Package: vellumplot (via r-universe)

October 4, 2026

Title A Grammar of Graphics on the 'vellum' Backend

Version 0.9.0.9000

Description A declarative, pipe-first grammar of graphics that compiles an inspectable plot specification into a 'vellum' scene and renders it. A plot is built up as a serializable spec (data, layers, scales) and nothing is drawn until it is compiled: encodings are resolved, scales are trained, a panel layout is measured, and guides and marks are emitted as 'vellum' grobs.

License MIT + file LICENSE

URL https://r-vellum.github.io/vellumplot/, https://github.com/r-vellum/vellumplot, https://schochastics.r-universe.dev/vellumplot

BugReports https://github.com/r-vellum/vellumplot/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 8.0.0.9000

Depends R (>= 4.2)

Imports cli, grDevices, farver, rlang, S7, scales, stats, utils, vellum (>= 0.7.0), xml2

Remotes r-vellum/vellum, schochastics/edgebundle, schochastics/graphlayouts

Suggests classInt, edgebundle (>= 1.0.0), gapminder, graphlayouts (>= 1.2.5), gt, igraph, interp, jsonlite, knitr, magick, MASS, mgcv, png, quantreg, sf, testthat (>= 3.0.0), vctrs, withr

Config/testthat/edition 3

Collate 'vellumplot-package.R' 'classes.R' 'print.R' 'alt.R' 'compile-resolve.R' 'coord.R' 'compile-train.R' 'facet.R' 'compile-facet.R' 'marginal.R' 'compile-marks.R' 'elements.R' 'theme-tree.R' 'theme.R' 'compile-layout.R' 'compile-guides.R'

'seam.R' 'animate.R' 'marks.R' 'annotate.R' 'vgraph.R'
'vplot.R' 'sankey.R' 'chord.R' 'clip.R' 'concat.R'
'compile-composition.R' 'compile-marks-basic.R'
'compile-marks-distribution.R' 'compile-marks-graph.R'
'compile-marks-helpers.R' 'compile-marks-interval.R'
'compile-marks-spatial.R' 'compile-marks-structure.R'
'compile-marks-text.R' 'compile-position.R' 'compile-sf.R'
'compile-stat.R' 'decorations.R' 'documents.R' 'effects.R'
'guides.R' 'hierarchy-pack.R' 'hierarchy.R' 'labs.R' 'lint.R'
'spec-serialize.R' 'spec-json.R' 'spec-fields.R' 'mcp.R'
'pattern.R' 'positions.R' 'provenance.R' 'reexports.R'
'repeat.R' 'repel.R' 'scales.R' 'scales-binned.R' 'select.R'
'sparkline.R' 'spec-vegalite.R' 'svg.R' 'table.R' 'venn.R'
'waffle.R' 'zzz.R'

Config/pak/sysreqs libfontconfig1-dev libfreetype6-dev libfribidi-dev libharfbuzz-dev libicu-dev libxml2-dev xz-utils libclang-dev

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

Date/Publication 2026-09-04 07:42:21 UTC

RemoteUrl https://github.com/r-vellum/vellumplot

RemoteRef HEAD

RemoteSha 2b9c15c7d05948b38fd8f8d6210edabf1c112c02

Contents

add_marginal 5
add_selection 6
after_stat 7
anim_save 7
animate 8
annotate 9
area 11
as_spec 11
bind_scale 13
clip 13
compose_annotation 14
concat 15
condition 17
coord_cartesian 18
coord_polar 19
coord_sf 20
coord_trans 21
ease_aes 22
element 23
facet_wrap 25
filter_by 26
glow 26

gradients . . . 27
gt_vsparkline . . . 28
guides . . . 29
inset . . . 32
inspect_source . . . 33
interaction_model . . . 34
labs . . . 35
lims . . . 36
map_decorations . . . 37
mark_abline . . . 38
mark_area . . . 40
mark_boxplot . . . 41
mark_contour . . . 43
mark_count . . . 44
mark_datashade . . . 45
mark_ecdf . . . 46
mark_ellipse . . . 48
mark_graph . . . 49
mark_halfeye . . . 55
mark_histogram . . . 57
mark_image . . . 59
mark_outlier_label . . . 60
mark_pie . . . 61
mark_point . . . 63
mark_raincloud . . . 66
mark_segment . . . 67
mark_series_label . . . 68
mark_sf . . . 69
mark_sf_label . . . 71
mark_signif . . . 72
mark_text . . . 73
mark_text_path . . . 76
mark_tile . . . 77
mark_violin . . . 78
mcp_serve . . . 79
md . . . 80
motion . . . 81
outline . . . 82
pattern_hatch . . . 83
patterns . . . 84
pdf_pages . . . 86
plot_alt . . . 87
plot_data_uri . . . 88
plot_lint . . . 89
plot_manifest . . . 90
plot_provenance . . . 91
plot_spacer . . . 92
plot_svg . . . 92

plot_verify . . . 93
PlotSpec . . . 94
position . . . 96
provenance_join . . . 97
provenance_payload . . . 98
render_all . . . 99
render_plot . . . 100
repeat_ . . . 101
resolve_scale . . . 101
scale_alpha . . . 102
scale_color_continuous . . . 103
scale_color_identity . . . 106
scale_edge . . . 107
scale_edge_width . . . 108
scale_fill_binned . . . 109
scale_linetype . . . 111
scale_linewidth . . . 112
scale_pattern . . . 114
scale_shape . . . 114
scale_size . . . 115
scale_viridis . . . 116
scale_x_binned . . . 119
scale_x_continuous . . . 120
scale_x_date . . . 121
sec_axis . . . 123
select_neighbours . . . 125
select_point . . . 126
shadow . . . 128
sketch . . . 129
spec_diagnose . . . 130
spec_fields . . . 131
spec_schema . . . 131
spec_to_json . . . 132
spec_to_vegalite . . . 133
theme_gray . . . 134
theme_sketch . . . 135
transition_reveal . . . 137
transition_states . . . 137
transition_time . . . 139
vchord . . . 139
vdendrogram . . . 141
vgraph . . . 142
vhierarchy . . . 144
vl_pattern . . . 146
vplot . . . 147
vplot_ask . . . 148
vsankey . . . 149
vsparkline . . . 150

vtable 152
vvenn 153
vwaffle 154

Index **156**

`add_marginal` *Add marginal distributions to a plot*

Description

`add_marginal()` draws a distribution of the panel's `x` variable along the top edge and/or of its `y` variable along the right edge, each sharing the main panel's axis so it lines up with the scatter (the vellumplot analogue of `ggExtra::ggMarginal()`). It is a plot-level modifier, like `facet_wrap()` or `coord_flip()`: it takes no encoding of its own and instead reads `x`, `y` (and, with `group = TRUE`, `color`) from the first layer that maps a numeric `x` and `y` (typically your `mark_point()`).

Usage

```
add_marginal(
  plot,
  type = c("density", "histogram"),
  sides = "tr",
  size = 0.15,
  adjust = 1,
  bins = 30,
  group = FALSE
)
```

Arguments

`plot`	A PlotSpec (from `vplot()`) with at least one x/y layer.
`type`	The marginal distribution: `"density"` (a kernel-density curve, the default) or `"histogram"` (binned counts).
`sides`	Which edges to draw, as a string of `"t"` (top, the x distribution) and/or `"r"` (right, the y distribution). Default `"tr"`.
`size`	The marginal size as a fraction of the panel, in `(0, 1)`.
`adjust`	Bandwidth multiplier for `type = "density"` (see `mark_density()`).
`bins`	Number of bins for `type = "histogram"` (see `mark_histogram()`).
`group`	When `TRUE` and the plot maps `color`/`fill` to a discrete variable, draw one distribution per group in the matching colour (like `ggMarginal(groupColour = TRUE)`). The scatter's legend already covers them, so no extra legend is added.

Details

The marginals are drawn without their own axes or background. This version supports a single panel only: combining it with `facet_wrap()` / `facet_grid()`, a non-Cartesian coordinate system (`coord_flip()`, `coord_polar()`, `coord_fixed()`, `coord_sf()`), or a locked aspect ratio is an error.

Value

The modified PlotSpec.

See Also

`mark_density()`, `mark_histogram()`, `facet_wrap()`

Examples

```
vplot(faithful) |>
  mark_point(x = eruptions, y = waiting) |>
  add_marginal()

vplot(faithful) |>
  mark_point(x = eruptions, y = waiting) |>
  add_marginal(type = "histogram", sides = "t")
```

---

add_selection *Register a free-standing selection on a plot*

Description

Attach a `SelectionSpec` (from the free-standing form of `select_point()` / `select_interval()`) to a plot, marking that the selection's gesture is active on this plot's marks. Use it when a selection is defined once and shared across views in a composition (define + `add_selection()` on one plot, `filter_by()` on another).

Usage

```
add_selection(plot, selection)
```

Arguments

`plot`	A PlotSpec.
`selection`	A `SelectionSpec`.

Value

The PlotSpec, with the selection registered.

See Also

select_point(), filter_by()

---

after_stat *Map an aesthetic to a statistic computed by a stat*

Description

Used inside a mark_*() encoding to reference a variable produced by the layer's statistical transform rather than a raw data column, e.g. y = after_stat(count) or y = after_stat(density).

Usage

```
after_stat(x)
```

Arguments

x	An expression in terms of the stat's computed variables.

Value

Its argument (the marker is interpreted at compile time).

---

anim_save *Write a keyframe animation to a file*

Description

anim_save() renders a vellum_animation (from animate()) to an animated image, tweening the frames between keyframes and encoding them in one parallel, streaming pass in vellum's Rust backend. The format follows the file extension:

Usage

```
anim_save(filename, animation)
```

Arguments

filename	Output path; .gif, .png, or .svg.
animation	A vellum_animation (from animate()).

Details

- `.gif` — a looping GIF (universal compatibility).
- `.png` — an animated PNG (APNG; lossless, alpha).
- `.svg` — a single **animated SVG**: every frame is emitted as vector markup and shown in turn by a CSS step animation. It is resolution-independent (crisp at any size, in print and on retina) and honours `prefers-reduced-motion` (a reader who has asked their system not to animate gets the first frame, held).

Value

`filename`, invisibly.

Choosing a format

Pick by scene complexity, not preference. An animated SVG emits *every frame in full*, so its size grows with the number of marks times the number of frames; a raster format (GIF/APNG) does not. So SVG wins clearly on line art — a few moving marks, the common explanatory case — and loses on a dense scatter, where a raster format is the right answer. `anim_save()` says so when a `.svg` scene is dense enough that a raster format would likely be smaller.

See Also

`animate()`

Examples

```
## Not run:
a <- vplot(mtcars) |>
  mark_point(x = wt, y = mpg) |>
  transition_states(cyl) |>
  animate()
anim_save("cyl.gif", a)
anim_save("cyl.svg", a) # resolution-independent

## End(Not run)
```

`animate` *Build a keyframe animation from a plot*

Description

`animate()` compiles a plot carrying a `transition_states()` into one keyframe scene per state, training the scales once over all states and freezing them, and returns a `vellum_animation`. Write it with `anim_save()`; printing it shows the first keyframe as a preview.

Usage

```
animate(plot, nframes = 50, fps = 25)
```

Arguments

plot	A PlotSpec with a transition_states() (and optionally an ease_aes()).
nframes	Total number of frames to render.
fps	Frames per second (the playback rate written into the file).

Details

Per-state statistics (e.g. a smooth) are recomputed per keyframe, but no scale domain retrains — the frozen scales are the observable proof the animation stayed non-reactive (axis breaks do not move between frames).

Value

A vellum_animation.

See Also

transition_states(), ease_aes(), anim_save()

Examples

```
## Not run:
vplot(mtcars) |>
  mark_point(x = wt, y = mpg) |>
  transition_states(cyl) |>
  animate(nframes = 50, fps = 25)

## End(Not run)
```

---

annotate *Add a one-off annotation*

Description

Draw a single mark (or a short vector of them) from values supplied directly, rather than mapping a data column. The values become a small inline layer data frame, so annotations are independent of the plot data (and repeat on every facet panel). Supported geoms: "text", "label", "point", "segment", "rect", "grob", and "sparkline".

Usage

```
annotate(
  plot,
  geom,
  x = NULL,
  y = NULL,
  xend = NULL,
```

```
  yend = NULL,
  xmin = NULL,
  xmax = NULL,
  ymin = NULL,
  ymax = NULL,
  label = NULL,
  ...
)
```

Arguments

plot	A PlotSpec.
geom	The annotation geometry: one of "text", "label", "point", "segment", "rect", "grob", "sparkline".
x, y	Position (text/label/point/grob/sparkline; segment start).
xend, yend	Segment end.
xmin, xmax, ymin, ymax	Rectangle extent.
label	Text to draw (text/label).
...	Constant aesthetics passed to the mark (e.g. color, fill, alpha, size). For "grob": grob = (a vellum grob or PlotSpec). For "sparkline": values = plus any vsparkline() argument. Both accept width/height/units (the box, default 20 x 6 mm) and halign ("left"/"centre"/"right") / valign ("top"/"centre"/"bottom") to anchor the box relative to (x, y).

Details

"grob" places an arbitrary vellum grob (or a PlotSpec, e.g. a vsparkline()) at data coordinate(s), in a box of physical size — a general "glyph / chart in a panel" seam. "sparkline" is the convenience for the common case: pass values = and it builds and places a vsparkline().

Value

The modified PlotSpec.

Examples

```
vplot(mtcars) |>
  mark_point(x = wt, y = mpg) |>
  annotate("text", x = 4, y = 30, label = "note") |>
  annotate("rect", xmin = 3, xmax = 4, ymin = 15, ymax = 20, alpha = 0.2)
## Not run:
vplot(mtcars) |>
  mark_point(x = wt, y = mpg) |>
  annotate("sparkline", x = 4, y = 30, values = cumsum(rnorm(30)))

## End(Not run)
```

area *Define a layout area for* `design =`

Description

An `area()` is a rectangular block of grid cells (1-based, inclusive) that a sub-plot occupies. Pass a list of `area()`s (one per plot, in order) as `concat(..., design = )` to place plots on an explicit, possibly spanning, grid.

Usage

```
area(t, l, b = t, r = l)
```

Arguments

`t`, `l`, `b`, `r`	Top, left, bottom, right cell indices (1-based, inclusive).

Value

An `area` spec.

Examples

```
a <- vplot(mtcars) |> mark_point(x = wt, y = mpg)
b <- vplot(mtcars) |> mark_point(x = hp, y = mpg)
concat(a, b, design = list(area(1, 1, 1, 2), area(2, 1, 2, 1)))
```

as_spec *Serialize a plot to a plain spec (and back)*

Description

`as_spec()` turns a PlotSpec into a plain, nested, **serializable** R list (data, layers, encodings, scales, coordinates, facets, labels, page size); `from_spec()` rebuilds a `PlotSpec` from that list. Together with `spec_to_json()` / `spec_from_json()` this makes a vellumplot plot a portable *document* — the substrate for the LLM / agent tooling (`spec_fields()`, `vplot_from_spec()`) and the Vega-Lite bridge (`spec_to_vegalite()`).

Usage

```
as_spec(plot)

from_spec(spec, data = NULL, env = globalenv())
```

Arguments

plot	A PlotSpec.
spec	A spec list from as_spec().
data	Optional data frame to render the spec against. Required when the spec stores its data by reference; for a spec that inlines its data (or has no data block), a supplied data takes precedence over the inlined copy.
env	The environment channel expressions are re-quoted in (default the global environment).

Details

The spec is a **subset** of the full grammar — the encoding-level state that a portable document can carry faithfully. State that a JSON document cannot represent (custom transform *functions*, paint/pattern fills, hand-drawn sketch() geometry, secondary axes, sf CRS objects, per-layer data) is **refused** with a classed vellumplot_unserializable error naming the exact slot — it is never silently dropped. The single exception is theme element customisation, which is styling orthogonal to the data spec: the theme *preset* name and its scalar settings round-trip, but custom element_*() overrides are dropped with a warning.

Channel expressions are stored as text (a column name, or an expression like log(x)); the quosure's captured environment is dropped, so an encoding that closes over a local variable will not round-trip.

Data is **inlined** when small (at most a few thousand cells and all-atomic columns) and otherwise stored **by reference** (name + content hash + column schema); a by-reference spec needs from_spec(spec, data =) to recompile.

Value

as_spec() returns a named list; from_spec() returns a PlotSpec.

See Also

spec_to_json(), spec_to_vegalite(), vplot_from_spec()

Examples

```
p <- vplot(mtcars) |> mark_point(x = wt, y = mpg, color = cyl)
spec <- as_spec(p)
spec$layers[[1]]$mark
p2 <- from_spec(spec)
```

bind_scale — *Bind a panel's view to a selection (overview + detail)*

Description

Make the `aes` view of this plot's panel follow an interval selection defined on another (overview) plot: dragging the overview brush pans and zooms this (detail) panel to the selected range. This is a display-tier view change (pan/zoom), not a scale retrain.

Usage

```
bind_scale(plot, selection, aes = c("x", "y"))
```

Arguments

`plot`	A PlotSpec (the detail view).
`selection`	A `SelectionSpec` (an interval selection), or a name string.
`aes`	The axis to bind, `"x"` (default) or `"y"`.

Value

The PlotSpec, with the bind registered.

See Also

`select_interval()`

clip — *Clip or mask a plot to a geometry*

Description

Restrict where a plot's marks show, by a shape rather than the panel rectangle. `clip_to()` is a **hard** clip: only marks inside `region` are drawn (or, with `invert = TRUE`, only those outside) – the way to fill a raster / hexbin / tile heatmap into a country outline (a textured choropleth). `set_mask()` is a **soft** mask: a radial luminance ramp that fades the panel out towards its edges (a vignette / spotlight).

Usage

```
clip_to(plot, region, invert = FALSE)

clip_layer(plot, region, invert = FALSE)

set_mask(plot, region = NULL, type = c("luminance", "alpha"), feather = 0.35)
```

Arguments

plot	A PlotSpec.
region	The clip geometry: an sf object (its polygons), or a data.frame / matrix with x / y columns (and an optional group column for several rings). For set_mask(), NULL (default) fades the whole panel.
invert	For clip_to(), TRUE keeps the marks *outside* region (punching the shape out as a hole) instead of inside.
type	For set_mask(), how the mask's pixels set coverage: "luminance" (default – brightness, white shows / black hides) or "alpha".
feather	For set_mask(), the soft-edge fraction of the radial ramp in [0, 0.9]: 0 a hard disc, larger a gentler fade. Default 0.35.

Details

clip_to() / set_mask() mask the **whole panel**; clip_layer() masks only the **most-recently-added layer** (the one just piped in), so one raster layer can be clipped to a shape while the axes and other layers stay full-bleed.

All three attach to the plot and resolve at render into an isolated masked layer (vellum::as_mask()), so the static output is unchanged when none is set. Cartesian coordinates only (not polar or a non-linear coord_trans()).

A smooth *feathered* edge on an arbitrary polygon needs a blur the renderer does not provide yet, so clip_to() is hard-edged; use set_mask() for a soft (radial) fade.

Value

The modified PlotSpec.

Examples

```
## Not run:
# clip a raster heatmap to a country outline
vplot(grid) |> mark_raster(x = x, y = y, fill = z) |> clip_to(country_sf)

# a vignette
vplot(mtcars) |> mark_point(x = wt, y = mpg) |> set_mask(feather = 0.5)

## End(Not run)
```

---

compose_annotation *Annotate a composition*

Description

Add figure-level text (a title, subtitle, caption) spanning the whole composition, and/or auto-tag the sub-plots (A, B, C, ...). Because every PlotComposition carries its own annotation, this works at any nesting level (unlike patchwork, where annotation is top-level only).

Usage

```
compose_annotation(
  plot,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  tag_levels = NULL,
  tag_prefix = "",
  tag_suffix = ""
)
```

Arguments

`plot`	A `PlotComposition` (from `concat()` / `hconcat()` / `vconcat()`).
`title, subtitle, caption`	Figure-level text (or `NULL`).
`tag_levels`	Auto-tag style: `"A"`, `"a"`, `"1"`, `"i"`, or `"I"` (`NULL` = no tags).
`tag_prefix, tag_suffix`	Strings wrapped around each tag.

Value

The modified `PlotComposition`.

See Also

`theme()`, which also accepts a composition to set the figure-level chrome (title bands, collected legend, panel spacing, tags).

Examples

```
a <- vplot(mtcars) |> mark_point(x = wt, y = mpg)
b <- vplot(mtcars) |> mark_point(x = hp, y = mpg)
hconcat(a, b) |> compose_annotation(title = "Fuel economy", tag_levels = "A")
```

---

concat *Arrange plots side by side*

Description

Compose several independent plots into one image. `hconcat()` lays them in a row, `vconcat()` in a column, and `concat()` on a grid of `ncol`/`nrow` cells. Each sub-plot keeps its own scales, axes, and legend (this is view composition, not faceting).

Usage

```
concat(
  ...,
  ncol = NULL,
  nrow = NULL,
  byrow = TRUE,
  widths = NULL,
  heights = NULL,
  guides = c("collect", "keep"),
  design = NULL,
  width = NULL,
  height = NULL,
  dpi = NULL
)

wrap_plots(
  plots,
  ncol = NULL,
  nrow = NULL,
  byrow = TRUE,
  widths = NULL,
  heights = NULL,
  guides = c("collect", "keep"),
  design = NULL,
  width = NULL,
  height = NULL,
  dpi = NULL
)

hconcat(..., guides = c("collect", "keep"), height = NULL)

vconcat(..., guides = c("collect", "keep"), width = NULL)
```

Arguments

`...`	PlotSpecs or `PlotCompositions` to arrange.
`ncol`, `nrow`	Grid dimensions for `concat()` (defaults to roughly square).
`byrow`	Fill the grid by row (default) or by column.
`widths`, `heights`	Relative panel sizes per column / row (recycled numeric vector), or `NULL` for equal panels.
`guides`	`"collect"` (default) to dedupe identical legends across sub-plots into one, or `"keep"` to leave each sub-plot's legend in place.
`design`	An explicit layout: a list of `area()`s (one per plot, in the order plots are passed) or a textual layout string (rows split by \n, # an empty cell). In a string, distinct letters bind to the plots in `...` in **alphabetical** order (the first letter to the first plot), so there must be exactly one letter per plot and every row must be the same width. Enables spanning; `NULL` (default) uses the regular `ncol`/`nrow` grid.

width, height	Output size in inches (defaults scale with the grid).
dpi	Output resolution in dots per inch. NULL (default) inherits the first sub-plot's resolution; overridable at render time via render_plot().
plots	A list of PlotSpecs / PlotCompositions (for wrap_plots()).

Details

Sub-plot panels are **aligned** across the grid (a shared track grid lines up panel edges even when axis labels differ), and identical legends are **collected** into one shared legend by default (guides = "collect"). Pass PlotCompositions in ... to nest a sub-grid. Annotate the whole figure with compose_annotation().

Value

A PlotComposition (renders via render_plot() / vellum::render()).

Examples

```
a <- vplot(mtcars) |> mark_point(x = wt, y = mpg)
b <- vplot(mtcars) |> mark_histogram(x = mpg, bins = 10)
hconcat(a, b)
```

---

condition *Conditional encoding: style by selection membership*

Description

Use condition() as the value of a mark aesthetic to make that aesthetic depend on whether an element is in a selection: members get if_true, non-members get if_false. This is the general form of highlight-on-interaction.

Usage

```
condition(selection, if_true, if_false, empty = TRUE)
```

Arguments

selection	The selection name (a string) that drives the condition.
if_true	The aesthetic value for selection members — a column expression (data-masked) or a constant.
if_false	The value for non-members — a constant, or omitted to use the theme's dim appearance.
empty	Whether an empty selection matches all elements (TRUE, default).

Details

`condition()` is recognised where it appears in a `mark_*()` encoding; calling it directly is an error. The `if_true` branch is drawn on a static render and trains scales exactly as the equivalent plain encoding, so `color = condition("s", g, "grey80")` produces the same colour scale and legend as `color = g`. With `empty = TRUE` (the default) an empty selection matches everything, so a static / initial render shows `if_true` for all elements and interaction dims non-members to `if_false`.

Value

Used only inside a `mark_*()` encoding; not called directly.

See Also

`select_point()`, `filter_by()`

---

coord_cartesian *Coordinate systems*

Description

`coord_cartesian()` is the default Cartesian system; pass `xlim`/`ylim` to **zoom** the view (out-of-range marks are clipped, not dropped — unlike a `scale_*(limits=)`, which here behaves the same but is the data-scale's job). `coord_flip()` swaps the x and y axes, e.g. for horizontal bars. `coord_fixed()` / `coord_equal()` lock the aspect ratio so one data unit on the y axis occupies `ratio` times the physical length of one unit on x (the panel shrinks to fit and is centred). Coordinate limits take precedence over scale limits.

Usage

```
coord_cartesian(plot, xlim = NULL, ylim = NULL)

coord_flip(plot, xlim = NULL, ylim = NULL)

coord_fixed(plot, ratio = 1, xlim = NULL, ylim = NULL)

coord_equal(plot, ratio = 1, xlim = NULL, ylim = NULL)
```

Arguments

`plot`	A PlotSpec.
`xlim`, `ylim`	Length-2 view-window limits, or `NULL` to use the trained range.
`ratio`	Aspect ratio for `coord_fixed()`: the device length of one y unit relative to one x unit (default 1).

Value

The modified PlotSpec.

Examples

```
vplot(mtcars) |> mark_point(x = wt, y = mpg) |> coord_cartesian(xlim = c(2, 4))
vplot(mtcars) |> mark_bar(x = factor(cyl)) |> coord_flip()
vplot(mtcars) |> mark_point(x = wt, y = mpg) |> coord_fixed()
```

---

coord_polar *Polar coordinates*

Description

`coord_polar()` projects the panel into polar space: one position aesthetic becomes the angle and the other the radius. With `theta = "x"` (the default) the x aesthetic maps to angle and y to radius — a categorical bar chart becomes a wind-rose / coxcomb, a line becomes a radar/spider trace, a point cloud is positioned by `(angle, radius)`. With `theta = "y"` a stacked bar becomes a pie (see also the `mark_pie()` / `mark_donut()` shortcuts). The panel is locked to a square. Lines, areas, and ribbons are interpolated into smooth arcs.

`coord_radial()` is a fuller polar system (ggplot2 3.5's name): besides `theta`/`start`/`direction` it takes `end` to sweep only a **partial arc** (e.g. a half-circle gauge) and `inner_radius` to open a **donut hole**. With `end = NULL` and `inner_radius = 0` it is identical to `coord_polar()`.

Usage

```
coord_polar(plot, theta = "x", start = 0, direction = 1)

coord_radial(
  plot,
  theta = "x",
  start = 0,
  end = NULL,
  direction = 1,
  inner_radius = 0
)
```

Arguments

`plot`	A PlotSpec.
`theta`	Which position aesthetic drives the angle: `"x"` (default) or `"y"`.
`start`	Angular offset of the zero position, in radians (`0` places the first value at twelve o'clock).
`direction`	Winding direction: `1` for clockwise (default), `-1` for counter-clockwise.
`end`	Arc end angle in radians; `NULL` (default) sweeps a full turn from `start`. Set it for a partial arc — e.g. `start = -pi/2, end = pi/2` for a semicircular gauge.
`inner_radius`	Radius of the central hole as a fraction of the outer radius (`0`–`1`); `0` (default) is a filled disc, `> 0` a donut/ring.

Value

The modified PlotSpec.

See Also

mark_pie(), mark_donut()

Examples

```
vplot(mtcars) |> mark_bar(x = factor(cyl)) |> coord_polar(theta = "x")
vplot(mtcars) |>
  mark_bar(x = factor(cyl)) |>
  coord_radial(theta = "x", inner_radius = 0.3)
```

---

coord_sf *Map coordinate system*

Description

coord_sf() is the coordinate system for mark_sf() maps. Before scale training it reprojects every sf layer to a common CRS (via sf::st_transform()), so the renderer only ever sees projected Cartesian coordinates, and it locks the panel aspect ratio so the map is not stretched: 1 for a projected CRS, and the equirectangular correction 1/cos(mean_latitude) for unprojected longitude/latitude data.

Usage

```
coord_sf(
  plot,
  crs = NULL,
  xlim = NULL,
  ylim = NULL,
  graticule = FALSE,
  graticule_labels = TRUE
)
```

Arguments

plot	A PlotSpec.
crs	Target coordinate reference system to project all layers into (anything sf::st_crs() accepts, e.g. an EPSG code, "OGC:CRS84", or a proj/WKT string). NULL (default) uses the CRS of the first sf layer. For guaranteed longitude/latitude order, use "OGC:CRS84" rather than EPSG:4326. For choropleths, prefer an equal-area CRS.
xlim, ylim	Length-2 view-window limits in the target CRS, or NULL.

graticule
: Draw meridians and parallels behind the map. FALSE (default) draws none; TRUE picks round longitude/latitude breaks automatically; a list list(lon = , lat =) sets the breaks explicitly (in degrees). For a projected CRS the lines are reprojected and therefore curved; for unprojected longitude/latitude they are straight.

graticule_labels
: Label the graticule lines with their degree values (default TRUE); ignored when graticule is FALSE.

Details

sf is an optional dependency (in Suggests); coord_sf() errors with an install hint if it is not available at render time.

Value

The modified PlotSpec.

See Also

mark_sf(), mark_scalebar(), mark_compass()

Examples

```
## Not run:
nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
vplot(nc) |> mark_sf(fill = BIR74) |> coord_sf(crs = "OGC:CRS84")
vplot(nc) |> mark_sf(fill = BIR74) |> coord_sf(crs = 3857, graticule = TRUE)

## End(Not run)
```

---

coord_trans *Transformed coordinate system*

Description

coord_trans() applies a nonlinear transform to the **display** of one or both position axes, *after* the scale has trained. This differs from a scale_*(trans=): a scale transform rescales the data and picks its breaks in the transformed space (so a log scale is labelled 1, 10, 100), whereas coord_trans() keeps the trained breaks at their original data values and only warps *where* they are drawn — so the axis is still labelled with the raw values, the gridlines bunch up, and straight lines curve. Use it to show data on, say, a log display without relabelling the axis in powers of ten.

