

STOR 320.1

Shiny R

Introduction

- Web Applications with [R Shiny](#)
- Requires the [Shiny Package](#) in R
- Check Out R Shiny [Cheat Sheet](#)
- [Gallery](#) of Shiny Applications
- Deployable by shinyapps.io



Introduction

- Planning What You Want to Do
 - User Controls _____
 - Output Given is _____
 - R Code I Need is _____



Getting Started

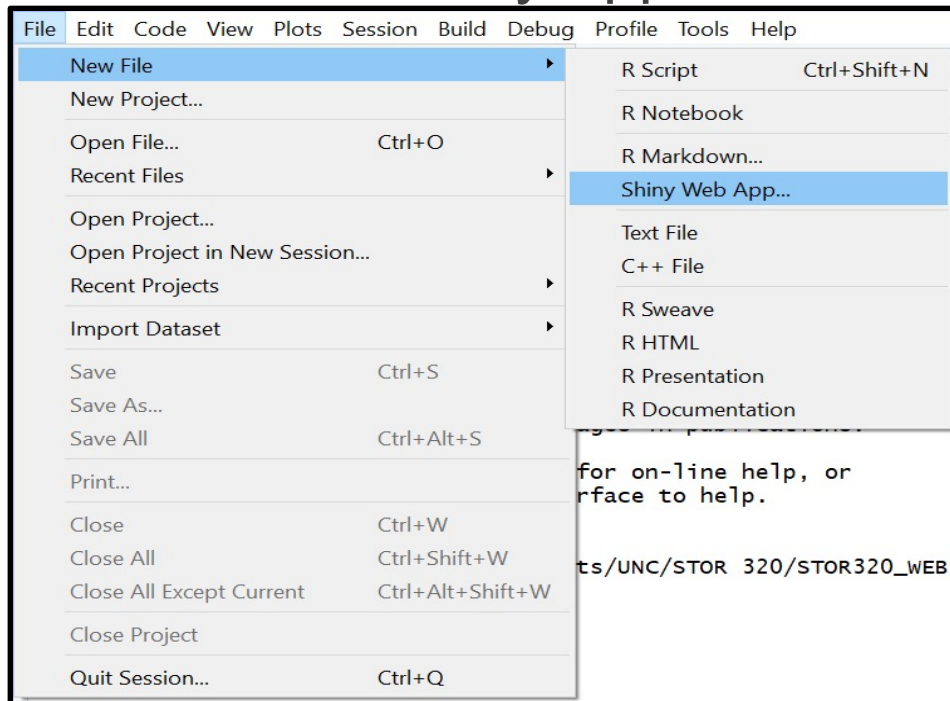
- Step 1: Install Shiny Package

```
> install.packages("shiny")
```

- Step 2: Load the Library

```
> library(shiny)
```


- Step 3: Create a New Shiny App



Getting Started

- Step 4: Initiate Your Shiny App

New Shiny Web Application

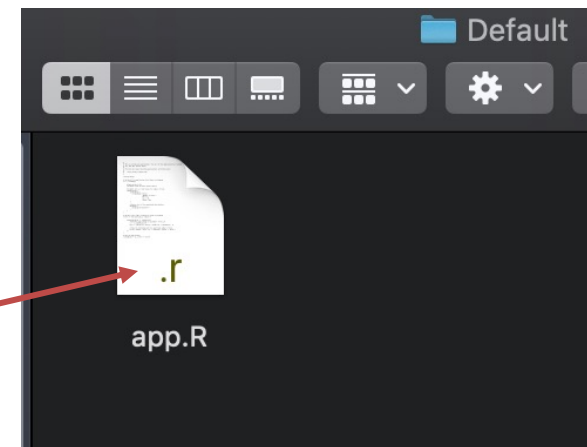


Application name:

Application type:
 Single File (app.R)
 Multiple File (ui.R/server.R)

Create within directory:

[? Shiny Web Applications](#)

