

Package: VISTA (via r-universe)

September 17, 2026

Type Package

Title Visualization and Integrated System for Transcriptomic Analysis

Version 1.0.0

Description The VISTA (Visualization and Integrated System for Transcriptomic Analysis) platform streamlines differential expression workflows by wrapping DESeq2 and edgeR into a SummarizedExperiment-based container with consistent metadata. The package includes visualization utilities, MSigDB enrichment helpers, and optional deconvolution support to simplify interactive exploration of RNA-seq experiments.

License GPL-3

Encoding UTF-8

Depends R (>= 4.3)

Imports AnnotationDbi, cli, clusterProfiler, colorspace, DESeq2, dplyr, edgeR, forcats, ggplot2, ggrepel, GGally, ggpubr, grid, matrixStats, methods, msigdb, limma, purrr, rlang, S4Vectors, scales, stringr, SummarizedExperiment, tibble, tidyr, tidyselect, viridis

Suggests airway, BiocStyle, circlize, ComplexHeatmap, DT, EnhancedVolcano, ggpointdensity, ggridges, ggalluvial, ggcorrplot, ggrain, ggvenn, enrichplot, knitr, magrittr, patchwork, org.Hs.eg.db, org.Mm.eg.db, quarto, rmarkdown, yaml, writexl, testthat (>= 3.0.0), uwot, xCell2

URL <https://github.com/cparsania/VISTA>,
<https://cparsania.github.io/VISTA/>

BugReports <https://github.com/cparsania/VISTA/issues>

biocViews RNASeq, DifferentialExpression, GeneExpression, Transcriptomics, Visualization

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

LazyData false

Config/testthat/edition 3

Config/pak/sysreqs libcairo2-dev cmake libfontconfig1-dev libfreetype6-dev libglpk-dev make libicu-dev libpng-dev libuv1-dev libxml2-dev libssl-dev zlib1g-dev

Repository Bioconductor 3.23

Date/Publication 2026-04-28 13:07:01 UTC

RemoteUrl <https://github.com/bioc/VISTA>

RemoteRef RELEASE_3_23

RemoteSha d7e91e75f34dc308af5d1536a452fad9707ad2fa

Contents

as_vista	4
benchmark_vista_equivalence	5
count_data	7
create_vista	8
derive_vista_metadata	11
enrichMsigDB	12
example_vista	14
export_vista_assets	14
get_cell_fractions	16
get_celltype_barplot	17
get_celltype_group_dotplot	18
get_celltype_heatmap	19
get_corr_heatmap	21
get_deg_alluvial	23
get_deg_count_barplot	24
get_deg_count_donutplot	25
get_deg_count_pieplot	26
get_deg_venn_diagram	27
get_enrichment_chord	28
get_enrichment_plot	31
get_expression_barplot	32
get_expression_boxplot	34
get_expression_chromosome_plot	35
get_expression_density	38
get_expression_heatmap	40
get_expression_joyplot	42
get_expression_lineplot	44
get_expression_lollipop	46
get_expression_matrix	48
get_expression_raincloud	49
get_expression_scatter	51
get_expression_violinplot	52
get_foldchange_barplot	54

get_foldchange_boxplot	56
get_foldchange_chromosome_plot	57
get_foldchange_heatmap	58
get_foldchange_lineplot	61
get_foldchange_lollipop	62
get_foldchange_matrix	64
get_foldchange_raincloud	65
get_foldchange_scatter	67
get_genes_by_regulation	69
get_go_enrichment	70
get_gsea	71
get_kegg_enrichment	72
get_ma_plot	73
get_mds_plot	75
get_msigdb_enrichment	76
get_pairwise_corr_plot	78
get_pathway_genes	79
get_pathway_heatmap	81
get_pca_plot	83
get_umap_plot	85
get_volcano_plot	87
match_vista_inputs	89
print.VISTA	90
read_vista_counts	91
read_vista_metadata	93
run_cell_deconvolution	94
run_deseq_analysis	95
run_vista_report	98
sample_metadata	100
save_vista_data	101
save_vista_plot	102
set_de_source	103
set_rowdata	103
set_vista_comparison_colors	105
set_vista_group_colors	106
validate_vista	107
validate_vista_deep	108
VISTA-accessors	110
VISTA-class	112

as_vista

*Coerce SummarizedExperiment to VISTA***Description**

Convert a pre-existing SummarizedExperiment into a valid VISTA object.

Usage

```
as_vista(
  se,
  assay_name = "norm_counts",
  group_column,
  comparisons = list(),
  deg_summary = list(),
  cutoffs = list(),
  group_palette = "Dark 3",
  validate = TRUE
)
```

Arguments

se	A SummarizedExperiment object.
assay_name	Assay in se to use as norm_counts (default: "norm_counts").
group_column	Grouping column in colData(se).
comparisons	Optional named list of differential expression tables.
deg_summary	Optional named list of DEG summary tables.
cutoffs	Optional named list of threshold metadata.
group_palette	Palette name for group colors.
validate	Logical; run object validation before returning.

Value

A VISTA object.

Examples

```
mat <- matrix(rnorm(60), nrow = 10)
rownames(mat) <- paste0("gene", seq_len(nrow(mat)))
colnames(mat) <- paste0("sample", seq_len(ncol(mat)))
se <- SummarizedExperiment::SummarizedExperiment(
  assays = list(norm_counts = mat),
  colData = S4Vectors::DataFrame(
    cond = rep(c("A", "B"), each = 3),
    row.names = colnames(mat)
  ),
)
```

```

rowData = S4Vectors::DataFrame(
  gene_id = rownames(mat),
  row.names = rownames(mat)
)
v <- as_vista(se, group_column = "cond")
v

```

benchmark_vista_equivalence

Benchmark VISTA against standalone differential-expression back-ends

Description

Run VISTA and direct DESeq2, edgeR, and limma pipelines with matched preprocessing, contrast definitions, and DEG thresholds, then compare the resulting tables, DEG calls, normalized matrices, and critical plot inputs.

Usage

```

benchmark_vista_equivalence(
  counts,
  sample_info,
  column_geneid,
  group_column,
  group_numerator,
  group_denominator,
  methods = c("deseq2", "edgeR", "limma"),
  min_counts = 10,
  min_replicates = 1,
  log2fc_cutoff = 1,
  pval_cutoff = 0.05,
  p_value_type = "padj",
  covariates = NULL,
  design_formula = NULL,
  tolerance = 1e-08,
  return_plots = FALSE
)

```

Arguments

counts	Raw counts (matrix/data.frame) with a gene-id column and sample columns.
sample_info	Data frame with sample metadata.
column_geneid	Column name in counts that contains gene identifiers.
group_column	Column in sample_info used to group samples.

group_numerator	Character vector of numerator groups for pairwise comparisons.
group_denominator	Character vector of denominator groups.
methods	Character vector of backends to benchmark. Any subset of c("deseq2", "edgeR", "limma").
min_counts	Minimum total counts per gene to retain.
min_replicates	Minimum samples per group meeting filtering criteria.
log2fc_cutoff	Absolute log2 fold-change threshold for DEG calling.
pval_cutoff	P-value (or adjusted p-value) threshold.
p_value_type	Either "p.adjust" or "pvalue".
covariates	Optional character vector of additional sample_info columns.
design_formula	Optional model formula (or formula string) including group_column.
tolerance	Numeric tolerance used for floating-point comparisons.
return_plots	Logical; if TRUE, return paired VISTA/reference plots for MA, volcano, DEG count, and PCA views.

Value

A list with fields valid, comparison_summary, visual_summary, and methods. Each element of methods contains the VISTA object, direct backend results, structural validation output, self-consistency checks, and optional plot objects.

Examples

```
v <- example_vista()
si <- as.data.frame(sample_info(v))
data("count_data", package = "VISTA")
count_subset <- count_data[seq_len(500), c("gene_id", si$sample_names), drop = FALSE]

bm <- benchmark_vista_equivalence(
  counts = count_subset,
  sample_info = si,
  column_geneid = "gene_id",
  group_column = "cond_long",
  group_numerator = "treatment1",
  group_denominator = "control",
  methods = "limma",
  min_counts = 5,
  min_replicates = 1
)

bm$comparison_summary

data("count_data", package = "VISTA")
data("sample_metadata", package = "VISTA")
```

```

target_groups <- c("control", "treatment1")
sample_subset <- sample_metadata[sample_metadata$cond_long %in% target_groups, ]
count_subset <- count_data[seq_len(150), c("gene_id", sample_subset$sample_names)]

bm <- benchmark_vista_equivalence(
  counts = count_subset,
  sample_info = sample_subset,
  column_geneid = "gene_id",
  group_column = "cond_long",
  group_numerator = "treatment1",
  group_denominator = "control",
  methods = c("deseq2", "edger"),
  min_counts = 5,
  min_replicates = 1
)

bm$comparison_summary
bm$visual_summary

```

count_data

*Example RNA-seq count matrix shipped with VISTA***Description**

A count matrix derived from the Bioconductor airway dataset and formatted for VISTA examples. The first column stores Ensembl gene IDs; the remaining columns are sample-level counts.

Usage

```
data(count_data)
```

Format

A tibble with 63,677 rows and 9 columns.

gene_id Character column storing Ensembl gene identifiers.

SRR1039508 Numeric count values for airway sample SRR1039508.

SRR1039509 Numeric count values for airway sample SRR1039509.

SRR1039512 Numeric count values for airway sample SRR1039512.

SRR1039513 Numeric count values for airway sample SRR1039513.

SRR1039516 Numeric count values for airway sample SRR1039516.

SRR1039517 Numeric count values for airway sample SRR1039517.

SRR1039520 Numeric count values for airway sample SRR1039520.

SRR1039521 Numeric count values for airway sample SRR1039521.

Source

Derived from the airway Bioconductor dataset.

create_vista

Create a VISTA Object with Internal DE Analysis

Description

This wrapper performs differential expression (DE) analysis (DESeq2, edgeR, limma, or both) and returns a fully initialized VISTA object. The object stores expression matrices and annotations in the SummarizedExperiment core, while all DE outputs and configuration live in `metadata(vista)`:

- `$de_results`: named SimpleList of per-contrast DE tables
- `$de_summary`: named SimpleList of summary tables
- `$de_cutoffs`: list of thresholds/method options
- `$group`: list with column, palette, colors

Usage

```
create_vista(
  counts,
  sample_info,
  column_geneid,
  group_column,
  group_numerator,
  group_denominator,
  method = c("deseq2", "edger", "limma", "both"),
  min_counts = 10,
  min_replicates = 1,
  log2fc_cutoff = 1,
  pval_cutoff = 0.05,
  p_value_type = "padj",
  covariates = NULL,
  design_formula = NULL,
  consensus_mode = c("intersection", "union"),
  consensus_log2fc = c("mean", "deseq2", "edger"),
  result_source = NULL,
  group_palette = "Dark 2",
  comparison_palette = "Dark 3",
  validate = TRUE
)
```

Arguments

<code>counts</code>	Raw counts (matrix/data.frame) with a gene-id column and sample columns.
<code>sample_info</code>	Data frame with sample metadata. Must contain <code>sample_names</code> (or have row-names equal to sample columns in <code>counts</code>) and the <code>group_column</code> .
<code>column_geneid</code>	Column name in <code>counts</code> that contains gene identifiers.

group_column	Column in sample_info used to group samples.
group_numerator	Character vector of numerator groups for pairwise comparisons.
group_denominator	Character vector of denominator groups (same length/order as numerator).
method	"deseq2", "edgeR", "limma", or "both".
min_counts	Minimum total counts per gene to retain (default: 10).
min_replicates	Minimum samples per group meeting min_counts (default: 1).
log2fc_cutoff	Absolute LFC threshold for DEG calling (default: 1).
pval_cutoff	p-value (raw or adjusted) threshold (default: 0.05).
p_value_type	Which p-value column to use ("padj" or "pvalue"). Default: "padj".
covariates	Optional character vector of additional sample_info columns to adjust for. These are included as additive terms in the DE design.
design_formula	Optional model formula (or formula string) overriding automatic construction from group_column + covariates. Must include group_column.
consensus_mode	When method = "both", how to define consensus calls: "intersection" (both methods significant in same direction) or "union" (either method significant; discordant directions excluded).
consensus_log2fc	When method = "both", how to populate consensus log2fc: "mean", "deseq2", or "edgeR".
result_source	Active DE source used in metadata(v)\$de_results. For method = "both", one of "consensus", "deseq2", "edgeR". For single-method runs, this must match method.
group_palette	Qualitative palette name for colorspace::qualitative_hcl(). One of c("Pastel 1", "Dark 2", "Dark 3", "Set 2", "Set 3", "Warm", "Cold", "Harmonic", "Dynamic"). Default: "Dark 2".
comparison_palette	Qualitative palette name used to assign colors per comparison (stored in metadata(v)\$comparison\$color). Defaults to "Dark 3".
validate	Logical; if TRUE (default), run full validate_vista() checks before returning the object.

Details

Contrast names follow "numerator_VS_denominator". Each DE table must have rownames identical to the final norm_counts rownames. When method = "both", method-specific and consensus DE tables are stored in metadata(v)\$de_results_by_method and metadata(v)\$de_summary_by_method, and the active source is tracked in metadata(v)\$de_active_source.

Value

A VISTA object:

- **assays(v)**: norm_counts (matrix)

- **colData(v)**: sample_info (DataFrame)
- **rowData(v)**: row_data (DataFrame)
- **metadata(v)**: de_results, de_summary, de_cutoffs, group, comparison, provenance

See Also

[as_vista](#), [VISTA-class](#), [qualitative_hcl](#)

Examples

```
# Load example data
data("count_data", package = "VISTA")
data("sample_metadata", package = "VISTA")

# Create VISTA object with DESeq2 (default method)
vista <- create_vista(
  counts = count_data[seq_len(100), ],
  sample_info = sample_metadata[seq_len(6), ],
  column_geneid = "gene_id",
  group_column = "cond_long",
  group_numerator = "treatment1",
  group_denominator = "control",
  log2fc_cutoff = 0.6,
  pval_cutoff = 0.05
)

# Examine the VISTA object
vista

# Access comparisons
names(comparisons(vista))

# View DEG summary
deg_summary(vista)

# View cutoffs used
cutoffs(vista)

# Multiple comparisons example

vista_multi <- create_vista(
  counts = count_data,
  sample_info = sample_metadata,
  column_geneid = "gene_id",
  group_column = "cell",
  group_numerator = c("N052611", "N080611"),
  group_denominator = c("N61311", "N61311"),
  method = "edger",
  log2fc_cutoff = 1.0,
  pval_cutoff = 0.01
)
```

derive_vista_metadata *Derive starter sample metadata from count sample names*

Description

derive_vista_metadata() creates a starter sample_info table from count sample names. It is intended for projects where users have count columns but do not yet have a separate metadata sheet. The derived table can be edited, passed through read_vista_metadata(), and then aligned with match_vista_inputs().

Usage

```
derive_vista_metadata(
  counts,
  column_geneid = NULL,
  sample_names = NULL,
  parser = c("auto", "split", "regex", "none"),
  split = "_",
  fields = NULL,
  pattern = NULL,
  sample_column = "sample_names",
  repair_sample_names = c("auto", "none"),
  return_type = c("data.frame", "template"),
  verbose = TRUE
)
```

Arguments

counts	Count input accepted by read_vista_counts(), or the list returned by read_vista_counts().
column_geneid	Optional gene identifier column for raw tabular count inputs. Ignored when counts is the list output of read_vista_counts().
sample_names	Optional explicit sample names to derive metadata from. When supplied, these override names extracted from counts.
parser	Metadata parsing mode. "auto" tries a simple delimiter-based split when sample names have a consistent structure. "split" uses split explicitly. "regex" uses pattern. "none" returns only the sample_names column.
split	Delimiter used when parser = "split" or when "auto" chooses split-based parsing.
fields	Optional field names for parsed metadata columns. When omitted, VISTA uses part_1, part_2, etc.
pattern	Regular expression used when parser = "regex". Capture groups are mapped to fields in order.

sample_column	Name of the sample identifier column in the returned metadata. Default is "sample_names".
repair_sample_names	Strategy passed to read_vista_counts() when sample names are taken from counts. One of "auto" or "none".
return_type	Return "data.frame" (default) or "template". Both return a data frame; "template" adds empty placeholder columns for group and batch.
verbose	Logical; print an informational derivation summary.

Value

A data frame containing sample_names plus any parsed metadata columns.

Examples

```
data("count_data", package = "VISTA")
data("sample_metadata", package = "VISTA")

counts_in <- count_data[seq_len(8), c("gene_id", sample_metadata$sample_names[seq_len(6)]), drop = FALSE]
meta <- derive_vista_metadata(
  counts_in,
  column_geneid = "gene_id",
  parser = "regex",
  pattern = "SRR(\\d+)",
  fields = "run_id"
)
head(meta)
```

enrichMsigDB

Perform MSigDB over-representation analysis on a VISTA object

Description

This function performs ORA on a set of genes from a VISTA object. Gene identifiers stored as "ENSG00000187634:SAMD11" will be split at the colon and the appropriate part extracted based on from_type. Genes are then optionally converted via orgdb to match the identifier type required by the MSigDB gene sets.

Usage

```
enrichMsigDB(
  x,
  gene_list,
  from_type = "SYMBOL",
  orgdb,
  msigdb_category = "H",
  msigdb_subcategory = NULL,
  species = "Mus musculus",
```

```

background = NULL,
col_genetype = "GENETYPE",
feature_type = "protein-coding",
...
)

```

Arguments

<code>x</code>	A VISTA object.
<code>gene_list</code>	A non-empty character vector of gene identifiers. This argument is required and must not be NULL.
<code>from_type</code>	Identifier type of the genes in <code>gene_list</code> . One of "SYMBOL", "ENSEMBL", or "ENTREZID" (default "SYMBOL"). Ensembl IDs have version suffixes stripped automatically.
<code>orgdb</code>	An OrgDb object used for ID conversion. If omitted, a default is chosen based on species: <code>org.Mm.eg.db</code> for mouse, <code>org.Hs.eg.db</code> for human.
<code>msigdb_category</code>	MSigDB category (e.g. "H", "C2", "C5"). Default "H".
<code>msigdb_subcategory</code>	Optional MSigDB sub-collection (e.g. "BP" for C5). Default NULL.
<code>species</code>	Species name for MSigDB (default "Mus musculus").
<code>background</code>	Optional character vector of background gene IDs. If NULL, all features in the VISTA object (optionally filtered by <code>feature_type</code>) are used.
<code>col_genetype</code>	Column name in <code>rowData(x)</code> used to filter the background by gene type (default "GENETYPE").
<code>feature_type</code>	Gene type to retain in the background when background is NULL (default "protein-coding").
<code>...</code>	Additional arguments to pass to <code>clusterProfiler::enricher()</code> .

Value

A list containing a single `enrichResult` object named "enrich".

Examples

```

v <- example_vista()
genes <- head(as.character(row_data(v)$gene_id), 20)
if (requireNamespace('msigdb', quietly = TRUE)) {
  out <- try(enrichMsigDB(v, gene_list = genes, from_type = 'ENSEMBL', msigdb_category = 'H'), silent = TRUE)
  if (!inherits(out, 'try-error')) out
}

```

example_vista	<i>Build a small example VISTA object</i>
---------------	---

Description

Creates a lightweight VISTA object from built-in package datasets (count_data and sample_metadata) for use in examples and tutorials. The default call returns a precomputed object to keep examples and package checks fast. Non-default argument combinations fall back to rebuilding the object from the packaged example inputs.

Usage

```
example_vista(n_genes = 150, n_per_group = 3, method = "deseq2")
```

Arguments

n_genes	Number of genes to include (default 150).
n_per_group	Number of samples per group (control and treatment1) to include (default 3).
method	Differential expression backend passed to create_vista() .

Value

A VISTA object.

Examples

```
v <- example_vista()
v
```

export_vista_assets	<i>Export a complete VISTA asset bundle</i>
---------------------	---

Description

Generates a standardized folder with selected VISTA plots, tabular outputs, and a manifest describing all saved files.

Usage

```
export_vista_assets(
  x,
  out_dir = "vista_assets",
  sample_comparison = NULL,
  display_id = NULL,
  include_plots = c("pca", "mds", "corr_heatmap", "deg_bar", "volcano", "ma",
```

```

    "expression_heatmap"),
  include_data = c("comparison", "norm_counts", "sample_info", "row_data", "deg_summary",
    "cutoffs"),
  plot_format = "png",
  width = 8,
  height = 6,
  heatmap_height = 10,
  units = "in",
  dpi = 300,
  top_n_labels = 50,
  heatmap_n_genes = 60,
  write_excel = FALSE,
  overwrite = TRUE
)

```

Arguments

<code>x</code>	A VISTA object.
<code>out_dir</code>	Output directory for exported assets.
<code>sample_comparison</code>	Optional comparison to use for comparison-specific outputs. Defaults to the first available comparison.
<code>display_id</code>	Optional gene identifier column used in labeling for volcano/MA/heatmap plots.
<code>include_plots</code>	Character vector of plot keys to export. Supported: "pca", "mds", "corr_heatmap", "deg_bar", "deg_pie", "deg_donut", "volcano", "ma", "expression_heatmap".
<code>include_data</code>	Character vector of data keys passed to save_vista_data() .
<code>plot_format</code>	Plot format (e.g. "png" or "pdf").
<code>width</code>	Base plot width.
<code>height</code>	Base plot height.
<code>heatmap_height</code>	Height used specifically for expression heatmap export.
<code>units</code>	Plot dimension units.
<code>dpi</code>	Raster resolution for plots.
<code>top_n_labels</code>	Number of top genes to annotate in MA plots.
<code>heatmap_n_genes</code>	Number of top genes used in exported expression heatmaps.
<code>write_excel</code>	Logical; if TRUE, also writes a combined XLSX workbook for all requested include_data tables (requires writexl).
<code>overwrite</code>	Logical; if FALSE, aborts when out_dir already contains files.

