Towards Fluent Interactive Data Visualization

Adam Bartonicek

2023-09-07

Contents

1 Introduction

There is a subtle yet profound question in the production and use of interactive data visualizations: *when we interact with a plot, what exactly are we interacting with?* On its face, the question might 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. However, in some way, this is just an illusion. The coloured rectangle on its own is just a geometric object, a visual representation of something else, that carries no information on its own. We can see this very easily - if we take the coloured rectangle and transpose it onto a blank area of the screen, we lose some crucial information that the context of the plot provides.

Geometric objects in plots are only useful within the context of the plot. That statement should not be surprising or controversial to people familiar with data visualization. So what is this context? First of all, we know that the objects in the plot must represent some underlying data - otherwise, the plot would not be of any use to use. Further, and crucially, when drawing plots, we rarely represent the raw data directly. Instead, we typically summarize the data in some way, by applying some mathematical function such as count, sum, mean, or quantiles. For example, the height of a bar in a barplot typically represents the count within the levels of some categorical variable.

So, when interacting with a bar in an interactive barplot, we are not just interacting with a geometric object, we are interacting with a mathematical function, or several of them. This is important since functions have properties. The core argument of the present text is that the mathematical functions we use to summarize the data impose limits on what kinds of visualizations and interactions we can meaningfully compose.

Before diving deeper, however, let's first define some key terms and draw a rough sketch of the data visualization process. To create a data visualization, be it static or interactive, we need several key ingredients: data, summaries, scales/coordinate systems, and geometric objects.

1.1 Rough sketch of the data visulization process

Firstly, every data visualization is built on top of some underlying data. We can represent this as a set of some arbitrary units of information D. Data in the wild usually comes with more structure than that - for example, we often encounter data stored in a tabular (or "tidy" [CITE]) format, stratified by rows and columns. In that case, we could substitute D by 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 description, we do not have to assume any special structure and just speak of the data units $d_i \in D$.

Secondly, the set of data units D is transformed into a set of collections of summaries S via some function α. The function α may be one-to-one (bijection), as in the case of a scatterplot, or, more often, many-to-one (surjection), as in the case of a typical barplot, histogram, density plot, or violin plot. When the function is many-to-one, it will typically reduce the cardinality of the set at hand such that $|S| \leq |D|$ (i.e. in a typical barplot, there will be fewer bars than there are rows of the data). This is done by stratifying on one or more variables which may either come from the data directly (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 s for a single boxplot "box" will consist of a median, first and third quartile, and the minimum and maximum of some variable, for a given level of some stratifying variable (which itself will also be an element of s). The output of these constituent functions may also depend on some external parameters, such as anchor and binwidth in a histogram.

Thirdly, each collection of summaries $s \in S$ needs to be translated from the data- (or summary-) coordinates to graphical coordinates/attributes $g \in G$, via a function β. This means that each summary value gets mapped to a graphical attribute via a constituent function, typically called a scale. Note that this mapping preserves cardinality - there are as many collections of summaries as there are collections of graphical attributes, $|G| = |S|$. For numeric summaries, scales typically 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. Scales may also provide additional non-linear transformations such as log-transformation or binning, or converting numeric values to colour values.

Finally, the collections of graphical coordinates $g \in G$ are drawn as geometric objects inside the plotting area which we can represent as the set of pixels P. While the act of drawing does take the graphical coordinates G as inputs, it does not simply produce outputs, but instead mutates the global state of the plotting area as a side effect, i.e. changing the state (colour) of pixels. As such, we cannot call it a true function, so let's just label it γ^*. The geometric objects may be simple, such as points, lines, or bars, or compound, such as a boxplot or pointrange. Importantly, each attribute necessary to draw the 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 such:

$$D \overset{\alpha}{\to} S \overset{\beta}{\to} G \overset{\gamma^*}{\Rightarrow} P$$

Or, equivalently:

$$\text{(data)} \overset{\text{summarize}}{\longrightarrow} \text{(summaries)} \overset{\text{translate/encode}}{\longrightarrow} \text{(graph. coordinates)} \overset{\text{draw}^*}{\Rightarrow} \text{(plotting area)}$$

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.

