Statistical Analysis of Proteomics data

Bernd Klaus and Frank Stein

24 August 2018

Contents

1 Required packages and other preparations

```
library(BiocStyle)
library(knitr)
library(Biobase)
library(geneplotter)
library(ggplot2)
library(reshape2)
library(plyr)
library(dplyr)
library(gplots)
library(RColorBrewer)
library(arrayQualityMetrics)
library(stringr)
library(matrixStats)
library(genefilter)
library(limma)
library(openxlsx)
library(smoothmest)
library(tidyr)
library(vsn)
library(MSnbase)
library(pheatmap)
library(fdrtool)
library(purrr)
library(tidyverse)
library(Hmisc)
library(gridExtra)
library(Peptides)
library("genefilter")

glog2 <- function(x) ((asinh(x)-log(2))/log(2))

# fcuntion to retrieve annotation data from Uniprot

getUniprotGoodies  <- function(query) # columns)
{
    ## query and columns start as a character vectors
    qstring <- paste(query, collapse="+or+")
    #cstring <- paste(columns, collapse=",")
    uri <- 'http://www.uniprot.org/uniprot/?query='
    fullUri <- paste0(uri,qstring,'&format=tab')#&columns='),cstring)
    dat <- read.delim(fullUri, stringsAsFactors=FALSE)
    ## now remove things that were not in the specific original query...
    dat <- dat[dat[,1] %in% query,]
    dat
}


plotProtein <- function(Acc){
```

```
    tmp <- exprs(time_data_vsn)[Acc, ]
    pl <- (qplot(time, tmp, color = time, main = Acc)
           + ylab("protein_exp_log2"))
    pl +  scale_color_brewer(type="qual", palette=2)
    }


# function to compute a robust mean
robustMean <- function(x){
    if(length(x) == 1){return(x)}
    else{
        return(smhuber(x)$mu)
    }
}
```

2 Overview of a typical proteomics experiment

A typical proteomics experiment follows the steps depicted in the Figure below.

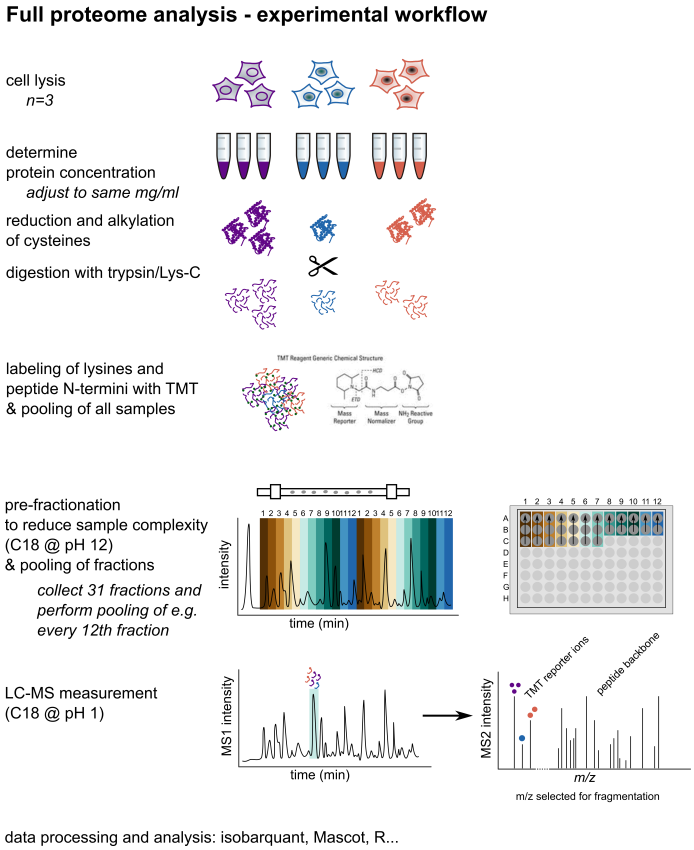


Figure 1: Workflow of a proteomics experiment

This is an example of a full proteome analysis of three different cell lines. The first cell line is the wildtype of HEK 293T (wt), the second cell line is a CRISPR knock-out (ko) of Sphingosine-1-phosphate lyase 1 (SPGL1 - single_ko) and the third cell line is a double CRISPR knock-out of Ceramide glucosyltransferase (UGCG) and SPGL1 (double_ko). For each biological conditions, three independent replicates were generated and analyzed as shown in the figure above.

In our case, the raw output files of the mass spectrometer were processed with IsobarQuant (Franken et. al., 2015), a software that was developed at Cellzome.

3 Processing strategy

3.1 Raw data

IsobarQuant gives three text files as an output, a summary.txt file, a peptides.txt file and a protein.txt file. The peptides.txt and proteins.txt files contain the intensities that were quantified for the different tmt channels. The peptides.txt file contain the raw intentisies for each identified peptide ("sig") whereas the proteins.txt file contains already summarized values for each protein for which the individual peptide tmt signals belonging to a particular protein were summed up ("signal_sum").

3.2 Use of intensies instead of ratios

We use the intensities directly here using the following steps:

1.) We first perform a glog transformation on the raw protein–intensities. The glog is very similar to a standard log transform, but performs a less severe shrinkage for low intensities. The figure below illustrates this point:

2.) Following this, we apply the vsn algorithm to all the summarized values on the protein level on the raw intensity scale (We exponentiate the summarized values again). vsn essentially centers and scales the values within a channel and the performs a glog transformation.

3.) We finally run *limma* on the *vsn* normalized data.

This is very similar to the strategy followed in Hughes et. al., 2014, although there *vsn* is applied on the peptide level data and the the protein group summarization is performed later on.

3.3 Some reasons why it is better to not work with ratios

Although ratios are widely used in proteomics, from a statistical point of view, it is better to use raw intensities values as their computing a ratio beforehand changes the distribution and especially the variance of your data dramatically: by dividing by a large intensity, small intensity values will be compressed, dividing by a small intensity will boost the values.

Also, once ratios have been formed, it is not easy to go back, thus essentially throwing data away.

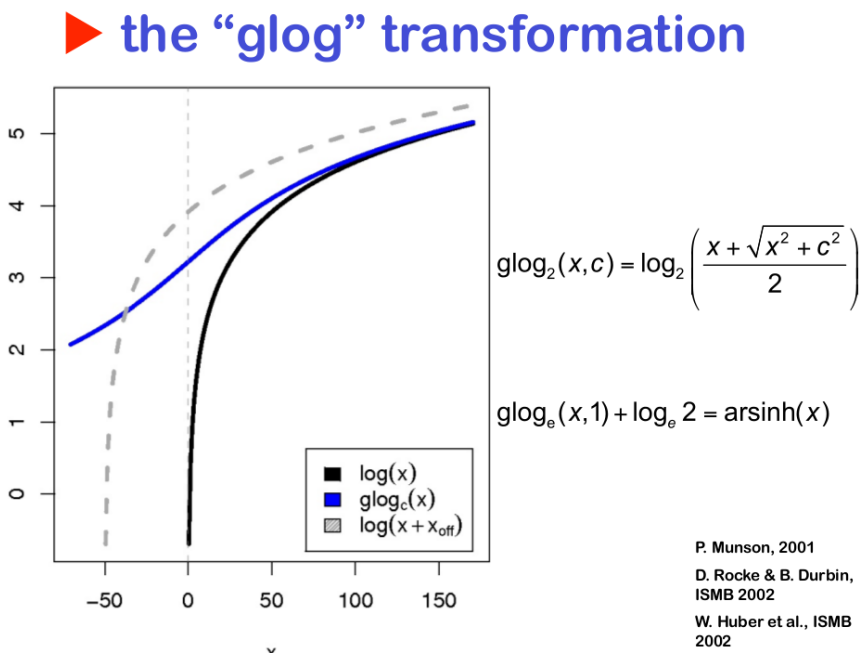


Figure 2: glog2

On top of that, if your experimental design becomes more complex, you can use full featured linear models (such as we do here) more easily instead of just simple two–group comparisons. Furthermore, one can find batch effects and the like more readily in the data.

4 Experimental design

In the data set at hand, we have single run with 10 TMT labels. This labels cover 3 conditions with 3 replicates in each condition.

We first import the table holding the experimental design.

```
expDesign <- as.data.frame(read_csv("metadata.csv"))
   Parsed with column specification:
   cols(
     tmt.label = col_character(),
     rep = col_character(),
     condition = col_character()
   )
rownames(expDesign) <- expDesign$tmt.label

expDesign
        tmt.label  rep condition
   126        126 rep1        wt
   127L      127L rep1 single_ko
   127H      127H rep1 double_ko
   128L      128L rep2        wt
   128H      128H rep2 single_ko
   129L      129L rep2 double_ko
   129H      129H rep3        wt
   130L      130L rep3 single_ko
   130H      130H rep3 double_ko
   131L      131L <NA>      <NA>
```

In the analysis below, we will only use the *wt* and *single_ko* conditions in order to keep it simple.

