Package: TiDEomics (via r-universe)

November 25, 2026

Title Time-course Differential Expression analysis of omics data

Version 1.0.0

Date 2026-11-17

Description TiDEomics provides an integrated workflow for multi-group time-course omics data analysis. It uses model-based differential expression, variance decomposition, and co-expression module analysis with WGCNA to characterise temporal, group, and group-specific temporal effects. Multi-dataset and multi-omics comparisons and consensus module identification can be performed using shared features. The package includes quality control, data processing, functional enrichment, and integrated visualisation. It supports data with missing values, and operates on SummarizedExperiment objects for interoperability with the Bioconductor ecosystem.

License GPL (>= 2)

URL https://bioconductor.org/packages/TiDEomics/, 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, 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 (>= 4.5.0)

LazyData false

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

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.1

Config/pak/sysreqs cmake libfontconfig1-dev libfreetype6-dev libglpk-dev make libicu-dev libpng-dev libuv1-dev libxml2-dev libssl-dev perl python3 zlib1g-dev

Repository Bioconductor 3.24

Date/Publication 2026-11-23 22:56:32 UTC

RemoteUrl https://github.com/bioc/TiDEomics

RemoteRef RELEASE_3_24

RemoteSha d8880125a939c6164277cf604381c04bb05a4987

Contents

calc_feature_property . . . 4
calc_mean_sd . . . 5
combine_tide . . . 5
create_input . . . 6
DE_between_group . . . 7
DE_between_time . . . 9
DE_timecourse . . . 10
decomp_variance . . . 12
enrich_msigdb . . . 14
enrichGO_list . . . 16
enrichGO_rank . . . 17
enrichR_list . . . 19
example_cons . . . 20
example_go . . . 21
example_net . . . 22
example_net_2 . . . 22
example_obj . . . 23
example_obj_2 . . . 23
example_res_list . . . 24
extract_consensus_hubs . . . 24
extract_hubs . . . 25
extract_segment_trends . . . 26
flatten_DE . . . 26
flatten_enrich . . . 27
get_custom_palette . . . 28
group_specific_features . . . 28
impute_groups . . . 30
merge_groups . . . 31
merge_replicates . . . 32

normalise_to_start . . . 32
plot_breakpoints . . . 33
plot_cor_matrix . . . 34
plot_cv . . . 35
plot_DE_between_group . . . 36
plot_DE_between_time . . . 37
plot_distribution . . . 38
plot_GO . . . 39
plot_ID . . . 40
plot_missing . . . 41
plot_modules_h . . . 42
plot_modules_v . . . 45
plot_pca . . . 47
plot_pca_3D . . . 49
plot_pca_arrows . . . 50
plot_pca_by_group . . . 51
plot_scatter . . . 52
plot_segments . . . 54
plot_trend . . . 55
plot_umap . . . 57
plot_umap_by_group . . . 58
plot_upset_h . . . 59
plot_upset_v . . . 61
plot_variance . . . 64
plot_volcano . . . 65
plot_WGCNA . . . 67
prepare_tide . . . 67
prepare_WGCNA . . . 70
run_consensus_WGCNA . . . 71
run_Trendy . . . 73
run_WGCNA . . . 74
set_custom_palette . . . 75
split_groups . . . 76
summarise_feature_property . . . 76
summarise_module_metrics . . . 77
summarise_module_pattern . . . 78
summarise_Trendy . . . 79
theme_custom . . . 79
tutorial_data . . . 80
tutorial_sample_info . . . 81
WGCNA_module . . . 82

Index **83**

`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), Net_FC (fold change of latest-earliest time points; NA if only one time point is 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
```

combine_tide *Combine multiple* prepare_tide() *outputs*

Description

Combine a list of `prepare_tide()` output objects for multi-dataset or multi-omics analysis.

Usage

```
combine_tide(tide_list, assay = 1)
```

Arguments

tide_list	A non-empty named list of results from `prepare_tide()`.
assay	The assay to use: a numeric index or character name, e.g. 1 or "orig" for original data, 2 or "norm" for time 0 normalised data.

Value

A list containing the filtered and merged SE objects (all features and shared features), variance decomposition, shared feature names, the resolved assay name, and a per-dataset filtering summary.

Examples

```
data(tutorial_data)
data(tutorial_sample_info)
tide <- prepare_tide(tutorial_data, tutorial_sample_info,
    keep = "threshold", residual_threshold = 100)
combined <- combine_tide(list(RNA = tide, Protein = tide), assay = "norm")
combined$filter_summary
```

---

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")
```

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 created by create_input()
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. The selected assay should contain log-transformed, normalised values. 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(). Use TRUE only when the mean-variance relationship is expected to vary with abundance. The default FALSE is usually appropriate for log-transformed, normalised data unless diagnostics support modelling a mean-variance trend.

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. The selected assay should contain log-transformed, normalised values. 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(). Use TRUE only when the mean-variance relationship is expected to vary with abundance. The default FALSE is usually appropriate for log-transformed, normalised data unless diagnostics support modelling a mean-variance trend.

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

`DE_timecourse` *DE over time course*

Description

Differential expression analysis of temporal changes within and between groups by limma, using time as a factor or a numeric variable for linear or spline models.

