

# Towards Fluent Interactive Data Visualization

Adam Bartonicek

2023-11-10

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>The illusion of objects</b>	<b>2</b>
<b>3</b>	<b>Rough sketch of the data visualization process</b>	<b>3</b>
<b>4</b>	<b>The problem of statistical summaries</b>	<b>5</b>
4.1	Linked brushing . . . . .	6
4.2	Stacking vs dodging . . . . .	6
4.3	Case study: linked brushing of counts and means . . . . .	7
4.4	Case for restricting statistics? . . . . .	8
<b>5</b>	<b>Few relevant bits of applied category theory</b>	<b>9</b>
5.1	Functions . . . . .	10
5.2	Partitions . . . . .	11
5.3	Preorders . . . . .	11
5.4	Monoids . . . . .	12
<b>6</b>	<b>Fluent interactive graphics</b>	<b>13</b>
6.1	Data and partitions . . . . .	13
6.2	Partition hierarchy . . . . .	14
6.3	Computing summaries . . . . .	16
6.4	Displaying summaries through graphical attributes . . . . .	17
<b>7</b>	<b>Conclusion</b>	<b>17</b>
	<b>References</b>	<b>19</b>

# 1 Introduction

Interactive data visualization (IDV) has seen a rapid growth in popularity over the last few decades. From the humble roots of the early 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. Of these, 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* (Burlinson 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 related to 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 coordinates 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 implement the data processing step 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 (the 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, to be able to change binwidth and anchor in an interactive histogram, we need to recompute the number of cases in each bin whenever either of the two parameters is updated. Similarly, to implement linked highlighting/brushing across multiple plots, such that the user can highlight some cases in the data by selecting the corresponding geometric objects in one plot, we need to keep track of which cases belong to which object and then recompute the relevant statistics on the fly whenever necessary.

There have been calls for a development of a theory of an interactive data visualization pipeline or “plumbing” (Wickham et al. 2009), that would allow us to reason about the data processing step in the production of IDVs in a similar way that, for example, the *Grammar of Graphics* (Wilkinson 2012) has allowed us to reason about the production of static visualizations. However, these calls 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 an IDV pipeline may lie in a subtle yet profound question that inevitably comes up in the production and use of 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 the 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.

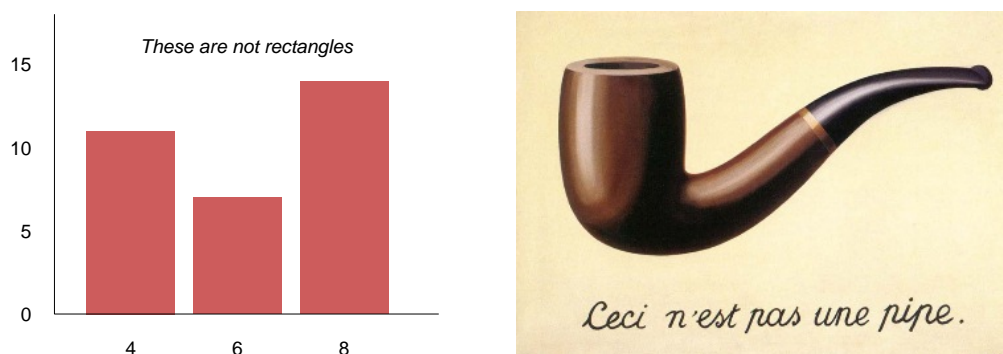


Figure 1: A rectangle is not a bar in a similar way that a painting of a pipe is not a pipe

Thus, objects in plots 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, geometric objects or **geoms** are considered just one component of a plot in the *Grammar of Graphics* (Wilkinson 2012) and its popular implementation *ggplot2* (Wickham et al. 2009).

However, what is this “structure” behind geometric objects? We may be able to muster a few ideas. 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”. Thus, every geometric object or “mark” in a plot (outside of auxiliary marks such as axis ticks, gridlines, etc...) has to directly represent one or more cases of the data.

But data is only a part of the story. When drawing plots, we rarely represent the data directly, in its raw form. Instead, we often summarize, aggregate, or transform (see Wilkinson 2012, chap. 7). We do this by applying mathematical functions such as count, sum, mean, log, or the quantile function. In *ggplot2*, these are the less well-known but no less important counterparts to **geoms**: statistics or **stats** (Wickham et al. 2009). It is the output of these summary functions that we represent by geometric objects.

