Package 'vegalite'

October 12, 2022

Type Package

Title Tools to Encode Visualizations with the 'Grammar of Graphics'-Like 'Vega-Lite' 'Spec'

Version 0.6.1

Date 2016-03-21

Maintainer Bob Rudis <bob@rudis.net>

Description The 'Vega-Lite' 'JavaScript' framework provides a higher-level grammar for visual analysis, akin to 'ggplot' or 'Tableau', that generates complete 'Vega' specifications. Functions exist which enable building a valid 'spec' from scratch or importing a previously created 'spec' file. Functions also exist to export 'spec' files and to generate code which will enable plots to be embedded in properly configured web pages. The default behavior is to generate an 'htmlwidget'.

URL http://github.com/hrbrmstr/vegalite

BugReports https://github.com/hrbrmstr/vegalite/issues

License AGPL + file LICENSE

Encoding UTF-8

Suggests testthat, knitr, rmarkdown

Depends R (>= 3.0.0)

Imports jsonlite, htmlwidgets (>= 0.6), htmltools, magrittr, digest, tools, clipr, utils, webshot, base64, stats

RoxygenNote 5.0.1

VignetteBuilder knitr

NeedsCompilation no

Author Bob Rudis [aut, cre],
Kanit Wongsuphasawat, [aut] (Vega-Lite library),
Jeffrey Heer [aut] (Vega-Lite library),
Arvind Satyanarayan [aut] (Vega-Lite library),
Mike Bostock [aut] (D3 library)

Repository CRAN

Date/Publication 2016-03-22 23:51:16

R topics documented:

`vegalite-package` *Create Vega-Lite specs using htmlwidget idioms*

Description

Creation of Vega-Lite spec charts is virtually 100% feature complete. Some of the parameters to functions are only documented in TypeScript source code which will take a bit of time to wade through. All the visualizations you find in the Vega-Lite Gallery work.

Functions also exist which enable creation of widgets from a JSON spec and turning a `vegalite` package created object into a JSON spec.

Details

You start by calling `vegalite()` which allows you to setup core configuration options, including whether you want to display links to show the source and export the visualization. You can also set the background here and the `viewport_width` and `viewport_height`. Those are very important as they control the height and width of the widget and also the overall area for the chart. This does *not* set the height/width of the actual chart. That is done with `cell_size()`.

Once you instantiate the widget, you need to `add_data()` which can be `data.frame`, local CSV, TSV or JSON file (that convert to `data.frames`) or a non-realive URL (wich will not be read and converted but will remain a URL in the Vega-Lite spec.

You then need to `encode_x()` & `encode_y()` variables that map to columns in the data spec and choose one `mark_...()` to represent the encoding.

Here's a sample, basic Vega-Lite widget:

```
  dat <- jsonlite::fromJSON('[
      {"a": "A","b": 28}, {"a": "B","b": 55}, {"a": "C","b": 43},
      {"a": "D","b": 91}, {"a": "E","b": 81}, {"a": "F","b": 53},
      {"a": "G","b": 19}, {"a": "H","b": 87}, {"a": "I","b": 52}
    ]')

vegalite()
    add_data(dat)
    encode_x("a", "ordinal")
    encode_y("b", "quantitative")
    mark_bar() -> vl

vl
```

That is the minimum set of requirements for a basic Vega-Lite spec and will create a basic widget.

You can also convert that R widget object `to_spec()` which will return the JSON for the Vega-Lite spec (allowing you to use it outside of R).

```
to_spec(vl)
```

```
{
  "description": "",
  "data": {
    "values": [
      { "a": "A", "b": 28 }, { "a": "B", "b": 55 }, { "a": "C", "b": 43 },
      { "a": "D", "b": 91 }, { "a": "E", "b": 81 }, { "a": "F", "b": 53 },
      { "a": "G", "b": 19 }, { "a": "H", "b": 87 }, { "a": "I", "b": 52 }
    ]
  },
  "mark": "bar",
  "encoding": {
```

```
    "x": {
      "field": "a",
      "type": "nominal"
    },
    "y": {
      "field": "b",
      "type": "quantitative"
    }
  },
  "config": [],
  "embed": {
    "renderer": "svg",
    "actions": {
      "export": false,
      "source": false,
      "editor": false
    }
  }
}
```

If you already have a Vega-Lite JSON spec that has embedded data or a non-realtive URL, you can create a widget from it via `from_spec()` by passing in the full JSON spec or a URL to a full JSON spec.

If you're good with HTML (etc) and want a more lightweight embedding options, you can also use `embed_spec` which will scaffold a minimum `div` + `script` source and embed a spec from a `vegalite` object.

If you like the way Vega-Lite renders charts, you can also use them as static images in PDF knitted documents with the new `capture_widget` function. (NOTE that as of this writing, you can just use the development version of `knitr` instead of this function.)

Author(s)

Bob Rudis (@hrbrmstr)

`add_data` *Add data to a Vega-Lite spec*

Description

Vega-Lite is more lightweight than full Vega. However, the spec is flexible enough to support embedded data or using external sources that are in JSON, CSV or TSV format.

Usage

```
add_data(vl, source, format_type = NULL)
```

Arguments

vl	a Vega-Lite object
source	you can specify a (fully qualified) URL or an existing data.frame (or ts) object or a reference to a local file. For the URL case, the url component of data will be set. You can help Vega-Lite out by giving it a hint for the data type with format_type but it is not required. For the local data.frame case it will embed the data into the spec. For the case where a local file is specified, it will be read in (either a JSON file, CSV file or TSV file) and converted to a data.frame and embedded.
format_type	if source is a URL, this should be one of json, csv or tsv). It is not required and it is ignored if source is not a URL.

References

Vega-Lite Data spec

Examples

```
dat <- jsonlite::fromJSON('[
    {"a": "A","b": 28}, {"a": "B","b": 55}, {"a": "C","b": 43},
    {"a": "D","b": 91}, {"a": "E","b": 81}, {"a": "F","b": 53},
    {"a": "G","b": 19}, {"a": "H","b": 87}, {"a": "I","b": 52}
  ]')

vegalite() %>%
  add_data(dat) %>%
  encode_x("a", "ordinal") %>%
  encode_y("b", "quantitative") %>%
  mark_bar()
```

---

add_filter *Add a filter*

Description

Add a filter

Usage

```
add_filter(vl, expr)
```

Arguments

vl	Vega-Lite object created by vegalite
expr	Vega Expression for filtering data items (or rows). Each datum object can be referred using bound variable datum. For example, setting expr to "datum.datum.b2 > 60" would make the output data includes only items that have values in the field b2 over 60.

Examples

```
vegalite(viewport_height=200, viewport_width=200) %>%
  cell_size(200, 200) %>%
  add_data("https://vega.github.io/vega-editor/app/data/population.json") %>%
  add_filter("datum.year == 2000") %>%
  calculate("gender", 'datum.sex == 2 ? "Female" : "Male"') %>%
  encode_x("gender", "nominal") %>%
  encode_y("people", "quantitative", aggregate="sum") %>%
  encode_color("gender", "nominal") %>%
  scale_x_ordinal(band_size=6) %>%
  scale_color_nominal(range=c("#EA98D2", "#659CCA")) %>%
  facet_col("age", "ordinal", padding=4) %>%
  axis_x(remove=TRUE) %>%
  axis_y(title="population", grid=FALSE) %>%
  axis_facet_col(orient="bottom", axisWidth=1, offset=-8) %>%
  facet_cell(stroke_width=0) %>%
  mark_bar()
```

---

axis_facet_col *General axis setttings (column facet)*

Description

Axes provide axis lines, ticks and labels to convey how a spatial range represents a data range. Simply put, axes visualize scales.

By default, Vega-Lite automatically creates axes for x, y, row, and column channels when they are encoded. Axis can be customized via the axis property of a channel definition.

Usage

```
axis_facet_col(vl, axisWidth = 0, layer = NULL, offset = NULL,
  grid = FALSE, labels = TRUE, labelAngle = NULL, labelAlign = NULL,
  labelBaseline = NULL, labelMaxLength = 25, shortTimeLabels = NULL,
  subdivide = NULL, ticks = NULL, tickPadding = NULL, tickSize = 0,
  tickSizeMajor = NULL, tickSizeMinor = NULL, tickSizeEnd = NULL,
  title = "", titleOffset = NULL, titleMaxLength = NULL,
  characterWidth = 6, orient = NULL, format = NULL, remove = FALSE)
```

Arguments

`vl`	Vega-Lite object
`axisWidth, layer, offset, grid, labels, labelAngle, labelAlign, labelBaseline`	see axis docs & axis base config
`labelMaxLength, shortTimeLabels, subdivide, ticks, tickPadding, tickSize`	see axis docs & axis base config
`tickSizeMajor, tickSizeMinor, tickSizeEnd, title, titleOffset, titleMaxLength`	see axis docs & axis base config

`characterWidth, orient, format, remove`
see axis docs & axis base config

References

Vega-List Axis spec

Examples

```
vegalite() %>%
  add_data("https://vega.github.io/vega-editor/app/data/population.json") %>%
  add_filter("datum.year == 2000") %>%
  calculate("gender", 'datum.sex == 2 ? "Female" : "Male"') %>%
  encode_x("gender", "nominal") %>%
  encode_y("people", "quantitative", aggregate="sum") %>%
  encode_color("gender", "nominal") %>%
  scale_x_ordinal(band_size=6) %>%
  scale_color_nominal(range=c("#EA98D2", "#659CCA")) %>%
  facet_col("age", "ordinal", padding=4) %>%
  axis_x(remove=TRUE) %>%
  axis_y(title="population", grid=FALSE) %>%
  axis_facet_col(orient="bottom", axisWidth=1, offset=-8) %>%
  facet_cell(stroke_width=0) %>%
  mark_bar()
```

---

axis_facet_row *General axis setttings (row facets)*

Description

Axes provide axis lines, ticks and labels to convey how a spatial range represents a data range. Simply put, axes visualize scales.

By default, Vega-Lite automatically creates axes for x, y, row, and column channels when they are encoded. Axis can be customized via the axis property of a channel definition.