Value

Invisibly, a list with `out_dir`, `sample_comparison`, `manifest`, `plot_files`, and `data_files`.

Examples

```
v <- example_vista()
out_dir <- file.path(tempdir(), "vista_assets_example")
res <- export_vista_assets(
  v,
  out_dir = out_dir,
  include_plots = "pca",
  include_data = "comparison"
)
names(res)
```

get_cell_fractions	<i>Retrieve stored cell fraction estimates</i>
--------------------	--

Description

Retrieve stored cell fraction estimates

Usage

```
get_cell_fractions(x)
```

Arguments

x A VISTA object.

Value

A data.frame of cell fraction estimates with samples in rows.

Examples

```
v <- example_vista()
if (requireNamespace('xCell2', quietly = TRUE)) {
  vx <- try(run_cell_deconvolution(v, method = 'xCell2'), silent = TRUE)
  if (!inherits(vx, 'try-error')) head(get_cell_fractions(vx))
}
```

get_celltype_barplot *Plot cell-type composition as stacked bars*

Description

Plot cell-type composition as stacked bars

Usage

```
get_celltype_barplot(
  x,
  group_column = NULL,
  sample_names = NULL,
  base_size = 12,
  cell_types = NULL,
  top_n = NULL,
  collapse_other = TRUE,
  normalize = c("sample", "none"),
  facet_by = c("group", "none")
)
```

Arguments

x	A VISTA object.
group_column	Optional column in sample_info(x) used to facet/order samples. If NULL, uses the active VISTA group column when available.
sample_names	Optional character vector of sample names to include.
base_size	Base font size.
cell_types	Optional character vector of cell types to keep.
top_n	Optional top-N cell types by mean score (ignored when cell_types is provided).
collapse_other	Logical; collapse non-selected cell types into "Other".
normalize	One of "sample" (default; per-sample relative scores) or "none".
facet_by	Faceting mode: "group" (default) or "none".

Value

A ggplot object.

Examples

```
mat <- matrix(rpois(20, lambda = 20), nrow = 5)
rownames(mat) <- paste0("gene", seq_len(5))
colnames(mat) <- paste0("sample", seq_len(4))
se <- SummarizedExperiment::SummarizedExperiment(
```

```

assays = list(norm_counts = mat),
colData = S4Vectors::DataFrame(
  cond = c("A", "A", "B", "B"),
  row.names = colnames(mat)
),
rowData = S4Vectors::DataFrame(
  gene_id = rownames(mat),
  row.names = rownames(mat)
)
)
v <- as_vista(se, group_column = "cond")
md <- S4Vectors::metadata(v)
md$cell_fractions <- data.frame(
  fibroblast = c(0.2, 0.3, 0.4, 0.5),
  epithelial = c(0.8, 0.7, 0.6, 0.5),
  row.names = colnames(mat)
)
S4Vectors::metadata(v) <- md
get_celltype_barplot(v, group_column = "cond")

```

get_celltype_group_dotplot

Plot group-level deconvolution scores as dot plot

Description

Plot group-level deconvolution scores as dot plot

Usage

```

get_celltype_group_dotplot(
  x,
  group_column = NULL,
  cell_types = NULL,
  top_n = 12,
  summary_fun = c("mean", "median"),
  error = c("se", "sd", "none"),
  add_points = TRUE,
  point_size = 2.5,
  base_size = 12
)

```

Arguments

x	A VISTA object.
group_column	Column in sample_info(x) that defines groups. If NULL, uses the active VISTA group column when available.
cell_types	Optional character vector of cell types to include.

top_n	Number of top cell types by mean score when cell_types is NULL.
summary_fun	One of "mean" or "median" for group summary.
error	Error-bar type: "se", "sd", or "none".
add_points	Logical; overlay sample-level jittered points.
point_size	Point size for summary points.
base_size	Base font size.

Value

A ggplot object.

Examples

```
mat <- matrix(rpois(20, lambda = 20), nrow = 5)
rownames(mat) <- paste0("gene", seq_len(5))
colnames(mat) <- paste0("sample", seq_len(4))
se <- SummarizedExperiment::SummarizedExperiment(
  assays = list(norm_counts = mat),
  colData = S4Vectors::DataFrame(
    cond = c("A", "A", "B", "B"),
    row.names = colnames(mat)
  ),
  rowData = S4Vectors::DataFrame(
    gene_id = rownames(mat),
    row.names = rownames(mat)
  )
)
v <- as_vista(se, group_column = "cond")
md <- S4Vectors::metadata(v)
md$cell_fractions <- data.frame(
  fibroblast = c(0.2, 0.3, 0.4, 0.5),
  epithelial = c(0.8, 0.7, 0.6, 0.5),
  row.names = colnames(mat)
)
S4Vectors::metadata(v) <- md
get_celltype_group_dotplot(v, group_column = "cond")
```

get_celltype_heatmap *Plot cell-type deconvolution heatmap*

Description

Plot cell-type deconvolution heatmap

Usage

```
get_celltype_heatmap(
  x,
  group_column = NULL,
  sample_names = NULL,
  cell_types = NULL,
  top_n = 20,
  transform = c("none", "zscore", "log1p"),
  cluster_rows = TRUE,
  cluster_columns = TRUE,
  label = FALSE,
  base_size = 11,
  return_type = c("plot", "matrix", "both")
)
```

Arguments

<code>x</code>	A VISTA object.
<code>group_column</code>	Optional grouping column from <code>sample_info(x)</code> . If provided and <code>cluster_columns = FALSE</code> , samples are ordered by this group.
<code>sample_names</code>	Optional character vector of sample names to include.
<code>cell_types</code>	Optional character vector of cell types to include.
<code>top_n</code>	Number of top cell types by mean score when <code>cell_types</code> is <code>NULL</code> .
<code>transform</code>	One of "none", "zscore", or "log1p".
<code>cluster_rows</code>	Logical; hierarchical cluster cell types.
<code>cluster_columns</code>	Logical; hierarchical cluster samples.
<code>label</code>	Logical; overlay numeric values on tiles.
<code>base_size</code>	Base font size.
<code>return_type</code>	One of "plot", "matrix", or "both".

Value

A ggplot object, matrix, or list depending on `return_type`.

Examples

```
mat <- matrix(rpois(20, lambda = 20), nrow = 5)
rownames(mat) <- paste0("gene", seq_len(5))
colnames(mat) <- paste0("sample", seq_len(4))
se <- SummarizedExperiment::SummarizedExperiment(
  assays = list(norm_counts = mat),
  colData = S4Vectors::DataFrame(
    cond = c("A", "A", "B", "B"),
    row.names = colnames(mat)
  ),
)
```

```

    rowData = S4Vectors::DataFrame(
      gene_id = rownames(mat),
      row.names = rownames(mat)
    )
  )
  v <- as_vista(se, group_column = "cond")
  md <- S4Vectors::metadata(v)
  md$cell_fractions <- data.frame(
    fibroblast = c(0.2, 0.3, 0.4, 0.5),
    epithelial = c(0.8, 0.7, 0.6, 0.5),
    row.names = colnames(mat)
  )
  S4Vectors::metadata(v) <- md
  get_celltype_heatmap(v, group_column = "cond")

```

get_corr_heatmap	<i>Draw a sample correlation heatmap</i>
------------------	--

Description

Plots sample-sample correlation matrix derived from normalized counts with optional clustering and annotations.

Usage

```

get_corr_heatmap(
  x,
  sample_group = NULL,
  group_column = NULL,
  genes = NULL,
  corr_method = "pearson",
  triangle = c("full", "lower", "upper"),
  cluster_by = c("correlation", "group", "input", "none"),
  show_diagonal = TRUE,
  label = TRUE,
  show_corr_values = NULL,
  label_color = "black",
  col_corr_values = NULL,
  label_size = 4,
  limits = NULL,
  base_size = 12,
  viridis_option = "viridis",
  viridis_direction = 1,
  viridis_begin = 0,
  viridis_end = 1
)

```

Arguments

<code>x</code>	A VISTA object.
<code>sample_group</code>	Optional character vector of groups (referencing <code>group_column</code>) to include.
<code>group_column</code>	Optional column name in <code>sample_info</code> defining the grouping used for filtering.
<code>genes</code>	Optional character vector of gene IDs to limit the matrix.
<code>corr_method</code>	Correlation method passed to <code>stats::cor()</code> (e.g., "pearson").
<code>triangle</code>	Either "full", "lower", or "upper" to control which triangle is drawn.
<code>cluster_by</code>	Ordering strategy for samples: "correlation" (default), "group", "input", or "none".
<code>show_diagonal</code>	Logical; include the correlation diagonal when TRUE.
<code>label</code>	Logical; overlay correlation coefficients as text.
<code>show_corr_values</code>	Deprecated alias for <code>label</code> . When supplied, it overrides <code>label</code> .
<code>label_color</code>	Color for the text labels.
<code>col_corr_values</code>	Deprecated alias for <code>label_color</code> . When supplied, it overrides <code>label_color</code> .
<code>label_size</code>	Numeric text size multiplier.
<code>limits</code>	Optional numeric vector of length two giving limits for the color scale.
<code>base_size</code>	Base theme size.
<code>viridis_option</code>	Character viridis palette name.
<code>viridis_direction</code>	Integer (1 or -1) controlling palette direction.
<code>viridis_begin, viridis_end</code>	Palette endpoints between 0 and 1.

Value

An object returned by this function.

Examples

```
v <- example_vista()
p <- get_corr_heatmap(v)
print(p)
```

get_deg_alluvial	<i>Plot alluvial diagram showing gene regulation transitions across comparisons</i>
------------------	---

Description

Plot alluvial diagram showing gene regulation transitions across comparisons

Usage

```
get_deg_alluvial(x, sample_comparisons, show_other = FALSE)
```

Arguments

x	A VISTA object (DE results in comparisons(x)).
sample_comparisons	Character vector of comparison names to include (≥ 2).
show_other	Logical; include "Other" genes. Default FALSE.

Value

A ggplot object.

Examples

```
data("count_data", package = "VISTA")
data("sample_metadata", package = "VISTA")
si <- as.data.frame(sample_metadata[seq_len(8), ], stringsAsFactors = FALSE)
trt_idx <- which(si$cond_long == "treatment1")
si$cond_long[trt_idx] <- rep(c("treatment1", "treatment2"), length.out = length(trt_idx))
si$groups <- si$cond_long
v <- create_vista(
  counts = count_data[seq_len(120), c("gene_id", si$sample_names)],
  sample_info = si,
  column_geneid = "gene_id",
  group_column = "cond_long",
  group_numerator = c("treatment1", "treatment2"),
  group_denominator = c("control", "control"),
  min_counts = 5,
  min_replicates = 1
)
if (requireNamespace('ggalluvial', quietly = TRUE)) {
  p <- get_deg_alluvial(v, sample_comparisons = names(comparisons(v)))
  print(p)
}
```

get_deg_count_barplot *Barplot of DEG counts (Up/Down) across comparisons*

Description

Barplot of DEG counts (Up/Down) across comparisons

Usage

```
get_deg_count_barplot(
  x,
  sample_comparisons = NULL,
  label = TRUE,
  base_size = 12,
  colors = c(Up = "red4", Down = "blue4"),
  facet_by = c("none", "regulation", "comparison"),
  facet_scales = "fixed"
)
```

Arguments

<code>x</code>	A VISTA object containing DEG summaries.
<code>sample_comparisons</code>	Optional character vector of comparison names to display.
<code>label</code>	Logical; overlay numeric counts above bars when TRUE.
<code>base_size</code>	Numeric base font size for the plot.
<code>colors</code>	Named vector giving fill colors for "Up" and "Down" bars.
<code>facet_by</code>	Either "none", "regulation", or "comparison" describing the facet variable. Use "none" for a single panel.
<code>facet_scales</code>	Scale option passed to <code>facet_wrap()</code> when faceting.

Value

An object returned by this function.

Examples

```
v <- example_vista()
p <- get_deg_count_barplot(v)
print(p)
```

get_deg_count_donutplot

Donut chart of DEG counts (Up/Down) across comparisons

Description

Donut chart of DEG counts (Up/Down) across comparisons

Usage

```
get_deg_count_donutplot(
  x,
  sample_comparisons = NULL,
  label = c("both", "count", "percent", "none"),
  label_digits = 1,
  base_size = 12,
  colors = c(Up = "red4", Down = "blue4"),
  show_other = FALSE,
  other_color = "grey70",
  text_color = "black",
  facet_by = c("comparison", "none"),
  ncol = NULL
)
```

Arguments

x	A VISTA object containing DEG summaries.
sample_comparisons	Optional character vector of comparison names to display.
label	Label mode: "both" (default), "count", "percent", or "none".
label_digits	Integer number of decimals used for percentage labels.
base_size	Numeric base font size for the plot.
colors	Named vector giving fill colors for "Up" and "Down" slices. When show_other = TRUE, an "Other" entry may also be supplied.
show_other	Logical; when TRUE, include non-DE genes as an "Other" slice using other_color unless overridden in colors.
other_color	Fill colour used for the "Other" slice when show_other = TRUE and colors does not include "Other".
text_color	Colour used for donut label text.
facet_by	Either "comparison" (default) to draw one donut per comparison, or "none" for a single donut.
ncol	Optional number of columns when faceting.

Value

An object returned by this function.

Examples

```
v <- example_vista()
p <- get_deg_count_donutplot(v)
print(p)
```

get_deg_count_pieplot *Pie chart of DEG counts (Up/Down) across comparisons*

Description

Pie chart of DEG counts (Up/Down) across comparisons

Usage

```
get_deg_count_pieplot(
  x,
  sample_comparisons = NULL,
  label = c("both", "count", "percent", "none"),
  label_digits = 1,
  base_size = 12,
  colors = c(Up = "red4", Down = "blue4"),
  show_other = FALSE,
  other_color = "grey70",
  text_color = "black",
  facet_by = c("comparison", "none"),
  ncol = NULL
)
```

Arguments

x	A VISTA object containing DEG summaries.
sample_comparisons	Optional character vector of comparison names to display.
label	Label mode: "both" (default), "count", "percent", or "none".
label_digits	Integer number of decimals used for percentage labels.
base_size	Numeric base font size for the plot.
colors	Named vector giving fill colors for "Up" and "Down" slices. When show_other = TRUE, an "Other" entry may also be supplied.
show_other	Logical; when TRUE, include non-DE genes as an "Other" slice using other_color unless overridden in colors.

other_color	Fill colour used for the "Other" slice when show_other = TRUE and colors does not include "Other".
text_color	Colour used for pie label text.
facet_by	Either "comparison" (default) to draw one pie per comparison, or "none" for a single pie.
ncol	Optional number of columns when faceting.

Value

An object returned by this function.

Examples

```
v <- example_vista()
p <- get_deg_count_pieplot(v)
print(p)
```

get_deg_venn_diagram *DEG Venn diagram*

Description

Visualizes overlaps between DEG sets for two to four comparisons.

Usage

```
get_deg_venn_diagram(
  x,
  sample_comparisons,
  regulation = "Up",
  palette = "Set 2",
  auto_scale = FALSE,
  show_percentage = TRUE,
  ...
)
```

Arguments

x	A VISTA object.
sample_comparisons	Character vector of 2–4 comparison names to include in the Venn diagram.
regulation	One of "Up", "Down", "Both", or "All" selecting which genes to include.
palette	Qualitative palette name passed to colorspace::qualitative_hcl() for fill colors.
auto_scale	Logical; pass through to ggvenn::ggvenn() to scale circles by size.
show_percentage	Logical; request percentage labels from ggvenn::ggvenn().
...	Additional arguments forwarded to ggvenn::ggvenn().

Value

An object returned by this function.

Examples

```
v <- example_vista()
comps <- names(comparisons(v))
if (length(comps) >= 2) {
  p <- get_deg_venn_diagram(v, sample_comparisons = comps[seq_len(2)])
  print(p)
}
```

get_enrichment_chord *Chord diagram of enrichment gene-pathway relationships*

Description

Draws a chord diagram linking genes to the enriched pathways they belong to. Chords can be coloured by fold-change, regulation direction, or source pathway.

Usage

```
get_enrichment_chord(
  x,
  vista = NULL,
  sample_comparison = NULL,
  pathways = NULL,
  top_n = 8,
  pathway_column = c("Description", "ID"),
  gene_column = c("auto", "geneID", "core_enrichment"),
  gene_sep = "/",
  min_pathways = 1,
  max_genes = 50,
  gene_order_by = c("none", "foldchange", "abs_foldchange"),
  gene_id_column = NULL,
  display_id = NULL,
  color_by = c("foldchange", "regulation", "pathway"),
  up_color = "#D73027",
  down_color = "#1A9850",
  other_color = "grey70",
  pathway_colors = NULL,
  transparency = 0.4,
  gap_degree = 2,
  label_cex = 0.7,
  title = NULL
)
```

Arguments

x	An enrichResult, gseaResult, or compareClusterResult from clusterProfiler, or a list containing an enrich element (e.g. output of get_msigdb_enrichment()).
vista	Optional VISTA object. Required when color_by is "foldchange" or "regulation".
sample_comparison	Character scalar naming the DE comparison in vista to pull log2FC values from. Required when vista is supplied.
pathways	Optional character vector of pathway names to include. Matches against pathway_column.
top_n	Number of top pathways to display when pathways is NULL (default 8).
pathway_column	Column in the enrichment result to match pathway names: "Description" (default) or "ID".
gene_column	Column storing gene members: "auto" (default), "geneID", or "core_enrichment".
gene_sep	Delimiter separating genes in gene_column (default "/").
min_pathways	Minimum number of pathways a gene must appear in to be shown. Set to 2 to display only hub genes shared across terms. Default 1 (show all genes).
max_genes	Maximum number of genes to display (default 50). A safety cap for readability.
gene_order_by	Order of gene sectors in the chord plot: "none" (default), "foldchange" (descending log2FC), or "abs_foldchange" (descending absolute log2FC). Fold-change based ordering requires vista + sample_comparison.
gene_id_column	Column in rowData(vista) used to map enrichment gene IDs to vista row-names (for FC lookup).
display_id	Column in rowData(vista) providing display-friendly gene names.
color_by	How to colour chords: "foldchange" (continuous gradient), "regulation" (Up / Down / Other), or "pathway" (source pathway). Falls back to "pathway" when vista is NULL.
up_color	Colour for up-regulated genes (default "#D73027").
down_color	Colour for down-regulated genes (default "#1A9850").
other_color	Colour for non-significant genes (default "grey70").
pathway_colors	Optional named vector of colours for pathway sectors. When NULL, colours are generated from an HCL palette.
transparency	Chord transparency, 0–1 (default 0.4).
gap_degree	Gap between sectors in degrees (default 2).
label_cex	Text size for sector labels (default 0.7).
title	Optional plot title.

Details

The plot reveals **hub genes** (those driving multiple enriched terms) and pathway redundancy (terms sharing many genes). This complements [get_enrichment_plot\(\)](#) (which shows significance) and [get_pathway_heatmap\(\)](#) (which shows expression patterns).

Value

Invisibly returns a list with:

gene_data Tibble of genes with pathway membership and (optionally) fold-change values.

hub_genes Character vector of genes appearing in two or more pathways.

The chord diagram is drawn as a side effect.

Examples

```
v <- example_vista()
msig <- get_msigdb_enrichment(
  v,
  sample_comparison = names(comparisons(v))[1],
  regulation = "Both",
  msigdb_category = "H",
  from_type = "ENSEMBL"
)
get_enrichment_chord(msig, top_n = 5)

data("count_data", package = "VISTA")
data("sample_metadata", package = "VISTA")

vista <- create_vista(
  counts = count_data[seq_len(200), ],
  sample_info = sample_metadata[seq_len(6), ],
  column_geneid = "gene_id",
  group_column = "cond_long",
  group_numerator = "treatment1",
  group_denominator = "control"
)

msig <- get_msigdb_enrichment(
  vista,
  sample_comparison = names(comparisons(vista))[1],
  regulation = "Up", from_type = "ENSEMBL"
)

# Simple: pathway-coloured chords
get_enrichment_chord(msig)

# With fold-change colouring
get_enrichment_chord(
  msig, vista = vista,
  sample_comparison = names(comparisons(vista))[1],
  color_by = "foldchange"
)

# Hub genes only (shared across 2+ pathways)
pw_long <- get_pathway_genes(msig, return_type = "long")
if (any(table(pw_long$gene) >= 2)) {
```

```

    get_enrichment_chord(msig, min_pathways = 2)
}

```

get_enrichment_plot	<i>Plot enrichment results using $-\log_{10}(\text{FDR})$</i>
---------------------	--

Description

Generates a dot plot of enrichment results for `enrichResult`, `gseaResult`, or `compareClusterResult` objects (including those returned by `enrichMsigDB()`). Points are sized by gene/set count and coloured by $-\log_{10}(\text{FDR})$. For `compareCluster` results, the plot is faceted by cluster with top terms selected per cluster.

