Heart Disease Prediction

Ananthajit Srikanth

25/07/2021

Contents

Introduction

Heart attacks are the leading cause of death worldwide (source: Our World in Data). Hence, understanding how to predict the possibility of heart disease is extremely important.

This analysis will use the UCI Machine Learning Repository's Heart Disease Dataset to predict the risk of heart disease in patients. It uses a variety of classification methods (such as discriminant analysis, random forests, neural networks, et cetera), to provide a final F_1 score of 0.88 on a test set.

Dataset Information

This analysis uses four datasets, each of which are from four different regions:

- Hungarian Institute of Cardiology
- University Hospital, Zurich, Switzerland and University Hospital, Basel, Switzerland
- Cleveland Clinic Foundation
- V.A. Medical Center, Long Beach

Each of these regions form the *processed.hungarian*,*processed.switzerland*,*processed.cleveland*, and *processed.va* datasets.

The *processed* prefix indicates that out of 76 variables, only 14 important variables are used. All the tables are also comma separated.

Given below are the variables in the dataset, as per the heart-disease.names file:

1. Age in years
2. Sex (assigned at birth) (1 is male, and 0 is female)
3. Chest Pain Type (one of four values): 1 - Typical Angina 2 - Atypical Angina 3 - Non anginal Pain 4 - Asymptomatic
4. trestbps - Resting Blood Pressure in mm Hg
5. chol - Serum Cholesterol in mg/dl
6. fbs - Is fasting blood sugar greater than 120 mg/dl? (1 = yes, 0 = no)
7. restecg - Resting Electrocardiogram (ECG) results (one of three values): 0 - Normal 1 - Having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of $>$ 0.05 mV): This is a symptom of Myocardial Infarction 2 - Showing probable or definite left ventricular hypertrophy by Estes' criteria (enlargment of the left ventricle, usually to compensate for tissue loss)
8. thalach - Maximum Heart Rate Achieved (in beats per minute)
9. exang - Exercise Induced Angina (1 = yes, 0 = no)
10. oldpeak - ST depression induced by exercise relative to rest
11. slope - Slope of ST segment (as seen in ECG) during excercise:

- 1 - Upsloping
- 2 - Flat
- 3 - Downsloping

12. ca - Number of vessels colored by fluoroscopy (0 - 3)
13. thal: One of three possible values:

- 3 - Normal
- 6 - Fixed Defect
- 7 - Reversible Defect

14. num (outcome) - Diagnosis of Heart Disease: 1 - More than 50% narrowing of diameter of any major heart vessel 0 - Less than 50% narrowing of diameter of any major heart vessel.
    2,3, and 4 also indicate heart disease, of varying severity. (Based on the cleve.mod file)

The goal of this project, is to predict, with a high degree of certainty, whether or not someone suffers from heart disease.

Creating the Datasets

The datasets are publicly available at the UCI Machine Learning repository website.

First, libraries have to be loaded for analysis:

```
# Heart attack analysis

# Loading the required libraries, can add required libraries later in vector (
   same code as used in previous project, source: Ananthajit Srikanth)
required_libraries <- c("caret", "matrixStats", "tidyverse", "knitr", "broom",
   "ranger", "knitr", "rmarkdown", "mice", "e1071", "kernlab", "nnet", "rpart",
    "randomForest", "MASS", "naivebayes","adabag", "plyr","nnet", "stringr")
# The for loop installs and loads libraries one by one from required_libraries
for(i in 1:length(required_libraries)){
  if(!require(required_libraries[i], character.only = TRUE)){
    install.packages(required_libraries[i], repos = "http://cran.us.r-project.
       org")
    library(required_libraries[i], character.only = TRUE)
  }
  else{
    require(required_libraries[i], character.only = TRUE)}}

# Run this line only if necessary:
# update.packages()
```

Once the libraries are loaded, the datasets can be downloaded: Looking at one file, we can see the structure of the data:

```
# Show that the files are comma-separated
download.file(url = "https://archive.ics.uci.edu/ml/machine-learning-databases/
   heart-disease/processed.cleveland.data", destfile = 'processed.cleveland.
   data')
readLines(con = 'processed.cleveland.data', n = 1)
```

```
## [1] "63.0,1.0,1.0,145.0,233.0,1.0,2.0,150.0,0.0,2.3,3.0,0.0,6.0,0"
```

We can see that the file has comma-separated values.

Looking at the other three tables:

```
download.file(url = "https://archive.ics.uci.edu/ml/machine-learning-databases/
   heart-disease/processed.hungarian.data", destfile = "processed.hungarian.
   data")

download.file(url = "https://archive.ics.uci.edu/ml/machine-learning-databases/
   heart-disease/processed.va.data", destfile = "processed.va.data")

download.file(url = "https://archive.ics.uci.edu/ml/machine-learning-databases/
   heart-disease/processed.switzerland.data", destfile = "processed.switzerland
   .data")

noquote("Hungarian")
```

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

```
readLines(con = 'processed.hungarian.data', n = 1)
```

```
## [1] "28,1,2,130,132,0,2,185,0,0,?,?,?,0"
```

```
noquote("Switzerland")
```

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

```
readLines(con = 'processed.switzerland.data', n = 1)
```

```
## [1] "32,1,1,95,0,?,0,127,0,.7,1,?,?,1"
```

```
noquote("va")
```

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

```
readLines(con = 'processed.va.data', n = 1)
```

```
## [1] "63,1,4,140,260,0,1,112,1,3,2,?,?,2"
```

```
# download extra files for help and reading
download.file(url = "https://archive.ics.uci.edu/ml/machine-learning-databases/
   heart-disease/heart-disease.names", destfile = "heart-disease.txt")

download.file(url = "https://archive.ics.uci.edu/ml/machine-learning-databases/
   heart-disease/cleve.mod", destfile = "cleve.txt")

# Why you shouldn't use cleveland.data
download.file(url = "https://archive.ics.uci.edu/ml/machine-learning-databases/
   heart-disease/WARNING", destfile = "warn.txt")
```

They are also comma-separated. It can also be seen that some fields have a ?. This can be replaced by an NA.

The files can now be downloaded in the csv format:

```
# Downloading the processed.data files as csv

download.file(url = "https://archive.ics.uci.edu/ml/machine-learning-databases/
   heart-disease/processed.cleveland.data", destfile = "cleveland.csv")

download.file(url = "https://archive.ics.uci.edu/ml/machine-learning-databases/
   heart-disease/processed.hungarian.data", destfile = "hungarian.csv")

download.file(url = "https://archive.ics.uci.edu/ml/machine-learning-databases/
   heart-disease/processed.va.data", destfile = "va.csv")

download.file(url = "https://archive.ics.uci.edu/ml/machine-learning-databases/
   heart-disease/processed.switzerland.data", destfile = "switzerland.csv")
```

Based on the documentation provided in heart-disease.names in the data folder, the column names can be stored in a vector:

```
# Column names for data frames
colnames_heartattack <- c("age", "sex", "cp", "trestbps", "chol", "fbs", "
   restecg", "thalach", "exang", "oldpeak", "slope", "ca", "thal", "num")
```

As the csv files have been downloaded, now they can be read.

```
cleveland <- read_csv("cleveland.csv", col_names = colnames_heartattack)
```

```
##
## -- Column specification
   --------------------------------------------------------
## cols(
##   age = col_double(),
##   sex = col_double(),
##   cp = col_double(),
##   trestbps = col_double(),
##   chol = col_double(),
##   fbs = col_double(),
##   restecg = col_double(),
##   thalach = col_double(),
##   exang = col_double(),
##   oldpeak = col_double(),
##   slope = col_double(),
##   ca = col_character(),
##   thal = col_character(),
##   num = col_double()
## )
```

As can be seen, read_csv converted ca and thal (which use numbers as categories) as characters, most likely due to the presence of '?'

```
any(str_detect(cleveland$thal, pattern = "\\?"))
```

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

This shows that there is at least one "?" in the column *thal*, hence it has been assigned the character variable type.

As all of the columns contain numbers/numerical values, `as.numeric()` can be used to convert all the columns into numerics. However, the ?s have to be converted to NAs first.

```
# Convert all ? to NA
cleveland[cleveland == "?"] <- NA
# Convert all columns to numeric
cleveland <- apply(cleveland, 2, function(x) as.numeric(x)) %>% as.data.frame()
```

As there are multiple datasets, the region of the dataset also has to be specified. In the case of the *cleveland* dataset, the variable *dataset* shall be set to *cleveland*

```
# specify the dataset region
cleveland <- cleveland %>% dplyr::mutate(dataset = 'cleveland')
```

The same process is repeated for the other three datasets:

```
switzerland <- read_csv("switzerland.csv", col_names = colnames_heartattack)
```

```
##
## -- Column specification
   --------------------------------------------------------
## cols(
##   age = col_double(),
##   sex = col_double(),
##   cp = col_double(),
##   trestbps = col_character(),
##   chol = col_double(),
##   fbs = col_character(),
##   restecg = col_character(),
##   thalach = col_character(),
##   exang = col_character(),
##   oldpeak = col_character(),
##   slope = col_character(),
##   ca = col_character(),
##   thal = col_character(),
##   num = col_double()
## )
```

```r
switzerland[switzerland == "?"] <- NA
switzerland <- apply(switzerland, 2, function(x) as.numeric(x)) %>% as.data.
   frame()
switzerland <- switzerland %>% dplyr::mutate(dataset = 'switzerland')

hungarian <- read_csv("hungarian.csv", col_names = colnames_heartattack)
```

```
##
## -- Column specification
   --------------------------------------------------------
## cols(
##   age = col_double(),
##   sex = col_double(),
##   cp = col_double(),
##   trestbps = col_character(),
##   chol = col_character(),
##   fbs = col_character(),
##   restecg = col_character(),
##   thalach = col_character(),
##   exang = col_character(),
##   oldpeak = col_double(),
##   slope = col_character(),
##   ca = col_character(),
##   thal = col_character(),
##   num = col_double()
## )
```

```r
hungarian[hungarian == "?"] <- NA
hungarian <- apply(hungarian, 2, function(x) as.numeric(x)) %>% as.data.frame()
hungarian <- hungarian %>% dplyr::mutate(dataset = "hungarian")

va <- read_csv("va.csv", col_names = colnames_heartattack)
```

```
##
## -- Column specification
   --------------------------------------------------------
## cols(
##   age = col_double(),
##   sex = col_double(),
##   cp = col_double(),
##   trestbps = col_character(),
##   chol = col_character(),
##   fbs = col_character(),
##   restecg = col_double(),
##   thalach = col_character(),
##   exang = col_character(),
##   oldpeak = col_character(),
##   slope = col_character(),
##   ca = col_character(),
##   thal = col_character(),
##   num = col_double()
## )
```

```r
va[va == "?"] <- NA
va <- apply(va, 2, function(x) as.numeric(x)) %>% as.data.frame()
va <- va %>% dplyr::mutate(dataset = "va")
```

The datasets can now be joined together in a table called *total_set*:

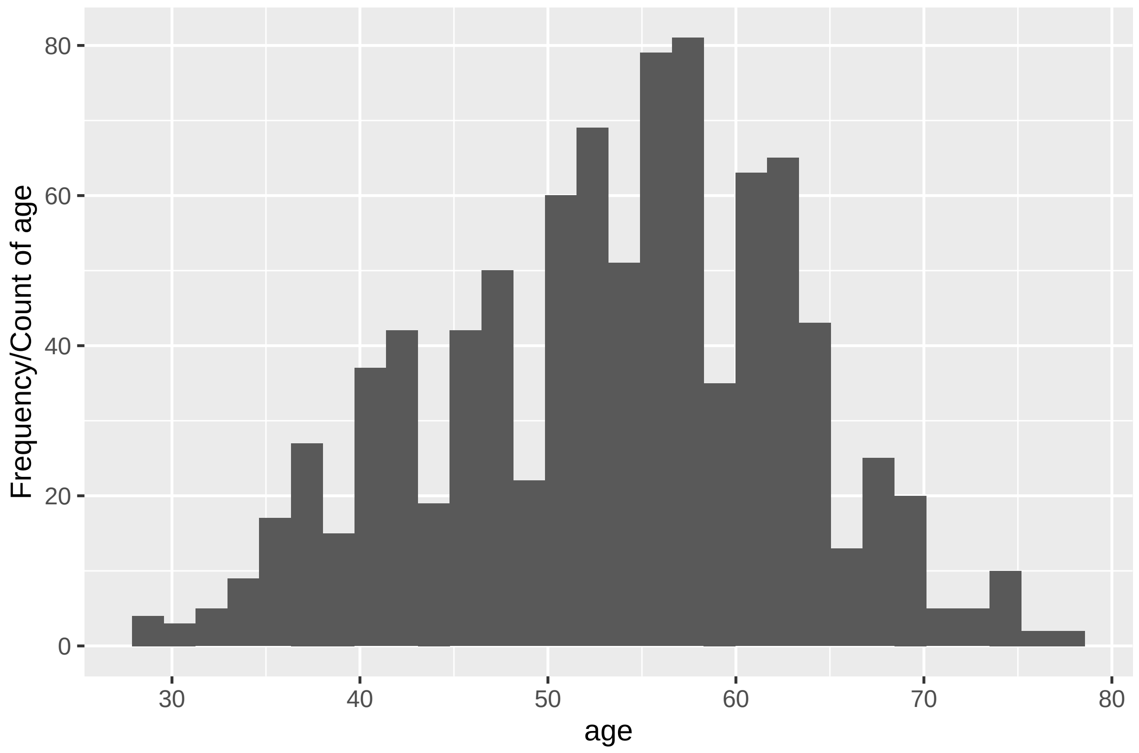
```
# join the four datasets together
total_set <- bind_rows(cleveland, switzerland, hungarian, va)
```

Exploratory data analysis

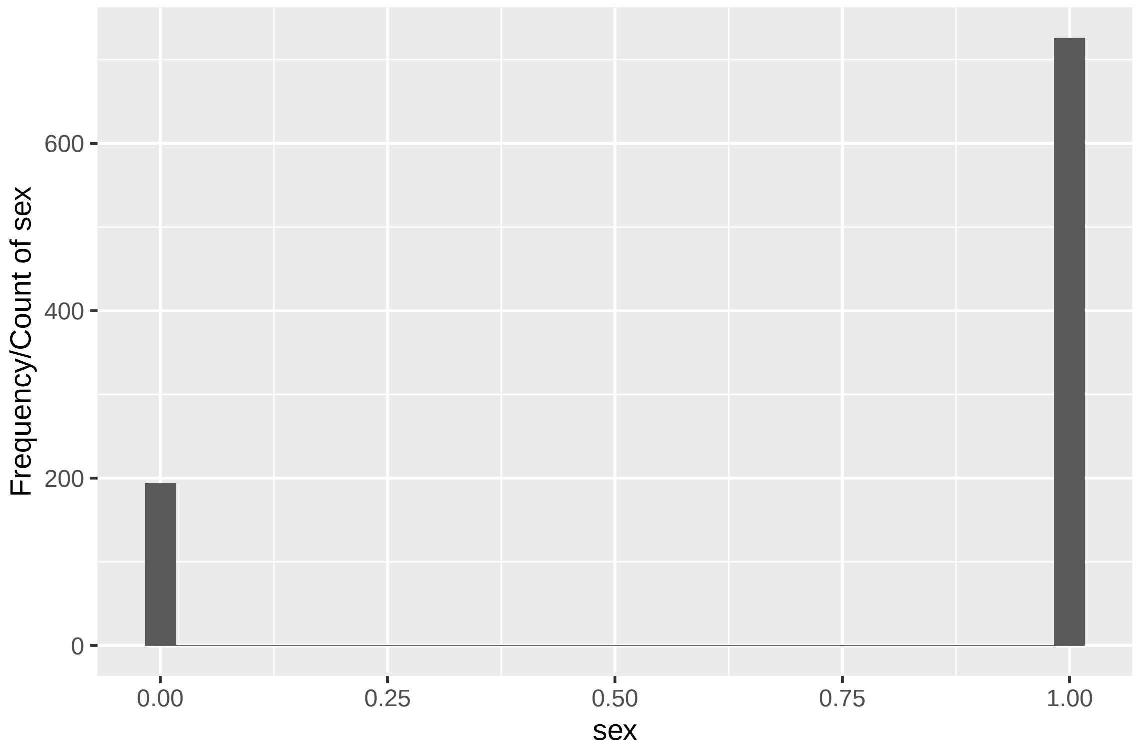
Before perfoming any further data cleaning, further analysis has to be done.

```
for(i in 1:(ncol(total_set)-1)) {
 x <- total_set[,i]
 x <- data.frame(x = x)
 name <- names(total_set)[i]
  print(x %>% ggplot(aes(x)) + geom_histogram() + xlab(label = name) + ylab(
     label = paste0("Frequency/Count of ",name)))
 }
```

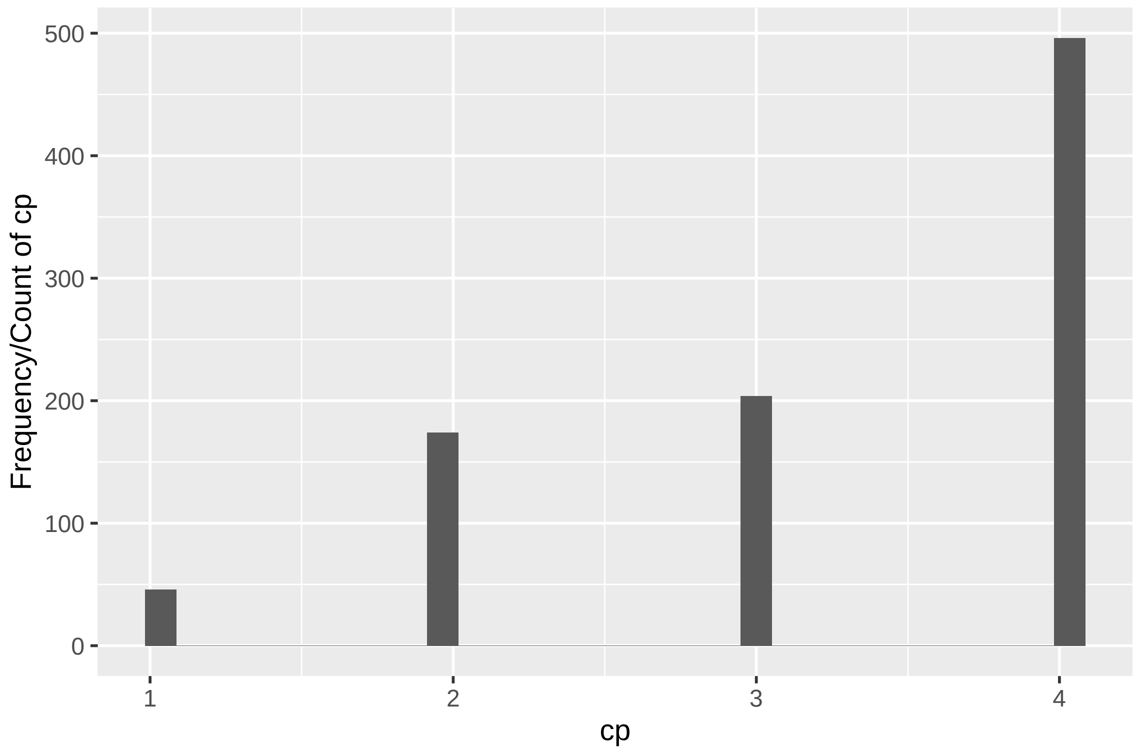
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

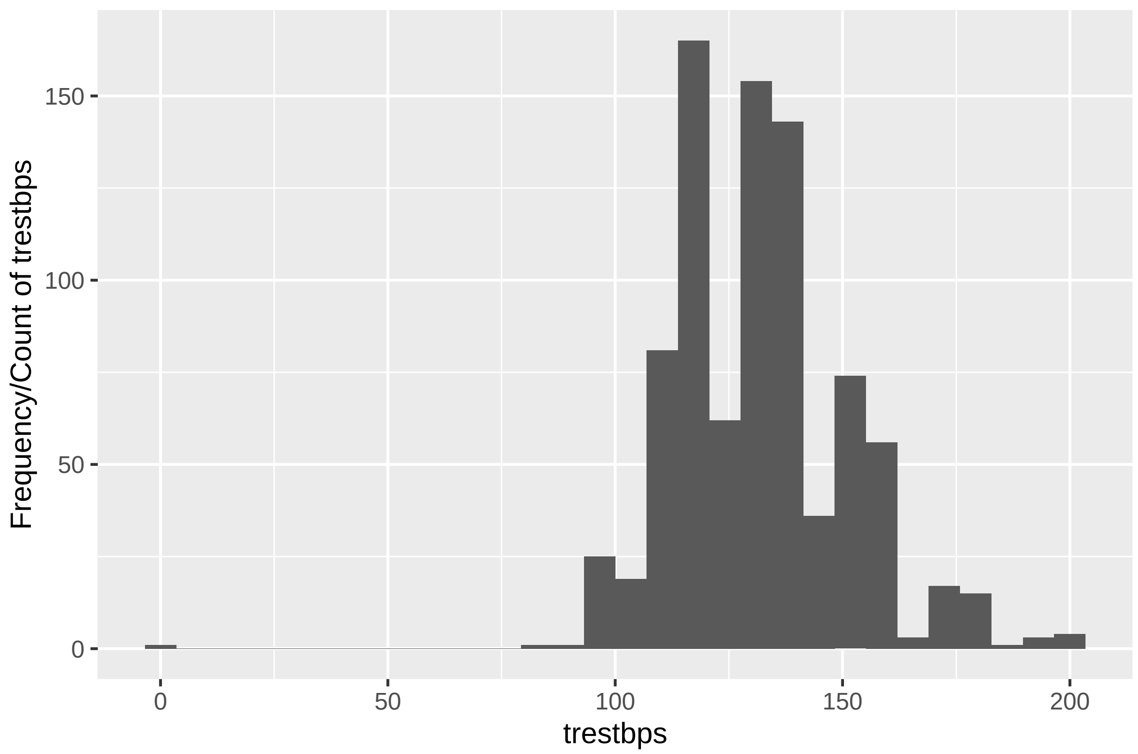


```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



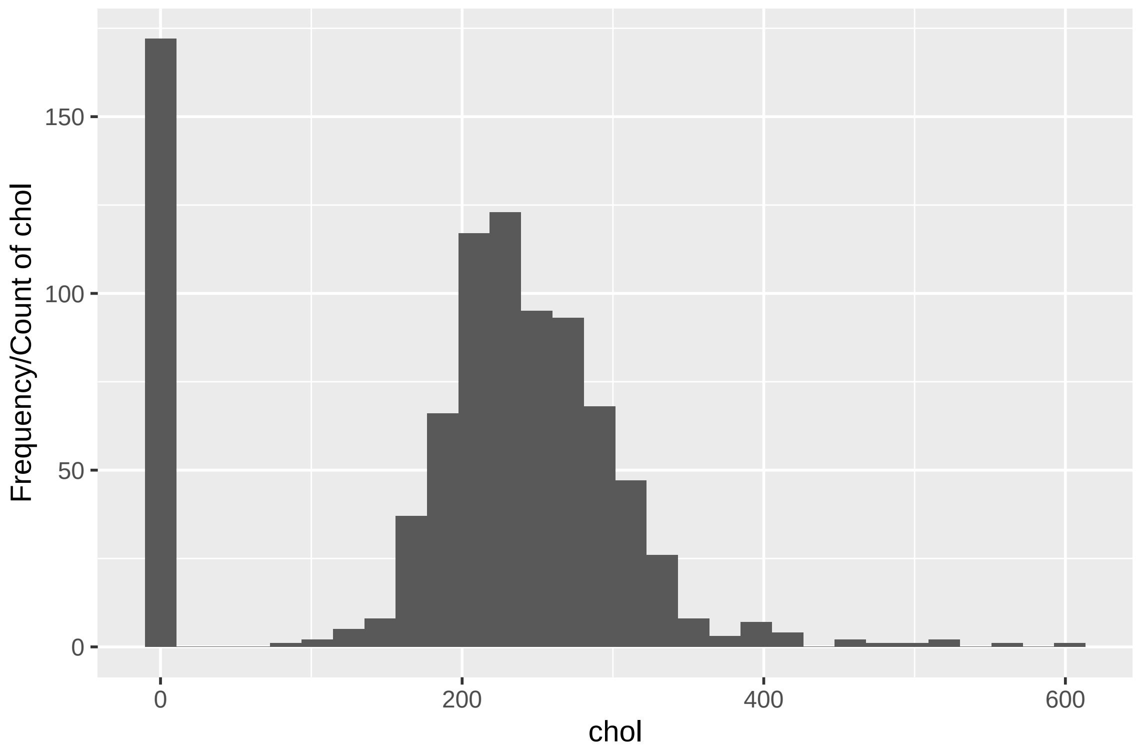
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 59 rows containing non-finite values (stat_bin).
```



