"Principled and Efficient Interactive Data Visualization" by Bartonicek, Adam for a PhD in Statistics

Student ID: 828803059
Email: abar435@aucklanduni.ac.nz
Department of Statistics

Supervisor: Dr. Simon Urbanek
Co-supervisor: Dr. Paul Murrell
Advisory Committee: Drs B. Efron, D. R. Cox, K. Pearson.

Date of enrolment in the programme and expected date of completion

April 12, 2023

This document represents the student's research proposal after one year of provisional PhD registration. Confirmed PhD registration is now sought.

1 Introduction

Humans learn about the world around them by interacting with it. Further, as major part of our cerebral cortex is devoted to visual processing, we learn best by interacting with things we can see. The same applies to data. If we want to learn from our data effectively, we need practical and reliable tools for visualizing data and interacting with it. As such, interactive data visualization is an important pursuit in statistics and data science, and this is evidenced by its rising popularity: currently, many interactive data visualization libraries exists across popular data-science-adjacent programming languages (such R, Python, Julia, and Javascript) and their ecosystems, including D3 (Bostock et al., 2011), plotly (Plotly Inc., 2023), Highcharts (Highcharts Team, 2023), and Vega (Satyanarayan et al., 2015) and Vega-lite (Satyanarayan et al., 2016).

Yet, many of these present-day interactive data visualization libraries libraries tend to suffer from a common set of drawbacks. Specifically, due to historical and other reasons, the libraries tend to fall into two camps:

1. Specialized, high-level suites of "off-the-shelf" interactive visualizations
2. Generic, highly customizable low-level frameworks.

While the libraries from the first camp are usually approachable to users with less programming experience, their major drawback is that they are not extendable. Conversely, libraries from the second camp lend the user great deal of power and flexibility, however, to use them effectively, the users are required to have significant programming experience and deep knowledge of the specific API, especially when it comes to setting up complex interaction between multiple plots. Further, the users are responsible for making sure that the interaction is coherent - there is nothing stopping them from creating figures with interactive features that are meaningless, confusing, or unpredictable. Finally, the design philosophy behind many of these libraries seems to be oriented towards and used for data presentation rather than data exploration (Batch and Elmqvist, 2017). As such, within the interactive data visualization sphere, there is currently a lack of a mid-level, multi-plot interaction system geared towards data exploration.

Crucially, the abovementioned absence of a mid-level interactive data visualization system is not just an implementation gap. Rather, it seems to be a symptom of an underlying lack of a strong theoretical foundation. Specifically, there do not seem to be any guidelines for how such a mid-level system should operate - what criteria should interactive plots meet so that they can be composed together and behave in predictable and consistent ways. As a consequence, the two options left to the creators of interactive data visualizations systems are either: 1) give the user a limited number of ready-made solutions that are known to behave well, or 2) put the onus of making sure that the interaction is coherent onto the user - which precisely correspond to the two interactive data visualization camps outlined above.

The goal of the proposed project is to lay down the theoretical foundations for such a mid-level interactive data exploration system, as well as implement it within a modern programming language. Specifically, at the theoretical level, I will map out the boundaries and constraints that parts of the system - underlying datastructures, plots, graphical primitives, etc... - need to meet in order to satisfy certain criteria of coherency and consistency, such that the user's interactions with the visualization are meaningful, understandable, and predictable. At the applied level, I will implement the system in **JavaScript** and provide a high-level interface in **R**. Finally, the broader teleological goal of the project is to create a tool that applied statisticians and data scientists can use to understand their data in a convenient, principled, and efficient way.

2 Background

2.1 What Event Counts as *Interactive* Data Visualization?

It may seem surprising, but despite the widespread popularity of interactive data visualizations, there seems to be little consensus as to what actually makes a data visualization interactive. The term gets used by different researchers in vastly different contexts, sometimes in arguably conflicting ways. As such, for the purposes of the present text, it is important to disambiguate what is meant by "interactive data visualization".

Firstly, there is the issue of whether "interactive data visualization" refers to an object or system, or to an action, undertaken by a human being. Pike et al. (2009) note that "interaction" is an overloaded term that can refer to either the concrete tools which users use to manipulate visual information or to the more abstract "human interaction with information" - the back-and-forth between the user and the visual information presented to them (see also Yi et al., 2007). The more abstract action definition is more often emphasized in the field of Human Computer Interaction (see e.g. Sinha et al., 2010). For the purpose of this text, the object/system definition will be used - interactive data visualizations are concrete objects that are produced by (typically computer-based) systems/pipelines that take in raw data and produce images that can be interpreted and manipulated by humans (Brodbeck et al., 2009).

2.1.1 Simple Interactivity

But even after narrowing down the focus to interactive data visualizations as objects, a lot of conceptual ambiguity remains. Some researchers use a **simple** definition and define interactive visualizations as any visualizations that can be actively manipulated by the user (Brodbeck et al., 2009). Other researchers emphasize time or the **temporal** aspect, with visualizations being "interactive" when there is little lag between the user's input and changes to the visualization (Becker and Cleveland, 1987; Buja et al., 1996). Complicating the matters futher, some even make the distinction between "interactive" and "dynamic" manipulation, where interactive manipulation happens discretely, such as when pressing a button or selecting an item from a drop-down menu, whereas dynamic manipulation happens continuously, for example when smoothly moving a slider or by clicking-and-dragging (Rheingans, 2002; Jankun-Kelly et al., 2007). These tentative definitions present relatively little restriction on what counts as interactive visualization: for example, one could argue that the process of a user typing code into a command line to generate new plots could be considered interactive visualization, as long as it happens fast enough.

2.1.2 Complex Interactivity

Other researchers do not seem to be satisfied with these broad definitions. For many, the defining feature of interactive data visualization is the ability to **query** different parts of the dataset (by e.g. zooming, panning, and filtering), and the reactive propagation of changes between connected or **"linked"** parts of interactive figures (Kehrer et al., 2012; Buja et al., 1996; Keim, 2002; Unwin, 1999). Similarly, in Visual Analytics (VA) research, a distinction is made between "surface-level" (or "low-level") interactions, which manipulate attributes of the visual domain only (e.g. zooming and panning), and **"parametric"** (or "high-level") interactions, which manipulate attributes of mathematical models or algorithms underlying the visualization (Leman et al., 2013; Pike et al., 2009).

