Sick dataset analysis

Tomasz Makowski

29 04 2020

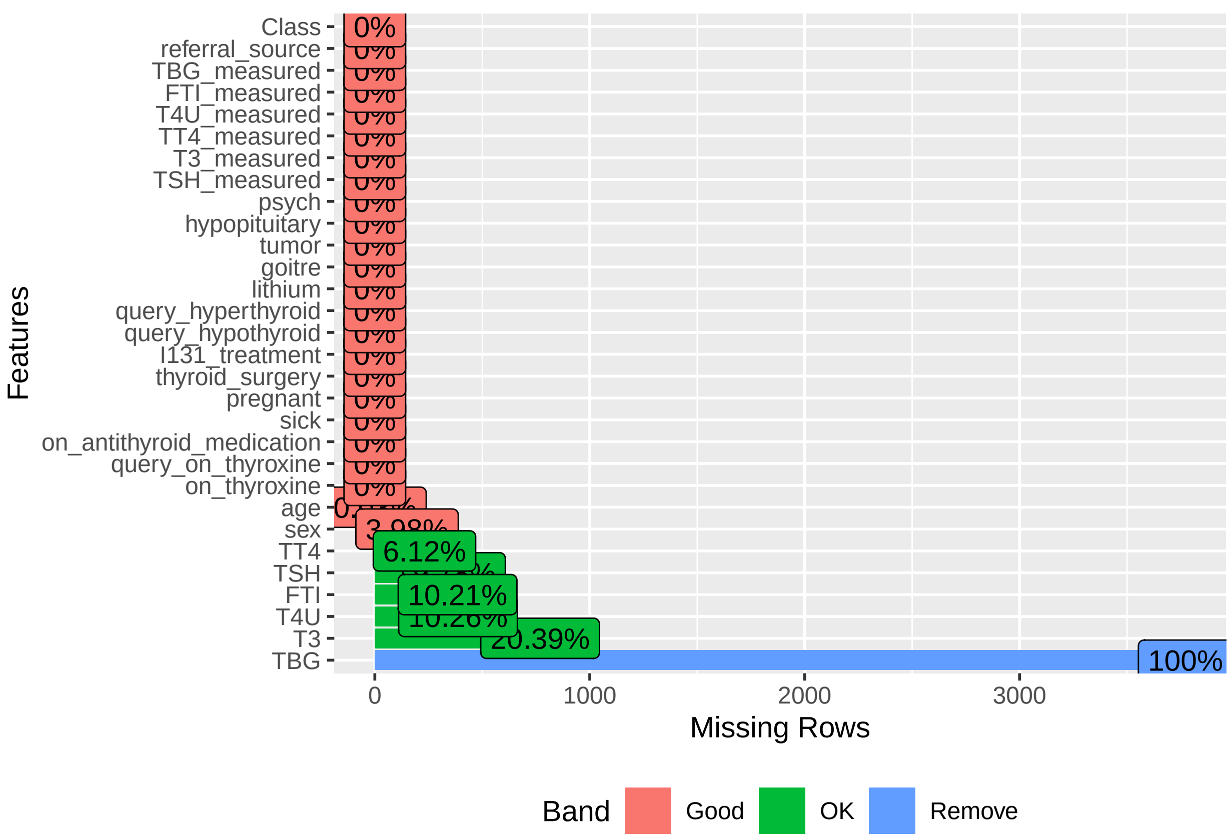
Introduction

The object of this analyse is to predict if person is sick or not. First, we can see the basic introduction of dataset.

```
introduce(data = dataset_raw)
```

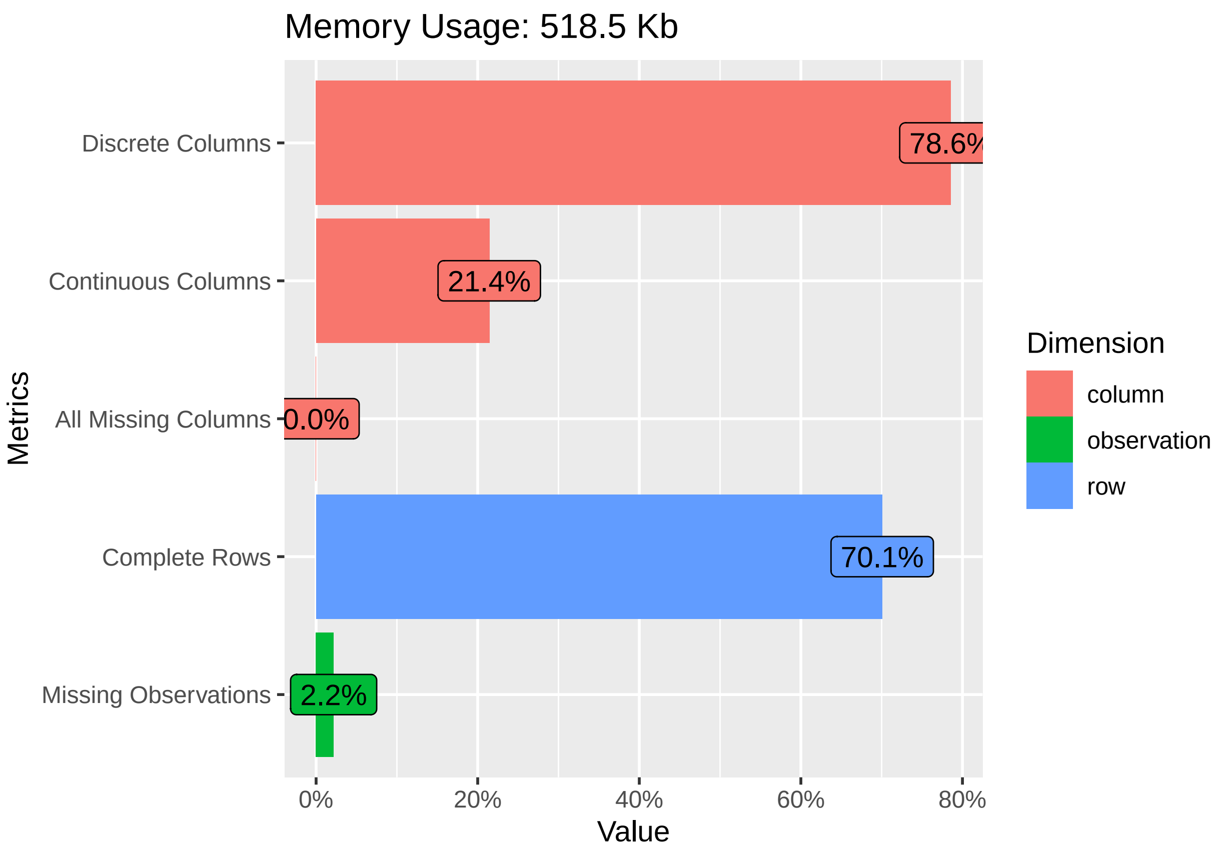
```
##   rows columns discrete_columns continuous_columns all_missing_columns
## 1 3772      30               23                  6                   1
##   total_missing_values complete_rows total_observations memory_usage
## 1                 6064             0             113160       577048
```

```
plot_missing(dataset_raw)
```



