Package: phiper (via r-universe)

May 15, 2026

Title Automated PhIP-seq Analysis and Reporting

Version 0.4.1

Description Provides an end-to-end toolkit for Phage ImmunoPrecipitation- sequencing (PhIP-seq) data. Functions import raw peptide-sample count matrices, apply quality control filters, normalise library sizes, compute enrichment statistics and diversity metrics, and identify differentially enriched peptides or motifs. Results can be explored through tidy data frames, visualised with publication-ready ggplot2 graphics, or rendered into fully fledged HTML reports via R Markdown.

License GPL (>= 3)

Depends R (>= 4.1.0)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

URL https://github.com/Polymerase3/phiper, https://polymerase3.github.io/phiper/

BugReports https://github.com/Polymerase3/phiper/issues

SystemRequirements C++17

LinkingTo Rcpp, RcppParallel

Suggests future, future.apply, ggsignif, ggpubr, knitr, kableExtra, mgcv, mockery, openxlsx, parallelDist, readr, rmarkdown, rprojroot, testthat (>= 3.0.0), vdiffr, withr, yaml

VignetteBuilder knitr

Config/testthat/edition 3

Remotes Polymerase3/phiperio

Imports phiperio, chk, cli, DBI, dbplyr, dplyr, duckdb, ggplot2, rlang, scales, tibble, tidyr, tidyselect, vegan, Rcpp, RcppParallel, utils, Rtsne, plotly, showtext, sysfonts

Config/pak/sysreqs
cmake libfreetype6-dev make libicu-dev libpng-dev libuv1-dev libssl-dev xz-utils zlib1g-dev

Repository https://polymerase3.r-universe.dev

Date/Publication 2026-04-15 18:35:49 UTC

RemoteUrl https://github.com/Polymerase3/phiper

RemoteRef main

RemoteSha 688367ea198d824e1db99e3025dfef7574553fdb

Contents

`compute_alpha` *Compute alpha diversity per sample / group across ranks*

Description

Computes **richness**, **Shannon**, **Simpson**, **Pielou's evenness**, and **Berger-Parker dominance** per sample and per grouping variable at one or more **ranks** (columns describing peptides).

Usage

```
compute_alpha(
  x,
  group_cols = NULL,
  ranks = "peptide_id",
  mode = c("binary", "threshold", "abundance"),
  abundance_col = NULL,
  threshold = NULL,
  abundance_agg = c("mean", "sum", "max"),
  metrics = .alpha_metric_names,
  shannon_base = c("ln", "log2", "log10"),
  carry_cols = NULL,
  group_interaction = FALSE,
  interaction_only = FALSE,
  interaction_sep = " * ",
  shannon_log = NULL
)
```

Arguments

`x`	A <phip_data> object or a long `data.frame`.
`group_cols`	Character vector of grouping columns, or `NULL` for a single aggregate (non-facetted) table. All columns must be present on the input.
`ranks`	Character vector of **exact column names** to aggregate by. Typical values: `"peptide_id"` or taxonomy/lineage columns.
`mode`	One of `"binary"` (default), `"threshold"`, or `"abundance"`. Controls how per-category counts are computed; see **Modes** in Details.
`abundance_col`	Character scalar naming the abundance column. Required for `mode = "abundance"`; optional for `mode = "threshold"` (defaults to `"fold_change"`); ignored for `mode = "binary"`.
`threshold`	Numeric scalar. Required for `mode = "threshold"`. Ignored for other modes.
`abundance_agg`	One of `"mean"` (default), `"sum"`, or `"max"`. Used only in `mode = "abundance"` when `rank != "peptide_id"` to aggregate peptide-level abundances within each rank category.

`metrics` Character vector of diversity metrics to compute. Any subset of: `"richness"`, `"shannon"`, `"simpson"`, `"pielou_evenness"`, `"berger_parker"`. Defaults to all five. Output columns follow the canonical names: `richness`, `shannon_diversity`, `simpson_diversity`, `pielou_evenness`, `berger_parker_dominance`.

`shannon_base` One of `"ln"`, `"log2"`, `"log10"`; reporting base for the Shannon index (via base change from natural log).

`carry_cols` Optional character vector of extra columns to carry forward into the output if present (e.g. sample metadata in your table).

`group_interaction` Logical; also compute the interaction of all `group_cols` (default `FALSE`).

`interaction_only` Logical; if `TRUE`, return only the interaction table (requires `group_interaction = TRUE` and at least two `group_cols`).

`interaction_sep` Separator used for the interaction label (default `" * "`). The resulting string (e.g. `"GroupA * T1"`) becomes both a list name and a value in the output column, so choose a separator that does not appear in your group levels.

`shannon_log` Deprecated. Use `shannon_base` instead.

Details

Ranks:

Ranks are the peptide identities or characteristics you aggregate by. They must be **exact column names**:

- For `<phip_data>`: columns in the PHIPER peptide library (e.g. `peptide_id`, lineage/taxa fields).
- For `data.frame`: columns present on your long count table.

Modes:

The `mode` parameter controls what `n` represents in every diversity formula:

- `"binary"` (default): presence = `exist > 0`. `n` = count of distinct enriched peptides per `(sample, rank_val)`.
- `"threshold"`: presence = `abundance_col > threshold`. `n` = count of peptides passing the threshold. `threshold` is required; `abundance_col` defaults to `"fold_change"` when `NULL`.
- `"abundance"`: no binarisation. `abundance_col` is required. At `rank = "peptide_id"`, `n` = the raw abundance value per peptide. At higher ranks, peptides are aggregated within each `(sample, rank_val)` using `abundance_agg` (`"mean"`, `"sum"`, or `"max"`).

Grouping, interactions, and interaction-only mode:

- `group_cols` can be a character vector; the return value is a **named list** of data frames, one per `group_col`.
- If `group_cols = NULL`, a single non-facetted table is returned under the name `"all_samples"`.
- If `group_interaction = TRUE` (and you supplied >= 2 `group_cols`), an additional element is computed for the interaction of all group columns, with labels joined by `interaction_sep`.
- If `interaction_only = TRUE`, you get **only** that interaction element (requires `group_interaction = TRUE` and at least 2 `group_cols`).

Value

A **named list** of data frames with S3 class "phip_alpha_diversity". Each element (per group_col, plus optional interaction or "all_samples") contains: rank, sample_id, the grouping column (or group when group_cols = NULL), any carry_cols, and one column per requested metric (see metrics). pielou_evenness is NA when richness <= 1; berger_parker_dominance is NA when richness == 0.

Examples

```
pd <- load_example_data()
# phip_data input: peptide-level diversity by group
out <- compute_alpha(
  pd, group_cols = "group", ranks = "peptide_id"
)

# include interaction of multiple grouping variables
out2 <- compute_alpha(
  pd,
  group_cols = c("group", "timepoint"),
  ranks = c("peptide_id", "family", "genus"),
  group_interaction = TRUE
)

# interaction only (returns a single element named "group * timepoint")
out3 <- compute_alpha(
  pd,
  group_cols = c("group", "timepoint"),
  ranks = "peptide_id",
  group_interaction = TRUE,
  interaction_only = TRUE
)

## Not run:
# data.frame input: ranks must be columns in the data
out_df <- compute_alpha(
  df_long, group_cols = NULL, ranks = "peptide_id"
)

## End(Not run)

# threshold mode: presence = fold_change > 1.5
out_thr <- compute_alpha(
  pd, group_cols = "group", ranks = "peptide_id",
  mode = "threshold", threshold = 1.5
)
```

---

compute_alpha_significance

Compute statistical significance of alpha diversity between groups

---

Description

Runs global (Kruskal-Wallis or one-way ANOVA) and pairwise (Wilcoxon or Tukey HSD) hypothesis tests on each (rank, metric) combination present in the output of compute_alpha(). Pairwise p-values are adjusted with p.adjust() and Cohen's d effect sizes are appended.

Usage

```
compute_alpha_significance(
  x,
  group_col = NULL,
  metric = NULL,
  global_test = c("kruskal", "anova"),
  pairwise_test = c("wilcoxon", "tukey"),
  p_adjust_method = "BH"
)
```

Arguments

x	A "phip_alpha_diversity" list (output of compute_alpha()) or a single alpha-diversity data frame.
group_col	Character scalar; name of the grouping column. Inferred from attr(x, "group_cols") when NULL (default).
metric	Character vector; subset of metrics to test. NULL (default) uses all numeric metric columns present in the data.
global_test	One of "kruskal" (Kruskal-Wallis, default) or "anova" (one-way ANOVA).
pairwise_test	One of "wilcoxon" (Wilcoxon rank-sum, default) or "tukey" (Tukey HSD from aov()).
p_adjust_method	Method passed to p.adjust(). Default "BH".

Value

A named list of class "phip_alpha_significance" with two tibbles:

$global One row per (rank, metric): rank, metric, statistic, p_value, test.

$pairwise One row per (rank, metric, pair): rank, metric, group1, group2, p_raw, p_adj, cohens_d, stars, test.

Attributes: group_col, global_test, pairwise_test, p_adjust_method, metrics, ranks.

Examples

```
## Not run:
alpha <- compute_alpha(phip_obj, group_cols = "group")
sig   <- compute_alpha_significance(alpha)
sig$global
sig$pairwise

## End(Not run)
```

compute_capscale *Constrained Ordination (db-rda / cap) on Distance Matrix*

Description

Performs distance-based redundancy analysis (constrained pcoa, a.k.a. cap) on a distance matrix using **vegan**::capscale, with optional negative eigenvalue correction. Returns constrained sample scores, eigenvalues, variance partitioning, and feature associations.

Usage

```
compute_capscale(
  dist_obj,
  ps,
  formula,
  neg_correction = c("none", "lingoes", "cailliez"),
  top_features = 30L,
  permutations = 999L,
  feature_assoc = c("weighted_average", "correlation", "regression", "none")
)
```

Arguments

dist_obj	A dist object returned by compute_distance(). The normalized abundance matrix used to compute the distances is expected to be attached as an attribute "abundances" (samples in rows, features in columns).
ps	A phip_data object or a table providing sample-level metadata. This table must contain sample_id and all variables referenced on the right-hand side of formula. Variable detection uses all.names(terms(formula), functions = FALSE), so transformed terms like log(age) are supported as long as age exists.
formula	An R formula specifying the constraints (independent variables) for the ordination, e.g. ~ sex + age. Do not include a response on the left-hand side; the distance matrix is provided via dist_obj.
neg_correction	One of "none", "lingoes", "cailliez". Method for negative eigenvalue correction. Default is "none". Passed to the add argument of vegan::capscale().
top_features	Integer scalar. Number of top features to return in associations (selected per constrained axis by absolute association, then unioned). Default is 30.
permutations	Integer scalar. Number of permutations for per-term permutation tests via vegan::anova.cca(by = "term"). Default is 999.
feature_assoc	character scalar. Type of feature-axis association to return. "weighted_average" returns weighted-average feature scores (centroid of sample scores weighted by feature abundance). "correlation" returns feature-axis correlations. "regression" returns regression slopes for axis scores on feature abundance. "none" skips feature associations.

Value

A list of class "beta_capscale" with elements:

sample_coords	Tibble of sample scores on constrained axes (CAP1, CAP2, ...). Contains sample_id and coordinates.
eigenvalues	Numeric vector of eigenvalues of the constrained axes.
variance_partition	Tibble with total inertia and inertia partitioned into constrained and unconstrained components, with their proportion of total.
feature_associations	Tibble of top feature-axis associations for constrained axes (possibly empty if the "abundances" attribute is missing or cannot be aligned). To limit runtime/memory, associations are computed for at most 10 constrained axes.
r2	Numeric scalar. Unadjusted R-squared from vegan::RsquareAdj().
r2_adj	Numeric scalar. Adjusted R-squared from vegan::RsquareAdj().
perm_terms	Tibble of per-term permutation tests from vegan::anova.cca(by = "term").
cap_model	The full vegan::capscale model object.

Examples

```
ps <- load_example_data("small_mixture")

# compute distance matrix
val_col <- "fold_change"

dist_bc <- compute_distance(
  ps,
  value_col = val_col,
  method_normalization = "hellinger",
  distance = "bray",
  n_threads = 2L
)

# pick a simple constraint that exists in the example data (fallback order)
dat <- ps
cand <- c("group", "big_group", "type_person", "sex", "age")
rhs_var <- cand[cand %in% dplyr::tbl_vars(dat)][1]

cap_res <- compute_capscale(
  dist_bc,
  ps = ps,
  formula = stats::as.formula(paste0("~ ", rhs_var)),
  neg_correction = "none",
  top_features = 30L
)

cap_res$variance_partition
cap_res$sample_coords
cap_res$feature_associations
```

compute_delta *Global Shift in Peptide-level Prevalence via Subject-level Permutation*

Description

Test for a **global** (antigen-wide) shift in peptide-level prevalence between two groups within each (rank, feature, group_col) stratum by aggregating per-peptide effects into a single Stouffer-type statistic and assessing significance via **label permutation**.

