A Bioconductor workflow for processing, evaluating, and interpreting expression proteomics data

Charlotte Hutchings[1], **Charlotte S. Dawson**[1], **Thomas Krueger**[2], **Kathryn S. Lilley**[1], **and Lisa M. Breckels**[1]

[1]**Cambridge Centre for Proteomics, Department of Biochemistry, University of Cambridge, UK**
[2]**Department of Biochemistry, University of Cambridge, UK**

Abstract

Background: Expression proteomics involves the global evaluation of protein abundances within a system. In turn, differential expression analysis can be used to investigate changes in protein abundance upon perturbation to such a system.

Methods: Here, we provide a workflow for the processing, analysis and interpretation of quantitative mass spectrometry-based expression proteomics data. This workflow utilizes open-source R software packages from the Bioconductor project and guides users end-to-end and step-by-step through every stage of the analyses. As a use-case we generated expression proteomics data from HEK293 cells with and without a treatment. Of note, the experiment included cellular proteins labelled using tandem mass tag (TMT) technology and secreted proteins quantified using label-free quantitation (LFQ).

Results: The workflow explains the software infrastructure before focusing on data import, pre-processing and quality control. This is done individually for TMT and LFQ datasets. The application of statistical differential expression analysis is demonstrated, followed by interpretation via gene ontology enrichment analysis.

Conclusions: A comprehensive workflow for the processing, analysis and interpretation of expression proteomics is presented. The workflow is a valuable resource for the proteomics community and specifically beginners who are at least familiar with R who wish to understand and make data-driven decisions with regards to their analyses.

Keywords

Bioconductor, QFeatures, proteomics, shotgun proteomics, bottom-up proteomics, differential expression, mass spectrometry, quality control, data processing, limma

R version: R version 4.4.0 (2024-04-24)

Bioconductor version: 3.19

Introduction

Proteins are responsible for carrying out a multitude of biological tasks, implementing cellular functionality and determining phenotype. Mass spectrometry (MS)-based expression proteomics allows protein abundance to be quantified and compared between samples. In turn, differential protein abundance can be used to explore how biological systems respond to a perturbation. Many research groups have applied such methodologies to understand mechanisms of disease, elucidate cellular responses to external stimuli, and discover diagnostic biomarkers (see [1, 2, 3] for recent examples). As the potential of proteomics continues to be realized, there is a clear need for resources demonstrating how to deal with expression proteomics data in a robust and standardized manner.

The data generated during an expression proteomics experiment are complex, and unfortunately there is no one-size-fits-all method for the processing and analysis of such data. The reason for this is two-fold. Firstly, there are a wide range of experimental methods that can be used to generate expression proteomics data. Researchers can analyze full-length proteins (top-down proteomics) or complete an enzymatic digestion and analyze the resulting peptides. This proteolytic digestion can be either partial (middle-down proteomics) or complete (bottom-up proteomics). The latter approach is most commonly used as peptides have a more favourable ionization capacity, predictable fragmentation patterns, and can be separated via reversed phase liquid chromatography, ultimately making them more compatible with MS [4]. Within bottom-up proteomics, the relative quantitation of peptides can be determined using one of two approaches: (1) label-free or (2) label-based quantitation. Moreover, the latter can be implemented with a number of different peptide labelling chemistries, for example, using tandem mass tag (TMT), stable-isotope labelling by amino acids in cell culture (SILAC), isobaric tags for relative and absolute quantitation (iTRAQ), among others [5]. MS analysis can also be used in either data-dependent or data-independent acquisition (DDA or DIA) mode [6, 7]. Although all of these experimental methods typically result in a similar output, a matrix of quantitative values, the data are different and must be treated as such. Secondly, data processing is dependent upon the experimental goal and biological question being asked.

Here, we provide a step-by-step workflow for processing, analysing and interpreting expression proteomics data derived from a bottom-up experiment using DDA and either LFQ or TMT label-based peptide quantitation. We outline how to process the data starting from a peptide spectrum match (PSM)- or peptide- level `.txt` file. Such files are the outputs of most major third party search software (e.g., Proteome Discoverer, MaxQuant, FragPipe). We begin with data import and then guide users through the stages of data processing including data cleaning, quality control filtering, management of missing values, imputation, and aggregation to protein-level. Finally, we finish with how to discover differentially abundant proteins and carry out biological interpretation of the resulting data. The latter will be achieved through the application of gene ontology (GO) enrichment analysis. Hence, users can expect to generate lists of proteins that are significantly up- or downregulated in their system of interest, as well as the GO terms that are significantly over-represented in these proteins.

Using the R statistical programming environment [8] we make use of several state-of-the-art packages from the open-source, open-development Bioconductor project [9] to analyze use-case expression proteomics datasets [10] from both LFQ and label-based technologies.

Package installation

In this workflow we make use of open-source software from the R Bioconductor [9] project. The Bioconductor initiative provides R software packages dedicated to the processing of high-throughput complex biological data. Packages are open-source, well-documented and benefit from an active community of developers. We recommend users to download the RStudio integrated development environment (IDE) which provides a graphical interface to R programming language.

Detailed instructions for the installation of Bioconductor packages are documented on the Bioconductor Installation page. The main packages required for this workflow are installed using the code below. Additional packages required for downstream statistics and interpretation are installed as required.

```
if (!require("BiocManager", quietly = TRUE)) {
  install.packages("BiocManager")
}

BiocManager::install(c("QFeatures",
                       "ggplot2",
                       "stringr"
```

```
                        "NormalyzerDE",
                        "corrplot",
                        "Biostrings",
                        "limma",
                        "impute",
                        "dplyr",
                        "tibble",
                        "org.Hs.eg.db",
                        "clusterProfiler",
                        "enrichplot"))
```

After installation, each package must be loaded before it can be used in the R session. This is achieved via the `library` function. For example, to load the `QFeatures` package one would type `library("QFeatures")` after installation. Here we load all packages included in this workflow.

```
library("QFeatures")
library("ggplot2")
library("stringr")
library("dplyr")
library("tibble")
library("NormalyzerDE")
library("corrplot")
library("Biostrings")
library("limma")
library("org.Hs.eg.db")
library("clusterProfiler")
library("enrichplot")
```

Methods

The use-case: exploring changes in protein abundance in HEK293 cells upon perturbation

As a use-case, we analyze two quantitative proteomics datasets derived from a single experiment. The aim of the experiment was to reveal the differential abundance of proteins in HEK293 cells upon a particular treatment, the exact details of which are anonymized for the purpose of this workflow. An outline of the experimental method is provided in Figure 1.

Briefly, HEK293 cells were either (i) left untreated, or (ii) provided with the treatment of interest. These two conditions are referred to as 'control' and 'treated', respectively. Each condition was evaluated in triplicate. At 96- hours post-treatment, samples were collected and separated into cell pellet and supernatant fractions containing cellular and secreted proteins, respectively. Both fractions were denatured, alkylated and digested to peptides using trypsin.

The supernatant fractions were de-salted and analysed over a two-hour gradient in an Orbitrap Fusion™ Lumos™ Tribrid™ mass spectrometer coupled to an UltiMate™ 3000 HPLC system (Thermo Fisher Scientific). LFQ was achieved at the MS1 level based on signal intensities. Cell pellet fractions were labelled using TMT technology before being pooled and subjected to high pH reversed-phase peptide fractionation giving a total of 8 fractions. As before, each fraction was analysed over a two-hour gradient in an Orbitrap Fusion™ Lumos™ Tribrid™ mass spectrometer coupled to an UltiMate™ 3000 HPLC system (Thermo Fisher Scientific). To improve the accuracy of the quantitation of TMT-labelled peptides, synchronous precursor selection (SPS)-MS3 data acquisition was employed [11, 12]. Of note, TMT labelling of cellular proteins was achieved using a single TMT6plex. Hence, this workflow will not include guidance on multi-batch TMT effects or the use of internal reference scaling. For more information about the use of multiple TMTplexes users are directed to [13, 14].

The cell pellet and supernatant datasets were handled independently and we take advantage of this to discuss the processing of TMT-labelled and LFQ proteomics data. In both cases, the raw MS data were processed using Proteome Discoverer v2.5 (Thermo Fisher Scientific). While the focus in the workflow presented below is differential protein expression analysis, the data processing and quality control steps described here are applicable to any TMT or LFQ proteomics dataset. Importantly, however, the experimental aim will influence data-guided decisions and the considerations discussed here likely differ from those of spatial proteomics, for example.

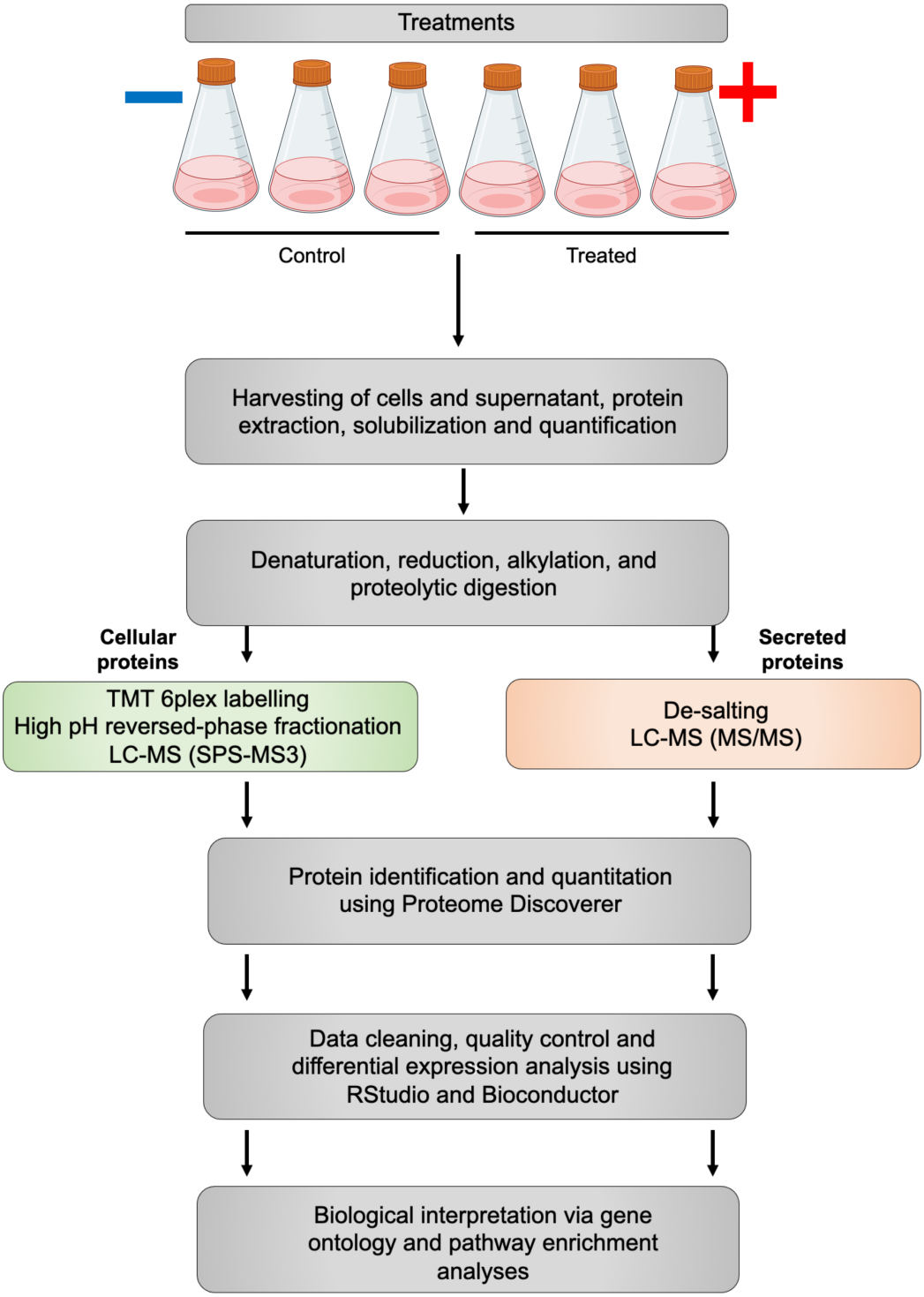


Figure 1. A schematic summary of the experimental protocol used to generate the use-case data.

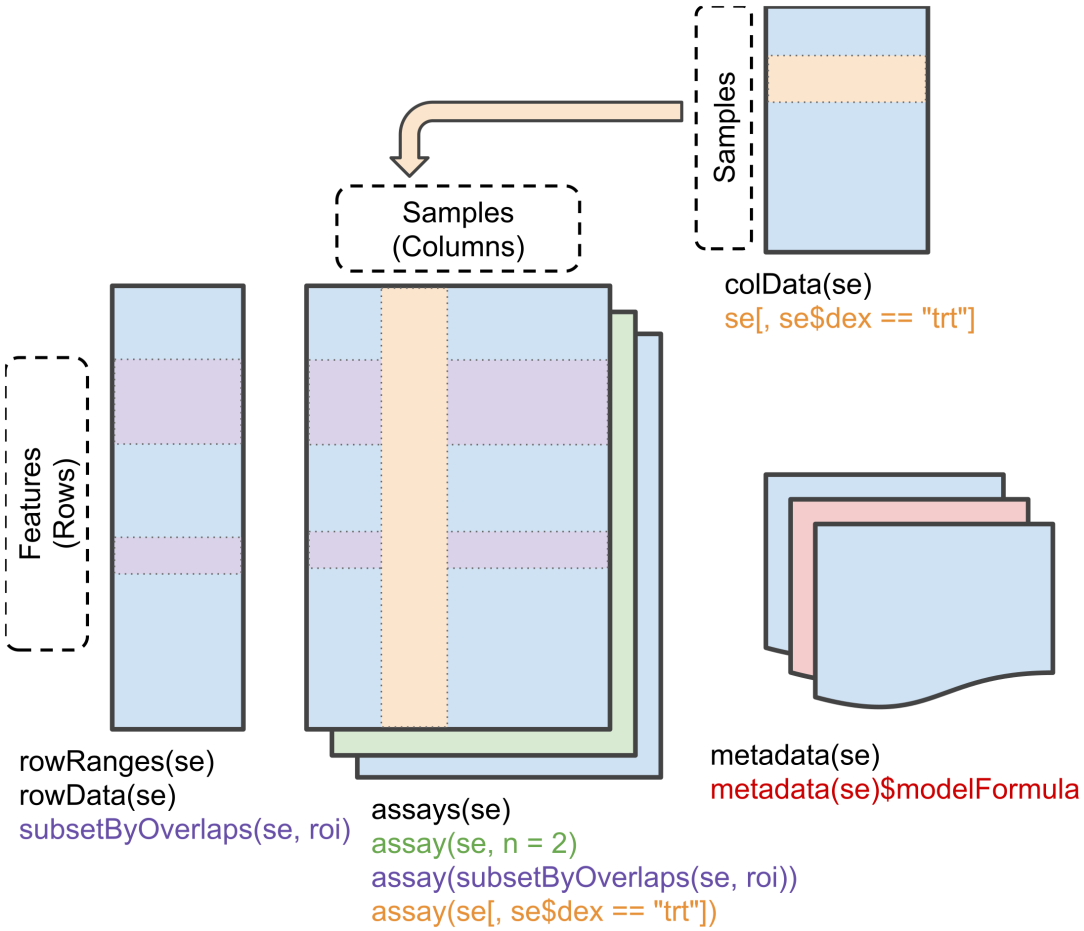


Figure 2. A graphic representation of the SummarizedExperiment (SE) object structure. Figure reproduced from the SummarizedExperiment package [18] vignette with permission.

Downloading the data

The files required for this workflow can be found deposited to the ProteomeXchange Consortium via the PRIDE [15, 16] partner repository with the dataset identifier PXD041794, Zenodo at `http://doi.org/10.5281/zenodo.11196770` and at the Github repository `https://github.com/CambridgeCentreForProteomics/f1000_expression_proteomics/`. Users are advised to download these files into their current working directory. In R the `setwd` function can be used to specify a working directory, or if using RStudio one can use the Session -> Set Working Directory menu.

The infrastructure: `QFeatures` and `SummarizedExperiment`s

To be able to conveniently track each step of this workflow, users should make use of the Quantitative features for mass spectrometry package, or `QFeatures`, Bioconductor package [17]. Prior to utilizing the `QFeatures` infrastructure, it is first necessary to understand the structure of a `SummarizedExperiment` [18] object as `QFeatures` objects are based on the `SummarizedExperiment` class. A `SummarizedExperiment`, often referred to as an SE, is a data container and S4 object comprised of three components: (1) the `colData` (column data) containing sample metadata, (2) the `rowData` containing data features, and (3) the `assay` storing quantitation data, as illustrated in Figure 2. The sample metadata includes annotations such as condition and replicate, and can be accessed using the `colData` function. Data features, accessed via the `rowData` function, represent information derived from the identification search. Examples include peptide sequence, master protein accession, and confidence scores. Finally, quantitative data is stored in the `assay` slot. These three independent data structures are neatly stored within a single `SummarizedExperiment` object.

A `QFeatures` object holds each level of quantitative proteomics data, namely (but not limited to) the PSM, peptide and protein-level data. Each level of the data is stored as its own `SummarizedExperiment` within a single `QFeatures` object. The lowest level data e.g. PSM, is first imported into a `QFeatures` object before aggregating upward towards protein-level (Figure 3). During this process of aggregation, `QFeatures` maintains the hierarchical links between quantitative levels whilst allowing easy access to all data levels for individual proteins of interest. This key aspect of `QFeatures` will be exemplified throughout this workflow.

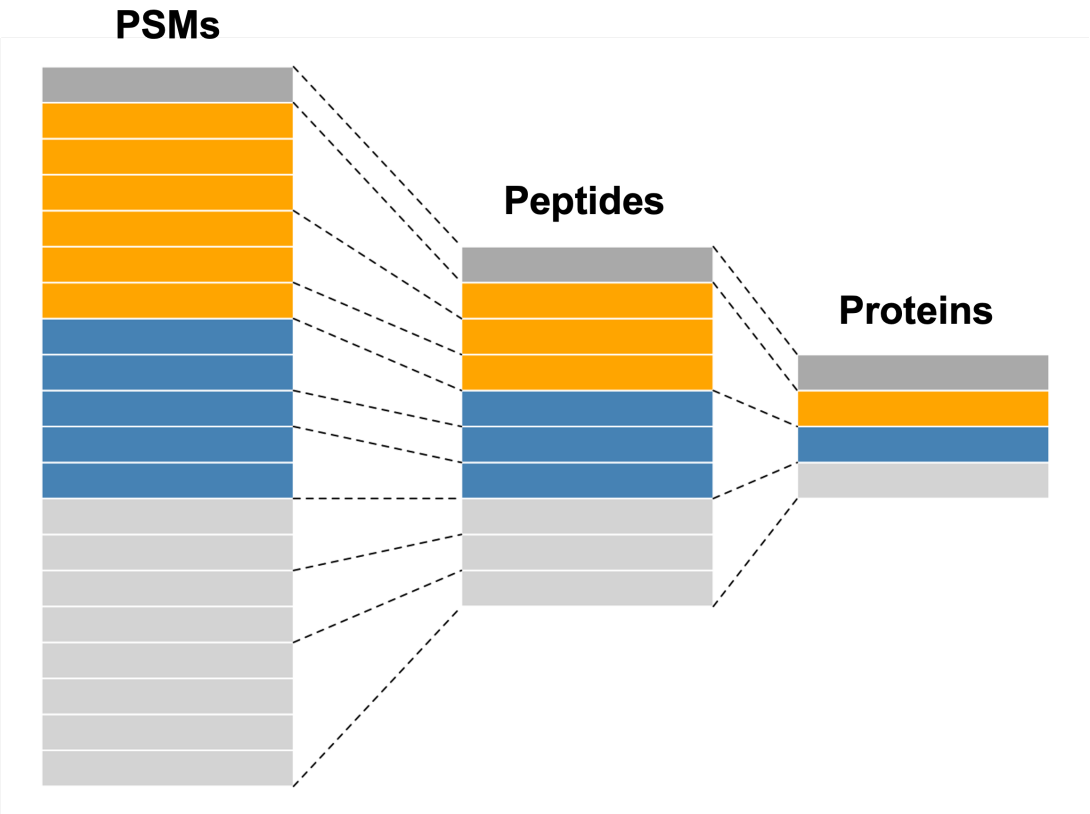


Figure 3. A graphic representation of the `QFeatures` object structure showing the relationship between assays. Figure modified from the `QFeatures` [17] vignette with permission.

Table 1. TMT labelling strategy in the use-case experiment

Sample Name	Condition	Replicate	Tag
S1	Treated	1	TMT128
S2	Treated	2	TMT127
S3	Treated	3	TMT131
S4	Control	1	TMT129
S5	Control	2	TMT126
S6	Control	3	TMT130

Additional guidance on the use of `QFeatures` can be found in [17]. For visualization of the data, all plots are generated using standard `ggplot` functionality, but could equally be produced using base R.

Processing and analysing quantitative TMT data

First, we provide a workflow for the processing and quality control of quantitative TMT-labelled data. As outlined above, the cell pellet fractions of triplicate control and treated HEK293 cells were labelled using a TMT6plex. Labelling was as outlined in Table 1.

Identification search of raw data

The first processing step in any MS-based proteomics experiment involves an identification search using the raw data. The aim of this search is to identify which peptide sequences, and therefore proteins, correspond to the raw spectra output from the mass spectrometer. Several third-party software exist to facilitate identification searches of raw MS data but ultimately the output of any search is a list of PSMs, peptides and protein identifications along with their corresponding quantification data.

The use-case data presented here was processed using Proteome Discoverer 2.5 and additional information about this search is provided in the appendix. Further, we provide template workflows for both the processing and consensus steps of the Proteome Discoverer identification run in the supplementary information. It is also possible to determine several of the key parameter settings during the preliminary data exploration. This step will be particularly important for those using publicly available data without detailed knowledge of

the identification search parameters. For now, we simply export the PSM-level `.txt` file from the Proteome Discoverer output.

Data import, housekeeping and exploration

Importing data into R and creating a QFeatures object

Data cleaning, exploration and filtering at the PSM-level is performed in R using `QFeatures`. First, data is imported from our PSM-level `.txt` file into a `data.frame` using the base R `read.delim` function (the equivalent for `.csv` files would be `read.csv`). This file should be stored within the users working directory. Next, we use the `names` function to inspect the column names and identify which columns contain quantitative data.

```
## Locate the PSM .txt file
cp_psm <- "cell_pellet_tmt_results_psms.txt"

## Import into a dataframe
cp_df <- read.delim(cp_psm)

## Identify columns containing quantitative data
cp_df %>%
  names()
```

```
##  [1] "PSMs.Workflow.ID"                  "PSMs.Peptide.ID"
##  [3] "Checked"                           "Tags"
##  [5] "Confidence"                        "Identifying.Node.Type"
##  [7] "Identifying.Node"                  "Search.ID"
##  [9] "Identifying.Node.No"               "PSM.Ambiguity"
## [11] "Sequence"                          "Annotated.Sequence"
## [13] "Modifications"                     "Number.of.Proteins"
## [15] "Master.Protein.Accessions"         "Master.Protein.Descriptions"
## [17] "Protein.Accessions"                "Protein.Descriptions"
## [19] "Number.of.Missed.Cleavages"        "Charge"
## [21] "Original.Precursor.Charge"         "Delta.Score"
## [23] "Delta.Cn"                          "Rank"
## [25] "Search.Engine.Rank"                "Concatenated.Rank"
## [27] "mz.in.Da"                          "MHplus.in.Da"
## [29] "Theo.MHplus.in.Da"                 "Delta.M.in.ppm"
## [31] "Delta.mz.in.Da"                    "Ions.Matched"
## [33] "Matched.Ions"                      "Total.Ions"
## [35] "Intensity"                         "Activation.Type"
## [37] "NCE.in.Percent"                    "MS.Order"
## [39] "Isolation.Interference.in.Percent" "SPS.Mass.Matches.in.Percent"
## [41] "Average.Reporter.SN"               "Ion.Inject.Time.in.ms"
## [43] "RT.in.min"                         "First.Scan"
## [45] "Last.Scan"                         "Master.Scans"
## [47] "Spectrum.File"                     "File.ID"
## [49] "Abundance.126"                     "Abundance.127"
## [51] "Abundance.128"                     "Abundance.129"
## [53] "Abundance.130"                     "Abundance.131"
## [55] "Quan.Info"                         "Peptides.Matched"
## [57] "XCorr"                             "Number.of.Protein.Groups"
## [59] "Percolator.q.Value"                "Percolator.PEP"
## [61] "Percolator.SVMScore"
```

Now that the quantitative data columns have been identified, we can pass our `data.frame` to the `readQFeatures` function and specify where our quantitative columns are. The latter is done by passing either a numeric or character vector to an argument called `quantCols` (`ecols` in previous package versions). Of note, the `readQFeatures` function can also take `fnames` as an argument to specify a column to be used as the row names of the imported object. Whilst previous `QFeatures` vignettes used the "Sequence" or "Annotated.Sequence" as row names, we advise against this because of the presence of PSMs matched to the same peptide sequence with different modifications. In such cases, multiple rows would have the same name forcing the `readQFeatures` function to output a "making assay row names unique" message and add an identifying number to the end of each duplicated row name. These sequences would then be considered as unique during the aggregation of PSM to peptide, thus resulting in two independent peptide- level quantitation values rather

than one. Therefore, we do not pass a `fnames` argument and the row names automatically become indices. Finally, we pass the `name` argument to indicate the type of data added.

```
## Create QFeatures
cp_qf <- readQFeatures(assayData = cp_df,
                       quantCols = 49:54,
                       name = "psms_raw")
```

```
## Checking arguments.

## Loading data as a 'SummarizedExperiment' object.

## Formatting sample annotations (colData).

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

Accessing the `QFeatures` infrastructure

As outlined above, a `QFeatures` data object is a list of `SummarizedExperiment` objects. As such, an individual `SummarizedExperiment` can be accessed using the standard double bracket nomenclature, as demonstrated in the code chunk below.

```
## Index using position
cp_qf[[1]]
```

```
## class: SummarizedExperiment
## dim: 48832 6
## metadata(0):
## assays(1): ''
## rownames(48832): 1 2 ... 48831 48832
## rowData names(55): PSMs.Workflow.ID PSMs.Peptide.ID ... Percolator.PEP
##   Percolator.SVMScore
## colnames(6): Abundance.126 Abundance.127 ... Abundance.130
##   Abundance.131
## colData names(0):
```

```
## Index using name
cp_qf[["psms_raw"]]
```

```
## class: SummarizedExperiment
## dim: 48832 6
## metadata(0):
## assays(1): ''
## rownames(48832): 1 2 ... 48831 48832
## rowData names(55): PSMs.Workflow.ID PSMs.Peptide.ID ... Percolator.PEP
##   Percolator.SVMScore
## colnames(6): Abundance.126 Abundance.127 ... Abundance.130
##   Abundance.131
## colData names(0):
```

A summary of the data contained in the slots is printed to the screen. To retrieve the `rowData`, `colData` or `assay` data from a particular `SummarizedExperiment` within a `QFeatures` object users can make use of the `rowData`, `colData` and `assay` functions. For plotting or data transformation it is necessary to convert to a `data.frame` or `tibble`.

```
## Access feature information with rowData
## The output should be converted to data.frame/tibble for further processing
cp_qf[["psms_raw"]] %>%
  rowData() %>%
  as_tibble() %>%
  summarise(mean_intensity = mean(Intensity))
```

```
## # A tibble: 1 x 1
##   mean_intensity
##            <dbl>
## 1      13281497.
```

To ensure that quantitative data has been correctly imported in as a numeric variable, we can look at the `assay` slot.

```
## Look at top 6 rows of the assay (quantitative data)
cp_qf[["psms_raw"]] %>%
  assay() %>%
  head()
```

```
##   Abundance.126 Abundance.127 Abundance.128 Abundance.129 Abundance.130
## 1            NA            NA            NA            NA            NA
## 2            NA            NA            NA            NA            NA
## 3            NA            NA            NA            NA            NA
## 4          14.1          18.1          11.8           9.4          18.9
## 5            NA           2.5            NA           3.4           3.2
## 6          16.7          26.2          18.7          22.4          25.3
##   Abundance.131
## 1            NA
## 2            NA
## 3            NA
## 4          13.5
## 5            NA
## 6          17.2
```

```
## Confirm numeric assay
cp_qf[["psms_raw"]] %>%
  assay() %>%
  type()
```

```
## [1] "double"
```

Our `assay` contains data of type `double`, a sub-type of numeric data in R. If users find that their quantitative data is of type `character`, it is necessary to correct this before moving on with the rest of the workflow.

Adding metadata

Having imported the data, each sample is first annotated with its TMT label, sample reference and condition. As this information is experimental metadata, it is added to the `colData` slot. It is important for users check that the `colData` annotations correspond to their correct samples, particularly if the column order in the imported results file does not align with the sample order. This is the case for the use-case data since TMT labels were randomized in an attempt to minimze the effect of TMT channel leakage.

```
## Add sample info as colData to QFeatures object
cp_qf$label <- c("TMT126",
                 "TMT127",
                 "TMT128",
                 "TMT129",
                 "TMT130",
                 "TMT131")

cp_qf$sample <- c("S5", "S2", "S1", "S4", "S6", "S3")

cp_qf$condition <- c("Control", "Treated", "Treated",
                     "Control", "Control", "Treated")

## Verify
colData(cp_qf)
```

```
## DataFrame with 6 rows and 3 columns
##                     label      sample   condition
##               <character> <character> <character>
## Abundance.126      TMT126          S5     Control
## Abundance.127      TMT127          S2     Treated
## Abundance.128      TMT128          S1     Treated
## Abundance.129      TMT129          S4     Control
## Abundance.130      TMT130          S6     Control
## Abundance.131      TMT131          S3     Treated
```

```
## Assign the colData to first assay as well
colData(cp_qf[["psms_raw"]]) <- colData(cp_qf)
```

It is also useful to clean up sample names such that they are short, intuitive and informative. This is done by editing the `colnames`. These steps may not always be necessary depending upon the identification search output.

