On Expressiveness and Conciseness of Data Graphics Templates

Romain Vuillemot*
Université de Lyon

THE VISTK TOOLKIT

VISTK is an open-source[1] JavaScript toolkit which allows the creation of standard charts such as scatterplots, bar charts, node-link diagrams and treemaps (see appendix for a more comprehensive list and illustrations). VISTK is intended to non-expert programmers, by easing charts creation using pre-defined templates. They consist in an abstraction layer over D3 [2] that draws charts on web pages using low-level manipulation of SVG graphical elements (rectangles, circles, etc.). Templates provide default configurations, for marks position (scales, etc.) and visual properties (color, fill, etc.), to generate charts. Following D3's recommendations, templates can be customized using Web Standards using CSS for aesthetics and JavaScript for interaction.

Our design focus with VISTK is on the templates options in order to find a sweet spot between **expressiveness** (the number of charts that can be built) and code **conciseness** (the number of line of codes required to draw and customize charts). From our observations, research prototypes such as Protovis [1] and the most popular D3 visualization libraries (e.g NVD3.JS, C3.JS, HIGHCHART.JS) also have a similar approach using templates description which allow charts composition, customization and interaction using options or by adding custom JavaScript code. Vega [4] (VISTK was developed at the same time as Voyageur) does not use templates but exposes charts configuration as an API with parameters following a grammar of graphics. All those approaches (including VISTK) to encapsulate code into APIs (Application Programming Interface) aim at allowing easier access to data or features, are standard in software architecture, but novel when it comes to captures visual design and interaction.

Our approach with VISTK is a bit singular as we also aimed at providing shared building blocks across charts, for configuration consistency and re-usability. The building blocks of VISTK rely on the following assumptions:

1. A dataset is composed of `items`: examples of items are countries, sport teams, stock market companies, etc., and their properties are attributes. In the code snippet below, we show items–which are countries have GDP per capita attributes–that appear as series of transactional data. The complete dataset has about 200 items, and values may vary over time.

```
[{country: USA, gdppc: 53,041.98, year: 2013},
 {country: FRA, gdppc: 42,503.30, year: 2013}
...
]
```

2. A chart is the mapping of `items` to graphical marks: which means that each item will be represented as one primary mark (circle, rectangle, text label, etc.), and its attributes will be mapped to visual properties such as position and color, for instance. Items can results as the composition of multiple marks, such as combining text labels with circles, for instance.

*e-mail: romain.vuillemot@gmail.com

[1] `https://github.com/cid-harvard/vis-toolkit` under a MIT-Licence

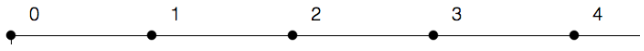

Figure 1: Dot plot chart with default configuration values.

3. A chart template should capture the minimal chart settings: which means a configuration should remain as simple as possible. But in the meantime, should allow flexibility to change graphical marks types and visual properties mapping to attributes. Default values should be informed and should allow the chart to be created despite missing properties. The code snippet below shows the internal configuration of VISTK for a dot plot chart template (shown on Figure 1).

```
vars.default_params["dotplot"] = function(scope){

  var params = {};

  params.x_scale = [{
    func: d3.scale.linear()
  }];

  params.y_scale = [{ }];

  params.items = [{
    marks: [{
      type: "circle",
    }, {
      type: "text",
      rotate: "-90"
    }]
  }];
```

The important part of the code snippet above is the `params.items` object that defines the mapping between how items are represented as graphical marks (here, circle and text). This snipped is an internal JavaScript Object that can be overridden later on for customization using the VISTK API. The API call to create (and customize) the dot plot from the HTML page is as follows (and result in the same dot plot as on Figure 1):

```
var data = [{__id: "A"}, {__id: "B"}, {__id: "C"}...

var visualization = vistk.viz()
    .params({
      dev: true,
      data: data,
      width: 800,
      height: 500,
      type: "dotplot",
      var_x: "__id",
      var_y: function() { return this.height/2; },
      var_text: "__value"
    });

d3.select("#viz").call(visualization);
```

Figure 2: Dotplot chart template with customization.