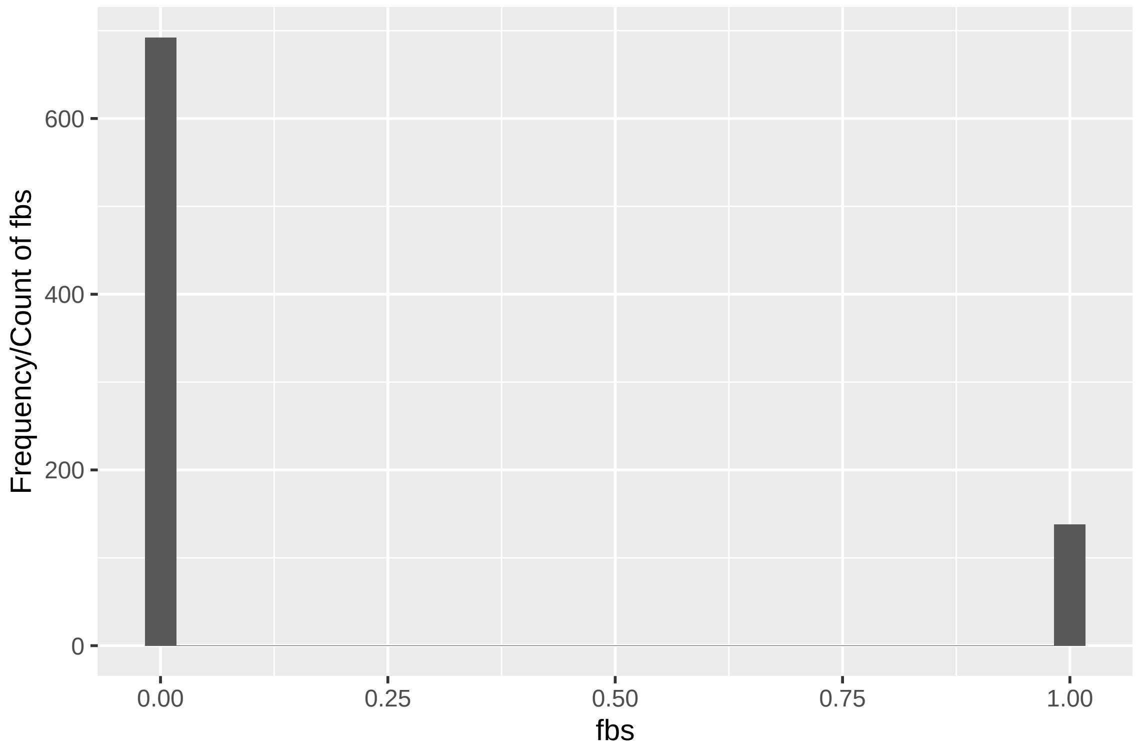
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 30 rows containing non-finite values (stat_bin).
```



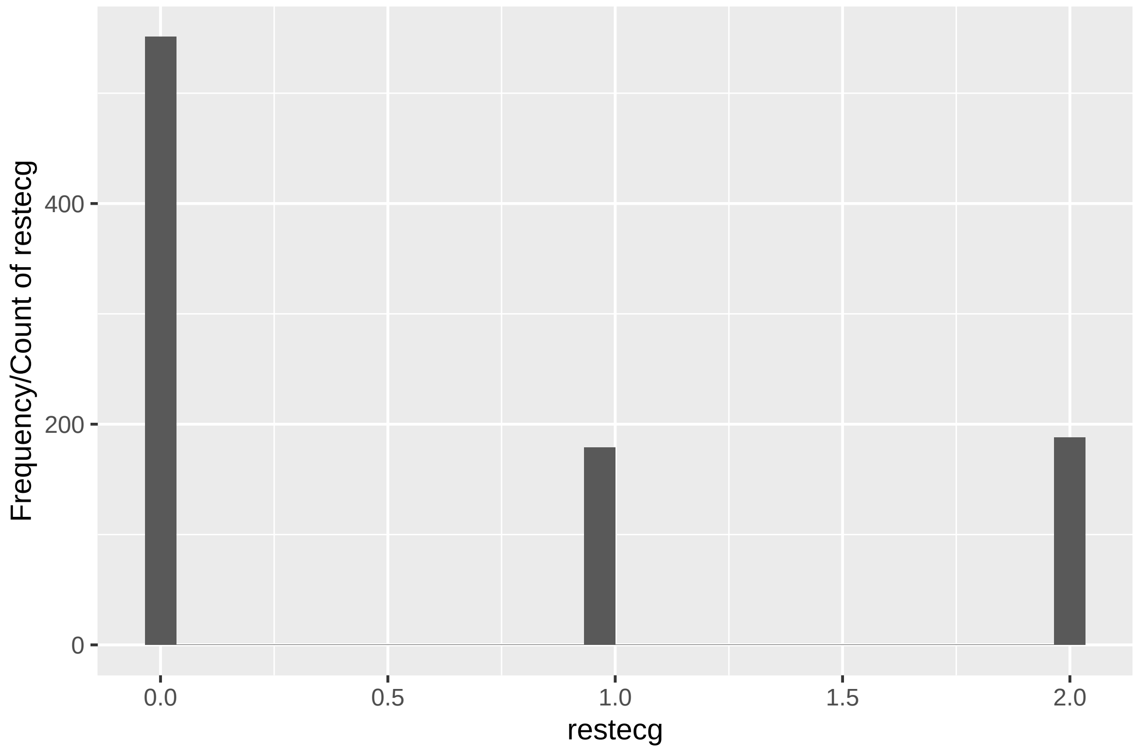
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 90 rows containing non-finite values (stat_bin).
```



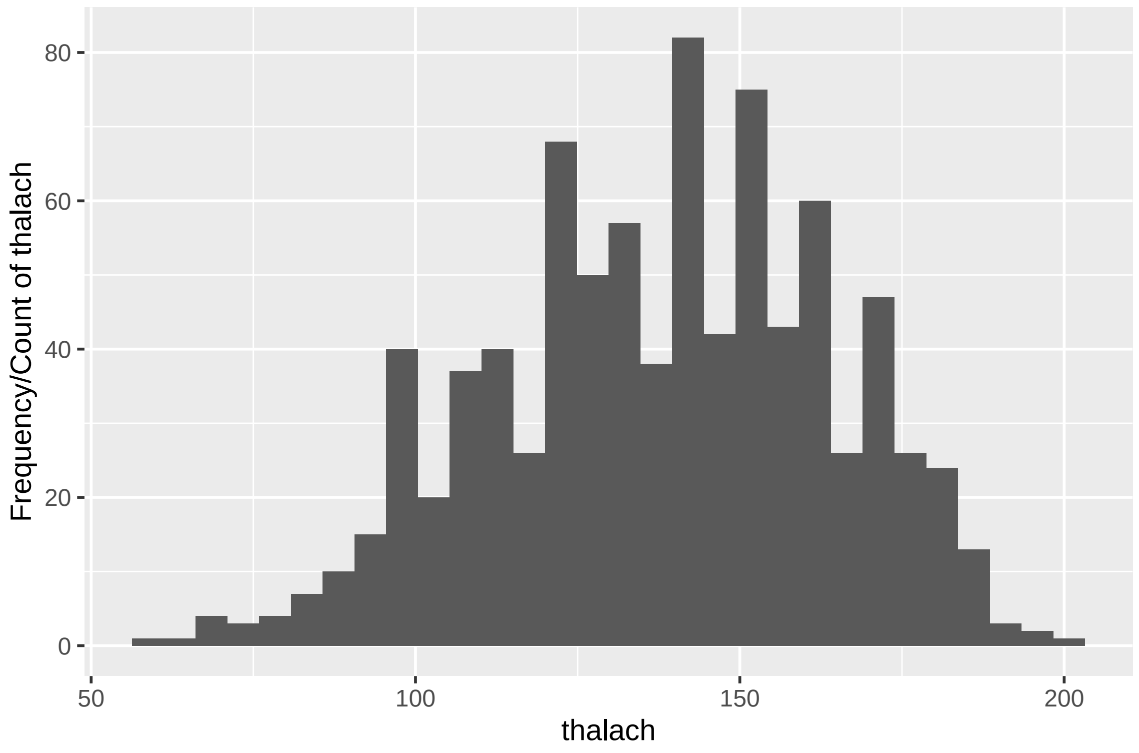
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 2 rows containing non-finite values (stat_bin).
```



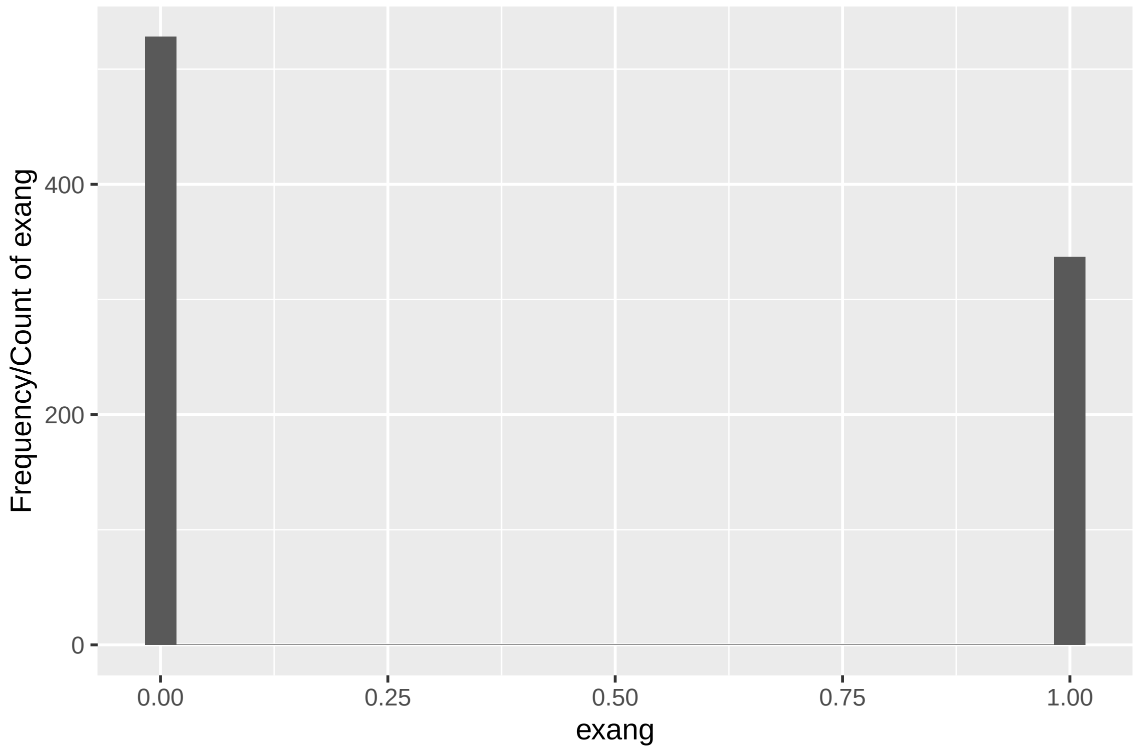
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 55 rows containing non-finite values (stat_bin).
```



