Package: glyclean (via r-universe)

May 18, 2026

Title Perform Preprocessing on Glycomics and Glycoproteomics Data

Version 0.14.1

Description Provides comprehensive data preprocessing tools for glycomics and glycoproteomics experiments. The package offers both automated and manual preprocessing pipelines, including normalization (median, quantile, vsn, total area), missing value handling (filtering, imputation using various methods), batch effect correction (using ComBat and other methods), data aggregation (at peptide, glycan, or site level), and quality control functions. The automated pipeline intelligently selects appropriate methods based on data characteristics and sample size, making it easy to prepare clean, analysis-ready data for downstream statistical analysis within the glycoverse ecosystem. Works seamlessly with 'glyexp::experiment()' objects to ensure data consistency and interoperability.

License MIT + file LICENSE

Suggests testthat (>= 3.2.0), vdiffr, glyread (>= 0.6.2), glyrepr (>= 0.7.4), vsn, impute, MASS, knitr, rmarkdown, pheatmap, ggplotify, factoextra, patchwork, compositions

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

URL https://glycoverse.github.io/glyclean/

Imports checkmate, cli, dplyr, glyexp (>= 0.12.1), limma, magrittr, matrixStats, missForest, pcaMethods, purrr, rlang, seqinr, stringr, tidyr, tibble, tidyselect, UniProt.ws, withr, forcats, stats, ggplot2, lifecycle, sva

Depends R (>= 4.1)

VignetteBuilder knitr

Config/pak/sysreqs libglpk-dev libicu-dev libpng-dev libxml2-dev libssl-dev zlib1g-dev

Repository https://glycoverse.r-universe.dev

Date/Publication 2026-05-18 02:52:54 UTC

RemoteUrl https://github.com/glycoverse/glyclean

RemoteRef v0.14.1

RemoteSha d10d13d69b9f8b3cb93d8fd4fe5b4623497c1572

Contents

`add_site_seq` *Add site-specific sequence information.*

Description

This function adds a new "site_sequence" column to the variable information table. It is the protein sequence around the glycosylation site. If the head and tail amino acids of the peptide sequence are insufficient, fill with "X".

This function requires the following columns in the variable information tibble:

- "protein": The protein uniprot accession.
- "protein_site": The site on the protein sequence.

Usage

```
add_site_seq(exp, fasta = NULL, n_aa = 7, taxid = 9606)

## S3 method for class 'glyexp_experiment'
add_site_seq(exp, fasta = NULL, n_aa = 7, taxid = 9606)

## Default S3 method:
add_site_seq(exp, fasta = NULL, n_aa = 7, taxid = 9606)
```

Arguments

`exp`	A `glyexp::experiment()` object with "glycoproteomics" type.
`fasta`	Either a file path to a FASTA file, a named character vector with protein IDs as names and sequences as value, or `NULL` to fetch from UniProt.
`n_aa`	The number of amino acids to the left and right of the glycosylation site. For example, if `n_aa = 5`, the resulting sequence will contain 11 amino acids.
`taxid`	NCBI taxonomy ID for UniProt lookup. Default: `9606` (human).

Value

A `glyexp::experiment()` object with the new "site_sequence" column.

`adjust_protein` *Adjust Protein Expression*

Description

This function adjusts the glycopeptide expression by the protein expression. In another word, it "strips out" the protein expression from the glycopeptide expression, so that the expression reflects the glycosylation status only.

Usage

```
adjust_protein(exp, pro_expr_mat, method = c("ratio", "reg"))

## S3 method for class 'glyexp_experiment'
adjust_protein(exp, pro_expr_mat, method = c("ratio", "reg"))

## Default S3 method:
adjust_protein(exp, pro_expr_mat, method = c("ratio", "reg"))
```

Arguments

`exp`	A `glyexp::experiment()` object with "glycoproteomics" type.
`pro_expr_mat`	A matrix of protein expression. Columns are samples, rows are uniprot protein accessions.
`method`	The method to use for protein adjustment. Either "ratio" or "reg". Default is "ratio".

Details

For simplicity, glycopeptide expression is denoted as `GP`, and protein expression is denoted as `PRO`.

"ratio" method:

`GP-adj` = (`GP` / `PRO`) / (median(`GP`) / median(`PRO`))

The first part is to adjust glycopeptide expression by protein expression. The second part is to rescale the expression to the original scale.

"reg" method:

Use linear regression to remove the protein expression from the glycopeptide expression. Both glycopeptide and protein expression values are log2-transformed (with +1 to avoid log(0)) before fitting the linear model: log2(GP+1) ~ log2(PRO+1). The residuals represent the glycosylation-specific signal and are converted back to the original scale using 2^residuals, ensuring all adjusted values are positive.

In both methods, only glycoproteins identified in both `exp` and `pro_expr_mat` will be retained.

Value

A `glyexp::experiment()` object with adjusted protein expression.

`aggregate` *Aggregate Data*

Description

Aggregate glycoproteomics data to different levels (glycoforms, glycopeptides, etc.). This function sums up quantitative values for each unique combination of specified variables. It is recommended to call this function after missing value imputation.

The following levels are available:

- "gf": Aggregate to glycoforms, which is the unique combination of proteins, protein sites, and glycan compositions. This is the default level.
- "gp": Aggregate to glycopeptides, which is the unique combination of peptides, proteins, protein sites, and glycan compositions.
- "gfs": Like "gf", but differentiates structures with the same composition.
- "gps": Like "gp", but differentiates structures with the same composition.

Different levels of aggregation require different columns in the variable information.

- "gf": "protein", "glycan_composition", "protein_site"
- "gp": "peptide", "protein", "glycan_composition", "peptide_site", "protein_site"
- "gfs": "protein", "glycan_composition", "glycan_structure", "protein_site"
- "gps": "peptide", "protein", "glycan_composition", "glycan_structure", "peptide_site", "protein_site"

Other columns in the variable information tibble with "many-to-one" relationship with the unique combination of columns listed above will be kept. A common example is the "gene" column. For each "glycoform" (unique combination of "protein", "protein_site", and "glycan_composition"), there should be only one "gene" value, therefore it is kept for "gf" level. On the other hand, the "peptide" column is removed for "gf" level, as one "glycoform" can contain multiple "peptides".

Usage

```
aggregate(
  exp,
  to_level = c("gf", "gp", "gfs", "gps"),
  standardize_variable = TRUE
)

## S3 method for class 'glyexp_experiment'
glyclean_aggregate(
  exp,
```

```
  to_level = c("gf", "gp", "gfs", "gps"),
  standardize_variable = TRUE
)

## Default S3 method:
glyclean_aggregate(
  exp,
  to_level = c("gf", "gp", "gfs", "gps"),
  standardize_variable = TRUE
)
```

Arguments

`exp`	A `glyexp::experiment()` object with "glycoproteomics" type.
`to_level`	The aggregation level, one of: "gf" (glycoforms), "gp" (glycopeptides), "gfs" (glycoforms with structures), or "gps" (glycopeptides with structures). See Details for more information.
`standardize_variable`	Whether to call `glyexp::standardize_variable()` after aggregation. Set to `FALSE` to skip network calls for faster testing. Default is `TRUE`.

Value

A modified `glyexp::experiment()` object with aggregated expression matrix and updated variable information.

`auto_aggregate` *Automatic Aggregation*

Description

Aggregates glycoproteomics data to "gfs" (glycoforms with structures) level if the glycan structure column exists, otherwise to "gf" (glycoforms with compositions) level.

Usage

```
auto_aggregate(exp, standardize_variable = TRUE)
```

Arguments

`exp`	A `glyexp::experiment()` object with "glycoproteomics" type.
`standardize_variable`	Whether to call `glyexp::standardize_variable()` after aggregation. Set to `FALSE` to skip network calls for faster testing. Default is `TRUE`.

Value

A modified `glyexp::experiment()` object with aggregated expression matrix and updated variable information.

