Information Visualisation - Project Group 2: Responsive Charts: Using and Extending RespVis

Valentin Adler, Ledio Jahaj, Markus Petritz, and Pooja Yeli

706.057 Information Visualisation SS 2021
Graz University of Technology

07 Jul 2021

Abstract

Data visualisations are usually static - they have a fixed layout and size. Making them responsive can help greatly with supporting various devices, and it can also improve the user experience. RespVis is a framework that aims to do that - provide responsive data visualisations. Our aim with this project was to extend RespVis with new types of data visualisations. In this paper we will first give an introduction to RespVis itself and then talk about our work.

Contents

List of Figures

List of Listings

Chapter 1

Introduction

The RespVis framework [Oberrauner 2021] was originally created by Peter Oberrauner. It is written in TypeScript and provides responsive data visualisations rendered as SVG. The layout of the visualisations is handled by a custom layouting system.

The framework works as an extension of D3 [Bostock 2021]. Each visualisation is implemented as a function that renders SVG components with D3. As such it supports existing D3 features, such as different scale types or forms of interaction. The RespVis source code can be found at github.com/AlmostBearded/respvis.

1.1 Layouting

The basic functionality of the layouting system is that the SVG structure (legend, x axis, y axis, chart itself) is recreated with hidden `div` elements. Elements requiring CSS rules for responsive behaviour essentially copy these over to their `div` counterparts. The browser calculates a layout for the mirrored `div` structure, after which RespVis can transfer the same layout back to the SVG structure.

This makes it possible to support a variety of responsive browser features with plain CSS. In particular, it supports CSS flexbox [Coyier 2021] and grid [House 2021] layouts. However, it only really works with inline CSS. There is no support for more complex CSS selectors, which could make some responsive behaviour easier to implement.

The framework tries to fit the chart into the given container. Making a visualisation the makes the container grow in size is not easy. This requires a workaround that might cause issues in some cases. For example, it is possible to use the grid display to have the browser arrange child elements in such a manner that they exceed the boundaries of their container. With having visible overflow, the chart elements can be viewed then. The only issue is that the actual container is not resized, making it necessary to give the chart container a bottom margin equal to the excess height. While this may not be a pretty solution, it does the job for this purpose. This workaround especially comes into play for our implementation of small multiples.

1.2 Code Example

An example of how to render charts with RespVis can be seen in Listing 1.1. It shows the HTML code for our scatterplot matrix example. The chart is rendered inside a `div` element with a certain width and height. With JavaScript the user would first have to transform the dataset with the appropriate function (depending on the chart type). Then div for the layouter has to be inserted into the chart container, where it is then initialized. The chart is inserted as a SVG into the layouter. The data is then assigned with d3's datum function, and then the chart is initialized by calling the appropriate RespVis library function. To ensure responsive behavior on window resize, the data is reassigned on every change. Depending on the chart type different things will happen whenever the data changes.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
  <head>
    <title>RespVis - Scatterplot Matrix</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta charset="UTF-8" />
    <style>
      body {
        background-color: floralwhite;
      }

      #chart {
        width: 100%;
        height: 80vh;
        min-height: 25rem;
      }
    </style>
  </head>
  <body>
    <h1>Scatterplot Matrix</h1>
    <div id="chart">
    </div>
    <script src="./vendor/d3.v6.js"></script>
    <script src="../respvis.js"></script>
    <script type="module">
      import data from './data/scatterplot-matrix.js';
      const root = d3.select('#chart');
      const layouter = root.append('div').call(respVis.layouter);
      const chartDatum = respVis.dataChartPointMatrix({
        datasets: data.datasets,
        radius: 2
      })
      const chart = layouter.append('svg').datum(chartDatum).call(respVis.
          scatterMatrix)
      window.addEventListener('resize', configure);
      configure();

