Co-lab Shiny Workshop

Integrating Shiny and `plotly`

November 5, 2020

thomas.balmat@duke.edu
rescomputing@duke.edu

In previous sessions of the series, we used features of Shiny, `Data Tables`, and `ggplot` to accept input from a visitor to your site and, in the context of analyses implemented, present informative results intended to guide the analyst through further exploration of a data set. One strength of an app developed in Shiny is that analyses, tables, and graphs are prepared with no requirement from the user other than to "fill out" the input form and wait a moment for results to appear. The ease of iterative adjustment of on-screen controls and review of results promotes idea generation and validation, making a well designed Shiny app a resource for exploratory data research. In this session, we will use `plotly` to add hover labels and clickable geoms to a `ggplot` graph, making it more dynamic through filtering, highlighting, and updating of views as the user probes visible features and relationships in the data. Because we are using `plotly` in a Shiny context, our emphasis will be on the interaction of `plotly` features and functions with your Shiny script, particularly within the `server()` function.

1 Overview

- Preliminaries
  - What can Shiny and `plotly` do for you?
  - What are your expectations of this workshop?
- Examples
- Resources
- Anatomy of a Shiny app
- Workshop material
- Review apps from previous sessions
  - `Data Tables` and `ggplot`: Office of Personnel Management Human Capital analysis
  - More `ggplot`: U.S. domestic flight map
- Hello `plotly`
- `plotly` app: GWAS pleiotropy
- Debugging

2 Examples

- `plotly` Visualizations
  - `plotly` gallery: https://plot.ly/r/shiny-gallery/
  - Frank Harrell
    - More with less: https://www.fharrell.com/post/interactive-graphics-less/
    - Hmisc package: https://cran.r-project.org/web/packages/Hmisc/Hmisc.pdf
    - Recent presentation: http://hbiostat.org/talks/rmedicine19.html
- Shiny Apps
  - Duke Data+ project: *Big Data for Reproductive Health*, http://bd4rh.rc.duke.edu:3838
  - Duke Med H2P2 Genome Wide Association Study: http://h2p2.oit.duke.edu

3 Resources

- R
  - Books
    - Norm Matloff, *The Art of R Programming*, No Starch Press
    - Wickham and Grolemund, *R for Data Science*, O'Reilly
    - Andrews and Wainer, *The Great Migration: A Graphics Novel*, https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1740-9713.2017.01070.x
    - Friendly, *A Brief History of Data Visualization*, http://datavis.ca/papers/hbook.pdf
  - Reference cards
    - R reference card: https://cran.r-project.org/doc/contrib/Short-refcard.pdf
    - Base R: https://rstudio.com/wp-content/uploads/2016/10/r-cheat-sheet-3.pdf
    - Shiny, `ggplot`, `markdown`, `dplyr`, `tidy`: https://rstudio.com/resources/cheatsheets/
    - `plotly`: https://plotly.com/r/reference/
- Shiny
  - `?shiny` from the R command line
  - Click `shiny` in the `Packages` tab of RStudio
  - https://cran.r-project.org/web/packages/shiny/shiny.pdf
- `plotly`
  - `?plotly` from the R command line
  - Click `plotly` in the `Packages` tab of RStudio
  - https://cran.r-project.org/web/packages/plotly/plotly.pdf
- Workshop materials
  - https://github.com/tbalmat/Duke-Co-lab/tree/master/Session-4

4 Anatomy of a Shiny App

A Shiny app is an R script executing in an active R environment that uses functions available in the Shiny package to interact with a web browser. The basic components of a Shiny script are

- `ui()` function
  - Contains your web page layout and screen objects for inputs (prompt fields) and outputs (graphs, tables, etc.)
  - Is specified in a combination of Shiny function calls and raw HTML

  - Defines variables that bind web objects to the execution portion of the app

- `server()` function
  - The execution portion of the app
  - Contains a combination of standard R statements and function calls, such as to `apply()`, `lm()`, `ggplot()`, etc., along with calls to functions from the Shiny package that enable reading of on-screen values and rendering of results

