

Package ‘vennDiagramLab’

October 6, 2026

Type Package

Title Headless Venn Diagram Analysis and Rendering

Version 2.9.0

Date 2026-10-05

Description Headless companion to the 'Venn Diagram Lab' web tool (<https://www.venndiagramlab.org/>). Build, render, and statistically analyze Venn / 'UpSet' diagrams from 'CSV' / 'TSV' / 'GMT' / 'GMX' inputs. Provides the same 44 SVG models, intersection / 'Jaccard' / hypergeometric statistics, and PDF report layout as the web tool, with byte-equivalent 'TSV' exports (parity-tested against the published Python package). Integrates with 'ggplot2', 'tidygraph', and 'broom'.

License MIT + file LICENSE

Language en-US

URL <https://zoliqua.github.io/Venn-Diagram-Lab/r/>,
<https://github.com/ZoliQua/Venn-Diagram-Lab>,
<https://www.venndiagramlab.org/>

BugReports <https://github.com/ZoliQua/Venn-Diagram-Lab/issues>

Depends R (>= 4.2)

Imports methods, stats, utils, jsonlite, xml2, ggplot2, ComplexUpset, ggraph, tidygraph, grDevices, rsvg, patchwork, gridExtra, openxlsx, zip, BiocGenerics

Suggests testthat (>= 3.0.0), knitr, rmarkdown, rprojroot, pdftools, svglite, broom, tibble, dplyr, covr, BiocCheck, lintr, pkgdown

VignetteBuilder knitr

biocViews Visualization, GeneSetEnrichment, Software

Encoding UTF-8

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

RoxygenNote 8.0.0

Collate 'classes.R' 'errors.R' 'analysis.R' 'broom-tidy.R' 'cluster.R'
 'data-quality.R' 'enrichment-plot-helpers.R' 'excel-workbook.R'
 'ggplot-layer.R' 'io.R' 'json-export.R' 'network-export.R'
 'proportional.R' 'region-accessors.R' 'region-expression.R'
 'render-enrichment-bar.R' 'render-enrichment-lollipop.R'
 'render-network.R' 'render-pdf.R' 'render-share-distribution.R'
 'render-svg-items.R' 'render-svg.R' 'render-upset.R'
 'samples.R' 'share-distribution.R' 'statistics.R'
 'tsv-export.R' 'zip-report.R' 'zzz.R'

NeedsCompilation no

Author Zoltán Dul [aut, cre] (ORCID: <<https://orcid.org/0000-0002-9523-3450>>),
 Márton Ölbei [aut] (ORCID: <<https://orcid.org/0000-0002-4903-6237>>),
 N. Shaun B. Thomas [aut],
 Azeddine Si Ammour [aut] (ORCID:
 <<https://orcid.org/0000-0002-5504-4444>>),
 Attila Csikász-Nagy [aut] (ORCID:
 <<https://orcid.org/0000-0002-2919-5601>>)

Maintainer Zoltán Dul <zoltan.dul@gmail.com>

Repository CRAN

Date/Publication 2026-10-06 08:30:02 UTC

Contents

analyze	3
analyze_data_quality	4
augment.RegionResult	6
bh_fdr	7
circle_intersection_area	7
cluster_set_order	8
compute_pairwise	9
dice	10
effective_universe	10
exclusive_items	11
fold_enrichment	12
fold_enrichment_ci	12
generate_proportional_svg	13
geom_venn	14
glance.RegionResult	15
hypergeometric_p_value	16
intersection_items	17
item_share_distribution	18
jaccard	18
list_models	19
list_samples	20
load_csv	20
load_gmt	21

load_gmx	21
load_sample	22
load_tsv	23
one_vs_rest_enrichment	23
overlap_coefficient	25
parse_region_expression	25
RegionData-class	26
RegionResult-class	27
render_cluster_heatmap	27
render_enrichment_bar	28
render_enrichment_lollipop	29
render_network	30
render_share_distribution	32
render_upset	32
render_venn_svg	34
solve_2set	35
solve_3set	36
statistics	37
StatisticsResult-class	38
SvgImage-class	38
tidy.RegionResult	39
to_excel_workbook	40
to_matrix_tsv	41
to_network_graphml	42
to_network_sif	43
to_one_vs_rest_tsv	44
to_pdf_report	45
to_region_summary_tsv	46
to_result_json	47
to_statistics_tsv	48
to_zip_report	49
union_items	50
validate_dataset	51
vdl_version	52
VennDataset-class	52

Index 53

analyze	<i>Analyze a Venn diagram dataset</i>
---------	---------------------------------------

Description

Compute the Venn region map for a [`VennDataset-class`] and bind it to a model.

Usage

```
analyze(dataset, model = "auto")
```

Arguments

dataset A [`'VennDataset-class'`] (from one of the `'load_*'` functions).

model Model identifier. `"auto"` picks the canonical model for the dataset's set count (alphabetical first match), e.g. 4 sets -> `'venn-4-set'`. `"proportional"` requests an area-proportional layout (only supports 2-3 sets, added in Phase 3). Otherwise pass an explicit name from `[list_models()]`.

Value

A [`'RegionResult-class'`] with the per-region item membership, set sizes, and (lazily) `'statistics(result)'`.

Examples

```
ds <- methods::new("VennDataset",
  set_names = c("A", "B"),
  items = list(A = c("x", "y"), B = c("y", "z")),
  item_order = c("x", "y", "z"),
  universe_size = 10L, source_path = NULL, format = "csv")
result <- analyze(ds)
result@model

ds <- load_sample("dataset_real_cancer_drivers_4")
result <- analyze(ds, model = "auto")
result@model
```

`analyze_data_quality` *Analyze a parsed table for data-quality issues*

Description

Read-only, pure companion to `[.binary_columns_to_dataset()]`/`[.aggregated_columns_to_dataset()]`. Runs the *same* scanning rules those loaders use internally (including their real dedup-via-`'unique()'` and blank-identifier-row skipping) and reports what would be silently collapsed or skipped, instead of collapsing/skipping it. It never mutates `'headers'`/`'rows'`, never lower-cases an item before recording it, and never calls into `'VennDataset'` construction – a purely descriptive pass.

Usage

```
analyze_data_quality(
  headers,
  rows,
  mode = c("binary", "aggregated"),
  prefix_cols = 1L
)
```

Arguments

headers	Character vector of column headers (as returned by <code>‘.parse_table()\$headers’</code>).
rows	List of character vectors, one per data row (as returned by <code>‘.parse_table()\$rows’</code>).
mode	<code>“binary”</code> (default) or <code>“aggregated”</code> .
prefix_cols	Number of leading metadata columns in binary mode (default 1). Only column 1 is ever read as the item identifier, matching <code>‘.binary_columns_to_dataset()’</code> . Ignored when <code>‘mode = “aggregated”</code> .

Details

Mirroring R’s real loader behaviour (not TS/Python byte-parity)

* **Aggregated mode***: like `[‘.aggregated_columns_to_dataset()’]`, a whole trimmed cell is treated as *one* item – there is no delimiter-splitting in R’s real loader, so none happens here either. * **Binary mode***: like `[‘.binary_columns_to_dataset()’]`, only column 1 is read as the item identifier (any extra `‘prefix_cols’` beyond the first are metadata the loader itself never reads). Rows whose identifier is blank after trimming are skipped *entirely* – not scored for duplicates, case collisions, or empty cells – exactly mirroring `‘valid_idx’` in `‘.binary_columns_to_dataset()’`. Duplicates are scored *per set column* (mirroring the loader’s own `‘unique()’` applied independently to each set’s truthy identifiers), not as one flat id-column entry – R has no “row contributes if any column is truthy” concept the way the TS array-based loader does, so unlike TS/Python, ALL non-blank-id rows count toward case-collision scope regardless of whether any of their set cells are truthy (this matches `‘item_order_seen <- unique(item_ids[valid_idx])’`, which is computed before/independent of the truthy test).

Value

A list with: * `‘duplicates_removed’` – list of `‘list(column, column_name, count, examples)’` entries, one per column that has duplicates (columns without duplicates are omitted). `‘column’` is the 1-based index into `‘headers’`. `‘examples’` holds up to 5 item strings, each captured at its 2nd occurrence, in encounter order. * `‘empty_cells_skipped’` – single integer count of blank cells found while scanning (scope depends on mode; see divergences above). * `‘case_collisions’` – list of `‘list(items = character())’` entries, each holding 2+ distinct case-sensitive spellings that share a lower-cased form, in first-appearance order. Purely descriptive: item identity is never folded or merged anywhere in this function. * `‘has_warnings’` – `‘TRUE’` if any of the above found something.

Examples

```
headers <- c("Gene", "SetA", "SetB")
rows <- list(c("g1", "1", "0"), c("g1", "1", "1"), c("", "1", "0"))
report <- analyze_data_quality(headers, rows, mode = "binary")
report$has_warnings
```

augment.RegionResult *Augment method for RegionResult (broom-compatible)*

Description

Returns one row per item in the dataset's universe, with boolean columns indicating set membership and a 'region_label' column naming the exact region (e.g. "A", "AB", "ABC") the item belongs to. Item ordering follows 'dataset@item_order' (first-seen across all sets, JS Set/Map semantics).

Usage

```
## S3 method for class 'RegionResult'
augment(x, ...)
```

Arguments

x	A ['RegionResult-class'].
...	Unused (broom convention).

Details

Region labels use the package's positional letter convention (A-I), matching the labels in 'RegionResult@regions' and the bundled SVG models, regardless of the dataset's 'set_names'.

Value

A tibble (or data.frame fallback) with 'nrow(out) == length(x@dataset@item_order)' and columns: 'item' (character), one logical column per set (named after the set), 'region_label' (character).