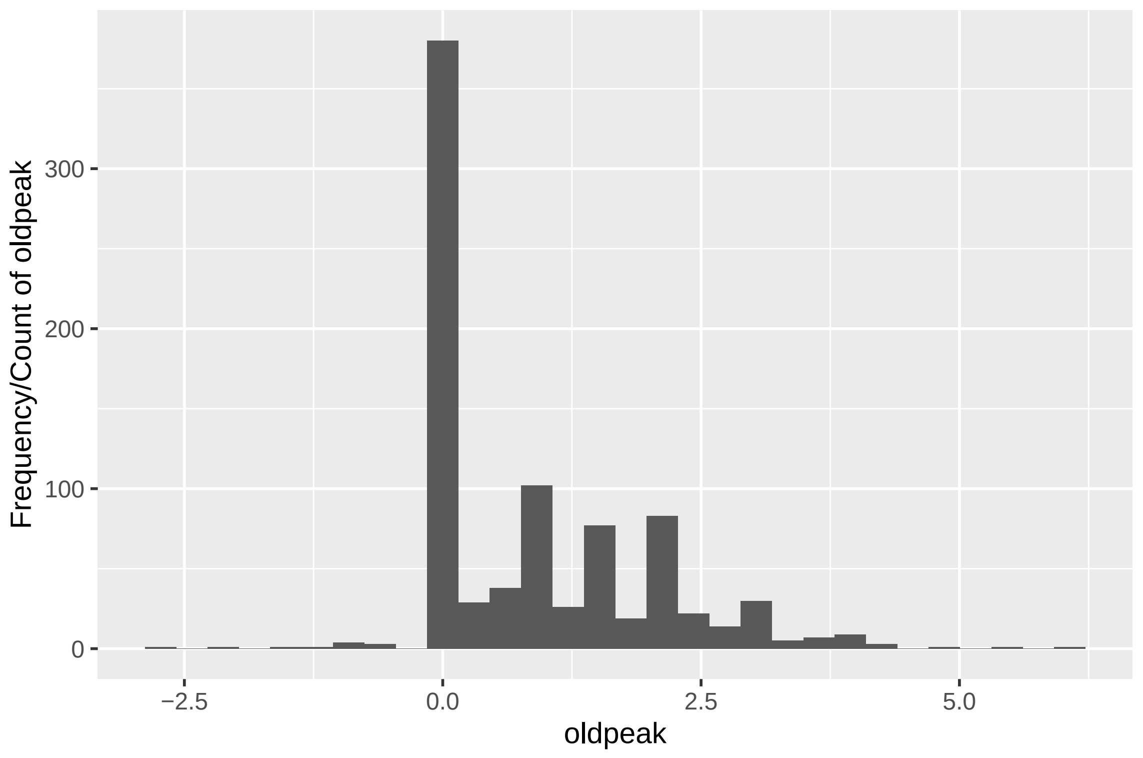
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 55 rows containing non-finite values (stat_bin).
```



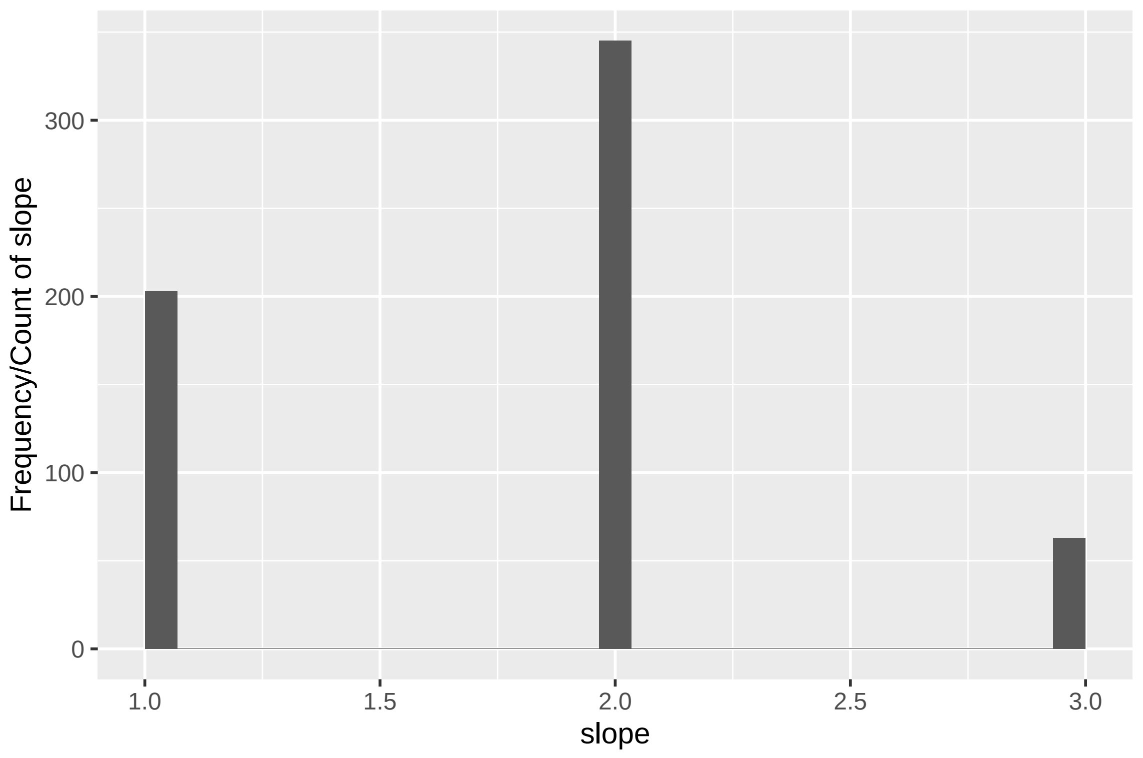
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 62 rows containing non-finite values (stat_bin).
```



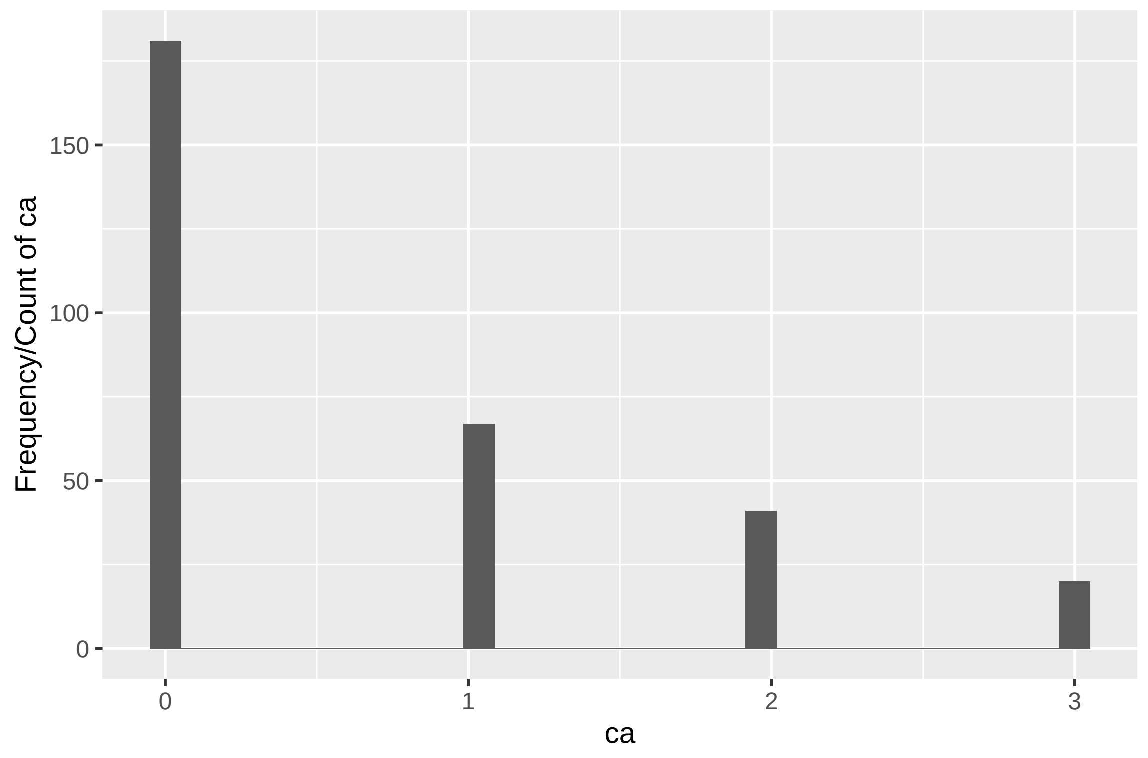
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 309 rows containing non-finite values (stat_bin).
```



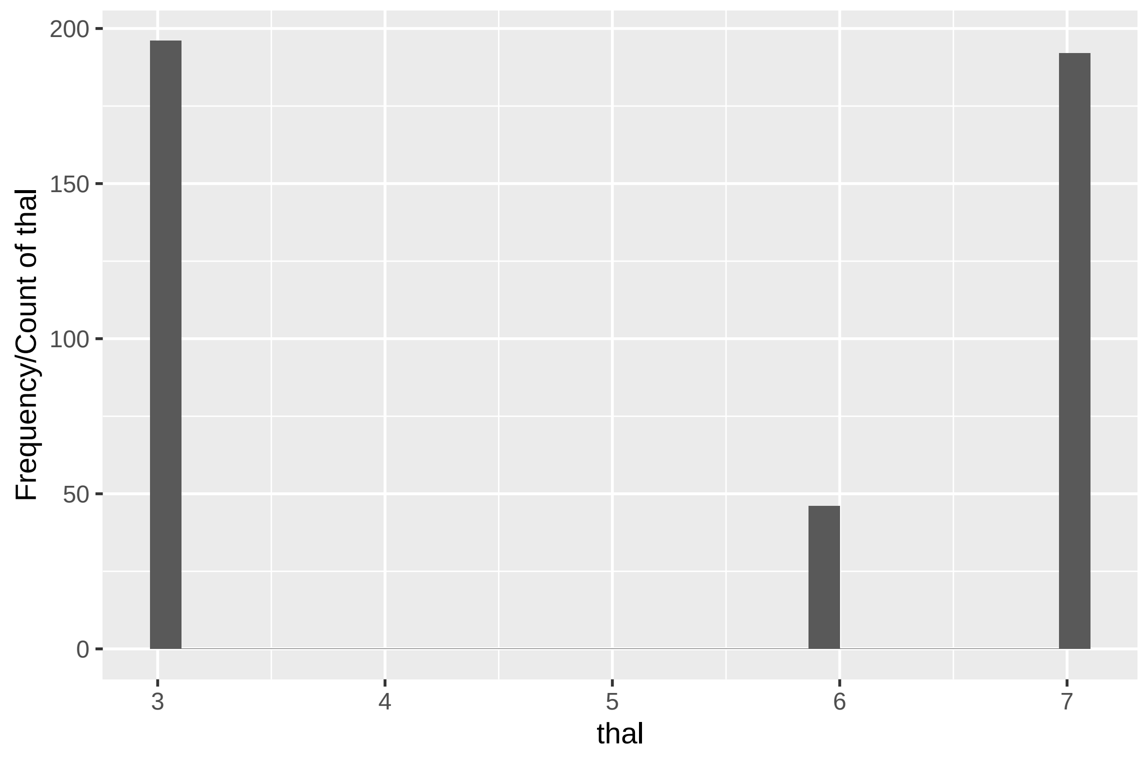
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 611 rows containing non-finite values (stat_bin).
```