      function configure() {
        chart.datum(chartDatum);
      }
    </script>
  </body>
</html>
```

Listing 1.1: Example HTML code for the scatterplot matrix.

1.3 Visualisation Types

Prior to our involvement with the framework, it supported scatter plot and bar chart (regular, stacked and grouped) visualisations. Examples of the existing visualisations can be seen at respvis.netlify.app.

We have implemented six further visualisation types, which are partially based on the existing types:

- Line Chart
- Multi-Line Chart
- Connected Scatterplot

- Small Multiples Line Chart
- Small Multiples Bar Chart
- Scatterplot Matrix

These new visualisations will be described in the following chapter.

Chapter 2

RespVis Extension

In this chapter we will describe our implementations and show examples.

2.1 Line Chart

The line chart implementation uses the same point placement as the scatterplot. To make it a line chart, the line has to be added. Therefore a path is added in the SVG starting at the point with the lowest x-value up to the point with the highest x-value - meaning the point which is the farthest on the right. This path can be configured with a colour and a thickness.

Optionally, the line can also be smoothed with a Bézier curve. Listing 2.1 shows how the Bézier command is called with the calculated control points. The code was adapted from Romain [2021] to fit into the RespVis project and our implementation. For comparison, Listing 2.2 shows the code to draw simple straight line segments. With all that being plain SVG, it is ultra-sharp in the browser because of the vector nature and thus can also be scaled up indefinitely. The smoothing operation can be configured by the user to apply or not, there are certain datasets where this would not make sense and a straight line is more intuitive.

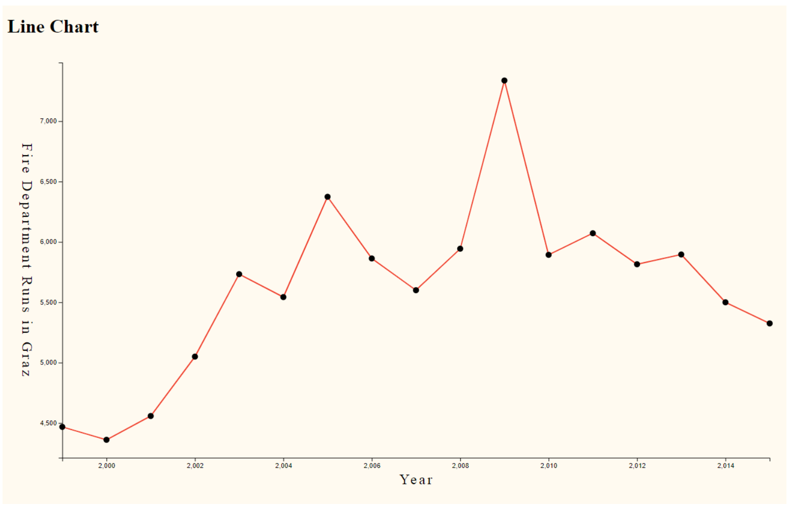
Figure 2.1 shows a simple line chart of call-outs of the Graz Fire Department from 1999 to 2015. The x-axis shows the year and the y-axis the number of call-outs. The dataset is from the City of Graz [Graz 2020] and is freely available under the Creative Commons License. On the right side, one can see how the line chart responds to a smaller width, it converts into a sparkline. The x and y-axis vanish completely, only the first and last values are displayed to the give the user a understanding of the dimensions. The rest is minimal on purpose and may not be suitable for every dataset. However, this is certainly a risk take for the more elegant mobile visualization. Conversion to a sparkline is not implemented as a feature for line charts in RespVis. Instead, it was implemented in the line chart example with regular JavaScript, where we simply manipulate the SVG nodes when the viewport matches a certain width. The responsive behavior and especially the conversion to a sparkline can be observed in the video at `https://youtu.be/SNbEC2gK1ls`.

```
function bezierTangent (a: DataPoint, b: DataPoint) : { length: number, angle:
    number }  {
  const diffX = b.x - a.x;
  const diffY = b.y - a.y;
  return {
    length: Math.sqrt(Math.pow(diffX, 2) + Math.pow(diffY,2)),
    angle: Math.atan2(diffY, diffX)
  }
}

function bezierControlPoint (current: DataPoint, previous: DataPoint|undefined, next
    : DataPoint|undefined, reverse?: Boolean) : {x:number, y:number} {
  const p = previous || current;
  const n = next || current;
  const tangent =  bezierTangent(p, n);
  const smoothing = 0.15;
  const angle = tangent.angle + (reverse ? Math.PI : 0);
  const length = tangent.length * smoothing;
  const x = current.x + Math.cos(angle) * length;
  const y = current.y + Math.sin(angle) * length;
  return {x, y};
}