Usage

```
axis_facet_row(vl, axisWidth = 0, layer = NULL, offset = NULL,
  grid = FALSE, labels = TRUE, labelAngle = NULL, labelAlign = NULL,
  labelBaseline = NULL, labelMaxLength = 25, shortTimeLabels = NULL,
  subdivide = NULL, ticks = NULL, tickPadding = NULL, tickSize = 0,
  tickSizeMajor = NULL, tickSizeMinor = NULL, tickSizeEnd = NULL,
  title = "", titleOffset = NULL, titleMaxLength = NULL,
  characterWidth = 6, orient = NULL, format = NULL, remove = FALSE)
```

Arguments

vl	Vega-Lite object
axisWidth, layer, offset, grid, labels, labelAngle, labelAlign, labelBaseline	see axis docs & axis base config
labelMaxLength, shortTimeLabels, subdivide, ticks, tickPadding, tickSize	see axis docs & axis base config
tickSizeMajor, tickSizeMinor, tickSizeEnd, title, titleOffset, titleMaxLength	see axis docs & axis base config
characterWidth, orient, format, remove	see axis docs & axis base config

References

Vega-List Axis spec

---

axis_x *General axis setttings (x-axis)*

Description

Axes provide axis lines, ticks and labels to convey how a spatial range represents a data range. Simply put, axes visualize scales.

By default, Vega-Lite automatically creates axes for x, y, row, and column channels when they are encoded. Axis can be customized via the axis property of a channel definition.

Usage

```
axis_x(vl, axisWidth = NULL, layer = NULL, offset = NULL, grid = NULL,
  labels = TRUE, labelAngle = NULL, labelAlign = NULL,
  labelBaseline = NULL, labelMaxLength = 25, shortTimeLabels = NULL,
  subdivide = NULL, ticks = NULL, tickPadding = NULL, tickSize = NULL,
  tickSizeMajor = NULL, tickSizeMinor = NULL, tickSizeEnd = NULL,
  title = "", titleOffset = NULL, titleMaxLength = NULL,
  characterWidth = 6, orient = NULL, format = NULL, remove = FALSE)
```

Arguments

vl	Vega-Lite object
axisWidth, layer, offset, grid, labels, labelAngle, labelAlign, labelBaseline	see axis docs & axis base config
labelMaxLength, shortTimeLabels, subdivide, ticks, tickPadding, tickSize	see axis docs & axis base config
tickSizeMajor, tickSizeMinor, tickSizeEnd, title, titleOffset, titleMaxLength	see axis docs & axis base config
characterWidth, orient, format, remove	see axis docs & axis base config

References

Vega-List Axis spec

Examples

```
vegalite() %>%
  add_data("https://vega.github.io/vega-editor/app/data/population.json") %>%
  add_filter("datum.year == 2000") %>%
  calculate("gender", 'datum.sex == 2 ? "Female" : "Male"') %>%
  encode_x("gender", "nominal") %>%
  encode_y("people", "quantitative", aggregate="sum") %>%
  encode_color("gender", "nominal") %>%
  scale_x_ordinal(band_size=6) %>%
  scale_color_nominal(range=c("#EA98D2", "#659CCA")) %>%
  facet_col("age", "ordinal", padding=4) %>%
  axis_x(remove=TRUE) %>%
  axis_y(title="population", grid=FALSE) %>%
  axis_facet_col(orient="bottom", axisWidth=1, offset=-8) %>%
  facet_cell(stroke_width=0) %>%
  mark_bar()
```

---

axis_y *General axis setttings (y-axis)*

Description

Axes provide axis lines, ticks and labels to convey how a spatial range represents a data range. Simply put, axes visualize scales.

By default, Vega-Lite automatically creates axes for x, y, row, and column channels when they are encoded. Axis can be customized via the axis property of a channel definition.

Usage

```
axis_y(vl, axisWidth = NULL, layer = NULL, offset = NULL, grid = NULL,
  labels = TRUE, labelAngle = NULL, labelAlign = NULL,
  labelBaseline = NULL, labelMaxLength = 25, shortTimeLabels = NULL,
  subdivide = NULL, ticks = NULL, tickPadding = NULL, tickSize = NULL,
  tickSizeMajor = NULL, tickSizeMinor = NULL, tickSizeEnd = NULL,
  title = "", titleOffset = NULL, titleMaxLength = NULL,
  characterWidth = 6, orient = NULL, format = NULL, remove = FALSE)
```

Arguments

vl	Vega-Lite object
axisWidth, layer, offset, grid, labels, labelAngle, labelAlign, labelBaseline	see axis docs & axis base config

`labelMaxLength`, `shortTimeLabels`, `subdivide`, `ticks`, `tickPadding`, `tickSize`
see axis docs & axis base config

`tickSizeMajor`, `tickSizeMinor`, `tickSizeEnd`, `title`, `titleOffset`, `titleMaxLength`
see axis docs & axis base config

`characterWidth`, `orient`, `format`, `remove`
see axis docs & axis base config

References

Vega-List Axis spec

Examples

```
vegalite() %>%
  add_data("https://vega.github.io/vega-editor/app/data/population.json") %>%
  add_filter("datum.year == 2000") %>%
  calculate("gender", 'datum.sex == 2 ? "Female" : "Male"') %>%
  encode_x("gender", "nominal") %>%
  encode_y("people", "quantitative", aggregate="sum") %>%
  encode_color("gender", "nominal") %>%
  scale_x_ordinal(band_size=6) %>%
  scale_color_nominal(range=c("#EA98D2", "#659CCA")) %>%
  facet_col("age", "ordinal", padding=4) %>%
  axis_x(remove=TRUE) %>%
  axis_y(title="population", grid=FALSE) %>%
  axis_facet_col(orient="bottom", axisWidth=1, offset=-8) %>%
  facet_cell(stroke_width=0) %>%
  mark_bar()
```

---

bin_x *Group continuous data values (x-axis)*

Description

The "bin" property is for grouping quantitative, continuous data values of a particular field into smaller number of “bins” (e.g., for a histogram).

Usage

```
bin_x(vl, min = NULL, max = NULL, base = NULL, step = NULL,
  steps = NULL, minstep = NULL, div = NULL, maxbins = NULL)
```

Arguments

`vl`	Vega-Lite object
`min`	the minimum bin value to consider.
`max`	the maximum bin value to consider.
`base`	the number base to use for automatic bin determination.

step	an exact step size to use between bins.
steps	an array of allowable step sizes to choose from.
minstep	minimum allowable step size (particularly useful for integer values).
div	Scale factors indicating allowable subdivisions. The default value is [5, 2], which indicates that for base 10 numbers (the default base), the method may consider dividing bin sizes by 5 and/or 2. For example, for an initial step size of 10, the method can check if bin sizes of 2 (= 10/5), 5 (= 10/2), or 1 (= 10/(5*2)) might also satisfy the given constraints.
maxbins	the maximum number of allowable bins.

References

Vega-Lite Binning

Examples

```
vegalite() %>%
  add_data("https://vega.github.io/vega-editor/app/data/movies.json") %>%
  encode_x("IMDB_Rating", "quantitative") %>%
  encode_y("Rotten_Tomatoes_Rating", "quantitative") %>%
  encode_size("*", "quantitative", aggregate="count") %>%
  bin_x(maxbins=10) %>%
  bin_y(maxbins=10) %>%
  mark_point()
```

---

bin_y *Group continuous data values (y-axis)*

Description

The "bin" property is for grouping quantitative, continuous data values of a particular field into smaller number of “bins” (e.g., for a histogram).

Usage

```
bin_y(vl, min = NULL, max = NULL, base = NULL, step = NULL,
  steps = NULL, minstep = NULL, div = NULL, maxbins = NULL)
```

Arguments

vl	Vega-Lite object
min	the minimum bin value to consider.
max	the maximum bin value to consider.
base	the number base to use for automatic bin determination.
step	an exact step size to use between bins.

steps	an array of allowable step sizes to choose from.
minstep	minimum allowable step size (particularly useful for integer values).
div	Scale factors indicating allowable subdivisions. The default value is [5, 2], which indicates that for base 10 numbers (the default base), the method may consider dividing bin sizes by 5 and/or 2. For example, for an initial step size of 10, the method can check if bin sizes of 2 (= 10/5), 5 (= 10/2), or 1 (= 10/(5*2)) might also satisfy the given constraints.
maxbins	the maximum number of allowable bins.

References

Vega-Lite Binning

Examples

```
vegalite() %>%
  add_data("https://vega.github.io/vega-editor/app/data/movies.json") %>%
  encode_x("IMDB_Rating", "quantitative") %>%
  encode_y("Rotten_Tomatoes_Rating", "quantitative") %>%
  encode_size("*", "quantitative", aggregate="count") %>%
  bin_x(maxbins=10) %>%
  bin_y(maxbins=10) %>%
  mark_point()
```

---

calculate *Derive new fields*

Description

Derive new fields

Usage

```
calculate(vl, field, expr)
```

Arguments

vl	Vega-Lite object created by vegalite
field	the field name in which to store the computed value.
expr	a string containing an expression for the formula. Use the variable "datum" to refer to the current data object.

Examples

```
vegalite() %>%
  add_data("https://vega.github.io/vega-editor/app/data/population.json") %>%
  add_filter("datum.year == 2000") %>%
  calculate("gender", 'datum.sex == 2 ? "Female" : "Male"') %>%
  encode_x("gender", "nominal") %>%
  encode_y("people", "quantitative", aggregate="sum") %>%
  encode_color("gender", "nominal") %>%
  scale_x_ordinal(band_size=6) %>%
  scale_color_nominal(range=c("#EA98D2", "#659CCA")) %>%
  facet_col("age", "ordinal", padding=4) %>%
  axis_x(remove=TRUE) %>%
  axis_y(title="population", grid=FALSE) %>%
  axis_facet_col(orient="bottom", axisWidth=1, offset=-8) %>%
  facet_cell(stroke_width=0) %>%
  mark_bar()
```

---

capture_widget *Capture a static (png) version of a widget (e.g. for use in a PDF knitr document)*

Description

Widgets are generally interactive beasts rendered in an HTML DOM with javascript. That makes them unusable in PDF documents. However, many widgets initial views would work well as static images. This function renders a widget to a file and make it usable in a number of contexts.

Usage

```
capture_widget(wdgt, output = c("path", "markdown", "html", "inline"), height,
  width, png_render_path = tempfile(fileext = ".png"))
```

Arguments

`wdgt`	htmlwidget to capture
`output`	how to return the results of the capture (see Details section)
`height, width`	it's important for many widget to be responsive in HTML documents. PDFs are static beasts and having a fixed image size works better for them. `height` & `width` will be passed into the rendering process, which means you should probably specify similar values in your widget creation process so the captured `<div>` size matches the size you specify here.
`png_render_path`	by default, this will be a temporary file location but a fully qualified filename (with extension) can be specified. It's up to the caller to free the storage when finished with the resource.

Details

What is returned depends on the value of `output`. By default (`"path"`), the full disk path will be returned. If `markdown` is specified, a markdown string will be returned with a `file:///...` URL. If `html` is specified, an `<img src='file:///...'/>` tag will be returned and if `inline` is specified, a base64 encoded `<img>` tag will be returned (just like you'd see in a self-contained HTML file from `knitr`).