There are other ways that the term "interactive data visualization" has been used (for more detailed taxonomies, see Yi et al., 2007), however, short summaries of the key definitions mentioned above are presented in Table 1 and will be referred to later on in the text. What is important is that the different types of interactivity imply very different levels of programming complexity. For example, simply changing the color of a point or a bar in a plot, irrespective of anything else, might be implemented by changing an attribute of the underlying graphical primitive only - the point/bar does not need to know about what data it represents. However, if the change happens in response to (linked) brushing within a different plot, then there does need to be some way of tracking which cases of the data belong to the primitive. Likewise, changing the width of a histogram bar does not affect the graphical attributes of the bar only, but the underlying operation (binning) needs to be recomputed with respect to the new value of the parameter.

The conceptual ambiguity about what gets called "interactive" visualization matters because it leads to radically different implementations in software packages (see Sections 2.2 and 2.3). For example, the **R Graph Gallery** page on **Interactive Charts** (Holtz, 2022) features several examples of interactive visualizations, however, none of them meet the linked and parametric definitions of interactivity outlined in Table 1. Interactive data visualization systems that exist within the open source data visualization ecosystem differ significantly in the amount of features and flexibility they offer, as well as in how much responsibility or "house-keeping" for maintaining the interactive state they offload onto the user.

2.2 Brief History of Interactive Data Visualization in Statistics

2.2.1 Static Visualization Goes Digital

Static data visualization has a rich and intricate history (see e.g. Dix and Ellis, 1998; Chen et al., 2008; Friendly and Wainer, 2021; Young et al., 2011). Briefly, for a long time, it was considered at best an auxiliary field, however, at the end of 1950's, a series of developments lead to a great increase in its prominence. Firstly, at the theoretical level, the work of Tukey (1962; 1977) and Bertin (1967) established data visualization as valuable discipline in its own right. Secondly, at the applied level, the development of personal computers (see e.g. Abbate, 1999) and high-level programming languages, most notably FORTRAN in 1954 (Backus, 1978), made production of figures easy and accessible to the wider public. Combined, these developments lead to a surge in the use and dissemination of data visualizations.

Table 1: Definitions of Interactive Data Visualization

Name	Short Definition	Details
Simple	Change happens	User can manipulate the visualization in some way
Temporal	Change happens in real time	There is little lag between the user's input and changes to the visualization
Querying	Change results from subsetting	The user can query different parts of the dataset, interaction is analogous to subsetting rows of the data (e.g. zooming, panning, and filtering)
Linked	Change propagates	Parts of the visualization are connected or "linked", such that interaction with one part produces a change in another (e.g. linked brushing)
Parametric	Change reflects an underlying model	The user can manipulate the parameters of some underlying model (e.g. rotating principal axes in a PCA scatterplot, changing the width of a histogram bar)
(Cognitive)	Change is caused and perceived by a human	The user engages in a back-and-forth with the visual information presented to them

Final development in the field of static data visualization that is important to mention was the Grammar of Graphics introduced by Leland Wilkinson (2012). Prior to Wilkinson's work, data visualization systems tended to come in two flavors: low-level ones, in which the users had to create visualizations from scratch using graphical primitives, and high-level ones, in which the users could select from a limited range of ready-made visualization types. Wilkinson, building upon the work of Bertin and Tukey, developed theory for a mid-level visualization system - Grammar of Graphics - which allows the users to specify a broad range of statistical graphics by declaratively combining abstract plot attributes such as aesthetics, scales, coordinates, and geometric objects (Wilkinson, 2012). Grammar of Graphics has been successfully implemented in several software packages, most notably the popular **ggplot2** R package (Wickham, 2010) and the proprietary software **Tableau** (Tableau Team, 2023).

2.2.2 Birth of Interactive Visualization

As static visualization entered the computer age, interactive data visualization would not be left far behind. Early systems appeared in the 1960's and 1970's, and tended to be specialized for one specific task. For example, Fowlkes (1969) used interactive visualization to show how probability densities reacted to change of parameters and transformations, and Kruskal (1965) used interactive visualization to showcase his multi-dimensional scaling algorithm (a way of embedding objects within a common space based on pairwise distance measurements). The first "general-purpose" system was **PRIM-9** (Fisherkeller et al., 1974), which allowed for exploration of high-dimensional data in scatterplots using projection, rotation, subsetting and masking. Later systems grew on to become even more general and ambitious. For example, **MacSpin** (Donoho et al., 1988) and **XGobi** (Swayne et al., 1998) provided features such as interactive scaling, rotation, linked selection (or "brushing"), and interactive plotting of smooth fits in scatterplots, as well as interactive parallel coordinate plots and grand tours.

Following the turn of the 21st century, interactive data visualization systems saw even scope and flexibility and also began to be integrated into general-purpose statistical computing software. The successor system to XGobi, **GGobi** (Swayne et al., 2003) was made to be directly embeddable in R. Java-based **Mondrian** (Theus, 2002) allowed for sophisticated linked interaction between many different types of plots including scatteplots, histograms, barplots, scatterplot, mosaic plots, parallel coordinates plots, and maps. Finally, **iPlots** (Urbanek and Theus, 2003) implemented a general framework for interactive plotting that was not only embedded in R but could be directly programmatically manipulated, and was later further expanded and made performant for big data in **iPlots eXtreme** (Urbanek, 2011).

2.2.3 Common Features of Statistical Systems

The statistics-based interactive data visualization systems came in various forms, however, they generally tended to support features such as multiple ready-made plot-types, interaction between multiple plots with shared underlying data and state, and interactive manipulation of model parameters. They tended to be oriented towards scientific audience, with data exploration as the primary goal. Finally, they tended to be made to be directly embedabble and interoperable with general-purpose statistical computing software.

2.3 The Web and Current Age Interactive Data Visualization

2.3.1 Web-native Interactivity

The developments in interactive data visualization within the field of Statistics were paralleled by those within Computer Science. Most notably, the rise of Web technologies in the mid 1990's and the appearance of JavaScript in 1995 as a high-level general-purpose programming language for the Web (for a description of the history, see e.g. Wirfs-Brock and Eich, 2020), created an extremely versatile platform for highly-reactive and portable applications. JavaScript was created with the explicit purpose of making the Web interactive, and the fast dissemination of standardized web-browsers meant that online interactive applications could be accessed by anyone, from anywhere. Interactive data visualization became just one of many technologies highly sought after within the fledgling Web ecosystem.

