

# shiny 인터랙티브 웹 만들기

[https://mrchypark.github.io/dabrp\\_classnote2/class8](https://mrchypark.github.io/dabrp_classnote2/class8)

박찬엽

2017년 8월 9일

## 목차

1. 과제 확인
2. shiny
  1. 웹 어플리케이션 기초
  2. shiny의 입출력
  3. 반응형을 위한 도구
3. htmlwidget
4. 과제

# 과제

질문

# 참고하면 좋은 자료

- 공식 페이지의 갤러리
- 공식 페이지의 튜토리얼
- 공식 페이지의 정보글
- 통계 분석 너머 R의 무궁무진한 활용(도서)

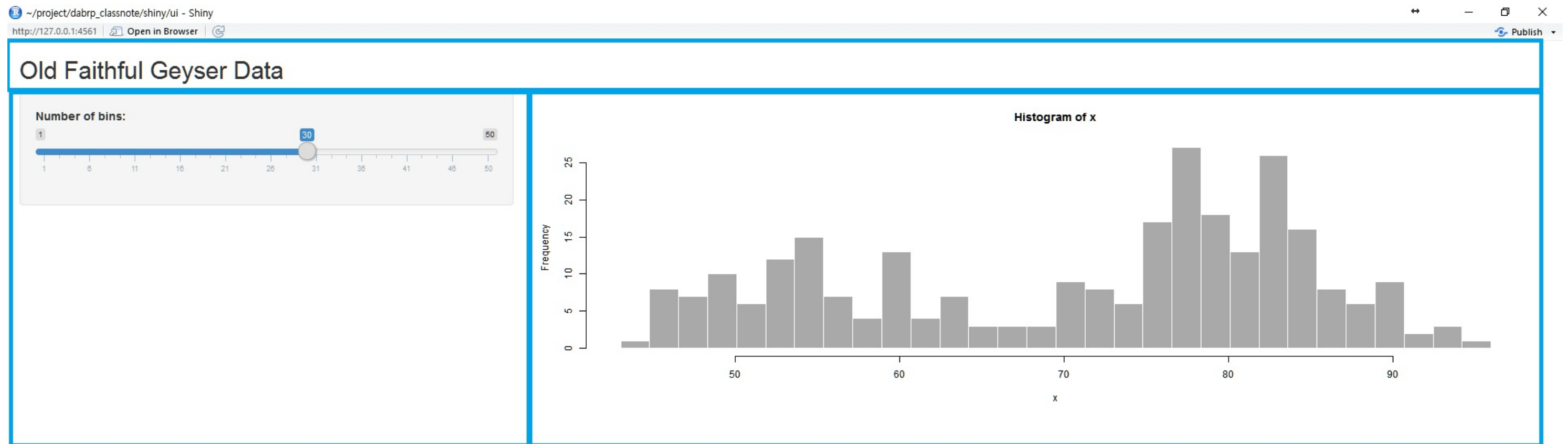
# shiny

**shiny**는 R로 웹 어플리케이션을 만들 수 있게 해주는 프레임워크입니다. shiny는 ui라고 불리는 화면, server라고 불리는 데이터 처리 및 관리, 그 안에 입출력과 render와 배포로 구성되어 있습니다.



