Introduction to proteomics data analysis: maxLFQ summarization

Lieven Clement

statOmics, Ghent University (https://statomics.github.io)

Contents

This is part of the online course Proteomics Data Analysis (PDA)

1 Background

This case-study is a subset of the data of the 6th study of the Clinical Proteomic Technology Assessment for Cancer (CPTAC). In this experiment, the authors spiked the Sigma Universal Protein Standard mixture 1 (UPS1) containing 48 different human proteins in a protein background of 60 ng/μL Saccharomyces cerevisiae strain BY4741. Two different spike-in concentrations were used: 6A (0.25 fmol UPS1 proteins/μL) and 6B (0.74 fmol UPS1 proteins/μL) [5]. We limited ourselves to the data of LTQ-Orbitrap W at site 56. The data were searched with MaxQuant version 1.5.2.8, and detailed search settings were described in Goeminne et al. (2016) [1]. Three replicates are available for each concentration.

- NOTE THAT maxLFQ SUMMARISATION IS SUBOPTIMAL!
- THIS IS FOR DIDACTICAL PURPOSES ONLY.

2 Data

We first import the data from proteinGroups.txt file. This is the file containing maxLFQ summarized protein-level intensities. For a MaxQuant search [6], this proteinGroups.txt file can be found by default in the "path_to_raw_files/combined/txt/" folder from the MaxQuant output, with "path_to_raw_files" the

folder where the raw files were saved. In this vignette, we use a MaxQuant proteinRaws file which is a subset of the cptac study. To import the data we use the `QFeatures` package.

We generate the object proteinRawFile with the path to the proteinGroups.txt file. Using the `grepEcols` function, we find the columns that contain the LFQ expression data of the proteinRaws in the proteinGroups.txt file.

3 Data

We first import the data from proteinGroups.txt file. This is the file containing maxLFQ summarized protein-level intensities. For a MaxQuant search [6], this proteinGroups.txt file can be found by default in the "path_to_raw_files/combined/txt/" folder from the MaxQuant output, with "path_to_raw_files" the folder where the raw files were saved. In this vignette, we use a MaxQuant proteinRaws file which is a subset of the cptac study. To import the data we use the QFeatures package.

We generate the object proteinRawFile with the path to the proteinGroups.txt file. Using the grepEcols function, we find the columns that contain the LFQ expression data of the proteinRaws in the proteinGroups.txt file.

```
library(tidyverse)
library(limma)
library(QFeatures)
library(msqrob2)
library(plotly)
proteinsFile <- "https://raw.githubusercontent.com/statOmics/PDA22GTPB/data/quantification/cptacAvsB_la

ecols <- grep("LFQ\\.intensity\\.", names(read.delim(proteinsFile)))
```

Next, we read the data and store it in QFeatures object

```
pe <- readQFeatures(
  assayData = read.delim(proteinsFile),
  fnames = 1,
  quantCols = ecols,
  name = "proteinRaw"
)
```

```
## Checking arguments.

## Loading data as a 'SummarizedExperiment' object.

## Formatting sample annotations (colData).

## Formatting data as a 'QFeatures' object.
```

The QFeatures object pe currently contains a single assay, named proteinRaw.

We extract the column names from the proteinRaw assay and see that this contains information about the spike-in condition.

```
colnames(pe[["proteinRaw"]])
```

```
## [1] "LFQ.intensity.6A_7" "LFQ.intensity.6A_8" "LFQ.intensity.6A_9"
## [4] "LFQ.intensity.6B_7" "LFQ.intensity.6B_8" "LFQ.intensity.6B_9"
```

We rename the colnames by dropping the "LFQ.intensity." from the name.

```
(newNames <-  sub(
  pattern = "LFQ\\.intensity\\.",
  replacement = "",
```

```
  colnames(pe[["proteinRaw"]]))
 )
```

```
## [1] "6A_7" "6A_8" "6A_9" "6B_7" "6B_8" "6B_9"
```

```
pe <- renameColname(pe,
                    i = "proteinRaw",
                    newNames)
pe <- renamePrimary(pe, newNames)
colnames(pe[["proteinRaw"]])
```

```
## [1] "6A_7" "6A_8" "6A_9" "6B_7" "6B_8" "6B_9"
```