The very early systems such as **Prefuse** in 2005 (Heer et al., 2005) and **Flare** in 2008 (Burleson, 2020) relied on plugins (Java and Adobe Flash Player, respectively). However, in the early 2010's, several true Web-native JavaScript-based interactive data visualization systems emerged. The most prominent among these is **D3**.js (Bostock et al., 2011). D3 is a broad and general framework for manipulating HTML documents using data and displaying it with visualizations. It is in popular use still to this day and has spawned a number of specialized, higher-level data visualization libraries that abstract away the details and provide an interface to commonly used statistical plots, such as the also very prominent **plotly**.js (Plotly Inc., 2022). Importantly, in the D3 model, D3's engine only renders graphics: interactivity is left the user. If a user wants to create a visualization that's interactive, she has to write JavaScript functions that take care of updating the underlying parameters. A radically different approach altogether was taken by **Vega** (Satyanarayan et al., 2015). In Vega, all aspects of a visualization including interactivity are specified declaratively, as a Plain Old JavaScript Object (POJO). As such, Vega carries a large number of reactive primitives. Similar to D3 and plotly, Vega also spawned its own higher-level libraries in **Vega-lite** (Satyanarayan et al., 2016), and **Altair** (VanderPlas et al., 2018). A final popular interactive data visualization library in JavaScript is **Highcharts** (Highcharts Team, 2023). Similar to Vega-lite, Highcharts is a high-level declarative framework in which plots are created based on a POJO specification object.

2.3.2 Common Features of Web-based Systems

These contemporary web-based interactive data visualization systems offer a great deal of expressiveness, however, it does seem to come at a cost. Specifically, the low-level frameworks like D3 and Vega allow the users to create almost arbitrarily complex interactive figures, and even the higher-level frameworks like plotly, Vega-lite, Altair, and Highcharts still offer a great deal of customizability. Yet, a lot of code and programmer time is required to create complex interactive figures, even in these higher-level frameworks. As a result, most examples that show up on the web or on the libraries' own showcase pages typically feature only shallow interactivity within a single plot - such as zooming, panning, and pop-up labels - and examples of more complex multi-plot interaction such as linked brushing or cross-filtering are far less common. Further, the coherence of interactivity is left to the user: there is nothing stopping them from creating figures with meaningless interactive features. As such, it is hard to call the frameworks like Vega-lite, Altair, and plotly true mid-level systems, the way that ggplot2 can be called a mid-level system for static graphics.

The target audience of the web-based systems is also very different from that of the statistical systems (discussed in Section 2.2). Most real-world use examples come from online news articles and business/government dashboards, with considerably fewer appearing in scientific outlets. Likewise, the focus seems to be more on data presentation rather than data exploration - communicating findings once they have been found rather than discovering them in the first place. This seems to make sense given the design of these systems - it is worth it to create complex interactive visualizations when there is the guarantee that many people will see the result; however, conversely, it may not be economical to do so with $n = 1$ (i.e. when the user is the only person who will see the visualization). Data presentation and data exploration are both very important pursuits, however, it does seem that the market for interactive data visualizations for EDA is currently underserved, and this may explain why seemingly few data scientists use interactive visualizations to explore their data (Batch and Elmqvist, 2017).

2.4 Specialization vs. Generality

To re-state the common thread from the previous sections, over the last 30 years or so, interactive data visualization has undergone a divergent evolution within two largely independent branches: statistical and web-based/computer science. Different goals and features were prioritized within each branch. The statistical branch focused on creating specialized systems for scientific data exploration. Users could easily create complex interactive figures by picking from a limited range of pre-made, "out-of-the-box" plots that were designed to behave coherently and consistently when composed together. Conversely, in the web-based branch, greater focus was put on generality and data presentation. Users were given a great deal of power and flexibility to create arbitrarily complex interactive figures from scratch, however, the onus for ensuring the interactivity was coherent and consistent was put on them.

The difference between the two branches represents a fundamental tension between specialization and generality. Specialized systems are easy to use but hard to extend; generic systems are extensible by definition but require time and effort to use effectively.

2.5 The Problem of Statistical Summaries

Every data visualization has at least one graphical primitive or geometric object that is used to represent some statistical summary of the data. This is true for both static and interactive visualizations. For example, the position of a point on a scatterplot represents the values of the variables on the x- and y-axes. Typically, the statistical summary used in a scatterplot is identity, meaning that the x- and y-position of the point represents the raw values of those variables. As another example, the height of a bar in a histogram conventionally represents the number of cases within a binned range of the x-axis variable.

In static data visualizations, we are free to compute any kind of statistical summary we like. However, this does not translate to interactive visualizations. Instead, interactive visualizations are subject to a unique set of constraints and challenges.

2.5.1 Computational Cost

First of, there is the problem of computing resources. In static visualizations, we only need to compute any summary once, before we render the plot. However, this may not the case in interactive visualizations. Specifically, if the the summaries can be affected by the user's input, we can no longer just "fire-and-forget" but instead our system needs to be able to be able to recompute the summaries reactively, on-the-fly. For example, if the width of the bins in an interactive histogram changes, we need to recompute the number of cases within each bin. This incurs an additional computational cost. If the statistical summary is too computationally expensive, the volume of data is too high, or if the user's input changes too rapidly, it may not be possible to render the interaction smoothly enough. To clarify, this problem only arises when the interaction needs to refer back to the original data (i.e. when the interaction is linked, querying, or parametric, as described in Section 2.1); no extra cost is incurred by e.g. interactively changing the opacity of graphical primitives, irrespective of the data. However, for the more complex types of interaction, the necessity to recompute summaries reactively can create computational bottlenecks.

