# Group 4 Project Report: Extending Vega-Lite

Patrick Draxler, Nikolina Jekic, Lukas Plechinger and Josef Suschnigg
706.057 Information Visualisation SS 2019
Graz University of Technology

01 Jul 2019

## Abstract

Vega and Vega-Lite are JavaScript implementations of Leland Wilkinson's grammar of graphics concept enabling the creation of flexible information visualization by a descriptive language. Vega-Lite offers some visualization types, whereas common charts and graphs are missing in the example gallery of the Vega-Lite website. In this project work, we investigate on how to implement missing common visualization techniques to Vega-Lite, what limitations occur and how to fix those limitations by i.e., extending the Vega-Lite compiler.

# Contents

# List of Figures

# List of Listings

# Chapter 1

# Introduction to Vega and Vega-Lite

## 1.1 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.. 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.

## 1.2 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.

## 1.3 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.

# Chapter 2

# Project Goals

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. Vega-Lite, a high-level grammar that enables rapid specification of interactive data visualizations. Nevertheless, in the current version of Vega-Lite only common visualization techniques exists. Therefore, we investigate the implementation of common visualization types and their limitations.

The main points that we investigated in our project work are:

- **Limitations of Vega-Lite** Chapter 4 lists in more details limitations of Vega-Lite, specially when implementing new visualization types to Vega-Lite, we often reached limits of the current Vega implementation.
- **Eliminate limitations by extending the compiler** Chapter 3 explains how to elimate current limitations in Vega-Lite by extending compiler.
- **Extending Vega-Lite with new visualization techniques** In Chapter 5, we explain more details about our practical part. We tried to implement new visualization techniques that are not available in Vega-Lite. We have described our approach and answered question how to implement new visualization techniques in Vega-Lite. Likewise we have given an explanation in the case that implementation of a specific visualization technique is not possible at this time.

# Chapter 3

# Extending the Compiler

## 3.1 Introduction to the Compiler

There is very litte information about the compiler available. Basically, the compiler reads a Vega-Lite specification and transforms it to a equivalent Vega specification.

The compiler steps are only illustrated in the file `compile.ts`.

The steps are described as:

1. Initialize the configuration
2. Normalize and autosize values.
3. Build the model, transform the normalized specification to the Model, which is a tree structure.
4. Enrich the Model with components. The components are intermediate representations which are equivalent to Vega specifications (1:1 mapping in assembly phase). This intermediate format is needed because later steps are also able to alter the specifications.
5. Modify and optimize the dataflow so that the Vega-Lite data specification (for example only a single datasource is allowed in Vega-Lite). Also transformations are transformed into their Vega equivalent in this step.
6. All model components are now in their final form and can be assembled to a Vega specification.

After examining the source code and the issues on Github [Github 2019d], we came to the conclusion that most of the highly requested issues and feature requests are not within our skill level. For example, in order to implement polar coordinates, a complete redesign would be necessary and the head of the development team said, that he would need several weeks in order to implement that. In fact, the request for multiple coordinate systems has been made back in 2017 [Github 2019e]. For that reason, we tried to implement new transformations.

## 3.2 Implement new transformations

In order to implement a new transformation in Vega-Lite, one needs the following prerequisites:

- The transformation must be already implemented in Vega.
- The output data format must be compatible with Vega-Lite.

In order to get an idea on how to implement a new transformation, pull requests which added new transformations have been examined, in particular the one which added "Pivot transform" [Github 2019a].

As an example, for the Count Pattern transformation, the following files had to be added or changed:

| Property | Type | Description |
|---|---|---|
| field | Field | Required. The data field containing the text data. |
| pattern | String | A string containing a well-formatted regular expression, defining a pattern to match in the text. All unique pattern matches will be separately counted. The default value is `[\w']+`, which will match sequences containing word characters and apostrophes, but no other characters. |
| case | String | A lower- or upper-case transformation to apply prior to pattern matching. One of `lower`, `upper` or `mixed` (the default). |
| stopwords | String | A string containing a well-formatted regular expression, defining a pattern of text to ignore. For example, the value `(foo\|bar\|baz)` will treat the words `foo`, `bar` and `baz` as stopwords that should be ignored. The default value is the empty string, indicating no stop words. |
| as | String[] | The output fields for the text pattern and occurrence count. The default is `["text", "count"]`. |

**Table 3.1:** The parameters available for Count Pattern Transform. Taken from the original documentation [Github 2019b].

