Exploratory Data Analysis - A case example

Marc A.T. Teunis

4/16/2019

Contents

Introduction - What is Exploratory Data Analysis?

The process of Exploratory Data Analysis is not a formal process with strict rules, steps or guidelines. Instead is a step-wise iterative process aimed at gaining more insight in the nature and peculiarities of the data. It is a process of several rounds of exploring:

- The data-types of the variables in the data
- The unique values in variables
- Errors and coding mistakes in the data
- Outliers and 'strange' values
- Data ditributions
- Exploratory graphs to investigate the data
- Descriptive statistics

For demo purposes, I explain a number of exploratory steps below on an example (real life) data set. There are also packages available in R to speed up this process and generate automatic reports for several steps in the EDA process. One example of such package is `{DataExplorer}`. In the last bit of this demo we will try out this package to compare it's output to our manual steps.

Document layout

My EDA reports all have a comparable structure and paragraph headings. Below you see the main paragraphs and their global respective content, in a typical EDA report:

Paragraph	Contents
Packages	Which packages are needed for this analysis/report
Data load	Code to load dataset and data origin
Unique values	What are the unique values / coding errors?
Variable types	What are the types of the variables (double etc.)
Missingness	What are missing values and how are they coded
Distributions	How does the distribution of the data/groups look?
Correlations	Are (some) variables correlated? Correlation matrix
Exploratory Plots	Exploratory plots to show trends in the data
Descriptive stats	Mean, standard deviation, max, min of variables
Conclusions	Major observations and suggestion for further (formal steps)

Packages

Packages and self-written functions are loaded.

```r
source(here::here("participant_cases", "R", "rotate_axis_labels.R"))
library(tidyverse)
library(here)
library(haven)
library(naniar)
library(tools)
library(corrr)
library(corrplot)
```

Data load

The dataset recieved from ... on 11 April 2019 came accompanied with the following email:

```
Beste ...

in de bijlage een dataset, waarin de PSK_T0 de afhankelijke variabele is.
De andere variabelen kunnen als onafhankelijke gezien worden.

een vriendelijke groet,

...
```

Data information

Most of the time the information you recieve as a data scientist is limited. In this case not much information was provided on beforehand:

Validation

One way to validate the match between the file and the analysis (and output of that analysis) is through a 'sumcheck'. The idea is that every file has it's own unique fingerprint on the basis of it's content. To calculate a fingerprint, a number of so-called 'hashing' systems have been developed. A simple one is the 'md5 sumcheck' hash.

Usually md5sums are calculated from a file and the resulting hash-key is stored in proximity to the data file. Preferabley in the same folder, carrying the same base name, with addition of "_md5sum.txt" to the orinal name of the datafile.

The md5sums can be calculated using the function `md5sum` from the `{tools}` package, or using the Linux Terminal and the Bash command:

`md5sum path_to_datafile_name > path_to_data_file_name_md5sum.txt`

Here we generate the md5sum from within R with `tools::md5sum()`. Next we load the .sav (SPSS archive) file. And validate the md5sum. This script will issue an ERROR if the md5sum file is absent, or if the hash is incorrect. It will also produce an ERROR if the number of records of the current version of the datafile you are using is different from the one used to generate this report.

```
## validation of the data with md5 sum
## produces a 32-character encrypted hash string that is unique for this datafile

md5sum <- tools::md5sum(files = here::here(
  "participant_cases",
  "data-raw",
  "D010",
  "Cursus R databse.sav")) %>%
    enframe()




## write filename and md5sum hash to file
write_lines(md5sum, path = here::here(
  "praticipant_cases",
  "data-raw",
  "D010",
  "Cursus R databse_sav_md5sum.txt"))
```

The version of the data that the analysis of the current document was based on has md5sum key:

`6b1084b7eb807088701cabfa3b980db7`

This key should be compared to the version of the file that you want to use to reproduce the analysis below.

To check the md5sum of your version of the data:

`tools::md5sum("path_to_your_version_of_the_data_file")`

The output should match the md5 key above.

Total record number

A simpeler method of data validation is to check the match between the expected total number of records to the total number of records available in the loaded dataset. Mind that this is a very simple check and it does not garantuee full validation.

The expected number of records for the correct version of the data is `18424`

Below we load the data and perform the two validation checks:

- Md5 sums
- Total number of records

The `stopifnot()` function is a handy tool to disable the script from executing if the validation rules are not compliant.

```r
data <- read_sav(file = here::here("participant_cases",
  "data-raw",
                                   "D010",
                                   "Cursus R databse.sav")) ## mind the typo


## total number of records:
x <- nrow(data) * ncol(data)

## hardcode number of record in code to check version
stopifnot(x == 18424)

## compare current md5sum to key
stopifnot(
tools::md5sum(here::here(
  "participant_cases",
  "data-raw",
  "D010",
  "Cursus R databse.sav")) ==
    "6b1084b7eb807088701cabfa3b980db7"
)
```

The advantage of using labelled variables in SPSS is evident: information about the variable is stored together with the data. We can access this information (also sometimes referred to as colum data) in R using:

```r
col_data <- lapply(data, function(x) attributes(x)$label)
col_data
```

```
## $geslacht
## Vrouw   Man
##   "1"   "2"
##
## $leeftijd
## NULL
##
## $DTF
##  ja nee
## "1" "2"
##
## $VAR00009
## NULL
##
## $locatie_klacht
##                    hoofd nek,schouder , boven rug      elleboog, pols/hand
##                      "1"                       "2"                       "3"
##                 lage rug                heup, knie                enkel,voet
##                      "4"                       "5"                       "6"
##        meerdere lokaties
##                      "7"
##
## $Werkstatus
##      werkend niet werkend
##          "1"          "2"
##
## $Werksetting
```

```
## Eerste lijn Tweede lijn  Derde lijn  Combinatie
##         "1"         "2"         "3"         "4"
##
## $duur_klachten
## [1] "In weken"
##
## $PSK_1_T0
## [1] "NRS hoe moeilijk het voor u was deze activiteit de afgelopen week uit te voeren"
##
## $VAR00005
## NULL
##
## $ip1_T0
## [1] "ip1 Hoeveel beïnvloedt uw ziekte uw leven"
##
## $ip2_T0
## [1] "ip2 Hoe lang denkt u dat uw ziekte zal duren"
##
## $ip3_T0
## [1] "ip3 Hoeveel controle vindt u dat u heeft over uw ziekte"
##
## $ip4_T0
## [1] "ip4 Hoeveel denkt u dat uw behandeling kan helpen bij uw ziekte"
##
## $ip5_T0
## [1] "ip5 Hoe sterk ervaart u klachten door uw ziekte"
##
## $ip6_T0
## [1] "ip6 Hoe bezorgd bent u over uw ziekte"
##
## $ip7_T0
## [1] "ip7 In welke mate vindt u dat u uw ziekte begrijpt"
##
## $ip8_T0
## [1] "ip8 Hoeveel invloed heeft de ziekte op uw stemming"
##
## $ip9_T0
## [1] "ip9 de drie belangrijkste factoren die naar uw opvatting uw ziekte hebben veroorzaakt"
##
## $pijn_acuut_chronisch
## [1] "pijn_acuut_chronisch"
##
## $dissomsc_T0
## [1] "Distress"
##
## $depsomsc_T0
## [1] "Depressie"
##
## $angsomsc_T0
## [1] "Angst"
##
## $somsomsc_T0
## [1] "Somatisatie"
##
```

```
## $dis_ord_T0
## [1] "Distress ordinaal"
##
## $dep_ord_T0
## [1] "Depressie ordinaal"
##
## $ang_ord_T0
## [1] "Angst ordinaal"
##
## $som_ord_T0
## [1] "Somatisatie ordinaal"
```

```
ind <- map_lgl(col_data, is.null)
cold_data_df <- col_data[!ind] %>% enframe()
```

Get individual labels

To get individual labels for each variable

```
col_data$locatie_klacht %>% knitr::kable()
```

	x
hoofd	1
nek,schouder , boven rug	2
elleboog, pols/hand	3
lage rug	4
heup, knie	5
enkel,voet	6
meerdere lokaties	7

We can see that most variables are labelled, some are not (`NULL`)

Unique values

To assess the correct variable type below we need to check the unique values of each variable. This will tell you what type of data is contained in each variable (column) of the dataset(s).

```
unique_values <- purrr::map(data, unique)
unique_values
```

```
## $geslacht
## [1] "2"   "1"   "999"
##
## $leeftijd
##  [1] 58 54 52 49 44 42 41 34 22 62 30 23 38 51 28 66 72 47 71 53 46 40 35
## [24] 50 39 56 24 18 61 48 63 67 37 64 19 33 36 60 59 43 55 45 73 68 29 57
## [47] 69 21 20 25 31 26 65 75 74 32 27 70
##
## $DTF
## [1] "1"   "2"   "999"
##
## $VAR00009
```

```
## [1] 0
##
## $locatie_klacht
## [1] "2"   "7"   "1"   "5"   "3"   "4"   "6"   "999"
##
## $Werkstatus
## [1] "1"   "2"   "999"
##
## $Werksetting
## [1] "1"        "999"      "4"        "Eerste l" "2"
##
## $duur_klachten
##  [1]   10    3    4   60   27   24  572    6   NA   12  104   28    5   25
## [15]    2    1   26    8    0  150   13   20  182  570  100   52   14  240
## [29]  520  106   40    7   15  200   16   30   72   90   35  234  670  260
## [43]   80  604   22  365   50  364    9   47  988 2000   36  500   70  156
## [57]  250  312   11 1612  400  468   84   32  125   48   18  720   53 3285
## [71]  130   45  304  530  532  372  108  256  288   75  416 1120   23  350
## [85] 1300   78  800 2496  728 2912  300  360  430  356  780   64  120   97
##
## $PSK_1_T0
##  [1]  2  3 10  0  4  6  9 NA  7  5  1  8
##
## $VAR00005
## [1] NA  0
##
## $ip1_T0
##  [1] NA  0  1  2 10  3  4  7  6  5  8  9
##
## $ip2_T0
##  [1] NA  0  1 10  2  5  3  4  8  9  6  7
##
## $ip3_T0
##  [1] NA  0  9  2  8  1  3  6  7 10  5  4
##
## $ip4_T0
##  [1] NA  7  0  1  2  5  3  9  4  6  8 10
##
## $ip5_T0
##  [1] NA  9  1  3  5  0  4  2  6  7  8 10
##
## $ip6_T0
##  [1] NA  0  1  2  3  4 10  8  5  7  9  6
##
## $ip7_T0
##  [1] NA  3  0  1  2  4  7  5 10  8  9  6
##
## $ip8_T0
##  [1] NA  0  2  1  4  3  5  6  7 10  8  9
##
## $ip9_T0
##  [1] ""         "999"      "1"        "5"        "2"        "4"
##  [7] "houding"  "3"        "werkdruk" "spanning" "stress"   "angst"
##
```

```
## $pijn_acuut_chronisch
## [1] 0 1
##
## $dissomsc_T0
##  [1]  0 14 11  1  8 NA  2 12  7 22 15  6  4  3  9 10 23 32  5 16 17 25 13
## [24] 20 26 18 19 30 29 24 31 21 28 27
##
## $depsomsc_T0
##  [1]  0  5  2  1  8  4 12  9  3  7 NA  6 11 10
##
## $angsomsc_T0
##  [1]  0  2  9  3  6  1 22 10 18 NA 13  7  8 11  4  5 16 24 21 15 12 17 14
## [24] 19 23 20
##
## $somsomsc_T0
##  [1]  6 16 NA  1  4  8 12  0 27  7  5 10 11  2  9 17  3 21 26 19 14 20 15
## [24] 28 18 22 23 13 25 30 29 24
##
## $dis_ord_T0
## [1]  1  2 NA  3
##
## $dep_ord_T0
## [1]  1  2  3 NA
##
## $ang_ord_T0
## [1]  1  2  3 NA
##
## $som_ord_T0
## [1]  1  2 NA  3
```

There are a couple of remarkable things:

- Some variables seem coded with digits, but also contain strings (e.g. `Werksetting` and `ip9_T0`)
- There are a few records coded with the string "999". I suspect this is an alternative annotation to indicate NA (missing values). It is a common practice for some fields or for people working with different tools than R to use a value for NA that does not (or cannot), normally occur in the data. In R this is not recognized as an `NA` value and it is feasible to do some recoding to get rid of these "999" values.

Below we will recode these to be actual NA values and investigate the missingness further.

Variables parsed while loading

Sometimes you can also learn whether there is something strange in the data when looking at the type of column after the dataload. Especially when you use functions from the tidyverse `{readr}` package. The functions from `{readr}` use a prediction model to assess the intended type of each column when a datafile is parsed. More information can be obtained from chapter "Importing Data" from the R4DS book.