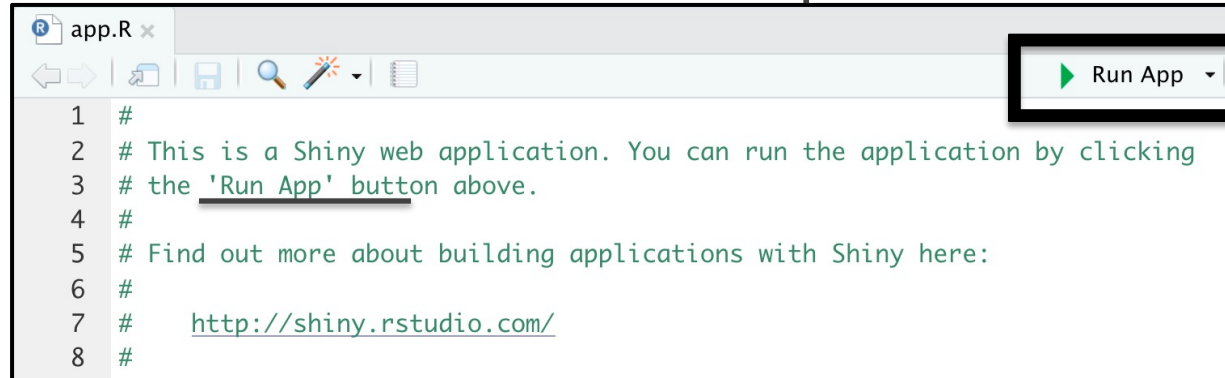


Name	Date Modified	Size	Kind
▶ Default	Yesterday at 9:58 PM	--	Folder
▶ Gapminder_Part1	Yesterday at 9:58 PM	--	Folder
▶ Gapminder_Part2	Yesterday at 9:58 PM	--	Folder
▶ Gapminder_Part3	Yesterday at 9:58 PM	--	Folder
▶ Gapminder_Start	Yesterday at 9:58 PM	--	Folder
▶ Previous	Yesterday at 9:58 PM	--	Folder
▶ Tutorial.zip	Jul 25, 2020 at 10:10 AM	9 KB	ZIP archi

iCloud Drive > Docume > ACADEM > Academ > 2020FA1 > STOR32 > Tutorial > 14 R Shiny

Example

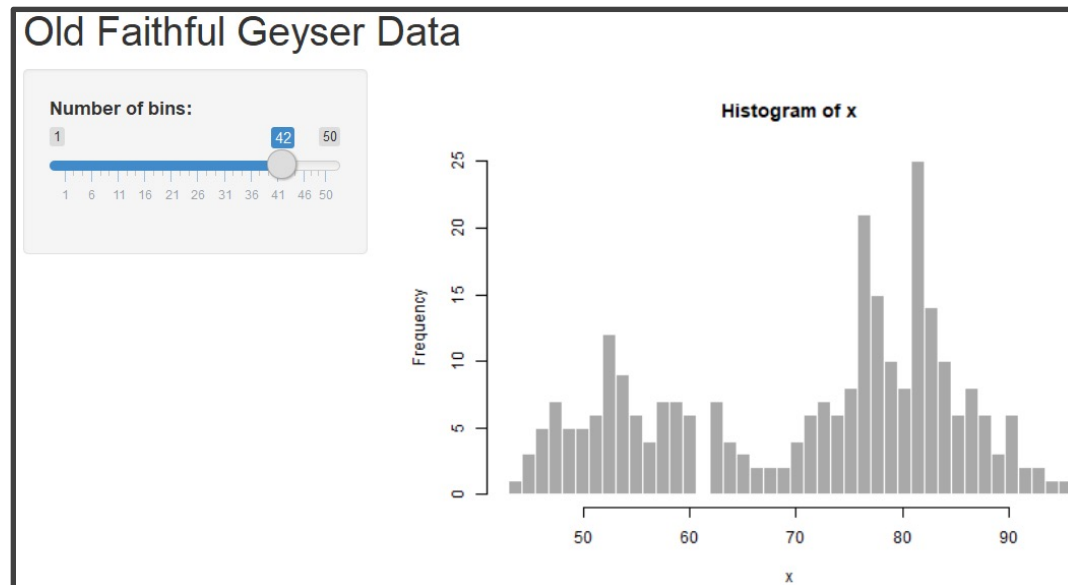
- Step 5: Run the Shiny App



```
1 #
2 # This is a Shiny web application. You can run the application by clicking
3 # the 'Run App' button above.
4 #
5 # Find out more about building applications with Shiny here:
6 #
7 # http://shiny.rstudio.com/
8 #
```