It should be mentioned that in some treatments of data visualization, the summarizing step ($\alpha : D \to S$) is de-emphasized, in one of two ways. In the first case, the data is already presumed to arrive pre-summarized (i.e. $S = D$, for example we can draw a barplot from a pre-summarized table of counts), and data visualization is just about encoding the data into graphical coordinates. This framing feels especially natural when considering plots which show a 1-to-1 mapping (bijection) between the data and the geometric objects, such as the scatterplot or the lineplot/time-series chart. Indeed, in these plots, the summarizing function is identity, and as such *can* be ignored. In the second case, the computation of summaries is considered, however, it is absorbed into the translation step, such that $\alpha = \beta$ (e.g., a histogram may be thought of as directly translating its underlying variable into bin coordinates). Both of these approaches produce a tight coupling between the summaries and graphical coordinates.

However, we can give a much richer account of visualizations by treating both the summarizing and the translating step as first class citizens. Crucially, some plots may encode the same summaries through different graphical attributes and geometric objects; others may compute different summaries but display them via the same means. As an example of the former, histograms and spineplots use the same summaries, i.e. binned counts of some underlying continuous variable x. However, histograms encodes the bin breaks into rectangle boundaries along the x-axis and the counts as the top rectangle boundary along the y-axis, whereas spineplot encodes the counts into both x- and y-axis rectangle boundaries, with the x- dimensions stacked and y-dimension scaled to 1, and encodes the bin breaks into x-axis labels (see Figure 1). As an example of the latter, scatterplot and bubbleplot both use points/circles to display their underlying summaries, however, these differ substantially.

There are also a few important caveats. Firstly, while not always the case, *order* can be important. Within each $s \in S$ and $g \in G$, some summaries may be ordered lists of values, as in the case of the lineplot (i.e. lines need to be drawn by connecting a series of points in the correct order). Secondly, the collections in S (and in turn, in G) may themselves be ordered and form a hierarchical or tree-like structure. This is particularly relevant in the case of stacking. For example, when drawing a stacked barplot, we need to stack the graphical coordinate values 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). Finally, the data limits (minimum and maximum) and values that are used for scales are often derived from S rather than from the raw data (e.g. the upper y-axis limit in barplot or histogram is the highest count across bins). Further, the limits may come from a higher level of hierarchy than the summaries that are actually being drawn - for example, in a stacked barplot, for the upper y-axis

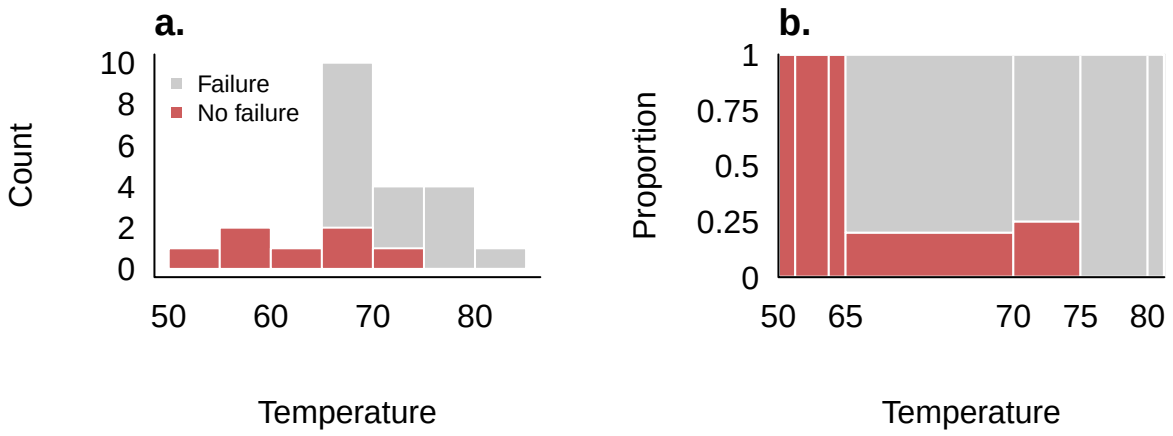

Figure 1: Histogram and spineplot use the same summaries but encode them in different ways. a) In histogram, the x-axis display bin breaks and y-axis displays count, stacked across groups (failure/no failure). b) In spineplot, the x-axis and y-axis both display count; the x-axis shows count stacked across bins, whereas the y-axis shows count stacked across groups and scaled by the total bin count (such that the total bin height = 1). The bin breaks are displayed as x-axis labels (however, the underlying summary is still stacked count).

limit we need to know the count (height) of the tallest *whole* bar but do not need to know the counts within the stacked sub-bars (since these are, by definition, smaller or equal to the whole bar).