Examples

```
ds <- methods::new("VennDataset",
  set_names = c("A", "B"),
  items = list(A = c("x", "y"), B = c("y", "z")),
  item_order = c("x", "y", "z"),
  universe_size = 10L, source_path = NULL, format = "csv")
result <- analyze(ds)
if (requireNamespace("broom", quietly = TRUE)) broom::augment(result)

result <- analyze(load_sample("dataset_real_cancer_drivers_4"))
broom::augment(result)
```

bh_fdr	<i>Benjamini-Hochberg FDR adjustment</i>
--------	--

Description

Wraps 'stats::p.adjust(p, method = "BH")'. Returns adjusted p-values in the same order as the input. Empty input -> empty output.

Usage

```
bh_fdr(p_values)
```

Arguments

p_values	Numeric vector of raw p-values in [0, 1].
----------	---

Value

Numeric vector of adjusted p-values, same length as input.

Examples

```
bh_fdr(c(0.001, 0.01, 0.05, 0.5))
```

circle_intersection_area	<i>Lens-shaped intersection area of two circles</i>
--------------------------	---

Description

Mirrors 'src/utils/proportionalLayout.ts circleIntersectionArea' (web tool) and Python 'circle_intersection_area' byte-for-byte.

Usage

```
circle_intersection_area(r1, r2, d)
```

Arguments

r1	Radius of circle 1 (positive numeric).
r2	Radius of circle 2 (positive numeric).
d	Distance between centers (non-negative numeric).

Value

Numeric: 0 if circles are disjoint, $\pi * \min(r1, r2)^2$ if fully nested, else the lens-shaped intersection area.

Examples

```
circle_intersection_area(1, 1, 1) # ~ 1.228
circle_intersection_area(1, 1, 3) # 0 (disjoint)
```

cluster_set_order	<i>Hierarchical clustering on a symmetric distance matrix.</i>
-------------------	--

Description

Wraps `stats::hclust` to produce a normalized output that mirrors the webtool's pure-JS `clusterSetOrder` and Python `cluster_set_order`.

Usage

```
cluster_set_order(D, linkage = c("average", "complete", "single"))
```

Arguments

D	Symmetric numeric distance matrix (NxN, zero diagonal).
linkage	One of "average" (UPGMA), "complete", "single".

Value

A list with:

- `leaf_order`: 1-based integer vector — left-to-right ordering of original indices. At each internal node the subtree whose minimum original leaf index is smaller is placed on the left (deterministic; mirrors the webtool / Python convention).
- `merges`: data.frame with columns `left`, `right` (0-based cluster ids matching the webtool/Python convention; leaves are 0..N-1, internal nodes are N..2N-2), `height` (linkage distance), `size` (number of leaves in the merged cluster).

Examples

```
D <- matrix(c(0, 0.2, 0.9, 0.2, 0, 0.85, 0.9, 0.85, 0), nrow = 3)
cluster_set_order(D, linkage = "average")
```

compute_pairwise	<i>Compute all 5 pairwise statistical tables</i>
------------------	--

Description

Orchestrator that returns a [`StatisticsResult-class`] populated with Jaccard, Dice, Overlap Coefficient, Fold Enrichment (square NxN matrices) plus a long-form hypergeometric table with BH-FDR adjustment.

Usage

```
compute_pairwise(
  set_names,
  inclusive_sizes,
  pairwise_intersections,
  universe_size
)
```

Arguments

set_names Ordered character vector of set identifiers (e.g. `c("A", "B", "C")`).

inclusive_sizes Named integer vector of inclusive set sizes (`'names(inclusive_sizes)'` matches `'set_names'`).

pairwise_intersections Named list of pair intersection counts. Keys are `"set_a|set_b"` with `set_a` appearing earlier in `'set_names'` than `set_b`.

universe_size Hypergeometric universe N (population size). Integer ≥ 1 .

Value

A [`StatisticsResult-class`] object.

Examples

```
compute_pairwise(
  set_names = c("A", "B"),
  inclusive_sizes = c(A = 10L, B = 8L),
  pairwise_intersections = list("A|B" = 5L),
  universe_size = 100L
)
```

dice	<i>Sorensen-Dice coefficient</i>
------	----------------------------------

Description

Computes $2 * |A \cap B| / (|A| + |B|)$. Returns 0 if both sets are empty (matches web tool convention).

Usage

```
dice(size_a, size_b, intersection)
```

Arguments

size_a	Inclusive size of set A (integer ≥ 0).
size_b	Inclusive size of set B (integer ≥ 0).
intersection	Inclusive intersection size $ A \cap B $.

Value

Numeric in $[0, 1]$.

Examples

```
dice(10, 10, 5)
```

effective_universe	<i>Effective hypergeometric universe size for a RegionResult</i>
--------------------	--

Description

Returns the universe N consistent with the web tool. Binary CSV/TSV datasets get 'dataset@universe_size' (= csv.rows.length, includes all-zero rows); aggregated/GMT/GMX datasets fall back to 'length(item_order)' (= lunion of items).

Usage

```
effective_universe(result)

## S4 method for signature 'RegionResult'
effective_universe(result)
```

Arguments

result	A ['RegionResult-class'].
--------	---------------------------

Value

Integer, the universe size N.

Examples

```
ds <- methods::new("VennDataset",
  set_names = c("A", "B"),
  items = list(A = c("x", "y"), B = c("y", "z")),
  item_order = c("x", "y", "z"),
  universe_size = 10L, source_path = NULL, format = "csv")
result <- analyze(ds)
effective_universe(result)

ds <- load_sample("dataset_real_cancer_drivers_4")
result <- analyze(ds)
effective_universe(result)  # 20000 for binary cancer drivers sample
```

exclusive_items	<i>Items exclusive to a specific set combination</i>
-----------------	--

Description

Returns items in EXACTLY the combination of sets listed in ‘sets’ — that is, in every set in ‘sets’ AND in none of the other sets in the dataset.

Usage

```
exclusive_items(result, sets)
```

Arguments

result	A [‘RegionResult-class’] from [analyze()].
sets	Character vector of set letters (‘‘A’’, ‘‘B’’, ...) or display names (values from ‘result@dataset@set_names’). May mix both.

Value

A character vector of item IDs. Empty character if none.

Examples

```
ds <- methods::new("VennDataset",
  set_names = c("A", "B", "C"),
  items = list(A = c("x", "y"), B = c("y", "z"), C = c("z", "w")),
  item_order = c("x", "y", "z", "w"),
  universe_size = 10L, source_path = NULL, format = "csv")
res <- analyze(ds)
exclusive_items(res, c("A", "B"))  # "y" (in A and B, not in C)
```

fold_enrichment	<i>Fold enrichment (observed / expected ratio)</i>
-----------------	--

Description

Computes $(k * N) / (K * n)$. Returns 0.0 if any denominator is zero (matches web tool convention).

Usage

```
fold_enrichment(N, K, n, k)
```

Arguments

N	Population size (total items in the universe). Integer ≥ 1 .
K	Number of success states in the population (e.g. inclusive A). Integer ≥ 0 .
n	Number of draws (e.g. inclusive B). Integer ≥ 0 .
k	Observed successes (e.g. A intersection B). Integer ≥ 0 .

Value

Numeric (≥ 0 ; can exceed 1 for over-representation).

Examples

```
fold_enrichment(20000, 138, 581, 126)
```

fold_enrichment_ci	<i>Approximate 95% confidence interval for the fold enrichment</i>
--------------------	--

Description

Log-scale Wald interval. Only k is random (K/N is the fixed population fraction), so with $p_{\text{hat}} = k/n$: $SE(\log FE) = SE(\log p_{\text{hat}})$, approximately $\sqrt{1/k - 1/n}$ by the delta method. A Jeffreys-style continuity correction ($k+0.5$ successes, $n+1$ trials) keeps the interval finite at $k = 0$. Bounds are exponentiated back. Byte-parity port of the TypeScript ‘foldEnrichmentCI’ (packages/core/src/statistics.ts); coverage is validated by Monte Carlo under the null in scripts/bio_validation.py.

Usage

```
fold_enrichment_ci(N, K, n, k)
```

Arguments

N	Population size (total items in the universe). Integer ≥ 1 .
K	Number of success states in the population (e.g. inclusive A). Integer ≥ 0 .
n	Number of draws (e.g. inclusive B). Integer ≥ 0 .
k	Observed successes (e.g. A intersection B). Integer ≥ 0 .

Value

Numeric length-2 vector 'c(low, high)' (≥ 0).

Examples

```
fold_enrichment_ci(20000, 138, 581, 126)
```

```
generate_proportional_svg
```

Generate an area-proportional SVG for a 2- or 3-set RegionResult

Description

Circle sizes and inter-circle distances are solved analytically (2-set, exact) or by triangulation (3-set, approximate) so that overlap areas match the requested intersection counts. The returned SVG matches the 44-model schema: ShapeA-I, NameA-I, Count_*, CountSUM_*, Bullet* elements are all present and addressable via xml2.

Usage

```
generate_proportional_svg(
  result,
  width = .PROP_DEFAULT_WIDTH,
  height = .PROP_DEFAULT_HEIGHT
)
```

Arguments

result	A ['RegionResult-class'].
width	Canvas width in pixels (default 600).
height	Canvas height in pixels (default 600).

Value

A 'character' (length 1) with the raw SVG.

Examples

```
tmp <- tempfile(fileext = ".tsv")
writeLines(c("Gene\tSetA\tSetB",
            "GENE1\t1\t0",
            "GENE2\t1\t1",
            "GENE3\t0\t1"), tmp)
ds <- load_tsv(tmp, binary = TRUE)
res <- analyze(ds, model = "proportional")
svg <- generate_proportional_svg(res)
nchar(svg) > 0
```

geom_venn

Embed a rendered Venn diagram as a ggplot2 layer

Description

Returns a list of ggplot2 layers that draw ‘data’ (a [‘RegionResult-class’]) as a rasterized Venn diagram on a unit-square coordinate system, ready to compose with other ggplot2 elements (titles, themes, additional annotations).