So, when interacting with a bar in an interactive barplot, we do not just interact with a rectangle. Instead, we interact with a mathematical function, or, in fact, several of them. To the end-user of visualizations this may not seem particularly relevant, but it is, for one very important reason: functions have properties, and these properties may 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 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 “data points”)  $D$ . Data in the wild usually comes with more structure than that - for example, we often encounter data stored in a tabular (or even “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 present purpose, we do not have to assume any special structure for the data, and can instead just speak of  $n$  data points  $d \in D$ .

Second, at some point during the visualization process, we need to transform the data points  $D$  into a set of collections of summaries  $S$  via some function  $\alpha$ . This summarizing function can come many different forms. It may be the case that  $\alpha$  is one-to-one (bijection), in which case there is one collection of summaries for every unit of data (and vice versa). This is the case, for example, in the typical scatterplot, in which  $\alpha$  is just the identity function (every row of data gets assigned one point). However, more often,  $\alpha$  is many-to-one, which means that multiple units of data may be merged or combined into a single summary. Examples of this include the typical barplot, histogram, density plot, violin plot, mosaic plot, and many others. When  $\alpha$  is many-to-one, it will often 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).

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  $\beta$ . 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,  $\beta$  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 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  $\gamma^*$ , 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,  $\gamma^*$  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:

$$D \xrightarrow{\alpha} S \xrightarrow{\beta} G \xRightarrow{\gamma^*} P( \quad ) \quad (1)$$

Or, equivalently:

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

The above should be fairly non-controversial description of how data visualizations are produced, and applies equally well to static as well as interactive visualizations. There are a few caveats, however. Firstly, while not always the case, *order* can be important. For example, while the order of the bars in a barplot along the x-axis is usually arbitrary (and typically defaults to alphabetical order), lines in a lineplot need to be drawn by connecting points in a specific order, otherwise we end up with completely different-looking objects (lines). Overall, 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  $\alpha$  or  $\beta$ , 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

*“This system cannot produce a meaningless graphic, however. This is a strong claim, vulnerable to a single counter-example. It is a claim based on the formal rules of the system, however, not on the evaluation of specific graphics it may produce. [...]”*

*“Some of the combinations of graphs and statistical methods may be degenerate or bizarre, but there is no moral reason to restrict them.”*

Grammar of Graphics, Wilkinson (2012), p. 15 and p. 112

*“It’s unclear to me what you want to do here – what do you expect to happen? If you highlight a line segment on one chart, what should be highlighted on the other chart? The line segments do not have a 1-1 mapping between panels because the points are in different orders, so it’s not obvious to me what the “right” behavior is.”*

Response to a StackOverflow post about linked brushing by user Leo (jakevdp 2019)

Returning to the example of the “illusory rectangle” in a barplot, we can discuss the process of interacting with the summarizing function  $\alpha$ . In a typical barplot,  $\alpha$  consists of the simple operation of counting how many times each of the  $k$  unique levels of the categorical variable appeared in the data. But  $\alpha$  does not *have* to consist of counting. We could substitute a different statistical summary for  $\alpha$ , e.g. taking the sum or mean of the values of some other (continuous) variable in the data. Indeed, this is what we can do by specifying the `stat` argument in `ggplot2` plots.

However, herein lies the trap - just because we can substitute a *different* function for  $\alpha$ , this does not mean *every* function will work. If our goal is to produce coherent visualizations with specific features, only some summarizing functions will do. This is especially true for interactive visualizations, since interactive features impose additional constraints on the visualization. We will illustrate the problem with a case study of implementing linked brushing.

## 4.1 Linked brushing

Linked brushing or highlighting is one of the most popular and useful interactive features used in IDVs (see e.g. Buja, Cook, and Swayne 1996; Wilhelm 2003). It allows the user to select cases of the data corresponding to an object or objects in one plot, typically by either clicking to select one object or clicking-and-dragging to select multiple objects within a region. The selected cases are then highlighted across all other “linked” 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.

When all plots contain objects with a 1-to-1 (bijective) mapping of the data only (e.g. all plots are scatterplots), linked brushing is simple, since we can just manipulate the colour attribute across the  $n$  points. Things become more difficult with 1-to- $k$  many-to-one mappings, however. For example, let’s imagine we have a linked barplot and a scatterplot. Then, we need to keep track of how which cases each bar in the barplot represents, so that whenever we select that bar we know which points to highlight. But more importantly, in the reverse scenario, when we select points in a scatterplot, how should the barplot respond? We could simply highlight every bar which includes any of the selected case. However, this kind of mapping does not preserve proportionality - a single selected point in the scatterplot will highlight an entire bar. Further, rather than the question of *whether any* cases in a category have been selected, a more interesting question may be *what proportion*. This question is also a strict upgrade in terms of information density, since if non-zero proportion of cases have been selected, we know that *some* have. Thus we may be interested in somehow comparing the summary (count) of the subgroups induced by the brushing (selected vs. non-selected cases).

