R SHINY

CALVIN SKALLA

JUSTIN KELANA

WHAT IS R SHINY?

- R shiny is a web application framework in R, and its main purpose is to enable R users to create interactive webpages with ease.
- All you need to do is download the Shiny package in R, and with the built-in functions in the package you could build a webpage with your computer as the server.
- What is nice about the shiny package is that it doesn't require you to know how to code in HTML.

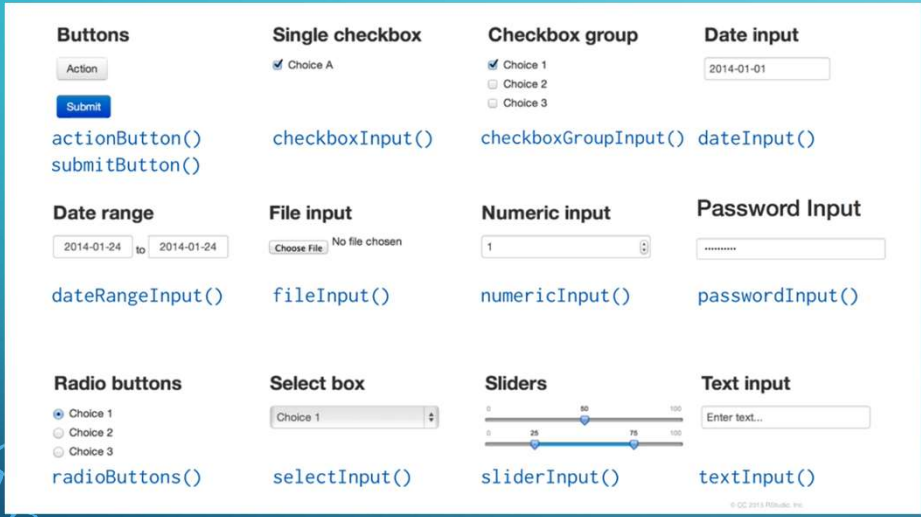
ARCHITECTURE OF A SHINY APP

USER INTERFACE (UI) SERVER

```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

USER INTERFACE

Input



Output

Function	Inserts
dataTableOutput()	an interactive table
htmlOutput()	raw HTML
imageOutput()	image
plotOutput()	plot
tableOutput()	table
textOutput()	text
uiOutput()	a Shiny UI element
verbatimTextOutput()	text

BASIC SYNTAX

```
...Input(inputId = "x", label = "this is a basic input", ....)

...Output(outputId = "y", ...)
```

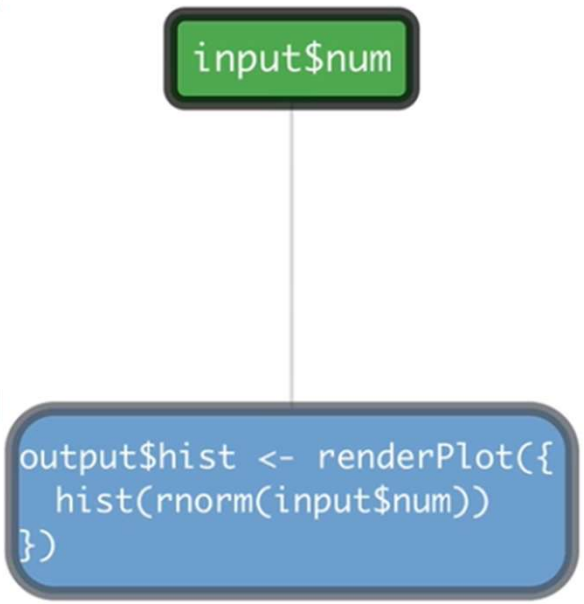
- All output and Input functions are common in that they need to be assigned a name.
- When you create an input or output function you will have to use its assigned name to call on its value when coding on the server side by using either

input$inputid output$outputid

REACTIVE VALUES

1 **Reactive values notify** the functions that use them when they become invalid

2 The objects created by **reactive functions respond** (different objects respond differently)



TAGS

Function	Creates
a()	A Hyperlink
br()	A line break
code()	Text formatted like computer code
em()	Italicized (emphasized) text
h1(),h2(),h3(),h4(),h5(),h6()	Headers (First level to sixth)
hr()	A horizontal rule (line)
img()	An image
p()	A new paragraph
strong()	Bold (strong) text

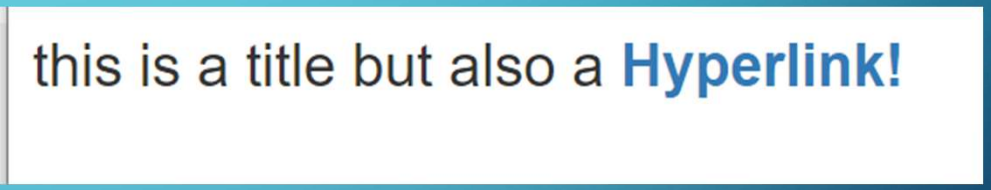

