heart_failure

Rob Meulenkamp

2022-09-14

EDA logbook

Introduction

Cardiovascular diseases (CVD) is one of the most common death globally. This makes up for 31% of all deaths worldwide. Heart disease is provoked by atherosclerosis. This means the buildup of plaques or fatty deposits in the walls of the coronary arteries in several years.
The coronary arteries enclose the outside of the heart and provide blood oxygen and nutrients to the heart muscle. If the plaque builds up in the arteries, there is fewer space for blood to flow naturally and deliver oxygen to the heart. This possibly cause angina (chest pain) or a heart attack. Four out of five CVD deaths are result of heart attacks and strokes. One-third of these deaths appear in people under the age of 70.

This data set include 11 features that are viable to use for predicting a heart disease.

Data: https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction?resource=download

Research question

Is it feasible to produce an accurate (False negatives <= 5%) machine learning algorithm that predicts the possibility of developing a heart disease in a wide array of patients?

Read the data

```
heartdata <- read.csv("data/heart.csv")
```

Parsing data

Parsing data to change the type of variables. If someone is diagnosed it's either 0 or 1. This can be confusing sometimes. It's easier to keep track with True and False values.

```
heartdata$HeartDisease <- as.logical(heartdata$HeartDisease)
heartdata$FastingBS <- as.logical(heartdata$FastingBS)



# Changes Y/N to Boolean
heartdata$ExerciseAngina <- heartdata$ExerciseAngina == "Y"


# Function split column into new column and make it True or False
# For example if the patient has the condition yes or no
# x = value in column you want to split
# y = column the value belongs to
# dataframe = dataframe the desired value is stored
```

```r
# Returns the desired column
splitColumn <- function(x, y, dataframe){
  dataframe[,x] <- NA
  dataframe[,x][which(dataframe[,y] == x)] <- x
  dataframe[,x][is.na(dataframe[,x])] <- FALSE
  dataframe[,x][dataframe[,x] == x] <- TRUE
  return(dataframe[,x])

}

value_targets <- c("ATA", "ASY", "NAP", "TA")

for (column in value_targets){
    new_column <- splitColumn(column, "ChestPainType", heartdata)
    frame <- as.data.frame(as.logical(new_column))
    colnames(frame) <- column
    heartdata <- add_column(heartdata, frame)

}
```

Splitting the different conditions is done for ChestPainType otherwise the column can't be used for making Principal component analysis.

Codebook

The data set doesn't include a codebook with annotated header or columns. The codebook consist of the column name, full name, and the type of the data. This codebook is handmade.

```
codebook <- read.csv("data/codebook.csv")
pander(codebook, style = "rmarkdown", caption ="Overview created codebook")
```

Table 1: Overview created codebook

Column.Name	Full.Name	Type
Age	Age	int
Sex	Sex	chr
RestingBP	Resting blood pressure	int
Cholesterol	Cholesterol	num
FastingBS	Fasting blood sugar	logi
RestingECG	Resting electrocardiogram	chr
MaxHR	Maximum heart rate	int
ExerciseAngina	Exercise-induces angina	logi
Oldpeak	ST(numeric values measured in depressiong)	num
ST_Slope	The slope of the peak exercise ST segment	chr
HeartDisease	Heart disease	logi
ATA	Atypical Angina pain	logi
ASY	Asymptomatic pain	logi
NAP	Non-Anginal pain	logi
TA	Typical angina pain	logi

Most of the attributes are categorical instead of numerical data. Extra information about every attribute can be found in the following link. (Attribute information)[https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction?resource=download]

The most important is that values are read correctly and has to be examined.

```
str(heartdata)
```

```
## 'data.frame':    918 obs. of  16 variables:
##  $ Age           : int  40 49 37 48 54 39 45 54 37 48 ...
##  $ Sex           : chr  "M" "F" "M" "F" ...
##  $ ChestPainType : chr  "ATA" "NAP" "ATA" "ASY" ...
##  $ RestingBP     : int  140 160 130 138 150 120 130 110 140 120 ...
##  $ Cholesterol   : int  289 180 283 214 195 339 237 208 207 284 ...
##  $ FastingBS     : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ RestingECG    : chr  "Normal" "Normal" "ST" "Normal" ...
##  $ MaxHR         : int  172 156 98 108 122 170 170 142 130 120 ...
##  $ ExerciseAngina: logi  FALSE FALSE FALSE TRUE FALSE FALSE ...
##  $ Oldpeak       : num  0 1 0 1.5 0 0 0 0 1.5 0 ...
##  $ ST_Slope      : chr  "Up" "Flat" "Up" "Flat" ...
##  $ HeartDisease  : logi  FALSE TRUE FALSE TRUE FALSE FALSE ...
##  $ ATA           : logi  TRUE FALSE TRUE FALSE FALSE FALSE ...
##  $ ASY           : logi  FALSE FALSE FALSE TRUE FALSE FALSE ...
##  $ NAP           : logi  FALSE TRUE FALSE FALSE TRUE TRUE ...
##  $ TA            : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
```

fortunately it looks like every column have been read correctly.

Table 2: 5 number summary of original dataset

	Age	RestingBP	Cholesterol	MaxHR	Oldpeak
Min	28.00	0.0	0.0	60.0	-2.6000
1st. Qu	47.00	120.0	173.2	120.0	0.0000
Median	54.00	130.0	223.0	138.0	0.6000
Mean	53.51	132.4	198.8	136.8	0.8874
3rd. Qu	60.00	140.0	267.0	156.0	1.5000
Max	77.00	200.0	603.0	202.0	6.2000

Inspecting the different columns if they contain Na's.This is done to get a better understanding about the data set.

```
colSums(is.na(heartdata))
```

```
##            Age            Sex  ChestPainType      RestingBP    Cholesterol
##              0              0              0              0              0
##      FastingBS     RestingECG          MaxHR ExerciseAngina        Oldpeak
##              0              0              0              0              0
##       ST_Slope   HeartDisease            ATA            ASY            NAP
##              0              0              0              0              0
##             TA
##              0
```

The data set don't have missing values. Now it's time to inspect the data with a summary.

Summary

```
summ <- summary(heartdata[c("Age", "RestingBP", "Cholesterol", "MaxHR", "Oldpeak")])
tab <- sub('.*:', '', summ)
rownames(tab) <- c("Min", "1st. Qu", "Median", "Mean", "3rd. Qu", "Max")
kable(tab, caption = "5 number summary of original dataset") %>%
  kable_styling(latex_options="scale_down")
```

The first thing that stands out is the 0 value as minimum value in the column RestingBP. If a person has a resting blood pressure of 0 it means that the person is dead. Column cholesterol has 0 values too as minimum value. A person has mostly of the time cholesterol level higher than 0. This needs to be investigate carefully when the data is cleaned before using the machine learning algorithm.

Visualisations

Histogram Age vs Heart disease

```
ggplot(heartdata, aes(Age, fill=HeartDisease)) +
  geom_histogram(bins=10) +
  ylab("Frequency") +
  ggtitle("Frequency heart disease for age")
```

Figure 1: Histogram of age against frequency for HD

In the graph above, there is possibly a correlation between age and heart disease. When the age increases the frequency of heart diseases do as well. This needs more investigation.

Age linked to heartdisease

```
ggplot(heartdata) +
  geom_smooth(aes(Age, HeartDisease * 100), method="loess", formula = "y ~ x", color=hue_pal()(1)) +
  ylab("Heart Disease (%)") +
  xlab("Age (years)") +
  ggtitle("Percentage of patients affected by heart disease") +
  ylim(c(0, 100)) +
  xlim(c(25, 80))
```

