STAT 231: Problem Set 3A

Alastair Poole

due by 5 PM on Monday, March 8

In order to most effectively digest the Shiny video tutorial from RStudio – and the new R commands it presents – this series A homework assignment is designed to encourage you to watch the tutorial "actively" and in line with the textbook's Prop Tip of page 33:

"**Pro Tip**: If you want to learn how to use a particular command, we highly recommend running the example code on your own"

Series A assignments are intended to be completed individually. The problems should be straightforward based on the video tutorial, but if you have any questions, feel free to ask me!

Steps to proceed:

1. In RStudio, go to File > Open Project, navigate to the folder with the course-content repo, select the course-content project (course-content.Rproj), and click "Open"
2. Pull the course-content repo (e.g. using the blue-ish down arrow in the Git tab in upper right window)
3. Copy ps3A.Rmd from the course repo to your repo (see page 6 of the GitHub Classroom Guide for Stat231 if needed)
4. Close the course-content repo project in RStudio
5. Open YOUR repo project in RStudio
6. In the ps3A.Rmd file in YOUR repo, replace "YOUR NAME HERE" with your name
7. Add in your responses, committing and pushing to YOUR repo in appropriate places along the way
8. Run "Knit PDF"
9. Upload the pdf to Gradescope. Don't forget to select which of your pages are associated with each problem. *You will not get credit for work on unassigned pages (e.g., if you only selected the first page but your solution spans two pages, you would lose points for any part on the second page that the grader can't see).*

Interactive Web Apps with Shiny

Chapter 11 in MDSR explores a few different alternatives for making more complex – and, in particular, dynamic – data graphics. In this course, we will focus on building interactive web apps with Shiny (Section 11.3). The textbook reading is optional this week; if you want to get a sense of other ways to create dynamic visualizations in R, you can read through the chapter.

This week, instead of coding along with the code in the textbook chapter, you will code along with the code in a Shiny video tutorial created by RStudio.

1. Set-up and initial exploration

If you're working in R/RStudio on your own machine, you may need to install the `shiny` package (`install.packages("shiny")`). To ensure you successfully installed Shiny, try running one of the demo apps.

Then, go to the Shiny gallery to explore various Shiny apps and get a sense of the (seemingly endlesss) possibilities for a Shiny app.

You do not have to write anything here. Just explore! Be curious!

```
# run some examples to ensure shiny package is successfully installed
runExample("01_hello")
runExample("06_tabsets")

# to see what other examples are available
runExample()
```

2. Shiny tutorial

Go to: https://shiny.rstudio.com/tutorial/ and watch the complete (except for certain chapters specified below) tutorial (~2.5 hours).

I've included the example code in the video tutorial below. Each new code chunk is a different .R file. There is a lot of code here! Don't be frightened. I don't expect you to understand everything at once. But I hope that you run some of the example code along with the video (e.g., pause the video, run the app, update the app in some way – it helps to be curious: what happens if I change this to that? Experiment with the code.). I also intend for this to be a resource you could go back to when designing your own app (e.g., you want to change the layout of your app to use a navigation bar menu, but don't remember how – you can start by following the bare-bones template from the `09-navbarMenu.R` code chunk below).

The full code and slides from the video are also available here: https://github.com/rstudio-education/shiny.rstudio.com-tutorial

a.

As you watch, or after watching, the tutorial, write down at least two questions you have about Shiny applications.

> ANSWER: What does the histogram used in the example acutally show? I don't understand how changes made by the user on the slider actually show up in the histogram that reflects these changes. I can see the histogram changing, but when talking about frequency or number of observations, what effect does moving around the slider have? My second question is about what is the conceptual difference between render functions and output functions, and when should I should I use which one? I understand both are needed, but I felt like sometimes the input function would communicate with the output function, and sometimes it communicates with a render function, and I'm not quite sure why that is.

b.

