Edx Capstone - Choose your own
Treatment Response Prediction in Behavioral Health

Hans Jacob Westbye

09 februar, 2022

Contents

Chapter 1

Introduction

Mental health problems are an enormous personal burden and a threat to global health systems. The increase in demand for mental health services far outweighs the growth in supply, and with traditional approaches and treatment processes this gap will never be closed. New approaches to mental health treatment and health system structures are urgently needed. One such novel approach, that already is implemented, is the Norse Feedback system, developed in Norway by researchers, treatment providers and patients in a collaborative process. In the US this feedback system is distributed and implemented by Mirah, Inc.

The Norwegian Outcome Response System for Evaluation (NORSE) is a second generation Routine Outcome Monitoring innovation, with a dynamic clinical feedback system, developed at Førde Hospital Trust. NORSE has advanced significantly on existing methodologies in that the system A) actively learns from, and B) adapts to, the individual patient´s feedback, and C) instantaneously personalizes the system to individual profiles of suffering and resources. A consequence of the computer adaptive nature of the feedback system is that patients are only asked to report on items that are relevant to them, this leads to differing length of responses in the dataset and missing values for many items. Implications of missing data and a strategy for handling this is discussed in Chapter 2: Handling Missing Data

For this project a dataset containing patient reported data from the US for a 5 year period has been made available. The data is collected using the first version of the Norse Feedback system, and several subsequent versions have rendered many items obsolete. The dataset used in this research only contains the items that still are relevant.

This report serves a two-fold purpose, first as the final submission in the Edx Data Science course, and second as an exploration of the data to provide new knowledge and foundation for further research on treatment response in mental health. The dataset has been anonymized and during this process every patient was given a unique ID for each year. This entails that the unique patients in the dataset could be in the middle of a treatment series spanning several years, and that sessions from different years will be stored with different patient IDs. Menaing for this analysis that the three sessions I use to predict outcome not necessarily are the three first sessions of a treatment series. In clinical practice this is not an unknown situation as patients can change therapists during treatment, or the feedback system can be introduced in the middle of a treatment series.

1.1 The Problem

The problem I will explore is defined by the research group developing the Norse Feedback system. Treatment providers often face unique challenges with individual patients, and as a group effect of therapy is only about 50%. It would be valuable for therapists to get an indication about the effect of therapy for an individual patient as early as possible, and even valuable more to get feedback on which features mostly impacted the prediction of lack of effect of therapy. Given such information, individualization of therapy could be possible, e.g. through conversations with the patient about what the treatment response predictions mean, and how therapist and patient could collaborate to improve treatment outcomes. Two main research questions have been identified:

- Can treatment response for an individual patient be predicted after three session?
- Which features most often predict poor treatment response?

Prediction of human behavior is notoriously difficult and expectations for the results from this project are limited. Any improvement in prediction accuracy beyond random prediction based on population averages would add value for treatment providers.

1.2 Clinical Usability Requirements

An important feature for a machine learning model for a clinical problem is interpretability. Therapists must be able to understand the logic behind predictions and the features driving lack of treatment response. My approach to exploring the research questions given the problem description is detailed in chapter 2: Methods and Analysis.

The raw dataset is not labeled. The researchers and therapists from Norse Feedback have advised using improvement on a single item from the dataset as outcome label. This is a crucial part of this analysis and the details concerning thresholds and outcome labeling is detailed in Chapter 3: Data Preprocessing.

1.3 Machine Learning Algorithms

Given the research questions and the clinical usability requirements I will proceed using only supervised shallow machine learning (ML) algorithms. Deliberations around choice of ML algorithms are detailed in Chapter 2: Methods and analysis, and results from the ML models are reported in Chapter 4: Results.

1.4 Executive Summary

This report summarizes the exploration of a dataset with patient reported data on 80 items in the Norse Feedback system for mental health. After data pre-processing, including labeling of the unlabeled dataset, 3490 patients and 29 items were included for further analysis and model training. With the addition of age and sex the total number of features is 31. Supervised shallow ML algorithms were chosen for model training given the clinical requirements included in the problem description. A major challenge with the dataset was the sparsity and variance in patient reported data, leading to a large number of NAs in the dataset. A basic model utilizing logistic regression with imputation of 0s for missing data was trained. However, this model have limited usability without further adaptation, and do not answer well on the problem description. Gradient boosting classification tree algorithms are popular, efficient and accurate for classification problems. The `XGBoost` library implements gradient boosting tree with some improvements versus former implementations of this algorithm and can also handle missing data. After splitting the dataset and some feature engineering the final model was trained on 2653 patients and 91 features and validated in 664 patients. The `XGBoost` model predicted lack of treatment effect with an accuracy of 0.7139 in the validation set, giving a ROC AUC of 0.7690.

Chapter 2

Methods and Analysis

In this chapter I will go through the process of initial preparation and exploration of the dataset. The methods and algorithms I intend to utilize are briefly described, as well as some key challenges with this dataset that I will need to handle during data pre-processing.

2.1 Data preparation

Load libraries needed for initial data preparation

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(ggthemes)) install.packages("ggthemes", repos = "http://cran.us.r-project.org")
if(!require(scales)) install.packages("scales", repos = "http://cran.us.r-project.org")
if(!require(pander)) install.packages("pander", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

I start with downloading and unzipping the dataset

```
# Download and unzip the raw data from github
dl <- tempfile()
download.file("https://github.com/hjwestbye/capstone2/raw/main/data_raw.zip", dl)


nfdata_raw <- fread(unzip(dl, "data_raw.csv"),
                    col.names = c("year", "patId", "age", "sex",
                                  "session", "itemName", "rating",
                                  "question", "n_sessions"))
```

A first look at the dataset to inspect dimensions and structure

```
# Inspect the raw data
dim(nfdata_raw)
```

```
## [1] 739531      9
```

```
str(nfdata_raw)
```

```
## Classes 'data.table' and 'data.frame':   739531 obs. of  9 variables:
##  $ year      : int  2016 2016 2016 2016 2016 2016 2016 2016 2016 2016 ...
##  $ patId     : chr  "9799f058d5db166c1b18129067dca299" "9799f058d5db166c1b18129067dca299" "9799f058d5db
##  $ age       : int  33 33 33 33 33 33 33 33 33 33 ...
##  $ sex       : chr  "F" "F" "F" "F" ...
##  $ session   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ itemName  : chr  "fearLosingFoodControl" "therapistTechniqueExerciseQuantity" "therapistAcceptsAsPer
##  $ rating    : chr  "2" "" "7" "1" ...
##  $ question  : chr  "I am afraid I will lose control when it comes to food. <span class=\"\"\"\"likertH
##  $ n_sessions: int  9 9 9 9 9 9 9 9 9 9 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

The question column seems to include html headers that I will need to clean up.

Inspecting the first 10 rows of the dataset without the messy questions column.

```r
# Inspect the 10 first rows (excluding question/text column)
pander(nfdata_raw[1:10,-8])
```

Table 2.1: Table continues below

year	patId	age	sex	session
2016	9799f058d5db166c1b18129067dca299	33	F	1
2016	9799f058d5db166c1b18129067dca299	33	F	1
2016	9799f058d5db166c1b18129067dca299	33	F	1
2016	9799f058d5db166c1b18129067dca299	33	F	1
2016	9799f058d5db166c1b18129067dca299	33	F	1
2016	9799f058d5db166c1b18129067dca299	33	F	1
2016	9799f058d5db166c1b18129067dca299	33	F	1
2016	9799f058d5db166c1b18129067dca299	33	F	1
2016	9799f058d5db166c1b18129067dca299	33	F	1
2016	9799f058d5db166c1b18129067dca299	33	F	1

itemName	rating	n_sessions
fearLosingFoodControl	2	9
therapistTechniqueExerciseQuantity		9
therapistAcceptsAsPerson	7	9
workToAvoidPainfulThoughts	1	9
tryToAvoidSomeEmotions	1	9
trappedAndUncertain	4	9
concernedDrugDependent	1	9
eatingRestrictsSocial	2	9
breathRacingOrNumbness	5	9
drugUseInterferes	1	9

Exploring total number of patients and items in the dataset

```r
# Unique patients in dataset
length(unique((nfdata_raw$patId)))
```

```
## [1] 3490
```

```
# Number of features
length(unique((nfdata_raw$itemName)))
```

```
## [1] 29
```

I want to tidy up the dataset and create a dataframe to store the item information.

```
item_df <- nfdata_raw %>% distinct(itemName, .keep_all = TRUE) %>% select(itemName, question)
```

To clean up the question column I need to take a closer look.

```
pander(item_df[1,])
```

itemName	question
fearLosingFoodControl	I am afraid I will lose control when it comes to food. <span class=“ “ ““likertHeader”“ ““><span class=”“ ““likertLeft”“ ““>Not at all<span class=”“ ““likertRight”“ “ “>True for me

The question column contains html headers and information I do not need. The central information is the question text which is located at the start of the string and ends with a punctuation mark. I extract the question string using `string_extract_all` and a regex. The questions are added to the item dataframe, items are given a unique item ID, and the itemIds are formated as factors.

```
question_list <- item_df$question %>% str_extract_all("^(.+)\\.")

# Convert questions from list to dataframe
question_df <- data.frame(t(data.frame(t(sapply(question_list,c)))))

# Update item dataframe with question texts
item_df <- item_df %>% mutate(question = question_df[,1]) %>%
  mutate(itemId = row_number()) %>%
  select(itemId, itemName, question)

# Convert itemId to factor
item_df$itemId <- as.factor(item_df$itemId)

head(item_df)
```

itemId	itemName	question
1	fearLosingFoodControl	I am afraid I will lose control when it comes to food.
2	therapistTechniqueExerciseQuantity	
3	therapistAcceptsAsPerson	
4	workToAvoidPainfulThoughts	I spend a lot of energy trying not to think about things that hurt.
5	tryToAvoidSomeEmotions	I try very hard not to feel certain emotions.
6	trappedAndUncertain	I feel like I am trapped and don’t know what to do.

For some items the questions are missing. I have received these missing questions from the Norse Feedback research team and add them manually.

```r
item_df[2,3] <- "I would like my therapist to use less/more techniques and exersises"
item_df[3,3] <- "I feel that my therapist accepts me as a person"
item_df[18,3] <- "I feel that the therapist understands me and understands why I am in treatment now"
item_df[19,3] <- "I would like my therapist to focus less/more on the relationship between us"
item_df[21,3] <- "I would like my therapist to show their personality more/be more formal"
item_df[22,3] <- "I have an understanding of how treatment is going to help me"
item_df[23,3] <- "I would like my therapist to focus more on my feelings / more on my cognitions"
item_df[25,3] <- "Now I understand what I need to do or work on to get better"

pander(item_df)
```

itemId	itemName	question
1	fearLosingFoodControl	I am afraid I will lose control when it comes to food.
2	therapistTechniqueExerciseQuantity	I would like my therapist to use less/more techniques and exersises
3	therapistAcceptsAsPerson	I feel that my therapist accepts me as a person
4	workToAvoidPainfulThoughts	I spend a lot of energy trying not to think about things that hurt.
5	tryToAvoidSomeEmotions	I try very hard not to feel certain emotions.
6	trappedAndUncertain	I feel like I am trapped and don't know what to do.
7	concernedDrugDependent	I am concerned that I'm dependent on alcohol/drugs.
8	eatingRestrictsSocial	My eating habits restrict my ability to be social.
9	breathRacingOrNumbness	I experience shortness of breath, racing heart, and/or numbness and tingling in my hands or face.
10	drugUseInterferes	My alcohol and/or drug use interferes with my ability to function.
11	everythingTooMuchToHandle	I constantly feel that I can't handle things in my life.
12	needToReduceDrugUse	I think that I need to cut down on my drinking / drug use.
13	betterOffDead	I think it would be better if I were dead.
14	wantToDiscussMedicationConcern	I have concerns about my medication that I would like to discuss.
15	fearLoseControlSuicide	I am scared that I might lose control and kill myself.
16	feelMyMedicationHelps	I think that my psychiatric medication helps me.
17	feelingDepressed	I am feeling depressed.
18	therapistUnderstandsMe	I feel that the therapist understands me and understands why I am in treatment now
19	focusOnTherapistRelationship	I would like my therapist to focus less/more on the relationship between us
20	importantToControlEating	It is very important to control what and how much I eat.
21	therapistHumorFormalQuantity	I would like my therapist to show their personality more/be more formal

itemId	itemName	question
22	understandTreatmentHelp	I have an understanding of how treatment is going to help me
23	therapistFeelingCognitionFocus	I would like my therapist to focus more on my feelings / more on my cognitions
24	discomfortPrefersAlone	I feel so uncomfortable around others that I often prefer to be alone.
25	understandPathToGetBetter	Now I understand what I need to do or work on to get better
26	givenUpHopeForFuture	I have given up hope for a better future.
27	effortReadingAndPlanningFood	I spend excessive time thinking about food and planning my meals.
28	iAmWorthless	I am worthless.
29	mostlyRestlessUneasy	I feel restless and uneasy most of the time.

I create a new dataframe to store the variables I want to use in the analysis, including the new itemId (and discard item names and questions).

```
# Create nfdata dataframe
nfdata <- nfdata_raw %>% left_join(item_df, by = "itemName") %>%
  select(patId, age, sex, session, itemId, rating)
```

Next, I create a dataframe for patients and assign each patient a number to replace the long unique patient ID.

```
# Create patient dataframe
patient_df <- nfdata_raw %>% distinct(patId, .keep_all = TRUE) %>%
  mutate(patN = row_number()) %>%
  select(patN, patId, age, sex, n_sessions)
```

As sessions are numbered by the total treatment series, but the dataset has unique patient IDs for each patient for each year, I am giving the sessions a new ranking to facilitate analysis.

```
# Rank sessions
session_ranked <- nfdata_raw %>%
  group_by(patId) %>%
  distinct(session, .keep_all = TRUE) %>%
  mutate(session_rank = order(order(session))) %>%
  select(patId, session, session_rank) %>%
  arrange(patId) %>%
  ungroup()

pander(tail(session_ranked, 50))
```

patId	session	session_rank
aa9dd236c65890fb0c03984a96095c15	10	3
aa9dd236c65890fb0c03984a96095c15	13	4
aaa9d6d24c48de4050c735b8ca6cc2e8	1	1
aaa9d6d24c48de4050c735b8ca6cc2e8	4	2
aaa9d6d24c48de4050c735b8ca6cc2e8	5	3
aaa9d6d24c48de4050c735b8ca6cc2e8	7	4

patId	session	session_rank
aaa9d6d24c48de4050c735b8ca6cc2e8	8	5
aaa9d6d24c48de4050c735b8ca6cc2e8	9	6
aaa9d6d24c48de4050c735b8ca6cc2e8	10	7
aad4ae72eba62bd71681b4b46ad80a7a	1	1
aad4ae72eba62bd71681b4b46ad80a7a	3	2
aad4ae72eba62bd71681b4b46ad80a7a	6	3
aad4ae72eba62bd71681b4b46ad80a7a	7	4
aad4ae72eba62bd71681b4b46ad80a7a	9	5
aad4ae72eba62bd71681b4b46ad80a7a	11	6
aade214d378b6a061067bbec2ac83010	1	1
aade214d378b6a061067bbec2ac83010	2	2
aade214d378b6a061067bbec2ac83010	3	3
aade214d378b6a061067bbec2ac83010	5	4
aade214d378b6a061067bbec2ac83010	6	5
aade214d378b6a061067bbec2ac83010	7	6
aade214d378b6a061067bbec2ac83010	8	7
aade214d378b6a061067bbec2ac83010	9	8
aade214d378b6a061067bbec2ac83010	10	9
aade214d378b6a061067bbec2ac83010	11	10
aae83f454c5186ca537e878692f18d54	1	1
aae83f454c5186ca537e878692f18d54	2	2
aae83f454c5186ca537e878692f18d54	3	3
aae83f454c5186ca537e878692f18d54	4	4
aaea1d6990eedad36412d6ced78f0c13	3	1
aaea1d6990eedad36412d6ced78f0c13	7	2
aaea1d6990eedad36412d6ced78f0c13	8	3
aaea1d6990eedad36412d6ced78f0c13	9	4
aaea1d6990eedad36412d6ced78f0c13	10	5
aaea1d6990eedad36412d6ced78f0c13	11	6
aafea28a62682dd50c99209ed0df32d1	41	1
aafea28a62682dd50c99209ed0df32d1	44	2
aafea28a62682dd50c99209ed0df32d1	49	3
aafea28a62682dd50c99209ed0df32d1	50	4
aafea28a62682dd50c99209ed0df32d1	53	5
aafea28a62682dd50c99209ed0df32d1	54	6
aafea28a62682dd50c99209ed0df32d1	56	7
aafea28a62682dd50c99209ed0df32d1	61	8
aafea28a62682dd50c99209ed0df32d1	62	9
aafea28a62682dd50c99209ed0df32d1	64	10
aafea28a62682dd50c99209ed0df32d1	66	11
aafea28a62682dd50c99209ed0df32d1	71	12
aafea28a62682dd50c99209ed0df32d1	75	13
aafea28a62682dd50c99209ed0df32d1	76	14
aafea28a62682dd50c99209ed0df32d1	79	15

Finally, I update the nfdata dataframe with session ranks and patient numbers, convert variables, and inspect the number of unique patients in the final dataset and the dataset structure.

```
# Update nfdata dataframe with session rank
nfdata <- nfdata %>%
  left_join(session_ranked, by = c("patId" = "patId", "session" = "session")) %>%
  arrange(patId)
```

```r
# Update nfdata with patient number and select features
nfdata <- nfdata %>% left_join(patient_df %>% select(patId, patN, n_sessions), by = "patId") %>%
  select(patN, age, sex, session, session_rank, n_sessions, itemId, rating)

# Convert variables
str(nfdata)
```

```
## Classes 'data.table' and 'data.frame':   739531 obs. of  8 variables:
##  $ patN        : int  1361 1361 1361 1361 1361 1361 1361 1361 1361 1361 ...
##  $ age         : int  19 19 19 19 19 19 19 19 19 19 ...
##  $ sex         : chr  "F" "F" "F" "F" ...
##  $ session     : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ session_rank: int  1 1 1 1 1 1 1 1 1 1 ...
##  $ n_sessions  : int  22 22 22 22 22 22 22 22 22 22 ...
##  $ itemId      : Factor w/ 29 levels "1","2","3","4",..: 27 25 24 20 17 5 15 16 13 28 ...
##  $ rating      : chr  "2" "6" "3" "2" ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

```r
nfdata$rating <- as.numeric(nfdata$rating)
nfdata$sex[nfdata$sex == ""] <- NA #Replace blanks with NAs
nfdata$sex <- as.factor(nfdata$sex)

length(unique(nfdata$patN))
```

```
## [1] 3490
```

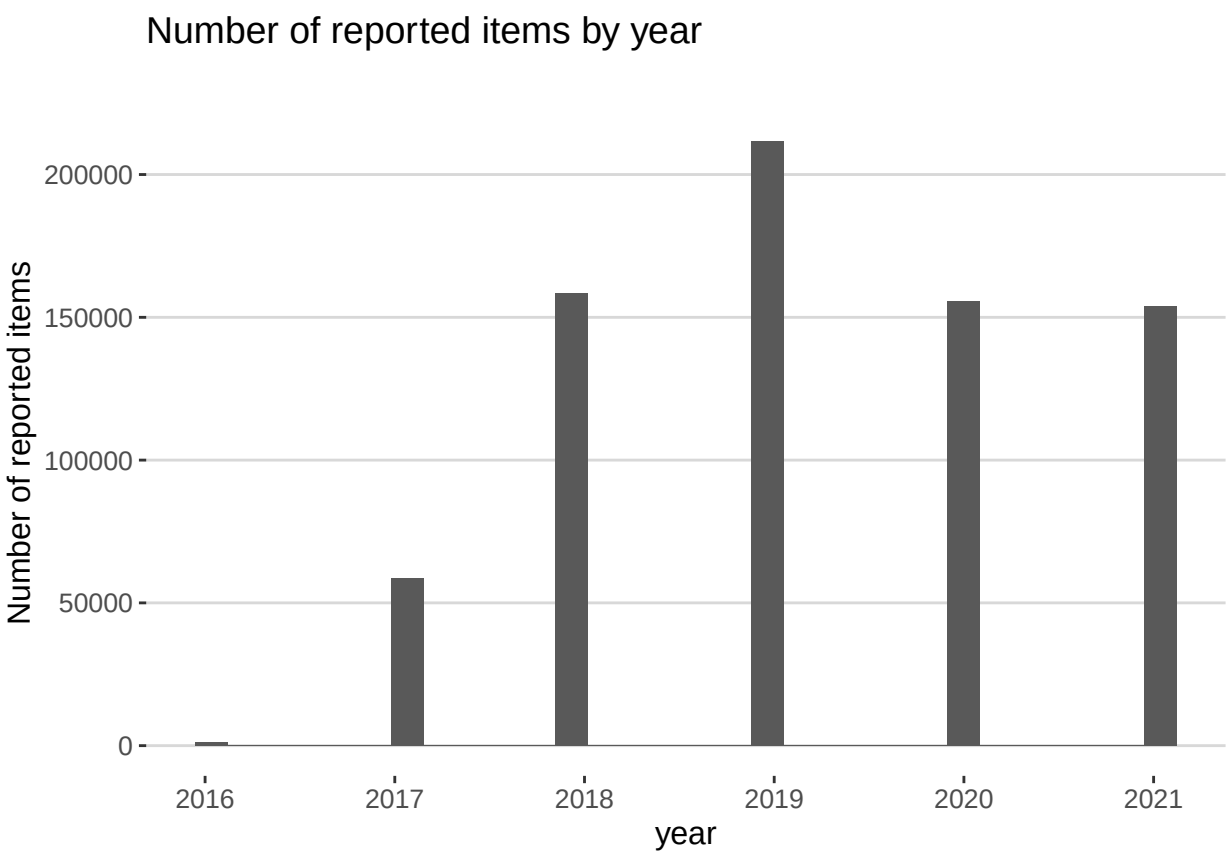
```r
str(nfdata)
```

```
## Classes 'data.table' and 'data.frame':   739531 obs. of  8 variables:
##  $ patN        : int  1361 1361 1361 1361 1361 1361 1361 1361 1361 1361 ...
##  $ age         : int  19 19 19 19 19 19 19 19 19 19 ...
##  $ sex         : Factor w/ 3 levels "F","M","O": 1 1 1 1 1 1 1 1 1 1 ...
##  $ session     : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ session_rank: int  1 1 1 1 1 1 1 1 1 1 ...
##  $ n_sessions  : int  22 22 22 22 22 22 22 22 22 22 ...
##  $ itemId      : Factor w/ 29 levels "1","2","3","4",..: 27 25 24 20 17 5 15 16 13 28 ...
##  $ rating      : num  2 6 3 2 4 3 1 4 1 2 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

2.2 Dataset Description and Exploration

Data used in this research have been collected as part of implementation follow up for the research and innovation project. The data is collected in the US over a 5 year period and use of the feedback system have varied over time.

```r
# Reported items by year
nfdata_raw %>% ggplot(aes(year)) +
  geom_histogram() +
  ggtitle("Number of reported items by year", subtitle = "") +
  ylab("Number of reported items") +
  theme_hc()
```



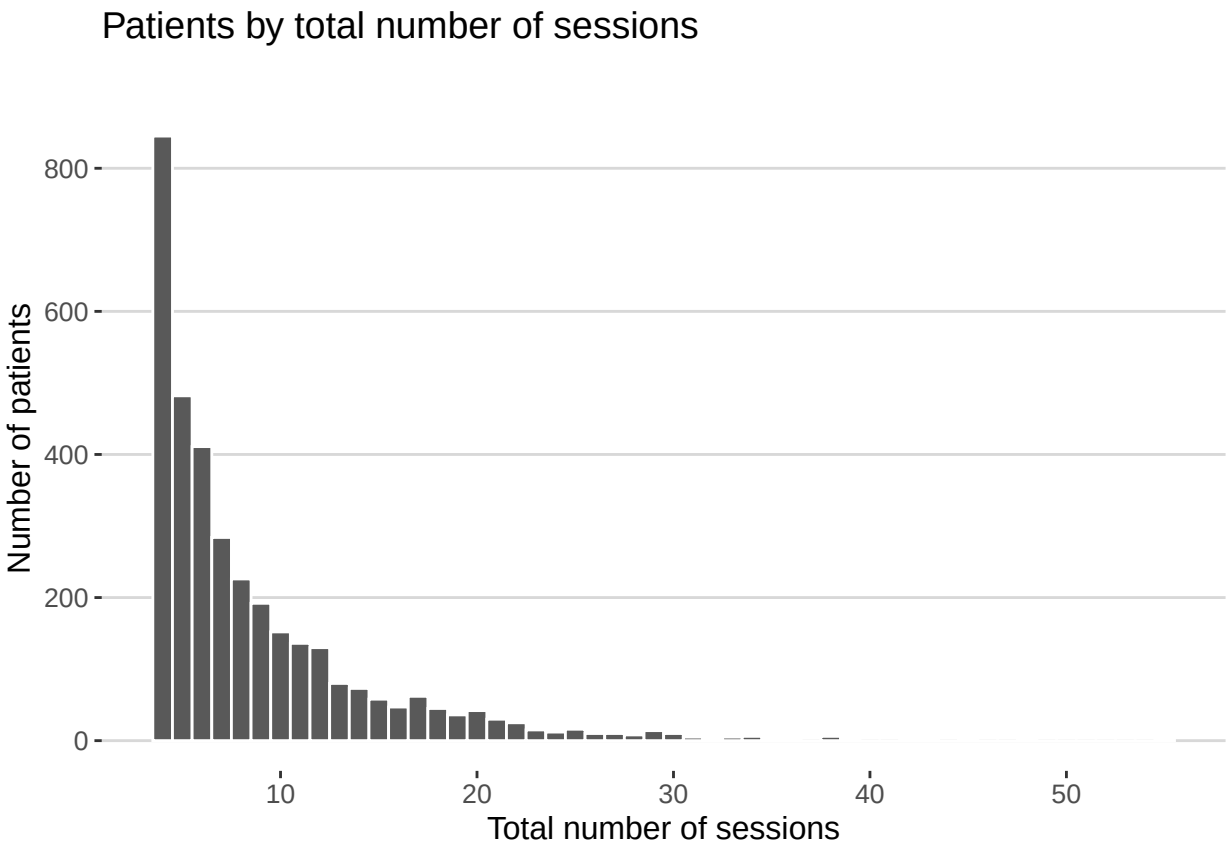
The dataset used for this research contains multivariate time series data. Real world data introduces some complexity to the analysis. An example is between patient variance in patient feedback. Some patients have registered feedback every session, whilst others have only registered every second or third session. Additionally, patient IDs are reset every year meaning treatment series spanning more than one year are split to different patient IDs making it impossible to find the actual first sessions for several patients. Consequently, when I use the term “three first sessions” in this report I refer to the three first patient feedback reports for a given patient ID that not necessarily corresponds to the three first treatment sessions for a patient.

This dataset reflects how patient reported outcome measures are used in real situations. Both patients and clinicians have preferences regarding frequency of reporting. This adds variance to the models we want to train, but the models will be more usable as reflections of the real situation.

The dataset contains time series data for individual patients over several sessions in a long/tidy format (each item observation is one row). This is the standard for data frames, but to analyze the treatment effect of each patient over time I need to transform the data into a wide format where each patient is on one row, and all the individual session responses are in columns. This process is described in Chapter 3: Data Pre-processing.

Visualization of the distribution of patients by the total number of sessions:

```
# Plot distributions of sessions per patient
patient_df %>% ggplot(aes(n_sessions)) +
  geom_histogram(binwidth = 1, color = "white") +
  ggtitle("Patients by total number of sessions", subtitle = "") +
  ylab("Number of patients") +
  xlab("Total number of sessions") +
  theme_hc()
```



The average number of sessions per patient:

```
# Mean number of sessions
mean(patient_df$n_sessions)
```

```
## [1] 8.742693
```

```
# Standard deviation of number of sessions
sd(patient_df$n_sessions)
```

```
## [1] 6.021052
```

Most patients in the dataset have less that 10 reports/sessions. The variance is quite high, and as we can see some patient have treatment series with between 30 and 50 reports/sessions.

The dataset is in a long format with the following variables:

```
# Inspect nf_data structure
glimpse(nfdata)
```

```
## Rows: 739,531
## Columns: 8
## $ patN         <int> 1361, 1361, 1361, 1361, 1361, 1361, 1361, 1361, 1361, 136~
## $ age          <int> 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 1~
## $ sex          <fct> F, F, F, F, F, F, F, F, F, F, F, F, F, F, F, F, F, F, F, ~
## $ session      <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
```

```
## $ session_rank <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ n_sessions   <int> 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 2~
## $ itemId       <fct> 27, 25, 24, 20, 17, 5, 15, 16, 13, 28, 22, 23, 3, 29, 26,~
## $ rating       <dbl> 2, 6, 3, 2, 4, 3, 1, 4, 1, 2, 6, 0, 6, 4, 1, 0, 0, 0, 1, ~
```

The total number of patients and items in itemId:

```
# Inspect number of items in dataset
length(unique((nfdata$itemId)))
```

```
## [1] 29
```

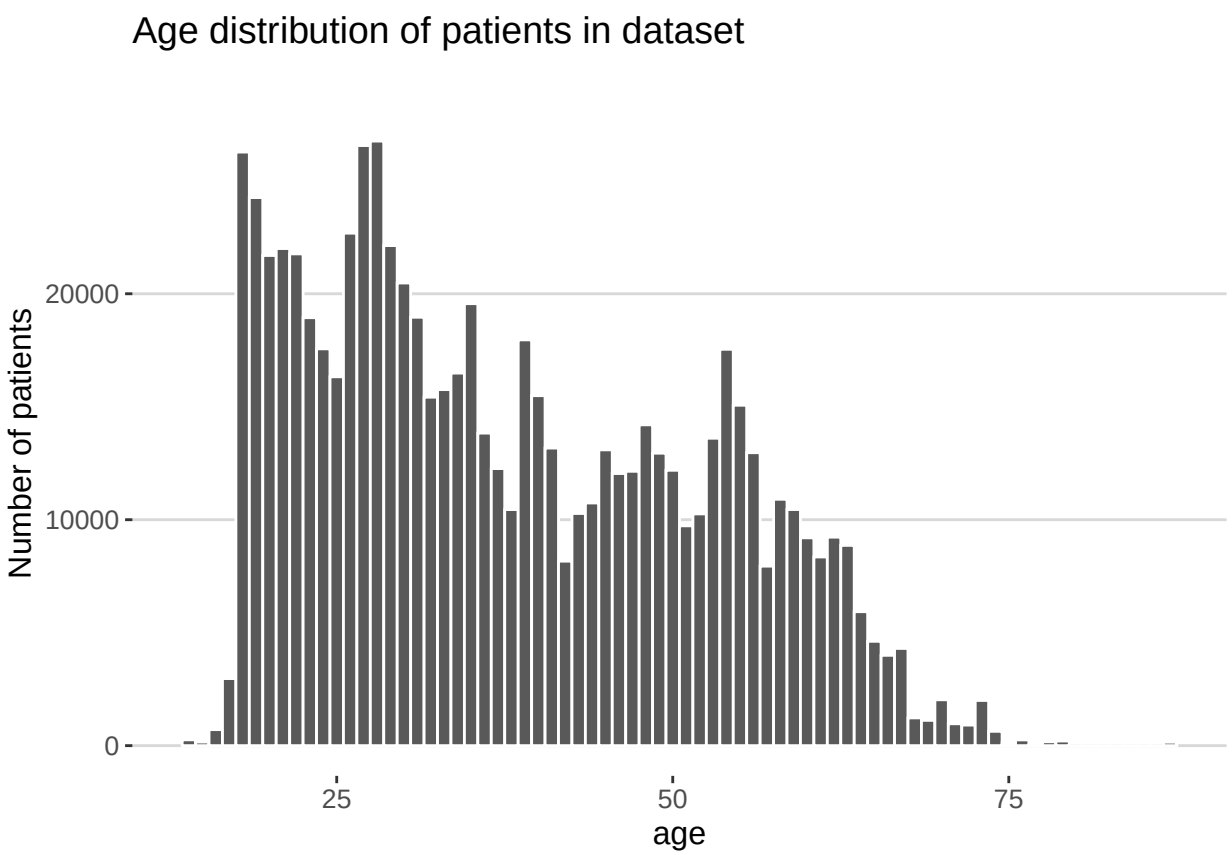
```
# Inspect number of patients in dataset
length(unique((nfdata$patN)))
```

