Northeastern University
College *of* Engineering

IE6600 Computation and Visualization for Analytics

Introduction to R-Shiny

Zhenyuan Lu

1.Introduction

R-Shiny *Introduction*

Shiny is an R package that makes it easy to build interactive web applications (apps) straight from R, which means you don’t need to know D3.js, JavaScript or other languages, e.g. CSS, Jquery, etc., *but if you know a little bit of HTML/CSS/JS which can make apps more fancy*

Why Shiny?

R Studio

Shiny

Dash

D3.js
JavaScript/TypeScript
HTML
CSS

Installation

R-Shiny *Installation*

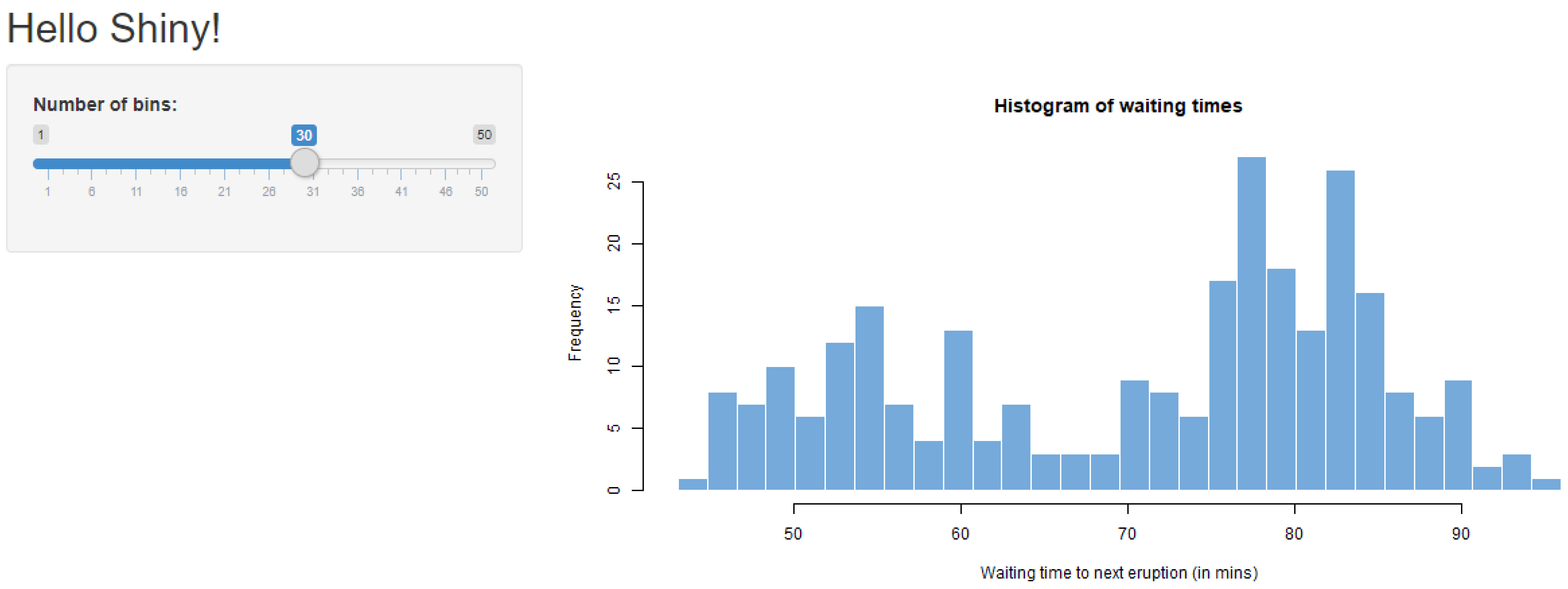
```
install.packages("shiny")
```

2.Basic Concept

R-Shiny *Internal examples from Rshiny packages*

```r
library(shiny)
runExample("01_hello")
```

R-Shiny *Example*

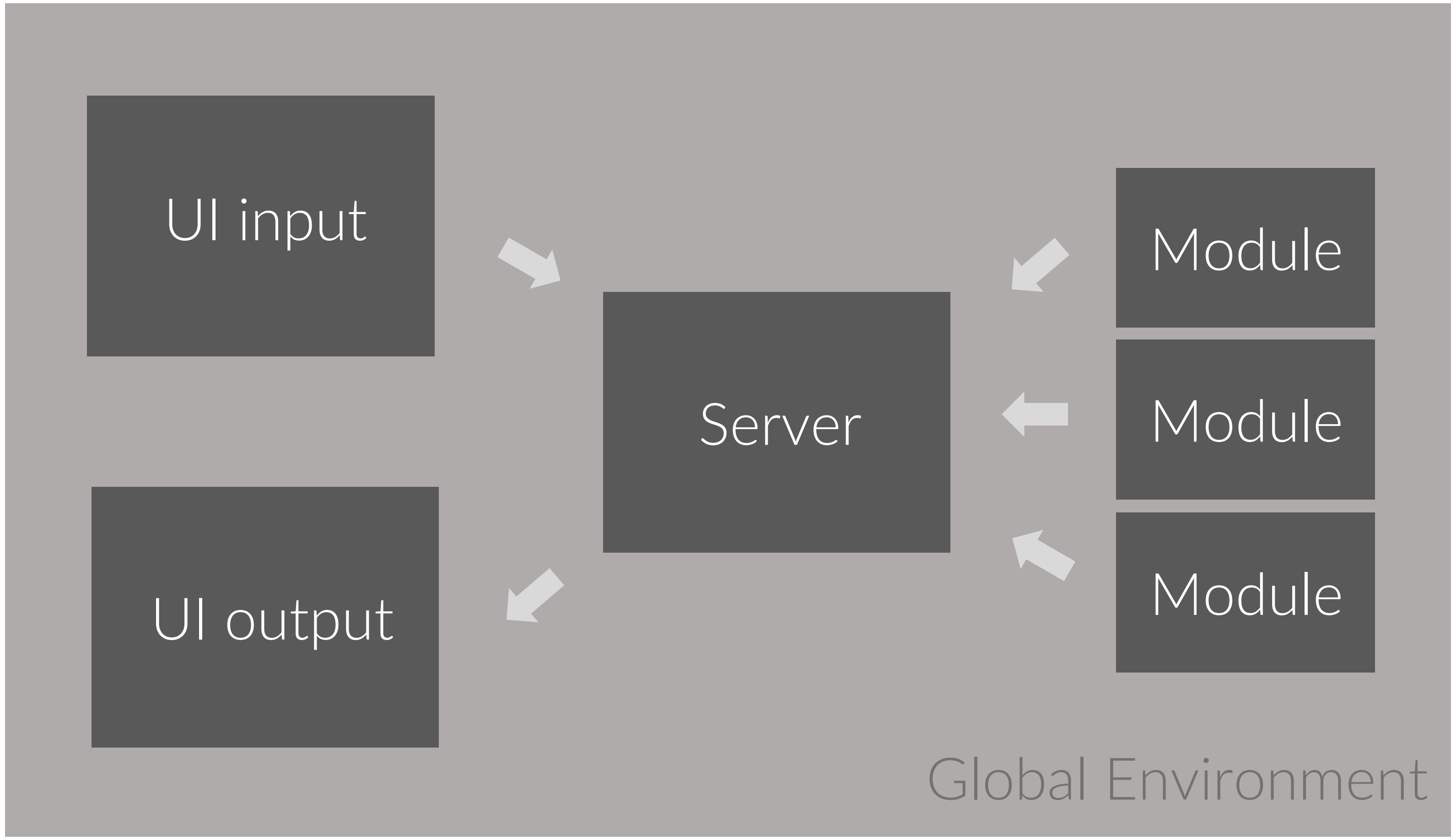

Hello Shiny!