Examples

```
library(glyexp)
exp <- real_experiment
auto_aggregate(exp)
```

`auto_clean` *Automatic Data Preprocessing*

Description

Perform automatic data preprocessing on glycoproteomics or glycomics data. This function applies an intelligent preprocessing pipeline that includes normalization, missing value handling, imputation, aggregation (for glycoproteomics data), and batch effect correction.

For glycomics data, this function calls these functions in sequence:

- `auto_remove()`
- `auto_normalize()`
- `normalize_total_area()`
- `auto_impute()`
- `auto_correct_batch_effect()`

For glycoproteomics data, this function calls these functions in sequence:

- `auto_normalize()`
- `auto_remove()`
- `auto_impute()`
- `auto_aggregate()`
- `auto_normalize()`
- `auto_correct_batch_effect()`

Usage

```
auto_clean(
  exp,
  group_col = "group",
  batch_col = "batch",
  qc_name = "QC",
  normalize_to_try = NULL,
  impute_to_try = NULL,
  remove_preset = "discovery",
  batch_prop_threshold = 0.3,
  check_batch_confounding = TRUE,
  batch_confounding_threshold = 0.4,
  standardize_variable = TRUE
)
```

Arguments

`exp`	A `glyexp::experiment()` containing glycoproteomics or glycomics data.
`group_col`	The column name in sample_info for groups. Default is "group". Can be NULL when no group information is available.
`batch_col`	The column name in sample_info for batches. Default is "batch". Can be NULL when no batch information is available.
`qc_name`	[**Deprecated**] This function no longer uses QC sample information. This parameter is ignored and will be removed in a future release.
`normalize_to_try`	[**Deprecated**] This parameter is no longer used and will be removed in a future release. The automatic normalization strategy is now deterministic and does not require user-specified methods to try.
`impute_to_try`	[**Deprecated**] This parameter is no longer used and will be removed in a future release.
`remove_preset`	The preset for removing variables. Default is "discovery". Available presets: • "simple": remove variables with more than 50% missing values. • "discovery": more lenient, remove variables with more than 80% missing values, but ensure less than 50% of missing values in at least one group. • "biomarker": more strict, remove variables with more than 40% missing values, and ensure less than 60% of missing values in all groups.
`batch_prop_threshold`	The proportion of variables that must show significant batch effects to perform batch correction. Default is 0.3 (30%).
`check_batch_confounding`	Whether to check for confounding between batch and group variables. Default to TRUE.
`batch_confounding_threshold`	The threshold for Cramer's V to consider batch and group variables highly confounded. Only used when `check_batch_confounding` is TRUE. Default to 0.4.
`standardize_variable`	Whether to call `glyexp::standardize_variable()` after aggregation. Set to `FALSE` to skip network calls for faster testing. Default is `TRUE`.

Value

A modified `glyexp::experiment()` object.

See Also

`auto_normalize()`, `auto_remove()`, `auto_impute()`, `auto_aggregate()`, `auto_correct_batch_effect()`

Examples

```
library(glyexp)
exp <- real_experiment
auto_clean(exp)
```

`auto_coda` *Automatical CoDA transformation*

Description

If the data has more than 50 variables, call `transform_alr()`. Otherwise, call `transform_clr()`, following the same strategy in glycowork.

Usage

```
auto_coda(x, by = NULL, gamma = 0.1, group_scales = NULL)
```

Arguments

`x`	Either a `glyexp_experiment` object or a matrix. If a matrix, rows should be variables and columns should be samples.
`by`	Either a column name in `sample_info` (for `glyexp_experiment` input) or a factor/vector with one value per sample.
`gamma`	Standard deviation of the scale-uncertainty model on the `log2` scale. Default is `0.1`. Set to `0` for deterministic transformation.
`group_scales`	Optional informed group scales. For binary comparisons, this can be a single positive ratio for the second group relative to the first, or two positive scales from which that ratio is derived. For multi-group data, provide a positive vector with one scale per group.

Value

Returns the same type as the input. If `x` is a `glyexp_experiment`, returns a `glyexp_experiment` with a CoDA-transformed expression matrix (ALR if >50 variables, CLR otherwise).

Algorithmic details

The stochastic branch follows `glycowork`: when `gamma > 0`, CLR noise is sampled per feature-sample entry rather than once per sample. Without informed scales, this corresponds to jittering around the sample-specific `log2` geometric mean of the non-zero components and then returning to ratio space.

The `group_scales` argument is interpreted in the same way as `glycowork`'s `custom_scale`. For exactly two groups, provide the total-signal ratio of the second group relative to the first, either directly as a scalar or implicitly through two per-group scales. For multi-group data, provide one positive scale per group; these scales are used only in the stochastic branch.

Motif quantification

If you intend to use ALR/CLR transformation with motif quantification, you should apply the transformation after motif quantification rather than before. In another words, the recommended workflow is:

1. Perform data preprocessing.

```
clean_exp <- auto_clean(exp)
```

1. Perform motif quantification on the cleaned experiment.

```
motif_exp <- glydet::quantify_motifs(clean_exp, motifs)
```

1. Perform ALR/CLR transformation on the motif-quantified experiment.

```
coda_motif_exp <- auto_coda(motif_exp, by = "group", gamma = 0.1)
```

1. Proceed with downstream analyses on the transformed motif experiment.

```
dea_res <- glystats::gly_ttest(coda_motif_exp)
```

---

auto_correct_batch_effect

Automatic Batch Correction

Description

Detects and corrects batch effects in the experiment. If batch information is available, this function performs ANOVA to detect batch effects. If more than 30% (controlled by `prop_threshold`) of variables show significant batch effects ($p < 0.05$), batch correction is performed using ComBat. If group information exists, it will be used as a covariate in both detection and correction to preserve biological variation. If no batch information is available, the function will return the original experiment.

Usage

```
auto_correct_batch_effect(
  exp,
  group_col = "group",
  batch_col = "batch",
  prop_threshold = 0.3,
  check_confounding = TRUE,
  confounding_threshold = 0.4
)
```

Arguments

exp	A glyexp::experiment() object.
group_col	The column name in sample_info for groups. Default is "group". Can be NULL when no group information is available.
batch_col	The column name in sample_info for batches. Default is "batch". Can be NULL when no batch information is available.
prop_threshold	The proportion of variables that must show significant batch effects to perform batch correction. Default is 0.3 (30%).
check_confounding	Whether to check for confounding between batch and group variables. Default to TRUE.
confounding_threshold	The threshold for Cramer's V to consider batch and group variables highly confounded. Only used when check_confounding is TRUE. Default to 0.4.

Value

A glyexp::experiment() object with batch effects corrected.

Examples

```
exp <- glyexp::real_experiment
exp <- auto_correct_batch_effect(exp)
```

---

auto_impute *Automatic Imputation*

Description

This function automatically selects and applies a deterministic default imputation method by sample count and experiment type. Quality Control (QC) samples are inspected for workflow consistency, but they are not used to benchmark or select the imputation method.

Usage

```
auto_impute(exp, group_col = "group", qc_name = "QC", to_try = NULL)
```

Arguments

exp	An glyexp::experiment().
group_col	The column name in sample_info for groups. Default is "group". Can be NULL when no group information is available.
qc_name	[**Deprecated**] This function no longer uses QC sample information. This parameter is ignored and will be removed in a future release.

to_try [**Deprecated**] This parameter is no longer used and will be removed in a future release. The automatic strategy is now deterministic and does not require user-specified methods to try.

Details

The automatic strategy uses these defaults:

- `n_samples < 30`: `impute_min_prob()` for glycomics and glycoproteomics.
- `30 <= n_samples <= 100`: `impute_bpca()` for glycomics and `impute_min_prob()` for glycoproteomics.
- `n_samples > 100`: `impute_miss_forest()` for glycomics and `impute_bpca()` for glycoproteomics.

Other experiment types use the glycoproteomics defaults as a conservative fallback. `impute_sample_min()` and `impute_half_sample_min()` remain available for manual use, but they are not selected automatically.

Value

The imputed experiment.

