Package: vegalite (via r-universe)

August 13, 2024

Type Package

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

Version 2.0.1

Maintainer Bob Rudis <bob@rud.is>

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, jsonvalidate

Depends R (>= 3.0.0)

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

RoxygenNote 6.0.1.9000

VignetteBuilder knitr

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

RemoteUrl https://github.com/hrbrmstr/vegalite

RemoteRef HEAD

RemoteSha 385369840c6b396ba49a698e3c291aab8b5e404b

Contents

`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 `vieew_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"
    }
  }
}
```

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) %>%
  view_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_vl(range_step=8) %>%
  scale_color_nominal_vl(range=c("#EA98D2", "#659CCA")) %>%
  facet_col("age", "ordinal", padding=4) %>%
  axis_x(remove=TRUE) %>%
  axis_y(title="population", grid=FALSE) %>%
  view_config(stroke_width=0) %>%
  mark_bar()
```

`axis_vl` *General axis settings (all)*

Description

axis_vl 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.

The axis_vl function works with all possible channels, but axis_x, axis_y, axis_facet_row and axis_facet_col are offered as conveniences.

Usage

```
axis_vl(vl, chnl = "x", domain = NULL, grid = NULL, maxExtent = NULL,
  minExtent = NULL, orient = NULL, offset = NULL, position = NULL,
  zindex = NULL, format = NULL, labels = NULL, labelAngle = NULL,
  labelOverlap = NULL, labelPadding = NULL, ticks = NULL,
  tickCount = NULL, tickSize = NULL, values = NULL, title = NULL,
  titleMaxLength = NULL, titlePadding = NULL, remove = FALSE, ...)

axis_x(vl, ...)

axis_y(vl, ...)

axis_facet_col(vl, ...)

axis_facet_row(vl, ...)
```

Arguments

`vl`	Vega-Lite object
`chnl`	x, y, column, or row
`domain`, `grid`, `maxExtent`, `minExtent`, `orient`, `offset`, `position`, `zindex`	see axis docs

format, labels, labelAngle, labelOverlap, labelPadding
: see axis docs

ticks, tickCount, tickSize, values
: see axis docs

title, titleMaxLength, titlePadding
: see axis docs

remove
: see axis docs

...
: deprecated arguments

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_vl(range_step=8) %>%
  scale_color_nominal_vl(range=c("#EA98D2", "#659CCA")) %>%
  facet_col("age", "ordinal", padding=4) %>%
  axis_x(remove=TRUE) %>%
  axis_y(title="population", grid=FALSE) %>%
  view_config(stroke_width=0) %>%
  mark_bar()

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_vl(range_step=8) %>%
  scale_color_nominal_vl(range=c("#EA98D2", "#659CCA")) %>%
  facet_col("age", "ordinal", padding=4) %>%
  axis_x(remove=TRUE) %>%
  axis_y(title="population", grid=FALSE) %>%
  view_config(stroke_width=0) %>%
  mark_bar()
```

---

bin_vl *Group continuous data values (all)*

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_vl(vl, chnl = "x", min = NULL, max = NULL, base = NULL,
  step = NULL, steps = NULL, minstep = NULL, div = NULL,
  maxbins = NULL)

bin_x(vl, ...)

bin_y(vl, ...)
```

Arguments

vl	Vega-Lite object
chnl	the x or y channel.
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.
...	additional arguments passed to bin_vl

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_vl(range_step=8) %>%
  scale_color_nominal_vl(range=c("#EA98D2", "#659CCA")) %>%
  facet_col("age", "ordinal", padding=4) %>%
  axis_x(remove=TRUE) %>%
  axis_y(title="population", grid=FALSE) %>%
  view_config(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 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 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_config` *Deprecated* `cell` *functions*

Description

Vega-lite 2 no longer has `cell` and `facet` properties under `config`. Thus functions setting those properties have been deprecated. Some of the functionality can be acheived with `view_config` and `view_size`.

