Principled and Efficient Interactive Data Exploration with plotscape

Adam Bartonicek

2022-11-28

1 Introduction

> Note: If this is a pdf version of the thesis, the interactive graphics showcased in Section 4. Use will be rendered as static snapshots only, due to the limitations of the pdf format. The thesis is also available in fully interactive HTML format, and the source code for the Typescript version of the `plotscape` package can be found at https://github.com/bartonicek/plotscape (https://github.com/bartonicek/plotscape)

Humans learn about the world around them by interacting with it. In the same way, if we want to learn about our data, we need to be able to interact with it. Currently, there are many options for interactive data visualization within the R ecosystem. However, they all tend to suffer from the same shortcomings. Most importantly, while these packages make it easy to create attractive-looking individual plots with surface-level kinds of interactivity, setting up more interesting kinds of interactivity between multiple plots is challenging and requires specialized expertise. Furthermore, the design philosophy behind these packages seems to favor data presentation over than data exploration. Finally, while these packages seem to offer a host of features, many of them may not be conducive to learning from data effectively. There is a need for a general purpose interactive data visualization system that allows for fast and effective exploratory data analysis (EDA), and the purpose of the present thesis is to develop a prototype of such a system in `plotscape`.

2 Literature Review

Interactive data visualization is becoming more and more popular. Interactive figures appear everywhere, from news articles, to business-analytic dashboards, to science communication outlets such as journal websites and personal blogs. Yet, while data scientists and researchers may use interactive visualizations to present their findings, they rarely use them to explore their data and come up with the findings in the first place (Batch and Elmqvist 2017). This seems like a great wasted opportunity. Static visualizations are the bedrock of applied research and data science - why should the arguably more effective interactive data visualizations be left behind? To understand the puzzling trend of interactive data visualizations being used for presentation but not exploration, it is important to first lay some theoretical foundations, as well as to outline the broader historical context. Together, these lines of inquiry may provide answers regarding the present state of things as well as directions for the future.

2.1 What even is interactivity?

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

What is an interactive visualization? It may seem like it should be easy to answer this question, given that interactive visualizations are so widely used. Unfortunately, that does not seem to be the case. When researchers and applied practitioners say “interactive visualization”, they tend to refer to many different things across many different contexts.

Firstly, it is necessary to disambiguate whether we refer to “interactive visualization” as a noun, a thing that exists, or to “interactive visualization” as a process, an action undertaken by a human being. Pike et al. (2009) note that “interaction” is an overloaded term that can refer to either the concrete set of tools through which users manipulate the visual information or to the more abstract “human interaction with information” - the back-and-forth between the user and the visual information presented to them (see also Yi et al. 2007). Which definition is used depends largely on the field: statisticians and computer scientists tend to talk about interactive visualizations as things, whereas researchers in the fields of cognitive science and human-computer interaction tend to emphasize interactive visualization as action, although there is also considerable overlap. While the cognitive aspect of humans interacting with visual information is definitely interesting, for the purpose of this thesis, I will mainly discuss “interactive visualizations” as things, i.e. visual objects that live on a computer screen.

Yet, even if we narrow down the focus on interactive visualizations as visual objects, there is still a lot of ambiguity. Some researchers define interactive visualizations very broadly. For example, there are those that talk about interactive visualizations as any visualizations can be actively manipulated by the user (Brodbeck, Mazza, and Lalanne 2009). To others, the key thing about interactivity is time, or rather, the lag between the user’s input and changes to the visualization, with less lag meaning more interactivity (Becker and Cleveland 1987; Buja, Cook, and Swayne 1996). Some even make the distinction between “interactive” and “dynamic” manipulation, where interactive manipulation happens discretely such as by pressing a button or selecting an item from a drop-down menu, whereas dynamic manipulation happens continuously, in real-time, for example by smoothly moving a slider or by clicking-and-dragging (Rheingans 2002; Jankun-Kelly, Ma, and Gertz 2007). What these two definitions - which I will label the “basic” and “temporal” - have in common is that they impose relatively few restrictions on what kinds of visualizations can be considered interactive. For example, one could argue if the user runs a code from a command line to generate a new plot, this could be considered interactive visualization, under these definitions.

In contrast, other researchers consider the category of “interactive visualizations” to be much narrower. For many, the defining features are the ability to query different parts of the dataset (by e.g. zooming, panning, and filtering), and the propagation of changes between connected or “linked” parts of the visualization (Kehrer et al. 2012; Buja, Cook, and Swayne 1996; Keim 2002; Unwin 1999). Similarly, in Visual Analytics (VA), a distinction is made between “surface-level” (or “low-level”) and “parametric” (or “high-level”) interactions, where surface-level interactions manipulate attributes of the visual domain only (e.g. zooming and panning), whereas parametric interactions manipulate attributes of mathematical models or algorithms, underlying the visualization Pike et al. (2009). These definitions of interactivity - which I will label “querying”, “linked”, and “parametric”, respectively - can be used to classify considerably more complex objects. Specifically, while some of the querying interactions like zooming and panning can be implemented by manipulating graphical attributes of the visualization only, linked and parametric interactions require special underlying data structures in all but the most simple cases.

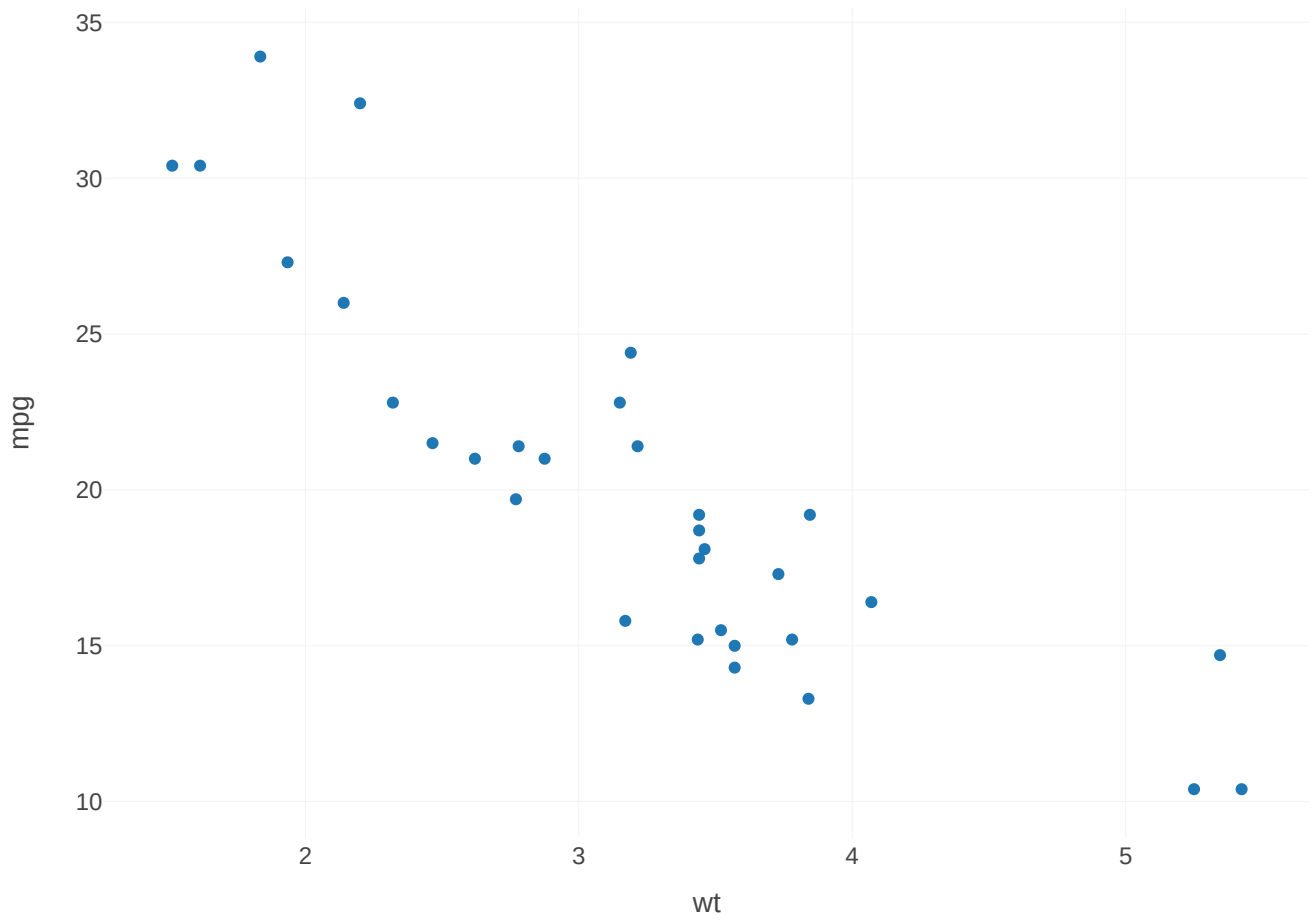
To sum up, the term “interactive visualization” means a lot of different things to a lot of different people. Below, in Table 2.1, I have attempted to summarize the main types of interactivity of data visualizations. This list is not supposed to be exhaustive as more complete and detailed taxonomies of interactive visualizations have been described (see Yi et al. 2007). Instead, the point of this list is to provide a rough sketch of the space interactive visualizations live in:

