Blood Infusion

Manuela Werkle

4th July 2021

1. Introduction

1.1 Description of the dataset

I found this dataset "Blood Transfusion Service Center" on https://archive-beta.ics.uci.edu/ml/datasets/176. It was published 2008 by creator Prof. I-Cheng Yeh with this published paper: Yeh, I-Cheng, Yang, King-Jang, and Ting, Tao-Ming, "Knowledge discovery on RFM model using Bernoulli sequence,"Expert Systems with Applications, 2008 (doi:10.1016/j.eswa.2008.07.018)

It is licensed under a Creative Commons Attribution 4.0 International (CC BY 4.0) license, which allows for the sharing and adaptation of the datasets for any purpose, provided that the appropriate credit is given. Starting the project on 4th July, it had 27 #Views mentioned on the homepage. This agrees with the requirement of using a publicly available dataset which is not well-known or used as example in previous courses.

The original dataset contains of 748 rows and 5 columns, where I will later change the column names to have it a little bit easier in typing. The columns are:

original column name	my column name	explanation
Recency..months.	recency	number of months since the last donation of this person
Frequency..times.	frequency	total number of donations, which this person made
Monetary..c.c..blood	monetary	total volume of all donations done by this person in unit "cc"
Time..months.	time	number of months since first donation of this person
whether.he.she.donated.blood.in.March.2007	donation	value, if the person donated blood at this donation day in March 2007 (0 stands for "no" and 1 stands for "yes")

Every row stands for an individual person. Because there's no indicator about these person, the dataset is fully anonymized.

1.2 Summary of the goal of the project and key steps that were performed

This data set is used for solving a classification problem using machine learning techniques, corresponding models and algorithms.

The goal of this project is to predict, if the person would come to the next blood donation again. This is done based on a data set, which represents a list of different persons regarding their former blood donations.

Overall we can say, that the goal of this projects stands for a prediction of a repetition of a customer behavior. So it can be seen as an example for an analysis regarding "Customer Value", where the result can be used as basis of decisionmaking for advertising.

I will split this dataset into train and test set using 75% / 25% ratio. Normally I would recommend using another ratio to have as much training data as possible, but as the data set is not that big in this case we also need enough data for our final test set. After the splitting we have 599 rows in our train data set and 149 in our test data set. For the training and validation phase we furthermore divide the train set into train and validation set. We use the same ratio 75% / 25% and in the end we then have a train set containing 450 rows and a validation set of 149 rows.

After data splitting, I will do some data exploration. This is necessary to have first key insights into the data. Another important step in my report will be the data cleaning and preparation for modeling. For this I will check for unnecessary data / features and on the other hand also create new columns = new features in the data set, which can be used for later analysis.

2. Methods and Analysis

2.1 Process and techniques used

First I will explore the data by calculating summary and creating visualizations of the data. Based on these points I will identify potential relationships and characteristics of the data. After the data exploration, I will create 5 models to predict, if a person will come to the next blood donation.

I will use the "No-free-Lunch" theoreme, which means, that it makes no sense to use an algorithm from the past again and again. It makes more sense to use several difference models. In this process it's good to start with several easy models instead of choosing complex ones. Because also easier models can have a good result.

The following 5 models will be used in my project:

1. Decision Trees: Decision Trees can be used for regression and classification. The advantage is, that it's easy to understand and easy to interprete.
2. Random Forest: Random Forest is a well-known model which can also be used for regression and classification and so it's quite popular. It's a combination of several decision trees and all trees are built in parallel. That's why this model is quite fast. Furthermore it's flexible, easy to understand and comprehensible.
3. Logistic Regression: I will use this model, because it's easy to understand, fast and a typical model used for classification problems.
4. kNN: I will use kNN because it's fast for small data sets and it can be used for both regression and classification. For large data sets it can be quite computating-intensive, because the prediction of new values need a calculate of the distances to all other values. But my data set is not that big, so that should be fine.
5. Neuronal Network: I will use Neuronal Network because it's a really effective model and can be used for nearly all data, also for not linear prediction, regression and classification (=Supervised Learning). Also Clustering is possible (Unsupervised Learning).

2.2 Data exploration and data visualization

To have a first overview of the data set I will explore the data and create some visualizations to gain some insights. To make reading a little bit easier, I will first change the column names.

```
# change the originals columns names to make it easier for typing
colnames(train) <- c("recency", "total_frequency", "total_monetary","time","donation")
colnames(test) <- c("recency", "total_frequency", "total_monetary","time","donation")
```

2.2.1 Some general data analysis To have a first overview of the data I will create a general overview and then go deeper into the different columns:

```
nrow(train) # count rows in train data set
```

```
## [1] 599
```

The train data set has got 599 rows. In addition we can find the following main key values when looking at the summary function:

```
summary(train) # Shows first summary of train data
```

```
##     recency       total_frequency  total_monetary       time      
##  Min.   : 0.000   Min.   : 1.000   Min.   :  250   Min.   : 2.00  
##  1st Qu.: 3.000   1st Qu.: 2.000   1st Qu.:  500   1st Qu.:16.00  
##  Median : 7.000   Median : 4.000   Median : 1000   Median :28.00  
##  Mean   : 9.451   Mean   : 5.623   Mean   : 1406   Mean   :34.71  
##  3rd Qu.:14.000   3rd Qu.: 7.000   3rd Qu.: 1750   3rd Qu.:49.50  
##  Max.   :74.000   Max.   :50.000   Max.   :12500   Max.   :98.00  
##     donation     
##  Min.   :0.0000  
##  1st Qu.:0.0000  
##  Median :0.0000  
##  Mean   :0.2387  
##  3rd Qu.:0.0000  
##  Max.   :1.0000
```

```
str(train) # Shows all data types of train data
```

```
## 'data.frame':    599 obs. of  5 variables:
##  $ recency        : num  2 0 1 2 1 4 2 1 2 4 ...
##  $ total_frequency: int  50 13 16 20 24 4 7 12 9 23 ...
##  $ total_monetary : int  12500 3250 4000 5000 6000 1000 1750 3000 2250 5750 ...
##  $ time           : num  98 28 35 45 77 4 14 35 22 58 ...
##  $ donation       : int  1 1 1 1 0 0 1 0 1 0 ...
```

The columns have got different data types. The data types of the columns total_frequency and total_monetary need to be changed to numeric to make calculations (mean, average) possible. Furthermore donation needs to be changed to type factor for later modeling purposes.

```r
# change data types for the different columns to make calculations and analysis easier
train <- as.data.frame(train) %>% mutate(total_frequency = as.numeric(total_frequency),
                                         total_monetary = as.numeric(total_monetary),
                                         donation = as.factor(donation))
test <- as.data.frame(test) %>% mutate(total_frequency = as.numeric(total_frequency),
                                         total_monetary = as.numeric(total_monetary),
                                         donation = as.factor(donation))
```

The new column types look like this:

```r
str(train)
```

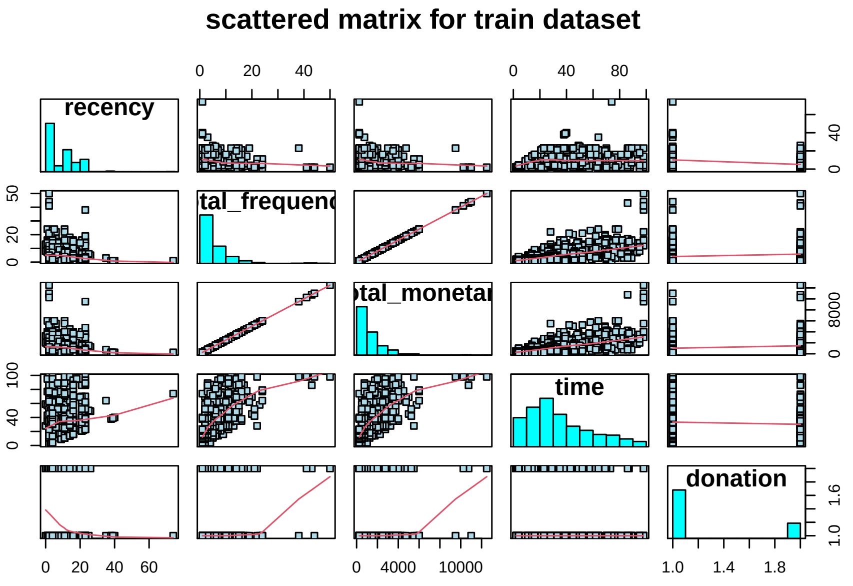
```
## 'data.frame':    599 obs. of  5 variables:
##  $ recency        : num  2 0 1 2 1 4 2 1 2 4 ...
##  $ total_frequency: num  50 13 16 20 24 4 7 12 9 23 ...
##  $ total_monetary : num  12500 3250 4000 5000 6000 1000 1750 3000 2250 5750 ...
##  $ time           : num  98 28 35 45 77 4 14 35 22 58 ...
##  $ donation       : Factor w/ 2 levels "0","1": 2 2 2 2 1 1 2 1 2 1 ...
```

```r
str(test)
```

```
## 'data.frame':    149 obs. of  5 variables:
##  $ recency        : num  5 0 1 2 2 4 4 2 4 2 ...
##  $ total_frequency: num  46 3 13 6 6 6 9 6 8 10 ...
##  $ total_monetary : num  11500 750 3250 1500 1500 1500 2250 1500 2000 2500 ...
##  $ time           : num  98 4 47 16 16 16 28 22 26 49 ...
##  $ donation       : Factor w/ 2 levels "0","1": 2 1 1 2 2 2 2 1 2 1 ...
```

