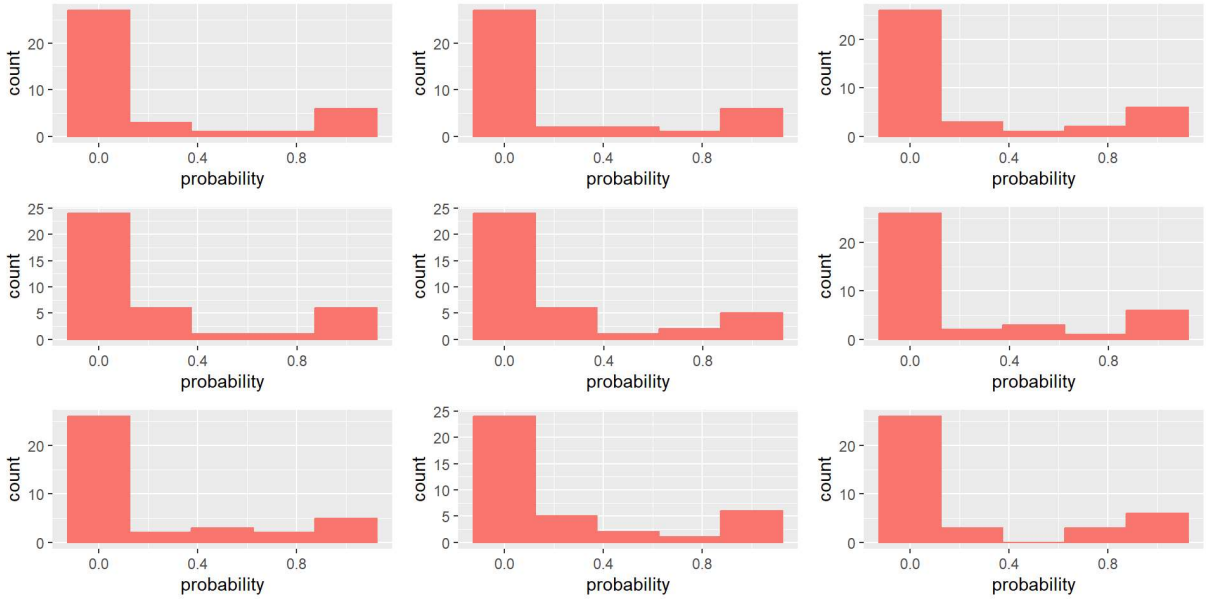
There are false predictions in all models, but not so many.

```
pred.prob1 <- predict(model.logit1, test.num1, type = "response")
pred.prob2 <- predict(model.logit2, test.num2, type = "response")
pred.prob3 <- predict(model.logit3, test.num3, type = "response")

pred.prob4 <- predict(model.logit4, test.num1, type = "response")
pred.prob5 <- predict(model.logit5, test.num2, type = "response")
pred.prob6 <- predict(model.logit6, test.num3, type = "response")

pred.prob7 <- predict(model.logit7, test.num1, type = "response")
pred.prob8 <- predict(model.logit8, test.num2, type = "response")
pred.prob9 <- predict(model.logit9, test.num3, type = "response")
```



Let's also plot histograms of predicted probabilities.

Most probabilities for all models are about 0 or 1, what we need.



Now, let's convert probabilities to class labels for a given cutoff

```
pred.labels1 <- prob.to.labels(probs=pred.prob1, cutoff=0.5)
pred.labels2 <- prob.to.labels(probs=pred.prob2, cutoff=0.5)
pred.labels3 <- prob.to.labels(probs=pred.prob3, cutoff=0.5)

pred.labels4 <- prob.to.labels(probs=pred.prob4, cutoff=0.5)
pred.labels5 <- prob.to.labels(probs=pred.prob5, cutoff=0.5)
pred.labels6 <- prob.to.labels(probs=pred.prob6, cutoff=0.5)

pred.labels7 <- prob.to.labels(probs=pred.prob7, cutoff=0.5)
pred.labels8 <- prob.to.labels(probs=pred.prob8, cutoff=0.5)
pred.labels9 <- prob.to.labels(probs=pred.prob9, cutoff=0.5)

real.labels <- test.num1$Class
```

4.6.2 Comparison

In the additional file, we can see all confusion matrices.

```
logit_statistics %>%
    flextable() %>%
    theme_box() %>%
    autofit()
```

name	logit1	logit2	logit3	logit4	logit5	logit6	logit7	logit8	logit9
Accuracy	0.87	0.89	0.89	0.92	0.89	0.92	0.87	0.87	0.87
Kappa	0.59	0.68	0.68	0.75	0.68	0.75	0.62	0.62	0.62

name	logit1	logit2	logit3	logit4	logit5	logit6	logit7	logit8	logit9
Sensitivity	0.62	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75
Specificity	0.93	0.93	0.93	0.97	0.93	0.97	0.90	0.90	0.90
F1	0.67	0.75	0.75	0.80	0.75	0.80	0.71	0.71	0.71
Balanced Accuracy	0.78	0.84	0.84	0.86	0.84	0.86	0.82	0.82	0.82
10-fold CV error	0.34	0.34	0.34	0.35	0.35	0.36	0.33	0.32	0.32
Bootstrap error	0.37	0.36	0.37	0.36	0.36	0.38	0.34	0.34	0.35

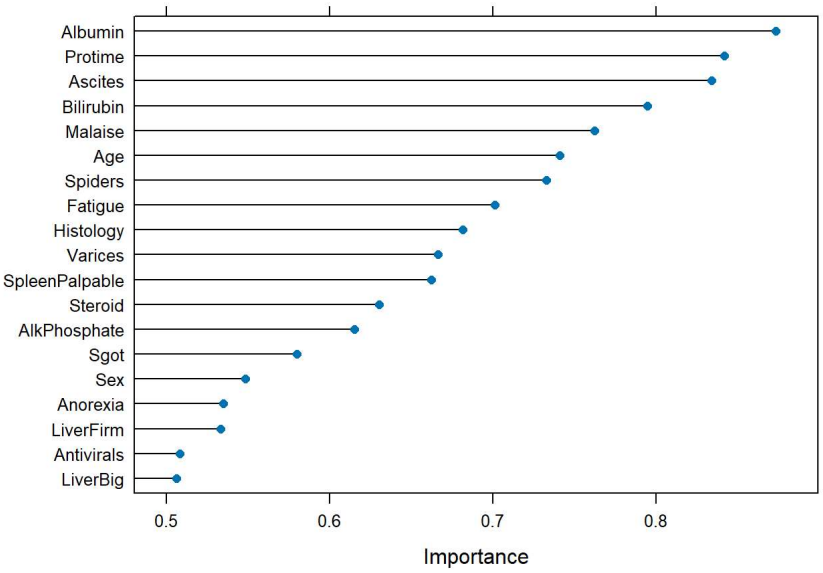
So, the best Logistic regression model when comparing the statistics:

- Accuracy: 4, 6
- Kappa: 4, 6
- Sensitivity: 2-9
- Specificity: 4, 6
- F1: 4, 6
- Balanced Accuracy: 4, 6
- 10-fold CV: 8, 9
- Bootstrap: 7, 8

To sum up, the best Logistic regression model is the **4th** one: model without useless variables and knn imputing method.

4.7 KNN

First we performed 5 times repeated 5-fold Cross Validation for a model that takes into account all variables. We chose the 5 most important features: Albumin, Ascites, Protime, Malaise, Bilirubin and check all their combinations.




```
set.seed(123)
comb <- list(c(18,13),c(18,19),c(18,7),c(18,15),c(13,19),c(13,7),c(13,15),c(19,7),c(19,15),c(7,15))

all_metrics <- list(c(metrices(predict(knn.model.0, newdata=test.num1.std[-1]), test1$Class), knn.model.0$bestTune[[1]]))
for (i in 1:10) {
  knn.model <- train(train.balanced1.std[comb[[i]]], as.factor(train.balanced1$Class), method="knn", tuneGrid =  k.grid, trC
ontrol=cvControl)
  all_metrics <- lappend(all_metrics,c(metrices(predict(knn.model, newdata=test.num1.std[comb[[i]]]), test1$Class), knn.mode
l$bestTune[[1]]))
}
```

metric	All	Albumin.Ascites	Albumin.Protime	Albumin.Malaise	Albumin.Bilirubin
Accuracy	0.74	0.50	0.76	0.74	0.66
Kappa	0.38	0.00	0.42	0.33	0.27
Sensitivity	0.75	0.50	0.75	0.62	0.75
Specificity	0.73	0.50	0.77	0.77	0.63
F1	0.55	0.30	0.57	0.50	0.48
Balanced Accuracy	0.74	0.50	0.76	0.70	0.69

metric	All	Albumin.Ascites	Albumin.Protime	Albumin.Malaise	Albumin.Bilirubin
Precision	0.43	0.21	0.46	0.42	0.35
k	1.00	1.00	24.00	10.00	17.00

