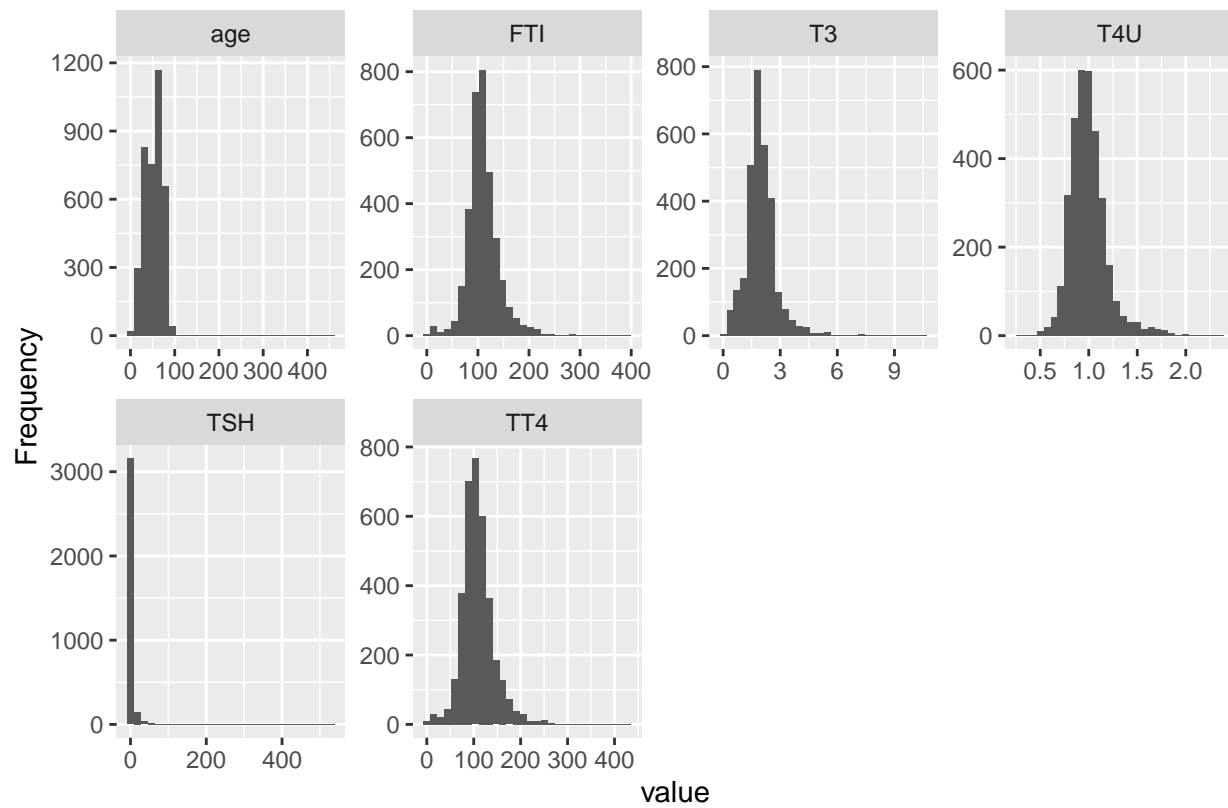


PD 1

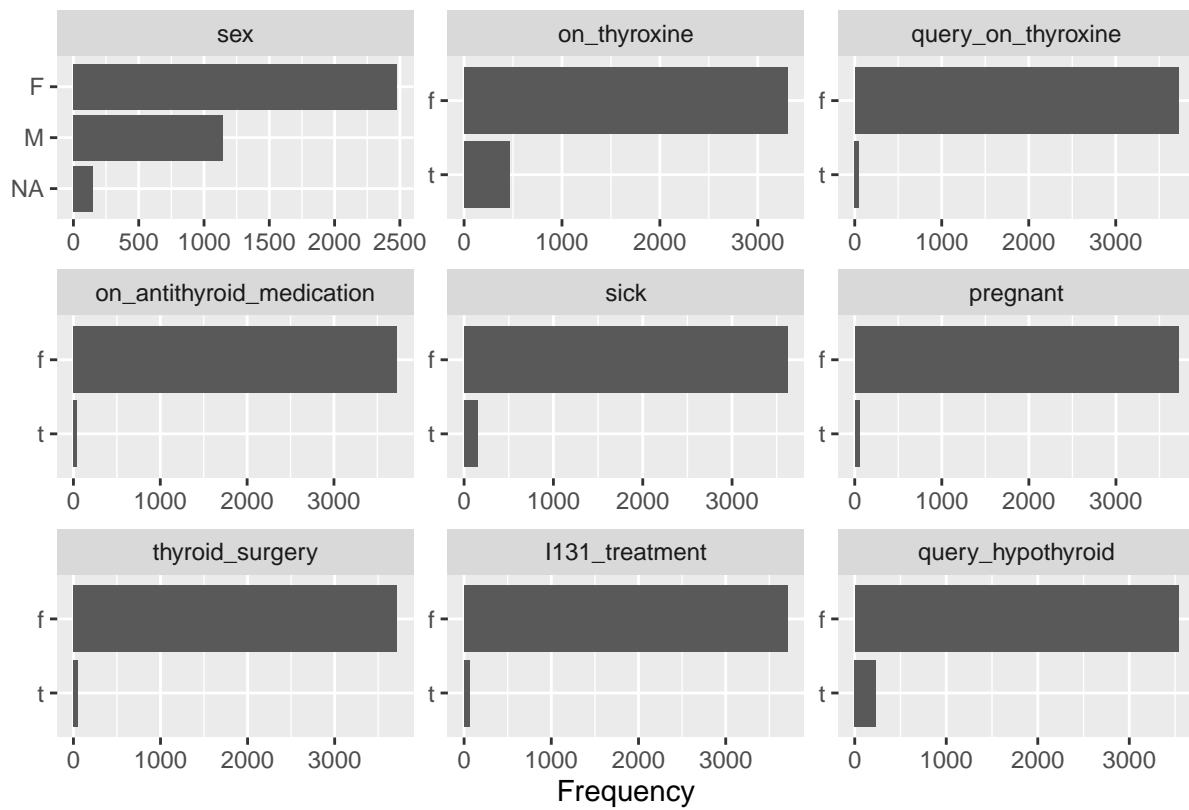
Witold Merkel