Usage

```
cell_config(vl, ...)

cell_size(vl, width = 200, height = 200, facet = FALSE)

facet_cell(vl, ...)

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

Arguments

`vl`	Vega-Lite object
`...`	additional arguments to pass to cell_config
`width`	width of cell
`height`	height of cell
`facet`	if facetting
`grid_color`	color of the grid between facets.
`grid_opacity`	`0.0-1.0`
`grid_offset`	offset for grid between facets.

References

Vega-Lite 2 config spec Vega-Lite 1 cell spec Vega-Lite 1 cell spec

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 *Encode all "channels"*

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.

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.

Create a horizontal ribbon of panels

Create a vertical ribbon of panels

Usage

```
encode_vl(vl, chnl = "x", field = NULL, type = "auto", value = NULL,
  aggregate = NULL, sort = NULL, padding = NULL, round = NULL,
  stack = NULL)

encode_x(vl, ...)

encode_y(vl, ...)

encode_color(vl, ...)

encode_shape(vl, ...)

encode_size(vl, ...)

encode_text(vl, ...)

encode_detail(vl, field = NULL, type = "auto", value = NULL, ...)

encode_order(vl, field = NULL, type = "auto", value = NULL, ...)

encode_path(vl, field = NULL, type = "auto", value = NULL, ...)

facet_col(vl, field = NULL, type = "auto", value = NULL, ...)

facet_row(vl, field = NULL, type = "auto", value = NULL, ...)
```

Arguments

vl	Vega-Lite object created by vegalite
chnl	a channel to encode like x, y, color, shape, size, text, detail, order, path
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	if field is not specified, a constant value in visual domain.
aggregate	perform aggregation 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
padding	facet padding
round	round values
stack	how to stack values, in case mark is bar or area. Should be "zero","center","normalize", or "none" or NA.
...	additional arguments to pass to encode_vl

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

Vega-Lite Faceting

Vega-Lite Faceting

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()
vegalite() %>%
  view_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()
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_vl(range_step=8) %>%
  scale_color_nominal_vl(range=c("#EA98D2", "#659CCA")) %>%
  facet_col("age", "ordinal") %>%
  axis_x(remove=TRUE) %>%
  axis_y(title="population", grid=FALSE) %>%
  view_config(stroke_width=0) %>%
  mark_bar()
```

`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, the JSON text of a spec, or the file path for a json 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")
```

JS *Mark character strings as literal JavaScript code*

Description

Mark character strings as literal JavaScript code

---

legend_vl *Legend settings (all)*

Description

Legend settings (all)

Usage

```
legend_vl(vl, chnl = "color", orient = NULL, offset = NULL,
  values = NULL, format = NULL, labelAlign = NULL, labelBaseline = NULL,
  labelColor = NULL, labelFont = NULL, labelFontSize = NULL,
  short_time_labels = NULL, symbolColor = NULL, symbolShape = NULL,
  symbolSize = NULL, symbolStrokeWidth = NULL, title = NULL,
  titleColor = NULL, titleFont = NULL, titleFontSize = NULL,
  titleFontWeight = NULL, remove = FALSE)

legend_color(vl, ...)
```

```
legend_size(vl, ...)

legend_shape(vl, ...)
```

Arguments

vl	a Vega-Lite object
chnl	a channel, by default vegalite creates legends for color, size, and shape
orient, offset, values	see legend docs
format, labelAlign, labelBaseline, labelColor, labelFont, labelFontSize, short_time_labels	see legend docs
symbolColor, symbolShape, symbolSize, symbolStrokeWidth	see legend docs
title, titleColor, titleFont, titleFontSize, titleFontWeight	see legend docs
remove	if TRUE, there will be no legend for this aesthetic.
...	additional arguments to pass to legend_vl

---

mark *Mark*

Description

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

Usage

