

Survey: Vega and Vega-Lite

Group 4: Patrick Draxler, Nikolina Jekic, Lukas Plechinger, and Josef Suschnigg

Course: Information Visualization
Summer Term 2019
Graz University of Technology

27 Jan 2019

Abstract

Vega and Vega-Lite are declarative, data-driven tools for creating powerful interactive data visualizations in the browser. Vega-Lite is built on top of Vega, a visualization grammar built using D3. Although Vega and D3 provide great flexibility for customized visualization designs, there are some limitations. With Vega and D3, a simple chart requires a lot of lines of code and specification for low-level components. By contrast, Vega-Lite inspired by Wilkinson's Grammar of Graphics, Wickham's ggplot2 and Tableau, is a higher-level language that simplifies the creation of common charts. In this survey, we provide an overview of Vega and Vega-Lite and evaluation of the tools through different examples.

© Copyright 2019 by the author(s), except as otherwise noted.

This work is placed under a Creative Commons Attribution 4.0 International (CC BY 4.0) licence.

Contents

Contents	ii
List of Figures	iii
List of Listings	v
1 Declarative Visualization - Grammar of Graphics	1
1.1 Grammar of Graphics	1
1.2 Fundamental components of the Grammar of Graphics	1
1.2.1 Data	1
1.2.2 Aesthetics	1
1.2.3 Scale	1
1.2.4 Geometry/Geometric objects	2
1.2.5 Facets	2
1.2.6 Coordinate system	2
1.2.7 Themes	2
2 Overview of Vega and Vega-Lite	3
2.1 What is Vega?	3
2.2 What is Vega-Lite?	3
2.3 D3.js	3
2.4 How do they work?	5
2.4.1 Parser	5
2.4.2 View	6
2.4.3 Renderer	7
2.5 Technology Stack Summary	7
3 Vega	8
3.1 Basic Elements of Vega	8
3.2 Basic Bar Chart Example	8
3.2.1 Visualization size and data set	9
3.2.2 Connect Data with Visualization Axes	9
3.2.3 Visualize the Bars	10
3.2.4 Interaction	10
3.3 Advanced Visualization Horizon Graph Example	11

4 Vega-Lite	12
4.1 Built-in Visualization Types	12
4.2 Basic Bar Chart Example in Vega-Lite	13
4.3 Differences between Vega and Vega-Lite?	16
5 Creation of a Custom Visualization using Vega	17
5.1 Bullet Graph	17
5.2 Creation of the Specification	18
6 Tools and Bindings	22
6.1 Tools for Authoring	22
6.1.1 Vega Live Editor	22
6.1.2 Vega Voyager 2	23
6.1.3 Vega Desktop	24
6.2 Bindings for Programming Languages	25
Bibliography	27

List of Figures

1.1	The fundamental components of the Grammar of Graphics. Redrawn from [Sarkar 2018]	2
2.1	Workflow of Vega and Vega-Lite Data Visualization	5
2.2	Example radial plot dataflow graph created by [<i>Dataflow Vis</i> 2019].	6
2.3	Schematic technical comparison of technologies used in regards with its possibilities and complexity	7
3.1	The Vega Bar Chart	9
3.2	Horizon Graph with 2 layers	11
3.3	Horizon Graph with 4 layers	11
4.1	The Vega-Lite Demo Bar Chart	14
4.2	The Vega Demo Bar Chart	16
5.1	Bullet graph example	17
5.2	Vega bullet graph custom visualization	21
6.1	The Vega Editor web-app [Screenshot taken by the authors of this survey]	22
6.2	Data viewer of the Vega editor [Screenshot taken by the authors of this survey]	23
6.3	Screenshot of Vega Voyager 2 [Screenshot taken by the authors of this survey]	24
6.4	Screenshot of Vega-Desktop [Screenshot taken by the authors of this survey]	24

List of Listings

2.1	D3.js line plot example	3
2.2	Vega and Vega-Lite schema declaration	5
2.3	Vega SQL data source example	6
3.1	Vega JSON data source example	9
3.2	Vega connect data source to visualization	9
3.3	Draw rectangles for bar chart	10
3.4	Add interactions to Vega visualization	10
4.1	Vega-Lite Bar Chart Example	13
5.1	Vega metadata specification	18
5.2	Vega data definition	18
5.3	Link data to two-dimensional scales	18
5.4	Add marks to custom visualization	19

Chapter 1

Declarative Visualization - Grammar of Graphics

1.1 Grammar of Graphics

The Grammar of Graphics has been developed by Leland Wilkinson [Wilkinson 2005]. It decomposes visualizations into their fundamental elements and describes rules how those elements are connected to each other, very similar to the grammar of a spoken language.

One of the most popular implementations of the Grammar of Graphics (GoG) is the visualization package `ggplot2` *ggplot2* [2019] for the programming language R [*R-Project* 2019]. But also the tools Vega and Vega-Lite which are described in this survey are using the principles of the GoG to declare visualization.

The goal of the development of the GoG was to provide an unified way to describe visualizations without the need to name the actual name of the visualization - instead of drawing a "Bar-Chart" you describe the elements of a Bar-Chart, for example, the bars like "blue rectangles for each value in the dataset A with a length proportional to the value of A". That is the reason why this is called "Declarative Visualization" - you declare which elements the visualization consists of.

1.2 Fundamental components of the Grammar of Graphics

The fundamental components can be described as a pyramid as seen in Figure 1.1.

1.2.1 Data

The most important thing is the data to plot. It might be necessary to preprocess the data to be in a well formatted state.

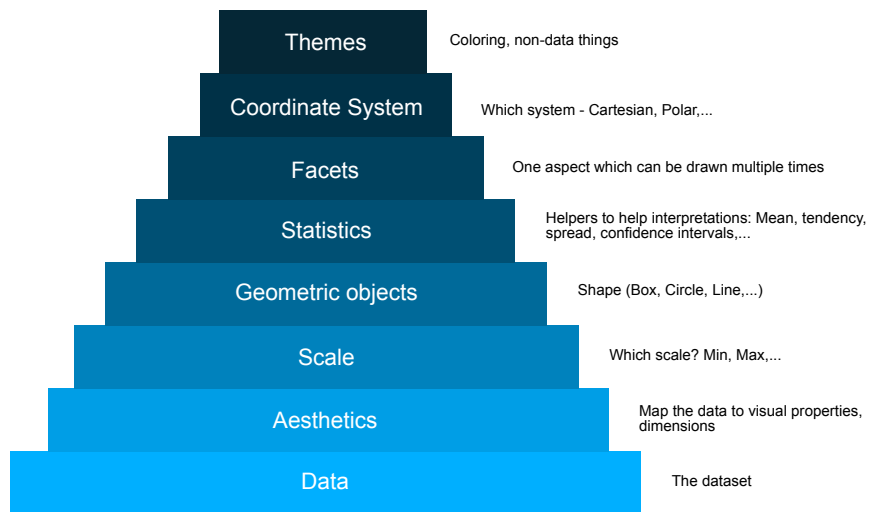
1.2.2 Aesthetics

The data is mapped to aesthetics which are the axes and the dimension of the plot itself.

1.2.3 Scale

After defining the aesthetics, the scale determines the size of those. Minimum or maximum values can be defined as well as mathematical scaling like a logarithmic or exponential scale.

Figure 1.1: The fundamental components of the Grammar of Graphics. Redrawn from [Sarkar 2018]



1.2.4 Geometry/Geometric objects

These are the elements which are mapped to the actual values in the data. This can be rectangles or boxes, for example, to draw the bars for a bar chart or points of a scatterplot.

1.2.5 Facets

If the same visualization should be drawn multiple times, for example the same temperature line-chart for every month, there is the possibility to define the chart as a facet and draw this facet multiple times.

1.2.6 Coordinate system

Definition of the coordinate system like polar coordinates or cartesian coordinates.