5 Import of the raw data

We first import the raw data, the protein quantifications saved as tab–separated file, with one column per label as well as the peptide level data. We extract the samples fraction from the column `source file` in the peptide table.

```
load("peptide_data.RData")

peptide_data
   # A tibble: 348,352 x 38
      protein_group_no protein_id sequence modifications    mw precursor_mz
                 <int> <chr>      <chr>    <chr>         <dbl>        <dbl>
    1                1 P30613     SQRDLAK  TMT10plex K7; 1046.         524.
    2                1 P30613     EMIKAGM~ TMT10plex N-~ 1691.         424.
    3                1 P30613     GPEIR    TMT10plex N-~  799.         401.
    4                1 P30613     GDLGIEI~ TMT10plex N-~ 1599.         800.
    5                1 P30613     GDLGIEI~ TMT10plex N-~ 1599.         800.
    6                1 P30613     GDLGIEI~ TMT10plex N-~ 1599.         800.
```

```
 7                1 P30613     GDLGIEI~ TMT10plex N-~ 1599.         800.
 8                1 P30613     GDLGIEI~ TMT10plex K1~ 1370.         686.
 9                1 P30613     GDLGIEI~ TMT10plex N-~ 1599.         534.
10                1 P30613     GDLGIEI~ TMT10plex N-~ 1599.         800.
# ... with 348,342 more rows, and 32 more variables: charge_state <int>,
#   ppm_error <dbl>, score <int>, fdr_at_score <dbl>, `delta mod` <dbl>,
#   rank <int>, msms_id <int>, source_file <chr>, `retention time` <dbl>,
#   peak_intensity <dbl>, s2i <dbl>, p2t <dbl>, is_unique <int>,
#   in_quantification_of_protein <int>, in_protein_inference <dbl>,
#   in_top3 <int>, seq_start <int>, seq_end <int>, is_decoy <int>, `prior ion
#   ratio` <dbl>, `least squares` <dbl>, `ms1 source` <chr>, sig_126 <dbl>,
#   sig_127L <dbl>, sig_127H <dbl>, sig_128L <dbl>, sig_128H <dbl>,
#   sig_129L <dbl>, sig_129H <dbl>, sig_130L <dbl>, sig_130H <dbl>,
#   sig_131L <dbl>

peptide_data$fraction <- gsub("(.+R1_)|(.raw$)", "", peptide_data$source_file)
peptide_data$fraction <- factor(peptide_data$fraction,
                                ordered = TRUE,
                                levels = c("F01","F02","F03","F04","F05",
                                           "F06","F07","F08","F09","F10",
                                           "F11","F12"))

protein_data <- read_tsv("H0027_student_PEP_merged_results_20180202_1618_proteins.txt")
  Parsed with column specification:
  cols(
    .default = col_double(),
    protein_group_no = col_integer(),
    protein_id = col_character(),
    description = col_character(),
    gene_name = col_character(),
    mw = col_character(),
    max_score = col_integer(),
    total_score = col_integer(),
    ssm = col_integer(),
    upm = col_integer(),
    fqssm = col_integer(),
    qssm = col_integer(),
    qupm = col_integer(),
    reference_label = col_integer()
  )
  See spec(...) for full column specifications.

protein_data
  # A tibble: 9,586 x 45
     protein_group_no protein_id description gene_name mw     top3 protein_fdr
                <int> <chr>      <chr>       <chr>     <chr> <dbl>       <dbl>
 1                1 P30613     Pyruvate k~ PKLR      6677~  5.40    0
 2                2 P23919     Thymidylat~ DTYMK     2764~  8.21    0
 3                3 P40425     Pre-B-cell~ PBX2      5117~  6.08    0
 4                4 P40429     60S riboso~ RPL13A    3003~  9.01    0
 5                5 F5GZF0|O1~ Cyclin-dep~ CDK2AP1   8136~  6.56    0
```

```
 6               6 A0A087WT5~ Transthyre~ TTR       2219~  6.80    0.00640
 7               7 P35251|P3~ Replicatio~ RFC1      1651~  7.67    0
 8               8 C9JBG0|Q9~ Ras-relate~ RAB34     2766~  7.04    0.000678
 9               9 I3L1H5|K7~ Diphthamid~ DPH1      5117~  7.28    0
10              10 P53701     Cytochrome~ HCCS      3510~  8.14    0
# ... with 9,576 more rows, and 38 more variables: max_score <int>,
#   total_score <int>, ssm <int>, upm <int>, fqssm <int>, qssm <int>,
#   qupm <int>, reference_label <int>, signal_sum_126 <dbl>,
#   signal_sum_127L <dbl>, signal_sum_127H <dbl>, signal_sum_128L <dbl>,
#   signal_sum_128H <dbl>, signal_sum_129L <dbl>, signal_sum_129H <dbl>,
#   signal_sum_130L <dbl>, signal_sum_130H <dbl>, signal_sum_131L <dbl>,
#   rel_fc_126 <dbl>, rel_fc_127L <dbl>, rel_fc_127H <dbl>, rel_fc_128L <dbl>,
#   rel_fc_128H <dbl>, rel_fc_129L <dbl>, rel_fc_129H <dbl>, rel_fc_130L <dbl>,
#   rel_fc_130H <dbl>, rel_fc_131L <dbl>, delta_conf_126 <dbl>,
#   delta_conf_127L <dbl>, delta_conf_127H <dbl>, delta_conf_128L <dbl>,
#   delta_conf_128H <dbl>, delta_conf_129L <dbl>, delta_conf_129H <dbl>,
#   delta_conf_130L <dbl>, delta_conf_130H <dbl>, delta_conf_131L <dbl>
```

6 Quality control analysis of workflow steps

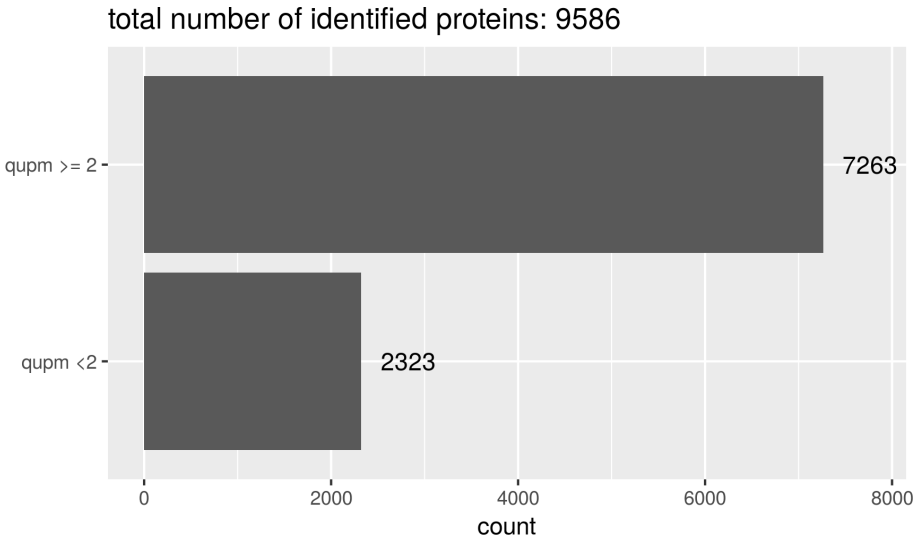
In the following analysis we will analyze the `peptides.txt` file and the `protein.txt` file in various ways in order to assess the results of each individual step of the experimental workflow.

6.1 Protein counts

The total number of identified proteins is useful to judge the overall quality of the experiment. This includes both the success of the cell lysis as well as the performance of the mass spectrometer.

The variable `qupm` indicates the number of peptides quantified per protein. We inspect how many proteins have at least 2 uniquely quantified peptides.

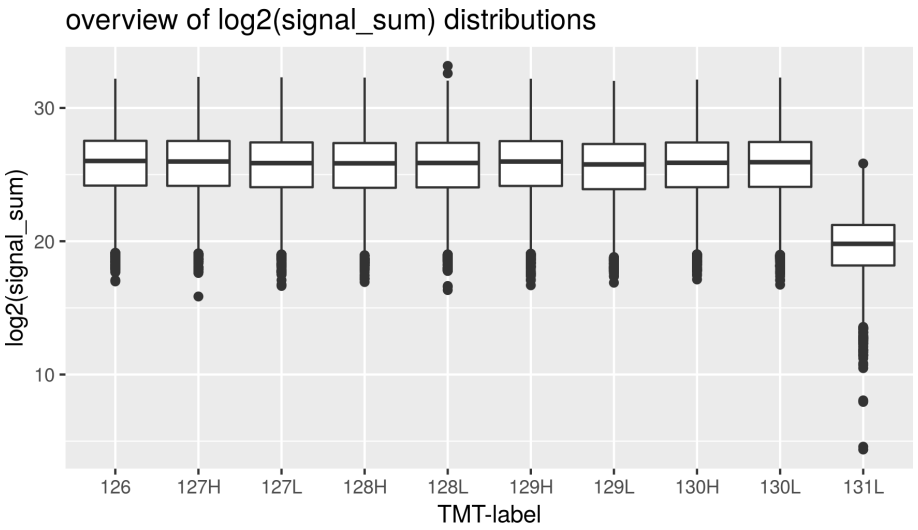
```
ggplot(data = protein_data, aes(ifelse(qupm >= 2, "qupm >= 2", "qupm <2"))) +
  geom_bar() +
  xlab("") +
  geom_text(stat='count', aes(label = ..count..), nudge_y = 500) +
  ggtitle(paste("total number of identified proteins:", nrow(protein_data))) +
  coord_flip()
```



6.2 Measurement of protein concentration

Prior to TMT labeling, the protein concentration of each sample is measured and the protein amount for the proteomics experiment might be adapted if needed. The distribution of the signal sums of all proteins for each TMT channel is a good estimate of the protein concentration in each sample.

```
sub <- protein_data %>%
  dplyr::select(starts_with("signal_sum")) %>%
  gather()

ggplot(data = sub,aes(gsub("signal_sum_","",key), log2(value))) +
  geom_boxplot() +
  xlab("TMT-label") + ylab("log2(signal_sum)") +
  ggtitle("overview of log2(signal_sum) distributions")
```



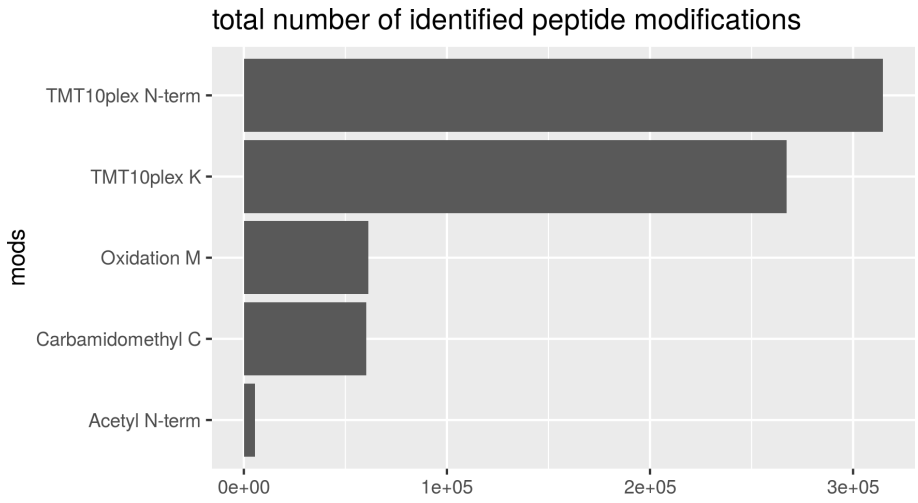
```
rm(sub)
```

We can see that label 131L has a very low signal intensity compared to the rest of the channels. This is expected, as this label was not used in the experiment.

6.3 Reduction and alkylation of cysteines

The 'modifications' column of the peptides.txt output file of IsobarQuant contains an overview of the identified modifications for each peptide. The count statistic of the various modifications enables us to judge both the reduction and alkylation step of Cysteines, as well as the labeling with the TMT reagents.

```
mods <- peptide_data$modifications
mods <- gsub("[0-9]+;",";",mods)
mods <- gsub(";$","",mods)
mods <- unlist(strsplit(mods,split = "; "))
mods <- na.omit(mods)

qplot(mods,geom = "bar") +
  coord_flip() +
  ggtitle("total number of identified peptide modifications")
```



Cysteines are first reduced with Dithiothreitol (DTT) and then alkylated with Iodoacetamide (IAA) in order to destroy disulfide bridges between them. They would otherwise link peptides together.

The reduction and alkylation results in a carbamidomethyl group attached to the sulphur of the cysteine side chains. This is a peptide modification that is included into the mascot search. Therefore, this modification can be found in the 'modifications' column. Missing carbamidomethyl modification could indicate problems with the reduction and alkylation step of the protocol.

Furthermore, TMT labels are identified on the N-terminus as well as on lysine side-chains. This is because they are attached via an succinimidylester which is reactive to amino groups. Those are present on the N-terminus of peptides as well as on the lysine side chains.

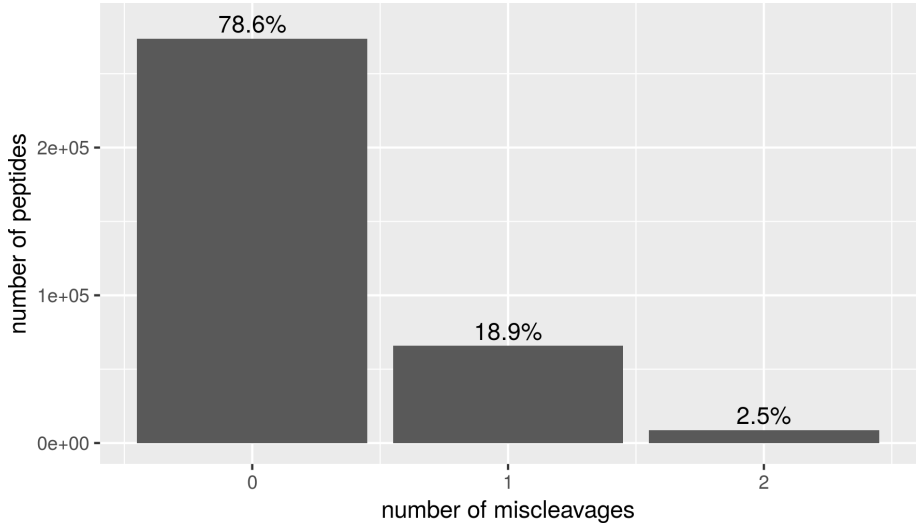
6.4 Tryptic digest

Trypsine is used to digest the proteins into smaller peptides. The enzyme cleaves after an arginine (R) or a lysine (K).

