proBatch

Jelena Cuklina, Chloe H. Lee, Patrick Pedrioli, Ruedi Aebersold

2018-12-04

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

1.1 Batch effects in large-scale data.

Recent advances in mass-spectrometry enabled fast and near-exhaustive identification and quantification of proteins in complex biological samples [1], which allows to profile large-scale datasets. Obtaining sufficiently large dataset is, however, associated with considerable logistics efforts of multiple biomaterial handlers on sample preparation and data acquisition steps e.g. protein extraction, peptide digestion, instrument cleaning. This introduces systematic technical variation known as the batch effects.

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

1.2 Batch effects analysis workflow

The fundamental objective of the batch effect 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 correction. 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.

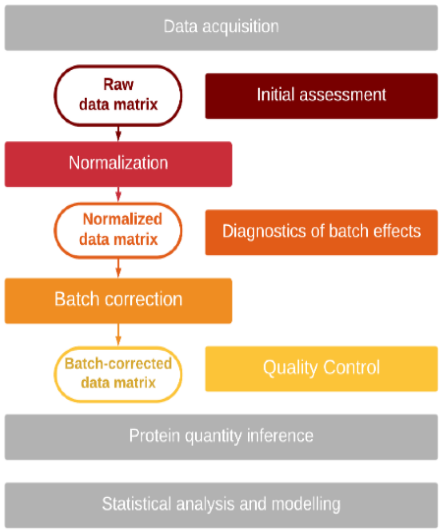


Figure 1: proBatch in batch correction workflow

1.3 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 of this vignette.

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

It is assumed thatthe initial data processing is completed. In mass spectrometry- based proteomics, this involves primarily peptide-spectrum matching [OpenSWATH, TPP] and FDR control [Isa&George papter].

Data filtering is commonly the next step of data processing. In context of batch 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 valuea before correction (also to exclude "requant" values, inferred from SWATH data). Imputed values either add the "average" measurements, or add 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 the protein quantification after the batch effect correction, as the correction procedure alters the abundances of peptides and peptide transitions, and these abundances are critical for protein quantity inference. However, we do recommend to correct the technical noise at the level, which is used to inter the proteins (thus, fragment-level for inference tools such as aLFQ or MSstats).

2 Preparation for the analysis

2.1 Installing dependencies and `proBatch`

```
# Dependencies required for installing proBatch package
bioc_deps <- c("GO.db", "impute", "preprocessCore", "pvca","sva" )
cran_deps <- c("corrplot", "data.table", "ggfortify","lazyeval", "pheatmap", "reshape2",
               "rlang", "tidyverse","wesanderson","WGCNA")
lapply(bioc_deps, require, character.only = TRUE)
lapply(cran_deps, require, character.only = TRUE)
```

If some of these dependencies are not installed, you will need to do that before running `proBatch`.

```
source("https://bioconductor.org/biocLite.R")
biocLite(bioc_deps)
#>
#> The downloaded binary packages are in
#>  /var/folders/77/9bhq5zfd6rz8_mwvc43myl3c0000gr/T//RtmpPvBheF/downloaded_packages
install.packages(cran_deps)
#>
#> The downloaded binary packages are in
#>  /var/folders/77/9bhq5zfd6rz8_mwvc43myl3c0000gr/T//RtmpPvBheF/downloaded_packages
```

To install the proBatch package, the following commands can be executed in R:

```
# Once the proBatch package is in Bioconductor, can easily install by:
install.packages("proBatch")

# Alternatively, install the development version from GitHub:
install.packages("devtools")
devtools::install_github("symbioticMe/proBatch")
```

2.2 Preparing the data for analysis

2.2.1 Format of the data

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

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

- **Measurement table (data matrix)**, either wide 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, requires at least three columns: feature ID column, measurement (intensity) column and sample ID column.

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 OpenSWATH tsv output format.

In the package, we provide the function to convert from long to matrix format (see section XXX "Utility functions").

Note that sample IDs (column names in `data_matrix`) should match the values in sample ID column in `sample_annotation` and feature ID column values should match 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.

- **Sample annotation** is 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 sample ID column, at least one technical and one biological factor column, and 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:

a. `SacrificeDate` - date when tissues were extracted
b. `ProteinPrepDate` - date when samples were prepared
c. `RunDate` (and `RunTime`, if available) - will be used to determine run order
d. `MS_batch` - number of MS batches (in this case, sets of runs between machine cleaning)

3. biological covatiates:

a. `Strain`
b. `Diet`
c. `Sex`

4. `biospecimen_id_col = "EarTag"` ' Thus, technical and biological factors are:

```
technical_covariates = c('MS_batch', 'digestion_batch', 'RunDate', 'RunTime')
biological_covariates = c('Strain', 'Diet', 'Sex', 'Age_Days')
biospecimen_id_col = "EarTag"
```

In the example dataset, you will also find `order` column, which is, however, best inferred from the run date/time with the corresponding function (see Utility functions below).

- **Feature (peptide) annotation**: is 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 is feature ID (`peptide_group_id`) and name of corresponding protein (in this vignette, we use `Gene` name).

2.2.2 Example dataset

The `proBatch` package can be applied to any dataset, for which intensity matrix and sample annotation are available, althogh the package was primarily designed with proteomic data in mind, and thoroughly tested on SWATH data. Thus, as and example data we include a reduced SWATH-MS measurement file, generated from BXD mouse aging study. In this study, liver proteome of mouse from BXD reference population have been profiled to identify proteome changes associated with age. The animals of each strain were subjected to Chow and High-Fat Diet and sacrificed at different age (age factor is excluded from the example data as age-related differences are in the focus of 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 `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 affect different peptides in a batch-specific way, and MS drift, that introduces continous bias associated with sample running order. We will illustrate, how such biases can be corrected. 3. Replicate structure: two animals were injected 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 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")
```

2.2.3 Prepare sample and peptide annotations

Before you skip this section, read whether you need the enhanced functionality that `proBatch` provides for preparing the annotation of samples and peptides.

We provide a few utility functions to facilitate the preparation of sample and peptide annotation.

2.2.3.1 Define the order of samples from running date and time

In proteomics, measurement of samples on after another might introduce order-related effects. To facilitate the examination of such effect, it is necessary to define an order column in sample annotation. Using `date_to_sample_order` function one can infer sample order by from date and time of the measurements.

You can specify the columns illustrating date and time in `time_column` and their formats in `dateTimeFormat` (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	2018-10-05 18:35:00	1
Oct_05	20:12:00	2	2018-10-05 20:12:00	2
Oct_05	21:50:00	3	2018-10-05 21:50:00	3
Oct_05	23:28:00	4	2018-10-05 23:28:00	4
Oct_06	01:51:00	5	2018-10-06 01:51:00	5

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

2.2.3.2 Generate peptide annotation from OpenSWATH data

From OpenSWATH ouput, you can generate peptide annotation using `create_peptide_annotation()` by denoting peptide ID in `feature_id_col` and annotation columns in `annotation_col`.

