Creating A Correlation Game Using Shiny

James Gao

11/28/2017

Introduction

Throughout the course, we've made many different shiny apps to create an interactive user interface. In this post, I'll be using shiny to create a game called Guess the Correlation. When I was in high school, my AP Statistics teacher showed us Guess the Correlation at http://guessthecorrelation.com/. In the game, you are presented with a scattterplot with randomly generated points, and the goal is to guess the correlation of the data set. The purpose of this post is to show that using R and shiny, we can create web apps and games.

Organization

To create Guess the Correlation, we'll be creating a shiny app. Shiny apps are composed of two parts: the UI and the server. First, we'll be creating the UI and deciding what options the user will have. Then, we'll create the server to display the randomized scatterplot. We've already created a couple shiny apps throughout the course, so the hard part won't be setting up the UI, it will be figuring out the best way to generate a data set given a randomized correlation.

User Interface

For the sidebar panel UI, we will have a slider for the user to select their guessed correlation, and a "guess" button that the user will press to submit their guess. Once the user guesses, we will also show the actual correlation. Then we'll have a generate button that will generate a new scatterplot. The center of the app will display the randomized scatterplot. Here is what the structure of our app will look like:

```r
#Loading the shiny library
library(shiny)

#Setting our UI object to a variable named ui
ui <- fluidPage(

   # Application title
   titlePanel("Guess the Correlation"),

   # This creates a layout that contains a sidebar
   sidebarLayout(
      #The sidebarPanel contains the widgets contained in the sidebar of our layout
      sidebarPanel(
        #WRITE CODE HERE
        #Slider for the user to guess the correlation

        #WRITE CODE HERE
        #Submit button

        #WRITE CODE HERE
        #Button to generate new scatterplot
      ),

      #The mainPanel contains the output in the main area of our layout
      mainPanel(
        #WRITE CODE HERE
        #Randomized scatterplot

        #WRITE CODE HERE
        #Actual correlation. Will only show once the user has guessed.
      )
   )
)
```

Now, we'll write the code for each of the parts of our UI.

Slider

Let's create the slider for the user to guess the correlation. The correlation between two variables can be anywhere between -1 and 1, so let's set the minimum of our slider to -1, the maximum of our slider to 1, and the default value to 0. Let's also set the step to 0.01 so the user can select a correlation up to two decimal points. We've already used sliders several times, so I don't need to spend too much time explaining how the code for sliders work:

```r
sliderInput("guess",
            "Correlation: ",
            min = -1,
            max = 1,
            value = 0,
            step = 0.01)
```

I've set the name of the input to guess, so later on in the server we can access the user's guess using input$guess.

Submit Button

Now let's create the submit button. We don't need to do too much on the UI side.

```r
actionButton("submit", label = "Guess")
```

When an action button is pressed, its value is increased by one. I initially wanted to reset the value of the button back to 0 whenever the generate new scatterplot button was pressed, so that we would have a convenient way of checking whether the user has pressed the submit button yet on each new scatterplot, but I found that there is no way to do that. While there is a package called shinyjs that can reset the value of inputs in shiny, I found that shinyjs supports literally every widget except action buttons. How unlucky.

Display Actual Correlation

The actual correlation will just be a text output that's calculated server side:

```r
verbatimTextOutput("actualCorrelation")
```

As default, we'll set actualCorrelation to a character vector "Guess First," and once the user guesses we'll change it to the correlation. Then, once the user generates a new scatterplot, we'll have to set it to "Guess First" again.

Generate New Scatterplot

This will be an action button:

```r
actionButton("generate", label = "New Scatterplot")
```

When we click the generate button, we want to do two things: Generate a new scatterplot, and set the actual correlation back to "Guess First". We'll do those in the server.

