Northeastern University
College *of* Engineering

IE6600 Computation and Visualization for Analytics

R-Shiny Application

Zhenyuan Lu

Class Schedule

Contents

- UI widgets
- Widgets input values
- Reactive output
- Render functions
- Three small exercises (each exercise roughly ~10mins)
- Two medium exercise
- One large exercise (if we have time)

4.UI Widgets

ui

R-Shiny *Basic widgets*

The standard Shiny widgets are:

function	widget
`actionButton`	Action Button
`checkboxGroupInput`	A group of check boxes
`checkboxInput`	A single check box
`dateInput`	A calendar to aid date selection
`dateRangeInput`	A pair of calendars for selecting a date range
`fileInput`	A file upload control wizard
`helpText`	Help text that can be added to an input form
`numericInput`	A field to enter numbers
`radioButtons`	A set of radio buttons
`selectInput`	A box with choices to select from
`sliderInput`	A slider bar
`submitButton`	A submit button
`textInput`	A field to enter text

UI Widgets | Basic widgets | shinyWidgets package

actionButton()

Action

Submit

checkboxInput()

- [x] Choice A

checkboxGroupInput()

- [x] Choice 1
- [ ] Choice 2
- [ ] Choice 3

dateInput()

2014-01-01

dateRangeInput()

2019-03-25 to 2019-03-25

fileInput()

Browse... No file selected

helpText()

Note: help text isn't a true widget, but it provides an easy way to add text to accompany other widgets.

numericInput()

1

radioButtons()

- (•) Choice 1
- ( ) Choice 2
- ( ) Choice 3

selectInput()

Choice 1

sliderInput()

0 50 100

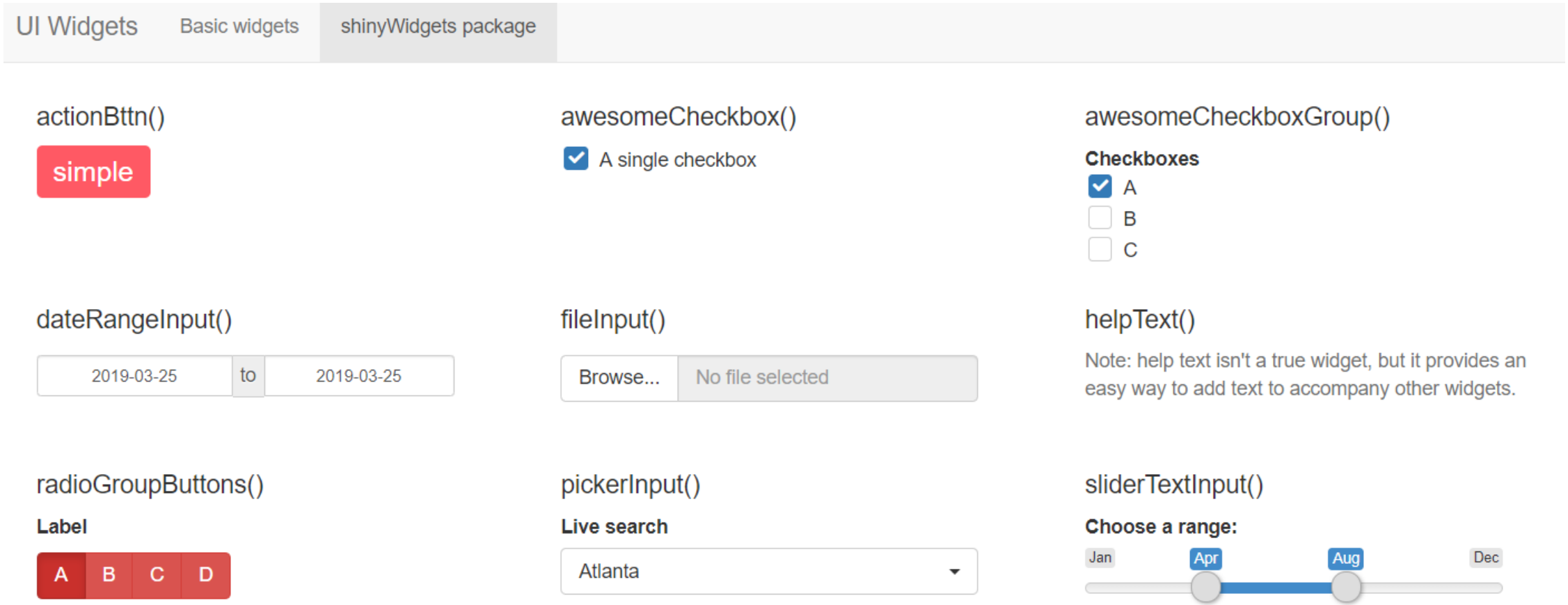
0 10 20 30 40 50 60 70 80 90 100

0 25 75 100

0 10 20 30 40 50 60 70 80 90 100

textInput()

Enter text...



R-Shiny *Excercise*

UI Widgets | Basic widgets | shinyWidgets package

actionButton()

Action

Submit

checkboxInput()

- [x] Choice A

checkboxGroupInput()

- [x] Choice 1
- [ ] Choice 2
- [ ] Choice 3

dateInput()

2014-01-01

dateRangeInput()

2019-03-25 to 2019-03-25

fileInput()

Browse... No file selected

helpText()

Note: help text isn't a true widget, but it provides an easy way to add text to accompany other widgets.

numericInput()

1

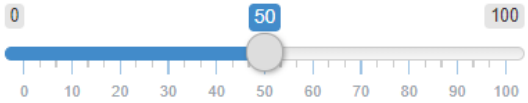
radioButtons()

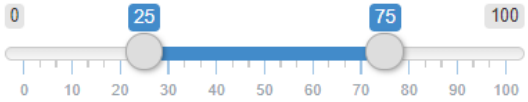
- (•) Choice 1
- ( ) Choice 2
- ( ) Choice 3

selectInput()

Choice 1

sliderInput()





textInput()

Enter text...