Usage

```
coord_trans(plot, x = "identity", y = "identity")
```

Arguments

plot	A PlotSpec.
x, y	Display transform for that axis (default "identity").

Details

The transform is separable per axis. Each of x/y is a transform name ("identity", "log10", "sqrt") or a scales::transform_*() object. It is intended for continuous position axes; "reverse" is better expressed as scale_*(trans = "reverse").

Value

The modified PlotSpec.

Examples

```
vplot(mtcars) |> mark_point(x = wt, y = mpg) |> coord_trans(y = "log10")
```

---

ease_aes *Set the easing of an animation's frames*

Description

ease_aes() chooses the easing function that shapes how the interpolation fraction moves through each transition — e.g. "cubic-in-out" starts and ends gently. By default it applies to every interpolated aesthetic; the position/color/size/alpha arguments override it for one class each, so marks can glide into place on a cubic curve while their colour crossfades evenly.

Usage

```
ease_aes(
  plot,
  ease = "linear",
  position = NULL,
  color = NULL,
  size = NULL,
  alpha = NULL,
  colour = NULL
)
```

Arguments

plot	A PlotSpec carrying a transition_states().
ease	An easing name: "linear", or a family (quad/cubic/quart/quint/sine/expo/circ/back/elastic/bounc with a direction suffix (-in, -out, -in-out), e.g. "cubic-in-out". Used for any class not given its own easing below.

`position`, `color`, `size`, `alpha`
: Optional per-class easings, named the same way as `ease`. `NULL` (the default) means "use `ease`". Each covers one class of drawn property:

  - `position` — x and y, and every other coordinate-space quantity (bar widths, angles, path vertices, text rotation). It also decides *when* discrete attributes flip, since those snap at the halfway point.
  - `color` — `color` and `fill`, including per-element fills.
  - `size` — `size`, `linewidth`, and radii.
  - `alpha` — opacity, **and the fade of marks entering or leaving** a state.

  `position` covers x and y together: the engine has a single positional curve, so the two axes cannot be eased apart.

`colour`
: An alias for `color`; supplying both is an error.

Value

The modified PlotSpec.

See Also

`transition_states()`, `animate()`

Examples

```
vplot(mtcars) |>
  mark_point(x = wt, y = mpg) |>
  transition_states(cyl) |>
  ease_aes("cubic-in-out")

# Marks settle into place gently, but the colour crossfade stays even.
vplot(mtcars) |>
  mark_point(x = wt, y = mpg, color = factor(gear)) |>
  transition_states(cyl) |>
  ease_aes("cubic-in-out", color = "linear")
```

`element` *Theme elements*

Description

Typed building blocks for `theme()`. Each describes how a family of theme slots is drawn; any property left `NULL` is inherited from the slot's parent in the theme tree. `element_blank()` draws nothing.

Usage

```
element_text(
  family = NULL,
  face = NULL,
  colour = NULL,
  color = NULL,
  size = NULL,
  cex = NULL,
  hjust = NULL,
  vjust = NULL,
  angle = NULL,
  lineheight = NULL,
  margin = NULL
)

element_line(
  colour = NULL,
  color = NULL,
  linewidth = NULL,
  linetype = NULL,
  lineend = NULL,
  sketch = NULL
)

element_rect(
  fill = NULL,
  colour = NULL,
  color = NULL,
  linewidth = NULL,
  linetype = NULL,
  sketch = NULL
)

element_blank()
```

Arguments

`family`, `face`, `size`, `cex`, `colour`, `color`, `hjust`, `vjust`, `angle`, `lineheight`, `margin`	Text properties. `color` is an alias for `colour`; `size` is in points and `cex` is a relative multiplier on the inherited `size`; `margin` is a numeric vector of millimetres (recycled to length 4).
`linewidth`, `linetype`, `lineend`	Line properties.
`sketch`	For `element_line()` / `element_rect()`, a `sketch()` spec giving that theme element (gridlines, axis lines, ticks, panel/strip/legend backgrounds) a hand-drawn look, `NA` to force it crisp, or `NULL` (default) to inherit from its parent element / the plot-wide `theme_sketch()` default. Text elements are never sketched.

`fill`	Fill colour (rectangles).

Value

An element object for use in `theme()`.

Examples

```
vplot(mtcars) |>
  mark_point(x = wt, y = mpg) |>
  theme(plot.title = element_text(size = 16), panel.grid.minor = element_blank())
```

---

facet_wrap *Facet a plot into a grid of panels*

Description

`facet_wrap()` lays panels out in a ribbon wrapped into `ncol`/`nrow`; `facet_grid()` arranges them on a 2-D grid defined by a `rows ~ cols` formula. Position scales are shared across panels by default (so axes align); pass `scales = "free_x"`, `"free_y"`, or `"free"` to train them per panel. Colour and size scales (and their legends) are always shared in this version.

Usage

```
facet_wrap(
  plot,
  facets,
  ncol = NULL,
  nrow = NULL,
  scales = c("fixed", "free_x", "free_y", "free")
)

facet_grid(plot, facets, scales = c("fixed", "free_x", "free_y", "free"))
```

Arguments

`plot`	A PlotSpec.
`facets`	A faceting formula. For `facet_wrap()`, one-sided such as `~ cyl` (one or more `+`-separated variables); for `facet_grid()`, a two-sided `rows ~ cols` (use `.` for no faceting on a side).
`ncol, nrow`	Number of columns/rows for `facet_wrap()`.
`scales`	One of `"fixed"` (default), `"free_x"`, `"free_y"`, `"free"`.

Value

The modified PlotSpec.

Examples

```
vplot(mtcars) |> mark_point(x = wt, y = mpg) |> facet_wrap(~cyl)
vplot(mtcars) |> mark_point(x = wt, y = mpg) |> facet_grid(am ~ cyl)
```

`filter_by` *Filter a view by a selection*

Description

Show only the rows in `selection`, hiding the rest (a display-tier hide, not a re-aggregation). Point a second view's `filter_by()` at a selection whose gesture lives in a first view and you get **cross-filtering**: brush one panel, a linked panel redraws to just those rows.

Usage

```
filter_by(plot, selection)
```

Arguments

`plot`	A PlotSpec.
`selection`	A `SelectionSpec`, or a selection name string.

Value

The PlotSpec, with the filter registered.

See Also

`select_interval()`, `add_selection()`

`glow` *Neon glow layer effect*

Description

A render effect for stroked, point, and text marks, in the spirit of mplcyberpunk: a widened copy of the mark is drawn beneath the crisp original and softened with a **real Gaussian blur** (`vellum`'s `vl_viewport(blur=)`), composited additively (a `"screen"` blend) into a soft neon halo. Pass it to a mark's `effects` argument, e.g. `mark_line(..., effects = list(glow()))`. Pairs naturally with `theme_cyberpunk()`.

Usage

```
glow(size = 6, layers = 6L, alpha = 0.12, blend = "screen", color = NULL)
```

Arguments

size	Halo spread in millimetres: the blur radius (and, for stroked/point marks, how much the copy is widened before blurring).
layers, blend	Retained for compatibility and the neon look: layers scales the halo opacity (it no longer stacks copies – one blurred layer replaces them), and blend (typically "screen"/"lighten") composites the halo.
alpha	Base halo opacity (scaled by layers).
color	Halo colour, or NULL (default) to inherit the mark's own resolved colour — the usual neon look.

Details

The glow is applied per style group, so a colour-mapped multi-series line glows each series in its own hue. It applies to mark_point(), mark_line(), mark_step(), mark_rule(), mark_segment(), mark_edges(), mark_nodes(), mark_edge_bundle(), mark_flow_map(), and text marks (mark_text() / mark_label()); other marks reject it.

Value

A GlowSpec object for a mark's effects list.

See Also

outline(), shadow(), theme_cyberpunk()

Examples

```
df <- data.frame(x = 1:20, y = cumsum(rnorm(20)))
vplot(df) |>
  mark_line(x = x, y = y, color = "#00e5ff", effects = list(glow())) |>
  theme_cyberpunk()
```

---

gradients *Gradient fill paints*

Description

Thin re-exports of vellum::linear_gradient() and vellum::radial_gradient(). A gradient is an unscaled *value* for the fill aesthetic: pass it directly, e.g. mark_area(x = t, y = y, fill = linear_gradient(c("#00e5ff", "#00e5ff00"))), and the filled region (area / ribbon / bar) is painted with the paint as a single grob. Use "transparent" (or an "#RRGGBB00" colour) as a stop to fade out — the "glow fade under a line" look. The gradient's x1/y1/x2/y2 (in units, "npc" by default) set its direction. A gradient cannot be *mapped* to a data column (it is one paint per region).

Usage

```
linear_gradient(colours, stops = NULL, ...)

radial_gradient(colours, stops = NULL, ...)
```

Arguments

colours, stops	See vellum::linear_gradient() / vellum::radial_gradient().
...	Further gradient arguments passed to the vellum constructor: x1/y1/x2/y2 (linear), cx/cy/r plus the optional focal circle fx/fy/fr (radial), units, extend, and interpolation ("srgb" default, or "oklab" / "oklch" to blend the stops perceptually — "oklch" additionally preserves chroma by rotating hue). See vellum::linear_gradient() / vellum::radial_gradient().

Value

A vellum_gradient object usable as a fill value.

See Also

glow(), theme_cyberpunk()

Examples

```
df <- data.frame(x = 1:20, y = cumsum(abs(rnorm(20))))
vplot(df) |>
  mark_area(x = x, y = y, fill = linear_gradient(c("#00e5ff", "#00e5ff00"),
    x1 = 0, y1 = 1, x2 = 0, y2 = 0))
```

---

gt_vsparkline *Add a vellumplot sparkline column to a* gt *table*

Description

gt_vsparkline() renders a **list-column of numeric vectors** as a per-row vsparkline(), embedded as inline SVG in a gt::gt() table — so a gt table (with all of gt's formatting and theming) gains a vellum sparkline column. Needs the **gt** package, and HTML output (inline SVG is not embedded by gt's LaTeX / Word backends).

Usage

```
gt_vsparkline(
  gt_object,
  column,
  type = "line",
  ...,
  width = 30,
```

```
  height = 8,
  units = "mm"
)
```

Arguments

gt_object	A gt::gt() object.
column	The list-column to render (bare name or string).
type	Sparkline type: "line" (default), "bar", or "winloss".
...	Further vsparkline() arguments (e.g. color, points).
width, height, units	Sparkline size (default 30 x 8 mm).

Details

The vectors are recovered from the gt object's data in its current row order; apply gt_vsparkline() **before** any gt row reordering / grouping.

Value

The modified gt object.

See Also

vsparkline(), vtable(), plot_svg()

Examples

```
## Not run:
df <- data.frame(metric = c("A", "B"))
df$trend <- list(cumsum(rnorm(20)), cumsum(rnorm(20)))
gt::gt(df) |> gt_vsparkline(trend, type = "line")

## End(Not run)
```

---

guides *Control a scale's legend*

Description

guides() overrides the legend (guide) for one or more aesthetics without respelling the whole scale_*(). Pass "none" (or guide_none()) to hide a legend, guide_legend() to tweak a keyed legend (reverse the key order, override the title, restyle the keys with override.aes), guide_colourbar() to size, tick and style a continuous colour bar, or guide_coloursteps() to draw a **binned** colour scale as a segmented bar instead of swatches. Applies to the non-position legends (color/fill, size, shape, alpha, linetype); position axes are unaffected.

Usage

```
guides(plot, ...)

guide_none()

guide_legend(
  title = NULL,
  reverse = FALSE,
  override.aes = NULL,
  nested = FALSE
)

guide_colourbar(
  title = NULL,
  barwidth = NULL,
  barheight = NULL,
  ticks = TRUE,
  ticks.colour = "white",
  label.position = NULL,
  n.breaks = NULL,
  reverse = FALSE
)

guide_colorbar(
  title = NULL,
  barwidth = NULL,
  barheight = NULL,
  ticks = TRUE,
  ticks.colour = "white",
  label.position = NULL,
  n.breaks = NULL,
  reverse = FALSE
)

guide_coloursteps(
  title = NULL,
  barwidth = NULL,
  barheight = NULL,
  ticks = FALSE,
  ticks.colour = "white",
  label.position = NULL,
  reverse = FALSE
)

guide_colorsteps(
  title = NULL,
  barwidth = NULL,
  barheight = NULL,
```

```
  ticks = FALSE,
  ticks.colour = "white",
  label.position = NULL,
  reverse = FALSE
)
```

Arguments

plot	A PlotSpec.
...	Named by aesthetic, e.g. guides(color = "none", shape = guide_legend(reverse = TRUE)).
title	An axis/legend title override, or NULL to keep the default.
reverse	Reverse the order of the legend keys / the colour bar.
override.aes	A named list of aesthetics to force on the legend **keys**, independent of the plotted data — the classic "make faint, small points legible in the key" fix. Recognised names: size (mm), alpha, colour/color, fill, shape, and linewidth. For example override.aes = list(size = 5, alpha = 1) draws big, opaque keys over a scatter of tiny translucent points. NULL (default) leaves the keys as drawn from the data.
nested	For a **size** legend, draw the keys as concentric, bottom-aligned circles (a proportional-symbol / "bubble" legend) with a leader from each circle to its label, instead of stacked rows. Best with a wide size range (e.g. scale_size(range = c(2, 12))) so the circles are large enough to read; ignored by non-size legends. Drawn for a vertical legend (a horizontal one keeps the stacked keys).
barwidth, barheight	The colour bar's own width and height, in millimetres (guide_colourbar()). barheight sets the bar's **length** on the default vertical bar (and its thickness on a horizontal one); barwidth its thickness (and the bar length on a horizontal legend). NULL auto-sizes.
ticks	Draw the break ticks on the colour bar? (Default TRUE for guide_colourbar(), FALSE for the segmented guide_coloursteps().)
ticks.colour	Colour of the break ticks (default "white").
label.position	Which side of a **vertical** colour bar the labels sit, "right" (default) or "left".
n.breaks	Roughly how many ticks (and labels) to put on a **continuous** colour bar — a target, not a promise: the break algorithm prefers round numbers over the bar's range and returns whatever count reads best near the one asked for, so n.breaks = 4 may draw 3 or 5 rather than tick a value like 23.33. NULL (default) keeps the automatic count. An explicit breaks = on the scale names the values that get a tick and outranks this; so does labels =, which is paired with those breaks. There is deliberately no nbin argument (ggplot2's band count for the gradient). The bar is drawn as one real gradient fill, not a stack of rectangles approximating one, so it has no bands to count; the segmented look is guide_coloursteps() on a binned scale, where the segments are the scale's own bins.

Value

guides(): the modified PlotSpec. guide_none() / guide_legend(): a guide specification for use inside guides().

Examples

```
vplot(mtcars) |>
  mark_point(x = wt, y = mpg, color = factor(cyl)) |>
  guides(color = "none")

vplot(mtcars) |>
  mark_point(x = wt, y = mpg, color = factor(cyl)) |>
  guides(color = guide_legend(reverse = TRUE))

# legible keys over faint, tiny points
vplot(mtcars) |>
  mark_point(x = wt, y = mpg, color = factor(cyl), alpha = 0.15, size = 0.6) |>
  guides(color = guide_legend(override.aes = list(size = 5, alpha = 1)))

# a taller, wider colour bar with left-side labels and no ticks
vplot(mtcars) |>
  mark_point(x = wt, y = mpg, color = hp) |>
  guides(color = guide_colourbar(
    barwidth = 8, barheight = 60, ticks = FALSE, label.position = "left"
  ))

# about four ticks on the bar instead of the automatic count
vplot(mtcars) |>
  mark_point(x = wt, y = mpg, color = hp) |>
  guides(color = guide_colourbar(n.breaks = 4))

# a binned colour scale as a segmented bar
vplot(mtcars) |>
  mark_point(x = wt, y = mpg, color = hp) |>
  scale_color_binned() |>
  guides(color = guide_coloursteps())

# a proportional-symbol (nested-circle) size legend
vplot(mtcars) |>
  mark_point(x = wt, y = mpg, size = hp) |>
  scale_size(range = c(2, 12)) |>
  guides(size = guide_legend(nested = TRUE))
```

---

inset *Overlay a plot as an inset*

Description

Place plot as an inset over base, positioned by fractional coordinates (0–1) of the chosen reference box. Returns a 1-cell PlotComposition carrying the inset (compose it further or render directly).

Usage

```
inset(
  base,
  plot,
  left = 0.6,
  bottom = 0.6,
  right = 0.98,
  top = 0.98,
  align_to = c("panel", "plot", "full"),
  on_top = TRUE
)
```

Arguments

base	A PlotSpec or PlotComposition to draw underneath.
plot	The PlotSpec to overlay.
left, bottom, right, top	Inset position as fractions (0–1).
align_to	Reference box: "panel", "plot", or "full" (currently the whole base cell).
on_top	Draw the inset above (TRUE) or below the base.

Value

A PlotComposition.

Examples

```
a <- vplot(mtcars) |> mark_point(x = wt, y = mpg)
b <- vplot(mtcars) |> mark_histogram(x = mpg, bins = 8)
inset(a, b, left = 0.55, bottom = 0.55, right = 0.98, top = 0.98)
```

---

inspect_source *Inspect the data behind a clicked element (click-to-source)*

Description

Declare that a host should surface the **source data rows** behind an element when the user clicks (or hovers) it — the "click-to-source" affordance. It is inert on a static render; a host (`vellumwidget::as_widget()`) reads the compiled scene's provenance (`provenance_join()` / `provenance_payload()`) to answer the gesture, and under Shiny reports the clicked element's rows to `input$<id>_source`.

Usage

```
inspect_source(plot, on = c("click", "hover"), values = FALSE)
```

Arguments

plot	A PlotSpec.
on	The gesture that reveals the source, "click" (default) or "hover".
values	If TRUE, also ship the referenced data rows so the host can display their values (a heavier payload); FALSE (default) ships only row indices.

Details

This is opt-in because the row mapping adds to the widget payload; declaring it on the plot (rather than as a host flag) keeps interactivity a property of the spec.

Value

The PlotSpec, with source inspection registered.

See Also

provenance_payload(), select_point()

Examples

```
vplot(mtcars) |>
  mark_point(x = wt, y = mpg) |>
  inspect_source()
```

---

interaction_model *The interaction model of a compiled plot*

Description

Returns the serialisable declaration block a host needs to enact a plot's declarative interactivity (selections, conditional encodings, filters, and scale-domain binds), or NULL when the plot declares none. This is the plot-level companion to the per-element metadata carried on the compiled scene; vellumwidget::as_widget() reads it to wire gestures to reactions.

Usage

```
interaction_model(x)
```

Arguments

x	A PlotSpec or a plot composition.

Value

A named list (selections, conditions, filters, binds, and source when inspect_source() is declared), or NULL if there is no declared interactivity.

`labs` *Set plot titles and axis/legend labels*

Description

Add plot-level text — a `title`, `subtitle`, `caption`, and `tag` — and override the titles of individual axes and legends (`x`, `y`, `color`, `size`). The title, subtitle, and tag are drawn in a band above the panels; the caption in a band below. Repeated `labs()` calls merge, with later values winning.

Usage

```
labs(
  plot,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  tag = NULL,
  x = NULL,
  y = NULL,
  color = NULL,
  colour = NULL,
  fill = NULL,
  size = NULL,
  alt = NULL,
  ...
)
```

Arguments

`plot`	A PlotSpec.
`title`, `subtitle`, `caption`, `tag`	Plot-level text (or `NULL` to leave unset).
`x`, `y`, `size`	Axis / legend title overrides for those aesthetics.
`color`, `colour`, `fill`	Colour-scale title override; `colour` and `fill` are aliases for `color`.
`alt`	A text alternative (alt text) describing the plot for screen readers and other assistive technology. Overrides the description vellumplot generates automatically; see `plot_alt()`. Emitted into the accessible SVG (`<desc>`) and tagged PDF (Figure `Alt`) by `render_plot()`.
`...`	Reserved; passing anything here is an error.

Details

Axis/legend overrides set here are used unless a matching `scale_*(name = )` is also given, which takes precedence. With neither, the title is derived from the mapping.

Value

The modified PlotSpec.

Examples

```
vplot(mtcars) |>
  mark_point(x = wt, y = mpg) |>
  labs(title = "Fuel economy", x = "Weight", y = "MPG")
```

---

lims *Set scale limits with a shortcut*

Description

Convenience wrappers that declare a scale carrying only its limits, instead of spelling out a full `scale_*()`. `xlim()` / `ylim()` set the position range; `lims()` sets limits for any named aesthetic. They are shortcuts for `scale_*(limits = ...)`: like there, an explicit range sets the trained (view) window and marks outside it are clipped.

Usage

```
lims(plot, ...)

xlim(plot, ...)

ylim(plot, ...)
```

Arguments

`plot`	A PlotSpec.
`...`	For `xlim()`/`ylim()`, the limits: two numbers `xlim(0, 10)` (or a length-2 vector) for a continuous axis, or a set of levels `xlim("a", "b", "c")` for a discrete one. For `lims()`, named limit vectors, one per aesthetic, e.g. `lims(x = c(0, 10), color = c(0, 100))`.

Value

The modified PlotSpec.

See Also

`scale_x_continuous()`

Examples

```
vplot(mtcars) |> mark_point(x = wt, y = mpg) |> xlim(0, 6)
vplot(mtcars) |> mark_point(x = wt, y = mpg) |> lims(x = c(0, 6), y = c(10, 35))
```

map_decorations *Map decorations*

Description

Cartographic furniture for coord_sf() maps. mark_scalebar() draws a segmented distance bar; mark_compass() draws a north arrow. Both are fixed-position decorations pinned to a panel corner (they take no data aesthetics). Graticule lines (meridians and parallels) are added through coord_sf()'s graticule argument, not a mark, so they render behind the map.

Usage

```
mark_scalebar(
  plot,
  distance = NULL,
  unit = "km",
  position = "bottomleft",
  segments = 4L,
  height = 2.5,
  pad = 4,
  text_size = 7,
  color = "black",
  fill = "white",
  data = NULL
)

mark_compass(
  plot,
  position = "topright",
  size = 10,
  pad = 4,
  rotation = 0,
  color = "black",
  fill = "black",
  text = TRUE,
  data = NULL
)
```

Arguments

plot	A PlotSpec.
distance	Bar length in unit. NULL (default) picks a round number near a quarter of the panel width.
unit	Distance unit for the label and distance: "km" (default), "m", "mi", or "ft".
position	Panel corner to anchor the decoration: "bottomleft" (scale bar default), "topright" (compass default), "bottomright", or "topleft" (the abbreviations "bl"/"br"/"tl"/"tr" also work).

segments	Number of alternating segments in the bar (default 4).
height	Bar thickness, in millimetres.
pad	Inset from the panel edge, in millimetres.
text_size	Label font size (points).
color, fill	Stroke and fill colours. For the scale bar the segments alternate color and fill; for the compass the arrow is fill with a color outline.
data	Optional layer data (rarely needed – decorations are not data-driven).
size	Glyph height, in millimetres.
rotation	Extra clockwise rotation of the arrow, in degrees (e.g. to point at grid north on a rotated layout).
text	Draw the "N" label above the arrow (default TRUE).

Value

The modified PlotSpec.

See Also

coord_sf(), mark_sf()

Examples

```
## Not run:
nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
vplot(nc) |>
  mark_sf(fill = BIR74) |>
  coord_sf(crs = 3857, graticule = TRUE) |>
  mark_scalebar() |>
  mark_compass()

## End(Not run)
```

---

mark_abline *Reference lines and function curves*

Description

mark_abline() draws a sloped reference line y = slope * x + intercept across the panel (the diagonal cousin of mark_rule()'s horizontal/vertical lines). mark_function() draws a curve y = fun(x) sampled over the panel's x range — e.g. a theoretical density over a histogram, or a reference curve over a scatter. Both take no data encodings; they reuse the panel's existing x/y scales, so add them on top of a data layer. Both assume linear axes.

Usage

```
mark_abline(
  plot,
  ...,
  slope = 1,
  intercept = 0,
  blend = NULL,
  sketch = NULL,
  data = NULL
)

mark_function(
  plot,
  ...,
  fun,
  n = 101,
  args = list(),
  blend = NULL,
  sketch = NULL,
  data = NULL
)
```

Arguments

plot	A PlotSpec.
...	For mark_abline()/mark_function(), styling passed through as constants (e.g. color, linewidth, linetype).
slope, intercept	For mark_abline(), the line's slope and intercept (each may be a vector to draw several lines). Defaults 1 and 0.
blend, sketch, data	See mark_point().
fun	For mark_function(), a function (or its name) mapping a numeric x vector to y, e.g. dnorm or \(x) x^2.
n	For mark_function(), the number of points sampled across the x range. Default 101.
args	For mark_function(), extra arguments passed to fun (e.g. args = list(mean = 5, sd = 2)).

Value

The modified PlotSpec.

Examples

```
vplot(mtcars) |> mark_point(x = wt, y = mpg) |> mark_abline(slope = -5, intercept = 37)
vplot(data.frame(x = rnorm(200))) |>
```

```
mark_histogram(x = x, y = after_stat(density)) |>
mark_function(fun = dnorm)
```

---

mark_area *Area, ribbon, and step marks*

Description

`mark_area()` fills the region between a `y` line and the zero baseline; `mark_ribbon()` fills between `ymin` and `ymax`; `mark_step()` draws a staircase line. All connect points in `x` order.

Usage

```
mark_area(
  plot,
  ...,
  position = "stack",
  blend = NULL,
  sketch = NULL,
  data = NULL
)

mark_ribbon(plot, ..., blend = NULL, sketch = NULL, data = NULL)

mark_step(
  plot,
  ...,
  direction = "hv",
  auto = FALSE,
  blend = NULL,
  effects = list(),
  sketch = NULL,
  data = NULL
)
```

Arguments

`plot`	A PlotSpec (from `vplot()`).
`...`	Encodings (tidy-eval): `x` and `y` for area/step; `x`, `ymin`, `ymax` for ribbon; plus `color`/`fill`/`alpha`. `mark_step()` also takes `linewidth` and `linetype`; a mapped `linewidth` varies the staircase's width per tread (see `scale_linewidth()`).
`position`	For `mark_area()`, how areas sharing an `x` combine when `fill`/`color` is mapped: `"stack"` (default) stacks them into a band, `"fill"` normalises each `x` to 1, `"identity"` overlays them from the zero baseline. With no fill mapping all three are equivalent (a single area).

blend	Optional blend mode for compositing this layer against what is already drawn beneath it (the panel and earlier layers), one of the CSS mix-blend-mode names, e.g. "multiply", "screen", "darken". The whole layer composites as one isolated group (not per element).
sketch	A sketch() spec giving this layer a hand-drawn look (wobbly outlines, hachure fills), NA/FALSE to force it crisp (overriding a plot-wide theme_sketch()), or NULL (default) to inherit. Geometry marks accept it; text, raster, hex and datashade marks do not.
data	Optional layer data frame; overrides the plot data for this layer.
direction	For mark_step(), "hv" (horizontal then vertical, default) or "vh".
auto	For mark_point(), mark_line(), mark_step(), mark_segment(), and mark_edges(), when TRUE and the layer has very many rows, automatically render it as a datashaded density raster (see mark_datashade()) instead of individual vector marks: points bin into a density grid (vellum::datashade()), dense lines/steps rasterise as connected polylines (vellum::datashade_lines()), and segments/edges as independent segments (vellum::datashade_segments()). The datashaded line/segment output is dynspread-ed so thin marks stay visible (see the spread argument of mark_datashade()). The fallback is skipped under a warped coordinate system (coord_polar() / coord_trans()), which draws the vector marks instead.
effects	A list of layer render effects applied to the mark at draw time — glow(), outline(), and shadow(). Available on stroked and point marks.

Value

The modified PlotSpec.

See Also

scale_linewidth()

Examples

```
vplot(pressure) |> mark_area(x = temperature, y = pressure)
```

---

mark_boxplot *Boxplot, error bar, and summary marks*

Description

mark_boxplot() draws a box-and-whisker per x category from the raw y values (box = Q1-Q3, median line, 1.5*IQR whiskers, outlier points). mark_errorbar() draws vertical bars from ymin to ymax with horizontal caps; mark_linerange() omits the caps. mark_summary() aggregates y per x with fun (default mean) and draws the result as points.

Usage

```
mark_boxplot(plot, ..., blend = NULL, sketch = NULL, data = NULL)

mark_errorbar(plot, ..., width = 0.5, blend = NULL, sketch = NULL, data = NULL)

mark_linerange(plot, ..., blend = NULL, sketch = NULL, data = NULL)

mark_summary(plot, ..., fun = mean, blend = NULL, sketch = NULL, data = NULL)

mark_pointrange(plot, ..., blend = NULL, sketch = NULL, data = NULL)

mark_crossbar(plot, ..., width = 0.5, blend = NULL, sketch = NULL, data = NULL)
```

Arguments

plot	A PlotSpec (from vplot()).
...	Encodings (tidy-eval): x, y for boxplot/summary; x, ymin, ymax for errorbar/linerange; plus color/fill (and linetype for errorbar/linerange).
blend	Optional blend mode for compositing this layer against what is already drawn beneath it (the panel and earlier layers), one of the CSS mix-blend-mode names, e.g. "multiply", "screen", "darken". The whole layer composites as one isolated group (not per element).
sketch	A sketch() spec giving this layer a hand-drawn look (wobbly outlines, hachure fills), NA/FALSE to force it crisp (overriding a plot-wide theme_sketch()), or NULL (default) to inherit. Geometry marks accept it; text, raster, hex and datashade marks do not.
data	Optional layer data frame; overrides the plot data for this layer.
width	For mark_errorbar(), the cap width as a fraction of the band.
fun	For mark_summary(), the aggregation function (default mean).

Details

mark_pointrange() draws a point at (x, y) with a vertical line from ymin to ymax (an identity mark — supply the summary values yourself, e.g. from a model). mark_crossbar() draws a box spanning ymin..ymax with a horizontal line at y. Both are the identity counterparts of the aggregating mark_interval().