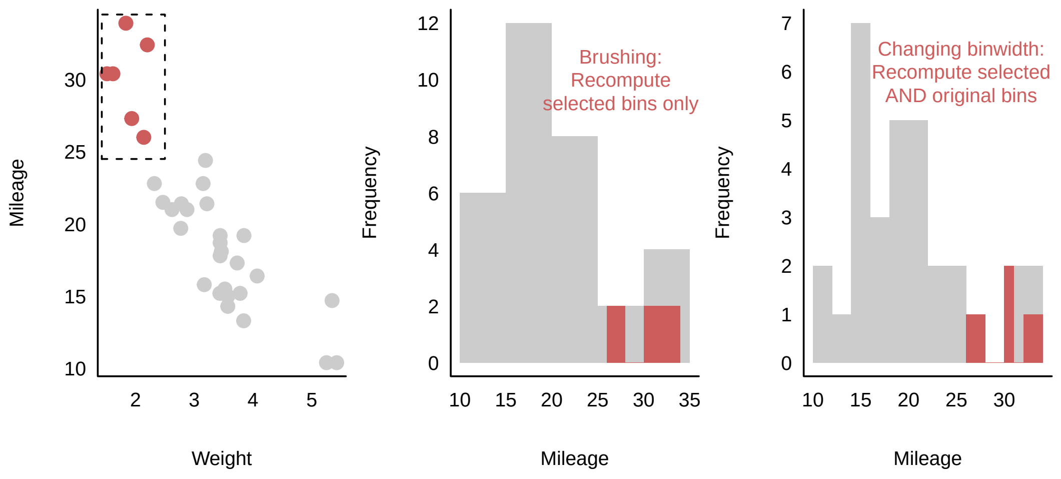

Figure 1: Reactive computations cascade

Importantly, these reactive computations can cascade and form hierarchies. Using the example of the interactive histogram above, if the histogram also responds to linked brushing, we have reactivity occuring on two different levels, as shown in Figure 1. Firstly, when linked brushing occurs, we only need to recompute the number of cases within each bin belonging to the selection. However, if the width of the bins changes, we need to recompute both the number of cases within each bin belonging to the selection as well as the number of cases within the original, unselected bins.

2.5.2 Visual and Computational Coherency

Perhaps more importantly, different types of interaction also place a limit on what summaries will be computationally and visually coherent. For example, suppose we have an interactive visualization that consists of a linked barplot and a scatterplot. Further, let's suppose that the barplot displays the means of some continuous variable, within the levels of the variable on the x-axis (this type of plot is also sometimes called ”dynamite plot”, when error bars are shown). We want the user to be able to perform linked brushing on the two plots in a reciprocal way - clicking-and-dragging to select points in the scatterplot should highlight parts of the corresponding bar or bars, and vice versa, selecting a bar should highghlight the corresponding points.

We immediately run into several problems. The first is shown in Figure 3: how do we draw an empty selection? In a barplot of sums or counts, 0 is a meaningful default value, since the sum or count of a set with no elements is zero. However, the mean of an empty set is not defined. We could simply not draw the bar, but this will decohere the statistical summary from the visual representation: the absence of a bar may signal that either no cases are selected and the mean is undefined, or that there are selected cases and their mean is equal to the lower y-axis limit.

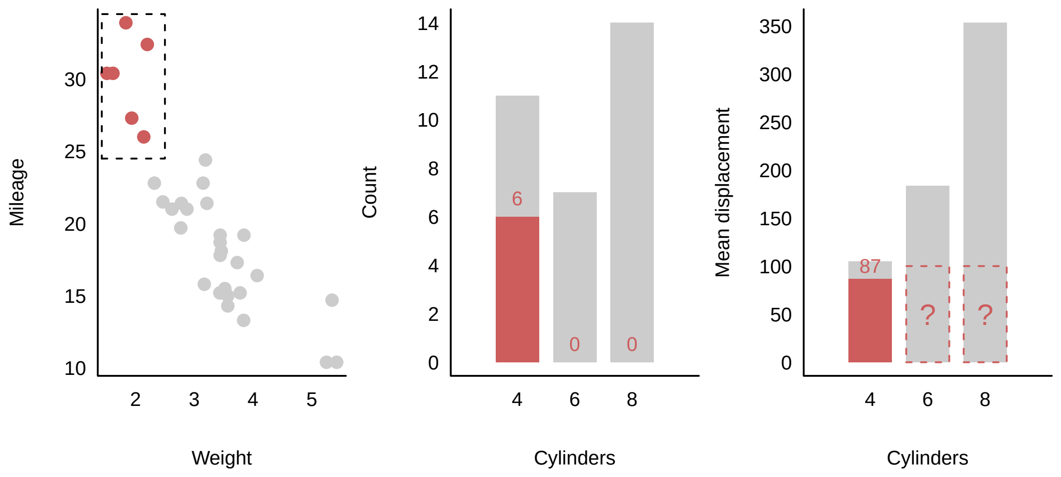

Figure 2: Sum/count of empty selection is zero but the mean of an empty selection is undefined

Second, if there are multiple selections present within a single x-axis variable level, how do we draw them? In a sum or count bar, we can stack selected groups on top of each other, and the total height of the stacked bar is equal to the sum of the heights of the sub-bars. This is not the case for mean. The mean of the group means does not have to equal the grand mean, and there is no idiosyncratic way of visually combining multiple means together. We could draw the means side-by-side as separate bars (a technique called ”dodging” in `ggplot2`), however, this complicates the two-way reciprocal nature of the linked brushing - the user can no longer simply select the single unambiguous stacked bar, but instead has to learn by experience whether they can select one of the dodged group bars individually or whether they have to select all bars jointly, or some combination of both.

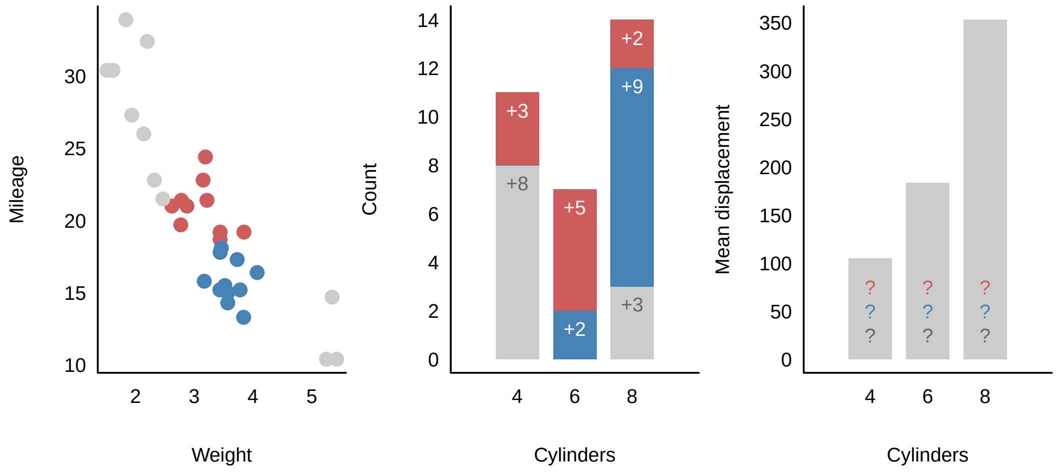

Figure 3: Sums/counts can be stacked but means cannot

Finally, and thirdly, when computing the mean, we have to keep track of two quantities: running sum and running count. If we wanted to combine two means, we would need to have access to both, not just the computed values. In contrast, for sum and count we need to keep track of one value only and the values can be combined without reference to anything else. This is not a big problem with mean and the computation can still be easily parallelized, but other types of summaries may be more tricky.

