**PSI Special Interest Group: Application and Implementation of Methodologies in Statistics (AIMS)**

Welcome to the second article produced by the PSI AIMS SIG. In this article I introduce Shiny; Shiny is a package which allows the user to create web-based graphical user interfaces for statistical analysis giving full access to the functionality of R.

We shall cover a simple example in this article. I have used RStudio throughout this article with a version of the shiny package that is version 0.14.2 or later.

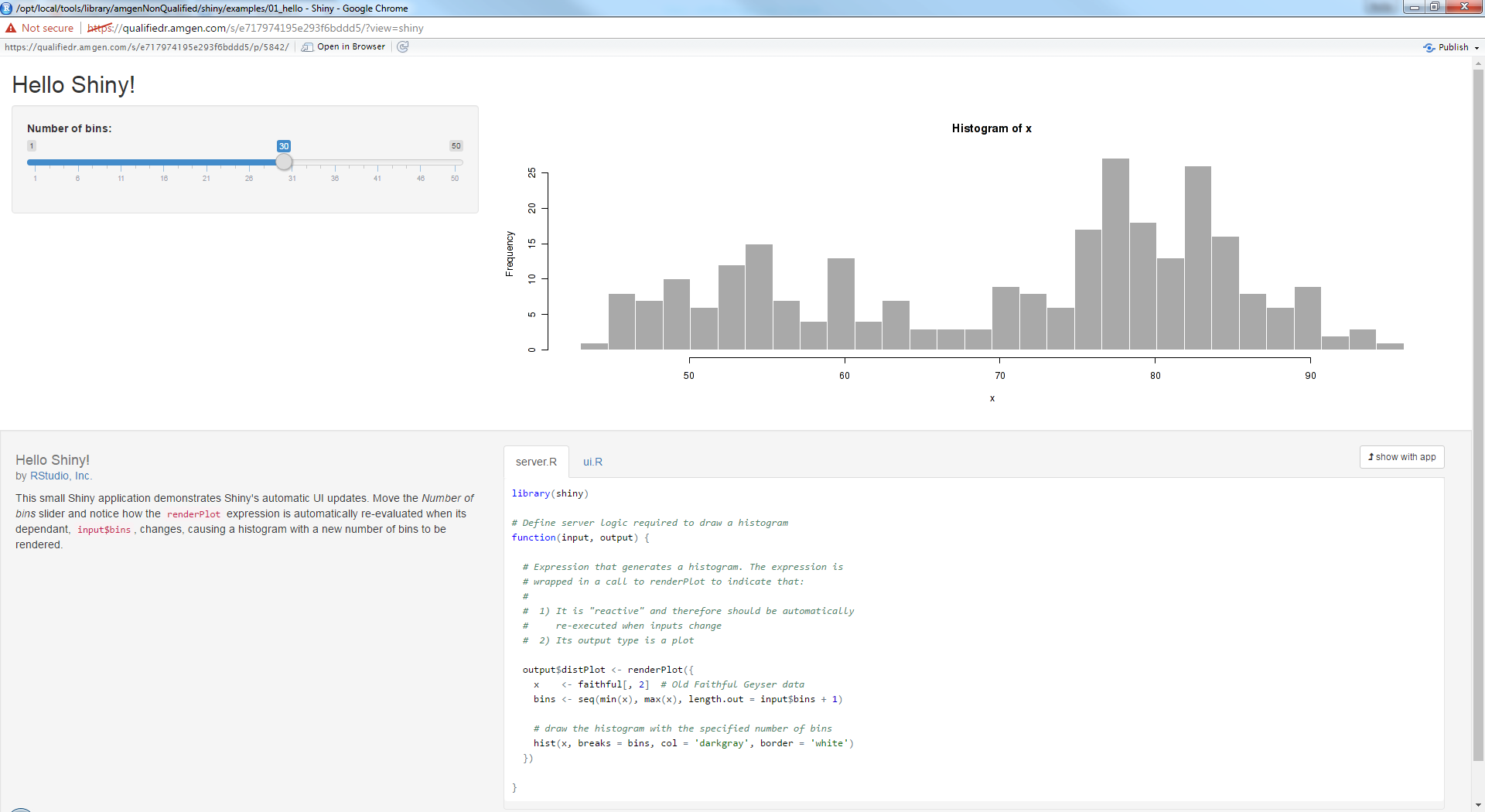
At the R terminal run the following commands:

**> install.packages("shiny")**

**> library(shiny)**

**> runExample("01\_hello")**

You’ll get something like the below:



**(4)**

**(3)**

**(2)**

**(1)**

This is a dynamic histogram of the Old Faithful dataset within R. It contains the waiting times between eruptions and the duration of the eruption for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA. The slider (1) changes the number of cells to be used in the histogram of waiting times; the plot (2) changes to reflect this. A short description (3) provides an introduction and the code (4) is present. Whenever the slider is changed, code on the server.R tab is highlighted yellow to indicate what is being run. This example is presented in “showcase” mode which displays the underlying R code as part of the graphical user interface.

Let’s delve into some trivial code to see how things work:

library(shiny)

ui <- fluidPage(

sidebarLayout(

sidebarPanel(

selectInput(inputId = "text",

label = "Please select your message",

choices = c("Good afternoon",

"Good evening",

"Good night"),

selected = "Good afternoon")

),

mainPanel(textOutput("message"))

)

)

server <- function(input, output) {

output$message <- renderText({input$text})

}

shinyApp(ui = ui, server = server)

***ui*  
inputs to *server***

***server***

**outputs to  
*ui***

**ui**  
This is what the users sees when interacting with the app. The user interacts with the data either by changing something and the result will be stored in a variable. In the code above, the user will see a dropdown menu and text (“Good afternoon”). Any changes to the dropdown will be stored in a variable **text** and **message** will be updated. But how are these elements accessed?

