Package: vellumplot (via r-universe)

July 12, 2026

Title A Grammar of Graphics on the 'vellum' Backend

Version 0.3.0.9000

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, rlang, S7, scales, stats, utils, vellum (>= 0.2.0.9000)

Remotes r-vellum/vellum

Suggests classInt, graphlayouts, igraph, isoband, MASS, sf, testthat (>= 3.0.0), vctrs, withr

Config/testthat/edition 3

Collate 'vellumplot-package.R' 'classes.R' 'print.R' 'alt.R' 'marks.R' 'annotate.R' 'marginal.R' 'compile-marks.R' 'elements.R' 'theme-tree.R' 'theme.R' 'compile-layout.R' 'compile-guides.R' 'compile-resolve.R' 'coord.R' 'compile-train.R' 'facet.R' 'compile-facet.R' 'seam.R' 'concat.R' 'compile-composition.R' 'compile-position.R' 'compile-sf.R' 'compile-stat.R'

'effects.R' 'guides.R' 'labs.R' 'provenance.R' 'reexports.R' 'repeat.R' 'scales.R' 'scales-binned.R' 'vgraph.R' 'vplot.R' 'zzz.R'

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

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

Date/Publication 2026-07-12 19:40:12 UTC

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

RemoteRef HEAD

RemoteSha 0eb7f89a505356a446f8c7783366bd5a2c2d8df8

Contents

add_marginal . . . 3
after_stat . . . 5
annotate . . . 5
area . . . 6
compose_annotation . . . 7
concat . . . 8
coord_cartesian . . . 9
coord_polar . . . 10
coord_sf . . . 11
coord_trans . . . 12
element . . . 13
facet_wrap . . . 14
glow . . . 15
gradients . . . 16
guides . . . 17
inset . . . 18
labs . . . 19
lims . . . 20
mark_area . . . 21
mark_boxplot . . . 22
mark_contour . . . 23
mark_datashade . . . 24
mark_ecdf . . . 26
mark_graph . . . 27
mark_histogram . . . 29
mark_pie . . . 30
mark_point . . . 31
mark_segment . . . 34
mark_sf . . . 35
mark_text . . . 36
mark_tile . . . 38
mark_violin . . . 39
md . . . 40

outline . . . 41
plot_alt . . . 41
plot_provenance . . . 42
plot_spacer . . . 43
PlotSpec . . . 44
render_plot . . . 45
repeat_ . . . 46
resolve_scale . . . 47
scale_alpha . . . 47
scale_color_continuous . . . 48
scale_color_identity . . . 50
scale_edge_width . . . 51
scale_fill_binned . . . 52
scale_linetype . . . 53
scale_shape . . . 54
scale_size . . . 54
scale_x_binned . . . 55
scale_x_continuous . . . 56
scale_x_date . . . 57
shadow . . . 59
sketch . . . 60
theme_gray . . . 61
theme_sketch . . . 62
vgraph . . . 64
vplot . . . 65

Index **66**

`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")
```

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).

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"`, and `"rect"`.

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".
x, y	Position (text/label/point; 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).

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)
```

---

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)))
```

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)
```

---

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.

Usage

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

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.

Value

The modified PlotSpec.

See Also

mark_pie(), mark_donut()

Examples

```
vplot(mtcars) |> mark_bar(x = factor(cyl)) |> coord_polar(theta = "x")
```

---

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)
```

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.

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()`

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")

## 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")
```

`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,
  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`, `colour`, `color`, `hjust`, `vjust`, `angle`, `lineheight`, `margin`	Text properties. `color` is an alias for `colour`; `size` is in points; `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)
```

---

glow *Neon glow layer effect*

Description

A render effect for stroked and point marks, in the spirit of mplcyberpunk: the mark is drawn as several widened, low-opacity copies composited additively (a "screen" blend) beneath the crisp original, producing 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	Extra visual spread, in millimetres, added to the stroke width (or point diameter) at the outermost copy.
layers	Number of stacked halo copies.
alpha	Opacity of each copy (they accumulate toward the centre).
blend	Blend mode compositing the halo copies, typically "screen" or "lighten" (any CSS mix-blend-mode name).
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(), and mark_nodes(); other marks reject it with an error.

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 (radial), units, and extend. 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))
```

---

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, or `guide_legend()` to tweak it (reverse the key order, override the title). 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)
```

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 (discrete legends).

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))
```

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)
```

`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))
```

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",
  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`.
`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".
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.