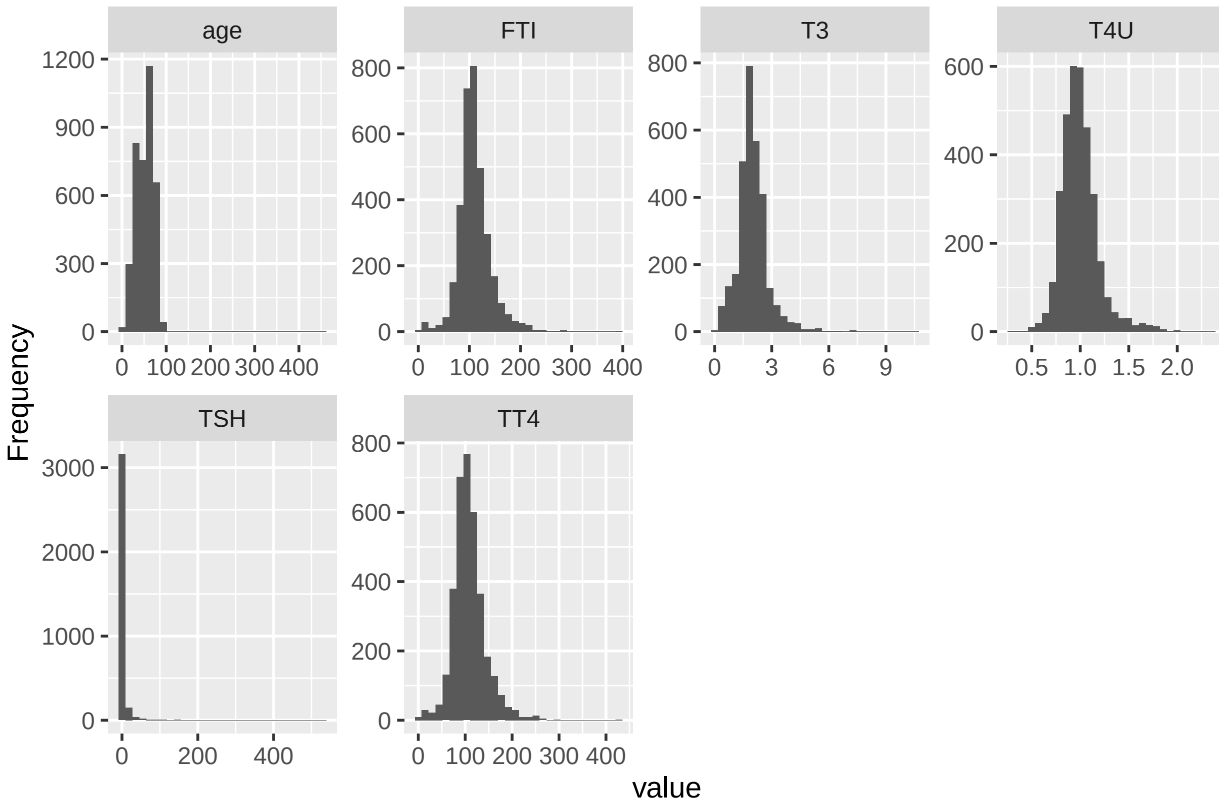
There is one column with all missing values so we can remove this column and see more information about dataset without this column. Few columns have missing values. We should drop those columns, drop proper rows or impute data.

```r
dataset <- dataset_raw %>%
  # drop 'TBG' and 'TBG_measured' - it is an empty column
  select(-c(TBG, TBG_measured))
plot_intro(dataset)
```