2.2.2 Correlations and relationships I try to identify relationships between different columns, also called features. The following scatterplot matrix is used to compare the features with each other:

```r
panel.hist <- function(x, ...) # scatterplot matrix to compare columns/features
{
  usr <- par("usr"); on.exit(par(usr))
  par(usr = c(usr[1:2], 0, 1.5) )
  h <- hist(x, plot = FALSE)
  breaks <- h$breaks; nB <- length(breaks)
  y <- h$counts; y <- y/max(y)
  rect(breaks[-nB], 0, breaks[-1], y, col = "cyan", ...)
}
pairs(train[1:5], main="scattered matrix for train dataset", panel = panel.smooth,
      cex = 1, pch = 22, bg = "light blue",
      diag.panel = panel.hist, cex.labels = 1.5, font.labels = 2)
```



Based on these plots we see, that there are columns, which seems to be correlated. We can see negative correlation for example between recency and total_frequency, moderate to strong correlations between time and total_frequency and a high correlation for example between frequency and total_monetary. This makes sense because every time you donate the same blood amount (in this case 500), which means that the more you go, the more blood you donate. This correlation can also be shown creating a correlation matrix:

```
cor(train[1:4]) # Show correlation matrix for all numeric features
```

```
##                    recency total_frequency total_monetary      time
## recency          1.0000000      -0.1901479     -0.1901479 0.1491981
## total_frequency -0.1901479       1.0000000      1.0000000 0.6323921
## total_monetary  -0.1901479       1.0000000      1.0000000 0.6323921
## time             0.1491981       0.6323921      0.6323921 1.0000000
```

The correlation value between total_frequency and monetary is 1. This means both columns are highly correlated and we do not need both. That's why I will now clean this and the column total_monetary will be deleted:

```
train[ , c("total_monetary")] <- list(NULL) # delete column total_monetary in train set
test[ , c("total_monetary")] <- list(NULL) # delete column total_monetary in test set
```

We will continue now with a table looking like this. You can see the first six entries as an example:

```
head(train) # show first 6 rows of train table
```

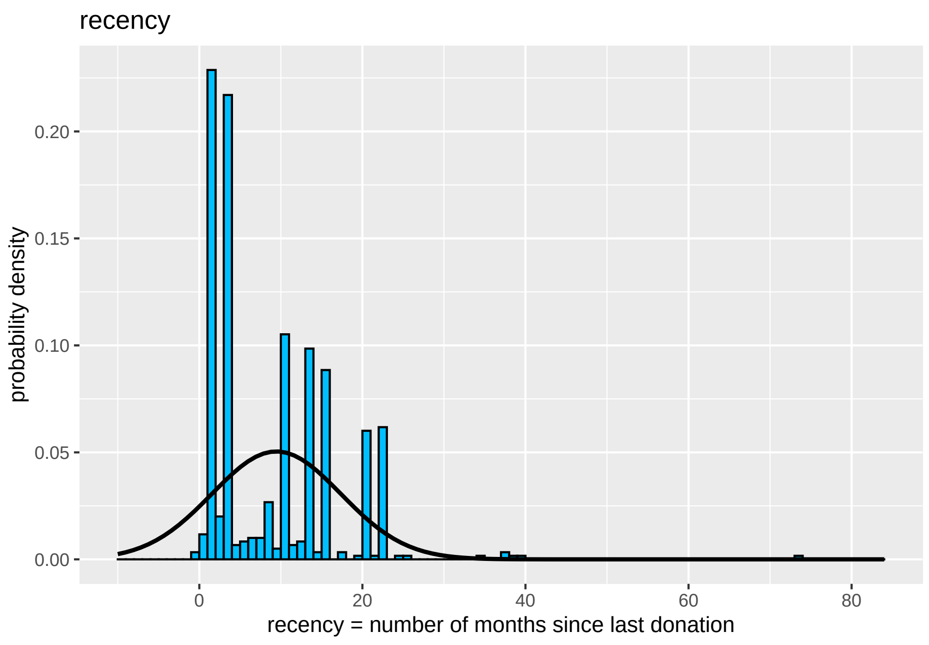
```
##   recency total_frequency time donation
## 1       2              50   98        1
## 2       0              13   28        1
## 3       1              16   35        1
## 4       2              20   45        1
## 5       1              24   77        0
## 6       4               4    4        0
```

Additionally We can also see in the scattered matrix, that we need to normalize some data, because we need to reduce skewness. This will be done later by scaling directly in the modeling function. (I will use `preProc = c("center", "scale")` for this.)

In the next sections, I will do some visualizations to better understand our data for analysis.

2.2.3 Analysing recency I will start our data analysis with the recency value (number of months since the last donation of a person):

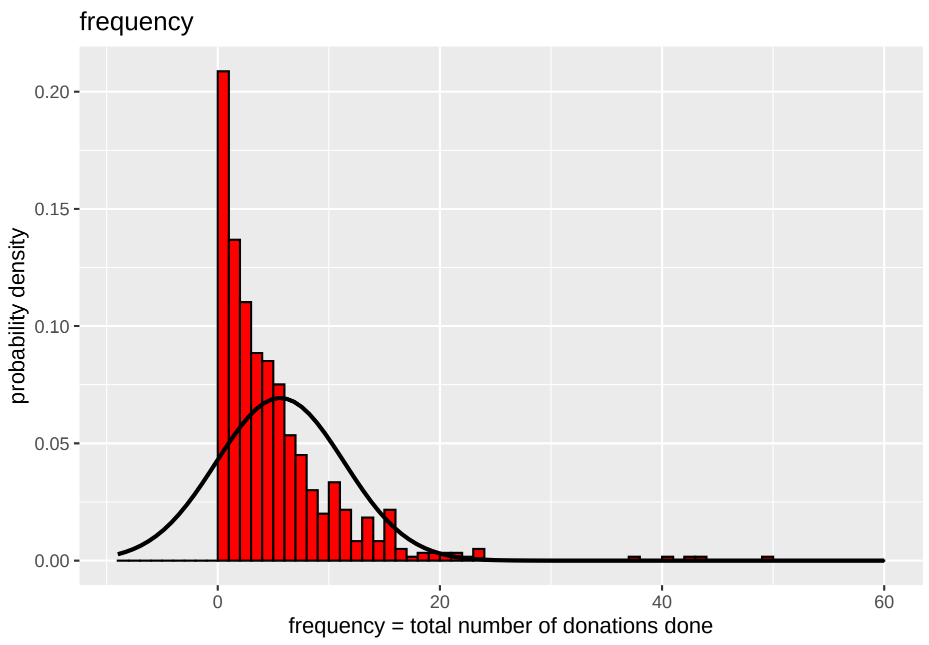
```
train %>%
  ggplot(aes(x = recency)) +
  geom_histogram(
    aes(y = stat(density)),
    breaks = seq(min(train$recency) - 10, max(train$recency) + 10),
    colour = "black",
    fill = "deepskyblue") +
  geom_function(
    fun = function(x) dnorm(x, mean(train$recency), sd(train$recency)),
    xlim = c(min(train$recency) - 10, max(train$recency) + 10),
    size = 1) +
  labs(
    title = "recency",
    x = "recency = number of months since last donation",
    y = "probability density"
  )
```



Based on the plot we can see, that there are many people who came regularly within the last year. There are only some outliers on the right side, which I will identify later.

2.2.4 Analysing total_frequency Next I will check total_frequency (total number of donations, which a person made):

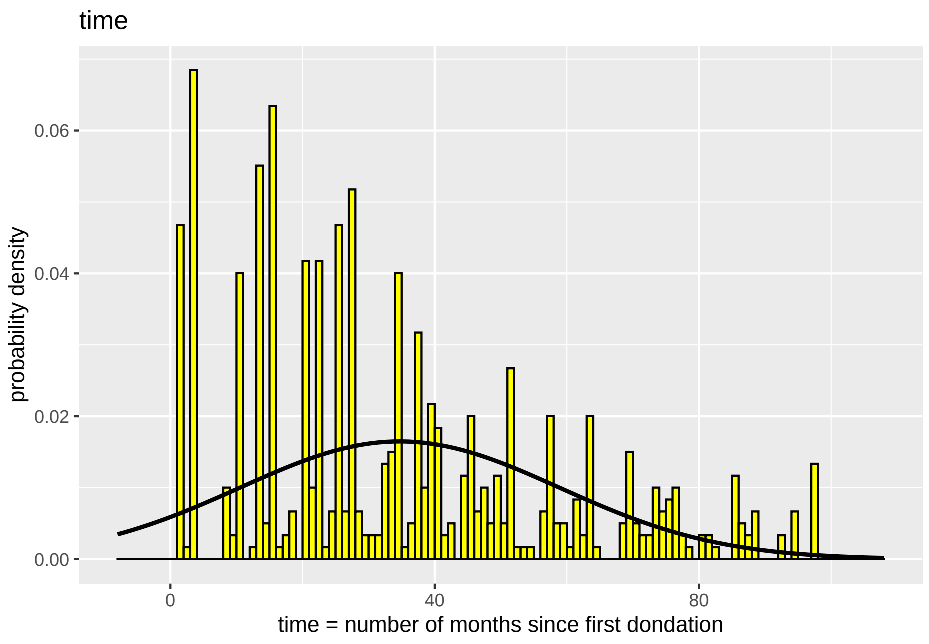
```
train %>%
  ggplot(aes(x = total_frequency)) +
  geom_histogram(
    aes(y = stat(density)),
    breaks = seq(min(train$total_frequency) - 10, max(train$total_frequency) + 10),
    colour = "black",
    fill = "red") +
  geom_function(
    fun = function(x) dnorm(x, mean(train$total_frequency), sd(train$total_frequency)),
    xlim = c(min(train$total_frequency) - 10, max(train$total_frequency) + 10),
    size = 1) +
  labs(
    title = "frequency",
    x = "frequency = total number of donations done",
    y = "probability density"
  )
```



We can see, that there are many people who donated only a few times. The more the number of donations increase, the less people are still donating. You can also see here outliers on the right side.