**server**

The server is the ‘back-end’ where user-modifiable elements are read, data is processed and a new output is passed to the ui to display to the user.

Elements *from the ui* are accessed via the **input** variable which is a “list” of variables. In the example above, the user can interact with the app and the subsequent value will be stored in **text**. This can be accessed by the server via **input$text** – anything that server wishes to access from ui can be accessed this way. In server, the variable **message** is accessed similar to **text** this time using **output$message**. This is the server’s way of outputting the variable to the ui so it can be displayed for the user.

**Brief tutorial**

As always with these things, actually coding an example aids with one’s understanding. We shall edit the “01\_hello” app slightly to include a drop down menu.

1. Go to RStudio -> File -> New File -> Shiny Web App.
2. You will be prompted to give it a name (use *choices* for now)
3. Select *Single File (app.R)* (if that option is available to you) and then click create.

Note: earlier versions of Shiny did not have single file support so the ui and server code segments were split across two files. If you are new to shiny it might be a good idea to use two files to begin with, this will help break things down into manageable chunks to aid in reading the code. Then, as you progress, you can switch to using a single file. Beware, however, that there are slight structural syntax differences between using the dual-file and single-file method which will prohibit copying and pasting everything into a single file.

The equivalent code from runExample(“01\_hello”) should now be displayed [if not follow this link to find the code: <https://shiny.rstudio.com/tutorial/lesson1/>]. If you run this app (by selecting all the code and pressing Ctrl+Enter or by clicking ‘Run App’ in the file window if you are using RStudio) it will act precisely like before but the short introductory text and visible code will not be present since it uses “normal” display mode by default and not “showcase” mode. For this tutorial we will learn how to:

* Add a drop down menu
* Select which dataset to use in our histogram via this drop down menu

Firstly, go to code in *ui* and modify it as follows:

**# Sidebar with a slider input for number of bins**

**sidebarLayout(**

**sidebarPanel(**

**sliderInput("bins",**

**"Number of bins:",**

**min = 1,**

**max = 50,**

**value = 30),**

**selectInput(inputId = "data",**

**label = "Select dataset",**

**choices = c("faithful", "mtcars"),**

**selected = "faithful")**

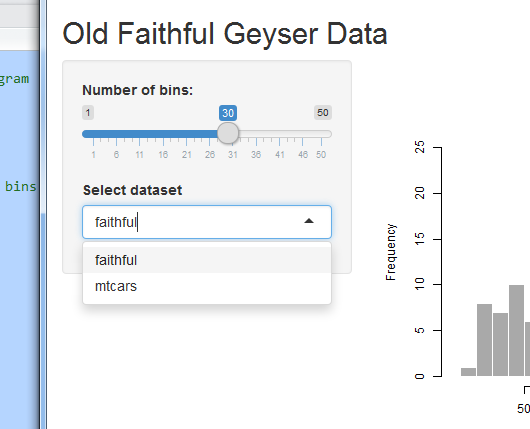
**),**

The **selectInput** function creates a drop-down menu. We have defined our menu as follows:

* inputId – the name of the variable our Shiny app will use to store the choice made by the user
* label – the text to display to the user
* choices – this vector stores all the choices the user can make in the drop-down
* selected – this is the ‘default’ selection from the drop-down. This is useful should we wish the app to read in a default value from the drop-down (as we do in this case).

You can find more information on other elements of this function by looking at the help page for **selectInput**.

Let us run our newly updated app – you should see something similar to this:



The first option (faithful) will be selected by default but you should be able to interact with the drop down to select either choice. The app will not do anything else right now as we have not coded how it will respond to different choices made from the dropdown.

Next we shall modify the server code to take advantage of the choices the user makes. Go to server and change the code as follows:

**output$distPlot <- renderPlot({**

**# generate bins based on input$bins from ui.R**

**if (input$data == "faithful") {**

**x <- faithful[, 2]**

**}**

**else {**

**x <- mtcars[, 5]**

**}**

**bins <- seq(min(x), max(x), length.out = input$bins + 1)**

Here we are making use of the choice made by the user from the drop-down (stored in variable **data**) in an ‘if statement’ to make a decision which dataset to use – recall we access this variable via **input$** as the ui variable is ‘inputted’ to the server. If the user has selected “faithful” we use that dataset (specifically the second column containing the waiting times between eruptions) otherwise we will use mtcars (specifically the fifth column containing the rear axle ratio).

The variables themselves are not that important – what is important, once you’ve run your app is that you can use the slider and histogram, as before, but with a new drop-down menu allowing you to switch between the two datasets.

**Things to consider**

* What happens if we omit selected = “faithful” from selectInput in ui?
* If you do not make the change to the ui above but only use the modification to server, the app will run but you will see this message instead of the plot:

**Error**: argument is of length zero.

What is this error message trying to say?

In summary, our first article introduced you to an IDE (RStudio) with a standard graphical user interface (GUI) which allows you to see datasets, create scripts, view history logs and display plots simultaneously. This second article in the series introduces the shiny package which allows you to create a customized graphical user interface with interactive input and dynamically updated output. To find out more about different Shiny interfaces please visit <http://shiny.rstudio.com/gallery> – the code is very easy to obtain and you can customise it to fit your needs.

All of the AIMS SIG’s work is located on the PSI website: <http://www.psiweb.org/about-us/sigs-special-interest-groups/aims>. Look out for our future articles and feel free to get in touch if you would like to share with the PSI community ways in which you have used R in the pharmaceutical industry.

Chris Toffis (Amgen),

On behalf of PSI AIMS SIG