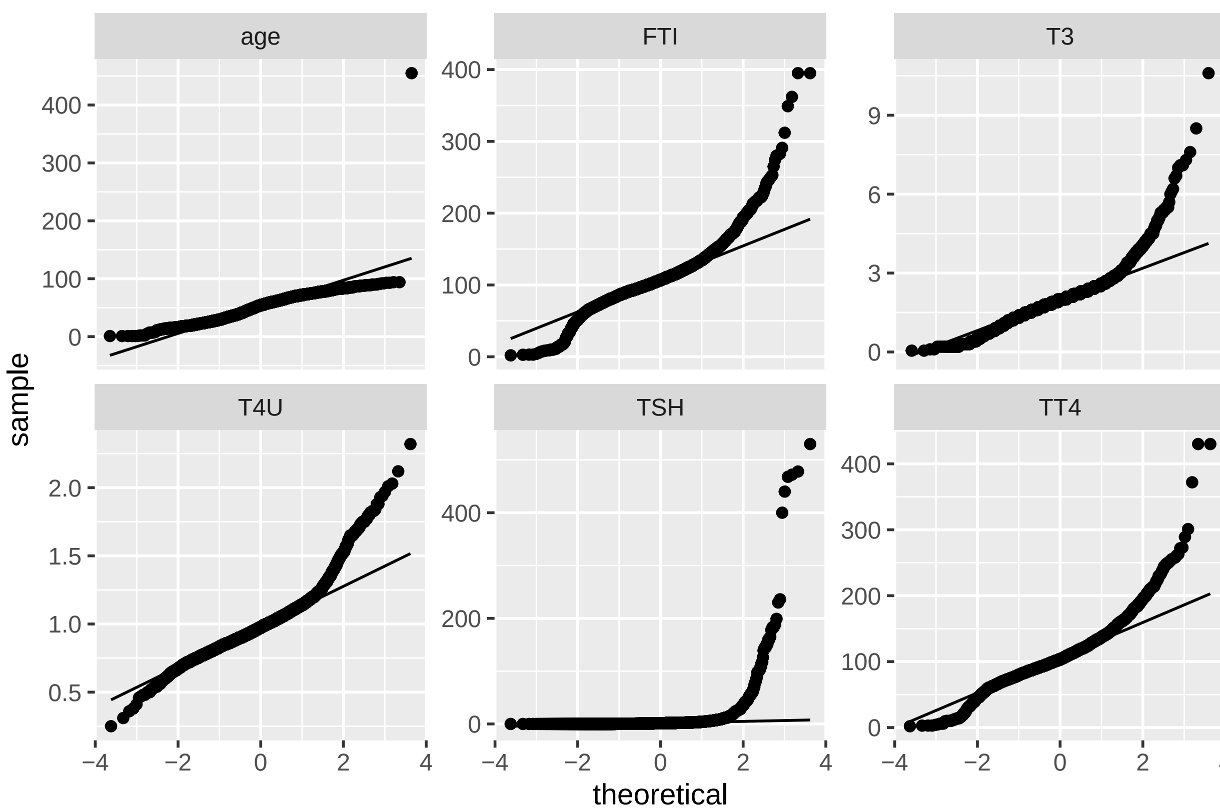


As we can see missing data are usually in the same rows. So dropping all columns is a bad idea.

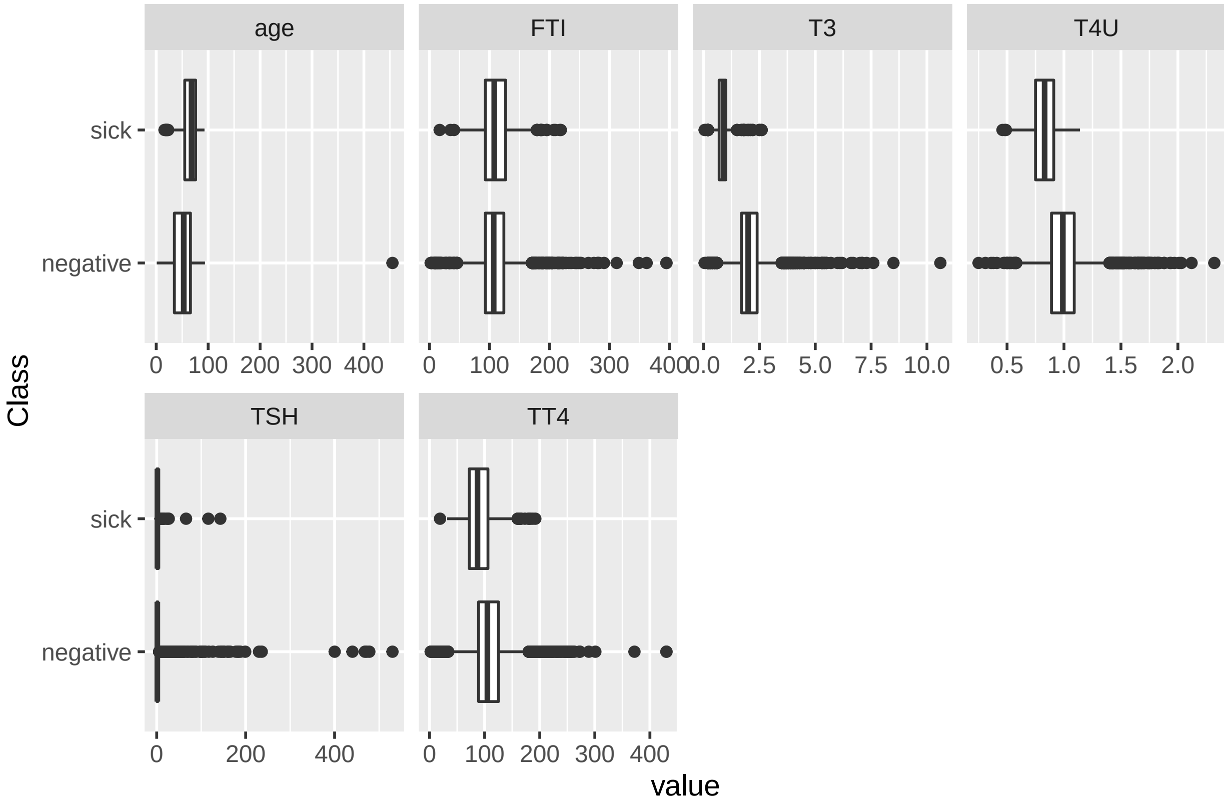
```r
plot_histogram(dataset)
```




```
plot_qq(dataset)
```