```
> tags$h1("Hello")
<h1>Hello</h1>
```

The table above shows commonly used tags and you could access the full list using names(tags)

NESTING

```r
```{r}
ui <- fluidPage(
  tags$h1("this is a title but also a ", tags$strong(tags$a(href =
"www.google.com", "Hyperlink!"))))

)
```


```

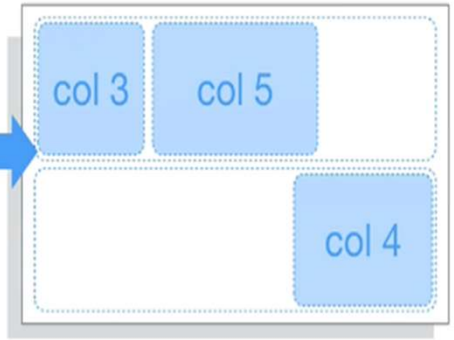
LAYOUTS

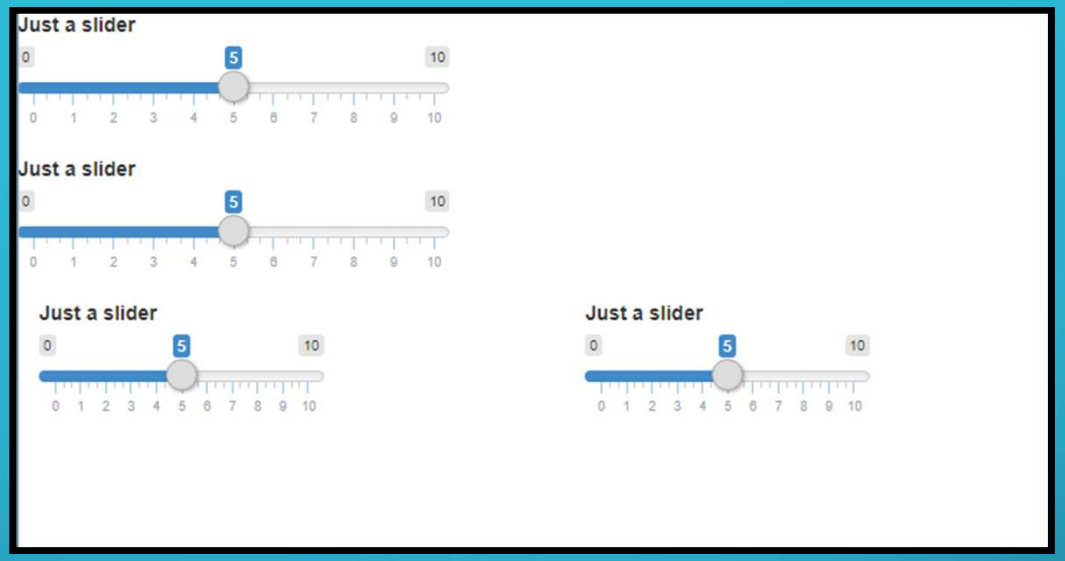
`fluidRow()` adds rows to the grid. Each new row goes below the previous rows.

