

Post01-Kayla-Simons

Kayla Simons
October 28, 2017

Touching Numbers: Using the Shiny Application to Interact with Data

Introduction

When analyzing and presenting data sets, the mode in which the data is presented is crucial. Presenting data through graphs, charts, and other figures is the best way for simple data analysis. However, through using the application Shiny in R, one is able to fully interact with the data to obtain the best understanding. Shiny allows the viewer to see different aspects of data through features like sliding bars, numeric inputs, text inputs, checkboxes, and several other ways. The code used to create these features only varies slightly, and is very simple to write. In this post, I intend to show the multiple ways Shiny allows interaction with data, the benefits of using Shiny, and explain how to write such code.

Writing Code For Shiny

Writing code for Shiny might seem complex at first, but it is actually very simple. The code is written in RStudio. If one is making an app with a single file, it's necessary to open app.R. If one is creating an app with multiple files, then open ui.R and server.R in RStudio. In my explanation, I will be using ui.R and server.R to show how to write code with Shiny. Then, load Shiny with `library(shiny)` into ui.R and server.R.

Note: `eval = FALSE` is written on all the code so that it does not actually run. This is because this code should be written in a ui.R or server.R file, not an .Rmd file. Further, there is no actual data behind the plot output, and thus would error regardless. Finally, this is an extremely simplified version of what code in ui.R or server.R could look like, and could actually be much more complex.

UI

In ui.R you need to define the user interface with the function `pageWithSidebar`, and pass those results through the Shiny UI function. Then you pass through three functions; one creates the title. It should look something like this:

```
shinyUI(pageWithSidebar(  
  headerPanel("Kayla's Post on Shiny"),  
  sidebarPanel(),  
  mainPanel()))
```

The next step is to include the different inputs that the viewer can choose, and the method in which the viewer can choose the data. In this case, the viewer is able to select the data from a drop down the lists. The options the viewer can choose are Option 1, Option 2, or Option 3.

```
shinyUI(pageWithSidebar(  
  headerPanel("Kayla's Post on Shiny"),  
  sidebarPanel(selectInput("options", "Options:",  
                           list("Option 1", "Option 2", "Option 3"))),  
  mainPanel()))
```

Underneath the sidebarPanel function, one could also add an extra specifying factor, such as a checkbox to further organize the data. However, for the sake of simplicity, I am not going to show that.

The next step is to code the different outputs for each option. The function `h3()`, binds certain outputs to data sets. Assume we have a data set called "Graph". We will also create a caption for our data set, which will be "Output".

```
shinyUI(pageWithSidebar(  
  headerPanel("Kayla's Post on Shiny"),  
  sidebarPanel(selectInput("options", "Options:",  
                           list("Option 1", "Option 2", "Option 3")),  
  mainPanel(h3(textOutput("Output")),  
            plotOutput("Graph")))))
```

Server

Server is used to define inputs and attach them to different outputs. We start with a very basic form of skeleton code:

```
shinyServer(function(input, output) {  
})
```

This code, paired with the most basic form of ui.R code, will show a simple screen with the title "Kayla's Post on Shiny" and a simple, blank entry box:

Kayla's Post on Shiny

The next thing we need to do in server.R is define the server-side inputs and outputs. This should be used to assign certain inputs to their outputs. The inputs will be assigned to the outputs after undergoing the desired computations. In this code, assume my data set is called "kaylas_data" and is factored. Also, if this code were to run, it would create a boxplot.

```
shinyServer(function(input, output) {  
  formulaText <- reactive({  
    paste("my data ~", input$variable)  
  })  
  output$caption <- renderText({  
    formulaText()  
  })  
  output$kaylaPlot <- renderPlot({  
    boxplot(as.formula(formulaText()),  
            data = kaylas_data,  
            outline = input$outliers)  
  })  
})
```

Note: `renderText` and `renderPlot` are used to generate output and are what enable the viewer to interact with the data. They change when their input changes.