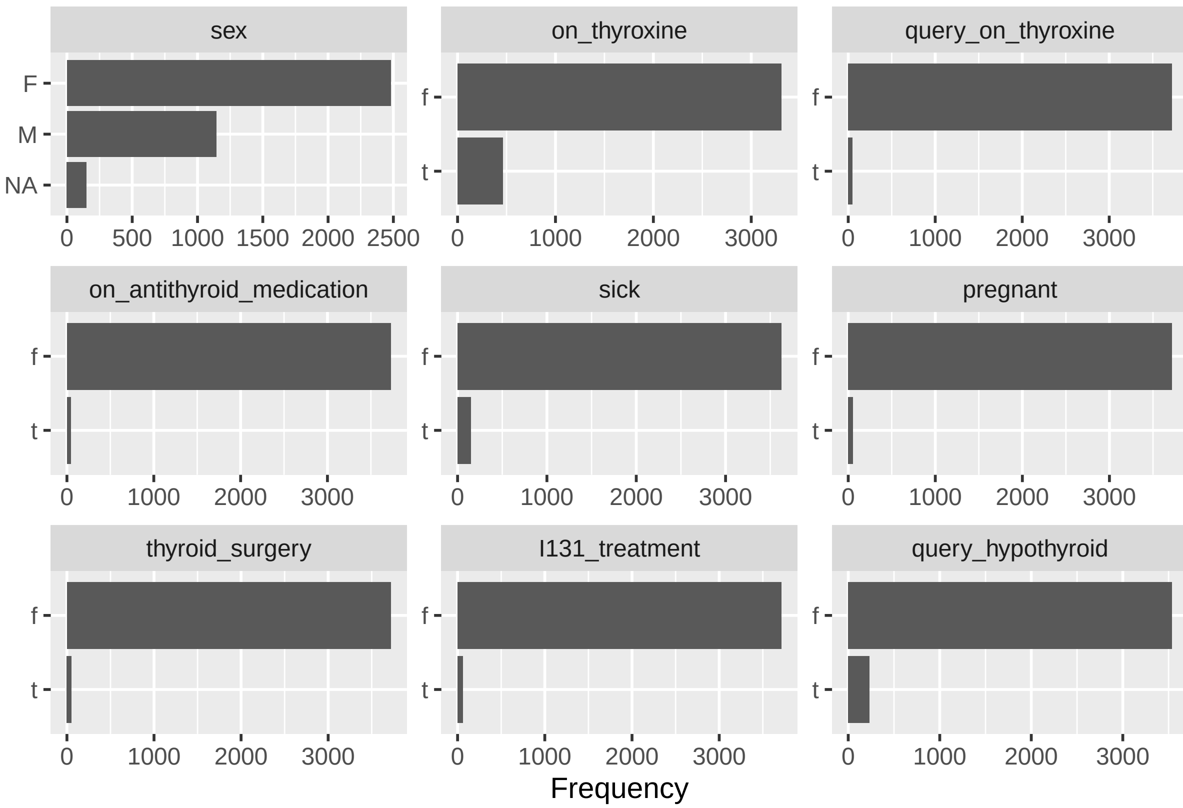


```
plot_boxplot(dataset, by="Class")
```