Below we review the type of the columns in relation to the data of the first few rows of the loaded data. Can you spot the peculiarities?

```r
head(data, 10)
```

```
## # A tibble: 10 x 28
##    geslacht leeftijd DTF     VAR00009 locatie_klacht Werkstatus Werksetting
##    <chr+lb>    <dbl> <chr+l>    <dbl> <chr+lbl>      <chr+lbl>  <chr+lbl>
##  1 2 [Man]        58 1 [ja]         0 2 [nek,schoud~ 1 [werken~ 1 [Eerste ~
```

```
##   2 2 [Man]        58 1 [ja]         0 7 [meerdere l~ 1 [werken~ 1 [Eerste ~
##   3 1 [Vrou~       54 2 [nee]        0 1 [hoofd]      1 [werken~ 999
##   4 1 [Vrou~       52 2 [nee]        0 5 [heup, knie] 1 [werken~ 1 [Eerste ~
##   5 2 [Man]        49 1 [ja]         0 3 [elleboog, ~ 1 [werken~ 1 [Eerste ~
##   6 1 [Vrou~       44 1 [ja]         0 4 [lage rug]   1 [werken~ 1 [Eerste ~
##   7 1 [Vrou~       42 2 [nee]        0 7 [meerdere l~ 2 [niet w~ 1 [Eerste ~
##   8 1 [Vrou~       41 2 [nee]        0 4 [lage rug]   1 [werken~ 1 [Eerste ~
##   9 2 [Man]        34 1 [ja]         0 5 [heup, knie] 1 [werken~ 1 [Eerste ~
##  10 1 [Vrou~       22 2 [nee]        0 5 [heup, knie] 1 [werken~ 1 [Eerste ~
## # ... with 21 more variables: duur_klachten <dbl>, PSK_1_T0 <dbl>,
## #   VAR00005 <dbl>, ip1_T0 <dbl>, ip2_T0 <dbl>, ip3_T0 <dbl>,
## #   ip4_T0 <dbl>, ip5_T0 <dbl>, ip6_T0 <dbl>, ip7_T0 <dbl>, ip8_T0 <dbl>,
## #   ip9_T0 <chr+lbl>, pijn_acuut_chronisch <dbl+lbl>, dissomsc_T0 <dbl>,
## #   depsomsc_T0 <dbl>, angsomsc_T0 <dbl>, somsomsc_T0 <dbl>,
## #   dis_ord_T0 <dbl+lbl>, dep_ord_T0 <dbl+lbl>, ang_ord_T0 <dbl+lbl>,
## #   som_ord_T0 <dbl+lbl>
```

Most columns show numeric values, like column `locatie_klacht` or `DTF`, yet they were read-in by the `read_sav()` function from the `{haven}` package as character variables. The SPSS labels that are also vissible in the 'pretty-print' of the data tibble, reviel the secret. Most variables that got parsed as character are actually categorical factor variables. Why were they parsed as characters and not as factors or even doubles (numeric)? For two reasons:

- Tidyverse readr and haven parsers never write strings as factors.
- Probably the variables that were parsed as characters contain not only nummeric values but also strings. Coercion rules state that the object combining numerical values and strings can only be converted to a character (and than if applicable to a factor)

This minimal example shows the above to be true.

```
## dataframe columns
read_csv("colA, colB
         1, 2
         3.77, 4.66
         3, 5
         test, 4"
         )
```

```
## # A tibble: 4 x 2
##   colA   colB
##   <chr> <dbl>
## 1 1      2
## 2 3.77   4.66
## 3 3      5
## 4 test   4
```

```
read_csv("colA, colB
         1, 2
         3.77, 4.66
         3, 5
         test, 4"
         ) %>%
  mutate(colA = as_factor(colA))
```

```
## # A tibble: 4 x 2
##   colA   colB
##   <fct> <dbl>
```

```
## 1 1       2
## 2 3.77    4.66
## 3 3       5
## 4 test    4
```

```
## indivicual vectors
A <- c(1,3.77,3, "test")
typeof(A)
```

```
## [1] "character"
```

```
B <- c(2, 4.66, 5, 4)
typeof(B)
```

```
## [1] "double"
```

Because `colA` contains the string `"test"` `read_csv` has to parse it as a character variable. `colB` only contains numeric values, so `read_csv()` will parse it as a double variable. The individual vectors also shows the coercion of vector A to a "character" vector and vector B to a "double". We can also convert colA to a factor with `as_factor()` from `{forcats}`. Or, maybe more feasible in this example, we could dicide that the value "test" is an entry-error and label it as an `NA`

```
read_csv("colA, colB
         1, 2
         3.77, 4.66
         3, 5
         test, 4"
         ) %>%
  replace_with_na_all(condition = ~.x == "test")
```

```
## # A tibble: 4 x 2
##   colA   colB
##   <chr> <dbl>
## 1 1       2
## 2 3.77    4.66
## 3 3       5
## 4 <NA>    4
```

Back to the case:

Missingness

In order to get an idea on which variable contain the value "999" we write our own function with a regular expression looking for a specific entry in a specific variable (x). We apply the function to our list of unique value from the chunk above to get a named integer vector, indicating for each variable whther that variable contains the "999" string (value = 1), or not (value = 0). This integer is than converted to a logical vectord and used as an index to get the actual variable names in a vector for later use.

```
## get all "999" values as a logical index, per variable
get_999 <- function(x, entry){
  ind <- str_detect(string = x, pattern = entry)
  sum(ind)
}

## loop the get_999 function over all columns, convert to lgl
has_999 <- map_int(unique_values, get_999, entry = "999") %>%
 as.logical()
```

```r
## select only those columns that have a "999" value somewhere
var_with_999 <- names(data)[has_999] %>%
  na.omit() %>%
  as.vector() %>%
  print()
```

```
## [1] "geslacht"       "DTF"            "locatie_klacht" "Werkstatus"
## [5] "Werksetting"    "ip9_T0"
```

In oder to get a complete picture on the missingness in this dataset we need to recode the "999" values into `NA` values.

Recoding records ("999" -> `NA`)

see also: https://cran.r-project.org/web/packages/naniar/vignettes/replace-with-na.html

```r
## replace all "999" values in all variables

data_clean <- data %>%
  replace_with_na_all(condition = ~.x == "999")

map(data_clean, unique) ## the "999" entries are gone.
```

```
## $geslacht
## [1] "2" "1" NA
##
## $leeftijd
##  [1] 58 54 52 49 44 42 41 34 22 62 30 23 38 51 28 66 72 47 71 53 46 40 35
## [24] 50 39 56 24 18 61 48 63 67 37 64 19 33 36 60 59 43 55 45 73 68 29 57
## [47] 69 21 20 25 31 26 65 75 74 32 27 70
##
## $DTF
## [1] "1" "2" NA
##
## $VAR00009
## [1] 0
##
## $locatie_klacht
## [1] "2" "7" "1" "5" "3" "4" "6" NA
##
## $Werkstatus
## [1] "1" "2" NA
##
## $Werksetting
## [1] "1"        NA         "4"        "Eerste l" "2"
##
## $duur_klachten
##  [1]   10    3    4   60   27   24  572    6   NA   12  104   28    5   25
## [15]    2    1   26    8    0  150   13   20  182  570  100   52   14  240
## [29]  520  106   40    7   15  200   16   30   72   90   35  234  670  260
## [43]   80  604   22  365   50  364    9   47  988 2000   36  500   70  156
## [57]  250  312   11 1612  400  468   84   32  125   48   18  720   53 3285
## [71]  130   45  304  530  532  372  108  256  288   75  416 1120   23  350
## [85] 1300   78  800 2496  728 2912  300  360  430  356  780   64  120   97
```

```
##
## $PSK_1_T0
##  [1]  2  3 10  0  4  6  9 NA  7  5  1  8
##
## $VAR00005
## [1] NA  0
##
## $ip1_T0
##  [1] NA  0  1  2 10  3  4  7  6  5  8  9
##
## $ip2_T0
##  [1] NA  0  1 10  2  5  3  4  8  9  6  7
##
## $ip3_T0
##  [1] NA  0  9  2  8  1  3  6  7 10  5  4
##
## $ip4_T0
##  [1] NA  7  0  1  2  5  3  9  4  6  8 10
##
## $ip5_T0
##  [1] NA  9  1  3  5  0  4  2  6  7  8 10
##
## $ip6_T0
##  [1] NA  0  1  2  3  4 10  8  5  7  9  6
##
## $ip7_T0
##  [1] NA  3  0  1  2  4  7  5 10  8  9  6
##
## $ip8_T0
##  [1] NA  0  2  1  4  3  5  6  7 10  8  9
##
## $ip9_T0
##  [1] ""         NA         "1"        "5"        "2"        "4"
##  [7] "houding"  "3"        "werkdruk" "spanning" "stress"   "angst"
##
## $pijn_acuut_chronisch
## [1] 0 1
##
## $dissomsc_T0
##  [1]  0 14 11  1  8 NA  2 12  7 22 15  6  4  3  9 10 23 32  5 16 17 25 13
## [24] 20 26 18 19 30 29 24 31 21 28 27
##
## $depsomsc_T0
##  [1]  0  5  2  1  8  4 12  9  3  7 NA  6 11 10
##
## $angsomsc_T0
##  [1]  0  2  9  3  6  1 22 10 18 NA 13  7  8 11  4  5 16 24 21 15 12 17 14
## [24] 19 23 20
##
## $somsomsc_T0
##  [1]  6 16 NA  1  4  8 12  0 27  7  5 10 11  2  9 17  3 21 26 19 14 20 15
## [24] 28 18 22 23 13 25 30 29 24
##
## $dis_ord_T0
```

```
## [1]  1  2 NA  3
##
## $dep_ord_T0
## [1]  1  2  3 NA
##
## $ang_ord_T0
## [1]  1  2  3 NA
##
## $som_ord_T0
## [1]  1  2 NA  3
```

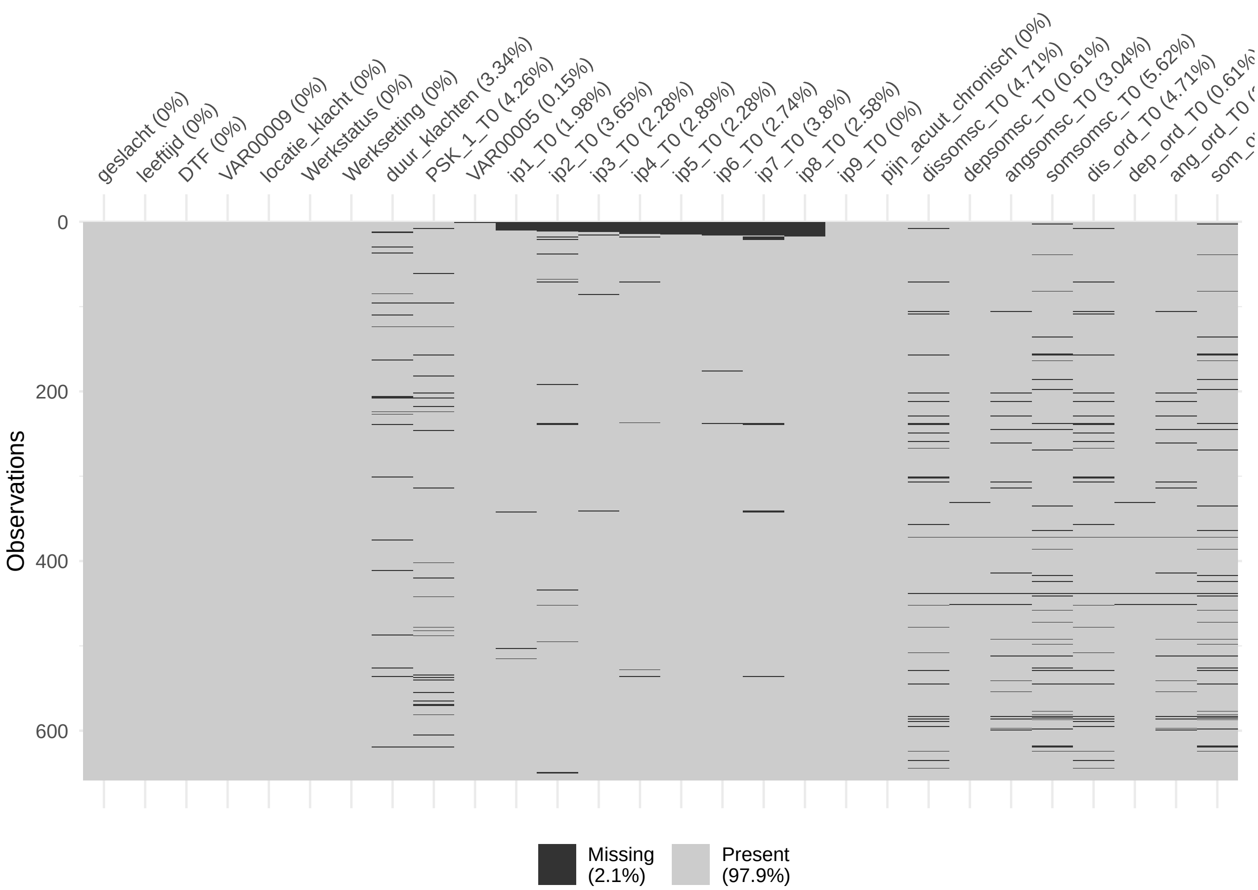
Visualize missingness in the data

see also: https://cran.r-project.org/web/packages/naniar/vignettes/naniar-visualisation.html

Now that we recoded all missing values in the data to actual `NA` values that make sense to R, we can start exploring the data and the missingness in more detail.

To get a genearal idea on the total mssingness (records coded with formal value `NA` in R)

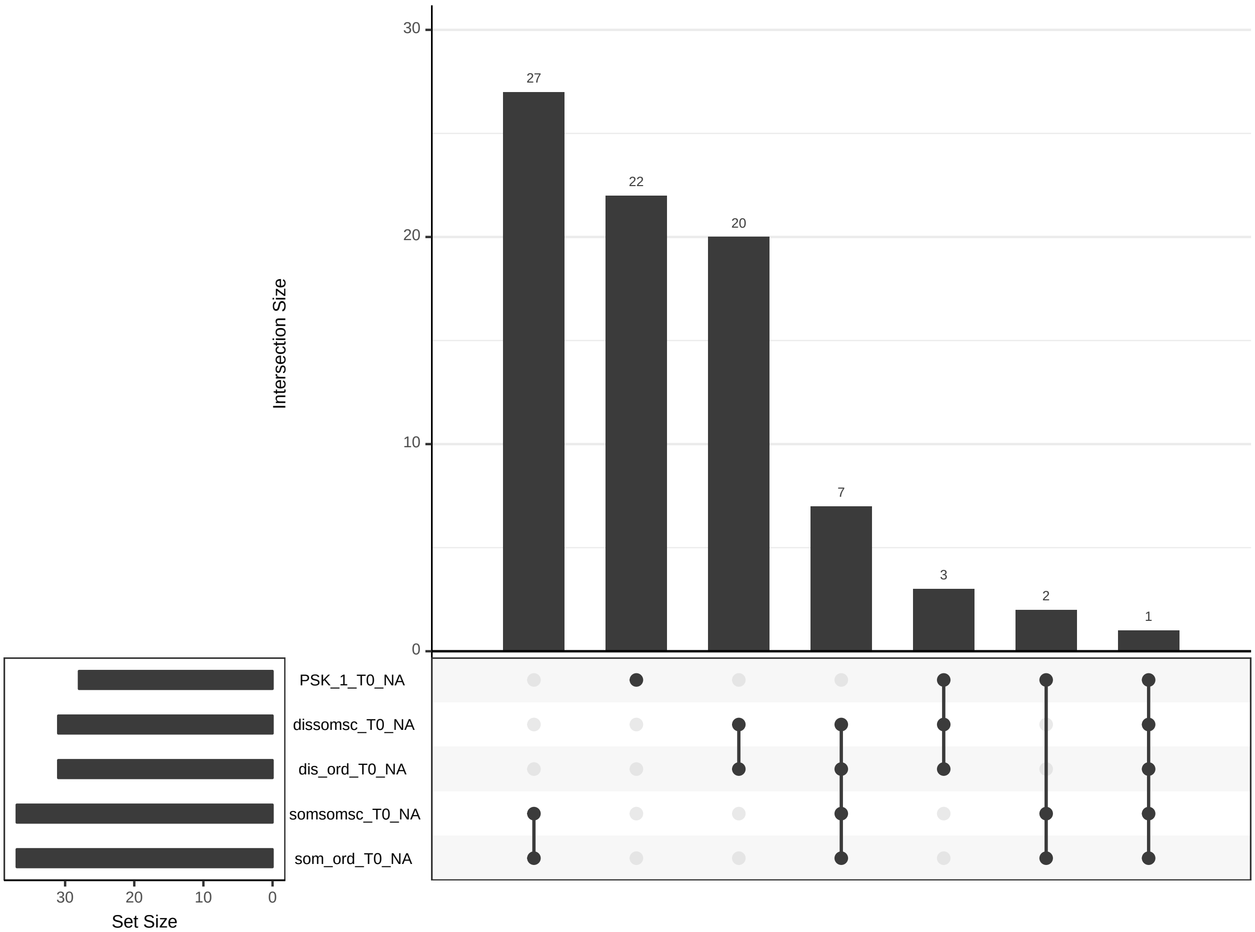
```
vis_miss(data)
```




```
## total sum NA values
sum(is.na(data))
```