See uiWidget2.R on Canvas

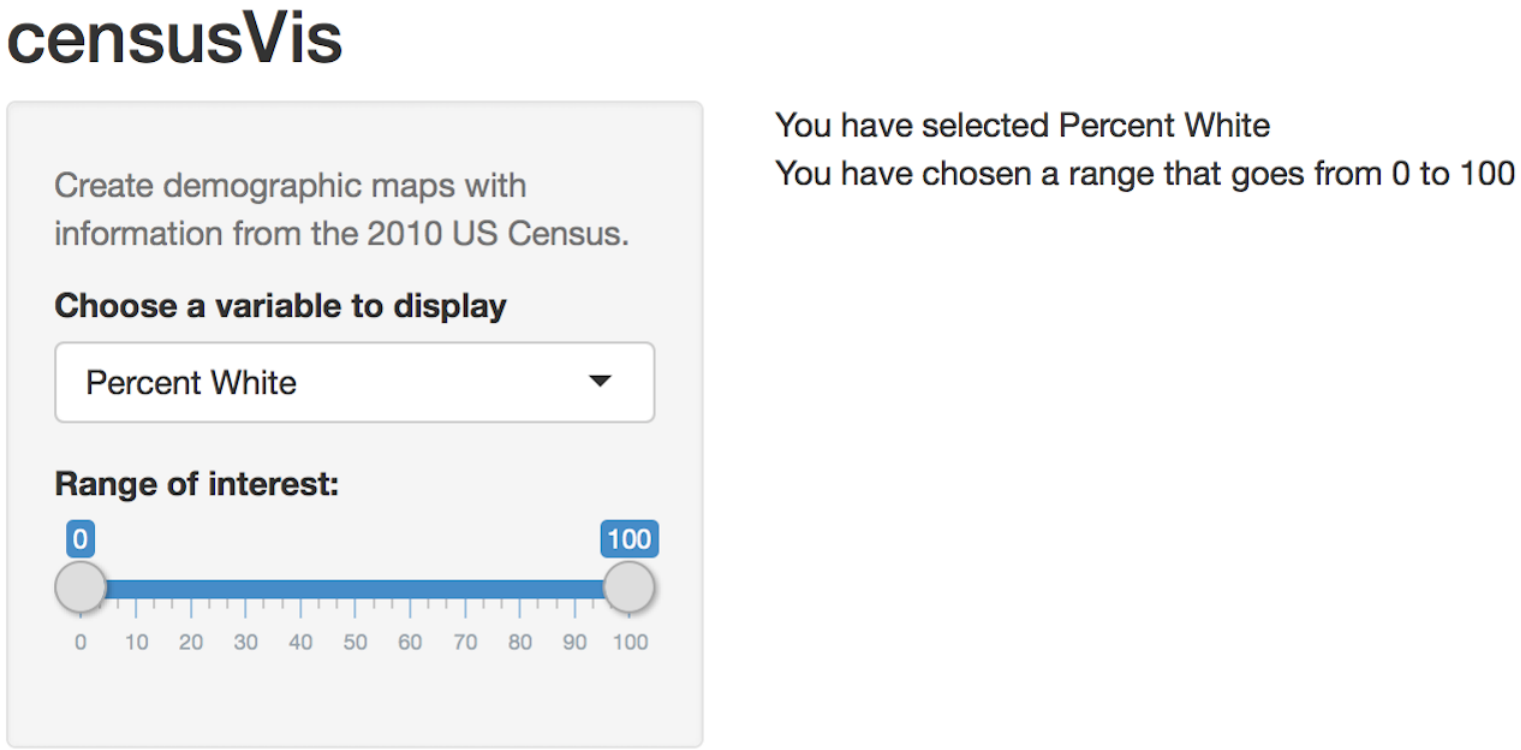
ui

output values

```
uiOutput()
verbattimTextOutput()
tableOutput()
plotlyOutput()
plotOutput()
.
.
.
```

5.Reactive Output, functions and data

server



Two steps

You can create reactive output with a two step process:

1. Add an R object to your user interface.
2. Tell Shiny how to build the object in the server function. The object will be reactive if the code that builds it calls a widget value.

R-Shiny *Step1: Add an R object to the UI*

Shiny provides a family of functions that turn R objects into output for your user interface. Each function creates a specific type of output, which included but not limited to the following objects:

Output function	Creates
`dataTableOutput`	DataTable
`htmlOutput`	raw HTML
`imageOutput`	image
`plotOutput`	plot
`tableOutput`	table
`textOutput`	text
`uiOutput`	raw HTML
`verbatimTextOutput`	text

R-Shiny *For example*

```
ui <- fluidPage(
  titlePanel("censusVis"),

  sidebarLayout(
    sidebarPanel(
      helpText("Create demographic maps with
               information from the 2010 US Census."),

      selectInput("var",
                  label = "Choose a variable to display",
                  choices = c("Percent White",
                              "Percent Black",
                              "Percent Hispanic",
                              "Percent Asian"),
                  selected = "Percent White"),

      sliderInput("range",
                  label = "Range of interest:",
                  min = 0, max = 100, value = c(0, 100))
    ),

    mainPanel(
      textOutput("selected_var")
    )
  )
)
```

For example, the ui object on the left uses textOutput to add a reactive line of text to the main panel of the Shiny app pictured above.

Notice that textOutput takes an argument, the character string "selected_var". Each of the *Output functions require a single argument: a character string that Shiny will use as the name of your reactive element. Your users will not see this name, but you will use it later.

R-Shiny *Step2: Provide R code to build the object*

Placing a function in ui tells Shiny where to display your object. Next, you need to tell Shiny how to build the object. We do this by providing the R code that builds the object in the server function.

```
server <- function(input, output) {

  output$selected_var <- renderText({
    "You have selected this"
  })

}
```

R-Shiny *Render function for each output*

Each entry to output should contain the output of one of Shiny's render* functions. These functions capture an R expression and do some light pre-processing on the expression. Use the render* function that corrresponds to the type of reactive object you are making.

render function	creates
`renderDataTable`	DataTable
`renderImage`	images (saved as a link to a source file)
`renderPlot`	plots
`renderPrint`	any printed output
`renderTable`	data frame, matrix, other table like structures
`renderText`	character strings
`renderUI`	a Shiny tag object or HTML

ui
output values

Output function	Creates
`dataTableOutput`	DataTable
`htmlOutput`	raw HTML
`imageOutput`	image
`plotOutput`	plot
`tableOutput`	table
`textOutput`	text
`uiOutput`	raw HTML
`verbatimTextOutput`	text

server
associated render

render function	creates
`renderDataTable`	DataTable
`renderImage`	images
`renderPlot`	plots
`renderPrint`	any printed output
`renderTable`	data frame, matrix
`renderText`	character strings
`renderUI`	a Shiny tag object or HTML

R-Shiny *Use widget(input) values*

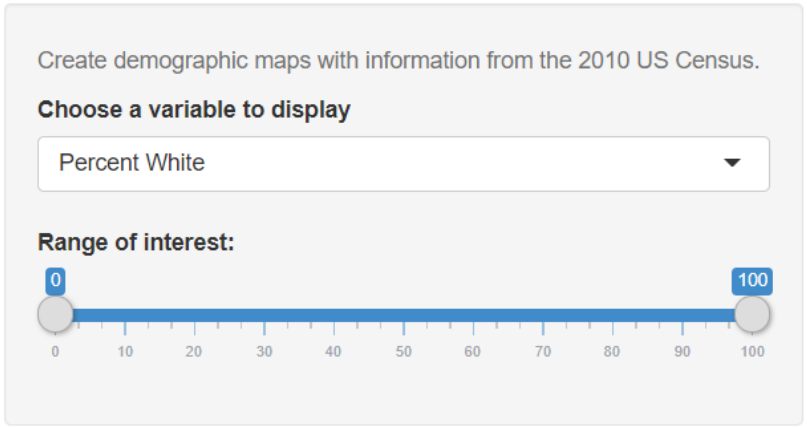
Shiny will automatically make an object reactive if the object uses an input value. For example, the server function below creates a reactive line of text by calling the value of the select box widget to build the text.

```
server <- function(input, output) {

  output$selected_var <- renderText({
    paste("You have selected", input$var)
  })

}
```

Exercise 1

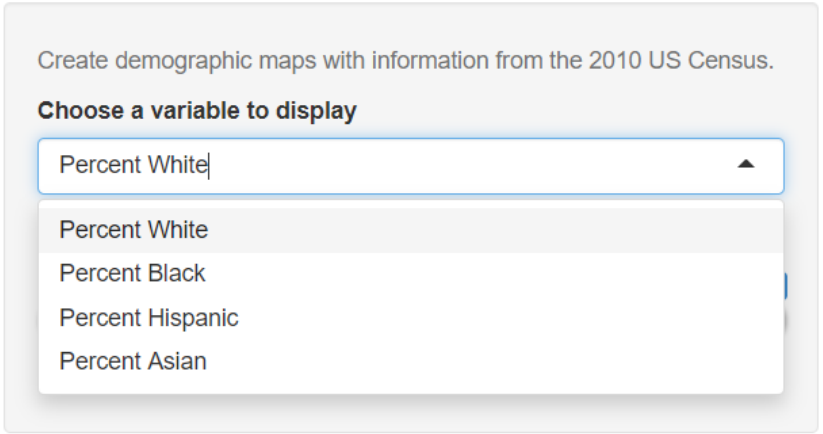
R-Shiny *Exercise(5min)*

exercise 1



You have selected Percent White
You have chosen a range that goes from 0 to 100

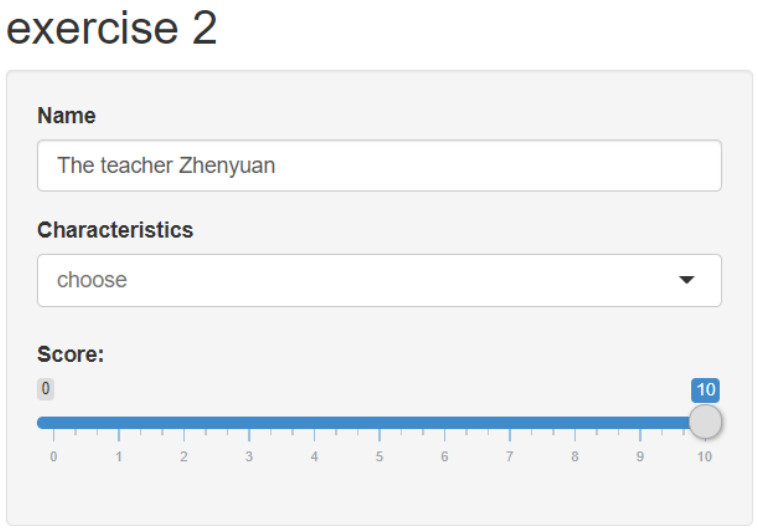
exercise 1



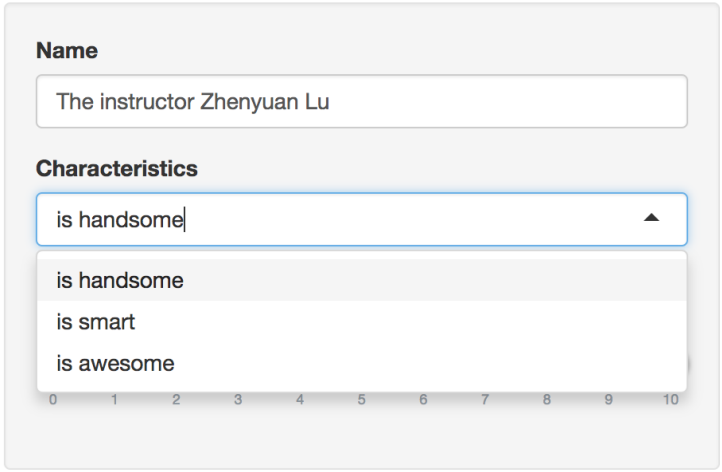
You have selected Percent White
You have chosen a range that goes from 0 to 100

