Package 'plotomics'

September 4, 2026

Type Package

Title High-Performance Bioinformatics Visualizations

Version 0.1.0

Description Lightweight, GPU-accelerated bioinformatics visualization widgets (volcano plots, expression and clustered heatmaps, dot plots, stacked violins, embeddings, spatial tissue maps, oncoprints, protein domain lollipops, Kaplan-Meier curves, mutational signature profiles, UpSet plots, treemaps, networks and Hi-C contact matrices) backed by a shared JavaScript core and exposed to R through 'htmlwidgets'. Designed for large datasets that render smoothly in the browser, the 'RStudio' Viewer, R Markdown, Quarto and Shiny.

License MIT + file LICENSE

Encoding UTF-8

Imports htmlwidgets

Suggests knitr, rmarkdown, shiny, testthat (>= 3.0.0), survival

VignetteBuilder knitr

Config/testthat/edition 3

URL https://github.com/samuelbharti/plotomics, https://doi.org/10.5281/zenodo.21926306

BugReports https://github.com/samuelbharti/plotomics/issues

RoxygenNote 8.0.0

NeedsCompilation no

Author Samuel Bharti [aut, cre] (ORCID: <https://orcid.org/0000-0003-4190-7058>)

Maintainer Samuel Bharti <samuelbharti.io@gmail.com>

Repository CRAN

Date/Publication 2026-09-04 21:50:02 UTC

Contents

`bioheatmap` *Expression heatmap*

Description

A GPU-accelerated heatmap for large expression matrices (samples x genes). The matrix is uploaded to the GPU as a single texture and colormapped in a fragment shader (via `regl`), so matrices with a million or more cells pan and zoom smoothly. The colorbar legend and row/column tick labels are drawn as crisp vector overlays; ticks appear only when few enough to be legible.

Usage

```
bioheatmap(
  mat,
  colormap = c("viridis", "rdbu"),
  z_score = FALSE,
  vmin = NULL,
  vmax = NULL,
  show_colorbar = TRUE,
  theme = NULL,
  width = NULL,
  height = NULL,
  element_id = NULL
)

heatmap_plotomics(
  mat,
  colormap = c("viridis", "rdbu"),
  z_score = FALSE,
  vmin = NULL,
  vmax = NULL,
  show_colorbar = TRUE,
  theme = NULL,
  width = NULL,
  height = NULL,
  element_id = NULL
)
```

Arguments

`mat`	A numeric matrix (rows x columns). `rownames(mat)` and `colnames(mat)`, when present, are used as row/column tick labels.
`colormap`	Color ramp: `"viridis"` (sequential) or `"rdbu"` (diverging).
`z_score`	Logical; if `TRUE`, each row is z-score normalized before coloring (a row-centered heatmap).
`vmin`, `vmax`	Lower/upper clamp of the color domain. `NULL` (the default) auto-scales from the data; for `"rdbu"` the auto domain is symmetric about zero.
`show_colorbar`	Logical; draw the colorbar legend.
`theme`	Optional named list of theme overrides (colors, fonts, ...) merged over the component defaults in the browser. `NULL` uses the default theme.
`width`, `height`	Widget dimensions (any valid CSS size).
`element_id`	Optional explicit DOM id.

Details

The function is exported as `bioheatmap()` (and aliased as `heatmap_plotomics()`) to avoid masking `stats::heatmap()`.

Value

An `htmlwidget` object.

Examples

```
set.seed(1)
m <- matrix(rnorm(50 * 30), nrow = 50, ncol = 30)
rownames(m) <- paste0("gene", seq_len(50))
colnames(m) <- paste0("sample", seq_len(30))
bioheatmap(m, z_score = TRUE)
```

`bioheatmap-shiny` *Shiny bindings for bioheatmap*

Description

Output and render functions for using `bioheatmap()` within Shiny applications and interactive R Markdown / Quarto documents.

Usage

```
bioheatmapOutput(output_id, width = "100%", height = "480px")

renderBioheatmap(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

`output_id`	Output variable to read from.
`width, height`	Element size, passed to `htmlwidgets::shinyWidgetOutput()`.
`expr`	An expression that generates a `bioheatmap()` widget.
`env`	The environment in which to evaluate `expr`.
`quoted`	Is `expr` already quoted? Defaults to `FALSE`.

Value

`bioheatmapOutput()` returns a Shiny output UI element; `renderBioheatmap()` returns a Shiny render function.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  ui <- fluidPage(bioheatmapOutput("hm"))
  server <- function(input, output) {
    output$hm <- renderBioheatmap({
      bioheatmap(matrix(rnorm(200), 20, 10))
    })
  }
  shinyApp(ui, server)
}
```

`bioprofile` *Grouped categorical bar profile*

Description

An ordered bar profile whose categories collapse into coloured header blocks. Built for the 96-context mutational signature plot, where the bars are the trinucleotide contexts and the six blocks are the substitution classes, a layout conventional enough that readers parse it without a legend. It generalises to any ordered categorical profile that groups into runs, hence the generic name.

Usage

```
bioprofile(
  data,
  groups = NULL,
  group_colors = NULL,
  title = NULL,
  bar_width = 0.62,
  as_fraction = FALSE,
  show_header = TRUE,
  show_bar_labels = TRUE,
  y_label = "mutations",
  theme = NULL,
  width = NULL,
  height = NULL,
  element_id = NULL
)

profile_plotomics(
  data,
  groups = NULL,
  group_colors = NULL,
  title = NULL,
  bar_width = 0.62,
  as_fraction = FALSE,
  show_header = TRUE,
  show_bar_labels = TRUE,
  y_label = "mutations",
  theme = NULL,
  width = NULL,
  height = NULL,
  element_id = NULL
)
```

Arguments

`data`	A data frame with a numeric `value` column. Optional `group` (category per bar, whose contiguous runs become the header blocks) and `label` (per-bar tick label)

	columns.
groups	Character vector fixing the group order and colour assignment. Defaults to order of appearance.
group_colors	One hex colour per group. NULL uses the component's categorical palette.
title	Optional title drawn above the header band.
bar_width	Fraction of each slot the bar occupies, in (0, 1].
as_fraction	Show values as a share of the total rather than raw counts.
show_header, show_bar_labels	Toggle the header band and tick labels.
y_label	Axis label.
theme	Optional named list of theme overrides.
width, height	Widget dimensions (any valid CSS size).
element_id	Optional explicit DOM id.

Details

Bars are canvas-drawn, so a few thousand bins (a binned copy-number profile, a coverage track) work as well as 96 contexts.

Bars are drawn in the order given. For SBS96 that order is part of the convention, so the component does not sort.

Named bioprofile() rather than profile() so that attaching the package does not mask the stats::profile() generic, which profiles a fitted model's likelihood. This follows bioheatmap(), which keeps clear of stats::heatmap() the same way. profile_plotomics() is an alias, for symmetry with heatmap_plotomics().

Value

An htmlwidget object.

Examples

```
df <- data.frame(
  value = c(3, 5, 2, 8),
  group = c("C>A", "C>A", "C>T", "C>T"),
  label = c("ACA", "ACC", "TCA", "TCT")
)
bioprofile(df)
```

`bioprofile-shiny` *Shiny bindings for bioprofile*

Description

Output and render functions for using `bioprofile()` within Shiny applications and interactive R Markdown / Quarto documents.