Usage

```
compute_delta(
  x,
  rank_cols,
  group_cols,
  exist_col = "exist",
  paired_by = NULL,
  interaction = FALSE,
  combine_cols = NULL,
  interaction_sep = "::",
  B_permutations = 2000L,
  weight_mode = c("equal", "se_invvar", "n_eff_sqrt"),
  stat_mode = c("srlr", "diff", "asin", "score", "mcnemar", "srlr_paired"),
  perm_method = c("mid_p", "standard"),
  aggregate_stat = c("stouffer", "maxmean", "af"),
  strat_bins = c(0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5),
  winsor_z = 4,
  rank_feature_keep = NULL,
  peptide_library = NULL,
  log = FALSE,
  log_file = "compute_delta.log"
)
```

Arguments

x	A phip_data object (recommended to attach a peptide_library when rank_cols includes non-"peptide_id" and not providing peptide_library argument) or a data frame with the necessary columns.
rank_cols	Character vector of rank columns (e.g. "peptide_id", "species", ...). For non-peptide ranks, annotations are joined from a peptide library (see Details).
group_cols	Character vector of grouping columns that define universes; pairwise group contrasts are constructed within each (rank, feature, group_col) stratum.
exist_col	Name of the 0/1 presence column. Default "exist".
paired_by	Optional single column name identifying paired samples (e.g. subject ID). When provided, paired tests are used where applicable.

`interaction` Logical; reserved for future use (currently ignored). If `TRUE`, an interaction of the first two `group_cols` may be used as an additional `group_col` in future versions.

`combine_cols` Optional length-2 character vector; reserved for future use. If non-`NULL`, only this interaction would be used as `group_col` in future versions.

`interaction_sep` Separator for interaction values. Default `"::"`.

`B_permutations` Number of permutations `B` used for the permutation p-value. Default `2000L`. Must be at least 100.

`weight_mode` One of `c("equal", "se_invvar", "n_eff_sqrt")`, controlling how peptide-level z-scores are weighted in the Stouffer combination (see Details).

`stat_mode` One of `c("diff", "asin", "score", "srlr", "mcnemar", "srlr_paired")`, determining the peptide-level test statistic (see Details). The paired-only modes require paired designs.

`perm_method` One of `c("standard", "mid_p")`, controlling how the permutation p-value is computed (see Details). Default `"standard"`.

`aggregate_stat` One of `c("stouffer", "maxmean", "af")`, controlling how peptide-level z-scores are aggregated into a single test statistic (see Details).

`strat_bins` Numeric vector of pooled prevalence cutpoints in (0, 1). If `0`, no stratification is used (single global Stouffer statistic). Otherwise, peptides are binned by pooled prevalence and the mean of bin-level z-scores is used as the global test statistic.

`winsor_z` Winsorization threshold applied to peptide-level z-scores. Values beyond `±winsor_z` are truncated. Default `4.0`.

`rank_feature_keep` Optional **named list** mapping `rank` to a vector of `feature` values to keep. Only rank–feature strata in this list are tested; others are dropped after the peptide-level pivot.

`peptide_library` Optional data frame providing peptide annotations for non-peptide ranks. Must at least contain `peptide_id` and all requested `rank_cols` besides `"peptide_id"`. If `NULL`, the function falls back to `x$peptide_library` or `get_peptide_library()` (from phiperio).

`log` Logical; if `TRUE`, write progress messages (per contrast and overall) using the package's logging helpers.

`log_file` Path to a log file used by the logging helpers if `log` is `TRUE`. Default `"compute_delta.log"`.

Details

For each stratum `(rank, feature, group_col)` that defines an ordered pair of groups `(g1, g2)`, the procedure is:

1. **Peptide-level prevalences.** For each peptide, prevalences in `g1` and `g2` are

$$\hat{p}_k = \frac{x_k}{n_k}, \quad k \in \{g1, g2\},$$

where `x_k` is the number of samples with presence (`exist > 0`) and `n_k` is the number of distinct samples (or paired subjects).

2. **Per-peptide effect and z-score (`stat_mode`).**
   - `"diff"`: effect $\Delta = \hat{p}_{g2} - \hat{p}_{g1}$ with
   $$SE = \sqrt{\hat{p}_{g1}(1-\hat{p}_{g1})/n_{g1} + \hat{p}_{g2}(1-\hat{p}_{g2})/n_{g2}},$$
   and z-score $z = \Delta/SE$ (with a small floor on `SE`).
   - `"asin"`: variance-stabilized difference of angles,
   $$z = \frac{\arcsin\sqrt{\hat{p}_{g2}} - \arcsin\sqrt{\hat{p}_{g1}}}{\sqrt{1/(4n_{g1}) + 1/(4n_{g2})}}.$$
   - `"score"`: pooled score z for equality of proportions (2×2 score test), using raw proportions $\tilde{p}_k = x_k/n_k$ and pooled $\tilde{p} = (x_1 + x_2)/(n_1 + n_2)$:
   $$z = \frac{\tilde{p}_{g2} - \tilde{p}_{g1}}{\sqrt{\tilde{p}(1-\tilde{p})\,(1/n_{g1} + 1/n_{g2})}}.$$
   - `"srlr"`: signed root likelihood ratio for $H_0 : p_{g1} = p_{g2}$:
   $$LR = 2\left[\ell(\hat{p}_{g1}; x_1, n_1) + \ell(\hat{p}_{g2}; x_2, n_2) - \ell(\hat{p}; x_1, n_1) - \ell(\hat{p}; x_2, n_2)\right],$$
   $$z = \operatorname{sign}(\hat{p}_{g2} - \hat{p}_{g1})\sqrt{LR},$$
   where $\ell(p; x, n) = x \log p + (n - x)\log(1-p)$ and $\hat{p}_{gk} = x_k/n_k$, $\hat{p} = (x_1 + x_2)/(n_1 + n_2)$.
   - `"mcnemar"` (paired-only): signed McNemar z based on discordant pairs $b = \#(g1 = 1, g2 = 0)$, $c = \#(g1 = 0, g2 = 1)$:
   $$z = \frac{b - c}{\sqrt{b + c}}.$$
   If $b + c = 0$, z is defined as 0.
   - `"srlr_paired"` (paired-only): signed root deviance for the paired binomial problem with $m = b + c$:
   $$z = \operatorname{sign}(b - c)\sqrt{2\left[b \log\frac{2b}{m} + c \log\frac{2c}{m}\right]}.$$
   Terms with $0 \log 0$ are defined as 0; if $m = 0$, z is 0.

   Each z_i is then **winsorized** to $\pm$ `winsor_z`.

3. **Weights (`weight_mode`).**
   - `"equal"`: all peptides get weight 1.
   - `"se_invvar"`: weights proportional to the inverse standard error (or the analogous denominator for `"asin"`).
   - `"n_eff_sqrt"`:
   $$w_i = \sqrt{n_{g1}\hat{p}_{g1} + n_{g2}\hat{p}_{g2}},$$
   i.e. the square root of the expected number of positives across both groups.

4. **Combine into a single test statistic (`aggregate_stat`).** Let w_i be the weights and z_i the peptide-level z-scores.

- If `aggregate_stat = "stouffer"` and `strat_bins = 0`:

$$T_{\text{obs}} = \frac{\sum_i w_i z_i}{\sqrt{\sum_i w_i^2}}.$$

- If `aggregate_stat = "stouffer"` and `strat_bins` is a numeric vector of pooled prevalence cutpoints (between 0 and 1), peptides are binned by pooled prevalence

$$\frac{n_{g1}\hat{p}_{g1} + n_{g2}\hat{p}_{g2}}{n_{g1} + n_{g2}},$$

a Stouffer statistic T_b is computed within each bin. The **mean** of the bin-level z-scores is reported as T_{obs}. For example, if `strat_bins` includes `0.10` and `0.20`, then peptides with pooled prevalence in (0.10, 0.20] share a bin.

5. **Multiplicity.** No multiple-testing correction is performed; `p_perm` is returned as computed.

Input requirements.

- `exist_col` is treated as 0/1 presence.
- There must be **at most one positive** per (`subject_id`, `peptide_id`, `group_col`, `group_value`); paired designs can have up to two positives across the two group levels. Violations trigger an error. Example (group levels A/B): for a single subject and peptide, you may have A=1 and B=0 (or A=0 and B=1, or A=1 and B=1), but you cannot have two rows both with A=1 (or two rows both with B=1).
- Non-peptide ranks specified in `rank_cols` must be resolvable from a peptide library (see `peptide_library` below).

Peptide library resolution.

When `rank_cols` includes non-`"peptide_id"` ranks, a peptide library is required. It is resolved in the following order:

1. The explicit `peptide_library` argument.
2. `x$peptide_library` if `x` is a `phip_data` with an attached library.
3. `get_peptide_library()` from phiperio (always available as a dependency).

Value

A tibble with one row per tested stratum:

- `rank`, `feature`: rank and feature identifiers.
- `group_col`, `group1`, `group2`: grouping column and the ordered pair of groups compared.
- `design`: `"paired"` or `"unpaired"`.
- `n_subjects_paired`: number of paired subjects used in a paired design (or `NA` for unpaired).
- `n_peptides_used`: number of peptides contributing to the test.
- `m_eff`: effective number of peptides after weighting (as returned by the C++ helper).
- `T_obs`: observed Stouffer-type test statistic.
- `p_perm`: two-sided permutation p-value.
- `b`: number of permutations actually used (may be `< B_permutations` if early stopping is implemented in the C++ helper).
- `max_delta`, `frac_delta_pos`, `frac_delta_pos_w`: maximum absolute peptide-level prevalence difference and unweighted/weighted fractions of positive peptide-level deltas.

Parallelization

The permutation contrasts are evaluated either sequentially or in parallel using the **current** `future` backend:

- The function does **not** change the global `future::plan()`.
- If `future` is installed, the number of workers is inferred from `future::nbrOfWorkers()`. To run in parallel, set up a `future` plan (e.g. `future::plan(multisession, workers = 8)`) **outside** the function.
- Internal C++ code is constrained to a single thread per R worker (`RcppParallel::setThreadOptions(numThreads = 1)` and BLAS/OpenMP limits), to avoid oversubscription when using multiple workers.

Reproducibility of permutations is controlled via the global RNG: call `set.seed()` before `compute_delta()` to obtain reproducible results; each contrast draws its own seed from the global RNG state.

Examples

```
# Load example PhIP-Seq data shipped with the package
pd <- load_example_data()

# Small unpaired subset with a mock peptide library
pd_filt <- pd |>
  dplyr::filter(
    peptide_id %in% c("16627", "5243", "24799", "16196", "18003"),
    timepoint == "T1"
  ) |>
  dplyr::collect()

mock_peplib <- data.frame(
  peptide_id = c("16627", "5243", "24799", "16196", "18003"),
  species    = rep("mock_species", 5),
  stringsAsFactors = FALSE
)

res <- compute_delta(
  x                  = pd_filt,
  exist_col          = "exist",
  rank_cols          = "species",
  group_cols         = "group",
  peptide_library    = mock_peplib,
  B_permutations     = 500L,  # smaller for speed
  weight_mode        = "n_eff_sqrt",
  stat_mode          = "asin",
  strat_bins         = 0,
  winsor_z           = Inf,
  rank_feature_keep  = list(species = NULL),
  log                = FALSE
)
res
```

compute_dispersion *Test Homogeneity of Dispersion (Beta Dispersion)*

Description

Computes distances of samples to group centroids (using vegan::betadisper) and tests for differences in dispersion among groups or time levels. Pairwise post-hoc tests are always performed when group_col or time_col has >1 level.

Usage

```
compute_dispersion(
  dist_obj,
  ps,
  group_col = NULL,
  time_col = NULL,
  subject_col = "subject_id",
  permutations = 999,
  p_adjust = "none"
)
```

Arguments

dist_obj	A dist object of sample distances (e.g. from compute_distance()).
ps	A phip_data object or a table providing sample-level metadata. This table must contain sample_id and the columns specified in group_col and/or time_col.
group_col	Name of the group factor column in ps (between-subjects). Use NULL if no group factor.
time_col	Name of the time factor column in ps (within-subjects, categorical only). Use NULL if not applicable.
subject_col	Name of subject identifier column (for reference only; not used directly in dispersion test calculations, but kept for API consistency). Default "subject_id".
permutations	Number of permutations for significance testing in vegan::permutest. Default 999.
p_adjust	P-value adjustment method applied within each contrast scope. Use "none" for raw p-values. Passed to stats::p.adjust().