2.2.5 Analysing time Now I will check the time (number of months since first donation of a person):

```
train %>%
  ggplot(aes(x = time)) +
  geom_histogram(
    aes(y = stat(density)),
    breaks = seq(min(train$time) - 10, max(train$time) + 10),
    colour = "black",
    fill = "yellow") +
  geom_function(
    fun = function(x) dnorm(x, mean(train$time), sd(train$time)),
    xlim = c(min(train$time) - 10, max(train$time) + 10),
    size = 1) +
  labs(
    title = "time",
    x = "time = number of months since first dondation",
    y = "probability density"
  )
```



Also here you can see that most of the people do not donate since a very long time. The number of months since first donation shows us when the people donated the first time. The max. value here is:

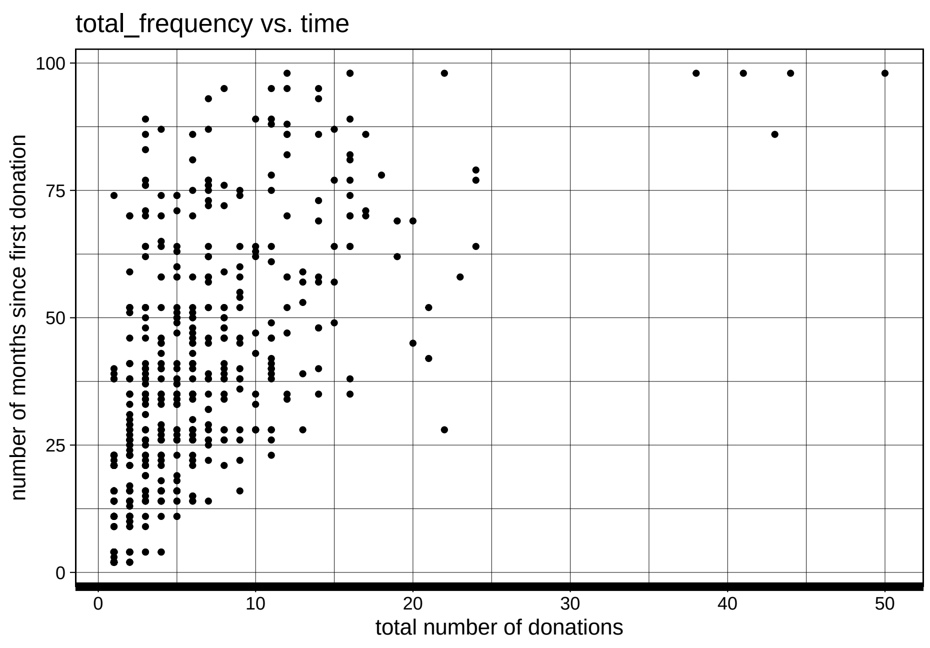
```
max(train$time)
```

```
## [1] 98
```

This means the first people started donating blood 98 months ago.

2.2.6 Analysing total_frequency vs. time I will now compare the total number of donations versus the number of months since first donatio:.

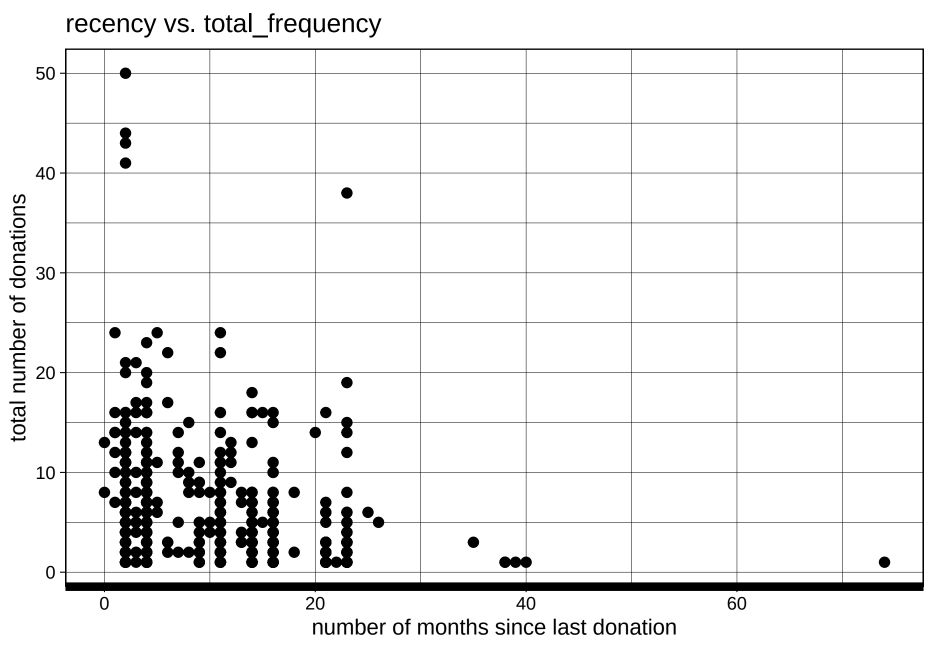
```
ggplot(train) +
  geom_point(aes(x = total_frequency, y = time),
             size = 1) +
  labs(title = "total_frequency vs. time",
       x = "total number of donations",
       y = "number of months since first donation") +
  theme_linedraw() +
  theme(axis.line.x = element_line(size = 2))
```



Based on this plot we see that there are many people coming quite often after they started donating the first time. We can calculate a donation rate to see if a person comes in short or larger time intervals: donation rate = total number of donations / number of months since first donation. I will add this information as a new column at a later stage.

2.2.7 Analysing recency vs. total_frequency I will compare now the number of months since last donation with the total number of donations:

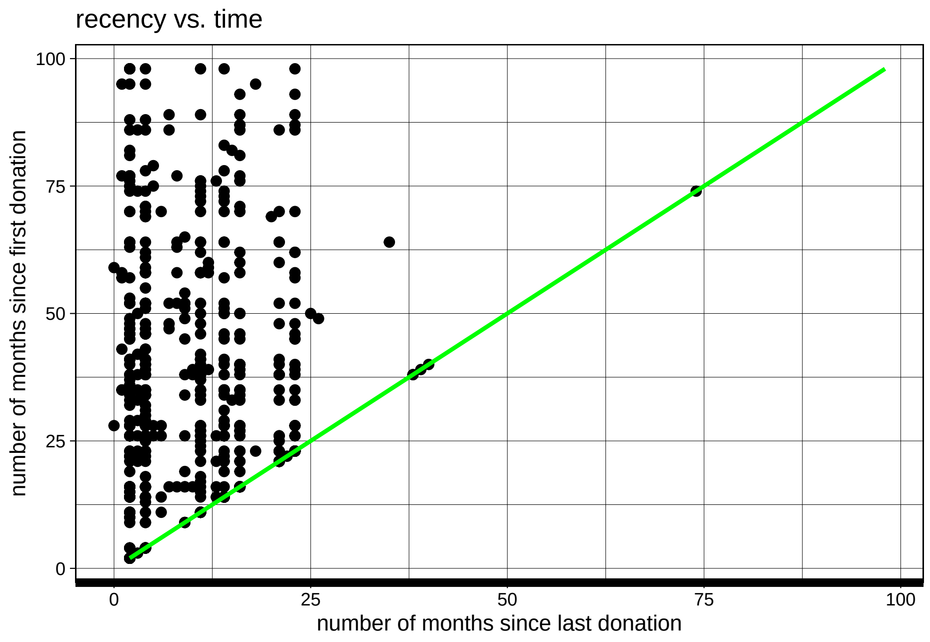
```r
ggplot(train) +
  geom_point(aes(x = recency, y = total_frequency),
             size = 2) +
  labs(title = "recency vs. total_frequency",
       x = "number of months since last donation",
       y = "total number of donations") +
  theme_linedraw() +
  theme(axis.line.x = element_line(size = 2))
```



We can identify here, if a persons comes periodically. We can calculate this "faithfulness": faithfullness = number of months since last donation / total number of donations. I will also add this information as a new column at a later stage.