Number of bins: 30
1
50
1
6
11
16
21
26
31
36
41
46
50
Histogram of waiting times
Frequency
0
5
10
15
20
25
50
60
70
80
90
Waiting time to next eruption (in mins)

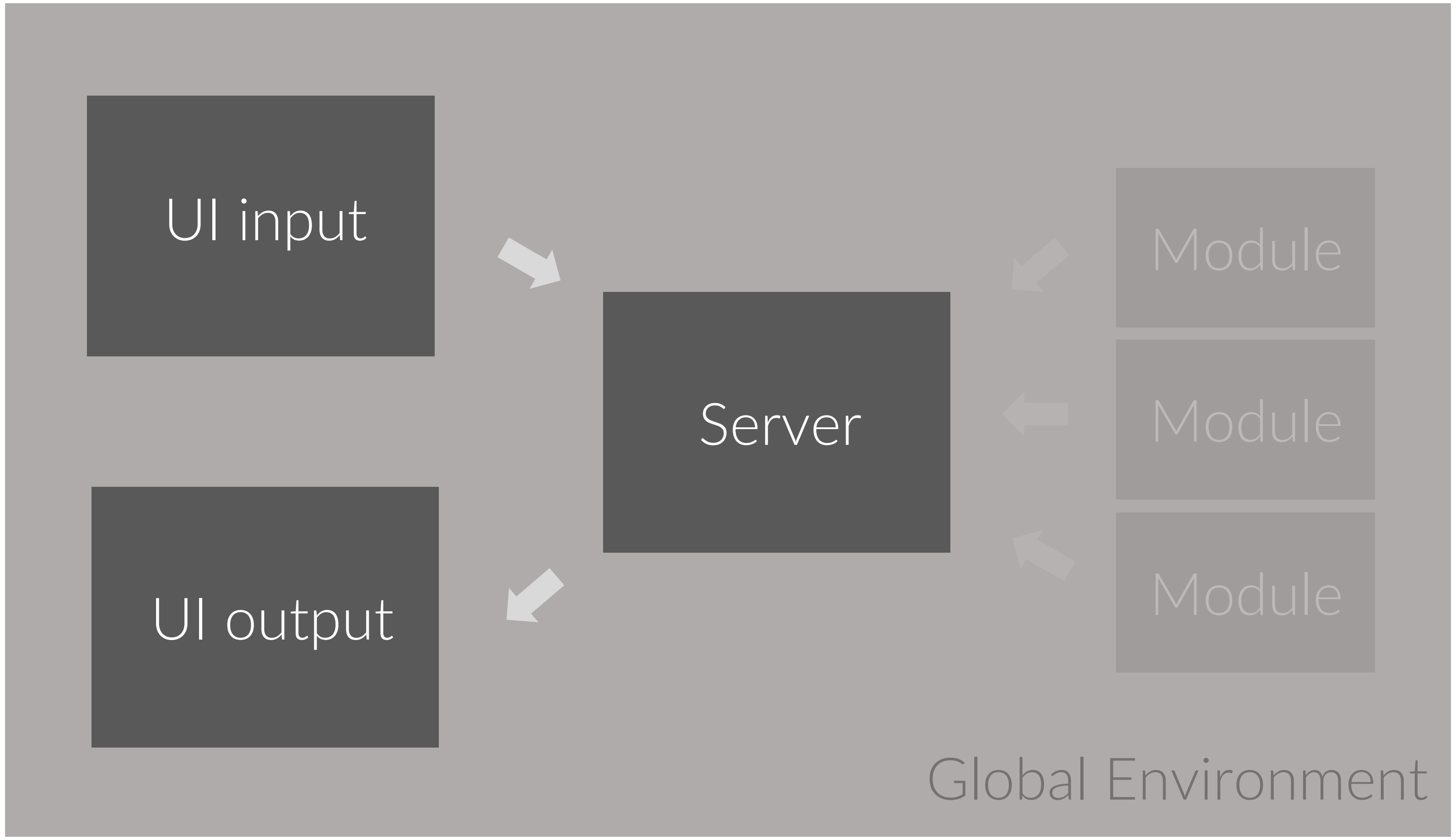
R-Shiny *Internal examples from Rshiny packages*

```r
runExample("02_text")        # tables and data frames
runExample("03_reactivity") # a reactive expression
runExample("04_mpg")         # global variables
runExample("05_sliders")     # slider bars
runExample("06_tabsets")     # tabbed panels
runExample("07_widgets")     # help text and submit buttons
runExample("08_html")        # Shiny app built from HTML
runExample("09_upload")      # file upload wizard
runExample("10_download")    # file download wizard
runExample("11_timer")       # an automated timer
```

R-Shiny *Basic Concept*



R-Shiny *Basic Concept*



R-Shiny *Build a user interface*

Shiny App Template
The shortest viable shiny app

```r
library(shiny)
# Define UI ----
ui <- fluidPage( )
# Define server logic ----
server <- function(input, output) { }
# Run the app ----
shinyApp(ui = ui, server = server)
```

R-Shiny *Basic Components*

1st way: all in one file app.R

- User interface object
- A server function
- A call to the shinyApp function

R-Shiny *Basic Components*

1st way: all in one file app.R

```
library(shiny)
# Define UI ----
ui <- fluidPage( )
# Define server logic ----
server <- function(input, output) { }
# Run the app ----
shinyApp(ui = ui, server = server)
```

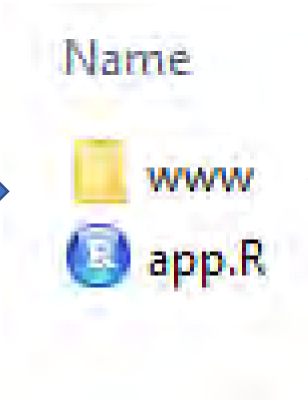
R-Shiny *Working Directory*

Windows

```
E:\youAppName\(contained)
```

Mac

```
~youAppName\(contained)
```



For storing other files, img, css, etc.

You shiny app (or it can be ui.R, server.R, and global.R

2nd way: contained in three different .R files

- ui.R
- server.R
- global.R

R-Shiny *Basic Components*

2nd way: contained in three different .R files

```r
# global.R ----
library(shiny)
```

```r
# Define UI ----
ui <- fluidPage( )
```

```r
# Define server logic ----
server <- function(input, output) { }
```

R-Shiny *Working Directory*

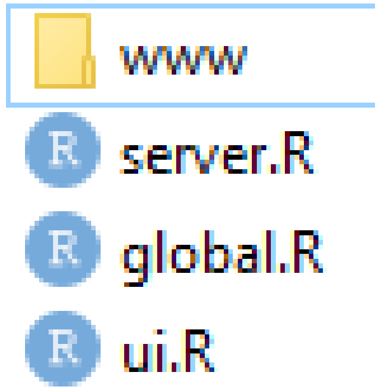
Windows

`E:\youAppName\(contained)`

yourAppName

Mac

`~youAppName\(contained)`


www
server.R
global.R
ui.R

For storing other files, img, css, etc.

You shiny app (ui.R, server.R, and global.R

R-Shiny

1st way: all in one file app.R

- User interface object
- A server function
- A call to the shinyApp function

```
library(shiny)

# Define UI ----
ui <- fluidPage( )
# Define server logic ----
server <- function(input, output) { }
# Run the app ----
shinyApp(ui = ui, server = server)
```

2nd way

Recommend!

