Fluent Graphics

Adam Bartonicek

Contents

Chapter 1

Abstract

Interactive data visualization has become a staple of modern data presentation. Yet, despite its growing popularity, there still exists many unresolved issues which make the process of producing rich interactive data visualizations difficult. Chief among these is the problem of data pipelines: how do we design a framework for turning raw data into summary statistics that can then be visualized, efficiently, on demand, and in a visually coherent way? Despite seeming like a straightforward task, there are in fact many subtle problems that arise when designing such a pipeline, and some of these may require a dramatic shift in perspective. In this thesis, I argue that, in order to design coherent generic interactive data visualization systems, we need to ground our thinking in concepts from some fairly abstract areas of mathematics including category theory and abstract algebra. By leveraging these algebraic concepts, we may be able to build more flexible and expressive interactive data visualization systems.

Chapter 2

Introduction

> It's written here: 'In the Beginning was the Word!' Here I stick already! Who can help me? It's absurd, [...] The Spirit helps me! I have it now, intact. And firmly write: 'In the Beginning was the Act!
>
> Faust, Part I, Johann Wolfgang von Goethe ([1808] 2015)

Humans are intensely visual creatures. About 20-30% of our brain is involved in visual processing (Van Essen 2003; Sheth and Young 2016), utilizing a highly sophisticated and powerful visual processing pipeline (see e.g. Goebel, Muckli, and Kim 2004; Knudsen 2020; for a brief review, see Ware 2019). It is well-established the brain can process certain salient visual stimuli in sub-20-millisecond times, outside of conscious attention (LeDoux 2000, 2003), and that people can make accurate, parallel, and extremely rapid visual judgements, in phenomena known as subitizing and pre-attentive processing (Mandler and Shebo 1982; Treisman 1985). These features make the visual cortex the most powerful information channel that humans possess, both in terms of bandwidth and throughput.

Statisticians have known about this power of visual presentation for a long time. Starting with early charts and maps, data visualization co-evolved alongside mathematical statistics, offering an alternative and complementary perspective (for a review, see Friendly 2006 or Section 3.1). While mathematical statistics tended to focus on confirmatory hypothesis testing, data visualization provided avenues for unsupervised exploration, "forcing us to notice that which we would never expect to see" (Tukey et al. 1977). Eventually, this valuable role of forcing us to see the unexpected established data visualization as a respected tool within the applied statistician's toolkit.

Seeing an object from a distance is one thing, but being able to also touch, manipulate, and probe it is another. Within the human brain, action and

perception are not independent, but are instead intricately linked, mutually reinforcing processes (see e.g. Dijkerman and De Haan 2007; Lederman and Klatzky 2009). Beginning in the 1970's, statisticians acquired a new set of tools for exploiting this connection. The advent of computer graphics and interactive data visualization transformed the idea of "interrogating a chart" from a mere turn of phrase into tangible reality. All of a sudden, it became possible to work with the visual representation of data in a tactile way, getting new perspectives and insights at the stroke of a key or click of a button.

This compelling union of the visual and the tactile has made interactive data visualization a popular method of presenting data. Nowadays, there are many packages and libraries for building interactive data visualizations across all the major data analytic languages. Interactive figures make frequent appearance in online news articles and commercial dashboards. However, despite this apparent popularity, significant gaps remain in the use and understanding of interactive visualizations. Individual analysts rarely utilize interactive data visualization tools (see e.g. Batch and Elmqvist 2017), the availability of certain more sophisticated interactive features is fairly limited (see Section 3), and researchers still point to a lack of a general interactive data visualization pipeline (Wickham et al. 2009; Vanderplas, Cook, and Hofmann 2020).

This thesis explores these interactive data visualization paradoxes and the inherent challenges surrounding interactive data visualization pipelines more specifically. I argue that, contrary to some prevailing views, interactivity is not simply an add-on to static graphics. Instead, interactive visualizations must be designed with interactivity as a primary consideration. Furthermore, I contend that certain interactive features fundamentally influence the types of visualizations that can be effectively presented. My claim is that popular types of interactive visualizations exhibit a particular kind congruence between graphics, statistics, and interaction, and that the absence of this congruence results in suboptimal visualizations. I formalize this congruence using the framework of category theory. Finally, I validate these theoretical concepts by developing an open-source interactive data visualization library and demonstrate its application to real-world data.

2.0.0.1 Thesis Overview

The thesis is organized as follows. Section 3 reviews the history of interactive data visualization and discusses general trends and issues in the field. Section **??**, focuses on specific problems encountered when designing an interactive data visualization pipeline. Section 4, outlines the the goals and aims that guided the development of the interactive data visualization library. Section 6 details the system's components and design considerations. Section 7, presents an applied example of exploring a real-world data set using the developed library. Finally, Section 8, discusses lessons learned and potential future research directions.

Chapter 3

Background

3.1 Brief history of interactive data visualization

Data visualization has a rich and intricate history, and a comprehensive treatment is beyond the scope of the present thesis. Nevertheless, in this section, I will provide a brief overview, with a particular focus on the later developments related to interactive visualization. For a more detailed historical account, readers should refer to Beniger and Robyn (1978), Dix and Ellis (1998), Friendly (2006), Friendly and Wainer (2021), or Young, Valero-Mora, and Friendly (2011).

3.1.1 Static data visualization: From ancient times to the space age

The idea of graphically representing abstract information is very old. As one concrete example, a clay tablet recording a land survey during the Old Babylonian period (approximately 1900-1600 BCE) has recently been identified as the earliest visual depiction of the Pythagorean theorem (Mansfield 2020). Other examples of early abstract visualizations include maps of geographic regions and the night sky, and these were also the first to introduce the idea of a system of coordinates (Beniger and Robyn 1978; Friendly and Wainer 2021).

For a long time, coordinate systems remained tied to geography and maps. However, with the arrival of the early modern age, this was about to change. In the 16-17th century, the works of the 9th century algebraist Al-Khwarizmi percolated into Europe, and with them the idea of representing unknown quantities by variables (Kvasz 2006). This idea culminated with Descartes, who introduced the concept of visualizing algebraic relationships as objects in a 2D plane, forging a powerful link between Euclidean geometry and algebra (Friendly

Figure 3.1: Photos of the tablet Si. 427 which has recently been identified as the earliest depiction of the Pythagorean theorem [@mansfield2020]. Left: the obverse of the tablet depicts a diagram of a field, inscribed with areas. Right: the reverse of the tablet contains a table of numbers, corresponding to the calculation of the areas. Source: Wikimedia Commons [@mansfield2024].

and Wainer 2021). Coordinate systems were thus freed of their connection to geography, and the x- and y-axes could now be used to represent an arbitrary "space" spanned by two variables.

Descartes' invention of drawing abstract relationships as objects in a 2D plane was initially only used to plot mathematical functions. However, it would not be long until people realized that observations of the real world could be visualized as well. A true pioneer in this arena was William Playfair, who popularized visualization as a way of presenting socioeconomic data and invented many types of plots still in use today, such as the barplot, lineplot, and pie chart (Friendly and Wainer 2021). Further, with the emergence of modern nation states in the 19th century, the collection of data and *statistics* ("things of the state," Online Etymology Dictionary 2024) became widespread, leading to a "golden age" of statistical graphics (Beniger and Robyn 1978; Friendly and Wainer 2021; Young, Valero-Mora, and Friendly 2011). This period saw the emergence of other graphical lumnaries, such as Étienne-Jules Marey and Charles Joseph Minard (Friendly and Wainer 2021), as well as some ingenious examples of the use of statistical graphics to solve real-world problems, including John Snow's investigation into the London cholera outbreak (Freedman 1999; Friendly and Wainer 2021) and Florence Nightingale's reporting on the unsanitary treatment of wounded British soldiers during the Crimean War (Brasseur 2005), both of which lead to a great reduction of preventable deaths.

Simultaneously, the field of mathematical statistics was also experiencing significant developments. Building upon the foundation laid by mathematical prodigies such as Jakob Bernoulli, Abraham de Moivre, Pierre Simon Laplace, and Carl Friedrich Gauss, early 19th century pioneers such as Adolph Quetelet and Francis Galton began developing statistical techniques for uncovering hidden trends in the newly unearthed treasure trove of socioeconomic data (Fienberg 1992; Freedman 1999). In the late 19th and early 20th century, these initial efforts were greatly advanced by the theoretical work of figures such as Karl Pearson, Ronald A. Fisher, Jerzy Neyman, and Harold Jeffreys, who established statistics as a discipline in its own right and facilitated its dissemination throughout many scientific fields (Fienberg 1992).

As mathematical statistics gained prominence in the early 20th century, data visualization declined. Perceived as less rigorous than "serious" statistical analysis, it got relegated to an auxiliary position, ushering in "dark age" of statistical graphics (Friendly 2006; Young, Valero-Mora, and Friendly 2011). This development may have been partly driven by the early frequentist statisticians' aspiration to establish statistics as a foundation for determining objective truths about the world and society, motivated by personal socio-political goals (see Clayton 2021). Be it as it may, while statistical graphics also did get popularized and entered the mainstream during this time, only a few interesting developments took place (Friendly and Wainer 2021).

However, beginning in the late 1950's, a series of developments took place which would restore the prominence of data visualization and make it more accessible than ever. Firstly, on the theoretical front, the work of certain academic heavyweights greatly elevated data visualization and its prestige. Particularly, John Tukey (1962; 1977) fervently championed exploratory data analysis and placed data visualization in its centre. Around the same time, Jacques Bertin published his famous *Sémiologie graphique* (1967), which was one of the first works to attempt to lay out a comprehensive system of visual encodings and scales. Secondly, at the more applied level, the development of personal computers (see e.g. Abbate 1999) and high-level programming languages such as FORTRAN in 1954 (Backus 1978), made the process of rendering production-grade figures easier and more accessible than ever before. Combined, these developments fueled a surge in the use and dissemination of data visualizations.

As the millennium drew to a close, several other important developments solidified the foundation of static data visualization. First, William Cleveland made significant contributions to the field, laying out many important principles for scientific data visualization (Cleveland 1985, 1993). Of note, his seminal study on the impact of the choice of visual encodings on statistical judgements remains widely cited today (Cleveland and McGill 1984). Similarly, Edward Tufte introduced essential principles for designing effective graphics, coining terms such as *chartjunk* and *data-to-ink ratio* (Tufte 2001). Finally, Leland Wilkinson's groundbreaking Grammar of Graphics (2012) introduced a comprehensive system for designing charts based on simple algebraic rules, influencing nearly every

subsequent software package and research endeavor in the field of visualization.

3.1.2 Early interactive data visualization: By statisticians for statisticians

Compared to static data visualization, interactive data visualization is much more of a recent development. Consequently, less has been written about its history, owing to the shorter timeline, as well as the rapid evolution of software in the time since its inception and the proprietary nature of some systems. Nevertheless, the brief history of interactive data visualization is still rather compelling.

Following the boom of static data visualization in the 1950's, interactive data visualization would not be left far behind. It started with tools designed for niche, specialized tasks. For example, Fowlkes (1969) designed a system which allowed the users to view probability plots under different configurations of parameters and transformations, whereas Kruskal (1965) created a tool for visualizing multidimensional scaling.

Figure 3.2: John Tukey showcasing the PRIM-9 system (left), with an example of a projected scatterplot [right, @fisherkeller1974]. Screenshots were taken from a video available at: [ASA Statistical Graphics Video Library](https://community.amstat.org/jointscsg-section/media/videos))

However, researchers soon recognized the potential of interactive data visualization as a general-purpose tool for exploring data. The first such general-purpose system was PRIM-9 (Fisherkeller, Friedman, and Tukey 1974). PRIM-9 allowed for exploration of multivariate data via interactive features such as projection, rotation, masking, and filtering. Following PRIM-9, the late 1980's saw the emergence of a new generation of systems which provided an even wider range of capabilities. Tools like MacSpin (Donoho, Donoho, and Gasko 1988), Data Desk (Velleman and Paul 1989), XLISP-STAT (L. Tierney 1990), and XGobi (Swayne, Cook, and Buja 1998) introduced features such as interactive scaling, rotation, linked views, and grand tours (for a glimpse into these systems,

excellent video-documentaries are available at ASA Statistical Graphics Video Library).

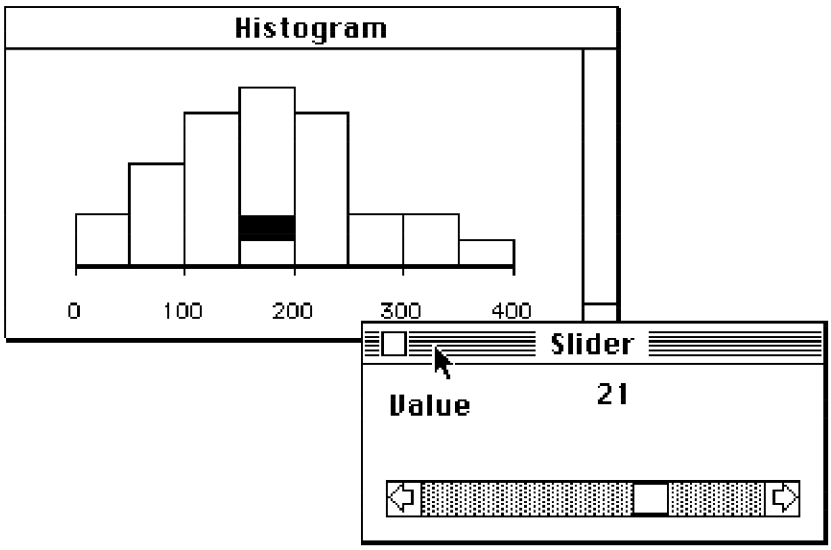


Figure 3.3: Example of interactive control of histogram highlighting in XLISP-STAT. Note that, unlike in many current data visualization systems, aggregation plots were sensitive to data order (not commutative). This non-commutative behavior meant that, for instance, a highlighted segment could appear in the middle of a bar (as seen in the figure above) or multiple non-adjacent highlighted cases might appear as 'stripes'. Figure reproduced from @tierney1990.

3.1.2.1 Open-source Statistical Computing

The proliferation of open-source, general-purpose statistical computing software such as S and R further democratized the access to interactive data visualization tools (see also **?**). Building on XGobi's foundation, GGobi (Swayne et al. 2003), expanded upon on XGobi and provided an integration layer for R. Other tools like MANET (Unwin et al. 1996) and Mondrian (Theus 2002) introduced sophisticated linking techniques, with features such as selection sequences, allowing the users to combine a series of selections via logical operators (see also Unwin et al. 2006). Further, iPlots (Urbanek and Theus 2003) implemented a general framework for interactive plotting in R, allowing not only for one-shot rendering interactive figures from R but also for direct programmatic manipulation. This package was later expanded expanded for big data capabilities in iPlots eXtreme (Urbanek 2011). Finally, the `cranvas` package (Xie, Hofmann,

and Cheng 2014) introduced a set of reactive programming primitives that could be used for building the infrastructure underlying interactive graphics directly in R, within the model-view-controller (MVC) framework.

