

Introduction to proteomics data analysis: robust summarization

Lieven Clement

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

Contents

1	Background	1
2	Data	2
2.1	Data exploration	3
3	Preprocessing	3
3.1	Log transform the data	3
3.2	Filtering	4
3.3	Normalize the data using median centering	4
3.4	Explore normalized data	4
3.5	Summarization to protein level	6
4	Data Analysis	7
4.1	Estimation	7
4.2	Inference	7
4.3	Plots	8
5	Session Info	51

All materials are licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)

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/ μ L *Saccharomyces cerevisiae* strain BY4741. Two different spike-in concentrations were used: 6A (0.25 fmol UPS1 proteins/ μ L) and 6B (0.74 fmol UPS1 proteins/ μ L) [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.

2 Data

We first import the data from peptideRaws.txt file. This is the file containing your peptideRaw-level intensities. For a MaxQuant search [6], this peptideRaws.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 peptideRaws file which is a subset of the cptac study. This data is available in the `msdata` package. To import the data we use the `QFeatures` package.

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

We first import the data from peptideRaws.txt file. This is the file containing your peptideRaw-level intensities. For a MaxQuant search [6], this peptideRaws.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 peptideRaws file which is a subset of the cptac study. This data is available in the `msdata` package. To import the data we use the `QFeatures` package.

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

```
library(tidyverse)
library(limma)
library(QFeatures)
library(msqrob2)
library(plotly)
library(gridExtra)

peptidesFile <- "https://raw.githubusercontent.com/statOmics/SGA2020/data/quantification/cptacAvsB_lab3"

ecols <- grep("Intensity\\.", names(read.delim(peptidesFile)))

pe <- readQFeatures(
  table = peptidesFile, fnames = 1, ecol = ecols,
  name = "peptideRaw", sep = "\t"
)
```

```
library(tidyverse)
library(limma)
library(QFeatures)
library(msqrob2)
library(plotly)

peptidesFile <- "https://raw.githubusercontent.com/statOmics/SGA2020/data/quantification/cptacAvsB_lab3"

ecols <- grep(
  "Intensity\\.",
  names(read.delim(peptidesFile))
)

pe <- readQFeatures(
  table = peptidesFile,
```

```
fnames = 1,
ecol = ecols,
name = "peptideRaw", sep="\t")

colnames(pe)
```

```
## CharacterList of length 1
## ["peptideRaw"] Intensity.6A_7 Intensity.6A_8 ... Intensity.6B_9
```

In the following code chunk, we can extract the spikein condition from the raw file name.

```
cond <- which(
  strsplit(colnames(pe)[[1]][1], split = "")[[1]] == "A") # find where condition is stored

colData(pe)$condition <- substr(colnames(pe), cond, cond) %>%
  unlist %>%
  as.factor
```

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

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

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

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

2.1 Data exploration

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

3 Preprocessing

This section preforms preprocessing for the peptide data. This include

- log transformation,
- filtering and
- summarisation of the data.

Note, that the summarisation is done using median summarisation instead of the default that is robust summarisation. This is mainly for didactical purposes to see what the impact is of suboptimal preprocessing.

3.1 Log transform the data

```
pe <- logTransform(pe, base = 2, i = "peptideRaw", name = "peptideLog")
```

3.2 Filtering

1. Handling overlapping protein groups

In our approach a peptide can map to multiple proteins, as long as there is none of these proteins present in a smaller subgroup.

```
pe[["peptideLog"]] <-  
  pe[["peptideLog"]][rowData(pe[["peptideLog"]])$Proteins  
  %in% smallestUniqueGroups(rowData(pe[["peptideLog"]])$Proteins),]
```

2. Remove reverse sequences (decoys) and contaminants

We now remove the contaminants, peptides that map to decoy sequences, and proteins which were only identified by peptides with modifications.

```
pe[["peptideLog"]] <- pe[["peptideLog"]][rowData(pe[["peptideLog"]])$Reverse != "+", ]  
pe[["peptideLog"]] <- pe[["peptideLog"]][rowData(pe[["peptideLog"]])$  
  Potential.contaminant != "+", ]
```

3. Drop peptides that were only identified in one sample

We keep peptides that were observed at least twice.

```
pe[["peptideLog"]] <- pe[["peptideLog"]][rowData(pe[["peptideLog"]])$nNonZero >= 2, ]  
nrow(pe[["peptideLog"]])
```

```
## [1] 7011
```

We keep 7011 peptides upon filtering.

3.3 Normalize the data using median centering

We normalize the data by subtracting 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 = "peptideLog",  
  name = "peptideNorm",  
  method = "center.median")
```

3.4 Explore normalized data

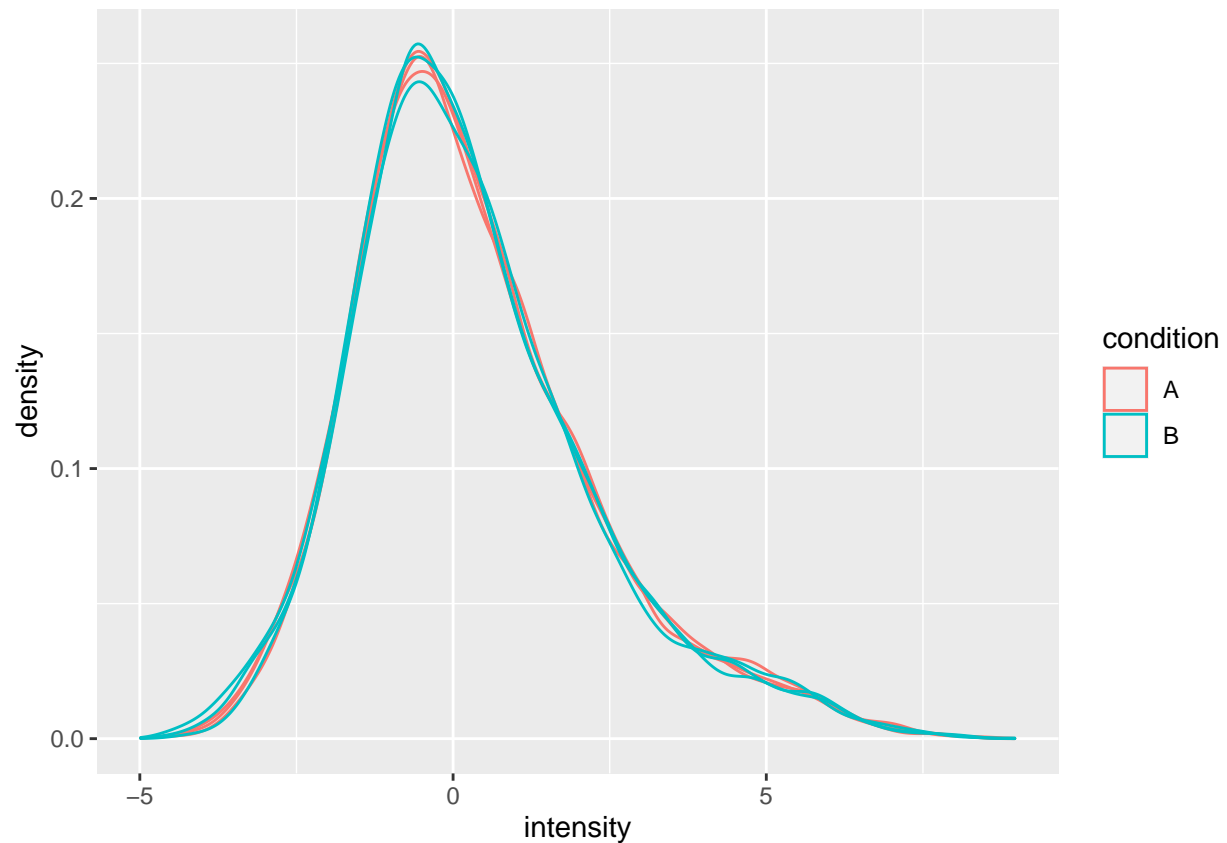
Upon the normalisation the density curves are nicely registered

```

pe[["peptideNorm"]] %>%
  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 8167 rows containing non-finite values (stat_density).



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

```

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

```



The first axis in the plot is showing the leading log fold changes (differences on the log scale) between the samples.

We notice that the leading differences (log FC) in the peptide data seems to be driven by technical variability. Indeed, the samples do not seem to be clearly separated according to the spike-in condition.

3.5 Summarization to protein level

- By default robust summarization is used: `fun = MsCoreUtils::robustSummary()`

```
pe <- aggregateFeatures(pe,
  i = "peptideNorm",
  fcol = "Proteins",
  na.rm = TRUE,
  name = "protein")
```

```
## Your quantitative and row data contain missing values. Please read the
## relevant section(s) in the aggregateFeatures manual page regarding the
## effects of missing values on data aggregation.
```

```
plotMDS(assay(pe[["protein"]]), col = as.numeric(colData(pe)$condition))
```



Note that the samples upon robust summarisation show a clear separation in the second dimension of the MDS plot.

4 Data Analysis

4.1 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)
```

4.2 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]])
```

```
## (Intercept) conditionB
## -2.672396 1.513682
```

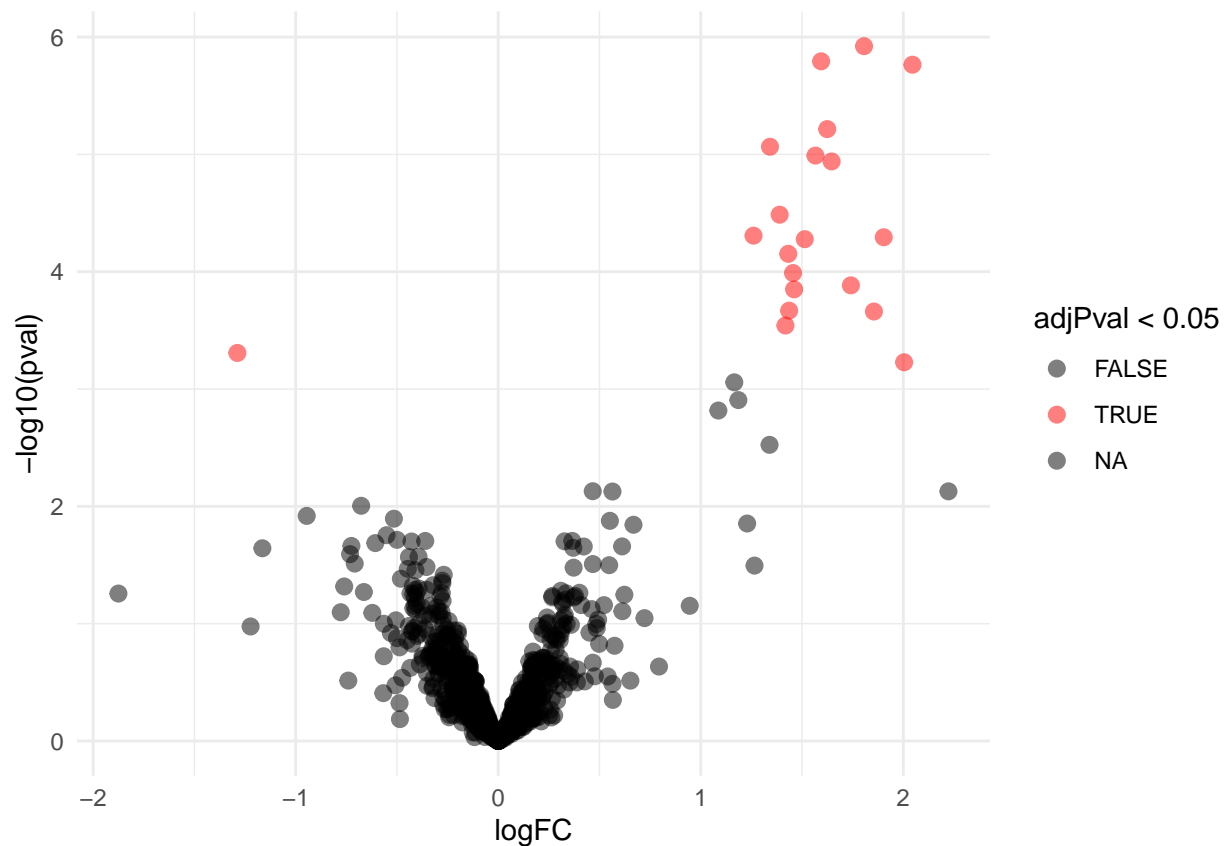
Spike-in condition A is the reference class. So the mean log₂ expression for samples from condition A is '(Intercept)'. The mean log₂ expression for samples from condition B is '(Intercept)+conditionB'. Hence, the average log₂ fold change between condition b and condition a is modelled using the parameter '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)
```

4.3 Plots

4.3.1 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
```

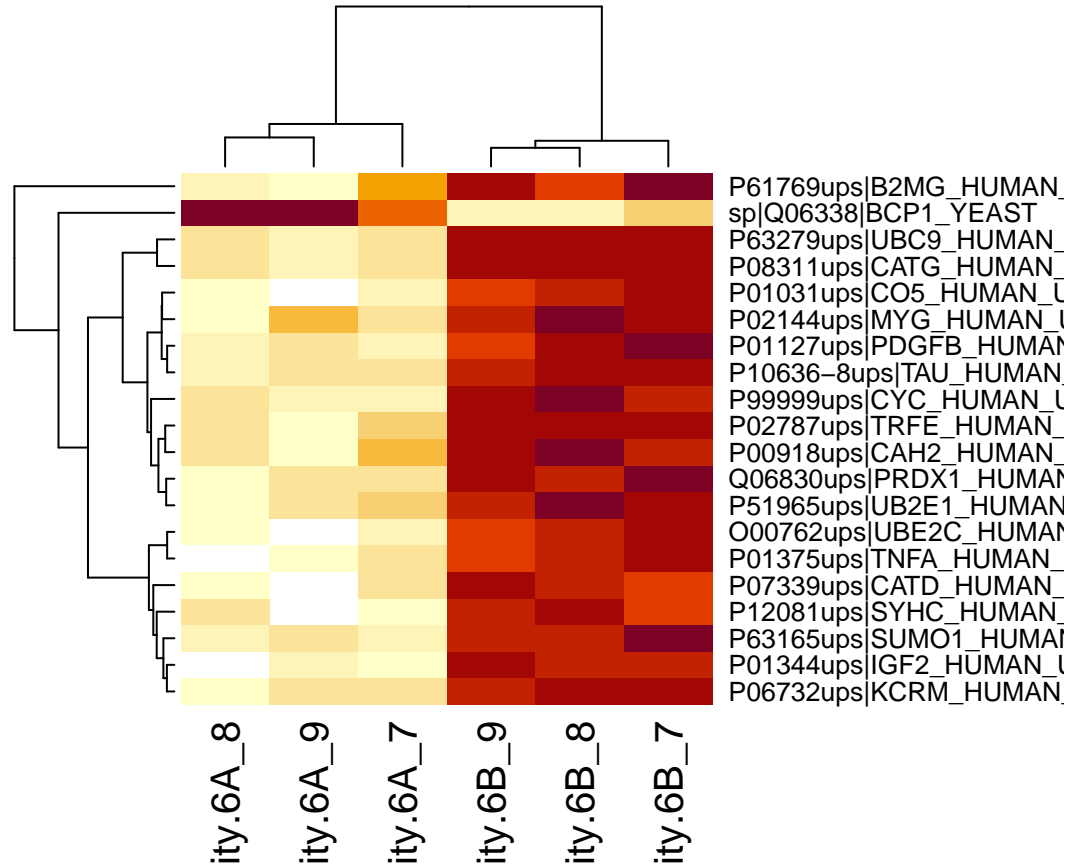


Note, that now 20 proteins are found to be differentially abundant!

4.3.2 Heatmap

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

```
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.

4.3.3 Boxplots

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

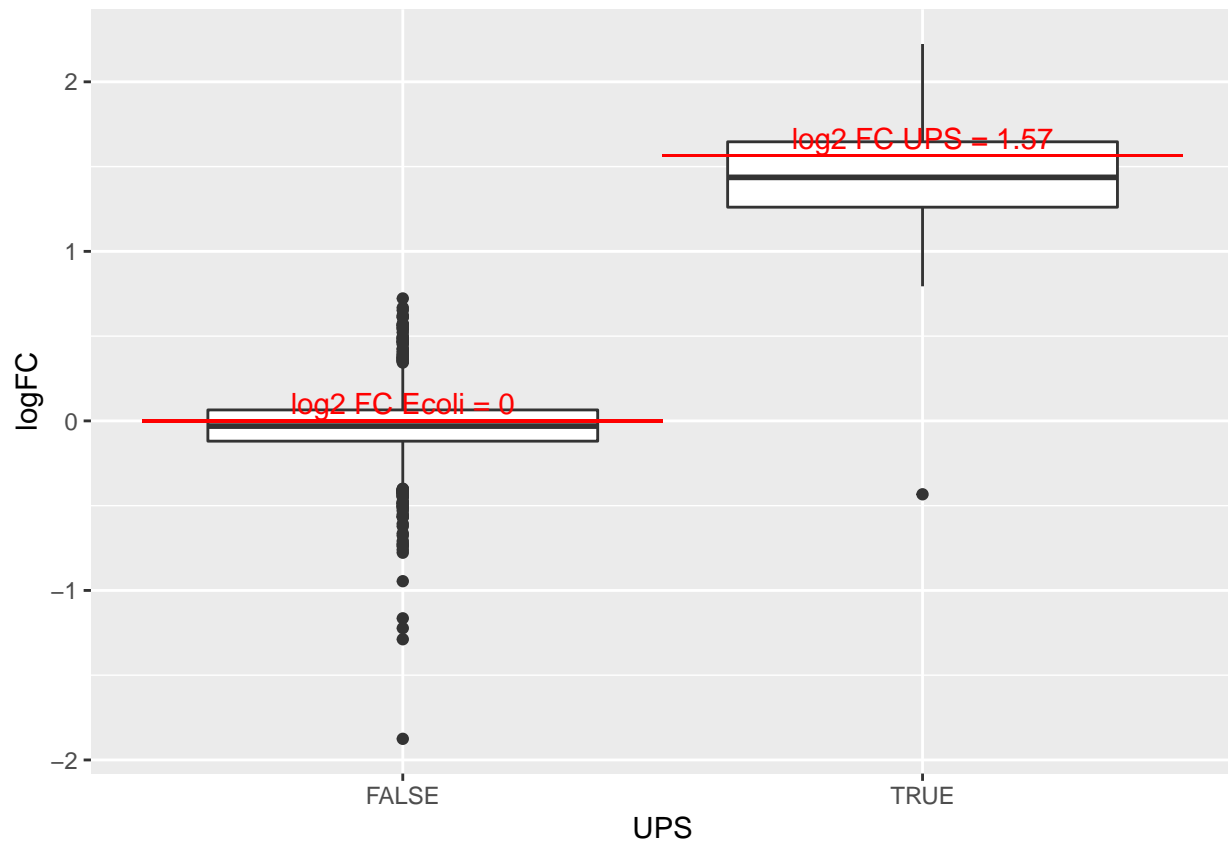
```
rowData(pe[["protein"]])$conditionB %>%
  rownames_to_column(var = "protein") %>%
  ggplot(aes(x=grepl("UPS",protein),y=logFC)) +
  geom_boxplot() +
  xlab("UPS") +
  geom_segment(
    x = 1.5,
    xend = 2.5,
```

```

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 167 rows containing non-finite values (stat_boxplot).



