Package 'glyph'

October 21, 2026

Title A Grammar of Interactive Graphics

Version 0.2.0

Description Builds interactive graphics from a declarative specification. Marks, scales, tooltips, zoom, brushing, linked selections, animated transitions and multi-plot layouts are declared in one pipeline and stored as a plain R list that can be inspected, saved as JSON or converted to 'Vega-Lite'. Plots are drawn in the browser with 'D3.js' and work at the console, in R Markdown and 'Quarto' documents, and in 'Shiny' apps, where brushed selections are returned to R.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1.0)

Config/testthat/edition 3

Imports htmlwidgets (>= 1.6.0), jsonlite (>= 1.8.0), rlang (>= 1.1.0), cli (>= 3.6.0), grDevices, stats, utils, tools

Suggests testthat (>= 3.0.0), knitr, rmarkdown, chromote, htmltools, withr, shiny (>= 1.5.0), callr, httpuv, pkgload

VignetteBuilder knitr

URL https://github.com/josh45-source/glyph, https://josh45-source.github.io/glyph/

BugReports https://github.com/josh45-source/glyph/issues

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Joash Joshua Ayo [aut, cre], Mike Bostock [ctb, cph] (Author of the bundled D3.js library)

Maintainer Joash Joshua Ayo <joashjoshua789@gmail.com>

Repository CRAN

Date/Publication 2026-10-21 13:10:34 UTC

Contents

animate . . . 3
animation . . . 4
compile . . . 4
compose . . . 5
export . . . 6
facet . . . 7
glyph . . . 7
glyph-shiny . . . 8
inset . . . 10
interact . . . 10
interactivity . . . 11
layout . . . 12
marginals . . . 12
marks . . . 13
mark_area . . . 13
mark_bar . . . 14
mark_boxplot . . . 14
mark_histogram . . . 15
mark_line . . . 16
mark_link . . . 17
mark_point . . . 17
mark_polygon . . . 18
mark_ribbon . . . 19
mark_rule . . . 19
mark_sina . . . 20
mark_text . . . 21
mark_violin . . . 22
print.glyph_layout . . . 23
print.glyph_spec . . . 23
render . . . 24
rendering . . . 25
scales . . . 25
selection . . . 26
set_scale . . . 27
summary.glyph_layout . . . 28
summary.glyph_spec . . . 29
theme_tokens . . . 29
theming . . . 30
titles . . . 31
to_vegalite . . . 31

Index **32**

`animate` *Add animation/transition behavior*

Description

Add animation/transition behavior

Usage

```
animate(
  spec,
  transition = "morph",
  duration = 500,
  stagger = 0,
  by = NULL,
  easing = "ease-in-out",
  loop = FALSE
)
```

Arguments

`spec`	A glyph_spec
`transition`	Transition type: "morph", "fade", "slide", "none"
`duration`	Duration in milliseconds
`stagger`	Delay between successive marks in ms (for entrance effects)
`by`	Column that defines animation keyframes. The plot shows one value of it at a time and moves between them, e.g. `by = year` plays through the years.
`easing`	Easing function: "linear", "ease-in", "ease-out", "ease-in-out", "bounce", "elastic"
`loop`	Repeat animation (TRUE, FALSE, or number of times)

Value

Modified glyph_spec

Examples

```
# Entrance animation: bars grow from baseline
glyph(mtcars, x = cyl, y = mpg) |>
  mark_bar() |>
  animate(transition = "slide", stagger = 50)

# Keyframe animation: morph between groups
glyph(mtcars, x = wt, y = mpg) |>
  mark_point(size = hp, color = cyl) |>
  animate(by = gear, transition = "morph", duration = 800)
```

animation *Animation Grammar*

Description

Animations in glyph are declarative state transitions, not frame-by-frame rendering. You describe *what changes* and *how it should interpolate*, and the renderer handles the rest.

Details

Two animation paradigms:

- **Transitions**: animate between data states (e.g. year 2020 → 2021). Marks morph smoothly, entering/exiting as data changes.
- **Entrances**: animate marks appearing on first render (stagger bars growing from zero, points fading in).

compile *Compile a glyph spec to a resolved representation*

Description

Compile a glyph spec to a resolved representation

Usage

```
compile(spec, engine = "auto", width = NULL, height = NULL)

## S3 method for class 'glyph_spec'
compile(spec, engine = "auto", width = NULL, height = NULL)

## S3 method for class 'glyph_layout'
compile(spec, engine = "auto", width = NULL, height = NULL)
```

Arguments

spec	A glyph_spec or glyph_layout
engine	Rendering backend. Only "html" (D3.js via htmlwidgets) is currently implemented; "auto" resolves to "html". Other values are rejected — earlier versions accepted "canvas"/"webgl" and silently ignored them (no such renderer ever existed), which was misleading.
width	Width in pixels (NULL for auto)
height	Height in pixels (NULL for auto)