```
## [1] 3490
```

Of the 29 items in itemID 25 are scored on a 1-7 likert scale. Four items are centered around 0 with a scale from -5 to 5. These items are: 2, 19, 21 and 23. Some items have a reversed score with 7 being the best response. These are items 3, 16, 18, 22 and 25.

Patients age distribution:

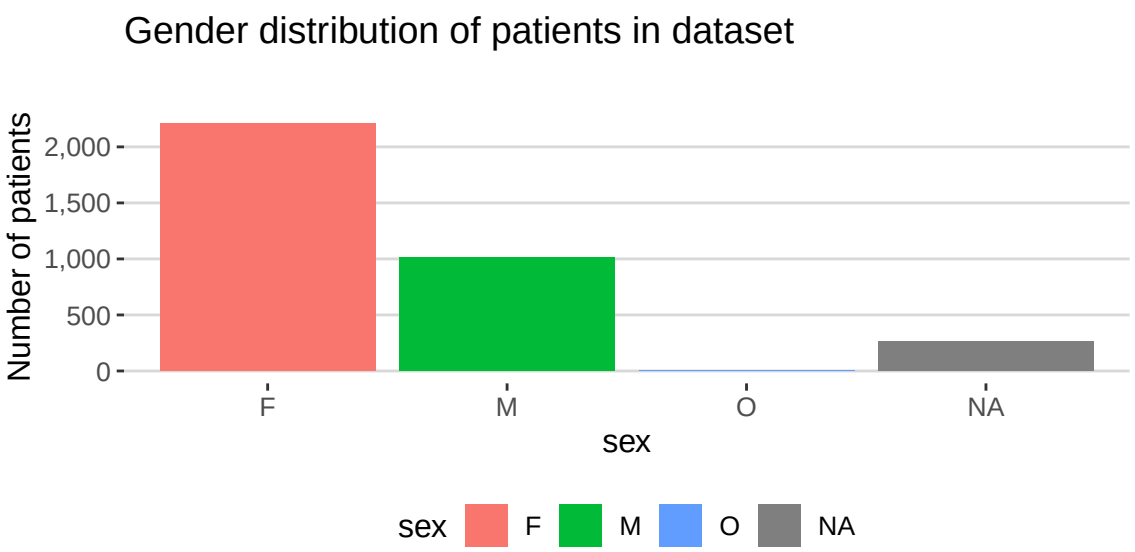
```
# Age distribution
nfdata %>% ggplot(aes(age)) +
  geom_histogram(binwidth = 1, color = "white") +
  ggtitle("Age distribution of patients in dataset", subtitle = "") +
  ylab("Number of patients") +
  theme_hc()
```



Patients are of all ages but the number is declining with increasing age.

Gender distribution of patients:

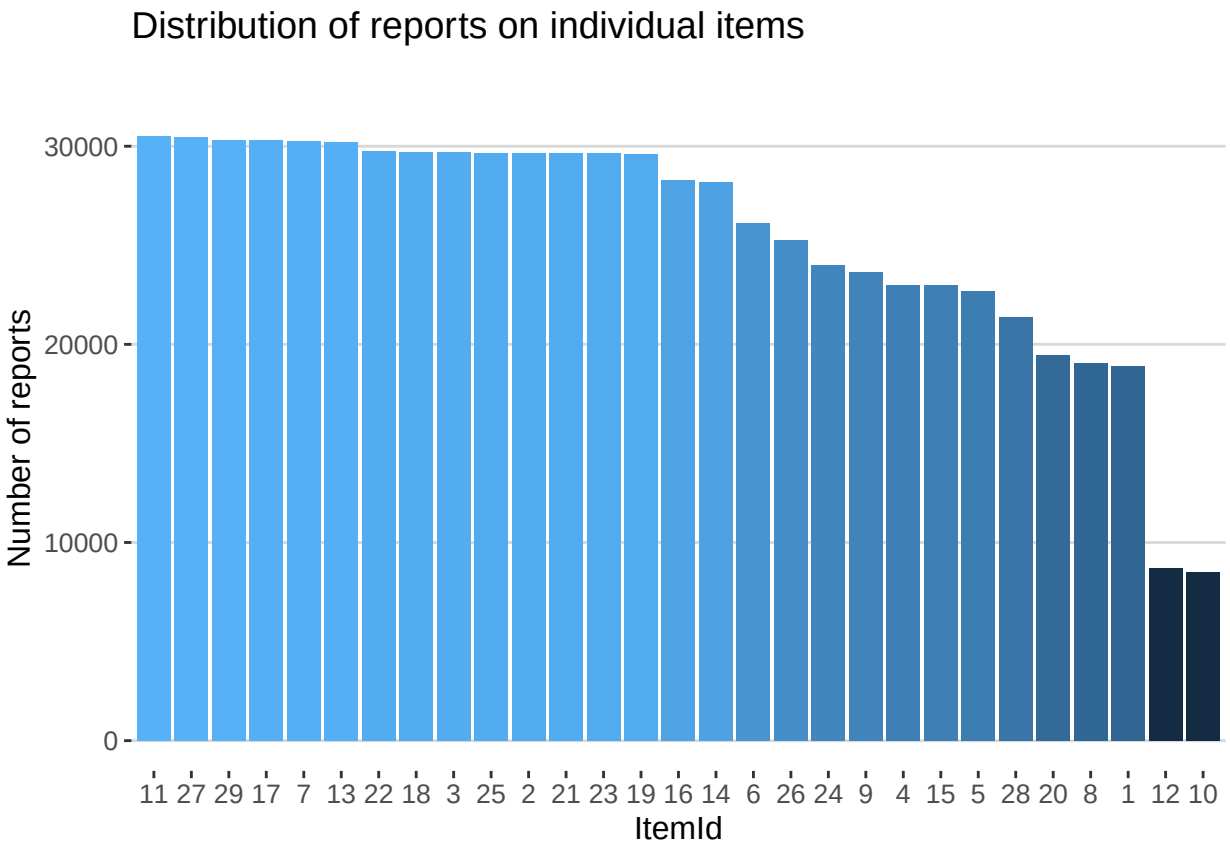
```
# Gender distribution
nfdata %>% distinct(patN, .keep_all = TRUE) %>%
  ggplot(aes(sex, fill = sex)) +
  geom_bar() +
  ggtitle("Gender distribution of patients in dataset", subtitle = "") +
  scale_y_continuous(labels = comma) +
  ylab("Number of patients") +
  theme_hc()
```



Females are twice as prevalent as males. A few patients are characterized as “other” sex, and there are quite a few missing values.

Distribution of reports on individual items:

```
# Item report distribution
nfdata %>% group_by(itemId) %>%
  mutate(count = n()) %>%
  ggplot(aes(x = reorder(itemId, -count), fill = count)) +
  geom_bar() +
  ggtitle("Distribution of reports on individual items", subtitle = "") +
  xlab("ItemId") +
  ylab("Number of reports") +
  theme_hc() +
  theme(legend.position = "NULL")
```



A little more than half of the items seems to be reported in almost every session. Two items, 12 and 10, are rarely reported. These are items on the substance abuse scale which are only relevant for a selection of patients.

2.3 How to define the outcome in unlabeled data?

A major challenge with this dataset is that it is unlabeled. It is unknown how patients and clinicians would characterize the effect of treatment. However, a major reason for using a patient reported feedback measure is to have continuous measurement of treatment effect and outcome. The answer might be present in the data. What defines a “good” outcome and what defines lack of effect of treatment? Through discussions with the research and clinical team of Norse Feedback we have arrived at using one item as the labeling outcome, item 11. This is considered the item that most often would correspond to treatment effect. This question is formulated: “I constantly feel that I can’t handle things in my life.”. Response is given on a likert scale between 1 and 7, where 1 corresponds to “Not at all” and 7 corresponds to “True for me”. In the dataset some patients have consistently low ratings on this item, indicating that this is not a concern for them. In the data pre-processing phase I will have to consider if this labeling method is suitable for all patients. Outside the scope of this research is considering that different items might correspond better to treatment effect in different patients, and that better outcome labels might be used for sub-classes of patients.

Another important consideration is what represents meaningful improvement on an item rating. In many studies outcomes needs to be improved by 50% or more to be considered a “good” outcome. On a 1-7 likert scale this is restrictive. I have decided to use an improvement of 1 for difference between first and last session as the “good” outcome cut-off. E.g. a patient starting with reporting 5 on item 11 in the first session would need to report 4 or lower in the final session to be labeled as a “good” outcome. As an exception to this rule a change from 7 as the first value to 6 on the final will not be regarded as meaningful improvement.

This is a simple approach to labeling the data, with multiple sources for error. The first time patients report on the

feedback system they have to respond to questions that are new to them. In the following sessions they are more familiar with the questions and how they relate to each other. Stable response on an item might not be present until the patient has responded more than a few times. A possible solution could be to use the average score over the two first sessions as the pre-value. Using only pre and post values as a measure for treatment effect excludes the datapoints between. In a treatment series the outcome measure could start low, increase as treatment is in progress and fall back to the starting level towards the end of treatment. Using only pre and post values this would be labeled as no treatment effect. This could be explored further to find if the curvature of the outcome item over time adds information about treatment effect. Using a single item as the outcome will not fit all patients. Approaches to solve this could be to use different outcomes for different patient groups, or to construct an outcome ensemble consisting of several outcomes measures, where the final outcome is a result of an ensemble vote.

The process of labeling the dataset is detailed in Chapter 3: Data Pre-processing.

2.4 Handling Missing Values

The adaptive methodology of Norse Feedback leads to a lot of missing values as the system only require the patient to report on relevant items at any given time. This is valuable for the patient, but makes it harder to analyze the data. The table below show the number of NAs for each of the variables in the dataset.

age	sex	session	session_rank	n_sessions	itemId	rating
0	48578	0	0	0	0	49010

We can also use the `lares` library to explore missing data and calculate missingness (in percent).

```
# Explore missing data using the lares package
if(!require(lares)) install.packages("lares", repos = "http://cran.us.r-project.org")
missingness(nfdata)
```

variable	missing	missingness
rating	49010	6.63
sex	48578	6.57

The proportion of missing data will increase when I pivot the dataset to a wide format and NAs are introduced for missing values in each column. Many ML algorithms struggle to make sense out of datasets with a lot of missing data and methods for preparing datasets with missing values have been developed. I will briefly describe important considerations when pre-processing sparse datasets, different practical approaches and discuss my conclusions for how to handle missing values in this research.

2.4.1 Missing data theoretical framework

In statistical theory missing values occur at two levels, the unit level and the item level.[1] The dataset includes no conclusive information about the missingness of data on a unit level. Reports are missing for every session for some patients, but we do not know if this is by design or a result of patients refusing to complete the feedback report. The missing values in the dataset occur on an item level and I will focus on this problem. Three aspects of missing values must be considered before deciding on an approach: (1) the proportion of missing data, (2) the missing

[1] Dong Y, Peng CY. Principled missing data methods for researchers. Springerplus. 2013;2(1):222. Published 2013 May 14. doi:10.1186/2193-1801-2-222

data mechanisms and (3) the patterns of missing data. These aspects are vital when considering imputation for missing data.

Proportion of missing data

According to the article by Dong and Chen the proportion of missing data is directly related to the quality of statistical inferences. Yet, there is no established cutoff from the literature regarding an acceptable percentage of missing data in a data set for valid statistical inferences.[1] The authors of a 2019 paper investigated the impact of missing data proportions when using methods for multiple imputations and concluded that the proportion should not be used to guide decisions.[2].

Missing data mechanisms

The theory of missing data mechanisms was formulated by Rubin in 1976.[3] In this article he described three mechanisms under which missing data can occur: (1) missing at random (MAR), (2) missing completely at random (MCAR), and (3) missing not at random (MNAR). Detailing these mechanisms is beyond the scope of this report, but I will include a brief summary of these mechanisms from a free online book by Stef Van Buuren.[4] If the probability of a value being missing is the same for all cases, then the data are said to be MCAR, if the probability of a value being missing is the same only within groups defined by the observed data, then the data are MAR, and if neither MCAR nor MAR holds, then we speak of MNAR. MNAR means that the probability of a value being missing varies for reasons that are unknown to us. A final mechanism is data missing by design.

For this dataset the data collection methodology actively facilitates missing data as a part of the adaptive and user-friendly goal of the system. The mechanisms of missing data in the dataset are thus probably a mix of MNAR (e.g. sex) and MAR for missing values that the system expected response for, but mostly missing by design.

Patterns of missing data

Three patterns of missing data can be observed: univariate, monotone and arbitrary. An intuitive explanation is provided by Dong and Peng[1]: *"Suppose there are p variables, denoted as,* Y_1, Y_2, ..., Y_p. *A data set is said to have a univariate pattern of missing if the same participants have missing data on one or more of the* p *variables. A dataset is said to have a monotone missing data pattern if the variables can be arranged in such a way that, when* Y_j *is missing,* Y_{j+1}, Y_{j+2}, ..., Y_p *are missing as well."……"If missing data occur in any variable for any participant in a random fashion, the data set is said to have an arbitrary missing data pattern."*

The data collection methodology of the Norse Feedback system adaptive methodology enable items that have been closed on previous response to be opened again based on trigger items. Due to this methodology the pattern of missing data in this dataset can be considered to be arbitrary.

2.4.2 Practical approaches to handling missing values

1. Deleting rows with missing values

A very basic approach would be to just delete patients with missing values. For my dataset this would reduce the total number of patients substantially. I have not yet made a wide dataset and seen the consequences of introducing NAs, but from the distribution of individual items reponses we can infer that for some items almost 2/3 of the responses will include NAs. A common sense strategy could be to remove items with low response proportion, and then remove the rows/patients that still included NAs. However, the APA Task Force on Statistical Inference have explicitly warned against this strategy as it has been shown to lead to bias and/or inefficient estimates in most situations.

2. Mean/median imputation

[2]Madley-Dowd, Paul & Hughes, Rachael & Tilling, Kate & Heron, Jon. (2019). The proportion of missing data should not be used to guide decisions on multiple imputation. Journal of Clinical Epidemiology. 110. 10.1016/j.jclinepi.2019.02.016.

[3]DONALD B. RUBIN, Inference and missing data, Biometrika, Volume 63, Issue 3, December 1976, Pages 581–592, https://doi.org/10.1093/biomet/63.3.581

[4]https://stefvanbuuren.name/fimd/

A very common approach to missing values is to impute the column/item mean or median for missing values. This makes sense if the variable e.g. is patient weight. The average weight for all patients would be a decent guess for a missing value. This approach could be refined by using the average for males for a male patient and the average for females for a female patient. For my dataset this would not make any sense. Missing values on items indicate that these items are of less relevance for the patient and using the mean or median for responses from patients where the item is relevant would result in a rating that is either too high (on items with 1 as best response), or too low (on items with 7 as best response). A better approach would be to impute 1 or 7 for these items, corresponding to the best response on the item. Another approach would be to impute 0. This would eliminate the effect of the response in a linear model.

3. Predicting missing values

In theory any algorithm could be used to predict the missing values. Two methods that often are used are linear regression and k-nearest-neighbors (KNN). With linear regression we would train a model using the available data to create a linear model that would predict missing values. KNN uses similarity/distance measures to find the k most similar observations and predicts the missing value from these observations. Missing values imputation is easy to implement with e.g. the `MICE` library[5]. This handy package contains 28 methods for imputation including various mean imputation methods, various linear and logistic regression methods and classifiers as random forest.

4. Using algorithms that tolerate missing values

The final approach would be to ignore the missing values and use an algorithm that tolerate datasets with missing values. Classification trees and KNN are algorithms that in theory tolerate missing values. The KNN algorithms available for R do not handle missing values however. A number of classification tree algorithms tolerate missing data. Random forests could be an approach as well as various gradient boosting tree classifiers. The "Extreme Gradient Boosting" algorithm `XGBoost`[6] is both an efficient and accurate classifier and have been used by winners of many Kaggle competitions. I will later in this report briefly explain how this algorithm handles missing values.

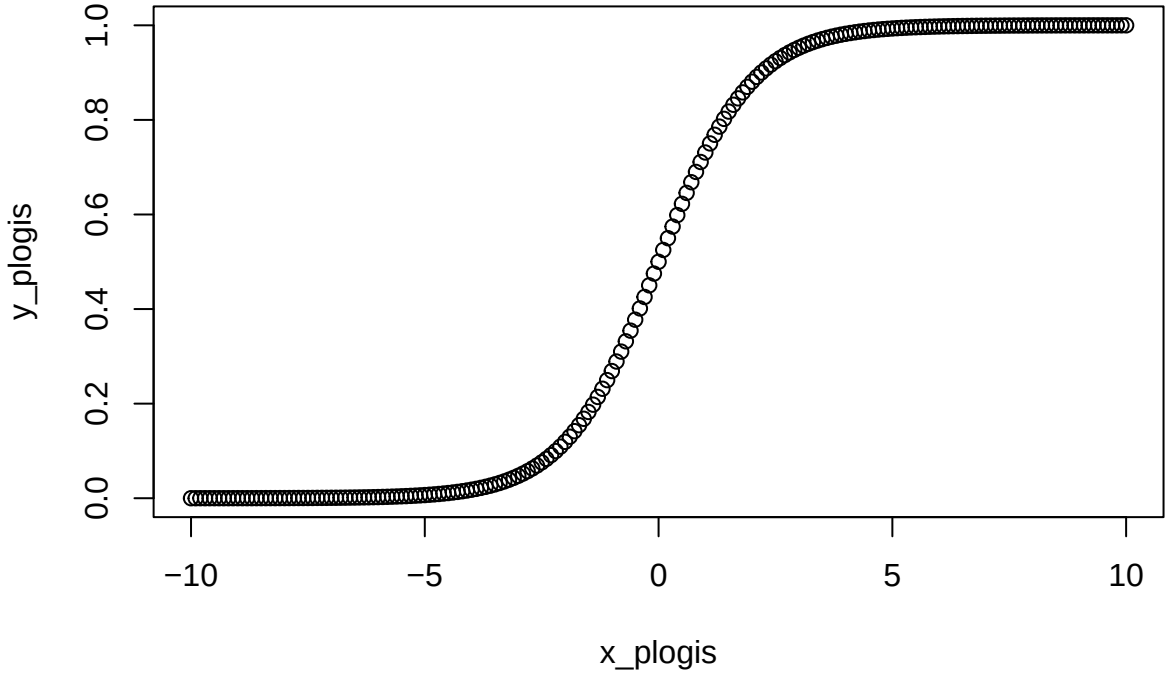
One goal of this project is to create an explainable model for clinical use. Using methods for multiple imputation may increase the accuracy of the model, but would introduce a new layer that potentially obfuscates the interpretability of results. For this reason I have decided to use algorithms that handle missing values and produce explainable results, and/or impute 0s for missing values.

2.5 Logistic Regression

Logistic regression is a powerful statistical method for classification and modeling a binominal (or multinominal) outcome with one or more features. To do so logistic regression measures the relationship between the categorical dependent variable and the features by estimating probabilities using the cumulative logistic distribution.

[5] van Buuren S, Groothuis-Oudshoorn K (2011). "mice: Multivariate Imputation by Chained Equations in R." Journal of Statistical Software, 45(3), 1-67. doi: 10.18637/jss.v045.i03

[6] Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785–794). New York, NY, USA: ACM. https://doi.org/10.1145/2939672.2939785



For the dataset at hand I plan for a binary outcome, “0” for no treatment response, and “1” for treatment response. Wikipedia on binary outcomes: *Like other forms of regression analysis, logistic regression makes use of one or more predictor variables that may be either continuous or categorical. Unlike ordinary linear regression, however, logistic regression is used for predicting dependent variables that take membership in one of a limited number of categories (treating the dependent variable in the binomial case as the outcome of a Bernoulli trial) rather than a continuous outcome. Given this difference, the assumptions of linear regression are violated.. In particular, the residuals cannot be normally distributed. In addition, linear regression may make nonsensical predictions for a binary dependent variable. What is needed is a way to convert a binary variable into a continuous one that can take on any real value (negative or positive). To do that, binomial logistic regression first calculates the odds of the event happening for different levels of each independent variable, and then takes its logarithm to create a continuous criterion as a transformed version of the dependent variable.*[7] Without going into further details that are outside of the scope of this report the logistic regression function can be formulated like this:

$$p = \frac{1}{(1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 ..... \beta_n X_n)})}$$

Where p is the probability of $Y = 1$, β_0 is the intercept, and $\beta_1 ... n$ are the coefficients for the features $X_1 ... n$.

Logistic regression in `R` using `glm()` from the `caret` library does not tolerate missing values. To use this algorithm I will have to either remove observations with missing values or do some sort of imputation for missing values. We can infer from the logistic regression function that imputing 0s will allow the function to be trained and to predict on new data. Features with missing values for an individual will after imputing 0s have no impact on the prediction for this individual.

[7] https://en.wikipedia.org/wiki/Logistic_regression

2.6 XGBoost

The `XGBoost` library[8] implements gradient boosting decision tree algorithms. The name of the library is short for e**X**treme **G**radient **Boost**ing and it was developed to push the limits of computations resources for gradient boosted tree algorithms by Tianqi Chen[9]. The story and lessons behind the evolution of XGBoost have been published by Chen[10]. XGBoost is one of the most winning algorithms in Kaggle competitions[11] and has become an essential part of many data scientists toolkit. Gradient boosting algorithms are ensemble algorithms training a fixed number of trees set by the user, or terminating training when no further improvement is achieved. For each iteration the algorithm tries to minimize the residual error of the previous tree. Results of each tree are combined for the final result. Since training is performed in an additive sequential manner the number of trees to train is an important parameter. Training more trees do not necessarily lead to better performance due to the possibility of overfitting the model to the training data. Compared to other gradient boosting algorithms XGBoost is distinguished by also implementing regularization mechanisms to prevent overfitting.

2.6.1 XGBoost advantages

There are two main reasons to use XGBoost, execution speed and model performance. The library was designed to optimalize computational speed and enables the use of multiple CPU cores during training, as well as more advanced options for training large datasets over several machines. In benchmarks XGBoost outperforms other gradient boosting decision tree algorithms on speed. Model performance is confirmed over time through consistent great performance in ML competetions e.g. Kaggle.

2.6.2 Hyperparameters

The `XGBoost` algorithm has many hyperparameters that can be used to tune a model for better performance on a dataset. The total number of hyperparameters and individual choices can be a bit daunting at first, I will give a brief introduction to hyperparameters here to enable the readers to follow the model training in the results section. The following are descriptions from the `XGBoost` library documentation for the selected hyperparameters I expect to tune and use in my model, see the documentation for a full list of parameters and choices.[12]

General Parameters

- `booster` [default= `gbtree`]
  - Which booster to use. Can be `gbtree`, `gblinear` or `dart`; `gbtree` and `dart` use tree based models while `gblinear` uses linear functions.
- `verbosity` [default=1]
  - Verbosity of printing messages. Valid values are 0 (silent), 1 (warning), 2 (info), 3 (debug). Sometimes XGBoost tries to change configurations based on heuristics, which is displayed as warning message. If there's unexpected behaviour, please try to increase value of verbosity.

Parameters for Tree Booster

- `eta` [default=0.3, alias: `learning_rate`]

[8] http://arxiv.org/abs/1603.02754

[9] https://www.quora.com/What-is-the-difference-between-the-R-gbm-gradient-boosting-machine-and-xgboost-extreme-gradient-boosting/answer/Tianqi-Chen-1

[10] https://sites.google.com/site/nttrungmtwiki/home/it/data-science---python/xgboost/story-and-lessons-behind-the-evolution-of-xgboost

[11] https://github.com/dmlc/xgboost/tree/master/demo#machine-learning-challenge-winning-solutions

[12] https://xgboost.readthedocs.io/en/stable/

  - Step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features, and `eta` shrinks the feature weights to make the boosting process more conservative.
  - range: [0,1]

- `gamma` [default=0, alias: `min_split_loss`]
  - Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger `gamma` is, the more conservative the algorithm will be.
  - range: [0,∞]

- `max_depth` [default=6]
  - Maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit. 0 is only accepted in `lossguide` growing policy when `tree_method` is set as `hist` or `gpu_hist` and it indicates no limit on depth. Beware that XGBoost aggressively consumes memory when training a deep tree.
  - range: [0,∞] (0 is only accepted in `lossguide` growing policy when `tree_method` is set as `hist` or `gpu_hist`)

- `min_child_weight` [default=1]
  - Minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than `min_child_weight`, then the building process will give up further partitioning. In linear regression task, this simply corresponds to minimum number of instances needed to be in each node. The larger `min_child_weight` is, the more conservative the algorithm will be.
  - range: [0,∞]

- `subsample` [default=1]
  - Subsample ratio of the training instances. Setting it to 0.5 means that XGBoost would randomly sample half of the training data prior to growing trees. and this will prevent overfitting. Subsampling will occur once in every boosting iteration.
  - range: (0,1]

- `colsample_bytree` [default=1]
  - `colsample_bytree` is the subsample ratio of columns when constructing each tree. Subsampling occurs once for every tree constructed.

- `lambda` [default=1, alias: `reg_lambda`]
  - L2 regularization term on weights. Increasing this value will make model more conservative.

- `alpha` [default=0, alias: `reg_alpha`]
  - L1 regularization term on weights. Increasing this value will make model more conservative.

Learning Task Parameters

- `objective` [default=reg:squarederror]
  - `binary:logistic`: logistic regression for binary classification, output probability

- `eval_metric` [default according to objective]
  - Evaluation metrics for validation data, a default metric will be assigned according to objective (rmse for regression, and logloss for classification, mean average precision for ranking)
    * `error`: Binary classification error rate. It is calculated as `#(wrong cases)/#(all cases)`. For the predictions, the evaluation will regard the instances with prediction value larger than 0.5 as positive instances, and the others as negative instances.

2.6.3 Missing values and XGBoost

XGBoost uses a process called sparsity-aware split finding by the authors of the paper introducing the algorithm. In practice this means that for every split in the tree a default direction is decided based on which of the two alternatives on average gives the best score. When there is a missing value for this split, the default direction is chosen and the path down the tree continues. This is a neat solution to the missing values problem, but introduces some bias into the model. However, the features with many missing values should in theory not be among the features that provides the best splits and as such will not be part of many trees. I will need to consider ow this might influence the model and which features to keep for analysis based on the proportion of missing values in the data pre-processing phase. Further reading for interested readers: "XGBoost is not black magic"[13]

2.7 SHAP - A machine learning output explanation approach

SHAP (SHapley Additive exPlanation) was proposed in 2016 by S.Lundberg and S. Lee.[14]. It is a game theoretic approach to explain the output of any machine learning model by measuring the features importance to the predictions. The foundation for SHAP is Shapeley values.[15] In the context of machine learning the core idea behind Shapley value based explanations of machine learning models is to use fair allocation results from cooperative game theory to allocate credit for a model's output $f(x)$ among its input features.[16]

The benefits of using SHAP are both at an overall and at a local level, as follows:

- At a global level, the collective SHAP values help to interpret and understand the model. They show how much each predictor contributes, either positively or negatively, to the target variable. It allows for very intuitive interpretation of the model structure and is generalisable across a number of different modelling methodologies.
- At a local level, each observation gets its own set of SHAP values (one for each predictor). This greatly increases transparency, by showing contributions to predictions on a case by case basis, which traditional variable importance algorithms are not able to do.[17]

The standard appraoch to evaluating feature importance in a tree based model is by assigning importance based on model gain from features. Scott Lundberg has published an excellent piece about the advantages of SHAP over this approach.[18]

Another popular model explanation approach is LIME. This approach works well with few features and on a global level, but SHAP has several advantages over LIME when dealing with a lot of features and in local explanations. Readers with further interest in approaches to interpretable machine learning might find this e-book interesting: "Molnar, Christoph. "Interpretable machine learning. A Guide for Making Black Box Models Explainable""[19]

[13] https://towardsdatascience.com/xgboost-is-not-black-magic-56ca013144b4
[14] https://proceedings.neurips.cc/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf
[15] https://en.wikipedia.org/wiki/Shapley_value
[16] https://shap.readthedocs.io/en/latest/overviews.html
[17] https://parker-fitzgerald.com/wp-content/uploads/2019/12/ML-Interpretability-SHAP-example.pdf
[18] https://towardsdatascience.com/interpretable-machine-learning-with-xgboost-9ec80d148d27
[19] https://christophm.github.io/interpretable-ml-book/

Chapter 3

Data Pre-processing

In this chapter I will prepare the dataset for analysis by calculating outcome and labeling the data, implement a strategy for missing values, do some feature engineering and finally split the dataset for training, test and validation.

3.1 Outcome Labeling

As discussed in Chapter 2 I need to add labels to the unlabeled dataset in order to perform supervised learning. First I want to again quickly inspect the dataset.

```
glimpse(nfdata)
```

```
## Rows: 739,531
## Columns: 8
## $ patN         <int> 1361, 1361, 1361, 1361, 1361, 1361, 1361, 1361, 1361, 136~
## $ age          <int> 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 1~
## $ sex          <fct> F, F, F, F, F, F, F, F, F, F, F, F, F, F, F, F, F, F, F, ~
## $ session      <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ session_rank <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ n_sessions   <int> 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 2~
## $ itemId       <fct> 27, 25, 24, 20, 17, 5, 15, 16, 13, 28, 22, 23, 3, 29, 26,~
## $ rating       <dbl> 2, 6, 3, 2, 4, 3, 1, 4, 1, 2, 6, 0, 6, 4, 1, 0, 0, 0, 1, ~
```

```
length(unique(nfdata$patN))
```

```
## [1] 3490
```

There are 3490 unique patients in the dataset. I will use item 11 as the outcome defining item.

