STAT 29000 Project 7

Topics: R, Shiny

Motivation: Sometimes it's useful to be able to more deeply interact with your data, and interface with it in a point-and-click fashion. One example would be if you need to present an analysis to a boss, but the ability to tweak values or information real-time is important to be able to do and show. In this situation, using a package called Shiny, could be very useful!

Context: We've dove into the python world and have had the opportunity to solve some data-driven problems using this new tool. Now we are going to switch gears and learn about buildling Shiny apps in R.

Scope: Using the `shiny` package to build simple web apps.

Rstudio provides *excellent* examples that are instantly available for you to play around with:

```r
library(shiny)
runExample("01_hello")      # a histogram
runExample("02_text")       # tables and data frames
runExample("03_reactivity") # a reactive expression
runExample("04_mpg")        # global variables
runExample("05_sliders")    # slider bars
runExample("06_tabsets")    # tabbed panels
runExample("07_widgets")    # help text and submit buttons
runExample("08_html")       # Shiny app built from HTML
runExample("09_upload")     # file upload wizard
runExample("10_download")   # file download wizard
runExample("11_timer")      # an automated timer
```

By running one of the examples you are immediately presented with an app for you to test out, as well as the associated (copy-and-pastable) code that you can use to run the app yourself.

Use the template found here or on scholar: `/class/datamine/data/spring2020/stat29000project07template.R` to submit your solutions.

After each problem, we've provided you with a list of keywords. These keywords could be package names, functions, or important terms that will point you in the right direction when trying to solve the problem, and give you accurate terminology that you can further look into online. You are not required to utilize all the given keywords. You will receive full points as long as your code gives the correct result.

Question 1: introduction to shiny

First and foremost, there is a lot to learn about `shiny` – far more than we will be able to cover in a single project. In addition, any examples that are included as a part of this project are simply there to fill any gaps that are required to be filled in order to complete this project. The written tutorial does a much better job of introducing `shiny` than an examples file would do. For each problem or sub-problem, I will provide you with the lesson(s) that you will be required to read in order to be able to finish the problem or sub-problem. In addition, I will provide you with a list of widgets and functions needed to complete each problem or sub-problem.

For this project you will submit a single app.R file that will contain multiple apps, where each app has 3 main components: the `ui` object that we will call `ui_1a` if the app was for problem (1a), the `server` function that we will call `server_1a` if the app was for problem (1a), and a call to the `shinyApp` function: `shinyApp(ui = ui_1a, server = server_1a)`. If the solution is for (2b) we would have: `ui_2b`, `server_2b`, and `shinyApp(ui=ui_2b, server=server_2b)`. The terminologies will be clearer after (1a).

1a. *(2 pts)* Read lesson 1. Run the first example, "01_hello". Copy and paste the source of this example app to your `app.R` file. Feel free to play around and modify the app to see how it changes based on the modifications you make. Once you are ready, modify the app in the following ways:

1. Rename the title from "Hello Shiny!" to "Firstname Lastname Solution for 1a!", where you replace Firstname and Lastname with your first and last name.
2. Change the slider so the minimum number of bins possible is 25.
3. Change the slider so the default value for the number of bins is 25.
4. Change the color of the histogram to Old Gold (using the hex triplet code).
5. Change `plotOutput(outputId = "distPlot")` in your `ui` object to `plotOutput(outputId = "oldFaithfulHistogram")`. Note that this will break your app and make the plot no longer appear. Make modification to your `server` function to enable your shiny app to function once more.

Item(s) to submit:
- A `ui_1a` `fluidPage`
- A `server_1a` function
- A line with `shinyApp(ui = ui_1a, server = server_1a)` as shown in the template.

1b. *(2 pts)* Read lesson 2. Copy and paste your solution to (1a) at the bottom of your app.R file and make the following modifications:

1. Rename the title from "Solution 1a!" to "Solution 1b!", wrap the text in the `h1` function, and center it.
2. Move the `sidebarPanel` to the right. **Hint:** *Try modifying `sidebarLayout` to accomplish this.*
3. Add an `h2` heading to the top center of the sidebarPanel called "Settings".
4. Add the following to the `mainPanel`, above your plot:
   - An `h1` heading to the top left called "Directions".
   - A `p` paragraph that explains how to use your app, include at least 1 **bolded** word, 1 *italicized* word, and 1 link.
   - Add a Purdue themed image to your `mainPanel`.
5. Modify your app to use `ggplot2` to generate the histogram instead of `hist`. Make the outline black, and fill color old gold.