function bezierCommand (point: DataPoint, i: number, a: DataPoint[]): string {
  const cps = bezierControlPoint(a[i - 1], a[i - 2], point);
  const cpe = bezierControlPoint(point, a[i - 1], a[i + 1], true);
  return `C ${cps.x},${cps.y} ${cpe.x},${cpe.y} ${point.x},${point.y}`;
}
```

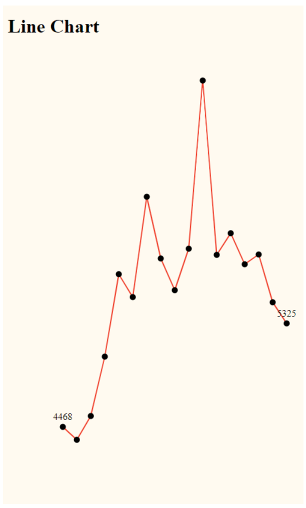
Listing 2.1: Typescript implementation of the Bézier curve adapted from Romain 2021.

```
function lineCommand (point: DataPoint) {
  return `L ${point.x},${point.y}`;
}
```

Listing 2.2: Command for drawing a line in svg.

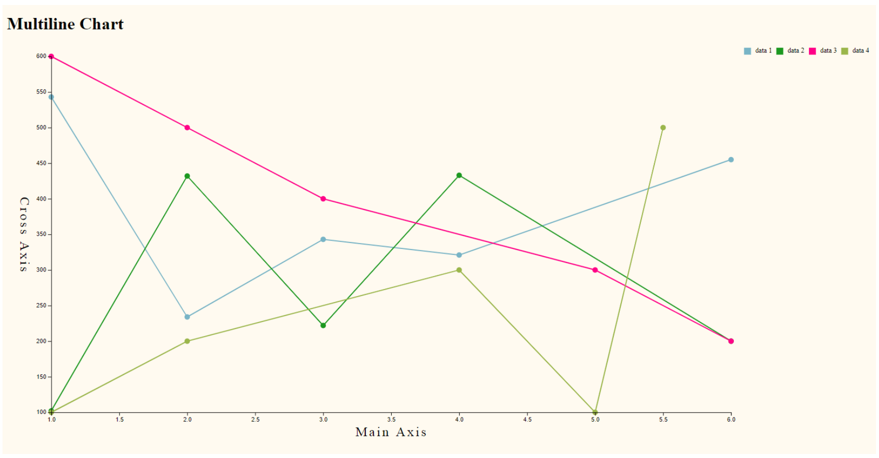


(a) Line chart in a wide window.

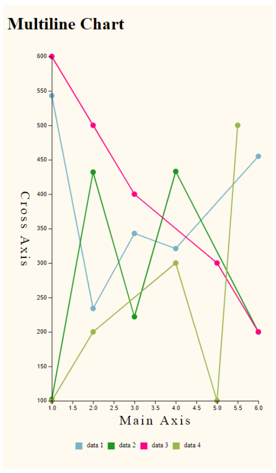


(b) Line chart in a narrow window.

Figure 2.1: An example of a line chart implemented with the RespVis library.



(a) Multi-line chart in a wide window.



(b) Multi-line chart in a narrow window.

Figure 2.2: An example of a multi-line chart implemented with the RespVis library.

2.2 Multi-Line Chart

The multi-line chart is similar to the line chart, but displays more than one line. The dataset therefore should have its data points in the same range to avoid any large empty areas. Currently, the user cannot customise the lines in this visualisation, the colours and thickness are hardcoded. The colours are chosen from a set of distinct categorical colours, which should make it easier to distinguish between them. It would however be fairly simple to make it possible for users to customise individual colours. The sparkline reduction is also not implemented. The multi-line chart example is also shown in the video at `https://youtu.be/y8B8KB6a7qc`.

2.3 Connected Scatterplot

The connected scatterplot implementation is based on the existing regular scatterplot in RespVis and our implementation of a line chart. All points are connected by a path in the order they appear in the dataset. This require the user to order the dataset apppropriately, so that the line is correctly rendered. Additionally, each point is labelled with the data from a third dimension of the dataset, where the first and the second dimensions are the x and y-axis.

In the example in Figure 2.3, the dimensions are:

- Cost per gallon
- Miles per person per year
- Year

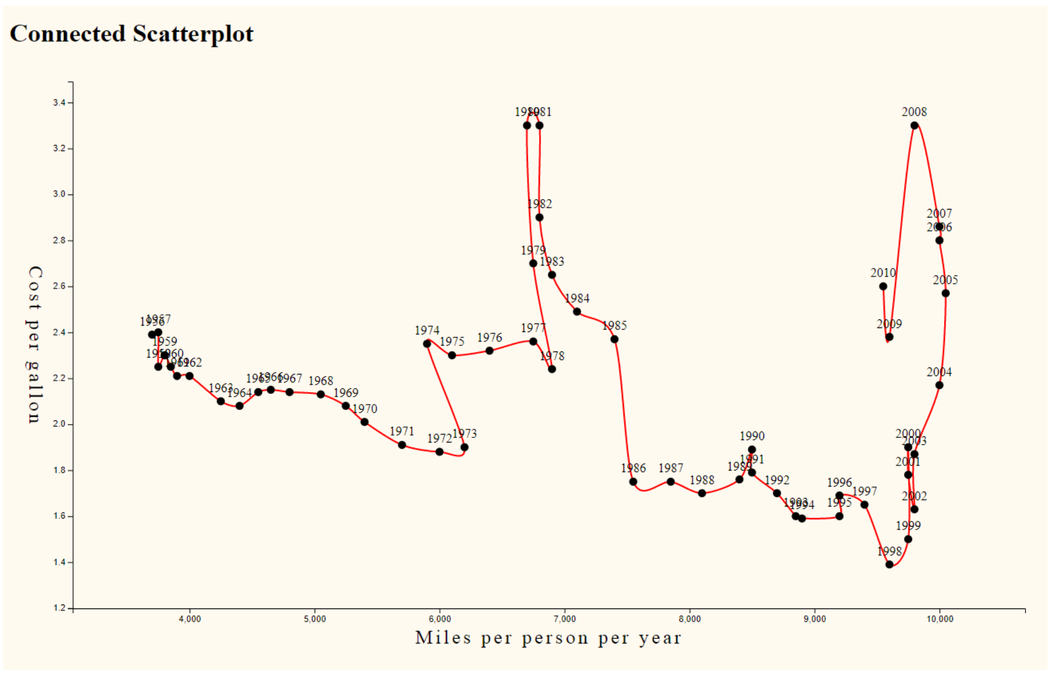
This dataset is based on an example from Mike Bostock [Bostock 2018]. Cost per gallon is rendered in the y-axis, miles per person per year is rendered in the x-axis, and year is rendered in the labels of the points. Figure 2.3a is fully zoomed out, whereas Figure 2.3b has been zoomed in somewhat.