Value

A list of class "beta_dispersion" with:

distances	Tibble of per-sample distances to centroid. Columns: sample_id, distance, level (group/time level for a given scope), scope (e.g. "group", "time", "group:time"), contrast (e.g. "<global>", "A vs B").
tests	Tibble of dispersion test results. Columns: scope, contrast, term = "dispersion", p_value, p_adjust (equals p_value when p_adjust = "none"), n_perm.

Examples

```
ps <- load_example_data("small_mixture")

# compute distance matrix
val_col <- "fold_change"

dist_bc <- compute_distance(
  ps,
  value_col = val_col,
  distance = "jaccard",
  n_threads = 2L
)

dispersion_res <- compute_dispersion(
  dist_bc,
  ps        = ps,
  group_col = "group",
  time_col  = "timepoint",
  p_adjust  = "BH"
)
dispersion_res$tests
head(dispersion_res$distances)
```

`compute_distance` *Compute Pairwise Sample Distances*

Description

This function builds a sample-by-feature abundance matrix from a `phip_data` object (using `ps$data_long`), optionally normalizes the matrix, and then computes pairwise distances between samples.

The normalized abundance matrix used for distance calculation is attached to the returned `dist` object as attribute `"abundances"`.

Note: this function pivots to a wide matrix in the database (via dbplyr) and then collects the result into memory. This can be large for big cohorts and/or large peptide sets.

Usage

```
compute_distance(
  ps,
  value_col = NULL,
  method_normalization = c("auto", "relative", "hellinger", "log", "none"),
  distance = "bray",
  n_threads = 1L
)
```

Arguments

ps	input data. either: • a phip_data object, in which case ps$data_long is used, or • a data_long-like table (a data.frame or dplyr tbl) containing at least sample_id, peptide_id, and the column given by value_col.
value_col	character scalar. Name of the abundance column in ps$data_long. If NULL, the function tries (in order) exist, counts_hits, counts_control, fold_change.
method_normalization	character scalar. Normalization applied to the abundance matrix before distance computation. One of: • "auto": uses "none" for binary (0/1) data, otherwise uses "relative". • "relative": divide each row by its row sum. • "hellinger": sqrt(relative). • "log": log1p transform. • "none": no normalization.
distance	character scalar. Distance method name. The string is lowercased internally. Supported methods depend on which packages are installed: • fast path (if package 'parallelDist' is installed): – "bray" (bray-curtis). Computed via threaded l1 distances and normalization (equivalent to bray-curtis on the normalized matrix). – "euclidean" – "minkowski" – "manhattan" – "canberra" – "binary" – "maximum" (maximum/supremum/chebyshev distance). Note: 'parallelDist' documents this as method "maximum"; passing "chebyshev" may fail unless you map it to "maximum" before calling parDist(). – "cosine" • fallback path (requires package 'vegan'). Any method supported by vegan::vegdist(), partial match allowed: "manhattan", "euclidean", "canberra", "clark", "bray", "kulczynski", "jaccard", "gower", "altGower", "morisita", "horn", "mountford", "raup", "binomial", "chao", "cao", "mahalanobis", "chisq", "chord", "hellinger", "aitchison", "robust.aitchison". If 'parallelDist' is installed but the requested method is not in the fast list above, the function falls back to vegan::vegdist().
n_threads	integer scalar. Number of cpu threads passed to parallelDist::parDist(threads = ...).

Value

a dist object of pairwise sample distances. The attribute "abundances" contains the normalized abundance matrix used for the calculation (rows are samples, columns are features).

Examples

```
# build an example <phip_data> object from the package example dataset
ps <- load_example_data("small_mixture")

# compute distances (needs either 'parallelDist' or 'vegan')
val_col <- "fold_change"

d <- compute_distance(
  ps,
  value_col = val_col,
  method_normalization = "hellinger",
  distance = "bray",
  n_threads = 2L
)

a <- attr(d, "abundances")
a[1:min(5, nrow(a)), 1:min(5, ncol(a)), drop = FALSE]
```

---

compute_pcoa *Principal Components Analysis (PCoA) on a Distance Matrix*

Description

Performs PCoA on a distance matrix (typically from `compute_distance()`), optionally correcting for negative eigenvalues, and returns coordinates, eigenvalues, variance explained, and feature-axis associations.

Usage

```
compute_pcoa(
  dist_obj,
  neg_correction = c("none", "lingoes", "cailliez"),
  n_axes = 5L
)
```

Arguments

`dist_obj`	a `dist` object (for example returned by `compute_distance()`). The normalized abundance matrix used to compute the distances is attached as attribute `"abundances"` (numeric matrix with samples in rows and features in columns). If missing, feature associations are skipped.
`neg_correction`	character scalar. Method for adjusting negative eigenvalues (if any). One of `"none"`, `"lingoes"`, or `"cailliez"`. Default is `"none"`.
`n_axes`	integer scalar. Number of PCoA axes to return in the sample scores. Must be > 0. Internally, `k = min(n_axes, n_samples - 1)`.

Details

Negative eigenvalues indicate that the distances are not perfectly euclidean. If `neg_correction` is `"lingoes"` or `"cailliez"`, a correction is applied via `vegan::wcmdscale(add = ...)`.

Value

a list of class `"beta_pcoa"` with elements:

- `sample_coords`: tibble with `sample_id` and columns `PCoA1`, `PCoA2`, ... up to `n_axes` (or fewer if `n_samples - 1` is smaller).
- `eigenvalues`: numeric vector of eigenvalues from the PCoA.
- `var_explained`: one-row tibble with percent variance explained by the returned axes and `%Other`. percentages are computed from the sum of positive eigenvalues.
- `eigen_diagnostics`: one-row tibble with eigenvalue diagnostics: `sum_negative`, `sum_positive`, their ratio, the minimum eigenvalue, and the number and fraction of negative eigenvalues.
- `correction_infos`: one-row tibble describing any negative eigenvalue correction applied, including `corrected`, `correction_method`, `correction_add`, and `correction_note`.

Examples

```
# compute a distance matrix with an attached abundance matrix
# build an example <phip_data> object from the package example dataset
ps <- load_example_data("small_mixture")

# compute distances (needs either 'parallelDist' or 'vegan')
val_col <- "fold_change"

d <- compute_distance(
  ps,
  value_col = val_col,
  distance = "jaccard",
  n_threads = 2L
)

pcoa_res <- compute_pcoa(d, neg_correction = "none", n_axes = 3L)
pcoa_res$sample_coords
pcoa_res$var_explained
```

`compute_pcoa_feature_associations`

Compute Feature Associations to PCoA Vectors

Description

Calculates feature-axis associations based on given PCoA results (output of `compute_pcoa()`)

Usage

```
compute_pcoa_feature_associations(
  dist_obj,
  pcoa_result,
  top_features = 30L,
  association_method = c("weighted_average", "correlation", "regression")
)
```

Arguments

dist_obj	a dist object (for example returned by compute_distance()). The normalized abundance matrix used to compute the distances *must* be attached as attribute "abundances" (numeric matrix with samples in rows and features in columns).
pcoa_result	a list of class "beta_pcoa". The result of compute_pcoa(), which contains the resulting PCoA eigen vectors.
top_features	integer scalar. Number of features to keep per axis when reporting associations. Features are selected by taking the union of the top top_features features (by absolute association) for each returned axis. Must be > 0.
association_method	character scalar. Type of feature-axis association to return. "weighted_average" returns weighted-average feature scores (centroid of sample scores weighted by feature abundance). "correlation" returns feature-axis correlations. "regression" returns regression slopes for axis scores on feature abundance. "none" skips feature associations.

Details

These feature associations are post-hoc summaries of how features relate to PCoA axes. Weighted-average scores (association_method = "weighted_average") compute t(X) %*% U / colSums(X), where X is the abundance matrix and U are the sample coordinates. Correlation and regression associations are computed between feature abundances and axis scores and are not "true" PCA loadings unless distances are Euclidean and derived compatibly.

Value

a tibble of feature-axis associations for the returned axes.

Examples

```
# compute a distance matrix with an attached abundance matrix
# build an example <phip_data> object from the package example dataset
ps <- load_example_data("small_mixture")

# compute distances (needs either 'parallelDist' or 'vegan')
val_col <- "fold_change"

d <- compute_distance(
  ps,
  value_col = val_col,
```

```
  distance = "jaccard",
  n_threads = 2L
)

# Compute PCoA vectors on these distances
pcoa_res <- compute_pcoa(d, neg_correction = "none", n_axes = 3L)

feature_associations <- compute_pcoa_feature_associations(d, pcoa_res)
feature_associations
```

`compute_permanova` *PERMANOVA with Global and Post-hoc Tests on Beta Diversity*

Description

Performs PERMANOVA (adonis2) on a distance matrix for overall group/time effects, and optionally conducts post-hoc pairwise or contrast tests (e.g., between each pair of groups, etc.). Supports stratified permutations for repeated measures.

Usage

```
compute_permanova(
  dist_obj,
  ps,
  group_col = NULL,
  time_col = NULL,
  subject_col = NULL,
  permutations = 999,
  p_adjust = "none"
)
```

Arguments

`dist_obj`	A `dist` object of distances between samples (e.g., output of `compute_distance()`).
`ps`	A `phip_data` object or a table providing sample-level metadata. This table must contain `sample_id` and the columns specified in `group_col`, `time_col`, and optionally `subject_col`.
`group_col`	Name of the grouping column in `ps` (between-subject factor). Use `NULL` if no group factor.
`time_col`	Name of the time factor column in `ps` (within-subject factor for longitudinal data). Use `NULL` if not applicable. This should be a *categorical* factor for this function (continuous time not supported).
`subject_col`	Name of the subject identifier column in `ps` (for repeated measures). Default `NULL`. If provided and this column is present and `time_col` is provided, permutations will be stratified by subject. This is a simplification and does not implement a full repeated-measures permutation design (see Details).

permutations	Number of permutations for significance testing (default 999).
p_adjust	P-value adjustment method applied within each contrast scope. Use "none" for raw p-values. Passed to stats::p.adjust().

Details

Global PERMANOVA uses a model with main effects of group_col and time_col and their interaction (if both are provided and have >1 level). Samples with missing values in the constrained variables (group/time and, when used for stratification, subject) are dropped *before* fitting the model, and the distance matrix is subset to the remaining samples so that distances and metadata are always aligned.

Pairwise post-hoc tests are always performed for group_col and time_col (when present and with >1 level), using adonis2 with appropriate subsetting and, where applicable, subject stratification. Stratification via strata = subject is a simplified approach and does not replace a full repeated-measures permutation design (e.g., via permute::how()), especially for multi-level time with missing timepoints.

P-values are optionally adjusted within each contrast scope (e.g., "group_pairwise", "time_pairwise") using stats::p.adjust().

Value

A tibble with columns:

scope	The scope of the test (e.g., "global", "group_pairwise", "time_pairwise").
contrast	Description of the contrast (e.g., "<global>" for overall test, "A vs B" for pairwise group or time comparisons).
term	The term being tested. For global tests, this will be the factor name (or interaction). For post-hoc tests, it may be "group" or "time" indicating which factor is being contrasted.
F_stat	F statistic of the PERMANOVA test (for global tests and some contrasts where applicable).
R2	R-squared (variance explained) for the term (global tests).
p_value	Permutation p-value for the test.
p_adjust	Adjusted p-value (within scope); equals p_value when p_adjust = "none".
n_perm	Number of permutations used.

Examples

```
ps <- load_example_data("small_mixture")

# compute distance matrix
val_col <- "fold_change"

dist_bc <- compute_distance(
  ps,
  value_col = val_col,
  distance = "jaccard",
```

```
  n_threads = 2L
)

permanova_res <- compute_permanova(
  dist_bc,
  ps        = ps,
  group_col = "group",
  time_col  = "timepoint"
)

permanova_res2 <- compute_permanova(
  dist_bc,
  ps        = ps,
  group_col = "group",
  time_col  = "timepoint",
  p_adjust  = "BH"
)
```

`compute_pop` *Prevalence comparison by group (POP framework)*

Description

Computes feature-level prevalence across binary group columns and performs pairwise statistical tests:

- **Unpaired:** Fisher's exact test (2x2) per (rank, feature, group pair).
- **Paired:** McNemar's exact binomial test per subject.

Presence per sample is determined by a k-of-n rule: a feature is considered present in a sample if at least `pop_k_min` of its contributing peptides are positive. Each `group_col` must have exactly 2 non-missing levels in the data.