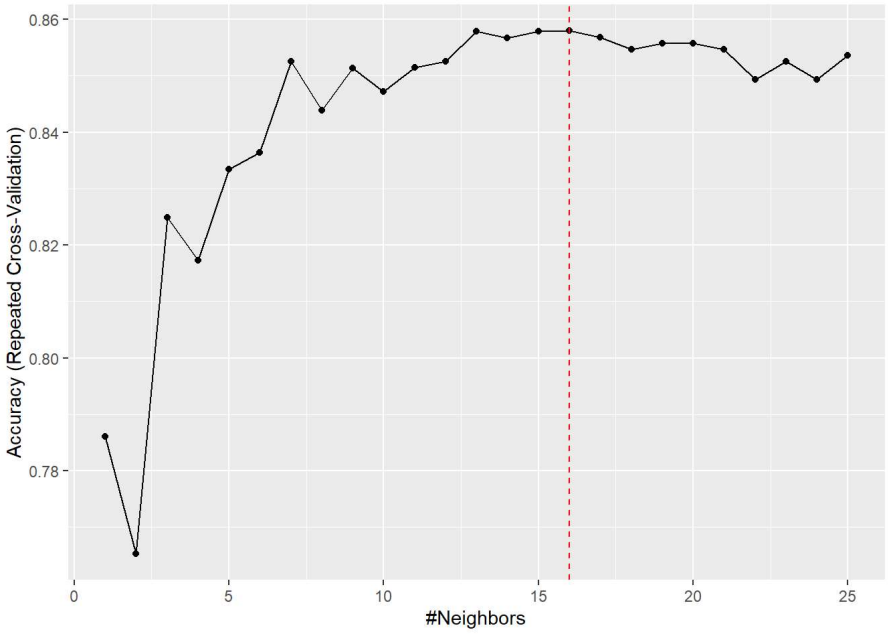
Ascites.Protime	Ascites.Malaise	Ascites.Bilirubin	Protime.Malaise	Protime.Bilirubin
0.76	0.21	0.21	0.58	0.82
0.42	0.00	-0.04	0.13	0.55
0.75	1.00	0.88	0.62	0.88
0.77	0.00	0.03	0.57	0.80
0.57	0.35	0.32	0.38	0.67
0.76	0.50	0.45	0.60	0.84
0.46	0.21	0.19	0.28	0.54
9.00	1.00	1.00	4.00	18.00

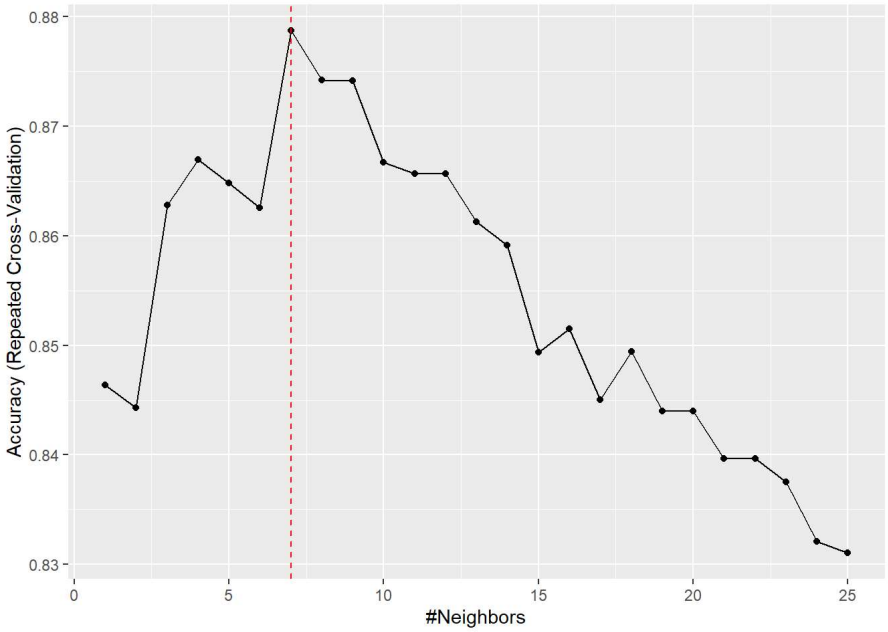
```
set.seed(123)
comb <- list(c(18,13,19),c(18,13,7),c(18,13,15),c(18,19,7),c(18,19,15),c(18,7,15),c(13,19,7),c(13,19,15),c(13,7,15),c(19,7,1
5))

all_metrics <- list(c(metrices(predict(knn.model.0, newdata=test.num1.std[-1]), test1$Class), knn.model.0$bestTune[[1]]))
for (i in 1:10) {
  knn.model <- train(train.balanced1.std[comb[[i]]], as.factor(train.balanced1$Class), method="knn", tuneGrid =  k.grid, trC
ontrol=cvControl)
  all_metrics <- lappend(all_metrics,c(metrices(predict(knn.model, newdata=test.num1.std[comb[[i]]]), test1$Class), knn.mode
l$bestTune[[1]]))
}
```

metric	All	Albumin.Ascites.Protime	Albumin.Ascites.Malaise	Albumin.Ascites.Bilirubin	Albumin.Protime.Malaise
Accuracy	0.74	0.66	0.53	0.63	0.74
Kappa	0.38	0.16	0.02	0.19	0.38
Sensitivity	0.75	0.50	0.50	0.62	0.75
Specificity	0.73	0.70	0.53	0.63	0.73
F1	0.55	0.38	0.31	0.42	0.55
Balanced Accuracy	0.74	0.60	0.52	0.63	0.74
Precision	0.43	0.31	0.22	0.31	0.43
k	1.00	3.00	1.00	1.00	5.00

Albumin.Protime.Bilirubin	Albumin.Malaise.Bilirubin	Ascites.Protime.Malaise	Ascites.Protime.Bilirubin	Ascites.Malaise.Bilirubin
0.74	0.71	0.79	0.76	0.16
0.38	0.29	0.46	0.46	-0.26
0.75	0.62	0.75	0.88	0.38
0.73	0.73	0.80	0.73	0.10
0.55	0.48	0.60	0.61	0.16
0.74	0.68	0.78	0.80	0.24
0.43	0.38	0.50	0.47	0.10
9.00	8.00	7.00	25.00	1.00

As can be seen in the tables above the best results can be obtain for the following features: Protime-Bilirubin and Ascites-Protime-Malaise. We will plot the accuracy as a function of number of neighbours to see how it behaves. For both cases the accuracy first increases and then started to decrease. Hence we will consider the value selected using cross validation to be optimal.





We will finally compare different imputation methods. Protime-Bilirubin for knn imputation method is again the best model.

```
set.seed(123)
train.balanced1.std$Class <- as.factor(train.balanced1$Class)
test.num1.std$Class <- as.factor(test.num1$Class)

model.1.1 <- knn(train.balanced1.std[c(19,15)], test.num1.std[c(19,15)], train.balanced1.std$Class, k=18)
model.1.2 <- knn(train.balanced2.std[c(19,15)], test.num2.std[c(19,15)], train.balanced2.std$Class, k=18)
model.1.3 <- knn(train.balanced3.std[c(19,15)], test.num3.std[c(19,15)], train.balanced3.std$Class, k=18)

model.2.1 <- knn(train.balanced1.std[c(7,13,19)], test.num1.std[c(7,13,19)], train.balanced1.std$Class, k=7)
model.2.2 <- knn(train.balanced2.std[c(7,13,19)], test.num2.std[c(7,13,19)], train.balanced2.std$Class, k=7)
model.2.3 <- knn(train.balanced3.std[c(7,13,19)], test.num3.std[c(7,13,19)], train.balanced3.std$Class, k=7)
```

metric	Protime.Bilirubin.1	Protime.Bilirubin.2	Protime.Bilirubin.3
Accuracy	0.79	0.71	0.68
Kappa	0.50	0.34	0.30
Sensitivity	0.88	0.75	0.75
Specificity	0.77	0.70	0.67
F1	0.64	0.52	0.50
Balanced Accuracy	0.82	0.72	0.71
Precision	0.50	0.40	0.38

metric	Ascites.Protime.Malaise.1	Ascites.Protime.Malaise.2	Ascites.Protime.Malaise.3
Accuracy	0.79	0.79	0.71

metric	Ascites.Protime.Malaise.1	Ascites.Protime.Malaise.2	Ascites.Protime.Malaise.3
Kappa	0.46	0.46	0.29
Sensitivity	0.75	0.75	0.62
Specificity	0.80	0.80	0.73
F1	0.60	0.60	0.48
Balanced Accuracy	0.78	0.78	0.68
Precision	0.50	0.50	0.38

4.8 Random tree

4.8.1 Method

We can use both numeric and categorical data. Let's specify model formula. We use all attributes and also model without variables that are useless from EDA.

```
mod1 <- Class ~ . - Class
mod2 <- Class ~ . - Class - Age - Steroid - AlkPhosphate - Sgot
```

Building full trees for different models and different datasets (for different imputation methods and initial with missing values).

```
set.seed(123)

full.tree1 <- rpart(mod1, data=train1, control=rpart.control(cp=-1, minsplit=5))
full.tree2 <- rpart(mod1, data=train2, control=rpart.control(cp=-1, minsplit=5))
full.tree3 <- rpart(mod1, data=train3, control=rpart.control(cp=-1, minsplit=5))
full.tree4 <- rpart(mod1, data=train0, control=rpart.control(cp=-1, minsplit=5))

full.tree5 <- rpart(mod2, data=train1, control=rpart.control(cp=-1, minsplit=5))
full.tree6 <- rpart(mod2, data=train2, control=rpart.control(cp=-1, minsplit=5))
full.tree7 <- rpart(mod2, data=train3, control=rpart.control(cp=-1, minsplit=5))
full.tree8 <- rpart(mod2, data=train0, control=rpart.control(cp=-1, minsplit=5))
```

In the additional file, we can see a visualization of all full trees. They are pretty different.