From the identified sequences in the peptides table, we can count how often an arginine or a lysine appears inside the sequence.

We interpret this the number of miscleavages as a performance measure for the tryptic digest.

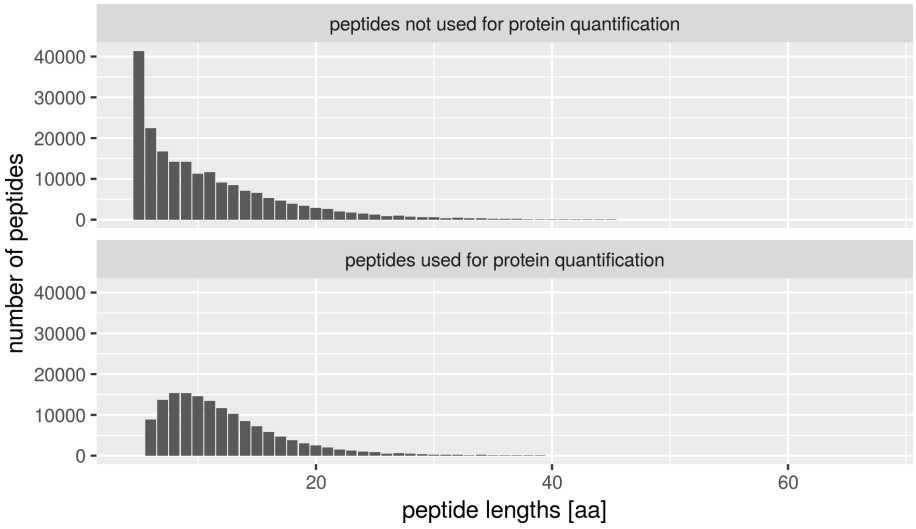
```
peptide_data <- peptide_data %>%
  mutate(sequence = toupper(sequence)) %>%
  mutate(no.of.miscleavages = gsub("[K,R]$","",sequence)) %>%
  mutate(no.of.miscleavages = gsub("[^KR]","",no.of.miscleavages)) %>%
  mutate(no.of.miscleavages = nchar(no.of.miscleavages))

bar_pl <- ggplot(data = peptide_data, aes(x = no.of.miscleavages)) +
  stat_count()

bar_data <- layer_data(bar_pl, 1) %>%
  mutate(prop_rounded = paste0(round(prop, 3)*100, "%"))

ggplot(data = bar_data, aes(x = x, y = y)) +
  geom_bar(stat = "identity") +
  geom_text(mapping = aes(x = x, y = y, label = prop_rounded),
            nudge_y = 10000) +
  xlab("number of miscleavages") +
  ylab("number of peptides")
```



Furthermore, the distribution of peptide lengths is another way to judge the quality of the tryptic digest. The minimum peptide length required for quantification is 5, so peptides used in the quantification step should typically be longer.

```
ggplot(data = peptide_data, aes(nchar(sequence))) +
  geom_bar() +
  xlab("peptide lengths [aa]") +
```

```
  ylab("number of peptides") +
  facet_wrap(~ifelse(in_quantification_of_protein == 1,
                     "peptides used for protein quantification",
                     "peptides not used for protein quantification"),
             ncol = 1)
```

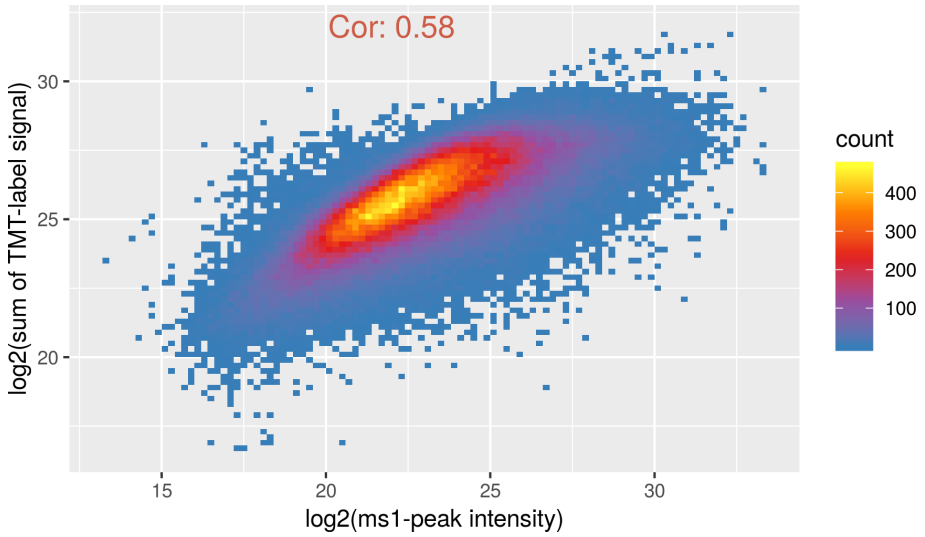


6.5 TMT based quantification vs MS 1 peak

In order to judge the TMT labeling efficiency, one can correlate the ms1 peak intensities with the sum of all tmt signals for each identified peptide.

Those two values should correlate. We first compute the sum the TMT labels and then create a scatterplot of this and the MS1 peak.

```
peptide_data$sig_sum <- peptide_data %>%
  dplyr::select(starts_with("sig")) %>%
  apply(1, sum, na.rm = TRUE)

scp <- ggplot(data = subset(peptide_data,in_quantification_of_protein == 1),
       mapping = aes(log2(peak_intensity), log2(sig_sum))) +
  stat_bin2d(binwidth = 0.2) +
  scale_fill_gradientn(colours = c("#377eb8","#984ea3","#e41a1c",
                                   "#ff7f00","#ffff33")) +
  xlab("log2(ms1-peak intensity)") +
  ylab("log2(sum of TMT-label signal)")

scp +
  annotate("text", label = paste0("Cor: ",
                                  round(cor(layer_data(scp)$x,
                                            layer_data(scp)$y,
                                            method = "spearman"), 2)),
           x = 22, y = 32, size = 5, colour = "coral3")
```

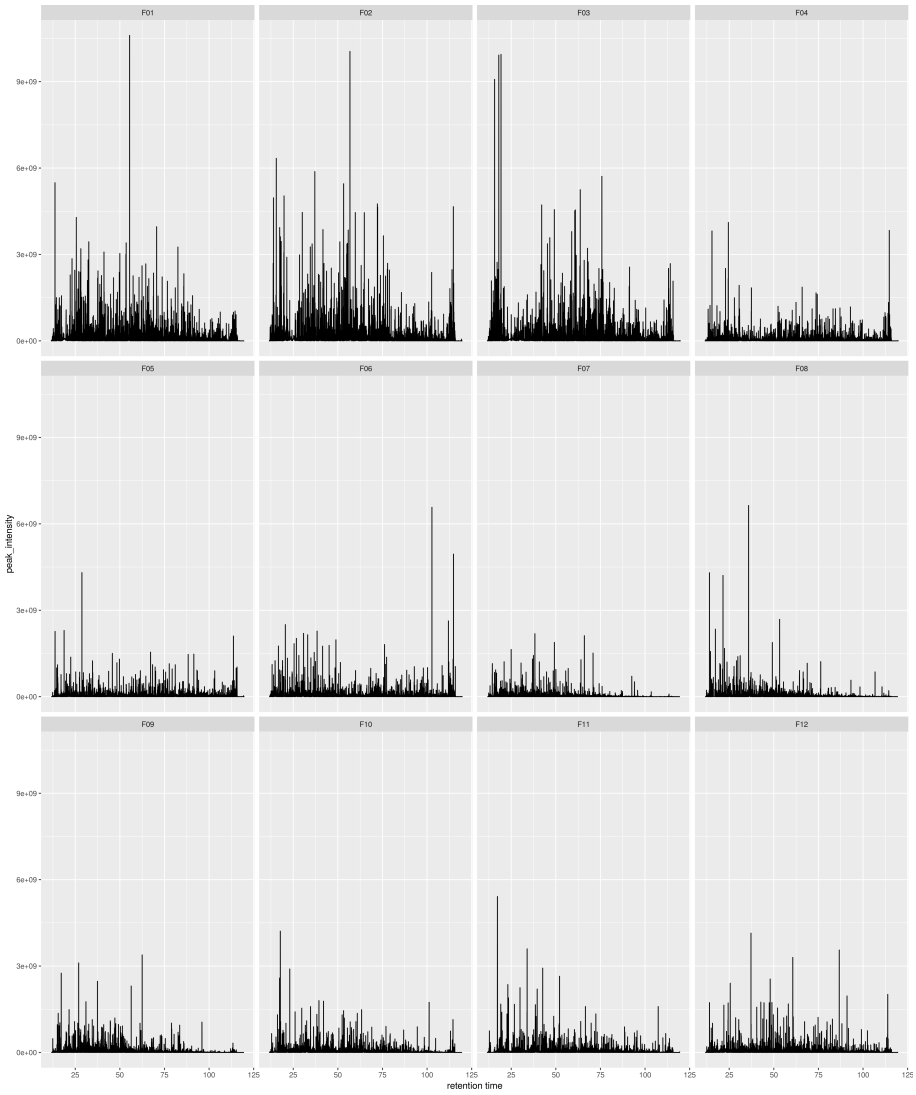


We can see that these values are reasonably correlated to each other. Otherwise, this could indicate problems with the TMT labeling.

6.6 Prefractionation and chromatography

Prefractionation of the input samples, results in 12 different fractions. Each of them was injected into the mass spectrometer separetely producing a ms1 spectrum, which we plot now:

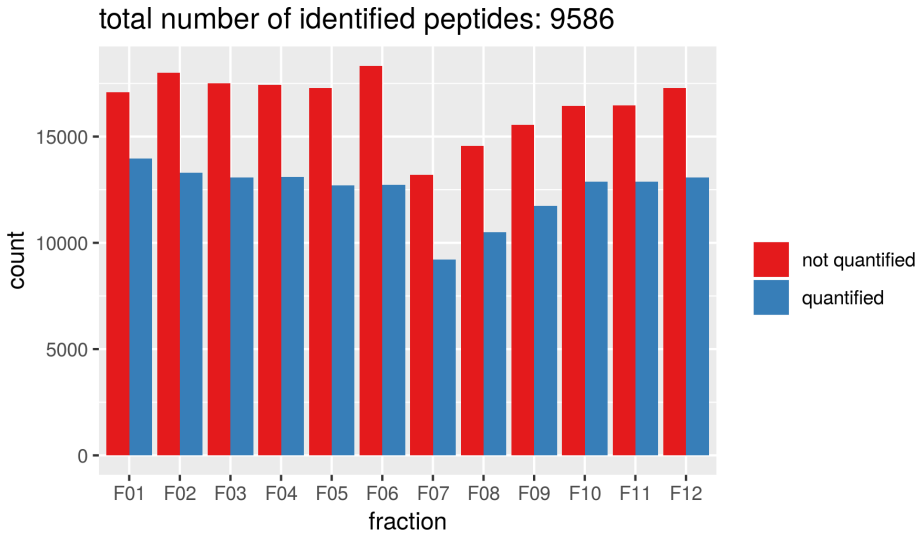
```
ggplot(data = subset(peptide_data, in_quantification_of_protein == 1),
       aes(`retention time`, peak_intensity)) +
  geom_line() +
  facet_wrap(~fraction)
```



In order to judge the performance of the instrument, we plot the number of peptide identifications in the different fractions.