1.2.7 Themes

The look of the visualization. This is optional and not a real part of the GoG. Also there could be interactive elements like tooltips which are in this layer.

Chapter 2

Overview of Vega and Vega-Lite

2.1 What is Vega?

"Vega is a visualization grammar, a declarative language for creating, saving, and sharing interactive visualization designs. With Vega, you can describe the visual appearance and interactive behavior of a visualization in a JSON format, and generate web-based views using Canvas or SVG." [Vega – A Visualization Grammar 2019]. Therefore, Vega is an implementation of the declarative visualization concept grammar of graphics, as described in chapter 1. One of the most famous visualization libraries, also implementing those concepts is ggplot [ggplot2 2019] for the R programming language [R-Project 2019]. Hence, Vega shows lots of similarities in comparison to ggplot, whereas Vega is based on web technologies and mainly used in web environments.. Vega uses the powerful and complex JavaScript visualization library D3.js [Bostock et al. 2011] as a basis for visualization and user interaction. A more detailed explanation of Vega is given in chapter 3.

2.2 What is Vega-Lite?

Vega-Lite is setup on Vega and is able to use predefined Vega visualizations. Its main advantage is the low complexity on how to create visualization, because they are based on those predefinitions. Customization in Vega-Lite visualizations can be added like in a common Vega documents and predefinitions can also be overridden. Vega-Lite is further described in chapter 4.

2.3 D3.js

D3.js is a highly complex, but powerful web-based visualization library, whereas powerful means, that almost no limits, at least for 2D visualizations, are given. As a disadvantage it is very complex and one need to programmatically take care of many basic visualization tasks. As an example, for a simple line plot visualization those abstract tasks may be: (1) connect data points by a line (2) draw and set scales (3) set borders (4) highlight data points by drawing circles. Also, the visualization is not proper encapsulated from the data, and is strongly connected on CSS definitions. Therefore, D3.js is more of a visualization framework, then a ready to use visualization library. Here is an example [D3 v5 Line Chart 2019] of a simple line plot, which is a one-liner in many other visualization libraries:

```
<!-- Load in the d3 library -->
<script src="https://d3js.org/d3.v5.min.js"></script>
<script>

// 2. Use the margin convention practice
var margin = {top: 50, right: 50, bottom: 50, left: 50}
, width = window.innerWidth - margin.left - margin.right // Use the
  window's width
```

```

, height = window.innerHeight - margin.top - margin.bottom; // Use
  the window's height

// The number of datapoints
var n = 21;

// 5. X scale will use the index of our data
var xScale = d3.scaleLinear()
  .domain([0, n-1]) // input
  .range([0, width]); // output

// 6. Y scale will use the randomly generate number
var yScale = d3.scaleLinear()
  .domain([0, 1]) // input
  .range([height, 0]); // output

// 7. d3's line generator
var line = d3.line()
  .x(function(d, i) { return xScale(i); }) // set the x values for the
    line generator
  .y(function(d) { return yScale(d.y); }) // set the y values for the
    line generator
  .curve(d3.curveMonotoneX) // apply smoothing to the line

// 8. An array of objects of length N. Each object has key -> value
  pair, the key being "y" and the value is a random number
var dataset = d3.range(n).map(function(d) { return {"y": d3.
  randomUniform(1)() } })

// 1. Add the SVG to the page and employ #2
var svg = d3.select("body").append("svg")
  .attr("width", width + margin.left + margin.right)
  .attr("height", height + margin.top + margin.bottom)
  .append("g")
  .attr("transform", "translate(" + margin.left + "," + margin.top +
    ")");

// 3. Call the x axis in a group tag
svg.append("g")
  .attr("class", "x axis")
  .attr("transform", "translate(0," + height + ")")
  .call(d3.axisBottom(xScale)); // Create an axis component with d3.
  axisBottom

// 4. Call the y axis in a group tag
svg.append("g")
  .attr("class", "y axis")
  .call(d3.axisLeft(yScale)); // Create an axis component with d3.
  axisLeft

// 9. Append the path, bind the data, and call the line generator
svg.append("path")
  .datum(dataset) // 10. Binds data to the line
  .attr("class", "line") // Assign a class for styling
  .attr("d", line); // 11. Calls the line generator

// 12. Appends a circle for each datapoint
svg.selectAll(".dot")

```

```

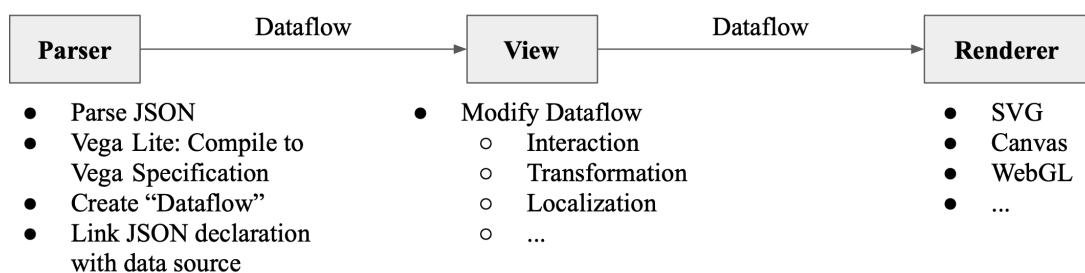
.data(dataset)
.enter().append("circle") // Uses the enter().append() method
.attr("class", "dot") // Assign a class for styling
.attr("cx", function(d, i) { return xScale(i) })
.attr("cy", function(d) { return yScale(d.y) })
.attr("r", 5)
.on("mouseover", function(a, b, c) {
  console.log(a)
  this.attr('class', 'focus')
})
.on("mouseout", function() { })
</script>

```

Listing 2.1: D3.js line plot example

2.4 How do they work?

Figure 2.1: Workflow of Vega and Vega-Lite Data Visualization



As mentioned above, Vega-Lite is set up on Vega, whereas Vega is set up on D3.js. An illustration on how a visualization is created, is given in Figure 2.1. In general the workflow consists of three subsequent components: (1) Parser, (2) Viewer and (3) Renderer. The components are explained in the following subsections:

2.4.1 Parser

The basis of any Vega and Vega-Lite File are JSON files, caused by Vega's focus on web and browser technologies. Therefore at first the JSON file needs to be parsed based on the "schema" variable of the JSON's root object. It can be either a Vega or a Vega-Lite definition:

```

"$schema": "https://vega.github.io/schema/vega/v5.json"
"$schema": "https://vega.github.io/schema/vega-lite/v3.json",

```

Listing 2.2: Vega and Vega-Lite schema declaration

Since Vega-Lite is a simplification/predefinition of Vega, for Vega-Lite files the first step is to compile it to a valid Vega specification. When a Vega specification is available, the parser creates a data-flow graph, which consists of all computations needed to map data to visual elements, like circles, axes, rectangles, but also user interactions on a higher abstract level. This dataflow will in the next step compiled to JavaScript code. Another task of the parser is to connect the data source to the JSON declaration. Vega

can handle data a JSON format easily, however there are other ways to pass data to the Vega visualization. As an example for an alternative data source, a select statement to access data from a SQL databases can be applied by Vega [Vega SQL 2019]:

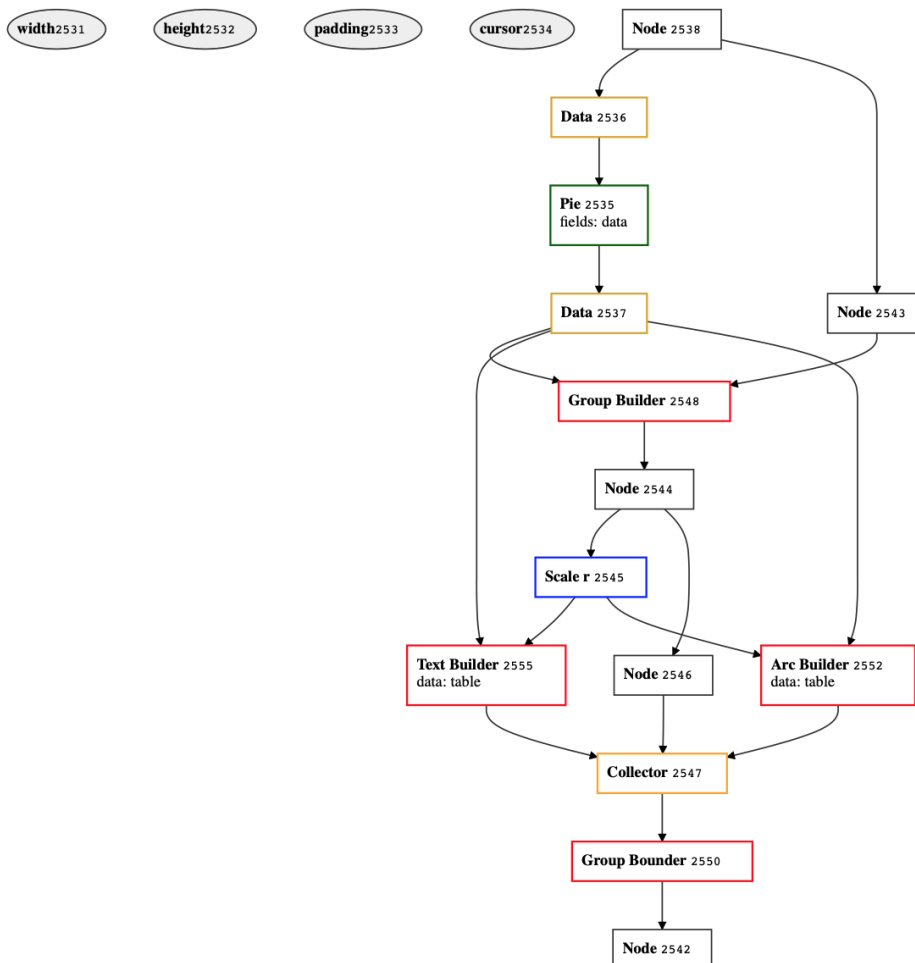
```
"data": [
  {
    "name": "tweets",
    "sql": "SELECT goog_x as x, goog_y as y, tweets_nov_feb.rowid FROM
      tweets_nov_feb"
  }
]
```

Listing 2.3: Vega SQL data source example

2.4.2 View

After the Vega file has been parsed, the view can be created from dataflow descriptions. The view is interactive, therefore hover and click events are also added to the visualization model. Vega dataflow can be highly dynamic, so creating the view can be quite complex. To visualize the creation of views by dataflows an open source tool "Dataflow Vis" is accessible on GitHub [Dataflow Vis 2019]. It has an online Vega editor, which dataflow graph can be dynamically visualized and explored. An example for a radial plot is given in Figure 2.2.

Figure 2.2: Example radial plot dataflow graph created by [Dataflow Vis 2019].

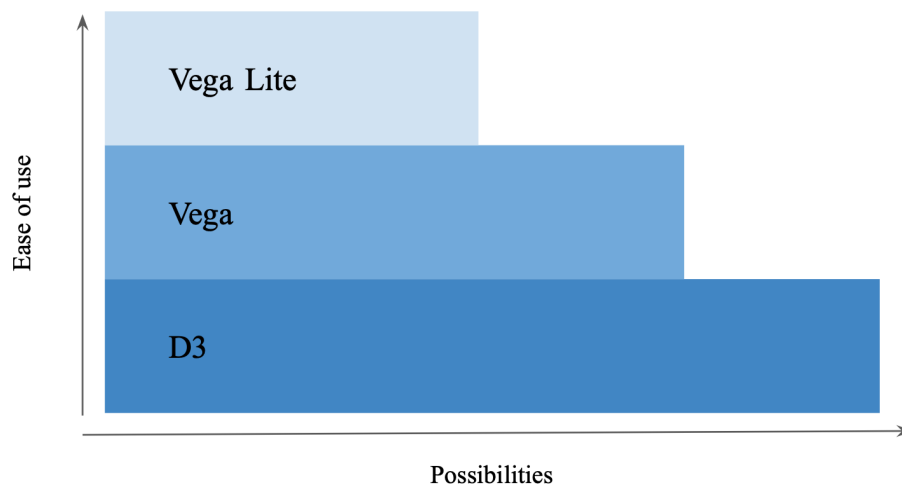


2.4.3 Renderer

The interactive Vega visualization can be either rendered as SVG, Canvas and WebGL, depending on the users need.

2.5 Technology Stack Summary

Figure 2.3: Schematic technical comparison of technologies used in regards with its possibilities and complexity



To summarize this chapter Vegas building blocks are visualized in Figure 2.3. It is set up on D3 with the highest possibilities, but on the other hand very high complexity in usage. Vega therefore is easier to use with a little less possibilities, since the dataflow creation and the complex parts of the visualization, especially when interaction are taken into account, is handled by the Vega library and the declarative grammar of graphics language. As the easiest to use technology Vega-Lite has the fewest possibilities. In general, Vega-Lite uses complete and pre defined Vega declarations, which can be used with or without further user customizations.

Chapter 3

Vega

3.1 Basic Elements of Vega

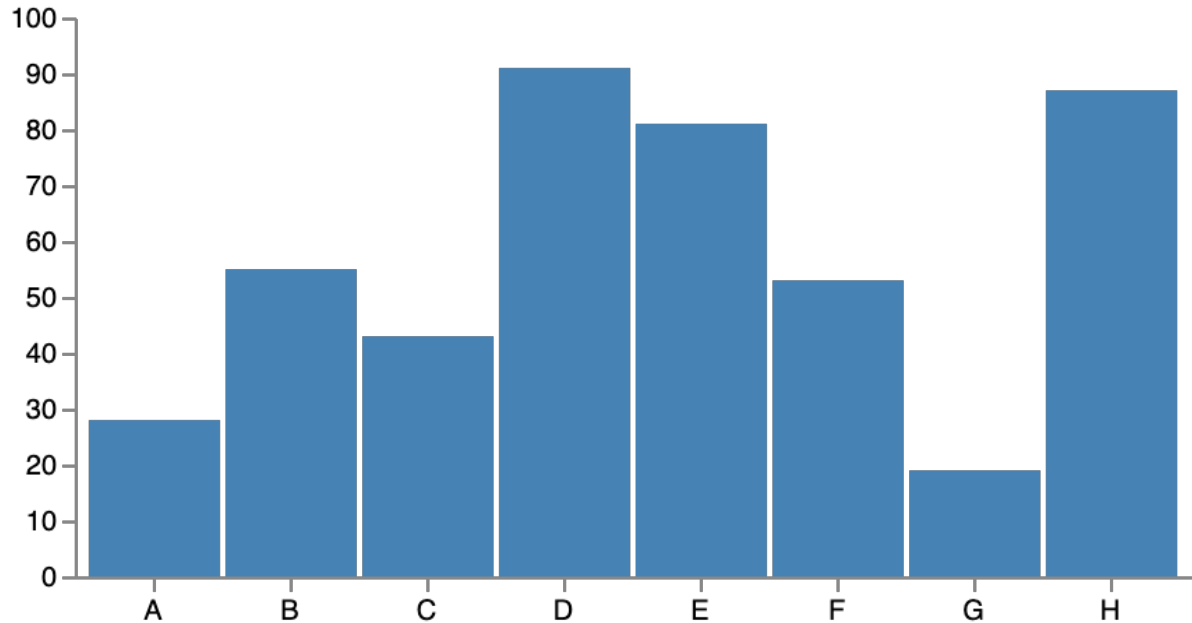
Vega is a visualization grammar that provides basic building block for wide range of visualization design. Example of Vega specification is presented in listings of the Chapter 3. Components of Vega are :