In the above code snippets, `items` are mapped to circles (which is the mark set by the template). To customize this mapping, and use red rotated rectangles instead of circles, as on figure 2, here are the changes to be made to the API call:

```
items: [{
  attr: "name",
  marks: [{
    type: "tick",
    rotate: "0"
  }, {
    type: "diamond",
    rotate: "0"
  }, {
    type: "text",
    rotate: "0",
    translate: [-10, -30],
    text_anchor: "middle"
  }]
```

All the figures provided as appendix allow to grasp the expressiveness of VISTK. We noticed some chart configurations had to be completely rewritten to reach a certain level of customization. A way to reduce the number of lines of code would be to create classes of charts, such as vertical and horizontal bar charts, and store them in separate templates. However, this approach opens the door to a more complex toolkit architecture with inheritances, and developers will have to browse a longer catalog of charts templates to figure out which one to begin with.

BUILT-IN FUNCTIONS AND APPLICATIONS

VISTK also contains many built-in functions. Most are related to data wrangling (dealing with missing values, re-shaping dataset, ..) but also to make it convenient to aggregate, filter, update over time. Such functions sometimes generate new data. For instance, if countries are aggregated as continents, continents become items that can be mapped to a graphical mark. See figure 9 where continents appear on a chart, using countries-level data for each pie chart.

VISTK is currently used in real-life production websites, such as the *The Colombian Atlas of Economic Complexity*[2], its variations to other countries (Mexico[3], Peru[4]) and a couple of standalone experiments such as onboarding for complex economic concepts[5] or press release[6]. It is integrated as an Ember.js[7] component which code is also open source[8].

LIMITS OF THE VISTK

The main limit of VISTK was the limited size of the community around the toolkit that did not allow particular feedback on its ease of use and flexibility. While we managed to fulfill all our needs, we did not collect any qualitative claims on the toolkit's expressiveness, as it was most built and updated with specific charts in mind.

[2]http://datlascolombia.com/

[3]http://complejidad.datos.gob.mx/

[4]http://growthlab.cid.harvard.edu/peru-atlas

[5]https://cid-harvard.github.io/atlas-exports-quality/colombia.html

[6]http://atlas.cid.harvard.edu/rankings/growth-predictions/

[7]http://emberjs.com/

[8]https://github.com/cid-harvard/atlas-subnational-frontend/

The toolkit being developed by the author of this article[9], all the knowledge and competences in the design process have not been shared or structured in a way to seamlessly pass on the baton.

Another limit is the lack of support for charts templates parameters exploration. Indeed, the process of creating charts is a lot of trial and errors. In a previous work, we have been working on a toolkit with Visual Sedimentation [3], which was configured using quantitative values (pace of updates, pace of aggregation, etc.) which has the main benefits of always resulting in a chart. And unexpected results allowed creative exploration, while with VISTK and other toolkits there is no room for error.

QUESTIONS AND DISCUSSION FOR THE WORKSHOP

As discussed previously discussed in this article, our main focus is on data graphics templates configurations and customization. We are particularly interested in how they can be expressive, while remaining concise. There are other questions we would like to discuss during the workshop:

- **Charts template taxonomy:** as far as we know, there is no official list of chart, despite many attempts. It is crucial to reach a consensus on charts naming, description, and genealogy. This would greatly ease the tools and toolkits comparisons, and make on boarding with new tools faster.
- **Comparison:** we need metrics to compare charts with each others, especially to measure expressiveness and conciseness (e.g. which charts are supported, lines of codes required to customize, default values, ..).
- **From modules to components:** modules are reusable pieces of code; and components are reusable parts of graphics meant to be combined together. There should be more efforts put on the later, while the former is often used as it is related to well-known programming paradigms.
- **Performance:** is key to successful for adoptions by a community of users and developers. VISTK has been heavily tweaked to load, run, animate smoothly when integrated as a component. But a general-purpose toolkit would not be possible to optimize that easily without knowing its end goal.
- **The last mile:** building tool tips, making sure charts are cross-browser compatible, code testing, continuous deployment, user feedback, documentations, etc. are sometimes pure engineering effort, but are the extra layers of polish needed for engaging and professional-looking result.

ACKNOWLEDGEMENTS

The author would like to thank Quinn Lee, Mali Akmanalp, Gus Wezerek and Greg Shapiro who helped improving and integrating VisTK to the Atlas websites and experiments.

