Lecture 13: Shiny

STAT 385 - James Balamuta

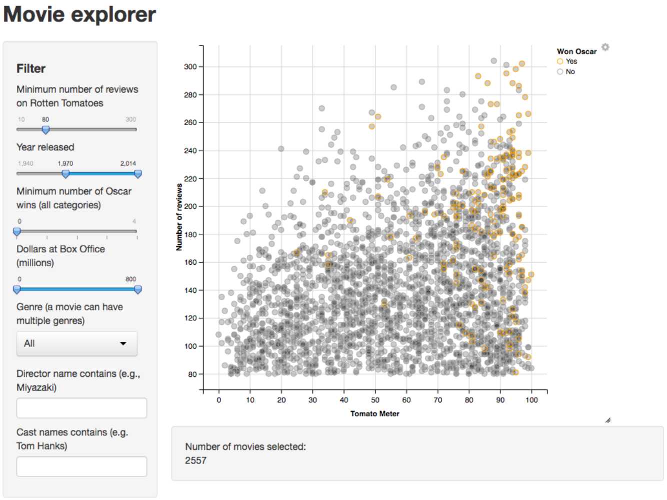
July 19, 2016

On the Agenda

- Administrative
  - Group Project Progress Reports are due tonight **Tuesday, July 19th at 11:59 PM CDT**.
- Shiny
  - Background information
  - Making an App
    - Frontend vs. Backend

What is Shiny?

Shiny is an R package that makes it easy to build interactive web applications (apps) straight from R.



Movie explorer

Filter

Minimum number of reviews on Rotten Tomatoes: 80

Year released: 1,970 – 2,014

Minimum number of Oscar wins (all categories): 0

Dollars at Box Office (millions): 0 – 800

Genre (a movie can have multiple genres): All

Director name contains (e.g., Miyazaki)

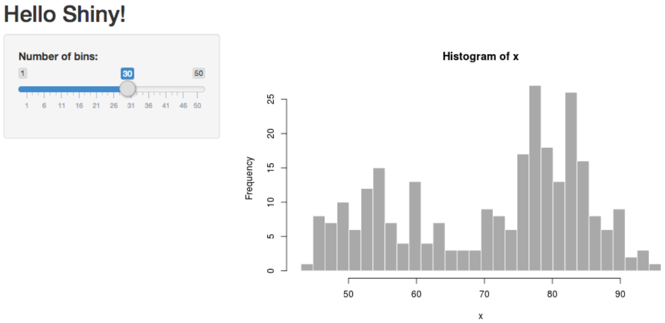
Cast names contains (e.g. Tom Hanks)

Number of movies selected:
2557

Why Shiny?

- Access features in the *R* ecosystem without knowing *R*!
- Standardized interactive explorations of data
- Easy deployments via:
  - **Local:** shiny::runApp()
    - development and package inclusion
  - **Server:** shiny-server
    - On premise use for companies
    - STATS@UIUC runs this on: rstudio.stat.illinois.edu/shiny
  - **Cloud:** shinyapps.io
    - Avoids management headaches and have easy access to scaling computational resources.

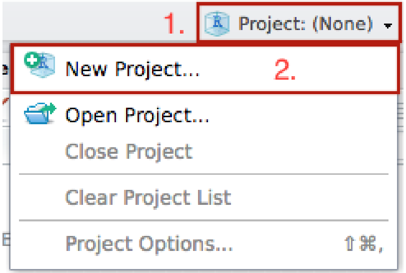
Hello Shiny World!



```r
# install.packages("shiny") # Install if on local
library(shiny)              # Load Shiny
runExample("01_hello")      # Run above example
```