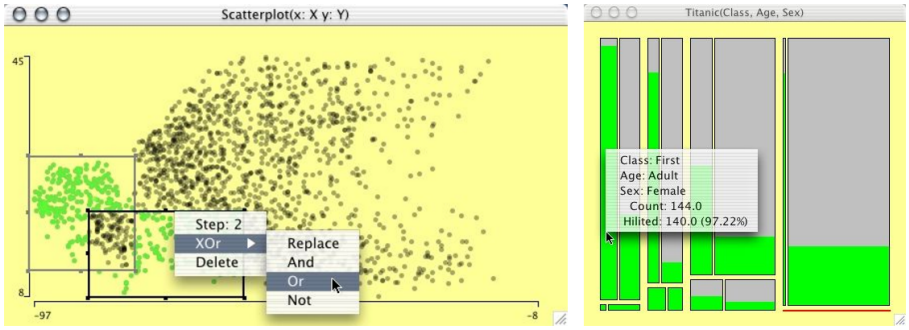


Figure 3.4: Examples of interactive features in Mondrian [@theus2002]: selection operators (left) and mosaic plot with querying (right).

Alongside the more general interactive data visualization frameworks mentioned above, there were also more specialized packages designed for specific techniques and models. For instance, `KLIMT` was developed for interactive visualization of classification and regression trees (Urbanek and Unwin 2001; Urbanek 2002). Similarly, packages like `tourr` (Wickham 2011), `spinifex` (Spyrison and Cook 2020), `liminal` (Lee 2021; Lee, Laa, and Cook 2022) provided tools for exploring large multivariate data sets via grand tour projections (see Cook et al. 1995).

Another important milestone in the history of interactivity in R was the development of Shiny (W. Chang et al. 2024; see also Sievert 2020; **?**), a general framework for developing web apps in R. Shiny uses a client-server MVC-like model, whereby the user defines an R-based server and a web-based UI/controller layer (which is scaffolded in R, but which transpiles to HTML, CSS, and JavaScript). Bi-directional communication between the client and the server is handled via WebSockets (**?**), facilitated through the `httpuv` package (Cheng et al. 2024). The primary advantage of Shiny is that it gives less technical R users the ability to easily create rich interactive web apps, including interactive data visualizations (by re-rendering static plots). The one major downside of Shiny is that, since every interactive event has to do a round-trip from the client to the R server and back again, high-frequency interactions (such as brushing scatterplot points) can become prohibitively slow, particularly at larger data volumes (although there are ways to mitigate this, Sievert 2020).

Over time, there seems to have been a trend towards more of the specialized tools within the R community, and fewer of the general, high-level frameworks (although there were some notable exceptions, such as the `loon` package, Waddell and Oldford 2023). Currently, it seems that R users typically encounter interactive visualizations as part of Shiny (W. Chang et al. 2024) dashboards,

or through R wrappers of interactive data visualization packages ported over from the JavaScript ecosystem (see Section 3.1.3).

3.1.2.2 Common features and limitations of early interactive systems

A common thread among these interactive data visualization systems is that they were designed by statisticians with primary focus on data exploration. High-level analytic features such as linked views, rotation/projection, and interactive manipulation of model parameters made frequent appearance. While these features were powerful, they also contributed to a steeper learning curve, potentially limiting adoption by users without a strong data analytic background. Furthermore, these early tools were typically standalone applications, with only later packages like GGobi and iplots offering integration with other data analysis software and languages. Finally, they often offerered only limited customization options and this made them less suitable for data presentation.

3.1.3 Interactive data visualization and the internet: Web-based interactivity

The end of the millennium marked the arrival of a new class of technologies which impacted interactive data visualization just as much as almost every other field of human endeavor. The rise of the internet in the mid 1990's made it possible to create interactive applications that could be accessed by anyone, from anywhere. This was aided by the dissemination of robust and standardized web browsers, as well as the development of JavaScript as a high-level programming language for the web (for a tour of the language's history, see e.g. Wirfs-Brock and Eich 2020). Soon, interactive visualizations became just one of many emerging technologies within the burgeoning web ecosystem.

Early web-based interactive data visualization systems tended to rely on external plugins. Examples of these include Prefuse (Heer, Card, and Landay 2005) and Flare (developed around 2008, Blokt 2020), which leveraged the Java runtime and Adobe Flash Player, respectively. However, as browser technologies advanced, particularly as JavaScript's performance improved thanks to advances in just-in-time compilation (JIT, see e.g. Clark 2017; Dao 2020), it became possible to create complex interactive experiences directly in the browser. This led to the emergence of several popular web-native interactive data visualization systems in the early 2010s, many of which remain widely used today.

3.1.3.1 D3

D3.js (Mike Bostock 2022) is one of the earliest and most influential web-based visualization systems. As a general, low-level framework for visualizing data, D3 provides of a suite of specialized JavaScript modules for various aspects of

the data visualization workflow, including data parsing, transformation, scaling, and DOM interaction.

For instance, here's how to create a basic scatterplot in D3:

```
import * as d3 from "d3";

const plot = document.querySelector<HTMLDivElement>("#d3-plot")!;
const data = [
  { x: 1, y: 0.41 },
  { x: 2, y: 4.62 },
  { x: 3, y: 7.62 },
  { x: 4, y: 6.54 },
  { x: 5, y: 9.61 },
];

const margin = { top: 10, right: 30, bottom: 30, left: 60 };
const width = parseFloat(plot.style.width);
const height = parseFloat(plot.style.height);

// Create a SVG element, resize it, and append it to #d3-plot
const svg = d3
  .select("#d3-plot")
  .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 + ")");

// Create x and y scales and append them to
const scaleX = d3.scaleLinear().domain([0, 6]).range([0, width]);
const scaleY = d3.scaleLinear().domain([10, 0]).range([0, height]);
svg
  .append("g")
  .attr("transform", "translate(0," + height + ")")
  .call(d3.axisBottom(scaleX));
svg.append("g").call(d3.axisLeft(scaleY));

// Add points
svg
  .append("g")
  .selectAll("dot")
  .data(data)
  .enter()
  .append("circle")
  .attr("cx", (d) => scaleX(d.x))
```

```
  .attr("cy", (d) => scaleY(d.y))
  .attr("r", 2);
```

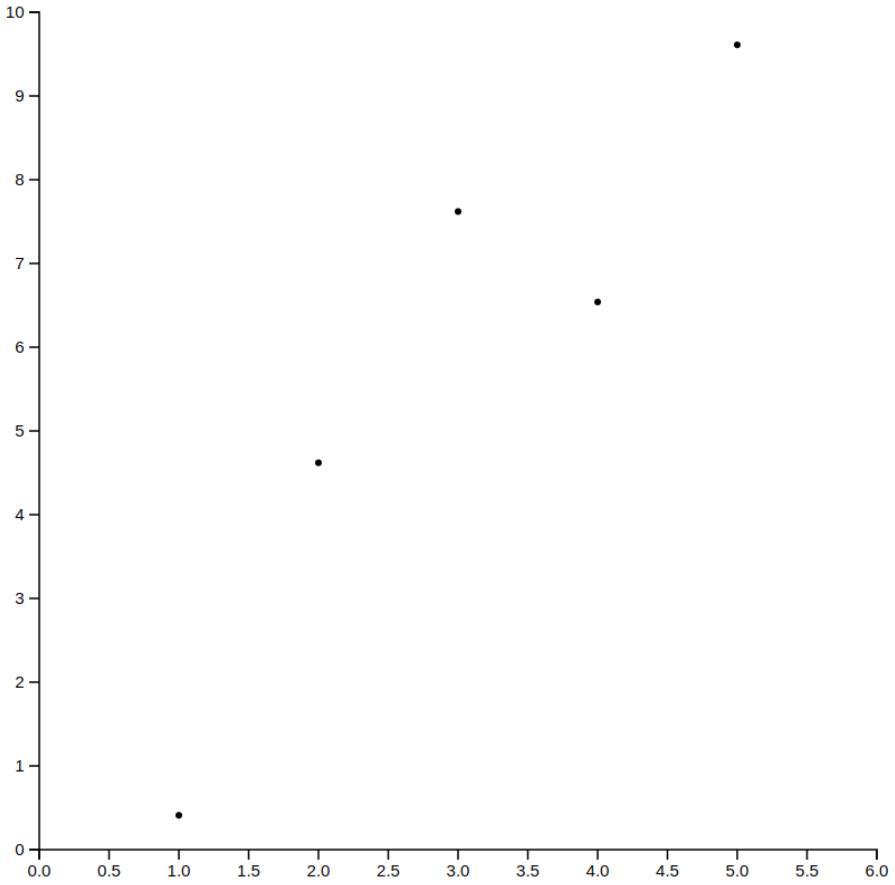


Figure 3.5: Example of a scatterplot built with D3.js. The code was taken from D3 Graph Gallery [@holtz2022b] and adjusted to use ES6 syntax and slightly more informative variable names/comments.

As you can see from Figure 3.5 and the corresponding code, D3 is a fairly low-level framework. Compared to typical high-level plotting functionalities such as those provided by base R or `ggplot2` (R Core Team 2024; Wickham 2016), the user has to handle many low-level details such as scaling and appending of primitives explicitly. This is also the case with interaction. While D3 does provide some methods for handling reactive DOM events, it does not itself provide a system for dispatching and coordinating these events - instead, it delegates this responsibility to the user, and encourages the use of reactive Web

frameworks such as React (Meta 2024), Vue (Evan You and the Vue Core Team 2024), or Svelte (Rich Harris and the Svelte Core Team 2024).

Finally, D3.js visualizations are rendered as Scalable Vector Graphics (SVG) by default. This ensures lossless scaling but may impact rendering performance at high data volumes. While various unofficial alternative rendering engines based on the HTML 5 Canvas element or WebGL, do exist, there are no official libraries with such functionalities as of this date.

3.1.3.2 Plotly and Highcharts

Building upon the low-level infrastructure that D3 provides, many packages such as Plotly.js (Plotly Inc. 2022) and Highcharts (Highsoft 2024) provide high-level abstractions which make the process of building interactive figures easier for the average user. Unlike D3 which provides low-level utilities such as data transformations, scales, and geometric objects, these packages provide a simple declarative framework for rendering entire plots using a static JSON schema.

Here's how we can render the same scatterplot in Plotly, using the R `plotly` package (Sievert 2020):

```r
library(plotly)
data <- data.frame(x = 1:5, y = c(0.41, 4.62, 7.62, 6.54, 9.61))
plot_ly(data, x = ~x, y = ~y)
```

Here's the correponding code in JavaScript:

```javascript
const data = [{
  x: [1, 2, 3, 4, 5],
  y: [0.41, 4.62, 7.62, 6.54, 9.61],
  mode: 'markers',
  type: 'scatter'
}];

Plotly.newPlot('app', data);
```

Clearly, compared to the D3 code used to create Figure 3.5, the code for creating Figure 3.6 is much terser. Many details, such as the axis limits and margins, point size and colour, gridlines, and widgets, are handled implicitly, via default values and automatic inference. Also, note that the figure provides some interactive features by default, such as zooming, panning, and tooltip on hover. Reactivity is handled automatically using systems built on the native DOM Event Target interface (MDN 2024a).

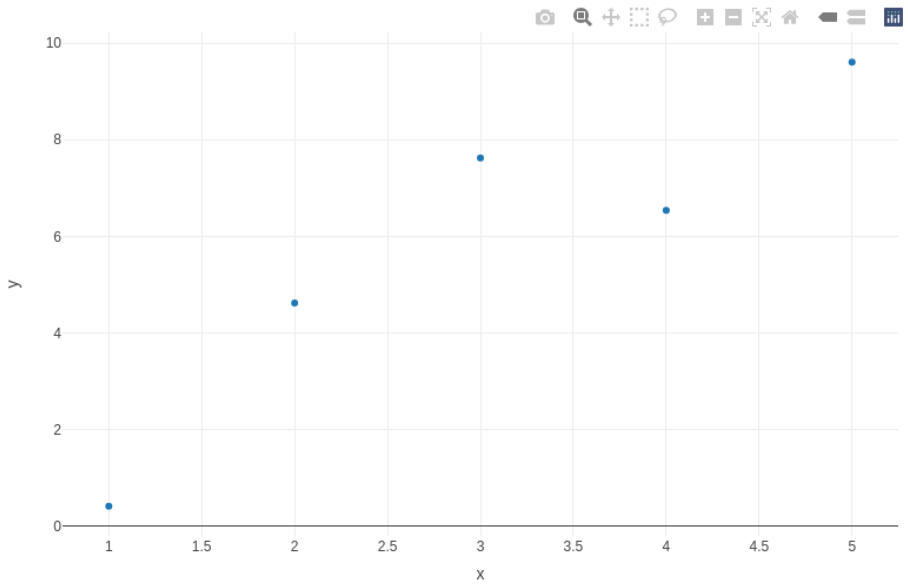


Figure 3.6: Example of a scatterplot with ‘plotly‘.

Highcharts provides a similar JSON-based interface for specifying plots. While perhaps slightly more flexible than Plotly, it also requires more verbose specifications. Because of the similarity, I will not provide a separate example here (interested reader should look up the package's website, Highsoft 2024).

Finally, like D3, both plotly.js and Highcharts also render the graphics in SVG by default. However, unlike D3, they both also provide alternative rendering engines based on WebGL (Highsoft 2022; Plotly Inc. 2024). This makes them more ergonomic for use with large data sets.

3.1.3.3 Vega and Vega-Lite

Vega (Satyanarayan et al. 2015; Vega Project 2024d) is another popular interactive data visualization package. Like Plotly and Highcharts, Vega is also partially built upon the foundation of D3 and uses JSON schema for plot specification. However, Vega is more low-level and implements a lot of custom functionality. This allows it to offer more fine-grained customization of graphics and interactive behavior, leading to greater flexibility.

However, this added flexibility does come at a cost. Compared to the high-level frameworks like Plotly and Highcharts, Vega is significantly more verbose. For instance, creating a scatterplot matrix with linked selection in Vega requires over 300 lines of JSON specification, not including the data and using default formatting (Vega Project 2024b).