We also distinguish between peptides used / not used for the final quantification.

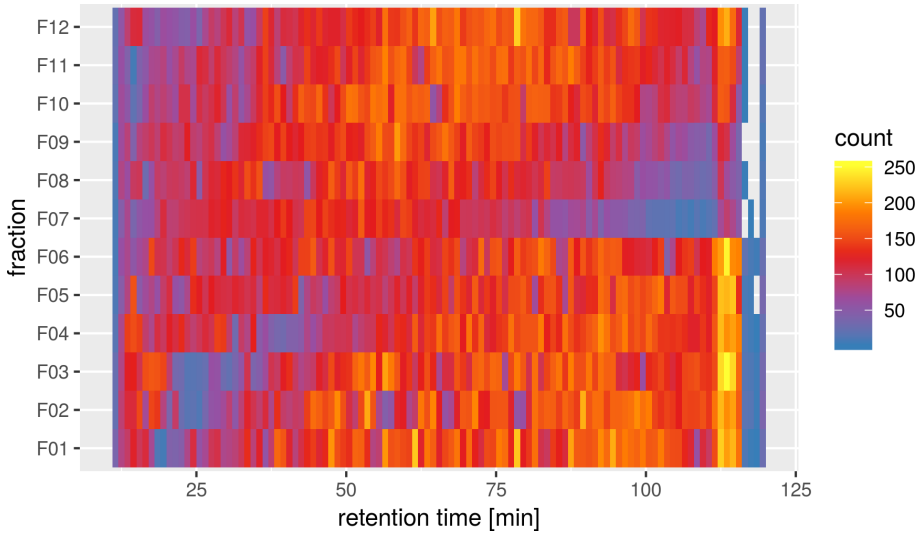
```
ggplot(data = peptide_data,
       aes(fraction, fill = ifelse(in_quantification_of_protein == 1,
                                   "quantified","not quantified"))) +
       geom_bar(position = position_dodge()) +
       xlab("fraction") +
       ggtitle(paste("total number of identified peptides:",
                     nrow(protein_data))) +
       scale_fill_brewer(palette = "Set1", name = "")
```



Fractions 7, 8 and 9 show smaller numbers of identified peptides. Here, the fractionation could possibly be optimized.

For each mass spec run, the peptides are seperated with a C18 column. The next plot shows the frequency of which the peptides are eluted from the column in each fraction.

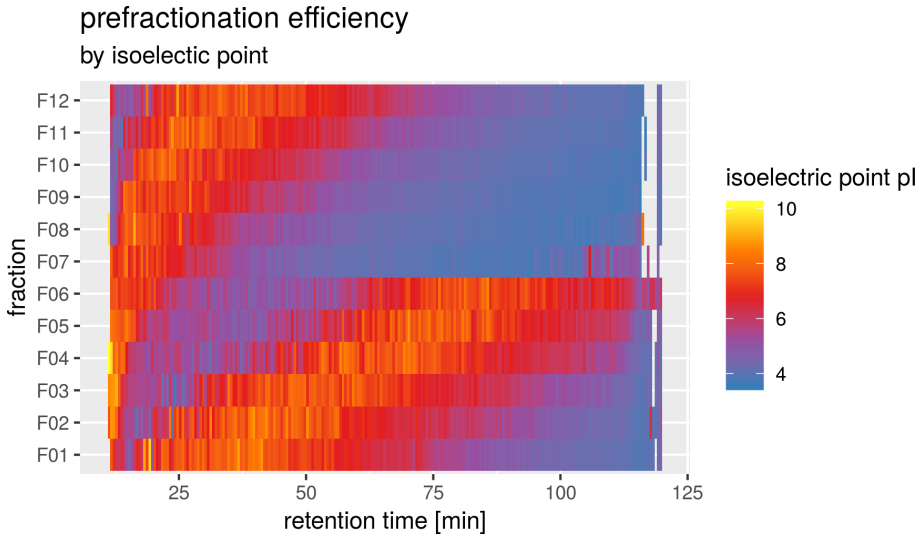
```
ggplot(data = subset(peptide_data,in_quantification_of_protein == 1),
       aes(`retention time`, fraction)) +
  stat_bin2d(binwidth = 1) +
  scale_fill_gradientn(colors = c("#377eb8","#984ea3","#e41a1c",
                                  "#ff7f00","#ffff33")) +
  xlab("retention time [min]")
```



The prefractionation took place at a much higher pH, resulting in the deprotonation of certain peptide residues. Therefore, the peptides were seperated based on their isoelectric point (pI). The isolectric point indicates the pH on which the peptide has a neutral net charge. Changing

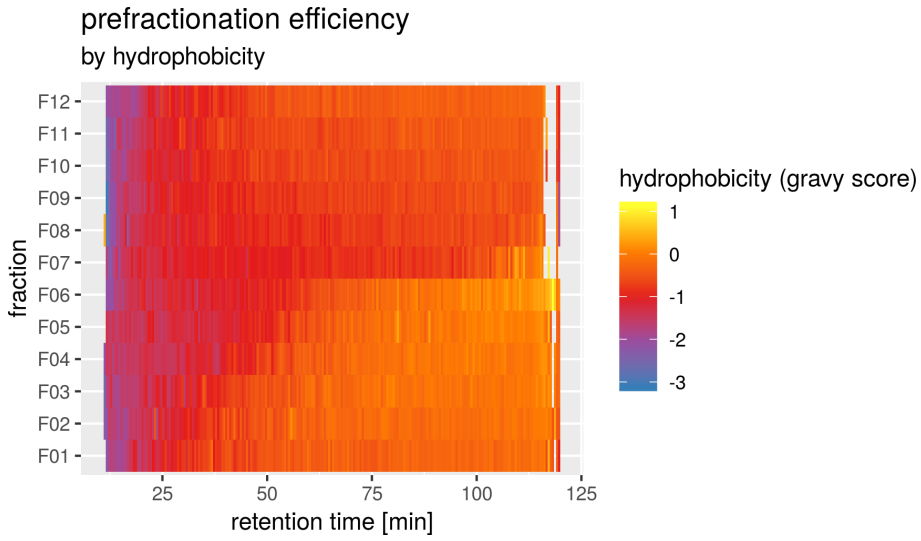
the pH changes the total charge of the peptide and thereby the interaction with the C18 column. This can be visualized by computing the pI for each peptide and summarizing the average pI per retention time for each fraction.

```
peptide_data$pI <- Peptides::pI(peptide_data$sequence)
ggplot(data = subset(peptide_data,in_quantification_of_protein == 1),
       aes(`retention time`,fraction)) +
  stat_summary_2d(aes(z = pI),binwidth = 0.5) +
  scale_fill_gradientn(colours = c("#377eb8","#984ea3","#e41a1c",
                                   "#ff7f00","#ffff33"),
                       name = "isoelectric point pI") +
  xlab("retention time [min]") +
  ggtitle("prefractionation efficiency",
          subtitle = "by isoelectic point")
```



A C18 column also seperates peptides based on their hydrophobicity. To visualize this, one can calculate a hydrophobicity score for each peptide and show again the average for a specific retention time.

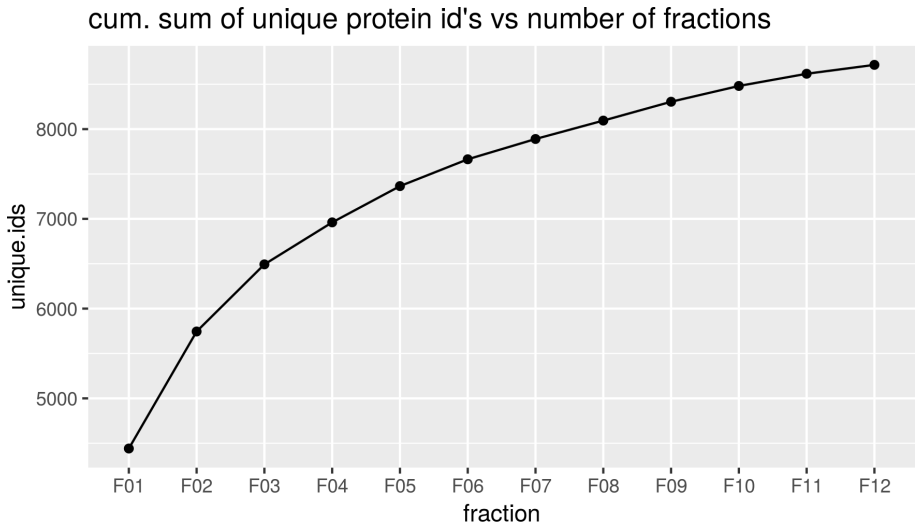
```
peptide_data$gravy <- Peptides::hydrophobicity(peptide_data$sequence)
ggplot(data = subset(peptide_data, in_quantification_of_protein == 1),
       aes(`retention time`,fraction)) +
  stat_summary_2d(aes(z = gravy), binwidth = 0.5) +
  scale_fill_gradientn(colours = c("#377eb8","#984ea3","#e41a1c",
                                   "#ff7f00","#ffff33"),
                       name = "hydrophobicity (gravy score)") +
  xlab("retention time [min]") +
  ggtitle("prefractionation efficiency",
          subtitle = "by hydrophobicity")
```



The cumulative sum of unique protein ids over the different fractions is helpful to judge the the gain of depth for the various fractions.