```
mark(vl, mark = "circle", filled = NULL, color = NULL, fill = NULL,
  stroke = NULL, opacity = NULL, fillOpacity = NULL,
  strokeOpacity = NULL, strokeWidth = NULL, strokeDash = NULL,
  strokeDashOffset = NULL, stacked = NULL, interpolate = NULL,
  tension = NULL, orient = NULL, barSize = NULL, shape = NULL,
  size = NULL, tickSize = NULL, tickThickness = NULL)

mark_bar(vl, ...)

mark_circle(vl, ...)

mark_square(vl, ...)

mark_tick(vl, ...)
```

```
mark_line(vl, ...)

mark_area(vl, ...)

mark_point(vl, ...)

mark_text(vl, ...)
```

Arguments

vl	Vega-Lite object
mark	can be "bar", "circle", "square", "tick", "line", "area", "point", and "text". These directly set how the data are drawn and are similar geoms in ggplot2.
filled, color, fill, stroke	see config.mark color docs
opacity, fillOpacity, strokeOpacity	see config.mark opacity docs
strokeWidth, strokeDash, strokeDashOffset	see config.mark stroke docs
stacked	Defunct; Use in encode_x or encode_y
interpolate, tension	for line and area mark, the line interpolation method. value for interpolate can be "linear", "step-before", "step-after", "basis", "basis-open", "basis-closed", "bundle", "cardinal", "cardinal-open", "cardinal-closed", or "monotone", For tension, see D3's line interpolation.
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.
barSize	the size of the bars (width or height depending on orient)
shape	applicable to point mark, "circle", "square", "cross", "diamond", "triangle-up", or "triangle-down", or else a custom SVG path string.
size	for point, circle or square mark, the pixel area of a point.
tickSize	the size of ticks
tickThickness	the thickness of ticks.
...	additional arguments passed to mark

References

Vega-Lite Mark spec, Vega-Lite config.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()
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()
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()
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()
vegalite() %>%
  add_data("https://vega.github.io/vega-editor/app/data/cars.json") %>%
  encode_x("Horsepower", "quantitative") %>%
  encode_y("Cylinders", "ordinal") %>%
  mark_tick()
vegalite() %>%
  view_size(300, 300) %>%
  add_data("https://vega.github.io/vega-editor/app/data/driving.json") %>%
  encode_x("miles", "quantitative") %>%
  encode_y("gas", "quantitative") %>%
  encode_order("year", "temporal") %>%
  scale_x_linear_vl(zero=FALSE) %>%
  scale_y_linear_vl(zero=FALSE) %>%
  mark_line()
vegalite() %>%
  view_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_vl(scheme="category20b") %>%
  timeunit_x("yearmonth") %>%
  scale_x_time_vl(nice="month") %>%
  axis_x(format="%Y", labelAngle=0) %>%
  mark_area()
```

```
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()
vegalite() %>%
  view_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()
```

`renderVegalite` *Widget render function for use in Shiny*

Description

Widget render function for use in Shiny

Usage

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

renderVegaliteSpec(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_vl *Vega-Lite Scales*

Description

Vega-Lite Scales

Usage

```
scale_vl(vl, chnl = "x", type = "linear", domain = NULL, range = NULL,
  scheme = NULL, round = NULL, clamp = NULL, exponent = NULL,
  base = NULL, nice = NULL, zero = NULL, useRawDomain = NULL,
  band_size = NULL, range_step = NULL, padding = NULL,
  interpolate = NULL)

scale_x_linear_vl(vl, ...)

scale_y_linear_vl(vl, ...)

scale_x_pow_vl(vl, ...)

scale_y_pow_vl(vl, ...)

scale_x_sqrt_vl(vl, ...)

scale_y_sqrt_vl(vl, ...)

scale_x_log_vl(vl, ...)

scale_y_log_vl(vl, ...)

scale_x_quantize_vl(vl, ...)

scale_y_quantize_vl(vl, ...)

scale_x_quantile_vl(vl, ...)

scale_y_quantile_vl(vl, ...)

scale_x_ordinal_vl(vl, ...)

scale_y_ordinal_vl(vl, ...)

scale_x_threshold_vl(vl, ...)

scale_y_threshold_vl(vl, ...)
```

```
scale_x_time_vl(vl, ...)

scale_y_time_vl(vl, ...)

scale_color_nominal_vl(vl, ...)

scale_color_sequential_vl(vl, type = "ordinal", range = NULL, ...)

scale_shape_vl(vl, range = NULL, ...)
```