After watching the tutorial, create a new folder called "ps3a_shiny" within your "homeworks" folder in your GitHub repository. Open a new R file (**not** R Markdown) by going to File > New File > R Script. (In a .R file, you can only write R code. Any comments need to have a hashtag (#) in front of them.) Save the file as `app.R` within the "ps3a_shiny" folder. Then,

- copy the code from the `02-two-outputs.R` file (in the `two-outputs` code chunk below)
- add a `textInput` widget that allows the user to change the title of the histogram (following code in `01-two-inputs.R`). Update the code in the server() function appropriately. Run the app to make sure it works as you expect.
- update the layout of the app to use a `navlistPanel` structure (following the code in `06-navlist.R`). Hint: put `navlistPanel` around the output objects only.

Make sure the app runs successfully, then save your changes in the app.R file, and push app.R to your GitHub repo. You do not need to write anything here. I will check your app in your GitHub repo. If you get stuck, email me or the TA!

Part 1. How to build a Shiny app (ch. 1-6)

Note: You do *NOT* need to publish your app with shinyapps.io for this problem set. We will use class time this week to publish an app. Feel free to *skip over* ch. 7-9 in part 1 of the video tutorial.

```
# 01-template.R

library(shiny)
ui <- fluidPage()

server <- function(input, output) {}

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

```
#02-hist-app.R

library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}

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

```
# app.R
# 01-kmeans-app

palette(c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3",
  "#FF7F00", "#FFFF33", "#A65628", "#F781BF", "#999999"))

library(shiny)

ui <- fluidPage(
  headerPanel('Iris k-means clustering'),
  sidebarPanel(
    selectInput('xcol', 'X Variable', names(iris)),
    selectInput('ycol', 'Y Variable', names(iris),
      selected = names(iris)[[2]]),
    numericInput('clusters', 'Cluster count', 3,
      min = 1, max = 9)
  ),
  mainPanel(
    plotOutput('plot1')
  )
)
```

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

  selectedData <- reactive({
    iris[, c(input$xcol, input$ycol)]
  })

  clusters <- reactive({
    kmeans(selectedData(), input$clusters)
  })

  output$plot1 <- renderPlot({
    par(mar = c(5.1, 4.1, 0, 1))
    plot(selectedData(),
         col = clusters()$cluster,
         pch = 20, cex = 3)
    points(clusters()$centers, pch = 4, cex = 4, lwd = 4)
  })

}

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

Part 2. How to customize reactions (ch. 11-23)

```r
# 01-two-inputs.R
# ~ 00:58:10

library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  textInput(inputId = "title",
    label = "Write a title",
    value = "Histogram of Random Normal Values"),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num), main = input$title)
  })
}

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

```r
# 02-two-outputs.R
# ~ 01:00:00

library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist"),
  verbatimTextOutput("stats")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
  output$stats <- renderPrint({
    summary(rnorm(input$num))
  })
}

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

```r
# 03-reactive.R
# ~ 01:04:20

library(shiny)
```

```r
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist"),
  verbatimTextOutput("stats")
)

server <- function(input, output) {

  data <- reactive({
    rnorm(input$num)
  })

  output$hist <- renderPlot({
    hist(data())
  })
  output$stats <- renderPrint({
    summary(data())
  })
}

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

```r
# 04-isolate.R
# ~ 01:10:17

library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  textInput(inputId = "title",
    label = "Write a title",
    value = "Histogram of Random Normal Values"),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num), main = isolate(input$title))
  })
}

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

```r
# 05-actionButton.R
# ~ 01:14:50


library(shiny)
```

```
ui <- fluidPage(
  actionButton(inputId = "clicks",
    label = "Click me")
)

server <- function(input, output) {
  observeEvent(input$clicks, {
    print(as.numeric(input$clicks))
  })
}

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

```
# 06-observeEvent.R
# ~ 01:18:00

library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    min = 1, max = 100, value = 25),
  actionButton(inputId = "go",
    label = "Print Value")
)

server <- function(input, output) {

  # observe responds to the print button
  # but not the slider
  observeEvent(input$go, {
    print(as.numeric(input$num))
  })
}

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

```
# 07-eventReactive.R
# ~ 01:20:22

library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  actionButton(inputId = "go",
    label = "Update"),
  plotOutput("hist")
)

server <- function(input, output) {
  data <- eventReactive(input$go, {
```

```
    rnorm(input$num)
  })

  output$hist <- renderPlot({
    hist(data())
  })
}

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

