Introduction to Shiny in R: developing a web application from scratch

Jaynal Abedin
Ph.D. Student
The Insight Centre for Data Analytics
National University of Ireland Galway

June 2018

Outline

- What is shiny?
- What can we do using shiny?
- Getting started with shiny
- Examples of shiny apps (exercise included)

shiny

- *shiny* is an open source R package
- It is a powerful framework for building *interactive* web application
- It helps to turn the analysis into interactive web applications without requiring HTML, CSS, or JavaScript knowledge (source:rstudio)
- *shiny* can be used for prototyping new ideas and communicate with stakeholders effectively
- Some of excellent examples developed using *shiny* http://shiny.rstudio.com/gallery/

End users do not require knowledge of R programming

Getting started with *shiny*

A *shiny* application R code consists of two parts; *ui* and *server*:

- The *ui* part defines graphical user interface of the application
- The *server* part contain actual R code for data processing, analysis and visualization
- The *server* part also contain instructions to interact with different objects in *ui* part
- Finally, to run the application call both *ui* and *server* part inside *shinyApp()* or *runApp()* function

Shiny Example-1-0: shinyDemo

Task: to write code for a blank web application

- No input
- No output
- Display only a blank webpage

Shiny Example-1-0:shinyDemo (cont.)

- The first *shiny* apps without any input or output, it will display a blank webpage

```
library(shiny)
ui <- shinyUI(
  fluidPage()
)

server <- shinyServer(
  function(input, output){
})
```

- To run the above app, save the code using *app.R* inside a folder "*shinyDemo*"
- Put this code `shinyApp(ui,server)` at the end of the code above

Shiny Example-1-1

Each shiny application comes with the following components

- Title
- Sidebar control panel
- Main panel

In this example we will update “shinyDemo” application using following code

Example-1-1: Components of an application

```r
library(shiny)
ui <- shinyUI(
  fluidPage(
    titlePanel("Title of Application"),
    sidebarLayout(
      sidebarPanel("Sidebar Control Panel"),
      mainPanel("Main Panel;
                this panel used primarily for output")
    )
  )
)
server <- shinyServer(
  function(input, output){
  })
```

- Now run the application

Shiny Example-1-2 (ui)

- First attempt to display simple text in output area

```r
library(shiny)
ui <- shinyUI(
  fluidPage(
    titlePanel("Title of Application"),
    sidebarLayout(
      sidebarPanel("Sidebar Control Panel"),
      mainPanel(
        fluidRow(
          column(width = 6,textOutput("txt1")),
          column(width = 6, textOutput("txt2"))
        )
      )
    )
  )
)
```

Shiny Example-1-2 (ui explanation)

- Important point to notice in this code is:

```
    fluidRow(
      column(width = 6,textOutput("txt1")),
      column(width = 6, textOutput("txt2"))
      )
```

- *fluidRow()* is defining the output area as a single row
- *column()* is defining the number of column along with its properties
- Actual output will be displayed in one row and two columns
- Output type will be text output whihc is defined by *textOutput()* function

Shiny Example 1-2 (server)

- Code for server part is:

```
server <- function(input, output){
  output$txt1 <- renderPrint({
    "My first attempt to Shiny apps"
  })
  output$txt2 <- renderText({
    "My first attempt to shiny apps"
  })
}
```

- *server*: a function with two primary argument *input* and *output*
- *output* argument render the output to the *ui* part by identifier used in *ui* part
- In our case we have two identifiers *txt1* and *txt2*
- To render the output *renderPrint()* and *renderText()* are the rendering functions
- Notice the difference in the output of both rendering functions

Shiny Example-1-3 (ui)

- Apps to calculate mean and SD from randomly generated data

```r
library(shiny)
ui <- fluidPage(
  titlePanel("Mean and SD of a Random Variable"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("n", "Sample Size",
                  min = 1, max = 10000,
                  value = 10, step = 10)
    ),
    mainPanel(
      fluidRow(
        column(width = 6,textOutput("txt1")),
        column(width = 6, textOutput("txt2"))
      )
    )
  )
)
```

Shiny Example-1-3 (ui cont.)