Usage

```
bioprofileOutput(output_id, width = "100%", height = "380px")

renderBioprofile(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

`output_id`	Output variable to read from.
`width, height`	Element size, passed to `htmlwidgets::shinyWidgetOutput()`.
`expr`	An expression that generates a `bioprofile()` widget.
`env`	The environment in which to evaluate `expr`.
`quoted`	Is `expr` already quoted? Defaults to `FALSE`.

Value

`bioprofileOutput()` returns a Shiny output UI element; `renderBioprofile()` returns a Shiny render function.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  ui <- fluidPage(bioprofileOutput("bp"))
  server <- function(input, output) {
    output$bp <- renderBioprofile({
      bioprofile(context = c("C>A", "C>G", "C>T"),
                 value = c(0.3, 0.5, 0.2))
    })
  }
  shinyApp(ui, server)
}
```

clustermap *Clustered heatmap with dendrograms*

Description

A hierarchically-clustered expression heatmap (in the spirit of `seaborn.clustermap` / Morpheus): the matrix is drawn on a GPU/canvas data layer so large matrices stay smooth, while dendrograms, tick labels and the colorbar are crisp vector overlays. Rows and columns are agglomeratively clustered and reordered so structure appears as blocks along the diagonal.

Usage

```
clustermap(
  mat,
  metric = c("euclidean", "correlation"),
  linkage = c("average", "complete", "ward"),
  colormap = c("viridis", "rdbu"),
  z_score = FALSE,
  cluster_rows = TRUE,
  cluster_cols = TRUE,
  show_row_dendrogram = TRUE,
  show_col_dendrogram = TRUE,
  show_labels = TRUE,
  legend_title = "value",
  row_linkage = NULL,
  col_linkage = NULL,
  theme = NULL,
  width = NULL,
  height = NULL,
  element_id = NULL
)
```

Arguments

`mat`	A numeric matrix. Row and column names, if present, are used as labels. Values are transported row-major to the browser.
`metric`	Distance metric for clustering: `"euclidean"` or `"correlation"` (1 - Pearson correlation).
`linkage`	Agglomeration method: `"average"`, `"complete"` or `"ward"`.
`colormap`	Color ramp: `"viridis"` (sequential) or `"rdbu"` (diverging).
`z_score`	Standardize each row to mean 0 / sd 1 before coloring.
`cluster_rows, cluster_cols`	Cluster and reorder rows / columns. Ignored for an axis when a precomputed `row_linkage` / `col_linkage` is supplied.
`show_row_dendrogram, show_col_dendrogram`	Draw the row / column dendrogram.

show_labels	Draw row/column tick labels (auto-hidden when cells get too small to be legible).
legend_title	Colorbar legend title.
row_linkage, col_linkage	Optional precomputed leaf order or dendrogram to skip clustering that axis. Either an integer vector giving the 0-based leaf order, or a list with order (0-based) and merges (each a list with left, right, height; leaves are 0..n-1, internal node k is n + k).
theme	Optional named list of theme overrides (colors, fonts, ...) merged over the component defaults in the browser. NULL uses the default theme.
width, height	Widget dimensions (any valid CSS size).
element_id	Optional explicit DOM id.

Details

Clustering is at least O(n^2) in the number of rows/columns (it builds a full distance matrix), so clustermap() only clusters automatically when a dimension has at most 2000 leaves. For larger matrices, precompute a leaf order (or a dendrogram) elsewhere and pass it via row_linkage / col_linkage to skip clustering; the heatmap rendering itself scales to much larger matrices.

Value

An htmlwidget object.

Examples

```
set.seed(1)
# Two clear blocks of correlated genes across two groups of samples.
mat <- rbind(
  matrix(rnorm(20 * 10, mean = 2), nrow = 20),
  matrix(rnorm(20 * 10, mean = -2), nrow = 20)
)
rownames(mat) <- paste0("gene", seq_len(nrow(mat)))
colnames(mat) <- paste0("s", seq_len(ncol(mat)))
clustermap(mat, colormap = "rdbu", z_score = TRUE)
```

---

clustermap-shiny *Shiny bindings for clustermap*

Description

Output and render functions for using clustermap() within Shiny applications and interactive R Markdown / Quarto documents.

Usage

```
clustermapOutput(output_id, width = "100%", height = "480px")

renderClustermap(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

output_id	Output variable to read from.
width, height	Element size, passed to htmlwidgets::shinyWidgetOutput().
expr	An expression that generates a clustermap() widget.
env	The environment in which to evaluate expr.
quoted	Is expr already quoted? Defaults to FALSE.

Value

clustermapOutput() returns a Shiny output UI element; renderClustermap() returns a Shiny render function.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  ui <- fluidPage(clustermapOutput("cm"))
  server <- function(input, output) {
    output$cm <- renderClustermap({
      clustermap(matrix(rnorm(200), 20, 10))
    })
  }
  shinyApp(ui, server)
}
```

---

dotplot *Marker gene dot plot*

Description

Features down the rows, groups across the columns, each cell a dot whose size is the fraction of the group expressing the gene and whose colour is the expression level. Two channels because colour alone cannot separate "high in a few cells" from "moderate in all of them", and that distinction is usually what decides whether a gene is a marker.

Usage

```
dotplot(
  data,
  genes = NULL,
  clusters = NULL,
  value_label = "mean expression",
```

```
  size_label = "% expressing",
  colormap = c("viridis", "rdbu", "ltc", "ltcdiv"),
  max_radius = 9,
  value_domain = NULL,
  show_grid = TRUE,
  show_legend = TRUE,
  theme = NULL,
  width = NULL,
  height = NULL,
  element_id = NULL
)
```

Arguments

`data`	A data frame in long form, one row per dot, with `gene` and `cluster` key columns, a `pct` column (percent expressing, 0-100) driving dot size, and a `value` column (expression level) driving dot colour.
`genes`	Character vector fixing the row order. A factor `gene` column supplies it from its levels. Defaults to order of appearance.
`clusters`	Character vector fixing the column order, likewise.
`value_label`, `size_label`	Legend titles.
`colormap`	Sequential ramp for the colour channel: `"viridis"`, `"rdbu"`, `"ltc"` (an earthy teal to sand to rust sequential ramp) or `"ltcdiv"` (its diverging counterpart, neutral cream at the midpoint).
`max_radius`	Radius in pixels of a dot at 100 percent.
`value_domain`	Length-2 numeric fixing the colour scale. `NULL` uses the data range. Set it when comparing two dot plots side by side.
`show_grid`, `show_legend`	Toggle the gridlines and the legends.
`theme`	Optional named list of theme overrides.
`width`, `height`	Widget dimensions (any valid CSS size).
`element_id`	Optional explicit DOM id.

Details

Dot area, not radius, is proportional to the percentage. Scaling radius linearly would quadruple the ink for a doubled percentage, which is the classic way a dot plot overstates its strongest cells.

Rows and columns are drawn in the order given. Sorting genes by the group they best mark is an analysis decision, so the component does not do it.

Value

An `htmlwidget` object.