The line is a smooth cubic Bézier curve, which is currently hardcoded. However, with minor code changes, it could easily be changed to a regular straight line, and the same goes for the line colour or width.

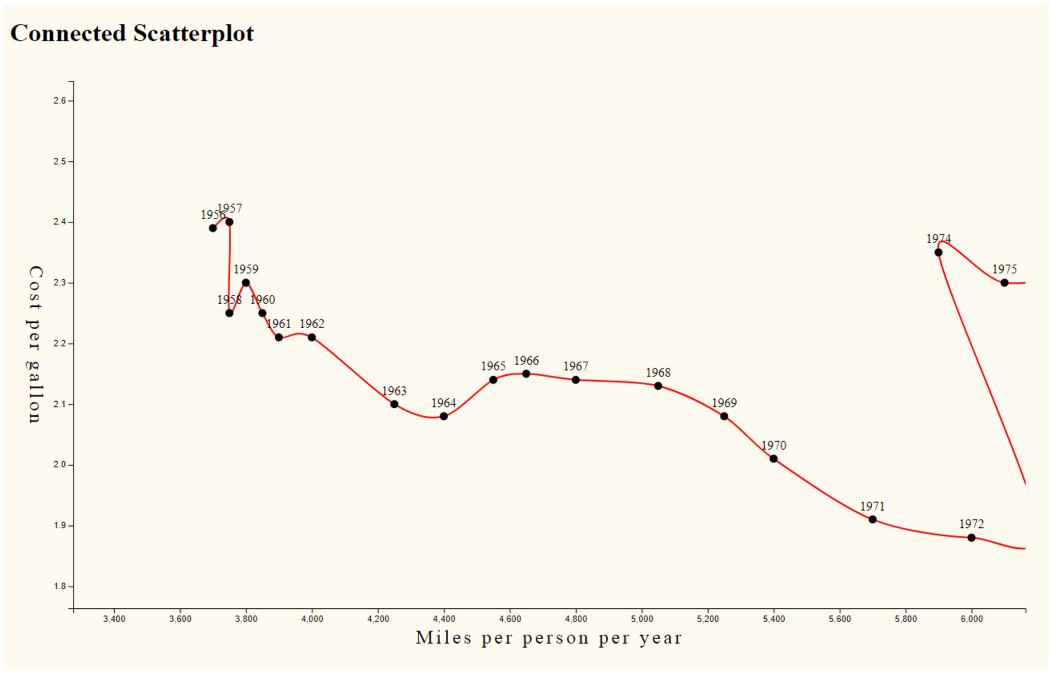
The following responsive patterns can be observed with the connected scatterplot:

- Removing Axis Ticks: axis ticks disappear when the chart container becomes narrower.
- Interactive Zoom: the user can zoom into the chart and move around.

The user can also hover the mouse pointer above one of the points to highlight it. On a mobile device, the user can simply touch a point to get it highlighted. The connected scatterplot is demonstrated in the video available at `https://youtu.be/wKbzVW354As`.



(a) Connected scatterplot in full size.



(b) Connected scatterplot zoomed in.

Figure 2.3: An example of a connected scatterplot implemented with RespVis library.

2.4 Small Multiples

A *small multiples* chart is a series of related charts placed side-by-side or in a grid for comparison. Each chart should generally use the same scale and axes for better comparison. The series of visualisations help the user compare different datasets or sometimes different parts of a dataset. The term was popularised by Edward Tufte. We implemented three small multiples charts for RespVis:

- Line Chart
- Bar Chart
- Scatterplot Matrix

2.4.1 Line and Bar Chart

For the bar and line chart variants there are three relevant dimensions: outer, cross, and main. The outer dimension specifies the number of charts. The main dimension is the horizontal axis and is the same for all charts. The cross dimension is also the same for all charts, but is a linear scale based on the minimum and maximum values of the data.

For the grid layout, we first compute the number of rows and columns. This is done by taking the square root of the number of charts to render and rounding it down. Additionally we have a base column width of 400 pixels defined. This width is multiplied by the device pixel ratio (to take e.g. zoomed-in browsers into consideration) and then multiplied by the number of computed columns. If the resulting total width exceeds the width of the chart container, then we simply divide the chart container's width by the scaled column width, which will give us the number of columns we could fit into the container. Now that we know how many columns and rows to render, we can define the grid layout for the CSS grid template, which is used on a container element inside the chart. The individual sub-charts are then inserted into