proBatch

Jelena Čuklina, Chloe H. Lee, Patrick Pedrioli and Ruedi Aebersold
Institute of Molecular Systems Biology, Department of Biology, ETH Zurich, Switzerland

2021-05-19

Abstract

This vignette describes how to apply different functions from the `proBatch` package to diagnose and correct for batch effects. Most of the functions are applicable any 'omic' data, however, the package has a number of functions, designed specifically for mass spectrometry-based proteomics, and has been tested on SWATH data.

The `proBatch` package provides a complete functionality for batch correction workflow: to prepare the data for analysis, diagnose and correct batch effects and finally, to evaluate the correction with quality control metrics.

The `proBatch` package was programmed and intended for use by researchers without extensive programming skills, but with basic R knowledge.

Contents

1 Introduction **2**
1.1 Batch effects analysis in large-scale data . . . 2

2 Preparation for the analysis **3**
2.1 Installing dependencies and `proBatch` . . . 3

3 Installation **3**
3.1 Preparing the data for analysis . . . 5
3.1.1 Loading the libraries . . . 5
3.1.2 Input data formats . . . 5
3.1.3 Example dataset . . . 6
3.1.4 Preparing sample and peptide annotations . . . 7
3.1.5 Other utility functions . . . 8

4 Step-by-step workflow **9**
4.1 Initial assessment of the raw data matrix . . . 9
4.1.1 Plotting the sample mean . . . 9
4.1.2 Plotting boxplots . . . 10
4.2 Normalization . . . 11
4.2.1 Median normalization . . . 11
4.2.2 Quantile normalization . . . 11
4.3 Diagnostics of batch effects in normalized data . . . 12
4.3.1 Hierarchical clustering . . . 12
4.3.2 Principal component analysis (PCA) . . . 14
4.3.3 Principal variance component analysis (PVCA) . . . 16
4.3.4 Peptide-level diagnostics and spike-ins . . . 17
4.4 Correction of batch effects . . . 18
4.4.1 Continuous drift correction . . . 18
4.4.2 Discrete batch correction: combat or peptide-level median centering . . . 19

4.4.3 Correct batch effects: universal function 20
4.5 Quality control on batch-corrected data matrix 21
4.5.1 Heatmap of selected replicate samples 21
4.5.2 Correlation distribution of samples 24
4.5.3 Correlation of peptide distributions within and between proteins 25

5 SessionInfo **26**

6 Citation **27**

7 References **28**

1 Introduction

1.1 Batch effects analysis in large-scale data

Recent advances in mass-spectrometry enabled fast and near-exhaustive identification and quantification of proteins in complex biological samples [1], allowing for the profiling of large-scale datasets. Obtaining a sufficiently large dataset is, however, associated with considerable logistics efforts. Often multiple handlers at the sample preparation and data acquisition steps are involved e.g. protein extraction, peptide digestion, instrument cleaning. This introduces systematic technical variations known as batch effects.

Batch effects can alter or obscure the biological signal in the data [2, 3]. Thus, the presence and severity of batch effects should be assessed, and, if necessary, corrected.

The fundamental objective of the batch effects adjustment procedure is to make all measurements of samples comparable for a meaningful biological analysis. Normalization brings the measurements into the same scale. Bias in the data, however, can persist even after normalization, as batch effects might affect specific features (peptides, genes) thus requiring additional batch correction procedures. This means, that the correction of technical bias has often two steps: normalization and batch effects correction.

The improvement of the data is best assessed visually at each step of the correction procedure. The initial assessment sets the baseline, before any correction is executed. After normalization, batch effects diagnostics allow to determine the severity of the remaining bias. Finally, the quality control step allows to determine whether the correction improved the quality of the data.

The pipeline, summarizing this workflow, is shown in Fig.1.

##Analysis of large-scale data: steps before and after batch correction

We recommend users to follow this batch correction workflow to ensure all measurements are comparable for downstream analysis. We provide step-by-step illustrations to implement this workflow in the next sections.

Before starting the description, we give a few hints about the steps preceding and following batch effects analysis and correction.

- It is assumed that the initial data processing is completed. In mass spectrometry- based proteomics, this involves primarily peptide-spectrum matching [4, 5] and FDR control [6].
- Data filtering is commonly the next step of data processing. In the context of batch effects correction, both peptide and sample filtering need to be approached with caution. First of all, decoy measurements should be filtered out to ensure correct sample intensity distribution alignment. However, non-proteotypic peptides should be retained. Filtering out low-quality samples,also substantially alters normalization and batch effects correction. The 'bad' samples, usually identified by the total intensity of identified peptides or correlation of samples, can be removed either before or after the correction for technical bias. Which option is best for a given dataset, should be decided in each case individually.
- We strongly advocate not to impute missing values before correction and to exclude 'requant' values, inferred from SWATH data. Two common strategies to impute values use either 'average' measurements,

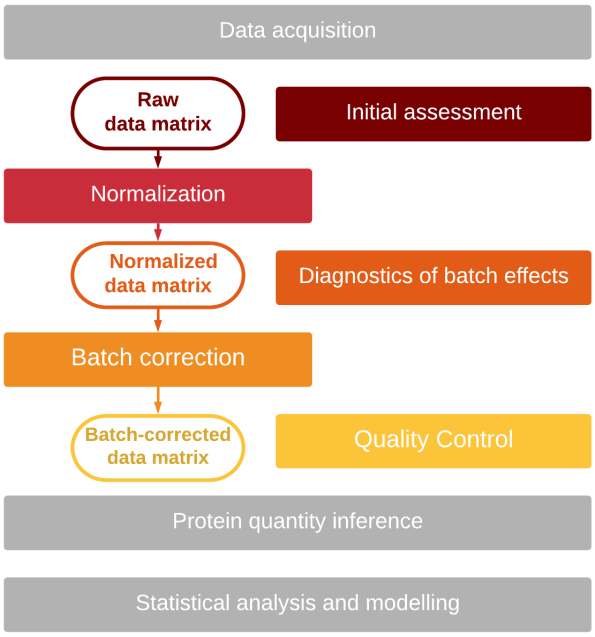


Figure 1: proBatch in batch correction workflow

or random noise-level measurements. Both strategies bias the mean/median estimate of the peptide and are detrimental to both normalization and batch effects correction.

- We suggest to perform protein quantification after batch effects correction, as the correction procedure alters the abundances of peptides and peptide transitions, and these abundances are critical for protein quantity inference. Instead, we do recommend to correct the technical noise at the level, which is used to infer the proteins (thus, fragment-level for inference tools such as aLFQ or MSstats).

2 Preparation for the analysis

2.1 Installing dependencies and `proBatch`

`proBatch` is primarily a wrapper of functions from other packages, therefore it has numerous dependencies. If some of these dependencies are not installed, you will need to do that before running `proBatch`.

```
bioc_deps <- c("GO.db", "impute", "preprocessCore", "pvca","sva" )
cran_deps <- c("corrplot", "data.table", "ggplot2", "ggfortify","lazyeval",
               "lubridate", "pheatmap", "reshape2","readr", "rlang",
               "tibble", "dplyr", "tidyr", "wesanderson","WGCNA")

if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install(bioc_deps)
install.packages(cran_deps)
```

3 Installation

To install the latest version of proBatch package, you need `devtools`:

```r
#install the development version from GitHub:
install.packages('devtools')
devtools::install_github('symbioticMe/proBatch', build_vignettes = TRUE)
```

3.1 Preparing the data for analysis

3.1.1 Loading the libraries

In this vignette, we use functions from `dplyr`, `tibble` , `ggplot2` and other `tidyverse` package family to transform some data frames

```
require(dplyr)
require(tibble)
require(ggplot2)
```

3.1.2 Input data formats

To analyze an experiment for batch effects, three tables need to be loaded into environment:

The package typically requires three datasets: 1) measurement (data matrix), 2) sample annotation, and 3) feature annotation (optional) tables. If you are familiar with the `Biobase` package, these correspond to 1) `assayData`, 2) joined `phenoData` and `protocolData`, and 3)`featureData`.

1) Measurement table

Either a wide data matrix or long format data frame. In the wide (matrix) format, referred in this vignette as `data_matrix`, rows represent features (for proteomics, peptides/fragments) and columns represent samples. In the long format, referred in this vignette as `df_long`, each row is a measurement of a specific feature (peptide, fragment) in the specific sample. At least three columns are required: feature ID column, measurement (intensity) column and sample ID column. For batch correction, we also assume, that the imputed values, e.g. requant values from OpenSWATH output, are flagged in quality column, such as m_score, so that they can be filtered out (see below).

In this vignette, the essential columns have the following names:

```
feature_id_col = 'peptide_group_label'
measure_col = 'Intensity'
sample_id_col = 'FullRunName'
essential_columns = c(feature_id_col, measure_col, sample_id_col)
```

The names of the columns can be technology-specific. These column names are specific to the OpenSWATH tsv output format.

In the package, we provide the functionality to convert from long to matrix format (see section 2.2.4.1 'Utility functions').

Note that the sample IDs (column names in `data_matrix`) should match the values of the sample ID column in `sample_annotation` and the feature ID column values should match the feature annotation table (here - `peptide_annotation`). For OpenSWATH tsv file, which is a long format data frame, `peptide_annotation` can be generated in the beginning of the analysis.

2) Sample annotation

A data frame, where one row corresponds to one sample (run/file), and the columns contain information on biological and technical factors. Minimally, sample annotation has to contain a sample ID column, at least one technical and one biological factor column, and a biological ID column (unique ID for the biological replicate, which is repeated for each technical replicate).

In our example data, these columns are:

1. `sample_id_col = 'FullRunName'`
2. technical covariates:
    - `digestion_batch` - date when samples were prepared
    - `RunDate` (and `RunTime`, if available) - will be used to determine run order;
    - `MS_batch` - number of MS batches (in this case, sets of runs between machine cleaning)

3. biological covariates:
    - `Strain`
    - `Diet`
    - `Sex`
4. `biospecimen_id_col = 'EarTag'`

For the analysis, we also need `order_col`. This column is inferred from `RunDate` and `RunTime`, creating a joined `DateTime` column. Although both `order` (the default name of `order_col`) and `DateTime` columns can be created with the package function `date_to_sample_order` (see Utility functions below), here they are provided already in the examples to allow user to skip this function, if `order` is defined already or not relevant/unknown for the specific dataset.