Vega-Lite (Satyanarayan et al. 2015) attempts to remedy this complexity by providing a high-level interface to Vega. Here's how we can define a scatterplot with zooming, panning, and tooltip on hover in Vega-Lite:

```
library(vegawidget)

plot_spec <- list(
    `$schema` = vega_schema(),
    width = 500,
    height = 300,
    data = list(values = data),
    mark = list(type = "point", tooltip = TRUE),
    encoding = list(
      x = list(field = "x", type = "quantitative"),
      y = list(field = "y", type = "quantitative")
    ),
    params = list(list(name = "grid",
                       select = "interval",
                       bind = "scales"))
  )

plot_spec |> vegawidget()
```

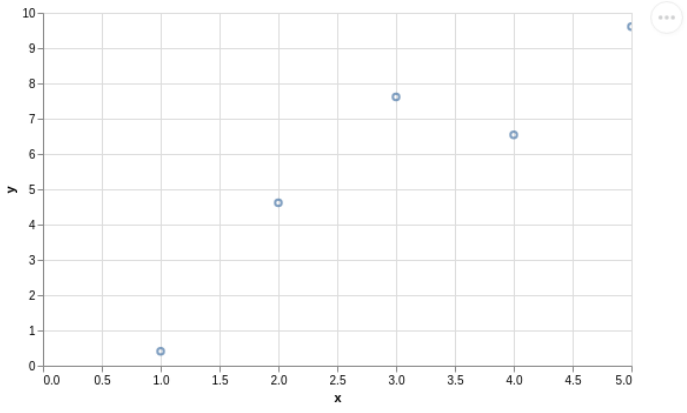


Figure 3.7: Example of a scatterplot built with 'vegalite'.

Just for clarity, the R code above corresponds to the following declarative JSON schema:

```
{
  $schema: "https://vega.github.io/schema/vega-lite/v5.json",
  width: 500,
  height: 300,
  data: { values: [
    { x: 1, y: 0.41 },
    { x: 2, y: 4.62 },
    { x: 3, y: 7.62 },
    { x: 4, y: 6.54 },
    { x: 5, y: 9.61 }]
  },
  mark: {"type": "point", "tooltip": true},
  encoding: {
    x: { field: "x", type: "quantitative" },
    y: { field: "y", type: "quantitative" }
  },
  params: [{ name: "grid", select: "interval", bind: "scales" }]
};
```

Note that the zooming and panning capability is provided by the `params` property, which declaratively specifies a list of plot parameters that can be modified by interaction (see Vega Project 2024c). In the case above, the specification creates a two-way binding between plot scales and mouse selection events (Vega Project 2024a).

3.1.3.4 Common features and limitations of web-based interactive systems

In general, these contemporary web-based interactive data visualization systems offer a great deal of flexibility, making them well-suited to modern data presentation. However, all of this expressiveness does seem to come at a cost. Compared to the earlier statistical graphics systems, described in Section 3.1.2, many of the more advanced features that used to be common are either missing or require a significant effort to implement, such that they are only accessible to expert users. This is evidenced by their infrequent appearance in documentation and example gallery pages.

For instance, the R Graph Gallery entry on Interactive Charts (Holtz 2022) features multiple interactive figures implemented in the JavaScript libraries described above. However, all of these examples show only surface-level, single-plot interactive features such zooming, panning, hovering, 3D rotation, and node repositioning. The Plotly Dash documentation page on Interactive Visualizations (Plotly Inc. 2022) does feature two examples of simple linked cross-filtering, however, the vast majority of visualizations in the Plotly R Open

Source Graphing Library documentation page (Plotly Inc. 2022) show examples only surface-level interactivity. Similarly, VegaLite Gallery pages on Interactive Charts (Vega Project 2022) feature many examples, however, only a limited number of examples show linked or parametric interactivity (see e.g. Interactive Multiview Displays). Finally, the Highcharter Showcase Page (Kunst 2022) does not feature any examples of linking.

Even when advanced features such as linking and parametric manipulation are supported, they are often limited in some way. For example, the following is a quote from the website of Crosstalk, a package designed to enable linking in R, using web-based interactive widgets created with the `htmlwidgets` package (Vaidyanathan et al. 2021) or R Shiny (W. Chang et al. 2024):

> "Crosstalk currently only works for linked brushing and filtering of views that show individual data points, not aggregate or summary views (where "observations" is defined as a single row in a data frame). For example, histograms are not supported since each bar represents multiple data points; but scatter plot points each represent a single data point, so they are supported."
>
> - Posit (formerly RStudio Inc.) (2025)

Of course, with enough effort, these web-based visualization systems can still be used to create rich figures with advanced interactive features such as linked views and parametric interaction. However, implementing these features often requires stepping down a level of abstraction and dealing with low-level language primitives. This defeats the purpose of using a high-level libraries and creates a barrier to entry for casual users (Keller, Manz, and Gehlenborg 2024). It also may explain why interactive visualizations are nowadays mainly used for data presentation, not data exploration (Batch and Elmqvist 2017). With the high upfront cost of learning these package's APIs, creating rich interactive figures may be a task best suited for dedicated developers working inside large organizations, rather than individual researchers/analysts.

3.2 What even is interactive data visualization?

> If it looks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck.
>
> […] The irony is that while the phrase is often cited as proof of abductive reasoning, it is not proof, as the mechanical duck is still not a living duck
>
> Duck Test entry, (Wikipedia 2022)

In the previous section (Section 3.1), I provided an overview of the history and present state of interactive data visualization, discussing a number of features and systems. However, while doing so, I avoided one crucial question: what constitutes an interactive data visualization?

Surprisingly, despite the widespread popularity of interactive visualizations, there is no universally agreed-upon definition of interactivity (Vanderplas, Cook, and Hofmann 2020). Within the data visualization literature, the terms "interactive" and "interaction" are rarely explicitly defined. And even when they are, the definitions are often incongruent or even contradictory (see e.g. Dimara and Perin 2019; Elmqvist et al. 2011; Pike et al. 2009). Finally, similar conceptual ambiguity extends to other terms commonly used in the field, such as a "dashboard" (Sarikaya et al. 2018).

This lack of a clear consensus makes the task of discussing interactive data visualization difficult. Ignoring the issue could lead to confusion, while a truly comprehensive dive into the terminology surrounding interactive data visualization could become excessively verbose, as evidenced by the existence of research papers dedicated to the topic (see e.g. Dimara and Perin 2019; Elmqvist et al. 2011). To address this issue, this section aims to provide a concise overview of how interactivity has been conceptualized in the literature. The goal is to establish a clear framework for understanding "interactive" and "interaction" within the context of this thesis.

3.2.1 Interactive vs. interacting with

First, the word "visualization" in "interactive data visualization" can be interpreted in two different ways:

1. As a noun: a concrete chart or figure
2. As a nominalized verb: the process of interacting with a figure

In data visualization literature, both interpretations are frequently used, leading to significant ambiguity (Dimara and Perin 2019; Pike et al. 2009; see also Yi et al. 2007). On one hand, some researchers focus on the mathematical and computational aspects of visualization, discussing specific systems and implementations (see e.g. Buja, Cook, and Swayne 1996; Kelleher and Levkowitz 2015; Leman et al. 2013; G. Wills 2008). Others prioritize the more cognitive or human-computer interaction (HCI) aspects of interactive data visualization, exploring what impact different kinds of visualization techniques have on the user's ability to derive insights from the data (see e.g. Brehmer and Munzner 2013; Dimara and Perin 2019; Dix and Ellis 1998; Pike et al. 2009; Quadri and Rosen 2021; Yi et al. 2007).

Of course, many interactive data visualization papers discuss both implementation and user experience. However, the dual interpretation of the term "interactive data visualization" does complicate literature search. It also highlights the

interdisciplinary nature of the field, showing its connections to statistics, computer science, applied mathematics, business analytics, HCI, and cognitive psychology (see Brehmer and Munzner 2013; Dimara and Perin 2019). While this interdisciplinary nature of interactive data visualization is certainly a strength, it can also lead to confusion. As such I think it is necessary to clearly define key terms.

To ensure clarity throughout thesis, the term *"interactive data visualization"* will primarily refer to concrete charts or figures. When referring to the *practice* of interactive data visualization, I will attempt to use more active phrasing such as *"interacting with a visualization"* or *"user's interaction with a visualization"*, to indicate that what is being referred to is the activity or process of visualization, rather than any concrete figure or chart.

3.2.2 Interactive *enough*?

Even when we use the term "interactive data visualization" to refer to concrete charts or figures, the meaning still remains fairly ambiguous. What is the bar for calling a figure "interactive"? What features should interactive figures have? Surprisingly, it is hard to find consensus on these topics among data visualization researchers, and the criteria tend to vary a lot, such that the same figure may be considered interactive by some but not by others.

Some researchers adopt a broad definition of interactive data visualization, considering almost any figure combined with an interactive graphical user interface (GUI) as interactive, as long as it allows for some level of user manipulation (Brodbeck, Mazza, and Lalanne 2009). For others, the speed of the computer's responses to user input is important, with faster updates translating to greater interactivity (Becker and Cleveland 1987; Buja, Cook, and Swayne 1996). Some also differentiate between "interactive" and "dynamic" manipulation, such that interactive manipulation involves discrete actions such as pressing a button or selecting an item from a drop-down menu, whereas dynamic manipulation involves continuous actions, like moving a slider or clicking-and-dragging to highlight a rectangular area (Rheingans 2002; Jankun-Kelly, Ma, and Gertz 2007; see also Dimara and Perin 2019).

However, many other researchers ascribe to a much narrower view of interactive data visualization, which hinges on high-level analytic features that allow efficient exploration of the data. These features include the ability to generate different views of the data (by e.g. zooming, panning, sorting, and filtering), and the reactive propagation of changes between connected or "linked" parts of a figure (Kehrer et al. 2012; Buja, Cook, and Swayne 1996; Keim 2002; Unwin 1999; Chen et al. 2008). An often cited guideline is the visual information seeking mantra: overview first, zoom and filter, then details-on-demand (Shneiderman 2003). Similarly, in visual analytics research, a distinction is made between "surface-level" (or "low-level") and "parametric" (or "high-level") interactions, where surface-level interactions manipulate attributes of the visual

Table 3.1: Summary of the perspectives on interactivity

Name	Details
User interaction	The user can interactively manipulate the figure in some way
Real-time updates	The user's interactions propagate into the visualization with little to no lag
Plot- and data-space manipulation	The user can interactively explore different parts of the data set by doing act
Linked views	The user's interactions propagate across multiple plots (e.g. linked highlighti
Parametric updates	The user can manipulate the parameters of some underlying mathematical m

domain only (e.g. zooming and panning), whereas parametric interactions manipulate attributes of mathematical models or algorithms underlying the visualization (Leman et al. 2013; Self et al. 2018; Pike et al. 2009).

Table 3.1 provides a concise summary of the several perspectives on interactivity discussed above. It meant to serve as a reference point for future discussions within the text, though it is important to note that this is not an exhaustive list. For a more comprehensive taxonomy of interactive visualization systems and features, see e.g. Dimara and Perin (2019), Yi et al. (2007).

3.2.3 Complexity of interactive features

The way we define interactivity is not just a matter of taste or preference: it has a significant impact on the complexity and feasibility of our systems. As we will see in Section 3.2.5, some simple features are fairly easy to implement, requiring just a thin interactive layer over a static data visualization system, whereas others come with a significant overhead, requiring an entirely different framework than static visualization.

To make the point with a particularly blunt example, many programming languages support a read-evaluate-print loop (REPL). This allows interactive code execution from the command line: the user inputs code, the interpreter evaluates it, outputs results, and waits for more input. If the language supports plotting, running code to generate plots could be considered an "interactive data visualization system." User interaction with the REPL modifies the visual output, and with fast-enough input, updates could appear almost instantly (thus satisfying the user interaction and real-time update definitions of interactivity, see table). This would make almost every programming language an "interactive data visualization system", requiring no additional effort.

However, I would argue that, today, this view stretches the concept of interactivity. It is true that, historically, the command line was considered a highly interactive user interface (see e.g. Foley 1990; Howard and MacEachren 1995). However, with advancements in processor speeds and the widespread adoption of graphical user interfaces (GUIs), user expectations have evolved. Nowadays, we typically associate interactivity with direct manipulation of visual elements

and immediate feedback (Dimara and Perin 2019; Urbanek 2011). Thus, we can see that what's considered "interactive" evolves over time.

But even with figures that are manipulated directly, there still are considerable differences in what different features imply for implementation requirements. Some features, like changing color or opacity of points in a scatterplot affect only the visual attributes of the plot and not the underlying data representation. This makes them simple to implement as they do not require any specialized data structures or complex computations, and the primary cost lies in re-rendering the visualization.

In contrast, some interactive features require a lot more infrastructure. For instance, filtering, linked highlighting, or parametric interaction require specialized data structures and algorithms beyond those that would be required in static plots. This is because, each time the user engages in an interaction, entirely new summaries of the underlying data may need to be computed.

To give a concrete example, when a user selects several points in a linked scatterplot (see Section 3.2.5.8), we first have to find the ids of all the selected cases, recompute the statistics underlying all other linked plots (such as counts/sums in barplots or histograms), train all of the relevant scales, and only then can we re-render the figure. Likewise, when interactively manipulating a histogram's binwidth, we need to recompute the number of cases in each bin whenever the binwidth changes. To maintain the illusion of smooth, "continuous" interaction (Dimara and Perin 2019), these computations need to happen fast, and as such, computational efficiency becomes imperative at high data volumes.

3.2.4 Working definition

As discussed in previous sections, the definition "interactive data visualization" varies across fields and researchers. Moreover, when building interactive data visualization systems, different definitions imply varying levels of implementation complexity. Thus, we need to establish clear criteria for our specific definition.

Data visualization can be broadly categorized into two primary modes: presentation and exploration. While both modes share a bulk of common techniques, each comes with a different set of goals and challenges (Kosara 2016). Data presentation starts from the assumption that we have derived most of the important insights from our data already, and the goal is now to communicate these insights clearly and make an impactful and lasting impression (Kosara 2016). In contrast, data exploration begins from a position of incomplete knowledge - we accept that there are facts about our data we might not be aware of. Thus, when we explore data with visualizations, the goal is to help us see what we might otherwise miss or might not even think to look for (Tukey et al. 1977; Unwin 2018).

However, it is not always the case that more complex visuals necessarily translate to better statistical insights. In static visualization, it is a well-established that

plots can include seemingly sophisticated features which do not promote the acquisition of statistical insights in any way (Cairo 2014, 2019; Gelman and Unwin 2013; Tufte 2001). Similarly, adding interactivity to a visualization does not always improve its statistical legibility (see e.g. Abukhodair et al. 2013; Franconeri et al. 2021).

I propose to treat interactive features the same way we treat visual features in static visualization. Specifically, I propose the following working definition:

> When building interactive data visualization systems, we should prioritize interactive features which promote statistical understanding.