Usage

```
DE_timecourse(
  se_obj,
  group = NULL,
  reference_group = NULL,
  assay,
  group_mode = c("per_group", "joint_group"),
  time_mode = c("factor", "numeric", "spline"),
  spline_df = 3,
  trend = FALSE,
  pAdjustMethod = "BH"
)
```

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 are used.
`reference_group`	A single reference group for a `joint_group` model. If NULL, the first group in `group` or `colData(se_obj)$Group` is used and reported by a message.
`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. The selected assay should contain log-transformed, normalised values. A warning is issued if the data appears to be un-logged raw counts.
`group_mode`	`"per_group"` or `"joint_group"`. • `"per_group"` tests each group separately. • `"joint_group"` fits one group-by-time model across the selected groups.
`time_mode`	`"factor"`, `"numeric"`, or `"spline"`. • `"factor"` treats time as a categorical variable. The earliest available time point is the reference, and each later time point is compared with it. All temporal terms are tested jointly with an omnibus F-test. With one temporal term, this is equivalent to a two-sided t-test.

- "numeric" treats time as continuous and tests one linear trend.
- "spline" models time with a natural cubic spline basis, allowing a smooth non-linear trajectory. The spline coefficients are tested jointly with an omnibus F-test.

spline_df — An integer of degrees of freedom when time_mode = "spline", controlling the flexibility of the fitted trajectory. Larger values allow more complex trajectories but may increase overfitting. It must be at least 2 and smaller than the number of time points used by the model. Default: 3.

trend — Logical, passed to limma::eBayes(). Use TRUE only when the mean-variance relationship is expected to vary with abundance. The default FALSE is usually appropriate for log-transformed, normalised data unless diagnostics support modelling a mean-variance trend.

pAdjustMethod — P-value adjustment method passed to stats::p.adjust() for the returned result tables. Must be one of stats::p.adjust.methods. Default: "BH".

Details

If colData(se_obj) contains Subject, repeated measurements are modelled with limma::duplicateCorrelation().

Missingness patterns are defined by whether each Group-Time combination contains at least one finite value.

The joint model is fit on the shared time points and shared features of all selected groups. Excluded or non-estimable features are retained in the results with NA.

Multiple testing correction is applied within each group for per_group results and separately within each joint_group result table, after combining all missingness patterns.

ReferenceTime is the earliest time point available in each fitting pattern and is used for group_effect in all time modes and for temporal comparisons in factor mode. It is NA for numeric and spline temporal tests.

In a joint model,

- reference_time_effect tests whether the reference group changes over time
- group_time_interaction tests whether the other groups have different temporal patterns
- global_time_effect tests all time and group-by-time terms jointly, asking whether any selected group has a temporal effect
- group_effect tests selected group differences at the earliest (reference) time point used by each missingness pattern
- global_group_effect tests all group and group-by-time terms jointly, asking whether group trajectories differ anywhere in the fitted time course.

Value

A list with results, fits, design, and metadata.

In a per_group model, results contains a table named time_effect.

In a joint_group model, results contains tables named reference_time_effect, global_time_effect, group_effect, global_group_effect, and group_time_interaction.

Result tables report an omnibus F statistic, P.Value, and adj.P.Val. For a table testing a single model term, F = t^2.

Columns in results tables have effect-specific names. For time effect tables,

- time_mode = "factor" uses logFC_TimeFactor<time> (log-scale changes relative to the reference time)
- time_mode = "numeric" uses logFC_TimeSlope (log-scale changes per time unit)
- time_mode = "spline" uses SplineBasis_1, SplineBasis_2, etc.

In joint_group results, group-by-time differences use names such as

- time_mode = "factor": delta_logFC_GroupX_TimeFactor<time>
- time_mode = "numeric": delta_logFC_GroupX_TimeSlope
- time_mode = "spline": delta_SplineBasis_GroupX_1, etc.

The global_group_effect table reports group main-effect model coefficients as GroupX (X is a non-reference group), alongside group-by-time terms; group_effect table reports each non-reference group's estimated difference from the reference group at the reference time (ReferenceTime) by columns of logFC_GroupX_at_reference_time.

These names apply only to results table columns; fits and design retain the original model coefficient names.

Features with missing values are fitted using their available observations when the model remains estimable. Features with insufficient observations or non-estimable designs return in the results with NA for model coefficients, test statistics, and p-values.

Examples

```
data(example_obj)

per_group_result <- DE_timecourse(
    example_obj, assay = 2, group_mode = "per_group"
)
joint_result <- DE_timecourse(
    example_obj, assay = 2,
    group = c("IFNbeta", "IFNgamma"),
    group_mode = "joint_group", reference_group = "IFNgamma"
)
utils::head(joint_result$results$global_time_effect)
```

---

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

`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)
}
```

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",
  simplify = FALSE,
  simplify_cutoff = 0.7,
  simplify_by = "p.adjust",
  simplify_select_fun = min,
  simplify_measure = "Wang",
  ...
)
```

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")
simplify	Whether to simplify the GO terms by removing redundant terms with `clusterProfiler::simplify()` (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 the representative term (default is "p.adjust")
simplify_select_fun	Parameter of `clusterProfiler::simplify()`, function to select the representative term (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::gseGO()`

Value

A list containing `all`, a list of `gseaResult` objects with the GSEA results. If `simplify = TRUE`, the output also contains `simplified`, a list of simplified `gseaResult` objects.