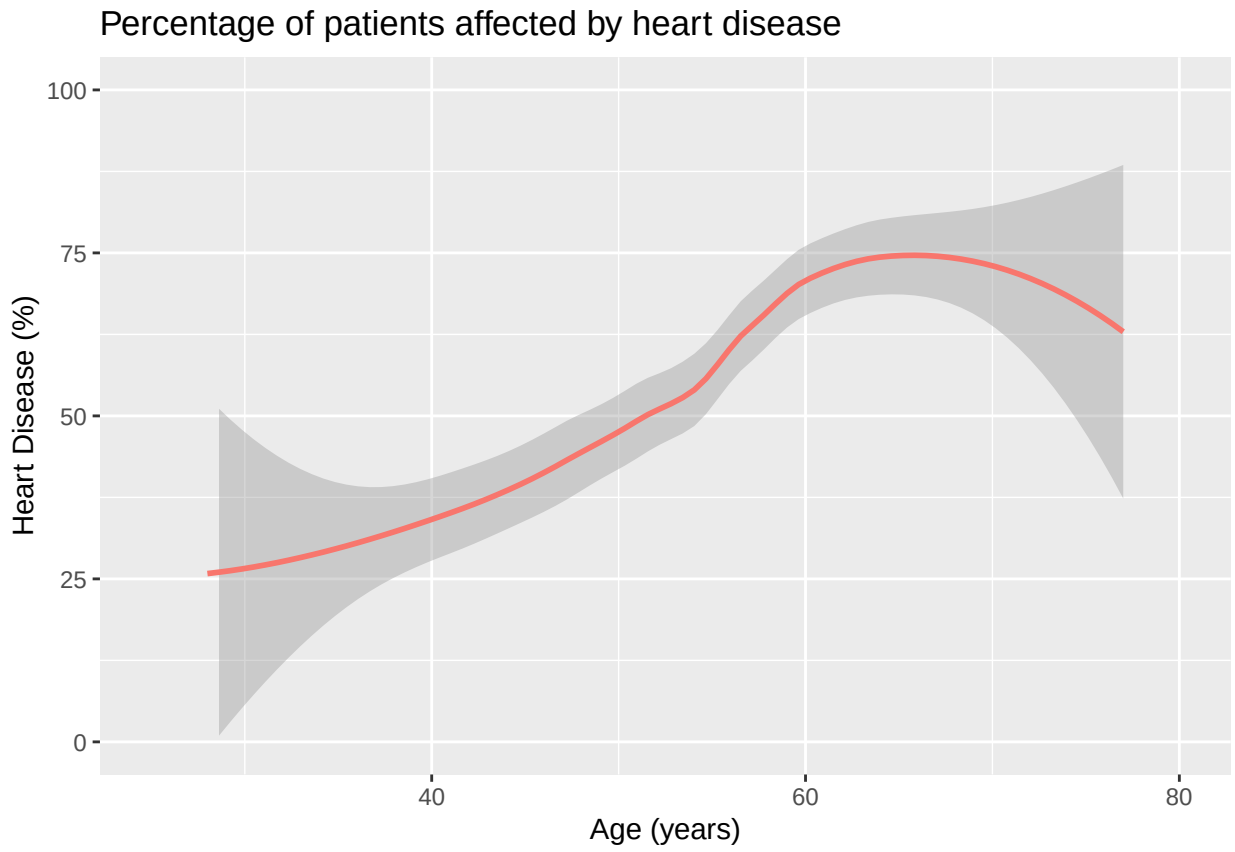


Figure 2: Percentage of patients impacted by HD

With Smooth line it is possible to see that the percentage of patients increases with age and heart disease. The only thing is that after the age of 65 there is starting a decline.
The reason for the decline is perhaps death for the people above 65 who have a heart disease.

For people above the age of 65 are more likely to develop a heart disease which can cause heart attacks, strokes or heart failures. This is potentially the cause of the decline. Source: Heart Health and Aging

Ratio between males and females

```
ggplot(heartdata) +
  geom_bar(aes(Sex, fill=HeartDisease)) +
  labs(title="Man vs Women having heartdisease") +
  ylab("frequency")
```

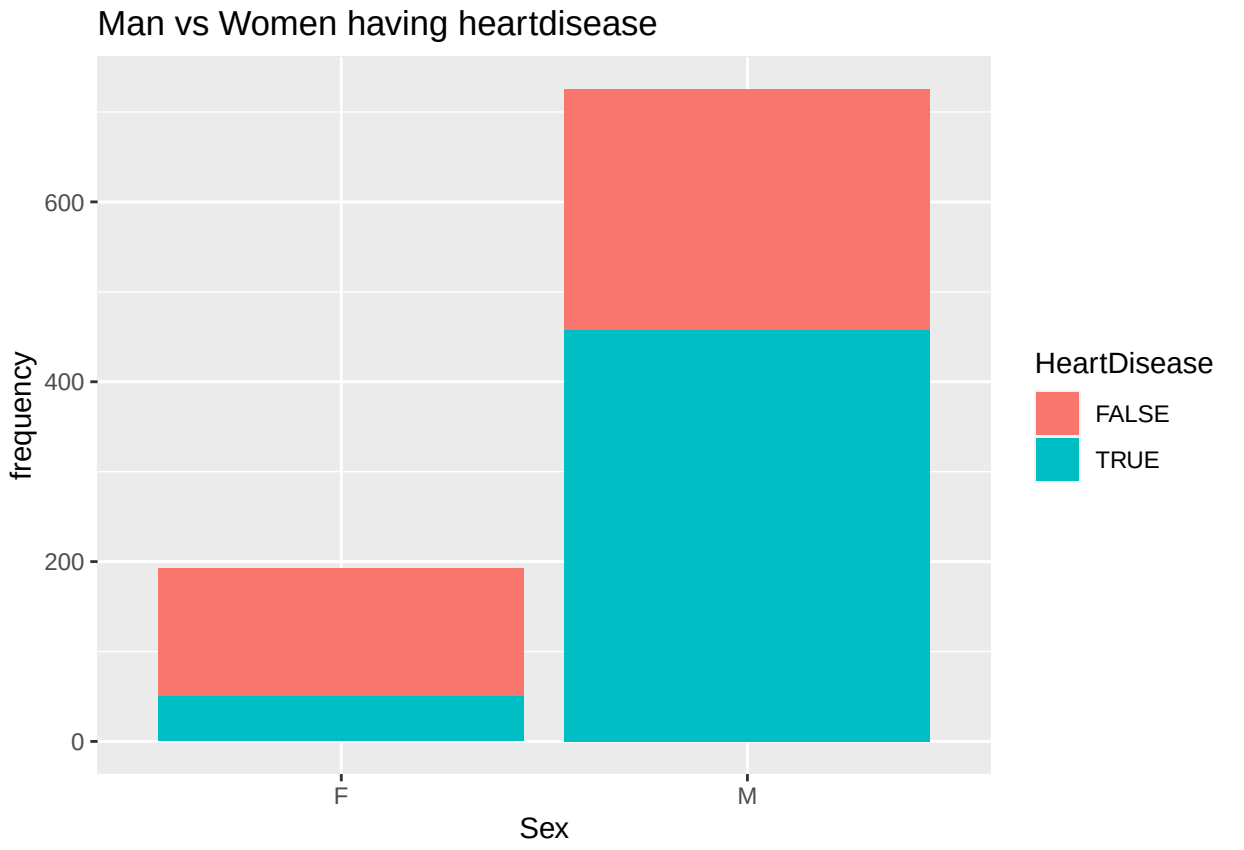


Figure 3: Ratio Male - Female

The barplot shows the ratio between male and female. The ratio is more skewed towards males. The most noticeable is that males have a higher proportion to have heartdisease in comparison with the females. For the female, it is more likely to not have a heartdisease.

Chest pain against heart disease

```
ggplot(heartdata) +
  geom_bar(aes(ChestPainType, fill=HeartDisease)) + ggtitle("Chestpaintype vs heart disease")   +
  ylab("Frequency")
```

In the barplot above, Asymptomatic pain (ASY) is the most frequent chest pain type for people with heart disease.

Figure 4: proportions between chest pain with or without heart disease

Effect of chest pain type against the resting blood pressure

```
ggplot(heartdata, aes(x=RestingBP, group=ChestPainType, colour=ChestPainType)) +
  geom_density()
```

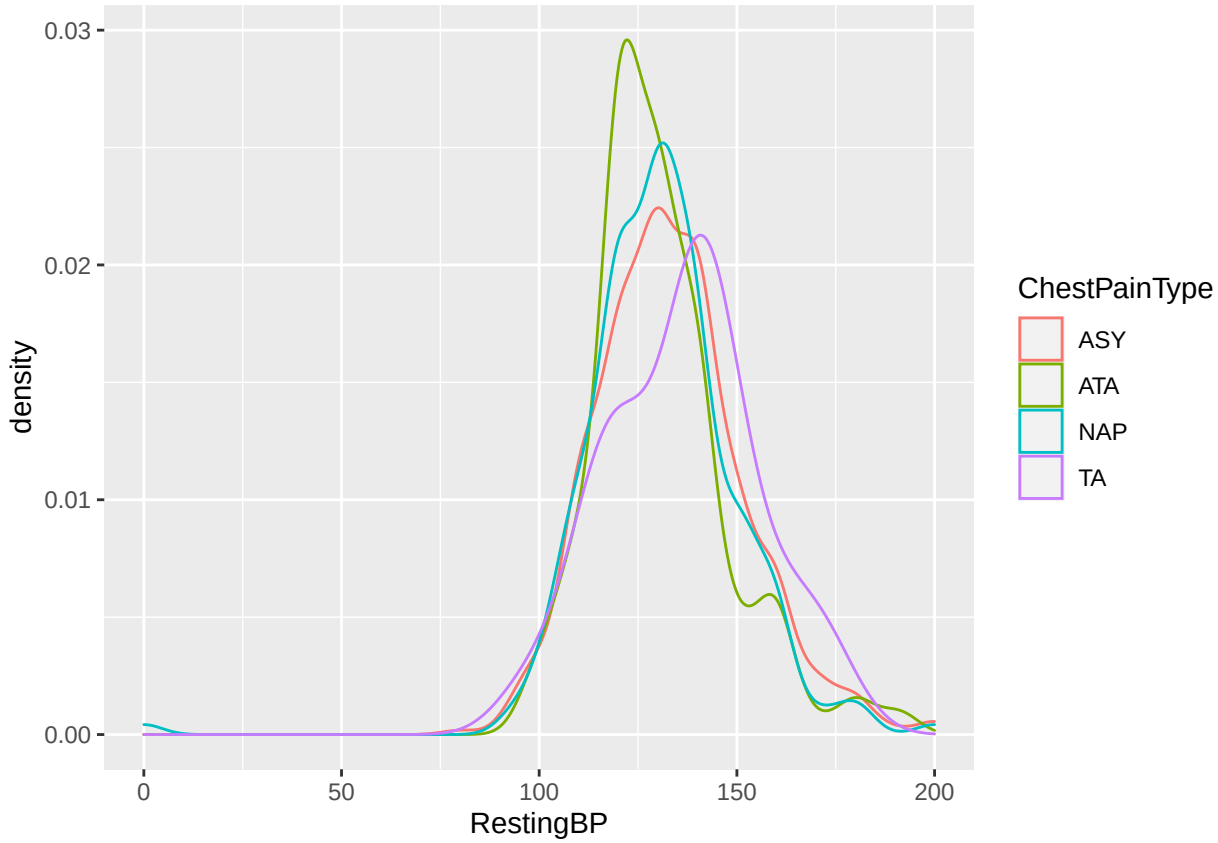


Figure 5: Density plot Resting blood pressue against chest pain

The pain type doesn't seem to have an effect on the blood pressure of the individual.

influence of Electrocardiogram (ECG) against heart disease

ECG type:

Code	Explanation
Normal	Normal
ST	Having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV).
LVH	Showing probable or definite left ventricular hypertrophy by Estes' criteria.