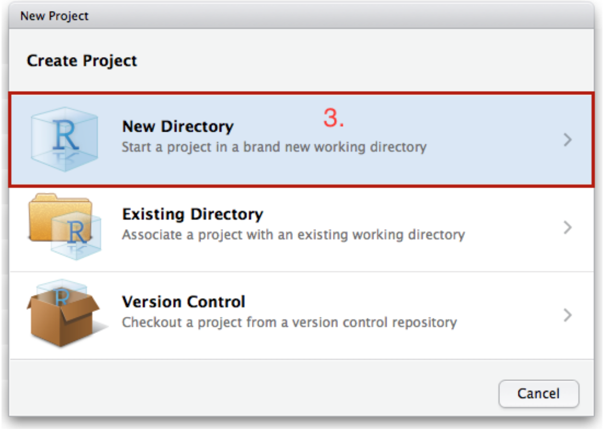
Setting up a Shiny Project - Dropdown Menu

- Select the project dropdown menu and press **New Project**



Setting up a Shiny Project - New Directory

- Select **New Directory**


New Project

Create Project

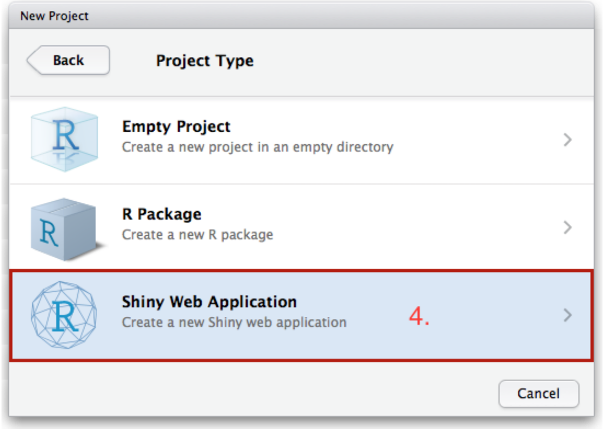
New Directory
Start a project in a brand new working directory

Existing Directory
Associate a project with an existing working directory

Version Control
Checkout a project from a version control repository

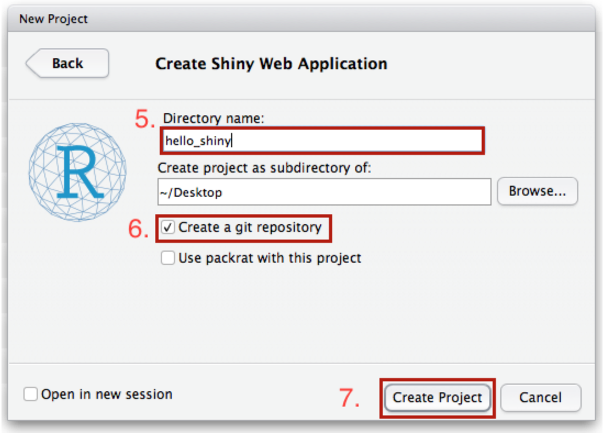
Setting up a Shiny Project - Project Type

- Select **Shiny Web Application**



Setting up a Shiny Project - Initialization Values

- Enter a project name (directory) for your shiny app.
- Check the **Create a git repository**
- Press **Create Project**



Directory name: hello_shiny

Create project as subdirectory of: ~/Desktop

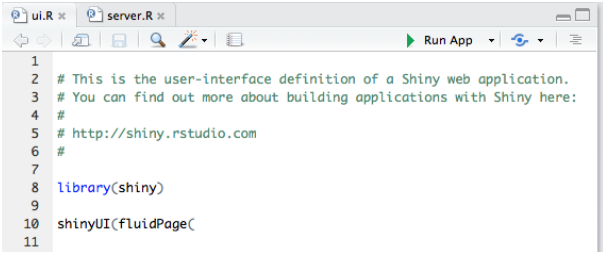
[✓] Create a git repository

[ ] Use packrat with this project

[ ] Open in new session

Exploring the Default Shiny App - Structure

- Once the project is created, an example shiny app is centerfold:



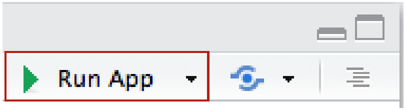
```r

# This is the user-interface definition of a Shiny web application.
# You can find out more about building applications with Shiny here:
#
# http://shiny.rstudio.com
#

library(shiny)

shinyUI(fluidPage(