```
## Clean sample names
colnames(cp_qf[["psms_raw"]]) <- c("S5", "S2", "S1", "S4", "S6", "S3")
```

Preliminary data exploration

As well as cleaning and annotating the data, it is always advisable to check that the import worked and that the data looks as expected. Further, preliminary exploration of the data can provide an early sign of whether the experiment and subsequent identification search were successful. Importantly, however, the names of key parameters will vary depending on the software used, and will likely change over time. Users will need to be aware of this and modify the code in this workflow accordingly.

```
## Check what information has been imported
cp_qf[["psms_raw"]] %>%
  rowData() %>%
  colnames()
```

```
##  [1] "PSMs.Workflow.ID"                 "PSMs.Peptide.ID"
##  [3] "Checked"                          "Tags"
##  [5] "Confidence"                       "Identifying.Node.Type"
##  [7] "Identifying.Node"                 "Search.ID"
##  [9] "Identifying.Node.No"              "PSM.Ambiguity"
## [11] "Sequence"                         "Annotated.Sequence"
## [13] "Modifications"                    "Number.of.Proteins"
## [15] "Master.Protein.Accessions"        "Master.Protein.Descriptions"
## [17] "Protein.Accessions"               "Protein.Descriptions"
## [19] "Number.of.Missed.Cleavages"       "Charge"
## [21] "Original.Precursor.Charge"        "Delta.Score"
## [23] "Delta.Cn"                         "Rank"
## [25] "Search.Engine.Rank"               "Concatenated.Rank"
## [27] "mz.in.Da"                         "MHplus.in.Da"
## [29] "Theo.MHplus.in.Da"                "Delta.M.in.ppm"
## [31] "Delta.mz.in.Da"                   "Ions.Matched"
## [33] "Matched.Ions"                     "Total.Ions"
## [35] "Intensity"                        "Activation.Type"
## [37] "NCE.in.Percent"                   "MS.Order"
## [39] "Isolation.Interference.in.Percent" "SPS.Mass.Matches.in.Percent"
## [41] "Average.Reporter.SN"              "Ion.Inject.Time.in.ms"
## [43] "RT.in.min"                        "First.Scan"
## [45] "Last.Scan"                        "Master.Scans"
## [47] "Spectrum.File"                    "File.ID"
## [49] "Quan.Info"                        "Peptides.Matched"
## [51] "XCorr"                            "Number.of.Protein.Groups"
## [53] "Percolator.q.Value"               "Percolator.PEP"
## [55] "Percolator.SVMScore"
```

```
## Find out how many PSMs are in the data
cp_qf[["psms_raw"]] %>%
  dim()
```

```
## [1] 48832     6
```

```
original_psms <- cp_qf[["psms_raw"]] %>%
  nrow() %>%
  as.numeric()
```

We can see that the original data includes 48832 PSMs across the 6 samples. It is also useful to make note of how many peptides and proteins the raw PSM data corresponds to, and to track how many we remove during the subsequent filtering steps. This can be done by checking how many unique entries are located within the "Sequence" and "Master.Protein.Accessions" for peptides and proteins, respectively. Of note, searching for unique peptide sequences means that the number of peptides does not include duplicated sequences with different modifications.

```
## Find out how many peptides and master proteins are in the data
original_peps <- cp_qf[["psms_raw"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(Sequence) %>%
  unique() %>%
  length() %>%
  as.numeric()

original_prots <- cp_qf[["psms_raw"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(Master.Protein.Accessions) %>%
  unique() %>%
  length() %>%
  as.numeric()

print(c(original_peps, original_prots))
```

```
## [1] 25969  5040
```

Hence, the output of the identification search contains 48832 PSMs corresponding to 25969 peptide sequences and 5040 master proteins. Finally, we confirm that the identification search was carried out as expected. For this, we print summaries of the key search parameters using the `table` function for discrete parameters and `summary` for those which are continuous. This is also helpful for users who are analyzing publicly available data and have limited knowledge about the identification search parameters.

```
## Check missed cleavages
cp_qf[["psms_raw"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(Number.of.Missed.Cleavages) %>%
  table()
```

```
## .
##     0     1     2
## 46164  2592    76
```

```
## Check precursor mass tolerance
cp_qf[["psms_raw"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(Delta.M.in.ppm) %>%
  summary()
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -8.9300 -0.6000  0.3700  0.6447  1.3100  9.6700
```

```
## Check fragment mass tolerance
cp_qf[["psms_raw"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(Delta.mz.in.Da) %>%
  summary()
```

```
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -0.0110400 -0.0004100  0.0002500  0.0006812  0.0010200  0.0135100
```

```
## Check PSM confidence allocations
cp_qf[["psms_raw"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(Confidence) %>%
  table()
```

```
## .
##  High
## 48832
```

Experimental quality control checks

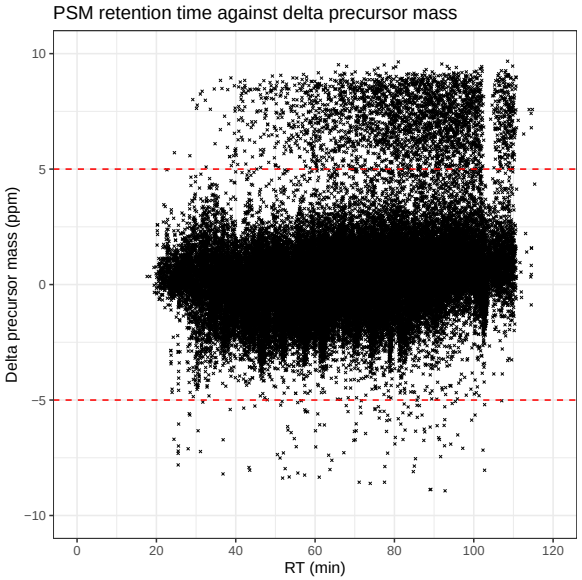
Experimental quality control of TMT-labelled quantitive proteomics data takes place in two steps: (1) assessment of the raw mass spectrometry data, and (2) evaluation of TMT labelling efficiency.

Quality control of the raw mass spectrometry data

Having taken an initial look at the output of the identification search, it is possible to create some simple plots to inspect the raw mass spectrometry data. Such plots are useful in revealing problems that may have occurred during the mass spectrometry run but are far from extensive. Users who wish to carry out a more in-depth evaluation of the raw mass spectrometry data may benefit from use of the `Spectra` Bioconductor package which allows for visualization and exploration of raw chromatograms and spectra, among other features [19].

The first plot we generate looks at the delta precursor mass, that is the difference between observed and estimated precursor mass, across retention time. Importantly, exploration of this raw data feature can only be done when using the raw data prior to recalibration. For users of Proteome Discoverer, this means using the spectral files node rather than the spectral files recalibration node.

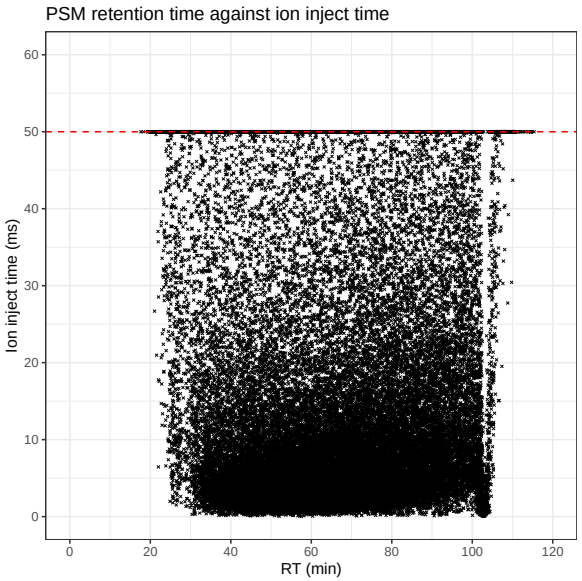
```
## Generate scatter plot of mass accuracy
cp_qf[["psms_raw"]] %>%
  rowData() %>%
  as_tibble() %>%
  ggplot(aes(x = RT.in.min, y = Delta.M.in.ppm)) +
  geom_point(size = 0.5, shape = 4) +
  geom_hline(yintercept = 5, linetype = "dashed", color = "red") +
  geom_hline(yintercept = -5, linetype = "dashed", color = "red") +
  labs(x = "RT (min)", y = "Delta precursor mass (ppm)") +
  scale_x_continuous(limits = c(0, 120), breaks = seq(0, 120, 20)) +
  scale_y_continuous(limits = c(-10, 10), breaks = c(-10, -5, 0, 5, 10)) +
  ggtitle("PSM retention time against delta precursor mass") +
  theme_bw()
```



Since we applied a precursor mass tolerance of 10 ppm during the identification search, all of the PSMs are within ±10 ppm. Ideally, however, we want the majority of the data to be within ±5 ppm since smaller delta masses correspond to a greater rate of correct peptide identifications. From the graph we have plotted we can see that indeed the majority of PSMs are within ±5 ppm. If users find that too many PSMs are outside of the desired ±5 ppm, it is advisable to check the calibration of the mass spectrometer.

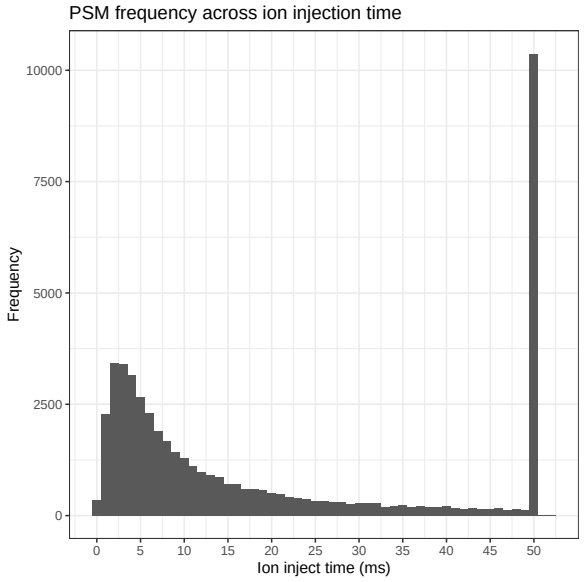
The second quality control plot of raw data is that of MS2 ion inject time across the retention time gradient. Here, it is desirable to achieve an average MS2 injection time of 50 ms or less, although the exact target threshold will depend upon the sample load. If the average ion inject time is longer than desired, then the ion transfer tube and/or front end optics of the instrument may require cleaning.

```
## Generate scatter plot of ion inject time across retention time
cp_qf[["psms_raw"]] %>%
  rowData() %>%
  as_tibble() %>%
  ggplot(aes(x = RT.in.min, y = Ion.Inject.Time.in.ms)) +
  geom_point(size = 0.5, shape = 4) +
  geom_hline(yintercept = 50, linetype = "dashed", color = "red") +
  labs(x = "RT (min)", y = "Ion inject time (ms)") +
  scale_x_continuous(limits = c(0, 120), breaks = seq(0, 120, 20)) +
  scale_y_continuous(limits = c(0, 60), breaks = seq(0, 60, 10)) +
  ggtitle("PSM retention time against ion inject time") +
  theme_bw()
```

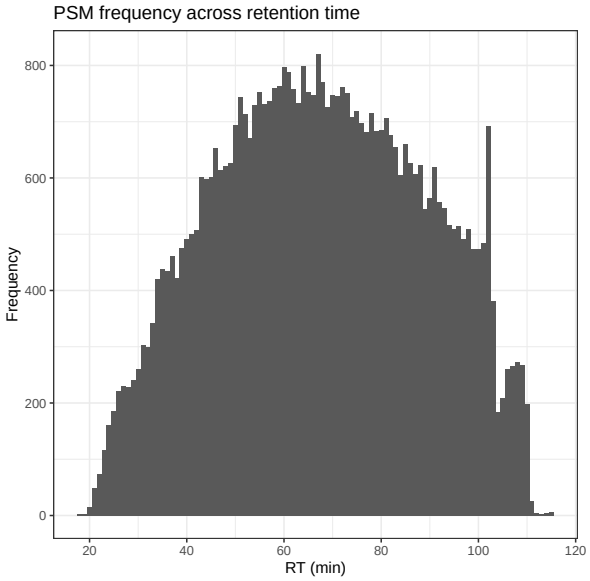


From this plot we can see that whilst there is a high density of PSMs at low inject times, there are also many data points found at the 50 ms threshold. This indicates that by increasing the time allowed for ions to accumulate in the ion trap, the number of PSMs could also have been increased. Finally, we inspect the distribution of PSMs across both the ion injection time and retention time by plotting histograms.

```
## Plot histogram of PSM ion inject time
cp_qf[["psms_raw"]] %>%
  rowData() %>%
  as_tibble() %>%
  ggplot(aes(x = Ion.Inject.Time.in.ms)) +
  geom_histogram(binwidth = 1) +
  labs(x = "Ion inject time (ms)", y = "Frequency") +
  scale_x_continuous(limits = c(-0.5, 52.5), breaks = seq(0, 50, 5)) +
  ggtitle("PSM frequency across ion injection time") +
  theme_bw()
```




```
## Plot histogram of PSM retention time
cp_qf[["psms_raw"]] %>%
  rowData() %>%
  as_tibble() %>%
  ggplot(aes(x = RT.in.min)) +
  geom_histogram(binwidth = 1) +
  labs(x = "RT (min)", y = "Frequency") +
  scale_x_continuous(breaks = seq(0, 120, 20)) +
  ggtitle("PSM frequency across retention time") +
  theme_bw()
```



The four plots that we have generated look relatively standard with no obvious problems indicated. Therefore, we continue by evaluating the quality of the processed data.

Checking the efficiency of TMT labelling

The most fundamental data quality control step in a TMT experiment is to check the TMT labelling efficiency. TMT labels react with amine groups present at the peptide N-terminus as well as the side chain of lysine (K) residues. Of note, lysine residues can be TMT modified regardless of whether they are present at the C-terminus of a trypic peptide or internally following miscleavage.

To evaluate the TMT labelling efficiency, a separate identification search of the raw data was completed with lysine (K) and peptide N-termini TMT labels considered as dynamic modifications rather than static. No additional residues (S or T) were evaluated for labelling in the search. This allows the search engine to assess the presence of both the modified (TMT labelled) and unmodified (original) forms of each peptide. The relative proportions of modified and unmodified peptides can then be used to calculate the TMT labelling efficiency. To demonstrate how to check for TMT labelling efficiency, only two of the eight fractions were utilized for this search.

As we will only look at TMT efficiency at the PSM-level, here we upload the `.txt` into a `data.frame` to format into a `SummarizedExperiment` object. This is done using the `readSummarizedExperiment` function and the same arguments as those in `readQFeatures`. The reason for this is that we only need the PSM-level data, so do not need to generate a `QFeatures` object with the function of holding multiple `SummarizedExperiments` across different levels.

```
## Locate the PSM .txt file
tmt_psm <- "cell_pellet_tmt_efficiency_psms.txt"

## Import into a data.frame
tmt_df <- read.delim(tmt_psm)

## Identify columns containing quantitative data
tmt_df %>%
  names()
```

```
##  [1] "PSMs.Workflow.ID"                "PSMs.Peptide.ID"
##  [3] "Checked"                         "Tags"
##  [5] "Confidence"                      "Identifying.Node.Type"
##  [7] "Identifying.Node"                "Search.ID"
##  [9] "Identifying.Node.No"             "PSM.Ambiguity"
## [11] "Sequence"                        "Annotated.Sequence"
## [13] "Modifications"                   "Number.of.Proteins"
## [15] "Master.Protein.Accessions"       "Master.Protein.Descriptions"
## [17] "Protein.Accessions"              "Protein.Descriptions"
## [19] "Number.of.Missed.Cleavages"      "Charge"
```

```
## [21] "Original.Precursor.Charge"          "Delta.Score"
## [23] "Delta.Cn"                           "Rank"
## [25] "Search.Engine.Rank"                 "Concatenated.Rank"
## [27] "mz.in.Da"                           "MHplus.in.Da"
## [29] "Theo.MHplus.in.Da"                  "Delta.M.in.ppm"
## [31] "Delta.mz.in.Da"                     "Ions.Matched"
## [33] "Matched.Ions"                       "Total.Ions"
## [35] "Intensity"                          "Activation.Type"
## [37] "NCE.in.Percent"                     "MS.Order"
## [39] "Isolation.Interference.in.Percent" "SPS.Mass.Matches.in.Percent"
## [41] "Average.Reporter.SN"                "Ion.Inject.Time.in.ms"
## [43] "RT.in.min"                          "First.Scan"
## [45] "Last.Scan"                          "Master.Scans"
## [47] "Spectrum.File"                      "File.ID"
## [49] "Abundance.126"                      "Abundance.127"
## [51] "Abundance.128"                      "Abundance.129"
## [53] "Abundance.130"                      "Abundance.131"
## [55] "Quan.Info"                          "Peptides.Matched"
## [57] "XCorr"                              "Number.of.Protein.Groups"
## [59] "Contaminant"                        "Percolator.q.Value"
## [61] "Percolator.PEP"                     "Percolator.SVMScore"
```

```
## Read in as a SummarizedExperiment
tmt_se <- readSummarizedExperiment(assayData = tmt_df,
                                   quantCols = 49:54)

## Clean sample names
colnames(tmt_se) <- c("S5", "S2", "S1", "S4", "S6", "S3")

## Add sample info as colData to QFeatures object
tmt_se$label <- c("TMT126",
                  "TMT127",
                  "TMT128",
                  "TMT129",
                  "TMT130",
                  "TMT131")

tmt_se$sample <- c("S5", "S2", "S1", "S4", "S6", "S3")

tmt_se$condition <- c("Control", "Treated", "Treated",
                      "Control", "Control", "Treated")
```

```
## Verify
colData(tmt_se)
```

```
## DataFrame with 6 rows and 3 columns
##          label      sample   condition
##    <character> <character> <character>
## S5      TMT126          S5     Control
## S2      TMT127          S2     Treated
## S1      TMT128          S1     Treated
## S4      TMT129          S4     Control
## S6      TMT130          S6     Control
## S3      TMT131          S3     Treated
```

Information about the presence of labels is stored within the 'Modifications' feature of the `rowData`. Using this information, the TMT labelling efficiency of the experiment is calculated using the code chunks below. Users should alter this code if TMTpro reagents are being used such that "TMT6plex" is replaced by "TMTpro".

First we consider the efficiency of peptide N-termini TMT labelling. We use the `grep` function to identify PSMs which are annotated as having an N-Term TMT6plex modification. We then calculate the number of PSMs with this annotation as a proportion of the total number of PSMs.

Table 2. ThermoFisher search strategy recommendations based on TMT labelling efficiency

N-term efficiency	K efficiency	Suggested search method
> 98 %	> 98 %	Both modifications as 'static'
85 - 95 %	> 98 %	N-terminal modification 'dynamic' and K modification 'static'
< 84 %	< 84 %	Data not suitable for quantitation

```
## Count the total number of PSMs
tmt_total <- length(tmt_se)

## Count the number of PSMs with an N-terminal TMT modification
nterm_labelled_rows <- grep("N-Term\\(TMT6plex\\)",
                            rowData(tmt_se)$Modifications)
nterm_psms_labelled <- length(nterm_labelled_rows)

## Calculate N-terminal TMT labelling efficiency
efficiency_nterm <- (nterm_psms_labelled / tmt_total) * 100

efficiency_nterm %>%
  round(digits = 1) %>%
  print()
```

```
## [1] 96.8
```

Secondly, we consider the TMT labelling efficiency of lysine (K) residues. As mentioned above, lysine residues can be TMT labelled regardless of their position within a peptide. Hence, we here calculate lysine labelling efficiency on a per lysine residue basis.

```
## Count the number of lysine TMT6plex modifications in the PSM data
k_tmt <- str_count(string = rowData(tmt_se)$Modifications,
                   pattern = "K[0-9]{1,2}\\(TMT6plex\\)") %>%
  sum() %>%
  as.numeric()

## Count the number of lysine residues in the PSM data
k_total <- str_count(string = rowData(tmt_se)$Sequence,
                     pattern = "K") %>%
  sum() %>%
  as.numeric()

## Determine the percentage of TMT labelled lysines
efficiency_k <- (k_tmt / k_total) * 100

efficiency_k %>%
  round(digits = 1) %>%
  print()
```

```
## [1] 98.5
```

Users should aim for an overall TMT labelling efficiency >90 % in order to achieve reliable quantitation. In cases where labelling efficiency is towards the lower end of the acceptable range, TMT labels should be set as dynamic modifications during the final identification search, although this will increase the search space and time as well as influencing FDR calculations. A summary of the current advice from Thermo Fisher is provided in Table 2. Where labelling efficiency is calculated as being between categories, how to progress is ultimately decided by the user.

Since the use-case data has a sufficiently high TMT labelling efficiency, we can continue to use the output of the identification search. This search considered TMT labelling of lysines as a static modification whilst N-terminal labelling was kept as dynamic, to investigate the presence of protein N- terminal modifications.

Basic data cleaning

Being confident that the experiment and identification search were successful, we can now begin with some basic data cleaning. However, we also want to keep a copy of the raw PSM data. Therefore, we first create

a second copy of the PSM `SummarizedExperiment`, called "psms_filtered", and add it to the `QFeatures` object. This is done using the `addAssay` function. All changes made at the PSM level will then only be applied to this second copy, so that we can refer back to the original data if needed.

```
## Extract the "psms_raw" SummarizedExperiment
data_copy <- cp_qf[["psms_raw"]]

## Add copy of SummarizedExperiment
cp_qf <- addAssay(x = cp_qf,
                  y = data_copy,
                  name = "psms_filtered")

## Verify
cp_qf
```

```
## An instance of class QFeatures containing 2 assays:
##  [1] psms_raw: SummarizedExperiment with 48832 rows and 6 columns
##  [2] psms_filtered: SummarizedExperiment with 48832 rows and 6 columns
```

Of note, manually adding an assay (or `SummarizedExperiment`) to the `QFeatures` object does not automatically generate links between these assays. We will manually add the explicit links later, after we complete data cleaning and filtering.

The cleaning steps included in this section are non-specific and should be applied to all quantitative proteomics datasets. The names of key parameters will vary in data outputs from alternative third party software, however, and users should remain aware of both terminology changes over time as well as the introduction of new filters. All data cleaning steps are completed in the same way. We first determine how many rows, here PSMs, meet the conditions for removal. This is achieved by using the `dplyr::count` function. The unwanted rows are removed using the `filterFeatures` function. Since we only wish to apply the filters to the "psms_filtered" level, we specify this by using the `i =` argument. If this argument is not used, `filterFeatures` will remove features from all assays within a `QFeatures` object.

Removing PSMs not matched to a master protein

The first common cleaning step we carry out is the removal of PSMs that have not been assigned to a master protein during the identification search. This can happen when the search software is unable to resolve conflicts caused by the presence of the isobaric amino acids leucine and isoleucine. Before implementing the filter, it is useful to find out how many PSMs we expect to remove. This is easily done by using the `dplyr::count` on the master protein column. Any master proteins that return `TRUE` will be removed by filtering. If this returns no `TRUE` values, users should move on to the next filtering step without removing rows as this will introduce an error.

```
## Find out how many PSMs we expect to lose
cp_qf[["psms_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  dplyr::count(Master.Protein.Accessions == "")
```

```
## # A tibble: 2 x 2
##   `Master.Protein.Accessions == ""`     n
##   <lgl>                             <int>
## 1 FALSE                             48660
## 2 TRUE                                172
```

For users who wish to explicitly track the process of data cleaning, the code chunk below demonstrates how to print a message containing the number of features removed.

```
paste("Removing",
      length(which(rowData(
        cp_qf[["psms_filtered"]])$Master.Protein.Accessions == "")),
      "PSMs without a master protein accession") %>%
  message()
```

```
## Removing 172 PSMs without a master protein accession
```

This code could be adapted to each cleaning and filtering step. To maintain simplicity of this workflow, we will not print explicit messages at each step. Instead, the decision to do so is left to the user.

```
## Remove PSMs without a master protein accession using filterFeatures
cp_qf <- cp_qf %>%
  filterFeatures(~ !Master.Protein.Accessions == "",
                 i = "psms_filtered")
```

Removing PSMs matched to a contaminant protein

Next we remove PSMs corresponding to contaminant proteins. Such proteins can be introduced intentionally as reagents during sample preparation, as is the case for digestive enzymes, or accidentally, as seen with human keratins derived from skin and hair. Since these proteins do not contribute to the biological question being asked and it is standard practice to remove them from the data. This is done by using a carefully curated, sample-specific contaminant database. Critically, the database used for filtering should be the same one that was used during the identification search. Whilst it is possible to remove contaminants using the `filterFeatures` function on a contaminants annotation column (as per the `QFeatures` processing vignette), we demonstrate how to filter using only contaminant protein accessions for users who do not have contaminant annotations within their identification data.

For this experiment, a contaminant database from [20] was used. The `.fasta` file for this database is available at the Hao Group's Github Repository for Protein Contaminant Libraries for DDA and DIA Proteomics and specifically can be found at `https://github.com/HaoGroup-ProtContLib/Protein-Contaminant-Libraries-for-DDA-a tree/main/Universal%20protein%20contaminant%20FASTA`. Here, we import this file using the `fasta.index` function from the `Biostrings` package [21]. This function requires a file path to the `.fasta` file and the asks users to specify the sequence type. In this case we have amino acid sequences so pass `seqtype = "AA"`. The function returns a `data.frame` with one row per FASTA entry. We then can extract the protein accessions from the fasta file. Users will need to alter the below code according to the contaminant file used.

```
## Load Hao group .fasta file used in search
cont_fasta <- "220813_universal_protein_contaminants_Haogroup.fasta"
conts <- Biostrings::fasta.index(cont_fasta, seqtype = "AA")

## Extract only the protein accessions (not Cont_ at the start)
cont_acc <- regexpr("(?<=\\_).*?(?=\\|)", conts$desc, perl = TRUE) %>%
  regmatches(conts$desc, .)
```

Now we have our contaminant list by accession number, we can identify and remove PSMs with any contaminant protein within their "Protein.Accessions". Importantly, filtering on "Protein.Accessions" ensures the removal of PSMs which matched to a protein group containing a contaminant protein, even if the contaminant protein is not the group's master protein.

```
## Define function to find contaminants
find_cont <- function(se, cont_acc) {
  cont_indices <- c()
  for (i in 1:length(cont_acc)) {
    cont_protein <- cont_acc[i]
    cont_present <- grep(cont_protein, rowData(se)$Protein.Accessions)
    output <- c(cont_present)
    cont_indices <- append(cont_indices, output)
  }
  cont_psm_indices <- cont_indices
}

## Store row indices of PSMs matched to a contaminant-containing protein group
cont_psms <- find_cont(cp_qf[["psms_filtered"]], cont_acc)

## If we find contaminants, remove these rows from the data
if (length(cont_psms) > 0)
  cp_qf[["psms_filtered"]] <- cp_qf[["psms_filtered"]][-cont_psms, ]
```

At this point, users can also remove any additional proteins which may not have been included in the contaminant database. For example, users may wish to remove human trypsin (accession P35050) should it appear in their data.

Several third party software also have the option to directly annotate which fasta file (here, the human proteome or contaminant database) a PSM is derived from. In such cases, filtering can be simplified by removing PSMs annotated as contaminants in the output file.

Removing PSMs which lack quantitative data

Now that we are left with only PSMs matched to proteins of interest, we filter out PSMs which cannot be used for quantitation. This includes some PSMs which lack quantitative information altogether. In outputs derived from Proteome Discoverer this information is included in the "Quan.Info" column where PSMs are annotated as having "NoQuanLabels". For users who have considered both lysine and N-terminal TMT labels as static modifications, the data should not contain any PSMs without quantitative information. However, since the use-case data was derived from a search in which N-terminal TMT modifications were dynamic, the data does include this annotation. Users are reminded that column names are typically software-specific as the "Quan.Info" column is found only in outputs derived from Proteome Discoverer. However, the majority of alternative third party softwares will have an equivalent column containing the same information.

```
## Find out how many PSMs we expect to lose
cp_qf[["psms_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  dplyr::count(Quan.Info == "NoQuanLabels")
```

```
## # A tibble: 2 x 2
##   `Quan.Info == "NoQuanLabels"`     n
##   <lgl>                         <int>
## 1 FALSE                         47241
## 2 TRUE                            228
```

```
## Drop these rows from the data
cp_qf <- cp_qf %>%
  filterFeatures(~ !Quan.Info == "NoQuanLabels",
                 i = "psms_filtered")
```

This point in the workflow is a good time to check whether there are any other annotations within the "Quan.Info" column. For example, if there are any PSMs which have been "ExcludedByMethod", this indicates that a PSM-level filter was applied in Proteome Discoverer during the identification search. If this is the case, users should determine which filter has been applied to the data and decide whether to remove the PSMs which were "ExcludedByMethod" (thereby applying the pre-set threshold) or leave them in (disregard the threshold).

```
## Are there any remaining annotations in the Quan.Info column?
cp_qf[["psms_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(Quan.Info) %>%
  table()
```

```
## .
##
## 47241
```

In the above code chunk we see there are no remaining annotations in the "Quan.Info" column so we can continue.

Removing PSMs which are not unique to a protein

The next step is to consider which PSMs are to be used for quantitation. There are two ways in which a PSM can be considered as unique. The first and most pure form of uniqueness comes from a PSM corresponding to a single protein only. This results in the PSM being allocated to one protein and one protein group. However, it is common to expand the definition of unique to include PSMs that map to multiple proteins within a single protein group. That is PSMs which are allocated to more than one protein but only one protein group. This distinction is ultimately up to the user. By contrast, PSMs corresponding to razor and shared peptides are linked to multiple proteins across multiple protein groups. In this workflow, the final grouping of peptides to

proteins will be done based on master protein accession. Therefore, differential expression analysis will be based on protein groups, and we here consider unique as any PSM linked to only one protein group. This means removing PSMs where "Number.of.Protein.Groups" is not equal to 1.

In the below code chunk we count the number of PSMs linked to more than 1 protein group.