Electrocardiogram (ECG or EKG) is a straightforward test that records and detects your heart's electrical activity. An ECG shows how fast your heart is beating and shows if your rhythm of your heartbeats is steady or irregular. But also indicates the strength and timing of the electical impulses passing through each part of your hart. Source: Electrocardiogram

```
ggplot(heartdata) +
  geom_bar(aes(RestingECG, fill=HeartDisease)) + ggtitle("resting electrocardiogram vs heart disease")
  ylab("Frequency")
```

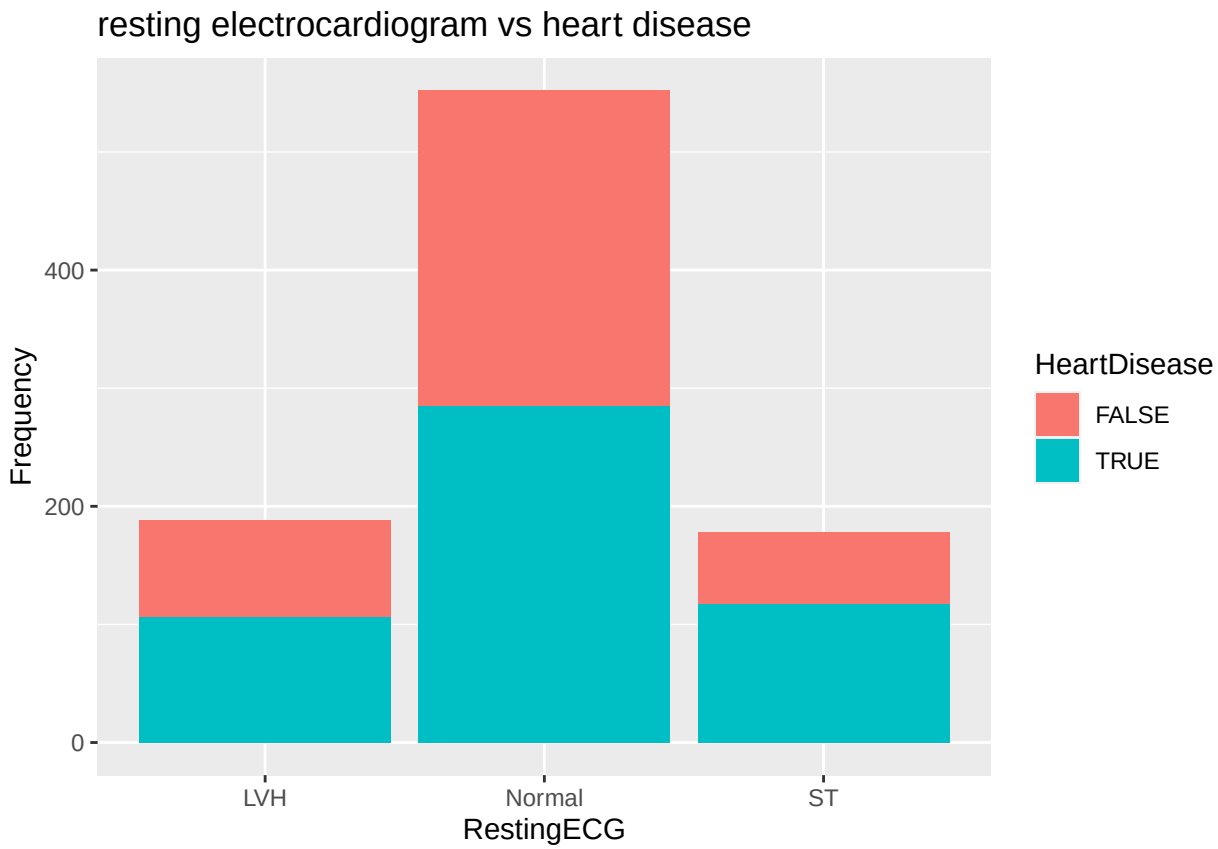


Figure 6: Frequency different ECG types against heart disease

```
freq_RestingECG <- table(heartdata$RestingECG)
```

The frequency for a normal (552) electrical activity is exceptionally high in comparison with ST (178) and LVH (188). The ratio between people with heart disease and without heart disease are almost even with a normal electrical activity. The restingECG column probably won't be the deciding factor in predicting people with heart disease. This needs more investigation with principal component analysis in order to determine the value of the restingECG column and other columns as well.

Comparison between MaxHR against age grouped by sex

```
ggplot(heartdata, aes(Age, MaxHR, group=Sex, colour = Sex)) +
  geom_point(alpha=0.10) + geom_smooth(se=F) + ggtitle("Comparison between MaxHR and age grouped by sex
```

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

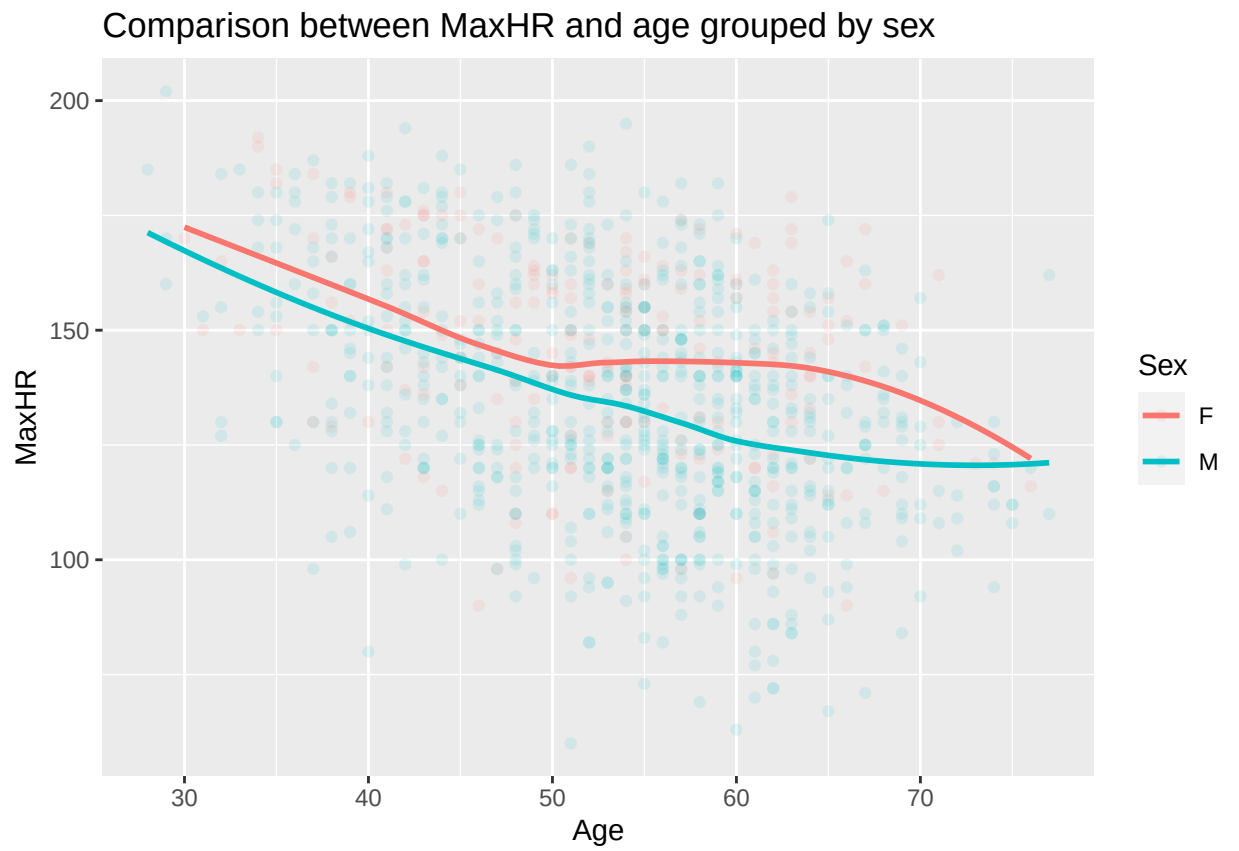


Figure 7: Maximum heart rate against age and grouped by sex

```
count_table <- heartdata %>% group_by(Sex) %>%
      filter(Age > 55) %>%
      count(Sex)
```

In the graph above goes MaxHR down when people become older. Only the line goes up again after the age of 55 for females. This is possible because of the lack of data for females after the age of 55. Females contain 71 amount of data and for males this is 335 data points. The ratio is not balanced. To analyse this a density plot is made for the difference in datapoints per sex.

