

proteomeR package

Arne Smits

10 April 2017

Package version: proteomeR 1.1.2

Contents

1 Overview of the analysis	1
2 Installation	2
3 Interactive analysis using the proteomeR Shiny apps	2
4 Differential analysis	2
4.1 Example dataset for differential enrichment analysis: Ubiquitin interactors	2
4.2 Loading of the data	2
4.3 Data preparation	3
4.4 Generate an SummarizedExperiment object	4
4.5 Filter for missing values	5
4.6 Normalization	7
4.7 Impute data for missing values	7
4.8 Differential enrichment analysis	10
4.9 Visualization of the results	11
4.10 Results table	16
5 Stoichiometry analysis	17
5.1 Example dataset for stoichiometry analysis: PRC2 interactors	17
5.2 Loading and preparation of the data, and generation of a SummarizedExperiment object	17
5.3 Filter on missing values, normalization and imputation	17
5.4 Differential enrichment analysis	21
5.5 Stoichiometry analysis	21
6 Wrapper functions for the entire workflow	23
6.1 LFQ analysis	23
6.2 iBAQ-based stoichiometry analysis	25
6.3 TMT analysis	27

1 Overview of the analysis

This analysis workflow provides an integrated solution for robust and reproducible differential analysis (protein expression or protein enrichment) of proteomics data. It requires tabular input (e.g. txt files) which are generated by quantitative analysis softwares of raw mass spectrometry data, such as [MaxQuant](#) or [IsobarQuant](#).

Functions are provided for data preparation, data filtering, variance normalization and imputation of missing values, as well as statistical testing of differential enriched/expressed proteins. It also includes tools to check intermediate steps within the workflow, such as normalization and missing values imputation. Finally, visualization tools are provided to explore the results, including heatmap, volcano plot and barplot representations. Besides differential analysis, the package includes a set of functions to calculate relative stoichiometry, as originally described in [Smits et al \(2013\)](#).

For scientists with limited experience in R, the package also entails wrapper functions that entail the complete analysis

workflow and generate a report (see the chapter on [Wrapper functions](#)). Even easier to use are the interactive Shiny apps that are provided by the package (see the chapter on [Shiny apps](#)).

2 Installation

Install and load the package

```
devtools::install_git("git@git.embl.de:smits/proteomeR.git")
library("proteomeR")
```

3 Interactive analysis using the proteomeR Shiny apps

The package contains two [shiny](#) apps, which allow for interactive analysis entailing the entire workflow described below. These are especially relevant for users with limited or no experience in R. Currently, there are three different apps. One for label-free quantification (LFQ)-based analysis (output from MaxQuant) and the second for tandem-mass-tags (TMT)-based analysis (output from IsoBarQuant). The third app, is a extension of the LFQ-based analysis for iBAQ-based stoichiometry analysis. To run the apps, simply run this single command:

```
# For LFQ analysis
run_app("LFQ")

# For TMT analysis
run_app("TMT")

# For iBAQ-based stoichiometry analysis
run_app("iBAQ")
```

4 Differential analysis

4.1 Example dataset for differential enrichment analysis: Ubiquitin interactors

We analyze a proteomics dataset in which Ubiquitin-protein interactors were characterized ([Zhang et al. Mol Cell 2017](#)). The raw mass spectrometry data were first analyzed using MaxQuant ([Cox and Mann, Nat Biotech 2007](#)).

4.2 Loading of the data

```
# Loading two packages required for data handling
library(tidyverse)
library(magrittr)

# The data is provided with the package
data <- UbiLength

# We filter for contaminant proteins and decoy database hits, which are indicated by "+"
# in the columns "Potential.contaminants" and "Reverse", respectively.
data %<>% filter(Reverse != "+", Potential.contaminant != "+")
```

The data.frame has the following dimensions:

```
dim(data)
## [1] 2941 35
```

The data.frame has the following column names:

```
colnames(data)
## [1] "Protein.IDs"          "Majority.protein.IDs"
## [3] "Protein.names"        "Gene.names"
## [5] "Fasta.headers"        "Peptides"
## [7] "Razor...unique.peptides" "Unique.peptides"
## [9] "Intensity.Ubi4_1"      "Intensity.Ubi4_2"
## [11] "Intensity.Ubi4_3"      "Intensity.Ubi6_1"
## [13] "Intensity.Ubi6_2"      "Intensity.Ubi6_3"
## [15] "Intensity.Ctrl_1"      "Intensity.Ctrl_2"
## [17] "Intensity.Ctrl_3"      "Intensity.Ubi1_1"
## [19] "Intensity.Ubi1_2"      "Intensity.Ubi1_3"
## [21] "LFQ.intensity.Ubi4_1"  "LFQ.intensity.Ubi4_2"
## [23] "LFQ.intensity.Ubi4_3"  "LFQ.intensity.Ubi6_1"
## [25] "LFQ.intensity.Ubi6_2"  "LFQ.intensity.Ubi6_3"
## [27] "LFQ.intensity.Ctrl_1"  "LFQ.intensity.Ctrl_2"
## [29] "LFQ.intensity.Ctrl_3"  "LFQ.intensity.Ubi1_1"
## [31] "LFQ.intensity.Ubi1_2"  "LFQ.intensity.Ubi1_3"
## [33] "Only.identified.by.site" "Reverse"
## [35] "Potential.contaminant"
```

The “LFQ.intensity” columns will be used for subsequent analysis.

4.3 Data preparation

The dataset has unique Uniprot identifiers, however those are not immediately informative. The associated gene names are informative, however these are not always unique.

```
# Are there any duplicated gene names?
data$Gene.names %>% duplicated(.) %>% any(.)
## [1] TRUE

# Make a table of duplicated gene names
data %>% group_by(Gene.names) %>% summarize(frequency = n()) %>%
  arrange(desc(frequency)) %>% filter(frequency > 1)
## # A tibble: 51 × 2
##   Gene.names frequency
##   <chr>         <int>
## 1              7
## 2      ATXN2         4
## 3    ATXN2L         4
## 4        SF1         4
## 5     HSPA8         3
## 6     RBM33         3
## 7     UGP2         3
## 8    ACTL6A         2
## 9    BCLAF1         2
## 10    BRAP         2
## # ... with 41 more rows
```

For further analysis these proteins must get unique names. Additionally, some proteins do not have an annotated gene

name and for those we will use the Uniprot ID.

```
# Make unique names using the annotation in the "Gene.names" column as primary names
# and the annotation in "Protein.IDs" as name for those that do not have an gene name.
data_unique <- unique_names(data, "Gene.names", "Protein.IDs", delim = ";")
## Joining, by = c("Protein.IDs", "Gene.names")

# Are there any duplicated names?
data$name %>% duplicated(.) %>% any(.)
## [1] FALSE
```

4.4 Generate an SummarizedExperiment object

Many [Bioconductor](#) packages use SummarizedExperiment objects as input and/or output. This class of objects contains and coordinates the actual (assay) data, information on the samples as well as feature annotation. We can generate the SummarizedExperiment object from our data using two different approaches. We can extract sample information directly from the column names or we add sample information using an experimental design template. For our example dataset, an experimental design is included in the package.

label	condition	replicate
Ubi4_1	Ubi4	1
Ubi4_2	Ubi4	2
Ubi4_3	Ubi4	3
Ubi6_1	Ubi6	1
Ubi6_2	Ubi6	2
Ubi6_3	Ubi6	3
Ctrl_1	Ctrl	1
Ctrl_2	Ctrl	2
Ctrl_3	Ctrl	3
Ubi1_1	Ubi1	1
Ubi1_2	Ubi1	2
Ubi1_3	Ubi1	3

```
# Generate an SummarizedExperiment object with an experimental design
LFQ_columns <- grep("LFQ.", colnames(data_unique)) # get LFQ column numbers
experimental_design <- UbiLength_ExpDesign
data_se <- make_se(data_unique, LFQ_columns, experimental_design)

# Generate an SummarizedExperiment object by parsing condition information from the column names
LFQ_columns <- grep("LFQ.", colnames(data_unique)) # get LFQ column numbers
data_se_parsed <- make_se_parse(data_unique, LFQ_columns)

# Let's have a look at the SummarizedExperiment object
data_se
## class: SummarizedExperiment
## dim: 2941 12
## metadata(0):
## assays(1): ''
## rownames(2941): RBM47 UBA6 ... ATXN2.3 X6RHB9
## rowData names(25): Protein.IDs Majority.protein.IDs ... name ID
## colnames(12): Ubi4_1 Ubi4_2 ... Ubi1_2 Ubi1_3
## colData names(4): label ID condition replicate
```