```
generated_peptide_annotation <- create_peptide_annotation(example_proteome,
                                      feature_id_col = 'peptide_group_label',
                                      annotation_col = c('ProteinName', 'Gene'))
```

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

```
example_proteome = example_proteome %>% select(one_of(essential_columns))
gc()
#>           used  (Mb) gc trigger  (Mb) max used  (Mb)
#> Ncells 3170117 169.4    4703850 251.3  4703850 251.3
#> Vcells 3977783  30.4   14357312 109.6 16610859 126.8
```

Additionally, smaller peptide annotation matrix allows faster mapping of UniProt identifiers to gene names and other IDs.

2.2.4 Other utility functions

2.2.4.1 Transform the data to long or wide format

The plotting tools accept the 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)
```

2.2.4.2 Transform the data to log scale

Additionally, the data is expected to be log-transformed

```
log_transformed_matrix <- log_transform(example_matrix)
```

2.2.4.3 Define the color scheme

For color annotation, you first assign colors to biological and technical covariates by `sample_annotation_to_colors()`. Simply denote columns whether they are factoric, non-factoric or numeric to specify colors into qualitative or sequential color scales.

```
color_scheme <- sample_annotation_to_colors (example_sample_annotation,
       factor_columns = c('MS_batch','EarTag', "Strain", "Diet", "digestion_batch", "Sex"),
       not_factor_columns = 'DateTime',
       numeric_columns = c('Age_Days', 'order'))
color_list = color_scheme$list_of_colors
```

3 Batch effects analysis workflow step-by-step

3.1 Initial assessment of raw data matrix

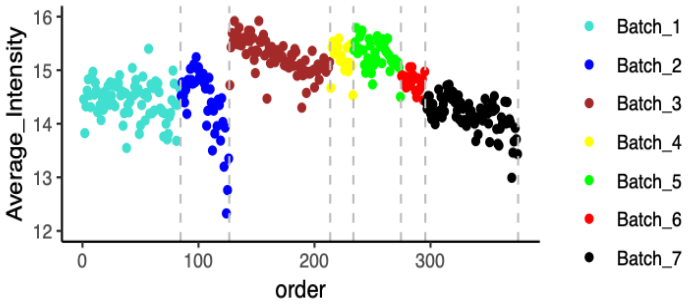
Before any correction, it is informative to set the baseline of the data quality by examining global quantitative patterns of raw data matrix. Commonly, batch effects manifest as batch-specific intensity distribution change

In proteomics, batch-specific intensity drifts of sample mean can occur. Thus, it is important to carefully keep an order of sample measurement by mass spectrometry. Order inference can be performed as shown in previous section "Define order of samples". If the order column is not available (`order_col = NULL`), order of samples in sample annotation is used for plotting.

3.1.1 Plot sample mean

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

```
batch_col = 'MS_batch'
plot_sample_mean(log_transformed_matrix, example_sample_annotation, order_col = 'order',
                 batch_col = batch_col, color_by_batch = T, ylimits = c(12, 16),
                 color_scheme = color_list[[batch_col]])
```

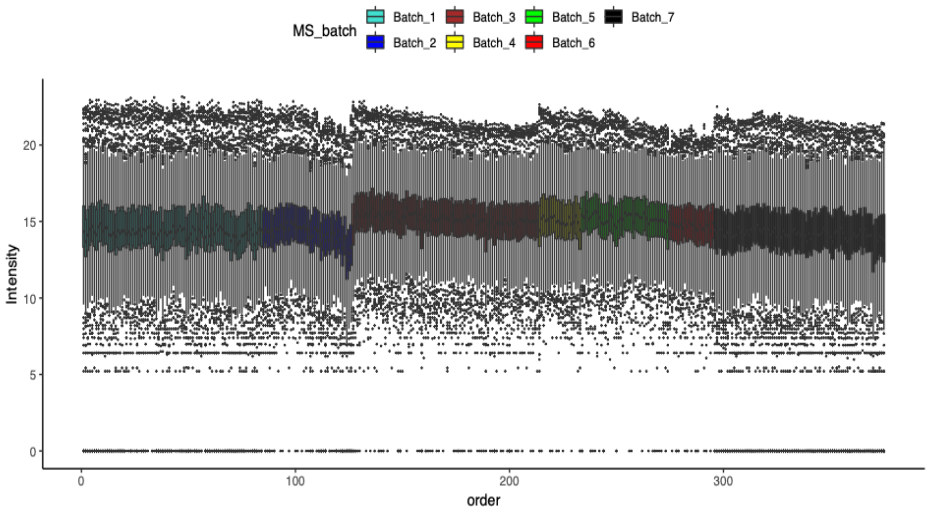


We can clearly see down-sloping trends in the BXD aging dataset. In fact, during the data acquisition, the mass-spectrometer had to be interrupted several times for tuning and/or column exchange as the signal was decreasing.

3.1.2 Plot 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.

3.2 Normalization

In large-scale experiments, 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, but if needed, log-transformation can be done by specifying `log_base = 2` for log2-transformation.