```
## Find out how many PSMs we expect to lose
cp_qf[["psms_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  dplyr::count(Number.of.Protein.Groups != 1)
```

```
## # A tibble: 2 x 2
##   `Number.of.Protein.Groups != 1`     n
##   <lgl>                           <int>
## 1 FALSE                           44501
## 2 TRUE                             2740
```

We again use the `filterFeatures` function to retain PSMs linked to only 1 protein group and discard any PSMs linked to more 1 group.

```
## Remove these rows from the data
cp_qf <- cp_qf %>%
  filterFeatures(~ Number.of.Protein.Groups == 1,
                 i = "psms_filtered")
```

Additional considerations regarding protein isoforms Users searching against a database that includes protein isoforms must take extra caution when defining 'unique' PSMs. A PSM that corresponds to a single protein when data is searched against the proteome without isoforms may correspond to multiple proteins once additional isoforms are included. As a result, PSMs or peptides that were previously mapped to one protein and one protein group could instead be mapped to multiple proteins and one protein group. These PSMs would be filtered out by defining 'unique' as corresponding to only one protein and one protein group, but would be retained if the definition was expanded to multiple proteins and one protein group. Users should be aware of these possibilities and select their filtering strategy based on the biological question of interest.

Removing PSMs that are not rank 1

Another filter that is important for quantitation is that of PSM rank. Since individual spectra can have multiple candidate peptide matches, Proteome Discoverer uses a scoring algorithm to determine the probability of a PSM being incorrect. Once each candidate PSM has been given a score, the one with the lowest score (lowest probability of being incorrect) is allocated rank 1. The PSM with the second lowest probability of being incorrect is rank 2, and so on. For the analysis, we only want rank 1 PSMs to be retained.

```
## Find out how many PSMs we expect to lose
cp_qf[["psms_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  dplyr::count(Rank != 1)
```

```
## # A tibble: 2 x 2
##   `Rank != 1`     n
##   <lgl>       <int>
## 1 FALSE       43426
## 2 TRUE         1075
```

```
## Drop these rows from the data
cp_qf <- cp_qf %>%
  filterFeatures(~ Rank == 1,
                 i = "psms_filtered")
```

The majority of search engines, including SequestHT, also provide their own PSM rank. To be conservative and ensure accurate quantitation, we also only retain PSMs that have a search engine rank of 1.

Table 3. Definitions of PSM ambiguity categories based on Proteome Discoverer outputs.

PSM category	Definition
Unambiguous	The only candidate PSM
Selected	PSM was selected from a group of candidates
Rejected	PSM was rejected from a group of candidates
Ambiguous	Two or more candidate PSMs could not be distinguished
Unconsidered	PSM was not considered suitable

```
## Find out how many PSMs we expect to lose
cp_qf[["psms_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  dplyr::count(Search.Engine.Rank != 1)
```

```
## # A tibble: 2 x 2
##   `Search.Engine.Rank != 1`     n
##   <lgl>                     <int>
## 1 FALSE                     43153
## 2 TRUE                        273
```

```
## Drop these rows from the data
cp_qf <- cp_qf %>%
  filterFeatures(~ Search.Engine.Rank == 1,
                 i = "psms_filtered")
```

Removing ambiguous PSMs

We also retain only unambiguous PSMs. Since there are several candidate peptides for each spectra, Proteome Discoverer allocates each PSM a level of ambiguity to indicate whether it was possible to determine a definite PSM or whether one had to be selected from a number of candidates. The allocation of PSM ambiguity takes place during the process of protein grouping and the definitions of each ambiguity assignment are given below in Table 3.

Importantly, depending upon the software being used, output files may already have excluded some of these categories. It is still good to check before proceeding with the data.

```
## Find out how many PSMs we expect to lose
cp_qf[["psms_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  dplyr::count(PSM.Ambiguity != "Unambiguous")
```

```
## # A tibble: 1 x 2
##   `PSM.Ambiguity != "Unambiguous"`     n
##   <lgl>                            <int>
## 1 FALSE                            43153
```

```
## No PSMs to remove so proceed
```

Controlling false discovery rate

Finally, it is necessary to control the false discovery rate. During a database search, peptide spectrum matches (PSMs) are generated by matching raw mass spectra to peptide sequences. However, only a number of these PSMs will be true positive matches, whilst the rest are false positives, or 'false discoveries'. Most third party software contain internal algorithms to determine the number of false discoveries and the probability of a given feature (PSM, peptide or protein) being a false discovery. This information can then be used to control the false discovery rate (FDR), typically to keep this below 5% for exploratory analyses or 1% for stringent analyses.

During the database search of the use-case data, only high confidence PSMs with an FDR $< 1\%$ were retained.

```
cp_qf[["psms_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(Confidence) %>%
  table()
```

```
## .
##  High
## 43153
```

Importantly, however, FDR becomes inflated as you aggregate up from PSM to peptide and then from peptide to protein. Therefore, a 1% FDR at PSM-level does not ensure a 1% FDR at protein-level. To access information about protein-level FDR, we have to import the protein-level `.txt` file from our database search. As before, users should store this file in their working directory.

```
## Import protein-level data
protein_cp <- read.delim(file = "cell_pellet_tmt_results_proteins.txt")

## Check column names
protein_cp %>%
  names()
```

```
##  [1] "Proteins.Unique.Sequence.ID"
##  [2] "Checked"
##  [3] "Tags"
##  [4] "Protein.FDR.Confidence.Combined"
##  [5] "Master"
##  [6] "Protein.Group.IDs"
##  [7] "Accession"
##  [8] "Description"
##  [9] "Sequence"
## [10] "FASTA.Title.Lines"
## [11] "Exp.q.value.Combined"
## [12] "Sum.PEP.Score"
## [13] "Number.of.Decoy.Protein.Combined"
## [14] "Coverage.in.Percent"
## [15] "Number.of.Peptides"
## [16] "Number.of.PSMs"
## [17] "Number.of.Protein.Unique.Peptides"
## [18] "Number.of.Unique.Peptides"
## [19] "Number.of.AAs"
## [20] "MW.in.kDa"
## [21] "calc.pI"
## [22] "Score.Sequest.HT.Sequest.HT"
## [23] "Coverage.in.Percent.by.Search.Engine.Sequest.HT"
## [24] "Number.of.PSMs.by.Search.Engine.Sequest.HT"
## [25] "Number.of.Peptides.by.Search.Engine.Sequest.HT"
## [26] "Abundance.F1.126.Sample"
## [27] "Abundance.F1.127.Sample"
## [28] "Abundance.F1.128.Sample"
## [29] "Abundance.F1.129.Sample"
## [30] "Abundance.F1.130.Sample"
## [31] "Abundance.F1.131.Sample"
## [32] "Abundances.Count.F1.126.Sample"
## [33] "Abundances.Count.F1.127.Sample"
## [34] "Abundances.Count.F1.128.Sample"
## [35] "Abundances.Count.F1.129.Sample"
## [36] "Abundances.Count.F1.130.Sample"
## [37] "Abundances.Count.F1.131.Sample"
## [38] "Found.in.Fraction.F11"
## [39] "Found.in.Fraction.F12"
## [40] "Found.in.Fraction.F13"
## [41] "Found.in.Fraction.F14"
## [42] "Found.in.Fraction.F15"
```

```
## [43] "Found.in.Fraction.F16"
## [44] "Found.in.Fraction.F17"
## [45] "Found.in.Fraction.F18"
## [46] "Found.in.Sample.in.S1.F1.126.Sample"
## [47] "Found.in.Sample.in.S2.F1.127.Sample"
## [48] "Found.in.Sample.in.S3.F1.128.Sample"
## [49] "Found.in.Sample.in.S4.F1.129.Sample"
## [50] "Found.in.Sample.in.S5.F1.130.Sample"
## [51] "Found.in.Sample.in.S6.F1.131.Sample"
## [52] "Found.in.Sample.Group.in.S1.F1.126.Sample"
## [53] "Found.in.Sample.Group.in.S2.F1.127.Sample"
## [54] "Found.in.Sample.Group.in.S3.F1.128.Sample"
## [55] "Found.in.Sample.Group.in.S4.F1.129.Sample"
## [56] "Found.in.Sample.Group.in.S5.F1.130.Sample"
## [57] "Found.in.Sample.Group.in.S6.F1.131.Sample"
## [58] "Number.of.Protein.Groups"
```

The FDR information is contained in a column called `Protein.FDR.Confidence.Combined`. We can get an overview of the data in this column by using the `table` function.

```
protein_cp %>%
  pull(Protein.FDR.Confidence.Combined) %>%
  table()
```

```
## .
##   High    Low Medium
##   4824     75    467
```

We see proteins that are annotated as high confidence = FDR < 1%, medium confidence = FDR < 5% and low confidence = FDR > 5%.

In order to use the `filterFeatures` function to remove PSMs corresponding to proteins which do not pass our FDR threshold (1%), we first need to add this annotation to the `rowData` of our `"psms_filtered"` experiment. To do this, we first extract the master protein accessions from our PSM-level data. Next, we extract these proteins from our protein-level search output along with their corresponding confidence ratings.

```
## Extract master protein accessions from our PSM-level data
proteins_in_data <- cp_qf[["psms_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  select(Master.Protein.Accessions)

## Extract protein accessions and corresponding confidence from search output file
protein_search_output <- protein_cp %>%
  select(Accession, Protein.FDR.Confidence.Combined)
```

Now we can use the `left_join` function to merge these two datasets.

```
## Combine data
protein_fdr <- left_join(x = proteins_in_data,
                         y = protein_search_output,
                         by = c("Master.Protein.Accessions" = "Accession"))

protein_fdr %>%
  head()
```

```
## # A tibble: 6 x 2
##   Master.Protein.Accessions Protein.FDR.Confidence.Combined
##   <chr>                     <chr>
## 1 Q92597                    High
## 2 Q9UGP4                    High
## 3 Q9BRX2                    High
## 4 P14868                    High
## 5 O00154                    High
## 6 P48651                    High
```

Finally, we add the annotations from our "Protein.FDR.Confidence.Combined" column to the rowData of our "psms_filtered" experiment.

```
rowData(cp_qf[["psms_filtered"]])$Protein.Confidence <- protein_fdr$Protein.FDR.Confidence.Combined
```

We can now use filterFeatures to remove PSMs corresponding to low or medium confidence master proteins.

```
cp_qf <- cp_qf %>%
  filterFeatures(~ Protein.Confidence == "High", i = "psms_filtered")
```

```
## 'Protein.Confidence' found in 1 out of 2 assay(s)
## No filter applied to the following assay(s) because one or more filtering variables are missing i
## You can control whether to remove or keep the features using the 'keep' argument (see '?filterFea
```

Assessing the impact of non-specific data cleaning

Now that we have finished the non-specific data cleaning, we can pause and check to see what this has done to the data. We determine the number and proportion of PSMs, peptides, and proteins lost from the original dataset.

```
## Determine number and proportion of PSMs removed
psms_remaining <- cp_qf[["psms_filtered"]] %>%
  nrow() %>%
  as.numeric()

psms_removed <- original_psms - psms_remaining
psms_removed_prop <- ((psms_removed / original_psms) * 100) %>%
  round(digits = 2)

## Determine number and proportion of peptides removed
peps_remaining <- cp_qf[["psms_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(Sequence) %>%
  unique() %>%
  length() %>%
  as.numeric()

peps_removed <- original_peps - peps_remaining
peps_removed_prop <- ((peps_removed / original_peps) * 100) %>%
  round(digits = 2)

## Determine number and proportion of proteins removed
prots_remaining <- cp_qf[["psms_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(Master.Protein.Accessions) %>%
  unique() %>%
  length() %>%
  as.numeric()

prots_removed <- original_prots - prots_remaining
prots_removed_prop <- ((prots_removed / original_prots) * 100) %>%
  round(digits = 2)
```

```
## Print as a table
data.frame("Feature" = c("PSMs",
                         "Peptides",
                         "Proteins"),
           "Number lost" = c(psms_removed,
                             peps_removed,
                             prots_removed),
```

```
           "Percentage lost" = c(psms_removed_prop,
                                 peps_removed_prop,
                                 prots_removed_prop))
```

```
##    Feature Number.lost Percentage.lost
## 1     PSMs        6066           12.42
## 2 Peptides        1925            7.41
## 3 Proteins         812           16.11
```

PSM quality control filtering

The next step is to take a look at the data and make informed decisions about in-depth filtering. Here, we focus on three key quality control filters for TMT data: 1) average reporter ion signal-to-noise (S/N) ratio, 2) percentage co-isolation interference, and 3) percentage SPS mass match. It is possible to set thresholds for these three parameters during the identification search. However, specifying thresholds prior to exploring the data could lead to unnecessarily excessive data exclusion or the retention of poor quality PSMs. We suggest that users set the thresholds for all three aforementioned filters to 0 during the identification search, thus allowing maximum flexibility during data processing. In all cases, quality control filtering represents a trade-off between ensuring high quality data and losing potentially informative data. This means that the thresholds used for such filtering will likely depend upon the initial quality of the data and the number of PSMs, as well as the experimental goal being stringent or exploratory.

Quality control: Average reporter ion signal-to-noise

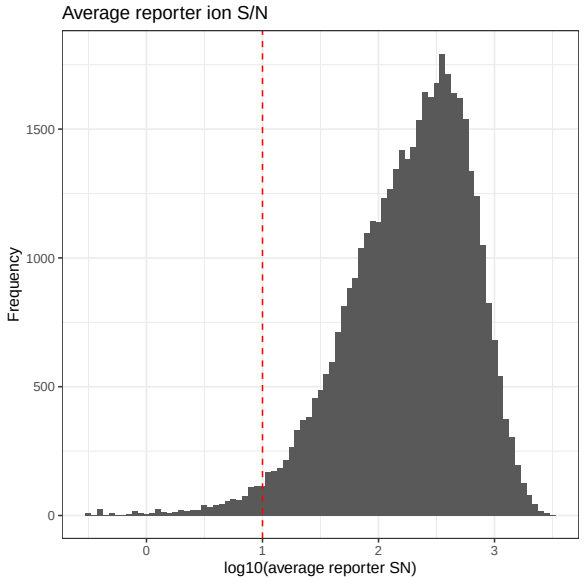
Intensity measurements derived from a small number of ions tend to be more variable and less accurate. Therefore, reporter ion spectra with peaks generated from a small number of ions should be filtered out to ensure accurate quantitation and avoid stochastic ion effects. When using an orbitrap analyzer, as was the case in the collection of the use-case data, the number of ions is proportional to the S/N value of a peak. Hence, the average reporter ion S/N ratio can be used to filter out quantification based on too few ions.

To determine an appropriate reporter ion S/N threshold we need to understand the original, unfiltered data. Here, we print a `summary` of the average reporter S/N before plotting a simple histogram to visualize the data. The default threshold for average reporter ion S/N when filtering within Proteome Discoverer is 10, or 1 on the base-10 logarithmic scale displayed here. We include a line to show where this threshold would be on the data distribution.

```
## Get summary information
cp_qf[["psms_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(Average.Reporter.SN) %>%
  summary()
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##     0.3    84.3   216.4   322.1   451.1  3008.2     138
```

```
## Plot histogram of reporter ion signal-to-noise
cp_qf[["psms_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  ggplot(aes(x = log10(Average.Reporter.SN))) +
  geom_histogram(binwidth = 0.05) +
  geom_vline(xintercept = 1, linetype = "dashed", color = "red") +
  labs(x = "log10(average reporter SN)", y = "Frequency") +
  ggtitle("Average reporter ion S/N") +
  theme_bw()
```



From the distribution of the data it is clear that applying such a threshold would not result in dramatic data loss. Whilst we could set a higher threshold for more stringent analysis, this would lead to unnecessary data loss. Therefore, we keep PSMs with an average reporter ion S/N threshold of 10 or more. We also remove PSMs that have an NA value for their average reporter ion S/N since their quality cannot be guaranteed. This is done by including `na.rm = TRUE`.

```
## Find out how many PSMs we expect to lose
cp_qf[["psms_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  dplyr::count(Average.Reporter.SN < 10)
```

```
## # A tibble: 3 x 2
##   `Average.Reporter.SN < 10`     n
##   <lgl>                      <int>
## 1 FALSE                      41696
## 2 TRUE                         932
## 3 NA                           138
```

```
## Drop these rows from the data
cp_qf <- cp_qf %>%
  filterFeatures(~ Average.Reporter.SN >= 10,
                 na.rm = TRUE,
                 i = "psms_filtered")
```

Quality control: Isolation interference

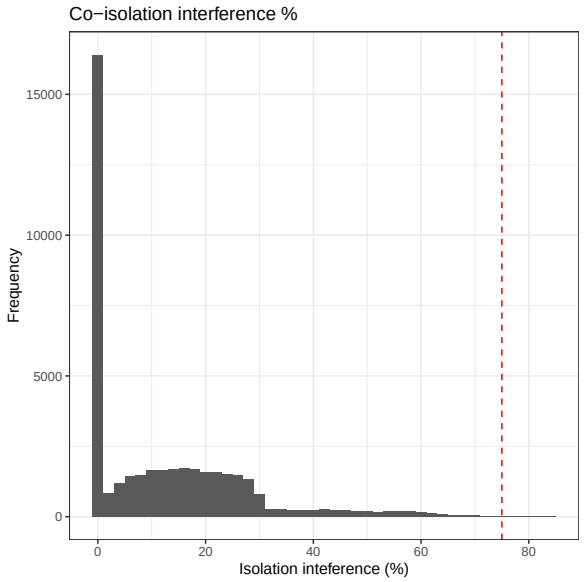
A second data-dependent quality control parameter which should be considered is the isolation interference. The first type of interference that occurs during a TMT experiment is reporter ion interference, also known as cross-label isotopic impurity. This type of interference arises from manufacturing-level impurities and experimental error. The former should be reduced somewhat by the inclusion of lot-specific correction factors in the search set-up and users should ensure that these corrections are applied. In Proteome Discoverer this means setting "Apply Quan Value Corrections" to "TRUE" within the reporter ions quantifier node. The second form of interference is co-isolation interference which occurs during the MS run when multiple labelled precursor peptides are co-isolated in a single data acquisition window. Following fragmentation of the co-isolated peptides, this results in an MS2 or MS3 reporter ion peak (depending upon the experimental design) derived from multiple precursor peptides. Hence, co-isolation interference leads to inaccurate quantitation of the identified peptide. This problem is reduced by filtering out PSMs with a high percentage isolation interference value. As was the case for reporter ion S/N, Proteome Discoverer has a suggested default threshold for isolation interference - 50% for MS2 experiments and 75% for SPS-MS3 experiments.

Again, we get a `summary` and visualize the data using the code chunk below.

```
## Get summary information
cp_qf[["psms_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(Isolation.Interference.in.Percent) %>%
  summary()
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   0.000   8.365  12.623  21.035  84.379
```

```
## Plot histogram of co-isolation interference
cp_qf[["psms_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  ggplot(aes(x = Isolation.Interference.in.Percent)) +
  geom_histogram(binwidth = 2) +
  geom_vline(xintercept = 75, linetype = "dashed", color = "red") +
  labs(x = "Isolation inteference (%)", y = "Frequency") +
  ggtitle("Co-isolation interference %") +
  theme_bw()
```



Looking at the data, very few PSMs have an isolation interference above the suggested threshold, and hence minimal data will be lost. Again, we choose to apply the standard threshold with the understanding that decreasing the threshold would result in greater data loss. Importantly, we are able to apply relatively standard thresholds here as the preliminary exploration did not expose any problems with the experimental data (in terms of labelling or MS analysis). If users have reason to believe the data is of poorer quality then more stringent thresholding should be considered.

```
## Find out how many PSMs we expect to lose
cp_qf[["psms_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  dplyr::count(Isolation.Interference.in.Percent > 75)
```

```
## # A tibble: 2 x 2
##   `Isolation.Interference.in.Percent > 75`     n
##   <lgl>                                    <int>
## 1 FALSE                                    41638
## 2 TRUE                                        58
```

```
## Remove these rows from the data
cp_qf <- cp_qf %>%
  filterFeatures(~ Isolation.Interference.in.Percent <= 75,
                 na.rm = TRUE,
                 i = "psms_filtered")
```

Quality control: SPS mass match

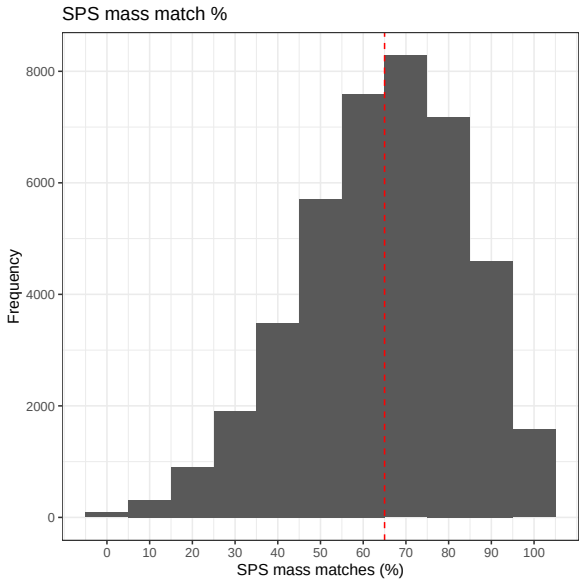
The final quality control filter that we will apply is a percentage SPS mass match threshold. SPS mass match is a metric which has been introduced by Proteome Discoverer versions 2.3 and above to quantify the percentage of SPS-MS3 fragments that can still be explicitly traced back to the precursor peptide. This parameter is of particular importance given that quantitation is based on the SPS-MS3 spectra. Unfortunately, the SPS Mass Match percentage is currently only a feature of Proteome Discoverer (2.3 and above) and will not be available to users of other third party software.

We follow the same format as before to investigate the SPS Mass Match (%) distribution of the data. The default threshold within Proteome Discoverer is a SPS Mass Match above 65%. In reality, since SPS Mass Match is only reported to the nearest 10%, removing PSMs annotated with a value below 65% means removing those with 60% or less. Hence, only PSMs with 70% SPS Mass Match or above would be retained. We can see how many PSMs would be lost based on such thresholds using the code chunk below.

```
## Get summary information
cp_qf[["psms_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(SPS.Mass.Matches.in.Percent) %>%
  summary()
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00   50.00   70.00   64.46   80.00  100.00
```

```
## Plot histogram of SPS mass match %
cp_qf[["psms_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  ggplot(aes(x = SPS.Mass.Matches.in.Percent)) +
  geom_histogram(binwidth = 10) +
  geom_vline(xintercept = 65, linetype = "dashed", color = "red") +
  labs(x = "SPS mass matches (%)", y = "Frequency") +
  scale_x_continuous(breaks = seq(0, 100, 10)) +
  ggtitle("SPS mass match %") +
  theme_bw()
```



From the summary and histogram we can see that the distribution of SPS Mass Matches is much less skewed than that of average reporter ion S/N or isolation interference. This means that whilst the application of thresholds on average reporter ion S/N and isolation interference led to minimal data loss, attempting to impose a threshold on SPS Mass Match represents a much greater trade-off between data quality and quantity. For simplicity, here we choose to use the standard threshold of 65%.

```
## Find out how many PSMs we expect to lose
cp_qf[["psms_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  dplyr::count(SPS.Mass.Matches.in.Percent < 65)
```

```
## # A tibble: 2 x 2
##   `SPS.Mass.Matches.in.Percent < 65`     n
##   <lgl>                              <int>
## 1 FALSE                              21631
## 2 TRUE                               20007
```

```
## Drop these rows from the data
cp_qf <- cp_qf %>%
  filterFeatures(~ SPS.Mass.Matches.in.Percent >= 65,
                 na.rm = TRUE,
                 i = "psms_filtered")
```

Assessing the impact of data-specific filtering

As we did after the non-specific cleaning steps, we check to see how many PSMs, peptides and proteins have been removed throughout the in-depth data-specific filtering.

```
## Summarize the effect of data-specific filtering

## Determine the number and proportion of PSMs removed
psms_remaining_2 <- cp_qf[["psms_filtered"]] %>%
  nrow() %>%
  as.numeric()

psms_removed_2 <- psms_remaining - psms_remaining_2
psms_removed_prop_2 <- ((psms_removed_2 / original_psms) * 100) %>%
  round(digits = 2)


## Determine number and proportion of peptides removed
peps_remaining_2 <- rowData(cp_qf[["psms_filtered"]])$Sequence %>%
  unique() %>%
  length() %>%
  as.numeric()

peps_removed_2 <- peps_remaining - peps_remaining_2
peps_removed_prop_2 <- ((peps_removed_2 / original_peps) * 100) %>%
  round(digits = 2)


## Determine number and proportion of proteins removed
prots_remaining_2 <- cp_qf[["psms_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(Master.Protein.Accessions) %>%
  unique() %>%
  length() %>%
  as.numeric()

prots_removed_2 <- prots_remaining - prots_remaining_2
prots_removed_prop_2 <- ((prots_removed_2 / original_prots) * 100) %>%
  round(digits = 2)
```

```
## Print as a table
data.frame("Feature" = c("PSMs",
                         "Peptides",
                         "Proteins"),
           "Number lost" = c(psms_removed_2,
                             peps_removed_2,
                             prots_removed_2),
           "Percentage lost" = c(psms_removed_prop_2,
                                 peps_removed_prop_2,
                                 prots_removed_prop_2))
```

```
##    Feature Number.lost Percentage.lost
## 1     PSMs       21135           43.28
## 2 Peptides        9861           37.97
## 3 Proteins         998           19.80
```

Managing missing data

Having finished the data cleaning at the PSM-level, the final step is to deal with missing data. Missing values represent a common challenge in quantitative proteomics and there is no consensus within the literature on how this challenge should be addressed. Indeed, missing values fall into different categories based on the reason they were generated, and each category is best dealt with in a different way. There are three main categories of missing data: missing completely at random (MCAR), missing at random (MAR) and missing not at random (MNAR). Within proteomics, values which are MCAR arise due to technical variation or stochastic fluctuations and emerge in a uniform, intensity-independent distribution. Examples include values for peptides which cannot be consistently identified or are unable to be efficiently ionized. By contrast, MNAR values are expected to occur in an intensity-dependent manner due to the presence of peptides at abundances below the limit of detection [22, 23, 17]. In many cases this is due to the biological condition being evaluated, for example the cell type or treatment applied.

To simplify this process, we consider the management of missing data in three steps. The first step is to determine the presence and pattern of missing values within the data. Next, we filter out data which exceed the desired proportion of missing values. This includes removing PSMs with a greater number of missing values across samples than we deem acceptable, as well as whole samples in cases where the proportion of missing values is substantially higher than the average. Finally, imputation can be used to replace any remaining NA values within the dataset. This final step is optional and can equally be done prior to filtering if the user wishes to impute all missing values without removing any PSMs, although this is not recommended. Further, whilst it is possible to complete such steps at the peptide- or protein-level, we advise management of missing values at the lowest data level to minimize the effect of implicit imputation during aggregation.

Exploring the presence of missing values

First, to determine the presence of missing values in the PSM-level data we use the `nNA` function within the `QFeatures` infrastructure. This function will return the absolute number (nNA) and proportion (pNA) of missing values both per sample and as an average. Importantly, alternative third-party software may output missing values in formats other than NA, such as zero, or infinite. In such cases, missing values can be converted directly into NA values through use of the `zeroIsNA` or `infIsNA` functions within the `QFeatures` infrastructure.

```
## Determine whether there are any NA values in the data
cp_qf[["psms_filtered"]] %>%
  assay() %>%
  anyNA()
```

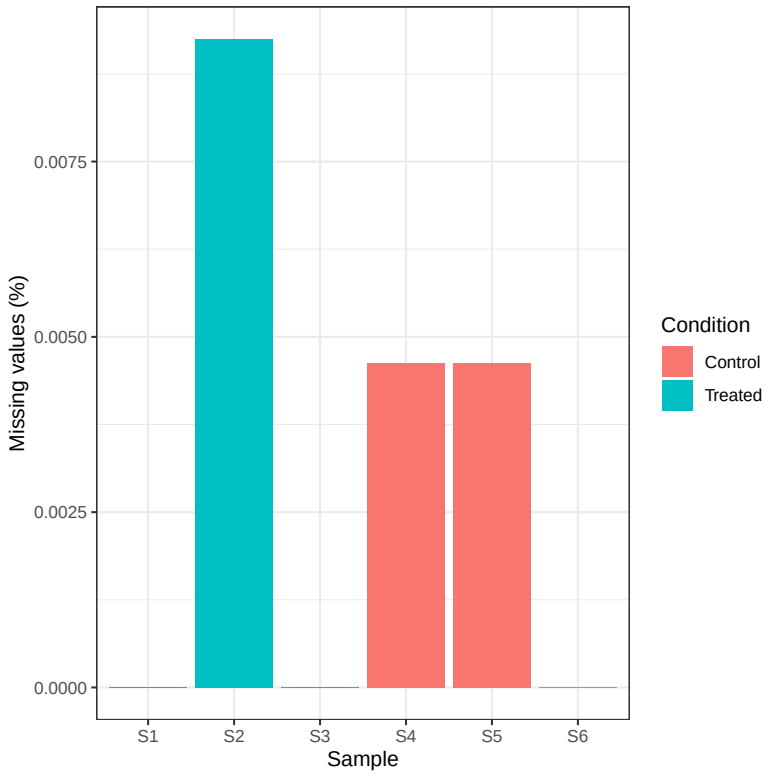
```
## [1] TRUE
```

```
## Determine the amount and distribution of NA values in the data
cp_qf[["psms_filtered"]] %>%
  nNA()
```

```
## $nNA
## DataFrame with 1 row and 2 columns
##         nNA       pNA
##   <integer> <numeric>
## 1         4 3.082e-05
##
## $nNArows
## DataFrame with 21631 rows and 3 columns
##              name       nNA       pNA
##       <character> <integer> <numeric>
## 1              13         0         0
## 2              20         0         0
## 3              25         0         0
## 4              26         0         0
## 5              29         0         0
## ...           ...       ...       ...
## 21627       48786         0         0
## 21628       48792         0         0
## 21629       48797         0         0
## 21630       48810         0         0
## 21631       48819         0         0
##
## $nNAcols
## DataFrame with 6 rows and 3 columns
##          name       nNA         pNA
##   <character> <integer>   <numeric>
## 1          S5         1 4.62299e-05
## 2          S2         2 9.24599e-05
## 3          S1         0 0.00000e+00
## 4          S4         1 4.62299e-05
## 5          S6         0 0.00000e+00
## 6          S3         0 0.00000e+00
```

We can see that the proportion of missing values in our data is only 3e-05, corresponding to 4 NA values. This low proportion is due to a combination of the TMT labelling strategy and the stringent PSM quality control filtering. In particular, co-isolation interference when using TMT labels often results in very low quantification values for peptides which should actually be missing or 'NA'. Nevertheless, we continue and check for sample-specific bias in the distribution of NAs by plotting a simple histogram. We will plot percentage instead of proportion as this is easier to visualise. We also use colour to indicate the condition of each sample as to check for condition-specific bias.