If we accept this proposition, then several important consequences follow. First, we must favor high-level, data-dependent, parametric interactions over the purely graphical ones. That is not to say that purely graphical interactive features cannot useful. For instance, in the case of overplotting, changing the size or alpha of points in a scatterplot can help us see features that would otherwise remain hidden. Nevertheless, I argue that the ability to see entirely new representations of the data is what makes some interactive data visualizations systems particularly powerful. The interactive features that enable this, such as linked highlighting and parameter manipulation, go beyond aesthetics, and empower the users to explore the data in a much more dynamic way, compared to static graphics.

3.2.5 Common interactive features

This section describes several common types of interactive features that tend to frequently appear in general interactive data visualization systems. It is only meant as an overview (for more comprehensive taxonomies of interactive features, see Dimara and Perin 2019; Unwin et al. 2006; Yi et al. 2007). For each feature, I highlight its core properties, common use cases, and implementation requirements.

3.2.5.1 Changing size and opacity

One of the simplest and most widely-implemented interactive features is the ability to adjust the size and opacity of geometric objects. This feature gives the user the ability to dynamically shrink or grow objects and make semi-transparent, fully transparent, or opaque.

The ability to shrink objects or make them semi-transparent can be particularly useful at high data volumes, since this can reveal trends that may be otherwise hidden due to overplotting. For example, in scatterplots, shrinking points and making them semi-transparent makes it possible to identify high-density regions and can in fact provide an approximation to a 2D kernel density plot (see e.g.

Dang, Wilkinson, and Anand 2010). The same applies to all other types of plots where the where objects or glyphs may be plotted on top of each other at high densities, such as parallel coordinate plots (Theus 2008).

This feature usually fairly easy to implement, since it involves manipulating visual attributes only. Specifically, in many interactive systems, size and alpha multipliers are independent parameters of the visual representation, which do not depend on the underlying data in any way. In other words, when we manipulate size or opacity of geometric objects in our plots, we do not need to worry about what data these objects represent. Compared to other interactive features, this makes it relatively simple to add this functionality to an existing static visualization system (see Braşoveanu et al. 2017).

3.2.5.2 Zooming and panning

Another two significantly related interactive features are zooming and panning. They are often used in tandem, and both involve interactive manipulation of scale limits. For this reason, I discuss them here simultaneously, in a single subsection.

Zooming, depicted in Figure 3.8, allows the user to magnify into a specific region of a plot. A common approach involves creating a rectangular selection and the axis scales are then automatically adjusted to match this region, however, other techniques do exist, for instance a symmetric zoom centered on a point using a mouse wheel. Zooming is useful because it allows the user to get a better sense of the trend within the magnified region, and discover patterns that may be otherwise obscured due to overplotting or improper aspect ratio (see e.g. Buja, Cook, and Swayne 1996; Dix and Ellis 1998; Unwin 1999; Theus 2008; Yi et al. 2007).

After zooming, it is useful to retain the ability to navigate the wider plot region while preserving the current zoom level and aspect ratio. Panning addresses this need. By performing some action, typically right-click and drag, the user can move the center of the zoomed-in region around, exploring different areas of the plot.

Zooming and panning can be implemented by manipulating scales only, and this also makes them generally fairly straightforward to implement, similar to changing size and opacity. However, there are a few issues to consider. First, whereas continuous axes can be be zoomed and/or panned by simply modifying the axis limits, zooming discrete axes requires a bit more nuance (see e.g. Wilkinson 2012). Second, it is often desirable to give the user the ability to zoom-in multiple levels deep, and this makes maintaining a reversible history of previous zoom-states essential (Unwin 1999). Third, at times, it can be useful to link scale updates across multiple plots, such that, for example, zooming or panning a plot in a scatterplot matrix produces the same actions in other plots with the same variable on one of the axes. Finally, an advanced feature that can

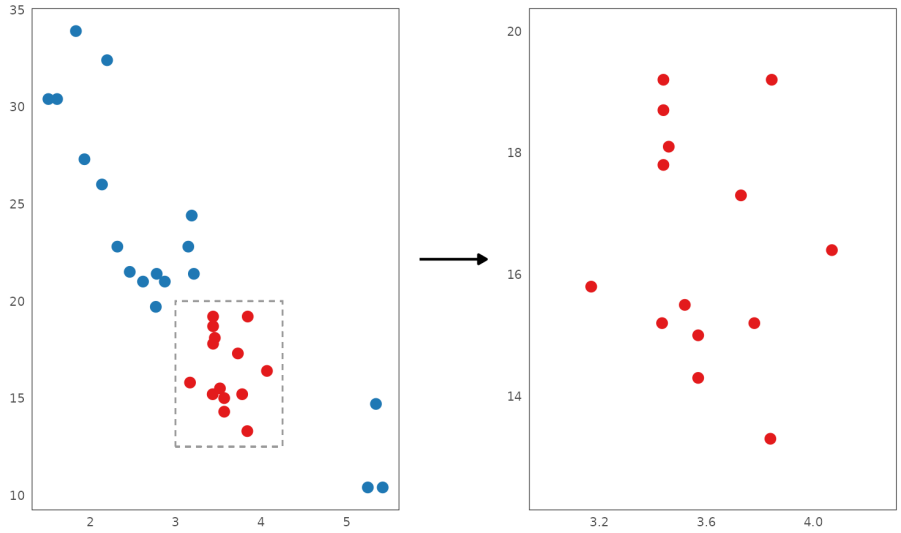


Figure 3.8: Zooming involves shrinking the axis limits to obtain a more detailed view of the data. Typically, the user selects a rectangular region of the plot (left) and the plot scales are then adjusted so that the region fills up the entire plot area (right). Notice the change in the axis limits.

be also quite useful is semantic or logical zooming (Keim 2002; Unwin 1999; Yi et al. 2007). This technique goes beyond magnifying objects; it also increases the level of detail the objects display as the user zooms in. Semantic zooming can be particularly powerful when combined with hierarchical data such as geographic information, however, it also introduces additional complexity, since the effects of the zoom action propagate beyond x- and y-axis scales.

3.2.5.3 Querying

Querying is another popular interactive feature that is usually fairly straightforward to implement. As shown in Figure 3.10, the way querying is typically implemented is that when a user mouses over a particular geometric object, a small table of key-value pairs is displayed via a tool-tip/pop-up, showing a summary of the underlying data point(s) (Urbanek and Theus 2003; Xie, Hofmann, and Cheng 2014). This makes it possible to look up precise values that would otherwise be available only approximately via the visual representation.

Querying is useful because it combines the best features of graphics and tables. Specifically, it allows the user to overcome Tukey's famous prescriptions: "graphics are for the qualitative/descriptive [...] never for the carefully quantitative (tables do that better)", and: "graphics are for comparison [...] not for access to individual amounts" (Tukey 1993). By providing the option to query

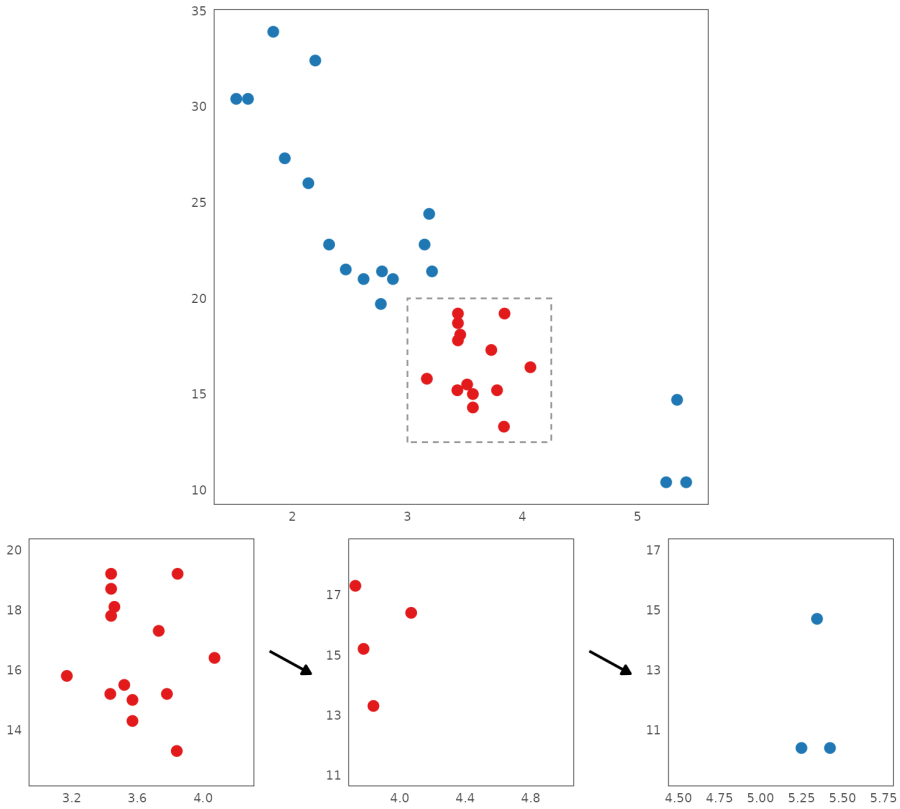


Figure 3.9: Panning involves moving the axis limits while retaining the same zoom level and axis ratio. After zooming into a rectangular region (top row), the user can around the plot region, usually by clicking and dragging (bottom row).

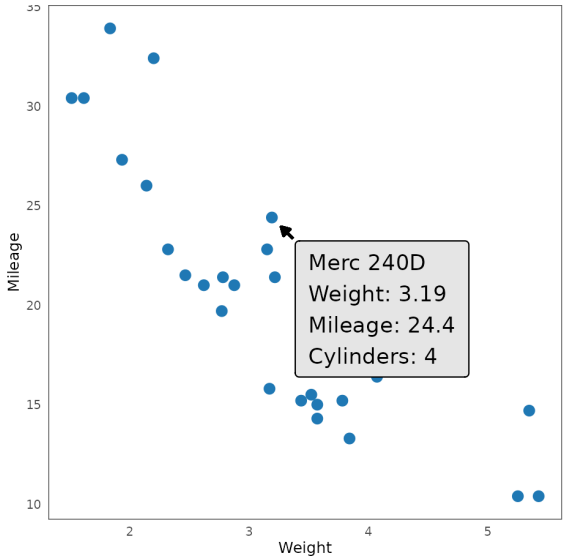


Figure 3.10: Querying involves hovering over an object to display its associated data values in a table or pop-up. Notice that this can include both plotted values (‘weight‘, ‘mileage‘) as well as values that are not directly represented in the plot (car name, ‘cylinders‘).

individual objects, the user can seamlessly transition between the high-level analytic overview of the graphic and low-level quantitative detail of a table. This facilitates high-precision analytic tasks, such as identifying specific outliers or calculating exact magnitudes of differences (Unwin et al. 2006).

Additionally, querying also allow us to show more information than is displayed via the visual encodings alone (see again Figure 3.10). Specifically, whereas most plots can encode only two or three variables, we can assign an arbitrary number of key-value pairs to the rows of the query table/pop-up. However, it is crucial to balance the level of detail against visual clutter. Too many rows may overtax the attention of the user and also can lead to clipping/overplotting issues, if the query table cannot fit inside the plotting area. Further, there are better methods for retrieving very detailed information from interactive visualizations.

Finally, while querying is also one of the more straightforward features, its implementation does present certain challenges. First, a naive implementation might simply display derived data values in the state just before they are mapped to visual attributes via scales, however, these are not always the most informative. For instance, in a stacked barplot, returning the original (unstacked) values is more useful than the stacked ones. Second, aggregate plots such as barplots or histograms do generally present some design decisions (see Unwin et al. 2006). In the case of one-to-one plots such as scatterplots, query data for an object (point) can be obtained by simply retrieving the corresponding row. However,

in aggregate plots like barplots and histograms, a single object may correspond to multiple rows. This necessitates summarizing the underlying data, and often there may be no single "correct" summary. For instance, when querying a bar in a barplot, should we return the sum of the underlying continuous variable, some other numeric summary such as the mean or maximum, the set of all unique values, multiple of these summaries, or perhaps something else entirely? Similar ambiguities arise when querying objects which are partially selected or highlighted (see Section 3.2.5.8): should the query return summaries corresponding to the entire object, the highlighted parts, or both?

3.2.5.4 Sorting and reordering

With plots of discrete (unordered) data, a highly useful feature can be to sort or reorder objects based on some criterion (see Unwin 2000; Unwin et al. 2006). For example, with barplots, in the absence of other ordering rules, bars are typically ordered by the lexographical order of the x-axis variable. However, sometimes, we can glean interesting patterns by sorting the bars in some other order, for example by their height, see Figure 3.11.

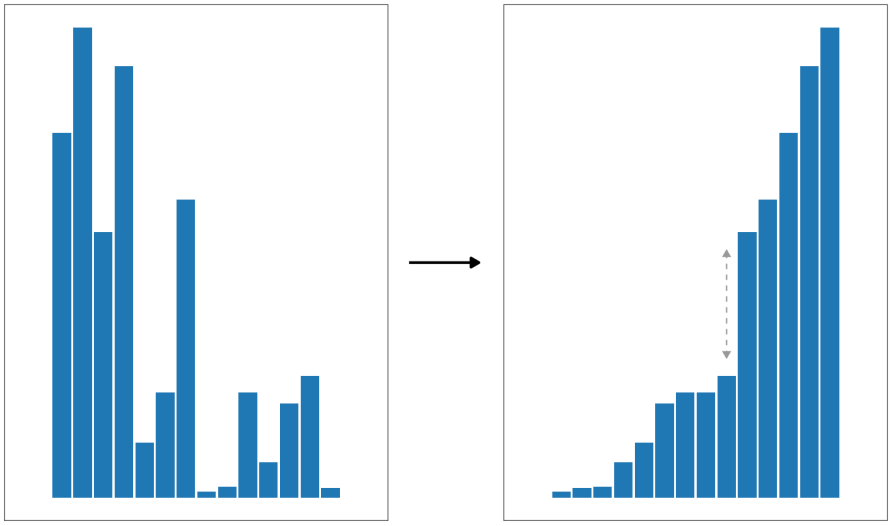


Figure 3.11: Sorting or reordering can highlight interesting trends. For instance, sorting lexicographically ordered bars (left) by bar height (right) in the figure above immediately reveals a significant gap between the five tallest bars and the rest (gray dashed line).

There are more sophisticated ways to sort objects in a plot than just sorting bars by height, however. For instance, in plots which show multiple summary statistics, any may serve as the basis for the sorting rule; for instance a boxplot may

be sorted by the median, upper and lower quartile, the maximum, and the minimum (Unwin et al. 2006). Likewise, in the presence of selection/highlighting, objects may be sorted by the summary statistic on the highlighted parts. Alternatively, some systems allow users to permute the order of discrete scales manually by swapping the position of categories pairwise, a feature which can be particularly useful in parallel coordinate plots (Unwin et al. 2006; Urbanek 2011). Finally, in the presence of many categories, sorting may also be usefully combined with lumping categories below a certain threshold together (Unwin 2000).