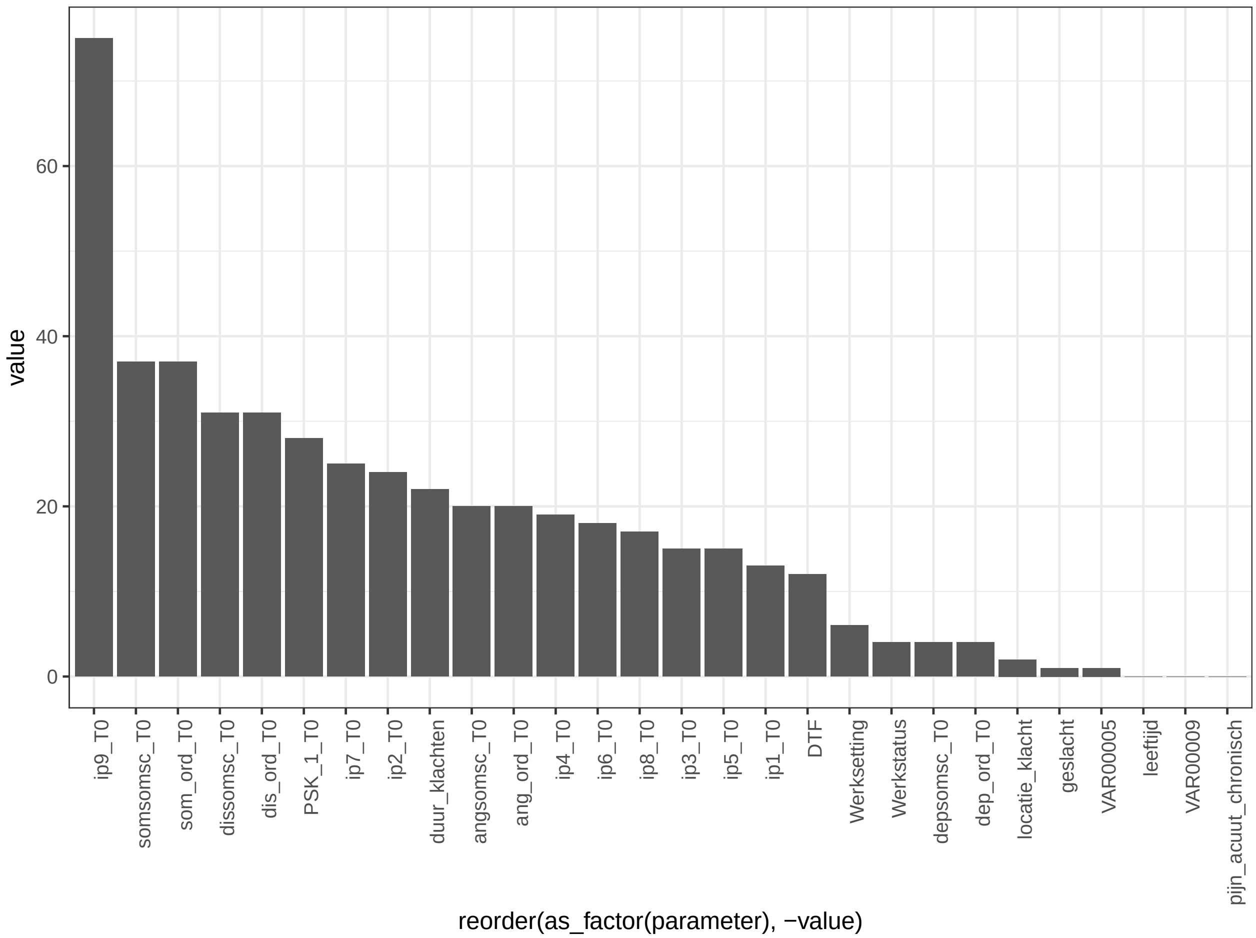
```
## [1] 381
```

```r
gg_miss_upset(data)
```



Sum of NA in each variable and make a ranked bar plot

```r
map_df(data_clean, function(x){sum(is.na(x))}) %>%
  gather(geslacht:som_ord_T0, key = "parameter", value = "value") %>%
  ggplot(aes(x = reorder(as_factor(parameter), -value), y = value)) +
  geom_col() +
  theme_bw() + rotate_axis_labels(axis = "x", angle = 90)
```



Variable types

After loading the data, and assessing (and maybe adapting NA values), the types of the variables can be assessed and possible adapted. We already assessed that some variables containing numeric values were converted to character. This tells us that there are string-values in these variables. Let's get the datatypes and the head of the data again

```
map(data_clean, typeof)
```

```
## $geslacht
## [1] "character"
##
## $leeftijd
## [1] "double"
##
## $DTF
## [1] "character"
##
## $VAR00009
## [1] "double"
##
## $locatie_klacht
## [1] "character"
```

```
##
## $Werkstatus
## [1] "character"
##
## $Werksetting
## [1] "character"
##
## $duur_klachten
## [1] "double"
##
## $PSK_1_T0
## [1] "double"
##
## $VAR00005
## [1] "double"
##
## $ip1_T0
## [1] "double"
##
## $ip2_T0
## [1] "double"
##
## $ip3_T0
## [1] "double"
##
## $ip4_T0
## [1] "double"
##
## $ip5_T0
## [1] "double"
##
## $ip6_T0
## [1] "double"
##
## $ip7_T0
## [1] "double"
##
## $ip8_T0
## [1] "double"
##
## $ip9_T0
## [1] "character"
##
## $pijn_acuut_chronisch
## [1] "double"
##
## $dissomsc_T0
## [1] "double"
##
## $depsomsc_T0
## [1] "double"
##
## $angsomsc_T0
## [1] "double"
```

```
##
## $somsomsc_T0
## [1] "double"
##
## $dis_ord_T0
## [1] "double"
##
## $dep_ord_T0
## [1] "double"
##
## $ang_ord_T0
## [1] "double"
##
## $som_ord_T0
## [1] "double"
```

```
head(data_clean)
```

```
## # A tibble: 6 x 28
##   geslacht leeftijd DTF   VAR00009 locatie_klacht Werkstatus Werksetting
##   <chr>       <dbl> <chr>    <dbl> <chr>          <chr>      <chr>
## 1 2              58 1            0 2              1          1
## 2 2              58 1            0 7              1          1
## 3 1              54 2            0 1              1          <NA>
## 4 1              52 2            0 5              1          1
## 5 2              49 1            0 3              1          1
## 6 1              44 1            0 4              1          1
## # ... with 21 more variables: duur_klachten <dbl>, PSK_1_T0 <dbl>,
## #   VAR00005 <dbl>, ip1_T0 <dbl>, ip2_T0 <dbl>, ip3_T0 <dbl>,
## #   ip4_T0 <dbl>, ip5_T0 <dbl>, ip6_T0 <dbl>, ip7_T0 <dbl>, ip8_T0 <dbl>,
## #   ip9_T0 <chr>, pijn_acuut_chronisch <dbl>, dissomsc_T0 <dbl>,
## #   depsomsc_T0 <dbl>, angsomsc_T0 <dbl>, somsomsc_T0 <dbl>,
## #   dis_ord_T0 <dbl>, dep_ord_T0 <dbl>, ang_ord_T0 <dbl>, som_ord_T0 <dbl>
```

```
attr(data_clean, "variable.labels")
```

```
## NULL
```

```
attr(data, "variable.labels")
```

```
## NULL
```

Using variable labels to learn more (maybe about coding mistakes?)

Above we investigated the variable labels that were assigned to the variables in the original SPSS file. We can access these in the `col_data` object. Maybe these labels will provide more insight.

Below we look at the unique values in every variable of our cleaned data. And we compare these with the labels, do you see the mismatch?

```
unique_values_clean <- map(data_clean, unique)
col_data
```

```
## $geslacht
## Vrouw   Man
##   "1"   "2"
##
```

```
## $leeftijd
## NULL
##
## $DTF
##  ja nee
## "1" "2"
##
## $VAR00009
## NULL
##
## $locatie_klacht
##                    hoofd nek,schouder , boven rug      elleboog, pols/hand
##                      "1"                        "2"                         "3"
##                 lage rug               heup, knie               enkel,voet
##                      "4"                        "5"                         "6"
##        meerdere lokaties
##                      "7"
##
## $Werkstatus
##      werkend niet werkend
##          "1"          "2"
##
## $Werksetting
## Eerste lijn Tweede lijn  Derde lijn  Combinatie
##         "1"         "2"         "3"         "4"
##
## $duur_klachten
## [1] "In weken"
##
## $PSK_1_T0
## [1] "NRS hoe moeilijk het voor u was deze activiteit de afgelopen week uit te voeren"
##
## $VAR00005
## NULL
##
## $ip1_T0
## [1] "ip1 Hoeveel beïnvloedt uw ziekte uw leven"
##
## $ip2_T0
## [1] "ip2 Hoe lang denkt u dat uw ziekte zal duren"
##
## $ip3_T0
## [1] "ip3 Hoeveel controle vindt u dat u heeft over uw ziekte"
##
## $ip4_T0
## [1] "ip4 Hoeveel denkt u dat uw behandeling kan helpen bij uw ziekte"
##
## $ip5_T0
## [1] "ip5 Hoe sterk ervaart u klachten door uw ziekte"
##
## $ip6_T0
## [1] "ip6 Hoe bezorgd bent u over uw ziekte"
##
## $ip7_T0
```

```
## [1] "ip7 In welke mate vindt u dat u uw ziekte begrijpt"
##
## $ip8_T0
## [1] "ip8 Hoeveel invloed heeft de ziekte op uw stemming"
##
## $ip9_T0
## [1] "ip9 de drie belangrijkste factoren die naar uw opvatting uw ziekte hebben veroorzaakt"
##
## $pijn_acuut_chronisch
## [1] "pijn_acuut_chronisch"
##
## $dissomsc_T0
## [1] "Distress"
##
## $depsomsc_T0
## [1] "Depressie"
##
## $angsomsc_T0
## [1] "Angst"
##
## $somsomsc_T0
## [1] "Somatisatie"
##
## $dis_ord_T0
## [1] "Distress ordinaal"
##
## $dep_ord_T0
## [1] "Depressie ordinaal"
##
## $ang_ord_T0
## [1] "Angst ordinaal"
##
## $som_ord_T0
## [1] "Somatisatie ordinaal"
```

Let's zoom in on one of the variables:

`Werksetting`

If we look at the labels for `Werksetting` we expect to see only 4 possible values in this column:

```
col_data$Werksetting
```

```
## Eerste lijn Tweede lijn  Derde lijn  Combinatie
##         "1"         "2"         "3"         "4"
```

If we look at the actual values for this column:

```
unique_values_clean$Werksetting
```

```
## [1] "1"        NA         "4"        "Eerste l" "2"
```

We see a striking discrepacy between the intended labels and the actual values in this variable. Probably "Eerste l" should have been coded as "1". Also striking is that the intended label "2" (Tweede lijn) en "3" (Derde lijn) do not exist in the data.

In order to assess whether these are really entry or conding errors we have to contact the supplier of the data. We can than decide what to do: relabel, remove, recode to `NA` or even recieve a new updated datafile from the supplier?

Distributions

Data distrubtion apply only to numeric data and can provide insight in:

- How are the values (between groups and overall) related and distributed
- Are there extreme outliers and in which variable or group

There are a number of graphs with which you can study ditributions and outliers. Usually it is good to combine a number of different plot types into one analysis to get a complete picture.

Below we first select the numeric variables only. Mind that when you should decide to relabel the data in the above step you would end up with more variables to study.

```
## select only numeric variables
is_numeric_lgl <- map_lgl(data, is.numeric)

names_numeric <- names(data)[is_numeric_lgl]

## select data with only numeric vars
data_numeric <- data %>%
  select(names_numeric) ## 22 vars left that are numeric
```

Reshape for ggplot

The {ggplot2} package works best with dataframes in long (or so-called stacked) format. Next we gather all variables into one column and the values in another.

```
names(data_numeric)
```

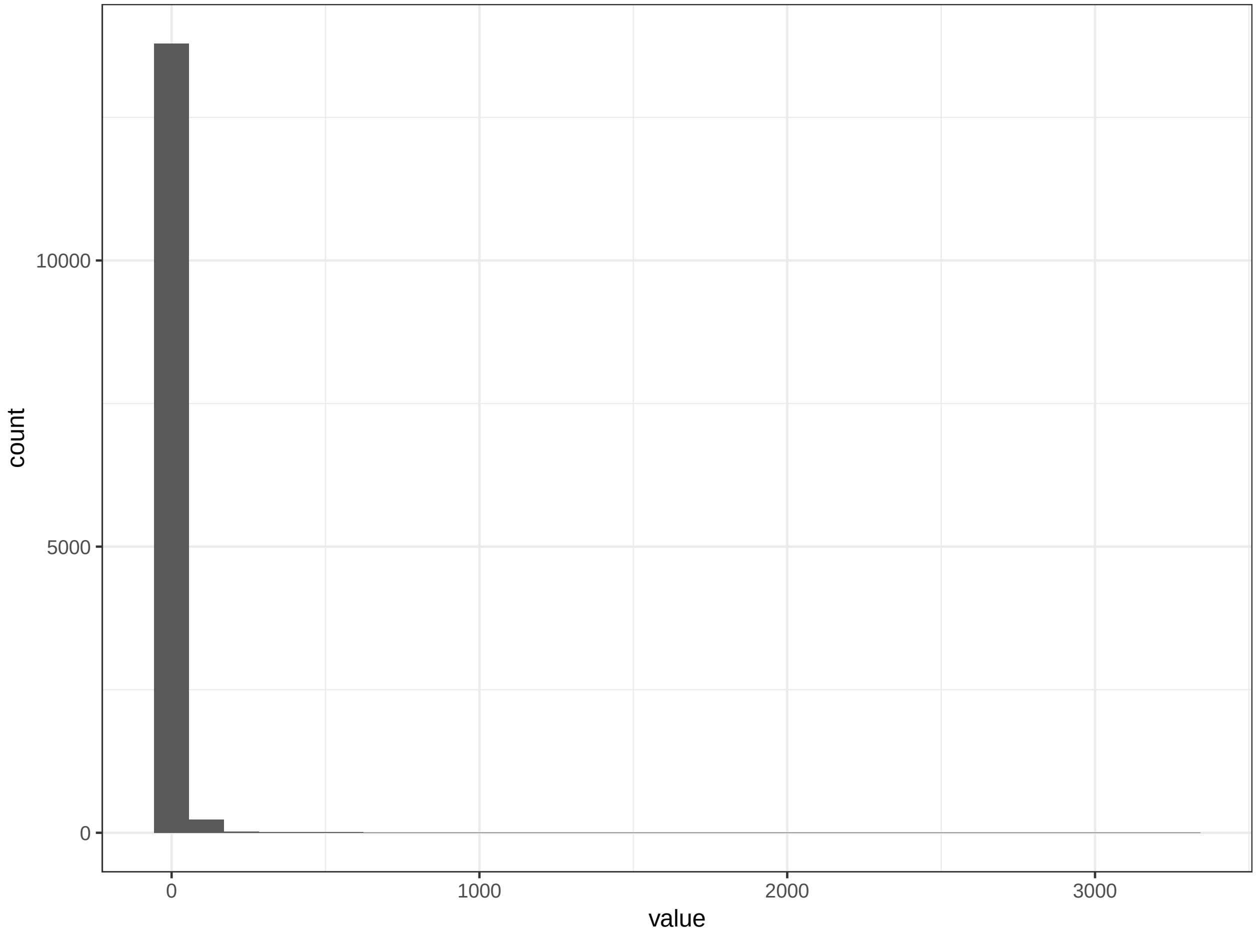
```
##  [1] "leeftijd"              "VAR00009"              "duur_klachten"
##  [4] "PSK_1_T0"              "VAR00005"              "ip1_T0"
##  [7] "ip2_T0"                "ip3_T0"                "ip4_T0"
## [10] "ip5_T0"                "ip6_T0"                "ip7_T0"
## [13] "ip8_T0"                "pijn_acuut_chronisch" "dissomsc_T0"
## [16] "depsomsc_T0"           "angsomsc_T0"           "somsomsc_T0"
## [19] "dis_ord_T0"            "dep_ord_T0"            "ang_ord_T0"
## [22] "som_ord_T0"
```

```
data_nummeric_long <- data_numeric %>%
  gather(leeftijd:som_ord_T0,
         key = "parameter",
         value = "value")
```