```
## Plot histogram to visualize the distribution of NAs
nNA(cp_qf[["psms_filtered"]])$nNAcols %>%
  as_tibble() %>%
  mutate(Condition = c("Control", "Treated", "Treated",
                       "Control", "Control", "Treated")) %>%
  ggplot(aes(x = name, y = (pNA * 100 ), group = Condition, fill = Condition)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Sample", y = "Missing values (%)") +
  theme_bw()
```



The proportion of missing values is sufficiently low that none of the samples need be removed. Further, there is no sample- or condition-specific bias in the data. We can get more information about the PSMs with NA values using the code below.

```
## Find out the range of missing values per PSM
nNA(cp_qf[["psms_filtered"]])$nNArows$nNA %>%
  table()
```

```
## .
##     0     1
## 21627     4
```

From this output we can see that the maximum number of NA values per PSM is one. This information is useful to know as it may inform the filtering strategy in the next step.

```
## Get indices of rows which contain NA
rows_with_na_indices <- which(nNA(cp_qf[["psms_filtered"]])$nNArows$nNA != 0)

## Subset rows with NA
rows_with_na <- cp_qf[["psms_filtered"]][rows_with_na_indices, ]

## Inspect rows with NA
assay(rows_with_na)
```

```
##         S5   S2   S1   S4   S6   S3
## 12087   NA 17.0 11.0 22.1 30.6 13.3
## 30824 69.7   NA 45.0 66.7 62.1 43.1
## 30846 65.5   NA 34.3 56.8 57.2 47.9
## 44791  3.8 28.7 22.8   NA 12.2 19.6
```

Filtering out missing values

First we apply some standard filtering. Typically, it is desirable to remove features, here PSMs, with greater than 20% missing values. We can do this using the `filterNA` function in `QFeatures`, as outlined below. We pass the function the `SummarizedExperiment` and use the `pNA =` argument to specify the maximum proportion of NA values to allow.

```
## Check how many PSMs we will remove
nNA(cp_qf[["psms_filtered"]])$nNArows %>%
  as_tibble() %>%
  dplyr::count(pNA >= 0.2)
```

```
## # A tibble: 1 x 2
##   `pNA >= 0.2`     n
##   <lgl>        <int>
## 1 FALSE        21631
```

Although the use-case data does not contain any PSMs with a higher proportion of missing values than 0.2, we demonstrate how to apply the desired filter below.

```
## Remove PSMs with more than 20 % (0.2) NA values
cp_qf <- cp_qf %>%
  filterNA(pNA = 0.2,
           i = "psms_filtered")
```

Since previous exploration of missing data did not reveal any sample with an excessive number of NA values, we do not need to remove any samples from the analysis.

Although not covered here, users may wish to carry out condition-specific filtering in cases where the exploration of missing values revealed a condition- specific bias, or where the experimental question requires. This would be the case, for example, if one condition was transfected to express proteins of interest whilst the control condition lacked these proteins. Filtering of both conditions together could, therefore, lead to the removal of proteins of interest.

Imputation (optional)

The final step is to consider whether to impute the remaining missing values within the data. Imputation refers to the replacement of missing values with probable values. Since imputation requires complex assumptions and can have substantial effects on downstream statistical analysis, we here choose to skip imputation. This is reasonable given that we only have 3 missing values at the PSM-level, and that some of these will likely be removed by aggregation. A more in-depth discussion of imputation will be provided below in the workflow.

Summary of PSM data cleaning

Thus far we have checked that the experimental data we are using is of high quality by visualizing the raw data and calculating TMT labelling efficiency. We then carried out non-specific data cleaning, data-specific filtering steps and management of missing data. Here, we present a combined summary of these PSM processing steps.

```
## Determine final number of PSMs, peptides and master proteins
psms_final <- cp_qf[["psms_filtered"]] %>%
  nrow() %>%
  as.numeric()

psms_removed_total <- original_psms - psms_final
psms_removed_total_prop <- ((psms_removed_total / original_psms) * 100) %>%
  round(digits = 2)

peps_final <- cp_qf[["psms_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(Sequence) %>%
  unique() %>%
  length() %>%
  as.numeric()

peps_removed_total <- original_peps - peps_final
peps_removed_total_prop <- ((peps_removed_total / original_peps) * 100) %>%
  round(digits = 2)

prots_final <- cp_qf[["psms_filtered"]] %>%
```

```
  rowData() %>%
  as_tibble() %>%
  pull(Master.Protein.Accessions) %>%
  unique() %>%
  length() %>%
  as.numeric()

prots_removed_total <- original_prots - prots_final
prots_removed_total_prop <- ((prots_removed_total / original_prots) * 100) %>%
  round(digits = 2)

## Print as table
data.frame("Feature" = c("PSMs",
                         "Peptides",
                         "Proteins"),
           "Number lost" = c(psms_removed_total,
                             peps_removed_total,
                             prots_removed_total),
           "Percentage lost" = c(psms_removed_total_prop,
                                 peps_removed_total_prop,
                                 prots_removed_total_prop),
           "Number remaining" = c(psms_final,
                                  peps_final,
                                  prots_final))
```

```
##    Feature Number.lost Percentage.lost Number.remaining
## 1     PSMs       27201           55.70            21631
## 2 Peptides       11786           45.38            14183
## 3 Proteins        1810           35.91             3230
```

Logarithmic transformation of quantitative data

Once satisfied that the PSM-level data is clean and of high quality, the PSM-level quantitative data is log transformed. log2 transformation is a standard step when dealing with quantitative proteomics data since protein abundances are dramatically skewed towards zero. Such a skewed distribution is to be expected given that the majority of cellular proteins present at any one time are of relatively low abundance, whilst only a few highly abundant proteins exist. To perform the logarithmic transformation and generate normally distributed data we pass the PSM-level data in the `QFeatures` object to the `logTransform` function, as per the below code chunk.

```
## log2 transform quantitative data
cp_qf <- logTransform(object = cp_qf,
                      base = 2,
                      i = "psms_filtered",
                      name = "log_psms")

## Verify
cp_qf
```

```
## An instance of class QFeatures containing 3 assays:
##  [1] psms_raw: SummarizedExperiment with 48832 rows and 6 columns
##  [2] psms_filtered: SummarizedExperiment with 21631 rows and 6 columns
##  [3] log_psms: SummarizedExperiment with 21631 rows and 6 columns
```

Aggregation of PSMs to proteins

For the aggregation itself we use the `aggregateFeatures` function and provide the base level from which we wish to aggregate, the log PSM-level data in this case. We also tell the function which column to aggregate which is specified by the `fcol` argument. We will first aggregate from PSM to peptide to create explicit `QFeatures` links. This means grouping by PSM "Sequence".

As well as grouping PSMs according to their peptide sequence, the quantitative values for each PSM must be aggregated into a single peptide-level value. The default aggregation method within `aggregateFeatures`

is the `robustSummary` function from the `MsCoreUtils` package [19]. This method is a form of robust regression and is described in detail elsewhere [24]. Nevertheless, the user must decide which aggregation method is most appropriate for their data and biological question. Further, an understanding of the selected method is critical given that aggregation is a form of implicit imputation and has substantial effects on the downstream data. Indeed, aggregation methods have different ways of dealing with missing data, either by removal or propagation. Options of aggregation methods within the `aggregateFeatures` function include `MsCoreUtils::medianPolish`, `MsCoreUtils::robustSummary`, `base::colMeans`, `base::colSums`, and `matrixStats::colMedians`. Users should also be aware that some methods have specific input requirements. For example, `robustSummary` assumes that intensities have already been log transformed.

Aggregating using robust summarization

Here, we use `robustSummary` to aggregate from PSM to peptide-level. This method is currently considered to be state-of-the-art as it is more robust against outliers than other aggregation methods [24, 25]. We also include `na.rm = TRUE` to exclude any NA values prior to completing the summarization.

```
## Aggregate PSM to peptide
cp_qf <- aggregateFeatures(cp_qf,
                           i = "log_psms",
                           fcol = "Sequence",
                           name = "log_peptides",
                           fun = MsCoreUtils::robustSummary,
                           na.rm = TRUE)
```

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

```
## Verify
cp_qf
```

```
## An instance of class QFeatures containing 4 assays:
##  [1] psms_raw: SummarizedExperiment with 48832 rows and 6 columns
##  [2] psms_filtered: SummarizedExperiment with 21631 rows and 6 columns
##  [3] log_psms: SummarizedExperiment with 21631 rows and 6 columns
##  [4] log_peptides: SummarizedExperiment with 14183 rows and 6 columns
```

We are now left with a `QFeatures` object holding the PSM and peptide-level data in their own `SummarizedExperiments`. Importantly, an explicit link has been maintained between the two levels and this makes it possible to gain information about all PSMs that were aggregated into a peptide.

Considerations for aggregating non-imputed data

If users did not impute prior to aggregation, NA values within the PSM-level data may have propagated into NaN values. This is because peptides only supported by PSMs containing missing values would not have any quantitative value to which a sum or median function, for example, can be applied. Therefore, we check for NaN and convert back to NA values to facilitate compatibility with downstream processing.

```
## Confirm the presence of NaN
assay(cp_qf[["log_peptides"]]) %>%
  is.nan() %>%
  table()
```

```
## .
## FALSE
## 85098
```

```
## Replace NaN with NA
assay(cp_qf[["log_peptides"]])[is.nan(assay(cp_qf[["log_peptides"]]))] <- NA
```

Next, using the same approach as above, we use the `aggregateFeatures` function to assemble the peptides into proteins. As before, we must pass several arguments to the function. Namely, the `QFeatures` object i.e. `cp_qf`, the data level we wish to aggregation from i.e. `log_peptides`, the column of the `rowData` defining how to aggregate the features i.e. by `"Master.Protein.Accessions"` and a name for the new data level e.g. `"log_proteins"`. We again choose to use `robustSummary` as our aggregation method and we pass `na.rm = TRUE` to ignore NA values. Users can type `?aggregateFeatures` to see more information. Users should be aware that peptides are grouped by their master protein accession and, therefore, downstream differential expression analysis will consider protein groups rather than individual proteins.

```
## Aggregate peptides to protein
cp_qf <- aggregateFeatures(cp_qf,
                           i = "log_peptides",
                           fcol = "Master.Protein.Accessions",
                           name = "log_proteins",
                           fun = MsCoreUtils::robustSummary,
                           na.rm = TRUE)
```

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

```
## Verify
cp_qf
```

```
## An instance of class QFeatures containing 5 assays:
##  [1] psms_raw: SummarizedExperiment with 48832 rows and 6 columns
##  [2] psms_filtered: SummarizedExperiment with 21631 rows and 6 columns
##  [3] log_psms: SummarizedExperiment with 21631 rows and 6 columns
##  [4] log_peptides: SummarizedExperiment with 14183 rows and 6 columns
##  [5] log_proteins: SummarizedExperiment with 3230 rows and 6 columns
```

Following aggregation, we have a total of 3230 proteins remaining within the data.

Normalization of quantitative data

After transforming the data, we normalize the protein-level abundances. Normalization is a process of correction whereby quantitative data is returned to its original, or 'normal', state. In expression proteomics, the aim of post- acquisition data normalization is to minimize the biases that arises due to experimental error and technological variation. Specifically, the removal of random variation and batch effects will allow samples to be aligned prior to downstream analysis. Importantly, however, users must also be aware of any normalization that has taken place within their sample preparation, as this will ultimately influence the presence of differentially abundant proteins downstream. An extensive review on normalization strategies, both experimental and computational, is provided in [26].

Unfortunately, there is not currently a single normalization method which performs best for all quantitative proteomics datasets. Within the Bioconductor packages, however, exists `NormalyzerDE`, a tool for evaluating different normalization methods [27]. By passing a `SummarizedExperiment` object to the `normalyzer` function it is possible to generate a report comparing common normalization strategies, such as total intensity (TI), median intensity (MedI), average intensity (AI), quantile (from the `preprocessCore` package)[28], NormFinder (NM)[29], Variance Stabilising Normalization (VSN, from the `vsn` package)[30], Robust Linear Regression (RLR), and LOESS (from the `limma` package)[31]. A number of qualitative and quantitative evaluation measures are provided within the report, including total intensity, Pooled intragroup Coefficient of Variation (PCV), Pooled intragroup Median Absolute Deviation (PMDA), CV-intensity plots, MA-plots, Pearson and Spearman correlation.

Normalyzer accepts intensity data in a raw format, prior to log transformation. Therefore, we first generate a protein-level `SummarizedExperiment` from our PSM-level data prior to transformation.

```
## Aggregate from PSM directly to protein
cp_qf <- aggregateFeatures(cp_qf,
                           i = "psms_filtered",
                           fcol = "Master.Protein.Accessions",
                           name = "proteins_direct",
                           fun = MsCoreUtils::robustSummary,
                           na.rm = TRUE)
```

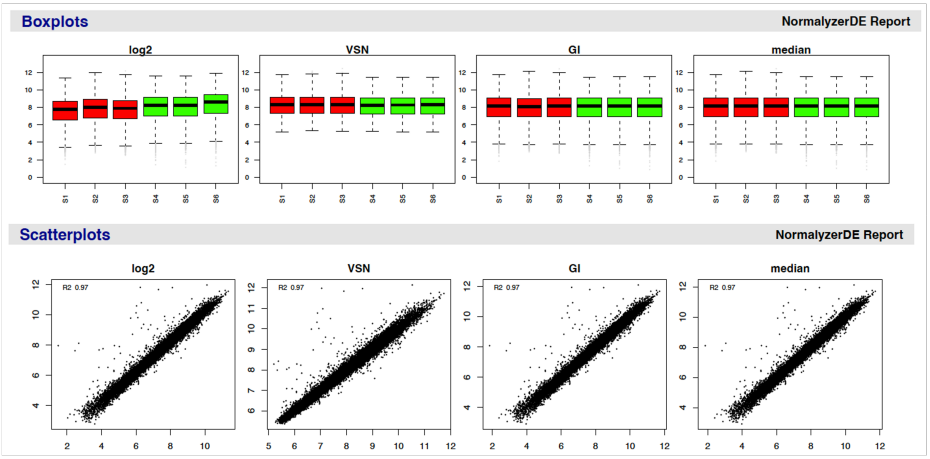


Figure 4. Example of plots generated by the normalyzer tool and provided in the .pdf report. Boxplots (top) and scatterplots (bottom) are two of the evaluation measures within the normalyzer report. Samples are grouped based on their condition to provide users with an easy way to evaluate the suitability of different normalization methods for their data. The log2 data can be used as a reference to compare the data pre- and post-normalization.

Hence, we will use the "proteins_direct" `SummarizedExperiment` here and the function will do the log2 transformation for us. A second important consideration is that missing values must be denoted 'NA', not zero, NaN or infinite. We can pass the `SummarizedExperiment` containing the protein data to the `normalyzer` function. With this, we provide a name for the report and the directory in which to save the report. The `normalyzer` function also expects two pieces of information, the sample name and corresponding experimental group. We previously annotated the data with this information through the sample and condition columns of the `colData`, so we tell the `normalyzer` function to look here.

```
## Generate normalyzer report
normalyzer(jobName = "normalyzer",
           experimentObj = cp_qf[["proteins_direct"]],
           sampleColName = "sample",
           groupColName = "condition",
           outputDir = ".")
```

The function will take a few minutes to run, particularly if there are many samples. Once complete, the report can be accessed as a `.pdf` file containing plots such as those displayed in (Figure 4).

Since the normalyzer report did not indicate any superior normalization method in this case, we will apply a center median approach here. To do this, we pass the log transformed protein-level data to the `normalize` function in `QFeatures`. We specify the method of normalization that we wish to apply i.e. `method = "center.median"` and name the new data level e.g. `name = "log_norm_proteins"`. Of note, for users who wish to apply VSN normalization the raw protein data must be passed (prior to any log transformation) as the log transformation is done internally when specify `method = "vsn"`. All other methods require users to explicitly perform log transformation on their data before use. More details can be found in the `QFeatures` documentation, please type `help("normalize,QFeatures-method")`.

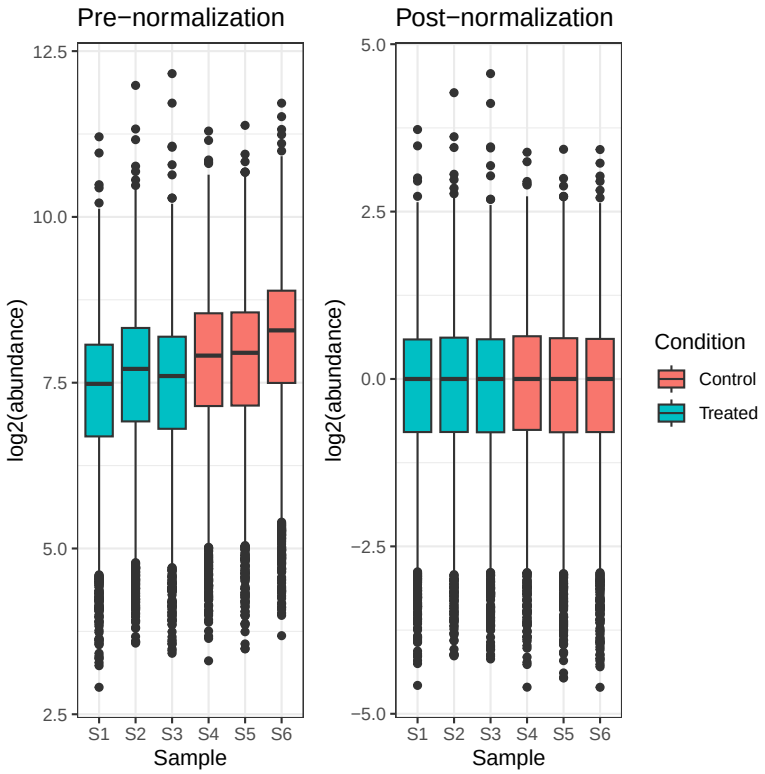
```
## normalize the log transformed peptide data
cp_qf <- normalize(cp_qf,
                   i = "log_proteins",
                   name = "log_norm_proteins",
                   method = "center.median")

## Verify
cp_qf
```

```
## An instance of class QFeatures containing 7 assays:
##  [1] psms_raw: SummarizedExperiment with 48832 rows and 6 columns
##  [2] psms_filtered: SummarizedExperiment with 21631 rows and 6 columns
##  [3] log_psms: SummarizedExperiment with 21631 rows and 6 columns
##  [4] log_peptides: SummarizedExperiment with 14183 rows and 6 columns
##  [5] log_proteins: SummarizedExperiment with 3230 rows and 6 columns
```

```
##  [6] proteins_direct: SummarizedExperiment with 3230 rows and 6 columns
##  [7] log_norm_proteins: SummarizedExperiment with 3230 rows and 6 columns
```

To evaluate the effect of normalization we plot a simple boxplot.

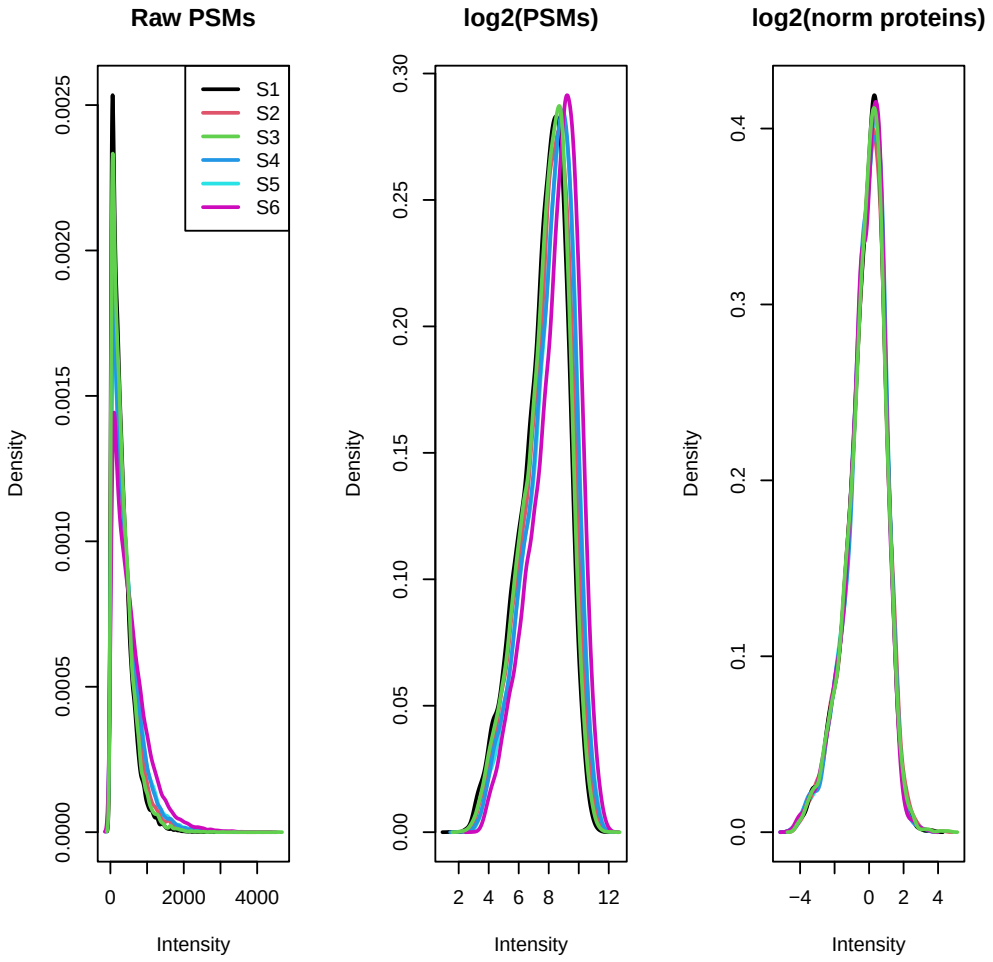
```
## Evaluate the effect of data normalization
pre_norm <- cp_qf[["log_proteins"]] %>%
  assay() %>%
  longFormat() %>%
  mutate(Condition = ifelse(colname %in% c("S1", "S2", "S3"),
                            "Treated", "Control"),
         colname = factor(colname, levels = paste0("S", 1:6))) %>%
  ggplot(aes(x = colname, y = value, fill = Condition)) +
  geom_boxplot() +
  labs(x = "Sample", y = "log2(abundance)", title = "Pre-normalization") +
  theme_bw()

post_norm <- cp_qf[["log_norm_proteins"]] %>%
  assay() %>%
  longFormat() %>%
  mutate(Condition = ifelse(colname %in% c("S1", "S2", "S3"),
                            "Treated", "Control"),
         colname = factor(colname, levels = paste0("S", 1:6))) %>%
  ggplot(aes(x = colname, y = value, fill = Condition)) +
  geom_boxplot() +
  labs(x = "Sample", y = "log2(abundance)", title = "Post-normalization") +
  theme_bw()

(pre_norm + theme(legend.position = "none")) +
  post_norm & plot_layout(guides = "collect")
```



We can now generate a density plot to help us visualize what the process of log transformation and normalization has done to the data. This is done using the `plotDensities` function from the `limma` package.

```
## visualize the process of log transformation and normalization
par(mfrow = c(1, 3))
```

```
cp_qf[["psms_filtered"]] %>%
  assay() %>%
  plotDensities(legend = "topright",
                main = "Raw PSMs")

cp_qf[["log_psms"]] %>%
  assay() %>%
  plotDensities(legend = FALSE,
                main = "log2(PSMs)")

cp_qf[["log_norm_proteins"]] %>%
  assay() %>%
  plotDensities(legend = FALSE,
                main = "log2(norm proteins)")
```



Exploration of data using QFeatures links

Creating assay links

After completing all data pre-processing, we now add explicit links between our final protein-level data and the raw PSM-level data which we created as an untouched copy. This allows us to investigate all data corresponding to the final proteins, including the data that has since been removed. To do this, we use the `addAssayLinks` function, demonstrated below. We can check that the assay links have been generated correctly by passing our `QFeatures` object to the `AssayLink` function along with the assay of interest (`i =`).

```
## Add assay links from log_norm_proteins to psms_raw
cp_qf <- addAssayLink(object = cp_qf,
                      from = "psms_raw",
                      to = "log_norm_proteins",
                      varFrom = "Master.Protein.Accessions",
```

```
                      varTo = "Master.Protein.Accessions")

## Verify
assayLink(cp_qf,
          i = "log_norm_proteins")
```

```
## AssayLink for assay <log_norm_proteins>
## [from:psms_raw|fcol:Master.Protein.Accessions|hits:42604]
```

visualizing aggregation

One of the characteristic attributes of the `QFeatures` infrastructure is that explicit links have been maintained throughout the aggregation process. This means that we can now access all data corresponding to a protein, its component peptides and PSMs. One way to do this is through the use of the `subsetByFeature` function which will return a new `QFeatures` object containing data for the desired feature across all levels. For example, if we wish to subset information about the protein "Q01581", that is hydroxymethylglutaryl-CoA synthase, we could use the following code:

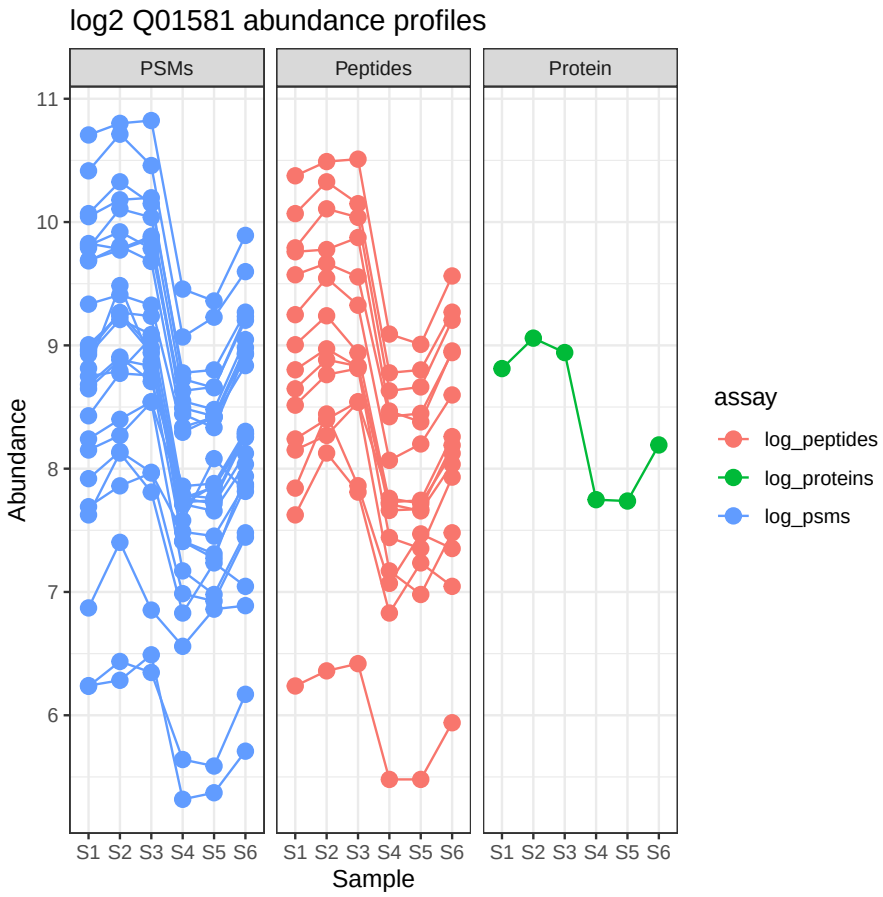
```
## Subset all data linked to the protein with accession Q01581
Q01581 <- subsetByFeature(cp_qf, "Q01581")

## Verify
Q01581
```

```
## An instance of class QFeatures containing 7 assays:
##  [1] psms_raw: SummarizedExperiment with 42 rows and 6 columns
##  [2] psms_filtered: SummarizedExperiment with 27 rows and 6 columns
##  [3] log_psms: SummarizedExperiment with 27 rows and 6 columns
##  [4] log_peptides: SummarizedExperiment with 15 rows and 6 columns
##  [5] log_proteins: SummarizedExperiment with 1 rows and 6 columns
##  [6] proteins_direct: SummarizedExperiment with 1 rows and 6 columns
##  [7] log_norm_proteins: SummarizedExperiment with 1 rows and 6 columns
```

We find that in this data the protein Q01581 has 15 peptides and 27 supporting its identification and quantitation. We also see that the original data prior to processing contained 42 PSMs in support of this protein.

Further, we can visualize the process of aggregation that has led to the protein- level abundance data for Q01581, as demonstrated below. Of note, this plot shows the protein data prior to transformation.