This is where the differences between static and interactive visualizations become noticeable. In static visualizations, all computation is done only once, before the plot is rendered, and so the issues of order, hierarchical structure of the summaries, and the tracking of axis limits are less important. Interactive visualizations, on the other hand, need to reactively respond to the user's input, and some computations may need to be run many times within a single second. As a result, it is imperative to organize the process in such a way that only as little work as is neccessary is done.

2 Interactivity and Hierarchy

3 The Problem of Statistical Summaries

There is an even deeper issue when it comes to interactivity and statistical summaries of the data. Specifically, not every statistical summary "works" equally well - instead, some summaries may be better than other. Let's first illustrate the problem with an example.

Linked brushing or highlighting is one of the most popular types of interactive features used in interactive data visualizations [CITE]. It allows the user to select objects (such as points or bars) within one plot by e.g. clicking or clicking-and-dragging, and the corresponding cases (rows of the data) are then highlighted in all other plots. Its usefulness comes from the fact that allows the user to rapidly "drill-down" [CITE] and explore the summaries that would result from subsetting different rows of the data, within the context of the entire dataset.

Now, 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, within levels of the same categorical variable x. The plots are linked such that they allow for linked brushing/highlighting. Intuitively, it might seem that the barplot of counts and the barplot of means are equally valid/useful representations of the underlying data. However, if we consider these plots in the context of linked brushing, few subtle-yet-fundamental differences become apparent.

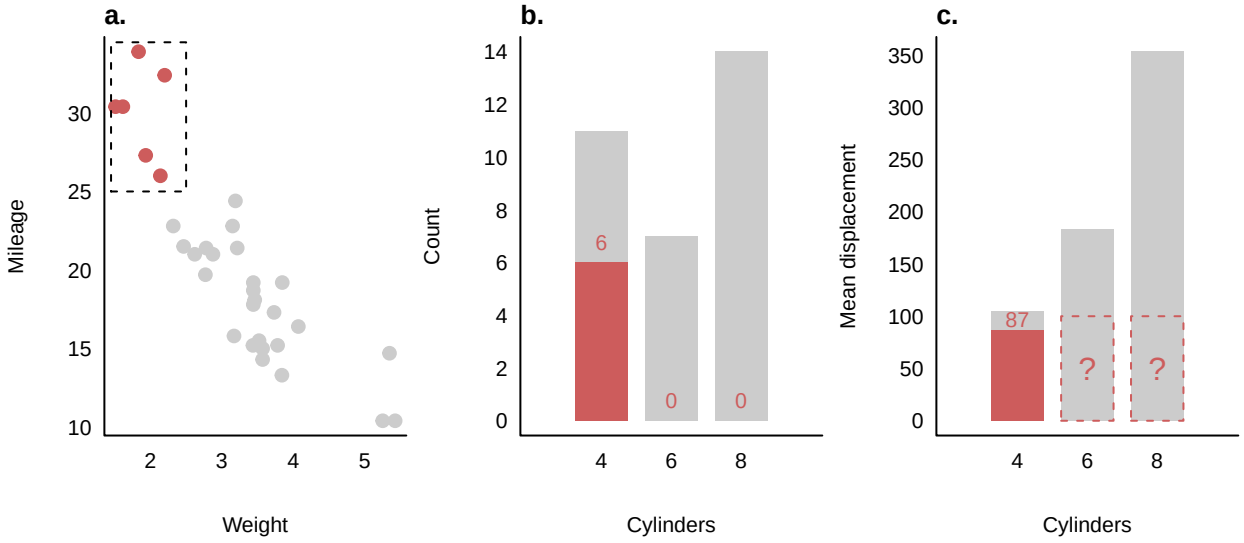

Figure 2: The problem of representing empty selection. a) An illustration of selection by linked brushing. b) In the barplot of counts, the count within an empty selection (red) is zero and so an absence of a bar accurately represents a count of zero. c) In the barplot of means, the mean of an empty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

3.1 Representing empty selections

Firstly, as is shown in Figure 2, how do we draw an empty selection? In the case of counts, we have a meaningful default value - zero - as in "the number of cases in an empty set is 0". When we seen an absence of a bar in a barplot, we know that the count for the corresponding level of the stratifying variables.