Usage

```
compute_pop(
  x,
  rank_cols,
  group_cols,
  exist_col = "exist",
  pop_k_min = 1L,
  paired = FALSE,
  peptide_library = NULL
)
```

Arguments

x	A phip_data object or a data.frame/tibble with at least columns sample_id, peptide_id, exist_col, and all group_cols. If paired is set, the paired linking column must also be present. If rank_cols include non-peptide taxa, x must provide peptide_library.
rank_cols	Character vector of rank columns, e.g. c("peptide_id", "species").
group_cols	Character vector of binary grouping columns. Each column must have exactly 2 non-missing levels in the data.
exist_col	Name of the binary presence column (default "exist").
pop_k_min	Integer >= 1; k-of-n POP threshold per sample (default 1).
paired	FALSE (default) or a single string naming the column that links related samples (e.g. "subject_id"). When set, paired McNemar tests are used instead of Fisher.
peptide_library	Optional peptide metadata table with peptide_id and any non-peptide rank columns. If NULL, taken from x$peptide_library.

Value

A data.frame with one row per (rank, feature, group_col) comparison: rank, feature, group_col, group1, group2, n1, N1, prop1, percent1, n2, N2, prop2, percent2, ratio, delta_ratio (unpaired only), p_raw, n_peptides. A view column is prepended when the input carries a view attribute.

Examples

```
## Not run:
res <- compute_pop(pd, rank_cols = "species", group_cols = "group")
res

## End(Not run)
```

---

compute_tsne *Compute t-SNE Embeddings for Sample Distances*

Description

Performs t-distributed stochastic neighbor embedding (t-SNE) on a sample distance matrix to create low-dimensional embeddings for visualization. Returns t-SNE coordinates with optional sample metadata attached.

Usage

```
compute_tsne(
  ps,
  dist_obj,
  dims = 3L,
  perplexity = 30,
  theta = 0.5,
  max_iter = 1000L,
  meta_cols = NULL,
  seed = NULL,
  check_duplicates = FALSE,
  ...
)
```

Arguments

ps	A phip_data object or a table providing sample-level metadata. This table must contain sample_id and any columns specified in meta_cols.
dist_obj	A sample distance object. Either: • A dist object (e.g., from compute_distance()). • A numeric, symmetric matrix with row/column names corresponding to sample_ids.
dims	Integer scalar. Number of t-SNE dimensions to compute (2 or 3). Default is 3 to enable both 2D and 3D visualizations.
perplexity	Numeric scalar. Perplexity parameter for t-SNE. Must be smaller than the number of samples. If too large, it is automatically reduced with a warning.
theta	Numeric scalar. Speed/accuracy tradeoff for Barnes-Hut approximation. Default is 0.5.
max_iter	Integer scalar. Maximum number of t-SNE iterations. Default is 1000.
meta_cols	Character vector. Column names from ps to attach as metadata. If NULL, uses ps$meta$extra_cols when available.
seed	Integer scalar. Random seed for reproducibility. If provided, the seed is temporarily set and restored after computation.
check_duplicates	Logical scalar. Whether to check for duplicate points. Default is FALSE for distance input.
...	Additional arguments passed to Rtsne::Rtsne().

Details

This function runs t-SNE in distance mode (is_distance = TRUE), using the supplied distance matrix directly. The distance computation should be performed separately using compute_distance().

t-SNE is a visualization method: it preserves local neighborhoods rather than global distances, axes are not directly interpretable, and embeddings can change with different seeds or perplexity settings.

Samples with missing values in metadata columns are retained in the t-SNE result but will have NA values for those metadata columns.

Value

A tibble of class `c("phip_tsne", "tbl_df", "tbl","data.frame")` with columns:

- `sample_id`: Sample identifier from distance labels.
- `tSNE1`, `tSNE2`: First two t-SNE dimensions.
- `tSNE3`: Third dimension if `dims >= 3`, otherwise `NA`.
- Additional metadata columns as specified in `meta_cols`.

Attributes:

- `"distance"`: The original `dist_obj`.
- `"tsne_params"`: List of t-SNE parameters and function call.
- `"meta_cols"`: Character vector of metadata columns used.

Examples

```
# Build example phip_data object
ps <- load_example_data("small_mixture")

# Compute distance matrix
val_col <- "fold_change"

d <- compute_distance(
  ps,
  value_col = val_col,
  method_normalization = "hellinger",
  distance = "bray",
  n_threads = 2L
)

# Compute t-SNE embeddings
tsne_res <- compute_tsne(
  ps = ps,
  dist_obj = d,
  dims = 3L,
  perplexity = 15,
  meta_cols = c("subject_id", "timepoint"),
  seed = 42
)

# View results
head(tsne_res)
```

deltaplot *Delta-prevalence vs Pooled Prevalence*

Description

Build a static ggplot showing the per-peptide shift in prevalence ($\Delta = group2 - group1$) as a function of pooled prevalence ($(group1 + group2)/2$). The input should be a tibble/data frame produced by `ph_compute_prevalence()` or equivalent with columns `group1`, `group2`, `prop1`, and `prop2`.

Usage

```
deltaplot(
  prev_tbl,
  group_pair_values = NULL,
  group_labels = NULL,
  point_jitter_width = 0.005,
  point_jitter_height = 0.005,
  point_alpha = 0.25,
  point_size = 0.6,
  add_smooth = TRUE,
  smooth_k = 5,
  arrow_color = "red",
  arrow_head_length_mm = 4,
  arrow_x_frac = 0.97,
  plot_title = NULL,
  plot_subtitle = NULL,
  x_label = NULL,
  y_label = NULL
)
```

Arguments

`prev_tbl`	Data frame with columns `group1`, `group2`, `prop1`, `prop2`. Optional `feature` is used for row identity only.
`group_pair_values`	Optional length-2 character vector `c(group1, group2)`. Use this when `prev_tbl` contains multiple group pairs.
`group_labels`	Optional length-2 character vector of display labels `c(label_group1, label_group2)`. Defaults to `group1`/`group2`.
`point_jitter_width`, `point_jitter_height`	Jitter amounts for the points. Defaults 0.005.
`point_alpha`	Point transparency. Default 0.25.
`point_size`	Point size. Default 0.6.
`add_smooth`	Add a GAM smooth curve (`mgcv`). Default `TRUE`.

`smooth_k` Basis dimension k for the smooth. Default 5.

`arrow_color` Color for the directional arrows and labels. Default `"red"`.

`arrow_head_length_mm`
Arrow head length in mm. Default 4.

`arrow_x_frac` Arrow X position as a fraction of the max pooled prevalence. Default 0.97 (near the right edge).

`plot_title`, `plot_subtitle`
Optional plot labels for the title/subtitle.

`x_label`, `y_label`
Optional axis labels. Defaults are generated from the group labels.

Details

The plot places each feature (peptide) as a point at:

- x-axis: pooled prevalence `(prop1 + prop2)/2`
- y-axis: prevalence shift `(prop2 - prop1)`

Points are optionally jittered for visibility. A dashed horizontal line marks $\Delta = 0$. Optional arrows and labels indicate the direction of increased prevalence for `group1` vs `group2`. If `add_smooth = TRUE`, a GAM smooth is overlaid to summarize the trend across pooled prevalence.

Value

A `ggplot` object.

Examples

```
set.seed(1)
n <- 40
prev_tbl <- data.frame(
  feature = paste0("pep", seq_len(n)),
  group1  = "A",
  group2  = "B",
  prop1   = runif(n),
  prop2   = runif(n)
)

p <- deltaplot(
  prev_tbl,
  group_pair_values = c("A", "B"),
  group_labels      = c("Group A", "Group B")
)
print(p)
```

`deltaplot_interactive` *Interactive Delta-prevalence vs Pooled Prevalence*

Description

Build an interactive plotly chart showing the per-peptide shift in prevalence ($\Delta = group2 - group1$) as a function of pooled prevalence ($(group1 + group2)/2$). The input should be a tibble/data frame produced by `ph_compute_prevalence()` or equivalent with columns `group1`, `group2`, `prop1`, and `prop2`.

Usage

```
deltaplot_interactive(
  prev_tbl,
  group_pair_values = NULL,
  group_labels = NULL,
  point_alpha = 0.6,
  point_size = 6,
  add_smooth = TRUE,
  smooth_k = 5,
  arrow_color = "red",
  arrow_x_frac = 0.97,
  arrow_length_frac = 0.3,
  label_x_gap_frac = 0.03,
  label_y_gap_frac = 0.02,
  plot_title = NULL,
  plot_subtitle = NULL,
  point_jitter_width = 0.005,
  point_jitter_height = 0.005
)
```

Arguments

`prev_tbl`	Data frame with columns `group1`, `group2`, `prop1`, `prop2`. Optional `feature` is used for labels.
`group_pair_values`	Optional length-2 character vector `c(group1, group2)`. Use this when `prev_tbl` contains multiple group pairs.
`group_labels`	Optional length-2 character vector of display labels `c(label_group1, label_group2)`. Defaults to `group1`/`group2`.
`point_alpha`	Point transparency. Default 0.6.
`point_size`	Point size. Default 6.
`add_smooth`	Add a GAM smooth curve (`mgcv`). Default `TRUE`.
`smooth_k`	Basis dimension k for the smooth. Default 5.
`arrow_color`	Color for the directional arrows and labels. Default `"red"`.

`arrow_x_frac` Arrow X position as a fraction of the x-range. Default 0.97.

`arrow_length_frac`
: Arrow length as a fraction of the y-range. Default 0.30.

`label_x_gap_frac`
: Horizontal label offset as a fraction of the x-range.

`label_y_gap_frac`
: Vertical label offset as a fraction of the y-range.

`plot_title`, `plot_subtitle`
: Optional plot labels for the title/subtitle.

`point_jitter_width`, `point_jitter_height`
: Jitter amounts. Defaults 0.005.

Details

The plot places each feature (peptide) as a point at:

- x-axis: pooled prevalence `(prop1 + prop2)/2`
- y-axis: prevalence shift `(prop2 - prop1)`

Points are optionally jittered for display, and hover text includes the feature identifier plus prevalence (percent and proportion) and counts (`n1`, `N1`, `n2`, `N2`) when available in the input table. A dashed horizontal line marks $\Delta = 0$. Optional arrows and labels indicate the direction of increased prevalence for `group1` vs `group2`. If `add_smooth = TRUE`, a GAM smooth is overlaid to summarize the trend.

Value

A plotly object.

Examples

```
## Not run:
set.seed(2)
n <- 40
prev_tbl <- data.frame(
  feature = paste0("pep", seq_len(n)),
  group1  = "A",
  group2  = "B",
  prop1   = runif(n),
  prop2   = runif(n)
)

p <- deltaplot_interactive(
  prev_tbl,
  group_pair_values = c("A", "B"),
  group_labels      = c("Group A", "Group B"),
  add_smooth        = FALSE
)
p

## End(Not run)
```

ecdf_plot *ECDF of Per-peptide Prevalences for Two Groups*

Description

Plot empirical cumulative distribution functions (ECDFs) of per-peptide prevalence for two groups using a ph_compute_prevalence()-style table. The plot compares the cumulative distribution of prevalence values between the two groups and optionally annotates median shifts and a Kolmogorov-Smirnov (KS) test summary.

Usage

```
ecdf_plot(
  prev_tbl,
  group_pair_values = NULL,
  group_labels = NULL,
  line_width_pt = 1,
  line_alpha = 1,
  group1_line_color = "#1f77b4",
  group2_line_color = "#d62728",
  show_median_lines = TRUE,
  show_ks_test = TRUE,
  plot_title = NULL,
  plot_subtitle = NULL,
  x_label = NULL,
  y_label = NULL
)
```

Arguments

prev_tbl	Data frame with columns group1, group2, prop1, prop2. Optional feature columns are ignored.
group_pair_values	Optional length-2 character vector c(group1,group2). Use this when prev_tbl contains multiple group pairs.
group_labels	Optional length-2 character vector of display labels c(label_group1, label_group2). Defaults to group1/group2.
line_width_pt	Line width for ECDF steps (ggplot units). Default 1.0.
line_alpha	Line alpha for ECDF steps. Default 1.0.
group1_line_color, group2_line_color	Line colors for group1 and group2.
show_median_lines	Logical; add median lines. Default TRUE.
show_ks_test	Logical; add KS test summary to subtitle. Default TRUE.

plot_title, plot_subtitle
: Optional plot labels.

x_label, y_label
: Optional axis labels.

Details

Each group is represented by a step function showing the fraction of features with prevalence less than or equal to a given value. Vertical median lines can be added for each group, and the subtitle can include the KS statistic and p-value along with the median difference.

Value

A ggplot object.

