Towards Fluent Interactive Data Visualization

Adam Bartonicek

2023-11-07

Contents

1 Introduction

Interactive data visualization (IDV) has seen a rapid growth in popularity over the last few decades. From the humble roots of early, highly specialized systems, such as those of Fowlkes (1969) and Kruskal (1965), there has been a steady progress towards more and more general and feature-rich frameworks. The first general-purpose system was *PRIM-9* (Fisherkeller, Friedman, and Tukey 1974), which allowed for exploration of high-dimensional data in scatterplots using projection, rotation, subsetting and masking. Later systems, such as *MacSpin* (Donoho, Donoho, and Gasko 1988), *Lisp-Stat* and *XLisp-Stat* (L. Tierney 1989, 2004), and *XGobi* (Swayne, Cook, and Buja 1998) provided rich 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. They were later followed by other systems such as *Mondrian* (Theus 2002), *GGobi* (Swayne et al. 2003), *iPlots* (Urbanek and Theus 2003), and *cranvas* (Xie, Hofmann, and Cheng 2014). Alongside these later developments, which have largely come from the field of statistics, the rise of Web technologies and interactive web apps has spawned its own family of IDV systems. Among these, the earlier systems such as *Prefuse* (Heer, Card, and Landay 2005) and *Flare* (Burleson 2020) relied on external plugins (Java and Adobe Flash Player, respectively) while later systems became truly Web-native by embracing JavaScript. Among them, *D3*.js (Bostock, Ogievetsky, and Heer 2011) has grown to great popularity, alongside its high-level interface *plotly*.js, (Plotly Inc. 2022), and so have other frameworks such as *Vega* (Satyanarayan et al. 2015), *Vega-lite* (Satyanarayan et al. 2016), *Altair* (Vanderplas et al. 2018), and *Highcharts* (Highcharts Team 2023).

Interactive figures now frequently appear in online news articles, business dashboards, and scientific publications and blogs. Yet, despite the wide proliferation of IDV systems and various taxonomies of interactive features (see e.g. Yi et al. 2007), there still looms a large unresolved problem in the production of IDVs. That is, interactive data visualizations, like all visualizations, require us to process data into statistical summaries that can then be translated into graphical attributes and drawn. However, in contrast to static visualizations, where all data processing needs to occur only once, IDVs need to respond to user input reactively, with relevant quantities often having to be recomputed many times a second (Wickham et al. 2009). Further, different types of interactions come with different levels of computational complexity (see e.g. Leman et al. 2013; Pike et al. 2009). Thus, the way we process the data and implement the interactions matters a lot. Some interactions can be implemented by manipulating graphical attributes of the visualization only. For example, to implement panning or zooming, all we need to do is compute the axis limits once. After that, we can "forget" the original data and merely update the four scalar values (= lower and upper x-axis and y-axis limits) whenever the user performs the requisite actions. However, this is not the case for other, more complex types of interactions. For example, if we want to implement an interactive histogram in which the user can change the binwidth and anchor, we need a way to recompute the number of cases in each bin after either of the two parameters is updated. This means that we have to refer to the original data. Similarly, to implement linked highlighting/brushing, whereby the user can highlight some cases in the data across multiple plots by either clicking or click-and-drag selecting the corresponding geometric objects in one plot, we need to keep track of which cases belong to which object.

As such, there have been calls for a general interactive data visualization pipeline or "plumbing" (Wickham et al. 2009) that would allow us to reason about the production of IDVs in the same way that for example the *grammar of graphics* (Wilkinson 2012) has. However, these have, despite some attempts (see e.g. Xie, Hofmann, and Cheng 2014; Crossfilter Organization 2023), been left largely unaddressed.

2 The illusion of objects

The key towards a general IDV framework may lie in a subtle yet profound question that inevitably appears when one tries to produce interactive data visualizations: *when we interact with a plot, what exactly are we interacting with?* On its face, it may seem trivial. A person clicking a bar in an interactive barplot may be convinced that they are interacting with the coloured rectangle on the screen, since, by design, that is the salient "thing" they see change in front of them. And in some way, this is true - by interacting with the bar, we can affect its graphical attributes: we can change its colour, we can squeeze it/stretch it, and so on. Yet, in another, deeper way, this perception of interacting with a plain geometric object is just an illusion. How so?

The illusion lies in the fact that the bar is not just a geometric object - the coloured rectangle it is represented by. Instead, the rectangle is only ever meaningful as a "bar" within the context of the plot. We can see this easily - if we were to take the coloured rectangle outside of the plot, we would lose some crucial information that the rest of the plot provides.

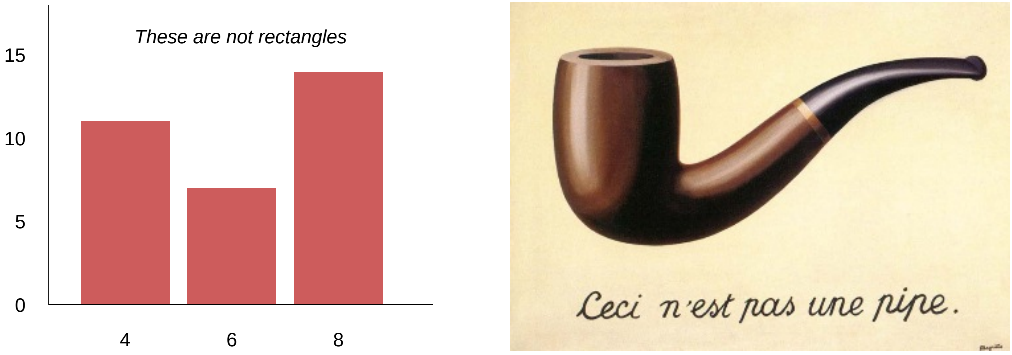

Figure 1: A rectangle is not a bar in a similar way that a painting of a pipe is not a pipe

Thus, objects in a plot are imbued with some additional information or structure, beyond their simple geometries. That should not seem surprising or controversial to people familiar with data visualization. Indeed, for example, geometric objects or `geoms` are considered just one attribute of a plot in the *grammar of graphics* (Wilkinson 2012). However, it may be more challenging to define in detail what exactly this "structure" is. There are a few ideas we may be able to muster. First of all, we know that the geometric objects in plots are supposed to represent some underlying data. That much is clear - if the objects in a graphic do not represent any external data but are instead drawn according to some arbitrary rules, we cannot really, in good conscience, call the resulting graphic a "plot". But data is only a part of the story.

When drawing plots, we rarely represent the raw data directly. Instead, we often summarize, aggregate, or transform. We do this by applying mathematical functions such as count, sum, mean, log, or the quantile function. And it is the output of these transformations that we then represent by the geometric objects.

So, when interacting with a bar in an interactive barplot, we do not just interact with a plain geometric object. Instead, we interact with a mathematical function, or, in fact, several of them. This is very important since functions have properties, and these properties impose limits on what kinds of visualizations and interactions we can meaningfully compose. This is the core argument of the present text. Before diving deeper, however, let's first define some key terms and draw a rough sketch of the data visualization process as a whole.

3 Rough sketch of the data visualization process

To create a data visualization, be it static or interactive, we need several ingredients: data, summaries, scales/coordinate systems, and geometric objects. These should be familiar to most users of interactive data visualization systems. However, it may still be useful to lay them out in order, and examine the specific features and quirks of each one of them.