Usage

```
get_enrichment_plot(x, top_n = 10, title = NULL)
```

Arguments

<code>x</code>	An object of class <code>enrichResult</code> , <code>gseaResult</code> , or <code>compareClusterResult</code> .
<code>top_n</code>	Integer; number of top terms to plot (per cluster for <code>compareCluster</code>).
<code>title</code>	Optional plot title.

Value

A `ggplot2` object.

Examples

```

## Not run:
v <- example_vista()
com <- names(comparisons(v))[1]
if (requireNamespace('msigdbR', quietly = TRUE)) {
  ms <- try(get_msigdb_enrichment(v, sample_comparison = com, regulation = 'Up', from_type = 'ENSEMBL'), silent = TRUE)
  if (!inherits(ms, 'try-error') && !is.null(ms$enrich)) print(get_enrichment_plot(ms$enrich))
}

## End(Not run)

```

get_expression_barplot

Plot gene expression as barplots

Description

Displays selected genes as grouped summary bars or individual sample-level bars. By default, expression is summarized per group with mean $\pm$ SD bars. With `by = "sample"`, each sample is drawn separately.

Usage

```
get_expression_barplot(
  x,
  genes,
  sample_group = NULL,
  group_column = NULL,
  log_transform = TRUE,
  stats_group = FALSE,
  facet_scale = "free_y",
  facet_scales = facet_scale,
  facet_nrow = NULL,
  facet_ncol = NULL,
  p.label = "p.signif",
  comparisons = NULL,
  display_id = NULL,
  display_from = NULL,
  display_orgdb = NULL,
  by = c("group", "sample"),
  sample_order = c("input", "group", "expression"),
  fill_by = NULL,
  facet_by = c("auto", "gene", "none")
)
```

Arguments

<code>x</code>	A VISTA object.
<code>genes</code>	Character vector (≤ 25 genes) to plot.
<code>sample_group</code>	Optional character vector of groups (from <code>group_column</code>) to include.
<code>group_column</code>	Optional column name in <code>sample_info</code> to use for grouping samples.
<code>log_transform</code>	Logical; log2-transform expression before plotting.
<code>stats_group</code>	Logical; add statistical comparisons between groups when TRUE.
<code>facet_scale</code>	Scaling option passed to <code>facet_wrap()</code> (deprecated; use <code>facet_scales</code>).
<code>facet_scales</code>	Facet scales argument passed to <code>facet_wrap()</code> when faceting by gene.

facet_nrow, facet_ncol	Optional layout passed to facet_wrap() when faceting.
p.label	Label format for ggpubr::stat_compare_means().
comparisons	Optional list of specific group comparisons for stat_compare_means().
display_id	Optional ID/column name to use for labels/facets. If supplied and present in rowData(x), those values are used; otherwise falls back to ID mapping.
display_from	Optional source ID type for mapping (used when display_id is not found in rowData).
display_orgdb	Optional OrgDb object used for ID mapping when display_id is set but not found in rowData.
by	One of "group" (default; summarize replicates by group) or "sample" (show one bar per sample).
sample_order	Ordering used when by = "sample": "input" preserves the current sample order, "group" groups samples by group_column, and "expression" ranks samples by mean expression across the selected genes.
fill_by	Fill mapping for by = "sample" barplots. Use "group" (default) or any discrete column from the joined plotting data, such as a sample metadata column or "sample". Group-summary barplots (by = "group") only support group-based fill because each bar already represents an aggregated group mean.
facet_by	Faceting mode: "auto" (default; facet by gene when more than one gene is requested), "gene", or "none". For multiple genes, "none" falls back to "gene" to preserve readability.

Value

A ggplot2 object.

Examples

```
# Create VISTA object
data("count_data", package = "VISTA")
data("sample_metadata", package = "VISTA")

vista <- create_vista(
  counts = count_data[seq_len(200), ],
  sample_info = sample_metadata[seq_len(6), ],
  column_geneid = "gene_id",
  group_column = "cond_long",
  group_numerator = "treatment1",
  group_denominator = "control"
)

# Plot expression for select genes
genes <- rownames(vista)[seq_len(3)]
get_expression_barplot(vista, genes = genes)

# With statistics
get_expression_barplot(vista, genes = genes, stats_group = TRUE)
```

```
# Without log transformation
get_expression_barplot(vista, genes = genes, log_transform = FALSE)
```

```
get_expression_boxplot
```

Plot gene expression distributions as boxplots

Description

Displays per-sample or per-group distributions for selected genes using normalized counts. The x-axis unit is controlled by `by`, and faceting is controlled explicitly by `facet_by`.

Usage

```
get_expression_boxplot(
  x,
  genes = NULL,
  sample_group = NULL,
  group_column = NULL,
  log_transform = TRUE,
  display_id = NULL,
  display_from = NULL,
  display_orgdb = NULL,
  facet_scales = "free_y",
  facet_nrow = NULL,
  facet_ncol = NULL,
  stats_group = FALSE,
  p.label = "p.signif",
  comparisons = NULL,
  pool_genes = FALSE,
  by = "group",
  facet_by = "auto",
  fill_by = NULL,
  sample_order = c("input", "group", "expression")
)
```

Arguments

<code>x</code>	A VISTA object.
<code>genes</code>	Optional character vector of genes to display (≤ 20). Defaults to all genes.
<code>sample_group</code>	Optional character vector specifying which groups (as defined by <code>group_column</code>) to include.
<code>group_column</code>	Optional column name in <code>sample_info</code> used as the grouping variable.
<code>log_transform</code>	Logical; apply $\log_2(x + 1)$ transform before plotting.

display_id	Optional column in rowData(x) to use for gene labels (facets).
display_from	Optional source ID type for mapping (used when display_id is not found in rowData).
display_orgdb	Optional OrgDb object used for ID mapping when display_id is set but not found in rowData.
facet_scales	Facet scales argument passed to facet_wrap() (default "free_y").
facet_nrow, facet_ncol	Optional layout passed to facet_wrap() when faceting.
stats_group	Logical; add statistical comparisons between groups when TRUE. Only supported when by = "group".
p.label	Label format for ggpubr::stat_compare_means().
comparisons	Optional list of specific group comparisons for stat_compare_means().
pool_genes	Logical; when TRUE, pool selected genes into one distribution per x-axis category (scenario 1).
by	When pool_genes = TRUE, either "group" or "sample" (x-axis and optional fill). When pool_genes = FALSE, either "group" or "gene" (x-axis for per-gene distributions).
facet_by	Faceting control: for pooled genes, "group" or "none"; for per-gene, "none" (default) or "gene". "auto" selects the most readable layout.
fill_by	Fill mapping. Special values are "group", "gene", and "x" (the plotted x-axis variable, when available). You may also supply a discrete column from the joined plotting data, such as a sample metadata column or "sample".
sample_order	Ordering used when sample names are shown on the x-axis: "input", "group", or "expression".

Value

An object returned by this function.

Examples

```
v <- example_vista()
genes <- head(rownames(v), 3)
p <- get_expression_boxplot(v, genes = genes)
print(p)
```

get_expression_chromosome_plot

Chromosome plot for expression

Description

Convenience wrapper around get_chromosome_plot() for expression-based colouring (optional group mean, rowData columns, or assay columns).

Usage

```

get_expression_chromosome_plot(
  x,
  txdb,
  keytype = "GENEID",
  id_column = NULL,
  genes = NULL,
  sample_group = NULL,
  group_column = NULL,
  value_column = NULL,
  value_from = c("rowdata", "assay"),
  value_assay = "norm_counts",
  facet_value_columns = FALSE,
  side_by_side_groups = FALSE,
  paginate = TRUE,
  panels_per_page = 24,
  summarise_replicates = FALSE,
  summarise_method = c("mean", "median"),
  group_value = NULL,
  label_n = 20,
  label_genes = c("top", "all", "none"),
  display_id = NULL,
  line_length = 0.02,
  line_width = 0.6,
  filter_chrom = NULL,
  log_transform = TRUE,
  value_label = "log2(mean expr)"
)

```

Arguments

x	A VISTA object.
txdb	A TxDb object (e.g., from GenomicFeatures).
keytype	Key type in the TxDb matching id_column (default "GENEID").
id_column	Optional column in rowData(x) used to match to TxDb keys. When NULL, rownames(x) are used as keys.
genes	Optional character vector of gene IDs to label (alternative to label_top_n). When provided, all genes are plotted but only these are labeled. Defaults to NULL (no explicit label set).
sample_group	Optional character vector of groups (from group_column) used to subset assay columns when value_from = "assay".
group_column	Optional column name in sample_info used for sample_group selection and replicate summarisation.
value_column	Optional column in rowData(x) used for colouring.
value_from	Source for value_column data: "rowdata" (default) or "assay". When "assay", the selected assay column is copied into rowData temporarily for colouring.

value_assay	Assay name to pull values from when value_from = "assay". Default "norm_counts".
facet_value_columns	Ignored (kept for compatibility); multiple value_columns are always arranged in a chromosome-by-column grid.
side_by_side_groups	Logical; when plotting multiple value_columns, arrange one row per chromosome with one panel per selected sample/group. This is useful for direct side-by-side group comparison.
paginate	Logical; when TRUE, split large multi-panel chromosome plots into pages to avoid patchwork viewport-size errors.
panels_per_page	Maximum number of panels per page when paginate = TRUE. Set to NULL to disable automatic paging.
summarise_replicates	Logical; when TRUE and value_from = "assay", aggregate replicates by group_column before plotting.
summarise_method	"mean" or "median" summarisation used when summarise_replicates = TRUE.
group_value	Optional group label (from group_column); when supplied, uses mean expression for that group for colouring (assay norm_counts).
label_n	Integer; number of genes with the largest absolute values to label when genes is not supplied. Set to 0 to disable automatic labels.
label_genes	One of "top" (default), "all", or "none" controlling chromosome text labels when genes is not supplied.
display_id	Optional column in rowData(x) to use for point labels (fallback to gene_id/rownames).
line_length	Horizontal half-length (in megabases) of the tick used to mark each gene position. Default 0.02. Increase for longer ticks.
line_width	Line width of the tick marks. Default 0.6.
filter_chrom	Optional character vector of chromosomes to keep (e.g., c("chr1", "chr2")). When NULL, all chromosomes returned by the TxDb are shown.
log_transform	Logical; when group_value is used (and value_column is NULL) or when value_from = "assay", apply $\log_2(x + 1)$ before coloring.
value_label	Optional legend title override for the colour scale.

Details

- If multiple value_columns are supplied, one plot is produced per column with its own colour legend titled by that column. Chromosomes are laid out top-down in a single column: for each chromosome, plots for all value_columns appear sequentially. Legends for each column are shown only on the first occurrence (at the bottom) to avoid overlap. Requires **patchwork**; otherwise a list of plots is returned.
- Set side_by_side_groups = TRUE to place selected groups/samples in separate columns for each chromosome (same-row comparison).
- paginate = TRUE automatically splits large panel collections into a list of patchwork pages, which prevents "viewport too small" rendering errors in IDE plot panes.

- For `value_from = "assay"`, colours follow the VISTA ecosystem: group colours are used when `summarise_replicates = TRUE`, while sample colours are derived from group colours when plotting individual replicates.
- For `value_from = "assay"` with `value_column = NULL`, all selected assay columns are used (samples after `sample_group` filtering, or groups after replicate summarisation).
- Labels are kept consistent across `value_columns`: if `genes` is provided it is used for all panels; otherwise the top `label_n` (by absolute value in the first `value_column`) are used for all panels.
- When `value_from = "assay"`, the specified assay column is copied into `rowData` on the fly so it can be used for colouring.
- When `group_value` is provided (and no `value_column`), colouring is based on log2 mean expression for that group (assay `norm_counts`), using a data-driven colour scale. Ignored when `value_column` is supplied.

Value

A `ggplot2` object or a list of `ggplot2` objects when multiple `value_columns` are provided and **patchwork** is unavailable.

Examples

```
v <- example_vista()
p <- try(get_expression_chromosome_plot(v), silent = TRUE)
if (!inherits(p, 'try-error')) print(p)
```

get_expression_density

Plot expression distributions as density curves

Description

Shows expression distributions pooled across the selected genes, coloured by group (or sample), with optional faceting by group or sample.

Usage

```
get_expression_density(
  x,
  genes = NULL,
  sample_group = NULL,
  group_column = NULL,
  log_transform = TRUE,
  facet_scales = "free",
  facet_nrow = NULL,
  facet_ncol = NULL,
  alpha = 0.4,
```

```

    adjust = 1,
    color_by = c("group", "sample"),
    facet_by = c("none", "group", "sample"),
    sample_order = c("input", "group", "expression"),
    palette = NULL,
    colors = NULL
  )

```

Arguments

<code>x</code>	A VISTA object.
<code>genes</code>	Optional character vector of genes to display. Defaults to all genes.
<code>sample_group</code>	Optional character vector specifying which groups (as defined by <code>group_column</code>) to include.
<code>group_column</code>	Optional column name in <code>sample_info</code> used as the grouping variable.
<code>log_transform</code>	Logical; apply $\log_2(x + 1)$ transform before plotting.
<code>facet_scales</code>	Facet scales argument passed to <code>facet_wrap()</code> (default "free").
<code>facet_nrow, facet_ncol</code>	Optional layout passed to <code>facet_wrap()</code> when faceting.
<code>alpha</code>	Numeric transparency for density fill.
<code>adjust</code>	Bandwidth adjustment factor passed to <code>geom_density()</code> .
<code>color_by</code>	Either "group" (default) or "sample" to choose fill/color variable.
<code>facet_by</code>	One of "none" (default), "group", or "sample" to facet densities.
<code>sample_order</code>	Ordering for sample-level display: "input", "group", or "expression".
<code>palette</code>	Optional qualitative palette name used when generating colours.
<code>colors</code>	Optional named character vector of manual colours overriding palette.

Value

A ggplot2 object.

Examples

```

v <- example_vista()
genes <- head(rownames(v), 3)
p <- get_expression_density(v, genes = genes)
print(p)

```

get_expression_heatmap

Expression heatmap

Description

Summarizes expression for selected genes/groups via ComplexHeatmap with optional transformations and annotations. With only a VISTA object, the function will plot the top variable genes across all samples.

Usage

```
get_expression_heatmap(
  x,
  genes = NULL,
  top_n = 50,
  sample_group = NULL,
  group_column = NULL,
  value_transform = c("zscore", "log2", "raw"),
  summarise_replicates = TRUE,
  summarise_method = c("mean", "median"),
  convert_rowmeans = FALSE,
  display_id = NULL,
  display_from = NULL,
  display_orgdb = NULL,
  repair_genes = FALSE,
  show_row_names = NULL,
  label_size = 10,
  label_specific_rows = NULL,
  label_specific_rows_gp = grid::gpar(fontsize = 5),
  show_column_names = TRUE,
  cluster_rows = TRUE,
  show_row_dend = TRUE,
  cluster_columns = TRUE,
  kmeans_k = NULL,
  annotate_columns = FALSE,
  cluster_by = NULL,
  column_anno_palette = "Dark 3",
  column_anno_colors = NULL,
  color_default = TRUE,
  col = NULL,
  heatmap_name = NULL,
  show_heatmap_legend = TRUE,
  return_type = c("heatmap", "clusters", "both"),
  ...
)
```

Arguments

x	A VISTA object.
genes	Optional character vector of gene identifiers to display. When omitted, VISTA selects the top variable genes across the included samples.
top_n	Integer number of genes to select automatically when genes = NULL. Defaults to 50.
sample_group	Character vector of group labels specifying which samples to include (based on the selected grouping column).
group_column	Optional column name in sample_info used to interpret sample_group.
value_transform	One of "zscore", "log2", or "raw" controlling how expression values are transformed.
summarise_replicates	Logical; average replicates per group before plotting when TRUE.
summarise_method	"mean" or "median" summarization used when summarise_replicates = TRUE.
convert_rowmeans	Logical; subtract row means prior to plotting.
display_id	Optional ID/column name to use for row labels. If supplied
display_from	Optional source ID type for mapping (used when display_id
display_orgdb	Optional OrgDb object used for ID mapping when
repair_genes	Logical; if TRUE, split gene_id strings such as ID:SYMBOL to display the symbol.
show_row_names	Logical; display row names (genes) beside the heatmap. When NULL, VISTA turns labels on automatically for auto-selected genes.
label_size	Numeric font size for row names.
label_specific_rows	Optional character vector of row names to highlight via anno_mark().
label_specific_rows_gp	grid::gpar() object controlling the highlighted row labels.
show_column_names	Logical; draw column names when TRUE.
cluster_rows	Logical; cluster rows when drawing the heatmap.
show_row_dend	Logical; display the row dendrogram.
cluster_columns	Logical; cluster columns.
kmeans_k	Optional integer specifying the number of k-means clusters to compute for rows.
annotate_columns	Logical or character vector. TRUE adds the group_column annotation and also includes cluster_by when supplied; a character vector adds multiple annotations from sample_info.
cluster_by	Optional annotation column used to split/cluster columns. Defaults to the first annotation column when annotate_columns is enabled.

column_anno_palette	Qualitative palette name used for the column annotation.
column_anno_colors	Optional named list of annotation color vectors. Each element should be a named character vector keyed by the levels of one annotation column.
color_default	Logical; use the default blue-white-red palette when TRUE. Set to FALSE to supply col.
col	Optional <code>circlize::colorRamp2</code> function used when <code>color_default = FALSE</code> .
heatmap_name	Optional legend title.
show_heatmap_legend	Logical; display the heatmap legend.
return_type	"heatmap", "clusters", or "both" selecting the returned object.
...	Additional arguments passed to <code>ComplexHeatmap::Heatmap()</code> .

Value

An object returned by this function.

A `ComplexHeatmap` object, a cluster data frame, or a list containing both depending on `return_type`.

Examples

```
v <- example_vista()
genes <- head(rownames(v), 20)
if (requireNamespace('ComplexHeatmap', quietly = TRUE) &&
    requireNamespace('circlize', quietly = TRUE)) {
  hm <- get_expression_heatmap(
    v,
    genes = genes,
    sample_group = unique(as.character(sample_info(v)$cond_long)),
    return_type = 'heatmap'
  )
  ComplexHeatmap::draw(hm)
}
v <- example_vista()
if (requireNamespace("ComplexHeatmap", quietly = TRUE) &&
    requireNamespace("circlize", quietly = TRUE)) {
  hm <- get_expression_heatmap(v, return_type = "heatmap")
  ComplexHeatmap::draw(hm)
}
```

get_expression_joyplot

Plot expression distributions as ridgelines

Description

Shows per-group (or per-sample) expression distributions pooled across the selected genes using ridge (joy) plots. Genes are pooled; no faceting to keep shapes comparable.

Usage

```
get_expression_joyplot(
  x,
  genes = NULL,
  sample_group = NULL,
  group_column = NULL,
  log_transform = TRUE,
  alpha = 0.7,
  scale = 1.2,
  y_by = c("group", "sample"),
  color_by = c("group", "sample"),
  sample_order = c("input", "group", "expression"),
  palette = NULL,
  colors = NULL
)
```

Arguments

<code>x</code>	A VISTA object.
<code>genes</code>	Optional character vector of genes to display. Defaults to all genes.
<code>sample_group</code>	Optional character vector specifying which groups (as defined by <code>group_column</code>) to include.
<code>group_column</code>	Optional column name in <code>sample_info</code> used as the grouping variable.
<code>log_transform</code>	Logical; apply $\log_2(x + 1)$ transform before plotting.
<code>alpha</code>	Numeric transparency for fills.
<code>scale</code>	Numeric scaling factor for ridges (passed to <code>geom_density_ridges()</code>).
<code>y_by</code>	Either "group" (default) or "sample" to choose the y-axis (ridge) grouping.
<code>color_by</code>	Either "group" (default) or "sample" to choose fill colors.
<code>sample_order</code>	Ordering for sample-level display: "input", "group", or "expression".
<code>palette</code>	Optional qualitative palette name used when generating colours.
<code>colors</code>	Optional named character vector of manual colours overriding palette.

Value

A ggplot2 object.

Examples

```
v <- example_vista()
genes <- head(rownames(v), 3)
p <- get_expression_joyplot(v, genes = genes)
print(p)
```

get_expression_lineplot

Gene expression line plot

Description

Plots normalized expression for selected genes across samples or summarized groups with optional transformations and group faceting.