2.2.8 Analysing recency vs. time Next I will compare the number of months since last donation versus the number of months since first donation:

```r
ggplot(train) +
  geom_point(aes(x = recency, y = time),
             size = 2) +
  geom_line(aes(x = time, y = time),
            color = "green",
            size = 1)+
  labs(title = "recency vs. time",
       x = "number of months since last donation",
       y = "number of months since first donation") +
  theme_linedraw() +
  theme(axis.line.x = element_line(size = 2))
```



We can see here, if a person perhaps stops coming to donations. People who only came once can be found on the green line here where both values are the same. We can also calculate this as an value: churn_rate = number of months since last donation & number of months since first donation. I will also add this information as a new column at a later stage.

2.2.9 Analysing donation If we check the donating results in the train data set, we can see that there are more people who did not donate compared to those, who donated:

```
summary(train$donation)
```

```
##   0   1
## 456 143
```

2.3 Data Cleaning

If we look at the data set, we recognize, that we need to clean some data.

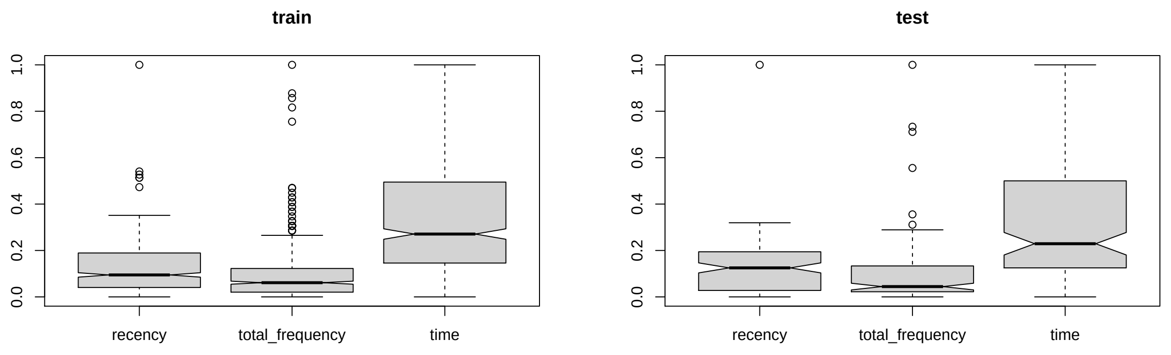
2.3.1 Check if data in train and test set are from the same population For a good and reliable modeling result it's important to have data in train and test set which is from the same population. This means that we need to check variance and also look for extreme values. We will do a visual check using boxplots for both data sets train and test. I normalized the date to have a better overview and comparison:

```
# definition of a normalization function
norm <- function(x){
            return((x-min(x)) / (max(x) - min(x)))
}

#apply norm function to columns 1-3 in the train and test data set
train_norm <- as.data.frame(lapply(train[1:3], norm))
test_norm <- as.data.frame(lapply(test[1:3], norm))
```

```
# plot the overview based on norm function
boxplot(train_norm,
        main="train",
        notch=TRUE,
        na.action=NULL,
        drop=FALSE,
        lex.order=FALSE)
boxplot(test_norm,
        main="test",
        notch=TRUE,
        na.action=NULL,
        drop=FALSE,
        lex.order=FALSE)

rm(train_norm, test_norm) # remove again to clear workspace
```



Based on the plot we see that both data sets seems to be similar and so we can assume that they are indeed from the same population.

2.3.2 Outlier detection - check for extreme values In above boxplots we saw that there might be some extreme values. Extreme values can have a negative influence in the modeling process. That's why we need to recognize them in the train data set.

```
# Check outliers recency
boxplot.stats(train$recency)$out # check for outliers in recency columns
```

```
## [1] 35 38 38 40 74 39
```

```
# Check outliers total_frequency
boxplot.stats(train$total_frequency)$out # check for outliers in total_frequency columns
```

```
##  [1] 50 16 20 24 23 15 16 20 19 16 16 17 17 17 24 15 22 18 16 16 16 38 16 15 43
## [26] 22 44 41 21 21 16 15 24 16 16 16 16 15 19
```

```
# Check outliers time
boxplot.stats(train$time)$out # check for outliers in time columns
```

```
## numeric(0)
```

I can identify some outliers in the data set, but to my point of view these are real data. There's no indication, that it's a kind of typing error or mistake. That's why I will leave them in.

2.3.3 Check for duplicates For good data quality it's also necessary to check, if there are duplicate values. If so, they should be removed.

```
nrow(train) # shows amount of all rows in train set
```

```
## [1] 599
```

```
nrow(distinct(train)) # shows amount of all distinct rows in train set
```

```
## [1] 446
```

We can see, that there are 155 duplicate rows, which would normally be removed in the train set using this code

```
#train <- distinct(train) # take all distinct values and save them back in train
```

BUT: The data set contains of different entries where one row stands for one single person. It can of course happen that coincidentally two people have the same values. That's the reason why I will also let these values in and I won't delete them.

2.4 Insights based on the data exploration

1. Checking all the columns from the source file, we saw that not all features are relevant. Total_monetary and total_frequency were highly correlated and indicating the same behavior: If somebody goes to a donation more often he/she will also have collected more blood in total. That's why I removed total_monetary from my table.
2. There are no high differences in the variables (scalability).
3. There's no missing data, so we do not need to think about imputation.
4. There are some outliers where we can see a deviation from the other data. But they are seen as real data and so they won't be removed.
5. There were some duplicate values in our train data set which could be removed to have a more clean data set. But as these are individual persons for each row, I will let them in the data set.
6. Regarding the column 'donation' we can see a big difference between people who are donating (value = 1) and people who are not donating (value = 0). The amount of people who are not donating is almost 4 times as high.
7. Some variables have got skewed distributions, for example recency.
8. The smallest value (besides 0) for the amount of months since last donation is 1. This means, that the donations are most probably done on a monthly basis.
9. Some values have a high variance. So we need to normalize them for our calculations..

2.5 Final data check

Before starting modelling I will check, if there are any NA's in our table:

```
anyNA(train) # Check for NA's in train data set
```

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

```
anyNA(test) # Check for NA's in test data set
```

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

The result is FALSE, which means, that all values are complete.

3 Modeling approach

For modeling it's necessary to prepare the data set. First I will divide the train data set into a train and validation data set. This is necessary to have an independent test data set for final prediction and to avoid overfitting. Models will be trained based on the train data set and the prediction check will be done on the validation data set. The test set will only be used for getting final results.

As metric I choose logLoss, which is a classificaton metric based on probabilities and its objective is minimization.

Afterwards I will choose several models for a first evaluation, compare the results and try to optimize them by using several optimization techniques. The models here are Decision Trees, Random Forest, Logistic Regression (GLM), k-nearest neighbor (kNN) and Neuronal Network.

3.1 Preparation for modeling

Using the caret package, I will now test several models to compare and check the results based on the "No-free-Lunch" theoreme. This means, that the result of solving a problem is statistically identical, although using different kinds of methods. Because of that, I will compare some models first and then choose models for further optimization and the final solution.

```
# I will change the factor value: "0 -> no", "1 -> yes" to make it more clear and to
# avoid mistakes by confusion of data
# change levels of factors from 0 to "no" and 1 to "yes" for later usage in kNN
levels(train$donation) <- c("no", "yes")
levels(test$donation) <- c("no", "yes")

# Caret library is used for data splitting. I choose a ratio of 75% / 25% to
# have enough train data
set.seed(1)
df <- train
partition <- createDataPartition(df[,1], times = 1, p = 0.75, list = FALSE)
train <- df[partition,] # Create the training sample
validation = df[-partition,] # Create the test sample
rm(df, partition) # remove temporary variables to clear cache
```

The train data set will be used for training and the validation data set for the comparison checks of the modeling results.

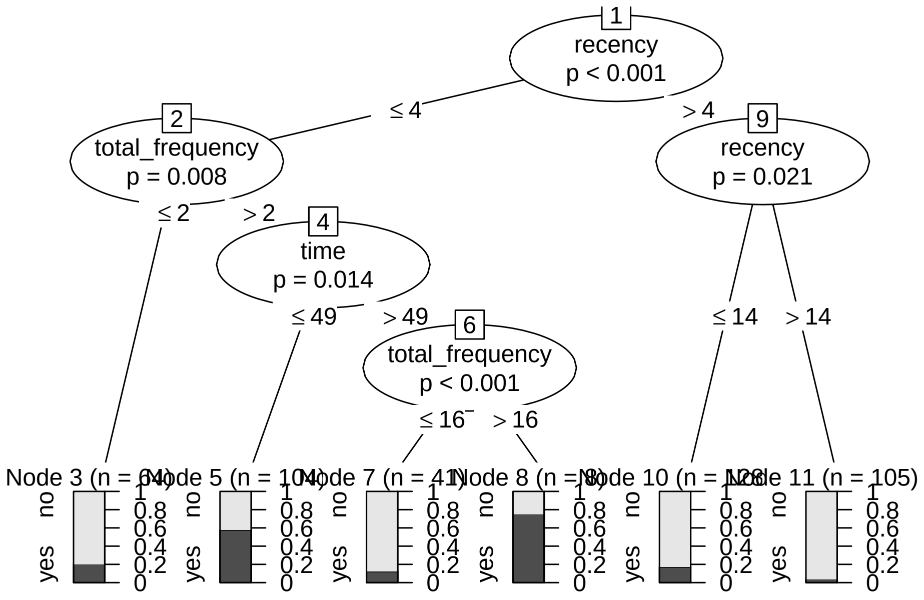
I will use crossvalidation in my training process, because this is a great possibility to tune the hyper parameter of a give algorithm. I will use 10 folds and 4 repeats, because the data set is not very big:

```
# 10 fold crossvalidation with 4 repeats are used for my models
crossvalidation <- trainControl(method = "repeatedcv",
                                number = 10,
                                repeats = 4,
                                classProbs = TRUE,
                                summaryFunction = mnLogLoss)
```

In the next sections I will start modelling with crossvalidation, but without any further optimization so far.