- `runApp()` function
  - Creates a process listening on a tcp port, launches a browser (optional), renders a screen by calling the specified `ui()` function, then executes the R commands in the specified server() function

5 Access Workshop Material

Repository: https://github.com/tbalmat/Duke-Co-lab

- Previous session (`Data Tables`): https://github.com/tbalmat/Duke-Co-lab/tree/master/Session-2
- Previous session (`ggplot`): https://github.com/tbalmat/Duke-Co-lab/tree/master/Session-3
- Current session (`plotly`): https://github.com/tbalmat/Duke-Co-lab/tree/master/Session-4

Execute locally:

- Download and expand (in separate local directories) zip file(s) from https://github.com/tbalmat/Duke-Co-lab, session-2, session-3, and session-4 directories, or download individual script and data files as needed
- Edit R source files (`CPDFTables.r`, `FlightEvaluation.r`, `Pleiotropy.r`, `ui.r`, `server.r`) and modify `setwd()` instructions to target directories on your computer
- Launch RStudio
- Install Shiny packages (execute following at R command prompt):
  - `install.packages("shiny")`
  - `install.packages("shinythemes")`
  - `install.packages("ggplot2")`
  - `install.packages("maps")`
  - `install.packages("ggridges")`
  - `install.packages("ggrepel")`
  - `install.packages("plotly")`
- Execute R script(s) (`CPDFTables.r`, `FlightEvaluation.r`, `Pleiotropy.r`)

6 Review apps from previous sessions

The example app for current session uses a single control (a statistical significance level) to filter observations to be selected for plotting. Before proceeding we should review previous apps, which involved many more controls and interacted with R scripts in a number of ways. As we review the example app today, try to imagine how we might incorporate additional controls and techniques employed in previous apps.

6.1 `Data Tables` and `ggplot()`: Office of Personnel Management Human Capital analysis

Features of interest:

- Use of tabs and side bar panels to organize app
- Use of notification window for status reporting
- Use off cascading style sheets
- Use of selection lists (multiple item selection), radio buttons, slider bars, and action buttons to initiate subsetting of observations, aggregating quantiles, and preparing a table of results
- Selection of a tabular row to initialize the plotting tab
- Various controls to promote experimentation with different views of detailed distributions within an subset of observations
- Use of an animation slide bar to dynamically shift views through independent axis values (session 1, CPDF-FYSliderBar)

6.2 `ggplot()`: U.S. Domestic Flight Map

Features of interest:

- Use of a map drawn with `geom_polygon()`
- Use of various line color, weight, and transparency levels to improve contrast of flight routes
- Use of faceting to subset plots by weekday, season, and carrier
- Selectable proportions by all flights or delayed flights
- Use of `ggrepel()` to prevent overlap of airport labels
- Plotting of density ridges to compare flight delay by weekday, season, or carrier
- Computation of arrival and departure delay difference to evaluate crew effectiveness
- Adjustment of y-axis order (natural, mean, or median)
- Overlay of vertical reference lines at zero, mean, or median
- Use of `ggplot_build()` to extract x-y coordinates of density ridges for median computation and vertical line placement

7 Hello `plotly`

```
library(ggplot2)
library(plotly)

x <- sample(1:100, 100, replace=T)
y <- 25 + 5*x + rnorm(length(x), sd=25)

g <- ggplot() +

geom_point(aes(x=x, y=y, msg="hi"))
gp <- ggplotly(g)
```

8 `plotly` App: GWAS Pleiotropy

Within a single genome wide association study (GWAS), a researcher may identify single nucleotide polymorphisms (SNP)s (positions on a chromosome), such that the configuration of genotype at a SNP appears correlated with phenotypic (disease, trait) response. With two independent GWAS studies, typically with disjoint sets of phenotypes, a researcher might find phenotypes in one GWAS that appear to be associated, by SNP, with phenotypes in the other GWAS. This process is termed "pleiotropy."

Workshop file: App/Pleiotropy/Pleiotropy.r

Figure 1 is an example screen shot of tab 1 of the pleiotropy app. Clicking phenotype/SNP points in the left hand (GWAS 1) plot identifies, with contrasting color and informative labels, phenotype points in the right hand plot (GWAS 2) that are associated by SNP.