Usage

```
get_expression_lineplot(
  x,
  genes = NULL,
  sample_group = NULL,
  group_column = NULL,
  log_transform = TRUE,
  display_id = NULL,
  display_from = NULL,
  display_orgdb = NULL,
  facet_scales = "free_y",
  facet_nrow = NULL,
  facet_ncol = NULL,
  stats_group = FALSE,
  p.label = "p.signif",
  comparisons = NULL,
  pool_genes = FALSE,
  by = c("sample", "group"),
  facet_by = c("auto", "group", "gene", "none"),
  fill_by = NULL,
  sample_order = c("input", "group", "expression"),
  value_transform = NULL,
  palette = NULL,
  colors = NULL,
  line_width = 1,
  point_size = 2,
  base_size = 12
)
```

Arguments

x	A VISTA object.
genes	Character vector of gene identifiers to plot.
sample_group	Optional character vector specifying which groups (values taken from group_column) to include.
group_column	Optional column name in sample_info defining the grouping/faceting variable.

log_transform	Logical; log2-transform expression values before plotting.
display_id	Optional ID/column name to use for gene labels. If supplied and present in rowData(x), those values are used.
display_from	Optional source ID type for mapping (reserved for compatibility with other expression plotting APIs).
display_orgdb	Optional OrgDb object used for ID mapping when display_id is set but not found in rowData.
facet_scales	Scaling option passed to facet_wrap().
facet_nrow, facet_ncol	Optional layout passed to facet_wrap() when faceting.
stats_group	Logical retained for API consistency. Statistical overlays are not currently added by get_expression_lineplot().
p.label	Label format retained for API consistency with other expression plots.
comparisons	Optional list of comparisons retained for API consistency.
pool_genes	Logical; when TRUE, average the selected genes into a single trajectory.
by	Plot unit: "sample" (default) or "group" to average replicates before plotting.
facet_by	Faceting mode: "auto" (default), "none", "group", or "gene".
fill_by	Argument retained for API consistency; ignored because line plots use colour rather than fill.
sample_order	Ordering used for sample-level plots: "input", "group", or "expression".
value_transform	Deprecated compatibility alias for transformation choice; one of "log2", "zscore", or "none".
palette	Optional qualitative palette name used for gene colours.
colors	Optional named character vector of manual gene colours.
line_width	Line width.
point_size	Point size.
base_size	Base theme size.

Value

An object returned by this function.

Examples

```
v <- example_vista()
genes <- head(rownames(v), 3)
p <- get_expression_lineplot(v, genes = genes)
print(p)
```

get_expression_lollipop

Plot expression as a lollipop chart

Description

Displays selected genes as stem-and-dot (lollipop) plots. By default, expression is summarized per group (mean across replicates). With `by = "sample"`, the plot shows individual samples.

Usage

```
get_expression_lollipop(
  x,
  genes,
  sample_group = NULL,
  group_column = NULL,
  by = c("group", "sample"),
  sample_order = c("input", "group", "expression"),
  facet_by = c("auto", "gene", "none"),
  log_transform = TRUE,
  facet_scale = "free_y",
  facet_nrow = NULL,
  facet_ncol = NULL,
  point_size = 6,
  line_size = 1.2,
  label = TRUE,
  label_digits = 1,
  display_id = NULL,
  display_from = NULL,
  display_orgdb = NULL
)
```

Arguments

<code>x</code>	A VISTA object.
<code>genes</code>	Character vector (≤ 15 genes) to plot.
<code>sample_group</code>	Optional character vector of groups (from <code>group_column</code>) to include.
<code>group_column</code>	Optional column name in <code>sample_info</code> to use for grouping samples.
<code>by</code>	One of "group" (default; summarize replicates by group) or "sample" (show individual samples).
<code>sample_order</code>	Ordering used when <code>by = "sample"</code> : "input" preserves the current sample order, "group" groups samples by <code>group_column</code> , and "expression" ranks samples by mean expression across the selected genes.
<code>facet_by</code>	Faceting mode: "auto" (default; facet by gene when more than one gene is requested), "gene", or "none". For multiple genes, "none" falls back to "gene" to avoid unreadable combined panels.

log_transform	Logical; log2-transform expression before plotting.
facet_scale	Scaling option passed to facet_wrap() when plotting multiple genes.
facet_nrow, facet_ncol	Optional layout passed to facet_wrap() when faceting.
point_size	Numeric size of the dots.
line_size	Numeric size of the stems.
label	Logical; draw numeric labels above the dots.
label_digits	Integer; digits to show in labels when label = TRUE.
display_id	Optional ID/column name to use for labels/facets. If supplied and present in rowData(x), those values are used; otherwise falls back to ID mapping.
display_from	Optional source ID type for mapping (used when display_id is not found in rowData).
display_orgdb	Optional OrgDb object used for ID mapping when display_id is set but not found in rowData.

Value

An object returned by this function.

A ggplot2 object.

Examples

```
v <- example_vista()
genes <- head(rownames(v), 5)
p <- get_expression_lollipop(v, genes = genes)
print(p)
data("count_data", package = "VISTA")
data("sample_metadata", package = "VISTA")

vista <- create_vista(
  counts = count_data[seq_len(200), ],
  sample_info = sample_metadata[seq_len(6), ],
  column_geneid = "gene_id",
  group_column = "cond_long",
  group_numerator = "treatment1",
  group_denominator = "control"
)

genes <- rownames(vista)[seq_len(3)]
get_expression_lollipop(vista, genes = genes)
get_expression_lollipop(vista, genes = genes[seq_len(2)], by = "sample")
```

get_expression_matrix *Retrieve an expression matrix from a VISTA object*

Description

Returns the specified assay (default "norm_counts") with optional gene/sample subsetting, replicate summarization by group, and simple transformations.

Usage

```
get_expression_matrix(
  x,
  assay_name = "norm_counts",
  genes = NULL,
  sample_names = NULL,
  group_column = NULL,
  summarise = FALSE,
  transform = c("none", "log2", "zscore")
)
```

Arguments

x	A VISTA object.
assay_name	Assay to extract (default: "norm_counts").
genes	Optional character vector of gene IDs to keep.
sample_names	Optional character vector of sample IDs (or group labels when summarise = TRUE) to keep.
group_column	Optional column in sample_info used when summarising replicates. Defaults to the stored group_column.
summarise	Logical; if TRUE, averages replicates per group_column.
transform	One of "none", "log2", or "zscore" applied after subsetting and optional summarisation. Z-scores are computed within the selected samples/columns.

Value

A numeric matrix with genes in rows and samples (or groups) in columns.

Examples

```
v <- example_vista()
m <- get_expression_matrix(v)
dim(m)
```

`get_expression_raincloud`*Raincloud plot of expression values*

Description

Uses `gggrain::geom_rain()` to combine a half-violin, boxplot, and jittered points per sample/group to show distribution, summary, and individual values.

Usage

```
get_expression_raincloud(  
  x,  
  genes = NULL,  
  sample_group = NULL,  
  group_column = NULL,  
  by = "group",  
  value_transform = c("log2", "zscore", "none"),  
  summarise = FALSE,  
  facet_by = c("auto", "gene", "none"),  
  fill_by = NULL,  
  facet_nrow = NULL,  
  facet_ncol = NULL,  
  sample_order = c("input", "group", "expression"),  
  rain_side = c("r", "l", "f", "f1x1", "f2x2"),  
  id.long.var = NULL,  
  alpha = 0.5,  
  point_size = 1.5,  
  p.label = "p.signif",  
  stats_group = FALSE,  
  stats_method = "t.test",  
  label = FALSE,  
  label_column = "gene",  
  label_size = 3,  
  label_max_overlaps = 50,  
  display_id = NULL,  
  display_from = NULL,  
  display_orgdb = NULL  
)
```

Arguments

<code>x</code>	A VISTA object.
<code>genes</code>	Optional character vector of gene IDs to include; defaults to all genes selected by the plotting mode.
<code>sample_group</code>	Optional subset of groups (values of <code>group_column</code>) to keep.

group_column	Grouping column in sample_info; defaults to the stored grouping.
by	Plot unit. Violin plots currently support only "group".
value_transform	Deprecated compatibility alias. "log2" maps to log_transform = TRUE, "none" maps to FALSE, and "zscore" applies a per-gene z-score transform.
summarise	Logical; when TRUE, collapse replicates within each group for each gene (one value per gene per group). This is useful for pooled multi-gene raincloud plots where each dot represents one gene-level summary in a group.
facet_by	Faceting mode. Uses the same argument pattern as get_expression_boxplot() , but pool_genes = TRUE falls back to "none" because pooled violins already aggregate across genes.
fill_by	Fill mapping. Uses the same values as get_expression_boxplot() , including discrete sample metadata columns.
facet_nrow, facet_ncol	Optional layout passed to facet_wrap() when faceting.
sample_order	Ordering for sample-level display before values are grouped into violins.
rain_side	Side specification passed to ggrain::geom_rain(); one of "r", "l", "f", "f1x1", or "f2x2".
id.long.var	Optional column name passed to ggrain::geom_rain() as id.long.var to identify repeated measurements.
alpha	Alpha for jittered points.
point_size	Point size for jittered points.
p.label	Label type passed to ggpubr::stat_compare_means().
stats_group	Logical; add pairwise statistical tests when TRUE.
stats_method	Statistical method passed to ggpubr::stat_compare_means().
label	Logical; add text labels to points using ggrepel.
label_column	Column name in the plotting data used for labels. Defaults to "gene" for expression raincloud plots.
label_size	Text size for point labels.
label_max_overlaps	Maximum overlaps passed to ggrepel::geom_text_repel().
display_id	Optional ID/column name to use for labels. If supplied and present in rowData(x), those values are used; otherwise falls back to ID mapping.
display_from	Optional source ID type for mapping (used when display_id is not found in rowData).
display_orgdb	Optional OrgDb object used for ID mapping when display_id is set but not found in rowData.

Details

id.long.var controls which repeated unit is connected by lines in ggrain::geom_rain().

Recommended usage for expression raincloud plots:

- `id.long.var = NULL` (default): best for clean distribution summaries.
- `id.long.var = "gene"`: best when plotting a small number of genes and showing gene-level trajectories across x levels.
- `id.long.var = "<subject_id_column>"`: best for paired/repeated-measure designs when a subject ID exists in `sample_info`.
- `id.long.var = "sample"` or the grouping variable is usually less informative and can over-connect points.
- Point labels (`label = TRUE`) work best with `facet_by = "none"` or a small number of genes.

For identifier display consistency with other VISTA plotting functions, set `display_id` (for example, "SYMBOL"). When provided, genes can be given in that ID space, and default point labels use the mapped display IDs.

Value

A `ggplot2` object.

Examples

```
v <- example_vista()
genes <- head(rownames(v), 5)
p <- get_expression_raincloud(v, genes = genes, summarise = TRUE)
print(p)
```

`get_expression_scatter`

Compare normalized expression between two samples or groups

Description

Plots gene-level expression for two selected samples or group means, colours points by local density (viridis), labels the most divergent genes, and reports Pearson/Spearman correlation.

Usage

```
get_expression_scatter(
  x,
  sample_x,
  sample_y,
  by = c("sample", "group"),
  group_column = NULL,
  genes = NULL,
  log_transform = TRUE,
  label_n = 20,
  label_size = 3,
  method = c("pearson", "spearman"),
  display_id = NULL
)
```

Arguments

<code>x</code>	A VISTA object.
<code>sample_x</code>	First sample or group to plot (character scalar).
<code>sample_y</code>	Second sample or group to plot (character scalar).
<code>by</code>	One of "sample" (use individual samples) or "group" (average replicates within group_column before plotting). Default "sample".
<code>group_column</code>	Column in sample_info used when by = "group" (defaults to stored grouping column).
<code>genes</code>	Optional character vector of genes to include; defaults to all.
<code>log_transform</code>	Logical; apply $\log_2(x + 1)$ transform. Default TRUE.
<code>label_n</code>	Integer; number of most divergent genes to label (ranked by $ x - y $). Set to 0 to disable labels.
<code>label_size</code>	Numeric size for labeled genes.
<code>method</code>	Correlation method for the subtitle; "pearson" (default) or "spearman".
<code>display_id</code>	Optional column in rowData(x) to use for point labels (fallback to gene_id/rownames when not available).

Value

A ggplot2 object.

Examples

```
v <- example_vista()
si <- as.data.frame(sample_info(v))
genes <- head(rownames(v), 50)
p <- get_expression_scatter(
  v,
  sample_x = si$sample_names[1],
  sample_y = si$sample_names[2],
  genes = genes
)
print(p)
```

get_expression_violinplot

Violin plot of expression values

Description

Mirrors the main user-facing arguments of `get_expression_boxplot()` so the two geoms can be swapped with minimal code changes. Violin plots currently keep group-based semantics (by = "group") because a violin requires replicate-level distributions within groups rather than one value per sample.

Usage

```

get_expression_violinplot(
  x,
  genes = NULL,
  sample_group = NULL,
  group_column = NULL,
  log_transform = TRUE,
  display_id = NULL,
  display_from = NULL,
  display_orgdb = NULL,
  facet_scales = "free_y",
  facet_nrow = NULL,
  facet_ncol = NULL,
  stats_group = FALSE,
  p.label = "p.signif",
  comparisons = NULL,
  pool_genes = FALSE,
  by = "group",
  facet_by = c("auto", "gene", "none"),
  fill_by = NULL,
  sample_order = c("input", "group", "expression"),
  value_transform = NULL,
  summarise = FALSE
)

```

Arguments

x	A VISTA object.
genes	Optional character vector of gene IDs to include; defaults to all genes selected by the plotting mode.
sample_group	Optional subset of groups (values of group_column) to keep.
group_column	Grouping column in sample_info; defaults to the stored grouping.
log_transform	Logical; log2-transform expression values before plotting.
display_id	Optional ID/column name to use for labels/facets. If supplied and present in rowData(x), those values are used.
display_from	Optional source ID type for mapping (reserved for compatibility with get_expression_boxplot()).
display_orgdb	Optional OrgDb object used for ID mapping when display_id is set but not found in rowData (reserved for compatibility with get_expression_boxplot()).
facet_scales	Scaling option passed to facet_wrap().
facet_nrow, facet_ncol	Optional layout passed to facet_wrap() when faceting.
stats_group	Logical; add statistical comparisons between groups when TRUE.
p.label	Label format for ggpubr::stat_compare_means().
comparisons	Optional list of specific group comparisons for stat_compare_means().

pool_genes	Logical; pool all selected genes into one violin per group.
by	Plot unit. Violin plots currently support only "group".
facet_by	Faceting mode. Uses the same argument pattern as <code>get_expression_boxplot()</code> , but <code>pool_genes = TRUE</code> falls back to "none" because pooled violins already aggregate across genes.
fill_by	Fill mapping. Uses the same values as <code>get_expression_boxplot()</code> , including discrete sample metadata columns.
sample_order	Ordering for sample-level display before values are grouped into violins.
value_transform	Deprecated compatibility alias. "log2" maps to <code>log_transform = TRUE</code> , "none" maps to <code>FALSE</code> , and "zscore" applies a per-gene z-score transform.
summarise	Logical retained for compatibility. Violin plots always use replicate-level values, so <code>summarise = TRUE</code> is ignored with a warning.

Value

A ggplot2 object.

Examples

```
v <- example_vista()
genes <- head(rownames(v), 4)
p <- get_expression_violinplot(v, genes = genes)
print(p)
```

get_foldchange_barplot

Plot fold-change barplots across comparisons for selected genes

Description

Plots log2 fold changes for selected genes across one or more comparisons. `facet_by` controls per-gene or per-comparison layout explicitly.

Usage

```
get_foldchange_barplot(
  x,
  genes,
  sample_comparisons = NULL,
  coord_flip = FALSE,
  display_id = NULL,
  display_from = NULL,
  display_orgdb = NULL,
  sort_by = c("input", "log2fc", "abs_log2fc"),
  facet_scales = "free_y",
```

```

    facet_nrow = NULL,
    facet_ncol = NULL,
    facet_by = c("auto", "gene", "comparison", "none")
  )

```

Arguments

<code>x</code>	A VISTA object containing differential expression results.
<code>genes</code>	Character vector of gene IDs to plot.
<code>sample_comparisons</code>	Optional character vector of comparison names to include; defaults to all available.
<code>coord_flip</code>	Logical; flip axes when TRUE.
<code>display_id</code>	Optional column in <code>rowData(x)</code> to use for gene labels. Input gene matching still uses <code>gene_id</code> .
<code>display_from</code>	Optional source ID type for mapping when <code>display_id</code> is not present in <code>rowData(x)</code> .
<code>display_orgdb</code>	Optional <code>OrgDb</code> object used for identifier mapping when <code>display_id</code> is not present in <code>rowData(x)</code> .
<code>sort_by</code>	How to order genes when faceting: "input" (use supplied order), "log2fc" (descending log2FC of the first comparison), or "abs_log2fc" (descending max absolute log2FC across comparisons).
<code>facet_scales</code>	Facet scales argument passed to <code>facet_wrap()</code> when faceting (default "free_y").
<code>facet_nrow, facet_ncol</code>	Optional layout passed to <code>facet_wrap()</code> when faceting.
<code>facet_by</code>	Faceting mode: "auto" (default), "gene", "comparison", or "none".

Value

An object returned by this function.

A `ggplot2` object.

Examples

```

v <- example_vista()
comp <- names(comparisons(v))[1]
genes <- head(as.character(comparisons(v)[[comp]]$gene_id), 10)
p <- get_foldchange_barplot(v, sample_comparison = comp, genes = genes)
print(p)
data("count_data", package = "VISTA")
data("sample_metadata", package = "VISTA")

vista <- create_vista(
  counts = count_data[seq_len(200), ],
  sample_info = sample_metadata[seq_len(6), ],
  column_geneid = "gene_id",
  group_column = "cond_long",
  group_numerator = "treatment1",

```

```

    group_denominator = "control"
  )

genes <- rownames(vista)[seq_len(3)]
get_foldchange_barplot(vista, genes = genes)
get_foldchange_barplot(vista, genes = genes, facet_by = "gene")

```

get_foldchange_boxplot

Plot fold-change distributions across comparisons

Description

Builds boxplots of log2 fold changes for selected genes and comparisons, optionally adding statistics.

Usage

```

get_foldchange_boxplot(
  x,
  genes = NULL,
  sample_comparisons = NULL,
  display_id = NULL,
  display_from = NULL,
  display_orgdb = NULL,
  facet_scales = "free_x",
  facet_nrow = NULL,
  facet_ncol = NULL,
  facet_by = c("auto", "comparison", "none"),
  p.label = "p.signif",
  stats_group = FALSE,
  stats_method = "t.test"
)

```

Arguments

x	A VISTA object containing differential expression results.
genes	Optional character vector of gene IDs to include.
sample_comparisons	Optional character vector of comparison names to plot.
display_id	Optional ID/column name used to interpret genes and, when possible, map fold-change gene identifiers to display-friendly labels.
display_from	Optional source ID type for mapping when display_id is not present in rowData(x).
display_orgdb	Optional OrgDb object used for identifier mapping when display_id is not present in rowData(x).

facet_scales	Facet scales argument passed to facet_wrap() when facet_by != "none" (default "free_x").
facet_nrow, facet_ncol	Optional layout passed to facet_wrap() when faceting.
facet_by	Faceting mode: "auto" (default), "comparison", or "none".
p.label	Label type passed to ggpubr::stat_compare_means().
stats_group	Logical; add pairwise statistical tests when TRUE.
stats_method	Statistical method passed to ggpubr::stat_compare_means().

Value

An object returned by this function.

Examples

```
v <- example_vista()
comp <- names(comparisons(v))[1]
genes <- head(as.character(comparisons(v)[[comp]]$gene_id), 10)
p <- get_foldchange_boxplot(v, sample_comparison = comp, genes = genes)
print(p)
```

get_foldchange_chromosome_plot

Chromosome plot for fold change

Description

Convenience wrapper around get_chromosome_plot() for fold-change colouring. When multiple comparisons are supplied, panels are faceted by comparison with log2FC clipped to +/-2.

Usage

```
get_foldchange_chromosome_plot(
  x,
  txdb,
  keytype = "GENEID",
  id_column = NULL,
  genes = NULL,
  sample_comparison = NULL,
  value_column = NULL,
  label_n = 20,
  display_id = NULL,
  display_from = NULL,
  display_orgdb = NULL,
  line_length = 0.02,
  line_width = 0.6,
  filter_chrom = NULL
)
```

Arguments

<code>x</code>	A VISTA object.
<code>txdb</code>	A TxDb object (e.g., from GenomicFeatures).
<code>keytype</code>	Key type in the TxDb matching <code>id_column</code> (default "GENEID").
<code>id_column</code>	Optional column in <code>rowData(x)</code> used to match to TxDb keys. When NULL, <code>rownames(x)</code> are used as keys.
<code>genes</code>	Optional character vector of gene IDs to label (alternative to <code>label_top_n</code>). When provided, all genes are plotted but only these are labeled. Defaults to NULL (no explicit label set).
<code>sample_comparison</code>	Optional comparison name (or vector of names) used for fold-change colouring.
<code>value_column</code>	Optional column in <code>rowData(x)</code> used for colouring.
<code>label_n</code>	Integer; number of genes with the largest absolute fold-change to label when genes is not supplied. Set to 0 to disable labels.
<code>display_id</code>	Optional column in <code>rowData(x)</code> to use for point labels (fallback to <code>gene_id/rownames</code>).
<code>display_from</code>	Optional source ID type for mapping when <code>display_id</code> is not present in <code>rowData(x)</code> .
<code>display_orgdb</code>	Optional OrgDb object used for identifier mapping when <code>display_id</code> is not present in <code>rowData(x)</code> .
<code>line_length</code>	Horizontal half-length (in megabases) of the tick used to mark each gene position. Default 0.02. Increase for longer ticks.
<code>line_width</code>	Line width of the tick marks. Default 0.6.
<code>filter_chrom</code>	Optional character vector of chromosomes to keep (e.g., <code>c("chr1", "chr2")</code>). When NULL, all chromosomes returned by the TxDb are shown.

Value

A ggplot2 object.