```
# Outcome parameter is item 11
item_df[11]
```

itemId	itemName	question
11	everythingTooMuchToHandle	I constantly feel that I can't handle things in my life.

As we observed in the missing data exploration in Chapter 2, item 11 is the most prevalent in the dataset. I want to know how many missing values there are for item 11.

```r
# Missing values on item 11
nfdata %>% filter(itemId == 11) %>% summarize(n = sum(is.na(rating)))
```

n
614

I start by creating a dataframe to store improvement on the outcome item

```r
# Create a dataframe to store improvement in outcomes
outcomes_df <- nfdata %>%
  filter(itemId == 11) %>%
  group_by(patN) %>%
  mutate(imp = rating[which.min(session_rank)]-rating[which.max(session_rank)]) %>%
  ungroup() %>%
  arrange(patN)
```

An improvement of 1 on the 1-7 likert scale is a marginal improvement, and it is doubtful that for patients starting at 7 an improvement to 6 represents meaningful treatment effect. I will filter these patients and set their improvement to 0.

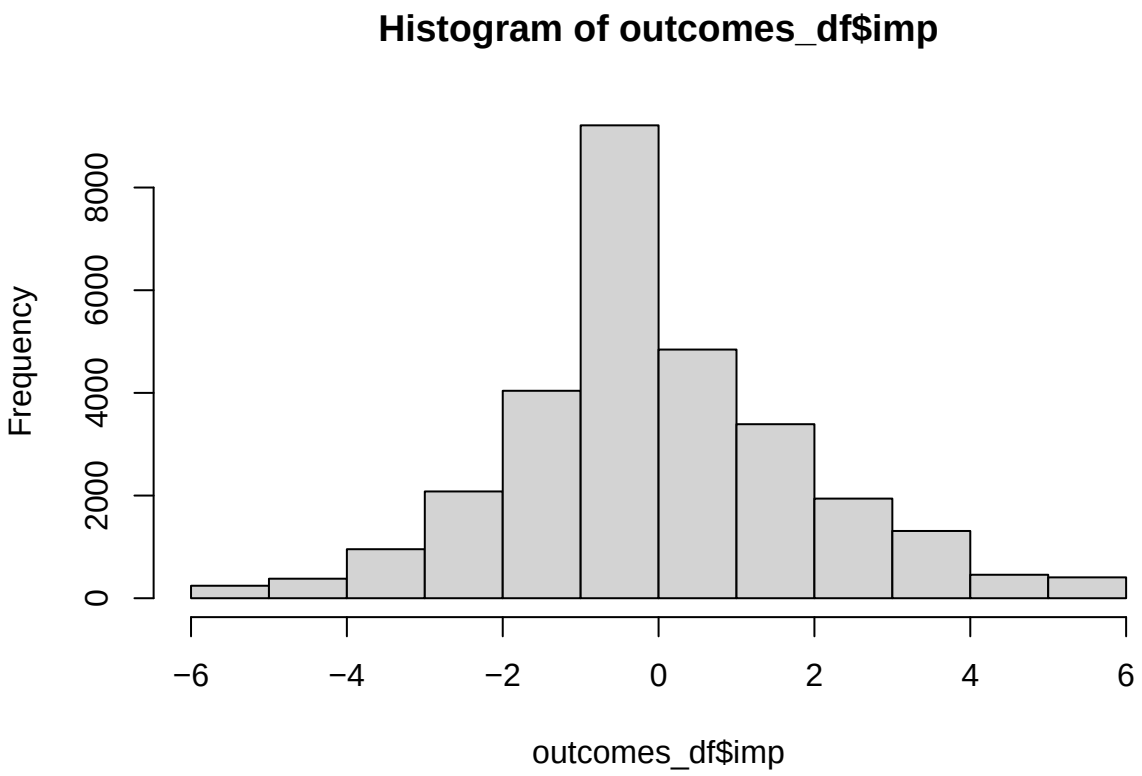
```r
# Find patients that start treatment with 7 and end with 6
filter_list <- nfdata %>%
  filter(itemId == 11, session_rank == 1, rating == 7) %>%
  select(patN, itemId, session_rank, rating)

filter_list <- nfdata %>% filter(itemId == 11, patN %in% filter_list$patN) %>%
  group_by(patN) %>%
  filter(session_rank == max(session_rank), rating == 6)

# Number of patients to filter
length(unique(filter_list$patN))
```

```
## [1] 76
```

```r
# Set the improvement for the filtered patients to 0
outcomes_df$imp[outcomes_df$patN %in% filter_list$patN] <- 0
```

Lets take a look at the distribution of improvement on the outcome item and calculate some summary statistics.

```r
# Plot distribution of outcomes
hist(outcomes_df$imp)
```




```
# Summary statistics of outcomes
summary(outcomes_df$imp)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
## -6.0000 -1.0000  0.0000  0.4618  2.0000  6.0000    1253
```

It's time to convert the improvement to a binary outcome and update the dataset.

```
# Binary outcome
outcomes <- outcomes_df %>%
  mutate(outcome = ifelse(imp > 0, 1, 0)) %>%
  select(patN, outcome) %>%
  distinct(patN, .keep_all = TRUE)

# Create final data set
nfdata <- nfdata %>%
  left_join(outcomes, by = "patN") %>%
  filter(!is.na(outcome))

glimpse(nfdata %>% distinct(patN, .keep_all = TRUE) %>% select(patN, outcome))
```

```
## Rows: 3,328
## Columns: 2
## $ patN    <int> 1361, 2535, 1609, 956, 23, 991, 1986, 1911, 3116, 1534, 2037, ~
## $ outcome <dbl> 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,~
```

After removing patients with missing outcome variable I have 3328 patients in the dataset. Of these the proportion that I have labeled as no treatment effect is 0.5799722.

Finally I will update the patient dataframe to only include patients that are also in the dataset.

```
# Update patient_df
patient_df <- patient_df %>% filter(patN %in% nfdata$patN)
```

3.2 Transforming the dataset

The default data format in the R tidyverse is "tidy" data in a long format, meaning that each observation is on a separate row with the variables in the columns. To transform this dataset into a tidy format each session for each patient should be on a row and the items should be in columns. However, since this is multivariate time series data the observations are not independent. To train models both across items and across sessions I will transform the dataset into a wide dataset where each patient is on the rows and all observations for that patient on the different variables over time is in the columns. Only the three first sessions will be included as I am interested in predicting outcome after three sessions.

```
# I want to use the first 3 sessions to train and predict outcome
nfdata_w <- nfdata %>% group_by(patN) %>%
  filter(session_rank <=3) %>%
  select(-session, -n_sessions) %>%
  ungroup()
```

Since this is time series data some variables like age can change over time. Let's check if there are any patients that had a birthday during the first three sessions, and if so set the age to the age at start of treatment.

```
# Some patients got a year older during treatment
age_changed <- nfdata_w %>%
  group_by(patN, age) %>%
  summarize(n = n()) %>%
  ungroup() %>%
  filter(patN %in% unique(.[["patN"]][duplicated(.[["patN"]])]))

head(age_changed, 20)
```

patN	age	n
10	48	16
10	49	36
21	31	28
21	32	14
24	47	39
24	48	22
25	45	25
25	46	50
31	56	27
31	57	34
43	28	45
43	29	18
48	41	41

patN	age	n
48	42	18
59	44	27
59	45	50
68	43	27
68	44	54
72	54	27
72	55	54

```
# Set patient age to age at first session
nfdata_w <- nfdata_w %>%
  group_by(patN) %>%
  mutate(age = min(age)) %>%
  ungroup()
```

Transform the data to a wide format

```
# Create wide dataframe
nfdata_w <- pivot_wider(nfdata_w,
                        names_from = c(itemId, session_rank),
                        values_from = rating,
                        values_fn = list(rating = mean))

# One-hot encoding
dmy <- dummyVars("~.", data = nfdata_w)
nfdata_w <- data.frame(predict(dmy, newdata = nfdata_w))

glimpse(nfdata_w)
```

```
## Rows: 3,328
## Columns: 93
## $ patN    <dbl> 1361, 2535, 1609, 956, 23, 991, 1986, 1911, 3116, 1534, 2037, ~
## $ age     <dbl> 19, 33, 22, 17, 35, 33, 25, 23, 42, 31, 59, 29, 43, 27, 33, 27~
## $ sex.F   <dbl> 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, NA, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1~
## $ sex.M   <dbl> 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, NA, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0~
## $ sex.O   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, NA, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ outcome <dbl> 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,~
## $ X.27_1. <dbl> 2, 5, 4, 1, 1, NA, 1, 4, 2, 1, 4, 1, 7, 6, 1, 1, 2, 5, 1, 5, 1~
## $ X.25_1. <dbl> 6, NA, 4, 5, 6, 7, NA, 7, 7, NA, NA, 7, 4, NA, 5, 7, NA, 7, NA~
## $ X.24_1. <dbl> 3, 7, 5, 5, 5, 3, 5, NA, NA, 1, 1, 1, 7, 3, 5, 1, 4, 6, 4, 5, ~
## $ X.20_1. <dbl> 2, 4, 4, NA, NA, NA, 2, 4, NA, 1, 1, 5, 7, 2, 1, 1, 4, 1, 1, 3~
## $ X.17_1. <dbl> 4, 6, 3, 7, 1, 5, 7, 4, 4, 2, 3, 5, 4, 6, 7, 4, 4, 5, 5, 5, 1,~
## $ X.5_1.  <dbl> 3, 4, 2, 5, 2, 5, 4, NA, NA, 4, 1, 1, 7, 4, 5, 2, 1, 5, 7, 6, ~
## $ X.15_1. <dbl> 1, 1, 6, 1, NA, 1, 2, 2, NA, 1, 1, 2, 2, 1, 1, 1, 1, 1, 2, 2, ~
## $ X.16_1. <dbl> 4, 1, 5, 5, NA, NA, NA, 6, 5, NA, NA, 7, 5, NA, 6, NA, 5, NA, ~
## $ X.13_1. <dbl> 1, 1, 1, 3, 1, 1, 5, 1, 1, 1, 1, 1, 2, 1, 5, 1, 1, 1, 1, 2, 1,~
## $ X.28_1. <dbl> 2, 3, 3, 5, 1, 2, 6, 4, 2, 1, 1, 1, 4, 4, 7, 1, 6, 5, 3, 4, 1,~
## $ X.22_1. <dbl> 6, NA, 5, 7, 5, 7, NA, 7, 7, NA, NA, 7, 5, NA, 5, 7, NA, 7, NA~
## $ X.23_1. <dbl> 0, NA, 0, 0, 0, 0, NA, 0, 0, NA, NA, -5, 0, NA, 0, 0, NA, NA, ~
## $ X.3_1.  <dbl> 6, NA, 3, 7, 6, 7, NA, 7, 7, NA, NA, 7, 5, NA, 6, 7, NA, 7, NA~
## $ X.29_1. <dbl> 4, 7, 4, 6, 4, 4, 6, 5, 3, 1, 1, 5, 6, 4, 5, 1, 4, 6, 4, 5, 1,~
## $ X.26_1. <dbl> 1, 5, 1, 3, 2, 1, 4, 3, 1, 1, 1, 1, 4, 4, 3, 1, 4, 1, 2, 4, 1,~
## $ X.21_1. <dbl> 0, NA, -3, 0, 0, 0, NA, 0, 0, NA, NA, 0, 0, NA, 0, 0, NA, NA, ~
```

```
## $ X.19_1. <dbl> 0, NA, 0, 0, 0, 0, NA, 0, 0, NA, NA, 0, 0, NA, NA, 0, NA, NA, ~
## $ X.2_1.  <dbl> 0, NA, 1, 0, 0, 0, NA, 0, 0, NA, NA, 0, 0, NA, 1, 0, NA, NA, N~
## $ X.10_1. <dbl> 1, 1, 1, NA, NA, 1, 1, NA, NA, 1, 1, 1, NA, 1, NA, 1, 1, NA, 1~
## $ X.1_1.  <dbl> 1, 5, 2, NA, NA, NA, 1, 7, NA, 1, 1, 1, 7, 4, 1, 1, 1, 1, 1, 6~
## $ X.8_1.  <dbl> 1, 1, 3, NA, NA, NA, 1, 3, NA, 1, 1, 1, 7, 2, 1, 1, 1, NA, 1, ~
## $ X.12_1. <dbl> 3, 5, 1, NA, NA, 1, 1, NA, NA, 1, 1, 1, NA, 1, NA, 1, 1, NA, 2~
## $ X.7_1.  <dbl> 3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4, 1, 1,~
## $ X.4_1.  <dbl> 3, 6, 1, 2, 3, 7, 6, NA, NA, 1, 1, 1, 7, 2, 5, 1, 1, 5, 1, 6, ~
## $ X.14_1. <dbl> 2, 7, 1, 2, NA, NA, NA, 1, 1, NA, NA, 7, 2, NA, 2, 2, 1, NA, 2~
## $ X.18_1. <dbl> 6, NA, 5, 7, 6, 7, NA, 7, 7, NA, NA, 6, 6, NA, NA, 7, NA, 7, N~
## $ X.6_1.  <dbl> 3, 7, 5, 2, 4, 4, 6, 4, 1, NA, 1, 5, 7, 7, 5, 7, 2, 5, 2, 5, 1~
## $ X.11_1. <dbl> 3, 6, 3, 5, 4, 7, 5, 4, 5, 3, 1, 2, 5, 6, 5, 2, 3, 5, 4, 7, 1,~
## $ X.9_1.  <dbl> 5, 7, 4, 5, 4, 1, 5, 1, 5, NA, 1, 2, 7, 6, 1, 1, 1, 5, 3, 2, 2~
## $ X.2_2.  <dbl> 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, NA, 3, 0, 0~
## $ X.19_2. <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, NA, 0, 0, 0~
## $ X.23_2. <dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, NA, 0, 0, ~
## $ X.21_2. <dbl> 0, 0, -3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, NA, 0, 0, ~
## $ X.11_2. <dbl> 3, 5, 5, 3, 4, 7, 5, 3, 3, 4, 1, 4, 5, 5, 6, 2, 3, 4, 5, 7, 1,~
## $ X.3_2.  <dbl> 7, 7, 6, 7, 6, 7, 6, 7, 7, 7, 7, 7, 6, 5, 7, 7, 5, 7, 6, 7, 7,~
## $ X.7_2.  <dbl> 6, 3, 1, 1, 1, 1, 1, 1, 1, 7, 1, 1, 2, 1, 1, 1, 1, 1, 7, 2, 1,~
## $ X.25_2. <dbl> 7, 6, 5, 5, 6, 4, 6, 5, 6, 7, 7, 6, 5, 4, 5, 6, 4, 7, 4, 6, 7,~
## $ X.12_2. <dbl> 7, 5, NA, NA, NA, NA, NA, NA, NA, 1, NA, NA, NA, NA, NA, NA, N~
## $ X.14_2. <dbl> NA, 2, 1, 2, NA, NA, NA, 1, 1, NA, 2, 7, NA, NA, 3, 2, 1, NA, ~
## $ X.13_2. <dbl> 2, 1, 1, 2, 1, 4, 3, 1, 1, 1, 1, 3, 3, 2, 5, 1, 1, 1, 1, 1, 1,~
## $ X.24_2. <dbl> 4, 6, 6, 6, 3, 4, 4, 2, 1, 1, NA, NA, 6, 4, 5, NA, NA, 6, 5, 6~
## $ X.18_2. <dbl> 7, 7, 5, 7, 6, 7, 6, 7, 7, 7, 7, 6, 6, 4, 6, 7, 5, 6, 6, 7, 7,~
## $ X.17_2. <dbl> 5, 3, 4, 6, 1, 6, 4, 5, 5, 1, 3, 7, 4, 6, 6, 3, 4, 4, 5, 4, 1,~
## $ X.16_2. <dbl> 5, 7, 6, 6, NA, NA, NA, 7, 5, NA, 6, 3, 5, NA, 6, 5, 6, NA, 3,~
## $ X.29_2. <dbl> 7, 5, 3, 5, 3, 7, 3, 7, 3, 1, 2, 5, 6, 5, 4, 4, 4, 5, 5, 3, 1,~
## $ X.22_2. <dbl> 6, 7, 6, 5, 6, 7, 6, 7, 7, 7, 7, 7, 5, 7, 6, 7, 5, 7, 6, 7, 7,~
## $ X.27_2. <dbl> 4, 4, 3, 1, 2, 1, 2, 4, 3, 1, 1, 1, 7, 6, 2, 5, 3, NA, 1, 5, 1~
## $ X.29_3. <dbl> 3, 2, 4, 5, 4, 4, 3, 7, 2, 5, 3, 6, 7, 6, 5, 3, 4, 5, 4, 3, 1,~
## $ X.4_3.  <dbl> 4, 2, 1, 3, 3, 5, 4, NA, NA, NA, NA, NA, 7, 2, 5, NA, 1, 4, 4,~
## $ X.15_3. <dbl> 1, 1, 2, 1, NA, 1, 1, 1, 2, NA, NA, 6, 2, 1, 1, NA, 1, 1, 1, 1~
## $ X.25_3. <dbl> 7, 7, 5, 7, 6, 7, 7, 7, 6, 7, 7, 5, 5, 4, 4, 7, 5, 7, 4, 7, 5,~
## $ X.11_3. <dbl> 3, 1, 4, 4, 4, 7, 5, 4, 5, 4, 2, 2, 7, 7, 5, 3, 4, 3, 3, 6, 2,~
## $ X.22_3. <dbl> 7, 7, 6, 7, 7, 7, 7, 7, 6, 7, 7, 7, 6, 7, 7, 7, 5, 7, 5, 7, 7,~
## $ X.13_3. <dbl> 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 5, 1, 1, 1, 2, 1, 1,~
## $ X.28_3. <dbl> 1, 1, 6, 4, 1, 1, 2, 1, NA, NA, NA, 1, 4, 4, 6, NA, 1, 4, 3, 2~
## $ X.14_3. <dbl> NA, 1, 1, 4, NA, NA, 1, 1, 1, NA, 1, 7, 4, NA, 4, 1, 1, NA, 7,~
## $ X.18_3. <dbl> 7, 7, 6, 7, 6, 7, 4, 7, 7, 7, 7, 7, 6, 4, 7, 7, 6, 7, 5, 7, 6,~
## $ X.20_3. <dbl> 4, 5, 2, 1, 1, NA, 2, 4, 2, NA, NA, NA, 7, 3, 3, NA, NA, 3, 3,~
## $ X.7_3.  <dbl> 4, 1, 1, 1, 1, 1, 1, 1, 1, 5, 1, 1, 1, 1, 2, 1, 1, NA, 5, 2, 1~
## $ X.6_3.  <dbl> 3, 1, 3, 3, 4, 4, 3, 4, 4, NA, NA, 4, 7, 6, 4, NA, 2, 4, 3, 4,~
## $ X.24_3. <dbl> 3, 4, 6, 4, 3, 1, 3, NA, 2, 4, NA, NA, 7, 3, 5, NA, 4, 6, 4, 5~
## $ X.26_3. <dbl> 2, 1, 1, 3, 1, 1, 1, 1, NA, NA, NA, 1, 3, 4, 3, NA, 3, 1, 1, 2~
## $ X.10_3. <dbl> 2, 1, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 1, NA, NA, NA, N~
## $ X.1_3.  <dbl> 2, 6, 1, 1, 1, NA, 1, 7, NA, NA, NA, NA, 7, 4, 1, NA, NA, NA, ~
## $ X.8_3.  <dbl> 3, 1, 1, 1, 1, NA, 2, 1, 1, NA, NA, NA, 7, 2, 1, NA, NA, 1, 1,~
## $ X.9_3.  <dbl> 2, 2, 2, 6, 4, 1, 2, 7, NA, NA, 1, 4, 7, 4, 1, NA, 3, 6, 5, 4,~
## $ X.5_3.  <dbl> 7, 2, 4, 6, 6, 7, 5, NA, NA, NA, NA, NA, 7, 4, 5, NA, 4, 4, 5,~
## $ X.17_3. <dbl> 2, 1, 3, 5, 3, 4, 3, 5, 4, 4, 4, 3, 6, 6, 7, 2, 4, 2, 3, 4, 2,~
## $ X.16_3. <dbl> NA, 7, 5, 5, NA, NA, 4, 7, 4, NA, 6, 7, 5, NA, 6, NA, 5, NA, 4~
## $ X.19_3. <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, NA, 0, 0, 0, 0, NA, 0, 0, ~
## $ X.23_3. <dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, -5, NA, 0, 0, 0, 0, NA, 0, 0,~
```

```
## $ X.21_3. <dbl> 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, NA, 0, 0, 0, 1, NA, 0, 0,~
## $ X.2_3.  <dbl> 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, NA, 0, 0, 0, 5, NA, 1, 0, ~
## $ X.27_3. <dbl> 3, 1, 2, 1, 3, 1, 2, 4, 3, 1, 1, 1, 7, 6, 2, 1, NA, 3, 2, 5, 1~
## $ X.3_3.  <dbl> 7, 7, 6, 7, 6, 7, 4, 7, 7, 7, 7, 7, 6, 6, 7, 7, 6, 7, 7, 7, 7,~
## $ X.12_3. <dbl> 4, 1, NA, NA, NA, NA, NA, NA, NA, 1, NA, NA, 1, NA, NA, NA, NA~
## $ X.1_2.  <dbl> NA, 6, 1, 1, 1, NA, NA, 2, 1, NA, NA, NA, 7, 2, NA, 1, 1, 1, N~
## $ X.20_2. <dbl> NA, 5, 3, 1, 1, NA, NA, 2, 2, NA, NA, NA, 7, 2, NA, 2, 4, NA, ~
## $ X.28_2. <dbl> NA, 1, 2, 3, 1, 4, 6, 2, 2, NA, NA, 1, 5, 5, 6, 1, 1, 4, 3, 2,~
## $ X.26_2. <dbl> NA, 1, 1, 2, 2, 1, 2, 2, 3, 1, NA, 2, 2, 3, 4, 1, 3, 1, 1, 4, ~
## $ X.10_2. <dbl> NA, 1, NA, NA, NA, NA, NA, NA, NA, 1, NA, NA, NA, NA, NA, NA, ~
## $ X.8_2.  <dbl> NA, 1, 2, 1, 1, NA, NA, 1, 1, NA, NA, NA, 5, 1, NA, 1, 1, 2, N~
## $ X.15_2. <dbl> NA, 1, 2, 1, NA, 1, 2, 3, NA, NA, NA, 5, 2, 1, 2, NA, 1, 1, 1,~
## $ X.4_2.  <dbl> NA, 5, 1, 2, 5, 7, 5, 3, 2, 1, NA, NA, 7, 4, 5, NA, NA, 5, 3, ~
## $ X.9_2.  <dbl> NA, 5, 2, 6, 4, 1, 3, 6, 5, NA, NA, 2, 5, 1, 1, 1, NA, 4, 7, 5~
## $ X.6_2.  <dbl> NA, 1, 4, 2, 4, 4, 5, 5, 6, 4, NA, 7, 5, 6, 5, 7, 1, 5, 5, 4, ~
## $ X.5_2.  <dbl> NA, 6, 1, 5, 5, 7, 6, 1, 3, 1, NA, NA, 7, 4, 5, NA, NA, 5, 5, ~
```

3.3 Missing Values

Although I have removed patients where outcome labeling resulted in NAs, there still is a possibility that item 11 was missing completely from the first session. This will become evident now that NAs have been introduced by creating the wide dataset. I will again use the `lares` library to inspect the dataset for missing values and remove entries where item 11 was missing from the first session.

```
# lares library - exploring and reducing missing data
missingness(nfdata_w)
```

variable	missing	missingness
X.10_3.	2496	75.00
X.12_3.	2474	74.34
X.10_2.	2355	70.76
X.12_2.	2307	69.32
X.14_3.	1586	47.66
X.14_2.	1582	47.54
X.14_1.	1537	46.18
X.16_2.	1503	45.16
X.16_3.	1485	44.62
X.16_1.	1484	44.59
X.8_3.	1242	37.32
X.1_3.	1232	37.02
X.8_2.	1222	36.72
X.1_2.	1206	36.24
X.20_3.	1194	35.88
X.10_1.	1172	35.22
X.20_2.	1152	34.62
X.19_1.	1113	33.44
X.12_1.	1080	32.45
X.21_1.	1063	31.94
X.23_1.	1056	31.73
X.2_1.	1047	31.46
X.28_3.	1022	30.71

variable	missing	missingness
X.3_1.	1005	30.20
X.18_1.	996	29.93
X.28_2.	975	29.30
X.25_1.	938	28.19
X.22_1.	897	26.95
X.4_3.	848	25.48
X.5_3.	848	25.48
X.5_2.	842	25.30
X.4_2.	835	25.09
X.9_3.	770	23.14
X.15_3.	763	22.93
X.9_2.	733	22.03
X.15_2.	710	21.33
X.24_3.	705	21.18
X.24_2.	655	19.68
X.8_1.	603	18.12
X.1_1.	582	17.49
X.26_3.	568	17.07
X.28_1.	560	16.83
X.26_2.	540	16.23
X.5_1.	528	15.87
X.4_1.	512	15.38
X.20_1.	507	15.23
X.6_3.	502	15.08
X.6_2.	489	14.69
X.9_1.	434	13.04
X.15_1.	402	12.08
X.26_1.	373	11.21
X.19_2.	365	10.97
X.19_3.	318	9.56
X.24_1.	304	9.13
X.6_1.	303	9.10
X.21_2.	284	8.53
X.23_2.	273	8.20
sex.F	253	7.60
sex.M	253	7.60
sex.O	253	7.60
X.21_3.	246	7.39
X.2_2.	244	7.33
X.23_3.	240	7.21
X.2_3.	233	7.00
X.7_3.	186	5.59
X.18_2.	178	5.35
X.3_2.	165	4.96
X.7_2.	160	4.81
X.18_3.	116	3.49
X.3_3.	115	3.46
X.29_3.	109	3.28
X.7_1.	108	3.25
X.27_3.	99	2.97
X.29_2.	97	2.91
X.25_2.	93	2.79
X.17_3.	92	2.76

variable	missing	missingness
X.27_2.	90	2.70
X.17_2.	83	2.49
X.25_3.	76	2.28
X.22_2.	72	2.16
X.11_3.	63	1.89
X.13_3.	62	1.86
X.29_1.	60	1.80
X.11_2.	58	1.74
X.22_3.	58	1.74
X.17_1.	55	1.65
X.27_1.	52	1.56
X.13_2.	49	1.47
X.13_1.	41	1.23
X.11_1.	11	0.33

```
# Removing patients with missing report on item 11 in the first session
nfdata_w <- nfdata_w %>%
  filter(!is.na(X.11_1.))

missingness(nfdata_w)
```

variable	missing	missingness
X.10_3.	2489	75.04
X.12_3.	2468	74.40
X.10_2.	2349	70.82
X.12_2.	2303	69.43
X.14_3.	1576	47.51
X.14_2.	1574	47.45
X.14_1.	1528	46.07
X.16_2.	1495	45.07
X.16_3.	1476	44.50
X.16_1.	1475	44.47
X.8_3.	1237	37.29
X.1_3.	1227	36.99
X.8_2.	1217	36.69
X.1_2.	1201	36.21
X.20_3.	1189	35.85
X.10_1.	1166	35.15
X.20_2.	1149	34.64
X.19_1.	1109	33.43
X.12_1.	1074	32.38
X.21_1.	1059	31.93
X.23_1.	1052	31.72
X.2_1.	1043	31.44
X.28_3.	1018	30.69
X.3_1.	1001	30.18
X.18_1.	993	29.94
X.28_2.	972	29.30
X.25_1.	934	28.16
X.22_1.	896	27.01

variable	missing	missingness
X.4_3.	844	25.44
X.5_3.	842	25.38
X.5_2.	837	25.23
X.4_2.	830	25.02
X.9_3.	767	23.12
X.15_3.	760	22.91
X.9_2.	730	22.01
X.15_2.	707	21.31
X.24_3.	701	21.13
X.24_2.	651	19.63
X.8_1.	596	17.97
X.1_1.	575	17.33
X.26_3.	565	17.03
X.28_1.	554	16.70
X.26_2.	536	16.16
X.5_1.	520	15.68
X.4_1.	505	15.22
X.20_1.	500	15.07
X.6_3.	499	15.04
X.6_2.	487	14.68
X.9_1.	428	12.90
X.15_1.	395	11.91
X.26_1.	366	11.03
X.19_2.	363	10.94
X.19_3.	317	9.56
X.24_1.	297	8.95
X.6_1.	296	8.92
X.21_2.	283	8.53
X.23_2.	272	8.20
sex.F	253	7.63
sex.M	253	7.63
sex.O	253	7.63
X.21_3.	245	7.39
X.2_2.	243	7.33
X.23_3.	239	7.21
X.2_3.	232	6.99
X.7_3.	185	5.58
X.18_2.	178	5.37
X.3_2.	165	4.97
X.7_2.	160	4.82
X.18_3.	116	3.50
X.3_3.	115	3.47
X.29_3.	108	3.26
X.7_1.	102	3.08
X.27_3.	98	2.95
X.29_2.	97	2.92
X.25_2.	93	2.80
X.17_3.	91	2.74
X.27_2.	89	2.68
X.17_2.	83	2.50
X.25_3.	76	2.29
X.22_2.	72	2.17
X.11_3.	62	1.87

variable	missing	missingness
X.13_3.	61	1.84
X.11_2.	58	1.75
X.22_3.	58	1.75
X.29_1.	54	1.63
X.17_1.	49	1.48
X.13_2.	49	1.48
X.27_1.	46	1.39
X.13_1.	35	1.06

```
length(unique(nfdata_w$patN))
```

```
## [1] 3317
```

```
# Update patient dataframe
patient_df <- patient_df %>% filter(patN %in% nfdata_w$patN)
```

Only a few patients were removed by this filtering.

As evident from the missingness table items 10 and 12 are distinguished by a high proportion of missing values. These are items specific for substance abuse that are not relevant for all patients. However, they could be highly relevant to predict treatment effect for the patients that report on this item. For that reason I will include these items in training models despite the high missingness.

Missing values imputation

As discussed in Chapter 2 it is extremely important to carefully consider how to handle missing values as wrong methodology can introduce bias and/or impact performance of the model. To summarize the three main aspects for this dataset the proportion of missing values (missingness) is varying between items and is as high as 75% for two items. Most items have less than 50% missingness. The mechanism of missing values is mostly missing by design, but for the variable "sex" MNAR missingness can not be excluded. Most imputation methods are built on the hypothesis that missing data is MAR making them unsuitable for this dataset. Additionally, the missing values pattern is arbitrary further complicating imputation.

The methodology of the Norse Feedback system that results in missing values is carefully designed to be adaptive and user-friendly. Items that are not relevant on sub-scales are "hidden" from patients until they respond over/under a given threshold and the sub-scales are opened. In this context missing values add relevant information, the item with a missing value is not very relevant to the patient, probably meaning it has less impact on their behavioral health problem. This information could be useful for the models when predicting outcomes. For this reason I will impute "0" for missing values.