Thus, technical and biological factors are:

```
technical_factors = c('MS_batch', 'digestion_batch')
biological_factors = c('Strain', 'Diet', 'Sex')
biospecimen_id_col = 'EarTag'
```

For illustration purposes, we will focus on one technical factor:

```
batch_col = 'MS_batch'
```

3) Feature (peptide) annotation

A dataframe, where one row corresponds to one feature (in MS proteomics - peptide or fragment), and the columns are names of proteins and corresponding genes. Thus, the minimum columns are feature ID (`peptide_group_id`) and name of corresponding protein (in this vignette, we use `Gene` name).

3.1.3 Example dataset

The `proBatch` package can be applied to any dataset, for which an intensity matrix and a sample annotation tables are available. However, the package was primarily designed with proteomic data in mind, and thoroughly tested on SWATH data. Thus, as and example dataset we include a reduced SWATH-MS measurement file, generated from a BXD mouse aging study. In this study, the liver proteome of mouse from BXD reference population have been profiled to identify proteome changes associated with aging. The animals of each strain were subjected to Chow and High-Fat Diet and sacrificed at different ages (the age factor is excluded from the example data as age-related differences are the focus of an unpublished manuscript).

This dataset has a few features, that make it a good illustrative example: 1. This is a large dataset of 371 samples, that was affected by multiple technical factors, described above in the `sample_annotation` subsection. Specifically, 7 MS batches drive the similarity of the samples. 2. The technical factors bias the data in at least two ways: discrete shifts (affecting different peptides in a batch-specific way), and continuous shifts from MS drift associated with sample running order. We will illustrate, how such biases can be corrected. 3. Replicate structure: samples from two animals were injected in the MS instrument every 10-15 samples. Additionally, several samples were repeated back-to-back in the end and in the beginning of two consecutive batches. This replication scheme allows to evaluate the coefficient of variation and is highly beneficial for assessment of sample correlation.

The example SWATH data and annotation files can be loaded from the package with the function `data()`.

```
library(proBatch)
data('example_proteome', 'example_sample_annotation', 'example_peptide_annotation',
     package = 'proBatch')
```

3.1.4 Preparing sample and peptide annotations

`proBatch` provides utility functions to facilitate the preparation of sample and peptide annotation. Feel free to skip this section if you don't require them.

3.1.4.1 Defining the order of samples from running date and time In proteomics, sequential measurement of samples may introduce order-related effects. To facilitate the examination of such effects, it is necessary to define an order column in the sample annotation. Using the `date_to_sample_order()` function one can infer sample order from the date and time of the measurements.

You can specify the columns illustrating date and time with `time_column` and their formats with the `dateTimeFormat` parameters (see `POSIX` date format for reference).

```
generated_sample_annotation <- date_to_sample_order(example_sample_annotation,
                                          time_column = c('RunDate','RunTime'),
                                          new_time_column = 'generated_DateTime',
                                          dateTimeFormat = c('%b_%d', '%H:%M:%S'),
                                          new_order_col = 'generated_order',
                                          instrument_col = NULL)
library(knitr)
kable(generated_sample_annotation[1:5,] %>%
  select(c('RunDate', 'RunTime', 'order', 'generated_DateTime', 'generated_order')))
```

RunDate	RunTime	order	generated_DateTime	generated_order
Oct_05	18:35:00	1	2021-10-05 18:35:00	1
Oct_05	20:12:00	2	2021-10-05 20:12:00	2
Oct_05	21:50:00	3	2021-10-05 21:50:00	3
Oct_05	23:28:00	4	2021-10-05 23:28:00	4
Oct_06	01:51:00	5	2021-10-06 01:51:00	5

The new time and order columns have been generated. Note that the generated_order has the same order as the manually annotated order column.

3.1.4.2 Generating peptide annotation from OpenSWATH data From the OpenSWATH output, you can generate peptide annotation using `create_peptide_annotation()` by denoting the peptide ID with the `feature_id_col` and the annotation columns with the `annotation_col` parameters.

```
generated_peptide_annotation <- create_peptide_annotation(example_proteome,
                                        feature_id_col = 'peptide_group_label',
                                        protein_col = 'Protein')
```

In practice, the generation of peptide annotation from proteomic data allows one to remove peptide annotation columns from the intensity dataframe, thereby reducing the memory load, and can be achieved as follows:

```
example_proteome = example_proteome %>% select(one_of(essential_columns))
gc()
#>            used  (Mb) gc trigger  (Mb) limit (Mb) max used  (Mb)
#> Ncells 5536164 295.7    9579117 511.6         NA  7331556 391.6
#> Vcells 9681849  73.9   15026398 114.7      65536 12455019  95.1
```

Additionally, smaller peptide annotation matrices allow for faster mapping of UniProt identifiers to gene names and other IDs.

3.1.5 Other utility functions

3.1.5.1 Transforming the data to long or wide format Plotting functions accept data in either data matrix or long data frame formats. Our package provides the helper functions `long_to_matrix()` and `matrix_to_long()` to conveniently convert datasets back and forth.

```
example_matrix <- long_to_matrix(example_proteome,
                                 feature_id_col = 'peptide_group_label',
                                 measure_col = 'Intensity',
                                 sample_id_col = 'FullRunName')
```

3.1.5.2 Transforming the data to log scale Additionally, if the data are expected to be log-transformed, one can:

```
log_transformed_matrix <- log_transform_dm(example_matrix,
                                           log_base = 2, offset = 1)
```

3.1.5.3 Defining the color scheme To guarantee uniform color annotation, function `sample_annotation_to_colors()` can be used. Using this function biological and technical factor columns are mapped to qualitative colors (maximally distinct), while date and numeric columns are mapped to sequential (gradient) colors.

```
color_list <- sample_annotation_to_colors(example_sample_annotation,
                                          factor_columns = c('MS_batch', 'digestion_batch',
                                                             'EarTag', 'Strain',
                                                             'Diet', 'Sex'),
                                          numeric_columns = c('DateTime','order'))
```

4 Step-by-step workflow

4.1 Initial assessment of the raw data matrix

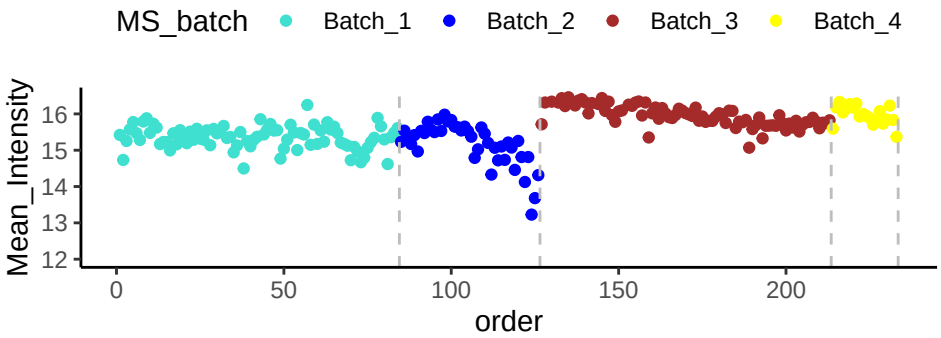
Before any correction, it is informative to set the baseline of the data quality by examining global quantitative patterns in the raw data matrix. Commonly, batch effects manifest as batch-specific intensity distribution changes.

In proteomics, batch-specific intensity drifts of sample mean can occur. Thus, it is important to carefully keep track of the order of sample measurement. Order inference can be performed as shown in the previous section 'Defining the order of samples from running date and time'. If the order column is not available (`order_col = NULL`), the samples order in the sample annotation is used for plotting.

4.1.1 Plotting the sample mean

The `plot_sample_mean()` function illustrates global average vs. sample running order. This can be helpful to visualize the global quantitative pattern and to identify discrepancies within or between batches.

```
plot_sample_mean(log_transformed_matrix, example_sample_annotation, order_col = 'order',
                 batch_col = batch_col, color_by_batch = TRUE, ylimits = c(12, 16.5),
                 color_scheme = color_list[[batch_col]])
```

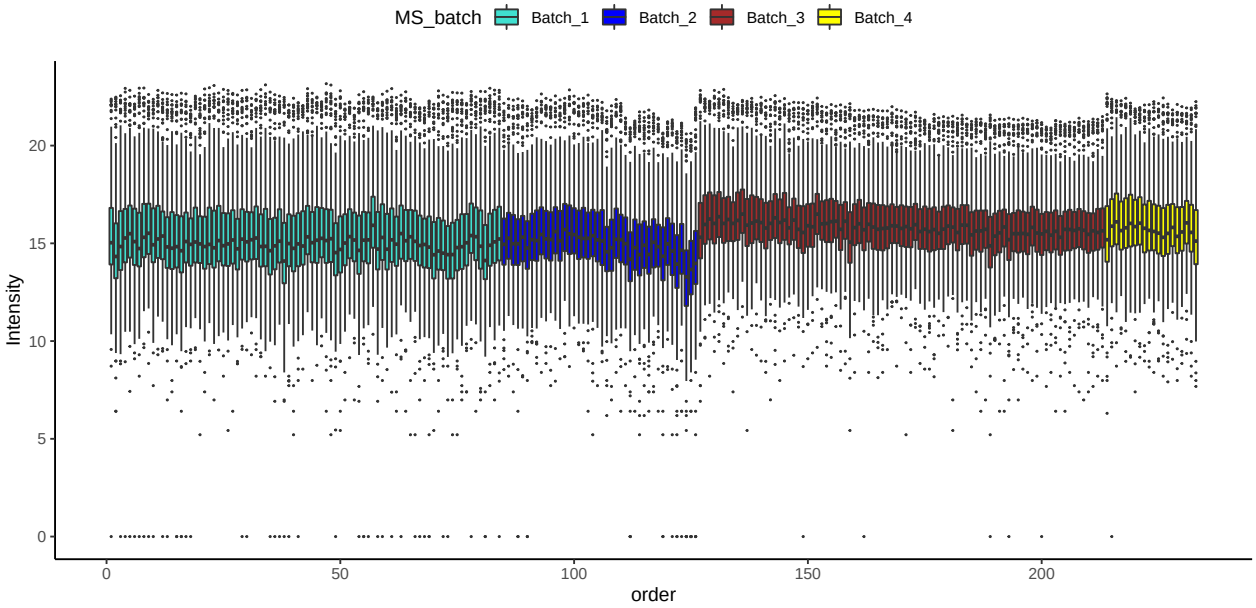


We can clearly see down-sloping trends in the BXD aging dataset. In fact, during the data acquisition, the mass-spectrometer had to be interrupted several times for tuning and/or column exchange as the signal was decreasing.