First of all, every data visualization needs to be built on top of some underlying data. We can represent this as a set of some arbitrary units of information (or "datapoints") D. Data in the wild usually comes with more structure than that - for example, we often encounter data stored in a tabular (or "tidy," Wickham 2014) format, stratified by rows and columns. In that case, we could represent D as the set of rows R, the set of columns C, or the set of cell values $R \times C$ (where $\times$ indicates the cartesian product). However, for the purpose of this general description, we do not have to assume any special structure and just speak of n datapoints $d \in D$.

Second, at some point during the visualization process, we need to transform the datapoints D into a set of collections of summaries S via some function α. This summarizing function can come many different forms. It may be the case that α is one-to-one (bijection), in which case there is one summary for every unit of data (and vice versa). This is the case, for example, in the typical scatterplot, in which α is just the identity function (every unit of data/row gets assigned one point). However, more often, α is many-to-one (surjection), which means that multiple units of data may be merged into a single summary. Examples of this include the typical barplot, histogram, density plot, or violin plot. When α is many-to-one, it will typically reduce the cardinality of the data, such that $k = |S| \leq |D| = n$ (e.g. in a typical barplot, there are fewer bars than there are rows of data, unless there is a unique level of the categorical variable in each row). Further, to turn the n units of data into k collections of summaries, we need to somehow stratify the data on one or more variables. These variables may either come from the data directly (i.e. the variables used are "factors", as in the case of a barplot or a treemap) or may themselves be a summary of the data (as in the case of histogram bins). Importantly also, each collection of summaries $s \in S$ may (and usually will) hold multiple values, produced by a different constituent function each. For example, the collection of summaries s for a single boxplot "box" will consist of the median, the first and third quartile, the minimum and maximum, and the outlier values of some variable, all for a given level of some stratifying variable (which itself will also be an element of s). Finally, the output of these constituent functions may also depend on some external parameters, which may be either directly supplied by the user or heuristically inferred from the data by the visualization system. Examples of such external parameters include the anchor and binwidth in a histogram, the density bandwidth in a density plot, or the list of knots for a smooth fit.

Third, each collection of summaries $s \in S$ needs to be translated from the data- (or summary-) coordinates into graphical coordinates/attributes $g \in G$, via a function β. This means that each summary value gets mapped or "scaled" to a graphical attribute via a constituent scaling function. Note that this mapping preserves cardinality - there are as many collections of graphical attributes as there are collections of summaries, $|G| = |S| = k$. For numeric/continuous summaries, scales often come in the form of linear transformations, such that the minimum and maximum of the data are mapped near the minimum and maximum of the plotting region, respectively. Continuous scales may also provide non-linear transformations such as the log-transformation or binning, and the values may get mapped to discrete graphical attributes (e.g. a binned value may get mapped to one of 5 different shades of a colour). Likewise, discrete summaries can be translated to either continuous (e.g. position) or discrete (e.g. colour) graphical attributes. There are also coordinate systems, which may further translate values from multiple scales simultaneously, e.g. by taking values in cartesian (rectangular plane) coordinates and mapping them to polar (radial) coordinates. Either way, β can be viewed as one function, representing the composition of any number of scales and coordinate systems applied to various summaries.

Finally, the collections of graphical attributes/coordinates $g \in G$ are drawn as geometric objects inside the plotting region via the graphic device. We can represent represent the plotting region as the set of pixels P. While the act of drawing does take the collections of graphical coordinates $g \in G$ as inputs, it does not simply return an output for each input (like a mathematical function would), but instead mutates the state of the graphical device via a side effect γ^*, i.e. changing the colour values of pixels in P. In other words, how the graphic ends up looking may depend, for example, on the order in which we draw the objects (a salient example of this is overplotting). As such, γ^* is not a simple mapping from G to P and we cannot call it a true mathematical function, since that would require it to merely assign an output to each input. The geometric objects may be simple, such as points, lines, or bars, or compound, such as a boxplot or pointrange. Finally, it is important to mention that each attribute necessary to draw a geometric object, such as x- and y-position, width, height, area, etc… needs to be present in the corresponding g.

The whole process can be summarized as follows:

$$D \xrightarrow{\alpha} S \xrightarrow{\beta} G \overset{\gamma^*}{\Rightarrow} P(\tag{1}$$

Or, equivalently:

$$\text{(data)} \xrightarrow{\text{summarize}} \text{(summaries)} \xrightarrow{\text{translate/encode}} \text{(graph. coordinates)} \overset{\text{draw}^*}{\Rightarrow} \text{(graph. device state)} \tag{2}$$

The above should be fairly non-controversial description of how a data visualization is produced, and applies equally well to static as well as interactive visualizations. There are a few caveats. Firstly, while not always the case, *order* can be important. For example, while we can draw the bars in a barplot in no particular order, lines in a lineplot need to be drawn by connecting points in specific order, otherwise we can end up with completely different-looking visualization. This means that either S, G, or their components, may need to be ordered. Secondly, *hierarchy* can also be important and the collections may form tree/graph-like structure. This is particularly relevant when stacking. For example, when drawing a stacked barplot, we need to stack the graphical coordinates representing the sub-bars on top of each other, such that each sub-bar is stacked within the appropriate parent-bar and in the correct order (e.g. such that sub-bars belonging to group 2 always go on top group 1 sub-bars). If it is the case that order matters for either α or β, then we cannot call them true functions either, since each of the elements in their domain needs to be aware of the other elements. That is, when stacking bars in a stacked barplot, we need to "remember" what bars we have stacked before.

4 The problem of statistical summaries

Circling back to the illusory rectangle in a barplot, we can now talk about interacting with the summarizing function α. In a typical barplot, α consists of the simple act of "counting" how many times each of the k unique levels of the categorical variable appears. We could substitute a different statistical for α, e.g. taking the sum or mean of the values of some other (continuous) variable in the data. But herein lies a problem.

While some summarizing functions α may produce perfectly valid static visualizations, when it comes to interactive visualizations, not every statistical summary "works" equally well. That is, introducing interactive features may impose additional constraints on what summaries will make up a coherent visualization. Let's illustrate the problem with a simple case study.

4.1 Case study: counts and means

Linked brushing or highlighting is one of the most popular and useful types of interactive features used in interactive data visualization (see e.g. Buja, Cook, and Swayne 1996). It allows the user to select cases within the data by by e.g. clicking or click-and-drag selecting objects within one plot and the selected cases are then highlighted across all other "linked" (hence the name) plots. The usefulness of linked brushing comes from the fact that it allows the user to rapidly "drill-down" (Dix and Ellis 1998; Theus 2002) and compare different subsets of the data.

Let's imagine we have three interactive plots: a classical scatterplot, a barplot of summarizing the count of cases within the levels of some categorical variable x, and a barplot summarizing the mean of some other variable y, also within the levels of x. The plots are linked such clicking/clicking-and-dragging objects in one plot will highlight the corresponding cases in the other plots. Intuitively, it might seem that the two barplots should be equally valid/useful representations of the data. However, if we consider these plots in the context of linked brushing, few subtle-yet-fundamental differences emerge.

4.2 Empty selections

First, as is shown in Figure 2, how do we represent empty selections? In the case of counts, we have a meaningful default value - zero - as in "the number of cases in an empty selection is zero". In other words, the absence of a bar in a barplot unambiguously indicates that the count for the corresponding level of the stratifying variables is zero.

