Basic concepts for differential abundance analysis of proteomics data

Lieven Clement

Largely adapted from chapter "Statistical analysis with msqrob2" in msqrob2book (https://statomics.github.io/msqrob2book/) by Christophe Vanderaa, Stijn Vandenbulcke and Lieven Clement.

Note, that learners who are following the course by using the interactive msqrob2gui APPs can largely ignore the code in this document. The code is here for their reference so that they could progress their coding skills in the future so as to exploit the full power of msqrob2 to develop and automate reproducible data analysis workflows.

Note, however, that the definition of the models and the contrasts in sections Section and Section are useful for all users as these also have to be defined in our interactive Apps.

This chapter explains the main concepts for data-processing and statistical analysis of (DIA) proteomics data using `msqrob2`. To illustrate these concepts, we will use a publicly available spike-in study published by Staes et al. (Staes et al. 2024). They spiked digested UPS proteins in a yeast digested background at the following ratios (yeast:ups ratio 10:1, 10:2, 10:4, 10:8, 10:10). Here we will use a subset of the data, i.e. dilutions 10:2, 10:4 and 10:8. We will use output of the search engine DIA-NN 2.2.0. The main search output for this DIA-NN version was stored in the report.parquet file in the DIA-NN output directory.

Background

Conventional MS-based workflow

In a nutshell, the wetlab workflow starts with sample preparation where the samples are collected, and the protein content is extracted and digested into peptides. To reduce the sample complexity, the peptides are then separated based on physicochemical properties (mostly hydrophobicity) using liquid chromatography (LC). Peptides are then ionised by an electrospray as they elute from the chromatographic column. The signal over time generated by the eluting ions is called the total ion chromatogram. The ions are then sent for a first round of MS to

record their m/z distribution for the intact ions. This provides an overview of the ions that elute from the column and allows for further separation of the ions in the m/z space. The second round of MS (MS2) records the fragmented ions for a selection of ions, generally the most intense MS1 peaks[1]. This process is repeated for every sample so that every sample is acquired in one MS run. This provides the ion's mass fingerprint. For LFQ workflows, the accumulated MS1 intensity over time, also known as the area under the curve, around the target mass is used as a quantification measure. On the other hand, the ion mass fingerprint, called the MS2 spectrum, enables computational identification of the corresponding peptide using search engines (e.g. Andromeda has been used for this data set) that will provide peptide-to-spectrum matches (PSM). The quantified PSM are further processed by the software (MaxQuant) to obtain a peptide table[2], where every row corresponds to an identified peptide and every column contains information about the peptide and its quantification in one of the samples.

[1]The ion selection for MS2 depends on the data recorded in MS1. Therefore, this approach is referred to as data dependent acquisition (DDA).

[2]MaxQuant also computes a protein table. However, we found that starting from MaxQuant's protein table leads to a decrease in performance. We will illustrate in this tutorial how to build the protein table.

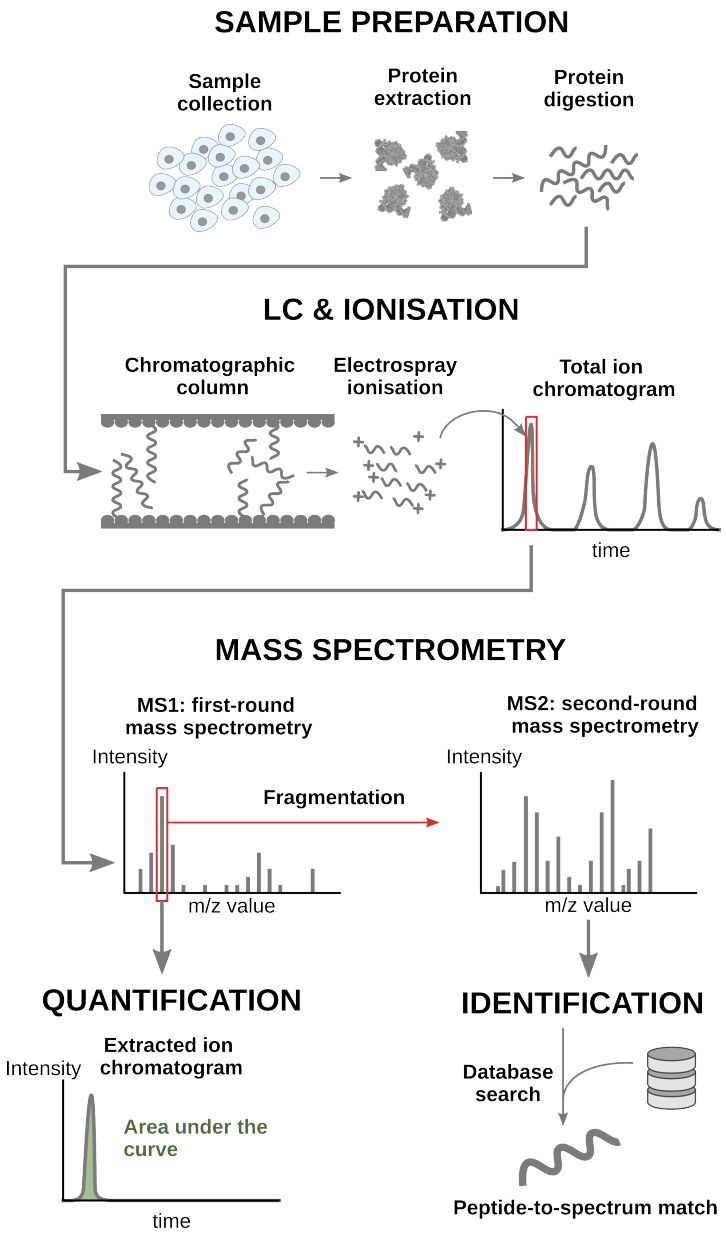


Figure 1: Overview of an LFQ-based proteomics workflow.

- Peptide Characteristics
  - Modifications
  - Ionisation Efficiency: huge variability
  - Identification
    - Misidentification → outliers

- MS^2 selection on peptide abundance
- Context depending missingness
- Non-random missingness

→ Unbalanced peptide identifications across samples and messy data

Data Independent Acquisition

With data independent acquisition, however, the second round of MS (MS2) does not record fragmented ions for selected ions. Instead the instrument divides the m/z range into sequential windows (e.g., 400–425, 425–450 m/z) and all ions within each window are fragmented together. This continues across the full mass range resulting in a systematic fragmentation of all peptides. Hence, the MS2 spectra are complex as many peptides mixed. So search engines and quantification software, such as DIA-NN and Spectronaut, have to deconvolute the complex MS2 spectra so as to identify and quantify individual precursors (MS1) in the m/z window using their MS2 fragments. Hence, the quantification can be done using either MS1 or MS2.

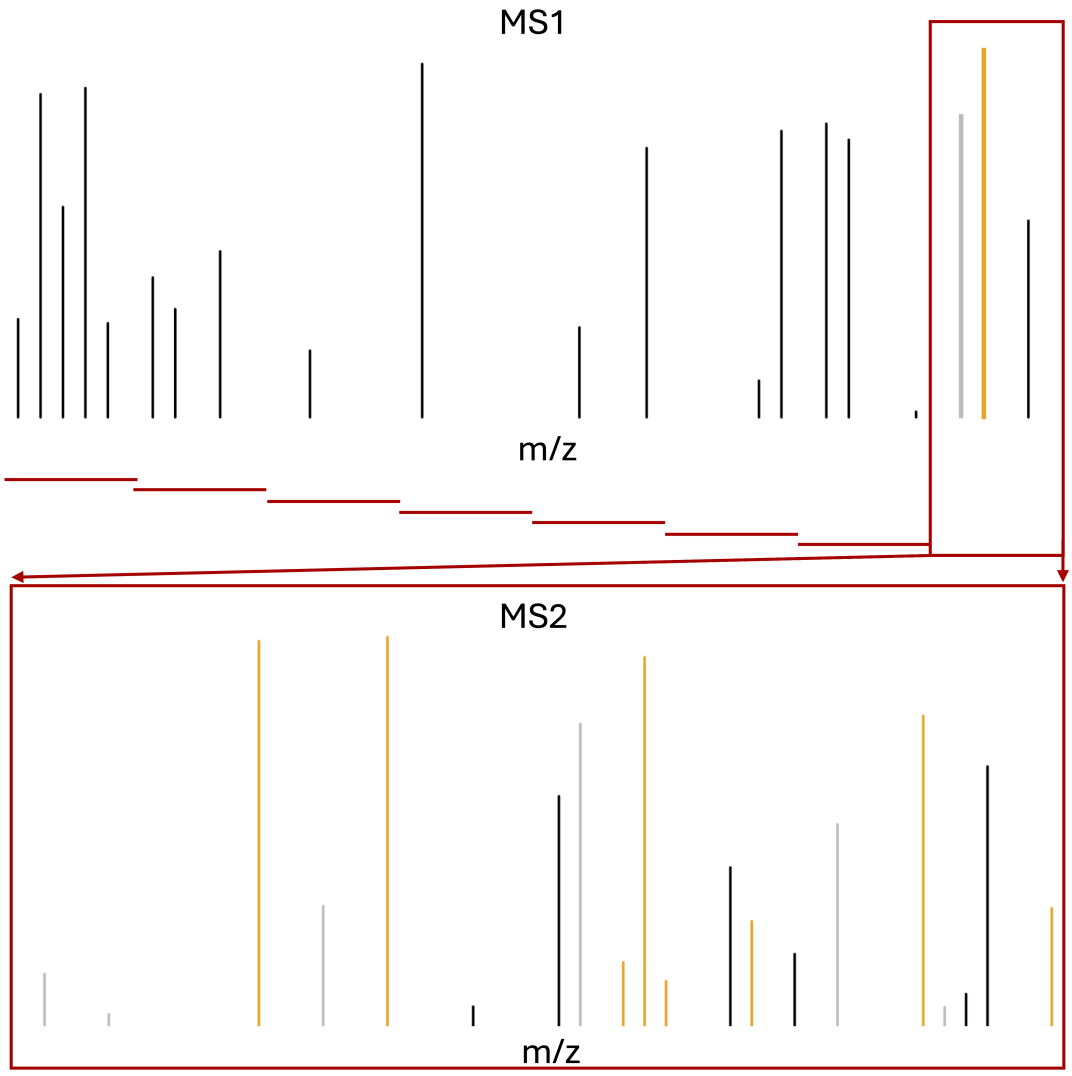


Figure 2: Data-independent acquisition.

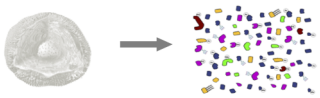
- m/z range into sequential windows (e.g., 400–425, 425–450 m/z)
- MS2 spectra are complex as many peptides mixed
- Deconvolution of the MS2 signal
- With dedicated software such as Spectronaut or DIA-NN
  - Identification based on spectral libraries
  - Library free: FASTA database to predict in silico spectra, retention times, and ion mobilities, which are then used to search the DIA data

  - Use of decoys sequences to estimate false discovery rate of ID

- Quantification using MS2 and/or MS1 peaks

Level of quantification

- MS-based proteomics returns peptides or precursors: pieces of proteins



- Quantification commonly required on the protein level

Challenges

Behind this workflow lies several challenges that will affect the data modelling:

- MS-based proteomics does not measure proteins directly, but their constituting **peptide ions**. The protein-level information needs to be reconstructed from the ion data. In this tutorial, we will start from the precursor-level data, which has been constructed from the ion data by DIA-NN.
- All peptides do not ionise with the same efficiency. Poor ionisation will lead to reduced signal as less ions will hit the detector, hence leading to a huge variability in intensity among different peptide species, even when they originate from the same protein.
- The identification step is not trivial and prone to errors[3]. PSM misidentification leads to the assignment of a quantitative value from another peptide with likely another ionisation efficiency and relative abundance. Hence this misassigned values often lead to outliers.
- Moreover, there is a widespread data missingness, which is often related to the underlying quantification value. This phenomenon is known as missingness not at random. Next to that, many reasons can lead to ions not being identified irrespective of their quantification value leading to missingness that is not related to its quantitative value. This is referred to as missingness completely at random. The missingness issue is not negligible as we shall see upon reading the data.
- Identification issues lead to unbalanced peptide missingness across samples, and the patterns of missing values are potentially different for every peptide, highlighting the need for an automatised solution that is robust against missing values.

[3]Improving peptide identification is outside the scope of this tutorial

- Technical variations during the experiment can lead to systematic fluctuations across samples. The most obvious reason is when different sample amounts are injected into the instruments, due to small pipetting inconsistencies for instance. However, these differences lead to unwanted variation that should be discarded when answering biological questions.

DIA workflow

DIA-NN provides multiple quantifications, e.g. derived from the MS1, MS2 or MS1 combined with MS2 spectra, and at precursor or protein (protein group) level. The term 'precursor' refers to a charged peptide species and is the basic unit of identification and quantification in DIA. Hence, in the context of DIA we refer to a precursor table, instead of to a PSM table in DDA.

Examples of different quantities are:

- raw MS1 area: Ms1.Area, normalised MS1 Area: Ms1.Normalised, MS1 and MS2 Precursor (fragment) Quantities: Precursor.Quantity, Normalised Precursor quantities: Precursor.Normalised, etc., which are all at the precursor level
- MS2 based summary at the protein (protein group)-level: PG.MaxLFQ

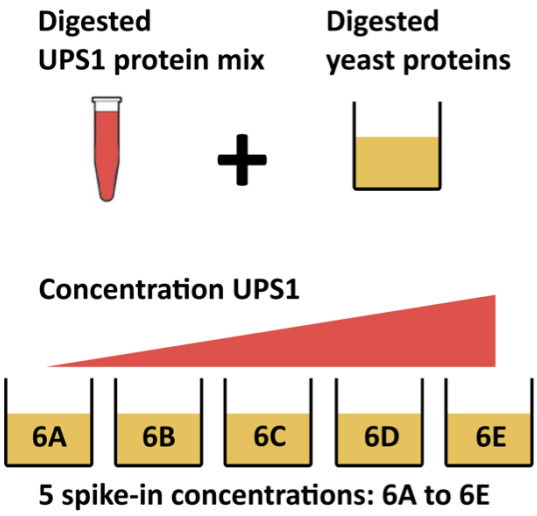
Here, we will use the `Precursor.Quantity` column.

Experimental context

This chapter explains how to analyse a proteomics data set that has been generated using data independent acquisition (DIA). We will again use an in-house spike-in study to illustrate the analysis. The DIA case-study is a subset of Staes et al. (Staes et al. 2024). They spiked digested UPS proteins in yeast at the following ratios (yeast:ups ratio 10:1, 10:2, 10:4, 10:8, 10:10) in a yeast digest background.

Each sample was analyzed in triplicate using an Ultimate 3000 RSLC ProFlow nano-LC system in-line connected to a Q Exactive HF BioPharma mass spectrometer (Thermo).

Here, we will only use the data of the samples from the middle 3 spike-in ratios (2,4 and 8) that were searched using DIA-NN 2.2.0. The main search output for this DIA-NN version is stored in the report.parquet file in the DIA-NN output directory.



Load packages

We load the `msqrob2` package, along with additional packages for data manipulation and visualisation.

Click to see code

```
library("QFeatures")
library("dplyr")
library("tidyr")
library("ggplot2")
library("msqrob2")
library("stringr")
library("ExploreModelMatrix")
library("MsCoreUtils")
library("matrixStats")
library("patchwork")
library("kableExtra")
library("ComplexHeatmap")
library("purrr")
library("tibble")
library("scater")
library("ggcorrplot")
```

Parallelisation

`msqrob2` can parallelise computations during the model estimation to improve speed. However, we will disable parallelisation to ensure this vignette can be run regardless of hardware. Parallelisation is controlled using the `BiocParallel` package.

```
library("BiocParallel")
register(SerialParam())
```

If you want to use `msqrob2` with parallelisation enabled and using 4 cores, you can run the following:

```
register(MulticoreParam(workers = 4))
```

Be mindful that, while parallelisation can improve speed, it will also consume more RAM because part of the data will be copied multiple times over your different workers. If you experience crashes because you exceeded the amount of available RAM on your machine, you should reduce the number of requested workers.

Data

Precursor table

We load the output from DIA-NN parquet file.

Click to see code

```
library("BiocFileCache")
bfc <- BiocFileCache()
precursorFile <- bfcrpath(bfc,"https://github.com/statOmics/msqrob2data/raw/refs/heads/main/
```

We can import the report.parquet file using the `read_parquet` function from the `arrow` package. Note, that older versions of DIA-NN store the output as report.tsv.

```
precursors <- arrow::read_parquet(precursorFile) # function from the arrow package
#precursors <- data.table::fread(precursorFile) # For older versions of DIA-NN, where the res
```

Each row in the precursor data table is in "long format" and contains information about one precursor in a specific run (the table below shows the first 6 rows). The columns contains various descriptors about the precursor, such as its sequence, its charge, run, etc. Some of these columns contain the quantification values, e.g. Ms1.Area, Precursor.Quantity etc.

Run.Index	Run	Precursor.Id	Modified.Sequence	Stripped.Sequence	Precursor.Charge	Precursor.Quantity	Precursor.Normalised	Ms1.Normalised	Normalisation.Quantity	Empirical.Quality	Normalised.Empirical.Quality	Evidence	Mass.Evidence	Channel.Evidence	Ms1.Profile.Correlation	Ms1.Apex.Area	Total.Ms1.Signal	Ms1.Total.Signal.After
0	B000[illegible]765_[illegible]ratio02_DIA	[illegible]	[illegible]	[illegible]	2	0	10[illegible]	[illegible]	0.0019836	0.0107658								
1	B000[illegible]765_[illegible]ratio04_DIA	[illegible]	[illegible]	[illegible]	2	0	16[illegible]	[illegible]	0.0012512									
2	B000[illegible]765_[illegible]ratio08_DIA	[illegible]	[illegible]	[illegible]	2	0	14[illegible]	[illegible]	0.0018616									
3	B000[illegible]765_[illegible]ratio02_DIA_2	[illegible]	[illegible]	[illegible]	2	0	12[illegible]	[illegible]	0.0038757									
5	B000[illegible]765_[illegible]ratio08_DIA_2	[illegible]	[illegible]	[illegible]	2	0	47[illegible]	[illegible]	0.0014648									
6	B000[illegible]765_[illegible]ratio02_DIA_3	[illegible]	[illegible]	[illegible]	2	0	12[illegible]	[illegible]	0.0023804	0.0058500								

Click to see code

We filter the data to reduce the memory footprint.

```
precursors <- precursors |>
  select(
         Run,
         Precursor.Id,
         Modified.Sequence,
         Stripped.Sequence,
         Precursor.Charge,
```

```
          Protein.Group,
          Protein.Names,
          Protein.Ids,
          Genes,
          Precursor.Quantity,
          Precursor.Normalised,
          Normalisation.Factor,
          Ms1.Area,
          Ms1.Normalised,
          PG.MaxLFQ,
          Q.Value,
          Lib.Q.Value,
          PG.Q.Value,
          Lib.PG.Q.Value,
          Proteotypic,
          Decoy, # Not available in older versions of DIA-NN
          RT)
```

We known the ground truth: UPS proteins are differentially abundant (DA, spiked in), Yeast proteins are not.

```
precursors <- precursors |>
  mutate(species = grepl(pattern = "UPS",Protein.Group) |>
           as.factor() |>
           recode("TRUE"="ups","FALSE" = "yeast"))
precursors |>
  pull(species) |>
  table()
```

```
 yeast    ups
301195   4481
```

Sample annotation table

The sample annotation table) is not available and can be generated from the run labels, as the researchers included information on the design in the filenames.

