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

Carson Sievert

2015-11-18

Contents

1

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, as a platform, provides solutions to problems that have vexed statistical computing for decades, such as portability, reproducibility, and remote collaboration. However, in many cases, we lack the proper interfaces to leverage web services in a way that is practically useful for people doing data analysis and statistics research. Part of the problem is that web technologies evolve at a rapid pace, and new web services appear nearly everyday, so writing custom interfaces for every useful tool on the web takes a continuous and serious community effort. As evident by the CRAN task view on web technologies and services (Scott Chamberlain), there are many such interfaces for the R project, the world's leading open source statistical programming software (R Core Team 2015).

R has a long history of serving as an interface to computational facilities for the use by people doing data analysis and statistics research. In fact, the motivation behind the birth of R's predecesor, 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 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/share data, create interactive web graphics, perform cloud computing, practice reproducible research, collaborate remotely, and share their work with a larger audience.

Reproducibility and portability have been a particularly big problem interactive statistical graphics. Historically, open source options are often hard to install since they typically assume non-standard software is available on users' systems. Modern web browsers with `HTML5` support are now ubiquitous, a provide an adequate platform for hosting interactive statistical graphics. However, interfacing web-based visualizations with statistical software remains difficult and often requires juggling a handful of languages and technologies. Again, by providing better interfaces for creating web-based interactive statistical graphics, we can make them more accessible, and therefore make it easier to communicate statistical concepts with a wider audience.

2 Overview

This section is a concise 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 provides some background on the area, describes similar work being done, points out remaining problems, and a bit on my vision for future work.

2.1 User interfaces vs software interfaces

(Unwin and Hofmann 2009) addresses this question and makes the important distinction of interfaces between software systems and interfaces between users and software. To emphasize the difference, the latter is called a user-interface. It is reasonable to judge a user-interface based on its effectiveness in aiding the 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)". Good user-interfaces for statistical computing often abstract functionality from more powerful (lower-level) interfaces in a way that allow users to focus on their primary task, rather than the implementation details, without limiting the possible use case. Using (Wickham 2014) as an example, (Donoho 2015) argues that these abstractions "May have more impact on today's practice of data analysis than many highly-regarded theoretical statistics papers".

Well designed user-interfaces make data analysts more efficient, and also make software developers more efficient. To date, more than 500 R packages depend on the **Rcpp** package which provides a seamless interface between R with C++ (Eddelbuettel 2013). R itself does provide some utilities to run Fortran/C/C++ code from R, but the implementation details are quite cumbersome and time-consuming in comparison. As far as interfacing R with other languages goes, some other notable efforts are the **rJava** and **rPython** packages (Urbanek 2015); (Bellosta 2015). These types of interfaces provide "low-level" bindings that make it *possible* to interface R with other languages, which is a major contribution in its own right, but do not necessarily make certain tasks more *efficient*.

2.1.1 Interfaces for Working with Data on the Web

R has a rich history of interfaces for acquiring data off the web. 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). However, there are a few important limitations to this interface. Most importantly, connection objects are external pointers, meaning that they point to where the data is located, not the data itself. Thankfully there is a host of base R functions for reading from connections, such as `scan`, `readLines`, `read.table`, `read.csv`, etc. These functions work well for well-structured plain-text files, but do not work well for common file formats found on the web such as HTML, XML, and JSON. Another important limitation of the standard R connection interface is its support for interacting with web servers.

The **RCurl**, **XML**, and **RJSONIO** packages were major contributions that addressed the interface limitations between R and popular web technologies (Nolan and Lang 2014). The **RCurl** package provides low-level bindings to the C library libcurl, making it possible to transfer data over more network protocols, interact 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, and create XML (and HTML) (Lang and CRAN Team 2015). The **RJSONIO** package provides a mapping between R objects and JavaScript Object Notation (JSON) (Lang 2014b). These packages were ahead of their time, and heavily used for years, but for many reasons, several more modern interfaces have gained traction in recent years.