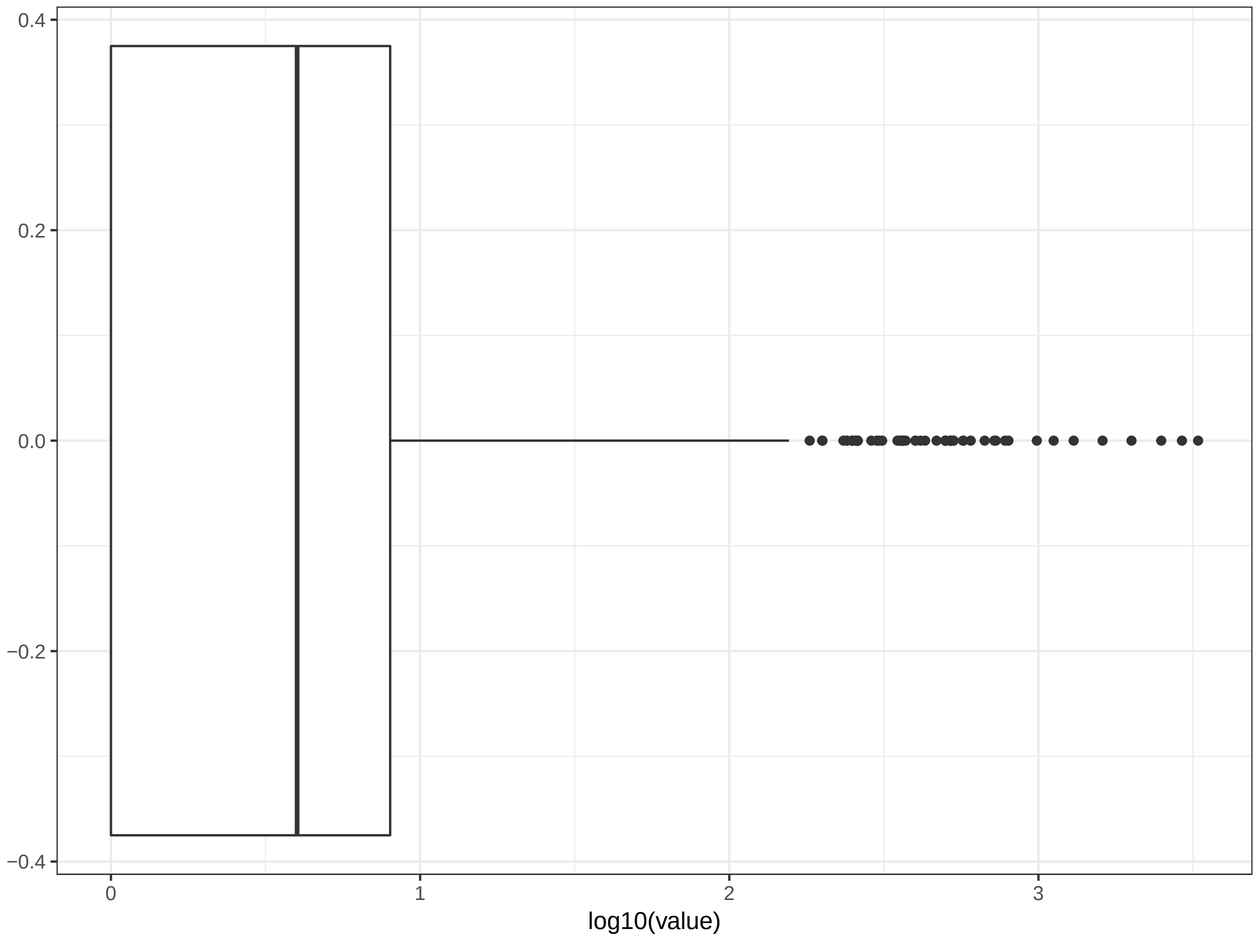
Overall distribution

We can look at the overall distribution of all numeric values in the data with a histogram or boxplot.

```
data_nummeric_long %>%
ggplot(aes(x = value)) +
  geom_histogram()
```



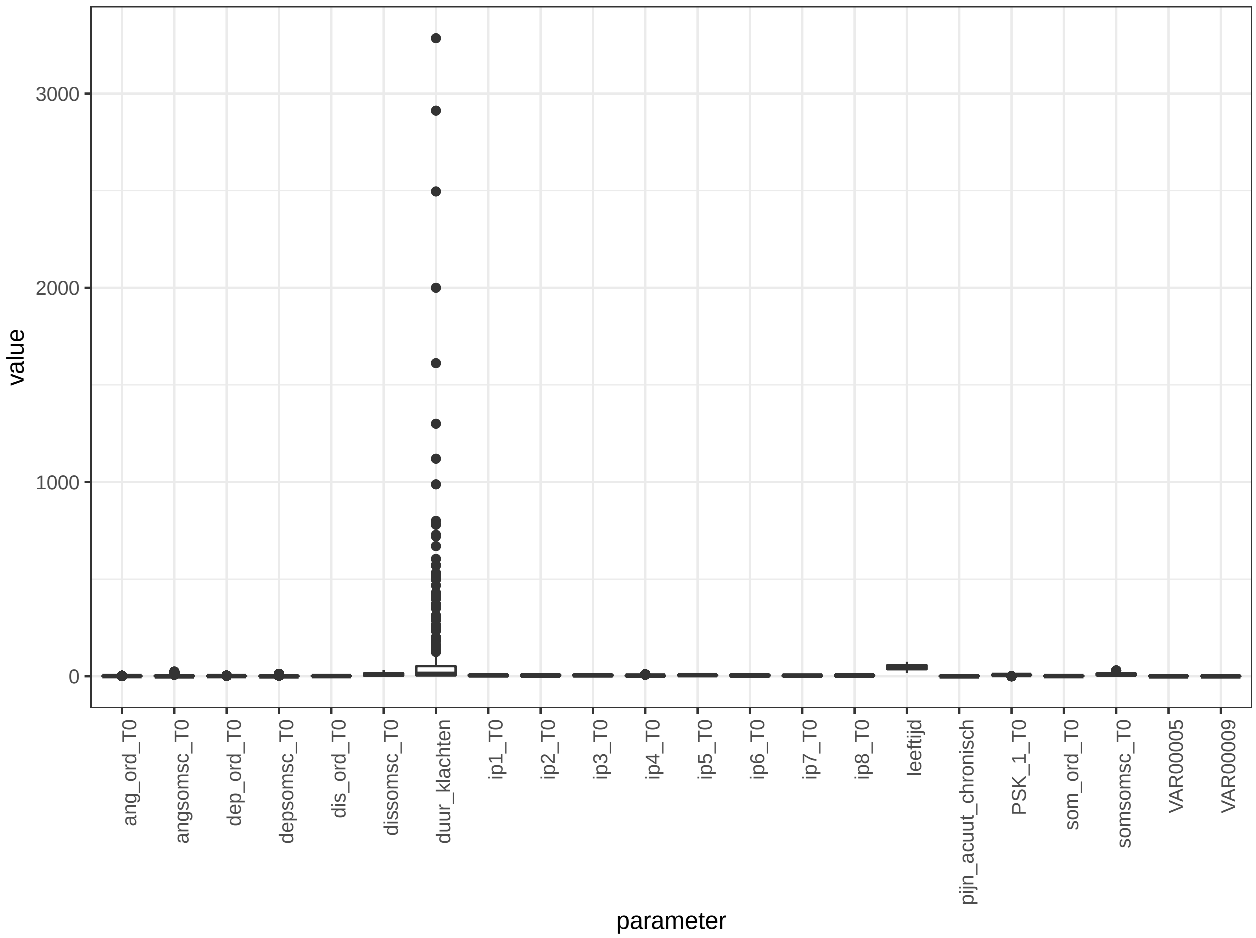

```r
## or
data_nummeric_long %>%
ggplot(aes(y = log10(value))) +
  geom_boxplot() +
  coord_flip()
```



We have a long tailed ditribution in both plots, suggesting outliers on the high end of the `value` variable. But we have all data here together, so maybe the scale of all variables is very unequal or there are group effects that are masked?

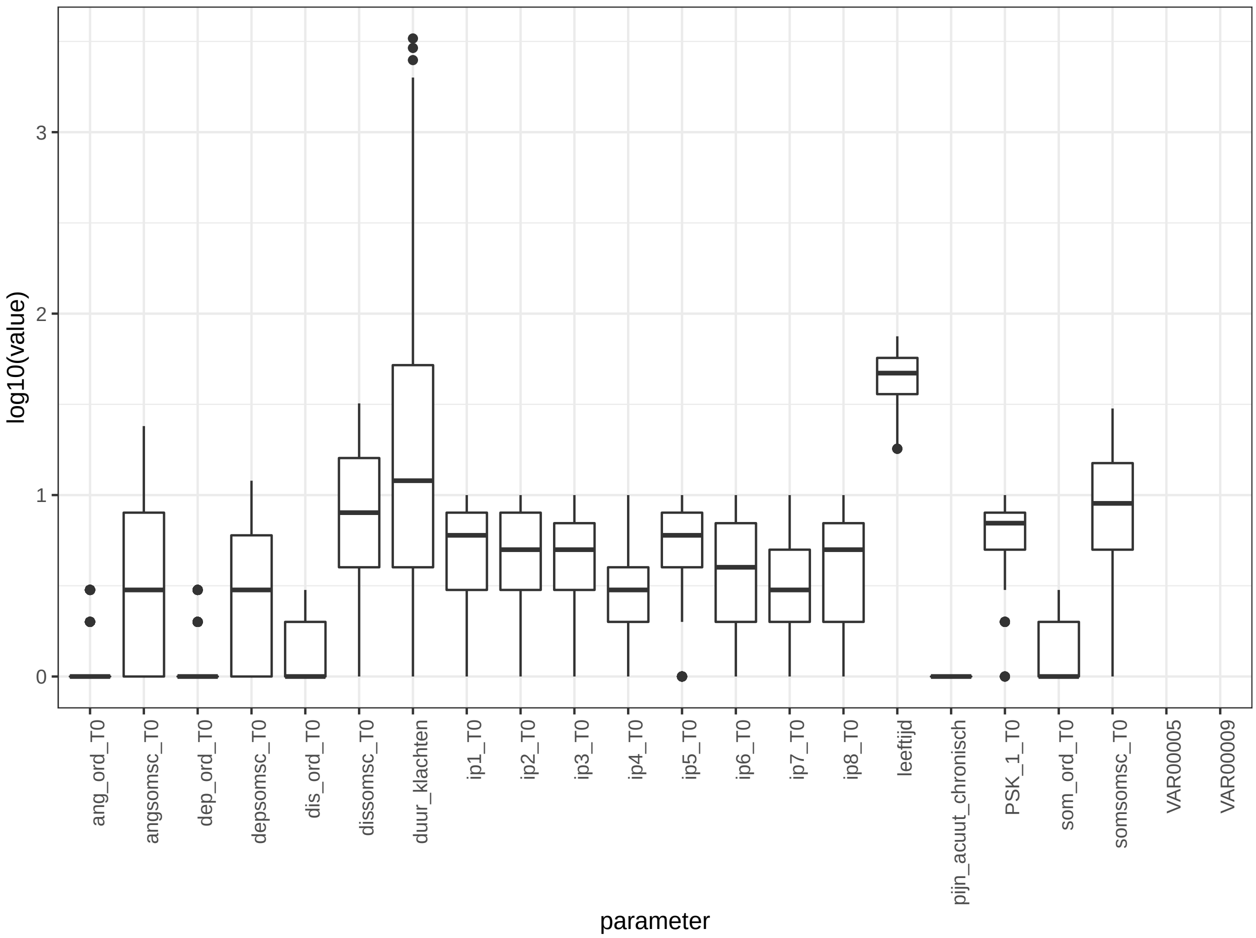
We can study the different scale of the variables with a boxplot per variable. In this way we can also quickly spot outliers.

```
data_nummeric_long %>%
  ggplot(aes(x = parameter,
             y = value)) +
  geom_boxplot(aes(group = parameter)) +
  rotate_axis_labels(axis = "x", angle = 90, hjust = 1)
```



It seems we have one variable which has a very diffrent scale than all the others. The extreme high value of this `duur_klachten` variable maskes our overview. Try a log10 transformation on the y-axis to do some rescaling of all variable values.

```
data_nummeric_long %>%
  ggplot(aes(x = parameter,
             y = log10(value))) +
  geom_boxplot(aes(group = parameter)) +
  rotate_axis_labels(axis = "x", angle = 90)
```