REFERENCES

[1] M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. *IEEE transactions on visualization and computer graphics*, 15(6):1121–1128, 2009.

[2] M. Bostock, V. Ogievetsky, and J. Heer. D3 Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, Dec. 2011.

[3] S. Huron, R. Vuillemot, and J.-D. Fekete. Visual sedimentation. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2446–2455, 2013.

[4] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE transactions on visualization and computer graphics*, 22(1):649–658, 2016.

[9]Which means VISTK bus factor is 1 `https://en.wikipedia.org/wiki/Bus_factor`

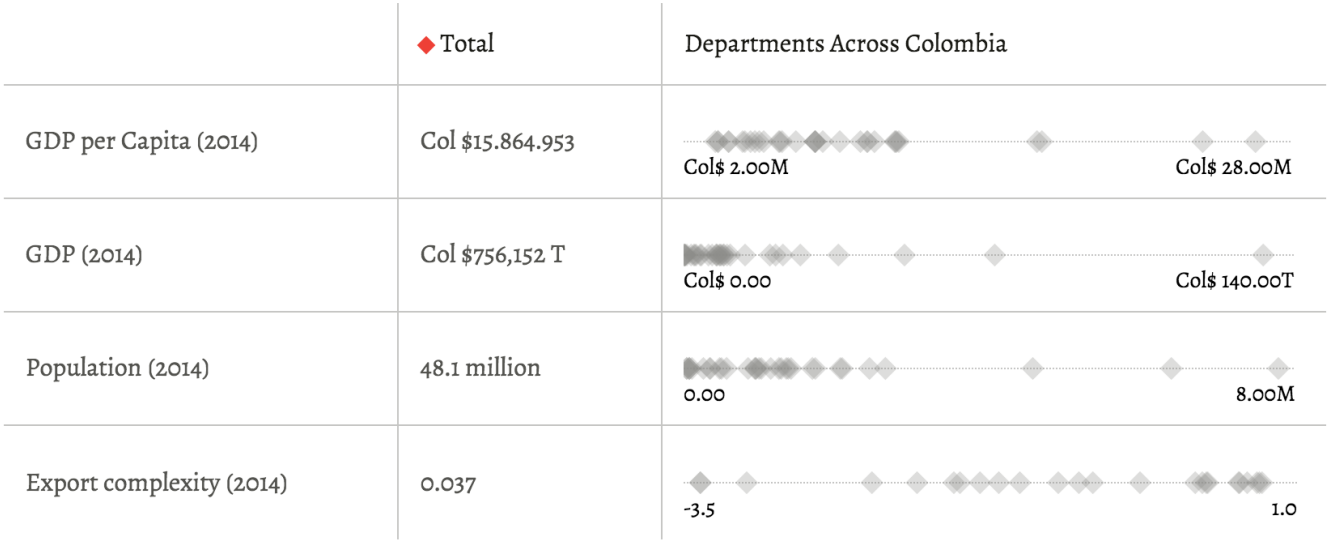

Figure 3: Multiple coordinated dot plots https://github.com/cid-harvard/atlas-subnational-frontend/blob/3b3c10609699bed5e7067d147a7a8a627c980f63/app/components/vistk-dotplot.js

What relatively complex industries have the most possibilities for this city?

