ProBatchFeatures

Yuliya Burankova
Institute for Computational Systems Biology, University of Hamburg, Germany

2025-09-25

Abstract

This vignette demonstrates how to do and evaluate preprocessing pipelines with `ProBatchFeatures`, the `QFeatures`-based extension in `proBatch`.

Contents

1 Overview **1**

2 Setup **2**
2.1 Load required packages . . . 2
2.2 Load and prepare example dataset . . . 2

3 Construct a `ProBatchFeatures` object **2**

4 Processing pipeline with diagnostics **3**
4.1 Step 0 - raw data exploration . . . 3
4.2 Step 1 - filter features with to many missing values . . . 4
4.3 Step 2 - log2~transformation . . . 7
4.3.1 Principal component analysis on log-transformed assays . . . 10
4.4 Step 3 - median normalization . . . 11
4.5 Step 4 - batch-effect correction . . . 13
4.6 Step 5 - assess processed assays . . . 15
4.6.1 Principal component analysis . . . 15
4.6.2 Hierarchical clustering . . . 17
4.6.3 Principal Variance Component Analysis (PVCA) . . . 18

5 Inspect operation log and lineage **19**
5.1 Extract matrices from `pbf` object . . . 20

6 Session info **20**

1 Overview

`ProBatchFeatures` is an extension around **QFeatures** (1) that stores each processing stage (for example, raw → log → normalization → BEC → . . .) as a new **assay**, and logs every step. It keeps data compatible with **SummarizedExperiment**/**QFeatures** functions, and enables quick benchmarking of chained pipelines.

This vignette shows how to:

1. Create a `ProBatchFeatures` object from matrix-formatted quantitation tables.
2. Diagnose raw assays.

3. Chain and log standard preprocessing steps (NA filtering → log2 transform → normalization → batch-effect correction).
4. Recover processed assays and the recorded pipeline for benchmarking or reuse.

2 Setup

2.1 Load required packages

We rely on `dplyr`, `tibble`, and `ggplot2` (alongside `proBatch`) for data manipulation and plotting.

```
library(dplyr)
library(tibble)
library(ggplot2)
```

2.2 Load and prepare example dataset

Here, as example, we will use the E.coli dataset, described in the FedProt paper (2):

```
# library(proBatch)
devtools::load_all("../")
data("example_ecoli_data", package = "proBatch")
```

The dataset contains data from 5 centers, which were analysed separately. Additionally, to simplify the example here, the data were merged from all centers into one dataset and a smaller part of this dataset is used (total 400 protein groups and corresponding peptides). We need only this merged data here:

```
# Extract data
all_metadata <- example_ecoli_data$all_metadata
all_precursors <- example_ecoli_data$all_precursors
all_protein_groups <- example_ecoli_data$all_protein_groups
all_precursor_pg_match <- example_ecoli_data$all_precursor_pg_match

# Keep only essential
rm(example_ecoli_data)
```

3 Construct a `ProBatchFeatures` object

The dataset consists of two levels. It is possible to work with more levels similar to QFEatures, but here we use only peptides and protein groups levels. The `all_precursor_pg_match` data frame stores links between them. We first use the `ProBatchFeatures()` constructor to create an object for the peptide-level data:

```
# Build directly from a wide (features x samples) matrix
pbf <- ProBatchFeatures(
    data_matrix = all_precursors, # matrix of peptide intensities (features x samples)
    sample_annotation = all_metadata, # data.frame of sample metadata
    sample_id_col = "Run", # column in metadata that matches sample IDs
    level = "peptide" # label this assay level as "peptide"
)

pbf # show basic info about the ProBatchFeatures object
#> An instance of class ProBatchFeatures containing 1 assays:
#>  [1] peptide::raw: SummarizedExperiment with 11131 rows and 118 columns
#>   Processing chain: unprocessed data (raw)
```

Assay naming convention is "::", e.g., "peptide::raw", "protein::median_on_log".

Next, we add the protein-level data as a second assay in the same ProBatchFeatures instance. Internally, it is done by creating a SummarizedExperiment for the protein data and then adding it to pbf object via QFeatures functionality. It's important to use the same sample annotation for the new assay to ensure column alignment:

```
# Add proteins as a new level and link via mapping
#    all_precursor_pg_match has columns: "Precursor.Id", "Protein.Ids"
pbf <- pb_add_level(
    object = pbf,
    from = "peptide::raw",
    new_matrix = all_protein_groups,
    to_level = "protein", # will name "protein::raw" by default
    mapping_df = all_precursor_pg_match,
    from_id = "Precursor.Id",
    to_id = "Protein.Ids",
    map_strategy = "as_is"
)

pbf
#> An instance of class ProBatchFeatures containing 2 assays:
#>  [1] peptide::raw: SummarizedExperiment with 11131 rows and 118 columns
#>  [2] protein::raw: SummarizedExperiment with 400 rows and 118 columns
#>   Processing chain:
#>   [1] add_level(protein)_byVar
#>   Steps logged: 1 (see get_operation_log())
```

If a precursor/peptide maps to multiple protein groups, parameter `map_strategy` is used to determine how to resolve multiple to-ids per from-id. Can be 'first' or 'longest', and 'as_is' expects one-to-one in mapping. First two rules ("first" or "longest") yield a single ProteinID per peptide for the linking variable. Here we know that there are no duplicates, so 'as_is' used.

```
# Check
validObject(pbf) # should be TRUE
#> [1] TRUE
assayLink(pbf, "protein::raw") # verify link summary
#> AssayLink for assay <protein::raw>
#> [from:peptide::raw|fcol:ProteinID|hits:11131]

# Keep only essential
rm(all_metadata, all_precursor_pg_match, all_precursors, all_protein_groups)
```

4 Processing pipeline with diagnostics

Now that we have the data loaded into a `ProBatchFeatures` object with peptide and protein assays, we can demonstrate an example of typical processing pipeline. This pipeline will include filtering out low-quality features, $\log_2$transformation, median normalization, and batch effect correction.

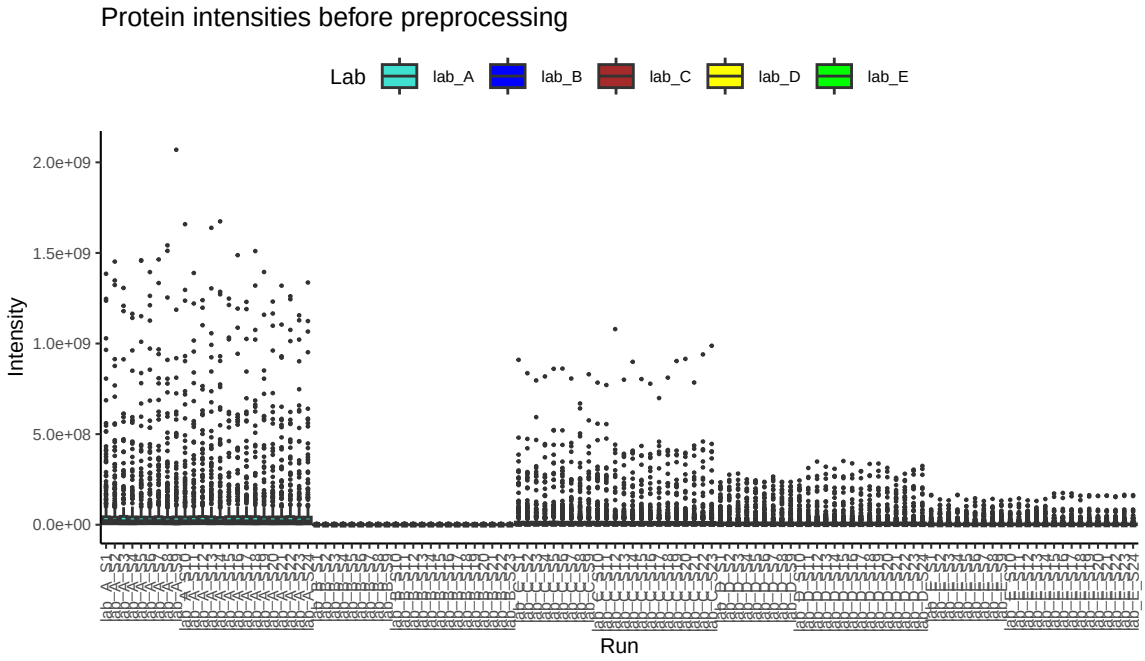
4.1 Step 0 - raw data exploration

Following the recommendations in the main `proBatch` vignette, we start with baseline diagnostics on the raw assays. These plots help confirm overall signal intensity, highlight obvious batch-specific shifts, and provide a reference for later comparisons.

```
# First, it is a good idea to make color map to use in all plots
color_scheme <- sample_annotation_to_colors(
```

```
    pbf,
    sample_id_col = "Run",
    factor_columns = c("Lab", "Condition"),
    numeric_columns = NULL
)
```

```
plot_boxplot(
    pbf,
    sample_id_col = "Run",
    order_col = "Run",
    batch_col = "Lab",
    color_by_batch = TRUE,
    color_scheme = color_scheme,
    base_size = 8,
    pbf_name = "protein::raw",
    plot_title = "Protein intensities before preprocessing"
)
```



4.2 Step 1 - filter features with to many missing values

Before normalisation, we review the proportion of missing values. Features measured in only a small subset of samples are typically unstable and can mask systematic effects in downstream diagnostics.