```
# Replace NAs in the item rating columns with 0s
nfdata_w[, 7:93][is.na(nfdata_w[,7:93])] <- 0
pander(nfdata_w[1:11,1:25])
```

Table 3.6: Table continues below

patN	age	sex.F	sex.M	sex.O	outcome	X.27_1.	X.25_1.	X.24_1.
1361	19	1	0	0	0	2	6	3
2535	33	1	0	0	1	5	0	7
1609	22	1	0	0	0	4	4	5
956	17	1	0	0	1	1	5	5

patN	age	sex.F	sex.M	sex.O	outcome	X.27_1.	X.25_1.	X.24_1.
23	35	0	1	0	0	1	6	5
991	33	1	0	0	1	0	7	3
1986	25	1	0	0	1	1	0	5
1911	23	1	0	0	1	4	7	0
3116	42	1	0	0	0	2	7	0
1534	31	1	0	0	1	1	0	1
2037	59	NA	NA	NA	0	4	0	1

Table 3.7: Table continues below

X.20_1.	X.17_1.	X.5_1.	X.15_1.	X.16_1.	X.13_1.	X.28_1.	X.22_1.
2	4	3	1	4	1	2	6
4	6	4	1	1	1	3	0
4	3	2	6	5	1	3	5
0	7	5	1	5	3	5	7
0	1	2	0	0	1	1	5
0	5	5	1	0	1	2	7
2	7	4	2	0	5	6	0
4	4	0	2	6	1	4	7
0	4	0	0	5	1	2	7
1	2	4	1	0	1	1	0
1	3	1	1	0	1	1	0

X.23_1.	X.3_1.	X.29_1.	X.26_1.	X.21_1.	X.19_1.	X.2_1.	X.10_1.
0	6	4	1	0	0	0	1
0	0	7	5	0	0	0	1
0	3	4	1	-3	0	1	1
0	7	6	3	0	0	0	0
0	6	4	2	0	0	0	0
0	7	4	1	0	0	0	1
0	0	6	4	0	0	0	1
0	7	5	3	0	0	0	0
0	7	3	1	0	0	0	0
0	0	1	1	0	0	0	1
0	0	1	1	0	0	0	1

3.4 Feature Engineering

The three first observations for each item could be noisy. I want to add a smoothed feature that represents treatment response over time and choose to use a simple trend line from the first to the last session. This could of course also have been a regression line that would fit all three observations better, but I want to keep the features simple and intuitive. The variance for the three observations could be of importance for confidence in predictions and I add the standard deviation as a feature. Finally the model should know how each patient scores each item by seeing the value for each item in the final (third) observation. The following code chunk performs all these operations and displays the final dataset.

```r
if(!require(matrixStats)) install.packages("matrixStats", repos = "http://cran.us.r-project.org")

# Sort columns by name
nfdata_w <- nfdata_w %>% select(order(colnames(.)))

# Create temporary dataframes for each session and sort columns by name
nfdata_1 <- nfdata_w %>% select(ends_with("_1.")) %>% select(order(colnames(.)))
nfdata_2 <- nfdata_w %>% select(ends_with("_2.")) %>% select(order(colnames(.)))
nfdata_3 <- nfdata_w %>% select(ends_with("_3.")) %>% select(order(colnames(.)))

# Calculate the trend for the first to the third session
nfdata_t <- data.frame(
  sapply(seq(1:ncol(nfdata_1)),
         function(i){(nfdata_1[,i] - nfdata_3[,i])/-2}))

# Rename columns
colnames(nfdata_t) <- gsub("_1.", "_trend", colnames(nfdata_1))

# Find the standard deviation for each item over the three first sessions
nfdata_sd <- data.frame(sapply(seq(7, ncol(nfdata_w), 3),
                           function(i){rowSds(as.matrix(nfdata_w[,i:(i+2)]))}))

# Rename columns
colnames(nfdata_sd) <- gsub("_1.", "_sd", colnames(nfdata_1))

# Update the dataset and select features to use for analysis
nfdata_w <- bind_cols(nfdata_w %>%
                        select(patN, age, sex.F, sex.M, sex.O, ends_with("_3."), outcome) %>%
                        select(order(colnames(.))), nfdata_t, nfdata_sd)

glimpse(nfdata_w)
```

```
## Rows: 3,317
## Columns: 93
## $ age        <dbl> 19, 33, 22, 17, 35, 33, 25, 23, 42, 31, 59, 29, 43, 27, 33,~
## $ outcome    <dbl> 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,~
## $ patN       <dbl> 1361, 2535, 1609, 956, 23, 991, 1986, 1911, 3116, 1534, 203~
## $ sex.F      <dbl> 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, NA, 1, 0, 1, 1, 1, 1, 0, 0, 1~
## $ sex.M      <dbl> 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, NA, 0, 1, 0, 0, 0, 0, 1, 1, 0~
## $ sex.O      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, NA, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ X.1_3.     <dbl> 2, 6, 1, 1, 1, 0, 1, 7, 0, 0, 0, 0, 7, 4, 1, 0, 0, 0, 2, 4,~
## $ X.10_3.    <dbl> 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 5, 1,~
## $ X.11_3.    <dbl> 3, 1, 4, 4, 4, 7, 5, 4, 5, 4, 2, 2, 7, 7, 5, 3, 4, 3, 3, 6,~
## $ X.12_3.    <dbl> 4, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 4, 3,~
## $ X.13_3.    <dbl> 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 5, 1, 1, 1, 2, 1,~
## $ X.14_3.    <dbl> 0, 1, 1, 4, 0, 0, 1, 1, 1, 0, 1, 7, 4, 0, 4, 1, 1, 0, 7, 1,~
## $ X.15_3.    <dbl> 1, 1, 2, 1, 0, 1, 1, 1, 2, 0, 0, 6, 2, 1, 1, 0, 1, 1, 1, 1,~
## $ X.16_3.    <dbl> 0, 7, 5, 5, 0, 0, 4, 7, 4, 0, 6, 7, 5, 0, 6, 0, 5, 0, 4, 7,~
## $ X.17_3.    <dbl> 2, 1, 3, 5, 3, 4, 3, 5, 4, 4, 4, 3, 6, 6, 7, 2, 4, 2, 3, 4,~
## $ X.18_3.    <dbl> 7, 7, 6, 7, 6, 7, 4, 7, 7, 7, 7, 7, 6, 4, 7, 7, 6, 7, 5, 7,~
## $ X.19_3.    <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ X.2_3.     <dbl> 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 1, 0,~
## $ X.20_3.    <dbl> 4, 5, 2, 1, 1, 0, 2, 4, 2, 0, 0, 0, 7, 3, 3, 0, 0, 3, 3, 5,~
## $ X.21_3.    <dbl> 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0~
## $ X.22_3.    <dbl> 7, 7, 6, 7, 7, 7, 7, 7, 6, 7, 7, 7, 6, 7, 7, 7, 5, 7, 5, 7,~
```

```
## $ X.23_3.    <dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, -5, 0, 0, 0, 0, 0, 0, 0, 0~
## $ X.24_3.    <dbl> 3, 4, 6, 4, 3, 1, 3, 0, 2, 4, 0, 0, 7, 3, 5, 0, 4, 6, 4, 5,~
## $ X.25_3.    <dbl> 7, 7, 5, 7, 6, 7, 7, 7, 6, 7, 7, 5, 5, 4, 4, 7, 5, 7, 4, 7,~
## $ X.26_3.    <dbl> 2, 1, 1, 3, 1, 1, 1, 1, 0, 0, 0, 1, 3, 4, 3, 0, 3, 1, 1, 2,~
## $ X.27_3.    <dbl> 3, 1, 2, 1, 3, 1, 2, 4, 3, 1, 1, 1, 7, 6, 2, 1, 0, 3, 2, 5,~
## $ X.28_3.    <dbl> 1, 1, 6, 4, 1, 1, 2, 1, 0, 0, 0, 1, 4, 4, 6, 0, 1, 4, 3, 2,~
## $ X.29_3.    <dbl> 3, 2, 4, 5, 4, 4, 3, 7, 2, 5, 3, 6, 7, 6, 5, 3, 4, 5, 4, 3,~
## $ X.3_3.     <dbl> 7, 7, 6, 7, 6, 7, 4, 7, 7, 7, 7, 7, 6, 6, 7, 7, 6, 7, 7, 7,~
## $ X.4_3.     <dbl> 4, 2, 1, 3, 3, 5, 4, 0, 0, 0, 0, 0, 7, 2, 5, 0, 1, 4, 4, 4,~
## $ X.5_3.     <dbl> 7, 2, 4, 6, 6, 7, 5, 0, 0, 0, 0, 0, 7, 4, 5, 0, 4, 4, 5, 4,~
## $ X.6_3.     <dbl> 3, 1, 3, 3, 4, 4, 3, 4, 4, 0, 0, 4, 7, 6, 4, 0, 2, 4, 3, 4,~
## $ X.7_3.     <dbl> 4, 1, 1, 1, 1, 1, 1, 1, 1, 5, 1, 1, 1, 1, 2, 1, 1, 0, 5, 2,~
## $ X.8_3.     <dbl> 3, 1, 1, 1, 1, 0, 2, 1, 1, 0, 0, 0, 7, 2, 1, 0, 0, 1, 1, 1,~
## $ X.9_3.     <dbl> 2, 2, 2, 6, 4, 1, 2, 7, 0, 0, 1, 4, 7, 4, 1, 0, 3, 6, 5, 4,~
## $ X.1_trend  <dbl> 0.5, 0.5, -0.5, 0.5, 0.5, 0.0, 0.0, 0.0, 0.0, -0.5, -0.5, -~
## $ X.10_trend <dbl> 0.5, 0.0, -0.5, 0.0, 0.0, -0.5, -0.5, 0.0, 0.0, -0.5, -0.5,~
## $ X.11_trend <dbl> 0.0, -2.5, 0.5, -0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.5, 0.5, 0.~
## $ X.12_trend <dbl> 0.5, -2.0, -0.5, 0.0, 0.0, -0.5, -0.5, 0.0, 0.0, 0.0, -0.5,~
## $ X.13_trend <dbl> 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -2.0, 0.0, 0.0, 0.0, 0.0, 0.0~
## $ X.14_trend <dbl> -1.0, -3.0, 0.0, 1.0, 0.0, 0.0, 0.5, 0.0, 0.0, 0.0, 0.5, 0.~
## $ X.15_trend <dbl> 0.0, 0.0, -2.0, 0.0, 0.0, 0.0, -0.5, -0.5, 1.0, -0.5, -0.5,~
## $ X.16_trend <dbl> -2.0, 3.0, 0.0, 0.0, 0.0, 0.0, 2.0, 0.5, -0.5, 0.0, 3.0, 0.~
## $ X.17_trend <dbl> -1.0, -2.5, 0.0, -1.0, 1.0, -0.5, -2.0, 0.5, 0.0, 1.0, 0.5,~
## $ X.18_trend <dbl> 0.5, 3.5, 0.5, 0.0, 0.0, 0.0, 2.0, 0.0, 0.0, 3.5, 3.5, 0.5,~
## $ X.19_trend <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ X.2_trend  <dbl> 0.0, 2.5, -0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0~
## $ X.20_trend <dbl> 1.0, 0.5, -1.0, 0.5, 0.5, 0.0, 0.0, 0.0, 1.0, -0.5, -0.5, -~
## $ X.21_trend <dbl> 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,~
## $ X.22_trend <dbl> 0.5, 3.5, 0.5, 0.0, 1.0, 0.0, 3.5, 0.0, -0.5, 3.5, 3.5, 0.0~
## $ X.23_trend <dbl> 0.0, 0.0, 0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,~
## $ X.24_trend <dbl> 0.0, -1.5, 0.5, -0.5, -1.0, -1.0, -1.0, 0.0, 1.0, 1.5, -0.5~
## $ X.25_trend <dbl> 0.5, 3.5, 0.5, 1.0, 0.0, 0.0, 3.5, 0.0, -0.5, 3.5, 3.5, -1.~
## $ X.26_trend <dbl> 0.5, -2.0, 0.0, 0.0, -0.5, 0.0, -1.5, -1.0, -0.5, -0.5, -0.~
## $ X.27_trend <dbl> 0.5, -2.0, -1.0, 0.0, 1.0, 0.5, 0.5, 0.0, 0.5, 0.0, -1.5, 0~
## $ X.28_trend <dbl> -0.5, -1.0, 1.5, -0.5, 0.0, -0.5, -2.0, -1.5, -1.0, -0.5, -~
## $ X.29_trend <dbl> -0.5, -2.5, 0.0, -0.5, 0.0, 0.0, -1.5, 1.0, -0.5, 2.0, 1.0,~
## $ X.3_trend  <dbl> 0.5, 3.5, 1.5, 0.0, 0.0, 0.0, 2.0, 0.0, 0.0, 3.5, 3.5, 0.0,~
## $ X.4_trend  <dbl> 0.5, -2.0, 0.0, 0.5, 0.0, -1.0, -1.0, 0.0, 0.0, -0.5, -0.5,~
## $ X.5_trend  <dbl> 2.0, -1.0, 1.0, 0.5, 2.0, 1.0, 0.5, 0.0, 0.0, -2.0, -0.5, -~
## $ X.6_trend  <dbl> 0.0, -3.0, -1.0, 0.5, 0.0, 0.0, -1.5, 0.0, 1.5, 0.0, -0.5, ~
## $ X.7_trend  <dbl> 0.5, -1.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.0, 0.0, 0.0~
## $ X.8_trend  <dbl> 1.0, 0.0, -1.0, 0.5, 0.5, 0.0, 0.5, -1.0, 0.5, -0.5, -0.5, ~
## $ X.9_trend  <dbl> -1.5, -2.5, -1.0, 0.5, 0.0, 0.0, -1.5, 3.0, -2.5, 0.0, 0.0,~
## $ X.1_sd     <dbl> 1.0000000, 0.5773503, 0.5773503, 0.5773503, 0.5773503, 0.00~
## $ X.10_sd    <dbl> 1.0000000, 0.0000000, 0.5773503, 0.0000000, 0.0000000, 0.57~
## $ X.11_sd    <dbl> 0.0000000, 2.6457513, 1.0000000, 1.0000000, 0.0000000, 0.00~
## $ X.12_sd    <dbl> 2.0816660, 2.3094011, 0.5773503, 0.0000000, 0.0000000, 0.57~
## $ X.13_sd    <dbl> 0.5773503, 0.0000000, 0.0000000, 0.5773503, 0.0000000, 1.73~
## $ X.14_sd    <dbl> 1.1547005, 3.2145503, 0.0000000, 1.1547005, 0.0000000, 0.00~
## $ X.15_sd    <dbl> 0.5773503, 0.0000000, 2.3094011, 0.0000000, 0.0000000, 0.00~
## $ X.16_sd    <dbl> 2.6457513, 3.4641016, 0.5773503, 0.5773503, 0.0000000, 0.00~
## $ X.17_sd    <dbl> 1.5275252, 2.5166115, 0.5773503, 1.0000000, 1.1547005, 1.00~
## $ X.18_sd    <dbl> 0.5773503, 4.0414519, 0.5773503, 0.0000000, 0.0000000, 0.00~
## $ X.19_sd    <dbl> 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.0000000, 0.00~
## $ X.2_sd     <dbl> 0.0000000, 2.8867513, 0.5773503, 0.0000000, 0.0000000, 0.00~
```

```
## $ X.20_sd    <dbl> 2.0000000, 0.5773503, 1.0000000, 0.5773503, 0.5773503, 0.00~
## $ X.21_sd    <dbl> 0.0000000, 0.0000000, 1.1547005, 0.0000000, 0.0000000, 0.00~
## $ X.22_sd    <dbl> 0.5773503, 4.0414519, 0.5773503, 1.1547005, 1.0000000, 0.00~
## $ X.23_sd    <dbl> 0.0000000, 0.0000000, 0.5773503, 0.0000000, 0.0000000, 0.00~
## $ X.24_sd    <dbl> 0.5773503, 1.5275252, 0.5773503, 1.0000000, 1.1547005, 1.52~
## $ X.25_sd    <dbl> 0.5773503, 3.7859389, 0.5773503, 1.1547005, 0.0000000, 1.73~
## $ X.26_sd    <dbl> 1.0000000, 2.3094011, 0.0000000, 0.5773503, 0.5773503, 0.00~
## $ X.27_sd    <dbl> 1.0000000, 2.0816660, 1.0000000, 0.0000000, 1.0000000, 0.57~
## $ X.28_sd    <dbl> 1.0000000, 1.1547005, 2.0816660, 1.0000000, 0.0000000, 1.52~
## $ X.29_sd    <dbl> 2.0816660, 2.5166115, 0.5773503, 0.5773503, 0.5773503, 1.73~
## $ X.3_sd     <dbl> 0.5773503, 4.0414519, 1.7320508, 0.0000000, 0.0000000, 0.00~
## $ X.4_sd     <dbl> 2.0816660, 2.0816660, 0.0000000, 0.5773503, 1.1547005, 1.15~
## $ X.5_sd     <dbl> 3.5118846, 2.0000000, 1.5275252, 0.5773503, 2.0816660, 1.15~
## $ X.6_sd     <dbl> 1.7320508, 3.4641016, 1.0000000, 0.5773503, 0.0000000, 0.00~
## $ X.7_sd     <dbl> 1.5275252, 1.5275252, 0.0000000, 0.0000000, 0.0000000, 0.00~
## $ X.8_sd     <dbl> 1.5275252, 0.0000000, 1.0000000, 0.5773503, 0.5773503, 0.00~
## $ X.9_sd     <dbl> 2.5166115, 2.5166115, 1.1547005, 0.5773503, 0.0000000, 0.00~
```

3.5 Splitting Datasets

Now that I have completed the data wrangling, missing values handling and feature engineering, it is time to split the dataset into a validation set, a training set and a test set.

```
# Split into train and validation set
set.seed(31)
test_index <- createDataPartition(y = nfdata_w$outcome, times = 1, p = 0.2, list = FALSE)
nfdata_train_full <- nfdata_w[-test_index,]
nfdata_validation <- nfdata_w[test_index,]

# Split into train and test set
set.seed(41)
test_index <- createDataPartition(y = nfdata_train_full$outcome, times = 1, p = 0.2, list = FALSE)
nfdata_test <- nfdata_train_full[test_index,]
nfdata_train <- nfdata_train_full[-test_index,]

nfdata_train <- nfdata_train %>% select(order(colnames(.)))
nfdata_test <- nfdata_test %>% select(order(colnames(.)))

summary(nfdata_train$outcome)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0000  0.0000  0.4147  1.0000  1.0000
```

```
summary(nfdata_test$outcome)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0000  0.0000  0.4313  1.0000  1.0000
```

```
summary(nfdata_validation$outcome)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0000  0.0000  0.4006  1.0000  1.0000
```

```
dim(nfdata_train)
```

```
## [1] 2122   93
```

```
dim(nfdata_test)
```

```
## [1] 531  93
```

```
dim(nfdata_validation)
```

```
## [1] 664  93
```

Chapter 4

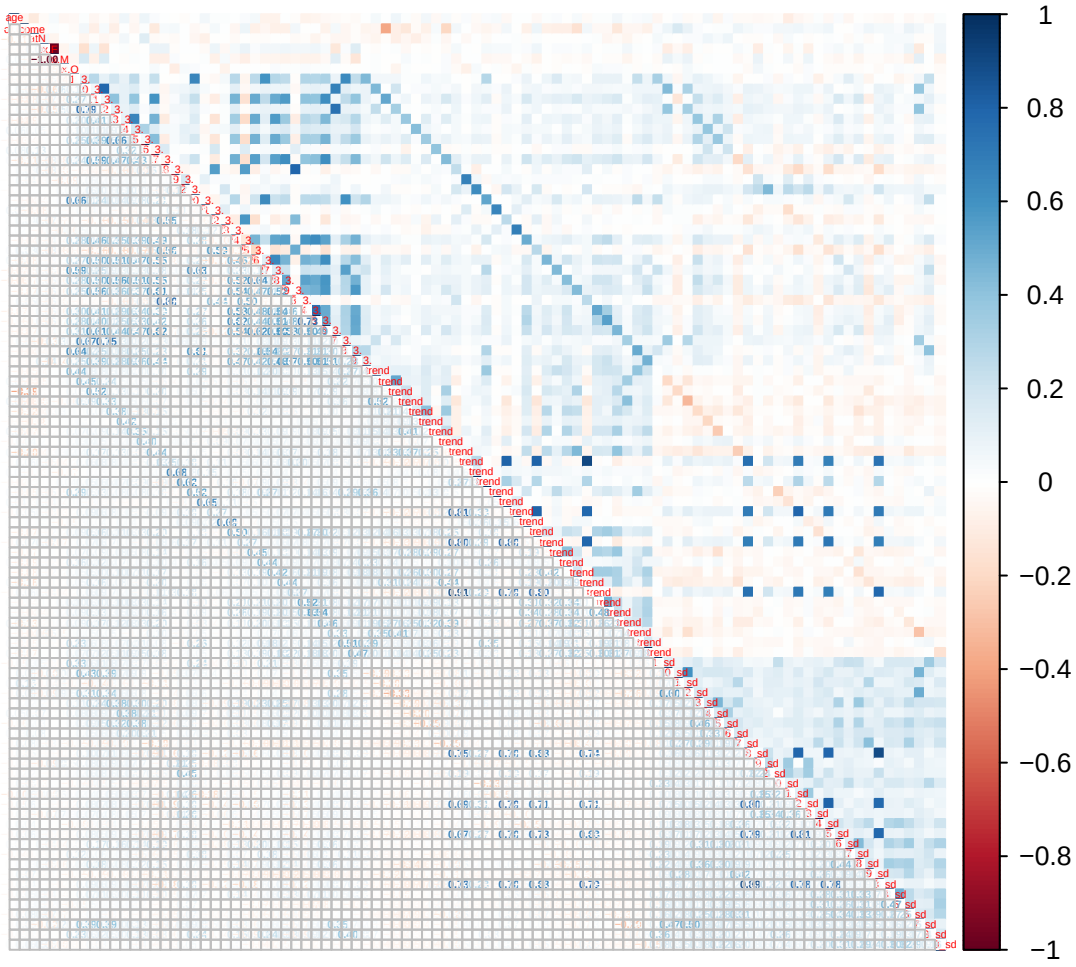
Results

In this chapter I will explore the relationship between the features in the training set and train models using linear regression and gradient boosted decision trees.

4.1 Exploring the training set

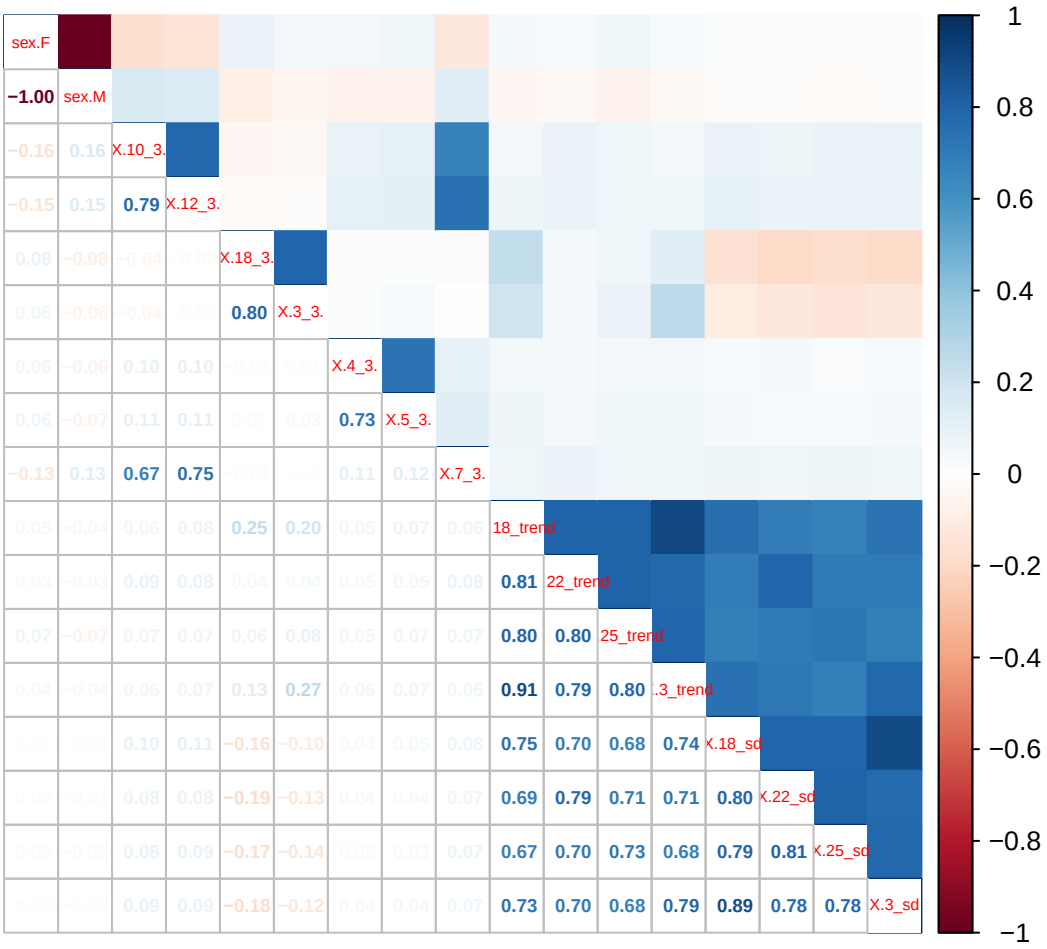
I want to explore the feature correlations in the training set. To visualize the correlations I am using functions from the `corrplot` library.

```
if(!require(corrplot)) install.packages("corrplot", repos = "http://cran.us.r-project.org")

# Correlation matrix for features
mcor <- cor(na.omit(nfdata_train_full))
corrplot.mixed(mcor, upper = "color", lower = "number", number.cex= 0.3, tl.cex = 0.3)
```



Visualizing all correlations makes it hard to identify correlated features, but the overall impression is that there are few features that have high negative correlations, but there seems to be several with high positive correlation. To get a more informative visualization I will filter correlations with an absolute value over 0.7.

```
# Filtered correlation matrix
threshold <- 0.7
mcor_1 <- mcor
diag(mcor_1) <- 0
mcor_2 <- apply(abs(mcor_1) >= threshold, 1, any)

corrplot.mixed(mcor[mcor_2, mcor_2], upper = "color", lower = "number", number.cex= 0.6, tl.cex = 0.5)
```

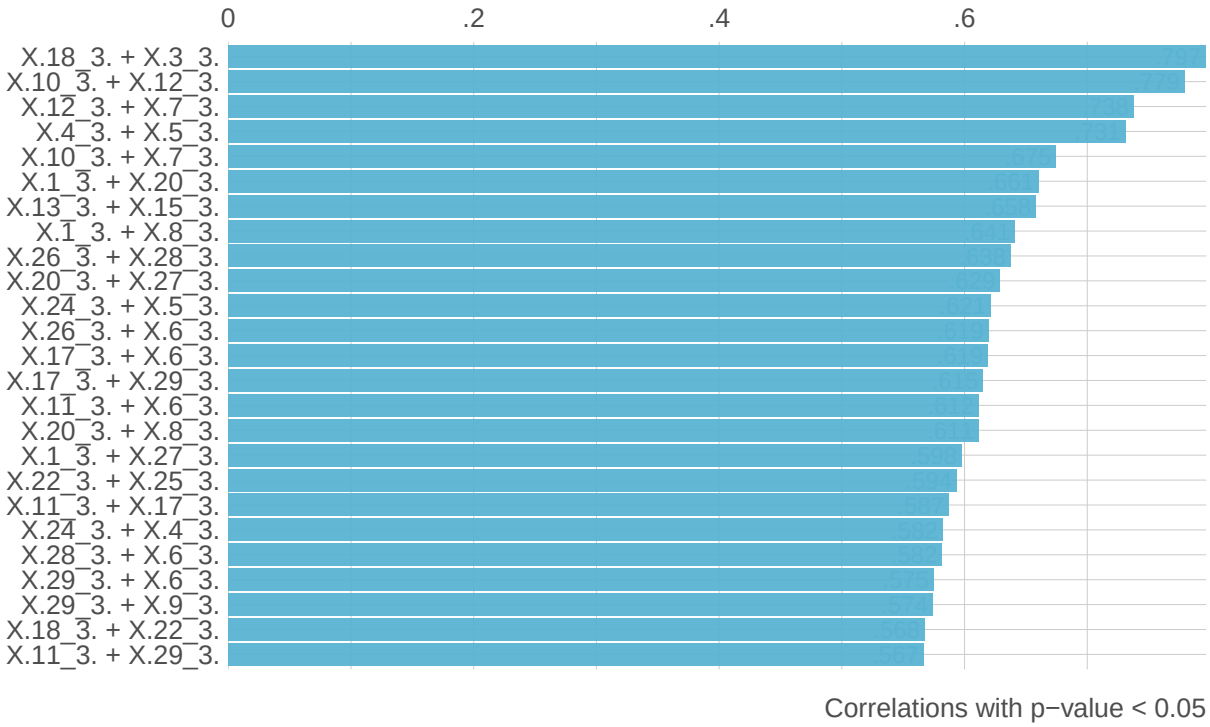


The top 20 correlations in the dataset:

```
# Exploring feature correlations
corr_cross(nfdata_train_full, max_pvalue = 0.05, top = 20)
```

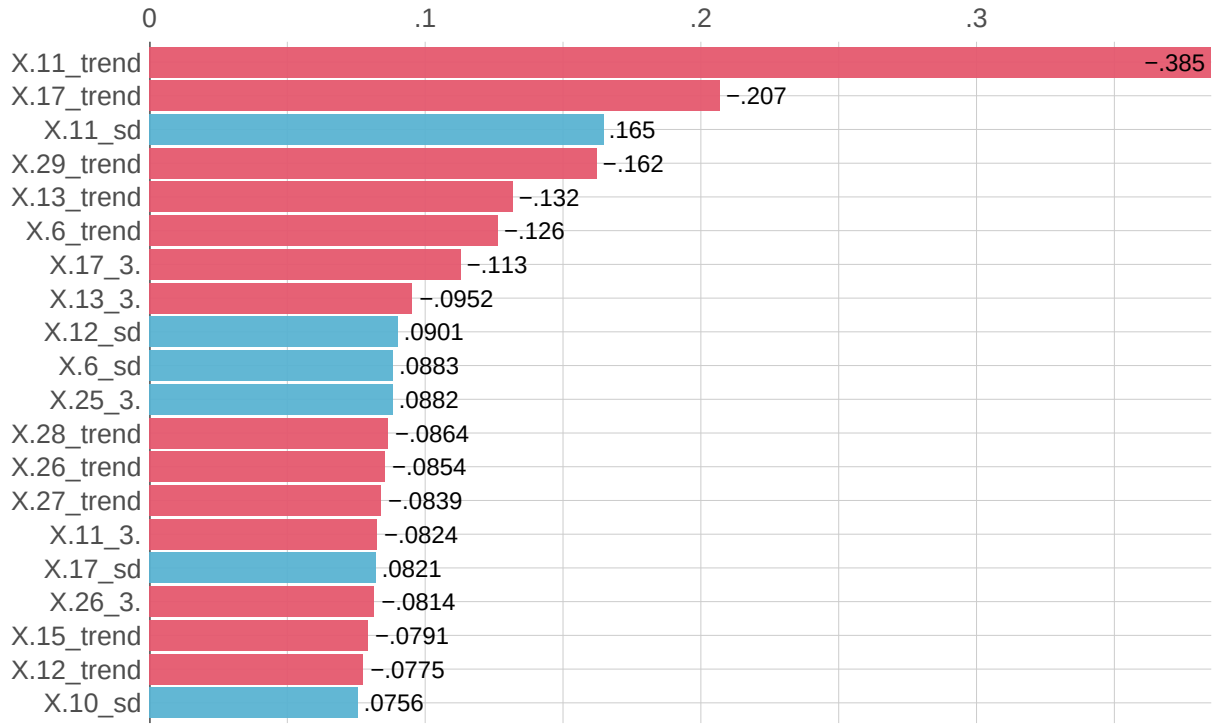
Item-item correlations for session 3:

```
nfdata_train_full %>% select(ends_with("3.")) %>% corr_cross(max_pvalue = 0.05, top = 25)
```



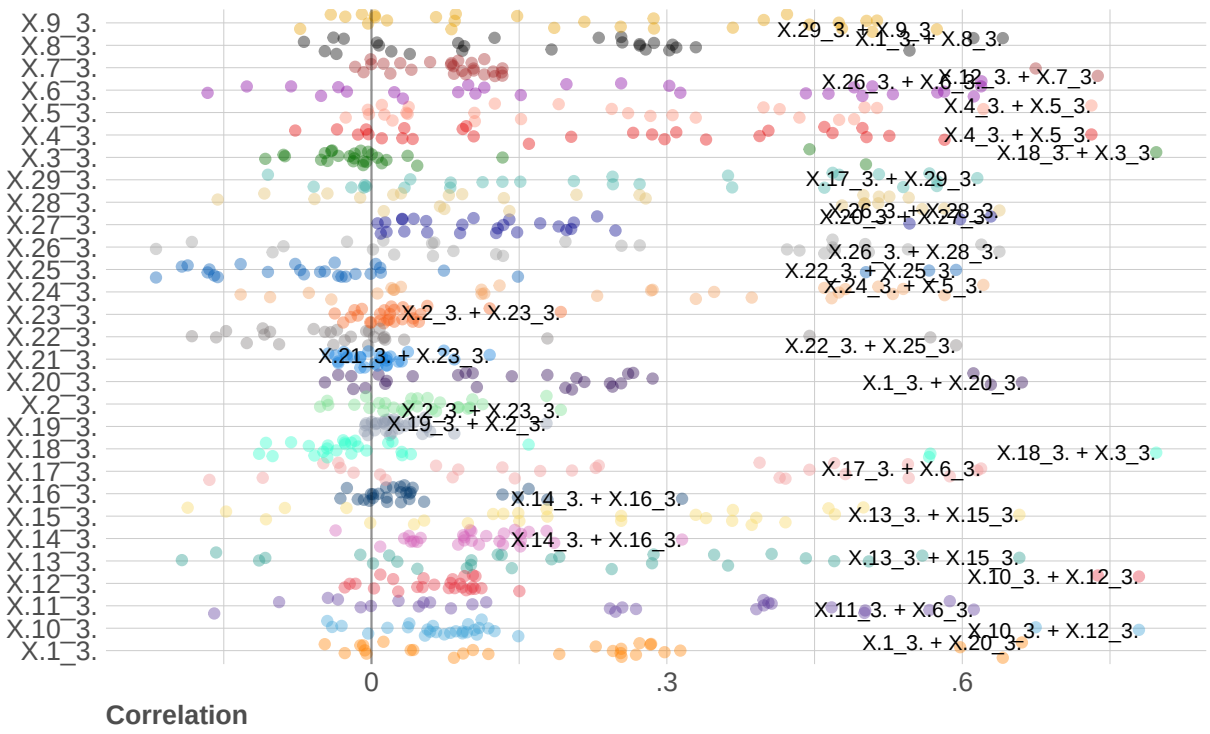
It would also be interesting to see which features are most correlated with outcome.