However, there is no similar default value for means: the mean of an empty set is not defined. We could just not draw the bar representing the empty selection, however, that decouples the statistical summary from the visual representation: the absence of a bar may now indicate that *either* no cases are selected *or* that some cases are selected and their mean is equal to the lower y-axis limit.

3.2 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 the whole bar will 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 of it, starting from the top y-coordinate of the selection bar and with **height** = (count of the whole − count of the selection). The resulting plots will be identical, visually.

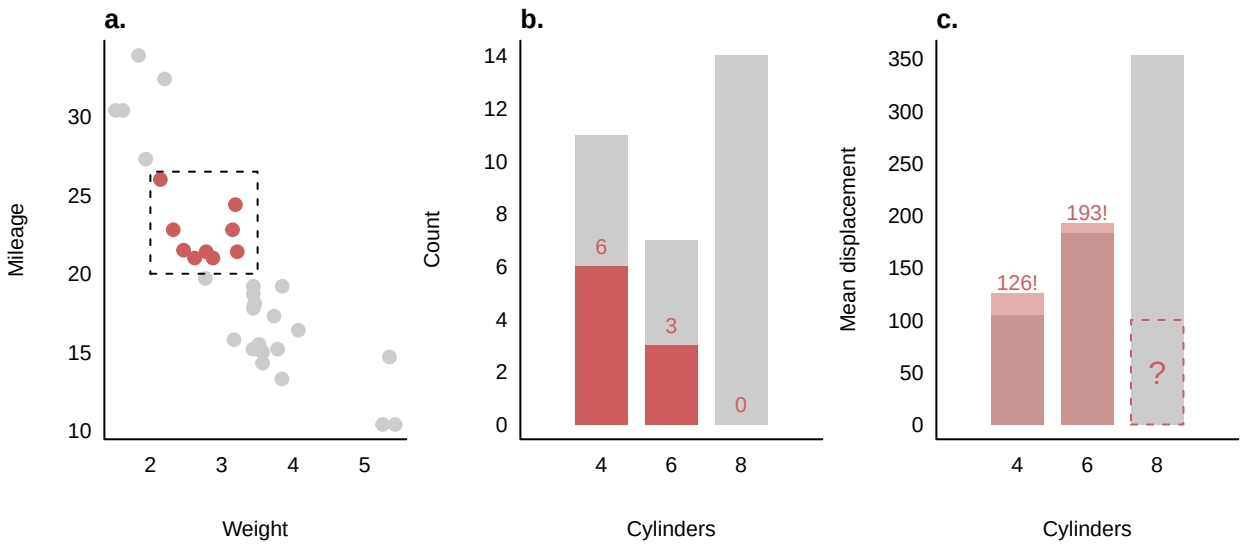

Figure 3: The relationship between selection vs. the whole. a) An illustration of selection by linked brushing. b) In the barplot of counts (middle), the count within a selection (red) is always less than or equal to the count within the whole and the outline of the bars does not change. c) In the barplot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

The barplot of means, does not share these nice properties. Specifically, the "sub-bars" can be taller than the whole bar (because the mean of a subset can be greater than the mean of the original set). As a result, if we draw the selection sub-bars on top of the whole bar, the whole bar may become completely obscured by it (as shown in Figure 3). We could choose to draw the selection sub-bars as semitransparent, or draw the the bars side-by-side instead of on top of each other (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 select individual bars or will brushing one select all of them?

3.3 Combining parts

Finally, as displayed in Figure 4, when multiple selections/groups are present, how do we combine them together? Again, in the case of the barplot of counts, there is an idiosyncratic way to do this: we can stack the counts across the selection groups and the corresponding bars on top of each other, and the height of the resulting bar will be identical to the count of the whole. And, as in the case of a single selection group, we can either draw the bars over each other, starting from the y-axis origin, or draw sub-bars stacked on top of each other, each starting where the last left off, and the resulting plots will be identical.