4.5 Filter for missing values

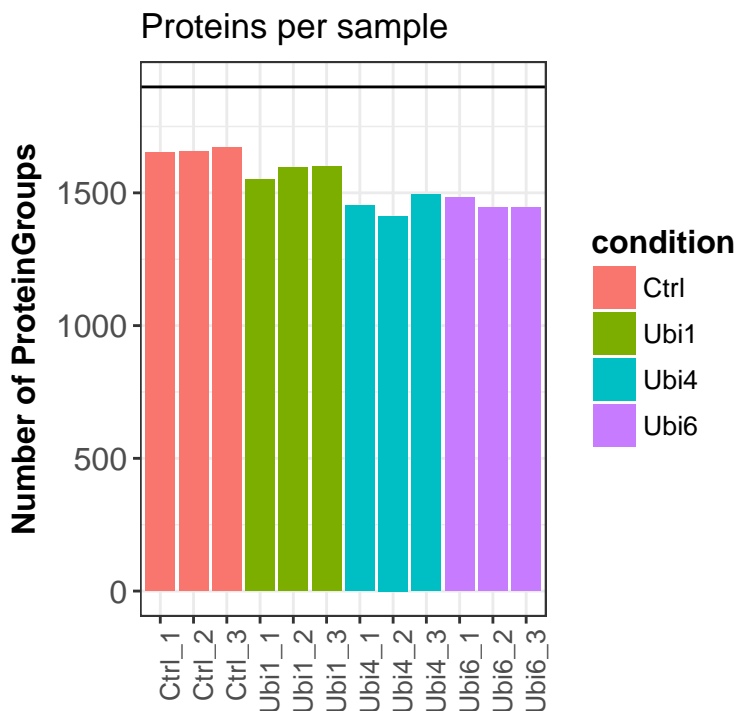
The dataset contains proteins which are not quantified in all replicates. Some proteins are even only detected in a single replicate. This leaves our dataset with missing values, which need to be imputed. However, this should not be done for proteins that contain too many missing values. Therefore, we first filter out proteins that contain too many missing values. This is done by setting the threshold for the allowed number of missing values per condition in the `filter_missval` function.

```
# Filter for proteins that are identified in all replicates of at least one condition
data_filt <- filter_missval(data_se, thr = 0)

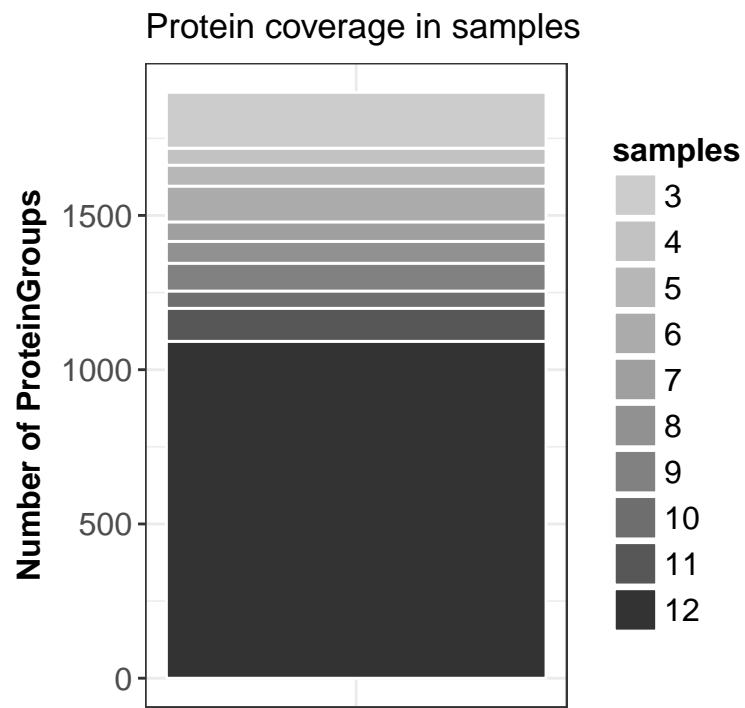
# Less stringent filtering:
# Filter for proteins that are identified in 2 out of 3 replicates of at least one condition
data_filt2 <- filter_missval(data_se, thr = 1)
```

After filtering the number of identified proteins per sample can be plotted as well as the overlap in identifications between samples.

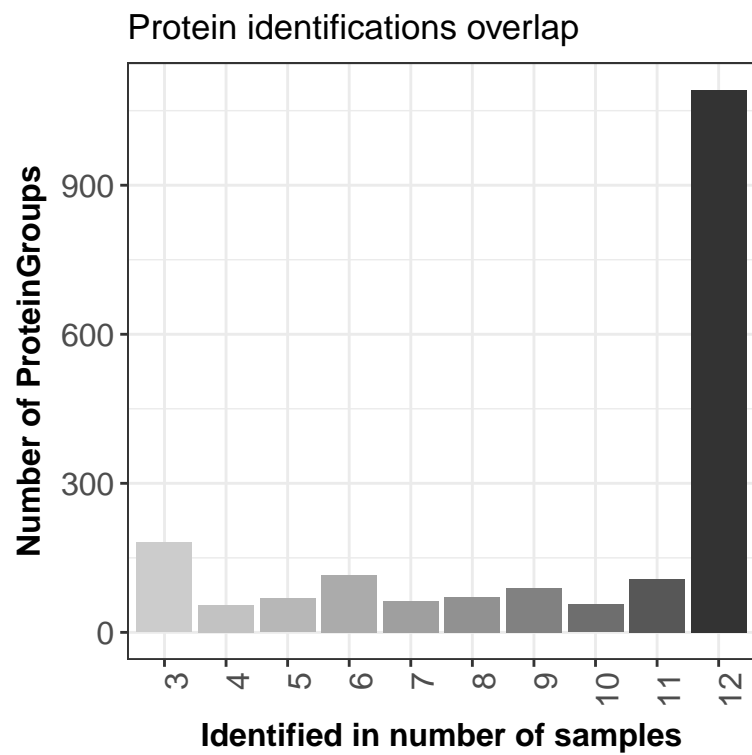
```
# Plot a barplot of the number of identified proteins per samples
plot_numbers(data_filt)
```



```
# Plot a barplot of the protein identification overlap between samples
plot_coverage(data_filt)
```



```
# Plot a barplot of the protein identification overlap between samples  
plot_frequency(data_filt)
```



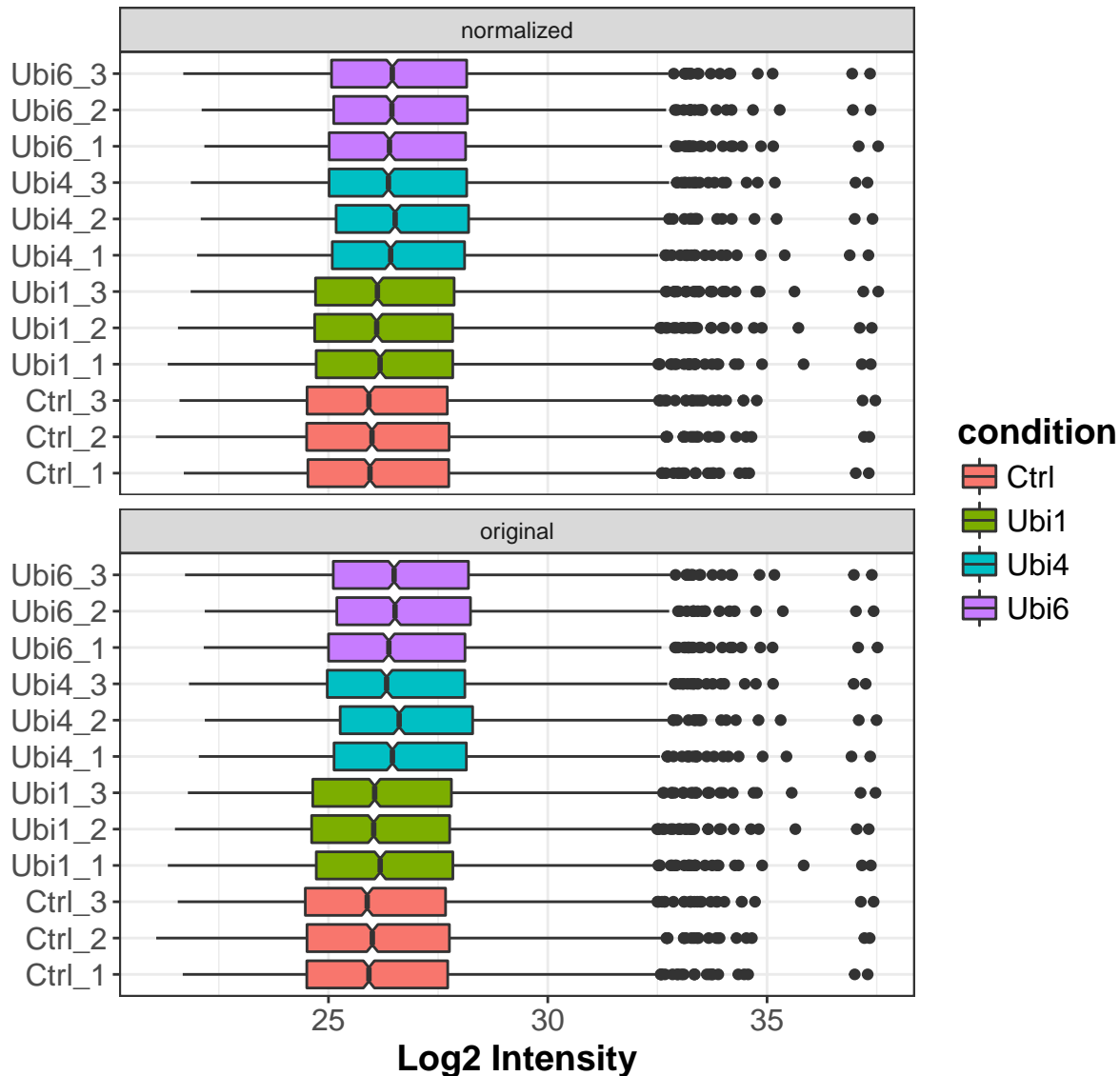
4.6 Normalization

The data is background corrected and normalized by variance stabilization transformation (*vsn*).

```
# Normalize the data
data_norm <- norm_vsn(data_filt)
## vsn2: 1899 x 12 matrix (1 stratum). Please use 'meanSdPlot' to verify the fit.
```

The normalization can be inspected by checking the distributions of the samples before and after normalization.

```
# Visualize normalization by boxplots for all samples before and after normalization
plot_norm(data_filt, data_norm)
```



