Make a web application with Shiny!

Julin Maloof

Goal

- Make and deploy interactive web applications with Shiny.
- Shiny allows you to display your R analysis on the web to anyone. For example:
- I used Shiny to write an app to visualize a Markov Chain simulation of genetic drift.
- John used Shiny to develop a GUI for doing Genome Predictions for plant breeding.
- A student from a previous year of BIS180L worked in my lab and made a visualizer for QTL data and gene expression.
- The Shiny website has plenty of additional examples.

Components of a Shiny App

A ShinyApp consists of two R scripts:

- **ui.R** This script controls the user interface (i.e. the design of the webpage, the input and the output).
- **server.R** This script does the work of performing any analysis, creating graphs, and creating tables

These two scripts must be saved together in a single directory.

Alternative configuration

Alternatively, `ui.R` and `server.R` can be combined into a single script called **app.R**.

For this class we will use the two-script method.

The Shiny apps are group projects and having two separate scripts will make it easier to collaborate in our teams.

Show scripts

(Go to RStudio and show scripts)

Sending information between `ui.R` and `server.R`

User input: `trait` in `ui.R` can be accessed as `input$trait` in `server.R`

ui.R

```
radioButtons("trait", #the input variable that the value will go into
             "Choose a trait to display:", #title
             c("Sepal.Length","Sepal.Width",
               "Petal.Length","Petal.Width") ) #options
```

server.R

```
  output$boxPlot <- renderPlot({
    plotTrait <- as.name(input$trait) # convert user input to a name
        pl <- ggplot(data = iris,aes(x=Species,
                                     y= !! plotTrait,
                                     fill=Species))
        pl + geom_boxplot()
  })
```

Sending information between `ui.R` and `server.R`

output to UI: `output$boxPlot` in `server.R` is accessed as `boxPlot` in `ui.R`

ui.R

```
mainPanel(plotOutput("boxPlot"))
```

server.R

```
  output$boxPlot <- renderPlot({
    plotTrait <- as.name(input$trait) # convert user input to a name
        pl <- ggplot(data = iris,aes(x=Species,
                                     y= !! plotTrait,
                                     fill=Species))
        pl + geom_boxplot()
  })
```

Sending information between `ui.R` and `server.R`

Summary:

- In the `server.R` script the objects `output` and `input` contain the information going to and from `ui.R`
- In `server.R` each item must be accessed as an element of `input` or `output` (e.g. `input$trait`)
- In `ui.R` the element names are used directly (with quotes).

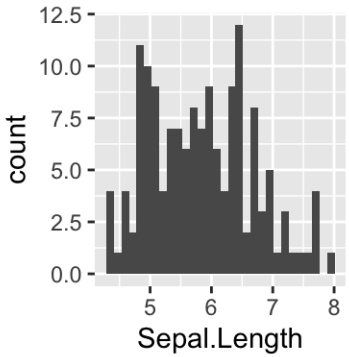
IMPORTANT: Force eval in ggplot with `!!`

ggplot `aes()` uses something called non-standard evaluation, which makes it easy to specify columns by name.

```
colnames(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

```
iris %>% ggplot(aes(x=Sepal.Length)) + geom_histogram()
```



IMPORTANT: Force eval in ggplot with !!

Unfortunately non-standard evaluation makes it difficult to specify the column indirectly.

For example, what if you have the column name specified in a variable?

This is exactly what is going to happen if you allow users to select traits in your Shiny app.