3.2.1 Median normalization

Median normalization is a conservative approach that shifts the intensity of the sample to the global median of the experiment. If 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(log_transformed_matrix,
                                          normalizeFunc = "medianCentering")
```

Same result will be achieved with:

```
median_normalized_matrix = normalize_data(example_matrix,
                                          normalizeFunc = "medianCentering", log_base = 2)
```

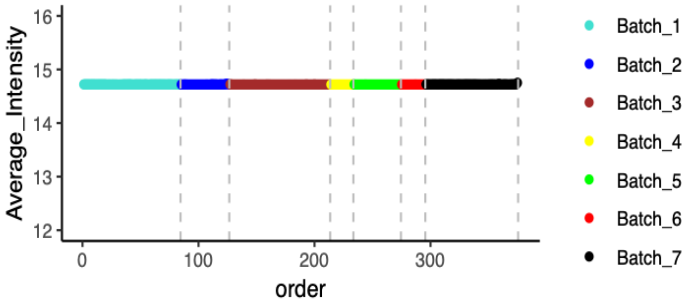
3.2.2 Quantile normalization

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(log_transformed_matrix,
                                            normalizeFunc = "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 = T, ylimits = c(12, 16),
                 color_scheme = color_list[[batch_col]])
```



```
batch_corrected_matrix <- correct_batch_effects(data_matrix = quantile_normalized_matrix,
                                  example_sample_annotation, discreteFunc = 'ComBat',
                                  abs.threshold = 5, pct.threshold = 0.20)
#> Standardizing Data across genes
```

3.3 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.

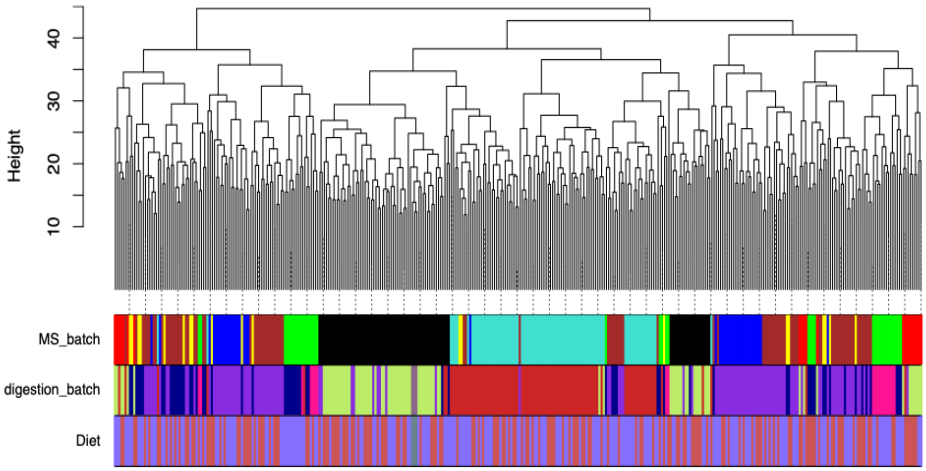
3.3.1 Hierarchical clustering

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 clustering.

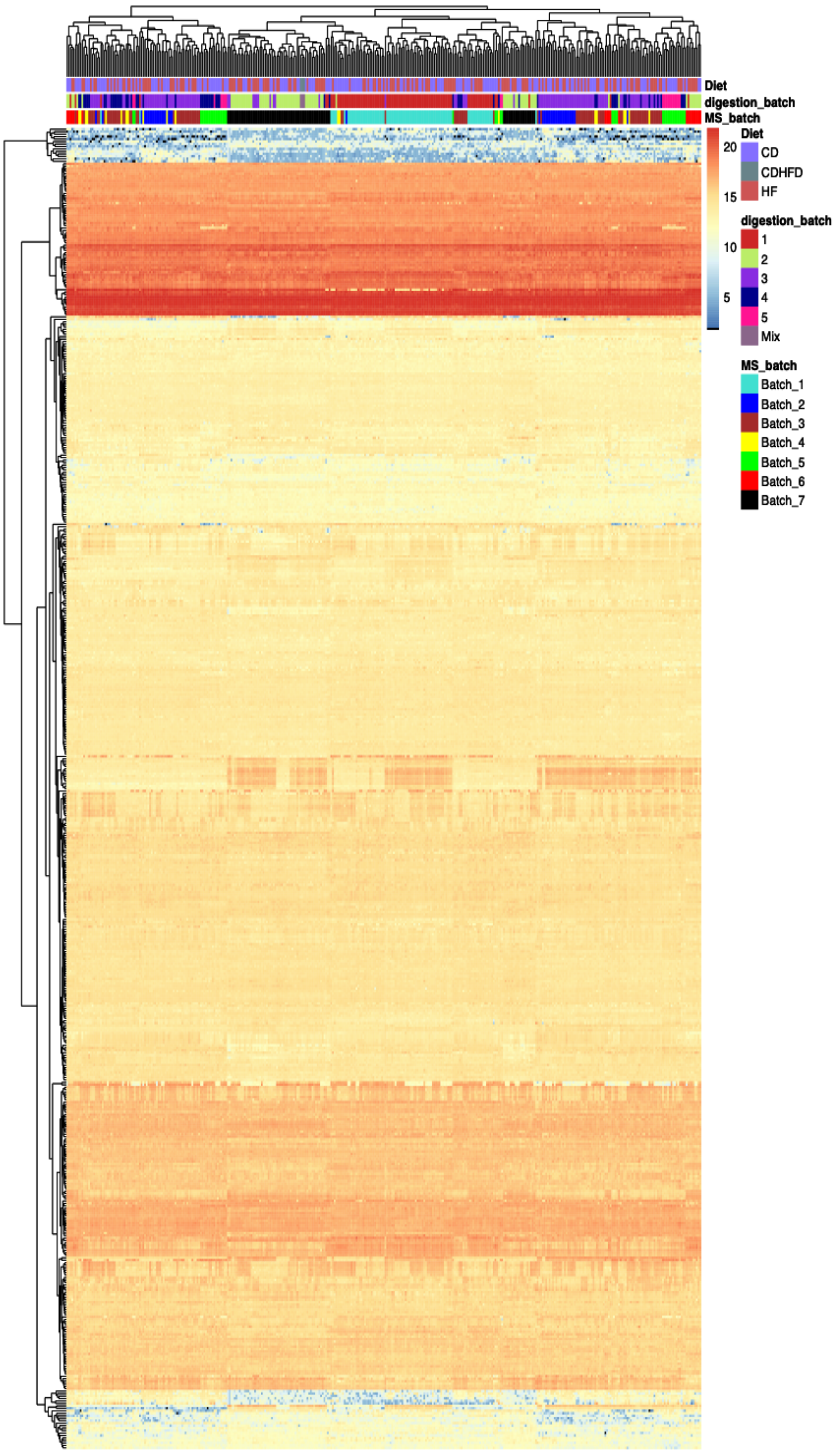
Once your color annotation is ready, for the specific covariates of interest, you can subset the color dataset and feed it into the clustering functions.

```
library(dplyr)
color_annotation <- color_scheme$color_df
selected_annotations <- c("MS_batch",  "digestion_batch", "Diet")
#select only a subset of samples for plotting
color_annotation <- color_annotation %>% select(one_of(selected_annotations))

#Plot clustering between samples
plot_hierarchical_clustering(quantile_normalized_matrix, color_annotation,
                             distance = "euclidean", agglomeration = 'complete',
                             label_samples = F)
```