Examples

```
df <- expand.grid(gene = c("CD3D", "MS4A1"), cluster = c("T", "B"),
                  stringsAsFactors = FALSE)
df$pct <- c(88, 4, 6, 91)
df$value <- c(2.4, 0.1, 0.2, 2.7)
dotplot(df)
```

`dotplot-shiny` *Shiny bindings for dotplot*

Description

Output and render functions for using `dotplot()` within Shiny applications and interactive R Markdown / Quarto documents.

Usage

```
dotplotOutput(output_id, width = "100%", height = "560px")

renderDotplot(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

`output_id`	Output variable to read from.
`width, height`	Element size, passed to `htmlwidgets::shinyWidgetOutput()`.
`expr`	An expression that generates a `dotplot()` widget.
`env`	The environment in which to evaluate `expr`.
`quoted`	Is `expr` already quoted? Defaults to `FALSE`.

Value

`dotplotOutput()` returns a Shiny output UI element; `renderDotplot()` returns a Shiny render function.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  df <- expand.grid(gene = c("CD3D", "MS4A1"), cluster = c("T", "B"),
                    stringsAsFactors = FALSE)
  df$pct <- c(88, 4, 6, 91)
  df$value <- c(2.4, 0.1, 0.2, 2.7)
  ui <- fluidPage(dotplotOutput("dp"))
  server <- function(input, output) {
    output$dp <- renderDotplot(dotplot(df))
  }
  shinyApp(ui, server)
}
```

`embedding` *Embedding scatter (UMAP / t-SNE / PCA)*

Description

A GPU-accelerated 2-D scatter viewer for dimensionality-reduction output. Points are rendered with WebGL (via regl-scatterplot) so hundreds of thousands to millions of cells stay interactive at 60fps, while the legend and an optional axis frame are drawn as crisp vector overlays. Lasso selection is enabled (drag from empty space to select points).

Usage

```
embedding(
  data,
  point_size = 3,
  point_scale_mode = c("asinh", "linear", "constant"),
  opacity = 0.8,
  color_mode = c("auto", "categorical", "continuous"),
  colormap = c("viridis", "rdbu"),
  mouse_mode = c("panZoom", "lasso"),
  aspect = c("fill", "equal"),
  padding = 0.04,
  x_label = "UMAP 1",
  y_label = "UMAP 2",
  show_axes = FALSE,
  show_legend = TRUE,
  theme = NULL,
  width = NULL,
  height = NULL,
  element_id = NULL
)
```

Arguments

`data`	A data frame with numeric columns `x` and `y` (the embedding coordinates). An optional `color` column drives coloring: a character or factor column is treated as categorical (discrete legend), a numeric column as continuous (sequential colormap + colorbar). A **factor** `color` fixes the legend order and the color assignment to its levels, and keeps unused levels in the legend, the way `drop = FALSE` does in ggplot2. An optional `label` column supplies per-point tooltip text.
`point_size`	Point radius in pixels. Under the default `point_scale_mode` the renderer scales this by the camera and clamps it to one pixel on a widely-scaled plot, so set `point_scale_mode = "constant"` if you want it honoured literally.
`point_scale_mode`	How `point_size` responds to zoom. `"asinh"` and `"linear"` shrink points as you zoom out, which keeps a dense embedding readable, but both floor at one

	pixel once the camera scale drops below `1 / point_size`. `"constant"` sizes points in literal pixels.
`opacity`	Point opacity in `[0, 1]`.
`color_mode`	How to interpret the `color` column: `"auto"` detects from its type, or force `"categorical"` / `"continuous"`.
`colormap`	Sequential color ramp for continuous coloring: `"viridis"` or `"rdbu"`.
`mouse_mode`	Primary drag gesture: `"panZoom"` (default) pans/zooms and `"lasso"` makes a plain drag draw a selection.
`aspect`	How the fitted view maps data units onto pixels. `"fill"` stretches each axis to fill the canvas, which suits a UMAP, whose axes carry no units. `"equal"` gives both axes the same units per pixel; use it when the axes share units and their relative spread is part of the claim, as in PCA scores.
`padding`	Fraction of the data range to pad around the fitted view. Larger values zoom out, leaving more empty space at the edges, which stops the outermost points being clipped by the canvas border.
`x_label`, `y_label`	Axis titles (shown when `show_axes = TRUE`).
`show_axes`	Draw the axis frame + ticks (embeddings usually hide axes).
`show_legend`	Draw the legend (discrete swatches or a colorbar).
`theme`	Optional named list of theme overrides (colors, fonts, ...) merged over the component defaults in the browser. `NULL` uses the default theme.
`width`, `height`	Widget dimensions (any valid CSS size).
`element_id`	Optional explicit DOM id.

Value

An `htmlwidget` object.

Examples

```
set.seed(1)
n <- 5000
k <- sample(0:5, n, replace = TRUE)
df <- data.frame(
  x = rnorm(n) + k * 4,
  y = rnorm(n) + (k %% 2) * 4,
  color = paste0("cluster ", k + 1),
  label = paste0("cell", seq_len(n))
)
embedding(df)
```

`embedding-shiny` *Shiny bindings for embedding*

Description

Output and render functions for using `embedding()` within Shiny applications and interactive R Markdown / Quarto documents.

Usage

```
embeddingOutput(output_id, width = "100%", height = "480px")

renderEmbedding(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

`output_id`	Output variable to read from.
`width, height`	Element size, passed to `htmlwidgets::shinyWidgetOutput()`.
`expr`	An expression that generates an `embedding()` widget.
`env`	The environment in which to evaluate `expr`.
`quoted`	Is `expr` already quoted? Defaults to `FALSE`.

Details

In a Shiny app the lasso selection is pushed back to the server as `input$<output_id>_selected`, a 0-based integer vector of the selected rows (so `embeddingOutput("umap")` populates `input$umap_selected`). It updates on every completed selection and is `NULL` until the first one.

Value

`embeddingOutput()` returns a Shiny output UI element; `renderEmbedding()` returns a Shiny render function.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  ui <- fluidPage(embeddingOutput("emb"))
  server <- function(input, output) {
    output$emb <- renderEmbedding({
      embedding(data.frame(x = rnorm(200), y = rnorm(200)))
    })
  }
  shinyApp(ui, server)
}
```

`hic` *Hi-C contact matrix*

Description

A GPU-accelerated Hi-C chromatin contact map. The matrix is uploaded once as a single-channel float texture and drawn as one WebGL quad (via `regl`), with the colormap and log/linear transform applied in the fragment shader, so pan/zoom stays smooth on very large matrices. A precomputed level-of-detail pyramid keeps interaction fast when zoomed out. Axes (genomic coordinate ticks) and the colorbar are drawn as crisp vector overlays. No tile server is required.