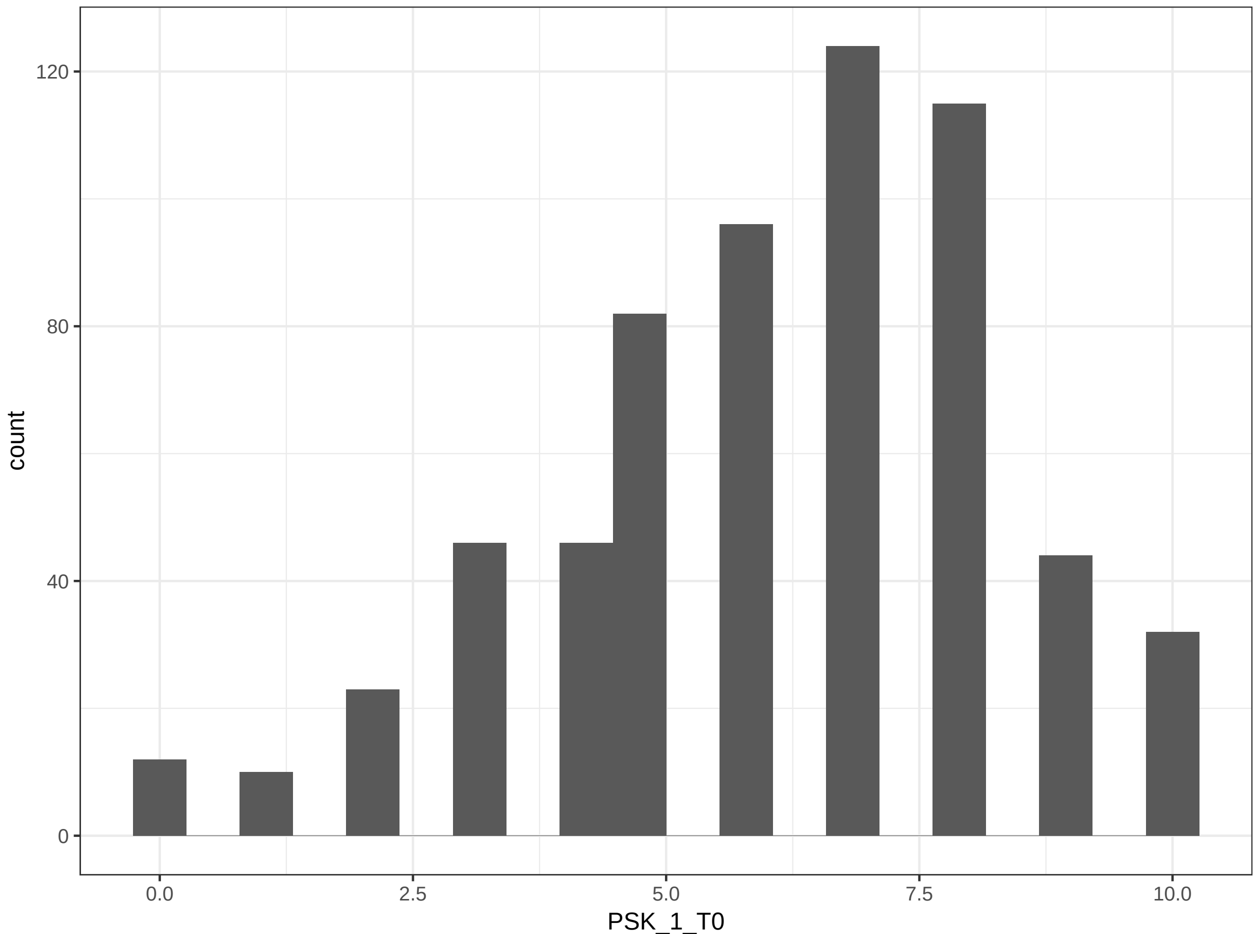


One variable in the data has been labelled by the supplier as the responsive or so-called dependent variable `PSK_1_T0`. Let's examine the distribution of this variable.

```
typeof(data_clean$PSK_1_T0)
```

```
## [1] "double"
```

```
data_clean %>%
  ggplot(aes(x = PSK_1_T0)) +
  geom_histogram(bins = 20)
```

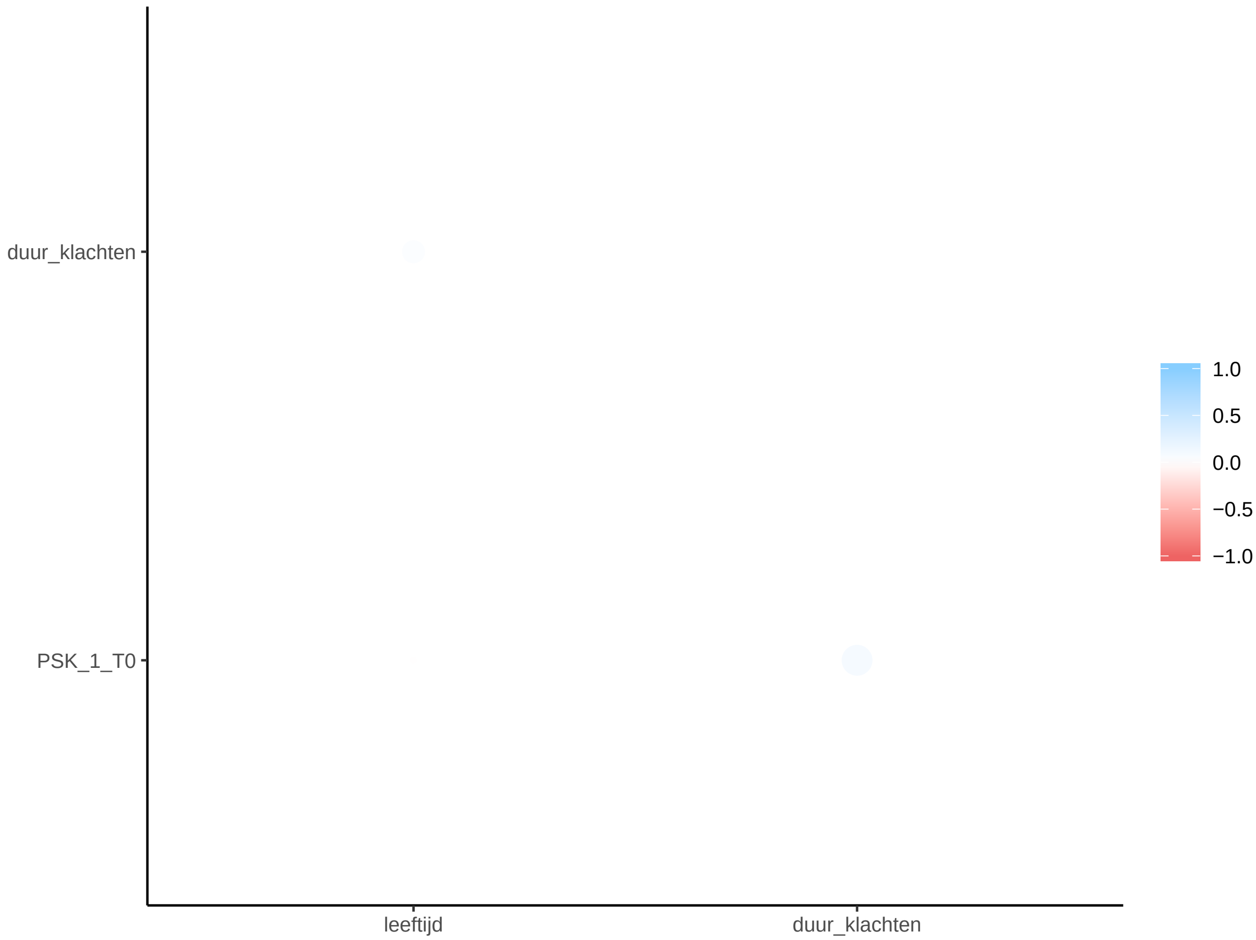


What can you conclude from this histogram?

Correlations

To investigate co-variates or autocorrelation we can generate a correlation matrix. We can only do this using the true numeric variables in the data: `leeftijd`, `duur_klachten` and `PSK_1_T0`. Most other variables in the data are in fact categorical variables (questionaire scores)

```
data_numeric %>%
  dplyr::select(leeftijd,
                duur_klachten,
                PSK_1_T0) %>%
  na.omit() %>%
  correlate() %>%
  rearrange() %>%
  shave() %>%
  rplot()
```



There is no evident large correlation between these numeric variables.

Exploratory plots

To start the exploratory plot iteration, I usually start bij plotting a scatter plot using the dependent variable, maybe another numeric variable and some evident grouping variables for which you would like to study their relationship with the dependent variable.

For this step we will use the `data_clean` version of the dataset

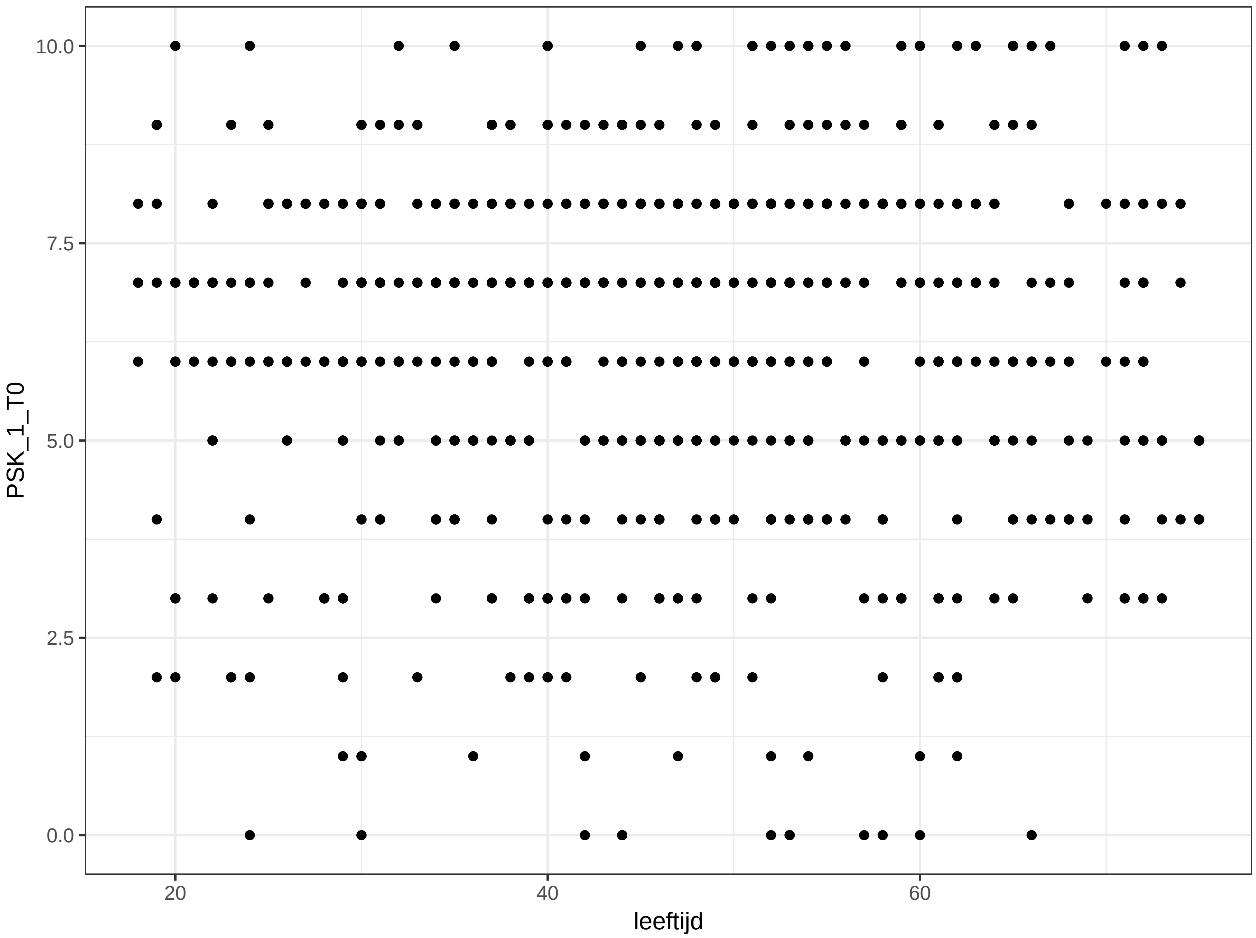
Age

```r
names(data_clean)
```

```
##  [1] "geslacht"            "leeftijd"            "DTF"
##  [4] "VAR00009"            "locatie_klacht"      "Werkstatus"
##  [7] "Werksetting"         "duur_klachten"       "PSK_1_T0"
## [10] "VAR00005"            "ip1_T0"              "ip2_T0"
## [13] "ip3_T0"              "ip4_T0"              "ip5_T0"
## [16] "ip6_T0"              "ip7_T0"              "ip8_T0"
## [19] "ip9_T0"              "pijn_acuut_chronisch" "dissomsc_T0"
## [22] "depsomsc_T0"         "angsomsc_T0"         "somsomsc_T0"
## [25] "dis_ord_T0"          "dep_ord_T0"          "ang_ord_T0"
```

```
## [28] "som_ord_T0"
```

```
data_clean %>%
  ggplot(aes(x = leeftijd,
             y = PSK_1_T0)) +
  geom_point()
```