```
ui <- fluidPage(
  fluidRow(),
  fluidRow()
)
```

Specify the **width** and **offset** of each column out of 12

```
ui <- fluidPage(
  fluidRow(
   column(3),
   column(5)),
  fluidRow(
   column(4, offset = 8)
)
```






```
ui <- fluidPage(
fluidRow(sliderInput("slider", label = "Just a slider", min = 0, max = 10, value = 5),
         sliderInput("slider", label = "Just a slider", min = 0, max = 10, value = 5)),


fluidRow(column(width = 3, sliderInput("slider", label = "Just a slider", min = 0, max = 10, value = 5)),
         column(width = 3, offset = 2, sliderInput("slider", label = "Just a slider", min = 0, max = 10, value = 5)))

  )
```

PANELS

absolutePanel()
Panel position set rigidly (absolutely), not fluidly

conditionalPanel()
A JavaScript expression determines whether panel is visible or not.

fixedPanel()
Panel is fixed to browser window and does not scroll with the page

headerPanel()
Panel for the app's title, used with pageWithSidebar()

inputPanel()
Panel with grey background, suitable for grouping inputs

mainPanel()
Panel for displaying output, used with pageWithSidebar()

navlistPanel()
Panel for displaying multiple stacked tabPanels(). Uses sidebar navigation

sidebarPanel()
Panel for displaying a sidebar of inputs, used with pageWithSidebar()

tabPanel()
Stackable panel. Used with navlistPanel() and tabsetPanel()

tabsetPanel()
Panel for displaying multiple stacked tabPanels(). Uses tab navigation

titlePanel()
Panel for the app's title, used with pageWithSidebar()

wellPanel()
Panel with grey background.

WRITING THE SERVER FUNCTION IN R