In the case of barplot of means, there is no meaningful way to combine the selections. If we were to stack the bars on top of each other, the resulting statistic (i.e. the sum of the group means) would not be meaningful. Worse yet, if we were to take the mean of the group means, the resulting statistic will almost surely be different from the mean of the whole: the mean of the group means $\neq$ the grand mean. And again, the grand mean may be less than any one 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, the bar width, and the selection.

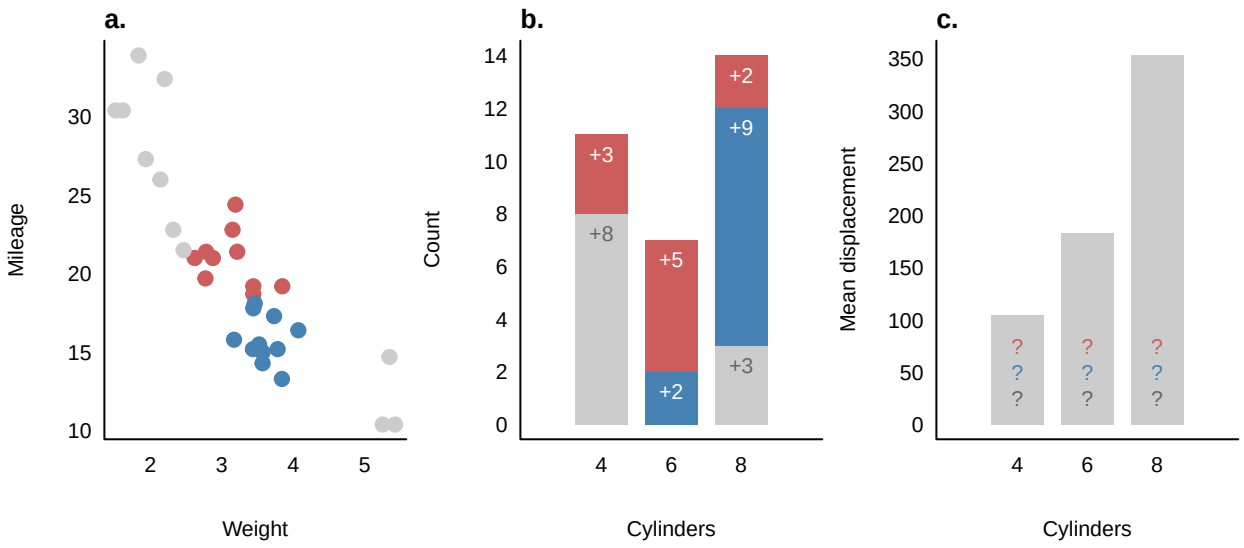

Figure 4: Combining selections. a) An illustration of selection by linked brushing, with two groups. b) In the barplot of counts, statistics can be stacked on top of each other such that the count of the stacked bar is identical to the count of the whole bar (i.e. one that would result from no selections).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

3.4 Statistics and their Properties

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

a. An unambiguous way to display empty selections (absence of a bar is a count of 0)
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 nice properties: mean of an empty selection is not defined, the mean of a selection cannot be subsumed into the mean of the whole, and the mean of group means is different from the grand mean. Importantly, these properties or lack thereof is not tied to any specific graphic but are instead the properties of the underlying statistic (count/mean).

There are, of course, ways to display means with linked brushing/selection (some have been described above). The key point, however, is that to display means, more decisions need to be made in order to produce a coherent interactive visualization. Consequently, since no choice is more "natural" than any other, a person who interacts with a linked barplot of means for the first time may be surprised in how the plot behaves. For the barplot of counts, on the other hand, there *are* such natural solutions readily available, and this may explain why barplots of counts are a popular type of plot in systems which implement linked brushing.