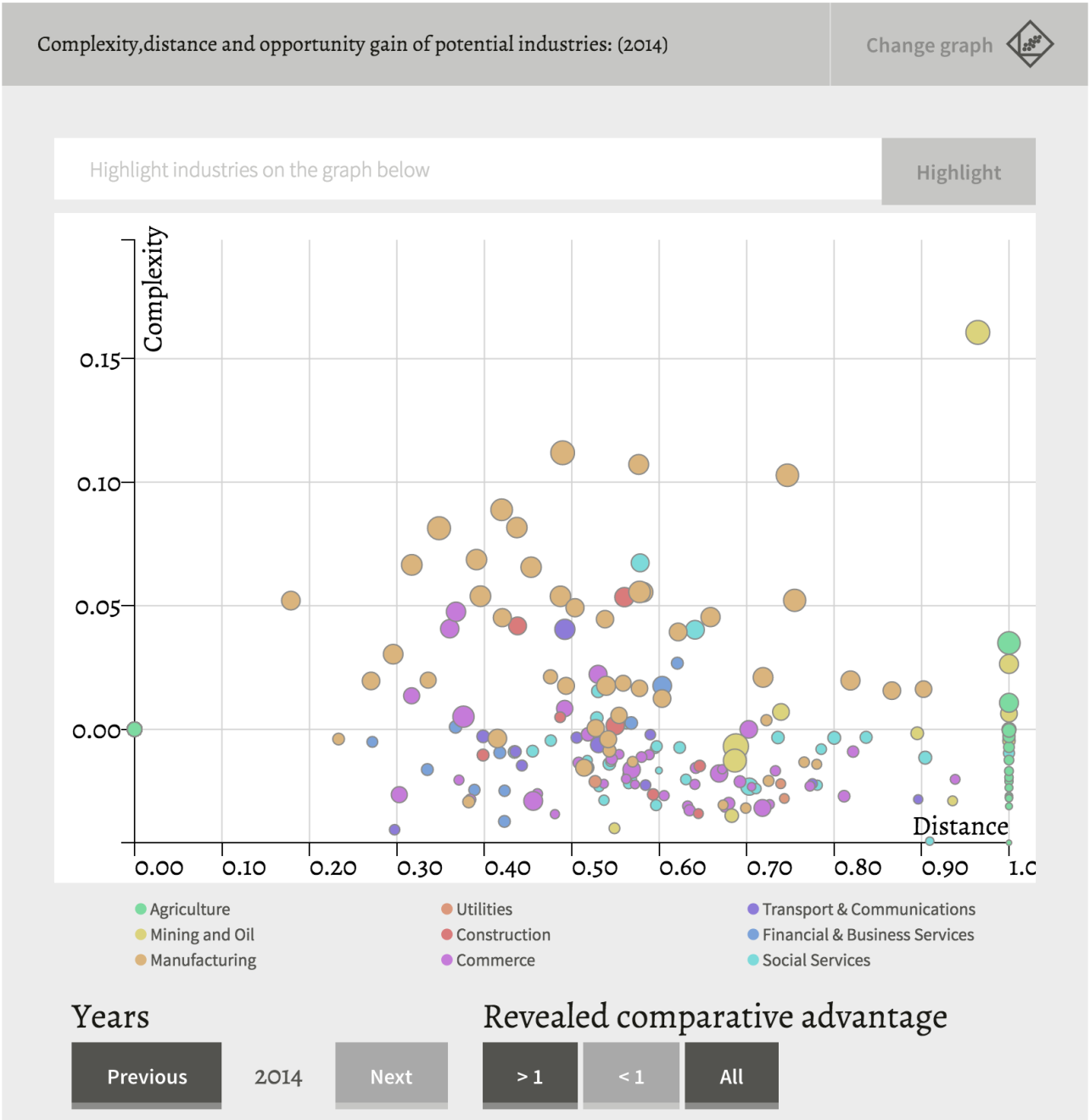

Figure 4: Scatterplot https://github.com/cid-harvard/atlas-subnational-frontend/blob/3b3c10609699bed5e7067d147a7a8a627c980f63/app/components/vistk-scatterplot.js