- Data
- Visualization size (size of canvas, uses SVG viewBox)
- Scales
 - Map data values to visuals (color, pixels, etc.)
- Marks
 - Enter, exit, hover and update events
 - Basic visual elements (rect, line, etc.)
- Signals (Interactivity)
 - Dynamic variables (for e.g. tooltips)
 - Parameterize visualization
 - Update due to external Events (Mouse, Keyboard, API)

A Vega specification defines an interactive visualization in a JSON format. Code is readable for user and not complex like in other languages e.g. D3. In Vega specification, it is possible to load data from the web, derived from a previously defined data set or to embed values directly into specifications. Definition for scales function allows us to map those data values to visual properties like color, position, size. Axes visualize scales using ticks and labels to help viewers interpret a chart. Marks are the main elements of visualization, determining which kind of marks (rect, area, line, symbol..) will be used. Along side with familiar visual encoding building block Vega introduces reactive ones to do interaction design. These events feed signals, which are dynamic variables. Their values automatically update visualization.

3.2 Basic Bar Chart Example

The creation of the visualization is similar to SVG, but looking at the data definition, it gets clear, that the data is independent from the visualization.

Figure 3.1: The Vega Bar Chart

3.2.1 Visualization size and data set

```
{
  "$schema": "https://vega.github.io/schema/vega/v5.json",
  "width": 400,
  "height": 200,
  "padding": 5,

  "data": [
    {
      "name": "table",
      "values": [
        {"year": "2018", "rate": 1249},
        {"year": "2017", "rate": 1265},
        {"year": "2016", "rate": 1152},
        {"year": "2015", "rate": 1068},
        {"year": "2014", "rate": 1200},
        {"year": "2013", "rate": 1221},
        {"year": "2012", "rate": 1287}
      ]
    }
  ],
}
```

Listing 3.1: Vega JSON data source example

3.2.2 Connect Data with Visualization Axes

```
"scales": [
  {
    "name": "xscale",
    "type": "band",
    "domain": {"data": "table", "field": "year"},
    "range": "width",
  }
]
```

```

    "padding": 0.05,
    "round": true
  },
  {
    "name": "yscale",
    "domain": {"data": "table", "field": "rate"},
    "nice": true,
    "range": "height"
  }
],

"axes": [
  { "orient": "bottom", "scale": "xscale" },
  { "orient": "left", "scale": "yscale" }
],

```

Listing 3.2: Vega connect data source to visualization

3.2.3 Visualize the Bars

```

"marks": [
  {
    "type": "rect",
    "from": {"data": "table"},
    "encode": {
      "enter": {
        "x": {"scale": "xscale", "field": "year"},
        "width": {"scale": "xscale", "band": 1},
        "y": {"scale": "yscale", "field": "rate"},
        "y2": {"scale": "yscale", "value": 0}
      },
      "update": {
        "fill": {"value": "steelblue"}
      },
      "hover": {
        "fill": {"value": "red"}
      }
    }
  }
],

```

Listing 3.3: Draw rectangles for bar chart

3.2.4 Interaction

Interactions can also be defined by Vega. As an example a mouseover tooltip visualization is given:

```

"signals": [
  {
    "name": "tooltip",
    "value": {},
    "on": [
      {"events": "rect:mouseover", "update": "datum"},
      {"events": "rect:mouseout", "update": "{}"}
    ]
  }
],
{
  "type": "text",
  "encode": {

```

```

"enter": {
  "align": {"value": "center"},
  "baseline": {"value": "bottom"},
  "fill": {"value": "#333"}
},
"update": {
  "x": {"scale": "xscale", "signal": "tooltip.year", "band":
    0.5},
  "y": {"scale": "yscale", "signal": "tooltip.rate", "offset":
    -2},
  "text": {"signal": "tooltip.rate"},
  "fillOpacity": [
    {"test": "isNaN(tooltip.rate)", "value": 0},
    {"value": 1}
  ]
}
}
}
]
}

```

Listing 3.4: Add interactions to Vega visualization

3.3 Advanced Visualization Horizon Graph Example

Vega supports declarative interaction using a reactive model. Horizon graphs present time-series data in compact space, by dividing an area chart into layers. By clicking, user can change number of layers. Changing input events, number of layers, Vega will automatically update visualization. Even though, new selection will change the chart size, the spatial resolution of the area chart will stay constant.

Figure 3.2: Horizon Graph with 2 layers

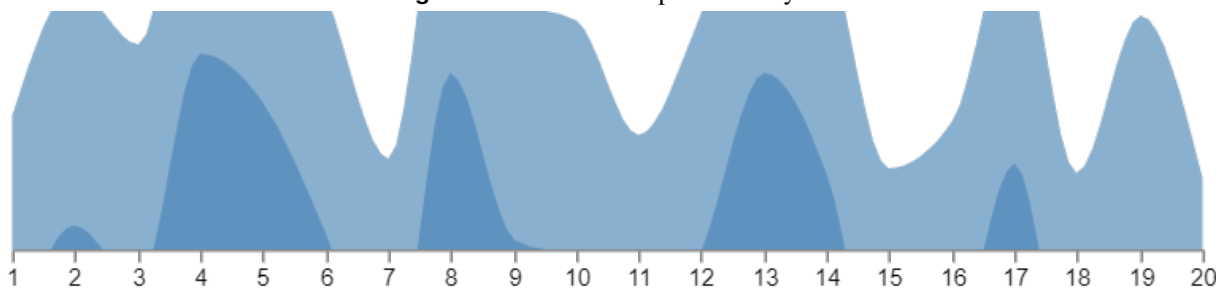
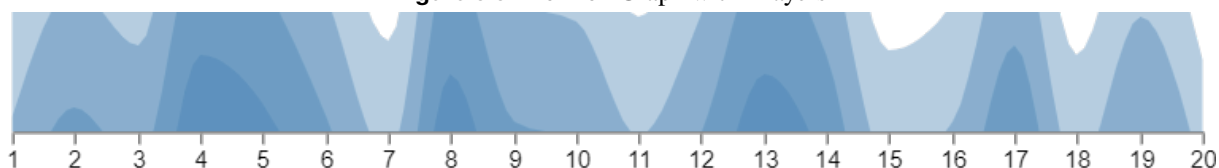


Figure 3.3: Horizon Graph with 4 layers



Chapter 4

Vega-Lite

Low-level grammars such as Protovis, D3, and Vega are useful for explanatory data visualization or as a basis for customized analysis tools. For research visualization, higher-level grammar, such as ggplot2, and grammatically based systems such as Tableau (gender Polaris) are usually preferred by users. They enable users to quickly visualize partially given specifications for visualization as they apply default values to solving low-level details to produce visualizations. However, existing high-level languages provide limited support for interactivity. An analyst can enable a predefined set of common techniques such as linking, panning, zooming, etc. For custom direct manipulation, interaction needs a callback for event handling. Recognizing that callbacks can be difficult and requires complex static analysis.

Interaction techniques can be specified in Vega using reactive signals. While these improvements facilitate the programming and targeting the desired interactive visualization, they remain at a low-level. A detailed specification disturbs quick writing and interferes with a systematic exploration of alternative designs.

Vega-Lite is a high-level grammar for interactive graphics that enables fast specification of interactive data visualizations. Vega-Lite combines a traditional grammar of graphics, providing visual encoding rules and a composition algebra for layered and multi-view displays.

In the next section, we will evaluate Vega-Lite with different examples that demonstrate a specification of both customized interaction methods and common techniques such as panning, zooming, and linked selection.

