Introduction to R Shiny

Dr. John Helevston, Lingmei Zhao, Yanjie He

George Washington University

March 30, 2020

Cool Projects with R Shiny

1. Shiny COVID-19 tracker.
2. A Dashboard for Conference Tweets
3. Medcare spending map
4. For more interesting R shiny projects, we can explore the Shiny Gallery and Example Shiny Apps.

Today's Data & New Packages

```
# Today's Data
federal_spending <- read_csv(here::here('data', 'federal_spending_long.csv'))
milk_production <- read_csv(here::here('data', 'milk_production.csv'))



# New Packages
install.packages("shiny")
```

Today's Content

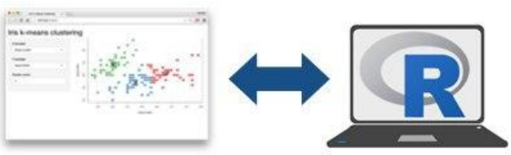
1. R shiny basic structure.
2. Introduction to user interface.
3. Introduction to server.

What is R Shiny?

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

Basics

A Shiny app is a web page (UI) connected to a computer running a live R session (Server).



Users can manipulate the UI, which will cause the server to update the UI's displays.

Structure of a Shiny App

Shiny apps are contained in a single script called app.R. The script app.R lives in a directory and the app can be run with function "runApp". app.R has three components:

- a user interface object
- a server function
- a call to the shinyApp function

App.R

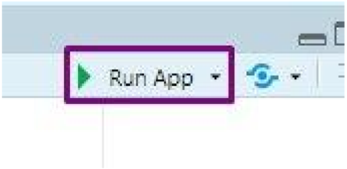
```r
library(shiny)

# User interface
ui <- fluidPage( )

# Server
server <- function(input, output) {    }

# Combines UI and server
shinyApp(ui=ui, server=server)
```

Hello Shiny

```r
library(shiny)

ui <- fluidPage(
    titlePanel("Hello Shiny!"),
    sidebarLayout(
    sidebarPanel(
        sliderInput(inputId = "bins",
                    label = "Number of bins:",
                    min = 1,
                    max = 50,
                    value = 30)
    ),
    mainPanel(
        h3("Bar Plot of Number of Bins"),
        plotOutput(outputId = "distPlot")
    )))

server <- function(input, output){
  output$distPlot <- renderPlot({
    x <- faithful$waiting
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    hist(x, breaks = bins, col = "#75AADB", border="white",
         xlab = "Waiting time to next eruption(in mins)",
         main = "Histogram of waiting times")
  })
}

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

Running an App

- Click the Run App button in the document toolbar



- Use a keyboard shortcut: Cmd/Ctrl + Shift + Enter
- shiny::runApp() with the path to the directory containing app.R

Your Turn

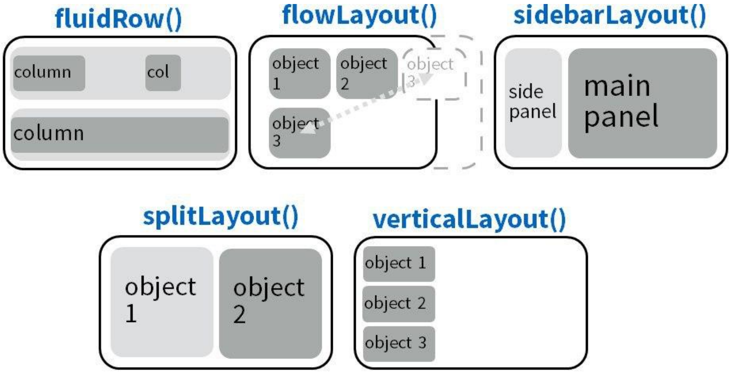
- Open the "helloShinyApp.R" file.
- Change the title from "Hello Shiny!" to "Hello World!"
- Set the minimum value of the slider bar to 5
- Change the histogram border color from "white" to "orange"
- Launch your app and see the changes

User Interface

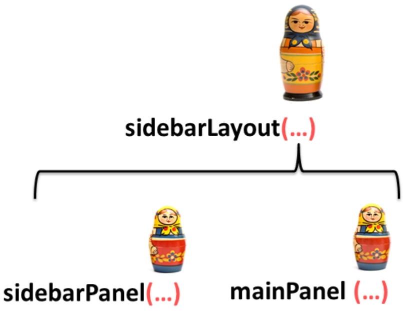
[0] source: Florencia D'Andrea

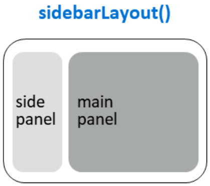
User Interface: layout

Shiny uses the function fluidPage to create a display that automatically adjusts to the dimensions of your user's browser window. You lay out the user interface of your app by placing elements in the fluidPage function.



Sidebar Layout



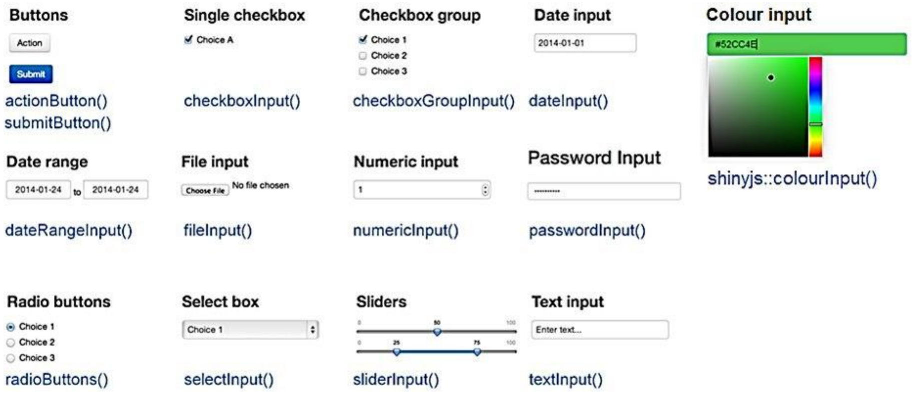