In the following code chunk, we add the spikein condition that we can read in the raw file name to the colData.

```
colData(pe)$condition <-
  colnames(pe[["proteinRaw"]]) %>%
  substr(start = 2, stop = 2) %>%
  as.factor
colData(pe)$condition
```

```
## [1] A A A B B B
## Levels: A B
```

We calculate how many non zero intensities we have per protein and this will be useful for filtering.

```
rowData(pe[["proteinRaw"]])$nNonZero <- rowSums(assay(pe[["proteinRaw"]]) > 0)
```

Proteins with zero intensities are missing and should be represent with a `NA` value rather than `0`.

```
pe <- zeroIsNA(pe, "proteinRaw") # convert 0 to NA
```

3.1 Data exploration

45% of all peptide intensities are missing and for some proteins we do not even measure a signal in any sample.

4 Preprocessing

This section preforms preprocessing for the peptide data. This include

- log transformation,
- filtering

4.1 Log transform the data

```
pe <- logTransform(pe, base = 2, i = "proteinRaw", name = "proteinLog")
```

4.2 Filtering

1. Remove reverse sequences (decoys) and contaminants

We now remove the contaminants and proteins that map to decoys.

```
pe <- filterFeatures(pe,~ Reverse != "+")
```

```
## 'Reverse' found in 2 out of 2 assay(s)
```

```
pe <- filterFeatures(pe,~ Potential.contaminant != "+")
```

```
## 'Potential.contaminant' found in 2 out of 2 assay(s)
```

We keep 1537 peptides upon filtering.

4.3 Normalize the data using median centering

We normalize the data by substracting the sample median from every intensity for peptide p in a sample i:

$$y_{ip}^{\text{norm}} = y_{ip} - \hat{\mu}_i$$

with $\hat{\mu}_i$ the median intensity over all observed peptides in sample i.

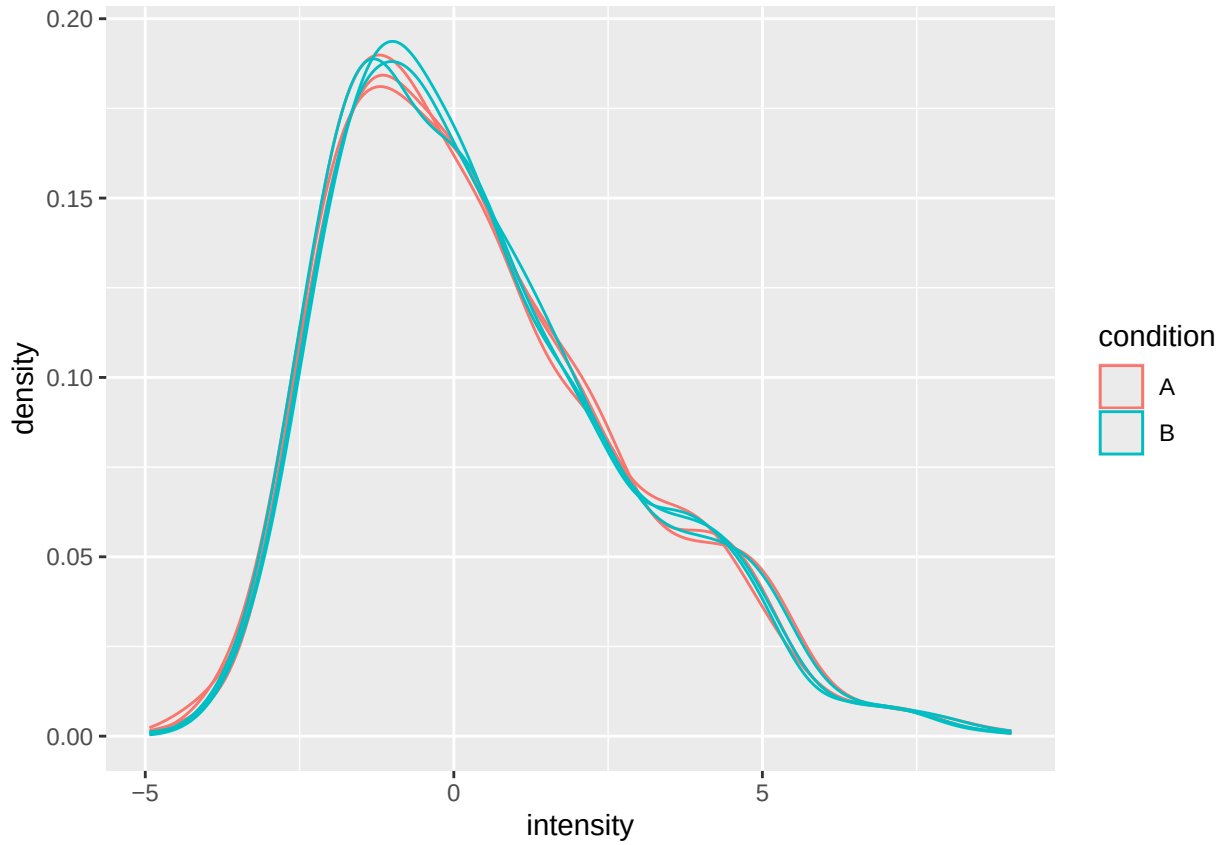
```
pe <- normalize(pe,
                i = "proteinLog",
                name = "protein",
                method = "center.median")
```