## 4.2 Stacking vs dodging

There are many different ways of graphically displaying subgroups. One technique known as “dodging” or “grouping” (as in “grouped barplot,” Wilke 2019) consists of drawing the objects representing the subgroups side-by-side. For example, in our interactive barplot, we could draw two bars representing selected and non-selected cases side-by-side, for each category. Another option is “stacking”, that is, drawing the subgroup bars on top of each other. Typically, this means adding the upper y-axis coordinate of the previous subgroup to the upper and lower y-axis coordinates of the next (although more on this later, see Section TODO).

While dodging is often recommended over stacking when it comes to static visualizations, since it has better visual perception properties (Cleveland and McGill 1984; for a short elaboration in the form of a blogpost, see Kosara 2016), stacking seems to be the more popular technique in linked interactive visualizations. Why might this be? In static visualizations, the variable that induces the subgroups is fixed, and typically has all or most levels nested within most of the levels of the (supra-)category. In contrast, in linked IDVs, the variable that represents the selection changes with user interaction. As a result, the subgroups change rapidly and can become empty (i.e. either none or all cases are selected, within a single category). With dodging, this can lead to the bars “flickering in and out of existence”, and the whole plot may end up looking “ragged”, whenever some selection groups are empty. Further, if bidirectional linking is desired, in a dodged barplot it may not be obvious what the objects that can be selected are - will selecting one subbar select the whole category or just the intersection of that category and the selection?

In contrast, with stacking, the height of the whole stacked bar representing one category does not change as a result of brushing - only the subbars do. As such, the visual outline of the plot remains constant throughout time. Further, when it comes to selection, it seems more intuitive that a stacked bar represents one whole objects, compared to two side-by-side bars. Thus, stacking is results in more visually pleasing and intuitive figures which tax the visual system less.

Thus, stacking seems to be a good choice when it comes to implementing linked brushing. However, if we stack, we have to be very mindful of *what* we stack. In order to be meaningfully stackable, the summaries we represent in a plot need to have specific properties. This is the topic of the next section.

### 4.3 Case study: linked brushing of counts and means

Let's imagine we have three linked plots: a scatterplot, a barplot of counts within some categories, and a barplot summarizing the mean of some other variable within the same categories. To a naive user of IDVs, it might seem that both types of barplot would behave equally well under linked brushing and stacking. However, let's examine the situation in more detail.

