Intro to Shiny

PUBPOL 599

The first step is to make sure that we have the packages we need loaded. For this exercise we'll use shiny

```
library(shiny)
```

Markdown dashboards

There are (at least) two ways to use shiny - in a markdown document and as a seperate app. We'll talk about the later later, but to use it in a Markdown document we need the output to be html and to add "runtime: shiny" to the top of the document. That will tell the document that there are interactive features included.

Inputs and Outputs

To use shiny in a markdown document, you must insert both inputs and outputs. Inputs provide users the options you want for their to interact with, and the outputs interpret those choices and create what is shown.

Inputs are input functions

Outputs are render functions

Inputs

Inputs are what allow users to interact with a Shiny app. Shiny provides many functions to support many kinds of actions you may want, which can be added to the app and paired with data to display. All of this is a menu you can order off of in creating your app.

Input Functions	**Widget**
actionButton	Action Button
checkboxGroupInput	A group of check boxes
checkboxInput	A single check box

dateInput	A calendar to aid date selection
dateRangeInput	A pair of calendars for selecting a date range
fileInput	A file upload control wizard
helpText	Help text that can be added to an input form
numericInput	A field to enter numbers
radioButtons	A set of radio buttons
selectInput	A box with choices to select from
sliderInput	A slider bar
submitButton	A submit button
textInput	A field to enter text

Render Functions

The render functions take R code and “render” it as HTML objects that can be used in web browsers in order to display your dashboard. What they do is translate what you write from R to HTML for you.

There are several different render functions, each named based on what it creates. Because they create different things, they’ll expect different things as inputs (from above)

function	**expects**	**creates**
renderDataTable	any table-like object	DataTables.js table
renderImage	list of image attributes HTML	image
renderPlot	plot	plot
renderPrint	any printed output	text
renderTable	any table-like object	plain table
renderText	character string	text
renderUI	Shiny tag object or HTML	UI element (HTML)

Anatomy of a Shiny object

Let's use data from the dataset on the aiting time between eruptions and the duration of the eruption for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA.

The goal is to create a slider that changes the number of bins that a histogram displays. Let's look under the hood for the sliderInput command.

Description

Constructs a slider widget to select a numeric value from a range.

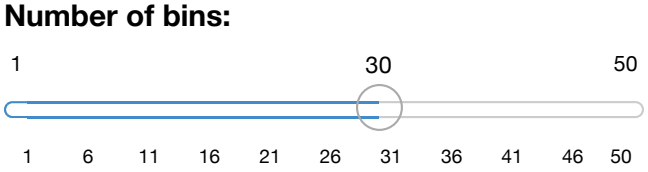
Usage

sliderInput(inputId, label, min, max, value, step = NULL, round = FALSE, format = NULL, locale = NULL, ticks = TRUE, animate = FALSE, width = NULL, sep = ",", pre = NULL, post = NULL, timeFormat = NULL, timezone = NULL, dragRange = TRUE)

animationOptions(interval = 1000, loop = FALSE, playButton = NULL, pauseButton = NULL) Arguments

- inputId - The input slot that will be used to access the value.
- label - Display label for the control, or NULL for no label.
- min - The minimum value (inclusive) that can be selected.
- max - The maximum value (inclusive) that can be selected.
- value - The initial value of the slider. A numeric vector of length one will create a regular slider; a numeric vector of length two will create a double-ended range slider. A warning will be issued if the value doesn't fit between min and max.