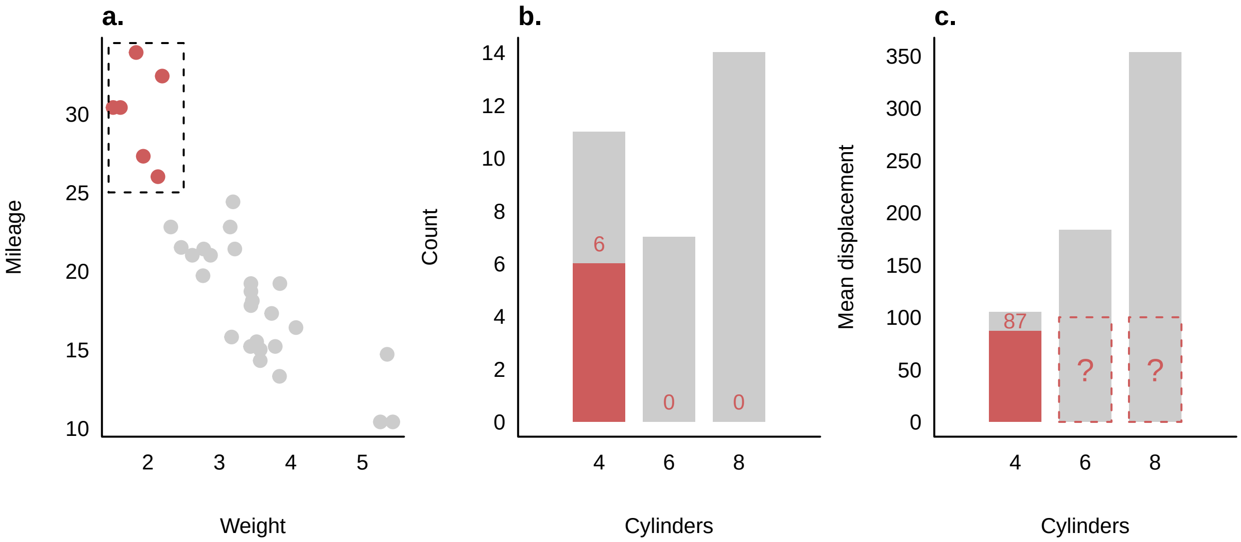

Figure 2: The problem of representing empty selection. a) An illustration of selection by linked brushing. b) In the barplot of counts, the count within an empty selection (red) is zero and so an absence of a bar accurately represents the count of zero. c) In the barplot of means, the mean of an empty selection is not defined. Absence of a bar could indicate that either no cases are selected or that some cases are selected and their mean is equal to the lower y-axis limit (zero in this case).

However, for means, there is no such default value: the mean of an empty set is not defined. As a result, whenever we encounter an empty selection, we are faced with the problem of how to represent it. We could choose not to draw the bar representing the empty selection, however, that decouples the statistical summary from the visual representation. That is, the absence of a bar may now indicate that *either* no cases are selected *or* that some cases are selected and that their mean is equal to the lower y-axis limit.

4.3 Part vs. the whole

Second, as shown in Figure 3, how does the summary on the selection relate to the summary on the whole? In the case of the barplot of counts, the height of the sub-bar is always less than or equal ($\leq$) to the height of the whole bar (because so is the count). Thus, we can always draw the sub-bar over the whole bar, and the whole bar act as a stable visual reference: the outline of multiple sub-bars stacked on top of each other will always remain the same no matter which cases of the data are selected. In fact, we can either draw the whole bar (as shown in grey in Figure 3) and draw the selected sub-bar (red) over it, both starting from the y-axis origin (0), or we can draw a red sub-bar and the "leftover" grey-sub bar stacked on top, starting from the top y-coordinate of the selection bar and with **height** = (count of the whole − count of the selection). The result will be visually identical.

Means do not share this property. Specifically, the mean of a subset can be greater than the mean of the superset, and so the "sub-bars" representing the selection may end up taller then the bar representing all cases within the level of the stratifying variable. As a result, if we draw the sub-bars on top of the whole bar, the whole bar may be completely obscured by the sub-bars (as shown in Figure 3). We could draw the selection sub-bars as semitransparent, or draw the the bars side-by-side instead of on top of each other (i.e. technique known as "dodging"), however, the question then remains how to display the non-selected (grey) cases - do we draw the "whole" bar that remains the same height throughout interaction, or the "leftover" bar whose height changes with the selection? Also, if we choose to draw the bars side-by-side, will the whole bar be initially wide and shrink in response to selection to accommodate the selection bars, or will it be narrow from the initial render? Finally, if the user brushes the side-by-side bars, will they be able to

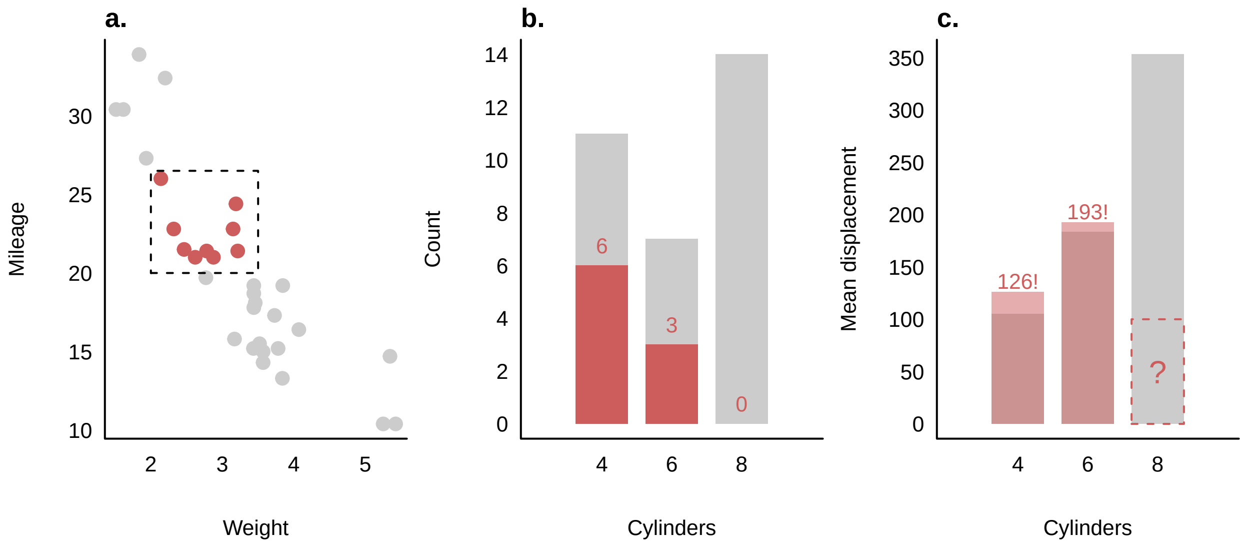

Figure 3: The relationship between selection vs. the whole. a) An illustration of selection by linked brushing. b) In the barplot of counts (middle), the count within a selection (red) is always less than or equal to the count within the whole and the outline of the bars does not change. c) In the barplot of means, the mean of a selection can be greater and so the outline of the bars will change in response to user input.

select individual bars or will brushing one select all of them?

4.4 Combining parts

Finally, as displayed in Figure 4, when multiple selections/groups are present, can we combine them in a meaningful way? Again, in the case of the barplot of counts, there is an idiomatic way of doing this: we can stack the counts across the groups and the corresponding bars on top of each other. The height of the resulting bar will then always be identical to the count of the whole (unselected) bar.