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)
```

Arguments

plot	A PlotSpec (from vplot()).
...	Encodings (tidy-eval): x, y for boxplot/summary; x, ymin, ymax for errorbar/linerange; plus color/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.
width	For mark_errorbar(), the cap width as a fraction of the band.
fun	For mark_summary(), the aggregation function (default mean).

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. Requires the **isoband** package.

Usage

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

mark_contour_filled(
  plot,
  ...,
  bins = 10,
  binwidth = NULL,
  breaks = NULL,
```

```
  n = 100,
  blend = NULL,
  data = NULL
)
```

Arguments

plot	A PlotSpec (from vplot()).
...	Encodings (tidy-eval): x, y (+ optional z surface, color / fill, linewidth).
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).
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_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. Per-point colour/size aesthetics do not apply; cell colour encodes density.

Usage

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

Arguments

plot	A PlotSpec (from vplot()).
...	Encodings; x and y are required.
width, height	Aggregation grid size in cells (output raster pixels).
colors	Two or more colours forming the low-to-high density ramp. For an additive per-category overlay, ramp from a transparent/black low end to the category hue and composite with blend = "screen" (see details).
how	Density-to-colour mapping: "eq_hist" (default), "log", "cbrt", or "linear".
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

Categorical shading (à la datashader's count_cat) has no single-call form, but is reproduced by stacking one datashade layer per category — each with a colors ramp from black to its hue — composited with blend = "screen", so overlapping densities mix additively.

Value

The modified PlotSpec.

Examples

```
n <- 1e5
d <- data.frame(x = rnorm(n), y = rnorm(n))
vplot(d) |> mark_datashade(x = x, y = y)
```

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, data = NULL)

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

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

mark_qq_line(plot, ..., distribution = "qnorm", blend = 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`.
`blend, 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_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), and `mark_node_text()` the vertex labels. Draw order is fixed regardless of the order you pipe them: edges under nodes under labels. Edges default to the edge table (`vgraph()`'s `edge_data`), nodes and 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,
  arrow = FALSE,
  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,
  blend = NULL,
  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, 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.
arrow	For mark_edges(), TRUE to draw an arrowhead at each edge's target end (directed graphs). 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.
blend	Optional blend mode (see mark_point()).
effects	A list of layer render effects (glow(), outline(), shadow()) applied to the mark at draw time.
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.
size, shape	For mark_nodes(), the node size (mm) / shape; a constant or a mapped expression.
fill, color, alpha	Convenience aesthetics; a constant or a mapped expression. For nodes, fill (or color) is the marker colour.
label	For mark_node_text(), the label expression (default the vertex name).

Details

These are thin over the point / segment / text marks; igraph need not be installed to use them (only vgraph() needs it).

Value

The modified PlotSpec.

See Also

vgraph(), scale_edge_width()

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_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 (`"lm"` for now) of `y` on `x` 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 = "lm",
  se = TRUE,
  level = 0.95,
  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; "lm" (linear) for now.
se	Draw a confidence ribbon around the smooth?
level	Confidence level for the ribbon.

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)
```

---

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 (hole, 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, ..., sketch = NULL, data = NULL)

mark_donut(
  plot,
  value,
  fill = NULL,
  hole = 0.5,
  ...,
  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).
sketch	A sketch() spec giving the layer a hand-drawn look, or NULL (default) to inherit.
data	Optional per-layer data frame.
hole	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, hole = 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,
  ...,
  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.
size, shape	Convenience arguments for the point size (in mm) / shape; may be a constant or a mapped expression. One of "circle", "square", "triangle", "diamond", "plus", "cross".
position	Position adjustment: "identity" (default), "jitter" (points), or "stack" / "dodge" / "fill" (bars).
auto	For mark_point(), when TRUE and the layer has very many rows, automatically render it as a datashaded density raster (see mark_datashade()) instead of individual markers.
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.

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_segment` *Segment mark*

Description

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

Usage

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

Arguments

`plot`	A PlotSpec (from `vplot()`).
`...`	Encodings (tidy-eval): `x`, `y`, `xend`, `yend` (+ `color`, `linewidth`, `alpha`).
`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.

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_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",
  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"`.
`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_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,
  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",
  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.
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.)

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_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,
  scale = 1.4,
  blend = NULL,
  sketch = NULL,
  data = NULL
)

mark_dotplot(plot, ..., binwidth = NULL, blend = 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()`).
`scale`	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)
```

---

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^)"))
```

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()))
```

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_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)
```

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 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
)
```

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.