4.4 Explore normalized data

Upon the normalisation the density curves are nicely registered

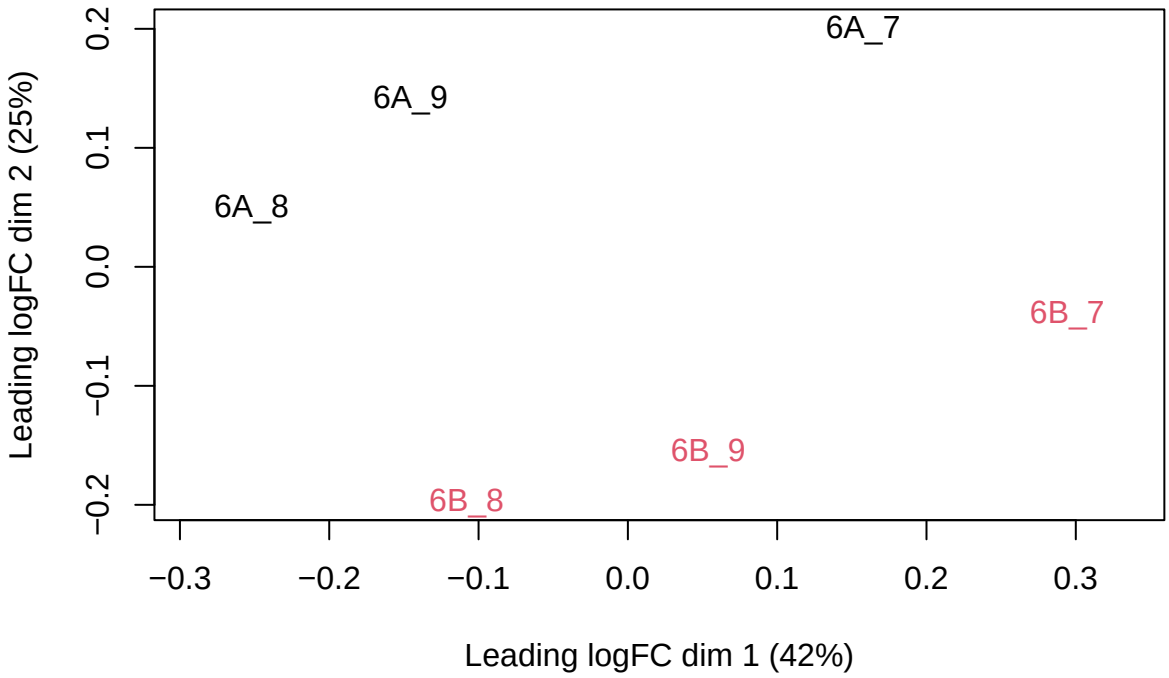
```
pe[["protein"]] %>%
  assay %>%
  as.data.frame() %>%
  gather(sample, intensity) %>%
  mutate(condition = colData(pe)[sample,"condition"]) %>%
  ggplot(aes(x = intensity,group = sample,color = condition)) +
    geom_density()
```

```
## Warning: Removed 4052 rows containing non-finite outside the scale range
## (`stat_density()`).
```



We can visualize our data using a Multi Dimensional Scaling plot, eg. as provided by the `limma` package.

```
pe[["protein"]] %>%
  assay %>%
  limma::plotMDS(col = as.numeric(colData(pe)$condition))
```



Note that the samples show a clear separation according to the spike-in condition in the second dimension of the MDS plot.