Value

The modified PlotSpec.

Examples

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

mark_contour *2-D density contours*

Description

mark_contour() draws iso-density contour **lines** of a 2-D point cloud (x, y); mark_contour_filled() fills the bands between them. By default the field is a kernel density estimate (needs the **MASS** package); map a z aesthetic to instead contour a supplied surface over a regular x/y grid. Contours are coloured by level automatically — mark_contour() maps color = after_stat(level), mark_contour_filled() maps fill. Contour tracing is done by the engine (vellum::vl_contour()), so no extra package is needed for the lines; a density field additionally needs **MASS**.

Usage

```
mark_contour(
  plot,
  ...,
  bins = 10,
  binwidth = NULL,
  breaks = NULL,
  n = 100,
  blend = NULL,
  sketch = NULL,
  data = NULL
)

mark_contour_filled(
  plot,
  ...,
  bins = 10,
  binwidth = NULL,
  breaks = NULL,
  n = 100,
  blend = NULL,
  sketch = NULL,
  data = NULL
)
```

Arguments

plot	A PlotSpec (from vplot()).
...	Encodings (tidy-eval): x, y (+ optional z surface, color / fill, linewidth, linetype).
bins	Target number of contour levels (when neither breaks nor binwidth is given).
binwidth	Spacing between contour levels, or NULL.
breaks	Explicit contour levels, or NULL to derive from bins / binwidth.

n	Density-estimate grid resolution per axis (KDE mode only).
blend	Optional blend mode for compositing this layer against what is already drawn beneath it (the panel and earlier layers), one of the CSS mix-blend-mode names, e.g. "multiply", "screen", "darken". The whole layer composites as one isolated group (not per element).
sketch	A sketch() spec giving this layer a hand-drawn look (wobbly outlines, hachure fills), NA/FALSE to force it crisp (overriding a plot-wide theme_sketch()), or NULL (default) to inherit. Geometry marks accept it; text, raster, hex and datashade marks do not.
data	Optional layer data frame; overrides the plot data for this layer.

Value

The modified PlotSpec.

Examples

```
vplot(faithful) |>
  mark_point(x = eruptions, y = waiting) |>
  mark_contour(x = eruptions, y = waiting)
vplot(faithful) |> mark_contour_filled(x = eruptions, y = waiting)
```

---

mark_count *Count overlapping points*

Description

mark_count() collapses coincident (x, y) observations to one point sized by how many fall there (ggplot2's geom_count() / stat_sum()) — the honest way to show overplotting on a discrete or rounded grid. size defaults to after_stat(n) (the overlap count); map it to after_stat(prop) for the proportion instead.

Usage

```
mark_count(plot, ..., blend = NULL, sketch = NULL, data = NULL)
```

Arguments

plot	A PlotSpec (from vplot()).
...	Encodings (tidy-eval): x, y (+ color/fill to count within groups, and size to override the default after_stat(n)).
blend	Optional blend mode for compositing this layer against what is already drawn beneath it (the panel and earlier layers), one of the CSS mix-blend-mode names, e.g. "multiply", "screen", "darken". The whole layer composites as one isolated group (not per element).

sketch	A sketch() spec giving this layer a hand-drawn look (wobbly outlines, hachure fills), NA/FALSE to force it crisp (overriding a plot-wide theme_sketch()), or NULL (default) to inherit. Geometry marks accept it; text, raster, hex and datashade marks do not.
data	Optional layer data frame; overrides the plot data for this layer.

Value

The modified PlotSpec.

Examples

```
vplot(mpg <- data.frame(cyl = mtcars$cyl, gear = mtcars$gear)) |>
  mark_count(x = cyl, y = gear)
```

---

mark_datashade *Datashade a large point cloud*

Description

For data too dense to draw one marker each (overplotted, up to millions of points), mark_datashade() bins the points into a canvas-sized grid in one pass and colours each cell by density (via vellum::datashade()), drawing a single raster that fills the panel. Cost is decoupled from point count and overplotting.

Usage

```
mark_datashade(
  plot,
  ...,
  width = 400,
  height = 300,
  colors = NULL,
  how = "eq_hist",
  span = NULL,
  clip = NULL,
  spread = NULL,
  blend = NULL,
  data = NULL
)
```

Arguments

plot	A PlotSpec (from vplot()).
...	Encodings; x and y are required. Map color/fill to a discrete variable for categorical (count_cat) shading.
width, height	Aggregation grid size in cells (output raster pixels).

colors	Two or more colours forming the low-to-high density ramp (plain density shading only; ignored when a color/fill aesthetic is mapped).
how	Density-to-colour mapping (and, categorically, per-cell opacity): "eq_hist" (default), "log", "cbrt", or "linear".
span, clip	Optional density-range clamping passed to vellum::datashade(): span = c(lo, hi) absolute limits, or clip = c(0.01, 0.99) percentiles, so a few extreme cells don't flatten the rest. Both default NULL.
spread	Optional post-aggregation spreading to keep sparse output visible (passed to vellum::datashade()): NULL (default) none — the raw one-pass aggregation; a positive integer dilates each shaded pixel by that radius (vellum::spread()); "auto" picks the radius from the image density (vellum::dynspread()). Datashaded lines/segments (auto = TRUE on mark_line() / mark_segment() etc.) default to "auto" since single-pixel lines otherwise vanish.
blend	Optional blend mode for compositing this layer against what is already drawn beneath it (the panel and earlier layers), one of the CSS mix-blend-mode names, e.g. "multiply", "screen", "darken". The whole layer composites as one isolated group (not per element).
data	Optional layer data frame; overrides the plot data for this layer.

Details

Map a discrete color (or fill) aesthetic to shade **categorically** (datashader's count_cat): each category is aggregated separately and every cell is coloured by the count-weighted average of the category hues it holds, with opacity from its total density — so you see which category dominates where, and where they mix, with a colour legend. The category hues come from the usual discrete colour scale (so scale_color_manual() etc. apply). Without a mapped colour, cell colour encodes plain density via the colors ramp.

Value

The modified PlotSpec.

Examples

```
n <- 1e5
d <- data.frame(x = rnorm(n), y = rnorm(n), g = sample(c("a", "b"), n, TRUE))
vplot(d) |> mark_datashade(x = x, y = y)
# categorical: colour by group
vplot(d) |> mark_datashade(x = x, y = y, color = g)
```

---

mark_ecdf *Distribution marks*

Description

Marks that summarise the distribution of a variable. `mark_ecdf()` draws the empirical cumulative distribution of `x` as a step; `mark_rug()` draws marginal ticks at each datum; `mark_qq()` draws a quantile-quantile plot of a `sample` against a theoretical distribution, with `mark_qq_line()` adding the reference line. All respect a mapped `color`/`fill` grouping.

Usage

```
mark_ecdf(plot, ..., blend = NULL, sketch = NULL, data = NULL)

mark_rug(plot, ..., sides = "bl", length = 0.03, blend = NULL, data = NULL)

mark_qq(
  plot,
  ...,
  distribution = "qnorm",
  blend = NULL,
  sketch = NULL,
  data = NULL
)

mark_qq_line(
  plot,
  ...,
  distribution = "qnorm",
  blend = NULL,
  sketch = NULL,
  data = NULL
)
```

Arguments

`plot`	A PlotSpec.
`...`	Encodings. `mark_ecdf()` needs `x`; `mark_qq()`/`mark_qq_line()` need `sample`; `mark_rug()` takes `x` and/or `y` (+ `color`, `alpha`, `linewidth`, `linetype`).
`blend, sketch, data`	Standard layer arguments (see `mark_point()`).
`sides`	Which edges `mark_rug()` draws ticks on: any of `"b"` (bottom), `"l"` (left), `"t"` (top), `"r"` (right); default `"bl"`.
`length`	Rug tick length as a fraction of the panel (default `0.03`).
`distribution`	Quantile function of the reference distribution for `mark_qq()` / `mark_qq_line()` (default stats::qnorm).

Value

The modified PlotSpec.

Examples

```
vplot(mtcars) |> mark_ecdf(x = mpg)
vplot(mtcars) |> mark_point(x = wt, y = mpg) |> mark_rug()
vplot(mtcars) |> mark_qq(sample = mpg) |> mark_qq_line(sample = mpg)
```

`mark_ellipse` *Group summary regions*

Description

Marks that summarise a set of `(x, y)` points (per group, when a `color` or `fill` is mapped) as a single enclosing region drawn over the raw points.

Usage

```
mark_ellipse(
  plot,
  ...,
  type = "t",
  level = 0.95,
  segments = 51L,
  blend = NULL,
  sketch = NULL,
  data = NULL
)

mark_hull(plot, ..., expand = 0, blend = NULL, sketch = NULL, data = NULL)
```

Arguments

`plot`	A PlotSpec (from `vplot()`).
`...`	Encodings (tidy-eval): `x`, `y`, and optionally `color`/`fill` (which also splits the points into per-group regions).
`type`	Ellipse type: `"t"`, `"norm"`, or `"euclid"`.
`level`	Confidence level (`"t"`/`"norm"`) or circle radius (`"euclid"`).
`segments`	Number of line segments approximating the ellipse.
`blend`	Optional blend mode for compositing this layer against what is already drawn beneath it (the panel and earlier layers), one of the CSS `mix-blend-mode` names, e.g. `"multiply"`, `"screen"`, `"darken"`. The whole layer composites as one isolated group (not per element).
`sketch`	A `sketch()` spec giving this layer a hand-drawn look (wobbly outlines, hachure fills), `NA`/`FALSE` to force it crisp (overriding a plot-wide `theme_sketch()`), or `NULL` (default) to inherit. Geometry marks accept it; text, raster, hex and datashade marks do not.
`data`	Optional layer data frame; overrides the plot data for this layer.

expand — For `mark_hull()`, grow each hull outward from its centroid by this fraction (e.g. `0.1` = 10% larger), so it encloses the markers it wraps rather than passing through their centres. `0` (default) is the tight hull.

Details

- `mark_ellipse()` draws a covariance ellipse: a `t` distribution (`type = "t"`, the default, robustly estimated via **MASS**), a multivariate normal (`"norm"`), or a Euclidean circle of radius `level` (`"euclid"`). The algorithm follows ggplot2's `stat_ellipse()`.
- `mark_hull()` draws the convex hull of the points.

Both are unfilled by default (a boundary), matching ggplot2; map or set a `fill` to shade the region. A region needs at least 3 points per group; smaller groups are skipped with a warning.

Value

The modified PlotSpec.

Examples

```
vplot(iris) |>
  mark_point(x = Sepal.Length, y = Sepal.Width, color = Species) |>
  mark_ellipse(x = Sepal.Length, y = Sepal.Width, color = Species)
vplot(iris) |>
  mark_point(x = Sepal.Length, y = Sepal.Width, color = Species) |>
  mark_hull(x = Sepal.Length, y = Sepal.Width, color = Species)
```

---

mark_graph — *Network (graph) marks*

Description

Draw a node-link diagram on a PlotSpec from `vgraph()`. `mark_edges()` draws the edges (straight lines, batched), `mark_nodes()` the vertices (points), `mark_node_text()` the vertex labels, and `mark_edge_text()` labels on the edges. Draw order is fixed regardless of the order you pipe them: edges under edge labels under nodes under node labels. Edges (and edge labels) default to the edge table (`vgraph()`'s `edge_data`), nodes and node labels to the node table; the `x`/`y`/`xend`/`yend`/`label`/`name` columns those tables carry are mapped automatically, so bare `mark_edges() |> mark_nodes()` just works.

Usage

```
mark_edges(
  plot,
  ...,
  color = NULL,
  linewidth = NULL,
  alpha = NULL,
```

```
  linetype = NULL,
  arrow = FALSE,
  routing = c("straight", "elbow"),
  elbow_at = c("mid", "start", "end"),
  elbow_axis = c("auto", "v", "h"),
  gradient = FALSE,
  auto = FALSE,
  blend = NULL,
  effects = list(),
  sketch = NULL,
  data = NULL
)

mark_edge_bundle(
  plot,
  ...,
  type = c("force", "divided", "stub", "path", "hammer", "mingle"),
  color = NULL,
  linewidth = NULL,
  alpha = NULL,
  linetype = NULL,
  params = list(),
  blend = NULL,
  effects = list(),
  sketch = NULL,
  data = NULL
)

mark_flow_map(
  plot,
  ...,
  root,
  type = c("spiral", "steiner"),
  weight = NULL,
  width_range = c(0.3, 3),
  color = NULL,
  alpha = NULL,
  params = list(),
  blend = NULL,
  effects = list(),
  sketch = NULL,
  data = NULL
)

mark_nodes(
  plot,
  ...,
  size = NULL,
```

```
  shape = NULL,
  fill = NULL,
  color = NULL,
  alpha = NULL,
  blend = NULL,
  effects = list(),
  sketch = NULL,
  data = NULL
)

mark_node_text(
  plot,
  ...,
  label = NULL,
  color = NULL,
  size = NULL,
  alpha = NULL,
  dist = 0,
  top_n = NULL,
  by = NULL,
  repel = FALSE,
  box_padding = 1,
  point_padding = 1,
  min_segment_length = 2,
  seed = NULL,
  effects = list(),
  blend = NULL,
  data = NULL
)

mark_edge_text(
  plot,
  ...,
  label = NULL,
  color = NULL,
  size = NULL,
  alpha = NULL,
  angle = NULL,
  effects = list(),
  blend = NULL,
  data = NULL
)

mark_node_hull(
  plot,
  ...,
  fill = NULL,
  color = NULL,
```

```
  alpha = 0.25,
  expand = 0.08,
  data = NULL
)

mark_node_pie(
  plot,
  cols,
  ...,
  size = 4,
  inner = 0,
  fill = NULL,
  color = "white",
  linewidth = 0.5,
  alpha = NA,
  data = NULL
)
```

Arguments

plot	A PlotSpec, normally from vgraph().
...	Encodings mapping node/edge attributes to aesthetics. Nodes: size, color/fill, shape, alpha. Edges: color, linewidth, linetype, alpha. The position channels are supplied by vgraph() and need not be mapped.
linewidth	For mark_edges(), the edge width; a constant or (via scale_edge_width()) a mapped expression such as linewidth = weight. For mark_node_pie(), the wedge border width.
linetype	For mark_edges(), the edge line type; a constant or (via scale_edge_linetype()) a mapped expression.
arrow	For mark_edges(), TRUE to draw a closed arrowhead at each edge's target end (the directed convention), FALSE/NULL for none, or a vellum::vl_arrow() spec for full control (ends, type, length, angle) – e.g. arrow = vellum::vl_arrow(ends = "both", type = "open"). Edges are capped exactly at each endpoint's node boundary (per vertex, at any size/resolution), so the head sits on the node edge; self-loops are drawn as teardrop loops sized to the node, with the head on the node boundary.
routing	For mark_edges(), edge routing: "straight" (default) or "elbow" – orthogonal right-angle steps for tree / DAG / dendrogram layouts (still straight segments, no curvature). Elbows keep node-boundary caps and arrowheads.
elbow_at, elbow_axis	For mark_edges(routing = "elbow"), the corner placement and axis. elbow_at is "mid" (default, an S-bend at the midpoint), or "start" / "end" for the corner at the source / target – "start" gives the dendrogram *bracket* (siblings share a bar at the parent's level). elbow_axis is "auto" (default, the longer side), "v", or "h" to force which axis the corner steps along.
gradient	For mark_edges(), TRUE to fade each (straight) edge from faint at its source to opaque at its target – a direction cue that needs no arrowhead (igraph's edge.gradient). Ignored with routing = "elbow".

auto — For mark_edges(), TRUE to datashade a large graph's edges as a density raster (vellum::datashade_segments()) once the edge count exceeds the datashade threshold, instead of drawing each edge as a vector segment — the fast, overplotting-honest path for hairballs. The device-space refinements of the vector path (parallel-edge offsets, node-boundary caps, arrowheads, teardrop self-loops) do not apply to the rasterised edges.

blend — Optional blend mode (see mark_point()).

effects — A list of layer render effects (glow(), outline(), shadow()) applied to the mark at draw time. On the text marks (mark_node_text() / mark_edge_text()) only shadow() applies.

sketch — A sketch() spec giving the layer a hand-drawn look, NA/FALSE to force it crisp, or NULL (default) to inherit.

data — Optional layer data; overrides the default table.

type — For mark_edge_bundle(), the bundling algorithm: "force" (default, force-directed / FDEB), "divided" (force-directed with direction-split bundles), "stub" (short stubs at each endpoint), "path" (shortest-path bundling), "hammer" (hammer bundling), or "mingle" (multilevel agglomerative edge bundling). Delegated to edgebundle::edge_bundle(). For mark_flow_map(), the flow-tree layout: "spiral" (default, angle-restricted spiral tree; **edgebundle** only) or "steiner" (approximate Steiner tree; also needs **interp**).

params — For mark_edge_bundle(), a named list of extra arguments passed to edgebundle::edge_bundle() for the chosen type (e.g. params = list(compatibility_threshold = 0.8)). For mark_flow_map(), extra arguments for the flow-tree builder – edgebundle::flow_tree() when type = "spiral" (e.g. list(alpha = 30)), edgebundle::tnss_tree() when type = "steiner" (e.g. list(gamma = 0.9)). Empty by default.

root — For mark_flow_map(), the source vertex the flow fans out from: a vertex name (matched against the node table) or a vertex index.

weight — For mark_flow_map(), the per-edge flow volume (a mapped edge column, e.g. weight = migrants). Defaults to the graph's weight edge attribute, or 1 if there is none.

width_range — For mark_flow_map(), the drawn branch width range c(min, max) (in linewidth units) that the computed flow is mapped onto. Default c(0.3, 3). The width is resolved at render inside the panel, so it is a physical width that holds at any figure size.

size, shape — For mark_nodes(), the node size (mm) / shape; a constant or a mapped expression. size is also the label font size for mark_node_text()/mark_edge_text().

fill, color, alpha — Convenience aesthetics; a constant or a mapped expression. For nodes, fill (or color) is the marker colour; for mark_edges(), color/alpha map through the edge scales.

label — For mark_node_text(), the label expression (default the vertex name); for mark_edge_text(), the edge label expression (no default – map an edge attribute, e.g. label = weight).

dist — For mark_node_text(), a radial offset (mm) pushing each label outward from the layout centroid, so labels clear the node markers instead of sitting on them. 0 (default) centres the label on the vertex.

`top_n`, `by` For `mark_node_text()`, label only the `top_n` vertices with the largest `by` (an edge/vertex metric column, e.g. `by = degree`) – the idiomatic "label just the hubs" filter. `NULL` (default) labels every vertex.

`repel` For `mark_node_text()`, `TRUE` to move overlapping labels apart with a force-directed layout (ggrepel-style), drawing a thin leader line back to each vertex. `box_padding`, `point_padding`, `min_segment_length`, and `seed` tune it exactly as in `mark_text()`.

`box_padding`, `point_padding`, `min_segment_length`, `seed`
Repel tuning for `mark_node_text(repel = TRUE)`; see `mark_text()`.

`angle` For `mark_edge_text()`, the label rotation: a constant in degrees, or `"along"` to rotate each label along its edge. `NULL` (default) draws horizontal labels.

`expand` For `mark_node_hull()`, the fraction to grow each hull outward from its centroid so it encloses the node markers (default `0.08`).

`cols` For `mark_node_pie()`, the compositional columns (a character vector of node-table column names, at least two) whose values size each node's wedges.

`inner` For `mark_node_pie()`, the inner-radius fraction: `0` (default) draws a pie, `> 0` a donut (e.g. `0.5`).

Details

Edge colour, opacity, and line type train on **their own scales** (`scale_edge_color()`, `scale_edge_alpha()`, `scale_edge_linetype()`, plus `scale_edge_width()`), independent of the node colour/alpha/linetype scales – so a plot can map node fill to a discrete community and edge colour to a continuous weight without the two legends colliding.

`mark_node_text()` has the label-legibility tools a dense graph needs: `repel = TRUE` nudges overlapping labels apart with leader lines, `dist` pushes labels radially clear of the node markers, `top_n`/`by` label only the hubs, and `effects = list(shadow())` drops a shadow behind each label so it stays readable over edges. `mark_edge_text()` takes the same `effects`. (Only `shadow()` applies to text; a glyph outline / glow is not available yet.)

Two more decorations: `mark_node_hull()` shades communities with a convex hull behind the graph (grouped by a mapped `fill`, drawn under the edges), and `mark_node_pie()` replaces node markers with pie / donut glyphs whose wedges come from a set of compositional columns.

`mark_edge_bundle()` is a drop-in alternative to `mark_edges()` that routes the edges as **bundled** curves instead of straight lines, so a hairball collapses into a few legible trunks. It delegates the geometry to the **edgebundle** package (install it) and draws the returned paths with the edge aesthetics – `type` picks the algorithm (`"force"`, `"divided"`, `"stub"`, `"path"`, `"hammer"`, `"mingle"`) and `params` passes tuning through. Bundled edges are faint by default (`alpha = 0.3`) so overlapping trunks read as density.

`mark_flow_map()` draws a **flow map**: a single `root` fans out to many destinations along smooth, merging branches whose width tracks the flow volume (the Minard / migration-map idiom). It expects a one-to-many (star) graph rooted at `root`, laid out at fixed coordinates; edge `weight` drives the flow. `type = "spiral"` (default, an angle-restricted spiral tree) needs only **edgebundle**; `type = "steiner"` (an approximate Steiner tree) additionally needs **interp**. Branch width is mapped from the computed flow into `width_range`, and each branch is drawn as **one tapering ribbon**: where tributaries merge and the flow changes along a branch, the width varies continuously rather than stepping. (That only arises for `type = "steiner"`, whose flow is per-vertex; a spiral tree's flow is

constant along each branch.) A `sketch = flow` map falls back to one uniform stroke per branch, since a hand-drawn ribbon would jitter its outline rather than the pen.

These are thin over the point / segment / text marks; `igraph` need not be installed to use them (only `vgraph()` needs it). `mark_edge_bundle()` and `mark_flow_map()` also need the **edgebundle** package.

Value

The modified PlotSpec.

See Also

`vgraph()`, `scale_edge_width()`, `scale_edge_color()`

Examples

```
## Not run:
g <- igraph::make_graph("Zachary")
vgraph(g) |>
  mark_edges(alpha = 0.5) |>
  mark_nodes(size = 4, fill = "steelblue")

## End(Not run)
```

---

mark_halfeye *Uncertainty marks (slab + interval)*

Description

Marks for visualising a distribution's shape *and* its summary intervals together, in the style of the **ggdist** package — a natural fit for posterior draws or bootstrap samples (one row per draw, grouped by a categorical `x`).

Usage

```
mark_halfeye(
  plot,
  ...,
  .width = c(0.66, 0.95),
  point = "median",
  adjust = 1,
  scale = 0.9,
  blend = NULL,
  sketch = NULL,
  data = NULL
)

mark_interval(
```

```
  plot,
  ...,
  .width = c(0.66, 0.95),
  point = "median",
  blend = NULL,
  data = NULL
)
```

Arguments

plot	A PlotSpec (from vplot()).
...	Encodings (tidy-eval): a categorical x and the sample y (+ color/fill).
.width	Interval probabilities, widest last (default c(0.66, 0.95)).
point	Central summary: "median" (default) or "mean".
adjust	mark_halfeye() density bandwidth multiplier.
scale	Slab width as a fraction of the category band (mark_halfeye()).
blend	Optional blend mode for compositing this layer against what is already drawn beneath it (the panel and earlier layers), one of the CSS mix-blend-mode names, e.g. "multiply", "screen", "darken". The whole layer composites as one isolated group (not per element).
sketch	A sketch() spec giving this layer a hand-drawn look (wobbly outlines, hachure fills), NA/FALSE to force it crisp (overriding a plot-wide theme_sketch()), or NULL (default) to inherit. Geometry marks accept it; text, raster, hex and datashade marks do not.
data	Optional layer data frame; overrides the plot data for this layer.

Details

- mark_halfeye() draws, per x category, a one-sided density "slab" (a half violin) with a **point-interval** at its base: the median (or mean) as a point, a thick bar for the inner interval and a thin bar for the outer one.
- mark_interval() draws the point-interval alone (no slab).

Intervals are equal-tailed quantile intervals at the .width probabilities (widest drawn thinnest). Supply many draws per x (e.g. posterior samples).

Value

The modified PlotSpec.

Examples

```
set.seed(1)
draws <- data.frame(
  grp = rep(c("a", "b", "c"), each = 400),
  val = rnorm(1200, rep(c(0, 1, 2), each = 400))
)
vplot(draws) |> mark_halfeye(x = grp, y = val)
vplot(draws) |> mark_interval(x = grp, y = val)
```

mark_histogram *Statistical marks*

Description

Marks that apply a statistical transform before drawing. `mark_histogram()` bins a continuous `x` and draws the per-bin counts as bars. `mark_smooth()` fits a model of `y` on `x` (per group) and draws the fitted line, with a confidence ribbon when `se = TRUE`.

Usage

```
mark_histogram(
  plot,
  ...,
  bins = 30,
  position = "stack",
  blend = NULL,
  sketch = NULL,
  data = NULL
)

mark_smooth(
  plot,
  ...,
  method = "auto",
  formula = NULL,
  span = 0.75,
  se = TRUE,
  level = 0.95,
  method.args = list(),
  blend = NULL,
  sketch = NULL,
  data = NULL
)
```

Arguments

`plot`	A PlotSpec (from `vplot()`).
`...`	Encodings (tidy-eval), e.g. `x`, `y`, `color`/`fill`.
`bins`	Number of histogram bins.
`position`	Position adjustment for the histogram bars (`"stack"`, `"dodge"`, `"fill"`).
`blend`	Optional blend mode for compositing this layer against what is already drawn beneath it (the panel and earlier layers), one of the CSS `mix-blend-mode` names, e.g. `"multiply"`, `"screen"`, `"darken"`. The whole layer composites as one isolated group (not per element).

sketch	A sketch() spec giving this layer a hand-drawn look (wobbly outlines, hachure fills), NA/FALSE to force it crisp (overriding a plot-wide theme_sketch()), or NULL (default) to inherit. Geometry marks accept it; text, raster, hex and datashade marks do not.
data	Optional layer data frame; overrides the plot data for this layer.
method	Smoothing method: one of "auto", "lm", "loess", "glm", "gam", "rq".
formula	Model formula in terms of x and y (e.g. y ~ poly(x, 2), y ~ s(x) for gam). Defaults to y ~ x (y ~ s(x) for gam).
span	loess neighbourhood size (larger = smoother).
se	Draw a confidence ribbon around the smooth? Ignored for "rq".
level	Confidence level for the ribbon.
method.args	Extra arguments to the fitting function, e.g. list(family = binomial()) for glm, or list(tau = 0.9) for rq.

Details

mark_smooth() supports several methods:

- "auto" (default) picks "loess" for small groups (< 1000 points) and "gam" for large ones.
- "lm" / "glm" — linear and generalised linear fits (glm takes a family via method.args, e.g. binomial() for logistic).
- "loess" — local regression, controlled by span.
- "gam" — a generalised additive model with a smooth term (default y ~ s(x)); needs the **mgcv** package.
- "rq" — quantile regression at a single method.args$tau (default the median); needs the **quantreg** package and draws the fitted line only (no confidence ribbon). For several quantiles, add one layer per tau.

Value

The modified PlotSpec.

Examples

```
vplot(mtcars) |> mark_histogram(x = mpg, bins = 10)
vplot(mtcars) |> mark_point(x = wt, y = mpg) |> mark_smooth(x = wt, y = mpg)
# local regression with a wider neighbourhood
vplot(mtcars) |>
  mark_point(x = wt, y = mpg) |>
  mark_smooth(x = wt, y = mpg, method = "loess", span = 0.9)
```

mark_image *Draw images at data points*

Description

`mark_image()` draws a bitmap image (e.g. a flag or logo) at each `(x, y)`, replacing the usual point marker. `src` is a column of local image file paths (per datum) or a single constant path (the same image at every point). Images are sized by `size` (height in millimetres); each image's native aspect ratio is preserved.

Usage

```
mark_image(
  plot,
  ...,
  src = NULL,
  size = 5,
  nudge_x = 0,
  nudge_y = 0,
  interpolate = TRUE,
  blend = NULL,
  data = NULL
)
```

Arguments

`plot`	A PlotSpec (from `vplot()`).
`...`	Encodings (tidy-eval): `x`, `y`, and `src` (the image path column).
`src`	Image source: a column of file paths (mapped, one per datum) or a single constant path (the same image at every point).
`size`	Image height in millimetres (constant or mapped); the width follows the image's own aspect ratio. Default `5`.
`nudge_x, nudge_y`	Shift each image by an exact absolute distance in millimetres (`+x` right, `+y` up). Default `0`.
`interpolate`	Smoothly interpolate when scaling (default `TRUE`)? `FALSE` keeps hard pixel edges.
`blend`	Optional blend mode for compositing this layer against what is already drawn beneath it (the panel and earlier layers), one of the CSS `mix-blend-mode` names, e.g. `"multiply"`, `"screen"`, `"darken"`. The whole layer composites as one isolated group (not per element).
`data`	Optional layer data frame; overrides the plot data for this layer.

Details

Reading image files requires the suggested magick package (which decodes PNG, JPEG, SVG, GIF and more); `mark_image()` errors with an install hint if it is not available.

Value

The modified PlotSpec.

Examples

```
## Not run:
d <- data.frame(x = 1:3, y = c(2, 1, 3),
                flag = c("de.png", "fr.png", "it.png"))
vplot(d) |> mark_image(x = x, y = y, src = flag, size = 8)

## End(Not run)
```

---

mark_outlier_label *Label the outliers*

Description

`mark_outlier_label()` labels only the **outlying** points — the extremes, where the story usually is — instead of every datum. It keeps the rows whose `y` is an outlier (per colour/fill group) and labels just those, repelled apart with the same solver as `mark_text()`. Layer it over the raw points.

