Interfacing R with the Web for Accessible, Portable, and Interactive Data Science

Carson Sievert

2015-12-02

Contents

1 Problem statement

> "[The web] has helped broaden the focus of statistics from the modeling stage to all stages of data science: finding relevant data, accessing data, reading and transforming data, visualizing the data in rich ways, modeling, and presenting the results and conclusions with compelling, interactive displays." - (Nolan and Lang 2014)

The web enables broad distribution and presentation of applied statistics products and research. Partaking often requires a non-trivial understanding of web technologies, unless a custom interface is designed for the particular task. The CRAN task views on open data and web services document such interfaces for the R language, the world's leading open source data science software (R Core Team 2015). This monumental community effort helps R users make their work accessible, portable, and interactive.

R has a long history of serving as an interface to computational facilities for the use of people doing data analysis and statistics research. In fact, the motivation behind the birth of R's predecessor, S, was to provide a direct, consistent, and interactive interface to the best computational facilities already available in languages such as FORTRAN and C (Becker and Chambers 1978). This empowers users to focus on the primary goal statistical modeling and exploratory data analysis, rather than the computational implementation details. By providing more and better interfaces to web services, we can continue to empower R users in a similar way, by making it easier to acquire and/or share data, create interactive web graphics and reports, distribute research products to a large audience in a portable way, and more generally, take advantage of modern web services.

Portability prevents the broad dissemination of statistical computing research, especially interactive statistical graphics. Interactive graphics software traditionally depend on toolkits like GTK+ or openGL that provide widgets for making interface elements, and also event loops for catching user input. These toolkits need to be installed locally on a user's computer, across various platforms, which adds to installation complexity, impeding portability. Modern web browsers with HTML5 support are now ubiquitous, and provide a cross-platform solution for sharing interactive statistical graphics. However, interfacing web-based visualizations with statistical analysis software remains difficult, and still requires juggling many languages and technologies. By providing better interfaces for creating web-based interactive statistical graphics, we can make them more accessible, and therefore make it easier to share statistical research to a wider audience. This research addresses this gap.

2 Overview

This section describes the background and an overview of my research on making web-based interactive graphics and data on the web more accessible. I currently maintain a number of software projects (including 7 different R packages) that address this common theme. It also points to my plans for completing my thesis research.

2.1 The importance of interface design

Unwin and Hofmann (2009) discuss the strengths, weaknesses, and differences between using graphical and command-line interfaces for data analysis. Graphical user interfaces (GUIs) can be much more intuitive to use, but at the cost of being less flexible, precise, and repeatable. Unwin and Hofmann argue statistical software should strive to achieve a synergy of two that leverages both of their strengths. That is, a command-line interface when we can precisely describe what we want and a graphical interface for "searching for information and interesting structures without fully specified questions."

Unwin and Hofmann further discuss the different audiences these interfaces attract. Command-line interfaces typically attract "power users" such as applied statisticians and statistical researchers in a university, whereas more casual users of statistical software typically prefer a GUI. In later sections, we discuss GUIs in greater detail within the context of interactive statistical graphics. For now, we briefly discuss some best practices for designing a command-line interface for statistical computing in R.

Before authoring an interface, one should establish the target audience, the class of problems it should address, and loosely define how the interface should actually work. During this process, it may also be helpful to identify your audience as being primarily composed of *software developers* or *data analysts*. Developers are typically more interested in using the interface to develop novel software or incorporating the functionality into a larger scientific computing environment (Jereon Ooms 2014). In this case, interactive exploration and troubleshooting is not always a luxury, so robust functionality is of utmost importance. On the other hand, analysts interfaces should work well in an interactive environment since this caters to rapid prototyping of ideas and troubleshooting of errors.

Good developer interfaces often make it easier to implement good analyst interfaces. A great recent example of a good developer interface is the R package **Rcpp**, which provides a seamless interface between R with C++ (Eddelbuettel 2013). To date, more than 500 R packages use **Rcpp** to make interfaces that are both expressive and efficient, including the highly influential analyst interfaces such as **tidyr** and **dplyr** (Wickham 2014); (Wickham and Francois 2015). These interfaces help analysts focus on the primary task of wrangling data into a form suitable for visualization and statistical modeling, rather than focusing on the implementation details behind how the transformations are performed. (Donoho 2015) argues that these interfaces "May have more impact on today's practice of data analysis than many highly-regarded theoretical statistics papers".

Evaluating statistical computing interfaces is certainly a subjective matter since we all have different tastes, different backgrounds, and have different needs. It seems reasonable to evaluate an interface based on its effectiveness and efficiency in aiding a user complete their task, but as (Unwin and Hofmann 2009) points out, "There is a tendency to judge software by the most powerful tools they provide (whether with a good interface or not)". As a result, all too often, analysts must spend time gaining the skills of a software developer. Good analyst interfaces often abstract functionality from developer interfaces in a way that allow analysts to focus on their primary task of acquiring/analyzing/modeling/visualizing data, rather than the implementation details. The following focuses on such work with respect to acquiring data from the web and interactive statistical web graphics.

2.2 Interfaces for working with web content

R has a rich history of interfacing with web technologies for accomplishing a variety of tasks such as requesting, manipulating, and creating web content. As an important first step, extending ideas from (Chambers 1999), Brian Ripley implemented the connections interface for file-oriented input/output in R (Ripley 2001). This interface supports a variety of common transfer protocols (HTTP, HTTPS, FTP), providing access to most files on the web that can be identified with a Uniform Resource Locator (URL). Connection objects are actually external pointers, meaning that, instead of immediately reading the file, they just point to the file, and make no assumptions about the actual contents of the file.