```
# 08-reactiveValues.R
# ~ 01:24:18

library(shiny)

ui <- fluidPage(
  actionButton(inputId = "norm", label = "Normal"),
  actionButton(inputId = "unif", label = "Uniform"),
  plotOutput("hist")
)

server <- function(input, output) {

  rv <- reactiveValues(data = rnorm(100))

  observeEvent(input$norm, { rv$data <- rnorm(100) })
  observeEvent(input$unif, { rv$data <- runif(100) })

  output$hist <- renderPlot({
    hist(rv$data)
  })
}

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

Part 3. How to customize appearance (ch. 24-30 & 32)

You can *skip over* the CSS chapter (ch. 31).

```
# 02-tags.R
# ~ 01:50:20

library(shiny)

ui <- fluidPage(
  h1("My Shiny App"),
  p(style = "font-family:Impact",
    "See other apps in the",
    a("Shiny Showcase",
      href = "http://www.rstudio.com/products/shiny/shiny-user-showcase/")
  )
)

server <- function(input, output){}

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

```
# 03-layout.R
# ~ 01:56:00

library(shiny)

ui <- fluidPage(
  fluidRow(
   column(3),
   column(5, sliderInput(inputId = "num",
     label = "Choose a number",
     value = 25, min = 1, max = 100))
  ),
  fluidRow(
    column(4, offset = 8,
      plotOutput("hist")
    )
  )
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}

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

```
# 04-well.R
# ~ 01:59:57

library(shiny)
```

```
ui <- fluidPage(
  wellPanel(
    sliderInput(inputId = "num",
      label = "Choose a number",
      value = 25, min = 1, max = 100),
    textInput(inputId = "title",
      label = "Write a title",
      value = "Histogram of Random Normal Values")
  ),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num), main = input$title)
  })
}

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

```
# 05-tabs.R
# ~ 02:01:37

library(shiny)

ui <- fluidPage(title = "Random generator",
  tabsetPanel(
    tabPanel(title = "Normal data",
      plotOutput("norm"),
      actionButton("renorm", "Resample")
    ),
    tabPanel(title = "Uniform data",
      plotOutput("unif"),
      actionButton("reunif", "Resample")
    ),
    tabPanel(title = "Chi Squared data",
      plotOutput("chisq"),
      actionButton("rechisq", "Resample")
    )
  )
)

server <- function(input, output) {

  rv <- reactiveValues(
    norm = rnorm(500),
    unif = runif(500),
    chisq = rchisq(500, 2))

  observeEvent(input$renorm, { rv$norm <- rnorm(500) })
  observeEvent(input$reunif, { rv$unif <- runif(500) })
  observeEvent(input$rechisq, { rv$chisq <- rchisq(500, 2) })
```

```
  output$norm <- renderPlot({
    hist(rv$norm, breaks = 30, col = "grey", border = "white",
      main = "500 random draws from a standard normal distribution")
  })
  output$unif <- renderPlot({
    hist(rv$unif, breaks = 30, col = "grey", border = "white",
      main = "500 random draws from a standard uniform distribution")
  })
  output$chisq <- renderPlot({
    hist(rv$chisq, breaks = 30, col = "grey", border = "white",
       main = "500 random draws from a Chi Square distribution with two degree of freedom")
  })
}