Usage

```
geom_venn(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  width_px = .GEOM_VENN_DEFAULT_WIDTH
)
```

Arguments

mapping	Accepted for ggplot2 layer-signature consistency. Currently ignored (the Venn diagram is rendered from ‘data’, not from aesthetic mappings). Reserved for a future Stat-based extension.
data	A [‘RegionResult-class’] (required). The Venn diagram to embed.
stat	Accepted for signature consistency; currently ignored.
position	Accepted for signature consistency; currently ignored.
...	Forwarded to ‘ggplot2::annotation_custom()’ (e.g. ‘xmin’, ‘xmax’, ‘ymin’, ‘ymax’ to position the venn on a non-unit coordinate system).
width_px	Raster width in pixels (default 800). Larger values give sharper output at the cost of memory.

Details

This is a NEW capability – the Python package has no equivalent. It uses the same rasterization pipeline as `[to_pdf_report()]`: render the SVG via `[render_venn_svg()]`, rasterize via `'rsvg::rsvg_nativeraster()'`, and wrap as a `'grid::rasterGrob()'` inside `'ggplot2::annotation_custom()'`.

Value

A list of ggplot2 layers: an `'annotation_custom'` carrying the rasterized Venn, a `'geom_blank'` establishing `'[0, 1] x [0, 1]'` limits, and a `'coord_fixed(ratio = 1)'` so the diagram remains square. Note that `'coord_fixed'` will override any coordinate system the user has already added; add `'geom_venn()'` before other coord layers to avoid a warning.

Examples

```
ds <- methods::new("VennDataset",
  set_names = c("A", "B"),
  items = list(A = c("x", "y"), B = c("y", "z")),
  item_order = c("x", "y", "z"),
  universe_size = 10L, source_path = NULL, format = "csv")
result <- analyze(ds)
if (getRversion() >= "4.6") {
  p <- ggplot2::ggplot() + geom_venn(data = result) + ggplot2::theme_void()
  inherits(p, "ggplot")
}

if (getRversion() >= "4.6") {
  result <- analyze(load_sample("dataset_real_cancer_drivers_4"))
  library(ggplot2)
  ggplot() +
    geom_venn(data = result) +
    theme_void() +
    ggtitle("Cancer driver overlap (4 sources)")
}
```

<code>glance.RegionResult</code>	<i>Glance method for RegionResult (broom-compatible)</i>
----------------------------------	--

Description

Returns a 1-row tibble summarizing the analysis: number of sets, number of non-empty regions, total unique items, hypergeometric universe size, resolved model name, whether the layout is approximate (proportional 3-set), and the count of statistically significant / highly significant pairs (FDR-adjusted $q < 0.05$ / < 0.001).

Usage

```
## S3 method for class 'RegionResult'
glance(x, ...)
```

Arguments

x A [`'RegionResult-class'`].
 ... Unused (broom convention).

Value

A 1-row tibble (or data.frame fallback) with columns: `'n_sets'`, `'n_regions'`, `'n_items'`, `'universe_size'`, `'model'`, `'is_approximate'`, `'n_significant'`, `'n_highly_significant'`.

Examples

```
ds <- methods::new("VennDataset",
  set_names = c("A", "B"),
  items = list(A = c("x", "y"), B = c("y", "z")),
  item_order = c("x", "y", "z"),
  universe_size = 10L, source_path = NULL, format = "csv")
result <- analyze(ds)
if (requireNamespace("broom", quietly = TRUE)) broom::glance(result)

result <- analyze(load_sample("dataset_real_cancer_drivers_4"))
broom::glance(result)
```

hypergeometric_p_value

One-sided hypergeometric p-value (over-representation)

Description

Computes $P(X \geq k)$ where $X \sim \text{Hypergeometric}(N, K, n)$. Returns 1.0 for invalid inputs so the metric is safe to feed into BH-FDR without filtering.

Usage

```
hypergeometric_p_value(N, K, n, k)
```

Arguments

N Population size (total items in the universe). Integer ≥ 1 .
 K Number of success states in the population (e.g. inclusive |A|). Integer ≥ 0 .
 n Number of draws (e.g. inclusive |B|). Integer ≥ 0 .
 k Observed successes (e.g. |A intersection B|). Integer ≥ 0 .

Details

Maps to R's `'phyper(k - 1, K, N - K, n, lower.tail = FALSE)'`. Note that R's `phyper` parameter convention differs from Python's `scipy`: R uses `'m'` for success-in-population and `'n'` for failure-in-population ($= N - K$), where Python uses `'N'` for total population.

Value

Numeric in [0, 1].

Examples

```
hypergeometric_p_value(20000, 138, 581, 126)
```

intersection_items	<i>Items in the intersection of the named sets</i>
--------------------	--

Description

Returns items that appear in every set listed in ‘sets’, regardless of whether they also appear in other (unlisted) sets. For "items in this specific combination only", see [exclusive_items()].

Usage

```
intersection_items(result, sets)
```

Arguments

result	A [‘RegionResult-class’] from [analyze()].
sets	Character vector of set letters (“A”, “B”, ...) or display names (values from ‘result@dataset@set_names’). May mix both.

Value

A character vector of item IDs. Empty character if none.

Examples

```
ds <- methods::new("VennDataset",
  set_names = c("A", "B", "C"),
  items = list(A = c("x", "y"), B = c("y", "z"), C = c("z", "w")),
  item_order = c("x", "y", "z", "w"),
  universe_size = 10L, source_path = NULL, format = "csv")
res <- analyze(ds)
intersection_items(res, c("A", "B")) # "y"
```

item_share_distribution

Item Share Distribution

Description

Per-membership-count item totals.

Usage

```
item_share_distribution(matrix)
```

Arguments

matrix Binary item $\times$ set matrix.

Details

Given a binary item $\times$ set matrix (rows = items, columns = sets, cells in $\{0, 1\}$), returns a named integer vector keyed by $k = 1..n_sets$ giving the number of items belonging to exactly k sets. All bins are present even when their count is zero. Rows that sum to zero are skipped (universe-rule violation; defensive).

Value

Named integer vector with names "1", "2", ..., "n_sets".

Examples

```
m <- matrix(c(
  1, 0, 0,
  1, 1, 0,
  1, 1, 1
), ncol = 3, byrow = TRUE)
item_share_distribution(m)
```

jaccard

Jaccard similarity index

Description

Computes $|A \cap B| / |A \cup B|$. Matches the web tool's convention of returning 0 when both sets are empty (NaN-safe).

Usage

```
jaccard(size_a, size_b, intersection)
```

Arguments

size_a	Inclusive size of set A (integer ≥ 0).
size_b	Inclusive size of set B (integer ≥ 0).
intersection	Inclusive intersection size $ A \cap B $.

Value

Numeric in $[0, 1]$.

Examples

```
jaccard(10, 10, 5)
jaccard(0, 0, 0)
```

list_models	<i>List all bundled Venn diagram models</i>
-------------	---

Description

Returns metadata for the 44 bundled SVG model templates plus the ‘proportional’ synthetic generator (added in Phase 3). Read from JSON region files in ‘inst/extdata/models/json/’.

Usage

```
list_models()
```

Value

A ‘data.frame’ with columns ‘name’ (filename stem), ‘set_count’ (2-9), and ‘display_name’ (from the JSON ‘name’ field). Sorted by ‘(set_count, name)’.

Examples

```
head(list_models())
```

list_samples	<i>List bundled sample dataset names</i>
--------------	--

Description

Returns the names of the 5 bundled sample datasets, sorted alphabetically. Use [load_sample()] to load one.

Usage

```
list_samples()
```

Value

Character vector of 5 sample identifiers.

Examples

```
list_samples()
```

load_csv	<i>Load a delimited file (CSV/TSV) into a ['VennDataset-class']</i>
----------	---

Description

Supports two layouts: * Binary mode (default): one row per item, with 0/1 columns marking membership in each set. The first 'prefix_cols' columns are item metadata; remaining columns are sets. * Aggregated mode ('binary = FALSE'): each column is a set, and cells contain item identifiers. Empty cells are ignored.

Usage

```
load_csv(path, binary = TRUE, delimiter = NULL, prefix_cols = 1L)
```

Arguments

path	Path to the file.
binary	'TRUE' for binary 0/1 mode (default), 'FALSE' for aggregated.
delimiter	Explicit delimiter override. 'NULL' auto-detects from ',', ';', tab, and space.
prefix_cols	Number of leading metadata columns in binary mode (default 1). Ignored when 'binary = FALSE'.

Value

A ['VennDataset-class'].

Examples

```
tmp <- tempfile(fileext = ".csv")
writeLines(c("Gene,SetA,SetB", "G1,1,0", "G2,1,1", "G3,0,1"), tmp)
ds <- load_csv(tmp, binary = TRUE)
ds@set_names
```

load_gmt	<i>Load a GMT (Gene Matrix Transposed) file into a ['VennDataset-class']</i>
----------	--

Description

Each line is one set: 'set_name<TAB>description<TAB>item1<TAB>item2<TAB>...'. Lines with fewer than 3 tab-separated columns or empty set names are skipped.

Usage

```
load_gmt(path)
```

Arguments

path	Path to the .gmt file.
------	------------------------

Value

A ['VennDataset-class'].