Display the Scatterplot

This will also be handled in the server. In the UI, we will just have:

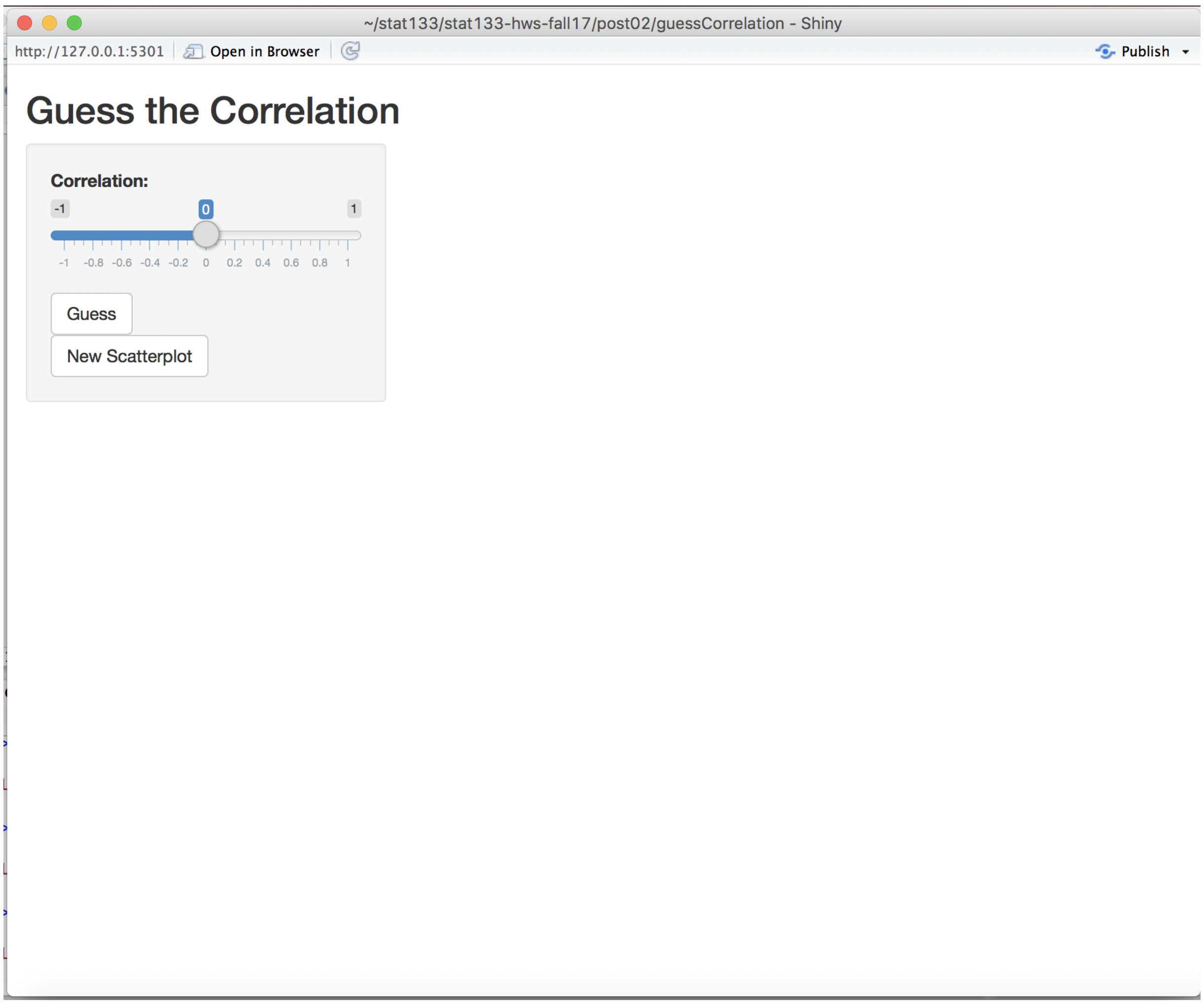
```r
plotOutput("scatterPlot")
```

UI Finished

We've already finished the code for the UI:

```r
#Loading the shiny library
library(shiny)

#Setting our UI object to a variable named ui
ui <- fluidPage(

   # Application title
   titlePanel("Guess the Correlation"),

   # This creates a layout that contains a sidebar
   sidebarLayout(
      #The sidebarPanel contains the widgets contained in the sidebar of our layout
      sidebarPanel(
        #Slider for the user to guess the correlation
        sliderInput("guess",
                    "Correlation: ",
                    min = -1,
                    max = 1,
                    value = 0,
                    step = 0.01),
        #Submit button
        actionButton("submit", label = "Guess"),
        #Button to generate new scatterplot
        actionButton("generate", label = "Generate Scatterplot")
      ),

      #The mainPanel contains the output in the main area of our layout
      mainPanel(
         #Randomized scatterplot
         plotOutput("scatterPlot"),
         #Actual correlation. Will only show once the user has guessed.
         verbatimTextOutput("actualCorrelation")
      )
   )
)
# Run the application
shinyApp(ui = ui, server = server)
```

This is what the app looks like so far:



As you can see, our verbatimTextOutputs and plotOutputs aren't displaying anything because we haven't set them in the server yet. Let's do that now.