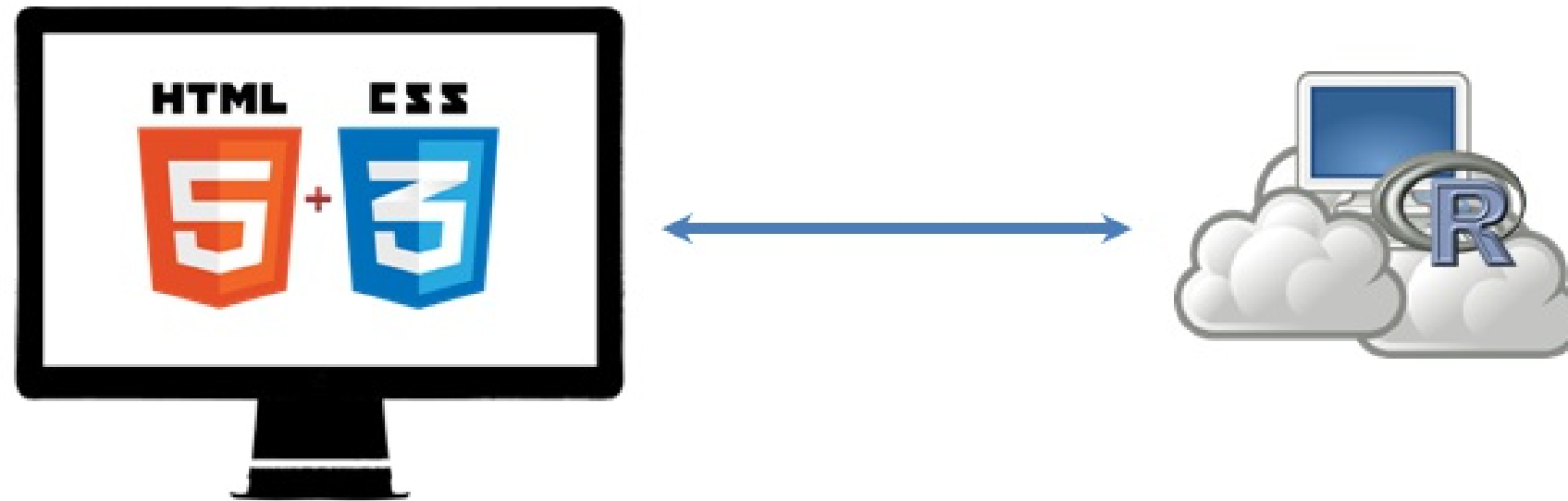
# shiny의 화면

ui라고 말하는 화면은 실제로 사용자가 보는 화면을 뜻합니다. shiny에서는 크게 `titlePanel`과 `sidebarPanel`, `mainPanel`의 세 가지로 구성되어 있습니다. 이렇게 지정함으로써 html에 대해 잘 몰라도 사용할 수 있게 shiny가 구성되어 있습니다.



# shiny의 서버

shiny에서의 server는 실제 서버가 브라우저와 통신하는 과정 전체를 간단하게 만들어주는 역할을 합니다. 화면에서 사용자가 동작하는 것에 대해서 받을 수 있는 input을 서버에서 데이터나 그림 조작에 사용을 하고 웹 기술이 이해할 수 있게 render하여 output으로 화면에 다시 보내주는 형태로 shiny가 동작합니다.



# shiny의 입출력

shiny는 다른 R 패키지와는 다르게 강제하는 변수가 3개 있습니다. 그것은 `input`, `output`, `session` 입니다. 각각 객체로써 존재하고 이번에는 `input`과 `output`으로 데이터를 `ui`와 `server`가 교환하는 방법을 소개하겠습니다. 각각의 객체는 `*Input` 함수와 `*Output` 함수로 데이터를 입력 받아 저장합니다. `*Output` 함수는 `render*` 함수로 선언됩니다.



# shiny의 입출력

shiny는 다른 R 패키지와는 다르게 강제하는 변수가 3개 있습니다. 그것은 `input`, `output`, `session` 입니다. 각각 객체로써 존재하고 이번에는 `input`과 `output`으로 데이터를 `ui`와 `server`가 교환하는 방법을 소개하겠습니다. 각각의 객체는 `*Input` 함수와 `*Output` 함수로 데이터를 입력 받아 저장합니다. `*Output` 함수는 `render*` 함수로 선언됩니다.

# shiny의 배포

shiny는 `shiny-server`를 통해서 동작합니다. `shiny-server`는 shiny app이 동작할 수 있는 서버를 의미합니다. 오픈소스와 기업용 솔루션이 모두 준비되어 있으며 실습에는 <http://www.shinyapps.io/>를 이용해 보겠습니다.

# shiny의 입출력

shiny는 웹 기술을 R로 사용할 수 있게 만드는 덕분에 render라는 과정이 필요합니다. 그래서 render 환경에서 변수들이 관리되어야 합니다. 입출력은 input 변수와 output 변수로 관리합니다. 이 두 가지는 render 밖에서 관리되는 변수로 다르게 취급해야 합니다.