Now, we need to choose a complexity parameter (cp). In the additional file, we can see plots and information about misclassification error.

We will use 1SE (one standard error) rule: As the optimal tree, we choose the tree with the smallest number of splits (i.e. the smallest size) for which a misclassification error is within one standard error from the minimum misclassification error.

```
cp.opt1 <- 0.125
cp.opt2 <- 0.083
cp.opt3 <- 0.083
cp.opt4 <- 0.041
cp.opt5 <- 0.083
cp.opt6 <- 0.083
cp.opt7 <- 0.166
cp.opt8 <- 0.062
```

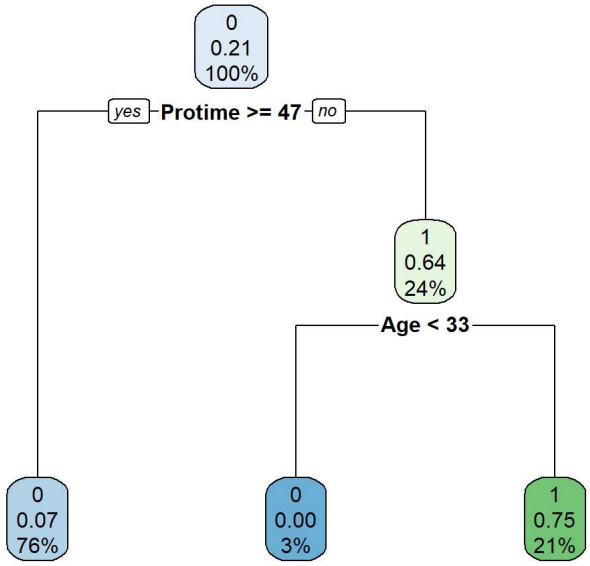
Now, we prune trees.

```
full.tree1.pruned <- prune(full.tree1, cp = cp.opt1)
full.tree2.pruned <- prune(full.tree2, cp = cp.opt2)
full.tree3.pruned <- prune(full.tree3, cp = cp.opt3)
full.tree4.pruned <- prune(full.tree4, cp = cp.opt4)
full.tree5.pruned <- prune(full.tree5, cp = cp.opt5)
full.tree6.pruned <- prune(full.tree6, cp = cp.opt6)
full.tree7.pruned <- prune(full.tree7, cp = cp.opt7)
full.tree8.pruned <- prune(full.tree8, cp = cp.opt8)
```

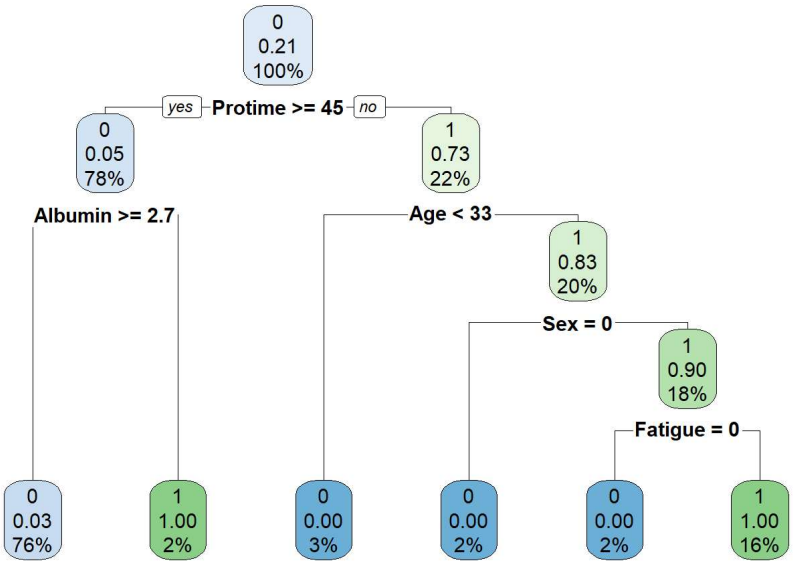
In the additional file, we can find basic information about the trees.

Now, let's visualize all the trees. They are pretty different: from a tree with only 2 splits to a tree with many splits.