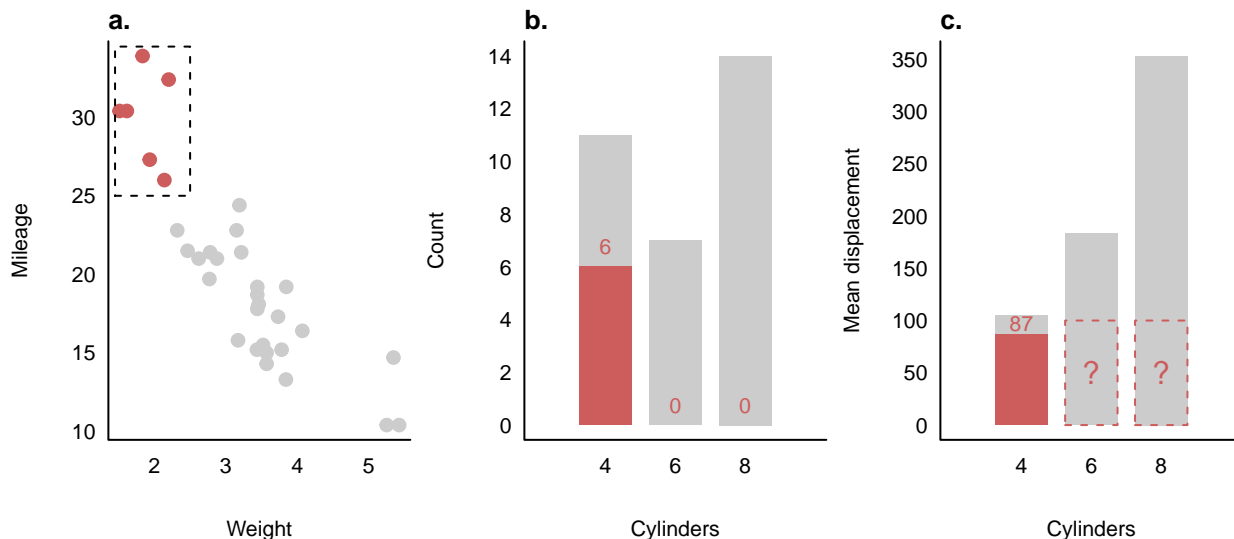


Figure 2: The problem of representing empty selection. a) An illustration of selection by linked brushing. b) In the barplot of counts, the count within an empty selection is zero and so an absence of a bar accurately represents the count of zero. c) In the barplot of means, the mean of an empty selection 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 (0, in this case).

#### 4.3.1 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 within the corresponding category is zero.

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.

#### 4.3.2 Subset vs. the whole

Second, as shown in Figure 3, how does the summary on the selection subset 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 stacked bar (because so is the count; see Figure ??fig:bigger)). Thus, the outline of the whole plot will not change with brushing. Further we can choose to draw the stacked bar in two different ways: either by “layering” bars on top of each other, starting with the tallest (whole category) bar, or by drawing the subbars as differences between their respective counts. The result will be visually identical.

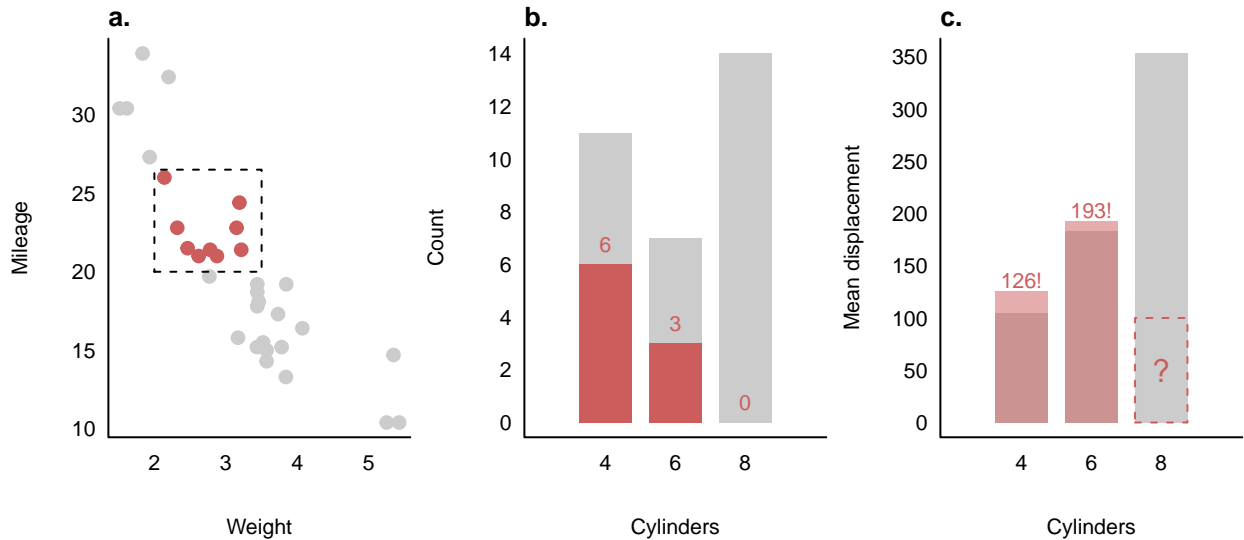


Figure 3: The relationship between subset vs. the whole. a) An illustration of selection by linked brushing. b) In the barplot of counts, the count within a selection (red) is always less than or equal to the count within the whole and the outline of the bars does not change. c) In the barplot of means,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

In contrast, the mean of a subset can be greater than the mean of the superset, and so the subbars representing the selection may end up taller than the bar representing all cases within the category. As a result, if we were to layer the bars on top of each other, we could end up with overplotting.

#### 4.3.3 Combining subsets

Finally, as displayed in Figure 4, when multiple selections/groups are present, can we combine the summaries in a meaningful way? This is the essence of stacking. For counts, there is the simple idiomatic way: simply sum up the counts within the category. The resulting quantity will be the same as if we had ignored the grouping variable.

For means, there is no such meaningful way to combine the statistics. If we were to apply the same operation as with counts and sum up the group means, the resulting statistic would not be meaningful. We could also try to take the mean of the group means, however, the resulting statistic would be different from the grand mean within the category.