- What is the change in the *ui* for this apps?

```r
    sidebarPanel(
      sliderInput("n", "Sample Size",
                  min = 1, max = 10000,
                  value = 10, step = 10)
    )
```

Sample Size

Shiny Example-1-3 (server)

```r
server <- function(input, output){
  output$txt1 <- renderText({
    x <- rnorm(n=input$n)
    paste("Mean = ",paste(round(mean(x),digits = 2)),
          paste("  Standard deviation = "),
          paste(round(sd(x),digits = 2)),sep = "")
  })

  output$txt2 <- renderText({
    x <- rnorm(n=input$n)
    paste("Mean = ",paste(round(mean(x),digits = 2)),
          paste("  Standard deviation = "),
          paste(round(sd(x),digits = 2)),sep = "")
  })
}
```

Shiny Example-1-4 (ui)

- Summary statistics and graphs

```r
ui <- fluidPage(titlePanel("Mean, SD and
             Histogram of a Random Variable "),
  sidebarLayout(sidebarPanel(
      sliderInput("n", "Sample Size",
                  min = 1, max = 1000, value = 10)
    ),
    mainPanel(fluidRow(
        column(width = 6,textOutput("txt1")),
        column(width = 6, textOutput("txt2"))
      ),
      fluidRow(
        column(width = 6,plotOutput("plot1",height = 500))
      )
    )
  )
)
```

Shiny Example-1-4 (ui cont.)

- What is/are the new item in the *ui* code?

```
    fluidRow(
      column(width = 6,plotOutput("plot1",height = 500))
    )
```

- Using this code we define output type, location and identifier

Shiny Example-1-4 (server part-1)

- Let's see the code part by part from server section

```r
server <- function(input, output){
  getData <- reactive({
    x <- rnorm(n=input$n)
    x
  })
  output$txt1 <-
    renderText({paste("Mean = ",
                      paste(round(mean(getData()),
                                  digits = 2)),
                      paste("  Standard deviation = "),
                      paste(round(sd(getData()),
                                  digits = 2)),sep = "")
  })
}
```

Shiny Example-1-4 (server part-2)

```
server <- function(input, output){
  getData <- reactive({
    x <- rnorm(n=input$n)
    x
  })
  output$txt2 <-
    renderText({paste("Mean = ",
                      paste(round(mean(getData()),
                                  digits = 2)),
                      paste("  Standard deviation = "),
                      paste(round(sd(getData()),
                                  digits = 2)),sep = "")
  })
  output$plot1 <- renderPlot({
    hist(getData(),main = "Histogram", xlab = "")
  })
}
```

Shiny Example 1-5

- Generate a Binomial population of size 20000
- Take a random sample of size 10, 50, 100, and 1000
- Calculate proportion of 1's
- Repeat the calculation many times for each sample size
- Create a density plot of the proportions

Shiny Exercise-1

Write code to build an apps to do the following things:

- Define *ui* to take three input, mean(0 to 5), sd(1 to 3) and n(10 to 1000)
- Generate a random sample from Normal(mean,sd,n)
- Calculate seven number summary statistics (min, q1, mean, median, q3, max and sd) and store them into a dataframe
- Create a histogram of the data generated from normal distribution
- Display the summary statistics data frame in output area
- Hints: *tableOutput()*, *plotOutput()* in ui section and *renderTable()*, *renderPlot()* in server section

Shiny Example 1-6 (file input)

- Upload user's own data into shiny apps
- Code to include in *ui* section:

```
fileInput('file1', 'Choose a file',
accept=c('text/csv', 'text/comma-separated-values,
text/plain', '.csv'))
```

- code to include in server section

```
dataInput <- reactive({  inFile <- input$file1  if
(is.null(inFile))  return(NULL)
read.csv(inFile$datapath, as.is=T) })
```

- Complete the apps and run

Shiny Example 1-7 (Use of default data)

- Load *Wage* data from *ISLR* library
- Create a boxplot of Sage for different level of education
- Hints: *data("Wage")*, *ggpplot*, *geom_boxplot()*