Many functions in the base R distribution for reading data (e.g., `scan`, `read.table`, `read.csv`, etc.) are built on top of connections, and provide additional functionality for parsing well-structured plain-text into basic R data structures (vector, list, data frame, etc.). However, the base R distribution does not provide functionality for parsing common file formats found on the web (e.g., HTML, XML, JSON). In addition, the standard R connection interface provides no support for communicating with web servers beyond a simple HTTP GET request (Lang 2006).

The **RCurl**, **XML**, and **RJSONIO** packages were major contributions that drastically improved our ability to request, manipulate, and create web content from R (Nolan and Lang 2014). The **RCurl** package provides a suite of high and low level bindings to the C library libcurl, making it possible to transfer files over more network protocols, communicate with web servers (e.g., submit forms, upload files, etc.), process their responses, and handle other details such as redirects and authentication (Lang 2014a). The **XML** package provides low-level bindings to the C library libxml2, making it possible to download, parse, manipulate, and create XML (and HTML) (Lang and CRAN Team 2015). To make this possible, **XML** also provides some data structures for representing XML in R. The **RJSONIO** package provides a mapping between R objects and JavaScript Object Notation (JSON) (Lang 2014b). These packages were heavily used for years, but several newer interfaces have made these tasks easier and more efficient.

The **curl**, **httr**, and **jsonlite** packages are more modern R interfaces for requesting content on the web and interacting with web servers. The **curl** package provides a much simpler interface to libcurl that also supports streaming data (useful for transferring large data), and generally has better performance than **RCurl** (Ooms 2015). The **httr** package builds on **curl** and organizes it's functionality around HTTP verbs (GET, POST, etc.) (Wickham 2015a). Since most web application programming interfaces (APIs) organize their functionality around these same verbs, it is often very easy to write R bindings to web services with **httr**. The **httr** package also builds on **jsonlite** since it provides consistent mappings between R/JSON and most most modern web APIs accept and send messages in JSON format (Jeroen Ooms 2014a). These packages have already had a profound impact on the investment required to interface R with web services, which are useful for many things beyond data acquisition. For example, it is now easy to install R packages hosted on the web (**devtools**), perform cloud computing (**analogsea**), and archive/share computational outputs (**dvn**, **rfigshare**, **RAmazonS3**, **googlesheets**, **rdrop2**, etc.).

The **rvest** package builds on **httr** and makes it easy to manipulate content in HTML/XML files (Wickham 2015b). Using **rvest** in combination with SelectorGadget, it is often possible to extract structured information (e.g., tables, lists, links, etc) from HTML with almost no knowledge/familiarity with web technologies. The **XML2R** package has a similar goal of providing an interface to acquire and manipulate XML content into tabular R data structures without any working knowledge of XML/XSLT/XPath (Sievert 2014a). As a result, these interfaces reduce the startup costs required for analysts to acquire data from the web.

Packages such as **XML**, **XML2R**, and **rvest** can download and parse the source of web pages, which is *static*, but extracting *dynamic* web content requires additional tools. The R package **rdom** fills this void and makes it easy to render and access the Document Object Model (DOM) using the headless browsing engine phantomjs (Sievert 2015). The R package **RSelenium** can also render dynamic web pages and simulate user actions, but its broad scope and heavy software requirements make it harder to use and less reliable compared to **rdom** (Harrison 2014). **rdom** is also designed to work seamlessly with **rvest**, so that one may use the `rdom()` function instead of `read_html()` to render, parse, and return the DOM as HTML (instead of just the HTML page source).

Any combination of these interfaces may be useful in acquiring data for personal use and/or providing a higher-level interface to specific data source(s) to increase their accessibility.

2.3 Interfaces for acquiring data on the web

The web provides access to the world's largest repository of publicly available information and data. If publishers follow best practices, a custom interface to the data source usually is not needed, but this is rarely the case. Many times structured data is embedded within larger unstructured documents, making it difficult to incorporate into a data analysis workflow. This is especially true of data used to inform downstream web applications, typically in XML and/or JSON format. There are two main ways to make such data more accessible: (1) package, document, and distribute the data itself (2) provide functionality to acquire the data.

If the data source is fairly small, somewhat static, and freely available with an open license, then we can directly provide data via R packaging mechanism. In this case, it is best practice for package authors include scripts used to acquire, transform, and clean the data. This model is especially nice for both teaching and providing examples, since users can easily access data by installing the R package.

R packages that just provide functionality to acquire data can be more desirable than repackaging the data for several reasons. In some cases, it helps avoid legal issues with rehosting copyrighted data. Furthermore, the source code of R packages can always be inspected, so users can verify the cleaning and transformations performed on the data to ensure its integrity. They are also versioned, which makes the data acquisition, and thus any downstream analysis, more reproducible and transparent. It is also possible handle dynamic data with such interfaces, meaning that new data can be acquired without any change to the underlying source code.

Perhaps the largest centralized effort in this direction is lead by rOpenSci, a community of R developers that, at the time of writing, maintains more than 50 packages providing access to scientific data ranging from bird sightings, species occurrence, and even text/metadata from academic publications. This provides a tremendous service to researchers who want to spend their time building models and deriving insights from data, rather than learning the programming skills necessary to acquire and clean it.

It's becoming increasingly clear that "meta" packages that standardize the interface to data acquisition/curation in a particular domain would be tremendously useful. However, it is not clear how such interfaces should be designed. The **etl** package (a joint work with Ben Baumer) is one step in this direction and actually aims to provide a standardized interface for *any* data access package that fits into an Extract-Transform-Load paradigm (Baumer and Sievert). The package provides generic `extract-transform-load` functions, but requires developers to write custom `extract-transform` methods for the specific data source. In theory, the default `load` method works for any application; as well as other database management operations such as `update` and `clean`.

2.4 Dynamic interactive statistical web graphics

2.4.1 Why interactive?

Unlike computer graphics which focuses on representing reality, virtually, data visualization is about garnering abstract relationships between multiple variables from visual representation. The dimensionality of data, the number of variables can be anything, usually more than 3D, which summons a need to get beyond 2D canvasses for display. Technology enables this, enabling the user to see many views, query and link components. Dynamic, interactive graphics permits a user to get beyond the constraints of low-dimensional displays to see into high-dimensional relationships in data.