1. add the Interface `CountPatternTransform` to the file `src/transform.ts`. Also provide a function `isCountPattern(t: Transform)` which returns `true` if the given Transform is a CountPatternTransform.

2. Create the file `src/compile/data/countpattern.ts` which contains the class `CountPatternTransformNode` .

3. In this class, there is a method `assemble()` which transforms an instance of `CountPatternTransformNode` to the equivalent Vega specification node class `VgCountPatternTransform` which has been created in the file `src/vega/schema.ts`.

4. In the File `src/compile/data/assemble.ts`, an new entry for the `CountPatternTransformNode` must be added.

5. Finally, in the parser `src/compile/data/parse.ts`, everything is glued together. A new rule which calls `isCountPattern(t: Transform)` is added, and if this evaluates to `true`, the parser is instructed to add a new instance of `CountPatternTransformNode`.

After compiling via the command `yarn build`, the new schema `build/vega-lite-schema.json` is compiled and also the script for createing a Vega specification out of an Vega-Lite specification is generated in `build/vl2vg` and is ready to be used with the new specifications by calling `build/vl2vg < vega-lite-spec.json > <vega-spec-output.json>`.

The changes which have been made for this project can be found in our repository [SS2019 2019].

## 3.3 Count Pattern Transform

The Count Pattern Transform provides a way to search for a regular expression and counts the values of this RegEx within a text. There is also the possibility to specify exclusions. All parameters are listed in Table 3.1. An example on how the transformation transforms the data is provided in Figure Figure 3.1.

```
[
  {"comment": "value is between 12 and 43"},
  {"comment": "value is 12"},
  {"comment": "value may be 5, 12, 13 or 43"}
]

"transform": [
  {
    "countpattern":{
      "field": "comment",     (datafield)
      "pattern": "\\d+",      (regex to extract & count)
      "stopwords": "(5|13)"   (regex for ignoring)
    }
  }
]

[
  {"text": "12", "count": 3},
  {"text": "43", "count": 2}
]
```

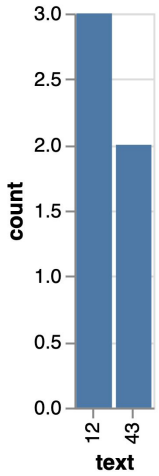


**Figure 3.1:** Example how Count Pattern transforms the data.

| Property | Type | Description |
|---|---|---|
| fields | Field[] | The data fields that should be copied over in the projection. If unspecified, all fields will be copied using their existing names. |
| as | String[] | For each corresponding field in the fields array, indicates the output field name to use for derived data objects. |

**Table 3.2:** The parameters available for Project Transform. Taken from the original documentation [Github 2019c].

## 3.4 Project Transform

Project Transform works like the SELECT of an SQL-Statement and changes field names or flatten nested data. The parameters of project transform are available in Table Table 3.1 and an overview on how the data is transformed is available in Figure Figure 3.2.

