Co-lab Shiny Workshop

Integrating Shiny and `ggplot`

November 7, 2019

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

In previous sessions of the series, we have used features of Shiny to accept input from a user combined with features of `ggplot` to present results corresponding to the user's input, in the context of data and analyses implemented in the apps we have reviewed. In this session, we will take a closer look at the properties and possible uses of the `ggplot` "geoms" we have used, along with faceting (creating multi-panel plots), controlling aesthetics (defining which variables are colored, sized, etc., based on their levels), configuring axes, and adjusting themes for overall appearance. Because we are using `ggplot` in a Shiny context, our emphasis will be on binding on-screen controls to Shiny script variables that are used to initialize parameters of various `ggplot` functions.

1 Overview

- Preliminaries
  - What can `Shiny and ggplot` do for you?
  - What are your expectations of this workshop?
  - Interest in R Day?
- Examples
- Resources
- Anatomy of a Shiny app
- Workshop material
  - From github (execute locally)
  - From RStudio Cloud
- Review previous app, data tables and the OPM Central Personnel Data File
- `ggplot` app: U.S. domestic flight map
- Debugging

2 Examples

- `ggplot` Visualizations
  - `ggplot` gallery: `https://www.r-graph-gallery.com/all-graphs.html`
  - `ggplot` extensions: `https://www.ggplot2-exts.org/gallery/`
- 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
  - Show me Shiny: https://www.showmeshiny.com/
  - Shiny gallery: https://shiny.rstudio.com/gallery/

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/
- 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
- `ggplot`
  - `?ggplot2` from the R command line
  - Click `ggplot2` in the `Packages` tab of RStudio
  - https://cran.r-project.org/web/packages/ggplot2/ggplot2.pdf
- Workshop materials
  - https://github.com/tbalmat/Duke-Co-lab/tree/master/Session-3

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

5.1 Execute Locally (copy workshop material from github repo)

- Copy scripts and data from https://github.com/tbalmat/Duke-Co-lab/tree/master/Session-2
- In a separate directory, copy scripts and data from https://github.com/tbalmat/Duke-Co-lab/tree/master/Session-3
- Install Shiny package: R command `install.packages("shiny")`
- Install plotting package: R command `install.packages("ggplot2")`
- Install plotting package: R command `install.packages("DT")`
- Install data tables package: R command `install.packages("maps")`
- Install data tables package: R command `install.packages("ggridges")`
- Install data tables package: R command `install.packages("ggrepel")`
- All workshop scripts should function locally

5.2 RStudio Cloud

- What is RStudio Cloud?
  - *We* [RStudio] *created RStudio Cloud to make it easy for professionals, hobbyists, trainers, teachers and students to do, share, teach and learn data science using R.*
- With RStudio Cloud
  - You do not need RStudio installed locally
  - Packages and data are available without installation and transfer
- Access workshop material
  - Create an Account: https://rstudio.cloud
  - Workshop project link: https://rstudio.cloud/project/580472
- All workshop scripts should function on RStudio Cloud, except OS shells that are used to automate execution

6 Review Data Tables and OPM CPDF Example from Session 2

Material: https://github.com/tbalmat/Duke-Co-lab/blob/master/Session-2, sections 8.2 and 8.3, discuss and experiment with:

- Table configuration parameters
- Data subsetting and filtering
- Table row and cell click events
- Data subsetting and filtering
- `ggplot` controls and plot appearance

7 `ggplot` App: U.S. Domestic Flight Map

Overview is available at:

https://github.com/tbalmat/Duke-Co-lab/blob/master/Session-3/Docs/FlightEvaluationShinyAppOverview-01.pdf

Discussion points:

- User interface (`ui.r`) features of interest
  - Use of `col()` for aligned placement of selection objects above plots
  - Use of `fluidRow()` for column within row formatting
  - Use of `div()` for CSS styles (width, alignment, margins)
- Server (`server.r`) features of interest
  - Use of a separate source file for functions (FlightEvaluation-Functions.r)
  - Functions for retrieving data, aggregating data, and composing plots
  - Reactive elements bound to R variables within `observe()` events, so that modifications of input objects (selection lists, slider bars, etc) cause immediate update of plots
  - Observation and label filtering results from change of on-screen p selectors
  - Arc size, color, and transparency set by on screen controls (they supply values to `aes()` parameters in `ggplot()` calls)
  - Faceting controlled by on-screen radio buttons
  - Facet labels are composed for use as a labeler in `facet_wrap()`
  - Overall rendered plot size is adjusted for facet panels displayed
- `ggplot` features of interest
  - Flight map (tab 1)
    - `geom_polygon()` used to draw a map from coordinates contained in a U.S. state map data frame
    - `geom_curve()` is used to draw arcs between flight origin and destination x-y (lat-long) coordinates (with color, size, and alpha aesthetics and fixed curvature and arrowheads)
    - Custom scales are defined for arc size, color, and alpha using on screen values (these control arc appearance in plots and legends)
    - Airport labels are added with `geom_label_repel()` to avoid label overlay
  - Density ridges (tab 2)

    * x-axis (measure of time), y-axis (categories forming ridges), and faceting variables specified on-screen
    * y and color orders are reversible
    * x-axis limits and distribution transparency are controllable
    * `geom_density_ridges()` is used to compose ridges form x (time) for each level of the selected y variable, with fill color specified as an aesthetic for y (each level of y receives a distinct color)
    * `scale_fill_manual()` is used to define a fixed set of colors composed from a call to `colorRampPalette()`, requesting a range of colors from the low and high range specified on-screen, with a number of gradations equal to the number of distinct levels of y
    * Vertical lines at mean or median time values are computed by using the density distributions composed by `geom_density_ridges()`. `ggplot_build()` is used to extract the densities for all ridges and x and y coordinates are used for mean and median computation. Distribution height is then used as an endpoint and individual line segments are plotted using `geom_segment()`.
  - Internals
    * `map_data()` function from the maps package: https://cran.r-project.org/web/packages/maps/maps.pdf
    * `ggplot_build()`: https://www.rdocumentation.org/packages/ggplot2/versions/3.2.1/topics/ggplot_build
    * `ggplot` innards: https://cran.r-project.org/web/packages/gginnards/vignettes/user-guide-2.html

8 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