Package 'TiDEomics'

August 25, 2026

Title Time-course Differential Expression analysis of omics data

Version 0.99.5

Date 2026-07-31

Description TiDEomics provides a comprehensive workflow for multi-group time-course omics data analysis, analysing time-dominant, group-dominant, and group-specific temporal effects through pairwise differential expression, variance decomposition, and co-expression module analysis (WGCNA). The package integrates quality control, data processing, functional enrichment, and extensive visualisation. It supports datasets with missing values (e.g., mass spectrometry-based proteomics), and operates on SummarizedExperiment objects to ensure compatibility with the Bioconductor ecosystem.

License GPL (>= 2)

URL https://github.com/hte123/TiDEomics, https://hte123.github.io/TiDEomics

BugReports https://github.com/hte123/TiDEomics/issues

biocViews Software, GeneExpression, DifferentialExpression, Proteomics, Transcriptomics, TimeCourse, MultipleComparison, Pathways, Visualization, MassSpectrometry, QualityControl

Encoding UTF-8

Roxygen list(markdown = TRUE)

Imports circlize, clusterProfiler, ComplexHeatmap, dplyr, enrichplot, ggforce, ggh4x, ggplot2, ggplotify, ggpubr, ggrepel, ggridges, ggsci, limma, lme4, methods, patchwork, pbapply, PCAtools, randtests, scales, SummarizedExperiment, tibble, tidyr, Trendy, umap, WGCNA

Depends R (>= 4.6.0)

LazyData false

Suggests knitr, rmarkdown, BiocStyle, testthat (>= 3.1.0), plotly, enrichR, org.Hs.eg.db, org.Mm.eg.db, msigdbr, DeeDeeExperiment

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

git_url https://git.bioconductor.org/packages/TiDEomics

git_branch devel

git_last_commit ca26312

git_last_commit_date 2026-07-31

Repository Bioconductor 3.24

Date/Publication 2026-08-24

Author Tianen He [aut, cre] (ORCID: <https://orcid.org/0000-0001-6864-0723>)

Maintainer Tianen He <tianen.he@ndm.ox.ac.uk>

Contents

`TiDEomics-package` *TiDEomics: Time-course Differential Expression analysis of omics data*

Description

TiDEomics provides a workflow for **multi-group time-course** omics data analysis, analysing **time-dominant**, **group-dominant**, and **group-specific temporal** effects through pairwise differential expression, variance decomposition, and co-expression module analysis (WGCNA). A **residual variance** filtering strategy prioritises features with structured differential expression. It supports datasets with missing values (e.g. mass spectrometry-based proteomics) and operates on `SummarizedExperiment` objects for compatibility with the Bioconductor ecosystem.

Details

Key functionality:

- **Data preparation & normalisation:** `create_input()`, `prepare_tide()`, `split_groups()`, `merge_replicates()`, `merge_groups()`, `normalise_to_start()`, `impute_groups()`
- **Quality control & exploration:** `plot_missing()`, `plot_distribution()`, `plot_ID()`, `plot_cv()`, `plot_cor_matrix()`, `plot_pca()`, `plot_pca_3D()`, `plot_pca_arrows()`, `plot_pca_by_group()`, `plot_umap()`, `plot_umap_by_group()`
- **Feature properties & variance decomposition:** `calc_feature_property()`, `summarise_feature_property()` `group_specific_features()`, `decomp_variance()`, `plot_variance()`, `plot_trend()`
- **Pairwise differential expression:** `DE_between_group()`, `DE_between_time()`, `plot_DE_between_group()`, `plot_DE_between_time()`, `plot_volcano()`
- **Segmentation regression with Trendy:** `run_Trendy()`, `summarise_Trendy()`, `plot_segments()`, `plot_breakpoints()`
- **Co-expression module identification with WGCNA:** `prepare_WGCNA()`, `run_WGCNA()`, `WGCNA_module()`, `extract_hubs()`, `plot_WGCNA()`, `plot_modules_h()`, `plot_modules_v()`, `summarise_module_pattern()`, `summarise_module_metrics()`
- **Functional enrichment:** `enrichGO_list()`, `enrichGO_rank()`, `enrichR_list()`, `enrich_msigdb()`, `plot_GO()`
- **Interoperability & settings:** `flatten_DE()`, `flatten_enrich()`, `set_custom_palette()`, `get_custom_palette()`, `theme_custom()`

Two experimental designs are auto-detected from the sample annotation: independent samples and repeated measures.

See `vignette("TiDEomics")` for a step-by-step tutorial.

Author(s)

Maintainer: Tianen He <tianen.he@ndm.ox.ac.uk> (ORCID)

Authors:

- Tianen He <tianen.he@ndm.ox.ac.uk> (ORCID)

See Also

Useful links:

- https://hte123.github.io/TiDEomics
- https://github.com/hte123/TiDEomics
- Report bugs at https://github.com/hte123/TiDEomics/issues

`.anno_multicol_textbox` *Multi-column textbox row annotation for ComplexHeatmap*

Description

Row annotation that displays multiple enrichment categories side by side in a single annotation block. Follows the same interface convention as `ComplexHeatmap:::anno_textbox`. Column headers are drawn above the first module via `decorate_annotation`-style header placement.

Usage

```
.anno_multicol_textbox(
  align_to,
  text,
  background_gp = grid::gpar(fill = "#DDDDDD", col = "#AAAAAA"),
  which = c("row", "column"),
  by = "anno_link",
  side = c("right", "left"),
  ...
)
```

Arguments

`align_to`	A vector of group labels, a list of indices, or a named list mapping groups to row indices (same as `ComplexHeatmap::anno_textbox`).
`text`	A list of per-module enrichment terms, where each element is a named list of category term data.frames (columns `text`, `col`, `fontsize`). Output from `.reformat_cat_terms`.
`background_gp`	Background graphical parameters for the annotation block (fill, col, lty, lwd).
`which`	`"row"` (only row annotations are supported).
`by`	Link type: `"anno_link"` (default, aligns to heatmap rows) or `"anno_block"`.
`side`	`"right"` (default) or `"left"`.
`...`	Additional arguments passed to `.multicol_textbox_grob`. Notable options: `show_headers` (logical), `header_gp` (gpar for header text), `header_offset` (unit offset above annotation).

Value

A ComplexHeatmap annotation object (from `anno_link` or `anno_block`).

`.build_marked_features_anno` *Build marked-feature term_list for multi-column textbox*

Description

Build a per-module term_list from `mark_features` for use as a column in `.anno_multicol_textbox`. Accepts a named list (per-category colours from `ggsci::pal_jco()`) or a character vector (all black). Colours are taken from `mark_state$feat_col`.

Usage

```
.build_marked_features_anno(
  mark_features,
  module_df,
  mark_state,
  fontsize,
  mod_levels
)
```

Arguments

`mark_features`	A named list of character vectors, mapping category names to feature IDs, or a plain character vector of feature IDs (all shown in black).
`module_df`	A data frame with columns `Feature` and `Module`.
`mark_state`	A list with element `feat_col`, a named character vector mapping feature IDs to colours.
`fontsize`	Numeric font size for term text.
`mod_levels`	Character vector of module levels to include.

Value

A named list of per-module data.frames (columns `text`, `col`, `fontsize`), or `NULL` if no marked features are found in the module data.

.check_character *Check that x is a character vector*

Description

Check that x is a character vector

Usage

```
.check_character(x, arg = deparse(substitute(x)))
```

Arguments

x	Object to check.
arg	Name of the argument (for error message).

.check_df *Check that x is a data.frame*

Description

Check that x is a data.frame

Usage

```
.check_df(x, arg = "data")
```

Arguments

x	Object to check.
arg	Name of the argument (for error message).

.check_df_feature_property *Check that a data.frame is from summarise_feature_property*

Description

Check that a data.frame is from summarise_feature_property

Usage

```
.check_df_feature_property(x, arg = deparse(substitute(x)))
```

Arguments

x	A data.frame.
arg	Name of the argument (for error message).

.check_df_trendy_summary *Check that a data.frame is from summarise_Trendy*

Description

Check that a data.frame is from summarise_Trendy

Usage

```
.check_df_trendy_summary(x, arg = deparse(substitute(x)))
```

Arguments

x	A data.frame.
arg	Name of the argument (for error message).

.check_limma_input *Warn if assay data looks like raw counts rather than log-transformed values*

Description

limma expects log-transformed input. Raw counts (integers, large max) produce unreliable results. This check warns if the data has characteristics of un-logged counts: all non-negative, >90% of values are integer-like, and max value > 100.

Usage

```
.check_limma_input(mat, assay_name)
```

Arguments

mat	A numeric matrix of expression values.
assay_name	Name of the assay (for the warning message).

.check_list *Check that x is a list*

Description

Check that x is a list

Usage

```
.check_list(x, arg = deparse(substitute(x)), expected_from = NULL)
```

Arguments

x	Object to check.
arg	Name of the argument (for error message).
expected_from	Optional: name of the function whose output is expected, added to the error message.

.check_logical *Check that x is a logical flag*

Description

Check that x is a logical flag

Usage

```
.check_logical(x, arg = deparse(substitute(x)))
```

Arguments

x	Object to check.
arg	Name of the argument (for error message).

.check_nonneg *Check that x is a single non-negative number*

Description

Check that x is a single non-negative number

Usage

```
.check_nonneg(x, arg = deparse(substitute(x)))
```

Arguments

x	Object to check.
arg	Name of the argument (for error message).

.check_numeric *Check that x is numeric*

Description

Check that x is numeric

Usage

```
.check_numeric(x, arg = deparse(substitute(x)))
```

Arguments

x	Object to check.
arg	Name of the argument (for error message).

.check_positive *Check that x is a single positive number*

Description

Check that x is a single positive number

Usage

```
.check_positive(x, arg = deparse(substitute(x)))
```

Arguments

x	Object to check.
arg	Name of the argument (for error message).

.check_positive_int *Check that x is a single positive integer*

Description

Check that x is a single positive integer

Usage

```
.check_positive_int(x, arg = deparse(substitute(x)))
```

Arguments

x	Object to check.
arg	Name of the argument (for error message).

.check_pval *Check that x is a single number between 0 and 1*

Description

Check that x is a single number between 0 and 1

Usage

```
.check_pval(x, arg = deparse(substitute(x)))
```

Arguments

x	Object to check.
arg	Name of the argument (for error message).

.check_se *Check that x is a SummarizedExperiment*

Description

Check that x is a SummarizedExperiment

Usage

```
.check_se(x, arg = "se_obj")
```

Arguments

x	Object to check.
arg	Name of the argument (for error message).

.check_se_has_properties *Check that an SE has calc_feature_property results in rowData*

Description

Check that an SE has calc_feature_property results in rowData

Usage

```
.check_se_has_properties(se_obj, arg = "se_obj")
```

Arguments

se_obj	A SummarizedExperiment object.
arg	Name of the argument (for error message).

.check_se_list *Check that x is a named list of SummarizedExperiment objects*

Description

Check that x is a named list of SummarizedExperiment objects

Usage

```
.check_se_list(x, arg = "se_obj_list")
```

Arguments

x	Object to check.
arg	Name of the argument (for error message).

.check_se_list_has_properties *Check that an SE list has calc_feature_property results*

Description

Check that an SE list has calc_feature_property results

Usage

```
.check_se_list_has_properties(se_obj_list, arg = "se_obj_list")
```

Arguments

se_obj_list	A list of SummarizedExperiment objects.
arg	Name of the argument (for error message).

.check_se_list_merged *Check that an SE list has merged data (one value per Group x Time)*

Description

After `merge_replicates()`, each SE should have exactly one sample per combination of Group and Time.

Usage

```
.check_se_list_merged(se_obj_list, arg = "se_obj_list")
```

Arguments

se_obj_list	A list of SummarizedExperiment objects.
arg	Name of the argument (for error message).

.check_se_list_no_na *Check that an SE list has no missing values (imputed)*

Description

After `impute_groups()`, no assay should contain NA values.

Usage

```
.check_se_list_no_na(se_obj_list, arg = "se_obj_list")
```

Arguments

`se_obj_list`	A list of SummarizedExperiment objects.
`arg`	Name of the argument (for error message).

.check_se_merged *Check that an SE has merged data (one value per Group x Time)*

Description

After `merge_replicates()`, each SE should have exactly one sample per combination of Group and Time.

Usage

```
.check_se_merged(se_obj, arg = "se_obj")
```

Arguments

`se_obj`	A SummarizedExperiment object.
`arg`	Name of the argument (for error message).

.hypergeometric_test *Hypergeometric enrichment test*

Description

One-tailed hypergeometric (Fisher's exact) test of query genes against named gene sets.

Usage

```
.hypergeometric_test(query, gene_sets, universe, min_overlap = 1L)
```

Arguments

query	Character vector of query genes.
gene_sets	Named list of gene vectors (gene set name -> genes, already filtered to universe).
universe	Character vector of all background genes.
min_overlap	Minimum number of overlapping genes to report (default: 1).

Value

A data.frame with columns Term, N, m, k, q, pvalue, or NULL if no gene set passes filters.

.match_assay *Resolve assay argument*

Description

Accepts assay as a name or index, validates it against the object, and returns the resolved value. Open to any assay present in the object.

Usage

```
.match_assay(assay, se_obj)
```

Arguments

assay	A numeric index or character name.
se_obj	A SummarizedExperiment object.

Value

The resolved assay identifier (name if available, otherwise index).

.measure_multicol_widths *Measure column widths for multi-column textbox layout*

Description

Computes the maximum grob width per category in millimetres, used to align columns across modules.

Usage

```
.measure_multicol_widths(text, textbox_args = list())
```

Arguments

text	A per-module list of category term data.frames (output from .reformat_cat_terms).
textbox_args	Additional arguments for .multicol_textbox_col_grob.

Value

A named numeric vector of column widths in mm.