Table 2.1: Types of interactivity

Type	Short definition	Details
Basic	Change happens at all	The user can interact with the visualization in some way
Temporal	Change happens “in real-time”	There is little lag between the user’s interactions and changes to the visualization, real-time change is sometimes called dynamic.
Querying	Change relates to querying different parts of the dataset	The user can query different parts of the dataset by interacting with the visualization, analogous to subsetting rows of the data (e.g. zooming, panning, and filtering)
Linked	Change propagates	Parts of the visual display are connected or “linked”, such that the user’s interaction with one part of the visualization produces a change in another (e.g. linked brushing)
Parametric	Change affects underlying models	The user can manipulate the underlying mathematical structure behind the visualization, not just its surface-level graphical attributes
(Cognitive)	Change is perceived by a human	There is a back-and-forth between the visual information being presented and the user’s cognition, insights are generated

Using the term “interactive” to cover such a wide range of visualizations necessarily leads to ambiguity. For example, imagine we have a single scatterplot. The user can click-and-drag on this scatterplot to highlight certain points (i.e. brush). Should this scatterplot be considered “interactive”? Under the most basic definition, yes, since the user *can* affect the visualization through interaction. However, some researcher may not be satisfied with that and may ask us to tell them whether the visualization meets linked or parametric defition of interactivity. With only a single plot, there’s nowhere for our interactions to propagate to, and changing the color of individual points can hardly be considered a change to a mathematical model, so our verdict would then have to be “not interactive”. What if the researcher was a fan of the temporal definition? If we assume the results of the user’s interactions render smoothly enough, we could certainly call the plot interactive. However, what if more data is added to the plot and the user now has to wait several hundreds of milliseconds, or even seconds, before the interaction is rendered? Does an interactive visualization stop being interactive as a result of our computational resources being reduced? Therefore, even with such a simple example, there are many arguments that could be made for and against calling a plot an “interactive visualization”.

```
library(plotly)
plot_ly(mtcars, x = ~wt, y = ~mpg)
```



> Is this an interactive visualization? Depends on who you ask.

It is possible to rank these types of interactivity on different metrics. The most obvious is perhaps the programming and computational complexity: how large codebase and how much computational resources is required implement this type of interactivity? Clearly, the basic or surface-level definition is the least restrictive, and so the simplest and the computationally most cheap interactive visualization systems will be basic. Refering back to the example of the “dubiously interactive” scatterplot, we can implement the highlighting by assigning each point in it a color attribute. Then, whenever the user click-and-drags, all we need to do is to manipulate that color attribute. We do not need to refer back to the data or do any additional computation, and this makes the interaction very cheap. Things get a bit more complicated with temporally interactive visualizations, as there is now the requirement that the visualizations respond fast enough to the user’s interactions. This adds some computational complexity, especially when the data volume is high, however if the interactive action is simple then this can still be done very efficiently. Next, querying interactivity can be somewhat more complex again, since while zooming and panning can be implemented very cheaply by manipulating only the graphical attributes of the visualization (i.e. axis limits) only, filtering requires an additional variable to keep track of which cases in the data are currently filtered. Finally, linked and parametric will usually be the most complex, both programmatically and computationally, as they typically require extra data structures and online computations. However, there is nuance. For example, if we want to implement linked plots that all show one-to-one representations of the cases in the data (e.g. points on a scatterplot), we might be able to get away with a single variable that tracks which cases are selected, similarly to filtering. However, if we also want to include plots with many-to-one representation of the data (e.g. barplot: the count of many cases describes the height of one bar), then we need to be able to compute the relevant summary

statistics (height of the bar) on the fly. That is, in addition of having a variable which keeps track of which cases are selected, the plots also need to "remember" how their summary statistics are computed and be able to re-do the computation each time the selection status changes.

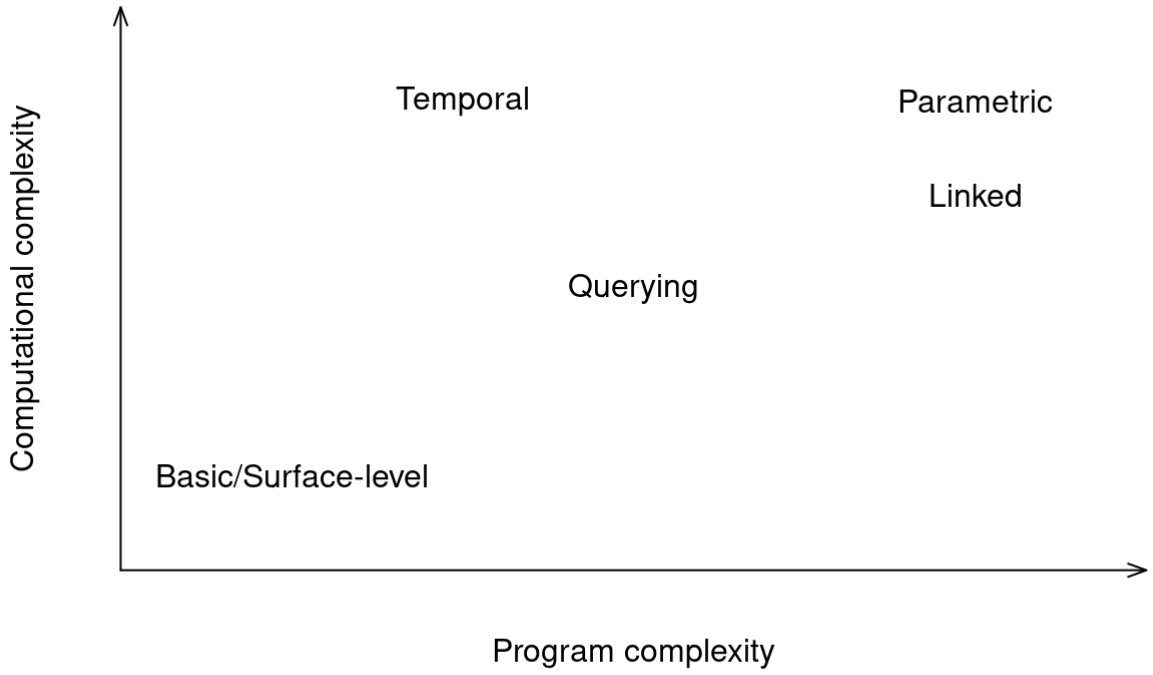


Figure 2.1: Programming vs. computational complexity of interactive visualization types

Another way to rank the types of interactivity is by their usefulness. I argue that this is inversely proportional to the programming and computational complexity. Parametric and linked interactions are the most useful since they allow the user to get a high-level understanding of the patterns and interactions within the data (Pike et al. 2009; Leman et al. 2013). Being able to ask questions such as: "which bar on a barplot do these points on a scatterplot belong to?" is akin to the conditioning operator in probability, and as such extremely useful. Querying interactions are fairly useful too, since the allow the user to "drill down" and answer questions about specific parts and subsets of the dataset (Dix and Ellis 1998). Temporal interactivity is useful in a "necessary but not sufficient" way, in that a large lag between the user's input and changes to the graphics renders any kind of interaction moot, but a visualization is not guaranteed to be useful, no matter how smoothly it renders. Finally, basic interaction may add little usefulness above and beyond the static plot that would exist without it.