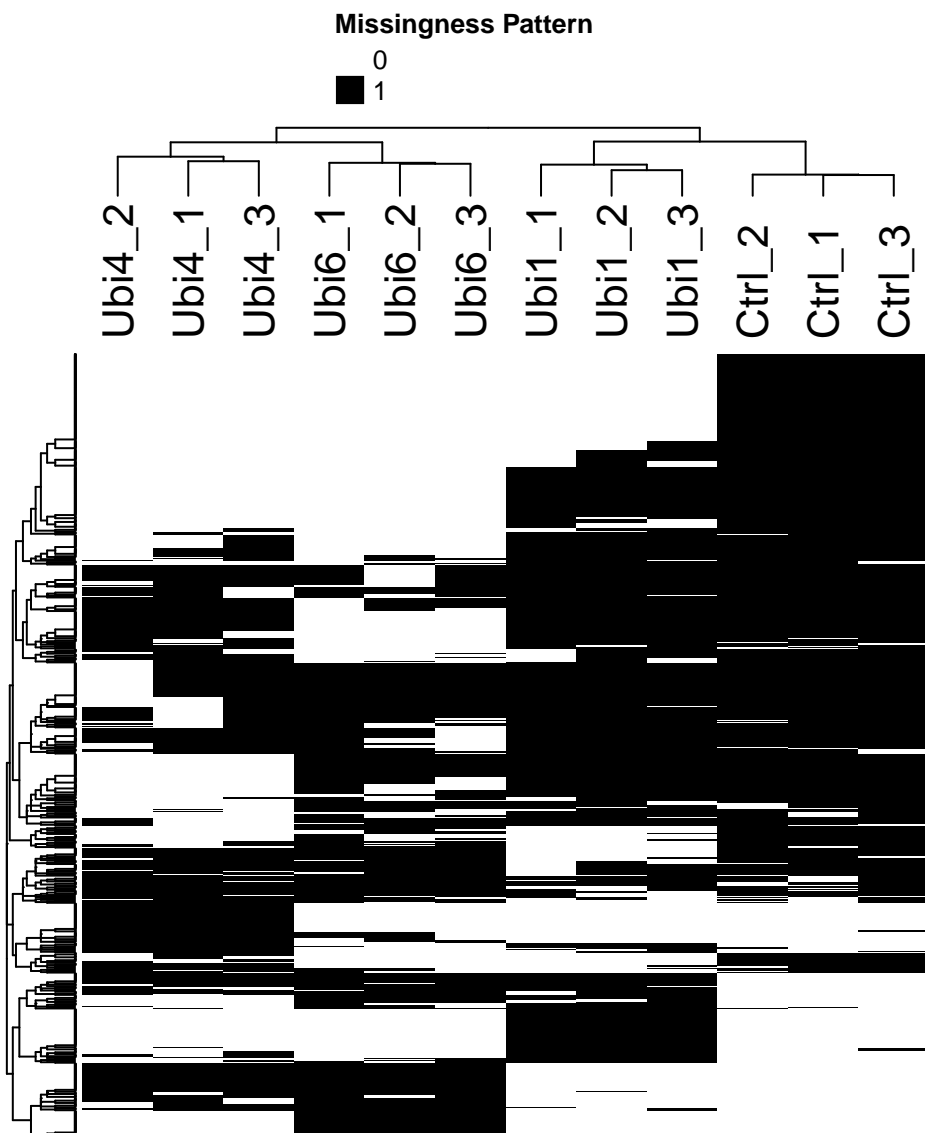
4.7 Impute data for missing values

The remaining missing values in the dataset need to be imputed. The data can be missing at random (MAR), for example if proteins are quantified in some replicates but not in others, or missing not at random (MNAR), for example if proteins which are quantified in some replicates but not in others (e.g. in all replicates of the control). MNAR can indicate that

the proteins are below the detection limit in specific samples, which could be very well the case for, for example, affinity enrichments. For these different conditions, different imputation methods have to be used, as described in the [MSnbase](#) vignette and more specifically in the *impute* function descriptions.

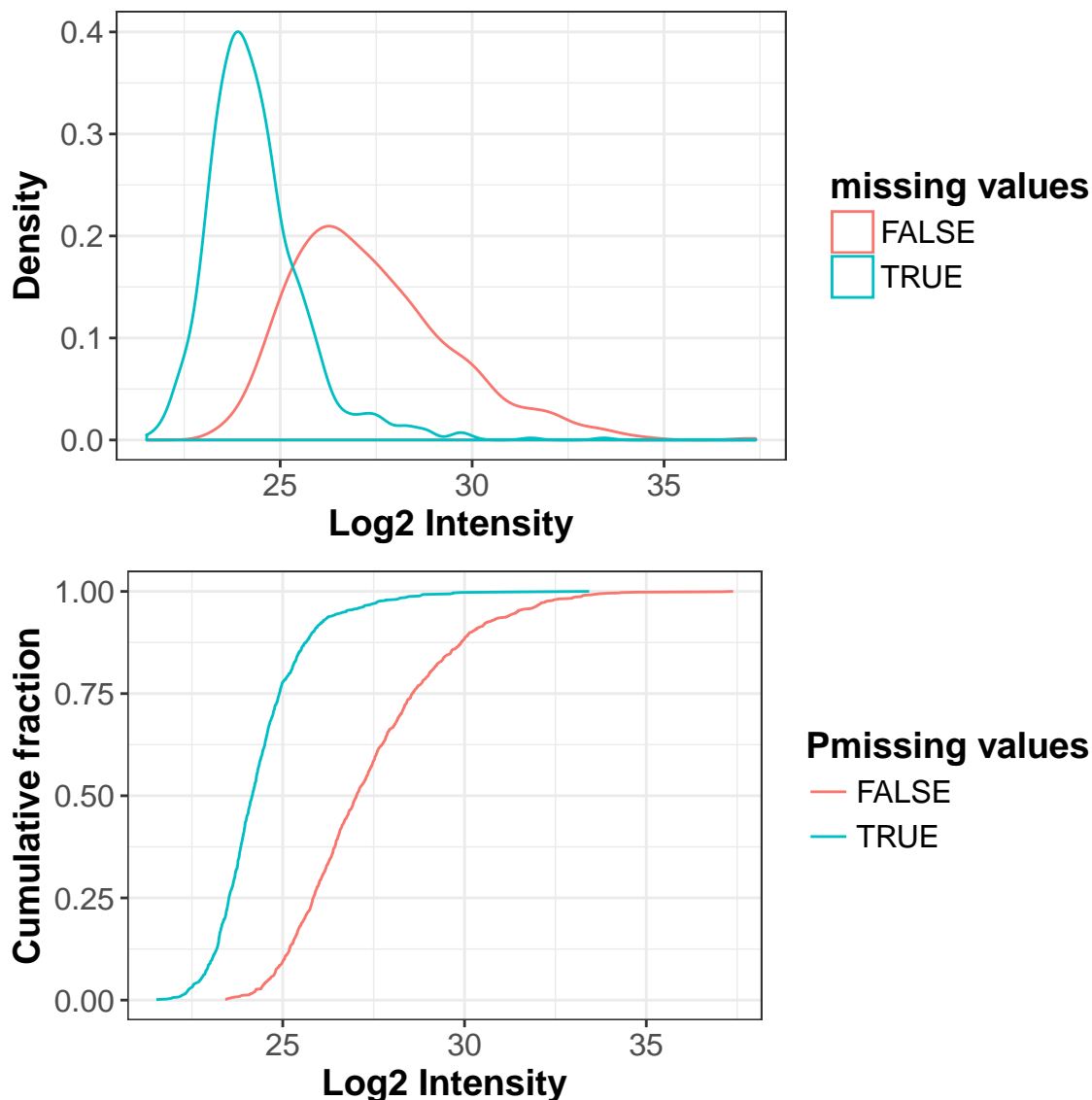
First, the missingness pattern of the data is explored by plotting a heatmap indicating whether values are missing (0) or not (1).

```
# Plot a heatmap of proteins with missing values
plot_missval(data_filt)
```



The example dataset is an affinity enrichment dataset of ubiquitin interactors, which is likely to have proteins which are below the detection limit in specific samples. These can be proteins that are specifically enriched in the ubiquitin purifications, but are not enriched in the controls samples, or vice versa. This expectation is also reflected in the missingness heatmap; missing values are highly biased to specific samples. To check whether missing values are biased to lower intense proteins, we visualize the densities and cumulative sums of proteins with and without missing values.

```
# Plot intensity distributions and cumulative fraction
# of proteins with and without missing values
plot_detect(data_filt)
```

The imputation method to be used depends on the missingness pattern of the data (see the [MSnbase](#) vignette and more specifically the `impute` function description for more information). Values which are missing not at random (MNAR) should be imputed by left-censored imputation method, such as the quantile regression-based left-censored function ("QRILC") or random draws from a left-shifted distribution ("MinProb"). In contrast, data missing at random (MAR) should be imputed with methods such as k-nearest neighbour ("knn") or maximum likelihood ("MLE") functions.

As the example data has left-censored missing values (missing not at random), 'MinProb' or 'QRILC' imputation are appropriate imputation methods.

```
# Impute missing data using random draws from a Gaussian distribution centered around
# a minimal value (for data not missing at random)
data_imp <- imputation(data_norm, fun = "MinProb", q = 0.01)

# Impute missing data using the quantile regression-based left-censored
# imputation method (QRILC) (for data not missing at random)
data_imp_QRILC <- imputation(data_norm, fun = "QRILC")

# Impute missing data using random draws from a manually defined
```

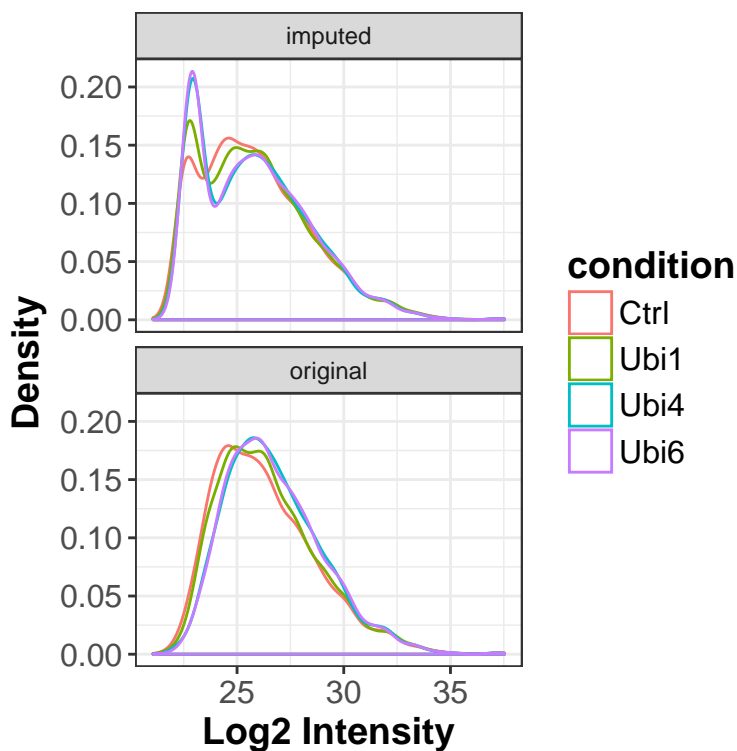
```
# left-shifted Gaussian distribution (for data not missing at random)
data_imp_man <- imputation(data_norm, fun = "man", shift = 1.8, scale = 0.3)

# Impute missing data using the k-nearest neighbour approach
# (for data missing at random)
data_imp_knn <- imputation(data_norm, fun = "knn")

# Impute missing data using the k-nearest neighbour approach
#(for data missing at random)
data_imp_MLE <- imputation(data_norm, fun = "MLE")
```

The effect of the imputation on the distributions can be visualized.

```
# Plot intensity distributions before and after imputation
plot_imp(data_norm, data_imp)
```