For means, there is no such way to combine the quantities. If we were to stack the bars on top of each other, the resulting statistic (the sum of the group means) would hardly be informative. Worse yet, if we were to take the mean of the group means, the resulting statistic will be (almost surely) different from the mean of the whole: the mean of the group means $\neq$ the grand mean. And again, as was mentioned in the previous section, the grand mean may be less than any of the group means. Since we cannot meaningfully combine the statistics, we could draw the bars side-by-side, but again, we run into considerations about how to render the base group and the selection, and how to choose the bar width.

4.5 Some statistics are better than others

To summarize, counts, as implemented in a typical barplot, provide:

a. An unambiguous way to display empty selections (absence of a bar = a count of zero)
b. A stable visual reference (the count within a sub-bar $\leq$ the count within the whole bar)
c. A way to combine the statistics together (the sum of the sub-bar counts = the count of whole bar).

Means do not share these properties: mean of an empty selection is not defined, the mean of a selection is not guaranteed to be less than the mean of the whole, and the mean of group means may be different from

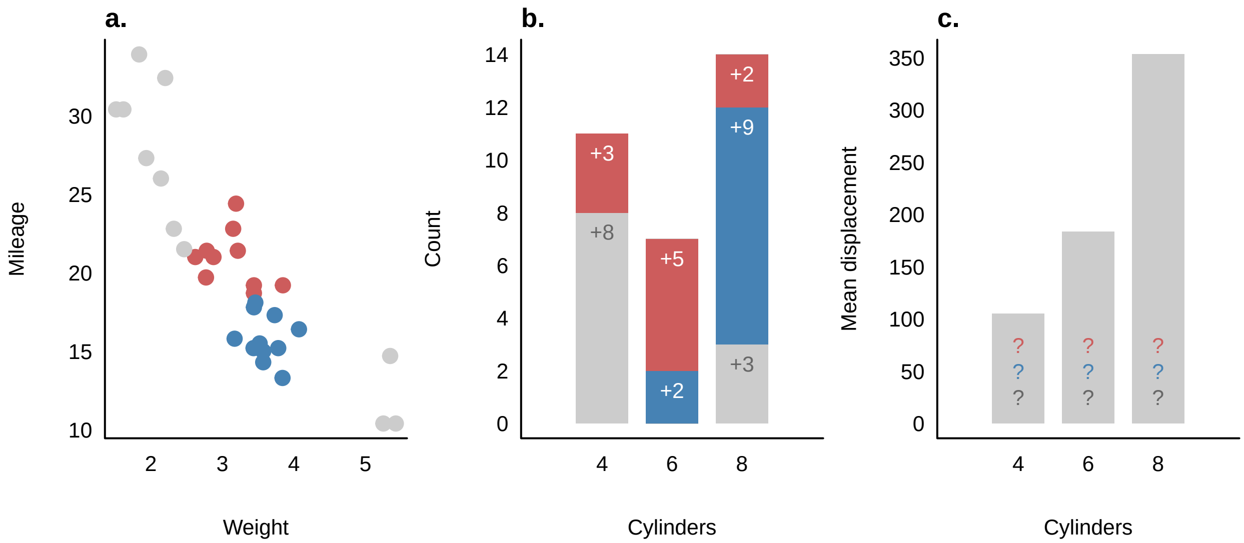

Figure 4: Combining selections. a) An illustration of selection by linked brushing, with two groups. b) In the barplot of counts, statistics can be stacked on top of each other such that the count of the stacked bar is identical to the count of the whole bar (i.e. one that would result from no selections). c) In the barplot of means, no such procedure for combining statistics exists.

the grand mean. Importantly, these properties or lack thereof are not tied to any specific plot type (e.g. a barplot) but are instead tied to the statistic underlying the plot (count/mean).

There are of course ways to display means such that linked brushing is possible. The key point, however, is that to make an IDV with means, many more decisions need to be made in order to produce a visualization that is at least somewhat visually and interactively coherent. Consequently, since many of these decision are arbitrary and none is more “natural” than any other, a person interacting with a linked barplot of means for the first time may be surprised by how the plot behaves. Conversely, for the barplot of counts, there *are* such natural solutions readily available. In fact, this may explain why barplots of counts are popular in systems which implement linked brushing.

One might wonder if these problems are just a quirk of linked brushing. However, one runs into them elsewhere as well. For example, stacking means will not make sense in static plots either. Further, we will run into some of the same issues with any other interactive feature that deals with summary statistics computed on parts of the data. If, for example, on top of linked brushing, we wanted to implement a pop-up window that displays summary statistics within a given object as text, we would still have to contend with the problem of how to display empty selections. Here, there is perhaps a simple solution: we could display the mean of an empty selection as some special value, for example `NA`, `undefined`, or an empty string, `""`. However, by doing so, we have to step outside of the value type of summaries on non-empty selections (real number/`float`). While this may not be a problem in most types of statistical software we may use, it still does add some complexity to our code, since whatever functions rely on means (such as the display function) now have to be generic over `float` and `NA`/`underfined`/`""`. On the other hand, there is something natural about counts having the value for empty selections (zero) be of the same type as all other values (`int`).

Are counts unique in being so well-behaved in this way? The short answer is “no”. For example, sums or products (of values ≥ 1) conform to all three properties listed above. In other words, in the context of linked brushing, a stacked barplot will behave equally well if it display sums (not surprising) or products (perhaps somewhat surprising?), see Figure 5. In fact, there is whole a class of mathematical functions, or more precisely mathematical structures, that share these nice properties. To discuss these, let’s first lay the groundwork with some relevant theory.

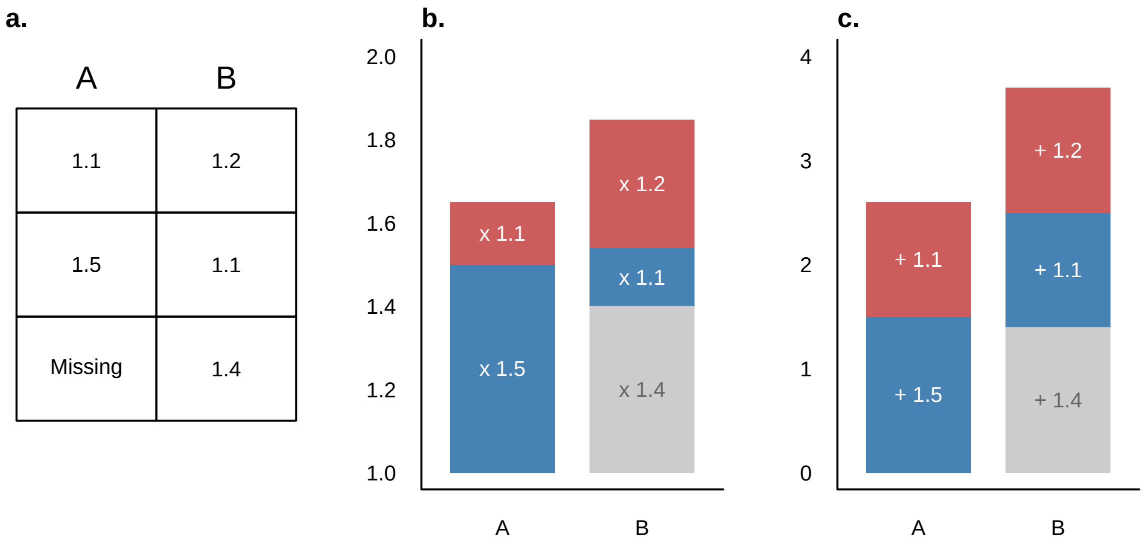

Figure 5: Sums and products work equally well as counts when it comes to displaying empty selections, providing a stable visual reference, and being able to be combined together. Note the different y-axis limits. a) The underlying data. b) Barplot of products. c) Barplot of sums