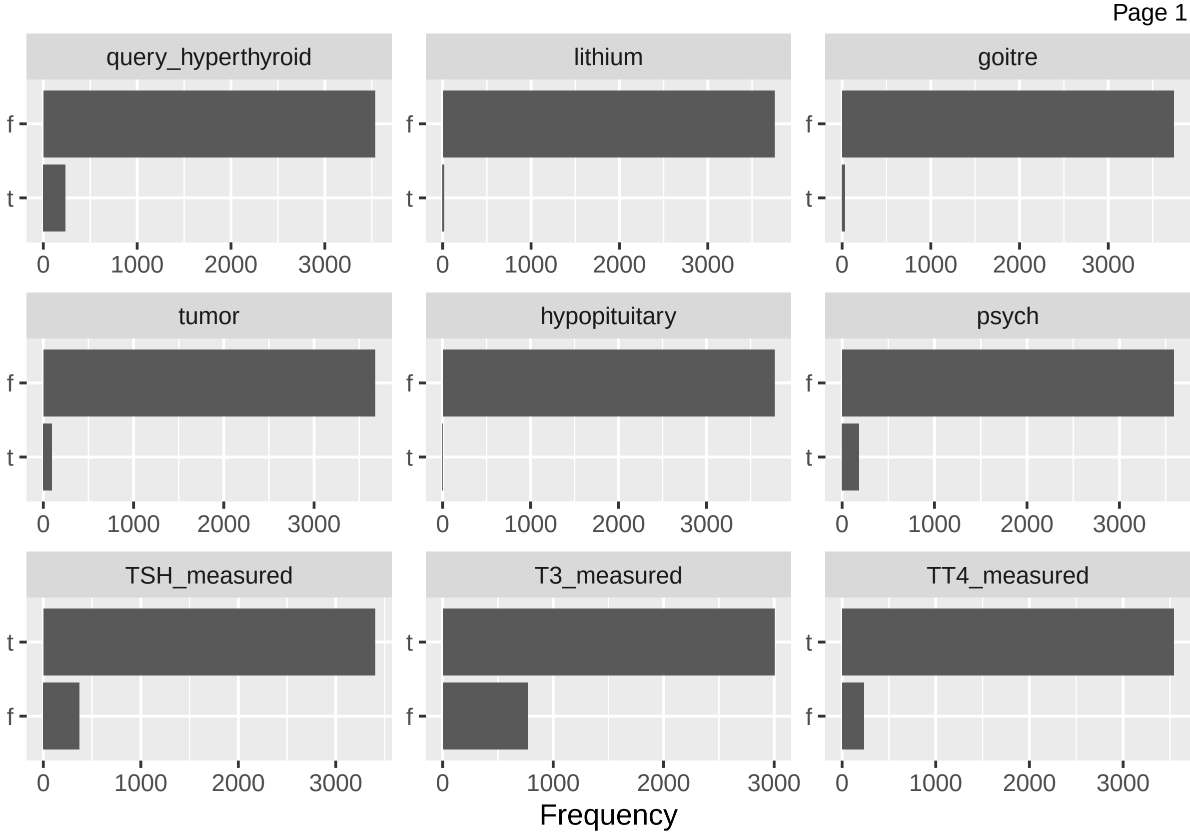


There are some values that should be removed - especially age around 400. Also in TSH, FTI and TT4 we can try to remove few outliers and replace them as the mean value. Also it could be good to change TSH by appling logarithm since there are many values near 0 and outliers are only bigger than 0.

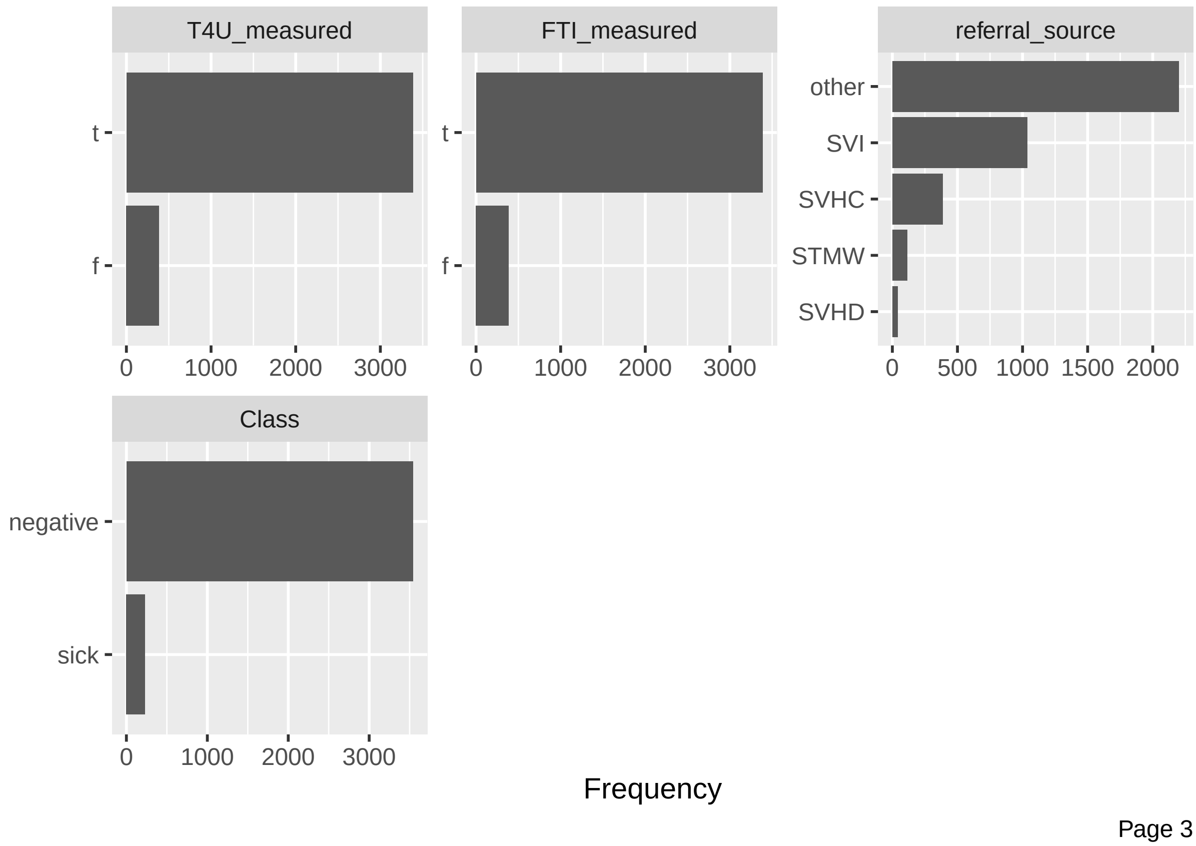
```
plot_bar(dataset)
```


sex
on_thyroxine
query_on_thyroxine
on_antithyroid_medication
sick
pregnant
thyroid_surgery
I131_treatment
query_hypothyroid
F
M
NA
f
t
0
500
1000
1500
2000
2500
3000
Frequency

Page 1


query_hyperthyroid
lithium
goitre
tumor
hypopituitary
psych
TSH_measured
T3_measured
TT4_measured
f
t
0
1000
2000
3000
Frequency