Value

A glyph_compiled object (a list with resolved data + JSON)

A glyph_compiled object containing the resolved spec, JSON string, and engine selection

A glyph_compiled object containing the resolved layout spec and JSON string

---

compose *Compose multiple glyph specs into a layout*

Description

Compose multiple glyph specs into a layout

Usage

```
compose(
  ...,
  type = "hstack",
  widths = NULL,
  heights = NULL,
  gap = 10,
  shared_scales = FALSE,
  linked_selections = FALSE,
  title = NULL
)
```

Arguments

...	glyph_spec objects or layout objects (for nesting)
type	Layout type: "hstack", "vstack", "grid", "wrap"
widths	Relative widths for columns (e.g. c(2, 1) for 2:1 split)
heights	Relative heights for rows
gap	Gap between plots in px
shared_scales	Share axis scales across plots: TRUE, FALSE, "x", "y", or "both"
linked_selections	Share interaction selections across plots
title	Overall layout title

Value

A glyph_layout object

Examples

```
p1 <- glyph(mtcars, x = wt, y = mpg) |> mark_point()
p2 <- glyph(mtcars, x = wt, y = hp) |> mark_point()
p3 <- glyph(mtcars, x = hp, y = mpg) |> mark_line()

# Horizontal stack
compose(p1, p2, type = "hstack")

# Grid with linked brushing
compose(p1, p2, p3, type = "wrap",
        linked_selections = TRUE, shared_scales = "x")
```

---

export *Export to various formats*

Description

Export to various formats

Usage

```
export(spec, file, width = 800, height = 600)
```

Arguments

spec	A glyph_spec or glyph_compiled
file	Output file path. Extension determines format. Currently implemented: `.html` (interactive widget) and `.json` (the compiled spec). Other extensions (`.svg`, `.png`, `.pdf`) are planned but not yet implemented, and currently just print a message.
width	Width in pixels
height	Height in pixels

Details

A single self-contained `.html` file needs pandoc. RStudio ships with pandoc, so this works out of the box there; elsewhere, install pandoc and make sure it is on your `PATH`. Without pandoc, `export()` still writes the `.html` file but puts the JavaScript it needs in a `<name>_files` folder next to it, and tells you so. Keep the two together when moving or sharing the plot.

Value

Invisibly returns the output file path

facet *Facet a plot by one or two variables*

Description

Splits a plot into small multiples, one panel for each value of `rows`, `cols`, or both. Panels share their axes unless `free_scales` lets them vary. With a single faceting variable, `wrap` sets how many panels go in each row.

Usage

```
facet(spec, rows = NULL, cols = NULL, free_scales = "none", wrap = NULL)
```

Arguments

`spec`	A glyph_spec
`rows`	Row faceting variable: a bare column name, a computed expression (e.g. `factor(cyl)`), or `NULL`
`cols`	Column faceting variable: a bare column name, a computed expression, or `NULL`
`free_scales`	"none", "x", "y", "both"
`wrap`	If only one variable, wrap into a grid with this many columns

Value

Modified glyph_spec

glyph *Create a Glyph Visualization*

Description

Entry point for the glyph grammar. Creates a specification that describes a visualization declaratively. The spec is a plain nested list: it can be inspected, serialized to JSON, and compiled into an interactive HTML widget.

Usage

```
glyph(data = NULL, ...)
```

Arguments

`data`	A data frame or tibble, or `NULL` if each mark will supply its own data.
`...`	Global aesthetic mappings, written as bare column names or expressions, e.g. `glyph(mtcars, x = wt, y = mpg)`. Every mark uses these unless it sets its own.

Value

A glyph_spec object

Examples

```
spec <- glyph(mtcars, x = wt, y = mpg) |>
  mark_point()
summary(spec)
```

---

glyph-shiny *Use glyph plots in Shiny*

Description

glyphOutput() places a plot in a Shiny app's UI and renderGlyph() draws it from the server. renderGlyph() takes a plot from glyph(), a layout from compose(), or a widget from render().

What the user selects in the plot is sent back to the server as input$<outputId>_selected, holding row numbers of the data frame given to glyph(). See the section below and vignette("shiny", package = "glyph").

shiny is not installed with glyph. Install it to use these functions.