15 04 2020

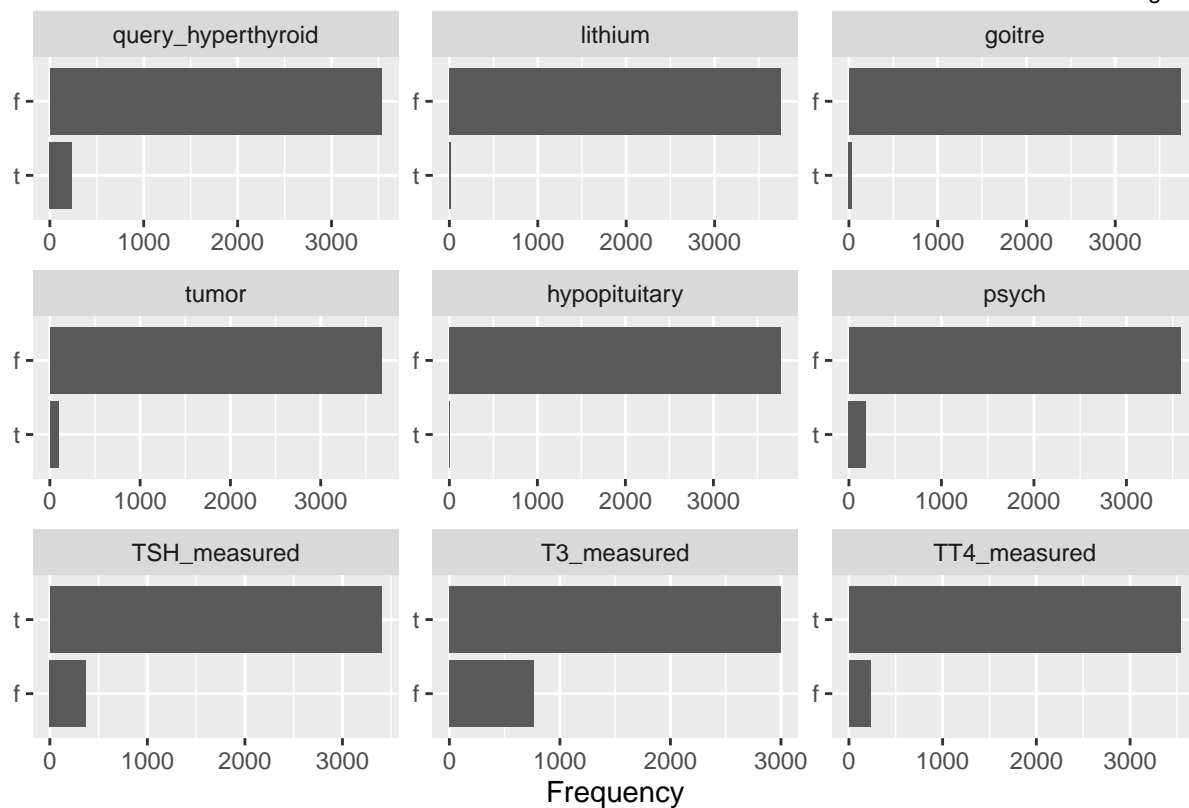
```
DataExplorer::plot_histogram(dataset_raw)
```