Function `pb_nNA()` summarises missingness per assay:

```
# nNA returns a DataFrame with counts of NAs per feature/sample
na_counts <- pb_nNA(pbf)

# na_counts contains info about total number of NAs and % in the data:
na_counts[["nNA"]] %>% as.tibble()
#> # A tibble: 2 x 3
#>   assay           nNA   pNA
#>   <chr>         <int> <dbl>
```

```
#> 1 peptide::raw 694976 0.529
#> 2 protein::raw   9163 0.194
```

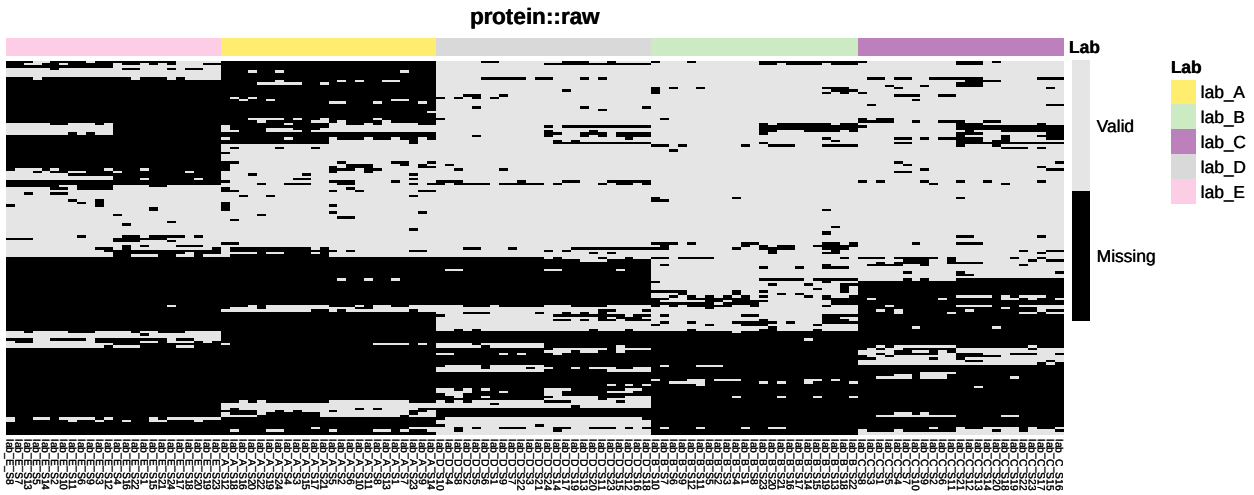
Here nNA - number of missing values in the assay and pNA is its proportion. It is possible to inspect the per-feature and per-sample breakdown for each assay:

```
# check NAs per sample:
head(na_counts[["peptide::raw"]]$nNAcols) %>% as.tibble()
#> # A tibble: 6 x 4
#>   assay        name        nNA   pNA
#>   <chr>        <chr>     <int> <dbl>
#> 1 peptide::raw lab_A_S1   6379 0.573
#> 2 peptide::raw lab_A_S2   6362 0.572
#> 3 peptide::raw lab_A_S3   6378 0.573
#> 4 peptide::raw lab_A_S4   6361 0.571
#> 5 peptide::raw lab_A_S5   6396 0.575
#> 6 peptide::raw lab_A_S6   6584 0.592
```

For each assay the next tables availabe: `print(names(na_counts[["peptide::raw"]]))`.

Visualisations help highlight batch-specific sparsity patterns:

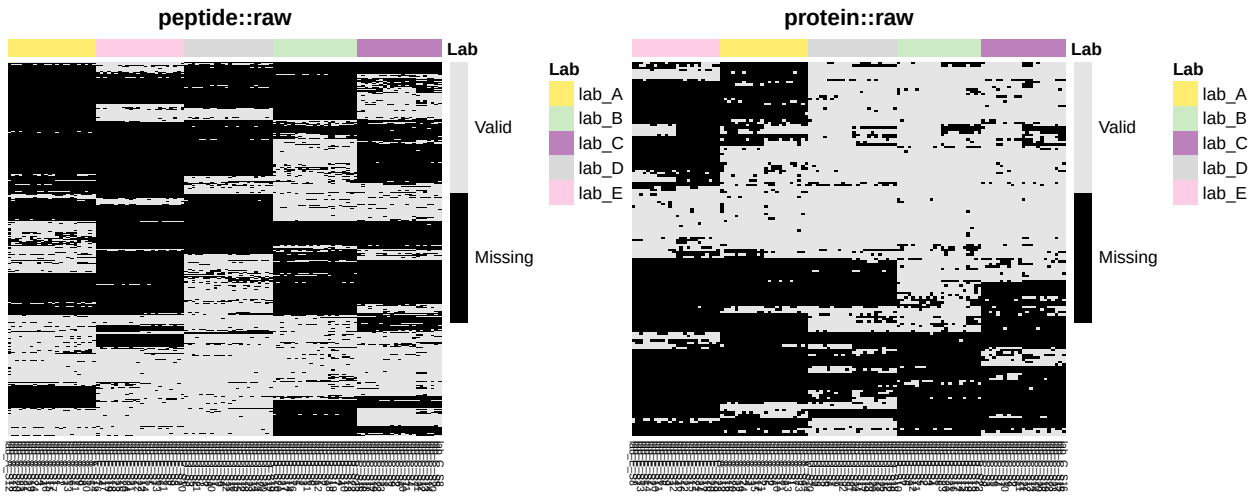
```
plot_NA_heatmap(
    pbf,    # by default the last created assay
    show_rownames = FALSE, show_row_dend = FALSE,
    color_by = "Lab"
)
```



In this plot only features that contains at least one missing value are plotted, all features can be used with `drop_complete=F` parameter. If the number of features or samples is more than 5000, the random subset of 5000 rows/columns are used by default. All dataset can be used for plots using `use_subset = T` parameter.

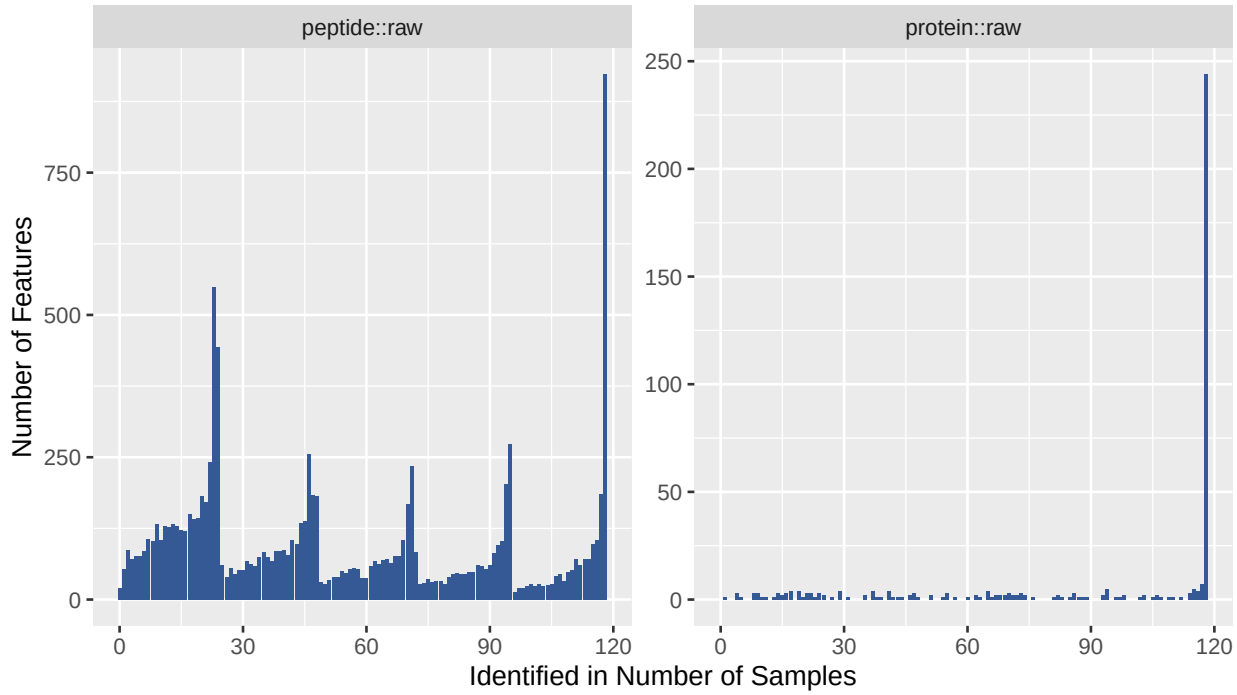
Also, it is possible to plot multiple assays in one plot:

```
plot_NA_heatmap(
    pbf,
    pbf_name = c("peptide::raw", "protein::raw"),
    show_rownames = FALSE, show_row_dend = FALSE,
    color_by = "Lab"
)
```



The peptide / protein identification overlap can be plotted:

```
plot_NA_frequency(
    pbf,
    pbf_name = c("peptide::raw", "protein::raw"),
    show_percent = FALSE
)
```



On the last plot, especially for peptides, we can see a clear batch pattern, as in our data we have 19–20 samples per lab.