Examples

```
set.seed(4)
n <- 50
prev_tbl <- data.frame(
  feature = paste0("pep", seq_len(n)),
  group1  = "A",
  group2  = "B",
  prop1   = runif(n),
  prop2   = runif(n)
)

p <- ecdf_plot(
  prev_tbl,
  group_pair_values = c("A", "B"),
  group_labels      = c("Group A", "Group B"),
  show_ks_test      = TRUE
)
print(p)
```

---

ecdf_plot_interactive *ECDF of Per-peptide Prevalence for Two Groups*

Description

Plotly version of ecdf_prevalence(), showing ECDF curves for two groups based on per-peptide prevalence values. The plot can annotate median shifts and an optional KS test summary.

Usage

```
ecdf_plot_interactive(
  prev_tbl,
  group_pair_values = NULL,
  group_labels = NULL,
```

```
  line_width_px = 2,
  line_alpha = 1,
  group1_line_color = "#1f77b4",
  group2_line_color = "#d62728",
  show_median_lines = TRUE,
  show_ks_test = TRUE,
  plot_title = NULL,
  plot_subtitle = NULL
)
```

Arguments

`prev_tbl`	Data frame with columns `group1`, `group2`, `prop1`, `prop2`.
`group_pair_values`	Optional length-2 character vector `c(group1, group2)`. Use this when `prev_tbl` contains multiple group pairs.
`group_labels`	Optional length-2 character vector of display labels `c(label_group1, label_group2)`. Defaults to `group1`/`group2`.
`line_width_px`	Line width for ECDF steps (plotly units). Default 2.0.
`line_alpha`	Line alpha for ECDF steps. Default 1.0.
`group1_line_color`, `group2_line_color`	Line colors for group1 and group2.
`show_median_lines`	Logical; add median lines. Default `TRUE`.
`show_ks_test`	Logical; add KS test summary to subtitle. Default `TRUE`.
`plot_title`, `plot_subtitle`	Optional plot labels.

Details

Each group is represented by a step curve showing the cumulative fraction of features with prevalence less than or equal to a given value. Vertical median lines can be added for each group, and the subtitle can include the KS statistic and p-value with the median difference.

Value

A plotly object.

Examples

```
## Not run:
set.seed(5)
n <- 50
prev_tbl <- data.frame(
  feature = paste0("pep", seq_len(n)),
  group1  = "A",
  group2  = "B",
  prop1   = runif(n),
```

```
  prop2   = runif(n)
)

p <- ecdf_plot_interactive(
  prev_tbl,
  group_pair_values = c("A", "B"),
  group_labels      = c("Group A", "Group B"),
  show_ks_test      = TRUE
)
p

## End(Not run)
```

`forestplot` *Forest Plot of Top/Bottom Raw Stouffer T by Rank*

Description

Builds a forest plot highlighting the most extreme positive and negative DELTA/Stouffer statistics within a chosen rank. The plot shows the top `n_pos_each` features on the positive side and the top `n_neg_each` features on the negative side, ordered by the selected statistic (`statistic_to_plot`). This provides a compact summary of which features shift most strongly between the two groups for a given rank.

Usage

```
forestplot(
  results_tbl,
  rank_of_interest,
  statistic_to_plot = c("T", "T_stand", "Z_from_p"),
  n_neg_each = 15,
  n_pos_each = 15,
  filter_significant = "none",
  sig_level = 0.05,
  left_label = "More in group1",
  right_label = "More in group2",
  arrow_length_frac = 0.35,
  label_x_gap_frac = 0.06,
  y_pad = 0.6,
  label_y_offset = 0,
  label_vjust = -0.3,
  arrow_color = "red",
  arrow_linewidth = 0.6,
  arrow_head_length_mm = 3,
  use_diverging_colors = FALSE,
  base_text_pt = 12,
  font_family = "Montserrat",
  seg_width = 1.2,
```

```
  point_size = 3.6,
  show_grid = FALSE
)
```

Arguments

results_tbl	Data frame/tibble with at least: rank, feature,group1, group2, design, T_obs, p_perm.
rank_of_interest	Character scalar specifying the rank to plot (e.g., "species").
statistic_to_plot	Which statistic to rank/plot: "T" (raw T_obs), "T_stand" (permutation-standardized), or "Z_from_p" (signed Z from permutation p).
n_neg_each	Number of most negative features to show. Default 15.
n_pos_each	Number of most positive features to show. Default 15.
filter_significant	Column name to filter on, or "none" to disable filtering. If the column is numeric, keep rows where col <= sig_level.
sig_level	Significance threshold used when filter_significant is numeric. Default 0.05.
left_label	Text for the left arrow/side label.
right_label	Text for the right arrow/side label.
arrow_length_frac	Fraction of max $\|T\|$ used as half-length of arrows.
label_x_gap_frac	Horizontal gap for arrow labels beyond arrow tips (fraction of max $\|T\|$).
y_pad	Vertical padding above the top category for arrows/labels (y-axis units).
label_y_offset	Additional vertical offset for arrow-end labels (y-axis units).
label_vjust	Vertical justification for arrow labels (passed to annotate()).
arrow_color	Arrow/label color.
arrow_linewidth	Arrow line width.
arrow_head_length_mm	Arrow head length in mm.
use_diverging_colors	Logical; if TRUE, lines/points are shaded blue (negative) to red (positive) with higher contrast; otherwise monochrome.
base_text_pt	Base text size in points for the plot.
font_family	Font family for plot text.
seg_width	Segment line width for the horizontal bars.
point_size	Point size for feature markers.
show_grid	Logical; show major/minor grid lines.

Details

The y-axis lists feature names (sorted by the chosen statistic), and the x-axis shows the signed effect size for each feature. Each feature is drawn as a horizontal segment from zero to its statistic value, with a point at the end of the segment. A dashed vertical line marks zero to separate negative from positive shifts. The subtitle reports the contrast (group1 vs group2) and how many negative/positive features are shown.

If `use_diverging_colors = TRUE`, segments/points are colored by magnitude on a blue-to-red scale (negative to positive), otherwise a single color is used. The arrow annotations at the top label the direction of enrichment for each group and help interpret the sign of the statistic.

`statistic_to_plot` controls the statistic used for both ranking and plotting: raw `T_obs`, permutation-standardized `T_obs_stand`, or `Z_from_p` (signed Z from permutation p-values). For calibrated inference or cross-figure comparability, prefer permutation Z-based results or T_stand.

Value

A list with:

- `data` - tibble used for plotting (selected top/bottom items),
- `plot` - ggplot object.

Examples

```
# in this example we mock the output of compute_delta with a simple
# data.frame - it works the same
set.seed(1)
n <- 20
results_tbl <- data.frame(
  rank = rep("species", n),
  feature = paste0("feat_", seq_len(n)),
  group1 = "control",
  group2 = "treated",
  design = "case-control",
  T_obs = rnorm(n, sd = 2),
  p_perm = runif(n),
  T_obs_stand = rnorm(n),
  Z_from_p = qnorm(1 - runif(n) / 2) * sign(rnorm(n))
)

out <- forestplot(
  results_tbl,
  rank_of_interest = "species",
  statistic_to_plot = "T",
  n_neg_each = 5,
  n_pos_each = 5,
  left_label = "More in control",
  right_label = "More in treated",
  use_diverging_colors = TRUE,
  font_family = "sans"
)
```

```
print(out$plot)
```

`forestplot_interactive` *Interactive Forest Plot of Top/Bottom DELTA/Stouffer Statistics*

Description

Plotly version of `forestplot()` that highlights the most extreme positive and negative features within a chosen rank. The plot shows the top `n_pos_each` features on the positive side and the top `n_neg_each` features on the negative side, ordered by the selected statistic (`statistic_to_plot`).

Usage

```
forestplot_interactive(
  results_tbl,
  rank_of_interest,
  statistic_to_plot = c("T", "T_stand", "Z_from_p"),
  n_neg_each = 15,
  n_pos_each = 15,
  filter_significant = "none",
  sig_level = 0.05,
  left_label = "More in group1",
  right_label = "More in group2",
  arrow_length_frac = 0.35,
  label_x_gap_frac = 0.06,
  label_y_offset = 0,
  arrow_color = "red",
  arrow_linewidth = 0.6,
  arrow_head_length_mm = 3,
  use_diverging_colors = FALSE,
  show_grid = FALSE,
  base_text_pt = 12,
  font_family = "Montserrat",
  seg_width = 1.6,
  point_size = 11
)
```

Arguments

`results_tbl`	Data frame/tibble with at least: `rank`, `feature`,`group1`, `group2`, `design`, `T_obs`, `p_perm`.
`rank_of_interest`	Character scalar specifying the rank to plot (e.g., `"species"`).
`statistic_to_plot`	Which statistic to rank/plot: `"T"` (raw `T_obs`), `"T_stand"` (permutation-standardized), or `"Z_from_p"` (signed Z from permutation p).

`n_neg_each` Number of most negative features to show. Default 15.

`n_pos_each` Number of most positive features to show. Default 15.

`filter_significant`
: Column name to filter on, or `"none"` to disable filtering. If the column is numeric, keep rows where `col <= sig_level`.

`sig_level` Significance threshold used when `filter_significant` is numeric. Default 0.05.

`left_label` Text for the left arrow/side label.

`right_label` Text for the right arrow/side label.

`arrow_length_frac`
: Fraction of max $|T|$ used as half-length of arrows.

`label_x_gap_frac`
: Horizontal label gap for arrow labels beyond arrow tips (fraction of max $|T|$).

`label_y_offset` Additional vertical offset for arrow-end labels (y-axis units).

`arrow_color` Arrow/label color.

`arrow_linewidth`
: Arrow line width (plotly units).

`arrow_head_length_mm`
: Arrow head size (approximate, plotly units).

`use_diverging_colors`
: Logical; if `TRUE`, lines/points are shaded blue (negative) to red (positive) with higher contrast; otherwise monochrome.

`show_grid` Logical; show grid lines.

`base_text_pt` Base text size.

`font_family` Font family name.

`seg_width` Segment line width.

`point_size` Point size.

Details

The y-axis lists feature names (sorted by the chosen statistic), and the x-axis shows the signed effect size for each feature. Each feature is drawn as a horizontal segment from zero to its statistic value, with a point at the end of the segment. A dashed vertical line marks zero to separate negative from positive shifts. The title and subtitle report the contrast and how many features are shown.

If `use_diverging_colors = TRUE`, segments/points are colored by magnitude on a blue-to-red scale (negative to positive), otherwise a single color is used. Arrow annotations label the direction of enrichment for each group.

`statistic_to_plot` controls the statistic used for both ranking and plotting: raw `T_obs`, permutation-standardized `T_obs_stand`, or `Z_from_p` (signed Z from permutation p-values).

Value

A list with `data` and `plot` (plotly object).

Examples

```
set.seed(1)
n <- 20
results_tbl <- data.frame(
  rank = rep("species", n),
  feature = paste0("feat_", seq_len(n)),
  group1 = "control",
  group2 = "treated",
  design = "case-control",
  T_obs = rnorm(n, sd = 2),
  p_perm = runif(n),
  T_obs_stand = rnorm(n),
  Z_from_p = qnorm(1 - runif(n) / 2) * sign(rnorm(n))
)

out <- forestplot_interactive(
  results_tbl,
  rank_of_interest = "species",
  statistic_to_plot = "T",
  n_neg_each = 5,
  n_pos_each = 5,
  left_label = "More in control",
  right_label = "More in treated",
  use_diverging_colors = TRUE,
  label_x_gap_frac = 0,
  label_y_offset = -0.05
)

out$plot
```

---

get_example_path *Path to Example PhIP-Seq Datasets*

Description

Return the path to an example dataset shipped with phiperio, suitable for use with `load_example_data` or `convert_standard`.

Usage

```
get_example_path(name = c("phip_mixture"))
```

Arguments

`name`	Character scalar. Name of the example dataset. Currently supported: `"phip_mixture"`.

Value

A character scalar with an absolute path to the file.

Examples

```
sim_path <- get_example_path("phip_mixture")
# phip_obj <- convert_standard(sim_path)
```

---

load_example_data *Load Example PhIP-Seq Dataset as <phip_data>*

Description

Convenience helper to quickly load a shipped example dataset ("phip_mixture") into a <phip_data> object, suitable for downstream analysis and visualization. This function wraps convert_standard, automatically supplying the correct parameters for the included example data.

Usage

```
load_example_data(name = c("phip_mixture", "small_mixture"))
```

Arguments

name	Character scalar. Name of the shipped example dataset. Currently supported: "phip_mixture", "small_mixture".

Value

A <phip_data> object created from the chosen example dataset.