.module_labels *Normalise and order module labels*

Description

Converts numeric or digit-string module labels to a factor with levels in natural numeric order. When `prefix = TRUE` (default), labels are formatted as `"M0"`, `"M1"`, `"M2"`, ... When `prefix = FALSE`, bare digit strings are returned (e.g. `"0"`, `"1"`, `"2"`). WGCNA colour names (e.g. `"turquoise"`, `"blue"`) are unaffected by `prefix` and returned as a factor preserving their original order of first appearance.

Usage

```
.module_labels(x, prefix = TRUE)
```

Arguments

`x`	A vector of module labels (numeric, character digits, or WGCNA colour names).
`prefix`	Logical; if `TRUE` (default), numeric labels are prefixed with `"M"`. Ignored for WGCNA colour names.

Details

Callers that need a different ordering (e.g. by module size) should re-factor the result after calling this function, as `plot_modules_h()` does.

Value

A factor of module labels. Numeric labels are ordered by their numeric value; non-numeric labels preserve their original order of first appearance.

.multicol_textbox_col_grob *Build a single-column textbox grob*

Description

Stacks term text strings vertically using the local `.textbox_grob`. Returns an invisible placeholder grob if `df` has no rows.

Usage

```
.multicol_textbox_col_grob(df, ...)
```

Arguments

`df`	A data.frame with columns `text`, `col`, `fontsize`, and optionally `fontfamily` and `fontface`.
`...`	Passed to `.textbox_grob`.

Value

A textbox grob.

`.multicol_textbox_grob` *Multi-column textbox grob*

Description

Arranges per-category textbox grobs side-by-side in absolute mm units. Returns a `gTree` of class `"multicol_textbox"` with attributes `column_labels`, `column_widths_mm`, `column_gap_mm`, and `padding_mm` for downstream header decoration.

Usage

```
.multicol_textbox_grob(
  x,
  col_gap = unit(2, "mm"),
  background_gp = grid::gpar(fill = "#F7F7F7", col = "#CCCCCC"),
  padding = unit(c(0, 0, 0, 0), "mm"),
  column_widths_mm = NULL,
  ...
)
```

Arguments

`x`	A named list of category term data.frames (one per column).
`col_gap`	Gap between columns as a `grid::unit` (default: 2 mm).
`background_gp`	Background graphical parameters.
`padding`	Padding around content as a `grid::unit` vector (top, left, bottom, right). Default: 0 mm all sides.
`column_widths_mm`	Optional numeric vector of pre-computed column widths in mm. If NULL, auto-computed from grob widths.
`...`	Passed to `.multicol_textbox_col_grob`.

Value

A `gTree` of class `"multicol_textbox"`.

.plot_modules_input *Prepare for plotting WGCNA modules*

Description

Prepare the input data for plotting WGCNA modules

Here, different from running WGCNA, the input data should have the replicates merged, instead of having multiple samples per group, time and feature (gene).

If certain time points are missing in some groups, NA values are added.

Usage

```
.plot_modules_input(module, se_obj_merged, assay, scale)
```

Arguments

module	A data frame with columns "Feature" and "Module"
se_obj_merged	A SummarizedExperiment object, with one value for each feature at each time point in each group (replicates merged). The colData of the object should contain columns "Sample", "Group", and "Time". The object can be produced by split_groups(), merge_replicates() and merge_groups().
assay	The assay to use in the SummarizedExperiment object: a numeric index or character name, e.g. 1 or "orig" for original data, 2 or "norm" for time 0 normalised data. (Required, no default.)
scale	Whether to scale the data (z-score) across samples for each feature

Value

A long-format data frame suitable for ggplot2, with columns "Feature", "Module", "Abundance", "Group", and "Time"

.plot_segments_one_group *Plot segmented regression (one group)*

Description

Plot segmented regression results for specified features

Usage

```
.plot_segments_one_group(se_obj_imp, res, feature, group, nrow = NULL, ...)
```

Arguments

se_obj_imp	A SummarizedExperiment object with no missing values
res	Result of the Trendy analysis, output of run_Trendy()
feature	A character vector of feature names to be plotted
group	A character string specifying the group name (for title purposes)
nrow	Number of rows in the plot layout. If NULL and multiple features are provided, it will be set to half the number of features (rounded up) (default is NULL)
...	Additional arguments to be passed to the Trendy::plotFeature()

Value

A plot of the segmented regression fits for the specified features.

.prepare_gene_list *Prepare input gene list for enrichment functions*

Description

Convert a WGCNA_module() output data.frame to a named gene list, or pass through an existing named list unchanged.

Usage

```
.prepare_gene_list(x)
```

Arguments

x	A data.frame with 'Feature' and 'Module' columns (as returned by WGCNA_module()), or a named list of gene vectors.

Value

A named list of gene vectors, where names correspond to module names.

.reformat_cat_terms *Reformat enrichment category terms into per-module text lists*

Description

Transposes cat_terms from category-first (cat_terms[[cate]][[mod]]) to module-first (text[[mod]][[cate]]) layout expected by .multicol_textbox_grob. Missing categories get empty data.frames.

Usage

```
.reformat_cat_terms(cat_terms)
```

Arguments

cat_terms	A named list of categories, each containing a named list of modules with term annotation data.frames.

Value

A named list of modules, each containing a named list of categories with term data.frames.

.run_Trendy_one_group *Segmented regression analysis (for one group)*

Description

Run segmented regression analysis with Trendy on imputed data for one group of samples in a SummarizedExperiment object. For use in the `run_Trendy()` function.

If less than 2 * minNumInSeg time points are available, Trendy analysis will not be performed and NULL will be returned.

Usage

```
.run_Trendy_one_group(
  se_obj_imp,
  minExp = 0.5,
  feature = NULL,
  maxK = 1,
  meanCut = 0,
  minNumInSeg = 3,
  NCores = 1,
  ...
)
```

Arguments

se_obj_imp	A SummarizedExperiment object with no missing values
minExp	Minimum expression ratio for a feature to be included in the analysis (default is 0.5)
feature	A character vector of feature names to be included in the analysis. If NULL, all features with expression ratio (Exp_ratio) >= minExp will be used, based on results of `calc_feature_property()` before imputation. (default is NULL)
maxK	Parameter of `Trendy::trendy()`, maximum number of breakpoints allowed in the segmented regression model (default is 1)
meanCut	Parameter of `Trendy::trendy()`, minimum mean expression required for a feature to be included in the analysis (default is 0)
minNumInSeg	Parameter of `Trendy::trendy()`, minimum number of samples required in each segment (default is 3)
NCores	Number of cores to use for parallel processing (default is 1)
...	Additional arguments to be passed to the `Trendy::trendy()`

Value

Trendy analysis result, including the fitted model parameters and statistics for each feature.

.summarise_Trendy_one_group *Summarise Trendy results (one group)*

Description

Summarise Trendy results into data frame (one group)

Usage

```
.summarise_Trendy_one_group(res, ...)
```

Arguments

res	Result of the Trendy analysis, output of `run_Trendy()`
...	Additional arguments to be passed to the `Trendy::topTrendy`

Value

A data frame of summary results of Trendy, including breakpoints, segment slopes and p-values for each fitted feature.

.textbox_grob *Local textbox grob*

Description

Stacks text strings vertically with per-string graphical parameters. Adapted from `ComplexHeatmap:::textbox_grob` to avoid dependency on unexported ComplexHeatmap internals.

Usage

```
.textbox_grob(
  text,
  x = unit(0.5, "npc"),
  y = unit(0.5, "npc"),
  just = "centre",
  gp = grid::gpar(),
  background_gp = grid::gpar(col = "transparent", fill = "transparent"),
  round_corners = FALSE,
  r = unit(0.1, "snpc"),
  line_space = unit(4, "pt"),
  text_space = unit(4, "pt"),
  max_width = unit(100, "mm"),
  padding = unit(4, "pt"),
  first_text_from = "top",
  add_new_line = FALSE,
  word_wrap = FALSE
)
```

Arguments

text	Character vector of text strings to display.
x	X position as a grid::unit (default: 0.5 npc).
y	Y position as a grid::unit (default: 0.5 npc).
just	Justification of the viewport (default: "centre").
gp	A grid::gpar object with graphical parameters for text. Elements col, fontsize, fontfamily, and fontface are recycled to match length(text).
background_gp	A grid::gpar object for the background rectangle (default: transparent fill and border).
round_corners	Logical; whether to draw rounded corners on the background rectangle (default: FALSE).
r	Corner radius as a grid::unit when round_corners = TRUE (default: 0.1 snpc).
line_space	Spacing between lines as a grid::unit (default: 4 pt).
text_space	Spacing between inline text segments as a grid::unit (default: 4 pt).
max_width	Maximum width as a grid::unit (default: 100 mm).
padding	Padding around content as a grid::unit vector of length 1, 2, or 4 (default: 4 pt all sides).
first_text_from	"top" (default) or "bottom".
add_new_line	Logical; force each text string to a new line (default: FALSE).
word_wrap	Logical; enable word wrapping (default: FALSE).

Value

A grid::gTree of class "textbox".

.umap_n_neighbors *UMAP neighbors number*

Description

Select UMAP n_neighbors parameter based on the number of samples For sample number > 5, use sample number / 5, with a min of 5 and max of 15. For sample number <= 5, use sample number - 1.

Usage

```
.umap_n_neighbors(sample_n)
```

Arguments

sample_n	Number of samples

Value

UMAP n_neighbors parameter

.WGCNA_input *Prepare WGCNA input data*

Description

Prepare input data for WGCNA, including transposing the data to have samples in rows and features in columns, and removing bad samples and features. Features can be pre-filtered, e.g. by residual variance calculated by `decomp_variance()`, to remove noisy features before preparing the data for WGCNA.

Usage

```
.WGCNA_input(se_obj, assay)
```

Arguments

`se_obj`	A SummarizedExperiment object. Data normalised to time point 0 can be in the second assay slot, created by `normalise_to_start()`.
`assay`	The assay to use in the SummarizedExperiment object: a numeric index or character name, e.g. 1 or "orig" for original data, 2 or "norm" for time 0 normalised data. (Default: 2.)

Value

A data frame with samples in rows and features in columns, filtered to remove bad samples and features, ready for use in `prepare_WGCNA()`.

calc_feature_property *Calculate feature property*

Description

Calculate per-feature temporal properties for each group from the mean of replicates at each time point.

Usage

```
calc_feature_property(se_obj_merged_list, threshold = NULL)
```

Arguments

`se_obj_merged_list`	A list of SummarizedExperiment objects created by `merge_replicates()`, each containing the mean of replicates for each feature at each time point for one group.
`threshold`	A numeric value to determine whether a feature is "expressed" in samples. (default is NULL, meaning any non-NA value is considered expressed).

Value

A list of SummarizedExperiment objects, each corresponding to one group, with updated rowData containing the following columns: Feature, Group, Exp_threshold, T_exp (number of time points with values > Exp_threshold), T_total (total number of time points), P_trend (p-value from Bartels' test for randomness against a trend), Max_FC (maximum fold change across time points), Max_FC_time (difference between the time points at which maximum and minimum expression occur; positive if max occurs after min, negative otherwise), Exp_ratio (T_exp / T_total), Rho_time (Spearman correlation with time; positive = up trend), Peak_ratio (number of local maxima / T_total; 0 = no peaks detected), Log2_CV (log2(1 + CV) of expression across time points), AUC (area under the time-0-normalised curve from assay 2; NA if assay 2 is not available).

Examples

```
data(example_obj)
example_obj <- normalise_to_start(example_obj)
example_obj_list <- split_groups(example_obj)
example_obj_merged_list <- merge_replicates(example_obj_list)

example_obj_merged_list <-
    calc_feature_property(example_obj_merged_list, threshold = 0)
```

---

calc_mean_sd *Calculate mean and SD*

Description

Calculate mean and standard deviation for each group and time point

Usage

```
calc_mean_sd(se_obj)
```

Arguments

se_obj	A SummarizedExperiment object with assays containing expression data. The first assay should be the original data, and the second (if present) should be normalized data.

Value

A list containing two data frames: one for original values and one for normalized values (if available). Each data frame includes columns for Mean, SD, Group, Time, and Feature.

Examples

```
data(example_obj)
table_mean_sd_list <- calc_mean_sd(example_obj)
table_mean_sd <- table_mean_sd_list$norm
```

`create_input` *Create object*

Description

Check format of input data and sample annotation, create a SummarizedExperiment object

Usage

```
create_input(
  data,
  sample_ann,
  subject_col = NULL,
  replicate_col = NULL,
  batch_col = NULL
)
```

Arguments

`data`	A data frame with rows as features (e.g., genes, proteins) and columns as samples. The first column should contain feature identifiers (e.g., gene symbols) and named as 'Feature'. The data should have been log transformed (and normalised if needed). Missing values are allowed in some of the downstream analyses.
`sample_ann`	A data frame containing sample annotations with required columns: 'Sample', 'Group', and 'Time'. 'Replicate' and 'Batch' are optional columns and will be auto-generated if not provided. 'Time', 'Replicate' and 'Batch' should be numeric.
`subject_col`	Optional: name of a column in `sample_ann` identifying biological subjects measured repeatedly across time points (e.g., `"PatientID"`). If provided, the column is renamed to 'Subject' and used for repeated-measures analyses downstream. If NULL (default), all samples are treated as independent, e.g. for cell culture experiments.
`replicate_col`	Optional: name of a column in `sample_ann` identifying replicate IDs. If provided, the column is renamed to 'Replicate'. If NULL (default), the function looks for a column named 'Replicate'; if absent, replicate IDs are auto-generated within each Group and Time (and Subject, if provided).
`batch_col`	Optional: name of a column in `sample_ann` identifying batch information. If provided, the column is renamed to 'Batch'. If NULL (default), the function looks for a column named 'Batch'; if absent, all samples are assigned to batch 1.

Value

A SummarizedExperiment object containing the input data and sample annotations.