Dynamic interactive statistical graphics is useful for descriptive statistics, and also to help build better inferential models. Any statistician is familiar with diagnosing a model by plotting data in the model space (e.g., residual plot, qqplot). This works well for determining if the assumptions of a model are adequate, but rarely suggests that our model neglects important features in the data. To combat this problem, (Wickham, Cook, and Hofmann 2015) suggest that we should plot the model in the data space and use dynamic interactive statistical graphics to do so. Interactive graphics have also proved to be useful for exploratory model analysis, a situation where we have many models to evaluate, compare, and critique (Unwin, Volinsky, and Winkler 2003); (Urbanek 2004); (Ripley 2004); (Unwin 2006); (Wickham 2007). With such power comes responsibility that we can verify that visual discoveries are real, and not due to random chance (Buja et al. 2009); (Majumder, Hofmann, and Cook 2013).

The ASA Section on Statistical Computing and Graphics maintains a video library which captures many useful dynamic interactive statistical graphics techniques. Several videos show how xgobi (predecessor to ggobi), a dynamic interactive statistical graphics system, can be used to reveal high-dimensional relationships and structures that cannot be easily identified using numerical methods alone (Swayne, Cook, and Buja 1998).[1] Another notable video shows how the interactive graphics system mondrian can be used to quickly find interesting patterns in high-dimensional data using exploratory data analysis (EDA) techniques (Theus and Urbanek 2008).[2] The most recent video is the first web-based visualization and shows how interactive techniques can be used to help interpret a topic model (a statistical mixture model applied to text data) using **LDAvis** (Sievert and Shirley 2014).[3]

In order to be practically useful, interactive statistical graphics must be fast, flexible, accessible, portable, and reproducible. In general, over the last 20-30 years interactive graphics systems were fast and flexible, but were also not easily accessible, portable, or reproducible. Web-based visualization provides the tools to combat these problems. For example, any visualization created with **LDAvis** can be shared through a Uniform Resource Locator (URL), meaning that anyone with a web browser and an internet connection can view and interact with a visualization. Furthermore, we can link anyone to any possible state of the visualization by encoding selections with a URL fragment identifier. This makes it possible to link readers to an interesting state of a visualization from an external document, while still allowing them to independently explore the same visualization and assess conclusions drawn from it.[4]

2.4.2 Indirect versus direct manipulation

Even within the statistical graphics community, the term *interactive* graphics can mean wildly different things to different people (Swayne and Klinke 1999). Some early statistical literature on the topic uses interactive in the sense that an interactive command-line prompt allows users to create graphics on-the-fly (R. A. Becker 1984). That is, users enter commands into the command-line prompt, the prompts evaluates the command, and prints the result (known as the read–eval–print loop (REPL)). Modifying a command to generate another variation of a particular result (e.g., to restyle a static plot) can be thought of as a type of interaction that some might call *indirect manipulation*.

Indirect manipulation can be achieved both from the command-line or from a graphical user interface (GUI). Indirect manipulation from the command-line is more flexible since we have complete control over the commands, but it is also more cumbersome since we must translate our thoughts into code. Indirect manipulation via a GUI is more restrictive, but it helps reduces the the gulf of execution for end-users (i.e., easier to generate desired output) (Hutchins, Hollan, and Norman 1985). In this sense, a GUI can be useful, even for experienced statistical programmers, when the command-line interface impedes our primary task of deriving insight from data.

In many cases, the gulf of execution can be further reduced through direct manipulation. Roughly speaking, within the context of interactive graphics, direct manipulation occurs whenever we interact with a plot and reveal new information tied to the event. (Cook and Swayne 2007) use the terms dynamic graphics and direct manipulation to characterize "plots that respond in real time to an analyst's queries and change dynamically to re-focus, link to information from other sources, and re-organize information." Perhaps the most powerful direct manipulation technique is the paradigm of linked views (Wilhelm 2005), which will be discussed in more detail in a later section.

A simple example to help demonstrate the differences between these interactive techniques would be in an analysis of variance (ANOVA) via multiple boxplots. By default, most plotting libraries sort categories alphabetically, but this is usually not optimal for visual comparison of groups. With a static plotting library

[1] For example, http://stat-graphics.org/movies/xgobi.html and http://stat-graphics.org/movies/grand-tour.html

[2] http://stat-graphics.org/movies/tour-de-france.html

[3] http://stat-graphics.org/movies/ldavis.html

[4] A good example of is http://cpsievert.github.io/LDAvis/reviews/reviews.html

such as **ggplot2**, we could indirectly manipulate the default by going back to the command-line, reordering the factor levels of the categorical variables, and regenerate the plot (Wickham 2009). This is flexible and precise since we may order the levels by any measure we wish (e.g., Median, Mean, IQR, etc.), but it would be much quicker and easier if we had a GUI with a drop-down menu for most of the reasonable sorting options. In a general purpose interactive graphics system such as mondrian, we can use direct manipulation to directly click and drag on the categories to reorder them, making it quick and easy to compare any two groups of interest (Theus and Urbanek 2008).

2.4.3 Linked views and pipelines

A general purpose interactive statistical graphics system should possess many direct manipulation techniques such as identifying (i.e., mousing over points to reveal labels), focusing (i.e., view size adjustment, pan and zoom), brushing/identifying, etc. However, it is the intricate management of information across multiple views of data in response to user events that is most valuable. Extending ideas from (Andreas Buja and McDonald 1988), (Wickham et al. 2010) point out that any visualization system with linked views must implement a data pipeline. That is, a "central commander" must be able to handle interaction(s) with a given view, translate its meaning to the data space, and update corresponding view(s) accordingly. In order to do so, the commander must know, and be able to compute, function(s) from data to visual space, as well as from visual space to the data. Implementing a pipeline that is fast, general, and able to handle statistical transformations is incredibly difficult. Unfortunately, literature on the implementation of such pipelines is virtually non-existent, but (Xie, Hofmann, and Cheng 2014) provides a nice overview of the implementation details in the R package **cranvas** (Yihui Xie 2013).