Usage

```
hic(
  mat,
  n = NULL,
  bin_size = NULL,
  chrom = NULL,
  colormap = c("viridis", "rdbu"),
  transform = c("log", "linear"),
  vmax = NULL,
  vmax_percentile = NULL,
  vmin = 0,
  symmetric = TRUE,
  label = NULL,
  theme = NULL,
  width = NULL,
  height = NULL,
  element_id = NULL
)
```

Arguments

`mat`	Either a square numeric matrix of contact counts, or a data frame / list giving a sparse COO triplet with integer columns `i`, `j` and a numeric `v`. When a triplet is supplied, `n` must be given (or inferable from `max(i, j) + 1`).
`n`	Number of bins per axis. Required for the sparse (i/j/v) form; ignored for a dense matrix (taken from `nrow(mat)`).
`bin_size`	Genomic bin size in base pairs; used to label axes in bp/kb/Mb. `NULL` labels axes by bin index.
`chrom`	Optional chromosome name shown as the axis title.
`colormap`	Sequential colormap for intensity: `"viridis"` or `"rdbu"`.
`transform`	Intensity transform, `"log"` (default) or `"linear"`.
`vmax`	Upper clip of the intensity scale; `NULL` auto-picks a high percentile.

vmax_percentile
: Percentile in (0, 1] used to auto-pick vmax when vmax is NULL. NULL uses the component default.

vmin
: Lower clip of the intensity scale.

symmetric
: Mirror sparse i/j/v entries across the diagonal.

label
: Axis title (overrides chrom when set).

theme
: Optional named list of theme overrides (colors, fonts, ...) merged over the component defaults in the browser. NULL uses the default theme.

width, height
: Widget dimensions (any valid CSS size).

element_id
: Optional explicit DOM id.

Value

An htmlwidget object.

Examples

```
set.seed(1)
n <- 200
# distance-decay background contact matrix
d <- abs(outer(seq_len(n), seq_len(n), `-`))
m <- 1000 / (d + 1)^1.2 + matrix(runif(n * n), n, n)
m <- (m + t(m)) / 2 # symmetrize
hic(m, bin_size = 10000, chrom = "chr1")
```

---

hic-shiny *Shiny bindings for hic*

Description

Output and render functions for using hic() within Shiny applications and interactive R Markdown / Quarto documents.

Usage

```
hicOutput(output_id, width = "100%", height = "480px")

renderHic(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

output_id
: Output variable to read from.

width, height
: Element size, passed to htmlwidgets::shinyWidgetOutput().

expr
: An expression that generates a hic() widget.

env
: The environment in which to evaluate expr.

quoted
: Is expr already quoted? Defaults to FALSE.

Value

`hicOutput()` returns a Shiny output UI element; `renderHic()` returns a Shiny render function.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  ui <- fluidPage(hicOutput("h"))
  server <- function(input, output) {
    output$h <- renderHic({
      hic(matrix(runif(100), 10, 10))
    })
  }
  shinyApp(ui, server)
}
```

---

km *Kaplan-Meier survival curves with a number-at-risk table*

Description

A right-continuous step curve per stratum, censoring ticks, an optional pointwise confidence band, and the number-at-risk table underneath. The table is on by default because a survival curve without one hides how much of its tail rests on a handful of patients, which is where readers over-read it.

Usage

```
km(
  data,
  groups = NULL,
  group_colors = NULL,
  risk_times = NULL,
  risk_counts = NULL,
  p_label = NULL,
  show_ci = TRUE,
  show_censors = TRUE,
  show_risk_table = TRUE,
  show_legend = TRUE,
  y_from_zero = TRUE,
  line_width = 2,
  x_label = "months",
  y_label = "overall survival",
  theme = NULL,
  width = NULL,
  height = NULL,
  element_id = NULL
)
```

Arguments

data	Either a survival::survfit object, or a data frame with numeric time and surv columns and optional lower, upper and group columns. Within a stratum, rows must be in ascending time order.
groups	Character vector fixing the stratum order and colour assignment. Defaults to order of appearance.
group_colors	One hex colour per stratum. NULL uses the component's categorical palette.
risk_times	Numeric vector of times for the at-risk table, also used as the x-axis ticks. NULL picks an even grid across the follow-up.
risk_counts	Integer matrix, strata x risk_times. Computed from a survfit object automatically; required alongside risk_times when you pass a data frame and want the table.
p_label	Optional annotation drawn inside the panel, e.g. "log-rank p = 0.02". Not computed here: pass what your test returned.
show_ci, show_censors, show_risk_table, show_legend	Toggle the confidence band, censoring ticks, at-risk table and legend.
y_from_zero	Start the y axis at zero. Opt-out rather than automatic: zooming y exaggerates separation between curves.
line_width	Curve stroke width in pixels.
x_label, y_label	Axis titles.
theme	Optional named list of theme overrides.
width, height	Widget dimensions (any valid CSS size).
element_id	Optional explicit DOM id.

Details

The widget draws, it does not estimate. Pass a survfit object and the estimates are read off it; pass a data frame and they are used as given. Both routes mean the numbers on screen are the ones your model produced, so a figure rendered here and the same figure rendered by plot() cannot disagree about where a curve steps.

Value

An htmlwidget object.

Examples

```
df <- data.frame(
  time = c(0, 5, 12, 0, 7, 15),
  surv = c(1, 0.9, 0.7, 1, 0.8, 0.5),
  group = rep(c("treated", "control"), each = 3)
)
km(df)
```

km-shiny *Shiny bindings for km*

Description

Output and render functions for using km() within Shiny applications and interactive R Markdown / Quarto documents.

Usage

```
kmOutput(output_id, width = "100%", height = "520px")

renderKm(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

output_id	Output variable to read from.
width, height	Element size, passed to htmlwidgets::shinyWidgetOutput().
expr	An expression that generates a km() widget.
env	The environment in which to evaluate expr.
quoted	Is expr already quoted? Defaults to FALSE.

Value

kmOutput() returns a Shiny output UI element; renderKm() returns a Shiny render function.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  ui <- fluidPage(kmOutput("surv"))
  server <- function(input, output) {
    output$surv <- renderKm({
      km(time = c(1, 2, 3, 4, 5), event = c(1, 0, 1, 0, 1))
    })
  }
  shinyApp(ui, server)
}
```

`lollipop` *Protein domain lollipop*

Description

Variants along a protein, drawn over its domain architecture: a backbone spanning the sequence with domain rectangles on it, mutation stems whose head area is proportional to recurrence, and an optional post-translational modification track below. Hotspots inside a functional domain read very differently from truncating variants scattered across one, which is what this figure exists to show.

Usage

```
lollipop(
  variants,
  length,
  gene = NULL,
  uniprot = NULL,
  domains = NULL,
  ptms = NULL,
  classes = NULL,
  class_colors = NULL,
  domain_colors = NULL,
  label_top_n = 12,
  show_ptms = TRUE,
  show_domains = TRUE,
  show_legend = TRUE,
  min_head_radius = 3,
  max_head_radius = 11,
  y_label = "samples",
  backbone_color = "#E6DCC8",
  stem_color = "#93a1b8",
  theme = NULL,
  width = NULL,
  height = NULL,
  element_id = NULL
)
```

Arguments

`variants`	A data frame with columns `position` (amino-acid position, 1-based) and `count` (recurrence). Optional `class` (variant class) and `label` (e.g. `"R175H"`) columns drive the colour and the text labels.
`length`	Protein length in residues.
`gene`, `uniprot`	Identifiers shown on the axis title.
`domains`	Optional data frame of domain rectangles with columns `name`, `start` and `end`.
`ptms`	Optional data frame of modification sites with columns `position` and `type`.

`classes` Character vector fixing the legend order and colour assignment. Defaults to the classes present, most frequent first.

`class_colors`, `domain_colors`
Character vectors of hex colours. `NULL` uses the component's categorical palette.