4.1 Built-in Visualization Types

Vega-Lite provides functionality for quickly creating common statistical graphics. The following types of graphics are build in Vega-Lite:

- Scatter & Strip Plots
- Line Charts
- Area Charts & Streamgraphs
- Table-based Plots
- Composite Plots
- Box Plots
- Layering
- Multi view Plots

- Maps (Geographic Displays)
- Interactivity

Additional Plots only available in Vega:

- Tree diagrams
- Network diagrams
- Parallel Coordinates
- Word Cloud
- Timeline
- Beeswarm

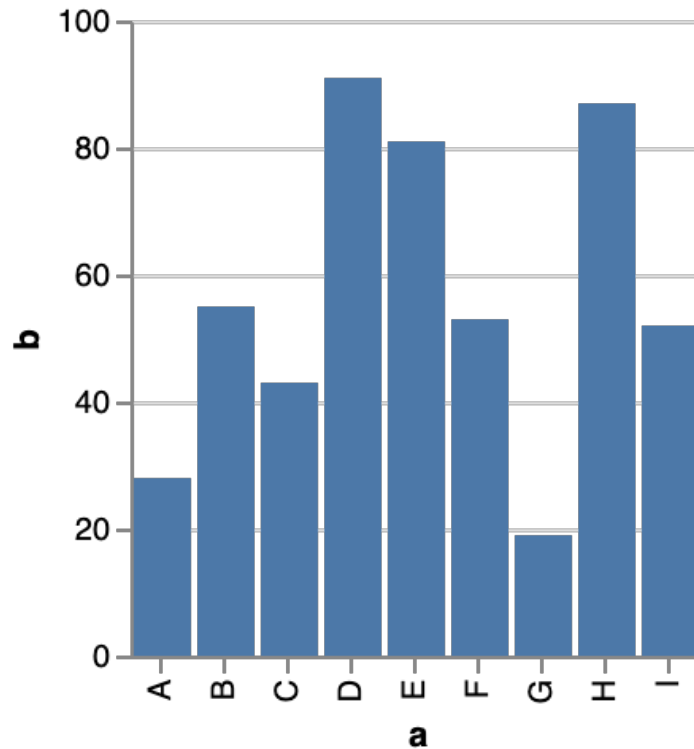
4.2 Basic Bar Chart Example in Vega-Lite

Vega-Lite specifications are significantly shorter (about 1/10th) than Vega specifications. When drawing the same dataset with both Vega and Vega-Lite, one can see a huge difference in the size of the listing. Example Bar Chart in Vega-Lite with 16 lines.

```
{
  "$schema": "https://vega.github.io/schema/vega-lite/v3.json",
  "description": "A simple bar chart with embedded data.",
  "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": "ordinal"},
    "y": {"field": "b", "type": "quantitative"}
  }
}
```

Listing 4.1: Vega-Lite Bar Chart Example

Figure 4.1: The Vega-Lite Demo Bar Chart



The same dataset visualized in Vega takes 95 lines. In Vega, you have to construct axis and legends and you have to decide how to map the data to visual properties. When using Vega-Lite we don't need to worry about low-level details such as of axes, legends, and scales.

```
{
  "$schema": "https://vega.github.io/schema/vega/v5.json",
  "width": 400,
  "height": 200,
  "padding": 5,

  "data": [
    {
      "name": "table",
      "values": [
        {"category": "A", "amount": 28},
        {"category": "B", "amount": 55},
        {"category": "C", "amount": 43},
        {"category": "D", "amount": 91},
        {"category": "E", "amount": 81},
        {"category": "F", "amount": 53},
        {"category": "G", "amount": 19},
        {"category": "H", "amount": 87}
      ]
    }
  ],

  "signals": [
    {
      "name": "tooltip",
```

```

    "value": {},
    "on": [
      {"events": "rect:mouseover", "update": "datum"},
      {"events": "rect:mouseout", "update": "{}"}
    ]
  }
],

"scales": [
  {
    "name": "xscale",
    "type": "band",
    "domain": {"data": "table", "field": "category"},
    "range": "width",
    "padding": 0.05,
    "round": true
  },
  {
    "name": "yscale",
    "domain": {"data": "table", "field": "amount"},
    "nice": true,
    "range": "height"
  }
],

"axes": [
  { "orient": "bottom", "scale": "xscale" },
  { "orient": "left", "scale": "yscale" }
],

"marks": [
  {
    "type": "rect",
    "from": {"data": "table"},
    "encode": {
      "enter": {
        "x": {"scale": "xscale", "field": "category"},
        "width": {"scale": "xscale", "band": 1},
        "y": {"scale": "yscale", "field": "amount"},
        "y2": {"scale": "yscale", "value": 0}
      },
      "update": {
        "fill": {"value": "steelblue"}
      },
      "hover": {
        "fill": {"value": "red"}
      }
    }
  },
  {
    "type": "text",
    "encode": {
      "enter": {
        "align": {"value": "center"},
        "baseline": {"value": "bottom"},
        "fill": {"value": "#333"}
      },
      "update": {

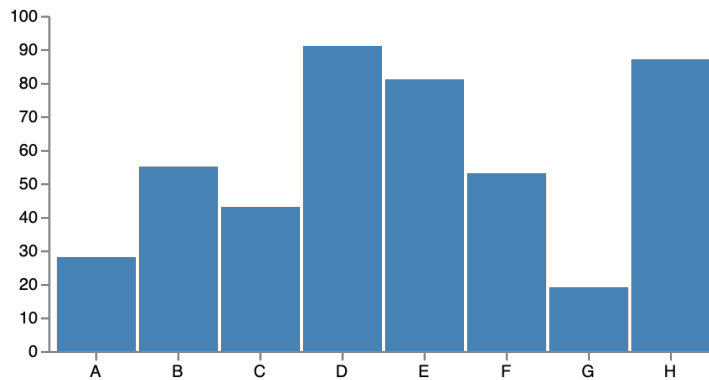
```

```

"x": {"scale": "xscale", "signal": "tooltip.category", "band
      ": 0.5},
"y": {"scale": "yscale", "signal": "tooltip.amount", "offset
      ": -2},
"text": {"signal": "tooltip.amount"},
"fillOpacity": [
{"test": "datum === tooltip", "value": 0},
{"value": 1}
]
}
}
}
]
}
}

```

Figure 4.2: The Vega Demo Bar Chart



4.3 Differences between Vega and Vega-Lite?

Vega-Lite is a lightweight version built on top of the Vega specification (subset of Vega). It supports existing chart types (bar chart, line chart, area chart, scatter plot, heatmap, trellis plots, ...) and functionality for data transformations (sorting, aggregation, faceting) as well as interactivity. Its goal is to offer a fast and easy functionality for data analysis. A portable JSON format is used, which is then compiled into the Vega language. Some visualizations (trees, graphs, word cloud, custom chart types...) and interaction techniques which Vega provides, can not be expressed in Vega-Lite.

Chapter 5

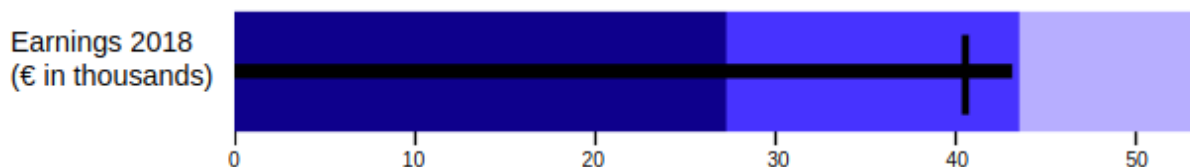
Creation of a Custom Visualization using Vega

In order to test the limits of Vega, we wanted to create a custom chart type, which is not simply available within Vega. We also wanted to see how much effort it takes to create a specification from scratch. For our test we choose to create a bullet graph, which is described in the next section.