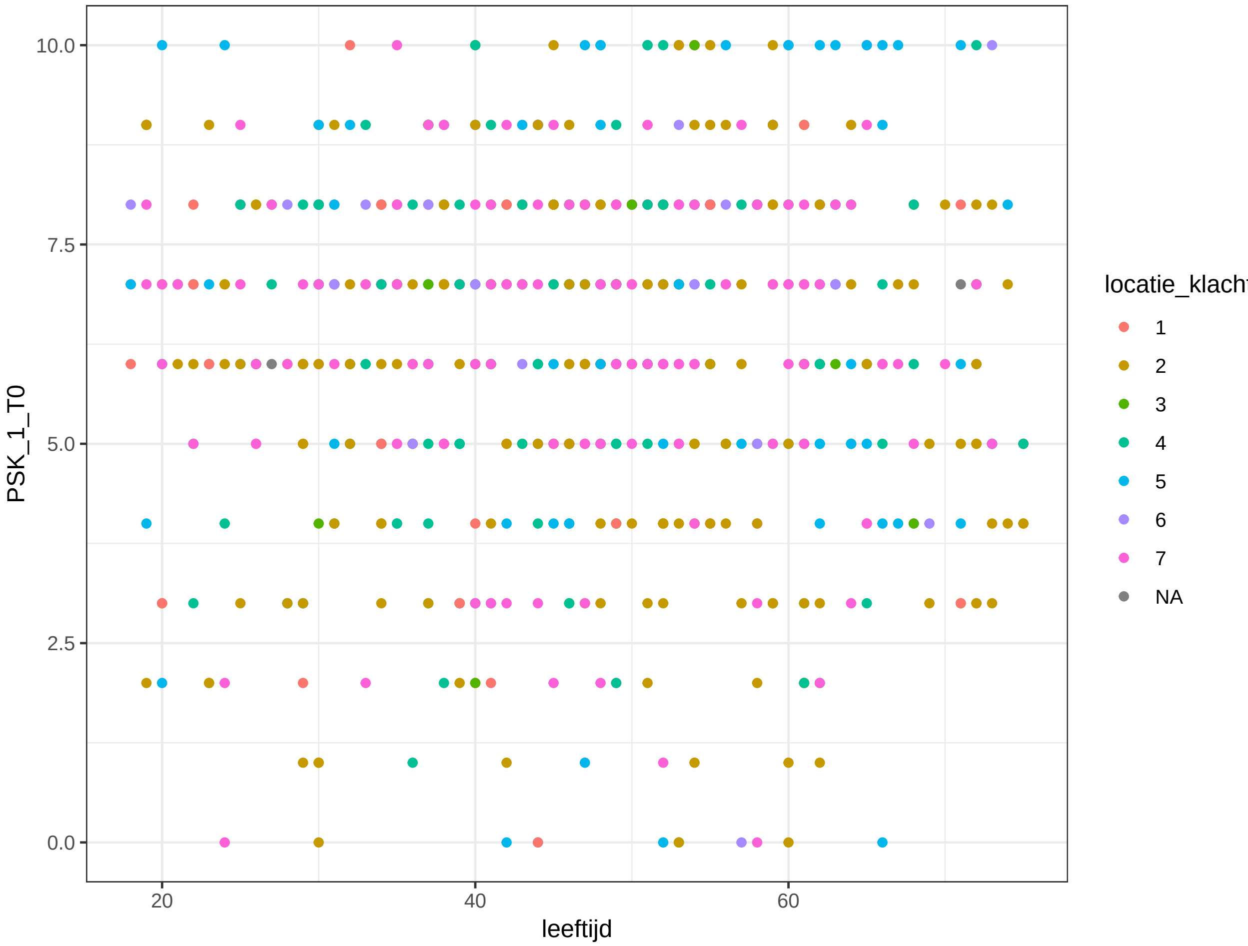
This plot clearly shows that the dependent variable is actually also a cathegorical variable, with a limited amount of discrete outcome values. There is no evident correlation between `PSK_1_T0` and `leeftijd`, which we already learned from the correlation plot.

Let's add some dimensions (vriables) to the dotplot.

```
names(data_clean)
```

```
##  [1] "geslacht"              "leeftijd"              "DTF"
##  [4] "VAR00009"              "locatie_klacht"        "Werkstatus"
##  [7] "Werksetting"           "duur_klachten"         "PSK_1_T0"
## [10] "VAR00005"              "ip1_T0"                "ip2_T0"
## [13] "ip3_T0"                "ip4_T0"                "ip5_T0"
## [16] "ip6_T0"                "ip7_T0"                "ip8_T0"
## [19] "ip9_T0"                "pijn_acuut_chronisch"  "dissomsc_T0"
## [22] "depsomsc_T0"           "angsomsc_T0"           "somsomsc_T0"
## [25] "dis_ord_T0"            "dep_ord_T0"            "ang_ord_T0"
## [28] "som_ord_T0"
```

```
data_clean %>%
  ggplot(aes(x = leeftijd,
             y = PSK_1_T0)) +
  geom_point(aes(colour = locatie_klacht))
```



Again we do not see any evident clustering of equally coloured points, indicating no relationship between the lcation of the medical complaint, age and `PSK_1_T0`

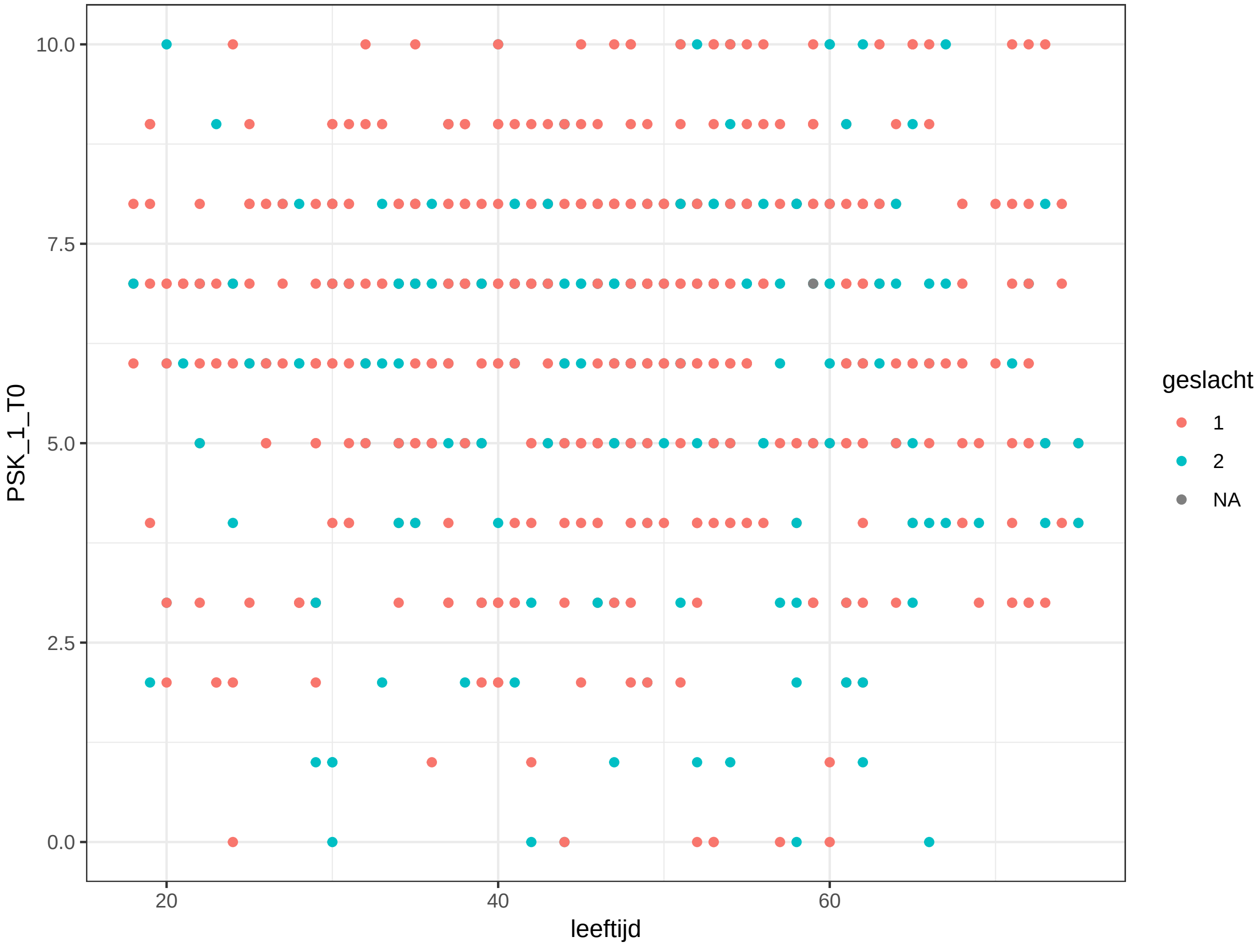
Gender

Maybe study a different relationship? Gender?

```
names(data_clean)
```

```
##  [1] "geslacht"           "leeftijd"           "DTF"
##  [4] "VAR00009"           "locatie_klacht"     "Werkstatus"
##  [7] "Werksetting"        "duur_klachten"      "PSK_1_T0"
## [10] "VAR00005"           "ip1_T0"             "ip2_T0"
## [13] "ip3_T0"             "ip4_T0"             "ip5_T0"
## [16] "ip6_T0"             "ip7_T0"             "ip8_T0"
## [19] "ip9_T0"             "pijn_acuut_chronisch" "dissomsc_T0"
## [22] "depsomsc_T0"        "angsomsc_T0"        "somsomsc_T0"
## [25] "dis_ord_T0"         "dep_ord_T0"         "ang_ord_T0"
## [28] "som_ord_T0"
```

```
data_clean %>%
  ggplot(aes(x = leeftijd,
             y = PSK_1_T0)) +
  geom_point(aes(colour = geslacht))
```



Remarkably, gender "1" seems overrepresented and there is one `NA` in this variable.

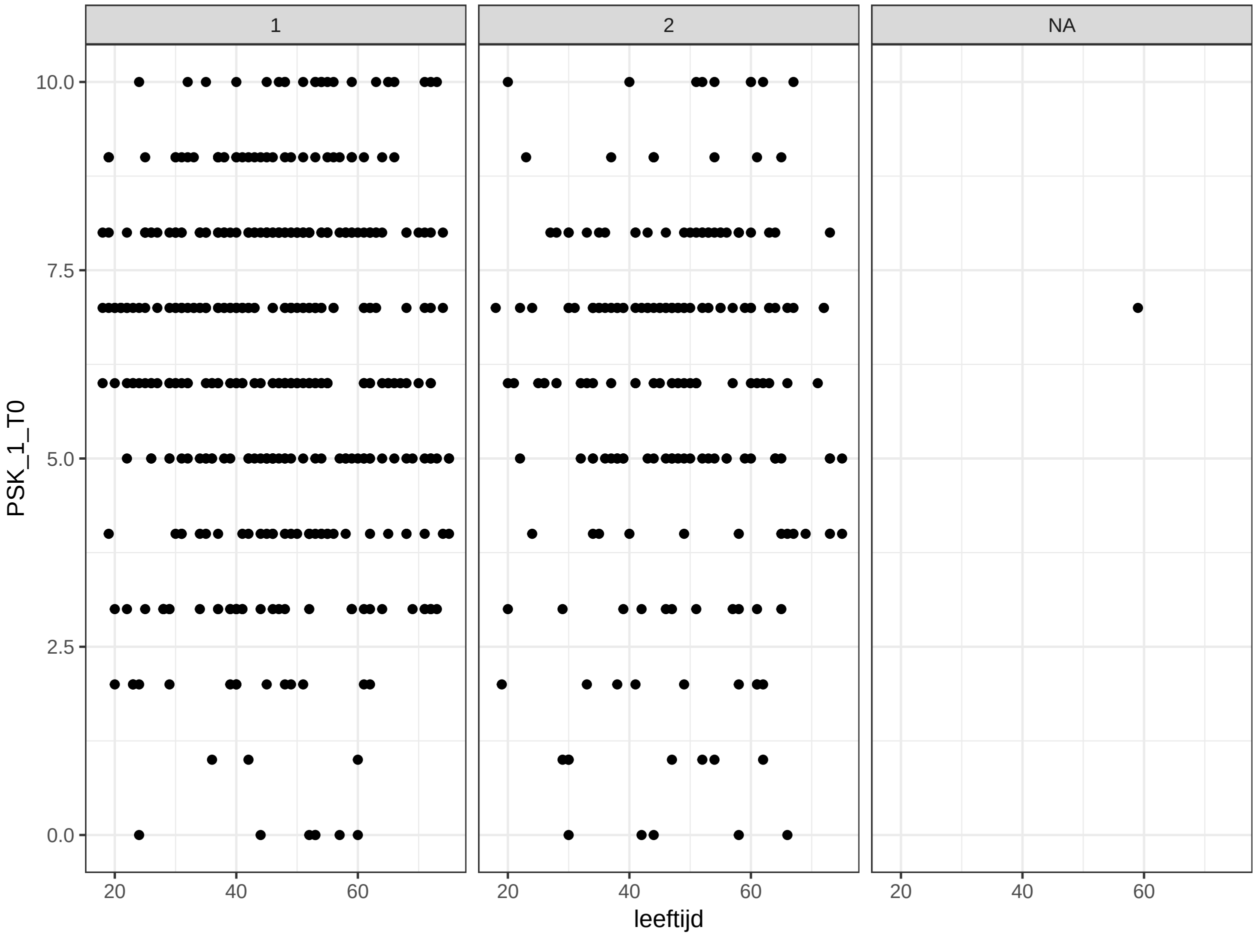
Using facets reveals this more clearly.

```
names(data_clean)
```

```
##  [1] "geslacht"            "leeftijd"            "DTF"
##  [4] "VAR00009"            "locatie_klacht"      "Werkstatus"
##  [7] "Werksetting"         "duur_klachten"       "PSK_1_T0"
## [10] "VAR00005"            "ip1_T0"              "ip2_T0"
## [13] "ip3_T0"              "ip4_T0"              "ip5_T0"
## [16] "ip6_T0"              "ip7_T0"              "ip8_T0"
## [19] "ip9_T0"              "pijn_acuut_chronisch" "dissomsc_T0"
## [22] "depsomsc_T0"         "angsomsc_T0"         "somsomsc_T0"
## [25] "dis_ord_T0"          "dep_ord_T0"          "ang_ord_T0"
## [28] "som_ord_T0"
```

```
data_clean %>%
  ggplot(aes(x = leeftijd,
             y = PSK_1_T0)) +
```

```
  geom_point() +
  facet_wrap(~ geslacht)
```



If we want to learn which gender is coded with which label we can access the label information (`col_data`) again.

```
cold_data_df %>%
  dplyr::filter(name == "geslacht") %>%
  dplyr::select(value) %>%
  unlist
```

```
## value.Vrouw   value.Man
##         "1"         "2"
```

We could repeat this iteration over all variables, bu that would be tedious. Let's concentrate on a cluster of variables `ip_...`

```
col_data
```

```
## $geslacht
## Vrouw   Man
##   "1"   "2"
##
## $leeftijd
## NULL
##
```