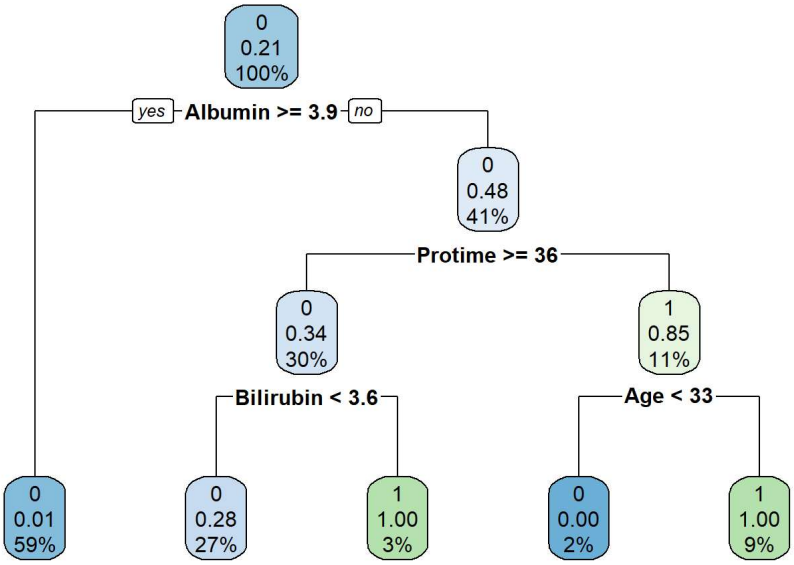
```
rpart.plot(full.tree1.pruned)
```



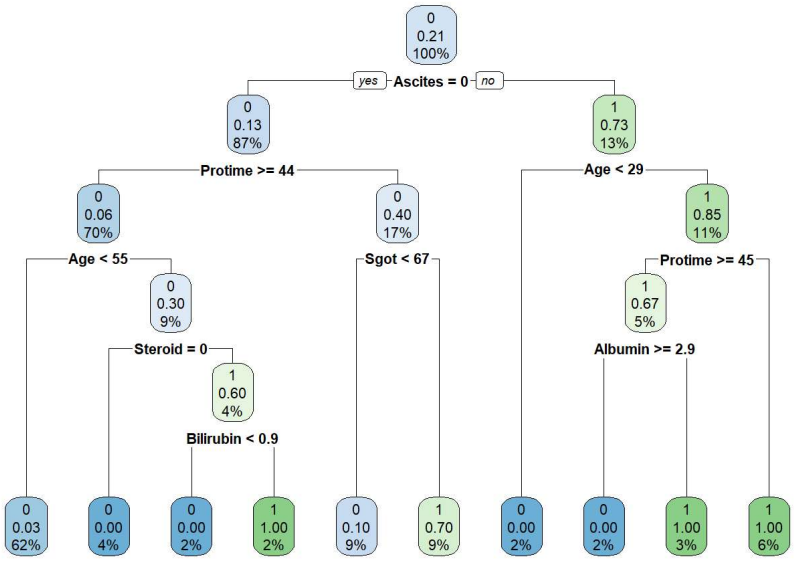

```
rpart.plot(full.tree2.pruned)
```



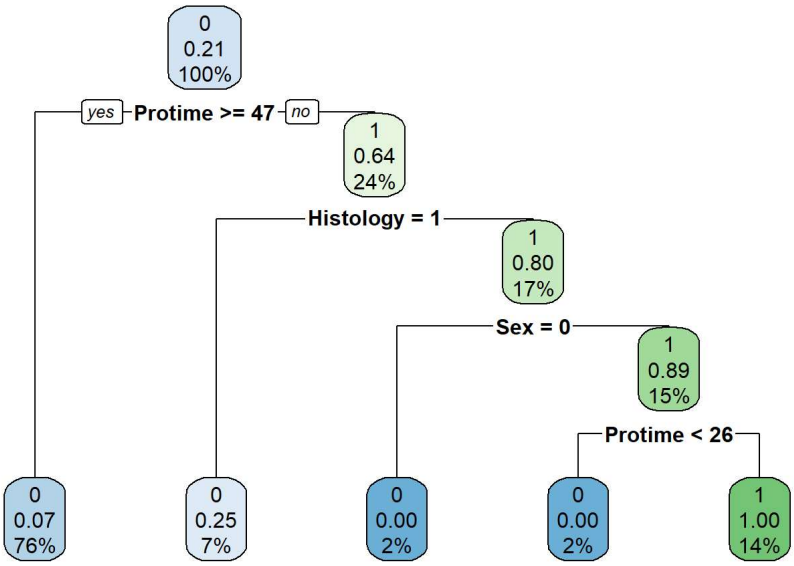

```
rpart.plot(full.tree3.pruned)
```



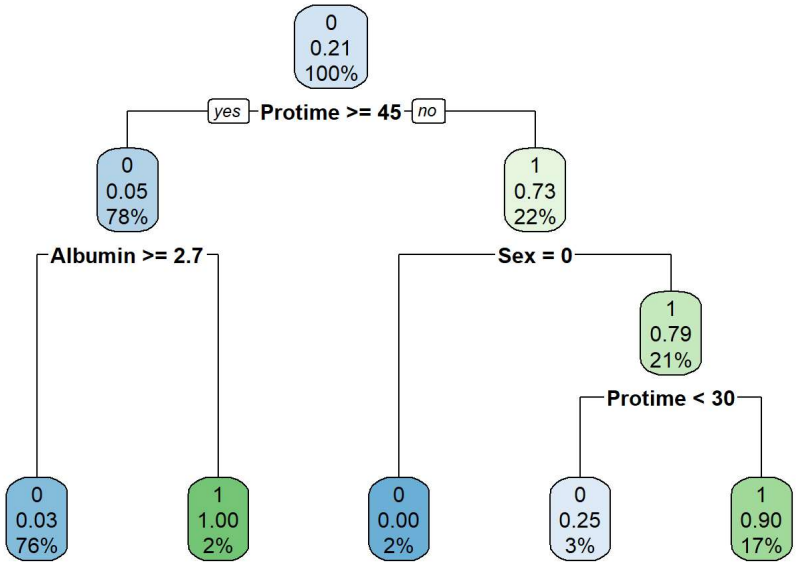
```
rpart.plot(full.tree4.pruned)
```



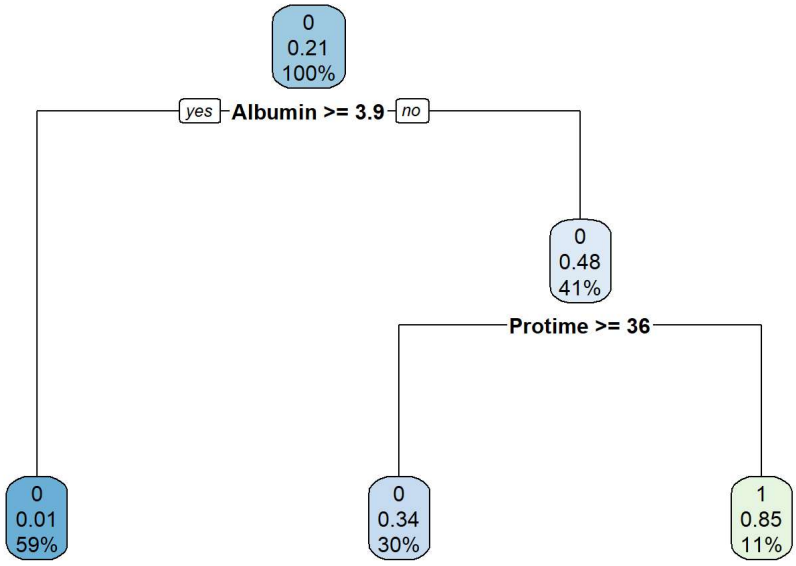

```
rpart.plot(full.tree5.pruned)
```



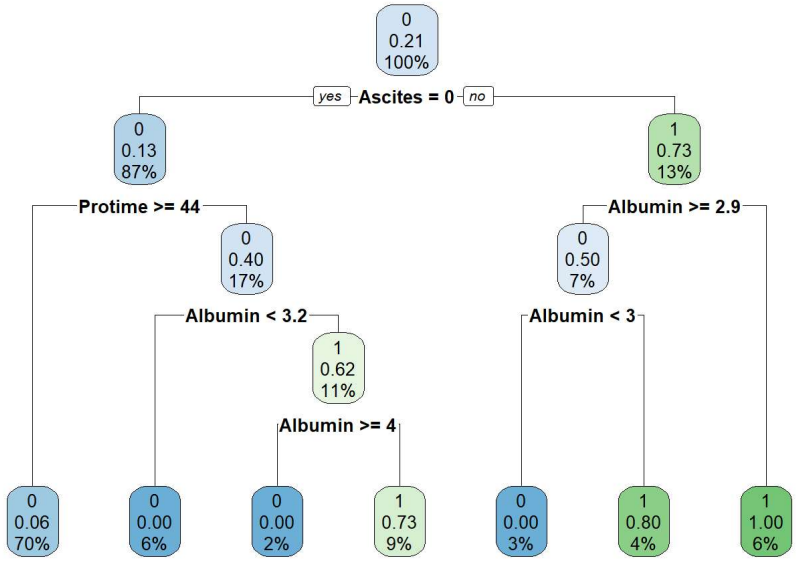

```
rpart.plot(full.tree6.pruned)
```




```
rpart.plot(full.tree7.pruned)
```



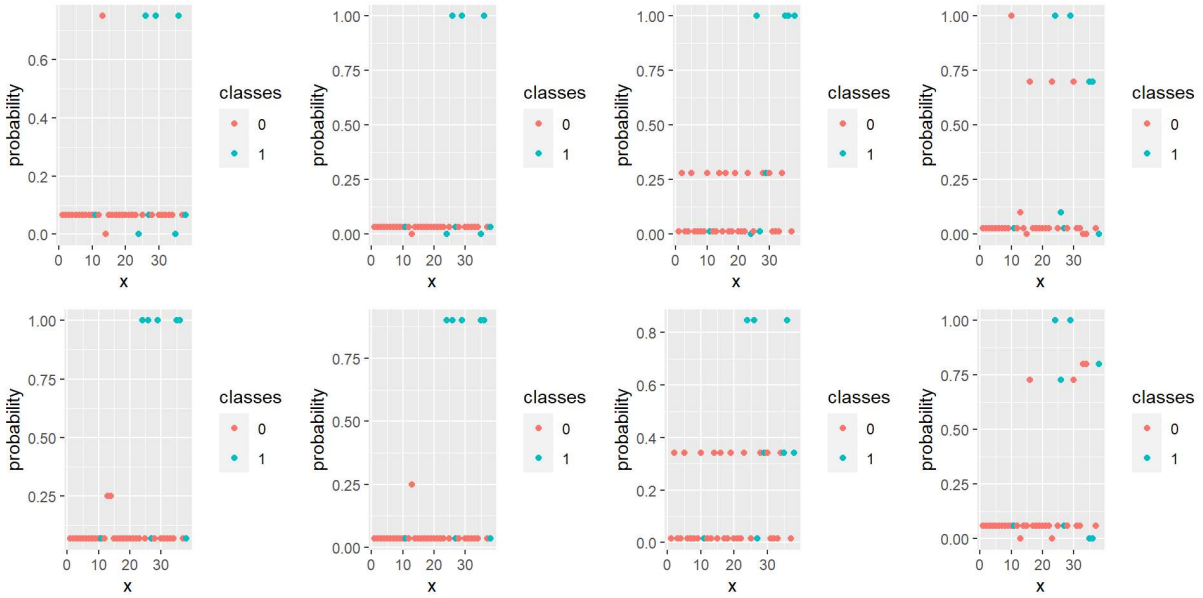

```
rpart.plot(full.tree8.pruned)
```



Predicted posterior probabilities. Plot results where color - class. Since we have the same splits, we can use the same n, p, colors.

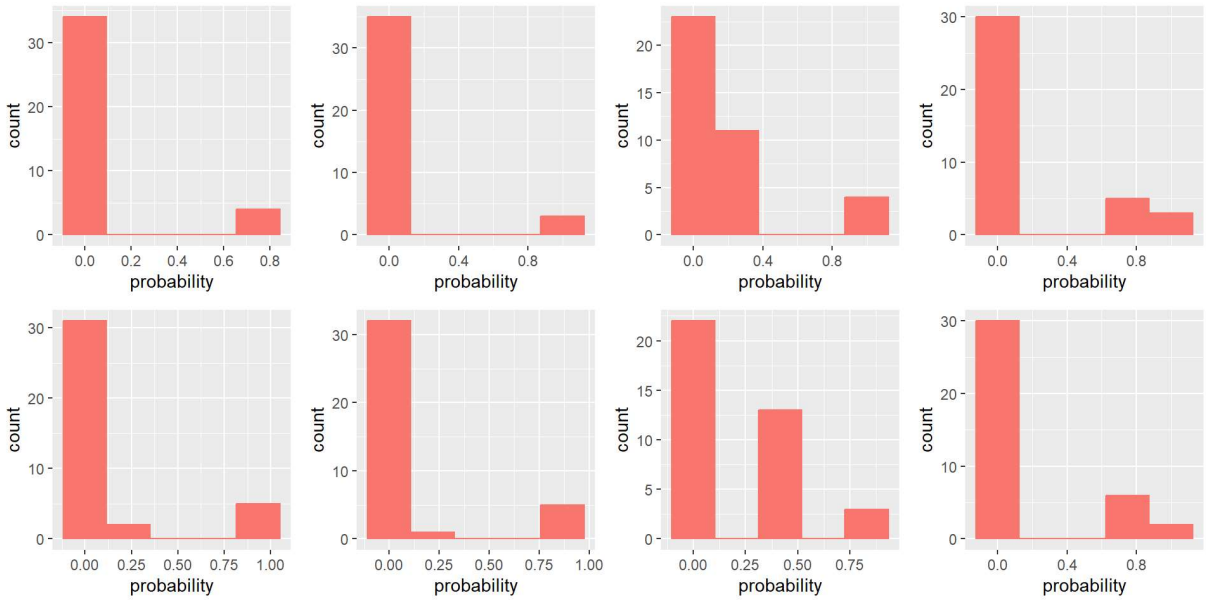
There are false predictions in all models, not so few, especially many false negative values.

```
pred.probs1 <- predict(full.tree1.pruned, newdata=test1, type = "prob")
pred.probs2 <- predict(full.tree2.pruned, newdata=test2, type = "prob")
pred.probs3 <- predict(full.tree3.pruned, newdata=test3, type = "prob")
pred.probs4 <- predict(full.tree4.pruned, newdata=test0, type = "prob")

pred.probs5 <- predict(full.tree5.pruned, newdata=test1, type = "prob")
pred.probs6 <- predict(full.tree6.pruned, newdata=test2, type = "prob")
pred.probs7 <- predict(full.tree7.pruned, newdata=test3, type = "prob")
pred.probs8 <- predict(full.tree8.pruned, newdata=test0, type = "prob")
```



Let's also plot histograms of predicted probabilities.