Arguments

vl	Vega-Lite object
chnl	x,y,color,shape
type	linear, log, pow, sqrt, quantize, quantile, threshold, time, ordinal
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.
scheme	color scheme to use
round	If true, rounds numeric output values to integers.
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
exponent	in the "pow" scale only, expresses each range value y as a power (exponential) function of the domain value x: y = mx^k + b where k is exponent
base	log base to use for log scale
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.
useRawDomain	If true, set scale domain to the raw data domain. If false, use the aggregated data domain for scale.
band_size	Deprecated – use range_step instead.
range_step	Width for each x or y ordinal band. This can be an integer value or a string "fit". For "fit", the band size will be automatically adjusted to fit the scale for the specified width (for x-axis) or height (for y-axis).
padding	For x and y channels, the padding is a multiple of the spacing between points. A reasonable value is 1.0, such that the first and last point will be offset from the minimum and maximum value by half the distance between points. (See D3's ordinalRangePoints() for illustration.)
interpolate	interpolation method to use for ranges. Legal values include rgb, hsl, hsl-long, lab, hcl, hcl-long, cubehelix and cubehelix-long
...	additional arguments to pass to scale_vl

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_vl(range_step=8) %>%
  scale_color_nominal_vl(range=c("#EA98D2", "#659CCA")) %>%
  facet_col("age", "ordinal") %>%
  axis_x(remove=TRUE) %>%
  axis_y(title="population", grid=FALSE) %>%
  view_config(stroke_width=0) %>%
  mark_bar()
```

---

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` *How to encode time values*

Description

How to encode time values

Usage

```
timeunit(vl, chnl = "x", unit)

timeunit_x(vl, unit)

timeunit_y(vl, unit)
```

Arguments

`vl`	Vega-Lite object
`chnl`	x,y
`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() %>%
  view_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", stack = "normalize") %>%
  encode_color("series", "nominal") %>%
  scale_x_time_vl(nice="month") %>%
  scale_color_nominal_vl(scheme="category20b") %>%
  axis_x(format="%Y", labelAngle=0) %>%
  axis_y(remove=TRUE) %>%
  timeunit_x("yearmonth") %>%
  mark_area()
```

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)
```

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.

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)
```

validate_vl *validate_vl*

Description

Validate the Vega-Lite widget against the schema

Usage

```
validate_vl(vl, verbose = TRUE)
```

Arguments

vl	Vega-Lite object
verbose	TRUE or FALSE – should error reason be added as "errors" attribute in case of invalid schema

Details

Use of this function requires jsonvalidate package to be installed.

Value

TRUE if valid, FALSE if not

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}
  ]')

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

validate_vl(vl)
```

---

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, padding = NULL, autosize = 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 view_size instead but you will want to configure both when making faceted plots.
padding	single number to be applied to all sides, or list specifying padding on each side, e.g list("top" = 5, "bottom" = 3, "left" = 2, "right" = 2). Unit is pixels.
autosize	sizing setting (autosize)
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
...	additional arguments

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")

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

Arguments

outputId	widget output id
width, height	widget height/width

view_config *view aesthetics*

Description

At its core, a Vega-Lite specification describes a single plot. This function works to format those views with all of the options avaiable under config.view.* 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 view. View configuration allows us to customize each individual single plot and each plot in a trellis plot.

Usage

```
view_config(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)

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

Arguments

vl	Vega-Lite object
width, height	width and height property of the view 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 view spec

Examples

```
vegalite() %>%
  view_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_vl(scheme="category20b") %>%
  timeunit_x("yearmonth") %>%
  scale_x_time_vl(nice="month") %>%
  axis_x(format="%Y", labelAngle=0) %>%
  mark_area()
```

Index