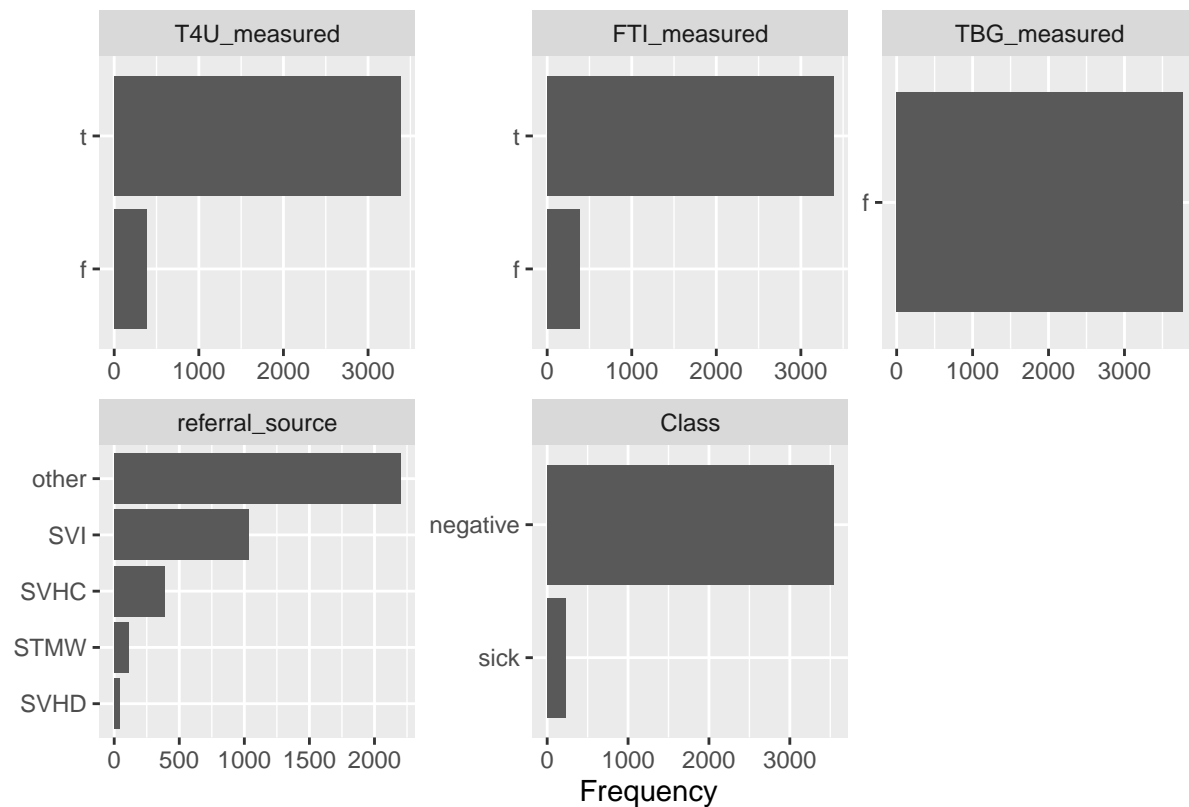
```
DataExplorer::plot_bar(dataset_raw)
```