There are both good and bad results. There are histograms, where most probabilities are about 0 or 1, what we need. But there are also histograms, where a lot of values are located in the middle between 0 and 1.



Predicted class labels.

```
pred.labels1 <- predict(full.tree1.pruned, newdata=test1, type = "class")
pred.labels2 <- predict(full.tree2.pruned, newdata=test2, type = "class")
pred.labels3 <- predict(full.tree3.pruned, newdata=test3, type = "class")
pred.labels4 <- predict(full.tree4.pruned, newdata=test0, type = "class")

pred.labels5 <- predict(full.tree5.pruned, newdata=test1, type = "class")
pred.labels6 <- predict(full.tree6.pruned, newdata=test2, type = "class")
pred.labels7 <- predict(full.tree7.pruned, newdata=test3, type = "class")
pred.labels8 <- predict(full.tree8.pruned, newdata=test0, type = "class")

real.labels <- test1$Class
```

4.8.2 Comparison

```
tree_statistics %>%
    flextable() %>%
    theme_box() %>%
    autofit()
```

name	tree1	tree2	tree3	tree4	tree5	tree6	tree7	tree8
Accuracy	0.84	0.87	0.89	0.79	0.92	0.92	0.87	0.79
Kappa	0.42	0.49	0.61	0.37	0.72	0.72	0.49	0.37
Sensitivity	0.38	0.38	0.50	0.50	0.62	0.62	0.38	0.50
Specificity	0.97	1.00	1.00	0.87	1.00	1.00	1.00	0.87
F1	0.50	0.55	0.67	0.50	0.77	0.77	0.55	0.50
Balanced Accuracy	0.67	0.69	0.75	0.68	0.81	0.81	0.69	0.68
10-fold CV error	0.17	0.13	0.17	0.20	0.18	0.14	0.18	0.16
Bootstrap error	0.20	0.16	0.21	0.18	0.16	0.15	0.19	0.18

So, the best Random tree model when comparing the statistics:

- Accuracy: 5, 6
- Kappa: 5, 6
- Sensitivity: 5, 6
- Specificity: 2, 3, 5, 6, 7
- F1: 5, 6
- Balanced Accuracy: 5, 6
- 10-fold CV: 2 -> 6
- Bootstrap: 6

To sum up, the best Random tree model is the **6th** one: model without variables that are useless from EDA and bag tree imputing method.

4.9 Bagging (bootstrap aggregating)

4.9.1 Method

We can use both numeric and categorical data. Let's specify model formula. We use all attributes and also model without variables that are useless from EDA.

```
mod1 <- Class ~ . - Class
mod2 <- Class ~ . - Class - Age - Steroid - AlkPhosphate - Sgot
```

Building random forest for different models and datasets (for different imputation methods and initial with missing values).

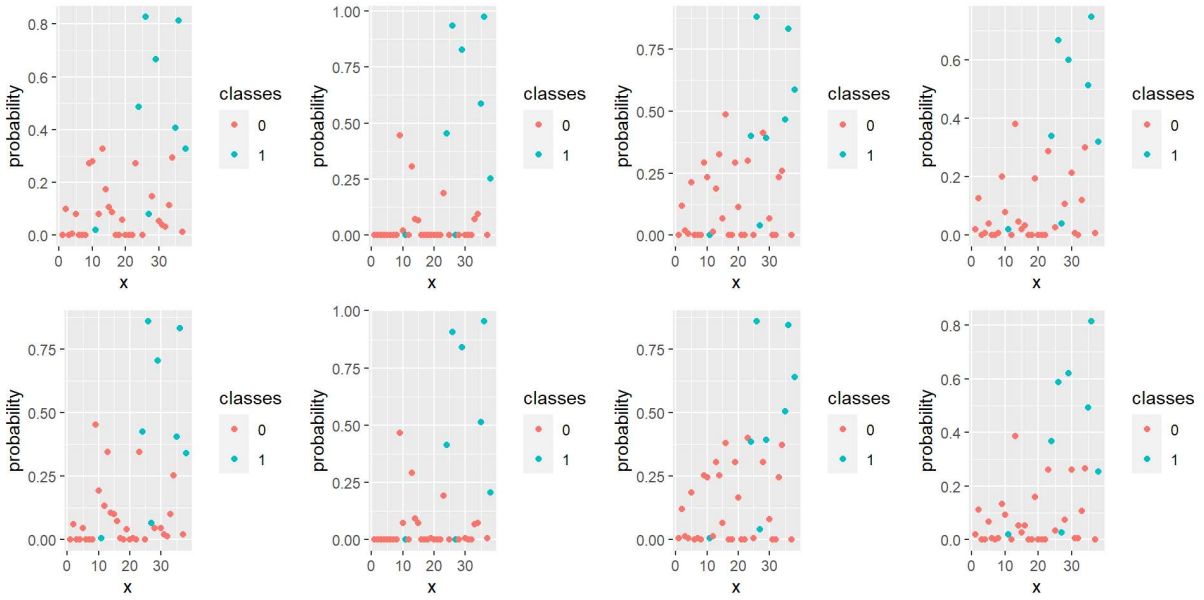
```
set.seed(123)

btree1 <- bagging(mod1, data=train1, nbagg=150, coob=TRUE, minsplit=2, cp=0)
btree2 <- bagging(mod1, data=train2, nbagg=150, coob=TRUE, minsplit=2, cp=0)
btree3 <- bagging(mod1, data=train3, nbagg=150, coob=TRUE, minsplit=2, cp=0)
btree4 <- bagging(mod1, data=train0, nbagg=150, coob=TRUE, minsplit=2, cp=0)

btree5 <- bagging(mod2, data=train1, nbagg=150, coob=TRUE, minsplit=2, cp=0)
btree6 <- bagging(mod2, data=train2, nbagg=150, coob=TRUE, minsplit=2, cp=0)
btree7 <- bagging(mod2, data=train3, nbagg=150, coob=TRUE, minsplit=2, cp=0)
btree8 <- bagging(mod2, data=train0, nbagg=150, coob=TRUE, minsplit=2, cp=0)
```

In the additional file, we can find basic information about the models.

Predicted posterior probabilities. Plot results where color - class. Since we have the same splits, we can use the same n, p, colors.

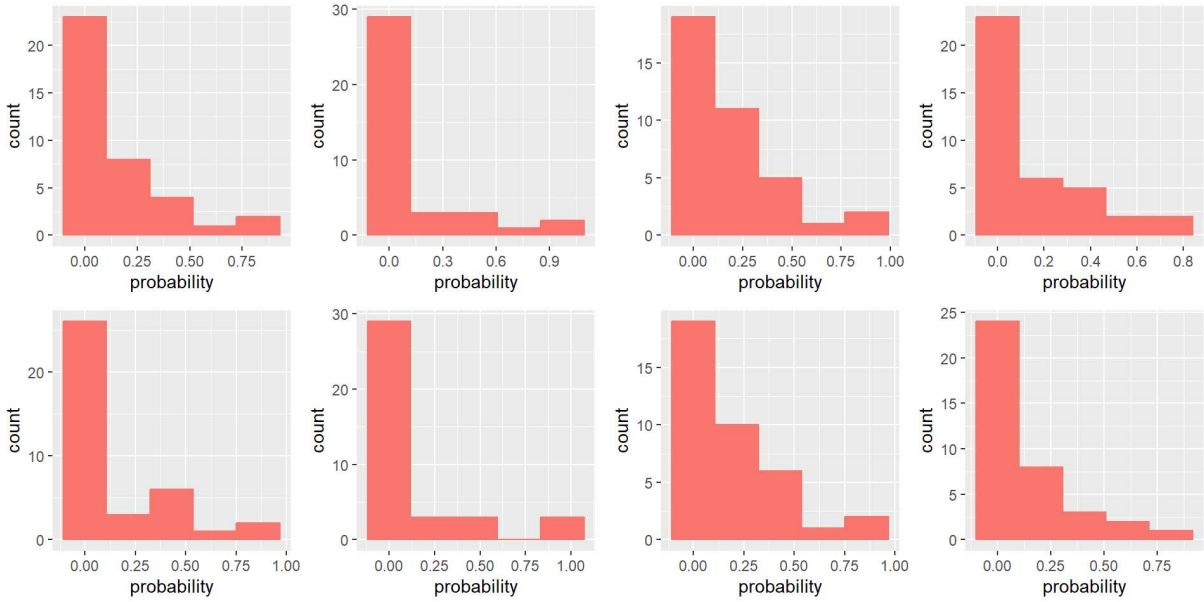
The results are not so good, because there are false predictions in all models and many probabilities in the middle.

```
pred.probs1 <- predict(btree1, newdata=test1, type = "prob")
pred.probs2 <- predict(btree2, newdata=test2, type = "prob")
pred.probs3 <- predict(btree3, newdata=test3, type = "prob")
pred.probs4 <- predict(btree4, newdata=test0, type = "prob")

pred.probs5 <- predict(btree5, newdata=test1, type = "prob")
pred.probs6 <- predict(btree6, newdata=test2, type = "prob")
pred.probs7 <- predict(btree7, newdata=test3, type = "prob")
pred.probs8 <- predict(btree8, newdata=test0, type = "prob")
```



Let's also plot histograms of predicted probabilities.

