R Shiny Tutorial

Hanying Ji

Digital Center – Columbia University Library

What is R Shiny

- An R package that makes it easy to build interactive web apps from R
- Aim: Fully interactive visualization
- JavaScript libraries: d3, Leaflet, and Google Charts.

How R Shiny can be used

- Build dashboard - Tell data story by interacting with data and analysis
- Host any apps on a webpage
- Map interaction

Shiny Gallery:
https://shiny.rstudio.com/gallery/

Let's see some example

- Library(shiny)
- runExample("01_hello")

Shiny works in two way:

(1) one script "app.R" which contains:

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

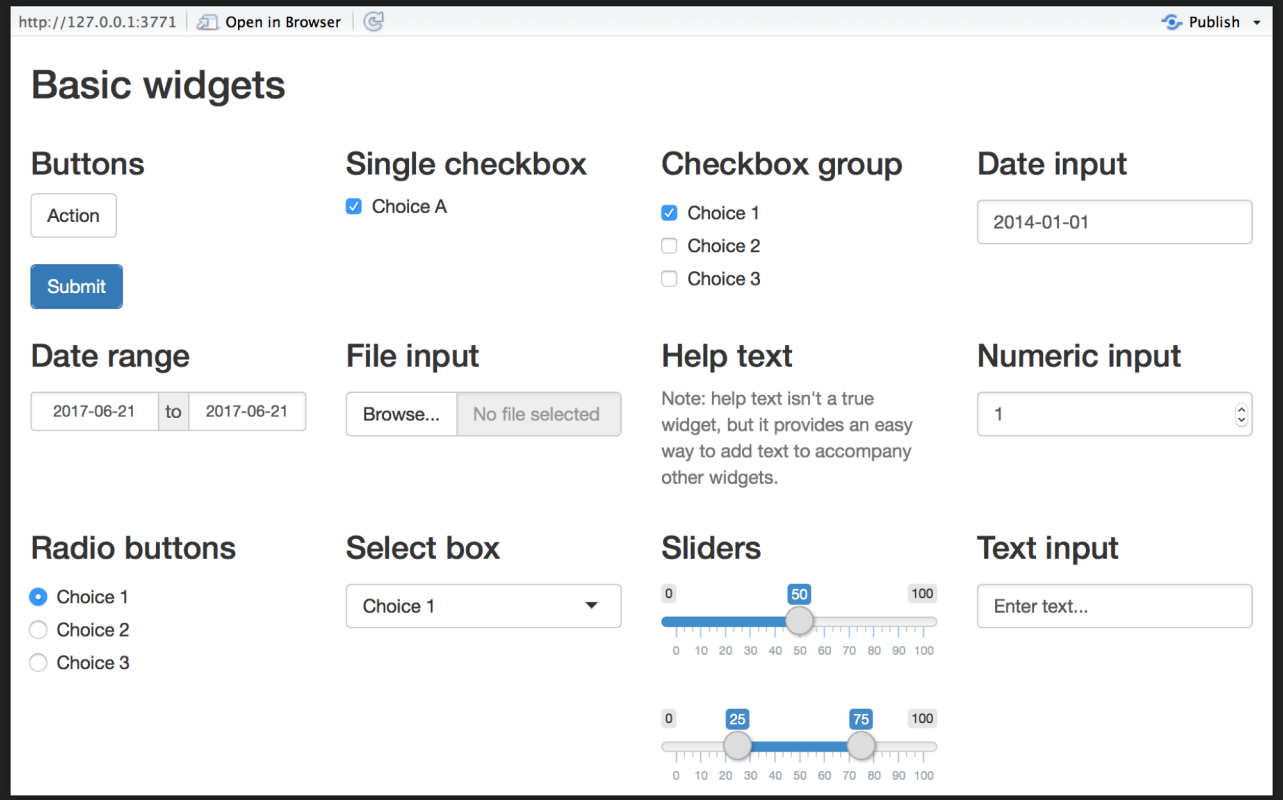
(2) two seperate scripts "ui.R" and "server.R"

ui.R

```
ui <- fluidPage(
        titlePanel("Hello Shiny!"),
        sidebarLayout(              # contains both input and output
            sidebarPanel(),         # usually contains input
            mainPanel()             # usually contains output
        )
)
```

Let's think:
How all the elements organized?
What are inputs, outputs in this example? How they connect?