One might also wonder if these problems are just a quirk of linked brushing. However, first of all, the problems are not unique to interactive visualizations but arise in static plots, with stacking. Second, while they are perhaps the most prominent with linked brushing, we can run into them with any other interactive features, whenever we deal with statistics on several 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 what to do if some selections are empty. Here, there is perhaps a simple solution in displaying an empty string or an `NA` for the mean of empty selection, however, we should point out that by doing this, we are introducing a value of a different type (string/`NA`) from that of an non-empty selection (real number/float). This may not be a problem in whatever statistical

software we are using to render the plots, however, there is still something more natural about counts having the default value (0) be an element of the same type of as all other values (integer).

Returning to the nice properties of counts, are counts uniquely "good" in this regard? The short answer is "no". For example, sums or products (of values ≥ 1) conform to these same properties as counts, and so we could also display them with a stacked barplot and it would behave equally well in the context of linked brushing. In fact, there is whole a class of mathematical functions, or more precisely mathematical objects, that share these nice properties. To discuss these, let's first lay the groundwork with some relevant theory.

4 Few Relevant Bits of Category Theory

4.1 Functions

A function is a mapping between two sets. More specifically, given the set of sources S (also called the *domain*) and the set of possible targets T (also called the *codomain*), we can think of a function 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 then can be thought of as "selecting" a t for any valid s it is given.

If a function f covers all of its codomain, i.e. for all $t \in T$ there exists a $s \in S$ such that $f(s) = t$, then it is a *surjective* or *onto* function. If no two sources lead to the same 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. Also, for any given subset of targets, we can ask about the subset of sources or *pre-image* that produced them, i.e. for $T_i \subseteq T$ we can define $f^{-1}(T_i) = \{s \in S | f(s) \in T_i\}$. Finally, functions can be composed together: if we have two functions $f : X \to Y$ and $g : Y \to Z$, we can combine them into new function $h = g \circ f$ such that $h : X \to Z$, i.e. $h(x) = g(f(x))$.

4.2 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 of A.

4.3 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 one element is less than or equal to the other, or the two elements do not relate.

An example of a preorder is a 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 a familial relation. Thus, for example, **daughter** $\leq$ **father**, since the daughter is related to the father, and **father** $\leq$ **father**, since

a person is by definition 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 restrict 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 c., we speak of a partially ordered set or *poset*. If it conforms to both c. and d., then it is a *total order*.

4.4 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 function $\otimes : M \times M \to M$ called the *monoidal product*

Such that:

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

In simple terms, monoids encapsulate the idea that *the whole is exactly the "sum" of its parts* (where "sum" can be replaced by the monoidal product). Specifically, we have some elements and a way to combine them, and if we combine the same elements we always get back the same result, no matter what order we do it in. Further, we have some neutral element that when combined with an element yields back the same element.

An example of a monoid is summation of natural numbers $(\mathbb{N}, 0, +)$, i.e. $x{+}0 = x$ and $x{+}(y{+}z) = (x{+}y){+}z$ for all $x, y, z \in \mathbb{N}$. Likewise, products of real numbers are also a monoid, $(\mathbb{R}, 1, *)$, and so is matrix multiplication $(\mathbf{Matrix}, \mathbf{I}, \cdot)$, 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: $x^{(y^z)} \neq (x^y)^z$.

5 Producing Graphics

We start with the data D which is a set of values that may or may not be structured in some way. Our goal is to try and learn something new about the data by drawing geometric objects that represent some aspects of it visually. This seems fairly uncontroversial - if we draw geometric objects that are not related to the data in any way, we cannot expect to learn much from them.

Since our goal is to draw one or more geometric objects which represent the data, we need to think about how to distribute the information from the data among these objects. If we want to draw multiple objects, it would not make sense to allocate the same information to them - the objects would all look the same, and as a result, having a multiplicity of them would be redundant. Conversely, if we wanted to display some information about the whole data set, just one object would do. However, we often want to slice up the data along some axis or axes, to be able to compare and contrast multiple different objects. This is where the data having structure becomes important. Typically, data comes organized in a tabular $R \times C$ format where R is the set of rows and C is the set of columns (and $R \times C$ are the cells), and so we could use any of these three sets as the basis for drawing our geometric objects. This is shown in Figure 5. Notice that,