What do you observe?

4.3.4 Detail plots

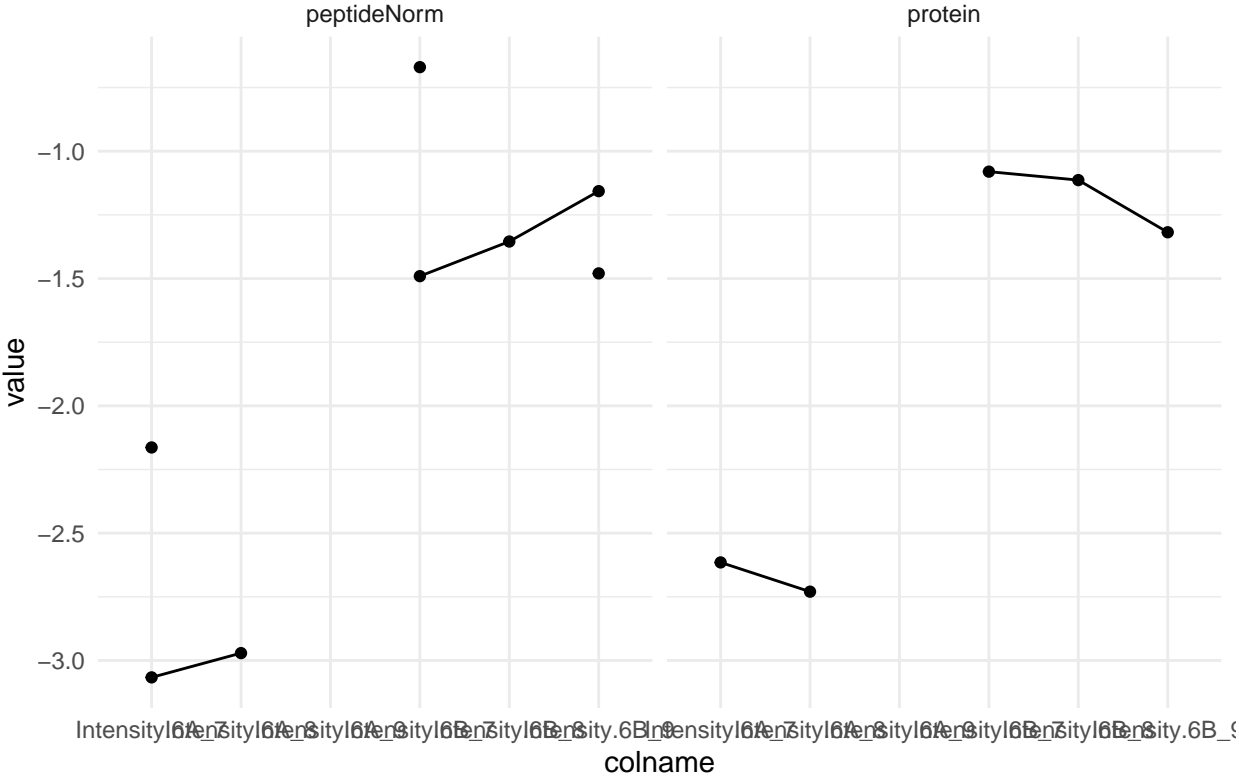
We first extract the normalized peptideRaw expression values for a particular protein.

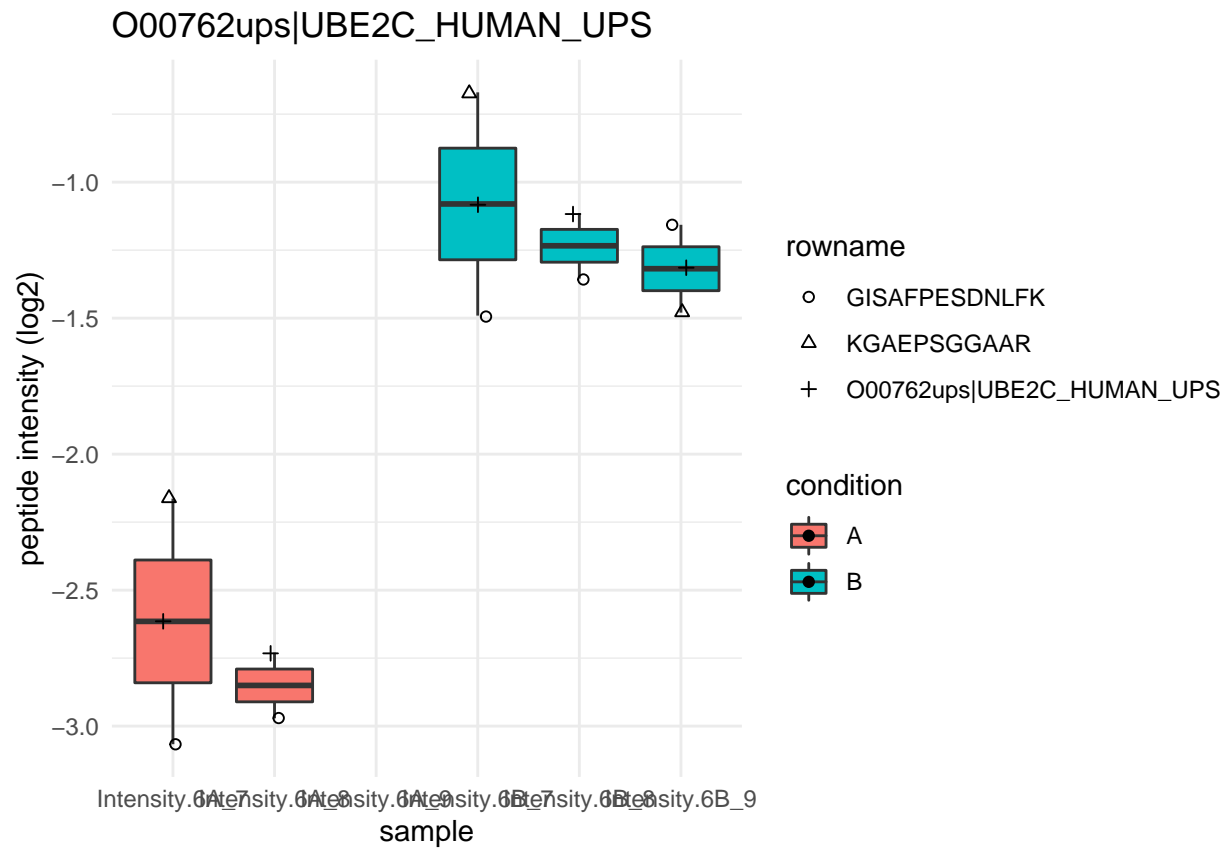
```
for (protName in sigNames)
{
  pePlot <- pe[protName, , c("peptideNorm","protein")]
  pePlotDf <- data.frame(longFormat(pePlot))
  pePlotDf$assay <- factor(pePlotDf$assay,
                          levels = c("peptideNorm", "protein"))
  pePlotDf$condition <- as.factor(colData(pePlot)[pePlotDf$colname, "condition"])

  # plotting
  p1 <- ggplot(data = pePlotDf,
               aes(x = colname, y = value, group = rowname)) +
    geom_line() + geom_point() + theme_minimal() +
    facet_grid(~assay) + ggtitle(protName)
  print(p1)

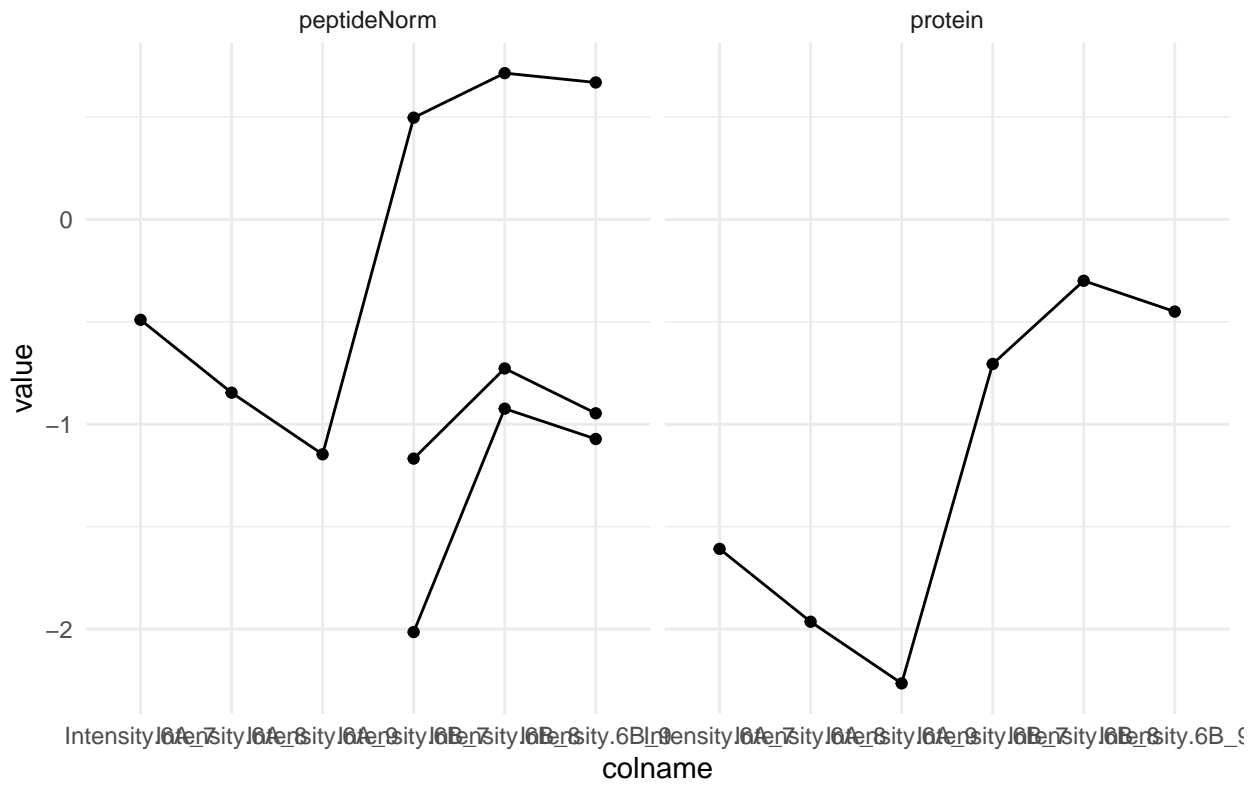
  # plotting 2
  p2 <- ggplot(pePlotDf, aes(x = colname, y = value, fill = condition)) +
    geom_boxplot(outlier.shape = NA) + geom_point(position = position_jitter(width = .1),
                                                  aes(shape = rowname)) +
    scale_shape_manual(values = 1:nrow(pePlotDf)) +
    labs(title = protName, x = "sample", y = "peptide intensity (log2)") + theme_minimal()
  facet_grid(~assay)
  print(p2)
}
```