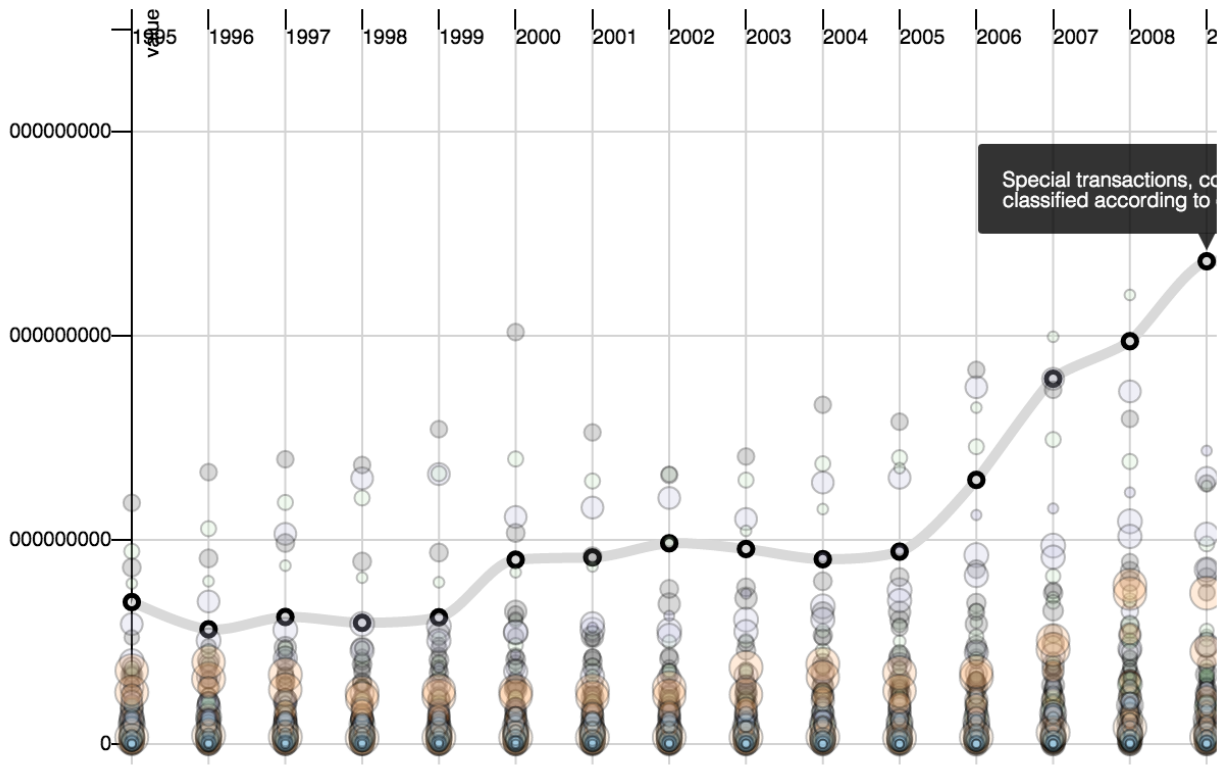

Figure 5: Categorical scatterplot `https://github.com/cid-harvard/vis-toolkit/blob/master/src/visualizations/caterplot_time.js`)

