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

Carson Sievert

2015-11-22

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, as a platform, has the potential to provide solutions to problems that have impeded broad distribution of statistical computing research products for decades. It requires interfaces to leverage web services in a way that is practically useful for people doing data analysis and statistics, and also develop new methodology in the process. Web technologies have evolved at a rapid pace, and as new web services have appeared new custom interfaces have been developed, which has taken substantial community effort. The CRAN task view on web technologies and services (Chamberlain et al. 2015), documents many customized interfaces for the R project, currently the world's leading open source data science 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 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 or share data, create interactive web graphics, perform cloud computing, practice reproducible research, collaborate remotely, and share their work with a larger audience.

Portability has prevented the broad dissemination of interactive statistical graphics. Interactive graphics software traditionally builds 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 acheive 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 upmost 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/analysing/modeling/visualizing data, rather than the implementation details. The next section focuses such work in the realm of 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 transfering 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 2014). These packages have already had a profound impact on the investment required to interface R with web services.

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 2014). 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 if we wish to extract *dynamic* web content from R, we need 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).

Any combination of these interfaces may be useful in developing high-level interfaces to specific web services or acquiring content for analysis.

2.3 Interfaces to data on the web

These high-level interfaces provide a nice resource for both teaching and practicing applied statistics, and serve as a model for providing access to clean versions of messy datasets on the web (Unwin 2010).

2.3.1 Data packages

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.

It is recommended that the package author include scripts used to acquire and transform

2.3.2 Packaging access to data

R packages that provide an interface to 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 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`.

The Open Data 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). 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.

2.4 Interfaces to web services

The scope of functionality readily available through R packages is constantly expanding thanks to these general purpose interfaces. For example, it is now easy to install R packages hosted on the web (**devtools**), perform cloud computing (**analogsea**), archive/share computational outputs (**dvn**, **rfigshare**, **RAmazonS3**, **googlesheets**, **rdrop2**, etc.), create/share web graphics (**plotly**), and acquire all sorts of data (too many to list).

2.5 Interfaces for interactive statistical web graphics

2.5.1 Interactive statistical graphics

Our ability to find interesting structures in high-dimensional data is fundamentally limited by the 2D display of a computer screen (or paper). As (Wilhelm 2008) points out, there are four main workarounds to this limitation:

1. virtual reality

2. dimension reduction (principal component analysis, projection pursuit, multidimensional scaling, correspondence analysis)
3. graphics with a non-orthogonal coordinate system (e.g., parallel coordinate plots)
4. linking low-dimensional displays (two or more plots share information with each other)

The statistical graphics community has done a lot of work in (2)-(4).

- Examples that illustrate why interactive is essential
- History

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.

2.5.2 Web-based technology

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.

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.

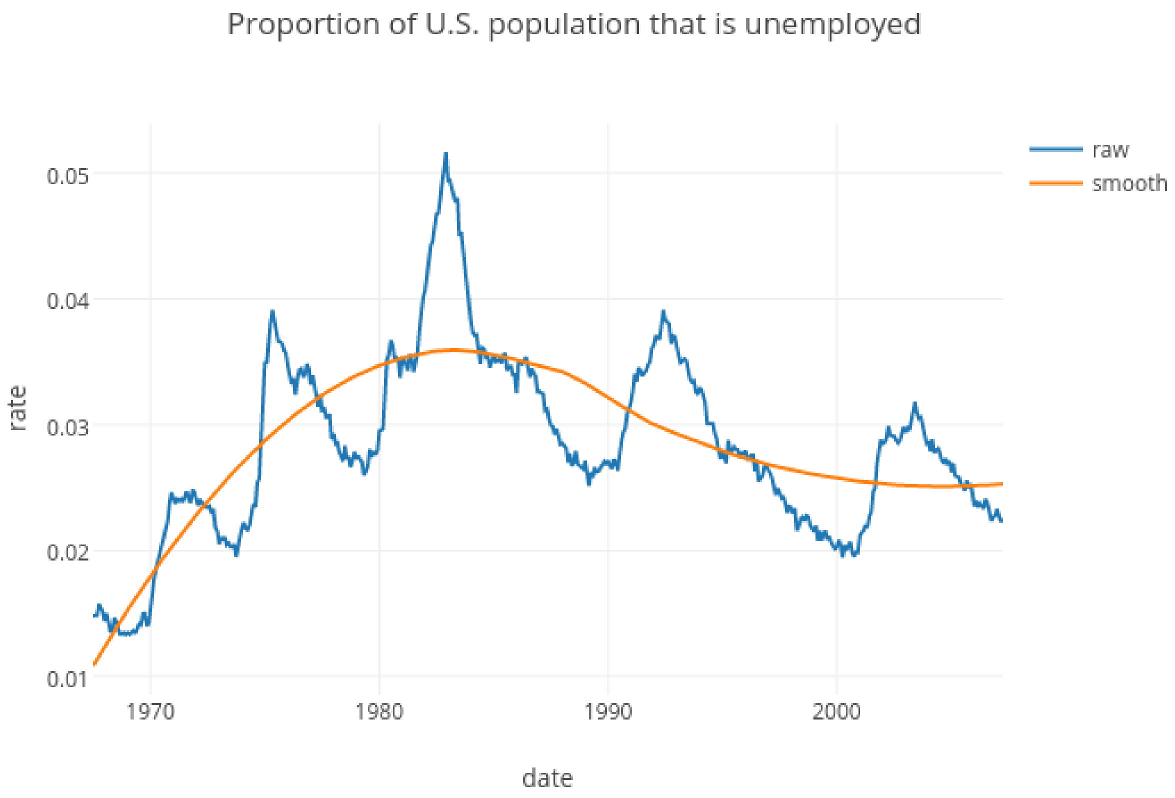
When I started on **plotly**, it's core functionality and philosophy was very similar to **animint**: create interactive web-based visualizations using **ggplot2** syntax (Sievert et al.). However, plotly's `JavaScript` graphing library supports chart types and certain customization that **ggplot2**'s 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 2015c).

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()`.

```r
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")
```



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

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 divert incoming data to a database using any valid R database connection (Databases 2014).

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.

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.

To see the 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

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: clickSelects 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.

Chamberlain, Scott, Thomas Leeper, Patrick Mair, Karthik Ram, and Christopher Gandrud. 2015. "CRAN Task View: Web Technologies and Services." http://cran.r-project.org/web/views/WebTechnologies.html.

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.

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

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. 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.

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, 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. 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.

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

———. 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.

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. *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.

———. 2015c. “Pipelines for Data Analysis.” http://bids.berkeley.edu/resources/videos/pipelines-data-analysis.

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.

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.