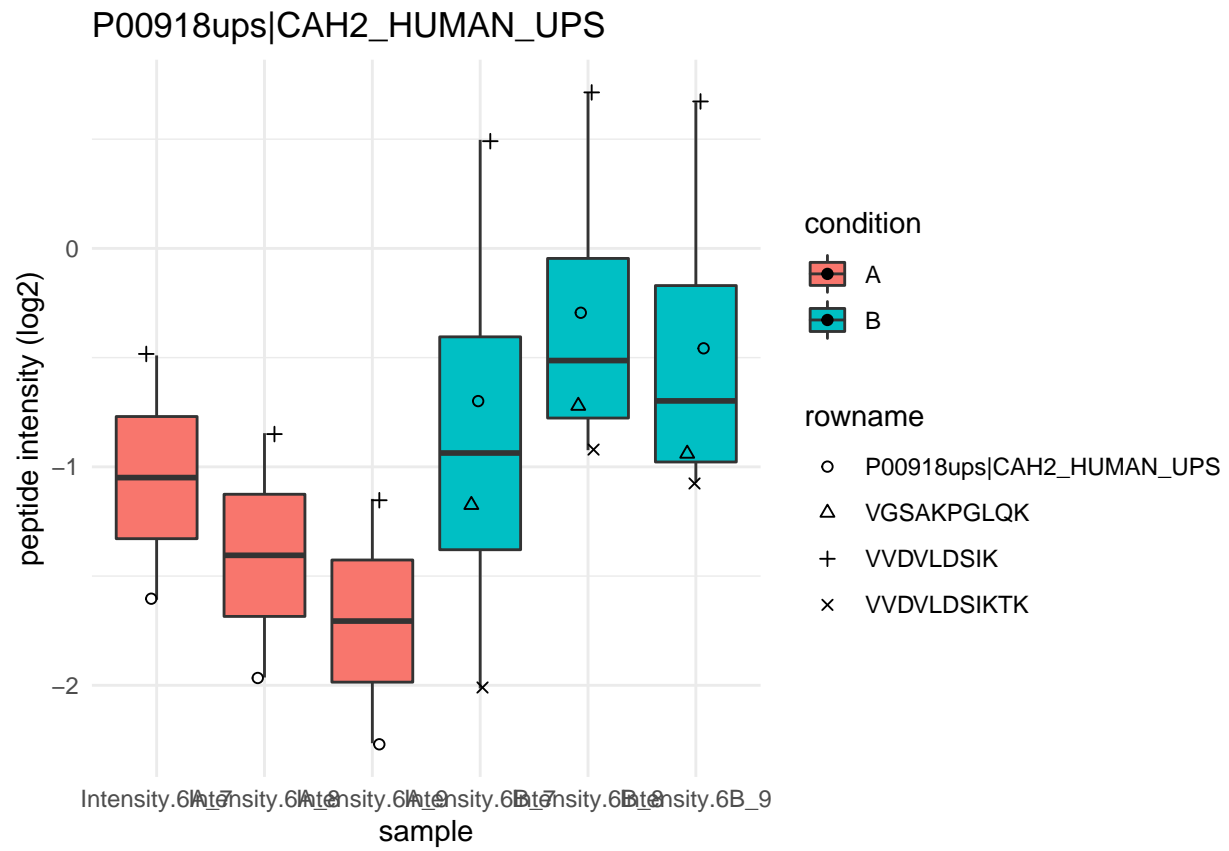
O00762ups|UBE2C_HUMAN_UPS

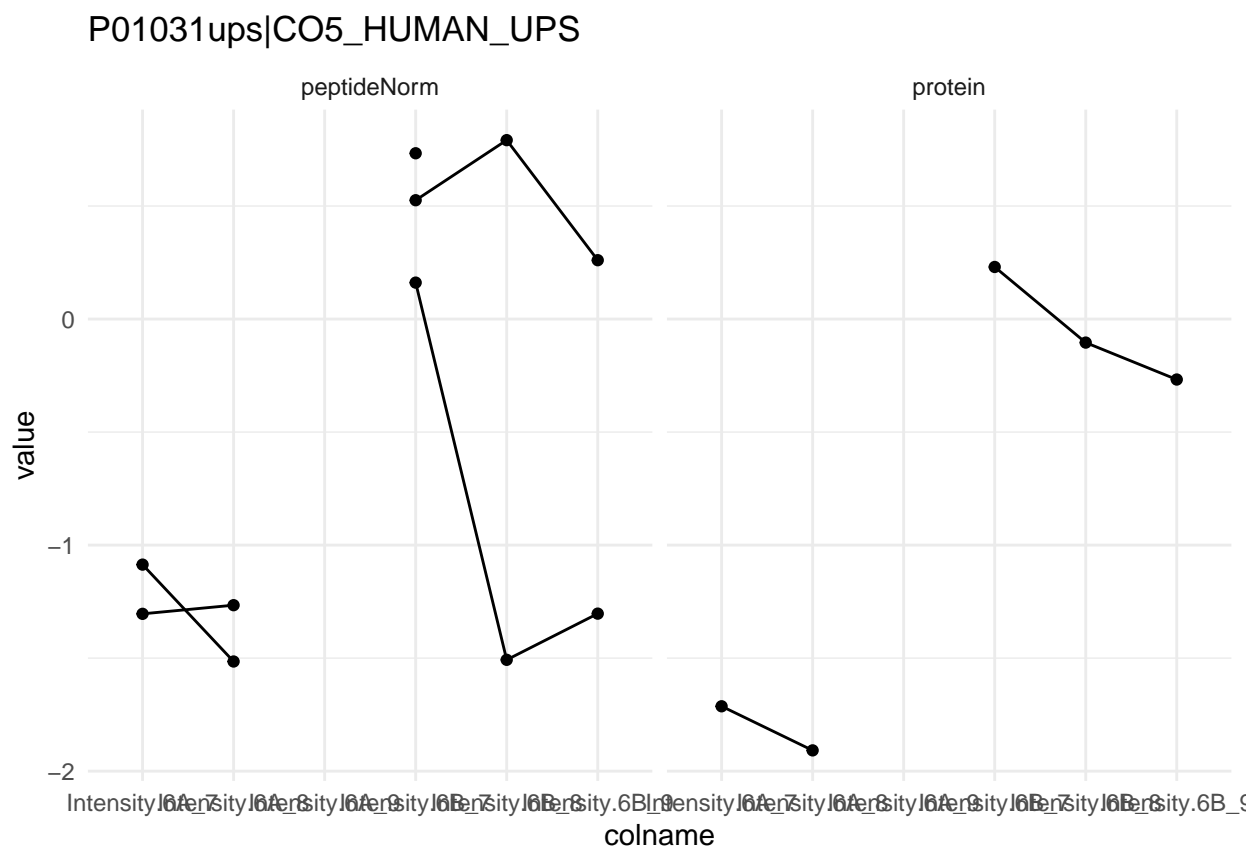


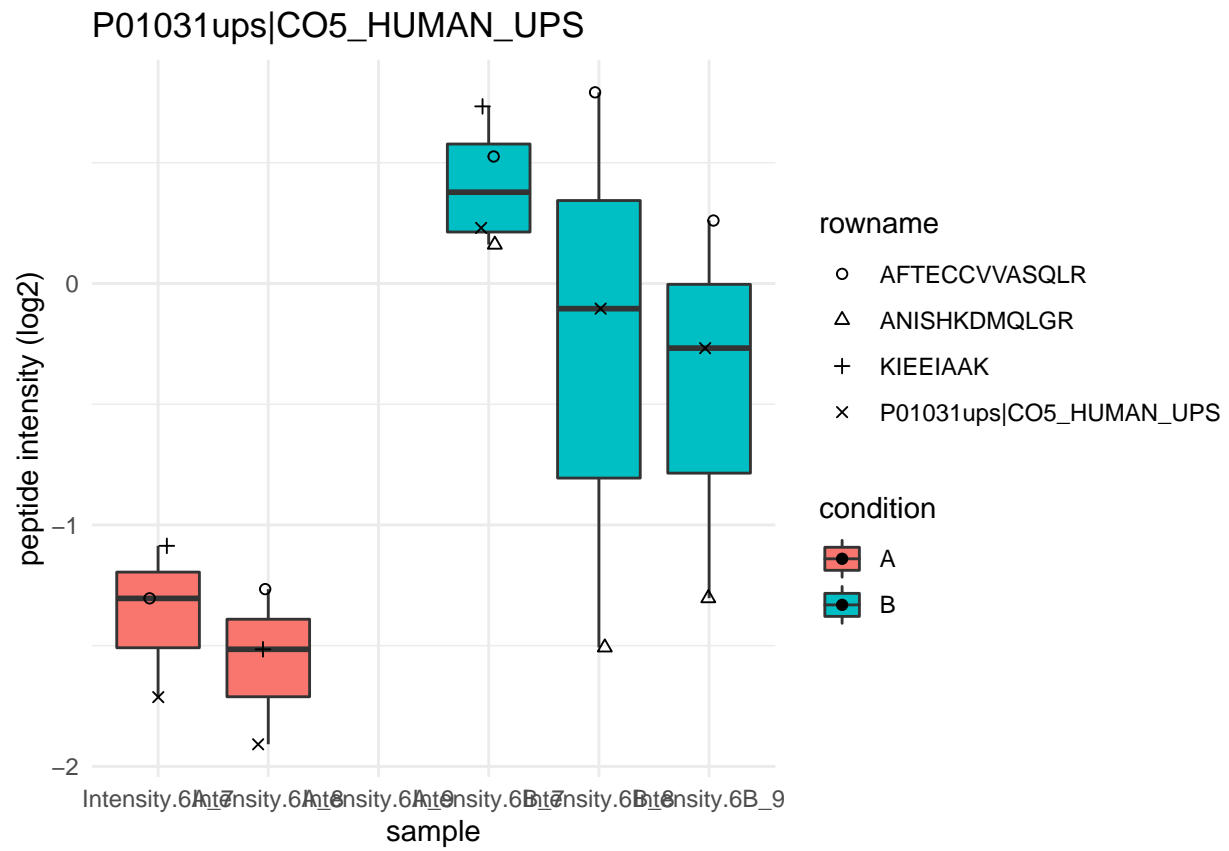


P00918ups|CAH2_HUMAN_UPS

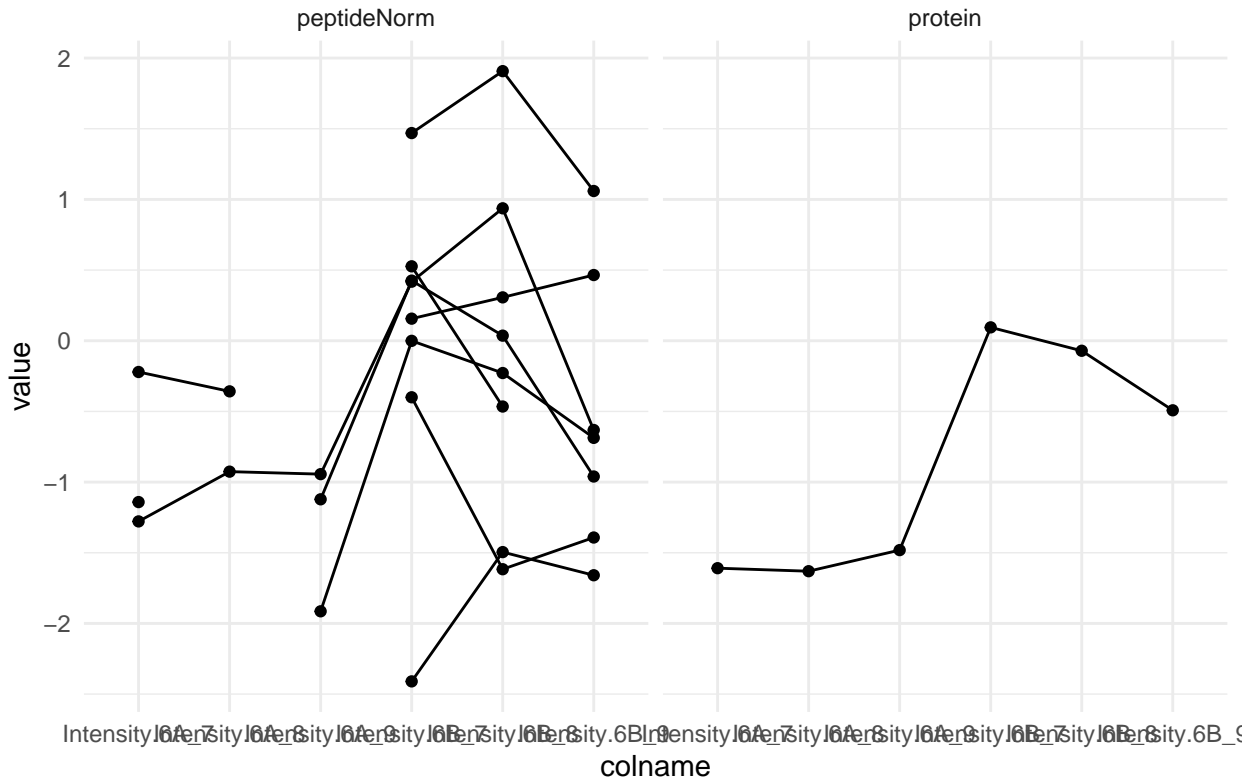


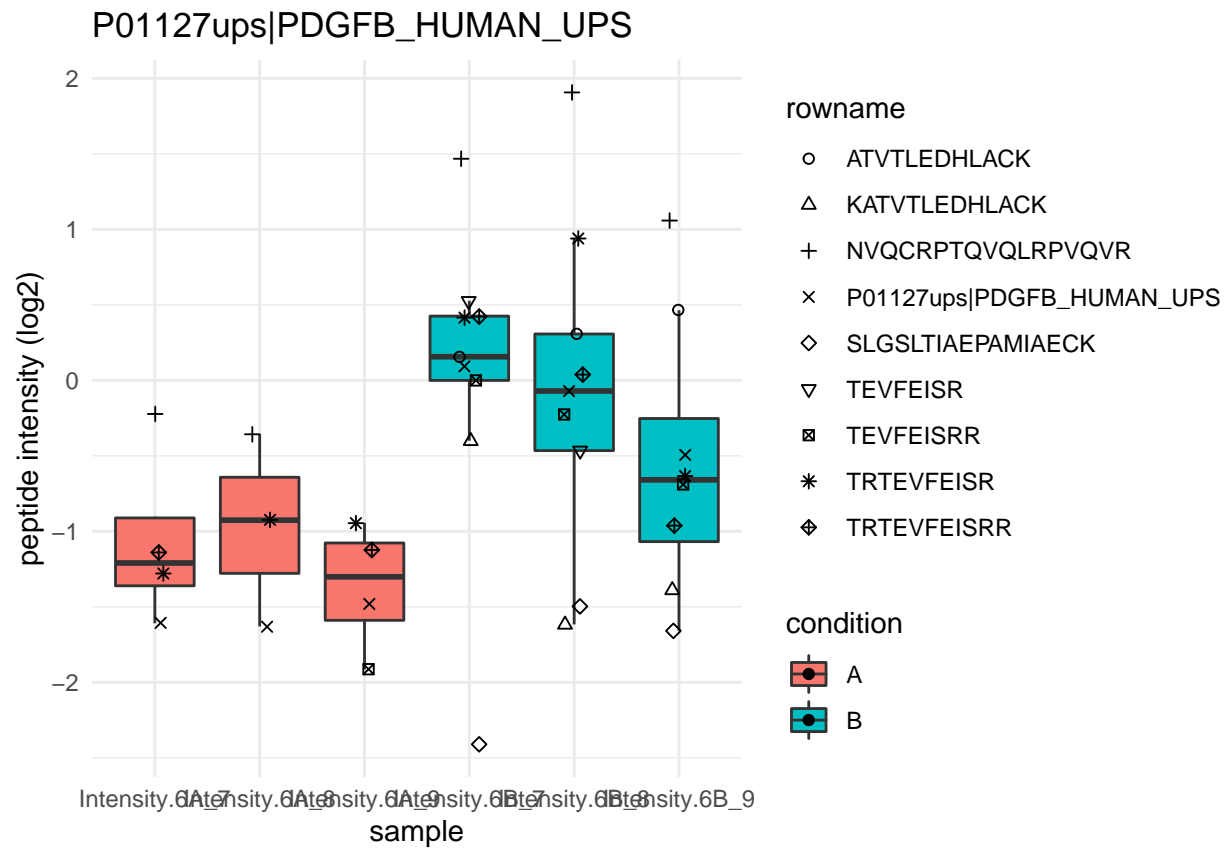






P01127ups|PDGFB_HUMAN_UPS

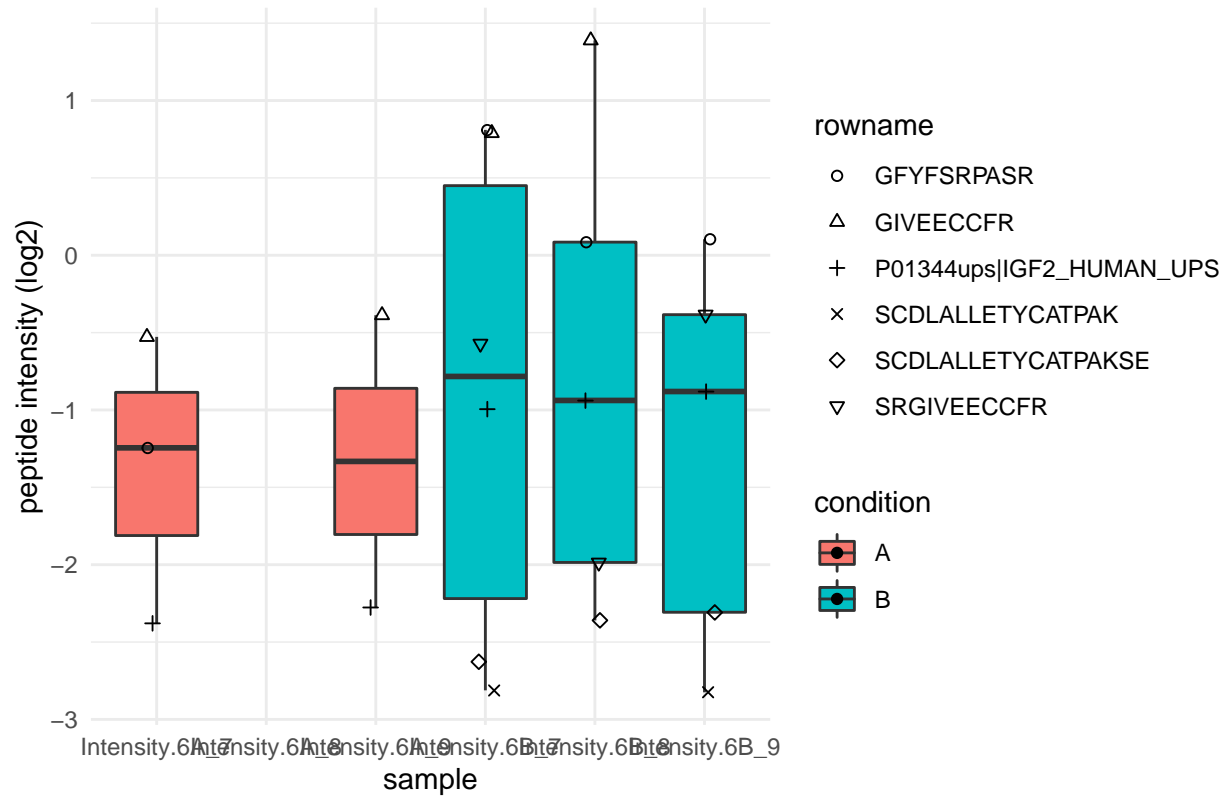




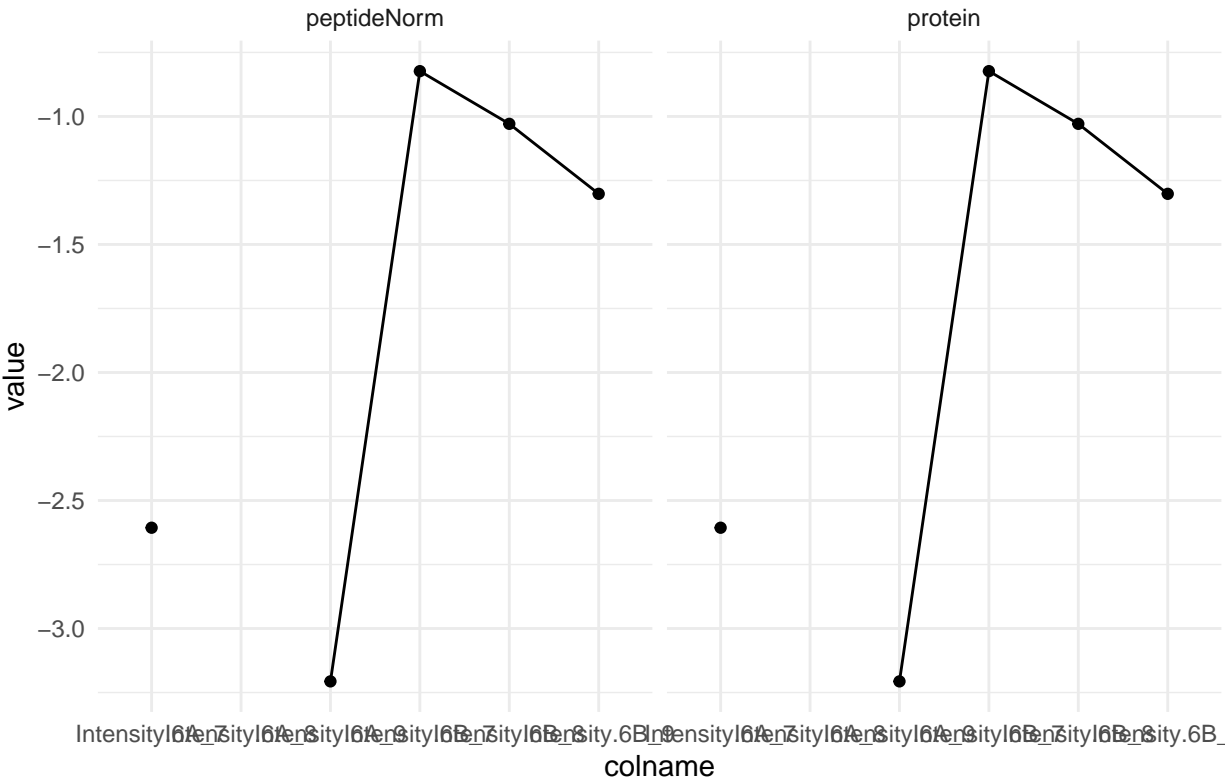
P01344ups|IGF2_HUMAN_UPS

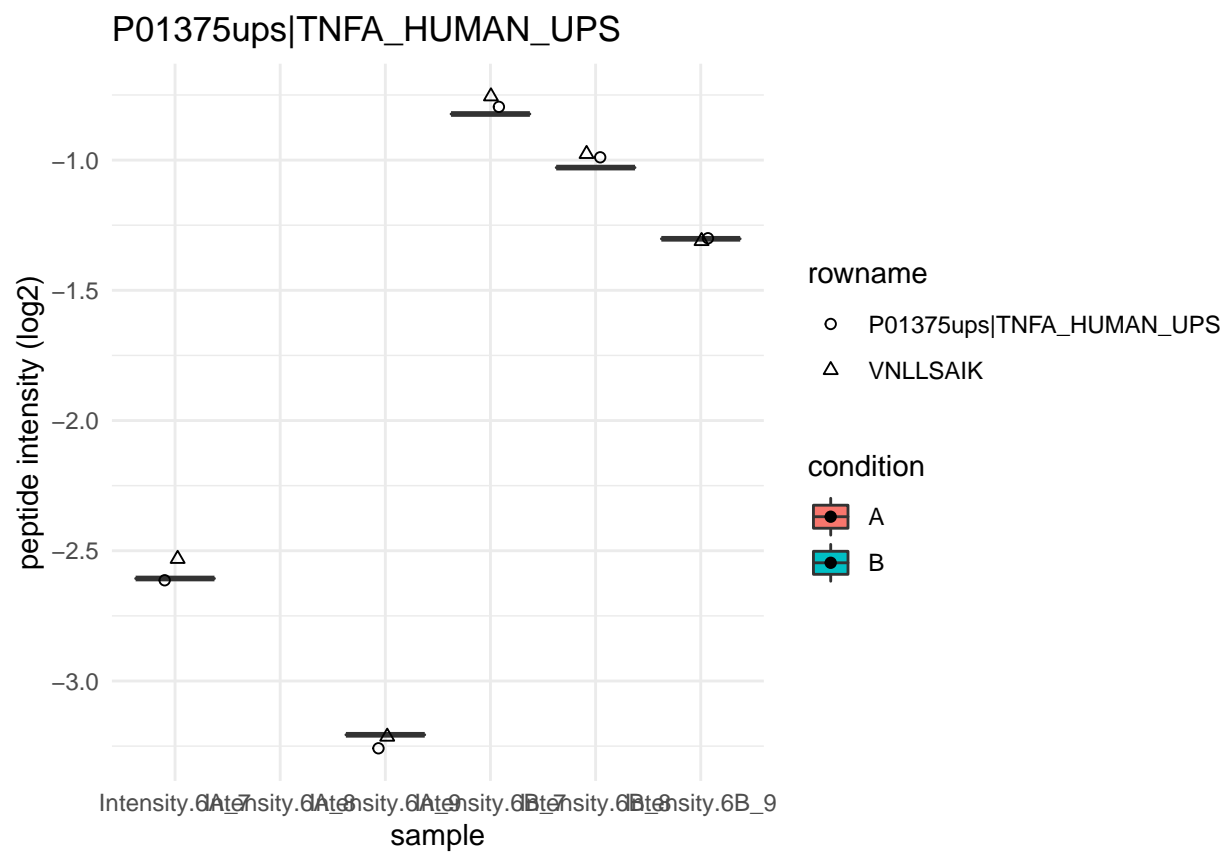


P01344ups|IGF2_HUMAN_UPS

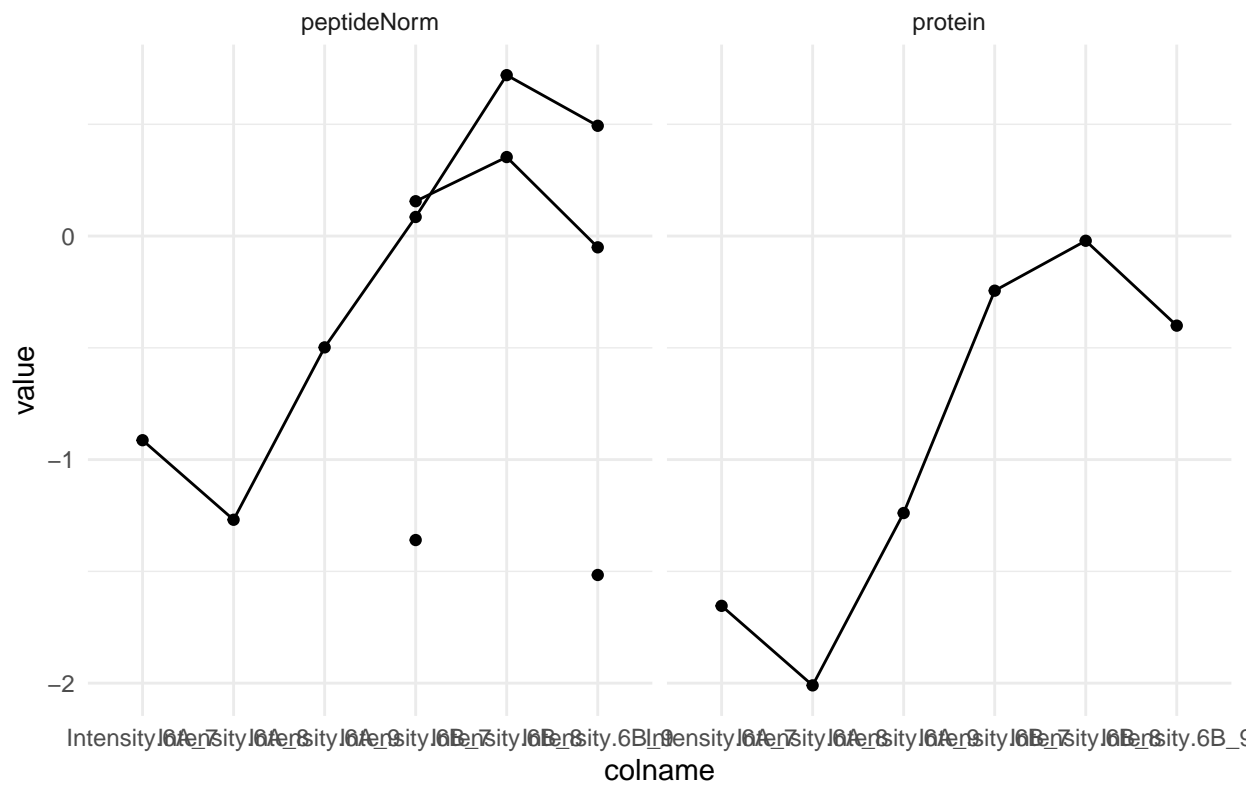


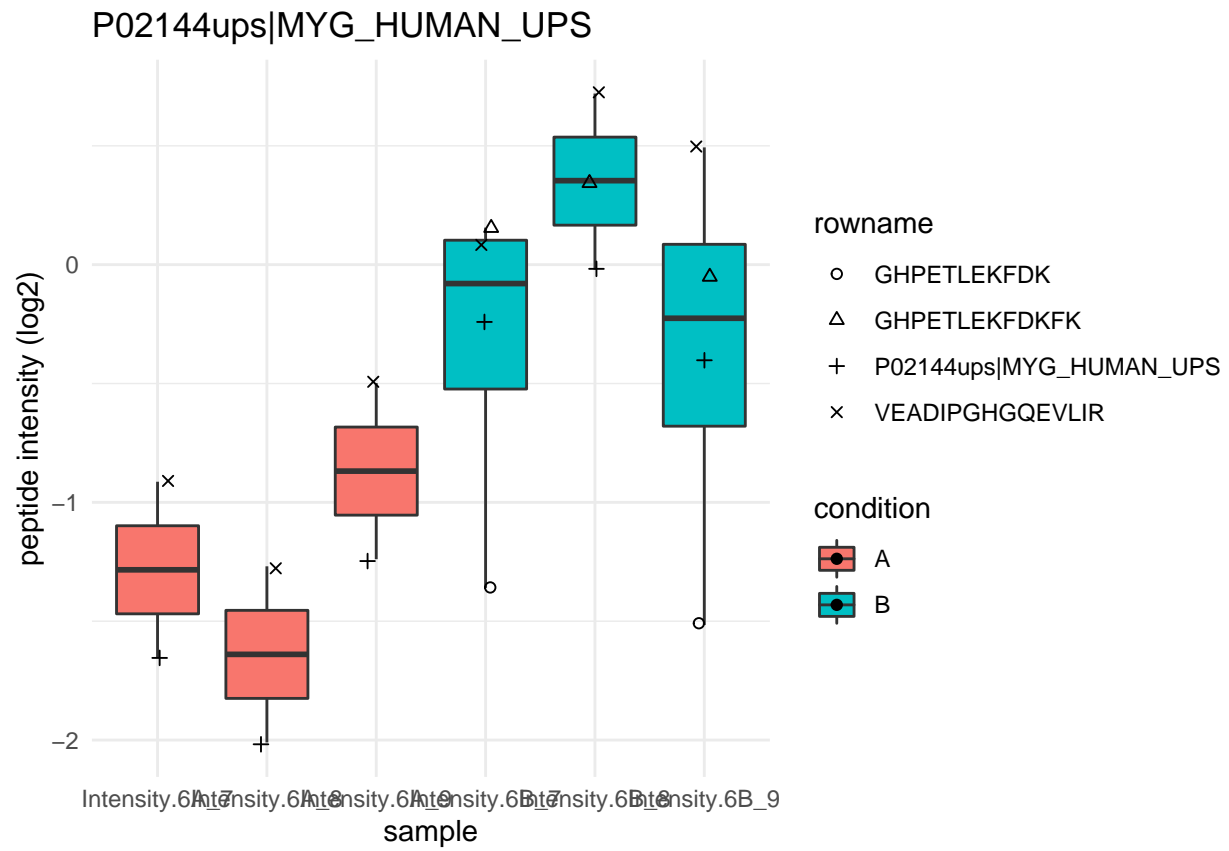
P01375ups|TNFA_HUMAN_UPS

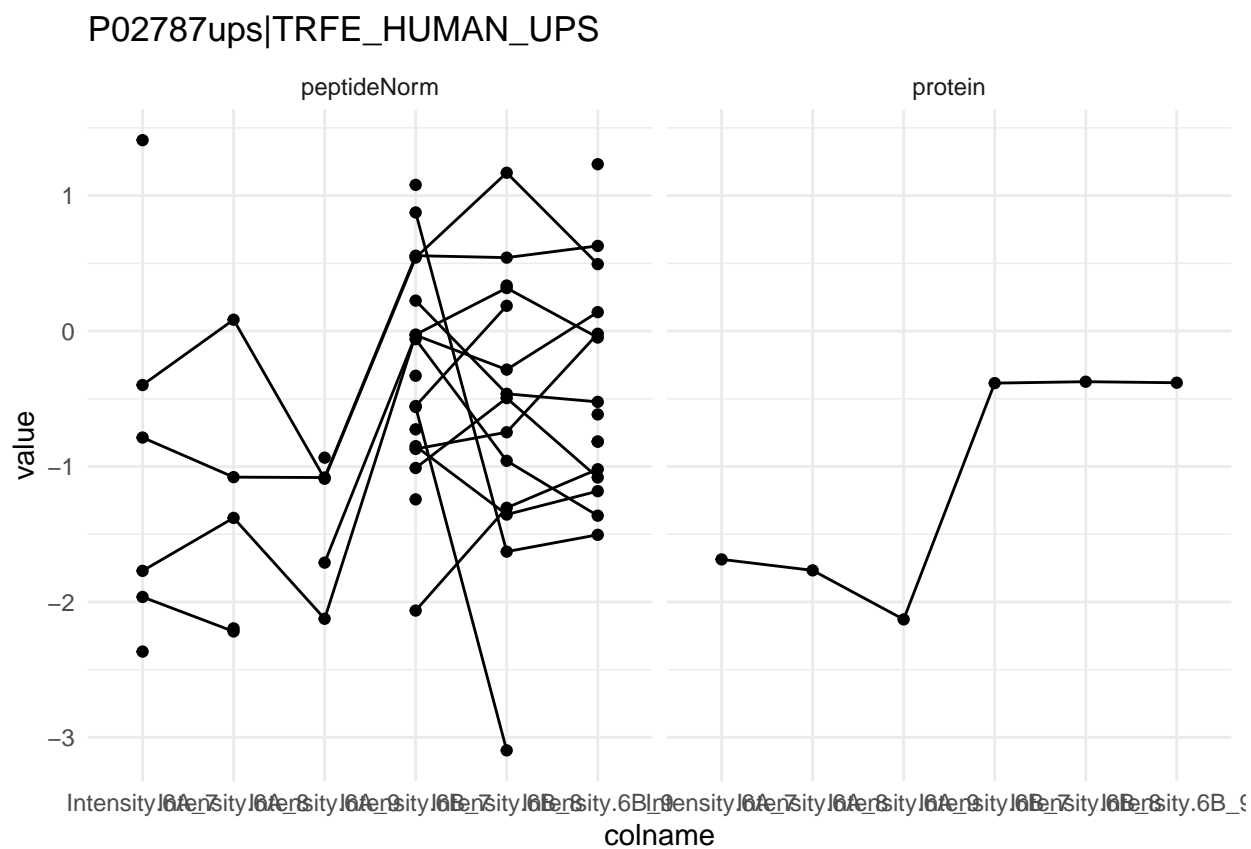


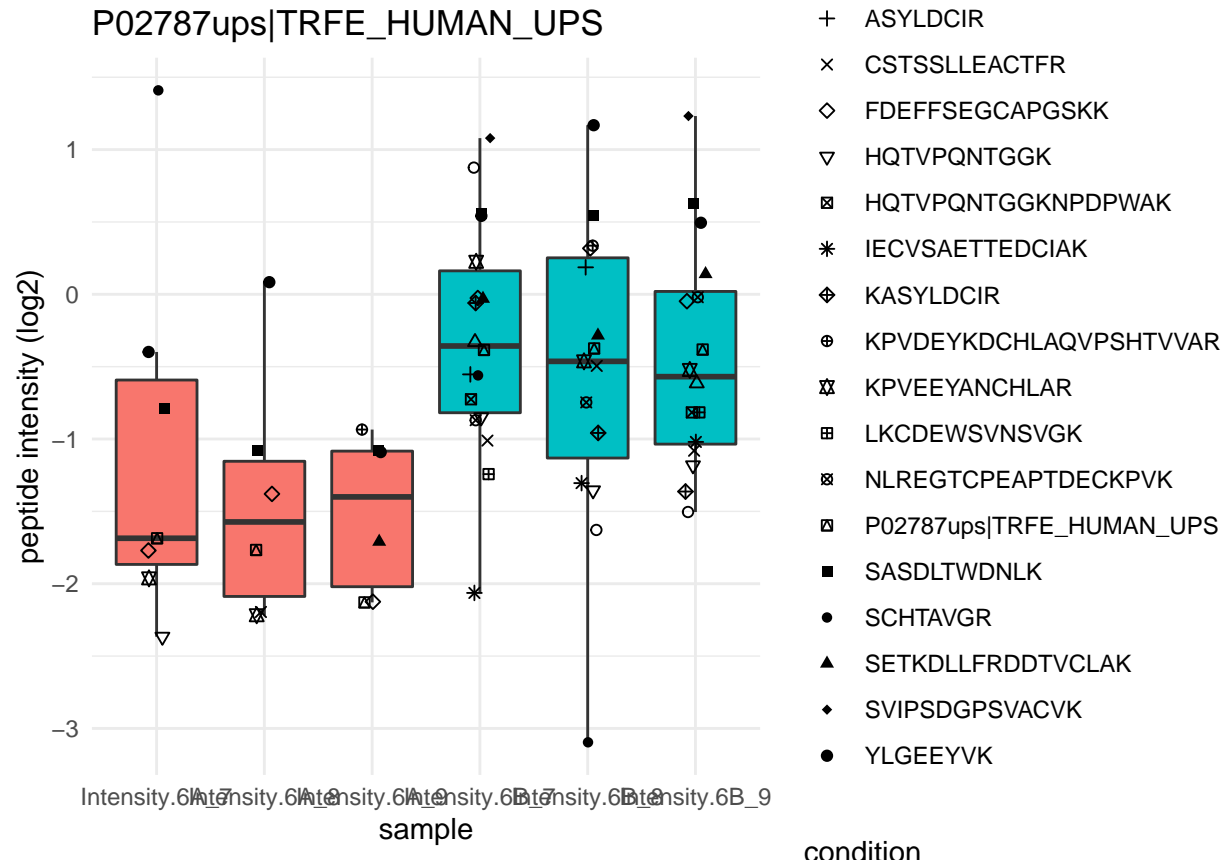


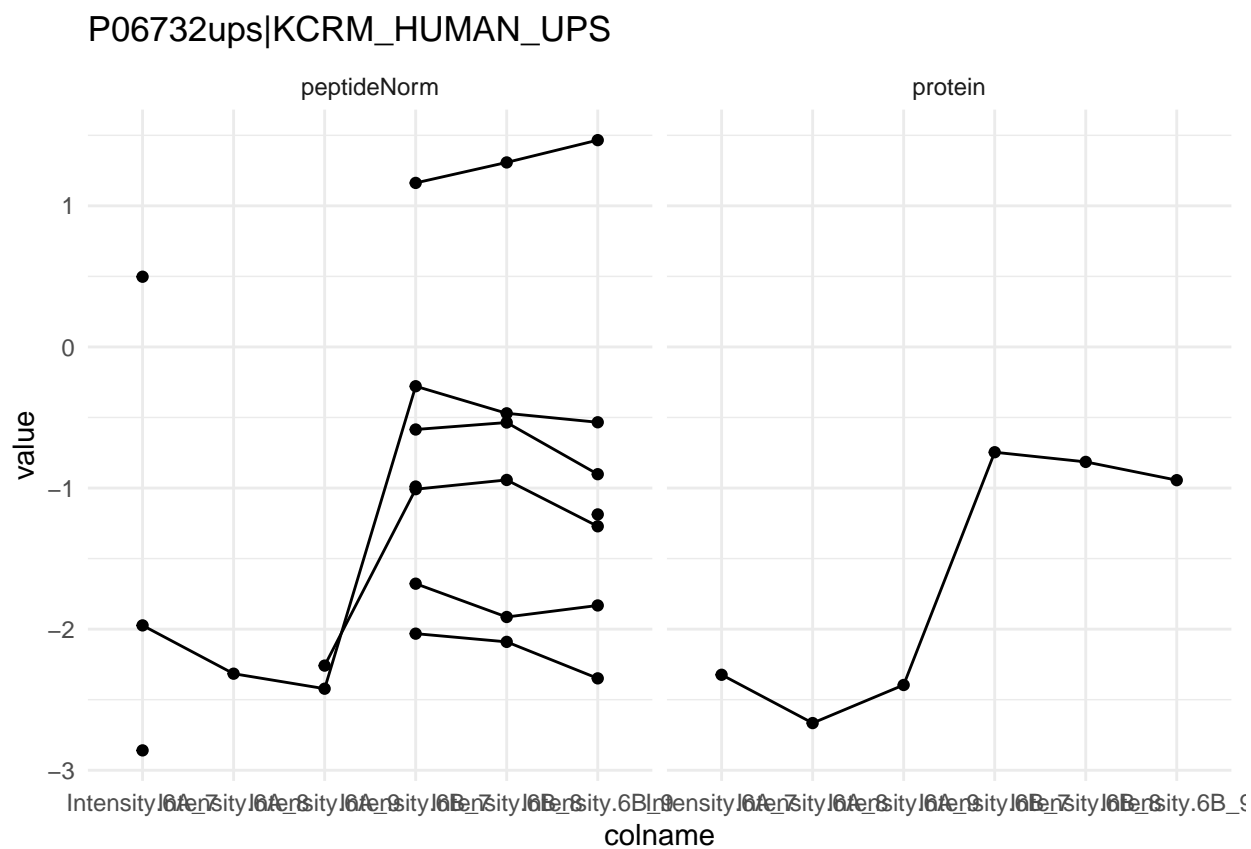
P02144ups|MYG_HUMAN_UPS

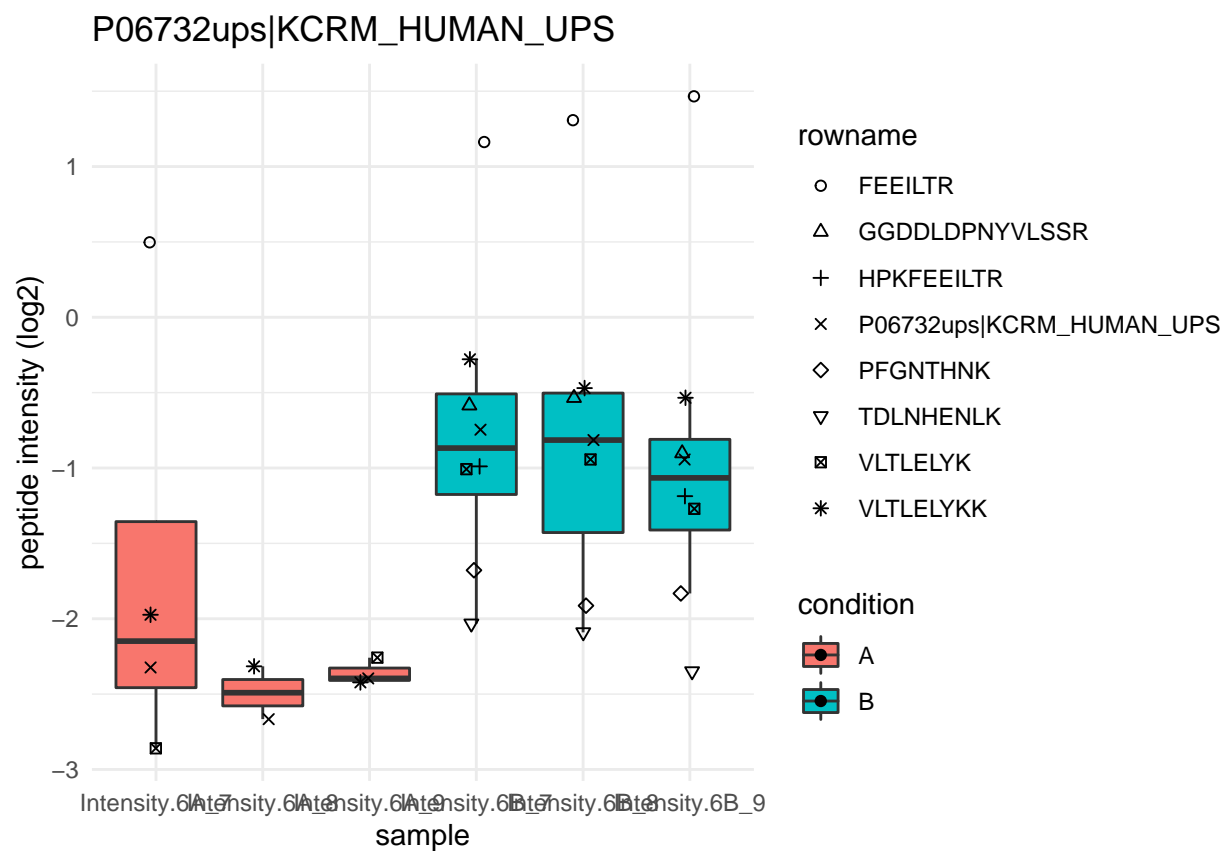




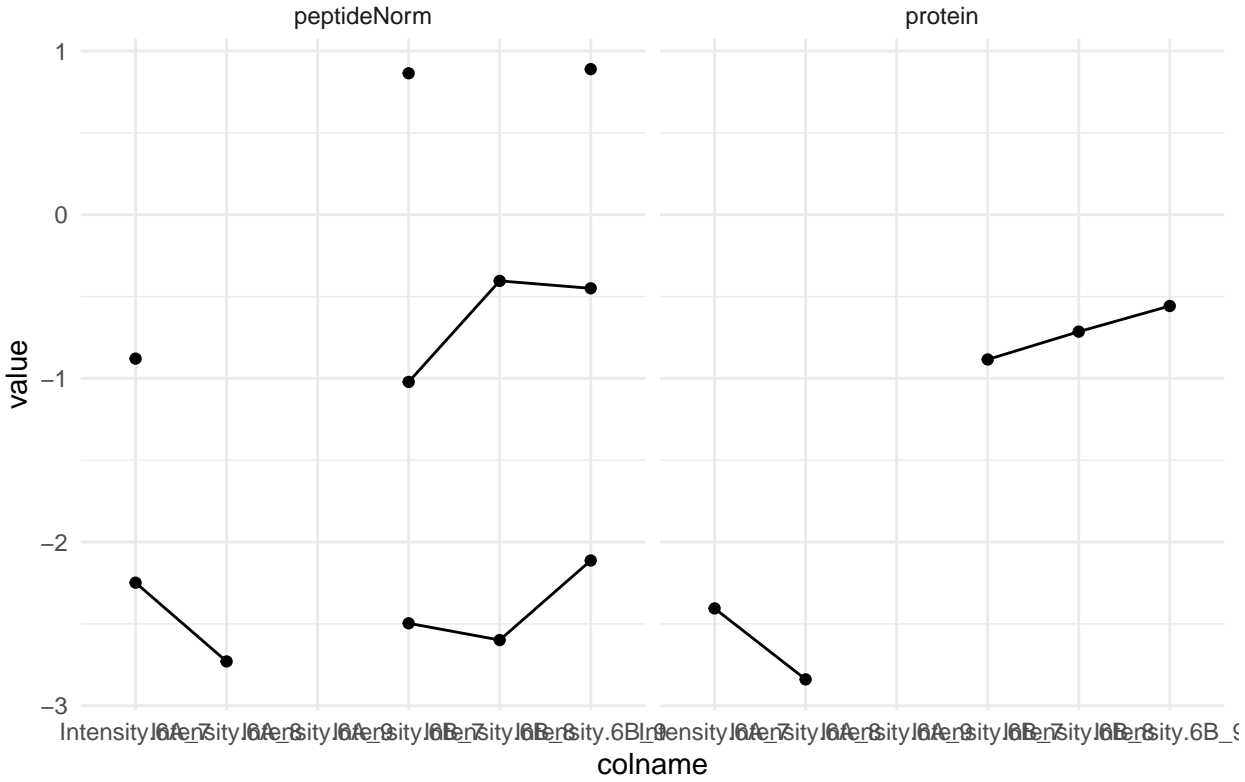