5 Few relevant bits of applied category theory

The short treatment below follows mainly from parts of Fong and Spivak (2019), Lawvere and Schanuel (2009), Baez (2023), and Milewski (2018). Category theory is a very rich and deep subject, and this treatment only goes over a few very basic concepts. Further, these concepts appear in other areas mathematics as well, including abstract algebra and group theory, and so it could be argued that the term “category theory” is out of place here. Nonetheless, I titled this section in this way because that is how I discovered the concepts, and I think it may provide a useful signpost for anyone who would want to dive deeper. I hope that any mathematician who might read this can tolerate the pedestrian nature on display here. But that is also somewhat the point - even a few basic concepts might be valuable when thinking about (interactive) visualizations.

5.1 Functions

A function is a mapping between two sets. More specifically, let S be the set of sources (also called the *domain*) and T be the set of possible targets (also called the *codomain*). Then, one way to think of a function is as a subset $F \subseteq S \times T$ of valid source-target pairs (s, t), such that for every $s \in S$ in there exists a unique $t \in T$ with $(s, t) \in F$. The function can then be thought of as the process of picking a target for any valid source it is given.

If every target in the function’s codomain can be reached via the function, that is, if for a function f and for all $t \in T$ there exists a $s \in S$ such that $f(s) = t$, then we call the function a *surjective* or *onto* function, see Figure 6a. If each source leads to a unique target, i.e. for $s_1, s_2 \in S$, if $f(s_1) = t$ and $f(s_2) = t$, then $s_1 = s_2$, then it is an *injective* or *one-to-one* function, see Figure 6b. Also, for any given subset of targets, we can ask about the subset of sources that could have produced them, a *pre-image*. That is, for $T_i \subseteq T$ we can define $f^{-1}(T_i) = \{s \in S | f(s) \in T_i\}$.

Finally, a very useful property of functions is that they can be composed. That is, if the domain of one function matches the codomain of another, the functions can be composed to form a new function. Specifically, if we have two functions $f : X \to Y$ and $g : Y \to Z$, we can form a new function $h = g(f(x)) = g \circ f$ (read "g after f") such that $h : X \to Z$.

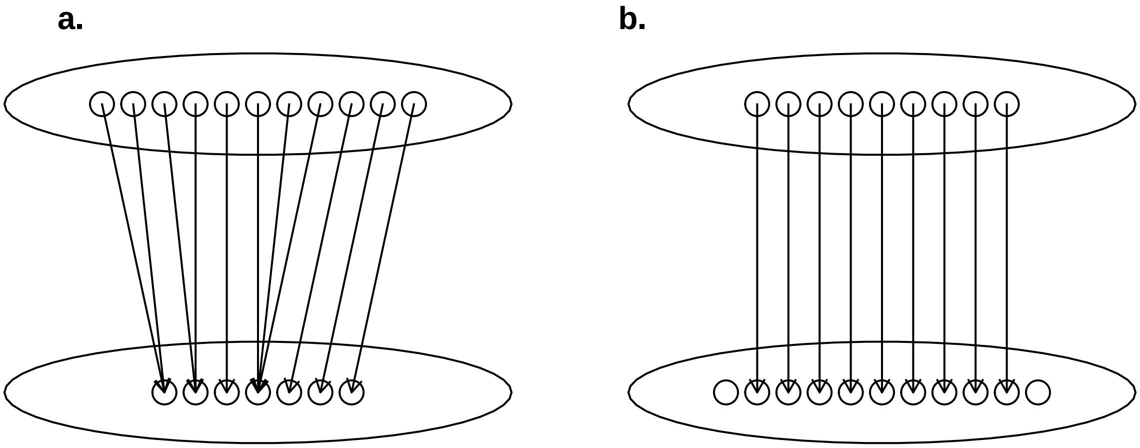

Figure 6: Two types of functions: a) surjective, b) injective.

5.2 Partitions

One useful thing we can do with functions is to form partitions. Specifically, given some arbitrary set A, we can assign every element a label from a set of part labels P via a surjective function $f : A \to P$. Conversely, we can then take any part label $p \in P$ and recover the corresponding subset of A by pulling out its pre-image: $f^{-1}(p) = A_p \subseteq A$. 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 and:

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

I.e. the parts A_p jointly cover the entirety of A and parts cannot share any elements.

We can rank partitions by their coarseness. That is, for any set A, the coarsest partition is one with only one part label $P = \{1\}$, such that each element of A gets assigned 1 as label. Conversely, the finest partition is one where each element gets assigned its own unique part label, such that $|A|= |P|$. Finally, given two partitions, we can form a finer (or at least as fine) partition by taking their intersection, i.e. by taking the set of all unique pairs of labels that co-occur for any $a \in A$ as the new part labels. For example, if $A = \{1, 2, 3\}$ and partition 1 assigns part labels $P_1 = \{x, y\}$ to A such that $f_1(1) = x,\ f(2) = x,\ f(3) = y$, and partition 2 assigns part labels $P_2 = \{\rho, \sigma\}$ such that $f_2(1) = \rho,\ f_2(2) = \sigma,\ f_2(3) = \sigma$, then the intersection partition will have part labels $P_3 = \{(x, \rho), (x, \sigma), (y, \sigma)\}$ such that $f_3(1) = (x, \rho),\ f_3(2) = (x, \sigma),\ f_3(3) = (y, \sigma)$.

5.3 Preorders

A preorder is a set X equipped with a binary relation $\leq$ that conforms to two simple properties:

1. $x \leq x$ for all $x \in X$ (reflexivity)
2. if $x \leq y$ and $y \leq z$, then $x \leq z$, for all $x, y, z \in X$ (transitivity)

Simply speaking, this means that between any two elements in X, there either is a relation and the elements relate (one element is somehow “less than or equal” to the other), or the two elements do not relate.

An example of a preorder is the family tree, with the underlying set being the set of family members ($X = \{\textbf{daughter}, \textbf{son}, \textbf{mother}, \textbf{father}, \textbf{grandmother}, ...\}$) and the binary relation being ancestry or familial relation. Thus, for example, **daughter** $\leq$ **father**, since the daughter is related to the father, and **father** $\leq$ **father**, since a person is related to themselves. However, there is no relation ($\leq$) between **father** and **mother**, since they are not related. Finally, since **daughter** $\leq$ **father** and **father** $\leq$ **grandmother**, then, by reflexivity, **daughter** $\leq$ **grandmother**.

We can further constrain preorders by imposing additional properties, such as:

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

If a preorder conforms to 3., we speak of a partially ordered set or *poset*. If it conforms to both 3. and 4., then it is a *total order*.