Value

See Details

Examples

```
## Not run:
library(webshot)
library(vegalite)

dat <- jsonlite::fromJSON('[
      {"a": "A","b": 28}, {"a": "B","b": 55}, {"a": "C","b": 43},
      {"a": "D","b": 91}, {"a": "E","b": 81}, {"a": "F","b": 53},
      {"a": "G","b": 19}, {"a": "H","b": 87}, {"a": "I","b": 52}
    ]')

vegalite(viewport_width=350, viewport_height=250) %>%
  add_data(dat) %>%
  encode_x("a", "ordinal") %>%
  encode_y("b", "quantitative") %>%
  mark_bar() -> vl

capture_widget(vl, "inline", 250, 350)

## End(Not run)
```

---

cell_size *Add cell size to main Vega-Lite spec*

Description

Short version: set this to control the height and with of a single plot panel. It will also be the size of panels in a faceted/trellis plot, so make sure your viewport height/width (set in the main call to the widget) is as large as you want it to be (otheriwse this will do it's best to calculate it but will probably not be what you ultimately want).

Usage

```
cell_size(vl, width = 200, height = 200)
```

Arguments

vl	a Vega-Lite object
width	the width of the single plot or each plot in a trellis plot when the visualization has continuous x-scale. (If the plot has ordinal x-scale, the width is determined by the x-scale's bandSize and the cardinality of the x-scale. If the plot does not have a field on x, the width is derived from scale config's bandSize for all marks except text and from scale config's textBandWidth for text mark.) Default value: 200
height	the height of the single plot or each plot in a trellis plot when the visualization has continuous y-scale. (If the visualization has ordinal y-scale, the height is determined by the bandSize and the cardinality of the y-scale. If the plot does not have a field on y, the height is scale config's bandSize.) Default value: 200

Details

At its core, a Vega-Lite specification describes a single plot. When a facet channel is added, the visualization is faceted into a trellis plot, which contains multiple plots. Each plot in either a single plot or a trellis plot is called a cell. Cell configuration allows us to customize each individual single plot and each plot in a trellis plot.

References

Vega-Lite Cell spec

Examples

```
vegalite() %>%
  cell_size(300, 200) %>%
 add_data("https://vega.github.io/vega-editor/app/data/unemployment-across-industries.json") %>%
  encode_x("date", "temporal") %>%
  encode_y("count", "quantitative", aggregate="sum") %>%
  encode_color("series", "nominal") %>%
  scale_color_nominal(range="category20b") %>%
  timeunit_x("yearmonth") %>%
  scale_x_time(nice="month") %>%
  axis_x(axisWidth=0, format="%Y", labelAngle=0) %>%
  mark_area()
```

---

config_color *Color config*

Description

Color config

Usage

```
config_color(vl, color = NULL, fill = NULL, stroke = NULL)
```

Arguments

vl	a Vega-Lite object
color	color of the mark – either fill or stroke color based on the filled mark config.
fill	fill color. This config will be overridden by color channel's specified or mapped values if filled is true.
stroke	stroke color. This config will be overridden by color channel's specified or mapped values if filled is false.

---

config_font *Font config*

Description

Font config

Usage

```
config_font(vl, font = NULL, font_size = NULL, font_style = NULL,
  font_weight = NULL)
```

Arguments

vl	a Vega-Lite object
font	typeface to set the text in (e.g., Helvetica Neue).
font_size	font size, in pixels. The default value is 10.
font_style	font style (e.g., italic).
font_weight	font weight (e.g., bold).

---

config_opacity *Opacity config*

Description

Opacity config

Usage

```
config_opacity(vl, opacity = NULL, fill_opacity = NULL,
  stroke_opacity = NULL)
```

Arguments

vl	a Vega-Lite object
opacity	0.0-1.0
fill_opacity	0.0-1.0
stroke_opacity	0.0-1.0

config_stroke *Stroke config*

Description

Stroke config

Usage

```
config_stroke(vl, stroke = NULL, stroke_width = NULL, stroke_dash = NULL,
  stroke_dash_offset = NULL, stroke_opacity = NULL)
```

Arguments

vl	a Vega-Lite object
stroke	stroke color
stroke_width	stroke of the width in pixels
stroke_dash	an array of alternating stroke, space lengths for creating dashed or dotted lines.
stroke_dash_offset	the offset (in pixels) into which to begin drawing with the stroke dash array.
stroke_opacity	0.0-1.0

config_text *Text config*

Description

Text config

Usage

```
config_text(vl, angle = NULL, align = NULL, baseline = NULL, dx = NULL,
  dy = NULL, radius = NULL, theta = NULL, format = NULL,
  short_time_labels = NULL, opacity = NULL)
```

Arguments

vl	a Vega-Lite object
angle	rotation angle of the text, in degrees.
align	horizontal alignment of the text. One of left, right, center.
baseline	vertical alignment of the text. One of top, middle, bottom.
dx, dy	horizontal/vertical in pixels, between the text label and its anchor point. The offset is applied after rotation by the angle property.

radius	polar coordinate radial offset, in pixels, of the text label from the origin determined by the x and y properties.
theta	polar coordinate angle, in radians, of the text label from the origin determined by the x and y properties. Values for theta follow the same convention of arc mark startAngle and endAngle properties: angles are measured in radians, with 0 indicating "north".
format	ormatting pattern for text value. If not defined, this will be determined automatically
short_time_labels	whether month names and weekday names should be abbreviated.
opacity	0-1

References

Vega-Lite Mark spec

`embed_spec` *Scaffold HTML/JavaScript/CSS code from* `vegalite`

Description

Create minimal necessary HTML/JavaScript/CSS code to embed a Vega-Lite spec into a web page. This assumes you have the necessary boilerplate javascript & HTML page shell defined as you see in the Vega-Lite core example.

Usage

```
embed_spec(vl, element_id = generate_id(), to_cb = FALSE)
```

Arguments

vl	a Vega-Lite object
element_id	if you don't specify one, an id will be generated. This should be descriptive, but short, and valid javascript & CSS identifier syntax as is is appended to variable names.
to_cb	if TRUE, will copy the spec to the system clipboard. Default is FALSE.

Details

If you are generating more than one object to embed into a single web page, you will need to ensure each `element_id` is unique. Each Vega-Lite `div` is classed with `vldiv` so you can provide both a central style (say, `display:inline-block; margin-auto;`) and targeted ones that use the `div id`.

Examples

```
dat <- jsonlite::fromJSON('[
    {"a": "A","b": 28}, {"a": "B","b": 55}, {"a": "C","b": 43},
    {"a": "D","b": 91}, {"a": "E","b": 81}, {"a": "F","b": 53},
    {"a": "G","b": 19}, {"a": "H","b": 87}, {"a": "I","b": 52}
  ]')

vegalite() %>%
  add_data(dat) %>%
  encode_x("a", "ordinal") %>%
  encode_y("b", "quantitative") %>%
  mark_bar() -> chart

embed_spec(chart)
```

---

encode_color *Encode color "channel"*

Description

Encode color "channel"

Usage

```
encode_color(vl, field = NULL, type, value = NULL, aggregate = NULL,
  sort = NULL)
```

Arguments

vl	Vega-Lite object created by vegalite
field	single element character vector naming the column
type	the encoded field's type of measurement. This can be either a full type name (quantitative, temporal, ordinal, and nominal) or an initial character of the type name (Q, T, O, N). This property is case insensitive. If auto is used, the type will be guessed (so you may want to actually specify it if you want consistency).
value	scale value
aggregate	perform aggregaton on field. See Supported Aggregation Options for more info on valid operations. Leave NULL for no aggregation.
sort	either one of ascending, descending or (for ordinal scales) the result of a call to sort_def

Note

right now, type == "auto" just assume "quantitative". It will eventually get smarter, but you are better off specifying it.

References

Vega-Lite Encoding spec

Examples

```
vegalite() %>%
  add_data("https://vega.github.io/vega-editor/app/data/cars.json") %>%
  encode_x("Horsepower", "quantitative") %>%
  encode_y("Miles_per_Gallon", "quantitative") %>%
  encode_color("Origin", "nominal") %>%
  encode_shape("Origin", "nominal") %>%
  mark_point()
```

`encode_detail` *Encode detail "channel"*

Description

Grouping data is another important operation in visualizing data. For aggregated plots, all encoded fields without aggregate functions are used as grouping fields in the aggregation (similar to fields in GROUP BY in SQL). For line and area marks, mapping a data field to color or shape channel will group the lines and stacked areas by the field.

detail channel allows providing an additional grouping field (level) for grouping data in aggregation without mapping data to a specific visual channel.

Usage

```
encode_detail(vl, field = NULL, type, aggregate = NULL, sort = NULL)
```

Arguments

`vl`	Vega-Lite object created by `vegalite`
`field`	single element character vector naming the column
`type`	the encoded field's type of measurement. This can be either a full type name (`quantitative`, `temporal`, `ordinal`, and `nominal`) or an initial character of the type name (`Q`, `T`, `O`, `N`). This property is case insensitive. If `auto` is used, the type will be guessed (so you may want to actually specify it if you want consistency).
`aggregate`	perform aggregaton on `field`. See Supported Aggregation Options for more info on valid operations. Leave `NULL` for no aggregation.
`sort`	either one of `ascending`, `descending` or (for ordinal scales) the result of a call to `sort_def`

Note

right now, `type == "auto"` just assume `"quantitative"`. It will eventually get smarter, but you are better off specifying it.

References

Vega-Lite Encoding spec

Examples

```
vegalite() %>%
  cell_size(200, 200) %>%
  add_data("https://vega.github.io/vega-editor/app/data/stocks.csv") %>%
  encode_x("date", "temporal") %>%
  encode_y("price", "quantitative") %>%
  encode_detail("symbol", "nominal") %>%
  mark_line()
```

---

encode_order *Encode detail "order"*

Description

Grouping data is another important operation in visualizing data. For aggregated plots, all encoded fields without aggregate functions are used as grouping fields in the aggregation (similar to fields in GROUP BY in SQL). For line and area marks, mapping a data field to color or shape channel will group the lines and stacked areas by the field.

order channel sorts the layer order or stacking order (for stacked charts) of the marks while path channel sorts the order of data points in line marks.