4.8 Differential enrichment analysis

Linear models combined with empirical Bayes statistics is used for the analysis of differential enrichment (or differential expression for analysis on expression levels). The *linear_model* wrapper function introduced here uses *limma* and automatically generates the contrasts to be tested. For the contrasts generation, the control sample has to be specified. Additionally, the types of contrasts to be produced need to be indicated, allowing the generation of all possible comparisons ("all") or the generation of contrasts of every sample versus control ("control").

```
# Differential enrichment analysis based on linear models and empirical Bayes statistics
# Test every sample versus control
data_lm <- linear_model(data_imp, control = "Ctrl", type = "control")
## Tested contrasts:
## [1] "Ubi1_vs_Ctrl" "Ubi4_vs_Ctrl" "Ubi6_vs_Ctrl"
```

```
# Test all possible comparisons of samples
data_lm_all_contrasts <- linear_model(data_imp_QRILC, control = "Ctrl", type = "all")
## Tested contrasts:
## [1] "Ubi1_vs_Ctrl" "Ubi4_vs_Ctrl" "Ubi6_vs_Ctrl" "Ubi1_vs_Ubi4"
## [5] "Ubi1_vs_Ubi6" "Ubi4_vs_Ubi6"
```

An ANOVA and post-hoc Tukey Honest Significant Differences analysis is also implemented, although the use of *linear_models* based on *limma* is recommended.

```
# Differential enrichment analysis based on ANOVA and Tukey
data_anova <- anova_tukey(data_imp, control = "Ctrl", type = "control")
```

Finally, significant proteins are defined by user-defined cutoffs.

```
# Denote significant proteins based on user defined cutoffs
data_sign <- cutoffs(data_lm, alpha = 0.05, lfc = log2(1.5))

# Number of significant proteins
results(data_sign) %>% filter(sign == "+") %>% nrow()
## [1] 257
```

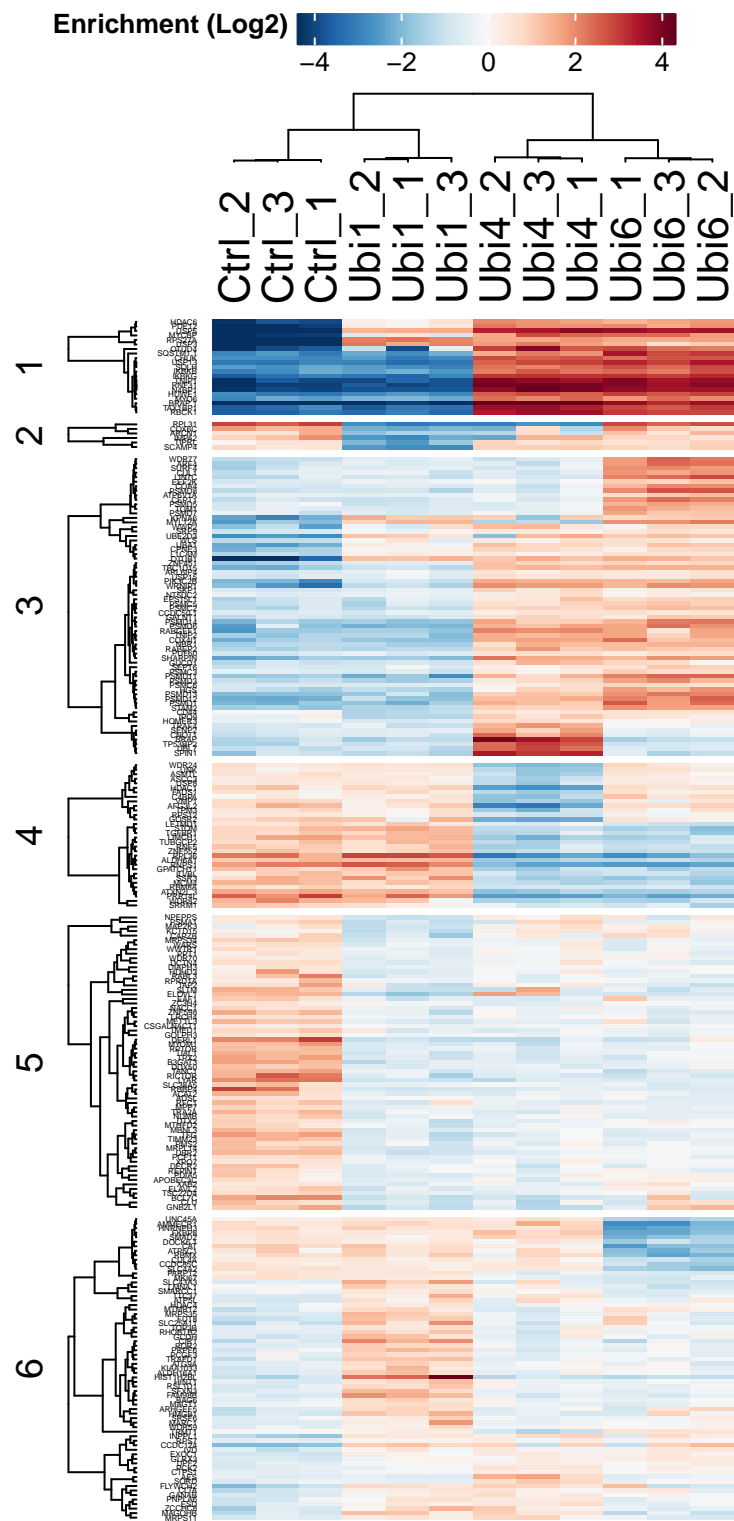
4.9 Visualization of the results

The data resulting from the previous analysis can easily be visualized by a number of functions. These visualizations assist in the determination of the optimal cutoffs to be used, highlight the most interesting samples and contrasts, and pinpoint differentially enriched/expressed proteins.

4.9.1 Heatmap of all significant proteins

The heatmap representation gives an overview of all significant proteins (rows) in all samples (columns). This allows to see general trends, e.g. one sample really different or one replicate which is off in all samples. Additionally, the clustering of samples (columns) can indicate closer related samples and clustering of proteins (rows) indicates similarly behaving proteins. The proteins are additionally clustered by k-means clustering and the number of clusters can be defined by k.

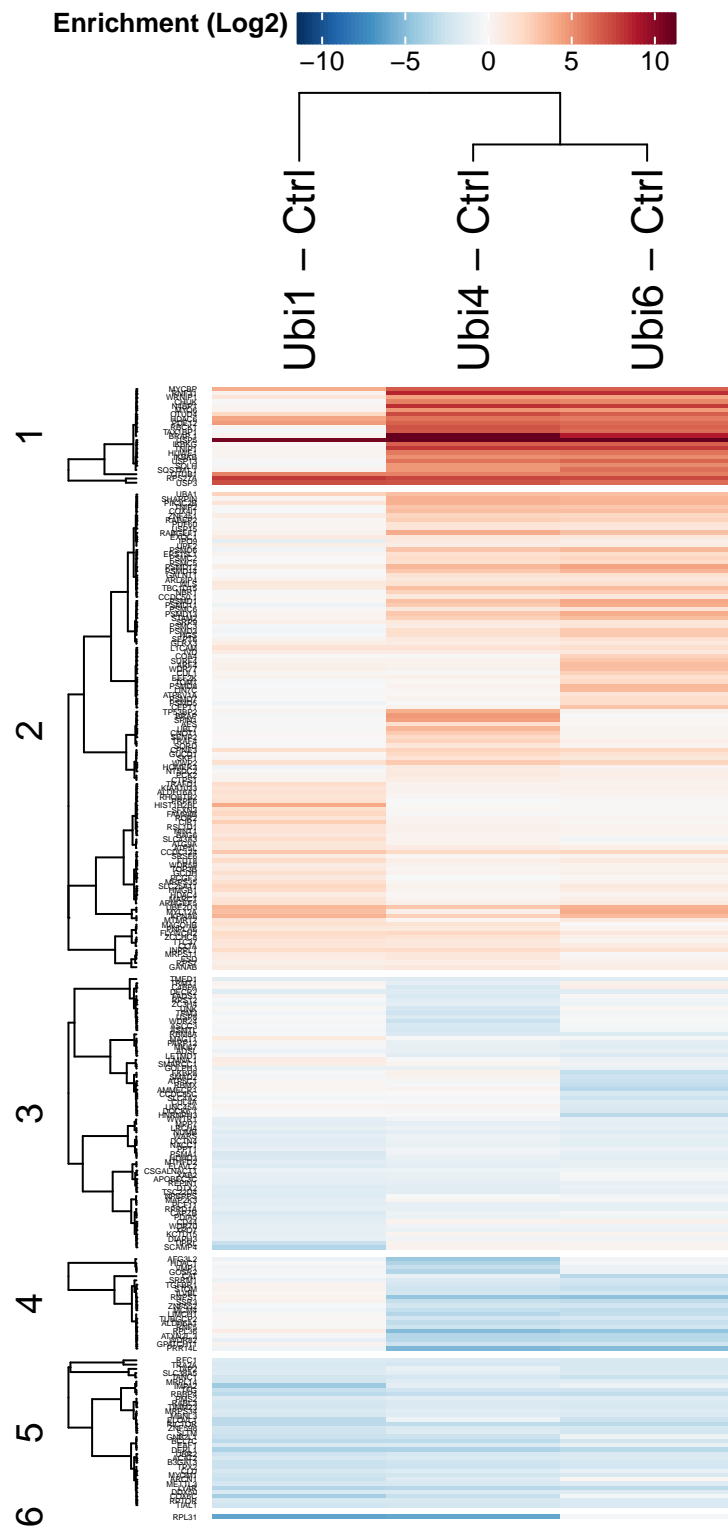
```
# Plot a heatmap of all significant proteins with the data centered per protein
plot_heatmap(data_sign, type = "centered", k = 6, col_limit = 4, labelsize = 3)
```



The heatmap shows a really nice clustering of the replicates and indicates that 4Ubi and 6Ubi enrich a similar repertoire of proteins. The k-means clustering of proteins (general clusters of rows) nicely separates protein classes with different binding behaviours.

Alternatively, a heatmap can be plotted with the direct sample comparisons (columns) for all significant proteins (rows). In this example this emphasises the enrichment of ubiquitin interactors over the control sample.