2.4.4 Web graphics

Thanks to the constant evolution and eventual adoption of `HTML5` as a web standard, the modern web browser now provides a viable platform for building an interactive statistical graphics systems. `HTML5` refers to a collection of technologies, each designed to perform a certain task, that work together in order to present content in a web browser. The Document Object Model (DOM) is a convention for managing all of these technologies to enable *dynamic* and *interactive* web pages. Among these technologies, there are several that are especially relevant for interactive web graphics:

1. HTML: A markup language for structuring and presenting web content.
2. SVG: A markup language for drawing scalable vector graphics.
3. CSS: A language for specifying styling of web content.
4. JavaScript: A language for manipulating web content.

Juggling all of these technologies to just create a simple statistical plot is a tall order. Thankfully, HTML5 technologies are publicly available, and benefit from thriving community of open source developers and volunteers. In the context of web-based visualization, the most influential contribution is Data Driven Documents (D3), a JavaScript library which provides high-level semantics for binding data to web content (e.g., SVG elements) and orchestrating scene updates/transitions (Heer 2011). D3 is wildly successful because is builds upon web standards, without abstracting them away, which fosters customization and interoperability. However, compared to a statistical graphics environments like R, creating basic charts is incredibly complicated and cumbersome. However, there are a number of ways to provide higher-level interfaces to web graphics, and we focus on R interfaces.

2.4.5 Translating R graphics to the web

There are a few ways to simply translate R graphics to a web format, such as SVG. R has built-in support for a SVG graphics device, made available through the `svg()` function, but it can be quite slow, which inspired the new **svglite** package (Wickham et al.). The **SVGAnnotation** package provides some functionality to post-process SVG files generated with `svg()` to add some basic interactivity and animation (Nolan and Lang 2012). The **gridSVG** package is specially designed to translate **grid** graphics (e.g., **ggplot2**, **lattice**, etc.) to SVG, and preserves the naming information of grid objects, making it easier to layer on interactive functionality (Potter and Murrell 2012). (Fujino 2015) uses **gridSVG** to enable linked brushing between **ggplot2** graphics, but only implements a few chart types. (Riutta et. al. and Russell 2015) uses **gridSVG** to provide pan and zoom capability to virtually any R graphic.

The **animint** and **plotly** packages take a different approach to translating **ggplot2** graphics to a web format (Hocking, VanderPlas, and Sievert 2015); (Sievert et al.). Instead of translating directly to SVG via **gridSVG**, they extract relevant information from the internal representation of a **ggplot2** graphic[5], store it in JavaScript Object Notation (JSON), and pass the JSON as input to a JavaScript function, which then produces a web based visualization. It is becoming more and more popular to see JavaScript graphing libraries use this design pattern (sometimes referred to as a JSON specification or schema), since it separates out *what* is information contained in the the graphic from *how* to actually draw it. This has a number of advantages; for example, **plotly** graphics can be rendered in SVG, or using WebGL (based on HTML5 canvas, not SVG) which allows the browser to render many more graphical marks by leveraging the GPU.

Converting static graphics to web formats such as SVG or canvas not only allows us to embed the graphics into larger HTML documents, but it also allows us to inject basic interactive features at no cost to the user. For example, in **animint** and **plotly** we provide tooltips and clickable legends that show/hide graphical marks corresponding to the legend entry. In the case of **animint**, we have also extended **ggplot2** grammar of graphics to enable animations and categorical linking between plots with relatively small amount of effort by users.

2.4.6 Creating interactive web graphics from R

Translating existing graphics to a web-based format is useful for quickly breathing new life into existing code, but it is fairly limited in how far we can take it. Assuming the goal is to have a general, yet high-level, interface for creating highly dynamic interactive web graphics from R, we're better off designing a new grammar exactly for this purpose. The first serious attempt in this direction was the R package **rCharts**, whose R interface is heavily inspired by **lattice** (Vaidyanathan 2013). The most impressive result of **rCharts**'s design is its ability to interface with many different JavaScript charting libraries. However, **rCharts** has little to no support for coordination of linked views from R.

Another popular interface for creating interactive web graphics from R is **ggvis**, a reworking of ggplot2's grammar of graphics to incorporate interactivity (Chang and Wickham 2015). Similar to **animint**, **ggvis** encodes plot specific information as JSON, but instead of writing a JavaScript renderer from the ground up, it uses Vega, a popular JSON schema for creating web-based graphics (Heer 2014). This limits the flexibility of **ggvis**, but it also drastically reduces the overhead in maintaining such a software project, allowing the focus to be on building a grammar for expressing interactions from R.

The current version of **ggvis** uses an old version of vega, before support for reactive components was added to its JSON schema. To implement its interactive features, **ggvis** currently uses a mix of custom JavaScript and **shiny** functionality. As a result, in order to view a **ggvis** visualization, one often needs access to a **shiny** server, which impedes portability.[6] It is plausible that, in the future, **ggvis** will leverage vega's new reactive components wherever possible, so that some visualizations can render entirely client-side, which enhances portability and performance.

Although it increases computational overhead, there are still scenarios where a web application (client-server approach) is more desirable than web pages that render entirely client-side. For example, an interactive visualization might need to dynamically perform statistical computations which are easy to program in R, but

[5]For a visual display of the internal representation used to render a **ggplot2** graph, see my **shiny** app here http://104.131.111.111:3838/ggtree/.