The **curl**, **httr**, **xml2**, and **jsonlite** packages are more modern R interfaces for working with web technologies (Wickham 2015a); (Ooms 2014). The **curl** package provides a much simpler interface to libcurl that also supports streaming data (useful for transfering large data), and also has better performance than **RCurl**

(Ooms 2015). The **httr** package builds on **curl** and organizes it's functionality around HTTP verbs (GET, POST, etc.) since most web application programming interfaces (APIs) do the same (Wickham 2015b). It also provides some useful facilities for handling HTTP errors. The **rvest** package builds on **httr** to make it easy to extract components of HTML/XML files which is quite useful since many web pages embed structured data within larger less-structured documents (Wickham 2015c).

However, the connections interface essentially treats everything as a character string, so special parsers

The **htmlwidgets** package provides a framework for creating HTML widgets that render in various contexts including the R console, 'R Markdown' documents, and 'Shiny' web applications (Vaidyanathan et al. 2015). (TODO: use this a transition point for moving to GUI/shiny applications?)

Functional programming paradigm works well for computational problems with well defined input/output. With interactive web graphics you want to the output to be dynamic, meaning that users can modify the "inputs" even after the output has been determined.

2.2   Acquiring Data on the Web

The R packages **pitchRx**, **bbscrapeR**, **XML2R**, and **rdom** all provide utilities for acquiring data hosted on the web. Amongst them, **pitchRx** and **bbscrapeR** have the highest level interface and are aimed at a known set of Uniform Resource Locators (URLs). Their interface is designed so that users do not need any understanding of underlying file formats that contain the data (e.g., `XML`, `JSON`). In order to acquire some data, users just need the package installed, so a very minimal amount of computational setup is required. If acquiring large amounts of data, to avoid memory limitations, users may divert incoming data to a database using any valid R database connection (Databases 2014).

For these reasons, **pitchRx** and **bbscrapeR** provide a nice resource for teaching and practicing applied statistics, and serve as a model for providing access to clean versions of messy datasets on the web (Unwin 2010). Providing *access* to data in this way is more desirable than rehosting data for several reasons. In some cases, it helps avoid legal issues with rehosting copyrighted data. Furthermore, the packages are self-documenting, so users can inspect 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. Perhaps most importantly, the data that these packages provide access to are updated at least several times a day, so users can keep their local copies up to date without any assistance from the maintainer.

pitchRx and **bbscrapeR** are specific to sports data, but the same idea is used in many different domains. 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`.

The Web Technologies and Service CRAN Task View does a great job of summarizing data access packages in R, and even breaks them down by their domain application (e.g., Government, Finance, Earth Science, etc) (Scott Chamberlain). It also details more general tools for requesting, parsing, and working with popular file formats on the web from R that many of the data access packages use to implement their functionality. Two other packages I maintain: **XML2R** and **rdom** fit into this broader category.

pitchRx and **bbscrapeR** leverage **XML2R**: a wrapper around the **XML** package for transforming XML content into tables (Lang and CRAN Team 2015). **XML** provides low-level R bindings to the libxml2 `C` library for parsing XML content (Veillard 2006). **XML2R** builds on this functionality and makes it possible to acquire, filter, and transform XML content into table(s) without any knowledge of the verbose `XPATH` and `XSLT` languages. These high-level semantics make it easier to maintain projects such as **pitchRx** and **bbscrapeR** since it drastically reduces the amount of code. Chapter Taming Pitch f/x data with XML2R and pitchRx details these abstractions, and how they are used in **pitchRx**, in more detail.