```

Note: The presence of two files *ui.R* and *server.R*

Exploring the Default Shiny App - Running

To run a shiny within a project there are three options:

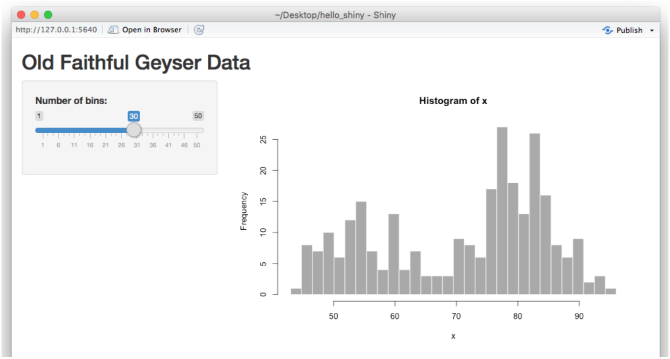
1. Type `runApp()` in **Console**
2. Use a keyboard shortcut
   - macOS: `Command` + `Shift` + `Enter`
   - Windows: `Control` + `Shift` + `Enter`
3. Press the `Run App` button at the upper right of the script editor.



Exploring the Default Shiny App - Live App

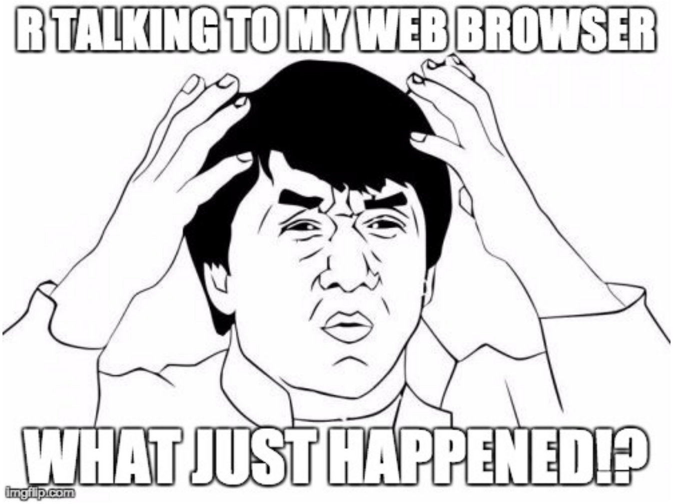
A secondary window will open and the Shiny app will be displayed.

- **Note:** Using RStudio on the analytical environment may require you to allow pop-ups!



Try moving the slider and comment to your group mates what happens to the histogram.

Lions, Tigers, and Bears. . . Oh my!



Behind the Scenes a Shiny App

As hinted to earlier, there are two files responsible for the creation of the shiny App: **ui.R** and **server.R**.

- **ui.R:** is responsible for providing the user interface (ui) for the shiny application.
- **server.R:** is responsible for providing the logic behind each change that occurs due to a button click, slider drag, et cetera on the UI front.

Behind the Scenes a Shiny App

The following is the bare minimum for a Shiny App to function.
ui.R

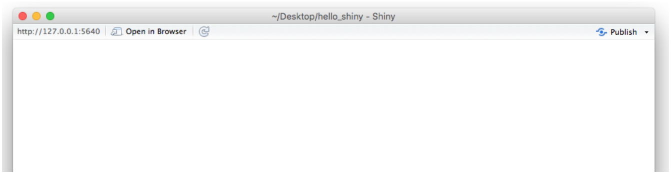
```
shinyUI(      # Initialize a UI container in Shiny
  fluidPage() # Make a page layout
)
```

server.R

```
shinyServer(                      # Initialize Server
  function(input, output) { # Input and output
  }
)
```

Blank Shiny

Note: Running the previous code will yield an empty app with a blank user-interface.