Examples

```
data <- data.frame(Feature = c("Gene1", "Gene2"),
                   Sample1 = c(1, 2), Sample2 = c(3, 4))
sample_ann <- data.frame(
```

```
    Sample = c("Sample1", "Sample2"),
    Group = c("A", "A"),
    Time = c(0, 1),
    Rep = c(1, 1),
    BatchID = c(1, 1)
)
se_obj <- create_input(data, sample_ann,
    replicate_col = "Rep", batch_col = "BatchID")
```

`decomp_variance` *Variance decomposition*

Description

Variance decomposition by linear mixed models (LMM), to estimate the contribution of Group and Time to the variance of each feature. Modified from `PALMO::lmeVariance()` function.

Group and Time are always included as random effects. The formula is extended automatically based on colData:

- Subject column present: adds `(1|Subject)` to model between-subject differences
- `interaction = TRUE`: adds `(1|Group:Time)` (or `(1|Subject:Time)` when Subject present) to capture interaction variance. Requires >= 2 replicates per combination. Default: FALSE.

Output always includes: Group, Time, Residual. Subject is included when present. All values are percentages of total variance.

Allow using original data or data normalised to time point 0.

Usage

```
decomp_variance(
  se_obj,
  features = NULL,
  fixed_effect_var = NULL,
  interaction = FALSE,
  assay,
  core = 1
)
```

Arguments

`se_obj`	A SummarizedExperiment object created by `create_input()`
`features`	A vector of features to include. If NULL, all features.
`fixed_effect_var`	Column names to include as fixed effects (regressed out, not appearing as variance components). Default is `NULL` (no fixed effects). Set to `"Batch"` to regress out batch effects, or provide other columns (e.g., `c("Sex", "Age")`).
`interaction`	Logical. If TRUE, adds `(1|Group:Time)` (or `(1|Subject:Time)` when Subject present) to the model to capture interaction variance. Default: FALSE.
`assay`	The assay to use in the SummarizedExperiment object: a numeric index or character name, e.g. 1 or "orig" for original data, 2 or "norm" for time 0 normalised data. (Required, no default.)
`core`	Number of cores for parallel processing (default: 1)

Value

A data frame with variance decomposition results (percentages).

References

https://github.com/aifimmunology/PALMO/blob/main/R/lmeVariance.R

Examples

```
data(example_obj)
example_obj <- normalise_to_start(example_obj)

var_decomp <- decomp_variance(example_obj, assay = 1)
```

DE_between_group *DE between groups*

Description

Differential expression analysis between groups at each time point by limma. The function compares each pair of groups at each time point, and returns a nested list of DE analysis results for each pair of groups and each time point including all features, as well as a list of filtered DE results for each pair of groups based on the specified thresholds including only significant features. Use `plot_DE_between_group()` to visualise the number of DE features between groups over time.

Usage

```
DE_between_group(
  se_obj,
  group = NULL,
  filter = NULL,
  assay,
  adjP_thres = 0.05,
  logFC_thres = 1,
  trend = FALSE
)
```

Arguments

`se_obj`	A SummarizedExperiment object
`group`	A character vector specifying which group to be compared to. If NULL, all groups in the 'Group' column will be compared to. (default is NULL)
`filter`	Minimum number of replicates required in both conditions for a feature to be tested. If NULL, the minimum number of replicates across all groups and time points will be used. (default is NULL)
`assay`	The assay to use in the SummarizedExperiment object: a numeric index or character name, e.g. 1 or "orig" for original data, 2 or "norm" for time 0 normalised data. (Required, no default.) The selected assay should contain log-transformed, normalised values (e.g. log2-CPM for RNA-seq, log2-intensity for proteomics). A warning is issued if the data appears to be un-logged raw counts.

adjP_thres	Threshold for adjusted p-value to consider a feature as differentially expressed (default is 0.05)
logFC_thres	Threshold for log2 fold change to consider a feature as differentially expressed (default is 1)
trend	Logical, passed to limma::eBayes(). Set to TRUE for RNA-seq count-derived data to model the mean-variance trend. Leave as FALSE (default) for microarray, proteomics, metabolomics, or other log-intensity data where the mean-variance relationship is typically flat.

Details

Only time points present in both groups are compared. No multiple-testing correction is applied across the pairwise group-by-time comparisons; each comparison is independent. Apply your own correction (e.g., stats::p.adjust()) across combined results if needed.

Value

A list with: all_list (nested list of DE results per group pair and time point, all features); de_list (significant features only); fit_list (nested list of limma MArrayLM fit objects, for use with limma::plotSA()); ref_groups (group to be compared to); all_groups (all groups). The output can be passed to plot_DE_between_group() for visualisation.

Examples

```
data(example_obj)
example_obj <- normalise_to_start(example_obj)

DE_between_group_out <- DE_between_group(example_obj, assay = 2)
```

DE_between_time *DE between time points*

Description

Differential expression analysis between time points within each group by limma

Usage

```
DE_between_time(
  se_obj,
  group = NULL,
  filter = NULL,
  assay,
  adjP_thres = 0.05,
  logFC_thres = 1,
  trend = FALSE
)
```

Arguments

se_obj	A SummarizedExperiment object created by create_input
group	A character vector specifying which groups to analyse. If NULL, all groups in the 'Group' column will be used. (default is NULL)
filter	Minimum number of replicates required in both conditions for a feature to be tested. If NULL, the minimum number of replicates across all groups and time points will be used. (default is NULL)
assay	The assay to use in the SummarizedExperiment object: a numeric index or character name, e.g. 1 or "orig" for original data, 2 or "norm" for time 0 normalised data. (Required, no default.) The selected assay should contain log-transformed, normalised values (e.g. log2-CPM for RNA-seq, log2-intensity for proteomics). A warning is issued if the data appears to be un-logged raw counts.
adjP_thres	Threshold for adjusted p-value to consider a feature as differentially expressed (default is 0.05)
logFC_thres	Threshold for log2 fold change to consider a feature as differentially expressed (default is 1)
trend	Logical, passed to limma::eBayes(). Set to TRUE for RNA-seq count-derived data to model the mean-variance trend. Leave as FALSE (default) for microarray, proteomics, metabolomics, or other log-intensity data where the mean-variance relationship is typically flat.

Value

A list with: all_list (nested list of DE results per group and time comparison, all features); de_list (nested list of significant features only); fit_list (nested list of limma MArrayLM fit objects, for use with limma::plotSA()); time_series (vector of all available time points). The output can be passed to plot_DE_between_time() for visualisation.

Examples

```
data(example_obj)
example_obj <- normalise_to_start(example_obj)

DE_between_time_out <- DE_between_time(example_obj, assay = 1)
```

---

enrichGO_list *GO enrichment with gene sets*

Description

Gene ontology enrichment analysis of multiple gene sets with clusterProfiler, allowing for using different or same background genes for each gene set

Usage

```
enrichGO_list(
  gene_list,
  keyType = "SYMBOL",
  OrgDb,
```

```
  universe = NULL,
  universe_list = NULL,
  pAdjustMethod = "BH",
  pvalueCutoff = 0.05,
  qvalueCutoff = 0.05,
  category = NULL,
  simplify = FALSE,
  simplify_cutoff = 0.7,
  simplify_by = "p.adjust",
  simplify_select_fun = min,
  simplify_measure = "Wang",
  ...
)
```

Arguments

gene_list	A named list of gene vectors, or a data.frame with Feature and Module columns from WGCNA_module().
keyType	Available options are AnnotationDbi::keytypes(OrgDb) (default is "SYMBOL")
OrgDb	Organism database, e.g. org.Hs.eg.db, org.Mm.eg.db
universe	Background genes for all input gene sets, used if universe_list is not provided
universe_list	Background genes for each input gene set, a list of gene vectors with the same names as gene_list
pAdjustMethod	Parameter of clusterProfiler::enrichGO() (default is "BH")
pvalueCutoff	Parameter of clusterProfiler::enrichGO() (default is 0.05)
qvalueCutoff	Parameter of clusterProfiler::enrichGO() (default is 0.05)
category	GO category to analyze (default is all three of BP, MF, CC)
simplify	Whether to simplify the GO terms by removing redundant terms with clusterProfiler::simplify(function. (default is FALSE)
simplify_cutoff	Parameter of clusterProfiler::simplify(), cutoff for similarity when simplifying GO terms (default is 0.7)
simplify_by	Parameter of clusterProfiler::simplify(), method to choose representative term when simplifying GO terms (default is "p.adjust")
simplify_select_fun	Parameter of clusterProfiler::simplify(), function to select representative term when simplifying GO terms (default is min)
simplify_measure	Parameter of clusterProfiler::simplify(), method to calculate similarity when simplifying GO terms (default is "Wang")
...	additional arguments passed to clusterProfiler::enrichGO()

Value

A nested list of GO enrichment results with sublists: 'all' and 'simplified' (if simplify = TRUE) containing merged results of all or simplified terms across gene sets for each GO category; 'unmerged_all' and 'unmerged_simplified' (if simplify = TRUE) including all or simplified terms for each gene set and GO category.

Examples

```
if (requireNamespace("org.Mm.eg.db", quietly = TRUE)) {
    library(org.Mm.eg.db)
    library(clusterProfiler)
    data(example_net)
    # select two modules for demonstration
    example_module <- WGCNA_module(example_net) |>
        dplyr::filter(Module %in% c("1", "2"))
    # set cutoff to 1 to show all results for demonstration
    example_go_list = enrichGO_list(example_module, OrgDb = org.Mm.eg.db,
        universe = WGCNA_module(example_net, exclude_grey = FALSE)$Feature,
        pvalueCutoff = 1, qvalueCutoff = 1,
        category = "BP", simplify = FALSE)
}
```

---

enrichGO_rank *GO enrichment with ranked gene list*

Description

Gene ontology enrichment analysis of a ranked gene list with clusterProfiler. Genes can be ranked by variance decomposition results.

Usage

```
enrichGO_rank(
  rank_table,
  gene_rank_by,
  OrgDb,
  keyType = "SYMBOL",
  go_rank_by = "p.adjust",
  category = NULL,
  pvalueCutoff = 0.05,
  pAdjustMethod = "BH",
  ...
)
```

Arguments

rank_table	A data frame with at least two columns: 'Feature' for gene names and one or more columns of variables for ranking the genes, e.g. output of `decomp_variance()` containing variance decomposition results
gene_rank_by	Variable in `rank_table` to rank the genes by, e.g. "Time", "Group" in the output of `decomp_variance()`
OrgDb	Organism database, e.g. org.Hs.eg.db, org.Mm.eg.db
keyType	Available options are `AnnotationDbi::keytypes(OrgDb)` (default is "SYMBOL")
go_rank_by	Variable in the GO enrichment result to rank the GO terms by (default is "p.adjust", other options include "pvalue", "qvalue", "NES", "setSize", "enrichmentScore", etc.)

category	GO category to analyze (default is all three of BP, MF, CC)
pvalueCutoff	Parameter of clusterProfiler::gseGO() (default is 0.05)
pAdjustMethod	Parameter of clusterProfiler::gseGO() (default is "BH")
...	additional arguments passed to clusterProfiler::gseGO()

Value

A list of gseaResult objects containing the GSEA results

Examples

```
if (requireNamespace("org.Mm.eg.db", quietly = TRUE)) {
    library(org.Mm.eg.db)
    data(example_obj)
    example_obj <- normalise_to_start(example_obj)

    var_decomp <- decomp_variance(example_obj,
        features = rownames(example_obj)[1:100], assay = 1)
    example_go_rank <- enrichGO_rank(var_decomp, gene_rank_by = "Time",
        OrgDb = org.Mm.eg.db, keyType = "SYMBOL",
        category = "BP")
}
```

---

enrichR_list *Gene-set enrichment via enrichR*

Description

Enrichment analysis of multiple gene sets against databases from enrichR (e.g. DSigDB for drug signatures, ChEA for TF targets, KEGG for pathways, DrugBank for drug targets).

Usage

```
enrichR_list(
  gene_list,
  databases = c("DSigDB", "DrugMatrix"),
  site = "Enrichr",
  universe = NULL,
  universe_list = NULL,
  pvalueCutoff = 0.05,
  include_overlap = FALSE
)
```

Arguments

gene_list	A named list of gene vectors, or a data.frame with Feature and Module columns from WGCNA_module(). Gene identifiers must match the convention of the chosen site (e.g. human gene symbols for Enrichr, fly symbols for FlyEnrichr).
databases	Character vector of enrichR database names to query. Available databases can be checked with enrichR::listEnrichrDbs(). (default: c("DSigDB", "DrugMatrix"))
site	enrichR site to query. See enrichR::listEnrichrSites(). (default: "Enrichr")

universe	Background genes for all input gene sets, used if universe_list is not provided.
universe_list	Background genes for each input gene set, a list of gene vectors with the same names as gene_list.
pvalueCutoff	Adjusted p-value cutoff for filtering enriched terms (default: 0.05)
include_overlap	Parameter passed to enrichR::enrichr(). If TRUE, databases are downloaded during each query to output 'Overlap' when analysing with a background. (default: FALSE)

Details

Multiple testing: P-values are adjusted (Benjamini–Hochberg) independently for each module within each database. No correction is applied across databases or across modules.

Requires an internet connection.

Value

A named list of data.frames, one per database. Each data.frame has columns Cluster, Description, p.adjust (Adjusted.P.value from enrichR output), Combined.Score, Genes, and any additional columns returned by the enrichR API. Compatible with plot_modules_h(enrich_list = result, enrich_category = "DSigDB").