- input: 웹 페이지의 입력을 통해서 들어오는 데이터를 사용하기 위한 변수
- output: 웹 페이지에 R 연산 결과물을 전달하기 위해 사용하는 변수

# output\$

output 변수는 list라고 이해하시면 좋을 것 같습니다. list인 output 변수는 output\$뒤에 변수명을 작성함으로써 output 객체에 필요한 결과물을 전달합니다. 만약에 화면에 보여줘야할 것의 이름을 sample이라고 하면 output\$sample에 필요한 내용을 선언하는 것으로 진행합니다.

```
...  
output$sample <- renderPlot({ ...plot()... })  
...
```

위 코드는 server쪽 코드에서 작성합니다. 위 예시는 plot함수로 만들어지는 이미지를 output\$sample에 저장하는 것을 뜻합니다. 그럼 ui쪽에서 이걸 어디에다 위치하게 하는지를 결정하는 함수에서 사용할 수 있습니다. 이때는 plotOutput 함수를 사용합니다.

# plotOutput

```
...  
plotOutput("sample")  
...
```

함수명이 output 이고, "로 변수명을 감싸며, output\$ 문법을 사용하지 않고 컬럼명인 sample만 사용한다는 점을 주목해 주세요. server쪽 코드에서 output\$에 컬럼명의 형태로 저장한 R의 결과물을 ui쪽 코드에서 \*Output 함수에서 "로 감싼 글자의 형태로 컬럼명만 작성해서 사용합니다. 그럼 이제 위해서 output\$sample에 선언할 때 사용한 render\*({})에 대해서 알아보겠습니다.

# render\*({})

Rmd 때 render라는 과정을 거쳐서 Rmd를 md로, md를 html로 바꾸는 것을 알아봤었습니다. shiny에서도 같은 과정이 필요합니다. 그래서 render\*({ }) 함수가 필요합니다. 예시로 보겠습니다.

```
...  
output$sample <- renderPlot({  
  plot(faithful)  
})  
...
```

`render*({ })` 함수 안에는 `plot` 함수가 R 문법으로 작성되어 있습니다. 그리고 `render*({ })`의 특이한 점은 `()` 안에 `{}`가 또 있다는 점입니다. 여기서 `({ })` 안쪽은 R의 세계이고, 그 바깥은 웹의 세계라고 이해하겠습니다. 그래서 이렇게 가능합니다.

```
...  
output$sample <- renderPlot({  
  data <- faithful[faithful$eruptions >3, ]  
  plot(data)  
})  
...
```

R의 세계 내에서 처리하는 것은 계속 사용할 수 있어서 `data` 변수에 선언한 내용을 `plot` 함수가 사용할 수 있습니다. 여기서 `input$`이 활용될 때 반응형으로 작성할 수 있는 것입니다.

# input\$

input\$은 output\$과 같이 웹의 세계에 있는 변수입니다. 그래서 \*Input() 함수들을 통해서 input\$의 컬럼 이름으로 웹 페이지 내에서 얻을 수 있는 데이터를 R의 세계에서 사용할 수 있게 해줍니다. 우선 input\$에 웹의 세계의 데이터를 R의 세계로 가져와 보겠습니다.

```
sliderInput("sdata1", "슬라이더 입력:", min=50, max=150, value=100)
```

위의 코드는 sliderInput()이라는 \*Input() 패밀리 함수를 통해서 input\$sdata1이라는 곳에 웹 데이터를 저장하는 것을 뜻합니다. \*Input() 패밀리 함수는 같은 규칙을 가지는데 입력 형태명 Input()의 함수명을 가지고, 첫번째 인자는 input\$ 뒤로 붙을 컬럼명, 두번째 인자는 화면에 보여줄 글자를 의미합니다. 나머지 인자는 입력형태에 따라 다양하게 달라집니다. shiny에서 지원하는 입력형태는 이곳을 확인해 주세요.