Server

The server needs to do these things:
1. Generate random scatterplots at the beginning and whenever the generate button is pressed.
2. Hide the actual correlation whenever the generate button is pressed.
3. Display the actual correlation when the submit button is pressed.
Let's start by creating our scatterplot.

Randomizing Scatterplots

This is the part that takes the most time. Everything else is pretty straightforward, but it's harder than you would think to generate a scatterplot with a random correlation. Your first instinct might be to just randomize a bunch of x and y values and just create a data set with those points and then calculate the correlation. The problem with this is that if we just randomize the x and y of a bunch of data points, then the chance of getting a scatterplot with a correlation between 0 and 0.5 is much much much higher than the chance of getting a scatterplot with a correlation between 0.5 and 1. We want the user to be seeing scatterplots with both low and high correlations. As a result, we must first randomize a correlation and then create the data points from there.

For the purpose of our app, let's just choose to always have 100 data points in our scatterplots. Once we have our random correlation between -1 and 1, how do we create the data points? Correlation is calculated using the formula:

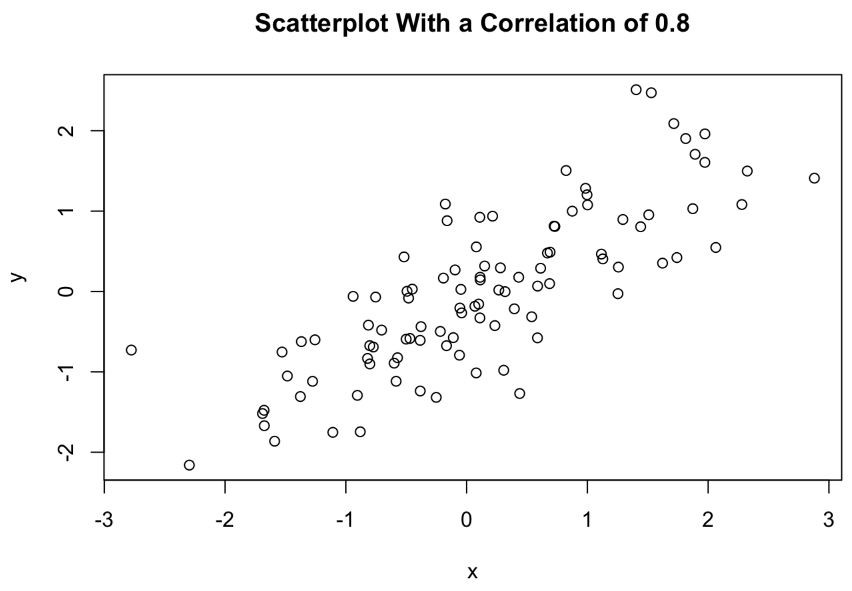
$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

We can already see how it would be very difficult to create a random data set given a correlation. Luckily, I found a very clever solution online: To get two variables given a sample correlation r, you first create two samples with exactly 0 sample correlation, and the exact same variance. Given these two variables x1 and x2, you can create a new variable y1 using the formula:

y1 = r * x2 + sqrt(1-r*r) x1

y1 is guaranteed to have a correlation of r with x2. The hard part is creating an x1 and x2 with a correlation of 0 and the same variance. A trick to do this is to generate two random data sets z1 and z2. If you take z2 and the residuals of the linear regression between z1 and z2, and then scale both to have unit variance, then they will have a correlation of 0. The reason this works is because in a linear regression, one of the assumptions is independence of errors, meaning that there is no relationship between the errors and the y variable. In other words, the residuals are uncorrelated with the y variable, so they will have a correlation of 0. Then, by scaling both to have unit variance using the method scale(), they will have the same variance. With these conditions met, we can create y1 using the formula above to create two variable with our given correlation r. Let's test it out in R to try and make two variables with a correlation of 0.8. Here is what our code will look like:

```r
n <- 100
correlation <- 0.8
x1 = rnorm(n)
x2 = rnorm(n)
y1 = scale(x2) * correlation + scale(residuals(lm(x1~x2))) * sqrt(1-correlation*correlation)
plot(x2, y1, xlab = "x", ylab = "y", main = "Scatterplot With a Correlation of 0.8")
```