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 required covariates in `sample_annotation_col`, `sample_annoation_row`, or both.

```
plot_heatmap(quantile_normalized_matrix, example_sample_annotation,
             sample_annotation_col = selected_annotations,
             cluster_cols = T,
             annotation_color_list = color_scheme$list_of_colors,
             show_rownames = F, show_colnames = F)
```


Diet
digestion_batch
MS_batch
20
15
10
5
Diet
CD
CDHFD
HF
digestion_batch
1
2
3
4
5
Mix
MS_batch
Batch_1
Batch_2
Batch_3
Batch_4
Batch_5
Batch_6
Batch_7

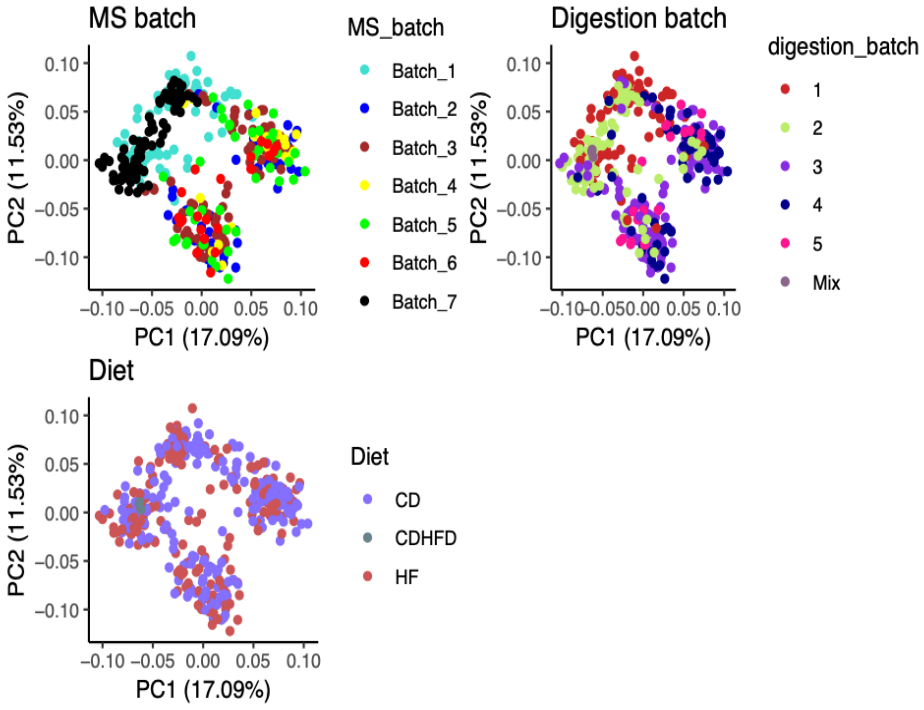
From the clustering analysis, we can clearly see that the driving force behind sample clustering is the MS batch.

3.3.2 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.

```
plot_PCA(quantile_normalized_matrix, example_sample_annotation, color_by = 'MS_batch',
             plot_title = "MS batch", colors_for_factor = color_list[[batch_col]])
plot_PCA(quantile_normalized_matrix, example_sample_annotation, color_by = "digestion_batch",
         plot_title = "Digestion batch", colors_for_factor = color_list[["digestion_batch"]])
plot_PCA(quantile_normalized_matrix, example_sample_annotation, color_by = "Diet",
         plot_title = "Diet", colors_for_factor = color_list[["Diet"]])
```



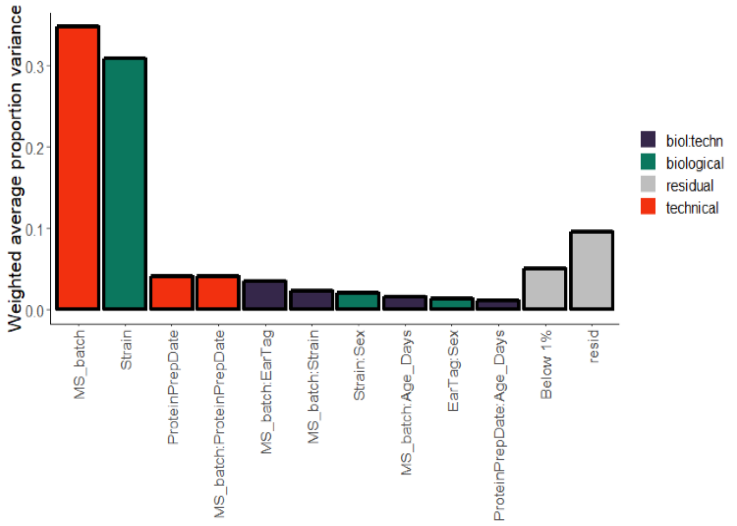
By plotting the first two principal components and applying different color overlaps, we see once again that clusters overlap nicely with MS batches.

3.3.3 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 [19]. Thus, the resulting value reflects the variance explained by that covariate.

