Post02: A quick guide to Shiny apps

Code

Jade Wang 11/30/2017

Introduction & Background

Shiny enables us to write powerful interactive web applications entirely in R. Using R, we can create a user interface and server and Shiny compiles our code into the HTML, CSS and JavaScript needed to display our app online.

From my own perspective, Shiny is one of the most challenging topics in this course, so I want to make some breif guidelines about how to do Shiny app in a simple way, and I hope it will help! Shiny may be hard to handle at first, but I hope my post could give everyone a more clear understanding of how Shiny works.

Let's nail it!

Preparation

Let's start by loading some pakages first.

In order to run Shiny and follow the code on this post you should make sure you have RStudio software and the shiny R package.

Hide

```
library(shiny)
```

The Most Basic Shiny App

In its simplest form, a Shiny application requires a server function to do the calculations and a user interface.

In the example, we will put only some text in UI, so the app may look strange, but it is a real Shiny app.

The code is as below:

Hide

```
library(shiny)

server <- function(input, output, session) { } # the server

ui <- basicPage(
'Here is a Shiny app at simplest.'
 ) # the user interface

shinyApp(ui = ui, server = server) # it runs the app
```

Yes, I know the simplest Shiny app looks ugly and strange, but I promise things will get much better as we add more elements into our Shiny app.

Set up your user interface (UI)

1. Layout your unser interface

When we run the function *shinyApp()* our R code is essentially compiled to web-friendly languages HTML, JavaScript and CSS. We can take advantage of Shiny page template functions to layout our app.

In this example, we will focus on the simplest function, by creating sidebarPanel and mainPanel.

The code is as below:

Hide

```
#Example app: basic user interface
library(shiny)

ui <- fluidPage(
  titlePanel("This is a Shiny app with the simplest layout"),
  sidebarLayout(
    sidebarPanel(
      'Sidebar'
    ),
    mainPanel(
      'Main panel'
    )
  )
)

server <- function(input, output, session) { } # the server

shinyApp(ui, server)
```

2. Adding elements with tags$xxx

It's easy to add HTML elements to our Shiny app using HTML tags.

There are more than 100 HTML elements available.

In the example, we add HTML tags using *tags$blockquote*, and common elements like *h1* and *h3*

The code is as below:

Hide

```
#Example app: using HTML tags in Shiny
library(shiny)

ui <- fluidPage(
  h1('Here is H1 without tags$'),
  tags$blockquote('The block quote here requires tags$'),
  h3('Here is H3 without tags$'),
  code('Here is the data: data.frame(a=1:20, b=1:20)')
)

server <- function(input, output, session) { } # the server

shinyApp(ui, server)
```

3. Layout Shiny app using Bootstrap's grid system manually

We can use Bootsrep's system directly by specifying rows and co;umns.

In the example, we'll use Boostrap elements like rows and columns to manually layout the app. We'll have 2 rows: one is offset by one column and the other has a button (1 column width) and text (5 column width).

The code is as below:

Hide

```
# Example app: Manual layout using Bootstrap's grid system

ui <- fluidPage(

  fluidRow(
    column(6, offset = 1,
           h1("Here is the title")
       )
  ),
  fluidRow(
    column(1,
           actionButton('Button','click')
   ),
   column(5,
          p("Row 2, Column 2 (buttion is col 1)")
          )
  )

)

server <- function(input, output, session) {

}

shinyApp(ui, server)
```

4. Style our user interface (UI) with exsiting themes

We've discussed some approaches to style our Shiny app as above.

Still, Bootstrap offers a number of pre-created themes that allow us to style our app with limited coding.

But first, we need to load the shinythemes package from RStudio.

The code is as below:

Hide

```
# Example app: using a pre-created Bootstrap theme
library(shinythemes)

server <- function(input, output, session) {

}

ui <- fluidPage(theme=shinytheme("cosmo"),

  titlePanel("App use an existing theme"),

  sidebarLayout(

    sidebarPanel(
      h3("Note the button has a color of black!!!"),
      actionButton("button", "This is a button")
    ),

    mainPanel(
      tabsetPanel(
        tabPanel("Plot1"),
        tabPanel("Plot2"),
        tabPanel("Plot3")
      )
    )
  )
)

shinyApp(ui = ui, server = server)
```

6. Add widgets to user interface (sliders, text boxes, etc)

At this point we have scaffolded and added styles to our UI, but we can also add widgets to make UI more interactive, like pull-down menus.

Note that all widgets must have a unique input ID.

In the example, we will add a slider and a text input to our UI.

The code is as below:

Hide

```
# Example app: allowing inputs

server <- function(input, output, session) {

}

ui <- basicPage(

  h3("Here are some simple widgets "),
  # a slider
  sliderInput(inputId = "myslider", label = "Here is the label ", min = 0,
              max = 100, value = c(50, 60)),
  # a text input box
  textInput(
    inputId = "mytext", label = "Text Input",
    value = "Enter some text here..."
    )
)


shinyApp(ui = ui, server = server)
```

Now, let's link UI with the server to display our outputs

In order to return values to the user we need a strategy to:

(1) grab needed values from the UI in the server

(2) process as necessary

(3) return the result to the UI

1. To print text in UI

In the example, we'll use functions of *renderText* and *textOutput* to demonstrate how to print text.

