

# Building dashboards in R/Shiny

Kimberly Zhang

June 12, 2024

# Presentation Overview

## 1 Why R/Shiny?

## 2 Shiny Basics

## 3 Beyond the Basics

- 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 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 and help users dive deeper
- See this [documentation](#) and [interactive example](#) for more details

# 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

- 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 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 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 reactivities 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 “short” format
- `merge()` for joining tables