Usage

```
glyphOutput(outputId, width = "100%", height = "400px")

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

Arguments

outputId	Output variable to read the plot from.
width, height	Size of the plot, as a CSS size such as "100%" or "400px", or a number of pixels.
expr	An expression that returns a glyph_spec, a glyph_layout or a widget made by render(). It can also return NULL, which shows nothing and clears the selection.
env	The environment in which to evaluate expr.
quoted	Is expr a quoted expression (with quote())? This is useful if you want to save an expression in a variable.

Value

glyphOutput() returns the UI element for the plot. renderGlyph() returns a render function to assign to an element of output.

Selections

A selection is sent when the user ends a brush, clicks a mark with `interact(click = "select")`, or clicks a legend entry added with `selection(type = "legend")`. It is not sent while a brush is being dragged. Clearing a brush (with the "Clear selection" button, Escape, or a click on an empty part of the plot) sends `NULL`, as does clicking a selected mark again. So does redrawing the plot, which happens when `renderGlyph()` runs again or the output is resized.

For a single plot the value is an integer vector of row numbers, so `data[input$plot_selected, ]` gives the selected rows. Row numbers are positions in the data frame, not row names. They are correct when rows with missing values were dropped and when a mapping is computed (such as `x = log(wt)`). Selecting a box, violin, histogram bin or polygon selects every row it summarises. A mark drawn from its own `data` takes no part in the value. A brush that takes in no rows gives `integer(0)`.

A legend selects the rows whose entries are still shown. With every entry shown there is no selection, and the value is `NULL`.

A `facet()` grid has one data frame, so its value is one vector: the rows selected in any of its panels.

A `compose()` layout gives a list with one entry per panel, in the order the panels were given, named after them when `compose()` was called with names. Each entry holds the rows selected in that panel, as row numbers of that panel's data, or `NULL` when nothing is selected there. A panel that is itself a layout gives a list of the same shape. When no panel has a selection the whole value is `NULL`.

```
output$plots <- renderGlyph(compose(cars = p1, iris = p2))
# input$plots_selected is NULL, or for example
# list(cars = c(3L, 18L), iris = NULL)
```

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  library(shiny)

  ui <- fluidPage(
    glyphOutput("plot"),
    tableOutput("rows")
  )

  server <- function(input, output) {
    output$plot <- renderGlyph(
      glyph(mtcars, x = wt, y = mpg) |>
        mark_point() |>
        interact(brush = TRUE)
    )
    output$rows <- renderTable(mtcars[input$plot_selected, ], rownames = TRUE)
  }

  shinyApp(ui, server)
}
```

inset *Add an inset plot*

Description

Add an inset plot

Usage

```
inset(spec, inset, position = "top-right")
```

Arguments

spec	The main glyph_spec
inset	A glyph_spec to place as an inset
position	Where to place: "top-right", "top-left", "bottom-right", "bottom-left", or a list(x, y, width, height) with proportions 0-1

Value

Modified glyph_spec

interact *Add interactive behaviors to a glyph spec*

Description

Add interactive behaviors to a glyph spec

Usage

```
interact(
  spec,
  tooltip = FALSE,
  zoom = FALSE,
  brush = FALSE,
  hover = NULL,
  click = NULL,
  crossfilter = FALSE,
  nearest = FALSE
)
```

Arguments

spec	A glyph_spec
tooltip	Show values on hover. TRUE for auto-generated, or a glue-style template string like "{x}: {y} ({color})".
zoom	Enable scroll-to-zoom and pan. The scroll wheel, a trackpad pinch, a two-finger pinch and a double-click/double-tap always zoom; dragging pans (see brush for what happens when both are enabled).
brush	Enable rectangular brush selection. When zoom is also enabled, dragging brushes by default and a small toggle next to the reset-zoom control switches dragging between selecting and panning, so one drag never does both at once.
hover	Highlight mark on hover ("enlarge", "brighten", "outline", or NULL)
click	Action on click: "select", "filter", "url", or a callback name
crossfilter	Link this plot's selections to other plots in a layout
nearest	Snap selection to nearest point (useful for line charts)

Value

Modified glyph_spec

Examples

```
glyph(mtcars, x = wt, y = mpg) |>
  mark_point(color = cyl) |>
  interact(
    tooltip = "{cyl} cylinders\n{mpg} mpg at {wt} tons",
    zoom = TRUE,
    hover = "enlarge",
    brush = TRUE,
    crossfilter = TRUE
  )
```

---

interactivity *Interactivity*

Description

Tooltips, hover effects, clicks, zoom, brushing and linked selections are part of the plot specification. They are declared with interact() and selection() in the same pipeline as the marks, stored in the spec, and run in the browser by the D3 renderer.

Details

Design philosophy: interactions are *selections* that filter or highlight marks. This follows Vega-Lite's insight that most interactions are really about defining subsets of the data. A tooltip is "select the nearest point, show its values." Brushing is "select points in a rectangle, highlight them."

layout *Layout Composition*

Description

Built-in multi-plot composition. No external packages needed. Layouts can express grids, stacks, insets, marginal plots, and arbitrary nesting. Composed plots can share selections for cross-filtering (linked brushing).

marginals *Add a marginal plot (histogram/density on axes)*

Description

Draws the distribution of the x and y variables along the edges of the plot, as a histogram, density curve or boxplot.