Examples

```
library(glyexp)
exp_imputed <- auto_impute(real_experiment)
```

---

auto_normalize *Automatic Normalization*

Description

This function automatically applies a deterministic default normalization method based on experiment type. Quality Control (QC) samples are not used to benchmark, select, or report normalization behavior.

Usage

```
auto_normalize(exp, group_col = "group", qc_name = "QC", to_try = NULL)
```

Arguments

exp An `glyexp::experiment()`.

group_col The column name in sample_info for groups. Default is "group". Can be NULL when no group information is available.

qc_name [**Deprecated**] This function no longer uses this argument. This parameter is ignored and will be removed in a future release.

to_try [**Deprecated**] This parameter is no longer used and will be removed in a future release. The automatic strategy is now deterministic and does not require user-specified methods to try.

Details

The automatic strategy uses these defaults:

- glycomics: normalize_total_area().
- glycoproteomics: normalize_median().
- missing or other experiment types: normalize_median().

Value

The normalized experiment.

Examples

```
library(glyexp)
exp_normed <- auto_normalize(real_experiment)
```

---

auto_remove *Automatic Removing Variables*

Description

This function uses preset rules to remove variables with low quality. Available presets:

- "simple": remove variables with more than 50% missing values.
- "discovery": more lenient, remove variables with more than 80% missing values, but ensure less than 50% of missing values in at least one group.
- "biomarker": more strict, remove variables with more than 40% missing values, and ensure less than 60% of missing values in all groups.

Usage

```
auto_remove(exp, preset = "discovery", group_col = "group", qc_name = "QC")
```

Arguments

exp	A glyexp_experiment object.
preset	One of "simple", "discovery", or "biomarker". Default "discovery" if group information is available, otherwise "simple".
group_col	The column name in sample_info for groups. Default is "group". Can be NULL when no group information is available.
qc_name	[**Deprecated**] This function no longer uses QC sample information. This parameter is ignored and will be removed in a future release.

Value

A modified glyexp::experiment() object.

Examples

```
library(glyexp)
exp <- real_experiment
auto_remove(exp)
```

`correct_batch_effect` *Correct Batch Effect*

Description

Correct batch effects in glycoproteomics/glycomics data using ComBat algorithm from the sva package or removeBatchEffect from the limma package.

Usage

```
correct_batch_effect(
  x,
  batch = "batch",
  group = NULL,
  check_confounding = TRUE,
  confounding_threshold = 0.4,
  method = c("combat", "limma")
)

## S3 method for class 'glyexp_experiment'
correct_batch_effect(
  x,
  batch = "batch",
  group = NULL,
  check_confounding = TRUE,
  confounding_threshold = 0.4,
  method = c("combat", "limma")
)

## S3 method for class 'matrix'
correct_batch_effect(
  x,
  batch = "batch",
  group = NULL,
  check_confounding = TRUE,
  confounding_threshold = 0.4,
  method = c("combat", "limma")
)

## Default S3 method:
correct_batch_effect(
```

```
  x,
  batch = "batch",
  group = NULL,
  check_confounding = TRUE,
  confounding_threshold = 0.4,
  method = c("combat", "limma")
)
```

Arguments

`x`	Either a `glyexp_experiment` object or a matrix. If a matrix, rows should be variables and columns should be samples.
`batch`	Either a factor/character vector specifying batch assignments for each sample, or a string specifying the column name in sample_info (for experiment input only). Default to "batch" for experiment input.
`group`	Either a factor/character vector specifying group assignments for each sample, or a string specifying the column name in sample_info (for experiment input only). If provided, it will be used as a covariate in the batch correction model. This is useful when you have an unbalanced design. Default to NULL.
`check_confounding`	Whether to check for confounding between batch and group variables. Default to TRUE.
`confounding_threshold`	The threshold for Cramer's V to consider batch and group variables highly confounded. Only used when `check_confounding` is TRUE. Default to 0.4.
`method`	The batch correction method to use. Either "combat" (default, uses sva::ComBat) or "limma" (uses limma::removeBatchEffect). Default to "combat".

Details

This function performs batch effect correction using either the ComBat algorithm or the limma removeBatchEffect function. It requires batch information provided via the `batch` parameter. If no batch information is available, the function will return the original data unchanged.

If group information is provided via `group`, the function will check for confounding between batch and group variables. If batch and group are highly confounded (|Cramer's V| > `threshold`), the function will issue a warning and return the original data unchanged to avoid over-correction.

When both batch and group information are available and not highly confounded, the group information will be included in the model to preserve biological variation while correcting for batch effects.

Value

For `glyexp_experiment` input, returns a modified `glyexp_experiment` object. For matrix input, returns a batch-corrected matrix.

Examples

```
# With glyexp_experiment and column names
exp <- glyexp::toy_experiment
exp$sample_info$batch <- c("A", "A", "A", "B", "B", "B")
exp$sample_info$group <- c("Ctrl", "Ctrl", "Treat", "Ctrl", "Treat", "Treat")
corrected_exp <- correct_batch_effect(exp, batch = "batch", group = "group")

# With matrix and factor vectors
mat <- matrix(abs(rnorm(200)), nrow = 20, ncol = 10)
batch_factor <- factor(rep(c("A", "B"), each = 5))
group_factor <- factor(rep(c("Ctrl", "Treat"), times = 5))
corrected_mat <- correct_batch_effect(mat, batch = batch_factor, group = group_factor)

# Using limma method
corrected_mat <- correct_batch_effect(
  mat, batch = batch_factor, group = group_factor, method = "limma"
)
```

---

detect_batch_effect *Detect batch effect*

Description

Use ANOVA to detect if batch effect is present in the data. If `group` is provided, it will be used as a covariate in the ANOVA model.

Usage

```
detect_batch_effect(x, batch = "batch", group = NULL)

## S3 method for class 'glyexp_experiment'
detect_batch_effect(x, batch = "batch", group = NULL)

## S3 method for class 'matrix'
detect_batch_effect(x, batch = "batch", group = NULL)

## Default S3 method:
detect_batch_effect(x, batch = "batch", group = NULL)
```

Arguments

`x`	Either a `glyexp_experiment` object or a matrix. If a matrix, rows should be variables and columns should be samples.
`batch`	Either a factor/character vector specifying batch assignments for each sample, or a string specifying the column name in sample_info (for experiment input only). Default to ”batch” for experiment input.

group — Either a factor/character vector specifying group assignments for each sample, or a string specifying the column name in sample_info (for experiment input only). If provided, it will be used as a covariate in the ANOVA model. This is useful when you have an unbalanced design. Default to NULL.

Value

A double vector of p-values for each variable, i.e., the same length as `nrow(x)` (for matrix) or `nrow(get_expr_mat(x))` (for experiment)

Examples

```
# With glyexp_experiment and column names
exp <- glyexp::toy_experiment
exp$sample_info$batch <- c("A", "A", "A", "B", "B", "B")
exp$sample_info$group <- c("Ctrl", "Ctrl", "Treat", "Ctrl", "Treat", "Treat")
p_values <- detect_batch_effect(exp, batch = "batch", group = "group")

# With matrix and factor vectors
mat <- matrix(rnorm(200), nrow = 20, ncol = 10)
batch_factor <- factor(rep(c("A", "B"), each = 5))
group_factor <- factor(rep(c("Ctrl", "Treat"), times = 5))
p_values <- detect_batch_effect(mat, batch = batch_factor, group = group_factor)
```

---

impute_bpca *BPCA Imputation*

Description

A wrapper around the `pcaMethods::pca()`. Impute missing values using Bayesian principal component analysis (BPCA). BPCA combines an EM approach for PCA with a Bayesian model. In standard PCA data far from the training set but close to the principal subspace may have the same reconstruction error. BPCA defines a likelihood function such that the likelihood for data far from the training set is much lower, even if they are close to the principal subspace.

Usage

```
impute_bpca(x, by = NULL, ...)

## S3 method for class 'glyexp_experiment'
impute_bpca(x, by = NULL, ...)

## S3 method for class 'matrix'
impute_bpca(x, by = NULL, ...)

## Default S3 method:
impute_bpca(x, by = NULL, ...)
```

Arguments

x	Either a glyexp_experiment object or a matrix. If a matrix, rows should be variables and columns should be samples.
by	Either a column name in sample_info (string) or a factor/vector specifying group assignments for each sample. Used for grouping when imputing missing values.
...	Additional arguments to pass to pcaMethods::pca().