Examples

```
v <- example_vista()
p <- try(get_foldchange_chromosome_plot(v, sample_comparison = names(comparisons(v))[1]), silent = TRUE)
if (!inherits(p, 'try-error')) print(p)
```

get_foldchange_heatmap

Fold-change heatmap

Description

Visualizes log2 fold-change matrices across comparisons with ComplexHeatmap, supporting clustering and annotations. With only a VISTA object, the function will plot the top DE genes across the stored comparisons.

Usage

```

get_foldchange_heatmap(
  x,
  sample_comparisons = NULL,
  genes = NULL,
  top_n = 10,
  display_id = NULL,
  display_from = NULL,
  display_orgdb = NULL,
  repair_genes = FALSE,
  show_row_names = NULL,
  label_size = 10,
  label_specific_rows = NULL,
  label_specific_rows_gp = grid::gpar(fontsize = 5),
  show_column_names = TRUE,
  cluster_rows = TRUE,
  show_row_dend = TRUE,
  cluster_columns = TRUE,
  kmeans_k = NULL,
  annotate_columns = FALSE,
  column_anno_palette = "Set2",
  color_default = TRUE,
  col = NULL,
  heatmap_name = NULL,
  show_heatmap_legend = TRUE,
  return_type = c("heatmap", "clusters", "both"),
  ...
)

```

Arguments

<code>x</code>	A VISTA object with stored differential expression results.
<code>sample_comparisons</code>	Optional character vector of comparison names to include. Defaults to all available comparisons.
<code>genes</code>	Optional character vector of gene identifiers to display. When omitted, VISTA selects the top DE genes from each comparison by absolute log2 fold-change.
<code>top_n</code>	Integer number of genes to select per comparison when <code>genes = NULL</code> . Defaults to 10.
<code>display_id</code>	Optional ID/column name to use for plot labels. If supplied
<code>display_from</code>	Optional source ID type for mapping (used when <code>display_id</code>
<code>display_orgdb</code>	Optional OrgDb object used for ID mapping when
<code>repair_genes</code>	Logical; attempt to simplify <code>gene_id</code> strings by removing prefixes.
<code>show_row_names</code>	Logical; draw row (gene) names. When <code>NULL</code> , VISTA turns labels on automatically for auto-selected genes.
<code>label_size</code>	Numeric font size for row names.

label_specific_rows Optional character vector of genes to highlight with `anno_mark()`.
label_specific_rows_gp `grid::gpar()` object controlling highlighted labels.
show_column_names Logical; draw column labels.
cluster_rows Logical; cluster rows.
show_row_dend Logical; display the row dendrogram.
cluster_columns Logical; cluster columns.
kmeans_k Optional integer specifying the number of k-means clusters for rows.
annotate_columns Logical; add annotation bars keyed to the sample grouping column.
column_anno_palette Qualitative palette name used for column annotations.
color_default Logical; use the default diverging palette when TRUE. Set to FALSE to supply col.
col Optional `circlize::colorRamp2` color function used when `color_default = FALSE`.
heatmap_name Optional legend title.
show_heatmap_legend Logical; display the heatmap legend.
return_type "heatmap", "clusters", or "both" selecting the returned value.
... Additional arguments forwarded to `ComplexHeatmap::Heatmap()`.

Value

An object returned by this function.

A `ComplexHeatmap` object, a cluster data frame, or a list containing both depending on `return_type`.

Examples

```

v <- example_vista()
comp <- names(comparisons(v))[1]
genes <- unique(stats::na.omit(as.character(comparisons(v)[[comp]]$gene_id)))[seq_len(20)]
if (requireNamespace('ComplexHeatmap', quietly = TRUE) &&
    requireNamespace('circlize', quietly = TRUE)) {
  hm <- get_foldchange_heatmap(
    v,
    sample_comparisons = comp,
    genes = genes,
    return_type = 'heatmap'
  )
  ComplexHeatmap::draw(hm)
}
v <- example_vista()

```

```

if (requireNamespace("ComplexHeatmap", quietly = TRUE) &&
    requireNamespace("circlize", quietly = TRUE)) {
  hm <- get_foldchange_heatmap(v, return_type = "heatmap")
  ComplexHeatmap::draw(hm)
}

```

get_foldchange_lineplot

Fold-change line plot across comparisons

Description

Plots log2 fold-change trajectories for selected genes across multiple comparisons, optionally clustering genes.

Usage

```

get_foldchange_lineplot(
  x,
  sample_comparisons,
  genes = NULL,
  display_id = NULL,
  display_from = NULL,
  display_orgdb = NULL,
  km = NULL,
  facet_by = c("none", "cluster"),
  facet_scales = "fixed",
  facet_nrow = NULL,
  facet_ncol = NULL,
  alpha = 0.5,
  palette = NULL,
  show_summary = TRUE,
  summary_color = NULL,
  summary_linewidth = 1,
  summary_fun = c("median", "mean"),
  base_size = 14
)

```

Arguments

x	A VISTA object containing differential expression results.
sample_comparisons	Character vector of comparison names to include.
genes	Optional character vector of gene identifiers to plot. Defaults to all genes.
display_id	Optional ID/column name used to interpret genes and map the returned cluster table to display-friendly labels.

display_from	Optional source ID type for mapping when display_id is not present in rowData(x).
display_orgdb	Optional OrgDb object used for identifier mapping when display_id is not present in rowData(x).
km	Optional integer specifying the number of k-means clusters to compute; NULL disables clustering.
facet_by	Faceting mode: "none" (default) or "cluster" when k-means clustering is requested.
facet_scales	Facet scales argument passed to facet_wrap() when facet_by = "cluster" (default "fixed").
facet_nrow, facet_ncol	Optional layout passed to facet_wrap() when faceting.
alpha	Numeric alpha applied to individual gene lines.
palette	Optional named or unnamed color vector used for cluster lines.
show_summary	Logical; overlay a summary line per cluster when TRUE.
summary_color	Color used for the summary line. When NULL, uses the first comparison color (if stored) for consistency across plots.
summary_linewidth	Numeric line width for the summary line.
summary_fun	Character string selecting "median" or "mean" for the summary statistic.
base_size	Numeric base theme size.

Value

An object returned by this function.

A list with plot (the ggplot2 object) and clustered_data (gene-to-cluster assignments).

Examples

```
v <- example_vista()
comp <- names(comparisons(v))[1]
genes <- head(as.character(comparisons(v)[[comp]]$gene_id), 5)
p <- get_foldchange_lineplot(v, sample_comparison = comp, genes = genes)
print(p)
```

get_foldchange_lollipop

Fold-change plotting helpers (overview)

Description

One-stop doc for fold-change plots:

- get_foldchange_barplot(): log2FC by comparison (bars).
- get_foldchange_boxplot(): log2FC distributions per comparison (boxes).
- get_foldchange_lollipop(): log2FC stems/dots; supports 1–2 comparisons.
- get_foldchange_lineplot(): log2FC trajectories across comparisons (optional clustering).

Usage

```

get_foldchange_lollipop(
  x,
  sample_comparison,
  genes = NULL,
  sort_by = c("input", "log2fc", "abs_log2fc"),
  palette = NULL,
  point_size = 6,
  line_size = 1.2,
  label = TRUE,
  label_digits = 2,
  display_id = NULL,
  display_from = NULL,
  display_orgdb = NULL,
  dodge_width = 0.5,
  facet_scales = "free_y",
  facet_nrow = NULL,
  facet_ncol = NULL,
  facet_by = c("auto", "gene", "comparison", "none")
)

```

Arguments

<code>x</code>	A VISTA object.
<code>sample_comparison</code>	Character vector of length 1 or 2 naming the comparison(s) to plot (must exist in <code>metadata(x)\$de_results</code>).
<code>genes</code>	Optional character vector of gene IDs to include. When <code>NULL</code> , all genes in the specified comparison(s) are shown.
<code>sort_by</code>	How to sort genes on the y-axis: "input" (use supplied order), "log2fc" (descending log2FC; for two comparisons uses the first comparison), or "abs_log2fc" (descending max abs log2FC across comparisons).
<code>palette</code>	For a single comparison, named vector of colors for "pos", "neg", and "zero" sign classes (set to <code>NULL</code> to disable color by sign). For two comparisons, a named or unnamed vector of colors with one entry per comparison (defaults to a qualitative palette).
<code>point_size</code>	Numeric size of dots.
<code>line_size</code>	Numeric size of stems (linewidth).
<code>label</code>	Logical; draw numeric labels next to the dots.
<code>label_digits</code>	Integer; digits to show in labels when <code>label = TRUE</code> .
<code>display_id</code>	Optional column in <code>rowData(x)</code> used to interpret genes and to label plotted genes.
<code>display_from</code>	Optional source ID type for mapping when <code>display_id</code> is not present in <code>rowData(x)</code> .
<code>display_orgdb</code>	Optional <code>OrgDb</code> object used for identifier mapping when <code>display_id</code> is not present in <code>rowData(x)</code> .

dodge_width	Horizontal separation between comparisons when plotting two comparisons on the same axis.
facet_scales	Facet scales argument passed to facet_wrap() when facet_by != "none" (default "free_y").
facet_nrow, facet_ncol	Optional layout passed to facet_wrap() when faceting.
facet_by	Faceting mode: "auto" (default), "gene", "comparison", or "none".

Details

Shared arguments: x (VISTA with DE results), sample_comparisons/sample_comparison, genes, display_id for label mapping, sort_by (where supported), faceting controls (facet_*), and comparison colours pulled from Plot log2 fold changes as a lollipop chart (one or two comparisons)

Extracts log2 fold changes from stored differential expression results and plots them as stems and dots, with labels and a zero reference line. You can optionally provide two comparisons; in that case both comparisons are drawn side-by-side, coloured by comparison.

Value

An object returned by this function.

A ggplot2 object.

Examples

```
v <- example_vista()
comp <- names(comparisons(v))[1]
genes <- head(as.character(comparisons(v)[[comp]]$gene_id), 10)
p <- get_foldchange_lollipop(v, sample_comparison = comp, genes = genes)
print(p)
vista <- example_vista()
comp_name <- names(comparisons(vista))[1]
genes <- rownames(vista)[seq_len(3)]
get_foldchange_lollipop(vista, sample_comparison = comp_name, genes = genes)
```

get_foldchange_matrix *Extract a log2 fold-change matrix*

Description

Returns a gene-by-comparison matrix of log2 fold changes stored in a VISTA object.

Usage

```
get_foldchange_matrix(x, sample_comparisons = NULL, genes = NULL)
```

Arguments

x	A VISTA object containing differential expression results.
sample_comparisons	Optional character vector of comparison names. Defaults to all available comparisons.
genes	Optional character vector of gene identifiers. When omitted, all genes present in row_data(x) are returned.

Value

A numeric matrix with genes in rows and comparisons in columns.

Examples

```
v <- example_vista()
mat <- get_foldchange_matrix(v)
dim(mat)
```

get_foldchange_raincloud

Raincloud plot of fold-change distributions across comparisons

Description

Uses `ggplot2::geom_rain()` to display log2 fold-change distributions for selected comparisons, with optional statistical testing across comparisons.

Usage

```
get_foldchange_raincloud(
  x,
  genes = NULL,
  sample_comparisons = NULL,
  facet_scales = "free_x",
  facet_nrow = NULL,
  facet_ncol = NULL,
  facet_by = c("auto", "comparison", "none"),
  rain_side = c("r", "l", "f", "f1x1", "f2x2"),
  id.long.var = NULL,
  alpha = 0.5,
  point_size = 1.5,
  p.label = "p.signif",
  stats_group = FALSE,
  stats_method = "t.test",
  label = FALSE,
  label_column = "gene_id",
```

```

    label_size = 3,
    label_max_overlaps = 50,
    display_id = NULL,
    display_from = NULL,
    display_orgdb = NULL
  )

```

Arguments

<code>x</code>	A VISTA object containing differential expression results.
<code>genes</code>	Optional character vector of gene IDs to include.
<code>sample_comparisons</code>	Optional character vector of comparison names to plot.
<code>facet_scales</code>	Facet scales argument passed to <code>facet_wrap()</code> when <code>facet_by != "none"</code> (default <code>"free_x"</code>).
<code>facet_nrow</code> , <code>facet_ncol</code>	Optional layout passed to <code>facet_wrap()</code> when faceting.
<code>facet_by</code>	Faceting mode: <code>"auto"</code> (default), <code>"comparison"</code> , or <code>"none"</code> .
<code>rain_side</code>	Side specification passed to <code>gggrain::geom_rain()</code> ; one of <code>"r"</code> , <code>"l"</code> , <code>"f"</code> , <code>"f1x1"</code> , or <code>"f2x2"</code> .
<code>id.long.var</code>	Optional column name passed to <code>gggrain::geom_rain()</code> as <code>id.long.var</code> to identify repeated measurements.
<code>alpha</code>	Alpha for jittered points.
<code>point_size</code>	Point size for jittered points.
<code>p.label</code>	Label type passed to <code>ggpubr::stat_compare_means()</code> .
<code>stats_group</code>	Logical; add pairwise statistical tests when TRUE.
<code>stats_method</code>	Statistical method passed to <code>ggpubr::stat_compare_means()</code> .
<code>label</code>	Logical; add text labels to points using <code>ggrepel</code> .
<code>label_column</code>	Column name in the plotting data used for labels. Defaults to <code>"gene_id"</code> for fold-change raincloud plots.
<code>label_size</code>	Text size for point labels.
<code>label_max_overlaps</code>	Maximum overlaps passed to <code>ggrepel::geom_text_repel()</code> .
<code>display_id</code>	Optional ID/column name to use for labels. If supplied and present in <code>rowData(x)</code> , those values are used; otherwise falls back to ID mapping.
<code>display_from</code>	Optional source ID type for mapping (used when <code>display_id</code> is not found in <code>rowData</code>).
<code>display_orgdb</code>	Optional <code>OrgDb</code> object used for ID mapping when <code>display_id</code> is set but not found in <code>rowData</code> .

Details

`id.long.var` controls which repeated unit is connected by lines in `ggplot::geom_rain()`.

Recommended usage for fold-change raincloud plots:

- `id.long.var = NULL` (default): best for clean distribution summaries.
- `id.long.var = "gene_id"`: most useful option; connects each gene across comparisons.
- `id.long.var = "comparison"` is generally not useful because comparison is already on the x-axis.
- Continuous value columns (e.g. `log2FoldChange`) are not suitable identifiers for line connections.
- Point labels (`label = TRUE`) work best with `facet_by = "none"` unless only a small set of genes is shown.

For identifier display consistency with other VISTA plotting functions, set `display_id` (for example, `"SYMBOL"`). When provided, genes can be given in that ID space, and default point labels use the mapped display IDs.

Value

A `ggplot2` object.

Examples

```
v <- example_vista()
comp <- names(comparisons(v))[1]
genes <- head(as.character(comparisons(v)[[comp]]$gene_id), 20)
p <- get_foldchange_raincloud(v, sample_comparison = comp, genes = genes)
print(p)
```

`get_foldchange_scatter`

Fold-change scatterplot between two comparisons

Description

Plots $\log_2$ fold changes from two stored comparisons against each other, with points coloured by concordant/discordant regulation based on the cutoffs saved in the VISTA object.

Usage

```
get_foldchange_scatter(
  x,
  sample_comparisons,
  genes = NULL,
  display_id = NULL,
  display_from = NULL,
```

```

display_orgdb = NULL,
label_n = 0,
alpha = 0.5,
geometry = c("point", "hex"),
method = c("pearson", "spearman"),
colors = c(`Up/Up` = "#1b9e77", `Down/Down` = "#7570b3", `Up/Down` = "#d95f02",
  `Down/Up` = "#e7298a", Other = "grey70"),
point_size = 1.5,
label_size = 3,
base_size = 12
)

```

Arguments

<code>x</code>	A VISTA object containing DE results.
<code>sample_comparisons</code>	Character vector of length 2 naming the comparisons.
<code>genes</code>	Optional character vector of gene identifiers used to subset the comparison overlap before plotting.
<code>display_id</code>	Optional ID/column name used to interpret genes and to label highlighted points.
<code>display_from</code>	Optional source ID type for mapping when <code>display_id</code> is not present in <code>rowData(x)</code> .
<code>display_orgdb</code>	Optional <code>OrgDb</code> object used for identifier mapping when <code>display_id</code> is not present in <code>rowData(x)</code> .
<code>label_n</code>	Integer; number of most extreme points to label (by $ \log_2FC1 + \log_2FC2 $).
<code>alpha</code>	Point transparency.
<code>geometry</code>	Geometry used for the data layer: "point" or "hex".
<code>method</code>	Correlation method for the subtitle; "pearson" or "spearman".
<code>colors</code>	Named vector of concordance colours.
<code>point_size</code>	Point size used when <code>geometry = "point"</code> .
<code>label_size</code>	Text size for labeled genes.
<code>base_size</code>	Base theme size.

Details

Points are coloured by concordance status using fixed colours:

- Up/Up = #1b9e77
- Down/Down = #7570b3
- Up/Down = #d95f02
- Down/Up = #e7298a
- Other = grey70

Regulation is derived from the `log2fc` and `pval` cutoffs stored in `cutoffs(x)` (and `p_value_type` from the same list, defaulting to "padj").

Value

A ggplot2 object.

Examples

```
## Not run:
data('count_data', package = 'VISTA')
data('sample_metadata', package = 'VISTA')
cell_levels <- unique(sample_metadata$cell)
if (length(cell_levels) >= 3) {
  v <- create_vista(count_data[seq_len(150), ], sample_metadata, column_geneid = 'gene_id', group_column = 'cell',
    group_numerator = cell_levels[2:3], group_denominator = rep(cell_levels[1], 2),
    min_counts = 5, min_replicates = 1)
  comp_names <- names(comparisons(v))[seq_len(2)]
  p <- get_foldchange_scatter(v, sample_comparisons = comp_names)
  print(p)
}

## End(Not run)
```

get_genes_by_regulation

Get Genes by Regulation

Description

Extract gene IDs by regulation class from selected comparisons in a VISTA object.

Usage

```
get_genes_by_regulation(
  x,
  sample_comparisons,
  regulation = "Both",
  top_n = NULL,
  display_id = NULL,
  return_type = c("list", "table")
)
```

Arguments

x	A VISTA object.
sample_comparisons	Character vector of comparison names to include.
regulation	One of "Up", "Down", "Both", or "All" (default: "Both").
top_n	Optional integer limiting each comparison to the top DE genes ranked by absolute log2 fold change after regulation filtering.

display_id	Optional column name in rowData(x) to append when return_type = "table", for example "SYMBOL".
return_type	Either "list" (default) to return gene ID vectors or "table" to return one data frame per comparison with gene metadata.

Value

A named list of character vectors (one per comparison) when return_type = "list", or a named list of data frames when return_type = "table".

Examples

```
v <- example_vista()
comp <- names(comparisons(v))[1]
genes <- get_genes_by_regulation(v, sample_comparisons = comp, regulation = 'Up', top_n = 20)
str(genes, max.level = 1)
```

get_go_enrichment	<i>Run GO enrichment directly from a VISTA comparison</i>
-------------------	---

Description

Run GO enrichment directly from a VISTA comparison

Usage

```
get_go_enrichment(
  x,
  sample_comparison,
  regulation = c("Up", "Down", "Both", "All"),
  ont = c("BP", "MF", "CC"),
  from_type = "SYMBOL",
  orgdb = NULL,
  species = "Mus musculus",
  background = NULL,
  ...
)
```

Arguments

x	A VISTA object with DE results.
sample_comparison	Comparison name to use.
regulation	One of "Up", "Down", "Both", or "All"; selects genes.
ont	GO ontology: "BP", "MF", or "CC".
from_type	Identifier type in the DE tables (default "SYMBOL").

orgdb	OrgDb object; defaults to mouse/human based on species.
species	Species name to infer default OrgDb.
background	Optional background gene set; default uses all features.
...	Passed to clusterProfiler::enrichGO().

Value

A list with enrich containing an enrichResult.

Examples

```
## Not run:
v <- example_vista()
comp <- names(comparisons(v))[1]
if (requireNamespace('org.Mm.eg.db', quietly = TRUE)) {
  out <- try(get_go_enrichment(v, sample_comparison = comp, ont = 'BP', from_type = 'ENSEMBL',
                              orgdb = org.Mm.eg.db::org.Mm.eg.db), silent = TRUE)
  if (!inherits(out, 'try-error')) out
}

## End(Not run)
```

get_gsea

Gene set enrichment analysis (GSEA) from a VISTA comparison

Description

Gene set enrichment analysis (GSEA) from a VISTA comparison

Usage

```
get_gsea(
  x,
  sample_comparison,
  set_type = c("msigdb", "go", "kegg"),
  from_type = "SYMBOL",
  orgdb = NULL,
  species = "Mus musculus",
  msigdb_category = "H",
  msigdb_subcategory = NULL,
  ...
)
```

Arguments

x A VISTA object with DE results.
sample_comparison Comparison name to use.
set_type One of "msigdb", "go", or "kegg" selecting the gene set source.
from_type Identifier type in the DE tables (default "SYMBOL").
orgdb OrgDb object; defaults to mouse/human based on species.
species Species name to infer default OrgDb.
msigdb_category, msigdb_subcategory Passed to msigdb::msigdb() when set_type = "msigdb".
... Additional arguments forwarded to the underlying GSEA function: clusterProfiler::GSEA() (msigdb TERM2GENE), gseGO(), or gseKEGG() depending on set_type.

Value

An object returned by this function.