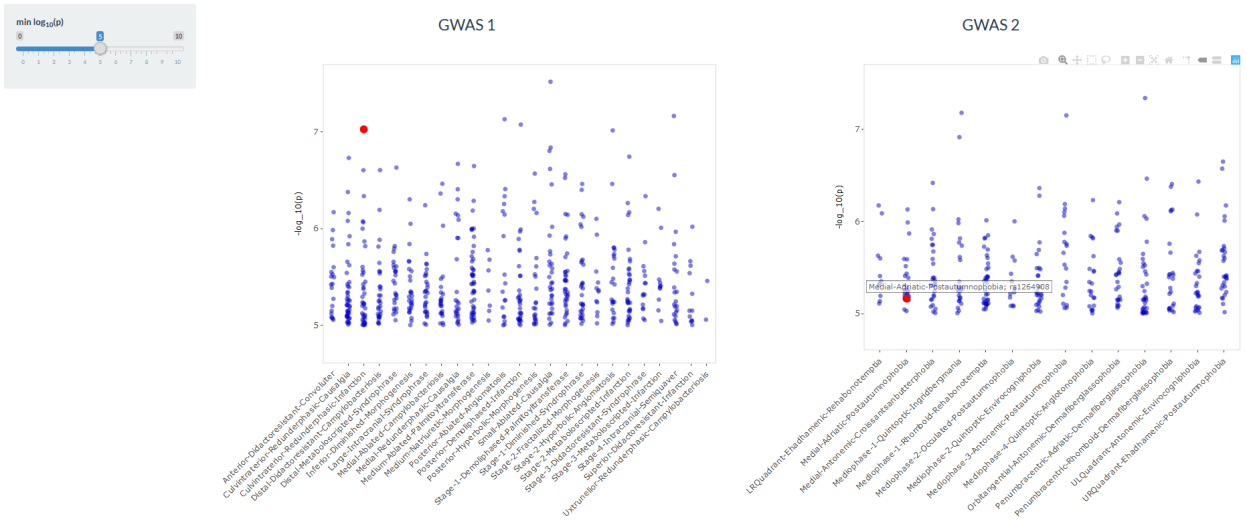

Figure 1: `plotly` pleiotropy app, tab 1, cross-GWAS phenotype SNP associations.

Important `plotly` features (numbers are lines in `Pleiotropy.r` source file):

- (69) `geom_jitter()`
- (85) Axis text angle
- (69) User defined variable in `aes()` for appearance in hover labels
- (151) Use of `plotly()` %>% to string function calls
- (380, 425) Use of `ggplotly()` to convert a `ggplot()` object to `plotly()`
- Hover label configuration (position and ID of aesthetics)
- (380, 388) Point selection and control of events
- (396) Subsetting observations for selected point
- (102) Coloring selected points using two geoms and data subsets in `t1ComposePlot2()`
- (411) Adding custom labels with `add_annotations`, since `ggplotly()` does not convert them
- (420-425) Nested rendering of plots

Figure 2 is an example screen shot of tab 2 of the pleiotropy app. After filtering within-GWAS associations by significance (p), groups of inter-GWAS phenotypes are identified by clicking points in either set (left and right vertical blue dots). Hovering over blue dots causes display of SNPs associated with a phenotype (within GWAS). Hovering over a green dot (on an edge, or line connecting the GWAS sets) causes display of SNPs that are associated with the connected phenotypes (between GWAS).

