Co-lab Shiny Workshop 3, Spring 2021

Session 3, Spring 2021

Integrating Shiny and `plotly`

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 that your users understand R. They simply "fill out" the input form you have designed and wait for results of your R script 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 a dynamic filter, highlight, and point association app that enables probing of relationships between two data sets. Because we are using `plotly` in a Shiny context, our emphasis will be on the interaction of `plotly` features and functions with a Shiny script.

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
- Reactivity
- Download Workshop Material and Configure R
- Review Apps from Previous Session
  - Data Tables and `ggplot()`
  - Fiscal Year Slider Bar
- 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/Spring-2021/Session-3-plotly

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 Reactivity

Reactivity is the single most important feature that Shiny offers. Variables are defined in your `ui()` function with an `input$` prefix and when these variables appear in `observe()` functions within in your `server()` function, execution events are triggered by on-screen changes to the corresponding `ui()` variables. In addition to referencing `input$` variables `observe()` functions include standard R commands, including those supported by any valid R package, so that the reactive variables become parameters to your R functions, enabling dynamic analysis of data. Output is rendered in the app by targeting `ui()` variables defined with an `output$` prefix. A simple example follows. It has a single, numeric input (`x`) and one plot output (`plot`). Changes in `x` cause the `observeEvent()` to be executed. The `observeEvent()` generates a histogram of `x` random, normal values. The histogram is a suitable input value to `renderPlot()`. Assignment of the `renderPlot()` result to `output$plot` causes the histogram to be displayed as defined in `ui()`. Notice how a Shiny input variable (`input$x`) is used as a parameter to an R function (`rnorm()`) and the result of an R function (`plot()`) is used as a parameter to a Shiny function (`renderPlot()`). Note that modifying `x` to its current value does not cause execution of the `observeEvent()` (try it).

```
library(shiny)

# Define UI
ui <- function(req) {
  numericInput(inputId="x", label="x"),
  plotOutput(outputId="plot")
}

# Define server function
server <- function function(input, output, session) {
  observeEvent(input$x, {
    output$plot <- renderPlot(hist(rnorm(input$x)))
  })
}

# Execute
runApp(list("ui"=ui, "server"=server), launch.browser=T)
```

6 Download Workshop Material and Configure R

- Copy course outline, scripts, and data from https://github.com/tbalmat/Duke-Co-lab/tree/master/Spring-2021/Session-3-plotly
  - App.zip
  - Co-lab-Session-3-plotly.pdf
  - Data.zip

- Expand zip files (one subdirectory per file)
- Launch RStudio
- Install packages:
  - `install.packages("shiny")`
  - `install.packages("shinythemes")`
  - `install.packages("plotly")`
  - `install.packages("ggplot2")`

7 Review Apps from Previous Session

7.1 Data Tables and `ggplot()`

Script loc: https://github.com/tbalmat/Duke-Co-lab/tree/master/Spring-2021/Session-1-2-DataTables-Plots

Files: App/V5 and App/AdvancedDataTableFeatures

Features to discuss:

- User file search and upload capability
- Download link for review of sample input file format
- Aggregation table download capability
- Progress indicators
- Dynamic insertion of HTML windows containing supplemental project info and links
- Error handling to avoid simple crashes
- Modal windows for communication with user
- Advanced data table features

7.2 Fiscal Year Slider Bar

Script loc: https://github.com/tbalmat/Duke-Co-lab/tree/master/Spring-2021/Session-1-2-DataTables-Plots

File App/V9-CPDF-FYSliderBar.r

Features to discuss:

- Animation of fiscal year
- Regeneration of plot for each year
- Visualization of trends as fiscal year advances

8 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)
```

9 `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."

Script location: https://github.com/tbalmat/Duke-Co-lab/tree/master/Spring-2021/Session-3-plotly

File: App.zip

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.

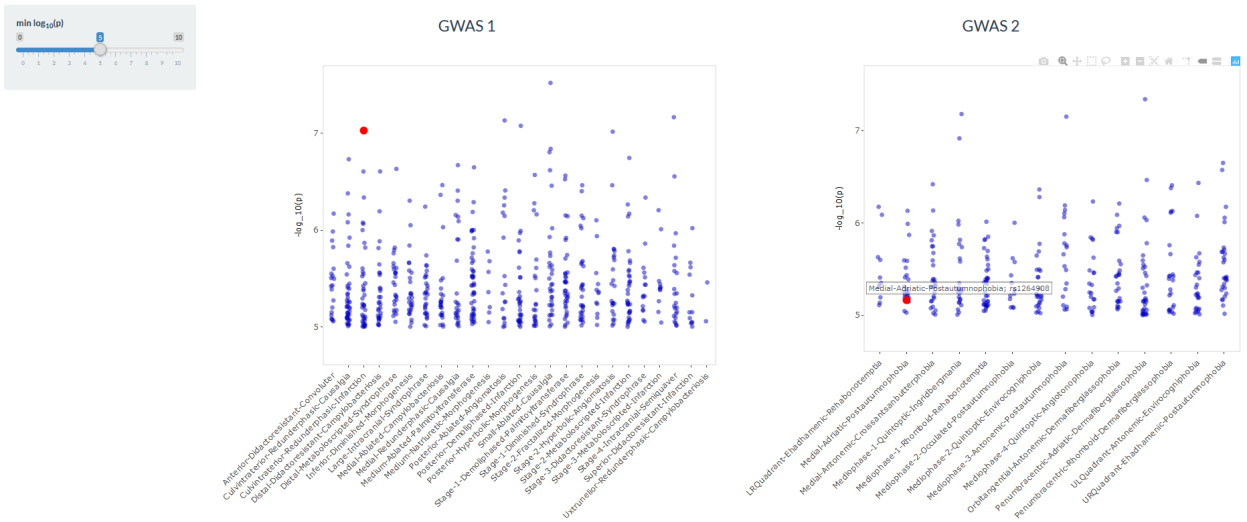

Figure 1: `plotly` pleiotropy app, tab 1, cross-GWAS phenotype SNP associations.

Important features (snnn indicates line number in `server.r`, fnnn indicates line number in the functions script):

- (f38, f76) `ggplot geom_jitter()` and `alpha` used to avoid point overlap
- (f54, f98) Axis text angle
- (Hover label configuration (position and ID of aesthetics)
- (f38, f42, f77, f81) Hover labels automatically configured `aes()` variables (note the inclusion of a user defined variable, SNP)
- (f41) Use of color to highlight selected points
- (s34, s65, s82) Use of `ggplotly()` to convert a `ggplot()` object to `plotly()`
- Point selection and control of events
  - (s34) Plot 1 is rendered with source ID "t1Plot1"
  - (s43) A reactive event data object is created by the click event on a plot 1 point

  - (s46-91) The selected plot 1 point (defined in the event data) is used to highlight associated points on plot 2
  - (s77-82) The selected plot 1 point is highlighted (note the specification of source ID "t1Plot1" again)
- (s37, s82) Nested rendering of plots
- (f76) Subsetting observations for selected point (`setdiff()`)
- (f80-81) Coloring selected points using two geoms and data subsets in `t1ComposePlot2()`
- (f83, s71) Custom labels are added with `add_annotations`, since `ggplotly()` does not convert them

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 features:

- Use of `plotly()` %>% to pipe output of one function into another
- Use of traces (similar to geoms) to display points, lines, and labels
  - (f121-168) Phenotype labels
  - (f237-247) Vertices (note the update of the graph object created during label trace addition)
  - (f180-198) Edges connecting vertices
  - (f203-232) Selected points
- 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

10 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