Usage

```
marginals(spec, x = "histogram", y = "histogram", size = 0.15)
```

Arguments

spec	A glyph_spec
x	Marginal type for x-axis: "histogram", "density", "boxplot", NULL
y	Marginal type for y-axis: "histogram", "density", "boxplot", NULL
size	Proportion of plot area for marginals (0.0 to 0.4)

Value

Modified glyph_spec

Examples

```
glyph(mtcars, x = wt, y = mpg) |>
  mark_point(color = cyl) |>
  marginals(x = "histogram", y = "density")
```

marks *Visual Marks*

Description

Marks are the layers that draw data: points, lines, bars, areas, text, rules, ribbons, links and polygons. Each `mark_*()` call adds one layer to the spec. `mark_boxplot()`, `mark_violin()`, `mark_sina()` and `mark_histogram()` summarise the distribution of a variable; their statistics are computed from the data when the plot is compiled.

Mappings are passed as bare column names or expressions, and are added to the global mappings set in `glyph()`. A mark can also take its own `data`, so one plot can draw from several data frames. Fixed visual properties that don't depend on the data, such as `size`, `opacity` or `stroke_width`, go in `style`. Interaction and animation are set for the whole plot with `interact()` and `animate()`.

mark_area *Add an area mark*

Description

Add an area mark

Usage

```
mark_area(spec, ..., data = NULL, style = list())
```

Arguments

`spec`	A glyph_spec
`...`	Aesthetic mappings (x, y, color, size, shape, alpha, tooltip)
`data`	Optional per-mark data override
`style`	Named list of fixed visual properties

Value

Modified `glyph_spec` object with the area mark added

mark_bar *Add a bar mark*

Description

Add a bar mark

Usage

```
mark_bar(spec, ..., data = NULL, style = list(), orient = "vertical")
```

Arguments

spec	A glyph_spec
...	Aesthetic mappings (x, y, color, size, shape, alpha, tooltip)
data	Optional per-mark data override
style	Named list of fixed visual properties
orient	"vertical" or "horizontal"

Value

Modified glyph_spec object with the bar mark added

mark_boxplot *Add a boxplot mark*

Description

Draws one box per group: the box spans the first to third quartile, the line inside it is the median, and the whiskers reach the most extreme values within coef times the interquartile range of the box. Values beyond the whiskers are drawn as outlier points.

Map y to the numeric variable and x to the grouping variable, which is treated as categorical even when it is numeric. Without x, all rows form one box. Mapping color to a different variable than x places one box per colour side by side within each x group.

The statistics are computed in R when the plot is compiled, using stats::quantile() with its default type = 7. Rows with a missing x or y are dropped first, with a warning.

Usage

```
mark_boxplot(
  spec,
  ...,
  data = NULL,
  style = list(),
  coef = 1.5,
  outliers = TRUE,
  width = 0.7
)
```

Arguments

spec	A glyph_spec
...	Aesthetic mappings: y (required, numeric), x and color.
data	Optional per-mark data override
style	Named list of fixed visual properties
coef	Length of the whiskers as a multiple of the interquartile range. Use Inf to extend them to the minimum and maximum.
outliers	Whether to draw values beyond the whiskers as points.
width	Width of each box as a fraction of the space for its group, between 0 and 1.

Value

Modified glyph_spec object with the boxplot mark added

Examples

```
glyph(mtcars, x = factor(cyl), y = mpg) |>
  mark_boxplot()
```

---

mark_histogram *Add a histogram mark*

Description

Counts how many rows fall into each bin of a numeric x variable and draws one bar per bin. Bins are closed on the right, like graphics::hist(), and the lowest bin also includes its left edge. Mapping color stacks one bar segment per colour within each bin.

With neither bins nor binwidth, the number of bins follows Sturges' rule and the edges are rounded to tidy values, as in graphics::hist(). All facet panels share the same bins.

The counts are computed in R when the plot is compiled. Rows with a missing x are dropped first, with a warning.

Usage

```
mark_histogram(
  spec,
  ...,
  data = NULL,
  style = list(),
  bins = NULL,
  binwidth = NULL
)
```

Arguments

spec	A glyph_spec
...	Aesthetic mappings: x (required, numeric) and color. There is no y mapping; the bar heights are the counts.
data	Optional per-mark data override
style	Named list of fixed visual properties
bins	Number of equal-width bins spanning the range of x.
binwidth	Width of each bin. Bin edges are multiples of binwidth. Takes precedence over bins.