Page 2



There are some columns with only few observation with one of the categories. Below are their number of occurence and how much it tell about sick class.

```r
sum(dataset_raw$hypopituitary == "t")
```

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

```r
sum(dataset_raw$lithium == "t")
```

```
## [1] 18
```

```r
sum(dataset_raw$goitre == "t")
```

```
## [1] 34
```

```r
sum(dataset_raw$on_antithyroid_medication == "t")
```

```
## [1] 43
```

```r
sum(dataset_raw$thyroid_surgery == "t")
```

```
## [1] 53
```

```r
sum(dataset_raw$referral_source == "SVHD")
```

```
## [1] 39
```

```r
sum(dataset_raw[dataset_raw$lithium == "t", "Class"] == "sick")
```

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

```r
sum(dataset_raw[dataset_raw$goitre == "t", "Class"] == "sick")
```

```
## [1] 2
```

```r
sum(dataset_raw[dataset_raw$on_antithyroid_medication == "t", "Class"] == "sick")
```

```
## [1] 0
```

```r
sum(dataset_raw[dataset_raw$thyroid_surgery == "t", "Class"] == "sick")
```

```
## [1] 0
```

```r
sum(dataset_raw[dataset_raw$referral_source == "SVHD", "Class"] == "sick")
```

```
## [1] 3
```

Based on additional data it can be good to remove hypopituitary, lithium, goitre, on_antithyroid_medication and thyroid_surgery columns because there are up to 53 occurences and only up to 2 sick people so those columns do not give any additional information but some algorithms can have problems with training models.

Preprocessing

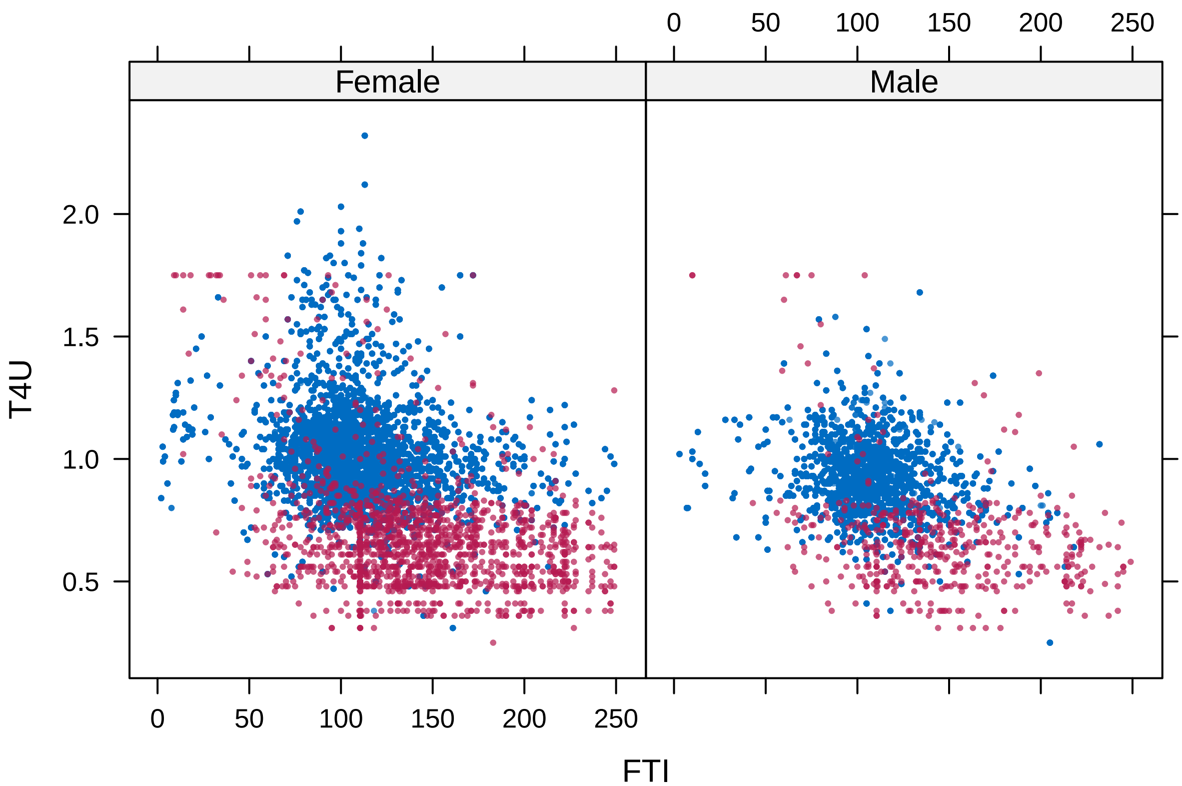
Because of first look for data we can remove values with probable mistakes during writing, then we can remove hypopituitary column. After that we can one hot encode categorical data to use in algorithms.

```r
# remove too big values - many written by mistake
dataset[dataset$age > 120 & (is.na(dataset$age) == FALSE), "age"] <- mean(dataset$age, na.rm = TRUE)
dataset[dataset$TT4 > 300 & (is.na(dataset$TT4) == FALSE), "TT4"] <- mean(dataset$TT4, na.rm = TRUE)
dataset[dataset$FTI > 250 & (is.na(dataset$FTI) == FALSE), "FTI"] <- mean(dataset$FTI, na.rm = TRUE)
dataset[dataset$TSH > 100 & (is.na(dataset$TSH) == FALSE), "TSH"] <- mean(dataset$TSH, na.rm = TRUE)

# drop column hypopituitary because there are very few values
dataset <- dataset %>%
  select(-c(hypopituitary, lithium, goitre, on_antithyroid_medication, thyroid_surgery))

# one_hot encoding
target <- dataset$Class
target <- data.frame(target = as.factor(as.numeric(target == "sick")))
observed <- select(dataset, -Class)
dummy <- dummyVars(" ~ .", observed)
data_ohe <- data.frame(predict(dummy, newdata = observed))
data_ohe <- data_ohe %>% select(-sex.M)
dataset <- cbind(target, data_ohe)
```