Like zooming and panning, basic sorting typically involves the manipulation of axis scales only, making it also a fairly straightforward feature to implement. However, the more sophisticated sorting features can pose non-trivial implementation challenges (Unwin et al. 2006). For instance, sorting by custom summary statistics or manually permuting discrete scale order may require specialized system components and behavior.

3.2.5.5 Parametric interaction

As discussed in Section 3.2.3, another valuable class of interactive features are those which affect the computation of the summary statistics underlying the graphic (also called "parametric interaction," Leman et al. 2013; Self et al. 2018; Urbanek 2011). These features extend beyond simple manipulation of visual attributes, requiring that user interaction penetrates much deeper into the data visualization pipeline. Fundamentally, these features involve the manipulation of the parameters of some underlying mathematical model or algorithm.

An illustrative and popular example of parameter manipulation is dynamically changing histogram binwidth or anchor. Assuming a fixed binwidth w and an anchor a, we can describe a histogram via a function h that, for each observation of a continuous variable x_i returns an index j of the corresponding bin, such that, for an ordered list of bins breaks b_j, we have $x_i \in [b_j, b_{j+1})$[1]:

$$h(x_i; a, w) = \lfloor (x_i - a)/w \rfloor + 1$$

Thus, a histogram really is a kind of a mathematical model, and can in fact be seen as a crude form of density estimation (see e.g. Bishop and Nasrabadi 2006, 4:120–22). Manipulating histogram bins amounts to manipulating the parameters of the function h. Crucially, unlike changes to surface-level visual attributes like size or opacity, changing binwidth or anchor requires recomputing the underlying summary statistics (Urbanek 2011). As noted in Section 3.2.3, these

[1] Technically, if there are any values $x_i < a$, we will have negative indices ($j < 0$), and if all values are significantly larger than the anchor, such that $x_i > a + w$, the indices will not start at 1. So, to implement the histogram properly, we should shift all indices by subtracting the minimum index. Finally, if the histogram binwidth is not fixed, h becomes more complex as well.

changes can have significant downstream effects. For instance, increasing the binwidth may cause certain bins to contain more data points than the current maximum, potentially requiring the adjustment of the upper y-axis limit, to prevent the bars from overflowing the plotting area.

There are other, more complex types of parametric interaction, than just changing histogram binwidth or anchor. These include, for example, modifying the bandwidth of a kernel density estimator, specifying the number of clusters in a clustering algorithm, or manipulating splitting criteria in classification and regression trees, as well as regularization parameters in smooth fits (for some more examples, see Leman et al. 2013; Self et al. 2018).

Because parametric interaction necessitates recalculating the plot's underlying summary statistics, it is both more computationally expensive and as well as more difficult to implement. The interactive system must be able to respond to user input by recomputing relevant summaries and updating dependent plot parameters. In some systems such as Shiny (W. Chang et al. 2024), the common approach is to re-render the entire plot from scratch each time any interaction occurs. However, this can become prohibitively expensive when these deep, parametric interactions are combined with rapid interactions closer to the surface of the visualization pipeline. Thus, the development of generic and efficient data visualization pipelines still remains an open research problem (Wickham et al. 2009; Vanderplas, Cook, and Hofmann 2020; Xie, Hofmann, and Cheng 2014).

3.2.5.6 Animation and projection

A particularly useful form of parametric interaction involves the ability to control a continuous traversal through a series of states, observing the resulting changes as animation. This technique is especially useful when combined with projective techniques such as the grand tour (see Chen et al. 2008; for a recent comprehensive review, see Lee et al. 2022), and for this reason I discuss them both here, within the same subsection.

A common and straightforward application of interactive animation is visualizing transitions in data subsets ordered by a specific variable, such as time. A particularly famous example of this technique is the interactive animation of the Gapminder data set (Rosling and Zhang 2011), which illustrates the joint evolution of GDP and life expectancy for countries worldwide. The interactive control of the timeline (play, pause, and pan) empowers users to explore time-dependent trends within this relatively high-dimensional data set, revealing trends that would be challenging to visualize by other means. For instance, the visualization clearly depicts the profound drop in both GDP and life expectancy during the second world war, followed by the subsequent rapid recovery and growth after 1945.

Interactive animation becomes particularly powerful when coupled with tech-

niques like the grand tour (Asimov 1985; Buja and Asimov 1986; Cook et al. 1995), designed for exploring high-dimensional datasets. Because data visualizations are typically limited to two dimensions, effectively representing high-dimensional data is challenging. The grand tour technique addresses this issue by projecting the data onto a series of lower-dimensional (two-dimensional) subspaces, interpolating between these projections, and animating the results to create a "tour" of different data views (Cook et al. 1995). By surveying this series of projections, the users may discover high-dimensional outliers, clusters, or non-linear dependencies (Wickham 2011), and this discovery can be greatly aided by interactive controls of the animation's timeline or even manual control of the tour's direction (Chen et al. 2008; Lee et al. 2022). Finally, the technique also integrates well with other interactive features, such as linked selection and querying/tooltips (Cook et al. 1995; Wickham 2011; Lee, Laa, and Cook 2022; Lee et al. 2022).

The implementation complexity of interactive animation varies considerably depending on its application. While animating data subsets based on a single variable, as in the Gapminder visualization (Rosling and Zhang 2011), presents no greater implementation challenges than previously discussed techniques, computing the grand tour path requires specialized algorithms (see, e.g., Chen et al. 2008, for a brief description). However, if the data subsets corresponding to each animation frame are pre-computed, the animation itself is generally fairly straightforward to implement.

3.2.5.7 Representation switching

Another specialized kind of parametric (or semi-parametric) interaction involves changing the representation of the underlying data. It is well known that the same data can often be visualized using various sets of visual encodings (Wilkinson 2012), with some being more effective for answering specific questions than others. Enabling users to switch between these various representations provides greater flexibility for data exploration (Yi et al. 2007). However, for certain plot types, changing the representation involves more than just altering surface-level visual attributes; it also necessitates recalculating derived statistics.

A typical example is switching between a barplot and a spineplot, see Figure 3.12. Barplots are effective for comparing absolute quantities. Specifically, by encoding categories along the x-axis and continuous quantities along the y-axis (bar height), we can easily compare the quantities across categories. Color-coding parts of the bars as segments allows us to visualize a second categorical variable, enabling subgroup comparisons of absolute values. However, barplots are less well-suited for comparing the *proportions* represented by these segments, particularly when bar heights vary considerably.

Spineplots, on the other hand, present a way of visualizing the same sort of data as a barplot while making it much easier to compare proportions. Specifically, in a spineplot, the heights of the bars are all normalized to 1, such that the

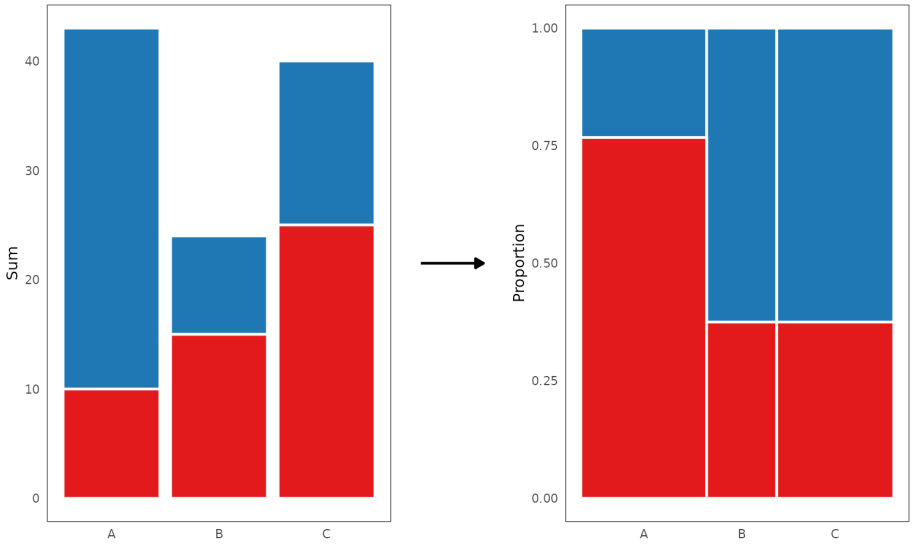


Figure 3.12: Switching representation can be an effective way to derive new insights from the data. A barplot (left) represents the same underlying data as a spineplot (right), however, the former is better for comparing absolute counts whereas the latter is better for comparing proportions. Note that in the spineplot, it is much easier to see that the proportion of the red cases is the same in categories B and C.

segments show a proportion of the total, and the original values are instead encoded as the bar width, which is stacked along the x-axis. Thus, the fixed height of bars makes it easy to compare the segments proportionally.

Other examples of switching of representations include switching from a histogram to spinogram (a normalized version of the histogram) and switching between aggregate geometric objects and individual points (e.g. boxplot, parallel coordinate plots).

3.2.5.8 Linked selection

Linked selection, also known as linked brushing, linked highlighting, or linked views, is often considered one of the most versatile and powerful interactive data visualization features (see e.g. Becker and Cleveland 1987; Buja, Cook, and Swayne 1996; Wilhelm 2003; Heer and Shneiderman 2012; Ward, Grinstein, and Keim 2015; Ware 2019). Fundamentally, it involves creating a figure with multiple "linked" plots. The user can then click or click-and-drag over objects in one plot, and the corresponding cases are highlighted across all the other plots, see Figure 3.13. This makes it possible to quickly quickly explore trends across different dynamically-generated subsets of the data (Dix and Ellis 1998). The ability to quickly materialize alternative views of the data makes this a particularly effective tool for data exploration (Wilhelm 2008; G. Wills 2008).

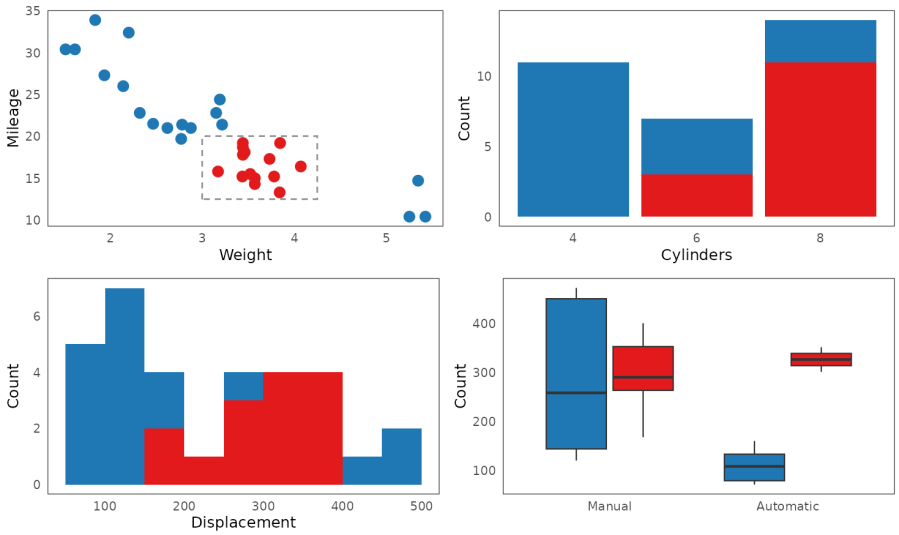


Figure 3.13: Linked selection involves highlighting the same cases across all plots. The user can select some objects in one plot, such as points in a scatterplot (top left), and the corresponding cases are higlighted in all the other plots. Source of the underlying data is the 'mtcars' dataset [@henderson1981].

Despite the fact that the user experience of linked selection is usually fairly intuitive, there are many subtle considerations that go into implementing the feature (for a good overview, see Wilhelm 2008). First, there is the issue of how the user makes the selection. Typically, clicking selects a single objects and clicking-and-dragging selects multiple objects in a rectangular region (similar to how selecting files and folders works on desktop GUIs of most operating systems). In some systems, the users may also drag the selection region around ("brushing"), form a continuous "lasso" selection, select lines in a particular angular range, or points at a particular distance from a centroid (see e.g. Hauser, Ledermann, and Doleisch 2002; Splechtna et al. 2018; G. Wills 2008). Further, when one variables is continuous and the other is derived (such as the x- and y-axes in a histogram), the interaction may also be simplified by restricting selection/brushing to the primary axis (Satyanarayan et al. 2016). Finally, the selections can be combined by various operators such as OR, AND, NOT, and XOR, to form unions, intersections, and other types of logical subsets (Theus 2002; Urbanek and Theus 2003; G. J. Wills 2000; G. Wills 2008).

Second, there is the issue of who should dispatch and respond to selection events. In presentation-focused interactive data visualization and dashboarding systems, this responsibility is kept flexible, such that some plots may only dispatch, only respond, do both, or neither (Satyanarayan et al. 2015, 2016). However, in systems focused on data exploration, the convention is typically for all plots to both dispatch and respond to selection events, such that they may be interacted with in the same way. (Theus 2002; Urbanek and Theus 2003; Urbanek 2011).

Third, there is the issue of what to link. In the case of data represented by a two-dimensional table or data frame, the most common method is to link cases taken on the same observational level (identity linking), such that each row gets assigned a value representing the selection status (Urbanek and Theus 2003; Wilhelm 2008; G. Wills 2008). However, in the case of more complex data, more advanced linking schemes are also available, such as hierarchical and distance-based linking (Wilhelm 2008; Urbanek 2011).

Third, there is the issue of displaying selection. This issue will be touched upon in more detail later, in Section **??**. Briefly, Wilhelm (2008) identifies three methods for displaying linked selection: replacement, overlaying, and repetition. Replacement involves replacing the entire plot with a new graphic; overlaying involves superimposing the objects representing the selected subsets on top of the original objects; and repetition involves displaying the selected objects alongside the original ones. Wilhelm identifies issues with all three techniques, although he does seem to generally come down in favor of repetition (however, see my argument in Section **??**).

A fourth and final issue in linked selection, and arguably one of the core concerns of the present thesis, is consistency. This topic will be coming up again and again, particularly in Section **??**. Consistent and predictable features are a cornerstone of good user interface design (see e.g. Ruiz, Serral, and Snoeck

2021). However, as discussed above, the design an interactive data visualization system supporting linked selection presents many design decisions, each with its own set of implementation constraints. Achieving a consistent user interface through the right combination of these decisions is a known challenge (Urbanek 2011; Pike et al. 2009).