5 Data Analysis

5.1 Estimation

We model the protein level expression values using `msqrob`. By default `msqrob2` estimates the model parameters using robust regression.

We will model the data with a different group mean. The group is incoded in the variable `condition` of the colData. We can specify this model by using a formula with the factor condition as its predictor: `formula = ~condition`.

Note, that a formula always starts with a symbol '~'.

```
pe <- msqrob(object = pe, i = "protein", formula = ~condition)
```

5.2 Inference

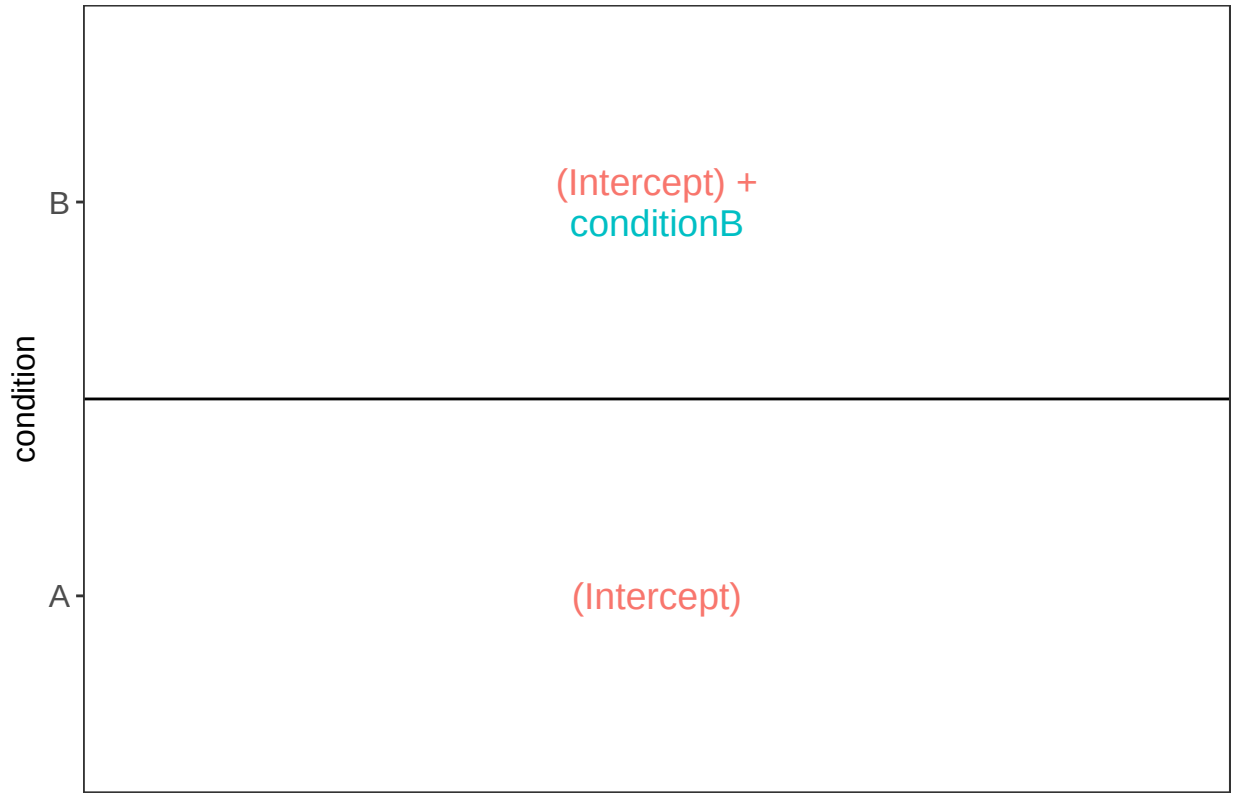
First, we extract the parameter names of the model by looking at the first model. The models are stored in the row data of the assay under the default name msqrobModels.

```
getCoef(rowData(pe[["protein"]])$msqrobModels[[1]])
```