Examples

```
tmp <- tempfile(fileext = ".gmt")
writeLines(c("SetA\tdesc\tGENE1\tGENE2\tGENE3",
             "SetB\tdesc\tGENE2\tGENE3\tGENE4"), tmp)
ds <- load_gmt(tmp)
ds@set_names
```

load_gmx	<i>Load a GMX file (transposed GMT) into a ['VennDataset-class']</i>
----------	--

Description

Row 0 = set names, row 1 = descriptions, rows 2+ = items column-aligned.

Usage

```
load_gmx(path)
```

Arguments

path Path to the .gmx file.

Value

A [`'VennDataset-class'`].

Examples

```
tmp <- tempfile(fileext = ".gmx")
writeLines(c("SetA\\tSetB",
            "desc_A\\tdesc_B",
            "GENE1\\tGENE2",
            "GENE2\\tGENE3"), tmp)
ds <- load_gmx(tmp)
length(ds@items)
```

load_sample	<i>Load a bundled sample dataset by name</i>
-------------	--

Description

Sample datasets ship inside the package under `'inst/extdata/samples/'` and cover biological (cancer drivers, MSigDB pathways) and mock (streaming platforms, gene sets) use cases. Use `[list_samples()]` to enumerate.

Usage

```
load_sample(name)
```

Arguments

name Sample identifier from `[list_samples()]`.

Value

A [`'VennDataset-class'`] with the appropriate format and mode applied.

Examples

```
ds <- load_sample("dataset_mock_gene_sets")
length(ds@set_names)

ds <- load_sample("dataset_real_cancer_drivers_4")
analyze(ds)@model
```

load_tsv	<i>Load a tab-separated file into a ['VennDataset-class']</i>
----------	---

Description

Equivalent to `load_csv(path, binary = binary, delimiter = "\t", prefix_cols = prefix_cols)`.

Usage

```
load_tsv(path, binary = TRUE, prefix_cols = 1L)
```

Arguments

path	Path to the file.
binary	'TRUE' for binary 0/1 mode (default), 'FALSE' for aggregated.
prefix_cols	Number of leading metadata columns in binary mode (default 1). Ignored when 'binary = FALSE'.

Value

A ['VennDataset-class'].

Examples

```
tmp <- tempfile(fileext = ".tsv")
writeLines(c("Gene\tSetA\tSetB", "G1\t1\t0", "G2\t1\t1", "G3\t0\t1"), tmp)
ds <- load_tsv(tmp, binary = TRUE)
ds@universe_size
```

one_vs_rest_enrichment	
------------------------	--

One-vs-rest enrichment: each set tested against the union of all OTHER sets

Description

Ported byte-for-byte from the web tool's 'oneVsRestEnrichment' ('packages/core/src/statistics.ts') / Python's 'one_vs_rest_enrichment' ('venn_diagram_lab.statistics'). For each set S, "rest" is the union of the inclusive members of all *other* sets, derived purely from region counts:

Usage

```
one_vs_rest_enrichment(
  set_names,
  inclusive_sizes,
  exclusive_only_sizes,
  union_size,
  universe_size
)
```

Arguments

set_names Ordered character vector of set identifiers.

inclusive_sizes Named integer vector: set name -> inclusive size (K).

exclusive_only_sizes Named integer vector: set name -> items present in exactly that set alone (the single-set region's exclusive count, `excl_S`). Names missing from this vector default to 0.

union_size U: union of ALL sets (sum of exclusive counts over every non-empty region, including multi-set regions).

universe_size N: the hypergeometric sampling universe (same value `[compute_pairwise()]` receives as `'universe_size'`).

Details

* `'U'` (`'union_size'`) = union of ALL sets = sum of the exclusive counts over every one of the $2^n - 1$ region labels (items in ≥ 1 set). Binary mode: `'U'` $\leq$ the dataset's row-count universe (rows may belong to no set). Aggregated mode: `'U'` == universe. * `'K'` = `'inclusive_sizes[[name]]'` (inclusive size of S) * `'excl_S'` = `'exclusive_only_sizes[[name]]'` (items only in S), 0 if absent * `'rest_size'` = `'U - excl_S'` (items in ≥ 1 non-S set) * `'k'` = `'K - excl_S'` (S items also in ≥ 1 other set) * `'N'` (`'universe_size'`) = sampling universe for the hypergeometric test

****Critical:**** `'rest_size'` is derived from `'union_size'` (U), NOT from `'universe_size'` (N). This is what makes the test meaningful: in binary mode $N > U$, so the observed `'k'` sits above the hypergeometric support minimum and the p-value is informative. When $N == U$ (aggregated mode, universe equals union) the p-value is ~ 1 – mathematically honest ("no background to enrich against"), not a bug. See `'superpowers/sdd/task-F6-ts-report.md'` for the full derivation.

Uses the same hypergeometric machinery as `[compute_pairwise()]`. Benjamini-Hochberg FDR is computed over the `'n'` one-vs-rest tests (one per set); Bonferroni = `'min(1, p * n)'`. Returned rows are sorted by p-value ascending (stable), matching the pairwise convention.

Value

A data.frame with columns `'name'`, `'size'`, `'rest_size'`, `'intersection'`, `'expected'`, `'fold_enrichment'`, `'p_value'`, `'p_adjusted'`, `'p_bonferroni'`, `'significant'`, sorted by `'p_value'` ascending.

Examples

```

one_vs_rest_enrichment(
  set_names = c("A", "B"),
  inclusive_sizes = c(A = 10L, B = 8L),
  exclusive_only_sizes = c(A = 5L, B = 3L),
  union_size = 13L,
  universe_size = 100L
)

```

overlap_coefficient	<i>Szymkiewicz-Simpson overlap coefficient</i>
---------------------	--

Description

Computes $|A \cap B| / \min(|A|, |B|)$. Useful when one set is much smaller than the other.

Usage

```
overlap_coefficient(size_a, size_b, intersection)
```

Arguments

size_a	Inclusive size of set A (integer ≥ 0).
size_b	Inclusive size of set B (integer ≥ 0).
intersection	Inclusive intersection size $ A \cap B $.

Value

Numeric in $[0, 1]$.

Examples

```
overlap_coefficient(10, 5, 3)
```

parse_region_expression	<i>Parse a Boolean region expression into bitmasks</i>
-------------------------	--

Description

Translates an expression like `"A & B + !C"` into a sorted integer vector of region bitmasks (each $1..2^{n_sets} - 1$). The output is intended for composition with `[render_venn_svg()]` (`'highlight = ...'`) and with the region-accessor family in user code.

Usage

```
parse_region_expression(expr, n_sets)
```

Arguments

`expr` Character scalar; a Boolean expression in the grammar above.
`n_sets` Integer; number of sets in the diagram (2..9).

Details

Grammar:

- ‘A’, ‘B’, ..., ‘I’ — set atoms (uppercase ASCII).
- ‘&’ — intersection.
- ‘|’ or ‘+’ — union.
- ‘~’ or ‘!’ — complement (unary, against ‘1..2^n_sets - 1’).
- ‘(’, ‘)’ — grouping.

Precedence (highest first): ‘~ !’, ‘&’, ‘| +’. Binary operators are left-associative.

Value

Sorted integer vector of region bitmasks. Empty integer vector when the expression is satisfiable by no region (e.g. “A & ~A”).

Examples

```
parse_region_expression("A & B", n_sets = 3L)      # c(3L, 7L)
parse_region_expression("A + B", n_sets = 3L)      # c(1, 2, 3, 5, 6, 7)
parse_region_expression("A & ~B", n_sets = 3L)     # c(1L, 5L)
parse_region_expression("(A | B) & C", n_sets = 3L) # c(5L, 6L, 7L)
```

RegionData-class	<i>RegionData: one region of a Venn diagram</i>
------------------	---

Description

Returned as elements of ‘RegionResult@regions’. Bitmask convention: bit ‘i’ set means “in set with index ‘i’” in ‘dataset@set_names’.

Slots

`bitmask` Region bitmask (1 to 2^n - 1).
`label` Human-readable label like “AB” or “ABC”.
`set_indices` Integer vector of 0-based set indices in this region.
`set_names` Names of the sets in this region.
`exclusive_items` Items present in exactly these sets.
`inclusive_items` Items present in at least these sets.

RegionResult-class	<i>RegionResult: result of analyze()</i>
--------------------	--

Description

Bundles the input dataset, chosen model, region map, set sizes, and a lazy [`StatisticsResult-class`] accessible via `statistics(result)`.

Slots

`dataset` The input [`VennDataset-class`].

`model` Resolved model name (e.g. `"venn-4-set"` or `"proportional"`).

`regions` Named list keyed by `as.character(bitmask)`, each value a [`RegionData-class`]. Only non-empty regions are stored (sparse for high set counts with few overlaps).

`set_sizes` Named integer vector: set name -> inclusive size.

`is_approximate` `'TRUE'` for the proportional 3-set layout where exact areas can't be achieved with circles.

<code>render_cluster_heatmap</code>	<i>Render a cluster-ordered Jaccard similarity heatmap</i>
-------------------------------------	--

Description

Mirrors the webtool's cluster-aware `buildEnrichmentHeatmapSvg` path. Distance $D = 1 - \text{Jaccard}$ is fed to `[cluster_set_order()]`; the resulting `leaf_order` permutes both axes, and merge heights drive the L-shaped overlays emitted under `<g class="hm-dendro-col">` (top band) and `<g class="hm-dendro-row">` (left band).