4.1.2 Plotting boxplots

Alternatively, `plot_boxplots()` captures the global distribution vs. the sample running order.

```
log_transformed_long <- matrix_to_long(log_transformed_matrix)
batch_col = 'MS_batch'
plot_boxplot(log_transformed_long, example_sample_annotation,
             batch_col = batch_col, color_scheme = color_list[[batch_col]])
```



In many cases, global quantitative properties such as sample medians or standard deviations won't match. The initial assessment via mean plots or boxplots can capture such information and hint at which normalization method is better suitable. If the distributions are comparable, methods as simple as global median centering can fix the signal shift, while quantile normalization can help in case of divergent distributions.

4.2 Normalization

In large-scale experiments, the total intensity of the samples is likely to be different due to a number of reasons, such as different amount of sample loaded or fluctuations in measurement instrument sensitivity. To make samples comparable, they need to be scaled. This processed is called normalization. In `proBatch`, two normalization approaches are used: median centering and quantile normalization. The normalization function `normalize_data()` by default takes log-transformed data, and if needed, log-transformation can be done on-the-fly by specifying `log_base = 2` for log2-transformation.

4.2.1 Median normalization

Median normalization is a conservative approach that shifts the intensity of the sample to the global median of the experiment. If the distributions of samples are dramatically different and this cannot be explained by non-technical factors, such as heterogeneity of samples, other approaches, such as quantile normalization need to be used.

```
median_normalized_matrix = normalize_data_dm(log_transformed_matrix,
                                             normalize_func = 'medianCentering')
```

Same result will be achieved with:

```
median_normalized_matrix = normalize_data_dm(example_matrix,
                                             normalize_func = 'medianCentering',
                                             log_base = 2, offset = 1)
```

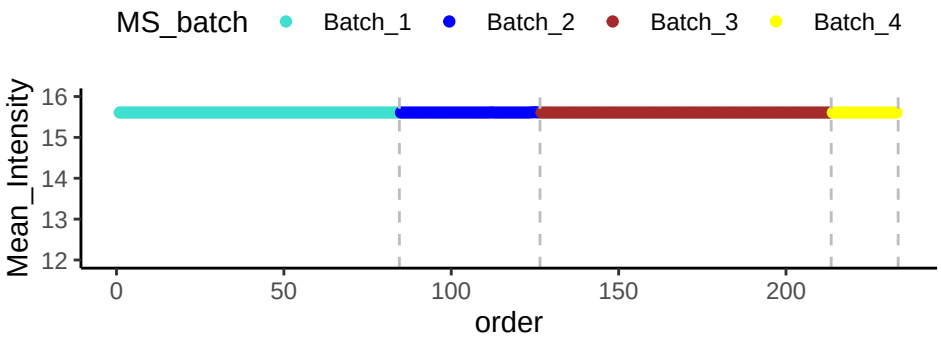
4.2.2 Quantile normalization

Quantile normalization sets different distributions of individual samples to the same quantiles, which forces the distribution of the raw signal intensities to be the same in all samples. This method is computationally effective and has simple assumption that the majority of features (genes, proteins) is constant among the samples, thus also the distribution in principle are identical.

```
quantile_normalized_matrix = normalize_data_dm(log_transformed_matrix,
                                               normalize_func = 'quantile')
```

After quantile or median normalization, you can easily check if the global pattern improved by generating mean or boxplots and comparing them side by side. Here are the mean plots before and after normalization of the log transformed dataset.

```
plot_sample_mean(quantile_normalized_matrix, example_sample_annotation,
                 color_by_batch = TRUE, ylimits = c(12, 16),
                 color_scheme = color_list[[batch_col]])
```



4.3 Diagnostics of batch effects in normalized data

Now is the time to diagnose for batch effects and evaluate to what extent technical variance still exists in the normalized data matrix. The positive effect of normalization is sometimes not sufficient to control for peptide and protein-specific biases associated with a certain batch source. These biases can be identified via diagnostic plots. Here, we describe our essential toolbox of batch effect diagnostic approaches. Note that sample annotation and/or peptide annotation are necessary for the implementation of these plots.

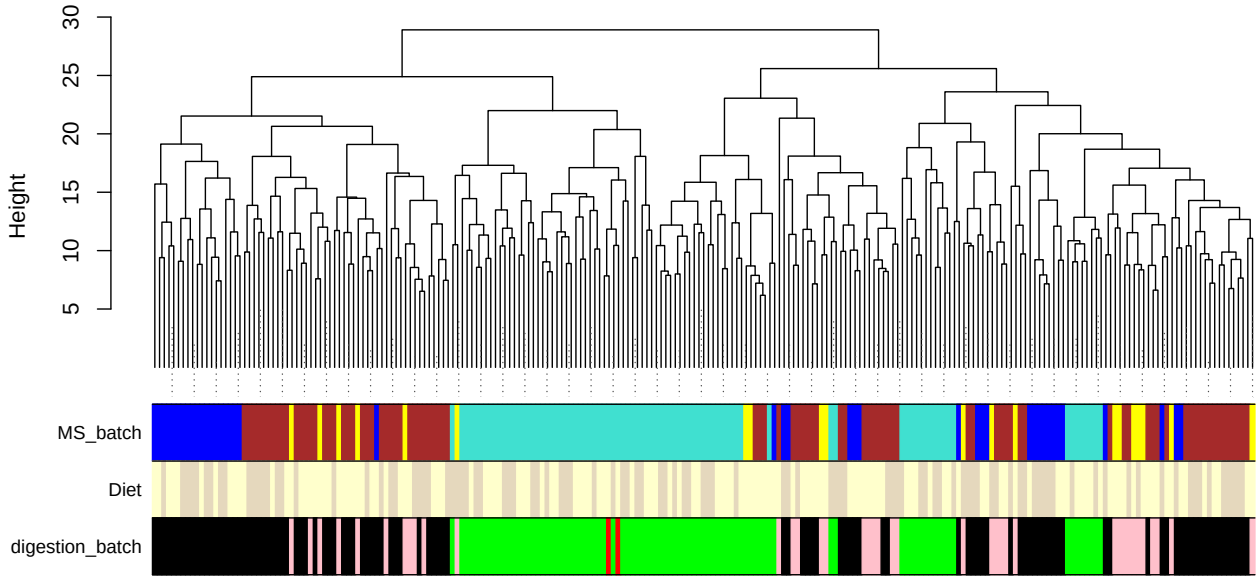
4.3.1 Hierarchical clustering

Hierarchical clustering is an algorithm that groups similar samples into a tree-like structure called dendrogram. Similar samples cluster together and the driving force of this similarity can be visualized by coloring the leaves of the dendrogram by technical and biological variables.

Our package provides `plot_sample_clustering()` and `plot_heatmap()` to plot the dendrogram by itself or with a heatmap. You can easily color annotations on the leaves of the dendrograms or heatmaps to identify what is the driving force of the clustering.

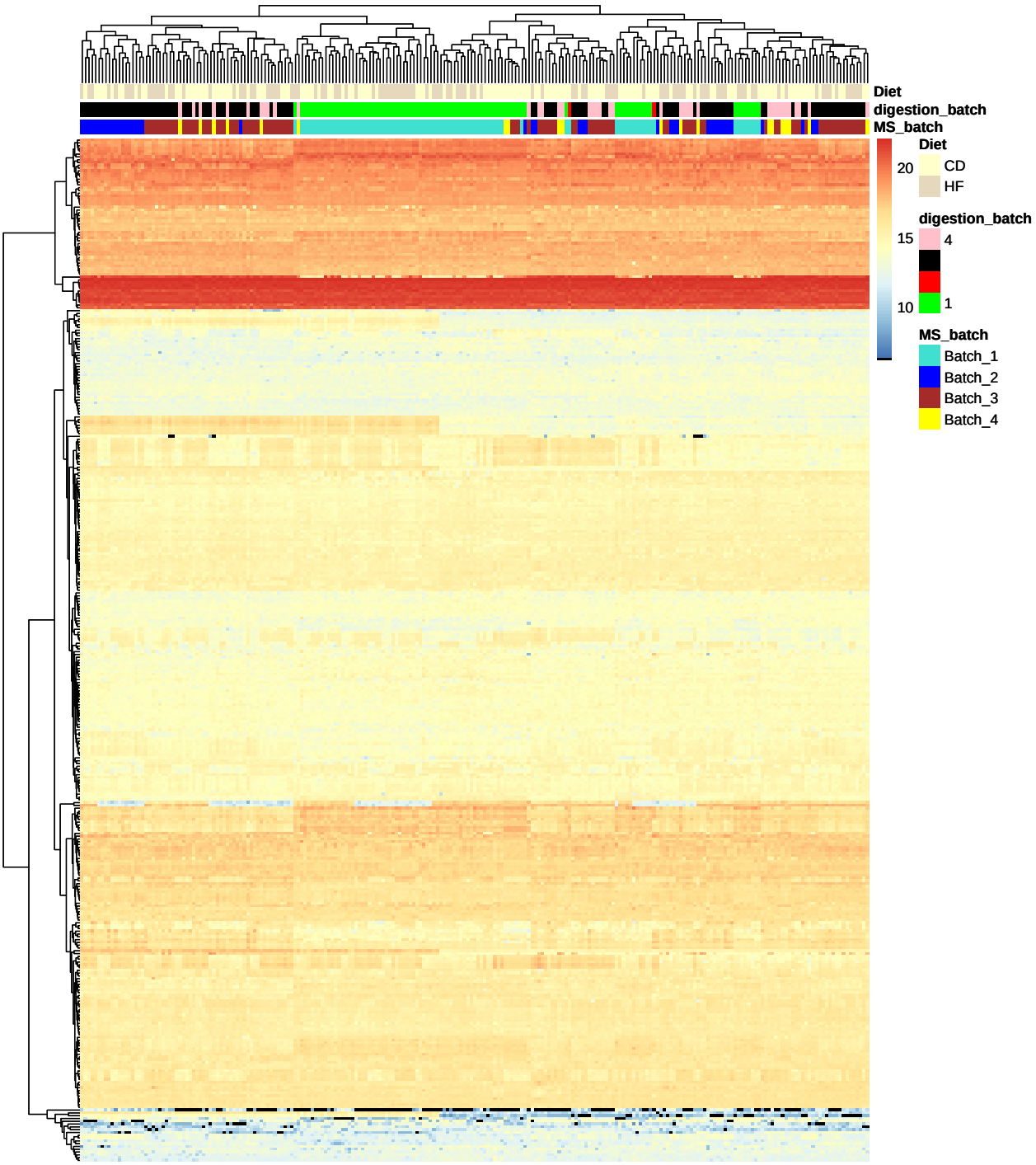
Once your color annotation is ready, for the specific covariates of interest, you can subset the color dataset and feed it into the clustering functions.

```
selected_annotations <- c('MS_batch',  'digestion_batch', 'Diet')

#Plot clustering between samples
plot_hierarchical_clustering(quantile_normalized_matrix,
                             sample_annotation = example_sample_annotation,
                             color_list = color_list,
                             factors_to_plot = selected_annotations,
                             distance = 'euclidean', agglomeration = 'complete',
                             label_samples = FALSE)
```