Usage

```
encode_order(vl, field = NULL, type, aggregate = NULL, sort = NULL)
```

Arguments

vl	Vega-Lite object created by vegalite
field	single element character vector naming the column
type	the encoded field's type of measurement. This can be either a full type name (quantitative, temporal, ordinal, and nominal) or an initial character of the type name (Q, T, O, N). This property is case insensitive. If auto is used, the type will be guessed (so you may want to actually specify it if you want consistency).
aggregate	perform aggregaton on field. See Supported Aggregation Options for more info on valid operations. Leave NULL for no aggregation.
sort	either one of ascending, descending or (for ordinal scales) the result of a call to sort_def

Note

right now, type == "auto" just assume "quantitative". It will eventually get smarter, but you are better off specifying it.

References

Vega-Lite Encoding spec

Examples

```
vegalite() %>%
  cell_size(200, 200) %>%
  add_data("https://vega.github.io/vega-editor/app/data/cars.json") %>%
  encode_x("Horsepower", "quantitative") %>%
  encode_y("Miles_per_Gallon", "quantitative") %>%
  encode_color("Origin", "nominal") %>%
  encode_order("Origin", "ordinal", sort="descending") %>%
  mark_point()
```

---

encode_path *Encode detail "path"*

Description

Grouping data is another important operation in visualizing data. For aggregated plots, all encoded fields without aggregate functions are used as grouping fields in the aggregation (similar to fields in GROUP BY in SQL). For line and area marks, mapping a data field to color or shape channel will group the lines and stacked areas by the field.

By default, line marks order their points in their paths by the field of channel x or y. However, to show a pattern of data change over time between x & y we use path channel to sort points in a paritcular order (e.g. by time).

Usage

```
encode_path(vl, field = NULL, type, aggregate = NULL, sort = NULL)
```

Arguments

vl	Vega-Lite object created by vegalite
field	single element character vector naming the column
type	the encoded field's type of measurement. This can be either a full type name (quantitative, temporal, ordinal, and nominal) or an initial character of the type name (Q, T, O, N). This property is case insensitive. If auto is used, the type will be guessed (so you may want to actually specify it if you want consistency).
aggregate	perform aggregaton on field. See Supported Aggregation Options for more info on valid operations. Leave NULL for no aggregation.
sort	either one of ascending, descending or (for ordinal scales) the result of a call to sort_def

Note

right now, type == "auto" just assume "quantitative". It will eventually get smarter, but you are better off specifying it.

References

Vega-Lite Encoding spec

Examples

```
vegalite() %>%
  cell_size(300, 300) %>%
  add_data("https://vega.github.io/vega-editor/app/data/driving.json") %>%
  encode_x("miles", "quantitative") %>%
  encode_y("gas", "quantitative") %>%
  encode_path("year", "temporal") %>%
  scale_x_linear(zero=FALSE) %>%
  scale_y_linear(zero=FALSE) %>%
  mark_line()
```

---

encode_shape *Encode shape "channel"*

Description

Encode shape "channel"

Usage

```
encode_shape(vl, field = NULL, type, value = NULL, aggregate = NULL,
  sort = NULL)
```

Arguments

vl	Vega-Lite object created by vegalite
field	single element character vector naming the column
type	the encoded field's type of measurement. This can be either a full type name (quantitative, temporal, ordinal, and nominal) or an initial character of the type name (Q, T, O, N). This property is case insensitive. If auto is used, the type will be guessed (so you may want to actually specify it if you want consistency).
value	scale value
aggregate	perform aggregaton on field. See Supported Aggregation Options for more info on valid operations. Leave NULL for no aggregation.
sort	either one of ascending, descending or (for ordinal scales) the result of a call to sort_def

Note

right now, type == "auto" just assume "quantitative". It will eventually get smarter, but you are better off specifying it.

References

Vega-Lite Encoding spec

Examples

```
vegalite() %>%
  add_data("https://vega.github.io/vega-editor/app/data/cars.json") %>%
  encode_x("Horsepower", "quantitative") %>%
  encode_y("Miles_per_Gallon", "quantitative") %>%
  encode_color("Origin", "nominal") %>%
  encode_shape("Origin", "nominal") %>%
  mark_point()
```

---

encode_size *Encode size "channel"*

Description

Encode size "channel"

Usage

```
encode_size(vl, field = NULL, type, value = NULL, aggregate = NULL,
  sort = NULL)
```

Arguments

vl	Vega-Lite object created by vegalite
field	single element character vector naming the column. Can be * is using aggregate.
type	the encoded field's type of measurement. This can be either a full type name (quantitative, temporal, ordinal, and nominal) or an initial character of the type name (Q, T, O, N). This property is case insensitive. If auto is used, the type will be guessed (so you may want to actually specify it if you want consistency).
value	scale value
aggregate	perform aggregaton on field. See Supported Aggregation Options for more info on valid operations. Leave NULL for no aggregation.
sort	either one of ascending, descending or (for ordinal scales) the result of a call to sort_def

Note

right now, type == "auto" just assume "quantitative". It will eventually get smarter, but you are better off specifying it.

References

Vega-Lite Encoding spec

Examples

```
vegalite() %>%
  add_data("https://vega.github.io/vega-editor/app/data/cars.json") %>%
  encode_x("Horsepower", "quantitative") %>%
  encode_y("Miles_per_Gallon", "quantitative") %>%
  encode_size("Acceleration", "quantitative") %>%
  mark_point()
```

---

encode_text *Encode text "channel"*

Description

Encode text "channel"

Usage

```
encode_text(vl, field, type, value = NULL, aggregate = NULL, sort = NULL)
```

Arguments

vl	Vega-Lite object created by vegalite
field	single element character vector naming the column. Can be * is using aggregate.
type	the encoded field's type of measurement. This can be either a full type name (quantitative, temporal, ordinal, and nominal) or an initial character of the type name (Q, T, O, N). This property is case insensitive. If auto is used, the type will be guessed (so you may want to actually specify it if you want consistency).
value	scale value
aggregate	perform aggregaton on field. See Supported Aggregation Options for more info on valid operations. Leave NULL for no aggregation.
sort	either one of ascending, descending or (for ordinal scales) the result of a call to sort_def

Note

right now, type == "auto" just assume "quantitative". It will eventually get smarter, but you are better off specifying it.

References

Vega-Lite Encoding spec

Examples

```
vegalite() %>%
  cell_size(300, 200) %>%
  add_data("https://vega.github.io/vega-editor/app/data/cars.json") %>%
  encode_x("Horsepower", "quantitative") %>%
  encode_y("Miles_per_Gallon", "quantitative") %>%
  encode_color("Origin", "nominal") %>%
  calculate("OriginInitial", "datum.Origin[0]") %>%
  encode_text("OriginInitial", "nominal") %>%
  mark_text()
```

---

encode_x *Encode x "channel"*

Description

Vega-Lite has many "encoding channels". Each channel definition object must describe the data field encoded by the channel and its data type, or a constant value directly mapped to the mark properties. In addition, it can describe the mapped field's transformation and properties for its scale and guide.

Usage

```
encode_x(vl, field, type = "auto", aggregate = NULL, sort = NULL)
```

Arguments

vl	Vega-Lite object created by vegalite
field	single element character vector naming the column. Can be * is using aggregate.
type	the encoded field's type of measurement. This can be either a full type name (quantitative, temporal, ordinal, and nominal) or an initial character of the type name (Q, T, O, N). This property is case insensitive. If auto is used, the type will be guessed (so you may want to actually specify it if you want consistency).
aggregate	perform aggregaton on field. See Supported Aggregation Options for more info on valid operations. Leave NULL for no aggregation.
sort	either one of ascending, descending or (for ordinal scales) the result of a call to sort_def

Note

right now, type == "auto" just assume "quantitative". It will eventually get smarter, but you are better off specifying it.

References

Vega-Lite Encoding spec

Examples

```
dat <- jsonlite::fromJSON('[
    {"a": "A","b": 28}, {"a": "B","b": 55}, {"a": "C","b": 43},
    {"a": "D","b": 91}, {"a": "E","b": 81}, {"a": "F","b": 53},
    {"a": "G","b": 19}, {"a": "H","b": 87}, {"a": "I","b": 52}
  ]')

vegalite() %>%
  add_data(dat) %>%
  encode_x("a", "ordinal") %>%
  encode_y("b", "quantitative") %>%
  mark_bar()
```

---

encode_y *Encode y "channel"*

Description

Vega-Lite has many "encoding channels". Each channel definition object must describe the data field encoded by the channel and its data type, or a constant value directly mapped to the mark properties. In addition, it can describe the mapped field's transformation and properties for its scale and guide.

Usage

```
encode_y(vl, field, type = "auto", aggregate = NULL, sort = NULL)
```

Arguments

vl	Vega-Lite object created by vegalite
field	single element character vector naming the column
type	the encoded field's type of measurement. This can be either a full type name (quantitative, temporal, ordinal, and nominal) or an initial character of the type name (Q, T, O, N). This property is case insensitive. If auto is used, the type will be guessed (so you may want to actually specify it if you want consistency).
aggregate	perform aggregaton on field. See Supported Aggregation Options for more info on valid operations. Leave NULL for no aggregation.
sort	either one of ascending, descending or (for ordinal scales) the result of a call to sort_def

Note

right now, type == "auto" just assume "quantitative". It will eventually get smarter, but you are better off specifying it.

Examples

```
dat <- jsonlite::fromJSON('[
    {"a": "A","b": 28}, {"a": "B","b": 55}, {"a": "C","b": 43},
    {"a": "D","b": 91}, {"a": "E","b": 81}, {"a": "F","b": 53},
    {"a": "G","b": 19}, {"a": "H","b": 87}, {"a": "I","b": 52}
  ]')

vegalite() %>%
  add_data(dat) %>%
  encode_x("a", "ordinal") %>%
  encode_y("b", "quantitative") %>%
  mark_bar()
```

---

facet_cell *Facet cell aesthetics*

Description

At its core, a Vega-Lite specification describes a single plot. When a facet channel is added, the visualization is faceted into a trellis plot, which contains multiple plots. Each plot in either a single plot or a trellis plot is called a cell. Cell configuration allows us to customize each individual single plot and each plot in a trellis plot.

Usage

```
facet_cell(vl, width = 200, height = 200, fill = NULL,
  fill_opacity = NULL, stroke = NULL, stroke_opacity = NULL,
  stroke_width = NULL, stroke_dash = NULL, stroke_dash_offset = NULL)
```

Arguments

vl	Vega-Lite object
width, height	width and height property of the cell configuration determine the width of a visualization with a continuous x-scale and the height of a visualization with a continuous y-scale respectively. Visit the URL in the References section for more information.
fill	fill color
fill_opacity	0.0-1.0
stroke	stroke color
stroke_opacity	0.0-1.0
stroke_width	stroke of the width in pixels

stroke_dash an array of alternating stroke, space lengths for creating dashed or dotted lines.

stroke_dash_offset
the offset (in pixels) into which to begin drawing with the stroke dash array.