Usage

```
render_cluster_heatmap(
  result,
  linkage = c("average", "complete", "single"),
  show_row_dendrogram = TRUE,
  show_col_dendrogram = TRUE,
  dendrogram_fraction = 0.12,
  style = NULL
)
```

Arguments

result	A [<code>'RegionResult-class'</code>] from <code>[analyze()]</code> .
linkage	Linkage method passed to <code>[cluster_set_order()]</code> : one of "average" (UPGMA, default), "complete", "single".
show_row_dendrogram	Logical, default TRUE. When FALSE, the row band is omitted from layout entirely (no reserved gap).
show_col_dendrogram	Logical, default TRUE. Same semantics for the column band.
dendrogram_fraction	Fraction of the grid extent reserved per dendrogram band (default 0.12, minimum effective height 20 pixels). Matches the webtool's <code>dendrogramFraction</code> option.
style	Optional named list of style overrides (reserved for v2.3+; currently ignored).

Details

Pure string construction – no **xml2**.

Value

An [`'SvgImage-class'`] with `'content'`, `'width'`, `'height'` set from the computed layout extents.

Examples

```
ds <- load_sample("dataset_real_cancer_drivers_4")
res <- analyze(ds)
img <- render_cluster_heatmap(res, linkage = "average")
nchar(slot(img, "content")) > 0
```

`render_enrichment_bar` *Render the pairwise enrichment bar chart*

Description

One bar per pairwise statistic, ordered by `'(set_a_index, set_b_index)'`. Bar height is proportional to the chosen `'metric'`:

- `"neglog10fdr"` (default): $-\log_{10}(\text{BH-FDR})$, floor 1e-300.
- `"foldEnrichment"`: $(k \cdot N) / (K \cdot n)$ from `[StatisticsResult-class]`'s `'fold_enrichment'` slot.

Usage

```
render_enrichment_bar(
  result,
  metric = c("neglog10fdr", "foldEnrichment"),
  width = 560L,
  height = 240L
)
```

Arguments

result	A [<code>'RegionResult-class'</code>] from <code>[analyze()]</code> .
metric	Bar-height metric – <code>"neglog10fdr"</code> or <code>"foldEnrichment"</code> .
width, height	Output SVG dimensions in pixels.

Details

Bars use #2e7d32 for significant pairs ($FDR < 0.05$) and #888888 otherwise. Significance markers `****` (< 0.001), `***` (< 0.01), and `**` (< 0.05) appear above each bar.

Pure string construction – no **xml2**.

Value

An [`'SvgImage-class'`] with `'content'`, `'width'`, `'height'`.

Examples

```
ds <- methods::new("VennDataset",
  set_names = c("A", "B"),
  items = list(A = c("x", "y"), B = c("y", "z")),
  item_order = c("x", "y", "z"),
  universe_size = 10L, source_path = NULL, format = "csv")
res <- analyze(ds)
img <- render_enrichment_bar(res)
nchar(slot(img, "content")) > 0
```

render_enrichment_lollipop

Render the pairwise enrichment lollipop chart

Description

Same data and significance scheme as `[render_enrichment_bar()]` but as a stem-and-dot plot: a vertical line rises from the baseline to the metric value, capped by a filled circle whose radius scales with $\sqrt{\text{intersection} / \text{max_intersection}}$ (range 2.5–8 px).

Usage

```
render_enrichment_lollipop(
  result,
  metric = c("neglog10fdr", "foldEnrichment"),
  width = 560L,
  height = 240L
)
```

Arguments

result A [`'RegionResult-class'`] from `[analyze()]`.

metric Stem-height metric – `"neglog10fdr"` or `"foldEnrichment"`.

width, height Output SVG dimensions in pixels.

Details

Pure string construction – no **xml2**.

Value

An [`'SvgImage-class'`] with `'content'`, `'width'`, `'height'`.

Examples

```
ds <- methods::new("VennDataset",
  set_names = c("A", "B"),
  items = list(A = c("x", "y"), B = c("y", "z")),
  item_order = c("x", "y", "z"),
  universe_size = 10L, source_path = NULL, format = "csv")
res <- analyze(ds)
img <- render_enrichment_lollipop(res)
nchar(slot(img, "content")) > 0
```

render_network

Render a force-directed network plot from a RegionResult

Description

Builds a ggraph plot where nodes are sets (sized by inclusive cardinality) and edges are pairwise overlaps (thickness proportional to the chosen metric; blue for FDR-significant edges below `'significance_threshold'`, grey otherwise). Layout uses the deterministic `'stress'` algorithm from graphlayouts.

Usage

```
render_network(
  result,
  edge_metric = "intersection",
  seed = 42L,
  significance_threshold = 0.05,
  node_color_map = NULL
)
```

Arguments

result	A [<code>'RegionResult-class'</code>].
edge_metric	One of <code>"intersection"</code> , <code>"jaccard"</code> , <code>"fold_enrichment"</code> (capped at 20.0), <code>"overlap_coefficient"</code> .
seed	Retained for API compatibility; currently unused. The <code>'stress'</code> layout algorithm is fully deterministic and does not rely on a random seed.
significance_threshold	FDR p_adjusted threshold below which edges are colored as significant (default 0.05).
node_color_map	Optional named character vector mapping letters (<code>"A"</code> , <code>"B"</code> , ...) to fill hex colors. Unspecified letters default to yellow (<code>"#FFF200"</code>).

Details

Idiomatic R port of Python `'render_network'` – same parameter contract, but renders via `ggraph` + `tidygraph` instead of `networkx` + `matplotlib`.

Value

A `'ggplot'` (`ggraph` subclass).

Examples

```
ds <- methods::new("VennDataset",
  set_names = c("A", "B"),
  items = list(A = c("x", "y"), B = c("y", "z")),
  item_order = c("x", "y", "z"),
  universe_size = 10L, source_path = NULL, format = "csv")
result <- analyze(ds)
p <- render_network(result)
inherits(p, "ggplot")

result <- analyze(load_sample("dataset_real_cancer_drivers_4"))
p <- render_network(result, edge_metric = "jaccard")
ggplot2::ggsave(tempfile(fileext = ".png"), p, width = 7, height = 7)
```

render_share_distribution

Render the Item Share Distribution histogram

Description

Vertical bar chart with N bars ($k = 1..N$), tier-gradient fill from #ffe4b5 ($k=1$) to #7e14ff ($k=N$). Mirrors the webtool's buildShareDistributionSvg layout (480x280 viewBox, Tahoma typography, sd-bar CSS class on every rect) so downstream CSS, PDF embed, and cross-package parity assertions can key on the same structure.

Usage

```
render_share_distribution(dataset, style = NULL)
```

Arguments

dataset	A ['VennDataset-class'].
style	Optional named list of style overrides (reserved for v2.3+; currently ignored).

Details

Pure string construction – no **xml2** – so the output is byte-stable.

Value

An ['SvgImage-class'] with 'content', 'width = 480', 'height = 280'.

Examples

```
ds <- load_sample("dataset_real_cancer_drivers_4")
img <- render_share_distribution(ds)
nchar(slot(img, "content")) > 0
```

render_upset

Render an UpSet plot from a RegionResult

Description

Builds a ComplexUpset ggplot showing intersection sizes (top bars), set membership matrix (middle dot grid), and per-set sizes (left bars). Idiomatic R port of Python 'render_upset' – same parameter contract, but renders via ComplexUpset (ggplot2) instead of matplotlib (not a 1:1 port).

Usage

```
render_upset(
  result,
  max_columns = 20L,
  sort_by = c("size", "degree"),
  threshold = 0L,
  color_mode = c("depth", "heatmap", "custom"),
  colors = NULL
)
```

Arguments

<code>result</code>	A [<code>'RegionResult-class'</code>].
<code>max_columns</code>	Maximum number of intersections to display (default 20). Top-N by the active sort.
<code>sort_by</code>	<code>"size"</code> (default – descending) or <code>"degree"</code> (membership count ascending then alphabetical).
<code>threshold</code>	Exclude intersections with size strictly below this value (default 0L = no filter).
<code>color_mode</code>	<code>"depth"</code> (default – viridis on degree), <code>"heatmap"</code> (Reds on size), or <code>"custom"</code> (use the <code>'colors'</code> mapping).
<code>colors</code>	Named character vector mapping intersection LABELS (e.g. <code>"AB"</code>) to fill hex colors when <code>'color_mode = "custom"</code> . Unspecified labels fall back to <code>"#ccc-ccc"</code> .

Value

A `'ggplot'` object (saveable via `'ggplot2::ggsave()'`).

Examples

```
ds <- methods::new("VennDataset",
  set_names = c("A", "B"),
  items = list(A = c("x", "y"), B = c("y", "z")),
  item_order = c("x", "y", "z"),
  universe_size = 10L, source_path = NULL, format = "csv")
result <- analyze(ds)
if (getRversion() >= "4.6") {
  p <- render_upset(result)
  inherits(p, "ggplot")
}

if (getRversion() >= "4.6") {
  result <- analyze(load_sample("dataset_real_cancer_drivers_4"))
  p <- render_upset(result, sort_by = "degree", color_mode = "heatmap")
  ggplot2::ggsave(tempfile(fileext = ".png"), p, width = 8, height = 5)
}
```

render_venn_svg	<i>Render a RegionResult onto its model SVG and return the raw SVG string</i>
-----------------	---

Description

Loads the bundled SVG template for 'result@model' (or the explicit 'model' override), walks the DOM via xml2 to overwrite text content ('Name*', 'Count_*', 'CountSUM_*', 'Title') and inline 'fill:' colors ('Shape*', 'Shape*2' for Euler extras, 'Bullet*'), and serializes back to a string.