Similarly, you can plot a heatmap by supplementing the color list. You decide whether to show annotations in the column, row or both by specifying the required covariates with `sample_annotation_col`, `sample_annoation_row`, or both.

```
plot_heatmap_diagnostic(quantile_normalized_matrix, example_sample_annotation,
             factors_to_plot = selected_annotations,
             cluster_cols = TRUE,
             color_list = color_list,
             show_rownames = FALSE, show_colnames = FALSE)
```



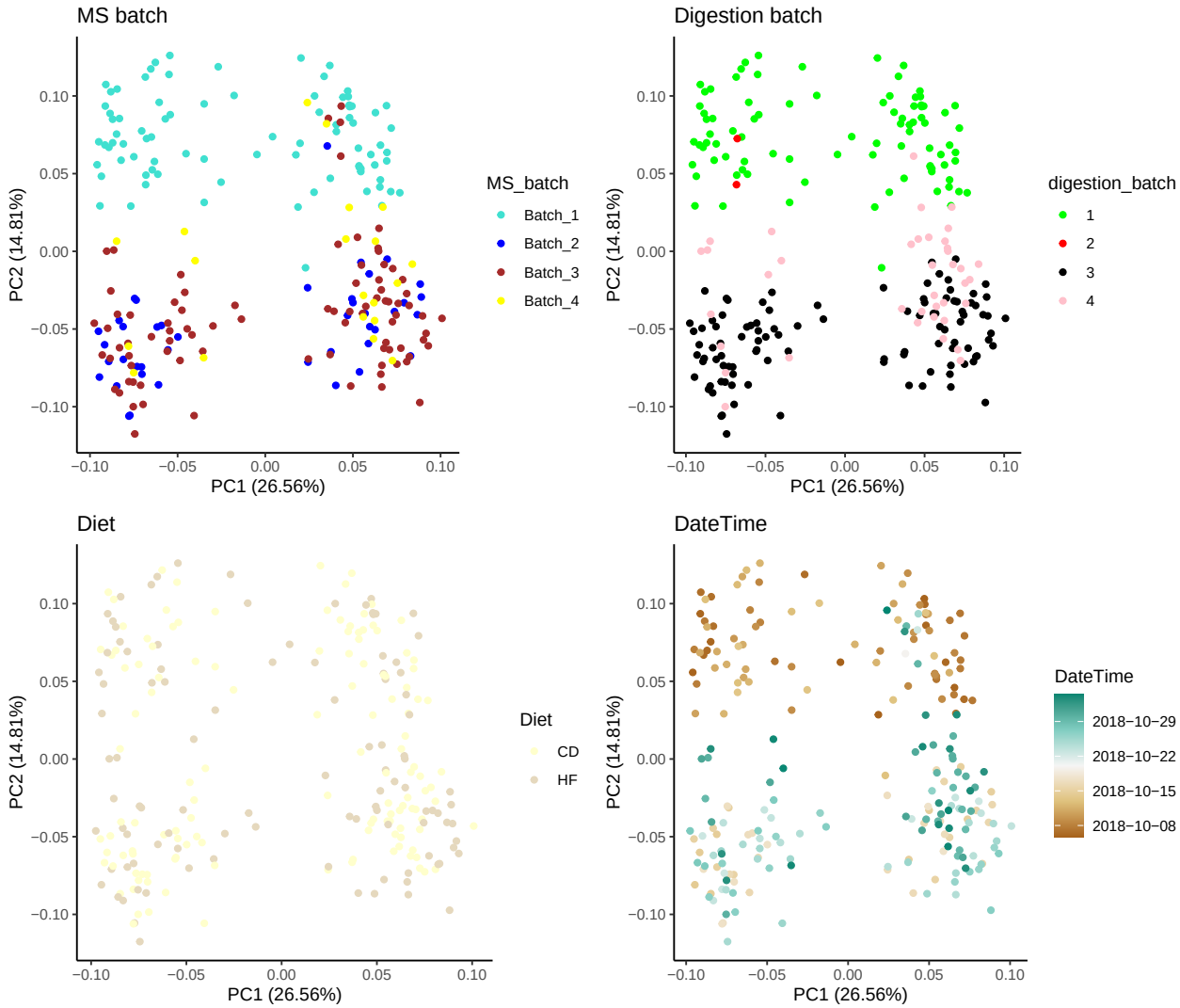
From the clustering analysis, we can clearly see that the driving force behind the sample clustering is the MS batch.

4.3.2 Principal component analysis (PCA)

PCA is a technique that identifies the leading directions of variation, known as principal components. The projection of data on two principal components allows to visualize sample proximity. This technique is particularly convenient to assess replicate similarity.

You can identify the covariate leading the direction of variations by coloring potential candidates.

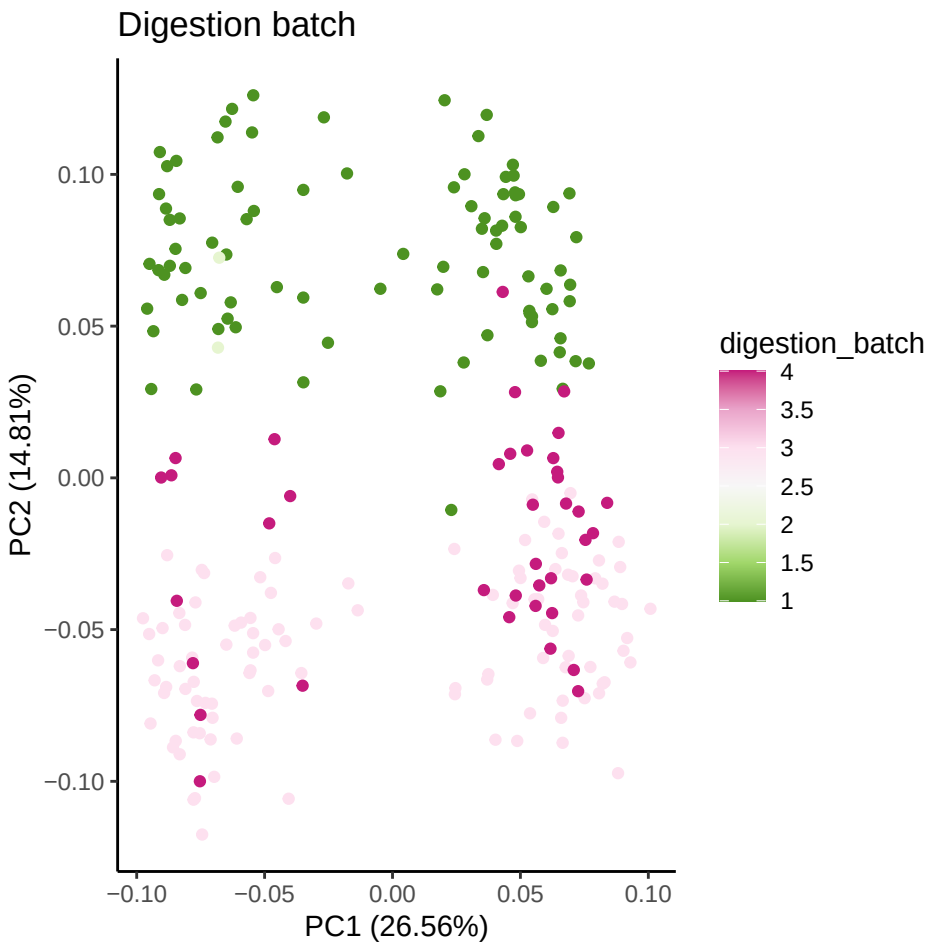
```
pca1 = plot_PCA(quantile_normalized_matrix, example_sample_annotation, color_by = 'MS_batch',
                plot_title = 'MS batch', color_scheme = color_list[['MS_batch']])
pca2 = plot_PCA(quantile_normalized_matrix, example_sample_annotation, color_by = 'digestion_batch',
          plot_title = 'Digestion batch', color_scheme = color_list[['digestion_batch']])
pca3 = plot_PCA(quantile_normalized_matrix, example_sample_annotation, color_by = 'Diet',
          plot_title = 'Diet', color_scheme = color_list[['Diet']])
pca4 = plot_PCA(quantile_normalized_matrix, example_sample_annotation, color_by = 'DateTime',
          plot_title = 'DateTime', color_scheme = color_list[['DateTime']])

library(ggpubr)
ggarrange(pca1, pca2, pca3, pca4, ncol = 2, nrow = 2)
```



By plotting the first two principal components and applying different color overlaps, we see once again that the clusters overlap nicely with the MS batches.