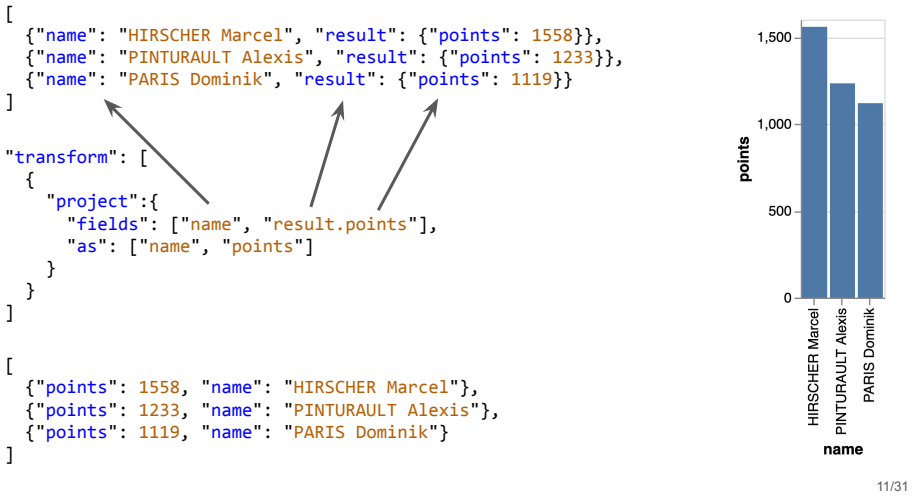


**Figure 3.2:** Example how Project transforms the data.

# Chapter 4

# Limitations of Vega-Lite

During our project work, especially when implementing new visualization types to Vega-Lite, we often reached limits of the current Vega implementation. Therefore in the next chapter, we emphasize those limitations by real examples. We assume that the reasons why specific visualizations are not existing in Vega-Lite yet lie in those limitations and are giving an overview as follows:

- **Axis** Even if Vega-Lite supports multiple Axis, mapping different scales on several axis is not possible. For example, among others parallel coordinates visualization need such scales for a proper implementation.
- **Projections** Many visualization types can be projected on the cartesian two-dimensional coordinate system. However, some visualization techniques need more axes or dimensions (i.e., radar chart), which therefore are not possible to implement in Vega-Lite.
- **Labels** Vega-Lite offers the capability of controlling axes labels in terms of font size, font color or tick frequency. On feature, which is missing, is the text values of tick labels. For example, data is transformed to be visualized on a specific position in the chart, but cannot be changed to the original value for labeling.
- **Data Transformations** Our concerns regarding the data transformation capabilities of Vega-Lite can be summarized in two points: (1) No recursion or hierarchies are possible, since the data structure of Vega-Lite is stuck two tabular or two-dimensional data. (2) Not all data transformation of Vega are enabled in Vega-Lite, which is a strong limitation in our opinion.

# Chapter 5

# Adding Visualization Types

For our practical project we planned to realize visualization techniques that are not available in Vega-Lite yet. Due to the limitations, we explained in the previous section, our expectations were to run into the limits caused by the current Vega-Lite implementation. In this chapter we present either the approach of new visualization techniques to Vega-Lite or we emphasize why a specific visualization technique cannot be realized in Vega-Lite. The selection of visualization techniques in this chapter is based on our assumption that visualizations don't exist for Vega-Lite, if they are not listed in the example gallery of the Vega-Lite website. However, we selected visualization techniques that are well known or that can be found in the visualization technique collection website "xeno.graphics".

## 5.1 Paired Bar Chart

Paired bar chart (also pyramid chart or mirrored bar chart) is a visualization technique to compare to variables, that depend on a third variable (i.e., time). It allows quick comparison of those two variables by facing them in horizontal bar charts. As an example for such a paired bar chart in, a bivariate time series is visualized in a paired bar chart. In Figure 5.1 an example is shown: the y-axis represents the time variable, whereas months as granularity level is chosen. In the x-axis two variables are visualized by bar charts of either red or blue color. Our Vega-Lite approach is illustrated by JSON code in Listing 5.1. In the listing only data values of the month of January is added in lines 6 + 7. From the data structure is is visible that each variable (or value) is related to a month and a group ("Product A + B"). The tricky part of our approach can be found in line 30, where we simply take the negative value of the group "Product B", to achieve the paired bar chart visualization. To empathize the limits in comparison to the example in Figure 5.1, we could not add values as labels to the bars. Another problem we had, was that the the transformation of one group to negative values enabled the paired bar chart visualization, but we were not able to change the tick labels in the x-axis accordingly (transform back to positive values for tick labels). The resulting Vega-Lite figure is shown in Figure 5.2.

## 5.2 Parallel Coordinates

Parallel coordinates is a visualization technique for multivariate data, whereas each data record is visualized by connected lines over several parallel y-axis representing each dimension of the data. The Vega-Lite example for the IRIS data set [Dua and Graff 2017] is shown in Figure 5.3. The limitation of that implementation in Vega-Lite was the difficulty of changing axis ticks label and the amount of ticks. As shown in the example, the y-axis on the left "petalLenght" has three tick value of 1, 3.95 and 6.9. Those values are either the minimum value, the maximum value and the average of the minimum and maximum value. As main limitations, we were not able to add more or proper ticks (i.e., no float values), having different scales for different axes or add typical interactions.