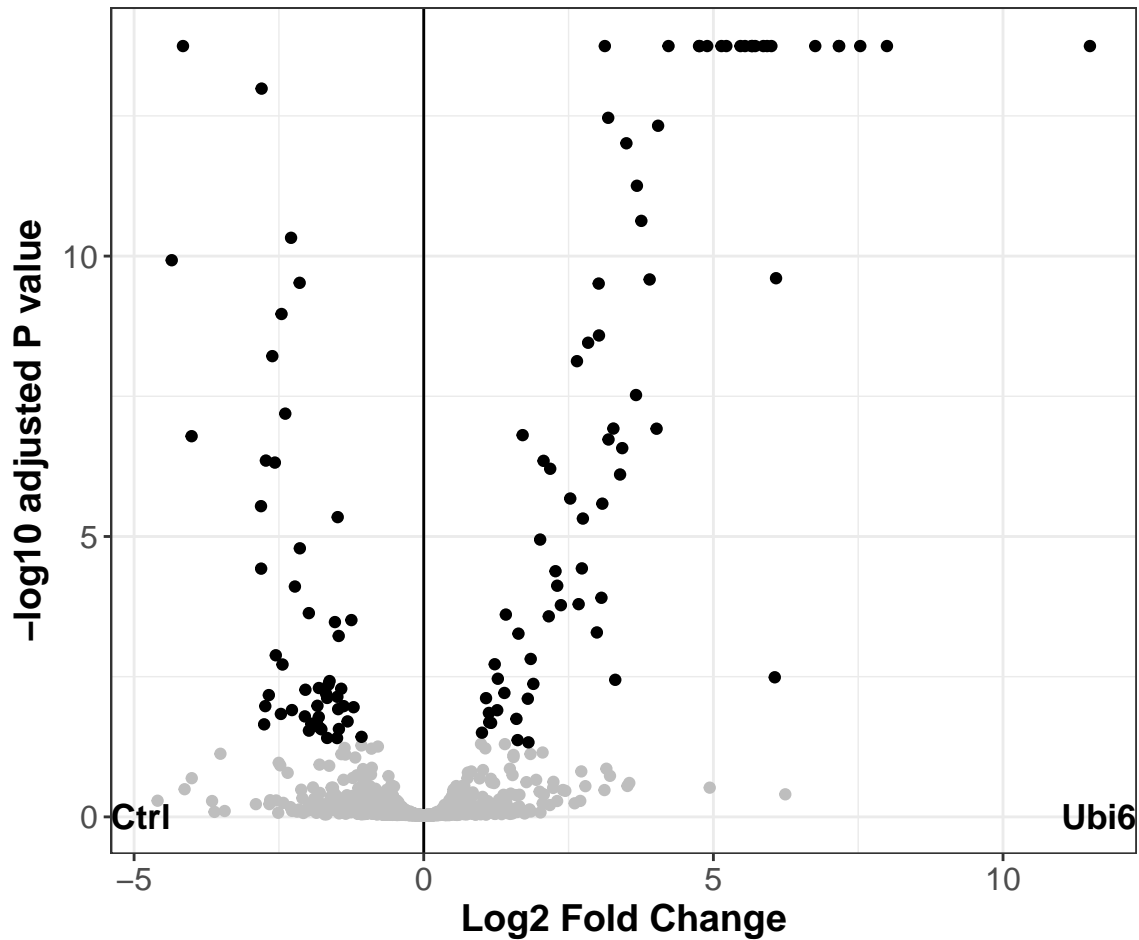
```
# Plot a heatmap of all significant proteins (rows) and the tested contrasts (columns)
plot_heatmap(data_sign, type = "contrast", k = 6, col_limit = 10, labels = 3)
```



Volcano plots can be used to visualize a specific contrast (comparison between two samples). This allows to inspect the enrichment of proteins between the two samples (x axis) and their corresponding adjusted p value (y axis).

```
plot_volcano(data_sign, contrast = "Ubi6_vs_Ctrl", labelsizes = 2)
```

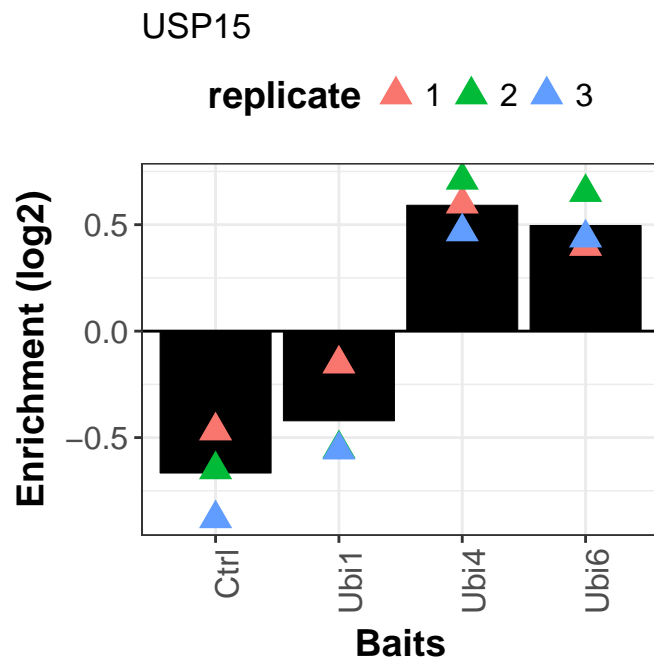




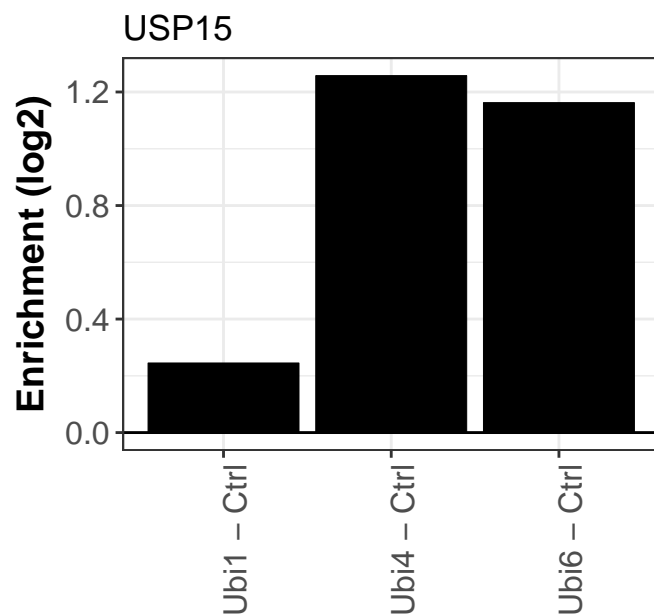
4.9.3 Barplots of a protein of interest

It can also be useful to plot the data of a single protein, for example if this protein is of special interest.

```
# Plot a barplot for the protein USP15 with the data centered  
plot_single(data_sign, protein = "USP15", type = "centered")
```



```
# Plot a barplot for the protein USP15
plot_single(data_sign, protein = "USP15", type = "contrast")
```



4.10 Results table

To extract a table containing the essential results, the *results* function can be used.

```
# Generate a results table
data_results <- results(data_sign)
```


The resulting table contains the following columns:

```
# Column names of the results table
colnames(data_results)
## [1] "name" "ID" "Ubi1_vs_Ctrl_p.adj"
## [4] "Ubi4_vs_Ctrl_p.adj" "Ubi6_vs_Ctrl_p.adj" "Ubi1_vs_Ctrl_sign"
## [7] "Ubi4_vs_Ctrl_sign" "Ubi6_vs_Ctrl_sign" "sign"
## [10] "Ubi1_vs_Ctrl_ratio" "Ubi4_vs_Ctrl_ratio" "Ubi6_vs_Ctrl_ratio"
## [13] "Ctrl_centered" "Ubi1_centered" "Ubi4_centered"
## [16] "Ubi6_centered"
```

Of these columns, the **p.adj** columns contain the adjusted p values for the contrast as depicted in the column name. The **ratio** columns contain the average log2 fold changes. The **sign** columns indicate whether the protein is differentially enriched/expressed, as defined by the chosen cutoffs. The **centered** columns contain the average log2 enrichment values scaled by protein-wise centering.

5 Stoichiometry analysis

5.1 Example dataset for stoichiometry analysis: PRC2 interactors

We analyze a proteomics dataset in which PRC2 interactors were characterized in Embryonic Stem Cells ([Kloet et al. Nature Structural & Molecular Biology 2016](#)). Again, the raw mass spectrometry data were analyzed using MaxQuant ([Cox and Mann, Nat Biotech 2007](#)).

5.2 Loading and preparation of the data, and generation of a SummarizedExperiment object

First, the data needs to be prepared for differential analysis.

```
# The data is provided with the package
data <- GFPIP
expdesign <- GFPIP_ExpDesign

# We filter for contaminant proteins and decoy database hits and generate unique names
data %<>% filter(Reverse != "+", Contaminant != "+")
data_unique <- unique_names(data, "Gene.names", "Protein.IDs")
## Joining, by = c("Protein.IDs", "Gene.names")

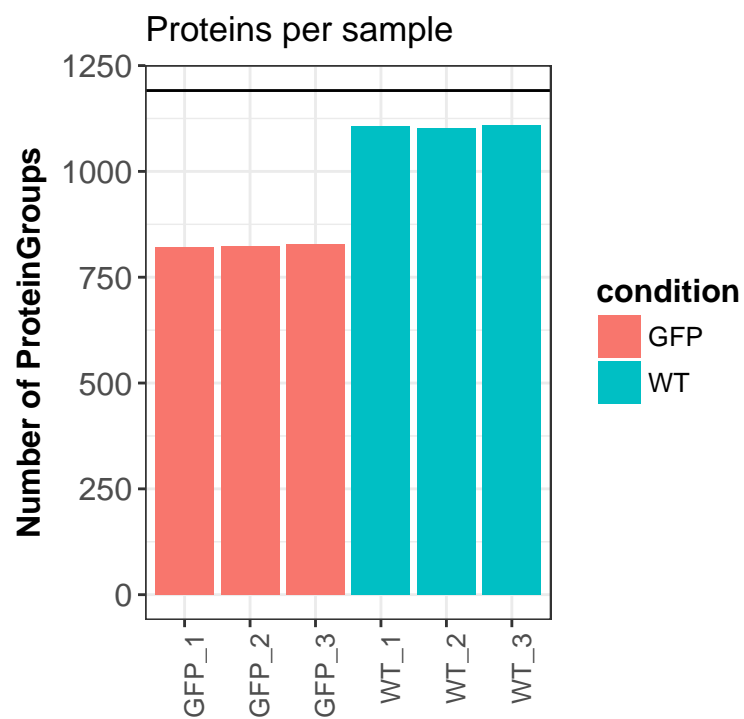
# The SummarizedExperiment object is generated
cols <- grep("LFQ.", colnames(data_unique))
data_se <- make_se(data_unique, cols, expdesign)
```

