Tutorial 5: Statistical analysis of DIA data

1. Introduction

One of the most frequently asked questions is: How to get a protein matrix to answer a particular biological question?

Nowadays there are several nice software tools available that can help to reach this goal.

Two of them are quite advanced:

a) MSstats which can also be used for DDA and SRM analysis
b) mapDIA

In this Tutorial, you will learn how to process your output of the OpenSWATH – pyProphet – TRIC alignment and how to extract biological information. We will focus on MSstats in R. In principle MSstats can also be used within Skyline as a plugin.

MSstats is composed of 4 individual parts:

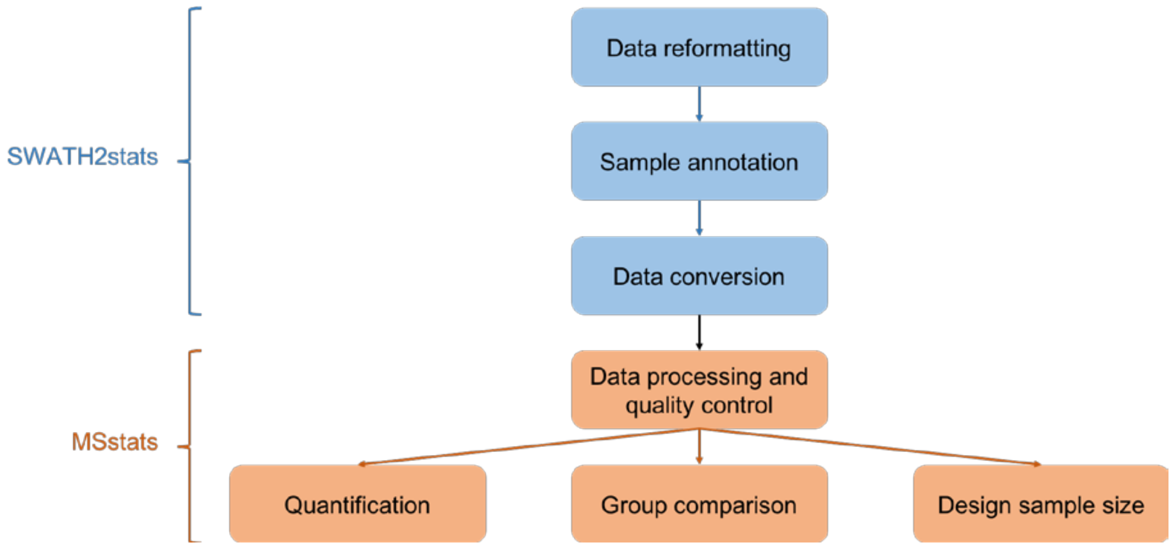
a) Data processing and Quality control
b) Group comparison
c) Protein quantification
d) Design sample size

Tip! For more detailed information about functionality and options, visit **msstats.org**

In order to feed our data into MSstats we need to modify and reformat the output of the TRIC alignment.
This can easily be done with the package SWATH2stats.