Value

Returns the same type as the input. If x is a glyexp_experiment, returns a glyexp_experiment with missing values imputed. If x is a matrix, returns a matrix with missing values imputed.

---

impute_fw_knn *Feature-wise KNN Imputation*

Description

A wrapper around the impute::impute.knn(). Impute missing values with values from the k-nearest neighbors of the corresponding feature.

Usage

```
impute_fw_knn(x, k = 5, by = NULL, ...)

## S3 method for class 'glyexp_experiment'
impute_fw_knn(x, k = 5, by = NULL, ...)

## S3 method for class 'matrix'
impute_fw_knn(x, k = 5, by = NULL, ...)

## Default S3 method:
impute_fw_knn(x, k = 5, by = NULL, ...)
```

Arguments

x	Either a glyexp_experiment object or a matrix. If a matrix, rows should be variables and columns should be samples.
k	The number of nearest neighbors to consider.
by	Either a column name in sample_info (string) or a factor/vector specifying group assignments for each sample. Used for grouping when imputing missing values.
...	Additional arguments to pass to impute::impute.knn().

Value

Returns the same type as the input. If `x` is a `glyexp_experiment`, returns a `glyexp_experiment` with missing values imputed. If `x` is a matrix, returns a matrix with missing values imputed.

`impute_half_sample_min`

Half Sample Minimum Imputation

Description

Impute missing values with half of the minimum value of the corresponding sample. This method assumes that missing values are MCAR, i.e. missing values are induced by an ion below the detection limit. Compared to `impute_sample_min()`, this method is more conservative.

Usage

```
impute_half_sample_min(x)

## S3 method for class 'glyexp_experiment'
impute_half_sample_min(x)

## S3 method for class 'matrix'
impute_half_sample_min(x)

## Default S3 method:
impute_half_sample_min(x)
```

Arguments

`x`	Either a `glyexp_experiment` object or a matrix. If a matrix, rows should be variables and columns should be samples.

Value

Returns the same type as the input. If `x` is a `glyexp_experiment`, returns a `glyexp_experiment` with missing values imputed. If `x` is a matrix, returns a matrix with missing values imputed.

`impute_min_prob` *Minimum Probability Imputation*

Description

Impute missing values using random draws from the left-censored gaussian distribution. Missing values are imputed on the log2 intensity scale and then transformed back to the original scale.

Usage

```
impute_min_prob(x, by = NULL, q = 0.01, tune.sigma = 1, ...)

## S3 method for class 'glyexp_experiment'
impute_min_prob(x, by = NULL, q = 0.01, tune.sigma = 1, ...)

## S3 method for class 'matrix'
impute_min_prob(x, by = NULL, q = 0.01, tune.sigma = 1, ...)

## Default S3 method:
impute_min_prob(x, by = NULL, q = 0.01, tune.sigma = 1, ...)
```

Arguments

`x`	Either a `glyexp_experiment` object or a matrix. If a matrix, rows should be variables and columns should be samples.
`by`	Either a column name in `sample_info` (string) or a factor/vector specifying group assignments for each sample. Used for grouping when imputing missing values.
`q`	Quantile used to estimate the lower-intensity center for each sample. Default is `0.01`.
`tune.sigma`	Non-negative multiplier for the standard deviation of the left-censored draw distribution. Default is `1`.
`...`	Reserved for backward compatibility. Extra arguments are not supported.

Value

Returns the same type as the input. If `x` is a `glyexp_experiment`, returns a `glyexp_experiment` with missing values imputed. If `x` is a matrix, returns a matrix with missing values imputed.

impute_miss_forest *MissForest Imputation*

Description

A wrapper around the `missForest::missForest()`. Impute missing values using recursive running of random forests until convergence. This is a non-parametric method and works for both MAR and MNAR missing data.

Usage

```
impute_miss_forest(x, by = NULL, seed = 123, ...)

## S3 method for class 'glyexp_experiment'
impute_miss_forest(x, by = NULL, seed = 123, ...)

## S3 method for class 'matrix'
impute_miss_forest(x, by = NULL, seed = 123, ...)

## Default S3 method:
impute_miss_forest(x, by = NULL, seed = 123, ...)
```

Arguments

`x`	Either a `glyexp_experiment` object or a matrix. If a matrix, rows should be variables and columns should be samples.
`by`	Either a column name in `sample_info` (string) or a factor/vector specifying group assignments for each sample. Used for grouping when imputing missing values.
`seed`	Integer seed for random number generation. Default is 123.
`...`	Additional arguments to pass to `missForest::missForest()`.

Value

Returns the same type as the input. If `x` is a `glyexp_experiment`, returns a `glyexp_experiment` with missing values imputed. If `x` is a matrix, returns a matrix with missing values imputed.

impute_ppca *PPCA Imputation*

Description

A wrapper around the `pcaMethods::pca()`. Impute missing values using probabilistic principal component analysis (PPCA). PPCA allows to perform PCA on incomplete data and may be used for missing value estimation.

Usage

```
impute_ppca(x, by = NULL, ...)

## S3 method for class 'glyexp_experiment'
impute_ppca(x, by = NULL, ...)

## S3 method for class 'matrix'
impute_ppca(x, by = NULL, ...)

## Default S3 method:
impute_ppca(x, by = NULL, ...)
```

Arguments

x	Either a `glyexp_experiment` object or a matrix. If a matrix, rows should be variables and columns should be samples.
by	Either a column name in `sample_info` (string) or a factor/vector specifying group assignments for each sample. Used for grouping when imputing missing values.
...	Additional arguments to pass to `pcaMethods::pca()`.

Value

Returns the same type as the input. If `x` is a `glyexp_experiment`, returns a `glyexp_experiment` with missing values imputed. If `x` is a matrix, returns a matrix with missing values imputed.

`impute_sample_min` *Sample Minimum Imputation*

Description

Impute missing values with the minimum value of the corresponding sample. This method assumes that missing values are MCAR, i.e. missing values are induced by an ion below the detection limit. See also `impute_half_sample_min()`.

Usage

```
impute_sample_min(x)

## S3 method for class 'glyexp_experiment'
impute_sample_min(x)

## S3 method for class 'matrix'
impute_sample_min(x)

## Default S3 method:
impute_sample_min(x)
```

Arguments

x	Either a glyexp_experiment object or a matrix. If a matrix, rows should be variables and columns should be samples.

Value

Returns the same type as the input. If x is a glyexp_experiment, returns a glyexp_experiment with missing values imputed. If x is a matrix, returns a matrix with missing values imputed.

---

impute_svd *SVD Imputation*

Description

A wrapper around the pcaMethods::pca(). Impute missing values using singular value decomposition (SVD) imputation. SVD is a matrix factorization technique that factors a matrix into three matrices: U, Σ, and V. SVD is used to find the best lower rank approximation of the original matrix.

Usage

```
impute_svd(x, by = NULL, ...)

## S3 method for class 'glyexp_experiment'
impute_svd(x, by = NULL, ...)

## S3 method for class 'matrix'
impute_svd(x, by = NULL, ...)

## Default S3 method:
impute_svd(x, by = NULL, ...)
```

Arguments

x	Either a glyexp_experiment object or a matrix. If a matrix, rows should be variables and columns should be samples.
by	Either a column name in sample_info (string) or a factor/vector specifying group assignments for each sample. Used for grouping when imputing missing values.
...	Additional arguments to pass to pcaMethods::pca().

Value

Returns the same type as the input. If x is a glyexp_experiment, returns a glyexp_experiment with missing values imputed. If x is a matrix, returns a matrix with missing values imputed.

`impute_sw_knn` *Sample-wise KNN Imputation*

Description

A wrapper around the `impute::impute.knn()`. Impute missing values with values from the k-nearest neighbors of the corresponding sample. If there are strong patterns among the samples (such as group clustering relationships or experimental conditions), this method can better utilize the overall relationships among samples.