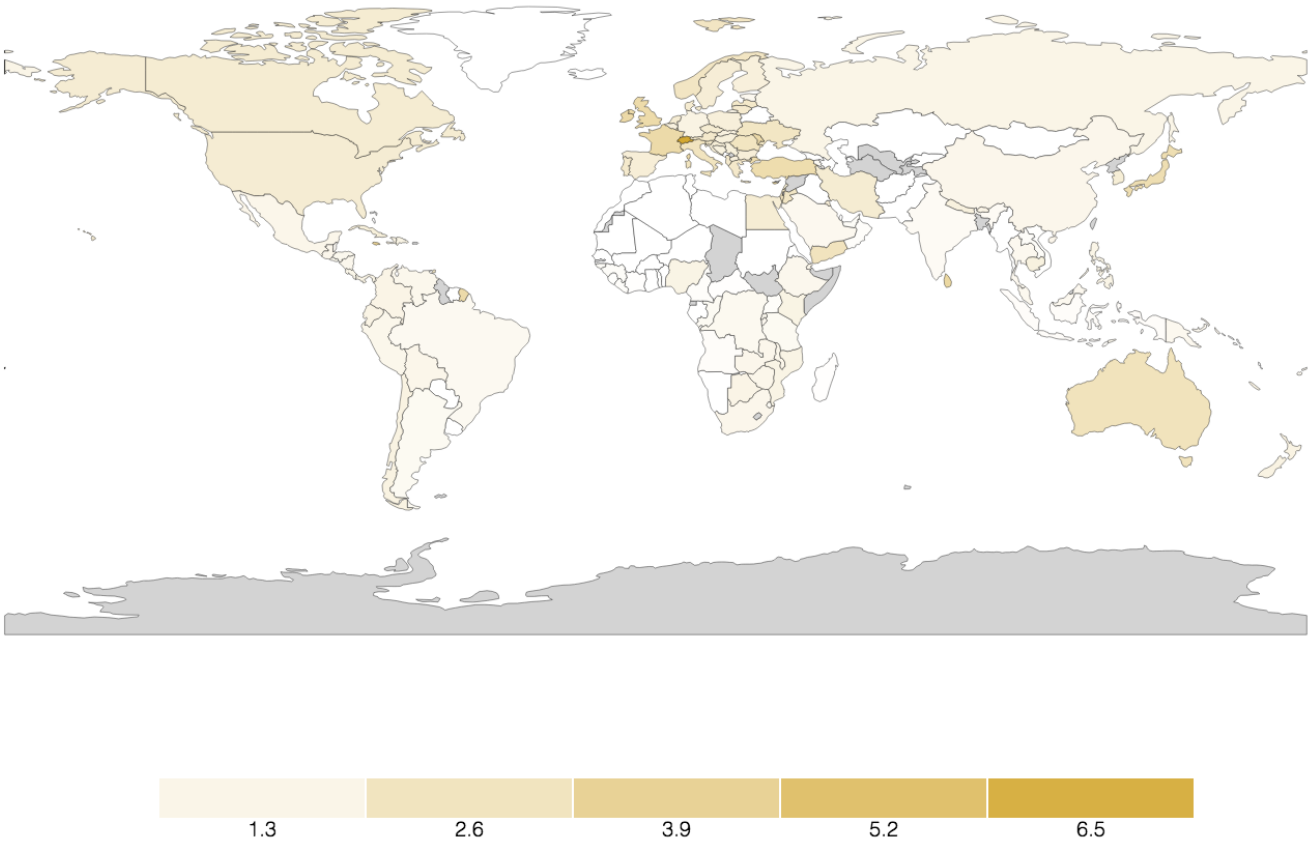

Figure 6: Geo map and legend `https://github.com/cid-harvard/vis-toolkit/blob/master/examples/geomap_legend_color.html`