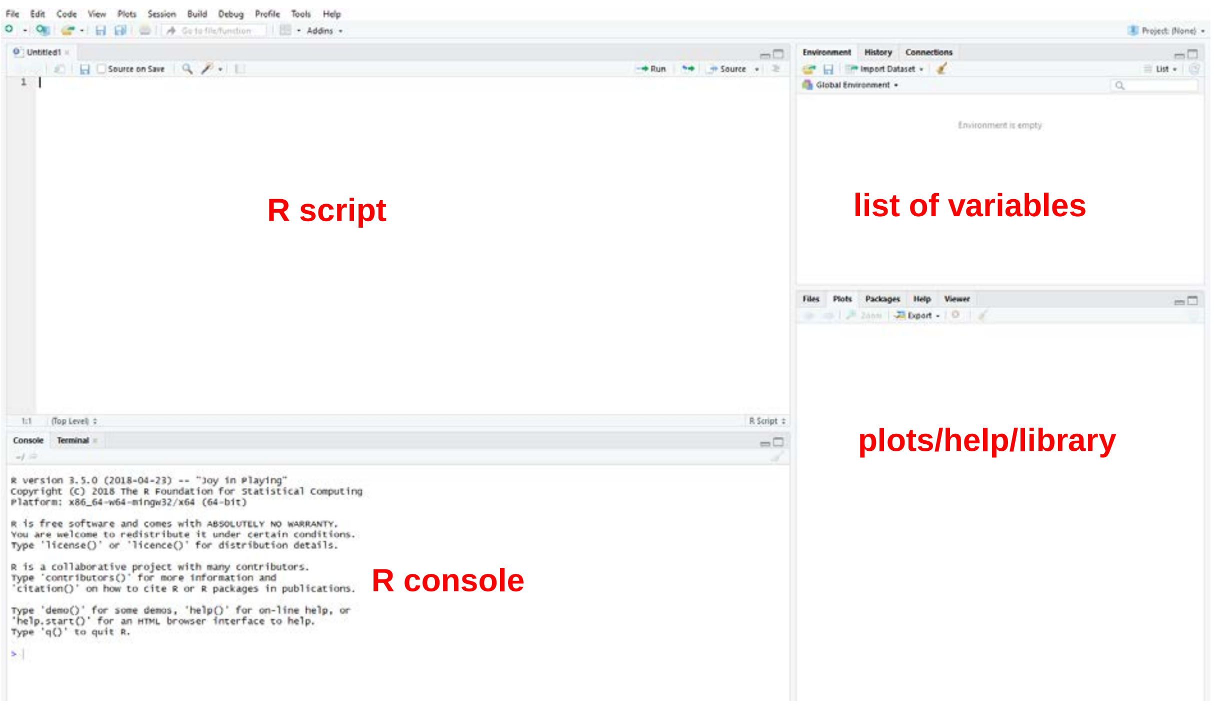
SWATH2stats provides functions to annotate the data with information about biological replicate and condition and easily convert the data into the correct format for various downstream softwares as MSstats or mapDIA. Additionally, it offers multiple ways to filter the data, for example to obtain a specific FDR.

Tip! Please find more detailed information about SWATH2stats at **https://bioconductor.org/packages/release/bioc/html/SWATH2stats.html**



2. Prepare data for MSstats

- Start by opening Rstudio.
- Go to "File" → "New File" → "R Script".
  - **Note!** In the previous Tutorials we have used the Terminal in the lower left corner. Make sure you are using the R console now.



```
R version 3.5.0 (2018-04-23) -- "Joy in Playing"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

- First define the working directory. This will avoid long paths when loading and saving data. Go to "Session" → "Set working directory" → "Choose directory" and search for your folder "Tutorial5_Statistics".
  - **Note!** Alternatively, you can also set the directory directly in the command line by copying this command in your script and then run it:

```
setwd('C:/DIA_Course/Tutorial5_Statistics/')
```

- Now load in the required packages.
  - **Note!** Alternatively, you can also search the respective packages in the lower box on the right side of R studio.

```
library(SWATH2stats)
library(MSstats)
```

- Load in the OpenSWATH output.

```
data <-
   read.delim2(
         'C:/DIA_Course/Tutorial4_OpenSWATH/aligned.tsv',
         dec='.',
         sep='\t',
         header=TRUE,
         stringsAsFactors = FALSE)
```

- **Note!** If you work on a UNIX environment you can use `fread()` from the package `data.table` which reads in the data similarly to `read.delim2`, but faster. The class of the data will be `data.table` instead of a `data.frame`. Therefore, you need to convert it into the correct class.
  `data <- as.data.frame(data)`
- `sep` defines the separator between columns, and `dec` the decimal separator. The first line in the OpenSWATH output contains column names, therefore `header=TRUE`. `read.delim2` has the disadvantage to transform columns into factors, that are hard to work with. Therefore, we set `stringsAsFactors = FALSE` to avoid this.
- Check the help of `read.delim2()` for more information by typing `?read.delim2`
- **Tip!** Just start typing the beginning of the file name and then press the tab key and the name will be completed automatically. This will help to avoid typos.

• Due to the new format of our data, we need to adjust some column names in order to be recognized in the following procedure.

```
names(data)[names(data) == "FullUniModPeptideName"] <-
   "FullPeptideName"
names(data)[names(data) == "aggr_fragment_annotation"] <-
   "aggr_Fragment_Annotation"
names(data)[names(data) == "aggr_peak_area"] <-
   "aggr_Peak_Area"
```

• Let's check the dimensions of our data.