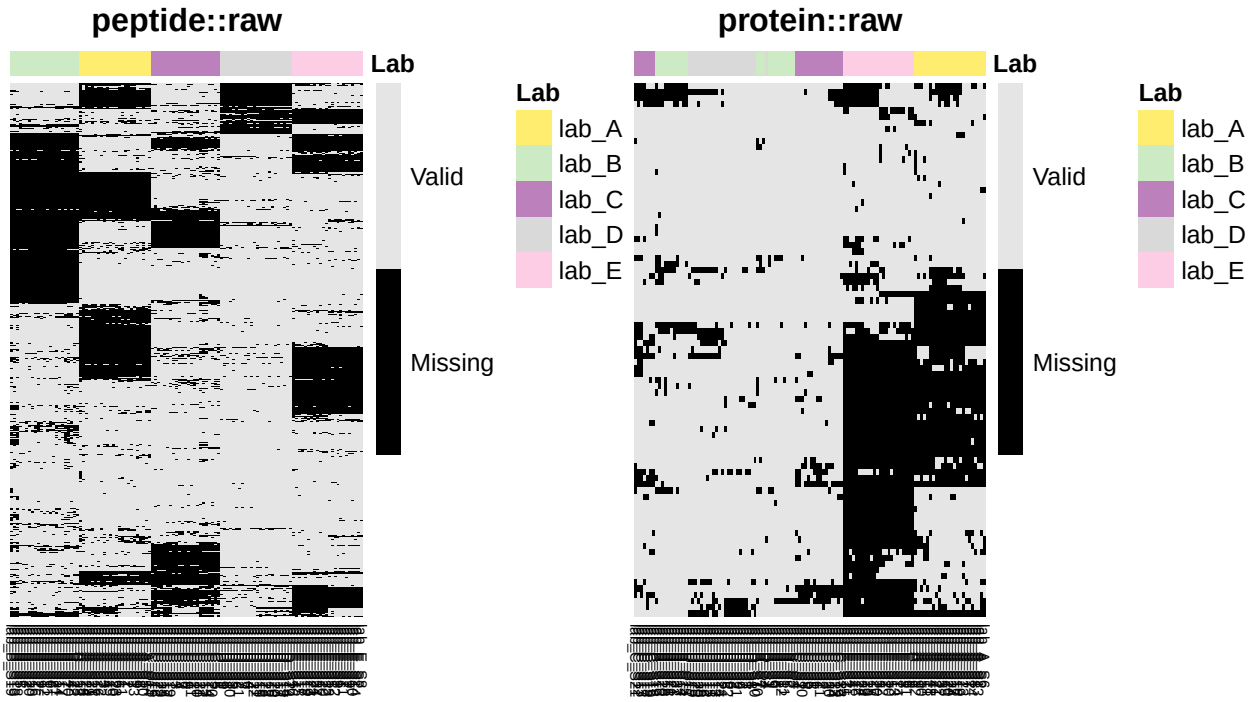
Here we apply a simple missing-data filter: we require each feature to be quantified in at least 40% of the samples (tune this threshold for your study):

```
pbf <- pb_filterNA(
    pbf,  # without specifying, the filter will be applied to all assays in the object
    inplace = TRUE,  # if false, filtered matrix will be saved as a new assay.
```

```
    pNA = 0.6   # the maximal proportion of missing values per feature
)
pbf
#> An instance of class ProBatchFeatures containing 2 assays:
#>  [1] peptide::raw: SummarizedExperiment with 5387 rows and 118 columns
#>  [2] protein::raw: SummarizedExperiment with 331 rows and 118 columns
#>   Processing chain:
#>    [1] add_level(protein)_byVar filterNA                 filterNA
#>   Steps logged: 3 (see get_operation_log())
```

After filtering, the `ProBatchFeatures` object stores fewer features, which sharpens downstream diagnostics.

```
plot_NA_heatmap(
    pbf,
    pbf_name = c("peptide::raw", "protein::raw"),
    show_rownames = FALSE, show_row_dend = FALSE,
    color_by = "Lab",    # currently only one level is supported
    border_color = NA    #pheatmap parameter to remove cell borders
)
```



The remaining missingness is still structured by lab, which justifies the normalisation and batch-effect correction steps applied next.

4.3 Step 2 - log2~transformation

Proteomics data is typically log-transformed to stabilize variance and make the data distribution more symmetric, which benefits many statistical methods. We'll apply a log2 transformation to both peptides and protein data.

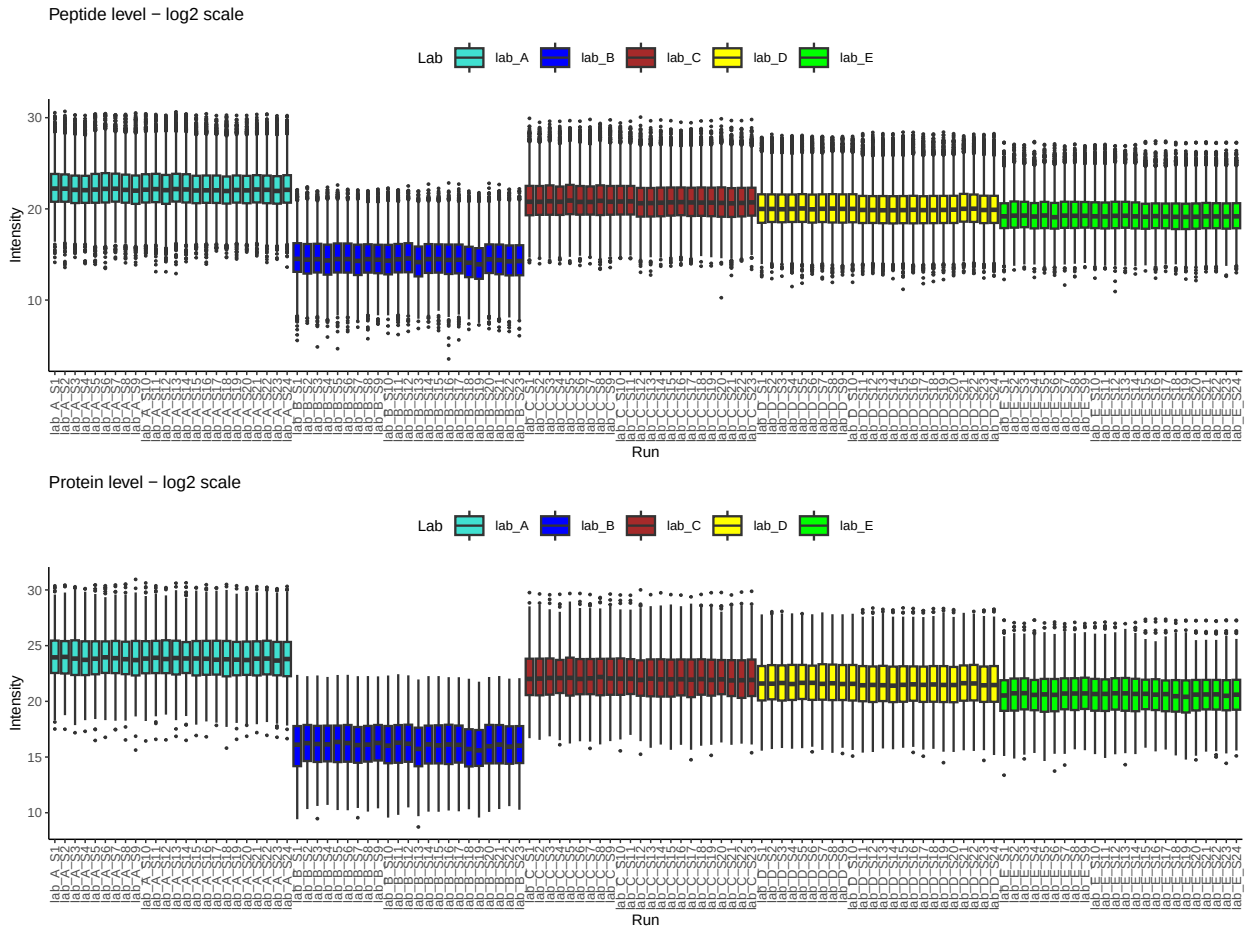
```
pbf <- log_transform_dm(
    pbf,
    log_base = 2, offset = 1,
```

```
    pbf_name = "protein::raw"
)

pbf <- log_transform_dm(
    pbf,
    log_base = 2, offset = 1,
    pbf_name = "peptide::raw"
)

pbf
#> An instance of class ProBatchFeatures containing 2 assays:
#>  [1] peptide::raw: SummarizedExperiment with 5387 rows and 118 columns
#>  [2] protein::raw: SummarizedExperiment with 331 rows and 118 columns
#>   Processing chain:
#>   [1] add_level(protein)_byVar filterNA                filterNA                ; [4] log2
#>    - peptide: log2_on_raw
#>    - protein: log2_on_raw
#>   Steps logged: 5 (see get_operation_log())
```

Printing `pbf` now shows additional assays (`peptide::log2_on_raw` and `protein::log2_on_raw`). If the assay was transformed using "fast to recompute" step (for exmaple, log-transformation or median normalization), the transformed df does not saved in the list of assay, but re-computed each time for plotting. It is possible to force storing this steps using `store_fast_steps = T` parameter.