Usage

```
impute_sw_knn(x, k = 5, by = NULL, ...)

## S3 method for class 'glyexp_experiment'
impute_sw_knn(x, k = 5, by = NULL, ...)

## S3 method for class 'matrix'
impute_sw_knn(x, k = 5, by = NULL, ...)

## Default S3 method:
impute_sw_knn(x, k = 5, by = NULL, ...)
```

Arguments

`x`	Either a `glyexp_experiment` object or a matrix. If a matrix, rows should be variables and columns should be samples.
`k`	The number of nearest neighbors to consider.
`by`	Either a column name in `sample_info` (string) or a factor/vector specifying group assignments for each sample. Used for grouping when imputing missing values.
`...`	Additional arguments to pass to `impute::impute.knn()`.

Value

Returns the same type as the input. If `x` is a `glyexp_experiment`, returns a `glyexp_experiment` with missing values imputed. If `x` is a matrix, returns a matrix with missing values imputed.

impute_zero *Zero Imputation*

Description

Impute missing values in an expression matrix by replacing them with zeros.

Usage

```
impute_zero(x)

## S3 method for class 'glyexp_experiment'
impute_zero(x)

## S3 method for class 'matrix'
impute_zero(x)

## Default S3 method:
impute_zero(x)
```

Arguments

x	Either a `glyexp_experiment` object or a matrix. If a matrix, rows should be variables and columns should be samples.

Value

Returns the same type as the input. If `x` is a `glyexp_experiment`, returns a `glyexp_experiment` with missing values imputed. If `x` is a matrix, returns a matrix with missing values imputed.

normalize_loesscyc *LoessCyc Normalization*

Description

This function is a wrapper around `limma::normalizeCyclicLoess()` with `method = "pairs"`. Each pair of columns is normalized mutually to each other. See this paper for more information. Also see `limma::normalizeCyclicLoess()`.

Usage

```
normalize_loesscyc(x, by = NULL, ...)

## S3 method for class 'glyexp_experiment'
normalize_loesscyc(x, by = NULL, ...)
```

```
## S3 method for class 'matrix'
normalize_loesscyc(x, by = NULL, ...)

## Default S3 method:
normalize_loesscyc(x, by = NULL, ...)
```

Arguments

x	Either a glyexp_experiment object or a matrix. If a matrix, rows should be variables and columns should be samples.
by	Either a column name in sample_info (string) or a factor/vector specifying group assignments for each sample. Optional. If provided, the normalization will be performed within each group.
...	Additional arguments to pass to limma::normalizeCyclicLoess().

Value

Returns the same type as the input. If x is a glyexp_experiment, returns a glyexp_experiment with normalized expression matrix. If x is a matrix, returns a normalized matrix.

---

normalize_loessf *LoessF Normalization*

Description

This function is a wrapper around limma::normalizeCyclicLoess() with method = "fast". Each column is simply normalized to a reference array, the reference array being the average of all the arrays. See this paper for more information. Also see limma::normalizeCyclicLoess().

Usage

```
normalize_loessf(x, by = NULL, ...)

## S3 method for class 'glyexp_experiment'
normalize_loessf(x, by = NULL, ...)

## S3 method for class 'matrix'
normalize_loessf(x, by = NULL, ...)

## Default S3 method:
normalize_loessf(x, by = NULL, ...)
```

Arguments

x	Either a glyexp_experiment object or a matrix. If a matrix, rows should be variables and columns should be samples.

by	Either a column name in sample_info (string) or a factor/vector specifying group assignments for each sample. Optional. If provided, the normalization will be performed within each group.
...	Additional arguments to pass to limma::normalizeCyclicLoess().

Value

Returns the same type as the input. If x is a glyexp_experiment, returns a glyexp_experiment with normalized expression matrix. If x is a matrix, returns a normalized matrix.

---

normalize_median *Median Normalization*

Description

Normalize the expression matrix by dividing each column (sample) by the median of that column (NA ignored), so that the median of each column is 1. This is the most common normalization method for proteomics data. It effectively and robustly removes the bias introduced by total protein abundance, and removes batch effects in part.

Usage

```
normalize_median(x)

## S3 method for class 'glyexp_experiment'
normalize_median(x)

## S3 method for class 'matrix'
normalize_median(x)

## Default S3 method:
normalize_median(x)
```

Arguments

x	Either a glyexp_experiment object or a matrix. If a matrix, rows should be variables and columns should be samples.

Value

Returns the same type as the input. If x is a glyexp_experiment, returns a glyexp_experiment with normalized expression matrix. If x is a matrix, returns a normalized matrix.

`normalize_median_abs` *Absolute Median Normalization*

Description

Normalize the expression matrix by dividing each column (sample) by the absolute median of that column (NA ignored), so that the absolute median of each column is 1.

Usage

```
normalize_median_abs(x)

## S3 method for class 'glyexp_experiment'
normalize_median_abs(x)

## S3 method for class 'matrix'
normalize_median_abs(x)

## Default S3 method:
normalize_median_abs(x)
```

Arguments

`x`	Either a `glyexp_experiment` object or a matrix. If a matrix, rows should be variables and columns should be samples.

Value

Returns the same type as the input. If `x` is a `glyexp_experiment`, returns a `glyexp_experiment` with normalized expression matrix. If `x` is a matrix, returns a normalized matrix.

`normalize_median_quotient`
Median Quotient Normalization

Description

This approach is based on the calculation of the dilution factor of each sample with respect to a reference sample. Here, the reference sample was calculated as the median value of each glycan's abundance across all measured samples. For each sample, a vector of quotients was then obtained by dividing each glycan measure by the corresponding value in the reference sample. The median of these quotients was then used as the sample's dilution factor, and the original sample values were subsequently divided by that value. The underlying assumption is that the different intensities observed across individuals are imputable to different amounts of the biological material in the collected samples. See this paper for more information.

Usage

```
normalize_median_quotient(x, by = NULL)

## S3 method for class 'glyexp_experiment'
normalize_median_quotient(x, by = NULL)

## S3 method for class 'matrix'
normalize_median_quotient(x, by = NULL)

## Default S3 method:
normalize_median_quotient(x, by = NULL)
```

Arguments

x	Either a `glyexp_experiment` object or a matrix. If a matrix, rows should be variables and columns should be samples.
by	Either a column name in `sample_info` (string) or a factor/vector specifying group assignments for each sample. Optional. If provided, the normalization will be performed within each group.

Value

Returns the same type as the input. If `x` is a `glyexp_experiment`, returns a `glyexp_experiment` with normalized expression matrix. If `x` is a matrix, returns a normalized matrix.

`normalize_quantile` *Quantile Normalization*

Description

This function is a wrapper around `limma::normalizeQuantiles()`. It normalizes the expression matrix by quantile normalization. This method is used to remove technical variation between samples. Proteomics data rarely uses this method, but it is common in microarray data. See wikipedia for more information.

Usage

```
normalize_quantile(x, by = NULL, ...)

## S3 method for class 'glyexp_experiment'
normalize_quantile(x, by = NULL, ...)

## S3 method for class 'matrix'
normalize_quantile(x, by = NULL, ...)

## Default S3 method:
normalize_quantile(x, by = NULL, ...)
```

Arguments

x	Either a glyexp_experiment object or a matrix. If a matrix, rows should be variables and columns should be samples.
by	Either a column name in sample_info (string) or a factor/vector specifying group assignments for each sample. Optional. If provided, the normalization will be performed within each group.
...	Additional arguments to pass to limma::normalizeQuantiles().

Value

Returns the same type as the input. If x is a glyexp_experiment, returns a glyexp_experiment with normalized expression matrix. If x is a matrix, returns a normalized matrix.

---

normalize_rlr *Robust Linear Regression Normalization*

Description

This method is based on robust linear regression. The reference sample is calculated as the median value of each variable's abundance across all measured samples. For each sample, a robust linear regression model is fitted to the sample's abundance values against the reference sample's abundance values. The fitted model is then used to normalize the sample's abundance values. The underlying assumption is that the different intensities observed across individuals are imputable to different amounts of the biological material in the collected samples.