```
# Feature correlation with outcome
corr_var(nfdata_train_full, var = outcome, top = 20)
```



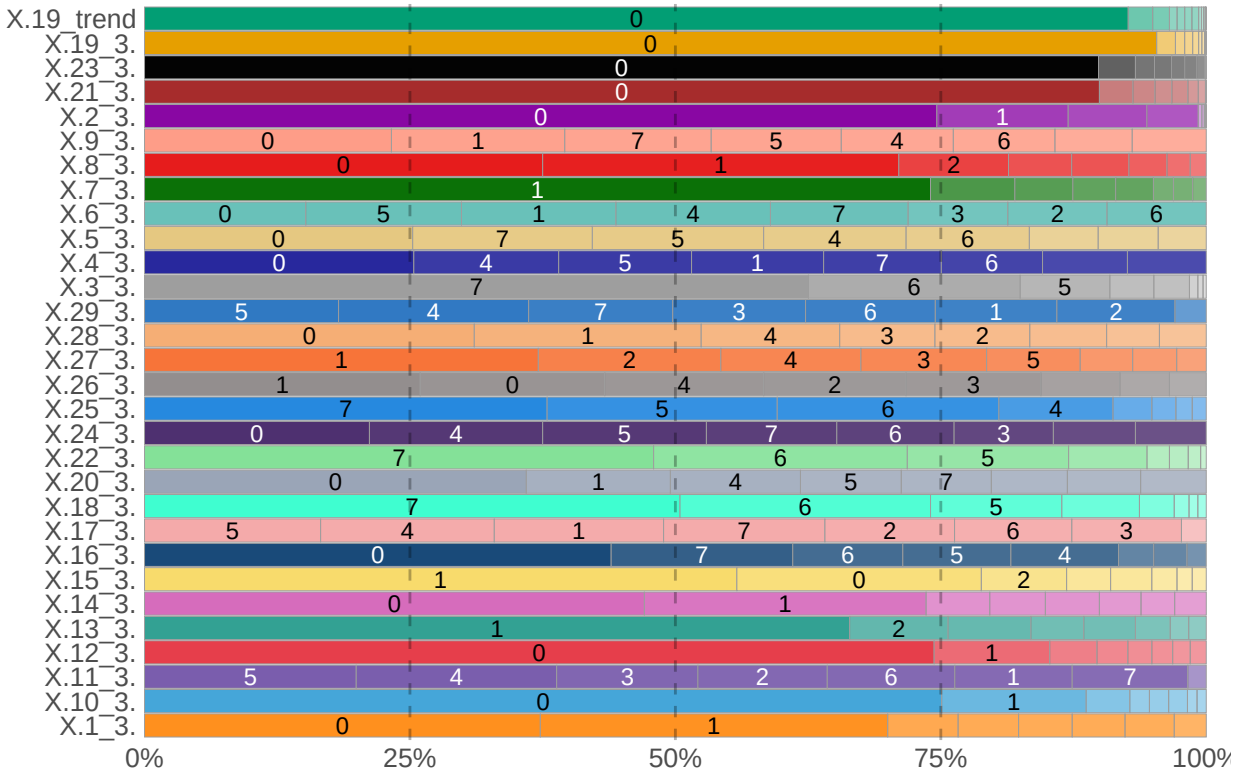
To see the variance in item-item correlation for each item we can check local correlation

```
#Local correlation for third session items
nfdata_train_full %>% select(ends_with("3.")) %>% corr_cross(type = 2)
```



And finally I will check the frequency of responses for each feature

```
# Frequency of response per item
freqs_df(nfdata_train_full[,7:92], plot = TRUE)
```



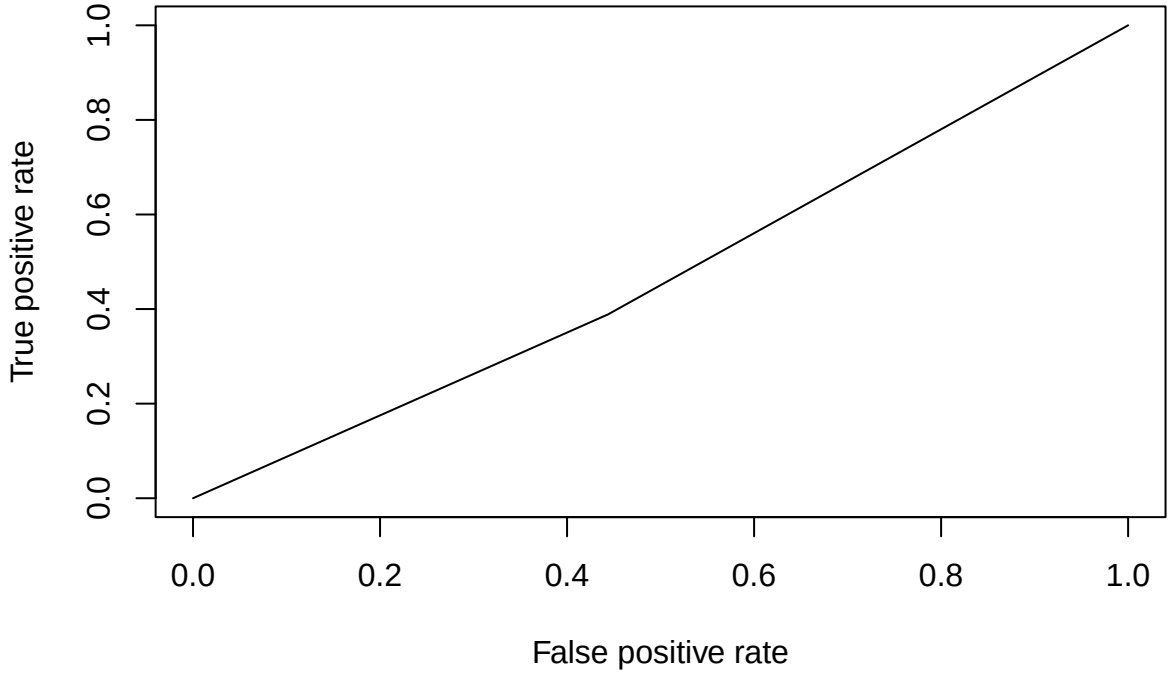
4.2 Baseline - Random Predictions

As a baseline model I create predictions based on population probabilities of outcomes.

```
if(!require(pscl)) install.packages("pscl", repos = "http://cran.us.r-project.org")
if(!require(ROCR)) install.packages("ROCR", repos = "http://cran.us.r-project.org")

# Finding the probability of a "good" outcome
prob_1 <- mean(nfdata_train$outcome == 1)

# Create random predictions based on probability
set.seed(574)
base_pred <- rbinom(n = nrow(nfdata_test), size = 1, prob = prob_1)

# Check proportion of "good" outcomes in the predictions
mean(base_pred == 1)

## [1] 0.4199623

# Create a confusion matrix
cfm <- confusionMatrix(as.factor(base_pred), as.factor(nfdata_test$outcome))
cfm

## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction   0   1
##          0 168 140
##          1 134  89
##
##                Accuracy : 0.484
##                  95% CI : (0.4407, 0.5274)
##     No Information Rate : 0.5687
##     P-Value [Acc > NIR] : 1.0000
##
##                   Kappa : -0.0552
##
##  Mcnemar's Test P-Value : 0.7626
##
##             Sensitivity : 0.5563
##             Specificity : 0.3886
##          Pos Pred Value : 0.5455
##          Neg Pred Value : 0.3991
##              Prevalence : 0.5687
##          Detection Rate : 0.3164
##    Detection Prevalence : 0.5800
##       Balanced Accuracy : 0.4725
##
##        'Positive' Class : 0
##
```

```r
# Plot ROC curve and calculate the AUC
pr <- prediction(base_pred, nfdata_test$outcome)
prf <- performance(pr, measure = "tpr", x.measure = "fpr")
plot(prf)
```




```
auc <- performance(pr, measure = "auc")
auc <- auc@y.values[[1]]
auc
```

```
## [1] 0.4724688
```

```
# Create a summary table for results
summary <- tribble( ~Model, ~Accuracy, ~"95% CI Lower", ~"95% CI Upper", ~AUC,
                    "Random", cfm$overall[[1]], cfm$overall[[3]], cfm$overall[[4]], auc)

# Print summary table
knitr::kable(summary)
```

Model	Accuracy	95% CI Lower	95% CI Upper	AUC
Random	0.4839925	0.4407364	0.5274276	0.4724688

As expected the outcome of these predictions are about the same as a coin toss. Note that the accuracy is reported as the accuracy in predicting lack of treatment outcome (“0”).

4.3 Logistic Regression

The first model I will train is a logistic regression model using the generalized linear models (glm) algorithm. For this algorithm to be trained on all data there can be no NAs in the dataset, and I choose to set the remaining NAs in the “sex” variable to 0s rather than these observations being omitted during training.

```r
# Set up datasets, remove the patient number variable.
glm_train <- nfdata_train %>% select(-patN)
glm_test <- nfdata_test %>% select(-patN)

# For glm there can be no NAs.
glm_train[is.na(glm_train)] <- 0
glm_test[is.na(glm_test)] <- 0

# Train a logistic regression model
glm_model <- glm(outcome ~., family=binomial, data = glm_train)
summary(glm_model)
```

```
##
## Call:
## glm(formula = outcome ~ ., family = binomial, data = glm_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7948  -0.8279  -0.4769   0.9170   2.7692
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.1389639  0.5309978  -5.911 3.39e-09 ***
## age         -0.0013082  0.0039358  -0.332 0.739595
## sex.F        0.2305888  0.2057147   1.121 0.262324
## sex.M        0.2750736  0.2205248   1.247 0.212266
## sex.O       -0.1424234  1.3509816  -0.105 0.916041
## X.1_3.       0.0252593  0.0486787   0.519 0.603831
## X.1_sd       0.0197650  0.0820239   0.241 0.809581
## X.1_trend    0.0976913  0.0878877   1.112 0.266333
## X.10_3.      0.2214950  0.0898035   2.466 0.013646 *
## X.10_sd     -0.1049120  0.1258762  -0.833 0.404589
## X.10_trend  -0.2273170  0.1397918  -1.626 0.103926
## X.11_3.      0.5721403  0.0537879  10.637  < 2e-16 ***
## X.11_sd      0.3132716  0.0832987   3.761 0.000169 ***
## X.11_trend  -1.7007228  0.1024796 -16.596  < 2e-16 ***
## X.12_3.     -0.0307245  0.0758007  -0.405 0.685233
## X.12_sd      0.2291625  0.1089248   2.104 0.035391 *
## X.12_trend  -0.0601779  0.1255791  -0.479 0.631794
## X.13_3.     -0.0812934  0.0633249  -1.284 0.199229
## X.13_sd     -0.0659596  0.0990082  -0.666 0.505281
## X.13_trend   0.0355567  0.1105414   0.322 0.747711
## X.14_3.      0.0035373  0.0373273   0.095 0.924502
## X.14_sd     -0.0698921  0.0579392  -1.206 0.227701
## X.14_trend   0.0121739  0.0627870   0.194 0.846260
## X.15_3.      0.0715086  0.0695419   1.028 0.303818
## X.15_sd     -0.1053626  0.1070559  -0.984 0.325025
## X.15_trend  -0.0698859  0.1165718  -0.600 0.548833
## X.16_3.      0.0178727  0.0230573   0.775 0.438256
## X.16_sd     -0.1012674  0.0537748  -1.883 0.059676 .
## X.16_trend   0.0279186  0.0579027   0.482 0.629689
## X.17_3.     -0.0816375  0.0456109  -1.790 0.073475 .
## X.17_sd     -0.0830148  0.0823026  -1.009 0.313141
## X.17_trend  -0.1395950  0.0876702  -1.592 0.111323
## X.18_3.     -0.0529924  0.0881019  -0.601 0.547514
```

```
## X.18_sd     -0.0497493  0.0992625  -0.501 0.616238
## X.18_trend   0.0007091  0.1408238   0.005 0.995983
## X.19_3.      0.0874915  0.1274725   0.686 0.492489
## X.19_sd     -0.0525919  0.1204779  -0.437 0.662454
## X.19_trend  -0.0419930  0.2037206  -0.206 0.836689
## X.2_3.      -0.0676033  0.0634644  -1.065 0.286778
## X.2_sd       0.0046487  0.0678332   0.069 0.945363
## X.2_trend    0.0059347  0.0925180   0.064 0.948854
## X.20_3.      0.0053812  0.0391487   0.137 0.890672
## X.20_sd      0.0593497  0.0624079   0.951 0.341606
## X.20_trend  -0.0828780  0.0660885  -1.254 0.209825
## X.21_3.      0.0594100  0.0721672   0.823 0.410379
## X.21_sd      0.1365386  0.0780495   1.749 0.080225 .
## X.21_trend  -0.2172619  0.1113958  -1.950 0.051133 .
## X.22_3.     -0.0465270  0.0698642  -0.666 0.505434
## X.22_sd      0.1556360  0.0919712   1.692 0.090603 .
## X.22_trend   0.0626952  0.1112403   0.564 0.573026
## X.23_3.      0.0185255  0.0869828   0.213 0.831343
## X.23_sd      0.0228964  0.0851514   0.269 0.788014
## X.23_trend   0.0347560  0.1471449   0.236 0.813275
## X.24_3.     -0.0821124  0.0379298  -2.165 0.030399 *
## X.24_sd      0.0128366  0.0679813   0.189 0.850229
## X.24_trend   0.1248425  0.0696143   1.793 0.072918 .
## X.25_3.      0.1885970  0.0627306   3.006 0.002643 **
## X.25_sd      0.1002562  0.0863039   1.162 0.245372
## X.25_trend  -0.0822933  0.1049716  -0.784 0.433065
## X.26_3.     -0.0929786  0.0516298  -1.801 0.071723 .
## X.26_sd      0.0188354  0.0787282   0.239 0.810915
## X.26_trend   0.1216767  0.0889603   1.368 0.171386
## X.27_3.     -0.0080271  0.0483268  -0.166 0.868077
## X.27_sd     -0.1510759  0.0786499  -1.921 0.054749 .
## X.27_trend  -0.1009752  0.0817949  -1.234 0.217019
## X.28_3.     -0.0941059  0.0473562  -1.987 0.046901 *
## X.28_sd     -0.1020167  0.0777784  -1.312 0.189644
## X.28_trend  -0.1067532  0.0870458  -1.226 0.220047
## X.29_3.     -0.0612528  0.0500774  -1.223 0.221268
## X.29_sd     -0.0460883  0.0810591  -0.569 0.569644
## X.29_trend   0.0154746  0.0877518   0.176 0.860023
## X.3_3.       0.0613961  0.0856757   0.717 0.473615
## X.3_sd      -0.0392858  0.0955409  -0.411 0.680931
## X.3_trend   -0.0105474  0.1371193  -0.077 0.938686
## X.4_3.      -0.0142425  0.0407253  -0.350 0.726548
## X.4_sd       0.0872166  0.0626704   1.392 0.164022
## X.4_trend    0.0892064  0.0636496   1.402 0.161057
## X.5_3.       0.0361220  0.0405810   0.890 0.373401
## X.5_sd      -0.0618546  0.0564443  -1.096 0.273143
## X.5_trend   -0.0095488  0.0619699  -0.154 0.877541
## X.6_3.      -0.0540761  0.0428272  -1.263 0.206711
## X.6_sd       0.1496230  0.0665135   2.250 0.024480 *
## X.6_trend    0.1032257  0.0728229   1.417 0.156339
## X.7_3.      -0.0064018  0.0809793  -0.079 0.936989
## X.7_sd      -0.1390796  0.1063840  -1.307 0.191099
## X.7_trend    0.1699694  0.1300076   1.307 0.191084
## X.8_3.      -0.0851435  0.0586024  -1.453 0.146251
## X.8_sd       0.1195343  0.0912157   1.310 0.190041
```

```
## X.8_trend    0.0572682  0.1002605   0.571 0.567868
## X.9_3.       0.0331901  0.0334360   0.993 0.320883
## X.9_sd       0.0387255  0.0617408   0.627 0.530510
## X.9_trend   -0.0557984  0.0651649  -0.856 0.391851
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2879.7  on 2121  degrees of freedom
## Residual deviance: 2234.6  on 2030  degrees of freedom
## AIC: 2418.6
##
## Number of Fisher Scoring iterations: 5
```

```
alias(glm_model)
```

```
## Model :
## outcome ~ age + sex.F + sex.M + sex.O + X.1_3. + X.1_sd + X.1_trend +
##     X.10_3. + X.10_sd + X.10_trend + X.11_3. + X.11_sd + X.11_trend +
##     X.12_3. + X.12_sd + X.12_trend + X.13_3. + X.13_sd + X.13_trend +
##     X.14_3. + X.14_sd + X.14_trend + X.15_3. + X.15_sd + X.15_trend +
##     X.16_3. + X.16_sd + X.16_trend + X.17_3. + X.17_sd + X.17_trend +
##     X.18_3. + X.18_sd + X.18_trend + X.19_3. + X.19_sd + X.19_trend +
##     X.2_3. + X.2_sd + X.2_trend + X.20_3. + X.20_sd + X.20_trend +
##     X.21_3. + X.21_sd + X.21_trend + X.22_3. + X.22_sd + X.22_trend +
##     X.23_3. + X.23_sd + X.23_trend + X.24_3. + X.24_sd + X.24_trend +
##     X.25_3. + X.25_sd + X.25_trend + X.26_3. + X.26_sd + X.26_trend +
##     X.27_3. + X.27_sd + X.27_trend + X.28_3. + X.28_sd + X.28_trend +
##     X.29_3. + X.29_sd + X.29_trend + X.3_3. + X.3_sd + X.3_trend +
##     X.4_3. + X.4_sd + X.4_trend + X.5_3. + X.5_sd + X.5_trend +
##     X.6_3. + X.6_sd + X.6_trend + X.7_3. + X.7_sd + X.7_trend +
##     X.8_3. + X.8_sd + X.8_trend + X.9_3. + X.9_sd + X.9_trend
```

As we can see there are only 9 features that are statistically significant (two-tailed $p < 0.05$). As we see in the alias the function for the regression gets quite long with all features. One option to simplify this model could be to reduce the number of features to those considered significant and then train another model to check results. Yet another option would be dimensionality reduction using principal component analysis (PCA) or matrix factorization (e.g. singular value decomposition (SVD)). However, these approaches would complicate the interpretability/intuition of the model and I will leave these approaches for future exploration for now.

Goodness of fit for this model can be evaluated by finding the pseudo-R^2 since R^2 can not be used directly for logistic regression.

```
# MacFaddens R^2 for the model
pR2(glm_model)
```

```
## fitting null model for pseudo-r2
```

```
##           llh       llhNull            G2      McFadden          r2ML
## -1117.2910260 -1439.8293047   645.0765574     0.2240115     0.2621352
##          r2CU
##     0.3530056
```

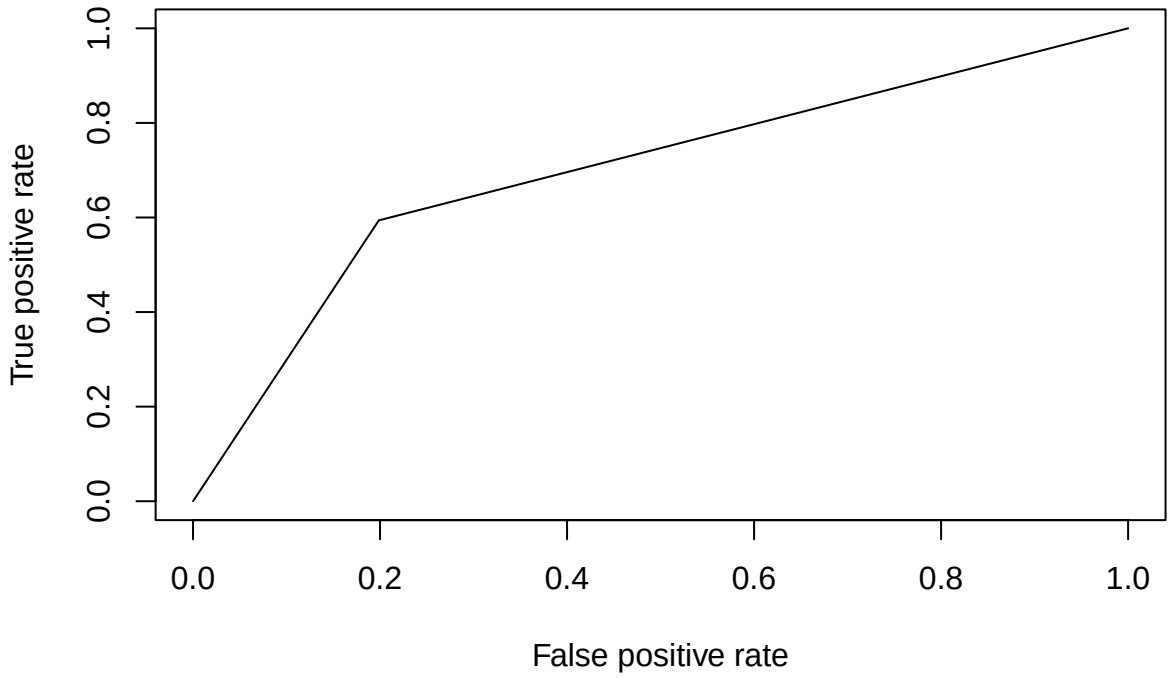
A psuedo MacFaddens R^2 of 0.224 indicates a reasonable, but not excellent fit.

With a model trained it is time to make some predictions and evaluate the performance of the model.

```
# Create predictions in the test set
glm_pred <- predict.glm(glm_model, newdata = glm_test, type='response')

# Predictions are given as probabilities, transform to binary outcome
glm_pred <- ifelse(glm_pred > 0.5,1,0)

# Construct a confusion matrix
cfm2 <- confusionMatrix(as.factor(glm_pred), as.factor(glm_test$outcome))
cfm2
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 242  93
##          1  60 136
##
##                Accuracy : 0.7119
##                  95% CI : (0.6713, 0.7501)
##     No Information Rate : 0.5687
##     P-Value [Acc > NIR] : 7.163e-12
##
##                   Kappa : 0.4022
##
##  Mcnemar's Test P-Value : 0.00968
##
##             Sensitivity : 0.8013
##             Specificity : 0.5939
##          Pos Pred Value : 0.7224
##          Neg Pred Value : 0.6939
##              Prevalence : 0.5687
##          Detection Rate : 0.4557
##    Detection Prevalence : 0.6309
##       Balanced Accuracy : 0.6976
##
##        'Positive' Class : 0
##
```

```
# Plot ROC curve and calculate the AUC
pr <- prediction(glm_pred, glm_test$outcome)
prf <- performance(pr, measure = "tpr", x.measure = "fpr")
plot(prf)
```




```
auc2 <- performance(pr, measure = "auc")
auc2 <- auc2@y.values[[1]]
auc2
```

```
## [1] 0.6976055
```

```
# Create a summary table for results
summary <- bind_rows(
  summary, tribble( ~Model, ~Accuracy, ~"95% CI Lower", ~"95% CI Upper", ~AUC,
                    "glm", cfm2$overall[[1]], cfm2$overall[[3]], cfm2$overall[[4]], auc2))
summary
```

Model	Accuracy	95% CI Lower	95% CI Upper	AUC
Random	0.4839925	0.4407364	0.5274276	0.4724688
glm	0.7118644	0.6712958	0.7500560	0.6976055

4.4 XGBoost

The second model to train is XGBoost. I will first train a model using the default parameters to evaluate performance and then try to optimize performance through hypermarameter tuning using the `caret` library. The final parameters will be used by the functions from the `XGBoost` library to train and predict on the test set. Finally, I will train a model with the best parameters on the full training data and make new predictions on unseen data in the validation set.

4.4.1 Dataset preparation

Before I can start training the algorithm I need to prepare the datasets.

```
xgb_train <- nfdata_train %>% select(-outcome, -patN)
xgb_test <- nfdata_test %>% select(-outcome, -patN)

xgb_tr_outcome <- nfdata_train$outcome
xgb_te_outcome <- nfdata_test$outcome

# Caret algorithm requires outcomes to be factors
xgb_tr_foutcome <- as.factor(xgb_tr_outcome)
xgb_te_foutcome <- as.factor(xgb_te_outcome)
```

4.4.2 Base XGBoost model

The default parameters are passed to the training algorithm in a parameter grid.