For example, while the approach of allowing objects to independently dispatch and display selection events offers great flexibility, it can also lead to a less intuitive user experience. Put simply, when users select objects in one linked plot by clicking them, they might reasonably expect the same functionality in other plots. If that is not the case (if, for instance, other plots support only displaying but not dispatching selection events), their expectation will be violated. Thus, it might be reasonable to require that all objects can both dispatch and display selection events. However, this places some fundamental constraints on these objects. For instance, how do we draw a lineplot line where only some of the underlying cases are selected? Do we draw a sequence of differently-coloured line segments, leading to a striped "candy cane" pattern (see Figure 3.14)? Do we draw two separate lines? If so, how do we then dispatch selection events on these lines which are already conditional on selection? Like turning over a rock and disturbing a host of creepy-crawlies, linked selection reveals a complex web of visualization design challenges that defy a satisfying, generic solution.

3.3 General data visualization theory

The following sections briefly explore several key theoretical topics in data visualization: the goals and purpose of visualizations, the mechanisms of visual perception, and the theory of scales and measurement. While mainly discussed in the context of static visualization, these topics are equally relevant to interactive visualization and present some unique challenges. My goal is not to give an exhaustive review - each of these topics is substantial enough to serve as a thesis topic in its own right. Instead, I just want to give a brief overview of these topics, highlight some key points, and discuss how they may relate to my own work.

3.3.1 Visualization goals

An important fact about data visualization is that, fundamentally, a chart can be used by many different people for many different things (for a review, see e.g. Brehmer and Munzner 2013; Franconeri et al. 2021; Sarikaya et al. 2018). For example, applied researchers may create figures as part of their workflow, aiming to better understand the data they had collected, spot errors and anomalies, and come up with new ideas and hypotheses (Brehmer and Munzner 2013; see also Kandel et al. 2012). Conversely, data scientists and data analysts in the public and private sector may visualize already familiar data sets to communicate

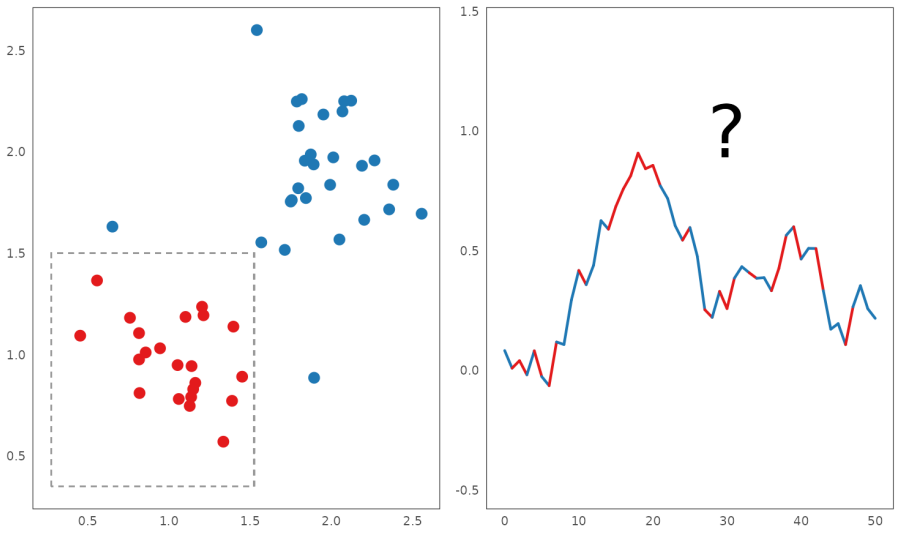


Figure 3.14: Displaying selection is not always trivial. A good example is a lineplot (right). Whereas a point in a scatterplot displays a single case (row) and a bar in a barplot displays a simple subset of cases, a line segment in a lineplot connects two data points. As such, it is not clear whether to highlight the segment *starting* at the selected point, *ending* at the selected point, or, e.g., half a segment on each side of the point. Further, since the geometry of a segmented line is not commutative (row order matters), we end up with a striped 'candy cane' pattern that is not easily interpretable.

important information, drive decisions, or convince or persuade stakeholders (Sarikaya et al. 2018). Finally, some figures may be created out of a sense of curiosity or for pure aesthetic enjoyment (Brehmer and Munzner 2013; Tufte 2001). Depending on the end-goals of the user and the desired target audience, certain visualization techniques, methods, or styles may become more useful than others.

As mentioned in Section 3.2.1, much has been written about the goals and experiences a user might have while creating data visualizations. For instance, Brehmer and Munzner (2013) formalized a typology of abstract visualization tasks, based around three adverbs: *why* is a task is performed, *how* it is performed, and *what* does it pertain to. In the *why* part of their typology, they list the following reasons for why a user may engage in the process of visualizing data: to consume (present, discover, or enjoy), produce, search (lookup, browse, locate, and explore), and query (identify, compare, summarize). As another example, Pike et al. (2009) list the following high-level goals a user might have when interacting with a visualization: explore, analyze, browse, assimilate, triage, asses, understand, compare. And there are many other typologies and taxonomies of data visualization tasks and goals in the literature.

Personally, when it comes to classifying interactive data visualization goals, I prefer the following short list provided by Ward, Grinstein, and Keim (2015):

- Exploration: The user wants to examine a data set
- Confirmation: The user wants to verify a fact or a hypothesis
- Presentation: The user wants to use the visualization to convince or inspire an audience
- Interactive presentation: The user wants to take the audience on a guided tour of key insights

I believe this list maps fairly well onto interactive data visualization systems found in the wild, such as the ones discussed in Section 3.1. Specifically, as mentioned before, in the history of interactive data visualization, the earlier statistical systems seemed to primarily focus on exploration and confirmation, whereas the newer web-based systems seem to prioritize presentation. The interactive presentation category is interesting, since, I would argue, it is far more specific and less common than the other categories, however, by singling it out, Ward et al. make an interesting point. By incorporating time and intentionality, sequential interactive presentations, such as those found in the Graphics section of the New York Times (The New York Times Company 2025), really are quite unique.

3.3.2 Visual perception

Another important research topic in data visualization is visual perception. Specifically, given that we use visual attributes such as position, color, length,

or area to encode various aspects of our data, researchers have tried to answer the question of how to use these attributes in a way that best leverages the human visual system. Fortunately, this research has been quite fruitful, yielding precise and actionable guidelines (for a review, see Franconeri et al. 2021; Quadri and Rosen 2021).

A landmark work in this area has been that of Cleveland and McGill (1984). In this study, the authors conducted a series of empirical experiments in which they investigated people's ability to accurately judge quantities based on different visual encodings. They found that judgments based on position along a common scale were the most accurate, followed by length-based comparisons, and then angle-based comparisons.

The findings were later corroborated by other authors. Heer and Bostock (2010) replicated the Cleveland and McGill (1984) study, and included judgements of circular and rectangular areas which were found to be less accurate than position, length, or angle. Other authors have extended these experiments to other visual encodings, such as color or density (e.g. Demiralp, Bernstein, and Heer 2014; Saket et al. 2017; Reda, Nalawade, and Ansah-Koi 2018). Together, these findings have been used to create rankings of visual encodings, with researchers generally agreeing on the following ordered list: position, length, area, angle, and intensity (from most effective to least, Mackinlay 1986; Franconeri et al. 2021; Quadri and Rosen 2021).

3.3.3 Scales and measurement

Visualizing data involves mapping values to graphical attributes. As discussed in the previous section, certain visual attributes are better for visualizing particular types of data, and vice versa. However, even when we pick an appropriate visual attribute to represent our data with, there are still many choices in how to perform the mapping. For instance, suppose we have some variable x with values $\{1, 2, 3\}$. Should these be treated as magnitudes, a simple ordering, or even just category labels that may be permuted at will? In most data visualization systems, this metadata encoding of values into visual attributes is handled specialized components called scales or coordinate systems, and I will discuss their implementation in detail later, in Section **??**. However, it is first necessary to discuss some theoretical issues involving scales.

A particular challenge when discussing scales in data visualization is that the topic unavoidably intersects with a research area that has a particularly long and contentious history: theory of measurement (see e.g. Hand 1996; Michell 1986; Tal 2025). Theory of measurement (not to be confused with measure theory, with which it nevertheless shares some overlap) is the research area which tries to answer the deceptively simple question: what does it mean to measure something? This seemingly trivial problem has inspired long and fiery debates within the fields of mathematics, philosophy, and social science. Particularly, in

Table 3.2: Types of scales identified by Stevens (1946)

Scale	Structure	Comparison	Valid transformations
Nominal	Isomorphism	Are x and y the same?	$x' = f(x)$, where f is a bijectio
Ordinal	Monotone map	Is x is greater than y?	$x' = f(x)$, where f is a monoto
Interval	Affine transformation	How far is x from y?	$x' = ax + b$, for $a, b \in \math
Ratio	Linear map	How many times is x greater than y?	$x' = ax$, for $a \in \mathbb{R}

psychology, where assigning numerical values non-physical phenomena such as moods and mental states is a central concern, the topic has garnered a significant amount of attention, creating a dense body of research (see e.g. Humphry 2013; Michell 2021).

Arguably, the most influential work in this field has been that of Stevens (1946). In his fairly concise paper, Stevens defined a *scale* as method of assigning numbers to values, and introduced a four-fold classification classification, namely: nominal, ordinal, interval, and ratio scales (see Table 3.2).

The Steven's (1946) typology is based on invariance under transformation. Specifically, for each class of scales, we define a set of transformations that preserve valid comparisons. The set of valid transformations shrinks as we move from one class of scales to another.

For nominal scales, any kind of bijective transformation is valid. Intuitively, we can think of the scale as assigning labels to values, and any kind relabeling is valid, as long as it preserves equality of the underlying values. For instance, given a nominal scale with three values, we can assign the labels $\{\text{red}, \text{green}, \text{blue}\}$ or $\{\text{monday}, \text{tuesday}, \text{wednesday}\}$ in any way we like, as long as each value maps to a unique label. This identifies the underlying mathematical structure as an isomorphism.

Ordinal scales are more restrictive, since, on top of preserving equality, transformations also need to preserve order. For example, if we want to assign the labels $\{\text{monday}, \text{tuesday}, \text{wednesday}\}$ to an ordinal scale with three values, there is only one way to do it that preserves the underlying order: assign the least values to monday, the middle value to tuesday, and the greatest value to wednesday (assuming we order the labels/days in the usual day-of-week order). However, there is no notion of distance between the labels: we could just as well assign the values labels in $\mathbb{N}$ such as $\{10, 20, 30\}$, $\{1, 2, 9999\}$, and so on. Thus, the fundamental mathematical structure is that of a monotone map.

Interval scales need to additionally preserve equality of intervals. This means that, for any three values a, b, and c, if the distances between a and b and b and c are equal, $d(a, b) = d(b, c)$, then so should be the distances between the scaled labels, $d^*(f(a), f(b)) = d^*(f(b), f(c)$. For most real applications, this limits interval scales to the class of affine transformations of the form $f(x) = ax + b$. A canonical example of an interval scale is the conversion formula of degrees Celsius

to Fahrenheit: $f(c) = 9/5 \cdot c + 32$ (Stevens 1946). This example also highlights an important property of interval scales: the zero point can be arbitrary and ratios are not meaningful. Specifically, since the zero points of both Celsius and Fahrenheit scales were chosen based on arbitrary metrics (freezing temperatures of water and brine, respectively), it does not make sense to say that, e.g. 20°C is "twice as hot" as 10°C, in the same way that it does not make sense to say that 2000 CE is "twice as late" as 1000 CE.

Finally, ratio scales also need to preserve the equality of ratios. Specifically, if $a/b = b/c$ then $f(a)/f(b) = f(b)/f(c)$. As a consequence, this also means that the scale must have a well-defined zero-point. Examples of ratio scales include physical magnitudes such as height and weight, which have a well-defined zero point (Stevens 1946).

Steven's (1946) typology sparked a considerable debate, on multiple fronts. First, since the original publication, many authors have sought to either expand upon or criticize Steven's typology. However, despite some monumental efforts towards a unified theory, such as that of Krantz et al. (1971), measurement has remained a hotly debated topic to this day (see e.g. Michell 2021; Tal 2025). Second, more relevant to statistics, some authors such as Stevens (1951) and Luce (1959) used the theory to define come up with prescriptive rules for statistical transformations, suggesting that, for example, taking the mean of an ordinal variable is wrong since the meaning of the average operator is not preserved under monotone transformations. However, this issue was hotly contested by statisticians such as Lord (1953), Tukey (1986), and Velleman and Wilkinson (1993), who argued that many well-established statistical practices, such as rank-based tests and coefficients of variations, rely on such "impermissible" statistics but can nevertheless yield valuable insights. More broadly, these authors also argued that data is not really meaningful on its own, but instead derives its meaning from the statistical questions it is used to answer (see also Wilkinson 2012).

At this point, the discussion around measurement has arguably become far too dense and theoretical, and most data visualization researchers seem to avoid delving into it too deeply (see e.g. Wilkinson 2012). Nevertheless, there are still some areas where the issues of measurement and Steven's typology do crop up. For instance, when scaling area based on a continuous variable, a common recommendation is to start the scale at zero to ensure accurate representations of ratios (see e.g. Wickham and Navarro 2024), aligning with Steven's definition of a ratio scale. Likewise, the long-standing debate around whether the base of a barplot should always start at zero (see e.g. Cleveland 1985; Wilkinson 2012) also carries echoes of the measurement debate. Ultimately, it may yet require long time to settle the issues around measurement, however, there are definitely some ideas within the literature that data visualization can benefit from.

Chapter 4

Challenges

Designing an interactive data visualization system presents a unique set of challenges. Some of these have been already touched on in the Section 3. This section homes in on these inherent challenges, discusses them in greater depth, and begins exploring avenues for possible solutions.

4.1 The structure of this chapter: Data visualization pipeline

When creating visualizations, be they static or interactive, our ultimate goal is to render geometric objects that will represent our data in some way. However, it is rarely the case that we can plot the raw data directly, as is. Instead, before the data can be rendered, it often has to pass through several distinct transformation steps or stages. Together, these steps form a data visualization pipeline (see e.g. **?**; Wickham et al. 2009; **?**). Each of these steps come with its inherent set of considerations and challenges, particularly when interaction is involved.

Take, for instance, the typical barplot. There are several steps to drawing a barplot. First, we have to divide the data into subsets, based on the levels of some categorical variable. Second, we need to summarize or aggregate these subsets by some metric, usually either sum or count. Third, we need to take these summaries and map them to visual encodings, such as x-axis position, y-axis position, and length. Finally, we use these encodings and render the individual bars as rectangles on the computer screen (see e.g. Franconeri et al. 2021).

Thus, the data visualization pipeline can be described by four fundamental steps:

- Partitioning
- Aggregation
- Scaling/encoding
- Rendering

These four steps are common to both static and interactive visualization systems, however, interactivity does introduce some unique challenges. User interaction may affect any of the four stages, and as a result, changes need to be propagated accordingly. Finding a general and efficient solution to this change-propagation remains an open research topic (Wickham et al. 2009; Franconeri et al. 2021). Consequently, discussions of the role interaction within the data visualization pipeline are often fairly vague (see Dimara and Perin 2019; **?**).