Click to see code

We will make a new data frame with the annotation.

1. Extract unique Run labels
2. Split Run label into variables using the "_" separator
3. Rename the variable Run to runCol (needed for `readQFeatures` function)
4. Add sampleId
5. Sort annotation file according to sampleId

```
(annot <- precursors |>
  dplyr::distinct(Run) |>
  mutate(runCol=Run) |>
  separate(Run,
           into = c(paste("label",1:8), "condition","label9","rep")) |>
  mutate(
    condition = substr(condition,7,7),
    ratio = as.double(condition),
    rep = ifelse(is.na(rep),1,rep),
    sampleId = paste(condition,rep, sep = "_")) |>
  select(runCol, sampleId, condition,ratio,rep)
)
```

```
# A tibble: 9 x 5
  runCol                                          sampleId condition ratio rep
  <chr>                                           <chr>    <chr>     <dbl> <chr>
1 B000254_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02~ 2_1      2             2 1
2 B000258_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio04~ 4_1      4             4 1
3 B000262_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio08~ 8_1      8             8 1
4 B000278_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02~ 2_2      2             2 2
5 B000286_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio08~ 8_2      8             8 2
6 B000302_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02~ 2_3      2             2 3
7 B000306_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio04~ 4_3      4             4 3
8 B000310_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio08~ 8_3      8             8 3
9 B000282_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio04~ 4_2      4             4 2
```

Convert to QFeatures

Bioconductor Data structures for MS-based Proteomics

`msqrob2` is built around the `QFeatures` class. We refer to the R for mass spectrometry book for a comprehensive description of the class. In a nutshell, the `QFeatures` package provides infrastructure to manage and analyse quantitative features from mass spectrometry experiments. It is based on the `SummarizedExperiment` and `MultiAssayExperiment` classes. It leverages the hierarchical structure of proteomics experiments: proteins are composed of peptides, themselves produced by spectra. Each piece of information is stored in an individual

`SummarizedExperiment` object, later referred to as a "set". Throughout the aggregation and processing of these data, the relations between sets are tracked and recorded, thus allowing users to easily navigate across Precursor(DIA)/PSM(DDA), peptide and protein quantitative data.

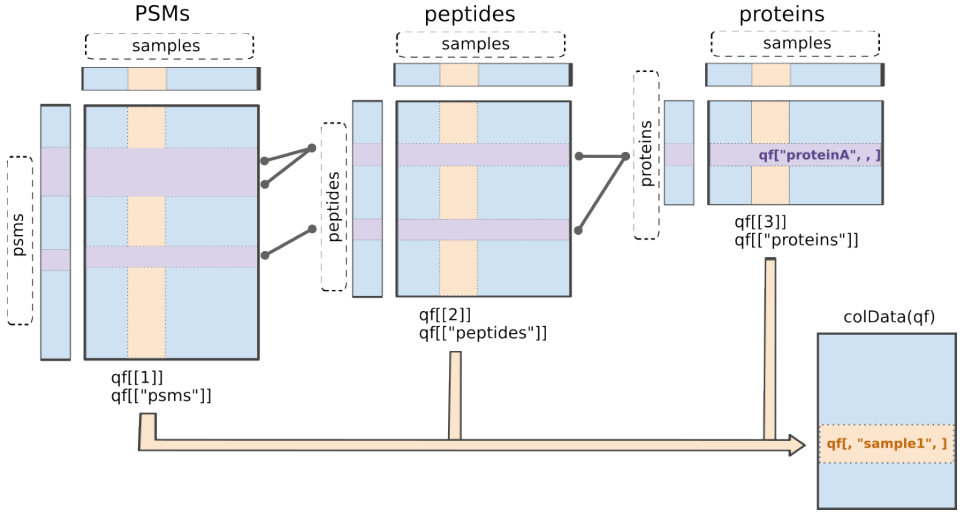


Figure 3: Illustration of the `QFeatures` data class.

The `readQFeatures()` enables a seamless conversion of tabular data into a `QFeatures` object. We provide the peptide table and the sample annotation table. The function will use the `quantCols` column in the sample annotation table to understand which columns in `peptides` contain the quantitative values, and automatically link the corresponding sample annotation with the quantitative values. We also tell the function to use the `Sequence` column as peptide identifier, which will be used as rownames. See `?readQFeatures()` for more details.

First, recall that the precursor table is a file in long format. Every quantitative column in the precursor table contains information for multiple runs. Therefore, the function splits the table based on the run identifier, given by the `runCol` argument (for DIA-NN, that identifier is contained in `run`). So, the `QFeatures` object after import will contain as many sets as there are runs. Next, the function links the annotation table with the PSM data. To achieve this, the annotation table must contain a `runCol` column that provides the run identifier in which each sample has been acquired, and this information will be used to match the identifiers in the `Run` column of the precursor table.

Here, we will use the `Precursor.Quantity` column as quantification input.

Note, that we filter a number of variables to reduce the footprint of the QFeatures object.

```
(qf <- readQFeatures(assayData = precursors,
                                colData = annot,
```

```
                                quantCols = "Precursor.Quantity",
                                runCol = "Run",
                                fnames = "Precursor.Id"))
```

```
  |
  |                                                                      |   0%
  |
  |========                                                              |  11%
  |
  |================                                                      |  22%
  |
  |========================                                              |  33%
  |
  |================================                                      |  44%
  |
  |========================================                              |  56%
  |
  |================================================                      |  67%
  |
  |========================================================              |  78%
  |
  |================================================================      |  89%
  |
  |========================================================================| 100%
```

```
An instance of class QFeatures (type: bulk) with 9 sets:

 [1] B000254_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA: SummarizedExperiment with 34471 rows
 [2] B000258_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio04_DIA: SummarizedExperiment with 35142 rows
 [3] B000262_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio08_DIA: SummarizedExperiment with 34028 rows
 ...
 [7] B000302_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA_3: SummarizedExperiment with 33091 ro
 [8] B000306_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio04_DIA_3: SummarizedExperiment with 34472 ro
 [9] B000310_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio08_DIA_3: SummarizedExperiment with 33724 ro
```

We now have a `QFeatures` object with 9 sets, one set for each run that are named based on their run name in the runCol. The sample annotations can be retrieved using `colData()`.

```
colData(qf)
```

```
knitr::kable(head(colData(qf)))
```

	runCol	sampleId	condition	ratio	rep
B000254_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA	B000254_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA	2_1	2	2	1
B000258_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio04_DIA	B000258_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio04_DIA	4_1	4	4	1
B000262_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio08_DIA	B000262_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio08_DIA	8_1	8	8	1
B000278_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA_2	B000278_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA_2	2_2	2	2	2
B000282_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio04_DIA_2	B000282_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio04_DIA_2	4_2	4	4	2
B000286_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio08_DIA_2	B000286_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio08_DIA_2	8_2	8	8	2

We can get a sample annotation directly using the `$` accessor.

```
qf$condition
```

```
[1] "2" "4" "8" "2" "4" "8" "2" "4" "8"
```

We can extract the `SummarizedExperiment` object for the `B000254_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA` set using double bracket subsetting[4]

```
qf[["B000254_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA"]]
```

```
class: SummarizedExperiment
dim: 34471 1
metadata(0):
```

[4] A `QFeatures` object can be seen as a special list of `SummarizedExperiment` objects.

```
assays(1): ''
rownames(34471): (UniMod:1)AAVTLHLR2 (UniMod:1)ADFLLRPIK2 ...
  YYYELAYPMGISASHK3 YYYITPISSK2
rowData names(22): Run Precursor.Id ... RT species
colnames(1): B000254_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA
colData names(0):
```

But notice that the sample annotations were not extracted along with the SummarizedExperiment (`colData names(0):`). This can be performed using `getWithColData()`, which extracts the set of interest (like `[[]]`) along with all the associated sample annotations.

```r
getWithColData(qf, "B000254_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA")
```

```
class: SummarizedExperiment
dim: 34471 1
metadata(0):
assays(1): ''
rownames(34471): (UniMod:1)AAVTLHLR2 (UniMod:1)ADFLLRPIK2 ...
  YYYELAYPMGISASHK3 YYYITPISSK2
rowData names(22): Run Precursor.Id ... RT species
colnames(1): B000254_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA
colData names(5): runCol sampleId condition ratio rep
```

The precursor annotations are available for in the corresponding `rowData`.

```r
rowData(qf[["B000254_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA"]])
```

```r
knitr::kable(head(rowData(qf[["B000254_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA"]])))
```

	Run	Precursor.Id	Modified.Sequence	Stripped.Sequence	Precursor.Charge	Protein.Group	Protein.Ids	Protein.Names	Genes	Precursor.Normalised	Normalisation.Factor	Ms1.Normalised	Ms1.Area	[illegible]	Proteotypic	RT	species
(UniMod:1)AAVTLHLR2	B000254_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA	(UniMod:1)AAVTLHLR2	(UniMod:1)AAVTLHLR	AAVTLHLR	2	P38998	P38998	LYS1_YEAST	LYS1	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	89.48647	yeast

	Run	Precursor.Id	Modified.Sequence	Stripped.Sequence	Precursor.Charge	Protein.Group	Protein.Ids	Protein.Names	Genes	Precursor.Normalised	Normalisation.Factor	Ms1.Area	Ms1.Normalised	…	Proteotypic	…	species
(UniMod:1)ADFLLRPIK2	B000254_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA	(UniMod:1)ADFLLRPIK2	(UniMod:1)ADFLLRPIK	ADFLLRPIK	[illegible]	P38768	P38768	PIH1_YEAST	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	121.34409	yeast
(UniMod:1)ADQENSPLHTVGFDAR2	B000254_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA	(UniMod:1)ADQENSPLHTVGFDAR2	(UniMod:1)ADQENSPLHTVGFDAR	ADQENSPLHTVGFDAR	[illegible]	Q01519	Q01519	COX12_YEAST	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	96.37570	yeast
(UniMod:1)ADRPAIQLIDEEK2	B000254_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA	(UniMod:1)ADRPAIQLIDEEK2	(UniMod:1)ADRPAIQLIDEEK	ADRPAIQLIDEEK	[illegible]	Q99287	Q99287	SEY1_YEAST	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	99.30938	yeast
(UniMod:1)ADSRDPASDQMQHWK3	B000254_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA	(UniMod:1)ADSRDPASDQMQHWK3	(UniMod:1)ADSRDPASDQMQHWK	ADSRDPASDQMQHWK	[illegible]	UPS|P04040ups|CATA_HUMAN_UPS	UPS|P04040ups|CATA_HUMAN_UPS	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	72.52413	ups
(UniMod:1)ADYSSLTVVQLKD2	B000254_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA	(UniMod:1)ADYSSLTVVQLKD2	(UniMod:1)ADYSSLTVVQLK	ADYSSLTVVQLK	[illegible]	P40014	P40014	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	116.96347	yeast

We can also retrieve the quantitative values for each set using `assay()`. Here we only show the first 5 rows of the intensity matrix for the first summarized experiment.

```r
assay(qf[[1]]) |> head(5)
```

```
                             B000254_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA
(UniMod:1)AAVTLHLR2                                                     1027105
(UniMod:1)ADFLLRPIK2                                                    5792544
(UniMod:1)ADQENSPLHTVGFDAR2                                             2756378
(UniMod:1)ADRPAIQLIDEEK2                                               25626726
(UniMod:1)ADSRDPASDQMQHWK3                                             20512458
```

Note, that there is only one column, because each run has been stored in a separate summarized experiment. We will join the data of the runs later.

Since we have a `QFeatures` object, we can directly make use of `QFeatures`' data preprocessing functionality[5]. A major advantage of `QFeatures` is it provides the power to build highly modular workflows, where each step is carried out by a dedicated function with a large choice of available methods and parameters. This means that users can adapt the workflow to their specific use case and their specific needs.

Data preprocessing

The data preprocessing workflow for DIA data is similar to the workflow for DDA-LFQ data, but there are subtle differences as we start from precursor level data and have additional columns.

Encoding missing values

The first preprocessing step is to correctly encode the missing values. It is important that missing values are encoded using `NA`. For instance, non-observed values should not be encoded with a zero because true zeros (the proteomic feature is absent from the sample) cannot be distinguished from technical zeros (the feature was missed by the instrument or could not be identified). We therefore replace any zero in the quantitative data with an `NA`.

```
qf <- zeroIsNA(qf, names(qf))
```

Note that `msqrob2` can handle missing data without having to rely on hard-to-verify imputation assumptions, which is our general recommendation. However, `msqrob2` does not prevent users from using imputation, which can be performed with `impute()` from the `QFeatures` package.

Precursor Filtering

Filtering removes low-quality and unreliable precursors that would otherwise introduce noise and artefacts in the data.

[5] see also the R for mass spectrometry book

Remove questionable identifications

We apply standard filtering advised by DIA-NN:

1. q-value threshold of 0.01 for the identification of precursors (`Q.Value`) and protein groups (`PG.Q.Value`). If the file is processed via matching between runs, it is also useful to filter on Lib.Q.Value and Lib.PG.Q.Value.
2. Remove precursors that could not be mapped, i.e. when `Precursor.Id` column is an empty string.
3. Filter decoys, i.e. only keep precursors for which the `Decoy` column equals 0. (Note, that the `Decoy` column is not present in the output of older versions of DIA-NN)
4. Keeping only proteotypic peptides, which map uniquely to a specific protein.

Click to see code

```
qf <-  qf |>
  filterFeatures(~ Q.Value <= 0.01 & #1.
                   PG.Q.Value <= 0.01 &  #1.
                   Lib.Q.Value <= 0.01 & #1.
                   Lib.PG.Q.Value <= 0.01 & #1.
                   Precursor.Id != "" & #2.
                   Decoy == 0 & #3. (Note, that this filter is not available with previous
                   Proteotypic == 1) #4.
```

Note, that it is important that the filtering criteria are not distorting the distribution of the test statistics in the downstream analysis for features that are non-DA. It can be shown that filtering will not induce biased results when the filtering criterion is independent of test statistic. The criteria that we proposed above are all based on the results of the identification step, hence, they are independent of the downstream test statistics that will be used to prioritize DA proteins.

Assay joining

Up to now, the data from different runs were kept in separate assays. We can now join the normalised sets into an precursor set using joinAssays(). Sets are joined by stacking the columns (samples) in a matrix and rows (features) are matched according to a row identifier, here the `Precursor.Id`. We will store the result in the assay with name: `precursors`.

Click to see code

```
(qf <- joinAssays(
  x = qf,
  i = names(qf),
  fcol = "Precursor.Id",
  name = "precursors"
  ))
```

```
An instance of class QFeatures (type: bulk) with 10 sets:

 [1] B000254_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA: SummarizedExperiment with 32848 rows
 [2] B000258_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio04_DIA: SummarizedExperiment with 33491 rows
 [3] B000262_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio08_DIA: SummarizedExperiment with 32406 rows
 ...
 [8] B000306_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio04_DIA_3: SummarizedExperiment with 32829 ro
 [9] B000310_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio08_DIA_3: SummarizedExperiment with 32128 ro
 [10] precursors: SummarizedExperiment with 38615 rows and 9 columns
```

We will again retrieve the quantitative values for each set using `assay()`. Here we only show the first 5 rows and columns of the intensity matrix and illustrate default subsetting that is possible.

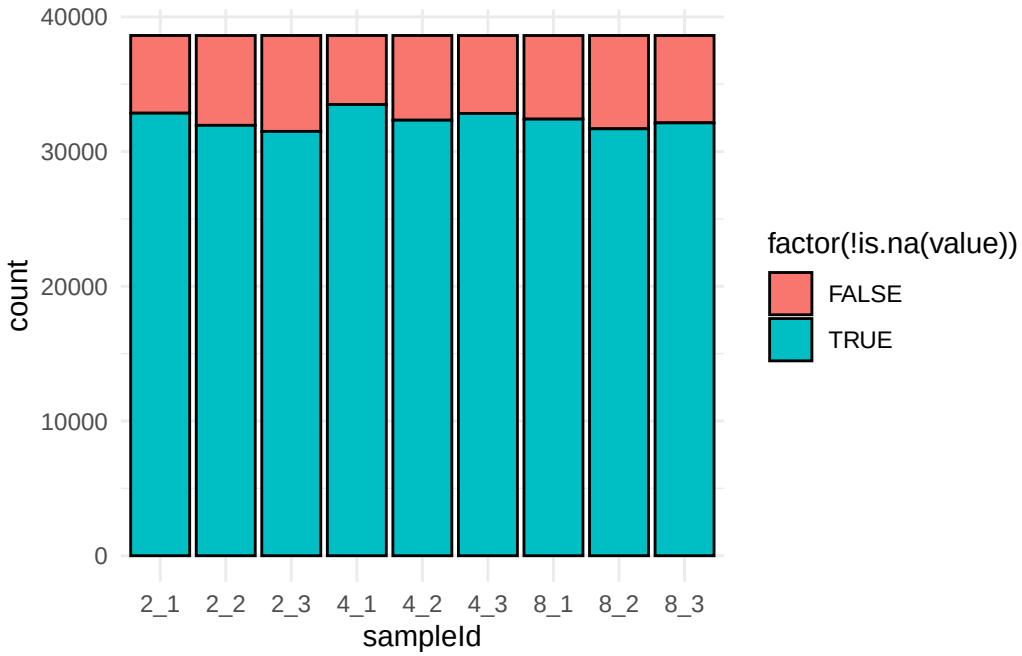
```
assay(qf [["precursors"]])[1:5, 1:5]
```

```
                          B000254_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA
(UniMod:1)ADFLLRPIK2                                                5792544.5
(UniMod:1)ADRPAIQLIDEEK2                                           25626726.0
(UniMod:1)ADSRDPASDQMQHWK3                                         20512458.0
(UniMod:1)ADYSSLTVVQLK2                                              970017.8
(UniMod:1)AEASIEK1                                                 74863536.0
                          B000258_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio04_DIA
(UniMod:1)ADFLLRPIK2                                                 5616620
(UniMod:1)ADRPAIQLIDEEK2                                            23988660
(UniMod:1)ADSRDPASDQMQHWK3                                          54094184
(UniMod:1)ADYSSLTVVQLK2                                              1394431
(UniMod:1)AEASIEK1                                                  92867856
                          B000262_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio08_DIA
(UniMod:1)ADFLLRPIK2                                                 4810394
(UniMod:1)ADRPAIQLIDEEK2                                            23561136
(UniMod:1)ADSRDPASDQMQHWK3                                         156667664
(UniMod:1)ADYSSLTVVQLK2                                              888451
(UniMod:1)AEASIEK1                                                  87658384
```

```
                                  B000278_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA_2
(UniMod:1)ADFLLRPIK2                                                           5575437
(UniMod:1)ADRPAIQLIDEEK2                                                      18285408
(UniMod:1)ADSRDPASDQMQHWK3                                                    21354380
(UniMod:1)ADYSSLTVVQLK2                                                        1183666
(UniMod:1)AEASIEK1                                                            55235320
                                  B000282_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio04_DIA_2
(UniMod:1)ADFLLRPIK2                                                           4381680
(UniMod:1)ADRPAIQLIDEEK2                                                      16335785
(UniMod:1)ADSRDPASDQMQHWK3                                                    50734624
(UniMod:1)ADYSSLTVVQLK2                                                        1069145
(UniMod:1)AEASIEK1                                                            63096800
```

Filtering: Remove highly missing precursors

```
qf[,,"precursors"] |>
  longForm(colvars = colnames(colData(qf))) |>
  ggplot(aes(x = sampleId)) +
  geom_bar(aes(fill = factor(!is.na(value))),
           colour = "black") +
  theme_minimal()
```



We keep peptides that were observed at least 3 times out of the $n = 9$ samples, so that we can estimate the peptide characteristics. We tolerate the following proportion of NAs: $\text{pNA} = \frac{(n-3)}{n} = 0.666$, so we keep peptides that are observed in at least 33.3% of the samples, which corresponds to one treatment condition. This is an arbitrary value that may need to be adjusted depending on the experiment and the data set.