Value

Modified glyph_spec object with the histogram mark added

Examples

```
glyph(mtcars, x = mpg) |>
  mark_histogram(binwidth = 2.5)
```

---

mark_line *Add a line mark*

Description

Add a line mark

Usage

```
mark_line(spec, ..., data = NULL, style = list(), interpolate = "monotone")
```

Arguments

spec	A glyph_spec
...	Aesthetic mappings (x, y, color, size, shape, alpha, tooltip)
data	Optional per-mark data override
style	Named list of fixed visual properties
interpolate	Interpolation method: "linear", "monotone", "step", "basis"

Value

Modified glyph_spec object with the line mark added

---

mark_link *Add a link/edge mark (for networks, sankeys, slope graphs)*

Description

Add a link/edge mark (for networks, sankeys, slope graphs)

Usage

```
mark_link(spec, ..., data = NULL, style = list())
```

Arguments

spec	A glyph_spec
...	Aesthetic mappings (x, y, color, size, shape, alpha, tooltip)
data	Optional per-mark data override
style	Named list of fixed visual properties

Value

Modified glyph_spec object with the link mark added

---

mark_point *Add a point mark (scatterplot)*

Description

Add a point mark (scatterplot)

Usage

```
mark_point(spec, ..., data = NULL, style = list())
```

Arguments

spec	A glyph_spec
...	Aesthetic mappings (x, y, color, size, shape, alpha, tooltip)
data	Optional per-mark data override
style	Named list of fixed visual properties

Value

Modified glyph_spec

Examples

```
spec <- glyph(mtcars, x = wt, y = mpg) |>
  mark_point(color = cyl, size = hp, tooltip = "{cyl} cyl, {mpg} mpg")
```

`mark_polygon` *Add a polygon mark*

Description

Draws closed shapes from `x`/`y` vertices. Rows that share a value of `group` are the vertices of one polygon, joined in row order, and the last vertex is joined back to the first. Without a `group` mapping, all rows form one polygon. Mapping `color` fills each polygon by its value; a polygon whose rows have different colours is split into one polygon per colour.

Each polygon is one element: its tooltip shows the polygon rather than a single vertex, and brushing or clicking it selects all of its rows. A tooltip template can use any column that has the same value on every row of the polygon, such as a name or an id. Rows with a missing `x` or `y` are dropped first, with a warning, and polygons left with fewer than three vertices are not drawn.

Usage

```
mark_polygon(spec, ..., data = NULL, style = list(), radius = 0)
```

Arguments

`spec`	A glyph_spec
`...`	Aesthetic mappings: `x` and `y` (required, numeric), `group` and `color`.
`data`	Optional per-mark data override
`style`	Named list of fixed visual properties: `fill` (a colour used when `color` isn't mapped), `stroke` (outline colour, the theme's foreground colour by default), `stroke_width` (in pixels, 0 for no outline) and `opacity` (of the fill).
`radius`	Radius of rounded corners, in pixels. Each corner is rounded by at most half the length of its shorter edge. 0 (the default) keeps sharp corners.

Value

Modified `glyph_spec` object with the polygon mark added

Examples

```
triangles <- data.frame(
  id = rep(c("a", "b"), each = 3),
  x = c(0, 1, 0.5, 1.5, 2.5, 2),
  y = c(0, 0, 1, 0, 0, 1)
)
glyph(triangles, x = x, y = y) |>
  mark_polygon(group = id, color = id)
```

---

mark_ribbon *Add a ribbon/band mark (for confidence intervals, ranges)*

Description

Add a ribbon/band mark (for confidence intervals, ranges)

Usage

```
mark_ribbon(spec, ..., data = NULL, style = list())
```

Arguments

spec	A glyph_spec
...	Aesthetic mappings (x, y, color, size, shape, alpha, tooltip)
data	Optional per-mark data override
style	Named list of fixed visual properties

Value

Modified glyph_spec object with the ribbon mark added

---

mark_rule *Add a rule (reference line) mark*

Description

Add a rule (reference line) mark

Usage

```
mark_rule(
  spec,
  ...,
  data = NULL,
  style = list(),
  x_intercept = NULL,
  y_intercept = NULL
)
```

Arguments

spec	A glyph_spec
...	Aesthetic mappings (x, y, color, size, shape, alpha, tooltip)
data	Optional per-mark data override
style	Named list of fixed visual properties
x_intercept	Fixed x position for vertical rule
y_intercept	Fixed y position for horizontal rule

Value

Modified glyph_spec object with the rule mark added

---

mark_sina *Add a sina mark*

Description

Draws every row as a point, spread sideways within its group in proportion to the density of values at that height. The points fill the same outline a mark_violin() with the same settings would draw, so the two combine well.