# shiny 앱 작성 구조

이제 input하나를 받고 output 하나를 출력하는 shiny 앱을 작성할 준비가 되었습니다. 이제 간단한 sample을 통해 작성된 코드를 확인해 보고 같이 작성을 진행해 보겠습니다. shiny는 그 복잡함 때문에 여러 사례를 풍부하게 제공하는 몇 안되는 패키지입니다. 사례 참고만 잘 하셔도 멋진 반응형 웹을 만들 수 있습니다.

```
library(shiny)
runExample()
```

```
## valid examples are "01_hello", "02_text", "03_reactivity", "04_mpg", "05_sliders", "06_tabsets", "07_
```

위 코드는 실행해 볼 수 있는 예제를 보여줍니다. 아래처럼 실행하시면 바로 실행되는 shiny 앱 예제를 확인하실 수 있습니다. shiny github에는 예제 repo가 있고, 활용할 수 있는 예제는 2017-08-09 기준 110개입니다.

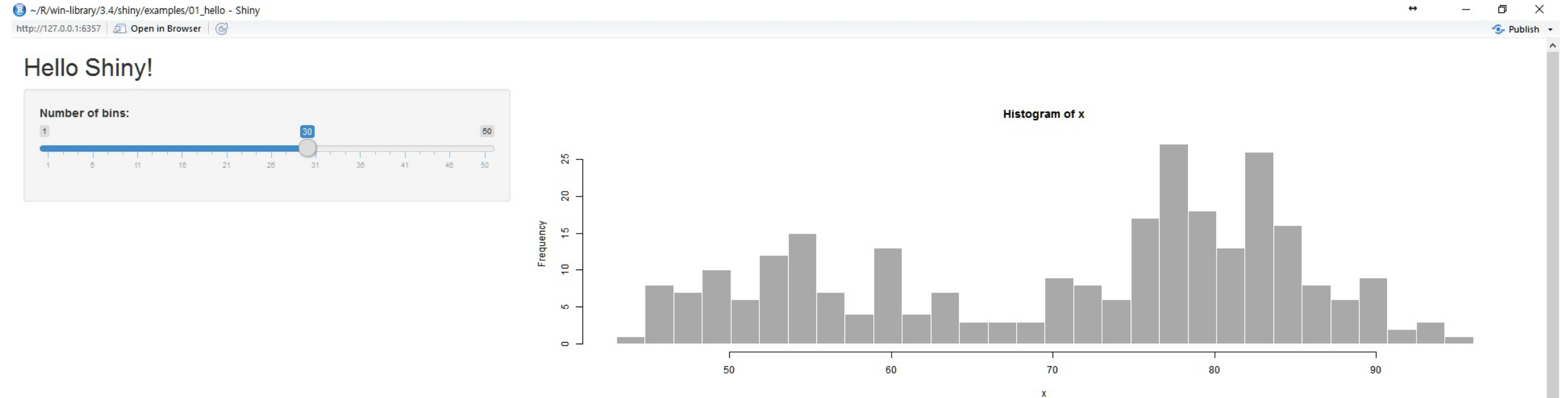


## \* 수집코드 예시

```
library(rvest)
url<-"https://github.com/rstudio/shiny-examples"
tem<- read_html(url)
fold<-tem %>% html_nodes("tr td span a.js-navigation-open") %>% html_text
cnt <- grep("^[0-9]",fold) %>% length
```

```
runExample("01_hello")
```

위를 실행해 보시면 아래와 같은 화면이 나옵니다. 예제는 코드도 함께 볼 수 있게 되어 있어서 공부하기 좋습니다.



Hello Shiny!  
by RStudio, Inc.

This small Shiny application demonstrates Shiny's automatic UI updates. Move the *Number of bins* slider and notice how the `renderPlot` expression is automatically re-evaluated when its dependant, `input$bins`, changes, causing a histogram with a new number of bins to be rendered.

server.R

ui.R

↑ show with app

```
library(shiny)

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

  # Expression that generates a histogram. The expression is
  # wrapped in a call to renderPlot to indicate that:
  #
  # 1) It is "reactive" and therefore should be automatically
  #    re-executed when inputs change
  # 2) Its output type is a plot

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

    # draw the histogram with the specified number of bins
    hist(x, breaks = bins, col = 'darkgray', border = 'white')
  })
}
```

# 실습 1

1. `runExample("01_hello")` 를 실행합니다.
  1. New File > Shiny Web App ... > Create 로 샤이니 파일을 만듭니다.
  2. `sliderInput` 을 Numeric input 형태로 바꿔주세요.
  3. `bins`의 최소값과 최대값을 입력하는 Slider range input을 추가해주세요.
    - server 코드 내에 `hist` 전 `bins`를 선언하는 곳에 `min`, `max`가 있습니다.