Examples

```
# Load the example data shipped with the package:
ex <- load_example_data()
# ex is now a <phip_data> object ready for analysis

# Specify the dataset name explicitly
ex2 <- load_example_data("small_mixture")
```

phip_palette — *PHIP default colour palette*

Description

A concise, publication-ready qualitative palette tailored for PHIP figures. Colors are stable across sessions to keep styling reproducible.

Usage

```
phip_palette
```

Format

A character vector of hex colors.

plot_alpha — *Plot alpha diversity (richness/Shannon/Simpson) from precomputed results*

Description

Plot alpha diversity metrics from the precomputed output of `compute_alpha()`. Supports filtering groups/ranks and optional faceting by rank. If the input contains an interaction table, you can request only that via `interaction_only = TRUE`.

Usage

```
plot_alpha(
  x,
 metric = c("richness", "shannon_diversity", "simpson_diversity", "pielou_evenness",
    "berger_parker_dominance"),
  group_col = "group",
  rank_col = "rank",
  filter_groups = NULL,
  filter_ranks = NULL,
  custom_colors = NULL,
  facet_by_rank = TRUE,
  ncol = 2,
  facet_scales = "fixed",
  interaction_only = FALSE,
  interaction_sep = NULL,
  jitter_width = 0.25,
  point_size = 1.8,
  point_alpha = 0.7,
```

```
  text_size = 12,
  font_family = "Montserrat",
  show_grids = TRUE,
  x_order = NULL,
  x_labels = NULL,
  y_range = NULL,
  x_tickangle = 0,
  significance = NULL,
  show_significance = FALSE,
  sig_p_threshold = 0.05,
  sig_step_increase = 0.05,
  sig_tip_length = 0.01,
  ...
)
```

Arguments

x	a "phip_alpha_diversity" list (output of compute_alpha()) or a single alpha-diversity data frame.
metric	one of "richness", "shannon_diversity", "simpson_diversity", "pielou_evenness", or "berger_parker_dominance".
group_col	name of the grouping column in the alpha table (default "group" when group_cols = NULL in the computation step).
rank_col	name of the rank column (default "rank").
filter_groups	optional character vector; keep only these group levels.
filter_ranks	optional character vector; keep only these ranks.
custom_colors	optional named vector of colors for groups.
facet_by_rank	logical; facet by rank_col if multiple ranks present.
ncol	integer; number of columns in facet wrap (default 2).
facet_scales	"fixed", "free_x", "free_y", or "free".
interaction_only	logical; if TRUE, plot only the interaction table (when available in x). useful when compute_alpha() was run with group_interaction = TRUE.
interaction_sep	character; separator used to join interaction labels. default is taken from attr(x, "interaction_sep") if present, otherwise " * ".
jitter_width	Numeric; horizontal jitter width for points.
point_size	Numeric; size of jittered points.
point_alpha	Numeric in (0,1); alpha for jittered points.
text_size	Numeric; base text size for plot labels.
font_family	Character; font family for plot text.
show_grids	Logical; whether to show panel grid lines.
x_order	Optional character vector specifying the x-axis order.

x_labels	Optional named character vector mapping x-axis labels.
y_range	Optional numeric length-2 vector for y-axis limits.
x_tickangle	Numeric; x-axis label angle in degrees.
significance	Optional "phip_alpha_significance" object from compute_alpha_significance(); used to add significance brackets when show_significance = TRUE.
show_significance	Logical; if TRUE and significance is supplied, add pairwise significance brackets via ggsignif (package must be installed). Default FALSE.
sig_p_threshold	Numeric; only pairs with p_adj <= sig_p_threshold receive a bracket. Default 0.05.
sig_step_increase	Numeric; vertical step between stacked brackets. Passed to ggsignif::geom_signif(). Default 0.05.
sig_tip_length	Numeric; length of bracket tips. Default 0.01.
...	Reserved for future extensions; ignored.

Details

- x can be:
  - the named list returned by compute_alpha() (class "phip_alpha_diversity"), or
  - a single data frame taken from that list.
- when interaction_only = TRUE, the function tries to select the element named by paste(attr(x, "group_cols"), collapse = interaction_sep). if not found, it falls back to the first element whose name contains the separator.

Value

a ggplot object.

Examples

```
## Not run:
# precomputed alpha (list) -> boxplot per group
p <- plot_alpha(alpha_list, metric = "richness", group_col = "Cohort")

# only the interaction table (if available)
p_int <- plot_alpha(
  alpha_list,
  metric = "shannon_diversity",
  group_col = "Cohort * timepoint",
  interaction_only = TRUE
)

## End(Not run)
```

plot_alpha_interactive *Plot alpha diversity (precomputed) — interactive (plotly)*

Description

native plotly version of `plot_alpha()`. mirrors the ggplot look: per-group boxplots with jittered points and optional faceting by rank.

Usage

```
plot_alpha_interactive(
  x,
 metric = c("richness", "shannon_diversity", "simpson_diversity", "pielou_evenness",
    "berger_parker_dominance"),
  group_col = "group",
  rank_col = "rank",
  filter_groups = NULL,
  filter_ranks = NULL,
  custom_colors = NULL,
  facet_by_rank = TRUE,
  ncol = 2,
  facet_scales = "fixed",
  interaction_only = FALSE,
  interaction_sep = NULL,
  x_order = NULL,
  x_labels = NULL,
  y_range = NULL,
  x_tickangle = 0,
  quartile_method = c("exclusive", "inclusive", "linear"),
  jitter_width = 0.25,
  point_size = 6,
  point_alpha = 0.85,
  text_size = 12,
  font_family = "Montserrat",
  show_grids = TRUE,
  ...
)
```

Arguments

x	a "phip_alpha_diversity" list (output of compute_alpha()) or a single alpha-diversity data frame.
metric	one of "richness", "shannon_diversity", "simpson_diversity", "pielou_evenness", or "berger_parker_dominance".
group_col	name of the grouping column in the alpha table (default "group" when group_cols = NULL in the computation step).

rank_col	name of the rank column (default "rank").
filter_groups	optional character vector; keep only these group levels.
filter_ranks	optional character vector; keep only these ranks.
custom_colors	named character vector of hex colors for groups (like ggplot scale_fill_manual()).
facet_by_rank	logical; facet by rank_col if multiple ranks present.
ncol	integer; number of columns in facet wrap (default 2).
facet_scales	"fixed", "free_x", "free_y", or "free".
interaction_only	logical; if TRUE, plot only the interaction table (when available in x). useful when compute_alpha() was run with group_interaction = TRUE.
interaction_sep	character; separator used to join interaction labels. default is taken from attr(x, "interaction_sep") if present, otherwise " * ".
x_order	optional character vector with desired group order (levels).
x_labels	optional named character vector mapping group -> label (used on x axis).
y_range	optional numeric length-2; y axis range (e.g., c(0, 2300)).
x_tickangle	numeric; tick label rotation in degrees (default 0; e.g., 25).
quartile_method	one of c("exclusive","inclusive","linear"); passed to plotly box (default "exclusive" ~ ggplot).
jitter_width	Numeric; horizontal jitter width for points.
point_size	Numeric; size of jittered points.
point_alpha	Numeric in (0,1); alpha for jittered points.
text_size	Numeric; base text size for plot labels.
font_family	Character; font family for plot text.
show_grids	Logical; whether to show panel grid lines.
...	Reserved for future extensions; ignored.

Value

A plotly htmlwidget.

Examples

```
## Not run:
plot_alpha_interactive(df, metric = "richness", group_col = "group")

## End(Not run)
```

plot_alpha_significance *Visualise alpha diversity significance results*

Description

Summarises the pairwise significance table from `compute_alpha_significance()` either as a filtered tibble (`type = "table"`) or a Cohen's d heatmap with significance annotations (`type = "heatmap"`).

Usage

```
plot_alpha_significance(
  x,
  metric = NULL,
  rank = NULL,
  type = c("table", "heatmap"),
  p_threshold = 0.05,
  ...
)
```

Arguments

`x`	A `"phip_alpha_significance"` object from `compute_alpha_significance()`.
`metric`	Character scalar; metric to display. Defaults to the first metric present in `x`.
`rank`	Character scalar; rank to display. Defaults to the first rank present in `x`. Ignored when the data has no `rank` column.
`type`	One of `"table"` (default) or `"heatmap"`.
`p_threshold`	Numeric; pairs with `p_adj > p_threshold` are shown in grey in the heatmap and excluded from the table. Default `0.05`.
`...`	Reserved; ignored.

Value

- `type = "table"`: a `tibble::tibble` of significant pairwise results.
- `type = "heatmap"`: a ggplot heatmap (Cohen's d fill, stars overlaid).

Examples

```
## Not run:
sig <- compute_alpha_significance(alpha_list)
plot_alpha_significance(sig, type = "table")
plot_alpha_significance(sig, type = "heatmap")

## End(Not run)
```

plot_cap *Plot CAP/db-RDA Results (Constrained Ordination)*

Description

Creates a 2D scatter plot from a "beta_capscale" object (output of compute_capscale()), showing sample scores on CAP axes. Points can be coloured by group and shaped by (categorical) time. Optional ellipses and centroids can be overlaid for group/time/group*time summaries.

Usage

```
plot_cap(
  cap_res,
  axes = c(1, 2),
  group_col = NULL,
  time_col = NULL,
  show_centroids = TRUE,
  centroid_by = c("auto", "group", "time", "group_time"),
  connect_centroids = c("none", "group", "time"),
  show_ellipses = TRUE,
  ellipse_by = c("group"),
  ellipse_type = c("t", "norm", "euclid"),
  ellipse_level = 0.95,
  point_size = 1.5,
  point_alpha = 0.6,
  centroid_size = 3
)
```

Arguments

cap_res	A "beta_capscale" object as returned by compute_capscale(). Must contain at least a sample_coords tibble with columns sample_id and CAP1, CAP2, ..., and a numeric eigenvalues vector for constrained axes.
axes	Integer vector of length 2 giving the CAP axes to plot (e.g., c(1, 2) for CAP1 vs CAP2). Default is c(1, 2).
group_col	Optional name of a grouping column in cap_res$sample_coords used to colour points (e.g., "group", "type_person"). If NULL, the function auto-detects "group" if present.
time_col	Optional name of a **categorical** time column in cap_res$sample_coords used to shape points (e.g., "time" or "timepoint"). If NULL, the function auto-detects "time" if present. Continuous time is not supported in this plotting function.
show_centroids	Logical; if TRUE (default), plots centroids for groups / time levels / group*time combinations depending on centroid_by.

`centroid_by` How to define centroids. One of `"auto"`, `"group"`, `"time"`, `"group_time"`. `"auto"` uses: group*time if both factors are available, otherwise group if present, otherwise time.

`connect_centroids`
How to connect centroids with lines. One of `"none"` (default), `"group"`, `"time"`. The logic is:

- if centroids are by `group_time` and `connect_centroids = "group"`, centroids of each group are connected along time levels;
- if centroids are by `group_time` and `connect_centroids = "time"`, centroids of each time level are connected across groups;
- for other `centroid_by` values, connections are ignored.

`show_ellipses` Logical; if `TRUE` (default), draws ellipses for selected factors.

`ellipse_by` Character vector describing which ellipses to draw. Any combination of `"group"`, `"time"`, `"group_time"`, or `"none"` (to disable). Default is `"group"`.

`ellipse_type` Ellipse type passed to `ggplot2::stat_ellipse()`, one of `"t"`, `"norm"`, `"euclid"`. Default `"t"`.

`ellipse_level` Numeric confidence level for ellipses (default `0.95`).

`point_size` Numeric size of sample points. Default `1.5`.

`point_alpha` Numeric opacity of sample points in `[0, 1]`. Default `0.6`.

`centroid_size` Numeric size for centroid points. Default `3`.

Details

Axis labels are annotated with the percentage of constrained variance explained by each CAP axis, computed as $100 * \lambda_k / \sum_j \lambda_j$, where λ_k are the constrained eigenvalues from `cap_res$eigenvalues`. Only positive parts of eigenvalues are used when computing percentages.

Styled with `theme_phip()`.

Typically you will want to join sample-level metadata into `cap_res$sample_coords` before plotting, e.g.:

```
cap_res$sample_coords <- dplyr::left_join(
  cap_res$sample_coords,
  meta,             # data frame with sample_id, group, time, etc.
  by = "sample_id"
)
```

Value

A ggplot2::ggplot object representing the CAP ordination.