The **openWAR** package provides high-level access to Pitch f/x data like **pitchRx**, but it is currently more limited in the range of data types it provides (Baumer, Jensen, and Matthews 2015). It also currently depends on the difficult to install **Sxslt** package, which provides an R interface to libxslt (Lang). **openWAR** depends on **Sxslt** to help transform XML files to R data frames via XSL Transformations (XSLT). Without advanced knowledge of the very verbose XSLT specification, packages like **openWAR** are forced into hard coding many assumptions about the XML format, such as the names of fields of interest. New fields have been added to the Pitch f/x XML source several times, and **pitchRx** automatically picks them up, since its **XML2R** transformations can accommodate new field names.

rdom makes it easy to render dynamic web pages and access the Document Object Model (DOM) from R via the headless browsing engine phantomjs (Sievert 2015). This fills a void where other web scraping packages in R (e.g., **XML**, **xml2**, **rvest**) currently fall short. These packages can download, parse, and extract bits of the HTML page source, which is static, but they lack a browsing engine to fully render the DOM. If the DOM cannot be rendered, content that is dynamically generated (e.g., with `JavaScript`) cannot be acquired. 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**.

2.3 Interactive Statistical Graphics

A fundamental limitation in data visualization is the 2D display of a computer screen. There are ways to perceive multi-variate relationships

As (Wilhelm 2008) points out, there are essentially four basic approaches to visualizing high-dimensional data.

- Examples that illustrate why interactive is essential

- History

2.4 Web-based Technology

What does it mean for a graphic to be *interactive*? The answer depends heavily on who is using the term, and the context in which it is used. Even within the statistical graphics community, the definition is not uniformly agreed upon, and one's requirement(s) to label a graphics system as interactive is quite variable (Swayne and Klinke 1999). This section lays out some more precise language for discussing interactive graphics, motivates their existence, and explains where my work fits into this landscape.

Before investigating *what* interactivity means, perhaps its better to ask why is it useful? Graphics are traditionally used to present information to a larger audience. Good statistical graphics ensure that information is portrayed accurately, and focuses particularly on conveying uncertainty. Historically, interactive statistical graphics are not used for present results of an analysis, but rather as a discovery tool, prior to, during, or even after the modeling stage. More specifically, interactive graphics are useful for identifying problems or refining preconceptions about a given dataset, gaining a deeper understanding of model fitting algorithms, and even as a model selection tool (Wickham, Cook, and Hofmann 2015); (Unwin, Volinsky, and Winkler 2003); (Gelman 2015).

With the rise of the web browser (and in particular `HTML5` technologies), like it or not, the role of interactive graphics is generally shifting from discovery to presentation. 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 allow readers to explore data that supplement a narrative. There are some exceptions to the rule, 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).

Historically, open source interactive graphics software is often hard to install and practically impossible to distribute to a wider audience. The web browser provides a viable solution to this problem, as sharing an interactive graphics (and even a specific *state* of the visualization) can be as easy as sharing a Uniform Resource Locator (URL). The web browser doesn't come without some restrictions; however, since it is impossible to maintain the state of multiple windows, a fundamental characteristic of most interactive graphics software. Fortunately, we can still produce linked views by putting multiple plots in a single window.

Some early statistical literature on the topic uses interactive in the sense that interactive programming environments allow users to create graphics on-the-fly (R. A. Becker 1984). That is, the programming environment has a prompt, which can read a command to generate a plot, evaluates that command, and prints the result.

The read–eval–print loop (REPL) is a generally useful quality for a statistical programming environment to possess, since in constrast to other programming paradigms, the emphasis is on exploring the content of the data, which is often riddled with imperfections that must be addressed before any statistical modeling takes place. Assuming that the print stage outputs a static plot, this "interactivity" is limited and can be time-consuming since commands must be modified in order to obtain new views or details.

Another common interpretation of interactivity involves a Graphical User Interface (GUI) which abstracts away the REPL from end users by providing widgets or controls to alter commands. In this sense, even for experienced statistical programmers, a GUI can still be useful when the REPL impedes our ability to perform graphical data analysis (Unwin and Hofmann 2009).

A wide array of GUI toolkits have been available in R for years, and many of them interface to GUI construction libraries written in lower-level languages. A couple fairly recent and popular examples include the **RGtk2** package which provides R bindings to the GTK+ 2.0 library written C and the **rJava** package which provides R bindings to Java. A more modern approach to GUI development is via the Web browser, and has been made quite easy thanks to the R package **shiny** (Chang et al. 2015).