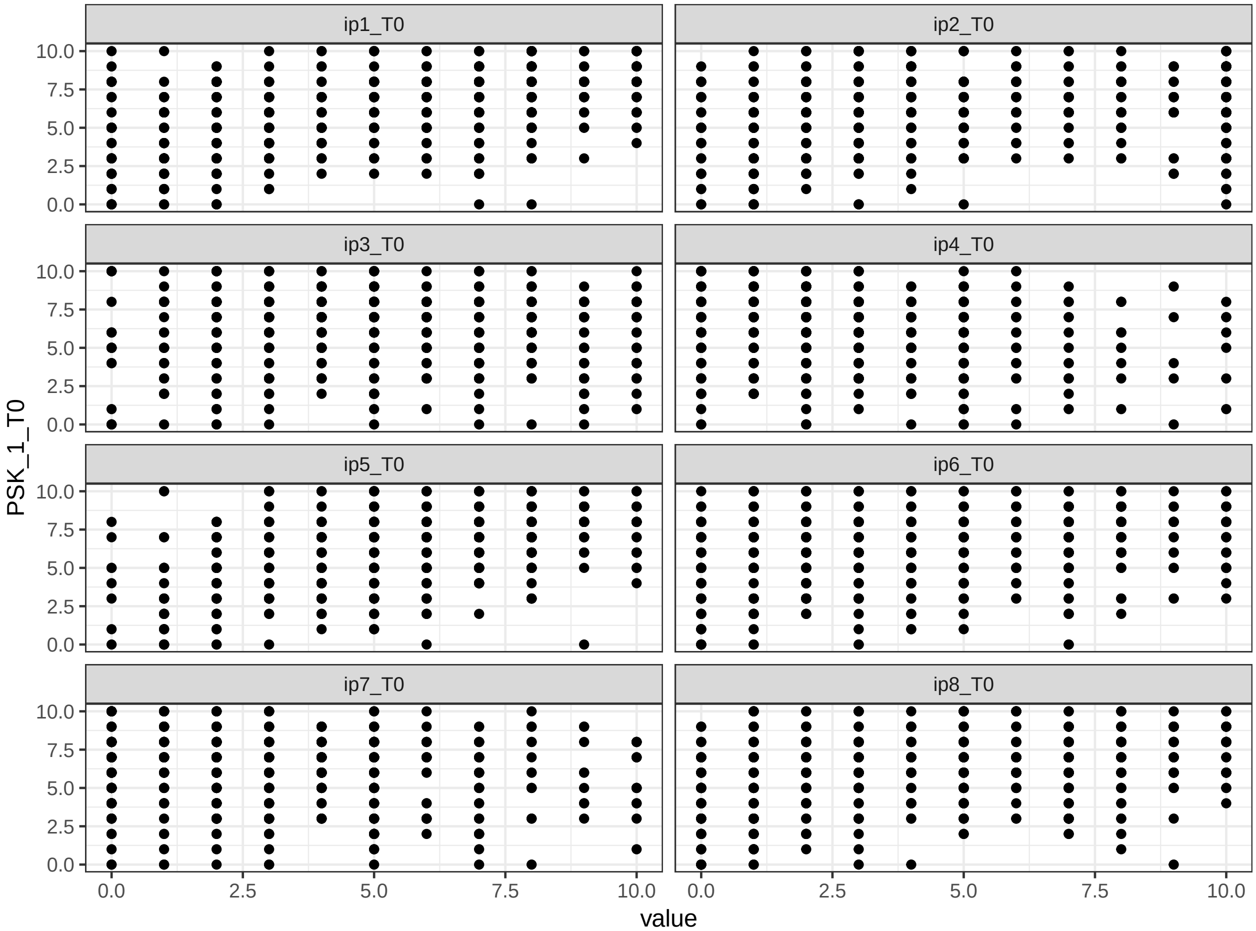
```
## $DTF
##  ja nee
## "1" "2"
##
## $VAR00009
## NULL
##
## $locatie_klacht
##                    hoofd nek,schouder , boven rug      elleboog, pols/hand
##                      "1"                      "2"                      "3"
##                 lage rug               heup, knie               enkel,voet
##                      "4"                      "5"                      "6"
##        meerdere lokaties
##                      "7"
##
## $Werkstatus
##      werkend niet werkend
##          "1"          "2"
##
## $Werksetting
## Eerste lijn Tweede lijn  Derde lijn  Combinatie
##         "1"         "2"         "3"         "4"
##
## $duur_klachten
## [1] "In weken"
##
## $PSK_1_T0
## [1] "NRS hoe moeilijk het voor u was deze activiteit de afgelopen week uit te voeren"
##
## $VAR00005
## NULL
##
## $ip1_T0
## [1] "ip1 Hoeveel beïnvloedt uw ziekte uw leven"
##
## $ip2_T0
## [1] "ip2 Hoe lang denkt u dat uw ziekte zal duren"
##
## $ip3_T0
## [1] "ip3 Hoeveel controle vindt u dat u heeft over uw ziekte"
##
## $ip4_T0
## [1] "ip4 Hoeveel denkt u dat uw behandeling kan helpen bij uw ziekte"
##
## $ip5_T0
## [1] "ip5 Hoe sterk ervaart u klachten door uw ziekte"
##
## $ip6_T0
## [1] "ip6 Hoe bezorgd bent u over uw ziekte"
##
## $ip7_T0
## [1] "ip7 In welke mate vindt u dat u uw ziekte begrijpt"
##
## $ip8_T0
```

```
## [1] "ip8 Hoeveel invloed heeft de ziekte op uw stemming"
##
## $ip9_T0
## [1] "ip9 de drie belangrijkste factoren die naar uw opvatting uw ziekte hebben veroorzaakt"
##
## $pijn_acuut_chronisch
## [1] "pijn_acuut_chronisch"
##
## $dissomsc_T0
## [1] "Distress"
##
## $depsomsc_T0
## [1] "Depressie"
##
## $angsomsc_T0
## [1] "Angst"
##
## $somsomsc_T0
## [1] "Somatisatie"
##
## $dis_ord_T0
## [1] "Distress ordinaal"
##
## $dep_ord_T0
## [1] "Depressie ordinaal"
##
## $ang_ord_T0
## [1] "Angst ordinaal"
##
## $som_ord_T0
## [1] "Somatisatie ordinaal"
```

```r
names(data_clean)
```

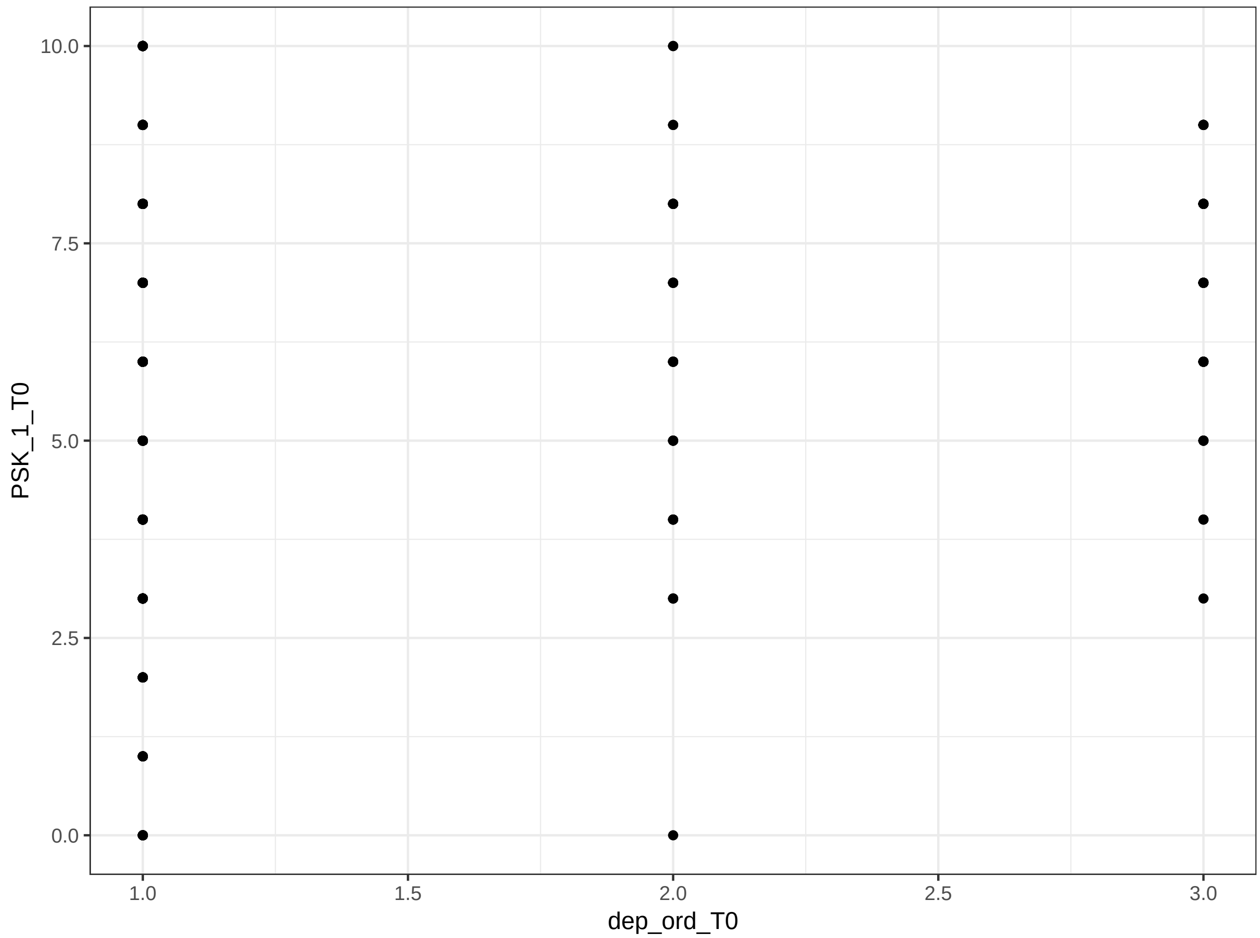
```
##  [1] "geslacht"             "leeftijd"             "DTF"
##  [4] "VAR00009"             "locatie_klacht"       "Werkstatus"
##  [7] "Werksetting"          "duur_klachten"        "PSK_1_T0"
## [10] "VAR00005"             "ip1_T0"               "ip2_T0"
## [13] "ip3_T0"               "ip4_T0"               "ip5_T0"
## [16] "ip6_T0"               "ip7_T0"               "ip8_T0"
## [19] "ip9_T0"               "pijn_acuut_chronisch" "dissomsc_T0"
## [22] "depsomsc_T0"          "angsomsc_T0"          "somsomsc_T0"
## [25] "dis_ord_T0"           "dep_ord_T0"           "ang_ord_T0"
## [28] "som_ord_T0"
```

```r
data_clean %>%
  dplyr::select(PSK_1_T0, ip1_T0:ip8_T0) %>%
  gather(ip1_T0:ip8_T0,
         key = "parameter",
         value = "value") %>%
  ggplot(aes(x = value,
             y = PSK_1_T0)) +
  geom_point() +
  facet_wrap(~ parameter, nrow = 4)
```