2.2 Brief History of Interactive Data Visualization in Statistics

Static data visualization has a rich and intricate history, and a full treatment would be beyond the scope of the current thesis (but see e.g. Dix and Ellis 1998; Friendly 2006; Friendly and Wainer 2021; Young, Valero-Mora, and Friendly 2011). Suffice it to say, prior to the second half of the 20th century, data visualization was largely seen as at best secondary to "serious" statistical analysis, although there were indeed some prominent examples (Friendly 2006; Young, Valero-Mora, and Friendly 2011). However, starting at the end of 1950's, a series of developments that lead to a great increase in prominence of data visualization took place. Firstly, at the theoretical level, the work of Tukey (1962; 1977) and Bertin (1967) established data visualization as an independent and valuable discipline in its own right. Secondly, at the practical level, the development of

personal computers (see e.g. Abbate 1999) and high-level programming languages, most notably FORTRAN in 1954 (Backus 1978), introduced powerful and widely available tools that rendered the production of figures near-effortless, in comparison to the earlier hand-drawn techniques. Combined, these developments lead to a surge in the use and dissemination of data visualization.

Interactive visualizations would not be left far behind. Early systems tended to be specialized for one specific task, such as the one of Fowlkes (1969) which allowed for interactive viewing of probability plots under different choices of parameters and transformations, and that of Kruskal (1965) which visualized multidimensional scaling. The first more general-purpose system was PRIM-9 (Fisherkeller, Friedman, and Tukey 1974), which allowed for exploration of high-dimensional data via projection, rotation, subsetting and masking. The systems that came after grew on to become even more general and even more ambitious. For example, MacSpin (Donoho, Donoho, and Gasko 1988) and XGobi (Swayne, Cook, and Buja 1998) provided features such as interactive scaling, rotation, linked selection (or “brushing”), and interactive plotting of smooth fits in scatterplots, as well as interactive parallel coordinate plots and grand tours. Excellent showcase videos of many of these interactive data visualization systems are available at ASA Statistical Graphics Video Library (https://community.amstat.org/jointscsg-section/media/videos).

Later systems saw even greater flexibility and also integration into general statistical computing software. The successor system to XGobi, GGobi (Swayne et al. 2003), expanded on XGobi and made it directly embeddable in R. Mondrian (Theus 2002) built on Java allowed for sophisticated linked interaction between many different types of plots such as scatteplots, histograms, barplots, scatterplot, mosaic plots, parallel coordinates plots, and maps. Finally, iPlots (Urbanek and Theus 2003) implemented a general framework for interactive plotting that was not only embedded in R but could be directly programmatically manipulated, and was further expanded and made performant for big data in iPlots eXtreme (Urbanek 2011).

What all of these interactive data visualization systems had in common is that they were designed by statisticians and with interesting, often ambitious interactive features in mind. Linked selection and brushing, rotation and projection, and interactive manipulation of model parameters made frequent appearances. While being a clear strength, the more complex nature of the systems may have also slowed down their adoption, as they often demanded greater and more specialized expertise.

2.3 Current-Age Interactive Data Visualization and the Rise of Web Technologies

The developments of interactive visualization within the field of statistics were mirrored by similar developments coming from the field from Computer Science. Most relevantly, the rise of Web technologies the mid 1990’s, and more specifically the appearance of JavaScript in 1995 as a high-level general-purpose programming language for the Web (for a thorough description of the history, see e.g. Wirfs-Brock and Eich 2020), created an extremely versatile toolbox for highly-reactive and portable applications. That is, JavaScript was created with the explicit purpose of making websites interactive, and the fast dissemination of robust and standardized web browser software meant that interactive websites could be viewed by anyone, from anywhere. Interactive data visualization became just one of many interactive technologies highly sought after within the fledgling Web ecosystem. Early systems such as Prefuse in 2005 (Heer, Card, and Landay 2005) and Flare in 2008 (“Flare | Data Visualization for the Web” 2020) relied on plugins, specifically Java and Adobe Flash Player, respectively.

In the late 2000’s and early 2010’s, several true Web-native interactive data visualization systems emerged. The most prominent among these is D3.js (Bostock 2022), a broad and general JavaScript framework for manipulating HTML documents using data and displaying it with visualizations. D3.js visualizations rendered as scalable vector graphics (SVG), meaning that they are lossless and retain a crisp look even when resized. However, a price to pay for this robustness is in performance, as visualizations can become slow to render at high data volumes. D3.js spawned a number of smaller, more specialized data visualization packages, such as plotly.js (Plotly Inc. 2022). Another popular JavaScript library for interactive visualizations is Highcharts

("Interactive javascript charts library" 2022). Like D3.js, Highcharts visualizations are also rendered in SVG by default, although there are also more performant rendering technologies based on WebGL available ("Render millions of chart points with the Boost Module Highcharts" 2022).

While these contemporary Web-based interactive data visualization systems offer great deal of flexibility and customizability, I argue that this comes at the cost of making them practical for applied researchers and data scientists. Most importantly, it seems that the conceptual ambiguity about what counts as "interactive visualization" described in Chapter 2.1 has propagated into the implementation. For example, in the R Graph Gallery entry on Interactive Charts (https://r-graph-gallery.com/interactive-charts.html) (Holtz 2022), which features several examples of interactive visualization derived from the above-mentioned JavaScript interactive data visualization libraries, the visualizations feature interactions such zooming, panning, hovering, 3D rotation, and node repositioning within a network graph. However, in all of the examples, the user only affect surface-level graphical attributes of a single plot, and so these visualizations do not meet the linked and parametric definitions of interactivity. In contrast, the Plotly Dash documentation page on Interactive Visualizations (https://dash.plotly.com/interactive-graphing) (Plotly Inc. 2022) does feature two examples of linked hovering and cross-filtering, i.e. examples of linked interactivity. However, it should be noted that vast majority of visualizations in the Plotly R Open Source Graphing Library documentation page (https://plotly.com/r/) (Plotly Inc. 2022) allow for only surface-level interactions. Similarly, VegaLite Gallery pages on Interactive Charts (https://vega.github.io/vega-lite/examples/#interactive-charts) and Interactive Multiview Displays (https://vega.github.io/vega-lite/examples/#interactive-multi-view-displays) (Vega Project 2022) feature many examples, however, only a few show limited examples of linked or parametric interactivity. Finally, the Highcharter Showcase Page (https://jkunst.com/highcharter/articles/showcase.html) (Kunst 2022) does not feature any examples of linking or parametric interactivity.