Page 1



Page 2



Page 3

Preprocessing

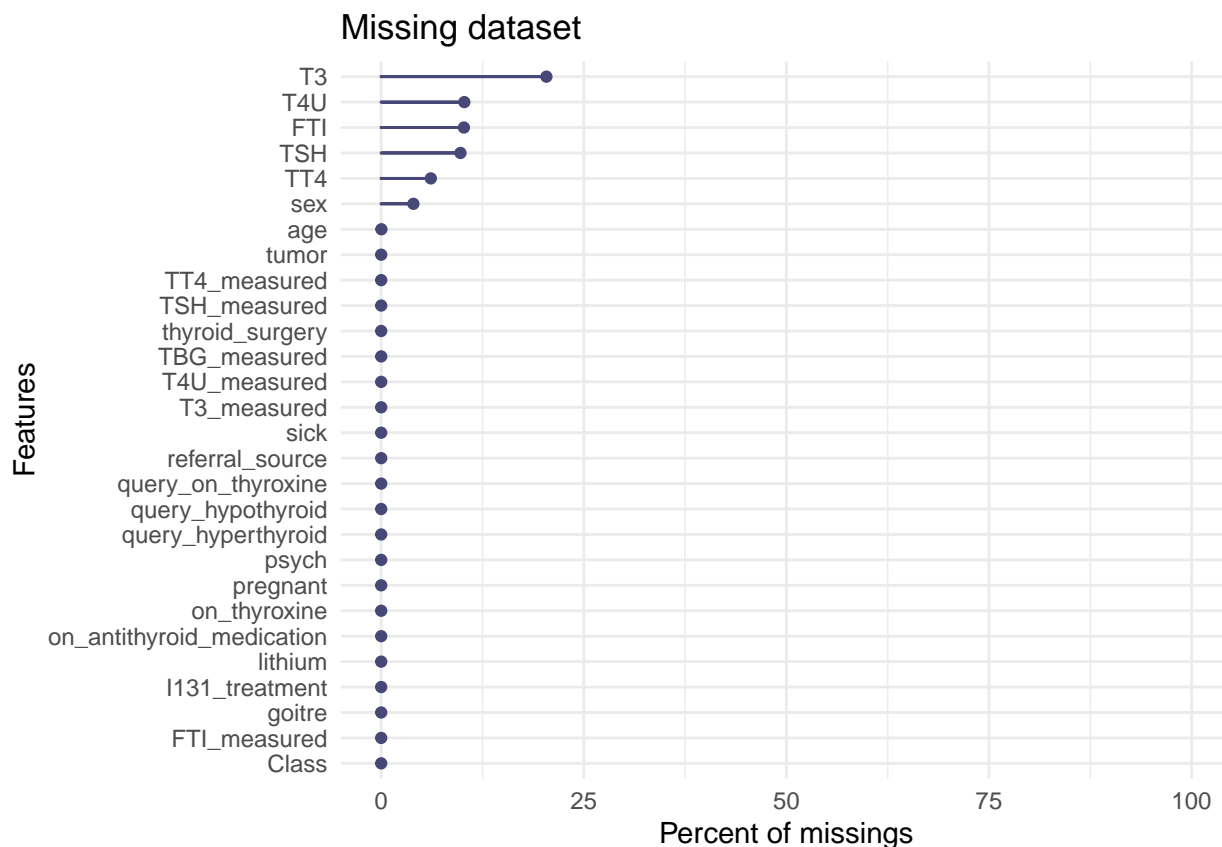
I tried different things to make my model a better one:

- logarithmic transformation of selected columns,
- getting rid of lines with missing elements,
- different types of imputing missing data,
- deleting columns.