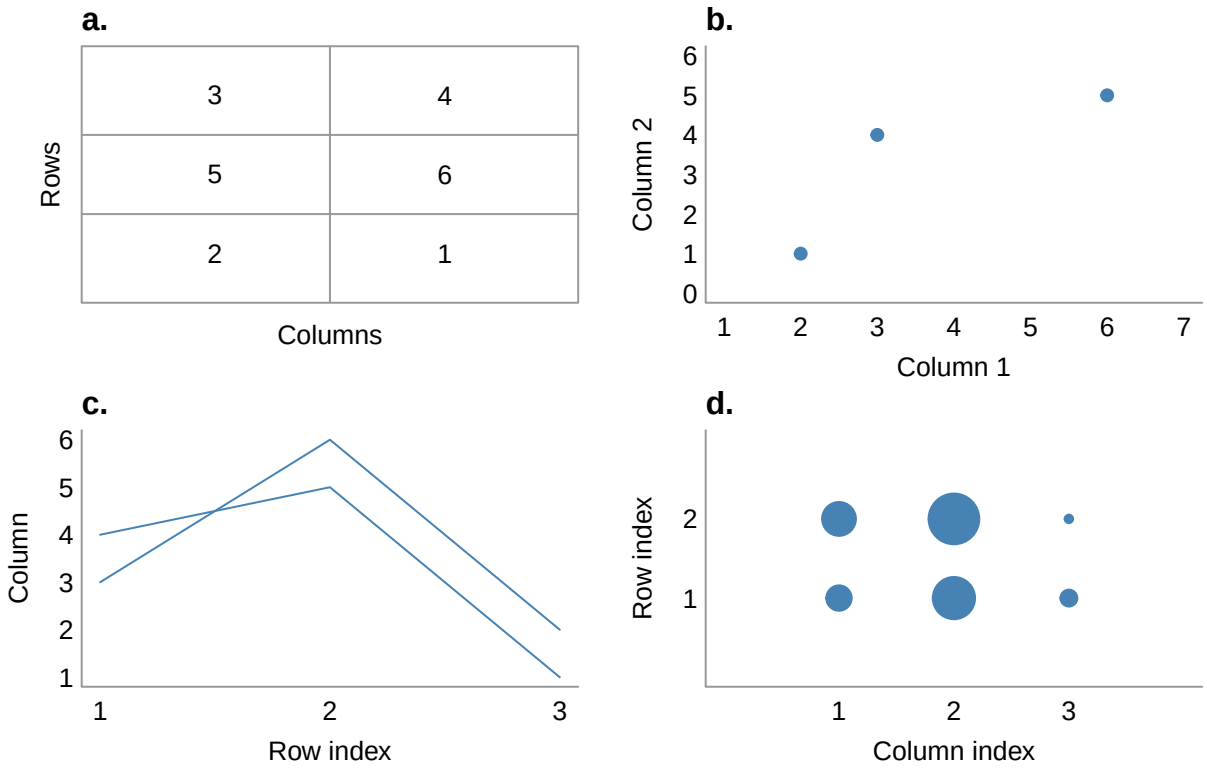

Figure 5: Three different ways of partitioning a tabular data set. a) Underlying data set represented as a matrix with 3 rows and 2 columns. b) In scatterplot, we slice by rows and points draw at the x- and y-position given by the columns. c) In lineplot, we can slice by columns & draw lines by connecting points given by the corresponding row values. d) In a dotplot, we can slice by both row and column and draw points at given row and column index, with size corresponding to the cell value.

when slicing along rows R, the number of objects (points) is equal to their cardinality: **# objects** $= |R|$. Likewise, when slicing along columns, **# objects** $= |C|$, and when slicing along cells, **# objects** $= |R \times C|$.

However, there is no reason to limit ourselves to slicing along axes that are already present in the structure of the data. We can also create new axes. For example, when computing the statistics underlying a classical barplot, we can slice along the levels of some discrete variable x. Likewise, when drawing a histogram or a heatmap, we can bin some continuous variable(s) and

We can formalize the process of producing graphics into several steps.