Each point is still a row of the data, so tooltips, brushing and linked selections work on individual rows, as they do for mark_point(). The sideways positions are computed in R when the plot is compiled and are the same every time.

Usage

```
mark_sina(
  spec,
  ...,
  data = NULL,
  style = list(),
  bw = "nrd0",
  adjust = 1,
  width = 0.9
)
```

Arguments

spec	A glyph_spec
...	Aesthetic mappings: y (required, numeric), x, color and size.
data	Optional per-mark data override
style	Named list of fixed visual properties
bw	Smoothing bandwidth, passed to stats::density(): a number or the name of a bandwidth rule such as "nrd0" (the default) or "SJ".

adjust	Multiplier for the bandwidth. Values above 1 give smoother curves.
width	Width of the spread at the highest density, as a fraction of the space for its group, between 0 and 1.

Value

Modified glyph_spec object with the sina mark added

Examples

```
glyph(mtcars, x = factor(cyl), y = mpg) |>
  mark_violin() |>
  mark_sina()
```

---

mark_text *Add a text/label mark*

Description

Add a text/label mark

Usage

```
mark_text(spec, ..., data = NULL, style = list(), smart_repel = TRUE)
```

Arguments

spec	A glyph_spec
...	Aesthetic mappings (x, y, color, size, shape, alpha, tooltip)
data	Optional per-mark data override
style	Named list of fixed visual properties
smart_repel	Automatically avoid label overlaps (TRUE by default). Inspired by **ggrepel**, but simpler: rather than a full force-directed simulation, glyph resolves overlaps by iterative pairwise nudging. It repeatedly compares every pair of label bounding boxes and, for any pair that still overlaps, pushes them a small step apart vertically (up to 50 passes, stopping as soon as none overlap).

Value

Modified glyph_spec object with the text mark added

`mark_violin` *Add a violin mark*

Description

Draws the distribution of a numeric variable for each group as a mirrored kernel density curve, trimmed to the range of the group's values. All violins in the mark use the same density scale, so their widths can be compared directly.

Map `y` to the numeric variable and `x` to the grouping variable, which is treated as categorical even when it is numeric. Without `x`, all rows form one violin. Mapping `color` to a different variable than `x` places one violin per colour side by side within each x group.

The densities are computed in R when the plot is compiled, with `stats::density()`. A group needs at least two distinct values; groups with fewer are left out with a warning.

Usage

```
mark_violin(
  spec,
  ...,
  data = NULL,
  style = list(),
  bw = "nrd0",
  adjust = 1,
  width = 0.9
)
```

Arguments

`spec`	A glyph_spec
`...`	Aesthetic mappings: `y` (required, numeric), `x` and `color`.
`data`	Optional per-mark data override
`style`	Named list of fixed visual properties
`bw`	Smoothing bandwidth, passed to `stats::density()`: a number or the name of a bandwidth rule such as `"nrd0"` (the default) or `"SJ"`.
`adjust`	Multiplier for the bandwidth. Values above 1 give smoother curves.
`width`	Width of the widest violin as a fraction of the space for its group, between 0 and 1.

Value

Modified `glyph_spec` object with the violin mark added

Examples

```
glyph(mtcars, x = factor(cyl), y = mpg) |>
  mark_violin()
```

print.glyph_layout *Draw a glyph layout when it is printed*

Description

A printed layout draws itself, the same way a single plot does. At the console it opens in the viewer, and in R Markdown, Quarto and vignettes it is embedded in the document. Call `render()` only to set the width and height, and use summary() for a text description.

In a non-interactive session that isn't knitting a document, such as `Rscript`, there is nowhere to draw, so printing gives a one-line description instead.

Usage

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

## S3 method for class 'glyph_layout'
knit_print(x, ..., options = NULL)
```

Arguments

`x`	A glyph_layout object
`...`	Passed on to knitr when knitting; otherwise ignored.
`options`	knitr chunk options, supplied by knitr.

Value

`print()` invisibly returns `x`. `knit_print()` returns the rendered widget's HTML for knitr to include in the document.

print.glyph_spec *Draw a glyph spec when it is printed*

Description

A printed spec draws itself. At the console it opens in the viewer, and in R Markdown, Quarto and vignettes it is embedded in the document. Call `render()` only to set the width and height, and use summary() for a text description.

In a non-interactive session that isn't knitting a document, such as `Rscript`, there is nowhere to draw, so printing gives a one-line description instead.

Usage

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

## S3 method for class 'glyph_spec'
knit_print(x, ..., options = NULL)
```

Arguments

x	A glyph_spec object
...	Passed on to knitr when knitting; otherwise ignored.
options	knitr chunk options, supplied by knitr.