```
## Define conditions
treament <- c("S1", "S2", "S3")
control <- c("S4", "S5", "S6")

## Plot abundance distributions across samples at PSM, peptide and protein-level
Q01581[, , c("log_psms", "log_peptides", "log_proteins")] %>%
  longFormat() %>%
  as_tibble() %>%
  mutate(assay_order = factor(
    assay,
    levels = c("log_psms", "log_peptides", "log_proteins"),
    labels = c("PSMs", "Peptides", "Protein")),
    condition = ifelse(colname %in% control, "control", "treatment")) %>%
  ggplot(aes(x = colname, y = value, colour = assay)) +
  geom_point(size = 3) +
  geom_line(aes(group = rowname)) +
  scale_x_discrete(limits = paste0("S", 1:6)) +
  facet_wrap(~assay_order) +
  labs(x = "Sample", y = "Abundance") +
  ggtitle("log2 Q01581 abundance profiles") +
  theme_bw()
```

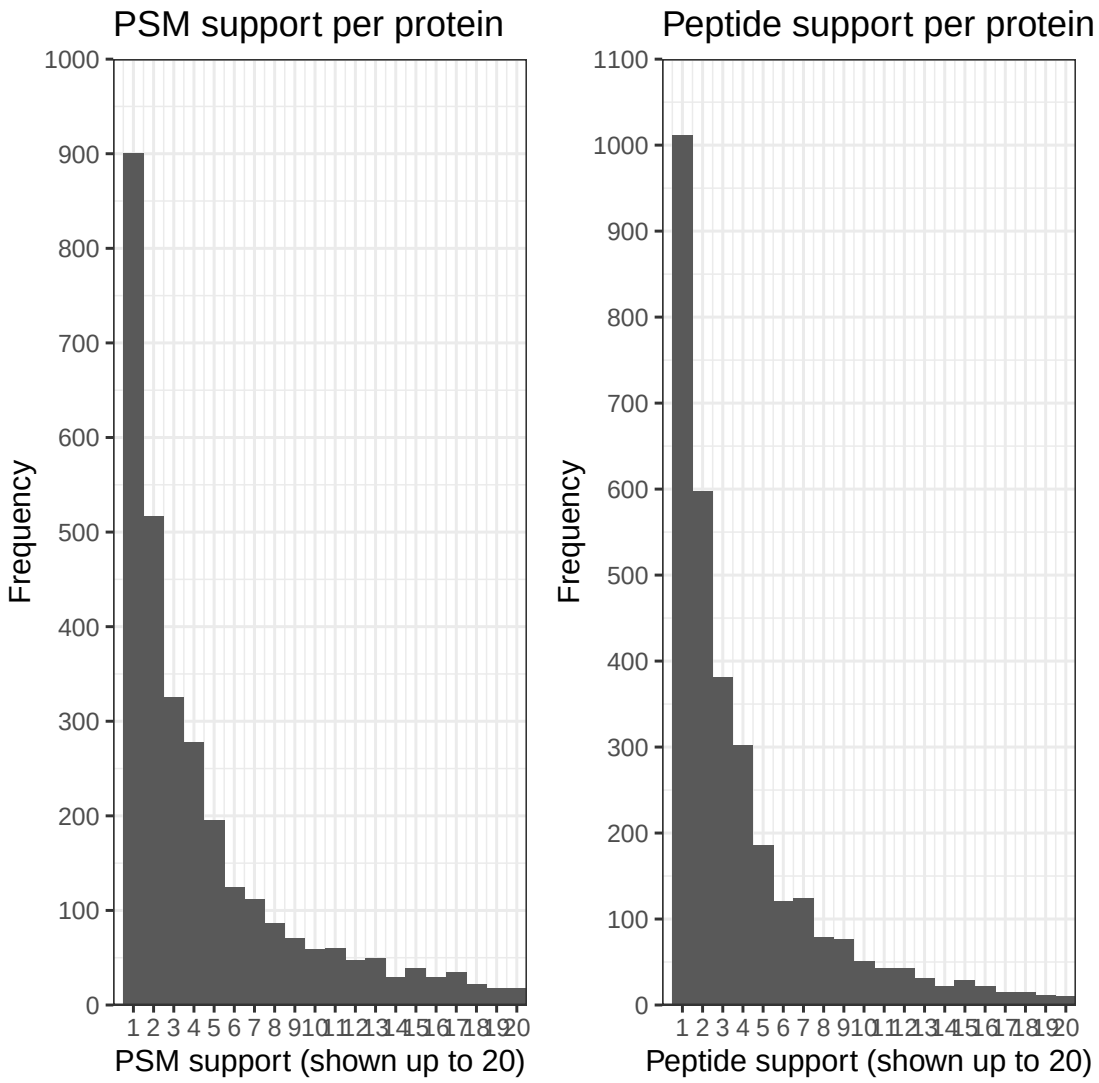


Determining PSM and peptide support

Another benefit of the explicit links maintained within a `QFeatures` object is the ease at which we can determine PSM and peptide support per protein. When applying the `aggregateFeatures` function a column, termed `".n"`, is created within the rowData of the new `SummarizedExperiment`. This column indicates how many lower-level features were aggregated into each new higher-level feature. Hence, `".n"` in the peptide-level data represents how many PSMs were aggregated into a peptide, whilst in the protein-level data it tells us how many peptides were grouped into a master protein. For ease of plotting, we will use the `"proteins_direct"` data generated above. Since this data was generated via direct aggregation of PSM to protein, `".n"` this will tell us PSM support per protein. We plot these data as simple histograms.

```
## Plot PSM support per protein - .n in the proteins_direct SE
psm_per_protein <- cp_qf[["proteins_direct"]] %>%
  rowData() %>%
  as_tibble() %>%
  ggplot(aes(x = .n)) +
  geom_histogram(binwidth = 1, boundary = 0.5) +
  labs(x = "PSM support (shown up to 20)",
       y = "Frequency") +
  scale_x_continuous(expand = c(0, 0),
                     limits = c(0, 20.5),
                     breaks = seq(1, 20, 1)) +
  scale_y_continuous(expand = c(0, 0),
                     limits = c(0, 1000),
                     breaks = seq(0, 1000, 100)) +
  ggtitle("PSM support per protein") +
  theme_bw()

## Plot peptide support per protein - .n in the proteins SE
peptide_per_protein <- cp_qf[["log_proteins"]] %>%
  rowData() %>%
  as_tibble() %>%
  ggplot(aes(x = .n)) +
```

```
  geom_histogram(binwidth = 1, boundary = 0.5) +
  labs(x = "Peptide support (shown up to 20)",
       y = "Frequency") +
  scale_x_continuous(expand = c(0, 0),
                     limits = c(0, 20.5),
                     breaks = seq(1, 20, 1)) +
  scale_y_continuous(expand = c(0, 0),
                     limits = c(0, 1100),
                     breaks = seq(0, 1100, 100)) +
  ggtitle("Peptide support per protein") +
  theme_bw()

psm_per_protein + peptide_per_protein
```



At this point, users may wish to include additional quality control filtering based on PSM and/or peptide support per protein. Given the extensive quality control filtering already applied in this workflow, we decide not to remove additional proteins based on PSM or peptide support.

Data export

Finally, we save the protein-level data and export the `QFeatures` object into an `.rda` file so that we can re-load it later at convenience.

```
## Save protein-level SE
cp_proteins <- cp_qf[["log_norm_proteins"]]
```

```
## Export the final TMT QFeatures object
save(cp_qf, file = "cp_qf.rda")
```

Label-free data processing workflow

Having discussed the processing of quantitative TMT-labelled data, we now move on to consider that of label-free quantitative (LFQ) data. As described previously, the cell culture supernatant fractions of triplicate control and treated HEK293 cells were kept label-free. As such, each sample was analyzed using an independent mass spectrometry run without pre-fractionation. Again, a two-hour gradient in an Orbitrap Lumos Tribrid mass spectrometer coupled to an UltiMate 3000 HPLC system was applied. Given that much of the TMT pre-processing workflow also applies to label-free data, we only discuss steps which are different to those previously described. Readers are advised to refer to the TMT processing workflow for a more in-depth explanation of any shared steps.

Identification search using Proteome Discoverer

As was the case for TMT labelled cell pellets, raw LFQ data from supernatant samples was searched using Proteome Discoverer 2.5. The workflows for this identification search are provided in the supplementary materials with an additional explanation of key parameters in the appendix. To begin processing LFQ data, users should export a peptide-level `.txt` file from the results of their identification search.

Data import, housekeeping and exploration

Unlike the TMT-labelled use-case data which was processed from the PSM-level, the label-free use-case data can only be considered from the peptide-level up. This is because a retention time alignment algorithm (equivalent to match between runs) was applied to the PSM-level data. This means that peptides can be identified in samples even without a corresponding PSM, simply by sharing feature information across runs.

Importing data into a QFeatures object

We locate the PeptideGroups `.txt` file and upload this into a `data.frame` container in the same way as before. Since the samples are already stored in the correct order, we simply identify the quantitative columns by their indices.

```
## Locate the PeptideGroups .txt file
sn_peptide <- "supernatant_lfq_results_peptides.txt"

## Import into a data.frame
sn_df <- read.delim(sn_peptide)

## Identify columns containing quantitative data
sn_df %>%
  names()
```

```
##  [1] "Peptide.Groups.Peptide.Group.ID"
##  [2] "Checked"
##  [3] "Tags"
##  [4] "Confidence"
##  [5] "PSM.Ambiguity"
##  [6] "Sequence"
##  [7] "Modifications"
##  [8] "Modifications.all.possible.sites"
##  [9] "Qvality.PEP"
## [10] "Qvality.q.value"
## [11] "SVM_Score"
## [12] "Number.of.Protein.Groups"
## [13] "Number.of.Proteins"
## [14] "Number.of.PSMs"
## [15] "Master.Protein.Accessions"
## [16] "Master.Protein.Descriptions"
## [17] "Protein.Accessions"
## [18] "Number.of.Missed.Cleavages"
```

```
## [19] "Theo.MHplus.in.Da"
## [20] "Sequence.Length"
## [21] "Abundance.F1.Sample"
## [22] "Abundance.F2.Sample"
## [23] "Abundance.F3.Sample"
## [24] "Abundance.F4.Sample"
## [25] "Abundance.F5.Sample"
## [26] "Abundance.F6.Sample"
## [27] "Abundances.Count.F1.Sample"
## [28] "Abundances.Count.F2.Sample"
## [29] "Abundances.Count.F3.Sample"
## [30] "Abundances.Count.F4.Sample"
## [31] "Abundances.Count.F5.Sample"
## [32] "Abundances.Count.F6.Sample"
## [33] "Quan.Info"
## [34] "Found.in.File.in.F1"
## [35] "Found.in.File.in.F2"
## [36] "Found.in.File.in.F3"
## [37] "Found.in.File.in.F4"
## [38] "Found.in.File.in.F5"
## [39] "Found.in.File.in.F6"
## [40] "Found.in.Sample.in.S1.F1.Sample"
## [41] "Found.in.Sample.in.S2.F2.Sample"
## [42] "Found.in.Sample.in.S3.F3.Sample"
## [43] "Found.in.Sample.in.S4.F4.Sample"
## [44] "Found.in.Sample.in.S5.F5.Sample"
## [45] "Found.in.Sample.in.S6.F6.Sample"
## [46] "Found.in.Sample.Group.in.S1.F1.Sample"
## [47] "Found.in.Sample.Group.in.S2.F2.Sample"
## [48] "Found.in.Sample.Group.in.S3.F3.Sample"
## [49] "Found.in.Sample.Group.in.S4.F4.Sample"
## [50] "Found.in.Sample.Group.in.S5.F5.Sample"
## [51] "Found.in.Sample.Group.in.S6.F6.Sample"
## [52] "Confidence.by.Search.Engine.Sequest.HT"
## [53] "Charge.by.Search.Engine.Sequest.HT"
## [54] "Delta.Score.by.Search.Engine.Sequest.HT"
## [55] "Delta.Cn.by.Search.Engine.Sequest.HT"
## [56] "Rank.by.Search.Engine.Sequest.HT"
## [57] "Search.Engine.Rank.by.Search.Engine.Sequest.HT"
## [58] "Concatenated.Rank.by.Search.Engine.Sequest.HT"
## [59] "mz.in.Da.by.Search.Engine.Sequest.HT"
## [60] "Delta.M.in.ppm.by.Search.Engine.Sequest.HT"
## [61] "Delta.mz.in.Da.by.Search.Engine.Sequest.HT"
## [62] "RT.in.min.by.Search.Engine.Sequest.HT"
## [63] "Percolator.q.Value.by.Search.Engine.Sequest.HT"
## [64] "Percolator.PEP.by.Search.Engine.Sequest.HT"
## [65] "Percolator.SVMScore.by.Search.Engine.Sequest.HT"
## [66] "XCorr.by.Search.Engine.Sequest.HT"
## [67] "Top.Apex.RT.in.min"
```

In the code chunk below, we again use the `readQFeatures` function to create a `QFeatures` object. We find the abundance data is located in columns 21 to 26 and thus pass this to `quantCols`. After import we annotate the `colData`.

```
## Create QFeatures object
sn_qf <- readQFeatures(assayData = sn_df,
                       quantCols = 21:26,
                       name = "peptides_raw")
```

```
## Checking arguments.

## Loading data as a 'SummarizedExperiment' object.

## Formatting sample annotations (colData).
```

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

```
## Clean sample names
colnames(sn_qf[["peptides_raw"]]) <- paste0("S", 1:6)

## Annotate samples
sn_qf$sample <- paste0("S", 1:6)

sn_qf$condition <- rep(c("Treated", "Control"), each = 3)

## Verify and allocate colData to initial SummarizedExperiment
colData(sn_qf)
```

```
## DataFrame with 6 rows and 2 columns
##         sample   condition
##    <character> <character>
## S1          S1     Treated
## S2          S2     Treated
## S3          S3     Treated
## S4          S4     Control
## S5          S5     Control
## S6          S6     Control
```

```
colData(sn_qf[["peptides_raw"]]) <- colData(sn_qf)
```

Preliminary data exploration

Next, we check the names of the features within the peptide-level `rowData`. These features differ from those found at the PSM-level and users should be aware that they have reduced post-search control over the quality of PSMs included in the peptide quantitation, and which method of aggregation is used to define these. Proteome Discoverer uses the sum of PSM quantitative values to calculate peptide-level values. Other third-party softwares may use different methods.

```
## Find out what information was imported
sn_qf[["peptides_raw"]] %>%
  rowData() %>%
  colnames()
```

```
##  [1] "Peptide.Groups.Peptide.Group.ID"
##  [2] "Checked"
##  [3] "Tags"
##  [4] "Confidence"
##  [5] "PSM.Ambiguity"
##  [6] "Sequence"
##  [7] "Modifications"
##  [8] "Modifications.all.possible.sites"
##  [9] "Qvality.PEP"
## [10] "Qvality.q.value"
## [11] "SVM_Score"
## [12] "Number.of.Protein.Groups"
## [13] "Number.of.Proteins"
## [14] "Number.of.PSMs"
## [15] "Master.Protein.Accessions"
## [16] "Master.Protein.Descriptions"
## [17] "Protein.Accessions"
## [18] "Number.of.Missed.Cleavages"
## [19] "Theo.MHplus.in.Da"
## [20] "Sequence.Length"
## [21] "Abundances.Count.F1.Sample"
## [22] "Abundances.Count.F2.Sample"
## [23] "Abundances.Count.F3.Sample"
## [24] "Abundances.Count.F4.Sample"
## [25] "Abundances.Count.F5.Sample"
```

```
## [26] "Abundances.Count.F6.Sample"
## [27] "Quan.Info"
## [28] "Found.in.File.in.F1"
## [29] "Found.in.File.in.F2"
## [30] "Found.in.File.in.F3"
## [31] "Found.in.File.in.F4"
## [32] "Found.in.File.in.F5"
## [33] "Found.in.File.in.F6"
## [34] "Found.in.Sample.in.S1.F1.Sample"
## [35] "Found.in.Sample.in.S2.F2.Sample"
## [36] "Found.in.Sample.in.S3.F3.Sample"
## [37] "Found.in.Sample.in.S4.F4.Sample"
## [38] "Found.in.Sample.in.S5.F5.Sample"
## [39] "Found.in.Sample.in.S6.F6.Sample"
## [40] "Found.in.Sample.Group.in.S1.F1.Sample"
## [41] "Found.in.Sample.Group.in.S2.F2.Sample"
## [42] "Found.in.Sample.Group.in.S3.F3.Sample"
## [43] "Found.in.Sample.Group.in.S4.F4.Sample"
## [44] "Found.in.Sample.Group.in.S5.F5.Sample"
## [45] "Found.in.Sample.Group.in.S6.F6.Sample"
## [46] "Confidence.by.Search.Engine.Sequest.HT"
## [47] "Charge.by.Search.Engine.Sequest.HT"
## [48] "Delta.Score.by.Search.Engine.Sequest.HT"
## [49] "Delta.Cn.by.Search.Engine.Sequest.HT"
## [50] "Rank.by.Search.Engine.Sequest.HT"
## [51] "Search.Engine.Rank.by.Search.Engine.Sequest.HT"
## [52] "Concatenated.Rank.by.Search.Engine.Sequest.HT"
## [53] "mz.in.Da.by.Search.Engine.Sequest.HT"
## [54] "Delta.M.in.ppm.by.Search.Engine.Sequest.HT"
## [55] "Delta.mz.in.Da.by.Search.Engine.Sequest.HT"
## [56] "RT.in.min.by.Search.Engine.Sequest.HT"
## [57] "Percolator.q.Value.by.Search.Engine.Sequest.HT"
## [58] "Percolator.PEP.by.Search.Engine.Sequest.HT"
## [59] "Percolator.SVMScore.by.Search.Engine.Sequest.HT"
## [60] "XCorr.by.Search.Engine.Sequest.HT"
## [61] "Top.Apex.RT.in.min"
```

We also determine the number of PSMs, peptides and proteins represented within the initial data. Since identical peptide sequences with different modifications are stored as separate entities, the output of `dim` will not tell us the number of peptides. Instead, we need to consider only unique peptide sequence entries, as demonstrated in the code chunk below.

```
## Determine the number of PSMs
original_psms <- sn_qf[["peptides_raw"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(Number.of.PSMs) %>%
  sum()

## Determine the number of peptides
original_peps <- sn_qf[["peptides_raw"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(Sequence) %>%
  unique() %>%
  length() %>%
  as.numeric()

## Determine the number of proteins
original_prots <- sn_qf[["peptides_raw"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(Master.Protein.Accessions) %>%
  unique() %>%
  length() %>%
```

```
  as.numeric()

## View
original_psms
```

```
## [1] 144302
```

```
original_peps
```

```
## [1] 20312
```

```
original_prots
```

```
## [1] 3941
```

Thus, the search identified 144302 PSMs corresponding to 20312 peptides and 3941 proteins. Finally, we take a look at some of the key parameters applied during the identification search. This is an important verification step, particularly for those using publicly available data with limited access to parameter settings.

```
## Check missed cleavages
sn_qf[["peptides_raw"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(Number.of.Missed.Cleavages) %>%
  table()
```

```
## .
##     0     1     2
## 22055  1248    72
```

```
## Check precursor mass tolerance
sn_qf[["peptides_raw"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(Delta.M.in.ppm.by.Search.Engine.Sequest.HT) %>%
  summary()
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -9.9600 -0.2500  0.1500  0.6576  0.6900  9.9900
```

```
## Check fragment mass tolerance
sn_qf[["peptides_raw"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(Delta.mz.in.Da.by.Search.Engine.Sequest.HT) %>%
  summary()
```

```
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -0.0113400 -0.0001400  0.0000900  0.0006618  0.0004800  0.0142300
```

```
## Check peptide confidence allocations
sn_qf[["peptides_raw"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(Confidence) %>%
  table()
```

```
## .
##  High
## 23375
```

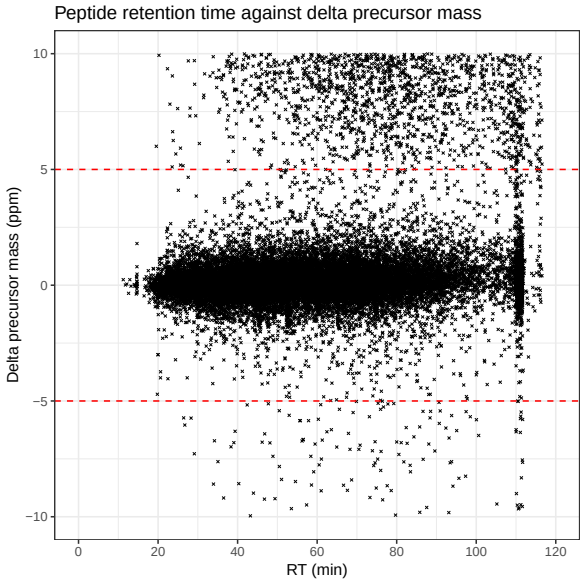
The preliminary data is as expected so we continue on to evaluate the quality of the raw data.

Experimental quality control checks

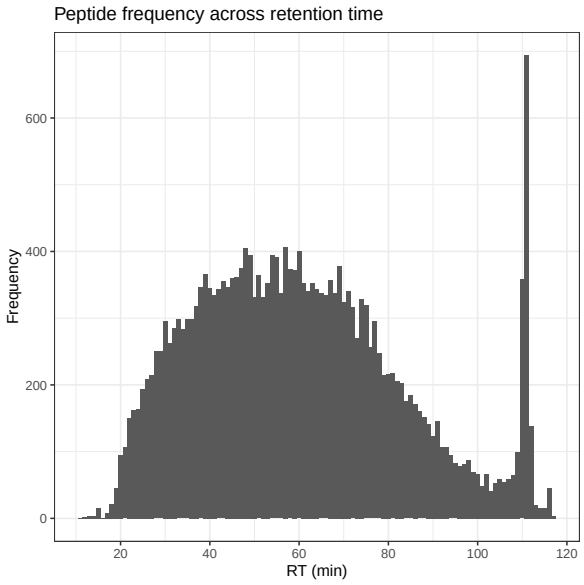
Quality control of the raw mass spectrometry data

To briefly assess the quality of the raw mass spectrometry data from which the search results were derived, we create simple plots. In contrast to the previous PSM processing workflow, we do not have access to information about ion injection times from the peptide-level file. However, we can still look at the peptide delta mass across retention time, as well as the frequency of peptides across the retention time gradient.

```
## Plot scatter plot of mass accuracy
sn_qf[["peptides_raw"]] %>%
  rowData() %>%
  as_tibble() %>%
  ggplot(aes(x = RT.in.min.by.Search.Engine.Sequest.HT,
             y = Delta.M.in.ppm.by.Search.Engine.Sequest.HT)) +
  geom_point(size = 0.5, shape = 4) +
  geom_hline(yintercept = 5, linetype = "dashed", color = "red") +
  geom_hline(yintercept = -5, linetype = "dashed", color = "red") +
  labs(x = "RT (min)", y = "Delta precursor mass (ppm)") +
  scale_x_continuous(limits = c(0, 120), breaks = seq(0, 120, 20)) +
  scale_y_continuous(limits = c(-10, 10), breaks = c(-10, -5, 0, 5, 10)) +
  ggtitle("Peptide retention time against delta precursor mass") +
  theme_bw()
```




```
## Plot histogram of peptide retention time
sn_qf[["peptides_raw"]] %>%
  rowData() %>%
  as_tibble() %>%
  ggplot(aes(x = RT.in.min.by.Search.Engine.Sequest.HT)) +
  geom_histogram(binwidth = 1) +
  labs(x = "RT (min)", y = "Frequency") +
  scale_x_continuous(breaks = seq(0, 120, 20)) +
  ggtitle("Peptide frequency across retention time") +
  theme_bw()
```



For a more in-depth discussion of these plots users should refer back to the TMT processing workflow. Since neither plot indicates any major problems with the MS runs, we continue on to basic data cleaning.

Basic data cleaning

As discussed in detail above, there are several basic data cleaning steps which are non-specific and should be applied to all quantitative datasets, regardless of the quantitation method or data level (PSM, peptide or protein). These steps are as follows:

1. Removal of features without a master protein accession
2. Removal of features corresponding to protein groups which contain a contaminant
3. Removal of features without quantitative data
4. (Optional) Removal of features which are not unique to a protein group
5. Removal of features not allocated rank 1 during the identification search
6. Removal of features not annotated as unambiguous
7. Control of FDR to 1% protein-level FDR

In addition to these standard steps, LFQ data should be filtered to remove peptides that were not quantified based on a monoisotopic peak. The monoisotopic peak is that which comprises the most abundant natural isotope of each constituent element. For bottom-up proteomics, this typically translates to the peptides containing carbon-12 and nitrogen-14. When the different isotopes are well resolved, the monoisotopic peak usually provides the most accurate measurement.

Before we remove any data, we first create a second copy of the original `SummarizedExperiment`, as to retain a copy of the raw data for reference. As before we use the `addAssay` function.

```
## Add second copy of data to be filtered
data_copy <- sn_qf[["peptides_raw"]]

sn_qf <- addAssay(x = sn_qf,
                  y = data_copy,
                  name = "peptides_filtered")

## Verify
sn_qf
```

```
## An instance of class QFeatures containing 2 assays:
##  [1] peptides_raw: SummarizedExperiment with 23375 rows and 6 columns
##  [2] peptides_filtered: SummarizedExperiment with 23375 rows and 6 columns
```

Here, cleaning is done is two steps. The first is the removal of contaminant proteins using the self-defined `find_cont` function. Refer back to the TMT processing workflow for more details.

```
## Store row indices of peptides matched to a contaminant-containing protein group
cont_peptides <- find_cont(sn_qf[["peptides_filtered"]], cont_acc)

## Remove these rows from the data
if (length(cont_peptides) > 0)
  sn_qf[["peptides_filtered"]] <- sn_qf[["peptides_filtered"]][-cont_peptides, ]
```

Second, we carry out all remaining cleaning using the `filterFeatures` function as before.

```
sn_qf <- sn_qf %>%
  filterFeatures(~ !Master.Protein.Accessions == "",
                 i = "peptides_filtered") %>%
  filterFeatures(~ !Quan.Info == "NoQuanValues",
                 i ="peptides_filtered") %>%
  filterFeatures(~ !Quan.Info == "NoneMonoisotopic",
                 i = "peptides_filtered") %>%
  filterFeatures(~ Number.of.Protein.Groups == 1,
                 i = "peptides_filtered") %>%
  filterFeatures(~ Rank.by.Search.Engine.Sequest.HT == 1,
                 i = "peptides_filtered") %>%
  filterFeatures(~ PSM.Ambiguity == "Unambiguous",
                 i = "peptides_filtered")
```

As before, we check to see whether additional annotations remain within the "Quan.Info" column.

```
## Check for remaining annotations
sn_qf[["peptides_filtered"]] %>%
  rowData %>%
  as_tibble() %>%
  pull(Quan.Info) %>%
  table()
```

```
## .
##
## 17999
```

Again, we need to import the protein-level data in order to complete FDR filtering at the highest possible level.

```
protein_sn <- read.delim(file = "supernatant_lfq_results_proteins.txt")

## Extract master protein accessions from our PSM-level data
proteins_in_data <- sn_qf[["peptides_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  select(Master.Protein.Accessions)

## Extract protein accessions and corresponding confidence from search output file
protein_search_output <- protein_sn %>%
  select(Accession, Protein.FDR.Confidence.Combined)
```

Now we can use the `left_join` function to merge these two datasets.

```
## Combine data
protein_fdr <- left_join(x = proteins_in_data,
                         y = protein_search_output,
                         by = c("Master.Protein.Accessions" = "Accession"))

protein_fdr %>%
  head()
```

```
## # A tibble: 6 x 2
##   Master.Protein.Accessions Protein.FDR.Confidence.Combined
##   <chr>                     <chr>
## 1 P55011                    High
## 2 O60341                    High
## 3 P36578                    High
## 4 A6NIH7                    High
## 5 Q9P258                    High
## 6 Q9P258                    High
```

Finally, we add the annotations from our "Protein.FDR.Confidence.Combined" column to the rowData of our "psms_filtered" experiment.

```
rowData(sn_qf[["peptides_filtered"]])$Protein.Confidence <- protein_fdr$Protein.FDR.Confidence.Combi
```

We can now use filterFeatures to remove PSMs corresponding to low or medium confidence master proteins.

```
sn_qf <- sn_qf %>%
  filterFeatures(~ Protein.Confidence == "High", i = "peptides_filtered")
```

```
## 'Protein.Confidence' found in 1 out of 2 assay(s)
## No filter applied to the following assay(s) because one or more filtering variables are missing i
## You can control whether to remove or keep the features using the 'keep' argument (see '?filterFea
```

Assessing the impact of non-specific data cleaning

As in the previous example, we assess the impact that cleaning has had on the data. Specifically, we determine the number and proportion of PSMs, peptides and proteins lost. Again, when we refer to the number of peptides we only consider unique peptide sequences, not those that differ in their modifications.

```
## Determine number of PSMs, peptides and proteins remaining
psms_remaining <- sn_qf[["peptides_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(Number.of.PSMs) %>%
  sum()

peps_remaining <- sn_qf[["peptides_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(Sequence) %>%
  unique() %>%
  length() %>%
  as.numeric()

prots_remaining <- sn_qf[["peptides_filtered"]] %>%
  rowData() %>%
  as_tibble() %>%
  pull(Master.Protein.Accessions) %>%
  unique() %>%
  length() %>%
  as.numeric()

## Determine the number of proportion of PSMs, peptides and proteins removed
psms_removed <- original_psms - psms_remaining
psms_removed_prop <- ((psms_removed /original_psms) * 100) %>%
  round(digits = 2)

peps_removed <- original_peps - peps_remaining
peps_removed_prop <- ((peps_removed / original_peps) * 100) %>%
  round(digits = 2)
```

```
prots_removed <- original_prots - prots_remaining
prots_removed_prop <- ((prots_removed / original_prots) * 100) %>%
  round(digits = 2)

## Present in a table
data.frame("Feature" = c("PSMs",
                         "Peptides",
                         "Proteins"),
           "Number lost" = c(psms_removed,
                             peps_removed,
                             prots_removed),
           "Percentage lost" = c(psms_removed_prop,
                                 peps_removed_prop,
                                 prots_removed_prop),
           "Number remaining" = c(psms_remaining,
                                  peps_remaining,
                                  prots_remaining))
```

```
##    Feature Number.lost Percentage.lost Number.remaining
## 1     PSMs       28367           19.66           115935
## 2 Peptides        3876           19.08            16436
## 3 Proteins         799           20.27             3142
```

Peptide quality control filtering

When extracting data from the peptide-level `.txt` file rather than aggregating up from a PSM file, additional parameters exist within the peptide `rowData`. Such parameters include Quality PEP, Quality q-value, and SVM score, as well as similar scoring parameters provided by the search engine. Although we will not complete additional filtering based on these parameters in this workflow, users may wish to explore this option.

Managing missing data

Having cleaned the peptide-level data we now move onto the management of missing data. This is of particular importance for LFQ workflows where the missing value challenge is amplified by intrinsic variability between independent MS runs. As before, the management of missing data can be divided into three steps: 1) exploring the presence and distribution of missing values, (2) filtering out missing values, and (3) optional imputation.

Exploring the presence of missing values

The aim of the first step is to determine how many missing values are present within the data, and how they are distributed between samples and/or conditions.