5.4 Monoids

A monoid is a tuple $(M, e, \otimes)$ consisting of:

a. A set of objects M
b. A neutral element e called the *monoidal unit*
c. A binary operation $\otimes : M \times M \to M$ called the *monoidal product*

Such that:

1. $m \otimes e = e \otimes m = m$ for all $m \in M$ (unitality)
2. $m_1 \otimes (m_2 \otimes m_3) = (m_1 \otimes m_2) \otimes m_3 = m_1 \otimes m_2 \otimes m_3$ for all $m_1, m_2, m_3 \in M$ (associativity)

In simple terms, monoids encapsulate the idea that *the whole is exactly the “sum” of its parts* (where “sum” can be replaced by any operation that conforms to the rules). Specifically, we have some elements in M and a way to combine them $\otimes$, such that when we combine the same elements we always get the same result, no matter where we put the brackets. Finally, we have some neutral element that when combined with some element m yields back the same element m.

For example, take summation on natural numbers, $(\mathbb{N}, 0, +)$:

$$1 + 0 = 0 + 1 = 1 \qquad \text{(unitality)}$$
$$1 + (2 + 3) = (1 + 2) + 3 = 1 + 2 + 3 \qquad \text{(associativity)}$$

Other examples of monoids include products of real numbers $(\mathbb{R}, 1, \times)$, the min and max operators $(\mathbb{R}, \infty, \min)$ and $(\mathbb{R}, -\infty, \max)$, and multiplication of $n \times n$ square matrices $(\mathbf{M}_{n \in \mathbb{Z}}, \mathbf{I}, \cdot)$, where $\mathbf{I}$ is the identity matrix and $\cdot$ stands for an infix operator that is usually omitted. As a counterexample, exponentiation does not meet the definition of a monoid, since it is not associative: $x^{(y^z)} \neq (x^y)^z$.

However, monoids do not need to concern numerical quantities only. For example, string concatenation (with empty string `""` as the monoidal unit) is also a monoid:

$$\texttt{"hello"} + \texttt{""} = \texttt{""} + \texttt{"hello"} = \texttt{"hello"}$$
$$(\texttt{"quick"} + \texttt{"brown"}) + \texttt{"fox"} = \texttt{"quick"} + (\texttt{"brown"} + \texttt{"fox"}) = \texttt{"quick brown fox"}$$

Likewise, with the set of booleans $\mathbb{B}$, the logical **AND** and **OR** operators also form a monoid.

We can impose further restrictions on monoids, for example:

3. $m_1 \otimes m_2 = m_2 \otimes m_1$ for all $m_1, m_2 \in M$ (commutativity)
4. If $m_1 \leq m_3$ and $m_2 \leq m_4$ then $m_1 \otimes m_2 \leq m_3 \otimes m_4$ for all $m_1, m_2, m_3, m_4 \in M$ (monotonicity)

Property 3. means that the order in which we combine terms does not matter, and property 4. means that combining two smaller terms cannot get us something bigger than combining two greater terms. Summation of natural numbers $(\mathbb{N}, 0, +)$ is both commutative and monotonic, multiplication of reals $(\mathbb{R}, 1, \times)$ is commutative but not monotonic, and the multiplication of matrices with real/integer entries $(\mathbf{M}_{n \in \mathbb{Z}}, \mathbf{I}, \cdot)$ is neither monotonic nor commutative.

One thing we left out of the definition above is that, to be able to determine whether a monoid is monotonic, we need a way to compare two elements, i.e. we need the $\leq$ operator. This means that the set M in the monoid needs to be a preorder such that we can claim monotonicity (or lack thereof). A monoid that is both commutative and monotonic is called a commutative monoidal preorder.

6 Fluent interactive graphics

With these ideas in place, we can look at the process of (interactive) data visualization in a new light.

6.1 Data and partitions

As before, we start with the data D. We know that we will want to summarize the datapoints $d \in D$ via some summarizing function α, before drawing the result as one or more geometric objects. How should this summarizing step work?

First, since we will often be drawing multiple objects, we need to think about splitting the data into multiple subsets. This is where it may be useful to frame the process as forming a partition on D, such that we end up with a set of non-empty parts $p \in P$, with each datapoint $d \in D$ belonging to exactly one $p \in P$. There are two consequences to this. First, every datapoint d must end up in *some* part p. This seems entirely reasonable - we want our visualization to be an accurate and complete representation of the data, and as such we cannot simply leave information out arbitrarily. This may become more complicated if the data set has missing or incomplete values (for some ideas, see e.g. N. J. Tierney and Cook 2018), or if it is too large to store in memory, but in the context of intact, reasonably-sized data sets, there does not seem to be any reason to throw data away. Second, *no* datapoint d can end up in *more than one* part p. This is a bit more subtle. We could, for example, receive the two arrays **employees** and **managers** as our data D, with every manager in **managers** also being included in **employees** ($\textbf{managers} \cap \textbf{employees} \neq \varnothing$, since every manager is also an employee). We could then, for example, choose the arrays as our parts $p \in P$, such that $p_1 = \textbf{employees}$ and $p_2 = \textbf{managers}$, count up the number of people in each array, and draw these as bars in a barplot. However, this kind of visualization is problematic in several ways. First, the representation of the data (two non-disjoint arrays) itself already violates the principle of tidy data (Wickham 2014) and/or the third normal form (Codd 1990): it unnecessarily duplicates information, and would be better represented as e.g. a single table with **employee** and **role** columns. Further, whatever query we may have about the data may also be better answered by a plot that uses the tidy representation. If we were interested, for example, in how many more employees there are relative to the managers, we may use the table with the two columns, partition on the **role** column, and draw bars showing the counts for managers and non-manager employees. If we were instead interested in what proportion of employees the managers make up, we could use the same partitioning but stack the counts of managers and non-manager employees into a single bar. Overall, it seems that the isomorphism **geometric objects** $\cong$ **disjoint parts** is desirable.

We may form the partitions in many different ways, some of which were already mentioned in previous sections. We may use some structure already present in the data such as rows, columns, or cells; we may partition on the levels of some categorical variable or “factor”; or we may transform some continuous variable into a categorical one, for example by binning. Either way, the important point is that we end up with some way of allocating the datapoints into several disjoint parts.

6.2 Partition hierarchy

Often, just a single partition of the data will not be enough. Instead, it may be useful or indeed necessary to form a (nested) hierarchy of partitions. The reason for this may be best explained with an example.

Let's imagine a linked histogram which supports linked brushing as well as interactive manipulation of binwidth and anchor. The key insight here is that, whenever we brush, some graphical attributes of the plot, such as the upper y-axis limit and borders of the histogram bins (plotted on the x-axis), do not change; only the heights of the stacked sub-bars do. However, whenever we change binwidth or anchor, all of these attributes - the upper y-axis limit, the bin borders, and the heights of sub-bars - may change.

We can model this dependency between summaries as a hierarchy of nested partitions. Specifically, we can represent the histogram via two partition levels: level 1) partitioning the data by the **bin** variable/factor (which depends on the binwidth and anchor parameters), and level 2) partitioning the data by the **bin** × **group** intersection (where **group** is the auxiliary variable that gets updated via brushing). The advantage of this approach is that any summaries that correspond to one partition level only need to be updated whenever that level or any higher/coarser level changes, but not when any of the lower/finer levels change. For example, the upper y-axis limit and the x-axis boundaries of the bins depend on level 1 partition (the **bin** variable) only: as such, we only need to update them when binwidth or anchor change, but not when brushing takes place. Conversely, the counts within sub-bars depend on level 2 partition (the **bin** × **group** intersection variable). In this way, the partitions and the corresponding summaries computed on them form a directed acyclic graph (DAG), see Figure 7.

