Shiny: Part 2

Daniel Anderson
Week 9, Class 1

UNIVERSITY OF OREGON

Agenda

- Review Lab 3 (quickly)
- Review Lab 4
- Review shiny basics
- Discuss some layout options
- Introduce shiny dashboard
- Some extensions worth exploring later

Learning objectives

- Solidify your understanding of the UI/Server relation
- Be able create tabsets and a navbar
- Be able to create basic shiny dashboards

Review labs

Basic shiny

Create a basic shiny template, as we did before.

03:00

UI

- The `ui` defines the look and feel of the app - the user interface
- Use it to define where output "lives"
- It also defines the inputs for the `server`, where functions are actually evaluated.
- In the template/default case, a `sliderInput` has been defined, which we're calling `"bins"`. It will take on values from 1 to 50, and will start at 30.
- Access the specific value the user selects within the `server`, through `input$bins`.

Server

- The `server` function takes the `input` from the UI and produces `output` with normal R code.
- In this case, we're creating one output object, called `distPlot`. The result is then called through the `ui` on line 30

Change the input

Let's switch from a slider to a drop down menu.

How?

Even if you don't know the specific code, what would we change?

`sliderInput` will become `selectInput`. The arguments will also be slightly different

Try!

02:00

layouts

Tabs

- Let's say we wanted a tabset with different things.
- First, we need at least two things!
- Let's create a table that has the lower/upper bound of the bin, and the counts within that range.

Table creation

- Because this is base syntax, I'll give you the basics for the table, you focus on the shiny part

```
input_bins <- 30 # placeholder for whatever the input is

bins <- seq(min(mtcars$mpg),
            max(mtcars$mpg),
            length.out = input_bins + 1)

h <- hist(mtcars$mpg, breaks = bins, plot = FALSE)

tibble(lower = lag(h$breaks),
       upper = h$breaks) %>%
  drop_na(lower) %>%
  mutate(counts = h$counts) %>%
  mutate_if(is.numeric, round, 2)
```

- You take it from here! Add a table below the plot
  - use `DT::datatable`
- Create the bins within each `render*`

04:00

[demo]

Move it to a tabset

- Just create a `tabsetPanel` within the `mainPanel`, then put the output for each tab within `tabPanel`.

05:00

Different pages

Add a navbar

- Instead of using a tabset with `tabsetPanel`, you might want to have a navbar at the top of the page, which you can create with `navbarPage`.
- Can be a bit more complicated - each `tabset` needs to include everything, including the `sidebarPanel` (if present), could include tabsets, `mainPanel`, etc.
- Essentially each tab from the `navbar` becomes an entirely new page.

More on the navbar

- Can really help with organization/flexibility (you could even have tabs within a page)
- Refactoring can help organization A LOT
    - Pull pieces out to try to make code more readable/less buggy.

[example]

{shinydashboard}

Example

This is a shiny dashboard I created for the Lane Early Learning Alliance

Get started

- Go here

First dashboard - ui

```r
library(shiny)
library(shinydashboard)

ui <- dashboardPage(
  dashboardHeader(title = "Basic dashboard"),
  dashboardSidebar(),
  dashboardBody(
    # Boxes need to be put in a row (or column)
    fluidRow(
      box(plotOutput("plot1", height = 250)),

      box(
        title = "Controls",
        sliderInput("slider", "Number of observations:", 1, 100, 50)
      )
    )
  )
)
```

First dashboard - server

```r
server <- function(input, output) {
  set.seed(122)
  histdata <- rnorm(500)

  output$plot1 <- renderPlot({
    data <- histdata[seq_len(input$slider)]
    hist(data)
  })
}
```

Run it

```r
shinyApp(ui, server)
```

[demo]

Main differences

- You now have `dashboardSidebar` and `dashboardBody`
- You also now have `fluidRow` and `box` arguments to arrange things in the main body

Sidebar

- Probably the defining characteristic of the dashboard
    - Define a `sidebarMenu` with `menuItems`

Example

```
sidebarMenu(
  menuItem("Histogram", tabName = "histo", icon = icon("chart-bar")),
  menuItem("Bin Counts", tabName = "bins", icon = icon("table"))
    )
```

You can also do things like put the slider in the `sidebarMenu`

[demo]

Referencing menu items

- If you define `menuItem`s, you'll have to give them a `tabName` (see previous slide).

- In the `dashboardBody`, create a `tabItems` with specific `tabItem` pieces. This should be how you control/refer to the `menuItem`.

[demo]

Extension

Consider {shinydashboardplus}

Example

Other layouts/themes

The RinteRface group founded by David Granjon has built a couple other packages that can help you make really fancy looking apps pretty efficiently/easily.

bs4dash

argonDash

And from Ian lyttle, there's the bsplus package for increased fanciness

Conclusions

- Shiny is super customizable - almost limitless (see more examples here)
- Great for building interactive plots, but you can use it for all sorts of other things too (including text and tables)
- Really helpful and fun way to build data tools for practitioners
- Consider styling with shinythemes
- If you end up wanting to go deep with shiny, you may want to read more about reactivity