```
## Are there any NA values within the peptide data?
sn_qf[["peptides_filtered"]] %>%
  assay() %>%
  anyNA()
```

```
## [1] TRUE
```

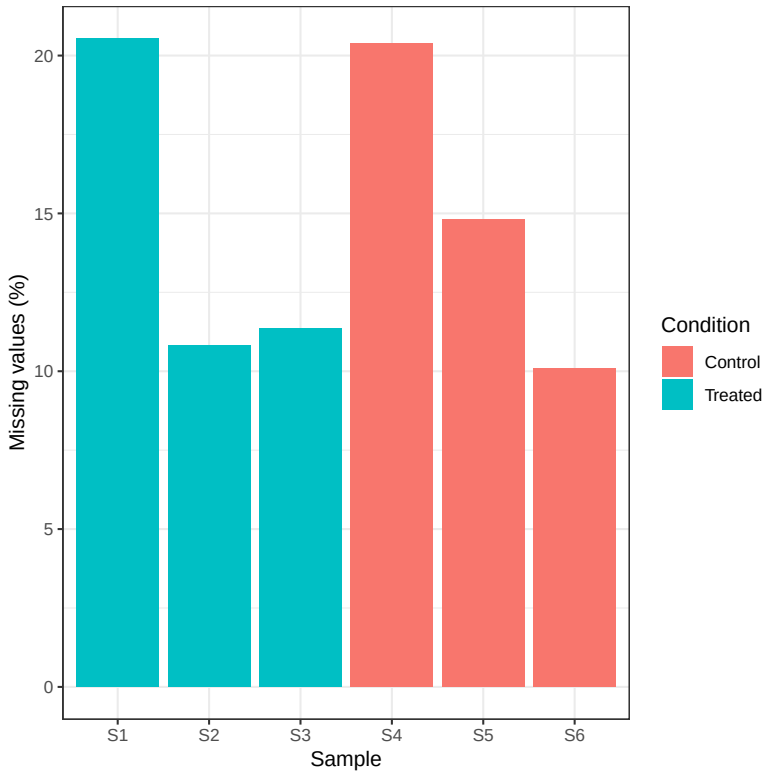
```
## How many NA values are there within the peptide data?
sn_qf[["peptides_filtered"]] %>%
  nNA()
```

```
## $nNA
## DataFrame with 1 row and 2 columns
##         nNA       pNA
##   <integer> <numeric>
## 1     15742  0.146655
##
## $nNArows
## DataFrame with 17890 rows and 3 columns
##              name       nNA       pNA
```

```
##       <character> <integer> <numeric>
## 1               1         4  0.666667
## 2               2         1  0.166667
## 3               3         0  0.000000
## 4               4         1  0.166667
## 5               5         0  0.000000
## ...           ...       ...       ...
## 17886       23371         0         0
## 17887       23372         0         0
## 17888       23373         0         0
## 17889       23374         0         0
## 17890       23375         0         0
##
## $nNAcols
## DataFrame with 6 rows and 3 columns
##          name       nNA       pNA
##   <character> <integer> <numeric>
## 1          S1      3674  0.205366
## 2          S2      1936  0.108217
## 3          S3      2030  0.113471
## 4          S4      3648  0.203913
## 5          S5      2650  0.148127
## 6          S6      1804  0.100838
```

As expected, the LFQ data contains a higher proportion of missing values as compared to the TMT-labelled data. There are 15742 missing (NA) values within the data, which corresponds to 0.1 of the data. We check for sample- and condition- specific biases in the distribution of these NA values, again plotting percentage (proportion multiplied by 100).

```
## Plot histogram to visualize sample-specific distribution of NAs
nNA(sn_qf[["peptides_filtered"]])$nNAcols %>%
  as_tibble() %>%
  mutate(Condition = rep(c("Treated", "Control"), each = 3)) %>%
  ggplot(aes(x = name, y = (pNA * 100), group = Condition, fill = Condition)) +
  geom_bar(stat = "identity") +
  labs(x = "Sample", y = "Missing values (%)") +
  theme_bw()
```



Whilst S1 and S4 have a slightly higher proportion of missing values, all of the samples are within an acceptable range to continue. Again, there is no evidence of a condition-specific bias in the data.

Filtering out missing values

We next filter out features, here peptides, which comprise 20% or more missing values.

```
## Check how many peptides we will remove
which(nNA(sn_qf[["peptides_filtered"]])$nNArows$pNA >= 0.2) %>%
  length()
```

```
## [1] 4332
```

```
## Remove peptides with 2 or more NA values
sn_qf <- sn_qf %>%
  filterNA(pNA = 0.2,
           i = "peptides_filtered")
```

Imputation (optional)

Finally, we check how many missing values remain in the data before making a decision as to whether imputation is required.

```
nNA(sn_qf[["peptides_filtered"]])$nNA
```

```
## DataFrame with 1 row and 2 columns
##         nNA       pNA
##   <integer> <numeric>
## 1      2430 0.0298717
```

There are 2430 missing values remaining. The presence of proteins with single or low peptide support means that some of these NA values will likely be propagated upward during aggregation. Whilst NA values were traditionally problematic during the application of downstream statistical methods, there are now a number of algorithms that allow statistics to be completed on data containing missing values. For example, the (MSqRob2)[https://www.bioconductor.org/packages/release/bioc/html/msqrob2.html] [24, Goeminne et al. [32]; 25] package facilitates statistical differential expression analysis on datasets without the need for imputation and functions within the `QFeatures` infrastructure. Nevertheless, for the purpose of demonstration, we here choose to impute the raw intensity data.

As eluded to above, the most appropriate method to determine such probable values is dependent upon why the value is missing, that is whether it is MCAR or MNAR. Although the optimal imputation method is specific to each dataset, left-censored methods (e.g. minimal value approaches, limit of detection) have proven favorable for data with a high proportion of MNAR values whilst hot deck methods (e.g. k-nearest neighbours, random forest, maximum likelihood methods) are more appropriate when the majority of missing data is MCAR [e.g., 33, 23]. Within the `QFeatures` infrastructure imputation is carried out by passing the data to the `impute` function, please see `?impute` for more information. To see which imputation methods are supported by this function we use the following code:

```
## Find out available imputation methods
MsCoreUtils::imputeMethods()
```

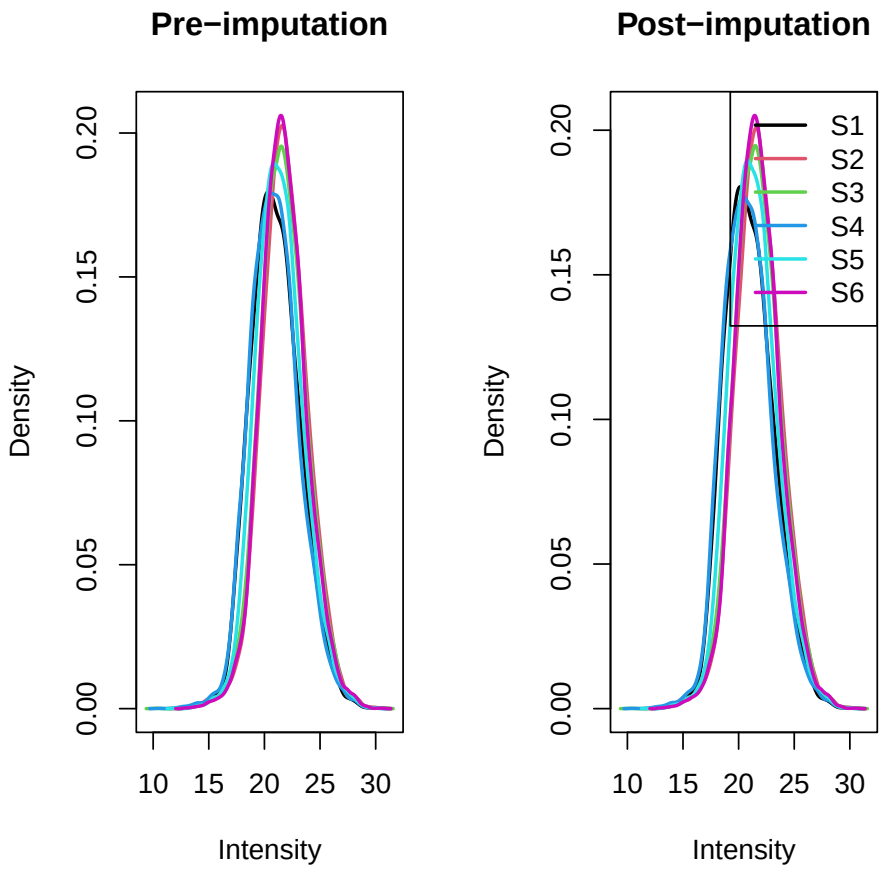
```
##  [1] "bpca"    "knn"     "QRILC"   "MLE"     "MLE2"    "MinDet"  "MinProb"
##  [8] "min"     "zero"    "mixed"   "nbavg"   "with"    "RF"      "none"
```

Unfortunately, it is very challenging to determine the reason(s) behind missing data, and in most cases experiments contain a mixture of MCAR and MNAR. For LFQ data where little is know about the cause of missing values it is advisable to use methods optimized for MCAR. Here we will use the baseline k-nearest neighbours (k-NN) imputation on the raw peptide intensities. Of note, users who wish to utilize an alternative imputation method should check whether their selected method has a requirement for normality. If the method requires data to display a normal distribution, users must log2 transform the data prior to imputation.

```
## Impute missing values using kNN
sn_qf <- impute(sn_qf,
                method = "knn",
                i = "peptides_filtered",
                name = "peptides_imputed")
```

Following imputation we check to ensure that the distribution of the data has not dramatically changed. To do so we create a density plot of the data pre- and post-imputation.

```
## visualize the impact of imputation
par(mfrow = c(1, 2))

sn_qf[["peptides_filtered"]] %>%
  assay() %>%
  log2() %>%
  plotDensities(main = "Pre-imputation",
                legend = FALSE)

sn_qf[["peptides_imputed"]] %>%
  assay() %>%
  log2() %>%
  plotDensities(main = "Post-imputation",
                legend = "topright")
```



From this plot the change in the data appears to be minimal. We can further validate this by comparing the summary statistics of the data pre- and post- imputation.

```
## Determine the impact of imputation on summary statistics
pre_imputation_summary <- sn_qf[["peptides_filtered"]] %>%
  assay() %>%
  longFormat() %>%
  group_by(colname) %>%
  summarise(sum_intensity = sum(value, na.rm = TRUE),
```

```
            max_intensity = max(value, na.rm = TRUE),
            median_intensity = median(value, na.rm = TRUE))

post_imputation_summary <- sn_qf[["peptides_imputed"]] %>%
  assay() %>%
  longFormat() %>%
  group_by(colname) %>%
  summarise(sum_intensity = sum(value, na.rm = TRUE),
            max_intensity = max(value, na.rm = TRUE),
            median_intensity = median(value, na.rm = TRUE))

print(pre_imputation_summary)
```

```
## # A tibble: 6 x 4
##   colname sum_intensity max_intensity median_intensity
##   <fct>           <dbl>         <dbl>            <dbl>
## 1 S1       98618011197.   1477162278.          1950392.
## 2 S2      155288272302.   1678988256.          3567233.
## 3 S3      145083815203.   1804842981.          3255696.
## 4 S4       94532147208.   1087946291.          1880489.
## 5 S5      121223290798.   1307181986           2511748.
## 6 S6      143149276808.   1608003894.          3288965.
```

```
print(post_imputation_summary)
```

```
## # A tibble: 6 x 4
##   colname sum_intensity max_intensity median_intensity
##   <fct>           <dbl>         <dbl>            <dbl>
## 1 S1       99501169413.   1477162278.          1814576.
## 2 S2      155691649219.   1678988256.          3493145.
## 3 S3      145325835653.   1804842981.          3210018.
## 4 S4       95867237989.   1087946291.          1726081.
## 5 S5      121738595609.   1307181986           2446493.
## 6 S6      143619358837.   1608003894.          3216396.
```

Comparison of the two tables reveals minimal change within the data. However, we find that S1 and S4 display greater differences between pre- and post-imputation statistics because of the higher number of missing values which required imputation.

Logarithmic transformation of quantitative data

In the following code chunk we log2 transform the peptide-level data to generate a near-normal distribution within the quantitative data. This is necessary prior to the use of `robustSummary` aggregation.

```
## log2 transform the quantitative data
sn_qf <- logTransform(object = sn_qf,
                      base = 2,
                      i = "peptides_imputed",
                      name = "log_peptides")

## Verify
sn_qf
```

```
## An instance of class QFeatures containing 4 assays:
##  [1] peptides_raw: SummarizedExperiment with 23375 rows and 6 columns
##  [2] peptides_filtered: SummarizedExperiment with 13558 rows and 6 columns
##  [3] peptides_imputed: SummarizedExperiment with 13558 rows and 6 columns
##  [4] log_peptides: SummarizedExperiment with 13558 rows and 6 columns
```

Aggregation of peptide to protein

Now that we are happy with the peptide-level data, we aggregate upward to proteins using the `aggregateFeatures` function.

```
## Aggregate peptide to protein
sn_qf <- aggregateFeatures(sn_qf,
                           i = "log_peptides",
                           fcol = "Master.Protein.Accessions",
                           name = "log_proteins",
                           fun = MsCoreUtils::robustSummary,
                           na.rm = TRUE)
```

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

```
## Verify
sn_qf
```

```
## An instance of class QFeatures containing 5 assays:
##  [1] peptides_raw: SummarizedExperiment with 23375 rows and 6 columns
##  [2] peptides_filtered: SummarizedExperiment with 13558 rows and 6 columns
##  [3] peptides_imputed: SummarizedExperiment with 13558 rows and 6 columns
##  [4] log_peptides: SummarizedExperiment with 13558 rows and 6 columns
##  [5] log_proteins: SummarizedExperiment with 2760 rows and 6 columns
```

Normalization of quantitative data

Finally, we complete the data processing by normalizing quantitation between samples. This is done using the `"center.median"` method via the `normalize` function.

```
## normalize protein-level quantitation data
sn_qf <- normalize(sn_qf,
                   i = "log_proteins",
                   name = "log_norm_proteins",
                   method = "center.median")

## Verify
sn_qf
```

```
## An instance of class QFeatures containing 6 assays:
##  [1] peptides_raw: SummarizedExperiment with 23375 rows and 6 columns
##  [2] peptides_filtered: SummarizedExperiment with 13558 rows and 6 columns
##  [3] peptides_imputed: SummarizedExperiment with 13558 rows and 6 columns
##  [4] log_peptides: SummarizedExperiment with 13558 rows and 6 columns
##  [5] log_proteins: SummarizedExperiment with 2760 rows and 6 columns
##  [6] log_norm_proteins: SummarizedExperiment with 2760 rows and 6 columns
```

The final dataset is comprised of 2760 proteins. We will save the protein-level `SummarizedExperiment` file as well as exporting the final `QFeatures` object.

```
## Save protein-level SE
sn_proteins <- sn_qf[["log_norm_proteins"]]

## Export TMT final QFeatures object
save(sn_qf, file = "sn_qf.rda")
```

Exploration of protein data

Having described the processing steps for quantitative proteomics data, we next demonstrate how to explore the protein-level data prior to statistical analysis. For this we will utilize the TMT-labelled cell pellet dataset since it contains a greater number of proteins.

Correlation plots

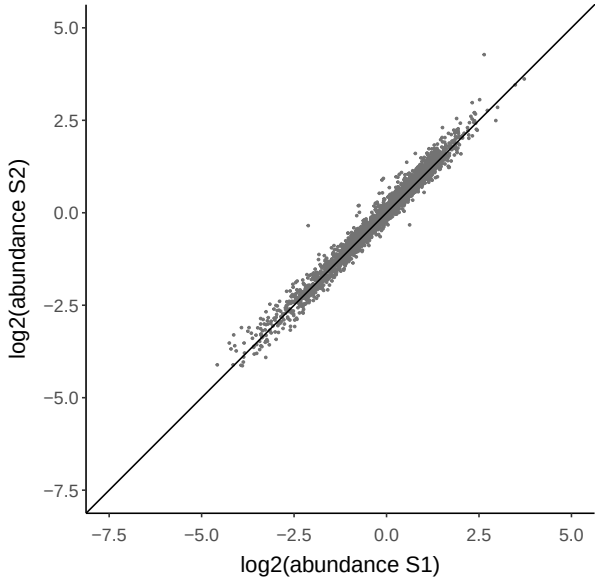
We will first generate correlation plots between pairs of samples. To do this we use the `corrplot` package to calculate and plot the Pearson's correlation coefficient between each sample pair. The `cor` function within the `corrplot` package will create a correlation matrix but requires a `data.frame`, `matrix` or a `vector` of class `numeric` as input. To convert the `QFeatures` `assay` data into a `data.frame` we use the `as.data.frame` function.

```
## Convert TMT CP protein assay into a dataframe
prot_df <- cp_qf[["log_norm_proteins"]] %>%
  assay() %>%
  as.data.frame()

## Calculate a correlation matrix between samples
corr_matrix <- cor(prot_df,
                   method = "pearson",
                   use = "pairwise.complete.obs")

print(corr_matrix)
```

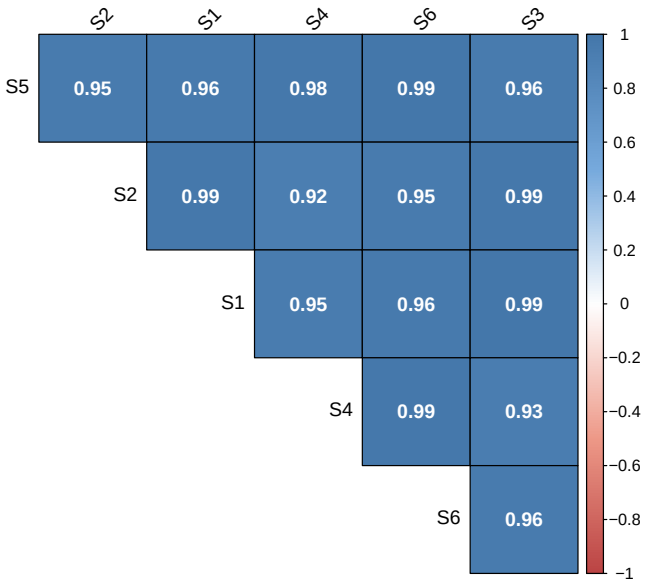
```
##           S5        S2        S1        S4        S6        S3
## S5 1.0000000 0.9540226 0.9634202 0.9821489 0.9927946 0.9558386
## S2 0.9540226 1.0000000 0.9862036 0.9207314 0.9508941 0.9886243
## S1 0.9634202 0.9862036 1.0000000 0.9451292 0.9637123 0.9926604
## S4 0.9821489 0.9207314 0.9451292 1.0000000 0.9864988 0.9348920
## S6 0.9927946 0.9508941 0.9637123 0.9864988 1.0000000 0.9557268
## S3 0.9558386 0.9886243 0.9926604 0.9348920 0.9557268 1.0000000
```

Now we can visualize the correlation data using pairwise scatter plots and a correlation heat map.

```
## Plot correlation between two samples - S1 and S2 used as example
prot_df %>%
  ggplot(aes(x = `S1`, y = `S2`)) +
  geom_point(colour = "grey45", size = 0.5) +
  geom_abline(intercept = 0, slope = 1) +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        plot.background = element_rect(fill = "white"),
        panel.background = element_rect(fill = "white"),
        axis.title.x = element_text(size = 15, vjust = -2),
        axis.title.y = element_text(size = 15, vjust = 3),
        axis.text.x = element_text(size = 12, vjust = -1),
        axis.text.y = element_text(size = 12),
        axis.line = element_line(linewidth = 0.5, colour = "black"),
        plot.margin = margin(10, 10, 10, 10)) +
  xlim(-7.5, 5) +
  ylim(-7.5, 5) +
  labs(x = "log2(abundance S1)", y = "log2(abundance S2)") +
  coord_fixed(ratio = 1)
```




```
## Create colour palette for continuum
col <- colorRampPalette(c("#BB4444", "#EE9988", "#FFFFFF",
                          "#77AADD", "#4477AA"))

## Plot all pairwise correlations
prot_df %>%
  cor(method = "pearson",
      use = "pairwise.complete.obs") %>%
  corrplot(method = "color",
           col = col(200),
           type = "upper",
           addCoef.col = "white",
           diag = FALSE,
           tl.col = "black",
           tl.srt = 45,
           outline = TRUE)
```



From these plots we can see that all replicate pairs have a Pearson's correlation coefficient >0.98 whilst the correlation between pairs of control and treated samples is somewhat lower. Users may interpret this information as an early indication that some proteins may be differentially abundant between the two groups.

Of note, whilst correlation is widely applied as a measure of reproducibility, users are reminded that correlation coefficients alone are not informative of reproducibility [34, 35]. This is especially true for expression proteomics data in which high correlation values are likely due to the majority of proteins remaining at similar

levels regardless of cellular perturbation. Users are directed to [36] for additional information regarding how to determine the calculation of experimental reproducibility.

Principal Component Analysis

Principal Component Analysis (PCA) is a dimensionality reduction method which aims to simplify complex datasets and facilitate the visualization of multi-dimensional data. Here we use the `prcomp` function from the `stats` package to perform the PCA. Since PCA does not accept missing values and we did not impute the TMT data, the `filterNA` function can be used to remove any missing values that may be present in the protein-level data. We then extract and transpose the assay data before passing it to the `prcomp` function to carry out PCA.

```
## Carry out principal component analysis
prot_pca <- cp_qf[["log_norm_proteins"]] %>%
  filterNA() %>%
  assay() %>%
  t() %>%
  prcomp(scale = TRUE, center = TRUE)
```

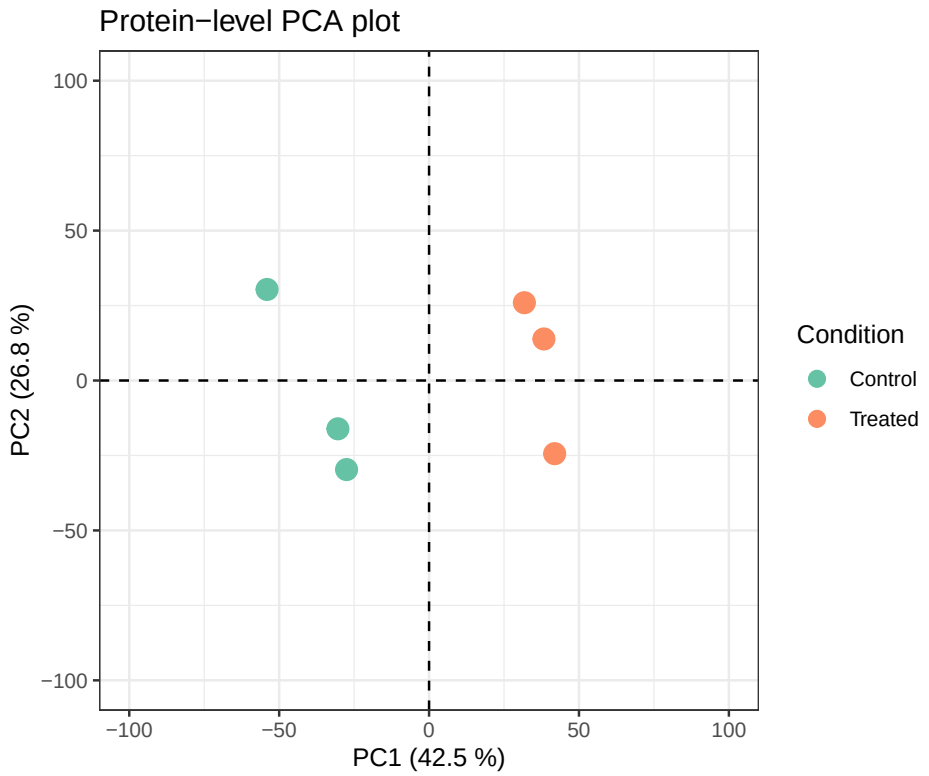
We can get an idea of the outcome of the PCA by running the `summary` function on the results of the PCA.

```
## Get a summary of the PCA
summary(prot_pca)
```

```
## Importance of components:
##                            PC1     PC2     PC3      PC4     PC5       PC6
## Standard deviation     42.035 26.5522 18.1232 14.97317 14.3323 4.334e-14
## Proportion of Variance  0.547  0.2183  0.1017  0.06941  0.0636 0.000e+00
## Cumulative Proportion   0.547  0.7653  0.8670  0.93640  1.0000 1.000e+00
```

Finally, we create a PCA plot. For additional PCA exploration and visualization tools users are directed to the `factoextra` package.

```
## Generate dataframe of each sample's PCA results
pca_df <- as.data.frame(prot_pca$x)

## Annotate samples with their corresponding condition
pca_df$condition <- cp_qf[["psms_raw"]]$condition

## Generate a PCA plot using PC1 and PC2
pca_df %>%
  ggplot(aes(x = PC1, y = PC2, colour = condition)) +
  geom_point(size = 4) +
  scale_color_brewer(palette = "Set2") +
  labs(colour = "Condition") +
  geom_hline(yintercept = 0, linetype = "dashed") +
  geom_vline(xintercept = 0, linetype = "dashed") +
  guides(colour = guide_legend(override.aes = list(size = 3))) +
  labs(x = "PC1 (42.5 %)", y = "PC2 (26.8 %)") +
  ggtitle("Protein-level PCA plot") +
  xlim(-100, 100) +
  ylim(-100, 100) +
  coord_fixed(ratio = 1) +
  theme_bw()
```



Exploring potential batch effects

Before carrying out differential expression analysis, it is first necessary to explore the presence of batch effects within the data. Batch effects are derived from non-biological factors which impact the experimental data. These include reagents, instrumentation, personnel and laboratory conditions. In most cases the increased variation caused by batch effects will lead to reduced downstream statistical power. On the other hand, if correlated with the experimental sub- groups, batch effects can also lead to confounded results and the incorrect biological interpretation of differential expression [37].

Given that the use-case data was derived from a small experiment with only six samples and a single TMTplex, there are minimal batch effects to explore here. For users analyzing larger experiments completed over long period of time, across several laboratories/individuals, or using multiple TMTplex reagents, it is advisable to annotate the PCA plot with all potential batch factors. If data is found to cluster based on any of these factors, batch effects should be incorporated into downstream analyses. For example, users can apply the `removeBatchEffect` function from the `limma` package.

Discovery and biological interpretation of differentially abundant proteins

The last section of this workflow demonstrates how to gain biological insights from the resulting list of proteins. Again we will utilize the TMT-labelled cell pellet data, although the process would be exactly the same for the LFQ supernatant protein list. Users are reminded that although referred to as differential 'expression' analysis, abundance is determined by both protein synthesis and degradation.

Given that there are a number of Bioconductor packages available for each of the remaining steps (statistical analysis and interpretation), users are prompted to install packages at the where required, at the beginning of each step.

Extracting and organising protein-level data

We first extract the protein-level `SummarizedExperiment` from the cell pellet TMT `QFeatures` object and specify the study factors. Here we are interested in discovering differences between conditions, control and treated. As well as assigning these conditions to each sample, we can define the control group as the reference level such that differential abundance is reported relative to the control. This means that when we get the results of the statistical analysis, 'upregulated' will refer to increased abundance in treated cells relative to control controls.

```
## Extract protein-level data and associated colData
cp_proteins <- cp_qf[["log_norm_proteins"]]
colData(cp_proteins) <- colData(cp_qf[["log_norm_proteins"]])

## Create factor of interest
cp_proteins$condition <- factor(cp_proteins$condition)

## Check which level of the factor is the reference level and correct
cp_proteins$condition
```

```
## [1] Control Treated Treated Control Control Treated
## Levels: Control Treated
```

```
cp_proteins$condition <- relevel(cp_proteins$condition, ref = "Control")
```

Differential expression analysis using `limma`

Bioconductor contains several packages dedicated to the statistical analysis of proteomics data. For example, `MSstats` and `MSstatsTMT` can be used to determine differential protein expression within both DDA and DIA datasets for LFQ and TMT, respectively [38, 39]. Of note, `MSstatsTMT` includes additional functionality for dealing with larger, multi-plexed TMT experiments. For LFQ experiments, `proDA`, `prolfqua` and `MSqRob2` can be utilized, among others [32, 40]. Here, we will use the `limma` package [41]. `limma` is widely used for the analysis of large omics datasets and has several models that allow differential abundance to be assessed in multifactorial experiments. This is useful because it allows multiple factors, including TMTplex, to be integrated into the model itself, thus minimising the effects of confounding factors. In this example we will apply `limma`'s empirical Bayes moderated t-test, a method that is appropriate for small sample sizes [31].

We first use the `model.matrix` function to create a matrix in which each of the samples are annotated based on the factors we wish to model, here the condition group. This ultimately defines the 'design' of the model, that is how the samples are distributed between the groups of interest. We then fit a linear model to the abundance data of each protein by passing the data and model design matrix to the `lmFit` function. Finally, we update the estimated standard error for each model coefficient using the `eBayes` function. This function borrows information across features, here proteins, to shift the per-protein variance estimates towards an expected value based on the variance estimates of other proteins with similar mean intensity. This empirical Bayes technique has been shown to reduce the number of false positives for proteins with small variances as well as increase the power of detection for differentially abundant proteins with larger variances [42]. Further, we use the `trend = TRUE` argument when passing the `eBayes` function so that an intensity-dependent trend can be fitted to the prior variances. For more information about the `limma` trend method users are directed to [43].