Usage

```
render_venn_svg(
  result,
  model = NULL,
  set_names = NULL,
  colors = NULL,
  title = NULL,
  show_names = TRUE,
  show_counts = TRUE,
  show_items = FALSE,
  item_options = NULL,
  highlight = NULL
)
```

Arguments

result	A ['RegionResult-class'].
model	Optional model id override (filename stem). Default = 'result@model'.
set_names	Optional named character vector mapping letters ("A", "B", ...) to display names. Unspecified letters fall back to 'result@dataset@set_names'.
colors	Optional named character vector mapping letters to fill hex colors. Applies to 'BulletX', 'ShapeX', and 'ShapeX2' (Euler extra shapes).
title	Optional title override. If 'NULL', the template's default title text is preserved.
show_names	If 'FALSE', blanks every 'NameA-I' element.
show_counts	If 'FALSE', blanks every 'Count_*' and 'CountSUM_*' element.
show_items	If 'TRUE', replace the per-region count text with the actual item identifiers (rendered as '<tspan>' lines inside each 'Count_*' text node). Default 'FALSE'.
item_options	Named list of overrides for the item-text engine. Recognised keys: 'max_items_per_region' (default 20), 'ncol_items' (default 1), 'truncate_long_names' (default 12; 0 disables), 'line_height' (default 10), 'font_size' (default 8), 'show_counts_with_items' (default 'FALSE'), 'ellipsis' (default "..."). Unknown keys raise a warning.
highlight	Character vector of region labels (e.g. 'c("AB", "ABC")') or an integer vector of region bitmasks (e.g. the output of [parse_region_expression()]). When set, only the listed regions keep their original fill colour; all other set-shapes are desaturated to '#cccccc' at 25% opacity. Default 'NULL' (no spotlight).

Details

For ‘model = "proportional"’, delegates to [generate_proportional_svg()].

Mirrors Python ‘render_venn_svg’ byte-for-byte except for: (a) the return type is ‘character’ instead of an ‘SvgImage’ wrapper class; (b) xml2 may emit slightly different whitespace/attribute ordering than lxml. Functional content (text, fill colors, structure) is identical.

Value

A ‘character’ (length 1) with the raw SVG.

Examples

```
ds <- methods::new("VennDataset",
  set_names = c("A", "B"),
  items = list(A = c("x", "y"), B = c("y", "z")),
  item_order = c("x", "y", "z"),
  universe_size = 10L, source_path = NULL, format = "csv")
result <- analyze(ds)
svg <- render_venn_svg(result)
nchar(svg) > 0

result <- analyze(load_sample("dataset_real_cancer_drivers_4"))
svg <- render_venn_svg(result)
nchar(svg) > 0
```

solve_2set

Area-proportional 2-set circle layout

Description

Solves for two circles whose areas equal ‘a_only + ab’ and ‘b_only + ab’ and whose intersection area equals ‘ab’, by analytical bisection on the inter-center distance. Always exact (returns ‘is_approximate = FALSE’).

Usage

```
solve_2set(a_only, b_only, ab)
```

Arguments

a_only	Items in A only (integer >= 0).
b_only	Items in B only (integer >= 0).
ab	Items in A intersection B (integer >= 0).

Details

Mirrors Python ‘solve_2set’ byte-for-byte.

Value

A named list with elements:

circles Length-2 list of 'list(cx, cy, r)' named lists.

error Relative error of the achieved area fit (typically $< 1e-4$).

is_approximate Always 'FALSE' for 2-set.

Examples

```
solve_2set(a_only = 30L, b_only = 30L, ab = 10L)
```

solve_3set	<i>Area-proportional 3-set circle layout (Wilkinson 2012-style triangulation)</i>
------------	---

Description

Computes pairwise inter-center distances via bisection on the lens intersection area, then places circle C via barycentric triangulation against AB. Always sets 'is_approximate = TRUE' because perfect 3-circle area-proportional fits don't always exist mathematically.

Usage

```
solve_3set(regions)
```

Arguments

regions	Named list keyed by 'as.character(bitmask)' (1..7) -> exclusive count. Missing keys are treated as 0.
---------	---

Details

Mirrors Python 'solve_3set' byte-for-byte (including the 'error = NaN' deliberate sentinel - 3-set fit error is not measured in v0.1).

Value

A named list with elements 'circles' (length 3), 'error' (always NaN in v0.1), 'is_approximate' (always TRUE).

Examples

```
solve_3set(list("1" = 100L, "2" = 80L, "3" = 30L,
               "4" = 60L, "5" = 20L, "6" = 15L, "7" = 5L))
```

statistics	<i>Lazy pairwise statistics for a RegionResult</i>
------------	--

Description

Computes (and on subsequent calls re-computes) the [`'StatisticsResult-class'`] for the pairwise metric tables. R has no built-in `'cached_property'` equivalent for S4 slots, so this is recomputed each call. Cache externally via `'stats <- statistics(result)'` if you need to access it many times.

Usage

```
statistics(result)

## S4 method for signature 'RegionResult'
statistics(result)
```

Arguments

`result` A [`'RegionResult-class'`].

Value

A [`'StatisticsResult-class'`].

Examples

```
ds <- methods::new("VennDataset",
  set_names = c("A", "B"),
  items = list(A = c("x", "y"), B = c("y", "z")),
  item_order = c("x", "y", "z"),
  universe_size = 10L, source_path = NULL, format = "csv")
result <- analyze(ds)
stats <- statistics(result)
stats@jaccard["A", "B"]

result <- analyze(load_sample("dataset_real_cancer_drivers_4"))
stats <- statistics(result)
stats@jaccard
stats@hypergeometric
```

StatisticsResult-class

StatisticsResult: container for pairwise statistical metric tables

Description

Returned by [compute_pairwise()] and (lazily) by ‘statistics()’ on a ‘RegionResult’. Holds five tables:

Slots

jaccard NxN named matrix of Jaccard indices.

dice NxN named matrix of Sorensen-Dice coefficients.

overlap_coefficient NxN named matrix of Szymkiewicz-Simpson overlap coefficients.

fold_enrichment NxN named matrix of fold-enrichment values.

hypergeometric Long-form data.frame (one row per set pair) with columns: set_a, set_b, intersection, expected, p_value, p_adjusted, p_bonferroni, p_two_sided, jaccard_ci_low, jaccard_ci_high, dice_ci_low, dice_ci_high, significant, highly_significant.

SvgImage-class

SvgImage: wrapper for rendered SVG output

Description

Returned by [render_share_distribution()] and [render_cluster_heatmap()]. Mirrors the Python ‘SvgImage’ dataclass so cross-package parity tests can use a uniform attribute name (‘content’). Note that [render_venn_svg()] still returns a plain ‘character’ for backward compatibility with the v2.0.x public API; the new plot renderers return ‘SvgImage’ so callers can introspect explicit ‘width’ / ‘height’ extents.

Slots

content The SVG document as a string.

width Pixel width of the rendered SVG.

height Pixel height of the rendered SVG.

tidy.RegionResult	<i>Tidy method for RegionResult (broom-compatible)</i>
-------------------	--

Description

Returns a long-form table with one row per ordered set pair, combining the five pairwise statistical metrics (Jaccard, Dice, overlap coefficient, fold enrichment, hypergeometric p-value + BH-FDR-adjusted q-value). Pair ordering is 'set_a, set_b' with 'set_a' appearing earlier in 'result@dataset@set_names'.

Usage

```
## S3 method for class 'RegionResult'
tidy(x, ...)
```

Arguments

x	A ['RegionResult-class'].
...	Unused (broom convention).

Value

A tibble (or data.frame if 'tibble' is not installed) with columns 'set_a', 'set_b', 'intersection', 'expected', 'jaccard', 'dice', 'overlap_coefficient', 'fold_enrichment', 'p_value', 'p_adjusted', 'significant', 'highly_significant'. One row per unordered pair, so 'n*(n-1)/2' rows for an 'n'-set dataset.

Examples

```
ds <- methods::new("VennDataset",
  set_names = c("A", "B"),
  items = list(A = c("x", "y"), B = c("y", "z")),
  item_order = c("x", "y", "z"),
  universe_size = 10L, source_path = NULL, format = "csv")
result <- analyze(ds)
if (requireNamespace("broom", quietly = TRUE)) broom::tidy(result)

result <- analyze(load_sample("dataset_real_cancer_drivers_4"))
broom::tidy(result)
```

to_excel_workbook	<i>Write a 3-sheet Excel workbook matching the webtool's ZIP bundle</i>
-------------------	---

Description

Sheets:

- Jaccard – NxN matrix of Jaccard indices.
- Sorensen-Dice (the actual sheet title uses the o-with-stroke character) – NxN matrix of Dice coefficients.
- Enrichment – long-form (set_a, set_b, intersection, union, expected, fold_enrichment, p_value, fdr, significant).

Usage

```
to_excel_workbook(result, path)
```

Arguments

result	A ['RegionResult-class'] from [analyze()].
path	Output xlsx file path.

Details

Mirrors the Python 'to_excel_workbook()' byte-for-byte in sheet titles, column order, and 4-decimal numeric formatting. Uses **openxlsx** (pure R, no Java dependency).

Value

Invisibly returns 'NULL'.

Examples

```
ds <- load_sample("dataset_real_cancer_drivers_4")
res <- analyze(ds)
to_excel_workbook(res, tempfile(fileext = ".xlsx"))
```

to_matrix_tsv

Write the Item Matrix TSV

Description

Mirrors the React webapp's "Export Matrix" button + Python's `RegionResult.to_matrix_tsv()` byte-for-byte.

Usage

```
to_matrix_tsv(result, path)

## S4 method for signature 'RegionResult'
to_matrix_tsv(result, path)
```

Arguments

result	A [<code>'RegionResult-class'</code>].
path	Destination file path.

Details

Columns: Item, <SetName1>, <SetName2>, ..., Region. Rows: one per item. Iteration order: mask $1..(2^n - 1)$; within each mask, items in `'dataset@item_order'`.

Value

Invisibly returns `'path'`.