#### 4.4 Case for restricting statistics?

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

- An unambiguous way to display empty selections (absence of a bar = a count of zero)
- A stable visual reference (the count within a sub-bar  $\leq$  the count within the whole bar)
- 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 will most likely be different from 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 underlying statistic.



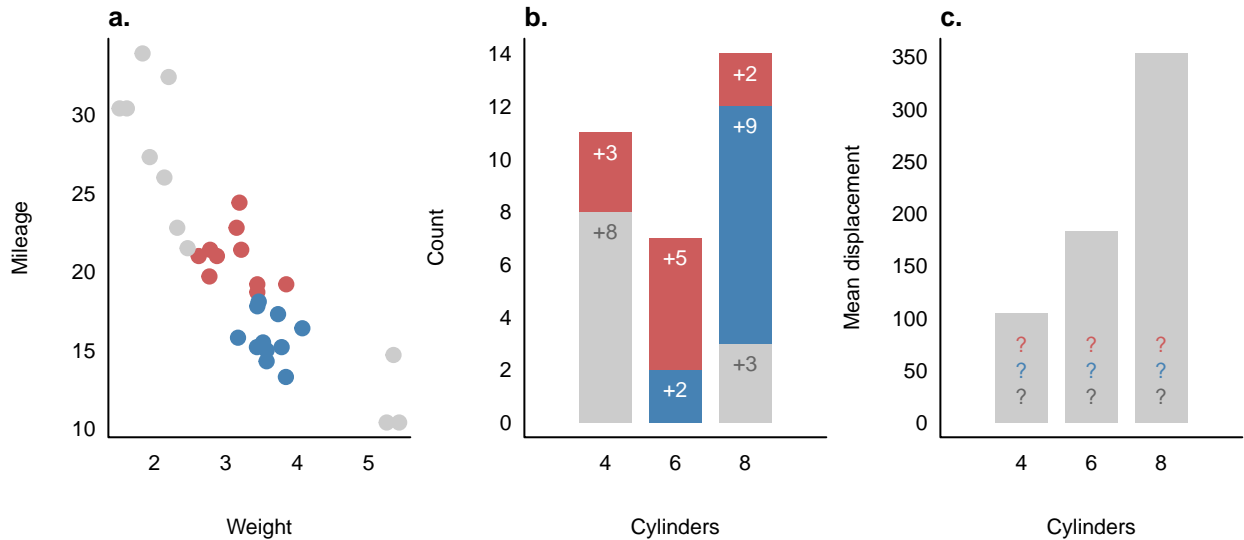


Figure 4: Combining selections. a) An illustration of selection by linked brushing, with two groups. b) In the barplot of counts, statistics can be stacked on top of each other such that the count of the stacked bar is identical to the count of the whole bar (i.e. one that would result from no selections).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

One might wonder if these problems are just a quirk of stacking. However, we 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 the summaries on non-empty selections (real number/float). While this may not be a major problem in most types of statistical software, it still does add some complexity to our code, since whatever functions use the summaries (such as the display function) now have to be generic over `float` and `NA/undefined/""`. 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`).

Claus Wilke summarized the problem as follows: “Stacking is useful when the sum of the amounts represented by the individual stacked bars is in itself a meaningful amount” (Wilke 2019, 52). This is a very concise and effective way to communicate the main point. However, it also misses something subtle yet important. Is “summing” the only way to implement stacking? The short answer is “no”. For example, products work just as well as sums for linked brushing of stacked objects, see Figure 5. In fact, there is whole a class of mathematical functions, or more precisely mathematical structures, that share the properties listed above. To discuss these, let’s first lay the groundwork with some theory.

## 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 these concepts, and I think it may provide a useful signpost for anyone who would want to dive deeper. I hope