Let's visualise effect of log-transformation by plotting box-plots:

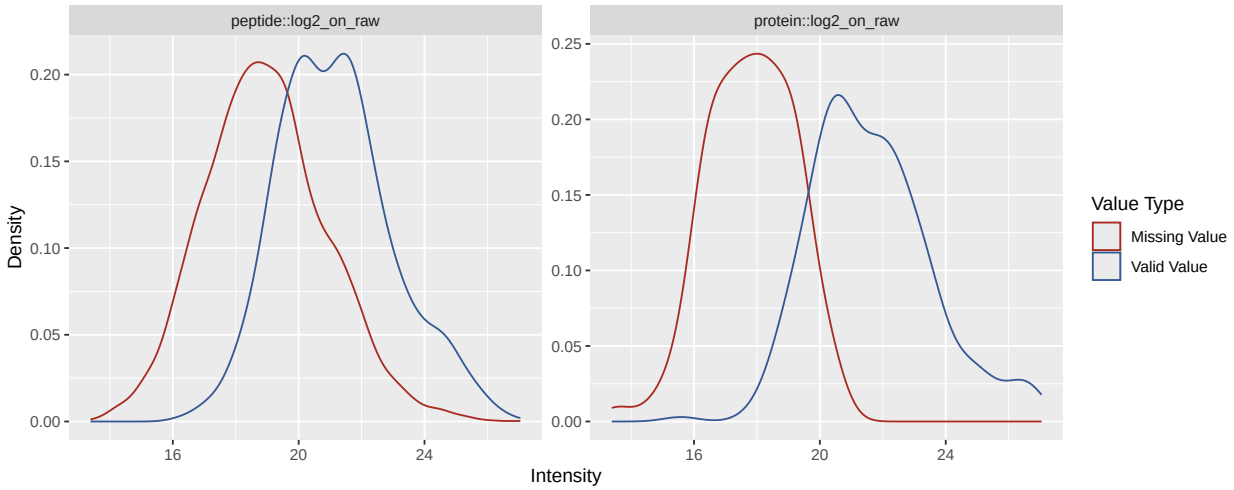
```
plot_boxplot(
    pbf,
    pbf_name = c("peptide::log2_on_raw", "protein::log2_on_raw"),   # plot multiple on one
    sample_id_col = "Run",
    order_col = "Run",
    batch_col = "Lab",
    color_by_batch = TRUE,
    color_scheme = color_scheme,
    base_size = 7,
    plot_ncol = 1,   # how many columns use for plotting multy-assay plot
    plot_title = c(     # title for each assay
      "Peptide level - log2 scale",
      "Protein level - log2 scale"
    )
)
```



The comparison confirms that log2 stabilises the variance for both peptides and proteins, yet the cross-lab shifts in median intensity persist and will need to be addressed downstream.

The intensity distribution of proteins / peptides with and without NAs can be plotted:

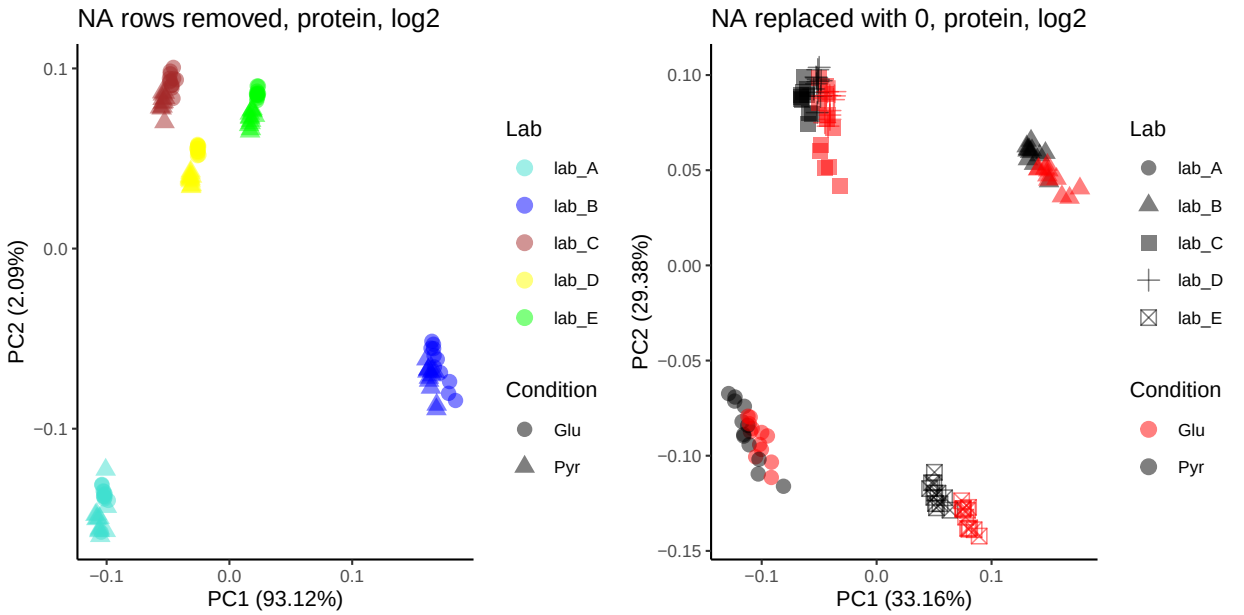
```
plot_NA_density(
    pbf,
    pbf_name = c("peptide::log2_on_raw", "protein::log2_on_raw")
)
```



4.3.1 Principal component analysis on log-transformed assays

Next, we'll perform a Principal Component Analysis (PCA) to get a high-level overview of the sample clustering. We expect to see a strong batch effect, where samples cluster by their center of origin ("Lab") rather than their biological condition ("Condition").

```
pca1 <- plot_PCA(
    pbf,
    pbf_name = "protein::log2_on_raw",
    sample_id_col = "Run",
    color_scheme = color_scheme,
    color_by = "Lab",
    shape_by = "Condition",
    fill_the_missing = NULL,  # remove all rows with missing values. Can be also "remove"
    plot_title = "NA rows removed, protein, log2",
    base_size = 10, point_size = 3, point_alpha = 0.5
)

pca2 <- plot_PCA(
    pbf,
    pbf_name = "protein::log2_on_raw",
    sample_id_col = "Run",
    color_scheme = color_scheme,
    color_by = "Condition",
    shape_by = "Lab",
    fill_the_missing = 0, # default value is -1
    plot_title = "NA replaced with 0, protein, log2",
    base_size = 10, point_size = 3, point_alpha = 0.5
)

gridExtra::grid.arrange(pca1, pca2, ncol = 2, nrow = 1)
```



As anticipated, the PCA plot clearly shows that the primary source of variation in the raw data is the Center, confirming a strong batch effect that we will need to address.

4.4 Step 3 - median normalization

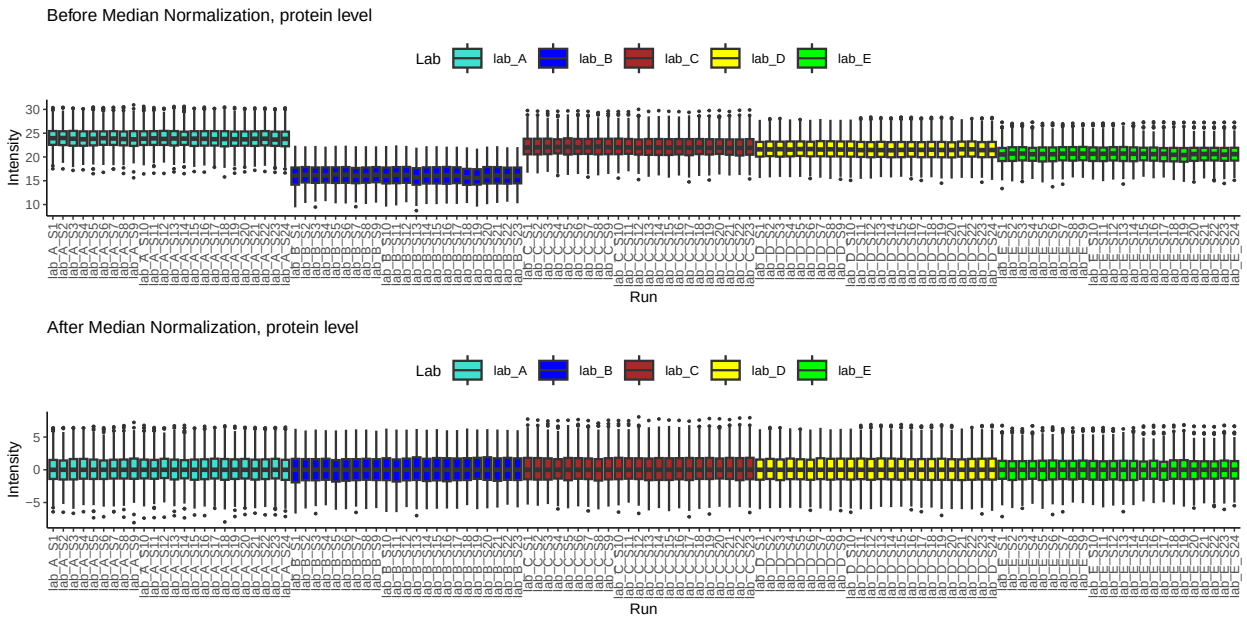
To make the samples more comparable, we will try to apply median normalization. This simple yet effective method aligns the distributions of intensities by subtracting the median intensity from each sample.