Beginning a Shiny App

- To motivatie our exploration of Shiny, we will create a shiny app that is able to *switch* between different datasets.
- We will begin by first constructing the User Interface (**ui.R**)
- Then we will write the backend logic (**server.R**)

Making Content

We can add content to the UI by using:

Function	Description
`titlePanel()`	Naming the application (e.g. Hello Shiny!)
`sidebarLayout()`	Creates a sidebar layout for the `fluidPage()`.
`sidebarPanel()`	Makes a side bar **menu** for UI Controls and Instruct
`mainPanel()`	Main content area to house graphs, tables, text outp

Making Content for the Interface

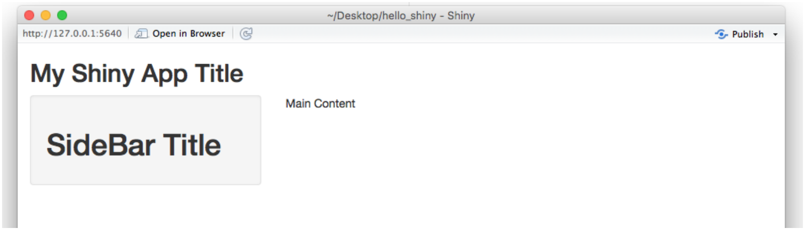
ui.R

```r
shinyUI(
  fluidPage(
    titlePanel("My Shiny App Title"), # Title

    sidebarLayout(

      sidebarPanel(
        h1("SideBar Title")               # Sidebar Text
        ),                                # Note HTML

      mainPanel("Main Content")           # Content Text
    )
  )
)
```

Note: You can use attributes such as `align = "center"` by `h1("SideBar Title", align = "Center")`

Making Content for the Interface - Preview

If we run our app, we will get:



My Shiny App Title

SideBar Title

Main Content

HTML in Shiny

Function	HTML	Description
`strong()`	`<strong></strong>`	Bold Text
`em()`	`<em></em>`	Italicize Text
`a()`	`<a></a>`	Makes a hyperlink
`p()`	`<p></p>`	Text Paragraph
`h1()`	`<h1></h1>`	Header (replace 1)
`br()`	`<br />`	Creates a page break
`div()`	`<div></div>`	Division of text
`span()`	`<span></span>`	Inline division of text
`pre()`	`<pre></pre>`	'as is' text field
`code()`	`<code></code>`	Code formated block
`HTML()`	-	Embed own HTML Code

Note: `h2()` up to `h6()` provides different heading styles.

- **More Shiny HTML Tags...** (About 110 of them!)
- **UI Customization with HTML**

Making Inputs

- Create HTML from within R is nice, but we want to be able to talk to R.
- To do that, we must make some sort of input control.
- In Shiny, the input control comes from *widgets*

Making Widgets for Input

To construct a **widget**, we must:

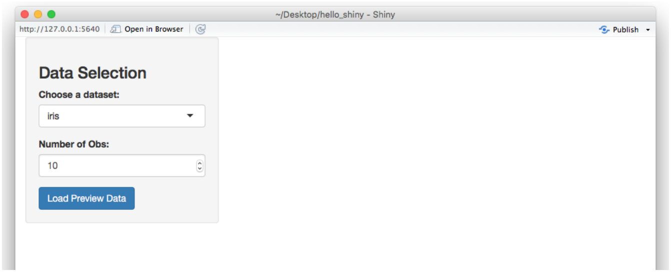
- Provide a `name=""`
  - We will use this to get the active value.
  - Users will not be able to see the name.
- Provide a `label=""`
  - This describes the widget to the user.