Usage

```
mark_outlier_label(
  plot,
  ...,
  method = c("iqr", "sd"),
  k = 1.5,
  size = NULL,
  repel = TRUE,
  box_padding = 1,
  min_segment_length = 2,
  blend = NULL,
  data = NULL
)
```

Arguments

`plot`	A PlotSpec (from `vplot()`).
`...`	Encodings (tidy-eval): `x`, `y`, an optional `label` (what to write on each outlier), and optionally `color` / `fill` to detect outliers within each group.
`method`	Outlier rule: `"iqr"` (Tukey's, the default) or `"sd"`.
`k`	Threshold multiplier — IQR whisker length (default `1.5`) or number of standard deviations.
`size`	Font size in points.

repel	Move overlapping labels apart (force-directed, ggrepel-style), with leader lines to the points? Single cartesian panel only.
box_padding	Extra space (mm) kept around each label box during repulsion.
min_segment_length	Shortest leader line (mm) worth drawing; a label that barely moved gets none.
blend	Optional blend mode for compositing this layer against what is already drawn beneath it (the panel and earlier layers), one of the CSS mix-blend-mode names, e.g. "multiply", "screen", "darken". The whole layer composites as one isolated group (not per element).
data	Optional layer data frame; overrides the plot data for this layer.

Details

The default method = "iqr" flags y outside [Q1 - k*IQR, Q3 + k*IQR] (the boxplot whisker rule); method = "sd" flags |y - mean| > k * sd. Map a label to name each outlier; with none mapped, the outlier's y value is the label.

Value

The modified PlotSpec.

See Also

mark_text(), mark_boxplot()

Examples

```
vplot(mtcars) |>
  mark_point(x = wt, y = mpg) |>
  mark_outlier_label(x = wt, y = mpg, label = rownames(mtcars))
```

---

mark_pie *Pie and donut charts*

Description

Convenience marks for part-of-whole charts. mark_pie() draws a pie: each value becomes a wedge whose angle is its share of the total, coloured by fill. mark_donut() is a pie with a hollow centre (inner_radius, a fraction of the radius). Both are shorthand for a stacked bar projected through coord_polar() with theta = "y", which they set on the plot; they error if the plot already carries a non-polar coordinate.

Usage

```
mark_pie(
  plot,
  value,
  fill = NULL,
  ...,
  blend = NULL,
  sketch = NULL,
  data = NULL
)

mark_donut(
  plot,
  value,
  fill = NULL,
  inner_radius = 0.5,
  ...,
  blend = NULL,
  sketch = NULL,
  data = NULL
)
```

Arguments

plot	A PlotSpec.
value	Encoding (tidy-eval) for each slice's magnitude.
fill	Encoding (tidy-eval) for the slice colour. Omit for a single slice.
...	Further constant aesthetics (e.g. alpha).
blend	Optional blend mode for compositing the layer (see mark_point()).
sketch	A sketch() spec giving the layer a hand-drawn look, or NULL (default) to inherit.
data	Optional per-layer data frame.
inner_radius	For mark_donut(), the inner-hole radius as a fraction of the rim (0 is a pie, the default 0.5 a medium donut).

Value

The modified PlotSpec.

See Also

coord_polar()

Examples

```
df <- data.frame(part = c("a", "b", "c"), n = c(3, 5, 2))
vplot(df) |> mark_pie(value = n, fill = part)
vplot(df) |> mark_donut(value = n, fill = part, inner_radius = 0.6)
```

mark_point *Add marks to a plot*

Description

Each `mark_*()` appends a drawing layer to a PlotSpec. Encodings are bare column names (or expressions) captured with tidy evaluation, e.g. `x = wt, y = mpg, color = hp`. Scalar values (e.g. `size = 3`, `color = "red"`) are treated as constant aesthetics rather than data mappings.

Usage

```
mark_point(
  plot,
  ...,
  size = NULL,
  shape = NULL,
  position = "identity",
  auto = FALSE,
  seed = NULL,
  blend = NULL,
  effects = list(),
  sketch = NULL,
  data = NULL
)

mark_line(
  plot,
  ...,
  window = NULL,
  auto = FALSE,
  blend = NULL,
  effects = list(),
  sketch = NULL,
  data = NULL
)

mark_rule(
  plot,
  ...,
  blend = NULL,
  effects = list(),
  sketch = NULL,
  data = NULL
)

mark_bar(
  plot,
```

```
  ...,
  position = "stack",
  blend = NULL,
  sketch = NULL,
  data = NULL
)
```

Arguments

plot	A PlotSpec (from vplot()).
...	Encodings: named channel expressions such as x, y, color, fill, size, shape, alpha. Stroked marks (line, step, segment, rule, linerange) also take linewidth and linetype. On mark_line(), mark_step() and mark_segment(), linewidth may be **mapped** as well as set: see scale_linewidth() for the per-segment width that gives, and what it costs. Everywhere else linewidth is a constant.
size, shape	Convenience arguments for the point size (in mm) / shape; may be a constant or a mapped expression. A shape is a built-in marker ("circle", "square", "triangle", "diamond", "plus", "cross", "triangle_down", "star") or an **SVG icon** — a path d string or a .svg file — drawn as a crisp vector marker. Use a literal string for a constant icon, or map shape and pass icons to scale_shape().
position	Position adjustment: a string — "identity" (default), "jitter" / "jitterdodge" (points), "stack" / "fill" / "dodge" / "dodge2" (bars), "nudge" — or a parameterised position_jitter() / position_dodge() / position_dodge2() / position_jitterdodge() / position_nudge() object.
auto	For mark_point(), mark_line(), mark_step(), mark_segment(), and mark_edges(), when TRUE and the layer has very many rows, automatically render it as a datashaded density raster (see mark_datashade()) instead of individual vector marks: points bin into a density grid (vellum::datashade()), dense lines/steps rasterise as connected polylines (vellum::datashade_lines()), and segments/edges as independent segments (vellum::datashade_segments()). The datashaded line/segment output is dynspread-ed so thin marks stay visible (see the spread argument of mark_datashade()). The fallback is skipped under a warped coordinate system (coord_polar() / coord_trans()), which draws the vector marks instead.
seed	For mark_point(position = "jitter"), an optional integer seed making the jitter reproducible. The global RNG stream is restored afterwards.
blend	Optional blend mode for compositing this layer against what is already drawn beneath it (the panel and earlier layers), one of the CSS mix-blend-mode names, e.g. "multiply", "screen", "darken". The whole layer composites as one isolated group (not per element).
effects	A list of layer render effects applied to the mark at draw time — glow(), outline(), and shadow(). Available on stroked and point marks.
sketch	A sketch() spec giving this layer a hand-drawn look (wobbly outlines, hachure fills), NA/FALSE to force it crisp (overriding a plot-wide theme_sketch()), or NULL (default) to inherit. Geometry marks accept it; text, raster, hex and datashade marks do not.

`data` Optional layer data frame; overrides the plot data for this layer.

`window` For `mark_line()`, an optional window (rolling / cumulative / offset) transform of `y` computed per group over rows ordered by `x`, before the line is drawn. Either an op name (`"mean"`, `"sum"`, `"median"`, `"min"`, `"max"`, `"cumsum"`, `"cummean"`, `"cummax"`, `"cummin"`, `"lag"`, `"lead"`, `"rank"`) or a list `list(op=, k=, align=, partial=)`: `k` is the window size (rolling; default 7) or shift (`lag`/`lead`; default 1), `align` is `"right"` (trailing, default), `"left"`, or `"center"`, and `partial` (default `TRUE`) computes at the edges from the shorter available window. For example `window = list(op = "mean", k = 7)` is a 7-point trailing average.

Details

`mark_bar()` draws bars from a zero baseline. With an explicit `y` it uses the `y` values as heights; with no `y` it counts rows per category (the `"count"` stat). When `color`/`fill` is mapped, grouped bars are stacked by default; use `position = "dodge"` for side-by-side bars or `"fill"` to normalise to 1.

Value

The modified PlotSpec.

Interactivity

Any mark accepts reserved, per-row arguments (captured like encodings, via tidy evaluation) that make its elements addressable — and stylable — by an interactive host without changing what a static render draws:

- `data_id` — a per-element **data key** (e.g. `data_id = model`). Emitted by the SVG backend as `data-key` on each element and returned by `vellum::scene_model()`; it is the join key a host uses to map a hover/click back to a datum, and to link the same datum across views.
- `tooltip` — per-element tooltip text (a column expression or a constant), surfaced in `scene_model()` metadata.
- `hover_group` — a field grouping elements for linked emphasis (consumed by a host in a later phase).
- `hover_color`, `selected_color` — per-element outline colours applied by the host when the element is hovered / selected (a constant, or mapped from a column so different marks highlight differently). They override the widget-wide theme set by `vellumwidget::as_widget(hover_color=, selected_color=)`.

These are inert for PNG/PDF and for an SVG opened without a JS host: a plot with none of them compiles and renders exactly as before. Declaring any of them without `data_id` defaults the key to the row index, so the element is still addressable. They currently apply to `stat = "identity"` marks (points, bars, tiles, segments, edges, hexbins, sf features, ...); aggregating stats (histogram/count/density) drop them, since rows no longer map 1:1 to elements.

How these flow into the vellum scene (the `scene_model()` element table, the SVG `data-key` / `data-vellum-*` attributes, and the reserved `meta` key vocabulary) is described in vellum's "The scene contract" vignette (`vignette("scene-contract", package = "vellum")`).

Examples

```
vplot(mtcars) |> mark_point(x = wt, y = mpg, color = hp)

# Declare interactivity (inert on a static render):
df <- data.frame(wt = mtcars$wt, mpg = mtcars$mpg, model = rownames(mtcars))
vplot(df) |> mark_point(x = wt, y = mpg, tooltip = model, data_id = model)
```

`mark_raincloud` *Raincloud plot*

Description

`mark_raincloud()` is a convenience that composes a **raincloud**: a one-sided density "cloud" (`mark_halfeye()`) with the raw observations as "rain" below it (`mark_point()` with sina spread). It is exactly `plot |> mark_halfeye(...) |> mark_point(..., position = position_sina())`, so any `color`/`fill` mapping in `...` flows to both layers.

Usage

```
mark_raincloud(plot, x, y, ..., width = 0.4, alpha = 0.5)
```

Arguments

`plot`	A PlotSpec.
`x, y`	The categorical `x` and the sample `y` (tidy-eval).
`...`	Further shared encodings (e.g. `color`, `fill`) forwarded to both the slab and the points.
`width`	Sina spread of the rain, as a fraction of the band (default `0.4`).
`alpha`	Point opacity for the rain (default `0.5`).

Value

The modified PlotSpec.

See Also

`mark_halfeye()`, `position_sina()`

Examples

```
set.seed(1)
d <- data.frame(g = rep(c("a", "b", "c"), each = 100),
                y = rnorm(300, rep(c(0, 1, 2), each = 100)))
vplot(d) |> mark_raincloud(x = g, y = y)
```

mark_segment *Segment mark*

Description

mark_segment() draws a straight line from (x, y) to (xend, yend) per row.

Usage

```
mark_segment(
  plot,
  ...,
  auto = FALSE,
  blend = NULL,
  effects = list(),
  sketch = NULL,
  data = NULL
)
```

Arguments

plot	A PlotSpec (from vplot()).
...	Encodings (tidy-eval): x, y, xend, yend (+ color, linewidth, linetype, alpha). A mapped linewidth gives each segment its own width – see scale_linewidth().
auto	For mark_point(), mark_line(), mark_step(), mark_segment(), and mark_edges(), when TRUE and the layer has very many rows, automatically render it as a datashaded density raster (see mark_datashade()) instead of individual vector marks: points bin into a density grid (vellum::datashade()), dense lines/steps rasterise as connected polylines (vellum::datashade_lines()), and segments/edges as independent segments (vellum::datashade_segments()). The datashaded line/segment output is dynspread-ed so thin marks stay visible (see the spread argument of mark_datashade()). The fallback is skipped under a warped coordinate system (coord_polar() / coord_trans()), which draws the vector marks instead.
blend	Optional blend mode for compositing this layer against what is already drawn beneath it (the panel and earlier layers), one of the CSS mix-blend-mode names, e.g. "multiply", "screen", "darken". The whole layer composites as one isolated group (not per element).
effects	A list of layer render effects applied to the mark at draw time — glow(), outline(), and shadow(). Available on stroked and point marks.
sketch	A sketch() spec giving this layer a hand-drawn look (wobbly outlines, hachure fills), NA/FALSE to force it crisp (overriding a plot-wide theme_sketch()), or NULL (default) to inherit. Geometry marks accept it; text, raster, hex and datashade marks do not.
data	Optional layer data frame; overrides the plot data for this layer.

Value

The modified PlotSpec.

See Also

`scale_linewidth()`

Examples

```
d <- data.frame(x = 1, y = 1, xend = 5, yend = 4)
vplot(d) |> mark_segment(x = x, y = y, xend = xend, yend = yend)
```

---

mark_series_label *Direct series labels*

Description

`mark_series_label()` labels each colour/fill **series at its end** (the point with the largest `x`) with the series name — a legend-free alternative that reads faster on a multi-line chart. Labels are pushed apart with the same repel solver as `mark_text()`. Map the same `x`/`y`/`color` as the lines; the label text and colour come from the series automatically.

Usage

```
mark_series_label(
  plot,
  ...,
  size = NULL,
  nudge_x = 2,
  repel = TRUE,
  box_padding = 1,
  min_segment_length = 2,
  blend = NULL,
  data = NULL
)
```

Arguments

`plot`	A PlotSpec (from `vplot()`).
`...`	Encodings (tidy-eval): the same `x`, `y`, and `color`/`fill` as the series being labelled.
`size`	Font size in points.
`nudge_x`	Rightward offset (mm) so a label clears its line end (default `2`). Widen the x range (or `coord_cartesian(clip = FALSE)`) if labels crowd the panel edge.
`repel`	Move overlapping labels apart (force-directed, ggrepel-style), with leader lines to the points? Single cartesian panel only.

box_padding Extra space (mm) kept around each label box during repulsion.

min_segment_length Shortest leader line (mm) worth drawing; a label that barely moved gets none.

blend Optional blend mode for compositing this layer against what is already drawn beneath it (the panel and earlier layers), one of the CSS `mix-blend-mode` names, e.g. `"multiply"`, `"screen"`, `"darken"`. The whole layer composites as one isolated group (not per element).

data Optional layer data frame; overrides the plot data for this layer.

Value

The modified PlotSpec.

See Also

`mark_text()`

Examples

```
d <- data.frame(
  t = rep(1:10, 3), v = c(1:10, (1:10) * 1.5, 10:1),
  s = rep(c("a", "b", "c"), each = 10)
)
# Give the panel a little x-room and drop the now-redundant colour legend.
vplot(d) |>
  mark_line(x = t, y = v, color = s) |>
  mark_series_label(x = t, y = v, color = s) |>
  xlim(1, 12) |>
  guides(color = "none")
```

---

mark_sf *Draw simple-feature (sf) geometries*

Description

`mark_sf()` draws the geometry column of an `sf` object as a map layer: `POINT`/`MULTIPOINT` render as points, `LINESTRING`/`MULTILINESTRING` as polylines, and `POLYGON`/`MULTIPOLYGON` as filled paths (holes cut with the even-odd rule, so ring winding need not be canonical). Coordinates come from the geometry, so there are no `x`/`y` encodings; other aesthetics map feature attributes as usual, e.g. `fill = AREA` for a choropleth. Pair with `coord_sf()` to reproject and lock the map aspect ratio.

Usage

```
mark_sf(
  plot,
  ...,
  fill = NULL,
  color = NULL,
```

```
  alpha = NULL,
  linewidth = NULL,
  size = NULL,
  na_value = "grey80",
  merge = FALSE,
  blend = NULL,
  sketch = NULL,
  data = NULL
)
```

Arguments

plot	A PlotSpec (from vplot()).
...	Encodings mapping feature attributes to aesthetics: fill, color, alpha, linewidth, size. A geometry column is not encoded — it is read from the data.
fill, color, alpha, linewidth, size	Convenience aesthetic arguments; a constant or a mapped expression.
na_value	Fill colour for features whose mapped fill/color value is NA (drawn as a distinct legend swatch). Default "grey80".
merge	Dissolve adjacent features that share a fill into one region, so the internal borders between same-valued features disappear and each region is a single crisp path (no seam artefacts, and exact in PDF). The union is real boolean geometry via vellum::vl_path_op(). Default FALSE. Ignored when the layer is interactive (merging would drop the per-feature keys).
blend	Optional blend mode (see mark_point()).
sketch	A sketch() spec giving the layer a hand-drawn look, or NULL (default) to inherit.
data	Optional layer data (an sf object); overrides the plot data.

Details

sf is an optional dependency (in Suggests); mark_sf() errors with an install hint if it is not available.

Value

The modified PlotSpec.

See Also

coord_sf(), scale_fill_binned()

Examples

```
## Not run:
nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
vplot(nc) |> mark_sf(fill = BIR74) |> coord_sf()

## End(Not run)
```

mark_sf_label *Label sf features at their interior point*

Description

mark_sf_label() draws one text label per sf feature, placed at the feature's **interior point** (sf::st_point_on_surface(), guaranteed to fall inside each polygon — unlike a centroid, which can land in a bay or a hole). The label points are reprojected through the *same* coord_sf CRS as mark_sf(), so they always land where the geometry is drawn. Layer it over a mark_sf() choropleth to name the regions; repel = TRUE (the default) pushes crowded labels apart.

Usage

```
mark_sf_label(
  plot,
  ...,
  size = NULL,
  repel = TRUE,
  box_padding = 1,
  min_segment_length = 2,
  blend = NULL,
  data = NULL
)
```

Arguments

plot	A PlotSpec (from vplot()).
...	Encodings (tidy-eval): the label to draw at each feature, plus an optional color / size. The position comes from the geometry, so x / y are not mapped.
size	Font size in points.
repel	Move overlapping labels apart (force-directed, ggrepel-style), with leader lines to the points? Single cartesian panel only.
box_padding	Extra space (mm) kept around each label box during repulsion.
min_segment_length	Shortest leader line (mm) worth drawing; a label that barely moved gets none.
blend	Optional blend mode for compositing this layer against what is already drawn beneath it (the panel and earlier layers), one of the CSS mix-blend-mode names, e.g. "multiply", "screen", "darken". The whole layer composites as one isolated group (not per element).
data	An sf data frame (defaults to the plot's). Its geometry supplies the label positions.

Value

The modified PlotSpec.

See Also

mark_sf(), mark_text()

Examples

```
if (requireNamespace("sf", quietly = TRUE)) {
  nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
  vplot(nc) |>
    mark_sf(fill = AREA) |>
    mark_sf_label(label = NAME, size = 5)
}
```

---

mark_signif *Significance brackets*

Description

mark_signif() runs a pairwise test between x groups and draws a significance **bracket** over each comparison with its p-value (or stars) — the ggsignif / ggpubr idiom. Add it on top of a mark_boxplot() / mark_violin(); the brackets stack above the data and the y-axis expands to fit them.

Usage

```
mark_signif(
  plot,
  ...,
  comparisons = NULL,
  method = c("wilcox.test", "t.test"),
  label = c("p", "stars"),
  step = 0.12,
  tip_length = 0.03,
  blend = NULL,
  sketch = NULL,
  data = NULL
)
```

Arguments

plot	A PlotSpec (from vplot()).
...	Encodings (tidy-eval): a categorical x and the numeric y.
comparisons	A list of length-2 character vectors naming the group pairs to test, e.g. list(c("a", "b"), c("b", "c")). NULL (default) tests each adjacent pair of levels.
method	The two-sample test: "wilcox.test" (default) or "t.test".
label	"p" (default, a formatted p-value) or "stars" (*/**/***/****/ns).

step	Vertical gap between stacked brackets, as a fraction of the data's y range (default 0.12).
tip_length	Bracket down-tick length, as a fraction of the y range (default 0.03).
blend	Optional blend mode for compositing this layer against what is already drawn beneath it (the panel and earlier layers), one of the CSS mix-blend-mode names, e.g. "multiply", "screen", "darken". The whole layer composites as one isolated group (not per element).
sketch	A sketch() spec giving this layer a hand-drawn look (wobbly outlines, hachure fills), NA/FALSE to force it crisp (overriding a plot-wide theme_sketch()), or NULL (default) to inherit. Geometry marks accept it; text, raster, hex and datashade marks do not.
data	Optional layer data frame; overrides the plot data for this layer.

Value

The modified PlotSpec.

Examples

```
set.seed(1)
d <- data.frame(g = rep(c("a", "b", "c"), each = 30),
                y = c(rnorm(30), rnorm(30, 1), rnorm(30, 0.4)))
vplot(d) |>
  mark_boxplot(x = g, y = y) |>
  mark_signif(x = g, y = y, comparisons = list(c("a", "b"), c("a", "c")))
```

---

mark_text *Text marks*

Description

mark_text() draws the label aesthetic as text at each (x, y); mark_label() adds a filled rounded background behind each label. size is the font size in points; angle (degrees) may be mapped or constant.

Usage

```
mark_text(
  plot,
  ...,
  size = NULL,
  family = NULL,
  fontface = NULL,
  hjust = "centre",
  vjust = "centre",
  angle = NULL,
  nudge_x = 0,
```

```
  nudge_y = 0,
  repel = FALSE,
  box_padding = 1,
  point_padding = 1,
  min_segment_length = 2,
  seed = NULL,
  effects = list(),
  blend = NULL,
  data = NULL
)

mark_label(
  plot,
  ...,
  size = NULL,
  family = NULL,
  fontface = NULL,
  hjust = "centre",
  vjust = "centre",
  angle = NULL,
  nudge_x = 0,
  nudge_y = 0,
  fill = "white",
  repel = FALSE,
  box_padding = 1,
  point_padding = 1,
  min_segment_length = 2,
  seed = NULL,
  effects = list(),
  blend = NULL,
  sketch = NULL,
  data = NULL
)
```

Arguments

`plot`	A PlotSpec (from `vplot()`).
`...`	Encodings (tidy-eval): `x`, `y`, `label` (+ `color`, `angle`).
`size`	Font size in points.
`family`, `fontface`	Font family / face (`"plain"`, `"bold"`, `"italic"`, `"bold.italic"`).
`hjust`, `vjust`	Horizontal / vertical justification (constant; `"left"`, `"centre"`, `"right"`, `"bottom"`, `"top"`, or numeric in `[0, 1]`).
`angle`	Text rotation in degrees.
`nudge_x`, `nudge_y`	Shift each label by an exact absolute distance in millimetres (`+x` right, `+y` up), device-resolved so the nudge is constant regardless of scale or panel aspect. Default `0`.

repel — Move overlapping labels apart (force-directed, ggrepel-style), with leader lines to the points? Single cartesian panel only.

box_padding — Extra space (mm) kept around each label box during repulsion.

point_padding — Retained for back-compatibility and currently ignored: the repulsion solver is deterministic and pads uniformly.

min_segment_length — Shortest leader line (mm) worth drawing; a label that barely moved gets none.

seed — Retained for back-compatibility and currently a no-op: the repel layout is deterministic, so there is nothing for a seed to vary.

effects — A list of layer render effects applied to the mark at draw time — glow(), outline(), and shadow(). Available on stroked and point marks.

blend — Optional blend mode for compositing this layer against what is already drawn beneath it (the panel and earlier layers), one of the CSS mix-blend-mode names, e.g. "multiply", "screen", "darken". The whole layer composites as one isolated group (not per element).

data — Optional layer data frame; overrides the plot data for this layer.

fill — For mark_label(), the label background: a constant colour, or a mapped encoding (e.g. fill = group) coloured through the fill/colour scale.

sketch — A sketch() spec giving this layer a hand-drawn look (wobbly outlines, hachure fills), NA/FALSE to force it crisp (overriding a plot-wide theme_sketch()), or NULL (default) to inherit. Geometry marks accept it; text, raster, hex and datashade marks do not.

Details

The label for mark_text() may be plain text (embedded newlines \n wrap onto stacked lines) or rich vellum::md() labels — map label = md(<expr>) for a per-datum styled label (bold/italic/super-/subscript/colour). (Rich labels are not yet supported by mark_label()'s background box.)

Set repel = TRUE to move overlapping labels apart with a force-directed layout (like ggrepel), drawing a thin leader line back to each label's point. Repulsion is resolved exactly against the true rendered panel size, so it does not depend on the data scale. It is currently limited to a single cartesian panel (no facets / composition / polar).

Value

The modified PlotSpec.

Examples

```
vplot(mtcars) |> mark_text(x = wt, y = mpg, label = rownames(mtcars))
vplot(mtcars) |> mark_point(x = wt, y = mpg) |>
  mark_text(x = wt, y = mpg, label = rownames(mtcars), nudge_y = 2)
```

mark_text_path *Text set along a path*

Description

mark_text_path() draws a label that *follows a curve* — one string per group, its glyphs placed along the group's x/y points (in data order) and rotated to the local tangent. Use it for a label riding a contour, an arc, or a trend line (a direct-labelling alternative to a legend), where mark_text()'s point-anchored, axis-aligned labels do not fit.

Usage

```
mark_text_path(
  plot,
  ...,
  size = NULL,
  family = NULL,
  fontface = NULL,
  hjust = "centre",
  vjust = "centre",
  offset = 0,
  blend = NULL,
  data = NULL
)
```

Arguments

plot	A PlotSpec (from vplot()).
...	Encodings (tidy-eval): x, y (the path), label (the string), and optionally color / group to split into separate runs.
size	Font size in points.
family, fontface	Font family and face ("plain", "bold", ...).
hjust	Where the run sits along the path: "left" starts it at the first point, "centre" centres it, "right" ends it at the last point.
vjust	Vertical placement of the glyphs against the baseline.
offset	Perpendicular standoff from the path, in points (positive is to the left of travel), for a label riding just above or below its curve.
blend	Optional blend mode for compositing this layer against what is already drawn beneath it (the panel and earlier layers), one of the CSS mix-blend-mode names, e.g. "multiply", "screen", "darken". The whole layer composites as one isolated group (not per element).
data	Optional layer data frame; overrides the plot data for this layer.

Details

One label is drawn per group. Split groups the usual way — a discrete `color` (or a distinct `label`) starts a new run — and the label is taken from the group's first row (it is constant along the path). The points are used in the order they appear in the data: pre-sort if the curve should be traversed in a particular direction. Because glyphs follow the tangent (like SVG `textPath`), a label on the underside of a closed curve reads upside-down; reverse the path to flip it.

Value

The modified PlotSpec.

Examples

```
# Traverse the arc left-to-right so the glyphs read upright (a path walked
# right-to-left would set the label upside-down).
t <- seq(pi, 0, length.out = 40)
d <- data.frame(x = cos(t), y = sin(t), lab = "along the arc")
vplot(d) |> mark_text_path(x = x, y = y, label = lab)
```

`mark_tile` *Heatmap marks*

Description

`mark_tile()` draws a rectangular tile at each `(x, y)` coloured by `fill`; `mark_raster()` draws the same as one raster image (a fast path requiring a complete regular grid). `mark_bin2d()` bins continuous `x`/`y` into a grid and colours each cell by count. `mark_density()` draws a 1-D kernel density of `x` as a filled curve.

Usage

```
mark_tile(plot, ..., blend = NULL, sketch = NULL, data = NULL)

mark_raster(plot, ..., blend = NULL, data = NULL)

mark_bin2d(plot, ..., bins = 30, blend = NULL, data = NULL)

mark_density(plot, ..., adjust = 1, blend = NULL, sketch = NULL, data = NULL)

mark_hex(plot, ..., bins = 30, blend = NULL, data = NULL)
```

Arguments

`plot`	A PlotSpec (from `vplot()`).
`...`	Encodings (tidy-eval): `x`, `y`, `fill` for tile/raster; `x`, `y` for bin2d; `x` (+ `fill`/`color`) for density.

blend	Optional blend mode for compositing this layer against what is already drawn beneath it (the panel and earlier layers), one of the CSS mix-blend-mode names, e.g. "multiply", "screen", "darken". The whole layer composites as one isolated group (not per element).
sketch	A sketch() spec giving this layer a hand-drawn look (wobbly outlines, hachure fills), NA/FALSE to force it crisp (overriding a plot-wide theme_sketch()), or NULL (default) to inherit. Geometry marks accept it; text, raster, hex and datashade marks do not.
data	Optional layer data frame; overrides the plot data for this layer.
bins	Number of bins per axis for mark_bin2d() / hex columns for mark_hex().
adjust	Bandwidth multiplier for mark_density().

Value

The modified PlotSpec.

Examples

```
d <- expand.grid(x = 1:5, y = 1:5)
d$z <- d$x * d$y
vplot(d) |> mark_tile(x = x, y = y, fill = z)
```

---

mark_violin *Density-shape marks*

Description

Marks that draw a variable's distribution as a filled shape. mark_violin() draws a mirrored kernel density of y per categorical x (like a boxplot's footprint); mark_ridgeline() draws a kernel density of x per categorical y as overlapping ridges. mark_dotplot() bins x and stacks one dot per observation.

Usage

```
mark_violin(plot, ..., adjust = 1, blend = NULL, sketch = NULL, data = NULL)

mark_ridgeline(
  plot,
  ...,
  adjust = 1,
  height = 1.4,
  blend = NULL,
  sketch = NULL,
  data = NULL
)
```

```
mark_dotplot(
  plot,
  ...,
  binwidth = NULL,
  blend = NULL,
  sketch = NULL,
  data = NULL
)
```

Arguments

plot	A PlotSpec.
...	Encodings. mark_violin() needs categorical x and numeric y; mark_ridgeline() needs numeric x and categorical y; mark_dotplot() needs x. A mapped color/fill sets the shape fill.
adjust	Kernel-density bandwidth multiplier (violin/ridgeline).
blend, sketch, data	Standard layer arguments (see mark_point()).
height	Ridge height as a multiple of the row band (ridgeline; default 1.4, so adjacent ridges overlap slightly).
binwidth	Dot-plot bin width, or NULL to use ~1/30 of the data range.

Value

The modified PlotSpec.