Exercise 2

R-Shiny *Exercise(10min)*



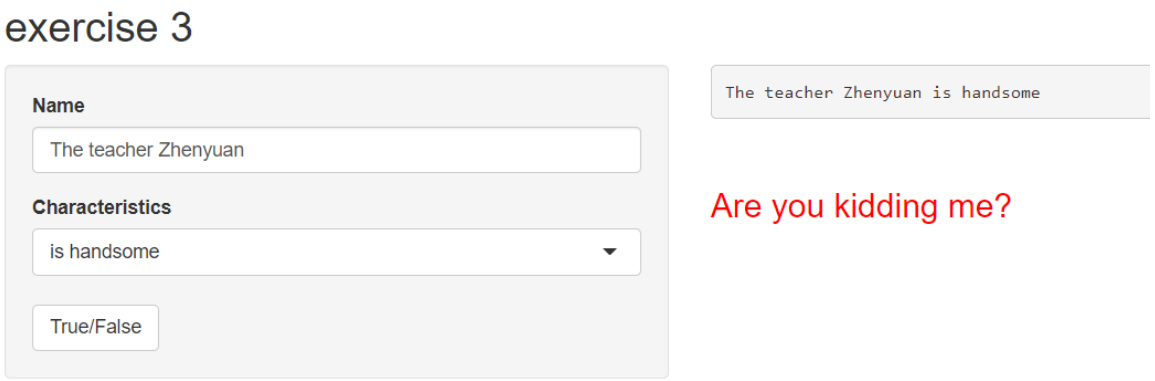

```
The teacher Zhenyuan  . (Score: 10 )
```

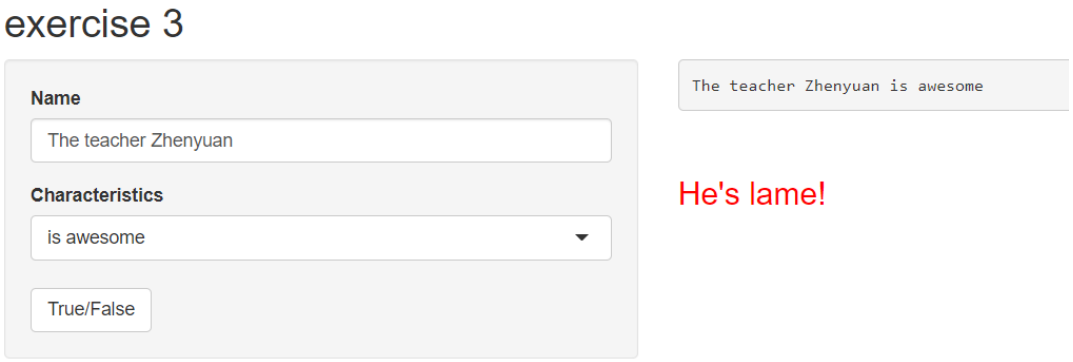


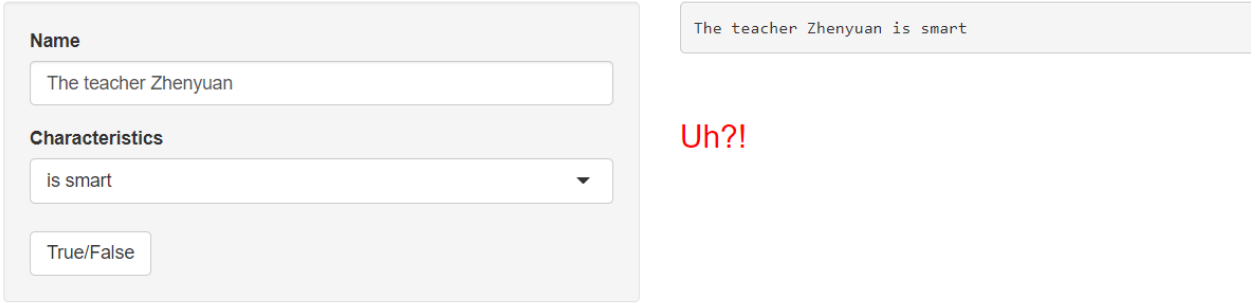

```
The instructor Zhenyuan Lu is handsome . (Score: 10 )
```

Exercise 3

R-Shiny *Exercise(10min)*







server functions

```
# Set up a trigger for dynamically
action ----
observeEvent({})
# Store a reactive value into shiny
server ----
reactiveValues({})
```