[6]The http://www.shinyapps.io/ service helps to provide easy access to a shiny server, so that shiny apps can be shared via a URL, for example: https://hadley.shinyapps.io/14-ggvis/linked-brushing.Rmd

difficult in JavaScript. (Urbanek and Horner 2015) and (Jeroen Ooms 2014b) also allow us to execute and retrieve R output from from a web browser via HTTP, but **shiny** is the most heavily used since apps can be written entirely in R using a very powerful, yet approachable, reactive programming framework for handling user events. There are also many convenient shortcuts for creating attractive HTML input forms, making it incredibly easy to go from R script to an web app powered by R that dynamically updates when users alter input values. In other words, **shiny** makes it quick and easy to write web-based GUIs with support for indirect manipulation.

Historically, it has required an advanced understanding of **shiny** and JavaScript to enable direct manipulation, but recently, support was added for triggering callbacks when users interact with R graphics (via `plotOutput()`) and/or arbitrary images (via `imageOutput()`) inside a **shiny** app. These special input bindings allow shiny app developers to build arbitrary direct manipulation functionality on top of base and **ggplot2** graphics with little to no JavaScript knowledge.[7] Although these input bindings can not be used to retrieve information about interaction with other graphical output, their selections can be sent to update other views.[8] This is useful, but it can also be useful to send selections from one arbitrary view to another arbitrary view.

Many JavaScript charting libraries have appeared since the advent of **rCharts**, making a centralized package for interfacing with *every* library infeasible. Many ideas deriving from work on **rCharts**, such as providing the glue to render plots in various contexts (e.g., the R console, shiny apps, and **rmarkdown** documents), evolved into the R package **htmlwidgets** (Vaidyanathan et al. 2015). Having built similar bridges for **animint** and **LDAvis**, I personally know and appreciate the amount of time and effort this package saves other package authors.

The **htmlwidgets** framework is not constrained to just graphics, it simply provides a set of conventions for authoring web content from R. Numerous JavaScript data visualization libraries are now made available using this framework, most designed for particular use cases, such as **leaflet** for geo-spatial mapping, **dygraphs** for time-series, and **networkD3** for networks (Cheng and Xie 2015); (Vanderkam and Allaire 2015); (Gandrud, Allaire, and Russell 2015).[9] There are also HTML widgets that provide an interface to more general purpose visualization JavaScript libraries such as **plotly**, **rbokeh**, and **rcdimple** (Sievert et al.); (Hafen and team 2015); (Kiernander et al. 2015). Most of these JavaScript libraries provide at least some native support for direct manipulation such as implement some type identifying (i.e., mousing over points to reveal labels), focusing (i.e., pan and zoom), and sometimes highlighting (i.e., brushing over points to highlight points in another view). Again, more often than not, the support for coordinating views is lacking, especially if we want to control coordination from R.

The R package **crosstalk** is a brand new framework for authoring coordinated HTML widgets (TODO: citation). It provides both an R and a JavaScript API for tracking user selections, meaning **crosstalk** powered HTML widgets can work with or without **shiny**, and if implemented correctly by HTML widget authors, provides a means for sharing user selections between HTML widgets. As of right now, **crosstalk** essentially just provides ways to get and set user selections, so the actual data pipeline for implementing linked views must be handled by the HTML widget author, which is far from trivial. (Cheng) demonstrates how **crosstalk** can be used to author an HTML widget for linked brushing[10], but it is not yet clear how to implement pipelines where the function between the data and visual marks something other than the identity function. **plotly** also has some support for sharing click events[11], with support for brush events coming soon.

[7] This example from the shiny gallery demonstrates how to access this information – http://shiny.rstudio.com/gallery/plot-interaction-basic.html

[8] An example of brushing a `plotOutput()` graphic (in this case a **ggplot2** graphic) to update the view of another graphic (in this case a **plotly** graphic) http://104.131.111.111:3838/brush2plotly/ (source code here – https://github.com/cpsievert/shiny_apps/blob/master/brush2plotly/app.R)

[9] For more examples and information, see http://www.htmlwidgets.org/ and http://hafen.github.io/htmlwidgetsgallery/

[10] See, for example, http://rpubs.com/jcheng/crosstalk-demo

[11] See, for example, http://bl.ocks.org/cpsievert/raw/560fcbbe8846d6af6413/

2.4.7 New challenges

As interactive graphics become more accessible and portable, they are being used more and more for presentation, rather than just a tool for discovery used by experts. Nowhere is this more evident than at major news outlets like the New York Times and The UpShot, where interactive graphics are constantly used in web publications to encourage readers to explore data that supplement a narrative. There are some exceptions to the rule[12], but all too often, these graphics ignore measures of uncertainty, and instead focus on conveying the most amount of information is the most effective way possible. To some degree, this highlights the difference in goals between the statistical graphics and InfoVis communities (Gelman and Unwin 2013).

- How to handle multiple, concurrent users? > opencpu and FastRWeb enjoy better overall performance compared to shiny since R sessions are stateless.
- What is missing is something akin to the mutaframe (**mutatr**?), that can work entirely client-side (inside the browser), but can also easily integrate with an R server framework (e.g. **shiny**).

3 Scope

This section describes work to be achieved before completion of the thesis. Most of the work involves writing, revising, and submitting papers. I have a very early start on two papers that will summarize modern interfaces in R for interactive web graphics as well as curating data on the web.

In February 2015, I was invited to write a chapter on MLB Pitching Expertise and Evaluation for the Handbook of Statistical Methods for Design and Analysis in Sports, a volume that is planned to be one of the Chapman & Hall/CRC Handbooks of Modern Statistical Methods. I've since brought on Brian Mills as a co-author, and we submitted a draft in early November. This chapter uses data collection and visualization functionality in the **pitchRx** package, but it more focused on modeling this data with Generalized Additive Models. The book likely won't be published until after this thesis is completed, and the chapter probably won't be included in the thesis, but I do intend on working on revisions of this chapter in the meantime.