Examples

```
data(example_net)
library(dplyr)
example_module <- WGCNA_module(example_net) |>
    dplyr::filter(Module %in% c("1", "2"))
# Use high pvalueCutoff for demonstration
# enrichr_out <- enrichR_list(example_module,
#     databases = c("KEGG_2019_Mouse"),
#     universe = WGCNA_module(example_net, exclude_grey = FALSE)$Feature,
#     pvalueCutoff = 0.5)
```

---

enrich_msigdb *Gene set enrichment via MSigDB*

Description

Enrichment analysis against MSigDB gene sets via hypergeometric test. Supports MSigDB categories, subcategories, and specified gene sets. Requires the msigdbr package.

Use cases:

- Drug perturbations: category = "C2", subcategory = "CGP" (chemical and genetic perturbations)
- TF targets: category = "C3", subcategory = "TFT:GTRD"
- Hallmarks: category = "H", subcategory = NULL (no subcategory)
- GO: category = "C5", subcategory = "BP" (GO biological process)

Multiple testing: P-values are adjusted (Benjamini-Hochberg) per module across the gene-set tests within that module.

Usage

```
enrich_msigdb(
  gene_list,
  universe,
  category = NULL,
  subcategory = NULL,
  gene_sets = NULL,
  species = "Homo sapiens",
  db_species = "HS",
  name = NULL,
  pvalueCutoff = 0.05,
  pAdjustMethod = "BH",
  minGSSize = 10,
  maxGSSize = 500
)
```

Arguments

gene_list	A named list of gene vectors, or a data.frame with Feature and Module columns from WGCNA_module().
universe	Background genes (required).
category	MSigDB category (default: NULL). See msigdbr::msigdbr_collections(db_species = "HS"/"MM") for available options. Not required when gene_sets is provided; in that case all collections are searched to locate the named gene sets.
subcategory	MSigDB subcategory (default: NULL). Set to NULL to include all subcategories of category.
gene_sets	Optional character vector of specific MSigDB gene set names to include (e.g., c("HALLMARK_APOPTOSIS", "HALLMARK_P53_PATHWAY")). When provided, msigdbr is queried across all collections to find these gene sets, so category and subcategory are not required. If NULL (default), all gene sets in the category/subcategory are used.
species	Species of input genes (default: "Homo sapiens"). See msigdbr::msigdbr_species() for available options.
db_species	Species of MSigDB genesets (default: "HS"). Use "MM" for mouse-native gene sets. If species and db_species mismatch, db_species MSigDB will be used with ortholog mapping to species.
name	Name for the output list element. If NULL, auto-derived from category (e.g., "C2" names the output element "C2"). If category is also NULL, it uses "msigdb".
pvalueCutoff	Adjusted p-value cutoff (default: 0.05).
pAdjustMethod	P-value adjustment method (default: "BH").
minGSSize	Minimum gene set size (default: 10).
maxGSSize	Maximum gene set size (default: 500).

Value

A named list with one element: a data.frame with columns Cluster, ID, Description, GeneRatio, BgRatio, pvalue, p.adjust, Count, geneID.

Examples

```
if (requireNamespace("msigdbr", quietly = TRUE)) {
    data(example_net)
    example_module <- WGCNA_module(example_net) |>
        dplyr::filter(Module %in% c("1", "2"))
    hallmark_msigdb <- enrich_msigdb(example_module, category = "MH",
        species = "Mus musculus", db_species = "MM",
        minGSSize = 1,
        pvalueCutoff = 0.9,
        universe = WGCNA_module(example_net, exclude_grey = FALSE)$Feature)
}
```

---

example_go enrichGO_list() *output object for runnable examples*

Description

Code for producing the data is in `inst/script/generate_example_go.R`

Usage

```
data(example_go)
```

Format

A nested list with GO enrichment results for modules "1" and "2" from 'example_net', containing categories "BP" and "CC", with simplify = TRUE applied:

all Merged results across gene sets for each GO category

simplified Simplified merged results

unmerged_all Per-gene-set results for each category

unmerged_simplified Simplified per-gene-set results

Source

GSE263759

---

example_net run_WGCNA() *output object for runnable examples*

Description

Code for producing the data is in `inst/script/generate_example_net.R`

Usage

```
data(example_net)
```

Format

A list including WGCNA module assignments, module eigengenes, dendrogram, input data, sample information, and parameters used

Source

GSE263759

---

example_obj *SummarizedExperiment object for runnable examples*

Description

A subset of `data_obj <- create_input(data = tutorial_data, sample_ann = tutorial_sample_info)` for use in runnable examples in function documentation.

Usage

```
data(example_obj)
```

Format

A SummarizedExperiment object with assays of 100 rows and 40 columns, colData of 40 rows and 5 columns:

colData tutorial_sample_info

assays tutorial_data first 100 rows, "Feature" column as rownames

Details

Code for preparing the data is available in `inst/script/tutorial_input.R`

Source

GSE263759

---

example_res_list `run_Trendy` *output object for runnable examples*

Description

Code for preparing the data is in `inst/script/generate_example_res_list.R`

Usage

```
data(example_res_list)
```

Format

A nested list, each element is a list with Trendy results for one group

Source

GSE263759

`extract_hubs` *Extract hub features from WGCNA modules*

Description

Identifies hub features within each module using module membership (kME). For unsigned networks, absolute kME values are used to reproduce the behaviour of `WGCNA::blockwiseModules()`. For signed and signed hybrid networks, signed kME values are retained.

Usage

```
extract_hubs(net, top_n = 3, exclude_grey = TRUE)
```

Arguments

`net`	An output object from `run_WGCNA()`
`top_n`	Number of hub features to return per module. Default: 3.
`exclude_grey`	Logical; if `TRUE`, exclude the grey/unassigned module. Default: `TRUE`.

Value

A character vector containing the top `top_n` features from each module concatenated together.

Examples

```
data(example_net)
hubs <- extract_hubs(example_net, top_n = 3)
```

`extract_segment_trends`
Extract feature trends

Description

Extract features with different segment trends from trendy segmented regression results summary

Usage

```
extract_segment_trends(trendy.summary)
```

Arguments

`trendy.summary`	A data frame, output of `summarise_Trendy()`, containing the segmented regression results for all features in all groups.

Value

A nested list of features grouped by their segment trend patterns within each group.

Examples

```
data(example_obj)
example_obj <- normalise_to_start(example_obj)
example_obj_list <- split_groups(example_obj)
example_obj_merged_list <- merge_replicates(example_obj_list)
example_obj_merged_list <- calc_feature_property(example_obj_merged_list,
    threshold = 0)
# no missing value in the example dataset, so imputation is not necessary
example_obj_merged_imp_list <- impute_groups(example_obj_merged_list)

data("example_res_list")
trendy_summary <- summarise_Trendy(example_res_list)
trendy_list <- extract_segment_trends(trendy_summary)
```

---

flatten_DE *Flatten nested differential expression results*

Description

Flattens the nested DE result structure returned by DE_between_group() or DE_between_time() into a named list of data frames, one entry per contrast-time combination. This is useful for exporting results to other tools such as DeeDeeExperiment.

Usage

```
flatten_DE(de_list)
```

Arguments

de_list	A list of DE results. Accepts individual output of DE_between_group() or DE_between_time() with an all_list element; or a named list (e.g. list(between_group = ..., between_time = ...)), each containing an all_list element.

Value

A named list of data frames with columns renamed for DeeDeeExperiment compatibility (log2FoldChange, pvalue, padj). Returns an empty list if de_list is empty.

Examples

```
data(tutorial_data)
data(tutorial_sample_info)
tide <- prepare_tide(tutorial_data, tutorial_sample_info,
    keep = "threshold", residual_threshold = 100)
tide$DE <- DE_between_group(tide$se, assay = "norm",
    filter = 1, trend = TRUE)
de_flat <- flatten_DE(tide$DE)
str(de_flat, max.level = 1)
```

flatten_enrich *Flatten nested enrichment results*

Description

Flattens the nested enrichment result structure returned by `enrichGO_list()`, `enrichR_list()`, or `enrich_msigdb()` into a flat named list of data frames or `enrichResult` objects. This is useful for exporting results to other tools such as `DeeDeeExperiment`.

Usage

```
flatten_enrich(enrich_list)
```

Arguments

`enrich_list`	Output of `enrichGO_list()`, `enrichR_list()`, or `enrich_msigdb()`; or a named list combining several such outputs (e.g. `list(GO = go_res, MSigDB = msigdb_res)`).

Value

A flat named list of data frames or `enrichResult` objects with `DeeDeeExperiment`-compatible names and columns (e.g. "clusterProfiler_BP_1", "enrichr_DSigDB"). Returns an empty list if `enrich_list` is empty.

Examples

```
data(example_go)
enrich_flat <- flatten_enrich(example_go)
str(enrich_flat, max.level = 1)
```

get_custom_palette *Get custom color palette*

Description

Get the custom color palette set by `set_custom_palette`. If no custom palette has been set, the function will return a default color palette based on the number of groups specified in the `groups` argument.

Usage

```
get_custom_palette(groups)
```

Arguments

`groups`	A character vector of group names for which to retrieve the colors from the custom palette.

Value

A named vector of colors corresponding to the specified group names.

Examples

```
get_custom_palette(c("untreated", "IFNbeta"))
```

`group_specific_features` *Group specific features*

Description

Identify features that are unique to selected groups (present in those groups but not in any others) and annotate them with gene names using the `clusterProfiler::bitr()` function. The function also provides an option to perform Gene Ontology (GO) enrichment analysis on the identified unique features using the `enrichGO_list()` function and visualize the results with a dot plot.

Usage

```
group_specific_features(
  property_random_fc,
  groups = NULL,
  filter_ratio = 0.5,
  group_pct = 1,
  genename = TRUE,
  GO = TRUE,
  OrgDb = NULL,
  keytype = NULL,
  ...
)
```

Arguments

`property_random_fc`	A data frame containing the results of the `calc_feature_property()` function, which includes the feature names, group names, and the proportion of expressed values for each feature in each group. The data frame should have at least the following columns: "Feature", "Group", "Exp_ratio" and "Exp_threshold".
`groups`	A character vector of group names to be compared. If NULL, all groups in the input data will be used (default is NULL).
`filter_ratio`	A numeric value between 0 and 1 specifying the minimum proportion of expressed time points (minimum Exp_ratio) for a feature to be considered present (default is 0.5, meaning that a feature must have >=50% values > threshold in a group (>= 0.5 Exp_ratio) to be considered present in that group).
`group_pct`	A numeric value between 0 and 1 specifying the percentage of groups in which a feature must be present. (default is 1, meaning that a feature must be present in all specified groups).
`genename`	A logical value indicating whether to output a table of gene names. If TRUE, the function will use the `clusterProfiler::bitr()` function to annotate the features with gene names based on the specified `OrgDb` and `keytype`. (default is TRUE).

GO	A logical value indicating whether to perform Gene Ontology (GO) enrichment analysis on the identified unique features. If TRUE, the function will use the enrichGO_list() function to perform GO enrichment analysis and visualize the results with a dot plot. (default is TRUE).
OrgDb	An OrgDb object from the AnnotationDbi package corresponding to the organism of interest (e.g., org.Hs.eg.db for human, org.Mm.eg.db for mouse). This will be used for gene annotation with the clusterProfiler::bitr() function.
keytype	A character string specifying the type of gene identifiers used in the row names of the assay data (e.g., "SYMBOL", "ENTREZID", "ENSEMBL"). This will be used for gene annotation with the clusterProfiler::bitr() function. Available key types depend on the OrgDb database and can be checked with the AnnotationDbi::keytypes function.
...	Additional arguments to be passed to the enrichGO_list() function for GO enrichment analysis (e.g., pvalueCutoff, qvalueCutoff, etc.).

Value

A named list with element features (character vector of features identified as unique to the specified groups based on the filtering criteria). If genename is TRUE, element genename contains a data.frame of gene annotations. If GO is TRUE and enrichment succeeds, element GO contains a dot plot of GO enrichment results. Returns NULL if no unique features pass the filter.

Examples

```
data(example_obj)
example_obj <- normalise_to_start(example_obj)
example_obj_list <- split_groups(example_obj)
example_obj_merged_list <- merge_replicates(example_obj_list)

example_obj_merged_list <-
    calc_feature_property(example_obj_merged_list, threshold = 0)
property_random_fc <- summarise_feature_property(example_obj_merged_list)

group_specific_features(property_random_fc, groups = c("untreated"),
    genename = FALSE, GO = FALSE)
```

---

impute_groups *Impute missing values*

Description

Impute missing values for each group of samples in a list of SummarizedExperiment objects. By default, replaces NA with the minimum non-NA value per group / subject. A custom function can be supplied for other strategies.

The input samples can contain replicates, or merged replicates (mean of replicates).

Usage

```
impute_groups(se_obj_list, fun = min, impute_by = c("group", "subject"))
```

Arguments

se_obj_list	A list of SummarizedExperiment objects, created by split_groups() or merge_replicates(), each corresponds to one group of samples.
fun	Function applied to the non-NA values to generate the replacement value. Default: min. Use function(x) min(x) / 2 for half-minimum, median for median imputation, etc.
impute_by	"group" (default): compute replacement value from all non-NA values in the group. "subject": compute per-subject replacement value (requires Subject column). If a subject is all NA, fall back to group-level replacement.

Value

A list of SummarizedExperiment objects with missing values imputed. Each object corresponds to one group of samples.

Examples

```
data(example_obj)
example_obj <- normalise_to_start(example_obj)
example_obj_list <- split_groups(example_obj)
example_obj_merged_list <- merge_replicates(example_obj_list)
example_obj_merged_imp_list <- impute_groups(example_obj_merged_list)
```

---

merge_groups *Merge groups into one object*

Description

Merge SummarizedExperiment object of different groups into one

Usage

```
merge_groups(se_obj_list)
```

Arguments

se_obj_list	A list of SummarizedExperiment objects, such as output of split_groups() and merge_replicates(). Names of the list components are the group names.

Value

A SummarizedExperiment object containing all samples from the input list. The 'Group' column in the colData will indicate the group of each sample. New sample names will be prefixed with the group name.