```
input <- list(trait="Sepal.Length")
input$trait
```

```
## [1] "Sepal.Length"
```

Since input
$traitcontains"Sepal.Length"wewouldhopethatRwouldsubstute"Sepal.Length"for`input`trait`

```
iris %>% ggplot(aes(x= input$trait)) + geom_histogram()
```

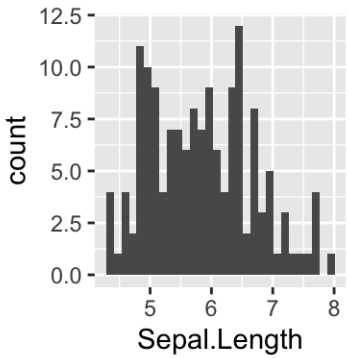
```
## Error: StatBin requires a continuous x variable: the x variable is discrete.Perhaps you want stat="count"?
```

IMPORTANT: Force eval in ggplot with `!!`

- The solution is to use `as.name()` and `!!`
- `as.name()` gets rid of the quotes around Sepal.Length and tells R that we want to use this as a name.
- `!!` Tells R to get the value of whatever follows it.

```
input <- list(trait="Sepal.Length")
plotTrait <- as.name(input$trait) # convert to a name

# use !! to substitute Sepal.Length for selected.column
iris %>% ggplot(aes(x= !! plotTrait )) + geom_histogram()
```



Keep your script fast

- Anything within a `renderNNN` statement will be run **every** time that the plot changes.
- So load data files and do one-time calculations at the **begining** of your server.R script.

server.R:

```
library(shiny)
library(ggplot2)

#load files and do one-time calculations here!

shinyServer(function(input, output) {
  output$boxPlot <- renderPlot({

  # Anything here gets re-run every time user input changes!
...
```

Running on your computer

To try the app on your computer, save the above scripts in ui.R and server.R, respectively, in a directory for this app.

- Click on the `RunApp` Button in R studio
- OR from the R console:

```
library(shiny)
runApp('PATH_TO_APP_DIRECTORY')
```

Sharing

Now that we have our awesome application how do we share it?

Multiple options:

If you are sharing it with someone that uses R and has the shiny library installed, then you can just send it to them, they can download it, and run it as above.

Sharing: GitHub

If you have it on GitHub and the person you want to share it with has R they can use:

```
library(shiny)
runGitHub(repo = "HamiltonDemos",username = "jnmaloof", subdir = "BinomialDrift")
```

Sharing: www.shinyapps.io

You can use Rstudio's **free** shiny server Once you have signed up for an account and authenticated, it is as simple as:

```
library(rsconnect)
rsconnect::deployApp('path/to/your/app')
```

You can see my version here

Sharing: set up your own server

If you are advanced you can run your own server

(I actually set up a server my lab–it isn't that hard)

Teams

We will work in our normal breakout room teams. Each team will produce and deploy a Shiny app that will be collectively graded.

Assignment

Your team should work together to create and deploy a ShinyApp that plots some aspect of the data from the BLAST or RICE labs, or from the tomato measurements data set that you used for the ggplot tutorial, available here. The app should allow user input that modifies the plot in a useful way.

I have listed some ideas below, **but feel free to choose something else**

RICE data ideas

You might want to limit the user input to 5 or 10 traits in the examples below, just to save yourself some typing and to keep the radio button list not too long

- Make an interacive version of any of the plots you made for Assignment 4
- Interactive plot showing histograms or violin plots or boxplots of user-selected phenotypic data split by ancestral population assignment or region.
    - You could also allow the user to choose whether it was a histogram or a violin plot.
    - You could allow the user to choose the binwidth for the histogram
- scatter plot of any two traits (user chosen), colored by the values of a third (user chosen).
- If you want to get fancy in either of the above then you could use the selectize tool to allow the user to select from all of the possible traits.

Other data ideas

- If any of you have a data set from your lab work you could use that.
- You could use the tomato data set that was used in the ggplot tutorial and explore relationship between altitude and plant height, or plot trait averages per species letting the user choose the trait, etc. (Link to data in webpage).

Scoring (out of 20 points)

16 points for a functional, interactive web app deployed on shinyapps.io and pushed to GitHub.

+ 2 points for using two or more input types (like a slide and a radio button).
+ 2 points for good annotation on the web page (a new user would understand what the app is about).
- 2 points for each student that does not make at least 2 commits to the team repository.