Toby Dylan Hocking, Susan VanderPlas, and I have a paper in progress which outlines the design of **animint** and it's interesting features https://github.com/tdhock/animint-paper/. We've submitted this paper to IEEE Transactions on Visualization and Computer Graphics, and were told to revise and resubmit. We intend on revising and submitting to the Journal of Computational and Graphical Statistics by January 2016. The revision includes a restructuring of the content/ideas and new features implemented during Google Summer of Code 2015. The paper will be included as one of the chapters in my thesis.

As of writing, I'm working on numerous bug fixes in **plotly**, introduced by a massive reworking of **ggplot2** internals in version 1.1. I intended on making similar fixes for **animint** so users can rely on the CRAN version of **ggplot2**, rather than our outdated fork of ggplot2. This work simply ensures packages are *usable*, but I'd also like to work on novel features. Currently the most interesting, and most valuable addition would be support for linked views in plotly.

4 Taming PITCHf/x Data with XML2R and pitchRx

Pitch f/x refers a massive, publicly available baseball dataset hosted on the web in `XML` and `JSON` format. Since this data is large, increases on a daily basis, and only licensed for individual use, the **pitchRx** package provides a simple interface to download, parse, clean, and transform the data from its source (instead of directly distributing the data). If acquiring large amounts of data, to avoid memory limitations, users may

[12]This report does a good job of demonstrating uncertainty in the Labor Department's monthly jobs report using dynamic interactive graphics http://www.nytimes.com/2014/05/02/upshot/how-not-to-be-misled-by-the-jobs-report.html

divert incoming data in chunks to a database using any valid R database connection (Databases 2014). It also provides a convenient function to update an existing database with the most recently available data.

The **openWAR** package also provides high-level access to Pitch f/x data, but it is currently more limited in the data it can acquire (Baumer, Jensen, and Matthews 2015). It also currently depends on the difficult to install **Sxslt** package, impeding portability (Lang). **openWAR** depends on **Sxslt** to help transform XML files to R data frames via XSL Transformations (XSLT). Without advanced knowledge of XSLT, one must define transformations by hard coding assumptions about the XML format, such as the names of fields of interest. New variables have been added into Pitch f/x several times, and **pitchRx** automatically picks them up, thanks to functionality provided by **XML2R**.

XML2R makes it easy to wrangle relational data stored as a collection of XML files into a list of data frames. Its interface satisfies principles from pure functional programming: the output of each function can be completely determined from the input. The interface is also predictable: each function inputs and outputs a list of observations (an observation is a matrix with one row). It also represents XML content as a list of observations (matrices with one row), allowing each function to operate on native R data structures, making it more intuitive for R programmers to work with compared to the non-native XMLDocumentContent. This new representation is slightly less computationally efficient in some cases, but it has also made it much easier to implement and maintain higher-level interfaces to specific XML data sources, such as **pitchRx** and **bbscrapeR** (Sievert 2014b).

To see the fully published article "Taming PITCHf/x Data with XML2R and pitchRx", see http://rjournal.github.io/archive/2014-1/sievert.pdf

5 Curating open data in R

Work in progress. See https://github.com/cpsievert/thesis/blob/master/curate.Rmd

6 LDAvis: A method for visualizing and interpreting topics

The R package **LDAvis** creates an interactive web-based visualization of a topic model that has been fit to a corpus of text data using Latent Dirichlet Allocation (LDA). Given the estimated parameters of the topic model, it computes various summary statistics as input to an interactive visualization built with D3. The goal is to help users interpret the topics in their LDA topic model, and the interactive visualization is useful for quickly viewing and altering rankings of terms for a given topic.

http://nlp.stanford.edu/events/illvi2014/papers/sievert-illvi2014.pdf

7 Two new keywords for interactive, animated plot design: clickSelects and showSelected

This paper explains the clickSelects/showSelected paradigm, implemented in **animint**, which makes it easy to select/query points belonging to arbitrary group(s) and visualize those points in another data space. This differs from the classing linked brushing approach where points must belong to contiguous regions within a subset of the data space.

https://github.com/tdhock/animint-paper/blob/master/HOCKING-animint.pdf

8 Web-based interactive statistical graphics

Work in progress. See https://github.com/cpsievert/thesis/blob/master/web-graphics.Rmd

9 Testing interactive web-based graphics software from R

The current trend in web-based interactive statistical graphics is provide various language bindings to `JavaScript` charting libraries. To test whether the entire software stack is working as intended, it's common to verify properties of the data sent to the binding, but this does not guarantee that the end result is what we expect. A proper testing framework for this type of software should be able to construct and manipulate the Document Object Model (DOM) using technologies available to modern web browsers. To our knowledge, **animint** is the first R package to implement this testing approach, and some of the lessons learned could be used to construct a more reliable and easier to use testing suite.

10 Timeline

- January: Submit animint paper.
- March: Submit curating data paper.
- May: Submit Web Graphics paper.
- July: Linked views in plotly.
- August: Thesis defense.

References

Andreas Buja, Catherine Hurley, Daniel Asimov, and John A. McDonald. 1988. "Elements of a Viewing Pipeline for Data Analysis." In *Dynamic Graphics for Statistics*, edited by William S. Cleveland and Marylyn E. McGill. Belmont, California: Wadsworth, Inc.

Baumer, Ben, and Carson Sievert. *Etl: Extract-Transfer-Load Framework for Medium Data*. http://github.com/beanumber/etl.

Baumer, Benjamin S., Shane T. Jensen, and Gregory J. Matthews. 2015. "openWAR: An Open Source System for Overall Player Performance in Major League Baseball." *Journal of Quantitative Analysis in Sports* 11 (2). http://arxiv.org/abs/1312.7158.

Becker, R. A., and J. M. Chambers. 1978. "Design and Implementation of the 'S' System for Interactive Data Analysis." *Proceedings of COMPSAC*, 626–29.