Click to see code

```
nObs <- 3
n <- ncol(qf[["precursors"]])

(qf <- filterNA(qf, i = "precursors", pNA = (n - nObs) / n))
```

```
An instance of class QFeatures (type: bulk) with 10 sets:

 [1] B000254_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA: SummarizedExperiment with 32848 rows
 [2] B000258_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio04_DIA: SummarizedExperiment with 33491 rows
 [3] B000262_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio08_DIA: SummarizedExperiment with 32406 rows
 ...
 [8] B000306_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio04_DIA_3: SummarizedExperiment with 32829 r
 [9] B000310_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio08_DIA_3: SummarizedExperiment with 32128 r
 [10] precursors: SummarizedExperiment with 36247 rows and 9 columns
```

Filter one-hit wonders

Here, we remove proteins that can only be found by one peptide, as such proteins may not be trustworthy.

1. We first calculate how many distinct peptides map to each protein (`Protein.Group`). We use the stripped precursor sequence, i.e. sequence of the base peptide for this purpose.
2. We store this information in the row data of the precursors assay.
3. We filter precursors of one-hit wonder proteins.

Click to see code

```
# Filter for peptides per protein
pepsPerProtDf <- qf[["precursors"]] |>
    rowData() |>
    data.frame() |>
    dplyr::select("Stripped.Sequence", "Protein.Group") |>
    group_by(Protein.Group) |>
```

```
    mutate(pepsPerProt = Stripped.Sequence |>
                unique() |> length()
           ) #1.

rowData(qf[["precursors"]])$pepsPerProt <- pepsPerProtDf$pepsPerProt #2.

qf <- filterFeatures(qf,
                               ~ pepsPerProt > 1,
                               keep = TRUE) #3.
```

Log-transformation

We typically start a MS-based proteomics data analysis by log2 transforming the intensities. We illustrate the rationale using the spike-in data.

Details for plot

For this, we perform a short data manipulation pipeline:

1. We use `longForm()` to convert the `QFeatures` object into a long table, where each row contains the quantitative information about one observation, in which column, row and set it was found. Long tables are particularly useful for manipulating data with the `tidyverse` ecosystem, namely with `ggplot2` for visualisation. `longForm()` also allows to include annotations, and we here include `Mixture` and `TechRepMixture` for filtering and colouring.
2. `longForm()` returns a `DataFrame` which we convert to a `data.frame`.
3. We filter the data to keep only the data from precursor "(UniMod:1)ADSRDPASDQMQHWK3".

```
dat <- longForm(qf, colvars = colnames(colData(qf))) |> ## 1.
  data.frame() |> ## 2.
  filter(rowname == "(UniMod:1)ADSRDPASDQMQHWK3") ## 3.
```

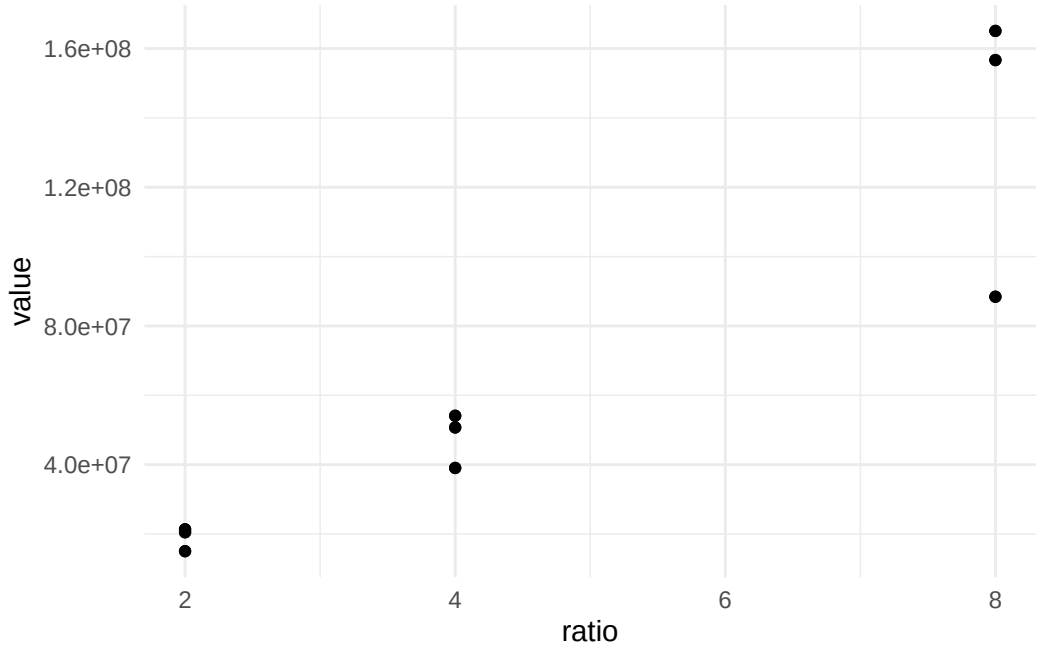
Next, we visualise the data using `ggplot2`. We will show the intensities and log2-intensities for one of the precursors, ADSRDPASDQMQHWK3, in function of the known spike-in ratio. We first define a common plot from which we generate two plots, one without and the other with log2 transformation of the quantitative values.

```
## Common plot
p <- ggplot(dat) +
  aes(x = ratio) +
  geom_point() +
```

```
  theme_minimal()
## Add the x and y axis as linear or log2-transformed
p2 <-
  p + aes(y = log2(value), x = log2(ratio)) +
  plot_annotation(title = "ups precursor ADSRDPASDQMQHWK3") +
  plot_layout(axis_titles = "collect_x")
```

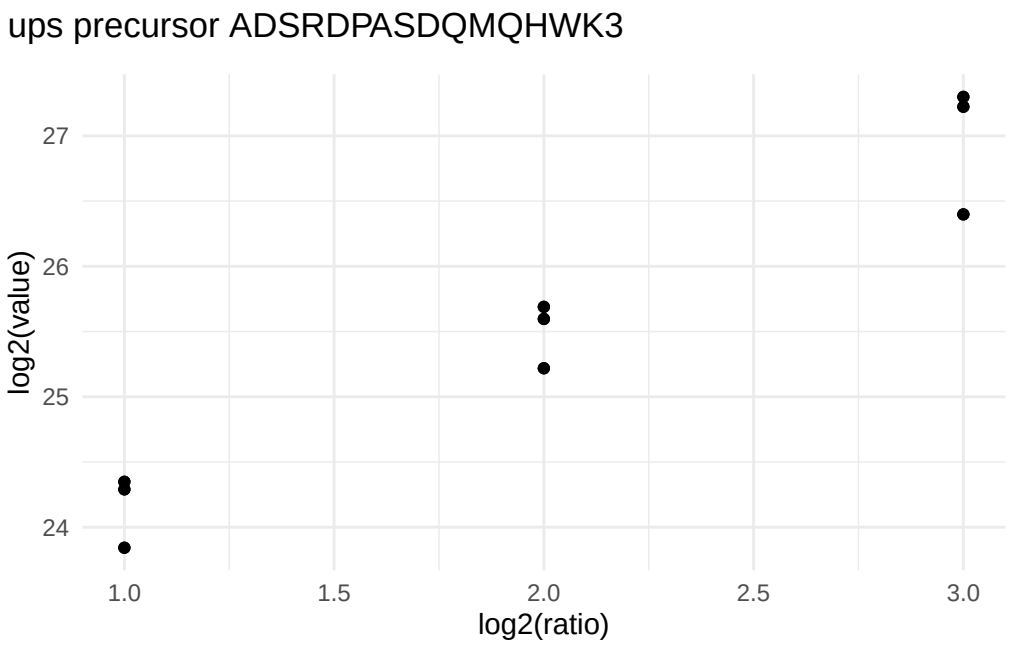
Original plot

```
p + aes(y = value)
```



Plot upon log2 -transformation

```
p2
```



We can see that the variation around the mean for each concentration slightly increases as the mean increases. This is known as heteroskedasticity. We will later see that `msqrob2` assumes that variance of the error is equal across the different conditions. Upon log2-transformation we can see that the problem of unequal variability is solved.

Another advantage of log2-transformation is that it provides a scale that directly relates to biological interpretation. In biology, a change induced by some condition often results in a fold change in concentration. Interestingly, the log2 fold change (logFC), which is the log2 of the ratio between two conditions, is identical to the difference between the log of each condition.

$$log_2 FC_{b-a} = log_2 b - log_2 a = log_2 \frac{b}{a}$$

This simplifies the modelling since the effects are now additive and it provides a straightforward interpretation, i.e. a logFC of 1 means that the abundances are on average $2\times$ higher in b than in a, a logFC of 2 means an average increase of $4\times$, for instance. Also, note that the averages calculated at the log2-scale have the interpretation of log2-transformed geometric means.

We perform log2-transformation with `logTransform()` from the `QFeatures` package. We use `base = 2` and store the result in a new summarized experiment named `precursors_log`

```
qf <- logTransform(qf,
                               base = 2,
                               i = "precursors",
                               name = "precursors_log")
```

Normalisation

The most common objective of MS-based proteomics experiments is to understand the biological changes in protein abundance between experimental conditions. However, changes in measurements between groups can be caused by technical factors. For instance, there are systematic fluctuations from run-to-run that shift the measured intensity distribution.

Click to see details for plot

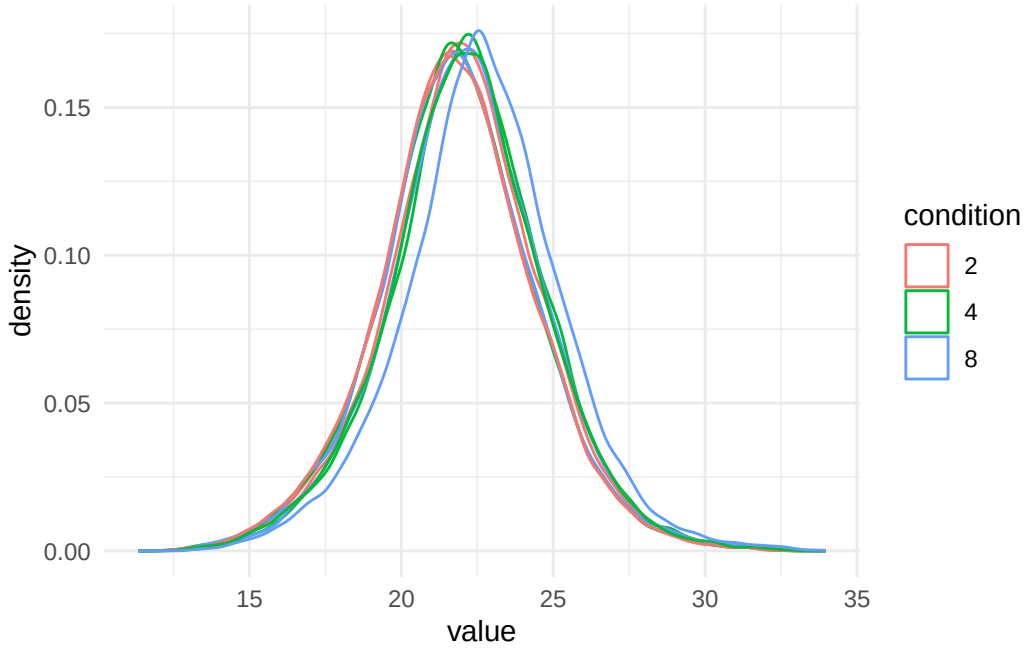
We can explore this as follows:

1. We extract the sets containing the log transformed data. This is performed using `QFeatures`' 3-way subsetting[6].
2. We use `longForm()` to convert the `QFeatures` object into a long table, including `condition` and `concentration` for filtering and colouring.
3. We visualise the density of the quantitative values within each sample. We colour each sample based on its spike-in condition.

```
pNeedNorm <- qf[, , "precursors_log"] |> #1.
  longForm(colvars = colnames(colData(qf))) |>  #2.
  data.frame() |>
  filter(!is.na(value)) |>
  ggplot() + #3.
  aes(x = value,
      colour = condition,
      group = colname) +
  geom_density() +
  theme_minimal()
```

```
pNeedNorm
```

[6]We will explain this with more details at the end of the summarisation step



Even in this clean synthetic data set (same yeast background with some spiked UPS proteins), the marginal precursor intensity distributions across samples are not well aligned. Ignoring this effect will increase the noise and reduce the statistical power of the experiment, and may also, in case of unbalanced designs, introduce confounding effects that will bias the results.

There are many ways to perform the normalisation, e.g. median centering is a popular choice. If we subtract the sample median at the log scale from each precursor log2 intensity, this basically boils down to calculating log-ratios between the precursor intensity and its sample median.

Here, we will use the Median of Ratios method of DESeq2, which was originally developed for RNA-seq data analysis and can also correct for differences in composition of the proteomes in the different samples. The method:

1. First calculates pseudo-reference sample (row-wise mean of log2 intensities, which corresponds to the log2 transformed geometric mean).
2. Then calculates log2 ratios of each sample w.r.t. the pseudo-reference.
3. Finally the column wise median of these log2 ratios is taken to obtain the sample based normalisation factor on the log2 scale.

The figure below illustrates the procedure on a small toy example with four features (rows) and three samples (columns).

This procedure is implemented in the `msqrob2` function `nfLogMedianOfRatios`.

1. We calculate the log transformed normalisation factor from the Median of Ratios method.

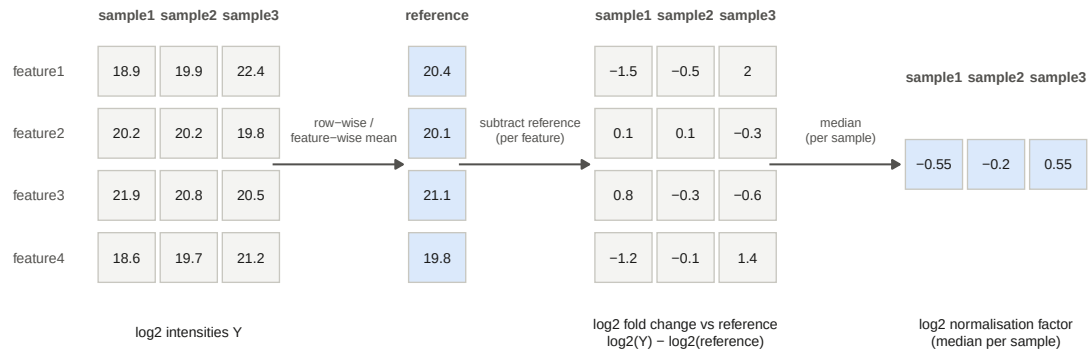


Figure 4: Illustration of the median-of-ratios normalisation on a toy example with four features and three samples. Starting from the matrix of log2 intensities (left), a pseudo-reference sample is obtained by taking the row-wise (feature-wise) mean. Log2 fold changes of every sample relative to this reference are then calculated (middle). Finally, the sample-specific normalisation factor is obtained as the median of these log2 fold changes over all features (right).

2. Subtract these log2-norm factors from the intensities of each corresponding column of the assay data and store the result in the new assay precursors_norm. (We adopt the sweep function to the `precursors_log` assay of the spikein QFeatures object with as statistic the log2 normfactor `STATS = nfLog` the default function `FUN = "-"`, `MARGIN = 2` to substract the column wise log2 norm factor from each entry of the corresponding assay data).

Click to see code

```
nfLog <- nfLogMedianOfRatios(qf, "precursors_log") #1.
qf<- sweep( #Subtract log2 norm factor column-by-column (MARGIN = 2) #2.
  qf,
  MARGIN = 2,
  STATS = nfLog,
  i = "precursors_log",
  name = "precursors_norm"
)
```

We explore the effect of the total normalisation in the subsequent plot.

Formally, the function applies the following operation on each sample i across all precursors p:

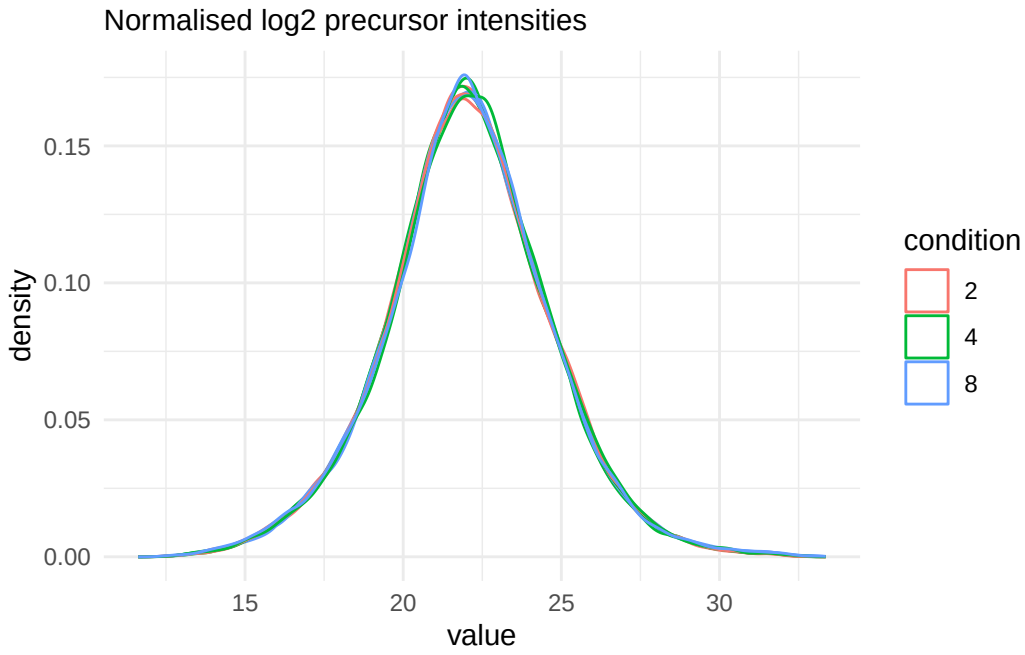
$$y_{ip}^{\text{norm}} = y_{ip} - nf_i$$

with y_ip the log2-transformed intensities and nf_i the log2-transformed norm factor. Upon normalisation, we can see that the distribution of the y_{ip}^{norm} nicely overlaps (using the same code as above)

Click to see code

```
pNormPlot <- qf[, , "precursors_norm"] |>
  longForm(colvars = colnames(colData(qf))) |>
  data.frame() |>
  filter(!is.na(value)) |>
  ggplot() +
  aes(x = value,
      colour = condition,
      group = colname) +
  geom_density() +
  labs(subtitle = "Normalised log2 precursor intensities") +
  theme_minimal()
```

```
pNormPlot
```



Note, that many different normalisation factors are proposed in the literature. We will evaluate different normalisation strategies in the tutorial session.