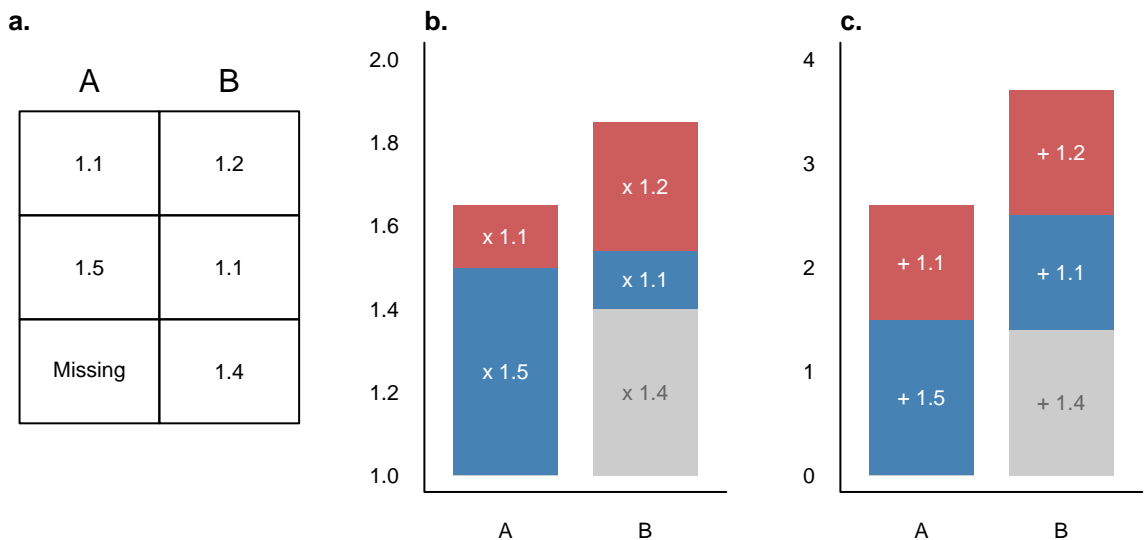


Figure 5: Sums and products work just as well as counts when used in an interactive barplot. Specifically, they provide a way of displaying empty selections, a stable visual reference, and the ability to combine subsets in a meaningful way. Note the different y-axis limits. a) The underlying data. b) Barplot of products. c) Barplot of sums

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.

Many of these ideas appear in Grammar of Graphics (Wilkinson 2012). Specifically, functions, partitions, and relations are all discussed therein. What is new however is the discussion of monoids, which is critical the core argument of the present text.

## 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  $f : X \rightarrow Y$  and  $g : Y \rightarrow Z$ , we can form a new function  $h = g(f(x)) = g \circ f$  (read “ $g$  after  $f$ ”) such that  $h : X \rightarrow Z$ .

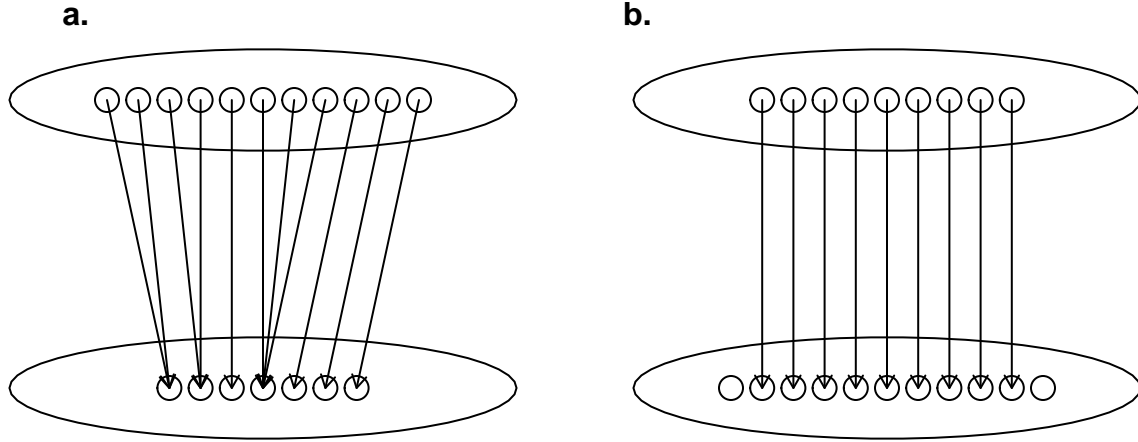


Figure 6: Two types of functions: a) surjective, b) injective.

## 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  $f : A \rightarrow P$ . 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:

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

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  $f_1(1) = x$ ,  $f_1(2) = x$ ,  $f_1(3) = y$ , 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 ( $X = \{\text{daughter, son, mother, father, grandmother, ...}\}$ ) 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  $\otimes : M \times M \rightarrow M$  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 in what order we do the operation (as long as the terms are in the same order - more on that below). 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,  $(\mathbb{N}, 0, +)$ :

$$1 + 0 = 0 + 1 = 1 \quad (\text{unitality})$$

$$1 + (2 + 3) = (1 + 2) + 3 = 1 + 2 + 3 \quad (\text{associativity})$$

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:

$$\text{"hello"} + \text{""} = \text{""} + \text{"hello"} = \text{"hello"}$$

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

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 of terms that we combine does not matter. This makes property 3., somewhat confusingly, similar to property 2., since that one is also about order, however, they are about the order of different things - associativity is about the order of operations, commutativity is about the order of terms. 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. If the underlying preorder is a poset (i.e. the elements of  $M$  satisfy comparability), then we speak of a commutative monoidal poset.