```
set.seed(93)

# Setting up the algorithm parameters
grid_default <- expand.grid(
  nrounds = 100,
  max_depth = 6,
  eta = 0.3,
  gamma = 0,
  colsample_bytree = 1,
  min_child_weight = 1,
  subsample = 1
)

# Defining
train_control <- caret::trainControl(
  method = "none" # No resampling
)

xgb_base <- caret::train(
  x = xgb_train,
  y = xgb_tr_foutcome,
  trControl = train_control,
  tuneGrid = grid_default,
  method = "xgbTree",
  verbose = FALSE
)

# Evaluating fit on test data
xgb_base_pred <- (as.numeric(predict(xgb_base, xgb_test)))-1
mean(xgb_te_outcome == xgb_base_pred)

## [1] 0.6911488

cfm3 <- confusionMatrix(as.factor(xgb_base_pred), xgb_te_foutcome)
cfm3
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 231  93
##          1  71 136
##
##                Accuracy : 0.6911
##                  95% CI : (0.6499, 0.7302)
##     No Information Rate : 0.5687
##     P-Value [Acc > NIR] : 4.568e-09
##
##                   Kappa : 0.363
##
##  Mcnemar's Test P-Value : 0.101
##
##             Sensitivity : 0.7649
##             Specificity : 0.5939
##          Pos Pred Value : 0.7130
##          Neg Pred Value : 0.6570
##              Prevalence : 0.5687
##          Detection Rate : 0.4350
##    Detection Prevalence : 0.6102
##       Balanced Accuracy : 0.6794
##
##        'Positive' Class : 0
##
```

The most important variables in the model can be inspected using the `varImp()`function.

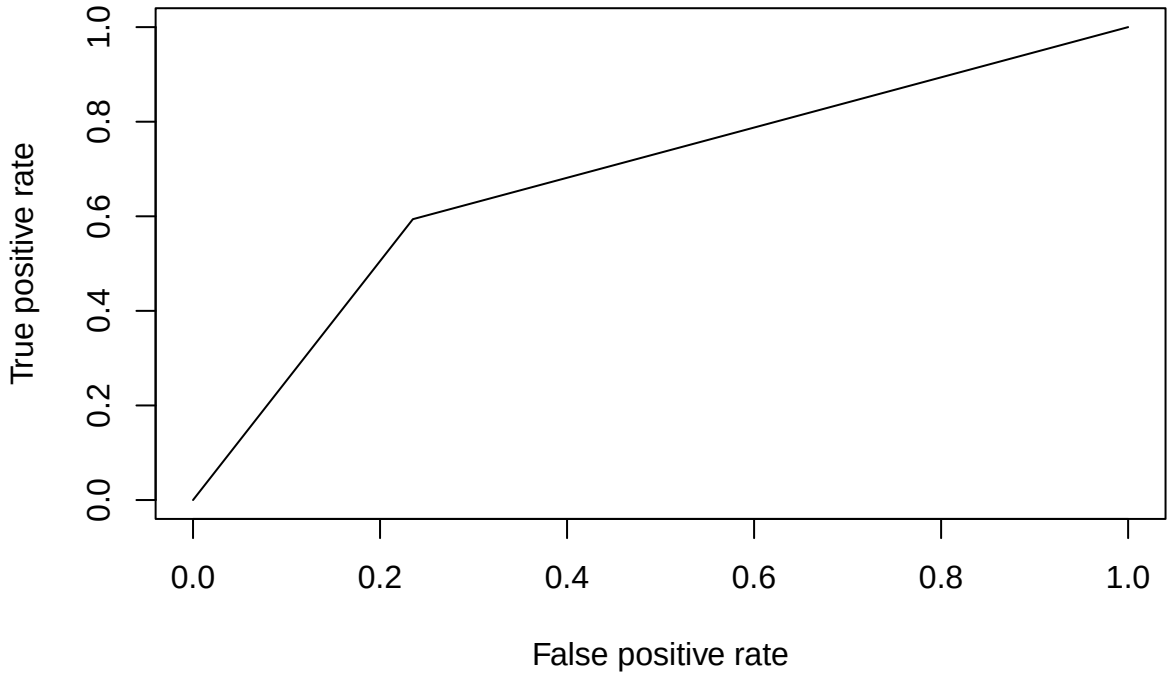
```
varImp(xgb_base)
```

```
## xgbTree variable importance
##
##   only 20 most important variables shown (out of 91)
##
##            Overall
## X.11_trend 100.000
## X.11_3.     30.392
## age         18.558
## X.4_sd      12.772
## X.16_sd     12.643
## X.6_sd      11.301
## X.5_sd      10.891
## X.11_sd     10.653
## X.17_3.     10.626
## X.26_3.      9.201
## X.9_sd       9.162
## X.20_trend   8.685
## X.24_trend   8.586
## X.17_trend   8.583
## X.24_sd      8.553
## X.24_3.      8.426
## X.26_trend   8.407
```

```
## X.28_3.      8.295
## X.14_sd      7.874
## X.25_3.      7.775
```

ROC curve for the model and the updated results table:

```
# Plot ROC curve
pr <- prediction(predictions = xgb_base_pred, labels = xgb_te_foutcome)
prf <- performance(pr, measure = "tpr", x.measure = "fpr")
plot(prf)
```




```
auc3 <- performance(pr, measure = "auc")
auc3 <- auc3@y.values[[1]]
auc3
```

```
## [1] 0.6793936
```

```
summary <- bind_rows(
  summary, tribble( ~Model, ~Accuracy, ~"95% CI Lower", ~"95% CI Upper", ~AUC,
                    "XGBoost - default", cfm3$overall[[1]], cfm3$overall[[3]], cfm3$overall[[4]], auc3))

summary
```

Model	Accuracy	95% CI Lower	95% CI Upper	AUC
Random	0.4839925	0.4407364	0.5274276	0.4724688
glm	0.7118644	0.6712958	0.7500560	0.6976055
XGBoost - default	0.6911488	0.6499247	0.7302296	0.6793936

4.4.3 Hyperparameter tuning

To see if I can improve on the default XGBoost performance I will test with different hyperparameter setting. Ideally, this should be done as one grid search through all parameters, but that is computationally very expensive. I will instead take step-wise approach where I identify the best parameter setting for 1-2 hyperparameters at a time.

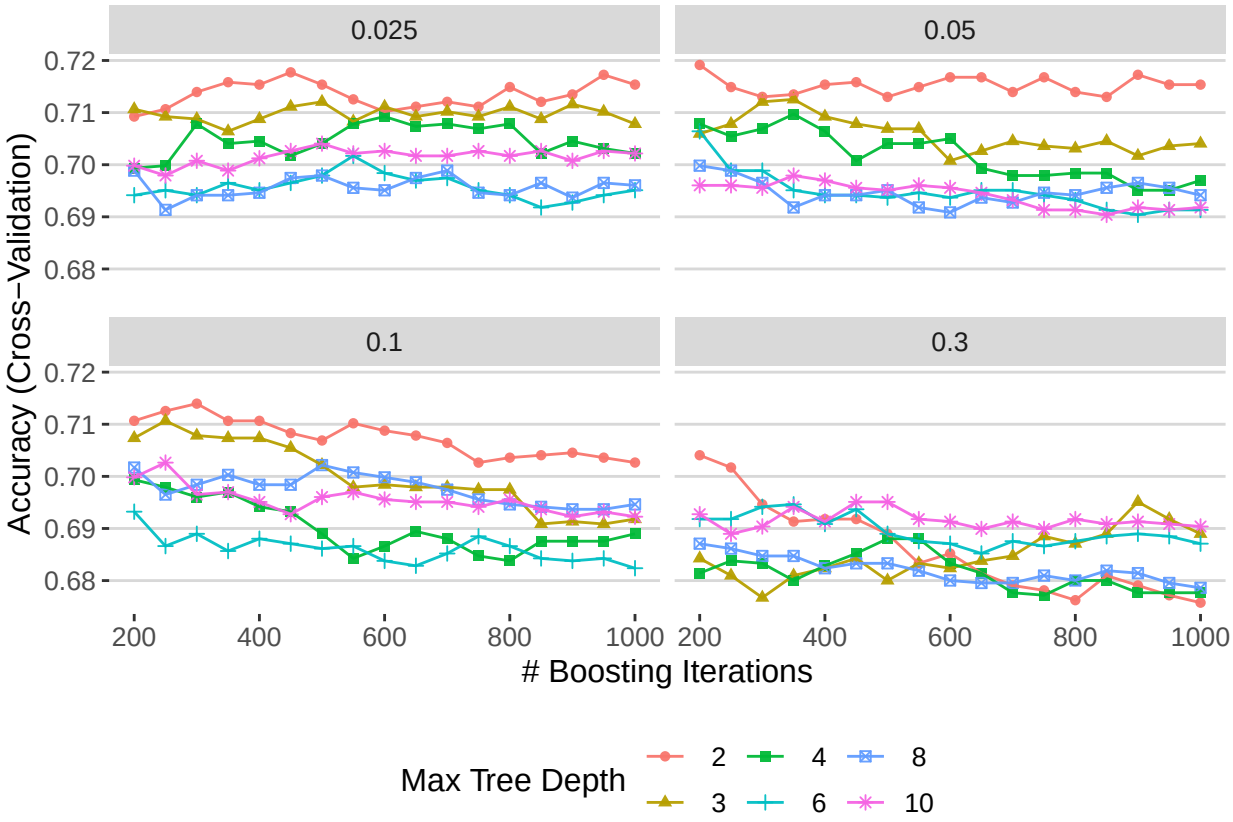
To ease visualization of the results I start by defining a plot function that I will use in all steps.

```
# Plot function
tuneplot <- function(x, probs = .99) {
  ggplot(x) +
    coord_cartesian(ylim = c(min(x$results$Accuracy), max(x$results$Accuracy))) +
    theme_hc()
}
```

First I will test different learning rates and tree depths using 5-fold cross-validation. I let the model iterate for 1000 rounds, and start at 200 rounds to avoid initial noise.

```
# Tune learning rate and tree depth
nrounds <- 1000

tune_grid <- expand.grid(
  nrounds = seq(from = 200, to = nrounds, by = 50),
  eta = c(0.025, 0.05, 0.1, 0.3),
  max_depth = c(2, 3, 4, 6, 8, 10),
  gamma = 0,
  colsample_bytree = 1,
  min_child_weight = 1,
  subsample = 1
)

tune_control <- caret::trainControl(
  method = "cv", # cross-validation
  number = 5, # with n folds
  verboseIter = FALSE,
  allowParallel = TRUE,
  seeds = set.seed(45)
)

xgb_tune_depth <- caret::train(
  x = xgb_train,
  y = xgb_tr_foutcome,
  trControl = tune_control,
  tuneGrid = tune_grid,
  method = "xgbTree",
  verbosity = 0
)
```

```
tuneplot(xgb_tune_depth)
```



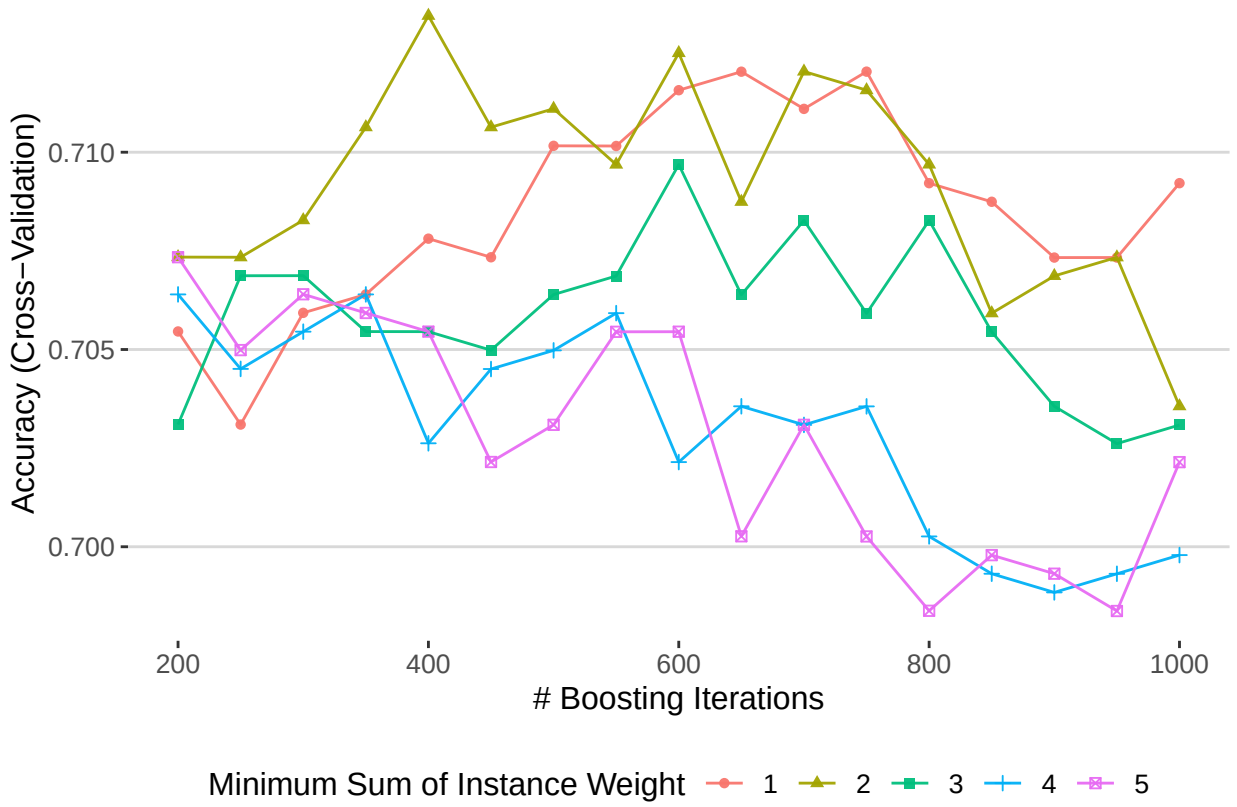

```
xgb_tune_depth$bestTune
```

	nrounds	max_depth	eta	gamma	colsample_bytree	min_child_weight	subsample
103	200	2	0.05	0	1	1	1

The best parameter setting seems to 0.05 for eta and a max tree depth of 2. I will use these parameters as I initialize another grid tune for optimal child weight.

```
# Tune child weight
set.seed(62)
nrounds <- 1000

tune_grid <- expand.grid(
  nrounds = seq(from = 200, to = nrounds, by = 50),
  eta = xgb_tune_depth$bestTune$eta,
  max_depth = xgb_tune_depth$bestTune$max_depth,
  gamma = 0,
  colsample_bytree = 1,
  min_child_weight = c(1, 2, 3, 4, 5),
  subsample = 1
```

```
)

tune_control <- caret::trainControl(
  method = "cv", # cross-validation
  number = 5, # with n folds
  verboseIter = FALSE,
  allowParallel = TRUE,
  seeds = set.seed(62)
)

xgb_tune_child <- caret::train(
  x = xgb_train,
  y = xgb_tr_foutcome,
  trControl = tune_control,
  tuneGrid = tune_grid,
  method = "xgbTree",
  verbosity = 0
)


tuneplot(xgb_tune_child)
```



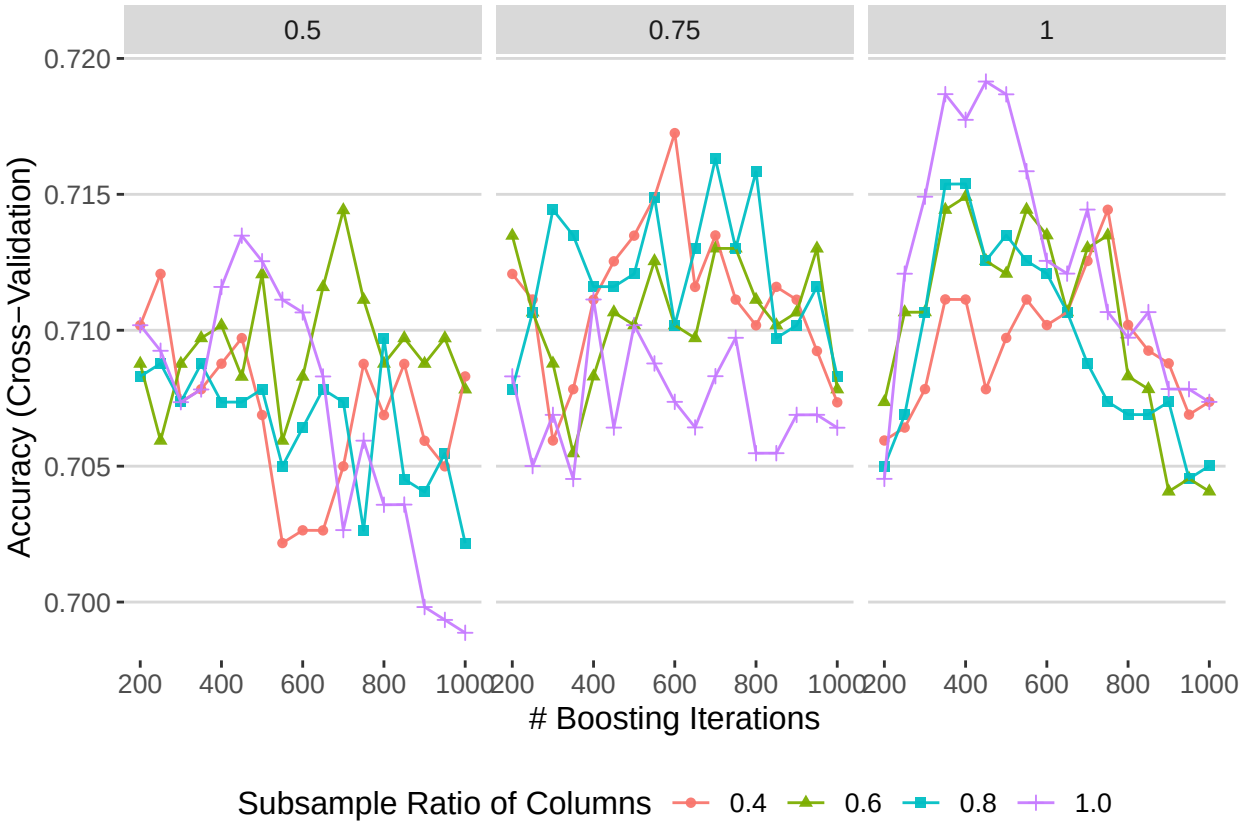

```
xgb_tune_child$bestTune
```

	nrounds	max_depth	eta	gamma	colsample_bytree	min_child_weight	subsample
22	400	2	0.05	0	1	2	1

The best setting for the minimum child wwight parameter seems to be 2.

Next, I will initialize another grid tune for the column sample and subsample rate hyperparameters.

```
# Tune column sample and subsample rate
nrounds <- 1000

tune_grid <- expand.grid(
  nrounds = seq(from = 200, to = nrounds, by = 50),
  eta = xgb_tune_depth$bestTune$eta,
  max_depth = xgb_tune_depth$bestTune$max_depth,
  gamma = 0,
  colsample_bytree = c(0.4, 0.6, 0.8, 1.0),
  min_child_weight = xgb_tune_child$bestTune$min_child_weight,
  subsample = c(0.5, 0.75, 1.0)
)

tune_control <- caret::trainControl(
  method = "cv", # cross-validation
  number = 5, # with n folds
  verboseIter = FALSE,
  allowParallel = TRUE,
  seeds = set.seed(27)
)

xgb_tune_sample <- caret::train(
  x = xgb_train,
  y = xgb_tr_foutcome,
  trControl = tune_control,
  tuneGrid = tune_grid,
  method = "xgbTree",
  verbosity = 0
)


tuneplot(xgb_tune_sample)
```




```
xgb_tune_sample$bestTune
```

	nrounds	max_depth	eta	gamma	colsample_bytree	min_child_weight	subsample
193	450	2	0.05	0	1	2	1

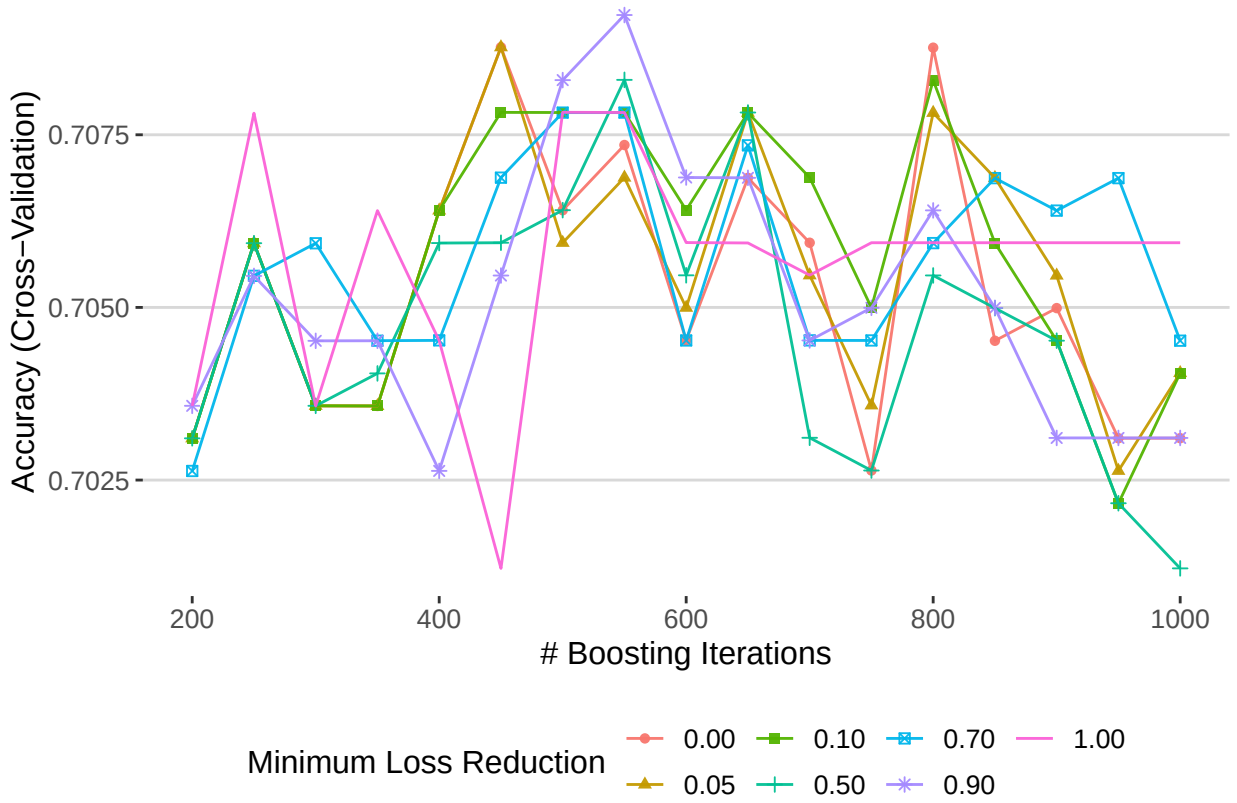
The default settings of column sample, and subsample rate of 1 seems to be the best settings.

The optimal gamma setting could be important to avoid overtraining the model. I initialize another grid tune for this parameter.

```
# Tune gamma
nrounds <- 1000

tune_grid <- expand.grid(
  nrounds = seq(from = 200, to = nrounds, by = 50),
  eta = xgb_tune_depth$bestTune$eta,
  max_depth = xgb_tune_depth$bestTune$max_depth,
  gamma = c(0, 0.05, 0.1, 0.5, 0.7, 0.9, 1.0),
  colsample_bytree = xgb_tune_sample$bestTune$colsample_bytree,
  min_child_weight = xgb_tune_child$bestTune$min_child_weight,
  subsample = xgb_tune_sample$bestTune$subsample
)

tune_control <- caret::trainControl(
```

```
  method = "cv", # cross-validation
  number = 5, # with n folds
  verboseIter = FALSE,
  allowParallel = TRUE,
  seeds = set.seed(15)
)

xgb_tune_gamma <- caret::train(
  x = xgb_train,
  y = xgb_tr_foutcome,
  trControl = tune_control,
  tuneGrid = tune_grid,
  method = "xgbTree",
  verbosity = 0
)
```

```
tuneplot(xgb_tune_gamma)
```



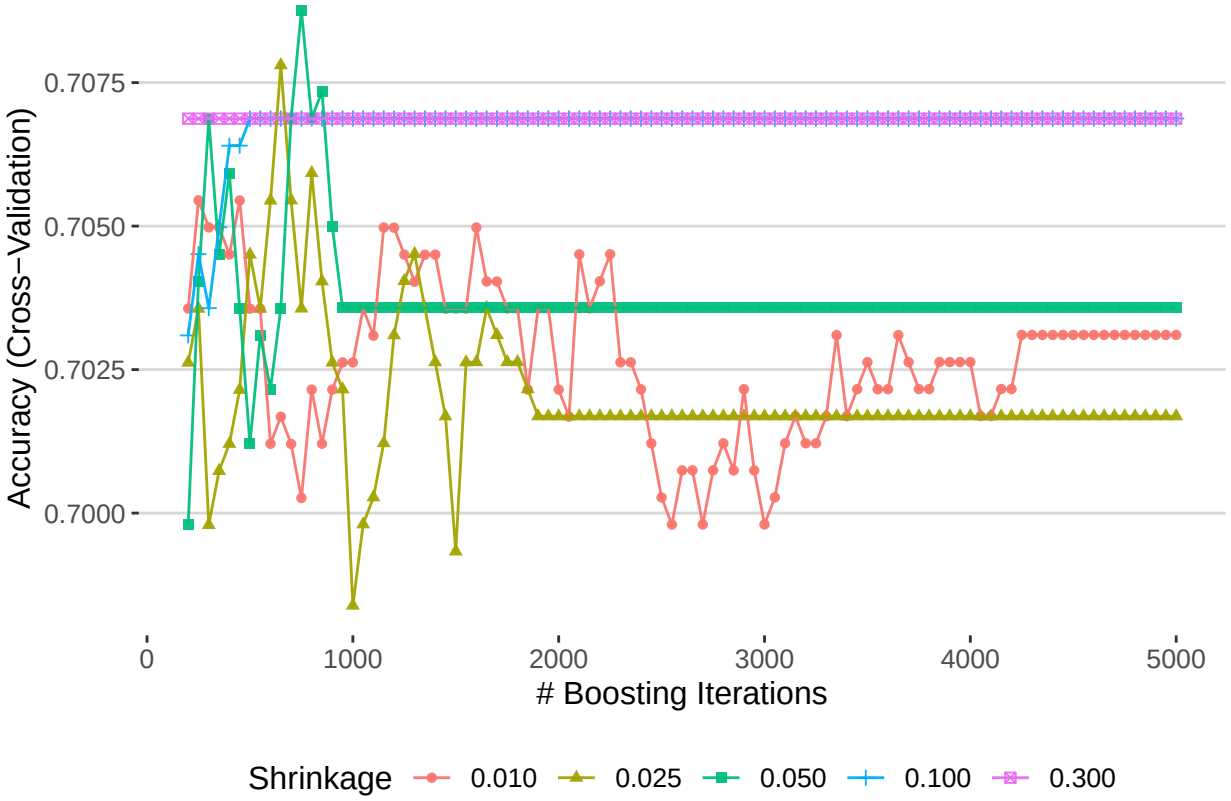

```
xgb_tune_gamma$bestTune
```

	nrounds	max_depth	eta	gamma	colsample_bytree	min_child_weight	subsample
93	550	2	0.05	0.9	1	2	1

A higher gamma seems to yield better results, with 0.9 being the optimal setting.

Finally, I will check to see if the learning rate, eta, can be optimalized.

```
# Tune eta
nrounds <- 5000

tune_grid <- expand.grid(
  nrounds = seq(from = 200, to = nrounds, by = 50),
  eta = c(0.01, 0.025, 0.05, 0.1, 0.3),
  max_depth = xgb_tune_depth$bestTune$max_depth,
  gamma = xgb_tune_gamma$bestTune$gamma,
  colsample_bytree = xgb_tune_sample$bestTune$colsample_bytree,
  min_child_weight = xgb_tune_child$bestTune$min_child_weight,
  subsample = xgb_tune_sample$bestTune$subsample
)

tune_control <- caret::trainControl(
  method = "cv", # cross-validation
  number = 5, # with n folds
  verboseIter = FALSE,
  allowParallel = TRUE,
  seeds = set.seed(52)
)

xgb_tune_eta <- caret::train(
  x = xgb_train,
  y = xgb_tr_foutcome,
  trControl = tune_control,
  tuneGrid = tune_grid,
  method = "xgbTree",
  verbosity = 0
)


tuneplot(xgb_tune_eta)
```




```
xgb_tune_eta$bestTune
```

	nrounds	max_depth	eta	gamma	colsample_bytree	min_child_weight	subsample
206	750	2	0.05	0.9	1	2	1

A learning rate of 0.05 gives the highest accuracy, and it seems like a higher learning rate quickly reaches a level where no further improvements can be made. Interestingly, lower learning rates also plateaus, but at lower accuracy, although the margins are small.

4.4.4 Optimal XGBoost

For the rest of the XGBoost model training I will use the `XGBoost` library.

```
if(!require(xgboost)) install.packages("xgboost", repos = "http://cran.us.r-project.org")
if(!require(DiagrammeR)) install.packages("DiagrammeR", repos = "http://cran.us.r-project.org") #required
if(!require(Ckmeans.1d.dp)) install.packages("Ckmeans.1d.dp", repos = "http://cran.us.r-project.org") #req
```

The `XGBoost` library requires data to be in specific formats.

```
# The XGBoost library requires a matrix for outcomes
xgb_tr_outcome <- as.matrix(xgb_tr_outcome)
xgb_te_outcome <- as.matrix(xgb_te_outcome)
```

```
# Converting the datasets into the xgb.DMatrix format for the XGBoost algorithms
xgb_train_xgb <- xgb.DMatrix(data = as.matrix(xgb_train), label = xgb_tr_outcome)
xgb_test_xgb <- xgb.DMatrix(data = as.matrix(xgb_test), label = xgb_te_outcome)
```

I will use the hyperparameters resulting from the tuning process with the `caret` library.

```
# Best parameteres from tuning
params <- list(booster = "gbtree",
               objective = "binary:logistic",
               eta=0.05,
               gamma= 0.9,
               alpha = 0,
               lambda = 0,
               max_depth = 2,
               min_child_weight= 2,
               subsample= 1,
               colsample_bytree= 1,
               eval_metric = "error")
```

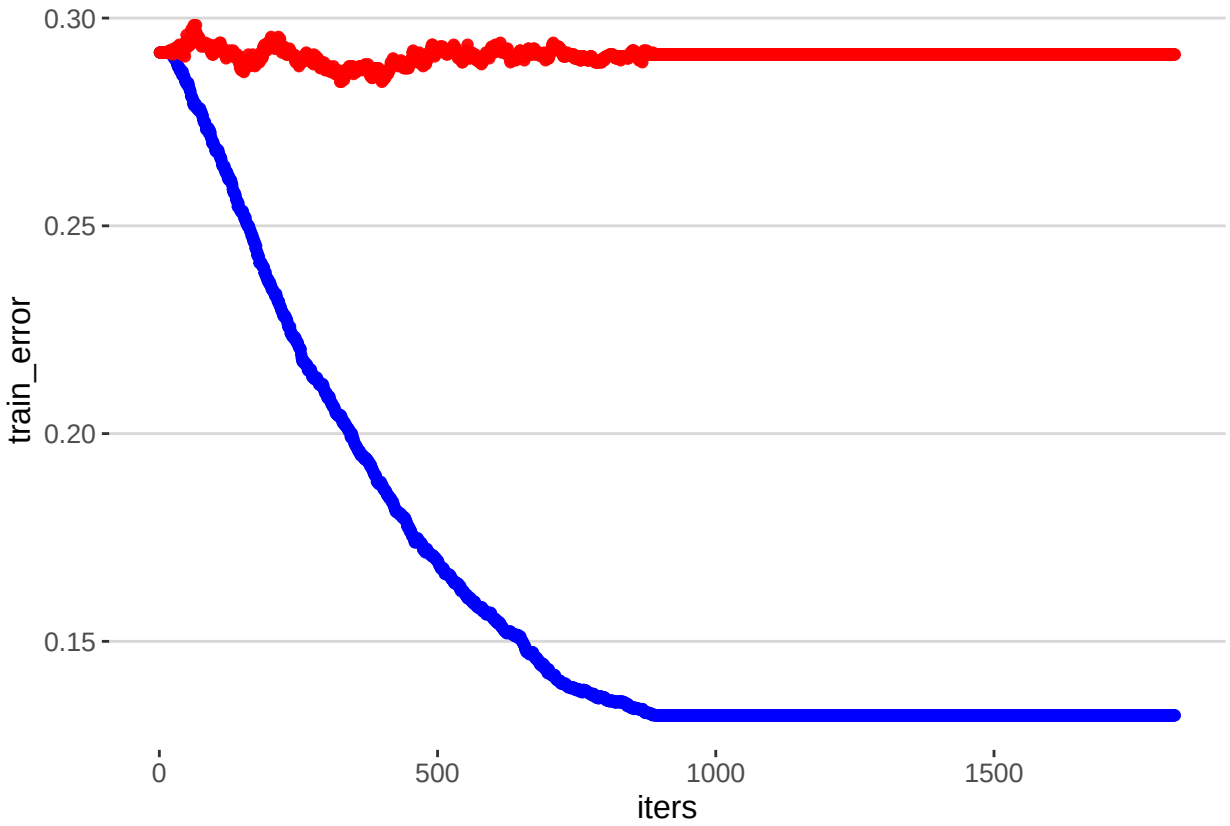
First, I want to find the optimal number of training rounds for the model using 5-fold cross validation in the training set.