2.5.3 Need for Structure

The picture that emerges is that, when it comes to interactive data visualization, some types of statistical summaries are better than others. Specifically, when more complex types of interactions are desired, such as reciprocal two-way linked brushing, not every kind of statistical summary will do. It is important to map out what types of constraints should statistical summaries meet in order to "work" and lead to visually and computationally coherent visualizations. Only by imposing some kind of structure can we lay the foundations for a true mid-level interactive data visualization system.

2.6 Few Relevant Bits of (Applied) Category Theory

Category theory is a branch of mathematics concerned with the study of universal structures and relations. It was developed as a way of unifying historically distinct areas of math such as abstract algebra, topology, and linear algebra. At the present time, it is also gaining traction in more applied fields, especially computer science and functional programming. Even a very basic complete treatment would be far outside the scope of the present proposal, however, few relevant pieces will be presented in a greatly simplified way, introducing examples from programming when relevant. The treatment here follows mainly from Fong and Spivak (2019) and Milewski (2018), as well as Leinster (2014).

2.6.1 Functions

Functions are the most fundamental building block of category theory (as well as many other areas of mathematics). In their more general form, they are also known as "morphisms" or "mappings". A function can be described in the following way: given two sets, the set of sources S and the set of targets T, a function is a subset $F \subseteq S \times T$ containing all source-target pairs (s, t), such that for all

$s \in S$ there exists a unique a $t \in T$. The sets S and T are also known as the domain and codomain, respectively. In programming terms, we should be hypothetically able to implement any function as a lookup table (in practice, this is only feasible when the number of possible arguments is finite and small - but then it can be handy a technique, called *memoization*).

A function which covers all of its codomain, i.e. one for which, for all $t \in T$, there exists an $s \in S$ such that $f(s) = t$ is called *surjective* or *onto*. A function which has a unique element in the codomain for every element in the domain, i.e. one for which for all $s_1, s_2 \in S$ and $t \in T$, if $f(s_1) = t$ and $f(s_2) = t$, $s_1 = s_2$, is called *injective* or *one-to-one*. Also, for any given subset $T_i \subseteq T$, we can define *pre-image* as the subset of S that maps into T_i: $f^{-1}(T_i) = \{s \in S : f(s) \in T_i\}$. Finally, functions can be composed: if we have two functions $f : X \to Y$ and $g : Y \to Z$, we can combine them into a new function $h = g \circ f$ such that $h : X \to Z$, i.e. $h(x) = g(f(x))$.

2.6.2 Partitions

We can do many things with functions, and one fundamental operation we can do with them is to form *partitions*. Specifically, given a set A and a set of part labels P, we can use a surjective function $f : A \to P$ to assign every element in A a part label in P. Further, if we then take one of the part labels $p \in P$ and pull it back through f to its pre-image $f^{-1}(p) \subseteq A$, we will then recover the corresponding part $A_p \subseteq A$. We can use this to define partitions in another way, without reference to f: a partition of A consists of a set of part labels P, such that, for all $p \in P$, there is a non-empty subset $A_p \subseteq A$ and:

$$A = \bigcup_{p \in P} A_p \qquad \text{and} \qquad \text{if } p \neq q, A_p \cap A_q = \varnothing \qquad (\forall p, q \in P)$$

I.e. the parts jointly cover the entirety of A and there is no overlap between their elements. Further, if we have two sets of part labels P and P', and if for every $p \in P$ there exists a $p' \in P'$ such that $A_p = A_{p'}$, the partitions are in some way fundamentally the same. The labels $p \in P$ and $p' \in P'$ may be superficially different, but the two structures encode the same information. This phenomenon is known as *isomorphism*, and will be expounded on in more detail in Section 2.6.5.

2.6.3 Preorders

Another class of fundamental structures in category theory are preordered sets or *preorders*. A preorder consists of a set X and a binary relation on X, often denoted $\leq$, such that:

1. $x \leq x$ (reflexivity)
2. if $x \leq y$ and $y \leq z$ then $x \leq z$ (transitivity)

Further, if we have one additional property:

3. If $x \leq y$ and $y \leq x$, then $x = y$ (anti-symmetry)

Then we can speak of a partially ordered set, or *poset*. Finally, with one more property:

4. Either $x \leq y$ or $y \leq x$ (comparability)

We can speak of a *total order*. Examples of total orders (which are also by definition posets and preorders) include the set of natural numbers $\mathbb{N}$, reals $\mathbb{R}$, booleans $\mathbb{B} = \{\text{True, False}\}$, etc... An example of a poset which is not a total order is a phylogenetic tree: a species *Homo sapiens* can be thought of as being in a subordinate relationship ($\leq$) to the genus *Homo* and the great ape family, but in no comparable relationship to the species *Canis familiaris*.

2.6.4 Monoids

Monoids are a simple yet surprisingly useful structure in category theory. Specifically, a monoid is a tuple $(M, e, \otimes)$ consisting of:

1. An object (set) M
2. A "neutral" or "empty" element $e \in M$ called the *monoidal unit*
3. A binary function $\otimes : M \times M \to M$ called the *monoidal product*

Such that, for all $m, m_1, m_2, m_3 \in M$:

a. $e \otimes m = m \otimes e = m$ (unitality)

b. $(m_1 \otimes m_2) \otimes m_3 = m_1 \otimes (m_2 \otimes m_3) = m_1 \otimes m_2 \otimes m_3$ (associativity)

In plain English, monoids encapsulate the idea that *the whole is exactly the sum of its parts* (where "sum" can be any arbitrary function that fulfills the properties). Examples of monoids include addition of natural numbers $(\mathbb{N}, 0, +)$, multiplication of reals $(\mathbb{R}, 1, *)$, and matrix multiplication $(\mathbf{Matrix}, \mathbf{I}, \cdot)$, where $\mathbf{I}$ is the identity matrix and the $\cdot$ infix operator is typically omitted by convention. As counterexample, exponentiation is not a monoid, because because it is not associative $(x^{(y^z)} \neq (x^y)^z)$.