Examples

```
data(example_obj)
example_obj <- normalise_to_start(example_obj)
example_obj_list <- split_groups(example_obj)
example_obj_merged_list <- merge_replicates(example_obj_list)
example_obj_merged <- merge_groups(example_obj_merged_list)
```

merge_replicates *Merge replicates*

Description

Calculate mean of replicates for each feature at each time point for each group.

When a Subject column is present, merging is done in two stages:

1. Average replicates within each Group-Subject-Time combination.
2. Average across subjects within each Group-Time combination. This gives equal weight to each subject regardless of replicate count.

Without a Subject column, all samples at the same Group-Time are averaged together in a single step.

Usage

```
merge_replicates(se_obj_list)
```

Arguments

se_obj_list	A list of SummarizedExperiment objects created by `split_groups()`, each corresponds to one group of samples.

Value

A list of SummarizedExperiment objects containing the mean of replicates for each feature at each time point for each group. Each object in the list corresponds to one group of samples.

Examples

```
data(example_obj)
example_obj <- normalise_to_start(example_obj)
example_obj_list <- split_groups(example_obj)
example_obj_merged_list <- merge_replicates(example_obj_list)
```

normalise_to_start *Normalise to starting time point*

Description

Normalise to starting time point, to make the mean of starting time point samples 0.

Two modes are available:

- `by_subject = FALSE` (default): computes the group-level mean at the first non-NA time point per feature and subtracts it. All samples within a group share the same baseline.
- `by_subject = TRUE`: computes each subject's value at the first non-NA time point per feature and subtracts that subject-specific baseline. Removes between-subject baseline differences. Use when each subject has their own initial condition. Requires a Subject column in colData.

Usage

```
normalise_to_start(se_obj, by_subject = FALSE)
```

Arguments

se_obj	A SummarizedExperiment object created by create_input()
by_subject	If FALSE (default), use group-level baseline. If TRUE, use subject-level baseline (requires Subject column in colData).

Value

A SummarizedExperiment object with normalised data in the second assay slot.

Examples

```
data(example_obj)
example_obj <- normalise_to_start(example_obj)
```

---

plot_breakpoints *Plot breakpoint distribution*

Description

Plot breakpoint distribution among time points for each group

Usage

```
plot_breakpoints(res_list, group = NULL, fontsize = 8, ...)
```

Arguments

res_list	A list of the Trendy analysis results, output of run_Trendy()
group	A character vector of group names to be plotted. If NULL, all groups in the input list will be used (default is NULL)
fontsize	Font size for the plot (default is 8)
...	Additional arguments to be passed to Trendy::topTrendy()

Value

A plot showing the distribution of breakpoints over time for each specified group

Examples

```
data("example_res_list")
plot_breakpoints(example_res_list)
```

plot_cor_matrix *Plot correlation matrix*

Description

Plot correlation matrix between samples as a heatmap

Usage

```
plot_cor_matrix(
  se_obj,
  assay = 1,
  use = "pairwise.complete.obs",
  method = c("spearman", "pearson", "kendall"),
  label_group = TRUE,
  label_time = TRUE,
  label_rep = TRUE,
  label_batch = TRUE,
  show_rownames = FALSE,
  show_colnames = FALSE,
  fontsize = 8,
  cellwidth = 1,
  cellheight = 1,
  title = NULL,
  ...
)
```

Arguments

se_obj	A SummarizedExperiment object created by `create_input()`
assay	The assay to use in the SummarizedExperiment object: a numeric index or character name, e.g. 1 or "orig" for original data, 2 or "norm" for time 0 normalised data. (Default: 1.)
use	Parameter of `stats::cor()` (default is "pairwise.complete.obs")
method	Parameter of `stats::cor()` (default is "spearman")
label_group	Whether to label Group (default is TRUE)
label_time	Whether to label Time (default is TRUE)
label_rep	Whether to label Replicate (default is TRUE)
label_batch	Whether to label Batch (default is TRUE)
show_rownames	Whether to show row names in the heatmap (default is FALSE)
show_colnames	Whether to show column names in the heatmap (default is FALSE)
fontsize	Font size for the plot (default is 8)
cellwidth	Cell width for the heatmap (default is 1)
cellheight	Cell height for the heatmap (default is 1)
title	Title of the heatmap. If NULL, auto-generated as "Sample correlation ()" with the method capitalised (e.g. "Sample correlation (Spearman)") (default: NULL)
...	Additional arguments to be passed to `ComplexHeatmap::Heatmap()`

Value

A heatmap showing the correlation between samples.

Examples

```
data(example_obj)
plot_cor_matrix(example_obj)
```

`plot_cv` *Plot coefficient of variation (CV)*

Description

Plot the distribution of coefficient of variation (CV) for each feature with replicates. The CV is calculated as the standard deviation divided by the mean of the abundance values.

Note: CV is only meaningful for positive-valued data.

Usage

```
plot_cv(se_obj, fontsize = 8)
```

Arguments

`se_obj`	A SummarizedExperiment object, produced by `create_input()` function, containing the abundance data and associated sample information.
`fontsize`	Font size for the plot (default is 8) (default is 8).

Value

A plot showing the distribution of coefficient of variation (CV) for each feature, grouped by Time and Group. If there are no replicates, a message will be printed indicating that CV cannot be calculated.

Examples

```
data(example_obj)
plot_cv(example_obj)
```

plot_DE_between_group *DE number between groups*

Description

Plot the number of differentially expressed features between groups over time, using the output of DE_between_group().

DE features were pre-filtered with DE_between_group(), but by specifying adjP_thres and logFC_thres, users can re-filter the DE features for plotting.

Usage

```
plot_DE_between_group(
  DE_between_group_out,
  group = NULL,
  fontsize = 8,
  adjP_thres = NULL,
  logFC_thres = NULL
)
```

Arguments

DE_between_group_out	Output of DE_between_group(), a list with all_list, de_list, ref_groups, all_groups elements.
group	A character vector specifying which groups to use as the reference (Cond1) for plotting. If NULL, all groups in the output will be plotted. (default is NULL)
fontsize	Font size for the plot (default is 8)
adjP_thres	Threshold for adjusted p-value to consider a feature as differentially expressed, for re-filtering the DE features. (default is NULL, no re-filtering)
logFC_thres	Threshold for log2 fold change to consider a feature as differentially expressed, for re-filtering the DE features. (default is NULL, no re-filtering)

Value

A series of line plots showing the number of DE features over time for each group comparison.

Examples

```
data(example_obj)
example_obj <- normalise_to_start(example_obj)

DE_between_group_out <- DE_between_group(example_obj, assay = 2)
plot_DE_between_group(DE_between_group_out, fontsize = 8)
```

plot_DE_between_time *DE number between time points*

Description

Plot the number of differentially expressed features between time points within each group with heatmaps

DE features were pre-filtered with `DE_between_time()`, but by specifying `adjP_thres` and `logFC_thres`, users can re-filter the DE features for plotting.

Usage

```
plot_DE_between_time(
  DE_between_time_out,
  value = TRUE,
  fontsize = 8,
  nrow = 1,
  heatmap_width = 4,
  heatmap_unit = "cm",
  adjP_thres = NULL,
  logFC_thres = NULL
)
```

Arguments

`DE_between_time_out`	Output of `DE_between_time()`, a list with `all_list`, `de_list`, `time_series` elements.
`value`	Whether to display the actual number of DE features in each heatmap cell (default is TRUE)
`fontsize`	Font size for the plot (default is 8)
`nrow`	Number of rows for arranging the heatmaps (default is 1)
`heatmap_width`	Width of each heatmap (default is 4)
`heatmap_unit`	Unit for the heatmap width (default is "cm")
`adjP_thres`	Threshold for adjusted p-value to consider a feature as differentially expressed, for re-filtering the DE features. (default is NULL, no re-filtering)
`logFC_thres`	Threshold for log2 fold change to consider a feature as differentially expressed, for re-filtering the DE features. (default is NULL, no re-filtering)

Value

One heatmap per group showing the number of DE features between time points for each group, with the same scale across groups for easy comparison. The heatmap cells are optionally annotated with the number of DE features.

Examples

```
data(example_obj)
example_obj <- normalise_to_start(example_obj)

DE_between_time_out <- DE_between_time(example_obj, assay = 1)
plot_DE_between_time(DE_between_time_out,
    fontsize = 8, value = TRUE, nrow = 1, heatmap_width = 3)
```

`plot_distribution` *Abundance distribution plot*

Description

This function generates density plots to visualise the distribution of abundance values across samples. The user can choose to facet the plot by "Group", "Time", or "Sample" to compare distributions across different conditions or samples. If no faceting variable is specified, a single density plot will be generated for all samples combined.

Usage

```
plot_distribution(se_obj, facet_by = NULL, fontsize = 8)
```

Arguments

`se_obj`	A SummarizedExperiment object, produced by `create_input()` function, containing the abundance data and associated sample information.
`facet_by`	A character string specifying the variable to facet the plot by. It can be one of "Group", "Time", or "Sample". If NULL, no faceting will be applied and a single density plot will be generated for all samples (default is NULL).
`fontsize`	Font size for the plot (default is 8) (default is 8).

Value

A plot showing the density distribution of abundance values, optionally faceted by the specified variable.

Examples

```
data(example_obj)
plot_distribution(example_obj)
plot_distribution(example_obj, facet_by = "Group")
```

plot_GO *Plot GO enrichment*

Description

Visualization of GO enrichment analysis with dotplot, cnetplot, or emapplot in clusterProfiler

Usage

```
plot_GO(
  go_list,
  plot_dotplot = TRUE,
  plot_cnetplot = FALSE,
  plot_emapplot = FALSE,
  showCategory_dotplot = 5,
  showCategory_cnetplot = 5,
  showCategory_emapplot = 5,
  fontsize = 8,
  label = "genes",
  ...
)
```

Arguments

go_list	A list of enriched GO results, output from enrichGO_list()
plot_dotplot	Whether to plot dotplot (default is TRUE)
plot_cnetplot	Whether to plot cnetplot (default is FALSE)
plot_emapplot	Whether to plot emapplot (default is FALSE)
showCategory_dotplot	showCategory parameter of clusterProfiler::dotplot() (default is 5)
showCategory_cnetplot	showCategory parameter of clusterProfiler::cnetplot() (default is 5)
showCategory_emapplot	showCategory parameter of clusterProfiler::emapplot() (default is 5)
fontsize	Font size for the plots (default is 8)
label	Title for the plots (default is "genes")
...	Additional parameters to pass to the plotting functions clusterProfiler::dotplot(), clusterProfiler::cnetplot(), or clusterProfiler::emapplot()

Value

A list of plots visualizing the GO enrichment results.

Examples

```
data(example_go)
plot_GO(example_go$all, plot_dotplot = TRUE,
    plot_emapplot = FALSE, plot_cnetplot = FALSE)
```

plot_ID *Plot number of identified features*

Description

Plot the number of identified features (i.e., features with non-missing values) for each sample

Usage

```
plot_ID(se_obj, fontsize = 8, signif = FALSE, ...)
```

Arguments

se_obj	A SummarizedExperiment object, produced by `create_input()` function, containing the abundance data and associated sample information.
fontsize	Font size for the plot (default is 8) (default is 8).
signif	A logical value indicating whether to perform significance testing between groups and add significance annotations to the plot (default is FALSE).
...	Additional arguments to be passed to `ggpubr::stat_compare_means()` when `signif` is TRUE, for customizing the significance annotations.

Value

A plot showing the ID number for each sample, with a dashed line indicating the average number across all samples. And a boxplot comparing the ID number between groups, with optional significance annotations.

Examples

```
# simulate data with random missing values
na_data <- matrix(rnorm(1500), nrow = 100, ncol = 150)
na_data[sample(length(na_data), size = 2000)] <- NA
na_data <- data.frame(Feature = paste0("Feature", 1:100), na_data)
colnames(na_data)[-1] <- paste0("Sample", 1:150)

na_obj <- create_input(na_data,
    data.frame(Sample = paste0("Sample", 1:150),
    Time = rep(rep(1:10, each = 5), 3),
    Group = rep(c("A", "B", "C"), each = 50),
    Replicate = rep(1:5, 30)))
plot_ID(na_obj, signif = TRUE)
```

plot_missing *Plot missing rate*

Description

Plot the ratio of missing values for each sample

Usage

```
plot_missing(se_obj, fontsize = 8, signif = FALSE, ...)
```

Arguments

se_obj	A SummarizedExperiment object, produced by `create_input()` function, containing the abundance data and associated sample information.
fontsize	Font size for the plot (default is 8) (default is 8).
signif	A logical value indicating whether to perform significance testing between groups and add significance annotations to the plot (default is FALSE).
...	Additional arguments to be passed to `ggpubr::stat_compare_means()` when `signif` is TRUE, for customizing the significance annotations.

Value

A plot showing the missing value ratio for each sample, with a dashed line indicating the global missing value ratio across all samples. And a boxplot comparing the missing value ratio between groups, with optional significance annotations.

Examples

```
# simulate data with random missing values
na_data <- matrix(rnorm(1500), nrow = 100, ncol = 150)
na_data[sample(length(na_data), size = 2000)] <- NA
na_data <- data.frame(Feature = paste0("Feature", 1:100), na_data)
colnames(na_data)[-1] <- paste0("Sample", 1:150)

na_obj <- create_input(na_data,
    data.frame(Sample = paste0("Sample", 1:150),
    Time = rep(rep(1:10, each = 5), 3),
    Group = rep(c("A", "B", "C"), each = 50),
    Replicate = rep(1:5, 30)))
plot_missing(na_obj, signif = TRUE)
```

`plot_modules_h` *Plot modules (horizontal layout)*

Description

Plot WGCNA modules' heatmaps, mean expression profiles, and enrichment terms, aligned horizontally.

Different from running WGCNA, the input data should have the replicates merged, instead of having multiple samples per group, time and feature (gene).