Value

A PlotSpec.

See Also

vplot(), mark_point(), scale_x_continuous()

---

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)
```

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.

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
```

---

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. 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,
  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_fill_continuous(
  plot,
  palette = NULL,
  breaks = NULL,
  labels = NULL,
  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_colour_continuous(
  plot,
  palette = 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)
```

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.

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(), the endpoint colours.

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_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
)
```

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.

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_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 (each one of `"circle"`, `"square"`, `"triangle"`, `"diamond"`, `"plus"`, `"cross"`), or `NULL` for the default.
`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))
```

`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.

Usage

```
scale_size(plot, range = NULL, 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.

Value

The modified PlotSpec.

Examples

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

---

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
)

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

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 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")
```

---

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))
```

---

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_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)
```

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", ..., 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).
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.

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)

## End(Not run)
```

---

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)
```

Index

add_marginal, 3
add_marginal(), *45*
after_stat, 5
annotate, 5
area, 6
area(), *9*

compose_annotation, 7
compose_annotation(), *9*
concat, 8
concat(), *7*, *43*, *46*
coord_cartesian, 9
coord_cartesian(), *44*
coord_equal *(*coord_cartesian*)*, 9
coord_fixed *(*coord_cartesian*)*, 9
coord_fixed(), *4*, *44*
coord_flip *(*coord_cartesian*)*, 9
coord_flip(), *3*, *4*, *44*
coord_polar, 10
coord_polar(), *4*, *30*, *31*
coord_sf, 11
coord_sf(), *35*, *36*
coord_trans, 12

effect, *60*
element, 13
element_blank *(*element*)*, 13
element_blank(), *62*
element_line *(*element*)*, 13
element_line(), *14*, *60–63*
element_rect *(*element*)*, 13
element_rect(), *14*, *60*, *62*, *63*
element_text *(*element*)*, 13
element_text(), *62*

facet_grid *(*facet_wrap*)*, 14
facet_grid(), *4*, *44*
facet_wrap, 14
facet_wrap(), *3*, *4*, *44*

glow, 15
glow(), *16*, *22*, *28*, *33*, *34*, *41*, *59*, *60*, *62*
gradients, 16
grDevices::hcl.colors(), *49*
guide_legend *(*guides*)*, 17
guide_legend(), *17*
guide_none *(*guides*)*, 17
guide_none(), *17*
guides, 17

hconcat *(*concat*)*, 8
hconcat(), *7*

inset, 18

labs, 19
labs(), *40–42*, *45*
lims, 20
linear_gradient *(*gradients*)*, 16
linear_gradient(), *62*

mark_area, 21
mark_bar *(*mark_point*)*, 31
mark_bin2d *(*mark_tile*)*, 38
mark_boxplot, 22
mark_contour, 23
mark_contour_filled *(*mark_contour*)*, 23
mark_datashade, 24
mark_datashade(), *32*
mark_density *(*mark_tile*)*, 38
mark_density(), *4*
mark_donut *(*mark_pie*)*, 30
mark_donut(), *10*, *11*
mark_dotplot *(*mark_violin*)*, 39
mark_ecdf, 26
mark_edges *(*mark_graph*)*, 27
mark_edges(), *44*, *51*, *64*
mark_errorbar *(*mark_boxplot*)*, 22
mark_graph, 27
mark_hex *(*mark_tile*)*, 38