Examples

```
if (requireNamespace("org.Mm.eg.db", quietly = TRUE)) {
    library(org.Mm.eg.db)
    data(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), Count, Combined.Score, Genes, and any additional columns returned by the enrichR API. Compatible with plot_modules_h(enrich_list = result, enrich_category = "DSigDB"). Count is parsed from Overlap column, counted from Genes column when Overlap is absent or invalid, or NA if both are unavailable.

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)
```

---

example_cons run_consensus_WGCNA() *output object for runnable examples*

Description

Consensus WGCNA of Exp1 (example_obj, real) and Exp2 (example_obj_2, simulated), run on the 95 features shared by both datasets.

Usage

```
data(example_cons)
```

Format

A list with elements: `network`, `multiExpr`, `common_features`, `names` and `parameters`

Details

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

Source

GSE263759

---

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_net_2 — run_WGCNA() *output object for the simulated second dataset (Exp2)*

Description

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

Usage

```
data(example_net_2)
```

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_obj_2 *SummarizedExperiment object for the simulated second dataset (Exp2)*

Description

Exp2 is a noise-perturbed copy of `example_obj` (Exp1): each gene's expression receives independent Gaussian noise scaled to 0.5x its standard deviation. The first 5 feature names have the suffix `_Exp2` to introduce difference in features; the remaining 95 features are shared with Exp1.

Usage

```
data(example_obj_2)
```

Format

A SummarizedExperiment object matching the structure of `example_obj` (100 rows, 40 columns)

Details

Code for producing the data is in `inst/script/generate_example_multi.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_consensus_hubs`
Extract hub features from consensus WGCNA network

Description

Identifies hub features within each consensus module using module membership (kME). Consensus hubs are ranked using `WGCNA::consensusKME()` for signed and signed-hybrid networks, and absolute kME for unsigned networks.

Usage

```
extract_consensus_hubs(consensus, top_n = 1, exclude_grey = TRUE)
```

Arguments

`consensus`	An output object from `run_consensus_WGCNA()`.
`top_n`	Number of hub features to return per module, for each dataset and consensus. Default: 1.
`exclude_grey`	Logical; if `TRUE`, exclude the grey/unassigned module. Default: `TRUE`.

Value

A data frame with columns `Feature`, `Module`, `Set`, `Rank`, `Score`, and `ScoreType`. `Set` identifies the consensus hub or a dataset-specific hub set, for example `"Consensus hub"` or `"RNA hub"`. Duplicate features across sets are allowed.

Requirements

The `consensus` object must be the output of `run_consensus_WGCNA()`, which augments the raw `WGCNA::blockwiseConsensusModules()` output with `multiExpr` and `parameters`.

Examples

```
data(example_cons)
hubs <- extract_consensus_hubs(example_cons, top_n = 1)
utils::head(hubs)
```

`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,
  shared_only = FALSE,
  names = NULL,
  plot_cophenetic = FALSE,
  ...
)
```

Arguments

`se_obj`	A SummarizedExperiment object created by `create_input()`, or a named list of SummarizedExperiment objects (one per dataset).
`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)
shared_only	Logical, whether to subset each dataset to the features shared by all datasets before computing correlations (default is FALSE; only applies when se_obj is a list of multiple datasets)
names	Optional character vector of dataset names for the per-heatmap titles. Default: NULL, i.e. the names of se_obj (when a list). Only applies when se_obj is a list of multiple datasets.
plot_cophenetic	Logical, whether to include the pairwise Pearson correlations of cophenetic distances in the combined plot title (default is FALSE) in addition to reporting by text.
...	Additional arguments to be passed to ComplexHeatmap::Heatmap()

Details

With a named list of datasets that share the same samples, the per-dataset dendrograms are compared pairwise using Pearson correlations of their cophenetic distances (see ?stats::cophenetic).

Value

A heatmap showing the correlation between samples. With a named list of datasets, all heatmaps share the same colour scale, including pairwise correlations of cophenetic distances in the combined title when plot_cophenetic is TRUE, and are returned invisibly as a named list of Heatmap objects.

Examples