Users who prefer to use the internal normalisation method can start their data analysis from the pre-normalised data, i.e. use the `Precursor.Normalised` in DIA-NN in the `readQFeatures` function or the msqrob2gui APPs. They can then simply skip the normalisation step, while keeping all other data processing steps as before.

Summarisation

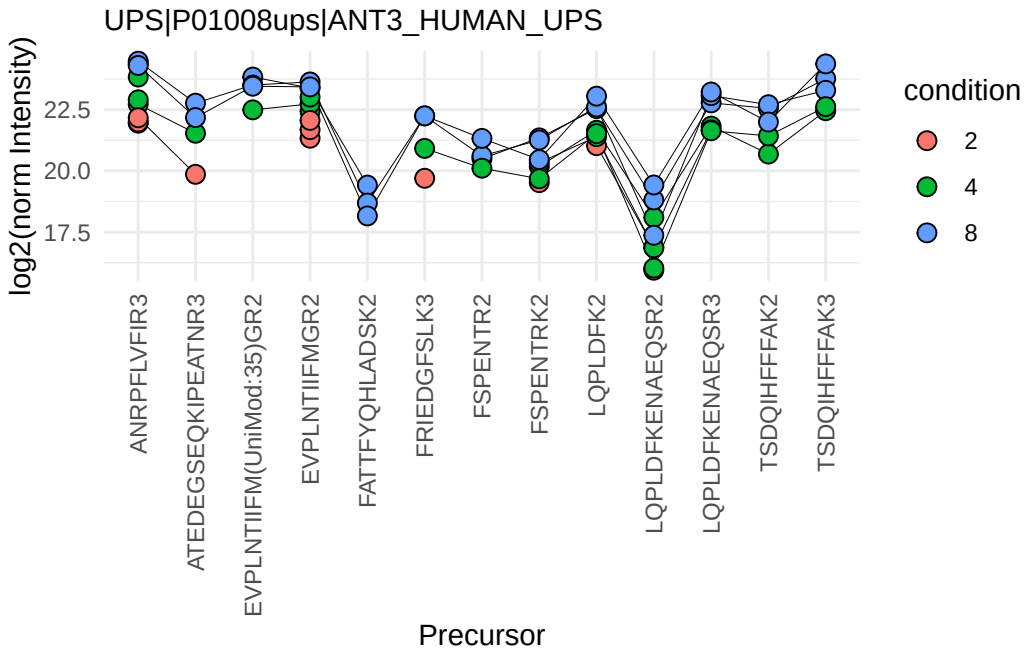
The objective of summarisation (also referred to as aggregation) is to summarise the precursor-level intensities into a protein expression value. We illustrate the motivation for summarisation using all peptide-level data for one of the spiked UPS proteins. The different precursors are shown on the x axis and plot their log2 normalised intensities across samples on y axis. All the points belonging to the same sample are linked through a grey line.

Click to see code

```
protName <- "UPS|P01008ups|ANT3_HUMAN_UPS"
pNeedSum <- qf[,,"precursors_norm"] |>
  longForm(colvars = colnames(colData(qf)), rowvars = "Protein.Ids") |>
  data.frame() |>
  filter(Protein.Ids == protName) |>
```

```
  ggplot() +
  aes(x = rowname,
      y = value,
      group = colname) +
  geom_line(linewidth = 0.1) +
  geom_point(aes(fill = condition), size = 3, shape = 21) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1, vjus = 0.5),
        legend.position = "top") +
  labs(
       subtitle = protName,
       x = "Precursor",
       y = "log2(norm Intensity)") +
  theme_minimal()+
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1))
```

```
pNeedSum
```



1. We can see that data for a protein can consist of many precursors, hence the need for summarisation.
2. We observe that different precursors have different intensities within the same sample (same line). This is because different precursors have different properties and hence differ in their ionisation efficiency, which will influence the detectability in the MS (Challenges).

3. The precursor identifications are inconsistent between groups of interest. We can see the low-concentration group (condition 2, red) displays more missing values than the high-concentration group (condition 8, blue). Moreover, which value is missing depends on the precursor characteristics. All the precursors found in the low-concentration group are the most intense precursors in the high concentration group.
4. We also often find outliers, for instance, due to misidentification or fluctuations during MS acquisition.
5. These data also suggest pseudo-replication. The precursor intensities in one same sample (dots connected by a line) are correlated, i.e. they are more alike than the precursor intensities between samples (the lines do not perfectly align).

The fact that different precursors have different intensities (2.), may be inconsistently identified (3.) and/or quantified (4.), and show sample correlations (5.) can lead to bias if we use simple summarisation approaches such as summing or averaging the precursor intensities for each sample. Instead, we will resort to more advanced summarisation approaches to accommodate for these issues.

Here, we summarise the precursor-level data into protein intensities using maxLFQ. maxLFQ first calculates all pairwise log ratios between samples only using their shared precursors. Particularly, it uses the median of the log ratios between the shared precursors s, i.e.

$$r_{ij} = median(y_{sj} - y_{si})$$

.
Hence, it first eliminates peptide effects. It then estimates the summaries by solving

$$\sum_i \sum_j (y_j^{prot} - y_i^{prot} - r_{ij})^2$$

It is implemented in the `maxLFQ` function of the `iq` package.

`aggregateFeatures()` streamlines summarisation. It requires the name of a `rowData` column to group the precursors into proteins (or protein groups), here `Protein.Group`. We provide the summarisation approach through the `fun` argument. Other summarisation methods are available from the `MsCoreUtils` package, see `?aggregateFeatures` for a comprehensive list. The function will return a `QFeatures` object with a new set that we called `proteins`.

Click to see code

```
(qf <- aggregateFeatures(
  qf, i = "precursors_norm",
  name = "proteins",
  fcol = "Protein.Group",
#  fun = MsCoreUtils::medianPolish,
  fun = function(X, ...) iq::maxLFQ(X)$estimate,
```

```
  na.rm = TRUE
))
```

```
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```
An instance of class QFeatures (type: bulk) with 13 sets:

 [1] B000254_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA: SummarizedExperiment with 32848 rows
 [2] B000258_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio04_DIA: SummarizedExperiment with 33491 rows
 [3] B000262_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio08_DIA: SummarizedExperiment with 32406 rows
 ...
 [11] precursors_log: SummarizedExperiment with 35880 rows and 9 columns
 [12] precursors_norm: SummarizedExperiment with 35880 rows and 9 columns
 [13] proteins: SummarizedExperiment with 3187 rows and 9 columns
```

Note that all the links between precursors and proteins are kept[7]. This comes particularly handy when we want to extract all the data from one protein, P16083 for instance. This can be performed using the 3-way indexing. Every `QFeatures` object contains one or more sets, each characterised by multiple rows (precursors or proteins) and multiple columns (samples). Therefore, `QFeatures` can subset data based on one or more of these indices, `data[set_k, feature_i, sample_j]`. The first entry will subset particular features. This can be the name of a precursor (e.g. `KGMVAAWSQR2`) or the name of a protein (`UPS|P01008ups|ANT3_HUMAN_UPS`). The second entry selects the samples columns of interest. The third entry selects the sets of interest. If an entry is left blank, all the corresponding features, samples or sets will be selected. Let's extract all the data (all samples and all sets) related to the ups protein (`UPS|P01008ups|ANT3_HUMAN_UPS`).

```
qf["UPS|P01008ups|ANT3_HUMAN_UPS", , ]
```

```
An instance of class QFeatures (type: bulk) with 13 sets:

 [1] B000254_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio02_DIA: SummarizedExperiment with 5 rows and
 [2] B000258_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio04_DIA: SummarizedExperiment with 9 rows and
 [3] B000262_Ap_6883_EXT-765_DIA_Yeast_UPS2_ratio08_DIA: SummarizedExperiment with 11 rows an
 ...
```

[7] In fact, this is also true for the previous transformation where the precursors are linked across sets.

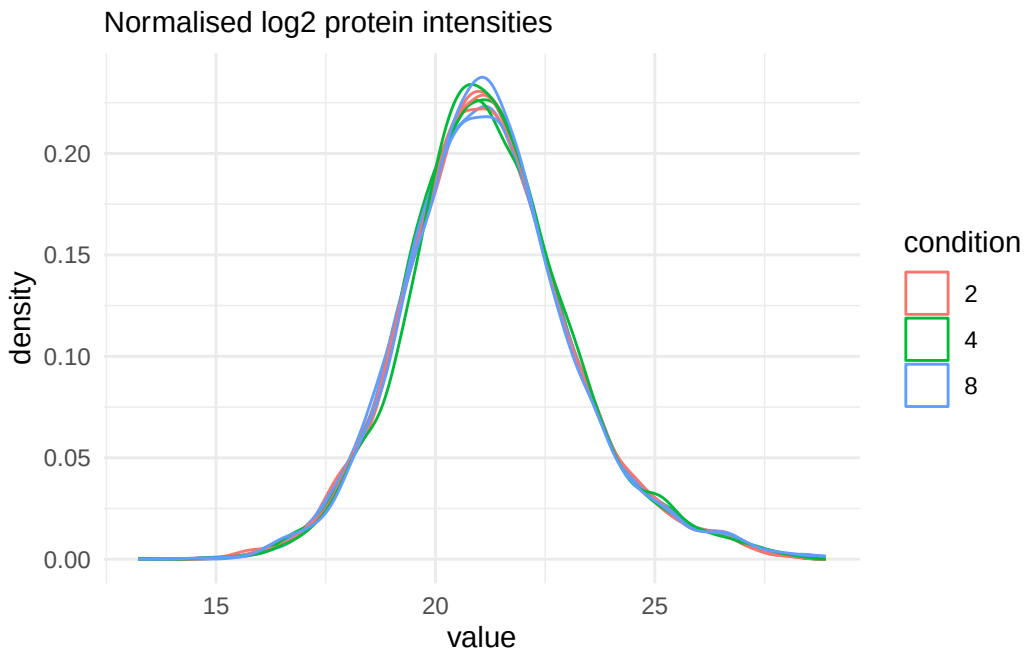
```
[11] precursors_log: SummarizedExperiment with 13 rows and 9 columns
[12] precursors_norm: SummarizedExperiment with 13 rows and 9 columns
[13] proteins: SummarizedExperiment with 1 rows and 9 columns
```

We finally evaluate the marginal distribution of the aggregated protein summaries.

Click to see code

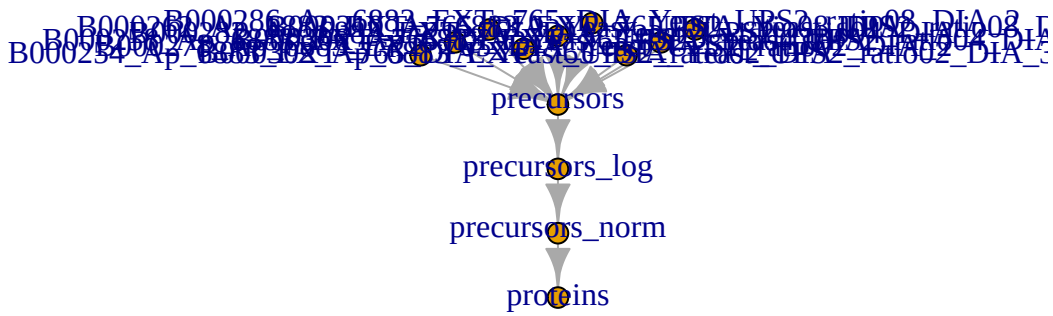
```
pAgg <- qf[, , "proteins"] |>
  longForm(colvars = colnames(colData(qf))) |>
  data.frame() |>
  filter(!is.na(value)) |>
  ggplot() +
  aes(x = value,
      colour = condition,
      group = colname) +
  geom_density() +
  theme_minimal() +
   labs(subtitle = "Normalised log2 protein intensities")
```

```
pAgg
```



The data processing is complete.

```
plot(qf)
```



Data exploration and QC

Data exploration aims to highlight the main sources of variation in the data prior to data modelling and can pinpoint to outlying or off-behaving samples.

Marginal distribution at precursor and protein level

Click to see code

```
pn1 <- qf[, , "precursors_norm"] |>
  longForm(colvars = colnames(colData(qf))) |>
  data.frame() |>
  filter(!is.na(value)) |>
  ggplot() +
  aes(x = value,
      colour = condition,
      group = colname) +
  geom_density() +
```

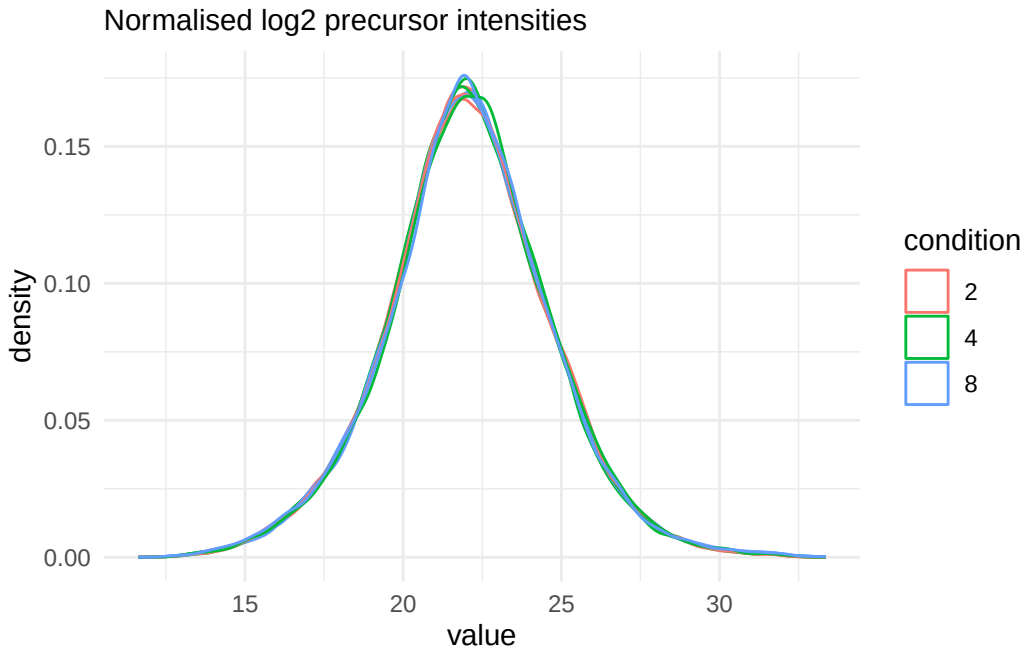
```
  theme_minimal() +
   labs(subtitle = "Normalised log2 precursor intensities")
```

```
pn2 <-qf[, , "precursors_norm"] |>
  longForm(colvars = colnames(colData(qf))) |>
  data.frame() |>
  filter(!is.na(value)) |>
  ggplot() +
  aes(x = sampleId,
      y = value,
      colour = condition,
      group = colname) +
  xlab("sample") +
  geom_boxplot() +
  theme_minimal() +
   labs(subtitle = "Normalised log2 precursor intensities")
```

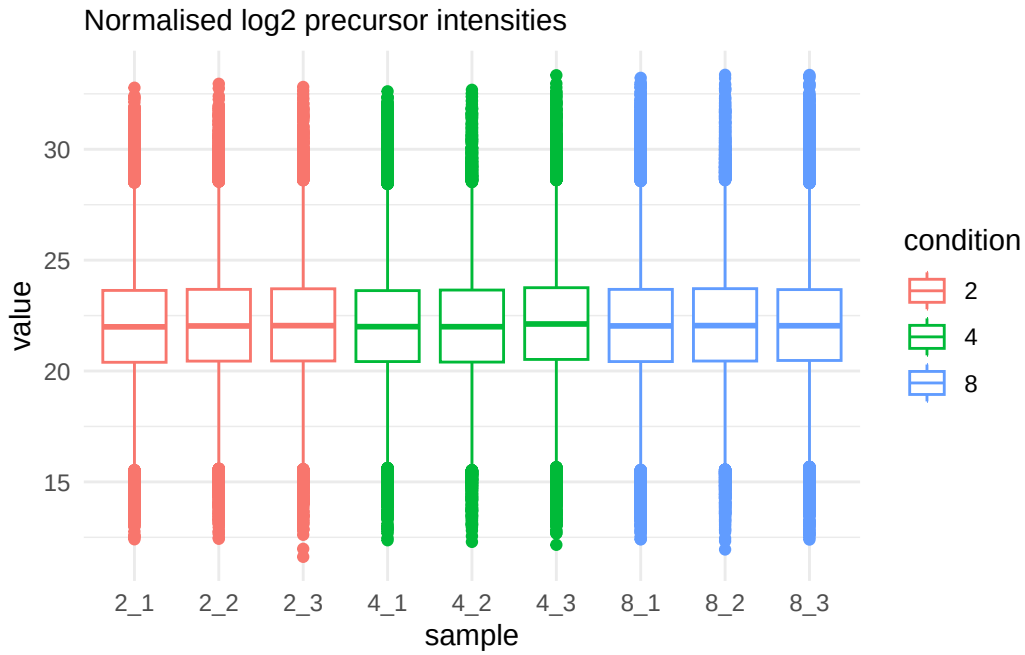
```
pn3 <- qf[, , "proteins"] |>
  longForm(colvars = colnames(colData(qf))) |>
  data.frame() |>
  filter(!is.na(value)) |>
  ggplot() +
  aes(x = sampleId,
      y = value,
      colour = condition,
      group = colname) +
  xlab("sample") +
  geom_boxplot() +
  theme_minimal() +
   labs(subtitle = "Normalised log2 protein intensities")
```

```
pn4 <- qf[, , "proteins"] |>
  longForm(colvars = colnames(colData(qf))) |>
  data.frame() |>
  filter(!is.na(value)) |>
  ggplot() +
  aes(x = value,
      colour = condition,
      group = colname) +
  geom_density() +
  theme_minimal() +
   labs(subtitle = "Normalised log2 protein intensities")
```