Value

`print()` invisibly returns `x`. `knit_print()` returns the rendered widget's HTML for knitr to include in the document.

`render` *Render a glyph spec as an htmlwidget*

Description

Render a glyph spec as an htmlwidget

Usage

```
render(spec, width = NULL, height = NULL)
```

Arguments

spec	A glyph_spec, glyph_layout, or glyph_compiled
width	Widget width
height	Widget height

Value

An `htmlwidget` object that renders the visualization in an HTML viewer

`rendering` *Rendering Pipeline*

Description

The rendering pipeline compiles a glyph_spec into output. The spec is first resolved (evaluate quosures, drop rows with missing mapped values, merge defaults), then serialized to JSON, then handed to a backend:

- **"html"** (default): htmlwidgets + D3.js for interactive viewing. This is currently the only implemented renderer; the visual output is SVG, drawn by D3 inside the widget.
- **"svg"**, **"canvas"**, **"webgl"**, **"pdf"**: standalone static/ large-data export backends. Planned, not yet implemented.

Details

A glyph spec is a plain data structure, and nothing is drawn until the `compile()` step resolves it into a concrete render tree. That means you can:

- Inspect the compiled spec as JSON (for debugging or export)
- Serialize it and render on a different machine
- Export it to Vega-Lite JSON (near 1:1 mapping)
- Add new backends later without changing the user-facing spec API

`scales` *Scales*

Description

Scales map data values to visual properties. Every scale is set with one function, `set_scale(<aesthetic>, <type>, ...)`, where the aesthetic is the channel (such as `"x"` or `"color"`) and the type is how values are mapped (such as `"log"` or `"time"`). `scale_color()`, `scale_log()` and `scale_time()` are shortcuts for common cases.

Usage

```
scale_color(spec, palette = "Tableau10", ...)

scale_log(spec, aesthetic = "y", base = 10, ...)

scale_time(spec, aesthetic = "x", ...)
```

Arguments

spec	A glyph_spec
palette	A named palette: "viridis", "Set2", "Tableau10", "Blues", etc.
...	Additional arguments passed to set_scale()
aesthetic	Which channel: "x", "y", "color", "size", "shape", "alpha"
base	Log base (default 10)

Value

Modified glyph_spec object

Modified glyph_spec object

Modified glyph_spec object

Functions

- scale_color(): Set color palette by name
- scale_log(): Log-transform an axis
- scale_time(): Time/date axis

---

selection *Add a selection parameter (advanced)*

Description

For complex interactions: define a named selection that can be referenced by marks and scales. This is how you build linked views, conditional encoding, and interactive legends.

Usage

```
selection(
  spec,
  name,
  type = "point",
  on = "click",
  fields = NULL,
  resolve = "global"
)
```

Arguments

spec	A glyph_spec
name	Selection name (referenced in conditional encodings)
type	"point", "interval", or "legend"
on	Event trigger: "click", "mouseover", "drag"
fields	Which data fields the selection projects onto
resolve	For composed views: "global", "union", "intersect"

Value

Modified glyph_spec

Examples

```
# Interactive legend: click legend entries to filter
glyph(mtcars, x = wt, y = mpg) |>
  mark_point(color = cyl) |>
  selection("legend_filter", type = "legend", fields = "cyl")
```

---

set_scale *Define a scale for an aesthetic channel*

Description

Define a scale for an aesthetic channel

Usage

```
set_scale(
  spec,
  aesthetic,
  type = "auto",
  domain = NULL,
  range = NULL,
  nice = TRUE,
  zero = FALSE,
  reverse = FALSE,
  label = NULL,
  format = NULL,
  base = NULL
)
```

Arguments

spec	A glyph_spec
aesthetic	Which channel: "x", "y", "color", "size", "shape", "alpha"
type	Scale type: "linear", "log", "sqrt", "time", "ordinal", "band", "quantize", "threshold"
domain	Explicit domain (data range). NULL for auto. A domain set here is used as given, so it is also how to start the value axis of a bar, area or histogram somewhere other than zero.
range	Explicit output range. For color: a palette name or vector.
nice	Round domain to nice values (TRUE/FALSE)
zero	Force zero in domain (TRUE/FALSE). Bar, area and histogram marks always include zero on their value axis unless domain is set.

reverse	Reverse the scale
label	Axis/legend label (NULL for auto from column name)
format	Format string for tick labels (e.g. "$.2f", "%b %Y")
base	Log base, used when type = "log" (default 10 if NULL)

Value

Modified glyph_spec

Examples

```
glyph(mtcars, x = wt, y = mpg) |>
  mark_point(color = cyl) |>
  set_scale("y", "linear", zero = TRUE, label = "Miles per gallon") |>
  set_scale("color", "ordinal", range = "Set2")
```