This information about `ui.R` and `server.R` is very simplified and is mostly just to show the basics of writing code in Shiny. Also, it is important to understand that since there is no actual data behind the code, I did not run it. I will show examples of Shiny applications later on in this post. Further, code can become much more complex, and with that, become much more interactive. I will now move on to discuss the different types of Shiny widgets that are available. A more comprehensive overview to making Shiny apps can be found from this video: https://www.youtube.com/watch?v=sJl0EE_RE4o

Different Widgets in Shiny

Now that we understand how the basics of writing code for Shiny, we will move on to the many different options available when choosing how to present data. According to "Shiny from R Studio" <https://shiny.rstudio.com/gallery/widget-gallery.html>, there are twelve different "Shiny Widgets" one could use on data. Each of these different widgets have slight variations in code, but are ultimately very similar.

Action button

The action button is a single button that starts at a value of zero. One can press the button to increase the value of the integer, which can often evoke action from data sets. Here is the code for both `ui.R` and `server.R`:

`ui.R`

```
fluidPage(
  actionButton("action", label = "Action"),
  hr(),
  fluidRow(column(2, verbatimTextOutput("value")))
)
```

`server.R`

```
function(input, output) {
  output$value <- renderPrint({ input$action })
}
```

Action button

Action

Current Value:

```
[1] 0
attr(,"class")
[1] "integer"
```

Single Checkbox

The single checkbox creates one checkbox that essentially acts as a TRUE or FALSE mechanism. If the checkbox is clicked, one is deciding to either include or exclude a certain piece of information from the data being viewed. The code for

ui.R shows a single checkbox that can be filled or unfilled.

ui.R

```
fluidPage(  
  checkboxInput("checkbox", label = "Choice A", value = TRUE),  
  hr(),  
  fluidRow(column(3, verbatimTextOutput("value")))  
)
```

server.R

```
function(input, output) {  
  output$value <- renderPrint({ input$checkbox })  
}
```

Single checkbox

☒ Choice A

Current Value:

[1] TRUE

Checkbox Group

Checkbox group is similar to single checkbox in that you select whether or not you want to see certain data. However, the checkbox group enables the user to choose how much data is shown. This is done by enabling the user to select the number of checkboxes desired, and sending this information to the server. ui.R shows three different checkboxes one could choose between.

ui.R

```
fluidPage(  
  checkboxGroupInput("checkGroup", label = h3("Checkbox group"),  
    choices = list("Choice 1" = 1, "Choice 2" = 2, "Choice 3" = 3),  
    selected = 1),  
  hr(),  
  fluidRow(column(3, verbatimTextOutput("value")))  
)
```

server.R

```
function(input, output) {  
  output$value <- renderPrint({ input$checkGroup })  
}
```

Checkbox group

- ☒ Choice 1
- ☐ Choice 2
- ☒ Choice 3

Current Values:

```
[1] "1" "3"
```

Date Input

Date input enables the user to select a date from a calendar. This allows the viewer to access date from an exact day.

ui.R

```
fluidPage(  
  dateInput("date", label = h3("Date input"), value = "2014-01-01"),  
  hr(),  
  fluidRow(column(3, verbatimTextOutput("value")))  
)
```

server.R

```
function(input, output) {  
  output$value <- renderPrint({ input$date })  
}
```

Date input

```
2017-10-31
```

Current Value:

```
[1] "2017-10-31"
```

Date Range

Date range is very similar to date input but it allows the viewer to select two dates from a calendar. This makes it so the viewer can see the data across a whatever interim desired.

ui.R

```
fluidPage(  
  dateRangeInput("dates", label = h3("Date range")),  
  hr(),  
  fluidRow(column(4, verbatimTextOutput("value")))  
)
```

server.R

```
function(input, output) {  
  output$value <- renderPrint({ input$dates })  
}
```

