Mind the Gap: Leveraging Web Technologies from R

Carson Sievert

2015-11-13

Contents

1 Problem Statement

Web technologies offer exciting new opportunities to advance the field of statistical computing; particularly in the areas of data acquisitions and interactive graphics. In theory, freely available data on the web is accessible; but in practice, one needs an extensive knowledge of web technologies to acquire it. With better software for curating data on the web, we can reduce barriers to access for researchers, and encourage a reproducible workflow.

Reproducibility and portability have remained a challenge for interactive graphics for decades. Open source interactive graphics software is often difficult to install since it typically assumes non-standard software is available on users' systems. As a result, Modern web browsers with `HTML5` support are now ubiquitous, and provide a rich ecosystem for interactive graphics. However, interfacing this ecosystem with statistical software remains difficult and, depending on the application, can require juggling a handful of languages and technologies. Again, with better software, we can reduce the startup costs involved with producing web-based interactive statistical graphics.

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.

2 Overview

This proposal is a collection of work towards making web-based interactive graphics and data on the web more accessible to `R` users. There are a lot of effort being put in this direction. This overview will explain some common approaches, how my work fits into that landscape, and my vision for the future.

I currently maintain packages 7 `R` packages in this direction (and have contributed to a handful of others): **pitchRx**, **bbscrapeR**, **XML2R**, **rdom**, **LDAvis**, **animint**, **plotly**. This section provides a high-level overview of this work within `R`'s broader ecosystem and points out holes remaining to be filled. It also explains my role in joint work such as **LDAvis**, **animint** and **plotly**.

2.1 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 (B. 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 (B. S. 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**.

2.2 Web-based Interactive Graphics

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, a GUI Even for experienced statistical programmers, a GUI can still be useful when the REPL impedes our ability to perform graphical data analysis.

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.

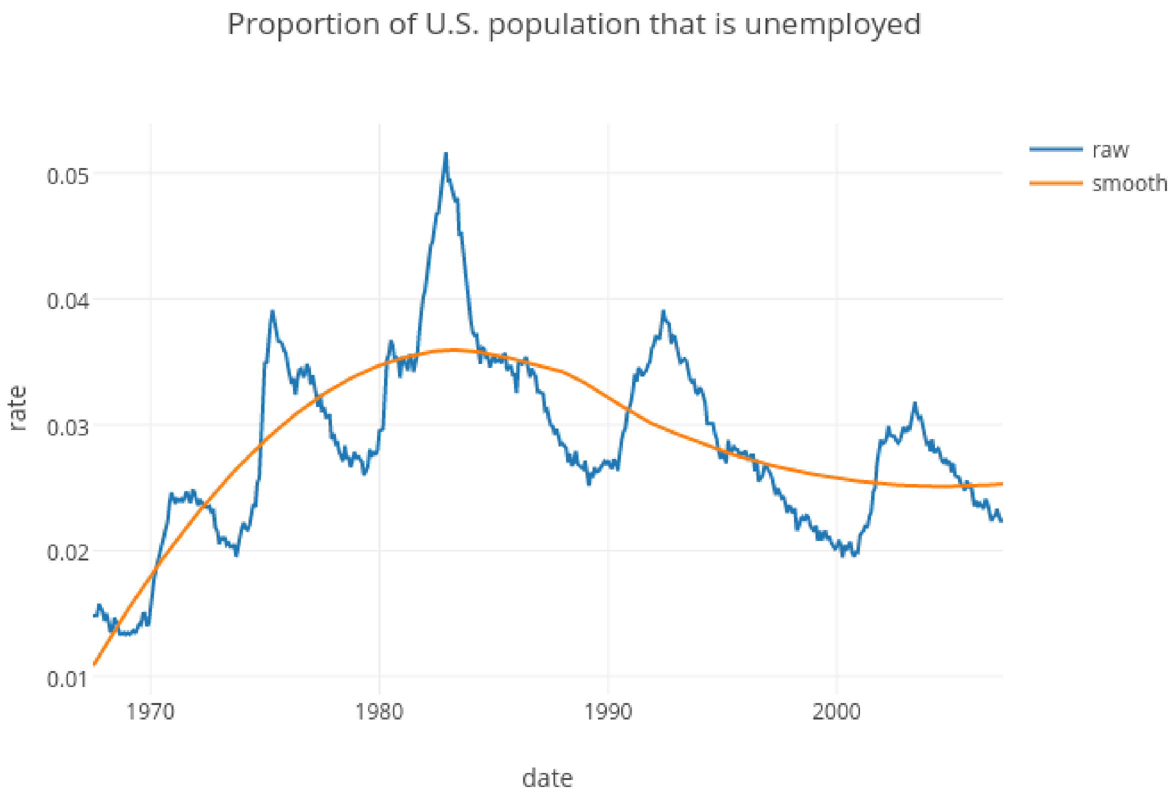
When I started on **plotly**, it's core functionality and philosophy was very similar to **animint**: create interactive web-based visualizations using **ggplot2** syntax (Sievert et al.). However, plotly's `JavaScript` graphing library supports chart types and certain customization that **ggplot2**'s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

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

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: clickSelects and showSelected

Currently in revision. See https://github.com/tdhock/animint-paper/blob/master/HOCKING-animint.pdf

8 Interactive and Dynamic Statistical Graphics for Data Analysis on the Web

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

9 Testing interactive web-based graphics software from R

Work in progress. No link yet.

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.

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.

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. *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.

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

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, 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. 2015. "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.