None of those things helped me with my score, so I decided to only delete the empty column and the “hypopituitary” column.

```
dataset_raw <- dataset_raw %>% select(-c("TBG", "hypopituitary"))
dataset_raw$Class <- as.factor(ifelse(dataset_raw$Class == 'sick', 1, 0))

gg_miss_var(dataset_raw,
             show_pct = TRUE) +
  ylim(0, 100) +
  labs(title = "Missing dataset",
       x = "Features",
       y = "Percent of missings")
```



Dividing data

Division that was given to us.

```
index <- read.table("index.txt")$x
train <- dataset_raw[index,]
test <- dataset_raw[-index,]
```

Basic model and its performance

After running some test I decided to go for decision trees. I choose the mlr implementation with rpart.

AUC - 5 cv on training data

```
task <- mlr::makeClassifTask(id = "1", data = train, target = "Class", positive = "1")
dec_tree <- mlr::makeLearner(cl = "classif.rpart", predict.type = "prob")

rd <- makeResampleDesc("CV", iters=5)
benchmark(learners = list(dec_tree), tasks = task, resamplings = rd, measures = auc)
```

```
## Task: 1, Learner: classif.rpart
```

```
## Resampling: cross-validation
```

```
## Measures:          auc
## [Resample] iter 1: 0.9689185
## [Resample] iter 2: 0.9196038
## [Resample] iter 3: 0.9584288
## [Resample] iter 4: 0.9402802
## [Resample] iter 5: 0.9701401
##
## Aggregated Result: auc.test.mean=0.9514743
##
##   task.id   learner.id auc.test.mean
## 1         1 classif.rpart      0.9514743
```

AUPRC - on test data

```
model1 <- train(dec_tree, task)
"AUPRC"
```

```
## [1] "AUPRC"
```

```
pred_c(model1, test)
```

```
## [1] 0.7939315
```

I think those scores aren't bad, but I would like to do something more, so it doesn't seem that I didn't do anything.

Tunning of the hyperparameters

Here I would like to present my excellent model, which I got by tuning the parameters of a basic rpart model, with random search method, which is better than searching through grid.

```
params <- makeParamSet(
  makeIntegerParam("minsplit", lower = 5, upper = 100),
  makeIntegerParam("minbucket", lower = 2, upper = 100),
  makeNumericParam("cp", lower = 0.001, upper = 0.8),
  makeIntegerParam("maxdepth", lower = 1, upper = 30))

ctrl <- makeTuneControlRandom(maxit = 5000)
classification_best <- tuneParams("classif.rpart", task = task, resampling = rd,
                                par.set = params, control = ctrl, show.info = FALSE,
                                measures = list(acc))

classification_best
```

```
## Tune result:
## Op. pars: minsplit=16; minbucket=2; cp=0.0104; maxdepth=17
## acc.test.mean=0.9867390
```

```

rpart <- makeLearner("classif.rpart", predict.type = "prob",
  par.vals = list("cp" = classification_best$x$cp,
    "maxdepth" = classification_best$x$maxdepth,
    "minsplit" = classification_best$x$minsplit,
    "minbucket" = classification_best$x$minbucket))
rf_hyper <- train(rpart, task)

```

Above we can see the optimal found parameters for our problem. After learning them I made a better model.

Performance of the better model

Let's check how much better our new model is.

AUC - 5 cv on training data

```

benchmark(learners = list(rpart), tasks = task, resamplings = rd, measures = auc)

## Task: 1, Learner: classif.rpart
## Resampling: cross-validation
## Measures:          auc
## [Resample] iter 1:  0.9358054
## [Resample] iter 2:  0.9667516
## [Resample] iter 3:  0.9595307
## [Resample] iter 4:  0.9858640
## [Resample] iter 5:  0.9319549
##
## Aggregated Result: auc.test.mean=0.9559813
##
##   task.id   learner.id auc.test.mean
## 1         1 classif.rpart    0.9559813

```

AUPRC - on test data

```

"AUPRC"

## [1] "AUPRC"
pred_c(rf_hyper, test)

## [1] 0.7991405

```

We can see that the model is better when it comes to AUC, but its performance according to AUPRC is worst, so we have to decide which measure is more important to us and act accordingly.