```
# 5-fold cross-validation with XGBoost library to find optimal nrounds
set.seed(68)
xgbcv <- xgb.cv(data = as.matrix(xgb_train),
               label = xgb_tr_outcome,
               params = params,
               nrounds = 2000,
               nfold = 5,
               showsd = T,
               stratified = T,
               print_every_n = 50,
               early_stopping_rounds = 1500,
               maximize = F)
```

```
## [1]  train-error:0.291706+0.003045   test-error:0.291711+0.012183
## Multiple eval metrics are present. Will use test_error for early stopping.
## Will train until test_error hasn't improved in 1500 rounds.
##
## [51] train-error:0.283813+0.006428   test-error:0.293596+0.014902
## [101]    train-error:0.268026+0.004626   test-error:0.292658+0.013977
## [151]    train-error:0.252946+0.003341   test-error:0.287472+0.012626
## [201]    train-error:0.235509+0.003404   test-error:0.295481+0.010935
## [251]    train-error:0.220783+0.003284   test-error:0.288412+0.016590
## [301]    train-error:0.209472+0.003798   test-error:0.287470+0.018760
## [351]    train-error:0.197573+0.003309   test-error:0.287943+0.019575
## [401]    train-error:0.187559+0.002997   test-error:0.285116+0.023725
## [451]    train-error:0.176837+0.003138   test-error:0.289360+0.026341
## [501]    train-error:0.169297+0.003314   test-error:0.291714+0.025195
## [551]    train-error:0.161169+0.001414   test-error:0.291715+0.027159
## [601]    train-error:0.155278+0.002954   test-error:0.292659+0.026519
## [651]    train-error:0.150212+0.003014   test-error:0.291243+0.024267
```

```
## [701]    train-error:0.142672+0.003597   test-error:0.291716+0.027660
## [751]    train-error:0.138195+0.003341   test-error:0.290301+0.025902
## [801]    train-error:0.136428+0.004233   test-error:0.290302+0.026841
## [851]    train-error:0.134072+0.006789   test-error:0.292189+0.029414
## [901]    train-error:0.132188+0.009372   test-error:0.291245+0.026859
## [951]    train-error:0.132188+0.009372   test-error:0.291245+0.026859
## [1001]   train-error:0.132188+0.009372   test-error:0.291245+0.026859
## [1051]   train-error:0.132188+0.009372   test-error:0.291245+0.026859
## [1101]   train-error:0.132188+0.009372   test-error:0.291245+0.026859
## [1151]   train-error:0.132188+0.009372   test-error:0.291245+0.026859
## [1201]   train-error:0.132188+0.009372   test-error:0.291245+0.026859
## [1251]   train-error:0.132188+0.009372   test-error:0.291245+0.026859
## [1301]   train-error:0.132188+0.009372   test-error:0.291245+0.026859
## [1351]   train-error:0.132188+0.009372   test-error:0.291245+0.026859
## [1401]   train-error:0.132188+0.009372   test-error:0.291245+0.026859
## [1451]   train-error:0.132188+0.009372   test-error:0.291245+0.026859
## [1501]   train-error:0.132188+0.009372   test-error:0.291245+0.026859
## [1551]   train-error:0.132188+0.009372   test-error:0.291245+0.026859
## [1601]   train-error:0.132188+0.009372   test-error:0.291245+0.026859
## [1651]   train-error:0.132188+0.009372   test-error:0.291245+0.026859
## [1701]   train-error:0.132188+0.009372   test-error:0.291245+0.026859
## [1751]   train-error:0.132188+0.009372   test-error:0.291245+0.026859
## [1801]   train-error:0.132188+0.009372   test-error:0.291245+0.026859
## Stopping. Best iteration:
## [325]    train-error:0.204170+0.003277   test-error:0.284642+0.018852
```

```
# Plot error curves
cv_model_error <- tibble(train_error = xgbcv$evaluation_log$train_error_mean,
                         test_error = xgbcv$evaluation_log$test_error_mean,
                         iters = seq(1, length(xgbcv$evaluation_log$train_error_mean), 1))

ggplot(cv_model_error) + geom_point(aes(iters, train_error), color = "blue") +
  geom_point(aes(iters, test_error), color = "red") +
  theme_hc()
```



The cross validation indicates that the optimal training rounds are around 325. No further improvement in model accuracy in cross-validation is achieved with continued training.

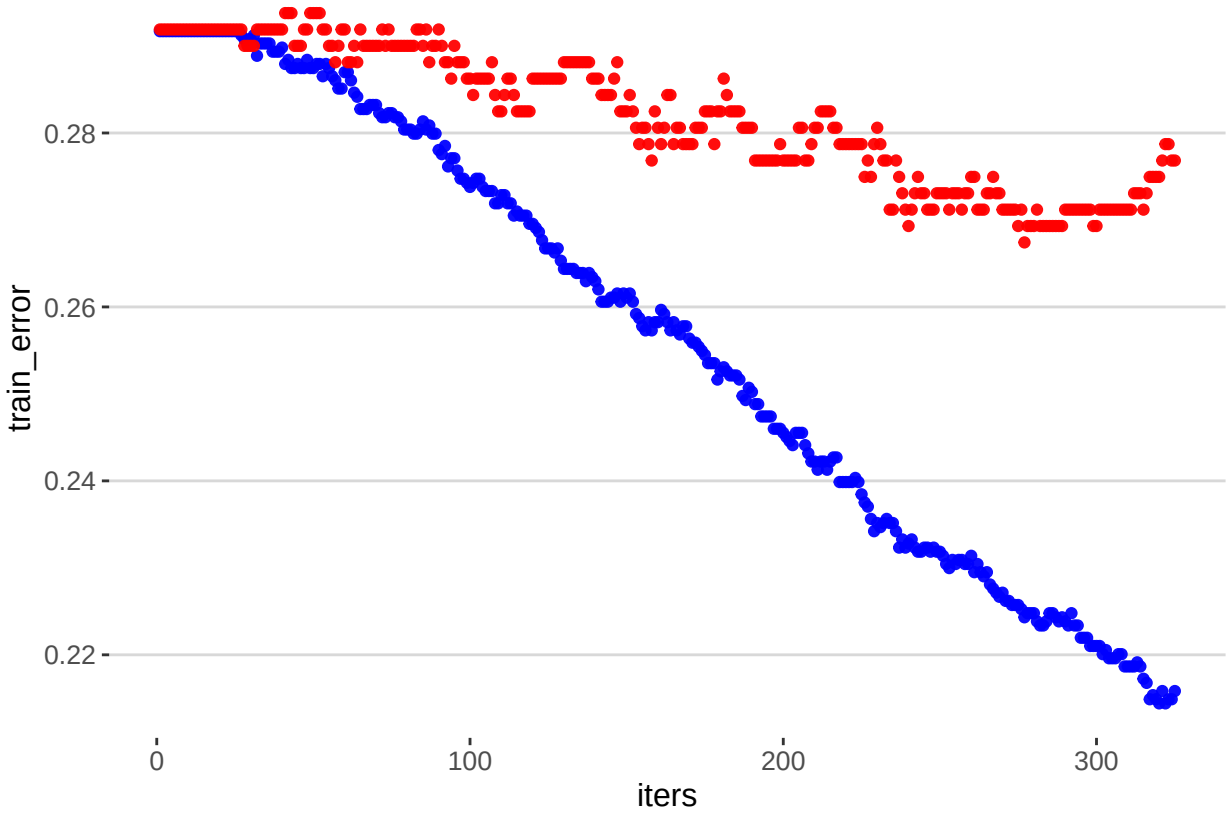
I train the model using the hyperparameters from the tuning process and the number of rounds found through cross-validation. The reference this time will be the test set that the model has not seen so far.

```
# Train model with hyperparameters from the tuning process

set.seed(75)
xgb <- xgb.train(params = params,
                 data = xgb_train_xgb,
                 nrounds = 325,
                 watchlist = list(test = xgb_test_xgb, train = xgb_train_xgb),
                 print_every_n = 10,
                 early_stopping_rounds = 200,
                 maximize = F)
```

```
## [1]  test-error:0.291902 train-error:0.291706
## Multiple eval metrics are present. Will use train_error for early stopping.
## Will train until train_error hasn't improved in 200 rounds.
##
## [11] test-error:0.291902 train-error:0.291706
## [21] test-error:0.291902 train-error:0.291706
## [31] test-error:0.290019 train-error:0.291235
## [41] test-error:0.293785 train-error:0.287936
## [51] test-error:0.293785 train-error:0.287936
## [61] test-error:0.288136 train-error:0.286993
```

```
## [71] test-error:0.290019 train-error:0.282281
## [81] test-error:0.290019 train-error:0.280396
## [91] test-error:0.290019 train-error:0.277568
## [101]    test-error:0.284369 train-error:0.274270
## [111]    test-error:0.284369 train-error:0.272856
## [121]    test-error:0.286252 train-error:0.269086
## [131]    test-error:0.288136 train-error:0.264373
## [141]    test-error:0.286252 train-error:0.262017
## [151]    test-error:0.284369 train-error:0.261546
## [161]    test-error:0.278719 train-error:0.259661
## [171]    test-error:0.278719 train-error:0.255891
## [181]    test-error:0.286252 train-error:0.253063
## [191]    test-error:0.276836 train-error:0.248822
## [201]    test-error:0.276836 train-error:0.245052
## [211]    test-error:0.280603 train-error:0.241282
## [221]    test-error:0.278719 train-error:0.239868
## [231]    test-error:0.278719 train-error:0.234684
## [241]    test-error:0.271186 train-error:0.233270
## [251]    test-error:0.273070 train-error:0.231385
## [261]    test-error:0.274953 train-error:0.229500
## [271]    test-error:0.271186 train-error:0.226202
## [281]    test-error:0.271186 train-error:0.223845
## [291]    test-error:0.271186 train-error:0.223374
## [301]    test-error:0.271186 train-error:0.221018
## [311]    test-error:0.271186 train-error:0.218662
## [321]    test-error:0.276836 train-error:0.215834
## [325]    test-error:0.276836 train-error:0.215834
```

```
# Plot error
model_error <- tibble(train_error = xgb$evaluation_log$train_error,
                      test_error = xgb$evaluation_log$test_error,
                      iters = seq(1, length(train_error), 1))

ggplot(model_error) + geom_point(aes(iters, train_error), color = "blue") +
  geom_point(aes(iters, test_error), color = "red") +
  theme_hc()
```



The models performance in the test set is similar to what was seen during cross-validation.

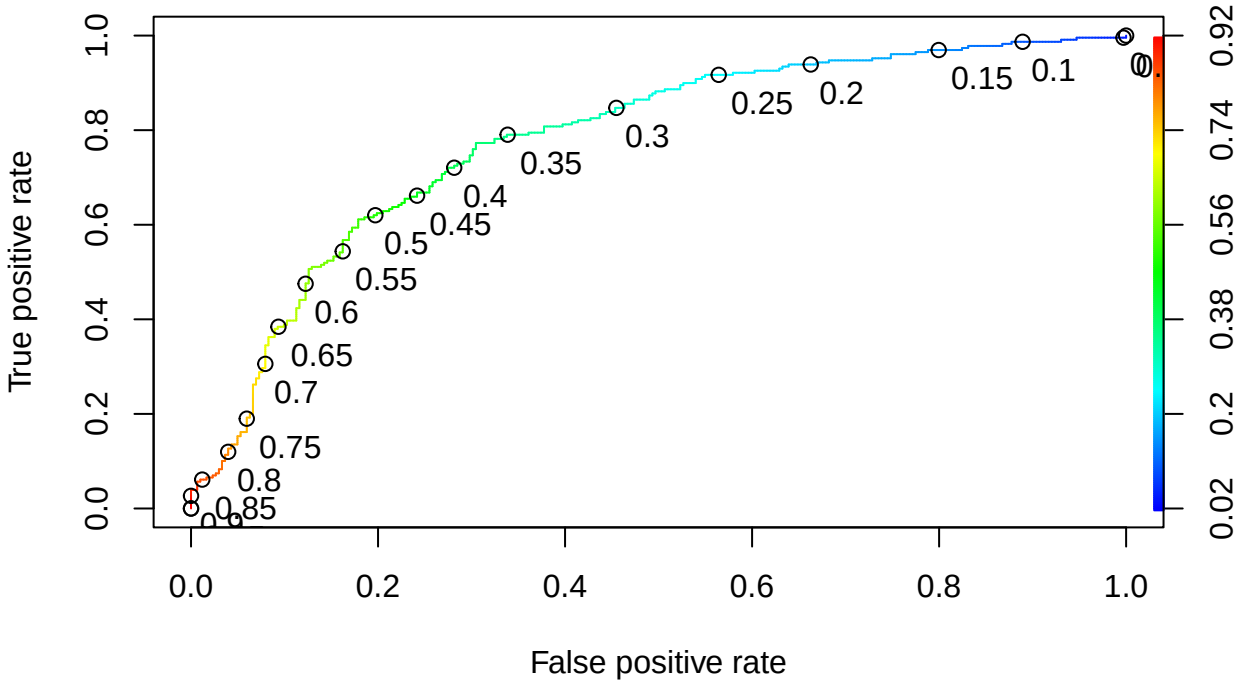
It's time to make some predictions!

```
# Make predictions on test set
xgbpred <- predict(xgb, xgb_test_xgb)
xgbpreds <- ifelse(xgbpred > 0.5,1,0)

cfm4 <- confusionMatrix(as.factor(xgbpreds), as.factor(xgb_te_outcome))
cfm4
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 243  87
##          1  59 142
##
##                Accuracy : 0.725
##                  95% CI : (0.6849, 0.7626)
##     No Information Rate : 0.5687
##     P-Value [Acc > NIR] : 6.686e-14
##
##                   Kappa : 0.4311
##
##  Mcnemar's Test P-Value : 0.02545
##
```

```
##             Sensitivity : 0.8046
##             Specificity : 0.6201
##          Pos Pred Value : 0.7364
##          Neg Pred Value : 0.7065
##              Prevalence : 0.5687
##          Detection Rate : 0.4576
##    Detection Prevalence : 0.6215
##       Balanced Accuracy : 0.7124
##
##        'Positive' Class : 0
##
```

```
# Plot ROC curve
pr <- prediction(predictions = xgbpred, labels = xgb_te_outcome)
prf <- performance(pr, measure = "tpr", x.measure = "fpr")
plot(prf,
     colorize = TRUE,
     print.cutoffs.at= seq(0,1,0.05),
     text.adj=c(-0.2,1.7))
```




```
auc4 <- performance(pr, measure = "auc")
auc4 <- auc4@y.values[[1]]
auc4
```

```
## [1] 0.7787385
```

```
summary <- bind_rows(
  summary, tribble( ~Model, ~Accuracy, ~"95% CI Lower", ~"95% CI Upper", ~AUC,
                    "XGBoost - tuned", cfm4$overall[[1]], cfm4$overall[[3]], cfm4$overall[[4]], auc4))

knitr::kable(summary)
```

Model	Accuracy	95% CI Lower	95% CI Upper	AUC
Random	0.4839925	0.4407364	0.5274276	0.4724688
glm	0.7118644	0.6712958	0.7500560	0.6976055
XGBoost - default	0.6911488	0.6499247	0.7302296	0.6793936
XGBoost - tuned	0.7250471	0.6849460	0.7626222	0.7787385

4.5 Validation

I will use the validation set for the final performance validation. This dataset has up until now been unseen by the algorithms training the model. Although the `glm` model performed similarly to the `XGBoost` model I will only test the `XGBoost` model on the validation data. Practical use and interpretability of the glm model might require reducing the amount of features which I have not included as part of this research.

4.5.1 Training the model

I train the model on the full training dataset (train+test).

```
# Train final model on all training data and test in validation set
xgb_train_full <- nfdata_train_full %>% select(-outcome, -patN)
xgb_valid <- nfdata_validation %>% select(-outcome, -patN)

xgb_train_outcome <- as.matrix(nfdata_train_full$outcome)
xgb_vali_outcome <- as.matrix(nfdata_validation$outcome)

xgb_train_f <- xgb.DMatrix(data = as.matrix(xgb_train_full), label = xgb_train_outcome)
xgb_vali <- xgb.DMatrix(data = as.matrix(xgb_valid), label = xgb_vali_outcome)

set.seed(1536489)
xgb_validation <- xgb.train(params = params,
                 data = xgb_train_f,
                 nrounds = 325,
                 print_every_n = 10,
                 early_stopping_rounds = NULL,
                 maximize = F)
```

4.5.2 Predictions on new data

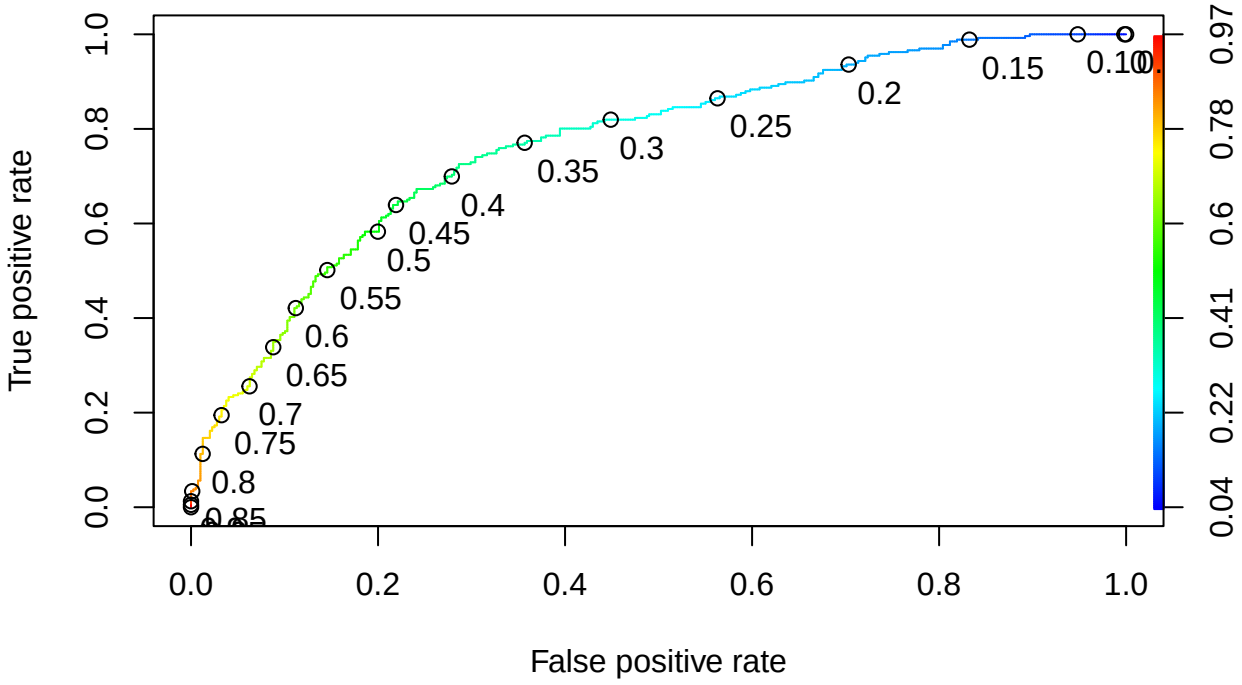
Make predictions, create confusion matrix and plot ROC curve.

```
# Make predictions on validation set
xgbpred_v <- predict(xgb_validation, xgb_vali)
xgbpreds_v <- ifelse (xgbpred_v > 0.5,1,0)
```

```r
cfm_v <- confusionMatrix (as.factor(xgbpreds_v), as.factor(xgb_vali_outcome))
cfm_v
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 319 111
##          1  79 155
##
##                Accuracy : 0.7139
##                  95% CI : (0.6778, 0.748)
##     No Information Rate : 0.5994
##     P-Value [Acc > NIR] : 5.178e-10
##
##                   Kappa : 0.392
##
##  Mcnemar's Test P-Value : 0.02451
##
##             Sensitivity : 0.8015
##             Specificity : 0.5827
##          Pos Pred Value : 0.7419
##          Neg Pred Value : 0.6624
##              Prevalence : 0.5994
##          Detection Rate : 0.4804
##    Detection Prevalence : 0.6476
##       Balanced Accuracy : 0.6921
##
##        'Positive' Class : 0
##
```

```r
# Plot ROC curve
pr <- prediction(predictions = xgbpred_v, labels = xgb_vali_outcome)
prf <- performance(pr, measure = "tpr", x.measure = "fpr")
plot(prf,
     colorize = TRUE,
     print.cutoffs.at= seq(0,1,0.05),
     text.adj=c(-0.2,1.7))
```




```
auc_v <- performance(pr, measure = "auc")
auc_v <- auc_v@y.values[[1]]
auc_v
```

```
## [1] 0.7689765
```

```
# Create a summary table for validation results
summary_v <- tribble( ~Model, ~Accuracy, ~"95% CI Lower", ~"95% CI Upper", ~AUC,
                      "Validation", cfm_v$overall[[1]], cfm_v$overall[[3]], cfm_v$overall[[4]], auc_v)

summary_v
```

Model	Accuracy	95% CI Lower	95% CI Upper	AUC
Validation	0.7138554	0.6778274	0.7479697	0.7689765

The accuracy in predicting lack of treatment improvement is 0.7138554 with an AUC of 0.7689765 on the ROC curve. These are not stellar results, but in line with what could be expected in this dataset. More interesting is how this model can be applied to clinical practice in helping therapists understand which items or dimensions that are important to improve together with the patients.

4.6 Model explanation

For a machine learning model being applied to this dataset, explaining the model and the results is crucial. To try to visualize how the model makes predictions I will first visualize the global feature effects before I present

visualizations to help understand individual patient predictions.

4.6.1 Examine results

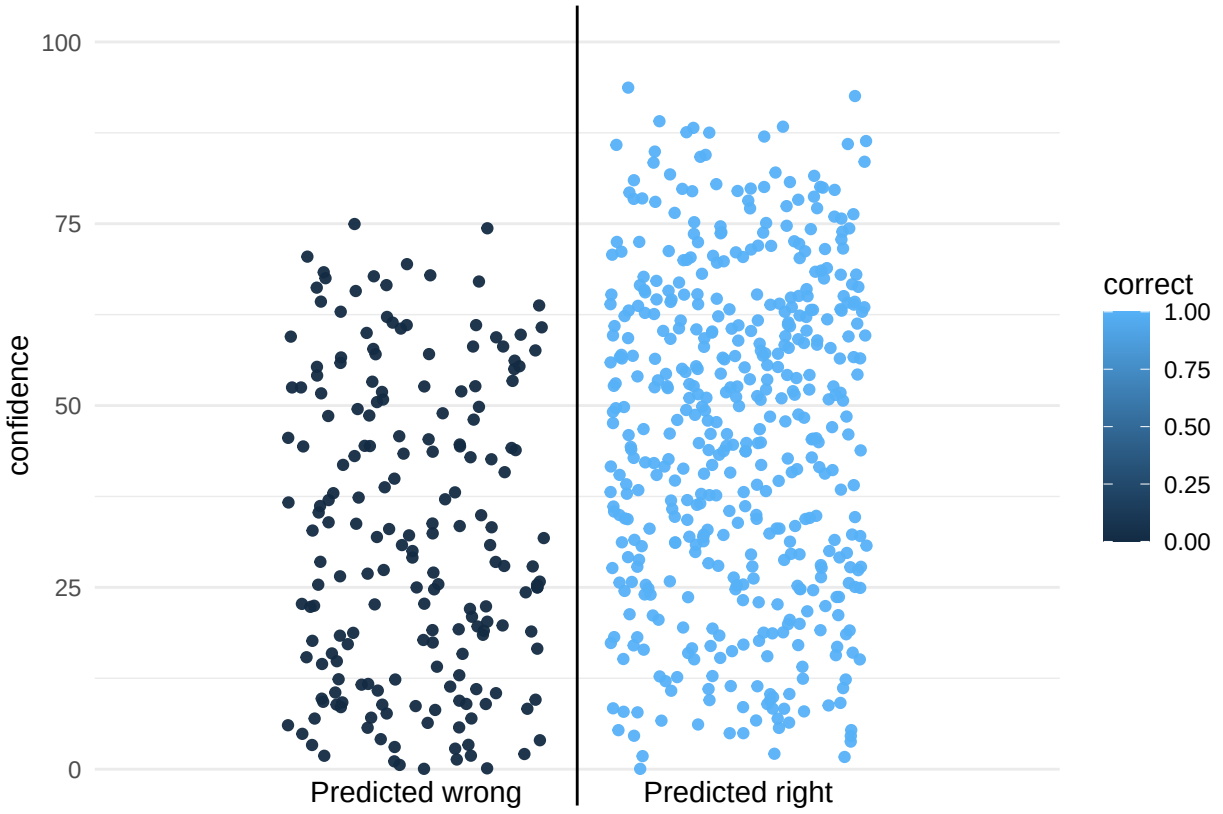
First, I will take a look at how the model have performed in terms of prediction/probability and calculate how confident the model was in the predictions as the distance from the prediction center (0.5), in percentage.

```
# Create a table of predictions, probability from model, confidence in prediction (in percent) and outcome
preds_v <- tibble("prediction" = xgbpreds_v,
                  "probability" = xgbpred_v,
                  "confidence" = abs(xgbpred_v/0.5 -1)*100,
                  "y" = xgb_vali_outcome,
                  "correct" = ifelse(xgbpreds_v == xgb_vali_outcome, 1, 0))

pander(head(preds_v, 20))
```

prediction	probability	confidence	y	correct
0	0.4522	9.553	1	0
1	0.5066	1.33	0	0
0	0.2908	41.84	1	0
0	0.3211	35.78	0	1
0	0.3444	31.13	0	1
0	0.2732	45.36	1	0
0	0.3717	25.66	0	1
0	0.1803	63.94	0	1
0	0.4072	18.55	0	1
1	0.7524	50.47	0	0
0	0.1855	62.9	1	0
1	0.6344	26.88	0	0
0	0.2752	44.96	0	1
0	0.4794	4.125	1	0
1	0.7042	40.84	0	0
0	0.4296	14.08	0	1
0	0.1947	61.06	1	0
0	0.2757	44.85	0	1
1	0.6697	33.95	0	0
0	0.1823	63.54	0	1

The model seems to have made wrong predictions despite relatively high confidence in some instances. Let's plot the confidence levels versus right and wrong predictions.

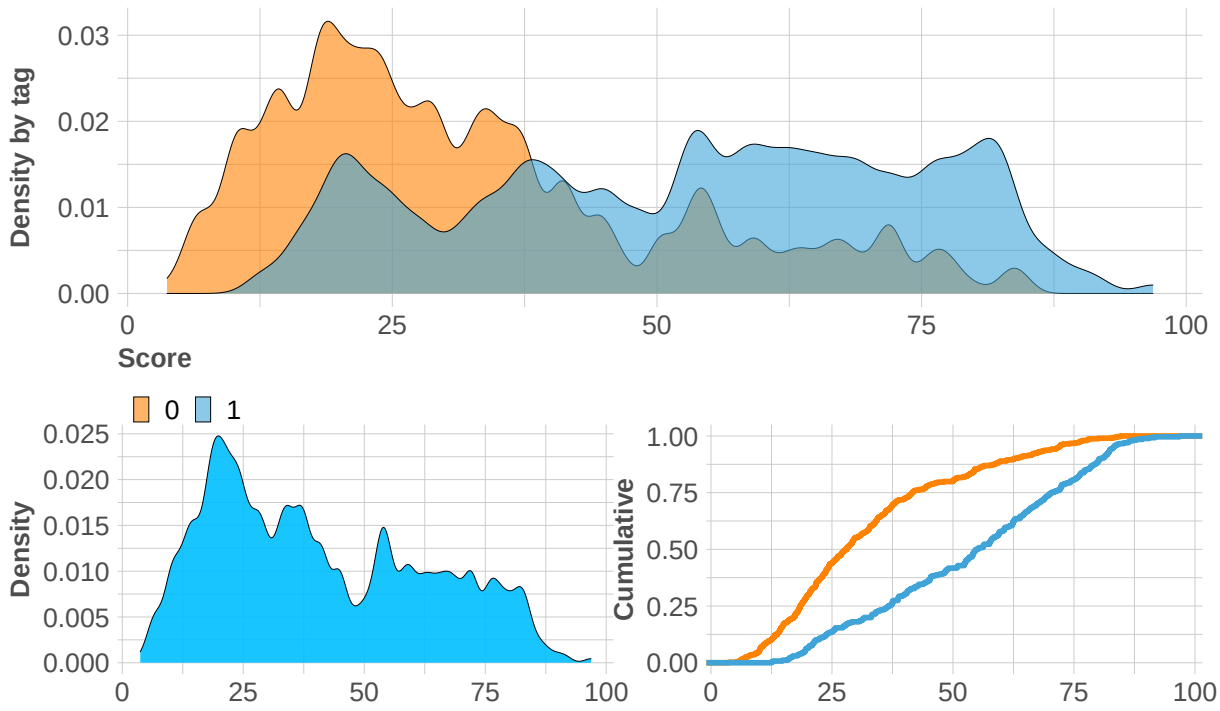
```
preds_v %>% ggplot(aes(correct, confidence, color = correct)) +
  geom_jitter() +
  geom_vline(xintercept = 0.5) +
  scale_x_discrete(labels = NULL) +
  labs(x = "") +
  annotate("text", x = 0, y =  -3, label = "Predicted wrong") +
  annotate("text", x = 1, y =  -3, label = "Predicted right") +
  coord_cartesian(ylim = c(0, 100), clip = "off") +
  theme_minimal()
```



The observation seems to hold true, it's not until over 75% confidence that the distribution of confidence versus right predictions seems to deviate from the wrong predictions.

We can also examine the distribution of predictions using a density plot from the `lares` library.