Examples

```
## Not run:
  # cap_res <- compute_capscale(dist_bc, ps = ps, formula = ~ type_person + age)
  cap_res$sample_coords <- dplyr::left_join(
    cap_res$sample_coords,
```

```
  meta,  # contains type_person, timepoint, etc.
  by = "sample_id"
)

plot_cap(
  cap_res,
  axes            = c(1, 2),
  group_col       = "type_person",
  time_col        = "timepoint",
  show_centroids  = TRUE,
  centroid_by     = "group_time",
  connect_centroids = "group",
  show_ellipses   = TRUE,
  ellipse_by      = c("group", "group_time")
)

## End(Not run)
```

`plot_dispersion` *Plot beta dispersion (distance to centroid)*

Description

Plot distances to group/time centroids from a `beta_dispersion` object (output of `compute_dispersion()`) as violins, hollow boxplots and/or jittered points, by level on the x-axis.

Usage

```
plot_dispersion(
  x,
  scope = "group",
  contrast = "<global>",
  show_violin = TRUE,
  show_box = TRUE,
  show_points = TRUE,
  point_size = 1.5,
  point_alpha = 0.6,
  width_violin = 0.9,
  width_box = 0.5,
  jitter_width = 0.1,
  jitter_height = 0,
  add_pvalue = TRUE
)
```

Arguments

`x`	A `beta_dispersion` object returned by `compute_dispersion()`.

scope	Character scalar indicating which dispersion scope to plot. Typically one of "group", "time" or "group:time". Must match values in x$distances$scope. Default "group".
contrast	Character scalar indicating which contrast within scope to plot, e.g. "<global>" or "A vs B". Must match values in x$distances$contrast. Default "<global>".
show_violin	Logical; if TRUE (default) draw a violin layer.
show_box	Logical; if TRUE (default) draw hollow boxplots (fill = NA) per level.
show_points	Logical; if TRUE (default) draw jittered points.
point_size	Numeric point size for jitter layer. Default 1.5.
point_alpha	Numeric alpha for jitter layer. Default 0.6.
width_violin	Width of violin layer. Default 0.9.
width_box	Width of boxplot layer. Default 0.5.
jitter_width, jitter_height	Jitter width/height for points. Defaults: 0.1 and 0.
add_pvalue	Logical; if TRUE (default) and a matching row is found in x$tests, the p-value is shown in the subtitle.

Value

A ggplot object.

Examples

```
## Not run:
  disp_res <- compute_dispersion(dist_bc, ps, group_col = "group_char")

  # Global group dispersion:
  plot_dispersion(disp_res, scope = "group", contrast = "<global>")

  # Pairwise contrast, with only boxplots + points:
  plot_dispersion(
    disp_res,
    scope        = "group",
    contrast     = "dementia vs control",
    show_violin  = FALSE,
    show_box     = TRUE,
    show_points  = TRUE
  )

## End(Not run)
```

plot_enrichment_counts *Plot enrichment counts per group (and optional interaction)*

Description

Visualizes per-sample peptide enrichment counts across groups in a <phip_data> object, with optional interaction of multiple grouping variables. Presence is defined by exist > 0. The plot(s) are produced by an internal helper .plot_enrichment_counts_one().

Usage

```
plot_enrichment_counts(
  phip_data,
  group_cols = NULL,
  prevalence_threshold = 0.05,
  custom_colors = NULL,
  binwidth = 1,
  group_interaction = FALSE,
  interaction_only = FALSE,
  interaction_sep = " * ",
  annotation_size = 4,
  ...
)
```

Arguments

phip_data	A <phip_data> object with data_long containing at least sample_id, peptide_id, and exist.
group_cols	Character vector of grouping columns in data_long, or NULL to plot all samples together.
prevalence_threshold	Numeric in [0,1]; minimum prevalence used by .plot_enrichment_counts_one() to filter/annotate bins (default 0.05).
custom_colors	Optional named vector for group colors passed through to .plot_enrichment_counts_one() (default NULL).
binwidth	Numeric bin width for histograms (default 1).
group_interaction	Logical; also compute a plot for the interaction of all group_cols (default FALSE).
interaction_only	Logical; if TRUE, return only the interaction plot (requires group_interaction = TRUE and at least two group_cols).
interaction_sep	Character separator for interaction labels (default " * ").

annotation_size
: Numeric; size of the in-plot threshold annotations (passed to `geom_text(size = ...)`). Typical range 3–6. Default 4.

...
: Reserved for future extensions; ignored.

Details

- If `group_cols = NULL`, a single plot is returned for all samples.
- If `group_cols` is a character vector, a list of plots is returned (one per grouping column) unless `interaction_only = TRUE`.
- If `group_interaction = TRUE` and at least two `group_cols` are supplied, an additional interaction plot is created whose label joins groups using `interaction_sep`.
- If `interaction_only = TRUE`, only the interaction plot is returned (this requires `group_interaction = TRUE` and at least two `group_cols`).

Value

- A single plot object (when `group_cols = NULL`, or when only one plot is produced), or
- A named list of plot objects (when multiple plots are produced).

Examples

```
# per-group plots
pd <- load_example_data()
p <- plot_enrichment_counts(pd, group_cols = c("group","timepoint"))

# add interaction plot
p2 <- plot_enrichment_counts(pd,
  group_cols = c("group","timepoint"),
  group_interaction = TRUE
)

# interaction only
p3 <- plot_enrichment_counts(pd,
  group_cols = c("group","timepoint"),
  group_interaction = TRUE,
  interaction_only = TRUE
)
```

---

plot_pcoa *Plot Principal Coordinates Analysis (PCoA)*

Description

Creates a 2D PCoA scatterplot from a `"beta_pcoa"` object (the output of `compute_pcoa()`), with optional grouping by a categorical variable, optional time factor, centroids, centroid trajectories and ellipses.

Usage

```
plot_pcoa(
  pcoa_res,
  axes = c(1, 2),
  group_col = NULL,
  time_col = NULL,
  show_centroids = TRUE,
  centroid_by = c("auto", "group", "time", "group_time"),
  connect_centroids = c("none", "group", "time"),
  show_ellipses = TRUE,
  ellipse_by = c("group"),
  ellipse_type = c("t", "norm", "euclid"),
  ellipse_level = 0.95,
  point_size = 1.5,
  point_alpha = 0.6,
  centroid_size = 3
)
```

Arguments

pcoa_res	A "beta_pcoa" object as returned by compute_pcoa(). Must contain at least a sample_coords component with columns sample_id and PCoA1, PCoA2, ..., and a var_explained component (one-row tibble) with columns such as "%PCoA1", "%PCoA2", etc.
axes	Integer vector of length 2 giving the PCoA axes to plot (e.g., c(1, 2) for PCoA1 vs PCoA2). Defaults to c(1, 2).
group_col	Optional name of the grouping column in pcoa_res$sample_coords (e.g., "group", "type_person"). Used for colouring points and for group-level centroids / ellipses. If NULL, the function attempts to auto-detect "group" if present.
time_col	Optional name of a **categorical** time column in pcoa_res$sample_coords (e.g., "time" or "timepoint"). Used for shaping points and for time/interaction centroids / ellipses. If NULL, the function attempts to auto-detect "time" if present. Continuous time is **not** supported here.
show_centroids	Logical; if TRUE (default), centroids of the chosen grouping are shown.
centroid_by	Granularity of centroids. One of: • "auto" (default): if both group_col and time_col are present, centroids are computed for each group-time combination ("group_time"); otherwise by "group" or "time" depending on which factor is available. • "group": centroids per group (requires group_col). • "time": centroids per time level (requires time_col). • "group_time": centroids per group*time combination (requires both group_col and time_col).
connect_centroids	Character; if centroids are shown and centroid_by = "group_time", this controls how they are joined by paths: • "none" (default): no connecting paths;

- "group": connect centroids along time within each group;
- "time": connect centroids across groups within each time level.

For other centroid_by values, connections are ignored.

show_ellipses	Logical; if TRUE (default), confidence ellipses (via ggplot2::stat_ellipse()) are drawn for selected factors.
ellipse_by	Character vector specifying which factor(s) to use for ellipses. Possible values are "group", "time", "group_time". Default is "group". Values that require a missing factor are ignored.
ellipse_type	Type of ellipse passed to ggplot2::stat_ellipse(), one of "t", "norm", "euclid". Defaults to "t".
ellipse_level	Numeric confidence level for ellipses (default 0.95).
point_size	Numeric size of points. Default 1.5.
point_alpha	Numeric alpha (opacity) of points in [0, 1]. Default 0.6.
centroid_size	Numeric size of centroid points. Default 3.

Details

The function expects that any grouping / time variables you wish to use (e.g., "type_person", "age_group", "timepoint") have already been merged into pcoa_res$sample_coords (typically by joining with a metadata table via sample_id).

Axis labels are automatically annotated with the percentage of variance explained, using the var_explained component of pcoa_res if available.

Styled with theme_phip().

Value

A ggplot2::ggplot object representing the PCoA scatter plot.

Examples

```
## Not run:
  # pcoa_res <- compute_pcoa(dist_bc)
  # Suppose you have metadata with columns sample_id, type_person, timepoint:
  meta_sc <- dplyr::left_join(
    pcoa_res$sample_coords,
    meta,
    by = "sample_id"
  )
  pcoa_res$sample_coords <- meta_sc

  # Basic PCoA coloured by type_person with group-level ellipses:
  plot_pcoa(
    pcoa_res,
    axes        = c(1, 2),
    group_col   = "type_person",
    time_col    = "timepoint",
    ellipse_by  = "group",
```

```
    show_centroids = TRUE,
    centroid_by    = "group_time",
    connect_centroids = "group"
  )

## End(Not run)
```

`plot_scree` *Scree Plot for PCoA Eigenvalues*

Description

Creates a scree plot from a `"beta_pcoa"` object (output of `compute_pcoa()`), showing the percentage of variance explained for the first `n_axes` axes.

Usage

```
plot_scree(pcoa_res, n_axes = NULL, type = c("bar", "line"))
```

Arguments

`pcoa_res`	A `"beta_pcoa"` object as returned by `compute_pcoa()`. Must contain an `eigenvalues` component (numeric vector of eigenvalues).
`n_axes`	Integer giving the number of axes to display. If `NULL` (default), the function uses `min(10, length(eigenvalues))`.
`type`	Type of scree plot to draw: `"bar"` (default) for a bar plot of variance explained, or `"line"` for a line/point plot.

Details

Percentages are computed from the positive part of the eigenvalues, i.e. `pmax(eigenvalues, 0)`, to handle possible small negative eigenvalues from non-perfectly Euclidean distance matrices.

Styled with `theme_phip()`.

Value

A ggplot2::ggplot object representing the scree plot.

Examples

```
## Not run:
  # pcoa_res <- compute_pcoa(dist_bc)
  plot_scree(pcoa_res)

  # More axes as line plot:
  plot_scree(pcoa_res, n_axes = 15, type = "line")

## End(Not run)
```

plot_tsne *Plot t-SNE embeddings*

Description

Plot t-SNE embeddings computed by `compute_tsne()`. Supports both 2D (ggplot2) and 3D (plotly) views.

Usage

```
plot_tsne(
  tsne_res,
  view = c("2d", "3d"),
  colour = NULL,
  size = 1.5,
  alpha = 0.8,
  palette = NULL,
  ...
)
```

Arguments

`tsne_res`	A `phip_tsne` object returned by `compute_tsne()`. The function also works with a plain tibble/data frame that contains at least columns `tSNE1` and `tSNE2` (and `tSNE3` for 3D).
`view`	Character, either `"2d"` or `"3d"`. `"2d"` returns a `ggplot` object; `"3d"` returns a `plotly` HTML widget.
`colour`	Optional name of a column in `tsne_res` to map to point colour. If `NULL`, the function uses the first metadata column stored in `attr(tsne_res, "meta_cols")`, if available.
`size`	Numeric point size for scatter plots. Defaults to `1.5`.
`alpha`	Numeric transparency for points (0-1). Defaults to `0.8`.
`palette`	Optional vector of colour values passed to `scale_color_manual()` (2D) or `colors` (3D plotly).
`...`	Currently ignored; reserved for future extensions.

Value

For `view = "2d"`, a `ggplot` object. For `view = "3d"`, a `plotly` object (`htmlwidget`).

Examples

```
## Not run:
tsne_res <- compute_tsne(ps, compute_distance(ps))

# 2D plot
```

```
p2d <- plot_tsne(tsne_res, view = "2d", colour = "type_person")

# 3D interactive plot
p3d <- plot_tsne(tsne_res, view = "3d", colour = "type_person")

## End(Not run)
```

`scale_colour_phip` *Discrete colour scale using the PHIP palette*

Description

A thin wrapper around `ggplot2::scale_colour_manual()` that applies `phip_palette`. Set as the session default on `library(phiper)`.

Usage

```
scale_colour_phip(...)

scale_color_phip(...)
```

Arguments

`...` Arguments passed on to `discrete_scale`

`limits` One of:

- `NULL` to use the default scale values
- A character vector that defines possible values of the scale and their order
- A function that accepts the existing (automatic) values and returns new ones. Also accepts rlang lambda function notation.