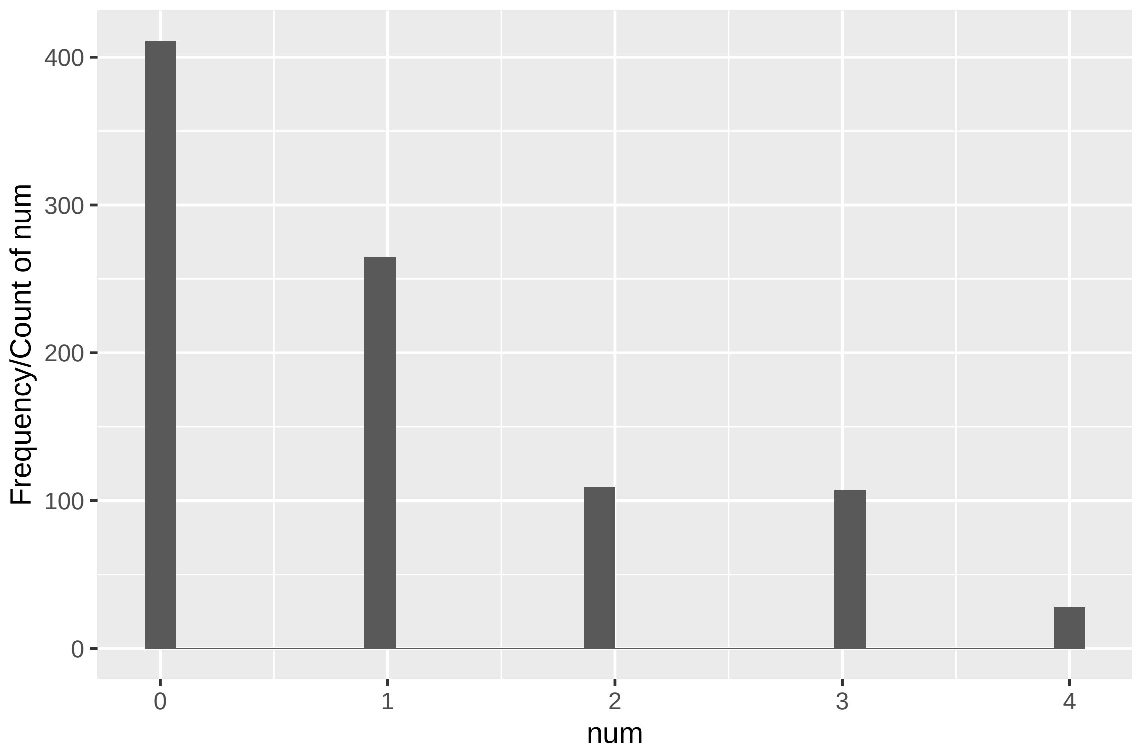


```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 486 rows containing non-finite values (stat_bin).
```



```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Cleaning up erroneous values

Given above is a histogram for all the variables. Most of the variables seems to show reasonable distibutions, except the *chol* variable, which has a skewed distribution. It seems quite unreasonable for anyone to have 0 mg/dl cholesterol.

```
# check error cholesterol

total_set %>% group_by(dataset) %>% dplyr::summarize(zeros = sum(chol == 0, na.
   rm = TRUE)) %>% knitr::kable()
```

dataset	zeros
cleveland	0
hungarian	0
switzerland	123
va	49

This shows that the data was probably not input correctly. So, all zeros will be converted to NAs:

```
mutated_chol <- total_set$chol
mutated_chol[mutated_chol == 0] <- NA
total_set <- total_set %>% dplyr::mutate(chol = mutated_chol)
```

The same has to be done for trestbps:

```
# covert all zeros in trestbps to NA, because 0 bp is only possible at death
# look at trestbps

sum(total_set$trestbps == 0, na.rm = TRUE)
```

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

```
# only one


# covert the one erroneous value into an NA
mutated_trestbps <- total_set$trestbps
mutated_trestbps[mutated_trestbps == 0] <- NA

total_set <- total_set %>% dplyr::mutate(trestbps = mutated_trestbps)
```

From the dataset documentation, although all columns use numerical values, some numerical values are used to represent categorical outcomes, so these have to be converted to factors:

```
# convert factors
total_set <- total_set %>% dplyr::mutate(num = as.factor(num), sex = as.factor(
   sex), cp = as.factor(cp), fbs = as.factor(fbs), restecg = as.factor(restecg)
   ,exang = as.factor(exang), slope = as.factor(slope), ca = as.factor(ca),
   thal = as.factor(thal), dataset=as.factor(dataset))
```

We can now look at the structure of *total_set*

```
str(total_set)
```

```
## 'data.frame':    920 obs. of  15 variables:
##  $ age     : num  63 67 67 37 41 56 62 57 63 53 ...
##  $ sex     : Factor w/ 2 levels "0","1": 2 2 2 2 1 2 1 1 2 2 ...
##  $ cp      : Factor w/ 4 levels "1","2","3","4": 1 4 4 3 2 2 4 4 4 4 ...
##  $ trestbps: num  145 160 120 130 130 120 140 120 130 140 ...
##  $ chol    : num  233 286 229 250 204 236 268 354 254 203 ...
##  $ fbs     : Factor w/ 2 levels "0","1": 2 1 1 1 1 1 1 1 1 2 ...
##  $ restecg : Factor w/ 3 levels "0","1","2": 3 3 3 1 3 1 3 1 3 3 ...
##  $ thalach : num  150 108 129 187 172 178 160 163 147 155 ...
##  $ exang   : Factor w/ 2 levels "0","1": 1 2 2 1 1 1 1 2 1 2 ...
##  $ oldpeak : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
##  $ slope   : Factor w/ 3 levels "1","2","3": 3 2 2 3 1 1 3 1 2 3 ...
##  $ ca      : Factor w/ 4 levels "0","1","2","3": 1 4 3 1 1 1 3 1 2 1 ...
##  $ thal    : Factor w/ 3 levels "3","6","7": 2 1 3 1 1 1 1 1 3 3 ...
##  $ num     : Factor w/ 5 levels "0","1","2","3",..: 1 3 2 1 1 1 4 1 3 2 ...
##  $ dataset : Factor w/ 4 levels "cleveland","hungarian",..: 1 1 1 1 1 1 1 1
   1 1 ...
```

While most of the documentation claims that the num variable is binary, looking at the structure of the dataset, there is also factor levels 2, 3 and 4 for the num variable. However, as these imply increasing levels of severity of heart disease, these can also be included with factor level 1. This coverts this problem from a multiclass classification problem into a binary classification problem.

```
# Converts 2,3 and 4 into 1.
# as.character has to be used since as.numeric converts factor into level index
   , for example 0 becomes 1, 1 becomes 2, etc.
temp_num <- as.numeric(as.character(total_set$num))

temp_num_2 <- case_when(temp_num == 0 ~ 0,
                        temp_num %in% c(1,2,3,4) ~ 1,
                        TRUE ~ NA_real_
                      )
remove(temp_num)

total_set <- total_set %>% dplyr::mutate(num = factor(temp_num_2))
str(total_set$num)
```

```
##  Factor w/ 2 levels "0","1": 1 2 2 1 1 1 2 1 2 2 ...
```

```
total_set$num %>% head()
```

```
## [1] 0 1 1 0 0 0
## Levels: 0 1
```

```
table(total_set$num)
```

```
##
##   0   1
## 411 509
```

We can also see that the prevalence of both 0 and 1 is very close to one another.

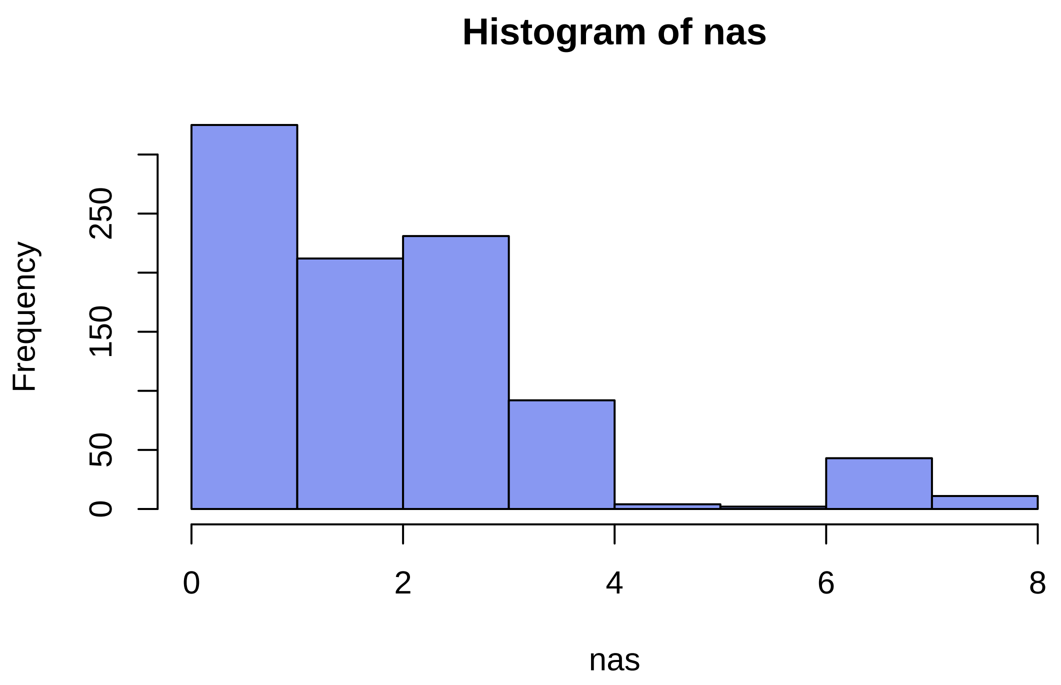
Now, the dataset can be checked for NAs.