Examples

```
df <- data.frame(g = rep(letters[1:3], each = 50), v = rnorm(150))
vplot(df) |> mark_violin(x = g, y = v)
vplot(df) |> mark_ridgeline(x = v, y = g)
vplot(df) |> mark_dotplot(x = v)
```

---

mcp_serve *Run the vellumplot MCP server*

Description

Starts a Model Context Protocol server on stdio so an agent can generate validated vellumplot specs. It exposes three tools — get_schema, list_fields, and render_spec — that delegate to spec_schema(), spec_fields(), and spec_diagnose() + render_plot(). A render_spec failure returns structured diagnostics (unknown field, compile error) so the agent can repair and retry, rather than a traceback.

Usage

```
mcp_serve(input = "stdin", output = stdout())
```

Arguments

input, output	Connections to read requests from / write responses to (defaults: standard input / output).

Details

The server is pure R (newline-delimited JSON-RPC 2.0 over stdio) and needs no Node or SDK. Point an MCP client at it with: `Rscript -e 'vellumplot::mcp_serve()'`, or the bundled launcher `system.file("mcp/server.R", package = "vellumplot")`.

Value

Invisibly `NULL`, when the input stream closes.

See Also

`spec_schema()`, `spec_fields()`, `vplot_from_spec()`

---

md *Rich-text labels*

Description

A thin wrapper around `vellum::md()` that builds a rich-text label from a markdown subset: `**bold**`, `*italic*` / `_italic_`, `^superscript^`, `~subscript~`, and a colour span `[text]{#c00}`. The result can be used anywhere vellumplot draws a *title*: `labs()` (`title` / `subtitle` / `caption` / `tag` / `x` / `y` / `color`) and `scale_*(name = )`. Per-element rich labels (in `mark_text()`) are not yet supported.

Usage

```
md(text)
```

Arguments

text	A length-one markdown string.

Value

A `vellum_md_label` object accepted by `vellum::text_grob()`.

Examples

```
vplot(mtcars) |>
  mark_point(x = wt, y = mpg) |>
  labs(title = md("**Fuel** economy"), y = md("Efficiency (mi gal^-1^)"))
```

`motion` *Motion-trail layer effects*

Description

Draw a fading trail of a stroked or point mark: `n` copies marching off along a direction (`x`, `y`), each further out and fainter than the last, composited beneath the crisp original — a speed-blur or animation-still look. `motion()` defaults to many close, low-opacity copies (a smooth blur streak); `echo()` defaults to a few wider-spaced, more-opaque copies (discrete ghost repeats). Both build the same effect, differing only in defaults, and apply to the same marks as `glow()`.

Usage

```
motion(
  x = 3,
  y = 0,
  n = 8L,
  alpha = 0.15,
  decay = 1,
  spread = 0,
  blend = "normal",
  color = NULL
)

echo(
  x = 4,
  y = 0,
  n = 3L,
  alpha = 0.45,
  decay = 1,
  spread = 0,
  blend = "normal",
  color = NULL
)
```

Arguments

`x`, `y`	Trail direction and length in millimetres (`+x` right, `+y` up): the offset of the furthest copy. Defaults to a rightward streak.
`n`	Number of trail copies.
`alpha`	Opacity of the nearest (strongest) copy; copies fade toward the tail.
`decay`	Fade exponent shaping the opacity ramp along the trail (higher = faster fade toward the tail; `0` = no fade).
`spread`	Optional widening, in millimetres, applied progressively toward the tail (`0` keeps a constant width).

blend	Blend mode compositing the trail copies (any CSS mix-blend-mode name); "normal" for opaque ghosts.
color	Trail colour, or NULL (default) to inherit the mark's own resolved colour.

Details

The direction is an **absolute distance in millimetres** (+x right, +y up), resolved device-side via vellum's compound npc + mm unit, so the trail keeps the same physical length and stays isotropic regardless of the panel's size or aspect (as shadow()'s offset does).

Value

A MotionSpec object for a mark's effects list.

See Also

glow(), shadow(), outline()

Examples

```
df <- data.frame(x = 1:20, y = cumsum(rnorm(20)))
vplot(df) |> mark_line(x = x, y = y, effects = list(motion(x = 4)))

vplot(mtcars) |>
  mark_point(x = wt, y = mpg, size = 4, effects = list(echo(x = 5)))
```

---

outline *Outline (halo) layer effect*

Description

Draws one opaque, wider copy of a stroked or point mark beneath the crisp original in a contrasting colour, so the mark stays legible over a busy or dark backdrop (the "sticker" look). Applies to the same marks as glow().

Usage

```
outline(size = 1, color = "white", alpha = 1)
```

Arguments

size	Halo width per side, in millimetres.
color	Outline colour.
alpha	Outline opacity.

Value

An OutlineSpec object for a mark's effects list.

See Also

glow(), shadow()

Examples

```
vplot(mtcars) |>
  mark_point(x = wt, y = mpg, color = factor(cyl), size = 3,
    effects = list(outline()))
```

---

pattern_hatch *Crisp vector hatch fill*

Description

pattern_hatch() is the *vector* companion of the tile patterns above: evenly spaced parallel lines produced as real geometry via vellum::vl_hatch() rather than a rasterised tile. Because it is vector, it stays crisp at any magnification and in PDF (no embedded image), which makes it the better redundant, non-colour encoding for accessible and print output — the "fix it" step alongside plot_lint() and render_plot(cvd =).

Usage

```
pattern_hatch(
  color = "grey20",
  bg = NA,
  angle = 45,
  spacing = 3,
  linewidth = 0.75
)
```

Arguments

color	Line colour (any R colour). Default "grey20".
bg	Background fill painted behind the lines, or NA (default) for a transparent field so what is under the shape shows through.
angle	Line orientation in degrees. Any angle is allowed (unlike the tile patterns, vector hatch has no seamless-tiling restriction). Default 45.
spacing	Distance between lines, in millimetres. Default 3.
linewidth	Line width. Default 0.75.

Details

Use it exactly like the other pattern constructors: as a constant fill on a filled mark, or as one of the values of scale_pattern() to map a discrete variable to hatch textures.

Value

A `vellum_hatch` object, usable as a `fill` value.

See Also

patterns, `scale_pattern()`, `plot_lint()`, `vellum::vl_hatch()`

Examples

```
df <- data.frame(g = c("a", "b", "c"), n = c(3, 5, 2))
vplot(df) |> mark_bar(x = g, y = n, fill = pattern_hatch(angle = 30))
```

`patterns` *Pattern (hatch) fills*

Description

Build a tiling **pattern** to use as a constant `fill` value on a filled mark, the texture counterpart of `linear_gradient()`. Distinguishing regions by texture (not only hue) keeps a plot legible in greyscale print and under colour-vision deficiency. Each constructor assembles a small tile and returns a `vellum_pattern` (via `vellum::vl_pattern()`); pass it straight to `fill`:

Usage

```
pattern_stripe(
  color = "grey20",
  bg = NA,
  angle = 45,
  spacing = 4,
  linewidth = 1,
  units = "mm"
)

pattern_crosshatch(
  color = "grey20",
  bg = NA,
  angle = 45,
  spacing = 4,
  linewidth = 1,
  units = "mm"
)

pattern_grid(
  color = "grey20",
  bg = NA,
  spacing = 4,
  linewidth = 1,
```

```
  units = "mm"
)

pattern_dot(color = "grey20", bg = NA, spacing = 4, size = 1.5, units = "mm")

pattern_checker(color = "grey20", bg = NA, size = 4, units = "mm")
```

Arguments

color	Line/dot colour (any R colour). Default "grey20".
bg	Background fill painted behind the motif, or NA (default) for a transparent tile so whatever is under the shape shows through.
angle	Stripe orientation in degrees, restricted to 0, 45, 90, or 135 (the orientations that tile seamlessly). Other values error.
spacing	Distance between repeats, in units. Default 4.
linewidth	Stripe / grid line width (as elsewhere in the package). Default 1.
units	Length unit for spacing/size: "mm" (default), "in", "pt", "npc", or "native".
size	For pattern_dot(), the dot diameter in units; for pattern_checker(), the square size in units.

Details

```
vplot(df) |> mark_bar(x = g, y = n, fill = pattern_stripe(angle = 45))
```

A pattern is an unscaled *value*, not a data-mapped channel (like a gradient). It applies to any filled mark: bars, areas, ribbons, rects, tiles, boxplots, violins, ridgelines, half-eyes, hulls/ellipses, and sf polygons. To vary the texture *by a variable*, map the pattern aesthetic instead (a discrete scale).

Patterns render on every backend (PNG, SVG, and — since the tile embeds as an image — PDF). The tile is rasterised at the scene resolution, so at extreme magnification it can soften; at ordinary sizes it is crisp.

Value

A vellum_pattern object, usable as a fill value.

See Also

linear_gradient(), vellum::vl_pattern()

Examples

```
df <- data.frame(g = c("a", "b", "c"), n = c(3, 5, 2))
vplot(df) |> mark_bar(x = g, y = n, fill = pattern_crosshatch())
```

`pdf_pages` *Write a multi-page PDF*

Description

`pdf_pages()` writes several plots into one PDF, one plot per page — a report, a slide deck, or one page per facet. It is the multi-page companion of `render_plot()` (which writes a single page). Pages may differ in size (each plot keeps its own `width`/`height`), and the per-page accessibility tags (structure tree + `Alt`, see the *Accessibility* article) are written for every page.

Usage

```
pdf_pages(x, path)
```

Arguments

`x`	Either a **list** of plots (each a PlotSpec, composition, or table) — one page each — or a **single faceted** `PlotSpec`, in which case it is split into one page per facet cell (the facet is dropped and the data filtered per page; each page trains its own scales).
`path`	Output `.pdf` path.

Value

`path`, invisibly.

See Also

`render_plot()`, `render_all()`

Examples

```
p1 <- vplot(mtcars) |> mark_point(x = wt, y = mpg)
p2 <- vplot(mtcars) |> mark_histogram(x = mpg, bins = 8)
f <- tempfile(fileext = ".pdf")
pdf_pages(list(p1, p2), f)

# one page per facet cell:
faceted <- vplot(mtcars) |> mark_point(x = wt, y = mpg) |> facet_wrap(~cyl)
pdf_pages(faceted, tempfile(fileext = ".pdf"))
```

`plot_alt` *Text alternative (alt text) for a plot*

Description

Returns the description used as the plot's text alternative for assistive technology. If `labs()` set an explicit `alt`, that string is returned verbatim; otherwise vellumplot generates a prose summary from the specification — the chart type, what each axis / colour / size encodes, the number of observations, and any faceting.

Usage

```
plot_alt(x)
```

Arguments

`x`	A PlotSpec or a plot composition.

Details

`render_plot()` (and any compile through `vellum::as_vellum_scene()`) passes this text to the scene's description, so the exported **SVG** carries it in `<desc>` (with `role="img"`) and the exported **PDF** tags the chart as a `Figure` whose `Alt` is this text. The plot **title** (from `labs(title=)`) becomes the accessible name. See the *Accessibility* article.

Value

A single string.

See Also

`labs()`, `vellum::describe()`

Examples

```
p <- vplot(mtcars) |> mark_point(x = wt, y = mpg, color = hp)
plot_alt(p)
plot_alt(labs(p, alt = "Heavier cars get fewer miles per gallon."))
```

plot_data_uri *Encode a plot as a data URI*

Description

plot_data_uri() renders a plot to a self-contained data: **URI** string — the bytes of the image inlined as base64 — ready to drop into an HTML <img src>, a Markdown image, an email, or anywhere a URL is expected without a separate file. "svg" (default) inlines the crisp vector SVG; "png" inlines a raster render.

Usage

```
plot_data_uri(
  plot,
  format = c("svg", "png"),
  width = NULL,
  height = NULL,
  dpi = NULL,
  ...
)
```

Arguments

plot	A PlotSpec (or any object plot_svg() / render_plot() accepts).
format	"svg" (vector, default) or "png" (raster).
width, height	Optional size override in inches.
dpi	For format = "png", the render resolution.
...	Passed to plot_svg() (svg) or render_plot() (png).

Value

A length-1 character data: URI.

See Also

plot_svg(), render_plot()

Examples

```
uri <- vplot(mtcars) |> mark_point(x = wt, y = mpg) |> plot_data_uri()
substr(uri, 1, 30)
```

plot_lint *Lint a plot for legibility and accessibility problems*

Description

`plot_lint()` compiles a plot and reports the design problems a static render hides from a green test suite: text too small to read, colour contrast below the WCAG threshold, two palette colours a colour-blind reader cannot tell apart, labels that overlap or fall off the panel, and grammar-level mistakes such as an encoding with a single level or a legend too long to read. It is the "flag it" step of the accessibility workflow — pair it with `render_plot(cvd = )` to *see* a failing palette and `scale_pattern()` / `pattern_hatch()` to *fix* it with a redundant non-colour encoding.

Usage

```
plot_lint(x, ..., min_text_px = 7, min_contrast = 3)
```

Arguments

`x`	A PlotSpec (or anything `vellum::as_vellum_scene()` accepts).
`...`	Passed to `vellum::vl_lint()`: `rules`, `exclude`, `severity`, `cvd`, `min_text_pt`, `max_overplot` and the rest of the thresholds.
`min_text_px`	Minimum legible text height in pixels (default `7`); text below it is flagged.
`min_contrast`	Minimum acceptable contrast ratio between a mark and its background (default `3`, the WCAG AA threshold for graphical objects).

Details

The geometric rules come from the engine's `vellum::vl_lint()`, which judges them in resolved device pixels. The grammar rules are registered into the same registry by vellumplot, so they are not special: `vellum::vl_lint_rules()` lists them, `rules =` selects them, and `vellum::vl_lint()` on a compiled plot reports them too. Findings are returned most-severe first.

Because every finding carries the box it refers to, `vellum::vl_lint_overlay()` can draw the report onto the plot, and `vellum::vl_lint_assert()` can fail a test on it.

A composition or table has no single set of trained scales, so it reports the geometric findings only — lint the cells individually for the grammar ones (vellumplot#147).

Value

A data frame (class `vellum_lint`) with one row per finding: `rule`, `severity` (`"warning"`/`"note"`), `node`, a human `message`, and the device-px box `x0`/`y0`/`x1`/`y1` — `NA` for a grammar finding, which is about a scale and so has no box. Zero rows when the plot is clean.

See Also

`render_plot()` (`cvd =`), `scale_pattern()`, `pattern_hatch()`, `vellum::vl_lint()`, `vellum::vl_lint_overlay()`, `vellum::vl_lint_assert()`

Examples

```
# a tiny-text, single-level-scale plot trips the linter
p <- vplot(transform(mtcars, grp = "one group")) |>
  mark_point(x = wt, y = mpg, color = grp) |>
  theme(axis.text = element_text(size = 2))
plot_lint(p)

# the engine's arguments come through, so a project can accept a finding
plot_lint(p, exclude = "scale:color")
```

`plot_manifest` *A reproducibility manifest for a plot*

Description

`plot_manifest()` returns a small, serializable fingerprint of a plot: a hash of its input data (order- and column-sensitive), the data's shape and column names, a structural hash of the spec, and the number of emitted elements. It is what `plot_svg()` can embed (with `manifest = TRUE`) so a figure carries its own provenance, and what `plot_verify()` recomputes to confirm a figure still matches its data.

Usage

```
plot_manifest(x)
```

Arguments

x	A PlotSpec.

Value

A named list: `version`, `data` (a `hash`/`nrow`/`columns` record), `spec_hash`, `n_elements`, and `fonts` (the faces the plot resolved to, via `vellum::font_pin()`, each a `path`/`index`/`glyphs` record). The fonts are what lets `plot_verify()` tell a font-stack difference apart from a data change — the same pixels are only guaranteed when the same fonts are present.

See Also

`plot_verify()`, `plot_svg()`, `provenance_join()`

Examples

```
plot_manifest(vplot(mtcars) |> mark_point(x = wt, y = mpg))$data$hash
```

`plot_provenance` *Inspect the compiled-scene provenance of a plot*

Description

Compiles `x` and returns its *provenance table*: one record per emitted mark grob, tying each low-level primitive back to the data rows and trained scales that produced it. Each record's `id` matches the grob's `data-vellum-id` in the SVG output (and the `id` column of `vellum::scene_model()`), so it is a stable join key between the rendered scene and the grammar — the substrate for interactivity, linked views, and accessibility.

Usage

```
plot_provenance(x)
```

Arguments

`x`	A PlotSpec or plot composition.

Details

Each record is a plain, serializable list:

`id` stable node id, equal to `grob@id` / SVG `data-vellum-id` (join key).

`layer` 1-based layer index within the (sub-)plot spec.

`mark` mark type (`"point"`, `"line"`, ...).

`kind` `"mark"` (the core layer) or `"effect"` (a decorative underlay).

`panel` facet panel key (`"panel-r-c"`), or `NA` for a single panel.

`channels` aesthetic → trained-scale reference (`scale`, `type`, `domain`).

`rows` the original input-data row indices this grob draws.

The `rows` field is refined per element for marks that map rows one-to-one to drawn elements (points, bars, tiles, segments, lines, areas, ribbons, steps, text, boxplots, sf features, edges); for aggregating marks (histogram, density, smooth, datashade, ...) it is the whole layer, since rows no longer map to single elements.

Value

A list of provenance records (see Details). Empty for a plot that emits no mark grobs.

See Also

`vellum::scene_model()` and the scene-contract vignette in vellum.

Examples

```
df <- data.frame(wt = mtcars$wt, mpg = mtcars$mpg, model = rownames(mtcars))
p <- vplot(df) |> mark_point(x = wt, y = mpg, data_id = model)
prov <- plot_provenance(p)
prov[[1]]$id
```

---

plot_spacer *Reserve an empty cell in a composition*

Description

Use inside `concat()` / `wrap_plots()` (or a `design`) to leave a gap.

Usage

```
plot_spacer()
```

Value

A spacer placeholder.

Examples

```
a <- vplot(mtcars) |> mark_point(x = wt, y = mpg)
concat(a, plot_spacer(), a, plot_spacer(), ncol = 2)
```

---

plot_svg *Render a plot to a self-contained SVG string*

Description

`plot_svg()` compiles any vellumplot object (a PlotSpec, `vtable()`, or a composition) to a standalone `<svg>` **string** rather than a file — the way to embed a chart *inline* in HTML: a `vsparkline()` in a `gt` / `reactable` / `DT` cell, a chart in a Quarto callout, or an inline glyph in a report. The string carries a `viewBox`, so it scales.

Usage

```
plot_svg(
  plot,
  width = NULL,
  height = NULL,
  scaling = c("fixed", "fit"),
  inline = FALSE,
  recolor = NULL,
  text = "native",
  manifest = FALSE
)
```

Arguments

plot	A PlotSpec, vtable(), or composition.
width, height	Optional page-size override in **inches** (defaults to the object's own size).
scaling	"fixed" (default) keeps the pixel size; "fit" sets the root <svg> width/height to 100% so it fills its container (the viewBox preserves the aspect).
inline	TRUE strips the leading <?xml . . . ?> prolog so the <svg> is a valid fragment to drop straight into an HTML paragraph / cell / Quarto inline expression. FALSE (default) keeps the stand-alone-file prolog.
recolor	Optional named character vector mapping a **source colour** to a replacement written verbatim into the SVG — e.g. c(grey30 = "currentColor") makes a grey30 sparkline follow the surrounding text colour (dark-mode-adaptive inline). Each name is resolved to its hex and swapped for the value; use for CSS keywords ("currentColor") or var(--x) that R's colour engine can't emit itself.
text	How glyphs are emitted: "native" (default) or "outline" (paths, for maximum portability).
manifest	If TRUE, embed a reproducibility manifest (see plot_manifest()) as an XML comment in the SVG, so the figure carries its own data fingerprint; plot_verify() reads it back. Default FALSE.

Value

A length-1 character SVG string.

See Also

vsparkline(), gt_vsparkline(), render_plot(), plot_manifest()

Examples

```
svg <- plot_svg(vsparkline(cumsum(rnorm(20))))
substr(svg, 1, 40)
# a transparent, prolog-free, dark-mode-adaptive inline sparkline:
plot_svg(vsparkline(1:9, color = "grey30"),
  inline = TRUE, recolor = c(grey30 = "currentColor")
)
```

---

plot_verify *Verify a rendered figure against its data*

Description

plot_verify() extracts the manifest embedded in an SVG written with plot_svg(manifest = TRUE) and recomputes the data fingerprint from data, reporting whether the figure still matches the data it was drawn from — a lightweight, self-contained reproducibility check.

Usage

```
plot_verify(svg, data)
```

Arguments

svg	An SVG string (from plot_svg()) or a path to an .svg file.
data	The data frame to check the figure against.

Details

A font mismatch is reported as a **distinct outcome** from a data mismatch: data_ok is whether the data still hashes the same, fonts_ok whether every font the figure was drawn with is still present on this machine, and ok their conjunction. A pixel difference with data_ok = TRUE but fonts_ok = FALSE is the font stack, not your data — a different cause with a different fix (install/register the font) than a data change.

Value

A list with ok (data *and* fonts match), data_ok, expected / actual (the embedded vs recomputed data hash), fonts_ok, and fonts_missing (the recorded font paths absent on this machine). Errors if the SVG carries no manifest.

See Also

plot_manifest(), plot_svg()

Examples

```
svg <- plot_svg(vplot(mtcars) |> mark_point(x = wt, y = mpg), manifest = TRUE)
plot_verify(svg, mtcars)$ok
```

PlotSpec *The plot specification*

Description

PlotSpec is the S7 class that vplot() creates and the mark_*() / scale_*() functions extend. It is a plain, inspectable, serializable data object: data, a list of layers, a list of scale overrides, and the page size. Nothing is drawn until it is compiled with vellum::as_vellum_scene() (e.g. via render_plot()). Printing it draws the plot; inspect its structure with base::summary().

Usage

```
PlotSpec(
  data = NULL,
  edge_data = NULL,
  layers = list(),
  scales = list(),
  facet = NULL,
  coord = NULL,
  resolve = list(),
  width = 6,
  height = 4,
  dpi = 96,
  theme = NULL,
  labels = list(),
  marginal = NULL,
  selections = list(),
  filters = list(),
  binds = list(),
  clip = NULL,
  transition = NULL,
  ease = NULL,
  source = NULL
)
```

Arguments

`data`	The data frame.
`edge_data`	For a graph plot (from `vgraph()`), the edge table; the default data for `mark_edges()`. `NULL` for ordinary plots.
`layers`	A list of layer specifications (one per `mark_*()`).
`scales`	A list of declared scale overrides.
`facet`	A faceting specification (from `facet_wrap()` / `facet_grid()`), or `NULL` for a single panel.
`coord`	A coordinate specification (from `coord_cartesian()` / `coord_flip()` / `coord_fixed()`), or `NULL` for the default Cartesian system.
`resolve`	A named list mapping an aesthetic to `"shared"` or `"independent"` (the scale-resolution lattice; see `resolve_scale()`).
`width, height`	Page size in inches.
`dpi`	Output resolution in dots per inch (pixels per inch). The exported PNG's pixel dimensions are `width * dpi` by `height * dpi`.
`theme`	A theme (a named list of resolved element/setting overrides, from `theme()` / a `theme_*()` preset), or `NULL` for the default theme.
`labels`	A named list of plot/axis/legend label overrides (see `labs()`).
`marginal`	A marginal-distribution specification (from `add_marginal()`), or `NULL` for no marginals.

selections	A list of interactive selection declarations (from select_point() / select_interval() / add_selection()).
filters	A list of filter declarations (from filter_by()).
binds	A list of scale-domain bind declarations (from bind_scale()).
clip	A clip_to() / set_mask() geometry clip (a ClipSpec), or NULL for none.
transition	A keyframe-animation transition (from transition_states() / transition_time() / transition_reveal()), or NULL. Read only by animate(); inert on a static render.
ease	An animation easing (from ease_aes()), or NULL.
source	A source-inspection declaration (from inspect_source()), or NULL. Inert on a static render; a host surfaces the data rows behind a clicked element.

Value

A PlotSpec.

See Also

vplot(), mark_point(), scale_x_continuous()

---

position *Position adjustments*

Description

Fine-grained control over how a mark's overlapping elements are placed. Pass the result to a mark's position argument (e.g. mark_point(position = position_jitter(width = 0.2))). A bare string (position = "dodge") still works and uses the defaults.

Usage

```
position_nudge(x = 0, y = 0)

position_jitter(width = NULL, height = NULL, seed = NULL)

position_dodge(width = NULL)

position_dodge2(padding = 0.1)

position_sina(width = 0.8, seed = NULL)

position_jitterdodge(
  jitter.width = NULL,
  jitter.height = 0,
  dodge.width = 0.75,
  seed = NULL
)
```

Arguments

x, y	position_nudge() shift (data units).
width, height	Maximum jitter (data units); NULL uses the default. For position_sina(), width is the spread as a fraction of the band.
seed	Optional integer seed for the random jitter.
padding	position_dodge2() gap between dodged elements, as a fraction of the element width.
dodge.width, jitter.width, jitter.height	position_jitterdodge() dodge / jitter extents.

Details

- position_nudge() shifts every element by a constant amount in **data** units (for continuous axes) — handy for offsetting labels from points.
- position_jitter() adds uniform random noise; width/height are the maximum shift in data units (default: 40% of the resolution), seed makes it reproducible.
- position_dodge() places grouped elements side by side; width is the total data-space width shared by the group (default: the category band).
- position_dodge2() dodges by the groups actually present at each x and splits the band between them with a padding gap — so ragged groupings stay centred and evenly spaced.
- position_jitterdodge() dodges grouped elements, then jitters within each dodged slot (points over dodged boxes).
- position_sina() spreads each category's points along x by a quasirandom offset scaled to the local y-density, so the cloud traces the distribution's shape (ggforce's sina). width is the maximum spread as a fraction of the category band (default 0.8).

Value

A vellumplot_position object for a mark's position argument.

Examples

```
vplot(mtcars) |>
  mark_point(x = factor(cyl), y = mpg, position = position_jitter(width = 0.15))
```

---

provenance_join *Join a plot's provenance to its rendered geometry*

Description

provenance_join() compiles x once and returns a tidy table with one row per emitted mark element, tying each drawn primitive to **both** the source data rows that produced it (rows, a list column) **and** its device-pixel geometry (x0, y0, x1, y1, x, y, w, h) from vellum::scene_model(). It is the consumer of the provenance table that plot_provenance() emits — the substrate for click-to-source interactivity, auditing "which rows made this element?", and linked views.

Usage

```
provenance_join(x)
```

Arguments

x	A PlotSpec or composition.

Value

A data frame: id, layer, mark, kind, panel, the pixel bbox (x0, y0, x1, y1, x, y, w, h), n_rows, and rows (a list column of integer source-row indices). Empty for a plot with no mark grobs.

See Also

plot_provenance(), vellum::scene_model()

Examples

```
df <- data.frame(wt = mtcars$wt, mpg = mtcars$mpg, cyl = factor(mtcars$cyl))
p <- vplot(df) |> mark_point(x = wt, y = mpg, color = cyl)
pj <- provenance_join(p)
pj[, c("id", "mark", "n_rows")]
```

---

provenance_payload *A widget-ready provenance payload (click-to-source)*

Description

provenance_payload() returns a JSON-serializable structure mapping each rendered element's stable id (data-vellum-id, the join key an SVG / widget carries) to the source data rows that produced it — the enabler for a host (e.g. vellumwidget) to answer "which rows made this element?" on click, without re-running the grammar. With values = TRUE the referenced data rows are inlined so the host can display them directly.

Usage

```
provenance_payload(x, values = FALSE)
```

Arguments

x	A PlotSpec or composition.
values	If TRUE, inline the source data rows (row-wise records) so the host needs no separate data access. Default FALSE (ids + row indices only).

Value

A list with fields (column names), elements (a list of list(id, rows) records), and — when values = TRUE — data (row-wise records of the plot's data).

See Also

provenance_join(), plot_provenance()

Examples

```
p <- vplot(mtcars) |> mark_point(x = wt, y = mpg, color = factor(cyl))
pl <- provenance_payload(p)
pl$elements[[1]]$id
```

---

render_all *Render many plots to separate files, in parallel*

Description

render_all() renders a list of independent plots to separate files across CPU cores — small multiples, a batch export, one file per group. The work is embarrassingly parallel (one whole plot per worker), and the result is byte-identical to rendering them one by one. Parallelism uses process forks, so it speeds things up on macOS/Linux and falls back to sequential on Windows; it is only worth it for several substantial plots.

Usage

```
render_all(plots, paths, workers = NULL, ...)
```

Arguments

plots	A named or unnamed list of plots.
paths	Output paths, one per plot (the format of each comes from its extension, as in render_plot()). As a shortcut, when plots is **named** paths may be a single existing **directory**, and each plot is written to <name>.png inside it.
workers	Number of parallel workers; NULL (default) uses the available cores. 1 forces sequential.
...	Passed to each vellum::render() (e.g. text, cvd).

Value

paths (the resolved file paths), invisibly.

See Also

render_plot(), pdf_pages(), repeat_()

Examples

```
plots <- list(
  wt = vplot(mtcars) |> mark_point(x = wt, y = mpg),
  hp = vplot(mtcars) |> mark_point(x = hp, y = mpg)
)
render_all(plots, tempdir())
```

render_plot *Render a plot to a file*

Description

Compiles a PlotSpec into a `vellum::vl_scene()` and writes it. The output format is taken from the file extension (`.png`, `.svg`, `.pdf`). `vellum::render()` also works on a plot directly, dispatching through the `as_vellum_scene()` seam.

Usage

```
render_plot(plot, path, text = "native", dpi = NULL, cvd = "none")
```

Arguments

`plot`	A PlotSpec.
`path`	Output file path.
`text`	For SVG output, how text is written (see `vellum::render()`).
`dpi`	Output resolution in dots per inch. `NULL` (default) uses the plot's authored resolution (set on `vplot()`); a number overrides it for this render, so the PNG's pixel dimensions become `width * dpi` by `height * dpi`. Ignored for `.svg`/`.pdf`, which are resolution-independent.
`cvd`	Colour-vision-deficiency simulation for a `.png` render: one of `"none"` (default), `"protanopia"`, `"deuteranopia"`, `"tritanopia"`, or `"achromatopsia"`. Use it to preview how a palette reads for a colour-blind viewer (pair with `plot_lint()`, which flags low contrast, and `scale_pattern()` / `pattern_hatch()`, which add a redundant non-colour encoding). Simulation is raster-only; it is ignored for `.svg`/`.pdf`.