- Step 6: Play With the App

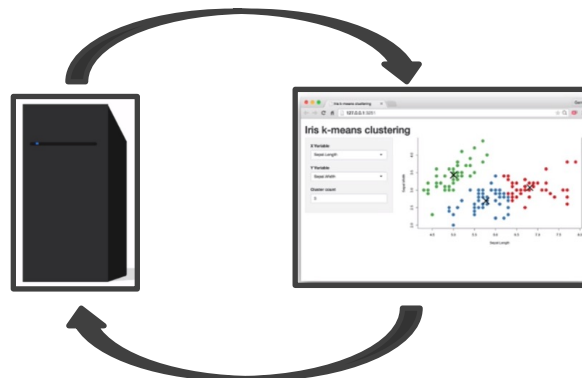


Architecture

- How it Works?
 - Communication Between Your Computer and Your App



- Sharing Through the Cloud From a Web Based Server



Architecture: Example

```
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")  
    )  
  )  
)
```

```
> ui  
<div class="container-fluid">  
  <h2>Old Faithful Geyser Data</h2>  
  <div class="row">  
    <div class="col-sm-4">  
      <form class="well">  
        <div class="form-group shiny-input-container">  
          <label class="control-label" for="bins">Number of bins:</label>  
          <input class="js-range-slider" id="bins" data-min="1" data-max="50" data-from="30" data-step="1" data-grid="true" data-grid-num="9.8" data-grid-snap="false" data-prettyfy-separator="," data-prettyfy-enabled="true" data-keyboard="true" data-data-type="number"/>  
        </div>  
      </form>  
    </div>  
    <div class="col-sm-8">  
      <div id="distPlot" class="shiny-plot-output" style="width: 100% ; height: 400px"></div>  
  </div>  
</div>  
</div>
```


Architecture: Example

```
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')  
  })  
}
```

```
# Run the application  
shinyApp(ui = ui, server = server)
```

Template

- Download Tutorial 14, Unzip
 - Open app.R under template folder.

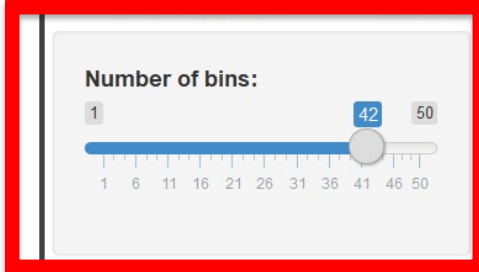
```
library(shiny)
ui <- fluidPage()

server <- function(input, output) {}

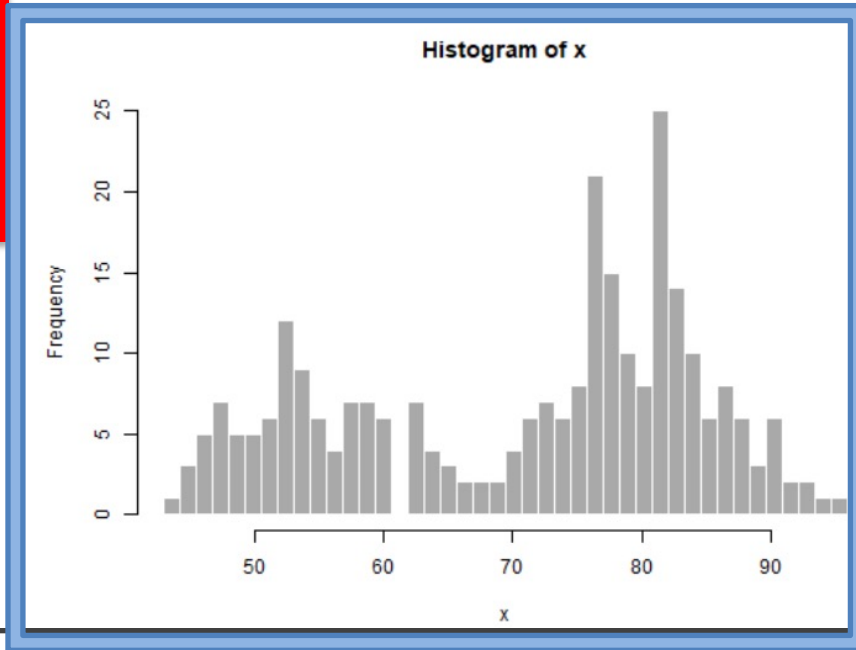
shinyApp(ui = ui, server = server)
```

UI: Inputs and Outputs

Old Faithful Geyser Data



Inputs



Outputs

```
ui <- fluidPage(  
  # *Input() functions,  
  # *Output() functions  
)
```