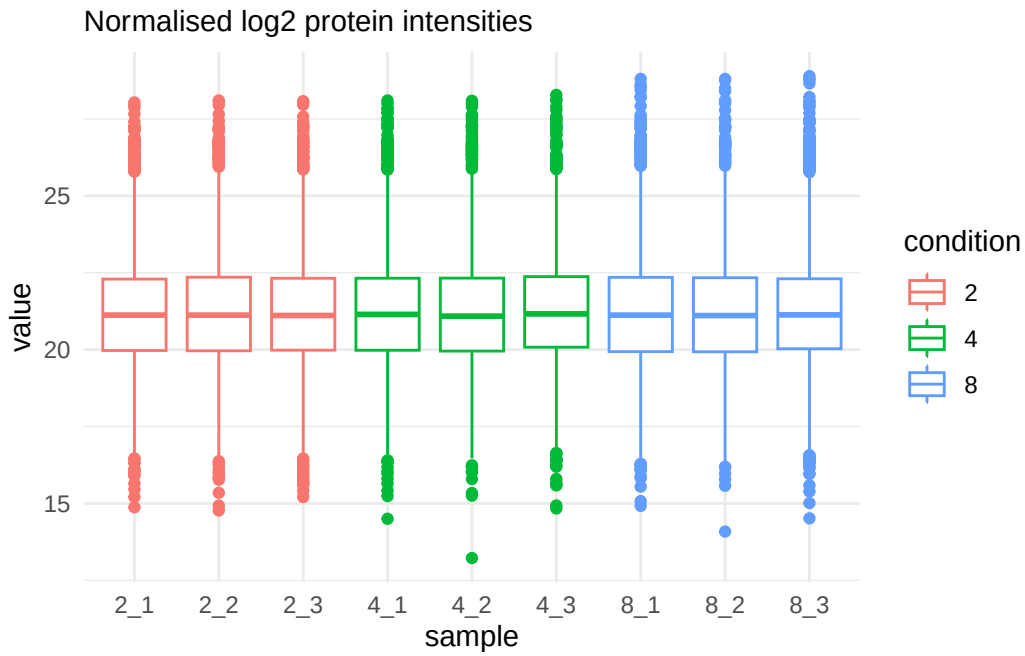
```
pn1
```


Normalised log2 precursor intensities
density
0.15
0.10
0.05
0.00
15
20
25
30
value
condition
2
4
8

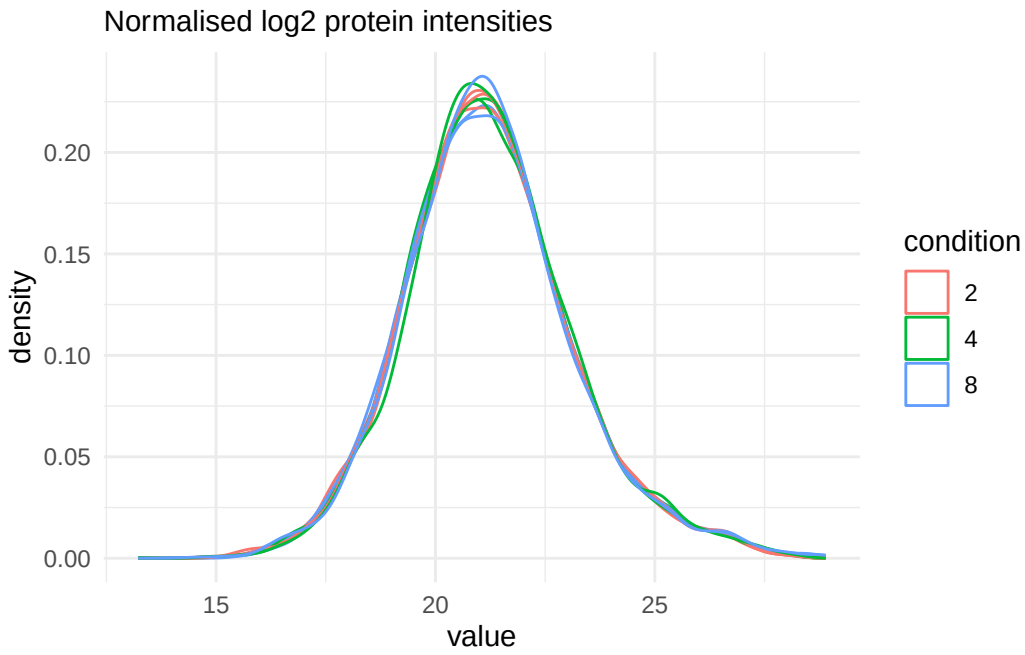
```
pn2
```


Normalised log2 precursor intensities
value
30
25
20
15
2_1
2_2
2_3
4_1
4_2
4_3
8_1
8_2
8_3
sample
condition
2
4
8

```
pn3
```


Normalised log2 protein intensities
value
25
20
15
2_1
2_2
2_3
4_1
4_2
4_3
8_1
8_2
8_3
sample
condition
2
4
8

```
pn4
```

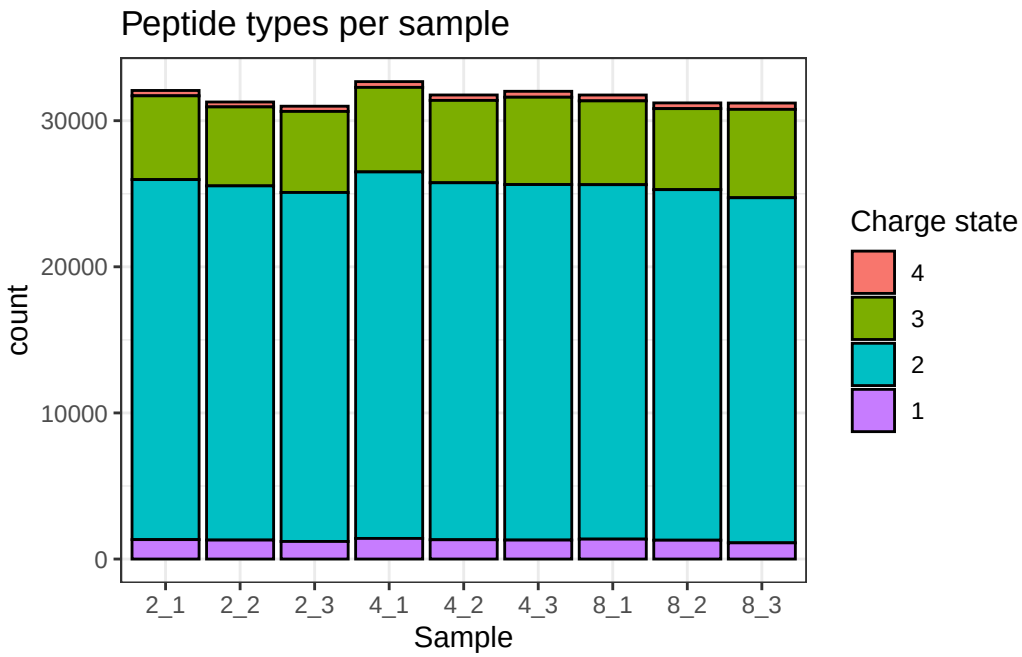


Charge state

Click to see code

```
pcs <- qf[, , "precursors_norm"] |>
  longForm(colvars = colnames(colData(qf)), rowvars = "Precursor.Charge") |>
  as.data.frame() |>
  filter(!is.na(value)) |>
  filter(Precursor.Charge<=4) |>
ggplot(aes(x = sampleId)) +
  geom_bar(aes(fill = factor(Precursor.Charge, levels = 4:1)),
           colour = "black") +
  labs(title = "Peptide types per sample",
       x = "Sample",
       fill = "Charge state") +
  theme_bw()
```

```
pcs
```



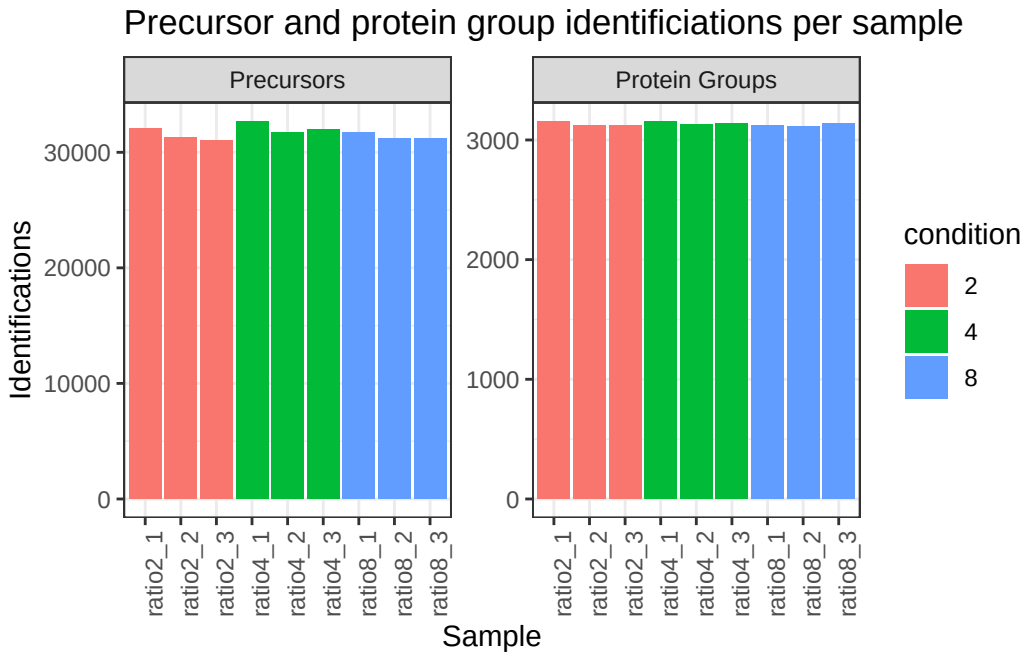
Identifications per sample

Click to see code

```
pId <- qf[,,"precursors_norm"] |>
  longForm(colvars = colnames(colData(qf)),
           rowvars= c("Precursor.Id",
                      "Protein.Group",
                      "Stripped.Sequence",
                      "Precursor.Charge")) |>
  data.frame() |>
  filter(!is.na(value)) |>
  mutate(cond_rep = paste0("ratio", condition, "_", rep)) |>
  group_by(condition, cond_rep) |>
  summarise(Precursors = length(unique(Precursor.Id)),
            `Protein Groups` = length(unique(Protein.Group))) |>
  pivot_longer(-(1:2),
               names_to = "Feature",
               values_to = "IDs") |>
  ggplot(aes(x = cond_rep, y = IDs, fill = condition)) +
  geom_col() +
  #scale_fill_observable() +
```

```
  facet_wrap(~Feature,
             scales = "free_y") +
  labs(title = "Precursor and protein group identificiations per sample",
       x = "Sample",
       y = "Identifications") +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 90))
# Data Modeling (Robust Regression){#sec-modelling}
```

```
pId
```



Dimensionality reduction plot

A common approach for data exploration is to perform dimension reduction, such as Multi Dimensional Scaling (MDS).

Click to see code

We will first extract the set to explore along the sample annotations (used for plot colouring).

```
se <- getWithColData(qf, "proteins")
```

We then use the `scater` package to compute and plot the PCA. For technical reasons, it requires `SingleCellExperiment` class object, but these can easily be generated from a `SummarizedExperiment` object.

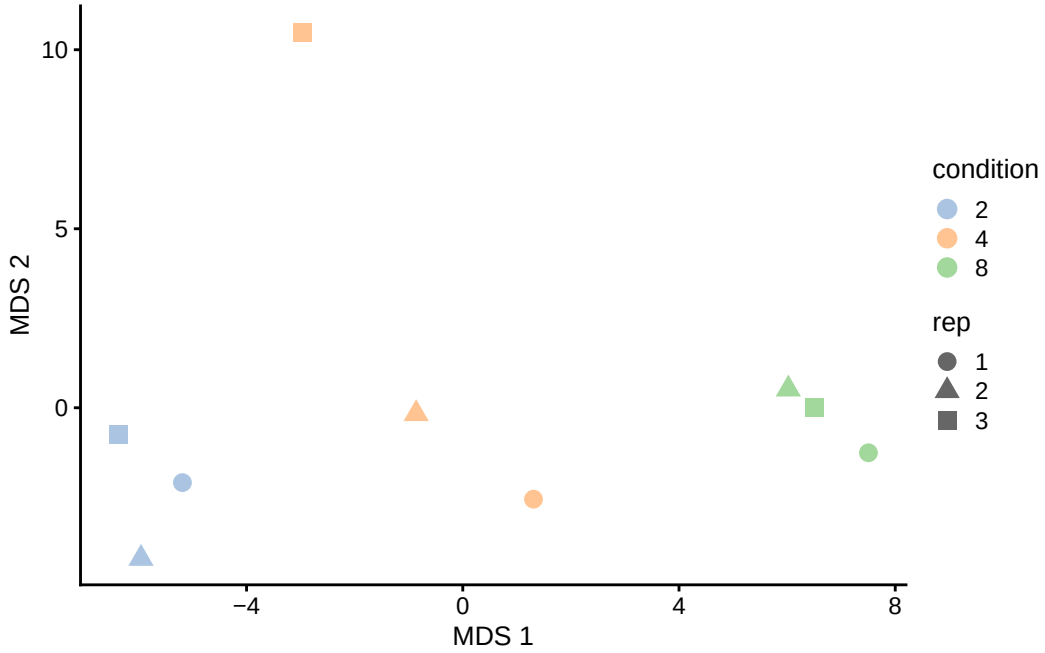
```
se <- runMDS(as(se, "SingleCellExperiment"), exprs_values = 1)
```

We can now explore the data structure while coloring for the factor of interest, here `condition`.

```
pMds <- plotMDS(se, colour_by = "condition", shape_by = "rep", point_size = 3)
```

This plot reveals interesting information. First, we see that the samples are nicely separated according to their spike-in condition. Interestingly, the conditions are sorted by the concentration.

```
pMds
```

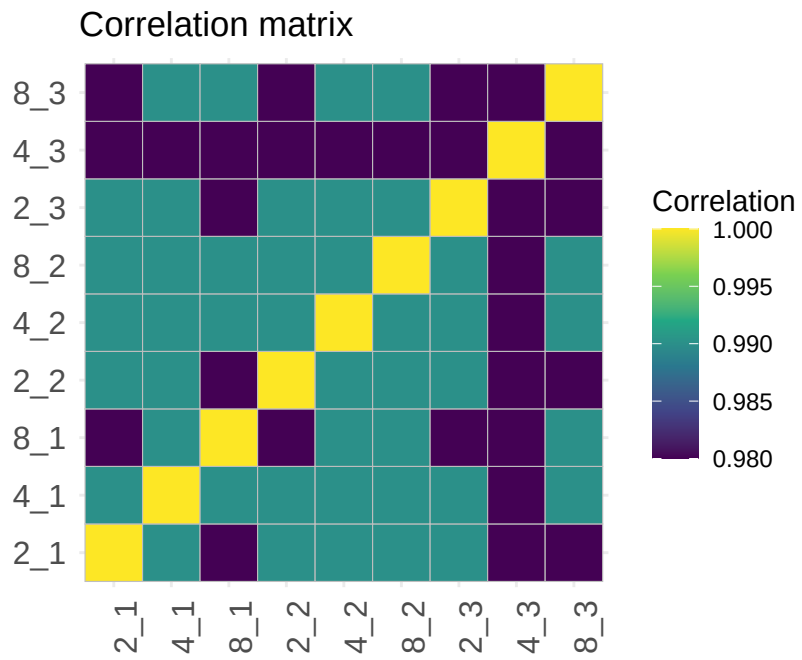


Correlation matrix

Click to see code

```
corMat <- qf[["proteins"]] |>
  assay() |>
  cor(method = "spearman", use = "pairwise.complete.obs")

colnames(corMat) <- qf$sampleId
rownames(corMat) <- qf$sampleId
pCor <- corMat |>
  ggcorrplot::ggcorrplot() +
  scale_fill_viridis_c() +
  labs(title = "Correlation matrix",
       fill = "Correlation") +
  theme(axis.text.x = element_text(angle = 90))
```

```
pCor
```



Data Modeling (Robust Regression)

We model the preprocessed data to answer biologically relevant questions. As described above, each sample has been created to contain a constant yeast background into which human UPS standard proteins were spiked in at three different concentrations (yeast:UPS ratios 2, 4 and 8). Each sample preparation has been replicated three times. So, the key question is "can our

model retrieve the proteins that have been spiked in across the conditions?" Moreover, since we know the expected protein abundance in each sample, we will also be able to assess how accurate the model can estimate the average fold changes between spike-in conditions.

Sources of variation

In this experimental design, we can model a single source of variation: spike-in concentration[8]. All remaining sources of variability, e.g. pipetting errors, MS run to run variability, etc will be lumped in the residual variability (error term).

Spike-in condition effects: we model the source of variation induced by the experimental treatment of interest as a fixed effect, which we consider non-random, i.e. the treatment effect is assumed to be the same in repeated experiments, but it is unknown and has to be estimated.

When modelling a typical label-free experiment at the protein level, the model boils down to the following linear model:

$$y_i = \mathbf{x}_i^T \beta + \epsilon_i$$

with y_i the $\log_2$-normalised and summarised protein intensities in sample i; $\mathbf{x}_i$ a vector with the covariate pattern for the sample encoding the intercept, spike-in condition[9], or other experimental factors; β the vector of parameters that model the association between the covariates and the outcome; and ϵ_i the residuals reflecting variation that is not captured by the fixed effects. Note that $\mathbf{x}_i$ allows for a flexible parameterisation of the treatment beyond a single covariate, i.e. including a 1 for the intercept, continuous and categorical variables as well as their interactions. We assume the residuals to be independent and to follow a normal distribution with zero mean and variance that can differ from protein to protein, i.e. $\epsilon_i \sim N(0, \sigma^2_{\epsilon_i})$.

In R, this model is encoded using the following simple formula[10]:

```
model <- ~ condition
```

[8] Note that we have the spike-in concentration encoded in two ways. `ratio` is encoded as a numerical value and provides the actual ratio for UPS on 10 for yeast. `condition` is encoded as a factor, meaning that every concentration is regarded as an independent group. We will use the latter encoding as most common research questions consist of group-wise comparisons. The modelling of concentration as a numeric will be discussed in a dedicated section

[9] Categorical variables are encoded using dummy coding.

[10] We don't need to specify the intercept as R automatically adds it during model estimation. you can explicitly add it using `~ 1 + ...`, you can also suppress it using `~ 0 +`. When an intercept is included, one of the groups is defined as the reference group and its corresponding model parameter is absorbed in the intercept. The residuals don't need to be specified as they are part of the model estimation process.

Model estimation

We estimate the model with `msqrob()`.

Click to see code

The function takes the `QFeatures` object, extracts the quantitative values from the `"proteins"` set generated during summarisation, and fits a simple linear model with `condition` as covariate, which are automatically retrieved from `colData(qf)`.

```
qf <- msqrob(
  qf,  i = "proteins",
  formula = model,
  robust = TRUE
)
```

We enable M-estimation (`robust = TRUE`) for improved robustness against outliers.

The fitting results are available in the `msqrobModels` column of the `rowData`. More specifically, the modelling output is stored in the `rowData` as a `statModel` object, one model per row (protein). We will see in a later section how to perform statistical inference on the estimated parameters.

```
models <- rowData(qf[["proteins"]])[["msqrobModels"]]
models[1:3]
```

```
$D6VTK4
Object of class "StatModel"
The type is "rlm"
There number of elements is 7

$O13297
Object of class "StatModel"
The type is "rlm"
There number of elements is 7

$O13329
Object of class "StatModel"
The type is "fitError"
There number of elements is 7
```

Statistical inference

We can now convert the research question "do the spike-in conditions affect the protein intensities?" into a statistical hypothesis.

In other words, we need to translate this question in a null and alternative hypothesis on a single model parameter or a linear combination of model parameters, which is also referred to with a contrast.