Density of age grouped by sex

```
ggplot(heartdata) +
  geom_density(aes(Age, colour=Sex, fill=Sex), alpha=0.05) +
  ggtitle("Density plot comparing sexes")
```

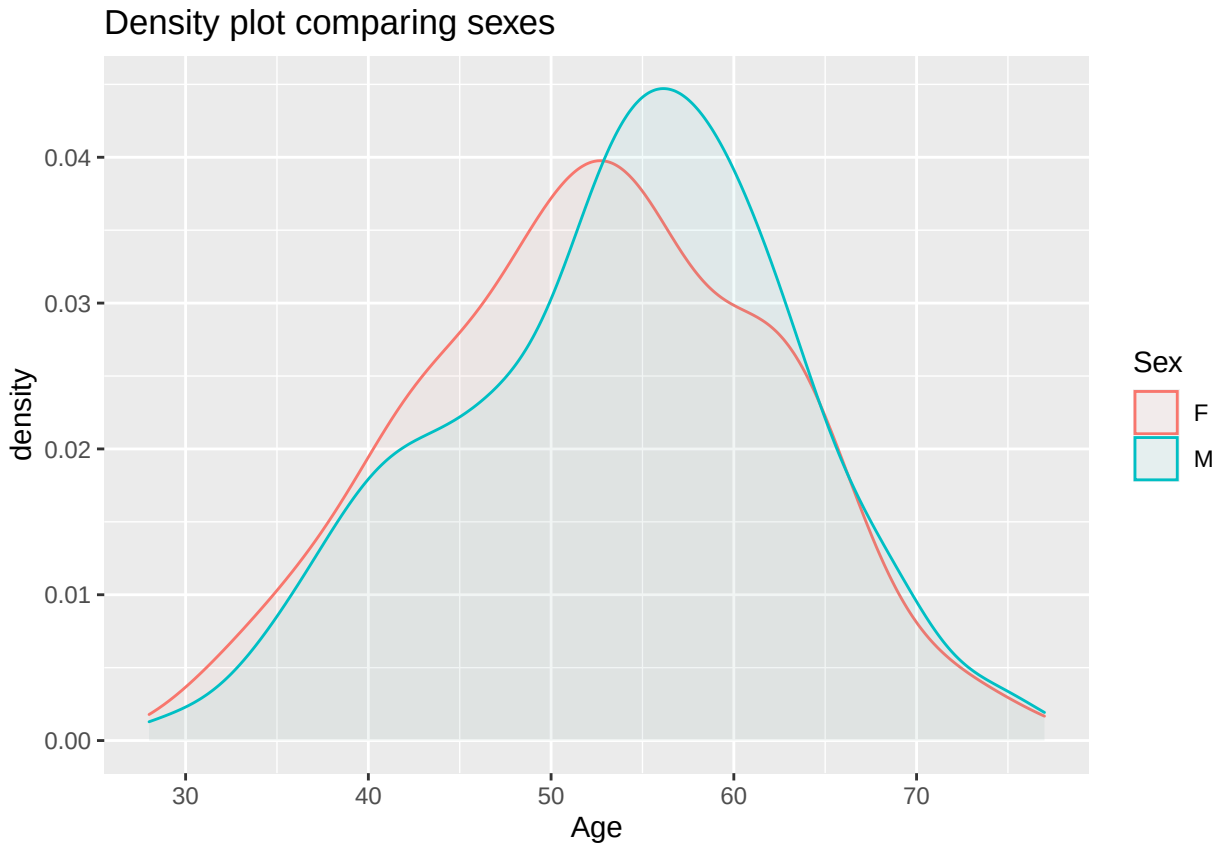


Figure 8: Amount of datapoints per age grouped by sex

The above graph shows that there is not much difference in age comparing male and female.

Principal component analysis

```
heartdataPca <- heartdata[, c(1,4,5,6,8, 9, 13:16)]
res.pca <- prcomp(heartdataPca, scale = TRUE)


fviz_pca_var(res.pca,
             col.var = "cos2",
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE
             )
```

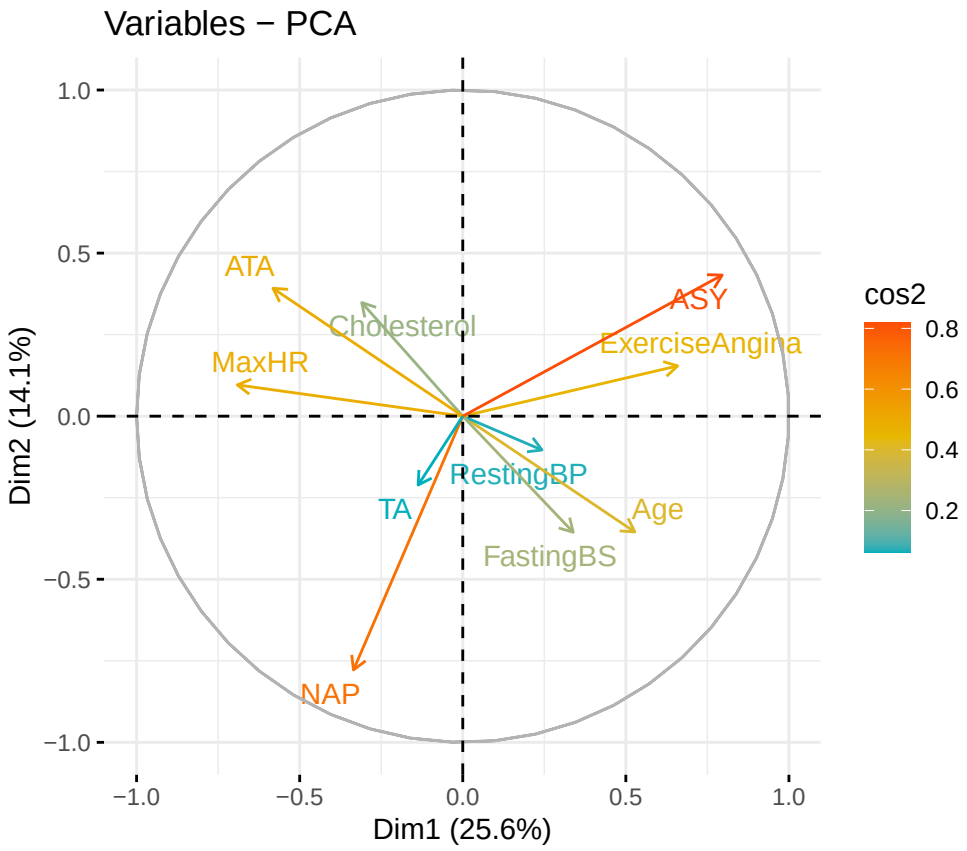


Figure 9: PCA variable plot

```
# Contributions of variables to PC1
fviz_contrib(res.pca, choice = "var", axes = 1, top = 10)
```

```
# Contributions of variables to PC2
fviz_contrib(res.pca, choice = "var", axes = 2, top = 10)
```

In the circle diagram and in the contribution diagram the asymptomatic pain (ASY) and non-Anginal pain (NAP) has the biggest contribution for a positive or negative diagnosing the heart disease. One downside is that it's hard to predict for patients who don't have visible symptoms. Two other important contributors are MaxHR and Age. These two variables are important because it could be measured for every patient thus are viable. Last, the column ExerciseAngina has an important contribution. If it's possible to diagnose the patient with exercise angina this column becomes vital for predicting people with heart disease.