```
ui <- fluidPage(
    titlePanel("title panel"),

    sidebarLayout(
        sidebarPanel("sidebar panel"),
        mainPanel("main panel")
    )
)
```

[0] source: Florencia D'Andrea

User Interface: widgets

What's a widget? A web element that your users can interact with. Widgets provide a way for your users to send messages to the Shiny app.



actionButton()
submitButton()

checkboxInput()

checkboxGroupInput()

dateInput()

shinyjs::colourInput()

dateRangeInput()

fileInput()

numericInput()

passwordInput()

radioButtons()

selectInput()

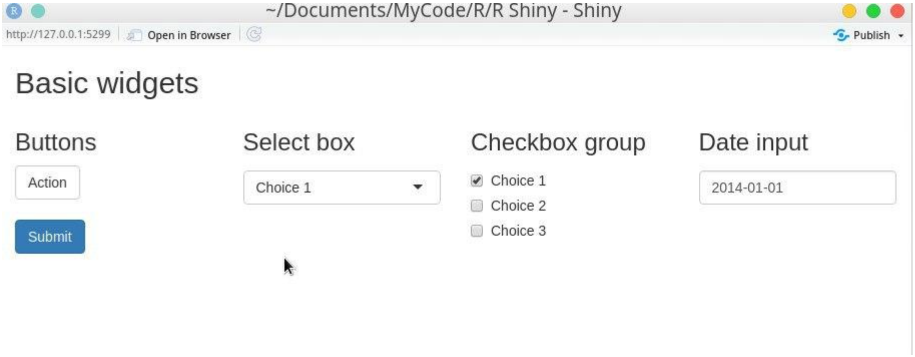
sliderInput()

textInput()

Widgets Example

```
ui <- fluidPage(
    titlePanel("Basic widgets"),
    fluidRow(
        column(3,
                h3("Buttons"),
                actionButton("action", "Action"),
                br(),
                br(),
                submitButton("Submit")),

        column(3,
            selectInput("select", h3("Select box"),
                        choices = list("Choice 1" = 1, "Choice 2" = 2,
                                        "Choice 3" = 3, "Choice 4" = 4))),