Finally, if the monoidal product also has an inverse (i.e. if  $m_1 \otimes m_2 = m_3$  then  $m_3 \otimes^{-1} m_1 = m_2$ ), then we speak of a *group*.

## 6 Fluent interactive graphics

With these ideas in place, we can try to outline something we provisionally call “fluent interactive graphics”. It starts from several axioms:

1. Linked brushing is useful
2. Ergo, every object in every plot should be selectable and able to represent selections
3. This should be congruous with other interactions (e.g. changing histogram binwidth/anchor)

### 6.1 Data and partitions

As before, we start with the data  $D$ . We know that we will want to summarize the data points  $d \in D$  via some summarizing function  $\alpha$ , 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 data point  $d \in D$  belonging to exactly one  $p \in P$ . There are two consequences to this. First, every data point  $d$  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* data point  $d$  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** (**managers**  $\cap$  **employees**  $\neq \emptyset$ , since every manager is also an employee). We could then, for example, choose the arrays as our parts  $p \in P$ , such that  $p_1 = \mathbf{employees}$  and  $p_2 = \mathbf{managers}$ , 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 data points 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**  $\times$  **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**  $\times$  **group** intersection variable). In this way, the partitions and the corresponding summaries computed on them form a directed acyclic graph (DAG), see Figure 7.

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 data point 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 data points. Besides, since the level 0 partition can never change (every data point 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

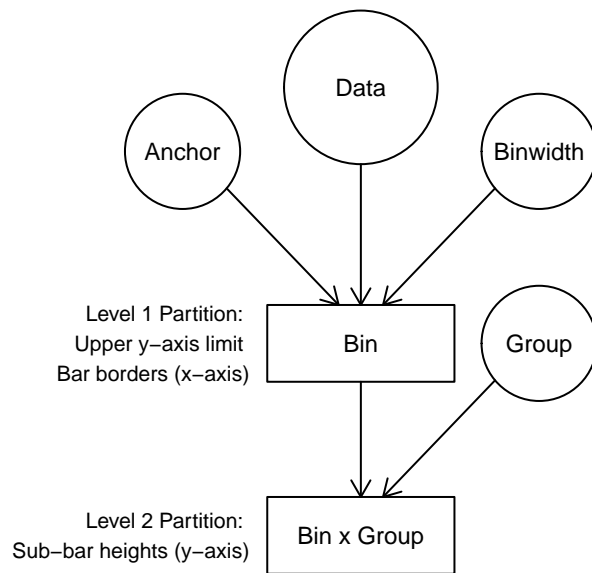


Figure 7: Partition hierarchy in a stacked histogram represented as DAG. Partitioning variables/factors are shown as rectangles. This dependency structure implies that, when group changes, only sub-bar heights need to be recomputed. However, when either the anchor or binwidth change, both the upper y-axis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

(e.g. bars defined by the **bin** variable), and not the object parts (e.g. sub-bars defined by **bin**  $\times$  **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**  $\times$  **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 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?

We propose that all summaries computed on the parts should be a monoidal posets. To rephrase Section 5.4, the summaries should consists of a set  $M$  that has elements that can be at least partially ordered, has a neutral element  $e$ , and is equipped with 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. 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 we want to set up things like so? Let’s break things down one by one.

First, why may it be useful for our summaries to be posets? As was discussed in Section 5.3, posets only need to satisfy three properties: reflexivity ( $x \leq x$  for all  $x \in M$ ) transitivity (if  $x \leq y$  and  $y \leq z$ ,  $x \leq z$  for all  $x, y, z \in M$ ), and comparability (if  $x \leq y$  and  $y \leq x$  then  $x = y$  for all  $x, y, \in M$ ). In plain words, this means that every element has to be at the very least comparable to itself, we can combine comparisons in an intuitive way, and if two elements compare in the same way one to the other and other to the one, then they are the same.