5.3 Filter on missing values, normalization and imputation

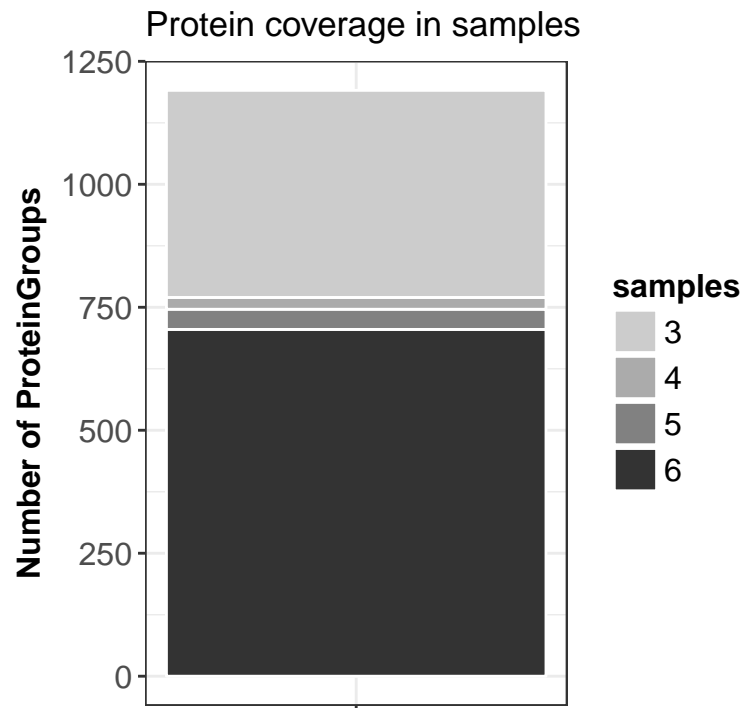
As described in “[differential analysis](#)”, the dataset is filtered on missing values, variance stabilized and remaining missing values are imputed.

```
# Filter for proteins that are identified in all replicates of at least one condition
data_filt <- filter_missval(data_se, thr = 0)
```

```
# Plot a barplot of the number of identified proteins per samples  
plot_numbers(data_filt)
```

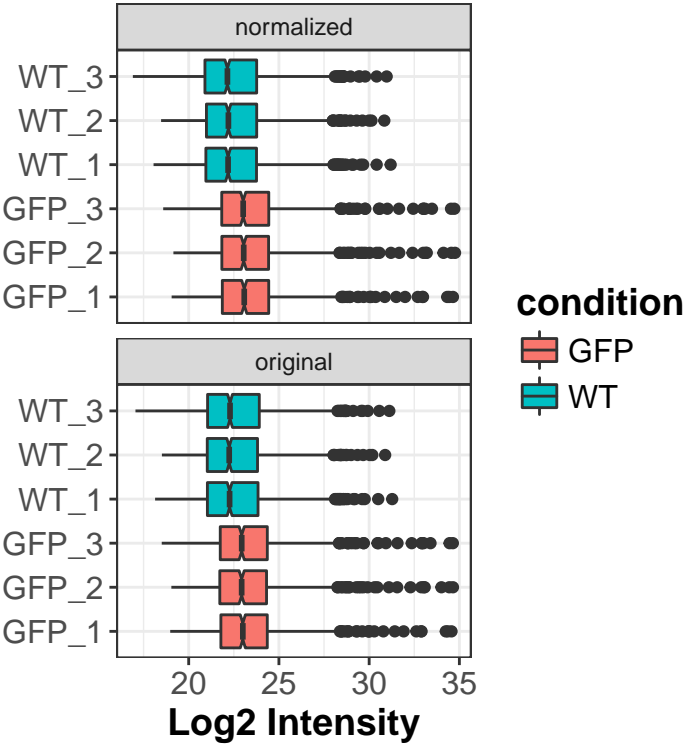


```
# Plot a barplot of the protein identification overlap between samples  
plot_coverage(data_filt)
```

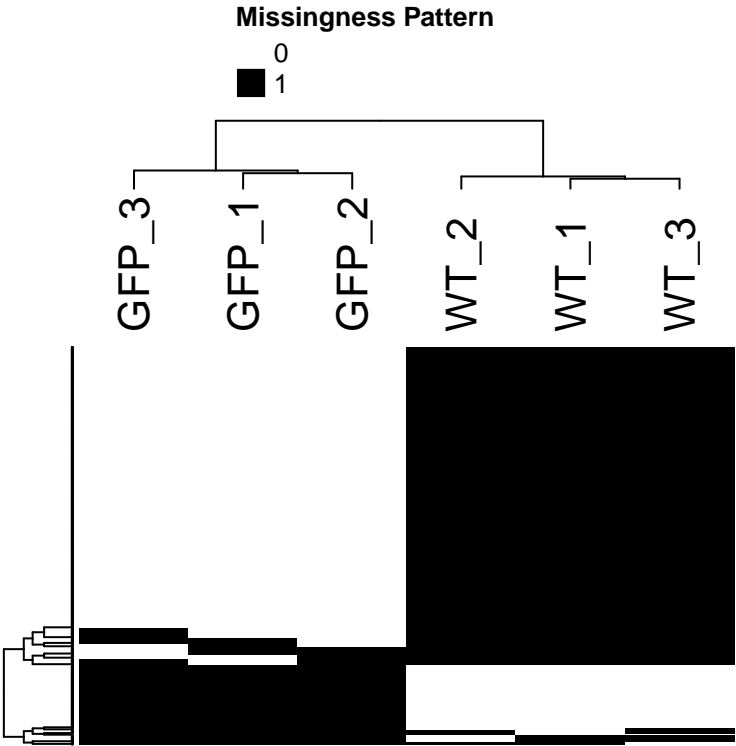


```
# Normalize
data_norm <- norm_vsn(data_filt)
## vsn2: 1191 x 6 matrix (1 stratum). Please use 'meanSdPlot' to verify the fit.

# Visualize normalization
plot_norm(data_filt, data_norm)
```



```
# Plot missingness pattern  
plot_missval(data_filt)
```



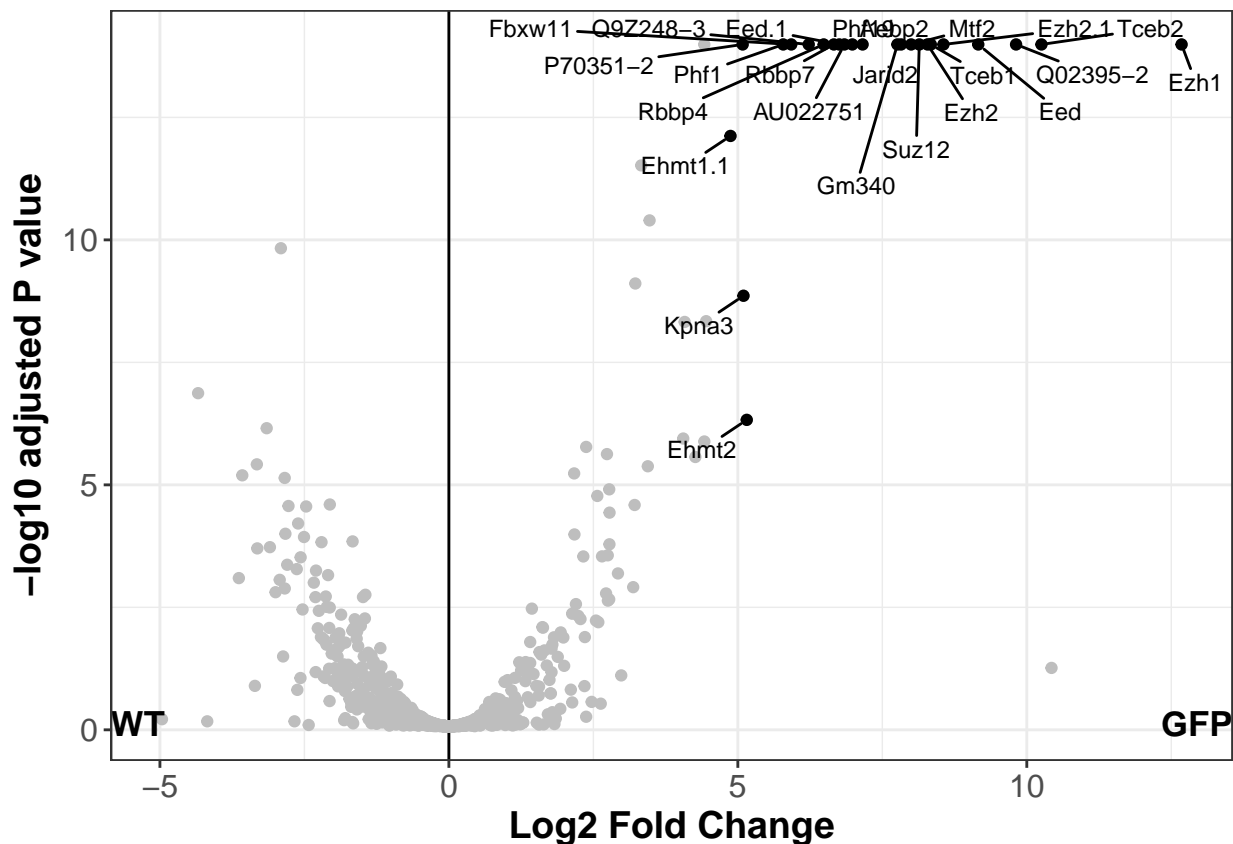
```
# Impute left-censored values, as the missingness pattern is not at random
data_imp <- imputation(data_norm, fun = "MinProb", q = 0.01)
## [1] 0.2707473
```

5.4 Differential enrichment analysis

```
# Differential enrichment analysis
data_lm <- linear_model(data_imp, control = "WT", type = "control")
## Tested contrasts:
## [1] "GFP_vs_WT"

# Call significant proteins based on user defined cutoffs
data_sign <- cutoffs(data_lm, alpha = 0.05, lfc = 4.5)

# Visualize the results
plot_volcano(data_sign, contrast = "GFP_vs_WT")
```