```
apply(total_set,2,function(x)sum(is.na(x))) %>% knitr::kable()
```

	x
age	0
sex	0
cp	0
trestbps	60
chol	202
fbs	90
restecg	2
thalach	55
exang	55
oldpeak	62
slope	309
ca	611
thal	486
num	0
dataset	0

Many columns have NAs. This has to be addressed before training as multiple models do not support datasets with NAs.

```
nas <- apply(total_set, 1, function(x)sum(is.na(x)))
hist(nas, col = '#8898f2')
```



The majority of data points have 0 - 3 NA values, so the mice library can be used to impute values.

Before addressing the NAs, the datasets have to be split into three parts, a training set, a 'debug set' to test models on, and a final hold-out test set.

Splitting the datasets

From looking at the structure of *total_set* there are 920 rows, so having approx. 125 debugging and 125 testing examples should be adequate.

```
# set.seed for consistency
set.seed(2000, sample.kind = 'Rounding') # remove sample.kind argument if you
   use R 3.5
```

```
## Warning in set.seed(2000, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
split_index <- createDataPartition(y = total_set$num, times = 1, p = 250/960,
   list = FALSE)

train_set <- total_set[-split_index,]
temp_set <- total_set[split_index,]

set.seed(9000, sample.kind = 'Rounding')
```

```
## Warning in set.seed(9000, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
# split debug and test into approx 125 examples each
split_index_2 <- createDataPartition(y = temp_set$num, times = 1, p = 0.5, list
    = FALSE)

debug_set <- temp_set[-split_index_2,]
test_set <- temp_set[split_index_2,]
```

```
remove(temp_set, split_index, split_index_2)
```

We can now look at the structure of the debug set:

```
# debug set structure

str(debug_set)
```

```
## 'data.frame':    120 obs. of  15 variables:
##  $ age     : num  67 63 56 54 48 58 51 54 44 52 ...
##  $ sex     : Factor w/ 2 levels "0","1": 2 2 2 2 1 1 2 1 2 2 ...
##  $ cp      : Factor w/ 4 levels "1","2","3","4": 4 4 3 4 3 1 1 3 4 2 ...
##  $ trestbps: num  120 130 130 140 130 150 125 135 110 120 ...
##  $ chol    : num  229 254 256 239 275 283 213 304 197 325 ...
##  $ fbs     : Factor w/ 2 levels "0","1": 1 1 2 1 1 2 1 2 1 1 ...
##  $ restecg : Factor w/ 3 levels "0","1","2": 3 3 3 1 1 3 3 1 3 1 ...
##  $ thalach : num  129 147 142 160 139 162 125 170 177 172 ...
##  $ exang   : Factor w/ 2 levels "0","1": 2 1 2 1 1 1 2 1 1 1 ...
##  $ oldpeak : num  2.6 1.4 0.6 1.2 0.2 1 1.4 0 0 0.2 ...
##  $ slope   : Factor w/ 3 levels "1","2","3": 2 2 2 1 1 1 1 1 1 1 ...
##  $ ca      : Factor w/ 4 levels "0","1","2","3": 3 2 2 1 1 1 2 1 2 1 ...
##  $ thal    : Factor w/ 3 levels "3","6","7": 3 3 2 1 1 1 1 1 1 1 ...
##  $ num     : Factor w/ 2 levels "0","1": 2 2 2 1 1 1 1 1 2 1 ...
##  $ dataset : Factor w/ 4 levels "cleveland","hungarian",..: 1 1 1 1 1 1 1 1
   1 1 ...
```

and the test set:

```
# test set structure
str(test_set)
```

```
## 'data.frame':    121 obs. of  15 variables:
##  $ age     : num  53 44 50 40 60 71 53 54 51 48 ...
##  $ sex     : Factor w/ 2 levels "0","1": 2 2 1 2 2 1 2 2 1 2 ...
##  $ cp      : Factor w/ 4 levels "1","2","3","4": 4 2 3 4 4 2 3 4 4 4 ...
##  $ trestbps: num  140 120 120 110 117 160 130 124 130 122 ...
##  $ chol    : num  203 263 219 167 230 302 197 266 305 222 ...
##  $ fbs     : Factor w/ 2 levels "0","1": 2 1 1 1 2 1 2 1 1 1 ...
##  $ restecg : Factor w/ 3 levels "0","1","2": 3 1 1 3 1 1 3 3 1 3 ...
##  $ thalach : num  155 173 158 114 160 162 152 109 142 186 ...
##  $ exang   : Factor w/ 2 levels "0","1": 2 1 1 2 2 1 1 2 2 1 ...
##  $ oldpeak : num  3.1 0 1.6 2 1.4 0.4 1.2 2.2 1.2 0 ...
##  $ slope   : Factor w/ 3 levels "1","2","3": 3 1 2 2 1 1 3 2 2 1 ...
##  $ ca      : Factor w/ 4 levels "0","1","2","3": 1 1 1 1 3 3 1 2 1 1 ...
##  $ thal    : Factor w/ 3 levels "3","6","7": 3 3 1 3 3 1 1 3 3 1 ...
##  $ num     : Factor w/ 2 levels "0","1": 2 1 1 2 2 1 1 2 2 1 ...
##  $ dataset : Factor w/ 4 levels "cleveland","hungarian",..: 1 1 1 1 1 1 1 1
   1 1 ...
```

The test and debug sets have approximately 120 entries each.

We can begin the analysis. However, multiple training algorithms require there to not be any NAs. So, the data must be imputed.

Imputing values using the *mice* library

Imputation is a method by which values that will be used for training are predicted, based on existing data. For example, if the ca variable is an NA, but restecg is 1 (heart disease risk) and chol (cholesterol) is high, then it would be illogical to predict 3 (3 arteries with bloodflow and less blockage in fluoroscopy), and an imputation method would predict, for example, a 1.

Imputation is only done after splitting the dataset. If performed before splitting, values in the debug/test set may be imputed based on predictions made not only using the debug/test set, but also using the training set, which may lead to skewed imputes, since there may be differences in the datasets after splitting. (Imputation relies on relations between variables, which can vary between datasets.) Imputation is technically part of the training/testing process, since it modifies data values, so when debugging/testing,

```
# set.seed for consistency
set.seed(1900, sample.kind = "Rounding") #remove sample.kind if you use R 3.5
```

```
## Warning in set.seed(1900, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
# impute (insert by prediction) values using mice
# rf works with all data types and is fast and efficient


# remove outcome column so that imputation does not rely on outcome:

train_set_no_outcomes <- train_set %>% dplyr::select(-num)
# run mice

mice_imputation <- mice(train_set_no_outcomes, blocks = c("trestbps", "chol", "
    fbs", "restecg", "thalach", "exang", "oldpeak","slope", "ca", "thal"),
    method = 'rf', maxit = 50, m = 10)
```

```
## Warning: Number of logged events: 500
```

```
train_set_2 <- complete(mice_imputation)

# First entries don't have NAs
# identical(head(test), head(total_set_no_outcomes)) is TRUE so order is
    maintained
# sum(is.na(train_set_2)) is 0

train_set_2 <- train_set_2 %>% dplyr::mutate(num = train_set$num)

set.seed(3000, sample.kind = "Rounding") #remove sample.kind if you use R 3.5
```

```
## Warning in set.seed(3000, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
# repeat for debug and test set
debug_set_no_outcomes <- debug_set %>% dplyr::select(-num)
# run mice

mice_imputation_debug <- mice(debug_set_no_outcomes, blocks = c("trestbps", "
    chol", "fbs", "restecg", "thalach", "exang", "oldpeak","slope", "ca", "thal"
    ), method = 'rf', maxit = 50, m = 10)
```

```
## Warning: Number of logged events: 1500
```

```
debug_set_2 <- complete(mice_imputation_debug)
debug_set_2 <- debug_set_2 %>% dplyr::mutate(num = debug_set$num)

set.seed(5000, sample.kind = "Rounding") #remove sample.kind if you use R 3.5
```

```
## Warning in set.seed(5000, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
test_set_no_outcomes <- test_set %>% dplyr::select(-num)
# run mice

mice_imputation_test <- mice(test_set_no_outcomes, blocks = c("trestbps", "chol
   ", "fbs", "restecg", "thalach", "exang", "oldpeak","slope", "ca", "thal"),
   method = 'rf', maxit = 50, m = 10)
```

```
## Warning: Number of logged events: 1000
```

```
test_set_2 <- complete(mice_imputation_test)
test_set_2 <- test_set_2 %>% dplyr::mutate(num = test_set$num)
```

Based on the structure of these values, classification algorithms can now be created. But first, an accuracy metric has to be defined.

The accuracy metric

Since this is a binary classification problem, an accuracy metric has to be used to gauge model performance. In this case, the F1 score can be used. F1 is defined as follows:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Where precision is the ratio of true positives to the number of terms predicted positive. (In this case, positive means that the person has heart disease). Recall is the proportion of true positives to the number of terms that are actually positive (source: https://rafalab.github.io/dsbook/introduction-to-machine-learning.html) Precision and recall are also known as specificity and sensitivity respectively. A high specificity implies a higher number of true positives to false positives. False positives are dangerous as it may lead individuals to seek treatment they do not need. This is not only expensive, but also potentially harmful. A high sensitivity implies that there are more true positives than false negatives. False negatives are also dangerous since it may lead people at risk of heart disease, to **not** pursue treatment. This is also dangerous.

Hence, sensitivity and specificity are equally important. So, the F__1 score is not adjusted for either one.

The F1 score can be calculated using the F_meas function in the caret library.

Now, we can proceed with the first models.

Analysis

Since the datasets are very small, the caret library can be used for classification. The available models are here.

This analysis will use classification models, since regression will not work on factor variables.

Before training any algorithms, however. the crossvalidation parameters must be set.

```
#set crossvalidation parameters
# bootstrap sampling is used as dataset is small
# 20 fold cross validation with 15% as samples. 15% is just large enough for 20
    folds.
# percentage is amount used in training set, which is 1-0.15
control <- trainControl(method = 'boot', number = 20, p = 0.85)
```

First Models: GLM, LDA, QDA and Naive Bayes

The first model to be trained on is a logistic regression model (as a baseline)

```
# logistic model, family=binomial sets logistic regression

fit_glm <- train(num~., data=train_set_2, method = 'glm', trControl = control,
   family = 'binomial')
```

```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
```

```
# see result on debug set
# glm has no tuning parameters
predict_glm <- predict(fit_glm, newdata = debug_set_2)

# levels(debug_set_2$num)[2] is '1'
fmeas1 <- F_meas(data = predict_glm, reference = debug_set_2$num, relevant =
   levels(debug_set_2$num)[2])
print(fmeas1)
```

```
## [1] 0.8759124
```

This is a very good result! However, can more be done?

The next models that can be implemented are discriminant analyses models. First is LDA:

```
# lda model
fit_lda <- train(num~., data=train_set_2, method = 'lda', trControl = control)
```

```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
```

```
# see result on debug set

predict_lda <- predict(fit_lda, newdata = debug_set_2)

# levels(debug_set_2$num)[2] is '1'
fmeas2 <- F_meas(data = predict_lda, reference = debug_set_2$num, relevant =
   levels(debug_set_2$num)[2])
print(fmeas2)
```

```
## [1] 0.8759124
```

```
print(fmeas2 > fmeas1)
```

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

A slight improvement is observed. Now, qda can be tested:

```
# qda model
fit_qda <- train(num~., data=train_set_2, method = 'qda', trControl = control)
```

```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
```

```
# see result on debug set

predict_qda <- predict(fit_qda, newdata = debug_set_2)

# levels(debug_set_2$num)[2] is '1'
fmeas3 <- F_meas(data = predict_qda, reference = debug_set_2$num, relevant =
   levels(debug_set_2$num)[2])
print(fmeas3)
```

```
## [1] 0.8571429
```

We see that there is a drop in F1. This suggests that qda is not an effective classification model.

The next model that can be tested is Naive Bayes.