```
#Server Template
```{r}
# Define server logic required to draw a output ----
server <- function(input, output) {

  #Use a reactive expression when the same output functions will be used
for different data
  dataInput <- reactive({

    #Encode the different data frame names into values that can be ran
through the render function

  })


  # Keep expression wrapped in a call
  # to renderPlot to indicate that:
  #
  # 1. It is "reactive" and therefore should be automatically
  #    re-executed when inputs (input$bins) change
  # 2. Its output type is a plot

  output$outputId <- renderPlot({
  #output$outputID is derived from the Output intiated in the UI

    #Fill in the plot information wanted to display on the UI

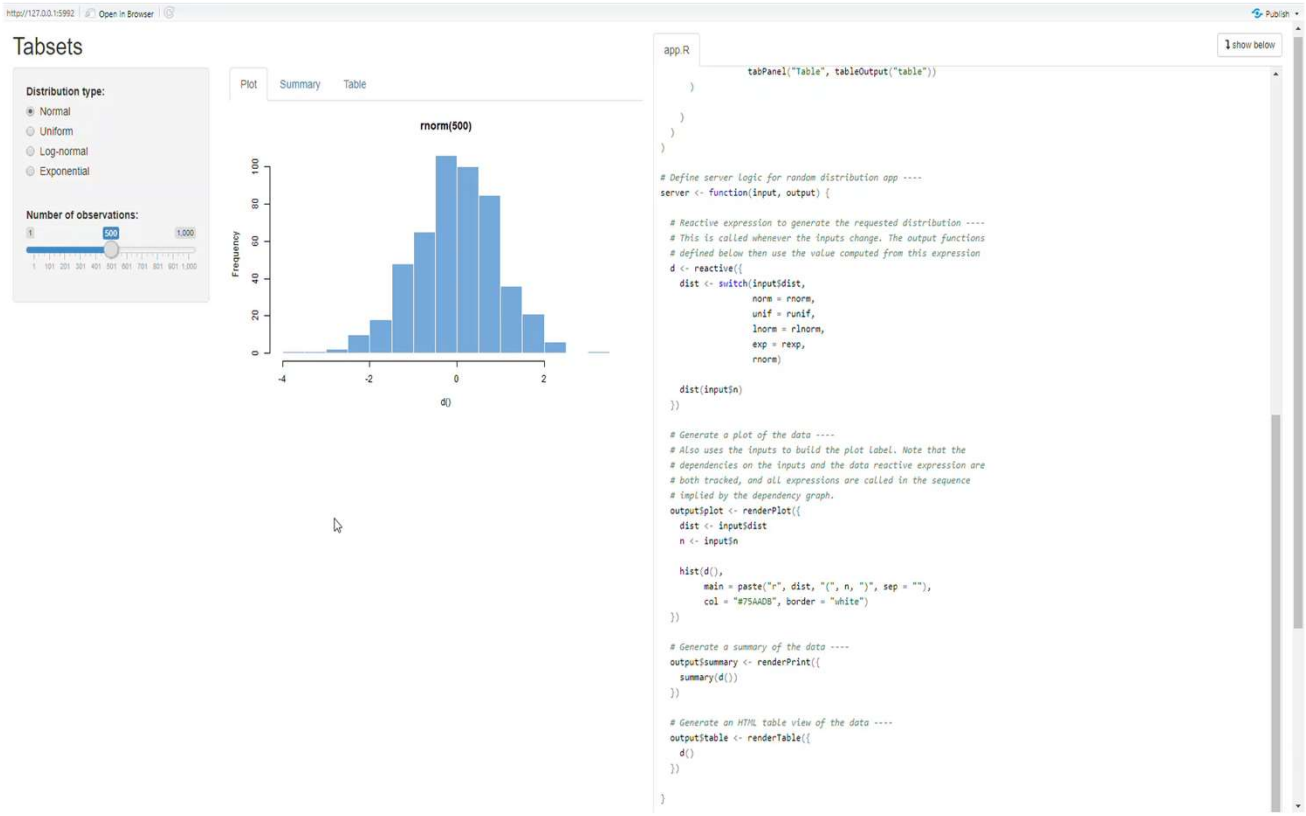
    })

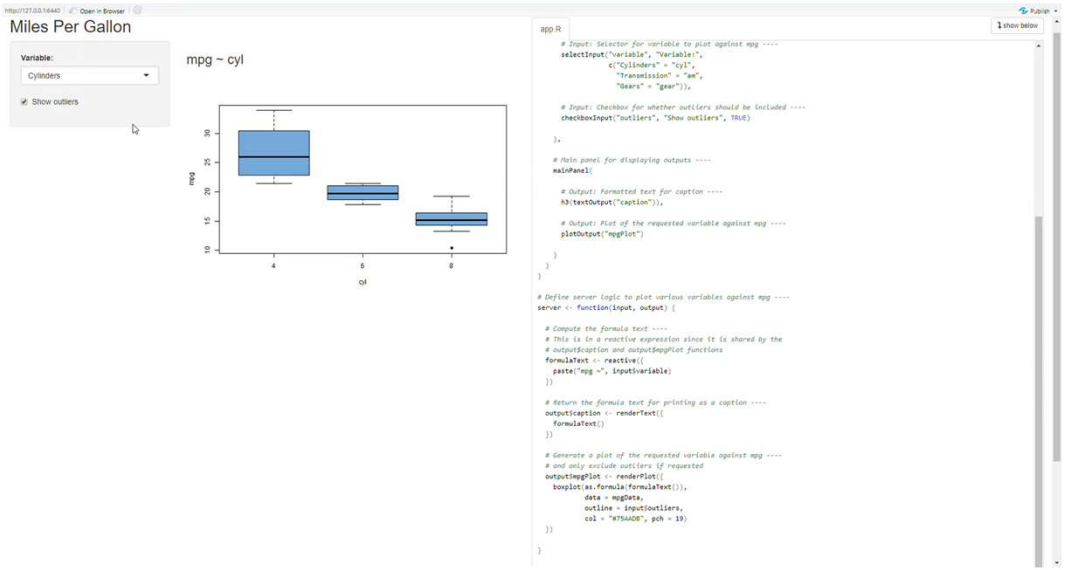
}
```
```

- Takes an input and output to know what data needs to be displayed on the User Interface
- Give reactive() and/or render() functions for the values to output onto the User Interface

REACTIVE FUNCTION

- Using different data requires shiny to react and produce plots with the desired data
- Does not produce output until desired data is selected



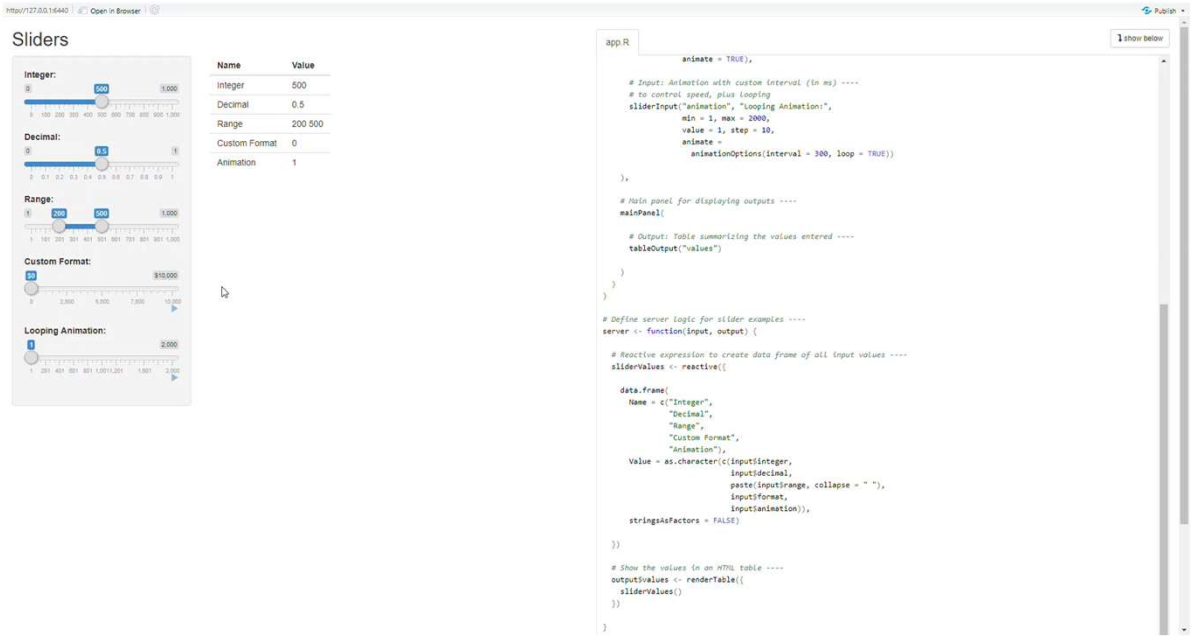


REACTIVE FUNCTION

USING PASTE

REACTIVE FUNCTION

USING DATAFRAME



RENDER FUNCTION

- Function for interaction on the application
- Automatically adjusts when the reactive information is changed

Output function	render function	creates
htmlOutput/uiOutput	renderUI	a Shiny tag object or HTML