(Wilhelm 2003) "Information overload that would prevent perception can be hidden at the 1st stage and detailed information can be made available on demand by responding to interactive user queries"

(Cook and Swayne 2007) "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."

- Talk about rggobi and controling a standalone application from the command-line?
- R bindings that talk to JSON specifications are most similar to this approach

The **animint** package uses **RSelenium** in order to test whether visualizations render correctly in a web browser. When I started on **animint**, we were only testing the R code that compiles **ggplot2** objects as `JSON` objects, but had no way to programmatically test the `JavaScript` code that takes the `JSON` as input. If **animint** were limited to static web-based graphics, we could use the lighter-weight **rdom** for testing, but we need to simulate user actions to verify animations and interactive features work as expected.

In addition to adding infrastructure for testing **animint**'s renderer, I've made a number of other contributions:

1. Wrote bindings for embedding **animint** plots inside of knitr/rmarkdown/shiny documents, before the advent of **htmlwidgets**, which provides standard conventions for writing such bindings (Vaidyanathan et al. 2015). At the time of writing, **htmlwidgets** can only be rendered from the R console, the R Studio viewer, and using R Markdown (v2). For this reason, we decide to not use **htmlwidgets** since users may want to incorporate this work into a different workflow.

2. Wrote `animint2gist`, which uploads an **animint** visualization as a GitHub gist, which allows users to easily share the visualizations with others via a URL link.

3. Implemented **ggplot2** facets (i.e., `facet_wrap` and `facet_grid`) as well as the fixed coordinate system (i.e., `coord_fixed`).

4. Mentored and assisted Kevin Ferris during his 2015 Google Summer of Code project where he implemented theming options (i.e., `theme`), legend interactivity, and selectize widgets for selecting values via a drop-down menu.

When I started on **plotly**, it's core functionality and philosophy was very similar to **animint**: create interactive web-based visualizations using **ggplot2** syntax (Sievert et al.). However, plotly's `JavaScript` graphing library supports chart types and certain customization that **ggplot2**'s syntax doesn't support. Realizing this, I initiated and designed a new domain-specific language (DSL) for using plotly's `JavaScript` graphing library from R. Although it's design is inspired by **ggplot2**'s `qplot` syntax, the DSL does not rely on **ggplot2**, which is desirable since its functionality won't break when **ggplot2** internals change.

plotly's 'native' R DSL is heavily influenced by concepts deriving from pure functional programming. The output of a pure function is completely determined by its input(s), and because we don't need any other context about the state of the program, it easy to read and understand the intention of any pure function. When a suite of pure functions are designed around a central object type, we can combine these simple pieces into a pipeline to solve complicated tasks, as is done in many popular R packages such as **dplyr**, **tidyr**, **rvest**, etc (Wickham 2015d).

plotly's pure functions are deliberately designed around data frames so we can conceptualize a visualization as a pipe-able sequence of data transformations, model specifications, and mappings from the data/model space to visual space. With the R package **ggvis** (Chang and Wickham 2015), one can also mix data transformation and visual specifications in a single pipeline, but it does so by providing S3 methods for **dplyr**'s generic functions, so all data transformations in a **ggvis** pipeline have to use these generics. By directly modeling visualizations as data frames, **plotly** removes this restriction that transformation must derive from a generic function, and removes the burden of exporting transformation methods on its developers.

plotly even respects transformations that remove attributes used to track visual properties and data mappings. To demonstrate, in the example below, we plot the raw time series with `plot_ly()`, fit a local polynomial regression with `loess()`, obtain the observation-level characteristics of the model with `augment()` from the **broom** package, layer on the fitted values to the original plot with `add_trace()`, and add a title with `layout()`.

```
library(plotly)
library(broom)
economics %>%
  transform(rate = unemploy / pop) %>%
  plot_ly(x = date, y = rate, name = "raw") %>%
  loess(rate ~ as.numeric(date), data = .) %>%
  augment() %>%
  add_trace(y = .fitted, name = "smooth") %>%
  layout(title = "Proportion of U.S. population that is unemployed")
```