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

We can also explore the design of the model that we specified using the the package `ExploreModelMatrix`

```
library(ExploreModelMatrix)
VisualizeDesign(colData(pe),~condition)$plotlist[[1]]
```



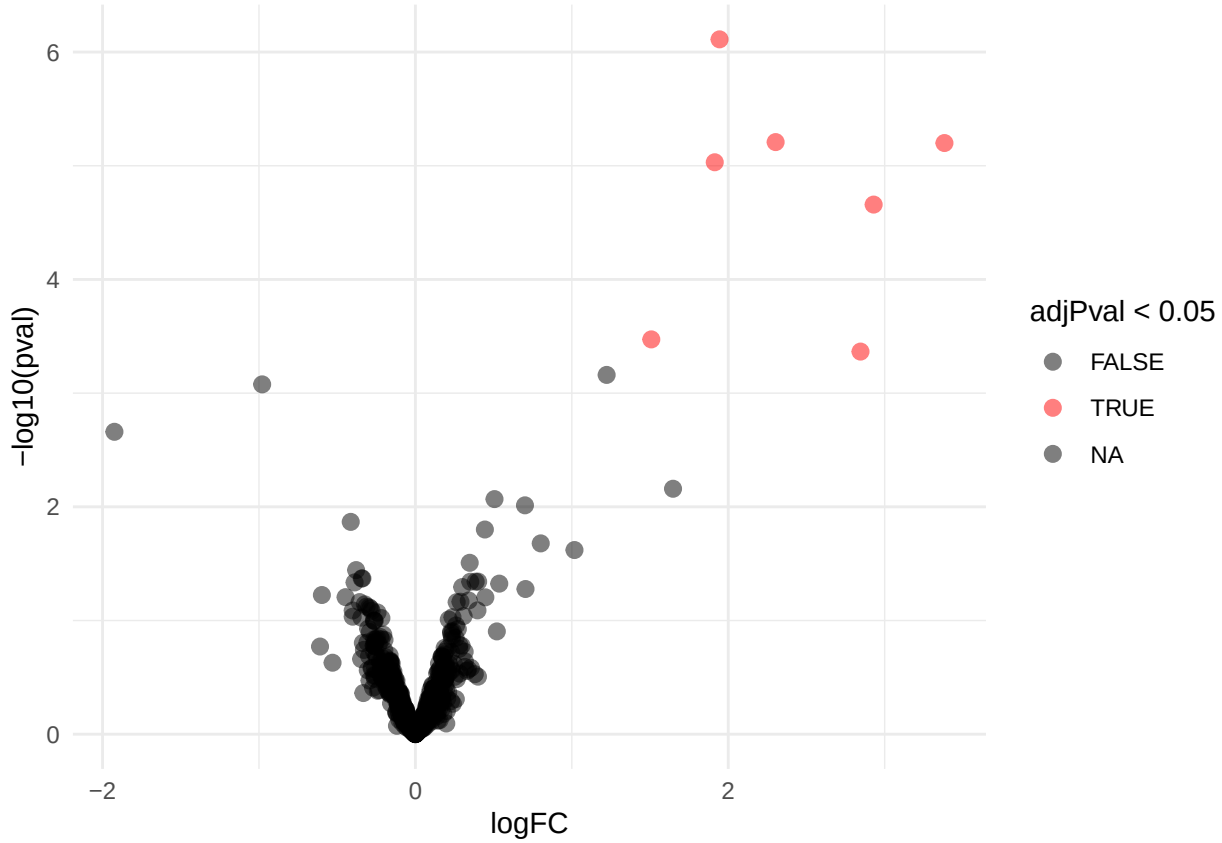
Spike-in condition `A` is the reference class. So the mean log2 expression for samples from condition A is '(Intercept). The mean log2 expression for samples from condition B is'(Intercept)+conditionB'. Hence, the average log2 fold change between condition b and condition a is modelled using the parameter 'conditionB'. Thus, we assess the contrast 'conditionB = 0' with our statistical test.

```
L <- makeContrast("conditionB=0", parameterNames = c("conditionB"))
pe <- hypothesisTest(object = pe, i = "protein", contrast = L)
```