imageOutput	renderImage	images (saved as a link to a source file)
plotOutput	renderPlot	plots
tableOutput	renderTable	data frame, matrix, other table like structures
textOutput	renderText	character strings
verbatimTextOutput	renderPrint	any printed output

OPTIMIZING YOUR APPLICATION – RSTUDIO PRO

SLOW TO LOAD, FAST AFTER LOADING

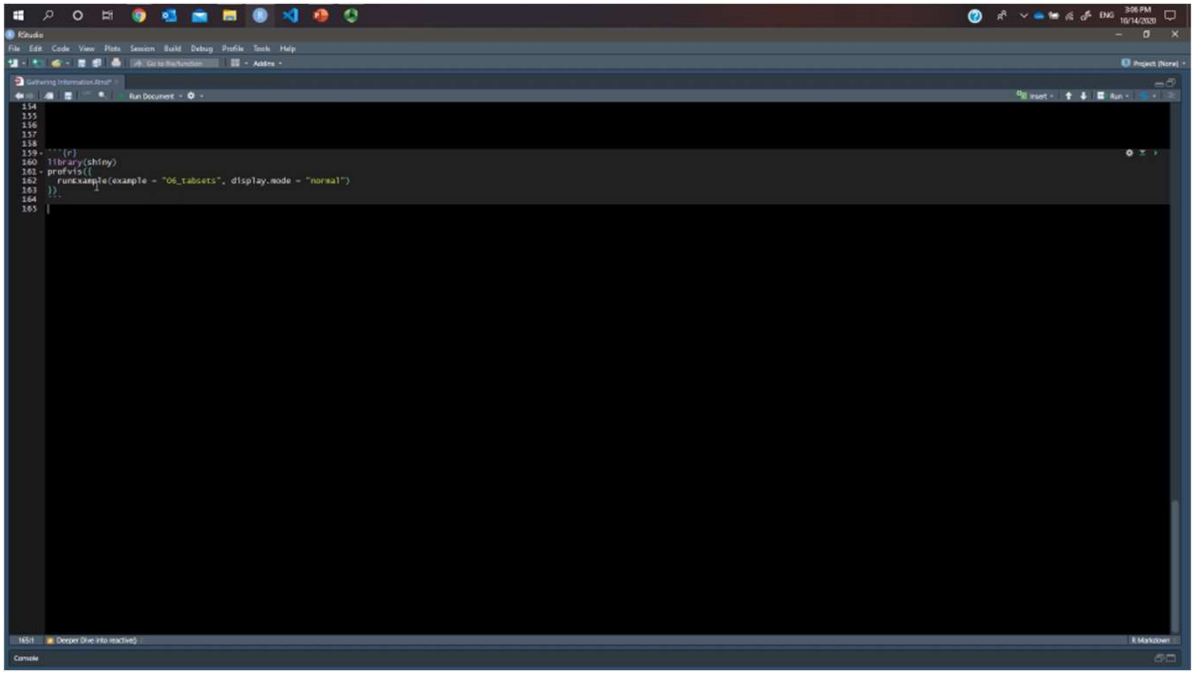
- Using large data set
- Set loadFactor closer to 1
- Have more users connected to the same R process that is powering the shiny application

FAST TO LOAD, SLOW AFTER LOADING

- Smaller data sets, more computations
- Set a small loadFactor
- Fewer connections to each R process by spawning additional R processes more aggressively

PROFILING YOUR R CODE

- Runs application and presents the interface
- Use some of the features to track how long they take
- Exit out of the application after testing some of the reactive and render functions
- R automatically displays the code's profile with a flame graph



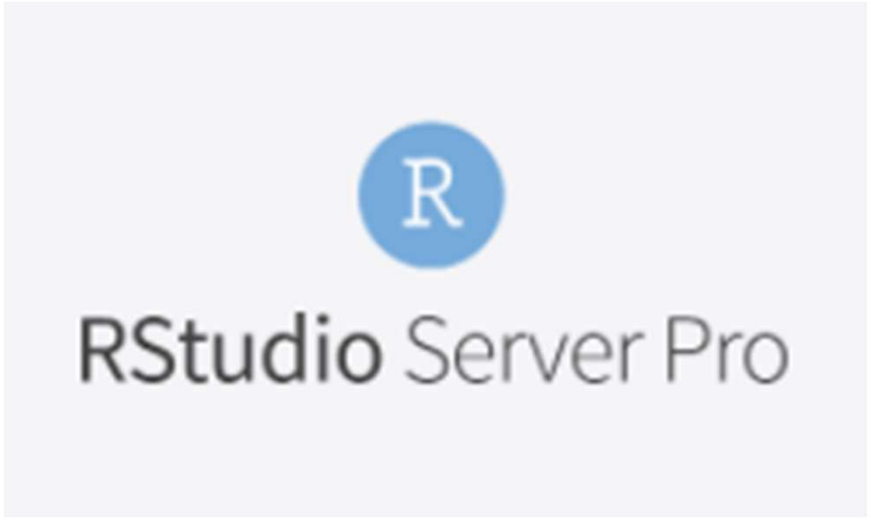
Product	Installed or Hosted	Push Button Publishing	Shiny Apps	R Markdown Documents	Python Content (Jupyter, Flask, Dash)	Plumber API's
RStudio Connect	Installed	•	•	• *(1)*	• *(1)*	•
Shiny Server Pro	Installed		•	•		
Shinyapps.io	Hosted	•	•	• *(2)*		

SERVER HOSTING

SHINY SERVER – OPEN SOURCE

FREE VERSION WITH LEAST AMOUNT OF FEATURES

Category	Description	Open Source Edition
Overview	Access the RStudio IDE from anywhere via a web browser	✓
	Move computation close to the data	✓