Another important aside is that the set M does not need to involve any numerical quantities. For example, given any set S, the set union $(S, \varnothing, \cup)$ and set intersection $(S, S, \cap)$ are also monoids. String concatenation (e.g. the `paste()` function in R) is also a perfectly valid monoid:

- `paste("hello", "") == paste("", "hello") == "hello"`
- `paste("hello", paste("world", "bye")) == paste(paste("hello", "world"), "bye")`

Monoids can be further specialized by imposing additional properties. For example, with:

c. $x \otimes y = y \otimes x$ (commutativity/symmetry)

We can make it so that the direction of applying the monoidal product does not matter (which would still admit regular addition, multiplication, set union, and set intersection, but not matrix multiplication and string concatenation). Additionally, if M is a preorder, we can impose an additional constraint:

d. If $x_1 \leq x_2$ and $y_1 \leq y_2$, $x_1 \otimes y_1 \leq x_2 \otimes y_2$ (monotonicity)

If all four properties a), b), c), and d) hold, we speak of a *symmetric monoidal preorder*.

In programming, monoids are most typically encountered in the context of arrays, whereby the elements of an array can be iteratively absorbed into a single value, through functions/methods called conventionally `reduce`, `fold`, or `accumulate`. Examples include the `Reduce()` function in R, `Array.reduce()` method in JavaScript, `fold` function in Haskell. Importantly, while `reduce/fold` are most often used with arrays, their generic forms are polymorphic and can be applied to arbitrary datastructures that implement the interface e.g. trees (Braithwaite, 2019). One handy, "free" fact about monoidal operations is that they can be readily parallelized - since the end-result of applying the operation recursively to parts is the same as applying it once to the whole, we are free to split the data across multiple clusters or machines.

2.6.5 Categories

A monoid is actually a special case of a much more general concept called *category*. A category $\mathcal{C}$ consists of:

1. A collection of elements $\mathrm{Ob}(\mathcal{C})$ called the *objects* of $\mathcal{C}$
2. For every pair of objects $a, b \in \mathrm{Ob}(\mathcal{C})$, a set $\mathcal{C}(a, b)$ of *morphisms* from a to b ($f \in \mathcal{C}(a, b)$ can also be denoted $f : a \to b$)
3. For every object a, one morphism $\mathrm{id}_a : a \to a$ called the *identity morphism*
4. For every three objects a, b, c and every two morphisms $f : a \to b$ and $g : b \to c$, a *composite morphism* $h : a \to c = g \circ f$

Such that:

a. For any morphism $f : a \to b$, $f \circ id_a = f = id_b \circ f$ (unitality)

b. For any four objects $a_1, a_2, a_3, a_4 \in \mathrm{Ob}(\mathcal{C})$ and morphisms $f : a_1 \to a_2$, $g : a_2 \to a_3$, and $h : a_3 \to a_4$, $h \circ (g \circ f) = (g \circ h) \circ f = g \circ h \circ f$ (associativity)

As was mentioned in Section 2.6.1, morphisms can be thought of as generalizations of functions: they take one object within a category to another object within the same category, and their composition is associative, just like with regular functions. However, unlike in regular functions, the objects in the category are left unspecified and do not have form elements of any set. The identity morphism guarantees there is always at least one way to start and arrive at the same object, and its composition with other morphisms has no effect.

Returning to monoids, these can recast as a category with one object with a single non-identity morphism going from the object back to itself. As a concrete example, given the monoid $(\mathbb{N}, 0, +)$, we can represent this as a category with a single object z and a morphism f (and identity morphism id_a). The identity morphism will then play the role of the monoidal unit $e = id_a$, function composition will play the role of the monoidal product, and the number of times we compose the function will determine the element of the set: $\{0, 1, 2, \ldots\} = \{id_a, f, (f \circ f), (f \circ f \circ f), \ldots\}$. The associativity and unitality properties of monoids will be fulfilled as a result of the same properties being defined on categories.

3 So What's New?

As was laid out in the previous section, when compared to traditional static visualization, interactive data visualization presents an idiosyncratic set of challenges and considerations.

This project seeks to develop a theory for a mid-level interactive data exploration system and fill its unoccupied niche within the interactive data visualization sphere. To accomplish this, the following goals will be actioned:

1. Map out boundaries and constraints that a mid-level interactive data visualization system should meet
   (a) Describe what contracts do graphical primitives and their underlying statistical summaries need to uphold in order to be interactable with in a coherent and predictable way
2. Develop a theoretical foundation for the system

   (a) Use concepts and syntax of category theory to describe the necessary structure
   (b) Consider links back to implementation via functional programming

3. Implement the system in JavaScript and publish as an `npm` package

   (a) Use plain ES2020 as much as possible, possibly incorporate a reactive programming library and functional programming utility library
   (b) Use the HTML `canvas` element as a drawing utility, explore using WebGL and other frameworks for performance
   (c) Provide a full documentation for any functions, classes, etc...

4. Implement a high-level interface to the system in R and publish on `CRAN`

   (a) Provide a full documentation for all functions
   (b) Provide a package vignette
   (c) Pass all `CRAN` checks

5. Publish an article in a international peer-reviewed journal such as the *Journal of Statistical Software* or the *R journal*

6. Present the system at an R developer or data visualization conference such as useR! or IEEE VIS

4 Data and Special Needs

The use of the system will be tested out on datasets available in the public domain such as the `diamonds` dataset in R. For these data sets, no ethics or disclosure statements are necessary. Possible collaboration with working scientists and corresponding data access may be negotiated at a later date.

The student's own personal computer should be sufficient to produce the developing and testing the system. Should extra computing resources become necessary, the student should be able to inquire and be granted access to the **Ihaka server** (`ihaka.stat.auckland.ac.nz.`) using SSH connection. Other options such as the New Zealand eScience Infrastructure (**NeSI**) should be available too.

The packages produced as part of the project will be published for free as open-source software, under the MIT license (The Open Source Initiative, 2023). No additional ethics approvals are necessary.

5 Budget

Development of open-source software is not very budget-intensive. The work related to the project will be funded by the University of Auckland Doctoral Scholarship. Expenses related to conferences (registration fees, travel) may be funded via the Postgraduate Research Student Support (PReSS) account, with an annual allocation of $1200 NZD.

6 Objectives and Goals

The objective(s) of this research project are to...

List *specific* goals/tasks that will be undertaken as part of the proposed research.

1. I wish to derive the first three moments of the Slash distribution. This will involve working out the characteristic function first.

    a. Then work out the observed information matrix based on ?.

    b. Then work out the expected information matrix (?).

    c. Apply the EM algorithm (?) to estimate λ.

    d. Write a R package to implement my method. It will be written in S4 and use object oriented methods (?).

    e. Apply the method to the radiation data set (?).

    f. Extend biplots (?) for my data.

    g. Release the package on CRAN (cf. ?).

    h. Publish my results in at least two papers, earmarked for JASA and JRSS-B. Also an applications paper in Biometrics. See Section 7.