5.1 Bullet Graph

This section is based on the design specification from Stephen Few. The bullet graph was designed by Stephen Few. It was originally designed for dashboards, because it is capable of displaying much information within a small space. An example of a bullet graph is shown in the figure below and the single components are describe above.

Figure 5.1: Bullet graph example



The bullet graph consists of three main elements which are mandatory. The first element is the text label, which shall describe the data which is displayed. The second one is the quantitative scale. It consists of text marks and labels which are equally distributed along the graph. Normally the scale starts at zero, but it is also possible to start with another value, if starting at zero is not appropriate. The third element, the featured measure, is the actual data. It is normally visualized with a centered horizontal bar, but it could also be marked with a sign.

In addition to the mandatory elements, there are two additional which are commonly used. The first is the comparative measure. This element is a vertical line, which should be less dominant than the featured measure. Its purpose is to display a target value. The second additional element are the qualitative ranges. These ranges visualize the quality of the data, encoded by different shades of a color. For example the first 65 percent are dark and symbolizes bad, the next 20 percent are a bit lighter and stand for an acceptable value and the last 15 percent are light represents good.

5.2 Creation of the Specification

The following specification is based on the official specification of Vega. The whole specification was done by scratch and the first step was to setup the schema, size and position of the visualization. The schema is set by simply referencing the used Vega schema. The height, width and padding are then set with the desired size in pixel. Vega also defines a auto-sizing option, but unfortunately we could not make it work. The result of the first part of the specification is shown below.

```
"$schema": "https://vega.github.io/schema/vega/v5.json",
"width": 1000,
"height": 500,
"padding": 5
```

Listing 5.1: Vega metadata specification

The second step was to define the data set. Our example data set looks like the following:

```
"data": [
  {
    "name": "table",
    "values": [
      {"company": "A", "amount": 28, "target": 30},
      {"company": "B", "amount": 55, "target": 45},
      {"company": "C", "amount": 43, "target": 20},
      {"company": "D", "amount": 91, "target": 120}
    ]
  }
]
```

Listing 5.2: Vega data definition

The data set is imaginary and shall represent companies with their names, the current amount they earned and the target amount they want to earn. For our example we decided to hard-code the data within the specification, because it is just a small one, but it would be possible to link to data from other sources. Afterwards the scales and axes were defined like this:

```
"scales": [
  {
    "name": "yscale",
    "type": "band",
    "domain": {"data": "table", "field": "company"},
    "range": "height",
    "padding": 0.5
  },
  {
    "name": "xscale",
    "domain": {"data": "table", "field": "target"},
    "domainMax": 140,
    "range": "width"
  }
],
"axes": [
  { "orient": "left", "scale": "yscale", "domain": false, "title": "Company"}
]
```

```
{ "orient": "bottom", "scale": "xscale", "title":"Money earned"}
]
```

Listing 5.3: Link data to two-dimensional scales

First, the scales get a name and then they are mapped to a data field with the "domain" field. The "yscale" is set to be of type band and then a padding is set, such that the bars have enough space in-between. In the "xscale" we also set the "domainMax" to be 140. This is a workaround for the qualitative scales and is used such that all bars in the bullet graph have the same length, even if they have a small value. After the scales are defined, the axes are mapped to them. In this case we have two axes, where the y-axis shows the companies and the x-axis shows the current money earned. In the y-scale, the domain field is set to false in order to hide the line which indicates the axes. The last step for the bullet graph specification was to add so called marks:

```
"marks": [
  {
    "type": "rect",
    "from": {"data":"table"},
    "encode": {
      "enter": {
        "y": {"scale": "yscale", "field": "company"},
        "height":{"scale":"yscale", "band": true},
        "width": {"scale":"xscale","value": 140},
        "fill": {"value":"#A9CCE3"}
      }
    }
  },
  {
    "type": "rect",
    "from": {"data":"table"},
    "encode": {
      "enter": {
        "y": {"scale": "yscale", "field": "company"},
        "height":{"scale":"yscale", "band": true},
        "width":{"scale":"xscale", "field": "target", "mult":1},
        "fill": {"value":"#2471A3"}
      }
    }
  },
  {
    "type": "rect",
    "from": {"data":"table"},
    "encode": {
      "enter": {
        "y": {"scale": "yscale", "field": "company"},
        "height":{"scale":"yscale", "band": true},
        "width":{"scale":"xscale", "field": "target", "mult":0.8},
        "fill": {"value":"#154360"}
      }
    }
  },
  {
    "type": "rect",
    "from": {"data":"table"},
    "encode": {
      "enter": {
        "y": {"scale": "yscale", "field": "company", "offset":15},
```

```

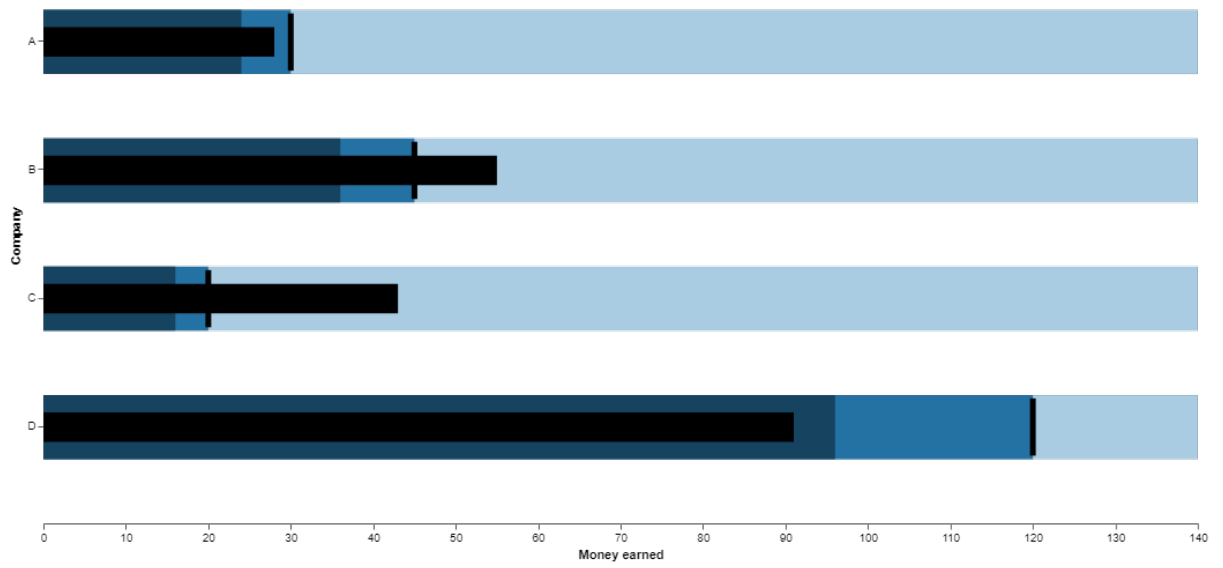
    "height":{"scale":"yscale", "band":true,"offset":-30},
    "width":{"scale":"xscale", "field":"amount"},
    "fill": {"value": "black"}
  }
},
{
  "type": "rect",
  "from": {"data":"table"},
  "encode": {
    "enter": {
      "y": {"scale": "yscale", "field": "company", "offset":3},
      "x": {"scale": "xscale", "field":"target","offset":-2.5},
      "height":{"scale":"yscale", "band":true,"offset":-6},
      "width":{"value": 5 },
      "fill": {"value": "black"}
    }
  }
}
]

```

Listing 5.4: Add marks to custom visualization

The marks are used to visualize the actual data, but it is also possible to draw for example rectangles of fixed size and this was done for the first mark in our example. Here, the "y" field is mapped to the "y-scale", but the "x" is mapped to 140 to have bars of the same size, as mentioned above. The "x" field of the next two marks are then mapped to the target value. They additionally have a multiplier added, which represents the quantitative scales. The marks are drawn in the sequence they are specified, which means the second and third overlay the first one. The fourth mark represents the current value of the earnings and thus is mapped to the data-field "amount". This mark has a offset set for the "y" and "height" field. This is set such that the bar gets thinner than the ones of the quantitative scales. The offset in the "y" shifts the bar down the defined pixels. Because the bar is shifted down, also the height has to be decreased and since we want the bar to be centered within the quantitative scales, we have to reduce the height by the doubled size of the offset. The last mark is representing the comparative measure. It is mapped to the companies and the target values and also has a fixed width. The result of our specification is shown in the figure below.

