Interactive web apps with Shiny

Jeff Oliver

08 April, 2021

The Shiny package for R provides a way to transform your R code into interactive applications accessible through a web browser. This lesson provides a hands-on example of how you can use Shiny to create intuitive graphical user interfaces.

Learning objectives

1. Explain the different responsibilities of the user interface and the server function
2. Manipulate user interface options in side panels
3. Apply defensive programming techniques to reduce errors
4. Diagnose problems with Shiny built-in debugging features
5. Share the application through shinyapps.io

Description

While the R programming language is immensely popular, not everyone has the time or resources to learn how to use it (a shame, I know). How then, can you share your awesome data visualization tools that you wrote in R? The Shiny package provides the architecture to build beautiful web applications without writing a single HTML tag. This workshop will introduce some of the functionality of the package as well as platforms you can use to share your work.

Getting started

To start with, we will need to install the shiny package for R. For this lesson, we also be using the ggplot2 package for data visualization and the palmerpenguins package for our data source.

```
install.packages("shiny")
install.packages("ggplot2")
install.packages("palmerpenguins")
```

Note, I misspell "palmerpenguins" more frequently that I would like to admit, so if you encounter installation problems with that package, double-check to make sure it is spelled correctly.

We are now ready to start our Shiny Application!

- From the File menu, select New Project…
- Choose "New Directory" in the first dialog

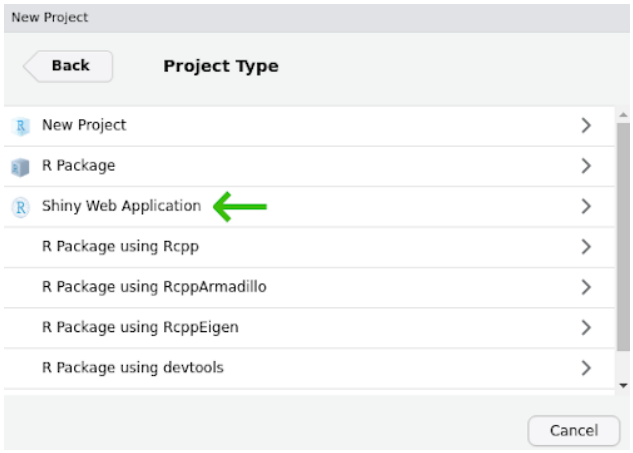

Figure 1: The new Shiny web application dialog

- In the Project Type dialog, select "Shiny Web Application", which is probably the third option in the list. If you do not see this as an option, try shutting down RStudio and starting it up again.

Name your project "shiny-lesson" and be sure you save it somewhere you can remember (I usually save these lessons to the Desktop or My Documents).

Two halves make an app

The shiny application file you just created has the two critical parts:

1. `ui`, which starts on line 13. `ui` handles all communication with the user: it records choices and information that the user inputs and displays any output based on those selections.
2. `server` which starts on line 36. `server` takes care of any calculations and does the heavy lifting for creating any data visualizations.

The `ui` code:

```
# Define UI for application that draws a histogram
ui <- fluidPage(

    # Application title
    titlePanel("Old Faithful Geyser Data"),

    # Sidebar with a slider input for number of bins
    sidebarLayout(
        sidebarPanel(
            sliderInput("bins",
                        "Number of bins:",
                        min = 1,
                        max = 50,
                        value = 30)
        ),
```

```
        # Show a plot of the generated distribution
        mainPanel(
           plotOutput("distPlot")
        )
    )
)
```

The `server` code:

```
# Define server logic required to draw a histogram
server <- function(input, output) {

    output$distPlot <- renderPlot({
        # generate bins based on input$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 = 'white')
    })
}
```

There is actually an important third part of the the file, too, at the very end. Line 49 has:

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

This command actually starts up the application so you can actually use it, so make sure not to delete it.

Test drive the app

Whenever you start a new Shiny app, the file will have the bare bones of an app as in the file you see here. Let us first see what this app does, then we will modify it for our own uses. Press the "Run App" button at the top of the app.R script.

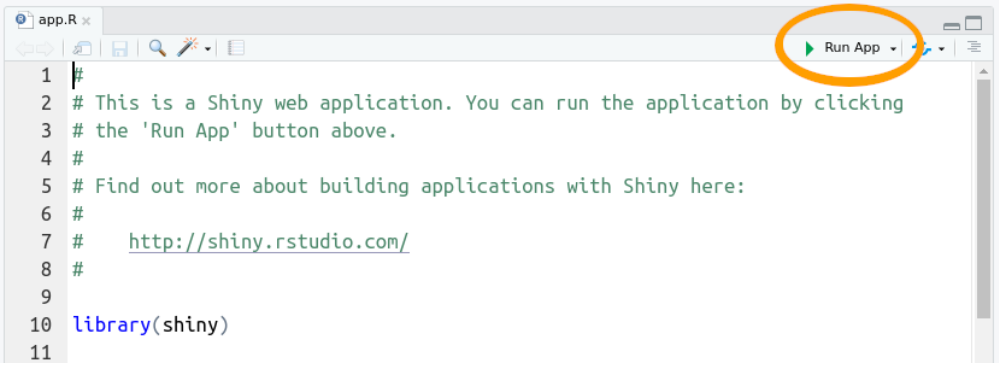

Figure 2: Run the shiny app

This app takes one piece of input from the user (the number of bins to use for the histogram) and updates the visualization based on the user's selection.