5.1 Create Partitions: Partition

First, we need to split the data into partitions. We can do this directly by treating some variables in the data as sets of part labels or *factors*, and indeed this will be the typical use case for discrete variables. Alternatively, we can also turn continuous variables into factors by e.g. binning or truncation, as in the case of a histogram or a heatmap. Either way, a convenient way to represent these factors is as tuples $(indices, labels, metadata)$, where $indices$ is an integer vector of length n (= the number of rows in the data) and values $j = 1, \ldots, k$, with ith index of value j assigning the ith row of data to jth part, $labels$ is a dictionary assigning each jth part a collection of part labels consisting of a set of key-value pairs (e.g. "level", "binMin", "binMax"), and $metadata$ being additional information shared across parts, also represented as key-value pairs (e.g. the list of all bin breaks within a histogram).

We can form finer partitions by taking the cartesian product of two or more coarser partitions. For example, given the variables *gender* with labels $\{(\textbf{male}), (\textbf{female})\}$ and *group* with labels $\{\textbf{A}, \textbf{B}, \textbf{C}\}$, we can form a product *gender* $\times$ *group* partition which will have the labels $\{(\textbf{male}, \textbf{A}), (\textbf{male}, \textbf{B}), \ldots\}$. The set of labels of the product partition will have maximum cardinality equal to the product of the cardinalities of the constituent partitions (i.e. $|gender \times group| = |gender| \cdot |group| = 2 \cdot 3 = 6$), although it may be convenient to only implement the product labels which actually occur in the data (for example, there may be no $(\textbf{male}, \textbf{B})$ records in the data, and so they do not have to be implemented). Importantly, we can also think of taking the product of two partitions as hierarchically nesting one partition within the other. By starting with the coarsest partition, i.e. one that assigns each row of the data to the same part, and then progressively nesting factors within one another, we can build up a hierarchy of partitions, for example J_1, J_2, and J_3 where J_1 is the set of part labels for the whole dataset $= \{(\ )\}$, $J_2 = labels_{gender} = \{(\textbf{male}), (\textbf{female})\}$, and $J_3 = labels_{gender \times group} = \{(\textbf{male}, \textbf{A}), (\textbf{male}, \textbf{B}), \ldots\}$.

5.2 Compute Statistics: Reduce

Second, for each jth part in a partition J, we need to compute a collection of summary statistics $s_j \in S$. If the summarizing functions that are used adhere to the monoidal contract (e.g. count, sum, product), this can be done in a single pass through the data or a *reduce* function, by looping through n rows of the data and for each ith row updating the jth part, where j is the corresponding ith factor index. If the summary statistic is not monoidal, we may be still able to compute auxiliary summary statistics that may be later used (in the *map* step) to compute the non-monoidal summary statistic, although breaking the monoidal contract in this way does come with the disadvantage of the representations no longer being guaranteed to be consistent.

5.3 Translate Statistics to Coordinates: Map

Third, we need to translate each collection of summary statistics $s_j \in S$ into a collection of graphical coordinates $g_j \in G$.

5.4 Combine Coordinates: Stack

6 Something else

Symmetric monoidal preorders also come with some4 computational advantages. Firstly, these summaries are guaranteed to be computable within a single pass through the data, and can also be updated online if new data arrives, without having to refer to the old. This does not necessarily make them superior to other summaries, such as mean or variance, which can also often be computed by collecting multiple constituent summaries in a single pass through the data and them combining them together in one step after (such as sum and count for the mean), and for which online algorithms also often exist too. However, the advantage is that every symmetric monoidal preorder is automatically single pass computable and online - we will never need to look for an algorithm. Secondly, if our plot implements reactive axis limits, since the count within the selection is always guaranteed to be less that or equal to the count within the whole, we only need to keep track of the counts on the whole. For example, in the case of the barplot of counts, we know that the upper y-axis limit will always be equal to the height of the tallest whole bar, and we can safely ignore the sub-bars. If there are N levels of the categorical variable and K selection groups, there will be N whole bars and $N \times K$ sub-bars. This difference might unimportant if N and K are small (as they are likely to be, in the case of the average barplot). However, if there are, for example, two categorical variables with N, and M levels, respectively, as in the case of a fluctuation diagram/bubbleplot/treemap, and both N and M have many levels, then having to iterate through all $N \times M \times K$ sub-bar summaries every time an interaction happens might become prohibitive.