Shiny Tutorial

Sophia Zheng '20

December 2019

About This Guide

This guide explores the basics of Shiny App, specifically as it pertains to the final project for **Gov 1005: Data**, which I took in the Fall of 2019. I use the Preceptor's sample repository on the NRA for example.

Helpful Resources

As the Preceptor points out in the syllabus, there are a number of resources on Shiny to get you started, so definitely take a look at the following materials in addition to using this guide:

1. Shiny Video Tutorials: this is a comprehensive video tutorial that goes through all of the basics of building a Shiny app and adding customization. It is 2 and a half hours long, but individual chapters can be found at the same link for walk throughs on smaller topics. The first 40 or so minutes of the tutorial are particularly helpful in getting a basic app running and available on shinyapps.io.
2. Shiny Written Tutorials: this tutorial is also quite comprehensive, and walks through templates for the example Shiny document. It breaks down the set up of UI, control widgets, reactive output (like dropdown variables), using R scripts and data, using reactive expressions, and sharing the app.
3. *Mastering Shiny* by Hadley Wickham: this book is a really nice guide to using Shiny and best practices for higher-quality apps, including reducing code duplication.

Set Up and Getting Started

- Sign up for a Shiny account at https://www.shinyapps.io/.
- Install the Shiny package with: `install.packages("shiny")`
- Create a Shiny app from RStudio with: `File > New File > Shiny Web App...`
  - In the popup, choose Single File (app.R) - this doesn't particularly matter, but is the way that the Preceptor taught in class, and is the way that his example is set up. If you would prefer to have two separate files for ui and server, this is also valid and might allow for cleaner code.
- You should now have a functioning example Shiny App - click `Run App` or use the keyboard shortcut `Cmd/Ctrl + Shift + Enter` to see this app in action! You can view it in a new window, or in an external web browser (see the dropdown arrow next to `Run App` for the

options). Notice that the example includes a slider that allows the viewer to change the number of the bins that the histogram has.

Building Your Basic App

Now that you have your functioning Shiny App up and running, let's take a closer look at our files and directories. Notice that the app.R file has been created within a directory. It is important to remember that only the files within this directory will be accessible to the app when it is online. Taking a look at the app.R file, we can see that there are four necessary elements in this file that create the working app:

1. First, it calls `library(shiny)`
2. Then, it defines a user interface with `ui <- fluidPage(...)`
3. It also defines server logic, which takes in an input from the UI, and produces an output based on that input, as defined by `server <- function(input, output) {...}`
4. Finally, it calls `shinyApp(ui, server)` to run the app.

Setting Up the Basic UI

According to the Preceptor, the Gov 1005 Final Projects should be formatted such that they have three tabs, as well as an embedded video. A basic set up for that might be something like this:

```
ui <- navbarPage(
  "Final Project Title",
  tabPanel("Model",
           fluidPage(
              titlePanel("Model Title"),
              sidebarLayout(
                  sidebarPanel(
                      selectInput(
                          "plot_type",
                          "Plot Type",
                          c("Option A" = "a", "Option B" = "b")
                      )),
                  mainPanel(plotOutput("line_plot")))
            )),
  tabPanel("Discussion",
           titlePanel("Discussion Title"),
           p("Tour of the modeling choices you made and
              an explanation of why you made them")),
  tabPanel("About",
          titlePanel("About"),
          h3("Project Background and Motivations"),
          p("Hello, this is where I talk about my project."),
          h3("About Me"),
          p("My name is ______ and I study ______.
             You can reach me at ______@college.harvard.edu.")))
```

You can run the example app in this repo to see this code in action. Breaking this code down, we have created a navigation bar page with three tabs. The first tab will display the model, and has two potential input options, Option A or Option B. This can be customized based on your model, and the number of inputs desired. Note that this is wrapped inside of a `fluidPage()`, which simply creates a basic page layout that can support rows and columns if so desired. Breaking down this tab further, we have `sidebarLayout()` defining the layout of this page with a sidebar panel and a main panel. Within the sidebar panel, we have inserted a `selectInput`, which will allow us to have reactive pages.

Note that I have used HTML formatting for the Discussion and About pages.

Setting up the Server

The server function is where we perform our backend logic for our app. We can put any R code there to create plots or tables as needed. The inputs come from UI definitions, like `selectInput()` or `sliderInput()`. In our UI example, we used a select input named "plot_type", which had two options: A or B. The outputs are defined within the server function, and then rendered as defined in the UI. In our UI example, we want to render a `plotOutput` named "line_plot".

So our server function would ideally use the \texttt{plot_type} input to create a \texttt{line_plot}. An example of a basic server function might be something like this:

```
server <- function(input, output) {
    output$line_plot <- renderPlot({
        # generate type based on input$plot_type from ui
        ifelse(
            input$plot_type == "a",
            # if input$plot_type is "a", plot histogram of "waiting" column
            # from the faithful dataframe
            x   <- faithful[, 2],
            # if input$plot_type is "b", plot histogram of "eruptions" column
            # from the faithful dataframe
            x   <- faithful[, 1]
        )
        # draw the histogram with the specified number of bins
        hist(x, col = 'darkgray', border = 'white')
    })
}
```

Organization

Now that we have the basic app running, let's talk about organization. One common problem that shows up a lot is that people tend to throw all of their code into their app.R file, causing it to grow to be hundreds or even thousands of lines of code long. This happens easily because not only can backend logic be written into the server function, but additionally entire HTML or markdown pages can be formatted in the UI section. Here are a few ways to stop the app.R file from becoming incredibly messy.

1. First, you can put all large blocks of text into separate files. For example, we could take the HTML formatting from the "About" tab section and put it into a separate about.html file, or similarly format the same section using markdown. Then you would simply use the shiny function `includeHTML()` or `includeMarkdown()` to insert your file contents. See the Preceptor's NRA repository for an example of this.

2. Second, put as much prep work as possible into a separate (or multiple separate) R file. This is helpful for readability, as it will generate more small files, but it is also helpful in that the app will not have to run all of the heavy prep calculations while it's trying to load. For this, a tip is to perform heavy operations in a document titled prep-shiny.R, for example, and to save any helpful outputs as graphic images or gifs, or as rds files that can then be loaded into the app.R file.

3. Third, keep your repository clean. Do not save unneccessary raw data files or the like to your repository, as this may just create clutter. Make sure to update your .gitignore file with anything that you would not like to have on your Github repository or on the Shiny App. This includes the .Rproj file, raw data files, and cache files.

Customizations

Finally, we can take a look at a simple way to customize the look of your Shiny App. The package `shinythemes` can easily be added to give your app a theme. The shinythemes website displays the various themes quite nicely, and you can use the theme selector to click through different themes.

The Shiny gallery is helpful for browsing through different layouts and types of visualization for inspiration.