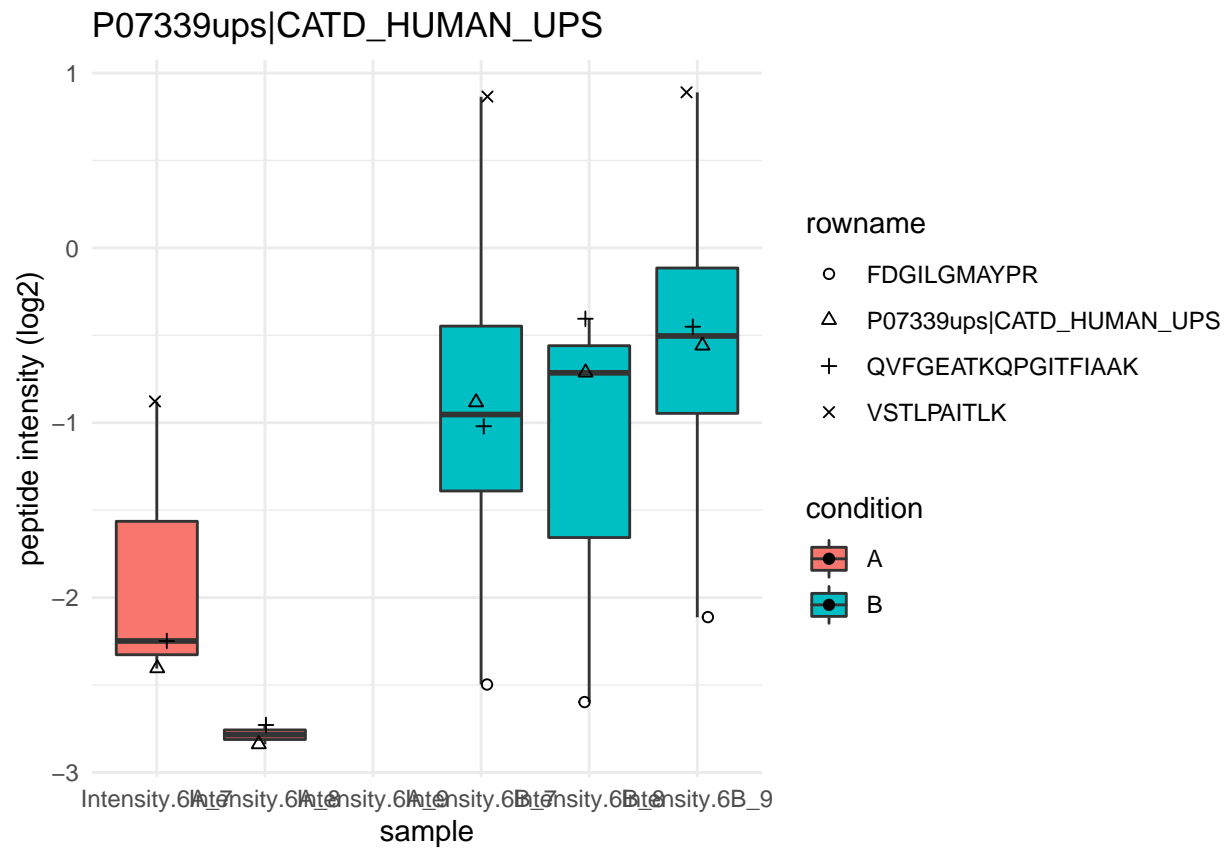


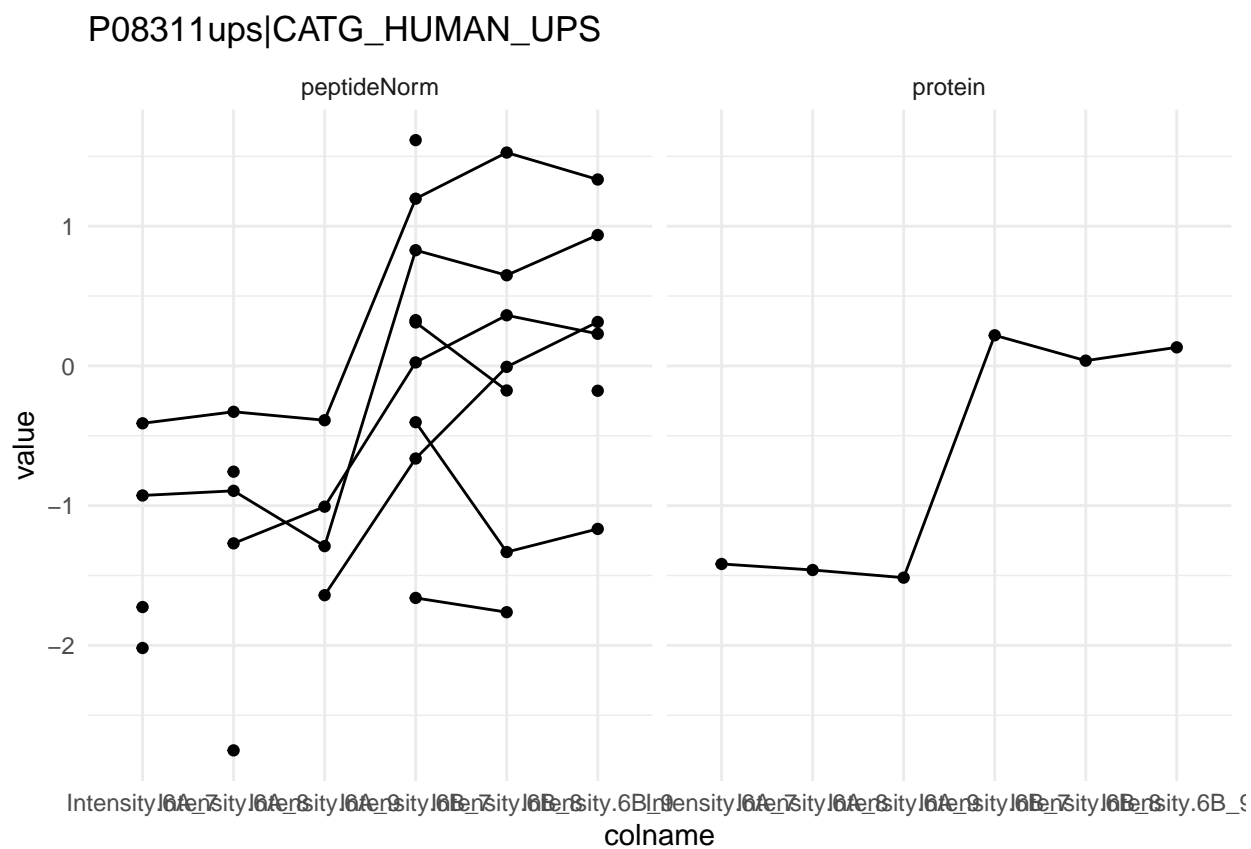


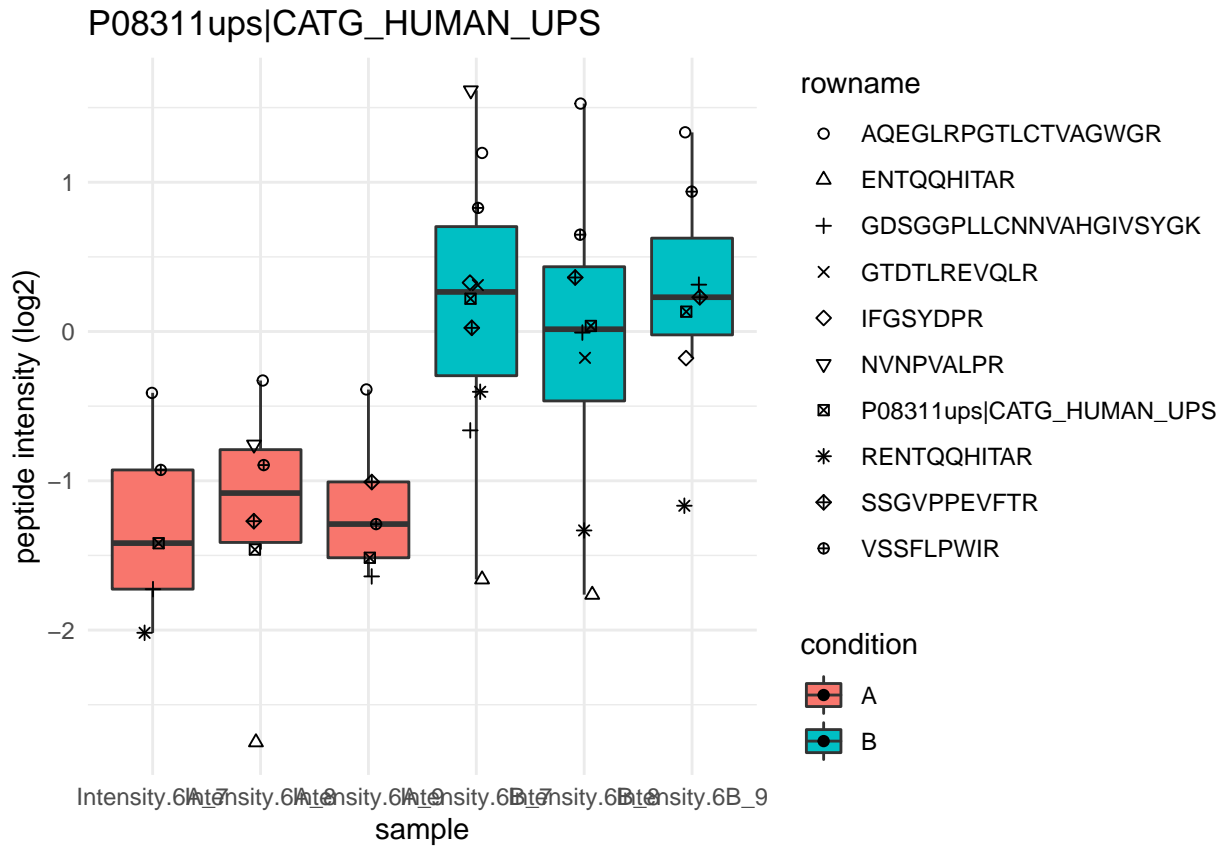


P07339ups|CATD_HUMAN_UPS

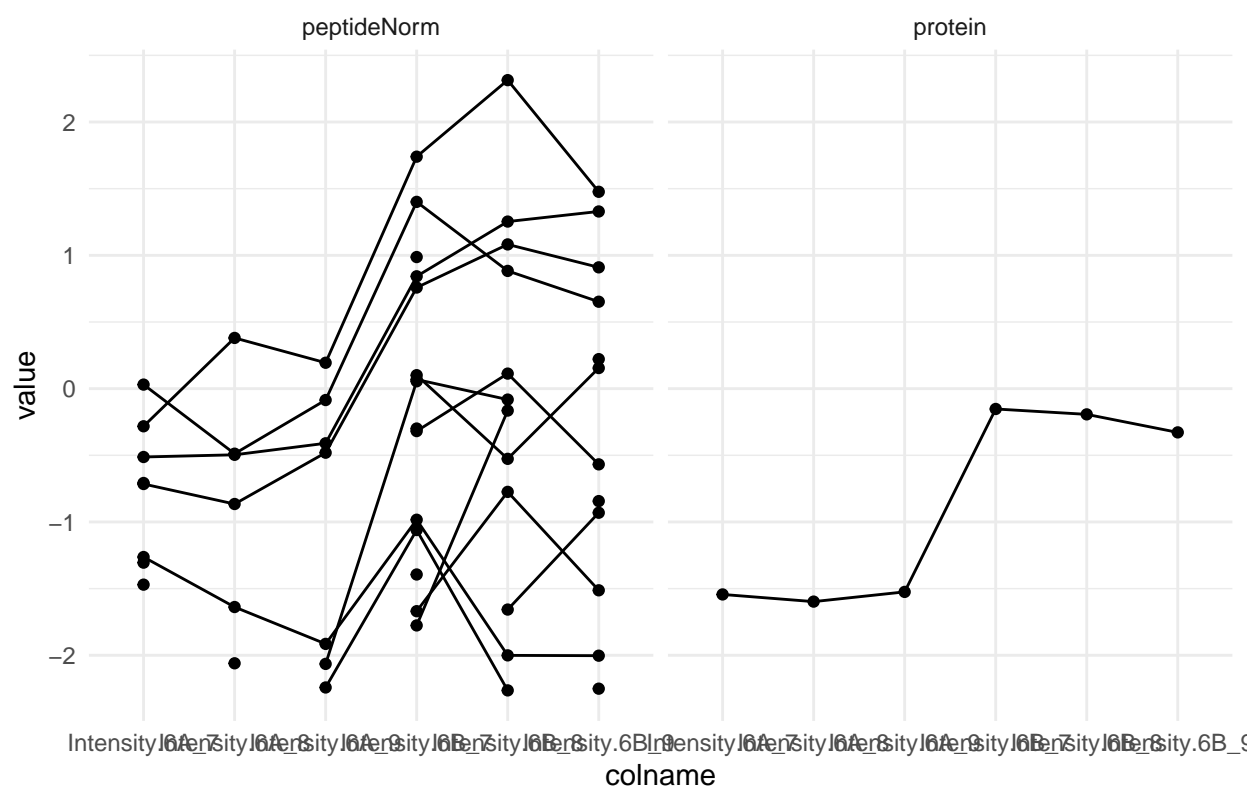


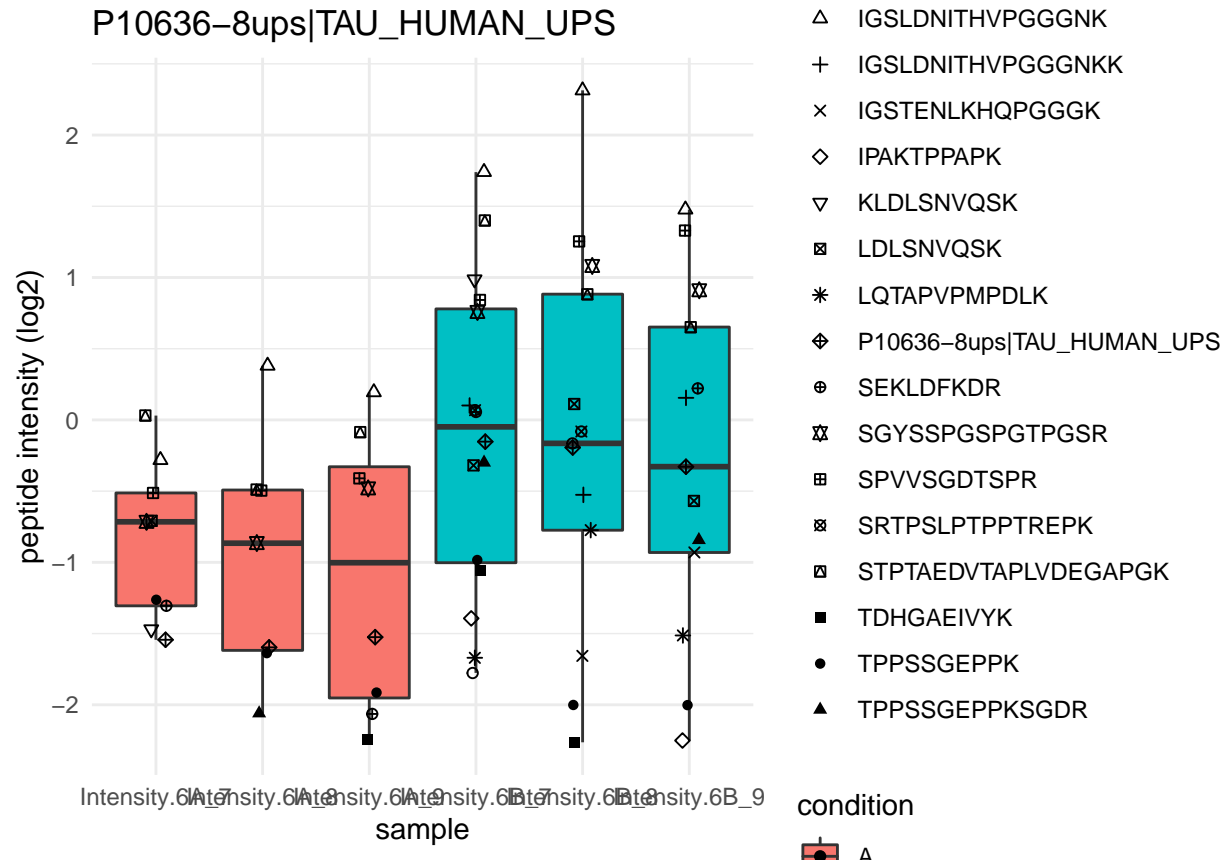


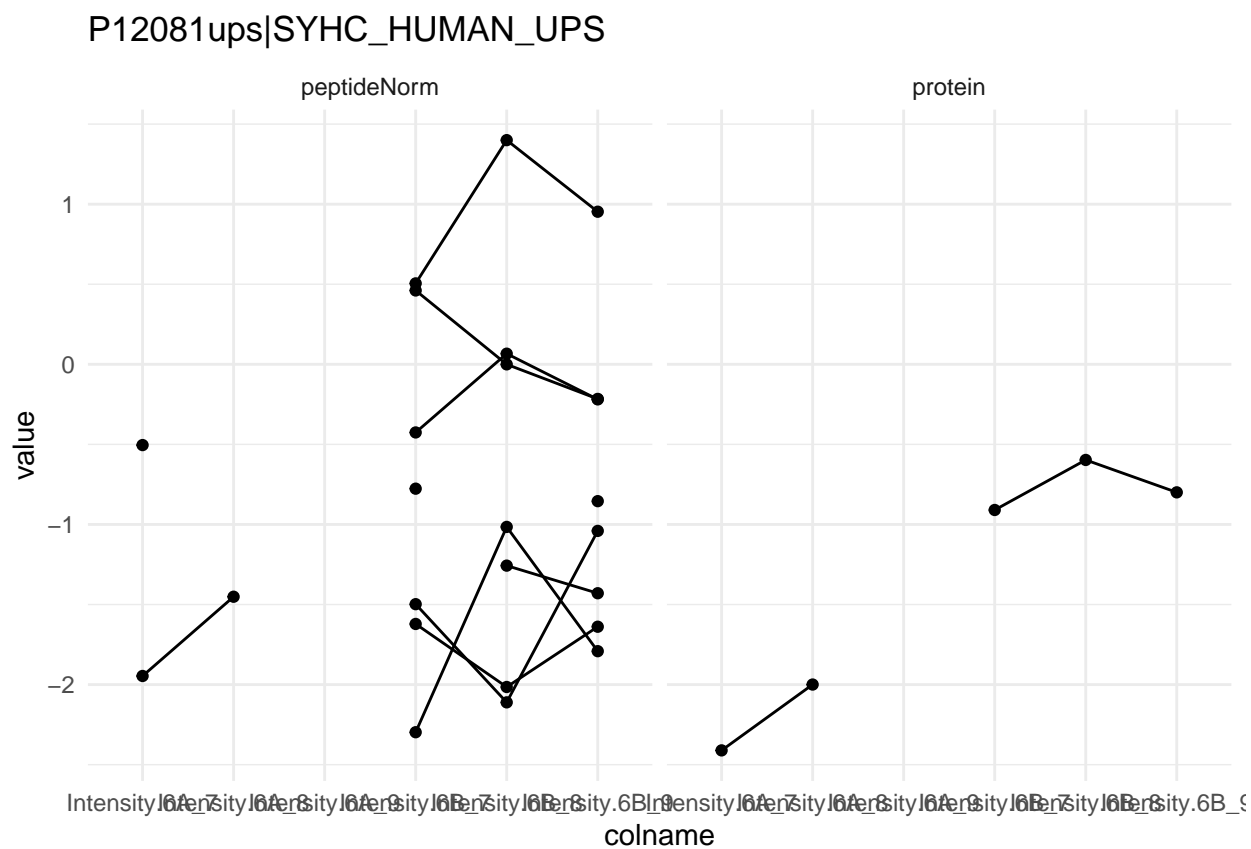




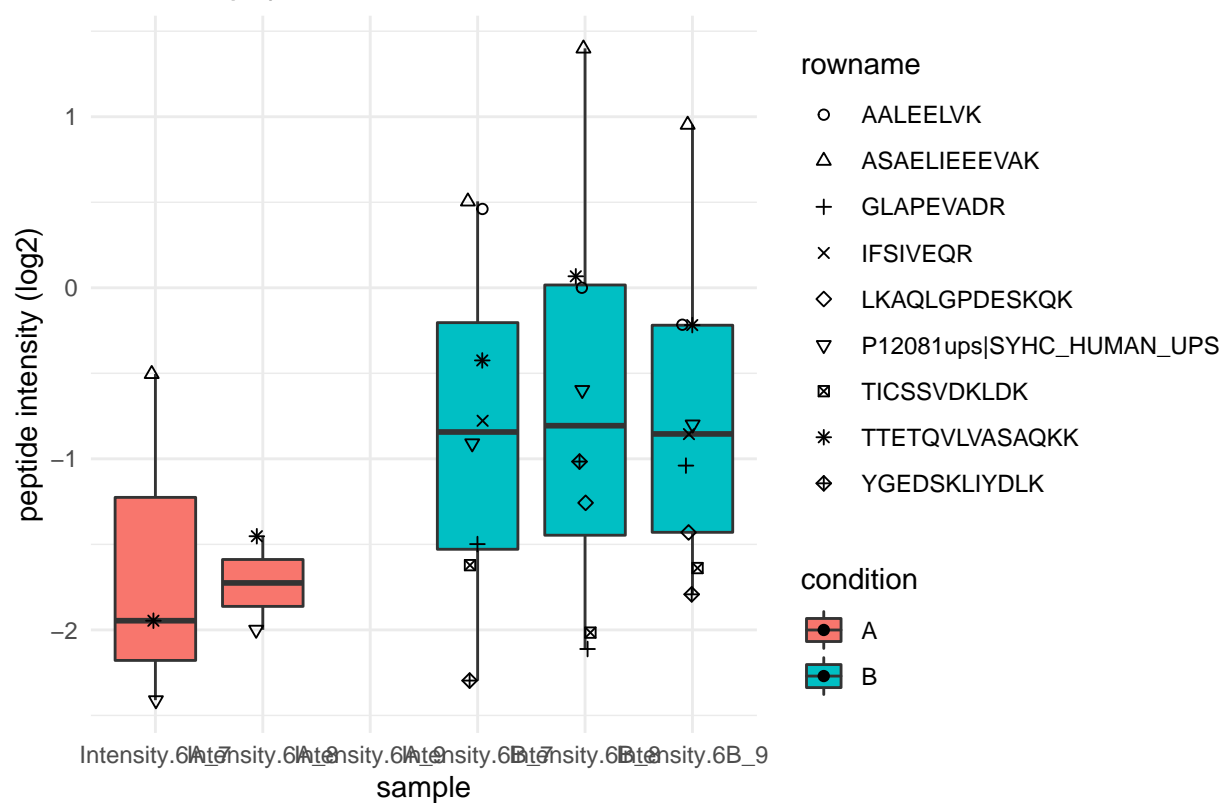
P10636-8ups|TAU_HUMAN_UPS

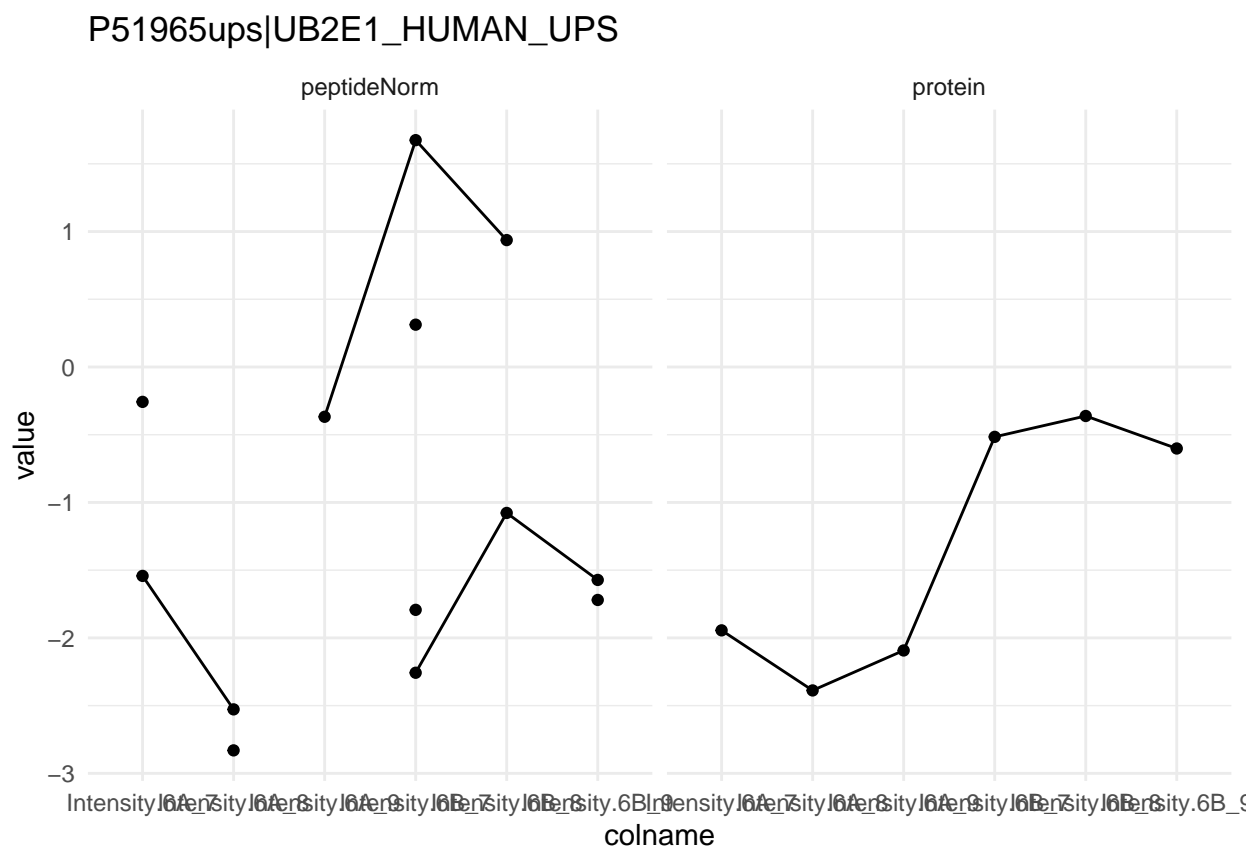




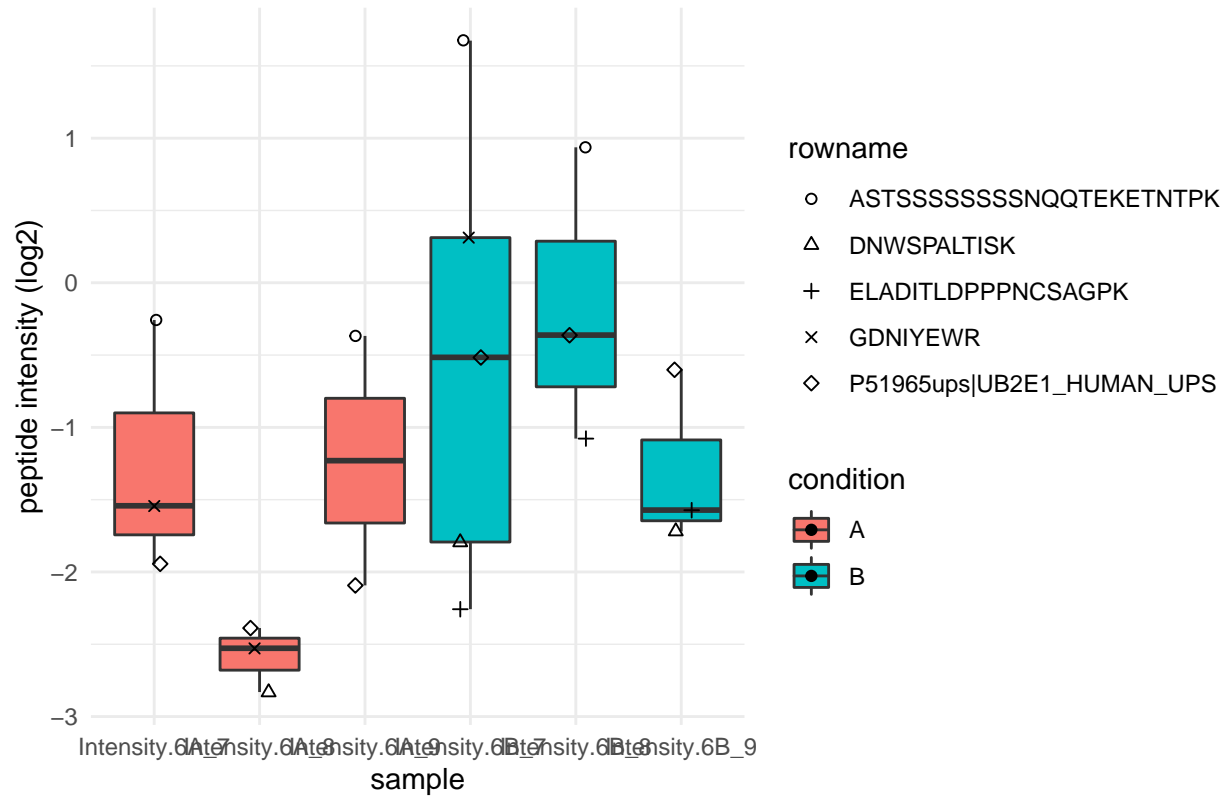


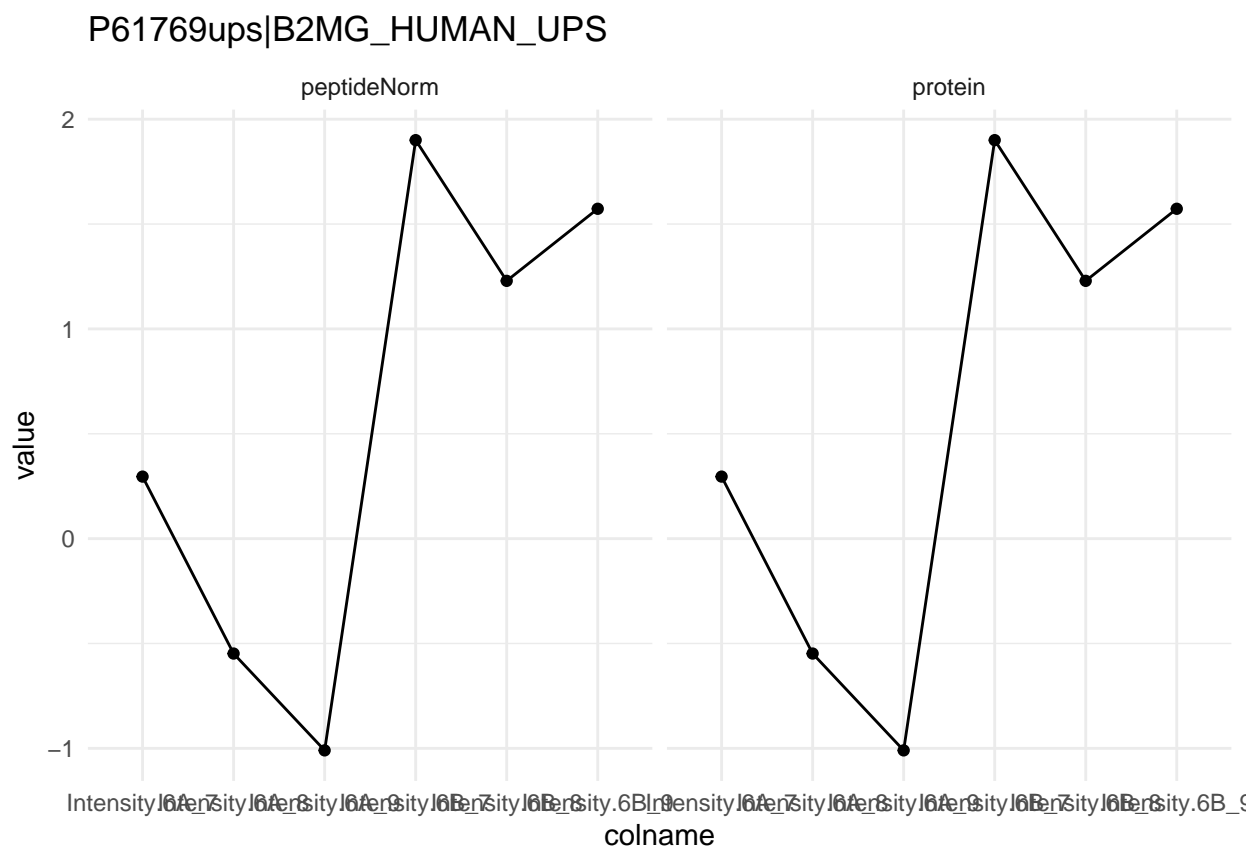
P12081ups|SYHC_HUMAN_UPS

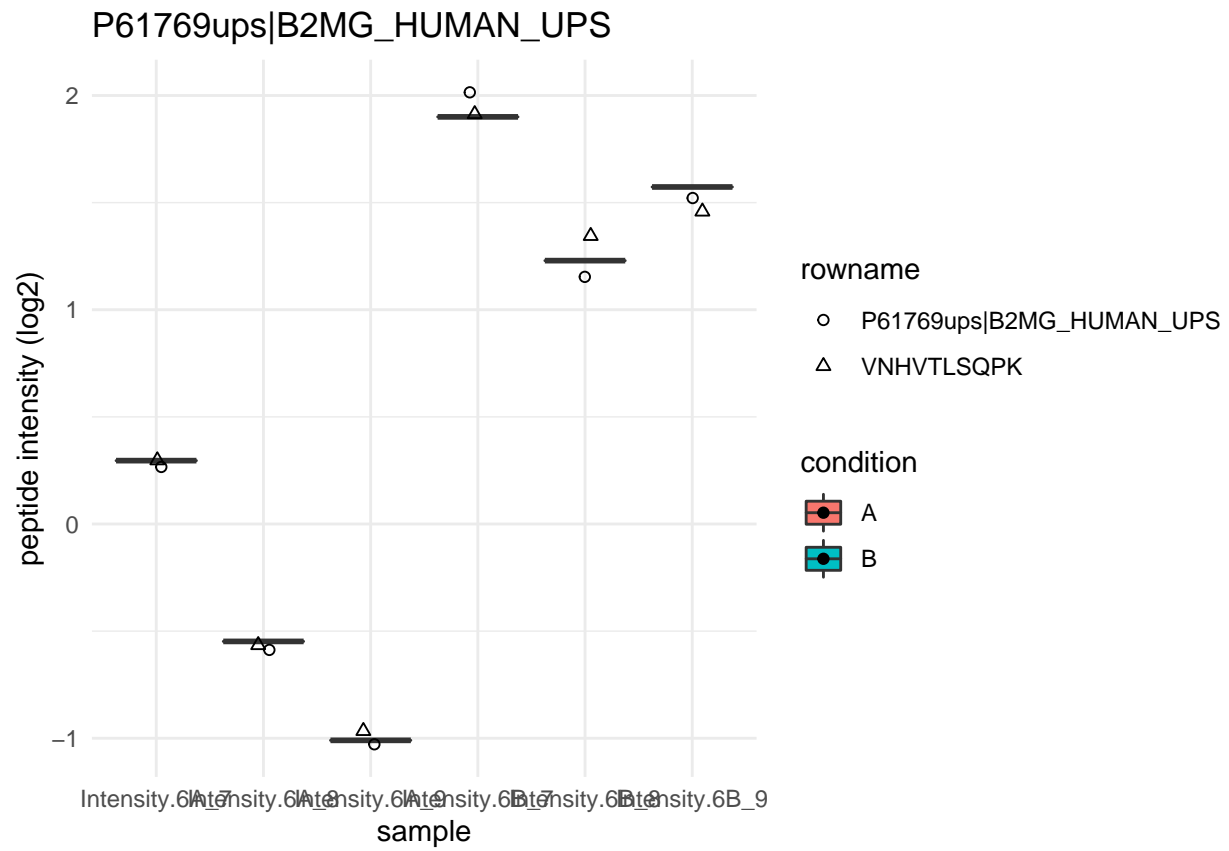




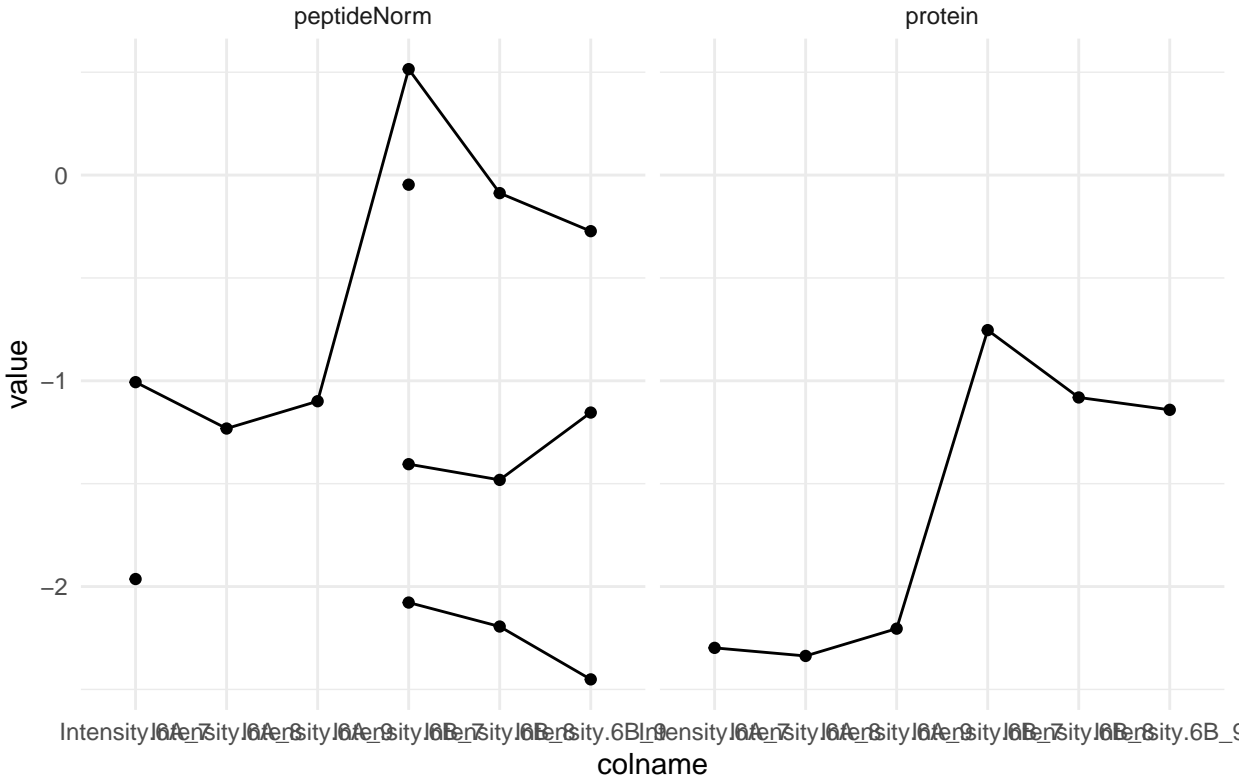
P51965ups|UB2E1_HUMAN_UPS



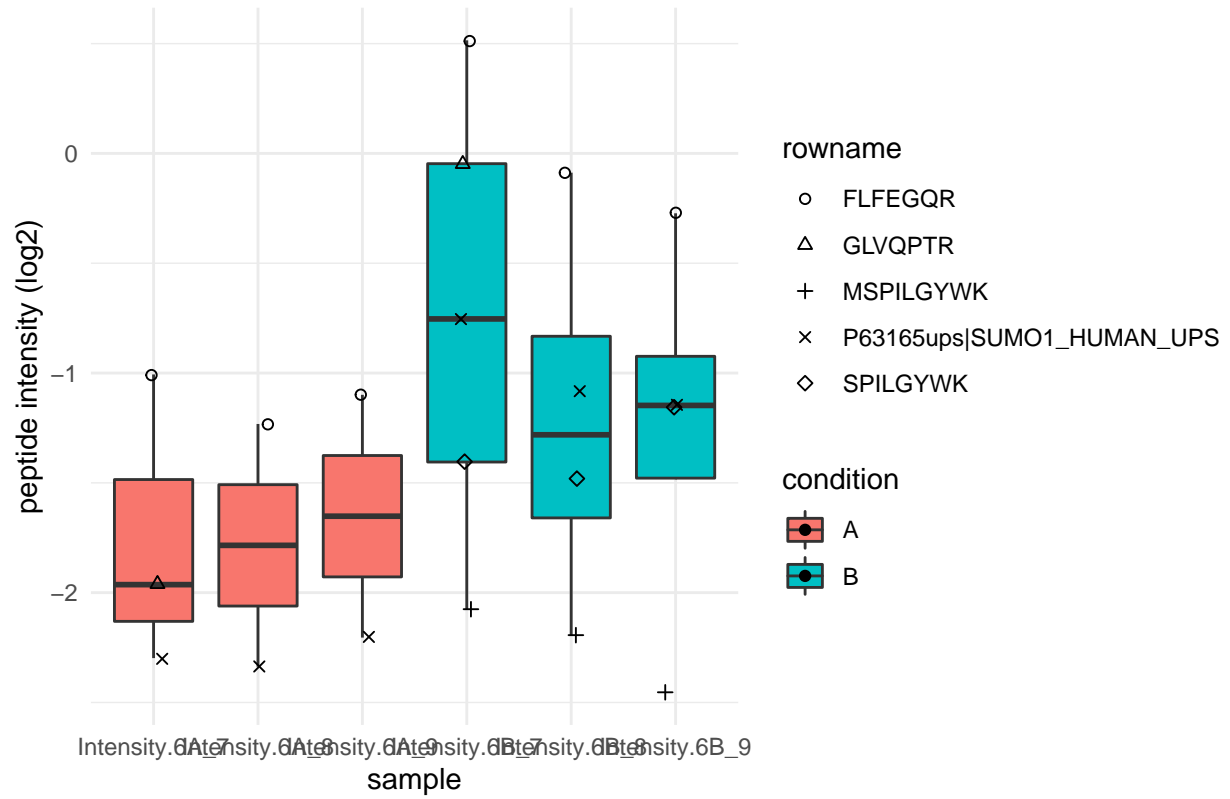




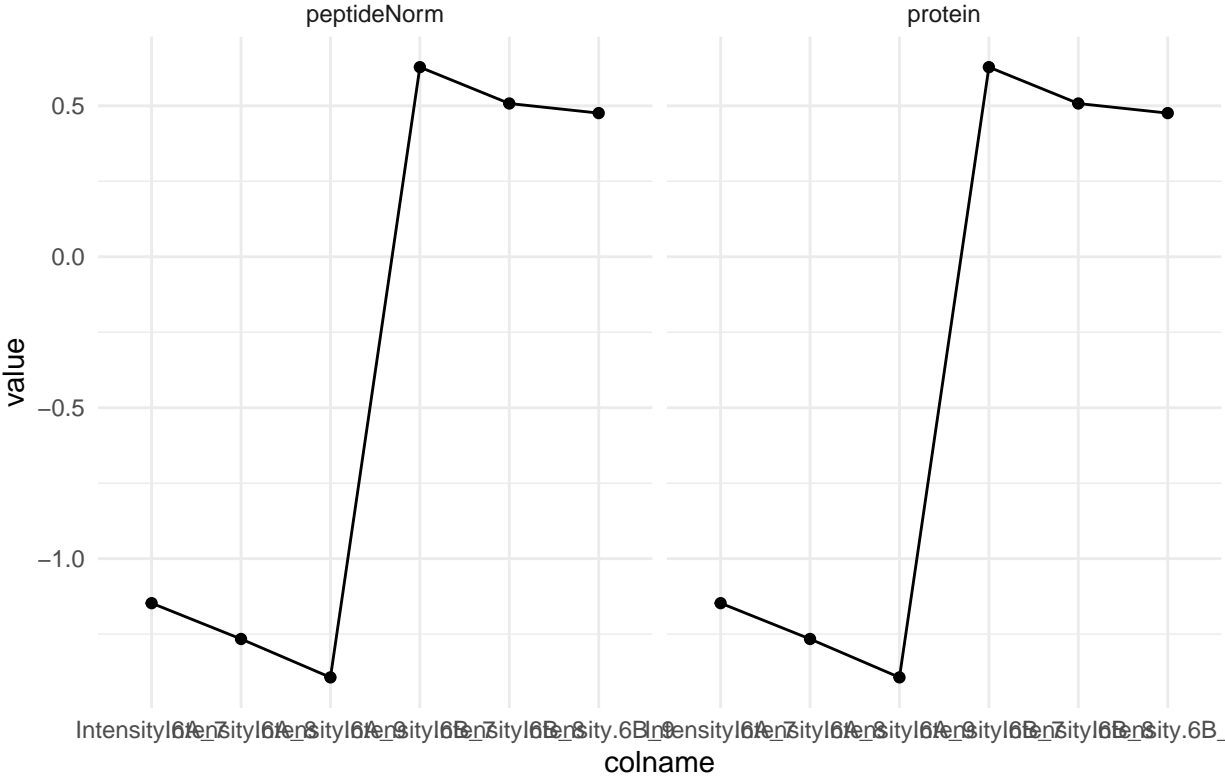
P63165ups|SUMO1_HUMAN_UPS

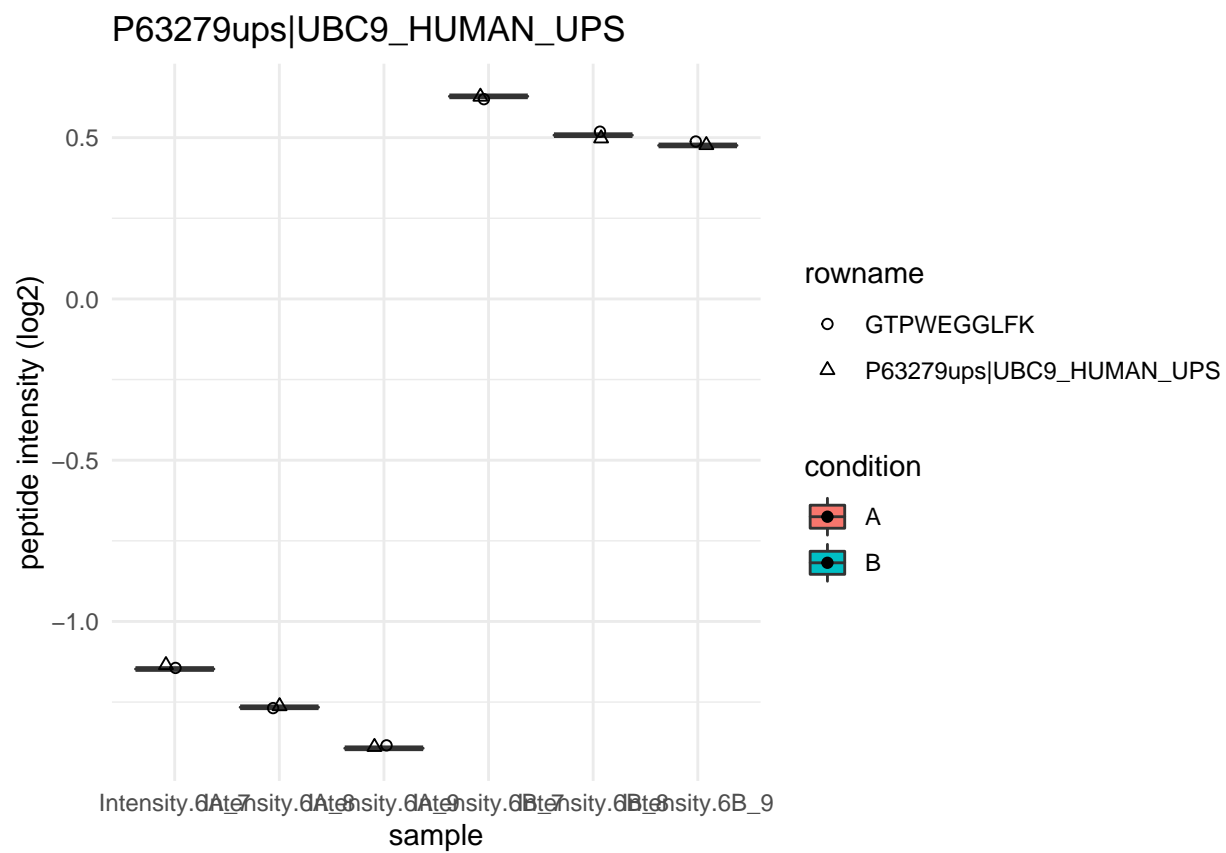


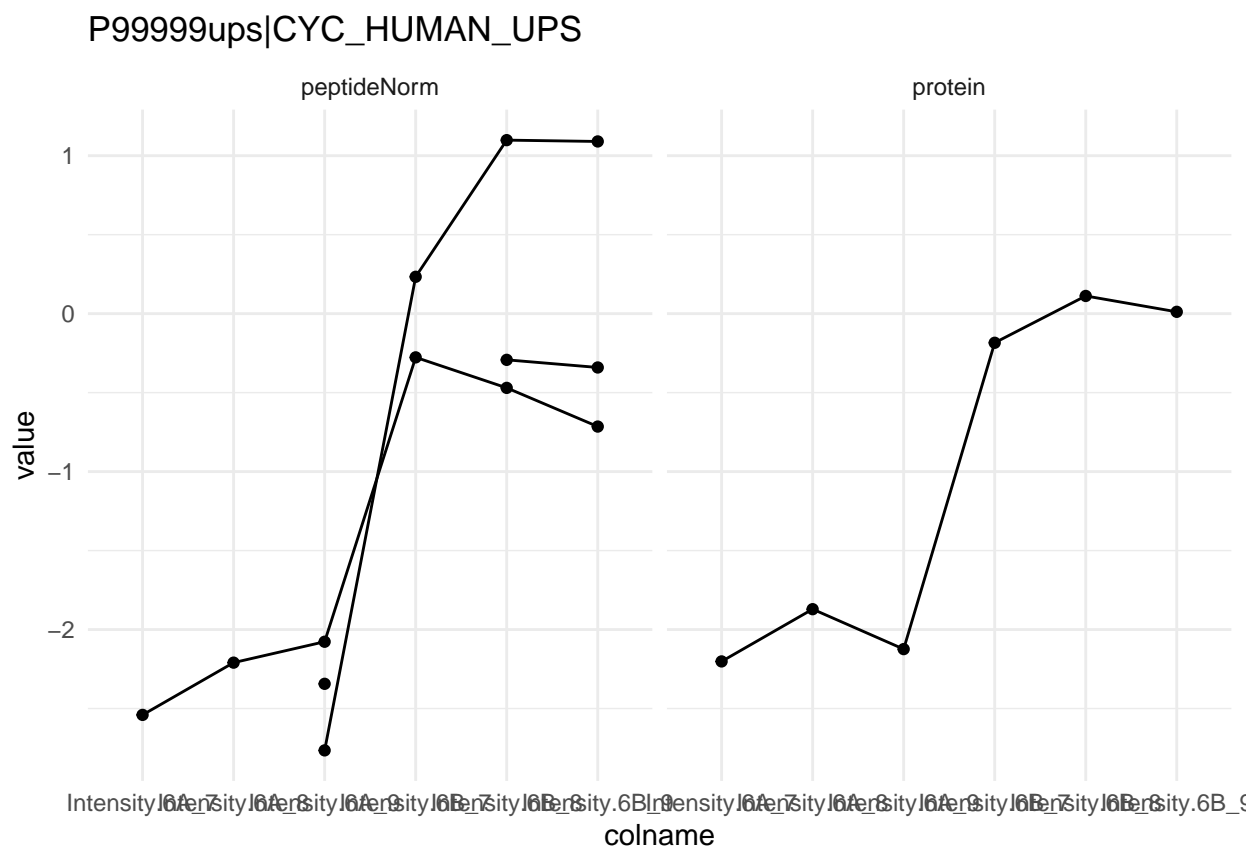
P63165ups|SUMO1_HUMAN_UPS