```
dim(data)
```

- **Note!** R always prints number of rows first and then number of columns.
- **Tip!** Click on the variable "data" in the top-right window of R studio, to have a glance in the data. A window will appear above your R console. Explore the data structure and make yourself familiar with the numerous columns. As you can see on the R console on the bottom left, the text "`View(data)`" appeared. This means that you can also type "`View(data)`" to open the viewing panel.
- **Note!** Many columns contain variables from the model built in OpenSWATH. We do not need most of them in our further statistical analysis.

• Before we apply this filter on the actual data, we reduce the number of columns to the ones we really need for the further analysis.

```
data.reduced <- reduce_OpenSWATH_output(data)

View(data.reduced)
```

- **Tip!** Check how the data looks now by typing `View(data.reduced)` or by clicking on "data.reduced" in the top right Environment panel.

• We also want to get rid of iRT peptides that are still in the data, we only needed them for RT calibration (as you hopefully remember ☺). Additionally we will filter out all non-proteotypic peptides.

```
data.reduced <-
   data.reduced[grep(
         "iRT", data.reduced$ProteinName,
         invert = TRUE),]

data.reduced <-
   data.reduced[grep(
         ";", data.reduced$ProteinName,
         invert = TRUE),]
```

- **Note!** The "`grep`" command searches the expression "`iRT`" in the column `data.reduced$ProteinName` and "`invert=TRUE`" means that the hits are removed from the table.

- We also need to annotate the data with additional information. In your folder "Tutorial5_Statistics" you can find a file called "DIA_course_annotation.txt" Open this with Excel and investigate what information you can find there.
- Then load in the annotation file into R studio.
  - **Note!** **Remember to change the filename if you have used the QE data**.

```
annotation.file <-
   read.delim(
         file = 'DIA_Course_anotation_TTOF.txt',
         sep='\t',
         header=TRUE)
```

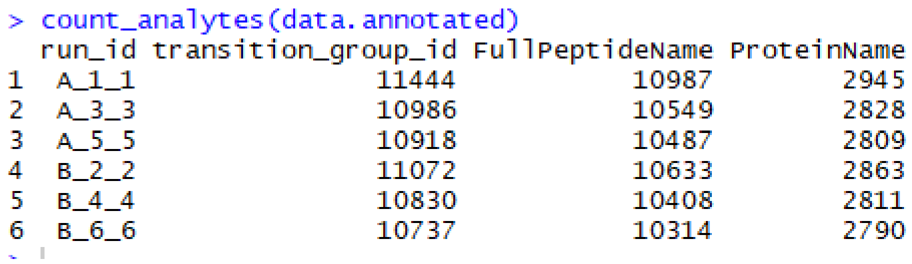
- **Tip!** Check if the data was correctly loaded by clicking on "annotation.file" in the data overview again.

- Now annotate the data:

```
data.annotated <-
   sample_annotation(data.reduced, annotation.file)
```

- In SWATH2stats we can also have a quick look for the number of proteins and peptides in each sample.

```
count_analytes(data.annotated)
```



```
> count_analytes(data.annotated)
  run_id transition_group_id FullPeptideName ProteinName
1  A_1_1               11444           10987        2945
2  A_3_3               10986           10549        2828
3  A_5_5               10918           10487        2809
4  B_2_2               11072           10633        2863
5  B_4_4               10830           10408        2811
6  B_6_6               10737           10314        2790
```

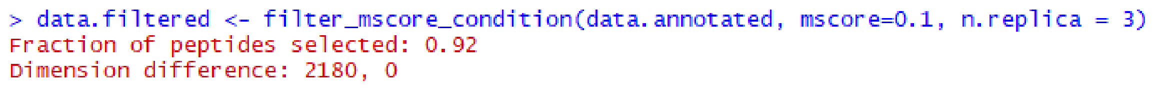
We can see that there are slight differences between the samples, but nothing worrying. If you would see here a sample that is showing far less proteins than other samples, you might consider not using this sample for your analysis.

- **Caution!** The screenshots you see are representing the results using the output that was generated with the DDA library (Tutorial 1). If you used the DIA-UMPIRE library you will have different numbers.

- We can now filter our data set. We want to make sure that we just use complete observations.