- ui.R
- server.R
- global.R

```
# global.R ----
library(shiny)
```

```
# Define UI ----
ui <- fluidPage( )
```

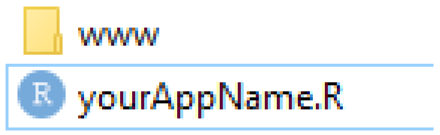
```
# Define server logic ----
server <- function(input, output) { }
```

R-Shiny *Exercise 0.1*

1. Create a folder with *#yourAppName#* (you name it)

yourAppName

2. Then create one folder named www and one *#yourAppName#* .R file


www
yourAppName.R

3. Write the following code to *#yourAppName#* .R file

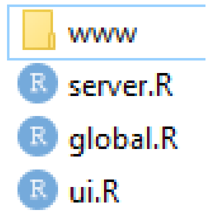
```r
library(shiny)

# Define UI ----
ui <- fluidPage( )
# Define server logic ----
server <- function(input, output) { }
# Run the app ----
shinyApp(ui = ui, server = server)
```

R-Shiny *Exercise 0.2*

1. Create a folder with *#yourAppName2#* (you name it)

yourAppName

2. Then create one folder named www and three .R files



www
server.R
global.R
ui.R

3. Put the following code into global.R, ui.R, and server.R, respectively

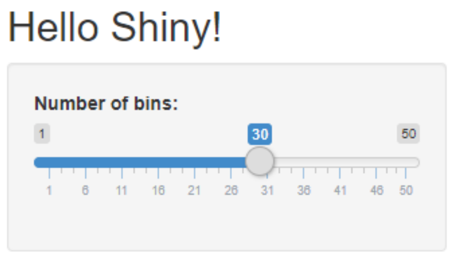
```
library(shiny)

# Define UI ----
ui <- fluidPage( )
# Define server logic ----
server <- function(input, output) { }
```

R-Shiny *One Example*

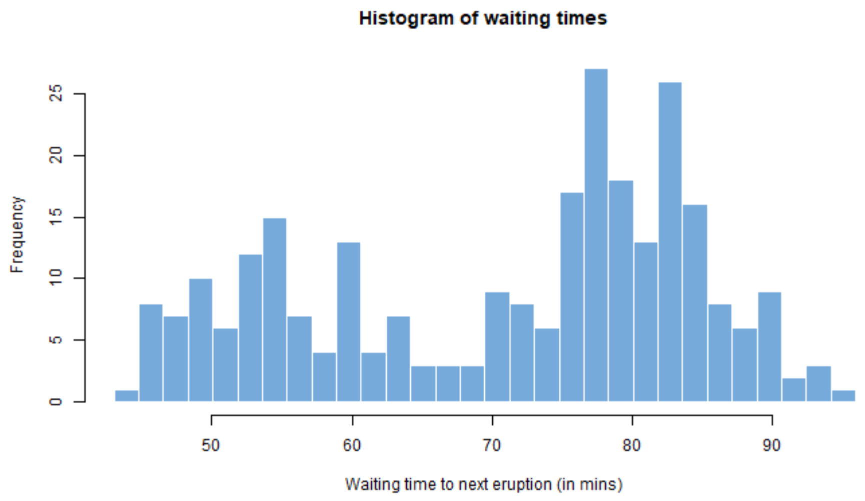
```
library(shiny)
runExample(“01_hello")
```

R-Shiny *UI for “Hello Shiny”*

```r
library(shiny)

# Define UI for app that draws a histogram ----
ui <- fluidPage(

  # App title ----
  titlePanel("Hello Shiny!"),

  # Sidebar layout with input and output definitions -
  sidebarLayout(

    # Sidebar panel for inputs ----
    sidebarPanel(

      # Input: Slider for the number of bins ----
      sliderInput(inputId = "bins",
                  label = "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)

    ),

    # Main panel for displaying outputs ----
    mainPanel(

      # Output: Histogram ----
      plotOutput(outputId = "distPlot")

    )
  )
)
```

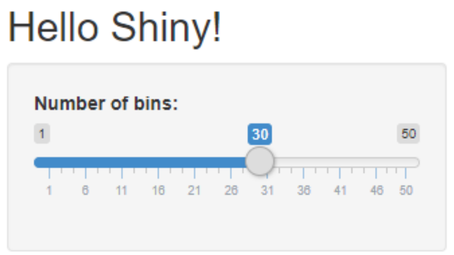

Hello Shiny!

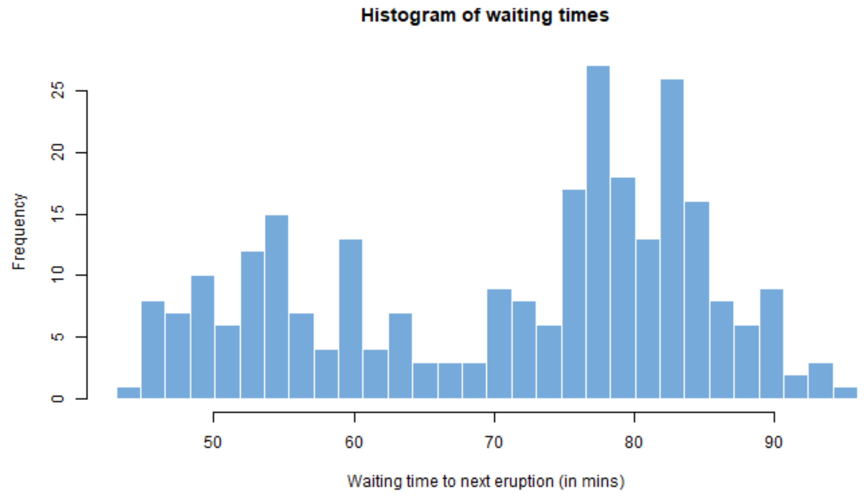
Number of bins: 30



R-Shiny *Server for "Hello Shiny"*

```r
# Define server logic required to draw a histogram ----
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")

    })

}
```





R-Shiny *Server for "Hello Shiny"*

UI

```r
library(shiny)

# Define UI for app that draws a histogram ----
ui <- fluidPage(

  # App title ----
  titlePanel("Hello Shiny!"),

  # Sidebar layout with input and output definitions ----
  sidebarLayout(

    # Sidebar panel for inputs ----
    sidebarPanel(

      # Input: Slider for the number of bins ----
      sliderInput(inputId = "bins",
                  label = "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)

    ),

    # Main panel for displaying outputs ----
    mainPanel(

      # Output: Histogram ----
      plotOutput(outputId = "distPlot")

    )
  )
)
```

Server

```r
# Define server logic required to draw a histogram ----
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")

    })

}
```