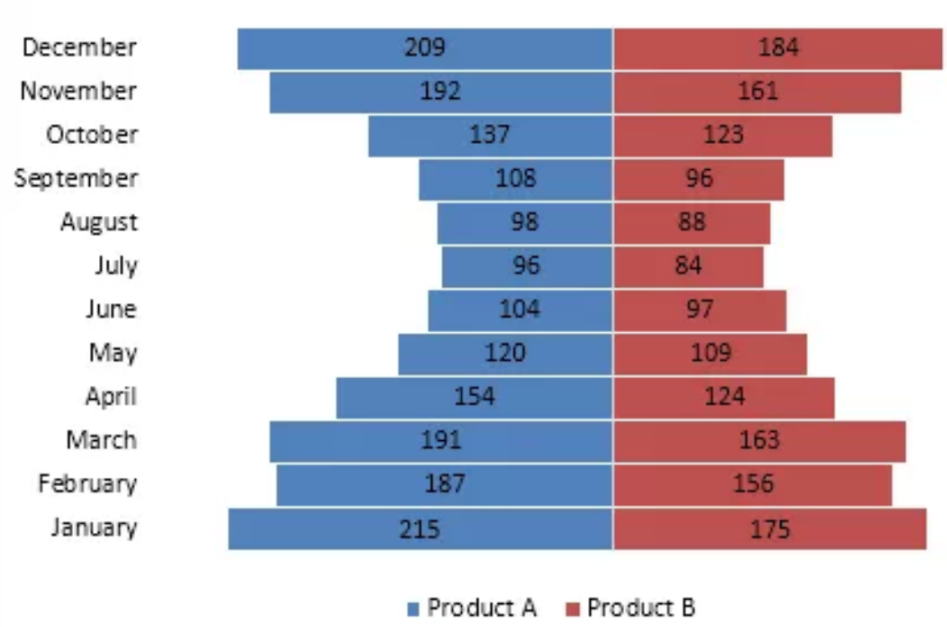


**Figure 5.1:** Example pair bar chart in Excel [*Excel Paired Bar Chart Example* 2019]

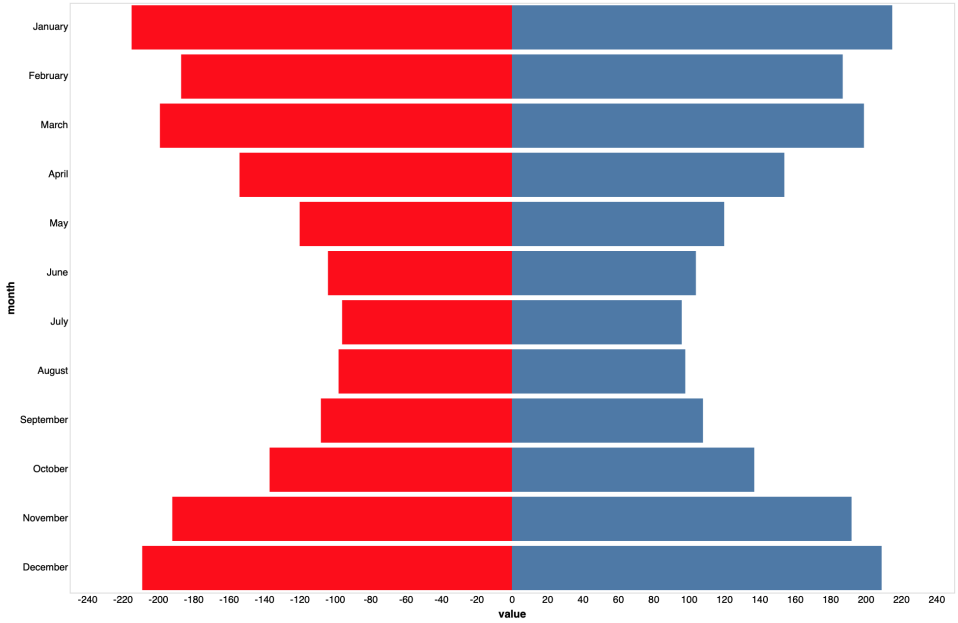


**Figure 5.2:** Vega-Lite approach for paired bar charts.

```
{
  "$schema": "https://vega.github.io/schema/vega-lite/v3.json",
  "description": "A simple bar chart with embedded data.",
  "data": {
    "values": [
    {"group": "Product A","month": "January","value": 215},
    {"group": "Product B","month": "January","value": 175},
    ...
    ]},
  "height": "600",
  "width": "900",
  "layer": [
  {
    "mark": "bar",
    "encoding": {
      "x": {
        "field": "value",
        "type": "quantitative"
      },
      "y": {
        "field": "month",
        "type": "ordinal",
        "sort": {"order": null},
        "axis": { "ticks": false, "domain": false, "grid": false
    }}},
    "transform": [
    {"filter": { "field": "group", "equal": "Product A"}}]},
  {
    "transform": [
    {"calculate": "datum.value * -1", "as": "value"},
    {"filter": { "field": "group", "equal": "Product B"}}
    ],
    "mark": { "type": "bar", "color": "#F00"},
    "encoding": {
      "x": {
        "field": "value",
        "type": "quantitative",
        "axis": { "ticks": false, "domain": false, "grid": false
        }
      },
      "y": {
        "field": "month",
        "type": "ordinal",
        "sort": {
          "order": null
  }}}}]}
```