Date range

2017-09-01	to	2017-12-31
------------	----	------------

Current Values:

```
[1] "2017-09-01" "2017-12-31"
```

File Input

File input takes in a file and essentially summarizes the contents of the file. For example, the widget will return the file's name, size (in bytes), type (.pdf, .docx, etc.), datapath (aka the local pathway for the file). In this example, I have uploaded my resume to be summarized by the file.

ui.R

```
fluidPage(  
  fileInput("file", label = h3("File input")),  
  hr(),  
  fluidRow(column(4, verbatimTextOutput("value")))  
)
```

server.R

```
function(input, output) {  
  output$value <- renderPrint({  
    str(input$file)  
  })  
}
```

File input

Browse...

Resume_Kayla Simons.pdf

Upload complete

Current Value:

	name	size	type
1	Resume_Kayla Simons.pdf	198110	application/pdf
			datapath
1	/tmp/RtmpycDKqV/409f04809eb60dfba125283c/0.pdf		

Numeric Input

The numeric inputs allows one to type or scroll through a number. The number is then passed through the server as an integer vector. This could be beneficial to the user in a variety of ways, such as placing a value of how much data the viewer wants to see. In addition, numeric input could enable the user to see how the data reacts when based on different numbers.

ui.R

```
fluidPage(  
  numericInput("num", label = h3("Numeric input"), value = 1),  
  hr(),  
  fluidRow(column(3, verbatimTextOutput("value")))  
)
```

server.R

```
function(input, output) {  
  output$value <- renderPrint({ input$num })  
}
```

Numeric input

15



Current Value:

[1] 15

Radio Buttons

Radio buttons allow the user to select one of however many options. Unlike checkbox groups, you cannot select multiple data inputs.

ui.R

```
fluidPage(  
  radioButtons("radio", label = h3("Radio buttons"),  
    choices = list("Choice 1" = 1, "Choice 2" = 2, "Choice 3" = 3),  
    selected = 1),  
  hr(),  
  fluidRow(column(3, verbatimTextOutput("value")))  
)
```

server.R

```
function(input, output) {  
  output$value <- renderPrint({ input$radio })  
}
```

```
## function(input, output) {  
##   output$value <- renderPrint({ input$radio })  
## }
```

Radio buttons

- ☐ Choice 1
- ☐ Choice 2
- ☒ Choice 3

Current Values:

```
[1] "3"
```

Select Box Select box is a drop-down list

where the user can select one option. This option has the same functioning as radio buttons, but presented in a different way. Select box would be a better option that radio buttons if one was had several options to choose between.

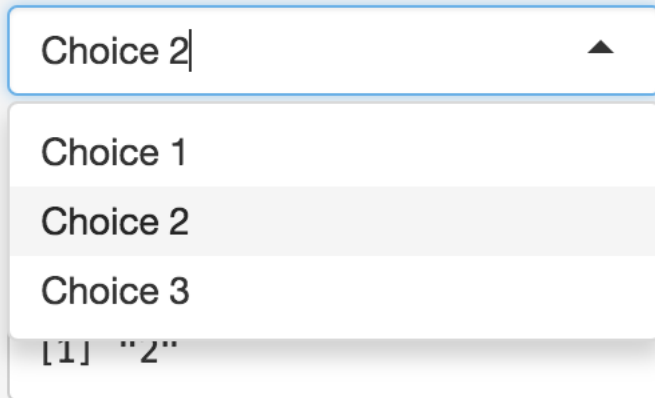
ui.R

```
fluidPage(  
  selectInput("select", label = h3("Select box"),  
    choices = list("Choice 1" = 1, "Choice 2" = 2, "Choice 3" = 3),  
    selected = 1),  
  hr(),  
  fluidRow(column(3, verbatimTextOutput("value")))  
)
```

server.R


```
function(input, output) {
  output$value <- renderPrint({ input$select })
}
```

Select box



Slider Bar and Slider Range

The slider bar is a bar that starts at zero and can move up to integer desired. The slider range can start at any integer and end at any integer. In both, the viewer has the ability to change the range of inputs to any size desire. I have only included code for the slider range because the function used is the same for both.