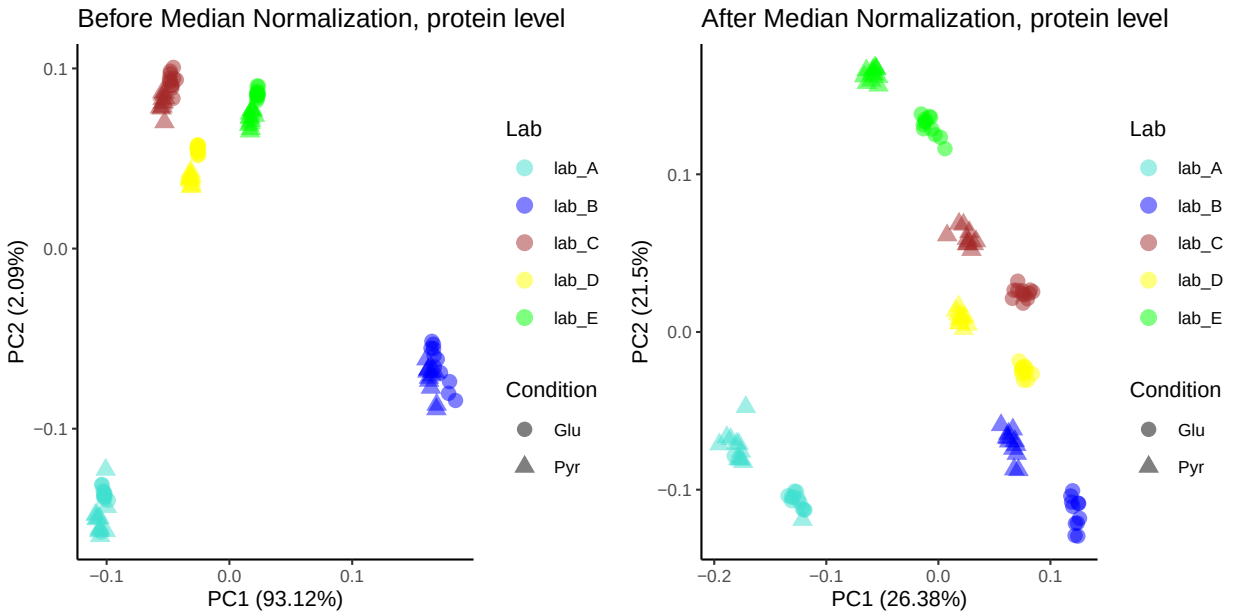
```
pbf <- pb_transform(
    object = pbf,
    from = "peptide::log2_on_raw",
    steps = "medianNorm"
)

pbf <- pb_transform(
    object = pbf,
    from = "protein::log2_on_raw",
    steps = "medianNorm"
)

pbf
#> An instance of class ProBatchFeatures containing 2 assays:
#>  [1] peptide::raw: SummarizedExperiment with 5387 rows and 118 columns
#>  [2] protein::raw: SummarizedExperiment with 331 rows and 118 columns
#>   Processing chain:
#>    [1] add_level(protein)_byVar filterNA                filterNA                ; [4] log2
#>     - peptide: log2_on_raw, medianNorm_on_log2_on_raw
#>     - protein: log2_on_raw, medianNorm_on_log2_on_raw
#>   Steps logged: 7 (see get_operation_log())
```

`pb_transform()` appends `peptide::medianNorm_on_log2_on_raw` and `protein::medianNorm_on_log2_on_raw`.

We can compare these assays with their log2 inputs using the same boxplot diagnostic.

```
# boxplot
plot_boxplot(
    pbf,
    sample_id_col = "Run",
    order_col = "Run",
    batch_col = "Lab",
    color_by_batch = TRUE,
    color_scheme = color_scheme,
    base_size = 7,
    pbf_name = c("protein::log2_on_raw", "protein::medianNorm_on_log2_on_raw"),
    plot_ncol = 1,
    plot_title = c(
        "Before Median Normalization, protein level",
        "After Median Normalization, protein level"
    )
)

# plot PCA plot
plot_PCA(
    pbf,
    pbf_name = c("protein::log2_on_raw", "protein::medianNorm_on_log2_on_raw"),
    sample_id_col = "Run",
    color_scheme = color_scheme,
    color_by = "Lab",
    shape_by = "Condition",
    fill_the_missing = NULL,  # remove all rows with missing values. Can be also "remove"
    plot_title = c(
      "Before Median Normalization, protein level",
      "After Median Normalization, protein level"
    ),
    base_size = 10, point_size = 3, point_alpha = 0.5
)
```





The median normalization helps, but batch effect is still visible on the PCA plot and needs to be corrected.

4.5 Step 4 - batch-effect correction

The package contains two methods for batch effects correction for ProBatchFeatures class: ComBat from sva and `removeBatchEffect()` from limma. Both of them can be accessed via the same function `pb_transform()`.

```
# sample annotations used by both correction methods
sa <- colData(pbf) %>% as.data.frame()
```

Apply `removeBatchEffect()` from limma to log-transformed data and normalized data:

```
# limma::removeBatchEffect wrapped via pb_transform()
pbf <- pb_transform(
    pbf,
    from = "protein::log2_on_raw",
    steps = "limmaRBE",
    params_list = list(list(
        sample_annotation = sa,
        batch_col = "Lab",
        covariates_cols = c("Condition"),
        fill_the_missing = FALSE
    ))
)

pbf <- pb_transform(
    pbf,
    from = "protein::medianNorm_on_log2_on_raw",
    steps = "limmaRBE",
    params_list = list(list(
        sample_annotation = sa,
        batch_col = "Lab",
        covariates_cols = c("Condition"),
        fill_the_missing = FALSE
    ))
```

```
)
```

As batch effect correction does not listed as a `fast step`, two new assay was added under "protein" level: [3] protein::limmaRBE_on_log2_on_raw and [4] protein::limmaRBE_on_medianNorm_on_log2_on_raw.

```
print(pbf)
#> An instance of class ProBatchFeatures containing 4 assays:
#>  [1] peptide::raw: SummarizedExperiment with 5387 rows and 118 columns
#>  [2] protein::raw: SummarizedExperiment with 331 rows and 118 columns
#>  [3] protein::limmaRBE_on_log2_on_raw: SummarizedExperiment with 331 rows and 118 columns
#>  [4] protein::limmaRBE_on_medianNorm_on_log2_on_raw: SummarizedExperiment with 331 rows and 118 colu
#>   Processing chain:
#>   [1] add_level(protein)_byVar filterNA                filterNA                ; [4] log2
#>    - peptide: log2_on_raw, medianNorm_on_log2_on_raw
#>    - protein: log2_on_raw, limmaRBE_on_log2_on_raw, medianNorm_on_log2_on_raw, limmaRBE_on_medianNor
#>   Steps logged: 9 (see get_operation_log())
```

Do the same using combat-based batch effect correction:

```
# sva::ComBat wrapped via pb_transform()
pbf <- pb_transform(
    pbf,
    from = "protein::log2_on_raw",
    steps = "combat",
    params_list = list(list(
        sample_annotation = sa,
        batch_col = "Lab",
        sample_id_col = "Run",
        par.prior = TRUE,
        fill_the_missing = "remove"
    ))
)

pbf <- pb_transform(
    pbf,
    from = "protein::medianNorm_on_log2_on_raw",
    steps = "combat",
    params_list = list(list(
        sample_annotation = sa,
        batch_col = "Lab",
        sample_id_col = "Run",
        par.prior = TRUE,
        fill_the_missing = "remove"
    ))
)
```

The combat method from sva cannot work with data containing missing values. Bacause of that, here we removed features with them. It is also possible to impute the data with used-defined value using `fill_the_missing` parameter (default is -1).

```
print(pbf)
#> An instance of class ProBatchFeatures containing 6 assays:
#>  [1] peptide::raw: SummarizedExperiment with 5387 rows and 118 columns
#>  [2] protein::raw: SummarizedExperiment with 331 rows and 118 columns
#>  [3] protein::limmaRBE_on_log2_on_raw: SummarizedExperiment with 331 rows and 118 columns
#>  [4] protein::limmaRBE_on_medianNorm_on_log2_on_raw: SummarizedExperiment with 331 rows and 118 colu
```

```
#>  [5] protein::combat_on_log2_on_raw: SummarizedExperiment with 244 rows and 118 columns
#>  [6] protein::combat_on_medianNorm_on_log2_on_raw: SummarizedExperiment with 244 rows and 118 columns
#>   Processing chain:
#>    [1] add_level(protein)_byVar filterNA                filterNA                ;  [4] log2
#>    - peptide: log2_on_raw, medianNorm_on_log2_on_raw
#>    - protein: log2_on_raw, combat_on_log2_on_raw, limmaRBE_on_log2_on_raw, medianNorm_on_log2_on_raw
#>   Steps logged: 11 (see get_operation_log())
```

Similarly, two new assay was added under "protein" level: - [5] protein::combat_on_log2_on_raw: ... with 244 rows - [6] protein::combat_on_medianNorm_on_log2_on_raw: ... with 244 rows.