Let’s also check to make sure that the correlation between x2 and y1 is 0.8.

```r
cor(x2, y1)
```

```
##      [,1]
## [1,]  0.8
```

Now that we’ve figured out how to create our scatterplots with a random correlation, let’s look at the code for the server to create the scatterplot in our shiny app.

```r
server <- function(input, output) {
   #Number of data points
   n = 100
   #Setting seed
   set.seed(.5)
   #Randomizing correlation
   correlation <- reactive({input$generate * 0 + round(runif(1, -1, 1), 2)})

   #Creating a new scatterplot whenever the button is pressed
   observeEvent(input$generate, {output$scatterPlot <- renderPlot({
     correlation <- correlation()
     x1 = rnorm(n)
     x2 = rnorm(n)
     y1 = scale(x2) * correlation + scale(residuals(lm(x1~x2))) * sqrt(1-correlation*correlation)
     plot(x2, y1, xlab = "x", ylab = "y")
   })})
}
```

I’ve done several things here. First, I’ve set n, the number of points, to 100, and I’ve set a seed for my runif() so that the contents of this post can be replicated. Then, I’m setting correlation using reactive, so that whenever the user requests a new scatterplot, the correlation is re-randomized. In order to do that, I add input$generate * 0 to the randomized correlation so that correlation will update whenever input$generate changes.

Next, in order to update the scatterplot whenever the new scatterplot button is pressed, I’m using the method observeEvent(), which will call its second argument whenever an event occurs on its first argument. So the code:

```r
observeEvent(input$generate, {output$scatterPlot <- renderPlot({
     correlation <- correlation()
     x1 = rnorm(n)
     x2 = rnorm(n)
     y1 = scale(x2) * correlation + scale(residuals(lm(x1~x2))) * sqrt(1-correlation*correlation)
     plot(x2, y1, xlab = "x", ylab = "y")
   })})
```

is saying that whenever the action button for “Generate Scatterplot” is pressed, we will plot a new scatterplot with a re-randomized correlation. To set correlation, we have to call correlation(), which essentially gets the correlation from outside of the renderPlot({}) call. We have to do this because renderPlot({}) creates its own local environment where variables declared outside of it cannot be accessed. The only way to access the correlation we defined earlier is through correlation().

Hide Correlation

Hide Correlation

We’ve now finished creating a new scatterplot whenever we press the new scatterplot button. Now let’s also make it so that whenever we press the new scatterplot button, our actualCorrelation shows: “Actual Correlation: Guess First,” so that the user won’t see the answer until they have at least tried to guess. To do this, we’ll use observeEvent, just like before, and then set output$actualCorrelation to “Actual Correlation: Guess First.”

```
observeEvent(input$generate, {
     output$actualCorrelation <- renderText({
       "Actual Correlation: Guess First"
     })
   })
```

Display Correlation

Whenever the user presses Submit, we want to show them the correct correlation, so we will use observeEvent() with input$submit:

```
observeEvent(input$submit, {
     output$actualCorrelation <- renderPrint({
       correlation <- correlation()
       cat(paste0("Actual Correlation: ", correlation))
       cat(paste0("\nYour Guess: ", input$guess))
       cat(paste0("\nYou were off by: ", round(abs(correlation - input$guess), 2)))
     })
   })
```

We’ll also show the user their guess, and how close they were to the actual correlation.