The code as below:

Hide

```
# Example app: linking UI and the server

server <- function(input, output, session) {

  # input$mytext comes from the UI. my_output_text
  # sent back
  output$my_output_text <- renderText({
    return(input$mytext)
    })
}

ui <- basicPage(
    h3("The value in the text box gets printed below with the *textOutput* function."),
    textInput("mytext", "Input goes here"),
    # my_output_text comes from the server

    "Output value is:",
    textOutput("my_output_text")
)

shinyApp(ui = ui, server = server)
```

2. Adding a plot to UI

Similiar as the previous example, we'll use functions of *renderPlot* and *plotOutput* to add a plot to UI.

The code as below:

Hide

```
server <- function(input, output, session) {

  output$my_output_text <- renderText({
    init <- "Input title is: "
    return(paste0(init, input$mytext))
  })

  # send plot to the ui as my_output_plot
  output$my_output_plot <- renderPlot({
    plot(1:7, 1:7, pch=16, col=1:7, cex=1:7, main = input$mytext)
  })


}

ui <- basicPage(

  h3("Now we have both text and plot output"),
  textInput("mytext", "Main title goes here"),

  # my_output_text comes from the server
  textOutput("my_output_text"),
  # my_output_plot comes from the server
  plotOutput("my_output_plot")

)

shinyApp(ui = ui, server = server)
```

3. Dynamic UI with functions of renderUI* and OutputUI *

By using *renderUI* and *OutputUI*, we can return a list of UI elements rather than a single object.

In the example, we can add new object to the selection list by typing the name in the textbox and then clicking the button.

The code as below:

Hide

```
# Example app: dynamic UI

server <- function(input, output, session) {

  # return a list of UI elements
  output$my_output_UI <- renderUI({

    list(
      h4(style = "color:blue;", "My selection list"),
      selectInput(inputId = "myselect", label="", choices = selections)
    )
  })

  # initial selections
  selections <- c("New York", "Philadelphia")

  # use observe event to notice when the user clicks the button
  # update the selection list. Note the double assignment <<-
  observeEvent(input$mybutton,{
    selections <<- c(input$mytext, selections)
    updateSelectInput(session, "myselect", choices = selections, selected = selections[1])
  })

}

ui <- basicPage(

    h3("Using renderUI and uiOutput"),
    uiOutput("my_output_UI"),
    textInput("mytext", ""),
    actionButton("mybutton", "Click to add to selections")

)

shinyApp(ui = ui, server = server)
```

Finally, let’s put some real data into our Shiny app

Hide

```
library(shiny)

dat <- read.csv('/Users/Jade/stat133-hws-fall17/post02/data/scores.csv')

ui <- fluidPage(
  titlePanel("Example App"),

  h4("Observations"),
  tableOutput("view"),


  selectInput("select", label = h3("Select box"),
              choices = list("HW1" = "HW1", "HW2" = "HW2",
                             "HW3" = "HW3", "HW4" = "HW4",
                             "HW5" = "HW5", "HW6" = "HW6",
                             "HW7" = "HW7", "HW8" = "HW8",
                             "HW9" = "HW9"),
              selected = "HW1"),
  sliderInput("width",
              "Bind width",
              min = 0,
              max = 120,
              value = 50),

  selectInput("variableX", label = h3("X-axis variable"),
              choices = list("Test1" = "Test1", "Test2" = "Test2"),
              selected = "Test1"),

  selectInput("variableY", label = h3("Y-axis variable"),
              choices = list("Overall" = "Overall"),
              selected = "Overall"),

  h3("Summary"),
  verbatimTextOutput("summary"),


    # Show a plot of the generated distribution
    mainPanel(
      tabsetPanel(
        tabPanel("Barplot",  plotOutput("bar")
      )
    )
  )
 )



server <- function(input, output, session) {

  output$bar <- renderPlot({
    # generate bins based on input$bins from ui.R

    # draw the histogram with the specified number of bins
    grades = factor(dat$Grade, levels = c('A+', 'A', 'A-', 'B+', 'B', 'B-', 'C+', 'C', 'C-', 'D', 'F'))
    barplot(table(grades), col = 1:11, border = 'black')
  })

}

shinyApp(ui, server)
```

Conclusions

In order to make a dynamic Shiny app, we need to first create user interface (UI) and server, and then link these two parts together to allow inputs and outputs.

From the above examples, we learn that, despite of the most basic structure, we can also style our UI layout with either HTML tags or pre-created Bootstrap themes.

Also, in order to link UI with server, we can use functions of *renderUI* and 'OutputUI' to create a dynamic Shiny app.

I hope everyone has a better idea about Shiny app now! Thank you:)

References

Shiny tutorial resources from GitHub

https://shiny.rstudio.com/tutorial/

https://github.com/ucb-stat133/stat133-fall-2017/blob/master/cheat-sheets/shiny-cheatsheet.pdf

https://github.com/ucb-stat133/stat133-fall-2017/blob/master/slides/16-shiny-tutorial.pdf

https://docs.google.com/presentation/d/1_XhkkhXA26izACJyoc5JYrWiF_f96a6tD9ZSl5Tm8Os/pub?start=false&loop=false&delayms=3000&slide=id.gd30d76405_0_29

Shiny UI layout guide

http://shiny.rstudio.com/articles/layout-guide.html

HTML tags

http://shiny.rstudio.com/articles/tag-glossary.html

Bootstrap Basics

https://getbootstrap.com/docs/3.3/examples/grid/