```
data(example_obj)
data(example_obj_2)
plot_cor_matrix(example_obj)
plot_cor_matrix(list(Exp1 = example_obj, Exp2 = example_obj_2))
```

---

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)

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 = NULL,
  profile_width = 3,
  profile_link_width = 1,
  enrich_list = NULL,
  enrich_category = NULL,
  enrich_rank_by = "p.adjust",
  enrich_top_n = 3,
  enrich_threshold = 0.05,
  enrich_wrap_width = NULL,
  fontsize = 8,
  group_annotation = c("name", "colour", "both", "none"),
  return_legends = FALSE,
  profile_scale = c("dataset", "shared", "free"),
  heatmap_width = 8,
  heatmap_height = 8,
  mark_features = NULL,
  width = 16,
  height = 12,
  res = 300,
  suffix = "",
```

```
  device = "png",
  save = NULL,
  names = NULL,
  selected_modules = NULL,
  legend_position = c("right", "bottom"),
  feature_colours = 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), or a named list of such objects (one per dataset). The colData of each 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. A single character string shared by all datasets. Default: NULL, i.e. "Z-score of log2 abundance" when scale = TRUE, "Log2 abundance" otherwise.
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", "Features")`). When `"Features"` is included, `mark_features` are shown as a column in the textbox. Categories must be present in `enrich_list`, except for `"Features"`.
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
: Threshold for colour-coding enrichment terms (default: 0.05). Terms where enrich_rank_by column value falls below this threshold are drawn in black, terms above in grey. Supply a vector matching enrich_category for per-category control, or set to NULL to skip colour-coding (all terms black). Set to NA for "Features".

enrich_wrap_width
: Optional maximum enrichment term width in cm. The default NULL disables wrapping.

fontsize
: Base font size (default: 8)

group_annotation
: Top annotation for group-split heatmap columns: "name" (default), "colour", "both", or "none". In "both" mode, group names are above the colour strip.

return_legends
: Logical; if TRUE, return a list containing the Heatmap/HeatmapList and the native ComplexHeatmap legend objects. (default: FALSE)

profile_scale
: Y-axis scaling for the mean expression profiles: "dataset" (default) uses one range across modules within each dataset, "shared" uses one range across all modules and datasets, and "free" uses an independent range for each module.

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 three forms:
  - A **character vector**: all marked in black by default on the left.
  - A **named list** of character vectors, e.g. list("Set 1" = c("gene1"), "Set 2" = c("gene2")). Names become categories with auto-assigned colours (via ggsci::pal_jco()).
  - A data frame with Feature and Set columns, such as the output of extract_consensus_hubs(). Set values become categories. Auto-routed: shown as anno_mark (left side, connecting lines) unless "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 and svg formats) (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", "svg", "tiff", "jpeg" (default: "png")

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

names
: Optional character vector of dataset names for the plot subtitles when multi-dataset. Default: NULL, i.e. the names of se_obj_merged.

selected_modules
: Optional character or numeric module labels for which profiles and enrichment panels should be displayed. The default NULL keeps all modules.

legend_position
: Legend position: "right" (default) or "bottom".

feature_colours
: Optional named character vector of colours for marked feature sets. Names must match feature set names in mark_features. If mark_features is a plain character vector, this parameter is ignored. Omitted sets retain their automatic colours. Default: NULL.

Value

A combined plot of heatmap, mean expression profile, and enrichment terms for each WGCNA module. If return_legends = TRUE, returns a list with elements heatmap and legends instead.

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_h(example_module |> dplyr::filter(Module != '0'),
    list(Exp1 = example_obj_merged, Exp2 = example_obj_merged),
    assay = 2, scale = TRUE,
    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 = NULL,
  suffix = "",
  device = "png",
  save = NULL,
  width = 12,
  height = 8,
  height_ratio = 2,
  fontsize = 8,
  res = 300,
  names = 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), or a named list of such objects (one per dataset). 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. A single character string shared by all datasets, or a character vector with one entry per dataset (multi-input only). Default: NULL, i.e. "Z-score of log2 abundance" when scale = TRUE, "Log2 abundance" otherwise.
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))
names	Optional character vector of dataset names for the plot subtitles when multi-dataset. Default: NULL, i.e. the names of se_obj_merged.

Value

For a single SummarizedExperiment, 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. For a named list of datasets, the per-dataset plot pairs are stacked vertically, with each dataset's line plot annotated with its name as a subtitle and an overall title "Module profiles: ...".

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_modules_v(example_module |> dplyr::filter(Module != '0'),
    list(Exp1 = example_obj_merged, Exp2 = example_obj_merged),
    assay = 2, scale = TRUE, 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,
  shared_only = FALSE,
  names = NULL
)
```

Arguments

se_obj	A SummarizedExperiment object created by `create_input()`, or a named list of SummarizedExperiment objects (one per dataset)
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.)
shared_only	Logical, whether to subset each dataset to the features shared by all datasets before PCA (default is FALSE; only applies when se_obj is a list of multiple datasets)
names	Optional character vector of dataset names for the plot titles. Default: NULL, i.e. the names of se_obj (when a list).

Value

A list with p_list (a named list of ggplot objects) and pca (a PCAtools output object for custom plotting). When se_obj is a named list of datasets, p_list and pca are named lists (one per dataset) and the combined PCA scatter plots are returned as plot.

Examples

```
data(example_obj)
data(example_obj_2)
PC = plot_pca(example_obj, morepc = seq(1, 3))
PC$p_list
PC_multi = plot_pca(list(Exp1 = example_obj, Exp2 = example_obj_2))
PC_multi$plot
```

---

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_scatter *Feature property scatter plot between two datasets*

Description

Compares two per-feature numeric properties. Features present in both input tables are matched by `Feature` and plotted using the selected numeric column from each table on the x- and y-axes. When `df2` is NULL, two columns from `df1` are compared.

Usage

```
plot_scatter(
  df1,
  df2 = NULL,
  col1,
  col2,
  name1 = NULL,
  name2 = NULL,
  highlight_features = NULL,
  label_top_n = 0,
  label_threshold = NULL,
  show_cor = TRUE,
  cor_method = c("spearman", "pearson", "kendall"),
  colour_threshold = NULL,
  fontsize = 8,
  show_diagonal = NULL,
  ...
)
```

Arguments

`df1`	Data frame with columns `Feature` and a numeric value column (e.g., logFC, variance proportion).
`df2`	Optional data frame for the second property. When NULL, `df1` is used for both properties.
`col1`	Name of the numeric column in `df1` to plot on the x-axis (e.g., `"logFC"`).
`col2`	Name of the numeric column in `df2` to plot on the y-axis.
`name1`	Label for dataset 1 (x-axis). Default: `"Dataset 1: <col1>"`.