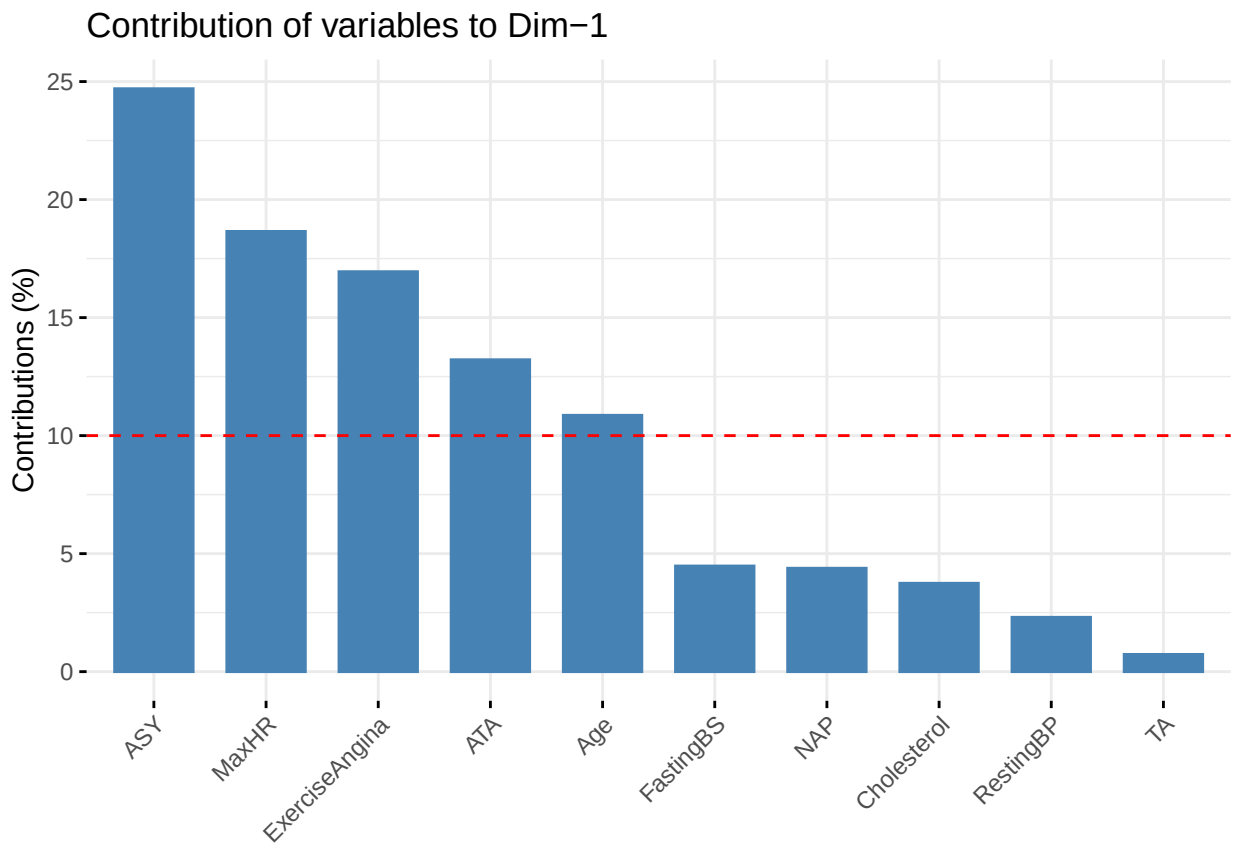


Figure 10: PCA variable plot

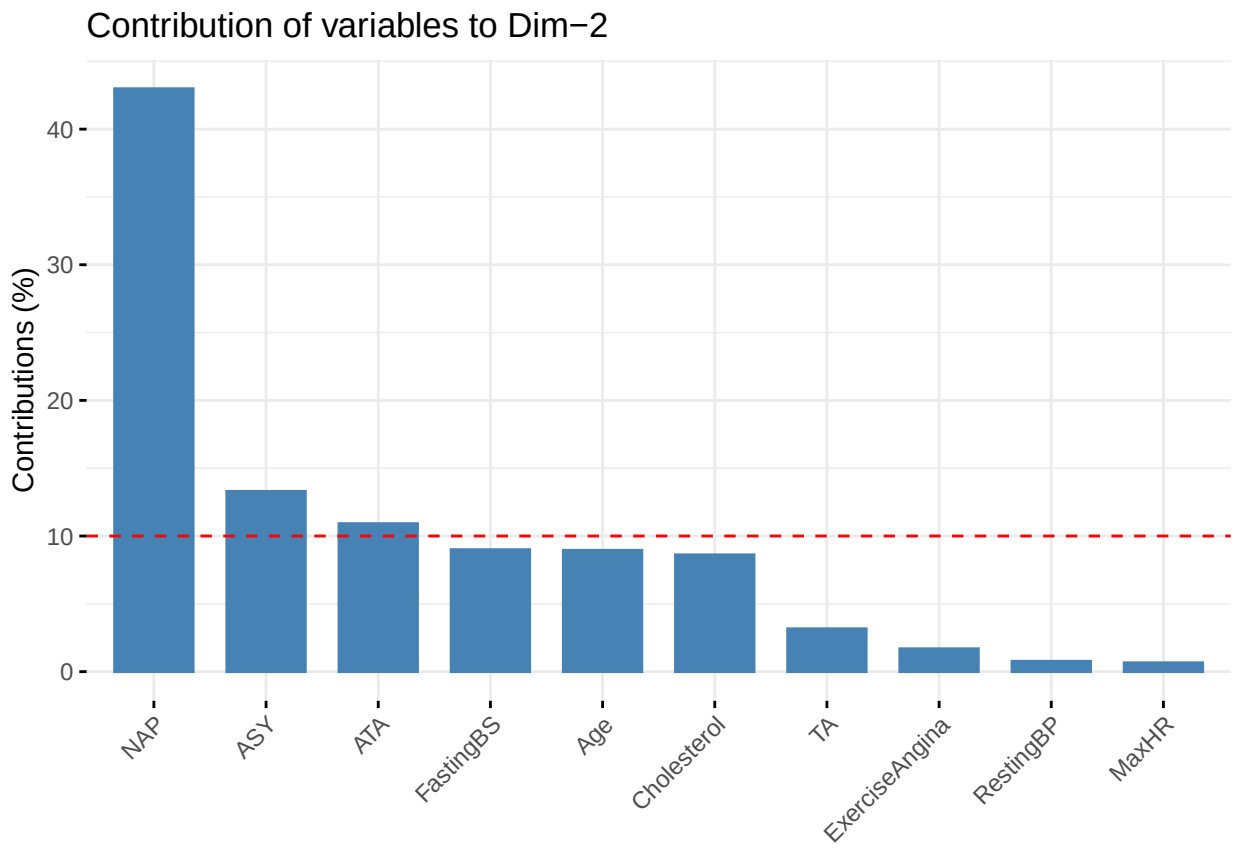


Figure 11: PCA variable plot

Cleaning

Cholesterol 0-values are replaced with the median of the cholesterol column due to missing measurements. The column RestingBP has one measurement with 0 and is replaced with the median as well. Median is less sensitive to outliers in comparison with the mean.

```
heartdata$Cholesterol[heartdata$Cholesterol == 0] <- NA
heartdata$RestingBP[heartdata$RestingBP == 0] <- NA



heartdata$ChestPainType <- NULL
# replace NA's with the median for every column
for(i in 1:ncol(heartdata)){
  heartdata[is.na(heartdata[,i]), i] <- median(heartdata[,i], na.rm = TRUE)
}
```

```
## Warning in mean.default(sort(x, partial = half + 0L:1L)[half + 0L:1L]): argument
## is not numeric or logical: returning NA

## Warning in mean.default(sort(x, partial = half + 0L:1L)[half + 0L:1L]): argument
## is not numeric or logical: returning NA

## Warning in mean.default(sort(x, partial = half + 0L:1L)[half + 0L:1L]): argument
## is not numeric or logical: returning NA
```

```
heartdata <- heartdata %>% relocate(HeartDisease, .after = last_col())
write.csv(heartdata, "data/clean_heart_.csv", row.names = FALSE)
```

Table 4: Used algorithms with respective Weka parameters

Algorithm	WekaParameter
ZeroR	rules.ZeroR '' 48055541465867954
OneR	rules.OneR '-B 6' -3459427003147861443
J48	trees.J48 '-C 0.25 -M 2' -217733168393644444
RandomForestTree	trees.RandomForest '-P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1' 1116839470751428698
AdaBoostM1	meta.AdaBoostM1 '-P 100 -S 1 -I 10 -W trees.DecisionStump' -1178107808933117974
IBK	lazy.IBk '-K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.EuclideanDistance -R first-last\"" -3080186098777067172
SMO	functions.SMO '-C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K "functions.supportVector.PolyKernel -E 1.0 -C 250007" -calibrator "functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4"' -6585883636378691736
SGD	functions.SGD '-F 0 -L 0.01 -R 1.0E-4 -E 500 -C 0.001 -S 1' -3732968666673530290
NaiveBayes	bayes.NaiveBayes '' 5995231201785697655