And as we removed features with missing values, the assays contain less protein groups compeating with the raw filtered data (331 rows).

4.6 Step 5 - assess processed assays

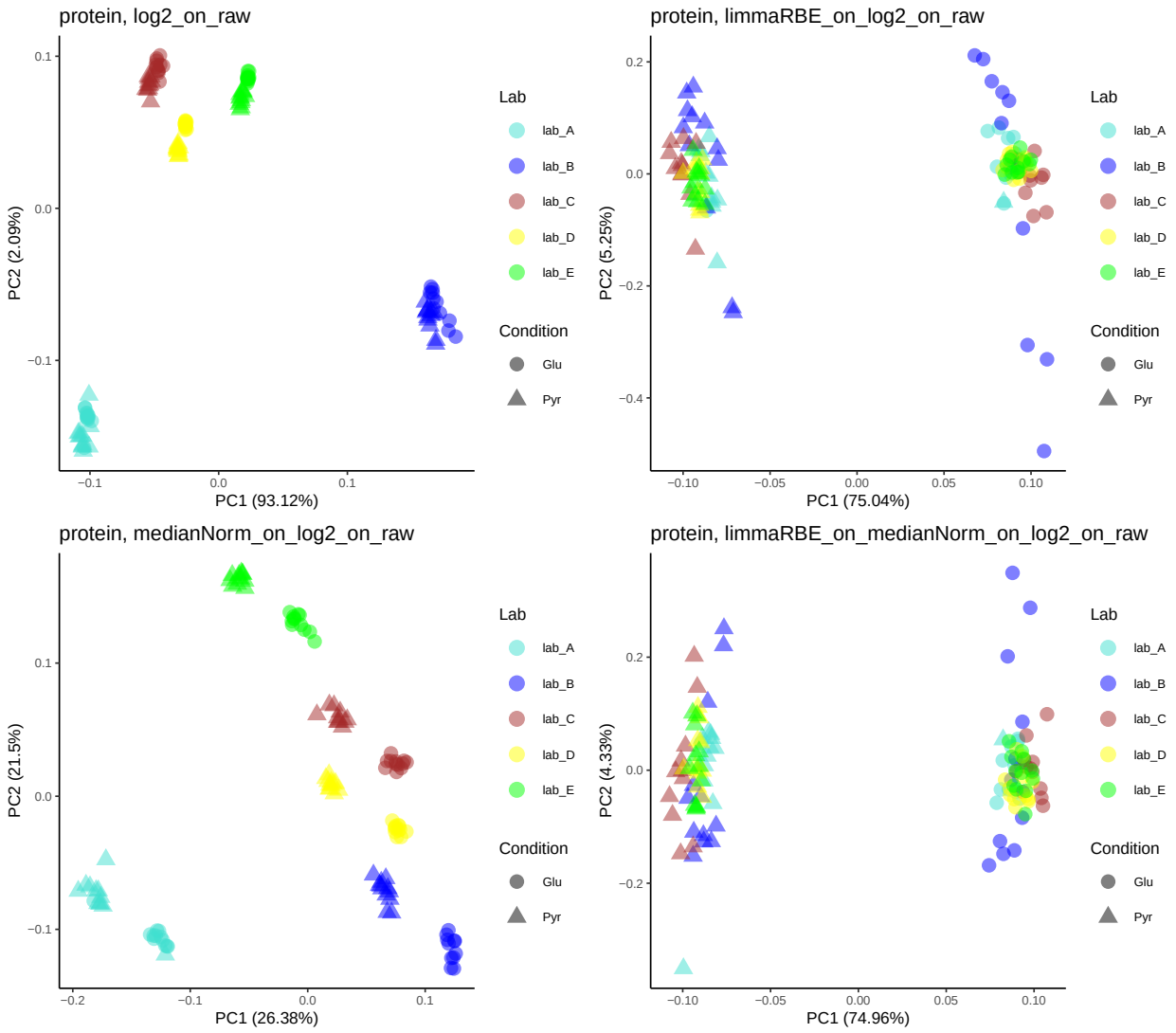
The four new assays (`protein::limmaRBE_on_...` and `protein::combat_on_...`) are now stored alongside the previous steps. We can compare them to the pre-correction assay using PCA to confirm that the batch effect was removed.

4.6.1 Principal component analysis

We will repeat the PCA on the final processed assay. We hope to see that the influence of the Lab (batch effect) is reduced and that biological variation is more prominent.

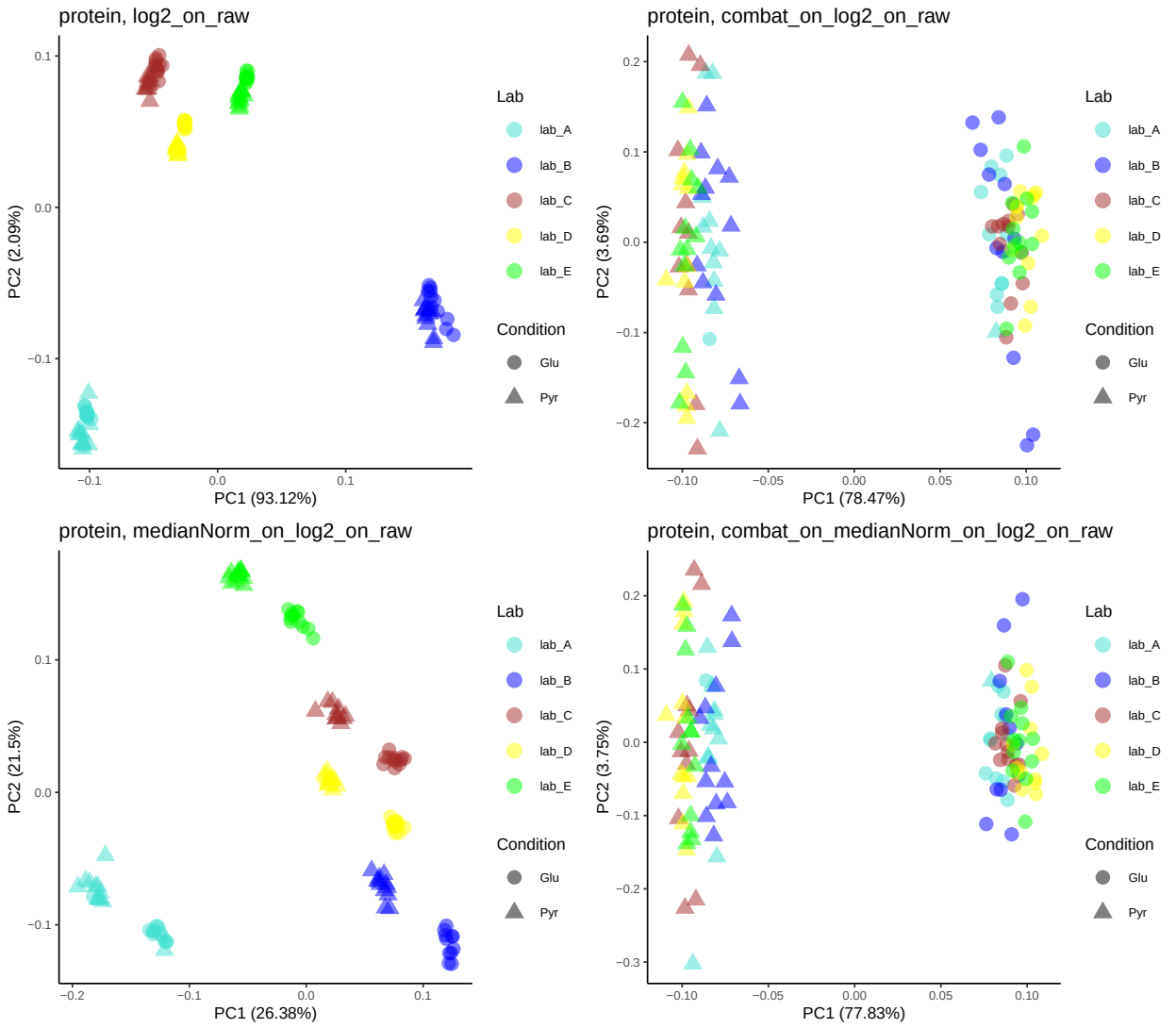
- for limma-corrected data:

```
plot_PCA(
    pbf,
    pbf_name = c(
        "protein::log2_on_raw",
        "protein::limmaRBE_on_log2_on_raw",
        "protein::medianNorm_on_log2_on_raw",
        "protein::limmaRBE_on_medianNorm_on_log2_on_raw"
    ),
    sample_id_col = "Run",
    color_scheme = color_scheme,
    color_by = "Lab",
    shape_by = "Condition",
    fill_the_missing = NULL,
    base_size = 8, point_size = 3, point_alpha = 0.5
)
```



- for combat-corrected data:

```
plot_PCA(
    pbf,
    pbf_name = c(
        "protein::log2_on_raw",
        "protein::combat_on_log2_on_raw",
        "protein::medianNorm_on_log2_on_raw",
        "protein::combat_on_medianNorm_on_log2_on_raw"
    ),
    sample_id_col = "Run",
    color_scheme = color_scheme,
    color_by = "Lab",
    shape_by = "Condition",
    fill_the_missing = NULL,
    base_size = 8, point_size = 3, point_alpha = 0.5
)
```