name2 Label for dataset 2 (y-axis). Default: "Dataset 2: <col2>".

highlight_features
Optional: character vector of feature names to label and mark in black. Features not shared by both datasets are reported and ignored.

label_top_n Number of top features to label per quadrant. Can be:

- A single integer: same number of labels for all four quadrants.
- A vector of 4 integers: labels for quadrants (++, +-, -+, –) respectively. Use c(5, 3, 3, 5) or named c("++"=5, "+-"=3, "-+"=3, "--"=5). Within each quadrant, features are ranked by distance from origin (sqrt(x^2 + y^2)). These features are labelled along with threshold-qualified and highlighted features. Default: 0 (no additional labels).

label_threshold
Minimum absolute value for labelling. A single value applies to both axes, or use a named numeric vector, e.g., c(x = 1, y = 0.5). Both the absolute x and absolute y values must meet their respective thresholds. Qualifying features are labelled along with the top-N and highlighted features. Default: NULL.

show_cor If TRUE (default), annotate the plot with the correlation coefficient and p-value between the two properties.

cor_method Correlation method: "spearman" (default), "pearson", or "kendall".

colour_threshold
Minimum absolute value for point colouring. A single value applies to both axes, or use a named numeric vector, e.g., c(x = 1, y = 0.5). Points are coloured only when both absolute x and y values meet their respective thresholds; points below either threshold are shown as grey. Default: NULL (colour all points).

fontsize Base font size (default: 8).

show_diagonal If NULL (default), show the dashed y = x line when comparing the same property across two data frames. Set TRUE or FALSE to override this behaviour.

... Additional arguments passed to ggrepel::geom_text_repel() for controlling label appearance.

Value

A ggplot object of scatter plot.

Examples

```
# Simulated per-feature numeric properties
df_1 <- data.frame(
    Feature = paste0("Gene", 1:100),
    Property_1 = rnorm(100, 0, 2),
    Property_2 = rnorm(100, 0, 1),
    stringsAsFactors = FALSE
)
df_2 <- data.frame(
    Feature = paste0("Gene", 1:100),
    Property_1 = rnorm(100, 0, 2),
    stringsAsFactors = FALSE
```

```
)
plot_scatter(df_1, df_2,
    col1 = "Property_1", col2 = "Property_1",
    colour_threshold = 1, label_top_n = 5,
    highlight_features = c("Gene1", "Gene2", "Gene3")
)
# Per-quadrant control: 5 labels in concordant, 2 in discordant
plot_scatter(df_1, df_2,
    col1 = "Property_1", col2 = "Property_1",
    colour_threshold = 1,
    label_top_n = c("++" = 5, "+-" = 2, "-+" = 2, "--" = 5)
)
# Compare two properties from one data frame
plot_scatter(df_1, col1 = "Property_1", col2 = "Property_2",
    colour_threshold = 1, label_top_n = 5)
```

`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()`), a pre-computed table from `calc_mean_sd()`, or a named list of SummarizedExperiment objects (multiple datasets, e.g. multiomics).

Usage

```
plot_trend(
  se_obj,
  assay = 1,
  groups = NULL,
  features,
  title = NULL,
  ylabel = "Log2 abundance",
  errorbar = TRUE,
  fontsize = 8,
  multi_layout = c("wrap", "vertical", "horizontal", "overlay"),
  ncol = NULL,
  independent_y = TRUE
)
```

Arguments

se_obj	A SummarizedExperiment object, a data.frame from `calc_mean_sd()` with columns: Feature, Time, Group, Mean, SD, or a named list of SummarizedExperiment objects (multiple datasets, e.g. multiomics).
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) for single input, "Trends across datasets" if NULL for a named list of SummarizedExperiment objects.
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)
multi_layout	Facet layout for multiple-datasets input: `"wrap"` (default): wraps features, datasets nested per feature; `"vertical"`: datasets as rows, features as columns; `"horizontal"`: features as rows, datasets as columns; `"overlay"`: one panel per feature, datasets by linetype. Ignored for single input.
ncol	Number of feature columns per row for `"wrap"` and `"overlay"` layouts (multiple datasets) or in `facet_wrap()` (single input). If NULL (default), autocomputed.
independent_y	If TRUE (default), each panel gets its own y-axis scale. If FALSE, y-axes are shared: within each facet row for "vertical" and "horizontal" layouts, fully for "wrap" and "overlay".

Value

Plot of feature abundances over time by mean and SD

Examples

```
data(example_obj)
data(example_obj_2)
plot_trend(example_obj,
    features = sample(rownames(example_obj), 4))

# Also accepts a named list of SummarizedExperiment objects
plot_trend(list(Exp1 = example_obj, Exp2 = example_obj_2),
    features = sample(rownames(example_obj), 2))
```

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,
  shared_only = FALSE,
  names = NULL
)
```

Arguments

se_obj	A SummarizedExperiment object created by `create_input()`, or a named list of SummarizedExperiment objects (one per dataset)
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.)
shared_only	Logical, whether to subset each dataset to the features shared by all datasets before UMAP (default is FALSE; only applies when se_obj is a list of multiple datasets)
names	Optional character vector of dataset names for the plot titles. Default: NULL, i.e. the names of se_obj (when a list).

Value

A list with p_list (a named list of ggplot objects) and umap_layout (a data frame containing UMAP coordinates and sample annotations for custom plotting). When se_obj is a named list of datasets, both are named lists (one per dataset) and the combined UMAP plots are returned as plot.

Examples