5.5 Stoichiometry analysis

```
# Load peptides data (available in the package)
peptides <- GFPip_pep
```

```

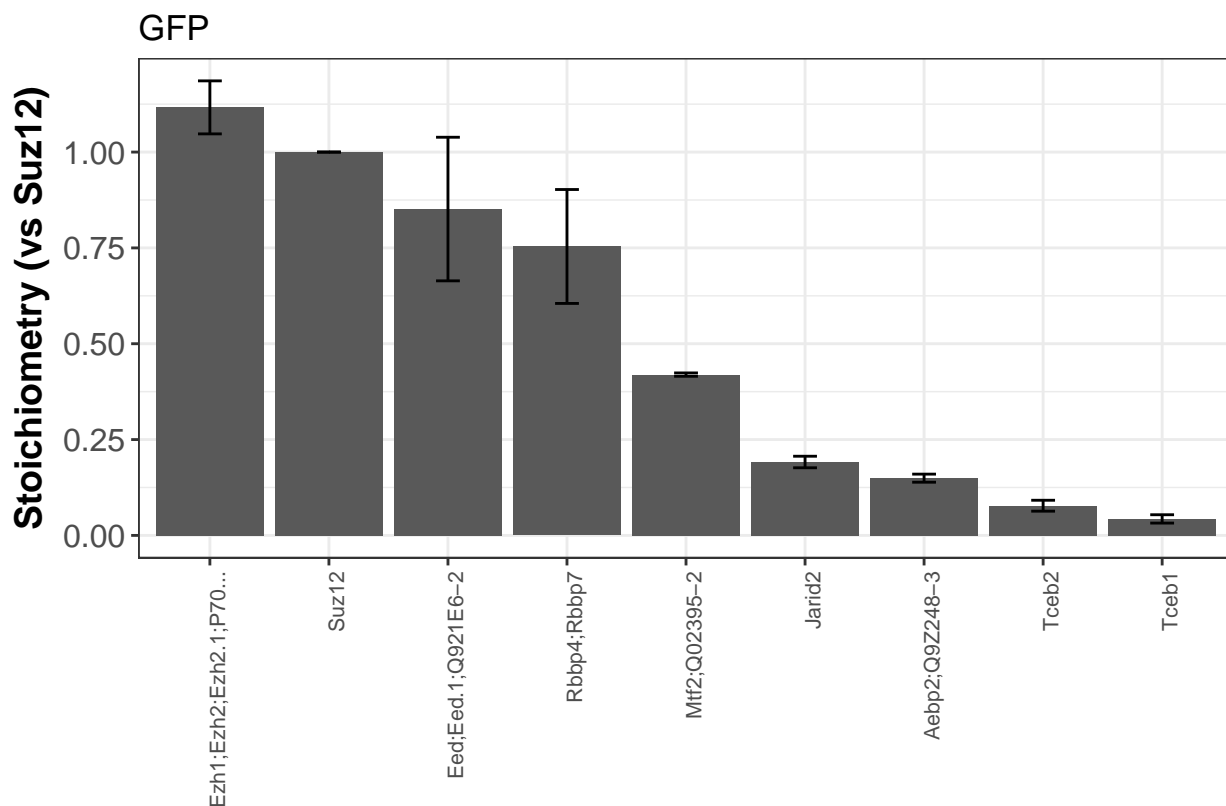
# Merge iBAQ values for the proteins with shared peptides
ibaq <- merge_ibaq(data_unique, peptides)

# Calculate stoichiometry
stoi <- stoichiometry(data_sign, ibaq, "GFP_vs_WT", "Suz12")

# Check the stoichiometry values for the ten most abundant proteins
head(stoi)
## # A tibble: 6 × 4
##           name condition stoichiometry      sd
##           <chr>      <chr>          <dbl>    <dbl>
## 1 Ezh1;Ezh2;Ezh2.1;P70351-2;Q61188-2    GFP    1.1165892 0.069132397
## 2           Suz12          GFP    1.0000000 0.000000000
## 3      Eed;Eed.1;Q921E6-2          GFP    0.8513677 0.187323358
## 4      Rbbp4;Rbbp7          GFP    0.7537290 0.148576018
## 5      Mtf2;Q02395-2          GFP    0.4194902 0.004399685
## 6      Jarid2          GFP    0.1914596 0.015055419

# Plot stoichiometry
plot_stoi(stoi, thr = 0.01)

```



6 Wrapper functions for the entire workflow

The package also entails wrapper functions that entail the complete analysis workflow and generate a report.

6.1 LFQ analysis

```
# The data is provided with the package
data <- UbiLength
experimental_design <- UbiLength_ExpDesign

# The wrapper function performs the full analysis
data_results <- LFQ(data, experimental_design, fun = "MinProb",
  control = "Ctrl", type = "control", alpha = 0.05, lfc = 1)
## Joining, by = c("Protein.IDs", "Gene.names")
## vsn2: 1899 x 12 matrix (1 stratum). Please use 'meanSdPlot' to verify the fit.
## [1] 0.2608395
## Tested contrasts:
## [1] "Ubi1_vs_Ctrl" "Ubi4_vs_Ctrl" "Ubi6_vs_Ctrl"
```

This wrapper produces a list of objects, which can be used to create a report and/or for further analysis. The reports function produces two reports (pdf and html) and a results table, which are saved in a generated "Report" folder.

```
# See all objects saved within the results object
names(data_results)
## [1] "data"      "se"        "filt"      "norm"      "imputed"   "lm"        "sign"
## [8] "results"   "param"

# Make a markdown report and save the results
report(data_results)
## Report and results.txt file saved in: '/EMBL/Packages/proteomeR/vignettes/Report'
```

The results generated by the LFQ function contain, among others, the *results* object (data.frame object) and the *sign* object (SummarizedExperiment object). The *results* object contains the essential results and the *sign* object contains the full SummarizedExperiment object which can be used for further visualization.

```
# Colnames of the results object
colnames(data_results$results)
## [1] "name"          "ID"            "Ubi1_vs_Ctrl_p.adj"
## [4] "Ubi4_vs_Ctrl_p.adj" "Ubi6_vs_Ctrl_p.adj" "Ubi1_vs_Ctrl_sign"
## [7] "Ubi4_vs_Ctrl_sign" "Ubi6_vs_Ctrl_sign" "sign"
## [10] "Ubi1_vs_Ctrl_ratio" "Ubi4_vs_Ctrl_ratio" "Ubi6_vs_Ctrl_ratio"
## [13] "Ctrl_centered"    "Ubi1_centered"    "Ubi4_centered"
## [16] "Ubi6_centered"

# the sign object
data_results$sign
## class: SummarizedExperiment
## dim: 1899 12
## metadata(0):
## assays(1): ''
## rownames(1899): AAAS ABCB7 ... ZSCAN26 ZYX
## rowData names(35): name Protein.IDs ... Ubi6_vs_Ctrl_sign sign
## colnames(12): Ubi4_1 Ubi4_2 ... Ubi1_2 Ubi1_3
## colData names(4): label ID condition replicate
```

The results table can be explored by selecting the `$results` object

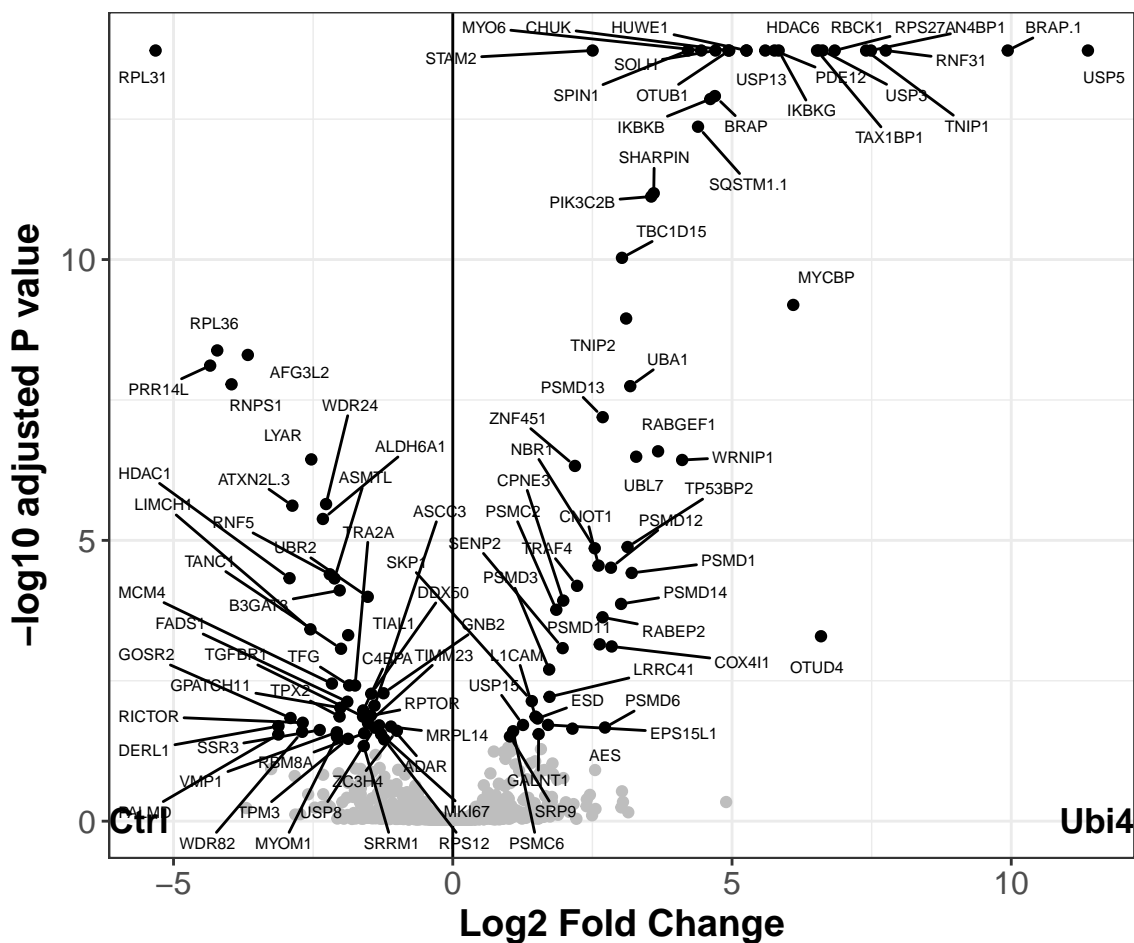
```
# Extract the results table
results_table <- data_results$results
```

```
# Number of significant proteins
results_table %>% filter(sign == "+") %>% nrow()
## [1] 236
```