```
data.filtered <-
   filter_mscore_condition(
         data.annotated,
         mscore=0.01,
         n.replica=3)
```

- o **Note!** This function filters the data based on completeness of the measured transition groups among samples. We use all 6 samples (1 = 100%) here that need to pass the mscore of 0.01. The mscore corresponds to the q-value, which means that filtering for 0.01 results in a table with 1% FDR at the transition level.

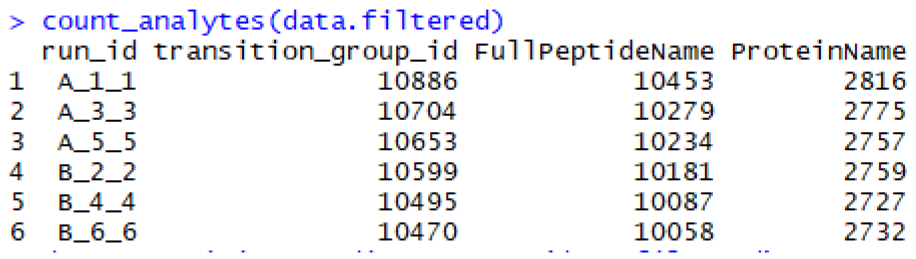


```
> data.filtered <- filter_mscore_condition(data.annotated, mscore=0.1, n.replica = 3)
Fraction of peptides selected: 0.92
Dimension difference: 2180, 0
```

R tells you here again, that it is keeping 92 % of the data. In total it deletes 2180 transition groups.

- Again we should check how the number of peptides/proteins changed after filtering.

```
count_analytes(data.filtered)
```



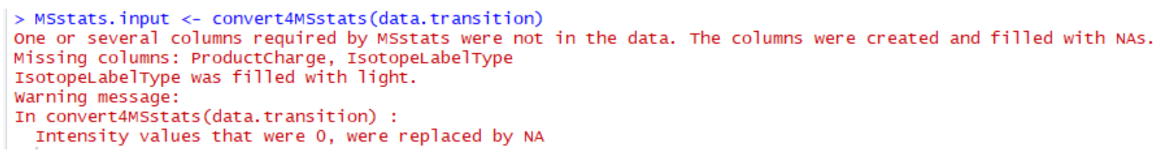
```
> count_analytes(data.filtered)
  run_id transition_group_id FullPeptideName ProteinName
1  A_1_1               10886           10453        2816
2  A_3_3               10704           10279        2775
3  A_5_5               10653           10234        2757
4  B_2_2               10599           10181        2759
5  B_4_4               10495           10087        2727
6  B_6_6               10470           10058        2732
```

- To feed the data into MSstats (or mapDIA) we need to split the transition groups into single transitions.

```
data.transition <- disaggregate(data.filtered)
```

- In the last step the columns are renamed to match the requirements for MSstats.

```
MSstats.input <- convert4MSstats(data.transition)
```



```
> MSstats.input <- convert4MSstats(data.transition)
One or several columns required by MSstats were not in the data. The columns were created and filled with NAs.
Missing columns: ProductCharge, IsotopeLabelType
IsotopeLabelType was filled with light.
Warning message:
In convert4MSstats(data.transition) :
  Intensity values that were 0, were replaced by NA
```

- o **Note!** Don't worry about the warning messages at this stage.

- Save the produced MSstats input and the intermediate data.

```
save(data.annotated, data.reduced, data.filtered, data.transition,
     file="SWATH2stats_workflow_TTOF.Rdata")