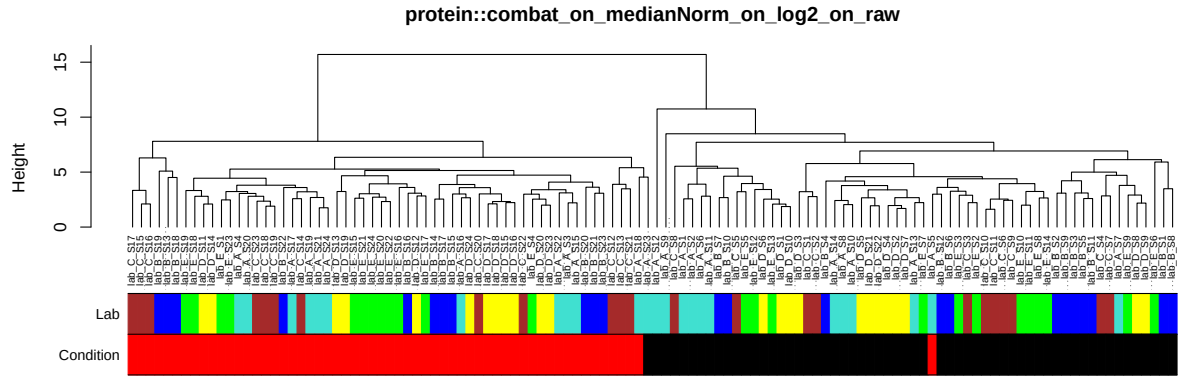


The progression from log2-only to batch-corrected assays shows the expected reduction of lab-driven clustering, while the biological condition becomes more visible on the final panel.

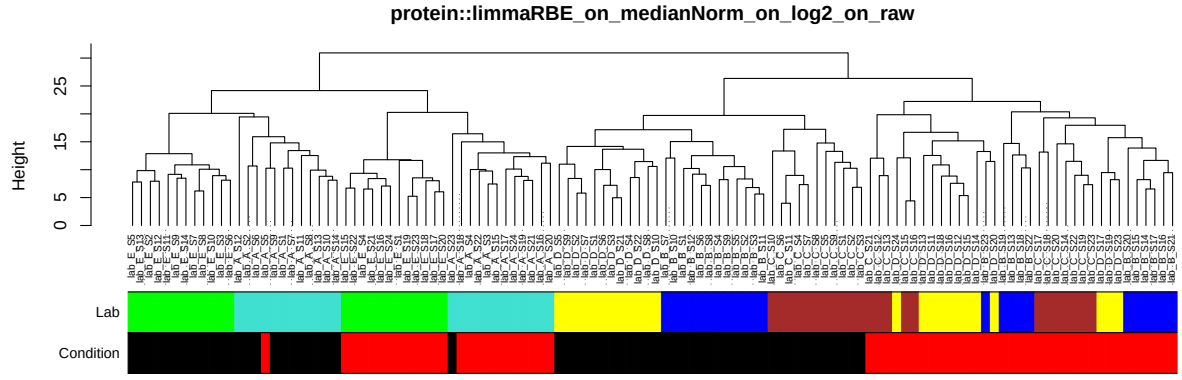
4.6.2 Hierarchical clustering

Hierarchical clustering provides another view of the sample distances. We focus on the `protein::combat_on_medianNorm_on_l` and `protein::limmaRBE_on_medianNorm_on_log2_on_raw` assays and colour samples by lab.

```
plot_hierarchical_clustering(
    pbf,
    pbf_name = "protein::combat_on_medianNorm_on_log2_on_raw",
    sample_id_col = "Run",
    label_font = 0.6,
    color_list = color_scheme
)
```




```
plot_hierarchical_clustering(
    pbf,
    pbf_name = "protein::limmaRBE_on_medianNorm_on_log2_on_raw",
    sample_id_col = "Run",
    label_font = 0.6,
    color_list = color_scheme
)
```

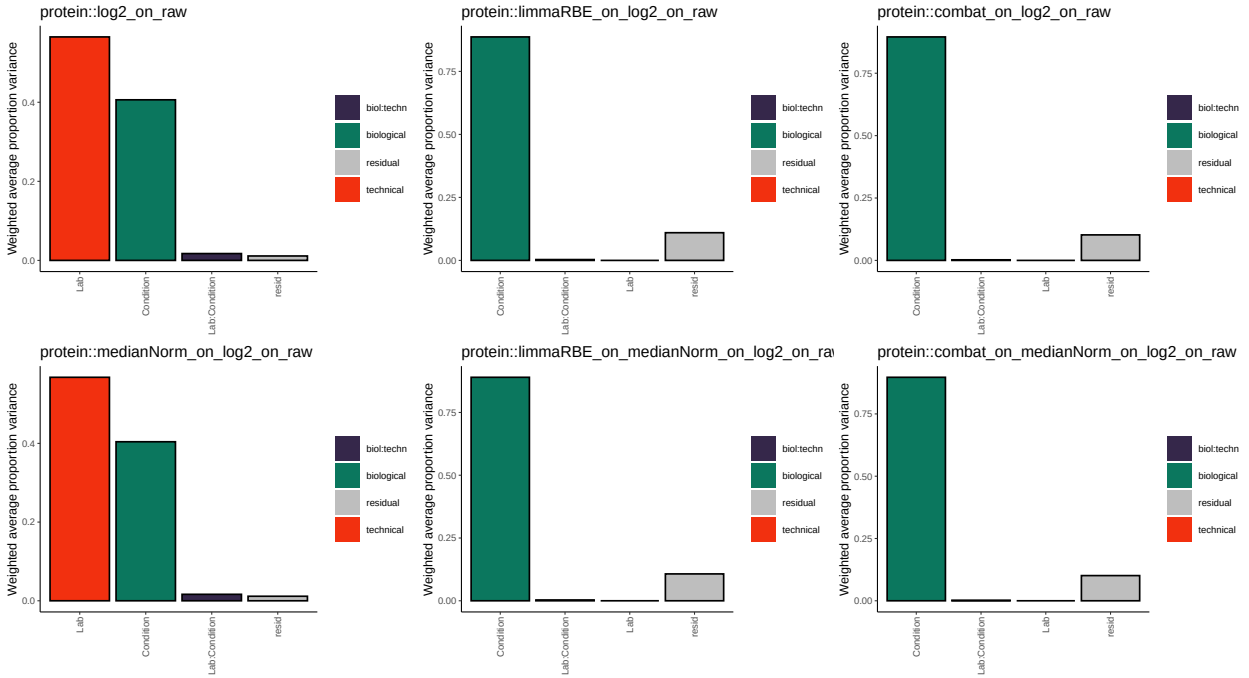


4.6.3 Principal Variance Component Analysis (PVCA)

PVCA quantifies the contribution of known factors to the observed variance. We contrast the raw, normalised, and ComBat-corrected assays.

```
plot_PVCA(
    pbf,
    pbf_name = c(
        "protein::log2_on_raw",
        "protein::limmaRBE_on_log2_on_raw",
        "protein::combat_on_log2_on_raw",

        "protein::medianNorm_on_log2_on_raw",
        "protein::limmaRBE_on_medianNorm_on_log2_on_raw",
        "protein::combat_on_medianNorm_on_log2_on_raw"

    ),
    sample_id_col = "Run",
    technical_factors = c("Lab"),
```

```
    biological_factors = c("Condition"),
    fill_the_missing = NULL,
    base_size = 7,
    plot_ncol = 3, # the number of plots in a row
    variance_threshold = 0  # the the percentile value of weight each of the
                            # covariates needs to explain
                            # (the rest will be lumped together)
)
```



The PVCA bar plot confirms the PCA observations: the lab factor dominates the log2 assay, shrinks after median normalisation, and dissapears after ComBat or limmaRBE. Residual variance increases slightly, which is expected once structured technical noise is removed.