What all of the packages listed above have in common is that most featured interaction is typically surface-level and takes place within a single plot, and the few examples that feature interesting types of interactivity (linked or parametric) often require a complicated setup. The main reason for this is most likely that all of these packages have been designed to be very general-purpose and flexible, and the price to pay for this flexibility is that complex types of interactivity require complex code. Another reason is that these packages have been built for static visualizations first, and interactivity second. Further, since all of these packages are native to JavaScript, the expectation may be that if more interesting types of interactivity are desired, the interactive "back-end" may be written separately, outside of the package. Finally, the typical use case for these packages seems to be presentation, not EDA.

Be it as it may, there is a fairly high barrier for entry for creating interesting types of interactivity (i.e. linked or parametric) with these packages. This may not be an issue for large organizations which can afford to hire computer science specialists to work on complex interactive dashboards and visualizations full-time. However, to the average applied scientist or data scientist, the upfront cost of producing a useful interactive data visualization may be too high, especially if one is only interested in exploratory data analysis for one's own benefit. This may be the reason why interactive visualizations are nowadays mainly used for data communication, not data exploration (Batch and Elmqvist 2017). On a higher level, the current options for interactive data visualization may reflect a broader cultural differences between Computer Science and Statistics, where Computer Science may be more oriented towards business and large-team collaboration, whereas Statistics may be more focused on applied research and individual/small-team workflow.

3 Design

> Graphical excellence is that which gives to the viewer the greatest number of ideas in the shortest time with the least ink in the smallest space
>
> The Visual Display of Quantitative Information (Tufte 2001)

The aim of `plotscape` is to provide a robust and efficient tool for exploratory data analysis (EDA). The profile of the hypothetical average user is someone who works with data a lot, has little spare time, and wants to understand the data as well as possible. As such, the broad design philosophy of `plotscape` is to provide an intuitive tool with a flat learning curve that can be easily used out-of-the-box.

3.1 Graphic Design

> [A good visualization] is "boring" in the right way. It's not trying to do too much, so you can focus on the message that is being communicated. […] I don't want [the visualization] to be "pretty", I want it to be visually pleasing enough that it doesn't distract [from the information being presented].
>
> Visualization Q&A Livestream (Betancourt 2020)

Graphic design is an important consideration since if a user finds the visualization unattractive, they will not want to spend as much time with it. While data should always come first, pure visuals are also an important part of the user experience. Therefore, it is necessary to take the different considerations of graphic design into account.

Firstly, when a user looks at a `plotscape` visualization, it is important that they perceive it as a coherent, unified whole. This is to reinforce that fact that the individual plots and other graphical elements are inextricably linked via a shared data structure. Making the interactive scene look visually unified can be achieved through several means. For example, a consistent color scheme should be used across the different plots and other elements that make up the visualization. Likewise, sizing of the different elements should also be consistent. Finally, the results of interaction should also be consistently implemented across different plots: if clicking and dragging shows up as linked selection in one plot, it should not present a different visual result in another.

Second and more generally, `plotscape` should also adhere to the big four principles of graphic design: **contrast**, **repetition**, **alignment**, and **proximity** (CRAP, Williams 2004). **Contrast** is intimately connected to the well-known figure-ground phenomenon in cognitive sciences, whereby sharp borders and edges tend to draw attention from the brain's visual system (see e.g. Qiu, Sugihara, and Heydt 2007). Thus, good use of contrast directs the viewer's attention and this makes it one of the most important concepts in graphic design (Reynolds 2011). To properly leverage contrast, the areas of the visualization that we want to draw the user's attention to should present large visual contrasts, and, conversely, the less important parts of the visualization should appear more visually uniform. As a concrete example, when highlighting as a result of linked selection, the highlight color should contrast strongly with the base color. I have tried to achieve this by using a muted (e.g. light gray) color for the base and a more vibrant color (e.g. green) for the highlight. Contrary to contrast, **repetition** provides a sense of visual unity and consistency. This is achieved in `plotscape` fairly automatically as all plots share the same graphical parameters, for example the same background color, object colors, font, etc… Similar to repetition, the purpose of **alignment** is to create a pleasing visual unity and a sense of connection between the aligned graphical elements (Williams 2004). Again, in `plotscape` alignment is resolved in a fairly straightforward way, as the default layout of plots within the visual scene is a rectangular grid with $\lfloor \sqrt{N_{plots}} \rfloor$ rows and $\lceil N_{plots} / \lfloor \sqrt{N_{plots}} \rfloor \rceil$ columns (where $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ indicate the floor and ceiling functions, respectively). I have plans to also implement a more sophisticated layout system based on the CSS grid functionality soon. Finally, the principle of **proximity** states that related things should be grouped together. In `plotscape` , all plots are related through the shared underlying data structure, so they should all be located close to each other. Again, this can be achieved via the rectangular grid. One could also ask if there is any other natural spatial ordering to the plots, outside of them all being related to each other. I think that, without specific domain knowledge about the data at hand, the answer is "no". The user can, however, arrange plots within the visual scene based on their own perception of relatedness between the different types of plots and the variables being presented.

Third, as is well known in the data visualization literature and graphic design literature in general, absence of graphical features can be equally as valuable as their presence (or arguably moreso). In the cult classic book on the design of comics and graphical novels, Understanding Comics: The Invisible Art, Scott McCloud (1994) repeatedly stresses the importance of not just the things that appear on the page but also the ones that are purposefully left out. For example, McCloud demonstrates how closure - the gap between the panels of a comic strip - is crucial for conveying meaning. In a similar vein, Tufte (2001) showed how the absence of non-data elements such as gridlines and axis ticks can greatly enhance the clarity and effectiveness of data visualization. Further, Tufte also disparaged about chartjunk, that is, completely redundant non-data elements and decorations that do nothing but distract from the data. While Tufte discussed these concepts in relation to static visualizations, other authors have also warned about the danger of regressing back to “artistic” rather than informative interactive visualizations as computational resources become cheaper (Dix and Ellis 1998). To summarize, in static visualization there is a danger in not just showing too little but also showing too much, and since an interactive visualization is static visualization at any single moment of time, the same principles apply here. Therefore, the default graphical parameters in `plotscape` are chosen to create a clean, fairly austere look, as not to distract from the data. In similar vein, I have tried to minimize the presence of non-plot elements (such as sliders/drop-down menus) in the visual scene, by binding most interactive actions to either mouse or keyboard commands, and presenting only a single small help button in the top-right corner, so that the user can look these commands up on the fly.

3.2 User Interface

Since `plotscape` is designed with exploration rather than presentation in mind, it follows that the average user will be more likely an applied researcher or data scientist rather than programmer specialized in creating intricate interactive visualizations. As such, much of `plotscape` ’s programming interface should be streamlined, at the cost of some generality. That said, the user should always have the option of customizing and extending their experience provided they have the necessary expertise. However, for the average user, as much of the implementation as possible should be hidden behind a facade of sensible defaults.

On the programming side, the user should be able to call simple, intuitively-named functions from R to produce `plotscape` scenes quickly and efficiently. The code should be simple enough so that the user can iterate through different visualizations until they converge at a scene that the greatest amount of information can be extracted from. Only a fairly basic knowledge fundamental R concepts such as functions, vectors, dataframes, and objects should be necessary to use `plotscape` effectively. If a user is more experienced, they may be able to add R-side JavaScript code, or even extend the TypeScript/JavaScript source code itself; however, no knowledge of TypeScript/JavaScript should be necessary for basic use.

On the interaction side, the user’s interactions with a `plotscape` scene should be intuitive and responsive. Interactions with graphical representations of the data should result in predictable visual changes, and no visual change should happen without the user’s intention. Finally, the program should be optimized well-enough so that the lag between interactions and visual changes remains close to imperceptible under moderate data volumes.

