Shiny App

What's Shiny?

Shiny is the library to make web applications.

The sample application shows the histogram. It has a slider to change the number of bins of the histogram and you can vidualize it immediately.

The Structure

To execute Shiny, we need two R files, `ui.R` and `server.R`. Only `app.R` can execute an application as well, but it is a bit messy, especially for a long source code.

- `ui.R`: creates UI (user interface)
  - checkbox to let user choose
  - input text form
  - vidualizations
  - tab to change display
- `server.R`: how to treat data
  - calculate statistics
  - make graphs
  - prediction

For sample code,

`ui.R`

```
library(shiny)

# Define UI for application that draws a histogram
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")
        )
    )
)
```

server.R

```
library(shiny)

# Define server logic required to draw a histogram
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')
    })
}
```

1. receive the number of bins on `ui.R`

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

`sliderInput()` decides how to input data into a slider. For text input, use `textInput()`. For check box input, use `checkboxInput()`.
`"bins"` of `sliderInput()` defines a name of a variable. The variable `"bins"` contains what a user input.

2. use the information from (1) to create a histogram on `server.R`

```
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')
    })
```

`input$bins` receives `"bins"` from `ui.R` (`input$name_of_variable_from_ui`) as well as `output$distPlot` receives `"distPlot"` from `server.R`, so `output$distPlot` contains a histogram.

3. reflect the histogram from (2) to display on UI on `ui.R`

```
mainPanel(
           plotOutput("distPlot")
        )
```

`plotOutput()` is a function that displays a graph.

If we use just `app.R`, it has

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

at the end.

Output Functions

Output	function in ui.R	function in server.R
table	tableOutput()	renderTable()
data table	dataTableOutput()	renderDataTable()
HTML	htmlOutput(), uiOutput()	renderUI()
image	imageOutput()	renderImage()
text	textOutput(), verbatimTextOutput()	renderText(). renderPrint()

Reactive Output

- Bad Example

```
library(shiny)

# Define UI for application that draws a histogram
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),
            selectInput("color", "select color",
                        c("red", "blue", "green", "black"))
        ),
```

```
        # Show a plot of the generated distribution
        mainPanel(
           plotOutput("distPlot")
        )
    )
)

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

    output$distPlot <- renderPlot({
        x = faithful[, 2]
        bins <- seq(min(x), max(x), length.out = input$bins + 1)

        hist(faithful[, 2], breaks = bins, col = input$color, border = "white")
    })
}

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

This executes inside `renderPlot()` every time, so it consumes lots of memory.

- Good Example Use `reactive()` outside `renderPlot()`.

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

    bins <- reactive({
        x = faithful[, 2]
        return(seq(min(x), max(x), length.out = input$bins + 1))
        })

    output$distPlot <- renderPlot({

        hist(faithful[, 2], breaks = bins(), col = input$color, border = "white")
    })
}
```

This reduces memory because it doesn't necessarily change the bins when the color is changed. Since `bins` is now a function, it needs parenthesis to call (`breaks = bins()`).

Another way to reduce memory is to use `isolate()`. It always executes all code inside `render~({})` when any of inputs is changed, but `isolate()` enables to avoid that behavior.

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

    output$distPlot <- renderPlot({
        x <- faithful[, 2]
        bins <- seq(min(x), max(x), length.out = input$bins + 1)

        hist(faithful[, 2], breaks = bins, col = isolate(input$color), border = "white")
    })
}
```

This doesn't change the histogram immediately when color is changed; it changes the color when the bins is also changed because it executes `renderPlot()`.

Customize UI

UI has mainly three parts: - Page - Layout - Panel

Page

function	
fluidPage()	create a page with fluid layout
fixedPage()	create a page with fixed layout
navbarPage()	create a page with a navigation bar

Layout

`sidebarLayout()` is commonly used; its grid is divided by 4:8 (side : main).

```
sidebarLayout(sidebarPanel, mainPanel,
              position = c("left", "right"), fluid = TRUE)
```

`position` decides that side bar is on the left side and main panel is on the right side. `fluid = TRUE` enables to change the layout dynamically by the window size.

`fluidRow()` creates rows and `column()` creates columns in the rows.

```
ui <- fluidPage(

    titlePanel("fluid row sample"),

    fluidRow(
        column(4, # number of grid (total is 12 grids)
               sliderInput("obs_1", "Number of observations:",
                           min = 0,
                           max = 1000,
                           value = 500) # default input is 500
        ),
        column(4,
               sliderInput("obs_2", "Number of observations:",
                           min = 0,
                           max = 1000,
                           value = 500)
        ),
        column(4, # 4 + 4 + 4 =12 grids
               mainPanel(
                   plotOutput("distPlot")
               )
        )
    )
)
```

Panel

function	
absolutePanel()	indicate a coordinate to make a panel
tabsetPanel()	create a panel with tab, used with `tabPanel`
tabPanel()	create a tab in a panel created by `tabsetPanel`
sidebarPanel()	create a side bar, used with `sidebarLayout`, default 4 grids
mainPanel()	create a main panel, used with `sidebarLayout`, default 8 grids
conditionalPanel()	display a panel only when satisfies a condition
navlistPanel()	close to `navbarPage`, create a navigation list on the top of left panel
wellPanel()	create a panel with gray background

```
shinyUI({ # or ui <-
  # 1. page
  fluidPage(...)({ # or navbarPage etc
    # 2. layout
    sidebarLayout(...)({ # or use fluidRow to set grids individually
      # 3. panel
      mainPanle(...)
    })
  })
})
```

UI Input

widget	
checkboxGroupInput()	create multiple check boxes
checkboxInput()	create a check box
dateInput()	input a date
dateRangeInput()	select a date range
numericInput()	input a number
selectInput()	select from a list of options
sliderInput()	input a value on a slider
textInput()	input a text