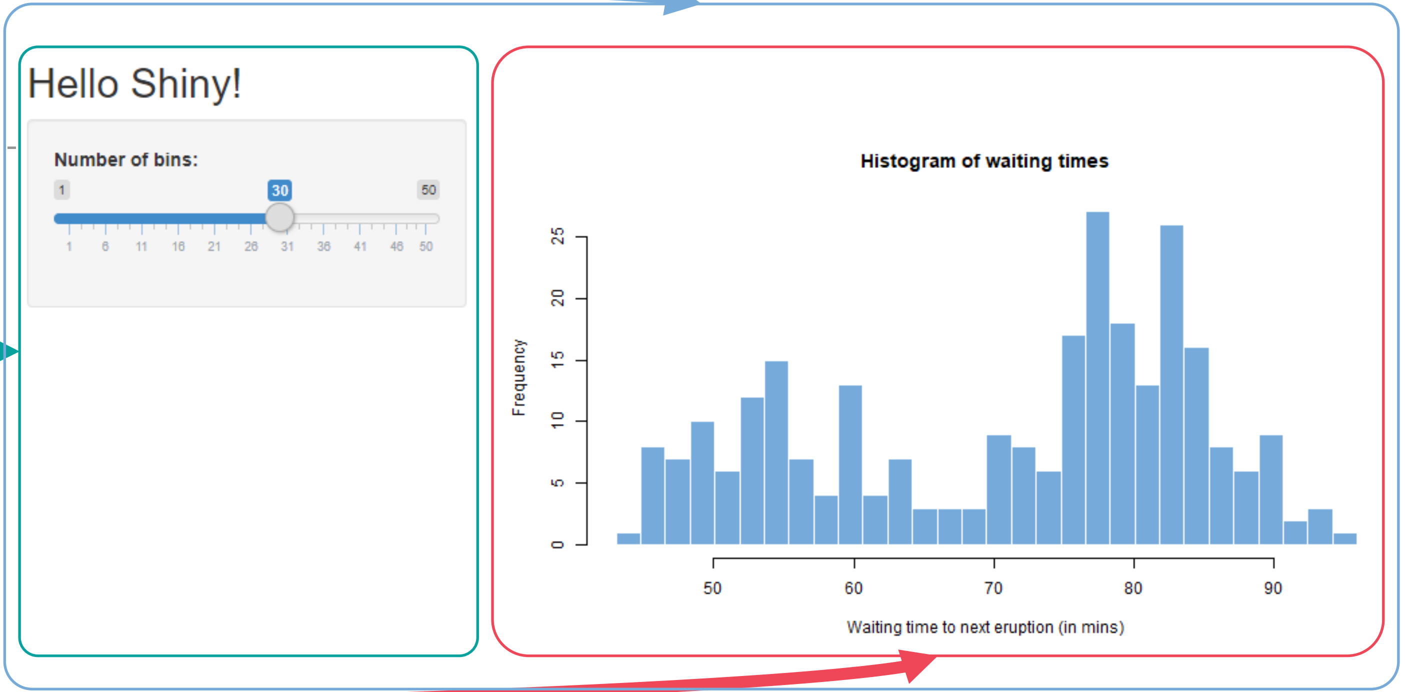
Run App

```r
shinyApp(ui, server)
```

3.VERY Basic Syntax

R-Shiny *Page structure*

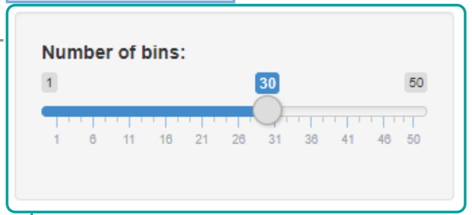
```
library(shiny)

# Define UI for app that draws a histogram ----
ui <- fluidPage(

  # App title ----
  titlePanel("Hello Shiny!"),

  # Sidebar layout with input and output definitions -
  sidebarLayout(

    # Sidebar panel for inputs ----
    sidebarPanel(

      # Input: Slider for the number of bins ----
      sliderInput(inputId = "bins",
                  label = "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)

    ),

    # Main panel for displaying outputs ----
    mainPanel(

      # Output: Histogram ----
      plotOutput(outputId = "distPlot")

    )
  )
)
```

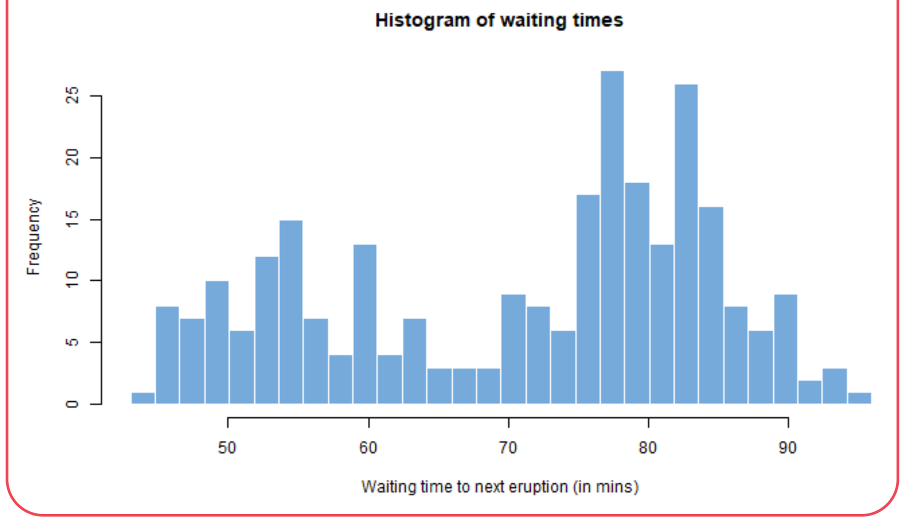


R-Shiny *UI components*

```r
library(shiny)

# Define UI for app that draws a histogram ----
ui <- fluidPage(

  # App title ----
  titlePanel("Hello Shiny!"),

  # Sidebar layout with input and output definitions -
  sidebarLayout(

    # Sidebar panel for inputs ----
    sidebarPanel(

      # Input: Slider for the number of bins ----
      sliderInput(inputId = "bins",
                  label = "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)

    ),

    # Main panel for displaying outputs ----
    mainPanel(

      # Output: Histogram ----
      plotOutput(outputId = "distPlot")

    )
  )
)
```

Hello Shiny!





R-Shiny *Server for “Hello Shiny”*

```r
library(shiny)

# Define UI for app that draws a histogram ----
ui <- fluidPage(

  # App title ----
  titlePanel("Hello Shiny!"),

  # Sidebar layout with input and output definitions ----
  sidebarLayout(

    # Sidebar panel for inputs ----
    sidebarPanel(

      # Input: Slider for the number of bins ----
      sliderInput(inputId = "bins",
                  label = "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)

    ),

    # Main panel for displaying outputs ----
    mainPanel(

      # Output: Histogram ----
      plotOutput(outputId = "distPlot")

    )
  )
)
```

```r
# Define server logic required to draw a histogram ----
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, server)
```