This chapter attempts to clarify some of this conceptual ambiguity. Mirroring the structure of the data visualization pipeline, it delves into each of the four steps and explores challenges related to their implementation in interactive systems. The central argument is that interaction is not just a thin veneer that can be layered on top of static graphics; instead, it fundamentally penetrates the abstract machinery of the pipeline. Moreover, for interaction to be predictable, intuitive, and efficient, the components of the pipeline must compose together in specific, well-defined ways, that may be described algebraically using the language of category theory. Mapping out this algebraic composition is crucial for building truly generic and robust interactive data visualization systems (see also **?**; Sievert 2020).

4.2 Partitioning

The first step of any data visualization pipeline is to divide the data into parts or subsets. The justification for this initial step lies in our ultimate goal: to draw one or (usually) more geometric objects (Wilkinson 2012; also known as graphic items, G. Wills 2008), representing some aspects of our data. Thus, before we can do anything else, we need to define the set of data points each geometric object will represent. In the typical case of two-dimensional tables or data frames, this amounts to slicing the table's rows into smaller sub-tables.

The partitioning operation is fairly intuitive for aggregate plots, where each object represents multiple rows of the data. For instance, in a barplot, each bar represents a set of cases corresponding to a category, while in histogram, each bar represents a set of cases in which fall within the same bin along some continuous dimension. However, even one-to-one representations of the data can be viewed this way. For example, in a scatterplot or parallel coordinate plots, each geometric object represents one row of the data, which we can view as its own small table. Similarly, plots with a single geometric object (e.g. density/radar plots) have the underlying set equal to the whole data set.

Thus, the process of splitting our data into subsets is in some way fairly straightforward. However, it does raise two fundamental questions:

- How much of the original data should the subsets contain?
- What should be the relations between the subsets?

While common data visualization practices provide implicit solutions to these questions, explicit formulations are rarely given in the data visualization literature. This lack of conceptual clarity is problematic because how we choose to partition our data is a consequential decision; when we split our data into subsets, we make assumptions, about the data itself as well as the goals of the visualization process. In interactive data visualization particularly, the relations between the parts of our data become of key importance. Therefore, discussing the two questions above in greater depth is essential.

4.2.1 Showing the full data

> "If someone hides data from you, it's probably because he has something to hide." (**?**)

A common recommendation that many data visualization experts provide is that faithful visual representations should show the full data and leave nothing out. The moral behind this recommendation is fairly intuitive. A visualization which hides or obscures information, be it by intent or negligence, cannot be considered a truthful representation of the underlying information (**?**, 2019).

However, data hiding can occur in many different ways. First, the data itself can be cherry-picked or massaged (see e.g. **?**). This is arguably the most egregious case, and can in some cases amount to malicious statistical practices such as HARKing or p-hacking (see e.g. **?**; **?**; **?**). However, even when showing the full data, some visualizations can obscure or downplay certain data features via poor design or incorrect use of visual encodings (**?**, 2019; Cleveland 1985; Ziemkiewicz and Kosara 2009). Finally, there is the issue of missing or incomplete data, where some data cannot be easily represented because it is simply not there.

An infamous example of data hiding leading to disastrous real-world consequences was the 1986 crash of the Space Shuttle Challenger (see **?**). During a pre-launch teleconference, engineers debated the effect of temperature on the performance of O-ring gaskets, as the forecasted temperature was significantly lower than during previous launches. The plot in the left panel of Figure **??** was used to argue that there was no correlation between temperature and O-ring failures. However, this plot had one significant flaw: it excluded launches where no failures occurred. After the disaster, when the data including the zero-failure launches was plotted, it revealed a clear trend of increasing number of failures as temperature decreased (see right panel of Figure **??**, see also **?**).

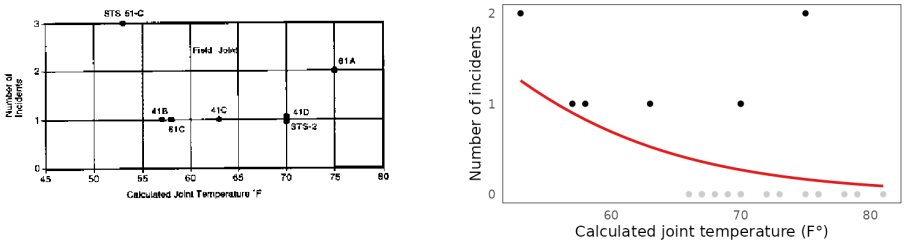


Figure 4.1: Relationship between temperature and the number of O-ring failures within the 1986 Challenger data. Left: the original plot as presented during the pre-launch teleconference. Right: a reproduced plot of the same data, including the original data points (black), the excluded data points with zero failures (grey), and an estimated logistic regression fit (red). The source of right-panel data is @dalal1989.

However, data hiding can also occur in more subtle ways, such as the above-mentioned poor design choices. Consider, for example, axis limits. Cleveland (1985) argues that axis limits should generally be expanded to avoid inadvertently obscuring data near these limits (see also e.g. Chen et al. 2008, 64). Take the following two scatterplots:

In the left scatterplot, the axis limits match the data limits exactly, whereas in the right plot, they are expanded by a small fraction (5%, `ggplot2` default, Wickham 2016). The left scatterplot provides a misleading representation of the underlying trend, as data points near or at the axis limits (top-left and bottom-right corners) are represented by a smaller area, compared to points near the centre of the plot. For instance, the point in the bottom-right corner of the plot lies simultaneously at the limits of the x- and y-axis, and is thus represented by one-quarter of the area of the points in the center.

Finally, there is the issue of data hiding due to missing or incomplete data, which is a bit more complicated. While techniques of visualizing data with missing values do exist (see e.g. Unwin et al. 1996; **?**), they are often tied to specific visualization types and styles, and few general solutions are available. Properly analyzing the patterns of missingness in the data often calls for a full-fledged separate visualization workflow (**?**).

Either way, data hiding is something we should be mindful of. Unless there is a clear and justifiable reason, no data should be arbitrarily removed or discarded, and we should pick good visual representations to represent all of our data faithfully. In the ideal case, the visualization should present a clear and unambiguous mapping between the graphics and the data (Ziemkiewicz and Kosara 2009).

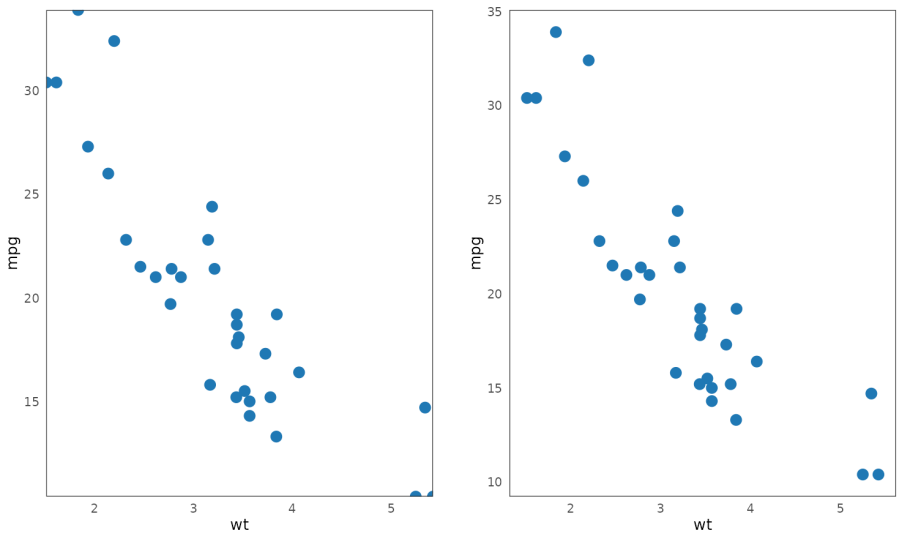


Figure 4.2: Without expanding axis limits, objects at or near the limits become less salient. Left: axis limits match the data limits exactly, and so the points in the top-left and bottom-right corner of the plot are represented by smaller area and the overall trend is distorted. Right: by expanding axis limits, we can ensure that trend is represented faithfully.

4.2.2 Disjointness and comparison

> "To be truthful and revealing, data graphics must bear on the question at the heart of quantitative thinking: 'compared to what'?" (Tufte 2001, 74).

> "Graphics are for comparison - comparison of one kind or another - not for access to individual amounts." (Tukey 1993)

An interesting yet underappreciated fact is that in many common visualization types, geometric objects tend to represent disjoint subsets of the data. That is, in most plots, each point, bar, line, or polygon corresponds a unique set of data points (rows of the data), with no overlap with other objects within the same graphical layer. While different layers can represent the same data (e.g., a smooth fit over a scatterplot, or point clouds over boxplots), objects within the same layer rarely represent the same data. This practice, despite being so common to border on a rule, it is surprisingly seldom discussed.

There are of course counter-examples. For instance, certain visualizations of set-typed data "double up" the contribution of data subsets, such that the same subset of the data may appear in multiple objects (see e.g. **?**, **?**; **?**; **?**). However, these types of visualizations are fairly rare, and represent the exception rather than the norm. When we see a barplot, we typically expect each bar to represent a unique set of cases.

But where does this unconscious "law" of showing disjoint parts of the data come from? I argue that it stems from the fundamental purpose of data visualization: comparison (Tufte 2001; Tukey 1993). When we visualize, we draw our graphics with the ultimate goal of comparing our data along a set of visual channels (**?**; Wilkinson 2012; Franconeri et al. 2021; **?**). This mirrors the comparisons we make about objects and events in the real world. And, in general, it is far easier to reason about objects and events that are independent, rather than ones which overlap or blend together. One example of this comes from basic probability theory, where the sum and product rules have independence as a prerequisite (**?**). Similarly, psychological research, such as the well-known "Linda experiment" (**?**), shows that people struggle with comparing the probability of non-disjoint events. Thus it seems that, in many ways, disjointness presents a more intuitive, "natural" model.

More fundamentally, disjointness may be more intuitive because it reflects a structure which mathematicians have long considered natural: a bijection or one-to-one mapping (see e.g. Fong and Spivak 2019; Lawvere and Schanuel 2009). Specifically, if we take a set, split it into disjoint subsets, and label each subset, then there is a one-to-one correspondence between these subsets and the subset labels (i.e. the parts form an equivalence class). Practically, this means that we can go back and forth between the subsets and the corresponding labels, without losing any information.

The fact that disjoint subsets form a bijection may be particularly useful in data visualization and this may explain its ubiquity, see Figure **??**. For instance, when drawing a barplot, if we divide our data into disjoint subsets and draw one bar corresponding to each part, then we can go back and forth between identifying subsets of the data corresponding to individual bars and vice versa. Thus, the function of identifying data subsets is invertible. In plots where the objects do not represent disjoint subsets, this correspondence is broken: if we select a subset of cases corresponding to a bar, there may be no simple way to identify the original bar from the cases alone. This issue applies both conceptually, when viewing static visualizations, and also more practically, when interacting with interactive visualizations, via features such as linked selection.

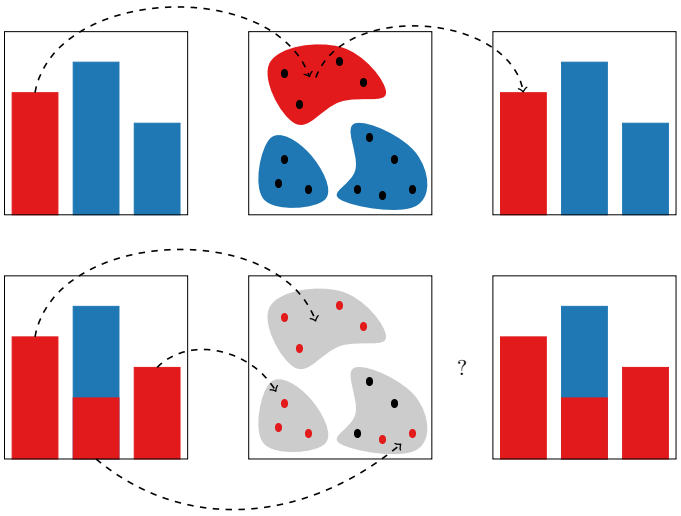


Figure 4.3: Disjointness induces a one-to-one mapping (bijection) between geometric objects and subsets of the data. Suppose we mark out the cases corresponding to the leftmost bar (red). Top row: when each geometric object (bar) represents unique subset of data points, we can easily go back and forth between the object and its underlying subset (middle panel), and so the function of picking cases corresponding to each object is invertible. Bottom row: if there is an overlap between the cases represented by each object, then there may be no way to identify the original object after we have picked out the corresponding cases.

4.2.2.1 Real-world example

To illustrate the idea of disjoint subsets on concrete, real-world example, take the following barplot representing the vote share among the top three parties in the 2023 New Zealand general election (**?**):

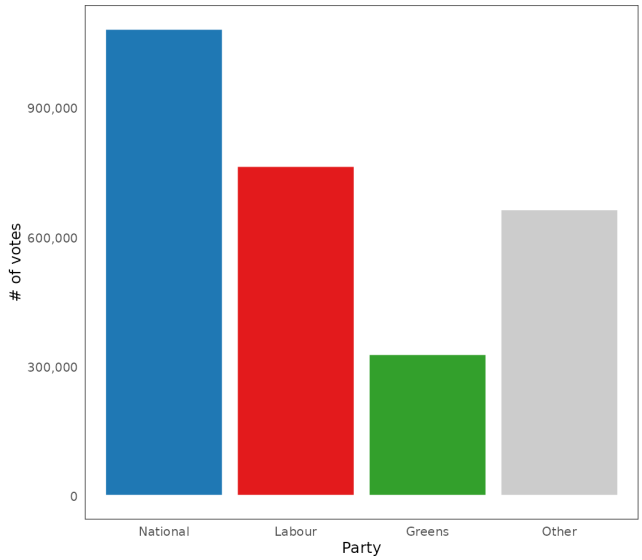


Figure 4.4: Barplot showing disjoint subsets of the data. The bars show the vote share among the top three parties in the 2023 New Zealand general election, with each bar representing a unique subset of voters.

Each bar represents a unique subset of voters and thus the bars show disjoint data. This is the type of data representation that we encounter most often, however, there are few explicit guidelines about this. Hypothetically, we could transform our data, and use the leftmost bar to show, for example, the union of the votes of National and Labour parties:

However, this way of representing the data has several problems. First, this type of visualization is arguably not very useful for addressing our visualization goals. For example, when visualizing election data such as the one above, we typically want to compare the relative number of votes each party received. Figure **??** makes this comparison needlessly difficult. Specifically, since the leftmost bar represents the union of National and Labour votes, we have to perform additional mental calculation if we want to compare the number of votes received by National and Labour directly (Cleveland 1985). Second, we have metadata knowledge (see e.g. Wilkinson 2012; Velleman and Wilkinson 1993) about the data actually being disjoint. We know that, in the New Zealand parliament electoral system, each voter can only vote for a single party. Hence, it does not make sense to arbitrarily combine the data in this way. Finally, Figure **??** also needlessly duplicates information: the number of votes the National party received is counted twice, once in the leftmost bar and again in the second-from-left bar. This goes against the general principle of representing our data parsimoniously (Tufte 2001).