Examples

```
## Not run:
v <- example_vista()
comp <- names(comparisons(v))[1]
if (requireNamespace('msigdb', quietly = TRUE)) {
  out <- try(get_gsea(v, sample_comparison = comp, set_type = 'msigdb', from_type = 'ENSEMBL', species = 'Homo sapiens'), silent = TRUE)
  if (!inherits(out, 'try-error')) out
}

## End(Not run)
```

get_kegg_enrichment	<i>Run KEGG enrichment directly from a VISTA comparison</i>
---------------------	---

Description

Run KEGG enrichment directly from a VISTA comparison

Usage

```
get_kegg_enrichment(
  x,
  sample_comparison,
  regulation = c("Up", "Down", "Both", "All"),
  from_type = "SYMBOL",
  orgdb = NULL,
  species = "Mus musculus",
  kegg_species = NULL,
```

```

    background = NULL,
    ...
)

```

Arguments

<code>x</code>	A VISTA object with DE results.
<code>sample_comparison</code>	Comparison name to use.
<code>regulation</code>	One of "Up", "Down", "Both", or "All"; selects genes.
<code>from_type</code>	Identifier type in the DE tables (default "SYMBOL").
<code>orgdb</code>	OrgDb object; defaults to mouse/human based on species.
<code>species</code>	Species name to infer default OrgDb.
<code>kegg_species</code>	KEGG organism code (e.g., "mmu" or "hsa"). If NULL, inferred from species.
<code>background</code>	Optional background gene set; default uses all features.
<code>...</code>	Passed to <code>clusterProfiler::enrichGO()</code> .

Value

An object returned by this function.

Examples

```

v <- example_vista()
comp <- names(comparisons(v))[1]
if (requireNamespace('org.Mm.eg.db', quietly = TRUE)) {
  out <- try(get_kegg_enrichment(v, sample_comparison = comp, from_type = 'ENSEMBL', orgdb = org.Mm.eg.db::org.Mm.eg.db), silent = TRUE)
  if (!inherits(out, 'try-error')) out
}

```

get_ma_plot

Generate MA plot from a VISTA object

Description

Create an MA plot (log2 fold change vs mean expression) for a selected comparison contained in a VISTA object. Genes are coloured by their regulation class and the top results can be optionally labeled with gene IDs.

Usage

```
get_ma_plot(
  x,
  sample_comparison,
  point_size = 1.2,
  alpha = 0.6,
  colors = c(Up = "#a40000", Down = "#16317d", Other = "gray70"),
  label_n = 0,
  label_size = 3,
  repair_genes = FALSE,
  display_id = NULL,
  display_from = NULL,
  display_orgdb = NULL
)
```

Arguments

<code>x</code>	A VISTA object.
<code>sample_comparison</code>	Character scalar naming the comparison to plot. Must match one of <code>names(comparisons(x))</code> .
<code>point_size</code>	Numeric point size. Default: 1.2.
<code>alpha</code>	Numeric transparency (0-1). Default: 0.6.
<code>colors</code>	Named character vector of colors for "Up", "Down", and "Other" genes.
<code>label_n</code>	Integer number of genes to label. Default: 0.
<code>label_size</code>	Text size for labels. Default: 3.
<code>repair_genes</code>	Logical; if TRUE, attempt to shorten gene identifiers to symbols by stripping prefixes. Default: FALSE.
<code>display_id</code>	Optional ID/column name to use for labels. If supplied and present in <code>rowData(x)</code> , those values are used; otherwise falls back to ID mapping.
<code>display_from</code>	Optional source ID type for mapping (used when <code>display_id</code> is not found in <code>rowData</code>).
<code>display_orgdb</code>	Optional OrgDb used for ID mapping when <code>display_id</code> is set but not found in <code>rowData</code> .

Value

A [ggplot](#) MA plot.

Examples

```
v <- example_vista()
p <- get_ma_plot(v, sample_comparison = names(comparisons(v))[1])
print(p)
```

get_mds_plot

*Generate an MDS plot for samples in a VISTA object***Description**

Runs classical multidimensional scaling on normalized counts, optionally restricting to groups or genes.

Usage

```
get_mds_plot(
  x,
  sample_group = NULL,
  group_column = NULL,
  genes = NULL,
  top_n_genes = NULL,
  label = FALSE,
  label_size = 3,
  point_size = 10,
  shape_by = NULL,
  shape_values = NULL,
  color_by = NULL,
  use_vista_colors = NULL,
  palette = NULL,
  colors = NULL,
  use_group_colors = TRUE
)
```

Arguments

x	A VISTA object.
sample_group	Optional character vector of groups to include (based on the column specified by group_column).
group_column	Optional column name in sample_info to use for grouping/filtering.
genes	Optional character vector of gene identifiers to restrict the matrix.
top_n_genes	Optional integer selecting the top variable genes to include.
label	Logical; draw sample labels when TRUE.
label_size	Numeric size of sample labels when label = TRUE.
point_size	Numeric size for points.
shape_by	Optional column name in sample_info used to map point shape. When NULL, shapes are not mapped.
shape_values	Optional vector of shapes passed to scale_shape_manual() when shape_by is set. Use a named vector to map shapes to specific levels.

color_by	Optional column name in sample_info used for point colour. Defaults to the active grouping column.
use_vista_colors	Deprecated alias for use_group_colors. When supplied, it overrides use_group_colors.
palette	Optional qualitative palette name used when generating colours for non-group metadata levels.
colors	Optional named character vector of manual colours overriding both palette and stored VISTA colours.
use_group_colors	Logical; when TRUE, prefer the stored VISTA group colours when colouring by the grouping column.

Value

An object returned by this function.

Examples

```
v <- example_vista()
p <- get_mds_plot(v)
print(p)
```

get_msigdb_enrichment *Run MSigDB enrichment directly from a VISTA comparison*

Description

Convenience wrapper that pulls regulated genes from a stored differential expression comparison in a VISTA object and runs `enrichMsigDB()` on them.

Usage

```
get_msigdb_enrichment(
  x,
  sample_comparison,
  regulation = c("Up", "Down", "Both", "All"),
  from_type = "SYMBOL",
  orgdb,
  msigdb_category = "H",
  msigdb_subcategory = NULL,
  species = "Mus musculus",
  background = NULL,
  col_genetype = "GENETYPE",
  feature_type = "protein-coding",
  ...
)
```

Arguments

x	A VISTA object with DE results in <code>metadata(x)\$de_results</code> .
sample_comparison	Character scalar naming the comparison to use.
regulation	One of "Up", "Down", "Both", or "All"; selects which genes to send to enrichment.
from_type	Identifier type of the genes in the DE table (passed to <code>enrichMsigDB()</code> , default "SYMBOL"). Ensembl versions are stripped automatically.
orgdb	An OrgDb object used for ID conversion (passed through). If omitted, the default is chosen from species (mouse/human).
msigdb_category	MSigDB category (e.g., "H", "C2", "C5"). Default "H".
msigdb_subcategory	Optional MSigDB sub-collection. Default NULL.
species	Species name for MSigDB (default "Mus musculus").
background	Optional background gene set (passed to <code>enrichMsigDB()</code>). Default NULL uses all features in x (optionally filtered by <code>feature_type</code>).
col_genetype	Column in <code>rowData(x)</code> used to filter background by gene type. Default "GENETYPE".
feature_type	Gene type to retain in the background when filtering. Default "protein-coding".
...	Additional arguments forwarded to <code>enrichMsigDB()</code> .

Value

A list with `enrich` containing an `enrichResult`.

Examples

```
if (requireNamespace("msigdb", quietly = TRUE)) {
  vista <- example_vista()
  comp <- names(comparisons(vista))[1]
  msig <- get_msigdb_enrichment(
    vista,
    sample_comparison = comp,
    regulation = "Both",
    msigdb_category = "H",
    from_type = "ENSEMBL"
  )
  class(msig$enrich)
}
```

```
# Create VISTA object
data("count_data", package = "VISTA")
data("sample_metadata", package = "VISTA")

vista <- create_vista(
  counts = count_data[seq_len(200), ],
```

```

sample_info = sample_metadata[seq_len(6), ],
column_geneid = "gene_id",
group_column = "cond_long",
group_numerator = "treatment1",
group_denominator = "control"
)
comp <- names(comparisons(vista))[1]

# Run MSigDB enrichment on upregulated genes
msig_up <- get_msigdb_enrichment(
  vista,
  sample_comparison = comp,
  regulation = "Up",
  msigdb_category = "H", # Hallmark gene sets
  from_type = "ENSEMBL"
)

if (!is.null(msig_up$enrich)) {
  # View results
  head(msig_up$enrich)
  # Visualize enrichment
  get_enrichment_plot(msig_up$enrich)
}

# Enrichment for downregulated genes
msig_down <- get_msigdb_enrichment(
  vista,
  sample_comparison = comp,
  regulation = "Down",
  msigdb_category = "C2" # Curated gene sets
)

```

```
get_pairwise_corr_plot
```

Plot pairwise correlations between samples

Description

Uses GGally::ggpairs on normalized expression to display correlations among samples from selected groups/genes.

Usage

```

get_pairwise_corr_plot(
  x,
  sample_group = NULL,
  group_column = NULL,
  genes = NULL,

```

```

    sample_order = c("input", "group"),
    value_transform = c("log2", "none"),
    title = "Pairwise Sample Correlations"
  )

```

Arguments

<code>x</code>	A VISTA object.
<code>sample_group</code>	Optional character vector of groups (from the column specified by <code>group_column</code>) used to subset samples.
<code>group_column</code>	Optional column name in <code>sample_info</code> defining the grouping used for filtering.
<code>genes</code>	Optional character vector of gene IDs to include; defaults to all genes.
<code>sample_order</code>	Ordering for selected samples: "input" keeps the current order, while "group" sorts by <code>group_column</code> .
<code>value_transform</code>	Value transformation: "log2" (default) or "none".
<code>title</code>	Plot title.

Value

An object returned by this function.

Examples

```

v <- example_vista()
p <- get_pairwise_corr_plot(v)
print(p)

```

<code>get_pathway_genes</code>	<i>Extract genes from enriched pathways</i>
--------------------------------	---

Description

Parses pathway-level gene members from an enrichment result and returns either a long table, pathway-indexed list, or a unique vector of genes.

Usage

```

get_pathway_genes(
  x,
  pathways = NULL,
  top_n = 10,
  pathway_column = c("Description", "ID"),
  gene_column = c("auto", "geneID", "core_enrichment"),
  gene_sep = "/",
  return_type = c("long", "list", "vector")
)

```

Arguments

<code>x</code>	An <code>enrichResult</code> / <code>gseaResult</code> , or a list containing element <code>enrich</code> (e.g. output from <code>get_msigdb_enrichment()</code>).
<code>pathways</code>	Optional character vector of pathway names to keep. Matches against <code>pathway_column</code> .
<code>top_n</code>	Number of top pathways to use when <code>pathways</code> is <code>NULL</code> . Ranking uses <code>p.adjust</code> (then <code>pvalue</code>) when available. Default: 10.
<code>pathway_column</code>	Which enrichment column to match pathway names against: "Description" (default) or "ID".
<code>gene_column</code>	Which column stores pathway members. "auto" (default) uses "geneID" when present, otherwise "core_enrichment".
<code>gene_sep</code>	Delimiter used in pathway gene strings (default "/").
<code>return_type</code>	One of "long", "list", or "vector".

Value

Depending on `return_type`:

- "long": data frame with `pathway_id`, `pathway`, and `gene`.
- "list": named list of character vectors (genes per pathway).
- "vector": unique character vector of genes.

Examples

```
if (requireNamespace("msigdbR", quietly = TRUE)) {
  vista <- example_vista()
  msig <- get_msigdb_enrichment(
    vista,
    sample_comparison = names(comparisons(vista))[1],
    regulation = "Both",
    msigdb_category = "H",
    from_type = "ENSEMBL"
  )
  pathway_tbl <- get_pathway_genes(msig, top_n = 5, return_type = "long")
  head(pathway_tbl)
}
```

```
data("count_data", package = "VISTA")
data("sample_metadata", package = "VISTA")
```

```
vista <- create_vista(
  counts = count_data,
  sample_info = sample_metadata,
  column_geneid = "gene_id",
  group_column = "cond_long",
  group_numerator = "treatment1",
  group_denominator = "control"
)
```

```

msig <- get_msigdb_enrichment(
  vista,
  sample_comparison = names(comparisons(vista))[1],
  regulation = "Up",
  species = "Homo sapiens",
  from_type = "ENSEMBL"
)

pathway_tbl <- get_pathway_genes(msig, top_n = 5, return_type = "long")
head(pathway_tbl)

```

get_pathway_heatmap *Plot pathway-specific expression heatmaps from enrichment output*

Description

This wrapper bridges enrichment results and expression heatmaps. It extracts genes from selected pathways (via [get_pathway_genes\(\)](#)), maps them to the VISTA feature IDs, and forwards to [get_expression_heatmap\(\)](#).

Usage

```

get_pathway_heatmap(
  x,
  enrichment,
  sample_group = NULL,
  pathways = NULL,
  top_n = 5,
  pathway_column = c("Description", "ID"),
  gene_column = c("auto", "geneID", "core_enrichment"),
  gene_sep = "/",
  gene_mode = c("union", "intersection"),
  gene_id_column = NULL,
  max_genes = NULL,
  return_type = c("heatmap", "both", "genes"),
  ...
)

```

Arguments

x	A VISTA object.
enrichment	An enrichResult/gseaResult, or a list with element enrich as returned by get_*_enrichment() helpers.
sample_group	Character vector of group labels to include (same semantics as get_expression_heatmap()).

pathways	Optional pathway names to include. When NULL, top pathways are selected using top_n.
top_n	Number of top pathways used when pathways = NULL. Default: 5.
pathway_column	Pathway matching column, "Description" (default) or "ID".
gene_column	Pathway gene-member column. "auto" uses "geneID" or "core_enrichment" based on availability.
gene_sep	Delimiter used to parse pathway gene strings (default "/").
gene_mode	How to combine pathway genes for plotting: "union" (default) or "intersection".
gene_id_column	Optional column in rowData(x) used to map enrichment genes back to VISTA rownames (e.g., "SYMBOL" or "ENTREZID"). Leave NULL when enrichment genes already match VISTA rownames.
max_genes	Optional cap on the number of genes passed to the heatmap.
return_type	One of "heatmap" (default), "both", or "genes".
...	Additional arguments passed to get_expression_heatmap() .

Value

Depending on return_type:

- "heatmap": a ComplexHeatmap object from [get_expression_heatmap\(\)](#).
- "both": list with heatmap, genes, and pathway_genes.
- "genes": character vector of mapped genes selected for plotting.

Examples

```
if (requireNamespace("msigdb", quietly = TRUE)) {
  vista <- example_vista()
  msig <- get_msigdb_enrichment(
    vista,
    sample_comparison = names(comparisons(vista))[1],
    regulation = "Both",
    msigdb_category = "H",
    from_type = "ENSEMBL"
  )
  genes <- get_pathway_heatmap(
    vista,
    enrichment = msig,
    sample_group = c("control", "treatment1"),
    top_n = 3,
    return_type = "genes"
  )
  head(genes)
}
```

```
data("count_data", package = "VISTA")
data("sample_metadata", package = "VISTA")
```

```
vista <- create_vista(
  counts = count_data,
  sample_info = sample_metadata,
  column_geneid = "gene_id",
  group_column = "cond_long",
  group_numerator = "treatment1",
  group_denominator = "control"
)

msig <- get_msigdb_enrichment(
  vista,
  sample_comparison = names(comparisons(vista))[1],
  regulation = "Up",
  species = "Homo sapiens",
  from_type = "ENSEMBL"
)

get_pathway_heatmap(
  vista,
  enrichment = msig,
  sample_group = c("control", "treatment1"),
  top_n = 3,
  value_transform = "zscore",
  annotate_columns = TRUE,
  summarise_replicates = FALSE
)
```

get_pca_plot

PCA plot

Description

Uses normalized counts to compute principal components and plot samples, optionally restricting to selected groups or genes.

Usage

```
get_pca_plot(
  x,
  sample_group = NULL,
  group_column = NULL,
  genes = NULL,
  top_n_genes = NULL,
  label = FALSE,
  label_size = 3,
  point_size = 10,
  shape_by = NULL,
```

```

    shape_values = NULL,
    sample.seed = 123,
    show_clusters = FALSE,
    color_by = NULL,
    use_vista_colors = NULL,
    palette = NULL,
    colors = NULL,
    use_group_colors = TRUE
  )

```

Arguments

<code>x</code>	A VISTA object containing normalized counts.
<code>sample_group</code>	Optional character vector of group labels (taken from the column specified by <code>group_column</code> , defaulting to the stored grouping column) used to subset samples prior to PCA. Use <code>NULL</code> to include all samples.
<code>group_column</code>	Optional column name in <code>sample_info</code> to use for grouping. Defaults to the stored grouping column.
<code>genes</code>	Optional character vector of gene identifiers to restrict the PCA input matrix. When <code>NULL</code> , all genes are used.
<code>top_n_genes</code>	Optional integer selecting the top most variable genes to include. Ignored when <code>genes</code> is supplied.
<code>label</code>	Logical; if <code>TRUE</code> , sample names are drawn next to the points.
<code>label_size</code>	Numeric size of sample labels when <code>label = TRUE</code> .
<code>point_size</code>	Numeric size of the plotted points.
<code>shape_by</code>	Optional column name in <code>sample_info</code> used to map point shape. When <code>NULL</code> , shapes are not mapped.
<code>shape_values</code>	Optional vector of shapes passed to <code>scale_shape_manual()</code> when <code>shape_by</code> is set. Use a named vector to map shapes to specific levels.
<code>sample.seed</code>	Deprecated/unused; retained for backward compatibility.
<code>show_clusters</code>	Logical; add normal ellipses per group when <code>TRUE</code> .
<code>color_by</code>	Optional column name in <code>sample_info</code> used to map point colour. Defaults to the active grouping column.
<code>use_vista_colors</code>	Deprecated alias for <code>use_group_colors</code> . When supplied, it overrides <code>use_group_colors</code> .
<code>palette</code>	Optional qualitative palette name used when generating colours for non-group metadata levels.
<code>colors</code>	Optional named character vector of manual colours overriding both <code>palette</code> and stored VISTA colours.
<code>use_group_colors</code>	Logical; when <code>TRUE</code> , prefer the stored VISTA group colours when colouring by the grouping column.

Value

A ggplot object showing the first two PCs.

Examples

```
# Create VISTA object
data("count_data", package = "VISTA")
data("sample_metadata", package = "VISTA")

vista <- create_vista(
  counts = count_data[seq_len(200), ],
  sample_info = sample_metadata[seq_len(6), ],
  column_geneid = "gene_id",
  group_column = "cond_long",
  group_numerator = "treatment1",
  group_denominator = "control"
)

# Basic PCA plot
get_pca_plot(vista)

# With sample labels
get_pca_plot(vista, label = TRUE)

# Using top variable genes
get_pca_plot(vista, top_n_genes = 100)

# With confidence ellipses
get_pca_plot(vista, show_clusters = TRUE)
```

get_umap_plot

*Generate a UMAP plot for samples in a VISTA object***Description**

Runs UMAP on normalized counts, optionally restricting to selected groups or genes. UMAP is intended for exploratory sample-level structure.

Usage

```
get_umap_plot(
  x,
  sample_group = NULL,
  group_column = NULL,
  color_by = NULL,
  genes = NULL,
  top_n_genes = NULL,
  label = FALSE,
  label_size = 3,
  point_size = 10,
  shape_by = NULL,
```

```

    shape_values = NULL,
    n_neighbors = 15,
    min_dist = 0.1,
    metric = "euclidean",
    seed = 123,
    use_vista_colors = NULL,
    palette = NULL,
    colors = NULL,
    use_group_colors = TRUE
  )

```

Arguments

<code>x</code>	A VISTA object.
<code>sample_group</code>	Optional character vector of groups to include (based on <code>group_column</code>).
<code>group_column</code>	Optional column name in <code>sample_info</code> used for filtering/grouping. Defaults to the stored grouping column.
<code>color_by</code>	Optional column name in <code>sample_info</code> used for point color. Defaults to <code>group_column</code> .
<code>genes</code>	Optional character vector of gene identifiers to restrict the matrix.
<code>top_n_genes</code>	Optional integer selecting top variable genes to include.
<code>label</code>	Logical; draw sample labels when TRUE.
<code>label_size</code>	Numeric label size when <code>label = TRUE</code> .
<code>point_size</code>	Numeric point size.
<code>shape_by</code>	Optional column name in <code>sample_info</code> used to map point shape.
<code>shape_values</code>	Optional vector passed to <code>scale_shape_manual()</code> when <code>shape_by</code> is set.
<code>n_neighbors</code>	UMAP <code>n_neighbors</code> parameter.
<code>min_dist</code>	UMAP <code>min_dist</code> parameter.
<code>metric</code>	UMAP distance metric.
<code>seed</code>	Integer random seed passed to UMAP.
<code>use_vista_colors</code>	Deprecated alias for <code>use_group_colors</code> . When supplied, it overrides <code>use_group_colors</code> .
<code>palette</code>	Optional qualitative palette name used when generating colours for non-group metadata levels.
<code>colors</code>	Optional named character vector of manual colours overriding both <code>palette</code> and stored VISTA colours.
<code>use_group_colors</code>	Logical; when TRUE, prefer the stored VISTA group colours when colouring by the grouping column.