this container element. The browser will automatically position these child elements inside the grid. As mentioned previously, this requires a workaround to make the chart resize its container. Especially when there is only one column, the number of rows will likely exceed the boundaries of the chart's container. With visible overflow, the sub-charts will still be visible, but it is also necessary to give the chart a bottom margin, so that the container pushes anything beneath it further down. Otherwise, the chart elements might be visible, but they would overlap with other browser elements.

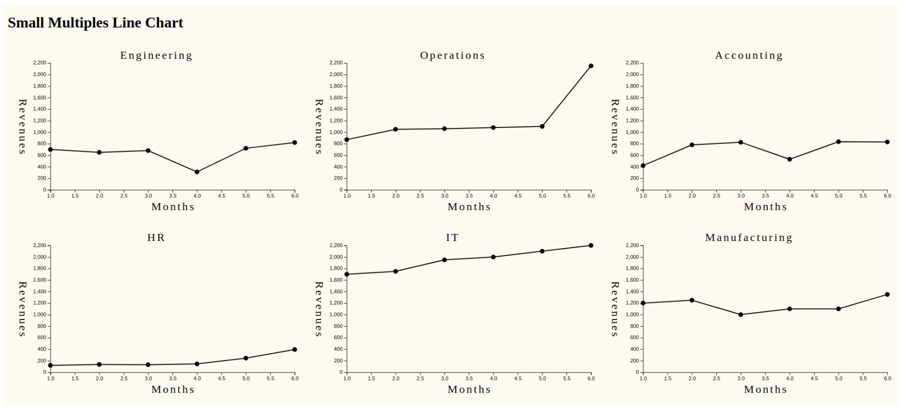
The main responsiveness of this chart type is the adjustment of the grid template. With small code adjustments it might also be possible to flip the charts by 90 degrees, but that is currently not implemented.

2.4.1.1 Line Chart

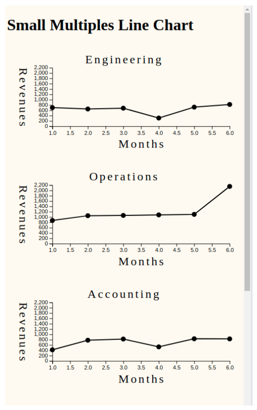
A small multiples line chart is a series of line related charts. An example of a small multiples line chart series is shown in Figure 2.4. The example shows 6 line charts in a grid, thus the outer dimension is 6. The main dimension is linear for all line charts and cross dimension is a linear scale over all data points. Figure 2.4a shows the example on a wide window with three columns and two rows. On a narrow window, the charts line up in a single column, as shown in Figure 2.4b and can be scrolled through. A video showcasing this behaviour can be seen at `https://youtu.be/Q6exaWs3fBg`.

2.4.1.2 Bar Chart

Figure 2.5 shows an example of a small multiples bar chart implemented with RespVis library. It contains a series of six bar chart visualizations. The main dimension is band for all bar charts and the cross dimension is a linear scale over all data points. Figure 2.5a shows the example on wide window and Figure 2.5b shows the same example on a narrow window. The three-by-two grid is adpated to become a single column at narrower widths, with vertical scrolling if necessary. A video showcasing this behavior can be seen at `https://youtu.be/7sA_--fC72Y`.

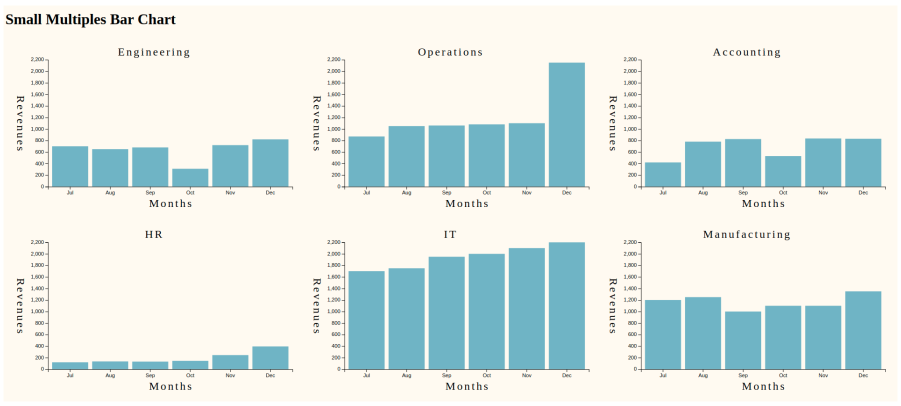


(a) Small multiples line charts in a wide window.

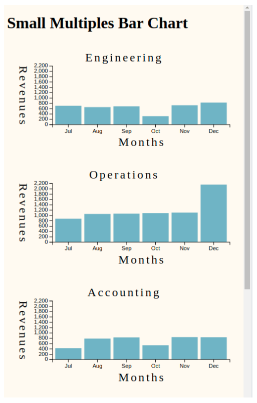


(b) Small multiples line charts in a narrow window.

Figure 2.4: An example of responsive small multiples line charts implemented with RespVis the library.



(a) Small multiples bar charts in a wide window.



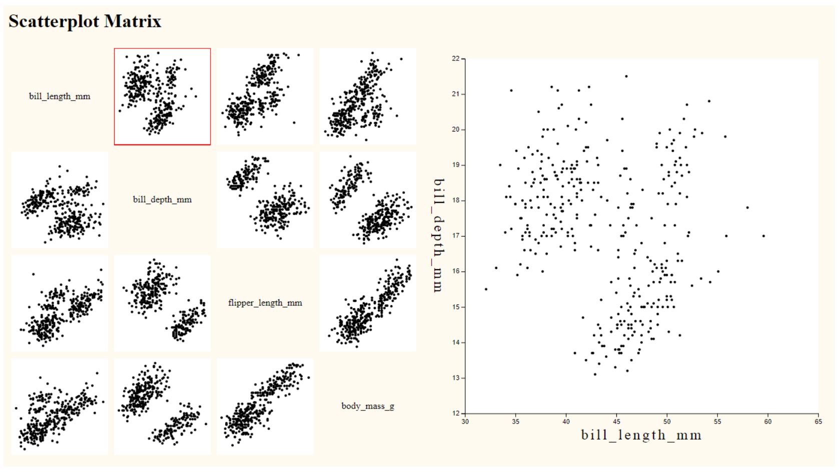
(b) Small multiples bar charts in a narrow window.

Figure 2.5: An example of responsive small multiples bar charts implemented with the RespVis library.