Keywords: *p, h1, h2, position, em, b, a, img, align, geom_histogram*

Item(s) to submit:
- A `ui_1b` `fluidPage`
- A `server_1b` function
- A line with `shinyApp(ui = ui_1b, server = server_1b)` as shown in the template.

Question 2:

2a. *(1 pt)* Read lesson 3. Read the "Grid Layout" section here. Using `fluidPage`, `fluidRow`, and `column` allows for a much more flexible system for designing your app's layout. Copy, paste, and run the example here or on scholar: `/class/datamine/data/spring2020/stat29000project07example01.R`.

Use the `fluidPage`, `fluidRow`, and `column` functions, and modify the example to replicate the layout found here. Take note that the `offset` argument to the `column` function is your friend here.

Keywords: *fluidPage, fluidRow, column, column argument offset*

Item(s) to submit:
- A `ui_2a` `fluidPage`.
- A `server_2a` function.
- A line with `shinyApp(ui = ui_2a, server = server_2a)` as shown in the template.

2b. *(3 pts)* Read lesson 4. Copy and paste your solution to (1b) and maket he following modifications:

1. (optional) Use what you learned in (2a) to rearrange the layout of your app to make its appearance more pleasing to you.
2. Use the `selectInput` function to add a Select box widget to your app, directly below your `sliderInput`. Give it a label named "Distribution:", and the choices: "Normal", "Uniform", and "Exponential". You can set the default to be whichever you'd like.
3. Change the slider so the maximum number of possible bins is 100.
4. Modify your `server_2b` function to randomly sample 100 values from the selected distribution using `rnorm`, `runif`, `rexp`, where you only need to specify `n`, the number of samples to draw. Base which distribution you draw samples from based on the user's selection. Modify your histogram to plot this data instead of the previous data.
5. Change the title from "Solution 1b!" to "Solution 2b!".

For step (4), you need to make sure that when you are accessing `input$some_variable` in the `server` function, that it is within a `render*` function. For example:

```r
library("shiny")
ui <- fluidPage(
    titlePanel(h1("Fake app", align="center")),
    selectInput("some_variable", label = "Some var",
                choices = list("OptA" = 1, "OptB" = 2, "OptC" = 3), selected=1),
    textOutput("selected_variable")
)

server <- function(input, output) {
    # this will trigger an error because input$some_variable
    # must be called within a function, reactive expression or observer
    my_value <- input$some_variable
    output$selected_variable <- renderText({
        paste0("This won't work!: ", my_value)
    })
}
shinyApp(ui=ui, server=server)
```

```r
library("shiny")
ui <- fluidPage(
    titlePanel(h1("Fake app", align="center")),
    selectInput("some_variable", label = "Some var",
                choices = list("OptA" = 1, "OptB" = 2, "OptC" = 3), selected=1),
    textOutput("selected_variable")
)

server <- function(input, output) {

    # our app works fine with input$some_variable in renderText()
    output$selected_variable <- renderText({
        paste0("This will work!: ", input$some_variable)
    })
}
shinyApp(ui=ui, server=server)
```

As a side note, all `input` variables should be accessed within reactive expressions or reactive observers (not needed in this project) – just FYI!

Keywords: *rnorm, runif, rexp, if, if else, selectInput*

Item(s) to submit:

- A `ui_2b` `fluidPage`.
- A `server_2b` function.
- A line with `shinyApp(ui = ui_2b, server = server_2b)` as shown in the template.

2c. *(2 pts)* Play around with `shiny` and create your own app to do anything you'd like. The only requirement is you need to have at a minimum a single widget, and output that is automatically ("reactively") changed, based on the widget. Get creative. The more you learn about `shiny` now, the easier the next couple of projects will be. Don't feel restricted to only use methods you've learned about in this project.

Item(s) to submit:

- A `ui_2c` `fluidPage`.
- A `server_2c` function.
- A line with `shinyApp(ui = ui_2c, server = server_2c)` as shown in the template.
- Ensure that your solution contains at least 1 widget.
- Ensure that your solution contains at least 1 output that is automatically ("reactively") changed based on the widget or widget(s).

Project Submission:

Submit your solutions for the project at this URL: https://classroom.github.com/a/6EHQWLdk using the instructions found in the GitHub Classroom instructions folder on Blackboard.

Important note: For this project you will submit a single app.R file that will contain multiple apps, where each app has 3 main components: the `ui` object that we will call `ui_1a` if the app was for problem (1a), the `server` function that we will call `server_1a` if the app was for problem (1a), and a call to the `shinyApp` function: `shinyApp(ui = ui_1a, server = server_1a)`. If the solution is for (2b) we would have: `ui_2b`, `server_2b`, and `shinyApp(ui=ui_2b, server=server_2b)`.