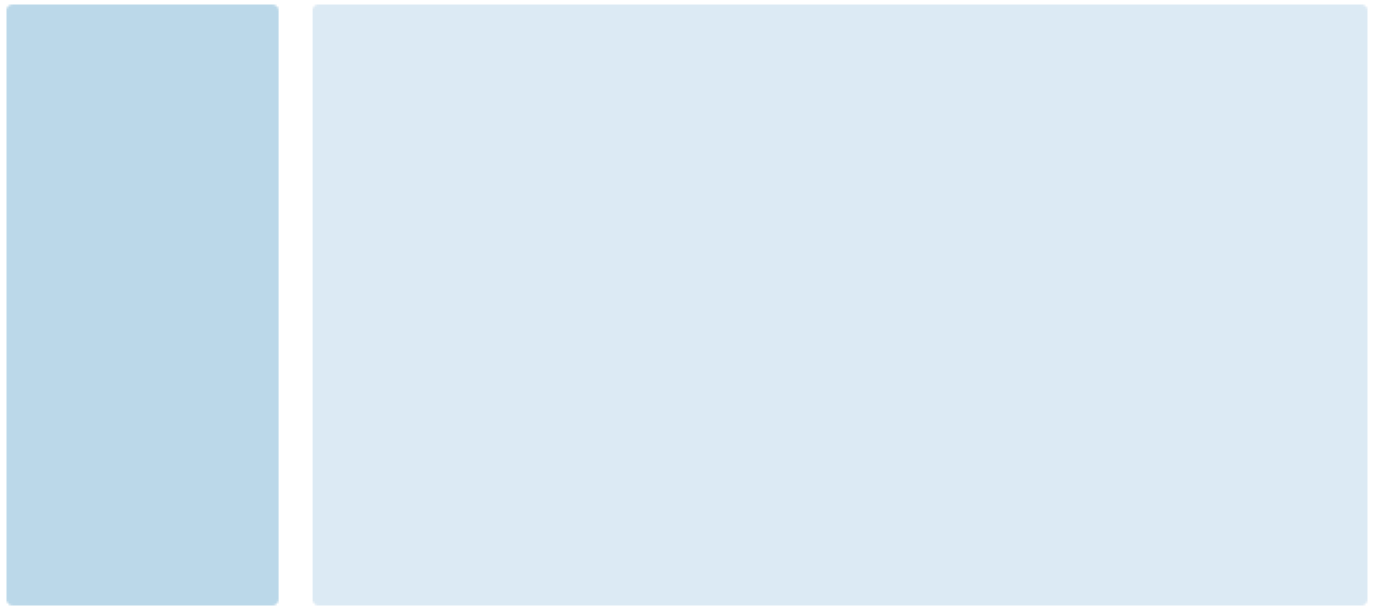
Input variables and other variables ->
server functions, rendering ->
output variables

R-Shiny *Grid system for UI*

The Bootstrap grid system utilizes 12 columns which can be flexibly subdivided into rows and columns. To create a layout based on the fluid system you use the `fluidPage()` function. To create rows within the grid you use the `fluidRow()` function; to create columns within rows you use the `column()` function.

For example, consider this high level page layout (the numbers displayed are columns out of a total of 12):

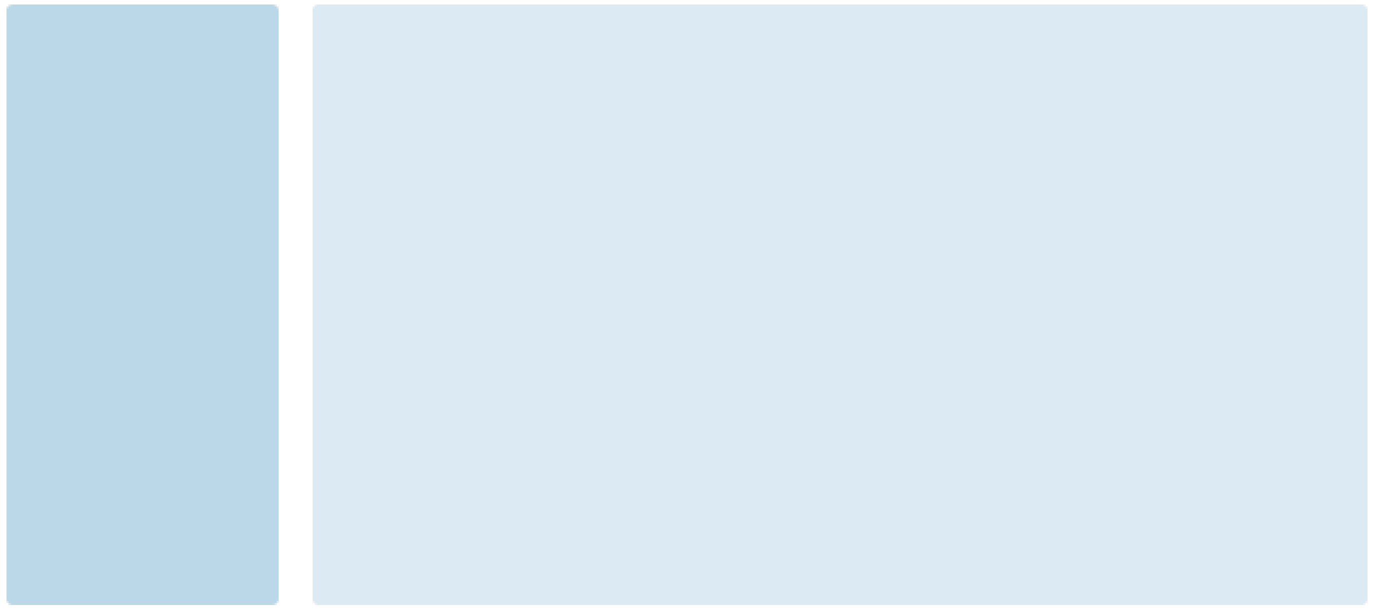
From <https://shiny.rstudio.com/articles/layout-guide.html>



Official bootstrap grid system
https://getbootstrap.com/docs/3.4/css/

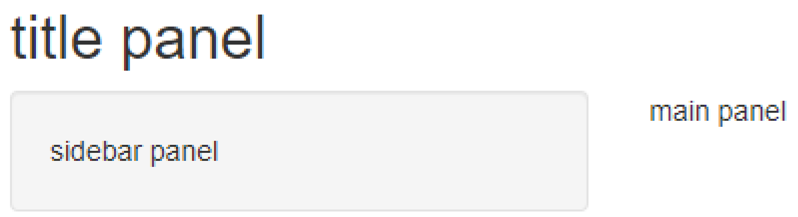
R-Shiny *Grid system for UI*

```
ui <- fluidPage(
  fluidRow(
    column(2,
      "sidebar"
    ),
    column(10,
      "main"
    )
  )
)
```



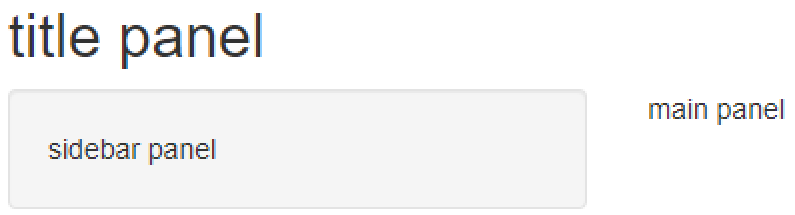
Official bootstrap grid system
https://getbootstrap.com/docs/3.4/css/