Control widgets - Input



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

Input - SelectInput

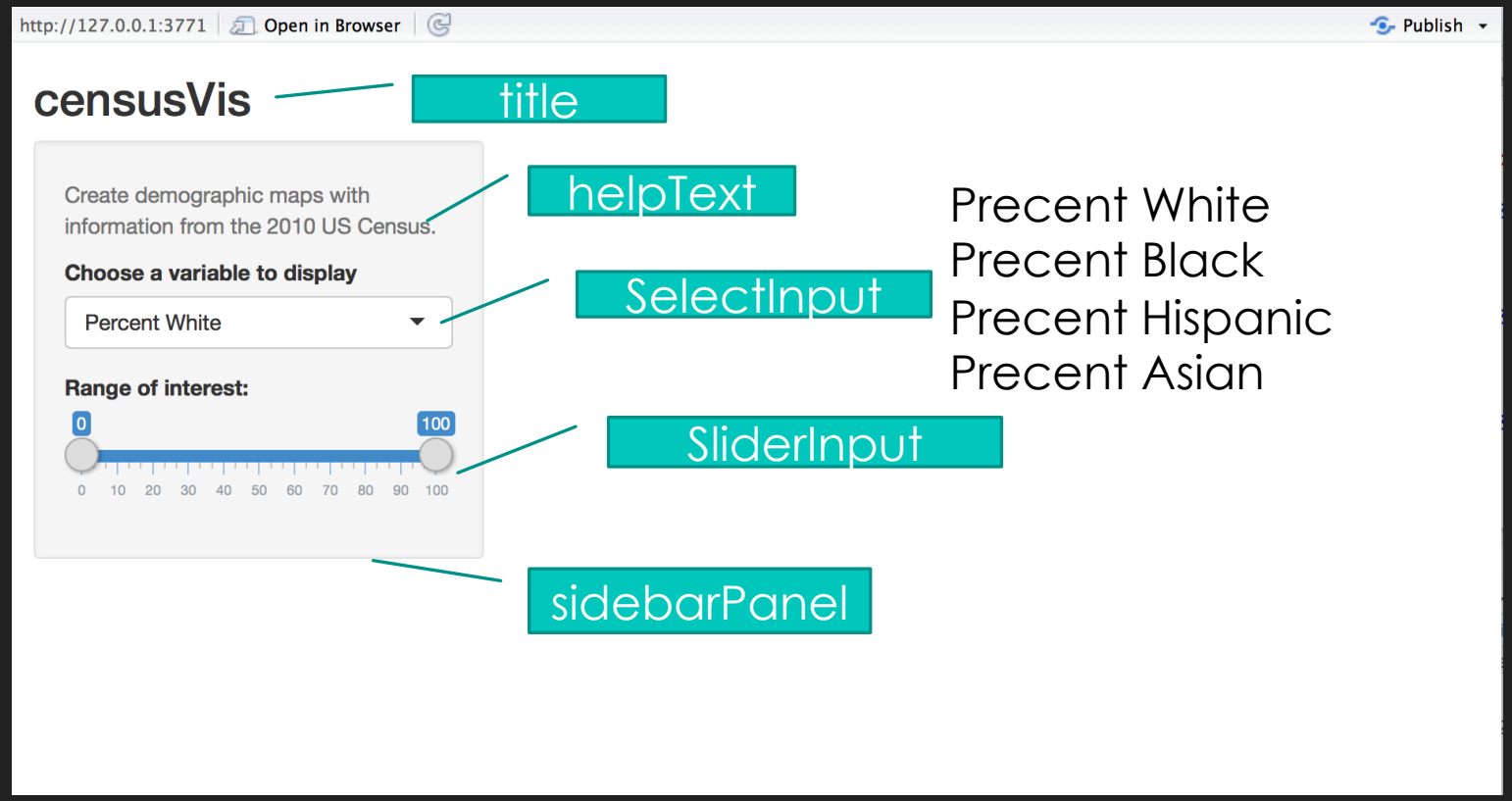
```
selectInput( “select_id”, h3(”This is a label of SelectInput"),
             choices = list("Choice 1" = 1, "Choice 2" = 2, "Choice 3" = 3),
             selected = 1
             )
```

Input - SliderInput

```
sliderInput('slider1', h3('This is a label of SliderInput'),
            min = 0,
            max = 100,
            value = c(25,75)
)
```

Let's do some exercise

- The final ui should look like this



Control widgets - Output

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

- Object can be:
  - input from control widgets(Here comes interactive)
  - Directly type in/ insert in

Output - textOutput

textOutput("name_give_to_input")

Till now, we didn't cover anything about server.R, just think some example of how server.R can be used here.

server.R

```
server <- function(input, output){
    output$weidget_name <- action
}
```

The server can use
(1) the input from ui,
(2) the result of the action(output$weidget_name) can be used in ui by calling the object's name.