        column(3,
                checkboxGroupInput("checkGroup",
                                h3("Checkbox group"),
                                choices = list("Choice 1" = 1,
                                                "Choice 2" = 2,
                                                "Choice 3" = 3),
                                selected = 1)),
        column(3,
                dateInput("date",
                        h3("Date input"),
                        value = "2014-01-01"))
    )
)
```

Widgets Example



Basic widgets

Buttons

Action

Submit

Select box

Choice 1

Checkbox group

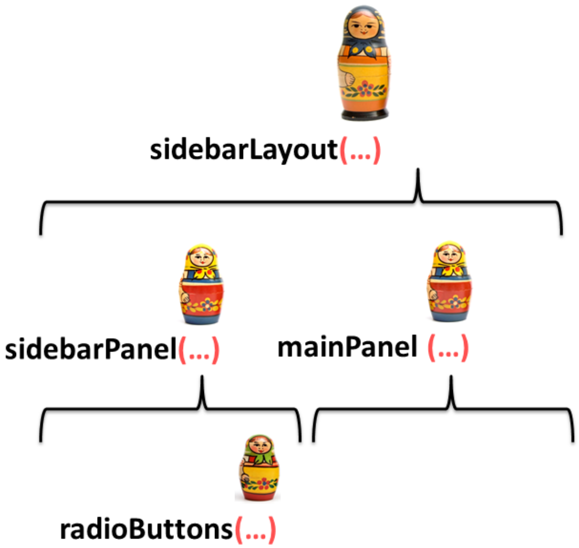
- [x] Choice 1
- [ ] Choice 2
- [ ] Choice 3

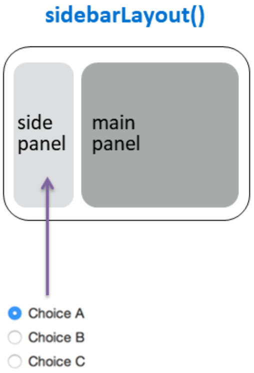
Date input

2014-01-01

User Interface: structure

The structure of the user interface:





Add an R Object to the UI

Shiny provides a family of functions that turn R objects into output for user interface. Each function creates a specific type of output.

Output function	creates
dataTableOutput	DataTable
imageOutput	image
plotOutput	plot
textOutput	text
verbatimTextOutput	text

You can place the output function inside sidebarPanel or mainPanel in the ui.

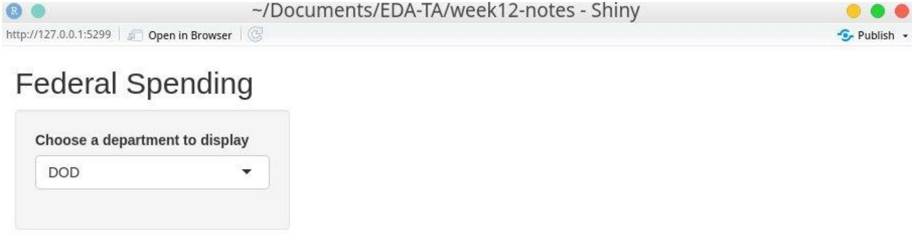
An Example of Building UI

The ui of federal spending display app

```r
library(shiny)

# load the data
federal_spending <- read_csv(here::here('data', 'federal_spending_long.csv'))

ui <- fluidPage(
    titlePanel("Federal Spending"),
    sidebarLayout(
    sidebarPanel(
        selectInput("department", # input name
                    label = "Choose a department to display",
                    choices = c("DOD", "NASA", "DOE", "HHS","NIH","NSF", "USDA",
                                "Interior","DOT", "EPA","DOC","DHS","VA","Other"),
                    selected = "DOD")
    ),
    mainPanel(
        plotOutput("barPlot") # output name
    )
    )
)

server <- function(input, output){

}

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

An Example of Building UI

The ui looks like:



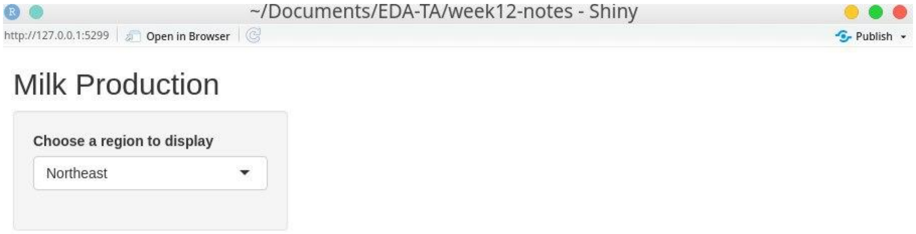
Federal Spending

Choose a department to display

DOD

Your Turn

- Open milkProductionDisplayApp.R file.
- Load the milk_production.csv into this file.
- Create a sidebar layout.
- Name the title "Milk Production"
- Create a selectBox with all the choices of regions in the milk production data frame.
- launch the App.



Milk Production

Choose a region to display

Northeast

Server

The server function builds a list-like object named output that contains all of the code needed to update the R objects in you app. Each R object needs to have its own entry in the list.
You can create an entry by defining a new element for output within the server function, like below. The element should match the name of the reactive element that you created in the ui.

```
    plotOutput("barPlot") # output name in the UI

    server <- function(input, output){
        output$barPlot <- renderPlot({
            # do something
        })
    }
```

Shiny's Render* Functions

Each entry to output should contain the output of one of Shiny's render* functions. Use the render* function that corresponds to the type of reactive object you are making.

render function	creates
renderDataTable	DataTable
renderImage	images
renderPlot	plots
renderText	character strings

Shiny's Render* Function

Explanations of render* function

- Each render* function takes a single argument: an R expression surrounded by braces, {}.
- The R expression is a set of instructions that you give Shiny to store for later. Shiny will run the instructions when you launch your app.
- The expression should return the object you want to display(a piece of text, a plot, a data frame).