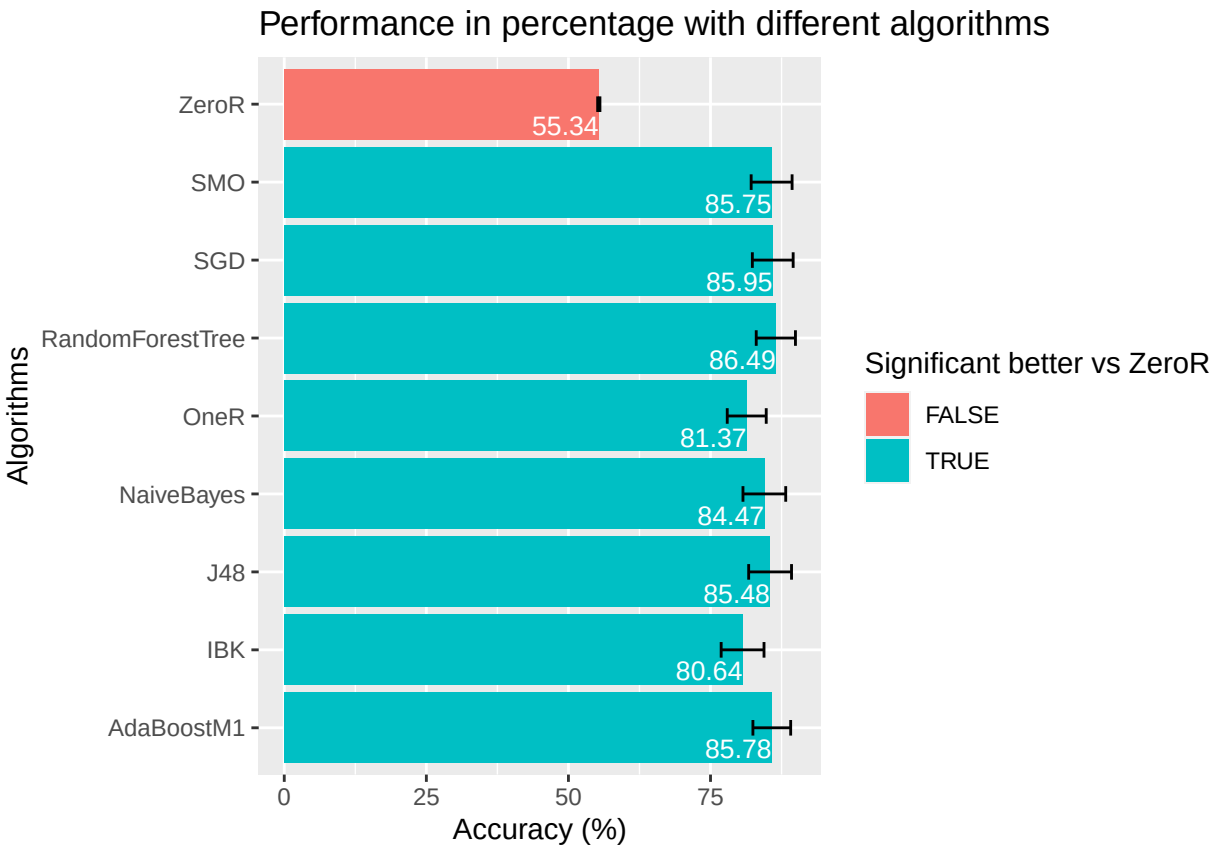
Machine Learning

Investigation is needed to determine at least the two best machine learning algorithms. ZeroR and OneR are used as baseline.

```
Algorithms <- c("ZeroR", "OneR", "J48", "RandomForestTree", "AdaBoostM1", "IBK", "SMO", "SGD", "NaiveBay

Params <- c("rules.ZeroR '' 48055541465867954", "rules.OneR '-B 6' -3459427003147861443", "trees.J48 '-C

Accuracy <- c(55.34,  81.37, 85.48,   86.49,   85.78,   80.64,  85.75,   85.95,  84.47)
se <- c(0.20, 3.43, 3.77, 3.45, 3.32, 3.77, 3.60, 3.59, 3.76)
sig_zero <- c(F, rep(T, 8))

used_algo <- data.frame(Algorithm = Algorithms, WekaParameter = Params)


kable(used_algo, caption = "Used algorithms with respective Weka parameters") %>%
  kable_styling(latex_options="scale_down")
```

Performance overview is made to display how well the algorithms performed and if the algorithm is significant better in comparison to ZeroR.

```
ggplot(used_algo, aes(Accuracy, Algorithms)) +
    geom_col(aes(fill = sig_zero)) +
    geom_text(aes(label=Accuracy), vjust=1.6, color="white", size=3.5,
              position = position_dodge(width = 1), hjust = 1) +
    xlab("Accuracy (%)") +
    ggtitle("Performance in percentage with different algorithms") +
    scale_fill_discrete(name = "Significant better vs ZeroR") +
    geom_errorbarh(aes(xmax = Accuracy + se, xmin = Accuracy - se, height = .2))
```



In the graph above, every algorithm is significant better in comparison with ZeroR. This indicates that the algorithm doesn't predict by chance. RandomForestTree and SGD are one of the best two performing algorithms in comparison with other algorithms. For every algorithm except ZeroR there is a certain overlap between the standard deviation error barplots (black sticks). This probably indicates that the difference is not statistically significant between the algorithms. Statistical test is needed to investigate the statistical significance. The standard deviation error bars are relative small and it suggest that the spread of the data are clumped around the mean.

Table 5: Meta-learners for extra exploration

Algorithm	WekaParameter
ZeroR	rules.ZeroR '' 48055541465867954
RandomForestTree	trees.RandomForest '-P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1' 1116839470751428698
J48	trees.J48 '-C 0.25 -M 2' -217733168393644444
NaiveBayes	bayes.NaiveBayes '' 5995231201785697655
AdaBoostM1	meta.AdaBoostM1 '-P 100 -S 1 -I 10 -W trees.DecisionStump' -1178107808933117974
SMO	functions.SMO '-C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K "functions.supportVector.PolyKernel -E 1.0 -C 250007" -calibrator "functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4"' -6585883636378691736
Vote	meta.Vote '-S 1 -B "trees.J48 -C 0.25 -M 2" -B "trees.RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1" -B "bayes.NaiveBayes " -B "meta.AdaBoostM1 -P 100 -S 1 -I 10 -W trees.DecisionStump" -B "functions.SMO -C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K \"functions.supportVector.PolyKernel -E 1.0 -C 250007\" -calibrator \"functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4\"" -R AVG' -637891196294399624
Stacking	meta.Stacking '-X 10 -M "meta.AdaBoostM1 -P 100 -S 1 -I 10 -W trees.DecisionStump" -S 1 -num-slots 1 -B "meta.AdaBoostM1 -P 100 -S 1 -I 10 -W trees.DecisionStump" -B "bayes.NaiveBayes " -B "functions.SMO -C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K \"functions.supportVector.PolyKernel -E 1.0 -C 250007\" -calibrator \"functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4\"" -B "trees.J48 -C 0.25 -M 2" -B "trees.RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1"' 5134738557155845452
Bagging	meta.Bagging '-P 100 -S 1 -num-slots 1 -I 10 -W trees.J48 -- -C 0.25 -M 2' -115879962237199703

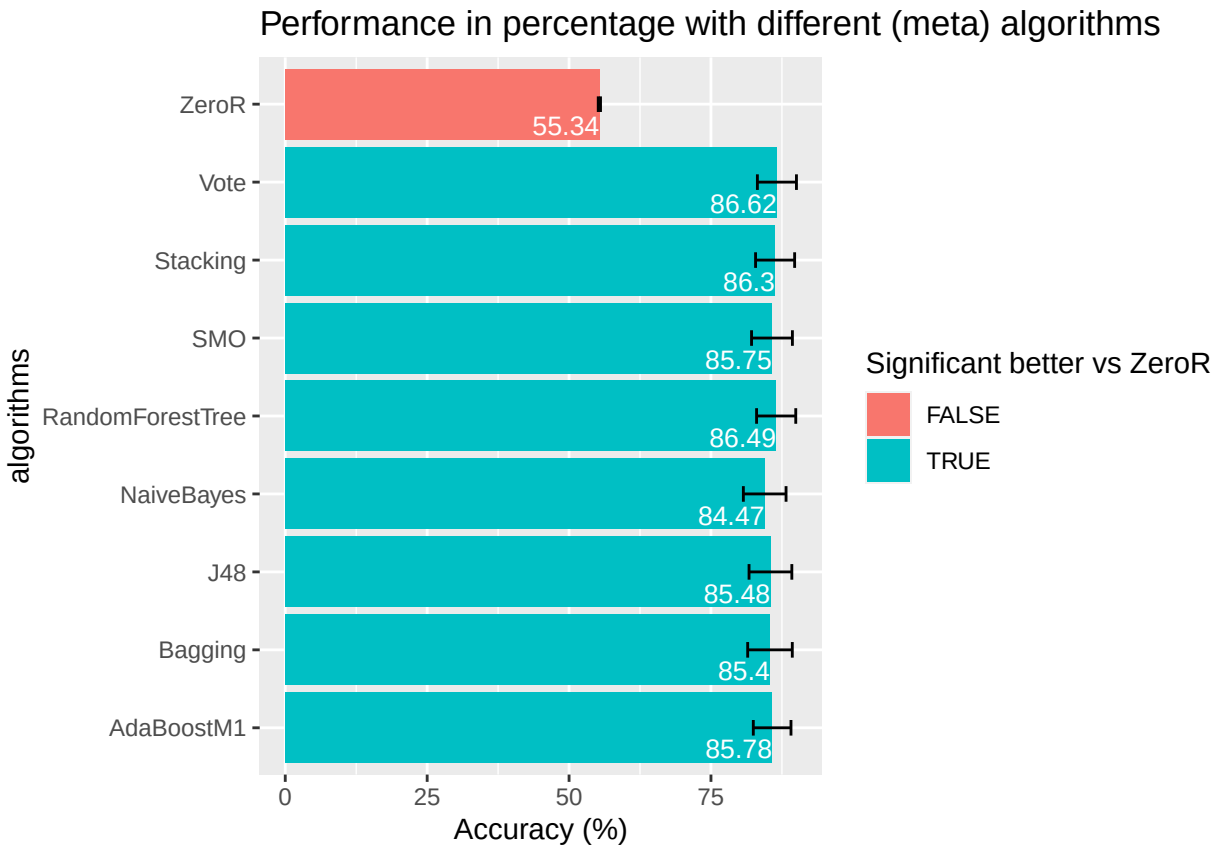
Exploring Meta-learners

Investigating the effect of meta-learners with the goal to improve or explore algorithms. For example Stacking, Bagging, and Voting are used for exploration.

```
algorithms <- c("ZeroR", "RandomForestTree", "J48", "NaiveBayes", "AdaBoostM1", "SMO", "Vote", "Stackin

params <- c("rules.ZeroR '' 48055541465867954", "trees.RandomForest '-P 100 -I 100 -num-slots 1 -K 0 -M

accuracy <- c(55.34, 86.49, 85.48, 84.47, 85.78, 85.75, 86.62, 86.30, 85.40)

Se <- c(0.20, 3.45, 3.77, 3.76, 3.32, 3.60, 3.44, 3.44, 3.93)

meta_algo <- data.frame(Algorithm = algorithms, WekaParameter = params)
kable(meta_algo, caption = "Meta-learners for extra exploration") %>%
  kable_styling(latex_options="scale_down")
```