mark_histogram, 29

mark_histogram(), *4*
mark_label *(*mark_text*)*, 36
mark_line *(*mark_point*)*, 31
mark_line(), *65*
mark_linerange *(*mark_boxplot*)*, 22
mark_node_text *(*mark_graph*)*, 27
mark_node_text(), *64*
mark_nodes *(*mark_graph*)*, 27
mark_nodes(), *64*
mark_pie, 30
mark_pie(), *10, 11*
mark_point, 31
mark_point(), *3*, *26*, *28*, *35*, *39*, *45*, *61*, *63*, *65*
mark_qq *(*mark_ecdf*)*, 26
mark_qq_line *(*mark_ecdf*)*, 26
mark_raster *(*mark_tile*)*, 38
mark_ribbon *(*mark_area*)*, 21
mark_ridgeline *(*mark_violin*)*, 39
mark_rug *(*mark_ecdf*)*, 26
mark_rule *(*mark_point*)*, 31
mark_rule(), *65*
mark_segment, 34
mark_sf, 35
mark_sf(), *11, 12*, *53*
mark_smooth *(*mark_histogram*)*, 29
mark_step *(*mark_area*)*, 21
mark_summary *(*mark_boxplot*)*, 22
mark_text, 36
mark_tile, 38
mark_violin, 39
md, 40

outline, 41
outline(), *16*, *22*, *28*, *33, 34*, *60*

plot_alt, 41
plot_alt(), *19*
plot_provenance, 42
plot_spacer, 43
PlotSpec, *4*, *6*, *8–13*, *15*, *17–40*, *42*, 44, *45–59*, *62–65*

radial_gradient *(*gradients*)*, 16
render_plot, 45
render_plot(), *9*, *19*, *42*, *44*, *65*
repeat_, 46
resolve_scale, 47
resolve_scale(), *44*

scale_alpha, 47
scale_alpha_continuous *(*scale_alpha*)*, 47
scale_alpha_identity *(*scale_color_identity*)*, 50
scale_color_binned *(*scale_fill_binned*)*, 52
scale_color_continuous, 48
scale_color_discrete *(*scale_color_continuous*)*, 48
scale_color_gradient *(*scale_color_continuous*)*, 48
scale_color_identity, 50
scale_color_manual *(*scale_color_continuous*)*, 48
scale_colour_continuous *(*scale_color_continuous*)*, 48
scale_colour_discrete *(*scale_color_continuous*)*, 48
scale_colour_gradient *(*scale_color_continuous*)*, 48
scale_colour_identity *(*scale_color_identity*)*, 50
scale_colour_manual *(*scale_color_continuous*)*, 48
scale_edge_width, 51
scale_edge_width(), *28*
scale_fill_binned, 52
scale_fill_binned(), *36*
scale_fill_continuous *(*scale_color_continuous*)*, 48
scale_fill_continuous(), *53*
scale_fill_discrete *(*scale_color_continuous*)*, 48
scale_fill_gradient *(*scale_color_continuous*)*, 48
scale_fill_identity *(*scale_color_identity*)*, 50
scale_fill_manual *(*scale_color_continuous*)*, 48
scale_linetype, 53
scale_linetype_discrete *(*scale_linetype*)*, 53
scale_linetype_identity *(*scale_color_identity*)*, 50
scale_shape, 54
scale_shape_identity *(*scale_color_identity*)*, 50
scale_size, 54
scale_size_identity

(scale_color_identity), 50
scale_x_binned, 55
scale_x_continuous, 56
scale_x_continuous(), 20, 45
scale_x_date, 57
scale_x_datetime (scale_x_date), 57
scale_x_discrete (scale_x_continuous), 56
scale_x_time (scale_x_date), 57
scale_y_binned (scale_x_binned), 55
scale_y_continuous (scale_x_continuous), 56
scale_y_date (scale_x_date), 57
scale_y_datetime (scale_x_date), 57
scale_y_discrete (scale_x_continuous), 56
scale_y_time (scale_x_date), 57
scales::breaks_width(), 59
scales::transform_log10(), 57
set_theme (theme_gray), 61
shadow, 59
shadow(), 16, 22, 28, 33, 34, 41
sketch, 60
sketch(), 14, 22, 23, 28, 30, 31, 33–35, 37, 38, 62, 63
stats::qnorm, 26
strftime(), 59
summary(), 44

theme (theme_gray), 61
theme(), 7, 13, 14, 45, 62
theme_bw (theme_gray), 61
theme_classic (theme_gray), 61
theme_cyberpunk (theme_gray), 61
theme_cyberpunk(), 15, 16, 63
theme_gray, 61
theme_gray(), 63
theme_minimal (theme_gray), 61
theme_sketch, 62
theme_sketch(), 14, 22, 23, 30, 33, 34, 37, 38, 60, 61
theme_void (theme_gray), 61

vconcat (concat), 8
vconcat(), 7
vellum::as_vellum_scene(), 42, 44, 65
vellum::datashade(), 24
vellum::describe(), 42
vellum::linear_gradient(), 16
vellum::md(), 37, 40
vellum::radial_gradient(), 16
vellum::render(), 9, 45
vellum::scene_model(), 42, 43
vellum::sketch(), 60
vellum::text_grob(), 40
vellum::vl_scene(), 45
vgraph, 64
vgraph(), 27, 28, 44, 51
vplot, 65
vplot(), 4, 21, 22, 24, 25, 29, 32, 34, 35, 37, 38, 44, 45, 64

wrap_plots (concat), 8
wrap_plots(), 43

xlim (lims), 20

ylim (lims), 20