```
# naive bayes model
# modelLookup('naive_bayes') there are three tuning parameters
# DO NOT SET adjust to zero you cannot have 0 bin width
fit_nb <- train(num~., data=train_set_2, method = 'naive_bayes', trControl =
   control, tuneGrid = data.frame(expand.grid(laplace = seq(0,1,0.05),adjust =
   seq(0.05,0.75,0.05),usekernel = c(TRUE, FALSE))))
```

```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
```

```
# see result on debug set
# modelLookup('naive_bayes') there are three tuning parameters
predict_nb <- predict(fit_nb, newdata = debug_set_2)

# levels(debug_set_2$num)[2] is '1'
fmeas4 <- F_meas(data = predict_nb, reference = debug_set_2$num, relevant =
   levels(debug_set_2$num)[2])
print(fmeas4)
```

```
## [1] 0.8682171
```

A considerable increase from qda, albeit worse than logistic regression or lda.

The next models that can be considered are classification trees.

Classification Trees and Random Forests

The first method that can be tested is classification trees using CART, and the rpart library.

```
# modelLookup('rpart') there is one tuning parameter
# classification trees model
fit_cart <- train(num~., data=train_set_2, method = 'rpart', trControl =
   control, tuneGrid = data.frame(cp = seq(0.1, 1.0, 0.05)))
```

```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
```

```
# see result on debug set

predict_cart <- predict(fit_cart, newdata = debug_set_2)

# levels(debug_set_2$num)[2] is '1'
fmeas5 <- F_meas(data = predict_cart, reference = debug_set_2$num, relevant =
   levels(debug_set_2$num)[2])
print(fmeas5)
```

```
## [1] 0.8244275
```

This is the worst result, compared to all previous models. However, rpart uses only one decision tree to arrive at a result. To use more than one, the rf (Random Forest) method can be used.

```
# modelLookup('rf') there is one tuning parameter
# random forest model
# based on help documentatation, rf randomly samples variables in each split
    using mtry
fit_rf <- train(num~., data=train_set_2, method = 'rf', trControl = control,
    tuneGrid = data.frame(mtry = seq(1, (ncol(train_set_2) - 1),1)))
```

```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
```

```
# see result on debug set

predict_rf <- predict(fit_rf, newdata = debug_set_2)

# levels(debug_set_2$num)[2] is '1'
fmeas6 <- F_meas(data = predict_rf, reference = debug_set_2$num, relevant =
    levels(debug_set_2$num)[2])
print(fmeas6)
```

```
## [1] 0.8695652
```

This is the best result, so far.

This can also be trained on the ranger library, which has a greater number of options.

```
# modelLookup('ranger') there is one tuning parameter
# ranger random forest model
# based on help documentatation, rf randomly samples variables in each split
    using mtry
# split rule is gini, extratrees or hellinger for classification (ranger also
    supports regression)
# min.node.size is the minimum size of a decision tree node (possibly to
    determine whether more splits should be done?)
tgrid <- data.frame(expand.grid(mtry = seq(1, (ncol(train_set_2) - 1),1),
    splitrule = c('gini', 'extratrees', 'hellinger'),min.node.size = seq
    (10,20,2)))

fit_ranger <- train(num~., data=train_set_2, method = 'ranger', trControl =
    control, tuneGrid = tgrid)
```

```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
```

```
# see result on debug set

predict_ranger <- predict(fit_ranger, newdata = debug_set_2)
```

```
# levels(debug_set_2$num)[2] is '1'
fmeas7 <- F_meas(data = predict_ranger, reference = debug_set_2$num, relevant =
     levels(debug_set_2$num)[2])
print(fmeas7)
```

```
## [1] 0.8823529
```

```r
print(fmeas7 == fmeas6)
```

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

A substantial result, albeit exactly the same as the original random forests implementation.

Another such implementation of random forests is the bagged adaboost implementation in the adabag library.

```r
# This takes time
# modelLookup('adabag') there are three tuning parameters
# adabag random forest model

fit_adaboost <- train(num~., data=train_set_2, method = 'AdaBag', trControl =
    control, tuneGrid = data.frame(expand.grid(mfinal = 200,maxdepth = 1)))
```

```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
```

```r
# see result on debug set

predict_adaboost <- predict(fit_adaboost, newdata = debug_set_2)

# levels(debug_set_2$num)[2] is '1'
fmeas8 <- F_meas(data = predict_adaboost, reference = debug_set_2$num, relevant
     = levels(debug_set_2$num)[2])
print(fmeas8)
```

```
## [1] 0.8244275
```

This is not as effective as previous models. So, further testing on other decision tree models is not a viable option. However, there is one more method that can be tested: neural networks.

Neural Networks

What is a neural network? A neural network is an algorithm that attempts to replicate neurons in the brain. It uses multiple units to replicate non-linear outputs with a linear functions. It is comprised of multiple nodes:

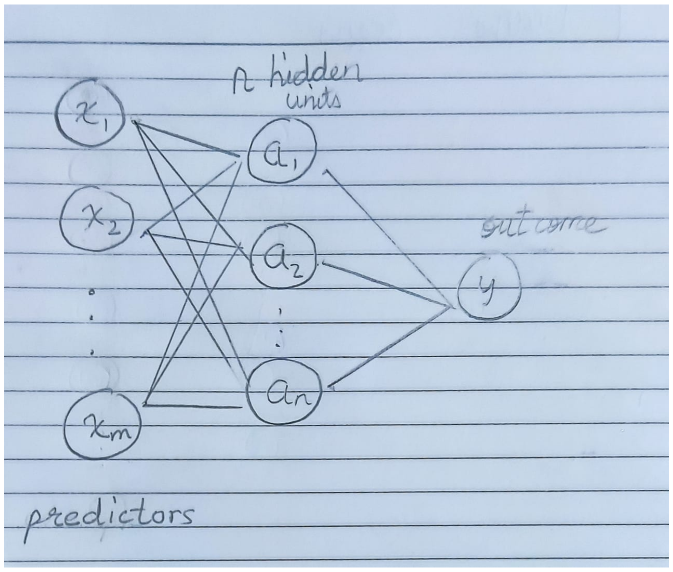


Figure 1: image of neural network

Given above is a diagram of a neural network. It contains the predictors $x_1, x_2, ..., x_m$, an outcome y and a hidden layer $a_1, a_2, ..., a_n$ in between the two.

The lines point from one node to the next, showing how the next layer is computed using values from the previous layer.

The values of the previous layer are multiplied by weights, added, and then input into the next layer.

For example

$$a_1 = f(\theta_{1,1}x_1 + \theta_{1,2}x_2 + ... + \theta_{1,m}x_m)$$

Where $\theta_{n,m}$ is the weight of the mth value of x, for the nth node of a. f is an activation function. Generally, the sigmoid function is used, as it has a range of 0 to 1, which is especially helpful for determining outcomes/binary probabilities for multi-class outcomes.

The above diagram assumes only one hidden layer, but there can be more layers. The nnet package allows for one hidden layer with a variable number of units, which allow for more complexity.

Neural Networks can be trained using the *nnet* method from the train function (note that the below code takes a while to train):

```r
# modelLookup('nnet') there are two tuning parameters
# Neural Network
# preprocess to center and scale numerics for neural networks
suppressWarnings(fit_nn <- train(num~., data=train_set_2, method = 'nnet',
   trControl = control, preProcess = c('center', 'scale'),tuneGrid = data.frame
   (expand.grid(size = seq(10,50,10), decay = seq(0.1,1,0.1)))))
# see result on debug set

predict_nn <- predict(fit_nn, newdata = debug_set_2)
```

```r
# levels(debug_set_2$num)[2] is '1'
fmeas9 <- F_meas(data = predict_nn, reference = debug_set_2$num, relevant =
   levels(debug_set_2$num)[2])
print(fmeas9)
```

```
## [1] 0.8613139
```

Neural networks provide a better result than other models such as naive bayes or CART. However, it is still not at the same level as random forests.

Another method can be implemented: support vector machines.

Support Vector Machines (SVM)

SVMs are built for nonlinear decision boundaries, and have increased computational efficiency, compared to neural networks. Note that centering and scaling numerical values is necessary for SVMs and Neural Networks so that less iterations (since these are iterative learning algorithms) are spent normalizing the values and more iterations are spent training.

```r
# SVMs
# first, linear boundary
# modelLookup('svmLinear') there is one tuning parameter, the cost
# cost of constraints violation (default: 1) this is the âĂŸCâĂŹ-constant of
   the regularization term in the Lagrange formulation.

# explanation of cost given in help file ^^
# also requires preprocess
```

```r
fit_svml <- train(num~., data=train_set_2, method = 'svmLinear', trControl =
    control, preProcess = c('center', 'scale'), tuneGrid = data.frame(C = seq
    (1,10,1)))
```

```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
```

```r
# see result on debug set

predict_svml <- predict(fit_svml, newdata = debug_set_2)

# levels(debug_set_2$num)[2] is '1'
fmeas10 <- F_meas(data = predict_svml, reference = debug_set_2$num, relevant =
    levels(debug_set_2$num)[2])
print(fmeas10)
```

```
## [1] 0.8759124
```

A considerable result, given a linear decision boundary. However, SVM also supports non linear kernels (similar to kernels in smoothing algorithms).

```r
# SVMs
# polynomial kernel
# modelLookup('svmPoly') there are three tuning parameters, the cost, the
    degree and the scale
# cost of constraints violation (default: 1) this is the âĂŸCâĂŹ-constant of
    the regularization term in the Lagrange formulation.
# degree is degree of polynomial kernel: 1,2,3 etc

# explanation of cost given in help file ^^

fit_svmp <- train(num~., data=train_set_2, method = 'svmPoly', trControl =
    control, preProcess = c('center', 'scale'), tuneGrid = data.frame(expand.
    grid(C = seq(1,10,1), degree = c(2:5), scale = seq(0.1,1,0.1))))
```

```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
```

```r
# see result on debug set

predict_svmp <- predict(fit_svmp, newdata = debug_set_2)

# levels(debug_set_2$num)[2] is '1'
fmeas11 <- F_meas(data = predict_svmp, reference = debug_set_2$num, relevant =
    levels(debug_set_2$num)[2])
print(fmeas11)
```

```
## [1] 0.8333333
```

```r
# fmeas11 < fmeas10 is true so it is a step down.
```

The F_1 score has dropped significantly from the previous model. However, SVM also supports a third kernel: the radial basis function.