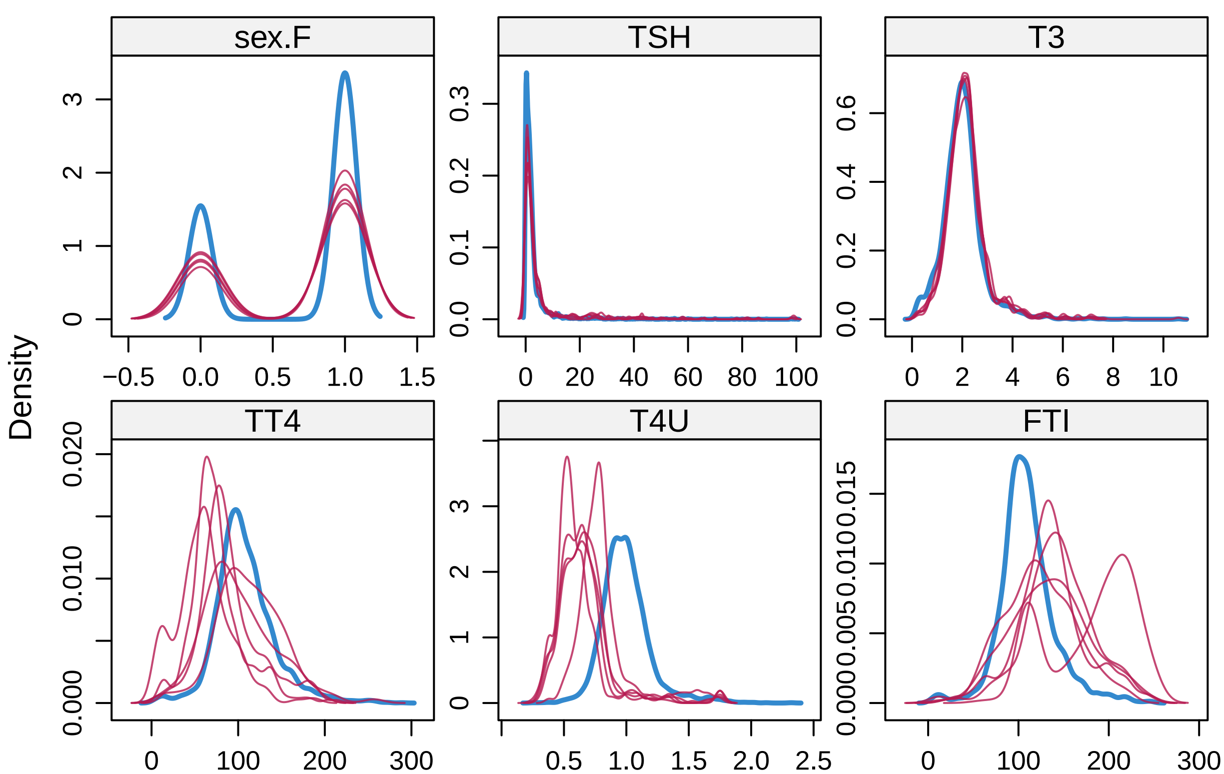
Then we can impute data. We will use mice package and we try to do it 5 times. The algorithm do it iteratively so in every time the imputed data will be diffrent. Then we can check which imputation gives the best results.

```r
set.seed(1221)
mice_imputes <- mice(data_ohe, m=5, maxit = 10)
```

```
xyplot(mice_imputes, T4U~FTI | ifelse(sex.F==TRUE, "Female", "Male"), pch = 20, cex = 0.4)
```




```
densityplot(mice_imputes)
```



```
datasets <- list()
for(i in 1:5) {
  data <- complete(mice_imputes, i)
  data <- cbind(target, data)
  datasets[[i]] <- data
}
```

As we can see on density plots imputed data are good only for half of columns. TT4, T4U and FTI are not imputed well.

Model testing

Then we can test our model. We will test random forest - ranger and xgboost. We will try it on 24 prepared dataset in different ways. One dataset without anything, 5 datasets with data imputed by mice. Then two times more because in every dataset we will apply logarithm to TSH column.