UI: Inputs

- Types of Inputs (UI)
 - First Two Arguments
 - `inputId` = Unique Variable Name So Server Knows When to Use It
 - `label` = Text That is Seen in Widget to Guide User
 - Other Arguments Depend on the Type of Input Function

```
sliderInput(inputId = "bins",  
            label = "Number of bins:",  
            min = 1,  
            max = 50,  
            value = 30)
```

UI: Input Functions

Buttons

Action

Submit

`actionButton()`
`submitButton()`

Single checkbox

Choice A

`checkboxInput()`

Checkbox group

Choice 1
 Choice 2
 Choice 3

`checkboxGroupInput()`

Date input

2014-01-01

`dateInput()`

Date range

2014-01-24 to 2014-01-24

`dateRangeInput()`

File input

Choose File No file chosen

`fileInput()`

Numeric input

1

`numericInput()`

Password Input

.....

`passwordInput()`

Radio buttons

Choice 1
 Choice 2
 Choice 3

`radioButtons()`

Select box

Choice 1

`selectInput()`

Sliders

0 50 100
0 25 75 100

`sliderInput()`

Text input

Enter text...

`textInput()`

UI: Outputs

- Types of Outputs (UI)
 - What User Can See
 - List of Possible Output Types

output function	creates
htmlOutput	raw HTML
imageOutput	image
plotOutput	plot
tableOutput	table
textOutput	text
uiOutput	raw HTML
verbatimTextOutput	text

- outputId = Connected to Output Created on the Server Side

```
plotOutput(outputId = "distPlot")
```

Server

- 3 Rules:
 - **Rule 1: Save objects to display to output\$**

```
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')  
  })  
}
```

```
plotOutput(outputId = "distPlot")
```

Server

- 3 Rules:
 - **Rule 2: Build objects to display with `render*()`**

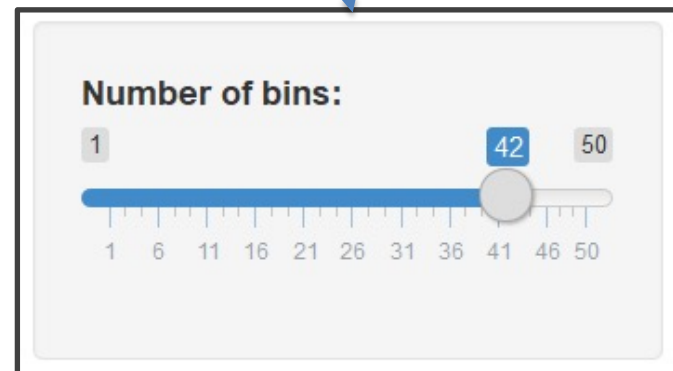
```
server <- function(input, output) {  
  
  output$distPlot <- renderPlot({  
    # generate bins based on input$bins from ui.R  
    x <- faithful[, 2]  
    bins <- seq(min(x), max(x), length.out = input$bins + 1)  
  
    # draw the histogram with the specified number of bins  
    hist(x, breaks = bins, col = 'darkgray', border = 'white')  
  })  
}
```

render function	creates
renderImage	images
renderPlot	plots
renderPrint	any printed output
renderTable	data frame, matrix, other table like structures
renderText	character strings
renderUI	a Shiny tag object or HTML

Server

- 3 Rules:
 - **Rule 3: Access `input` values with `input$`**

```
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')  
  })  
}
```



Server: Recap

- Rule 1: Save objects to display to `output$`
- Rule 2: Build objects to display with `render*()`
- Rule 3: Access `input` values with `input$`

Gapminder Shiny App

- Instructions
 - Step 1: Open app.R Files in Gapminder
 - Step 2: Install Gapminder
 - `> install.packages("gapminder")`
 - Step 5: Run the App
- [Gapminder](#) Data
 - Used in Chapter 20 (25 online, R4DS)
 - Non-Profit Project Promoting a Fact-based World
 - Popularized by Hans Rosling
 - Packaged in R
 - `> library(gapminder)`

Gapminder Shiny App

- Data Content in Gapminder

```
> head(gapminder)
# A tibble: 6 x 6
  country      continent  year lifeExp      pop gdpPercap
  <fct>        <fct>    <int> <dbl>    <int>    <dbl>
1 Afghanistan Asia      1952  28.8  8425333  779.
2 Afghanistan Asia      1957  30.3  9240934  821.
3 Afghanistan Asia      1962  32.0 10267083  853.
4 Afghanistan Asia      1967  34.0 11537966  836.
5 Afghanistan Asia      1972  36.1 13079460  740.
6 Afghanistan Asia      1977  38.4 14880372  786.
```