`label_top_n` Label the `n` most recurrent variants. Which stems get a label is resolved here and sent to the browser, so a redraw, an export and any static counterpart all label the same ones.

`show_ptms`, `show_domains`, `show_legend`
Toggle the surrounding tracks.

`min_head_radius`, `max_head_radius`
Stem head radius range in pixels. Head *area* is proportional to recurrence, so these bound the mapping rather than setting a size: raise `max_head_radius` when one hotspot dwarfs the rest and you want the difference to read at a glance.

`y_label` Axis title for the recurrence axis.

`backbone_color`, `stem_color`
Hex colours for the protein backbone rectangle and the mutation stems.

`theme` Optional named list of theme overrides.

`width`, `height` Widget dimensions (any valid CSS size).

`element_id` Optional explicit DOM id.

Details

Stems and domains are canvas-drawn so a protein with thousands of variants stays responsive; labels, axis and legend are a vector overlay.

Value

An `htmlwidget` object.

Examples

```
v <- data.frame(
  position = c(175, 248, 273),
  count = c(21, 15, 13),
  class = c("Missense", "Missense", "Missense"),
  label = c("R175H", "R248Q", "R273H")
)
d <- data.frame(name = "P53 DNA-binding", start = 100, end = 288)
lollipop(v, length = 393, gene = "TP53", uniprot = "P04637", domains = d)
```

`lollipop-shiny` *Shiny bindings for lollipop*

Description

Output and render functions for using `lollipop()` within Shiny applications and interactive R Markdown / Quarto documents.

Usage

```
lollipopOutput(output_id, width = "100%", height = "440px")

renderLollipop(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

`output_id`	Output variable to read from.
`width, height`	Element size, passed to `htmlwidgets::shinyWidgetOutput()`.
`expr`	An expression that generates a `lollipop()` widget.
`env`	The environment in which to evaluate `expr`.
`quoted`	Is `expr` already quoted? Defaults to `FALSE`.

Value

`lollipopOutput()` returns a Shiny output UI element; `renderLollipop()` returns a Shiny render function.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  ui <- fluidPage(lollipopOutput("lp"))
  server <- function(input, output) {
    output$lp <- renderLollipop({
      lollipop(position = c(100, 250, 400), label = c("A", "B", "C"),
               protein_length = 500)
    })
  }
  shinyApp(ui, server)
}
```

network *Network graph*

Description

A GPU-accelerated node-link diagram for large gene/protein interaction networks. Nodes and edges are rendered with WebGL (via sigma v3 over a graphology graph) so tens of thousands of elements stay interactive. When node coordinates are not supplied, a bounded ForceAtlas2 layout positions the nodes in the browser; otherwise the supplied x/y are used. Categorical node groups are colored from a colorblind-safe palette. Hovering a node highlights it and its neighbors; zoom and pan use sigma's camera. In a Shiny app, clicking a node sets `input$<id>_selected` to the clicked node id, and clicking the empty canvas clears it.

Usage

```
network(
  nodes,
  edges,
  layout = c("forceatlas2", "precomputed"),
  iterations = 200,
  default_node_color = "#7c8598",
  default_edge_color = "#d6dae1",
  label_threshold = 8,
  default_node_size = 4,
  directed = FALSE,
  palette = NULL,
  theme = NULL,
  width = NULL,
  height = NULL,
  element_id = NULL
)
```

Arguments

nodes	A data frame of nodes. Must contain an `id` column (coerced to character). Optional columns: `x`, `y` (precomputed coordinates), `size` (node radius in px), `group` (categorical, mapped to a palette color) and `label` (display name; defaults to `id`).
edges	A data frame of edges with columns `source` and `target` holding node ids, an optional numeric `weight` column (mapped to edge width), and an optional `color` column giving a per-edge color (else `default_edge_color`).
layout	Either `"forceatlas2"` (run a layout when coordinates are missing) or `"precomputed"` (require and use x/y from `nodes`).
iterations	Number of ForceAtlas2 iterations (bounded internally).
default_node_color	Fallback node color for nodes without a `group`.

`default_edge_color`
: Edge color.

`label_threshold`
: Minimum node size (px) for its label to render.

`default_node_size`
: Node radius (px) used when `size` is absent.

`directed`
: Draw the graph as directed, with arrowheads. When `TRUE`, `A -> B` and `B -> A` are kept as distinct edges; when `FALSE` (the default) the graph is undirected and a reciprocal pair collapses to one line.

`palette`
: Optional character vector of hex colors overriding the default categorical palette used for node groups.

`theme`
: Optional named list of theme overrides (colors, fonts, ...) merged over the component defaults in the browser. `NULL` uses the default theme.

`width, height`
: Widget dimensions (any valid CSS size).

`element_id`
: Optional explicit DOM id.

Value

An `htmlwidget` object.

Examples

```
set.seed(1)
nodes <- data.frame(
  id = paste0("N", 1:60),
  group = sample(c("A", "B", "C"), 60, replace = TRUE)
)
edges <- data.frame(
  source = paste0("N", sample(1:60, 120, replace = TRUE)),
  target = paste0("N", sample(1:60, 120, replace = TRUE))
)
network(nodes, edges)
```

`network-shiny` *Shiny bindings for network*

Description

Output and render functions for using `network()` within Shiny applications and interactive R Markdown / Quarto documents.

Usage

```
networkOutput(output_id, width = "100%", height = "480px")

renderNetwork(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

output_id	Output variable to read from.
width, height	Element size, passed to htmlwidgets::shinyWidgetOutput().
expr	An expression that generates a network() widget.
env	The environment in which to evaluate expr.
quoted	Is expr already quoted? Defaults to FALSE.

Details

In a Shiny app, clicking a node pushes its **id** (a character scalar, not a row index) to input$<output_id>_selected, so networkOutput("graph") populates input$graph_selected. It is NULL until the first click.

Value

networkOutput() returns a Shiny output UI element; renderNetwork() returns a Shiny render function.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  ui <- fluidPage(networkOutput("net"))
  server <- function(input, output) {
    output$net <- renderNetwork({
      network(
        nodes = data.frame(id = c("A", "B", "C")),
        edges = data.frame(source = c("A", "B"), target = c("B", "C"))
      )
    })
  }
  shinyApp(ui, server)
}
```

---

oncoplot *Oncoplot (OncoPrint)*

Description

The cohort alteration landscape: a gene x sample grid of categorical alteration classes, with a per-sample burden barplot above, a per-gene frequency barplot to the right, and optional clinical annotation strips below. The grid is drawn on a canvas, so cohort-scale matrices (hundreds of genes by thousands of samples) stay interactive; labels and the legend are a crisp vector overlay.