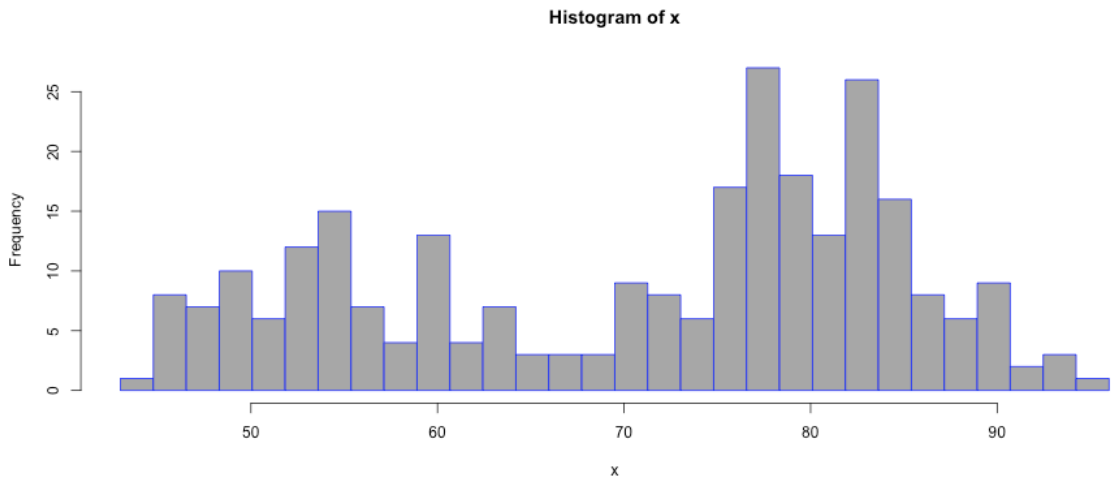
```
sliderInput(inputId="bins",
                         label="Number of bins:",
                         min = 1,
                         max = 50,
                         value = 30)
```



Geyser Eruption Duration

```
renderPlot({
        # generate bins based on input$bins from ui.R
        x    <- faithful[, 2]
        bins <- seq(min(x), max(x), length.out = input$bins + 1)

        # draw the histogram with the specified number of bins
        hist(x, breaks = bins, col = 'darkgray', border = 'blue')
    })
```



We can use similar features in displaying data in a table. Let's use a numericInput to choose how many rows to display from the cars dataset.

```
numericInput("rowsA",
             "How many cars?",
             5)
```

How many cars?

5

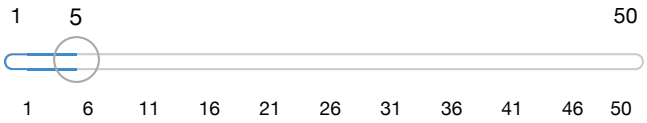
```
renderTable({
  head(cars, input$rowsA)
})
```

speed	dist
4.00	2.00
4.00	10.00
7.00	4.00
7.00	22.00
8.00	16.00

We can do something similar with a slider.

```
sliderInput("rowsB",
            "How many cars?",
            min = 1,
            max = 50,
            value = 5)
```

How many cars?




```
renderTable({
  head(USArrests, input$rowsB)
})
```

Murder	Assault	UrbanPop	Rape
13.20	236	58	21.20
10.00	263	48	44.50
8.10	294	80	31.00

8.80	190	50	19.50
9.00	276	91	40.60

Or we can do a checkbox

```
checkboxInput("checkbox", label = "Choice A", value = TRUE)
```


Choice A

```
renderPrint({ input$checkbox })
```

```
[1] TRUE
```

Takeaways

1. You have to create an input, and use those inputs with the render function
2. the input is its own command
3. the output uses something else avaliable in R (plot, head, etc.) within the wrapped of a render function

Creating Shiny App

As we just discussed, we can create a shiny widget within a Markdown document. However, you may want to fully create your project in Shiny without using a Markdown document first. These steps will provide a basic background on how to do that.

Shiny app basics

A Shiny app is composed of a two parts: a web page that shows the app to the user, and a computer that powers the app. In Shiny terminology, they are called UI (user interface) and server. Those two components overlay what we discussed earlier - the inputs and outputs.