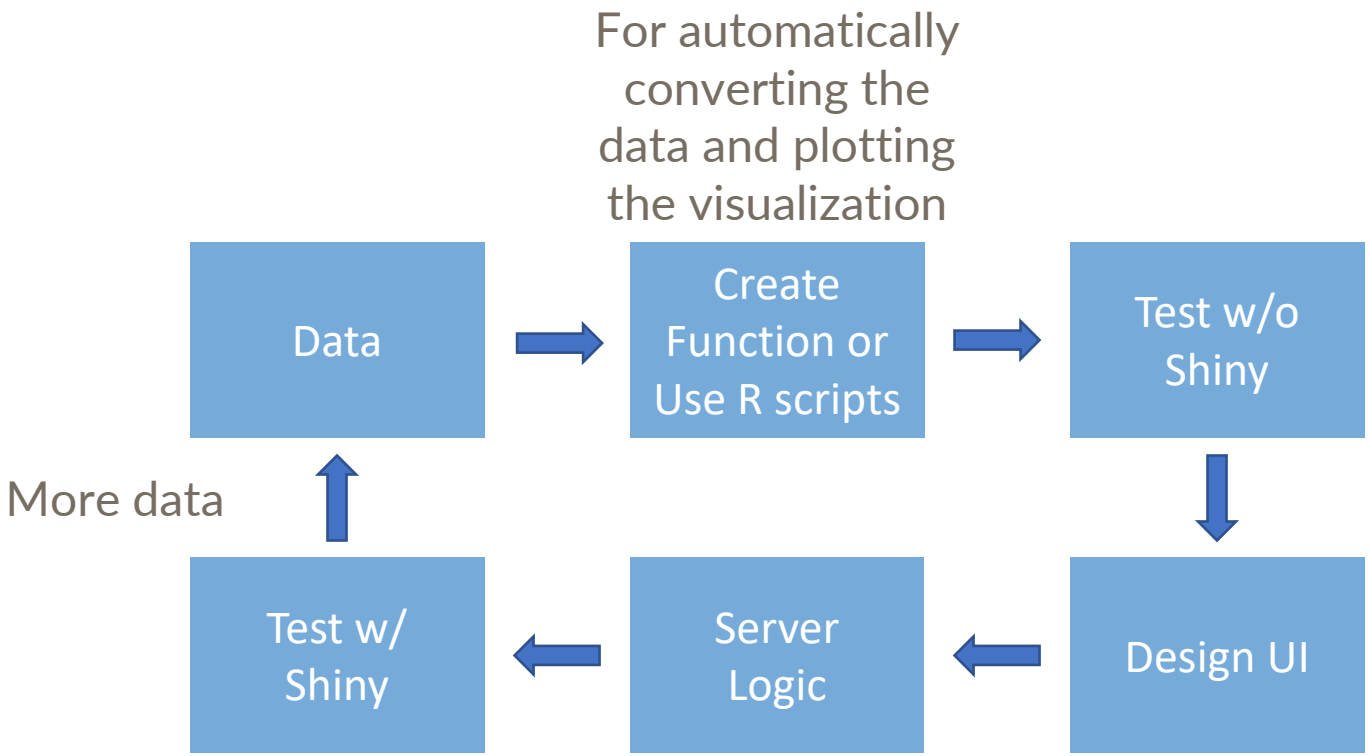
`global`

global settings

Library()
Global settings
Global dataset

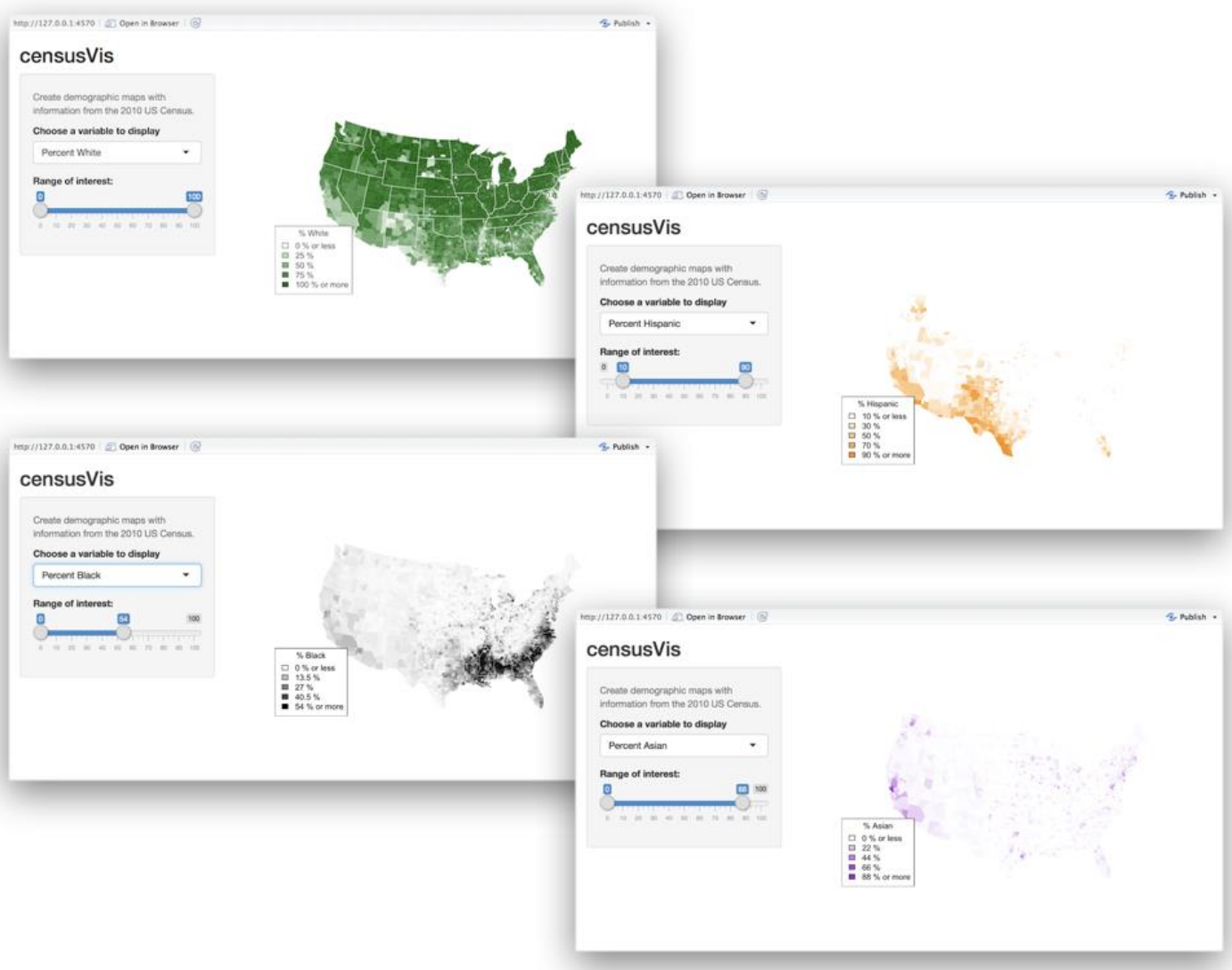
A simple workflow

R-Shiny *Work Flow*

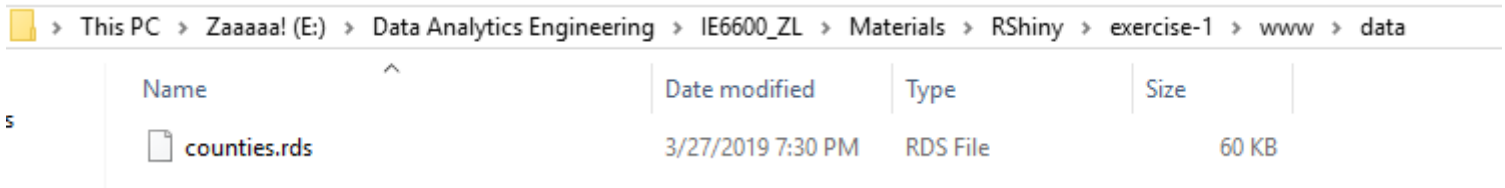


Exercise 4

R-Shiny *This is what we want to create*



R-Shiny *Find the data - counties.RDS*



Please download the .RDS file from Canvas

Then create a ShinyApp folder -> www -> data

R-Shiny *Create a function for plot map based on countries.RDS*

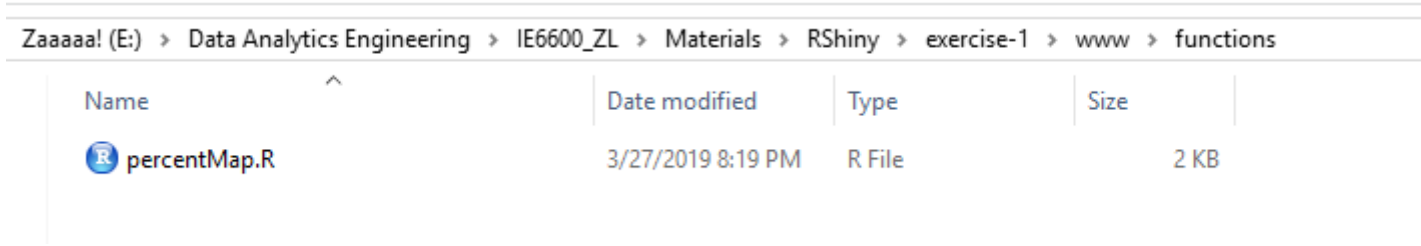
```
# Note: percent map is designed to work with the counties data set
# It may not work correctly with other data sets if their row order does
# not exactly match the order in which the maps package plots counties
percent_map <- function(var, color, legend.title, min = 0, max = 100) {

  # generate vector of fill colors for map
  shades <- colorRampPalette(c("white", color))(100)

  # constrain gradient to percents that occur between min and max
  var <- pmax(var, min)
  var <- pmin(var, max)
  percents <- as.integer(cut(var, 100,
    include.lowest = TRUE, ordered = TRUE))
  fills <- shades[percents]

  # plot choropleth map
  map("county", fill = TRUE, col = fills,
    resolution = 0, lty = 0, projection = "polyconic",
    myborder = 0, mar = c(0,0,0,0))

  # overlay state borders
  map("state", col = "white", fill = FALSE, add = TRUE,
    lty = 1, lwd = 1, projection = "polyconic",
    myborder = 0, mar = c(0,0,0,0))

  # add a legend
  inc <- (max - min) / 4
  legend.text <- c(paste0(min, " % or less"),
    paste0(min + inc, " %"),
    paste0(min + 2 * inc, " %"),
    paste0(min + 3 * inc, " %"),
    paste0(max, " % or more"))

  legend("bottomleft",
    legend = legend.text,
    fill = shades[c(1, 25, 50, 75, 100)],
    title = legend.title)
}
```

This is just an example how to make a plot function:

percent_map()