Buja, Andreas, Dianne Cook, Heike Hofmann, Michael Lawrence, Eun-Kyung Lee, Deborah F Swayne, and Hadley Wickham. 2009. "Statistical inference for exploratory data analysis and model diagnostics." *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 367 (1906): 4361–83.

Chambers, John. 1999. *Programming with Data*. Springer Verlag.

Chang, Winston, and Hadley Wickham. 2015. *Ggvis: Interactive Grammar of Graphics*. http://CRAN.R-project.org/package=ggvis.

Cheng, Joe. *D3scatter: Demo of D3 Scatter Plot; Testbed for Crosstalk Library*. https://github.com/jcheng5/d3scatter.

Cheng, Joe, and Yihui Xie. 2015. *Leaflet: Create Interactive Web Maps with the JavaScript 'Leaflet' Library*. http://rstudio.github.io/leaflet/.

Cook, Dianne, and Deborah F. Swayne. 2007. *Interactive and Dynamic Graphics for Data Analysis : With R and GGobi*. Use R ! New York: Springer. http://www.ggobi.org/book/.

Databases, R Special Interest Group on. 2014. *DBI: R Database Interface*. http://CRAN.R-project.org/package=DBI.

Donoho, David. 2015. "50 years of Data Science." https://dl.dropboxusercontent.com/u/23421017/50YearsDataScience.pdf.

Eddelbuettel, Dirk. 2013. *Seamless R and C++ Integration with Rcpp*. Springer, New York.

Fujino, Tomokazu. 2015. *VdmR: Visual Data Mining Tools for R*. http://CRAN.R-project.org/package=vdmR.

Gandrud, Christopher, J.J. Allaire, and Kenton Russell. 2015. *NetworkD3: D3 JavaScript Network Graphs from R*. http://CRAN.R-project.org/package=networkD3.

Gelman, Andrew, and Antony Unwin. 2013. "Infovis and Statistical Graphics: Different Goals, Different Looks." *Journal of Computational and Graphical Statistics* 22 (1): 2–28.

Hafen, Ryan, and Bokeh team. 2015. *Rbokeh: R Interface for Bokeh*.

Harrison, John. 2014. *RSelenium: R Bindings for Selenium WebDriver*. http://CRAN.R-project.org/package=RSelenium.

Heer, Arvind Satyanarayan AND Kanit Wongsuphasawat AND Jeffrey. 2014. "Declarative Interaction Design for Data Visualization." In *ACM User Interface Software & Technology (UIST)*. http://idl.cs.washington.edu/papers/reactive-vega.

Heer, Michael Bostock AND Vadim Ogievetsky AND Jeffrey. 2011. "D3: Data-Driven Documents." *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*. http://vis.stanford.edu/papers/d3.

Hocking, Toby Dylan, Susan VanderPlas, and Carson Sievert. 2015. *Animint: Interactive Animations*.

Hutchins, Edwin L, James D Hollan, and Donald A Norman. 1985. "Direct Manipulation Interfaces." *HUMAN-COMPUTER INTERACTION* 1 (January): 311–38.

Kiernander, John, Mike Bostock, Jeremy Stucki, Kenton Russell, and Ramnath Vaidyanathan. 2015. *Rcdimple: Dimple Htmlwidget*.

Lang, Duncan Temple. 2006. "R as a Web Client the RCurl package." *Journal of Statistical Software*, July, 1–42.

———. 2014a. *RCurl: General Network (HTTP/FTP/.) Client Interface for R*. http://CRAN.R-project.org/package=RCurl.

———. 2014b. *RJSONIO: Serialize R Objects to JSON, JavaScript Object Notation*. http://CRAN.R-project.org/package=RJSONIO.

———. *Sxslt: R Interface to Libxslt*. http://www.omegahat.org/Sxslt, http://www.omegahat.org.

Lang, Duncan Temple, and the CRAN Team. 2015. *XML: Tools for Parsing and Generating XML Within R and S-Plus*. http://CRAN.R-project.org/package=XML.

Majumder, Mahbubul, Heike Hofmann, and Dianne Cook. 2013. "Validation of Visual Statistical Inference, Applied to Linear Models." *Journal of the American Statistical Association* 108 (503): 942–56.

Nolan, Deborah, and Duncan Temple Lang. 2012. "Interactive and Animated Scalable Vector Graphics and R Data Displays." *Journal of Statistical Software* 46 (1): 1–88. http://www.jstatsoft.org/v46/i01/.

———. 2014. *XML and Web Technologies for Data Sciences with R*. Edited by Robert Gentleman Kurt Hornik and Giovanni Parmigiani. Springer.

Ooms, Jereon. 2014. "Embedded Scientific Computing: A Scalable, Interoperable and Reproducible Approach to Statistical Software for Data-Driven Business and Open Science." PhD thesis, UCLA. https://escholarship.org/uc/item/4q6105rw.

Ooms, Jeroen. 2014a. "The Jsonlite Package: A Practical and Consistent Mapping Between JSON Data and R Objects." *ArXiv:1403.2805 [Stat.CO]*. http://arxiv.org/abs/1403.2805.

———. 2014b. "The OpenCPU System: Towards a Universal Interface for Scientific Computing Through Separation of Concerns." *ArXiv:1406.4806 [Stat.CO]*. http://arxiv.org/abs/1406.4806.

———. 2015. *Curl: A Modern and Flexible Web Client for R*. http://CRAN.R-project.org/package=curl.

Potter, Simon, and Paul Murrell. 2012. "A Structured Approach for Generating SVG." https://www.stat.auckland.ac.nz/~paul/Reports/gridSVGviaXML/generating-svg.html.

R Core Team. 2015. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. http://www.R-project.org/.

R. A. Becker, J. M. Chambers. 1984. *S: An Interactive Environment for Data Analysis and Graphics*. Wadsworth & Brooks/Cole.