R-Shiny *Exercise 0.3 – A simple UI*



R-Shiny *Exercise 0.3 – A simple UI*

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

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

server <- function(input, output){}

shinyApp(ui, server)
```


title panel

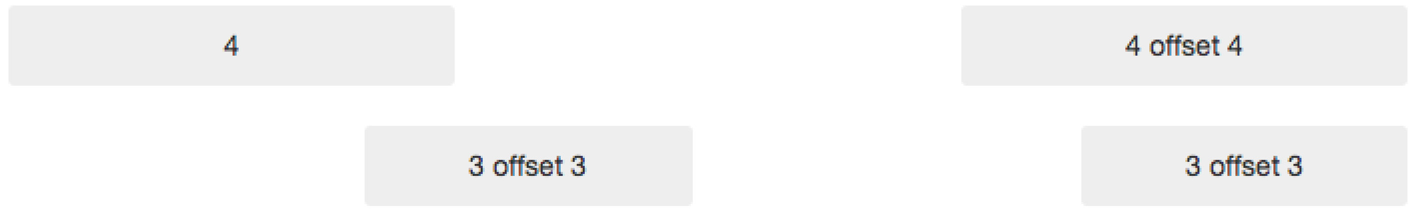
sidebar panel

main panel

R-Shiny *Column Offsetting*

Move columns to the right by adding the `offset` parameter to the `column()` function. Each unit of offset increases the left-margin of a column by a whole column. Consider this layout:

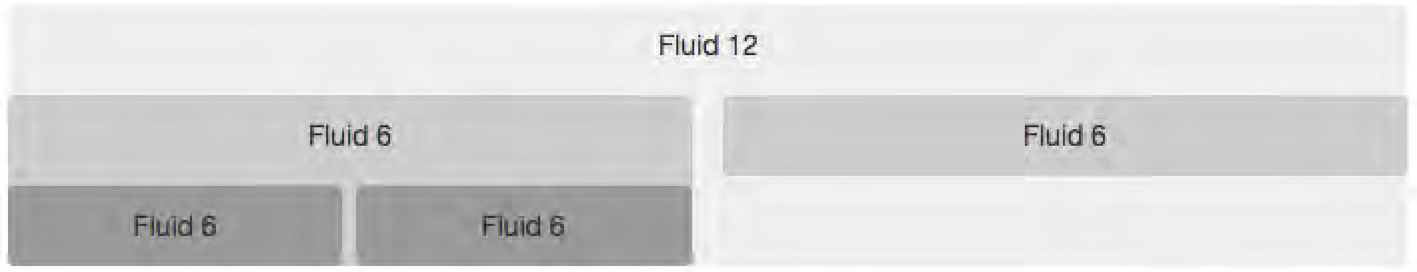
```
ui <- fluidPage(
  fluidRow(
    column(4,
      "4"
    ),
    column(4, offset = 4,
      "4 offset 4"
    )
  ),
  fluidRow(
    column(3, offset = 3,
      "3 offset 3"
    ),
    column(3, offset = 3,
      "3 offset 3"
    )
  )
)
```



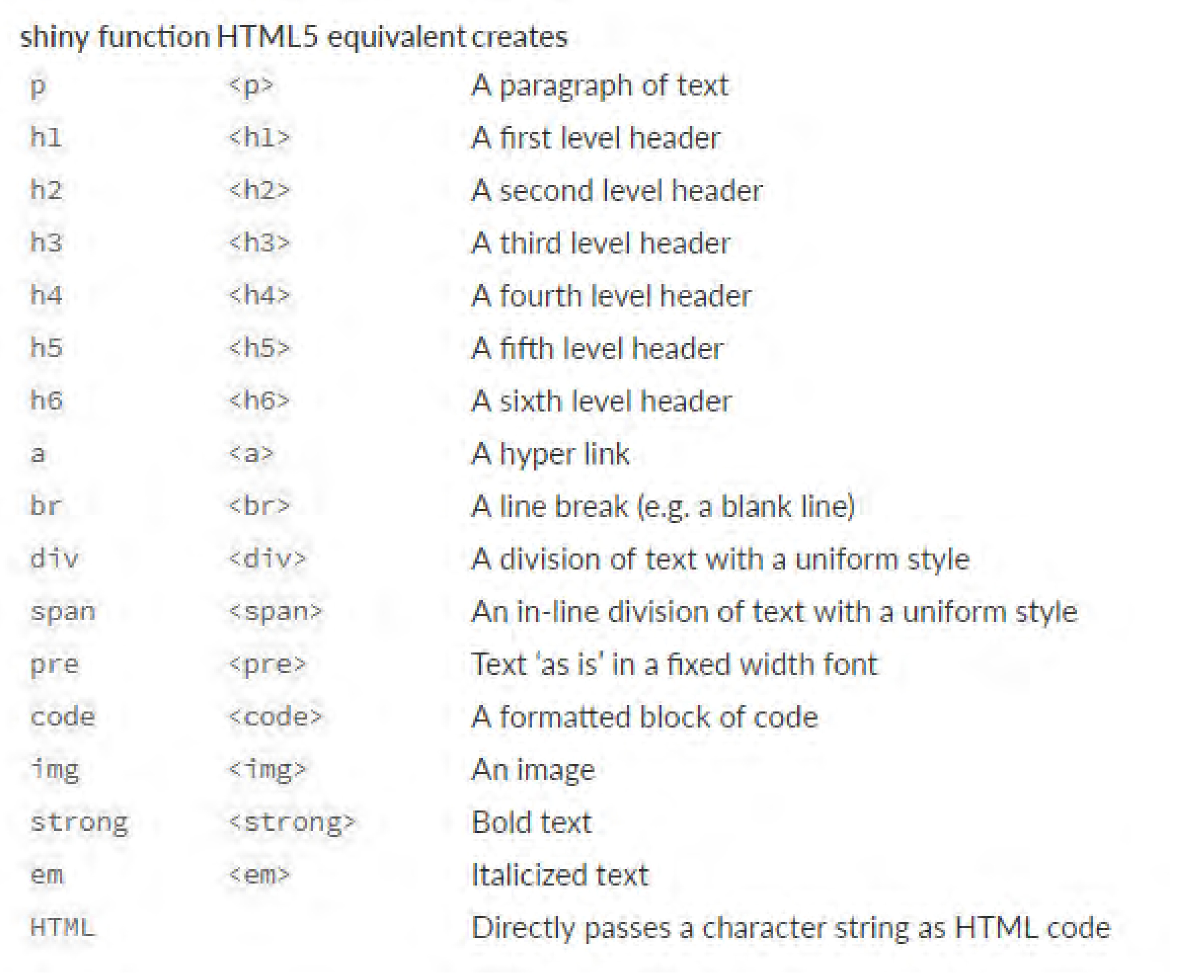
R-Shiny *Column Nesting*

When you nest columns within a fluid grid, each nested level of columns should add up to 12 columns. This is because the fluid grid uses percentages, not pixels, for setting widths. Consider this page layout:

```
ui <- fluidPage(
  fluidRow(
    column(12,
      "Fluid 12",
      fluidRow(
        column(6,
          "Fluid 6",
          fluidRow(
            column(6,
              "Fluid 6"),
            column(6,
              "Fluid 6")
          )
        ),
        column(width = 6,
          "Fluid 6")
      )
    )
  )
)
```



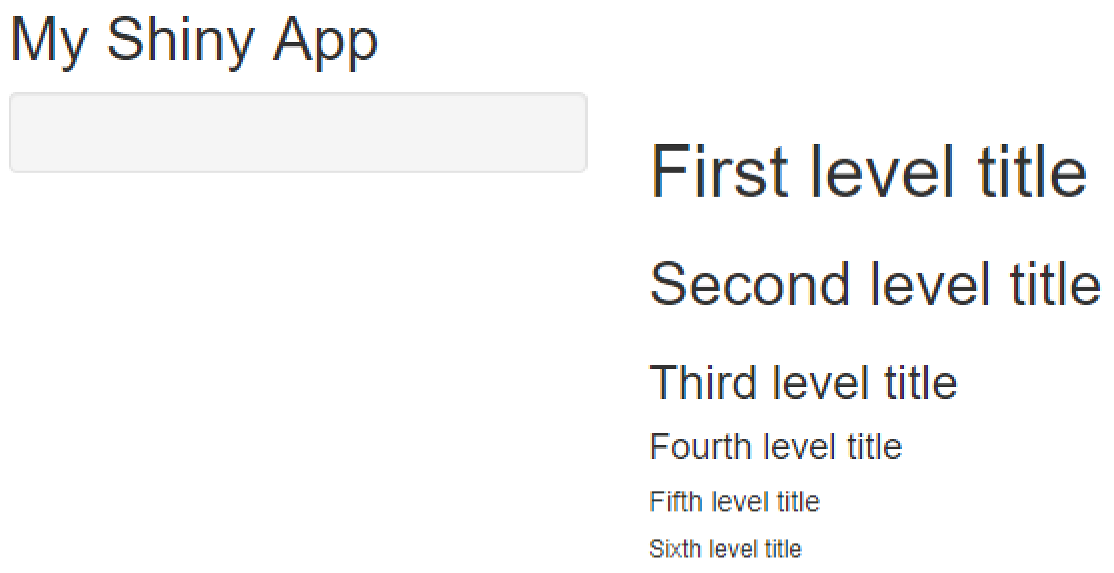
R-Shiny *A little of HTML*



shiny function	HTML5 equivalent	creates
`p`	`<p>`	A paragraph of text
`h1`	`<h1>`	A first level header
`h2`	`<h2>`	A second level header
`h3`	`<h3>`	A third level header
`h4`	`<h4>`	A fourth level header
`h5`	`<h5>`	A fifth level header
`h6`	`<h6>`	A sixth level header
`a`	`<a>`	A hyper link
`br`	`<br>`	A line break (e.g. a blank line)
`div`	`<div>`	A division of text with a uniform style
`span`	`<span>`	An in-line division of text with a uniform style
`pre`	`<pre>`	Text 'as is' in a fixed width font
`code`	`<code>`	A formatted block of code
`img`	`<img>`	An image
`strong`	`<strong>`	Bold text
`em`	`<em>`	Italicized text
`HTML`		Directly passes a character string as HTML code

R-Shiny *Headers*