2. Solve the Fisher-Behrens problem (???).

7 Deliverables and Program Schedule

Table 2: Timeline for my thesis. Itemize the list of deliverables with specific dates so that you can make concerted effort to achieve them. Here are *some* activities—fill in more.

Date	Activity
2018-04-01	Provisional PhD registration (PhD in Statistics).
2018-05-01	(Optional) Updated my personal webpage at www.stat.auckland.ac.nz/∼myStudentName.
2018-05-20	Attended one of the Doctoral Skills Programme's Induction Days.
2018-09-01	Gave my first talk at NZSA conference. Won first prize.
2018-11-01	Gave a talk to PhD Talks Day.
2019-01-15	Submit my first paper to *Annals of Statistics* (co-authored with supervisor).
2019-05-01	Presented my research progress to a departmental seminar.
2020-01-20	Achieve Goal 1.
2020-08-20	Achieve Goal 2.
2020-10-23	Submit my second paper to *Annals of Applied Statistics* (co-authored with supervisor).
2020-01-15	Submit my third paper to *Biometrics* (sole authorship).
2020-02-01	Submit my PhD thesis.

I have fulfulled all my first year requirements. These are:

1. All AFA provisional goals:

    (a) *Full thesis proposal normally completed within 6 months*: Here it is!

(b) *Completion of one substantial piece of written work within 12 months*: Wrote 90 percent of Chapter 1 of my thesis, submitted one journal article, wrote 50 percent of Chapter 2 of my thesis.

(c) *Presentation of research progress to a departmental seminar*: 2019-05-01.

(d) *Approval of the full thesis proposal by the appropriate departmental/faculty postgraduate committee*: Being done.

(e) *Ethics approval(s)/permissions obtained for the research (if required)*: Not necessary.

(f) *Attendance at one of the Doctoral Skills Programme's Induction Days*: 2018-05-20.

2. Although optional, it is a good idea to try update a personal webpage (ideally www.stat.auckland.ac.nz/∼ giving details of my thesis, links to other research resources in my topic, and some personal stuff to make it interesting.

3. I have diligently attend as many Statistics Department seminars as I could. They are given in Table 3. This is much more than the minimum quota set by the department. Consequently there should be no problem due to this when getting my annual report signed off[1]. The 2018-10-08 seminar was particularly useful because it gave me an idea on how to solve one of my problems.

4. I did STATS 730 in Semester 1 of 2018 and obtained an A+.

5. I did STATS 782 in Semester 2 of 2018 and obtained an A.

Table 3: Departmental seminars I have attended (top part of the table). Talks from another UoA department are in the middle tier. Conferences and workshops are in the bottom tier, e.g., 2019-05-29 event was an all-day workshop. **Note**: I filled in the required document and submitted it within the required time period after each seminar I attended.

Date	Speaker	Title
2018-04-08	David Brillinger	Random trajectories, some theory and applications
2018-02-30	David Cox	Frequentist statistics as a theory of inductive inference
2018-10-08	David Matthews	Estimating diagnostic test likelihood ratios
yyyy-mm-dd	Speaker Name	Title of Talk
yyyy-mm-dd	Speaker Name	Title of Talk
yyyy-mm-dd	Speaker Name	Title of Talk
yyyy-mm-dd	Speaker Name	Title of Talk
yyyy-mm-dd	Speaker Name	Title of Talk
yyyy-mm-dd	Speaker Name	Title of Talk
yyyy-mm-dd	Speaker Name	Title of Talk
yyyy-mm-dd	Speaker Name	Title of Talk
yyyy-mm-dd	Speaker Name	Title of Talk
yyyy-mm-dd	Speaker Name	Title of Talk
2018-06-16	James B. Conant	"Geography is not a university subject" (geo Department)
yyyy-mm-dd	Speaker Name	Title of Talk
2019-05-29	David Siegmund	Workshop on 'Genetic Mapping' at UoA.

[1] If insufficient seminars have been attended then sign off will occur *after* the minimum number is reached.

Appendix A

Delete or replace the contents of this section with any appendices you may have.

The *University of Auckland Statute and Guidelines for the Degree of Doctor of Philosophy (PhD)* (2016) reads[2]:

"The PhD degree is awarded for a formal and systematic exposition of a coherent programme of advanced research work. The work is carried out over the period of enrolment for the degree, and in the opinion of the examiners and the Board of Graduate Studies, satisfies all of the following criteria:

(i) is an original contribution to knowledge or understanding in its field, and

(ii) meets internationally recognised standards for such work, and

(iii) demonstrates knowledge of the literature relevant to the subject and the field or fields to which the subject belongs, and the ability to exercise critical and analytical judgement of it, and

(iv) is satisfactory in its methodology, in the quality and coherence of its expression, and in its scholarly presentation and format."

References

Abbate, J., Sep. 1999. Getting small: a short history of the personal computer. Proc. IEEE 87 (9), 1695–1698.

Backus, J., 1978. The history of fortran i, ii, and iii. ACM Sigplan Notices 13 (8), 165–180.

Batch, A., Elmqvist, N., 2017. The interactive visualization gap in initial exploratory data analysis. IEEE transactions on visualization and computer graphics 24 (1), 278–287.

Becker, R. A., Cleveland, W. S., 1987. Brushing scatterplots. Technometrics 29 (2), 127–142.

Bertin, J., 1967. Sémiologie Graphique: Les diagrammes, les réseaux, les cartes. Gauthier-Villars.

Bostock, M., Ogievetsky, V., Heer, J., 2011. D^3 data-driven documents. IEEE transactions on visualization and computer graphics 17 (12), 2301–2309.

Braithwaite, R., 2019. Javascript Allongé. Leanpub.

Brodbeck, D., Mazza, R., Lalanne, D., 2009. Interactive Visualization - A Survey. In: Human Machine Interaction. Springer, Berlin, Germany, pp. 27–46.

Buja, A., Cook, D., Swayne, D. F., 1996. Interactive high-dimensional data visualization. Journal of computational and graphical statistics 5 (1), 78–99.

Burleson, T., Nov. 2020. Flare | Data Visualization for the Web. [Online; accessed 18. Oct. 2022]. URL `https://blokt.com/tool/prefuse-flare`

[2]Regulation 1, Preamble, item (e).

Chen, C.-h., Härdle, W., Unwin, A., Friendly, M., 2008. A brief history of data visualization. Handbook of data visualization , 15–56.