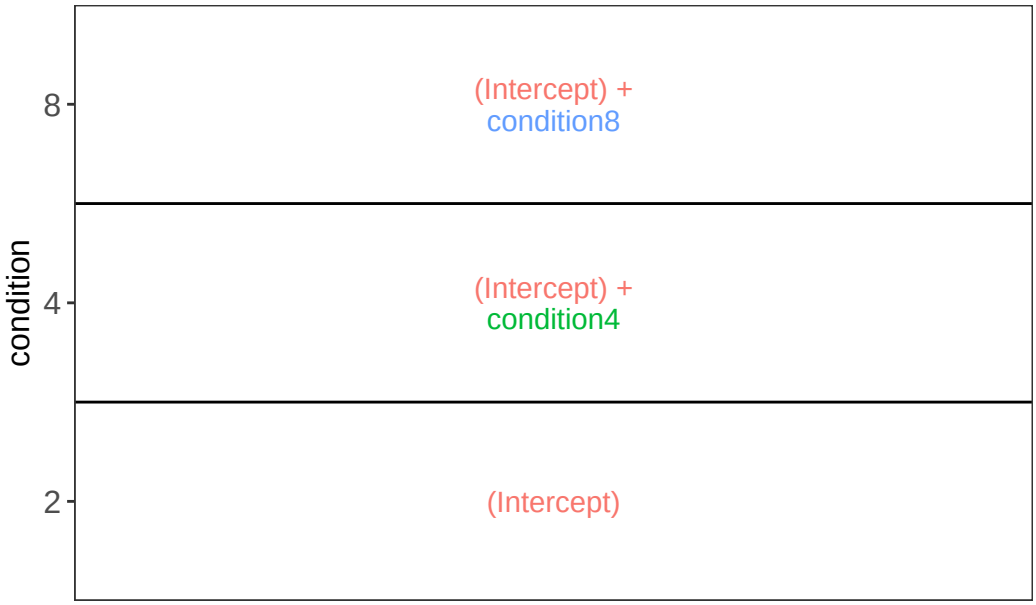
To aid defining contrasts, we will visualise the experimental design using the `ExploreModelMatrix` package.

Click to see code

```
vd <- ExploreModelMatrix::VisualizeDesign(
    sampleData =  colData(qf),
    designFormula = model,
    textSizeFitted = 4
)
```

```
vd$plotlist
```

```
[[1]]
```



This plot shows that the average protein $\log_2$ intensity for condition2 is modelled by `(Intercept)`, for the `condition4` group by `(Intercept) + condtion4` and for the `condition8` group by `(Intercept) + condtion8`.

Hence, to assess differential abundance of proteins between spike-in conditions, we will have to assess all pairwise combinations between the spike-in groups, which leads to 3 different contrasts:

- the log2 fold change between ratio 4 and ratio 2, $\log_2 FC_{4-2} = (Intercept + condition4) - Intercept = condition4$.
- the log2 fold change between ratio 8 and ratio 2, $\log_2 FC_{8-2} = (Intercept + condition8) - Intercept = condition8$.
- the log2 fold change between ratio 8 and ratio 4, $\log_2 FC_{8-4} = (Intercept + condition8) - (Intercept + condition4) = condition8 - condition4$.

Hypothesis testing

As shown above, the average difference in log2 intensity between condition 4 and 2 is `condition4`. This combination of parameters is also called a contrast. Thus, we assess the following null hypothesis for this contrast: `condition4 = 0` with our statistical test.

```
contrast <- "condition4 = 0"
```

Implementation details

`makeContrast()` converts the hypothesis into a formal contrast matrix with parameter names as rows and hypotheses in columns[11]. The function `makeContrast()` also has an argument `parameterNames` in which we have to provide all parameter names involved in the contrast. Since we already used the `VisualizeDesign` we have a design matrix with all parameter names in the columns.

```
(L <- makeContrast(
  contrast,
   parameterNames = colnames(vd$designmatrix)
))
```

[11] Note, that we only need to specify the names of the model parameters that are involved in the contrast when using the make contrast function. The hypothesisTest function below will then subset the relevant part from the model output. Also, note that the contrast matrix in this example is trivial. However, we will later show how to asses multiple hypotheses at once and then the contrast matrix consists of multiple columns, with one column for each contrast

```
            condition4
(Intercept)          0
condition4           1
condition8           0
```

We can now test our null hypothesis using `hypothesisTest()` which takes the `QFeatures` object with the fitted model and the contrast we just built. `msqrob2` automatically applies the hypothesis testing to all proteins in the data.

```
qf <- hypothesisTest(qf, i = "proteins", contrast = L)
```

The results are stored in the set containing the model, here `proteins`. We retrieve the results from the `rowData`. Note, that for this spike-in study we know the ground truth, so we also add variable `isUPS` to indicate if the protein is spiked (UPS).

```
inference <- rowData(qf[["proteins"]])$condition4
head(inference)
```

```
             logFC         se       df          t      pval   adjPval
D6VTK4 -0.23798967 0.14067418 8.075799 -1.6917793 0.1287946 0.9699605
O13297  0.05834120 0.08247122 7.276175  0.7074129 0.5013337 0.9937709
O13329  0.00000000         NA       NA         NA        NA        NA
O13539  0.04744902 0.16400341 8.252461  0.2893173 0.7794743 0.9937709
O13563  0.17345681 0.15034445 8.295944  1.1537294 0.2807650 0.9937709
O13585 -0.09176139 0.13394584 7.508295 -0.6850634 0.5138917 0.9937709
```

However, it is easier to use the novel getter function from msqrob2 `msqrobCollect`

```
inference <- msqrobCollect(qf[["proteins"]], L)
head(inference)
```

```
                        logFC         se       df          t      pval
condition4.D6VTK4 -0.23798967 0.14067418 8.075799 -1.6917793 0.1287946
condition4.O13297  0.05834120 0.08247122 7.276175  0.7074129 0.5013337
condition4.O13329  0.00000000         NA       NA         NA        NA
condition4.O13539  0.04744902 0.16400341 8.252461  0.2893173 0.7794743
condition4.O13563  0.17345681 0.15034445 8.295944  1.1537294 0.2807650
condition4.O13585 -0.09176139 0.13394584 7.508295 -0.6850634 0.5138917
                     adjPval   contrast feature
condition4.D6VTK4 0.9699605 condition4  D6VTK4
```

```
condition4.O13297 0.9937709 condition4  O13297
condition4.O13329        NA condition4  O13329
condition4.O13539 0.9937709 condition4  O13539
condition4.O13563 0.9937709 condition4  O13563
condition4.O13585 0.9937709 condition4  O13585
```

```
head(inference)
```

```
                      logFC         se       df          t      pval
condition4.D6VTK4 -0.23798967 0.14067418 8.075799 -1.6917793 0.1287946
condition4.O13297  0.05834120 0.08247122 7.276175  0.7074129 0.5013337
condition4.O13329  0.00000000         NA       NA         NA        NA
condition4.O13539  0.04744902 0.16400341 8.252461  0.2893173 0.7794743
condition4.O13563  0.17345681 0.15034445 8.295944  1.1537294 0.2807650
condition4.O13585 -0.09176139 0.13394584 7.508295 -0.6850634 0.5138917
                    adjPval   contrast feature
condition4.D6VTK4 0.9699605 condition4  D6VTK4
condition4.O13297 0.9937709 condition4  O13297
condition4.O13329        NA condition4  O13329
condition4.O13539 0.9937709 condition4  O13539
condition4.O13563 0.9937709 condition4  O13563
condition4.O13585 0.9937709 condition4  O13585
```

Note, that missing values can occur because data modelling resulted in a `fitError` (the advanced vignette describes how to deal with proteins that could not be fit).

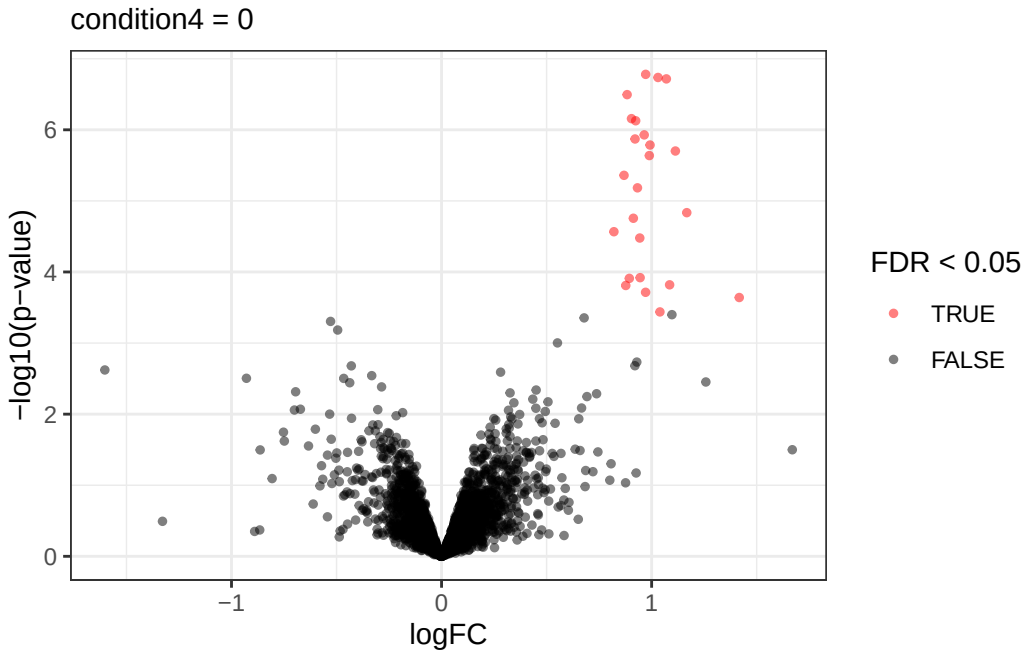
The results also include an adjusted p-value for each protein j to correct for multiple testing using the Benjamini-Hochberg False Discovery Rate (FDR) method, which represents an estimate of the minimum FDR at which the test result for protein j is considered significant, i.e. the expected fraction of false positives that are reported in the shortest top list of the most significant proteins that include protein j.

Volcano plots

A volcano plot is a common visualisation that provides an overview of the hypothesis testing results, plotting the $-\log_{10}$ p-value[12] as a function of the estimated log fold change. Volcano plots can be used to highlight the most interesting proteins that have large fold changes and/or are highly significant. We can use the table above directly to build a volcano plot using `ggplot2` functionality. We also highlight which proteins are UPS standards, known to be differentially abundant by experimental design.

[12] Note, that upon this transformation a value of 1 represents a p-value of 0.1, 2 a p-value of 0.01, 3 a p-value of 0.001, etc.

```
alpha <- 0.05
volcano <- plotVolcano(inference,significanceLevel = alpha) +
  labs(subtitle = paste0(colnames(L), " = 0"))
volcano
```



Note, that the log fold changes are nicely around the real log2 fold change: $\log_2(4/2) = 1$

```
signif <- inference |>
  filter(adjPval < alpha) |>
  arrange(pval)
knitr::kable(signif)
```

	logFC	se	df	t	pval	adjPval	contrast	feature
condition4.UPS|P02768ups|ALBU_HUMAN_UPS	0.971664	0.0486005	19.92426	16.17835	5.000000e-13	2.000020e-10	condition4	UPS|P02768ups|ALBU_HUMAN_UPS
condition4.UPS|P63165ups|SUMO1_HUMAN_UPS	1.030404	0.0761937	19.95230	16.63820	2.000000e-13	2.000020e-10	condition4	UPS|P63165ups|SUMO1_HUMAN_UPS

	logFC	se	df	t	pval	adjP-val	contrast	feature
condition4.UPS|P06732ups|KCRM_HUMAN_UPS	1.07012	0.0662858	58.9938	16.1441	5.80000e-23	2.00020e-16	condition4	UPS|P06732ups|KCRM_HUMAN_UPS
condition4.UPS|Q06830ups|PRDX1_HUMAN_UPS	0.883337	0.0600184	58.9694	14.7186	5.80000e-21	3.00025e-18	condition4	UPS|Q06830ups|PRDX1_HUMAN_UPS
condition4.UPS|P04040ups|CATA_HUMAN_UPS	0.904162	0.0679189	58.8891	13.3123	4.20000e-19	7.00039e-15	condition4	UPS|P04040ups|CATA_HUMAN_UPS
condition4.UPS|P01008ups|ANT3_HUMAN_UPS	0.924083	0.0704985	58.4373	13.1084	1.90000e-18	7.00039e-15	condition4	UPS|P01008ups|ANT3_HUMAN_UPS
condition4.UPS|P15559ups|NQO1_HUMAN_UPS	0.964678	0.0768785	58.1235	12.5486	4.10000e-18	1.20052e-14	condition4	UPS|P15559ups|NQO1_HUMAN_UPS
condition4.UPS|P00918ups|CAH2_HUMAN_UPS	0.921052	0.0767867	57.5952	12.0057	3.60000e-17	1.30052e-14	condition4	UPS|P00918ups|CAH2_HUMAN_UPS
condition4.UPS|P12081ups|SYHC_HUMAN_UPS	0.992435	0.0728224	58.4067	13.6281	5.60000e-19	1.60057e-15	condition4	UPS|P12081ups|SYHC_HUMAN_UPS
condition4.UPS|P02753ups|RETBP_HUMAN_UPS	1.11308	0.0886237	57.7532	12.5597	3.50000e-18	2.00062e-14	condition4	UPS|P02753ups|RETBP_HUMAN_UPS
condition4.UPS|P62937ups|PPIA_HUMAN_UPS	0.989126	0.0686662	58.8308	14.4055	7.10000e-21	2.00065e-17	condition4	UPS|P62937ups|PPIA_HUMAN_UPS
condition4.UPS|P00915ups|CAH1_HUMAN_UPS	0.868547	0.0824185	58.2917	10.5386	2.60000e-14	4.00114e-11	condition4	UPS|P00915ups|CAH1_HUMAN_UPS
condition4.UPS|P02144ups|MYG_HUMAN_UPS	0.932594	0.0841375	58.4984	11.0841	8.10000e-16	6.00158e-12	condition4	UPS|P02144ups|MYG_HUMAN_UPS
condition4.UPS|O76070ups|SYUG_HUMAN_UPS	1.16735	0.0982962	58.3280	11.8765	6.00001e-17	4.60329e-13	condition4	UPS|O76070ups|SYUG_HUMAN_UPS
condition4.UPS|P69905ups|HBA_HUMAN_UPS	0.913046	0.102058	54.1233	8.94657	6.00001e-12	7.60036e-09	condition4	UPS|P69905ups|HBA_HUMAN_UPS

	logFC	se	df	t	pval	adjP-val	contrast	feature
condition4.UPS|P62988ups|UBIQ_HUMAN_UPS	0.820432	0.091107	8.528329	9.004971	1.000027e-05	0.0053378	condition4	UPS|P62988ups|UBIQ_HUMAN_UPS
condition4.UPS|P63279ups|UBC9_HUMAN_UPS	0.943580	0.106807	7.940748	8.834537	3.000033e-05	0.0061631	condition4	UPS|P63279ups|UBC9_HUMAN_UPS
condition4.UPS|P01031ups|CO5_HUMAN_UPS	0.945226	0.138338	6.771768	6.832799	0.000120	0.0203698	condition4	UPS|P01031ups|CO5_HUMAN_UPS
condition4.UPS|P68871ups|HBB_HUMAN_UPS	0.893816	0.132898	6.733738	6.725498	0.000123	0.0203698	condition4	UPS|P68871ups|HBB_HUMAN_UPS
condition4.UPS|Q15843ups|NEDD8_HUMAN_UPS	1.085628	0.158757	7.172946	6.838298	0.000150	0.0232038	condition4	UPS|Q15843ups|NEDD8_HUMAN_UPS
condition4.UPS|P08263ups|GSTA1_HUMAN_UPS	0.876936	0.126197	7.756056	6.949295	0.000154	0.0232038	condition4	UPS|P08263ups|GSTA1_HUMAN_UPS
condition4.UPS|P00167ups|CYB5_HUMAN_UPS	0.971253	0.145357	8.308767	6.732431	0.000193	0.0276263	condition4	UPS|P00167ups|CYB5_HUMAN_UPS
condition4.UPS|P55957ups|BID_HUMAN_UPS	1.417080	0.211987	7.133738	6.684968	0.000228	0.0312842	condition4	UPS|P55957ups|BID_HUMAN_UPS
condition4.UPS|P41159ups|LEP_HUMAN_UPS	1.039466	0.173712	6.794137	5.983829	0.000364	0.0478010	condition4	UPS|P41159ups|LEP_HUMAN_UPS

Note, that we return many true positives and one false positive.

Heatmap

We can explore the quantitative data of the significant proteins using a heatmap. First, we select the names of the proteins that were declared significant between condition 4 and condition 2 and extract their quantitative data.

Click to see code

```
sig <- inference |>
  filter(adjPval < alpha) |>
  arrange(pval) |>
  pull(feature) #1.
```

Next, we extract the quantitative data and scale by rows[13] with `assay()`. We will create a heatmap using the `ComplexHeatmap` package, which enables heatmap annotations. We will annotate the heatmap using our model variable `condition`.

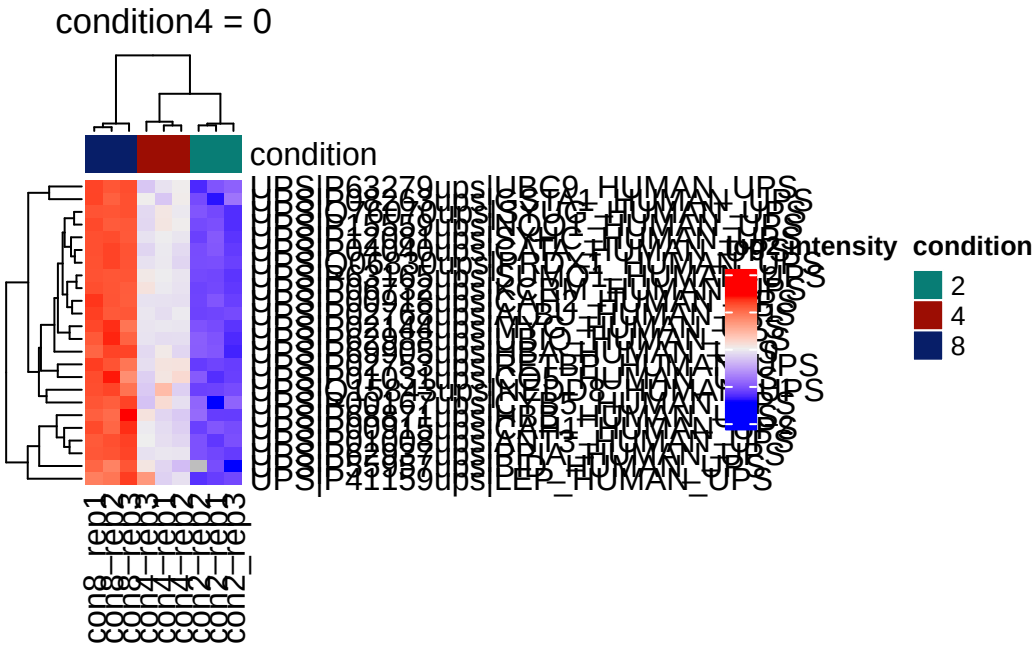
```
se <- getWithColData(qf, "proteins")
quants <- t(scale(t(assay(se[sig,])))) #2.

colnames(quants) <- paste0("con", se$condition,"_rep",se$rep) #specific to this dataset to g
annotations <- columnAnnotation(
  condition = se$condition
) #3.
```

We now make the heatmap.

```
set.seed(1234) ## annotation colours are randomly generated by default
heatmap <- Heatmap(
                       quants,
                       name = "log2 intensity",
                       top_annotation = annotations,
                       column_title = contrast
                       ) #4.
```

```
heatmap
```

[13]The `scales()` function aligns the means and the centered standard deviations by column. Since we want to scale by row, we need to transpose (`t()`) before scaling and transpose back after scaling. Scaling the data will remove changes in feature intensities that are caused by uninteresting changes in ionisation efficiency.