---

summary.glyph_layout *Summarize a glyph layout*

Description

Summarize a glyph layout

Usage

```
## S3 method for class 'glyph_layout'
summary(object, ...)
```

Arguments

object	A glyph_layout object
...	Additional arguments (ignored)

Value

No return value, called for side effects (prints a summary of the layout to the console)

summary.glyph_spec *Summarize a glyph specification*

Description

Summarize a glyph specification

Usage

```
## S3 method for class 'glyph_spec'
summary(object, ...)
```

Arguments

object	A glyph_spec object
...	Additional arguments (ignored)

Value

No return value, called for side effects (prints a summary of the specification to the console)

theme_tokens *Set theme tokens*

Description

Set theme tokens

Usage

```
theme_tokens(
  spec,
  preset = NULL,
  font = NULL,
  font_size = NULL,
  bg = NULL,
  fg = NULL,
  accent = NULL,
  grid = NULL,
  border = NULL,
  padding = NULL,
  title_size = NULL
)
```

Arguments

spec	A glyph_spec
preset	A named preset: "light" (default), "dark", "minimal", "publication", "presentation". Presets set all tokens to coherent defaults.
font	Font family for all text
font_size	Base font size in px (axis labels scale relative to this)
bg	Background color
fg	Primary text/axis color
accent	Primary accent color (used for single-series marks)
grid	Grid line visibility: TRUE, FALSE, "x", "y"
border	Plot border: TRUE/FALSE
padding	Padding around the plot area in px
title_size	Title font size multiplier (relative to font_size)

Value

Modified glyph_spec

Examples

```
glyph(mtcars, x = wt, y = mpg) |>
  mark_point() |>
  theme_tokens(preset = "dark")

glyph(mtcars, x = wt, y = mpg) |>
  mark_point() |>
  theme_tokens(font = "IBM Plex Sans", bg = "#fafafa", grid = "y")
```

---

theming *Token-Based Theming*

Description

A theme is set with design tokens: a small set of named values (font, colours, spacing, sizes) that cascade through the whole plot, much like CSS custom properties.

Details

Tokens cascade: setting bg changes the background, but also automatically adjusts text color for contrast, grid line opacity, and tooltip styling. You override only what you want; everything else adapts.

titles *Add a title, subtitle, or caption*

Description

Add a title, subtitle, or caption

Usage

```
titles(spec, title = NULL, subtitle = NULL, caption = NULL)
```

Arguments

spec	A glyph_spec
title	Main title
subtitle	Subtitle (below title)
caption	Caption (bottom of plot, for source attribution)

Value

Modified glyph_spec object with updated title metadata

to_vegalite *Export the spec as Vega-Lite JSON (interop)*

Description

Because glyph's spec is structurally similar to Vega-Lite, we can export to Vega-Lite JSON for use in Python (Altair), JavaScript, or the Vega Editor.

Usage

```
to_vegalite(spec)
```

Arguments

spec	A glyph_spec

Value

A JSON string in Vega-Lite format

Index

animate, 3
animate(), *13*
animation, 4

compile, 4
compose, 5
compose(), *8, 9*

export, 6

facet, 7
facet(), *9*

glyph, 7
glyph(), *8*, *13*
glyph-shiny, 8
glyphOutput *(*glyph-shiny*)*, 8
graphics::hist(), *15*

inset, 10
interact, 10
interact(), *11*, *13*
interactivity, 11

knit_print.glyph_layout
        *(*print.glyph_layout*)*, 23
knit_print.glyph_spec
        *(*print.glyph_spec*)*, 23

layout, 12

marginals, 12
mark_area, 13
mark_bar, 14
mark_boxplot, 14
mark_boxplot(), *13*
mark_histogram, 15
mark_histogram(), *13*
mark_line, 16
mark_link, 17
mark_point, 17
mark_point(), *20*
mark_polygon, 18
mark_ribbon, 19
mark_rule, 19
mark_sina, 20
mark_sina(), *13*
mark_text, 21
mark_violin, 22
mark_violin(), *13*, *20*
marks, 13

print.glyph_layout, 23
print.glyph_spec, 23

quote(), *8*

render, 24
render(), *8*, *23*
renderGlyph *(*glyph-shiny*)*, 8
rendering, 25

scale_color *(*scales*)*, 25
scale_color(), *25*
scale_log *(*scales*)*, 25
scale_log(), *25*
scale_time *(*scales*)*, 25
scale_time(), *25*
scales, 25
selection, 26
selection(), *11*
set_scale, 27
stats::density(), *20*, *22*
stats::quantile(), *14*
summary(), *23*
summary.glyph_layout, 28
summary.glyph_spec, 29

theme_tokens, 29
theming, 30
titles, 31
to_vegalite, 31