```
data(example_obj)
data(example_obj_2)
umap_out <- plot_umap(example_obj)
umap_out$p_list
umap_multi <- plot_umap(list(Exp1 = example_obj, Exp2 = example_obj_2))
umap_multi$plot
```

---

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_upset_h *UpSet plot of feature sets with horizontally aligned enrichment terms*

Description

Combines an UpSet plot with row annotations for intersection size and optional textboxes of top enriched functional terms.

Usage

```
plot_upset_h(
  set_list,
  enrich_list = NULL,
  enrich_category = NULL,
  enrich_top_n = 3,
  enrich_rank_by = "p.adjust",
  enrich_threshold = 0.05,
  enrich_wrap_width = NULL,
  set_order = NULL,
  intersection_order = NULL,
  intersection_labels = NULL,
```

```
  intersection_colours = NULL,
  fontsize = 8,
  ...
)
```

Arguments

set_list	A named list of character vectors of feature sets (e.g., list(Exp1_Up = ..., Exp1_Down = ..., Exp2_Up = ..., Exp2_Down = ...)). Empty input sets and zero-size intersections are excluded from the plot.
enrich_list	A named enrichment list for intersections, such as the output of enrichGO_list()$all or enrichR_list(). Each named element should be a data.frame with Cluster, Description, and the column named by enrich_rank_by. If NULL (default), no enrichment terms are displayed. The Cluster column may contain binary combination names or displayed UpSet labels. Displayed labels are auto-generated from names(set_list) by joining participating set names with _, and may be overridden with intersection_labels. See plot_modules_h() for the same per-category enrichment parameter convention. When enrich_category is a vector, each category is shown as a separate column in the annotation textbox.
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")). Categories must be present in enrich_list.
enrich_top_n	Top N enrichment terms per intersection. 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_rank_by	Numeric column used to rank enrichment terms within each intersection. 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_threshold	Threshold for colour-coding enrichment terms (default: 0.05). Terms where enrich_rank_by column value falls below this threshold are drawn in black, terms above in grey. Supply a vector matching enrich_category for per-category control, or set to NULL to skip colour-coding (all terms black).
enrich_wrap_width	Optional maximum enrichment term width in cm. The default NULL disables wrapping.
set_order	Optional order of the sets. When NULL (default), sets are ordered by decreasing size. Pass a character vector of set names to control display order.
intersection_order	Optional order of intersection rows. When NULL (default), intersections are ordered by decreasing combination size. Pass a character vector of combination names (as returned by ComplexHeatmap::comb_name()), or a numeric permutation of their indices, to specify a custom order.

`intersection_labels`
: Optional named character vector mapping `comb_name` binary patterns to custom intersection names, overriding the auto-generated defaults. E.g., `c("1010" = "Concordant_Up", "0101" = "Concordant_Down")`. Entries for patterns not present in the combination matrix cause an error.

`intersection_colours`
: Optional named colour vector for intersections. Names must match the binary combination patterns used by `intersection_labels` (e.g. `"1010"`). Colours are used for the intersection bars and combination marks.

`fontsize`
: Base font size (default: 8).

`...`
: Additional arguments passed to `ComplexHeatmap::UpSet()`.

Details

The binary combination pattern and displayed intersection label are printed when the function is called. These mappings can be used as keys for `intersection_labels` and `intersection_colours`.

Value

A ComplexHeatmap UpSet plot, with optional enrichment term annotations.

Examples

```
# Mock feature sets with reproducible intersections
genes <- paste0("G", 1:200)
set_list <- list(Exp1_Up = genes[1:60], Exp1_Down = genes[61:100],
    Exp2_Up = genes[c(1:40, 101:115)], Exp2_Down = genes[c(61:80, 116:130)])
enrich_list <- list(BP = data.frame(
    Cluster = c("Exp1_Up_Exp2_Up", "Exp1_Down_Exp2_Down"),
    Description = c("term A", "term B"), p.adjust = c(0.001, 0.04)))
intersection_colours <- c("1010" = "#0072B5", "0101" = "#D55E00")
plot_upset_h(set_list, enrich_list = enrich_list,
    enrich_category = "BP", intersection_colours = intersection_colours)
# Custom intersection order uses the binary combination patterns.
plot_upset_h(set_list,
    intersection_order = c("1010", "1000", "0101", "0100", "0010", "0001"))
```

---

plot_upset_v *UpSet plot of feature sets with vertically aligned enrichment dot matrix*

Description

Draws a ComplexHeatmap UpSet plot with intersections as columns and a clusterProfiler-style enrichment dot matrix underneath. The two panels use the same intersection order, so each enrichment column is vertically aligned with its UpSet intersection.

Usage