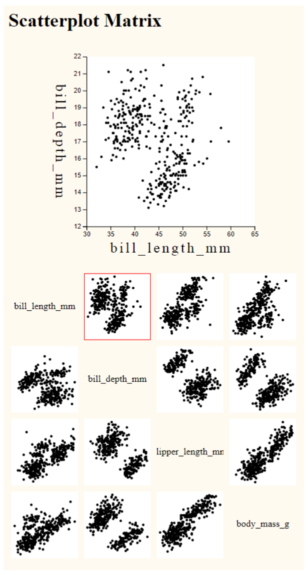
2.4.2 Scatterplot Matrix

Our scatterplot matrix implementation is based on our small multiples implementation. A scatterplot matrix is a square matrix of scatterplots of every pair of dimensions in a multidimensional dataset. For example, a scatterplot matrix of seven dimensions would be rendered as a seven by seven square grid of scatterplots. The titles of the dimensions are shown along the diagonal.

In addition to the matrix, we implemented a focus view on the right side of the container. When a user clicks on a particular scatterplot in the matrix, it is highlighted in red, and is rendered in the focus view in more detail. In order to implement this, we created two containers, one for the matrix and one for the focus view. To create the matrix we iterate through all the dimensions of the data and for each pair of dimensions, we create a scatterplot. Unlike the other small multiples implementations, we do not compute a grid template based on the container width. The template is always a square grid with as many rows and columns as there are dimensions in the dataset. The scatterplots in the matrix do not show their



(a) Scatterplot matrix in a wide window.



(b) Scatterplot matrix in a narrow window.

Figure 2.6: An example of a responsive scatterplot matrix implemented with the RespVis library.

axes and their axes ticks. This is done for the sake of simplicity of the matrix and for the purpose that in order to see more details of a particular scatterplot, the user clicks it and it renders in the focus view. For the focus view functionality, we iterate through all the nodes of the columns and when the node is clicked, we render its data to the focus view matrix. The currently highlighted chart is also given a red border in the matrix view.

The responsiveness of the scatterplot matrix is rescaling of the components and rearrangement of the focus and matrix views depending on the container width. The breakpoint is currently defined as 960 pixels, which is again scaled with the device pixel ratio. Above the breakpoint, the matrix is rendered to the left and the focus view to the right, but as soon as the container width goes beneath the breakpoint, the focus view shifts to the upper half of the container and the matrix to the bottom. It is necessary for the focus view to be above the matrix, as the matrix might grow bigger than the container boundaries. Just like with the other small multiple implementations, we give the chart container a