Creating a Shiny App

This is the basic template of code we need:

```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

Along with those four lines of code, you need to run the code in a R script (not a Markdown document) that is named app.R

The file has to have that name to recognize that it's a shiny project. You should see the run button change to "run app" if you've set the code up correctly.

You can see the explicit mention of "ui" and "server" in the code there. Those align somewhat with the inputs and outputs we used earlier, but they also become more complex and require additional bits of code. They're an extra command you need if you're creating a shiny app, and everything will go inside of those two functions.

All of the inputs now go into the ui, which are then area seen by the user and collect their parameters (their manipulations). Based on those manipulations, the server processes the code to display what the user wants. The server runs the analysis, and then renders the output back to the user. It is then displayed as something new, an output function that goes in the server.

Output Function	**Creates**
htmlOutput	raw HTML
imageOutput	image
plotOutput	plot
tableOutput	table
textOutput	text
uiOutput	raw HTML
verbatimTextOutput	text

So we have inputs, renderings, and outputs now. Let's work through that in an example.

Getting Started

First, let's identify the spaces in a shiny app. Most Shiny apps have two spaces, the sidebar and the main panel. The sidebar can be on either the left or right, but we'll start at the left. The sidebar is where your inputs where go, and you have your output in the main panel. Let's label those spaces and add a title

```
ui <- fluidPage(sidebarPanel("This is the sidebar"),
                mainPanel("This is the main panel"),
                titlePanel("TITLE HERE"))
```

We'll use the same code we used to create the Old Faithful Geyser earlier in R Markdown, and see how we'd run that code in a shiny app.

The code from Markdown creating inputs

```
selectInput("n_breaks", label = "Number of bins:",
            choices = c(10, 20, 35, 50), selected = 20)
sliderInput("bw_adjust", label = "Bandwidth adjustment:",
            min = 0.2, max = 2, value = 1, step = 0.2)
```

And the code for the ui

```
ui <- fluidPage(
    sidebarPanel(
      selectInput("n_breaks", label = "Number of bins:",
                  choices = c(10, 20, 35, 50), selected = 20),
      sliderInput("bw_adjust", label = "Bandwidth adjustment:",
                  min = 0.2, max = 2, value = 1, step = 0.2)
    ),

    mainPanel(
      plotOutput("plot")
    )
)
```

Notice that the output function is used in the ui, because it needs to communicate to the server what to do with the analysis. But what about the rendering?

The code from Markdown creating renderings

```
renderPlot({
  hist(faithful$eruptions, probability = TRUE, breaks = as.numeric(i
nput$n_breaks),
       xlab = "Duration (minutes)", main = "Geyser Eruption Duration
")

  dens <- density(faithful$eruptions, adjust = input$bw_adjust)
  lines(dens, col = "blue")
})
```

And in shiny

```
server <- function(input, output)
  {
  output$plot <- renderPlot({
  hist(faithful$eruptions, probability = TRUE, breaks = as.numeric(i
nput$n_breaks),
       xlab = "Duration (minutes)", main = "Geyser Eruption Duration
")

  dens <- density(faithful$eruptions, adjust = input$bw_adjust)
  lines(dens, col = "blue")
  })
  }
```

We use the same renderPlot code, but need to assign it to an output. There are some stylistic differences, but it looks fairly similar. Here's the final code for the shiny app which you can take a run.

```
library(shiny)
library(datasets)
data(faithful)


ui <- fluidPage(

  sidebarPanel(
    selectInput("n_breaks", label = "Number of bins:",
                choices = c(10, 20, 35, 50), selected = 20),
    sliderInput("bw_adjust", label = "Bandwidth adjustment:",
                min = 0.2, max = 2, value = 1, step = 0.2)
  ),


  mainPanel(
    plotOutput("plot")
  )
)


server <- function(input, output)
  {
  output$plot <- renderPlot({
  hist(faithful$eruptions, probability = TRUE, breaks = as.numeric(i
nput$n_breaks),
       xlab = "Duration (minutes)", main = "Geyser Eruption Duration
")

  dens <- density(faithful$eruptions, adjust = input$bw_adjust)
  lines(dens, col = "blue")
  })
  }

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

Takeaways

1. Shiny apps offer a lot of options for customization (See “Application Layout Guide” reading)

2. To do that, they require a few extra commands to set up the page
3. You need to have outputs in both the ui and server components