3.2 Modeling: Decision Trees Decision Trees give us the possibility of visualization. It's easy to understand. Here is the plot for our train data:

```
# Decision Tree plotting using tree
set.seed(1)
model.decisiontree = ctree(donation ~ ., data = train)
plot(model.decisiontree)
```



The resulting criteria in the end is first checked by recency. If this value is higher or lower than 6 further checks are done as shown in the plot.

From now on, I will use the caret package for all further modeling approaches:

```
# Decision tree using caret package
set.seed(1)
model.tree = train(donation ~ ., # Create Decision Tree with LogLoss based on Caret
                   data = train,
                  method = "rpart",
                  metric = "logLoss",
                  maximize = FALSE,
                  trControl = crossvalidation)
score = predict(model.tree, newdata = validation)
conf_matrix = confusionMatrix(score,validation$donation)
ll = min(model.tree$results$logLoss)

# create result table to collect all accuracy values from the different models
modeling_results <- tibble(method = "Decision Tree",
                           logLoss = ll,
                           Accuracy = conf_matrix$overall['Accuracy'])
```

Unfortunately Decision Trees are often only a quite weak model. If the data set changes a little bit, then the whole tree would look in another way.

```
set.seed(1)
model.rf <- train(donation ~ .,
                   data = train,
                   method = "rf",
                   metric = "logLoss",
                   maximize = FALSE,
                   trControl = crossvalidation)
score <- predict(model.rf, newdata = validation)
ll = min(model.rf$results$logLoss)
conf_matrix <- confusionMatrix(score,validation$donation)
modeling_results <- bind_rows(modeling_results, # add to result overview
                          tibble(method = "Random Forest",
                              logLoss = ll,
                              Accuracy = conf_matrix$overall['Accuracy']))
```

3.3 Modeling: Random Forest

3.4 Modeling: Logistic Regression

```
set.seed(1)
model.LogReg <- train(donation~.,
                     data=train,
                     method="glm",
                     family=binomial,
                     metric = "logLoss",
                     maximize = FALSE,
                     trControl = crossvalidation)
score <- predict(model.LogReg, newdata = validation)
ll = min(model.LogReg$results$logLoss)
conf_matrix <- confusionMatrix(score,validation$donation)
modeling_results <- bind_rows(modeling_results,  # add to result overview
                          tibble(method = "Logistic Regression",
                          logLoss = ll,
                          Accuracy = conf_matrix$overall['Accuracy']))
```

```


set.seed(1)
model.knn <- train(donation ~ .,
                   data = train,
                   method = "knn",
                   metric = "logLoss",
                   maximize = FALSE,
                   trControl = crossvalidation)
score <- predict(model.knn, newdata = validation)
ll = min(model.knn$results$logLoss)
conf_matrix <- confusionMatrix(score, validation$donation) # Check accuracy with validation set
modeling_results <- bind_rows(modeling_results, # add to result overview
                              tibble(method = "kNN",
                                     logLoss = ll,
                                     Accuracy = conf_matrix$overall['Accuracy']))
```

3.5 Modeling: kNN kNN is unfortunately computation-intensive for large data-sets. But my data set is not that big.

3.6 Modeling: Neuronal Network

When using nnet model it's important to scale the data before. That's why I will use "PreProc" for scaling parameter here:

```
set.seed(1)
model.nnet <- train(donation~.,
                    data=train,
                    method="nnet",
                    metric="logLoss",
                    maximize = FALSE,
                    preProc=c("center","scale"), # for nnet values should always be scaled
                    trControl = crossvalidation)
score <- predict(model.nnet, newdata = validation)
ll = min(model.nnet$results$logLoss)
conf_matrix <- confusionMatrix(score, validation$donation) # Check accuracy with validation set

modeling_results <- bind_rows(modeling_results, # add to result overview
                              tibble(method = "Neuronal Network",
                                     logLoss = ll,
                                     Accuracy = conf_matrix$overall['Accuracy']))
```

The training of this model can be computation-intensive. In addition this model can't be easy interpreted. Because it has many parameters so it can also easily lead to overfitting when using many of them.

3.7 Overview of different model results

Here you can see the result of the first modeling process:

```r
knitr::kable(modeling_results, caption = "modeling results")
```

Table 2: modeling results

method	logLoss	Accuracy
Decision Tree	0.4980457	0.7852349
Random Forest	0.8690659	0.7718121
Logistic Regression	0.4796696	0.7986577
kNN	0.8351866	0.7583893
Neuronal Network	0.4651303	0.8187919

The results don't look bad, but I will now try to improve them. First I will try to optimize the models by adding new features / columns.

3.8 Optimizing modeling results

3.8.1 Optimizing features To have more features and possibilities to use in my models, I will create new columns to have more insights:

I will add a column for the donation rate = total number of donations divided through number of months since fist donation. If this value is near 1, it means that the person is likely to come more often:

```r
# add a new column for donation rate into train data set
train <- mutate(train, don_rate = total_frequency / time)
```

I will add a column for faithfulness = number of months since last donation divided through total number of donations. If this number is small, it shows that this person did a lot of donations in the last time:

```r
# add a new column for faithfulness into train data set
train <- mutate(train, faith = recency / total_frequency)
```

I will add a column for churn_rate = number of months since last donation divided through number of months since firs donation. If this value is closed to 1, it shows that this person has perhaps stopped with blood donation:

```r
# add new column for people, who probably won't come back again
train <- mutate(train, churn_rate = recency / time)

# move donation column to the last column
train <- train %>% select(-donation,donation)

# check correlation of the new columns
cor(train[1:6])
```

```
##                    recency total_frequency       time    don_rate      faith
## recency          1.0000000      -0.1870654  0.1292167 -0.48660002  0.7195965
## total_frequency -0.1870654       1.0000000  0.6229414  0.26193906 -0.3936889
## time             0.1292167       0.6229414  1.0000000 -0.31150029 -0.2367116
## don_rate        -0.4866000       0.2619391 -0.3115003  1.00000000 -0.4028478
## faith            0.7195965      -0.3936889 -0.2367116 -0.40284781  1.0000000
## churn_rate       0.4281098      -0.5347593 -0.6345384  0.07073624  0.6600233
##                  churn_rate
## recency          0.42810978
## total_frequency -0.53475931
## time            -0.63453836
## don_rate         0.07073624
## faith            0.66002335
## churn_rate       1.00000000
```

I checked the correlations again to make sure, that I did not create highly correlated values. But that's fine and I can continue to add them also to my validation and test set:

```
# add all three new columns above also into our train and validation data set
test <- mutate(test, don_rate = total_frequency / time,
               faith = recency / total_frequency,
               churn_rate = recency / time)
validation <- mutate(validation, don_rate = total_frequency / time,
                     faith = recency / total_frequency,
                     churn_rate = recency / time)
```

I will now run my models again, based on the new added columns:

```
# Decision Tree
set.seed(1)
model.tree_opt_f = train(donation ~ ., # Create Decision Tree with LogLoss based on Caret
                   data = train,
                   method = "rpart",
                   metric = "logLoss",
                   maximize = FALSE,
                   trControl = crossvalidation)
score = predict(model.tree_opt_f, newdata = validation)
ll = min(model.tree_opt_f$results$logLoss)
conf_matrix = confusionMatrix(score,validation$donation)
modeling_results <- bind_rows(modeling_results, # add to result overview
                              tibble(method = "Decision Tree Caret optimized by new features",
                                     logLoss = ll,
                                     Accuracy = conf_matrix$overall['Accuracy']))

# Random Forest
set.seed(1)
model.rf_opt_f <- train(donation ~ .,
                  data = train,
                  method = "rf",
                  metric = "logLoss",
                  maximize = FALSE,
                  trControl = crossvalidation)
score <- predict(model.rf_opt_f, newdata = validation)
ll = min(model.rf_opt_f$results$logLoss)
conf_matrix <- confusionMatrix(score,validation$donation)
modeling_results <- bind_rows(modeling_results, # add to result overview
                              tibble(method = "Random Forest optimized by new features",
                                     logLoss = ll,
                                     Accuracy = conf_matrix$overall['Accuracy']))
```

```
# Logistic Regression
set.seed(1)
model.LogReg_opt_f <- train(donation~.,
                      data=train,
                      method="glm",
                      family=binomial,
                      metric = "logLoss",
                      maximize = FALSE,
                      trControl = crossvalidation)
score <- predict(model.LogReg_opt_f, newdata = validation)
ll = min(model.LogReg_opt_f$results$logLoss)
conf_matrix <- confusionMatrix(score,validation$donation)
modeling_results <- bind_rows(modeling_results,  # add to result overview
                              tibble(method = "Logistic Regression optimized by new features",
                                     logLoss = ll,
                                     Accuracy = conf_matrix$overall['Accuracy']))

# Modeling: kNN
set.seed(1)
model.knn_opt_f <- train(donation ~ .,
                   data = train,
                   method = "knn",
                   metric = "logLoss",
                   maximize = FALSE,
                   trControl = crossvalidation)
score <- predict(model.knn_opt_f, newdata = validation)
ll = min(model.knn_opt_f$results$logLoss)
conf_matrix <- confusionMatrix(score, validation$donation) # Check accuracy with validation set
modeling_results <- bind_rows(modeling_results, # add to result overview
                              tibble(method = "kNN optimized by new features",
                                     logLoss = ll,
                                     Accuracy = conf_matrix$overall['Accuracy']))

# Modeling: Neuronal Network
set.seed(1)
model.nnet_opt_f <- train(donation~.,
                    data=train,
                    method="nnet",
                    metric="logLoss",
                    maximize = FALSE,
                    preProc=c("center","scale"), # for nnet values should always be scaled
                    trControl = crossvalidation)
score <- predict(model.nnet_opt_f, newdata = validation)
ll = min(model.nnet_opt_f$results$logLoss)
conf_matrix <- confusionMatrix(score, validation$donation) # Check accuracy with validation set

modeling_results <- bind_rows(modeling_results, # add to result overview
                              tibble(method = "Neuronal Network optimized by new features",
                                     logLoss = ll,
                                     Accuracy = conf_matrix$overall['Accuracy']))
```