bottom margin, so that it doesn't seem like the charts exceed the container. There is however a limitation to this behavior: if the chart container is too small in both width and height, then focus and grid view might overlap, as all the scatterplots have a fixed aspect ratio to force them into square shape. Unfortunately, this means that they might have a larger height than what would fit into their respective container.

Figure 2.6 shows an example of our implementation, depicting what it could look like both in a wide and a narrow window respectively. The dataset is taken from Waskom 2014, which originates from Horst 2020. The behaviour can also be observed in the video at `https://youtu.be/XAnfRDW2Rso`.

Chapter 3

Future Work

This chapter provides some ideas for potential future developers. They describe necessary changes to make our implemented visualisations more responsive and thus more compatible with the overall theme of the RespVis framework.

3.1 Line Chart

More customisation options could be added. Additionally, the way these options are configured, as part of the dataset, could be improved upon. This could be something that needs to be considered for the entire framework. Additionally the conversion to a sparkline is not implemented as a feature of the line chart component, but is done with regular JavaScript in the line chart example. If it was to be implemented as a standard feature of the line chart, then one would also have to make it optional, so that the line chart does not always convert to a sparkline.

3.2 Multi-Line Chart

Currently the multi-line chart implementation does not convert to a sparkline once the width is made smaller to a mobile phone. This could be done in a similar fashion to the line chart example, but it might not work with every type of data. The colours for the individual lines are currently picked from a set of distinct categorial colors. The user could replace them with JavaScript by manipulating the SVG nodes, but it would be better if it was possible to assign colours to the individual lines when configuring the chart.

3.3 Connected Scatterplot

For the line chart, it is already possible for the user to specify line width and color, and whether to have a smooth curve or not, but that is not yet customisable for the connected scatterplot.

3.4 Small Multiples

For all small multiples examples, so-called brushing could be implemented. Whenever the mouse points over certain data points in one of the views, the same data points light up in the other views. One can also brush over multiple data points and have the same effect. The idea is that the user gets a better understanding of the visualized data by quickly examining the selected data in the multiples views.