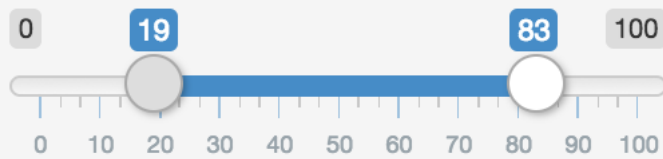
ui.R

```
fluidPage(
  fluidRow(
    column(4,
      sliderInput("slider1", label = h3("Slider"), min = 0,
        max = 100, value = 50)),
    column(4,
      sliderInput("slider2", label = h3("Slider Range"), min = 0,
        max = 100, value = c(40, 60)))),
  hr(),
  fluidRow(
    column(4, verbatimTextOutput("value")),
    column(4, verbatimTextOutput("range"))))
```

server.R

```
function(input, output) {
  output$value <- renderPrint({ input$slider1 })
  output$range <- renderPrint({ input$slider2 })
}
```

Slider range



Current Values:

[1] 19 83

Text Input

Lastly, there is text input. Text input allows the viewer to type anything into a text bar and it will be passed through the server as a character vector. In this example, I typed "hello" into the text input.

ui.R

```
fluidPage(  
  textInput("text", label = h3("Text input"), value = "Enter text..."),  
  hr(),  
  fluidRow(column(3, verbatimTextOutput("value")))  
)
```

server.R

```
function(input, output) {  
  output$value <- renderPrint({ input$text })  
}
```

Text input

hello

Current Value:

[1] "hello"

Each widget is very beneficial for different functions. Deciding the type of widget to use heavily depends on the data one is trying to display.

Example of Use

As already mentioned, Shiny contains many different ways of interacting with data. Choosing a method like sliding bars, checkboxes, or date input heavily depends on the type of data set, and the way in which one desires for it to be portrayed. For example, an application (that is surprisingly relevant to this course) made by Jiashen Liu, shows the data of shots for over 300 NBA players. The data is very detailed and looks at multiple different aspects of players' shots in the NBA. The app effectively uses Shiny to allow the viewer to see certain aspects within the data. Amongst opening the app, there are three different data types the viewer can choose from: Player Data, Player Comparison, and Shot Type Analysis for two seasons. There are also two other tabs that allow further research of the data, and enable the viewer to download the data into a .csv file. When looking at the default tab, Player Data, there are six uses of Shiny. Liu uses three select boxes, two radio buttons, and one checkbox group. These enable the viewer to see a very specific set of data, including which player, season, quarter, type of graph, and whether or not to display only shots made. Rather than a simple graph, these features make the data visualization much more aesthetically appealing, understandable, interesting. Here is a link to Liu's app: <https://www.showmeshiny.com/nba-shot-plot/>

Other Types of Data Visualization

Another type of data visualization in R is ggplot. ggplot can create a variety of different ways to represent data. These include scatter plots, bar graphs, histograms, etc. This tool is extremely useful in creating graphical representations of data, however, unlike graphs made with Shiny applications, ggplots are not interactive.

Here is an example of a ggplot graph. I used data from 2017 NBA player statistics to show the contrast between Jiashen Liu's Shiny Application, and a simple ggplot graph. My graph shows the correlation between shots made and shots attempted for each player in the 2017 NBA season. I chose to graph these points because they were as close to Liu's graph as I could get.