```
ui <- fluidPage(

    titlePanel("INPUTS"),

    fluidRow(
        column(4,
               h3("checkboxInput"), # title
               checkboxInput(inputId = "checkbox",
                             label = "Choice A", # label of check box
                             value = TRUE)
               ),
        column(4,
               checkboxGroupInput(inputId = "checkboxGroupInput",
```

```
                                label = h3("checkboxGroupInput"), # title
                                choices = list("Choice 1" = 1,
                                               "Choice 2" = 2,
                                               "Choice 3" = 3),
                                selected = 1,
                                inline = TRUE)
                ),
            column(4,
                dateInput(inputId = "date",
                          label = h3("dateInput"),
                          value = "2016-01-01")
                ),
        ),

        fluidRow(
            column(4,
                dateRangeInput("dateRangeInput",
                               h3("dateRangeInput"))
                ),
            column(4,
                textInput("text", h3("textInput"), value = "Enter text...")
                ),
            column(4,
                numericInput("num", h3("numericInput"), value = 1)
                ),
        ),

        fluidRow(
            column(4,
                selectInput("select", h3("selectInput"),
                            choices = list("Choice 1" = 1,
                                           "Choice 2" = 2,
                                           "Choice 3" = 3,
                                           "Choice 4" = 4),
                            selected = 1)
                ),
            column(4,
                sliderInput("sliderInput1", h3("sliderInput1"),
                            min = 0, max = 100, value = 50),
                sliderInput("sliderInput2", h3("sliderInput2"),
                            min = 0, max = 100, value = c(25, 75) # range of input
                            )
                )
        )
)

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

Arguments

- checkboxInput()

arguments	
inputId	ID to receive a value on server.R
label	name of choice. empty if `NULL`
value	default setting. checked if `TRUE`
width	change width

- `checkboxGroupInput()`

arguments	
inputID	ID to receive a value on server.R
label	title of check box. empty if `NULL`
choices	list of choices. “display” = value
selected	default choice
inline	default is `FALSE`. display check box horizontally if `TRUE`
width	change width

- `dateInput()`

arguments	
inputID	
label	title
value	default value. format: “yyyy-mm-dd”
startview	default is “month”. can be “year” or “decade”
format	default is “yyyy-mm-dd”
min, max	min, max of date
weekstart	0 = Sunday, 6 = Saturday
language	default is “en” (English). Japanese: “ja”
autoclose	closes a window after choose a date if `TRUE`

- `dateRangeInput()`

arguments	
inputID	
label	title
start	default start date. format: “yyyy-mm-dd”
end	default end date. format: “yyyy-mm-dd”
startview	
min, max	
weekstart	
separator	defalut is “to”. how to connect start date and end date
language	
width	
autoclose	

- `numericInput()`

arguments	
inputID	
label	title
value	default value
min	min value
max	max value
step	number of step
width	

- `sliderInput()`

arguments	
inputID	
label	title
min	min value of a slider
max	max value of a slider
value	default value. can be a vector to set a range
step	number of step
round	round a number if `TRUE`
ticks	ticks disappears if `FALSE`
animate	animation is `TRUE`
width	

- `textInput()`

arguments	
inputId	
label	title
value	default value
width	

- `selectInput()`

arguments	
inputID	
label	title
choices	list of choices. “display” = value
selected	default choice
multiple	allows multiple choices
width	

CSS/JavaScript/Image

To execute an app, ui.R and server.R are in the same directory. If you want to use CSS or JavaScript files, or images, create “www” directory in the same directory with ui.R and server.R and put those files in “www”

directory.

```
ui.R
server.R
www/
  |- sample.css
  |- sample.js
  |- sample.jpg
```

Insert Images

Same as HTML.

ui.R

```
img(src = "name of file", height = "height", width = "width")
```

Insert CSS

Create a CSS file. The following changes the background color.

www/styles.css

```
body {
  background-color:#b0c4de;
}
```

ui.R

```
ui <- fluidPage(
        tags$head(tags$link(rel = "stylesheet", type = "text/css", href = "styles.css")),

        titlePanel("Old Faithful Geyser Data"),

        sliderLayout(
          sidebarPanel(
            sliderInput("bins", "Number of bins",
                        min = 1, max = 50, value = 30),
            img(src = "sample.jpg", height = 70, width = 90)
          ),

          mainPanel(plotOutput("distPlot")
          )
        )
)
```

Insert JavaScript

Almost the same as images and CSS files. The following code changes the background color when clicking texts.

www/color_change.js

```
function changeBG(color) {
  document.body.style.backgroundColor = color;
}
```

ui.R

```
ui <- fluidPage(
        tags$head(tags$link(rel = "stylesheet", type = "text/css", href = "styles.css"),
        tags&script(src = "color_change.js")), # added

        titlePanel("Old Faithful Geyser Data"),

        sliderLayout(
          sidebarPanel(
            sliderInput("bins", "Number of bins",
                        min = 1, max = 50, value = 30),
            img(src = "sample.jpg", height = 70, width = 90),

            a(href = "javascript:changeBG('red')", "RED"),
            a(href = "javascript:changeBG('blue)", "BLUE"),
            a(href = "javascript:changeBG('green')", "GREEN"),
            a(href = "javascript:changeBG('#b0c4de')", "RESET")
          ),

          mainPanel(plotOutput("distPlot")
          )
        )
)
```

global.R

Define variables or functions commonly refered in ui.R and server.R.