3.3 Technical Design

The source files were written in TypeScript version 4.6.3 (Bierman, Abadi, and Torgersen 2014) and bundled and transpiled into JavaScript using the `typescript-bundle` package version 1.0.18, using the VS Code IDE 1.73.1. All code written was in native Typescript/JavaScript, no external JavaScript packages were used. The R interface to the package was written in R version 4.2.2 (2022-10-31) – “Innocent and Trusting”, using the RStudio IDE. The binding between JavaScript and R was specified using the `htmlwidgets` package (Vaidyanathan et al. 2021), version 1.5.4.

3.3.1 Scene

Scene sits at the top of the `plotscape` object hierarchy. It represents the unified whole of the visual information that the user sees and interacts with. All other visual and interactive elements are defined as or on its proerties, and scene is responsible for constructing and organizing and coordinating them. The two main classes of elements defined on the scene are:

- Global Handlers
- Plots

3.3.2 Global Handlers

Global handlers take care of global responsibilities such as keyboard input, changes in state, and selection membership of the cases (rows of the data). The following global handlers are defined on the Scene:

- Marker
- State handler
- Keyboard handler

3.3.2.1 Marker

Marker is probably the most important global handler. Its primary task is to monitor changes in the membership of cases in the data and answers queries about them. Each case (row) of the data can be assigned one of several different memberships: base (no membership, group 0), persistent (group 1-3), transient (yes/no). Further, there can be a cross-product between transient and persistent membership: for example, a case can be either pure base, transiently selected base, pure group 1, transiently selected group 1, etc… The membership information is encoded on the Marker in a single unsigned 8-bit integer vector, with persistent group memberships being stored on the least significant bit (e.g. group 0: 00000000, group 1: 00000001, etc…) and transient membership being stored on the most significant (i.e. 8th) bit.

Table 3.1: Membership representation

Transiently selected	**Group**	**Bit representation**
No	Base	00000000
No	Group 1	00000001
No	Group 2	00000010
No	Group 3	00000011
No	Group 4	00000100
Yes	Base	10000000
Yes	Group 1	10000001
Yes	Group 2	10000010
Yes	Group 3	10000011
Yes	Group 4	10000100

The Marker constantly listens to selection events. When selection happens, it queries the State Handler to find out the current membership state. It then assigns that membership to the selected cases. For example, if we are currently holding down the “1” key (associated with group 1) and click on a bar in a barplot corresponding to cases 10, 11, and 12, the Marker will assigns those cases the group 1 membership. Conversely, other objects can query the marker to find the membership of the cases. This will most often happen when graphical

objects are being drawn: for example, when a barplot is drawing its bars, it draws one set of bars for each membership. At each step, it queries the marker to find which cases belong that particular membership, and it then draws bars of the relevant height and color.

3.3.2.2 State Handler

The State Handler keeps track of the global state (i.e. state of the entire scene) as well local state of the individual plots, by listening to user input captured by other handlers such as the Keyboard Handler. It also answers queries about state to other handlers and elements. For example, a Marker might enquire about the global state when a selection event is happening, to determine which membership it should assign to the selected cases. Conversely, when a selection action is happening somewhere within the scene, a single plot may enquire about its own state, specifically about whether it is active or not, and the plot will only draw its contents if the State Handler responds that it is active.

3.3.2.3 Keyboard Handler

The Keyboard Handler listens to user's input through keyboard. It records what keys are currently pressed and passes that information down to other handlers.

3.3.3 Plots

Plots are the core visible elements of a `plotscape` scene. Each plot consists of:

- Graphic Stack: draw graphical primitives in one of three Layers: Base, Highlight, and User
- Wranglers: organize, transform, and summarize variables from the data
- Scales: transform data into screen coordinates
- Representations: draw and represent data with graphical objects
- Auxiliaries: draw and represent non-data elements such as axis lines, text, etc…
- Local handlers: handle local behaviors such as clicking and dragging
- (Global handlers): passed by reference from Scene, handle global behavior such as keyboard input, sizing, etc…

The core difference between distinct types of plots such as scatterplot and barplot is a unique combination of a Wrangler and Representations. To create a new plot, we can either define a new Wrangler and assign one or more Representations to it, or we can use convenient, ready-made plot wrappers. The currently supported plot wrappers are: scatterplot, bubbplot, barplot, histogram, squareplot/square bubbleplot and a square heatmap.

3.3.3.1 Graphic Stack

Graphic stack is a container for three layers: Base, Highlight, and User. Together, these three layers provide an interface for drawing all relevant graphical primitives (such as lines, points, rectangles, etc…) on the the screen. First, the Base layer draws all parts of the plot that do not need to be re-rendered often, such as the base membership representations, axis lines, axis text, etc… The only time the Base layer gets redrawn is if either the scene gets re-scaled, or if some non-membership graphic attributes of objects such as size or opacity get changed. Next, the Highlight layer draws representations that belong to the highlight memberships (i.e. all possible combinations of transient and persistent membership). As such, the Highlight layer gets redrawn each time a selection event occurs. Finally, the User layer draws auxiliary graphical elements that correspond to user interaction. Currently, the only elements drawn in the User layer are screen dimming and a rectangular window that shows where a click-and-drag selection has occured.

3.3.3.2 Wranglers

Wranglers provide a way to organize, transform, and summarize data into statistical summaries that underpin the graphical objects presented on the screen. Each wrangler is made up of several Casts. A cast is an augmented version of a data variable that: a) knows which aesthetic it represents, b) remembers which

graphical object each of its elements belongs to via a list of indices, and c) knows how to apply a transformation to itself to return the relevant summaries of the data. This is necessary so that scene can respond to user input on the fly, while also retaining the flexibility to represent many different types of plots.

To illustrate how Wranglers and Casts work, the simplest example is of course a scatterplot. A scatterplot a one-to-one representation of the data, meaning that each point represents one row of the dataset and its coordinates are given by the two columns that map onto the `x` and `y` axes. No transformations of the data are necessary. Thus, a scatterplot will have a single wrangler, with two casts: the `x` and `y`. The two casts will be independent, and since each row of the data is one point in the scatterplot, their lists of indices will consist of all unique values (one per each row of the dataset). Finally, the summary transformation applied to the data will simply be identity, meaning that the cast will simply return the raw values of the `x` and `y` variables from the data.

More complex summaries of the data require more complex Wranglers. For example, the height of a histogram is determined by the number of cases within a specific range of the `x` variable. Thus, while histogram is also defined by a single wrangler with casts of `x` and `y` variables, these are no longer independent and we cannot get away with a simple identity transformation. First off, the `x` cast takes the original data variable and transforms it by assigning each value to one of n bins. Then it also discards all non-unique values, ending up with n values, one for each bin. In contrast, the `y` cast needs to count the number of cases within each bin. It can do this by taking a vector of 1's, splitting it according to indices indicating bin membership, (e.g. indices [1, 1, 2, 2, 1, 3] mean that the first, second, and fifth values belong to bin one, third and fourth belong to bin two, and the sixth value belong to bin 3), and summing within each split. Here's how we could implement the whole operation in R:

```
set.seed(123456)
N <- 20
bins <- seq(0, 30, 5)

(numeric_variable <- sample(1:30, N))
```

```
##  [1] 28 10 29 17  7 13 22  4  6 30  3 20 25 15 26 19 16 12 24 18
```

```
(binned_numeric_variable <- cut(numeric_variable, bins))
```

```
##  [1] (25,30] (5,10]  (25,30] (15,20] (5,10]  (10,15] (20,25] (0,5]   (5,10]
## [10] (25,30] (0,5]   (15,20] (20,25] (10,15] (25,30] (15,20] (15,20] (10,15]
## [19] (20,25] (15,20]
## Levels: (0,5] (5,10] (10,15] (15,20] (20,25] (25,30]
```

```
indices <- as.numeric(binned_numeric_variable)

(indicator <- rep(1, length(numeric_variable)))
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
(split_indicator <- split(indicator, indices))
```

```
## $`1`
## [1] 1 1
##
## $`2`
## [1] 1 1 1
##
## $`3`
## [1] 1 1 1
##
## $`4`
## [1] 1 1 1 1 1
##
## $`5`
## [1] 1 1 1
##
## $`6`
## [1] 1 1 1 1
```

```
x <- unique(binned_numeric_variable)
y <- sapply(split_indicator, sum)

knitr::kable(data.frame(x, y))
```

x	y
(25,30]	2
(5,10]	3
(15,20]	3
(10,15]	5
(20,25]	3
(0,5]	4

There are several questions one might ask about this way of doing things. First off, why go into the trouble of representing `y` as a vector of 1's and then summing it up, instead of just counting the number of times each bin occurs? The answer is that having `y` represented with a separate variable gives us greater flexibility and generality. For example, imagine that instead of just counting the number of cases within each bin, we wanted to sum the values of another variable in each bin. In our framework, we can easily do this by swapping the vector of 1's in `y` by an actual variable in the data.