**Listing 5.1:** Example Vega schema declaration.

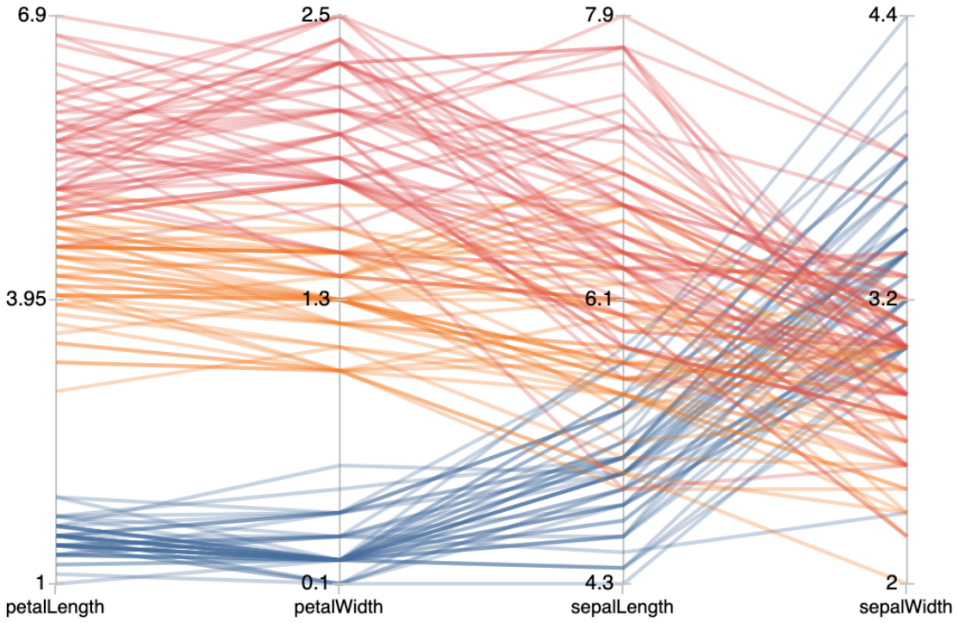


**Figure 5.3:** Vega-Lite approach for parallel coordinates.

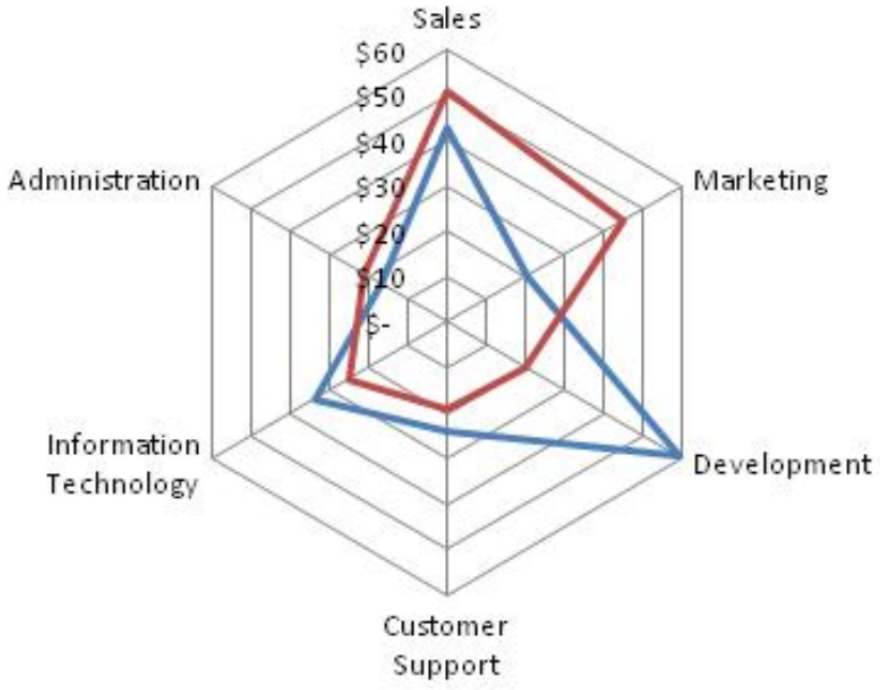


**Figure 5.4:** Example radar chart from NASA (No copyright, no licence)

## 5.3 Radar Charts

Radar charts are similar to parallel coordinates, whereas as the main difference axes, which represents several dimensions are not visualized parallel, but in a circular shape (see example in Figure 5.4). During our project work we figured out, that radar charts are an often requested visualization type in the Vega community. However, those multiple axis in non-cartesian coordinate systems are not possible to be implemented in Vega-Lite, because the Vega-Core is designed for two dimensional charts. As a workaround, absolute positions of axes, data points and their connections may be calculated, but such complex workaround are not the purpose of the grammar of graphics.

## 5.4 Hierarchical Visualizations

Some visualization techniques represent hierarchical data (i.e., tree maps, dendrogram, tree graph...). As mentioned in the previous section, Vega was designed for the cartesian coordinate system. Similar to this limitation, Vega can only handle two-dimensional (or tabular) data structures. As a workaround, hierarchical data must not be stored in a encapsulated way, but with a simple child-parent-reference two-dimensional data structure. However, Vega was not designed to parse such complex data structure and therefore hierarchical visualizations are not possible.

## 5.5 Stacked Bump Chart

Stacked bump charts [Xenographics 2019] are multiple line charts, where the lines are stacked on top of each other. The x-axis shows a time interval, like for examples weeks left till an event and the y-axis shows the number of active entities at a given point on the x-axis.

The initial approach to implement this chart type was to have a data-set which consists of republic candidates and the start and end week when they started their campaign for the election. Next, the data should be extended such that the data-set for every candidate contains every week where their campaign was active. The last step then would be to calculate the active candidates for every week and enumerate them.