Use Widgets Values

You can make an interactive app by asking Shiny to call a widget value when it builds the content(text, plot, data frame, ect).
The server function has two arguments: output and input.

- Output is a list-like object that stores instructions for building the R objects in your app.
- input is a second list-like object. It stores the current values of all of the widgets in you app. These values will be saved under the names that you gave the widgets in your ui.

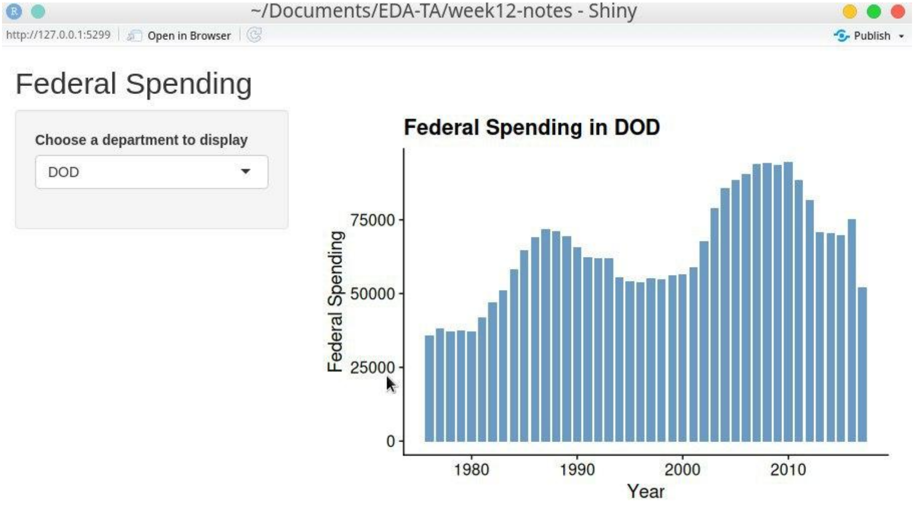
```
server <- function(input, output) {
    output$selected_department <- renderText({
        paste("You have selected ", input$department)
    })
}
```

An Example of Bulding Server

The server of federal spending display app.

```
server <- function(input, output){
    output$barPlot <- renderPlot({
    inputDepartment <- switch (input$department,
                               "DOD" = "DOD",
                               "NASA" = "NASA",
                               "DOE" = "DOE",
                               "HHS" = "HHS",
                               "NIH" = "NIH",
                               "NSF" = "NSF",
                               "USDA" = "USDA",
                               "Interior" = "Interior",
                               "DOT" = "DOT",
                               "EPA" = "EPA",
                               "DOC" = "DOC",
                               "DHS" = "DHS",
                               "VA" = "VA",
                               "Other" = "Other"
    )
    federal_spending %>% filter(department == inputDepartment) %>%
    ggplot(aes(x = year, y = rd_budget)) +
    geom_col(fill = "steelblue", width = 0.7, alpha = 0.8) +
    theme_half_open(font_size = 18) +
    labs(x = "Year",
         y = "Federal Spending",
         title = paste("Federal Spending in", inputDepartment))

  })
}
```

Federal Spending Display App



Federal Spending

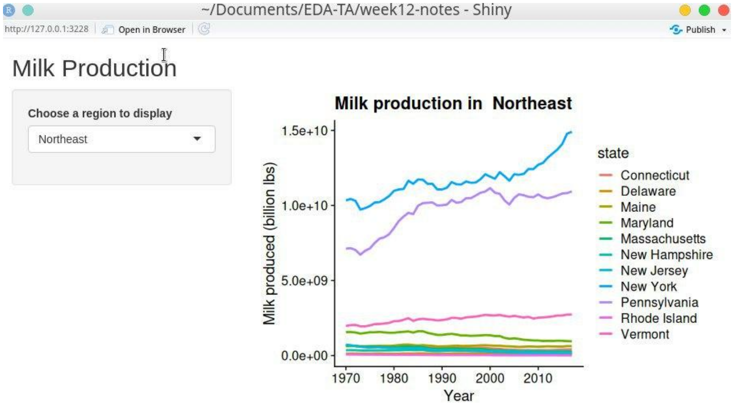
Choose a department to display

DOD

Your Turn

- Creata a plot to display the trend(only use line) of milk production which can choose different region.
- Launch the app.

The app will looks like:



Learning Material

The best tutorial we found so far is official shiny tutorial from R studio.