Proportion of U.S. population that is unemployed
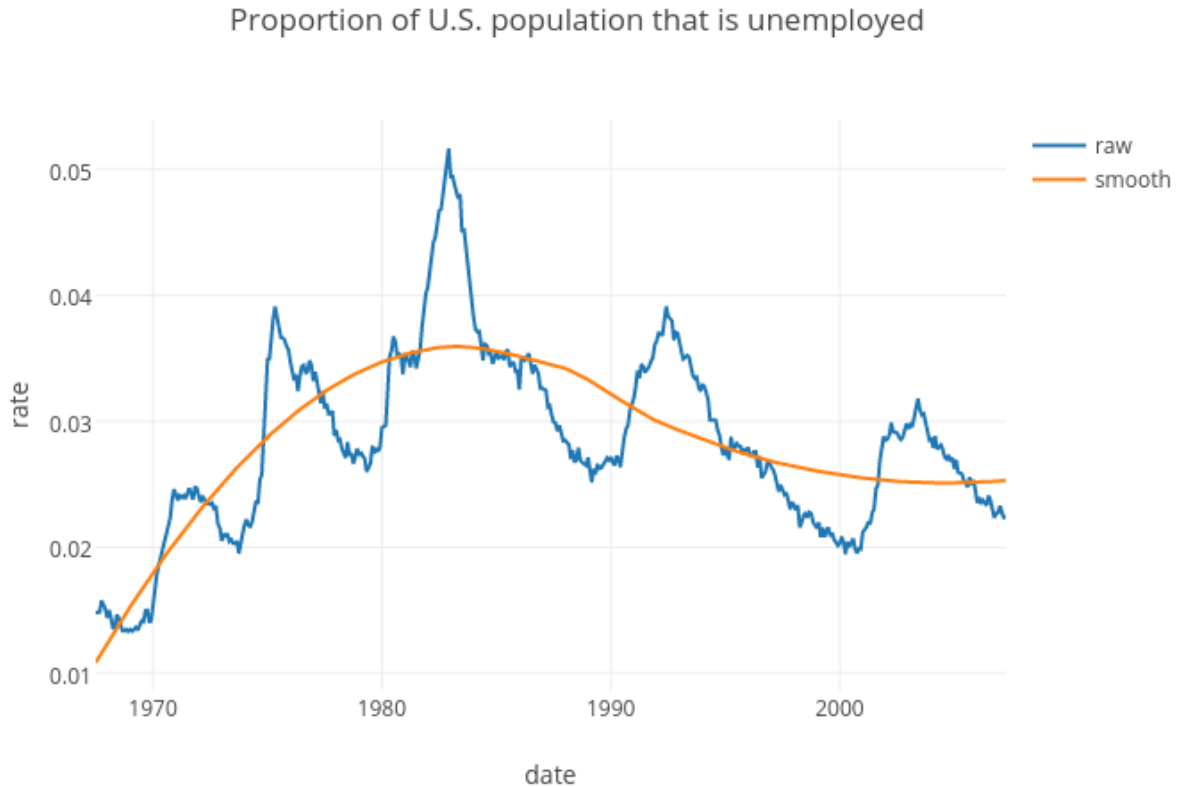
To make this possible, a special environment within **plotly**'s namespace tracks not only visual mappings/properties, but also the order in which they are specified. So, if a **plotly** function used to modify a visualization (e.g., `add_trace()` or `layout()`) receives a data frame without any special attributes, it retrieves the last plot created, and modifies that plot.

animint and **plotly** could be classified as general purpose software for web-based interactive and dynamic statistical graphics; whereas **LDAvis**, could be classified as software for solving a domain specific problem. The **LDAvis** package creates an interactive web-based visualization of a topic model fit to a corpus of text data using Latent Dirichlet Allocation (LDA) to assist in interpretation of topics. The visualization itself is written entirely with `HTML5` technologies and makes use of the `JavaScript` library d3js (Heer 2011) to implement advanced interaction techniques that higher-level tools such as **plotly**, **animint**, and/or **shiny** do not currently support.