save(MSstats.input, file="MSstats_input_TTOF.Rdata")
```

3. MSstats analysis

3.1. Data processing and Quality control

- First run the processing function to log2-transform. We do not normalize the data because we expect varying abundances for the proteins, peptides and transitions of E.coli and yeast. We will later check for equal intensity distributions and whether another type of normalization is necessary in the QCplots.

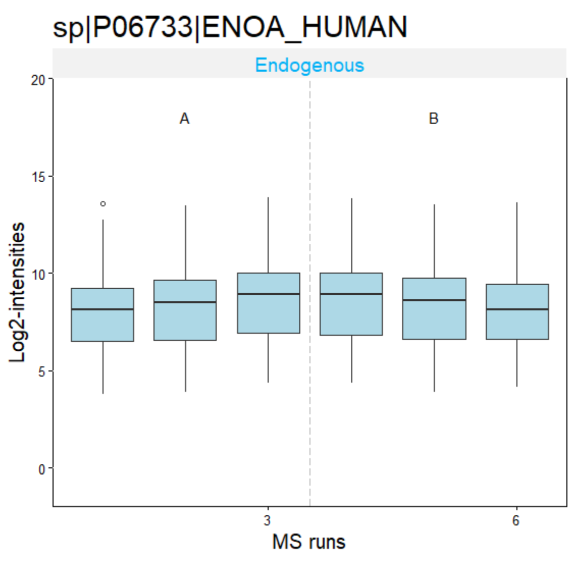
```
data.processed <- dataProcess(MSstats.input,
                  normalization = FALSE,
                  summaryMethod = "TMP", censoredInt = "NA",
                  cutoffCensored = "minFeature", MBimpute = FALSE)
```

  - The summary method `"TMP"` stands for Tukey's median polish which is a robust parameter estimation method with median across rows and columns. `censoredInt = "NA"` assumes that missing values are censored, meaning below the limit of detection. Because `MBimpute` is false, missing values will get the minimum value for each feature.
  - **Note!** The result of this function will be a large list with several variables from which `data.processed$ProcessedData` contains our processed data set.

- Now we want to check for potential quality issues. Therefore, we will make a quality control plot.

```
dataProcessPlots(
   data.processed,
   type = "QCPlot",
   legend.size = 4,
   which.Protein = "sp|P06733|ENOA_HUMAN",
   address = FALSE)
```

  - **Note!** As we have data for thousands of proteins, plotting all of them would exceed the time we have in this tutorial. Therefore, the option `which.Protein` defines which proteins should be plotted with either protein number or protein name. In this and the following plots we focus making the plot for 1 random example protein. With the option `address = FALSE`, plots will be shown in the graphical panel, but not saved in a file.



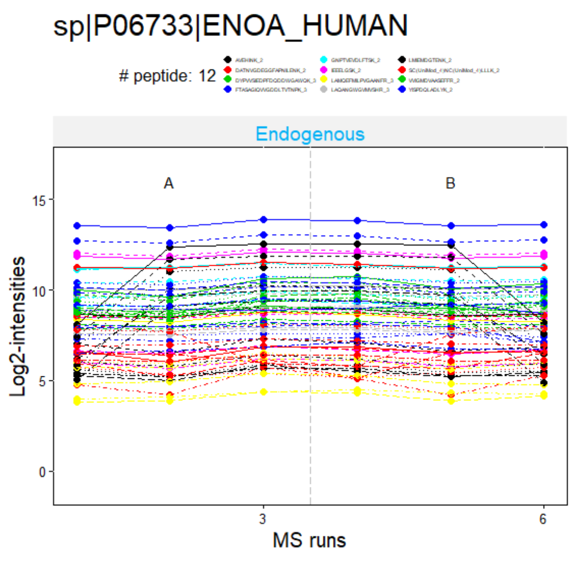
As you can see the distributions of the log2 protein intensities in the different samples are very similar. For the example human protein this is also the case. We can go ahead with our analysis.

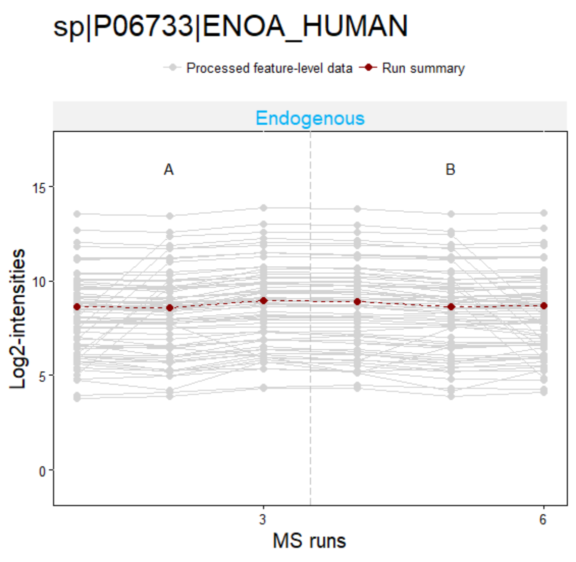
- **Question:** What do you expect this plot to look like for a yeast or E.coli protein?
    - **Note!** In RStudio you can see the plots in the lower right corner. Using the arrows at the top left you can navigate back and forth to look at the generated plots. If you want to save a plot, click on the drop-down Export button.

- Additionally, we can also plot the peptide profiles.