```
plot_upset_v(
  set_list,
  enrich_list = NULL,
  enrich_category = NULL,
  enrich_top_n = 3,
  include_all = TRUE,
  enrich_rank_by = "p.adjust",
  set_order = NULL,
  intersection_order = NULL,
  intersection_labels = NULL,
  intersection_colours = NULL,
  fontsize = 8,
  dot_size_col = "Count",
  dot_colour_col = "p.adjust",
  dot_min_size = 1,
  dot_max_size = 3,
  dot_outline = "black",
  dot_outline_width = 0.5,
  term_wrap_width = 30,
  row_gap_mm = 2,
  show_dot_column_names = FALSE,
  enrich_term_order = "rank",
  enrich_term_colours = NULL,
  ...
)
```

Arguments

set_list	A named list of character vectors of feature sets.
enrich_list	A named enrichment list for intersections, such as the output of enrichGO_list()$all or enrichR_list(). Each named element should be a data.frame with Cluster, Description, and the columns named by enrich_rank_by, dot_size_col, and dot_colour_col. If NULL (default), no enrichment terms are displayed. The Cluster column may contain binary combination names or displayed UpSet labels. Displayed labels are auto-generated from names(set_list) by joining participating set names with _, and may be overridden with intersection_labels. When enrich_category is a vector, each category is shown as a separate row section in the enrichment dot matrix.
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")). Categories must be present in enrich_list.
enrich_top_n	Top N enrichment terms per intersection. A single integer (default: 3) is recycled for all categories in enrich_category. Supply an integer vector of matching length for per-category control.
include_all	Whether to include every occurrence of a term selected in any intersection, matching clusterProfiler's includeAll. (default: TRUE).

enrich_rank_by Numeric column used to rank enrichment terms within each intersection. 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.

set_order Optional order of the sets. When NULL (default), sets are ordered by decreasing size. Pass a character vector of set names to control display order.

intersection_order
Optional order of intersection columns. When NULL (default), intersections are ordered by decreasing combination size. Pass a character vector of combination names (as returned by ComplexHeatmap::comb_name()), or a numeric permutation of their indices, to specify a custom order.

intersection_labels
Optional named character vector mapping comb_name binary patterns to custom intersection names, overriding the auto-generated defaults. E.g., c("1010" = "Concordant_Up", "0101" = "Concordant_Down"). Entries for patterns not present in the combination matrix cause an error.

intersection_colours
Optional named colour vector for intersections. Names must match the binary combination patterns used by intersection_labels (e.g. "1010"). Colours are used for the intersection bars and combination marks.

fontsize Base font size. Default is 8.

dot_size_col Numeric column used for dot size in the enrichment annotation. Default is "Count".

dot_colour_col Numeric column used for dot colour in the enrichment annotation. Default is "p.adjust".

dot_min_size Minimum dot radius in mm. Default is 1.

dot_max_size Maximum dot radius in mm. Default is 3.

dot_outline Outline colour for enrichment dots. Use NA to remove outlines. Default is "black".

dot_outline_width
Outline width in points. Default is 0.5.

term_wrap_width
Maximum characters per displayed enrichment term before wrapping to a new line. Default is 30.

row_gap_mm Gap between enrichment categories in mm. Default is 2.

show_dot_column_names
Whether to show intersection labels on the enrichment dot matrix. Default: FALSE.

enrich_term_order
Controls how to order the enrichment term rows. "rank" (default) orders terms by their best enrich_rank_by value. "intersection" groups terms by intersection_order, with terms within each intersection ordered by enrich_rank_by.

enrich_term_colours
Optional named colour vector for enrichment terms. Names are matched to enrichment ID values when available, and then to Description values. Unspecified terms remain black.

... Additional arguments passed to ComplexHeatmap::UpSet().

Details

The binary combination pattern and displayed intersection label are printed when the function is called. These mappings can be used as keys for `intersection_labels` and `intersection_colours`.

Value

A ComplexHeatmap UpSet plot, with optional enrichment dot annotations.

References

Dot matrix mimicking `clusterProfiler::dotplot()`

Examples

```
# Mock feature sets with reproducible intersections
genes <- paste0("G", 1:200)
set_list <- list(Exp1_Up = genes[1:60], Exp1_Down = genes[61:100],
    Exp2_Up = genes[c(1:40, 101:115)], Exp2_Down = genes[c(61:80, 116:130)])
enrich_list <- list(BP = data.frame(
    Cluster = c("Exp1_Up_Exp2_Up", "Exp1_Down_Exp2_Down"),
    Description = c("term A", "term B"), p.adjust = c(0.001, 0.04),
    Count = c(10, 5)))
intersection_colours <- c("1010" = "#0072B5", "0101" = "#D55E00")
plot_upset_v(set_list, enrich_list = enrich_list,
    enrich_category = "BP", intersection_colours = intersection_colours)
# Custom intersection order uses the binary combination patterns.
plot_upset_v(set_list,
    intersection_order = c("1010", "1000", "0101", "0100", "0010", "0001"))
```

---

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)

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_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("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(): a numeric value, features with values > feature_threshold are considered expressed; or NULL, any non-NA value is 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 > fea-

ture_threshold, or non-NA when feature_threshold is NULL, 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: `"quantile"` (default): keep features with residual variance at or below the `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 = "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 "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. , 12. , 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, WGCNA's `powerEstimate`, a custom `custom_powerEstimate` based on the signed scale-free topology fit, network type, and prepared input data used for reuse in `run_WGCNA()`.

The custom estimate selects the lowest power whose signed scale-free topology fit exceeds `RsquaredCut`, consistent with the signed-R-squared panel in the diagnostic plot.

References

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

Examples

```
data(example_obj)

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

`run_consensus_WGCNA` *Consensus WGCNA across multiple datasets*

Description

Run blockwise consensus WGCNA on more than one dataset sharing a common feature ID space.