```
## Design a matrix containing all of the factors we wish to model the effects of
model_design <- model.matrix(~ cp_proteins$condition)

## Verify
print(model_design)
```

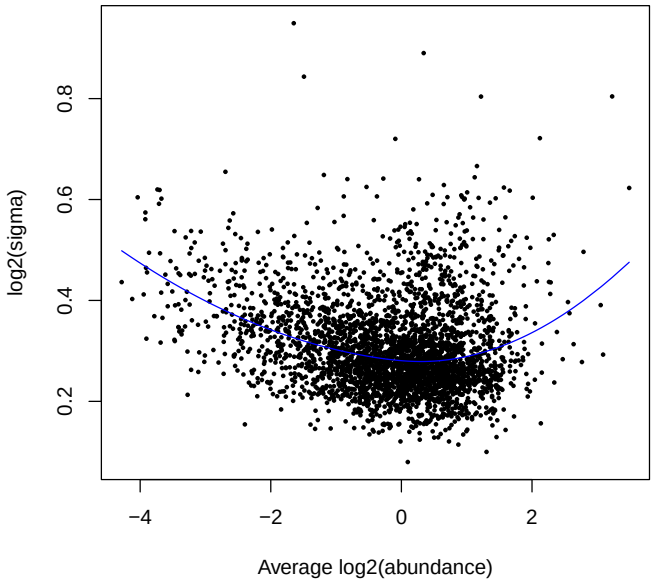
```
##   (Intercept) cp_proteins$conditionTreated
## 1           1                            0
## 2           1                            1
## 3           1                            1
## 4           1                            0
## 5           1                            0
## 6           1                            1
## attr(,"assign")
## [1] 0 1
## attr(,"contrasts")
## attr(,"contrasts")$`cp_proteins$condition`
## [1] "contr.treatment"
```

```
## Create a linear model using this design
fitted_lm <- cp_proteins %>%
  assay() %>%
  lmFit(design = model_design)
```

```
## Update the model based on Limma eBayes algorithm
fitted_lm <- eBayes(fit = fitted_lm,
                    trend = TRUE)

## Save results of the test
limma_results <- topTable(fit = fitted_lm,
                          coef = "cp_proteins$conditionTreated",
                          adjust.method = "BH",
                          number = Inf) %>%
  rownames_to_column("Protein") %>%
  as_tibble() %>%
  mutate(TP = grepl("ups", Protein))
```

Having applied the model to the data, we need to verify that this model was appropriate and that the statistical assumptions were met. To do this we first generate an SA plot using the `plotSA` function within `limma`. An SA plot shows the log2 residual standard deviation (sigma) against log average abundance and is a simple way to visualize the trend that has been fitted to the data.

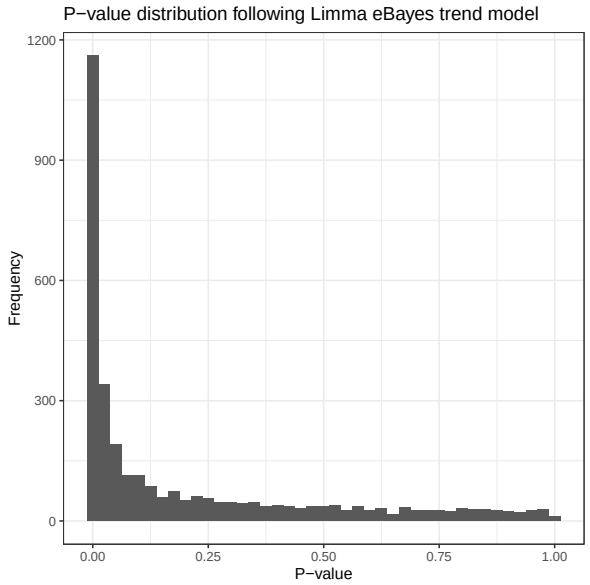
```
## Plot residual SD against average log abundance
plotSA(fitted_lm,
       xlab = "Average log2(abundance)",
       ylab = "log2(sigma)",
       cex = 0.5)
```



The residual standard deviation is a measure of model accuracy and is most easily conceptualized as a measurement of how far from the model prediction each data point lies. The smaller the residual standard deviation, the closer the fit between the model and observed data.

Next we will plot a p-value histogram. Importantly, this histogram shows the distribution of p-values prior to any multiple hypothesis test correction or false discovery rate control. This means plotting the `P.value` variable, not the `adj.P.Val`.

```
## Plot histogram of raw p-values
limma_results %>%
  ggplot(aes(x = P.Value)) +
  geom_histogram(binwidth = 0.025) +
  labs(x = "P-value", y = "Frequency") +
  ggtitle("P-value distribution following Limma eBayes trend model") +
  theme_bw()
```



The figure displayed shows an anti-conservative p-value distribution. The flat distribution across the base of the graph represents the non-significant p-values spread uniformly between 0 and 1, whilst the peak close to 0 contains significant p-values, along with some false positives. For a more thorough explanation of interpreting p-value distributions, including why your data may not produce an anti-conservative distribution if your statistical model is inappropriate, please see [44]. Now, having applied the statistical model and verified it's suitability, we take an initial look at the outputs.

```
## Look at limma results table
head(limma_results)
```

```
## # A tibble: 6 x 8
##   Protein logFC AveExpr     t  P.Value  adj.P.Val     B TP
##   <chr>   <dbl>   <dbl> <dbl>    <dbl>      <dbl> <dbl> <lgl>
## 1 Q01581   1.50   0.592  28.4 6.35e-10 0.00000205  13.1 FALSE
## 2 P15104   1.32   1.36   23.2 3.63e- 9 0.00000558  11.7 FALSE
## 3 Q9UK41   1.46  -1.49   21.7 6.36e- 9 0.00000558  11.2 FALSE
## 4 P37268   1.32  -0.674  21.0 8.58e- 9 0.00000558  10.9 FALSE
## 5 P04183   1.29   0.943  21.0 8.64e- 9 0.00000558  10.9 FALSE
## 6 Q9UHI8   2.45   0.368  18.8 2.21e- 8 0.0000119   10.0 FALSE
```

The results table contains several important pieces of information. Each master protein is represented by its accession number and has an associated log2 fold change, that is the log2 difference in mean abundance between conditions, as well as a log2 mean expression across all six samples, termed `AveExpr`. Since we carried out an empirical Bayes moderated t-test, each protein also has a moderated t-statistic and associated p-value. The moderated t-statistic can be interpreted in the same way as a standard t-statistic. Each protein also has an adjusted p-value which accounts for multiple hypothesis testing to control the overall FDR. The default method for multiple hypothesis corrections within the `topTable` function that we applied is the Benjamini and Hochberg (BH) adjustment [45], although we could have specified an alternative. Finally, the B-statistic represents the log-odds that a protein is differentially abundant between the two conditions, and the data is presented in descending order with those with the highest log-odds of differential abundance at the top.

We can add annotations to this results table based on the user-defined significance thresholds. In the literature, for stringent analyses a FDR-adjusted p-value threshold of 0.01 is most frequently used, or 0.05 for exploratory analyses. Ultimately these thresholds are arbitrary and set by the user. The addition of a log-fold change (logFC) threshold is at the users discretion and can be useful to determine significant results of biological relevance. When using a TMT labelling strategy the co-isolation interference can lead to substantial and uneven ratio compression, thus it is not recommended to apply a fold change threshold here.

```
## Add direction of log fold change relative to control
limma_results$direction <- ifelse(limma_results$logFC > 0,
                                  "up", "down") %>%
  as.factor()

## Add significance thresholds
```

```
limma_results$significance <- ifelse(limma_results$adj.P.Val < 0.01,
                                     "sig", "not.sig") %>%
  as.factor()


## Verify
str(limma_results)
```

```
## tibble [3,230 x 10] (S3: tbl_df/tbl/data.frame)
##  $ Protein     : chr [1:3230] "Q01581" "P15104" "Q9UK41" "P37268" ...
##  $ logFC       : num [1:3230] 1.5 1.32 1.46 1.32 1.29 ...
##  $ AveExpr     : num [1:3230] 0.592 1.363 -1.485 -0.674 0.943 ...
##  $ t           : num [1:3230] 28.4 23.2 21.7 21 21 ...
##  $ P.Value     : num [1:3230] 6.35e-10 3.63e-09 6.36e-09 8.58e-09 8.64e-09 ...
##  $ adj.P.Val   : num [1:3230] 2.05e-06 5.58e-06 5.58e-06 5.58e-06 5.58e-06 ...
##  $ B           : num [1:3230] 13.1 11.7 11.2 10.9 10.9 ...
##  $ TP          : logi [1:3230] FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ direction   : Factor w/ 2 levels "down","up": 2 2 2 2 2 2 1 2 1 2 ...
##  $ significance: Factor w/ 2 levels "not.sig","sig": 2 2 2 2 2 2 2 2 2 2 ...
```

In the next code chunk, we use the `decideTests` function to determine how many proteins are significantly up- and down- regulated in the treated compared to control HEK293 cells. We tell this function to classify the significance of each t-statistic based on a BH-adjusted p-value of 0.01. If we had not used TMT labels and wished to include a logFC threshold, we could have included `lfc =` as an argument. The function will then output a numerical matrix containing either -1, 0, or 1 for each protein in each condition, where a value of -1 indicates significant downregulation, 0 not significant and 1 significant upregulation. To simplify interpretation, we print a `summary` of this matrix.

```
## Get a summary of statistically significant results
fitted_lm %>%
  decideTests(adjust.method = "BH",
              p.value = 0.01) %>%
  summary()
```

```
##        (Intercept) cp_proteins$conditionTreated
## Down          1423                          381
## NotSig         404                         2536
## Up            1403                          313
```

From this table we can see that 381 proteins were downregulated in treated HEK293 cells compared to the control group whilst 313 were upregulated. Given that no logFC threshold was applied some of the significant differences in abundance may be small. Further, these results mean little without any information about which proteins these were and what roles they play within the cell. We subset the significant proteins so that we can investigate them further.

```
## Subset proteins that show significantly different abundance
sig_proteins <- subset(limma_results,
                       adj.P.Val <= 0.01)

length(sig_proteins$Protein)
```

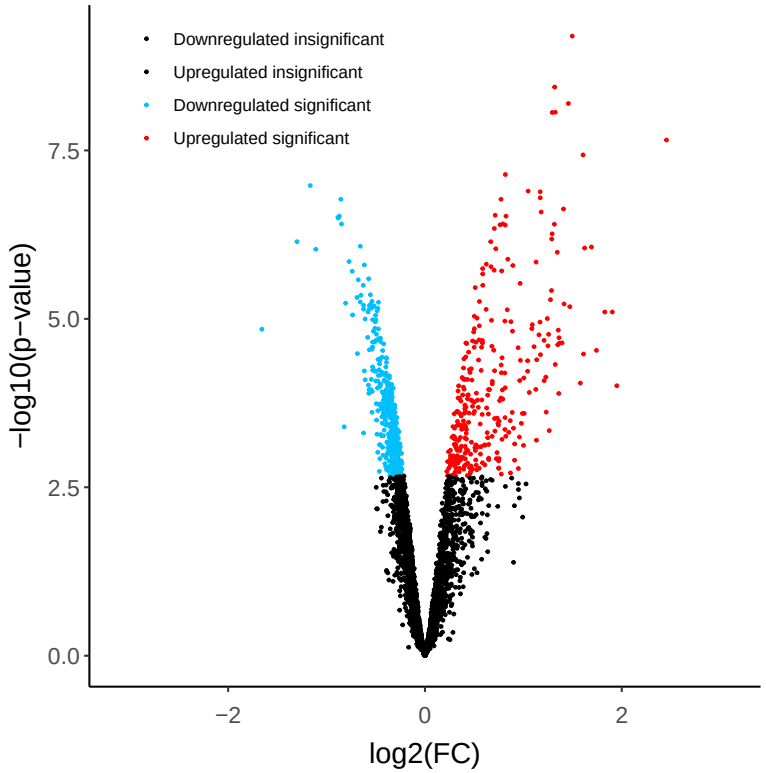
```
## [1] 694
```

Visualizing differentially abundant proteins

Before looking deeper into which proteins have differential abundance, we first create some simple plots to visualize the results. Volcano plots and MA plots are two of the common visualisations used in this instance. When plotting the former, users are advised to plot raw p-values rather than their derivative BH-adjusted p-values. Point colours can be used to indicate significance based on BH- adjusted p-values, as is shown in the code chunk below.

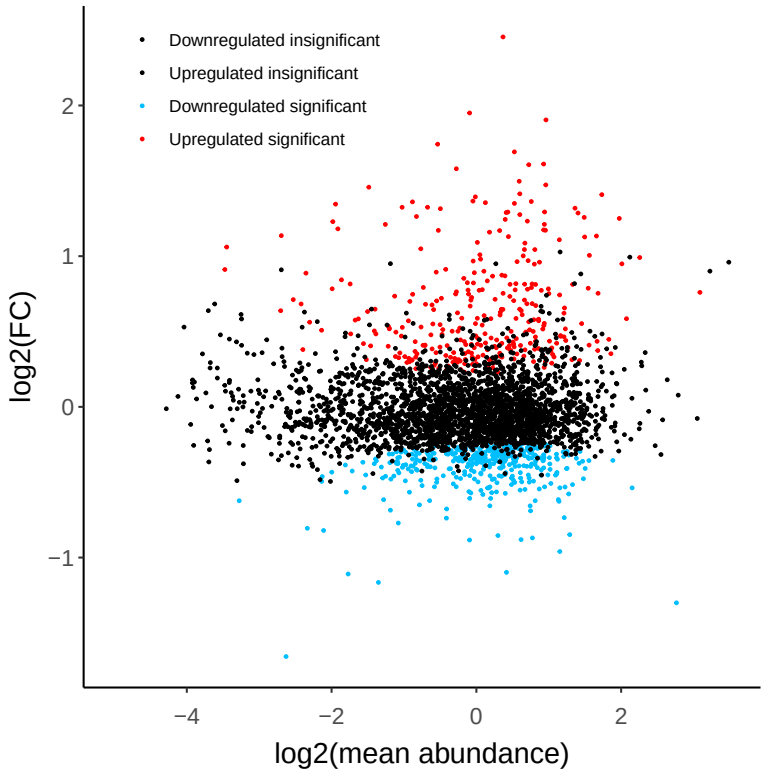
```
## Generate a volcano plot
limma_results %>%
  ggplot(aes(x = logFC, y = -log10(P.Value))) +
  geom_point(aes(colour = significance:direction), size = 0.5) +
  scale_color_manual(
    values = c("black", "black", "deepskyblue", "red"), name = "",
    labels = c("Downregulated insignificant",
               "Upregulated insignificant",
               "Downregulated significant",
               "Upregulated significant")) +
  theme(axis.title.x = element_text(size = 15, vjust = -2),
        axis.title.y = element_text(size = 15, vjust = 2),
        axis.text.x = element_text(size = 12, vjust = -1),
        axis.text.y = element_text(size = 12),
        plot.background = element_rect(fill = "white"),
        panel.background = element_rect(fill = "white"),
        axis.line = element_line(linewidth = 0.5, colour = "black"),
        plot.margin = margin(10, 10, 10, 10),
        legend.position = c(0.25, 0.9)) +
  labs(x = "log2(FC)", y = "-log10(p-value)") +
  xlim(-3.1, 3.1)
```

```
## Warning: A numeric `legend.position` argument in `theme()` was deprecated in ggplot2
## 3.5.0.
## i Please use the `legend.position.inside` argument of `theme()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```




```
## Generate MA plot
limma_results %>%
  ggplot(aes(x = AveExpr, y = logFC)) +
  geom_point(aes(colour = significance:direction), size = 0.5) +
  scale_color_manual(
    values = c("black", "black", "deepskyblue", "red"), name = "",
    labels = c("Downregulated insignificant",
```

```
              "Upregulated insignificant",
              "Downregulated significant",
              "Upregulated significant")) +
  theme(axis.title.x = element_text(size = 15, vjust = -2),
        axis.title.y = element_text(size = 15, vjust = 2),
        axis.text.x = element_text(size = 12, vjust = -1),
        axis.text.y = element_text(size = 12),
        plot.background = element_rect(fill = "white"),
        panel.background = element_rect(fill = "white"),
        axis.line = element_line(linewidth = 0.5, colour = "black"),
        plot.margin = margin(10, 10, 10, 10),
        legend.position = c(0.25, 0.9)) +
  xlab("log2(mean abundance)") +
  ylab("log2(FC)") +
  xlim(-5, 3.5)
```



Gene Ontology enrichment analysis

The final step in the processing workflow is to apply Gene Ontology (GO) enrichment analyses to gain a biological understanding of the proteins which were either up or downregulated in HEK293 cells upon treatment. GO terms provide descriptions for genes and their corresponding proteins in the form of Molecular Functions (MF), Biological Processes (BP) and Cellular Components (CC). By carrying out GO enrichment analysis we can determine whether the frequency of any of these terms is higher than expected in the proteins of interest compared to all of the proteins which were detected. Such results can indicate whether proteins that were increased or decreased in abundance in treated HEK293 cells represent particular cellular locations, biological pathways or cellular functions.

Although GO enrichment analysis can be carried out online using websites such as GOrilla [46] or PantherDB [47, 48], we advise against this due to a lack of traceability and reproducibility. Instead, readers are advised to make use of GO enrichment packages within the Bioconductor infrastructure. Many such packages exist, including `topGO` [49], `GOfuncR` [50], and `clusterProfiler` [51]. Here we will use `enrichGO` function in the `clusterProfiler` package.

First, we subset the accessions of proteins that we consider to be significantly up or downregulated. These will be our proteins of interest.

```
## Subset significantly upregulated and downregulated  proteins
sig_up <- limma_results %>%
  filter(direction == "up") %>%
  filter(significance == "sig") %>%
  pull(Protein)

sig_down <- limma_results %>%
  filter(direction == "down") %>%
  filter(significance == "sig") %>%
  pull(Protein)
```

Next, we input the UniProt IDs of up and downregulated proteins into the GO enrichment analyses, as demonstrated below. Importantly, we provide the protein list of interest as the foreground and a list of all proteins identified within the study as the background, or 'universe'. The `keyType` argument is used to tell the function that our protein accessions are in UniProt format. This allows mapping from UniProt ID back to a database containing the entire human genome (`org.Hs.eg.db`). We also inform the function which GO categories we wish to consider, here "ALL", meaning BP, MF and CC.

As well as the information outlined above, there is the opportunity for users to specify various thresholds for statistical significance. These include thresholds on original and adjusted p-values (using the `pvalueCutoff` argument) as well as q-values (via the `qvalueCutoff` argument). Although many papers often use 'q- value' to mean 'BH-adjusted p-value', the two are not always the same and users should be explicit about the statistical thresholds that they have applied. For exploratory purposes we will use the standard BH method for FDR control and set p-value, BH-adjusted p-value, and q-value thresholds of 0.05.

```
## Search for enriched GO terms within upregulated proteins
ego_up <- enrichGO(gene = sig_up,
                   universe = limma_results$Protein,
                   OrgDb = org.Hs.eg.db,
                   keyType = "UNIPROT",
                   ont = "ALL",
                   pAdjustMethod = "BH",
                   pvalueCutoff = 0.05,
                   qvalueCutoff = 0.05,
                   readable = TRUE)

## Check results
ego_up
```

```
## #
## # over-representation test
## #
## #...@organism     Homo sapiens
## #...@ontology     GOALL
## #...@keytype      UNIPROT
## #...@gene     chr [1:313] "Q01581" "P15104" "Q9UK41" "P37268" "P04183" "Q9UHI8" "Q13907" ...
## #...pvalues adjusted by 'BH' with cutoff <0.05
## #...2 enriched terms found
## 'data.frame':    2 obs. of  10 variables:
##  $ ONTOLOGY   : chr  "CC" "CC"
##  $ ID         : chr  "GO:0005758" "GO:0031970"
##  $ Description: chr  "mitochondrial intermembrane space" "organelle envelope lumen"
##  $ GeneRatio  : chr  "13/308" "13/308"
##  $ BgRatio    : chr  "40/3176" "44/3176"
##  $ pvalue     : num  5.59e-05 1.68e-04
##  $ p.adjust   : num  0.0208 0.0313
##  $ qvalue     : num  0.0207 0.0312
##  $ geneID     : chr  "CHCHD2/TIMM9/AK2/TIMM8B/COA4/COA6/MIX23/TIMM8A/DIABLO/TIMM13/TIMM10/TRIAP1/
##  $ Count      : int  13 13
## #...Citation
##  T Wu, E Hu, S Xu, M Chen, P Guo, Z Dai, T Feng, L Zhou, W Tang, L Zhan, X Fu, S Liu, X Bo, and G
##  clusterProfiler 4.0: A universal enrichment tool for interpreting omics data.
##  The Innovation. 2021, 2(3):100141
```

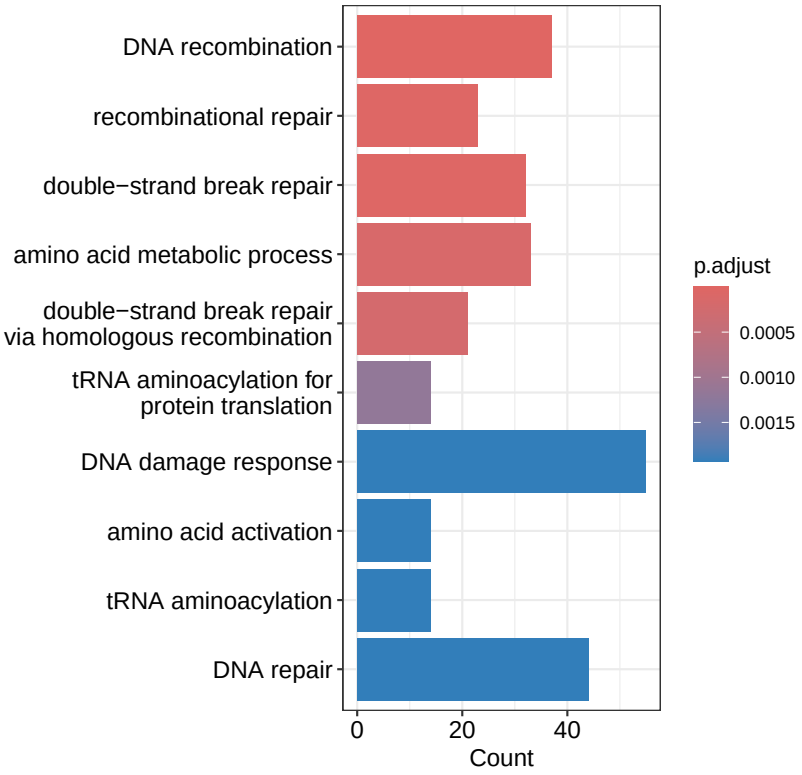
We can see from the results that there are 2 significantly enriched terms associated with the upregulated proteins. Next, we take a look at the downregulated proteins.

```
## Search for enriched GO terms within downregulated proteins
ego_down <- enrichGO(gene = sig_down,
                     universe = limma_results$Protein,
                     OrgDb = org.Hs.eg.db,
                     keyType = "UNIPROT",
                     ont = "ALL",
                     pAdjustMethod = "BH",
                     pvalueCutoff = 0.05,
                     qvalueCutoff = 0.05,
                     readable = TRUE)

## Check results
ego_down
```

```
## #
## # over-representation test
## #
## #...@organism     Homo sapiens
## #...@ontology     GOALL
## #...@keytype      UNIPROT
## #...@gene     chr [1:381] "Q53EL6" "P08243" "P35716" "Q92878" "P26583" "Q92522" "O43657" ...
## #...pvalues adjusted by 'BH' with cutoff <0.05
## #...61 enriched terms found
## 'data.frame':    61 obs. of  10 variables:
##  $ ONTOLOGY   : chr  "BP" "BP" "BP" "BP" ...
##  $ ID         : chr  "GO:0006310" "GO:0000725" "GO:0006302" "GO:0006520" ...
##  $ Description: chr  "DNA recombination" "recombinational repair" "double-strand break repair" "a
##  $ GeneRatio  : chr  "37/366" "23/366" "32/366" "33/366" ...
##  $ BgRatio    : chr  "95/3121" "58/3121" "101/3121" "113/3121" ...
##  $ pvalue     : num  3.61e-12 3.61e-08 4.54e-08 2.48e-07 4.46e-07 ...
##  $ p.adjust   : num  8.65e-09 3.62e-05 3.62e-05 1.48e-04 2.14e-04 ...
##  $ qvalue     : num  8.33e-09 3.48e-05 3.48e-05 1.43e-04 2.06e-04 ...
##  $ geneID     : chr  "RAD50/HMGB2/H1-10/RADX/MRE11/H1-0/H1-2/ZMYND8/MCM5/HMGB3/NUCKS1/RAD21/PRKDC
##  $ Count      : int  37 23 32 33 21 14 55 14 14 44 ...
## #...Citation
##  T Wu, E Hu, S Xu, M Chen, P Guo, Z Dai, T Feng, L Zhou, W Tang, L Zhan, X Fu, S Liu, X Bo, and G
##  clusterProfiler 4.0: A universal enrichment tool for interpreting omics data.
##  The Innovation. 2021, 2(3):100141
```

The downregulated proteins contain 61 significantly enriched GO terms. There are many ways in which users can represent these results visually. Here, we create a barplot using the `barplot` function from the `enrichplot` package [52]. Users are directed to the vignette of the `enrichplot` package for additional visualization options and guidance. We plot the first 10 GO terms i.e. the 10 GO terms with the greatest enrichment.

```
## Plot the results
barplot(ego_down,
        x = "Count",
        showCategory = 10,
        font.size = 12,
        label_format = 28,
        colorBy = "p.adjust")
```



Writing and exporting data

Finally, we export the results of our statistical analyses as `.csv` files.

```
## Save results of Limma statistics
write.csv(gene_results, file = "all_limma_results.csv")

## Save subsets of upregulated and downregulated proteins
write.csv(sig_upregulated, file = "upregulated_results.csv")
write.csv(sig_downregulated, file = "downregulated_results.csv"))

## Save results of GO enrichment
write.csv(ego_up, file = "upregulated_go_enrichment.csv")
write.csv(ego_down, file = "downregulated_go_enrichment.csv")
```

Users can also use the `ggsave` function to export any of the figures generated.

Discussion and conclusion

Expression proteomics is becoming an increasingly important tool in modern molecular biology. As more researchers participate in expression proteomics, either by collecting data or accessing data collected by others, there is a need for clear illustration(s) of how to deal with such complex data.

Existing bottom-up proteomics workflows for differential expression analysis either provide pipelines with limited user control and flexibility (e.g., `MSstats` and `MSstatsTMT` [38, 39]), can only be applied to specific data formats (e.g., `Proteus` which is limited to input from MaxQuant [53]), or provide very limited commentary. The latter directly contributes to a problematic disconnect between researchers and their data whereby the users do not understand if or why each step is necessary for their given dataset and biological question. This can prevent researchers from refining a workflow to fit their specific needs. Finally, the majority of proteomics workflows utilize `data.frame` or `tibble` structures which limits their traceability, as is the case for `protti`, `promor` and `prolfqua` [54, 55, 56].

The workflow presented here outlines in completion how to process, analyze and interpret LFQ and TMT expression proteomics data derived from a bottom-up DDA experiment. Critically, we emphasize quality control and data-guided decisions with an extensive explanation of all key steps and how they may differ in various scenarios (e.g., the quantitation method, instrumentation and biological question). Our workflow takes advantage of the relatively recent `QFeatures` infrastructure to ensure explicit and transparent data pre-processing

as well as to provide an easy way for users to trace back through their analyses. These features are particularly important for beginners who wish to gain a better understanding of their data and how it changes throughout this workflow.

No single workflow can demonstrate the processing, analysis and interpretation of all proteomics data. Our workflow is currently suitable for DDA datasets with label-free or TMT-based quantitation. We do not include examples of experiments that combine data from multiple TMTplexes, although the code provided could easily be expanded to include such a scenario. This workflow provides an in-depth user-friendly pipeline for both new and experienced proteomics data analysts.