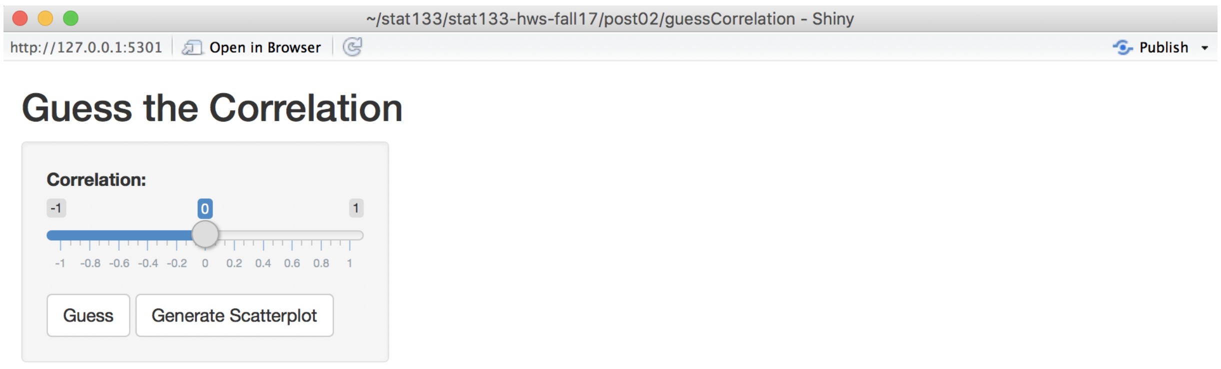
Combining Everything

Combining all of the components that we’ve written, here is the final code for the server:

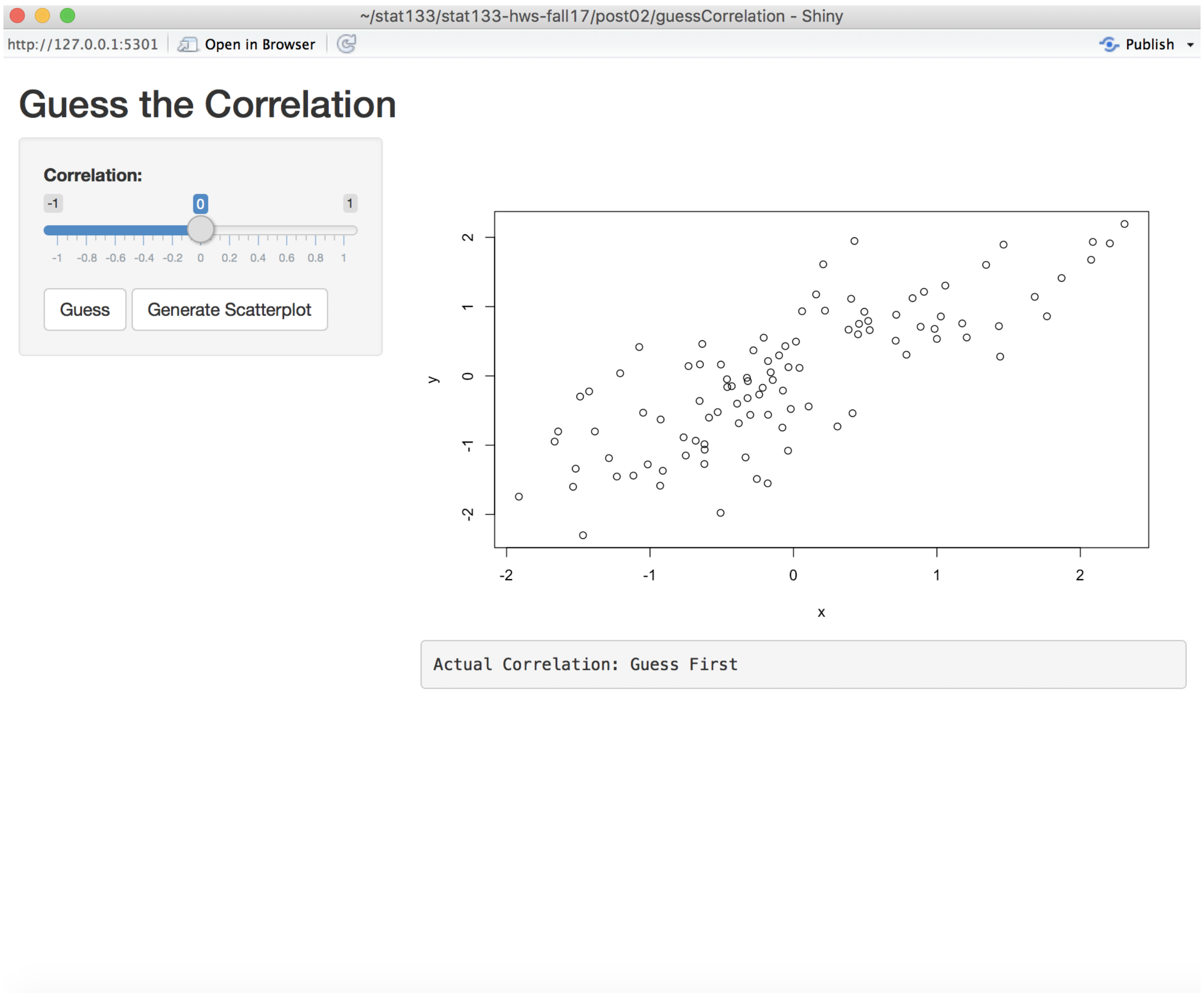
```
server <- function(input, output) {
   #Number of data points
   n = 100
   #Setting seed
   set.seed(.5)
   #Randomizing correlation
   correlation <- reactive({input$generate * 0 + round(runif(1, -1, 1), 2)})

   #Creating a new scatterplot whenever the button is pressed
   observeEvent(input$generate, {output$scatterPlot <- renderPlot({
     correlation <- correlation()
     x1 = rnorm(n)
     x2 = rnorm(n)
     y1 = scale(x2) * correlation + scale(residuals(lm(x1~x2))) * sqrt(1-correlation*correlation)
     plot(x2, y1, xlab = "x", ylab = "y")
   })})
   #Resetting the output text when a new scatterplot is generated
   observeEvent(input$generate, {
     output$actualCorrelation <- renderText({
       "Actual Correlation: Guess First"
     })
   })
   #Revealing the actual correlation when a guess is submitted
   observeEvent(input$submit, {
     output$actualCorrelation <- renderPrint({
       correlation <- correlation()
       cat(paste0("Actual Correlation: ", correlation))
       cat(paste0("\nYour Guess: ", input$guess))
       cat(paste0("\nYou were off by: ", round(abs(correlation - input$guess), 2)))
     })
   })
}
```