Usage

```
oncoplot(
  alterations,
  genes = NULL,
  samples = NULL,
  classes = NULL,
  class_colors = NULL,
  burden = NULL,
  annotations = NULL,
  show_burden = TRUE,
  show_frequency = TRUE,
  show_annotations = TRUE,
  show_legend = TRUE,
  empty_color = "#EFE9DC",
  burden_color = "#0E7175",
  frequency_color = "#ED773C",
  x_label = "samples",
  burden_label = "alterations",
  cell_gap_x = 0.12,
  cell_gap_y = 0.16,
  theme = NULL,
  width = NULL,
  height = NULL,
  element_id = NULL
)
```

Arguments

`alterations`	A data frame of altered pairs with columns `gene`, `sample` and `class`. Unaltered pairs are simply absent; the full grid is reconstructed from `genes` and `samples`.
`genes`	Character vector of genes, top row first. Defaults to the genes present in `alterations`, most frequently altered first.
`samples`	Character vector of samples, left column first. Defaults to the memo-sorted order.
`classes`	Character vector of alteration classes, which fixes both the legend order and the colour assignment. Defaults to the classes present.
`class_colors`	Character vector of hex colours, one per entry of `classes`. `NULL` uses the component's categorical palette.
`burden`	Numeric vector, one value per sample, for the top barplot. Defaults to the number of altered genes per sample.
`annotations`	Optional list of clinical strips. Each element is a list with `name` (character scalar), `values` (one value per sample) and an optional `colors`.
`show_burden, show_frequency, show_annotations, show_legend`	Toggle the surrounding panels.
`empty_color`	Fill for a gene x sample cell with no alteration.

`burden_color`, `frequency_color`
: Hex fills for the per-sample burden bars above the grid and the per-gene frequency bars to its right.

`x_label`, `burden_label`
: Axis titles for the sample axis and the burden barplot.

`cell_gap_x`, `cell_gap_y`
: Gap between cells as a fraction of cell size. Set both to `0` for a solid block, which is what you want once a cohort is wide enough that the gaps eat more pixels than the cells.

`theme`
: Optional named list of theme overrides merged over the component defaults in the browser.

`width`, `height`
: Widget dimensions (any valid CSS size).

`element_id`
: Optional explicit DOM id.

Details

The component renders rows and columns in exactly the order it is given and uses the burden and frequency values as supplied. Ordering an oncoplot (memo sort, burden sort, sort by a clinical variable) is the caller's decision, and re-deriving it in the browser would let two renderings of the same data disagree. Use `oncoplot_memo_sort()` to get the conventional order.

Value

An `htmlwidget` object.

Examples

```
alt <- data.frame(
  gene = c("TP53", "TP53", "PIK3CA", "PIK3CA", "GATA3"),
  sample = c("S1", "S2", "S2", "S3", "S1"),
  class = c("Missense", "Truncating", "Missense", "Amplification", "Missense")
)
oncoplot(alt)
```

---

oncoplot-shiny *Shiny bindings for oncoplot*

Description

Output and render functions for using `oncoplot()` within Shiny applications and interactive R Markdown / Quarto documents.

Usage

```
oncoplotOutput(output_id, width = "100%", height = "560px")

renderOncoplot(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

output_id	Output variable to read from.
width, height	Element size, passed to htmlwidgets::shinyWidgetOutput().
expr	An expression that generates an oncoplot() widget.
env	The environment in which to evaluate expr.
quoted	Is expr already quoted? Defaults to FALSE.

Value

oncoplotOutput() returns a Shiny output UI element; renderOncoplot() returns a Shiny render function.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  ui <- fluidPage(oncoplotOutput("onco"))
  server <- function(input, output) {
    output$onco <- renderOncoplot({
      oncoplot(data.frame(
        gene = c("TP53", "KRAS"), sample = c("S1", "S2"),
        class = "missense"
      ))
    })
  }
  shinyApp(ui, server)
}
```

---

oncoplot_memo_sort *Conventional oncoplot row and column ordering*

Description

Genes are ordered by descending alteration frequency, then samples are ordered so that the most frequently altered gene's carriers come first, ties broken by the next gene down. This is the "memo sort" cBioPortal popularised; it is what makes mutual exclusivity between drivers visible as a staircase.

Usage

```
oncoplot_memo_sort(alterations, genes = NULL, samples = NULL)
```

Arguments

alterations	A data frame with columns gene, sample and class.
genes	Optional character vector to use instead of the derived gene order.
samples	Optional character vector to use instead of the derived sample order.

Details

Exposed separately so a caller can compute the order once and reuse it for both an interactive `oncoplot()` and a static rendering, rather than letting two implementations tie-break differently.

Value

A list with `genes` and `samples` character vectors.

Examples

```
alt <- data.frame(
  gene   = c("TP53", "TP53", "KRAS", "KRAS", "EGFR"),
  sample = c("S1",   "S2",   "S2",   "S3",   "S1"),
  class  = "missense"
)
oncoplot_memo_sort(alt)
```

`spatial` *Spatial map over a tissue image*

Description

Measurements plotted at their real coordinates on a slide, drawn on top of the histology image they came from. This is the layout of a spatial transcriptomics experiment: for a spatial assay the tissue *is* the axis, and a cluster tracing the edge of an invasive front says something an embedding cannot.

Usage

```
spatial(
  data,
  image,
  img_width,
  img_height,
  spot_diameter = 4,
  levels = NULL,
  colors = NULL,
  color_mode = c("auto", "categorical", "continuous"),
  colormap = c("viridis", "rdbu", "ltc", "ltcdiv"),
  spot_scale = 1,
  spot_opacity = 0.85,
  image_opacity = 1,
  show_image = TRUE,
  show_legend = TRUE,
  theme = NULL,
  width = NULL,
  height = NULL,
  element_id = NULL
)
```

Arguments

`data`	A data frame with numeric `x` and `y` columns giving spot centres **in image pixel coordinates**. Optional `color` (character or numeric) and `label` (tooltip text) columns.
`image`	URL or path of the tissue image, as the browser will fetch it.
`img_width`, `img_height`	Natural size of that image in pixels.
`spot_diameter`	Spot diameter in image pixels.
`levels`, `colors`	Character vectors fixing the categorical order and colours. `NULL` derives them from the data and the theme palette.
`color_mode`	`"auto"` (detect from the column type), `"categorical"` or `"continuous"`.
`colormap`	Sequential ramp for continuous colouring: `"viridis"`, `"rdbu"`, `"ltc"` (an earthy teal to sand to rust sequential ramp) or `"ltcdiv"` (its diverging counterpart, neutral cream at the midpoint).
`spot_scale`	Multiplier on `spot_diameter`; 1 draws true size.
`spot_opacity`, `image_opacity`	Opacities in `[0, 1]`. Lower the spot opacity to read the histology underneath.
`show_image`, `show_legend`	Toggle the underlay and the legend.
`theme`	Optional named list of theme overrides.
`width`, `height`	Widget dimensions (any valid CSS size).
`element_id`	Optional explicit DOM id.

Details

The image and the spots share one "contain" fit, computed once, so histology and overlay cannot drift apart on resize, full-screen, or a high-DPI display.

`color` may be a character vector (categorical, discrete legend) or numeric (continuous, sequential ramp with a colourbar), which is what lets one view toggle between colouring by cluster and by a gene's expression.

Value

An `htmlwidget` object.

Scale

Spots are drawn one at a time on a 2-D canvas, which suits a Visium-scale slide of a few thousand spots. This is **not** a renderer for Xenium or CosMx-scale single-cell output: a million cells will not stay interactive here. For that many points use `embedding()`, which draws on the GPU via regl-scatterplot, and accept that it has no image underlay. Plotting both a histology image and a million single cells is not something this package currently does.