3.3.3.3 Scales

Scales translate the data coordinates into screen coordinates and vice versa, and come in two flavors: discrete and continuous. The most basic types of scales are the x- and y-axis scales. As of right now, there is also the area scale implemented, and other scales such as color will also be most likely implemented in the future.

The most important property of a scale is its length. Continuous scales translate data values to length values by taking the data values as a fraction of the total data range and then multiplying it by the length. Discrete scales partition the length between a finite number of discrete values. Apart from length, other important properties of scales are direction, offset, zero, and expansion factor. Direction is fairly self-explanatory, for example the default plotting coordinate directions in HTML Canvas Element are left-to-right top-to-bottom, whereas in most typical plots, the y-axis direction is bottom-to-top, so in our y-axis scale we need to reverse

the direction. Offset specifies whether the length values should start from some number other than zero, for example, if we want our x-axis to start 50px off the left border of the plot, we need to specify offset to be 50px. Zero is just a boolean value that specifies whether 0 is a meaningful value in the data: for example, in a barplot, we might have counts of in each bar > 0, however, we want to plot the bars starting at 0, so we need to set zero to `true`. Finally, the expansion factor specifies how much beyond the range of the data should the scale expand. This is typically useful so that e.g. the smallest-valued points on a scatterplot do not get plotted over the x- or y- axis.

3.3.3.4 Representations

Representations are one or more (usually more) graphical objects that directly represent the data. For example, all of the points on a scatterplot are a representation. Each representation knows how to query the Wrangler(s) for the mappings it needs to plot, how to draw its objects under different memberships, and where the boundaries of its objects are (such that they can be selected).

3.3.3.5 Auxiliaries

Auxiliaries are graphical objects that do not directly represent the data. Typical examples are axis lines, axis text/title, etc… Because the design philosophy of `plotscape` is minimalism, only the most necessary few are currently implemented. Other auxiliaries such as axis ticks may be added later on, so that the user can display them if they want, however they will most likely not be on by default.

3.3.3.6 Local Handlers

Like Global Handlers, Local Handlers manage responsibilities that are however local to a single plot. Examples currently include clicking with Click Handler, and dragging with Drag Handler.

4 Use

4.1 A minimal example: linked scatterplot and barplot

To get started with `plotscape`, we first need to load the package:

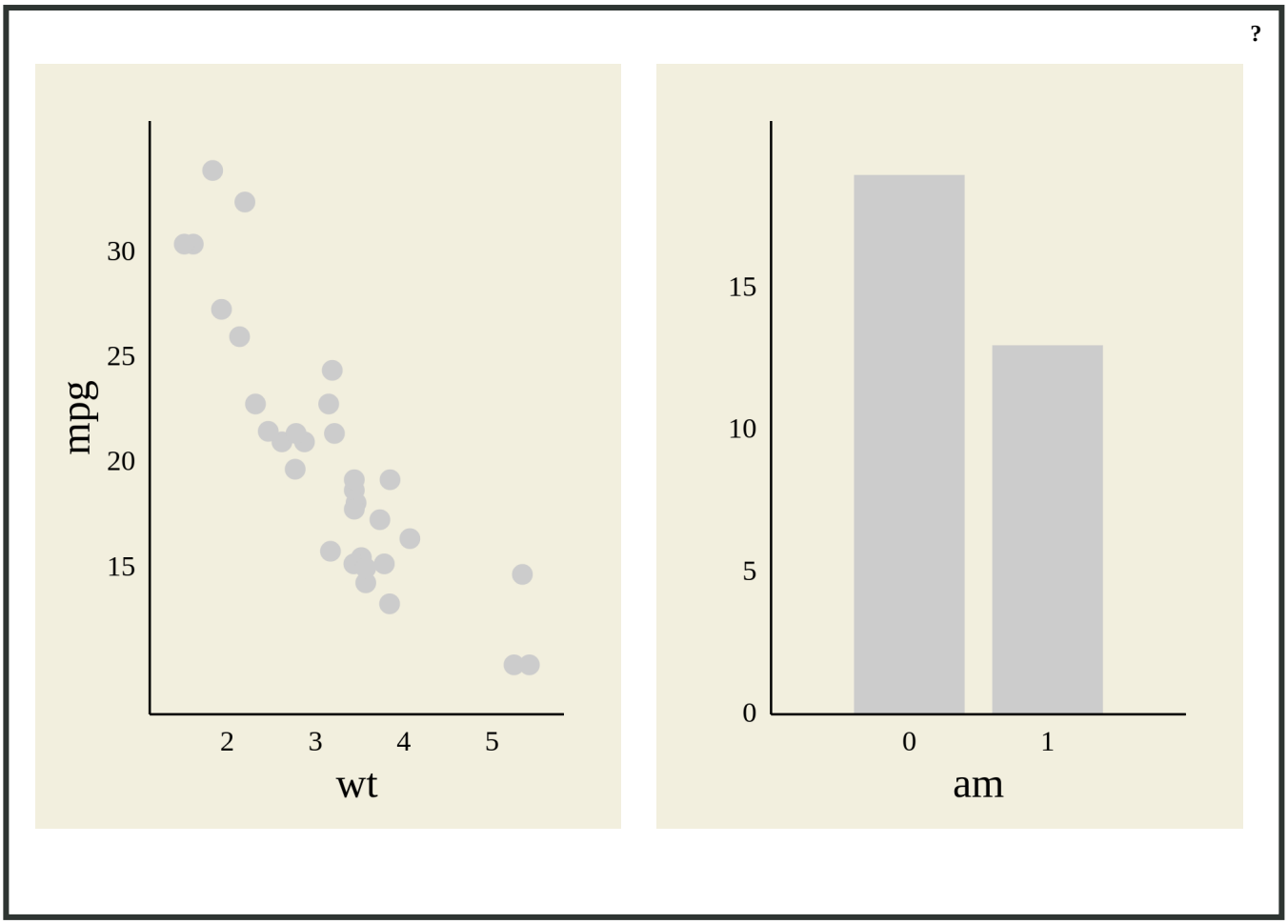
```
library(plotscapeTest)
```

Now, the first step of any `plotscape` visualization is creating a visual scene. We assign the scene to an object and then print it:

```
scene1 <- ps_scene(mtcars)
scene1
```

All we can see now is a gray rectangle. This is because we have not populated the scene with any plots yet yet. Let's add a scatterplot and a barplot:

```
scene1 |>
  ps_scatter(ps_map(x = "wt", y = "mpg")) |>
  ps_bar(ps_map(x = "am"))
```



Notice that we have used the base R pipe `|>` operator to take an existing `plotscape` scene and append plots to it using plot wrapper functions with the `ps_{plot type}()` syntax. Instead of the native pipe, we could have also used the `%>%` pipe operator from the `magrittr` / `tidyverse` packages, but that would require us loading an additional package, so let's keep using the native pipe here.