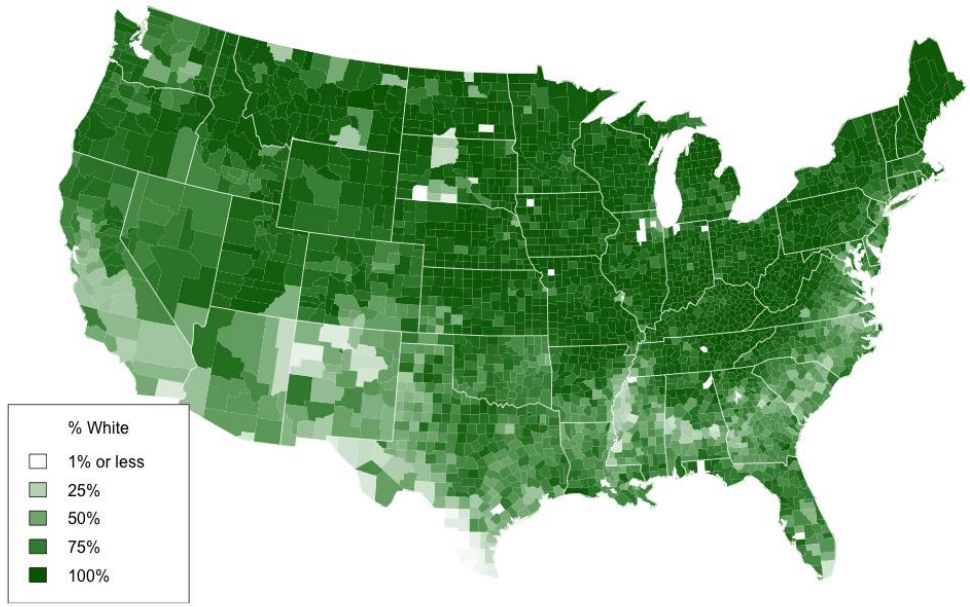
You don't have to understand the code syntax

Copy and paste the left code chunk into a new .R file named percentMap, then place it into your ShinyApp folder -> www -> functions



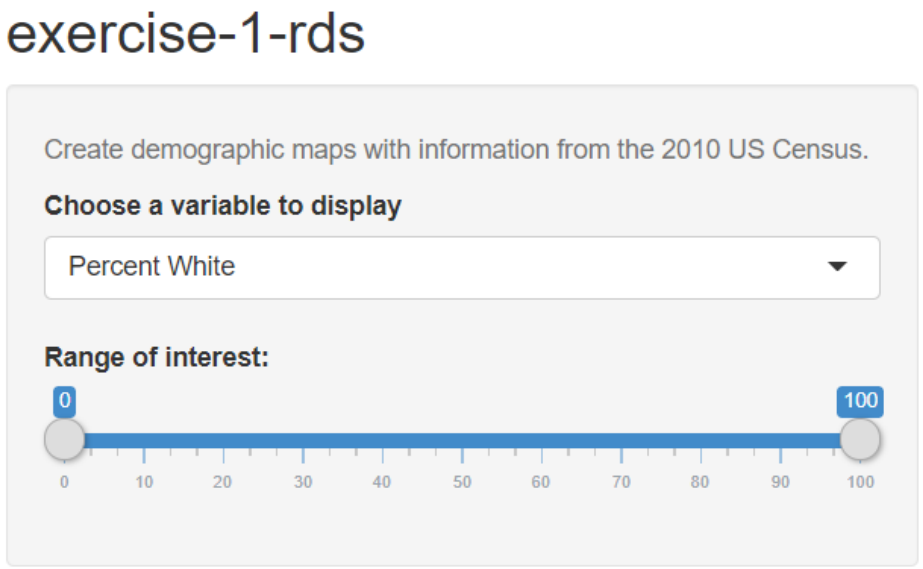
R-Shiny *Test without Shiny*

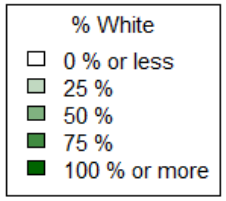
```
library(maps)
library(mapproj)
source(yourFunctionPath)
counties <- readRDS(YourDataPath)
percent_map(counties$white, "darkgreen",
"% White")
```

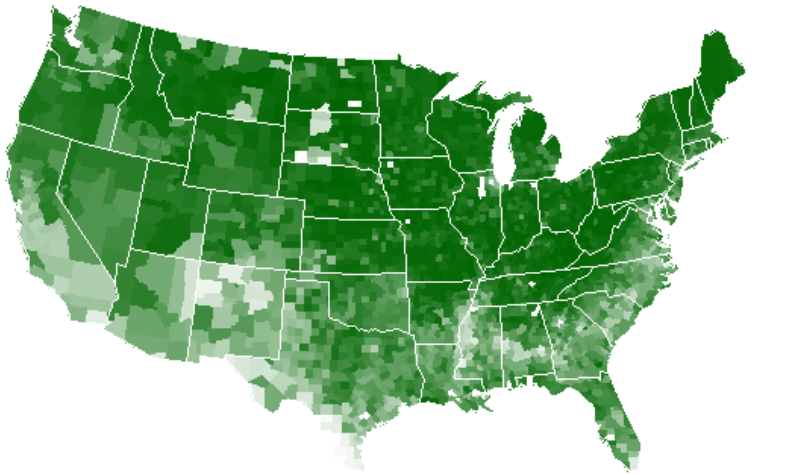


BAAM!

R-Shiny *Big Picture*

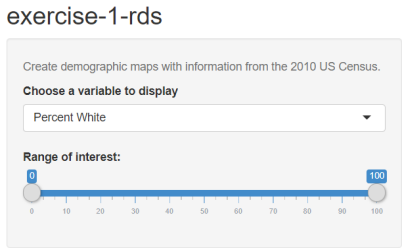


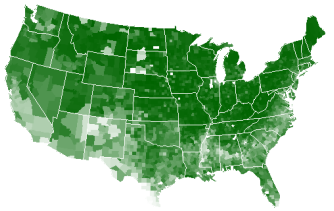




R-Shiny *Design UI*

```
ui <- fluidPage(titlePanel("exercise-1-rds"),
                sidebarLayout(
                  sidebarPanel(
                    helpText("Create demographic maps with
                             information from the 2010 US Census."),

                    selectInput(
                      "var",
                      label = "Choose a variable to display",
                      choices = c(
                        "Percent White",
                        "Percent Black",
                        "Percent Hispanic",
                        "Percent Asian"
                      ),
                      selected = "Percent White"
                    ),

                    sliderInput(
                      "range",
                      label = "Range of interest:",
                      min = 0,
                      max = 100,
                      value = c(0, 100)
                    )
                    ),

                  mainPanel()
                ))
```

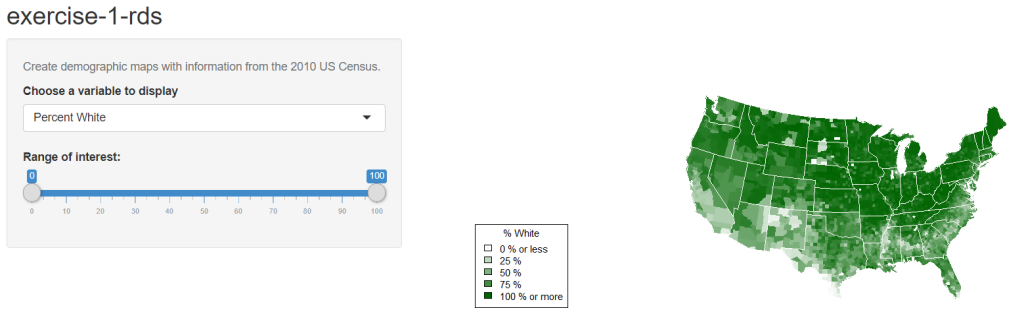




Which part did we miss?

R-Shiny *Design Server*

```
# Server logic ----
server <- function(input, output) {
  # some arguments
}
```

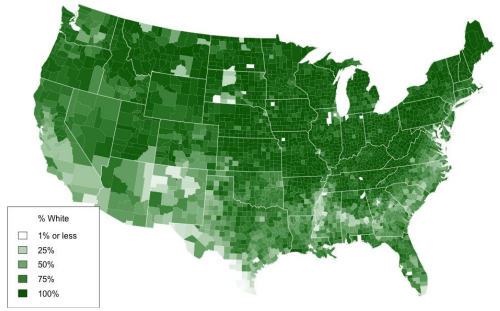


What do we need for the arguments?