Extending the data could be done with a sequence generator, which is capable of creating new data. One can specify a start, stop and step-size value and it will create the specified sequence withing the defined data field. The first problem here is that every candidate then would have to be specified as sequence instead of in the data and the second problem is that it is not possible to add additional information to the generated sequence, like for example the name of the candidate.

For the last step, an identifier transform would be needed, but it is currently not available withing Vega-Lite. The identifier transform takes the data-set and gives every entry a unique id. But also if this would be available, there are some problems with this transformation. In order to make it work for this example, one would need as many transformations as points on the x-axis, each one with a filter for one point.

In order to still create a stacked bump chart, we decided to transform the data beforehand. This was done by modifying the data-set within Excel, such that there is a point for every active candidate, every time there is a change in the number of active candidates. Also, an additional field for the y-axis called "currNum" has been introduced, which decodes the position of each active candidate. This basically means that the data is transformed, such a normal line graph is drawn. A snipped of the resulting data set is shown in Listing 5.2, the specification without data in Listing 5.3 and the resulting graph is depicted in Figure 5.5.

```
{ "name": "Cruz", "week": 86, "currNum": 1 },
{ "name": "Cruz", "week": 27, "currNum": 1 },
{ "name": "Paul", "week": 83, "currNum": 2 },
{ "name": "Paul", "week": 40, "currNum": 2 },
{ "name": "Rubio", "week": 83, "currNum": 3 },
{ "name": "Rubio", "week": 40, "currNum": 3 },
{ "name": "Rubio", "week": 39, "currNum": 2 },
{ "name": "Rubio", "week": 34, "currNum": 2 },
{ "name": "Carson", "week": 80, "currNum": 4 },
{ "name": "Carson", "week": 40, "currNum": 4 },
{ "name": "Carson", "week": 39, "currNum": 3 },
{ "name": "Carson", "week": 36, "currNum": 3 },
```