Figure 5.2: Vega bullet graph custom visualization



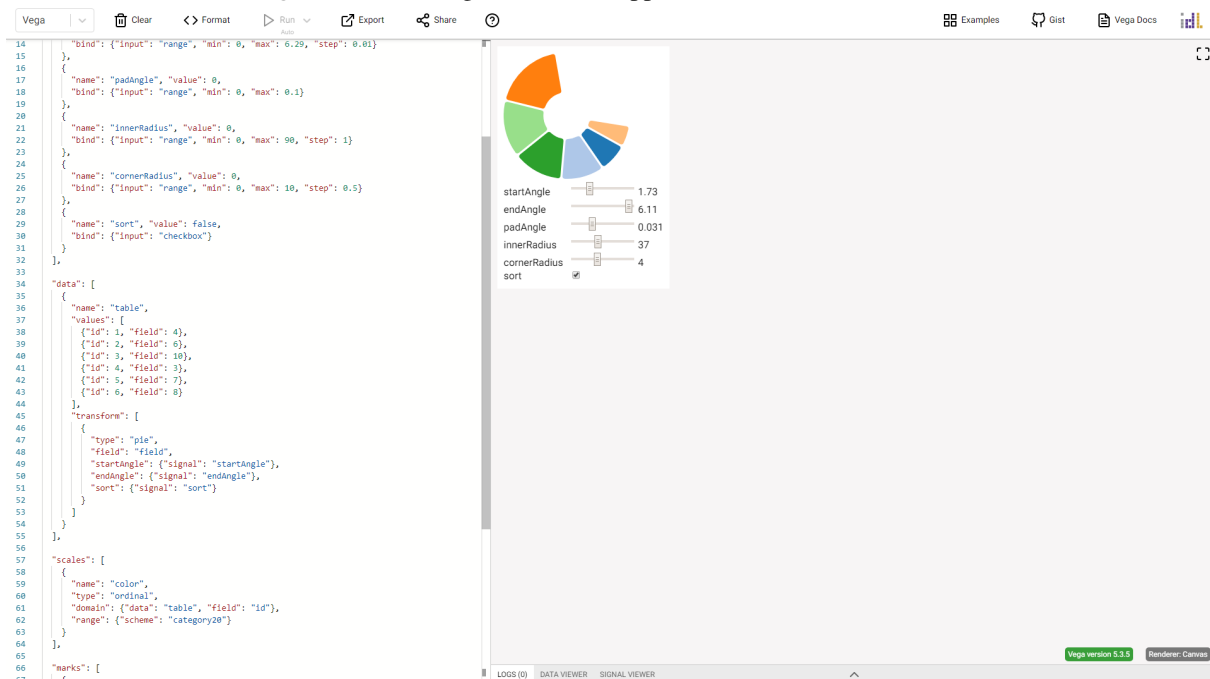
Chapter 6

Tools and Bindings

6.1 Tools for Authoring

6.1.1 Vega Live Editor

Figure 6.1: The Vega Editor web-app [Screenshot taken by the authors of this survey]

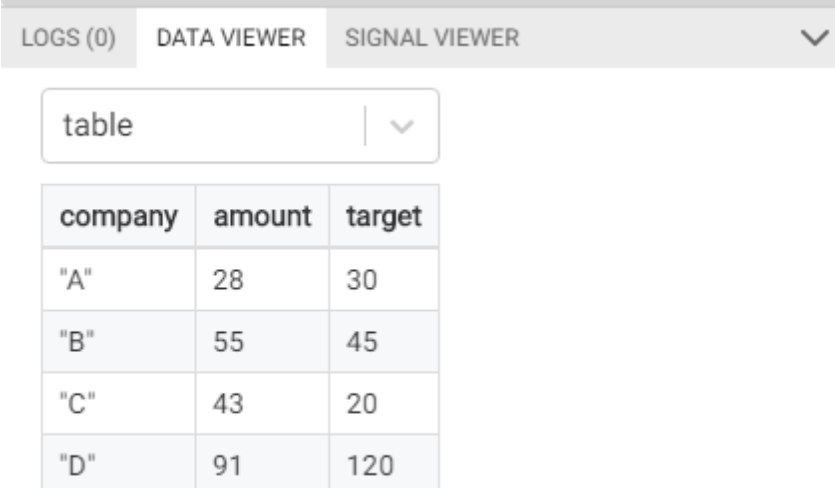


The Vega Editor is a freely online available web-app, which supports upon creating and viewing specifications. It supports Vega and Vega-Lite and automatically detects if a Vega or Vega-Lite specification is pasted into the editor. At its current state it provides over 60 examples for Vega and over 120 for Vega-Lite out of the box. These examples can be easily imported into the editor and then reworked to the special needs of the user. A screenshot of the editor is shown in Figure 6.1.

Other main features of the Vega Editor are:

- Live update of the view while editing the specification.
- Changing between Canvas and SVG rendering.
- Automatic formatting of the specified JSON.
- Exporting as PGN, SVG and JSON.
- Sharing specifications by links.
- Intellisense for automatic completion.
- Data and signal viewer as depicted in Figure 6.2

Figure 6.2: Data viewer of the Vega editor [Screenshot taken by the authors of this survey]



company	amount	target
"A"	28	30
"B"	55	45
"C"	43	20
"D"	91	120

6.1.2 Vega Voyager 2

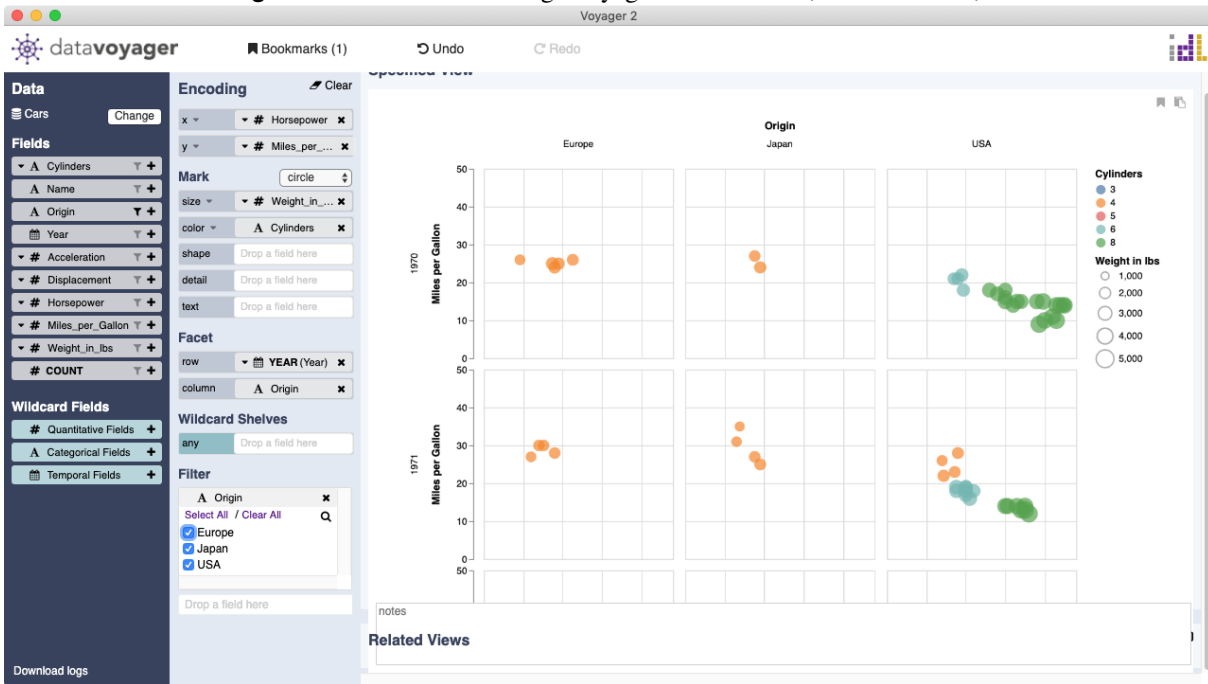
Vega Voyager 2 [Voyager2 2019] is an data exploration and visualization tool inspired by Tableau and Polaris. After loading a dataset, the user is able to interactively explore and visualize the dataset by dragging and dropping different variables of the dataset into properties like x and y coordinates, color, size, shape, etc. which then affect the look of the visualization.