Value

`path`, invisibly.

Examples

```
p <- vplot(mtcars) |> mark_point(x = wt, y = mpg)
f <- tempfile(fileext = ".png")
render_plot(p, f)
render_plot(p, f, dpi = 300) # denser raster, same physical size
render_plot(p, f, cvd = "deuteranopia") # preview for red-green colour blindness
```

repeat_ *Repeat a view across fields*

Description

Replicate a plot, re-pointing one or more encodings at a different data field each time, and arrange the copies as a composition (like `concat()`, so each sub-plot keeps its own scales and axes). Supply each aesthetic as a character vector of column names; all vectors must be the same length `N`, and are zipped to produce `N` sub-plots.

Usage

```
repeat_(plot, ..., ncol = NULL, nrow = NULL)
```

Arguments

`plot`	A PlotSpec; the repeated aesthetic(s) are set on **every** layer (added if not already mapped). In a multi-layer plot every layer therefore receives the re-pointed field, so any layer carrying its own `data` must contain the named field columns.
`...`	Named aesthetics, each a character vector of field names, e.g. `x = c("wt", "hp", "disp")`.
`ncol, nrow`	Grid dimensions (passed to `concat()`).

Value

A `PlotComposition`.

Examples

```
repeat_(vplot(mtcars) |> mark_point(y = mpg), x = c("wt", "hp", "disp"))
```

resolve_scale *Resolve scales as shared or independent across panels*

Description

The scale-resolution lattice (after Vega-Lite): for each aesthetic, choose whether faceted panels share one trained scale or train independent ones. Position scales default to `"shared"`; `facet_*(scales = "free_*")` is sugar for setting `x`/`y` to `"independent"`.

Usage

```
resolve_scale(plot, ...)
```

Arguments

plot	A PlotSpec.
...	Named arguments mapping an aesthetic (x, y, ...) to "shared" or "independent".

Value

The modified PlotSpec.

Examples

```
vplot(mtcars) |> mark_point(x = wt, y = mpg) |> facet_wrap(~cyl) |>
  resolve_scale(y = "independent")
```

---

scale_alpha *Alpha (opacity) scale*

Description

Declare the scale for a mapped alpha aesthetic: data values map to an opacity range in [0, 1]. An alpha legend is drawn automatically.

Usage

```
scale_alpha(plot, range = NULL, limits = NULL, breaks = NULL, name = NULL)

scale_alpha_continuous(
  plot,
  range = NULL,
  limits = NULL,
  breaks = NULL,
  name = NULL
)
```

Arguments

plot	A PlotSpec.
range	Numeric length-2 output opacity range, or NULL for the default c(0.1, 1).
limits	Numeric length-2 data domain, or NULL to train from the data.
breaks	Explicit legend breaks, or NULL.
name	Legend title, or NULL to derive from the encoding.

Value

The modified PlotSpec.

Examples

```
vplot(mtcars) |> mark_point(x = wt, y = mpg, alpha = hp) |> scale_alpha(range = c(0.2, 1))
```

scale_color_continuous *Colour scales*

Description

Declare a colour scale for the `color`/`fill` channel. Continuous data get a perceptual ramp; discrete data get a qualitative palette. `scale_*_manual()` sets exact colours, `scale_*_gradient()` a two-point ramp, and `scale_*_gradient2()` a diverging three-point ramp (`low`–`mid`–`high`) centred on `midpoint` — for signed or anomaly data where a meaningful zero should sit at the neutral colour. The `fill` variants are identical (colour and fill share one scale). A legend is drawn automatically when colour is mapped.

Usage

```
scale_color_continuous(
  plot,
  palette = NULL,
  limits = NULL,
  breaks = NULL,
  labels = NULL,
  name = NULL
)

scale_color_discrete(
  plot,
  palette = NULL,
  breaks = NULL,
  labels = NULL,
  name = NULL
)

scale_color_manual(plot, values, name = NULL)

scale_color_gradient(plot, low = "#132B43", high = "#56B1F7", name = NULL)

scale_color_gradient2(
  plot,
  low = "#832424",
  mid = "#FFFFFF",
  high = "#3A3A98",
  midpoint = 0,
  name = NULL
)

scale_fill_continuous(
  plot,
```

```
  palette = NULL,
  limits = NULL,
  breaks = NULL,
  labels = NULL,
  name = NULL
)

scale_color_gradientn(plot, colours, name = NULL)

scale_fill_gradientn(plot, colours, name = NULL)

scale_colour_gradientn(plot, colours, name = NULL)

scale_fill_discrete(
  plot,
  palette = NULL,
  breaks = NULL,
  labels = NULL,
  name = NULL
)

scale_fill_manual(plot, values, name = NULL)

scale_fill_gradient(plot, low = "#132B43", high = "#56B1F7", name = NULL)

scale_fill_gradient2(
  plot,
  low = "#832424",
  mid = "#FFFFFF",
  high = "#3A3A98",
  midpoint = 0,
  name = NULL
)

scale_colour_continuous(
  plot,
  palette = NULL,
  limits = NULL,
  breaks = NULL,
  labels = NULL,
  name = NULL
)

scale_colour_discrete(
  plot,
  palette = NULL,
  breaks = NULL,
  labels = NULL,
```

```
  name = NULL
)

scale_colour_manual(plot, values, name = NULL)

scale_colour_gradient(plot, low = "#132B43", high = "#56B1F7", name = NULL)

scale_colour_gradient2(
  plot,
  low = "#832424",
  mid = "#FFFFFF",
  high = "#3A3A98",
  midpoint = 0,
  name = NULL
)
```

Arguments

plot	A PlotSpec.
palette	A vector of colours, or a single palette name passed to grDevices::hcl.colors() (e.g. "Batlow", "Blues", "Set 2"; matched case/space-insensitively). NULL uses a sensible default.
limits	For the continuous colour scales, a length-2 numeric c(min, max) fixing the mapped data range (the direct form of lims(color = ...)).
breaks, labels	Explicit legend breaks / labels, or NULL.
name	Legend title, or NULL to derive from the encoding.
values	For scale_*_manual(), a vector of colours; if named, matched to data levels by name (unmatched levels draw grey).
low, high	For scale_*_gradient()/scale_*_gradient2(), the endpoint colours.
mid	For scale_*_gradient2(), the midpoint colour.
midpoint	For scale_*_gradient2(), the data value placed at mid (default 0); values above and below diverge to high and low.
colours	For scale_*_gradientn(), an n-stop vector of colours the continuous ramp interpolates through.

Details

Continuous and binned ramps built from a plain colour vector are interpolated in the perceptually-uniform **Oklab** space, so they avoid the muddy, over-dark midtones and hue drift of sRGB blending. Designed perceptual palettes (the default, and hcl.colors() names) are already uniform and unaffected. Set options(vellumplot.color.interpolation = "srgb") (or "lab") to change the blend space globally.

Value

The modified PlotSpec.

Examples

```
vplot(mtcars) |>
  mark_point(x = wt, y = mpg, color = hp) |>
  scale_color_continuous(palette = "Batlow")
```

---

scale_color_identity *Identity scales*

Description

Use the data values *directly* as the aesthetic — no training and no legend. The mapped column must already contain valid aesthetic values: colours for `scale_color_identity()` / `scale_fill_identity()`, sizes in mm for `scale_size_identity()`, and shape names for `scale_shape_identity()`. Useful when a column already holds the exact colours/sizes you want to draw.

Usage

```
scale_color_identity(plot)

scale_fill_identity(plot)

scale_colour_identity(plot)

scale_size_identity(plot)

scale_shape_identity(plot)

scale_alpha_identity(plot)

scale_linetype_identity(plot)
```

Arguments

`plot`	A PlotSpec.

Value

The modified PlotSpec.

Examples

```
df <- data.frame(x = 1:3, y = 1:3, col = c("red", "green", "blue"))
vplot(df) |> mark_point(x = x, y = y, color = col) |> scale_color_identity()
```

scale_edge *Edge colour / alpha / line-type scales*

Description

Declare the scale for a mapped edge aesthetic on a vgraph() plot – the colour (mark_edges(color =)), opacity (alpha =), or line type (linetype =) of the edges. These are the edge counterparts of scale_color_continuous() / scale_alpha() / scale_linetype(): an edge aesthetic is trained and legended **independently of the node scales**, so a figure can map, say, node fill to a discrete community *and* edge colour to a continuous weight without the two collapsing into one legend. Each draws its own legend automatically.

Usage

```
scale_edge_color(
  plot,
  palette = NULL,
  breaks = NULL,
  labels = NULL,
  name = NULL,
  midpoint = NULL
)

scale_edge_colour(
  plot,
  palette = NULL,
  breaks = NULL,
  labels = NULL,
  name = NULL,
  midpoint = NULL
)

scale_edge_alpha(plot, range = NULL, limits = NULL, breaks = NULL, name = NULL)

scale_edge_linetype(plot, values = NULL, name = NULL)
```

Arguments

plot	A PlotSpec, normally from vgraph().
palette	For scale_edge_color(), a vector of colours or a single palette name (as in scale_color_continuous()); NULL for a default.
name	Legend title, or NULL to derive from the encoding.
midpoint	For scale_edge_color(), the data value placed at the ramp's midpoint (a diverging scale); NULL for an ordinary ramp.
range	For scale_edge_alpha(), the output opacity range c(min, max), or NULL for the default.

`limits`, `breaks`, `labels`
: Data limits, explicit legend breaks, and explicit labels, or `NULL` to derive from the data.

`values`
: For `scale_edge_linetype()`, a character vector of line types (as in `scale_linetype()`); `NULL` for the default palette.

Details

`scale_edge_color()` infers discrete vs continuous from the mapped data (a factor/character trains a discrete swatch legend; a number a colour bar), the same as the node colour scale. `scale_edge_colour()` is a British-spelling alias.

Value

The modified PlotSpec.

See Also

`mark_edges()`, `scale_edge_width()`, `vgraph()`

Examples

```
## Not run:
g <- igraph::make_graph("Zachary")
g <- igraph::set_edge_attr(g, "w", value = runif(igraph::ecount(g)))
vgraph(g) |>
  mark_edges(color = w) |>
  mark_nodes(fill = factor(igraph::membership(igraph::cluster_louvain(g)))) |>
  scale_edge_color(palette = "Grays")

## End(Not run)
```

---

scale_edge_width *Edge-width scale*

Description

Declare the scale for a mapped edge `linewidth` aesthetic (e.g. `mark_edges(linewidth = weight)` on a `vgraph()` plot). The data range is rescaled to a line-width range and an edge-width legend is drawn automatically.

Usage

```
scale_edge_width(plot, range = NULL, limits = NULL, breaks = NULL, name = NULL)
```

Arguments

plot	A PlotSpec.
range	Output line-width range c(min, max), or NULL for the default c(0.3, 3).
limits	Data limits c(min, max), or NULL to train from the data.
breaks	Explicit legend breaks, or NULL.
name	Legend title, or NULL to derive from the encoding.

Value

The modified PlotSpec.

See Also

mark_edges(), vgraph()

Examples

```
## Not run:
g <- igraph::sample_gnp(20, 0.2)
g <- igraph::set_edge_attr(g, "w", value = runif(igraph::ecount(g)))
vgraph(g) |> mark_edges(linewidth = w) |> mark_nodes() |> scale_edge_width(range = c(0.3, 4))

## End(Not run)
```

---

scale_fill_binned *Binned (classed) colour scales*

Description

scale_fill_binned() / scale_color_binned() cut a continuous aesthetic into classes and colour each class with one step of a sequential palette — the standard choropleth scale. Class breaks are chosen by a classification style. quantile, equal, and pretty need no extra packages; other styles (e.g. "jenks", "fisher", "kmeans", "headtails") delegate to the optional classInt package (in Suggests) and error if it is absent.

Usage

```
scale_fill_binned(
  plot,
  style = "quantile",
  n = 5,
  breaks = NULL,
  palette = NULL,
  labels = NULL,
  na_value = "grey80",
  name = NULL
```

```
)

scale_color_binned(
  plot,
  style = "quantile",
  n = 5,
  breaks = NULL,
  palette = NULL,
  labels = NULL,
  na_value = "grey80",
  name = NULL
)

scale_color_steps(
  plot,
  style = "quantile",
  n = 5,
  breaks = NULL,
  palette = NULL,
  labels = NULL,
  na_value = "grey80",
  name = NULL
)

scale_colour_steps(
  plot,
  style = "quantile",
  n = 5,
  breaks = NULL,
  palette = NULL,
  labels = NULL,
  na_value = "grey80",
  name = NULL
)

scale_fill_steps(
  plot,
  style = "quantile",
  n = 5,
  breaks = NULL,
  palette = NULL,
  labels = NULL,
  na_value = "grey80",
  name = NULL
)
```

Arguments

plot	A PlotSpec.

style	Classification style: one of "quantile", "equal", "pretty" (base R), or any classInt::classIntervals() style. Default "quantile".
n	Number of classes (default 5). Ignored when breaks is supplied.
breaks	Explicit class boundaries (length = number of classes + 1), overriding style/n.
palette	A sequential palette: NULL (batlow), an grDevices::hcl.pals() name, or a vector of colours interpolated across the classes.
labels	Optional class labels (one per class); defaults to interval ranges like "[0, 10)".
na_value	Fill colour for NA values (shown as a distinct legend swatch). Default "grey80".
name	Legend title, or NULL to derive from the encoding.

Details

scale_color_steps() / scale_fill_steps() are aliases of the binned colour scales, for ggplot2 users who reach for the _steps name — a continuous variable cut into steps with an interval-key legend.

Value

The modified PlotSpec.

See Also

mark_sf(), scale_fill_continuous()

Examples

```
## Not run:
nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
vplot(nc) |>
  mark_sf(fill = SID74) |>
  scale_fill_binned(style = "quantile", n = 5) |>
  coord_sf()

## End(Not run)
```

---

scale_linetype *Line-type scale*

Description

Declare the scale for a mapped (discrete) linetype aesthetic. Levels cycle through a default set of line types unless values is given. A line-type legend is drawn automatically. Applies to line-like marks (mark_line(), mark_step()).

Usage

```
scale_linetype(plot, values = NULL, name = NULL)

scale_linetype_discrete(plot, values = NULL, name = NULL)
```

Arguments

plot	A PlotSpec.
values	Character vector of line types (each one of "solid", "dashed", "dotted", "dotdash", "longdash", "twodash"), or NULL for the default set.
name	Legend title, or NULL to derive from the encoding.

Value

The modified PlotSpec.

Examples

```
df <- data.frame(x = rep(1:10, 2), y = rnorm(20), g = rep(c("a", "b"), each = 10))
vplot(df) |> mark_line(x = x, y = y, linetype = g)
```

---

scale_linewidth *Line-width scale*

Description

Declare the scale for a mapped linewidth aesthetic on a stroked mark – mark_line(), mark_step(), or mark_segment(). Data values are rescaled linearly onto a line-width range (in lwd units, as everywhere else in the package) and a line-width legend is drawn automatically.

Usage

```
scale_linewidth(plot, range = NULL, limits = NULL, breaks = NULL, name = NULL)

scale_linewidth_continuous(
  plot,
  range = NULL,
  limits = NULL,
  breaks = NULL,
  name = NULL
)
```

Arguments

plot	A PlotSpec.
range	Output line-width range c(min, max), or NULL for the default c(0.5, 4).
limits	Data limits c(min, max), or NULL to train from the data.
breaks	Explicit legend breaks, or NULL.
name	Legend title, or NULL to derive from the encoding.

Details

A mapped linewidth gives each **segment** one constant width, taken as the mean of its two endpoint values. So the width changes in visible steps at each vertex rather than tapering smoothly along the line, and the round joins do not perfectly fill the corners of a sharp bend. That is inherent to per-segment width; a constant linewidth = is unaffected and still draws the line as a single stroke.

No backend can stroke the segments of one path at different widths in a single call, so a mapped linewidth emits one path element per segment in SVG and PDF. On a long, dense line that is a lot of elements – worth knowing before mapping linewidth on tens of thousands of rows.

For edges on a vgraph() plot, use scale_edge_width() instead: an edge's width is trained and legended independently of the line-width scale. That also means guides(linewidth =) and lims(linewidth =) address *this* scale; use guides(edge_width =) / lims(edge_width =) for a graph's edges.

A missing (NA) width takes its segment's width from whichever endpoint is known, so one missing value thins the line rather than erasing the two segments that meet at it. A segment missing a width at both ends is dropped.

Value

The modified PlotSpec.

See Also

mark_line(), mark_step(), mark_segment(), scale_edge_width()

Examples

```
vplot(pressure) |>
  mark_line(x = temperature, y = pressure, linewidth = pressure) |>
  scale_linewidth(range = c(0.5, 5))
```

scale_pattern *Pattern (texture) scale*

Description

Map a discrete variable to fill **textures** – the greyscale- and colour-vision-safe companion of a fill palette – via a filled mark's `pattern` aesthetic: `mark_bar(x = g, y = n, pattern = series)`. Each level takes a distinct `pattern_stripe()`-family texture. Applies to the filled marks (bar, area, tile, rect, boxplot, violin, hull).

Usage

```
scale_pattern(plot, values = NULL, name = NULL)
```

Arguments

`plot`	A PlotSpec.
`values`	Optional textures, one per level: a character vector of builder names (`"stripe"`, `"crosshatch"`, `"grid"`, `"dot"`, `"checker"`), or a list of `vellum_pattern` objects (e.g. from `pattern_stripe()`). `NULL` (default) cycles a built-in palette of distinct textures.
`name`	Optional legend title.

Value

The modified PlotSpec.

See Also

`pattern_stripe()`, `scale_shape()`

Examples

```
df <- data.frame(g = c("a", "b", "c"), n = c(3, 5, 2))
vplot(df) |> mark_bar(x = g, y = n, pattern = g) |> scale_pattern()
```

scale_shape *Shape scale*

Description

Declare the scale for a mapped (discrete) `shape` aesthetic. Levels cycle through a default set of marker shapes unless `values` is given. A shape legend is drawn automatically.

Usage

```
scale_shape(plot, values = NULL, name = NULL)
```

Arguments

plot	A PlotSpec.
values	Character vector of shapes, or NULL for the default. Each is either a built-in marker ("circle", "square", "triangle", "diamond", "plus", "cross", "triangle_down", "star") **or an SVG icon** — a path d string (what icon sets ship) or a path to a .svg file. SVG values are drawn as crisp vector markers (via vellum::svg_grob()) and the legend shows the icon; the size aesthetic scales them.
name	Legend title, or NULL to derive from the encoding.

Value

The modified PlotSpec.

Examples

```
vplot(mtcars) |> mark_point(x = wt, y = mpg, shape = factor(cyl))
# SVG icon markers, one per level:
star <- "M12 2l3 7h7l-5.5 4.5 2 7-6.5-4.5-6.5 4.5 2-7L2 9h7z"
vplot(mtcars) |>
  mark_point(x = wt, y = mpg, shape = factor(cyl)) |>
  scale_shape(values = c(star, star, star))
```

---

scale_size *Size scale*

Description

Declare the scale for a mapped size aesthetic: data values map linearly to a point-size range (in mm). A size legend is drawn automatically.

scale_size_area() maps the data value to the marker's **area** rather than its radius, with value 0 at size 0 — the perceptually honest default for bubble charts (a value twice as large looks twice as big in ink). max_size is the size (mm) of the largest value.

Usage

```
scale_size(plot, range = NULL, limits = NULL, breaks = NULL, name = NULL)

scale_size_area(plot, max_size = 6, limits = NULL, breaks = NULL, name = NULL)
```

Arguments

plot	A PlotSpec.
range	Numeric length-2 output size range in mm, or NULL for the default c(1, 4).
limits	Numeric length-2 data domain, or NULL to train from the data.
breaks	Explicit legend breaks, or NULL.
name	Legend title, or NULL to derive from the encoding.
max_size	For scale_size_area(), the point size (mm) of the largest value.

Value

The modified PlotSpec.

Examples

```
vplot(mtcars) |> mark_point(x = wt, y = mpg, size = hp) |> scale_size(range = c(1, 8))
vplot(mtcars) |>
  mark_point(x = wt, y = mpg, size = hp) |>
  scale_size_area(max_size = 10)
```

---

scale_viridis *Viridis and ColorBrewer scales*

Description

Named convenience constructors for popular palettes, easing migration from ggplot2. They are thin wrappers over the palette = argument of scale_color_continuous() / scale_color_discrete() (any grDevices::hcl.colors() name works there directly).

Usage

```
scale_color_viridis_c(
  plot,
  option = "viridis",
  breaks = NULL,
  labels = NULL,
  name = NULL
)

scale_color_viridis_d(
  plot,
  option = "viridis",
  breaks = NULL,
  labels = NULL,
  name = NULL
)
```

```
scale_fill_viridis_c(
  plot,
  option = "viridis",
  breaks = NULL,
  labels = NULL,
  name = NULL
)

scale_fill_viridis_d(
  plot,
  option = "viridis",
  breaks = NULL,
  labels = NULL,
  name = NULL
)

scale_color_brewer(
  plot,
  palette = "Blues",
  breaks = NULL,
  labels = NULL,
  name = NULL
)

scale_fill_brewer(
  plot,
  palette = "Blues",
  breaks = NULL,
  labels = NULL,
  name = NULL
)

scale_color_distiller(
  plot,
  palette = "Blues",
  breaks = NULL,
  labels = NULL,
  name = NULL
)

scale_fill_distiller(
  plot,
  palette = "Blues",
  breaks = NULL,
  labels = NULL,
  name = NULL
)
```

```
scale_colour_viridis_c(
  plot,
  option = "viridis",
  breaks = NULL,
  labels = NULL,
  name = NULL
)

scale_colour_viridis_d(
  plot,
  option = "viridis",
  breaks = NULL,
  labels = NULL,
  name = NULL
)

scale_colour_brewer(
  plot,
  palette = "Blues",
  breaks = NULL,
  labels = NULL,
  name = NULL
)

scale_colour_distiller(
  plot,
  palette = "Blues",
  breaks = NULL,
  labels = NULL,
  name = NULL
)
```

Arguments

`plot`	A PlotSpec.
`option`	Which viridis map (see above).
`name, breaks, labels`	As for `scale_color_continuous()`.
`palette`	A `grDevices::hcl.colors()` palette name.

Details

- `scale_*_viridis_c()` / `scale_*_viridis_d()` — the perceptually-uniform, colour-vision-deficient-safe viridis maps, continuous (`_c`) or discrete (`_d`). `option` picks the map: `"viridis"`, `"plasma"`, `"inferno"`, `"cividis"`, `"rocket"`, or `"mako"` (unknown options fall back to viridis).
- `scale_*_brewer()` — a discrete ColorBrewer-style qualitative/sequential palette; `scale_*_distiller()` — the same ramp interpolated for continuous data. `palette` is any `hcl.colors()` name (e.g. `"Blues"`, `"Set 2"`).

Value

The modified PlotSpec.

See Also

scale_color_continuous()

Examples

```
vplot(mtcars) |> mark_point(x = wt, y = mpg, color = hp) |> scale_color_viridis_c()
vplot(mtcars) |>
  mark_point(x = wt, y = mpg, color = factor(cyl)) |>
  scale_color_brewer(palette = "Set 2")
```

---

scale_x_binned *Binned position scales*

Description

scale_x_binned() / scale_y_binned() cut a continuous position variable into bins: axis ticks fall at the bin **boundaries** and each datum is drawn at its bin **centre**. Breaks use the same classification as the binned colour scales (style/n; classInt for jenks/fisher/… — a Suggest). Handy to summarise a dense continuous axis, or to place mark_bar() on a binned continuous x.

Usage

```
scale_x_binned(
  plot,
  style = "pretty",
  n = 10,
  breaks = NULL,
  labels = NULL,
  name = NULL
)

scale_y_binned(
  plot,
  style = "pretty",
  n = 10,
  breaks = NULL,
  labels = NULL,
  name = NULL
)
```

Arguments

plot	A PlotSpec.
style	Binning style: "pretty" (default), "equal", "quantile", or a classInt style.
n	Target number of bins.
breaks	Explicit bin boundaries (overrides style/n), or NULL.
labels	Explicit boundary labels, or NULL to format the boundaries.
name	Axis title, or NULL to derive from the encoding.

Value

The modified PlotSpec.

Examples

```
vplot(mtcars) |> mark_point(x = wt, y = mpg) |> scale_x_binned(n = 6)
```

---

scale_x_continuous *Position scales*

Description

Declare a position scale to override the trained default. scale_x_continuous() / scale_y_continuous() handle numeric (and date/time) axes; scale_x_discrete() / scale_y_discrete() handle categorical (band) axes and let you set the level order via limits.

Usage

```
scale_x_continuous(
  plot,
  limits = NULL,
  trans = "identity",
  breaks = NULL,
  labels = NULL,
  name = NULL,
  expand = NULL,
  sec.axis = NULL
)

scale_y_continuous(
  plot,
  limits = NULL,
  trans = "identity",
  breaks = NULL,
  labels = NULL,
  name = NULL,
```

```
  expand = NULL,
  sec.axis = NULL
)

scale_x_discrete(plot, limits = NULL, name = NULL)

scale_y_discrete(plot, limits = NULL, name = NULL)
```

Arguments

plot	A PlotSpec.
limits	For continuous scales a numeric length-2 domain c(min, max); for discrete scales a character vector of levels (sets order / subset). NULL trains from the data.
trans	Transformation: "identity" (default), "log10", "sqrt", "symlog" (symmetric log — linear through zero, logarithmic in the tails, so signed data and zero/negatives read on one axis), "reverse", or a scales::transform_log10()-style transform object.
breaks, labels	Explicit break positions (data units) and their labels, or NULL to compute them.
name	Axis title, or NULL to derive from the encoding.
expand	Axis padding beyond the data. NULL (default) keeps the usual 5% breathing room; a numeric c(mult, add) sets a proportion of the range plus a constant (either may be 0), applied to both ends. Both act in the scale's *transformed* space, so on a trans = "log10" axis add is in log units (matching how the axis is drawn). Use expand = c(0, 0) to clamp the axis exactly to the data (e.g. a bar chart's baseline), or expand = c(0.1, 0) for a wider margin.
sec.axis	A secondary axis from sec_axis() / dup_axis(), drawn on the opposite edge, or NULL for none. Continuous Cartesian plots only (see sec_axis() for the current limitations).

Value

The modified PlotSpec.

Examples

```
vplot(mtcars) |> mark_point(x = wt, y = mpg) |> scale_x_continuous(limits = c(0, 6))
vplot(mtcars) |> mark_point(x = wt, y = mpg) |> scale_y_continuous(trans = "sqrt")
```

---

scale_x_date *Date and time position scales*

Description

Position scales for Date (scale_*_date), date-time POSIXct (scale_*_datetime), and time-of-day / difftime / hms (scale_*_time) axes. A Date/POSIXct column already gets a sensible date axis automatically; declare one of these to control the break interval or the label format.

Usage

```
scale_x_date(
  plot,
  limits = NULL,
  date_breaks = NULL,
  date_labels = NULL,
  breaks = NULL,
  labels = NULL,
  name = NULL
)

scale_y_date(
  plot,
  limits = NULL,
  date_breaks = NULL,
  date_labels = NULL,
  breaks = NULL,
  labels = NULL,
  name = NULL
)

scale_x_datetime(
  plot,
  limits = NULL,
  date_breaks = NULL,
  date_labels = NULL,
  breaks = NULL,
  labels = NULL,
  name = NULL
)

scale_y_datetime(
  plot,
  limits = NULL,
  date_breaks = NULL,
  date_labels = NULL,
  breaks = NULL,
  labels = NULL,
  name = NULL
)

scale_x_time(
  plot,
  limits = NULL,
  date_breaks = NULL,
  date_labels = NULL,
  breaks = NULL,
  labels = NULL,
```

```
  name = NULL
)

scale_y_time(
  plot,
  limits = NULL,
  date_breaks = NULL,
  date_labels = NULL,
  breaks = NULL,
  labels = NULL,
  name = NULL
)
```

Arguments

plot	A PlotSpec.
limits	Length-2 vector giving the axis range (same class as the data), or NULL to train from the data.
date_breaks	A break interval string, e.g. "1 month", "2 weeks", "10 years" (passed to scales::breaks_width()). NULL uses pretty().
date_labels	A base::strftime() format string for the tick labels, e.g. "%b %Y". NULL uses the default format.
breaks, labels	Explicit break positions / labels (override date_breaks/date_labels), or NULL.
name	Axis title, or NULL to derive from the encoding.

Value

The modified PlotSpec.

Examples

```
df <- data.frame(day = as.Date("2020-01-01") + 0:364, y = cumsum(rnorm(365)))
vplot(df) |>
  mark_line(x = day, y = y) |>
  scale_x_date(date_breaks = "3 months", date_labels = "%b %Y")
```

---

sec_axis *Secondary axes*

Description

Add a secondary axis to a continuous position scale: a second set of ticks and labels on the opposite edge (top for x, right for y), computed as a 1:1 monotonic transform of the primary axis. Pass the result to the sec.axis argument of scale_x_continuous() / scale_y_continuous(). This is a labelling convenience (unit conversions, count/proportion dual readouts, duplicating an axis on a wide plot) — **not** an independent second axis with its own data.

Usage

```
sec_axis(transform = ~., name = NULL, breaks = NULL, labels = NULL)

dup_axis(transform = ~., name = NULL, breaks = NULL, labels = NULL)
```

Arguments

transform	The primary-to-secondary mapping: a formula using . (e.g. ~ . * 1.8 + 32), a function, or a scales::transform_log10()-style transform object. Must be monotonic over the axis range. Defaults to the identity, ~ ..
name	Secondary axis title, or NULL for none.
breaks	Break positions in **secondary** units, or NULL to compute them.
labels	A character vector of labels (one per break) or a labelling function, or NULL for the default number format.

Details

sec_axis() maps the primary axis through transform; dup_axis() is the identity special case (a plain duplicate) with tidy defaults.

Value

A SecAxisSpec to pass to sec.axis.