As we saw before, the results are not so good: many values are about 0, but also a lot of values are located in the middle between 0 and 1.



Predicted class labels.

```
pred.labels1 <- predict(btree1, newdata=test1, type = "class")
pred.labels2 <- predict(btree2, newdata=test2, type = "class")
pred.labels3 <- predict(btree3, newdata=test3, type = "class")
pred.labels4 <- predict(btree4, newdata=test0, type = "class")

pred.labels5 <- predict(btree5, newdata=test1, type = "class")
pred.labels6 <- predict(btree6, newdata=test2, type = "class")
pred.labels7 <- predict(btree7, newdata=test3, type = "class")
pred.labels8 <- predict(btree8, newdata=test0, type = "class")

real.labels <- test1$Class
```

4.9.2 Comparison

```
btree_statistics %>%
    flextable() %>%
    theme_box() %>%
    autofit()
```

name	btree1	btree2	btree3	btree4	btree5	btree6	btree7	btree8
Accuracy	0.87	0.89	0.87	0.89	0.87	0.89	0.89	0.87
Kappa	0.49	0.61	0.49	0.61	0.49	0.61	0.61	0.49
Sensitivity	0.38	0.50	0.38	0.50	0.38	0.50	0.50	0.38
Specificity	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

name	btree1	btree2	btree3	btree4	btree5	btree6	btree7	btree8
F1	0.55	0.67	0.55	0.67	0.55	0.67	0.67	0.55
Balanced Accuracy	0.69	0.75	0.69	0.75	0.69	0.75	0.75	0.69
10-fold CV error	0.17	0.15	0.17	0.16	0.14	0.12	0.16	0.15
Bootstrap error	0.17	0.15	0.18	0.16	0.16	0.14	0.18	0.17

So, the best Bagging model when comparing the statistics:

- Accuracy: 2, 4, 6, 7
- Kappa: 2, 4, 6, 7
- Sensitivity: 2, 4, 6, 7
- Specificity: all
- F1: 2, 4, 6, 7
- Balanced Accuracy: 2, 4, 6, 7
- 10-fold CV: 6
- Bootstrap: 6

To sum up, the best Bagging model is the **6th** one: model without variables that are useless from EDA and bag tree imputing method

4.10 Boosting

4.10.1 Method

We can use both numeric and categorical data. Let's specify model formula. We use all attributes and also model without variables that are useless from EDA.

```
mod1 <- Class ~ . - Class
mod2 <- Class ~ . - Class - Age - Steroid - AlkPhosphate - Sgot
```

Building random forest for different models and datasets (for different imputation methods).

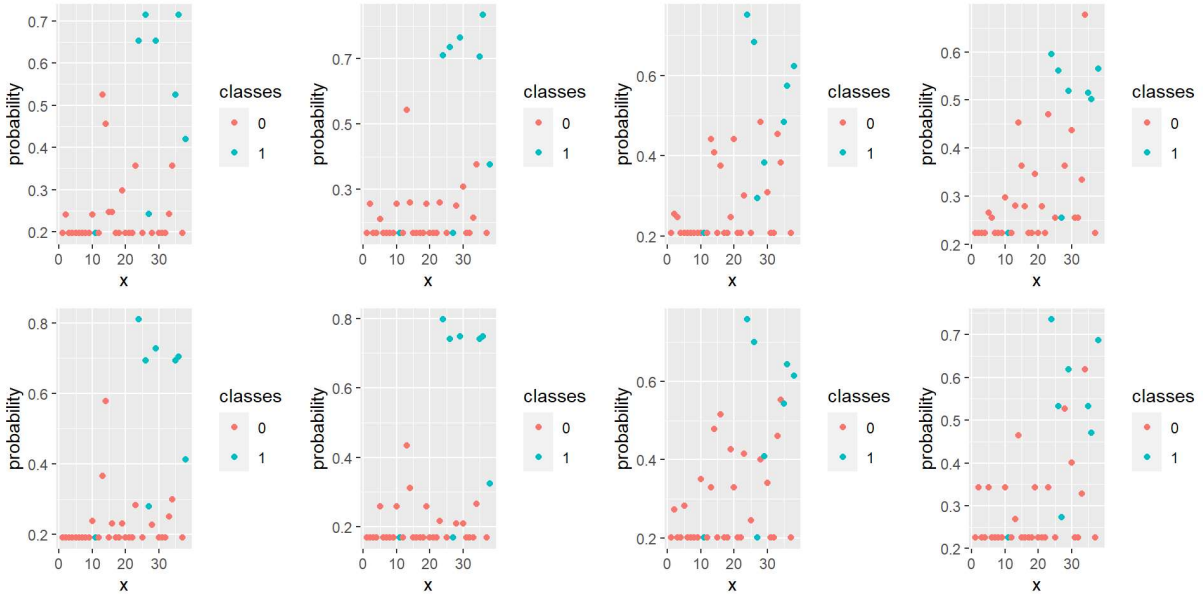
```
set.seed(123)

boost1 <- ada(mod1, data=train1, iter = 10)
boost2 <- ada(mod1, data=train2, iter = 10)
boost3 <- ada(mod1, data=train3, iter = 10)
boost4 <- ada(mod1, data=train0, iter = 10)

boost5 <- ada(mod2, data=train1, iter = 10)
boost6 <- ada(mod2, data=train2, iter = 10)
boost7 <- ada(mod2, data=train3, iter = 10)
boost8 <- ada(mod2, data=train0, iter = 10)
```

In the additional file, we can find basic information about all models.

Predicted posterior probabilities. Plot results where color - class. Since we have the same splits, we can use the same n, p, colors.

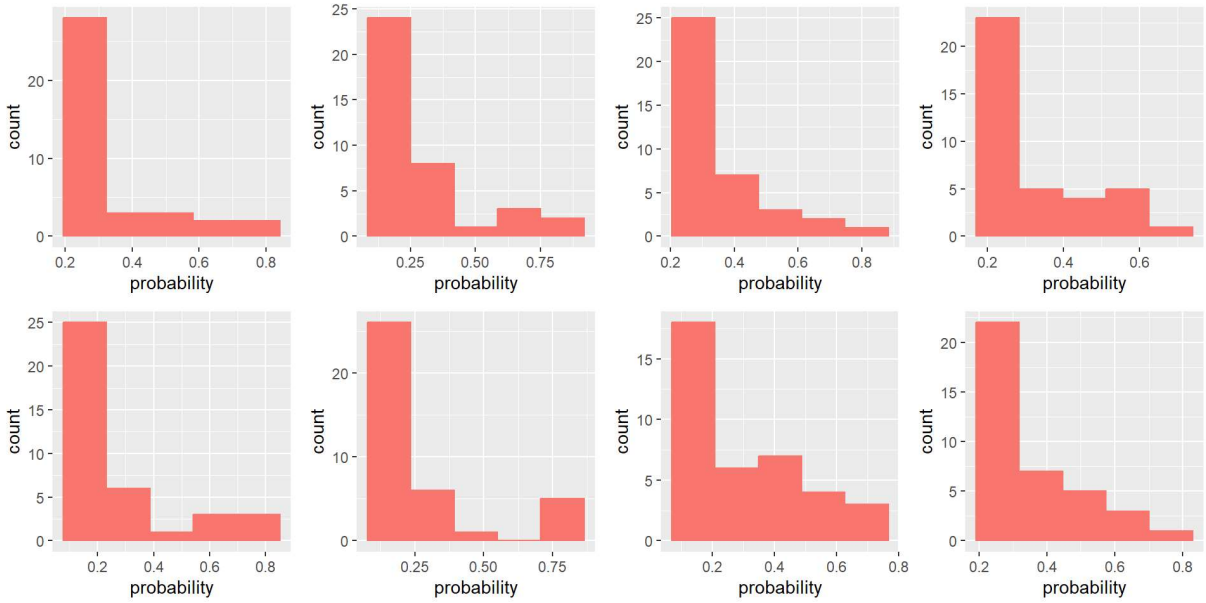
The results are not so good, because there are false predictions in all models and many probabilities in the middle.

```
pred.probs1 <- predict(boost1, newdata=test1, type = "prob")
pred.probs2 <- predict(boost2, newdata=test2, type = "prob")
pred.probs3 <- predict(boost3, newdata=test3, type = "prob")
pred.probs4 <- predict(boost4, newdata=test0, type = "prob")

pred.probs5 <- predict(boost5, newdata=test1, type = "prob")
pred.probs6 <- predict(boost6, newdata=test2, type = "prob")
pred.probs7 <- predict(boost7, newdata=test3, type = "prob")
pred.probs8 <- predict(boost8, newdata=test0, type = "prob")
```



Let's also plot histograms of predicted probabilities.

As we saw before, the results are not so good: many values are about 0, but also a lot of values are located in the middle between 0 and 1.