This is my result after feature optimization:

```r
knitr::kable(modeling_results, caption = "modeling results")
```

Table 3: modeling results

method	logLoss	Accuracy
Decision Tree	0.4980457	0.7852349
Random Forest	0.8690659	0.7718121
Logistic Regression	0.4796696	0.7986577
kNN	0.8351866	0.7583893
Neuronal Network	0.4651303	0.8187919
Decision Tree Caret optimized by new features	0.5746059	0.7785235
Random Forest optimized by new features	0.7873720	0.7919463
Logistic Regression optimized by new features	0.4643207	0.7919463
kNN optimized by new features	0.8339863	0.7718121
Neuronal Network optimized by new features	0.4671644	0.7852349

3.8.2 Optimizing with TuneGrid Next I will prepare tuneGrid. TuneGrid is used to check different possible parameter combination during the modelling process. I will use different TuneGrids adapted to the corresponding models:

```r
# Decision Tree
set.seed(1)
tunegrid_dc <- expand.grid(cp=.2) # I will use 1 to 30 here to make it not too big
model.tree_opt_tg = train(donation ~ ., # Create Decision Tree with LogLoss based on Caret
                         data = train,
                         method = "rpart",
                         metric = "logLoss",
                         maximize = FALSE,
                         trControl = crossvalidation,
                         tuneGrid = tunegrid_dc)
score = predict(model.tree_opt_tg, newdata = validation)
ll = min(model.tree_opt_tg$results$logLoss)
conf_matrix = confusionMatrix(score,validation$donation)
modeling_results <- bind_rows(modeling_results, # add to result overview
                              tibble(method = "Decision Tree Caret optimized by TuneGrid",
                                     logLoss = ll,
                                     Accuracy = conf_matrix$overall['Accuracy']))

# Random Forest
tunegrid_rf <- expand.grid(mtry=2) # 1 would be less and >5 more complex, I choose middle
set.seed(1)
model.rf_opt_tg <- train(donation ~ .,
                        data = train,
                        method = "rf",
                        metric = "logLoss",
                        maximize = FALSE,
                        trControl = crossvalidation,
                        tuneGrid = tunegrid_rf)
score <- predict(model.rf_opt_tg, newdata = validation)
ll = min(model.rf_opt_tg$results$logLoss)
conf_matrix <- confusionMatrix(score,validation$donation)
```

```
modeling_results <- bind_rows(modeling_results, # add to result overview
                              tibble(method = "Random Forest optimized by TuneGrid",
                                     logLoss = ll,
                                     Accuracy = conf_matrix$overall['Accuracy']))

# Logistic Regression
set.seed(1)
tunegrid_logreg <- expand.grid(parameter=2)
model.LogReg_opt_tg <- train(donation~.,
                            data=train,
                            method="glm",
                            family=binomial,
                            metric = "logLoss",
                            maximize = FALSE,
                            trControl = crossvalidation,
                            tuneGrid = tunegrid_logreg)
score <- predict(model.LogReg_opt_tg, newdata = validation)
ll = min(model.LogReg_opt_tg$results$logLoss)
conf_matrix <- confusionMatrix(score,validation$donation)
modeling_results <- bind_rows(modeling_results,  # add to result overview
                              tibble(method = "Logistic Regression optimized by TuneGrid",
                                     logLoss = ll,
                                     Accuracy = conf_matrix$overall['Accuracy']))

# Modeling: kNN
set.seed(1)
tunegrid_knn <- expand.grid(k=c(1:30))
model.knn_opt_tg <- train(donation ~ .,
                         data = train,
                         method = "knn",
                         metric = "logLoss",
                         maximize = FALSE,
                         trControl = crossvalidation,
                         tuneGrid = tunegrid_knn)
score <- predict(model.knn_opt_tg, newdata = validation)
ll = min(model.knn_opt_tg$results$logLoss)
conf_matrix <- confusionMatrix(score, validation$donation) # Check accuracy with validation set
modeling_results <- bind_rows(modeling_results, # add to result overview
                              tibble(method = "kNN optimized by TuneGrid",
                                     logLoss = ll,
                                     Accuracy = conf_matrix$overall['Accuracy']))

# Modeling: Neuronal Network
set.seed(1)
tunegrid_nnet = expand.grid(size=c(3:5), decay=c(0.1,0.5))
model.nnet_opt_tg <- train(donation~.,
                          data=train,
                          method="nnet",
                          metric="logLoss",
                          maximize = FALSE,
                          preProc=c("center","scale"), # for nnet values should always be scaled
                          trControl = crossvalidation,
                          tuneGrid = tunegrid_nnet)
```

```
score <- predict(model.nnet_opt_tg, newdata = validation)
ll = min(model.nnet_opt_tg$results$logLoss)
conf_matrix <- confusionMatrix(score, validation$donation) # Check accuracy with validation set
modeling_results <- bind_rows(modeling_results, # add to result overview
                              tibble(method = "Neuronal Network optimized by TuneGrid",
                                     logLoss = ll,
                                     Accuracy = conf_matrix$overall['Accuracy']))
```

This is my result after feature and TuneGrid optimization:

```
knitr::kable(modeling_results, caption = "modeling results")
```

Table 4: modeling results

method	logLoss	Accuracy
Decision Tree	0.4980457	0.7852349
Random Forest	0.8690659	0.7718121
Logistic Regression	0.4796696	0.7986577
kNN	0.8351866	0.7583893
Neuronal Network	0.4651303	0.8187919
Decision Tree Caret optimized by new features	0.5746059	0.7785235
Random Forest optimized by new features	0.7873720	0.7919463
Logistic Regression optimized by new features	0.4643207	0.7919463
kNN optimized by new features	0.8339863	0.7718121
Neuronal Network optimized by new features	0.4671644	0.7852349
Decision Tree Caret optimized by TuneGrid	0.5561490	0.7785235
Random Forest optimized by TuneGrid	0.8151919	0.7718121
Logistic Regression optimized by TuneGrid	0.4643207	0.7919463
kNN optimized by TuneGrid	0.4950776	0.7852349
Neuronal Network optimized by TuneGrid	0.4646511	0.8120805

3.8.4 Optimizing using scaling

The summary of our new train table will look like this:

```
summary(train)
```

```
##     recency       total_frequency       time          don_rate
##  Min.   : 0.000   Min.   : 1.000   Min.   : 2.00   Min.   :0.02500
##  1st Qu.: 3.000   1st Qu.: 2.000   1st Qu.:16.00   1st Qu.:0.08696
##  Median : 7.000   Median : 4.000   Median :29.00   Median :0.15385
##  Mean   : 9.311   Mean   : 5.793   Mean   :35.95   Mean   :0.19404
##  3rd Qu.:14.000   3rd Qu.: 7.000   3rd Qu.:52.00   3rd Qu.:0.25000
##  Max.   :40.000   Max.   :50.000   Max.   :98.00   Max.   :1.00000
##      faith          churn_rate      donation
##  Min.   : 0.0000   Min.   :0.0000   no :340
##  1st Qu.: 0.5714   1st Qu.:0.1176   yes:110
##  Median : 2.0000   Median :0.2602
##  Mean   : 4.1224   Mean   :0.4195
##  3rd Qu.: 4.1917   3rd Qu.:0.8113
##  Max.   :40.0000   Max.   :1.0000
```

We can see in this summary output, that the numerical features have different units (months, time) and different scales. So we need to normalize the data. Otherwise the algorithm will be dominated by the feature with the larger scale and so affect the model performance. I will use the parameter preProcess in the train function to correct this.