- Begin Using the App
 - Enter Name, Select Countries, Select Variable, and Submit
 - Observe the Use of CSS Code
 - Observe the tabsetPanel Style
 - Observe the Use of renderUI with uiOutput

Gapminder: UI

```
sidebarPanel(  
  helpText("Instructions: Select Countries of Interest for  
    Comparison and Analysis For Key Variables "),  
  br(),  
  #Input to Enter Name  
  textInput(inputId="name",label="First Name"),  
  
  #Input to Select Country  
  uiOutput(outputId="OUTcountry"),  
  
  #Input to Select Variable  
  uiOutput(outputId="OUTvariable"),  
  
  #Part 3:Line Width for Trend Graphic  
  sliderInput(inputId="width",  
    label="Width of Trend Lines",  
    min=1,max=3,value=1,step=1),  
  
  #Submit Button For Updates  
  submitButton("Stay Woke!")  
)
```

HTML Tags

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

Gapminder: Server

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

  #Display the Options Selected By the User
  output$OUTsynopsis1<-renderText({
    expr=paste("User:",input$name)
  })
  output$OUTsynopsis2<-renderText({
    expr=paste("Countries Selected:",paste(input$INcountry,collapse=", "))
  })
  output$OUTsynopsis3<-renderText({
    expr=paste("Variable for Analysis:",input$INvariable)
  })
}
```

Gapminder: Part 1

- Part 1: Data Selected
 - Server:

```
#Part 1: Create a Table Previewing Data
output$OUTpreview<-renderTable({
  gapminder2 %>%
    select(Country,Continent,Year,input$INvariable)%>%
    filter(Country %in% input$INcountry) %>%
    arrange(Year)
})|
```

- UI:

```
h2("Data Selected"),
br(),
#1: Print Data for Desired
# | Countries and Variable
tableOutput("OUTpreview"),
br(),
```


Gapminder: Part 2

- Part 2: Data Summary
 - Server

```
#Part 2: Create a Table Summarizing Data by Country
output$OUTsummary<-renderTable({
  gapminder2 %>%
    select(Country,Continent,Year,input$INvariable)%>%
    filter(Country %in% input$INcountry) %>%
    arrange(Year)%>%
    group_by(Country) %>%
    summarize(N=n(),
              MIN=min(get(input$INvariable)),
              Q1=quantile(get(input$INvariable),0.25),
              Q2=quantile(get(input$INvariable),0.5),
              Q3=quantile(get(input$INvariable),0.75),
              MAX=max(get(input$INvariable)),|
              CHANGE=MAX-MIN,
              MEAN=mean(get(input$INvariable)),
              SD=sd(get(input$INvariable))
    )
})
```

- UI

```
h2("Country Comparison"),
br(),
#2: Print Summary
tableOutput("OUTsummary"),
br()
```

Gapminder: Part 3

- Part 3: Trend Plots
 - Server

```
#Part 3: Create a Graphic Showing Trends
output$OUTtrendvar<-renderText({
  expr=paste("Trend Comparison for",input$INvariable)
})

output$OUTtrendplot<-renderPlot({
  gapminder2 %>%
    select(Country,Continent,Year,input$INvariable)%>%
    filter(Country %in% input$INcountry) %>%
    arrange(Year)%>%
    ggplot(aes(x=Year,y=get(input$INvariable))) +
    geom_line(aes(color=Country),size=input$width)+
    ylab(input$INvariable) +
    theme_minimal()
})
```

Gapminder: Part 3

- Part 3: Trend Plots (Continued)
 - UI
 - Creation of Slider Input

```
#Part 3:Line width for Trend Graphic
sliderInput(inputId="width",
            label="width of Trend Lines",
            min=1,max=3,value=1,step=1),

#Submit Button For Updates
```

- Displaying Graphic Output

```
tabPanel("Graphics",

        #3:Print Trend Graphic
        h2(textOutput("OUTtrendvar")),
        br(),
        plotOutput("OUTtrendplot"),
        br())
```

R Shiny Tutorial

- Official 3 Part [Video Tutorial](#)
- Official Shiny [Cheat Sheet](#)
- [Shiny Widget Gallery](#)
- Video Tutorials by [Abhinav Agrawal](#)
- [Video](#) Combining Shiny with Rmd
- [Video Tutorial](#) on Shiny Dashboard
- [Video Tutorials](#) by Johns Hopkins Data Science Lab
Produced by [Brian Caffo](#)