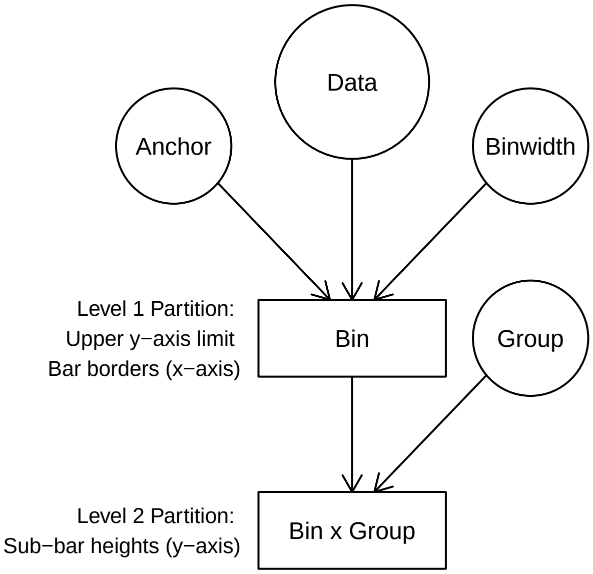

Figure 7: Parition hierarchy in a stacked histogram represented as DAG. Partitioning variables/factors are shown as rectangles. This dependency structure implies that, when group changes, only sub-bar heights need to be recomputed. However, when either the anchor or binwidth change, both the upper y-axis limit and bar borders, and the sub-bar heights, need to be recomputed.

The same hierarchy of partitions, with one small modification, will work equally well for a spineplot. The modification is that, since we are now also stacking the counts in whole bins (along the x-axis), we might benefit from having one more partition level: the coarsest possible level 0 partition, which consists of the entire data set. In other words, we can think of this partition as a constant function which maps every datapoint to the same single part. We can then, same as in level 1 and level 2 partitions, count the number of cases corresponding to the one part, which in this case is just n (the size of the data set), and use this as our upper x-axis limit, since this will give the same result as stacking counts across all histogram bins. Now, one might wonder why bother with the summarizing the partition at all - why not just use n, if our data provides this information? Without jumping ahead too much, the reason is that we may want to use a different summarizing function, and so we cannot assume that the data set object/class will provide every possible summary that can be computed across all datapoints. Besides, since the level 0 partition can never change (every datapoint will always belong to the one part that is the whole data set), we only need to compute the level 0 summaries once, and so this operation should be fairly cheap. In fact, it may be reasonable to always include the whole-data level 0 partition by default, since it will not affect the time which it takes to render interactions, only the time it takes to render the visualization the first time.

The hierarchical nature of the partitions may be useful not just when drawing/rendering but also when implementing interactions. For example, let's say we want to implement linked brushing. To do this, we need a way to check which cases of the data have been selected via the brush. We do this by querying the bounds of the geometric objects, for example by taking the four corners of each rectangle in a barplot/histogram and checking if any of them falls inside the brushing regions. Importantly, we want to query the "whole" objects (e.g. bars defined by the **bin** variable), and not the object parts (e.g. sub-bars defined by **bin** × **group**). There are two reasons for this. First, this is the conventional way of implementing linked brushing. Of the packages listed in introduction, to the author's knowledge, all implement brushing in this way. In plain words, when users brush a linked barplot or histogram, they do not expect to select individual sub-bars that make up the stacked whole bars. Even the systems which implement selection operators such as `AND` and `OR` (e.g. *Mondrian*, Theus 2002), typically operate on the level of whole objects. Second, and more importantly, querying whole bars is less work. If the **bin** variable has k_1 unique levels and the **bin** × **group** has k_2 unique levels, then $k_1 \leq k_2$ always ($k_1 = k_2$ only when there is only a single group present in **group**). Either way, to query whole objects, we can use the summaries/coordinates corresponding to higher level/coarser partitions and ignore lower level/finer partitions. For example, to implement brushing in an interactive histogram, we can query summaries on the level 1 partitions and and ignore the summaries on level 2 partition entirely.

The picture that emerges is that a convenient representation of interactive visualizations may be the following hierarchy of $m \geq 3$ partition levels: level 0, representing the whole data set, some intermediate levels 1 through to $m - 2$, representing partitioning of the data into smaller and smaller objects (with level $m - 2$ representing the smallest "whole" objects), and finally level $m - 1$, representing the partitioning induced by the linked brushing variable **group**. For histogram and spineplot, $m = 3$, and so $m - 1 = 2$ and the finest level is level 2 and there is just one intermediate object level, namely level 1. However, for other types of plots, e.g. mosaic plot, we may want several object partitions.

Finally, since the parts within partitions form a tree, e.g. the whole-data part has bar parts as children, and each bar part has sub-bar parts as children, it may be tempting to think we could compute the parent parts by combining the children. However, this would defeat the purpose of the hierarchy since the lower level (finer) partitions change more frequently than higher level (coarser) ones. For example, the counts on sub-bars change often (as a result of brushing) and so it would not make sense to compute counts on the whole bars by adding them up - we would have to do this every time brushing occurs, even though the resulting statistic would be the same (e.g. the sum across sub-bars $5 + 3 + 10 = 18$ is the same as $7 + 7 + 4 = 18$). Instead, each partition level needs to run its own computation across the whole data set and update the summaries on its respective parts when and only when it needs to.

6.3 Computing summaries

This whole time, we have been talking about computing summaries on partitions (e.g. counts within bins), but what should these summaries be and how should they be computed?

First, the summaries may be either discrete or numeric. Most often, discrete summaries or "labels" will be computed as part of the process of creating the factor/partition variable, which needs to occur before the summarizing step. For example, in a barplot, we partition on some categorical variable and we use the levels of that variable as summaries that get encoded into graphical attributes, i.e. x-axis positions/labels. Thus, once we have a factor, we get the factor labels for free, and most of the time those are probably all the discrete summaries we need. But we could also envision a discrete summary that is not one of the factor labels, for example, we could find the name of the student with the highest GPA within levels of **school** variable.

Either way, numerical summaries are usually what we think of when it comes to statistical summaries in plots. The classic examples include counts, sums, or means. However, as was discussed in Section 4, some types of summaries such as means do not make for well-behaved interactive graphics. Provided that we want our interactive plots to support two-way linked brushing, we need to impose some constraints on the summaries that we will compute.

We propose that any summary computed on the parts should form a commutative monoidal preorder. To rephrase Section 5.4, the summaries should consists of a set M with elements that can be at least partially ordered (such that, for any m_1 and m_2, we may determine whether $m_1 \leq m_2$), a neutral element e, and a binary operation $\otimes$ such that:

1. $m \otimes e = e \otimes m = m$ for all $m \in M$ (unitality)
2. $m_1 \otimes (m_2 \otimes m_3) = (m_1 \otimes m_2) \otimes m_3 = m_1 \otimes m_2 \otimes m_3$ for all $m_1, m_2, m_3 \in M$ (associativity)
3. $m_1 \otimes m_2 = m_2 \otimes m_1$ for all $m_1, m_2 \in M$ (commutativity)
4. If $m_1 \leq m_3$ and $m_2 \leq m_4$ then $m_1 \otimes m_2 \leq m_3 \otimes m_4$ for all $m_1, m_2, m_3, m_4 \in M$ (monotonicity)

Why might these properties be desirable for summaries represented by interactive visualizations? Let's go through them one by one. Property 1. says that there is a valid way of representing "nothing" in our summary. As we have seen in 4.2, this is useful since, for example, the absence of an object unambiguously indicates an empty selection. Property 2. says that combining

Requiring summaries to be commutative monoidal preorder has another benefit. This condition means that we can always compute the summaries in a single pass through the data, via a *reduce/fold/accumulate* function (see e.g. Abelson and Sussman 2022, 53). Specifically, we can reduce the data into k collections of summaries on the k parts (within one partition level) by first initializing each collection with the default values (monoidal units), and then iterating through the n data points and updating jth collection on each step, with j being the factor level corresponding to the ith datapoint.