References

Vega-Lite Facet spec

`facet_col` *Create a horizontal ribbon of panels*

Description

Create a horizontal ribbon of panels

Usage

```
facet_col(vl, field, type, round = TRUE, padding = 16)
```

Arguments

vl	Vega-Lite object
field	single element character vector naming the column.
type	the encoded field's type of measurement.
round	round values
padding	facet padding

References

Vega-Lite Faceting

Examples

```
vegalite() %>%
  add_data("https://vega.github.io/vega-editor/app/data/population.json") %>%
  add_filter("datum.year == 2000") %>%
  calculate("gender", 'datum.sex == 2 ? "Female" : "Male"') %>%
  encode_x("gender", "nominal") %>%
  encode_y("people", "quantitative", aggregate="sum") %>%
  encode_color("gender", "nominal") %>%
  scale_x_ordinal(band_size=6) %>%
  scale_color_nominal(range=c("#EA98D2", "#659CCA")) %>%
  facet_col("age", "ordinal", padding=4) %>%
  axis_x(remove=TRUE) %>%
  axis_y(title="population", grid=FALSE) %>%
  axis_facet_col(orient="bottom", axisWidth=1, offset=-8) %>%
  facet_cell(stroke_width=0) %>%
  mark_bar()
```

facet_row *Create a vertical ribbon of panels*

Description

Create a vertical ribbon of panels

Usage

```
facet_row(vl, field, type, round = TRUE, padding = 16)
```

Arguments

vl	Vega-Lite object
field	single element character vector naming the column.
type	the encoded field's type of measurement.
round	round values
padding	facet padding

References

Vega-Lite Faceting

Examples

```
# see facet_col
```

filter_null *Filter 'null' values*

Description

Whether to filter null values from the data.

Usage

```
filter_null(vl, setting = NULL)
```

Arguments

vl	Vega-Lite object created by vegalite
setting	if NULL only quantitative and temporal fields are filtered. If TRUE, all data items with 'null' values are filtered. If FALSE, all data items are included.

from_spec *Take a JSON Vega-Lite Spec and render as an htmlwidget*

Description

Vega-Lite is - at the core - a JSON "Grammar of Graphics" specification for how to build a data- & stats-based visualization. While Vega & D3 are the main targets, the use of Vega-Lite does not have to be restricted to just D3. For now, this function takes in a JSON spec (full text or URL) and renders it as an htmlwidget. Data should either be embedded or use a an absolute URL reference.

Usage

```
from_spec(spec, width = NULL, height = NULL, renderer = c("svg",
  "canvas"), export = FALSE, source = FALSE, editor = FALSE)
```

Arguments

spec	URL to a Vega-Lite JSON file or the JSON text of a spec
width, height	widget width/height
renderer	the renderer to use for the view. One of canvas or svg (the default)
export	if TRUE the *"Export as..."* link will be displayed with the chart.(Default: FALSE.)
source	if TRUE the *"View Source"* link will be displayed with the chart. (Default: FALSE.)
editor	if TRUE the *"Open in editor"* link will be displayed with the cahrt. (Default: FALSE.)

Examples

```
from_spec("http://rud.is/dl/embedded.json")
```

grid_facet *Facet grid aesthetics*

Description

Facet grid aesthetics

Usage

```
grid_facet(vl, grid_color = NULL, grid_opacity = NULL, grid_offset = NULL)
```

Arguments

vl	Vega-Lite object
grid_color	color of the grid between facets.
grid_opacity	0.0-1.0
grid_offset	offset for grid between facets.

References

Vega-Lite Facet spec

`JS` *Mark character strings as literal JavaScript code*

Description

Mark character strings as literal JavaScript code

`legend_color` *Legend settings (color)*

Description

Legend settings (color)

Usage

```
legend_color(vl, orient = NULL, title = NULL, format = NULL,
  short_time_labels = NULL, value = NULL, remove = FALSE)
```

Arguments

`vl`	a Vega-Lite object
`orient`	the orientation of the legend. One of "left" or "right". This determines how the legend is positioned within the scene.
`title`	the title for the legend.
`format`	the formatting pattern for axis labels. This is D3's number format pattern for quantitative axis and D3's time format pattern for time axis.
`short_time_labels`	whether month and day names should be abbreviated.
`value`	explicitly set the visible legend values.
`remove`	if `TRUE`, there will be no legend for this aesthetic.

legend_shape *Legend settings (shape)*

Description

Legend settings (shape)

Usage

```
legend_shape(vl, orient = NULL, title = NULL, format = NULL,
  short_time_labels = NULL, value = NULL, remove = FALSE)
```

Arguments

vl	a Vega-Lite object
orient	the orientation of the legend. One of "left" or "right". This determines how the legend is positioned within the scene.
title	the title for the legend.
format	the formatting pattern for axis labels. This is D3's number format pattern for quantitative axis and D3's time format pattern for time axis.
short_time_labels	whether month and day names should be abbreviated.
value	explicitly set the visible legend values.
remove	if TRUE, there will be no legend for this aesthetic.

legend_size *Legend settings (size)*

Description

Legend settings (size)

Usage

```
legend_size(vl, orient = NULL, title = NULL, format = NULL,
  short_time_labels = NULL, value = NULL, remove = FALSE)
```

Arguments

vl	a Vega-Lite object
orient	the orientation of the legend. One of "left" or "right". This determines how the legend is positioned within the scene.
title	the title for the legend.
format	the formatting pattern for axis labels. This is D3's number format pattern for quantitative axis and D3's time format pattern for time axis.
short_time_labels	whether month and day names should be abbreviated.
value	explicitly set the visible legend values.
remove	if TRUE, there will be no legend for this aesthetic.

---

mark_area *Area mark*

Description

An area represent multiple data element as a single area shape.

Usage

```
mark_area(vl, orient = NULL, stack = NULL, interpolate = NULL,
  tension = NULL, opacity = NULL, filled = NULL, color = NULL,
  fill = NULL, stroke = NULL)
```

Arguments

vl	Vega-Lite object
orient	the orientation of a non-stacked bar, area, and line charts. The value is either "horizontal", or "vertical" (default). For bar and tick, this determines whether the size of the bar and tick should be applied to x or y dimension. For area, this property determines the orient property of the Vega output. For line, this property determines the path order of the points in the line if path channel is not specified. For stacked charts, this is always determined by the orientation of the stack; therefore explicitly specified value will be ignored.
stack	stacking modes for bar and area marks. zero - stacking with baseline offset at zero value of the scale (for creating typical stacked bar and area chart). normalize - stacking with normalized domain (for creating normalized stacked bar and area chart). center - stacking with center baseline (for streamgraph). none - No-stacking. This will produces layered bar and area chart.
interpolate	The line interpolation method to use. One of linear step-before, step-after, basis, basis-open, basis-closed, bundle, cardinal, cardinal-open, cardinal-closed, monotone. For more information about each interpolation method, please see D3's line interpolation.

tension	Depending on the interpolation type, sets the tension parameter. (See D3's line interpolation.)
opacity	0.0-1.0
filled	whether the shape's color should be used as fill color instead of stroke color.
color	color of the mark – either fill or stroke color based on the filled mark config.
fill	fill color. This config will be overridden by color channel's specified or mapped values if filled is true.
stroke	stroke color. This config will be overridden by color channel's specified or mapped values if filled is false.

References

Vega-Lite Mark spec

Examples

```
vegalite() %>%
  cell_size(300, 200) %>%
 add_data("https://vega.github.io/vega-editor/app/data/unemployment-across-industries.json") %>%
  encode_x("date", "temporal") %>%
  encode_y("count", "quantitative", aggregate="sum") %>%
  encode_color("series", "nominal") %>%
  scale_color_nominal(range="category20b") %>%
  timeunit_x("yearmonth") %>%
  scale_x_time(nice="month") %>%
  axis_x(axisWidth=0, format="%Y", labelAngle=0) %>%
  mark_area()
```

---

mark_bar *Bar mark*

Description

A bar mark represents each data point as a rectangle, where the length is mapped to a quantitative scale.

Usage

```
mark_bar(vl, orient = NULL, stack = NULL, size = NULL, opacity = NULL,
  filled = NULL, color = NULL, fill = NULL, stroke = NULL)
```

Arguments

vl	Vega-Lite object
orient	the orientation of a non-stacked bar, area, and line charts. The value is either "horizontal", or "vertical" (default). For bar and tick, this determines whether the size of the bar and tick should be applied to x or y dimension. For area, this property determines the orient property of the Vega output. For line, this property determines the path order of the points in the line if path channel is not specified. For stacked charts, this is always determined by the orientation of the stack; therefore explicitly specified value will be ignored.
stack	stacking modes for bar and area marks. `zero` - stacking with baseline offset at zero value of the scale (for creating typical stacked bar and area chart). `normalize` - stacking with normalized domain (for creating normalized stacked bar and area chart). `center` - stacking with center baseline (for streamgraph). `none` - No-stacking. This will produces layered bar and area chart.
size	The pixel area each the point. For example: in the case of circles, the radius is determined in part by the square root of the size value.
opacity	`0.0-1.0`
filled	whether the shape's color should be used as fill color instead of stroke color.
color	color of the mark – either fill or stroke color based on the filled mark config.
fill	fill color. This config will be overridden by color channel's specified or mapped values if filled is true.
stroke	stroke color. This config will be overridden by color channel's specified or mapped values if filled is false.

References

Vega-Lite Mark spec

Examples

```
dat <- jsonlite::fromJSON('[
    {"a": "A","b": 28}, {"a": "B","b": 55}, {"a": "C","b": 43},
    {"a": "D","b": 91}, {"a": "E","b": 81}, {"a": "F","b": 53},
    {"a": "G","b": 19}, {"a": "H","b": 87}, {"a": "I","b": 52}
  ]')

vegalite() %>%
  add_data(dat) %>%
  encode_x("a", "ordinal") %>%
  encode_y("b", "quantitative") %>%
  mark_bar()
```

mark_circle *Circle mark*

Description

Circle and square marks are similar to point mark, except that (1) the shape value is always set to circle or square (2) they are filled by default.

Usage

```
mark_circle(vl, size = NULL, opacity = NULL, filled = NULL,
  color = NULL, fill = NULL, stroke = NULL)
```

Arguments

vl	a Vega-Lite object
size	The pixel area each the point. For example: in the case of circles, the radius is determined in part by the square root of the size value.
opacity	0.0-1.0
filled	whether the shape's color should be used as fill color instead of stroke color.
color	color of the mark – either fill or stroke color based on the filled mark config.
fill	fill color. This config will be overridden by color channel's specified or mapped values if filled is true.
stroke	stroke color. This config will be overridden by color channel's specified or mapped values if filled is false.