5.3 Plots

5.3.1 Volcano-plot

```
volcano <- ggplot(rowData(pe[["protein"]])$conditionB,
                  aes(x = logFC, y = -log10(pval), color = adjPval < 0.05)) +
  geom_point(cex = 2.5) +
  scale_color_manual(values = alpha(c("black", "red"), 0.5)) + theme_minimal()
volcano
```

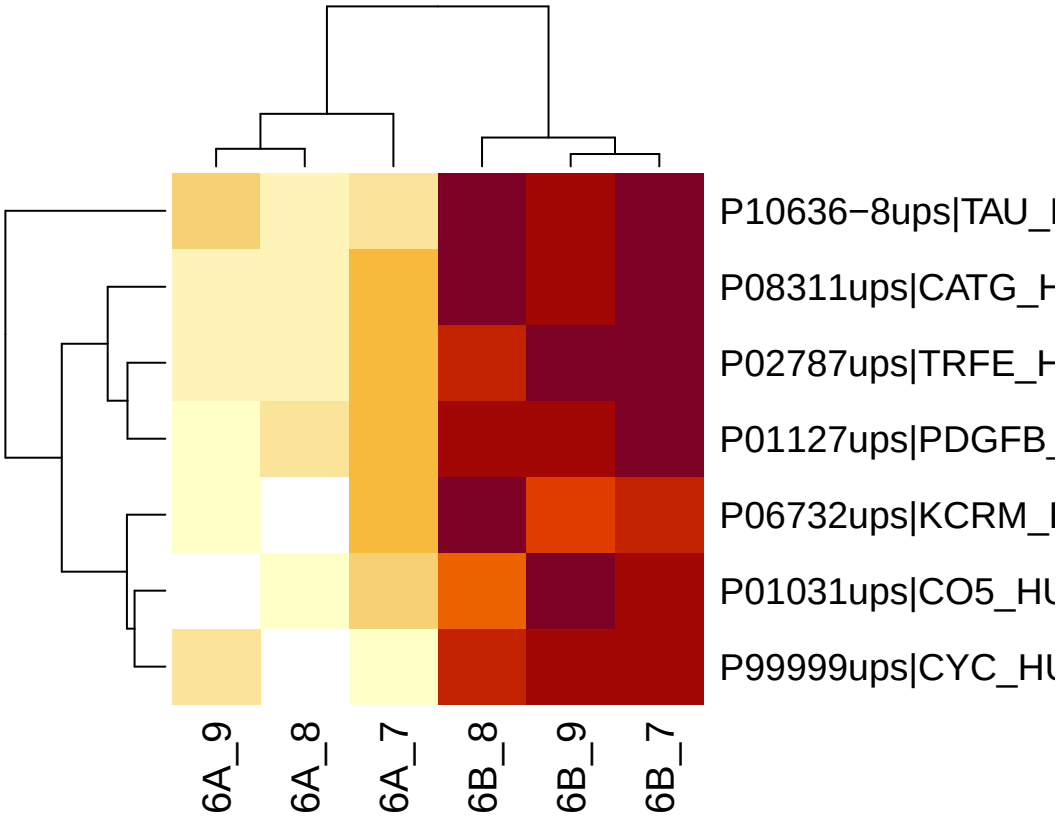


Note, that only 7 proteins are found to be differentially abundant.

5.3.2 Heatmap

We first select the names of the proteins that were declared signficant.

```
sigNames <- rowData(pe[["protein"]])$conditionB %>%
  rownames_to_column("protein") %>%
  filter(adjPval<0.05) %>%
  pull(protein)
heatmap(assay(pe[["protein"]])[sigNames, ])
```



The majority of the proteins are indeed UPS proteins. 1 yeast protein is returned. Note, that the yeast protein indeed shows evidence for differential abundance.