The plot wrapper functions take in several arguments, the most important of which is the `mapping` argument. This specifies which variables in the data map onto which visual aesthetic or encoding, similar to the `ggplot2` package. We specify the mapping using the `ps_map()` function (equivalent to `ggplot2`'s `aes()`) which takes in string as arguments, where the name of the argument specifies aesthetic mapping and the string specifies that name of the variable (column) in the data to map onto the aesthetic. We can either specify the aesthetics explicitly by writing e.g. `x = wt`, or more tersely by using positional argument matching, without providing explicit names for the aesthetics, such that the first argument gets automatically treated as `x` and second as `y`. We can do this like so:

```
scene1 |>
  ps_scatter(ps_map("wt", "mpg")) |>
  ps_bar(ps_map("am"))
```

Final option is to supply a named list instead of `ps_map()`:

```
scene1 |>
  ps_scatter(list(x = "wt", y = "mpg")) |>
  ps_bar(list(x = "am"))
```

This works, and in fact, all the `ps_map()` function does is take the provided arguments and convert it to a named list. However, an advantage of `ps_map()` over just `list()` is the positional matching, i.e. `ps_scatter(list("wt", "mpg"))` will not work. Therefore, it is better to always use `ps_map()` just to be on the safe side.

We will stick with the terser positional matching syntax from now on. But enough about functions and arguments, `plotscape` is about interactivity! Try interacting with the barplot by clicking on the right bar:

```
scene1 |>
  ps_scatter(ps_map("wt", "mpg")) |>
  ps_bar(ps_map("am"))
```

You should now see all of the automatic cars show up as highlighted points in the scatterplot. Now, for change, try selecting the four points in the top-left corner of the scatterplot by clicking-and-dragging. To make things more interesting, hold down the “1” key on your keyboard while dragging assign the points to Group 1 (and color them green!).

```
scene1 |>
  ps_scatter(ps_map("wt", "mpg")) |>
  ps_bar(ps_map("am"))
```

Notice that while the default (transient) selection disappears once you click the anywhere in the scene again, the grouped (persistent) selection persists. Further, we can do transiently select points that have been assigned to Group 1, and they will show up as both transiently selected *and* Group 1. This is a feature that is designed to help us ask questions that drill down to specific subgroups in the data, e.g. “what observations belong to this bar on a barplot *and* also are located within this region of a scatterplot”. Had we instead used another persistent selection to select Group 1 points, e.g. Group 2 (the “2” key), we would instead just overwrite the current selection. The key lesson here is that transient and persistent selections compose, however, persistent and persistent don’t.

If you’re struggling to see the points, try increasing their size with the “+” key (and if they get too big for your liking, you can make them smaller again with “-”). Or try increasing the transparency of the points with “[” (or decreasing with ”]”):

```
scene1 |>
  ps_scatter(ps_map("wt", "mpg")) |>
  ps_bar(ps_map("am"))
```

There are many different interactive actions you can take. Fortunately, you do not have to remember them all, they’re written in the pop-up help bar that you can access by pressing the button marked “?” in the top-right corner of the scene (the grey box around the plots).

You might be getting a bit bored of the old scatterplot and barplot at this point. How about adding a histogram and a squareplot into the mix:

```
ps_scene(mtcars) |>
  ps_scatter(ps_map(x = "wt", y = "mpg")) |>
  ps_bar(ps_map(x = "cyl")) |>
  ps_histo(ps_map(x = "disp")) |>
  ps_wrapper_plot("square", ps_map(x = "gear", y = "am"))
```

Now, in the squareplot, try selecting the square that encodes all 5-gear automatic (hint: the top-right square) and look at the scatterplot. Notice anything interesting?

All of the 5-gear automatic cars are neatly aligned below the main diagonal, meaning that they are less efficient for their weight than we would expect. To arrive at an insight like this with traditional static plots, we would either have to know where to look or try out many different plots (personally, I've used the `mtcars` dataset as test data for years never noticed this trend). With `plotscape`, we can generate insights on the fly!

4.2 Creating more complex scenes with layouts

To create more interesting scenes, we can use two tools: a) adding plots programmatically, and b) arranging them using the CSS grid layout. Adding plots programmatically means we can just write R code (e.g. a `for` loop) and let R handle adding plots to a scene for us, instead of explicitly declaring each plot one by one. The CSS grid layout should come naturally to those used to R's `layout()` function (part of the `graphics` package). In short, we can define the scene layout using a matrix that encodes the relative sizes and positions of the different plots. For example, the following matrix:

```
1 1 1 4
2 2 3 4
```

would give us a scene where most of the top half is taken up by a wide plot (plot 1), the right side is taken up by a tall narrow plot (plot 4), and the rest of the space is split up between two plots, with one (plot 2) being twice as wide as the other (plot 3).

To give an example, using CSS grid layout and adding plots programmatically, here's how we can create a visualization that combines a barplot and a scatterplot matrix, using the `iris` data:

```
# Create a matrix of all unique combinations
# of 2 variables in the iris data
vars <- names(iris)
xy <- matrix(vars[combn(1:4, 2)],
                    ncol = 2, byrow = TRUE)

layout1 <- matrix(c(
  7, 7, 7, 1,
  7, 7, 7, 2,
  7, 7, 7, 3,
  4, 5, 6, 0 # Put 0 instead of plot number to leave the space empty
), nrow = 4, byrow = TRUE)

scene2 <- ps_scene(iris, layout = layout1)
for (i in seq_len(nrow(xy))) {
  scene2 <- scene2 |> ps_scatter(ps_map(xy[i, 1], xy[i, 2]))
}
scene2 <- scene2 |> ps_bar(ps_map("Species"))
scene2
```

This is not a very efficient visualization, as the barplot eats up a lot of the available space, to the detriment of the scatterplots. However, it does serve its purpose in highlighting the power and flexibility of grid layouts. As long as we can frame it in terms of composing several rectangular plots into a grid, we can create any kind of scene we want!

5 Conclusion

5.1 Concluding remarks

Currently, most available software for interactive data visualization is geared towards three things: 1) presentation, not exploration, 2) generality rather than ease of use, and 3) shallow types of interactivity rather than more interesting ones. While it is important to have general frameworks that computer science specialists can use to build attractive data presentations, the absence of alternatives means that applied researchers and data scientist lack easy-to-use tools for understanding their data and deriving insights effectively. It is hard to to quantify just how much of an opportunity cost this presents.

The goal of the present thesis was to develop a prototype or a proof-of-concept of an interactive data visualization system that would fill the data exploration niche outlined above. Further, it was to do so in line with solid data visualization principles. I believe that I have created a prototype of such a system in `plotscape`, which, while still far from being completely general and robust, is already able to produce a range of interesting and useful visualization.

5.2 Future Directions

There is a variety of features that could still be implemented. Currently, two types of data representations are supported: bars and points. Using just these two representations, I have implemented the following types of wrapper plots: scatterplot, bubbleplot, barplot, histogram, squareplot/square bubble plot, and square heatmap. However, using the Wrangler and Cast framework, it should not be difficult to implement other types of plots. To make even more types of plots available, it would be worthwhile to implement other types of representations such as lines and polygons. Further, it could be very useful to add more sophisticated, derived data representations such as linear/smooth fits. On a more abstract level, it may also be worthwhile to develop the logic behind the Wrangler-and-Cast framework more deeply. Conversely, while there are some concrete features such as color and outline of representations that can be modified, many features of the `plotscape` plots or scenes are currently either hard-coded or automatically derived: it would be useful to add user customization for graphical features such as axis text/title/ticks, margins, etc… Finally, it is important to mention that `plotscape` is still in a very experimental state at the present date - to make it into a full R/JavaScript package, it would be necessary to round out the documentation, add proper error messages/handling, etc…

To summarize, I have created a proof-of-concept interactive data visualization system geared towards data exploration and ease of use. The package is directly callable from R and, using few core elements, it can already be used to create a fairly wide range of interesting and useful interactive visualizations. As such, I believe it has great potential to bring great benefit to applied researchers. However, to make it into a fully-fledged and roboust interactive visualization system, more work would be required.