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

```
pvca <- plot_PVCA(quantile_normalized_matrix, example_sample_annotation,
                  technical_covariates = c('MS_batch', 'digestion_batch'),
                  biological_covariates = c(biological_covariates, biospecimen_id_col))
```



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

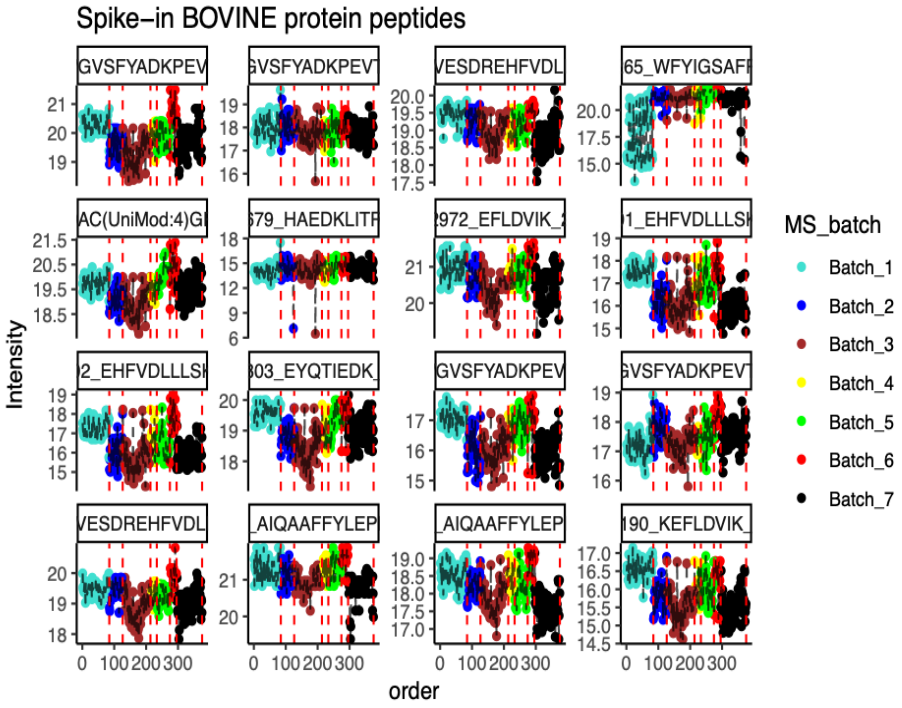
3.3.4 Peptide-level diagnostics and spike-ins

Featre-level diagnostics is very informative for batch effect correction. To assess the bias in the data, one can choose a feature (peptide, protein, gene), quantitative behaviour of which is known. Through our package, `plot_peptides_of_one_protein()` allow plotting peptides of interest e.g. a biologically well understood protein. If spike-in proteins or peptides have been added to the mixture, one can use the `plot_spike_ins()` function. In most DIA datasets iRT peptides [22] are added in controlled quantities and can be visualised with `plot_iRT()` function.

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

```
quantile_normalized_long <- matrix_to_long(quantile_normalized_matrix, example_sample_annotation)
plot_spike_in(quantile_normalized_long, example_sample_annotation,
              peptide_annotation = generated_peptide_annotation,
              protein_col = 'Gene', spike_ins = "BOVINE_A1ag",
              plot_title = 'Spike-in BOVINE protein peptides',
              color_by_batch = T, color_scheme = color_list[[batch_col]])
```

[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 [20], and most commonly, it is suggested that missing values need to be imputed [20 -21]. 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.



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.

3.4 Batch correction

Depending on the type of batch effect, 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, ComBat approach can be used. If there is continuous drift in the data, one has start from continuous drift correction.

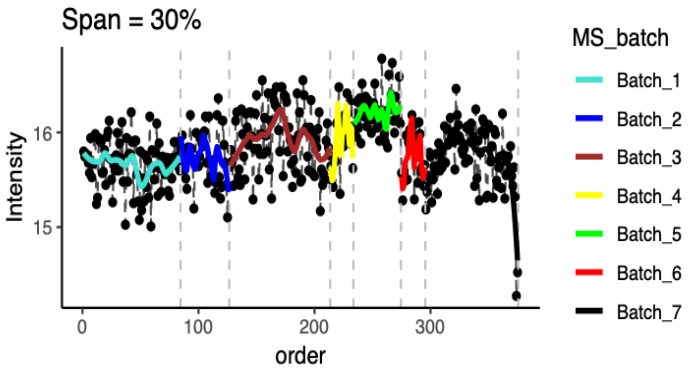
3.4.1 Continuous drift correction

Continuous drifts are specific for 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 discrete methods such as median-centering or ComBat.

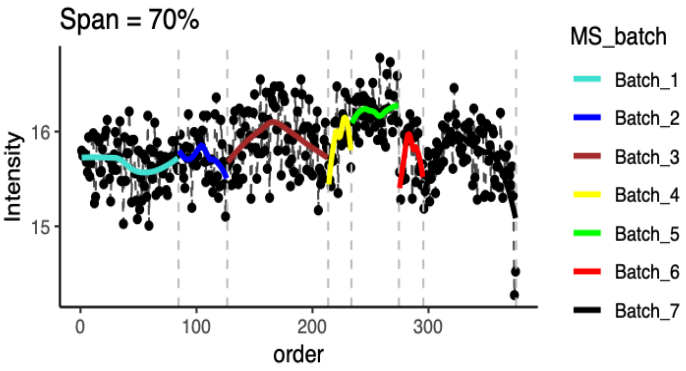
```
loess_fit <- adjust_batch_trend(quantile_normalized_matrix, example_sample_annotation)
loess_fit_matrix <- loess_fit$data_matrix
```

One important parameter in LOESS fitting is span, which determinesthe 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(quantile_normalized_matrix, example_sample_annotation, span = 0.3)

quantile_normalized_long <- matrix_to_long(quantile_normalized_matrix)
plot_with_fitting_curve(pep_name = "10231_QDVDVWLWQQEGSSK_2",
            df_long = quantile_normalized_long, example_sample_annotation,
            color_by_batch = T, color_scheme = color_list[[batch_col]],
            fit_df = loess_fit_30$fit_df, plot_title = "Span = 30%")
```