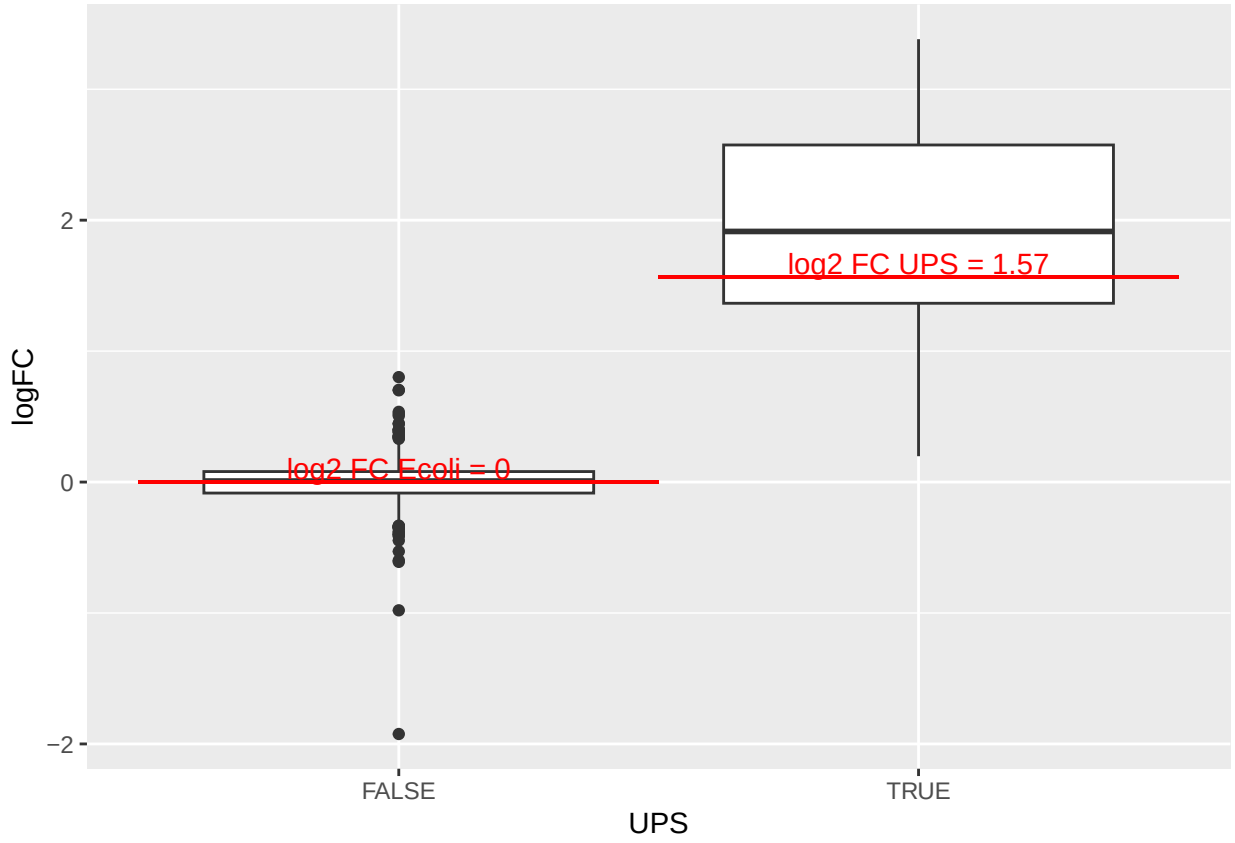
5.3.3 Boxplots

We make boxplot of the log2 FC and stratify according to the whether a protein is spiked or not.

```
rowData(pe[["protein"]])$conditionB %>%
  rownames_to_column(var = "protein") %>%
  ggplot(aes(x=grepl("UPS",protein),y=logFC)) +
  geom_boxplot() +
  xlab("UPS") +
  geom_segment(
    x = 1.5,
    xend = 2.5,
    y = log2(0.74/0.25),
    yend = log2(0.74/0.25),
    colour="red") +
  geom_segment(
    x = 0.5,
    xend = 1.5,
    y = 0,
    yend = 0,
    colour="red") +
  annotate(
    "text",
    x = c(1,2),
```

```
    y = c(0,log2(0.74/0.25))+.1,
    label = c(
      "log2 FC Ecoli = 0",
      paste0("log2 FC UPS = ",round(log2(0.74/0.25),2))
      ),
    colour = "red")
```

```
## Warning: Removed 746 rows containing non-finite outside the scale range
## (`stat_boxplot()`).
```



What do you observe?