References

Abbate, J. 1999. “Getting small: a short history of the personal computer.” *Proc. IEEE* 87 (9): 1695–98. https://doi.org/10.1109/5.784256 (https://doi.org/10.1109/5.784256).

Backus, John. 1978. “The History of Fortran i, II, and III.” *ACM Sigplan Notices* 13 (8): 165–80.

Batch, Andrea, and Niklas Elmqvist. 2017. “The Interactive Visualization Gap in Initial Exploratory Data Analysis.” *IEEE Transactions on Visualization and Computer Graphics* 24 (1): 278–87.

Becker, Richard A, and William S Cleveland. 1987. “Brushing Scatterplots.” *Technometrics* 29 (2): 127–42.

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

Betancourt, Michael. 2020. “Visualization Q and A.” *YouTube*. Youtube. https://www.youtube.com/watch?v=vPNxo9XdeLk (https://www.youtube.com/watch?v=vPNxo9XdeLk).

Bierman, Gavin, Martín Abadi, and Mads Torgersen. 2014. “Understanding Typescript.” In *European Conference on Object-Oriented Programming*, 257–81. Springer.

Bostock, Mike. 2022. “D3.js - Data-Driven Documents.” https://d3js.org (https://d3js.org).

Brodbeck, Dominique, Riccardo Mazza, and Denis Lalanne. 2009. “Interactive Visualization - A Survey.” In *Human Machine Interaction*, 27–46. Berlin, Germany: Springer. https://doi.org/10.1007/978-3-642-00437-7_2 (https://doi.org/10.1007/978-3-642-00437-7_2).

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

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 (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).

“Flare │ Data Visualization for the Web.” 2020. *Blokt - Privacy, Tech, Bitcoin, Blockchain & Cryptocurrency*. https://blokt.com/tool/prefuse-flare (https://blokt.com/tool/prefuse-flare).

Fowlkes, EB. 1969. “User’s Manual for a System for Interactive Probability Plotting on Graphic-2.” *Tech-Nical Memorandum, AT&T Bell Labs, Murray Hill, NJ*.

Friendly, Michael. 2006. “A Brief History of Data Visualization.” In *Handbook of Computational Statistics: Data Visualization*, edited by C. Chen, W. Härdle, and A Unwin, III???–. Heidelberg: Springer-Verlag.

Friendly, Michael, and Howard Wainer. 2021. *A History of Data Visualization and Graphic Communication*. Harvard University Press.

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 (https://doi.org/10.1145/1054972.1055031).

Holtz, Yan. 2022. “Interactive charts │ the R Graph Gallery.” https://r-graph-gallery.com/interactive-charts.html (https://r-graph-gallery.com/interactive-charts.html).

“Interactive javascript charts library.” 2022. *Highcharts*. https://www.highcharts.com (https://www.highcharts.com).

Jankun-Kelly, TJ, Kwan-Liu Ma, and Michael Gertz. 2007. “A Model and Framework for Visualization Exploration.” *IEEE Transactions on Visualization and Computer Graphics* 13 (2): 357–69.

Kehrer, Johannes, Roland N Boubela, Peter Filzmoser, and Harald Piringer. 2012. “A Generic Model for the Integration of Interactive Visualization and Statistical Computing Using r.” In *2012 IEEE Conference on Visual Analytics Science and Technology (VAST)*, 233–34. IEEE.

Keim, Daniel A. 2002. “Information Visualization and Visual Data Mining.” *IEEE Transactions on Visualization and Computer Graphics* 8 (1): 1–8.

Kruskal, J. B. 1965. “Multidimensional Scaling.” https://community.amstat.org/jointscsg-section/media/videos (https://community.amstat.org/jointscsg-section/media/videos).

Kunst, Joshua. 2022. *Highcharter: A Wrapper for the ’Highcharts’ Library*.

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

McCloud, Scott. 1994. *Understanding Comics: The Invisible Art*. New York, NY, USA: William Morrow Paperbacks.

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 (https://dash.plotly.com/interactive-graphing).

Qiu, Fangtu T., Tadashi Sugihara, and Rüdiger von der Heydt. 2007. “Figure-ground mechanisms provide structure for selective attention.” *Nat. Neurosci.* 10 (November): 1492–99. https://doi.org/10.1038/nn1989 (https://doi.org/10.1038/nn1989).

“Render millions of chart points with the Boost Module Highcharts.” 2022. *Highcharts*. https://www.highcharts.com/blog/tutorials/highcharts-high-performance-boost-module

(https://www.highcharts.com/blog/tutorials/highcharts-high-performance-boost-module).

Reynolds, Garr. 2011. *Presentation Zen: Simple Ideas on Presentation Design and Delivery*. New Riders. https://books.google.co.nz/books/about/Presentation_Zen.html?id=m1xt5IMJbXAC&source=kp_book_description&redir_esc=y (https://books.google.co.nz/books/about/Presentation_Zen.html?id=m1xt5IMJbXAC&source=kp_book_description&redir_esc=y).

Rheingans, Penny. 2002. “Are We There yet? Exploring with Dynamic Visualization.” *IEEE Computer Graphics and Applications* 22 (1): 6–10.

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

Tufte, Edward R. 2001. *The Visual Display of Quantitative Information*. Cheshire, Connecticut: Graphics Press LLC.

Tukey, John W. 1962. “The Future of Data Analysis.” *The Annals of Mathematical Statistics* 33 (1): 1–67.

Tukey, John W et al. 1977. *Exploratory Data Analysis*. Vol. 2. Reading, MA.

Unwin, Antony. 1999. “Requirements for interactive graphics software for exploratory data analysis.” *Comput. Statist.* 14 (1): 7–22. https://doi.org/10.1007/PL00022706 (https://doi.org/10.1007/PL00022706).

Urbanek, Simon. 2011. “iPlots eXtreme: Next-Generation Interactive Graphics Design and Implementation of Modern Interactive Graphics.” *Computational Statistics* 26 (3): 381–93.

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

Vaidyanathan, Ramnath, Yihui Xie, JJ Allaire, Joe Cheng, Carson Sievert, and Kenton Russell. 2021. *Htmlwidgets: HTML Widgets for r*. https://CRAN.R-project.org/package=htmlwidgets (https://CRAN.R-project.org/package=htmlwidgets).

Vega Project. 2022. “Example Gallery: Interactive.” https://vega.github.io/vega-lite/examples/#interactive (https://vega.github.io/vega-lite/examples/#interactive).

Wikipedia. 2022. “Duck test - Wikipedia.” https://en.wikipedia.org/w/index.php?title=Duck_test&oldid=1110781513 (https://en.wikipedia.org/w/index.php?title=Duck_test&oldid=1110781513).

Williams, Robin. 2004. *The Non-designers Design Book: Design and Typographic Principles for the Visual Novice*. Peachpit Press.

Wirfs-Brock, Allen, and Brendan Eich. 2020. “JavaScript: the first 20 years.” *Proc. ACM Program. Lang.* 4 (HOPL): 1–189. https://doi.org/10.1145/3386327 (https://doi.org/10.1145/3386327).

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.

Young, Forrest W, Pedro M Valero-Mora, and Michael Friendly. 2011. *Visual Statistics: Seeing Data with Dynamic Interactive Graphics*. John Wiley & Sons.