**Listing 5.2:** Stacked Bump Chart Data.

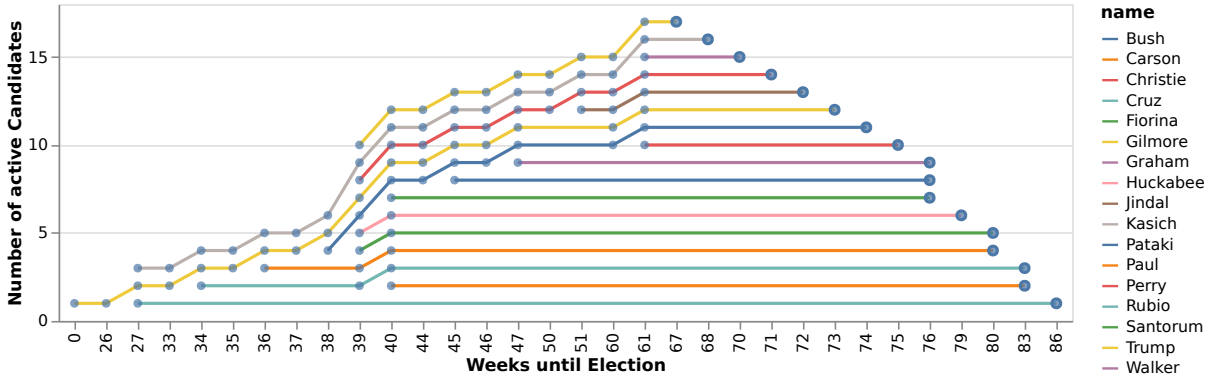


**Figure 5.5:** Stacked Bump Chart Example

## 5.6 Funnel Chart

The last diagram we tried to create in Vega-Lite was the funnel chart. Funnel charts are used to visualize processes and the reduction of entities after each step. An example of a funnel chart is shown in Figure 5.6.

Our approach to implement the funnel chart was to first create a top level trapezoid which size is based on the data of the first step in the process and then to scale the lower ones based on that. The main problem we encountered here was that there is no suitable mark in Vega-Lite to create trapezoids. The only solution at the moment would be to draw the trapezoids with geo-shapes, but this would mean that all coordinates of the trapezoids would have to be calculated beforehand. Therefore we conclude that it is currently not feasible to create charts like this within Vega-Lite.

```
{
  "$schema": "https://vega.github.io/schema/vega-lite/v3.json",
  "description": "Stacked Bump Chart of republican candidate campains 2016 in weeks until election.",
  "layer": [
  {
    "mark": {
      "type": "line"
    },
    "encoding": {
      "x": {
        "field": "week",
        "type": "ordinal"
      },
      "y": {
        "field": "currNum",
        "type": "quantitative"
      },
      "color": {
        "field": "name",
        "type": "nominal"
      }
    }
  },
  {
    "mark": {
      "type": "point",
      "filled": false
    },
    "encoding": {
      "x": {
        "field": "week",
        "type": "ordinal",
        "aggregate": "max"
      },
      "y": {
        "field": "currNum",
        "type": "quantitative"
      }
    }
  },
  {
    "mark": {
      "type": "point",
      "filled": true
    },
    "encoding": {

      "x": {
        "field": "week",
        "type": "ordinal",
        "axis": {
          "title": "Weeks until Election"
        }
      },
      "y": {
        "field": "currNum",
        "type": "quantitative",
        "axis": {
          "title": "Number of active Candidates"
        }
      }
    }
  }
  ]
}
```