Usage

```
run_consensus_WGCNA(
  nets,
  powers,
  consensusQuantile = 0,
  numericLabels = TRUE,
  plot = TRUE,
  ...
)
```

Arguments

nets	A named list of WGCNA input objects (output from prepare_WGCNA).
powers	A named numeric vector of soft-thresholding powers to be used in WGCNA::blockwiseConsensusModules Values may be selected based on the output of prepare_WGCNA(). Names must match names of nets.
consensusQuantile	Quantile at which consensus is to be defined. (default: 0). See WGCNA::blockwiseConsensusModules() for details.
numericLabels	Whether to use numeric labels for modules in the output (default is TRUE).
plot	Logical. If TRUE (default), plot consensus dendrogram with module colours for each feature block via WGCNA::plotDendroAndColors().
...	Additional parameters to be passed to WGCNA::blockwiseConsensusModules. The network type is inherited from nets, and a networkType passed via ... is accepted only if it matches the value stored in nets.

Value

A list with elements: network (the consensus result), multiExpr (the input multi-expression list), common_features (features shared across all datasets), names (dataset names), and parameters.

Examples

```
data(example_obj)
data(example_obj_2)
shared_features <- intersect(
    rownames(example_obj), rownames(example_obj_2)
)
inp1 <- prepare_WGCNA(example_obj[shared_features, , drop = FALSE],
    powers = seq(1, 20), assay = 2, networkType = "signed")
inp2 <- prepare_WGCNA(example_obj_2[shared_features, , drop = FALSE],
    powers = seq(1, 20), assay = 2, networkType = "signed")
cons <- run_consensus_WGCNA(
    nets = list(Exp1 = inp1, Exp2 = inp2),
    powers = c(Exp1 = 8, Exp2 = 8),
    minModuleSize = 10
)
```

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)

wgcna_input <- prepare_WGCNA(example_obj, assay = 2, powers = seq(1, 30),
    networkType = "signed", RsquaredCut = 0.8)
wgcna_input$plot
picked_power <- wgcna_input$custom_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_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 and net 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 and net 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. For consensus WGCNA output, metrics are returned for the consensus and for each individual dataset.

Usage

```
summarise_module_metrics(net)
```

Arguments

net
: A WGCNA network object from run_WGCNA() or run_consensus_WGCNA().

Details

MeanKME is mean module membership (kME = cor(feature, module eigengene)). MeanKME2 is the mean squared module membership (mean(kME^2)), equivalent to the mean per-feature proportion of variance explained by the module eigengene. Q25KME and Q75KME are the 25th and 75th kME percentiles. For consensus output, it is calculated from consensus or individual-dataset kME scores, respectively.

Value

For an individual WGCNA network, a data frame with columns: `Module`, `Size`, `Proportion`, `MeanKME`, `MeanKME2`, `MedianKME`, `SDKME`, `Q25KME`, `Q75KME`, `MinKME`, and `MaxKME`. For consensus WGCNA, a named list of these data frames, with `Consensus` followed by one element for each individual dataset.

Examples

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

---

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",
  base_family = "sans"
)
```

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").
base_family	Base font family (default: "sans").

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

∗ **datasets**
  example_cons, 20
  example_go, 21
  example_net, 22
  example_net_2, 22
  example_obj, 23
  example_obj_2, 23
  example_res_list, 24
  tutorial_data, 80
  tutorial_sample_info, 81

calc_feature_property, 4
calc_mean_sd, 5
combine_tide, 5
create_input, 6

DE_between_group, 7
DE_between_group(), *26, 27*
DE_between_time, 9
DE_between_time(), *26, 27*
DE_timecourse, 10
decomp_variance, 12

enrich_msigdb, 14
enrich_msigdb(), *27*
enrichGO_list, 16
enrichGO_list(), *27*
enrichGO_rank, 17
enrichR_list, 19
enrichR_list(), *27*
example_cons, 20
example_go, 21
example_net, 22
example_net_2, 22
example_obj, 23
example_obj_2, 23
example_res_list, 24
extract_consensus_hubs, 24
extract_hubs, 25
extract_segment_trends, 26

flatten_DE, 26
flatten_enrich, 27

get_custom_palette, 28
group_specific_features, 28

impute_groups, 30

merge_groups, 31
merge_replicates, 32

normalise_to_start, 32

plot_breakpoints, 33
plot_cor_matrix, 34
plot_cv, 35
plot_DE_between_group, 36
plot_DE_between_time, 37
plot_distribution, 38
plot_GO, 39
plot_ID, 40
plot_missing, 41
plot_modules_h, 42
plot_modules_v, 45
plot_pca, 47
plot_pca_3D, 49
plot_pca_arrows, 50
plot_pca_by_group, 51
plot_scatter, 52
plot_segments, 54
plot_trend, 55
plot_umap, 57
plot_umap_by_group, 58
plot_upset_h, 59
plot_upset_v, 61
plot_variance, 64
plot_volcano, 65
plot_WGCNA, 67
prepare_tide, 67
prepare_WGCNA, 70

run_consensus_WGCNA, 71
run_Trendy, 73
run_WGCNA, 74

set_custom_palette, 75
split_groups, 76
summarise_feature_property, 76
summarise_module_metrics, 77
summarise_module_pattern, 78
summarise_Trendy, 79

theme_custom, 79
tutorial_data, 80
tutorial_sample_info, 81

WGCNA_module, 82