This gain will be decreasing with an increasing number of fractions.

```
id_data <- NULL

pfractions <- levels(peptide_data$fraction)
for (i in seq_along(pfractions)) {
  sub <- subset(peptide_data,fraction %in% pfractions[1:i]&
                  in_quantification_of_protein==1)
  if (nrow(sub) > 0) {
    id_sub <- data.frame(fraction = pfractions[i],
                         unique.ids = length(unique(sub$protein_id)))
    id_data <- rbind(id_data,id_sub)
    rm(id_sub)
  }
}

id_data$fraction <- factor(id_data$fraction,
                           ordered = TRUE, levels = pfractions)
id_data$file <- "file"
ggplot(data = id_data, aes(fraction,unique.ids)) +
  geom_line(aes(group = file)) +
  geom_point() +
  ggtitle(label = "cum. sum of unique protein id's vs number of fractions")
```

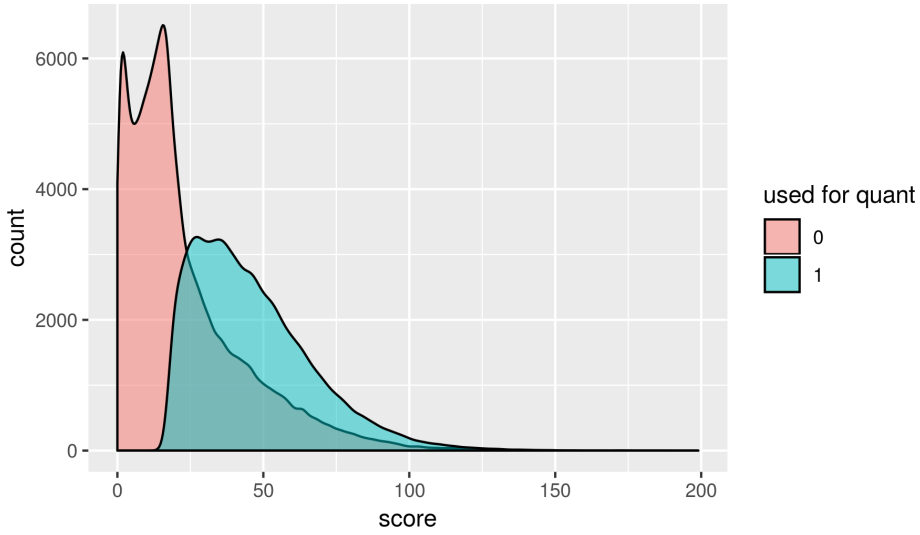


6.7 Peptide identification

6.7.1 Mascot score

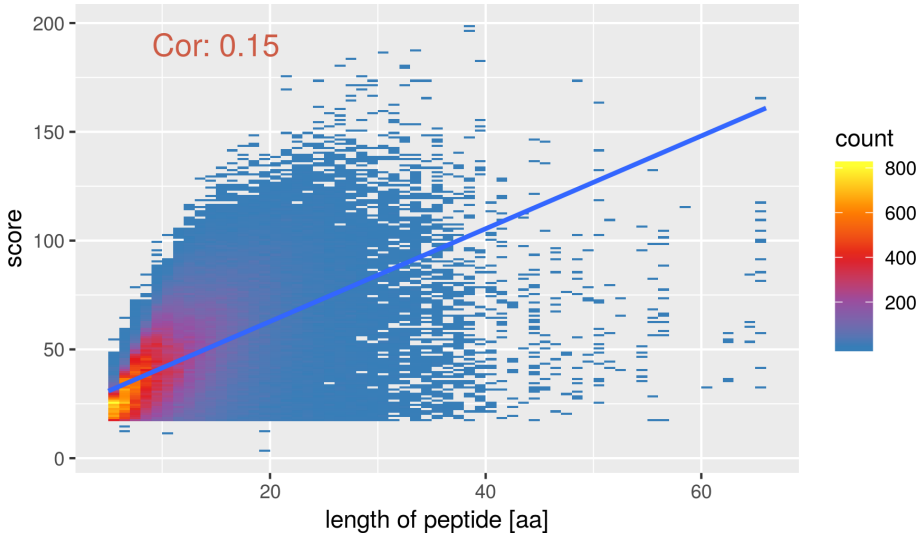
Each identified ms2 spectrum is annotated to a specific peptide sequence. The quality of the annotation is quantified by a mascot score, a probability- based scoring system. The better the quality of the identification, the higher the score. Peptides which were used for the protein quantification have a higher mascot score than the ones not used. Their scores have a lower bound of 20, our quality cut–off.

```
ggplot(data = peptide_data,aes(score,
       fill = factor(in_quantification_of_protein))) +
  geom_density(aes(y = ..count..), alpha = 0.5) +
  scale_fill_discrete(guide = guide_legend(title = "used for quant"))
```



One determining factor to the mascot score is the peptide length. Longer peptides tend to have a better identification quality since they result in more ion fragments.

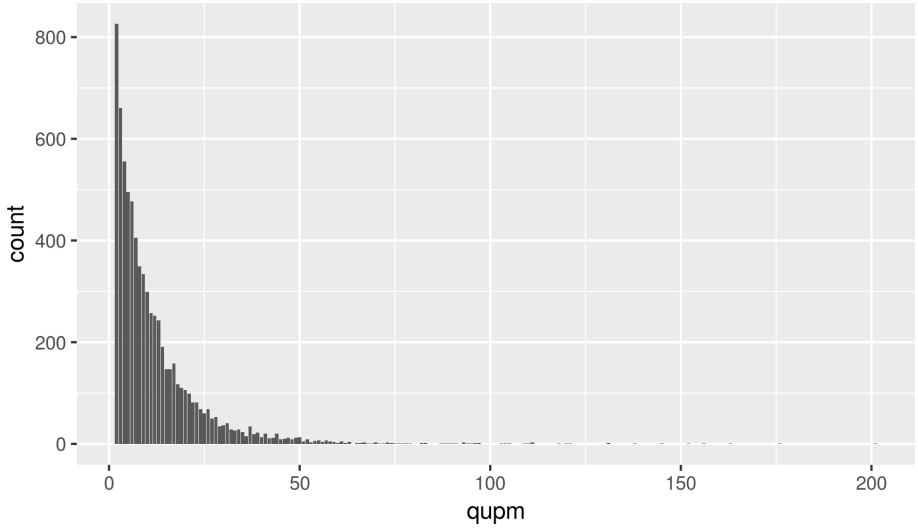
```
scp_mascot <- ggplot(data = subset(peptide_data,
                   in_quantification_of_protein == 1),
       aes(nchar(sequence),score)) +
  stat_bin2d(binwidth = 1) +
  geom_smooth(method = "lm") +
  scale_fill_gradientn(colours =
                         c("#377eb8","#984ea3",
                           "#e41a1c","#ff7f00","#ffff33")) +
  xlab("length of peptide [aa]")

scp_mascot +
  annotate("text", label = paste0("Cor: ",
                                  round(cor(layer_data(scp_mascot)$x,
                                            layer_data(scp_mascot)$y,
                                            method = "spearman"), 2)),
           x = 15, y = 190, size = 5, colour = "coral3")
```



6.7.2 Number of unique peptides per protein

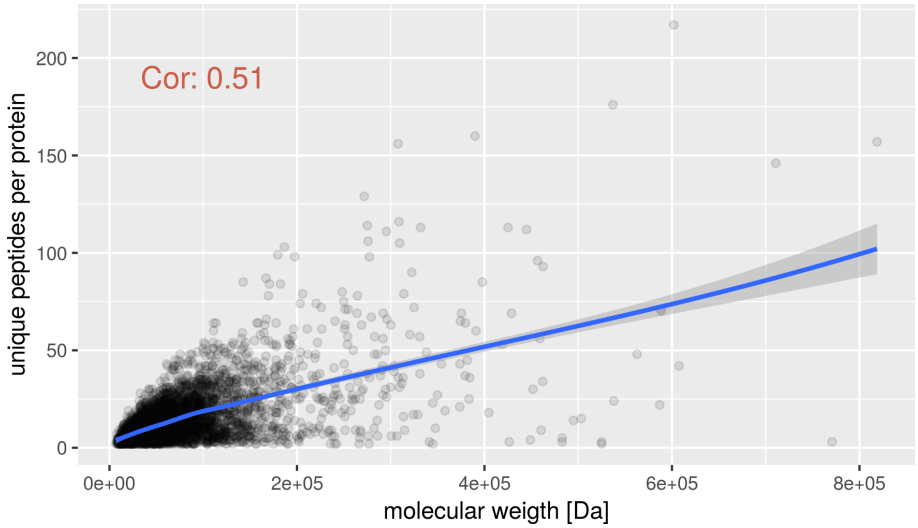
There is a different number of unique peptides identified for each protein.

```
ggplot(data = subset(protein_data,qupm >= 2), aes(qupm)) +
  geom_bar()
```



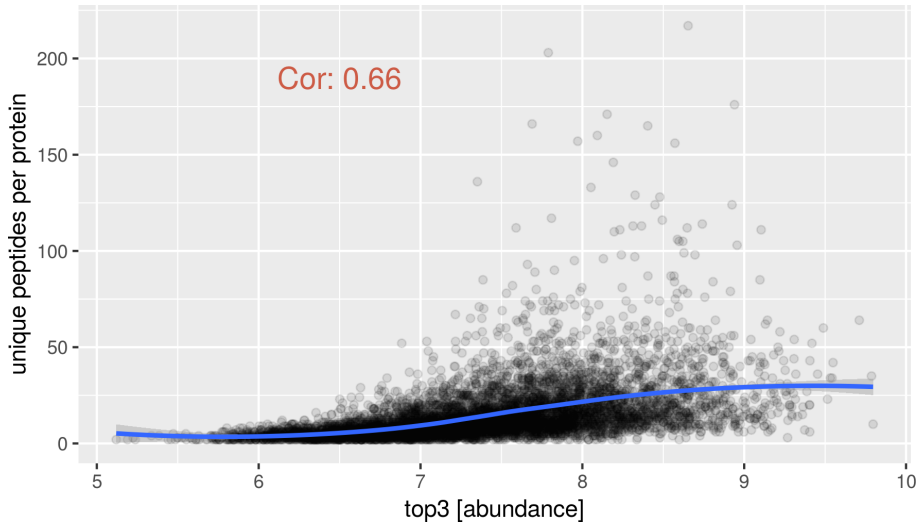
This depends on two things: The length of the protein and its molecular weight:

```
scp_mw <- ggplot(data = subset(protein_data, qupm >= 2),
       aes(as.numeric(as.character(mw)), upm)) +
  geom_point( alpha = I(0.1)) +
  geom_smooth(method = "loess") +
  scale_fill_gradientn(colours = c("#377eb8","#984ea3",
                                   "#e41a1c","#ff7f00","#ffff33")) +
  xlab("molecular weigth [Da]") +
  ylab("unique peptides per protein")
scp_mw +
  annotate("text", label = paste0("Cor: ",
                                  round(cor(layer_data(scp_mw)$x,
                                            layer_data(scp_mw)$y,
                                            use = "pairwise.complete.obs",
                                            method = "spearman"), 2)),
           x = 1e5, y = 190, size = 5, colour = "coral3")
```



As well as the abundance of the protein in the sample.

```
scp_top3 <- ggplot(data = subset(protein_data,qupm >= 2),aes(top3,upm)) +
  geom_point(alpha = I(0.1) ) +
  geom_smooth(method = "loess") +
  scale_fill_gradientn(colours = c("#377eb8","#984ea3","#e41a1c",
                                   "#ff7f00","#ffff33")) +
  xlab("top3 [abundance]") +
  ylab("unique peptides per protein")
scp_top3 +
  annotate("text", label = paste0("Cor: ",
                                  round(cor(layer_data(scp_top3)$x,
                                            layer_data(scp_top3)$y,
                                            use = "pairwise.complete.obs",
                                            method = "spearman"), 2)),
           x = 6.5, y = 190, size = 5, colour = "coral3")
```



7 Statistical analysis

7.1 Filter protein data

Identified proteins which are known to be a contaminant as well as proteins for which the TMT quantification is based on less than two unique peptides (qupm) are removed.

```
protein_data <- protein_data %>%
  filter(!grepl("###",gene_name), qupm >= 2)
```

7.2 Extract intensity data as matrix

We gather the protein intensity data in a data matrix. The columns interesting to us start with `signal sum` and then the respective TMT label. We discard the label `131L` as this does not correspond to a sample in our data.