References

Vega-Lite Mark spec

Examples

```
vegalite() %>%
  add_data("https://vega.github.io/vega-editor/app/data/cars.json") %>%
  encode_x("Horsepower", "quantitative") %>%
  encode_y("Miles_per_Gallon", "quantitative") %>%
  mark_circle()
```

mark_line *Line mark*

Description

A line mark represents the data points stored in a field with a line connecting all of these points. Unlike other marks except area that represents one data element per mark, one line mark represent multiple data element as a single line.

Usage

```
mark_line(vl, orient = NULL, interpolate = NULL, tension = NULL,
  opacity = NULL, color = NULL, fill = NULL, stroke = NULL)
```

Arguments

vl	Vega-Lite object
orient	the orientation of a non-stacked bar, area, and line charts. The value is either "horizontal", or "vertical" (default). For bar and tick, this determines whether the size of the bar and tick should be applied to x or y dimension. For area, this property determines the orient property of the Vega output. For line, this property determines the path order of the points in the line if path channel is not specified. For stacked charts, this is always determined by the orientation of the stack; therefore explicitly specified value will be ignored.
interpolate	The line interpolation method to use. One of `linear` `step-before`, `step-after`, `basis`, `basis-open`, `basis-closed`, `bundle`, `cardinal`, `cardinal-open`, `cardinal-closed`, `monotone`. For more information about each interpolation method, please see D3's line interpolation.
tension	Depending on the interpolation type, sets the tension parameter. (See D3's line interpolation.)
opacity	`0.0-1.0`
color	color of the mark – either fill or stroke color based on the filled mark config.
fill	fill color. This config will be overridden by color channel's specified or mapped values if filled is true.
stroke	stroke color. This config will be overridden by color channel's specified or mapped values if filled is false.

References

Vega-Lite Mark spec

Examples

```
vegalite() %>%
  cell_size(300, 300) %>%
  add_data("https://vega.github.io/vega-editor/app/data/driving.json") %>%
  encode_x("miles", "quantitative") %>%
  encode_y("gas", "quantitative") %>%
  encode_path("year", "temporal") %>%
  scale_x_linear(zero=FALSE) %>%
  scale_y_linear(zero=FALSE) %>%
  mark_line()
```

---

mark_point *Point mark*

Description

A point mark represents each data point with a symbol.

Usage

```
mark_point(vl, shape = "circle", size = NULL, opacity = NULL,
  filled = NULL, color = NULL, fill = NULL, stroke = NULL)
```

Arguments

vl	Vega-Lite object
shape	The symbol shape to use. One of `circle`, `square`, `cross`, `diamond`, `triangle-up`, or `triangle-down`. Default value: `circle`.
size	The pixel area each the point. For example: in the case of circles, the radius is determined in part by the square root of the size value.
opacity	`0.0-1.0`
filled	whether the shape's color should be used as fill color instead of stroke color.
color	color of the mark – either fill or stroke color based on the filled mark config.
fill	fill color. This config will be overridden by color channel's specified or mapped values if filled is true.
stroke	stroke color. This config will be overridden by color channel's specified or mapped values if filled is false.

References

Vega-Lite Mark spec

Examples

```
vegalite() %>%
  add_data("https://vega.github.io/vega-editor/app/data/cars.json") %>%
  encode_x("Horsepower", "quantitative") %>%
  encode_y("Miles_per_Gallon", "quantitative") %>%
  mark_point()
```

---

mark_square *Square mark*

Description

Circle and square marks are similar to point mark, except that (1) the shape value is always set to circle or square (2) they are filled by default.

Usage

```
mark_square(vl, size = NULL, opacity = NULL, filled = NULL,
  color = NULL, fill = NULL, stroke = NULL)
```

Arguments

vl	a Vega-Lite object
size	The pixel area each the point. For example: in the case of circles, the radius is determined in part by the square root of the size value.
opacity	0.0-1.0
filled	whether the shape's color should be used as fill color instead of stroke color.
color	color of the mark – either fill or stroke color based on the filled mark config.
fill	fill color. This config will be overridden by color channel's specified or mapped values if filled is true.
stroke	stroke color. This config will be overridden by color channel's specified or mapped values if filled is false.

References

Vega-Lite Mark spec

Examples

```
vegalite() %>%
  add_data("https://vega.github.io/vega-editor/app/data/cars.json") %>%
  encode_x("Horsepower", "quantitative") %>%
  encode_y("Miles_per_Gallon", "quantitative") %>%
  mark_square()
```

mark_text *Text mark*

Description

A text mark represents each data point with a text instead of a point.

Usage

```
mark_text(vl, opacity = NULL, color = NULL, fill = NULL, stroke = NULL)
```

Arguments

vl	a Vega-Lite object
opacity	0.0-1.0
color	color of the mark – either fill or stroke color based on the filled mark config.
fill	fill color. This config will be overridden by color channel's specified or mapped values if filled is true.
stroke	stroke color. This config will be overridden by color channel's specified or mapped values if filled is false.

References

Vega-Lite Mark spec

Examples

```
vegalite() %>%
  cell_size(300, 200) %>%
  add_data("https://vega.github.io/vega-editor/app/data/cars.json") %>%
  encode_x("Horsepower", "quantitative") %>%
  encode_y("Miles_per_Gallon", "quantitative") %>%
  encode_color("Origin", "nominal") %>%
  calculate("OriginInitial", "datum.Origin[0]") %>%
  encode_text("OriginInitial", "nominal") %>%
  mark_text()
```

mark_tick *Tick mark*

Description

A tick mark represents each data point as a short line. This is a useful mark for displaying the distribution of values in a field.

Usage

```
mark_tick(vl, orient = NULL, size = NULL, thickness = 1, opacity = NULL,
  color = NULL, fill = NULL, stroke = NULL)
```

Arguments

vl	Vega-Lite object
orient	the orientation of a non-stacked bar, area, and line charts. The value is either "horizontal", or "vertical" (default). For bar and tick, this determines whether the size of the bar and tick should be applied to x or y dimension. For area, this property determines the orient property of the Vega output. For line, this property determines the path order of the points in the line if path channel is not specified. For stacked charts, this is always determined by the orientation of the stack; therefore explicitly specified value will be ignored.
size	The pixel area each the point. For example: in the case of circles, the radius is determined in part by the square root of the size value.
thickness	Thickness of the tick mark. Default value: 1
opacity	0.0-1.0
color	color of the mark – either fill or stroke color based on the filled mark config.
fill	fill color. This config will be overridden by color channel's specified or mapped values if filled is true.
stroke	stroke color. This config will be overridden by color channel's specified or mapped values if filled is false.

References

Vega-Lite Mark spec

Examples

```
vegalite() %>%
  add_data("https://vega.github.io/vega-editor/app/data/cars.json") %>%
  encode_x("Horsepower", "quantitative") %>%
  encode_y("Cylinders", "ordinal") %>%
  mark_tick()
```

renderVegalite *Widget render function for use in Shiny*

Description

Widget render function for use in Shiny

Usage

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

Arguments

expr	expr to render
env	evaluation environemnt
quoted	quote expression?

saveWidget *Save a widget to an HTML file*

Description

Save a widget to an HTML file

scale_color_nominal *Nominal Color Scale*

Description

Nominal Color Scale

Usage

```
scale_color_nominal(vl, domain = NULL, range = NULL)
```

Arguments

vl	Vega-Lite object
domain	Custom domain values. For quantitative data, this can take the form of a two-element array with minimum and maximum values.
range	The range of the scale represents the set of output visual values. Vega-Lite automatically determines appropriate range based on the scale's channel and type, but range property can be provided to customize range values.

References

Vega-Lite Scales spec

`scale_color_sequential` *Sequential Color Scale*

Description

Sequential Color Scale

Usage

```
scale_color_sequential(vl, domain = NULL, range = NULL)
```

Arguments

`vl`	Vega-Lite object
`domain`	Custom domain values. For quantitative data, this can take the form of a two-element array with minimum and maximum values.
`range`	The range of the scale represents the set of output visual values. Vega-Lite automatically determines appropriate range based on the scale's channel and type, but range property can be provided to customize range values.

References

Vega-Lite Scales spec

`scale_shape` *Shape Scale*

Description

Shape Scale

Usage

```
scale_shape(vl, domain = NULL, range = NULL)
```

Arguments

`vl`	Vega-Lite object
`domain`	Custom domain values. For quantitative data, this can take the form of a two-element array with minimum and maximum values.
`range`	The range of the scale represents the set of output visual values. Vega-Lite automatically determines appropriate range based on the scale's channel and type, but range property can be provided to customize range values.

References

Vega-Lite Scales spec

`scale_x_linear` *Quantitative Scale*

Description

Quantitative Scale

Usage

```
scale_x_linear(vl, domain = NULL, range = NULL, clamp = NULL,
  nice = NULL, zero = NULL)
```

Arguments

`vl`	Vega-Lite object
`domain`	Custom domain values. For quantitative data, this can take the form of a two-element array with minimum and maximum values.
`range`	The range of the scale represents the set of output visual values. Vega-Lite automatically determines appropriate range based on the scale's channel and type, but range property can be provided to customize range values.
`clamp`	if true, values that exceed the data domain are clamped to either the minimum or maximum range value. Default value: derived from scale config (true by default) Supported Types: only linear, pow, sqrt, and log
`nice`	If true, modifies the scale domain to use a more human-friendly number range (e.g., 7 instead of 6.96). Default value: true only for quantitative x and y scales and false otherwise.
`zero`	If true, ensures that a zero baseline value is included in the scale domain. Default value: true if the quantitative field is not binned.

References

Vega-Lite Scales spec

scale_x_log *Log Scale*

Description

Log Scale

Usage

```
scale_x_log(vl, domain = NULL, range = NULL, clamp = NULL, nice = NULL,
  zero = NULL)
```

Arguments

vl	Vega-Lite object
domain	Custom domain values. For quantitative data, this can take the form of a two-element array with minimum and maximum values.
range	The range of the scale represents the set of output visual values. Vega-Lite automatically determines appropriate range based on the scale's channel and type, but range property can be provided to customize range values.
clamp	if true, values that exceed the data domain are clamped to either the minimum or maximum range value. Default value: derived from scale config (true by default) Supported Types: only linear, pow, sqrt, and log
nice	If true, modifies the scale domain to use a more human-friendly number range (e.g., 7 instead of 6.96). Default value: true only for quantitative x and y scales and false otherwise.
zero	If true, ensures that a zero baseline value is included in the scale domain. Default value: true if the quantitative field is not binned.