Session information and getting help

The workflows provided involve use of functions from many different R/Bioconductor packages. The `sessionInfo` function provides an easy way to summarize all packages and corresponding their versions used to generate this document. Should software updates lead to the generation of errors or different results to those demonstrated here, such changes should be easily traced.

```
## Print session information
sessionInfo()
```

```
## R version 4.4.0 (2024-04-24)
## Platform: aarch64-apple-darwin20
## Running under: macOS Ventura 13.2
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack.dylib;  LAPAC
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: Europe/London
## tzcode source: internal
##
## attached base packages:
## [1] stats4    stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] patchwork_1.2.0             enrichplot_1.24.0
##  [3] clusterProfiler_4.12.0      org.Hs.eg.db_3.19.1
##  [5] AnnotationDbi_1.66.0        limma_3.60.0
##  [7] Biostrings_2.72.0           XVector_0.44.0
##  [9] corrplot_0.92               NormalyzerDE_1.22.0
## [11] tibble_3.2.1                dplyr_1.1.4
## [13] stringr_1.5.1               ggplot2_3.5.1
## [15] QFeatures_1.14.0            MultiAssayExperiment_1.30.1
## [17] SummarizedExperiment_1.34.0 Biobase_2.64.0
## [19] GenomicRanges_1.56.0        GenomeInfoDb_1.40.0
## [21] IRanges_2.38.0              S4Vectors_0.42.0
## [23] BiocGenerics_0.50.0         MatrixGenerics_1.16.0
## [25] matrixStats_1.3.0
##
## loaded via a namespace (and not attached):
##   [1] RColorBrewer_1.1-3       rstudioapi_0.16.0        jsonlite_1.8.8
##   [4] magrittr_2.0.3           farver_2.1.1             rmarkdown_2.26
##   [7] fs_1.6.4                 zlibbioc_1.50.0          vctrs_0.6.5
##  [10] memoise_2.0.1            ggtree_3.12.0            tinytex_0.51
##  [13] BiocBaseUtils_1.6.0      htmltools_0.5.8.1        S4Arrays_1.4.0
##  [16] usethis_2.2.3            gridGraphics_0.5-1       SparseArray_1.4.3
##  [19] plyr_1.8.9               impute_1.78.0            cachem_1.0.8
##  [22] igraph_2.0.3             lifecycle_1.0.4          pkgconfig_2.0.3
##  [25] gson_0.1.0               Matrix_1.7-0             R6_2.5.1
##  [28] fastmap_1.1.1            GenomeInfoDbData_1.2.12  clue_0.3-65
##  [31] digest_0.6.35            aplot_0.2.2              colorspace_2.1-0
```

```
##  [34] RSQLite_2.3.6            labeling_0.4.3           fansi_1.0.6
##  [37] httr_1.4.7               polyclip_1.10-6          abind_1.4-5
##  [40] compiler_4.4.0           bit64_4.0.5              withr_3.0.0
##  [43] BiocParallel_1.38.0      viridis_0.6.5            DBI_1.2.2
##  [46] ggforce_0.4.2            MASS_7.3-60.2            DelayedArray_0.30.1
##  [49] HDO.db_0.99.1            tools_4.4.0              scatterpie_0.2.2
##  [52] ape_5.8                  glue_1.7.0               nlme_3.1-164
##  [55] GOSemSim_2.30.0          shadowtext_0.1.3         grid_4.4.0
##  [58] cluster_2.1.6            reshape2_1.4.4           fgsea_1.30.0
##  [61] generics_0.1.3           gtable_0.3.5             preprocessCore_1.66.0
##  [64] tidyr_1.3.1              data.table_1.15.4        tidygraph_1.3.1
##  [67] utf8_1.2.4               ggrepel_0.9.5            pillar_1.9.0
##  [70] yulab.utils_0.1.4        BiocWorkflowTools_1.30.0 splines_4.4.0
##  [73] tweenr_2.0.3             treeio_1.28.0            lattice_0.22-6
##  [76] bit_4.0.5                tidyselect_1.2.1         GO.db_3.19.1
##  [79] knitr_1.46               git2r_0.33.0             gridExtra_2.3
##  [82] bookdown_0.39            ProtGenerics_1.36.0      xfun_0.43
##  [85] graphlayouts_1.1.1       statmod_1.5.0            stringi_1.8.4
##  [88] UCSC.utils_1.0.0         lazyeval_0.2.2           ggfun_0.1.4
##  [91] yaml_2.3.8               evaluate_0.23            codetools_0.2-20
##  [94] ggraph_2.2.1             MsCoreUtils_1.16.0       qvalue_2.36.0
##  [97] BiocManager_1.30.23      ggplotify_0.1.2          cli_3.6.2
## [100] munsell_0.5.1            Rcpp_1.0.12              png_0.1-8
## [103] parallel_4.4.0           blob_1.2.4               DOSE_3.30.0
## [106] AnnotationFilter_1.28.0  tidytree_0.4.6           viridisLite_0.4.2
## [109] scales_1.3.0             purrr_1.0.2              crayon_1.5.2
## [112] rlang_1.1.3              cowplot_1.1.3            fastmatch_1.1-4
## [115] KEGGREST_1.44.0
```

Users are advised to update R itself as well as packages as required. Bioconductor packages can be updated using the `BiocManager::install()` function, as shown below.

```
if (!require("BiocManager", quietly = TRUE)) {
  install.packages("BiocManager")
}
BiocManager::install()
```

Data availability

This workflow is written in the R statistical programming language and uses freely available open-source software packages from CRAN and Bioconductor. Version numbers for all packages are shown in the Session information section.

Raw mass spectrometry data is freely available online through the ProteomeXchange Consortium via the PRIDE repository with identifier PXD041794. All processed data is available at `http://doi.org/10.5281/zenodo.11196770`, and at GitHub repository `https://github.com/CambridgeCentreForProteomics/f1000_expression_proteomics`.

Author contributions

C. H. conceptualization, investigation, methodology, project administration, software, validation, writing – original draft preparation, review and editing; C. S. D. software and writing - review and editing; T. K. methodology, supervision, software, writing - review and editing; K. S. L. funding acquisition, supervision, writing - review and editing; L. M. B. conceptualization, methodology, supervision, writing - review and editing.

Competing interests

No competing interests were disclosed.

Grant information

C. H. is funded through a BBSRC CASE award with AstraZeneca (BB/W509929/1). C. S. D. is funded by a Herchel Smith Research Studentship at the University of Cambridge, United Kingdom. T. K. is funded by a

Gordon and Betty Moore Foundation Grant (#7872). K. S. L. is funded by Wellcome Trust (110071/Z/15/Z) and European Union Horizon 2020 program INFRAIA project EPIC-XS (project no.: 823839). L. M. B. is funded by European Union Horizon 2020 program INFRAIA project EPIC-XS (project no.: 823839).

The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Acknowledgements

The authors would like to thank Savvas Kourtis from the Centre for Genomic Regulation, Barcelona, and Oliver M. Crook, from the Department of Statistics, Oxford University, UK, for trialing this workflow.

Appendix

Identification search with Proteome Discoverer

The use-case data analyzed in this workflow was initially processed using Proteome Discoverer version 2.5. Whilst much of the identification and quantification takes place out of sight of the user, Proteome Discoverer incorporates several user-defined search parameters which must be specified according to the sample preparation methods and MS instrumentation used. There is also the option to apply both basic and advanced data filtering parameters during the search. Users must be aware of these parameters as they will directly influence the data output and downstream processing.

Whilst an in-depth discussion of identification searches is outside of the scope of this workflow, a few key parameters are discussed to put the data into context. During sample preparation, TMT-labelled cell pellets were combined and separated into 8 fractions using a Pierce High pH Reversed-Phase Peptide Fractionation Kit (Thermo Fisher Scientific). After being analyzed by MS, the 8 resulting raw files were uploaded to Proteome Discoverer 2.5 and processed using a single processing and consensus workflow. LFQ supernatant fractions were each analyzed on a separate mass spectrometry run resulting in 6 raw files. These files were imported into Proteome Discoverer with each sample having its own independent processing step followed by a single multi-consensus step. All processing and consensus workflow templates are provided in the supplementary materials.

For both TMT and LFQ workflows, SequestHT was selected as the search engine and trypsin specified as the enzyme used for proteolytic digestion. Since the digestion was carried out overnight with a 1:20 w/w ratio of trypsin:protein, digestion was expected to be complete and a low threshold of 2 missed cleavages was allowed. For MS analysis, a Fourier Transform orbitrap with a resolving power of 120,000 m/z was used as the mass analyzer for precursor ion mass, and a linear ion trap was used to measure fragment ion mass. This information determined the thresholds for precursor and fragment mass tolerances, two key parameters for the identification search. The precursor mass tolerance determines which mass range of peptide sequences are considered for each observed spectrum, whilst the fragment mass tolerance specifies how similar the observed and theoretical peptide fragment spectra should be for a match. If these tolerances are too narrow then the correct peptide sequence may be omitted and true positives are lost. However, if thresholds are set too wide then incorrect peptide sequences are considered and false positives arise. Based on the instrumentation used in this experiment, standard mass tolerances of 10 ppm and 0.5 Da were allowed for precursors and fragments, respectively. Given the intrinsic variability of LFQ between MS runs, RT alignment was used for the label-free samples with a 10-minute retention time window.

In addition to the parameters based on the experimental protocol, we also applied some basic non-specific filtering. We only retained high confidence PSMs from the identification search. Such filtering is necessary because only a fraction of the PSMs outputted by any given search engine will be genuine matches, or true discoveries, whilst the remainder are incorrect false discoveries. To deal with this problem, PSM confidence level (high, medium or low) is determined via the Proteome Discoverer Percolator node [57] which estimates each PSM's false discovery rate (FDR). The raw spectra are searched against the database of interest as well as a decoy database containing randomised peptide sequences, often generated by shuffling or reversing the original peptide sequences. False discovery rate is then defined as the proportion of total PSMs that are matched to the decoy database, and, therefore, are known false discoveries. This is done for all spectra and we considered a PSM to be of 'high confidence' if it had a false discovery rate <1 %, 'medium confidence' if <5 %, and 'low confidence' if the false discovery rate exceeded 5 %. Only PSMs annotated as high confidence were kept.

Whilst the basic filtering steps completed during this identification search could just have easily been carried out in R using the `SummarizedExperiment` and `QFeatures` infrastructure, applying them here saves time later on and reduces the burden of storing large data files. These steps are also relatively standard and non-specific so we do not need to assess the data prior to their implementation. However, Proteome Discoverer also provides the option to carry out more in-depth filtering through the use of parameters such as the SPS Mass Match %, co-isolation interference % and signal-to-noise thresholds. We advise against implementing

such filtering at this stage since decisions regarding thresholds will likely be influenced by the quality of data output, as demonstrated later in this workflow. Instead, thresholds for the three aforementioned parameters were set to 0 during the identification search.

References

[1] Emmanuel Pina-Jiménez, Fernando Calzada, Elihú Bautista, Rosa María Ordoñez-Razo, Claudia Velázquez, Elizabeth Barbosa, and Normand García-Hernández. Incomptine a induces apoptosis, ROS production and a differential protein expression on non-hodgkin's lymphoma cells. *International Journal of Molecular Sciences*, 22(19):10516, September 2021. doi: 10.3390/ijms221910516. URL https://doi.org/10.3390/ijms221910516.

[2] Nasrin Amiri-Dashatan, Nayebali Ahmadi, Mostafa Rezaei-Tavirani, and Mehdi Koushki. Identification of differential protein expression and putative drug target in metacyclic stage of leishmania major and leishmania tropica: A quantitative proteomics and computational view. *Comparative Immunology, Microbiology and Infectious Diseases*, 75:101617, April 2021. doi: 10.1016/j.cimid.2021.101617. URL https://doi.org/10.1016/j.cimid.2021.101617.

[3] Eduardo Anitua, María de la Fuente, Francisco Muruzabal, Ronald Mauricio Sánchez-Ávila, Jesús Merayo-Lloves, Mikel Azkargorta, Felix Elortza, and Gorka Orive. Differential profile of protein expression on human keratocytes treated with autologous serum and plasma rich in growth factors (PRGF). *PLOS ONE*, 13(10):e0205073, October 2018. doi: 10.1371/journal.pone.0205073. URL https://doi.org/10.1371/journal.pone.0205073.

[4] Emmalyn J. Dupree, Madhuri Jayathirtha, Hannah Yorkey, Marius Mihasan, Brindusa Alina Petre, and Costel C. Darie. A critical review of bottom-up proteomics: The good, the bad, and the future of this field. *Proteomes*, 8(3):14, July 2020. doi: 10.3390/proteomes8030014. URL https://doi.org/10.3390/proteomes8030014.

[5] Christian Obermaier, Anja Griebel, and Reiner Westermeier. Principles of protein labeling techniques. In *Methods in Molecular Biology*, pages 153–165. Springer New York, 2015. doi: 10.1007/978-1-4939-2550-6_13. URL https://doi.org/10.1007/978-1-4939-2550-6_13.

[6] Carolina Fernández-Costa, Salvador Martínez-Bartolomé, Daniel B. McClatchy, Anthony J. Saviola, Nam-Kyung Yu, and John R. Yates. Impact of the identification strategy on the reproducibility of the DDA and DIA results. *Journal of Proteome Research*, 19(8):3153–3161, June 2020. doi: 10.1021/acs.jproteome.0c00153. URL https://doi.org/10.1021/acs.jproteome.0c00153.

[7] Alex Hu, William S. Noble, and Alejandro Wolf-Yadlin. Technical advances in proteomics: new developments in data-independent acquisition. *F1000Research*, 5:419, March 2016. doi: 10.12688/f1000research.7042.1. URL https://doi.org/10.12688/f1000research.7042.1.

[8] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. URL http://www.R-project.org/. ISBN 3-900051-07-0.

[9] Wolfgang Huber, Vincent J Carey, Robert Gentleman, Simon Anders, Marc Carlson, Benilton S Carvalho, Hector Corrada Bravo, Sean Davis, Laurent Gatto, Thomas Girke, Raphael Gottardo, Florian Hahne, Kasper D Hansen, Rafael A Irizarry, Michael Lawrence, Michael I Love, James MacDonald, Valerie Obenchain, Andrzej K Oleś, Hervé Pagès, Alejandro Reyes, Paul Shannon, Gordon K Smyth, Dan Tenenbaum, Levi Waldron, and Martin Morgan. Orchestrating high-throughput genomic analysis with bioconductor. *Nature Methods*, 12(2):115–121, January 2015. doi: 10.1038/nmeth.3252. URL https://doi.org/10.1038/nmeth.3252.

[10] Charlotte Hutchings, Charlotte Dawson, Thomas Krueger, Kathryn Lilley, , and Lisa Breckels. *A Bioconductor workflow for processing, evaluating and interpreting expression proteomics data*, 2023. URL https://zenodo.org/record/7837375.

[11] Graeme C. McAlister, David P. Nusinow, Mark P. Jedrychowski, Martin Wühr, Edward L. Huttlin, Brian K. Erickson, Ramin Rad, Wilhelm Haas, and Steven P. Gygi. MultiNotch MS3 enables accurate, sensitive, and multiplexed detection of differential expression across cancer cell line proteomes. *Analytical Chemistry*, 86(14):7150–7158, July 2014. doi: 10.1021/ac502040v. URL https://doi.org/10.1021/ac502040v.

[12] Lily Ting, Ramin Rad, Steven P Gygi, and Wilhelm Haas. MS3 eliminates ratio distortion in isobaric multiplexed quantitative proteomics. *Nature Methods*, 8(11):937–940, October 2011. doi: 10.1038/nmeth.1714. URL https://doi.org/10.1038/nmeth.1714.

[13] Deanna L. Plubell, Phillip A. Wilmarth, Yuqi Zhao, Alexandra M. Fenton, Jessica Minnier, Ashok P. Reddy, John Klimek, Xia Yang, Larry L. David, and Nathalie Pamir. Extended multiplexing of tandem mass tags (TMT) labeling reveals age and high fat diet specific proteome changes in mouse epididymal adipose tissue. *Molecular and Cellular Proteomics*, 16(5):873–890, May 2017. doi: 10.1074/mcp.m116.065524. URL https://doi.org/10.1074/mcp.m116.065524.

[14] Alejandro Brenes, Jens Hukelmann, Dalila Bensaddek, and Angus Lamond. Multibatch TMT reveals false positives, batch effects and missing values. *Molecular and Cellular Proteomics*, 18(10):1967–1980, October 2019. doi: 10.1074/mcp.ra119.001472. URL https://doi.org/10.1074/mcp.ra119.001472.

[15] Yasset Perez-Riverol, Jingwen Bai, Chakradhar Bandla, David García-Seisdedos, Suresh Hewapathirana, Selvakumar Kamatchinathan, Deepti Kundu, Ananth Prakash, Anika Frericks-Zipper, Martin Eisenacher, Mathias Walzer, Shengbo Wang, Alvis Brazma, and Juan Antonio Vizcaíno. The PRIDE database resources in 2022: a hub for mass spectrometry-based proteomics evidences. *Nucleic Acids Research*, 50(D1):D543–D552, November 2021. doi: 10.1093/nar/gkab1038. URL https://doi.org/10.1093/nar/gkab1038.

[16] Eric W Deutsch, Nuno Bandeira, Yasset Perez-Riverol, Vagisha Sharma, Jeremy Carver, Luis Mendoza, Deepti J Kundu, Shengbo Wang, Chakradhar Bandla, Selvakumar Kamatchinathan, Suresh Hewapathirana, Benjamin Pullman, Julie Wertz, Zhi Sun, Shin Kawano, Shujiro Okuda, Yu Watanabe, Brendan MacLean, Michael MacCoss, Yunping Zhu, Yasushi Ishihama, and Juan Antonio Vizcaíno. The ProteomeXchange consortium at 10 years: 2023 update. *Nucleic Acids Research*, 51(D1):D1539–D1548, November 2022. doi: 10.1093/nar/gkac1040. URL https://doi.org/10.1093/nar/gkac1040.

[17] Laurent Gatto and Christophe Vanderaa. *QFeatures: Quantitative features for mass spectrometry data*, 2023. URL https://github.com/RforMassSpectrometry/QFeatures. R package version 1.9.2.

[18] Martin Morgan, Valerie Obenchain, Jim Hester, and Hervé Pagès. *SummarizedExperiment: SummarizedExperiment container*, 2022. URL https://bioconductor.org/packages/SummarizedExperiment. R package version 1.29.1.

[19] Johannes Rainer, Andrea Vicini, Liesa Salzer, Jan Stanstrup, Josep M. Badia, Steffen Neumann, Michael A. Stravs, Vinicius Verri Hernandes, Laurent Gatto, Sebastian Gibb, and Michael Witting. A modular and expandable ecosystem for metabolomics data annotation in r. *Metabolites*, 12(2):173, February 2022. doi: 10.3390/metabo12020173. URL https://doi.org/10.3390/metabo12020173.

[20] Ashley M. Frankenfield, Jiawei Ni, Mustafa Ahmed, and Ling Hao. Protein contaminants matter: Building universal protein contaminant libraries for DDA and DIA proteomics. *Journal of Proteome Research*, 21(9):2104–2113, July 2022. doi: 10.1021/acs.jproteome.2c00145. URL https://doi.org/10.1021/acs.jproteome.2c00145.

[21] H. Pages, P. Aboyoun, R. Gentleman, and S. DebRoy. *Biostrings: Efficient Manipulation of Biological Strings*, 2022. R package version 2.66.0.

[22] Yuliya Karpievitch, Jeff Stanley, Thomas Taverner, Jianhua Huang, Joshua N. Adkins, Charles Ansong, Fred Heffron, Thomas O. Metz, Wei-Jun Qian, Hyunjin Yoon, Richard D. Smith, and Alan R. Dabney. A statistical framework for protein quantitation in bottom-up MS-based proteomics. *Bioinformatics*, 25(16):2028–2034, June 2009. doi: 10.1093/bioinformatics/btp362. URL https://doi.org/10.1093/bioinformatics/btp362.

[23] Cosmin Lazar, Laurent Gatto, Myriam Ferro, Christophe Bruley, and Thomas Burger. Accounting for the multiple natures of missing salues in label-free quantitative proteomics data sets to compare imputation strategies. *Journal of Proteome Research*, 15(4):1116–1125, March 2016. doi: 10.1021/acs.jproteome.5b00981. URL https://doi.org/10.1021/acs.jproteome.5b00981.

[24] Adriaan Sticker, Ludger Goeminne, Lennart Martens, and Lieven Clement. Robust summarization and inference in proteome-wide label-free quantification. *Molecular and Cellular Proteomics*, 19(7):1209–1219, July 2020. doi: 10.1074/mcp.ra119.001624. URL https://doi.org/10.1074/mcp.ra119.001624.

[25] Ludger J. E. Goeminne, Kris Gevaert, and Lieven Clement. Peptide-level robust ridge regression improves estimation, sensitivity, and specificity in data-dependent quantitative label-free shotgun proteomics. *Molecular and Cellular Proteomics*, 15(2):657–668, February 2016. doi: 10.1074/mcp.m115.055897. URL https://doi.org/10.1074/mcp.m115.055897.

[26] Matthew B. O'Rourke, Stephanie E. L. Town, Penelope V. Dalla, Fiona Bicknell, Naomi Koh Belic, Jake P. Violi, Joel R. Steele, and Matthew P. Padula. What is normalization? the strategies employed in top-down and bottom-up proteome analysis workflows. *Proteomes*, 7(3):29, August 2019. doi: 10.3390/proteomes7030029. URL https://doi.org/10.3390/proteomes7030029.

[27] Jakob Willforss, Aakash Chawade, and Fredrik Levander. NormalyzerDE: Online tool for improved normalization of omics expression data and high-sensitivity differential expression analysis. *Journal of Proteome Research*, 18(2):732–740, October 2018. doi: 10.1021/acs.jproteome.8b00523. URL https://doi.org/10.1021/acs.jproteome.8b00523.

[28] Ben Bolstad. *preprocessCore: A collection of pre-processing functions*, 2023. R package version 1.60.2.

[29] Claus Lindbjerg Andersen, Jens Ledet Jensen, and Torben Falck Ørntoft. Normalization of real-time quantitative reverse transcription-PCR data: A model-based variance estimation approach to identify genes suited for normalization, applied to bladder and colon cancer data sets. *Cancer Research*, 64(15):5245–5250, August 2004. doi: 10.1158/0008-5472.can-04-0496. URL https://doi.org/10.1158/0008-5472.can-04-0496.

[30] Wolfgang Huber, Anja von Heydebreck, Holger Sültmann, Annemarie Poustka, and Martin Vingron. Variance stabilization applied to microarray data calibration and to the quantification of differential expression. *Bioinformatics*, 18 (suppl_1):S96–S104, July 2002. doi: 10.1093/bioinformatics/18.suppl_1.s96. URL https://doi.org/10.1093/bioinformatics/18.suppl_1.s96.

[31] Gordon K Smyth. Linear models and empirical bayes methods for assessing differential expression in microarray experiments. *Statistical Applications in Genetics and Molecular Biology*, 3(1):1–25, January 2004. doi: 10.2202/1544-6115.1027. URL https://doi.org/10.2202/1544-6115.1027.

[32] Ludger J. E. Goeminne, Adriaan Sticker, Lennart Martens, Kris Gevaert, and Lieven Clement. MSqRob takes the missing hurdle: Uniting intensity- and count-based proteomics. *Analytical Chemistry*, 92(9):6278–6287, March 2020. doi: 10.1021/acs.analchem.9b04375. URL https://doi.org/10.1021/acs.analchem.9b04375.

[33] Mingyi Liu and Ashok Dongre. Proper imputation of missing values in proteomics datasets for differential expression analysis. *Briefings in Bioinformatics*, 22(3), June 2021. doi: 10.1093/bib/bbaa112. URL https://doi.org/10.1093/bib/bbaa112.

[34] Rafael Irizarry. *Correlation is not a measure of reproducibility*, 2015. URL https://simplystatistics.org/posts/2015-08-12-correlation-is-not-a-measure-of-reproducibility/.

[35] Karina V. Bunting, Richard P. Steeds, Luke T. Slater, Jennifer K. Rogers, Georgios V. Gkoutos, and Dipak Kotecha. A practical guide to assess the reproducibility of echocardiographic measurements. *Journal of the American Society of Echocardiography*, 32(12):1505–1515, December 2019. doi: 10.1016/j.echo.2019.08.015. URL https://doi.org/10.1016/j.echo.2019.08.015.

[36] Behrooz Darbani and Charles Neal Stewart. Reproducibility and reliability assays of the gene expression-measurements. *Journal of Biological Research-Thessaloniki*, 21(1), May 2014. doi: 10.1186/2241-5793-21-3. URL https://doi.org/10.1186/2241-5793-21-3.

[37] Jeffrey T. Leek, Robert B. Scharpf, Héctor Corrada Bravo, David Simcha, Benjamin Langmead, W. Evan Johnson, Donald Geman, Keith Baggerly, and Rafael A. Irizarry. Tackling the widespread and critical impact of batch effects in high-throughput data. *Nature Reviews Genetics*, 11(10):733–739, September 2010. doi: 10.1038/nrg2825. URL https://doi.org/10.1038/nrg2825.

[38] Meena Choi, Ching-Yun Chang, Timothy Clough, Daniel Broudy, Trevor Killeen, Brendan MacLean, and Olga Vitek. MSstats: an r package for statistical analysis of quantitative mass spectrometry-based proteomic experiments. *Bioinformatics*, 30(17):2524–2526, May 2014. doi: 10.1093/bioinformatics/btu305. URL https://doi.org/10.1093/bioinformatics/btu305.

[39] Ting Huang, Meena Choi, Manuel Tzouros, Sabrina Golling, Nikhil Janak Pandya, Balazs Banfai, Tom Dunkley, and Olga Vitek. MSstatsTMT: Statistical detection of differentially abundant proteins in experiments with isobaric labeling and multiple mixtures. *Molecular and Cellular Proteomics*, 19(10):1706–1723, October 2020. doi: 10.1074/mcp.ra120.002105. URL https://doi.org/10.1074/mcp.ra120.002105.

[40] Witold E. Wolski, Paolo Nanni, Jonas Grossmann, Maria d'Errico, Ralph Schlapbach, and Christian Panse. prolfqua: A comprehensive R-package for proteomics differential expression analysis. *Journal of Proteome Research*, 22(4):1092–1104, March 2023. doi: 10.1021/acs.jproteome.2c00441. URL https://doi.org/10.1021/acs.jproteome.2c00441.

[41] Matthew E. Ritchie, Belinda Phipson, Di Wu, Yifang Hu, Charity W. Law, Wei Shi, and Gordon K. Smyth. limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research*, 43(7):e47–e47, January 2015. doi: 10.1093/nar/gkv007. URL https://doi.org/10.1093/nar/gkv007.

[42] Belinda Phipson, Stanley Lee, Ian J. Majewski, Warren S. Alexander, and Gordon K. Smyth. Robust hyperparameter estimation protects against hypervariable genes and improves power to detect differential expression. *The Annals of Applied Statistics*, 10(2), June 2016. doi: 10.1214/16-aoas920. URL https://doi.org/10.1214/16-aoas920.

[43] Charity W Law, Yunshun Chen, Wei Shi, and Gordon K Smyth. voom: precision weights unlock linear model analysis tools for RNA-seq read counts. *Genome Biology*, 15(2):R29, 2014. doi: 10.1186/gb-2014-15-2-r29. URL https://doi.org/10.1186/gb-2014-15-2-r29.

[44] David Robinson. *How to interpret a p-value histogram*, 2014. URL http://varianceexplained.org/statistics/interpreting-pvalue-histogram/.

[45] Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of Royal Statistical Society*, 57(1):289–300, 1995. doi: 10.1111/j.2517-6161.1995.tb02031.x.

[46] Eran Eden, Roy Navon, Israel Steinfeld, Doron Lipson, and Zohar Yakhini. GOrilla: a tool for discovery and visualization of enriched GO terms in ranked gene lists. *BMC Bioinformatics*, 10(1), February 2009. doi: 10.1186/1471-2105-10-48. URL https://doi.org/10.1186/1471-2105-10-48.

[47] Huaiyu Mi, Anushya Muruganujan, and Paul D. Thomas. PANTHER in 2013: modeling the evolution of gene function, and other gene attributes, in the context of phylogenetic trees. *Nucleic Acids Research*, 41(D1):D377–D386, November 2012. doi: 10.1093/nar/gks1118. URL https://doi.org/10.1093/nar/gks1118.

[48] Paul D. Thomas, Dustin Ebert, Anushya Muruganujan, Tremayne Mushayahama, Laurent-Philippe Albou, and Huaiyu Mi. PANTHER: Making genome-scale phylogenetics accessible to all. *Protein Science*, 31(1):8–22, November 2021. doi: 10.1002/pro.4218. URL https://doi.org/10.1002/pro.4218.

[49] Adrian Alexa. *topGO: Enrichment Analysis for Gene Ontology*, 2022. R package version 2.50.0.

[50] Steffi Grote. *GOfuncR: Gene ontology enrichment using FUNC*, 2022. R package version 1.18.0.

[51] Tianzhi Wu, Erqiang Hu, Shuangbin Xu, Meijun Chen, Pingfan Guo, Zehan Dai, Tingze Feng, Lang Zhou, Wenli Tang, Li Zhan, Xiaocong Fu, Shanshan Liu, Xiaochen Bo, and Guangchuang Yu. clusterProfiler 4.0: A universal enrichment tool for interpreting omics data. *The Innovation*, 2(3):100141, August 2021. doi: 10.1016/j.xinn.2021.100141. URL https://doi.org/10.1016/j.xinn.2021.100141.

[52] Guangchuang Yu. *enrichplot: Visualization of Functional Enrichment Result*, 2022. R package version 1.18.3.

[53] Marek Gierlinski, Francesco Gastaldello, Chris Cole, and Geoffrey J. Barton. Proteus: an r package for downstream analysis of MaxQuant output. September 2018. doi: 10.1101/416511. URL https://doi.org/10.1101/416511.

[54] Chathurani Ranathunge, Sagar S. Patel, Lubna Pinky, Vanessa L. Correll, Shimin Chen, O. John Semmes, Robert K. Armstrong, C. Donald Combs, and Julius O. Nyalwidhe. promor: a comprehensive r package for label-free proteomics data analysis and predictive modeling. August 2022. doi: 10.1101/2022.08.17.503867. URL https://doi.org/10.1101/2022.08.17.503867.

[55] Jan-Philipp Quast, Dina Schuster, and Paola Picotti. protti: an r package for comprehensive data analysis of peptide- and protein-centric bottom-up proteomics data. *Bioinformatics Advances*, 2(1), December 2021. doi: 10.1093/bioadv/vbab041. URL https://doi.org/10.1093/bioadv/vbab041.

[56] Witold E. Wolski, Paolo Nanni, Jonas Grossmann, Maria d'Errico, Ralph Schlapbach, and Christian Panse. prolfqua: A comprehensive r-package for proteomics differential expression analysis. June 2022. doi: 10.1101/2022.06.07.494524. URL https://doi.org/10.1101/2022.06.07.494524.

[57] Lukas Käll, Jesse D Canterbury, Jason Weston, William Stafford Noble, and Michael J MacCoss. Semi-supervised learning for peptide identification from shotgun proteomics datasets. *Nature Methods*, 4(11):923–925, October 2007. doi: 10.1038/nmeth1113. URL https://doi.org/10.1038/nmeth1113.