This definition permits a great number of possible summaries. For example, discrete summaries such as the levels/labels of a categorical variable (e.g. “Group A”, “Group B”, “Group C”, ...) are posets, albeit somewhat boring ones, since any of their elements is only ever comparable to itself (this special case is



also called a *discrete preorder*, see Fong and Spivak 2019). Typical numerical summaries represented by e.g. integers or floats are also posets, in this case *total orders* since we can compare *any* two elements. Coming up with a summary poset that is neither a discrete nor a total order is a bit harder, but we could again imagine the example of students ranked within schools, with each school using a different ranking measure (so we can say that, e.g. student *A* is the best student in their respective school, but we cannot compare students *A* and *B* if they each come from a different school).

The only real restriction of having the summaries be posets is that our summaries have to be comparable in a way that prohibits cycles: e.g. cycles such as  $A \leq B$ ,  $B \leq C$ , and  $C \leq A$  (technically, we should specify “cycles of length  $\geq 2$ ” since  $A \leq A$  is also a cycle). Consequently, this also means we could represent any summary via a DAG. This all seems reasonable, since the essence of data visualization is comparison: the graphical attributes of the objects we draw must either indicate no particular relation (e.g. if the “*female*” bar is left of the “*male*” bar in a barplot, this does not indicate that one category is “more” or “less” than the other, in any meaningful sense, outside of alphabetical order possibly) or be directly comparable (if the “*female*” bar is higher than the “*male*” bar, this may mean that there are, for example more females in the data than males).

The constraints imposed on the binary operation (the monoidal product  $\otimes$ ) are more restrictive, however, for certain types of interactions they may be useful or indeed necessary. *Unitality* says that there is a valid way of representing “nothing” in our summary. As we have seen in 4.3.1, this is useful since, for example, the absence of an object unambiguously indicates an empty selection. *Associativity* says that the order of operations should not matter. This is intuitively desirable: it should not matter whether we summarize our entire data set, or whether we first summarize some parts or “chunks” of it, and then summarize across those parts. Finally, *monotonicity* says that every summary should only ever “increase” along its axis.

Combined, these properties mean that every unit of the data or data point can be thought of as “visibly contributing” towards its respective summary. Conversely, we could imagine “recovering” the influence of single data point, by e.g. making a bar in a barplot of sums “shorter” by the data point’s value. If there indeed is such an inverse function to the monoidal product (e.g. “subtraction” being inverse to “summation”), then the summary is in fact not just a monoid but also a group. This is in fact how Crossfilter manages to implement fast single-case incremental updates, by requiring every grouped summary to also have an inverse (see the `group.reduce` function in the documentation, Crossfilter Organization 2023).

Requiring summaries to be commutative monoidal preorder also has computational advantages. It means that we can always compute all required 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  $k$  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  $j$ th collection on each step, with  $j$  being the factor level corresponding to the  $i$ th data point.

## 6.4 Displaying summaries through graphical attributes

TODO

## 7 Conclusion

The core argument of the present text has been that, when it comes to interactive data visualization, we need to think deeply about the statistical summaries underlying our plots. Conversely, no matter how tempting it may be, we cannot think of the objects in our plots simply in terms of their geometry. Instead, we have to consider the properties of the functions that underlie the geometric objects, and, based on these properties and the interactive features we want to implement, some functions may come up as better or more appropriate than others. Specifically, if we want to implement features such as general two-way linked brushing, the summaries we compute on the data may have to conform to certain algebraic structures, such as a monoidal poset.

This has strong implications for the design of interactive data visualization systems. Specifically, the systems should be structured in such a way that makes it easy to produce graphics which conform to the algebraic structure, and disallow or discourage the production of those which do not.

## 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](https://math.ucr.edu/home/baez/act_course).
- Bostock, Michael, Vadim Ogievetsky, and Jeffrey Heer. 2011. “D<sup>3</sup> 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>.
- Cleveland, William S, and Robert McGill. 1984. “Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods.” *Journal of the American Statistical Association* 79 (387): 531–54.
- 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>.
- jakevdp. 2019. *StackOverflow*. <https://stackoverflow.com/questions/56417028/altair-how-to-create-multi-panel-line-plot-with-linked-brushing>.
- Kosara, Robert. 2016. “Stacked Bars Are the Worst.” <https://eagereyes.org/blog/2016/stacked-bars-are-the-worst>.
- 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](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.
- Wilhelm, Adalbert. 2003. "User interaction at various levels of data displays." *Comput. Statist. Data Anal.* 43 (4): 471–94. [https://doi.org/10.1016/S0167-9473\(02\)00288-8](https://doi.org/10.1016/S0167-9473(02)00288-8).
- Wilke, Claus O. 2019. *Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures*. O'Reilly Media.
- 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.