All contrasts

With the createPairwiseContrast function we can make all pairwise contrasts for a model.

Click to see code

```
(allHypotheses <- createPairwiseContrasts(
  model, colData(qf), "condition"
  )
)
```

```
[1] "condition4 = 0"              "condition8 = 0"
[3] "condition8 - condition4 = 0"
```

```
(L <- makeContrast(
  allHypotheses,
  parameterNames = colnames(vd$designmatrix)
))
```

```
            condition4 condition8 condition8 - condition4
(Intercept)          0          0                      0
```

```
condition4           1          0                     -1
condition8           0          1                      1
```

```
qf <- hypothesisTest(qf,
                                i = "proteins",
                                contrast = L,
                                overwrite = TRUE)
```

List with significant differentially abundant proteins

```
inferences <- msqrobCollect(qf[["proteins"]], L)
```

```
for (j in colnames(L)) #3.
{
  inference <- inferences |> filter(adjPval < alpha & contrast == j)
  cat("**Contrast:**", j, "= 0 (", nrow(inference |> filter(adjPval < alpha)), "significant p
  print(kable(inference |> arrange(pval) |> relocate("feature"), row.names = FALSE)) #9.
  cat("\n\n\n---\n\n") #10.
}
```

Contrast: condition4 = 0 (24 significant proteins)

feature	logFC	se	df	t	pval	adjP-val	con-trast
UPS|P02768ups|ALBU_HU-MAN_UPS	0.9716648	0.0600596	8.192420	16.178350	0.0000002	0.0002016	condi-tion4
UPS|P63165ups|SUMO1_HU-MAN_UPS	1.0304047	0.0619300	7.952301	16.638209	0.0000002	0.0002016	condi-tion4
UPS|P06732ups|KCRM_HU-MAN_UPS	1.0701290	0.0662858	8.093870	16.144158	0.0000002	0.0002016	condi-tion4
UPS|Q06830ups|PRDX1_HU-MAN_UPS	0.8833372	0.0600148	8.269417	14.718658	0.0000003	0.0002518	condi-tion4
UPS|P04040ups|CATA_HU-MAN_UPS	0.9041624	0.0679191	8.288912	13.312342	0.0000007	0.0003915	condi-tion4
UPS|P01008ups|ANT3_HU-MAN_UPS	0.9240834	0.0704954	8.337388	13.108419	0.0000007	0.0003915	condi-tion4
UPS|P15559ups|NQO1_HU-MAN_UPS	0.9646786	0.0768751	8.235205	12.548641	0.0000012	0.0005295	condi-tion4

feature	logFC	se	df	t	pval	adjP-val	contrast
UPS|P00918ups|CAH2_HUMAN_UPS	0.9210527	0.0678675	7.595280	13.571336	0.0000013	0.0005295	condition4
UPS|P12081ups|SYHC_HUMAN_UPS	0.9924350	0.0728224	7.406783	13.628156	0.0000016	0.0005728	condition4
UPS|P02753ups|RETBP_HUMAN_UPS	1.1130812	0.0886232	7.753222	12.559705	0.0000020	0.0006254	condition4
UPS|P62937ups|PPIA_HUMAN_UPS	0.9891264	0.0686628	6.830835	14.405571	0.0000023	0.0006593	condition4
UPS|P00915ups|CAH1_HUMAN_UPS	0.8685470	0.0824156	8.291745	10.538626	0.0000044	0.0011457	condition4
UPS|P02144ups|MYG_HUMAN_UPS	0.9325948	0.0841375	7.484986	11.084181	0.0000066	0.0015862	condition4
UPS|O76070ups|SYUG_HUMAN_UPS	1.1673500	0.0982902	6.328066	11.876560	0.0000146	0.0032918	condition4
UPS|P69905ups|HBA_HUMAN_UPS	0.9130468	0.1020554	8.123395	8.946576	0.0000176	0.0036825	condition4
UPS|P62988ups|UBIQ_HUMAN_UPS	0.8204321	0.0911088	7.528321	9.004971	0.0000271	0.0053378	condition4
UPS|P63279ups|UBC9_HUMAN_UPS	0.9435803	0.1068059	7.440741	8.834537	0.0000333	0.0061631	condition4
UPS|P01031ups|CO5_HUMAN_UPS	0.9452268	0.1383367	8.177184	6.832799	0.0001203	0.0203698	condition4
UPS|P68871ups|HBB_HUMAN_UPS	0.8938152	0.1328995	8.337388	6.725498	0.0001230	0.0203698	condition4
UPS|Q15843ups|NEDD8_HUMAN_UPS	1.0856282	0.1587571	7.772949	6.838298	0.0001516	0.0232038	condition4
UPS|P08263ups|GSTA1_HUMAN_UPS	0.8769365	0.1261907	7.560563	6.949295	0.0001548	0.0232038	condition4
UPS|P00167ups|CYB5_HUMAN_UPS	0.9712534	0.1535745	8.308707	6.324313	0.0001931	0.0276263	condition4
UPS|P55957ups|BID_HUMAN_UPS	1.4170803	0.2119801	7.337388	6.684968	0.0002286	0.0312842	condition4
UPS|P41159ups|LEP_HUMAN_UPS	1.0394662	0.1737126	7.794137	5.983829	0.0003645	0.0478010	condition4

Contrast: condition8 = 0 (29 significant proteins)

feature	logFC	se	df	t	pval	adjP-val	contrast
UPS|P02768ups|ALBU_HUMAN_UPS	2.1127824	0.0608172	8.1924203	34.7398880	0.0000000	0.0000004	condition8
UPS|Q06830ups|PRDX1_HUMAN_UPS	2.0219179	0.0596700	8.2694173	33.8850180	0.0000000	0.0000004	condition8
UPS|P06732ups|KCRM_HUMAN_UPS	2.1677156	0.0648686	8.0938703	33.4170070	0.0000000	0.0000004	condition8
UPS|P63165ups|SUMO1_HUMAN_UPS	2.0977684	0.0613926	7.9523013	34.1697230	0.0000000	0.0000004	condition8
UPS|P04040ups|CATA_HUMAN_UPS	2.1157477	0.0679191	8.2889123	31.1509940	0.0000000	0.0000004	condition8
UPS|P01008ups|ANT3_HUMAN_UPS	2.0869380	0.0704954	8.3373883	29.6038840	0.0000000	0.0000005	condition8
UPS|P15559ups|NQO1_HUMAN_UPS	1.9749715	0.0763916	8.2352052	25.8532570	0.0000000	0.0000016	condition8
UPS|P00915ups|CAH1_HUMAN_UPS	2.0563121	0.0827333	8.2917452	24.8547160	0.0000000	0.0000017	condition8
UPS|P00918ups|CAH2_HUMAN_UPS	2.0641036	0.0716202	7.5952802	28.8201340	0.0000000	0.0000017	condition8
UPS|P12081ups|SYHC_HUMAN_UPS	2.1120359	0.0728224	7.4067832	29.0025610	0.0000000	0.0000021	condition8
UPS|P62937ups|PPIA_HUMAN_UPS	2.1646410	0.0718392	6.8308353	30.1317490	0.0000000	0.0000046	condition8
UPS|P02753ups|RETBP_HUMAN_UPS	2.1479744	0.0923251	7.7532222	23.2653290	0.0000000	0.0000049	condition8
UPS|P69905ups|HBA_HUMAN_UPS	2.0686241	0.1020554	8.1233952	20.2696110	0.0000000	0.0000073	condition8
UPS|P02144ups|MYG_HUMAN_UPS	2.0368937	0.0920377	7.4849862	22.1310730	0.0000000	0.0000096	condition8
UPS|P63279ups|UBC9_HUMAN_UPS	2.2123582	0.1043718	7.4407412	21.1969000	0.0000001	0.0000133	condition8
UPS|P62988ups|UBIQ_HUMAN_UPS	1.8870767	0.0937624	7.5283212	20.1261510	0.0000001	0.0000157	condition8
UPS|O76070ups|SYUG_HUMAN_UPS	2.4734976	0.0908469	6.3280662	27.2271120	0.0000001	0.0000157	condition8
UPS|P68871ups|HBB_HUMAN_UPS	2.0749401	0.1328995	8.3373883	15.6128550	0.0000002	0.0000320	condition8
UPS|Q15843ups|NEDD8_HUMAN_UPS	2.3629055	0.1502884	7.7729491	15.7224750	0.0000004	0.0000596	condition8
UPS|P00167ups|CYB5_HUMAN_UPS	2.1656992	0.1535745	8.3087071	14.1019420	0.0000004	0.0000676	condition8

feature	logFC	se	df	t	pval	adjP-val	contrast
UPS|P08263ups|GSTA1_HUMAN_UPS	1.9516727	0.1257383	7.560563	15.521705	0.0000005	0.0000786	condition8
UPS|P01031ups|CO5_HUMAN_UPS	1.8743353	0.1402741	8.177184	13.361944	0.0000008	0.0001099	condition8
UPS|P55957ups|BID_HUMAN_UPS	2.8068108	0.2119801	7.337388	13.240914	0.0000022	0.0002989	condition8
UPS|P41159ups|LEP_HUMAN_UPS	1.8822282	0.1648389	7.794137	11.418594	0.0000039	0.0005053	condition8
UPS|P61626ups|LYSC_HUMAN_UPS	2.0418133	0.1695186	7.169186	12.044776	0.0000051	0.0006457	condition8
UPS|P16083ups|NQO2_HUMAN_UPS	2.1471522	0.1927058	7.487862	11.142126	0.0000063	0.0007618	condition8
P40547	0.8003233	0.1077763	7.767913	7.425782	0.0000865	0.0100814	condition8
UPS|P10599ups|THIO_HUMAN_UPS	2.1253387	0.2940739	7.234291	7.227226	0.0001479	0.0166135	condition8
P40344	0.7006607	0.1174848	8.130494	5.963843	0.0003161	0.0342934	condition8

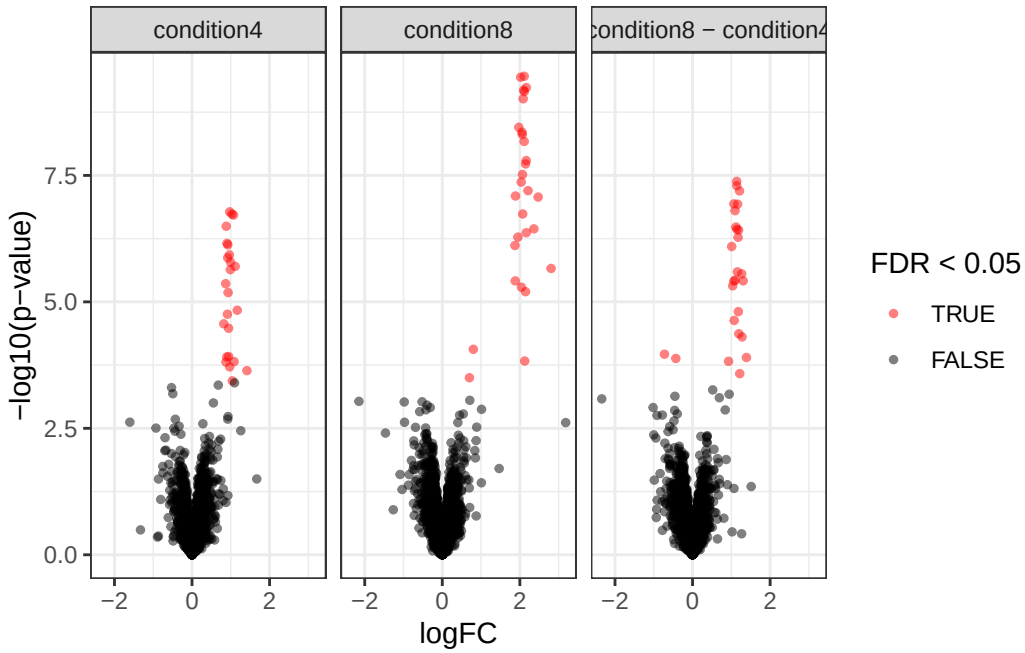
Contrast: condition8 - condition4 = 0 (26 significant proteins)

feature	logFC	se	df	t	pval	adjP-val	contrast
UPS|Q06830ups|PRDX1_HUMAN_UPS	1.1385808	0.0600148	8.269417	18.971670	0.0000000	0.0000674	condition8 - condition4
UPS|P02768ups|ALBU_HUMAN_UPS	1.1411177	0.0608178	8.192420	18.763077	0.0000000	0.0000674	condition8 - condition4
UPS|P04040ups|CATA_HUMAN_UPS	1.2115853	0.0676418	8.288912	17.911747	0.0000000	0.0000674	condition8 - condition4
UPS|P63165ups|SUMO1_HUMAN_UPS	1.0673637	0.0604327	7.952301	17.662021	0.0000001	0.0000738	condition8 - condition4
UPS|P01008ups|ANT3_HUMAN_UPS	1.1628546	0.0704954	8.337388	16.495465	0.0000001	0.0000738	condition8 - condition4
UPS|P06732ups|KCRM_HUMAN_UPS	1.0975866	0.0662858	8.093870	16.558388	0.0000002	0.0000826	condition8 - condition4
UPS|P12081ups|SYHC_HUMAN_UPS	1.1196009	0.0657997	7.406783	17.015302	0.0000003	0.0001330	condition8 - condition4

feature	logFC	se	df	t	pval	adjP-val	contrast
UPS|P00918ups|CAH2_HU	1.1430509	0.0706607	7.595280	16.176672	0.0000004	0.0001330	condition8 - condition4
MAN_UPS							
UPS|P00915ups|CAH1_HU	1.1877651	0.0827338	8.291745	14.356558	0.0000004	0.0001330	condition8 - condition4
MAN_UPS							
UPS|P62937ups|PPIA_HU	1.1755146	0.0654559	6.830835	17.958891	0.0000005	0.0001666	condition8 - condition4
MAN_UPS							
UPS|P15559ups|NQO1_HU	1.0102929	0.0767008	8.235205	13.172013	0.0000008	0.0002304	condition8 - condition4
MAN_UPS							
UPS|P69905ups|HBA_HU	1.1555774	0.1001508	8.123395	11.538417	0.0000025	0.0006687	condition8 - condition4
MAN_UPS							
UPS|P63279ups|UBC9_HU	1.2687779	0.1008897	7.440741	12.575928	0.0000028	0.0006770	condition8 - condition4
MAN_UPS							
UPS|P02144ups|MYG_HU	1.1042989	0.0920047	7.484986	12.002646	0.0000037	0.0007612	condition8 - condition4
MAN_UPS							
UPS|O76070ups|SYUG_HU	1.3061476	0.0884346	8.328066	14.769740	0.0000038	0.0007612	condition8 - condition4
MAN_UPS							
UPS|P62988ups|UBIQ_HU	1.0666447	0.0899568	7.528321	11.857303	0.0000039	0.0007612	condition8 - condition4
MAN_UPS							
UPS|P02753ups|RETBP_HU	1.0348931	0.0929123	7.753222	11.138388	0.0000048	0.0008935	condition8 - condition4
MAN_UPS							
UPS|P68871ups|HBB_HU	1.1811250	0.1328995	8.337388	8.8873570	0.0000150	0.0027301	condition8 - condition4
MAN_UPS							
UPS|P08263ups|GSTA1_HU	1.0747362	0.1172297	7.560563	9.1678150	0.0000239	0.0038620	condition8 - condition4
MAN_UPS							
UPS|P00167ups|CYB5_HU	1.1944457	0.1532058	8.308707	7.7963740	0.0000426	0.0067109	condition8 - condition4
MAN_UPS							
UPS|Q15843ups|NEDD8_HU	1.2772773	0.1587571	7.772949	8.0454830	0.0000493	0.0073955	condition8 - condition4
MAN_UPS							
Q05521	- 0.7242161	0.1000166	7.687778	- 7.240960	0.0001084	0.0155054	condition8 - condition4
UPS|P55957ups|BID_HU-	1.3897305	0.1896008	8.337388	7.3297710	0.0001259	0.0172298	condition8 - condition4
MAN_UPS							
P32573	- 0.4308098	0.0646557	8.337388	- 6.663139	0.0001315	0.0172427	condition8 - condition4
UPS|P01031ups|CO5_HU-	0.9291085	0.1402748	8.177184	6.6235190	0.0001498	0.0188584	condition8 - condition4
MAN_UPS							
UPS|P16083ups|NQO2_HU	1.2177468	0.1886623	7.487862	6.4546370	0.0002623	0.0317615	condition8 - condition4
MAN_UPS							

Volcanoplots

```
volcanoplots <- inferences |>
  plotVolcano() +
  facet_wrap(~contrast)
volcanoplots
```



Heatmaps

We conduct heatmaps for the significant proteins for each contrast.

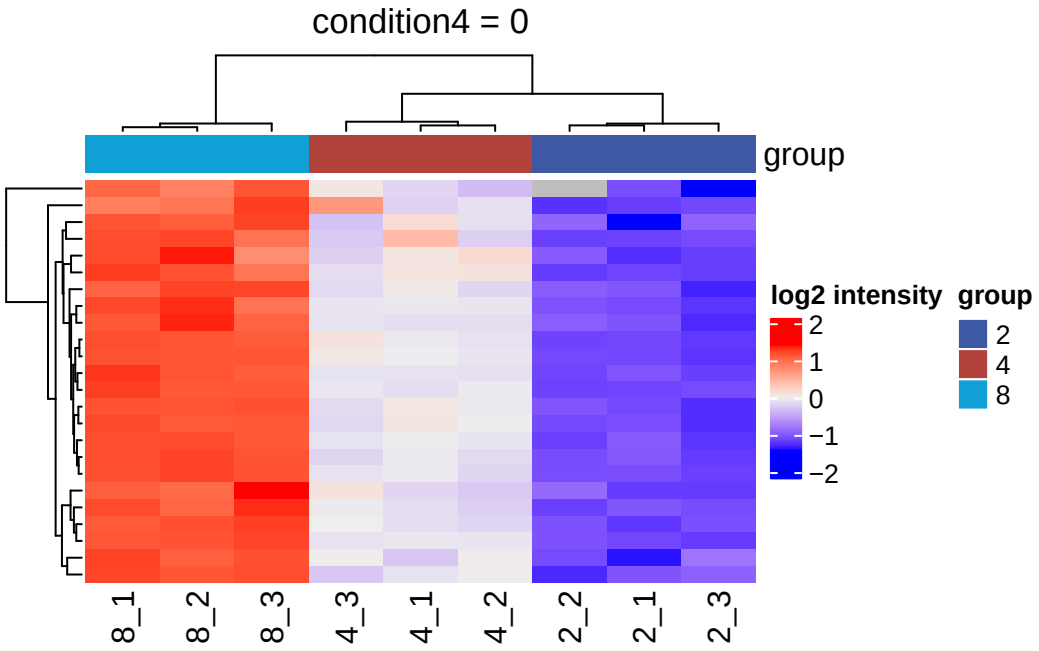
Click to see code

1. We first extract the `proteins` assay along with its colData.
2. We make an empty list `heatmaps` to store the plots.
3. We will loop over the column names of the contrast matrix `L`
4. We extract the names of the significant features for contrast `i`
5. We extract the quant data for the significant proteins
6. We extract annotation
7. We make the heatmap using the quants data. We do not cluster columns (samples) to keep them together according to the design.