References

Abelson, Harold, and Gerald Jay Sussman. 2022. *Structure and Interpretation of Computer Programs: JavaScript Edition*. MIT Press.

Baez, John. 2023. "Applied Category Theory Course." https://math.ucr.edu/home/baez/act_course.

Bostock, Michael, Vadim Ogievetsky, and Jeffrey Heer. 2011. "D^3 Data-Driven Documents." *IEEE Transactions on Visualization and Computer Graphics* 17 (12): 2301–9.

Buja, Andreas, Dianne Cook, and Deborah F Swayne. 1996. "Interactive High-Dimensional Data Visualization." *Journal of Computational and Graphical Statistics* 5 (1): 78–99.

Burleson, Thomas. 2020. "Flare | Data Visualization for the Web." *Blokt - Privacy, Tech, Bitcoin, Blockchain & Cryptocurrency*. https://blokt.com/tool/prefuse-flare.

Codd, Edgar F. 1990. *The Relational Model for Database Management: Version 2*. Addison-Wesley Longman Publishing Co., Inc.

Crossfilter Organization. 2023. "Crossfilter." *GitHub*. https://github.com/crossfilter/crossfilter.

Dix, Alan, and Geoffrey Ellis. 1998. "Starting simple: adding value to static visualisation through simple interaction." In *AVI '98: Proceedings of the working conference on Advanced visual interfaces*, 124–34. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/948496.948514.

Donoho, Andrew W, David L Donoho, and Miriam Gasko. 1988. "MacSpin: Dynamic Graphics on a Desktop Computer." *IEEE Computer Graphics and Applications* 8 (4): 51–58.

Fisherkeller, Mary Anne, Jerome H Friedman, and John W Tukey. 1974. "An Interactive Multidimensional Data Display and Analysis System." SLAC National Accelerator Lab., Menlo Park, CA (United States).

Fong, Brendan, and David I Spivak. 2019. *An Invitation to Applied Category Theory: Seven Sketches in Compositionality*. Cambridge University Press.

Fowlkes, EB. 1969. "User's Manual for a System Fo Active Probability Plotting on Graphic-2." *Tech-Nical Memorandum, AT&T Bell Labs, Murray Hill, NJ*.

Heer, Jeffrey, Stuart K. Card, and James A. Landay. 2005. "prefuse: a toolkit for interactive information visualization." In *CHI '05: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 421–30. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/1054972.1055031.

Highcharts Team. 2023. "Interactive javascript charts library." https://www.highcharts.com.

Kruskal, J. B. 1965. "Multidimensional Scaling." https://community.amstat.org/jointscsg-section/media/videos.

Lawvere, F William, and Stephen H Schanuel. 2009. *Conceptual Mathematics: A First Introduction to Categories*. Cambridge University Press.

Leman, Scotland C, Leanna House, Dipayan Maiti, Alex Endert, and Chris North. 2013. "Visual to Parametric Interaction (V2pi)." *PloS One* 8 (3): e50474.

Milewski, Bartosz. 2018. *Category Theory for Programmers*. Blurb.

Pike, William A, John Stasko, Remco Chang, and Theresa A O'connell. 2009. "The Science of Interaction." *Information Visualization* 8 (4): 263–74.

Plotly Inc. 2022. "Part 4. Interactive Graphing and Crossfiltering | Dash for Python Documentation | Plotly." https://dash.plotly.com/interactive-graphing.

Satyanarayan, Arvind, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2016. "Vega-Lite: A Grammar of Interactive Graphics." *IEEE Transactions on Visualization and Computer Graphics* 23 (1): 341–50.

Satyanarayan, Arvind, Ryan Russell, Jane Hoffswell, and Jeffrey Heer. 2015. "Reactive Vega: A Streaming Dataflow Architecture for Declarative Interactive Visualization." *IEEE Transactions on Visualization and Computer Graphics* 22 (1): 659–68.

Swayne, Deborah F., Dianne Cook, and Andreas Buja. 1998. "XGobi: Interactive Dynamic Data Visualization in the X Window System." *J. Comput. Graph. Stat.* 7 (1): 113–30. https://doi.org/10.1080/10618600.1998.10474764.

Swayne, Deborah F., Duncan Temple Lang, Andreas Buja, and Dianne Cook. 2003. "GGobi: evolving from XGobi into an extensible framework for interactive data visualization." *Comput. Statist. Data Anal.* 43 (4): 423–44. https://doi.org/10.1016/S0167-9473(02)00286-4.

Theus, Martin. 2002. "Interactive Data Visualization using Mondrian." *J. Stat. Soft.* 7 (November): 1–9.

https://doi.org/10.18637/jss.v007.i11.

Tierney, Luke. 1989. “Xlisp-Stat.” School of Statistics Report.

———. 2004. “Some Notes on the Past and Future of Lisp-Stat.” *Journal of Statistical Software* 13: 1–15.

Tierney, Nicholas J, and Dianne H Cook. 2018. “Expanding Tidy Data Principles to Facilitate Missing Data Exploration, Visualization and Assessment of Imputations.” *arXiv Preprint arXiv:1809.02264*.

Urbanek, Simon, and Martin Theus. 2003. “iPlots: High Interaction Graphics for r.” In *Proceedings of the 3rd International Workshop on Distributed Statistical Computing*. Citeseer.

Vanderplas, Jacob, Brian Granger, Jeffrey Heer, Dominik Moritz, Kanit Wongsuphasawat, Arvind Satyanarayan, Eitan Lees, Ilia Timofeev, Ben Welsh, and Scott Sievert. 2018. “Altair: Interactive Statistical Visualizations for Python.” *Journal of Open Source Software* 3 (32): 1057.

Wickham, Hadley. 2014. “Tidy Data.” *J. Stat. Soft.* 59 (September): 1–23. https://doi.org/10.18637/jss.v059.i10.

Wickham, Hadley, Michael Lawrence, Dianne Cook, Andreas Buja, Heike Hofmann, and Deborah F Swayne. 2009. “The Plumbing of Interactive Graphics.” *Computational Statistics* 24: 207–15.

Wilkinson, Leland. 2012. *The Grammar of Graphics*. Springer.

Xie, Yihui, Heike Hofmann, and Xiaoyue Cheng. 2014. “Reactive Programming for Interactive Graphics.” *Statistical Science*, 201–13.

Yi, Ji Soo, Youn ah Kang, John Stasko, and Julie A Jacko. 2007. “Toward a Deeper Understanding of the Role of Interaction in Information Visualization.” *IEEE Transactions on Visualization and Computer Graphics* 13 (6): 1224–31.