```r
ui <- fluidPage(
  titlePanel("My Shiny App"),
  sidebarLayout(
    sidebarPanel(),
    mainPanel(
      h1("First level title"),
      h2("Second level title"),
      h3("Third level title"),
      h4("Fourth level title"),
      h5("Fifth level title"),
      h6("Sixth level title")
    )
  )
)

server <- function(input, output){}

shinyApp(ui, server)
```



My Shiny App

First level title

Second level title

Third level title

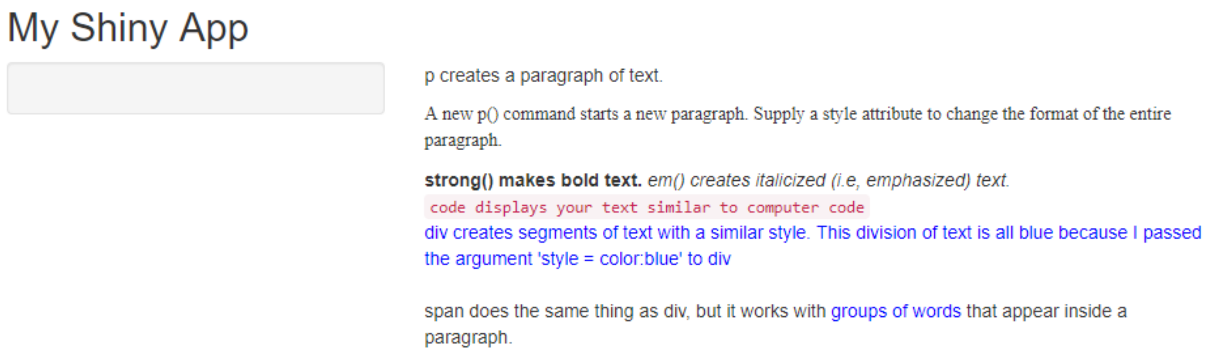
Fourth level title

Fifth level title

Sixth level title

R-Shiny *Text*

```r
ui <- fluidPage(
  titlePanel("My Shiny App"),
  sidebarLayout(
    sidebarPanel(),
    mainPanel(
      p("p creates a paragraph of text."),
      p("A new p() command starts a new paragraph. Supply a style attribute to change the format of the entire 
paragraph.", style = "font-family: 'times'; font-si16pt"),
      strong("strong() makes bold text."),
      em("em() creates italicized (i.e, emphasized) text."),
      br(),
      code("code displays your text similar to computer code"),
      div("div creates segments of text with a similar style. This division of text is all blue because I passed the 
argument 'style = color:blue' to div", style = "color:blue"),
      br(),
      p("span does the same thing as div, but it works with",
        span("groups of words", style = "color:blue"),
        "that appear inside a paragraph.")
    )
  )
)

server <- function(input, output){}

shinyApp(ui, server)
```



My Shiny App

p creates a paragraph of text.

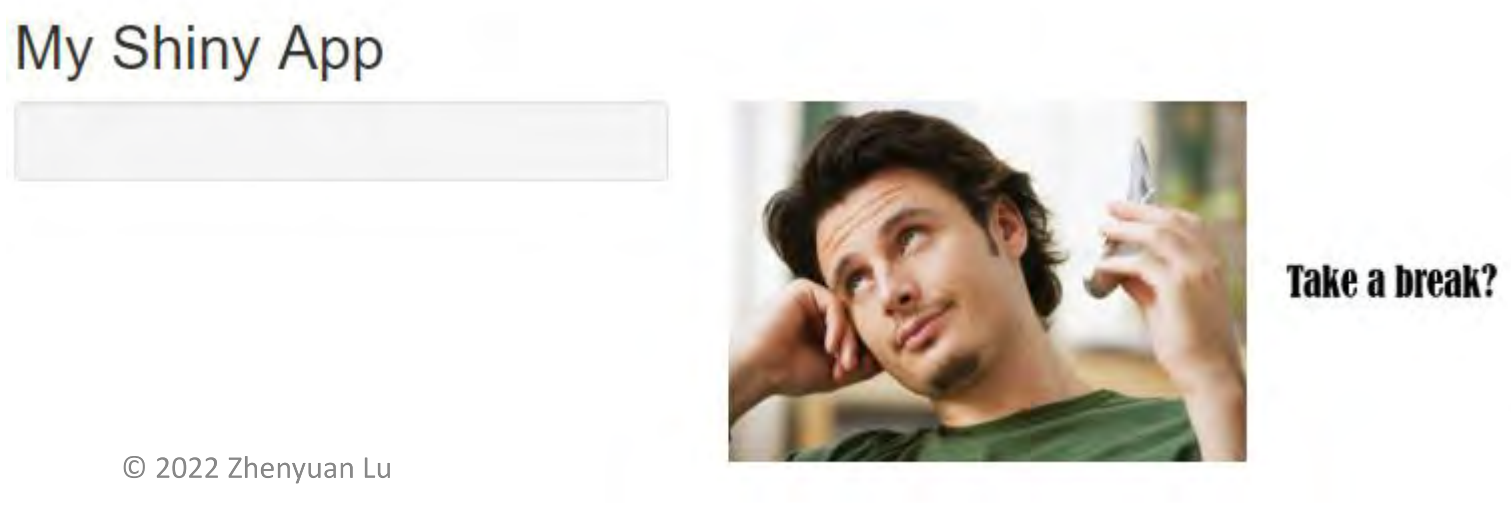
A new p() command starts a new paragraph. Supply a style attribute to change the format of the entire paragraph.

strong() makes bold text. *em() creates italicized (i.e, emphasized) text.*
`code displays your text similar to computer code`
div creates segments of text with a similar style. This division of text is all blue because I passed the argument 'style = color:blue' to div

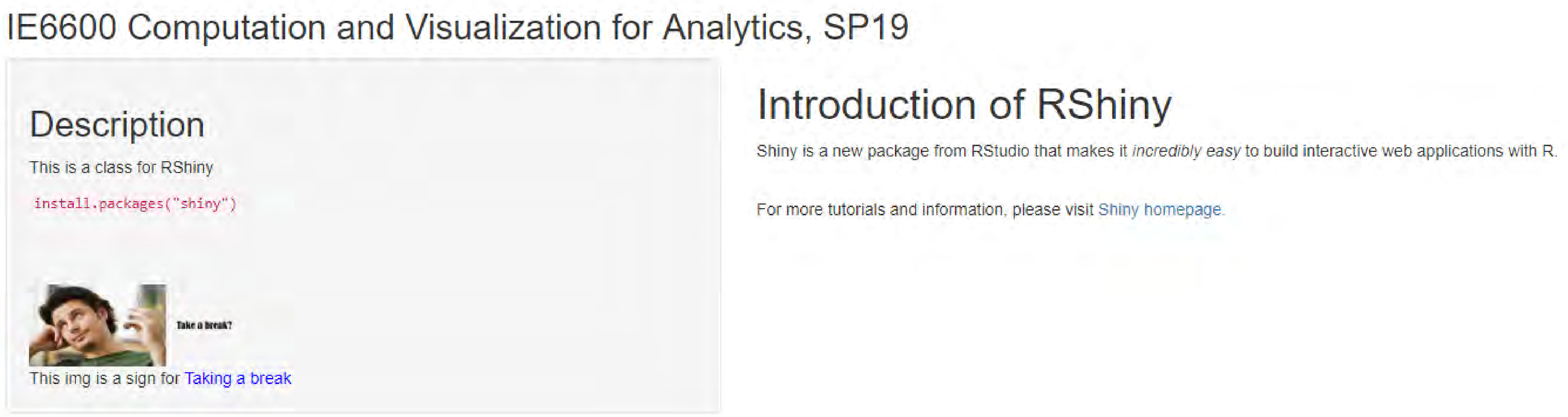
span does the same thing as div, but it works with groups of words that appear inside a paragraph.

R-Shiny *Image*