If we look back at the code in our app script, the `ui` section adds the title we see at the top of the app ("Old Faithful Geyser Data") and calls the function `sidebarLayout()`. This is where most of your code for communicating with the user is going to be. Here it does two things:

1. Creates the slider with `sliderInput()`
2. Displays the plot with `plotOutput()`

You probably also noted that the slider controls are with a call to `sidebarPanel()` and the plot display happens inside a call to `mainPanel()`. We will not mess around too much with this, but in general we will use `sidebarPanel()` to gather user input and `mainPanel()` for any output we want to display.

Don't start from scratch

Our goal is to create an app that allows users to choose among three species of penguins (Gentoo, Adelie, and Chinstrap) and display a plot of bill length vs. bill depth.

Now, we want an application that has nothing to do with geyser eruptions, but instead of deleting everything in the file and starting from scratch, we start by adding comments to indicate what we want to do and commenting out the original code. We can delete the old code later, but for now we can keep it to remind us of how shiny apps work.

Since our app will use the ggplot2 and palmerpenguins packages, go ahead and add library calls to those two packages right after the shiny package is loaded.

```
library(shiny)
library(palmerpenguins)
library(ggplot2)
```

Start by updating the title in the call to `titlePanel()`

```
titlePanel("Penguin size relationships")
```

Change the comment above sidebar layout to indicate what we want users to do, that is, we will want them to select the name of a penguin species.

```
# Sidebar with text input for penguin species
    sidebarLayout(
      sidebarPanel(
```

We can also update the comment above the main panel code to reflect the type of plot we will display.

```
# Show a size plot for selected species
```

We are still at the point of writing comments, which are really instructions to ourselves about the code we need to write. So scroll on down to the `server` section add add a comment before `output$distPlot <- renderPlot` (around line 37 in the file):

```
# Create size plot for penguin species
```

Now we start updating the code

At this point we haven't changed much besides some comments. If you run the app, it is still going to be the one plotting eruptions of Old Faithful. There are two changes we need to make to the actual code: one in the `ui` function and one in the `server` function. The first (in `ui`) is all about getting information from the user.

Navigate to the sidebar panel, has a `sliderInput` function:

```
# Sidebar with text input for penguin species
    sidebarLayout(
        sidebarPanel(
            sliderInput("bins",
                        "Number of bins:",
                        min = 1,
                        max = 50,
                        value = 30)
        ),
```

Start by deleting the `sliderInput`:

```
# Sidebar with text input for penguin species
    sidebarLayout(
        sidebarPanel(
        ),
```

And add, inside the call to `sidebarPanel`, a call `textInput`:

```
# Sidebar with text input for penguin species
    sidebarLayout(
        sidebarPanel(
            textInput(inputId = "species",
                      label = "Species",
                      value = "Gentoo")
        ),
```

For `textInput`:

1. `inputId = "species"` is the identifier for the information the user enters; we will see this in use later in the `server` section. Just remember that we used all lower-case letters.
2. `label = "Species"` is the text that will appear to the user. Note here we capitalized "Species".
3. `value = "Gentoo"` indicates the default value that `species` will take.

The next change occurs in the `server` section. Navigate to the comment we added about creating a plot (around line 35 of the file):

```
# Create size plot for penguin species
    output$distPlot <- renderPlot({
        # generate bins based on input$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 = 'white')
    })
```

Since we are not going to use the code for plotting a histogram, we can delete all the code within `renderPlot` (but be sure to leave the opening `({` and closing `})`):

```
# Create size plot for penguin species
    output$distPlot <- renderPlot({

    })
```

And we now add our ggplot2 commands to create a scatter plot of bill depth versus bill length. (not familiar with ggplot2? Check out the introduction to ggplot lesson).

```
# Create size plot for penguin species
    output$distPlot <- renderPlot({
              ggplot(data = penguins[penguins$species == input$species, ],
               mapping = aes(x = bill_length_mm,
                             y = bill_depth_mm,
                             color = sex)) +
            geom_point()
    })
```

Note the data we specified to use for the plot:

```
data = penguins[penguins$species == input$species, ]
```

`penguins$species` is the data set we loaded from the palmerpenguins package, but what about that `input` thing? Where did that come from?

Adding the interactivity

Do panel manipulation with Species first as a text field, then change this to a drop-down menu (defensive programming)

`textInput`

PEBKAC

We've all been there. Here's the point for defensive programming (changing the text input field to a dropdown dialog)

Maybe also add an option checkbox to color by sex

[TOPIC THREE]

Add some bells & whistles, like the regression model output to the screen with `tableOutput` or `textOutput` functions.

Introduce some error here, so it doesn't show up. Use `runApp(display.mode = "showcase")`

Fix the problem

Sharing is caring

Share the app via shinyapps.io. Acknowledge other sharing resources are in the Additional resources section.

Additional resources

- resource one
- The invaluable Shiny cheat sheet
- A PDF version of this lesson

Back to learn-r main page

Questions? e-mail me at jcoliver@email.arizona.edu.