Usage

```
normalize_rlr(x, by = NULL)

## S3 method for class 'glyexp_experiment'
normalize_rlr(x, by = NULL)

## S3 method for class 'matrix'
normalize_rlr(x, by = NULL)

## Default S3 method:
normalize_rlr(x, by = NULL)
```

Arguments

x	Either a glyexp_experiment object or a matrix. If a matrix, rows should be variables and columns should be samples.
by	Either a column name in sample_info (string) or a factor/vector specifying group assignments for each sample. Optional. If provided, the normalization will be performed within each group.

Value

Returns the same type as the input. If `x` is a `glyexp_experiment`, returns a `glyexp_experiment` with normalized expression matrix. If `x` is a matrix, returns a normalized matrix.

`normalize_rlrma` *Robust Linear Regression with Median Adjustment Normalization*

Description

This method is based on robust linear regression with median adjustment. First, the median of each variable's abundance across all measured samples is subtracted from the sample's abundance values. Then, it's like the `normalize_rlr()` method.

Usage

```
normalize_rlrma(x, by = NULL)

## S3 method for class 'glyexp_experiment'
normalize_rlrma(x, by = NULL)

## S3 method for class 'matrix'
normalize_rlrma(x, by = NULL)

## Default S3 method:
normalize_rlrma(x, by = NULL)
```

Arguments

`x`	Either a `glyexp_experiment` object or a matrix. If a matrix, rows should be variables and columns should be samples.
`by`	Either a column name in `sample_info` (string) or a factor/vector specifying group assignments for each sample. Optional. If provided, the normalization will be performed within each group.

Value

Returns the same type as the input. If `x` is a `glyexp_experiment`, returns a `glyexp_experiment` with normalized expression matrix. If `x` is a matrix, returns a normalized matrix.

`normalize_rlrmacyc` *Robust Linear Regression with Median Adjustment and Cyclic Normalization*

Description

This method is based on robust linear regression with median adjustment and cyclic normalization. The method is applied iteratively to each pair of samples. For each pair of samples, the median of the differences between the two samples is calculated. A robust linear regression model is fitted to the differences against the averages of the two samples. The fitted model is then used to normalize the two samples. The process is repeated for a number of iterations.

Usage

```
normalize_rlrmacyc(x, n_iter = 3, by = NULL)

## S3 method for class 'glyexp_experiment'
normalize_rlrmacyc(x, n_iter = 3, by = NULL)

## S3 method for class 'matrix'
normalize_rlrmacyc(x, n_iter = 3, by = NULL)

## Default S3 method:
normalize_rlrmacyc(x, n_iter = 3, by = NULL)
```

Arguments

`x`	Either a `glyexp_experiment` object or a matrix. If a matrix, rows should be variables and columns should be samples.
`n_iter`	The number of iterations to perform. Default is 3.
`by`	Either a column name in `sample_info` (string) or a factor/vector specifying group assignments for each sample. Optional. If provided, the normalization will be performed within each group.

Value

Returns the same type as the input. If `x` is a `glyexp_experiment`, returns a `glyexp_experiment` with normalized expression matrix. If `x` is a matrix, returns a normalized matrix.

normalize_total_area *Total Area Normalization*

Description

Normalize the expression matrix by dividing each column (sample) by the sum of that column, so that the sum of each column is 1. This is the most common normalization method for glycomics data. It removes the bias introduced by total glycan abundance. However, it results in compositional data, which may result in unrealistic downstream analysis results.

Usage

```
normalize_total_area(x)

## S3 method for class 'glyexp_experiment'
normalize_total_area(x)

## S3 method for class 'matrix'
normalize_total_area(x)

## Default S3 method:
normalize_total_area(x)
```

Arguments

x	Either a `glyexp_experiment` object or a matrix. If a matrix, rows should be variables and columns should be samples.

Value

Returns the same type as the input. If `x` is a `glyexp_experiment`, returns a `glyexp_experiment` with normalized expression matrix. If `x` is a matrix, returns a normalized matrix.

normalize_vsn *Variance Stabilizing Normalization*

Description

This function is a wrapper around `limma::normalizeVSN()`. Evidence proved that this method performs well in reducing noise and boosting differential expression detection. Log-transformation is not needed for downstream statistical analysis, for this normalization method performs a log-like transformation internally. Due to the same reason, fold change estimates will be severely distorted. Please use this method with caution. See this paper for more information.

Usage

```
normalize_vsn(x, by = NULL, ...)

## S3 method for class 'glyexp_experiment'
normalize_vsn(x, by = NULL, ...)

## S3 method for class 'matrix'
normalize_vsn(x, by = NULL, ...)

## Default S3 method:
normalize_vsn(x, by = NULL, ...)
```

Arguments

x	Either a `glyexp_experiment` object or a matrix. If a matrix, rows should be variables and columns should be samples.
by	Either a column name in `sample_info` (string) or a factor/vector specifying group assignments for each sample. Optional. If provided, the normalization will be performed within each group.
...	Additional arguments to pass to `limma::normalizeVSN()`.

Details

At least 42 variables should be provided for this method. This follows the convention of the `vsn` package.

Value

Returns the same type as the input. If `x` is a `glyexp_experiment`, returns a `glyexp_experiment` with normalized expression matrix. If `x` is a matrix, returns a normalized matrix.

`plot_batch_pca` *Plot PCA Score by Batch*

Description

Draw a PCA score plot for samples and color points by batch. PCA is computed on log2-transformed intensities after removing variables with missing values.

Usage

```
plot_batch_pca(exp, batch_col = "batch")
```

Arguments

exp	A `glyexp::experiment()` object.
batch_col	Column name in `sample_info`, or a factor/vector with length equal to the number of samples.

Value

A ggplot object of PCA scores.

Examples

```
exp <- glyexp::toy_experiment
exp$sample_info$batch <- rep(c("A", "B"), each = 3)
plot_batch_pca(exp, batch_col = "batch")
```

---

plot_cv_dent *Plot CV Density*

Description

Compute coefficient of variation (CV) for each variable and plot its density. When `by` is provided, CVs are computed within each group and densities are shown with different fills.

Usage

```
plot_cv_dent(exp, by = NULL)
```

Arguments

`exp`	A `glyexp::experiment()` object.
`by`	Grouping variable for samples. Can be a column name in `sample_info` or a vector/factor with length equal to the number of samples. When provided, CVs are computed within each group and densities are shown with different fills.

Value

A ggplot object of CV density.

Examples

```
plot_cv_dent(glyexp::toy_experiment)
exp <- glyexp::toy_experiment
exp$sample_info$group <- rep(c("A", "B"), length.out = ncol(exp$expr_mat))
plot_cv_dent(exp, by = "group")
```

plot_int_boxplot *Plot Log-Intensity Boxplots by Sample*

Description

Draw boxplots of log2-transformed intensities for each sample. Optionally color and group samples by a metadata variable.

Usage

```
plot_int_boxplot(exp, by = NULL)
```

Arguments

exp	A glyexp::experiment() object.
by	Grouping variable for samples. Can be a column name in sample_info or a vector/factor with length equal to the number of samples. When provided, samples are grouped along the x-axis and boxplots are colored by group.

Value

A ggplot object of log-intensity boxplots.

Examples

```
plot_int_boxplot(glyexp::toy_experiment)
plot_int_boxplot(glyexp::toy_experiment, by = "group")
```

plot_missing_bar *Plot Missing Value Proportions by Sample or Variable*

Description

Draw a bar plot of missing value proportions for each sample or variable. Items are ordered from low to high missing proportion.

Usage

```
plot_missing_bar(exp, on = "sample")
```

Arguments

exp	A glyexp::experiment() object.
on	Whether to plot missingness by "sample(s)" or "variable(s)". Defaults to "sample".

Value

A ggplot object of missing value proportions by item.

