Building dashboards in R/Shiny

Kimberly Zhang

July 7, 2024

Presentation Overview

1. Why R/Shiny?
2. Shiny Basics
3. Beyond the Basics
   - UI Features
   - Visuals
   - Click and hover events
   - Tables
   - Reactive Expressions
   - Caching
   - Debugging
4. Data manipulation

Why R/Shiny?

- Shiny gives R users the power to build a dashboard without prior knowledge of HTML, CSS, and JavaScript, but retain the ability to use them if needed

Shiny Basics

- Start with some basic examples from the Shiny gallery like the telephones by region dashboard
- ui.R is your “road map” for every feature in the dashboard
- server.R connects user inputs from the widgets set up in the UI to calculations in the server through the inputId argument

UI Features

- Widgets
- Even more widgets
- Progress bars

Visuals

- The `ggplot` and `plotly` packages are commonly used to create scatter plots, line plots, box plots, etc.
- These packages also have some specialized charts like candlestick plots, which are helpful for visualizing stock prices.
- These charting libraries are based on principle of “layering” visualization elements, and give developers tremendous flexibility.

Click and hover events

- Use `event_data()` to create linked events (e.g., dynamically generate a table or chart based on something a user clicked in a plot)
- See this documentation and interactive example for more information on the types of events (e.g., hover, click, brush)
- See Returns by Sector for an example of how to link a plot with a table

Tables

- Make pretty tables with the `reactable` package (e.g., 2019 Women's World Cup Predictions)
- For very large tables, use `DT::datatable()` with the option server set to TRUE so that the browser receives only the displayed data.

Reactive Expressions

- Reactive expressions use inputs from UI widgets and return a value
- The key advantage is lazy evaluation, which prevents the server from running code unnecessarily.
  - The code inside the reactive expression is never run if the reactive isn't called.
  - The output of a reactive expression is cached the first time it's run.
  - The reactive expression will only be re-run if the server detects a change in any of the input values or other reactives inside the reactive expression.

Key Advantage of Reactive Expressions

What's the difference?

- getData <- reactive({
  Pull data based on input$a
  Filter, sort data based on input$b
  Run calculations on data based on input$c
  })

- pullData <- reactive({Pull data based on input$a })
- filterSortData <- reactive({ Filter, sort pullData() based on input$b })
- calcData <- reactive({ Run calculations on filterSortData() based on input$c })

Caching

- Use bindCache() to improve performance via caching
- Important to carefully select cache keys, which will determine when cache needs to be refreshed. For example:
  - Sys.Date() (today's date) to refresh cache file once per day
  - Last modified date and time for a file
  - Input values

Debugging

- Place the `browser()` function inside the server wherever you want to pause the server and investigate further
- Use `renderPrint()` and `verbatimTextOutput()` to print values and display them directly in the UI
- Use reactive log to understand order in which reactives are being called
- For more details: https://shiny.posit.co/r/articles/improve/debugging/

Data manipulation

- Aggregation (e.g., operations like summing, grouping) using `dplyr` or `data.table`
- melt() and dcast() functions from `reshape2` to transform data between “long” and “wide” format
- merge() for joining tables