Making Widgets for Input - Example

ui.R

```
sidebarLayout(
  sidebarPanel(
    h3("Data Selection"),            # Note the ,

    # Dropdown
    selectInput("ds",                # Name
                "Choose a dataset:",  # Label
                choices = c("iris", "Spam", "mtcars")),

    numericInput("obs",              # Name
                 "Number of Obs:",    # Label
                 10),                 # Default Value

    submitButton("Load Preview Data") # Update data
  ),
  mainPanel())# Not Displayed         # Content
```

Making Widgets for Input - Preview



Data Selection

Choose a dataset: iris

Number of Obs: 10

UI Input Controls

Shiny features a lot of different ways to accept user input

Function	Description
`numericInput()`	Number entry input
`radioButtons()`	Radio button selection
`selectInput()`	Dropdown menu
`sliderInput()`	Range slider (1/2 values)
`submitButton()`	Submission button
`textInput()`	Text input box
`checkboxInput()`	Single checkbox input
`dateInput()`	Date Selection input
`fileInput()`	Upload a file to Shiny
`helpText()`	Describe input field

See **Shiny Widgets Gallery** for examples.

Making Render UI Areas

- So far, we have managed to make stylistic features and input controls.
- However, in order for the *Shiny* app to be dynamic and display data, we must have output control or render areas.
- To do so:
  1. We add an output control to **ui.R**.
  2. Make some logic in **server.R** to talk with it! (Yes, we're almost there.)

Making Render UI Areas - Example

```r
sidebarLayout(
  sidebarPanel(), # Given previously
  mainPanel(
    h3("Head of the Dataset"),    # HTML
    tableOutput("view"),          # Table View

    h3("Dataset Summary"),        # HTML
    verbatimTextOutput("summary") # Output Asis
  )
)
```

Note: Like the input control, we do *name* the output values.

UI Output Controls

There are many ways to render the results

Function	Description
`plotOutput()`	Display a rendered plot
`tableOutput()`	Display in Table
`textOutput()`	Formatted Text Output
`uiOutput()`	Dynamic UI Elements
`verbatimTextOutput()`	“as is” Text Output
`imageOutput()`	Render an Image
`htmlOutput()`	Render Pure HTML

Also see:

- **Dyanmically Generated User Interface Components**
- **Changing the Values of Inputs from the Server**

Moving over to **server.R**

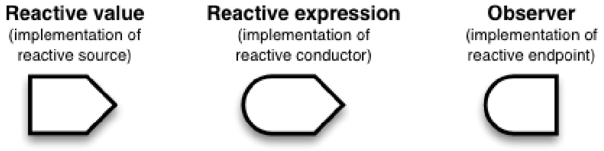
- We've finished what we needed to accomplish in the **ui.R** file.
- Now, we must write the backend logic in **server.R**.

What is Reactivity?

"For every action, there is an equal and opposite reaction."
– Issac Newton

What is Reactivity?

- **Reactive Sources (Reactive Values)**
  - UI element inputs
- **Reactive Conductors (Reactive Expressions)**
  - Server Catches for UI elements `reactive({})`
- **Reactive Endpoints (Observers)**
  - Render functions in the UI and `observer({})` in Server


Reactive value
(implementation of reactive source)
Reactive expression
(implementation of reactive conductor)
Observer
(implementation of reactive endpoint)

View **Reactivity Explanation**

Note: Reactive expressions return values, but observers don't.

Creating a Reactive Catch

server.R

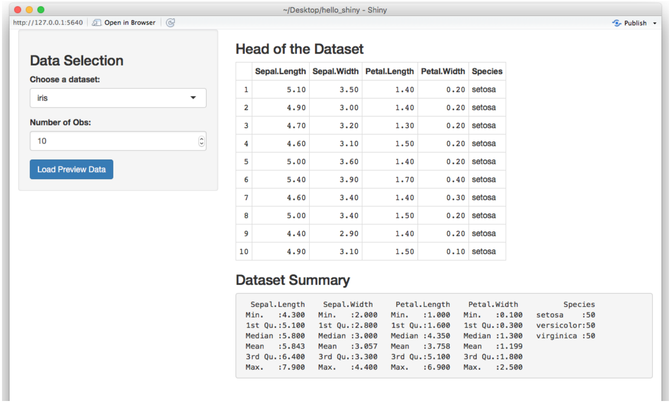
```r
library("msos"); library("dataset")
data("Spam")
shinyServer(function(input, output) {

  dsInput = reactive({   # Reactive
    switch(input$ds,     # Load dataset
           "iris" = iris,
           "Spam" = Spam,
           "mtcars" = mtcars)
  })

})
```