Limitations (current version)

Secondary axes are supported only on continuous position scales under the default Cartesian coordinate system, with **shared** scales across facets. Combining sec.axis with coord_flip() / coord_polar() / coord_trans(), with free facet scales, or with add_marginal() raises an error. In a plot composition (| / /) the secondary axis is not drawn.

See Also

scale_x_continuous()

Examples

```
# Celsius with a Fahrenheit axis on top
vplot(data.frame(t = 0:100, y = (0:100)^2)) |>
  mark_line(x = t, y = y) |>
  scale_x_continuous(name = "°C", sec.axis = sec_axis(~ . * 1.8 + 32, name = "°F"))

# Duplicate the y axis on the right
vplot(mtcars) |>
  mark_point(x = wt, y = mpg) |>
  scale_y_continuous(sec.axis = dup_axis())
```

`select_neighbours` *Highlight a node's graph neighbourhood*

Description

A network-aware `select_point()` preset for `vgraph()` plots: pointing at (or clicking) a node selects **its neighbourhood** — the node, its incident edges, and its adjacent nodes — so a host spotlights them and dims the rest. Pointing at an edge highlights the edge and its two endpoint nodes. It builds on the node/edge identity `vgraph()` emits for an interactive plot (each node keyed by its vertex `name`, each edge carrying its endpoint names); the host reconstructs the adjacency and projects the gesture across it.

Usage

```
select_neighbours(
  plot,
  name = "neighbours",
  on = c("hover", "click"),
  degree = 1L,
  edges = TRUE,
  empty = TRUE
)
```

Arguments

`plot`	A PlotSpec, normally from `vgraph()` (piped form), or a selection **name** string (free-standing form).
`name`	The selection name. Defaults to `"neighbours"`.
`on`	The gesture: `"hover"` (default) or `"click"`.
`degree`	How many hops out from the pointed node to include (`1`, the default, is the immediate neighbourhood; `2` adds neighbours-of-neighbours).
`edges`	Whether to include the incident edges in the highlight (`TRUE`, default) or only the nodes.
`empty`	Whether an empty selection matches *all* elements (`TRUE`, default — an un-hovered graph shows its full self) or none.

Details

Like the other selections it is **inert on a static render** and enacted by a capable host (`vellumwidget`). On its own it spotlights the neighbourhood; pair it with `condition()` to restyle members explicitly, or `filter_by()` to show only the neighbourhood.

Value

A PlotSpec (piped form) or a `SelectionSpec` (free-standing form).

See Also

select_point(), condition(), vgraph(), mark_edges()

Examples

```
## Not run:
g <- igraph::make_graph("Zachary")
vgraph(g) |>
  mark_edges() |>
  mark_nodes(size = 3) |>
  select_neighbours(on = "hover")

## End(Not run)
```

---

select_point *Declare an interactive selection*

Description

A **selection** is a named set of data elements defined by a user gesture. On its own it does nothing visible; refer to it from condition() (style by membership), filter_by() (show only members), or bind_scale() (drive another panel's view). Selections are part of the plot spec, so they travel with it, serialise with it, and are enacted by any capable host (vellumwidget). They are inert on a static render.

Usage

```
select_point(
  plot,
  name,
  on = c("click", "hover"),
  fields = NULL,
  group_by = NULL,
  toggle = TRUE,
  empty = TRUE
)

select_interval(
  plot,
  name,
  on = c("xy", "x", "y"),
  region = c("rect", "lasso"),
  empty = TRUE
)
```

Arguments

plot	A PlotSpec (piped form), or a selection **name** string (free-standing form).
name	The selection name (piped form) — a string other nodes reference.
on	The gesture. Point: "click" (default) or "hover". Interval: "xy" (default, both axes), "x", or "y" (a single-axis brush).
fields	For a point selection, the column name(s) whose match extends membership: clicking one element selects every row equal to it on these fields (e.g. fields = "grp" selects the whole group). NULL (default) selects only the clicked element. group_by is an alias.
group_by	Alias for fields.
toggle	For a point selection, whether clicking toggles membership (TRUE, default) or replaces it (FALSE, single-select).
empty	Whether an **empty** selection contains *all* elements (TRUE, default — so an unselected plot shows its full self and selecting *narrows*) or *none* (FALSE).
region	For an interval selection, "rect" (default, a brush rectangle) or "lasso" (a free-hand polygon).

Details

select_point() is driven by a click or hover (or a legend value); select_interval() by a drag (a brush rectangle or a lasso), optionally locked to an axis.

Both are **pipe-first** (attach to the plot and return it) *or* **free-standing** (call with the name as the first argument to get a SelectionSpec you can share across views, then attach with add_selection()).

Value

A PlotSpec (piped form) or a SelectionSpec (free-standing form).

See Also

condition(), filter_by(), add_selection(), bind_scale()

Examples

```
df <- data.frame(x = 1:10, y = runif(10), g = rep(c("a", "b"), 5))

# highlight on hover (piped)
vplot(df) |>
  mark_point(x = x, y = y, color = condition("hi", g, "grey80")) |>
  select_point("hi", on = "hover")

# free-standing, for cross-view use
sel <- select_interval("brush", on = "x")
```

shadow *Shadow layer effect*

Description

Draws dark, low-opacity copies of a stroked or point mark beneath the original, offset by (x, y) and softened by stacking a few widened copies — a drop shadow (with an offset) or an ambient shadow (offset 0). Applies to the same marks as glow().

Usage

```
shadow(
  x = 0.5,
  y = -0.5,
  color = "black",
  alpha = 0.3,
  spread = 1.5,
  layers = 3L
)
```

Arguments

x, y	Shadow offset in millimetres (+x right, +y up). Defaults to a small down-right drop.
color	Shadow colour.
alpha	Opacity of each copy.
spread	Softening spread, in millimetres, over which the copies widen.
layers	Number of stacked copies (more = softer).

Details

The offset is an **absolute distance in millimetres** (+x right, +y up), resolved device-side, so a drop shadow stays the same physical distance and is isotropic regardless of the panel's size or aspect (via vellum's compound npc + mm unit).

Value

A ShadowSpec object for a mark's effects list.

See Also

glow(), outline()

Examples

```
df <- data.frame(x = 1:20, y = cumsum(rnorm(20)))
vplot(df) |> mark_line(x = x, y = y, effects = list(shadow()))
```

sketch *Hand-drawn ("sketch") rendering*

Description

A re-export of `vellum::sketch()` — the one vocabulary vellumplot speaks for the hand-drawn look (wobbly outlines, hachure fills, à la Rough.js). Pass a `sketch()` value to any mark's `sketch =` argument, to an `element_line()` / `element_rect()` `sketch =` slot, or set it plot-wide with `theme_sketch()`:

Usage

```
sketch(...)
```

Arguments

`...`	Sketch parameters passed straight to `vellum::sketch()` — `roughness`, `bowing`, `fill_style`, `fill_weight`, `hachure_angle`, `hachure_gap`, `curve_tightness`, `disable_multi_stroke`, `preserve_vertices`, `seed`. See there for defaults.

Details

```
vplot(mpg) |> mark_point(x = displ, y = hwy, sketch = sketch(roughness = 1.2))
```

Sketch is a geometry property, not a layer effect: it perturbs the mark itself (its wobble is generated natively in the vellum engine, so it is exact, cross-backend, and works in PDF), rather than compositing extra copies. Text is never sketched — pair a handwriting `family` with it for a fully hand-drawn plot.

Resolution is most-specific-wins: a mark's `sketch =` beats an element slot, which beats the plot-wide `theme_sketch()` default. At any level `sketch = NA` (or `FALSE`) forces that element crisp, overriding a broader default; `sketch = NULL` inherits.

Value

A `vellum_sketch` object.

See Also

`theme_sketch()`, `mark_point()`, `element_line()`

Examples

```
vplot(mtcars) |>
  mark_point(x = wt, y = mpg, sketch = sketch(roughness = 1.5, seed = 7))
```

spec_diagnose *Diagnose a spec against its data*

Description

spec_diagnose() checks a spec (a list, JSON string, or path) without throwing: it parses it, validates every referenced field against the data, and dry-run compiles it, returning a structured verdict. vplot_from_spec() is the strict form — it returns the PlotSpec on success and raises a classed vellumplot_spec_invalid error (carrying the same diagnostics) on failure.

Usage

```
spec_diagnose(spec, data = NULL, env = globalenv())

vplot_from_spec(spec, data = NULL, env = globalenv())
```

Arguments

spec	A spec list (from as_spec()), a JSON string, or a .json path.
data	The data frame the spec is drawn from. Required when the spec stores data by reference; optional (for field validation) when inlined.
env	Environment channel expressions are re-quoted in.

Details

This is the agent repair loop: a failure comes back as data an agent can act on (which field is unknown, the nearest real field) rather than a traceback.

Value

spec_diagnose() returns a list with ok (logical), plot (the PlotSpec or NULL), and diagnostics (a list of records with severity, field, message, hint). vplot_from_spec() returns a PlotSpec.

See Also

spec_fields(), spec_schema(), vplot_ask()

Examples

```
spec <- as_spec(vplot(mtcars) |> mark_point(x = wt, y = mpg))
d <- spec_diagnose(spec, data = mtcars)
d$ok
```

spec_fields *Summarise a data frame's fields for a model*

Description

spec_fields() returns one row per column of data with its name, inferred encoding type (quantitative / nominal / ordinal / temporal), R class, distinct-value count, and a few example values. It is the *grounding* step for LLM plot generation (vplot_ask(), the MCP list_fields tool): handing a model the real vocabulary up front is what prevents hallucinated column names, and vplot_from_spec() then validates against exactly these.

Usage

```
spec_fields(data, n_examples = 3L)
```

Arguments

data	A data frame.
n_examples	How many example values to show per field (default 3).

Value

A data frame with columns name, type, class, n_unique, examples.

See Also

vplot_from_spec(), spec_schema()

Examples

```
spec_fields(mtcars)
```

spec_schema *The vellumplot spec JSON Schema*

Description

Returns the bundled JSON Schema (Draft 2020-12) describing a serializable vellumplot spec — the input contract for spec_from_json() and the MCP render_spec / get_schema tools. An agent can validate a generated spec against it before rendering.

Usage

```
spec_schema(as = c("string", "list", "path"))
```

Arguments

as	A character scalar: "string" (raw JSON, the default), "list" (parsed R list), or "path" (the file path in the installed package).

Value

The schema as a string, a list, or a file path.

See Also

spec_to_json(), spec_fields()

Examples

```
schema <- spec_schema(as = "list")
schema$title
```

---

spec_to_json *Serialize a plot to / from a JSON spec string*

Description

spec_to_json() renders a PlotSpec as a JSON string (via as_spec()); spec_from_json() parses one back into a PlotSpec (via from_spec()). This is the portable wire format for the LLM / agent tooling and the Vega-Lite bridge. See as_spec() for the serializable subset and its guarantees.

Usage

```
spec_to_json(plot, pretty = TRUE)

spec_from_json(json, data = NULL, env = globalenv())
```

Arguments

plot	A PlotSpec.
pretty	Whether to pretty-print (default TRUE).
json	A JSON spec string (or a path to a .json file).
data, env	Passed to from_spec() — a data frame for a by-reference spec, and the environment channel expressions are re-quoted in.

Value

spec_to_json() returns a length-1 JSON string; spec_from_json() returns a PlotSpec.

See Also

as_spec(), spec_schema(), vplot_from_spec()

Examples

```
p <- vplot(mtcars) |> mark_point(x = wt, y = mpg)
json <- spec_to_json(p)
p2 <- spec_from_json(json)
```

`spec_to_vegalite` *Convert between a vellumplot spec and a Vega-Lite specification*

Description

`spec_to_vegalite()` translates a PlotSpec into a Vega-Lite specification; `spec_from_vegalite()` translates one back. Both are layered grammars of graphics, so the mapping is direct for the common cases — marks, encodings, scales (domain / log-sqrt type / categorical range / title), `bin` + `count`, faceting, inline data, and the plot title.

Usage

```
spec_to_vegalite(plot, json = FALSE)

spec_from_vegalite(vl, data = NULL, env = globalenv())
```

Arguments

`plot`	A PlotSpec.
`json`	For `spec_to_vegalite()`, return a JSON string instead of a list.
`vl`	A Vega-Lite spec as a parsed list or a JSON string.
`data`	Optional data frame for a Vega-Lite spec whose data is external.
`env`	Environment channel expressions are re-quoted in (for `spec_from_vegalite()`).

Details

The bridge covers a **documented subset** (see the "Specs, agents, and interoperability" article). Anything it cannot map — polar/flipped coordinates, layer effects, patterns, statistical marks without a Vega-Lite equivalent — is **reported** via a warning listing the dropped features, never silently diverged.

Value

`spec_to_vegalite()` returns a Vega-Lite spec (a list, or JSON string if `json = TRUE`); `spec_from_vegalite()` returns a PlotSpec.

See Also

`as_spec()`, `spec_to_json()`

Examples

```
p <- vplot(mtcars) |> mark_point(x = wt, y = mpg, color = cyl)
vl <- spec_to_vegalite(p)
vl$mark
```

`theme_gray` *Plot themes*

Description

Control the non-data look of a plot. `theme_gray()` is the default (grey panel, white gridlines); `theme_minimal()` drops the panel fill for light gridlines on the page; `theme_bw()` is a white panel with light grey gridlines; `theme_classic()` has axis lines and no gridlines; `theme_void()` strips everything but the marks, legend, and titles; `theme_cyberpunk()` is a dark neon theme (see Details).

Usage

```
theme_gray(plot)

theme_minimal(plot)

theme_bw(plot)

theme_classic(plot)

theme_void(plot)

theme_light(plot)

theme_dark(plot)

theme_linedraw(plot)

theme_cyberpunk(plot)

theme(plot, ...)

set_theme(
  plot,
  panel_bg = NULL,
  grid_col = NULL,
  label_col = NULL,
  strip_bg = NULL
)
```

Arguments

plot	A PlotSpec. theme() also accepts a PlotComposition, setting the figure-level chrome (title bands, collected legend, panel spacing, tags).
...	Named theme elements, e.g. plot.title = element_text(size = 16), panel.grid.minor = element_blank(), or settings like legend.position, one of "right" (default), "left", "top", "bottom", or "none". Legend geometry is tunable via legend.key.size (key/swatch side, mm), legend.spacing (gap between stacked guides, mm), and legend.margin (inset around the legend block, one or four millimetres, t, r, b, l).
panel_bg, grid_col, label_col, strip_bg	Colours (or NA to draw nothing) for the panel background, gridlines, axis-label/legend text, and facet strip background.

Details

theme() overrides individual elements on top of the current theme using element_text() / element_line() / element_rect() / element_blank(). set_theme() is a small back-compatible shortcut for the most common colours.

theme_cyberpunk() sets a dark canvas with dim neon gridlines and a bright neon default palette (both discrete and continuous), in the spirit of mplcyberpunk. It pairs with the glow() layer effect and linear_gradient() fills for the full neon look; the palette is only a *default*, so scale_* overrides still win.

Value

The modified PlotSpec.

Examples

```
vplot(mtcars) |> mark_point(x = wt, y = mpg) |> theme_minimal()
vplot(mtcars) |>
  mark_point(x = wt, y = mpg) |>
  theme(panel.grid.minor = element_blank(), legend.position = "none")
```

---

theme_sketch *Hand-drawn plot theme*

Description

theme_sketch() turns a whole plot hand-drawn in one line: it sets a plot-wide sketch() default that every geometry mark and theme element inherits (wobbly gridlines, axis lines, ticks, and marks), on a warm paper-toned background. Text is never sketched — pass a handwriting font to complete the look.

Usage

```
theme_sketch(
  plot,
  roughness = 1,
  bowing = 1,
  fill_style = "hachure",
  seed = 1L,
  font = NULL,
  paper = "#f4ecd8",
  ink = "#2b2b2b"
)
```

Arguments

plot	A PlotSpec.
roughness, bowing, fill_style, seed	Passed to sketch() to build the plot-wide default (see sketch() for the full set of knobs).
font	Font family for all text (text is not sketched; a handwriting font such as "Comic Sans MS" or "Chilanka" sells the hand-drawn look). NULL (default) keeps the system default family.
paper, ink	Background and foreground (text/line) colours.

Details

It is a complete theme (like theme_cyberpunk()): the sketch it sets is only a *default*, so any mark's sketch = argument, any element_line() / element_rect() sketch = slot, or a scale_* override still wins. Force an individual element crisp with sketch = NA.

Value

The modified PlotSpec.

See Also

sketch(), theme_gray(), mark_point()

Examples

```
vplot(mtcars) |>
  mark_point(x = wt, y = mpg, color = factor(cyl)) |>
  theme_sketch()

# tune the wobble; a crosshatch-filled bar layer that stays crisp
df <- data.frame(g = c("a", "b", "c"), n = c(3, 5, 2))
vplot(df) |>
  mark_bar(x = g, y = n, fill = g) |>
  theme_sketch(roughness = 1.4, fill_style = "crosshatch", seed = 7)
```

transition_reveal *Reveal a plot progressively along a variable*

Description

transition_reveal() wipes the plot into view left to right — the classic "line draws itself" animation. Unlike transition_states(), it does not tween between states: the full plot is compiled and a clip rectangle grows across the panel, revealing the marks in along order. Best for a line or path over a continuous along (typically the x variable, e.g. time).

Usage

```
transition_reveal(plot, along)
```

Arguments

plot	A PlotSpec (from vplot()).
along	The variable the reveal follows (mapped to x, increasing left to right).

Value

The modified PlotSpec.

See Also

transition_states(), transition_time(), animate()

Examples

```
vplot(data.frame(t = 1:20, y = cumsum(rnorm(20)))) |>
  mark_line(x = t, y = y) |>
  transition_reveal(t)
```

transition_states *Animate a plot across the states of a variable*

Description

transition_states() turns a plot into a keyframe animation: the plot is compiled once per level of states (a "keyframe"), and animate() tweens the frames between successive keyframes. The scales are trained **once over all states and frozen**, so axis breaks, colour ramps and sizes stay put across the whole animation (the interpolation is non-reactive: the states are fixed at author time, nothing retrains in response to anything).

Usage

```
transition_states(
  plot,
  states,
  transition_length = 1,
  state_length = 1,
  wrap = TRUE
)
```

Arguments

plot	A PlotSpec (from vplot()).
states	A column (bare or a string) whose distinct values, in level order, are the animation's states.
transition_length	Relative duration of the moving segments between states.
state_length	Relative duration of the held pause on each state.
wrap	Loop the last state back to the first?

Details

The step only records the intent; call animate() to build the animation and anim_save() to write it. A plain (non-animated) render ignores it.

Value

The modified PlotSpec.

See Also

ease_aes(), animate(), anim_save()

Examples

```
if (requireNamespace("gapminder", quietly = TRUE)) {
  library(gapminder)
  p <- vplot(gapminder) |>
    mark_point(x = gdpPercap, y = lifeExp, size = pop, color = continent) |>
    scale_x_continuous(trans = "log10") |>
    transition_states(year)
}
```

transition_time *Animate a plot along a continuous time*

Description

`transition_time()` is like `transition_states()` but treats `time` as a continuous quantity: the states are its distinct values and each transition is allocated frames **in proportion to its time gap**, so the animation plays at a constant rate through unevenly spaced times. There is no pause on a state and no wrap — time flows once, start to finish.

Usage

```
transition_time(plot, time)
```

Arguments

`plot`	A PlotSpec (from `vplot()`).
`time`	A numeric column giving each row's time.

Value

The modified PlotSpec.

See Also

`transition_states()`, `ease_aes()`, `animate()`

Examples

```
if (requireNamespace("gapminder", quietly = TRUE)) {
  vplot(gapminder::gapminder) |>
    mark_point(x = gdpPercap, y = lifeExp, size = pop, color = continent) |>
    scale_x_continuous(trans = "log10") |>
    transition_time(year)
}
```

vchord *Chord diagram*

Description

`vchord()` draws a weighted flow as a circular chord diagram: one arc (*sector*) per node, sized by the node's total incident weight, and one ribbon per flow joining a slice of the source sector to a slice of the target sector through the centre. Input is a flow list – `from`, `to`, `value` – or a square flow matrix (row = from, column = to; `dimnames` are the node names). Like `vgraph()` / `vsankey()` it returns a ready, axis-free, aspect-locked PlotSpec; `mark_chord()` is the layer it adds.

Usage

```
vchord(
  data,
  from,
  to,
  value,
  gap = 0.02,
  sort = c("input", "value"),
  link_color = c("source", "target"),
  direction = c("gradient", "gap", "both", "none"),
  label = TRUE,
  width = 6,
  height = 6,
  dpi = 96
)

mark_chord(
  plot,
  from,
  to,
  value,
  gap = 0.02,
  sort = c("input", "value"),
  link_color = c("source", "target"),
  direction = c("gradient", "gap", "both", "none"),
  label = TRUE
)
```

Arguments

`data`	A data frame of flows, or a square numeric matrix (row = from, column = to).
`from`, `to`, `value`	Columns (tidy-eval) for the data-frame form: the source node, target node, and flow weight. Omit for the matrix form.
`gap`	Gap between sectors, as a fraction of the circle each (default `0.02`).
`sort`	Sector and ribbon order: `"input"` (default, first-appearance) or `"value"` (largest weight first).
`link_color`	Colour ribbons by their `"source"` (default) or `"target"` node.
`direction`	How to show a ribbon's direction: `"gradient"` (default, fade from opaque at the source to faint at the target), `"gap"` (stop the ribbon short of the target sector, leaving a small gap), `"both"`, or `"none"`.
`label`	Label each sector with its node name? Default `TRUE`.
`width`, `height`, `dpi`	Page size (inches) and resolution.
`plot`	A PlotSpec.

Details

Flows are directed: each node's arc splits into an outgoing block then an incoming block, so `a -> b` and `b -> a` are separate ribbons and a self-flow `a -> a` loops from the node's out-block to its own in-block. Sectors are coloured by node (a qualitative palette); ribbons take their source node's colour by default (`link_color = "target"` to colour by target).

Value

A PlotSpec (`vchord()`) or the modified plot (`mark_chord()`).

Examples

```
flows <- data.frame(
  from = c("A", "A", "B", "C"),
  to = c("B", "C", "C", "A"),
  value = c(3, 2, 4, 1)
)
vchord(flows, from, to, value)
```

---

vdendrogram *Dendrogram from a clustering*

Description

`vdendrogram()` turns a base `hclust`/`dendrogram` (e.g. from `stats::hclust()`) into a ready node-link dendrogram: leaves spread along a line, each merge drawn at its height, joined by right-angle brackets. It is a thin preset over `vgraph(<clustering>, layout = "dendrogram")` plus bracket edges and leaf labels; for full control, use that path and add marks yourself.

Usage

```
vdendrogram(
  x,
  direction = c("down", "up", "left", "right"),
  k = NULL,
  labels = TRUE,
  label_size = 5,
  width = 7,
  height = 5,
  dpi = 96
)
```

Arguments

`x`	An `hclust` or `dendrogram` object.
`direction`	Growth direction (leaves opposite the root): `"down"` (default; root at top), `"up"`, `"left"`, or `"right"`.

k	Optionally cut the tree into k clusters (stats::cutree()) and colour branches and leaf labels by cluster; branches above the cut stay neutral. NULL (default) draws a single-colour tree.
labels	Draw the leaf labels? Default TRUE.
label_size	Leaf-label font size (points). Default 5; lower it for a tree with many leaves.
width, height, dpi	Page size (inches) and resolution.

Value

A PlotSpec.

See Also

vgraph() for the general node-link path (layout = "dendrogram" or "unrooted").

Examples

```
hc <- hclust(dist(USArrests))
vdendrogram(hc, k = 3)
```

---

vgraph *Start a graph (network) plot*

Description

vgraph() begins a node-link diagram from an igraph graph. It computes a layout (vertex x/y), builds a node table and an edge table, and returns a PlotSpec with the aspect locked (coord_equal) and a void theme (no axes or gridlines) – publication-quality defaults with no tuning. Add layers with mark_edges(), mark_nodes(), and mark_node_text(); edges default to the edge table and nodes to the node table.

Usage

```
vgraph(
  g,
  layout = "stress",
  ...,
  augment = FALSE,
  filter_nodes = NULL,
  filter_edges = NULL,
  k_core = NULL,
  seed = 42L,
  width = 6,
  height = 4,
  dpi = 96
)
```

Arguments

g	An igraph graph (or a two-column edge-list matrix).
layout	A layout: a name ("stress" (default), "sparse_stress", "backbone", "fr", "kk", "circle", "tree", "sugiyama", ...), a supplied N x 2 coordinate matrix, or a function f(g, ...) returning one.
...	Extra arguments forwarded to the layout function (e.g. pivots = for sparse stress).
augment	Opt-in vertex metrics to attach as graph attributes (so they can be mapped – mark_nodes(size = degree) – or used by filter_nodes). FALSE (default) attaches nothing; TRUE attaches degree and components; a character vector picks from "degree", "in_degree", "out_degree", "betweenness", "closeness", "eigen", "coreness", "components", "community". Never computed silently – vellum does not guess which centrality you meant.
filter_nodes, filter_edges	Data-masked predicates that keep the vertices / edges for which they are TRUE, evaluated against the vertex / edge attribute table – e.g. filter_edges = weight > 0.5, filter_nodes = degree >= 2 (with augment). The graph is reduced *before* the layout, so the picture is of the subgraph, not a full layout with holes.
k_core	Keep only the k-core: vertices with coreness >= k_core (and their induced edges). NULL (default) keeps everything.
seed	Integer seed for stochastic layouts ("fr", "drl"), making the figure reproducible. The global RNG stream is restored afterwards.
width, height	Page size in inches.
dpi	Output resolution in dots per inch (see vplot()).

Details

The default layout is stress majorization (graphlayouts::layout_with_stress): it is deterministic (same graph -> same picture) and minimizes the difference between drawn and graph-theoretic distance. Above 10^4 nodes it falls back to sparse stress automatically. Reciprocal and parallel edges are drawn as parallel straight lines offset off the centre line (not curved); self-loops as small loops.

igraph (and graphlayouts, for its layouts) are optional dependencies (in Suggests); vgraph() errors with an install hint if they are not available.

Reductions apply in a fixed order: augment (metrics computed on the input graph) -> filter_edges -> filter_nodes -> k_core. Augmented metrics thus reflect the *input* graph, not the post-filter subgraph.

Value

A PlotSpec whose data is the node table and whose edge_data is the edge table.

See Also

mark_edges(), mark_nodes(), mark_node_text()

Examples

```
## Not run:
g <- igraph::make_graph("Zachary")
vgraph(g, layout = "stress") |>
  mark_edges() |>
  mark_nodes(size = 3)

# attach degree, then plot only the 2-core, sized by degree
vgraph(g, augment = TRUE, k_core = 2) |>
  mark_edges() |>
  mark_nodes(size = degree)

## End(Not run)
```

---

vhierarchy *Hierarchy diagrams: sunburst, icicle, treemap, circle-pack*

Description

`vhierarchy()` draws a tree as a space-filling diagram, choosing the geometry with `type`: `"sunburst"` (concentric ring sectors), `"icicle"` (rectangular partition), `"treemap"` (squarified nested rectangles), or `"circlepack"` (circles within circles). All four take the same *parent list* — `id`, `parent` (`NA`/`""` at the root), and `value` (leaf values; an internal node's value is the sum of its children) — so switching `type` re-encodes the same data. Like `vgraph()` / `vsankey()` it returns a ready, axis-free, aspect-locked PlotSpec; `mark_hierarchy()` is the layer it adds.

Usage

```
vhierarchy(
  data,
  id,
  parent,
  value,
  fill = NULL,
  type = c("sunburst", "icicle", "treemap", "circlepack"),
  inner_radius = 0,
  flow = c("down", "up", "right", "left"),
  lighten = 0.6,
  label = TRUE,
  show_values = FALSE,
  orientation = c("auto", "radial", "tangential", "horizontal"),
  root_label = FALSE,
  width = 6,
  height = 6,
  dpi = 96
)
```

```
mark_hierarchy(
  plot,
  id,
  parent,
  value,
  fill = NULL,
  type = c("sunburst", "icicle", "treemap", "circlepack"),
  inner_radius = 0,
  flow = c("down", "up", "right", "left"),
  lighten = 0.6,
  label = TRUE,
  show_values = FALSE,
  orientation = c("auto", "radial", "tangential", "horizontal"),
  root_label = FALSE
)
```

Arguments

`data`	A data frame describing a hierarchy (a parent list).
`id`, `parent`, `value`	Columns (tidy-eval): the node id, its parent id (`NA`/`""` for the root), and its value (used for leaves).
`fill`	Optional column (tidy-eval) to colour nodes by. Unmapped (default), nodes are coloured by their depth-1 branch and lightened with depth; mapped, each node takes its `fill` value's colour with no depth fade.
`type`	Diagram geometry: `"sunburst"` (default), `"icicle"`, `"treemap"`, or `"circlepack"`.
`inner_radius`	Sunburst only: central hole radius, a fraction in `[0, 1)`; `0` (default) fills to the centre.
`flow`	Icicle only: the direction of increasing depth — `"down"` (default), `"up"`, `"right"`, or `"left"`.
`lighten`	Branch mode only: how far the deepest level fades toward white, a fraction in `[0, 1]` (default `0.6`; `0` = flat colour per branch). Ignored when `fill` is mapped.
`label`	Label each node with its `id`? Default `TRUE`. Nodes too small for their label are left unlabelled.
`show_values`	Append each node's value to its label, e.g. `"A1 (3)"`. Default `FALSE`.
`orientation`	Sunburst labels only: `"auto"` (default) angles each label tangentially / radially to fit its wedge, or force `"radial"`, `"tangential"`, or `"horizontal"`. Labels are kept upright.
`root_label`	Write the root's name (and, with `show_values`, its total) in the centre? Default `FALSE`. Most useful for `"sunburst"`.
`width`, `height`, `dpi`	Page size (inches) and resolution.
`plot`	A PlotSpec.

Details

The root is structural and never drawn (in a sunburst it is the centre). Each node is labelled with its `id` where the label fits, and `show_values` appends the value; small nodes are left unlabelled so a dense diagram stays legible.