Ripley, Brian D. 2001. "Connections." *R News* 1 (1): 1–32.

———. 2004. "Selecting Amongst Large Classes of Models." *Symposium in Honour of David Coxs 80th Birthday*.

Riutta et. al., Anders, and Kent Russell. 2015. *SvgPanZoom: R 'Htmlwidget' to Add Pan and Zoom to Almost Any R Graphic*. http://CRAN.R-project.org/package=svgPanZoom.

Sievert, Carson. 2014a. "Taming PITCHf/x Data with pitchRx and XML2R." *The R Journal* 6 (1). http://journal.r-project.org/archive/2014-1/sievert.pdf.

———. 2014b. *BbscrapeR: Tools for Collecting Basketball Data from Nba.com and Wnba.com*. https://github.com/cpsievert/bbscrapeR.

———. 2015. *Rdom: Access the DOM of a Webpage as HTML Using Phantomjs*. https://github.com/cpsievert/rdom.

Sievert, Carson, and Kenneth E Shirley. 2014. "LDAvis: A method for visualizing and interpreting topics." *Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces*, June, 1–8. http://nlp.stanford.edu/events/illvi2014/papers/sievert-illvi2014.pdf.

Sievert, Carson, Chris Parmer, Toby Hocking, Scott Chamberlain, Karthik Ram, Marianne Corvellec, and Pedro Despouy. *Plotly: Create Interactive Web-Based Graphs via Plotly's API*. https://github.com/ropensci/plotly.

Swayne, Deborah F, Dianne Cook, and Andreas Buja. 1998. "XGobi: Interactive Dynamic Data Visualization in the X Window System." *Journal of Computational and Graphical Statistics* 7 (1): 113–30.

Swayne, Deborah F., and Sigbert Klinke. 1999. "Introduction to the Special Issue on Interactive Graphical Data Analysis: What Is Interaction?" *Computational Statistics* 14 (1).

Theus, Martin, and Simon Urbanek. 2008. *Interactive Graphics for Data Analysis: Principles and Examples*. Chapman & Hall / CRC.

Unwin, Antony. 2006. "Exploratory Modelling Analysis: Visualizing the Value of Variables." In *Compstat 2006 - Proceedings in Computational Statistics*, edited by Alfredo Rizzi and Maurizio Vichi, 221–30. Physica-Verlag HD. http://dx.doi.org/10.1007/978-3-7908-1709-6_17.

Unwin, Antony, and Heike Hofmann. 2009. "GUI and Command-line - Conflict or Synergy?" *Proceedings of the St Symposium on the Interface*, September, 1–11.

Unwin, Antony, Chris Volinsky, and Sylvia Winkler. 2003. "Parallel Coordinates for Exploratory Modelling Analysis." *Computational Statistics & Data Analysis* 43 (4): 553–64.

Urbanek, Simon. 2004. "Model Selection and Comparison Using Interactive Graphics." PhD thesis.

Urbanek, Simon, and Jeffrey Horner. 2015. *FastRWeb: Fast Interactive Framework for Web Scripting Using R*. http://CRAN.R-project.org/package=FastRWeb.

Vaidyanathan, Ramnath. 2013. *RCharts: Interactive Charts Using Javascript Visualization Libraries*. https://github.com/ramnathv/rCharts/.

Vaidyanathan, Ramnath, Yihui Xie, JJ Allaire, Joe Cheng, and Kenton Russell. 2015. *Htmlwidgets: HTML Widgets for R*. http://CRAN.R-project.org/package=htmlwidgets.

Vanderkam, Dan, and JJ Allaire. 2015. *Dygraphs: Interface to Dygraphs Interactive Time Series Charting Library*. http://CRAN.R-project.org/package=dygraphs.

Wickham, Hadley. 2007. "Meifly: Models explored interactively." *Website ASA Sections on Statistical Computing and Graphics (Student Paper Award Winner)*.

———. 2009. *Ggplot2: Elegant Graphics for Data Analysis*. Springer New York. http://had.co.nz/ggplot2/book.

———. 2014. "Tidy Data." *The Journal of Statistical Software* 59 (10). http://www.jstatsoft.org/v59/i10/.

———. 2015a. *Httr: Tools for Working with URLs and HTTP*. http://CRAN.R-project.org/package=httr.

———. 2015b. *Rvest: Easily Harvest (Scrape) Web Pages*. http://CRAN.R-project.org/package=rvest.

Wickham, Hadley, and Romain Francois. 2015. *Dplyr: A Grammar of Data Manipulation*. http://CRAN.R-project.org/package=dplyr.

Wickham, Hadley, Dianne Cook, and Heike Hofmann. 2015. "VISUALIZING STATISTICAL MODELS: REMOVING THE BLINDFOLD." *Statistical Analysis and Data Mining The ASA Data Science Journal* 8 (4): 203–25.

Wickham, Hadley, Michael Lawrence, Dianne Cook, Andreas Buja, Heike Hofmann, and Deborah F Swayne. 2010. "The Plumbing of Interactive Graphics." *Computational Statistics*, April, 1–7.

Wickham, Hadley, T Jake Luciani, Matthieu Decorde, and Vaudor Lise. *Svglite: A SVG Graphics Device*. https://github.com/hadley/svglite.

Wilhelm, Adalbert. 2005. "Interactive Statistical Graphics: The Paradigm of Linked Views." In *Data Mining and Data Visualization*, edited by C.R. Rao, E.J. Wegman, and J.L. Solka. Elseviewer.

Xie, Yihui, Heike Hofmann, and Xiaoyue Cheng. 2014. "Reactive Programming for Interactive Graphics." *Statistical Science* 29 (2): 201–13.

Yihui Xie, Di Cook, Heike Hofmann. 2013. *Interactive Statistical Graphics Based on Qt*.