우선 서버쪽 부터 보겠습니다.

```
# shiny 패키지를 불러옵니다.
library(shiny)

# server를 function으로 선언합니다. 웹의 세계에서 사용할 변수는 input과 output 입니다.
function(input, output) {

  # 웹의 세계의 변수인 output에 list 컬럼인 distPlot을 선언합니다.
  # 그 선언을 renderPlot({})으로 해서 distPlot이 plot()으로 작성한 그림이라는 것을 이해합니다.
  # renderPlot({ ... }) 함수 안에 R의 세계에서 hist() 함수의 결과가 나오게 작성합니다.
  # 이때 input$bin으로 웹에서 사용자가 주는 데이터를 활용할 것입니다.
  # bin은 경계라는 뜻으로 연속형 변수를 histogram을 그릴 때 얼마나 구간을 나눌 것인지를 결정합니다.
  # ?hist를 통해 어떤 그림을 그리는 함수인지를 확인해 보세요.
  # ?seq를 통해 어떤 결과물을 만드는 함수인지 확인해 보세요.

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

    # draw the histogram with the specified number of bins
    hist(x, breaks = bins, col = 'darkgray', border = 'white')
  })
}
```

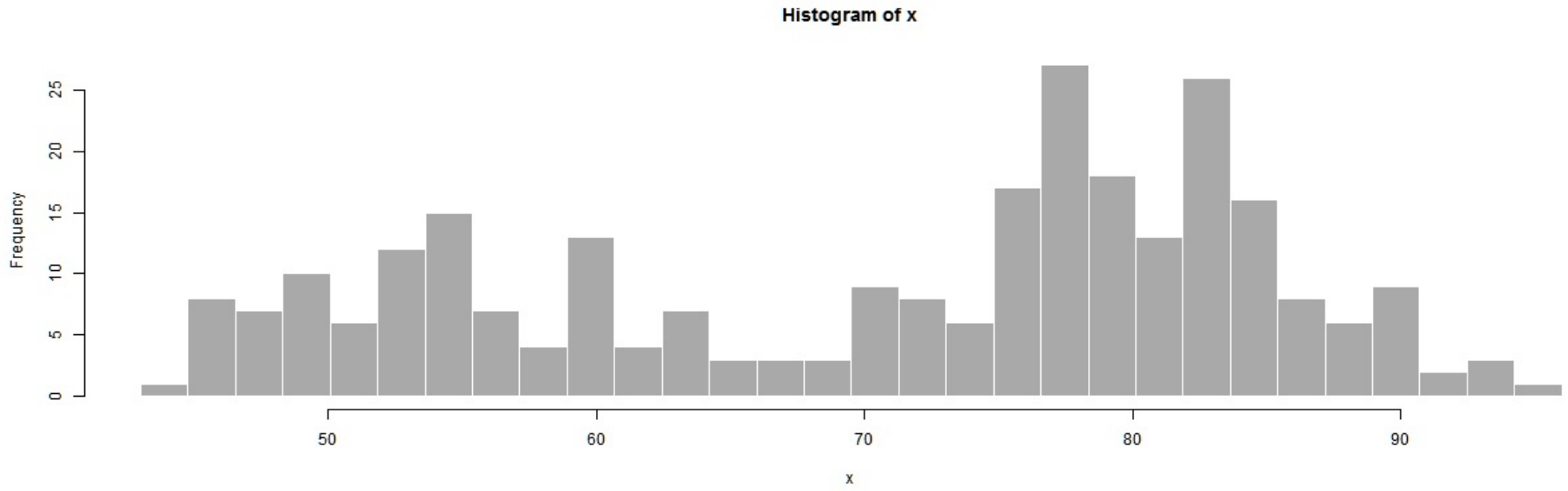
위에 코드에서 `input$bins`가 `render*({})` 함수 안에서 사용되었습니다. `output$`은 웹의 세계에서 사용하지만 `input$`은 R의 세계에서 사용합니다. 이부분은 `reactive({})` 함수에서 추가적으로 다루기로 하겠습니다. 이제 ui 쪽 코드를 확인해 보겠습니다.