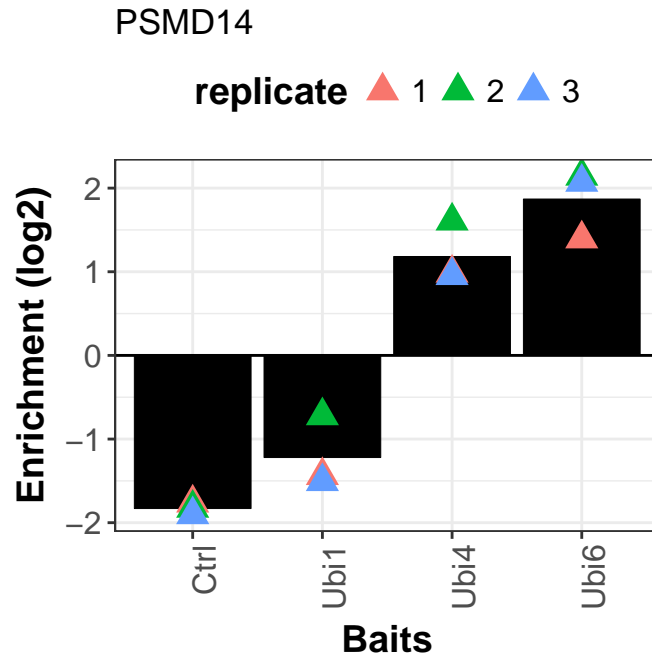
The full data (`sign` object) can be used for the plotting functions as described in the chapter “Visualization of the results”.

```
# Extract the sign object
full_data <- data_results$sign

# Use the full data to generate a volcano plot
plot_volcano(full_data, contrast = "Ubi4_vs_Ctrl", labels = 2)
```



```
# Use the full data to generate a barplot
plot_single(full_data, protein = "PSMD14", type = "centered")
```

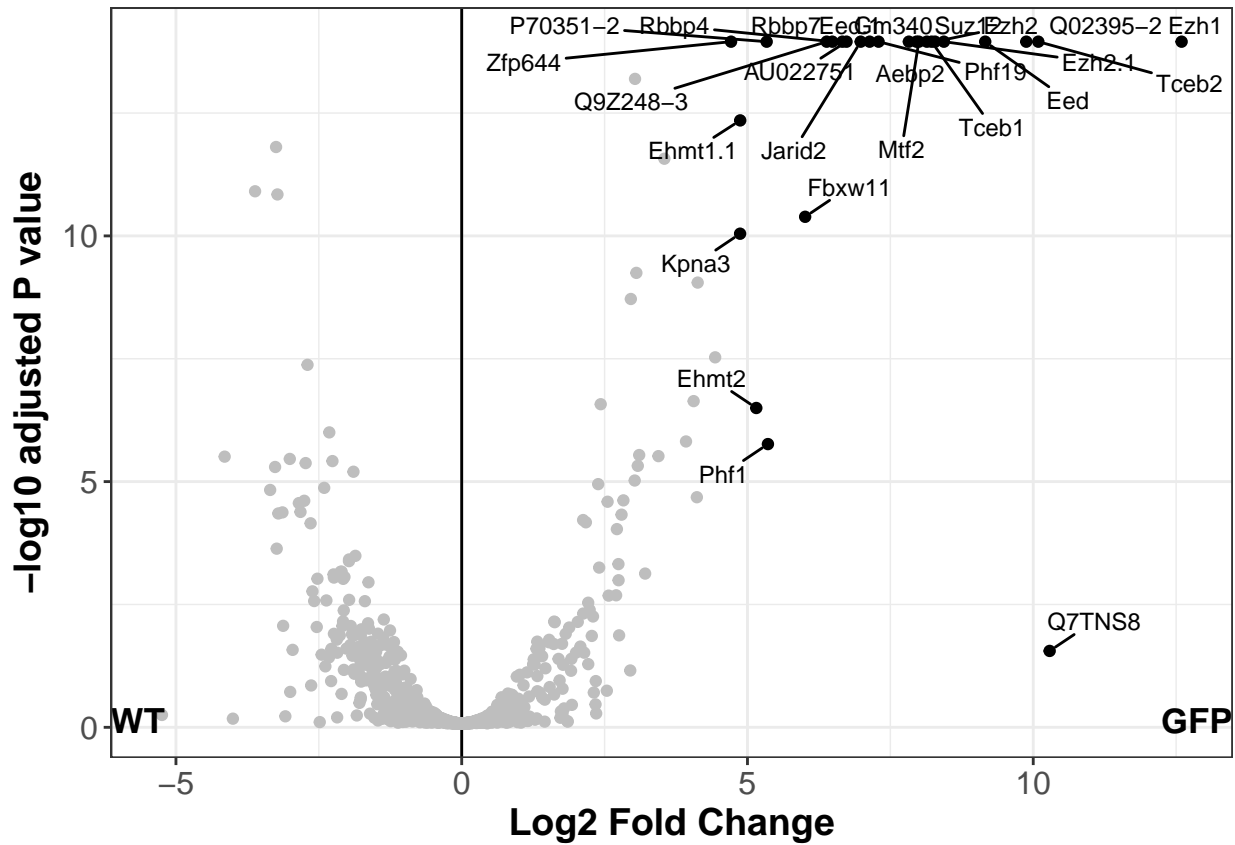



6.2 iBAQ-based stoichiometry analysis

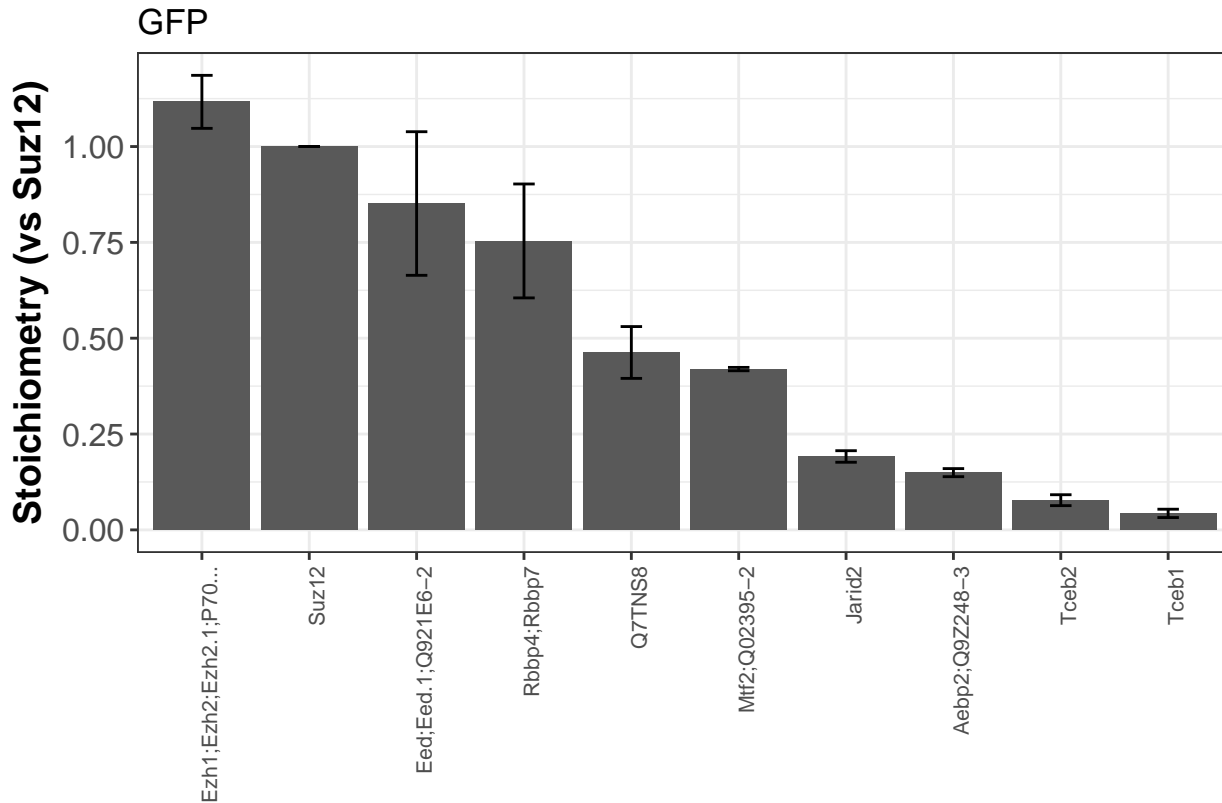
```
# The data is provided with the package
data <- GFPip
expdesign <- GFPip_ExpDesign

# The LFQ wrapper function performs the full differential analysis
data_results <- LFQ(data, expdesign, fun = "MinProb",
  control = "WT", type = "control",
  filter = c("Reverse", "Contaminant"), alpha = 0.05, lfc = 4.5)
## Joining, by = c("Protein.IDs", "Gene.names")
## vsn2: 1191 x 6 matrix (1 stratum). Please use 'meanSdPlot' to verify the fit.
## [1] 0.2707473
## Tested contrasts:
## [1] "GFP_vs_WT"

# Plot volcano from LFQ results
plot_volcano(data_results$sign, "GFP_vs_WT")
```



```
# The iBAQ wrapper function performs the stoichiometry analysis
stoi <- iBAQ(data_results, peptides, "GFP_vs_WT", "Suz12")
```



```
head(stoi)
## # A tibble: 6 × 4
##               name condition stoichiometry      sd
##               <chr>      <chr>          <dbl>    <dbl>
## 1 Ezh1;Ezh2;Ezh2.1;P70351-2;Q61188-2      GFP      1.1165892 0.069132397
## 2                               Suz12      GFP      1.0000000 0.000000000
## 3                               Eed;Eed.1;Q921E6-2      GFP      0.8513677 0.187323358
## 4                               Rbbp4;Rbbp7      GFP      0.7537290 0.148576018
## 5                               Q7TNS8      GFP      0.4627623 0.067585344
## 6                               Mtf2;Q02395-2      GFP      0.4194902 0.004399685
```

6.3 TMT analysis

```
# Need example data
TMTdata <- example_data
Exp_Design <- example_Exp_Design

# The wrapper function performs the full analysis
TMTdata_results <- TMT(data, expdesign = Exp_Design, fun = "QRILC",
  control = "Con_", type = "control", alpha = 0.05, lfc = 1)
```