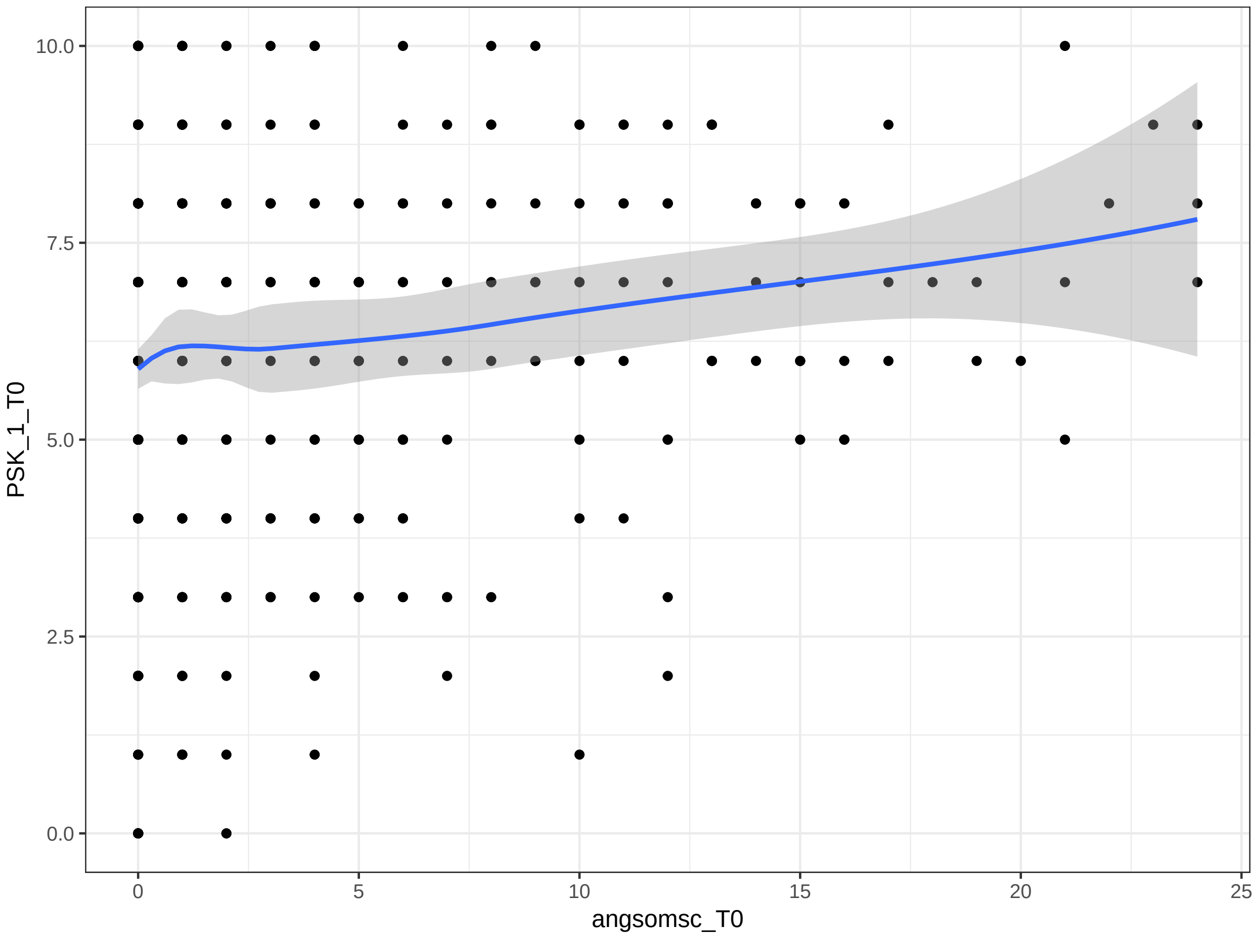


Angst en depressie

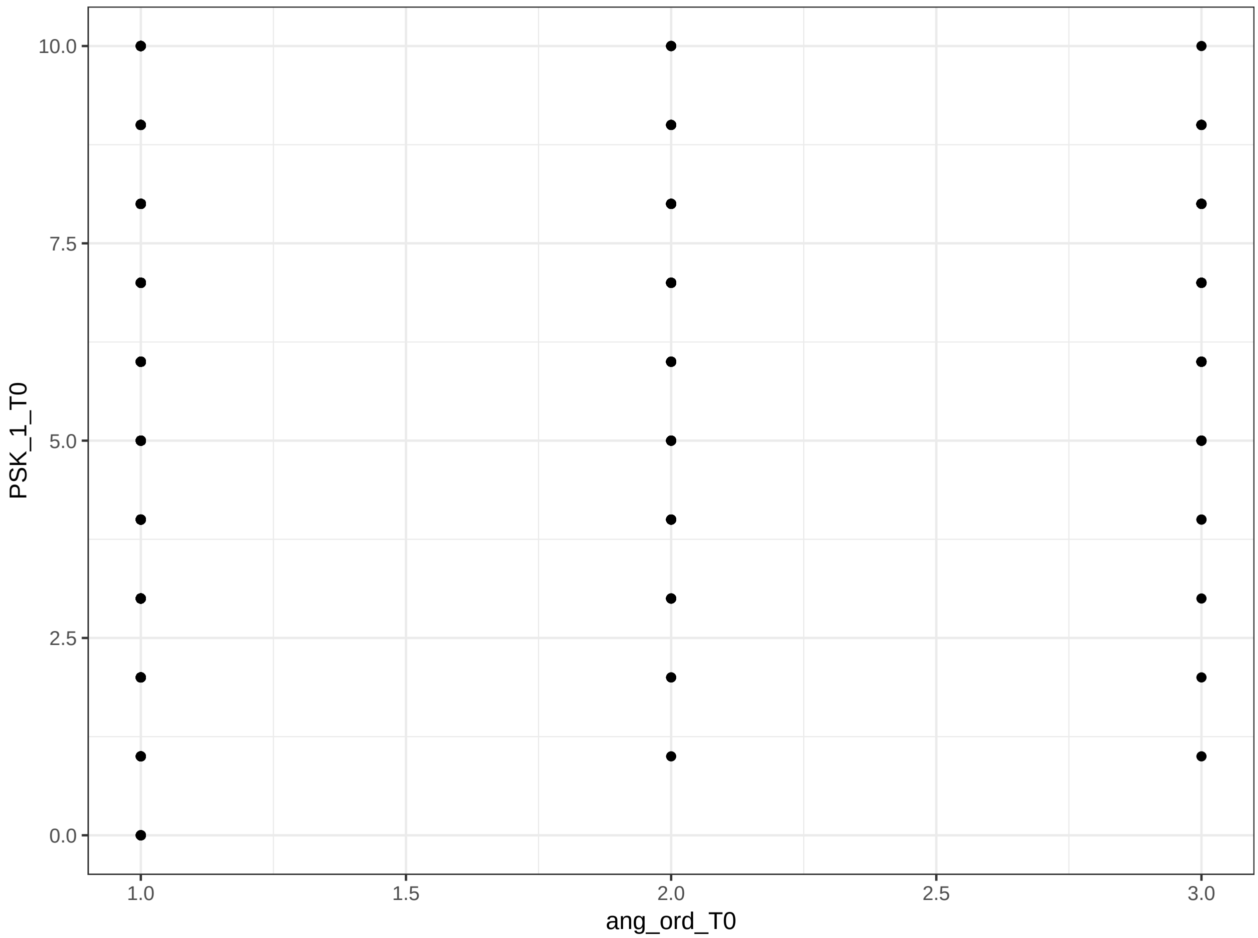
```
data_clean %>%
  ggplot(aes(x = dep_ord_T0,
             y = PSK_1_T0)) +
  geom_point()
```




```
## range scale for fear
data_clean %>%
  ggplot(aes(x = angsomsc_T0,
             y = PSK_1_T0)) +
  geom_point() +
  geom_smooth()
```



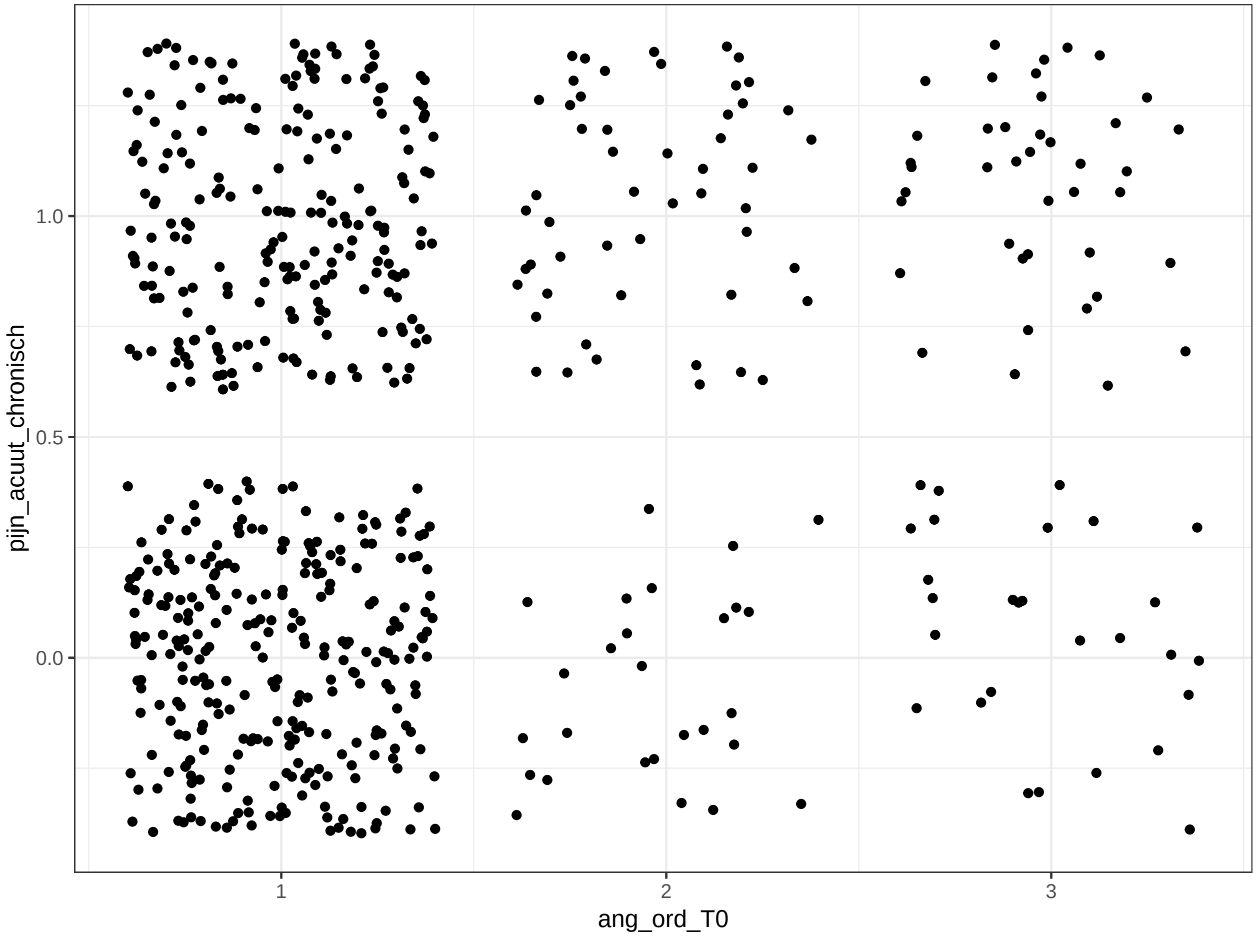

```
## ordinal scale for fear
data_clean %>%
  ggplot(aes(x = ang_ord_T0,
             y = PSK_1_T0)) +
  geom_point()
```



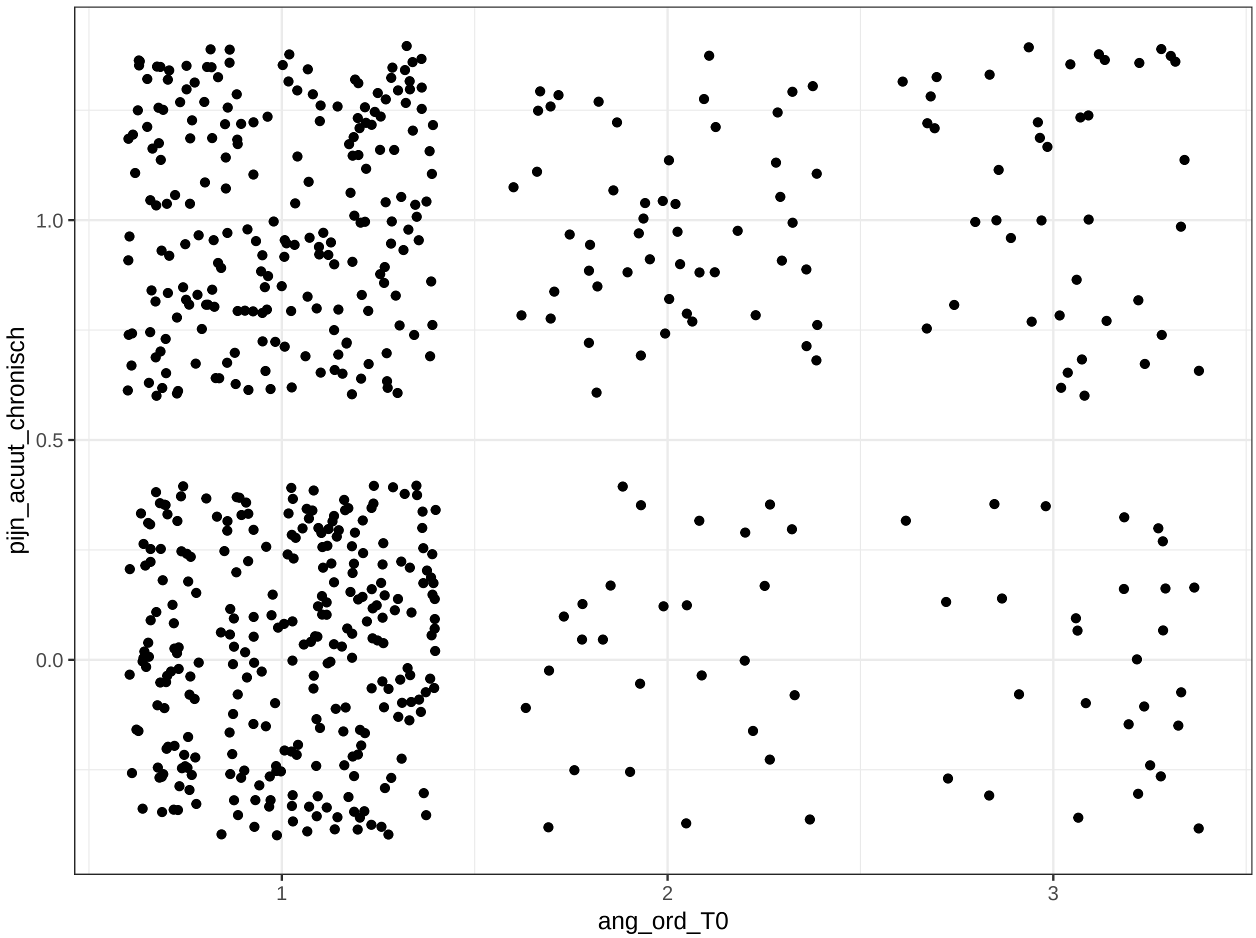

```
## the inital trend of positive correlation between anxiety and PSK disappears
```

Pain

```
data_clean %>%
  ggplot(aes(x = ang_ord_T0,
             y = pijn_acuut_chronisch)) +
  geom_point(position = "jitter")
```

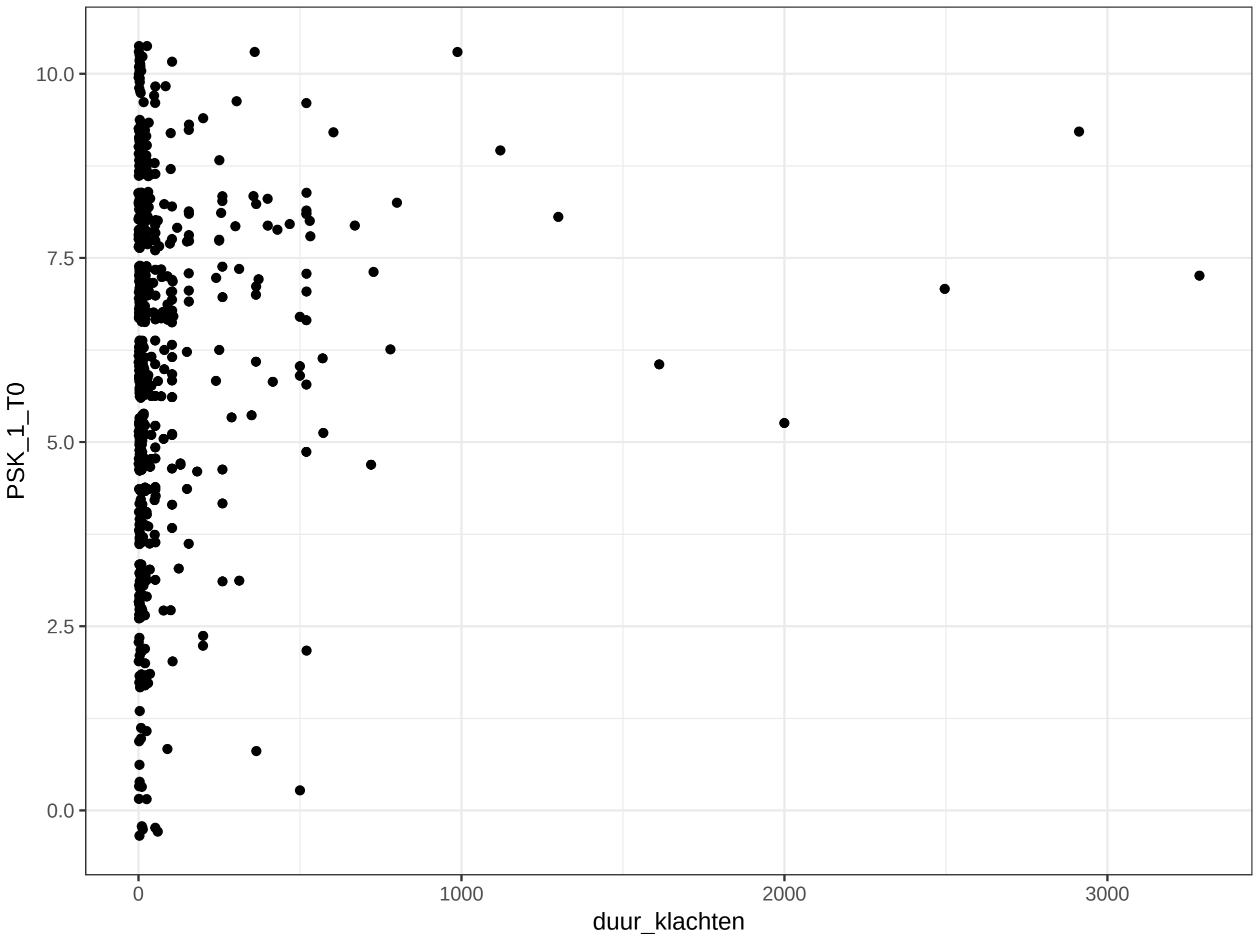



```r
data_clean %>%
  ggplot(aes(x = ang_ord_T0,
             y = pijn_acuut_chronisch)) +
  geom_point(position = "jitter")
```



Duration of clinical complaints

```
data_clean %>%
  ggplot(aes(x = duur_klachten,
             y = PSK_1_T0)) +
  geom_point(position = "jitter")
```



Descriptive statistics