Value

A ggplot object with UMAP1/UMAP2 coordinates.

Examples

```
if (requireNamespace("uwot", quietly = TRUE)) {
  vista <- example_vista()
  get_umap_plot(vista, top_n_genes = 50)
}
```

get_volcano_plot	<i>Generate a volcano plot for a comparison in a VISTA object</i>
------------------	---

Description

Wraps EnhancedVolcano to visualize log2FC vs p-values for a selected comparison.

Usage

```
get_volcano_plot(
  x,
  sample_comparison,
  log2fc_cutoff = 1,
  pval_cutoff = 0.05,
  label_genes = NULL,
  label_size = 3,
  point_size = 1,
  colors = c(Up = "#a40000", Down = "#007e2f", Other = "grey"),
  repair_genes = TRUE,
  display_id = NULL,
  display_from = NULL,
  display_orgdb = NULL,
  ...
)
```

Arguments

x	A VISTA object containing differential expression results.
sample_comparison	Character scalar naming the comparison to display.
log2fc_cutoff	Numeric absolute log2 fold-change threshold used to color significant points.
pval_cutoff	Numeric p-value threshold used to color significant points.
label_genes	Optional character vector of gene identifiers to force-label.
label_size	Numeric label text size.
point_size	Numeric point size.
colors	Named colour vector with entries for "Up", "Down", and "Other".
repair_genes	Logical; when TRUE, split gene_id values like ID:SYMBOL to display the symbol.
display_id	Optional ID/column name to use for plot labels. If supplied and present in rowData(x), those values are used; otherwise falls back to ID mapping.

<code>display_from</code>	Optional source ID type for mapping (used when <code>display_id</code> is not found in <code>rowData</code>).
<code>display_orgdb</code>	Optional <code>OrgDb</code> object used for ID mapping when <code>display_id</code> is set but not found in <code>rowData</code> .
<code>...</code>	Additional parameters forwarded to <code>EnhancedVolcano::EnhancedVolcano()</code> .

Value

A `ggplot2` object.

Examples

```
vista <- example_vista()
comps <- names(comparisons(vista))
get_volcano_plot(vista, sample_comparison = comps[1])

# Create VISTA object
data("count_data", package = "VISTA")
data("sample_metadata", package = "VISTA")

vista <- create_vista(
  counts = count_data,
  sample_info = sample_metadata,
  column_geneid = "gene_id",
  group_column = "cond_long",
  group_numerator = "treatment1",
  group_denominator = "control"
)

# Basic volcano plot
comps <- names(comparisons(vista))
get_volcano_plot(vista, sample_comparison = comps[1])

# With custom thresholds
get_volcano_plot(
  vista,
  sample_comparison = comps[1],
  log2fc_cutoff = 1.5,
  pval_cutoff = 0.01
)

# Highlight specific genes
genes_of_interest <- rownames(vista)[seq_len(5)]
get_volcano_plot(
  vista,
  sample_comparison = comps[1],
  label_genes = genes_of_interest
)
```

match_vista_inputs	<i>Match count and metadata inputs for VISTA</i>
--------------------	--

Description

match_vista_inputs() aligns standardized counts and sample metadata so they can be passed directly to create_vista(). It accepts the raw output from read_vista_counts() or a count data frame/matrix plus sample metadata.

Usage

```
match_vista_inputs(
  counts,
  sample_info,
  column_geneid = NULL,
  sample_column = NULL,
  reorder = TRUE,
  drop_unmatched = FALSE,
  verbose = TRUE
)
```

Arguments

counts	Standardized counts from read_vista_counts(), or a compatible count matrix/data frame.
sample_info	Sample metadata from read_vista_metadata() or a data frame coercible to that format.
column_geneid	Optional gene identifier column for raw tabular counts. Ignored when counts is the list output of read_vista_counts().
sample_column	Optional sample identifier column in sample_info.
reorder	Logical; if TRUE (default), reorder sample_info to match count columns.
drop_unmatched	Logical; if TRUE, keep only the intersection of count samples and metadata samples. Otherwise mismatches raise an error.
verbose	Logical; print an informational alignment summary.

Value

A list with standardized counts, aligned sample_info, column_geneid, sample_names, sample_name_map, row_data, and a small report.

Examples

```
data("count_data", package = "VISTA")
data("sample_metadata", package = "VISTA")

cnt <- read_vista_counts(
```

```

count_data[seq_len(25), ],
format = "matrix",
gene_id_column = "gene_id",
verbose = FALSE
)
si <- read_vista_metadata(
  sample_metadata[sample_metadata$sample_names %in% cnt$sample_names, ],
  verbose = FALSE
)
matched <- match_vista_inputs(cnt, si, verbose = FALSE)

matched$column_geneid
identical(matched$sample_info$sample_names, colnames(matched$counts)[-1])

```

`print.VISTA`*Print a VISTA object like a SummarizedExperiment*

Description

Forwards to `SummarizedExperiment`'s `show()` so the output is identical to a plain SE. Invisibly returns `x`.

Usage

```
## S3 method for class 'VISTA'
print(x, ...)
```

Arguments

`x` A VISTA object.

`...` Ignored.

Value

The input object `x`, returned invisibly.

Examples

```
v <- example_vista()
print(v)
```

read_vista_counts	<i>Read and standardize count inputs for VISTA</i>
-------------------	--

Description

read_vista_counts() helps standardize common RNA-seq count inputs into a count table that can be passed directly to create_vista(). It supports plain matrices/data frames, featureCounts outputs, STAR gene counts, HTSeq-count outputs, tximport-like lists, and RSEM gene result files.

Usage

```
read_vista_counts(
  x,
  format = c("auto", "matrix", "featurecounts", "star", "htseq", "tximport", "rsem"),
  gene_id_column = NULL,
  sample_columns = NULL,
  sample_names = NULL,
  annotation_columns = NULL,
  count_column = NULL,
  tx2gene = NULL,
  counts_from = c("counts", "abundance", "length"),
  drop_technical = TRUE,
  remove_special_rows = TRUE,
  make_unique_ids = FALSE,
  repair_sample_names = c("auto", "none"),
  return_type = c("list", "data.frame", "matrix"),
  verbose = TRUE
)
```

Arguments

x	Count input. Supported values depend on format and include a matrix, data frame, single file path, vector of file paths, or a tximport-like list with counts, abundance, and/or length.
format	Input format. One of "auto", "matrix", "featurecounts", "star", "htseq", "tximport", or "rsem".
gene_id_column	Optional gene identifier column in tabular inputs. When omitted, VISTA uses common names such as gene_id/Geneid, or falls back to rownames for matrices/data frames with unique rownames.
sample_columns	Optional character vector of sample count columns to retain from tabular inputs.
sample_names	Optional sample names to use when x is a vector of per-sample files.
annotation_columns	Optional feature annotation columns to retain in the returned row_data.
count_column	Optional count column selector for formats that expose multiple count choices. For STAR, use one of "unstranded", "stranded_first", or "stranded_second". For RSEM, this defaults to "expected_count".

tx2gene	Optional two-column mapping used to summarize transcript-level tximport-like inputs to genes. The first column should contain transcript IDs and the second column gene IDs.
counts_from	Which matrix to extract from a tximport-like input: "counts", "abundance", or "length".
drop_technical	Logical; when TRUE, drop known technical summary rows from STAR/HTSeq inputs.
remove_special_rows	Logical; alias for drop_technical, retained for clarity in file-based imports.
make_unique_ids	Logical; if TRUE, duplicate gene IDs are repaired with <code>make.unique()</code> . Otherwise duplicated gene IDs raise an error.
repair_sample_names	Strategy for repairing sample column names. "auto" (default) strips common file-path and alignment/count suffixes when the repaired names are unique, while "none" leaves sample columns unchanged. In automatic mode VISTA currently: • strips directory paths to the basename • uses the parent directory for generic quantification files such as <code>quant.sf</code> or <code>abundance.tsv</code> • removes common RNA-seq output suffixes such as <code>Aligned.sortedByCoord.out.bam</code>, <code>ReadsPerGene.out.tab</code>, <code>.genes.results</code>, <code>.isoforms.results</code>, <code>.bam</code>, and <code>.fastq.gz</code> • removes common lane/read suffixes such as <code>_S1_L001_R1_001</code>, <code>_L001_R2_001</code>, <code>_R1</code>, and <code>_R2</code> Repaired names are only applied when they remain non-empty and unique. Otherwise VISTA keeps the original count column names and records the unchanged mapping in <code>sample_name_map</code>.
return_type	Return "list" (default), standardized "data.frame", or numeric "matrix".
verbose	Logical; print an informational import summary.

Details

Internally, VISTA uses a format-specific importer for each supported input type, then normalizes the result into a common structure with:

- a count table with a `gene_id` column plus sample columns
- optional feature metadata in `row_data`
- sample names inferred from columns or file names
- an auditable `sample_name_map` showing original and repaired names

Value

If `return_type = "list"`, a list with:

counts A standardized count table with `gene_id` plus sample columns.

row_data Feature metadata aligned to the count table.

column_geneid Always "gene_id" for the standardized output.

sample_names Sample columns in the standardized count table.

sample_name_map A two-column mapping of original and repaired sample names.

input_format Resolved import format.

report Basic import summary.

If `return_type = "data.frame"`, returns the standardized count table. If `return_type = "matrix"`, returns a numeric matrix with gene IDs as rownames.

Examples

```
data("count_data", package = "VISTA")

cnt <- read_vista_counts(
  count_data[seq_len(25), ],
  format = "matrix",
  gene_id_column = "gene_id"
)

head(cnt$counts[, seq_len(4)])
cnt$sample_names
```

read_vista_metadata	<i>Read and standardize sample metadata for VISTA</i>
---------------------	---

Description

`read_vista_metadata()` standardizes a sample sheet for use as `sample_info` in `create_vista()`. It infers or creates the required `sample_names` column using the same conventions VISTA already accepts in the constructor.

Usage

```
read_vista_metadata(
  x,
  sample_column = NULL,
  required_columns = NULL,
  drop_empty = TRUE,
  standardize_names = TRUE,
  verbose = TRUE
)
```

Arguments

x	Sample metadata as a data frame or file path.
sample_column	Optional column to use as sample_names. If omitted, VISTA uses an existing sample_names column, non-default rownames, or common aliases such as sample, sample_id, or Run.
required_columns	Optional character vector of columns that must be present after import.
drop_empty	Logical; if TRUE, remove columns that are entirely NA or empty strings.
standardize_names	Logical; if TRUE, coerce the final sample_names column to character and set rownames to match it.
verbose	Logical; print an informational import summary.

Value

A data frame suitable for use as sample_info in create_vista().

Examples

```
data("sample_metadata", package = "VISTA")

si <- read_vista_metadata(sample_metadata[seq_len(6), ])
head(si$sample_names)
```

run_cell_deconvolution

Run Cell Deconvolution on Bulk RNA-seq from VISTA Object

Description

Estimates cell-type proportions in bulk RNA-seq using single-cell reference or xCell2.

Usage

```
run_cell_deconvolution(
  x,
  method = c("xCell2"),
  single_cell_reference = NULL,
  reference_labels = NULL,
  gene_id_type = c("auto", "symbol", "ensembl", "ensembl_symbol"),
  xcell2_reference = NULL,
  xcell2_min_shared_genes = NULL,
  transform = c("log2", "raw"),
  ...
)
```

Arguments

x	A VISTA object.
method	Deconvolution method. Currently only "xCell2" is supported.
single_cell_reference	Reserved for future reference-based methods (ignored).
reference_labels	Reserved for future reference-based methods (ignored).
gene_id_type	Type of gene identifiers: "auto", "symbol", "ensembl", or "ensembl_symbol".
xcell2_reference	Optional xCell2 reference object or dataset name (e.g., "DICE_demo.xCell2Ref"). Used when xCell2 exposes xCell2Analysis().
xcell2_min_shared_genes	Optional numeric shortcut for xCell2's minSharedGenes argument (when supported by the installed xCell2 API).
transform	Expression transformation: "log2" or "raw".
...	Additional arguments passed to the specific method.

Value

VISTA object with cell_fractions added to metadata.

Examples

```
v <- example_vista()
if (requireNamespace('xCell2', quietly = TRUE)) {
  out <- try(run_cell_deconvolution(v, method = 'xCell2'), silent = TRUE)
  if (!inherits(out, 'try-error')) out
}
```

run_deseq_analysis	<i>Run Differential Expression Analysis with DESeq2, edgeR, or limma-voom</i>
--------------------	---

Description

These functions encapsulate the standard RNA-seq analysis workflow using DESeq2 ([run_deseq_analysis](#)), edgeR ([run_edger_analysis](#)), or limma-voom ([run_limma_analysis](#)), including: gene filtering, design matrix setup, normalization, model fitting, differential testing, DEG classification ("Up", "Down", "Other"), and result formatting.

Both methods return output in a harmonized structure ready for downstream use in [create_vista](#) or standalone DEG summaries.

Usage

```
run_deseq_analysis(  
  counts,  
  sample_info,  
  column_geneid,  
  group_column,  
  group_numerator,  
  group_denominator,  
  covariates = NULL,  
  design_formula = NULL,  
  min_counts = 10,  
  min_replicates = 1,  
  log2fc_cutoff = 1,  
  pval_cutoff = 0.05,  
  p_value_type = "padj"  
)
```

```
run_edger_analysis(  
  counts,  
  sample_info,  
  column_geneid,  
  group_column,  
  group_numerator,  
  group_denominator,  
  covariates = NULL,  
  design_formula = NULL,  
  min_counts = 10,  
  min_replicates = 1,  
  log2fc_cutoff = 1,  
  pval_cutoff = 0.05,  
  p_value_type = "FDR"  
)
```

```
run_limma_analysis(  
  counts,  
  sample_info,  
  column_geneid,  
  group_column,  
  group_numerator,  
  group_denominator,  
  covariates = NULL,  
  design_formula = NULL,  
  min_counts = 10,  
  min_replicates = 1,  
  log2fc_cutoff = 1,  
  pval_cutoff = 0.05,  
  p_value_type = "FDR"  
)
```

Arguments

counts	A data frame or matrix of raw counts with one gene per row. Must include a column defined by column_geneid, and column names must match entries in sample_info\$sample_names.
sample_info	A data frame with sample metadata. Must contain sample_names and the specified grouping column.
column_geneid	A string identifying the column name containing gene identifiers.
group_column	The name of the column in sample_info that defines experimental groups.
group_numerator	A character vector of numerator group(s) for fold-change comparisons.
group_denominator	A character vector of denominator group(s) for fold-change comparisons.
covariates	Optional character vector of additional sample_info columns to adjust for.
design_formula	Optional model formula (or formula string). When provided, it overrides automatic design construction from group_column + covariates. Must include group_column.
min_counts	Minimum total read count across all samples to retain a gene. Default: 10.
min_replicates	Minimum number of replicates within each group that must exceed min_counts. Default: 1.
log2fc_cutoff	Absolute log2 fold-change threshold to define DEGs. Default: 1.
pval_cutoff	P-value or adjusted p-value cutoff for significance. Default: 0.05.
p_value_type	For DESeq2: one of "padj" or "pvalue". For edgeR/limma: one of "FDR" or "PValue".

Details

Perform differential expression (DE) analysis across multiple group comparisons using DESeq2, edgeR, or limma-voom. These functions process raw count data, normalize it, execute pairwise group-level tests, and return standardized DEG outputs compatible with VISTA-based visualization and analysis.

- For DESeq2, normalization is performed via [DESeq](#), and DE testing uses [results](#).
- For edgeR, normalization uses [calcNormFactors](#), and testing uses [glmLRT](#).
- For limma, normalization uses [calcNormFactors](#) + [voom](#), and testing uses [eBayes](#).

Low-abundance filtering is applied before model fitting. Gene regulation status is determined via `.categorize_deg_results()` based on user thresholds.

All output comparison results are internally standardized via `.tidy_de_results()` to ensure a uniform column schema compatible with VISTA plotting tools.

Value

A named list with components:

- `norm_counts`: Matrix of normalized expression values (CPM for edgeR/limma, DESeq2-normalized counts).
- `sample_info`: Updated sample metadata.
- `row_data`: Gene-level metadata, including mean expression.
- `comparisons`: Named list of DEG result tibbles (one per comparison), each containing standardized columns: `gene_id`, `log2fc`, `pvalue`, `p.adj`, and `regulation`.
- `deg_summary`: List of summary tables showing DEG regulation counts.

See Also

[create_vista](#), [DESeq](#), [glmLRT](#), [voom](#)

Examples

```
v <- example_vista()
si <- as.data.frame(sample_info(v))
data("count_data", package = "VISTA")
counts_small <- count_data[seq_len(200), c("gene_id", si$sample_names), drop = FALSE]
limma_results <- run_limma_analysis(
  counts = counts_small,
  sample_info = si,
  column_geneid = "gene_id",
  group_column = "cond_long",
  group_numerator = "treatment1",
  group_denominator = "control",
  min_counts = 5,
  min_replicates = 1
)
names(limma_results$comparisons)
```

run_vista_report

Generate a publication-ready VISTA workflow report

Description

Builds a VISTA object (or uses a precomputed one), computes a comprehensive single-comparison analysis panel, exports publication assets, and renders an automated Quarto HTML/PDF report from YAML-driven parameters.

Usage

```
run_vista_report(config, output_file = "vista-report.html")
```

Arguments

config	Path to a YAML config file (see template at <code>inst/reports/vista-report-template.yml</code>) or a named list of parameters.
output_file	Optional output filename overriding <code>config\$output_file</code> .

Details

The report focuses on one differential comparison (`primary_comparison`) and includes:

- QC plots (PCA, MDS, correlation heatmap),
- DE plots (volcano, MA, DEG bar/pie/donut summaries),
- expression-focused views (boxplot, fold-change barplot, expression heatmap),
- enrichment outputs (MSigDB/GO/KEGG when available),
- downloadable artifacts (tables + plots + optional zip bundle),
- interactive HTML tables via **DT** when installed.

Value

Invisibly, the normalized output report path.

Examples

```
## Not run:
data('count_data', package = 'VISTA')
data('sample_metadata', package = 'VISTA')
cfg <- list(
  counts = count_data[seq_len(100), ],
  sample_info = sample_metadata[seq_len(6), ],
  column_geneid = 'gene_id',
  group_column = 'cond_long',
  group_numerator = 'treatment1',
  group_denominator = 'control',
  include_msigdb = FALSE, include_go = FALSE, include_kegg = FALSE
)
if (requireNamespace('quarto', quietly = TRUE)) {
  out <- tempfile(fileext = '.html')
  try(run_vista_report(cfg, output_file = out), silent = TRUE)
}

## End(Not run)
```

sample_metadata*Sample metadata accompanying the VISTA airway example counts*

Description

Metadata describing each column in `count_data`, including friendly sample names and the experimental group assignment used throughout the vignettes and tests.

Usage

```
data(sample_metadata)
```

Format

A tibble with 8 rows and 13 columns:

sample_names Character identifiers matching the column names in `count_data`.

groups Group labels used in examples ("control", "treatment1").

cond_long Long-form condition labels ("control", "treatment1").

cond_short Short-form condition labels ("CTRL", "TREAT1").

SampleName Original sample label from the airway `colData`.

dex Original airway treatment labels ("untrt"/"trt").

cell Donor/cell-line identifiers from airway.

albut Original airway albuterol indicator.

Run SRA run identifier (also copied into `sample_names`).

avgLength Average transcript length metadata from airway.

Experiment SRA experiment accession from airway.

Sample SRA sample accession from airway.

BioSample NCBI BioSample accession from airway.

Source

Derived from the airway Bioconductor dataset.

See Also

[count_data](#)

save_vista_data	<i>Save VISTA tabular outputs to disk</i>
-----------------	---

Description

Exports selected data components from a VISTA object to CSV/TSV/RDS/XLSX.

Usage

```
save_vista_data(
  x,
  what = c("comparison", "comparisons", "norm_counts", "sample_info", "row_data",
    "deg_summary", "cutoffs"),
  file,
  sample_comparison = NULL,
  format = NULL,
  include_rownames = TRUE
)
```

Arguments

x	A VISTA object.
what	Character vector specifying which object(s) to export. Supported values are "comparison", "comparisons", "norm_counts", "sample_info", "row_data", "deg_summary", and "cutoffs".
file	Output file path.
sample_comparison	Optional comparison name used when what includes "comparison". Defaults to the first comparison in comparisons(x).
format	Output format. One of "csv", "tsv", "rds", "xlsx". If NULL, inferred from file extension.
include_rownames	Logical; include meaningful row identifiers (e.g., gene IDs or sample names) as explicit columns where applicable.

Value

Invisibly, the normalized output file path.

Examples

```
v <- example_vista()
save_vista_data(v, what = "comparison", file = tempfile(fileext = ".csv"), format = "csv")
```

save_vista_plot	<i>Save a VISTA plot object to disk</i>
-----------------	---

Description

Saves plot objects returned by VISTA plotting functions to file. Supports both ggplot-like objects (saved via `ggplot2::ggsave()`) and `ComplexHeatmap` objects (`Heatmap` / `HeatmapList`) saved via graphics devices.