```
ui <- fluidPage(
  titlePanel("My Shiny App"),
  sidebarLayout(
    sidebarPanel(),
    mainPanel(
      img(src = "takeABreak.png", height = 180, width = 400)
    )
  )
)

server <- function(input, output){}

shinyApp(ui, server)
```



R-Shiny *Exercise*



IE6600 Computation and Visualization for Analytics, SP19

Description

This is a class for RShiny

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

This img is a sign for Taking a break

Introduction of RShiny

Shiny is a new package from RStudio that makes it *incredibly easy* to build interactive web applications with R.

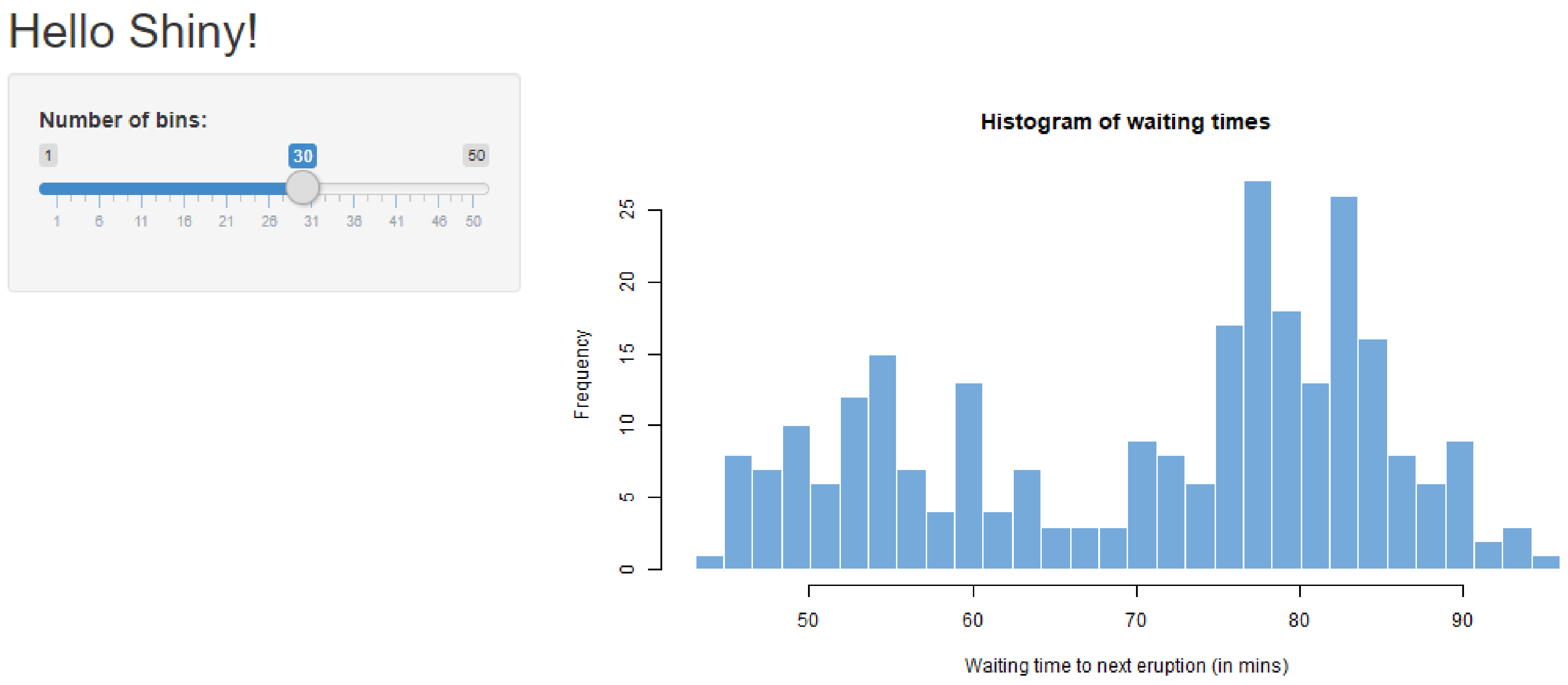
For more tutorials and information, please visit Shiny homepage.

R-Shiny *Answer*

```r
library(shiny)

# Define UI ----
ui <- fluidPage(
  titlePanel("IE6600 Computation and Visualization for Analytics, SP19"),
  sidebarLayout(
    sidebarPanel(
      h2("Description"),
      p("This is a class for RShiny"),
      code('install.packages("shiny")'),
      br(),
      br(),
      br(),
      br(),
      img(src = "takeABreak.png", height = 70, width = 180),
      br(),
      "This img is a sign for",
      span("Taking a break", style = "color:blue")
    ),
    mainPanel(
      h1("Introduction of RShiny"),
      p("Shiny is a new package from RStudio that makes it ",
        em("incredibly easy "),
        "to build interactive web applications with R."),
      br(),
      p("For more tutorials and information, please visit",
        a("Shiny homepage.",
          href = "http://shiny.rstudio.com"))
)
  )
)
```

```r
# Define server logic ----
server <- function(input, output) {

}

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

R-Shiny *Page structure*



R-Shiny *Server for “Hello Shiny”*

UI

```r
library(shiny)

# Define UI for app that draws a histogram ----
ui <- fluidPage(

  # App title ----
  titlePanel("Hello Shiny!"),

  # Sidebar layout with input and output definitions ----
  sidebarLayout(

    # Sidebar panel for inputs ----
    sidebarPanel(

      # Input: Slider for the number of bins ----
      sliderInput(inputId = "bins",
                  label = "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)

    ),

    # Main panel for displaying outputs ----
    mainPanel(

      # Output: Histogram ----
      plotOutput(outputId = "distPlot")

    )
  )
)
```

Server

```r
# Define server logic required to draw a histogram ----
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")

    })

}
```

Run App

```r
shinyApp(ui, server)
```

Resources

Resource

R-Shiny: basic tutorial and examples
https://shiny.rstudio.com/gallery/