References

Vega-Lite Scales spec

scale_x_ordinal *Ordinal Scale*

Description

Ordinal Scale

Usage

```
scale_x_ordinal(vl, band_size = NULL, padding = NULL)
```

Arguments

vl	Vega-Lite object
band_size	band size
padding	padding

References

Vega-Lite Scales spec

Examples

```
vegalite() %>%
  add_data("https://vega.github.io/vega-editor/app/data/population.json") %>%
  add_filter("datum.year == 2000") %>%
  calculate("gender", 'datum.sex == 2 ? "Female" : "Male"') %>%
  encode_x("gender", "nominal") %>%
  encode_y("people", "quantitative", aggregate="sum") %>%
  encode_color("gender", "nominal") %>%
  scale_x_ordinal(band_size=6) %>%
  scale_color_nominal(range=c("#EA98D2", "#659CCA")) %>%
  facet_col("age", "ordinal", padding=4) %>%
  axis_x(remove=TRUE) %>%
  axis_y(title="population", grid=FALSE) %>%
  axis_facet_col(orient="bottom", axisWidth=1, offset=-8) %>%
  facet_cell(stroke_width=0) %>%
  mark_bar()
```

---

scale_x_pow *Quantitative Scale*

Description

Quantitative Scale

Usage

```
scale_x_pow(vl, domain = NULL, range = NULL, clamp = NULL, exp = NULL,
  nice = NULL, zero = NULL)
```

Arguments

vl	Vega-Lite object
domain	Custom domain values. For quantitative data, this can take the form of a two-element array with minimum and maximum values.
range	The range of the scale represents the set of output visual values. Vega-Lite automatically determines appropriate range based on the scale's channel and type, but range property can be provided to customize range values.

clamp	if true, values that exceed the data domain are clamped to either the minimum or maximum range value. Default value: derived from scale config (true by default) Supported Types: only linear, pow, sqrt, and log
exp	exponent
nice	If true, modifies the scale domain to use a more human-friendly number range (e.g., 7 instead of 6.96). Default value: true only for quantitative x and y scales and false otherwise.
zero	If true, ensures that a zero baseline value is included in the scale domain. Default value: true if the quantitative field is not binned.

References

Vega-Lite Scales spec

`scale_x_quantile` *Quantile Scale*

Description

Quantile Scale

Usage

```
scale_x_quantile(vl, domain = NULL, range = NULL, clamp = NULL,
  nice = NULL, zero = NULL)
```

Arguments

vl	Vega-Lite object
domain	Custom domain values. For quantitative data, this can take the form of a two-element array with minimum and maximum values.
range	The range of the scale represents the set of output visual values. Vega-Lite automatically determines appropriate range based on the scale's channel and type, but range property can be provided to customize range values.
clamp	if true, values that exceed the data domain are clamped to either the minimum or maximum range value. Default value: derived from scale config (true by default) Supported Types: only linear, pow, sqrt, and log
nice	If true, modifies the scale domain to use a more human-friendly number range (e.g., 7 instead of 6.96). Default value: true only for quantitative x and y scales and false otherwise.
zero	If true, ensures that a zero baseline value is included in the scale domain. Default value: true if the quantitative field is not binned.

References

Vega-Lite Scales spec

scale_x_quantize *Quantize Scale*

Description

Quantize Scale

Usage

```
scale_x_quantize(vl, domain = NULL, range = NULL, clamp = NULL,
  nice = NULL, zero = NULL)
```

Arguments

vl	Vega-Lite object
domain	Custom domain values. For quantitative data, this can take the form of a two-element array with minimum and maximum values.
range	The range of the scale represents the set of output visual values. Vega-Lite automatically determines appropriate range based on the scale's channel and type, but range property can be provided to customize range values.
clamp	if true, values that exceed the data domain are clamped to either the minimum or maximum range value. Default value: derived from scale config (true by default) Supported Types: only linear, pow, sqrt, and log
nice	If true, modifies the scale domain to use a more human-friendly number range (e.g., 7 instead of 6.96). Default value: true only for quantitative x and y scales and false otherwise.
zero	If true, ensures that a zero baseline value is included in the scale domain. Default value: true if the quantitative field is not binned.

References

Vega-Lite Scales spec

scale_x_sqrt *Sqrt Scale*

Description

Sqrt Scale

Usage

```
scale_x_sqrt(vl, domain = NULL, range = NULL, clamp = NULL, nice = NULL,
  zero = NULL)
```

Arguments

vl	Vega-Lite object
domain	Custom domain values. For quantitative data, this can take the form of a two-element array with minimum and maximum values.
range	The range of the scale represents the set of output visual values. Vega-Lite automatically determines appropriate range based on the scale's channel and type, but range property can be provided to customize range values.
clamp	if true, values that exceed the data domain are clamped to either the minimum or maximum range value. Default value: derived from scale config (true by default) Supported Types: only linear, pow, sqrt, and log
nice	If true, modifies the scale domain to use a more human-friendly number range (e.g., 7 instead of 6.96). Default value: true only for quantitative x and y scales and false otherwise.
zero	If true, ensures that a zero baseline value is included in the scale domain. Default value: true if the quantitative field is not binned.

References

Vega-Lite Scales spec

---

scale_x_threshold *Threshold Scale*

Description

Threshold Scale

Usage

```
scale_x_threshold(vl, domain = NULL, range = NULL, clamp = NULL,
  nice = NULL, zero = NULL)
```

Arguments

vl	Vega-Lite object
domain	Custom domain values. For quantitative data, this can take the form of a two-element array with minimum and maximum values.
range	The range of the scale represents the set of output visual values. Vega-Lite automatically determines appropriate range based on the scale's channel and type, but range property can be provided to customize range values.
clamp	if true, values that exceed the data domain are clamped to either the minimum or maximum range value. Default value: derived from scale config (true by default) Supported Types: only linear, pow, sqrt, and log

nice	If true, modifies the scale domain to use a more human-friendly number range (e.g., 7 instead of 6.96). Default value: true only for quantitative x and y scales and false otherwise.
zero	If true, ensures that a zero baseline value is included in the scale domain. Default value: true if the quantitative field is not binned.

References

Vega-Lite Scales spec

`scale_x_time` *Temporal Scale*

Description

Temporal Scale

Usage

```
scale_x_time(vl, domain = NULL, range = NULL, clamp = NULL, nice = NULL,
  zero = NULL)
```

Arguments

vl	Vega-Lite object
domain	Custom domain values. For quantitative data, this can take the form of a two-element array with minimum and maximum values.
range	The range of the scale represents the set of output visual values. Vega-Lite automatically determines appropriate range based on the scale's channel and type, but range property can be provided to customize range values.
clamp	if true, values that exceed the data domain are clamped to either the minimum or maximum range value. Default value: derived from scale config (true by default) Supported Types: only linear, pow, sqrt, and log
nice	If true, modifies the scale domain to use a more human-friendly number range (e.g., 7 instead of 6.96). Default value: true only for quantitative x and y scales and false otherwise.
zero	If true, ensures that a zero baseline value is included in the scale domain. Default value: true if the quantitative field is not binned.

References

Vega-Lite Scales spec

scale_y_linear *Linear Scale*

Description

Linear Scale

Usage

```
scale_y_linear(vl, domain = NULL, range = NULL, clamp = NULL,
  nice = NULL, zero = NULL)
```

Arguments

vl	Vega-Lite object
domain	Custom domain values. For quantitative data, this can take the form of a two-element array with minimum and maximum values.
range	The range of the scale represents the set of output visual values. Vega-Lite automatically determines appropriate range based on the scale's channel and type, but range property can be provided to customize range values.
clamp	if true, values that exceed the data domain are clamped to either the minimum or maximum range value. Default value: derived from scale config (true by default) Supported Types: only linear, pow, sqrt, and log
nice	If true, modifies the scale domain to use a more human-friendly number range (e.g., 7 instead of 6.96). Default value: true only for quantitative x and y scales and false otherwise.
zero	If true, ensures that a zero baseline value is included in the scale domain. Default value: true if the quantitative field is not binned.

References

Vega-Lite Scales spec

scale_y_log *Log Scale*

Description

Log Scale

Usage

```
scale_y_log(vl, domain = NULL, range = NULL, clamp = NULL, nice = NULL,
  zero = NULL)
```

Arguments

vl	Vega-Lite object
domain	Custom domain values. For quantitative data, this can take the form of a two-element array with minimum and maximum values.
range	The range of the scale represents the set of output visual values. Vega-Lite automatically determines appropriate range based on the scale's channel and type, but range property can be provided to customize range values.
clamp	if true, values that exceed the data domain are clamped to either the minimum or maximum range value. Default value: derived from scale config (true by default) Supported Types: only linear, pow, sqrt, and log
nice	If true, modifies the scale domain to use a more human-friendly number range (e.g., 7 instead of 6.96). Default value: true only for quantitative x and y scales and false otherwise.
zero	If true, ensures that a zero baseline value is included in the scale domain. Default value: true if the quantitative field is not binned.

References

Vega-Lite Scales spec

---

scale_y_ordinal *Ordinal Scale*

Description

Ordinal Scale

Usage

```
scale_y_ordinal(vl, band_size = NULL, padding = NULL)
```

Arguments

vl	Vega-Lite object
band_size	band size
padding	padding

References

Vega-Lite Scales spec

scale_y_pow *Power Scale*

Description

Power Scale

Usage

```
scale_y_pow(vl, domain = NULL, range = NULL, clamp = NULL, exp = NULL,
  nice = NULL, zero = NULL)
```

Arguments

vl	Vega-Lite object
domain	Custom domain values. For quantitative data, this can take the form of a two-element array with minimum and maximum values.
range	The range of the scale represents the set of output visual values. Vega-Lite automatically determines appropriate range based on the scale's channel and type, but range property can be provided to customize range values.
clamp	if true, values that exceed the data domain are clamped to either the minimum or maximum range value. Default value: derived from scale config (true by default) Supported Types: only linear, pow, sqrt, and log
exp	exponent
nice	If true, modifies the scale domain to use a more human-friendly number range (e.g., 7 instead of 6.96). Default value: true only for quantitative x and y scales and false otherwise.
zero	If true, ensures that a zero baseline value is included in the scale domain. Default value: true if the quantitative field is not binned.