3   Scope

This section describes work to be achieved before completion of the thesis. Most of the work involves writing and revising 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. I also intend on adding a new case study on using ideas from cognostics in **animint** for guiding visualizations that contain many states/views. The paper will be included as one of the chapters in my thesis.

We have a long TODO list with known bugs and features we'd like to implement in **animint** after we submit to JCGS. 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 fork on GitHub. This work simply ensures packages are *usable*, but I'm also interested in expanding the scope of **animint**, which may lead to paper(s) after the thesis is submitted:

1. The current design of **animint** requires pre-computation of every state the visualization can possibly take on. One benefit of this approach is that we don't need any special software besides a web browser for viewing, and bodes well for cognostic-like applications, but when the number of states is very large, pre-computation can take a long time, and the amount of data that the browser tries to upload can be very large. Instead of pre-computing these states, we could dynamically compute states, only when user requests them, using a HTTP requests. The **plumbr** and **opencpu** packages assist in creating a REST API providing the ability to execute arbitrary R functions over HTTP, allowing us to define endpoints at compile time, create/destroy them during the rendering/viewing stage, and all of this could be done on a viewer's machine if R is installed. This addition to **animint** would be helpful for visualizations with many states, but in order to retain responsiveness, each state would need to be relatively cheap to compute.

2. Integrate crossfilter.js into **animint**. This should help relax current restrictions that summary statistics impose on showing/selecting values.

I also intend being the sole author on two independent papers: one on strategies for testing interactive web-based graphics software from R, and one on curating data with R.

4 Taming PITCHf/x Data with XML2R and pitchRx

Completed, and published in the RJournal. 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

Completed, and published in the Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces. See http://nlp.stanford.edu/events/illvi2014/papers/sievert-illvi2014.pdf

7 Two new keywords for interactive, animated plot design: click-Selects and showSelected

Currently in revision. See 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.
- Feburary: Submit curating data paper.
- March: Submit Web Graphics paper
- April:
- May:

References

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.

Bellosta, Carlos J. Gil. 2015. *RPython: Package Allowing R to Call Python.* http://CRAN.R-project.org/package=rPython.

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.

Chang, Winston, Joe Cheng, JJ Allaire, Yihui Xie, and Jonathan McPherson. 2015. *Shiny: Web Application Framework for R.* http://CRAN.R-project.org/package=shiny.

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."

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

Gelman, Andrew. 2015. "Exploratory data analysis for complex models." *Journal of Computational and Graphical Statistics*, February, 1–29.

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

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.

Lang, Duncan Temple. 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.

Nolan, Deborah, and Duncan Temple Lang. 2014. *XML and Web Technologies for Data Sciences with R.* Edited by Robert Gentleman Kurt Hornik and Giovanni Parmigiani. Springer.

Ooms, Jeroen. 2014. "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.

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

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.

Scott Chamberlain, Patrick Mair, Thomas Leeper. "CRAN Task View: Web Technologies and Services." http://cran.r-project.org/web/views/WebTechnologies.html.

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

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

Unwin, Antony. 2010. "Datasets on the web: a resource for teaching Statistics?" *MSOR Connections*, November, 1–4.

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. 2015. *RJava: Low-Level R to Java Interface.* http://CRAN.R-project.org/package=rJava.

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

Veillard, Daniel. 2006. "Libxml: The XML c Parser and Toolkit of Gnome Parsing." http://www.xmlsoft.org.

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

———. 2015a. *Xml2: Parse XML.* http://CRAN.R-project.org/package=xml2.

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

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

———. 2015d. "Pipelines for Data Analysis." http://bids.berkeley.edu/resources/videos/pipelines-data-analysis.

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.

Wilhelm, Adalbert. 2003. "User interaction at various levels of data displays." *Computational Statistics & Data Analysis* 43 (4): 471–94.

———. 2008. "Linked Views for Visual Exploration." In *Handbook of Data Visualization*, 199–215. Berlin, Heidelberg: Springer Berlin Heidelberg.