Even when our goal is not to compare absolute counts, there are usually better

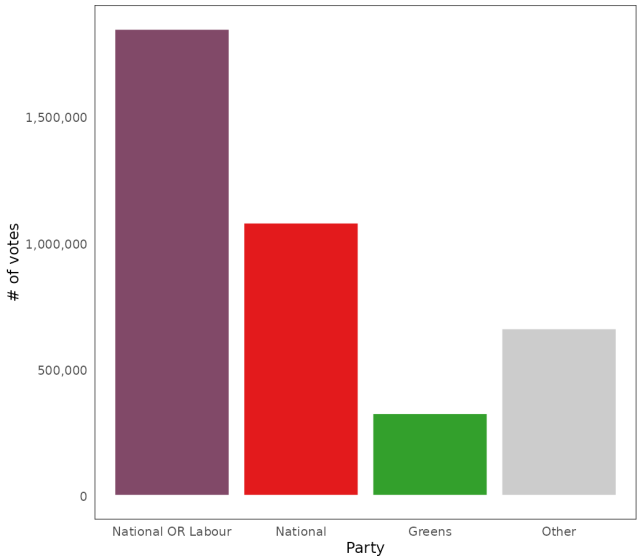


Figure 4.5: Barplot showing non-disjoint subsets of the data. Most of the bars show the same data as in Figure
reffig:geoms-bijection, however, the leftmost bar representing a union of National and Labour voters. The two leftmost bars are thus not disjoint. For a more realistic example, see Figure
reffig:union-geoms2.

disjoint data visualization methods available. For instance, if we were interested in visualizing the *proportion* of votes that each party received, we could instead draw the following plot:

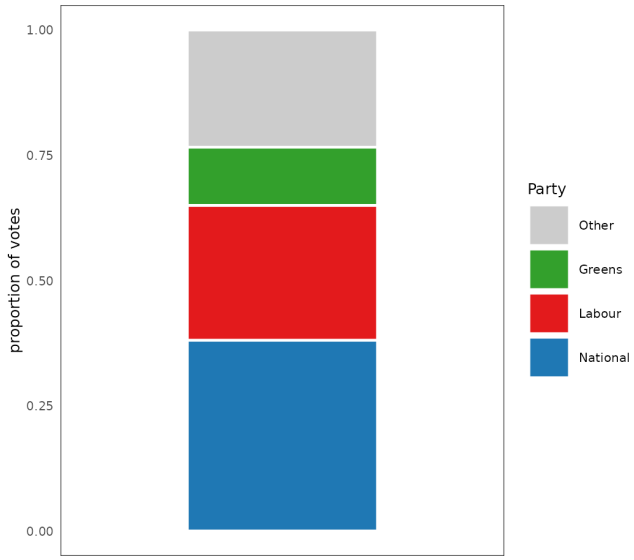


Figure 4.6: Even when proportions are of interest, there are usually disjoint data visualization techniques available. The plot shows proportion of vote share of the top three parties in the 2023 New Zealand general election, with each bar segment again representing a unique subset of voters.

By stacking the bar segments on top of each other as in **??**, we can easily compare proportion of the total number of votes each party received, while retaining a disjoint representation. Each bar segment now again represents a unique subset of voters.

The example above is fairly clear-cut case of where disjoint data representation is the better choice. However, there are also more ambiguous situations, such as when multiple attributes of the data are simultaneously present or absent for each case. Take, for example, the 2020 New Zealand joint referendum on the legalization of euthanasia and cannabis. In this referendum, the two issues were included on the same ballot and voters would vote on them simultaneously. The legalization of euthanasia was accepted by the voters, with 65.1% of votes supporting the decision, whereas the legalization of cannabis was rejected, with 50.7% of voters rejecting the decision (**?**).

We could visualize the referendum data in the following way:

In Figure **??**, both bars include votes cast by the same voter (ignoring the votes where no preference was given for either issue, **?**), making the representation non-disjoint. In this case, the visualization works, since the underlying data is

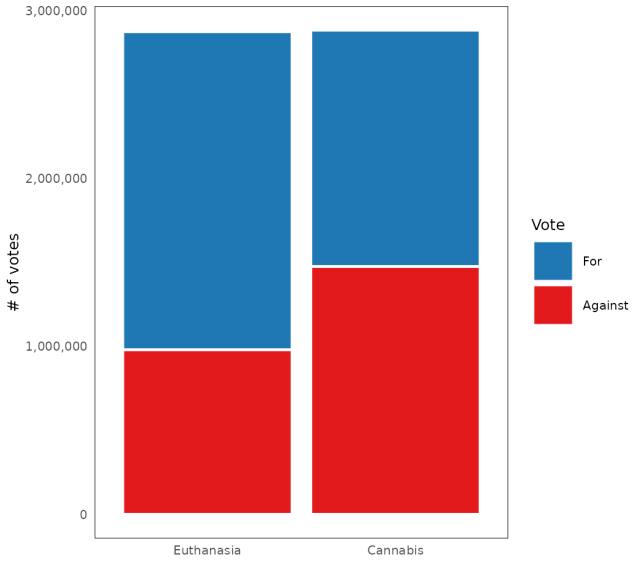


Figure 4.7: Barplot showing a more realistic example of non-disjoint data representation. The bars show the vote share cast by New Zealand voters in the joint 2020 referendum on euthanasia and cannabis. The two bars show (mostly) the same set of ballots, with each single ballot contributing to the height of one segment in each bar.

genuinely non-independent (each person cast two votes). If we had information about individual votes, it might be interesting to see how many people voted for both euthanasia and cannabis, how many voted for euthanasia but against cannabis, and so on. As was mentioned before, these types of visualizations can be useful for set-typed data (see e.g. **?**).

However, even though the data here is fundamentally non-independent, there is often a way to represent it in a disjoint way that preserves most of the desirable properties. Specifically, we can split the data and draw it as separate plots or small multiples (Tufte 2001):

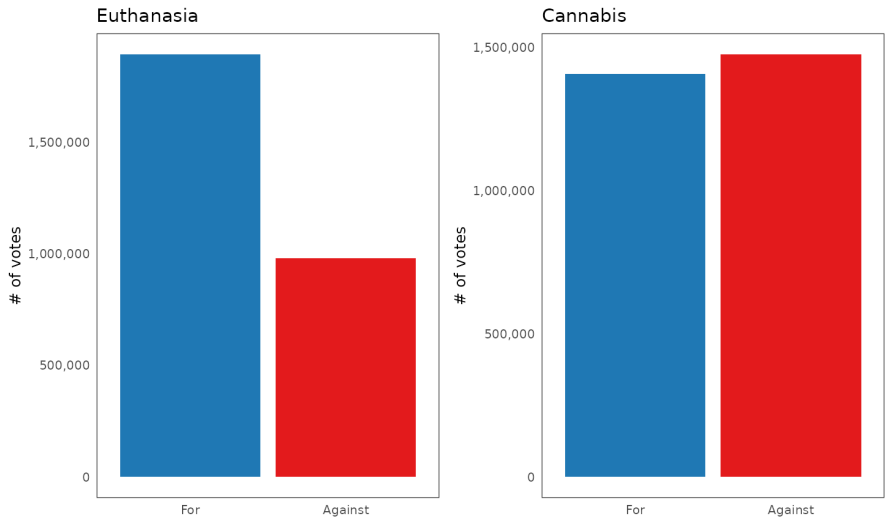


Figure 4.8: Small multiple figure showing the non-disjoint data represented as disjoint bars. The bars again show the vote share cast by New Zealand voters in the joint 2020 referendum on euthanasia and cannabis, however, this time, each bar within one plot represents a unique subset of the cases.

Here again, in Figure **??**, each bar (segment) in each plot represents a disjoint subset of voters.

4.2.2.2 Disjointness and interaction

As I argued above, disjoint subsets offer a simpler mental model for understanding data visualizations. When each geometric object represents a unique set of data points, it becomes easier to reason about the comparisons being made. Conversely, when objects overlap or share underlying data points, additional cognitive effort is required to track the relationships between them.

Further, I argue that disjointness presents a particularly good model for interactive visualization (see also Wilhelm 2008). The natural correspondence between geometric objects and subsets of the data makes certain interactions more intuitive, and conversely, overlapping subsets can produce unexpected or unintuitive behavior. For instance, when a user clicks on a bar in a linked barplot, they might reasonably expect to highlight *that particular bar*, within the active plot, and the corresponding cases within all the other (passive) plots. If they see parts of other bars within the active plot get highlighted as well, they have to spend additional mental effort thinking about the relation between the objects (bars) and the subsets of the data, since this is no longer one-to-one.

Similar issue arises during querying. When a user queries an object that does not represent a disjoint subset of the data, should the returned summary statistics match the object or the (non-disjoint) subset? And how do we signal this to the user? Again, lack of disjointness introduces subtle ambiguities and complicates the interpretation of the presented information.

This does not mean that non-disjoint subsets cannot be usefuly combined with interaction, in specific contexts (see e.g. **?**; Wilhelm 2008). However, I argue that, as a general model, disjointness provides a very good default. Disjoint subsets simplify our mental model, and this may be the reason why some authors discuss interactive features in the context of partitions, which are by definition disjoint (see e.g. Buja, Cook, and Swayne 1996; Keim 2002). Likewise, many common data analytic operations, such as SQL aggregation queries (`GROUP BY`, **?**), operate on disjoint subsets, and this may be another reason why this model is familiar.

4.2.3 Plots as partitions

In the two preceding sections, I have argued that it is generally desirable for plots in our (interactive) data visualization system to have two fundamental features:

- Completeness: They should show the full data
- Distinctness: Geometric objects should represent distinct subsets of data points

These two features actually map onto two fundamental mathematical properties: surjectivity and disjointness. In turn, these two properties define a well-known mathematical structure: a partition. Therefore, partitions offer a compelling model for structuring our plots. I propose the following definition of a *regular plot*:

Definition 4.1 (Regular plot)**.** Regular plot is a plot where the geometric objects within one layer represent a partition of the data, such that there is a bijection between these objects and (possibly aggregated) subsets of the original data.

Note that this definition still allows for plots where geometric objects in different layers represent overlapping data subsets, such as boxplots with overlaid points, or scatterplots with a smooth fit.

I propose regular plots as a fundamental building block of our interactive data visualization system. By building our interactive figures out of regular plots (as small multiples, Tufte 2001), we can ensure that the resulting visualization will be easily interpretable, even when combined with interactive features such as linking and querying.

4.2.3.1 Bijection on cases vs. bijection on subsets

Although I have not been able to find references conceptualizing plots as partitions in the same general way as I do here, some data visualization researchers have used the language of bijections when discussing graphics. For example, Dastani (**?**) discusses plots as bijections (homomorphisms) between data tables and visual attribute tables. Similarly, Ziemkiewicz and Kosara (2009), and Vickers, Faith, and Rossiter (2012) argue that, in order to be visually unambiguous, plots should represent bijections of the underlying data. Essentially, these researchers argue that plots should represent bijective mappings of the data tables, such that each object represents one row of the data.

However, this "one-row-one-object" model sidesteps the issue of aggregation (see also Section **??**). It operates on the assumption that the data is pre-aggregated, such that, for instance, when we draw a barplot or a histogram, we start with a table that has one row per bar. This is rarely the case in practice. Most visualization systems incorporate aggregation as an explicit component of the data visualization pipeline (see e.g. **?**; Wickham 2016; Satyanarayan et al. 2015, 2016; **?**). Howeever, acknowledging aggregation presents a problem for the one-row-one-object model, since aggregation is, by definition, not injective. Once we aggregate multiple data points into a summary or a set of summaries, we cannot recover the original cases (see also **?**). Thus, the model proposed by authors such as Dastani (**?**), Ziemkiewicz and Kosara (2009), and Vickers, Faith, and Rossiter (2012) would exclude many common aggregation-based types of plots, such as barplots and histograms. Ziemkiewicz and Kosara (2009) indeed do acknowledge that this is a problem, and admit that, at times, aggregation can be an acceptable trade-off, despite the inherent information loss.

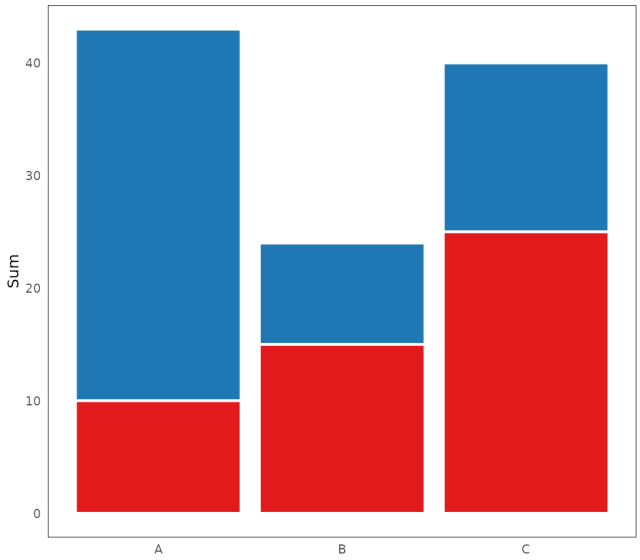
However, if we instead model plots as bijection between *parts of data* and the geometric objects, rather than between individual *data points* and geometric objects, aggregation ceases to be a problem. Even after we aggregate a multiple-row subset of the data into a single row summary, this bijection is preserved. Thus, aggregation can be considered a part of the bijection. For instance, if we split our data into ten tables and aggregate each table, we are still left with ten tables of one row each that we can map bijectively to geometric objects.

	group	selection	value
1	A	1	12
2	A	1	21
3	A	2	10
4	B	1	9
5	B	2	15
6	C	1	15
7	C	2	12
8	C	2	13

4.2.3.2 Products of partitions

Many types of plots involve data that has partitioned or split across multiple dimensions. This is especially true in interactive data visualization, where features such as linking automatically induce another level of partitioning (Wilhelm 2008). This necessitates a general mechanism for combining partitions into products.

The concept of "product of partitions" may be best illustrated with code examples. Suppose we want to draw the following barplot:



We start with the following data, which includes a categorical variable (`group`, plotted along the x-axis), a variable representing selection status (`selection`, used to colour the bar segments), and a continuous variable that we want to summarize (`value`):