References

Vega-Lite Scales spec

scale_y_quantile *Quantile Scale*

Description

Quantile Scale

Usage

```
scale_y_quantile(vl, domain = NULL, range = NULL, clamp = NULL,
  nice = NULL, zero = NULL)
```

Arguments

vl	Vega-Lite object
domain	Custom domain values. For quantitative data, this can take the form of a two-element array with minimum and maximum values.
range	The range of the scale represents the set of output visual values. Vega-Lite automatically determines appropriate range based on the scale's channel and type, but range property can be provided to customize range values.
clamp	if true, values that exceed the data domain are clamped to either the minimum or maximum range value. Default value: derived from scale config (true by default) Supported Types: only linear, pow, sqrt, and log
nice	If true, modifies the scale domain to use a more human-friendly number range (e.g., 7 instead of 6.96). Default value: true only for quantitative x and y scales and false otherwise.
zero	If true, ensures that a zero baseline value is included in the scale domain. Default value: true if the quantitative field is not binned.

References

Vega-Lite Scales spec

---

scale_y_quantize *Quantize Scale*

Description

Quantize Scale

Usage

```
scale_y_quantize(vl, domain = NULL, range = NULL, clamp = NULL,
  nice = NULL, zero = NULL)
```

Arguments

vl	Vega-Lite object
domain	Custom domain values. For quantitative data, this can take the form of a two-element array with minimum and maximum values.
range	The range of the scale represents the set of output visual values. Vega-Lite automatically determines appropriate range based on the scale's channel and type, but range property can be provided to customize range values.
clamp	if true, values that exceed the data domain are clamped to either the minimum or maximum range value. Default value: derived from scale config (true by default) Supported Types: only linear, pow, sqrt, and log

nice	If true, modifies the scale domain to use a more human-friendly number range (e.g., 7 instead of 6.96). Default value: true only for quantitative x and y scales and false otherwise.
zero	If true, ensures that a zero baseline value is included in the scale domain. Default value: true if the quantitative field is not binned.

References

Vega-Lite Scales spec

`scale_y_sqrt` *Sqrt Scale*

Description

Sqrt Scale

Usage

```
scale_y_sqrt(vl, domain = NULL, range = NULL, clamp = NULL, nice = NULL,
  zero = NULL)
```

Arguments

vl	Vega-Lite object
domain	Custom domain values. For quantitative data, this can take the form of a two-element array with minimum and maximum values.
range	The range of the scale represents the set of output visual values. Vega-Lite automatically determines appropriate range based on the scale's channel and type, but range property can be provided to customize range values.
clamp	if true, values that exceed the data domain are clamped to either the minimum or maximum range value. Default value: derived from scale config (true by default) Supported Types: only linear, pow, sqrt, and log
nice	If true, modifies the scale domain to use a more human-friendly number range (e.g., 7 instead of 6.96). Default value: true only for quantitative x and y scales and false otherwise.
zero	If true, ensures that a zero baseline value is included in the scale domain. Default value: true if the quantitative field is not binned.

References

Vega-Lite Scales spec

scale_y_threshold *Threshold Scale*

Description

Threshold Scale

Usage

```
scale_y_threshold(vl, domain = NULL, range = NULL, clamp = NULL,
  nice = NULL, zero = NULL)
```

Arguments

vl	Vega-Lite object
domain	Custom domain values. For quantitative data, this can take the form of a two-element array with minimum and maximum values.
range	The range of the scale represents the set of output visual values. Vega-Lite automatically determines appropriate range based on the scale's channel and type, but range property can be provided to customize range values.
clamp	if true, values that exceed the data domain are clamped to either the minimum or maximum range value. Default value: derived from scale config (true by default) Supported Types: only linear, pow, sqrt, and log
nice	If true, modifies the scale domain to use a more human-friendly number range (e.g., 7 instead of 6.96). Default value: true only for quantitative x and y scales and false otherwise.
zero	If true, ensures that a zero baseline value is included in the scale domain. Default value: true if the quantitative field is not binned.

References

Vega-Lite Scales spec

scale_y_time *Temporal Scale*

Description

Temporal Scale

Usage

```
scale_y_time(vl, domain = NULL, range = NULL, clamp = NULL, nice = NULL,
  zero = NULL)
```

Arguments

vl	Vega-Lite object
domain	Custom domain values. For quantitative data, this can take the form of a two-element array with minimum and maximum values.
range	The range of the scale represents the set of output visual values. Vega-Lite automatically determines appropriate range based on the scale's channel and type, but range property can be provided to customize range values.
clamp	if true, values that exceed the data domain are clamped to either the minimum or maximum range value. Default value: derived from scale config (true by default) Supported Types: only linear, pow, sqrt, and log
nice	If true, modifies the scale domain to use a more human-friendly number range (e.g., 7 instead of 6.96). Default value: true only for quantitative x and y scales and false otherwise.
zero	If true, ensures that a zero baseline value is included in the scale domain. Default value: true if the quantitative field is not binned.

References

Vega-Lite Scales spec

---

sort_def *Create a sort definition object*

Description

You can sort by aggregated value of another "sort" field by creating a sort field definition object. All three properties must be non-NULL.

Usage

```
sort_def(field, op = NULL, order = c("ascending", "descending"))
```

Arguments

field	the field name to aggregate over.
op	a valid aggregation operator.
order	either ascending or descending

Examples

```
vegalite() %>%
  add_data("https://vega.github.io/vega-editor/app/data/cars.json") %>%
  encode_x("Horsepower", type="quantitative", aggregate="mean") %>%
  encode_y("Origin", "ordinal", sort=sort_def("Horsepower", "mean")) %>%
  mark_bar()
```

`timeunit_x` *How to encode x-axis time values*

Description

How to encode x-axis time values

Usage

```
timeunit_x(vl, unit)
```

Arguments

`vl`	Vega-Lite object
`unit`	the property of a channel definition sets the level of specificity for a temporal field. Currently supported values are 'year', 'yearmonth', 'yearmonthday', 'yearmonthdate', 'yearday', 'yeardate', 'yearmonthdayhours' and 'yearmonthdayhoursminutes' for non-periodic time units & 'month', 'day', 'date', 'hours', 'minutes', 'seconds', 'milliseconds', 'hoursminutes', 'hoursminutesseconds', 'minutesseconds' and 'secondsmilliseconds' for periodic time units.

References

Vega-Lite Time Unit

Examples

```
vegalite() %>%
  cell_size(300, 300) %>%
 add_data("https://vega.github.io/vega-editor/app/data/unemployment-across-industries.json") %>%
  encode_x("date", "temporal") %>%
  encode_y("count", "quantitative", aggregate="sum") %>%
  encode_color("series", "nominal") %>%
  scale_x_time(nice="month") %>%
  scale_color_nominal(range="category20b") %>%
  axis_x(axisWidth=0, format="%Y", labelAngle=0) %>%
  axis_y(remove=TRUE) %>%
  timeunit_x("yearmonth") %>%
  mark_area(stack="normalize")
```

timeunit_y *How to encode y-axis time values*

Description

How to encode y-axis time values

Usage

```
timeunit_y(vl, unit)
```

Arguments

vl	Vega-Lite object
unit	the property of a channel definition sets the level of specificity for a temporal field. Currently supported values are 'year', 'yearmonth', 'yearmonthday', 'yearmonthdate', 'yearday', 'yeardate', 'yearmonthdayhours' and 'yearmonthdayhoursminutes' for non-periodic time units & 'month', 'day', 'date', 'hours', 'minutes', 'seconds', 'milliseconds', 'hoursminutes', 'hoursminutesseconds', 'minutesseconds' and 'secondsmilliseconds' for periodic time units.

References

Vega-Lite Time Unit

Examples

```
# see timeunit_y()
```

to_spec *Convert a spec created with widget idioms to JSON*

Description

Takes an htmlwidget object and turns it into a JSON Vega-Lite spec

Usage

```
to_spec(vl, pretty = TRUE, to_cb = FALSE)
```

Arguments

vl	a Vega-Lite object
pretty	if TRUE (default) then a "pretty-printed" version of the spec will be returned. Use FALSE for a more compact version.
to_cb	if TRUE, will copy the spec to the system clipboard. Default is FALSE.

Value

JSON spec

Examples

```
dat <- jsonlite::fromJSON('[
    {"a": "A","b": 28}, {"a": "B","b": 55}, {"a": "C","b": 43},
    {"a": "D","b": 91}, {"a": "E","b": 81}, {"a": "F","b": 53},
    {"a": "G","b": 19}, {"a": "H","b": 87}, {"a": "I","b": 52}
  ]')

vegalite() %>%
  add_data(dat) %>%
  encode_x("a", "ordinal") %>%
  encode_y("b", "quantitative") %>%
  mark_bar() -> chart

to_spec(chart)
```

---

vegalite *Create and (optionally) visualize a Vega-Lite spec*

Description

Create and (optionally) visualize a Vega-Lite spec

Usage

```
vegalite(description = "", renderer = c("svg", "canvas"), export = FALSE,
  source = FALSE, editor = FALSE, viewport_width = NULL,
  viewport_height = NULL, background = NULL, time_format = NULL,
  number_format = NULL)
```

Arguments

description	a single element character vector that provides a description of the plot/spec.
renderer	the renderer to use for the view. One of canvas or svg (the default)
export	if TRUE the *"Export as..."* link will be displayed with the chart.(Default: FALSE.)
source	if TRUE the *"View Source"* link will be displayed with the chart. (Default: FALSE.)
editor	if TRUE the *"Open in editor"* link will be displayed with the cahrt. (Default: FALSE.)
viewport_width, viewport_height	height and width of the overall visualziation viewport. This is the overall area reserved for the plot. You can leave these NULL and use cell_size instead but you will want to configure both when making faceted plots.
background	plot background color. If NULL the background will be transparent.

time_format	the default time format pattern for text and labels of axes and legends (in the form of D3 time format pattern). Default: %Y-%m-%d
number_format	the default number format pattern for text and labels of axes and legends (in the form of D3 number format pattern). Default: s

References

Vega-Lite top-level config

Examples

```
dat <- jsonlite::fromJSON('[
    {"a": "A","b": 28}, {"a": "B","b": 55}, {"a": "C","b": 43},
    {"a": "D","b": 91}, {"a": "E","b": 81}, {"a": "F","b": 53},
    {"a": "G","b": 19}, {"a": "H","b": 87}, {"a": "I","b": 52}
  ]')

vegalite() %>%
  add_data(dat) %>%
  encode_x("a", "ordinal") %>%
  encode_y("b", "quantitative") %>%
  mark_bar()
```

---

vegaliteOutput *Widget output function for use in Shiny*

Description

Widget output function for use in Shiny

Usage

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

Arguments

outputId	widget output id
width, height	widget height/width

Index