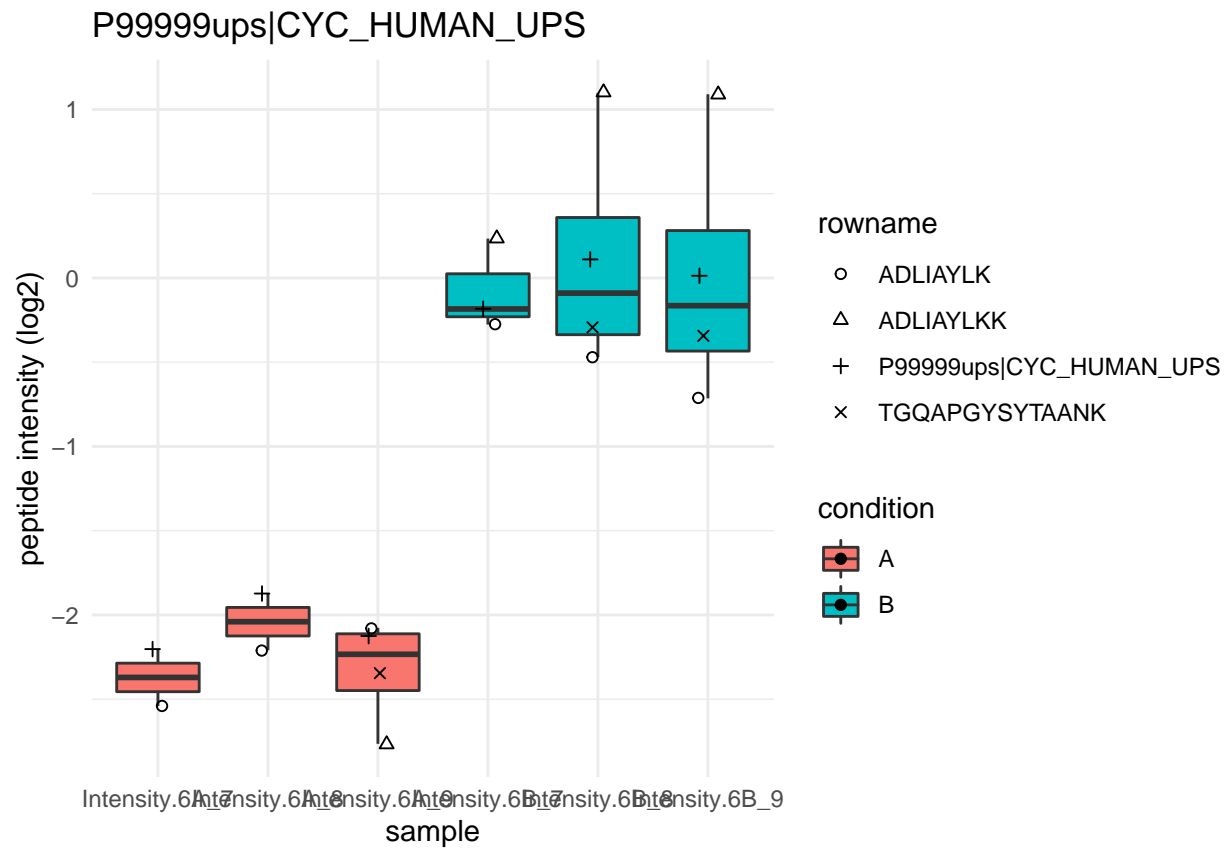


P63279ups|UBC9_HUMAN_UPS

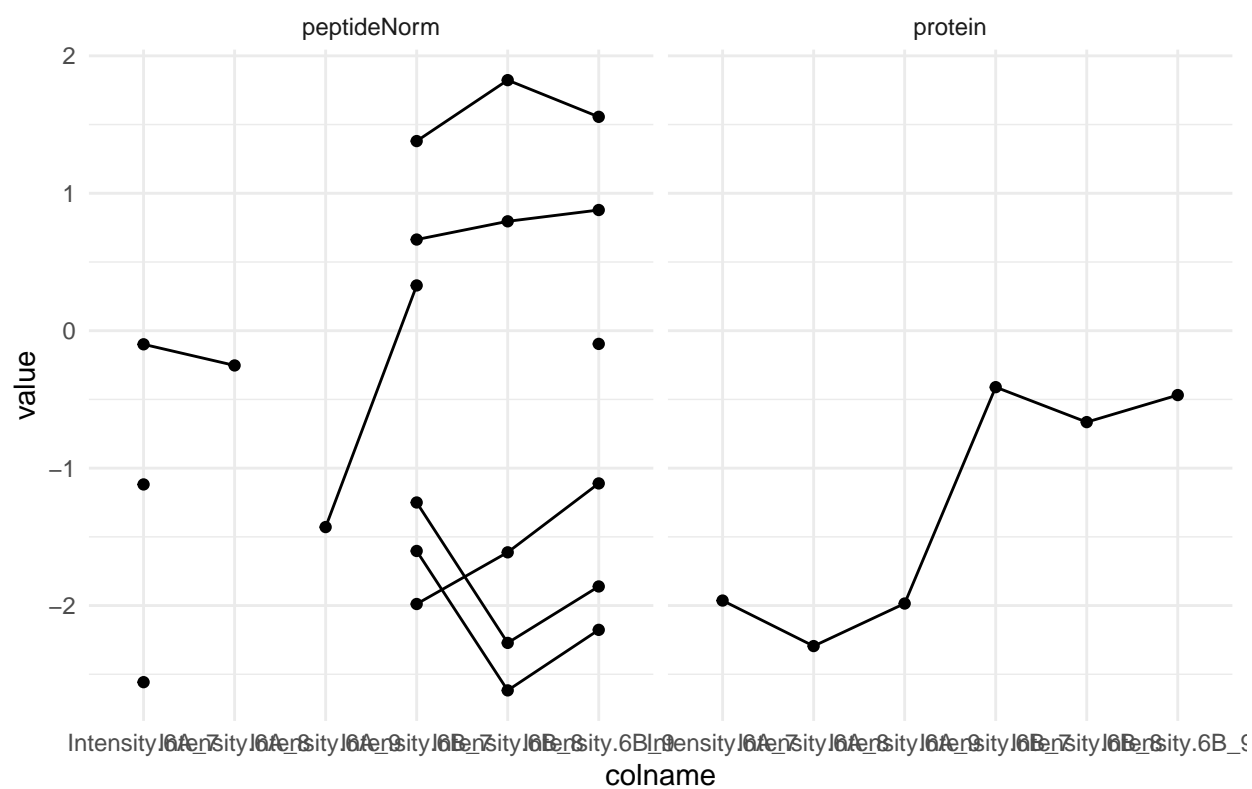


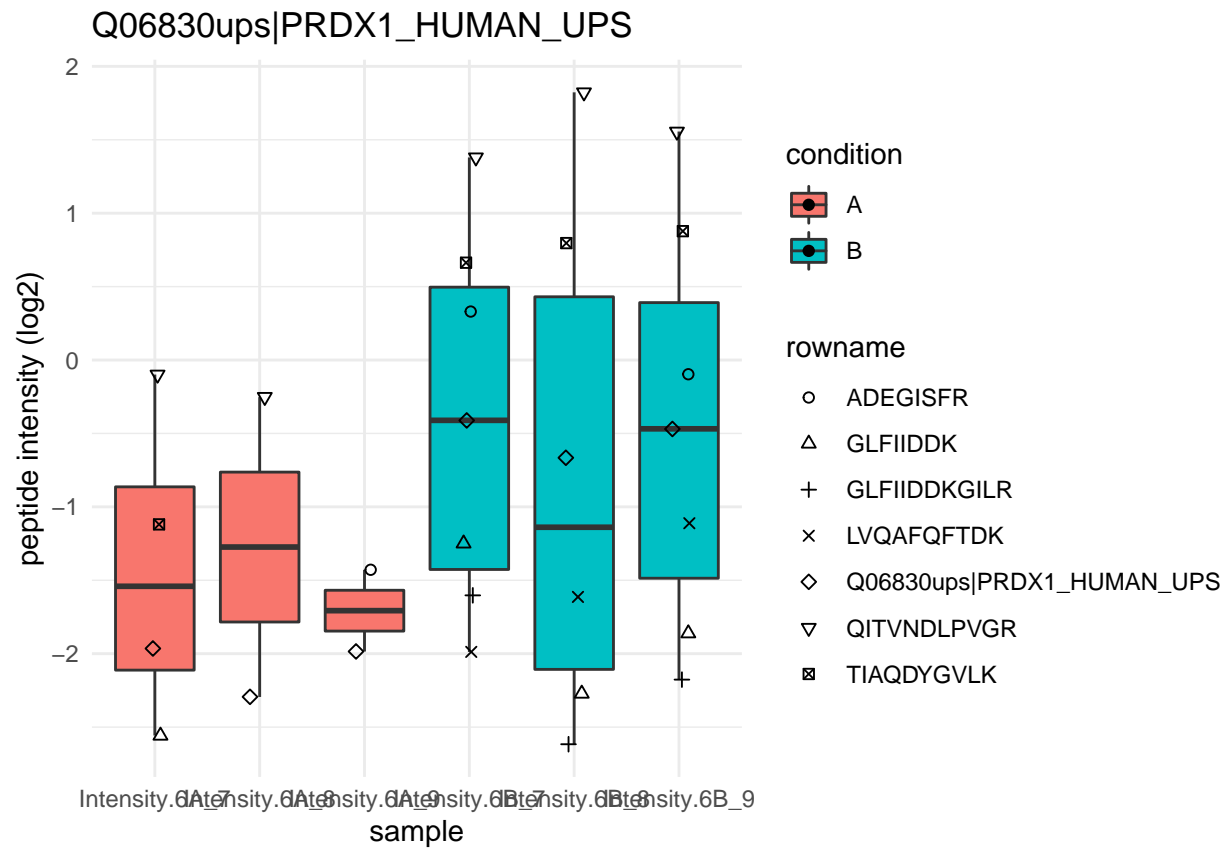


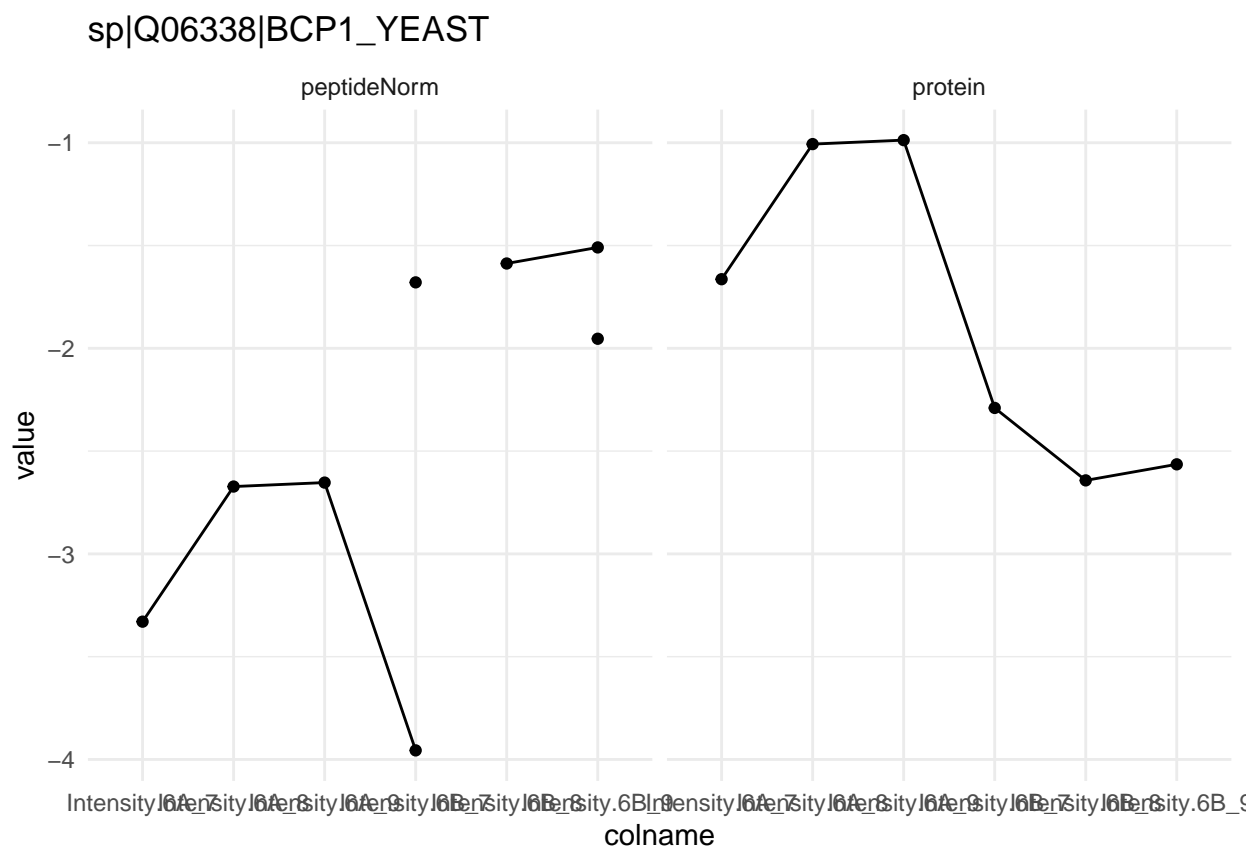


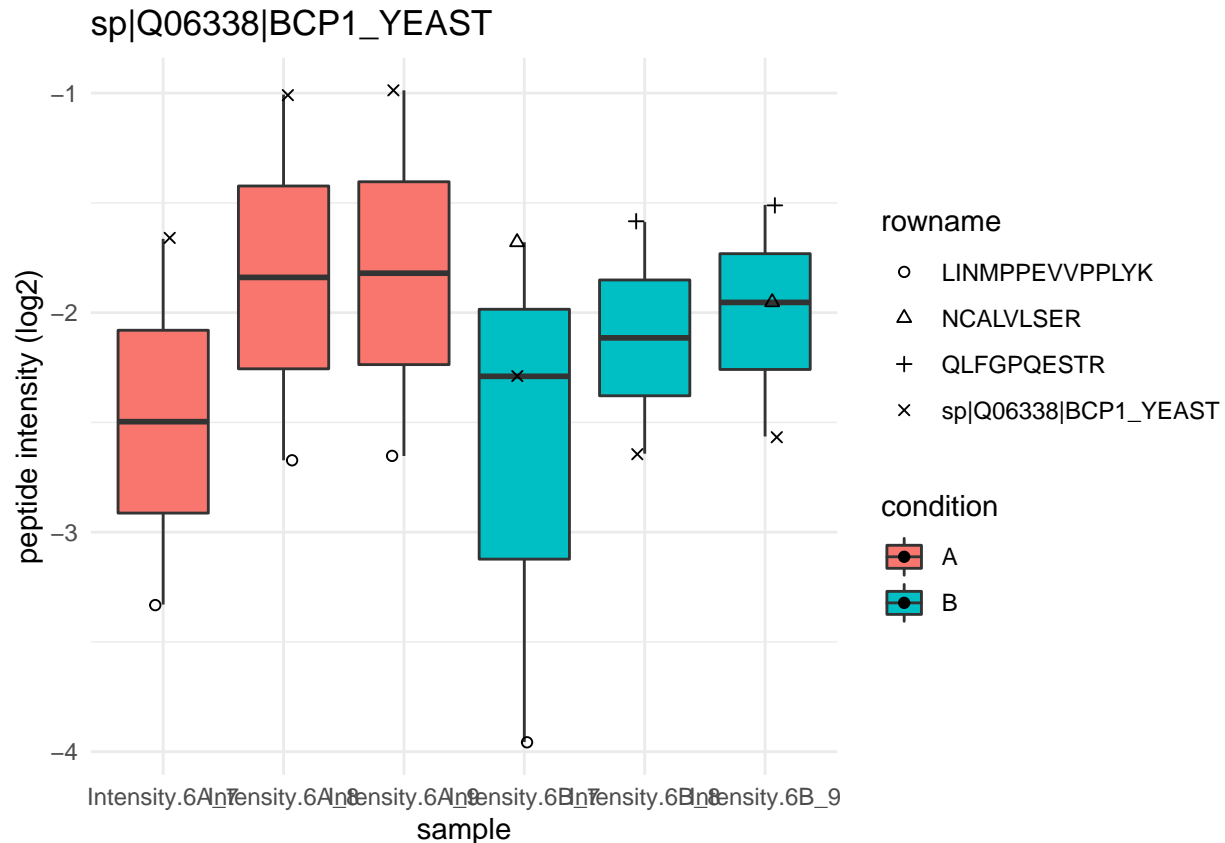


Q06830ups|PRDX1_HUMAN_UPS









Note, that the yeast protein is only covered by 3 peptides. Only one peptide is picked up in condition A. This peptide is also only once observed in spike-in condition B. This puts a considerable burden upon the inference and could be avoided by more stringent filtering.

5 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 4.1.0 (2021-05-18)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Catalina 10.15.7
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRlapack.dylib
##
## locale:
##  [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
##  [1] parallel stats4 stats      graphics grDevices utils      datasets
```