5 Inspect operation log and lineage

Each call to `pb_transform()` or `pb_filterNA()` records a new entry that captures the source assay, operation, and resulting assay name. Inspecting the log keeps long pipelines reproducible.

```
get_operation_log(pbf) %>%
    as_tibble()
#> # A tibble: 11 x 7
#>    step                fun   from  to    params       timestamp           pkg
#>    <chr>               <chr> <chr> <chr> <list>       <dttm>              <chr>
#>  1 add_level(protein)_~ addA~ pept~ prot~ <named list> 2025-09-25 14:34:12 proB~
#>  2 filterNA            filt~ pept~ pept~ <named list> 2025-09-25 14:34:37 proB~
#>  3 filterNA            filt~ prot~ prot~ <named list> 2025-09-25 14:34:38 proB~
#>  4 log2                log2  prot~ prot~ <named list> 2025-09-25 14:34:55 proB~
#>  5 log2                log2  pept~ pept~ <named list> 2025-09-25 14:34:55 proB~
#>  6 medianNorm          medi~ pept~ pept~ <list [0]>   2025-09-25 14:35:05 proB~
#>  7 medianNorm          medi~ prot~ prot~ <list [0]>   2025-09-25 14:35:05 proB~
#>  8 limmaRBE            limm~ prot~ prot~ <named list> 2025-09-25 14:35:11 proB~
#>  9 limmaRBE            limm~ prot~ prot~ <named list> 2025-09-25 14:35:12 proB~
```

```
#> 10 combat                comb~ prot~ prot~ <named list> 2025-09-25 14:35:17 proB~
#> 11 combat                comb~ prot~ prot~ <named list> 2025-09-25 14:35:17 proB~
```

If you need a compact name for the current assay chain (for example, to label benchmark results), use `pb_pipeline_name(pbf, "protein::combat_on_medianNorm_on_log2_on_raw")`.

```
pb_pipeline_name(pbf, "protein::combat_on_medianNorm_on_log2_on_raw")
#> [1] "combat_on_medianNorm_on_log2_on_raw"
```

5.1 Extract matrices from pbf object

It is possible to extract a specific matrix from a `ProBatchFeatures` object using the assay name:

```
extracted_matrix <- pb_assay_matrix(
    pbf,
    assay = "peptide::raw"
)
# show
extracted_matrix[1:5, 1:5]
#>                    lab_A_S1  lab_A_S2   lab_A_S3   lab_A_S4  lab_A_S5
#> AAADLISR2          14647900  13463600    8518130   12172600  11612300
#> AAADVQLR1           1430600   1467280    1819720    1179440    735616
#> AAADVQLR2          73128300  56024800   70081100   65148600  77244700
#> AAAFEGELIPASQIDR2 974753000 900284000 1314820000 1271960000 822345000
#> AAAFEGELIPASQIDR3  26860200  22756400   35418900   33923100  38674600
rm(extracted_matrix)
```

The assay can also be extracted into "long" format. In this case, the `sample_id_col` needs to be specified:

```
extracted_long <- pb_as_long(
    pbf,
    sample_id_col = "Run",
    pbf_name = "peptide::raw"
)

# show
extracted_long[1:3, ]
#>   feature_label      Run Intensity   Lab Condition
#> 1     AAADLISR2 lab_A_S1  14647900 lab_A       Pyr
#> 2     AAADVQLR1 lab_A_S1   1430600 lab_A       Pyr
#> 3     AAADVQLR2 lab_A_S1  73128300 lab_A       Pyr
rm(extracted_long)
```

6 Session info

```
sessionInfo()
#> R version 4.4.3 (2025-02-28)
#> Platform: x86_64-conda-linux-gnu
#> Running under: Ubuntu 24.04.2 LTS
#>
#> Matrix products: default
#> BLAS:   /home/yuliya-cosybio/miniforge3/envs/ProBatch2/lib/libblis.so.4.0.0
#> LAPACK: /home/yuliya-cosybio/miniforge3/envs/ProBatch2/lib/liblapack.so.3.9.0
#>
```

```
#> locale:
#>  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
#>  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
#>  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
#>  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
#>  [9] LC_ADDRESS=C               LC_TELEPHONE=C
#> [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
#>
#> time zone: Europe/Berlin
#> tzcode source: system (glibc)
#>
#> attached base packages:
#> [1] stats     graphics  grDevices utils     datasets  methods   base
#>
#> other attached packages:
#> [1] proBatch_1.3.1 testthat_3.2.3 ggplot2_3.5.2  tibble_3.3.0   dplyr_1.1.4
#>
#> loaded via a namespace (and not attached):
#>   [1] splines_4.4.3               later_1.4.4
#>   [3] ggplotify_0.1.3             preprocessCore_1.68.0
#>   [5] XML_3.99-0.17               rpart_4.1.24
#>   [7] lifecycle_1.0.4             Rdpack_2.6.4
#>   [9] fastcluster_1.3.0           edgeR_4.4.0
#>  [11] doParallel_1.0.17           rprojroot_2.1.1
#>  [13] lattice_0.22-7              MASS_7.3-64
#>  [15] MultiAssayExperiment_1.32.0 backports_1.5.0
#>  [17] magrittr_2.0.3              limma_3.62.1
#>  [19] Hmisc_5.2-3                 rmarkdown_2.29
#>  [21] yaml_2.3.10                 wesanderson_0.3.7
#>  [23] remotes_2.5.0               httpuv_1.6.16
#>  [25] sessioninfo_1.2.3           pkgbuild_1.4.8
#>  [27] MsCoreUtils_1.18.0          DBI_1.2.3
#>  [29] minqa_1.2.8                 RColorBrewer_1.1-3
#>  [31] lubridate_1.9.4             abind_1.4-5
#>  [33] pkgload_1.4.0               zlibbioc_1.52.0
#>  [35] GenomicRanges_1.58.0        purrr_1.1.0
#>  [37] AnnotationFilter_1.30.0     BiocGenerics_0.52.0
#>  [39] yulab.utils_0.2.1           nnet_7.3-20
#>  [41] rappdirs_0.3.3              sva_3.54.0
#>  [43] GenomeInfoDbData_1.2.13     IRanges_2.40.0
#>  [45] S4Vectors_0.44.0            genefilter_1.88.0
#>  [47] pheatmap_1.0.13             annotate_1.84.0
#>  [49] codetools_0.2-20            DelayedArray_0.32.0
#>  [51] tidyselect_1.2.1            UCSC.utils_1.2.0
#>  [53] farver_2.1.2                lme4_1.1-37
#>  [55] viridis_0.6.5               matrixStats_1.5.0
#>  [57] stats4_4.4.3                dynamicTreeCut_1.63-1
#>  [59] base64enc_0.1-3             jsonlite_2.0.0
#>  [61] ellipsis_0.3.2              Formula_1.2-5
#>  [63] survival_3.8-3              iterators_1.0.14
#>  [65] foreach_1.5.2               tools_4.4.3
#>  [67] Rcpp_1.1.0                  glue_1.8.0
#>  [69] BiocBaseUtils_1.8.0         gridExtra_2.3
```

```
#>  [71] SparseArray_1.6.0          xfun_0.53
#>  [73] mgcv_1.9-3                 MatrixGenerics_1.18.0
#>  [75] usethis_3.2.0              ggfortify_0.4.19
#>  [77] GenomeInfoDb_1.42.0        withr_3.0.2
#>  [79] BiocManager_1.30.26        fastmap_1.2.0
#>  [81] pvca_1.46.0                boot_1.3-31
#>  [83] digest_0.6.37              gridGraphics_0.5-1
#>  [85] timechange_0.3.0           R6_2.6.1
#>  [87] mime_0.13                  colorspace_2.1-1
#>  [89] GO.db_3.20.0               RSQLite_2.4.3
#>  [91] utf8_1.2.6                 tidyr_1.3.1
#>  [93] generics_0.1.4             data.table_1.17.8
#>  [95] httr_1.4.7                 htmlwidgets_1.6.4
#>  [97] S4Arrays_1.6.0             pkgconfig_2.0.3
#>  [99] gtable_0.3.6               blob_1.2.4
#> [101] impute_1.80.0              XVector_0.46.0
#> [103] brio_1.1.5                 htmltools_0.5.8.1
#> [105] profvis_0.4.0              ProtGenerics_1.38.0
#> [107] clue_0.3-66                scales_1.4.0
#> [109] Biobase_2.66.0             png_0.1-8
#> [111] reformulas_0.4.1           corrplot_0.95
#> [113] knitr_1.50                 rstudioapi_0.17.1
#> [115] reshape2_1.4.4             checkmate_2.3.3
#> [117] nlme_3.1-168               nloptr_2.2.1
#> [119] cachem_1.1.0               stringr_1.5.1
#> [121] parallel_4.4.3             miniUI_0.1.2
#> [123] foreign_0.8-90             AnnotationDbi_1.68.0
#> [125] vsn_3.74.0                 desc_1.4.3
#> [127] pillar_1.11.0              grid_4.4.3
#> [129] vctrs_0.6.5                urlchecker_1.0.1
#> [131] promises_1.3.3             xtable_1.8-4
#> [133] cluster_2.1.8.1            htmlTable_2.4.3
#> [135] evaluate_1.0.5             cli_3.6.5
#> [137] locfit_1.5-9.12            compiler_4.4.3
#> [139] rlang_1.1.6                crayon_1.5.3
#> [141] labeling_0.4.3             QFeatures_1.16.0
#> [143] affy_1.84.0                plyr_1.8.9
#> [145] fs_1.6.6                   stringi_1.8.7
#> [147] viridisLite_0.4.2          WGCNA_1.73
#> [149] BiocParallel_1.40.0        Biostrings_2.74.0
#> [151] lazyeval_0.2.2             devtools_2.4.5
#> [153] Matrix_1.7-3               bit64_4.6.0-1
#> [155] KEGGREST_1.46.0            statmod_1.5.0
#> [157] shiny_1.11.1               SummarizedExperiment_1.36.0
#> [159] rbibutils_2.3              igraph_2.1.4
#> [161] memoise_2.0.1              affyio_1.76.0
#> [163] bit_4.6.0
```


1. Gatto, L. & Vanderaa, C. *QFeatures: Quantitative features for mass spectrometry data.* (2025). at <https://github.com/RforMassSpectrometry/QFeatures>
2. Burankova, Y. *et al.* Privacy-preserving multicenter differential protein abundance analysis with FedProt. *Nature Computational Science* 1–14 (2025).