Examples

```
plot_missing_bar(glyexp::toy_experiment)
```

`plot_missing_heatmap` *Plot Missing Value Heatmap*

Description

Draw a binary heatmap showing the missing value pattern in an experiment. Values are binarized: 1 (present) or 0 (missing). Rows (variables) are sorted by missing value proportion from low to high. Columns (samples) are clustered using hierarchical clustering.

Usage

```
plot_missing_heatmap(exp, ...)
```

Arguments

`exp`	A `glyexp::experiment()` object.
`...`	Other arguments passed to `pheatmap::pheatmap()`.

Value

A ggplot object of the missing value heatmap.

Examples

```
plot_missing_heatmap(glyexp::toy_experiment)
```

plot_rank_abundance *Plot Rank Abundance*

Description

Draw a scatter plot of proteins ranked by mean log2 intensity. Proteins are ordered from high to low mean intensity along the x-axis.

Usage

```
plot_rank_abundance(exp)
```

Arguments

exp	A glyexp::experiment() object.

Value

A ggplot object of protein rank abundance.

Examples

```
plot_rank_abundance(glyexp::toy_experiment)
```

plot_rep_scatter *Plot Replicate Scatter Plots*

Description

Randomly draw replicate sample pairs and plot log2 intensity scatter plots. The plot title shows sample names, and the subtitle reports the R2 value.

Usage

```
plot_rep_scatter(exp, rep_col, n_pairs = 9)
```

Arguments

exp	A glyexp::experiment() object.
rep_col	Column name in sample_info used to define replicate groups. Samples with the same value in this column are treated as replicates (e.g. c("A", "A", "A", "B", "B", "B") indicates three replicates for sample A and three for sample B).
n_pairs	Number of replicate pairs to draw at random.

Value

A patchwork object containing replicate scatter plots.

Examples

```
exp <- glyexp::toy_experiment
exp$sample_info$replicate <- rep(c("A", "B"), length.out = ncol(exp$expr_mat))
plot_rep_scatter(exp, rep_col = "replicate", n_pairs = 4)
```

---

plot_rle *Plot Relative Log Expression (RLE) Boxplots*

Description

Draw boxplots of relative log expression (log2 intensity minus row median) for each sample. Optionally color and group samples by a metadata variable.

Usage

```
plot_rle(exp, by = NULL)
```

Arguments

exp	A glyexp::experiment() object.
by	Grouping variable for samples. Can be a column name in sample_info or a vector/factor with length equal to the number of samples. When provided, samples are grouped along the x-axis and boxplots are colored by group.

Value

A ggplot object of RLE boxplots.

Examples

```
plot_rle(glyexp::toy_experiment)
plot_rle(glyexp::toy_experiment, by = "group")
```

plot_tic_bar *Plot Total Intensity by Sample*

Description

Draw a bar plot showing total intensity (TIC) for each sample. Samples are ordered from high to low TIC from left to right.

Usage

```
plot_tic_bar(exp)
```

Arguments

exp	A glyexp::experiment() object.

Value

A ggplot object of total intensity by sample.

Examples

```
plot_tic_bar(glyexp::toy_experiment)
```

remove_constant *Remove Constant Variables*

Description

Constant variables are variables with the same value in all samples. This function is equivalent to `remove_low_var(x, var_cutoff = 0, by = by, strict = strict)`.

Usage

```
remove_constant(x, by = NULL, strict = FALSE)
```

Arguments

x	Either a `glyexp_experiment` object or a matrix.
by	Either a column name in `sample_info` (string) or a vector specifying group assignments for each sample.
strict	If `FALSE`, remove a variable only if it is constant in all groups. If `TRUE`, remove a variable if it is constant in any group. Defaults to FALSE.

Value

For `glyexp_experiment` input, returns a modified `glyexp_experiment` object. For matrix input, returns a filtered matrix.

See Also

`remove_low_var()`

`remove_low_cv` *Remove Variables with Low Coefficient of Variation*

Description

Filters variables whose coefficient of variation falls below a threshold. Default behavior is to remove variables with zero coefficient of variation.

Usage

```
remove_low_cv(x, cv_cutoff = 0, by = NULL, strict = FALSE)

## S3 method for class 'glyexp_experiment'
remove_low_cv(x, cv_cutoff = 0, by = NULL, strict = FALSE)

## S3 method for class 'matrix'
remove_low_cv(x, cv_cutoff = 0, by = NULL, strict = FALSE)

## Default S3 method:
remove_low_cv(x, cv_cutoff = 0, by = NULL, strict = FALSE)
```

Arguments

`x`	Either a `glyexp_experiment` object or a matrix.
`cv_cutoff`	The cutoff for coefficient of variation. Defaults to 0.
`by`	A factor specifying the groupings. Defaults to NULL.
`strict`	If `FALSE`, remove a variable only if it passes the coefficient of variation threshold in all groups. If `TRUE`, remove a variable if it passes the coefficient of variation threshold in any group.

Value

For `glyexp_experiment` input, returns a modified `glyexp_experiment` object. For matrix input, returns a filtered matrix.

See Also

`remove_low_var()`

remove_low_expr *Remove Variables with Low Expression*

Description

Filters variables based on median expression values. Variables with median expression values below certain percentile will be removed.

Usage

```
remove_low_expr(x, percentile = 0.05, by = NULL, strict = FALSE)

## S3 method for class 'glyexp_experiment'
remove_low_expr(x, percentile = 0.05, by = NULL, strict = FALSE)

## S3 method for class 'matrix'
remove_low_expr(x, percentile = 0.05, by = NULL, strict = FALSE)

## Default S3 method:
remove_low_expr(x, percentile = 0.05, by = NULL, strict = FALSE)
```

Arguments

x	Either a `glyexp_experiment` object or a matrix.
percentile	The percentile for median expression values. Defaults to 0.05, i.e., the 5% lowest median expression values will be removed.
by	Either a column name in `sample_info` (string) or a vector specifying group assignments for each sample.
strict	If `FALSE`, remove a variable only if it passes the abundance thresholds in all groups. If `TRUE`, remove a variable if it passes the abundance thresholds in any group. Defaults to FALSE.

Value

For `glyexp_experiment` input, returns a modified `glyexp_experiment` object. For matrix input, returns a filtered matrix.

remove_low_var *Remove Variables with Low Variance Remove Variables with Low Variance*

Description

Filters variables whose variance falls below a threshold. Default behavior is to remove variables with zero variance.

Usage

```
remove_low_var(x, var_cutoff = 0, by = NULL, strict = FALSE)

## S3 method for class 'glyexp_experiment'
remove_low_var(x, var_cutoff = 0, by = NULL, strict = FALSE)

## S3 method for class 'matrix'
remove_low_var(x, var_cutoff = 0, by = NULL, strict = FALSE)

## Default S3 method:
remove_low_var(x, var_cutoff = 0, by = NULL, strict = FALSE)
```

Arguments

x	Either a glyexp_experiment object or a matrix.
var_cutoff	The cutoff for variance. Defaults to 0.
by	A factor specifying the groupings. Defaults to NULL.
strict	If FALSE, remove a variable only if it passes the variance threshold in all groups. If TRUE, remove a variable if it passes the variance threshold in any group.

Value

For glyexp_experiment input, returns a modified glyexp_experiment object. For matrix input, returns a filtered matrix.