```
loess_fit_70 <- adjust_batch_trend(quantile_normalized_matrix, example_sample_annotation, span = 0.7)
plot_with_fitting_curve(pep_name = "10231_QDVDVWLWQQEGSSK_2",
            df_long = quantile_normalized_long, example_sample_annotation,
            color_by_batch = T, color_scheme = color_list[[batch_col]],
            fit_df = loess_fit_70$fit_df, plot_title = "Span = 70%")
```



Curve fitting is largely dependent on 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 peptdies, absolute and relative threshold (`abs.threshold` and `pct.threshold`) on number of missing values can be substitutes as paramters for `normalize_custom_fit()` is the absolute and relative threshold on the number of missing values for each peptide.

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

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

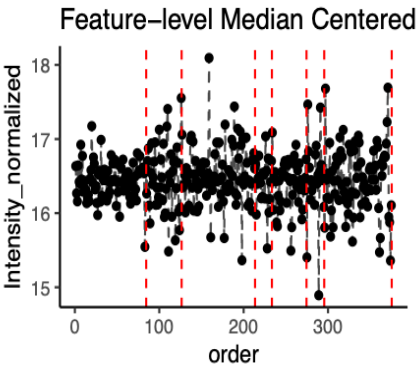
Currently, both batch correction procedures accept long format of the data:

```
loess_fit_df <- matrix_to_long(loess_fit_matrix)
```

3.4.2.1 Feature-level median centering

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

```
peptide_median_df <- center_peptide_batch_medians(loess_fit_df, example_sample_annotation)
plot_single_feature(pep_name = "46213_NVGVSFYADKPEVTQEQK_2", df_long = peptide_median_df,
            example_sample_annotation, color_by_col = NULL, measure_col = 'Intensity_normalized',
            plot_title = "Feature-level Median Centered")
```



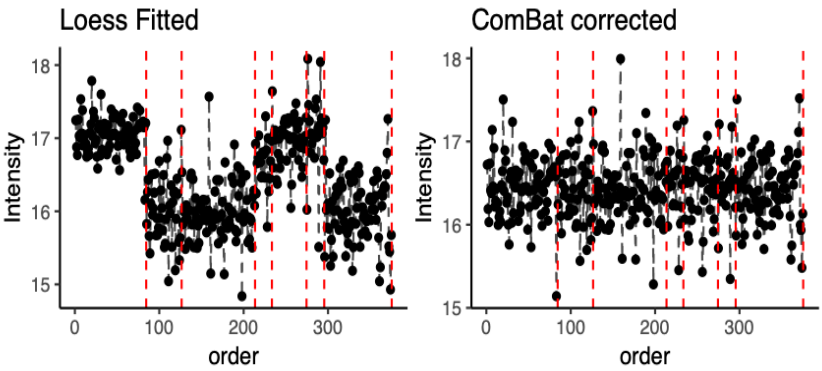
3.4.2.2 ComBat

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 [23]. The function `correct_with_ComBat()` can incorporate several covariates and make data comparable across batches.

```
comBat_matrix <- correct_with_ComBat(loess_fit_matrix, example_sample_annotation)
#> Standardizing Data across genes
```

To illustrate the correction we use the "46213_NVGVSFYADKPEVTQEQK_2" spike-in peptide.

```
combat_df <- matrix_to_long(comBat_matrix)
plot_single_feature (pep_name = "46213_NVGVSFYADKPEVTQEQK_2", loess_fit_df,
          example_sample_annotation, plot_title = "Loess Fitted", color_by_col = NULL)
plot_single_feature (pep_name = "46213_NVGVSFYADKPEVTQEQK_2", combat_df,
          example_sample_annotation, plot_title = "ComBat corrected", color_by_col = NULL)
```



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

3.4.3 Correct batch effects: universal function

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

```
batch_corrected_matrix <- correct_batch_effects(data_matrix = quantile_normalized_matrix,
                                  example_sample_annotation, discreteFunc = 'ComBat',
                                  abs.threshold = 5, pct.threshold = 0.20)
#> Standardizing Data across genes
```

3.5 Quality control on batch-corrected data matrix

In most cases, the batch effect 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, increased 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.

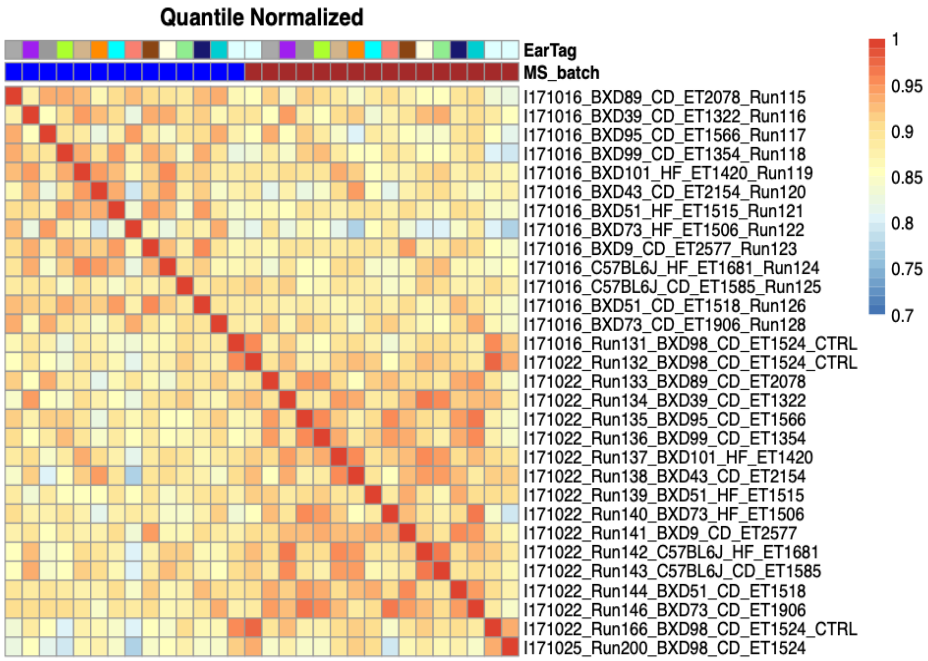
3.5.1 Heatmap of selected replicate samples

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 corrplot (`flavor = 'corrplot'`) or by "pretty heatmap", of "pheatmap"" (`flavor = 'pheatmap'`)

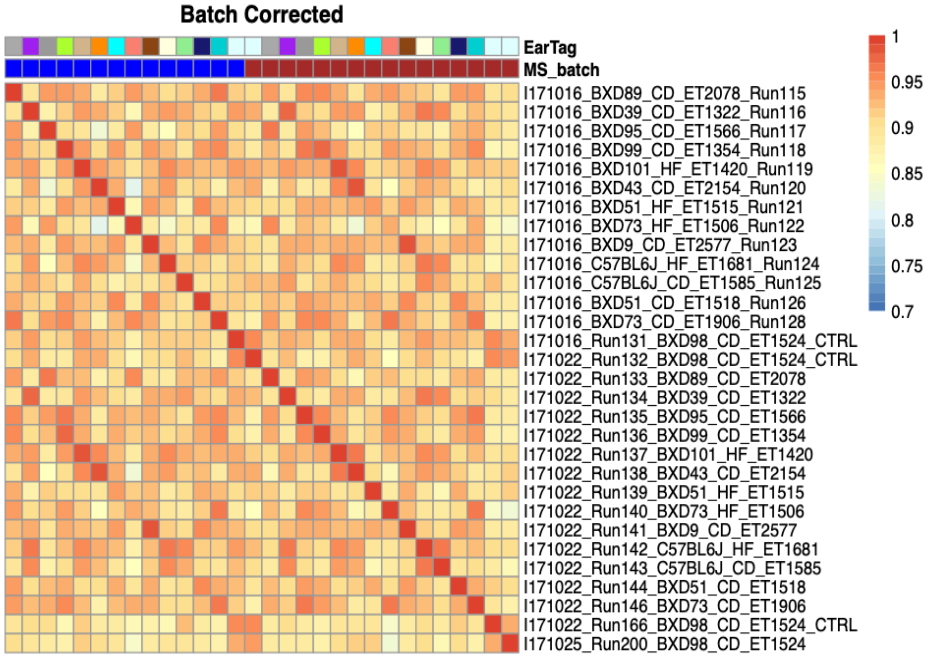
First, we specify, which samples we want to correlate

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

# Prepare color annotation
factors_to_show = c("MS_batch", "EarTag")
replicate_annotation <- example_sample_annotation %>%
  filter(MS_batch == 'Batch_2' | MS_batch == "Batch_3") %>%
  filter(EarTag %in% earTags) %>%
  remove_rownames %>%
  column_to_rownames(var="FullRunName") %>%
  select(factors_to_show) # Annotate MS_batch and EarTag on pheatmap

# sample ID of biological replicates
replicate_filenames = replicate_annotation %>%
  rownames()

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