Creating Output Hooks

server.R

```r
shinyServer(function(input, output) {

  ## Hiding data set reactive

  output$summary = renderPrint({   # Summary Render
    summary(dsInput())
  })

  output$view = renderTable({      # Table Render
    head(dsInput(), n = input$obs)
  })
})
```

Creating Observer Hooks - Preview



Data Selection

Choose a dataset: iris

Number of Obs: 10

Load Preview Data

Head of the Dataset

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.10	3.50	1.40	0.20	setosa
2	4.90	3.00	1.40	0.20	setosa
3	4.70	3.20	1.30	0.20	setosa
4	4.60	3.10	1.50	0.20	setosa
5	5.00	3.60	1.40	0.20	setosa
6	5.40	3.90	1.70	0.40	setosa
7	4.60	3.40	1.40	0.30	setosa
8	5.00	3.40	1.50	0.20	setosa
9	4.40	2.90	1.40	0.20	setosa
10	4.90	3.10	1.50	0.10	setosa

Dataset Summary

```
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species  
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50  
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50  
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50  
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199                  
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800                  
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500                  
```

Displaying Reactivity

The functions below are meant to interface with the `*Output()` UI functions.

Function	Description
`renderPlot()`	Display Plots
`renderPrint()`	Output Print (Verbatim)
`renderTable()`	Tables for 2D Data Structures
`renderText()`	Display Character Strings
`renderUI()`	Dynamic UI render
`renderImage()`	Saved Images on Disk

Understanding Shiny Runtime Components

Shiny runtime components is slightly different than normal. Certain areas of the **server.R** are either run:

- Once on startup
  - Initializing the application on server
- Once per user visit
  - Loading user info
- Many times per session
  - Reactive control

Understanding Shiny Runtime Components - Startup

server.R

```r
load("data.rda")                # Once during startup

shinyServer(                    # Once during startup

  function(input, output) {
    toad = "Hello"

    output$test = renderUI({

    })
  }
)
```

Understanding Shiny Runtime Components - User Session

server.R

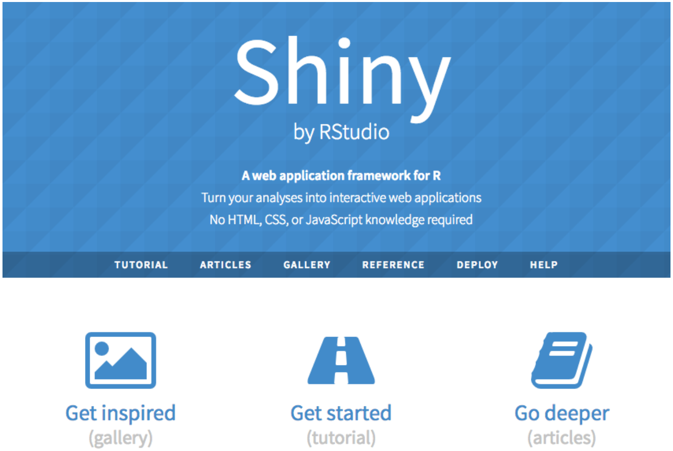
```r
load("data.rda")

shinyServer(

  function(input, output) {  # Once per user
    toad = "Hello"

    output$test = renderUI({

    })
  }
)
```

Understanding Shiny Runtime Components - Actions

server.R

```
load("data.rda")

shinyServer(

  function(input, output) {
    toad = "Hello"

    output$test = renderUI({ # Many Times

    })
  }
)
```

Resources for Shiny


Shiny
by RStudio

A web application framework for R
Turn your analyses into interactive web applications
No HTML, CSS, or JavaScript knowledge required

Shiny Page - **Real Live Apps** - **Video** and **Written Tutorials**

More Resources for Shiny

- Shiny on Github
- Shiny Development Mailing List
- Shiny Function Reference

Acknowledgement

This lecture goes into depth about the Shiny More Widgets Example on Shiny Gallery