If certain time points are missing in some groups, NA values are added.

Usage

```
plot_modules_h(
  module,
  se_obj_merged,
  scale = TRUE,
  assay,
  ylabel = "Log2 abundance",
  profile_width = 3,
  profile_link_width = 1,
  enrich_list = NULL,
  enrich_category = "BP",
  enrich_rank_by = "p.adjust",
  enrich_top_n = 3,
  enrich_threshold = 0.05,
  fontsize = 8,
  heatmap_width = 8,
  heatmap_height = 8,
  mark_features = NULL,
  width = 16,
  height = 12,
  res = 300,
  suffix = "",
  device = "png",
  save = NULL
)
```

Arguments

`module`	A data frame with columns "Feature" and "Module"
`se_obj_merged`	A SummarizedExperiment object, with one value for each feature at each time point in each group (replicates merged). The colData of the object should contain columns "Sample", "Group", and "Time". The object can be produced by `split_groups()`, `merge_replicates()` and `merge_groups()`.
`scale`	Whether to scale the data (z-score) across samples for each feature (default is TRUE)
`assay`	The assay to use in the SummarizedExperiment object: a numeric index or character name, e.g. 1 or "orig" for original data, 2 or "norm" for time 0 normalised data. (Required, no default.)

ylabel Y axis label prefix (default is "Log2 abundance")

profile_width Width of the mean expression profile panels in cm (default: 3)

profile_link_width
Width of the link between heatmap and mean expression profile panels in cm (default: 1)

enrich_list A named enrichment list for modules, as produced by `enrichGO_list()$all`. Each named element should be a data.frame with columns `Cluster`, `Description`, and the rank column. Accepts any enrichment result with compatible format. When `enrich_category` is a vector, each category is shown as a separate column in the annotation textbox. If NULL (default), no enrichment terms will be displayed.

enrich_category
Category name(s) of enrichment to display: a single string (e.g. `"BP"`), or a vector for multiple categories (e.g. `c("BP", "CC", "Hub features")`). When `"Hub features"` is included, `mark_features` are shown as a column in the textbox.

enrich_rank_by Numeric column used to rank enrichment terms within each module. A single string (default: `"p.adjust"`) is recycled for all categories in `enrich_category`. Supply a character vector of matching length for per-category control. Also used for colour-coding: terms with values below `enrich_threshold` are shown in black, terms above in grey.

enrich_top_n Top N enrichment terms per module. A single integer (default: 3) is recycled for all categories in `enrich_category`. Supply an integer vector of matching length for per-category control.

enrich_threshold
P-value / adjusted p-value threshold for colour-coding enrichment terms. A single value (default: 0.05) is recycled for all categories; supply a vector of matching length for per-category control. Terms below threshold are drawn in black, terms above in grey70. Set to `NULL` to skip colour-coding (all terms black). Set to `NA` for `"Hub features"`.

fontsize Base font size (default: 8)

heatmap_width Width of the heatmap body in cm (default: 8)

heatmap_height Height of the heatmap body in cm (default: 8). Set to `NULL` to auto-size.

mark_features Features to label. Accepts two forms:

- A **character vector**: all marked in black on the left.
- A **named list** of character vectors, e.g. `list("Hub 1" = c("gene1"), "Hub 2" = c("gene2"))`. Names become categories with auto-assigned colours (via `ggsci::pal_jco()`). Auto-routed: shown as `anno_mark` (left side, connecting lines) unless `"Hub features"` is present in `enrich_category`, in which case an `anno_multicol_textbox` (right side) is built instead, using the same colours. Set to `NULL` (default) to skip marking.

width Width of the saved image (default is 16 (cm))

height Height of the saved image (default is 12 (cm))

res Resolution of the saved image (except pdf format) (default: 300 (ppi))

suffix Suffix for the saved image file name (default: "")

device Image file format(s) for saving. Can be a character vector with one or more of `"png"`, `"pdf"`, `"tiff"`, `"jpeg"` (default: `"png"`)

save Directory to save the plot, no saving if is NULL (default is NULL)

Value

A combined plot of heatmap, mean expression profile, and enrichment terms for each WGCNA module

References

https://github.com/junjunlab/ClusterGVis

Examples

```
data(example_obj)
example_obj <- normalise_to_start(example_obj)
example_obj_list <- split_groups(example_obj)
example_obj_merged_list <- merge_replicates(example_obj_list)
example_obj_merged <- merge_groups(example_obj_merged_list)

data(example_net)
# select two modules for demonstration
example_module <- WGCNA_module(example_net) |>
    dplyr::filter(Module %in% c("1", "2"))

data(example_go)

plot_modules_h(example_module |> dplyr::filter(Module != '0'),
    example_obj_merged, assay = 2, scale = TRUE,
    ylabel = "Z-score of log2 expression",
    enrich_list = example_go$all,
    enrich_category = "BP",
    heatmap_width = 6, heatmap_height = 4)
```

---

plot_modules_v *Plot modules (vertical layout)*

Description

Plot WGCNA modules' mean expression profiles and heatmaps, align vertically.

Different from running WGCNA, the input data should have the replicates merged, instead of having multiple samples per group, time and feature (gene).

If certain time points are missing in some groups, NA values are added.

Usage

```
plot_modules_v(
  module,
  se_obj_merged,
  scale = TRUE,
  assay,
  ylabel = "Log2 abundance",
  suffix = "",
  device = "png",
  save = NULL,
  width = 12,
```

```
  height = 8,
  height_ratio = 2,
  fontsize = 8,
  res = 300
)
```

Arguments

module	A data frame with columns "Feature" and "Module"
se_obj_merged	A SummarizedExperiment object, with one value for each feature at each time point in each group (replicates merged). The colData of the object should contain columns "Sample", "Group", and "Time". The object can be produced by `split_groups()`, `merge_replicates()` and `merge_groups()`.
scale	Whether to scale the data (z-score) across samples for each feature (default is TRUE)
assay	The assay to use in the SummarizedExperiment object: a numeric index or character name, e.g. 1 or "orig" for original data, 2 or "norm" for time 0 normalised data. (Required, no default.)
ylabel	Y axis label prefix (default is "Log2 abundance")
suffix	Suffix for the saved image file name (default is an empty string)
device	Image file format(s) for saving. Can be a character vector, e.g. `c("png", "pdf")`, to save in multiple formats (default: `"png"`).
save	Directory to save the plot, no saving if is NULL (default is NULL)
width	Width of the saved image (default is 12 (cm))
height	Height of the saved image (default is 8 (cm))
height_ratio	Ratio of height of heatmap to the line plot (default is 2)
fontsize	Font size for the plot (default is 8)
res	Resolution of the saved image (except pdf format) (default is 300 (ppi))

Value

Two plots aligned vertically by groups: the top one is a line plot of module feature mean expression profiles, the bottom one is a heatmap of feature expression across time points.

Examples

```
data(example_obj)
example_obj <- normalise_to_start(example_obj)
example_obj_list <- split_groups(example_obj)
example_obj_merged_list <- merge_replicates(example_obj_list)
example_obj_merged <- merge_groups(example_obj_merged_list)

data(example_net)
example_module <- WGCNA_module(example_net)

plot_modules_v(example_module |> dplyr::filter(Module != '0'),
    example_obj_merged, assay = 2, scale = TRUE,
    ylabel = "Z-score of log2 (expression)",
    height_ratio = 2,
    fontsize = 6)
```

plot_pca *Plot PCA*

Description

Plot principal component analysis (PCA), using features without missing values

The samples are coloured by Group and sized by Time. Ellipses are drawn for each Group.

Usage

```
plot_pca(
  se_obj,
  plot_screeplot = TRUE,
  plot_loadings = TRUE,
  plot_morepc = TRUE,
  morepc = seq(1, 5),
  pc1 = 1,
  pc2 = 2,
  circle = FALSE,
  xlim_min = NULL,
  xlim_max = NULL,
  ylim_min = NULL,
  ylim_max = NULL,
  fontsize = 8,
  assay = 1
)
```

Arguments

se_obj	A SummarizedExperiment object created by `create_input()`
plot_screeplot	Logical, whether to plot screeplot with `PCAtools::screeplot()` (default is TRUE)
plot_loadings	Logical, whether to plot loadings with `PCAtools::plotloadings()` (default is TRUE)
plot_morepc	Logical, whether to plot pairs of more PCs with `PCAtools::pairsplot()` (default is TRUE)
morepc	A numeric vector of PCs to plot in the pairs plot (default is 1:5)
pc1	Numeric, which PC to use for the x-axis (default is 1)
pc2	Numeric, which PC to use for the y-axis (default is 2)
circle	Logical, whether to draw circles (ellipses) around samples of each group (default is FALSE)
xlim_min	Minimum x-axis limit when drawing ellipses (default: NULL, auto-computed with 0.35x padding around the PC1 range)
xlim_max	Maximum x-axis limit when drawing ellipses (default: NULL, auto-computed with 0.35x padding around the PC1 range)
ylim_min	Minimum y-axis limit when drawing ellipses (default: NULL, auto-computed with 0.35x padding around the PC2 range)

ylim_max	Maximum y-axis limit when drawing ellipses (default: NULL, auto-computed with 0.35x padding around the PC2 range)
fontsize	Font size for the PCA plot (default is 8)
assay	The assay to use in the SummarizedExperiment object: a numeric index or character name, e.g. 1 or "orig" for original data, 2 or "norm" for time 0 normalised data. (Default: 1.)

Value

A PCA plot showing the distribution of samples, other plots provided by PCAtools package, and PCAtools output object for custom plotting.

Examples

```
data(example_obj)
PC = plot_pca(example_obj, morepc = seq(1, 3))
```

---

plot_pca_3D *Plot PCA in 3D*

Description

Plot principal component analysis (PCA) results in 3D. This function takes the output PCA object of the plot_pca function and visualizes the samples in a 3D space defined by the specified principal components.

The samples are coloured by Group and sized by Time.

Usage

```
plot_pca_3D(pca, pcs = seq(1, 3))
```

Arguments

pca	A PCA object returned by the plot_pca() function ($pca) or PCAtools::pca().
pcs	A numeric vector specifying which three principal components to plot (default is 1:3)

Value

An interactive 3D PCA plot showing the distribution of samples in the space defined by the specified principal components. The samples are coloured by Group and sized by Time.

Examples

```
data(example_obj)
PC = plot_pca(example_obj, morepc = seq(1, 3))
if (requireNamespace("plotly", quietly = TRUE)) {
    plot_pca_3D(PC$pca, pcs = seq(1, 3))
}
```

`plot_pca_arrows` *Plot PCA with arrows*

Description

Plot PCA with arrows indicating trajectory over time, for a single group.

Usage

```
plot_pca_arrows(
  se_obj,
  circle = TRUE,
  arrow = TRUE,
  pc1 = 1,
  pc2 = 2,
  fontsize = 8,
  assay = 1
)
```

Arguments

`se_obj`	A SummarizedExperiment object (single group).
`circle`	Logical, whether to draw ellipses around samples of each time point (default: TRUE).
`arrow`	Logical, whether to draw arrows indicating the trajectory over time (default: TRUE).
`pc1`	Principal component for the x-axis (default: 1).
`pc2`	Principal component for the y-axis (default: 2).
`fontsize`	Font size for the plot (default is 8)
`assay`	The assay to use in the SummarizedExperiment object: a numeric index or character name, e.g. 1 or "orig" for original data, 2 or "norm" for time 0 normalised data. (Default: 1.)

Value

A PCA plot coloured by Time, with optional ellipses and trajectory arrows.

Examples

```
data(example_obj)
plot_pca_arrows(example_obj[, example_obj$Group == "IFNbeta"])
```

plot_pca_by_group *Plot PCA by group*

Description

Plot PCA for each group separately, with optional circles around time points and arrows indicating trajectory over time. Accepts either a SummarizedExperiment object or a list of them (from `split_groups()`).

Usage

```
plot_pca_by_group(
  se_obj,
  circle = TRUE,
  arrow = TRUE,
  pc1 = 1,
  pc2 = 2,
  nrow = 1,
  fontsize = 8,
  assay = 1,
  legend_pos = "right"
)
```

Arguments

`se_obj`	A SummarizedExperiment object, or a named list of them (e.g. from `split_groups()`).
`circle`	Logical, whether to draw ellipses around samples of each time point (default: TRUE).
`arrow`	Logical, whether to draw arrows indicating the trajectory over time (default: TRUE).
`pc1`	Principal component for the x-axis (default: 1).
`pc2`	Principal component for the y-axis (default: 2).
`nrow`	Number of rows for arranging the plots (default: 1).
`fontsize`	Font size for the plot (default is 8)
`assay`	The assay to use in the SummarizedExperiment object: a numeric index or character name, e.g. 1 or "orig" for original data, 2 or "norm" for time 0 normalised data. (Default: 1.)
`legend_pos`	Legend position (default: "right").

Value

A series of PCA plots showing the distribution of samples in each group, coloured by Time.

Examples

```
data(example_obj)
plot_pca_by_group(example_obj, circle = TRUE, arrow = TRUE)

# Also accepts a list from split_groups()
example_obj_list <- split_groups(example_obj)
plot_pca_by_group(example_obj_list)
```

plot_segments *Plot segmented regression*

Description

Plot segmented regression results for specified features in each group, using the `Trendy::plotFeature()` function. The input is a list of SummarizedExperiment objects with no missing values and a list of Trendy analysis results for each group.

Usage

```
plot_segments(
  se_obj_imp_list,
  res_list,
  feature,
  group = NULL,
  nrow = NULL,
  ...
)
```

Arguments

`se_obj_imp_list`	A list of SummarizedExperiment objects with no missing values
`res_list`	A list of the Trendy analysis results, output of `run_Trendy()`
`feature`	A character vector of feature names to be plotted
`group`	A character vector of group names to be analysed. If NULL, all groups in the input list will be used (default is NULL)
`nrow`	Number of rows in the plot layout. If NULL and multiple features are provided, it will be set to half the number of features (rounded up) (default is NULL)
`...`	Additional arguments to be passed to the `Trendy::plotFeature()`

Value

Plots of the segmented regression fits for the specified features in each group.