```
# Plot the heatmap
plot_sample_corr_heatmap(quantile_normalized_matrix, samples_to_plot = replicate_filenames,
                         flavor = 'pheatmap', plot_title = 'Quantile Normalized',
                         annotation_colors = color_list[factors_to_show],
                         annotation_col = replicate_annotation,
                         color = heatmap_colors, breaks = breaksList,
                         cluster_rows=F, cluster_cols=F,
                         annotation_names_col = TRUE, annotation_legend = FALSE,
                         show_colnames = F)
```


Quantile Normalized
EarTag
MS_batch
I171016_BXD89_CD_ET2078_Run115
I171016_BXD39_CD_ET1322_Run116
I171016_BXD95_CD_ET1566_Run117
I171016_BXD99_CD_ET1354_Run118
I171016_BXD101_HF_ET1420_Run119
I171016_BXD43_CD_ET2154_Run120
I171016_BXD51_HF_ET1515_Run121
I171016_BXD73_HF_ET1506_Run122
I171016_BXD9_CD_ET2577_Run123
I171016_C57BL6J_HF_ET1681_Run124
I171016_C57BL6J_CD_ET1585_Run125
I171016_BXD51_CD_ET1518_Run126
I171016_BXD73_CD_ET1906_Run128
I171016_Run131_BXD98_CD_ET1524_CTRL
I171022_Run132_BXD98_CD_ET1524_CTRL
I171022_Run133_BXD89_CD_ET2078
I171022_Run134_BXD39_CD_ET1322
I171022_Run135_BXD95_CD_ET1566
I171022_Run136_BXD99_CD_ET1354
I171022_Run137_BXD101_HF_ET1420
I171022_Run138_BXD43_CD_ET2154
I171022_Run139_BXD51_HF_ET1515
I171022_Run140_BXD73_HF_ET1506
I171022_Run141_BXD9_CD_ET2577
I171022_Run142_C57BL6J_HF_ET1681
I171022_Run143_C57BL6J_CD_ET1585
I171022_Run144_BXD51_CD_ET1518
I171022_Run146_BXD73_CD_ET1906
I171022_Run166_BXD98_CD_ET1524_CTRL
I171025_Run200_BXD98_CD_ET1524
1
0.95
0.9
0.85
0.8
0.75
0.7