Examples

```
ds <- methods::new("VennDataset",
  set_names = c("A", "B"),
  items = list(A = c("x", "y"), B = c("y", "z")),
  item_order = c("x", "y", "z"),
  universe_size = 10L, source_path = NULL, format = "csv")
result <- analyze(ds)
to_matrix_tsv(result, tempfile(fileext = ".tsv"))

result <- analyze(load_sample("dataset_real_cancer_drivers_4"))
to_matrix_tsv(result, tempfile(fileext = ".tsv"))
```

to_network_graphml	<i>Write the Cytoscape GraphML network export</i>
--------------------	---

Description

Mirrors the React webapp's "GraphML" Cytoscape export button + Python's 'to_network_graphml()' byte-for-byte. Nodes are sets (id = letter, label = set name, size = inclusive cardinality); edges are ALL pairwise overlaps (weight = the 'intersection' metric by default) carrying jaccard, foldEnrichment, overlapCoeff, dice, pValue, fdr, and significant attributes. Node/edge order and numeric rendering are pinned – see 'packages/core/src/networkExport.ts' for the parity contract.

Usage

```
to_network_graphml(result, path)

## S4 method for signature 'RegionResult'
to_network_graphml(result, path)
```

Arguments

result	A ['RegionResult-class'].
path	Destination file path.

Value

Invisibly returns 'path'.

Examples

```
ds <- methods::new("VennDataset",
  set_names = c("A", "B"),
  items = list(A = c("x", "y"), B = c("y", "z")),
  item_order = c("x", "y", "z"),
  universe_size = 10L, source_path = NULL, format = "csv")
result <- analyze(ds)
to_network_graphml(result, tempfile(fileext = ".graphml"))

result <- analyze(load_sample("dataset_real_cancer_drivers_4"))
to_network_graphml(result, tempfile(fileext = ".graphml"))
```

to_network_sif	<i>Write the Cytoscape SIF network export</i>
----------------	---

Description

Mirrors the React webapp's "SIF" Cytoscape export button + Python's 'to_network_sif()' byte-for-byte. One line per edge (in edge order), tab-separated: source letter, the literal interaction type 'overlap', then target letter. Isolated nodes (degree 0) are emitted as lone single-token lines after all edges, in node order.

Usage

```
to_network_sif(result, path)

## S4 method for signature 'RegionResult'
to_network_sif(result, path)
```

Arguments

result	A ['RegionResult-class'].
path	Destination file path.

Value

Invisibly returns 'path'.

Examples

```
ds <- methods::new("VennDataset",
  set_names = c("A", "B"),
  items = list(A = c("x", "y"), B = c("y", "z")),
  item_order = c("x", "y", "z"),
  universe_size = 10L, source_path = NULL, format = "csv")
result <- analyze(ds)
to_network_sif(result, tempfile(fileext = ".sif"))

result <- analyze(load_sample("dataset_real_cancer_drivers_4"))
to_network_sif(result, tempfile(fileext = ".sif"))
```

to_one_vs_rest_tsv	<i>Write the one-vs-rest Enrichment TSV</i>
--------------------	---

Description

Mirrors the React webapp's "Enrichment: one-vs-rest (TSV)" export button + Python's 'RegionResult.to_one_vs_rest_tsv()' byte-for-byte.

Usage

```
to_one_vs_rest_tsv(result, path)

## S4 method for signature 'RegionResult'
to_one_vs_rest_tsv(result, path)
```

Arguments

result	A ['RegionResult-class'].
path	Destination file path.

Details

Each set *S* is tested against the union of all OTHER sets. 'Rest_Size' is derived from 'U' (the union of ALL sets = sum of exclusive counts over every non-empty region), NOT from [effective_universe()] ('N') – see [one_vs_rest_enrichment()] for the full derivation. Float formatting mirrors [to_statistics_tsv()] byte-for-byte:

Columns: Set, Name, Size, Rest_Size, Intersection, Expected, Fold_Enrichment, P_value, FDR, Bonferroni, Significant. * Expected: 2 decimals * Fold_Enrichment: 3 decimals * P_value / FDR / Bonferroni: scientific (JS toExponential(2)) if '< 0.001', else 6 decimals * Significant: one of "****", "***", "**", "ns" keyed off FDR thresholds (0.001, 0.01, 0.05).

Rows are sorted by P_value ascending (matches the underlying [one_vs_rest_enrichment()] table).

Value

Invisibly returns 'path'.

Examples

```
ds <- methods::new("VennDataset",
  set_names = c("A", "B"),
  items = list(A = c("x", "y"), B = c("y", "z")),
  item_order = c("x", "y", "z"),
  universe_size = 10L, source_path = NULL, format = "csv")
result <- analyze(ds)
to_one_vs_rest_tsv(result, tempfile(fileext = ".tsv"))

result <- analyze(load_sample("dataset_real_cancer_drivers_4"))
```

```
to_one_vs_rest_tsv(result, tempfile(fileext = ".tsv"))
```

to_pdf_report

Compose a multi-page PDF report from a RegionResult

Description

Writes a US Letter landscape PDF with overview, venn+upset, statistics tables, and (by default) network and methodology pages. Each page has a footer with package version, generation timestamp, and page number.

Usage

```
to_pdf_report(
  result,
  path,
  title = NULL,
  include_network = TRUE,
  include_about = TRUE,
  include_share = TRUE,
  include_cluster = FALSE
)
```

Arguments

result	A [<code>'RegionResult-class'</code>].
path	Output PDF file path.
title	Optional title override for the overview page.
include_network	If <code>'TRUE'</code> (default), include the network page.
include_about	If <code>'TRUE'</code> (default), include the methodology page.
include_share	If <code>'TRUE'</code> (default), include the Item Share Distribution page.
include_cluster	If <code>'TRUE'</code> , include the Cluster Heatmap page (default <code>'FALSE'</code> — opt-in like Python's <code>'cluster_heatmap=True'</code>).

Value

Invisibly returns `'NULL'`. The PDF is written to `'path'`.

Examples

```
if (getRversion() >= "4.6") {
  result <- analyze(load_sample("dataset_real_cancer_drivers_4"))
  to_pdf_report(result, tempfile(fileext = ".pdf"))
}
```

to_region_summary_tsv *Write the Region Summary TSV*

Description

Mirrors the React webapp's "Export Region Summary" button + Python's `RegionResult.to_region_summary_tsv()` byte-for-byte.

Usage

```
to_region_summary_tsv(result, path)

## S4 method for signature 'RegionResult'
to_region_summary_tsv(result, path)
```

Arguments

<code>result</code>	A [<code>'RegionResult-class'</code>].
<code>path</code>	Destination file path.

Details

Columns: Region, Sets, Depth, Exclusive_Count, Inclusive_Count, Exclusive_Pct, Items. Rows: every region ($1..2^n - 1$). Sorted by (Depth ASC, Region label ASC). Items: semicolon-joined, ordered by `'dataset@item_order'`. Cells starting with `'=/+/'-/'@'` (after optional leading whitespace) are escape-prefixed with a single quote.

Value

Invisibly returns `'path'`.

Examples

```
ds <- methods::new("VennDataset",
  set_names = c("A", "B"),
  items = list(A = c("x", "y"), B = c("y", "z")),
  item_order = c("x", "y", "z"),
  universe_size = 10L, source_path = NULL, format = "csv")
result <- analyze(ds)
to_region_summary_tsv(result, tempfile(fileext = ".tsv"))

result <- analyze(load_sample("dataset_real_cancer_drivers_4"))
to_region_summary_tsv(result, tempfile(fileext = ".tsv"))
```

to_result_json	Write the full Venn result + statistics as canonical JSON
----------------	---

Description

Mirrors the React webapp's "Full Result (JSON)" export ('exportResultJson', 'packages/core/src/jsonExport.ts') and Python's 'RegionResult.to_json()' byte-for-byte.

Usage

```
to_result_json(result, path)

## S4 method for signature 'RegionResult'
to_result_json(result, path)
```

Arguments

result	A ['RegionResult-class'].
path	Destination file path.

Details

Schema (key order PINNED):

```
{
  "schemaVersion": "1",
  "model": "<model id>",
  "setNames": { "A": "...", ... },
  "universeSize": <int>,
  "regions": [
    { "label", "sets": [...], "depth": <int>,
      "exclusiveCount": <int>, "inclusiveCount": <int>,
      "exclusiveItems": [...] }, ...
  ],
  "setSizes": { "A": <int>, ... },
  "statistics": [
    { "a", "b", "jaccard", "dice", "overlapCoeff",
      "intersection", "union", "expected", "foldEnrichment",
      "pValue", "fdr", "bonferroni", "pTwoSided",
      "significant": "***" | "**" | "*" | "ns" }, ...
  ]
}
```

'regions' covers all $2^n - 1$ non-empty subsets, sorted by depth ascending then label ascending (ASCII); 'exclusiveItems' preserves the dataset item order. 'statistics' is sorted by p-value ascending, with 'significant' rendered as the FDR star label. Every number is emitted through a shared number-rendering rule (fixed 6-decimal, trailing zeros stripped, never scientific) so the bytes match the webapp and Python exports. No trailing newline is written.

Value

Invisibly returns 'path'.

Examples

```
ds <- methods::new("VennDataset",
  set_names = c("A", "B"),
  items = list(A = c("x", "y"), B = c("y", "z")),
  item_order = c("x", "y", "z"),
  universe_size = 10L, source_path = NULL, format = "csv")
result <- analyze(ds)
to_result_json(result, tempfile(fileext = ".json"))

result <- analyze(load_sample("dataset_real_cancer_drivers_4"))
to_result_json(result, tempfile(fileext = ".json"))
```

to_statistics_tsv	<i>Write the pairwise Statistics TSV</i>
-------------------	--

Description

Mirrors the React webapp’s DataSummaryPanel "Export Statistics" button + Python’s ‘RegionResult.to_statistics_tsv()’ byte-for-byte.