Examples

```
df <- data.frame(
  x = c(100, 150, 200), y = c(120, 160, 90),
  color = c("Cluster 1", "Cluster 2", "Cluster 1")
)
spatial(df, image = "tissue.png", img_width = 600, img_height = 600,
        spot_diameter = 8)
```

`spatial-shiny` *Shiny bindings for spatial*

Description

Output and render functions for using `spatial()` within Shiny applications and interactive R Markdown / Quarto documents.

Usage

```
spatialOutput(output_id, width = "100%", height = "560px")

renderSpatial(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

`output_id`	Output variable to read from.
`width, height`	Element size, passed to `htmlwidgets::shinyWidgetOutput()`.
`expr`	An expression that generates a `spatial()` widget.
`env`	The environment in which to evaluate `expr`.
`quoted`	Is `expr` already quoted? Defaults to `FALSE`.

Value

`spatialOutput()` returns a Shiny output UI element; `renderSpatial()` returns a Shiny render function.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  ui <- fluidPage(spatialOutput("sp"))
  server <- function(input, output) {
    output$sp <- renderSpatial({
      spatial(x = runif(50), y = runif(50), color = rnorm(50))
    })
  }
  shinyApp(ui, server)
}
```

`treemap` *Gene / pathway treemap*

Description

A hierarchical treemap of gene-set / pathway composition. The hierarchy is built from a flat edge list with d3-hierarchy (`stratify` + `treemap`) and tiles are rendered on a canvas so thousands of leaves stay interactive; tile labels and a drill-down breadcrumb are drawn as a crisp vector overlay. Click a tile to zoom into that node and the breadcrumb to zoom back out.

Usage

```
treemap(
  data,
  tile = c("squarify", "binary"),
  padding_inner = 1,
  color_by = c("parent", "value"),
  colormap = c("viridis", "rdbu"),
  label_min_size = 32,
  theme = NULL,
  width = NULL,
  height = NULL,
  element_id = NULL
)
```

Arguments

`data`	A data frame describing a tree as an edge list. Required columns: `id` (unique node id) and `parent` (id of the parent; the root's parent is `NA` or `""`). A numeric `value` column supplies leaf weights (internal nodes are summed automatically); an optional `label` column supplies display names.
`tile`	Tiling algorithm: `"squarify"` (golden-ratio rectangles) or `"binary"` (balanced binary partition).
`padding_inner`	Padding between sibling tiles, in pixels.
`color_by`	Color leaves by `"parent"` (top-level ancestor, categorical) or by `"value"` (a sequential/diverging ramp).
`colormap`	Ramp used when `color_by = "value"`: `"viridis"` or `"rdbu"`.
`label_min_size`	Minimum tile side (px) before a label is drawn.
`theme`	Optional named list of theme overrides (colors, fonts, ...) merged over the component defaults in the browser. `NULL` uses the default theme.
`width, height`	Widget dimensions (any valid CSS size).
`element_id`	Optional explicit DOM id.

Value

An `htmlwidget` object.

Examples

```
df <- data.frame(
  id = c("root", "P1", "P2", "g1", "g2", "g3"),
  parent = c(NA, "root", "root", "P1", "P1", "P2"),
  value = c(0, 0, 0, 3, 5, 2),
  label = c("All", "Pathway 1", "Pathway 2", "Gene 1", "Gene 2", "Gene 3")
)
treemap(df)
```

`treemap-shiny` *Shiny bindings for treemap*

Description

Output and render functions for using `treemap()` within Shiny applications and interactive R Markdown / Quarto documents.

Usage

```
treemapOutput(output_id, width = "100%", height = "480px")

renderTreemap(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

`output_id`	Output variable to read from.
`width, height`	Element size, passed to `htmlwidgets::shinyWidgetOutput()`.
`expr`	An expression that generates a `treemap()` widget.
`env`	The environment in which to evaluate `expr`.
`quoted`	Is `expr` already quoted? Defaults to `FALSE`.

Value

`treemapOutput()` returns a Shiny output UI element; `renderTreemap()` returns a Shiny render function.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  ui <- fluidPage(treemapOutput("tm"))
  server <- function(input, output) {
    output$tm <- renderTreemap({
      treemap(id = c("root", "A", "B"), parent = c(NA, "root", "root"),
              value = c(NA, 3, 7))
    })
  }
  shinyApp(ui, server)
}
```

upset *UpSet plot of set intersections*

Description

Set intersections as a bar chart over a membership matrix. Venn diagrams stop being readable at four sets and stop being drawable at five; UpSet replaces the areas with an explicit matrix, so it scales to dozens of sets and stays exact.

Usage

```
upset(
  data,
  sets,
  membership,
  set_sizes = NULL,
  total = NULL,
  bar_fraction = 0.55,
  show_set_sizes = TRUE,
  dot_radius = 5,
  bar_color = NULL,
  empty_dot_color = NULL,
  y_label = "intersection size",
  theme = NULL,
  width = NULL,
  height = NULL,
  element_id = NULL
)
```

Arguments

`data`	A data frame with a numeric `size` column, one row per intersection, in the order to draw them.
`sets`	Character vector of set names, top to bottom.
`membership`	Logical or 0/1 matrix, intersections x sets, saying which sets each intersection belongs to.
`set_sizes`	Numeric vector of per-set totals for the left-hand bars.
`total`	Universe size, shown in the corner.
`bar_fraction`	Fraction of the height given to the intersection bars.
`show_set_sizes`	Draw the per-set total bars.
`dot_radius`	Matrix dot radius in pixels.
`bar_color, empty_dot_color`	Fills for the bars and filled dots, and for dots not in the intersection. `NULL` uses the theme.

y_label	Axis label for the intersection bars.
theme	Optional named list of theme overrides.
width, height	Widget dimensions (any valid CSS size).
element_id	Optional explicit DOM id.

Details

Intersections are **exclusive**: a column counts the elements in precisely that combination of sets and no others. That is what makes the columns sum to the union rather than double-counting, and it is why a small A + B bar next to large A and B bars is evidence of mutual exclusivity rather than an artefact. upset_intersections() computes them from a logical matrix.

Value

An htmlwidget object.

Examples

```
m <- matrix(c(TRUE, FALSE, FALSE, TRUE, TRUE, TRUE), nrow = 3, byrow = TRUE)
upset(data.frame(size = c(40, 25, 12)), sets = c("A", "B"), membership = m)
```

---

upset-shiny *Shiny bindings for upset*

Description

Output and render functions for using upset() within Shiny applications and interactive R Markdown / Quarto documents.

Usage

```
upsetOutput(output_id, width = "100%", height = "520px")

renderUpset(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

output_id	Output variable to read from.
width, height	Element size, passed to htmlwidgets::shinyWidgetOutput().
expr	An expression that generates an upset() widget.
env	The environment in which to evaluate expr.
quoted	Is expr already quoted? Defaults to FALSE.

Value

upsetOutput() returns a Shiny output UI element; renderUpset() returns a Shiny render function.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  sets <- list(A = c("x", "y", "z"), B = c("y", "z", "w"))
  ui <- fluidPage(upsetOutput("us"))
  server <- function(input, output) {
    output$us <- renderUpset(upset(upset_intersections(sets)))
  }
  shinyApp(ui, server)
}
```

`upset_intersections` *Exclusive set intersections from a membership matrix*

Description

Counts the elements belonging to precisely each observed combination of sets. Elements in no set are excluded, since they have no column to sit in.