Dix, A., Ellis, G., May 1998. Starting simple: adding value to static visualisation through simple interaction. In: AVI '98: Proceedings of the working conference on Advanced visual interfaces. Association for Computing Machinery, New York, NY, USA, pp. 124–134.

Donoho, A. W., Donoho, D. L., Gasko, M., 1988. Macspin: Dynamic graphics on a desktop computer. IEEE Computer Graphics and applications 8 (4), 51–58.

Fisherkeller, M. A., Friedman, J. H., Tukey, J. W., 1974. An interactive multidimensional data display and analysis system. Tech. rep., SLAC National Accelerator Lab., Menlo Park, CA (United States).

Fong, B., Spivak, D. I., 2019. An invitation to applied category theory: seven sketches in compositionality. Cambridge University Press.

Fowlkes, E., 1969. User's manual for a system fo active probability plotting on graphic-2. Tech-nical Memorandum, AT&T Bell Labs, Murray Hill, NJ .

Friendly, M., Wainer, H., 2021. A history of data visualization and graphic communication. Harvard University Press.

Heer, J., Card, S. K., Landay, J. A., Apr. 2005. prefuse: a toolkit for interactive information visualization. In: CHI '05: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. Association for Computing Machinery, New York, NY, USA, pp. 421–430.

Highcharts Team, Mar. 2023. Interactive javascript charts library. [Online; accessed 15. Mar. 2023]. URL `https://www.highcharts.com`

Holtz, Y., Aug. 2022. Interactive charts | the R Graph Gallery. [Online; accessed 22. Aug. 2022]. URL `https://r-graph-gallery.com/interactive-charts.html`

Jankun-Kelly, T., Ma, K.-L., Gertz, M., 2007. A model and framework for visualization exploration. IEEE Transactions on Visualization and Computer Graphics 13 (2), 357–369.

Kehrer, J., Boubela, R. N., Filzmoser, P., Piringer, H., 2012. A generic model for the integration of interactive visualization and statistical computing using r. In: 2012 IEEE Conference on Visual Analytics Science and Technology (VAST). IEEE, pp. 233–234.

Keim, D. A., 2002. Information visualization and visual data mining. IEEE transactions on Visualization and Computer Graphics 8 (1), 1–8.

Kruskal, J. B., Oct. 1965. Multidimensional Scaling. [Online; accessed 13. Oct. 2022]. URL `https://community.amstat.org/jointscsg-section/media/videos`

Leinster, T., 2014. Basic category theory. Vol. 143. Cambridge University Press.

Leman, S. C., House, L., Maiti, D., Endert, A., North, C., 2013. Visual to parametric interaction (v2pi). PloS one 8 (3), e50474.

Milewski, B., 2018. Category theory for programmers. Blurb.

Pike, W. A., Stasko, J., Chang, R., O'connell, T. A., 2009. The science of interaction. Information visualization 8 (4), 263–274.

Plotly Inc., Aug. 2022. Part 4. Interactive Graphing and Crossfiltering | Dash for Python Documentation | Plotly. [Online; accessed 22. Aug. 2022].
URL `https://dash.plotly.com/interactive-graphing`

Plotly Inc., Mar. 2023. Plotly: Low-Code Data App Development. [Online; accessed 15. Mar. 2023].
URL `https://plotly.com`

Rheingans, P., 2002. Are we there yet? exploring with dynamic visualization. IEEE Computer Graphics and Applications 22 (1), 6–10.

Satyanarayan, A., Moritz, D., Wongsuphasawat, K., Heer, J., 2016. Vega-lite: A grammar of interactive graphics. IEEE transactions on visualization and computer graphics 23 (1), 341–350.

Satyanarayan, A., Russell, R., Hoffswell, J., Heer, J., 2015. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. IEEE transactions on visualization and computer graphics 22 (1), 659–668.

Sinha, G., Shahi, R., Shankar, M., 2010. Human computer interaction. In: 2010 3rd International Conference on Emerging Trends in Engineering and Technology. IEEE, pp. 1–4.

Swayne, D. F., Cook, D., Buja, A., Mar. 1998. XGobi: Interactive Dynamic Data Visualization in the X Window System. J. Comput. Graph. Stat. 7 (1), 113–130.

Swayne, D. F., Lang, D. T., Buja, A., Cook, D., Aug. 2003. GGobi: evolving from XGobi into an extensible framework for interactive data visualization. Comput. Statist. Data Anal. 43 (4), 423–444.

Tableau Team, Mar. 2023. Tableau: Business Intelligence and Analytics Software. [Online; accessed 16. Mar. 2023].
URL `https://www.tableau.com`

The Open Source Initiative, Feb. 2023. The MIT License. [Online; accessed 29. Mar. 2023].
URL `https://opensource.org/license/mit`

Theus, M., Nov. 2002. Interactive Data Visualization using Mondrian. J. Stat. Soft. 7, 1–9.

Tukey, J. W., 1962. The future of data analysis. The annals of mathematical statistics 33 (1), 1–67.

Tukey, J. W., et al., 1977. Exploratory data analysis. Vol. 2. Reading, MA.

Unwin, A., Mar. 1999. Requirements for interactive graphics software for exploratory data analysis. Comput. Statist. 14 (1), 7–22.

Urbanek, S., 2011. iplots extreme: next-generation interactive graphics design and implementation of modern interactive graphics. Computational Statistics 26 (3), 381–393.

Urbanek, S., Theus, M., 2003. iplots: high interaction graphics for r. In: Proceedings of the 3rd International Workshop on Distributed Statistical Computing. Citeseer.

VanderPlas, J., Granger, B., Heer, J., Moritz, D., Wongsuphasawat, K., Satyanarayan, A., Lees, E., Timofeev, I., Welsh, B., Sievert, S., 2018. Altair: interactive statistical visualizations for python. Journal of open source software 3 (32), 1057.

Wickham, H., 2010. A layered grammar of graphics. Journal of Computational and Graphical Statistics 19 (1), 3–28.

Wilkinson, L., 2012. The grammar of graphics. Springer.

Wirfs-Brock, A., Eich, B., Jun. 2020. JavaScript: the first 20 years. Proc. ACM Program. Lang. 4 (HOPL), 1–189.

Yi, J. S., ah Kang, Y., Stasko, J., Jacko, J. A., 2007. Toward a deeper understanding of the role of interaction in information visualization. IEEE transactions on visualization and computer graphics 13 (6), 1224–1231.

Young, F. W., Valero-Mora, P. M., Friendly, M., 2011. Visual statistics: seeing data with dynamic interactive graphics. John Wiley & Sons.