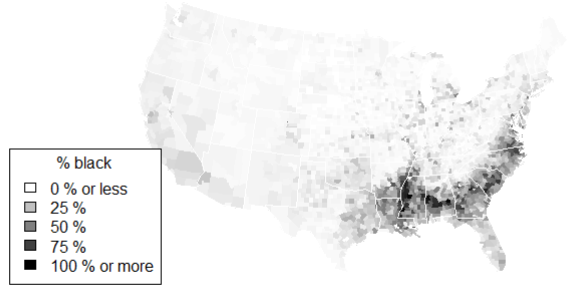
R-Shiny *Design Server*

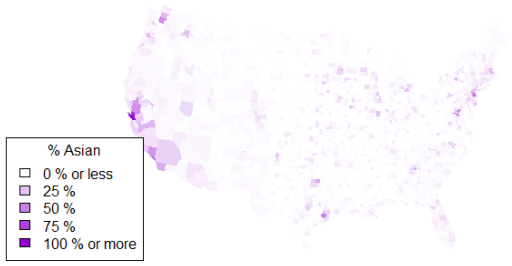
Let's get back to the following scripts.

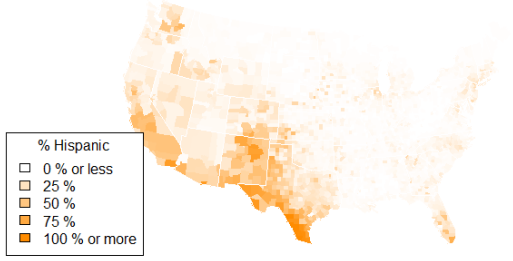
```
percent_map(counties$white, "darkgreen", "% White")

percent_map(counties$black, "black", "% Black")

percent_map(counties$hispanic, "darkorange", "% Hispanic")

percent_map(counties$asian, "darkviolet", "% Asian")
```



We have three arguments/variables: var, color, and legend.title



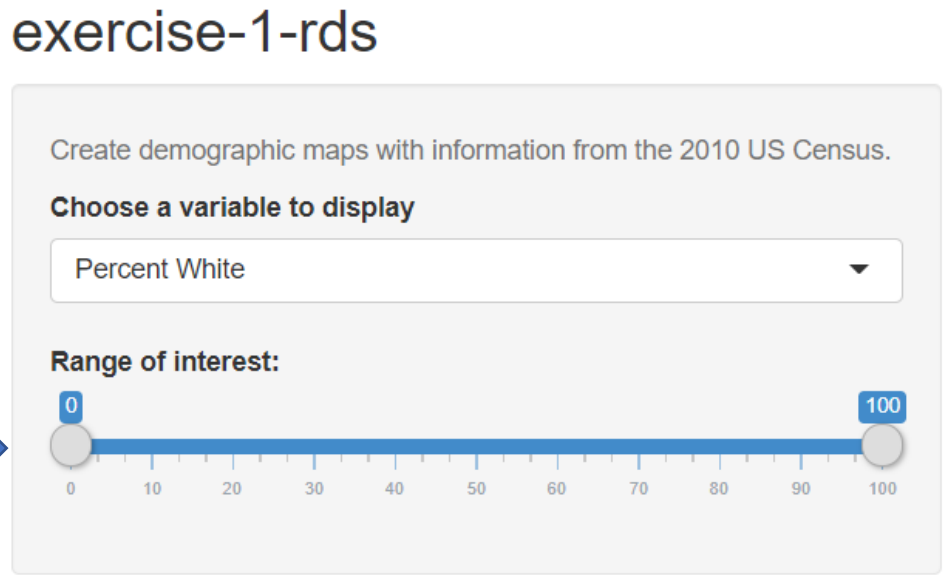




https://shiny.rstudio.com/tutorial/written-tutorial/lesson5/

R-Shiny *Design Server*

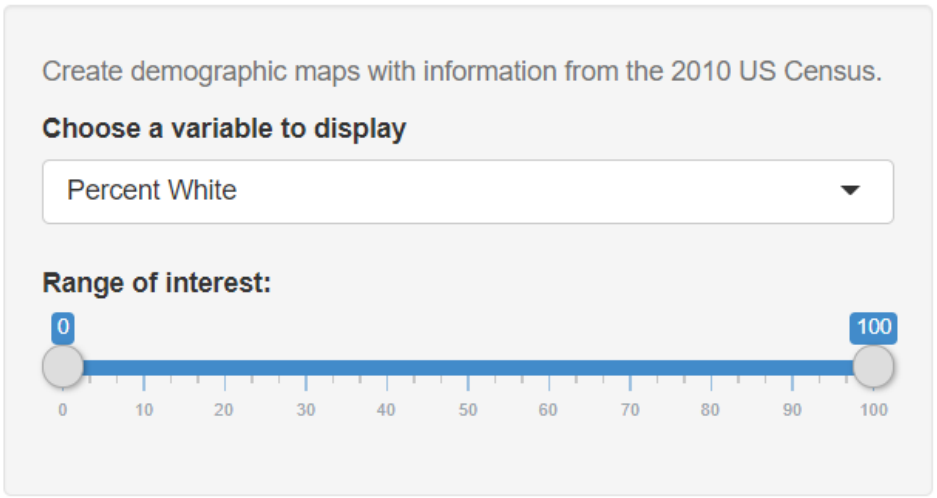
We have another two arguments: max and min



https://shiny.rstudio.com/tutorial/written-tutorial/lesson5/

R-Shiny *Design Server*

```
ui <- fluidPage(titlePanel("exercise-1-rds"),
                sidebarLayout(
                  sidebarPanel(
                    helpText("Create demographic maps with
                             information from the 2010 US Census."),

                    selectInput(
                      "var",
                      label = "Choose a variable to display",
                      choices = c(
                        "Percent White",
                        "Percent Black",
                        "Percent Hispanic",
                        "Percent Asian"
                      ),
                      selected = "Percent White"
                    ),

                    sliderInput(
                      "range",
                      label = "Range of interest:",
                      min = 0,
                      max = 100,
                      value = c(0, 100)
                    )
                    ),

                  mainPanel()
                ))
```




```
server <- function(input, output) {
  output$map <- renderPlot({
    data <- switch(input$var,
                   "Percent White" = counties$white,
                   "Percent Black" = counties$black,
                   "Percent Hispanic" = counties$hispanic,
                   "Percent Asian" = counties$asian)

    percent_map(var = data, color = ?, legend.title = ?, max = ?,
min = ?)
  })
}
```

Finish the rest four arguments

https://shiny.rstudio.com/tutorial/written-tutorial/lesson5/

```
Rep("Exercise 4", 1)
```

One more time

Exercise 5

server functions

```
# Set up a trigger for dynamically
action ----
observeEvent({})
# Store a reactive value into shiny
server ----
reactiveValues({})
```

R-Shiny *Titanic data*

exercise-titanic

Create GLM based on titanic dataset.