Examples

```
data(example_obj)
example_obj <- normalise_to_start(example_obj)
example_obj_list <- split_groups(example_obj)
example_obj_merged_list <- merge_replicates(example_obj_list)
example_obj_merged_list <-
    calc_feature_property(example_obj_merged_list, threshold = 0)
# no missing value in the example dataset, so imputation is not necessary
example_obj_merged_imp_list <- impute_groups(example_obj_merged_list)

data("example_res_list")
plot_segments(example_obj_merged_imp_list, example_res_list,
    feature = c("Kif3a"))
```

plot_trend *Plot feature abundance over time*

Description

Plot trend of feature abundances / expression over time by mean and standard deviation (SD) for each group. Accepts either a SummarizedExperiment object (computing mean/SD internally via `calc_mean_sd()`) or a pre-computed table from `calc_mean_sd()`.

Usage

```
plot_trend(
  se_obj,
  assay = 1,
  groups = NULL,
  features,
  title = NULL,
  ylabel = "Log2 abundance",
  errorbar = TRUE,
  fontsize = 8
)
```

Arguments

`se_obj`	A SummarizedExperiment object, or a data.frame from `calc_mean_sd()` with columns: Feature, Time, Group, Mean, SD.
`assay`	The assay to use in the SummarizedExperiment object: a numeric index or character name, e.g. 1 or "orig" for original data, 2 or "norm" for time 0 normalised data. (Default: 1.)
`groups`	Groups to be plotted, if NULL, all groups will be used (default is NULL)
`features`	Features to be plotted, if NULL, an error will be raised
`title`	Title of the plot. "Feature" if NULL (default).
`ylabel`	Y axis label of the plot (default is "Log2 abundance")
`errorbar`	Whether to plot error bars (default is TRUE)
`fontsize`	Font size for the plot (default is 8)

Value

Plot of feature abundances over time by mean and SD

Examples

```
data(example_obj)
plot_trend(example_obj,
    features = sample(rownames(example_obj), 4))
```

plot_umap *Plot UMAP*

Description

Plot UMAP of samples, using features without missing values

The UMAP plots are labelled by Group, Time, Replicate (if more than 1), Batch (if more than 1), and number of identified features (non-missing values).

Usage

```
plot_umap(
  se_obj,
  seed = 1234,
  plot_ID = FALSE,
  circle = FALSE,
  xlim_min = NULL,
  xlim_max = NULL,
  ylim_min = NULL,
  ylim_max = NULL,
  umap_neighbors = NULL,
  fontsize = 8,
  assay = 1
)
```

Arguments

se_obj	A SummarizedExperiment object created by `create_input()`
seed	Random seed for UMAP (default is 1234)
plot_ID	Whether to include a UMAP plot coloured by number of identified features (default is FALSE)
circle	Logical, whether to draw circles (ellipses) around samples of each group (default is FALSE)
xlim_min	Minimum x-axis limit when drawing ellipses (default: NULL, auto-computed with 0.35x padding around the UMAP x range)
xlim_max	Maximum x-axis limit when drawing ellipses (default: NULL, auto-computed with 0.35x padding around the UMAP x range)
ylim_min	Minimum y-axis limit when drawing ellipses (default: NULL, auto-computed with 0.35x padding around the UMAP y range)
ylim_max	Maximum y-axis limit when drawing ellipses (default: NULL, auto-computed with 0.35x padding around the UMAP y range)
umap_neighbors	UMAP n_neighbors parameter (default is selected by `.umap_n_neighbors()` function based on the number of samples)
fontsize	Font size for the plot (default is 8)
assay	The assay to use in the SummarizedExperiment object: a numeric index or character name, e.g. 1 or "orig" for original data, 2 or "norm" for time 0 normalised data. (Default: 1.)

Value

A list of UMAP plots showing the distribution of samples. And a data frame containing UMAP coordinates and sample annotations for custom plotting.

Examples

```
data(example_obj)
umap_layout <- plot_umap(example_obj)
```

`plot_umap_by_group` *Plot UMAP by group*

Description

Plot UMAP for each group separately. Accepts either a SummarizedExperiment object or a named list of them (from `split_groups()`).

Usage

```
plot_umap_by_group(
  se_obj,
  seed = 1234,
  nrow = 1,
  umap_neighbors = NULL,
  fontsize = 8,
  assay = 1,
  legend_pos = "right"
)
```

Arguments

`se_obj`	A SummarizedExperiment object, or a named list of them (e.g. from `split_groups()`).
`seed`	Random seed for UMAP (default: 1234).
`nrow`	Number of rows for arranging the plots (default: 1).
`umap_neighbors`	UMAP n_neighbors parameter (default: auto-selected based on sample count).
`fontsize`	Base font size (default: 8).
`assay`	The assay to use in the SummarizedExperiment object: a numeric index or character name, e.g. 1 or "orig" for original data, 2 or "norm" for time 0 normalised data. (Default: 1.)
`legend_pos`	Legend position (default: "right").

Value

A series of UMAP plots showing the distribution of samples in each group, coloured by Time.

Examples

```
data(example_obj)
plot_umap_by_group(example_obj)

# Also accepts a list from split_groups()
example_obj_list <- split_groups(example_obj)
plot_umap_by_group(example_obj_list)
```

`plot_variance` *Plot variance decomposition*

Description

Based on `PALMO::variancefeaturePlot()` function

Usage

```
plot_variance(
  var_decomp,
  rank = "Group",
  features = NULL,
  top_n = 20,
  show_ylab = TRUE,
  fontsize = 8
)
```

Arguments

`var_decomp`	A data frame output from `decomp_variance()`
`rank`	Rank the plot by selected variable (default is "Group")
`features`	Features to be plotted (default is NULL)
`top_n`	Top n features ranked by `rank` to be plotted when `features` is not specified (default is 20)
`show_ylab`	Whether to show y axis text (default is TRUE)
`fontsize`	Font size for the plot (default is 8)

Value

A stacked bar plot showing the percentage of variance explained by each model component (Group, Time, Residual, plus Subject and interaction terms when present). Features are ranked by the specified `rank` variable (Group or Time) and only the top n features are plotted if `features` is not specified.

Examples

```
data(example_obj)
example_obj <- normalise_to_start(example_obj)

var_decomp <- decomp_variance(example_obj, assay = 1)
plot_variance(var_decomp, rank = "Time", top_n = 20)
```

plot_volcano *Volcano plot of DE results*

Description

This function creates a volcano plot to visualize the results of differential expression (DE) analysis performed by DE_between_group() or DE_between_time(). The plot displays log2 fold change on the x-axis and -log10 adjusted p-value on the y-axis, with DE features highlighted based on specified thresholds. If no thresholds are provided, the function will use the thresholds that were applied when running DE_between_group() or DE_between_time().

Usage

```
plot_volcano(
  DE_out,
  group1,
  group2,
  time,
  group,
  time1,
  time2,
  logFC_thres = NULL,
  adjP_thres = NULL,
  label = FALSE,
  fontsize = 8,
  ...
)
```

Arguments

DE_out	Output from DE_between_group() or DE_between_time()
group1	(Required for DE_between_group() output) Name of the first group for comparison
group2	(Required for DE_between_group() output) Name of the second group for comparison
time	(Required for DE_between_group() output) Time point for comparison
group	(Required for DE_between_time() output) Name of the group for comparison
time1	(Required for DE_between_time() output) First time point for comparison
time2	(Required for DE_between_time() output) Second time point for comparison
logFC_thres	Log2 fold change threshold for highlighting DE features, default is NULL (using threshold when DE_between_group() or DE_between_time() was run)
adjP_thres	Adjusted p-value threshold for highlighting DE features, default is NULL (using threshold when DE_between_group() or DE_between_time() was run)
label	Whether to label names of DE features on the plot (default is FALSE)
fontsize	Font size for the plot (default is 8)
...	Additional arguments for ggrepel::geom_text_repel() when label = TRUE

Value

A volcano plot of DE results

Examples

```
data(example_obj)
example_obj <- normalise_to_start(example_obj)

DE_between_group_out <- DE_between_group(example_obj, assay = 2)
plot_volcano(DE_between_group_out, group1 = "untreated",
    group2 = "IFNbeta", time = 24,
    logFC_thres = 0.5, adjP_thres = 0.05, label = TRUE)
```

---

plot_WGCNA *Plot WGCNA results*

Description

Plot WGCNA module eigengenes and module-group/time correlations based on output of `run_WGCNA()`

Usage

```
plot_WGCNA(net, fontsize = 8)
```

Arguments

`net`	WGCNA network object output by `run_WGCNA()`
`fontsize`	Font size for plots (default is 8)

Value

Plots of WGCNA module dendrogram, module eigengenes, pairwise scatterplots of eigengenes, clustering of module eigengenes, and module-trait correlation heatmap

References

https://github.com/edo98811/WGCNA_official_documentation/blob/main/FemaleLiver-03-relateModsToExt.R

Examples

```
data(example_obj)
example_obj <- normalise_to_start(example_obj)

data("example_net")
plot_WGCNA(example_net, fontsize = 8)
```

`prepare_tide` *Prepare TiDEomics input*

Description

One-step data preparation wrapper for TiDEomics: creates the SummarizedExperiment object, normalises to starting time point, filters out group-specific features, runs variance decomposition, and filters out noisy features with high residual variance. Returns both unmerged and merged (mean of replicates) outputs, with all-feature and filtered versions of each, ready for downstream analysis.

Usage

```
prepare_tide(
  data,
  sample_ann,
  subject_col = NULL,
  replicate_col = NULL,
  batch_col = NULL,
  feature_threshold = 0,
  filter_ratio = 0.5,
  min_groups = 2,
  keep = c("below_quantile", "top_n", "threshold"),
  residual_threshold = NULL,
  n_keep = 5000,
  interaction = FALSE,
  n_cores = 1
)
```

Arguments

`data`	A data frame with rows as features (e.g., genes, proteins) and columns as samples. The first column should contain feature identifiers (e.g., gene symbols) and be named 'Feature'. The data should have been normalised and log transformed. Missing values are allowed.
`sample_ann`	A data frame containing sample annotations with required columns: 'Sample', 'Group', and 'Time'. 'Replicate' and 'Batch' are optional columns and will be auto-generated if not provided. 'Time', 'Replicate' and 'Batch' should be numeric.
`subject_col`	Optional: name of a column in `sample_ann` identifying biological subjects measured repeatedly across time points (e.g., `"PatientID"`). If provided, the column is renamed to 'Subject' and used for repeated-measures analyses downstream. If NULL (default), all samples are treated as independent, e.g. for cell culture experiments.
`replicate_col`	Optional: name of a column in `sample_ann` identifying replicate IDs. If provided, the column is renamed to 'Replicate'. If NULL (default), the function looks for a column named 'Replicate'; if absent, replicate IDs are auto-generated within each Group and Time (and Subject, if provided).
`batch_col`	Optional: name of a column in `sample_ann` identifying batch information. If provided, the column is renamed to 'Batch'. If NULL (default), the function looks for a column named 'Batch'; if absent, all samples are assigned to batch 1.

feature_threshold
: Threshold for calc_feature_property(). Feature values > feature_threshold are considered expressed (default: 0).

filter_ratio
: Minimum ratio of time points a feature must be expressed in, to be considered expressed in a group (default: 0.5, meaning that a feature must be > feature_threshold in >=50% time points in a group to be considered expressed). Passed to group_specific_features().

min_groups
: Minimum number of groups a feature must be expressed in, with the above filter_ratio, to pass the cross group filter (default: 2). Passed to group_specific_features().

keep
: How to select features after variance decomposition: "below_quantile" (default): keep features with residual variance below residual_threshold quantile (0-1 scale). "top_n": keep the top n_keep lowest-residual features. "threshold": keep features with residual percentage below residual_threshold (0-100 scale; use 100 to skip filtering).

residual_threshold
: Numeric: when keep = "below_quantile", the quantile of residual variance to use as cutoff (0-1 scale). When keep = "threshold", the percentage cutoff (0-100 scale, 100 = keep all features). (default: NULL, auto-assigned as 0.75 for "below_quantile", 100 for "threshold")

n_keep
: Number of features to keep when keep = "top_n" (default: 5000).

interaction
: Passed to decomp_variance(): if TRUE, adds (1|Group:Time) (or (1|Subject:Time) when Subject present) interaction term (default: FALSE).

n_cores
: Number of cores for variance decomposition (default: 1).

Details

The pipeline:

1. create_input(): validate and build SE
2. normalise_to_start(): add time-0-normalised assay
3. split_groups() -> merge_replicates() -> calc_feature_property() -> summarise_feature_property(): per-feature expression statistics
4. group_specific_features(): filter out group-specific features
5. decomp_variance() : LMM variance decomposition on both "orig" and "norm" assays
6. Residual filter: filter out features with high residual variance, calculated from decomp_variance() on each assay
7. Produce filtered-unmerged and filtered-merged SEs for each assay

Value

A named list with elements. The elements suffixed with "_filtered" have passed both the cross-group and residual filters, and are split into $orig and $norm sub-lists, with variance decomposition and residual filtering done per assay.

1. se (SE with replicates, all features, both assays);
2. se_filtered (a list of two SEs with replicates, filtered features by orig / norm, use for WGCNA);
3. merged_list (per-group merged SEs, all features, use for Trendy);

4. merged_list_filtered (a list of two per-group merged SEs, filtered features by orig / norm);
5. merged_se (single merged SE, all groups combined, all features, use for plot_modules_v/h);
6. merged_se_filtered (a list of two single merged SEs, filtered features by orig / norm, use for plot_modules_v/h);
7. variance (a list of two variance decomposition data.frames, computed from orig / norm assay);
8. feature_property (feature property summary, all features);
9. filter_summary (per-stage filtering statistics);
10. group_specific_filter (a vector of group-specific features, removed by the cross-group filter); 11-13. DE, enrichment, WGCNA (empty on initialization).