Usage

```
to_statistics_tsv(result, path)

## S4 method for signature 'RegionResult'
to_statistics_tsv(result, path)
```

Arguments

result	A [‘RegionResult-class’].
path	Destination file path.

Details

Columns: Set_A, Set_B, Name_A, Name_B, Size_A, Size_B, Intersection, Union, Jaccard, Overlap_Coeff, Dice, Expected, Fold_Enrichment, P_value, FDR, Bonferroni, P_two_sided, Jaccard_CI_low, Jaccard_CI_high, Dice_CI_low, Dice_CI_high, FE_CI_low, FE_CI_high, Significant. Float formatting: * Jaccard / Overlap_Coeff / Dice: 4 decimals via [.js_to_fixed()] * Expected: 2 decimals * Fold_Enrichment: 3 decimals * P_value / FDR / Bonferroni / P_two_sided: scientific (JS toExponential(2)) if ‘< 0.001’, else 6 decimals * Jaccard_CI_low/high, Dice_CI_low/high, FE_CI_low/high: 4 decimals * Significant: one of ‘****’, ‘***’, ‘**’, ‘*’, ‘ns’ keyed off FDR thresholds (0.001, 0.01, 0.05).

Rows are sorted by P_value ascending (matches the underlying StatisticsResult).

Value

Invisibly returns 'path'.

Examples

```
ds <- methods::new("VennDataset",
  set_names = c("A", "B"),
  items = list(A = c("x", "y"), B = c("y", "z")),
  item_order = c("x", "y", "z"),
  universe_size = 10L, source_path = NULL, format = "csv")
result <- analyze(ds)
to_statistics_tsv(result, tempfile(fileext = ".tsv"))

result <- analyze(load_sample("dataset_real_cancer_drivers_4"))
to_statistics_tsv(result, tempfile(fileext = ".tsv"))
```

to_zip_report

Write a Full Report ZIP bundle

Description

Bundles the multi-page PDF report alongside the supporting SVGs (Venn, UpSet, Network, Share Distribution), TSVs (Region Summary, Item Matrix, Statistics), the 3-sheet xlsx workbook, and a README.txt provenance header into a single ZIP. Mirrors the webtool's *Download Everything* button and Python's `vd1 report zip`.

Usage

```
to_zip_report(result, path, include_share = TRUE, include_cluster = FALSE)
```

Arguments

result	A ['RegionResult-class'] from [analyze()].
path	Output '.zip' file path.
include_share	Passed through to [to_pdf_report()] (default 'TRUE').
include_cluster	Passed through to [to_pdf_report()] (default 'FALSE').

Value

Invisibly returns 'NULL'. The ZIP is written to 'path'.

Examples

```
if (getRversion() >= "4.6") {
  ds <- load_sample("dataset_real_cancer_drivers_4")
  res <- analyze(ds)
  to_zip_report(res, tempfile(fileext = ".zip"))
}
```

union_items	<i>Items in the union of the named sets</i>
-------------	---

Description

Returns items that appear in ANY of the sets listed in ‘sets’. Equivalent to ‘unique(c(items_in_A, items_in_B, ...))’.

Usage

```
union_items(result, sets)
```

Arguments

result	A [‘RegionResult-class’] from [analyze()].
sets	Character vector of set letters (“A”, “B”, ...) or display names (values from ‘result@dataset@set_names’). May mix both.

Value

A character vector of item IDs (deduplicated, first-seen order).

Examples

```
ds <- methods::new("VennDataset",
  set_names = c("A", "B", "C"),
  items = list(A = c("x", "y"), B = c("y", "z"), C = c("z", "w")),
  item_order = c("x", "y", "z", "w"),
  universe_size = 10L, source_path = NULL, format = "csv")
res <- analyze(ds)
union_items(res, c("A", "B"))  # "x", "y", "z"
```

validate_dataset	<i>Load a CSV/TSV file and report data-quality issues, with an opt-in warning</i>
------------------	---

Description

Convenience wrapper around [`analyze_data_quality()`] that reads and parses a file the same way [`load_csv()`] does (auto-detecting the delimiter unless overridden), **without** building a `VennDataset`. This is the opt-in surface for data-quality feedback: [`load_csv()`] / [`load_tsv()`] themselves never emit warnings and are unchanged by this function's existence.

Usage

```
validate_dataset(
  path,
  mode = c("binary", "aggregated"),
  delimiter = NULL,
  prefix_cols = 1L,
  warn = TRUE
)
```

Arguments

path	Path to the file.
mode	"binary" (default) or "aggregated".
delimiter	Explicit delimiter override. <code>NULL</code> auto-detects, same as [<code>load_csv()</code>].
prefix_cols	Number of leading metadata columns in binary mode (default 1). Only column 1 is ever read as the item identifier, matching <code>.binary_columns_to_dataset()</code> . Ignored when <code>mode = "aggregated"</code> .
warn	If <code>TRUE</code> (default) and the report has any findings, emits a single <code>warning()</code> summarizing the counts. Set <code>FALSE</code> to only get the returned report silently.

Value

The `analyze_data_quality()` report list, returned in addition to (not instead of) any `warning()` raised.

Examples

```
tmp <- tempfile(fileext = ".csv")
writeLines(c("Gene,SetA,SetB", "G1,1,0", "G1,1,1", "G2,0,1"), tmp)
report <- validate_dataset(tmp, mode = "binary")
report$has_warnings
```

vdl_version	<i>Get the vennDiagramLab package version</i>
-------------	---

Description

Returns the installed version of vennDiagramLab as a character string. This trivial function exists so the Phase 0 skeleton has one public export; Phase 1 introduces the real `analyze()` / `load_*` API.

Usage

```
vdl_version()
```

Value

Character string, the package version (e.g. "2.0.0").

Examples

```
vdl_version()
```

VennDataset-class	<i>VennDataset: in-memory representation of a Venn-diagram input</i>
-------------------	--

Description

Returned by the `'load_*()'` family and consumed by `[analyze()]`. Holds the deduplicated set members, first-seen item ordering for byte-equivalent TSV output, and source metadata (path, format, optional hypergeometric universe size).

Slots

`set_names` Ordered character vector of set identifiers (length 2-9).

`items` Named list (`'names(items)' == set_names`) of character vectors, each containing the deduplicated members of the corresponding set.

`item_order` First-seen insertion order of all items across all sets, matching JS Set/Map semantics. Used by TSV writers (Phase 2) for byte-equivalent output to the web tool.

`universe_size` Hypergeometric universe N (population size) from the source file, when known. Binary CSV/TSV loaders set this to the row count (matching the web tool's `'csv.rows.length'`); other formats leave it `'NULL'`, signaling "compute as `length(item_order)`" downstream.

`source_path` Original file path if loaded from disk; `'NULL'` for in-memory datasets.

`format` Source format: one of `"csv"`, `"tsv"`, `"gmt"`, `"gmx"`.

Index

analyze, [3](#)
analyze_data_quality, [4](#)
augment.RegionResult, [6](#)

bh_fdr, [7](#)

circle_intersection_area, [7](#)
cluster_set_order, [8](#)
compute_pairwise, [9](#)

dice, [10](#)

effective_universe, [10](#)
effective_universe,RegionResult-method
 (effective_universe), [10](#)
exclusive_items, [11](#)

fold_enrichment, [12](#)
fold_enrichment_ci, [12](#)

generate_proportional_svg, [13](#)
geom_venn, [14](#)
glance.RegionResult, [15](#)

hypergeometric_p_value, [16](#)

intersection_items, [17](#)
item_share_distribution, [18](#)

jaccard, [18](#)

list_models, [19](#)
list_samples, [20](#)
load_csv, [20](#)
load_gmt, [21](#)
load_gmx, [21](#)
load_sample, [22](#)
load_tsv, [23](#)

one_vs_rest_enrichment, [23](#)
overlap_coefficient, [25](#)

parse_region_expression, [25](#)

RegionData-class, [26](#)
RegionResult-class, [27](#)
render_cluster_heatmap, [27](#)
render_enrichment_bar, [28](#)
render_enrichment_lollipop, [29](#)
render_network, [30](#)
render_share_distribution, [32](#)
render_upset, [32](#)
render_venn_svg, [34](#)

solve_2set, [35](#)
solve_3set, [36](#)
statistics, [37](#)
statistics,RegionResult-method
 (statistics), [37](#)
StatisticsResult-class, [38](#)
SvgImage-class, [38](#)

tidy.RegionResult, [39](#)
to_excel_workbook, [40](#)
to_matrix_tsv, [41](#)
to_matrix_tsv,RegionResult-method
 (to_matrix_tsv), [41](#)
to_network_graphml, [42](#)
to_network_graphml,RegionResult-method
 (to_network_graphml), [42](#)
to_network_sif, [43](#)
to_network_sif,RegionResult-method
 (to_network_sif), [43](#)
to_one_vs_rest_tsv, [44](#)
to_one_vs_rest_tsv,RegionResult-method
 (to_one_vs_rest_tsv), [44](#)
to_pdf_report, [45](#)
to_region_summary_tsv, [46](#)
to_region_summary_tsv,RegionResult-method
 (to_region_summary_tsv), [46](#)
to_result_json, [47](#)

`to_result_json`, `RegionResult`-method
 (`to_result_json`), [47](#)
`to_statistics_tsv`, [48](#)
`to_statistics_tsv`, `RegionResult`-method
 (`to_statistics_tsv`), [48](#)
`to_zip_report`, [49](#)

`union_items`, [50](#)

`validate_dataset`, [51](#)
`vdl_version`, [52](#)
`VennDataset`-class, [52](#)