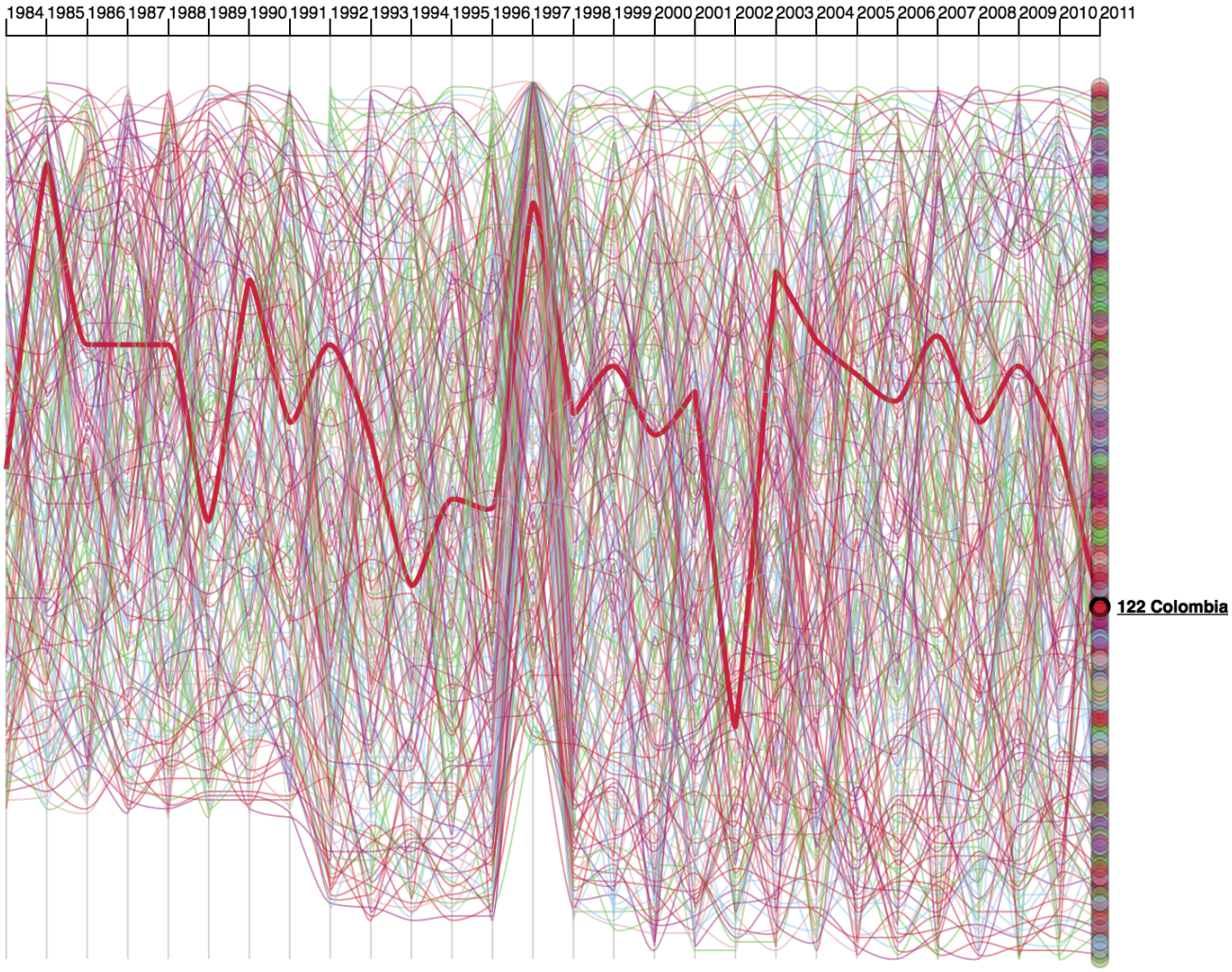

Figure 7: Line chart `https://github.com/cid-harvard/vis-toolkit/blob/master/examples/linechart_eci_rankings.html`

What industries in this city employ the most people?