Overview of the performance for different (meta) algorithms. The plots contain standard deviation error bar. This is done for investigating the spread around the mean or maybe indication about statistically signification between different algorithms. Most important aspect is to find the best performing algorithm which can be used as model.

```
ggplot(meta_algo, aes(accuracy, algorithms)) +
    geom_col(aes(fill = sig_zero)) +
    geom_text(aes(label=accuracy), vjust=1.6, color="white", size=3.5,
              position = position_dodge(width = 1), hjust = 1) +
    xlab("Accuracy (%)") +
    ggtitle("Performance in percentage with different (meta) algorithms") +
    scale_fill_discrete(name = "Significant better vs ZeroR") +
    geom_errorbarh(aes(xmax = accuracy + Se, xmin = accuracy - Se, height = .2))
```



The barplot shows that every algorithm is significant better in comparison with ZeroR. The algorithms Voting, Stacking, and Bagging are added to the graph. Voting and Stacking makes the top three best algorithms with RandomForest. Bagging has a percentage of 85.4 for classifying. It match the accuracy of AdamBoostm1, J48 and SMO. The black sticks are the standard deviation error bars and there is overlap between the algorithms except for ZeroR. The standard deviation error bars are small. This conclude
that the spread of the data are clumped around the mean and the overlap indicate that the difference is not statistically significant between different algorithms. Statistical test needs to be performed to confirm if there is not a statistically significant difference between the algorithms.

ROC and AUC analysis

ROC curve is used as model validation for SGD, AdaBoostm1, and RandomForest, these are on of the best three algorithms left to investigate further. ROC is plotted with the True positive on y-axis and False positive on the x-axis. The goal of ROC analysis is to create an excellent model with AUC value near to the one. The worst possible model has a AUC value near zero and when the AUC value is equal to 0.5 the model has no idea about the class separation for example ZeroR.

```
makePlots <- function(filePath) {
  file <- paste0(filePath, list.files(filePath))
  plot <- lapply(file, rocPlot)
  print(file)

  return(plot)
}


rocPlot <- function(y){
  rocLoad <- read.arff(y)
  title <- file_path_sans_ext(basename(y))
  plot <- ggplot(rocLoad) +
   geom_path(aes(x = `False Positive Rate`, y=`True Positive Rate`, color = Threshold), size=2) + scale_
  return(plot) }

ROCres <- makePlots("data/ROC/")
```

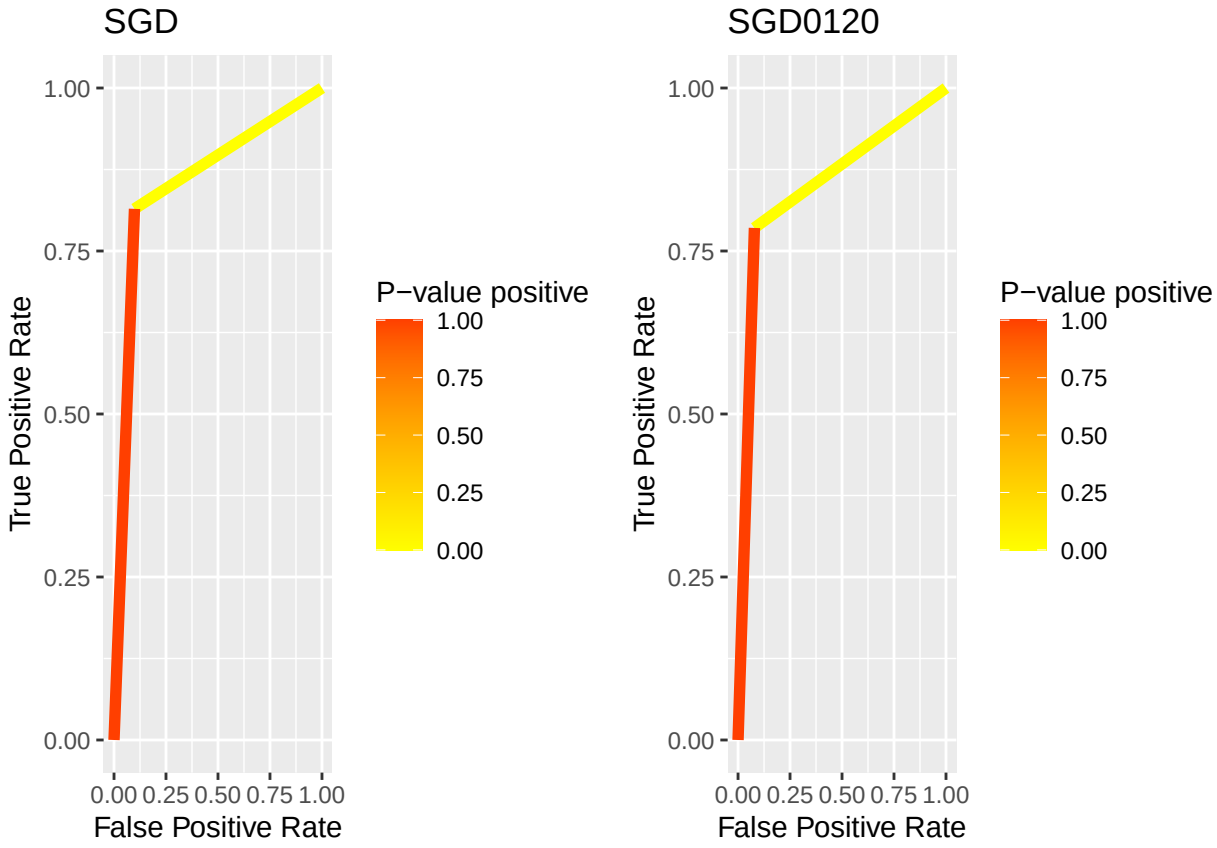
```
## Warning in read.table(file, sep = ",", na.strings = "?", colClasses =
## col_types, : incomplete final line found by readTableHeader on 'data/ROC/
## SGD.arff'
```

```
## Warning in read.table(file, sep = ",", na.strings = "?", colClasses =
## col_types, : incomplete final line found by readTableHeader on 'data/ROC/
## SGD0120.arff'
```