Usage

```
upset_intersections(m, max_n = NULL)
```

Arguments

`m`	A logical matrix, elements x sets, with set names as column names.
`max_n`	Keep only the `max_n` largest intersections. `NULL` keeps all.

Value

A list with `size` (integer vector), `membership` (logical matrix, intersections x sets), `sets`, `set_sizes` and `total`.

Examples

```
m <- cbind(A = c(TRUE, TRUE, FALSE), B = c(TRUE, FALSE, TRUE))
upset_intersections(m)
```

`violin` *Stacked violin plot*

Description

One row per feature, one violin per group. A box plot hides bimodality, which in single-cell data is usually the whole story: a gene expressed in half a cluster and silent in the other half has the same median as one expressed weakly everywhere. The violin shows the shape, and stacking rows on a shared x lets a marker panel be read down the page.

Usage

```
violin(
  data,
  grid,
  density,
  grids = NULL,
  median = NULL,
  features = NULL,
  groups = NULL,
  group_colors = NULL,
  violin_width = 0.85,
  scale_per_violin = FALSE,
  show_median = TRUE,
  show_feature_labels = TRUE,
  theme = NULL,
  width = NULL,
  height = NULL,
  element_id = NULL
)
```

Arguments

`data`	A data frame with `feature` and `group` key columns, one row per violin, in the order to draw them.
`grid`	Numeric vector, the shared evaluation grid, ascending.
`density`	Numeric matrix, violins x `grid`, of density values.
`grids`	Optional numeric matrix, features x `grid`, giving each feature its own y range. Without it every row shares `grid`, which lets one highly expressed feature compress the rest into flat lines.
`median`	Optional numeric vector, one median per violin, drawn as a tick.
`features, groups`	Character vectors fixing the row and column order. Factor `feature` / `group` columns supply them from their levels.
`group_colors`	One hex colour per group. `NULL` uses the categorical palette.

`violin_width` Fraction of a cell's width the widest violin fills.

`scale_per_violin`
Scale each violin to its own maximum rather than the row's. Per-row is the default so groups stay comparable within a feature.

`show_median, show_feature_labels`
Toggle the median tick and row labels.

`theme` Optional named list of theme overrides.

`width, height` Widget dimensions (any valid CSS size).

`element_id` Optional explicit DOM id.

Details

The widget draws densities, it does not estimate them. Each violin arrives as a vector of density values on a shared grid, because kernel bandwidth choice changes what the figure claims and belongs with the data. `violin_density()` computes them from raw values with `stats::density()`.

Value

An `htmlwidget` object.

Examples

```
d <- violin_density(list(`CD3D|T` = rnorm(50, 2), `CD3D|B` = rnorm(50, 0)))
violin(d$data, grid = d$grid, density = d$density, median = d$median)
```

`violin-shiny` *Shiny bindings for violin*

Description

Output and render functions for using `violin()` within Shiny applications and interactive R Markdown / Quarto documents.

Usage

```
violinOutput(output_id, width = "100%", height = "560px")

renderViolin(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

`output_id` Output variable to read from.

`width, height` Element size, passed to `htmlwidgets::shinyWidgetOutput()`.

`expr` An expression that generates a `violin()` widget.

`env` The environment in which to evaluate `expr`.

`quoted` Is `expr` already quoted? Defaults to `FALSE`.

Value

`violinOutput()` returns a Shiny output UI element; `renderViolin()` returns a Shiny render function.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  ui <- fluidPage(violinOutput("vln"))
  server <- function(input, output) {
    output$vln <- renderViolin({
      violin(data.frame(feature = rep("G1", 100), group = "A",
                        value = rnorm(100)))
    })
  }
  shinyApp(ui, server)
}
```

---

violin_density *Kernel densities on a shared grid*

Description

Evaluates every group's density on one grid spanning all of them, which is what lets the violins be compared. Groups with fewer than two distinct values get a flat zero row rather than an error, since a cluster with one cell is a real thing to encounter.

Usage

```
violin_density(values, n = 64L, adjust = 1)
```

Arguments

`values`	A named list of numeric vectors, one per violin. Names of the form `"feature|group"` are split into the two key columns.
`n`	Grid resolution.
`adjust`	Bandwidth multiplier, passed to `stats::density()`.

Value

A list with `data` (the key columns), `grid`, `density` and `median`.

Examples

```
violin_density(list(`A|x` = rnorm(20), `A|y` = rnorm(20, 3)))
```

volcano *Volcano plot*

Description

A GPU-accelerated volcano plot for differential-expression results. Points are rendered with WebGL (via regl-scatterplot) so hundreds of thousands to millions of genes/features stay interactive, while axes, threshold guides and gene labels are drawn as crisp vector overlays.

Usage

```
volcano(
  data,
  fc_threshold = 1,
  p_threshold = 0.05,
  point_size = 3,
  opacity = 0.8,
  colors = NULL,
  x_label = "log2 fold change",
  y_label = "-log10 p-value",
  show_threshold_lines = TRUE,
  label_top_n = 10,
  theme = NULL,
  width = NULL,
  height = NULL,
  element_id = NULL
)
```

Arguments

data	A data frame with numeric columns `x` (log2 fold change) and `y` (`-log10` p-value). An optional `label` column supplies gene names for tooltips and top-hit labels.
fc_threshold	Absolute log2 fold-change cutoff for calling a hit.
p_threshold	P-value cutoff (applied on the `-log10` scale).
point_size	Point radius in pixels.
opacity	Point opacity in `[0, 1]`.
colors	Optional named list of hex colors for the three point classes: `up`, `down` and `ns` (not significant). `NULL` (the default) uses the component's built-in palette.
x_label, y_label	Axis titles.
show_threshold_lines	Draw the fold-change / p-value threshold guides.
label_top_n	Number of top up- and down-regulated genes to label.

theme	Optional named list of theme overrides (colors, fonts, ...) merged over the component defaults in the browser. NULL uses the default theme.
width, height	Widget dimensions (any valid CSS size).
element_id	Optional explicit DOM id.

Value

An htmlwidget object.

Examples

```
set.seed(1)
df <- data.frame(
  x = rnorm(2000),
  y = abs(rnorm(2000)) * 3,
  label = paste0("GENE", seq_len(2000))
)
volcano(df)
```

---

volcano-shiny *Shiny bindings for volcano*

Description

Output and render functions for using volcano() within Shiny applications and interactive R Markdown / Quarto documents.

Usage

```
volcanoOutput(output_id, width = "100%", height = "480px")

renderVolcano(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

output_id	Output variable to read from.
width, height	Element size, passed to htmlwidgets::shinyWidgetOutput().
expr	An expression that generates a volcano() widget.
env	The environment in which to evaluate expr.
quoted	Is expr already quoted? Defaults to FALSE.

Value

volcanoOutput() returns a Shiny output UI element; renderVolcano() returns a Shiny render function.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  ui <- fluidPage(volcanoOutput("v"))
  server <- function(input, output) {
    output$v <- renderVolcano({
      volcano(data.frame(x = rnorm(100), y = abs(rnorm(100)) * 3))
    })
  }
  shinyApp(ui, server)
}
```

Index