Tips and Tricks to Buff Up Your Shiny App

Nicholas Weis

December 3, 2017

Shiny applications are one of the most useful tools at a statistician's disposal, customized and crafted to best represent the associated data sets. When visualizing data, sometimes there may be no existing template that perfectly fits your vision. It is in these cases that it is highly recommended that one has a knowledge of Shiny's ins and outs. In this post, I would like to spotlight a few of the little-known tips and tricks for decking out your Shiny app with all the most pertinent bells and whistles. By the end of this lesson, you should be able to make the application that best fits your individual needs.

1.) Tabs

Organization of information is key to an effective shiny application, and tabs are a simple and clean way of achieving this goal. Any use of tabsets is a step up over a format that forgoes that tool entirely. However, there are ways to further simplify and enhance your Shiny application by organizing your tab code itself more succinctly. Here's an example of code that does just that:

server <- function(input, output) {

This part can be in different source file for example component1.R output$component1 <- renderUI({ sidebarLayout(sidebarPanel( ), mainPanel(tabsetPanel(tabPanel("Plot", plotOutput("plot")), tabPanel("Summary", verbatimTextOutput("summary")), tabPanel("Table", tableOutput("table"))))) })

} ui <- shinyUI(navbarPage("My Application", tabPanel("Component 1", uiOutput("component1")), tabPanel("Component 2"), tabPanel("Component 3")))

shinyApp(ui = ui, server = server)

In this example, each of the tabs are actualized in the UI as per usual, and then are distributed accordingly in the server section of the code. This stands in contrast to more common amateur methods of shoving all of the code into the UI, where parentheses tend to rack up and complicate things exponentially. We thereby create a main panel that sources data to each of the relevant tabs.

The goal is to create something similar to the image above, making for easy navigation between sorted pockets of data. Each of these tabs can then be shaped to best fit their individual purposes, with sidebars and other features exclusive to themselves. We'll discuss some of these features more extensively in the remainder of this post.

2.) Collapsible Sidebars

The collapsible sidebar is another of the simple, effective organizational tools that can amplify the quality of your Shiny app. It might take additional coding languages to accomplish this, such as CSS and Javascript, but most of it can be completed within the normal Shiny confines. There are multiple ways to build a sidebar, but this is an example of code for an effective version:

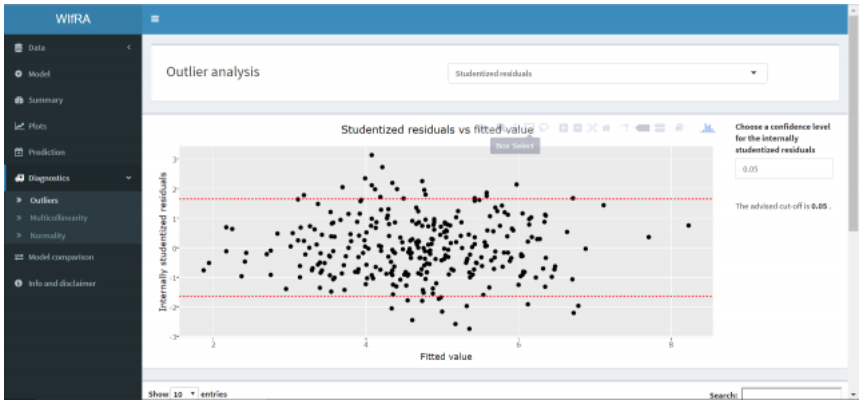
library(shiny) library(shinydashboard)

dashboardPage(dashboardHeader(title="Semi-collapsible Sidebar"),

dashboardSidebar(sidebarMenuOutput("Semi_collapsible_sidebar")),

dashboardBody(tags*head*(*tags*link(rel = "stylesheet", type = "text/css", href = "style.css"))))

The first thing you need to build this feature is a dashboard, which the above code lays out. This code is included in the server section rather than the UI, as we need the functionality of it collapsing at the press of a button. There are additional steps and linked files necessary to make it work effectively, but the eventual goal is to create something similar to this:

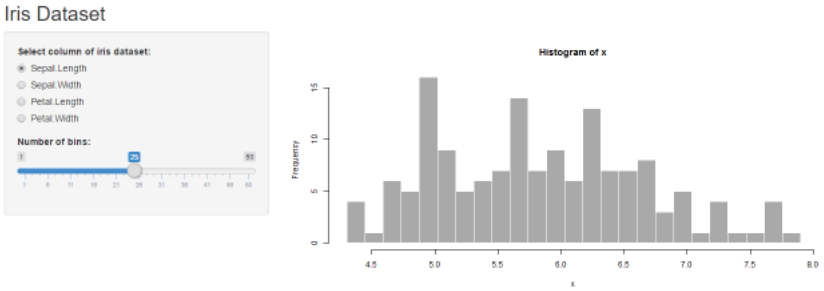


The collapsible sidebar may be even more cleanly presentable than the use of tabs, as the navigation tools can be swept away and out of sight. This makes the user interface even more simple, which can only be a good thing. We'll move on now from these navigational tools, but it's definitely worth putting an effort into customizing one of the options we've discussed so far.

3.) Interactive Data Visualizations

Now we're moving on to ways you can manage the data presented in your Shiny app. Tabs and collapsible sidebars allow users to edit the display of the application in one way, and editable data visualizations are just another. Let's say you want to create a histogram accompanied by

a side panel that influences the dataset. Here's an actualized example of this:



Here's the UI code for this specific example:

library(shiny) shinyUI(fluidPage(titlePanel("Iris Dataset"), sidebarLayout(sidebarPanel(radioButtons("p", "Select column of iris dataset:", list("Sepal.Length"='a', "Sepal.Width"='b', "Petal.Length"='c', "Petal.Width"='d')), sliderInput("bins", "Number of bins:", min = 1, max = 50, value = 30)), mainPanel(plotOutput("distPlot")))))

You can see that the author of this code utilized the action widgets that are included in the standard Shiny packages. While you are probably educated in the variety of widgets that are available and their functionality, but there are ways to use these tools in accordance with each other that take time and experience to fully appreciate. In this case, radio buttons are used in conjunction with a slider to achieve the desired effect. Now let's take a peek at the server code:

library(shiny) shinyServer(function(input, output) { output distPlot <- renderPlot({ if(input p=='a'){ i<-1 }

```
if(input$p=='b'){
  i<-2
}

if(input$p=='c'){
  i<-3
}

if(input$p=='d'){
  i<-4
}

x    <- iris[, i]

bins <- seq(min(x), max(x), length.out = input$bins + 1)
hist(x, breaks = bins, col = 'darkgray', border = 'white')
```

}) })

Each of the variations on the dataset are linked to a simple digit, which determines the bit the user sees at any given time. The server repeatedly checks for changes based on user interaction with the interface, so changes to the application's display are almost instantaneous.

4.) Miscellaneous Bits

For the remainder of this post, I'd like to highlight a few of the interesting and useful packages that you should consider installing in order to better take advantage of Shiny's deep capabilities. The first one is called shinyjqui, a package authored by Yang-Tang. This package fixes one of the major issues users have with the standard Shiny app interface- that is is too static and unflashy in its presentation. There is a new level of customizability here that can't be found in the basic set-up, allowing you to quickly change the layout of the app with a few simple clicks and drags of the mouse.

The second package I'd like to recommend to you is Formattable, authored by Renkun. It adds additional functionality related to sorting datasets into tables, themselves organized in a variety of ways and easily color-codable. You can choose to order the data entries by rounded figures or percentage statistics, and the available styling options range from a selection of colors to various insertable icons. The best Shiny applications balance the wealth of information they present with an interface that is both easily comprehensible and pleasing to the eye. This package will put you on the road to creating something that does exactly that.

References: 1 - https://www.r-bloggers.com/three-r-shiny-tricks-to-make-your-shiny-app-shines-23-semi-collapsible-sidebar/ 2 - https://shiny.rstudio.com/articles/js-widget-functionality.html 3 - https://www.analyticsvidhya.com/blog/2016/10/creating-interactive-data-visualization-using-shiny-app-in-r-with-examples/ 4 - http://gerinberg.com/2017/06/06/shiny-tips-tricks/ 5 - https://stackoverflow.com/questions/27080089/how-to-organize-large-r-shiny-apps 6 - http://enhancedatascience.com/2017/07/10/the-ur-r-shiny-application/ 7 - https://deanattali.com/blog/advanced-shiny-tips/