	Scale compute and RAM centrally	✓
	Powerful coding tools to enhance your productivity	✓
	Easily publish apps and reports	✓



Consider Shiny Server Pro if you can answer yes to all of these questions:

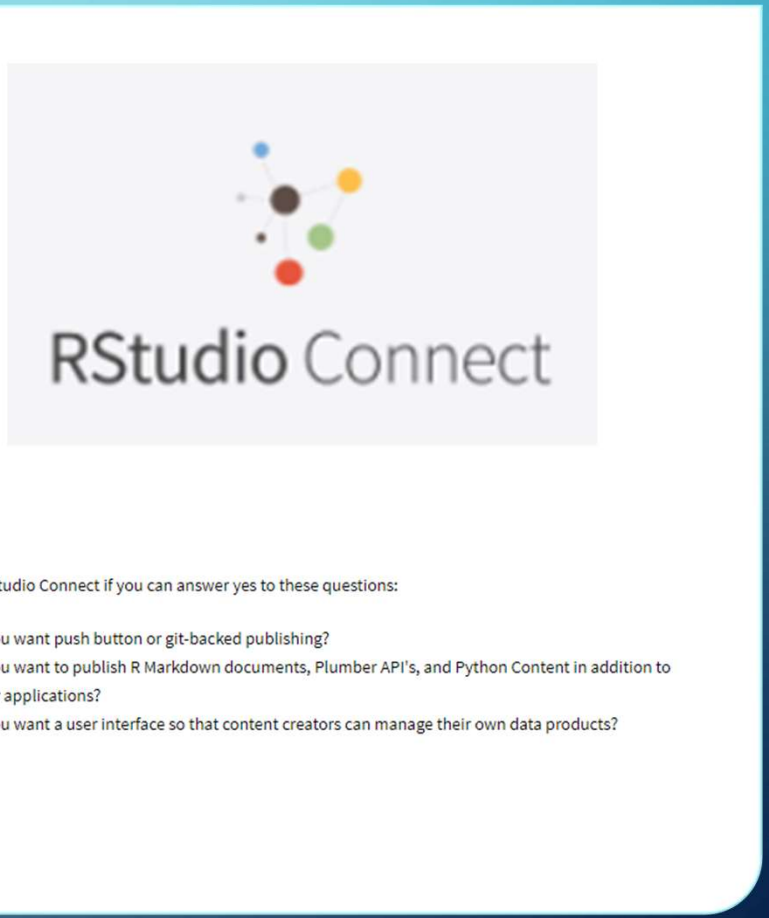
1. Do you want to deploy apps without push button publishing?
2. Do you want to manage apps with configuration files rather than a UI?
3. Do you feel comfortable manually managing and updating the R packages used by your apps?
4. Are you only interested in sharing Shiny and apps (and not other data products)?
5. Do you want concurrent user licensing?

RSTUDIO SERVER PRO

PAID VERSION THAT OFFERS TWO DIFFERENT TIERS

RSTUDIO CONNECT

FREE VERSION OR TWO DIFFERENT TIERS OF PAID VERSIONS REGARDING NECESSITIES



Use Shinyapps.io if you can answer yes to all of these questions:

1. Are you okay with your application being outside your firewall?
2. Are you okay with the data that the application is pulling from being accessible to our cloud? (You have to open up a hole in your firewall if the data is behind the firewall today.)
3. Are you okay with your end client creating a user account on shinyapps.io (if you are looking to use authentication).
4. Are you okay with a shared computation platform for your analyses? (for example, we don't have any SLAs today on performance)

SHINYAPPS.IO

MORE AVAILABLE OPTIONS DEPENDING ON WHAT IS NEEDED TO HOST YOUR APPLICATION(S)

HOSTING & RUNNING ON GITHUB

- Save UI and Server code within one R file titled “app.R” onto desired repository