```r
# SVM with radial basis function kernel (think bin smoothing with gaussian
    kernel but it's an SVM)

# modelLookup('svmRadial') there are two tuning parameters, the cost, and sigma
# cost of constraints violation (default: 1) this is the âĂŸCâĂŹ-constant of
   the regularization term in the Lagrange formulation.


# explanation of cost given in help file ^^
# optimal values of the sigma width parameter are shown to lie in between the
   0.1 and 0.9 quantile of the \|x- x'\| statistics, also from help file
fit_svmr <- train(num~., data=train_set_2, method = 'svmRadial', trControl =
   control, preProcess = c('center', 'scale'), tuneGrid = data.frame(expand.
   grid(C = seq(1,10,1), sigma = seq(0.1,0.9,0.05))))
```

```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
```

```
# see result on debug set

predict_svmr <- predict(fit_svmr, newdata = debug_set_2)

# levels(debug_set_2$num)[2] is '1'
fmeas12 <- F_meas(data = predict_svmr, reference = debug_set_2$num, relevant =
   levels(debug_set_2$num)[2])
print(fmeas12)
```

```
## [1] 0.893617
```

This is the best result of all of the above methods.

K-Nearest Neighbours

Besides these techniques, a smoothing method can also be considered, such as k-nearest neighbours:

```
# k nearest neighbours

# modelLookup('knn') there is one tuning parameter, k, which is the k closest
   values that will be selected when smoothing
# There are 679 training examples, so ks between 5 to 400 can be used

fit_knn <- train(num~., data=train_set_2, method = 'knn', trControl = control,
   tuneGrid = data.frame(k = seq(5,400,5)))
```

```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
```

```
# see result on debug set

predict_knn <- predict(fit_knn, newdata = debug_set_2)

# levels(debug_set_2$num)[2] is '1'
fmeas13 <- F_meas(data = predict_knn, reference = debug_set_2$num, relevant =
   levels(debug_set_2$num)[2])
print(fmeas13)
```

```
## [1] 0.6849315
```

This is a very poor result, suggesting that smoothing methods are not an effective solution.

Ensembling

While most of these methods provided a range of F1 scores, from 0.8 to 0.9, their combined effectiveness may be better, or worse. This method of combining models is also known as ensembling. First, all the models will be ensembled together:

```
# Results table for later use

results_table <- tibble(model = c("Logistic␣Regression", "LDA", "QDA", "Naive␣
    Bayes", "CART", "Random␣Forests␣-␣randomForest", "Random␣Forests␣-␣ranger",
    "Random␣Forests␣-␣adaboost", "Neural␣Network", "SVM␣-␣Linear␣Kernel", "SVM␣-
    ␣Polynomial␣Kernel", "SVM␣-␣Radial␣Basis␣Kernel", "K-Nearest␣Neighbours"),
    f1 = c(fmeas1,fmeas2,fmeas3,fmeas4,fmeas5,fmeas6,fmeas7,fmeas8,fmeas9,
    fmeas10,fmeas11,fmeas12, fmeas13), result = list(predict_glm, predict_lda,
    predict_qda, predict_nb, predict_cart, predict_rf, predict_ranger, predict_
    adaboost, predict_nn, predict_svml, predict_svmp, predict_svmr, predict_knn)
    )

# results_ensembling contains each model's predictions as one column
foo <- results_table$result
names(foo) <- results_table$model

results_ensembling <- data.frame(foo)

# to show that each column matches the respective model, you can run this code

# test <- apply(results_ensembling,2,function(x)F_meas(as.factor(x), debug_set_
    2$num, relevant = levels(debug_set_2$num)[2]))
# test <- unname(test)
# identical(test, results_table$f1)
# this returns true so the column names match the results vectors
# # Best model
# results_table$model[which.max(results_table$f1)]
```

As there are 13 models, (and the factors are zero and one), if we take the mean of any row, it will give the proportion of models that return one. If that mean is greater than 0.5, then the ensemble will return one. For example, in a list $1, 1, 1, 0, 0, 0$ the proportion of ones is 50%, and the mean of the vector is $\frac{1+1+1+0+0+0}{6}$, which is also 0.5. (This also works with other vectors solely comprised of zeros and ones). So, the ensemble can be made by converting the *results_ensembling* table into a matrix and calculating the row means.

```
# convert results_ensembling into a matrix
# first convert all factors to numerics
results_ensembling_temp <- apply(results_ensembling,2,function(x)as.numeric(as.
    character(x))) %>% as.data.frame()


results_ensembling_matrix <- as.matrix(results_ensembling_temp)
results_ensembling_means <- rowMeans(results_ensembling_matrix)

predict_ensemble_all <- ifelse(results_ensembling_means >= 0.5, 1, 0) %>% as.
    factor()

fmeas14 <- F_meas(data = predict_ensemble_all, reference = debug_set_2$num,
    relevant = levels(debug_set_2$num)[2])
print(fmeas14)
```

```
## [1] 0.8571429
```

This is a very good result, since it is able to balance better and poorer performing models.

Ensembling can be done with only a selection of columns as well. Now, only the three best models shall be considered for ensembling.

```
# top 3 models
dplyr::arrange(results_table %>% dplyr::select(-result), dplyr::desc(f1)) %>%
   head(3) %>% knitr::kable()
```

model	f1
Random Forests - ranger	0.8823529
Random Forests - randomForest	0.8759124
SVM - Radial Basis Kernel	0.8732394

The best three models are the random forest implementations by randomForest and ranger, and the Support Vector Machine with a Radial Basis Kernel.

```
# select the best three models

results_ensembling_top3 <- results_ensembling_temp %>% dplyr::select(Random.
   Forests...randomForest, Random.Forests...ranger, SVM...Radial.Basis.Kernel)

results_ensembling_mat_top3 <- as.matrix(results_ensembling_top3)

results_ensembling_means_top3 <- rowMeans(results_ensembling_mat_top3)

predict_ensemble_top3 <- ifelse(results_ensembling_means_top3 >= 0.5, 1,0) %>%
   as.factor()

fmeas15 <- F_meas(data = predict_ensemble_top3, reference = debug_set_2$num,
   relevant = levels(debug_set_2$num)[2])
print(fmeas15)
```

```
## [1] 0.8759124
```

This is a substantial result. However, it does not match the effectiveness of the Support Vector Machine.

Results

Having tested all of these models, they can now be compared:

```
# Results table for debug set:

results_table_debug <- tibble(model = c("Logistic␣Regression", "LDA", "QDA", "
   Naive␣Bayes", "CART", "Random␣Forests␣-␣randomForest", "Random␣Forests␣-␣
   ranger", "Random␣Forests␣-␣adaboost", "Neural␣Network", "SVM␣-␣Linear␣Kernel
   ", "SVM␣-␣Polynomial␣Kernel", "SVM␣-␣Radial␣Basis␣Kernel", "K-Nearest␣
   Neighbours", "Ensembling␣-␣All␣Models", "Ensembling␣-␣Top␣3␣Models"), f1 = c
   (fmeas1,fmeas2,fmeas3,fmeas4,fmeas5,fmeas6,fmeas7,fmeas8,fmeas9,fmeas10,
   fmeas11,fmeas12, fmeas13, fmeas14, fmeas15))

results_table_debug %>% knitr::kable()
```

model	f1
Logistic Regression	0.8444444
LDA	0.8507463
QDA	0.8461538
Naive Bayes	0.8527132
CART	0.8244275
Random Forests - randomForest	0.8759124
Random Forests - ranger	0.8823529
Random Forests - adaboost	0.8244275
Neural Network	0.8613139
SVM - Linear Kernel	0.8507463
SVM - Polynomial Kernel	0.8382353
SVM - Radial Basis Kernel	0.8732394
K-Nearest Neighbours	0.6853147
Ensembling - All Models	0.8571429
Ensembling - Top 3 Models	0.8759124

We can now look at the best-performing model

```
print(results_table_debug$model[which.max(results_table_debug$f1)])
```

```
## [1] "Random Forests - ranger"
```

This can now be tested on the final test set:

```
# create prediction vectors
predict_svmr_test <- predict(fit_svmr, newdata = test_set_2)

# F1 score

fmeas16 <- F_meas(data = predict_svmr_test, reference = test_set_2$num,
   relevant = levels(test_set_2$num)[2])
print(fmeas16)
```

```
## [1] 0.88
```

This is a very good result.

Now, we can look at all of the results:

```
results_table_final <- tibble(model = c("Logistic Regression", "LDA", "QDA", "
   Naive Bayes", "CART", "Random Forests - randomForest", "Random Forests -
   ranger", "Random Forests - adaboost", "Neural Network", "SVM - Linear Kernel
   ", "SVM - Polynomial Kernel", "SVM - Radial Basis Kernel", "K-Nearest
   Neighbours", "Ensembling - All Models", "Ensembling - Top 3 Models", "Best
   Model (SVM, Radial Basis Kernel) on Test Set"), f1 = c(fmeas1,fmeas2,fmeas3,
   fmeas4,fmeas5,fmeas6,fmeas7,fmeas8,fmeas9,fmeas10,fmeas11,fmeas12, fmeas13,
   fmeas14, fmeas15, fmeas16))

results_table_final %>% knitr::kable()
```

model	f1
Logistic Regression	0.8759124
LDA	0.8759124
QDA	0.8571429
Naive Bayes	0.8682171

model	f1
CART	0.8244275
Random Forests - randomForest	0.8695652
Random Forests - ranger	0.8823529
Random Forests - adaboost	0.8244275
Neural Network	0.8613139
SVM - Linear Kernel	0.8759124
SVM - Polynomial Kernel	0.8333333
SVM - Radial Basis Kernel	0.8936170
K-Nearest Neighbours	0.6849315
Ensembling - All Models	0.8571429
Ensembling - Top 3 Models	0.8759124
Best Model (SVM, Radial Basis Kernel) on Test Set	0.8800000

Conclusion

This analysis used a range of classification models, such as neural networks, support vector machines, logistic regression, and random forests to predict the possibility of heart disease. The final model on the testing set used a Support Vector Machine with a Radial Basis Kernel, to give a final F_1 score of 0.88.

This model can be used as an aide to gauge whether someone has heart disease or not, which is very useful in the field of medicine. This tool, however, is not a substitute for a diagnosis.

This research also used NA imputation methods via the *mice* library to ensure easier training of models. (As many models cannot handle NAs)

The main limitation of this research was the dataset size. Although this issue was alleviated using methods such as bootstrap sampling, the training set only has 679 training examples, out of 920 from all four datasets. This could be improved by collecting more data for training/testing.

Another limitation was the need for imputing NAs. As there were many NAs, this may have limited the effectiveness of the model on some examples. Other imputation methods could be implemented for further research.

The dataset was originally multi-class, so further research could involve creating models using all four classes. Other classification models could also be considered when training.

Bibliography

1. Hungarian Institute of Cardiology. Budapest: Andras Janosi, M.D.
2. University Hospital, Zurich, Switzerland: William Steinbrunn, M.D.
3. University Hospital, Basel, Switzerland: Matthias Pfisterer, M.D.
4. V.A. Medical Center, Long Beach and Cleveland Clinic Foundation: Robert Detrano, M.D., Ph.D.

https://ourworldindata.org/causes-of-death

https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/

https://rafalab.github.io/dsbook/introduction-to-machine-learning.html

https://topepo.github.io/caret/available-models.html

https://bookdown.org/yihui/rmarkdown-cookbook/text-width.html

https://ctan.asis.ai/macros/latex/contrib/listings/listings.pdf