Note that when the color scheme is not provided, `digestion_batch` is mapped to continuous scale. Which is why defining color scheme is so important.

```
pca_spec = plot_PCA(quantile_normalized_matrix, example_sample_annotation,
                    color_by = 'digestion_batch',
          plot_title = 'Digestion batch')
pca_spec
```

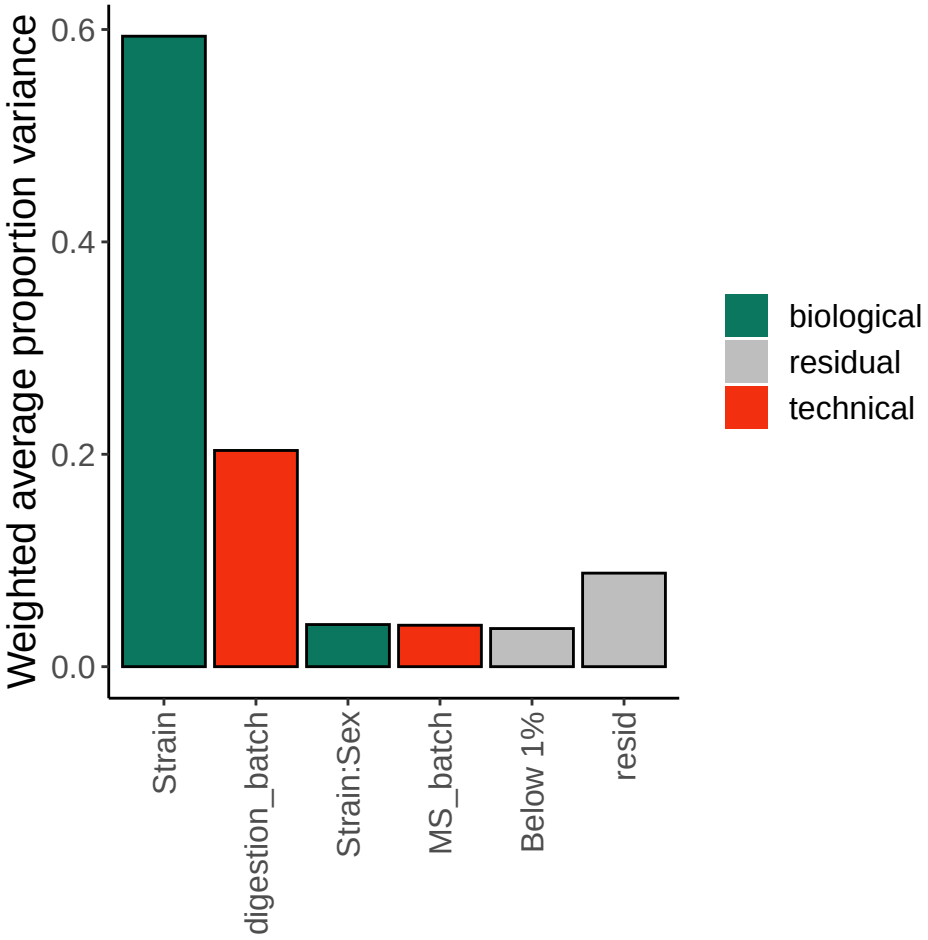


4.3.3 Principal variance component analysis (PVCA)

The main advantage of this approach is the quantification of the variance, associated with both technical and biological covariates. Briefly, principal variance component analysis uses a linear model to match each principal component to the sources of variation and weighs the variance of each covariate by the eigenvalue of the PC [7]. Thus, the resulting value reflects the variance explained by that covariate.

NB: PVCA calculation is a computationally demanding procedure. For a data matrix of several hundred samples and several thousands of peptides it can easily take several hours. So it is generally a good idea to run this analysis as a stand-alone script on a powerful machine.

```
plot_PVCA(quantile_normalized_matrix, example_sample_annotation,
                  technical_factors = technical_factors,
                  biological_factors = biological_factors)
```



The biggest proportion of variance in the peptide measurement was derived from mass spectrometry batches. In a typical experiment, the overall magnitude of variances coming from biological factors should be high while technical variance should be kept at minimum[1].

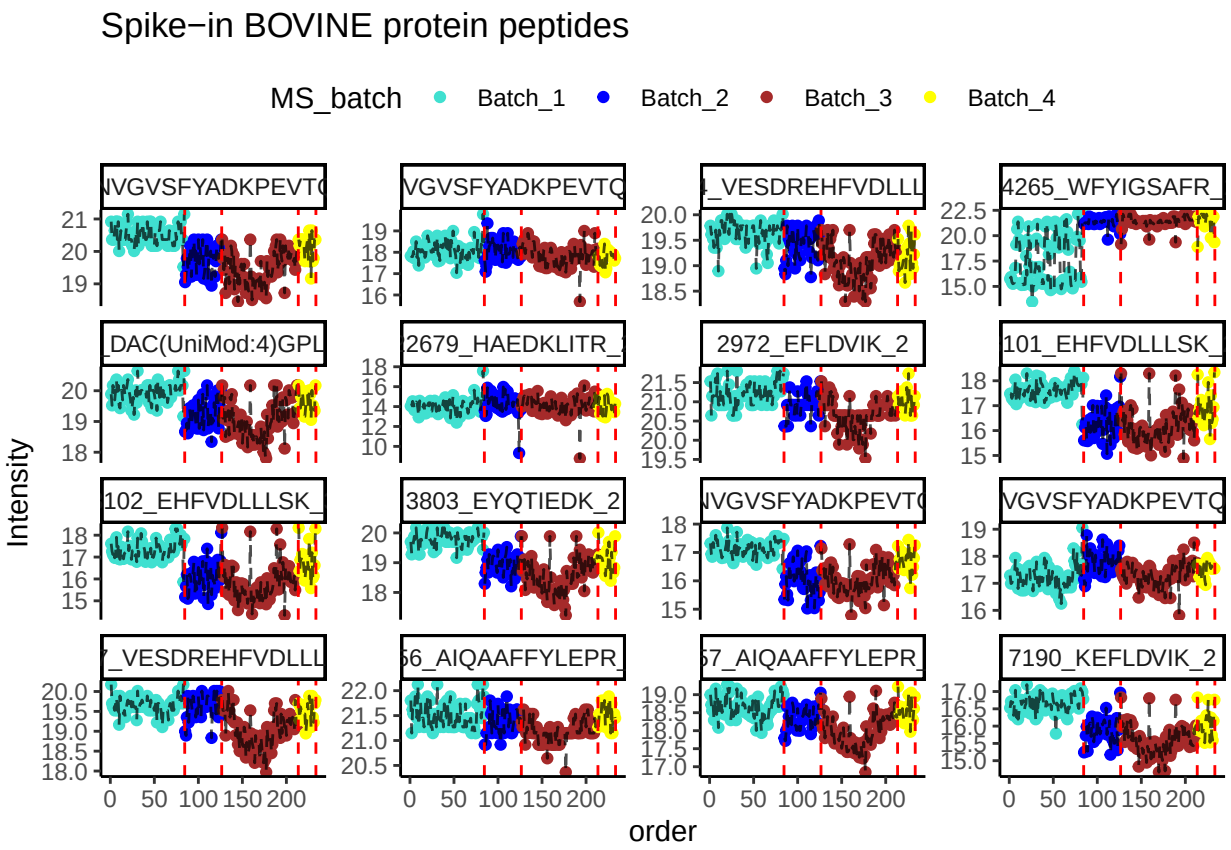
[1]Application of hierarchical clustering, PCA and PVCA in their classical implementation is not possible if missing values are present in the matrix. It has been noticed previously that missing values can be associated with technical bias [8], and most commonly, it is suggested that missing values need to be imputed [8-9]. However, we would like to suggest to use missing value imputation with extreme caution. First of all, missing value imputation alters the sample proximity. Additionally, imputed missing values, which can be obtained for SWATH data, can alter the correction of the batches.

4.3.4 Peptide-level diagnostics and spike-ins

Feature-level diagnostics are very informative for batch effect correction. To assess the bias in the data, one can choose a feature (peptide, protein, gene), the quantitative behavior of which is known. In our package, `plot_peptides_of_one_protein()` allows plotting peptides of interest e.g. from biologically well understood protein. If spike-in proteins or peptides have been added to the mixture, one can use the `plot_spike_ins()` function instead. In most DIA datasets iRT peptides [10] are added in controlled quantities and can be visualized with the `plot_iRT()` function.

In mass-spectrometry, also the trends associated with this order can be assessed for a few representative peptides, thus the `order` column is also important for this diagnostics.

```
quantile_normalized_long <- matrix_to_long(quantile_normalized_matrix)
plot_spike_in(quantile_normalized_long, example_sample_annotation,
              peptide_annotation = example_peptide_annotation,
              protein_col = 'Gene', spike_ins = 'BOVINE_A1ag',
              plot_title = 'Spike-in BOVINE protein peptides',
              color_by_batch = TRUE, color_scheme = color_list[[batch_col]])
```



It is clear that while the pre-determined quantities of spike-ins or peptides of known biology have their expected intensities, the trend is dominated by mass spectrometry signal drift. After confirming either continuous or discrete batch effects exist in a dataset, by one or more of these methods, proceed by selecting a batch correction method.

4.4 Correction of batch effects

Depending on the type of batch effects, different batch correction methods should be implemented. In most cases, batch-specific signal shift needs to be corrected. For this case, feature median-centering can be applied, or, to use across-feature information in a Bayesian framework, the ComBat approach can be used. If there is continuous drift in the data, one has to start from continuous drift correction.

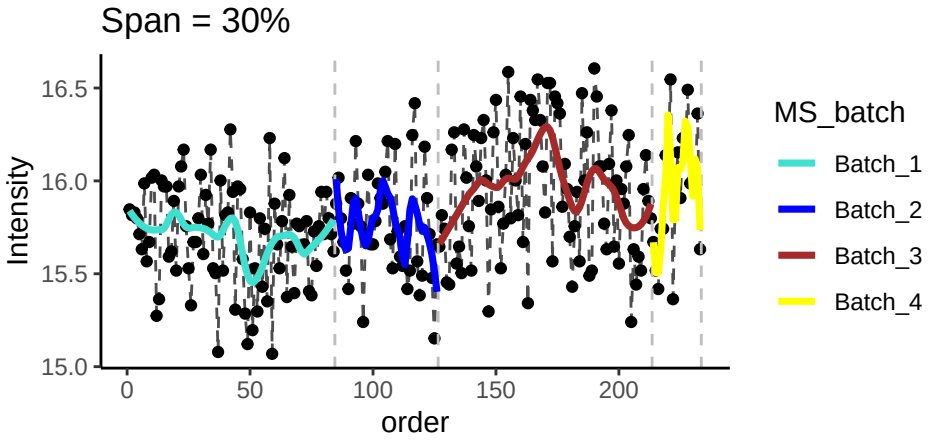
4.4.1 Continuous drift correction

Continuous drifts are specific to mass-spectrometry, thus, the user-friendly methods for its correction have not been implemented before. In this package, we suggest a novel procedure to correct for MS signal drift. We developed a procedure based on nonlinear LOESS fitting. For each peptide and each batch, a non-linear trend is fitted to the normalized data and this trend is subtracted to correct for within-batch variation. Note, that the resulting data are not batch-free as within-batch means and variances are batch-dependent. However, now the batches are discrete and thus can be corrected using discrete methods such as median-centering or ComBat.