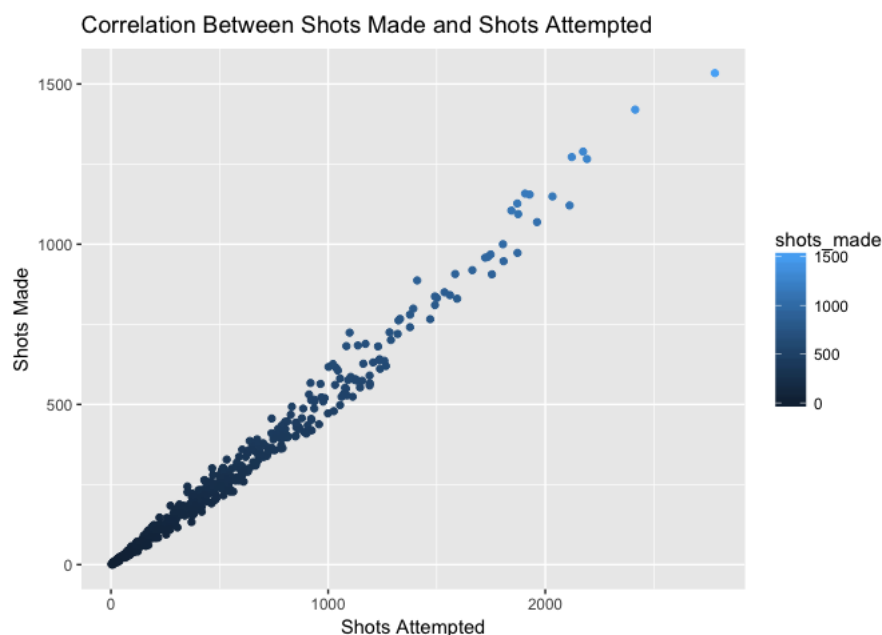
```
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##   filter, lag

## The following objects are masked from 'package:base':
##
##   intersect, setdiff, setequal, union
```

```
nba_dat <- read.csv("nba2017-stats.csv")
nba_dat <- nba_dat %>% group_by(player) %>% mutate(atts_sum = sum(points1_atts, points2_atts, points3_atts))
ggplot(nba_dat, aes(atts_sum, shots_made, color = shots_made)) + geom_point() + ylab("Shots Made") + xlab("Shots Attempted")
```



The results of the graph are seemingly obvious and do not depict nearly the same complexity as Liu's Shiny App. Further, each data point represents a

player, but the points are so close together that it is impossible to distinguish the points. Even with labels and a more zoomed-in axis, it would be nearly impossible to separate each point. This graph thus allows the conclusion that Shiny apps provide the viewer with much more information, organization, and pleasurable viewing.

Conclusion

There are several important ideas to take away from this post. The first, and most important, concept that I hope I portrayed is how useful and beneficial the Shiny Application is in displaying and presenting data. The application allows for maximum organization and enables the viewer to interact with the data in a way that is understandable and enjoyable. Further, there are a multitude of different widgets one could use when working with Shiny. These widgets can be tailored to fit best with data set, and chosen specifically for a each data set. Shiny is also a stronger representation of data than just using ggplot. This is because ggplot is not interactive, so it is much more difficult to see the multiple layers of information, and data points can get swamped together.

Finally, there are several important factors to keep in my when reviewing this post. Mainly, it is important to note that I was not able to provide any data-backed examples of Shiny because the application is written outside of R Markdown. Further, Shiny apps run on web browsers instead of inside RStudio, and thus I could not provide any interactive applications in my post. To learn even more about Shiny, I would suggest exploring <https://shiny.rstudio.com/>. Thanks for reading!

References: <https://shiny.rstudio.com/gallery/widget-gallery.html>

https://www.youtube.com/watch?v=sJI0EE_RE4o

<https://jiashenliu.shinyapps.io/NBAShotChart/>

<http://rstudio.github.io/shiny/tutorial/#ui-and-server>

<https://shiny.rstudio.com/gallery/>

<http://zevross.com/blog/2016/04/19/r-powered-web-applications-with-shiny-a-tutorial-and-cheat-sheet-with-40-example-apps/>

<https://shiny.rstudio.com/articles/plot-interaction.html>

<http://rstudio.github.io/shiny/tutorial/#inputs-and-outputs>