We remove the `signal_sum_` part from the sample labels in order to match the `tmt.label` column in the experimental Design table.

```
prot_matrix <- dplyr::select(protein_data,
                             signal_sum_126:signal_sum_130H) %>%
               as.matrix()

colnames(prot_matrix) <- str_remove(colnames(prot_matrix), "signal_sum_")
rownames(prot_matrix) <- protein_data$protein_id


stopifnot(colnames(prot_matrix)  == expDesign$tmt.label[-length(expDesign$tmt.label)])



to_exclude <- apply(prot_matrix, 1, function(x){any(!is.finite(x))})
table(to_exclude)
   to_exclude
   FALSE
    7212

prot_matrix <- prot_matrix[!to_exclude, ]

head(prot_matrix)
                                     126     127L     127H     128L     128H
   P23919                       3.51e+08 3.57e+08 3.47e+08 3.08e+08 3.88e+08
   P40425                       1.06e+07 8.46e+06 8.68e+06 9.06e+06 8.30e+06
   P40429                       7.40e+08 7.31e+08 7.28e+08 6.80e+08 6.48e+08
   F5GZF0|O14519|O14519-2       2.35e+07 1.97e+07 2.31e+07 2.09e+07 2.10e+07
   A0A087WT59|A0A087WV45|P02766 7.33e+07 8.89e+07 3.76e+07 4.01e+07 8.90e+07
   P35251|P35251-2              3.63e+08 3.87e+08 3.96e+08 3.48e+08 3.50e+08
                                    129L     129H     130L     130H
   P23919                       2.76e+08 3.34e+08 3.99e+08 3.07e+08
   P40425                       7.91e+06 9.50e+06 9.03e+06 9.22e+06
   P40429                       6.36e+08 7.65e+08 7.40e+08 6.51e+08
   F5GZF0|O14519|O14519-2       1.82e+07 2.15e+07 2.13e+07 2.17e+07
   A0A087WT59|A0A087WV45|P02766 2.34e+07 4.37e+07 8.52e+07 3.17e+07
   P35251|P35251-2              3.43e+08 3.76e+08 3.71e+08 3.50e+08
```

7.3 Prepare a feature annotation

We now create another matrix holding the protein annotation.

```
feature_anno <- as.matrix(dplyr::select(protein_data, protein_id, gene_name,
                                top3, qupm, description))

rownames(feature_anno) <- protein_data$protein_id

feature_anno <- as.data.frame(feature_anno)[!to_exclude, ]
```

7.4 Create an expressionSet

We now join the replicates and add annotation data. We then turn the data into an **expressionSet**.

7.4.1 About Bioconductor ExpressionSets

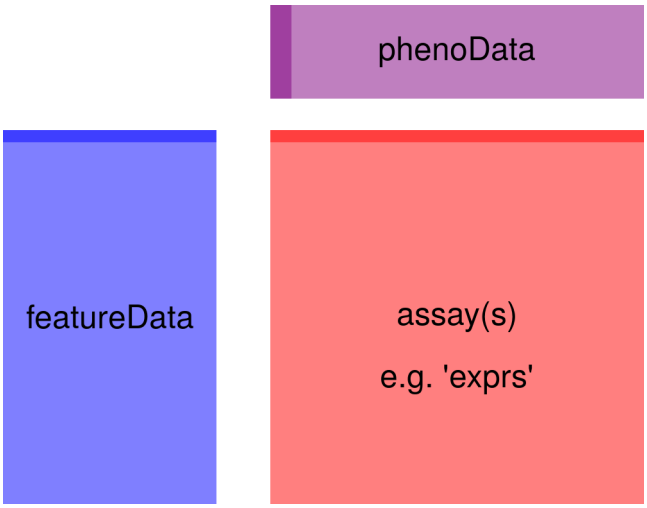
Genomic and proteomic data can be very complex, usually consisting of a number of different bits and pieces, e.g. information on the experimental samples, annotation of genomic features measured as well as the experimental data itself In Bioconductor the approach is taken that these pieces should be stored in a single structure to easily manage the data.

The package *Biobase* contains standardized data structures to represent genomic data. The `ExpressionSet` class is designed to combine several different sources of information into a single convenient structure. An ExpressionSet can be manipulated (e.g., subsetted, copied), and is the input to or output of many Bioconductor functions.

The data in an ExpressionSet consist of

- **assayData**: Expression data.
- **metaData**: A description of the samples in the experiment (phenoData), metadata about the features/proteins (featureData), and further annotations for the proteins, for example gene annotations from biomedical databases (annotation).
- **experimentData**: A flexible structure to describe the experiment.

The ExpressionSet class coordinates all of these data, so that one does not have to worry about the details. However, some constrains have to be met. In particular, the rownames of the `phenoData` have to match the column names of the assay data (as they represent the sample identifiers), while the row names of the expression data have to match the row names of the `featureData` (as they represent the feature identifiers). This is illustrated in the figure.



7.4.2 Creating the ExpressionSet

You can use the functions `pData` and `fData` to extract the sample and feature annotation respectively from an `ExpressionSet`. The function `exprs` will return the expression data itself as a matrix.

We can now combine the data into an `ExpressionSet` object.

```
eset_raw <- ExpressionSet(prot_matrix,
                  phenoData = AnnotatedDataFrame(expDesign[-nrow(expDesign),]),
                  featureData = AnnotatedDataFrame(feature_anno))

validObject(eset_raw)
   [1] TRUE
pData(eset_raw)
        tmt.label  rep condition
   126        126 rep1        wt
   127L      127L rep1 single_ko
   127H      127H rep1 double_ko
   128L      128L rep2        wt
   128H      128H rep2 single_ko
   129L      129L rep2 double_ko
   129H      129H rep3        wt
   130L      130L rep3 single_ko
   130H      130H rep3 double_ko

# prots with very low expression / zero expression
strange_prots <- apply(exprs(eset_raw), 1, function(x){min(x) < 1})
eset_raw <- eset_raw[!strange_prots, ]
save(eset_raw, file = "eset_raw.RData")
```

7.5 Quality control of the raw data

7.6 Array quality metrics report of the raw data

The package *arrayQualityMetrics* produces an html report, containing lots of quality control plots together with a description of their aims. We check our raw data using this reporting tool.

```
try(arrayQualityMetrics(expressionset = eset_raw,
      outdir = "Report_for_eset_raw",
    force = TRUE,
    do.logtransform = TRUE,
   intgroup = c("condition", "rep")
))
```

We can see a clear clustering by by replicate, rep3 of WT seems to be an outlier on the PCA plot.

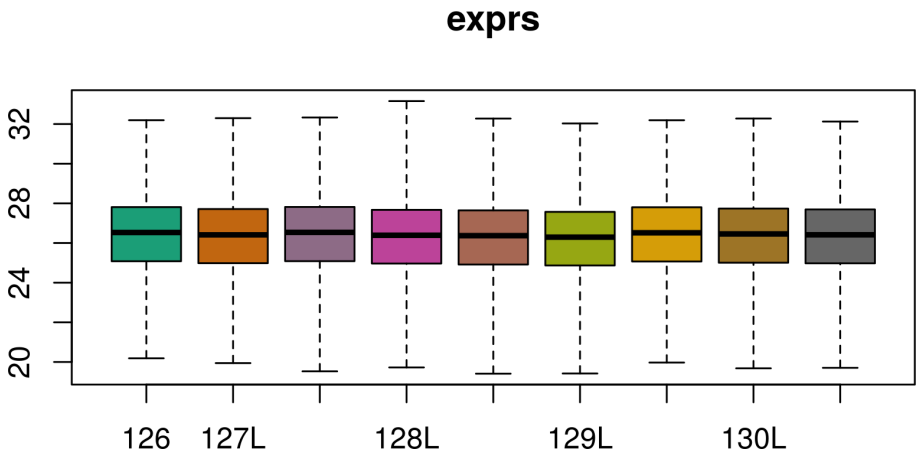
7.7 vsn normalization and feature annotation

vsn (variance stabilization transformation, *vsn*) is a Microarray normalization algorithm that performs background correction and normalization by robustly shifting and scaling intensity values within a sample before glog–transforming them. This is less "severe" than quantile normalization. vsn does perform a robust within–sample centering and scaling.

It is useful in general for intensity based data, e.g. also proteomics, see Hughes et. al., 2014 for an example usage in proteomics .

We first create boxplots for the raw data.

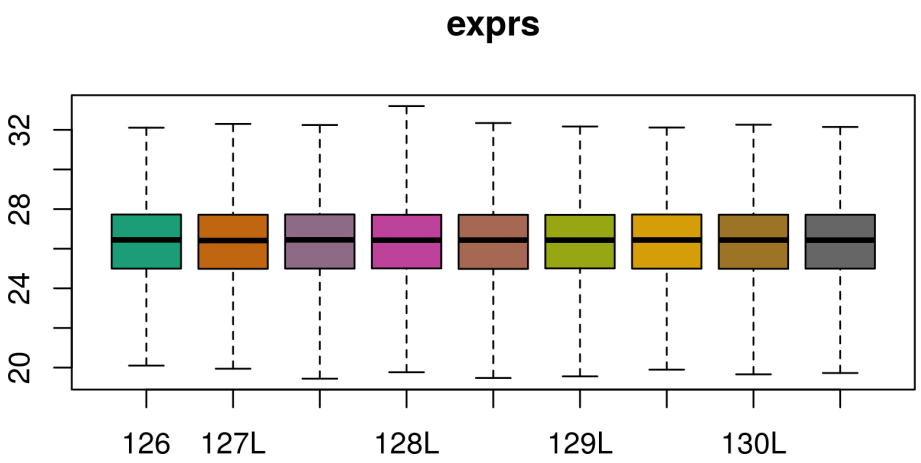
```
oligo::boxplot(eset_raw, transfo = log2, range = 0)
```



We can see that there only very minor differences between the different samples in the tags in the raw data are quite minimal in our example data.

Nonetheless, we use vsn to normalize the data an create another set of boxplots.

```
vsn_fit <- vsn2(eset_raw)
   vsn2: 7211 x 9 matrix (1 stratum).
   Please use 'meanSdPlot' to verify the fit.
eset_vsn <- predict(vsn_fit, newdata = eset_raw)

oligo::boxplot(eset_vsn, transfo = identity)
```




```
save(eset_vsn, file = "eset_vsn.RData")
```

As expected both boxplots are quite similar to each other.