Value

A PlotSpec (`vhierarchy()`) or the modified plot (`mark_hierarchy()`).

Colour

By default nodes are coloured by their depth-1 *branch* and lightened one step per level, so sibling branches stay distinct and depth reads as shade. The branch is an ordinary discrete fill scale, so `scale_fill_*()` recolours the branches (e.g. `scale_fill_brewer()`), and `lighten` controls the depth fade (`0` = flat colour per branch). Map `fill` to a node column instead to colour every node by that variable — discrete or continuous, with the matching `scale_fill_*()` — in which case the depth fade is not applied.

Examples

```
h <- data.frame(
  id = c("root", "A", "B", "A1", "A2", "B1"),
  parent = c(NA, "root", "root", "A", "A", "B"),
  value = c(NA, NA, NA, 3, 2, 4)
)
vhierarchy(h, id, parent, value, show_values = TRUE)
vhierarchy(h, id, parent, value, type = "treemap")
```

---

vl_pattern *Custom tiling-pattern fill*

Description

A thin re-export of `vellum::vl_pattern()` for building a pattern fill from an arbitrary tile grob, when the ready-made `pattern_stripe()` family does not fit. Like a gradient, the result is an unscaled `fill` *value*.

Usage

```
vl_pattern(
  grob,
  width = 0.1,
  height = 0.1,
  x = 0.5,
  y = 0.5,
  units = "npc",
  extend = "repeat"
)
```

Arguments

grob, width, height, x, y, units, extend
: See vellum::vl_pattern().

Value

A vellum_pattern object usable as a fill value.

See Also

pattern_stripe(), linear_gradient()

Examples

```
dots <- vellum::circle_grob(r = 0.25, gp = vellum::vl_gpar(fill = "grey30"))
vl_pattern(dots, width = 0.08, height = 0.08)
```

---

vplot *Start a plot specification*

Description

vplot() begins a declarative, pipe-first plot. It captures the data and page size and returns an inspectable PlotSpec; nothing is drawn until the spec is compiled (via render_plot() or vellum::as_vellum_scene()). Build the plot up with mark_point() / mark_line() / mark_rule() and the scale_*() functions.

Usage

```
vplot(data, width = 6, height = 4, dpi = 96)
```

Arguments

data	A data frame. Encoding expressions in mark_*() are evaluated against it with tidy evaluation.
width, height	Page size in inches.
dpi	Output resolution in dots per inch. The exported PNG's pixel dimensions are width * dpi by height * dpi; raising dpi yields a denser image at the same physical size. Overridable at render time via render_plot()'s dpi argument.

Value

A PlotSpec.

Examples

```
vplot(mtcars) |> mark_point(x = wt, y = mpg)
```

vplot_ask *Generate a plot from a natural-language request*

Description

vplot_ask() is a thin, **model-agnostic** convenience: it assembles the grounding payload (the JSON spec_schema() plus spec_fields() for data and the user's prompt), hands it to a responder function you supply — any function that takes the payload and returns a JSON spec string (wrapping the Claude API, an MCP client, a local model, or a canned fixture) — and validates the result with vplot_from_spec().

Usage

```
vplot_ask(prompt, data, responder, env = globalenv())
```

Arguments

prompt	A natural-language description of the desired plot.
data	The data frame to plot.
responder	A function function(payload) -> json_string. payload is a list with prompt, fields (from spec_fields()), and schema.
env	Environment channel expressions are re-quoted in.

Details

vellumplot deliberately ships no built-in model client: the value is the schema, the grounding, and the validated repair loop, not a hard dependency on one provider. For a ready-to-run agent integration, point an MCP client at the bundled server (system.file("mcp/server.R", package = "vellumplot")).

Value

A PlotSpec.

See Also

spec_fields(), vplot_from_spec(), spec_schema()

Examples

```
# a canned responder standing in for a model call:
responder <- function(payload) {
  spec_to_json(vplot(mtcars) |> mark_point(x = wt, y = mpg))
}
p <- vplot_ask("scatter of weight vs mpg", mtcars, responder)
```

vsankey *Sankey (flow) diagram*

Description

`vsankey()` draws a layered flow diagram from a *flow list* — one row per flow with a `from` node, a `to` node, and a `value` (the ribbon width). Nodes are the union of `from`/`to`; a node that is both a source and a target makes the diagram multi-stage. Like `vgraph()`, it returns a ready PlotSpec with an axis-free, aspect-free panel; `mark_sankey()` is the layer it adds and can be used directly on a plot you have set up yourself.

Usage

```
vsankey(
  data,
  from,
  to,
  value,
  label = TRUE,
  show_values = FALSE,
  flow_color = "source",
  node_width = 0.04,
  node_gap = 0.02,
  width = 8,
  height = 5,
  dpi = 96
)

mark_sankey(
  plot,
  from,
  to,
  value,
  label = TRUE,
  show_values = FALSE,
  flow_color = "source",
  node_width = 0.04,
  node_gap = 0.02
)
```

Arguments

`data`	A data frame of flows.
`from`, `to`, `value`	Columns (tidy-eval): the source node, target node, and flow value.
`label`	Draw node labels? Default `TRUE`.

show_values	Append each node's value to its label (e.g. "Grid (60)")? Default FALSE. Ignored when label = FALSE.
flow_color	Ribbon fill: "source" (default) colours each ribbon by its source node, "target" by its target node, and "gradient" fades each ribbon from its source colour to its target colour.
node_width	Node-rectangle width, as a fraction of the plotting width (default 0.04).
node_gap	Vertical gap between the nodes in a column, as a fraction of the column height (default 0.02).
width, height, dpi	Page size (inches) and resolution.
plot	A PlotSpec.

Details

The flows must form a DAG (no cycles). Nodes within a column are ordered to minimise ribbon crossings (a deterministic Sugiyama barycenter sweep), and ribbons are stacked to meet each node in matching order. Nodes are coloured from the built-in qualitative palette.

Value

A PlotSpec (vsankey()) or the modified plot (mark_sankey()).

Examples

```
flows <- data.frame(
  from = c("A", "A", "B", "C", "C"),
  to = c("B", "C", "D", "D", "E"),
  value = c(4, 6, 4, 4, 2)
)
vsankey(flows, from, to, value)
vsankey(flows, from, to, value, show_values = TRUE, flow_color = "target")
```

---

vsparkline *Sparklines — tiny word-sized charts*

Description

vsparkline() builds a compact, axis-free chart of a single numeric series, sized in physical units (mm by default) so it reads as a *word-sized graphic* (Tufte's sparkline) — for a table cell, a caption, or a dashboard tile. It is a self-contained PlotSpec: render it with render_plot(), drop it into a composition with inset(), or (soon) a table cell.

Usage

```
vsparkline(
  values,
  type = c("line", "bar", "winloss"),
  color = "grey30",
  points = c("extremes", "last", "none"),
  point_color = "firebrick",
  baseline = 0,
  win_color = "#2c7fb8",
  loss_color = "#d7301f",
  linewidth = 1,
  point_size = 1.4,
  width = 20,
  height = 6,
  units = "mm",
  dpi = 96
)
```

Arguments

values	A numeric vector (the series), length >= 2.
type	"line" (default), "bar", or "winloss".
color	Trend / bar colour. Default "grey30".
points	For type = "line", which points get a dot: "extremes" (default – the min and max), "last", or "none".
point_color	Dot colour (default "firebrick").
baseline	For "bar", the value bars grow from (default 0); for "winloss", the threshold separating a win (>=) from a loss (default 0).
win_color, loss_color	For "winloss", the up / down bar colours (defaults blue / red).
linewidth	Trend line width (default 1).
point_size	Dot **diameter** in mm (default 1.4).
width, height, units	Physical size of the sparkline; units is one of "mm" (default), "cm", "in", "pt". Default 20 x 6 mm.
dpi	Resolution for raster output.

Details

Three shapes via type: a "line" trend (with optional dots on its extremes or last point), a "bar" column micro-chart, and a "winloss" chart of equal up/down bars about a baseline (for streaks of wins/losses, gains/drops).

The chart fills its box with no axes, gridlines, labels, or legend.

Value

A PlotSpec.

See Also

render_plot(), inset()

Examples

```
set.seed(1)
vsparkline(cumsum(rnorm(30)))
vsparkline(rpois(20, 5), type = "bar")
vsparkline(sample(c(-1, 1), 20, replace = TRUE), type = "winloss")
```

---

vtable *Tables with sparkline columns*

Description

vtable() lays a data frame out as a grid of cells where an ordinary column renders as text and a **list-column of numeric vectors** renders as a per-row vsparkline() — chart-in-table, drawn as one vector scene (so it renders on PNG / SVG / PDF like any plot). It returns a compiled object; render_plot() it, print() it, or drop it into a composition.

Usage

```
vtable(
  data,
  spark = list(),
  cols = NULL,
  align = NULL,
  header = TRUE,
  font_size = 9,
  spark_width = 24,
  row_height = 7,
  cell_pad = 1.5,
  units = "mm",
  dpi = 96
)
```

Arguments

data	A data frame. Sparkline columns must be **list-columns** whose cells are numeric vectors.
spark	A named list mapping a (list-)column name to how it draws: a type string ("line"/"bar"/"winloss") or a function function(values) returning a vsparkline().
cols	Character vector of columns to show, in order (default all).

`align`	Optional named list of per-column alignment (`"left"`/`"right"`/`"centre"`); defaults to right for numeric, left otherwise.
`header`	Draw a bold header row with an underline (default `TRUE`).
`font_size`	Text size in points (default `9`).
`spark_width`	Width of a sparkline column, in `units` (default `24`).
`row_height`	Row height, in `units` (default `7`).
`cell_pad`	Horizontal padding inside a text cell, in `units` (default `1.5`).
`units`	Length unit for the sizes above: `"mm"` (default), `"cm"`, `"in"`, `"pt"`.
`dpi`	Resolution for raster output.

Value

A `VTable` (renders via `render_plot()`).

See Also

`vsparkline()`

Examples

```
df <- data.frame(name = c("A", "B", "C"), mean = c(3.1, 5.4, 2.2))
df$trend <- list(cumsum(rnorm(20)), cumsum(rnorm(20)), cumsum(rnorm(20)))
vtable(df, spark = list(trend = "line"))
```

---

vvenn *Venn / Euler diagrams*

Description

`vvenn()` draws a Venn diagram of 2 or 3 sets: overlapping circles whose disjoint regions are filled by how many elements fall in exactly that combination of sets. Each region is drawn as solid **geometry** (via `vellum::vl_path_op()`'s boolean set operations), so overlapping regions do not alpha-composite and stay crisp — including in PDF, where a rasterised mask would degrade.

Usage

```
vvenn(sets, width = 5, height = 5, dpi = 96)
```

Arguments

`sets`	Either a **named list** of member vectors (`list(A = ..., B = ...)`), or a **data frame** of logical membership columns (each column is a set; `TRUE` means that row is a member). 2 or 3 sets.
`width, height, dpi`	Figure size (inches) and resolution.

Value

A PlotSpec; print or `render_plot()` it.

Examples

```
vvenn(list(
  Coffee = c("Ann", "Bo", "Cy", "Di"),
  Tea = c("Bo", "Di", "Ed", "Fi")
))
```

`vwaffle` *Waffle chart*

Description

`vwaffle()` draws a **waffle** (square pie): a grid of cells coloured by category, where each category takes a share of the cells proportional to its count — a part-of-whole chart that is easier to read than a pie because the eye counts squares. It is a self-contained chart (like `vsankey()` / `vvenn()`): it returns a PlotSpec with no axes.

Usage

```
vwaffle(
  data,
  category,
  value = NULL,
  n_cells = 100,
  rows = 10,
  flip = FALSE,
  pad = 0.12,
  width = 5,
  height = 5,
  dpi = 96
)
```

Arguments

`data`	A data frame.
`category`	The categorical column that colours the cells (tidy-eval).
`value`	Optional per-row weight column; if omitted, each row counts once.
`n_cells`	Total number of cells in the grid (default `100`). Each category gets `round(share * n_cells)` cells.
`rows`	Number of rows in the grid (default `10`); cells fill column by column from the bottom-left.
`flip`	Fill row by row (left to right) instead of column by column.

pad Gap between cells, as a fraction of a cell (default 0.12).

width, height, dpi
Page size (inches) and resolution for the standalone chart.

Value

A PlotSpec.

See Also

vvenn(), mark_pie()

Examples

```
df <- data.frame(part = c("a", "b", "c"), n = c(50, 30, 20))
vwaffle(df, category = part, value = n)
```

Index

add_marginal, 5
add_marginal(), *95*, *124*
add_selection, 6
add_selection(), *26*, *96*, *127*
after_stat, 7
anim_save, 7
anim_save(), *8, 9*, *138*
animate, 8
animate(), *7, 8*, *23*, *96*, *137–139*
annotate, 9
area, 11
area(), *16*
as_spec, 11
as_spec(), *130*, *132, 133*

base::strftime(), *123*
base::summary(), *94*
bind_scale, 13
bind_scale(), *96*, *126, 127*

clip, 13
clip_layer *(*clip*)*, 13
clip_to *(*clip*)*, 13
clip_to(), *96*
compose_annotation, 14
compose_annotation(), *17*
concat, 15
concat(), *15*, *92*, *101*
condition, 17
condition(), *125–127*
coord_cartesian, 18
coord_cartesian(), *95*
coord_equal *(*coord_cartesian*)*, 18
coord_fixed *(*coord_cartesian*)*, 18
coord_fixed(), *6*, *95*
coord_flip *(*coord_cartesian*)*, 18
coord_flip(), *5, 6*, *95*, *124*
coord_polar, 19
coord_polar(), *6*, *61, 62*, *124*
coord_radial *(*coord_polar*)*, 19
coord_sf, 20
coord_sf(), *37, 38*, *69, 70*
coord_trans, 21
coord_trans(), *124*

dup_axis *(*sec_axis*)*, 123
dup_axis(), *121*

ease_aes, 22
ease_aes(), *9*, *96*, *138, 139*
echo *(*motion*)*, 81
edgebundle::edge_bundle(), *53*
edgebundle::flow_tree(), *53*
edgebundle::tnss_tree(), *53*
effect, *129*
element, 23
element_blank *(*element*)*, 23
element_blank(), *135*
element_line *(*element*)*, 23
element_line(), *24*, *129*, *135, 136*
element_rect *(*element*)*, 23
element_rect(), *24*, *129*, *135, 136*
element_text *(*element*)*, 23
element_text(), *135*

facet_grid *(*facet_wrap*)*, 25
facet_grid(), *6*, *95*
facet_wrap, 25
facet_wrap(), *5, 6*, *95*
filter_by, 26
filter_by(), *6, 7*, *18*, *96*, *125–127*
from_spec *(*as_spec*)*, 11
from_spec(), *132*

glow, 26
glow(), *28*, *41*, *53*, *64*, *67*, *75*, *81–83*, *128*, *135*
gradients, 27
grDevices::hcl.colors(), *105*, *116*
gt::gt(), *28, 29*
gt_vsparkline, 28

gt_vsparkline(), *93*
guide_colorbar *(*guides*)*, 29
guide_colorsteps *(*guides*)*, 29
guide_colourbar *(*guides*)*, 29
guide_coloursteps *(*guides*)*, 29
guide_coloursteps(), *31*
guide_legend *(*guides*)*, 29
guide_legend(), *29*
guide_none *(*guides*)*, 29
guide_none(), *29*
guides, 29

hconcat *(*concat*)*, 15
hconcat(), *15*

inset, 32
inset(), *150*, *152*
inspect_source, 33
inspect_source(), *34*, *96*
interaction_model, 34

labs, 35
labs(), *80*, *87*, *95*
lims, 36
linear_gradient *(*gradients*)*, 27
linear_gradient(), *84, 85*, *135*, *147*

map_decorations, 37
mark_abline, 38
mark_area, 40
mark_bar *(*mark_point*)*, 63
mark_bin2d *(*mark_tile*)*, 77
mark_boxplot, 41
mark_boxplot(), *61*
mark_chord *(*vchord*)*, 139
mark_compass *(*map_decorations*)*, 37
mark_compass(), *21*
mark_contour, 43
mark_contour_filled *(*mark_contour*)*, 43
mark_count, 44
mark_crossbar *(*mark_boxplot*)*, 41
mark_datashade, 45
mark_datashade(), *41*, *64*, *67*
mark_density *(*mark_tile*)*, 77
mark_density(), *5, 6*
mark_donut *(*mark_pie*)*, 61
mark_donut(), *19, 20*
mark_dotplot *(*mark_violin*)*, 78
mark_ecdf, 46
mark_edge_bundle *(*mark_graph*)*, 49
mark_edge_text *(*mark_graph*)*, 49
mark_edges *(*mark_graph*)*, 49
mark_edges(), *95*, *108, 109*, *126*, *142, 143*
mark_ellipse, 48
mark_errorbar *(*mark_boxplot*)*, 41
mark_flow_map *(*mark_graph*)*, 49
mark_function *(*mark_abline*)*, 38
mark_graph, 49
mark_halfeye, 55
mark_halfeye(), *66*
mark_hex *(*mark_tile*)*, 77
mark_hierarchy *(*vhierarchy*)*, 144
mark_histogram, 57
mark_histogram(), *5, 6*
mark_hull *(*mark_ellipse*)*, 48
mark_image, 59
mark_interval *(*mark_halfeye*)*, 55
mark_label *(*mark_text*)*, 73
mark_line *(*mark_point*)*, 63
mark_line(), *113*, *147*
mark_linerange *(*mark_boxplot*)*, 41
mark_node_hull *(*mark_graph*)*, 49
mark_node_pie *(*mark_graph*)*, 49
mark_node_text *(*mark_graph*)*, 49
mark_node_text(), *142, 143*
mark_nodes *(*mark_graph*)*, 49
mark_nodes(), *142, 143*
mark_outlier_label, 60
mark_pie, 61
mark_pie(), *19, 20*, *155*
mark_point, 63
mark_point(), *5*, *39*, *47*, *53*, *62*, *66*, *70*, *79*, *96*, *129*, *136*, *147*
mark_pointrange *(*mark_boxplot*)*, 41
mark_qq *(*mark_ecdf*)*, 46
mark_qq_line *(*mark_ecdf*)*, 46
mark_raincloud, 66
mark_raster *(*mark_tile*)*, 77
mark_ribbon *(*mark_area*)*, 40
mark_ridgeline *(*mark_violin*)*, 78
mark_rug *(*mark_ecdf*)*, 46
mark_rule *(*mark_point*)*, 63
mark_rule(), *147*
mark_sankey *(*vsankey*)*, 149
mark_scalebar *(*map_decorations*)*, 37
mark_scalebar(), *21*
mark_segment, 67

mark_segment(), *113*
mark_series_label, 68
mark_sf, 69
mark_sf(), *20, 21*, *38*, *71, 72*, *111*
mark_sf_label, 71
mark_signif, 72
mark_smooth *(*mark_histogram*)*, 57
mark_step *(*mark_area*)*, 40
mark_step(), *113*
mark_summary *(*mark_boxplot*)*, 41
mark_text, 73
mark_text(), *54*, *60, 61*, *68, 69*, *72*, *76*
mark_text_path, 76
mark_tile, 77
mark_violin, 78
mcp_serve, 79
md, 80
motion, 81

outline, 82
outline(), *27*, *41*, *53*, *64*, *67*, *75*, *82*, *128*

pattern_checker *(*patterns*)*, 84
pattern_crosshatch *(*patterns*)*, 84
pattern_dot *(*patterns*)*, 84
pattern_grid *(*patterns*)*, 84
pattern_hatch, 83
pattern_hatch(), *89*, *100*
pattern_stripe *(*patterns*)*, 84
pattern_stripe(), *114*, *146, 147*
patterns, 84, *84*
pdf_pages, 86
pdf_pages(), *99*
plot_alt, 87
plot_alt(), *35*
plot_data_uri, 88
plot_lint, 89
plot_lint(), *83, 84*, *100*
plot_manifest, 90
plot_manifest(), *93, 94*
plot_provenance, 91
plot_provenance(), *97–99*
plot_spacer, 92
plot_svg, 92
plot_svg(), *29*, *88*, *90*, *93, 94*
plot_verify, 93
plot_verify(), *90*, *93*
PlotSpec, *5, 6*, *9–14*, *16–23*, *25, 26*, *31–49*, *52*, *55–79*, *86–93*, 94, *98*, *100–102*, *105–116*, *118–121*, *123*, *125*, *127*, *130*, *132, 133*, *135–150*, *152*, *154, 155*
position, 96
position_dodge *(*position*)*, 96
position_dodge(), *64*
position_dodge2 *(*position*)*, 96
position_dodge2(), *64*
position_jitter *(*position*)*, 96
position_jitter(), *64*
position_jitterdodge *(*position*)*, 96
position_jitterdodge(), *64*
position_nudge *(*position*)*, 96
position_nudge(), *64*
position_sina *(*position*)*, 96
position_sina(), *66*
provenance_join, 97
provenance_join(), *33*, *90*, *99*
provenance_payload, 98
provenance_payload(), *33, 34*

radial_gradient *(*gradients*)*, 27
render_all, 99
render_all(), *86*
render_plot, 100
render_plot(), *17*, *35*, *79*, *86–89*, *93, 94*, *99*, *147*, *150*, *152–154*
repeat_, 101
repeat_(), *99*
resolve_scale, 101
resolve_scale(), *95*

scale_alpha, 102
scale_alpha(), *107*
scale_alpha_continuous *(*scale_alpha*)*, 102
scale_alpha_identity *(*scale_color_identity*)*, 106
scale_color_binned *(*scale_fill_binned*)*, 109
scale_color_brewer *(*scale_viridis*)*, 116
scale_color_continuous, 103
scale_color_continuous(), *107*, *116*, *118, 119*
scale_color_discrete *(*scale_color_continuous*)*, 103
scale_color_discrete(), *116*
scale_color_distiller *(*scale_viridis*)*, 116

scale_color_gradient
(scale_color_continuous), 103
scale_color_gradient2
(scale_color_continuous), 103
scale_color_gradientn
(scale_color_continuous), 103
scale_color_identity, 106
scale_color_manual
(scale_color_continuous), 103
scale_color_manual(), *46*
scale_color_steps (scale_fill_binned), 109
scale_color_viridis_c (scale_viridis), 116
scale_color_viridis_d (scale_viridis), 116
scale_colour_brewer (scale_viridis), 116
scale_colour_continuous
(scale_color_continuous), 103
scale_colour_discrete
(scale_color_continuous), 103
scale_colour_distiller (scale_viridis), 116
scale_colour_gradient
(scale_color_continuous), 103
scale_colour_gradient2
(scale_color_continuous), 103
scale_colour_gradientn
(scale_color_continuous), 103
scale_colour_identity
(scale_color_identity), 106
scale_colour_manual
(scale_color_continuous), 103
scale_colour_steps (scale_fill_binned), 109
scale_colour_viridis_c (scale_viridis), 116
scale_colour_viridis_d (scale_viridis), 116
scale_edge, 107
scale_edge_alpha (scale_edge), 107
scale_edge_alpha(), *54*
scale_edge_color (scale_edge), 107
scale_edge_color(), *54, 55*
scale_edge_colour (scale_edge), 107
scale_edge_linetype (scale_edge), 107
scale_edge_linetype(), *52*, *54*
scale_edge_width, 108
scale_edge_width(), *52*, *54, 55*, *108*, *113*
scale_fill_binned, 109
scale_fill_binned(), *70*
scale_fill_brewer (scale_viridis), 116
scale_fill_continuous
(scale_color_continuous), 103
scale_fill_continuous(), *111*
scale_fill_discrete
(scale_color_continuous), 103
scale_fill_distiller (scale_viridis), 116
scale_fill_gradient
(scale_color_continuous), 103
scale_fill_gradient2
(scale_color_continuous), 103
scale_fill_gradientn
(scale_color_continuous), 103
scale_fill_identity
(scale_color_identity), 106
scale_fill_manual
(scale_color_continuous), 103
scale_fill_steps (scale_fill_binned), 109
scale_fill_viridis_c (scale_viridis), 116
scale_fill_viridis_d (scale_viridis), 116
scale_linetype, 111
scale_linetype(), *107, 108*
scale_linetype_discrete
(scale_linetype), 111
scale_linetype_identity
(scale_color_identity), 106
scale_linewidth, 112
scale_linewidth(), *40, 41*, *64*, *67, 68*
scale_linewidth_continuous
(scale_linewidth), 112
scale_pattern, 114
scale_pattern(), *83, 84*, *89*, *100*
scale_shape, 114
scale_shape(), *64*, *114*
scale_shape_identity
(scale_color_identity), 106
scale_size, 115
scale_size_area (scale_size), 115
scale_size_identity
(scale_color_identity), 106
scale_viridis, 116

scale_x_binned, 119
scale_x_continuous, 120
scale_x_continuous(), *36*, *96*, *123*, *124*
scale_x_date, 121
scale_x_datetime *(*scale_x_date*)*, 121
scale_x_discrete *(*scale_x_continuous*)*, 120
scale_x_time *(*scale_x_date*)*, 121
scale_y_binned *(*scale_x_binned*)*, 119
scale_y_continuous *(*scale_x_continuous*)*, 120
scale_y_continuous(), *123*
scale_y_date *(*scale_x_date*)*, 121
scale_y_datetime *(*scale_x_date*)*, 121
scale_y_discrete *(*scale_x_continuous*)*, 120
scale_y_time *(*scale_x_date*)*, 121
scales::breaks_width(), *123*
scales::transform_log10(), *121*, *124*
sec_axis, 123
sec_axis(), *121*
select_interval *(*select_point*)*, 126
select_interval(), *6*, *13*, *26*, *96*
select_neighbours, 125
select_point, 126
select_point(), *6*, *7*, *18*, *34*, *96*, *125*, *126*
selection, *13*, *17*
set_mask *(*clip*)*, 13
set_mask(), *96*
set_theme *(*theme_gray*)*, 134
shadow, 128
shadow(), *27*, *41*, *53*, *54*, *64*, *67*, *75*, *82*, *83*
sketch, 129
sketch(), *12*, *24*, *41*, *42*, *44*, *45*, *48*, *53*, *56*, *58*, *62*, *64*, *67*, *70*, *73*, *75*, *78*, *135*, *136*
spec_diagnose, 130
spec_diagnose(), *79*
spec_fields, 131
spec_fields(), *11*, *79*, *80*, *130*, *132*, *148*
spec_from_json *(*spec_to_json*)*, 132
spec_from_json(), *11*, *131*
spec_from_vegalite *(*spec_to_vegalite*)*, 133
spec_schema, 131
spec_schema(), *79*, *80*, *130–132*, *148*
spec_to_json, 132
spec_to_json(), *11*, *12*, *132*, *133*
spec_to_vegalite, 133
spec_to_vegalite(), *11*, *12*
stats::cutree(), *142*
stats::hclust(), *141*
stats::qnorm, *47*

theme *(*theme_gray*)*, 134
theme(), *15*, *23*, *25*, *95*, *135*
theme_bw *(*theme_gray*)*, 134
theme_classic *(*theme_gray*)*, 134
theme_cyberpunk *(*theme_gray*)*, 134
theme_cyberpunk(), *26–28*, *136*
theme_dark *(*theme_gray*)*, 134
theme_gray, 134
theme_gray(), *136*
theme_light *(*theme_gray*)*, 134
theme_linedraw *(*theme_gray*)*, 134
theme_minimal *(*theme_gray*)*, 134
theme_sketch, 135
theme_sketch(), *24*, *41*, *42*, *44*, *45*, *48*, *56*, *58*, *64*, *67*, *73*, *75*, *78*, *129*
theme_void *(*theme_gray*)*, 134
transition_reveal, 137
transition_reveal(), *96*
transition_states, 137
transition_states(), *8*, *9*, *22*, *23*, *96*, *137*, *139*
transition_time, 139
transition_time(), *96*, *137*

vchord, 139
vconcat *(*concat*)*, 15
vconcat(), *15*
vdendrogram, 141
vellum::as_mask(), *14*
vellum::as_vellum_scene(), *87*, *89*, *94*, *147*
vellum::datashade(), *45*, *46*
vellum::datashade_segments(), *53*
vellum::describe(), *87*
vellum::dynspread(), *46*
vellum::font_pin(), *90*
vellum::linear_gradient(), *27*, *28*
vellum::md(), *75*, *80*
vellum::radial_gradient(), *27*, *28*
vellum::render(), *17*, *99*, *100*
vellum::scene_model(), *91*, *97*, *98*
vellum::sketch(), *129*
vellum::spread(), *46*

vellum::svg_grob(), *115*
vellum::text_grob(), *80*
vellum::vl_arrow(), *52*
vellum::vl_hatch(), *83, 84*
vellum::vl_lint(), *89*
vellum::vl_lint_assert(), *89*
vellum::vl_lint_overlay(), *89*
vellum::vl_lint_rules(), *89*
vellum::vl_path_op(), *70*
vellum::vl_pattern(), *84, 85*, *146, 147*
vellum::vl_scene(), *100*
vgraph, 142
vgraph(), *49*, *52*, *55*, *95*, *107–109*, *113*, *125, 126*, *139*, *142*, *144*, *149*
vhierarchy, 144
vl_pattern, 146
vplot, 147
vplot(), *5*, *40*, *42–45*, *48*, *56, 57*, *59, 60*, *64, 67, 68*, *70–72*, *74*, *76, 77*, *94*, *96*, *100*, *137–139*, *143*
vplot_ask, 148
vplot_ask(), *130, 131*
vplot_from_spec *(*spec_diagnose*)*, 130
vplot_from_spec(), *11, 12*, *80*, *131, 132*, *148*
vsankey, 149
vsankey(), *139*, *144*, *154*
vsparkline, 150
vsparkline(), *10*, *28, 29*, *92, 93*, *152, 153*
vtable, 152
vtable(), *29*, *92, 93*
vvenn, 153
vvenn(), *154, 155*
vwaffle, 154

wrap_plots *(*concat*)*, 15
wrap_plots(), *92*

xlim *(*lims*)*, 36

ylim *(*lims*)*, 36