For NNET this parameter has already been set. I will now add it to the other 4 models too:

```
# Decision Tree
set.seed(1)
tunegrid_dc <- expand.grid(cp=.2) # I will use 1 to 30 here to make it not too big
model.tree_opt_tgs = train(donation ~ ., # Create Decision Tree with LogLoss based on Caret
                          data = train,
                          method = "rpart",
                          metric = "logLoss",
                          maximize = FALSE,
                          trControl = crossvalidation,
                          tuneGrid = tunegrid_dc,
                          preProc = c("center","scale"))
score = predict(model.tree_opt_tgs, newdata = validation)
ll = min(model.tree_opt_tgs$results$logLoss)
conf_matrix = confusionMatrix(score,validation$donation)
modeling_results <- bind_rows(modeling_results, # add to result overview
                              tibble(method = "Decision Tree Caret optimized by TuneGrid and scaling",
                                     logLoss = ll,
                                     Accuracy = conf_matrix$overall['Accuracy']))

# Random Forest
tunegrid_rf <- expand.grid(mtry=2) # 1 would be less and >5 more complex, I choose middle
set.seed(1)
model.rf_opt_tgs <- train(donation ~ .,
                         data = train,
                         method = "rf",
                         metric = "logLoss",
                         maximize = FALSE,
                         trControl = crossvalidation,
                         tuneGrid = tunegrid_rf,
                         preProc = c("center","scale"))
score <- predict(model.rf_opt_tgs, newdata = validation)
ll = min(model.rf_opt_tgs$results$logLoss)
conf_matrix <- confusionMatrix(score,validation$donation)
modeling_results <- bind_rows(modeling_results, # add to result overview
                              tibble(method = "Random Forest optimized by TuneGrid and scaling",
                                     logLoss = ll,
                                     Accuracy = conf_matrix$overall['Accuracy']))

# Logistic Regression
set.seed(1)
tunegrid_logreg <- expand.grid(parameter=2)
model.LogReg_opt_tgs <- train(donation~.,
                             data=train,
                             method="glm",
                             family=binomial,
                             metric = "logLoss",
                             maximize = FALSE,
                             trControl = crossvalidation,
```

```r
                              tuneGrid = tunegrid_logreg,
                              preProc = c("center","scale"))
score <- predict(model.LogReg_opt_tgs, newdata = validation)
ll = min(model.LogReg_opt_tgs$results$logLoss)
conf_matrix <- confusionMatrix(score,validation$donation)
modeling_results <- bind_rows(modeling_results,  # add to result overview
                               tibble(method = "Logistic Regression optimized by TuneGrid and scaling",
                                      logLoss = ll,
                                      Accuracy = conf_matrix$overall['Accuracy']))

# Modeling: kNN
set.seed(1)
tunegrid_knn <- expand.grid(k=c(1:30))
model.knn_opt_tgs <- train(donation ~ .,
                           data = train,
                           method = "knn",
                           metric = "logLoss",
                           maximize = FALSE,
                           trControl = crossvalidation,
                           tuneGrid = tunegrid_knn,
                           preProc = c("center","scale"))
score <- predict(model.knn_opt_tgs, newdata = validation)
ll = min(model.knn_opt_tgs$results$logLoss)
conf_matrix <- confusionMatrix(score, validation$donation) # Check accuracy with validation set
modeling_results <- bind_rows(modeling_results, # add to result overview
                               tibble(method = "kNN optimized by TuneGrid and scaling",
                                      logLoss = ll,
                                      Accuracy = conf_matrix$overall['Accuracy']))
```

4. Results

4.1 Presentation of modeling results

This is my final modeling result:

```r
# Show results based on logLoss value (desc sorting)
modeling_results %>% arrange(modeling_results$logLoss)
```

```
## # A tibble: 19 x 3
##    method                                                logLoss Accuracy
##    <chr>                                                   <dbl>    <dbl>
##  1 Logistic Regression optimized by new features           0.464    0.792
##  2 Logistic Regression optimized by TuneGrid               0.464    0.792
##  3 Logistic Regression optimized by TuneGrid and scaling   0.464    0.792
##  4 Neuronal Network optimized by TuneGrid                  0.465    0.812
##  5 Neuronal Network                                        0.465    0.819
##  6 Neuronal Network optimized by new features              0.467    0.785
##  7 Logistic Regression                                     0.480    0.799
##  8 kNN optimized by TuneGrid                               0.495    0.785
##  9 Decision Tree                                           0.498    0.785
## 10 kNN optimized by TuneGrid and scaling                   0.546    0.805
## 11 Decision Tree Caret optimized by TuneGrid               0.556    0.779
## 12 Decision Tree Caret optimized by TuneGrid and scaling   0.556    0.779
## 13 Decision Tree Caret optimized by new features           0.575    0.779
## 14 Random Forest optimized by new features                 0.787    0.792
## 15 Random Forest optimized by TuneGrid and scaling         0.795    0.779
## 16 Random Forest optimized by TuneGrid                     0.815    0.772
## 17 kNN optimized by new features                           0.834    0.772
## 18 kNN                                                     0.835    0.758
## 19 Random Forest                                           0.869    0.772
```

```
# Show results based on Accuracy value (asc sorting)
modeling_results %>% arrange(desc(modeling_results$Accuracy))
```

```
## # A tibble: 19 x 3
##    method                                                logLoss Accuracy
##    <chr>                                                   <dbl>    <dbl>
##  1 Neuronal Network                                        0.465    0.819
##  2 Neuronal Network optimized by TuneGrid                  0.465    0.812
##  3 kNN optimized by TuneGrid and scaling                   0.546    0.805
##  4 Logistic Regression                                     0.480    0.799
##  5 Random Forest optimized by new features                 0.787    0.792
##  6 Logistic Regression optimized by new features           0.464    0.792
##  7 Logistic Regression optimized by TuneGrid               0.464    0.792
##  8 Logistic Regression optimized by TuneGrid and scaling   0.464    0.792
##  9 Decision Tree                                           0.498    0.785
## 10 Neuronal Network optimized by new features              0.467    0.785
## 11 kNN optimized by TuneGrid                               0.495    0.785
## 12 Decision Tree Caret optimized by new features           0.575    0.779
## 13 Decision Tree Caret optimized by TuneGrid               0.556    0.779
## 14 Decision Tree Caret optimized by TuneGrid and scaling   0.556    0.779
## 15 Random Forest optimized by TuneGrid and scaling         0.795    0.779
## 16 Random Forest                                           0.869    0.772
## 17 kNN optimized by new features                           0.834    0.772
## 18 Random Forest optimized by TuneGrid                     0.815    0.772
## 19 kNN                                                     0.835    0.758
```

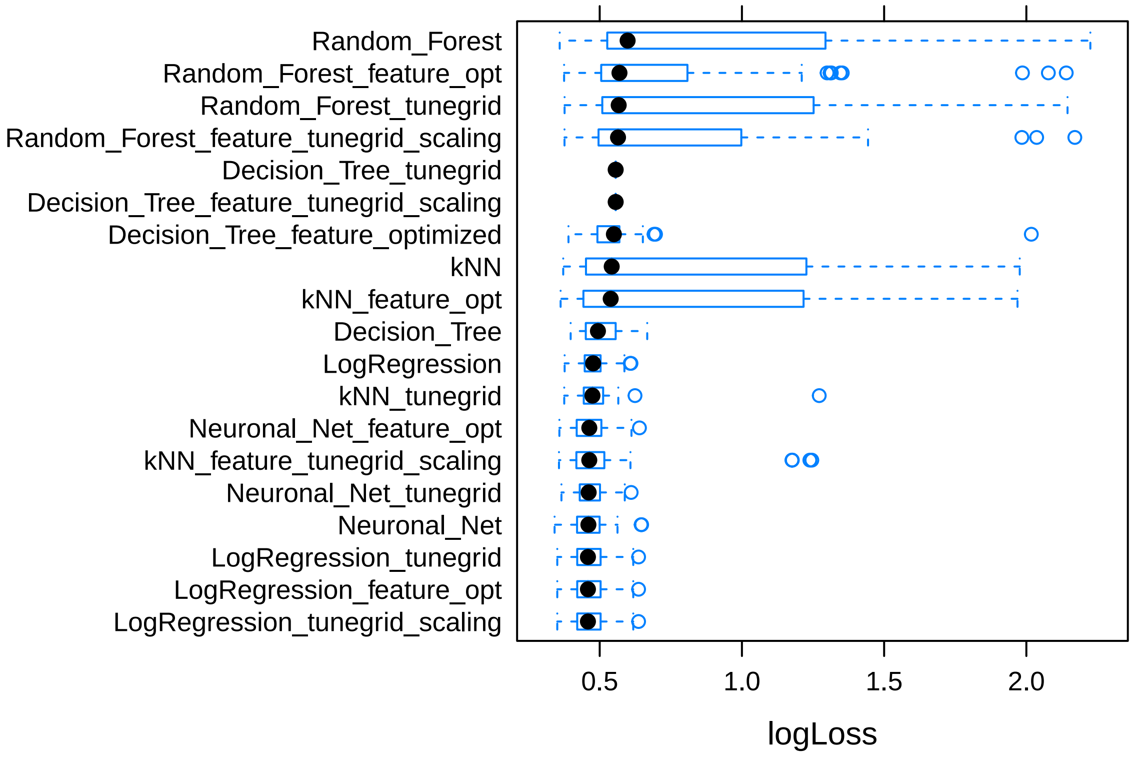
Looking at the models above, we see that we received the lowest logLoss for the Neuronal Network model with feature and TuneGrid optimization. And the model with the highest accuracy it the Neuronal Network model without optimization.