Another area that needs improvement is how the grid elements are rendered. As mentioned previously, this is done with a workaround to ensure that the charts make their container grow. While it usually works, it is not a clean solution, so maybe there could be a better way to do that. Especially for the scatterplot matrix a different approach could be better. If the container of the chart becomes too small, then both the focus view and the grid might overlap, as all scatterplots are forced to be in a square shape, which causes them to potentially overflow.

Chapter 4

Concluding Remarks

The framework was pleasant work with and make extensions to. Our implementations however are not perfect - there are still possible improvements, especially when it comes to customization. Especially when it comes to customization of charts the framework could use some improvement. Whenever we made it possible for the user to customize the styling of a chart component, we did that as variables of the chart dataset, which is alternatively done by manipulating the SVG directly. This could be done for other charts as well, which might be more accessible to users.

Bibliography

Bostock, Mike [2018]. *Connected Scatterplot*. 09 May 2018. `https://observablehq.com/@d3/connected-scatterplot` (cited on page 8).

Bostock, Mike [2021]. *D3 Data-Driven Documents*. `https://d3js.org/` (cited on page 1).

Coyier, Chris [2021]. *A Complete Guide to Flexbox*. 01 Jul 2021. `https://css-tricks.com/snippets/css/a-guide-to-flexbox/` (cited on page 1).

Graz [2020]. *Fire Department Call-Outs from 1999 to 2015*. City of Graz, 17 Mar 2020. `https://data.gv.at/katalog/dataset/9e96d656-225d-4129-85cc-2279a7adf0e7` (cited on page 5).

Horst, Allison [2020]. *Palmer Penguins*. 10 Jun 2020. `https://github.com/allisonhorst/palmerpenguins` (cited on page 12).

House, Chris [2021]. *A Complete Guide to Grid*. 12 May 2021. `https://css-tricks.com/snippets/css/complete-guide-grid/` (cited on page 1).

Oberrauner, Peter [2021]. *RespVis*. `https://github.com/AlmostBearded/respvis` (cited on page 1).

Romain, François [2021]. *Smooth a Svg path with cubic bezier curves*. 05 Jul 2021. `https://francoisromain.medium.com/smooth-a-svg-path-with-cubic-bezier-curves-e37b49d46c74` (cited on pages 5–6).

Waskom, Michael [2014]. *seaborn-data*. 24 Feb 2014. `https://github.com/mwaskom/seaborn-data` (cited on page 12).