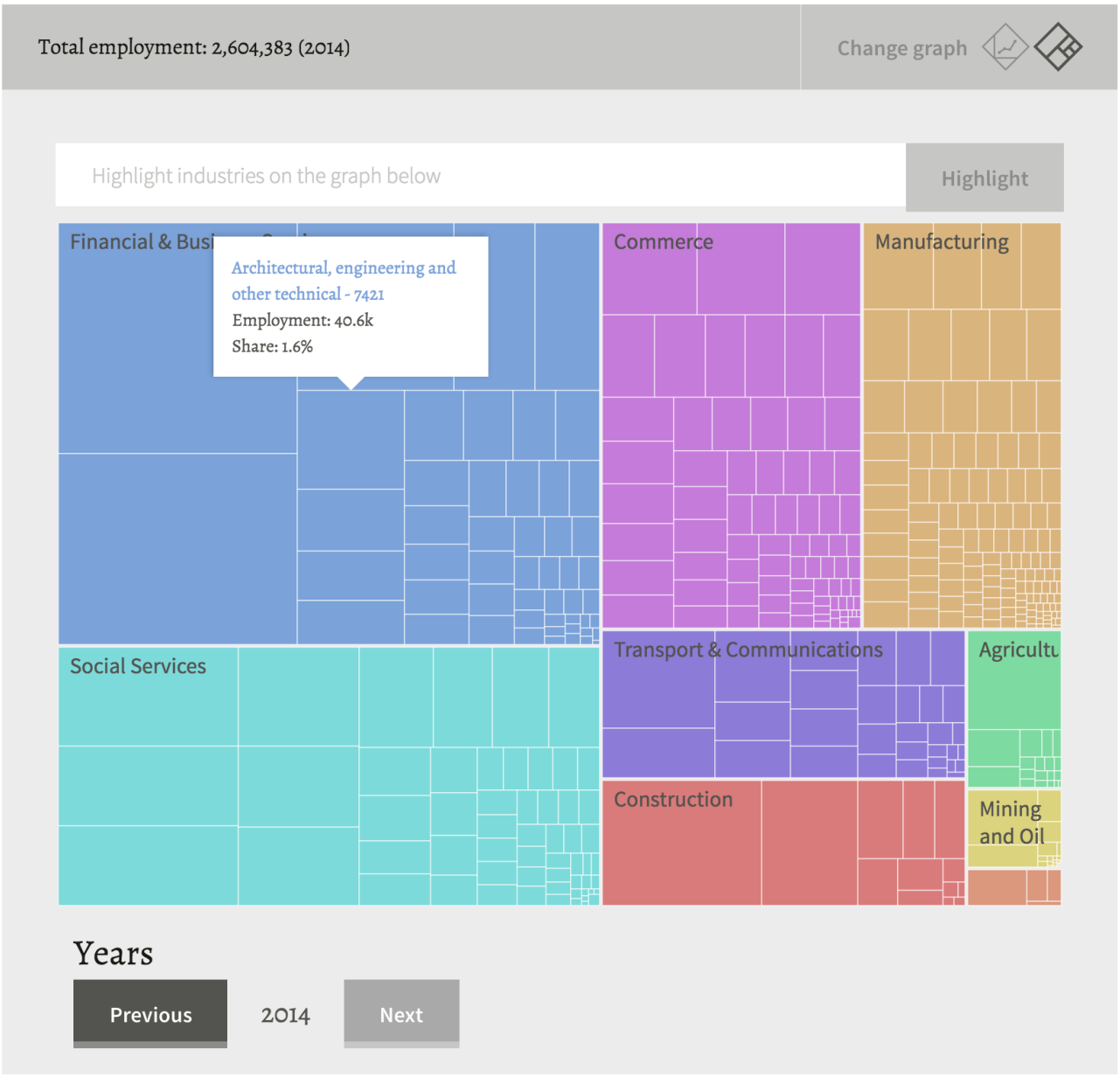

Figure 8: Treemap https://github.com/cid-harvard/atlas-subnational-frontend/blob/3b3c10609699bed5e7067d147a7a8a627c980f63/app/components/vistk-treemap.js

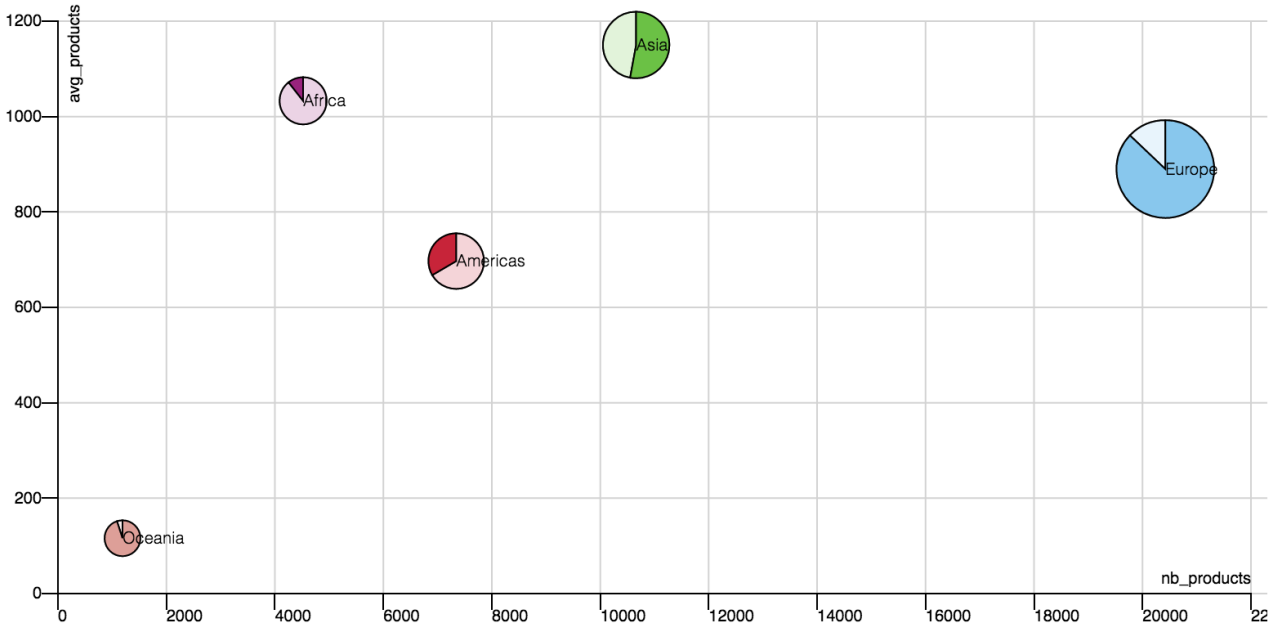

Figure 9: Pie scatter charts. `https://github.com/cid-harvard/vis-toolkit/blob/master/examples/piechart_continent.html`