That's the reason why it is a function which takes two parameters.

Let’s do some exercise

Paste the input value of the selectInput with “You select ”, then show it in the mainpanel

Hint: The action here called “renderText”

Use Reactive expressions

Reactive expressions let you control which parts of your app update when, which prevents unnecessary computation that can slow down your app.

- A reactive expression saves its result the first time you run it.
- The next time the reactive expression is called, it checks if the saved value has become out of date (i.e., whether the widgets it depends on have changed).
- If the value is out of date, the reactive object will recalculate it (and then save the new result).
- If the value is up-to-date, the reactive expression will return the saved value without doing any computation.

Let's see an example

```
fib <- function(n) ifelse(n<3, 1, fib(n-1)+fib(n-2))

server <- function(input, output) {
    currentFib <- reactive({ fib(as.numeric(input$n)) })
    output$nthValue <- renderText({ currentFib() })
    output$nthValueInv <- renderText({ 1 / currentFib() })
}
```

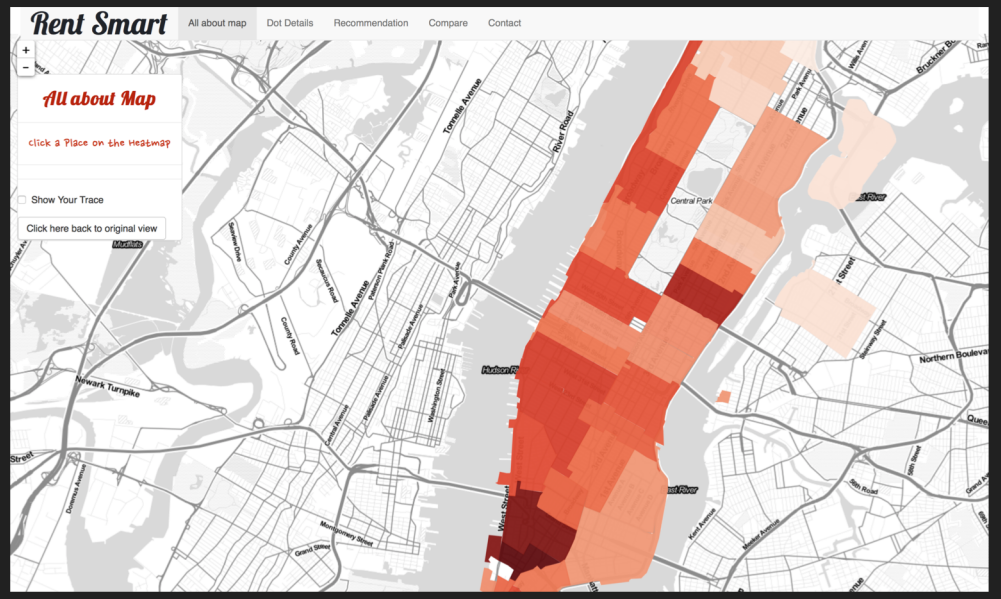
We want the textOutput reactive with the input n but don't slow down our app.

If we don't use reactive object currentFib(), every time we make any change in our app and excute the second line in server function, it will go back to check the input$n no matter it change or not.

reactive can store the value unless it is updated

Topic - Leaflet

- Leaflet is one of the most popular open-source JavaScript libraries for interactive maps. It's used by websites ranging from The New York Times and The Washington Post to GitHub and Flickr, as well as GIS specialists like OpenStreetMap, Mapbox, and CartoDB.



https://github.com/TZstatsADS/Spring2018-Project2-Group3

Basic Usage

Do you still remember how ggplot2 works?

- Create a map widget by calling leaflet().
- Add *layers* (i.e., features) to the map by using layer functions (e.g. addTiles, addMarkers, addPolygons) to modify the map widget.
- Repeat step 2 as desired.
- Print the map widget to display it.

```
leaflet() %>%
        addTiles() %>% # Add default OpenStreetMap map tiles
        addMarkers(lng=174.768, lat=-36.852, popup="The birthplace of R")
```

Reference

- R Shiny

https://shiny.rstudio.com/tutorial/written-tutorial/lesson1/

- Leaflet

https://rstudio.github.io/leaflet/