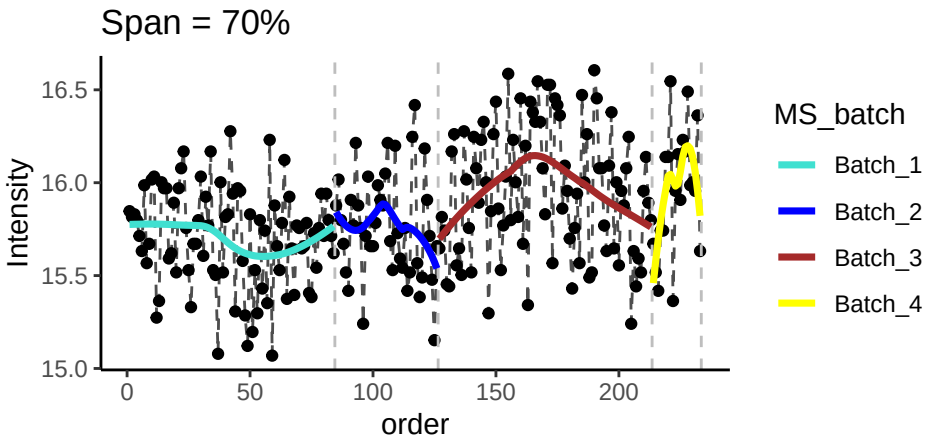
```
loess_fit_df <- adjust_batch_trend_df(quantile_normalized_long, example_sample_annotation)
```

One important parameter in LOESS fitting is span, which determines the degree of smoothing. The LOESS span ranges from 0 to 1: the greater the value of span, the smoother is the fitted curve. Since we want the curve to reflect signal drift, we want to avoid overfitting but be sensitive to fit the trend accurately. Currently, we suggest to evaluate several peptides to determine the best smoothing degree for a given dataset.

```
loess_fit_30 <- adjust_batch_trend_df(quantile_normalized_long, example_sample_annotation,
                                      span = 0.3)

plot_with_fitting_curve(feature_name = '10231_QDVDVWLWQQEGSSK_2',
                        fit_df = loess_fit_30, fit_value_col = 'fit',
                        df_long = quantile_normalized_long,
                        sample_annotation = example_sample_annotation,
                        color_by_batch = TRUE, color_scheme = color_list[[batch_col]],
                        plot_title = 'Span = 30%')
```




```
loess_fit_70 <- adjust_batch_trend_df(quantile_normalized_long, example_sample_annotation,
                                      span = 0.7)
plot_with_fitting_curve(feature_name = '10231_QDVDVWLWQQEGSSK_2',
                        fit_df = loess_fit_70, fit_value_col = 'fit',
                        df_long = quantile_normalized_long,
                        sample_annotation = example_sample_annotation,
                        color_by_batch = TRUE, color_scheme = color_list[[batch_col]],
                        plot_title = 'Span = 70%')
```



Curve fitting is largely dependent on the number of consecutive measurements. In proteomics, missing values are not uncommon. If too many points are missing, a curve cannot be fit accurately. This is especially common for small batches. In this case, we suggest to not fit the curve to the specific peptide within the specific batch, and proceed directly to discrete correction methods. To identify such peptides, absolute and relative thresholds (`abs_threshold` and `pct_threshold`) on the number of missing values for each peptide can be passed as parameters to `adjust_batch_trend_df()`.

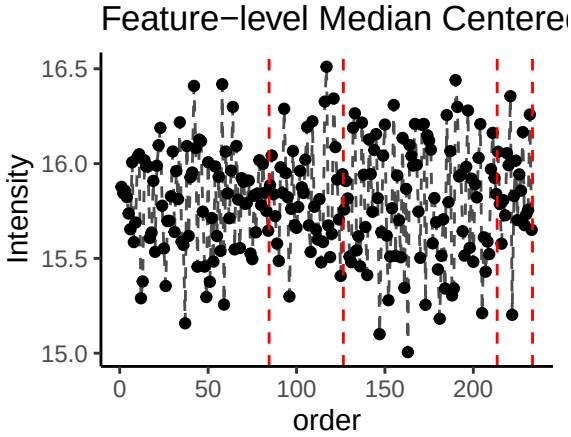
4.4.2 Discrete batch correction: combat or peptide-level median centering

Once the data are normalized and corrected for continuous drift, only discrete batch effects is left to be corrected.

Currently, two methods of batch correction are implemented: * median centering (per feature per batch) * ComBat

4.4.2.1 Feature-level median centering Feature-level median centering is the simplest approach for batch effects correction. However, if the variance is different between batches, other approaches need to be used.

```
peptide_median_df <- center_feature_batch_medians_df(loess_fit_df, example_sample_annotation)
plot_single_feature(feature_name = '10231_QDVDVWLWQQEGSSK_2', df_long = peptide_median_df,
            sample_annotation = example_sample_annotation, measure_col = 'Intensity',
            plot_title = 'Feature-level Median Centered')
```



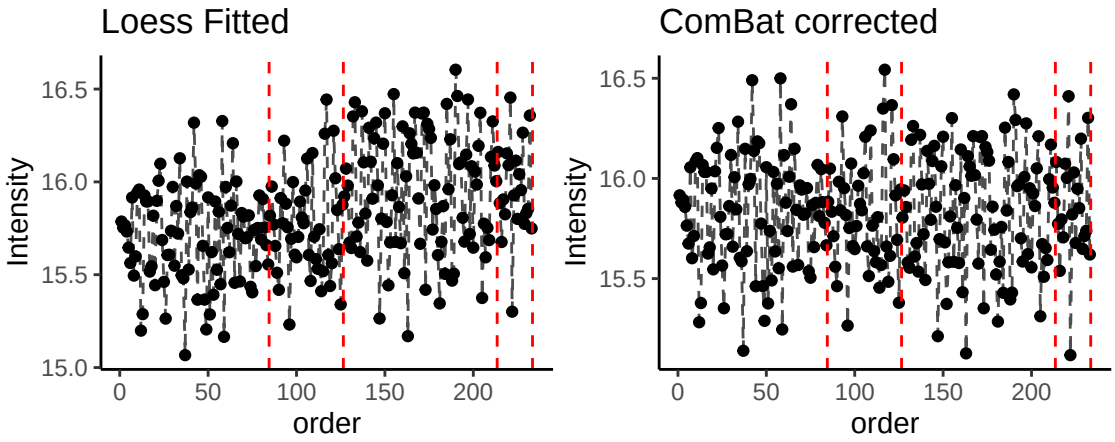
4.4.2.2 ComBat ComBat is well-suited for batches with distinct distributions, but restricted to peptides that don't have missing batch measurements. ComBat, uses parametric and non-parametric empirical Bayes

framework for adjusting data for batch effects [11]. The function `correct_with_ComBat_df()` can incorporate several covariates and make data comparable across batches.

```
comBat_df <- correct_with_ComBat_df(loess_fit_df, example_sample_annotation)
```

To illustrate the correction we use the '10231_QDVDVWLWQQEGSSK_2' spike-in peptide.

```
plot_single_feature(feature_name = '10231_QDVDVWLWQQEGSSK_2',
                    df_long = loess_fit_df,
                    sample_annotation = example_sample_annotation,
                    plot_title = 'Loess Fitted')
plot_single_feature(feature_name = '10231_QDVDVWLWQQEGSSK_2',
                    df_long =  comBat_df,
                    sample_annotation = example_sample_annotation,
                    plot_title = 'ComBat corrected')
```



ComBat fixed the discrete batch effects and also made the distributions between batches similar to one another.

4.4.3 Correct batch effects: universal function

We provide a convenient all-in-one function for batch correction. The function `correct_batch_effects_df()` corrects MS signal drift and discrete shift in a single function call. Simply specify which discrete correction method is preferred at `discrete_func` either "ComBat" or "MedianCentering" and supplement other arguments such as `span`, `abs_threshold` or `pct_threshold` as in `adjust_batch_trend_df()`.

```
batch_corrected_df <- correct_batch_effects_df(df_long = quantile_normalized_long,
                                               sample_annotation = example_sample_annotation,
                                               discrete_func = 'ComBat',
                                               continuous_func = 'loess_regression',
                                               abs_threshold = 5, pct_threshold = 0.20)
batch_corrected_matrix = long_to_matrix(batch_corrected_df)
```

4.5 Quality control on batch-corrected data matrix

In most cases, the batch effects correction method is evaluated by its ability to remove technical confounding, visible on hierarchical clustering or PCA. However, it is rarely shown whether the biological signal is not destroyed, or, better even, improved. Often, and increase in the number of differentially expressed genes is presented as an improvement. However, every reasonably designed experiment has replicates that can serve as an excellent control. In addition, peptides within a given protein should behave similarly and correlation of these peptides should improve after batch correction.