```
dataProcessPlots(
   data.processed,
   type = "ProfilePlot",
   featureName = "Peptide",
   legend.size = 4,
   originalPlot= TRUE,
   summaryPlot = TRUE,
   which.Protein = "sp|P06733|ENOA_HUMAN",
   address = FALSE)
```

In the first profile plot you can see the transitions of several peptides (black, red and green) of the human example protein (you will need to click the back arrow in the plotting window to see this). The figure on the right shows the profile plot where the dotted red line indicates the protein summarization.

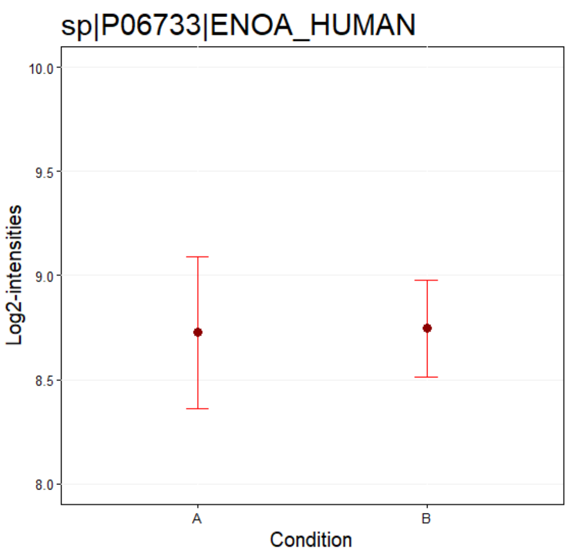




- The last plot summarizes mean of log2-intensities with confidence interval for each condition per protein

```
dataProcessPlots(
   data.processed,
   type = "Conditionplot",
   which.Protein = "sp|P06733|ENOA_HUMAN",
   address = FALSE)
```

Here you can see the condition plot of our human example protein. Does this look as you would expect?



- Select other proteins or a few random numbers and check how the different plots you generated above look for different proteins from different organisms. For this, you have to exchange `which.Protein = "sp|P06733|ENOA_HUMAN"` by any other name e.g. `which.Protein = "tr|C8ZBP1|C8ZBP1_YEAS8"` or **`which.Protein = "sp|P29745|PEPT_ECOLI"` for TTOF or `"sp|P25738|MSYB_ECOLI"` for QE**

- **Questions:** Can you already identify trends? Do you see what you would expect from our sample?

- As usual, save the processed data:

```
save(data.processed, file="MSstats_processed_TTOF.Rdata")
```

3.2. Group Comparison

To perform a group comparison, we first need to define what should compared.
We would like to compare Mix A with Mix B. For each of these groups we have 3 samples.

- In order to define the comparison matrix in the correct order we will check the order of the group levels, which R automatically assigns alphanumerically.

```
levels(data.processed$ProcessedData$GROUP_ORIGINAL)
```

- Groups are A and B and we would like to calculate the changes in B vs. A
    - **Caution!** If you have more than 2 groups you will need to design the comparison matrix differently. For this check the manual at **msstats.org**

```
comparison <- matrix(c(-1, 1), nrow=1)