Predicted class labels.

```
pred.labels1 <- predict(boost1, newdata=test1)
pred.labels2 <- predict(boost2, newdata=test2)
pred.labels3 <- predict(boost3, newdata=test3)
pred.labels4 <- predict(boost4, newdata=test0)

pred.labels5 <- predict(boost5, newdata=test1)
pred.labels6 <- predict(boost6, newdata=test2)
pred.labels7 <- predict(boost7, newdata=test3)
pred.labels8 <- predict(boost8, newdata=test0)

real.labels <- test1$Class
```

4.10.2 Comparison

```
boost_statistics %>%
    flextable() %>%
    theme_box() %>%
    autofit()
```

name	boost.1	boost.2	boost.3	boost.4	boost.5	boost.6	boost.7	boost.8
Accuracy	0.89	0.89	0.89	0.92	0.89	0.92	0.87	0.87
Kappa	0.65	0.65	0.61	0.75	0.65	0.72	0.59	0.59
Sensitivity	0.62	0.62	0.50	0.75	0.62	0.62	0.62	0.62
Specificity	0.97	0.97	1.00	0.97	0.97	1.00	0.93	0.93

name	boost.1	boost.2	boost.3	boost.4	boost.5	boost.6	boost.7	boost.8
F1	0.71	0.71	0.67	0.80	0.71	0.77	0.67	0.67
Balanced Accuracy	0.80	0.80	0.75	0.86	0.80	0.81	0.78	0.78
10-fold CV error	0.15	0.11	0.17	0.16	0.14	0.11	0.18	0.16
Bootstrap error	0.16	0.12	0.16	0.18	0.15	0.13	0.16	0.18

So, the best Boosting model when comparing the statistics:

- Accuracy: 4, 6
- Kappa: 4 -> 6
- Sensitivity: 4 -> 1, 2, 5, 6, 7, 8
- Specificity: 3, 6 -> 1, 2, 4, 5
- F1: 4 -> 6
- Balanced Accuracy: 4 -> 6
- 10-fold CV: 2, 6
- Bootstrap: 2 -> 6

To sum up, the best Boosting model is the **6th** one: model without variables that are useless from EDA and bag tree imputing method.

4.11 Random forest

4.11.1 Method

We can use both numeric and categorical data. Let's specify model formula. We use all attributes and also model without variables that are useless from EDA. Also we need number of features.

```
mod1 <- Class ~ . - Class
mod2 <- Class ~ . - Class - Age - Steroid - AlkPhosphate - Sgot

p1 <-  ncol(test1) - 1
p2 <-  ncol(test1) - 5
```

Building random forest for different models, parameters (ntree - number of trees, mtry - number of randomly selected features), and datasets (for different imputation methods).

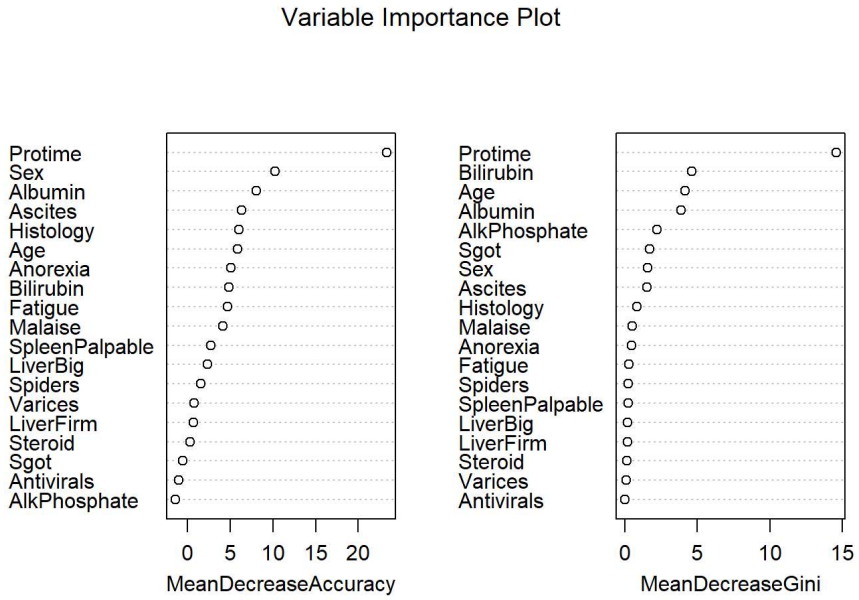
```
set.seed(123)

rf1 <- randomForest(mod1, data=train1, ntree=500, mtry=p1, importance=TRUE)
rf2 <- randomForest(mod1, data=train2, ntree=500, mtry=p1, importance=TRUE)
rf3 <- randomForest(mod1, data=train3, ntree=500, mtry=p1, importance=TRUE)

rf4 <- randomForest(mod1, data=train1, ntree=500, mtry=sqrt(p1), importance=TRUE)
rf5 <- randomForest(mod1, data=train2, ntree=500, mtry=sqrt(p1), importance=TRUE)
rf6 <- randomForest(mod1, data=train3, ntree=500, mtry=sqrt(p1), importance=TRUE)

rf7 <- randomForest(mod2, data=train1, ntree=500, mtry=p2, importance=TRUE)
rf8 <- randomForest(mod2, data=train2, ntree=500, mtry=p2, importance=TRUE)
rf9 <- randomForest(mod2, data=train3, ntree=500, mtry=p2, importance=TRUE)

rf10 <- randomForest(mod2, data=train1, ntree=500, mtry=sqrt(p2), importance=TRUE)
rf11 <- randomForest(mod2, data=train2, ntree=500, mtry=sqrt(p2), importance=TRUE)
rf12 <- randomForest(mod2, data=train3, ntree=500, mtry=sqrt(p2), importance=TRUE)
```

In the additional file, we can find basic information and classification error plot for the random forests.

Now, let's create a plot that displays the importance of each predictor variable in the final model. The x-axis displays the average increase in node purity of the regression trees based on splitting on the various predictors displayed on the y-axis. From the plot we can see that Protime is the most important predictor variable.

In the additional file, we can see variable importance ranking for other random forest models. The results are different, but for each model, Protime is in the first place.

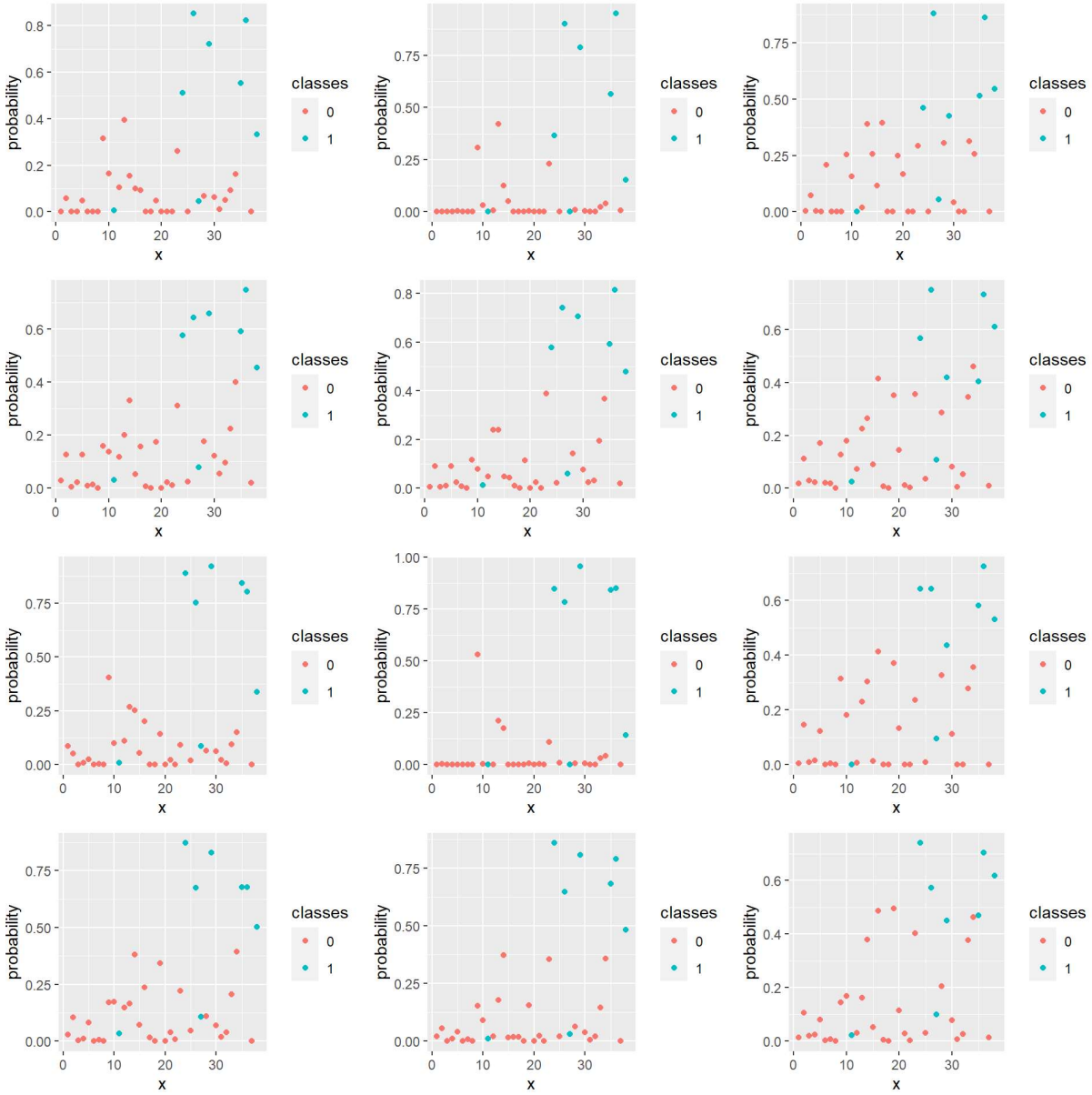
```
varImpPlot(rf1, main = "Variable Importance Plot")
```



Predicted posterior probabilities. Plot results where color - class. Since we have the same splits, we can use the same n, p, colors.