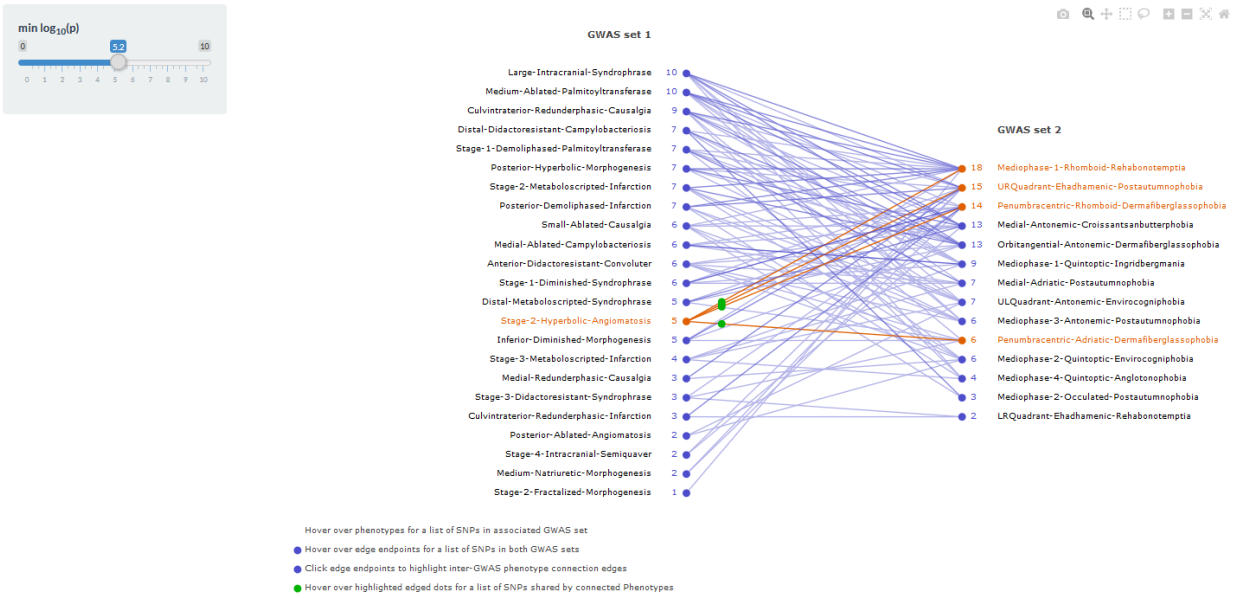

Figure 2: `plotly` pleiotropy app, tab 2, identification of phenotype association groups. Associated SNPs appear as points are hovered over.

Important `plotly` features:

- Subsetting plotted points by p-filter
- Phenotype ordering by number of associated SNPs
- Display of number of SNPs (`geom_text()`)
- Phenotype point hover labels
- Coloring vertices (phenotype) and edges (lines) of selected point (left and right)
- Display of green intersection point
- SNP intersection of lines, hover labels

9 Debugging

It is important that you have a means of communicating with your app during execution. Unlike a typical R script, that can be executed one line at a time, with interactive review of variables, once a Shiny script launches, it executes without the console prompt. Upon termination, some global variables may be available for examination, but you may not have reliable information on when they were last updated. Error and warning messages are displayed in the console (and the terminal session when executed in a shell) and, fortunately, so are the results of `print()` and `cat()`. When executed in RStudio, Shiny offers sophisticated debugger features (more info at https://shiny.rstudio.com/articles/debugging.html). However, one of the simplest methods of communicating with your app during execution is to use `print()` (for a formatted or multi-element object, such as a data frame) or `cat(, file=stderr())` for "small" objects. The `file=stderr()` causes displayed items to appear in red. Output may also be written to an error log, depending on your OS. Considerations include

- Shiny reports line numbers in error messages relative to the related function (`ui()` or `server()`) and, although not always exact, reported lines are usually in the proximity of the one which was executed at the time of error
- `cat("your message here")` displays in RStudio console (generally, consider Shiny Server)
- `cat("your message here", file=stderr())` is treated as an error (red in console, logged by OS)
- Messages appear in RStudio console when Shiny app launched from within RStudio
- Messages appear in terminal window when Shiny app launched with the `rscript` command in shell
- There exists a "showcase" mode (`runApp(display.mode="showcase")`) that is intended to highlight each line of your script as it is executing
- The reactivity log may be helpful in debugging reactive sequencing issues (`options(shiny.reactlog=T)`, https://shiny.rstudio.com/reference/shiny/0.14/showReactLog.html It may be helpful to initially format an apps appearance with an empty `server()` function, then include executable statements once the screen objects are available and configured
- Although not strictly related to debugging, the use of `gc()` to clear defunct memory (from R's recycling) may reduce total memory in use at a given time