```

## [8] methods      base
##
## other attached packages:
## [1] gridExtra_2.3          plotly_4.9.4.1
## [3] msqrob2_1.0.0          QFeatures_1.2.0
## [5] MultiAssayExperiment_1.18.0 SummarizedExperiment_1.22.0
## [7] Biobase_2.52.0         GenomicRanges_1.44.0
## [9] GenomeInfoDb_1.28.0    IRanges_2.26.0
## [11] S4Vectors_0.30.0       BiocGenerics_0.38.0
## [13] MatrixGenerics_1.4.0   matrixStats_0.59.0
## [15] limma_3.48.1           forcats_0.5.1
## [17] stringr_1.4.0          dplyr_1.0.7
## [19] purrr_0.3.4            readr_1.4.0
## [21] tidyr_1.1.3            tibble_3.1.2
## [23] ggplot2_3.3.5          tidyverse_1.3.1
##
## loaded via a namespace (and not attached):
## [1] nlme_3.1-152           ProtGenerics_1.24.0    bitops_1.0-7
## [4] fs_1.5.0               lubridate_1.7.10      httr_1.4.2
## [7] tools_4.1.0            backports_1.2.1       utf8_1.2.1
## [10] R6_2.5.0               DBI_1.1.1             lazyeval_0.2.2
## [13] colorspace_2.0-2       withr_2.4.2           tidyselect_1.1.1
## [16] compiler_4.1.0         cli_2.5.0             rvest_1.0.0
## [19] xml2_1.3.2             DelayedArray_0.18.0   labeling_0.4.2
## [22] scales_1.1.1           digest_0.6.27         minqa_1.2.4
## [25] rmarkdown_2.9          XVector_0.32.0        pkgconfig_2.0.3
## [28] htmltools_0.5.1.1     lme4_1.1-27.1         highr_0.9
## [31] dbplyr_2.1.1           htmlwidgets_1.5.3     rlang_0.4.11
## [34] readxl_1.3.1           rstudioapi_0.13       farver_2.1.0
## [37] generics_0.1.0         jsonlite_1.7.2        BiocParallel_1.26.0
## [40] RCurl_1.98-1.3         magrittr_2.0.1        GenomeInfoDbData_1.2.6
## [43] Matrix_1.3-3           Rcpp_1.0.6            munsell_0.5.0
## [46] fansi_0.5.0            MsCoreUtils_1.4.0     lifecycle_1.0.0
## [49] stringi_1.6.2          yaml_2.2.1            MASS_7.3-54
## [52] zlibbioc_1.38.0        grid_4.1.0            crayon_1.4.1
## [55] lattice_0.20-44        splines_4.1.0         haven_2.4.1
## [58] hms_1.1.0             knitr_1.33            ps_1.6.0
## [61] pillar_1.6.1           boot_1.3-28           codetools_0.2-18
## [64] reprex_2.0.0           glue_1.4.2            evaluate_0.14
## [67] data.table_1.14.0      modelr_0.1.8          nloptr_1.2.2.2
## [70] vctrs_0.3.8            cellranger_1.1.0      gtable_0.3.0
## [73] clue_0.3-59            assertthat_0.2.1      xfun_0.24
## [76] broom_0.7.8            AnnotationFilter_1.16.0 viridisLite_0.4.0
## [79] cluster_2.1.2          ellipsis_0.3.2

```