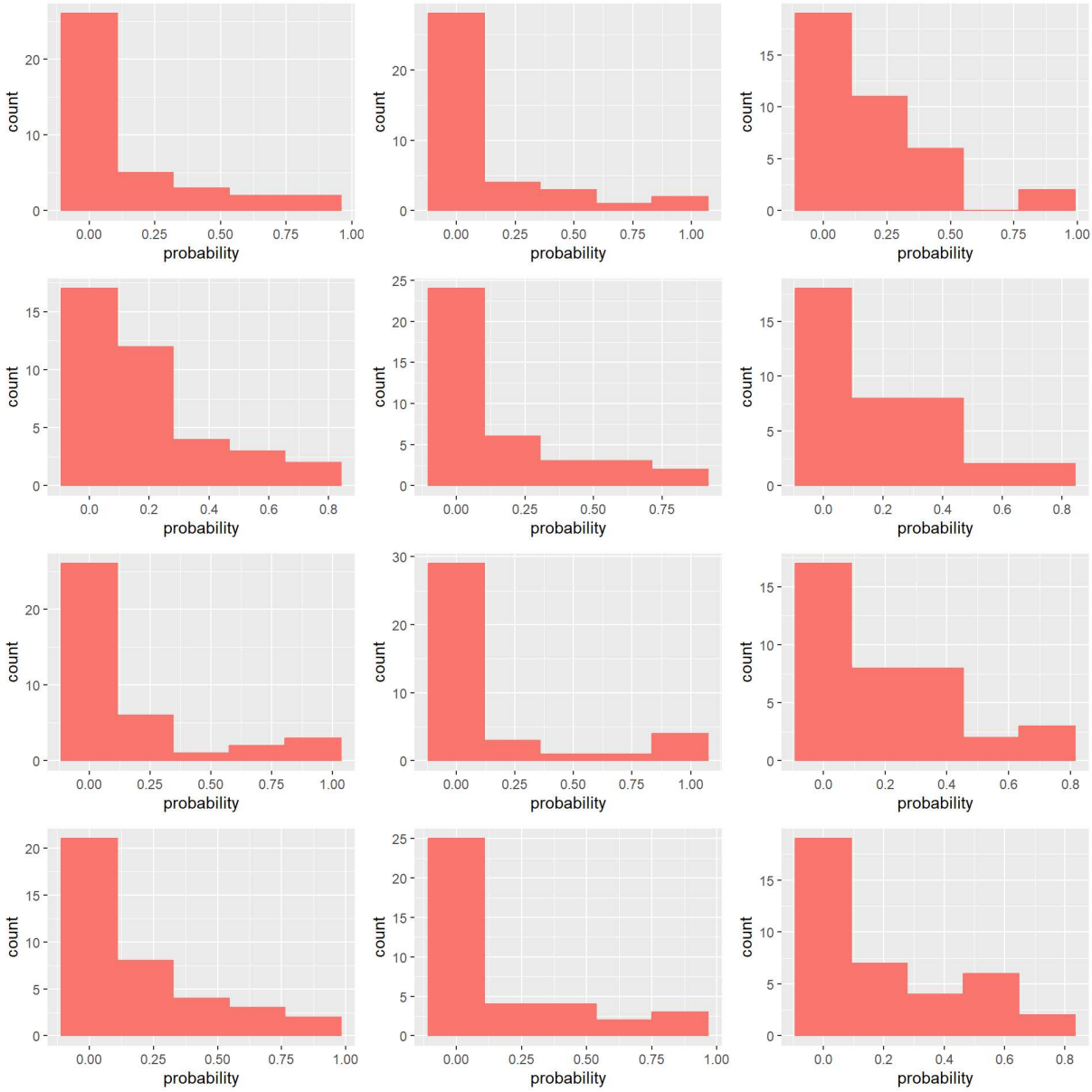
The results are not so good, because there are false predictions in all models and many probabilities in the middle.

```
pred.probs1 <- predict(rf1, newdata=test1, type = "prob")
pred.probs2 <- predict(rf2, newdata=test2, type = "prob")
pred.probs3 <- predict(rf3, newdata=test3, type = "prob")

pred.probs4 <- predict(rf4, newdata=test1, type = "prob")
pred.probs5 <- predict(rf5, newdata=test2, type = "prob")
pred.probs6 <- predict(rf6, newdata=test3, type = "prob")

pred.probs7 <- predict(rf7, newdata=test1, type = "prob")
pred.probs8 <- predict(rf8, newdata=test2, type = "prob")
pred.probs9 <- predict(rf9, newdata=test3, type = "prob")

pred.probs10 <- predict(rf10, newdata=test1, type = "prob")
pred.probs11 <- predict(rf11, newdata=test2, type = "prob")
pred.probs12 <- predict(rf12, newdata=test3, type = "prob")
```



Let's also plot histograms of predicted probabilities.

As we saw before, the results are not so good: many values are about 0, but also a lot of values are located in the middle between 0 and 1.



Predicted class labels.

```
pred.labels1 <- predict(rf1, newdata=test1, type = "class")
pred.labels2 <- predict(rf2, newdata=test2, type = "class")
pred.labels3 <- predict(rf3, newdata=test3, type = "class")

pred.labels4 <- predict(rf4, newdata=test1, type = "class")
pred.labels5 <- predict(rf5, newdata=test2, type = "class")
pred.labels6 <- predict(rf6, newdata=test3, type = "class")

pred.labels7 <- predict(rf7, newdata=test1, type = "class")
pred.labels8 <- predict(rf8, newdata=test2, type = "class")
pred.labels9 <- predict(rf9, newdata=test3, type = "class")

pred.labels10 <- predict(rf10, newdata=test1, type = "class")
pred.labels11 <- predict(rf11, newdata=test2, type = "class")
pred.labels12 <- predict(rf12, newdata=test3, type = "class")

real.labels <- test1$Class
```

4.11.2 Comparison

```
rf_statistics %>%
    flextable() %>%
    theme_box() %>%
    autofit()
```

name	rf.1	rf.2	rf.3	rf.4	rf.5	rf.6	rf.7	rf.8	rf.9	rf.10	rf.11	rf.12
Accuracy	0.92	0.89	0.89	0.92	0.92	0.89	0.92	0.89	0.92	0.95	0.92	0.89
Kappa	0.72	0.61	0.61	0.72	0.72	0.61	0.72	0.65	0.72	0.83	0.72	0.61
Sensitivity	0.62	0.50	0.50	0.62	0.62	0.50	0.62	0.62	0.62	0.75	0.62	0.50

name	rf.1	rf.2	rf.3	rf.4	rf.5	rf.6	rf.7	rf.8	rf.9	rf.10	rf.11	rf.12
Specificity	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.97	1.00	1.00	1.00	1.00
F1	0.77	0.67	0.67	0.77	0.77	0.67	0.77	0.71	0.77	0.86	0.77	0.67
Balanced Accuracy	0.81	0.75	0.75	0.81	0.81	0.75	0.81	0.80	0.81	0.88	0.81	0.75
10-fold CV error	0.12	0.11	0.17	0.13	0.13	0.16	0.12	0.12	0.16	0.11	0.10	0.15
Bootstrap error	0.16	0.13	0.18	0.13	0.13	0.14	0.14	0.13	0.17	0.12	0.11	0.15

So, the best Random forest model when comparing the statistics:

- Accuracy: 10
- Kappa: 10
- Sensitivity: 10
- Specificity: 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12
- F1: 10
- Balanced Accuracy: 10
- 10-fold CV: 11 -> 2, 10
- Bootstrap: 11 -> 10

To sum up, the best Random forest model is the **10th** one: model without variables that are useless from EDA, knn imputing method, and parameter mtry=sqrt(p).

4.12 Conclusions

The best models by methods:

```
all_statistics[1:5] %>%
    flextable() %>%
    theme_box() %>%
    autofit()
```

metric	Linear regression	LDA	QDA	Logistic regression
Accuracy	0.92	0.87	0.87	0.92
Kappa	0.75	0.65	0.62	0.75
Sensitivity	0.75	0.88	0.75	0.75
Specificity	0.97	0.87	0.90	0.97
F1	0.80	0.74	0.71	0.80
Balanced Accuracy	0.86	0.87	0.82	0.86
10-fold CV error				0.35
Bootstrap error				0.36

```
all_statistics[6:10] %>%
    flextable() %>%
    theme_box() %>%
    autofit()
```

KNN	Random Tree	Bagging Tree	Boosted Tree	Random Forest
0.79	0.92	0.89	0.92	0.95
0.50	0.72	0.61	0.72	0.83
0.88	0.62	0.50	0.62	0.75
0.77	1.00	1.00	1.00	1.00
0.64	0.77	0.67	0.77	0.86
0.82	0.81	0.75	0.81	0.88
	0.14	0.12	0.11	0.11
	0.15	0.14	0.13	0.12

So, the best model when comparing the statistics:

- Accuracy: Random forest
- Kappa: Random forest
- Sensitivity: KNN, LDA
- Specificity: Random tree, Bagging, Boosting, Random forest
- F1: Random forest

- Balanced Accuracy: Random forest
- 10-fold CV: Boosting, Random forest
- Bootstrap: Random forest

To sum up, the best model is **Random forest** model without variables that are useless from EDA, knn imputing method, and parameter mtry=sqrt(p).