6 Session Info

With respect to reproducibility, it is highly recommended to include a session info in your script so that readers of your output can see your particular setup of R.

```
sessionInfo()
```

```
## R version 4.4.0 RC (2024-04-16 r86468)
## Platform: aarch64-apple-darwin20
## Running under: macOS Big Sur 11.6
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack.dylib;  LAPACK v
##
```

```
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: Europe/Brussels
## tzcode source: internal
##
## attached base packages:
## [1] stats4    stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] ExploreModelMatrix_1.16.0   plotly_4.10.4
##  [3] msqrob2_1.12.0              QFeatures_1.14.2
##  [5] MultiAssayExperiment_1.30.3 SummarizedExperiment_1.34.0
##  [7] Biobase_2.64.0              GenomicRanges_1.56.1
##  [9] GenomeInfoDb_1.40.1         IRanges_2.38.1
## [11] S4Vectors_0.42.1            BiocGenerics_0.50.0
## [13] MatrixGenerics_1.16.0       matrixStats_1.4.1
## [15] limma_3.60.5                lubridate_1.9.3
## [17] forcats_1.0.0               stringr_1.5.1
## [19] dplyr_1.1.4                 purrr_1.0.2
## [21] readr_2.1.5                 tidyr_1.3.1
## [23] tibble_3.2.1                ggplot2_3.5.1
## [25] tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##  [1] rlang_1.1.4             magrittr_2.0.3          shinydashboard_0.7.2
##  [4] clue_0.3-65             compiler_4.4.0          vctrs_0.6.5
##  [7] reshape2_1.4.4          ProtGenerics_1.36.0     pkgconfig_2.0.3
## [10] crayon_1.5.3            fastmap_1.2.0           XVector_0.44.0
## [13] fontawesome_0.5.2       labeling_0.4.3          utf8_1.2.4
## [16] promises_1.3.0          rmarkdown_2.28          tzdb_0.4.0
## [19] UCSC.utils_1.0.0        nloptr_2.1.1            xfun_0.47
## [22] zlibbioc_1.50.0         cachem_1.1.0            jsonlite_1.8.9
## [25] later_1.3.2             highr_0.11              DelayedArray_0.30.1
## [28] BiocParallel_1.38.0     parallel_4.4.0          cluster_2.1.6
## [31] R6_2.5.1                bslib_0.8.0             stringi_1.8.4
## [34] boot_1.3-31             jquerylib_0.1.4         Rcpp_1.0.13
## [37] knitr_1.48              BiocBaseUtils_1.6.0     httpuv_1.6.15
## [40] Matrix_1.7-0            splines_4.4.0           igraph_2.0.3
## [43] timechange_0.3.0        tidyselect_1.2.1        rstudioapi_0.16.0
## [46] abind_1.4-8             yaml_2.3.10             codetools_0.2-20
## [49] lattice_0.22-6          plyr_1.8.9              shiny_1.9.1
## [52] withr_3.0.1             evaluate_1.0.0          pillar_1.9.0
## [55] DT_0.33                 shinyjs_2.1.0           generics_0.1.3
## [58] hms_1.1.3               munsell_0.5.1           scales_1.3.0
## [61] minqa_1.2.8             xtable_1.8-4            glue_1.8.0
## [64] lazyeval_0.2.2          tools_4.4.0             data.table_1.16.0
## [67] lme4_1.1-35.5           cowplot_1.1.3           grid_4.4.0
## [70] MsCoreUtils_1.16.1      colorspace_2.1-1        nlme_3.1-166
## [73] GenomeInfoDbData_1.2.12 cli_3.6.3               fansi_1.0.6
## [76] S4Arrays_1.4.1          viridisLite_0.4.2       AnnotationFilter_1.28.0
## [79] gtable_0.3.5            rintrojs_0.3.4          sass_0.4.9
## [82] digest_0.6.37           SparseArray_1.4.8       htmlwidgets_1.6.4
```

```
## [85] farver_2.1.2           htmltools_0.5.8.1       lifecycle_1.0.4
## [88] httr_1.4.7             mime_0.12               statmod_1.5.0
## [91] MASS_7.3-61
```