```
# Density plot of predictions
mplot_density(tag = xgb_vali_outcome, score = xgbpred_v, subtitle = "Distribution of predictions in XGBoos
```



Ideally the density plot would have minimal overlap and the cumulative curves should be well separated. As we can observe there are quite a bit of overlap in the distribution from this model indicating that the model misclassifies many patients.

4.6.2 SHAP global feature importance

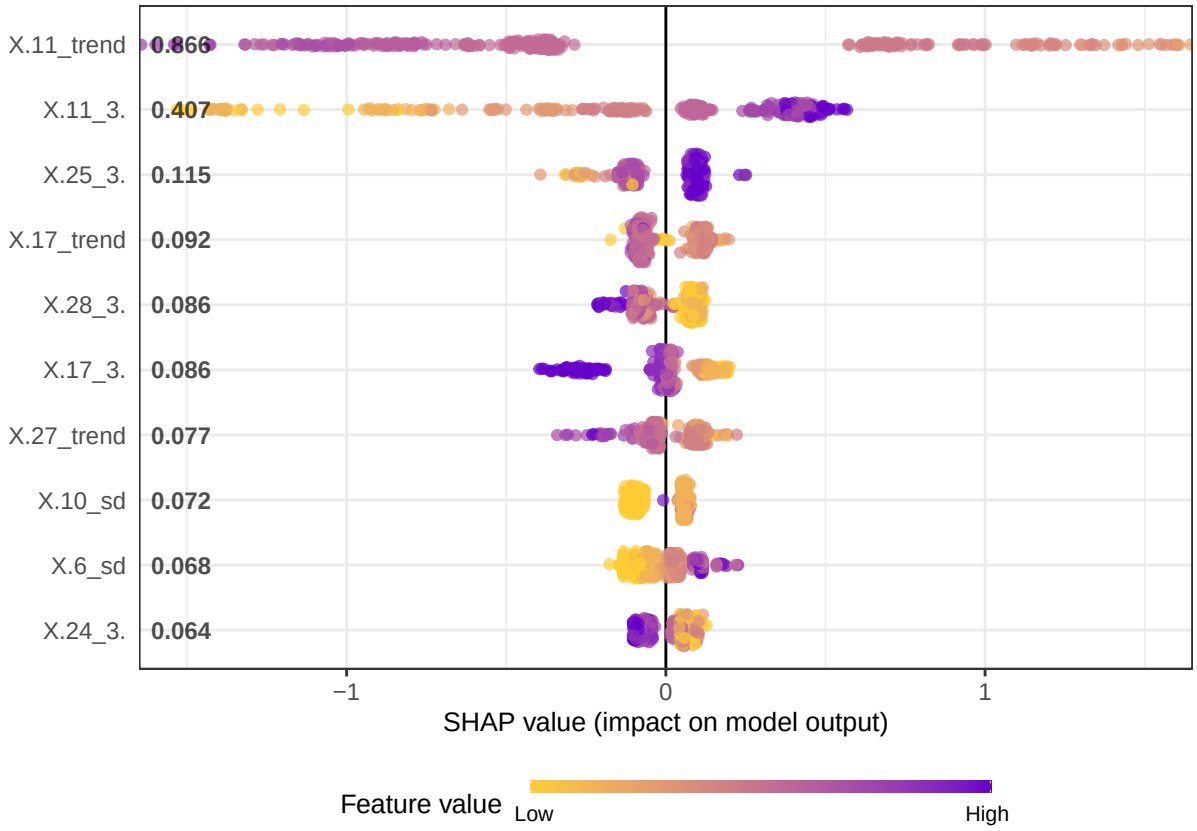
The top 10 model feature importance. This plot shows the relative impact on model prediction of each feature for each observation (patient).

```
if(!require(SHAPforxgboost)) install.packages("SHAPforxgboost", repos = "http://cran.us.r-project.org")

# Validation set: SHAP values and ranked features by mean|SHAP|
if(!require(SHAPforxgboost)) install.packages("SHAPforxgboost", repos = "http://cran.us.r-project.org")
shap_values_v <- shap.values(xgb_model = xgb_validation, X_train = as.matrix(xgb_valid))

# Validation set: Ranked features by mean |SHAP|
shap_values_v$mean_shap_score
```

```
##   X.11_trend      X.11_3.      X.25_3.   X.17_trend      X.28_3.      X.17_3.
## 0.8657811402 0.4070933482 0.1151937478 0.0916271112 0.0860668400 0.0857641121
##   X.27_trend      X.10_sd       X.6_sd      X.24_3.       X.7_3.   X.26_trend
## 0.0772559610 0.0721051433 0.0678023005 0.0642438832 0.0597639064 0.0501699041
##      X.11_sd      X.26_3.      X.29_3.   X.22_trend      X.20_sd   X.14_trend
## 0.0429784663 0.0419747018 0.0409502279 0.0406296784 0.0341536082 0.0322005507
##   X.12_trend    X.4_trend      X.21_sd          age      X.26_sd      X.27_sd
```

```
## 0.0303760656 0.0293857571 0.0277226466 0.0277179316 0.0273461660 0.0236860634
##   X.24_trend       X.9_sd       X.5_sd      X.28_sd      X.20_3.      X.16_sd
## 0.0204426749 0.0195746075 0.0187806641 0.0184007852 0.0173733370 0.0163468786
##    X.9_trend   X.21_trend      X.10_3.       X.3_3.    X.8_trend      X.24_sd
## 0.0161745745 0.0160944054 0.0150392713 0.0149282687 0.0142254513 0.0134791536
##      X.19_3.   X.28_trend      X.27_3.      X.17_sd      X.12_sd      X.13_3.
## 0.0113457088 0.0109617850 0.0107752052 0.0097853939 0.0096521141 0.0095160014
##       X.4_sd       X.8_sd       X.9_3.    X.6_trend      X.29_sd       X.8_3.
## 0.0088042806 0.0085577615 0.0082744984 0.0081693360 0.0076918620 0.0076506229
##      X.23_sd   X.20_trend      X.12_3.       X.1_sd   X.16_trend       X.5_3.
## 0.0074223906 0.0071814024 0.0070022003 0.0063550221 0.0053762737 0.0053536803
##      X.15_sd      X.22_sd    X.7_trend      X.15_3.       X.3_sd       X.4_3.
## 0.0047882728 0.0043813458 0.0043070117 0.0039135831 0.0037581696 0.0035492074
##   X.29_trend   X.23_trend   X.25_trend   X.19_trend   X.18_trend       X.7_sd
## 0.0034502791 0.0025333292 0.0024998907 0.0024880376 0.0023587508 0.0023064213
##      X.14_sd   X.13_trend       X.2_3.      X.25_sd    X.2_trend      X.23_3.
## 0.0022496618 0.0020050308 0.0016119456 0.0015089624 0.0014061669 0.0009320338
##        sex.F        sex.M        sex.0       X.1_3.      X.14_3.      X.16_3.
## 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
##      X.18_3.      X.21_3.      X.22_3.       X.6_3.    X.1_trend   X.10_trend
## 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
##   X.15_trend    X.3_trend    X.5_trend      X.13_sd      X.18_sd      X.19_sd
## 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
##       X.2_sd
## 0.0000000000
```

```
# Validation set: Calculate SHAP values and plot summary
shap_long_v <- shap.prep(xgb_model = xgb_validation, X_train = as.matrix(xgb_valid), top_n = 10)
shap.plot.summary(shap_long_v, x_bound = 1.5, dilute = 1)
```



As we see in the plot above the trend for item 11 is the most important feature, not surprisingly. High values for the trend (indicating rising values over the three sessions) is pushing the predictions in the negative direction (towards predicting no treatment effect) and low values are pushing the predictions in the positive direction. The second most important feature is the reported level of item 11 in session 3. Somewhat counter intuitively this has opposite polarity of the trend, meaning higher values push model towards predicting treatment effect and lower values towards no treatment effect. This is probably a result of the outcome labeling process. Patients with very low initial scores on item 11 have less probability of getting an even lower score in the final session and thus less probability of being labeled with treatment effect. This is a flaw in the outcome labeling due to using a single item as outcome label and is noted as a bias to reduce in future work.

The rest of the features have more or less dichotomous effects, but we can see a mixed effect on individual observations, e.g. for the feature X.10_sd where a low value apparently can push the prediction any of the two ways for individual patients.

The most interesting and important items are 25, 17, 28 and 27. Let's take a look at these items to see if that makes sense.

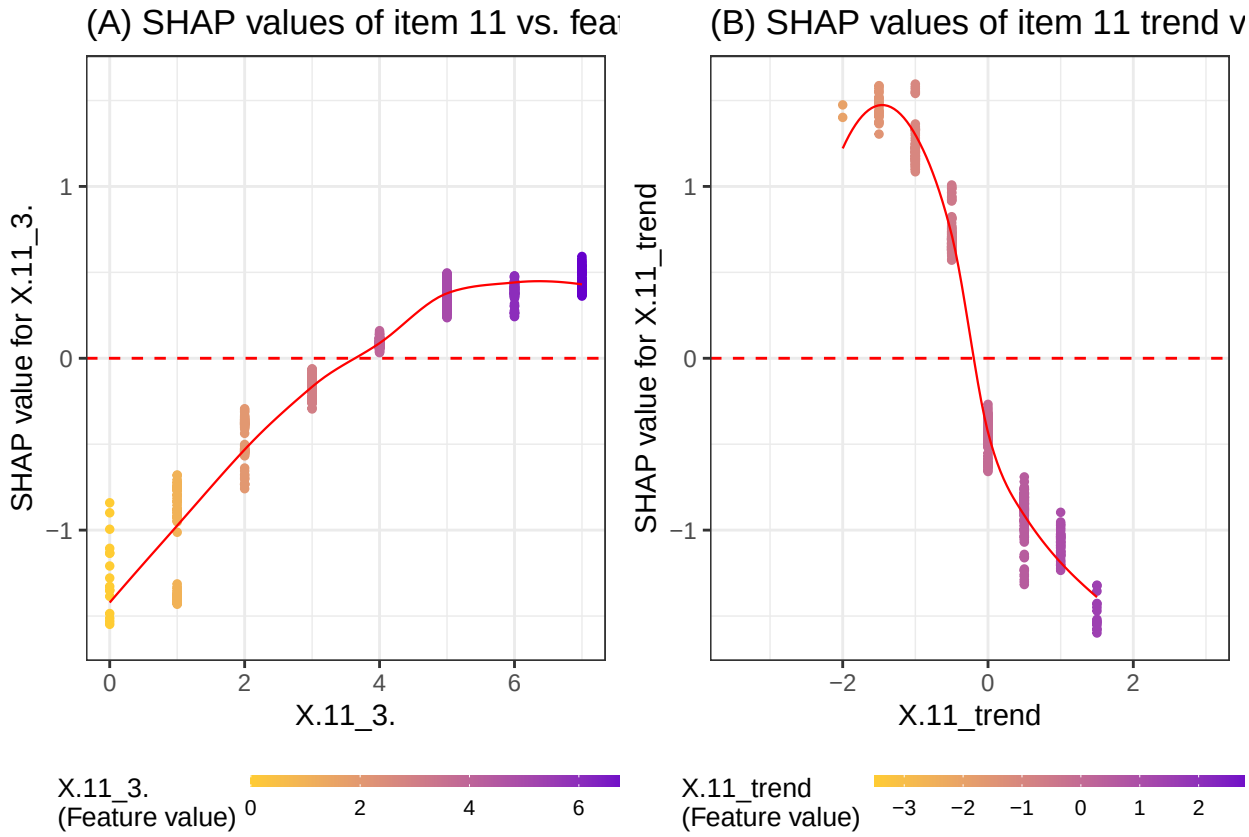
```
# Inspecting items 25, 17, 28 and 27
pander(item_df[c(25,17,28,27), c(1,3)])
```

itemId	question
25	Now I understand what I need to do or work on to get better
17	I am feeling depressed.
28	I am worthless.

itemId	question
27	I spend excessive time thinking about food and planning my meals.

Since item 11 score in session 3 and trend unsurprisingly have large impact on prediction it is worth to examine these a bit further.

```
# Exploring the impact of item 11 features in model
g1 <- shap.plot.dependence(data_long = shap_long_v,
                           x = 'X.11_3.',
                           y = 'X.11_3.',
                           color_feature = 'X.11_3.') +
  geom_hline(yintercept = 0, linetype="dashed", color = "red") +
  ylim(-1.6, 1.6) +
  ggtitle("(A) SHAP values of item 11 vs. feature values of item 11")

g2 <- shap.plot.dependence(data_long = shap_long_v,
                           x = 'X.11_trend',
                           y = 'X.11_trend',
                           color_feature = 'X.11_trend') +
  geom_hline(yintercept = 0, linetype="dashed", color = "red") +
  ylim(-1.6, 1.6) +
  ggtitle("(B) SHAP values of item 11 trend vs. feature values of item 11 trend")

gridExtra::grid.arrange(g1, g2, ncol = 2)
```

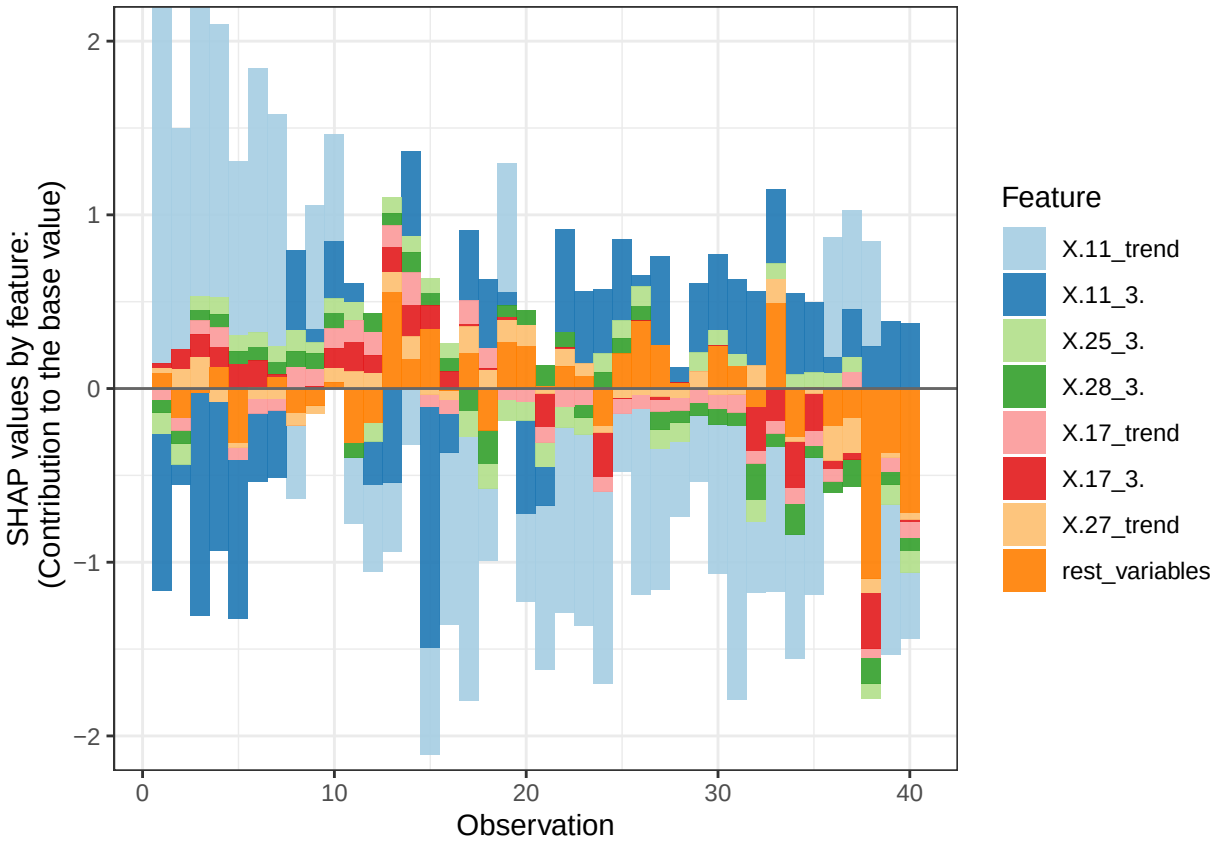


The impact of the trend is clearly the largest of the two features. Low values have a high positive impact and high values have a high negative impact. For the session 3 score for item 11 the most extreme low scores clearly have the highest impact.

4.6.3 SHAP patient feature importance

The impact of features on model prediction on a patient level is highly interesting and vital for clinical application. Let's start by looking at the 40 first patients and see how feature impact varies. I will only plot the top 7 features to include the ones I consider most clinically relevant.

```
# Force plot for the 40 first patients. Showing top 7 features by importance
plot_data <- shap.prep.stack.data(shap_contrib = shap_values_v$shap_score[1:40], top_n = 7)
shap.plot.force_plot(plot_data, zoom_in = FALSE, y_parent_limit = c(-2,2))
```



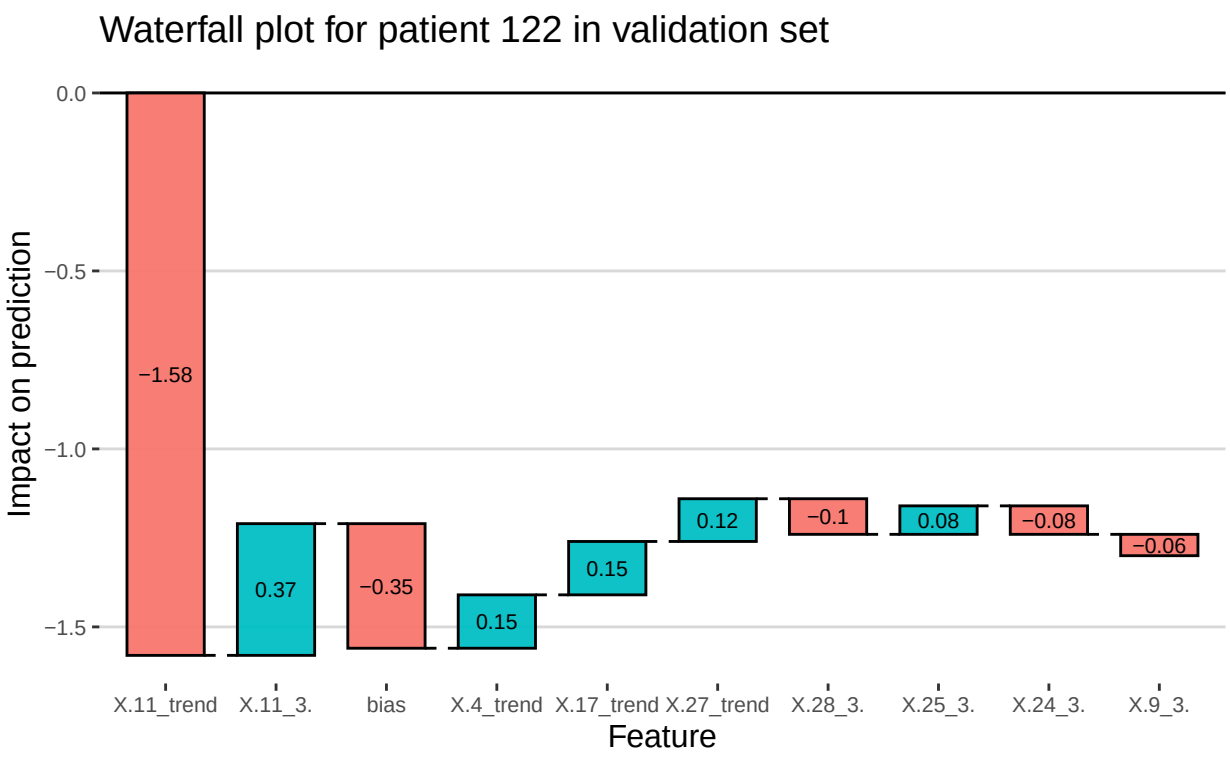
To be able to visualize the effects on an individual level I create a function that uses the `waterfall` library to make a waterfall plot.

```
if(!require(ggalluvial)) install.packages("ggalluvial", repos = "http://cran.us.r-project.org")
if(!require(waterfalls)) install.packages("waterfalls", repos = "http://cran.us.r-project.org")

plot_waterfall <- function(x, top_n = NULL){
  if (is.null(top_n)) top_n <- dim(x)[2]
  shap_long <- pivot_longer(shap_values_v$shap_score[x,],
                            cols = 1:ncol(shap_values_v$shap_score),
                            names_to = "feature",
```

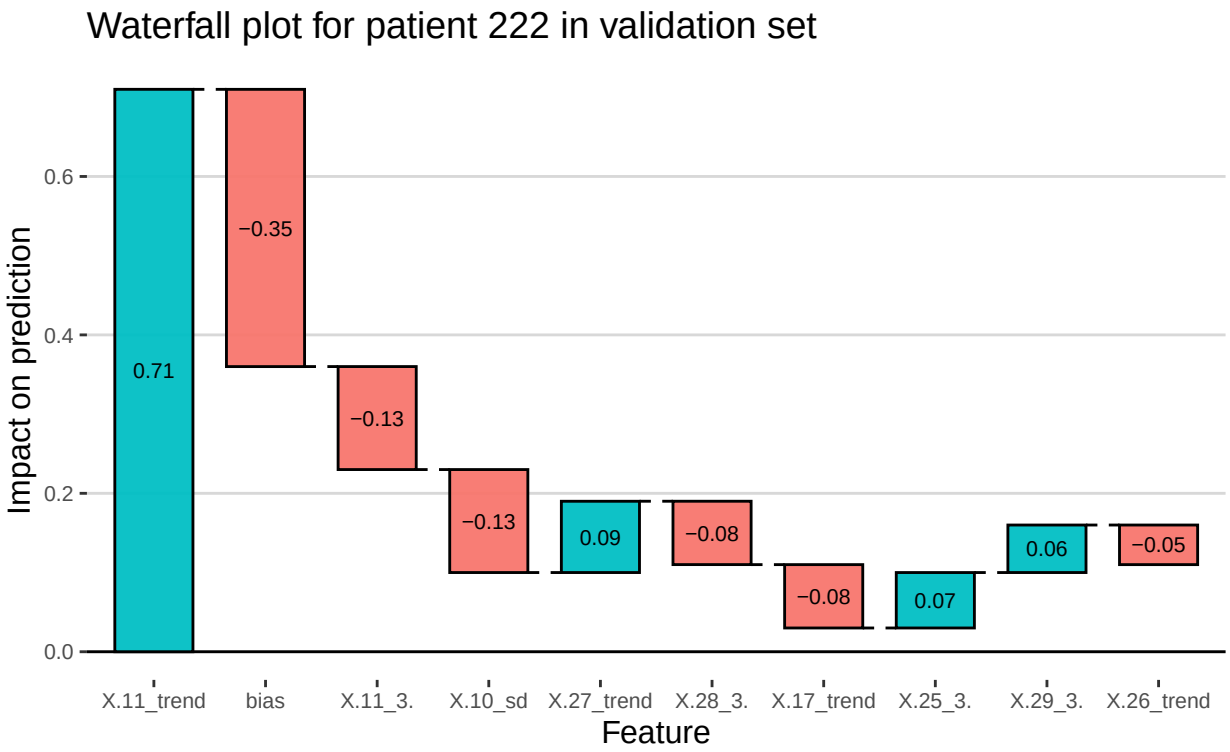
```
                            values_to = "value")
  bias <- tibble(feature = "bias", value = shap_values_v$BIAS0$BIAS)
  shap_long <- shap_long %>% bind_rows(bias) %>%
    select(feature, value) %>%
    arrange(desc(abs(value))) %>%
    head(top_n)

  shap_long %>%
  mutate(value = round(value, 2)) %>%
  waterfall() +
    ggtitle(paste("Waterfall plot for patient", x, "in validation set")) +
    xlab("Feature") +
    ylab("Impact on prediction") +
    theme(axis.text=element_text(size=8)) +
    theme_hc()
}
```

I can now visualize any patient using this simple function. Let's take a look at a few random patients.

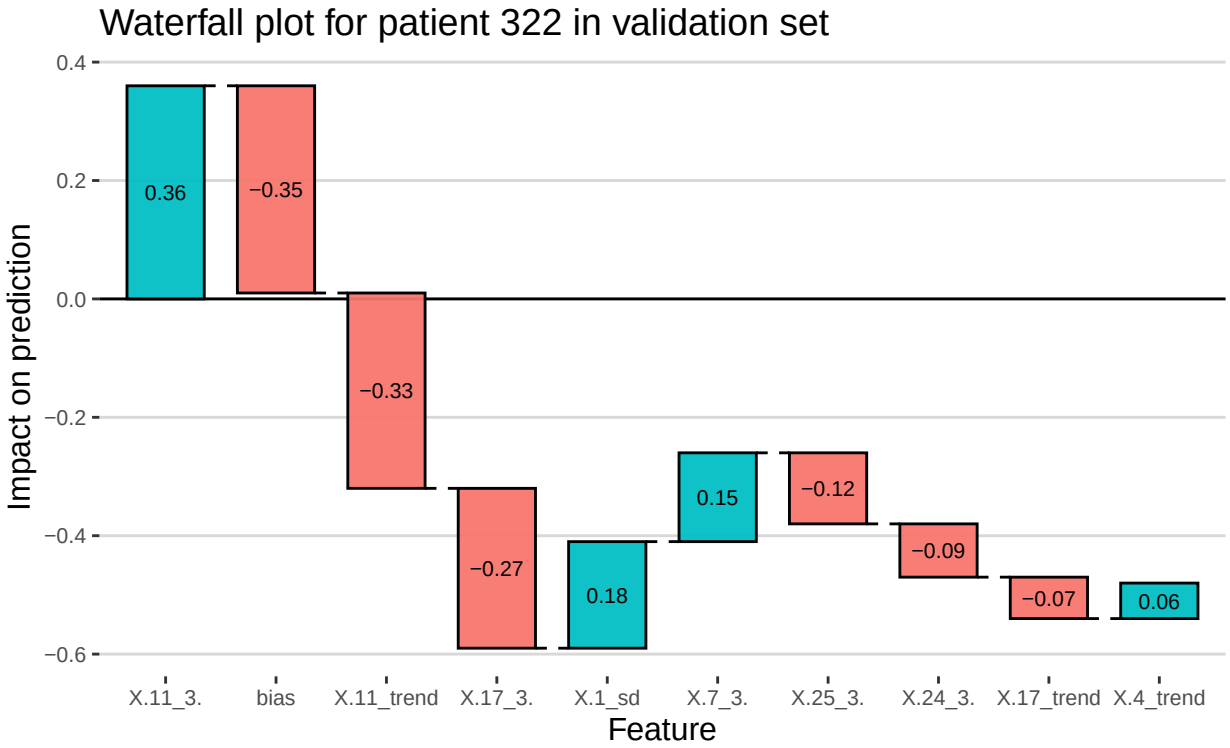
```
plot_waterfall(122, 10)
```



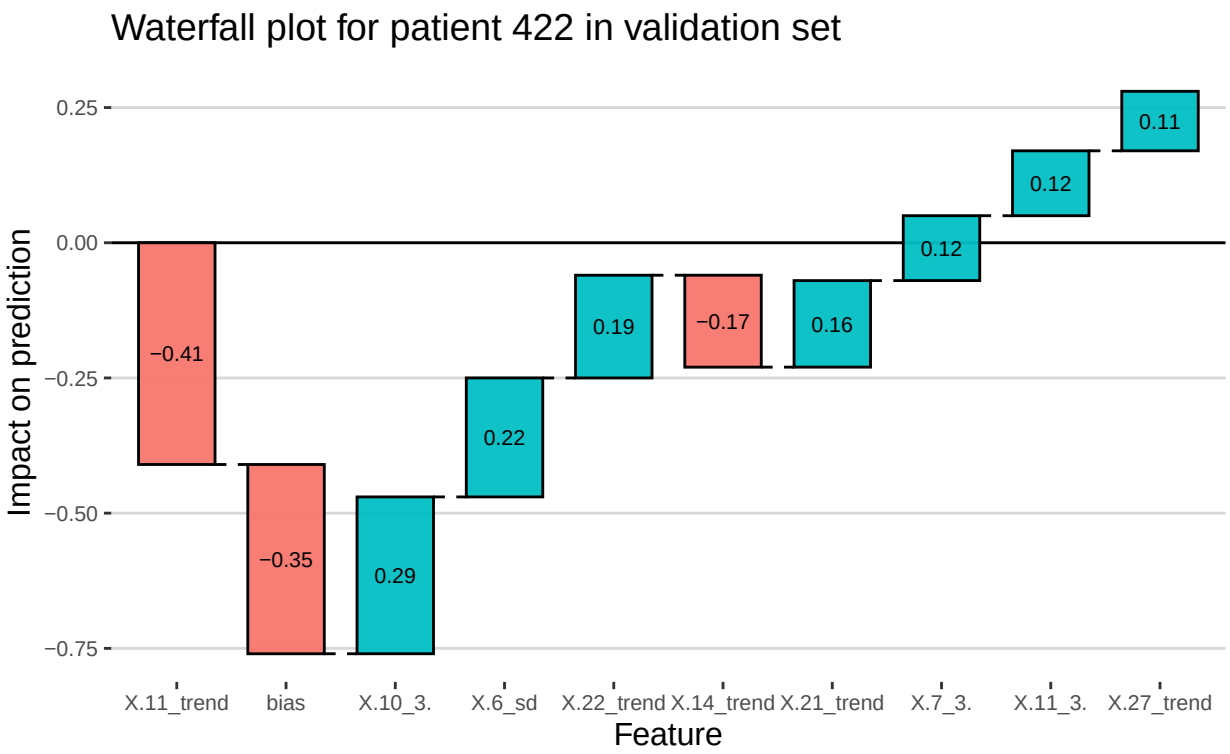

```
plot_waterfall(222, 10)
```




```
plot_waterfall(322, 10)
```




```
plot_waterfall(422, 10)
```



Features are ordered by their magnitude of impact, showing top 10 features. The most important features varies between patients. A new feature can be observed in these plots, the bias. This is similar to the intercept in a linear model. Clearly, the trend of item 11 has a large impact on these predictions. For patient 122 the negative impact of the trend is so high that it can not be outweighed by the other features. For patient 222 the positive impact of the item 11 trend is substantial, but it is nearly cancelled by the remaining features. For patient 322 the negative impact of item 11 trend is less than the bias, but other features contribute to the negative prediction. For patient 422 the negative impact of item 11 trend and bias is outweighed by positive impacts from other features.

Chapter 5

Conclusions and future work

Training machine learning models on the Norse Feedback dataset has been both challenging and rewarding. Handling raw real world data required me to clean and wrangle data and think carefully through datastructure for further analysis. A particular challenge was presented in the missing labeling of the dataset. I am not entirely confident that I have found the best approach to labeling the data, but as an initial exercise and demonstration of machine learning modeling on this dataset it will suffice. I will continue to explore other labeling options and lean towards using an ensemble technique to label data more representatively across all patients, also those that start with low values on item 11.

Another challenge was the missingness of data in the dataset that became evident as the data was pivoted into a wide/matrix format. This problem introduced me to the theories underlying handling missing data and I chose to impute 0s for missing values to be able to train the models since data is mainly missing by design. The `XGBoost` algorithm is robust when it comes to missing values and would have been able to create a model without any imputation.

Finally, a goal of this project was to train a model that was interpretable and explainable. Fortunately various libraries exist for R that assist in visualizing and explaining data from a `XGBoost` model. I have experimented with various visualizations to give readers an intuition of how the model makes predictions.

The final validation results of the trained `XGBoost` model are not very impressive in terms of accuracy (0.7138554) and ROC AUC (0.7689765), but the model could provide clinical insights that might help therapists in their work with individual patients. Future work may improve model performance.