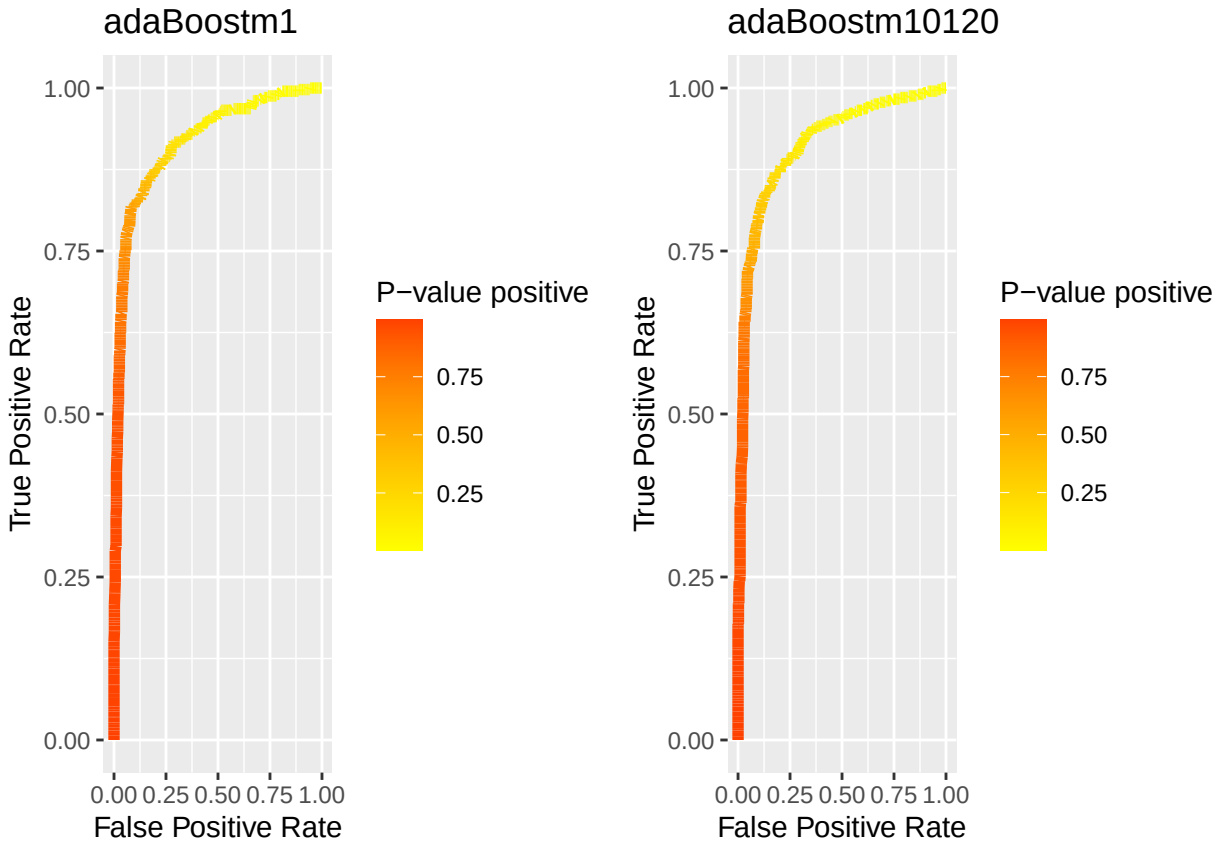
```
## [1] "data/ROC/SGD.arff"              "data/ROC/SGD0120.arff"
## [3] "data/ROC/adaBoostm1.arff"       "data/ROC/adaBoostm10120.arff"
## [5] "data/ROC/randomforest.arff"     "data/ROC/randomforest0130.arff"
```

```
do.call(ggarrange, c(ROCres, ncol = 2))
```

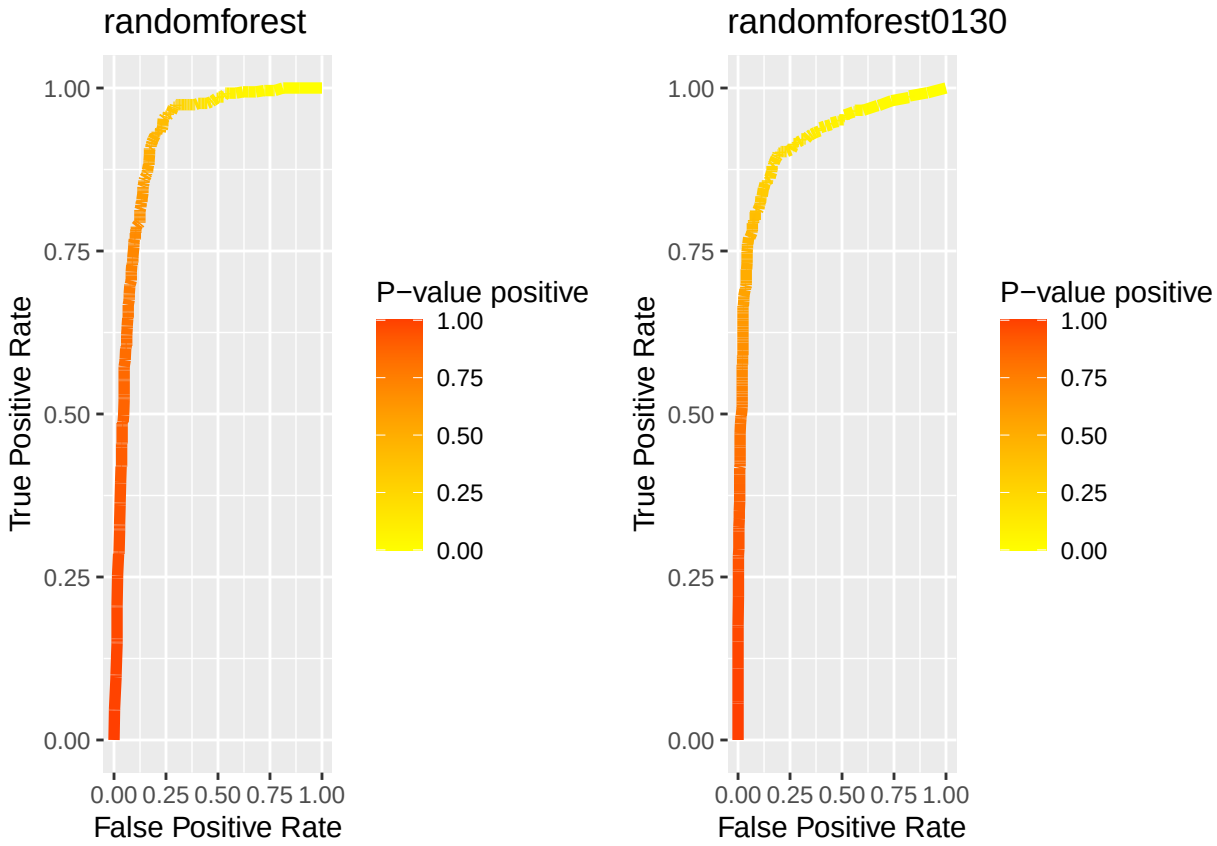
```
## $`1`
```


SGD
True Positive Rate
False Positive Rate
0.00
0.25
0.50
0.75
1.00
P−value positive
SGD0120
True Positive Rate
False Positive Rate
P−value positive

```
##
## $`2`
```


adaBoostm1
1.00
0.75
0.50
0.25
0.00
True Positive Rate
0.00 0.25 0.50 0.75 1.00
False Positive Rate
P−value positive
0.75
0.50
0.25
adaBoostm10120
1.00
0.75
0.50
0.25
0.00
True Positive Rate
0.00 0.25 0.50 0.75 1.00
False Positive Rate
P−value positive
0.75
0.50
0.25

```
##
## $`3`
```




```
##
## attr(,"class")
## [1] "list"      "ggarrange"
```

The algorithms SGD has the lowest AUC value from the three different algorithms. Both AdaBoostm1 and RandomForest AUC values are almost the same value. The little difference isn't significant enough to decide which algorithm to pick. False positives and false negatives are not the same. It's more important to lower the false negatives because of this area under the curve won't be used for further analysis.

Cost-sensitive classification

One the most important thing to take into account are the resulting false negatives when doing classification for people diagnosing with heart disease. The amount of people who are diagnosed negative but are actually positive for heart disease must be low as possible. In order to achieve this, it's possible to use a cost matrix. This result in a penalty for the algorithm when the prediction is negative for heart disease but the patient is actually positive for the heart disease. Comparing the confusion matrix between the default settings algorithm and the algorithm with cost sensitive matrix shows an excellent overview. Applying a cost matrix could possibly lead to lowering in accuracy for the algorithm. It's crucial to find balance between keeping the accuracy high as possible but lowering the false negatives as well.

After choosing a cost matrix of [0 0 4 0] for false negatives Randomforest performs significant lower in comparison with the default settings. The algorithm voting won't be used as final model, because applying a cost matrix significantly lower the accuracy.

RandomForest:

Accuracy: 87.08% Default cost matrix: [0.0,1.0;1.0,0.0] Confusion matrix: a b <– classified as 338 72 | a = FALSE 45 463 | b = TRUE

Accuracy: 86.71% Cost matrix: [0.0,1.0;2.0,0.0]
Confusion matrix: a b <– classified as 323 87 | a = FALSE 28 480 | b = TRUE

Accuracy: 85.33% Cost matrix: [0.0,1.0;3.0,0.0] Confusion matrix: a b <– classified as 297 113 | a = FALSE 17 491 | b = TRUE

AdaBoostm1

Accuracy: 83.44% Cost matrix: [0.0,1.0;3.0,0.0] Confusion matrix: a b <– classified as 293 117 | a = FALSE 31 477 | b = TRUE

Randomforest accuracy decrease 0.37 percent with cost matrix of [0 0 2 0] and lowering the false negatives with 19 instances. The accuracy decrease around 1.4% when using the cost matrix [0 0 3 0] and still maintaining an accuracy of 85.33%. The false negatives decrease with 11 instances but the false positives increases with 26 instances compared with cost matrix [0 0 2 0]. Using the cost matrix of [0 0 3 0] decreases false negatives a little more than 50% in comparison with the default matrix. It's more viable to lower the false negatives instead of increasing the false positives. AdaBoostm1 performs worse in accuracy and the amount of false negatives in comparison with Randomforest with a cost matrix of [0 0 3 0]. That's why the final model Randomforest with cost matrix of [0 0 3 0] will be selected.

Weka scheme: **weka.classifiers.meta.CostSensitiveClassifier -cost-matrix "[0.0 1.0; 3.0 0.0]" -S 1 -W weka.classifiers.trees.RandomForest – -P 100 -I 100 -num-slots 12 -K 2 -M 1.0 -V 0.001 -S 1** With an accuracy of 85.33%