7.7.1 check array quality metrics again

We now check the normalized data again.

```
try(arrayQualityMetrics(expressionset = eset_vsn,
      outdir = "Report_for_eset_after_vsn",
    force = TRUE,
    do.logtransform = FALSE,
   intgroup = c("condition", "rep")
))
```

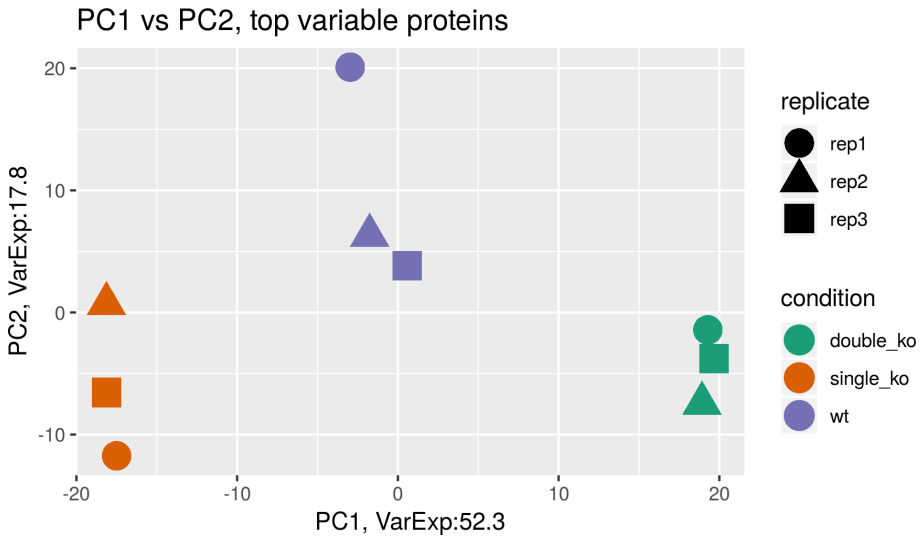
7.8 PCA plots of the normalized data

7.8.1 PCA using the most variable proteins

We now look at a PCA plot. Fist, we perform a simple variable selection and use only the top variable proteins for the computation of the principal components.

```
ntop <- 500

Pvars <- rowVars(exprs(eset_vsn))
select <- order(Pvars, decreasing = TRUE)[seq_len(min(ntop,
```

```
        length(Pvars)))]
PCA <- prcomp(t(exprs(eset_vsn)[select, ]), scale = TRUE)
percentVar <- round(100*PCA$sdev^2/sum(PCA$sdev^2),1)


dataGG = data.frame(PC1 = PCA$x[,1], PC2 = PCA$x[,2],
                    PC3 = PCA$x[,3], PC4 = PCA$x[,4],
                    replicate = pData(eset_vsn)$rep,
                    condition = pData(eset_vsn)$condition)

(qplot(PC1, PC2, data = dataGG, color =  condition,
       main = "PC1 vs PC2, top variable proteins", size = I(6),
       shape = replicate)
 + labs(x = paste0("PC1, VarExp:", round(percentVar[1],4)),
        y = paste0("PC2, VarExp:", round(percentVar[2],4)))
 + scale_colour_brewer(type = "qual", palette = 2)
 )
```



The PCA plot shows a very separation between the experimental conditions.

7.9 Limma analysis

We now perform a differential analysis using *limma*. We first create a design matrix that then allows us to compare between the differnt conditions. Here, the intercept represent the WT baseline, and the other coefficients represent the fold changes. See the limma user guide, chapter 9 for further details on typical designs.

http://bioconductor.org/packages/release/bioc/vignettes/limma/inst/doc/usersguide.pdf

```
condition <- as_factor(pData(eset_vsn)$condition )
des <- model.matrix(~ condition)
colnames(des) <- str_remove(colnames(des), "condition")
des
```

```
  (Intercept) single_ko double_ko
1           1         0         0
2           1         1         0
3           1         0         1
4           1         0         0
5           1         1         0
6           1         0         1
7           1         0         0
8           1         1         0
9           1         0         1
attr(,"assign")
[1] 0 1 1
attr(,"contrasts")
attr(,"contrasts")$condition
[1] "contr.treatment"
```

7.9.1 Comparison of WT vs single KO

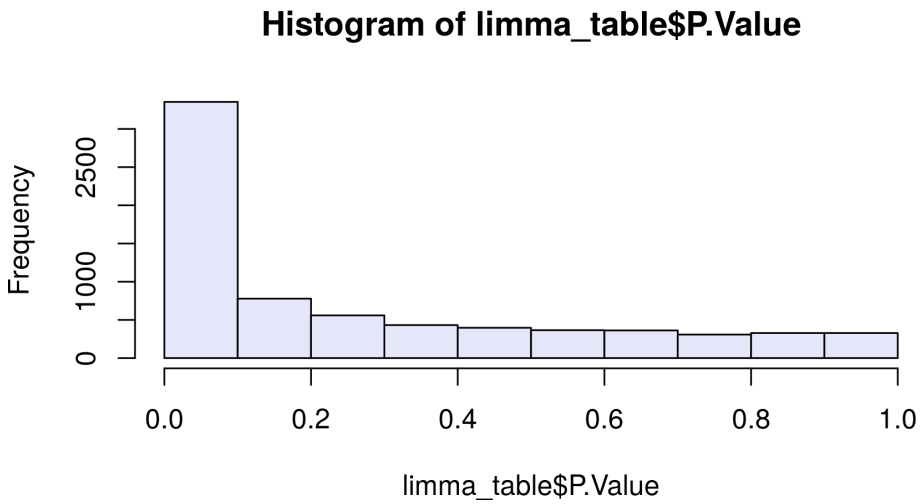
As we have a coefficient, `"single_ko"` that encodes the the fold change between WT the single knockout.

The result we get is a big table, with all significant proteins and a short description of what they are doing.

```
lm_fit <- eBayes(lmFit(eset_vsn, design = des))

limma_table <- topTable(lm_fit, sort.by = "t",  coef = "single_ko",
                        number = Inf)

as.tibble(limma_table)
   # A tibble: 7,211 x 11
      protein_id gene_name top3  qupm  description  logFC AveExpr     t  P.Value
    * <fct>      <fct>     <fct> <fct> <fct>        <dbl>   <dbl> <dbl>    <dbl>
    1 O95470     SGPL1     6.71  "  2" Sphingosin~ -2.25     23.9 -45.9 6.59e-12
    2 Q9HBL8     NMRAL1    7.01  "  7" NmrA-like ~ -1.48     25.7 -31.2 2.04e-10
    3 P47712     PLA2G4A   7.25  "  5" Cytosolic ~ -2.00     25.5 -28.3 4.82e-10
    4 P10620     MGST1     6.98  "  2" Microsomal~ -1.53     25.1 -24.3 1.85e- 9
    5 C9JYQ9|H0~ RPL22L1   7.88  "  4" 60S riboso~ -2.34     26.1 -23.0 2.94e- 9
    6 Q9NZU0     FLRT3     6.67  "  9" Leucine-ri~ -0.627    25.4 -21.4 5.72e- 9
    7 F2Z2X0|O0~ NME4      7.18  "  5" Nucleoside~ -0.764    25.2 -20.8 7.13e- 9
    8 P55809     OXCT1     7.79  " 12" Succinyl-C~ -0.874    27.6 -20.4 8.47e- 9
    9 P00918     CA2       8.10  " 11" Carbonic a~ -2.04     27.5 -18.9 1.69e- 8
   10 Q9ULP9|Q9~ TBC1D24   6.83  " 10" TBC1 domai~ -0.671    26.0 -18.0 2.57e- 8
   # ... with 7,201 more rows, and 2 more variables: adj.P.Val <dbl>, B <dbl>


table(limma_table$adj.P.Val < 0.0001)

   FALSE  TRUE
    7174    37
```

Here, we use a very stringent FDR cutoff of 0.0001 (represented by the "adjusted" p–value). The FDR is proportion of false positives in our hit list, this means we have 0.0001 * 37 = 0.004, so almost zero false positives in this list.

```
hist(limma_table$P.Value, col = "lavender")
```



Another very important diagnostic is the p–value histogram: if everything worked well, we should have a uniform background of p–values (non–differentially abundant proteins) and a peak near zero representing the differentially abundant proteins.

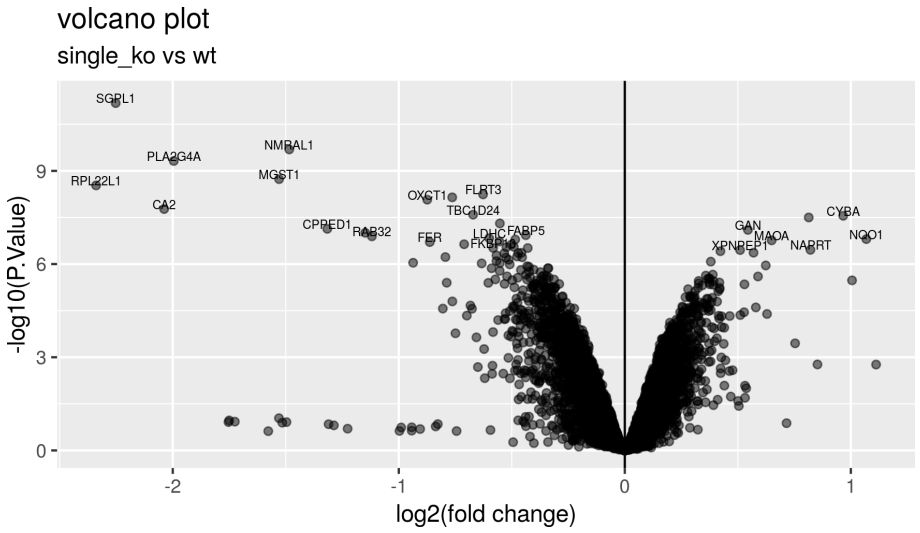
Unsual p–value histogram can point to batch effects and variance estimation issues.

7.9.2 Volcano plot

We now visualize the the results of the differential expression analysis in a Volcano plot, where we plot the log2 fold change vs the -log10 p–value. This allows us to assess statistical significance and effect size simultaneously.

The p–values should related to the fold changes, but not directly correlated, as the p–values are (apart from the fold change) also based on the protein variance and sample size.

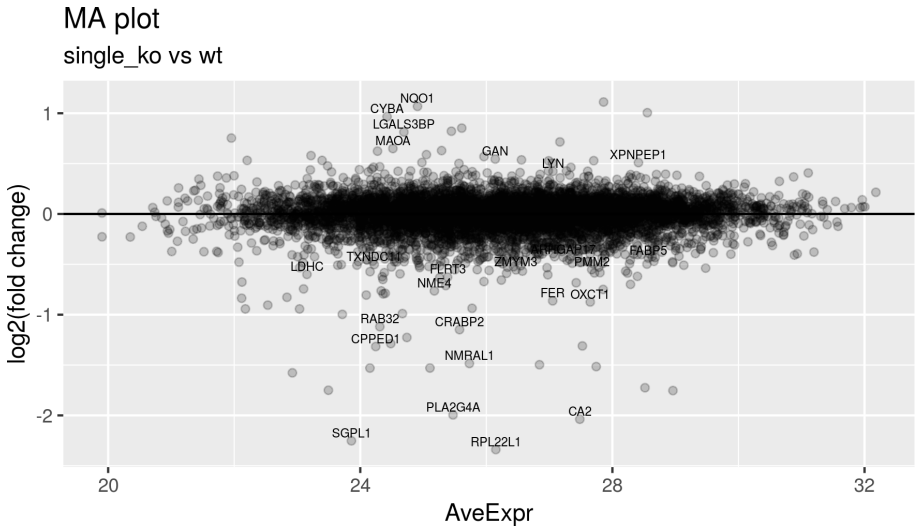
```
ggplot(data = limma_table,aes(logFC,-log10(P.Value))) +
  geom_point(alpha = I(0.5)) +
  geom_vline(xintercept = 0) +
  xlab("log2(fold change)") +
  geom_text(data = subset(limma_table, adj.P.Val <= 0.0001),
            aes(label = gene_name),
            size = 2,nudge_y = 0.15,check_overlap = TRUE) +
  ggtitle("volcano plot", subtitle = "single_ko vs wt")
```



7.9.3 MA-plot

We can also visualize the results in an MA plot. It plots the average protein abundance (log2 scale) vs. the log2 fold change. This allows us to see whether our differences between the conditions depend on the protein abundance.

```
ggplot(data = limma_table,aes(AveExpr, logFC)) +
  geom_point(alpha = I(0.2)) +
  geom_hline(yintercept = 0) +
  ylab("log2(fold change)") +
  geom_text(data = subset(limma_table,adj.P.Val <= 0.0001),
            aes(label = gene_name),
            size = 2, nudge_y = 0.08,check_overlap = TRUE) +
  ggtitle("MA plot",subtitle = "single_ko vs wt")
```

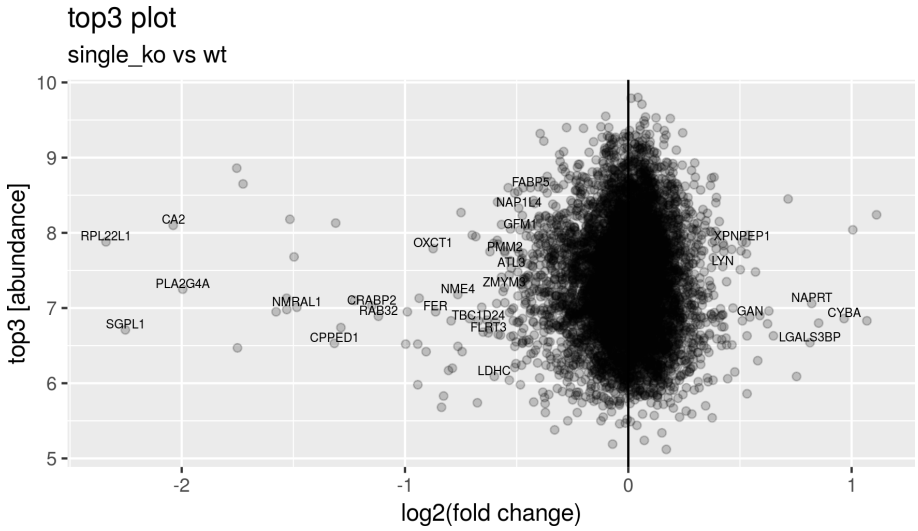


From the plot, we can see that our differentially abundant proteins are also abundant, and that there are no systematic biases between the experimental conditions.