rownames(comparison) <- "B vs. A"
```

- Perform the group comparison.

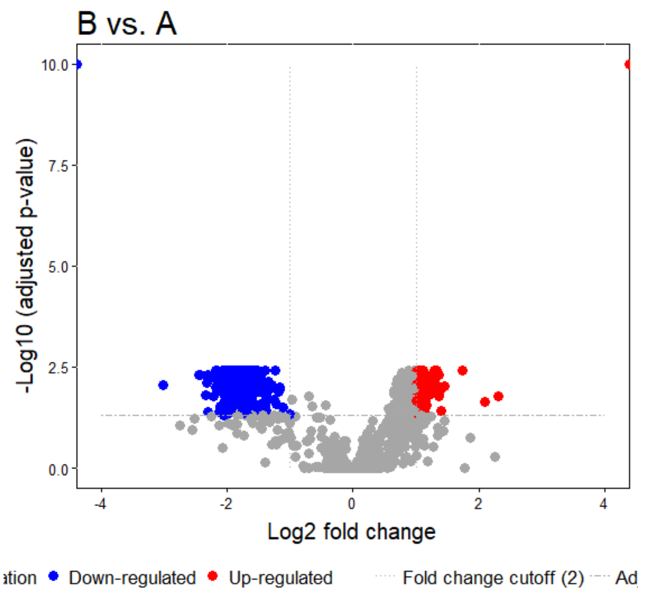
```
result.GroupComparison <- groupComparison(
                               contrast.matrix = comparison,
                               data = data.processed)
```

  - The result is a list consisting of several variables. To check the group comparison outcome, we need `result.GroupComparison$ComparisonResult`

- Now we plot our group comparison result. MSstats also offers an easy solution for this.

```
groupComparisonPlots(result.GroupComparison$ComparisonResult,
   type = "VolcanoPlot",
   sig = 0.05,
   FCcutoff = 2,
   ProteinName = FALSE,
   address = FALSE)
```

  - **Note!** With `sig` we define the significance level and with `FCcutoff` the desired fold change.

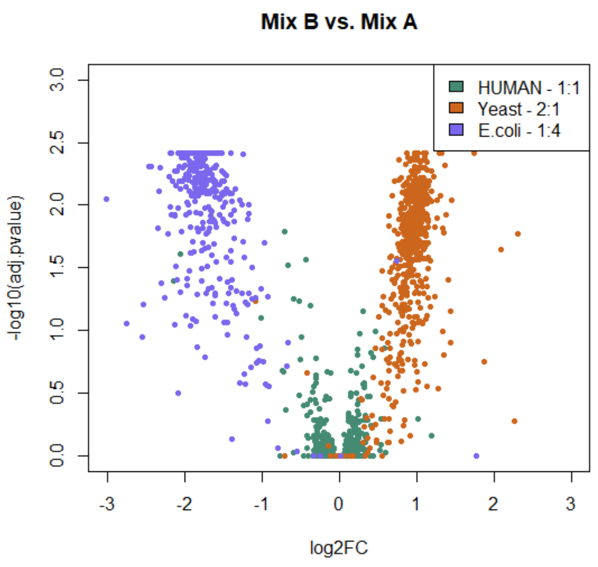


Question! Can you distinguish the different species? Does the significance level and fold change cutoff make sense in our case?

- To make the biology behind our data clearer we will make another Volcano plot with proteins colored by species.

```
with( result.GroupComparison$ComparisonResult,
      plot( log2FC,
            -log10(adj.pvalue),
            pch=20,
            main="Mix B vs. Mix A",
            xlim=c(-3, 3),
            ylim=c(0,3)))
with( subset(result.GroupComparison$ComparisonResult,
      grepl("HUMAN", Protein)),
      points(log2FC,
            -log10(adj.pvalue),
            pch=20,
            col="aquamarine4"))
with( subset(result.GroupComparison$ComparisonResult,
      grepl("YEAS8", Protein)),
      points(log2FC,
            -log10(adj.pvalue),
            pch=20,
            col="chocolate3"))
with( subset(result.GroupComparison$ComparisonResult,
      grepl("ECOLI", Protein)),
      points(log2FC,
            -log10(adj.pvalue),
            pch=20,
            col="slateblue2"))
legend("topright",
      legend = c("HUMAN – 1:1", "Yeast – 2:1", "E.coli – 1:4"),
      fill = c("aquamarine4", "chocolate3", "slateblue2"),
      col = NULL)
```

  - **Note!** Don't worry if you don't fully understand how this second volcano plot was created – this is only to make you see the nice difference between the organisms and is specific to our exemplary dataset ☺



- You can always go back to the plots we produced earlier, by clicking the little arrow above the plot in the “Plots” tab. If you want to save a plot, click “Export” in the “Plots” tab and save the plot in your favorite format.
- If you would like to further work on the groupComparsion result outside of R, you can also export the table:

```
write.table(result.GroupComparison$ComparisonResult,
          file = "groupComparison_result_TTOF.tsv",
          quote = FALSE,
          row.names = FALSE,
          sep = "\t")
```

  - **Note!** `quote = FALSE` will avoid that all entries will be written in quotation marks and `row.names = FALSE` avoids having the rowname-numbers being written in our file.

3.3. Quantification

In order to perform other downstream analysis, it is useful to have a matrix of protein intensities for each sample.

- Perform a protein quantification.