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

```
# 06-navlist.R
# ~ 02:03:53

library(shiny)

ui <- fluidPage(title = "Random generator",
  navlistPanel(
    tabPanel(title = "Normal data",
      plotOutput("norm"),
      actionButton("renorm", "Resample")
    ),
    tabPanel(title = "Uniform data",
      plotOutput("unif"),
      actionButton("reunif", "Resample")
    ),
    tabPanel(title = "Chi Squared data",
      plotOutput("chisq"),
      actionButton("rechisq", "Resample")
    )
  )
)

server <- function(input, output) {

  rv <- reactiveValues(
    norm = rnorm(500),
    unif = runif(500),
    chisq = rchisq(500, 2))

  observeEvent(input$renorm, { rv$norm <- rnorm(500) })
  observeEvent(input$reunif, { rv$unif <- runif(500) })
  observeEvent(input$rechisq, { rv$chisq <- rchisq(500, 2) })

  output$norm <- renderPlot({
    hist(rv$norm, breaks = 30, col = "grey", border = "white",
      main = "500 random draws from a standard normal distribution")
  })
```

```
  output$unif <- renderPlot({
    hist(rv$unif, breaks = 30, col = "grey", border = "white",
      main = "500 random draws from a standard uniform distribution")
  })
  output$chisq <- renderPlot({
    hist(rv$chisq, breaks = 30, col = "grey", border = "white",
       main = "500 random draws from a Chi Square distribution with two degree of freedom")
  })
}

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

```
# 07-sidebar.R
# ~ 02:05:39

library(shiny)

ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      sliderInput(inputId = "num",
        label = "Choose a number",
        value = 25, min = 1, max = 100),
      textInput(inputId = "title",
        label = "Write a title",
        value = "Histogram of Random Normal Values")
    ),
    mainPanel(
      plotOutput("hist")
    )
  )
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num), main = input$title)
  })
}

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

```
# 08-navbarPage.R
# ~ 02:07:56

library(shiny)

ui <- navbarPage(title = "Random generator",
    tabPanel(title = "Normal data",
      plotOutput("norm"),
      actionButton("renorm", "Resample")
    ),
    tabPanel(title = "Uniform data",
      plotOutput("unif"),
```

```
      actionButton("reunif", "Resample")
    ),
    tabPanel(title = "Chi Squared data",
      plotOutput("chisq"),
      actionButton("rechisq", "Resample")
    )

)

server <- function(input, output) {

  rv <- reactiveValues(
    norm = rnorm(500),
    unif = runif(500),
    chisq = rchisq(500, 2))

  observeEvent(input$renorm, { rv$norm <- rnorm(500) })
  observeEvent(input$reunif, { rv$unif <- runif(500) })
  observeEvent(input$rechisq, { rv$chisq <- rchisq(500, 2) })

  output$norm <- renderPlot({
    hist(rv$norm, breaks = 30, col = "grey", border = "white",
      main = "500 random draws from a standard normal distribution")
  })
  output$unif <- renderPlot({
    hist(rv$unif, breaks = 30, col = "grey", border = "white",
      main = "500 random draws from a standard uniform distribution")
  })
  output$chisq <- renderPlot({
    hist(rv$chisq, breaks = 30, col = "grey", border = "white",
       main = "500 random draws from a Chi Square distribution with two degree of freedom")
  })
}

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

```
# 09-navbarMenu.R
# ~ 02:09:43

library(shiny)

ui <- navbarPage(title = "Random generator",
  tabPanel(title = "Normal data",
    plotOutput("norm"),
    actionButton("renorm", "Resample")
  ),
  navbarMenu(title = "Other data",
    tabPanel(title = "Uniform data",
      plotOutput("unif"),
      actionButton("reunif", "Resample")
    ),
    tabPanel(title = "Chi Squared data",
      plotOutput("chisq"),
```

```
      actionButton("rechisq", "Resample")
    )
  )
)

server <- function(input, output) {

  rv <- reactiveValues(
    norm = rnorm(500),
    unif = runif(500),
    chisq = rchisq(500, 2))

  observeEvent(input$renorm, { rv$norm <- rnorm(500) })
  observeEvent(input$reunif, { rv$unif <- runif(500) })
  observeEvent(input$rechisq, { rv$chisq <- rchisq(500, 2) })

  output$norm <- renderPlot({
    hist(rv$norm, breaks = 30, col = "grey", border = "white",
      main = "500 random draws from a standard normal distribution")
  })
  output$unif <- renderPlot({
    hist(rv$unif, breaks = 30, col = "grey", border = "white",
      main = "500 random draws from a standard uniform distribution")
  })
  output$chisq <- renderPlot({
    hist(rv$chisq, breaks = 30, col = "grey", border = "white",
       main = "500 random draws from a Chi Square distribution with two degree of freedom")
  })
}

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