7.9.4 Top3 vs logFC

Here, we plot the top3 value as reported by IsobarQuant against the fold change. The top3 value is a measure of abundance computed from the top3 MS1 peptide–peaks for a protein (mean of log10 intensities; the top3 peptides can be found in the peptides file). Again, we see that our hits are not in the low abundance range.

```
ggplot(data = limma_table,
       aes(logFC,as.numeric(as.character(top3)))) +
  geom_point(alpha = I(0.2)) +
  geom_vline(xintercept = 0) +
  xlab("log2(fold change)") +
  ylab("top3 [abundance]") +
  geom_text(data = subset(limma_table,adj.P.Val <= 0.0001),
            aes(label = gene_name),
            size = 2, nudge_y = 0.08,check_overlap = TRUE) +
  ggtitle("top3 plot",subtitle = "single_ko vs wt")
```



Note that both this and the previous plot would not possible to if we were working with ratios.

7.10 Testing relative to a fold-change threshold

Very often, you want to test relative to a specific fold-change threshold, e.g. you want to identify proteins which have a log2 fold-change of greater than 2.

This is often implemented via a "post-hoc" strategy: one thresholds the FDR and the fold change after testing using the null hypothesis that the fold change is zero.

7.10.1 The naive post-hoc strategy is biased

This naive strategy might lead to a loss of FDR control. We illustrate this in the following: We first simulate data with 10000 proteins, 1000 of which are differentially abundant with a FC that scatters around 2, and 20 samples, with 10 in each group.

We now perform a t-test for each protein.

```
n <- 20
p <- 10000

true_fcs <- numeric(10000)
idx_ab <- sample(x = p, 1000)
true_fcs[idx_ab] <- rnorm(length(idx_ab), mean = 2, sd = 0.5)

x <- matrix(rnorm(n*p), nrow = p, ncol = n)
fac <- factor(c(rep(0, 10), rep(1, 10)))
x[idx_ab, fac == 1] <- x[idx_ab, fac == 1]  + true_fcs[idx_ab]
rt <- rowttests(x, fac)
```

We now compute the proportion of false positives among all positives, the false discovery proportion (FDP), the expected value of which the Benjamini-Hochberg method controls at a pre-specified level ($E(FDP) = FDR$).

Indeed, this is controlled for our t-test:

```
FDR <- p.adjust(rt$p.value, method = "BH")

P <- which(FDR < 0.1)
TP <- intersect(idx_ab, P)
FP <- setdiff(P, idx_ab)

FDP <- length(FP) / length(P)

FDP
   [1] 0.0928
```

The actual level is 0.093 around the 10% we set. If we now do a post-hoc thresholding, we can again compute the FDP:

7.10.2 Explore the post–Hoc thresholding approach

```
fcs <- rt$dm

thresh_hits <- which(abs(fcs) > 2 & FDR < 0.1)

P_thresh <- thresh_hits

true_fcs_2 <- intersect(idx_ab,  which(abs(true_fcs) > 2))

FP_thresh <- setdiff(P_thresh, true_fcs_2)

FDP_thresh <- length(FP_thresh) / length(P_thresh)

FDP_thresh
   [1] 0.212
```

Of 481 proteins we identify this way, 0.212 are actually false positives, much higher than the target value of 10%.

Therefore, post-hoc thresholding does not work.

7.10.3 Using limma treat

limma implements a strategy to specifically test relative to a fold change threshold, for details see the paper by McCarthy and Smyth:

Testing significance relative to a fold-change threshold is a TREAT

Running it, we get an FDP lower than the threshold:

```
treat_res <- topTreat(treat(lmFit(x, design = model.matrix(~ fac)), lfc = 2),
                      coef = 2, number = Inf, sort.by = "none")

P_treat <- which(treat_res$adj.P.Val < 0.1)

FP_treat <- setdiff(P_treat, true_fcs_2)

FDP_treat <-  length(FP_treat) / length(P_treat)

FDP_treat
   [1] 0
```

However, here we only identify a handful of proteins, therefore, we might want to lower the threshold in practice.

8 Session Info

```
sessionInfo()
   R version 3.5.1 (2018-07-02)
   Platform: x86_64-pc-linux-gnu (64-bit)
   Running under: Ubuntu 16.04.5 LTS

   Matrix products: default
   BLAS: /usr/lib/openblas-base/libblas.so.3
   LAPACK: /usr/lib/libopenblasp-r0.2.18.so

   locale:
    [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
    [3] LC_TIME=de_DE.UTF-8        LC_COLLATE=en_US.UTF-8
    [5] LC_MONETARY=de_DE.UTF-8    LC_MESSAGES=en_US.UTF-8
    [7] LC_PAPER=de_DE.UTF-8       LC_NAME=C
    [9] LC_ADDRESS=C               LC_TELEPHONE=C
   [11] LC_MEASUREMENT=de_DE.UTF-8 LC_IDENTIFICATION=C

   attached base packages:
```

```
[1] stats4    parallel  stats     graphics  grDevices utils     datasets
[8] methods   base

other attached packages:
 [1] bindrcpp_0.2.2             Peptides_2.4
 [3] gridExtra_2.3              Hmisc_4.1-1
 [5] Formula_1.2-3              survival_2.42-6
 [7] forcats_0.3.0              readr_1.1.1
 [9] tibble_1.4.2               tidyverse_1.2.1
[11] purrr_0.2.5                fdrtool_1.2.15
[13] pheatmap_1.0.10            MSnbase_2.6.2
[15] ProtGenerics_1.12.0        BiocParallel_1.14.2
[17] mzR_2.14.0                 Rcpp_0.12.18
[19] vsn_3.48.1                 tidyr_0.8.1
[21] smoothmest_0.1-2           MASS_7.3-50
[23] openxlsx_4.1.0             limma_3.36.2
[25] genefilter_1.62.0          matrixStats_0.54.0
[27] stringr_1.3.1              arrayQualityMetrics_3.36.0
[29] RColorBrewer_1.1-2         gplots_3.0.1
[31] dplyr_0.7.6                plyr_1.8.4
[33] reshape2_1.4.3             ggplot2_3.0.0
[35] geneplotter_1.58.0         annotate_1.58.0
[37] XML_3.98-1.16              AnnotationDbi_1.42.1
[39] IRanges_2.14.10            S4Vectors_0.18.3
[41] lattice_0.20-35            Biobase_2.40.0
[43] BiocGenerics_0.26.0        knitr_1.20
[45] BiocStyle_2.8.2

loaded via a namespace (and not attached):
  [1] readxl_1.1.0                backports_1.1.2
  [3] lazyeval_0.2.1              oligo_1.44.0
  [5] splines_3.5.1               GenomeInfoDb_1.16.0
  [7] digest_0.6.15               foreach_1.4.4
  [9] BiocInstaller_1.30.0        htmltools_0.3.6
 [11] fansi_0.3.0                 gdata_2.18.0
 [13] magrittr_1.5                checkmate_1.8.5
 [15] memoise_1.1.0               affyPLM_1.56.0
 [17] cluster_2.0.7-1             doParallel_1.0.11
 [19] gcrma_2.52.0                Biostrings_2.48.0
 [21] modelr_0.1.2                beadarray_2.30.0
 [23] colorspace_1.3-2            rvest_0.3.2
 [25] blob_1.1.1                  haven_1.1.2
 [27] xfun_0.3                    crayon_1.3.4
 [29] RCurl_1.95-4.11             jsonlite_1.5
 [31] bindr_0.1.1                 impute_1.54.0
 [33] iterators_1.0.10            glue_1.3.0
 [35] gtable_0.2.0                zlibbioc_1.26.0
 [37] XVector_0.20.0              DelayedArray_0.6.5
 [39] BeadDataPackR_1.32.0        scales_1.0.0
 [41] setRNG_2013.9-1             DBI_1.0.0
 [43] xtable_1.8-2                htmlTable_1.12
```

```
 [45] foreign_0.8-71              bit_1.1-14
 [47] preprocessCore_1.42.0       httr_1.3.1
 [49] htmlwidgets_1.2             acepack_1.4.1
 [51] ff_2.2-14                   pkgconfig_2.0.2
 [53] nnet_7.3-12                 utf8_1.1.4
 [55] labeling_0.3                tidyselect_0.2.4
 [57] rlang_0.2.2                 cellranger_1.1.0
 [59] munsell_0.5.0               tools_3.5.1
 [61] cli_1.0.0                   RSQLite_2.1.1
 [63] broom_0.5.0                 evaluate_0.11
 [65] mzID_1.18.0                 yaml_2.2.0
 [67] bit64_0.9-7                 oligoClasses_1.42.0
 [69] zip_1.0.0                   caTools_1.17.1.1
 [71] nlme_3.1-137                xml2_1.2.0
 [73] compiler_3.5.1              rstudioapi_0.7
 [75] affyio_1.50.0               stringi_1.2.4
 [77] Matrix_1.2-14               pillar_1.3.0
 [79] MALDIquant_1.18             data.table_1.11.4
 [81] bitops_1.0-6                GenomicRanges_1.32.6
 [83] R6_2.2.2                    latticeExtra_0.6-28
 [85] pcaMethods_1.72.0           affy_1.58.0
 [87] hwriter_1.3.2               bookdown_0.7
 [89] affxparser_1.52.0           KernSmooth_2.23-15
 [91] gridSVG_1.6-0               codetools_0.2-15
 [93] gtools_3.8.1                assertthat_0.2.0
 [95] SummarizedExperiment_1.10.1 openssl_1.0.2
 [97] rprojroot_1.3-2             withr_2.1.2
 [99] GenomeInfoDbData_1.1.0      hms_0.4.2
[101] grid_3.5.1                  rpart_4.1-13
[103] base64_2.0                  rmarkdown_1.10
[105] illuminaio_0.22.0           Cairo_1.5-9
[107] lubridate_1.7.4             base64enc_0.1-3
```