There is also the possibility to filter the data before visualizing it.

To simplify this process, the tool generates an "Univariate Summary", which is a visualization of all the different values within every field in the dataset. This summary can be used to get an initial overview of the data to then combine different fields in the final dataset.

The visualizations can then be exported as a Vega-Lite specification file.

Figure 6.3: Screenshot of Vega Voyager 2 [Screenshot taken by the authors of this survey]

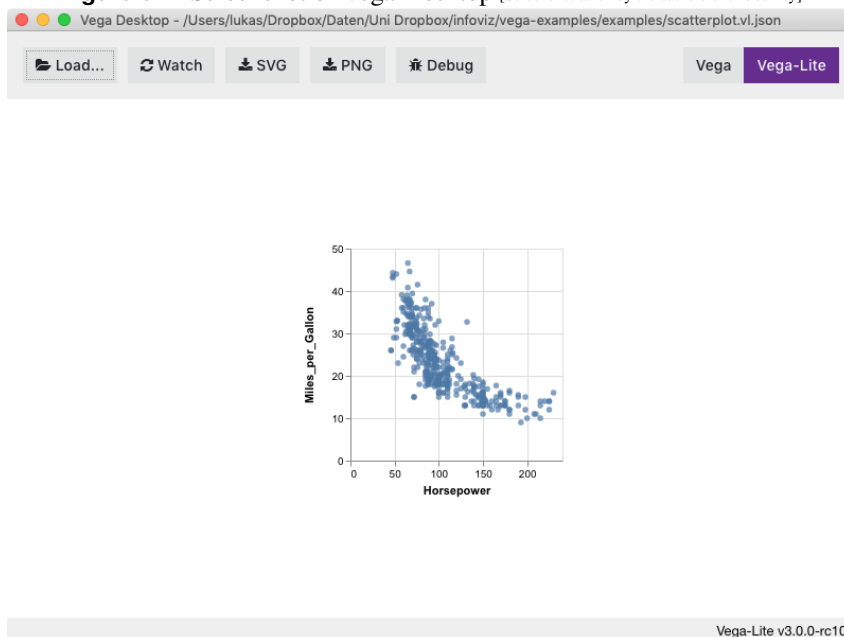


6.1.3 Vega Desktop

Vega Desktop [Desktop 2019] is a tool to view Vega or Vega-Lite specification files. A user can open specifications, enable watching so that the visualization changes automatically when the JSON-Specification has been changed and it is also possible to export visualizations as SVG and PNG.

One shortcoming of Vega Desktop is, that the built-in version of Vega and Vega-Lite is lagging behind the Vega development. The built-in version of the current Vega Desktop is 4.4.0 but at the time of writing, the current version on Github is 5.4.0 [Repository 2019].

Figure 6.4: Screenshot of Vega-Desktop [Screenshot taken by the authors of this survey]



6.2 Bindings for Programming Languages

Due to the use of JSON for the specification, it is fairly simple to generate specification files as every common programming language has support for generating text or JSON files.

Vega and Vega-Lite are developed in Javascript, so the only native possibility to generate graphics using one of them is by using a browser or Javascript Environments such as Node.JS.

The command line interface (CLI) `vega-cli` can be used to generate images such as SVG or PNG out of specification files. The CLI is developed using Node.JS. A developer can execute the CLI to generate graphics in the SVG or PNG format from within a programming language which supports execution of 3rd party programs.

To include visualizations in an application with a GUI, a web-view with support for Javascript is needed. Most modern GUI-frameworks have support for that.

However, there is no official interface for 3rd party programming languages. Some unofficial implementations are:

- Altair for python [Vallandingham 2018]
- VegaLite.jl - Vega-Lite for Julia [VegaLite.jl 2019]
- Elm-Vega - for creating Vega-Lite specifications in ELM [Elm-Vega 2019]
- Vegas for Scala and Spark [Vegas 2019]

Bibliography

- Bostock, Michael, Vadim Ogievetsky, and Jeffrey Heer [2011]. *D³ data-driven documents*. IEEE transactions on visualization and computer graphics 17.12 (2011), pages 2301–2309 (cited on page 3).
- D3 v5 Line Chart* [2019]. 20 May 2019. <https://bl.ocks.org/gordlea/27370d1eea8464b04538e6d8ced39e89> (cited on page 3).
- Dataflow Vis* [2019]. 20 May 2019. <https://github.com/vega/dataflow-vis> (cited on page 6).
- Desktop, Vega [2019]. *Vega Lite Wrapper for Julia*. Github Repository. 05 May 2019. <https://github.com/vega/vega-desktop> (cited on page 24).
- Elm-Vega [2019]. *Vega(Lite) Wrapper for Elm*. Github Repository. 05 May 2019. <https://github.com/gicentre/elm-vega> (cited on page 25).
- ggplot2* [2019]. 05 May 2019. <https://ggplot2.tidyverse.org/> (cited on pages 1, 3).
- R-Project* [2019]. 05 May 2019. <https://www.r-project.org/about.html> (cited on pages 1, 3).
- Repository, Vega [2019]. *Vega on Github*. Github Repository. 05 May 2019. <https://github.com/vega/vega> (cited on page 24).
- Sarkar, Dipanjan [2018]. *A Comprehensive Guide to the Grammar of Graphics for Effective Visualization of Multi-dimensional Data*. Blog article. 12 Sep 2018. <https://towardsdatascience.com/a-comprehensive-guide-to-the-grammar-of-graphics-for-effective-visualization-of-multi-dimensional-1f92b4ed4149> (cited on page 2).
- Vallandingham, Jim [2018]. *An Introduction to Altair*. Blog article. 23 Mar 2018. https://vallandingham.me/altair_intro.html (cited on page 25).
- Vega – A Visualization Grammar* [2019]. 26 Mar 2019. <https://vega.github.io/vega/> (cited on page 3).
- Vega SQL* [2019]. 20 May 2019. https://www.omnisci.com/docs/latest/6_VegaAtaGlance.html (cited on page 6).
- VegaLite.jl* [2019]. *Vega Lite Wrapper for Julia*. Github Repository. 05 May 2019. <https://github.com/fredo-dedup/VegaLite.jl> (cited on page 25).
- Vegas* [2019]. *Vegas - The missing Mathplotlib for Scala*. Website. 05 May 2019. <https://www.vegas-viz.org/> (cited on page 25).
- Voyager2* [2019]. *Voyager2*. Online Tool. 05 May 2019. <http://vega.github.io/voyager/> (cited on page 23).
- Wilkinson, Leland [2005]. *The Grammar of Graphics*. Berlin, Heidelberg: Springer-Verlag, 2005. ISBN 0387245448. doi:10.1007/978-1-4757-3100-2 (cited on page 1).