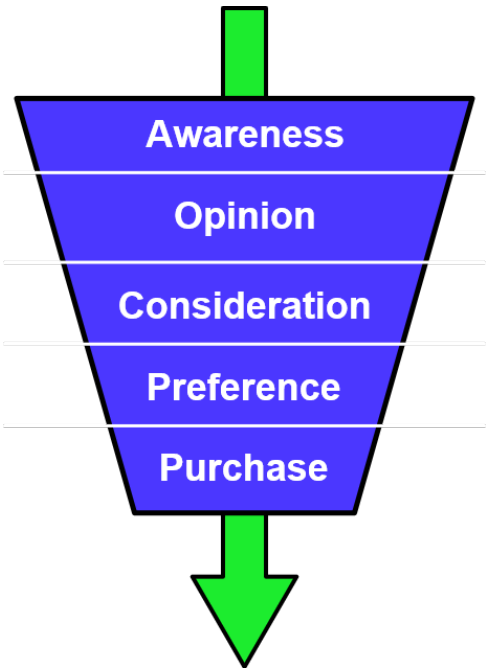


**Figure 5.6:** Funnel chart example. [Image created by Steve simple extracted from https://commons.wikimedia.org/wiki/File:Purchase-funnel-diagram.svg and user under Creative Commons Attribution 3.0 Unported license.]

# Chapter 6

# Conclusion

In our project work we gave an brief introduction on Vega-Lite and the grammar of graphics concept Vega-Lite is based on. Then we showed how the Vega-Lite compiler can be extended by a custom data transformation and what limitations exist, hindering us from realizing new visualization types to Vega-Lite. Based on that, we introduced new visualization types, that are partly or not working in Vega-Lite, to prove and show how those limitations are hindering us of implementing not existing visualizations. We assume that visualization types are missing in Vega-Lite, because it is strongly limited due to wrong design of the whole Vega implementation. That may be reasoned in the small development team of Vega and the small community. We think that in general the grammar of graphics approach is a great concept, and that Vega-Lite is a state-of-the-art system to create visualizations easily. It can take over lots of development work in information visualization from a software development perspective. However, it is not perfect yet, and probably need to be completely rewritten / redesigned to support many more, and also common, visualization types.

# Bibliography


Bostock, Michael, Vadim Ogievetsky, and Jeffrey Heer [2011]. *$D^3$ data-driven documents*. IEEE transactions on visualization and computer graphics 17.12 (2011), pages 2301–2309 (cited on page 1).

Dua, Dheeru and Casey Graff [2017]. *UCI Machine Learning Repository*. 2017. http://archive.ics.uci.edu/ml (cited on page 8).

*Excel Paired Bar Chart Example* [2019]. 26 Jun 2019. http://www.excel-board.com/how-to-create-a-mirror-bar-chart-in-excel (cited on page 9).

*ggplot2* [2019]. 05 May 2019. https://ggplot2.tidyverse.org/ (cited on page 1).

Github [2019a]. *New Transformation Pull Request*. 26 Jun 2019. https://github.com/vega/vega-lite/pull/4958 (cited on page 3).

Github [2019b]. *Vega Count Pattern*. 26 Jun 2019. https://vega.github.io/vega/docs/transforms/countpattern/ (cited on page 4).

Github [2019c]. *Vega Project Transform*. 26 Jun 2019. https://vega.github.io/vega/docs/transforms/project/ (cited on page 5).

Github [2019d]. *Vega-Lite Issues*. 26 Jun 2019. https://github.com/vega/vega-lite/issues (cited on page 3).

Github [2019e]. *Vega-Lite Issues*. 26 Jun 2019. https://github.com/vega/vega-lite/issues/408#issuecomment-350536783 (cited on page 3).

*R-Project* [2019]. 05 May 2019. https://www.r-project.org/about.html (cited on page 1).

SS2019, Group Four - InfoVis [2019]. *GIT Repository for InfoVis Project Work*. 26 Jun 2019. https://github.com/plechi/vega-lite (cited on page 4).

*Vega – A Visualization Grammar* [2019]. 26 Mar 2019. https://vega.github.io/vega/ (cited on page 1).

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).

Xenographics [2019]. *Stacked Bump Chart*. 01 Jul 2019. https://xeno.graphics/stacked-bump-chart/ (cited on page 12).