```
plot_sample_corr_heatmap(batch_corrected_matrix, samples_to_plot = replicate_filenames,
                         flavor = 'pheatmap', plot_title = 'Batch Corrected',
                         annotation_colors = color_list[factors_to_show],
                         annotation_col = replicate_annotation,
                         color = heatmap_colors, breaks = breaksList,
                         cluster_rows=F, cluster_cols=F,
                         annotation_names_col = TRUE, annotation_legend = FALSE,
                         show_colnames = F)
```



Before the correction, samples from one batch correlate higher and often 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.

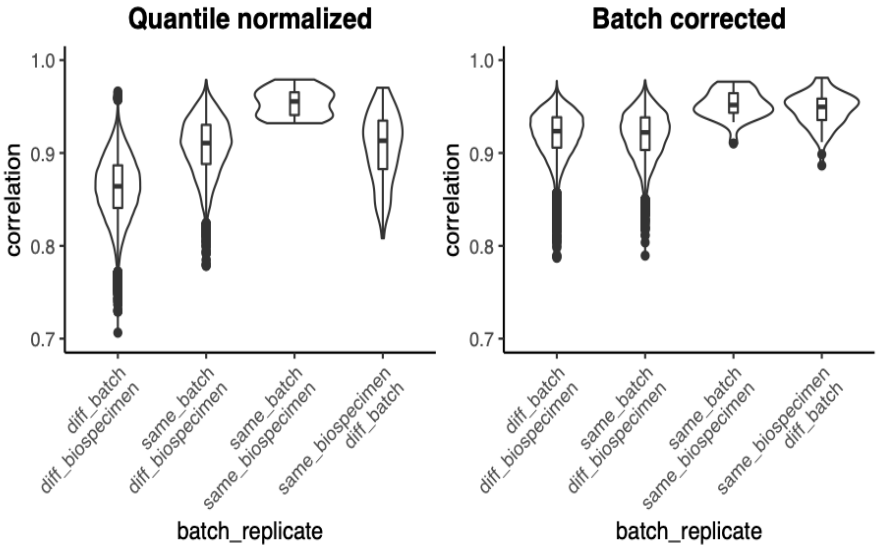
3.5.2 Correlation distribution of samples

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()` plots correlation distribution between biological replicates and non-replicates in the same or differnet 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 in 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).

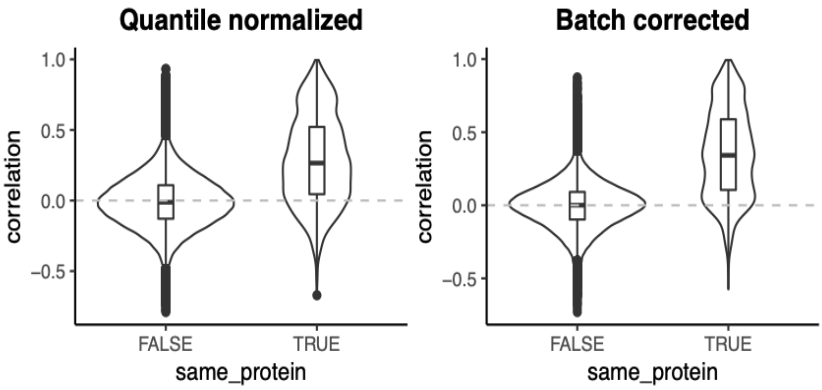
```
sample_cor_norm <- plot_sample_corr_distribution(quantile_normalized_matrix,
                                                 example_sample_annotation,
                                                 batch_col = 'MS_batch',
                                                 biospecimen_id_col = "EarTag",
                                                 plot_title = 'Quantile normalized',
                                                 plot_param = 'batch_replicate')
sample_cor_norm + theme(axis.text.x = element_text(angle = 45, hjust = 1)) + ylim(0.7,1)

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')
sample_cor_batchCor + theme(axis.text.x = element_text(angle = 45, hjust = 1)) + ylim(0.7, 1)
```



3.5.3 Correlation distribution of peptides 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 if the correlation between the same proteins increases while that of different proteins stays the same.

```
peptide_cor_norm <- plot_peptide_corr_distribution(quantile_normalized_matrix,
          generated_peptide_annotation, protein_col = 'Gene', plot_title = 'Quantile normalized')
peptide_cor_norm + geom_hline(yintercept=0, linetype="dashed", color = "grey")

peptide_cor_batchCor <- plot_peptide_corr_distribution(batch_corrected_matrix,
          generated_peptide_annotation, protein_col = 'Gene', plot_title = 'Batch corrected')
peptide_cor_batchCor + geom_hline(yintercept=0, linetype="dashed", color = "grey")
```



4 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] T. Sajic, Y. Liu, E. Arvaniti et al. «Similarities and Differences of Blood N-Glycoproteins in Five Solid Carcinomas at Localized Clinical Stage Analyzed by SWATH-MS». Cell Reports 23:9 (2018), 2819– 2831.e5.

[5] H. Okada, H. A. Ebhardt, S. C. Vonesch, R. Aebersold, and E. Hafen. «Proteome-wide association studies identify biochemical modules associated with a wing-size phenotype in Drosophila melanogaster». Nature Communications 7 (2016), p. 12649.

[6] J. T. Leek, R. B. Scharpf, H. C. H. C. Bravo et al. «Tackling the widespread and critical impact of batch effects in high-throughput data». Nat Rev Genet 11:10 (2010), pp. 733–739.

[7] H. S. Parker and J. T. Leek. «The practical effect of batch on genomic prediction». Statistical applications in genetics and molecular biology 11:3 (2012), Article 10.

[8] A. L. Oberg and O. Vitek. «Statistical design of quantitative mass spectrometry-based proteomic experiments». Journal of Proteome Research 8:5 (2009), pp. 2144–2156.

[9] J. Hu, K. R. Coombes, J. S. Morris, and K. A. Baggerly. «The importance of experimental design in proteomic mass spectrometry experiments: Some cautionary tales». Brie ngs in Functional Genomics and Proteomics 3:4 (2005), pp. 322–331.

[10] Y. Gilad and O. Mizrahi-Man. «A reanalysis of mouse ENCODE comparative gene expression data». F1000Research 4 (2015).

[11] M.-A. A. Dillies, A. Rau, J. Aubert et al. «A comprehensive evaluation of normalization methods for Illumina high-throughput RNA sequencing data analysis». Brief Bioinform 14:6 (2013), pp. 671–683.

[12] J. Luo, M. Schumacher, A. Scherer et al. «A comparison of batch effect removal methods for enhancement of prediction performance using MAQC-II microarray gene expression data». Pharmacogenomics Journal 10:4 (2010), pp. 278–291.

[13] W. E. Johnson, C. Li, and A. Rabinovic. «Adjusting batch effects in microarray expression data using empirical Bayes methods». Biostatistics 8:1 (2007), pp. 118–127.

[14] A. H. Sims, G. J. Smethurst, Y. Hey et al. «The removal of multiplicative, systematic bias allows integration of breast cancer gene expression datasets - improving meta-analysis and prediction of prognosis». BMC Med Genomics 1 (2008), p. 42.

[15] J. T. Leek and J. D. Storey. «Capturing heterogeneity in gene expression studies by surrogate variable analysis». PLoS Genet 3:9 (2007), pp. 1724–1735.

[16] M. Benito, J. Parker, Q. Du et al. «Adjustment of systematic microarray data biases.» Bioinformatics (Oxford, England) 20:1 (2004), pp. 105–14.

[17] C. Chen, K. Grennan, J. Badner et al. «Removing batch effects in analysis of expression microarray data: an evaluation of six batch adjustment methods». PLoS One 6:2 (2011), e17238.

[18] B. M. Bolstad, R. A. Irizarry, M. Astrand, and T. P. Speed. «A comparison of normalization methods for high density oligonucleotide array data based on variance and bias.» Bioinformatics 19:2 (2003), pp. 185–93.

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

[20] 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.

[21] 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.

[22] 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.

[23] 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.