Choose CSV File

Browse... titanic.csv

Upload complete

Choose a x variable

age

Choose a color variable

sex

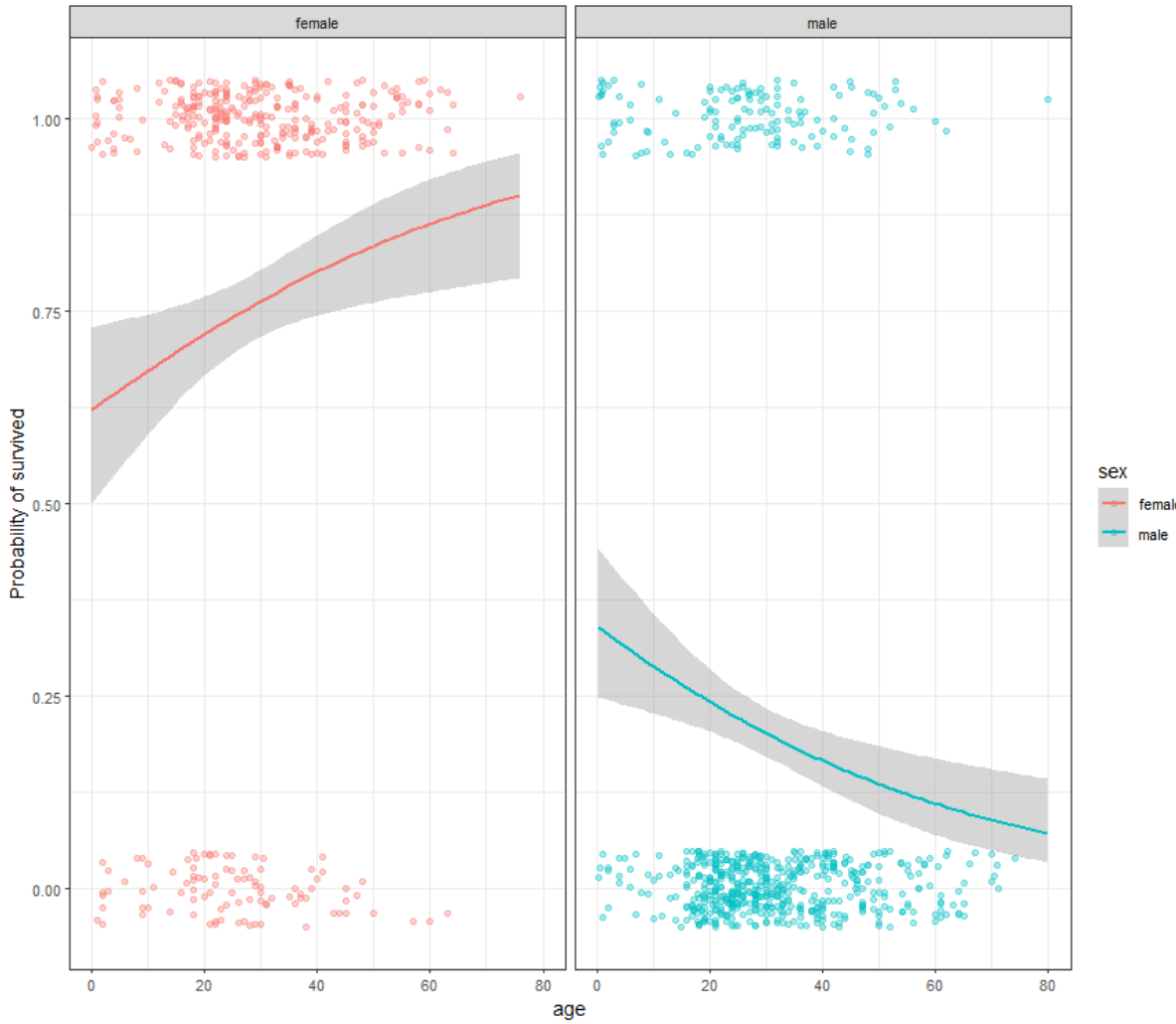
Choose a facet variable

sex

Search:

	X1	pclass	survived	name
1	1	1	1	Allen, Miss. Elisabeth Walto
2	2	1	1	Allison, Master. Hudson Tre
3	3	1	0	Allison, Miss. Helen Loraine
4	4	1	0	Allison, Mr. Hudson Joshua
5	5	1	0	Allison, Mrs. Hudson J C (E
6	6	1	1	Anderson, Mr. Harry
7	7	1	1	Andrews, Miss. Kornelia Th
8	8	1	0	Andrews, Mr. Thomas Jr
9	9	1	1	Appleton, Mrs. Edward Dal
10	10	1	0	Artagaveytia, Mr. Ramon

Showing 1 to 11 of 1,309 entries



Exercise 6

Exercise 6: 107 lines

2nd Example

Title:
Fall2018

Course:
IE6600

Country:
US

Numbers:
16

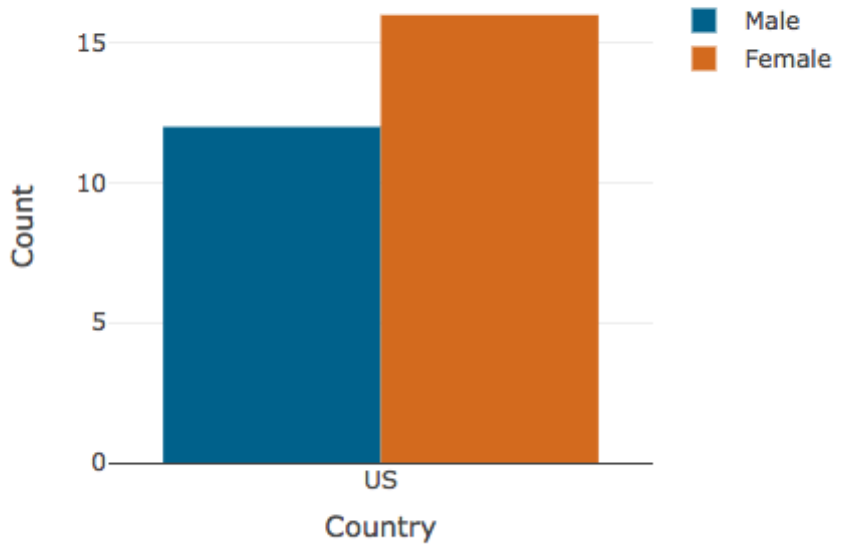
Gender:
Female

Submit/Update

recall

Fall2018

Course	Country	Numbers	Gender
IE6600	US	12	Male
IE6600	US	16	Female



Answers

Lu, Zhenyuan. Data Visualization Tutorial in R.. zhenyuanlu.github.io, 2022.

R-Shiny *Answer for Exercise 1*

```
ui <- fluidPage(
  titlePanel("exercise 1"),

  sidebarLayout(
    sidebarPanel(
      helpText("Create demographic maps with
               information from the 2010 US Census."),

      selectInput("var",
                  label = "Choose a variable to display",
                  choices = c("Percent White",
                              "Percent Black",
                              "Percent Hispanic",
                              "Percent Asian"),
                  selected = "Percent White"),

      sliderInput("range",
                  label = "Range of interest:",
                  min = 0, max = 100, value = c(0, 100))
      ),

    mainPanel(
      textOutput("selected_var"),
      textOutput("selected_num")
    )
  )
  )
```

```
server <- function(input, output) {

  output$selected_var <- renderText({
    paste("You have selected", input$var)
  })
  output$selected_num <- renderText({
    paste("You have chosen a range that goes from ",
input$range[1], "to", input$range[2])
  })

}

shinyApp(ui, server)
```

R-Shiny *Answer for Exercise 2*

```
library(shiny)


ui <- fluidPage(
  titlePanel("exercise 2"),

  sidebarLayout(
    sidebarPanel(
      textInput("text","Name",value="The teacher Zhenyuan"),

      selectInput("variable",
                  label = "Characteristics",
                  choices = list(choose="","is handsome",
                                 "is smart",
                                 "is awesome")),

      sliderInput("range","Score:", min=0, max=10, value=10)
      ),

    mainPanel(fluidPage(
      fluidRow(
        verbatimTextOutput("textouput")
      )
    ))
  )
  )

server <- function(input ,output){
  output$textouput <- renderText({
    paste(input$text,input$variable,".",
          "(Score:",input$range,")",sep=" ")

  })

}

shinyApp(ui, server)
```

R-Shiny *Answer for Exercise 3*

```
library(shiny)

ui <- fluidPage(
  titlePanel("exercise 3"),

  sidebarLayout(
    sidebarPanel(
      textInput("text","Name",value="The teacher Zhenyuan"),

      selectInput("variable",
                  label = "Characteristics",
                  choices = list(choose="","is handsome",
                                 "is smart",
                                 "is awesome")),
      actionButton("evaluation", "True/False")
      ),

    mainPanel(fluidPage(
      fluidRow(
        verbatimTextOutput("textouput"),
        br(),
        uiOutput("truth")
      )
    ))
  )
  )
```

```
server <- function(input ,output){
  values <- reactiveValues()

  output$textouput <- renderText({
    paste(input$text,input$variable,sep=" ")})

  observeEvent(input$evaluation,{
    if(input$variable=="is handsome"){
      output$truth <- renderUI({
        h3(helpText(paste("Are you kidding me?"), style="color:red"))
      })}else{
        if(input$variable=="is smart"){
          output$truth <- renderUI({h3(helpText(paste("Uh?!"),
style="color:red"))})
        }else{if(input$variable=="is awesome"){
          output$truth <- renderUI({h3(helpText(paste("He's lame!"),
style="color:red"))})
        }}
      }

  })



}

shinyApp(ui, server)
```