4.5.1 Heatmap of selected replicate samples

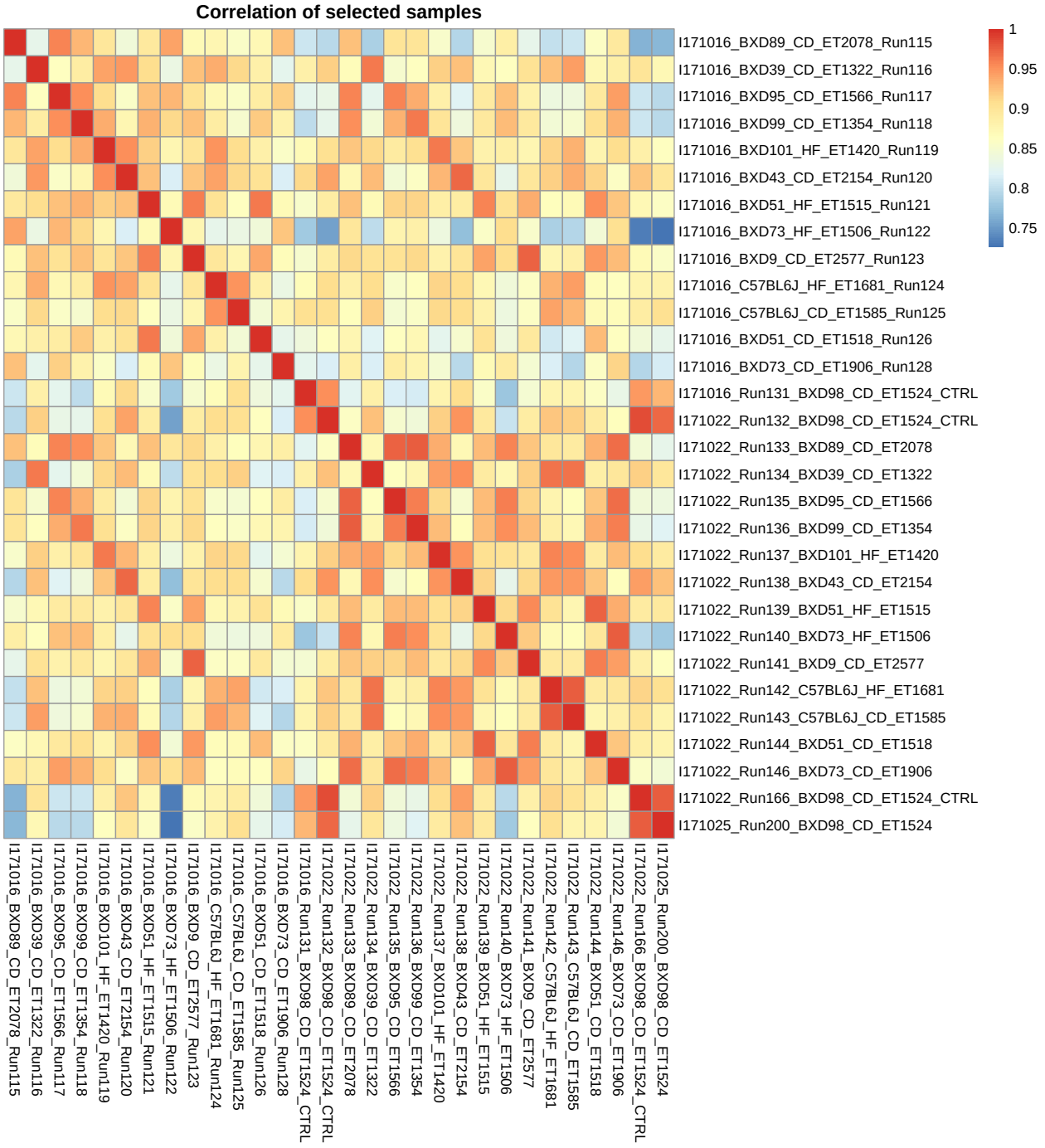
In this study, 10 samples were run in the same order before and after the tuning of the mass-spectrometer, which marks the boundary between batches 2 and 3. The correlation between these replicates can be illustrated by correlation plot heatmap.

First, we specify, which samples we want to correlate

```
earTags <- c('ET1524', 'ET2078', 'ET1322', 'ET1566', 'ET1354', 'ET1420', 'ET2154',
             'ET1515', 'ET1506', 'ET2577', 'ET1681', 'ET1585', 'ET1518', 'ET1906')

replicate_filenames = example_sample_annotation %>%
  filter(MS_batch %in% c('Batch_2', 'Batch_3')) %>%
  filter(EarTag %in% earTags) %>%
  pull(!!sym('FullRunName'))
```

We plot the 'exploratory' correlation matrix, which can be further beautified, see the next chunks.

```
p1_exp = plot_sample_corr_heatmap(log_transformed_matrix,
                                  samples_to_plot = replicate_filenames,
                                  plot_title = 'Correlation of selected samples')
```

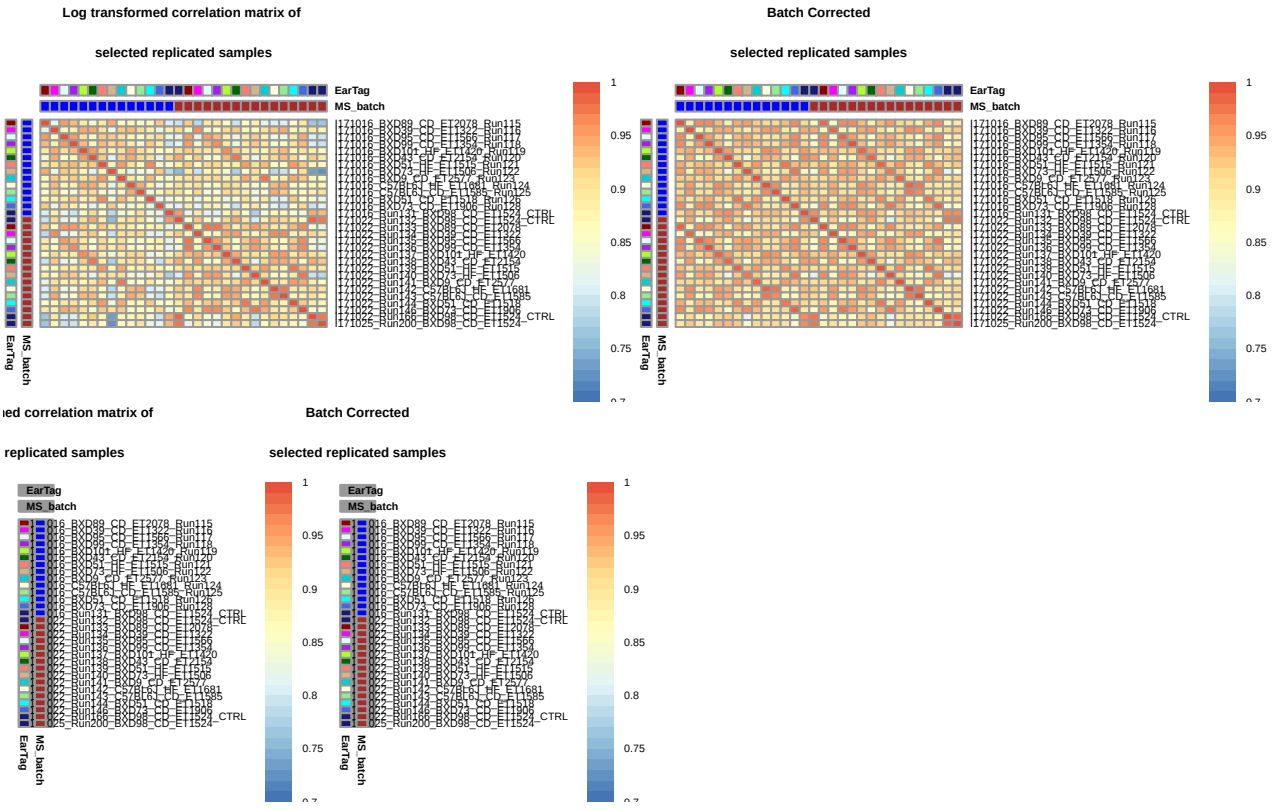


To ensure the color scale for correlation is consistent between two plots, we create a color vector and breaks

```
breaksList <- seq(0.7, 1, by = 0.01) # color scale of pheatmap
heatmap_colors = colorRampPalette(
  rev(RColorBrewer::brewer.pal(n = 7, name = 'RdYlBu')))(length(breaksList) + 1)
```

```
# Plot the heatmap with annotations for the chosen samples
factors_to_show = c(batch_col, biospecimen_id_col)

p1 = plot_sample_corr_heatmap(log_transformed_matrix,
                              samples_to_plot = replicate_filenames,
```

```
                            sample_annotation = example_sample_annotation,
                            factors_to_plot = factors_to_show,
                            plot_title = 'Log transformed correlation matrix of
                            \nselected replicated samples',
                            color_list = color_list,
                            heatmap_color = heatmap_colors, breaks = breaksList,
                            cluster_rows= FALSE, cluster_cols=FALSE,fontsize = 4,
                            annotation_names_col = TRUE, annotation_legend = FALSE,
                            show_colnames = FALSE)

p2 = plot_sample_corr_heatmap(batch_corrected_matrix,
                            samples_to_plot = replicate_filenames,
                            sample_annotation = example_sample_annotation,
                            factors_to_plot = factors_to_show,
                             plot_title = 'Batch Corrected
                            \nselected replicated samples',
                            color_list = color_list,
                            heatmap_color = heatmap_colors, breaks = breaksList,
                            cluster_rows= FALSE, cluster_cols=FALSE,fontsize = 4,
                            annotation_names_col = TRUE, annotation_legend = FALSE,
                            show_colnames = FALSE)
library(gridExtra)
grid.arrange(grobs = list(p1$gtable, p2$gtable), ncol=2)
```



Before the correction, samples from one batch correlate better and of ten higher than the replicates. However, after the correction, the correlation between replicates becomes higher than the correlation between non-related samples regardless of the batch.

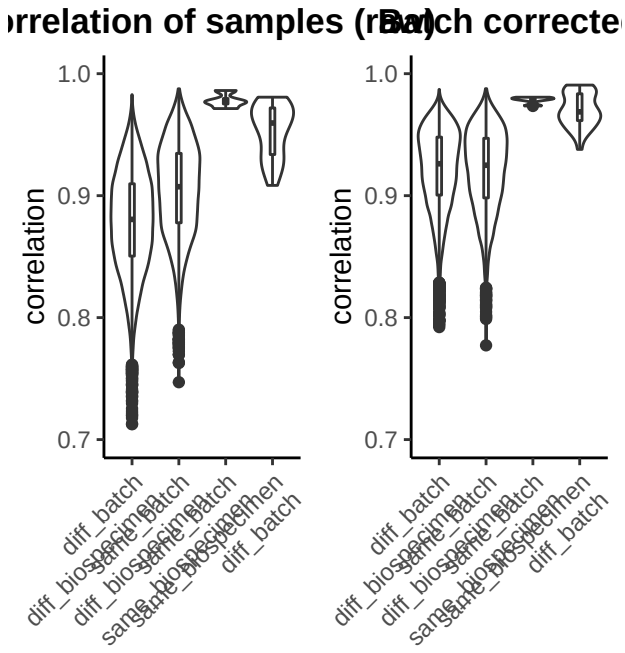
4.5.2 Correlation distribution of samples

For example, in the mice aging experiment, biological replicates, `ET1506` and `ET1524`, were injected every 30-40 MS runs. The correlation between these biological replicates should improve after normalization and batch correction.

The `plot_sample_corr_distribution()` function plots correlation distribution between biological replicates and non-replicates in the same or different batches by `plot_param = 'batch_replicate'`. Alternatively, you can compute the correlation between different batches by `plot_param = 'batches'`.