Let's create an overview for logLoss:

```
# collect the results using resamples function into one plot
results <- resamples(list(Decision_Tree = model.tree,
                          Random_Forest = model.rf,
                          kNN = model.knn,
                          LogRegression = model.LogReg,
                          Neuronal_Net = model.nnet,
                          Decision_Tree_feature_optimized = model.tree_opt_f,
                          Random_Forest_feature_opt = model.rf_opt_f,
                          kNN_feature_opt = model.knn_opt_f,
                          LogRegression_feature_opt = model.LogReg_opt_f,
                          Neuronal_Net_feature_opt = model.nnet_opt_f,
                          Decision_Tree_tunegrid = model.tree_opt_tg,
                          Random_Forest_tunegrid = model.rf_opt_tg,
                          kNN_tunegrid = model.knn_opt_tg,
                          LogRegression_tunegrid = model.LogReg_opt_tg,
                          Neuronal_Net_tunegrid = model.nnet_opt_tg,
                          Decision_Tree_feature_tunegrid_scaling = model.tree_opt_tgs,
                          Random_Forest_feature_tunegrid_scaling = model.rf_opt_tgs,
                          kNN_feature_tunegrid_scaling = model.knn_opt_tgs,
                          LogRegression_tunegrid_scaling = model.LogReg_opt_tgs))
# show summary of the results
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: Decision_Tree, Random_Forest, kNN, LogRegression, Neuronal_Net, Decision_Tree_feature_optimi
## Number of resamples: 40
##
## logLoss
##                                             Min.   1st Qu.    Median      Mean
## Decision_Tree                          0.3979040 0.4524525 0.4938329 0.4980457
## Random_Forest                          0.3592861 0.5271804 0.5981336 0.8690659
## kNN                                    0.3716777 0.4522784 0.5421625 0.8351866
## LogRegression                          0.3766535 0.4478851 0.4769168 0.4796696
## Neuronal_Net                           0.3413413 0.4209069 0.4602333 0.4651303
## Decision_Tree_feature_optimized        0.3901399 0.4934221 0.5498445 0.5746059
## Random_Forest_feature_opt              0.3747671 0.5098812 0.5694369 0.7873720
## kNN_feature_opt                        0.3625088 0.4433627 0.5385961 0.8339863
## LogRegression_feature_opt              0.3508246 0.4216668 0.4585836 0.4643207
## Neuronal_Net_feature_opt               0.3582132 0.4194632 0.4634099 0.4671644
## Decision_Tree_tunegrid                 0.5561490 0.5561490 0.5561490 0.5561490
## Random_Forest_tunegrid                 0.3765325 0.5104208 0.5666946 0.8151919
## kNN_tunegrid                           0.3752584 0.4447231 0.4744539 0.4950776
## LogRegression_tunegrid                 0.3508246 0.4216668 0.4585836 0.4643207
## Neuronal_Net_tunegrid                  0.3654355 0.4299613 0.4611221 0.4646511
## Decision_Tree_feature_tunegrid_scaling 0.5561490 0.5561490 0.5561490 0.5561490
## Random_Forest_feature_tunegrid_scaling 0.3761951 0.5000387 0.5643899 0.7949197
## kNN_feature_tunegrid_scaling           0.3568404 0.4185346 0.4633202 0.5457698
## LogRegression_tunegrid_scaling         0.3508246 0.4216668 0.4585836 0.4643207
##                                          3rd Qu.      Max. NA's
## Decision_Tree                          0.5560827 0.6670430    0
## Random_Forest                          1.2925610 2.2250714    0
## kNN                                    1.2226603 1.9767670    0
## LogRegression                          0.5028944 0.6100001    0
## Neuronal_Net                           0.4989025 0.6475075    0
## Decision_Tree_feature_optimized        0.5685021 2.0174993    0
## Random_Forest_feature_opt              0.7965408 2.1402886    0
## kNN_feature_opt                        1.2114626 1.9688802    0
## LogRegression_feature_opt              0.5020981 0.6367281    0
## Neuronal_Net_feature_opt               0.5047195 0.6397931    0
## Decision_Tree_tunegrid                 0.5561490 0.5561490    0
## Random_Forest_tunegrid                 1.2311876 2.1447747    0
## kNN_tunegrid                           0.5108835 1.2718772    0
## LogRegression_tunegrid                   0.5020981 0.6367281    0
## Neuronal_Net_tunegrid                    0.4998374 0.6106593    0
## Decision_Tree_feature_tunegrid_scaling   0.5561490 0.5561490    0
## Random_Forest_feature_tunegrid_scaling   0.8936707 2.1705068    0
## kNN_feature_tunegrid_scaling             0.5136812 1.2457538    0
## LogRegression_tunegrid_scaling           0.5020981 0.6367281    0
```

```
# create a dot plot and a bwplot to visually see the results
bwplot(results)
```



We see that the ranges are quite near together but there are some models with a higher variance and some outliers to the right side.

4.2 Discussion of the model performance

According to the "No-free-Lunch" theoreme there are several ways to improve an algorithm. Every algorithm has it advantages and disadvantages and many different hyper parameters for tuning. I've used the Resamples function to compare the different model performances regarding logLoss. Using metric logLoss we are interested in minimizing the value here, because it measures uncertainty and so a low logLoss means a low uncertainty of the model. Compared to Accuracy the main focus is on classes in the output like we can see it in the data here. The disadvantage of logLoss is, that it's not easy to interpret and we need other values to check, if a performance is better or not.

Another good performance indicator is the Confusion Matrix. If we check the Confusion Matrix for example for our model with the lowest logLoss, we see that the Accuracy is also quite good. Accuracy means: How often the classifier correct. Important for us in this classification model is the TN rate (true negative), which can be seen in the Confusion Matrix:

```
# show Confusion Matrix for our Neuronal Network with feature and TuneGrid optimization
confusionMatrix(predict(model.nnet_opt_tg, newdata = validation), validation$donation)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  110  22
##        yes   6  11
##
##                Accuracy : 0.8121
##                  95% CI : (0.74, 0.8713)
##     No Information Rate : 0.7785
##     P-Value [Acc > NIR] : 0.188335
##
##                   Kappa : 0.3407
##
##  Mcnemar's Test P-Value : 0.004586
##
##             Sensitivity : 0.9483
##             Specificity : 0.3333
##          Pos Pred Value : 0.8333
##          Neg Pred Value : 0.6471
##              Prevalence : 0.7785
##          Detection Rate : 0.7383
##    Detection Prevalence : 0.8859
##       Balanced Accuracy : 0.6408
##
##        'Positive' Class : no
##
```

It's 85 in this case using this model above. Because we need to know, who would not come to the next blood donation to find possibilities to tout them. This can be tranferred to marketing advertising and customer behaviour analysis; There the companies are also interested who would not come back to buy things during the next time to actively create for example personalized discounts for them.

4.3 Use final model on final test data set

As a final test, we can now use my so far untouched test data set to check the model:

```
score <- predict(model.nnet_opt_tg, newdata = test)
confusionMatrix(score, test$donation) # Check accuracy with validation set
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  110  25
##        yes   4  10
```

```
##
##                Accuracy : 0.8054
##                  95% CI : (0.7326, 0.8656)
##     No Information Rate : 0.7651
##     P-Value [Acc > NIR] : 0.1432810
##
##                   Kappa : 0.3164
##
##  Mcnemar's Test P-Value : 0.0002041
##
##             Sensitivity : 0.9649
##             Specificity : 0.2857
##          Pos Pred Value : 0.8148
##          Neg Pred Value : 0.7143
##              Prevalence : 0.7651
##          Detection Rate : 0.7383
##    Detection Prevalence : 0.9060
##       Balanced Accuracy : 0.6253
##
##        'Positive' Class : no
##
```

The accuracy here is 0.7986 which is not as good as in den validation data set, but also a good value, because nearly 80% of the prediction is correct.

5. Conclusion

5.1 Brief summary of the report

We can use former data to analyse future behavior. But as behaviour is human, we can never be 100% sure.

I know that there's a minimum time range between two donations where a person is not allowed to donate. So it's not possible that a person come in a very short time frame again. In addition, different impairments to health can have an influence, that a person is not allowed to go (for example if sb. is sick). Further effects of preventing sb. to continue going to donations are heart issues, because people who have problems with the heart (for example a former heart attack) are not allowed anymore to do blood donations.

5.2 Potential impact of the report

This analysis is a typical example of a marketing analysis. People with 0 could be "bothered" with advertising / discounts until they come back.

5.3 Limitations of the report

I chose a data set, which is not that big to make sure that all analysis are working on my old laptop. You always need to make sure to have "real" and valid data, because a report is only as good as the data quality. So if you start with a bad data set or data quality, it's harder to create a good model. That's why it's important to have as much data as possible. So you have more possibilities in your modeling process.

5.4 Future work

In regards to the grading rubric it was only necessary to use at least two different models or algorithms with at least one being more advanced than linear or logistic regression. I used 5 for my report here, but I was also limited a little bit regarding computer performance. With the possibility of more RAM on the laptop and more time to do further analysis it would be interesting to use further models or column combinations to improve the results.

It would also be interesting to compare based on other performance measurements, for example F1 score which is the harmonic mean of precision and recall. Also AUC would be an interesting possibility to summary the performances of the models. Based on that it would be possible to use ROC curve to visualize the performace of binary classifiers.