Examples

```
data(tutorial_data)
data(tutorial_sample_info)
tide <- prepare_tide(tutorial_data, tutorial_sample_info,
    keep = "threshold", residual_threshold = 100)
tide$filter_summary
```

---

prepare_WGCNA *Prepare data and choose power for WGCNA*

Description

Prepare input data for WGCNA (format, QC), then choose the appropriate soft thresholding power for network construction, by analysing scale free topology with different soft thresholding powers.

Usage

```
prepare_WGCNA(
  se_obj,
  assay,
  networkType = "signed",
  RsquaredCut = 0.8,
  MeanConnectivity = 100,
  powers = NULL,
  fontsize = 8,
  ...
)
```

Arguments

se_obj	A SummarizedExperiment object. Data normalised to time point 0 can be in the second assay slot, created by normalise_to_start(). Features can be pre-filtered, e.g. by residual variance calculated by decomp_variance(), to remove noisy features before running WGCNA.
assay	The assay to use in the SummarizedExperiment object: a numeric index or character name, e.g. 1 or "orig" for original data, 2 or "norm" for time 0 normalised data. (Required, no default.)

networkType	Parameter of WGCNA::pickSoftThreshold() (default is "signed")
RsquaredCut	Parameter of WGCNA::pickSoftThreshold() (default is 0.8)
MeanConnectivity	Line of mean connectivity (default is 100)
powers	Parameter of WGCNA::pickSoftThreshold() (default: NULL, auto-assigned as seq(1, 20))
fontsize	Font size for the plot (default is 8)
...	Additional parameters to be passed to WGCNA::pickSoftThreshold()

Value

A list containing results of the scale-free topology fit indices with different powers, suggested power, network type and prepared input data used for reuse in run_WGCNA()

References

https://github.com/edo98811/WGCNA_official_documentation/

Examples

```
data(example_obj)
example_obj <- normalise_to_start(example_obj)

wgcna_input <- prepare_WGCNA(example_obj, assay = 2, powers = seq(1, 30),
    networkType = "signed", RsquaredCut = 0.8)
wgcna_input$fitIndices
picked_power <- wgcna_input$powerEstimate
```

---

run_Trendy *Segmented regression analysis*

Description

Run segmented regression analysis with Trendy on imputed data for multiple groups of samples in a list of SummarizedExperiment objects. The results will be returned in a list format. Each element in the list corresponds to one group of samples and contains the Trendy analysis results for that group.

If the feature parameter is not specified, all features expressed in at least minExp time points in each group will be included in the analysis for the group. calc_feature_property() should be run before imputation to calculate the expression ratio for each feature in each group. Missing values can be imputed after calc_feature_property() using the impute_groups() function.

If a group has less than 4 time points are available, Trendy analysis cannot be performed and NULL will be returned for the group.

Usage

```
run_Trendy(
  se_obj_imp_list,
  group = NULL,
  feature = NULL,
  minExp = 0.5,
  maxK = 1,
  meanCut = 0,
  minNumInSeg = 3,
  NCores = 1,
  ...
)
```

Arguments

se_obj_imp_list	A list of SummarizedExperiment objects with no missing values
group	A character vector of group names to be analysed. If NULL, all groups in the input list will be used (default is NULL)
feature	A character vector of feature names to be included in the analysis. If NULL, all features with expression ratio (Exp_ratio) >= minExp will be used, based on results of `calc_feature_property()` before imputation. (default is NULL)
minExp	Minimum expression ratio for a feature to be included in the analysis (default is 0.5)
maxK	Parameter of `Trendy::trendy()`, maximum number of breakpoints allowed in the segmented regression model (default is 1)
meanCut	Parameter of `Trendy::trendy()`, minimum mean expression required for a feature to be included in the analysis (default is 0)
minNumInSeg	Parameter of `Trendy::trendy()`, minimum number of samples required in each segment (default is 3)
NCores	Number of cores to use for parallel processing (default is 1)
...	Additional arguments to be passed to the `Trendy::trendy()`

Value

A list of Trendy analysis results, including the fitted model parameters and statistics for each feature in each group.

Examples

```
data(example_obj)
example_obj <- normalise_to_start(example_obj)
example_obj_list <- split_groups(example_obj)
example_obj_merged_list <- merge_replicates(example_obj_list)
example_obj_merged_list <- calc_feature_property(example_obj_merged_list,
    threshold = 0)
# no missing value in the example dataset, so imputation is not necessary
example_obj_merged_imp_list <- impute_groups(example_obj_merged_list)

# "untreated" group has only 3 time points, so Trendy analysis will not be
# performed for this group
```

```
example_res_list <- run_Trendy(example_obj_merged_imp_list, maxK = 1,
    minNumInSeg = 2, meanCut = 0,
    feature = rownames(example_obj_merged_imp_list[[1]])[1:10])
```

---

run_WGCNA *Weighted gene co-expression network analysis*

Description

Run WGCNA based on output of `prepare_WGCNA()`, to identify co-expression modules of features with `WGCNA::blockwiseModules()`

Usage

```
run_WGCNA(wgcna_input, power, numericLabels = TRUE, ...)
```

Arguments

`wgcna_input`	A list output by `prepare_WGCNA()`, containing the prepared input data and networkType parameter used in power selection.
`power`	Soft-thresholding power to be used in `WGCNA::blockwiseModules()`, selected automatically or manually based on the output of `prepare_WGCNA()`
`numericLabels`	Whether to use numeric labels for modules in the output (default is TRUE)
`...`	Additional parameters to be passed to `WGCNA::blockwiseModules()`

Value

The built network and parameters of `WGCNA::blockwiseModules()`, and the input data and sample information for use in `plot_WGCNA()`.

References

https://github.com/edo98811/WGCNA_official_documentation/blob/main/FemaleLiver-03-relateModsToExt.R

Examples

```
data(example_obj)
example_obj <- normalise_to_start(example_obj)

wgcna_input <- prepare_WGCNA(example_obj, assay = 2, powers = seq(1, 30),
    networkType = "signed", RsquaredCut = 0.8)
wgcna_input$fitIndices
picked_power <- wgcna_input$powerEstimate
example_net <- run_WGCNA(wgcna_input,
    power = picked_power,
    minModuleSize = 10, # only 100 genes in the example data
    numericLabels = TRUE)
```

set_custom_palette *Set custom color palette*

Description

Set a custom color palette for groups to use in all subsequent plotting functions that support custom palettes.

Usage

```
set_custom_palette(palette)
```

Arguments

palette	A named vector where names correspond to group names and values are the corresponding colors (e.g., `c("Group1" = "#FF0000", "Group2" = "#00FF00")`).

Value

The function does not return a value but sets the custom palette as an option that can be accessed by other plotting functions. The palette will be stored in the R session options under the name "custom.palette".

Examples

```
custom_palette <- c("untreated" = "#1b9e77", "IFNbeta" = "#d95f02",
    "IFNgamma" = "#7570b3", "LPS" = "#e7298a")
set_custom_palette(custom_palette)
```

split_groups *Split groups*

Description

Split SummarizedExperiment object by groups

Usage

```
split_groups(se_obj)
```

Arguments

se_obj	A SummarizedExperiment object created by `create_input()`

Value

A list of SummarizedExperiment objects for each group in the input object. Each object in the list corresponds to one group of samples.

Examples

```
data(example_obj)
example_obj <- normalise_to_start(example_obj)
example_obj_list <- split_groups(example_obj)
```

---

summarise_feature_property *Summarise feature properties*

Description

This function takes a list of merged SummarizedExperiment objects, which are the output of the calc_feature_property() function, and summarizes the feature properties across all groups. It extracts the row data from each SummarizedExperiment object, combines them into a single data frame, and returns this summary data frame. Each row in the resulting data frame represents a feature, and the columns contain the properties of that feature for each group, including the feature name, group name, proportion of NA values for that feature in that group, randomness p-value, and maximum fold change over time.

Usage

```
summarise_feature_property(se_obj_merged_list)
```

Arguments

se_obj_merged_list	A list of merged SummarizedExperiment objects, output of calc_feature_property() function

Value

A data frame summarizing the feature properties across all groups, with each row representing a feature and columns containing the properties including the feature name, group name, and the proportion of NA values for that feature in that group, randomness p-value, and maximum fold change over time.

Examples

```
data(example_obj)
example_obj <- normalise_to_start(example_obj)
example_obj_list <- split_groups(example_obj)
example_obj_merged_list <- merge_replicates(example_obj_list)

example_obj_merged_list <-
    calc_feature_property(example_obj_merged_list, threshold = 0)
property_random_fc <- summarise_feature_property(example_obj_merged_list)
```

summarise_module_metrics
Summarise WGCNA module metrics

Description

Computes per-module metrics for a WGCNA network: module size, proportion of features assigned, and module membership (kME). Unassigned module is excluded.

Usage

```
summarise_module_metrics(net)
```

Arguments

net	A WGCNA network object from run_WGCNA().

Details

MeanKME and MeanKME2 are the mean and mean squared module membership (kME = cor(feature, module eigengene)). MeanKME2 = mean(kME^2) is the average proportion of per-feature variance explained by the module eigengene.

Value

A data frame with columns: Module, Size, Proportion, MeanKME, MeanKME2, MedianKME, SDKME, MinKME, MaxKME.

Examples

```
data(example_net)
summarise_module_metrics(example_net)
```

summarise_module_pattern
Summarise module patterns

Description

Summarise module patterns by integrating WGCNA output with Trendy results

Usage

```
summarise_module_pattern(module, trendy_summary)
```

Arguments

module	A data frame with columns "Feature" and "Module"
trendy_summary	A data frame of trendy results, output of summarise_Trendy()

Value

A list of data frames, each data frame shows the pattern counts in a module

Examples

```
data("example_res_list")
trendy_summary <- summarise_Trendy(example_res_list)

data(example_net)
example_module <- WGCNA_module(example_net)
summarise_module_pattern(example_module, trendy_summary)
```

---

summarise_Trendy *Summarise Trendy results*

Description

Summarise Trendy results into data frame

Usage

```
summarise_Trendy(res_list, ...)
```

Arguments

res_list	A list of Trendy analysis results, output of run_Trendy()
...	Additional arguments to be passed to the Trendy::topTrendy()

Value

A data frame of summary results of Trendy, including breakpoints, segment slopes and p-values for each fitted feature in each group. A "Pattern" column is added to show the combination of segment trends (up, down, stable) for each feature.

Examples

```
data("example_res_list")
trendy_summary <- summarise_Trendy(example_res_list)
```

theme_custom *Custom ggplot2 theme*

Description

A minimal theme used by all TiDEomics plotting functions, built on `theme_minimal()`. Exported so users can apply it to their own plots for consistent styling.

Usage

```
theme_custom(base_size = 8, panel_border = FALSE, legend_position = "right")
```

Arguments

`base_size`	Base font size (default: 8).
`panel_border`	Logical, whether to draw a border around the panel (default: FALSE).
`legend_position`	Legend position (default: "right").

Value

A ggplot2 theme object.

Examples

```
library(ggplot2)
ggplot(mtcars, aes(wt, mpg)) +
    geom_point() +
    theme_custom(base_size = 10)
```

tutorial_data *Dataset for TiDEomics tutorial, expression matrix*

Description

A subset of GSE263759 data set published in Integrated time-series analysis and high-content CRISPR screening delineate the dynamics of macrophage immune regulation

Usage

```
data(tutorial_data)
```

Format

A data.frame with 500 rows and 41 variables:

Feature Feature ID, e.g. gene symbols

Sample1, Sample2, ... log2(CPM + 1) normalised expression values

Details

Code for preparing the data is available in `inst/script/tutorial_input.R`

- Ensembl IDs were mapped to symbols, genes with all zero counts were excluded.
- Use time 0 untreated samples for other groups' time 0.
- Include "untreated", "IFNbeta", "IFNgamma", "LPS" groups.
- Sample 500 random genes.
- Normalised to log2(CPM + 1).

Source

GSE263759

`tutorial_sample_info` *Dataset for TiDEomics tutorial, sample information*

Description

A subset of GSE263759 data set published in Integrated time-series analysis and high-content CRISPR screening delineate the dynamics of macrophage immune regulation

Usage

```
data(tutorial_sample_info)
```

Format

A data.frame with 40 rows and 5 variables:

Sample Sample ID

Group Experimental group (untreated, different treatments)

Time Time point

Replicate Replicate ID for each group and time point

Batch Batch information

Details

Code for preparing the data is available in `inst/script/tutorial_input.R`

- Ensembl IDs were mapped to symbols, genes with all zero counts were excluded.
- Use time 0 untreated samples for other groups' time 0.
- Include "untreated", "IFNbeta", "IFNgamma", "LPS" groups.
- Sample 500 random genes.

Source

GSE263759

WGCNA_module *Convert WGCNA output to feature-module data frame*

Description

Converts the module assignments from `run_WGCNA()` output into two-column data.frame with `Feature` and `Module` columns, expected by `plot_modules_v()`, `plot_modules_h()`, `summarise_module_pattern()`, and all enrichment functions.

Usage

```
WGCNA_module(net, exclude_grey = FALSE)
```

Arguments

`net`	The output of `run_WGCNA()` (a list containing `$colors`)
`exclude_grey`	Logical. If `TRUE`, features assigned to module `0` (grey / unassigned) are removed. Default: FALSE

Value

A data.frame with columns `Feature` (character) and `Module` (factor ordered by decreasing module size). When `exclude_grey = TRUE`, grey/unassigned features are excluded.

Examples

```
data(example_net)
module <- WGCNA_module(example_net)
```

Index