Usage

```
save_vista_plot(
  plot,
  file,
  width = 8,
  height = 6,
  units = "in",
  dpi = 300,
  device = NULL,
  ...
)
```

Arguments

<code>plot</code>	A plot object. Typically <code>ggplot</code> , <code>patchwork</code> , <code>Heatmap</code> , or <code>HeatmapList</code> .
<code>file</code>	Output file path.
<code>width</code>	Plot width.
<code>height</code>	Plot height.
<code>units</code>	Units for width and height. One of "in", "cm", "mm", or "px".
<code>dpi</code>	Resolution for raster outputs.
<code>device</code>	Optional graphics device (e.g. "png", "pdf"). If <code>NULL</code> , inferred from file extension (defaults to "png" when missing).
<code>...</code>	Additional arguments passed to <code>ggplot2::ggsave()</code> for ggplot-like objects.

Value

Invisibly, the normalized output file path.

Examples

```
v <- example_vista()
p <- get_pca_plot(v)
out_file <- tempfile(fileext = ".pdf")
save_vista_plot(p, file = out_file, width = 7, height = 5, units = "in")
```

set_de_source	<i>Set active DE source in a VISTA object</i>
---------------	---

Description

Switches the DE result source used by all downstream VISTA plotting and accessor functions that read the active metadata slots.

Usage

```
set_de_source(  
  object,  
  source = c("deseq2", "edger", "limma", "consensus", "active")  
)
```

Arguments

object	A VISTA object.
source	One of "active", "deseq2", "edger", "limma", or "consensus". When "active", the currently active DE source is kept.

Value

A modified VISTA object with updated active DE source.

Examples

```
v <- example_vista(method = "both")  
v <- set_de_source(v, "edger")  
names(comparisons(v, source = "active"))
```

set_rowdata	<i>Set or append rowData annotations on a VISTA object</i>
-------------	--

Description

Accepts a data.frame/tibble/DataFrame of gene-level annotations, aligns it to the VISTA row order, and stores it in `rowData(x)`. Rows are matched by a key column (default: tries `gene_id` or `rownames`); Ensembl version suffixes can be stripped for matching.

Usage

```
set_rowdata(
  x,
  annotations = NULL,
  orgdb = NULL,
  key_col = NULL,
  keytype = NULL,
  columns = c("SYMBOL", "GENENAME", "ENSEMBL", "ENTREZID", "TXCHROM", "TXSTART", "TXEND"),
  drop_version = TRUE,
  overwrite = FALSE
)
```

Arguments

<code>x</code>	A VISTA object.
<code>annotations</code>	Optional data.frame/tibble/DataFrame with one row per gene and a column containing the gene IDs to match against rownames(<code>x</code>). If omitted, annotations are pulled from <code>orgdb</code> .
<code>orgdb</code>	Optional OrgDb object; when supplied (and <code>annotations</code> is <code>NULL</code>), annotations are retrieved via <code>AnnotationDbi::select()</code> .
<code>key_col</code>	Name of the column in <code>annotations</code> that holds the gene IDs. If <code>NULL</code> , the function will try <code>gene_id</code> , <code>gene</code> , <code>ENSEMBL</code> , <code>SYMBOL</code> , or use <code>rownames(annotations)</code> . Ignored when <code>annotations</code> is <code>NULL</code> and <code>orgdb</code> is used.
<code>keytype</code>	Key type for <code>orgdb</code> lookups (e.g., <code>"ENSEMBL"</code> , <code>"SYMBOL"</code>). If <code>NULL</code> , inferred from <code>rownames(x)</code> (<code>ENSEMBL</code> if they start with <code>"ENS"</code> , otherwise <code>SYMBOL</code>).
<code>columns</code>	Character vector of OrgDb columns to retrieve when using <code>orgdb</code> . Default: <code>c("SYMBOL", "GENENAME", "ENSEMBL", "ENTREZID", "TXCHROM", "TXSTART", "TXEND")</code> . The <code>TXCHROM/TXSTART/TXEND</code> fields carry basic genomic coordinates when available in the OrgDb.
<code>drop_version</code>	Logical; if <code>TRUE</code> , strips Ensembl version suffixes (e.g., <code>.1</code>) from both the VISTA rownames and the key column/keys before matching.
<code>overwrite</code>	Logical; if <code>TRUE</code> , replaces existing rowdata. If <code>FALSE</code> , new columns are appended (overwriting by name when names collide).

Details

OrgDb packages rarely include full genomic coordinates; the default `TXCHROM/TXSTART/TXEND` columns may therefore be `NA` unless your OrgDb provides them. For reliable coordinates, fetch them from an `EnsDb/TxDb` (via `genes()` or `biomaRt/AnnotationHub`), build an annotation table keyed on your gene IDs, and supply that via the `annotations` argument. When fetching from an OrgDb, only columns available in that database will be filled.

Value

The updated VISTA object with rowdata populated/appended.

Examples

```

vista <- example_vista()
custom_annot <- data.frame(
  gene_id = rownames(vista)[seq_len(10)],
  custom_info = paste0("Info_", seq_len(10))
)
vista2 <- set_rowdata(vista, annotations = custom_annot, key_col = "gene_id")
head(SummarizedExperiment::rowData(vista2)$custom_info)

# Load example VISTA object
data("count_data", package = "VISTA")
data("sample_metadata", package = "VISTA")

vista <- create_vista(
  counts = count_data[seq_len(100), ],
  sample_info = sample_metadata[seq_len(6), ],
  column_geneid = "gene_id",
  group_column = "cond_long",
  group_numerator = "treatment1",
  group_denominator = "control"
)

# Add annotations from OrgDb (human)
if (requireNamespace("org.Hs.eg.db", quietly = TRUE)) {
  vista <- set_rowdata(
    vista,
    orgdb = org.Hs.eg.db::org.Hs.eg.db,
    columns = c("SYMBOL", "GENENAME", "ENTREZID")
  )

  # View updated rowData
  head(SummarizedExperiment::rowData(vista))
}

# Or provide custom annotations
custom_annot <- data.frame(
  gene_id = rownames(vista)[seq_len(10)],
  custom_info = paste0("Info_", seq_len(10))
)
vista <- set_rowdata(vista, annotations = custom_annot, key_col = "gene_id")

```

set_vista_comparison_colors

Set manual comparison colors in a VISTA object

Description

Updates `metadata(x)$comparison$colors` using a user-supplied named color vector. Comparison colors are used in fold-change and comparison-level plots. The color map must include all currently active comparisons.

Usage

```
set_vista_comparison_colors(object, color_map)
```

Arguments

<code>object</code>	A VISTA object.
<code>color_map</code>	Named character vector of colors, with names equal to comparison names (for example "A_VS_B").

Value

A modified VISTA object with updated comparison colors.

Examples

```
v <- example_vista()
comps <- names(comparisons(v))
if (length(comps)) {
  cmap <- stats::setNames(rep('#1b9e77', length(comps)), comps)
  set_vista_comparison_colors(v, cmap)
}
```

```
set_vista_group_colors
```

Set manual group colors in a VISTA object

Description

Updates `metadata(x)$group$colors` using a user-supplied named color vector. This controls group-level coloring across VISTA plots that use group mapping (for example PCA, MDS, and expression plots). The color map must include all groups currently present in the object.

Usage

```
set_vista_group_colors(object, color_map)
```

Arguments

<code>object</code>	A VISTA object.
<code>color_map</code>	Named character vector of colors, with names equal to group labels.

Value

A modified VISTA object with updated group colors.

Examples

```
v <- example_vista()
groups <- unique(as.character(sample_info(v)$cond_long))
gmap <- stats::setNames(c('#1b9e77', '#d95f02')[seq_along(groups)], groups)
set_vista_group_colors(v, gmap)
```

validate_vista	<i>Validate a VISTA object</i>
----------------	--------------------------------

Description

Performs structural and metadata contract checks on a VISTA object. Use `level = "core"` for invariants that all VISTA objects must satisfy, or `level = "full"` to additionally validate method-specific metadata used by advanced workflows.

Usage

```
validate_vista(x, level = c("core", "full"), error = TRUE)
```

Arguments

<code>x</code>	A VISTA object.
<code>level</code>	Validation depth. One of "core" or "full".
<code>error</code>	Logical; if TRUE, aborts on validation failures. If FALSE, returns the validation report and emits a warning when issues are found.

Value

Invisibly returns a list with fields `valid`, `level`, and `issues`.

Examples

```
mat <- matrix(rnorm(60), nrow = 10)
rownames(mat) <- paste0("gene", seq_len(nrow(mat)))
colnames(mat) <- paste0("sample", seq_len(ncol(mat)))
se <- SummarizedExperiment::SummarizedExperiment(
  assays = list(norm_counts = mat),
  colData = S4Vectors::DataFrame(
    cond = rep(c("A", "B"), each = 3),
    row.names = colnames(mat)
  ),
  rowData = S4Vectors::DataFrame(
    gene_id = rownames(mat),
    row.names = rownames(mat)
  )
)
```

```

    )
  )
  v <- as_vista(se, group_column = "cond")
  validate_vista(v)

```

validate_vista_deep	<i>Deep validation of VISTA differential-expression fidelity</i>
---------------------	--

Description

This validator combines `validate_vista()` with backend-to-backend numerical equivalence checks against standalone DESeq2, edgeR, and limma runs.

Usage

```

validate_vista_deep(
  counts,
  sample_info,
  column_geneid,
  group_column,
  group_numerator,
  group_denominator,
  methods = c("deseq2", "edgeR", "limma"),
  min_counts = 10,
  min_replicates = 1,
  log2fc_cutoff = 1,
  pval_cutoff = 0.05,
  p_value_type = "padj",
  covariates = NULL,
  design_formula = NULL,
  tolerance = 1e-08,
  return_plots = FALSE,
  error = TRUE
)

```

Arguments

<code>counts</code>	Raw counts (matrix/data.frame) with a gene-id column and sample columns.
<code>sample_info</code>	Data frame with sample metadata.
<code>column_geneid</code>	Column name in counts that contains gene identifiers.
<code>group_column</code>	Column in sample_info used to group samples.
<code>group_numerator</code>	Character vector of numerator groups for pairwise comparisons.
<code>group_denominator</code>	Character vector of denominator groups.

methods	Character vector of backends to benchmark. Any subset of c("deseq2", "edger", "limma").
min_counts	Minimum total counts per gene to retain.
min_replicates	Minimum samples per group meeting filtering criteria.
log2fc_cutoff	Absolute log2 fold-change threshold for DEG calling.
pval_cutoff	P-value (or adjusted p-value) threshold.
p_value_type	Either "padj" or "pvalue".
covariates	Optional character vector of additional sample_info columns.
design_formula	Optional model formula (or formula string) including group_column.
tolerance	Numeric tolerance used for floating-point comparisons.
return_plots	Logical; if TRUE, return paired VISTA/reference plots for MA, volcano, DEG count, and PCA views.
error	Logical; if TRUE, abort when any discrepancy is detected.

Value

Invisibly returns the full benchmark report.

Examples

```
v <- example_vista()
si <- as.data.frame(sample_info(v))
data("count_data", package = "VISTA")
count_subset <- count_data[seq_len(500), c("gene_id", si$sample_names), drop = FALSE]

report <- validate_vista_deep(
  counts = count_subset,
  sample_info = si,
  column_geneid = "gene_id",
  group_column = "cond_long",
  group_numerator = "treatment1",
  group_denominator = "control",
  methods = "limma",
  min_counts = 5,
  min_replicates = 1,
  error = FALSE
)

report$valid

data("count_data", package = "VISTA")
data("sample_metadata", package = "VISTA")

target_groups <- c("control", "treatment1")
sample_subset <- sample_metadata[sample_metadata$cond_long %in% target_groups, ]
count_subset <- count_data[seq_len(150), c("gene_id", sample_subset$sample_names)]
```

```

validate_vista_deep(
  counts = count_subset,
  sample_info = sample_subset,
  column_geneid = "gene_id",
  group_column = "cond_long",
  group_numerator = "treatment1",
  group_denominator = "control",
  methods = c("deseq2", "edgeR"),
  min_counts = 5,
  min_replicates = 1
)

```

VISTA-accessors

Accessor Methods for VISTA Object

Description

These accessor functions expose the analysis components stored in a VISTA object. Core expression matrices and annotations live in the underlying SummarizedExperiment, while differential expression results, summaries, and configuration details are kept inside `metadata(x)`.

Usage

```

## S4 method for signature 'VISTA'
comparisons(object, source = "active")

## S4 method for signature 'VISTA'
deg_summary(object, source = "active")

## S4 method for signature 'VISTA'
cutoffs(object)

## S4 method for signature 'VISTA'
norm_counts(object, summarise = FALSE)

## S4 method for signature 'VISTA'
sample_info(object)

## S4 method for signature 'VISTA'
row_data(object)

## S4 method for signature 'VISTA'
group_colors(object)

## S4 method for signature 'VISTA'
group_palette(object)

```

Arguments

<code>object</code>	An object of class VISTA.
<code>source</code>	Which DE result source to use for <code>comparisons()</code> / <code>deg_summary()</code> . One of "active", "deseq2", "edgeR", "limma", or "consensus". "active" uses the currently selected source stored in metadata.
<code>summarise</code>	Logical. If TRUE, returns mean-normalized counts grouped by the grouping column stored in the VISTA object (e.g., condition or treatment). Default is FALSE.

Value

The content of the respective slot or processed data:

comparisons A named list of differential expression tables stored in `metadata(x)$de_results`.

deg_summary A named list of DEG summary tables stored in `metadata(x)$de_summary`.

cutoffs A list of analysis thresholds held in `metadata(x)$de_cutoffs` (empty list if absent).

norm_counts A matrix of normalized counts, optionally averaged by group.

sample_info A `DataFrame` of sample metadata.

row_data A `DataFrame` of gene-level annotation (e.g., `baseMean`, gene ID).

group_colors A named character vector of colours from `metadata(x)$group$colors`.

group_palette The qualitative palette name stored in `metadata(x)$group$palette`.

See Also

[create_vista\(\)](#), [as_vista\(\)](#), [run_deseq_analysis\(\)](#)

Examples

```
# Create example VISTA object
data("count_data", package = "VISTA")
data("sample_metadata", package = "VISTA")

vista <- create_vista(
  counts = count_data[seq_len(100), ],
  sample_info = sample_metadata[seq_len(6), ],
  column_geneid = "gene_id",
  group_column = "cond_long",
  group_numerator = "treatment1",
  group_denominator = "control"
)

# Access differential expression comparisons
comps <- comparisons(vista)
names(comps)
comparisons(vista, source = "active")

# View DEG summary statistics
deg_summary(vista)
```

```

# Get analysis cutoffs
cutoffs(vista)

# Access normalized counts
nc <- norm_counts(vista)
head(nc)

# Get group-summarized counts
nc_summary <- norm_counts(vista, summarise = TRUE)
head(nc_summary)

# Access sample metadata
sample_info(vista)

# Access gene-level annotations
head(row_data(vista))

# Get group colors
group_colors(vista)

# Get palette name
group_palette(vista)

# For method = "both", inspect method-specific outputs

vista_both <- create_vista(
  counts = count_data,
  sample_info = sample_metadata,
  column_geneid = "gene_id",
  group_column = "cond_long",
  group_numerator = "treatment1",
  group_denominator = "control",
  method = "both",
  result_source = "consensus"
)
comparisons(vista_both, source = "deseq2")
comparisons(vista_both, source = "edger")
vista_both <- set_de_source(vista_both, "edger")

```

VISTA-class

VISTA S4 Class Definition

Description

The VISTA class extends SummarizedExperiment. All analysis metadata (DE results, summaries, cutoffs, grouping info, etc.) is stored in the `metadata()` slot of the object.

Details

Core elements stored in `metadata(v)`:

- `$de_results`: named `SimpleList` of DE result tables.
- `$de_summary`: named `SimpleList` of DEG summary tables.
- `$de_cutoffs`: named list of thresholds (log2FC, p-value type/cutoff, method settings).
- `$group`: list with column, palette, colors describing the grouping/fill scheme.
- `$provenance`: list with constructor version, timestamp, session info.
- `$vista_schema_version`: metadata schema tag used for compatibility checks.

Feature-level annotations live in `rowData(v)`, sample metadata in `colData(v)`, and normalized counts (and any additional assays) in `assay(v, "norm_counts")` by default. Use `create_vista()` as the primary end-user constructor. Advanced users can convert an existing `SummarizedExperiment` with `as_vista()`.

Value

A VISTA S4 object.

See Also

[create_vista](#), [as_vista](#), [validate_vista](#)

Examples

```
methods::showClass('VISTA')
```

Index

* datasets

- count_data, [7](#)
- sample_metadata, [100](#)
- as_vista, [4](#), [10](#), [113](#)
- as_vista(), [111](#)
- benchmark_vista_equivalence, [5](#)
- calcNormFactors, [97](#)
- comparisons (VISTA-accessors), [110](#)
- comparisons, VISTA-method (VISTA-accessors), [110](#)
- count_data, [7](#), [100](#)
- create_vista, [8](#), [95](#), [98](#), [113](#)
- create_vista(), [14](#), [111](#)
- cutoffs (VISTA-accessors), [110](#)
- cutoffs, VISTA-method (VISTA-accessors), [110](#)
- deg_summary (VISTA-accessors), [110](#)
- deg_summary, VISTA-method (VISTA-accessors), [110](#)
- derive_vista_metadata, [11](#)
- DESeq, [97](#), [98](#)
- eBayes, [97](#)
- enrichMsigDB, [12](#)
- enrichMsigDB(), [31](#), [76](#), [77](#)
- enrichMsigDB, VISTA-method (enrichMsigDB), [12](#)
- example_vista, [14](#)
- export_vista_assets, [14](#)
- get_cell_fractions, [16](#)
- get_celltype_barplot, [17](#)
- get_celltype_group_dotplot, [18](#)
- get_celltype_heatmap, [19](#)
- get_corr_heatmap, [21](#)
- get_deg_alluvial, [23](#)
- get_deg_count_barplot, [24](#)
- get_deg_count_donutplot, [25](#)
- get_deg_count_pieplot, [26](#)
- get_deg_venn_diagram, [27](#)
- get_enrichment_chord, [28](#)
- get_enrichment_plot, [31](#)
- get_enrichment_plot(), [29](#)
- get_expression_barplot, [32](#)
- get_expression_boxplot, [34](#)
- get_expression_boxplot(), [50](#), [52–54](#)
- get_expression_chromosome_plot, [35](#)
- get_expression_density, [38](#)
- get_expression_heatmap, [40](#)
- get_expression_heatmap(), [81](#), [82](#)
- get_expression_joyplot, [42](#)
- get_expression_lineplot, [44](#)
- get_expression_lollipop, [46](#)
- get_expression_matrix, [48](#)
- get_expression_raincloud, [49](#)
- get_expression_scatter, [51](#)
- get_expression_violinplot, [52](#)
- get_foldchange_barplot, [54](#)
- get_foldchange_boxplot, [56](#)
- get_foldchange_chromosome_plot, [57](#)
- get_foldchange_heatmap, [58](#)
- get_foldchange_lineplot, [61](#)
- get_foldchange_lollipop, [62](#)
- get_foldchange_matrix, [64](#)
- get_foldchange_raincloud, [65](#)
- get_foldchange_scatter, [67](#)
- get_genes_by_regulation, [69](#)
- get_go_enrichment, [70](#)
- get_gsea, [71](#)
- get_kegg_enrichment, [72](#)
- get_ma_plot, [73](#)
- get_mds_plot, [75](#)
- get_msigdb_enrichment, [76](#)
- get_msigdb_enrichment(), [29](#)
- get_pairwise_corr_plot, [78](#)
- get_pathway_genes, [79](#)

`get_pathway_genes()`, 81
`get_pathway_heatmap`, 81
`get_pathway_heatmap()`, 29
`get_pca_plot`, 83
`get_umap_plot`, 85
`get_volcano_plot`, 87
`ggplot`, 74
`glmLRT`, 97, 98
`group_colors` (VISTA-accessors), 110
`group_colors`, VISTA-method
 (VISTA-accessors), 110
`group_palette` (VISTA-accessors), 110
`group_palette`, VISTA-method
 (VISTA-accessors), 110

`make.unique()`, 92
`match_vista_inputs`, 89

`norm_counts` (VISTA-accessors), 110
`norm_counts`, VISTA-method
 (VISTA-accessors), 110

`print.VISTA`, 90
`print.vista` (`print.VISTA`), 90

`qualitative_hcl`, 10

`read_vista_counts`, 91
`read_vista_metadata`, 93
`results`, 97
`row_data` (VISTA-accessors), 110
`row_data`, VISTA-method
 (VISTA-accessors), 110
`run_cell_deconvolution`, 94
`run_deseq_analysis`, 95, 95
`run_deseq_analysis()`, 111
`run_edger_analysis`, 95
`run_edger_analysis`
 (`run_deseq_analysis`), 95
`run_limma_analysis`, 95
`run_limma_analysis`
 (`run_deseq_analysis`), 95
`run_vista_report`, 98

`sample_info` (VISTA-accessors), 110
`sample_info`, VISTA-method
 (VISTA-accessors), 110
`sample_metadata`, 100
`save_vista_data`, 101
`save_vista_data()`, 15

`save_vista_plot`, 102
`set_de_source`, 103
`set_rowdata`, 103
`set_vista_comparison_colors`, 105
`set_vista_group_colors`, 106

`validate_vista`, 107, 113
`validate_vista_deep`, 108
`VISTA`, 74
`VISTA-accessors`, 110
`VISTA-class`, 10, 112
`voom`, 97, 98