```
runGitHub( "<your repository name>", "<your user name>")
```

COMPLEX EXAMPLE FROM THE SHINY GALLERY

BIODIVERSITY IN NATIONAL PARKS

HTTPS://ABENEDETTI.SHINYAPPS.IO/BIONPS/?_GA=2.186502380.410627015.1602709874-922484645.1568241926

HOMEWORK!

For the homework we'd like you guys to make your own shiny app to display a histogram made from randomly generated exponential values. A user should be able to set the mean of the exponential distribution, the sample size, and the color of the bins on your histogram.

When a user changes the specified value for mean and sample size, the app should update the data used to create the histogram. When the user changes the specified color, the output should update but your app should not generate a new dataset.

Shiny Homework

Sample Size

30

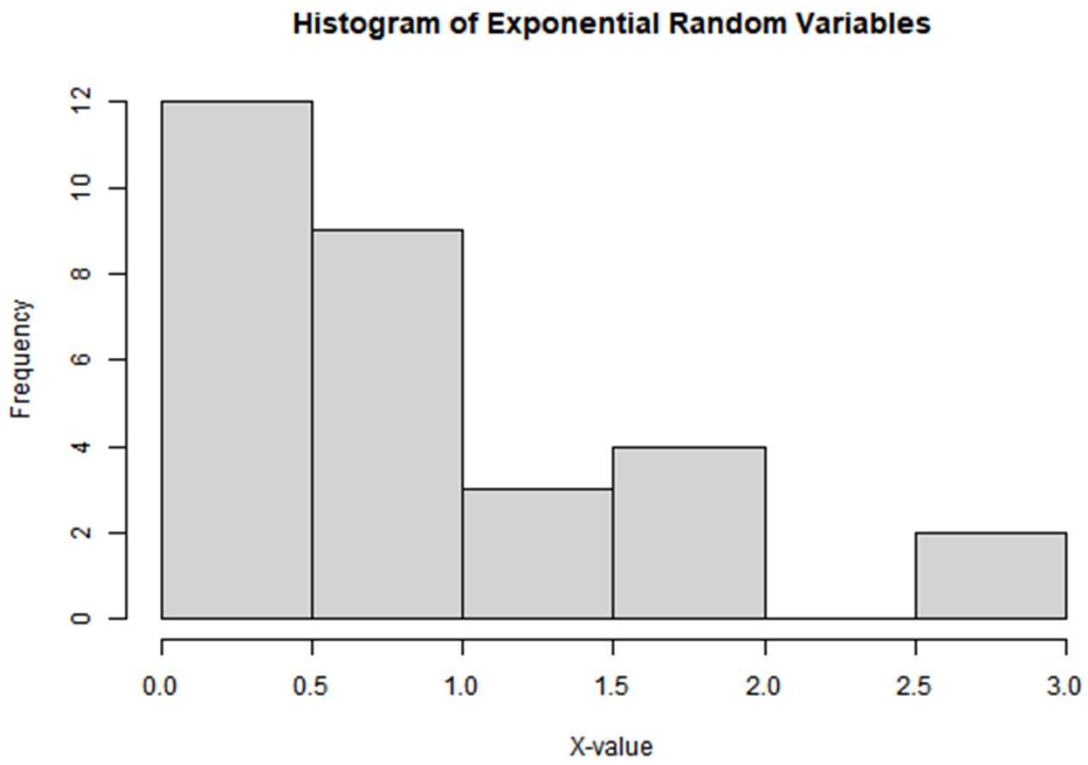
Mean of Exponential

1

color of bins

default

- default
- red
- blue
- green



HOMEWORK HINT

Your app should have a reactive function to generate a new dataset every-time the user updates the mean and sample size.

Your script to make the histogram should be reactive to two things:

the dataset (that is your app will generate a new histogram whenever a new dataset is created) and the input value for color.