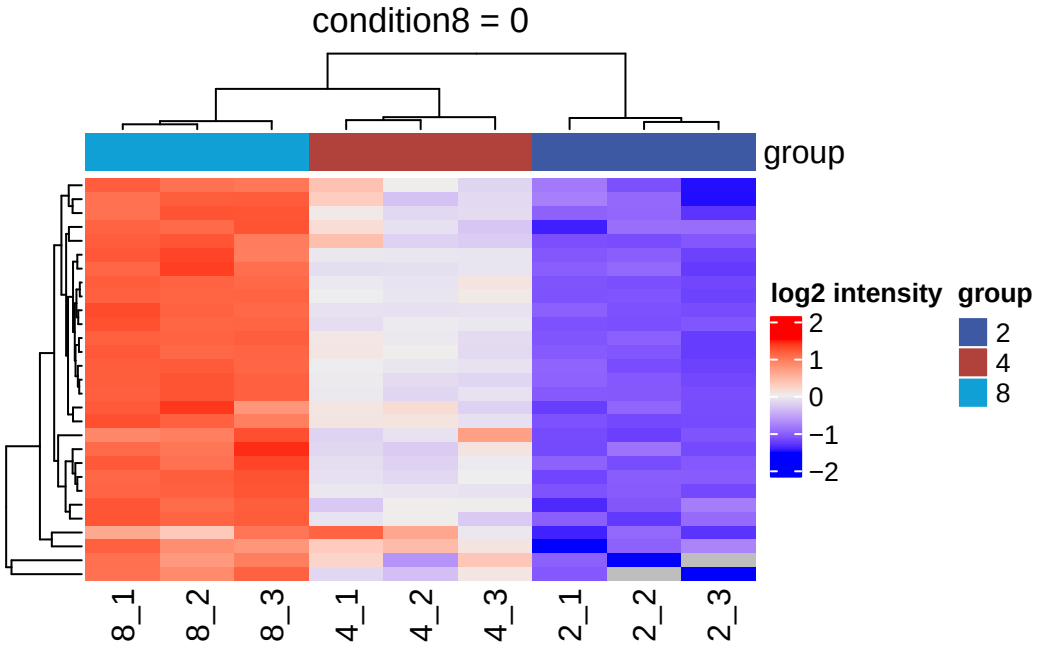
```
heatmaps <- lapply(colnames(L),
                   function(contrast, se, alpha)
                   {
                     sig <- rowData(se)[[contrast]] |>
                       filter(adjPval < alpha) |>
                       rownames()
                     if (length(sig) > 2)
                     {
                     quants <- t(scale(t(assay(se[sig,]))))
                     colnames(quants) <- se$sampleId #specific to this dataset to get short
                     rowclushlp <- quants
                     rowclushlp[is.na(rowclushlp)] <- min(quants,na.rm=TRUE) - 2
                     rowclus <- hclust(dist(rowclushlp))
                     annotations <- columnAnnotation(
                       group = se$condition
                       ) #3.
                     set.seed(1234) ## annotation colours are randomly generated by default
                     return(
                       Heatmap(show_row_names = FALSE,
                       quants,
                       name = "log2 intensity",
                       top_annotation = annotations,
                       column_title = paste0(contrast, " = 0"),
                       cluster_rows = rowclus
                       )
                     )
                     } else return(ggplot() + theme_minimal() + ggtitle(paste0(contrast, " =
                   },
                   se = getWithColData(qf, "proteins"),
                   alpha = alpha)
```

```
heatmaps
```

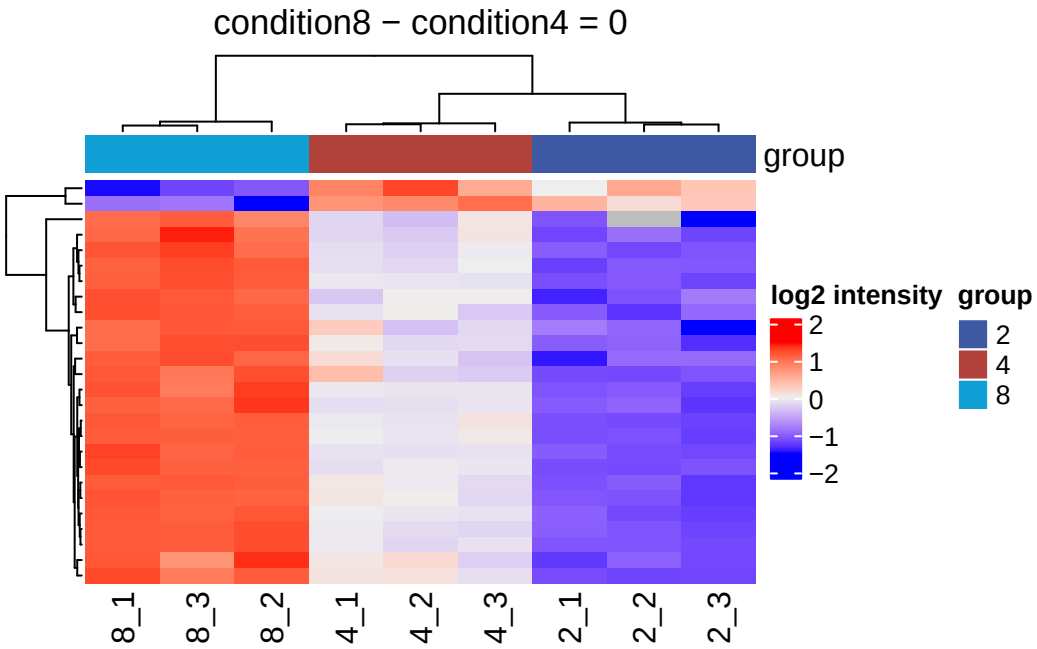
```
[[1]]
```


condition4 = 0
group
log2 intensity
2
1
0
−1
−2
group
2
4
8
8_1
8_2
8_3
4_3
4_1
4_2
2_2
2_1
2_3

[[2]]


condition8 = 0
group
log2 intensity
2
1
0
−1
−2
group
2
4
8
8_1
8_2
8_3
4_1
4_2
4_3
2_1
2_2
2_3

```
[[3]]
```



Assess performance

Note, that the evaluations in this section are typically not possible on real experimental data. Indeed, we are using a spike-in dataset so we know the ground truth: all human UPS proteins are differentially abundant (DA) between the conditions and the yeast proteins are non-DA.

True and false positives

All human UPS proteins are differentially abundant (DA) between the conditions and the yeast proteins are non-DA. For human UPS proteins the pattern "UPS" is in their protein name. We can now assess the number of false positives and true positives at the significance level $\alpha = 0.05$.

Click to see code

```
inferences <- inferences |>
  mutate(spikein = grepl("UPS", feature) |>
              as.factor() |>
              recode("FALSE" = "yeast",
                     "TRUE" = "ups")
         )

tpFpTable <- group_by(inferences, contrast) |>
    filter(adjPval < alpha) |>
    summarise("TP" = sum(spikein=="ups"),
              "FP" =  sum(spikein!="ups"),
              "FDP" = mean(spikein!="ups"))
```

```
tpFpTable
```

```
# A tibble: 3 x 4
  contrast                   TP    FP    FDP
  <chr>                   <int> <int>  <dbl>
1 condition4                 24     0 0
2 condition8                 27     2 0.0690
3 condition8 - condition4    24     2 0.0769
```

log-fold changes

As this is a spike-in study with known ground truth, we can also plot the log2 fold change distributions against the expected values, in this case 0 for the yeast proteins and the difference of the log concentration for the spiked-in UPS standards.

Implementation details

We first create a small table with the real values.

1. We extract the levels from factor condition
2. We convert then into a number as they refer to the real ratio
3. We log2-transform them
4. We calculate all pairwise differences between log2 transformed ratios to obtain the log2 fold changes

We use a similar operation to construct the name of the corresponding contrast.

```
(realLogFC <- data.frame(
    logFC = colData(qf)$condition |>
        unique() |> # 1.
        as.double() |> # 2.
        log2() |> # 3.
        combn(m=2, FUN = diff), #4.
    contrast = colData(qf)$condition |>
        unique() |>
        combn(m=2, FUN= function(x) paste0("condition",x[2]," - condition",x[1])) |>
        recode("condition4 - condition2" = "condition4",
               "condition8 - condition2" = "condition8")
    )
)
```

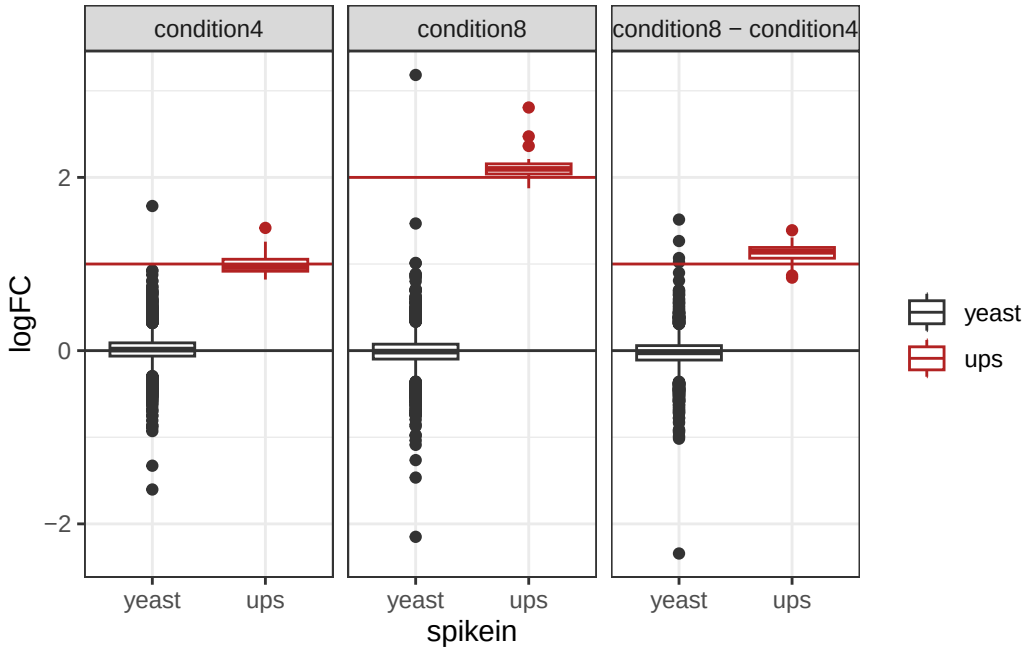
```
  logFC                contrast
1     1              condition4
2     2              condition8
3     1 condition8 - condition4
```

We now evaluate whether the estimated fold changes correspond to the real fold changes.

1. We plot the log fold changes as a function of the spikein condition and color according to spikein condition.
2. We add a boxplot layer.
3. We use custom colors.
4. We add a vertical line at 0, which corresponds to the known log2 fold change for yeast proteins
5. We add a vertical lines for known log2 fold changes of the spiked UPS proteins.

```
logFC <- inferences |>
  filter(!is.na(logFC)) |>
  ggplot(aes(x = spikein, y = logFC, color= spikein)) + #1.
  geom_boxplot() + #2.
  theme_bw() +
  scale_color_manual(
    values = c("grey20", "firebrick"), #3.
    name = "",
    labels = c("yeast", "ups")
    ) +
  geom_hline(yintercept = 0, color="grey20") + # 4.
  facet_wrap(~contrast) +
  geom_hline(aes(yintercept = logFC), color="firebrick", data=realLogFC) #5.
```

```
logFC
```



volcano-plots

Click to see code

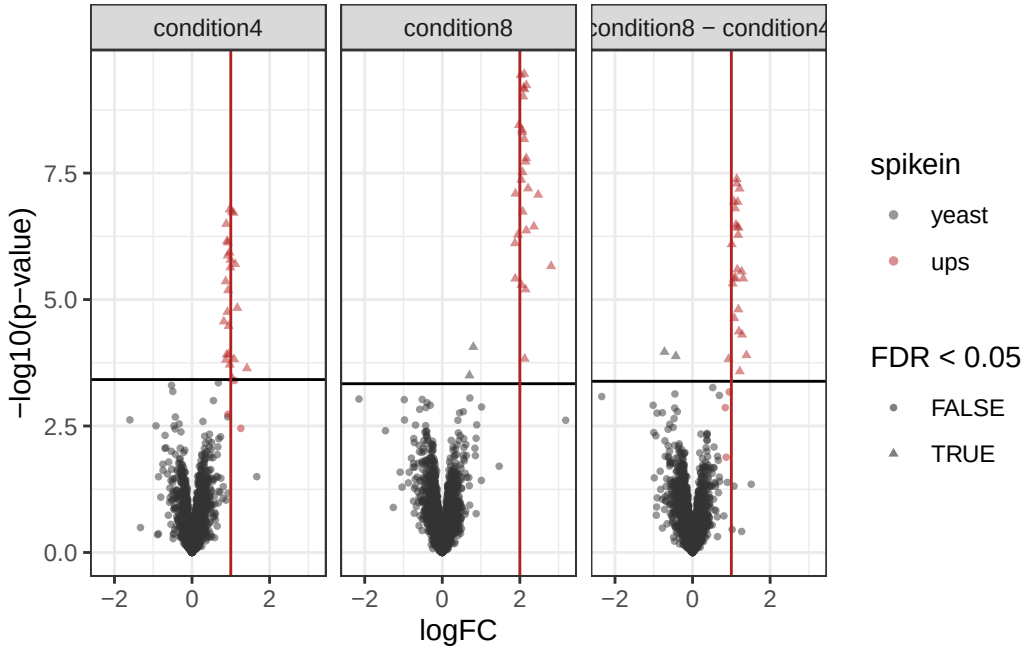
1. We plot the volcano-plots again per contrast,
2. We now add information on the protein type ("yeast" or "ups")
3. We add horizontal lines with the adjusted -log10 significant level for the original p-values that corresponds with the nominal 5% FDR-level.

4. We add vertical lines that correspond to the real log2 FC of UPS proteins.

```
volcanoplots <-  inferences |>
  plotVolcano() +
  facet_wrap(~contrast) + #1.
  aes(color = spikein, shape = adjPval < alpha) +
    scale_color_manual(
      values = c("grey20", "firebrick"),
      name = "spikein",
      labels = c("yeast", "ups")
      ) + #2.
```

```
  labs(shape="FDR < 0.05")  +
  geom_hline(aes(yintercept = -log10(adjAlpha)), data = inferences |>
               group_by(contrast) |>
               summarize(adjAlpha = alpha *mean(adjPval < alpha, na.rm=TRUE), .groups="drop")
  geom_vline(aes(xintercept = logFC), data = realLogFC, color = "firebrick") #4.
```

```
volcanoplots
```



True positive - False Discovery Proportion plots

We generate the TPR-FDP curves to assess the performance of the different workflows to prioritise differentially abundant proteins. Again, these curves are built using the ground truth information about which proteins are differentially abundant (spiked in) and which proteins are constant across samples.

Implementation details

We create two functions to compute the TPR and the FDP.

```
computeFDP <- function(pval, tp) {
    ord <- order(pval)
    fdp <- cumsum(!tp[ord]) / 1:length(tp)
```

```
    fdp[order(ord)]
}
computeTPR <- function(pval, tp, nTP = NULL) {
    if (is.null(nTP)) nTP <- sum(tp)
    ord <- order(pval)
    tpr <- cumsum(tp[ord]) / nTP
    tpr[order(ord)]
}
```

We apply these functions and compute the corresponding metric using the statistical inference results and the ground truth information.

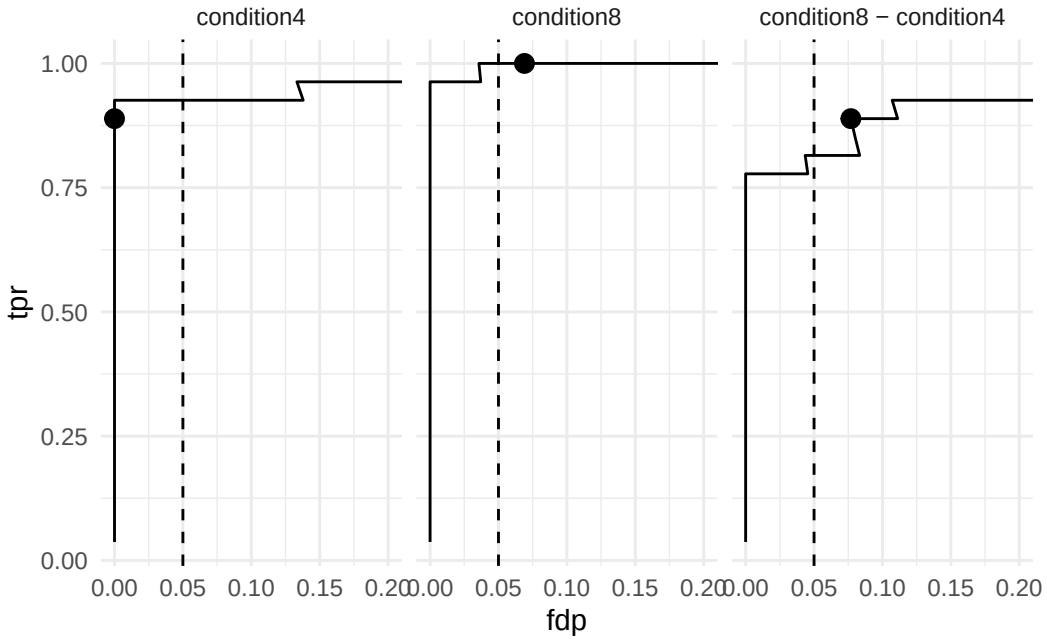
```
performance <- inferences |>
  group_by(contrast) |>
    na.exclude() |>
    mutate(tpr = computeTPR(pval, spikein == "ups"),
           fdp = computeFDP(pval, spikein == "ups")) |>
    arrange(fdp)
```

We also highlight the observed FDP at a $alpha = 5\%$ FDR threshold.

```
workPoints <- performance |>
  group_by(contrast) |>
    filter(adjPval < alpha) |>
    slice_max(pval)
```

```
pTprFdp <- ggplot(performance) +
    aes(
        y = fdp,
        x = tpr,
    ) +
    geom_line() +
    geom_point(data = workPoints, size = 3) +
    geom_hline(yintercept = 0.05, linetype = 2) +
    facet_wrap( ~ contrast) +
    coord_flip(ylim = c(0, 0.2)) +
    theme_minimal()
```

```
pTprFdp
```



Staes, An, Teresa Mendes Maia, Sara Dufour, et al. 2024. "Benefit of in Silico Predicted Spectral Libraries in Data-independent Acquisition Data Analysis Workflows." *Journal of Proteome Research* 23 (6): 2078–89. https://doi.org/10.1021/acs.jproteome.4c00048.