Using the App

Launching the app, this is what we see:

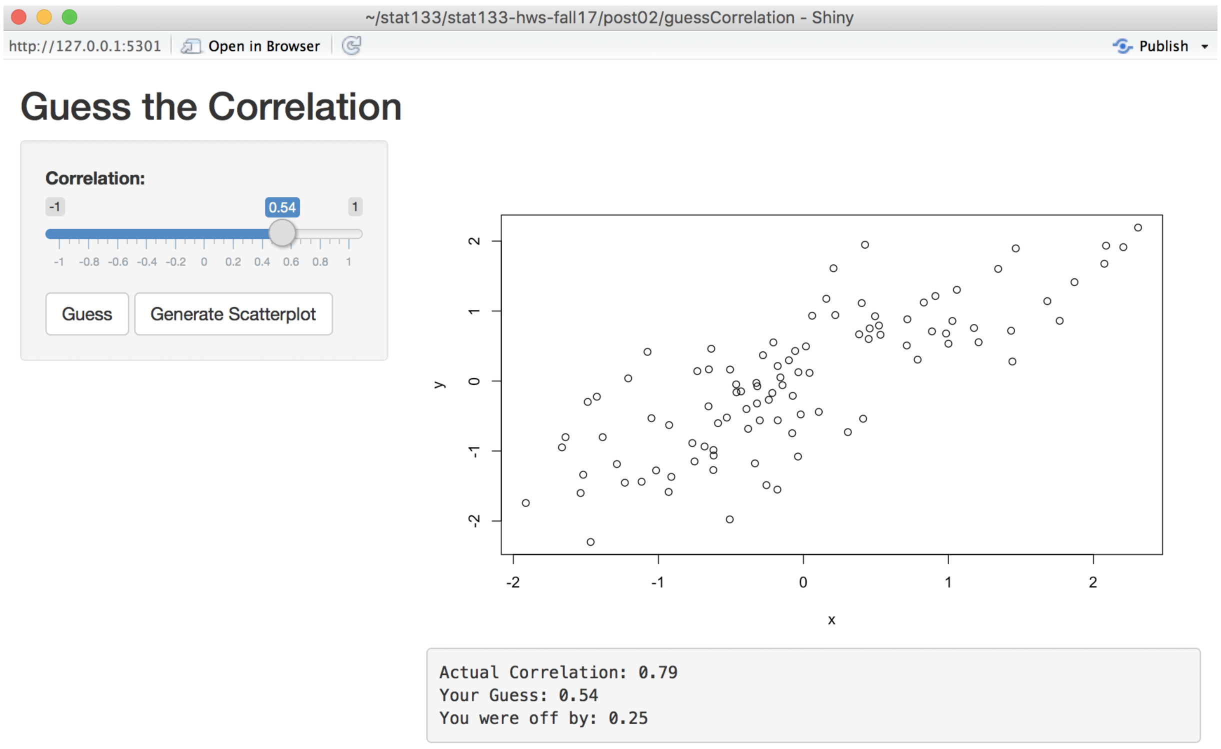


Then, after clicking Generate Scatterplot, a scatterplot appears:



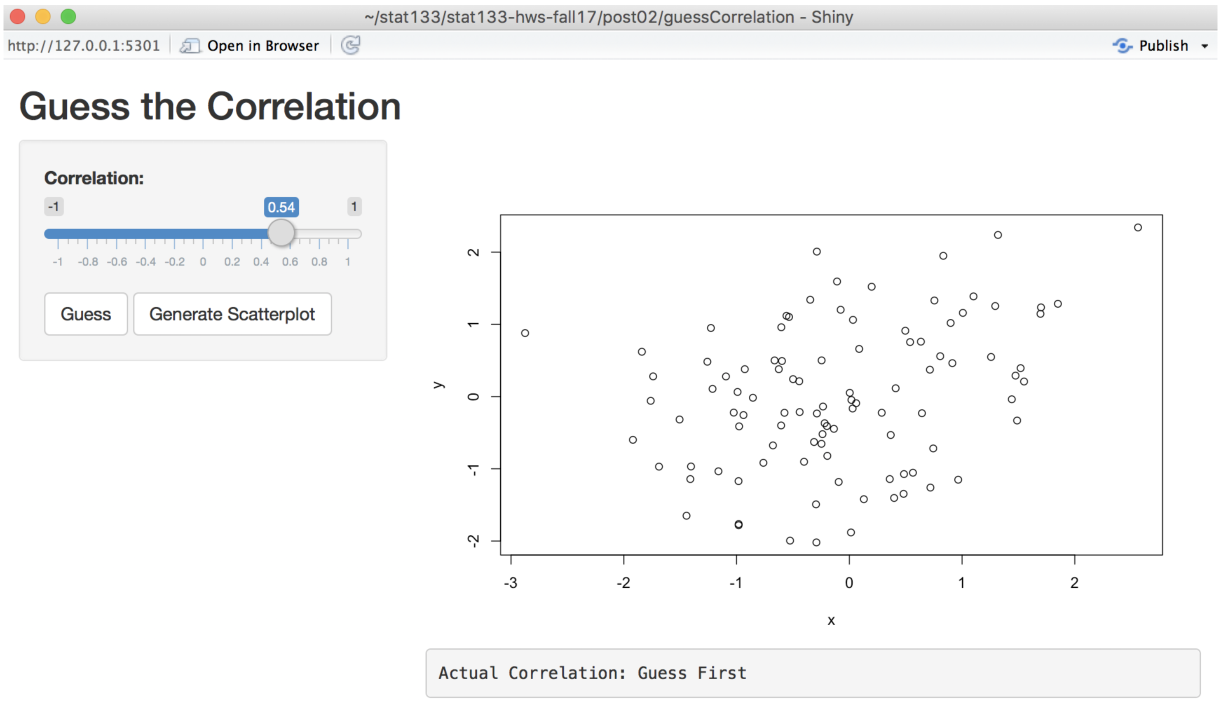
```
Actual Correlation: Guess First
```

After setting the slider to my guess, and clicking Guess, we see the actual correlation displayed in the output:



```
Actual Correlation: 0.79
Your Guess: 0.54
You were off by: 0.25
```

Then, after clicking Generate Scatterplot again, a new scatterplot is generated and the output is set back to default:



```
Actual Correlation: Guess First
```

Conclusion

Throughout this post, I’ve shown how you can use shiny in R to easily create web apps that can serve as games for users. Additionally, by creating two data sets with a given correlation, I’ve shown how R can be used to easily manipulate variables to do tasks that would normally be extremely difficult to do by hand. The take-home message is that by itself, R is already very useful for manipulating data sets, but with all the packages that people have created, R can serve many other purposes as well.

References

-https://shiny.rstudio.com/gallery/widget-gallery.html
-http://guessthecorrelation.com/
-http://www.mathsisfun.com/data/correlation.html
-http://rmarkdown.rstudio.com/authoring_basics.html
-http://shiny.rstudio.com/articles/action-buttons.html
-http://blog.revolutionanalytics.com/2009/02/how-to-choose-a-random-number-in-r.html
-https://stats.stackexchange.com/questions/111865/tool-for-generating-correlated-data-sets
-https://onlinecourses.science.psu.edu/stat200/book/export/html/78