```r
library(shiny)

ui <- fluidPage(
  titlePanel("Shiny Homework"),
  sidebarPanel(
    #input for sample size, initialize value at 30 and min = 0
     ...Input(inputid = "n", ......),

     #input for mean, intialize value at 1 and min = 0
    ...Input(inputid = "mean", .......),

    #drop-down input for color
    ...Input(inputid = "color", "color of bins", choices = c( "default" = "lightgray", "red" = "red", "blue" = "blue", "green" = "green") )),

mainPanel(plotOutput("hist"))

)

server = function(input, output){
  #generate your data here
  data <- ...({

  })

  # generate the function to create your histogram here.
  output$hist <- ....({

    hist( ... , ylab = "Frequency", xlab = "X-value", main = "Histogram of Exponential Random Variables", ... )


  })

}

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

REFERENCES

1. https://shiny.rstudio.com/articles/basics.html
2. https://shiny.rstudio.com/articles/shiny-server.html
3. https://github.com/rstudio/shiny-examples
4. https://rstudio.com/products/shiny/shiny-server/
5. https://bookdown.org/weicheng/shinyTutorial/getting-started.html
6. https://rstudio.github.io/profvis/examples.html?_ga=2.50765739.367297665.1602538216-922484645.1568241926#example-3---profiling-a-shiny-application
7. https://lukesingham.com/shiny-r-performance-profiling/


QUESTIONS!