It should be noted, however, that the comparison of sample correlation should not be approached by evaluating the individual examples of within-replicate vs within-batch corrections, but rather by comparing the distribution. Unless these examples are shown in the context of the whole distribution structure, they can lead to erroneous conclusion. The sample correlation is often used to prove the quality of the measurement, as it is typically very high (examples of the replicate correlation above .95 are common for mass spectrometry).

```
sample_cor_raw <- plot_sample_corr_distribution(log_transformed_matrix,
                                                example_sample_annotation,
                                                #repeated_samples = replicate_filenames,
                                                batch_col = 'MS_batch',
                                                biospecimen_id_col = 'EarTag',
                                                plot_title = 'Correlation of samples (raw)',
                                                plot_param = 'batch_replicate')
raw_corr = sample_cor_raw +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  ylim(0.7,1) + xlab(NULL)

sample_cor_batchCor <- plot_sample_corr_distribution(batch_corrected_matrix,
                                                     example_sample_annotation,
                                                     batch_col = 'MS_batch',
                                                     plot_title = 'Batch corrected',
                                                     plot_param = 'batch_replicate')
corr_corr = sample_cor_batchCor +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  ylim(0.7, 1) + xlab(NULL)

library(gtable)
library(grid)
g2 <- ggplotGrob(raw_corr)
g3 <- ggplotGrob(corr_corr)
g <- cbind(g2, g3, size = 'first')
grid.draw(g)
```



4.5.3 Correlation of peptide distributions within and between proteins

Peptides of the same protein are likely to correlate. Therefore, we can compare within- vs between-protein peptide correlation before and after batch correction to check if the correlation of peptides between the same proteins increases while that of different proteins stays the same.

NB: For a data matrix containing several thousands of peptides, calculation of peptide correlation is a computationally demanding procedure. It can easily take several hours. Therefore, it is generally recommended to run this analysis as a stand-alone script on a powerful machine.

The `plot_peptide_corr_distribution()` function plots correlation distribution between peptides of the same protein. However, an improvement of peptide correlation may not be clearly exhibited for a reduced dataset.

```
peptide_cor_raw <- plot_peptide_corr_distribution(log_transformed_matrix,
                                                  example_peptide_annotation,
                                                  protein_col = 'Gene',
                                                  plot_title = 'Peptide correlation (raw)')

peptide_cor_batchCor <- plot_peptide_corr_distribution(batch_corrected_matrix,
                                                       example_peptide_annotation,
                                                       protein_col = 'Gene',
                                                       plot_title = 'Peptide correlation (batch correcte
g2 <- ggplotGrob(peptide_cor_raw+
  ylim(c(-1, 1)))
g3 <- ggplotGrob(peptide_cor_batchCor+
  ylim(c(-1, 1)))
g <- cbind(g2, g3, size = 'first')
grid.draw(g)
```

5 SessionInfo

```
sessionInfo()
#> R version 4.1.0 RC (2021-05-16 r80304)
#> Platform: x86_64-apple-darwin17.0 (64-bit)
#> Running under: macOS Mojave 10.14.6
#>
#> Matrix products: default
#> BLAS:   /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRblas.dylib
#> LAPACK: /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRlapack.dylib
#>
#> locale:
#> [1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
#>
#> attached base packages:
#> [1] grid      stats     graphics  grDevices utils     datasets  methods
#> [8] base
#>
#> other attached packages:
#> [1] gtable_0.3.0   gridExtra_2.3  ggpubr_0.4.0   knitr_1.33     proBatch_1.9.0
#> [6] ggplot2_3.3.3  tibble_3.1.2   dplyr_1.0.6
#>
#> loaded via a namespace (and not attached):
#>   [1] readxl_1.3.1           backports_1.2.1        Hmisc_4.5-0
#>   [4] corrplot_0.88          plyr_1.8.6             lazyeval_0.2.2
#>   [7] splines_4.1.0          BiocParallel_1.27.0    GenomeInfoDb_1.29.0
#>  [10] sva_3.41.0             digest_0.6.27          foreach_1.5.1
#>  [13] htmltools_0.5.1.1      viridis_0.6.1          GO.db_3.13.0
#>  [16] wesanderson_0.3.6      fansi_0.4.2            ggfortify_0.4.11
#>  [19] magrittr_2.0.1         checkmate_2.0.0        memoise_2.0.0
#>  [22] cluster_2.1.2          doParallel_1.0.16      openxlsx_4.2.3
#>  [25] limma_3.49.0           fastcluster_1.1.25     Biostrings_2.61.0
#>  [28] annotate_1.71.0        matrixStats_0.58.0     jpeg_0.1-8.1
#>  [31] colorspace_2.0-1       blob_1.2.1             haven_2.4.1
#>  [34] xfun_0.23              crayon_1.4.1           RCurl_1.98-1.3
#>  [37] genefilter_1.75.0      lme4_1.1-27            impute_1.67.0
#>  [40] survival_3.2-11        iterators_1.0.13       glue_1.4.2
#>  [43] zlibbioc_1.39.0        XVector_0.33.0         car_3.0-10
#>  [46] BiocGenerics_0.39.0    abind_1.4-5            scales_1.1.1
#>  [49] pheatmap_1.0.12        vsn_3.61.0             DBI_1.1.1
#>  [52] edgeR_3.35.0           rstatix_0.7.0          Rcpp_1.0.6
#>  [55] viridisLite_0.4.0      xtable_1.8-4           htmlTable_2.2.1
#>  [58] foreign_0.8-81         bit_4.0.4              preprocessCore_1.55.0
#>  [61] Formula_1.2-4          stats4_4.1.0           htmlwidgets_1.5.3
#>  [64] httr_1.4.2             RColorBrewer_1.1-2     ellipsis_0.3.2
#>  [67] pkgconfig_2.0.3        XML_3.99-0.6           farver_2.1.0
#>  [70] nnet_7.3-16            locfit_1.5-9.4         utf8_1.2.1
#>  [73] dynamicTreeCut_1.63-1  tidyselect_1.1.1       labeling_0.4.2
#>  [76] rlang_0.4.11           reshape2_1.4.4         AnnotationDbi_1.55.0
#>  [79] cellranger_1.1.0       munsell_0.5.0          tools_4.1.0
#>  [82] cachem_1.0.5           generics_0.1.0         RSQLite_2.2.7
#>  [85] broom_0.7.6            evaluate_0.14          stringr_1.4.0
#>  [88] fastmap_1.1.0          yaml_2.2.1             bit64_4.0.5
```

```
#>  [91] zip_2.1.1             purrr_0.3.4           KEGGREST_1.33.0
#>  [94] nlme_3.1-152          compiler_4.1.0        rstudioapi_0.13
#>  [97] curl_4.3.1            png_0.1-7             affyio_1.63.0
#> [100] ggsignif_0.6.1        stringi_1.6.2         highr_0.9
#> [103] forcats_0.5.1         lattice_0.20-44       Matrix_1.3-3
#> [106] nloptr_1.2.2.2        vctrs_0.3.8           pillar_1.6.1
#> [109] lifecycle_1.0.0       BiocManager_1.30.15   pvca_1.33.0
#> [112] cowplot_1.1.1         data.table_1.14.0     bitops_1.0-7
#> [115] R6_2.5.0              latticeExtra_0.6-29   affy_1.71.0
#> [118] rio_0.5.26            IRanges_2.27.0        codetools_0.2-18
#> [121] boot_1.3-28           MASS_7.3-54           assertthat_0.2.1
#> [124] withr_2.4.2           S4Vectors_0.31.0      GenomeInfoDbData_1.2.6
#> [127] hms_1.1.0             mgcv_1.8-35           parallel_4.1.0
#> [130] rpart_4.1-15          tidyr_1.1.3           minqa_1.2.4
#> [133] rmarkdown_2.8         carData_3.0-4         Biobase_2.53.0
#> [136] WGCNA_1.70-3          lubridate_1.7.10      base64enc_0.1-3
```

6 Citation

To cite this package, please use:

```
citation('proBatch')
#>
#> To cite proBatch in publications use:
#>
#>   Cuklina J., Lee C.H., Williams E.G., Collins B., Sajic T., Pedrioli
#>   P., Rodriguez-Martinez M., Aebersold R. Chapter 3: Systematic
#>   overview of batch effects in proteomics Doctoral thesis, ETH Zurich,
#>   2018 https://doi.org/10.3929/ethz-b-000307772
#>
#> A BibTeX entry for LaTeX users is
#>
#>   @PhdThesis{,
#>     title = {Computational challenges in biomarker discovery from high-throughput proteomic data},
#>     author = {Jelena Cuklina and Chloe H. Lee and Evan G. Willams and Ben Collins and Tatjana Sajic
#>     url = {https://doi.org/10.3929/ethz-b-000307772},
#>     school = {ETH Zurich},
#>     year = {2018},
#>   }
```

7 References

[1] O. T. Schubert, H. L. Röst, B. C. Collins, G.Rosenberger, and R. Aebersold. «Quantitative proteomics: challenges and opportunities in basic and applied research». Nature Protocols 12:7 (2017), pp. 1289–1294.

[2] E. G. Williams, Y. Wu, P. Jha et al. «Systems proteomics of liver mitochondria function». Science 352:6291 (2016), aad0189.

[3] Y. Liu, A. Buil, B. C. Collins et al. «Quantitative variability of 342 plasma proteins in a human twin population». Molecular Systems Biology 11:2 (2015), pp. 786–786.

[4] H. L. Röst, G. Rosenberger, P. Navarro et al. «OpenSWATH enables automated, targeted analysis of data-independent acquisition MS data.» Nature biotechnology 32:3 (2014), pp. 219–23.

[5] P. G. Pedrioli, «Trans-Proteomic Pipeline: A Pipeline for Proteomic Analysis.» Methods in Molecular Biology Proteome Bioinformatics, May 2009, pp. 213–238.

[6] G. Rosenberger et al. «Statistical control of peptide and protein error rates in large-scale targeted data-independent acquisition analysis.» Nature Methods 14:9 (2017), pp. 921–927.

[7] P. R. Bushel. pvca: Principal Variance Component Analysis (PVCA). Package version 1.18.0. 2013.

[8] Y. V. Karpievitch, A. R. Dabney, and R. D. Smith. «Normalization and missing value imputation for label-free LC-MS analysis». BMC Bioinformatics 13:Suppl 16 (2012), S5.

[9] S. Tyanova, T. Temu, P. Sinitcyn et al. «The Perseus computational platform for comprehensive analysis of (prote)omics data». Nat Methods 13:9 (2016), pp. 731–740.

[10] C. Escher, L. Reiter, B. Maclean et al. «Using iRT, a normalized retention time for more targeted measurement of peptides». Proteomics 12:8 (2012), pp. 1111–1121.

[11] A. W. B. Johnston, Y. Li, and L. Ogilvie. «Metagenomic marine nitrogen fixation–feast or famine?» Trends in microbiology 13:9 (2005), pp. 416–20.