See Also

remove_low_cv(), remove_constant()

---

remove_rare *Remove Rare Variables with Too Many Missing Values*

Description

Remove Rare Variables with Too Many Missing Values

Usage

```
remove_rare(x, prop = NULL, n = NULL, by = NULL, strict = FALSE, min_n = NULL)

## S3 method for class 'glyexp_experiment'
remove_rare(x, prop = NULL, n = NULL, by = NULL, strict = FALSE, min_n = NULL)

## S3 method for class 'matrix'
remove_rare(x, prop = NULL, n = NULL, by = NULL, strict = FALSE, min_n = NULL)
```

```
## Default S3 method:
remove_rare(x, prop = NULL, n = NULL, by = NULL, strict = FALSE, min_n = NULL)
```

Arguments

x	Either a glyexp_experiment object or a matrix. If a matrix, rows should be variables and columns should be samples.
prop	The proportion of missing values to use as a threshold. Variables with missing values above this threshold will be removed. Defaults to 0.5.
n	The number of missing values to use as a threshold. An alternative to prop.
by	Either a column name in sample_info (string) or a factor/vector specifying group assignments for each sample. Missing value counts or proportions will be calculated within each group.
strict	Works with by. If FALSE, remove a variable only if it passes the missing threshold in all groups. If TRUE, remove a variable if it passes the missing threshold in any group. See examples for more details.
min_n	The minimum number of non-missing values required for a variable to be kept. If NULL (default), it is calculated dynamically: • For datasets with 1, 2, or 3 samples: min_n equals the sample count • For datasets with >3 samples: min_n = 3 • When using by, the rule is applied within each group

Value

For glyexp_experiment input, returns a modified glyexp_experiment object. For matrix input, returns a filtered matrix.

Examples

```
# With glyexp_experiment
exp <- glyexp::toy_experiment
exp$expr_mat[1, 1] <- NA    # V1: 1/6 missing
exp$expr_mat[2, 1:3] <- NA  # V2: 3/6 missing
exp$expr_mat[3, 1:5] <- NA  # V3: 5/6 missing
exp$expr_mat[4, 1:6] <- NA  # V4: 6/6 missing
exp$expr_mat

# Remove variables with more than 50% missing values.
remove_rare(exp, prop = 0.5)$expr_mat

# Remove variables with more than 2 missing values.
remove_rare(exp, n = 2)$expr_mat

# Remove variables if they have more than 1 missing value in all groups.
# In another word, keep variables as long as they have 1 or 0 missing value
# in any group.
remove_rare(exp, by = "group", strict = FALSE)$expr_mat
```

```
# Keep only variables with no missing values.
remove_rare(exp, prop = 0)$expr_mat

# Use custom min_n to require at least 4 non-missing values
remove_rare(exp, min_n = 4)$expr_mat

# With matrix
mat <- matrix(c(1, 2, NA, 4, 5, NA, 7, 8, 9), nrow = 3)
mat_filtered <- remove_rare(mat, prop = 0.5)
```

`transform_alr` *Additive Log-Ratio Transformation*

Description

This function implements a glycoWork-compatible ALR preprocessing strategy. Internally, the data are transformed relative to an automatically selected reference glycan using glycoWork-style ALR logic, then back-transformed to the original ratio space before returning the result.

Usage

```
transform_alr(x, by = NULL, gamma = 0.1, group_scales = NULL)

## S3 method for class 'glyexp_experiment'
transform_alr(x, by = NULL, gamma = 0.1, group_scales = NULL)

## S3 method for class 'matrix'
transform_alr(x, by = NULL, gamma = 0.1, group_scales = NULL)

## Default S3 method:
transform_alr(x, by = NULL, gamma = 0.1, group_scales = NULL)
```

Arguments

`x`	Either a `glyexp_experiment` object or a matrix. If a matrix, rows should be variables and columns should be samples.
`by`	Either a column name in `sample_info` (for `glyexp_experiment` input) or a factor/vector with one value per sample.
`gamma`	Standard deviation of the scale-uncertainty model on the `log2` scale. Default is `0.1`. Set to `0` for deterministic transformation.
`group_scales`	Optional informed group scales. For binary comparisons, this can be a single positive ratio for the second group relative to the first, or two positive scales from which that ratio is derived. For multi-group data, provide a positive vector with one scale per group.

Value

Returns the same type as the input. If `x` is a `glyexp_experiment`, returns a `glyexp_experiment` with an ALR-transformed expression matrix. If `x` is a matrix, returns an ALR-transformed matrix. When ALR succeeds, the reference glycan is excluded from the result and the output therefore has one fewer row than the input. When ALR falls back to CLR, the returned object keeps the original dimensions. The returned values are back-transformed to the original ratio space, corresponding to `x / x_ref`.

Algorithmic details

The stochastic branch follows `glycowork`: when `gamma > 0`, CLR noise is sampled per feature-sample entry rather than once per sample. Without informed scales, this corresponds to jittering around the sample-specific `log2` geometric mean of the non-zero components and then returning to ratio space.

The `group_scales` argument is interpreted in the same way as `glycowork`'s `custom_scale`. For exactly two groups, provide the total-signal ratio of the second group relative to the first, either directly as a scalar or implicitly through two per-group scales. For multi-group data, provide one positive scale per group; these scales are used only in the stochastic branch.

Motif quantification

If you intend to use ALR/CLR transformation with motif quantification, you should apply the transformation after motif quantification rather than before. In another words, the recommended workflow is:

1. Perform data preprocessing.

```
clean_exp <- auto_clean(exp)
```

1. Perform motif quantification on the cleaned experiment.

```
motif_exp <- glydet::quantify_motifs(clean_exp, motifs)
```

1. Perform ALR/CLR transformation on the motif-quantified experiment.

```
coda_motif_exp <- auto_coda(motif_exp, by = "group", gamma = 0.1)
```

1. Proceed with downstream analyses on the transformed motif experiment.

```
dea_res <- glystats::gly_ttest(coda_motif_exp)
```

`transform_clr` *Centered Log-Ratio Transformation*

Description

This function implements a glycoWork-compatible CLR preprocessing strategy. Internally, the data are transformed on the `log2` scale following `glycowork::clr_transformation()`, then back-transformed to the original ratio space before returning the result.

Usage

```
transform_clr(x, by = NULL, gamma = 0.1, group_scales = NULL)

## S3 method for class 'glyexp_experiment'
transform_clr(x, by = NULL, gamma = 0.1, group_scales = NULL)

## S3 method for class 'matrix'
transform_clr(x, by = NULL, gamma = 0.1, group_scales = NULL)

## Default S3 method:
transform_clr(x, by = NULL, gamma = 0.1, group_scales = NULL)
```

Arguments

`x`	Either a `glyexp_experiment` object or a matrix. If a matrix, rows should be variables and columns should be samples.
`by`	Either a column name in `sample_info` (for `glyexp_experiment` input) or a factor/vector with one value per sample.
`gamma`	Standard deviation of the scale-uncertainty model on the `log2` scale. Default is `0.1`. Set to `0` for deterministic transformation.
`group_scales`	Optional informed group scales. For binary comparisons, this can be a single positive ratio for the second group relative to the first, or two positive scales from which that ratio is derived. For multi-group data, provide a positive vector with one scale per group.

Value

Returns the same type as the input. If `x` is a `glyexp_experiment`, returns a `glyexp_experiment` with transformed expression matrix. If `x` is a matrix, returns a transformed matrix. The returned values are back-transformed to the original ratio space. Zeros in the input therefore remain zeros in the output.

Algorithmic details

The stochastic branch follows `glycowork`: when `gamma > 0`, CLR noise is sampled per feature-sample entry rather than once per sample. Without informed scales, this corresponds to jittering around the sample-specific `log2` geometric mean of the non-zero components and then returning to ratio space.

The `group_scales` argument is interpreted in the same way as `glycowork`'s `custom_scale`. For exactly two groups, provide the total-signal ratio of the second group relative to the first, either directly as a scalar or implicitly through two per-group scales. For multi-group data, provide one positive scale per group; these scales are used only in the stochastic branch.

Motif quantification

If you intend to use ALR/CLR transformation with motif quantification, you should apply the transformation after motif quantification rather than before. In another words, the recommended workflow is:

1. Perform data preprocessing.

```
clean_exp <- auto_clean(exp)
```

1. Perform motif quantification on the cleaned experiment.

```
motif_exp <- glydet::quantify_motifs(clean_exp, motifs)
```

1. Perform ALR/CLR transformation on the motif-quantified experiment.

```
coda_motif_exp <- auto_coda(motif_exp, by = "group", gamma = 0.1)
```

1. Proceed with downstream analyses on the transformed motif experiment.

```
dea_res <- glystats::gly_ttest(coda_motif_exp)
```

Index