`drop` Should unused factor levels be omitted from the scale? The default, `TRUE`, uses the levels that appear in the data; `FALSE` includes the levels in the factor. Please note that to display every level in a legend, the layer should use `show.legend = TRUE`.

`na.translate` Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify `na.translate = FALSE`.

`name` The name of the scale. Used as the axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.

`minor_breaks` One of:

- `NULL` for no minor breaks
- `waiver()` for the default breaks (none for discrete, one minor break between each major break for continuous)
- A numeric vector of positions

- A function that given the limits returns a vector of minor breaks. Also accepts rlang lambda function notation. When the function has two arguments, it will be given the limits and major break positions.

`labels` One of the options below. Please note that when `labels` is a vector, it is highly recommended to also set the `breaks` argument as a vector to protect against unintended mismatches.

- `NULL` for no labels
- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)
- An expression vector (must be the same length as breaks). See ?plotmath for details.
- A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

`guide` A function used to create a guide or its name. See `guides()` for more information.

`call` The `call` used to construct the scale for reporting messages.

`super` The super class to use for the constructed scale

Value

A ggplot2 scale.

See Also

Other php-ggplot: `scale_fill_phip()`, `theme_phip()`

Examples

```
ggplot2::ggplot(iris,
  ggplot2::aes(Sepal.Length, Sepal.Width, colour = Species)) +
  ggplot2::geom_point() +
  scale_colour_phip()
```

---

scale_fill_phip *Discrete fill scale using the PHIP palette*

Description

A thin wrapper around `ggplot2::scale_fill_manual()` that applies `phip_palette`. Set as the session default on `library(phiper)`.

Usage

```
scale_fill_phip(...)
```

Arguments

`...` Arguments passed on to `discrete_scale`

`limits` One of:

- `NULL` to use the default scale values
- A character vector that defines possible values of the scale and their order
- A function that accepts the existing (automatic) values and returns new ones. Also accepts rlang lambda function notation.

`drop` Should unused factor levels be omitted from the scale? The default, `TRUE`, uses the levels that appear in the data; `FALSE` includes the levels in the factor. Please note that to display every level in a legend, the layer should use `show.legend = TRUE`.

`na.translate` Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify `na.translate = FALSE`.

`name` The name of the scale. Used as the axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.

`minor_breaks` One of:

- `NULL` for no minor breaks
- `waiver()` for the default breaks (none for discrete, one minor break between each major break for continuous)
- A numeric vector of positions
- A function that given the limits returns a vector of minor breaks. Also accepts rlang lambda function notation. When the function has two arguments, it will be given the limits and major break positions.

`labels` One of the options below. Please note that when `labels` is a vector, it is highly recommended to also set the `breaks` argument as a vector to protect against unintended mismatches.

- `NULL` for no labels
- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)
- An expression vector (must be the same length as breaks). See ?plotmath for details.
- A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

`guide` A function used to create a guide or its name. See `guides()` for more information.

`call` The `call` used to construct the scale for reporting messages.

`super` The super class to use for the constructed scale

Value

A ggplot2 scale.

See Also

Other phip-ggplot: scale_colour_phip(), theme_phip()

Examples

```
ggplot2::ggplot(iris,
  ggplot2::aes(Species, Sepal.Length, fill = Species)) +
  ggplot2::geom_boxplot() +
  scale_fill_phip()
```

---

scatter_interactive *Interactive prevalence scatter for prevalence results*

Description

Creates an interactive scatter (plotly) comparing prevalence in **group a** vs **group b** for per-feature results produced by compute_pop(). Accepts a plain data.frame with columns percent1, percent2, feature, group1, group2, p_raw, etc.

When pair is given, subsetting uses .ph_filter_pairs().

Color mapping:

- Default (color_by = NULL): BH-corrected p-values computed per-plot from p_raw ("significant (BH)", "nominal only", "not significant").
- color_by as a named vector highlights points matching the specified peptide-library values:

```
color_by = c("is_flagellum" = TRUE)
color_by = c("species" = "Staphylococcus aureus")
color_by = c("is_flagellum" = TRUE, "species" = "Staphylococcus aureus")
```

Usage

```
scatter_interactive(
  df,
  pair = NULL,
  rank = NULL,
  xlab = NULL,
  ylab = NULL,
  alpha = 0.05,
  color_by = NULL,
  color_title = NULL,
  peplib = NULL,
  background_df = NULL,
  ...
)
```

Arguments

df	A data.frame with columns percent1, percent2, feature, group1, group2, p_raw, optionally n_peptides, rank, peptide_id.
pair	optional length-2 character, e.g. c("kid_serum::T2","kid_serum::T8").
rank	optional single rank (character) to keep.
xlab, ylab	axis labels; defaults to pair[1]/pair[2] when pair is given.
alpha	numeric in (0,1]; significance threshold; not the plotly alpha.
color_by	optional named vector identifying peptide-library values to highlight, e.g. c("is_flagellum" = TRUE) or c("species" = "Staphylococcus aureus").
color_title	optional legend title when color_by is used.
peplib	Optional peptide metadata table used to resolve color_by when not available via the global library.
background_df	Optional data frame of background points.
...	graphical parameters: category_colors, show_background, background_name, background_color, background_size, background_alpha, background_max_n, background_seed, point_line_width, point_line_color, point_size, point_alpha, jitter_width_pp, jitter_height_pp, font_family, font_size.

Value

a plotly object.

Examples

```
## Not run:
p <- scatter_interactive(scatters,
  pair        = c("kid_serum::T2", "kid_serum::T8"),
  rank        = "peptide_id",
  color_by    = c("is_flagellum" = TRUE),
  color_title = "Flagellum"
)
p

## End(Not run)
```

---

scatter_static *Static scatterplot of percent1 vs percent2 from* compute_pop()

Description

A ggplot2 scatterplot comparing prevalence in **group a** vs **group b**. Default coloring uses BH-corrected p-values computed per-plot from `p_raw`: "significant (BH)", "nominal only", "not significant". If only one category is present the plot falls back to p-value bins.

When `color_by` is supplied as a named vector, peptide metadata is joined and points matching the specified values are highlighted. Multiple groups may be given simultaneously:

```
color_by = c("is_flagellum" = TRUE)
color_by = c("species" = "Staphylococcus aureus")
color_by = c("is_flagellum" = TRUE, "species" = "Staphylococcus aureus")
```

Usage

```
scatter_static(
  df,
  pair = NULL,
  rank = NULL,
  xlab = NULL,
  ylab = NULL,
  alpha = 0.05,
  color_by = NULL,
  color_title = NULL,
  ...
)
```

Arguments

`df`	A data frame with prevalence results.
`pair`	optional group pair (character length-2).
`rank`	optional single rank (character) to keep.
`xlab`, `ylab`	axis labels; defaults to `pair[1]`/`pair[2]` when `pair` is given.
`alpha`	numeric in (0,1]; significance threshold for category labels.
`color_by`	optional named vector identifying peptide-library values to highlight, e.g. `c("is_flagellum" = TRUE)` or `c("species" = "Staphylococcus aureus")`.
`color_title`	optional legend title when `color_by` is used.
`...`	graphical parameters: `point_size` (default 2), `point_alpha` (default 0.85), `jitter_width_pp` (default 0), `jitter_height_pp` (default 0), `font_family`, `font_size` (default 12).

Value

A ggplot object.

Examples

```
set.seed(1)
prev <- data.frame(
  rank       = "peptide_id",
  feature    = paste0("pep", 1:30),
  group1     = "A",
  group2     = "B",
  prop1      = runif(30),
  prop2      = runif(30),
  percent1   = runif(30, 0, 100),
  percent2   = runif(30, 0, 100),
  ratio      = runif(30, 0.1, 10),
  p_raw      = runif(30),
  n_peptides = 1L
)

# basic plot
scatter_static(prev)

# filter to a specific pair and set axis labels
scatter_static(prev,
  pair  = c("A", "B"),
  xlab  = "Group A (%)",
  ylab  = "Group B (%)",
  alpha = 0.05
)
```

---

theme_phip *Theme* theme_phip

Description

A clean, publication-ready ggplot2 theme tuned for **facetted** plots with the **Montserrat** font. The font is registered and **showtext** rendering is enabled automatically when the package loads — no setup required.

Usage

```
theme_phip(base_size = 14, base_family = "Montserrat")
```

Arguments

base_size	Base font size.
base_family	Base font family (default "Montserrat").

Value

A ggplot2 theme object.

See Also

Other phip-ggplot: `scale_colour_phip()`, `scale_fill_phip()`

Examples

```
## Not run:
ggplot2::ggplot(iris, ggplot2::aes(Sepal.Length, Sepal.Width, colour = Species)) +
  ggplot2::geom_point() +
  scale_colour_phip() +
  theme_phip()

## End(Not run)
```

`volcano_interactive` *Interactive volcano plot (log2 ratio vs -log10 p)*

Description

Interactive volcano plot (log2 ratio vs -log10 p)

Usage

```
volcano_interactive(
  df,
  pair = NULL,
  rank = NULL,
  color_by = NULL,
  color_title = NULL,
  fc_cut = 1,
  p_cut = 0.05,
  p_mode = c("raw", "bh"),
 significant_colors = c(`not significant` = "#386cb0", `significant prior correction` =
    "#1b9e77", `significant post fdr correction` = "#e31a1c")
)
```

Arguments

`df`	A data frame with prevalence results.
`pair`	optional group pair (character length-2).
`rank`	optional single rank (character) to keep.
`color_by`	optional named vector identifying peptide-library values to highlight, e.g. `c("is_flagellum" = TRUE)`.
`color_title`	optional legend title when `color_by` is used.
`fc_cut`	Numeric; absolute log2 fold-change cutoff.
`p_cut`	Numeric; p-value cutoff.
`p_mode`	One of `c("raw","bh")`; `"bh"` applies BH correction per-plot from `p_raw`.
`significant_colors`	Named vector of colors for significance categories.

Value

A plotly htmlwidget.

Examples

```
## Not run:
set.seed(3)
prev <- data.frame(
  rank       = "peptide_id",
  feature    = paste0("pep", 1:40),
  group1     = "A",
  group2     = "B",
  prop1      = runif(40),
  prop2      = runif(40),
  percent1   = runif(40, 0, 100),
  percent2   = runif(40, 0, 100),
  ratio      = runif(40, 0.1, 10),
  p_raw      = c(runif(10, 0, 0.01), runif(30, 0.1, 1)),
  n_peptides = 1L
)

# interactive volcano — hover to inspect each peptide
volcano_interactive(prev)

# BH correction
volcano_interactive(prev, p_mode = "bh", fc_cut = 1.5, p_cut = 0.01)

## End(Not run)
```

---

volcano_static *Static volcano plot (log2 ratio vs -log10 p)*

Description

Static volcano plot (log2 ratio vs -log10 p)

Usage

```
volcano_static(
  df,
  pair = NULL,
  rank = NULL,
  color_by = NULL,
  color_title = NULL,
  fc_cut = 1,
  p_cut = 0.05,
  p_mode = c("raw", "bh"),
 significant_colors = c(`not significant` = "#386cb0", `significant prior correction` =
    "#1b9e77", `significant post fdr correction` = "#e31a1c")
)
```

Arguments

df	A data frame with prevalence results.
pair	optional group pair (character length-2).
rank	optional single rank (character) to keep.
color_by	optional named vector identifying peptide-library values to highlight, e.g. c("is_flagellum" = TRUE).
color_title	optional legend title when color_by is used.
fc_cut	Numeric; absolute log2 fold-change cutoff.
p_cut	Numeric; p-value cutoff.
p_mode	One of c("raw","bh"); "bh" applies BH correction per-plot from p_raw.
significant_colors	Named vector of colors for significance categories.

Value

A ggplot object.

Examples

```
set.seed(2)
prev <- data.frame(
  rank       = "peptide_id",
  feature    = paste0("pep", 1:40),
  group1     = "A",
  group2     = "B",
  prop1      = runif(40),
  prop2      = runif(40),
  percent1   = runif(40, 0, 100),
  percent2   = runif(40, 0, 100),
  ratio      = runif(40, 0.1, 10),
  p_raw      = c(runif(10, 0, 0.01), runif(30, 0.1, 1)),
  n_peptides = 1L
)

# basic volcano
volcano_static(prev)

# BH correction, custom cutoffs
volcano_static(prev, fc_cut = 1.5, p_cut = 0.01, p_mode = "bh")

# filter to one pair
volcano_static(prev, pair = c("A", "B"), rank = "peptide_id")
```

Index