```
quantification.result <-
   quantification(data.processed,
         type = "Sample",
         format = "matrix")
```

- Again we can write the table into a file to use it outside the R environment.

```
write.table(quantification.result,
   file = "quantification_result_TTOF.tsv",
   quote = FALSE,
   row.names = FALSE,
   sep = "\t")
```

With this data, you can now perform other downstream analyses, like classification or regression analyses. In our tutorial, this would exceed the time we have and would neither fit the specific needs you will have with your own data.

3.4. Design sample size

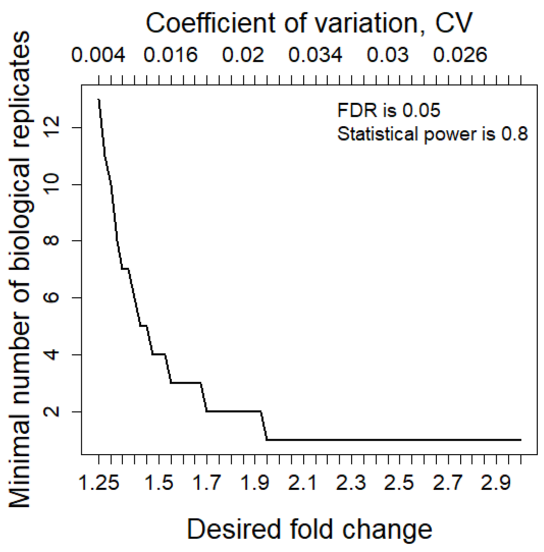
For future experiments it might be useful to estimate the required samples or to predict the power a specific analysis will have if you are restricted in the number of replicates you use.

- First perform an anlysis of the required sample size if you wish for a specific power and fold change of your group comparison.

```
result.sample <-
      designSampleSize(result.GroupComparison$fittedmodel,
            numSamples = TRUE,
            desiredFC = c(1.25, 3),
            FDR = 0.05,
            power = 0.8)
```

- Now visualize the result in a plot:

```
designSampleSizePlots(result.sample)
```

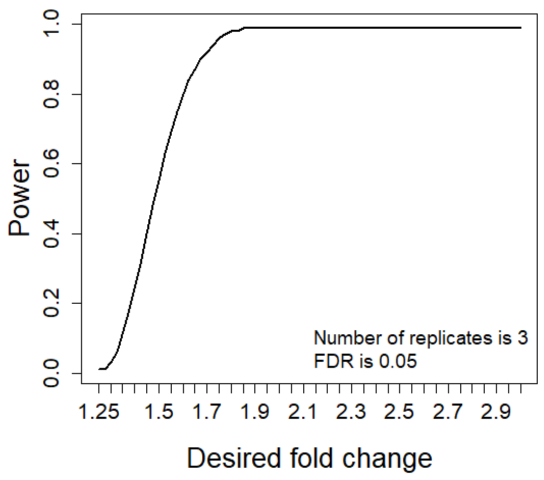


- Now do the same analysis the other way around and ask how much power your analysis will have for a given sample size:

```
result.power <-
     designSampleSize(result.GroupComparison$fittedmodel,
          numSamples = 3,
          desiredFC = c(1.25, 3),
          FDR = 0.05,
          power = TRUE)
```

- Check again the result in a plot:

```
designSampleSizePlots(result.power)
```



We are now finished with the statistical data analysis of our 6 SWATH runs. As you could see, we were able to very well separate the proteins coming from the different species.

Well done! ☺
You managed to get through all of the tutorials. We hope that you learned how to build a library based on DDA and DIA data, perform an OpenSWATH analyses, followed by pyProphet scoring and TRIC alignment and finally how to use this output to extract biological information.
We also hope you can apply the new knowledge now on your own data and wish you a lot of fun with this.

We would like to thank SystemsX for supporting the Zurich DIA / SWATH Course 2018.

SystemsX.ch
The Swiss Initiative in Systems Biology