The best model will be with the biggest AUPRC measure. It is good measure for inbalanced target classes. In our case there are less than 10% of sick people so AUPRC is better measure than AUC.

```
# 80% train i 20% test data
train_ind <- as.matrix(read.table("indeksy_treningowe.txt")[,2])
test_ind <- setdiff(seq_len(nrow(dataset)), train_ind)

# cross-validation
create_task_log <- function(dataset_log, id) {
  dataset_log$TSH <- log(dataset_log$TSH)
  task_log <- TaskClassif$new(id=id, backend=dataset_log[train_ind, ], target="target", positive="1")
  return(task_log)
}
task_base <- TaskClassif$new(id='basic', backend=dataset[train_ind, ], target="target", positive="1")
task_log <- create_task_log(dataset, 'TSH_logarithm')

imputed_tasks <- list()
imputed_tasks_log <- list()
for(i in 1:5) {
  imputed_tasks[[i]] <- TaskClassif$new(id=paste0('imputed_data', as.character(i)), backend=datasets[[i
  imputed_tasks_log[[i]] <- create_task_log(datasets[[i]], paste0('imputed_data_logarithm', as.characte
}

tasks1 <- c(list(task_base, task_log), imputed_tasks, imputed_tasks_log)
tasks2 <- c(imputed_tasks, imputed_tasks_log)

learners <- c("classif.ranger", "classif.xgboost")
learners <- lapply(learners, lrn, predict_type = "prob", predict_sets = c("train", "test"))

resamplings <- rsmp("cv", folds=5)

set.seed(1233)
bmr1 <- benchmark(benchmark_grid(tasks1, learners[[2]], resamplings))
bmr2 <- benchmark(benchmark_grid(tasks2, learners, resamplings))

measures <- list(
  msr("classif.auc", id = "auc_train", predict_sets = "train"),
```

```r
  msr("classif.auc", id = "auc_test"),
  msr("classif.auprc", id = "auprc_train", predict_sets = "train"),
  msr("classif.auprc", id = "auprc_test")
)
results1 <- bmr1$aggregate(measures)
results2 <- bmr2$aggregate(measures)
print_results <- function(results) {
  results <- results[, c("task_id", "learner_id", "auc_test", "auprc_train", "auprc_test")]
  results[order(-results$auprc_test),]
}
```

```r
print_results(results1)
```

```
##                       task_id      learner_id  auc_test auprc_train auprc_test
##  1:             TSH_logarithm classif.xgboost 0.9695213   0.9620181  0.8888087
##  2:                     basic classif.xgboost 0.9648450   0.9606102  0.8873550
##  3:             imputed_data2 classif.xgboost 0.9570373   0.9287849  0.8767831
##  4: imputed_data_logarithm5 classif.xgboost 0.9619918   0.9314856  0.8758384
##  5:             imputed_data1 classif.xgboost 0.9617187   0.9217672  0.8582636
##  6: imputed_data_logarithm3 classif.xgboost 0.9574165   0.9314284  0.8575655
##  7:             imputed_data3 classif.xgboost 0.9612384   0.9315050  0.8562040
##  8: imputed_data_logarithm1 classif.xgboost 0.9574440   0.9315840  0.8524386
##  9:             imputed_data4 classif.xgboost 0.9585430   0.9274843  0.8511685
## 10: imputed_data_logarithm2 classif.xgboost 0.9624231   0.9238833  0.8509810
## 11:             imputed_data5 classif.xgboost 0.9552859   0.9341010  0.8381971
## 12: imputed_data_logarithm4 classif.xgboost 0.9582100   0.9200971  0.8213008
```

```r
print_results(results2)
```

```
##                       task_id      learner_id  auc_test auprc_train auprc_test
##  1:             imputed_data3  classif.ranger 0.9949752   0.9911154  0.9257234
##  2:             imputed_data4  classif.ranger 0.9953168   0.9910389  0.9230015
##  3:             imputed_data5  classif.ranger 0.9946256   0.9916500  0.9229402
##  4: imputed_data_logarithm2  classif.ranger 0.9948519   0.9910637  0.9223770
##  5: imputed_data_logarithm4  classif.ranger 0.9930487   0.9910035  0.9214038
##  6:             imputed_data2  classif.ranger 0.9947360   0.9910553  0.9211327
##  7: imputed_data_logarithm3  classif.ranger 0.9931204   0.9914233  0.9185031
##  8: imputed_data_logarithm1  classif.ranger 0.9941200   0.9917543  0.9163159
##  9: imputed_data_logarithm5  classif.ranger 0.9918568   0.9918396  0.9055597
## 10:             imputed_data1  classif.ranger 0.9833525   0.9911299  0.8982120
## 11: imputed_data_logarithm4 classif.xgboost 0.9599822   0.9271889  0.8949636
## 12:             imputed_data2 classif.xgboost 0.9630995   0.9291706  0.8771713
## 13: imputed_data_logarithm2 classif.xgboost 0.9621863   0.9291144  0.8741668
## 14: imputed_data_logarithm1 classif.xgboost 0.9569296   0.9270840  0.8733927
## 15:             imputed_data5 classif.xgboost 0.9631500   0.9309667  0.8604019
## 16:             imputed_data1 classif.xgboost 0.9498566   0.9204476  0.8593035
## 17: imputed_data_logarithm5 classif.xgboost 0.9614597   0.9323896  0.8584762
## 18:             imputed_data4 classif.xgboost 0.9551566   0.9278586  0.8581508
## 19:             imputed_data3 classif.xgboost 0.9615114   0.9313320  0.8564333
## 20: imputed_data_logarithm3 classif.xgboost 0.9573533   0.9279376  0.8353648
```

Better results are on basic dataset instead of any imputation. But generally ranger is better than xgboost. Unfortunately ranger does not support missing data. But the best result is with ranger on imputed_data3.

Final model

Below there are written results of final model on test dataset. Also there is a plot of precision recall curve of this model.

```
final_dataset <- datasets[[3]]
task_final <- TaskClassif$new(id='final', backend=final_dataset, target="target", positive="1")

learner <- learners[[1]]
learner$train(task_final, row_ids=train_ind)
prediction <- learner$predict(task_final, row_ids=test_ind)
print(prediction$score(msr("classif.auprc")))
```

```
## classif.auprc
##     0.8907985
```

```
print(prediction$score(msr("classif.auc")))
```

```
## classif.auc
##   0.9927077
```

```
auprc::precision_recall_curve(prediction$data$prob[,"1"], prediction$data$tab$truth, "1")
```