R-Shiny *Answer for Exercise 4*

```
# Load packages ----
library(shiny)
library(maps)
library(mapproj)

# Load data ----
counties <- readRDS("www/data/counties.RDS")

# Source helper functions -----
source("www/functions/percentMap.R")


# User interface ----
ui <- fluidPage(titlePanel("exercise-4-rds"),

                sidebarLayout(
                  sidebarPanel(
                    helpText("Create demographic maps with
                             information from the 2010 US Census."),

                    selectInput(
                      "var",
                      label = "Choose a variable to display",
                      choices = c(
                        "Percent White",
                        "Percent Black",
                        "Percent Hispanic",
                        "Percent Asian"
                      ),
                      selected = "Percent White"
                    ),

                    sliderInput(
                      "range",
                      label = "Range of interest:",
                      min = 0,
                      max = 100,
                      value = c(0, 100)
                    )
                    ),

                  mainPanel(plotOutput("map"))
                ))
```

```
# Server logic ----
server <- function(input, output) {
  output$map <- renderPlot({
    data <- switch(
      input$var,
      "Percent White" = counties$white,
      "Percent Black" = counties$black,
      "Percent Hispanic" = counties$hispanic,
      "Percent Asian" = counties$asian
    )

    color <- switch(
      input$var,
      "Percent White" = "darkgreen",
      "Percent Black" = "black",
      "Percent Hispanic" = "darkorange",
      "Percent Asian" = "darkviolet"
    )

    legend <- switch(
      input$var,
      "Percent White" = "% White",
      "Percent Black" = "% Black",
      "Percent Hispanic" = "% Hispanic",
      "Percent Asian" = "% Asian"
    )

    percent_map(data, color, legend, input$range[1], input$range[2])
  })

}

shinyApp(ui, server)
```

R-Shiny *Answer for Exercise 4*

```
# Server logic ----
server <- function(input, output) {
  output$map <- renderPlot({
    data <- switch(
      input$var,
      "Percent White" = counties$white,
      "Percent Black" = counties$black,
      "Percent Hispanic" = counties$hispanic,
      "Percent Asian" = counties$asian
    )

    color <- switch(
      input$var,
      "Percent White" = "darkgreen",
      "Percent Black" = "black",
      "Percent Hispanic" = "darkorange",
      "Percent Asian" = "darkviolet"
    )

    legend <- switch(
      input$var,
      "Percent White" = "% White",
      "Percent Black" = "% Black",
      "Percent Hispanic" = "% Hispanic",
      "Percent Asian" = "% Asian"
    )

    percent_map(data, color, legend, input$range[1], input$range[2])
  })

}
```

```
# More brief ----
server <- function(input, output) {
  output$map <- renderPlot({
    args <- switch(input$var,
                   "Percent White" = list(counties$white, "darkgreen", "% White"),
                   "Percent Black" = list(counties$black, "black", "% Black"),
                   "Percent Hispanic" = list(counties$hispanic, "darkorange", "% Hispanic"),
                   "Percent Asian" = list(counties$asian, "darkviolet", "% Asian"))

    args$min <- input$range[1]
    args$max <- input$range[2]

    do.call(percent_map, args)
  })
}
```

R-Shiny *Answer for Exercise 5*

```
# Load packages ----
library(shiny)
library(tidyverse)
library(plotly)
library(DT)
library(shinyWidgets)

# Source helper functions -----
source("www/functions/titanicGlm.R")

xis <- c("age", "fare")
color <- c(
  "plcass",
  "survived",
  "name" ,
  "sex",
  "age",
  "sibsp",
  "parch",
  "ticket",
  "fare",
  "cabin",
  "embarked",
  "boat",
  "body",
  "home.dest"
)
facet.1 <- c("pclass", "survived", "sex", "age")
```

```
# User interface ----
ui <- fluidPage(titlePanel("exercise-titanic"),
                sidebarLayout(
                  sidebarPanel(
                    width = 2,
                    helpText("Create GLM based on titanic dataset. "),

                    # Input: Select a file ----
                    fileInput(
                      "titanic",
                      "Choose CSV File",
                      multiple = FALSE,
                      accept = c("text/csv",
                                 "text/comma-separated-values,text/plain",
                                 ".csv")
                    ),

                    selectInput(
                      "xv",
                      label = "Choose a x variable",
                      choices = xis,
                      selected = "age"
                    ),

                    selectInput(
                      "colr",
                      label = "Choose a color variable",
                      choices = color,
                      selected = "sex"
                    )
                    ,

                    selectInput(
                      "fac",
                      label = "Choose a facet variable",
                      choices = facet.1,
                      selected = "sex"
                    )
                  ),

                  mainPanel(fluidPage(fluidRow(
                    column(6,
                           DT::dataTableOutput("dataSet")),
                    column(6,
                           plotOutput(
                             "glm", width = "700px", height = "600px"
                           ))
                  )))
                ))
```

R-Shiny *Answer for Exercise 5*

```
# Server logic ----
server <- function(input, output) {
  values <- reactiveValues(tbl=NULL)

  observeEvent(input$titanic, {
    # Store the uploaded file ----
    values$tbl <- read_csv(input$titanic$datapath)
    output$dataSet <- DT::renderDataTable({
      tryCatch({
        df <- values$tbl
      },
      error = function(e) {
        stop(safeError(e))
      })
    },
    extensions = c('Scroller', 'FixedColumns'),
    options = list(
      deferRender = TRUE,
      scrollX = TRUE,
      scrollY = 400,
      scroller = TRUE,
      dom = 'Bfrtip',
      fixedColumns = TRUE
    ))
  })

  output$glm <- renderPlot({

    titanicGlm(values$tbl, input$xv, input$colr, facet = input$fac)
  })


}




# Run app ----
shinyApp(ui, server)
```

R-Shiny *Answer for Exercise 6*

```
library(shiny)
library(plotly)

df.path <- file.path("www/data/students.csv")
ui <- fluidPage(#theme = shinytheme("paper"),
  titlePanel("2nd Example-Zhenyuan Lu"),
  sidebarLayout(
    sidebarPanel(
      width=2,
      textInput(
        width = "100%",
        inputId = "title",
        label = "Title:",
        value = NA
      ),
      textInput(
        width="100%",
        inputId = "course",
        label="Course:",
        value = NA
      ),
      textInput(
        width = "100%",
        inputId = "country",
        label = "Country:",
        value = NA
      ),

      selectInput(
        width="100%",
        inputId = "gender",
        label="Gender:",
        choices = c(choose='',"Male","Female","Others")
        ),

      numericInput(
        width="100%",
        inputId="numbers",
        label="Numbers:",
        value = NA
      ),
      actionButton(
        inputId = "update",
        label = "Submit/Update",
        icon = icon("database"),
        width = "100%"
      ),
      actionButton(
        inputId="recall",
        label="recall",
        icon=icon("refresh"),
        width="100%"
      )
      #verbatimTextOutput("testTxt")
    ),

    mainPanel(fluidPage(fluidRow(
      column(6,
             h3(textOutput("title",container = span)),
             tableOutput("dataSet")),
      column(6,
             plotlyOutput(
               "barchart", width = "100%", height = "300px"
             ))
    )))
  ))
```

R-Shiny *Answer for Exercise 6*

```
server <- function(input, output) {
  values <- reactiveValues()

  observeEvent(input$update,{
    values$dataInput <- data.frame(
      Course=input$course,
      Country=input$country,
      Numbers=input$numbers,
      Gender=input$gender
    )
    values$df <- as.data.frame(read.csv(df.path)[-1])
    if(!"TRUE"%in%is.na(values$dataInput)){
      values$newStudent <- na.omit(unique(rbind(values$df,values$dataInput)))
      write.csv(values$newStudent, df.path)}
  })

  output$dataSet <- renderTable(
    values$newStudent
  )

  output$testTxt <- renderPrint({
    values$newStudent
  })

  output$title <- renderText({
    input$title
  })

  observeEvent(input$recall, {
    values$newStudent <- values$newStudent[-dim(values$newStudent)[1],]
    write.csv(values$newStudent, df.path)
  })
  output$barchart <- renderPlotly({
    shiny::validate(need(values$newStudent,""))
    plot_ly(values$newStudent, x = ~Country, y = ~Numbers, type = 'bar', color=~Gender, colors =
c("#1F618D", "#cc6900")) %>%
      layout(yaxis = list(title = 'Count'), barmode = 'group')
  })
}

shinyApp(ui, server)
```

R-Shiny *Deploy your web app on server*

```
# 1st step, register an account ----
https://www.shinyapps.io/
# 2nd step ----
install.packages('rsconnect')
# 3rd step, go to your account page ----
Copy the following code to your Rstudio commend
```

```
rsconnect::setAccountInfo(name='
                          token=
                          secret=
```

```
# 3rd step, go to your account page ----
rsconnect::deployApp('yourAppDirectory')
```