~/R/win-library/3.4/shiny/examples/01\_hello - Shiny  
http://127.0.0.1:6357 | Open in Browser | Publish

## Hello Shiny!

Number of bins:

1 30 50



Frequency

x

### Hello Shiny!

by RStudio, Inc.

This small Shiny application demonstrates Shiny's automatic UI updates. Move the *Number of bins* slider and notice how the `renderPlot` expression is automatically re-evaluated when its dependant, `input$bins`, changes, causing a histogram with a new number of bins to be rendered.

server.R ui.R

show with app

```
library(shiny)

# Define UI for application that draws a histogram
fluidPage(

  # Application title
  titlePanel("Hello Shiny!"),

  # Sidebar with a slider input for the 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")
    )
  )
)
```

```

# shiny 패키지를 불러옵니다.
library(shiny)

# fluidPage는 shiny의 페이지를 보여주는 핵심 기능입니다.
# titlePanel, sidebarPanel, mainPanel로 구성되어 있습니다.
# titlePanel은 제목을 작성하는 곳입니다.
# sidebarPanel은 전체 화면을 3등분해서 왼쪽을 뜻합니다.
# mainPanel은 전체 화면을 3등분해서 오른쪽을 뜻합니다.
fluidPage(
  # Application title
  titlePanel("Hello shiny!"),
  # Sidebar with a slider input for the 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")
    )
  )
)

```

# input 및 output 설명

이제 코드를 다시 보면 `sliderInput`을 통해서 `input$bins` 데이터를 웹의 세상에서 받아오는 것을 볼 수 있습니다. 지금 기본 값으로 30을 기록해 두었지만, 웹의 세상에서 변화시켜주는 것에 따라 `input$bins`의 데이터가 변해서 `server`에서 `input$bins`의 형태로 그 데이터를 사용할 수 있습니다.

웹의 세상이나 R의 세상이란 말은 정확한 표현은 아니지만 이해를 돕기 위해서 활용했습니다.

그리고 `ui`에서는 웹의 세계로 나온 `output$distPlot`을 `plotOutput()` 함수로 웹 페이지에 위치시킬 수 있습니다. `output$distPlot`은 현재 `mainPanel`에 위치해 있네요. 위치의 세부적인 조정은 [이곳](#)을 참고하세요.

# 샤이니 테마

shiny의 전체 화면 구성 기능은 [이곳](#)를 참고하세요. 그리고 이것은 [bootstrap](#)을 사용하여 만들었습니다. 그래서 [bootstrap](#)의 화면 구성에 대한 개념을 이해하시면 더 쉽게 다양한 세부 조작이 가능합니다. fluid라고 작성하는 모든 부분은 화면의 크기에 반응하여 적절하게 변화한다는 뜻입니다. [bootswatch](#)에서 제공하는 테마를 바로 적용해서 사용할 수 있습니다.

```
fluidPage(theme = "Journal",  
  titlePanel("테마를 바꿀 수 있습니다.")  
)
```



# html tag

shiny는 대부분의 html tag를 지원합니다. html에 대한 사전 지식이 있으시면 활용하시는데 도움이 됩니다. 추가적인 내용은 [이 곳](#)을 참고하세요.

```
names(tags)
```

```
##      [1] "a"          "abbr"       "address"    "area"       "article"
##      [6] "aside"     "audio"      "b"          "base"       "bdi"
##     [11] "bdo"       "blockquote" "body"       "br"         "button"
##     [16] "canvas"    "caption"    "cite"       "code"       "col"
##     [21] "colgroup"  "command"    "data"       "datalist"   "dd"
##     [26] "del"       "details"    "dfn"        "div"        "dl"
##     [31] "dt"        "em"         "embed"      "eventsource" "fieldset"
##     [36] "figcaption" "figure"     "footer"     "form"       "h1"
##     [41] "h2"        "h3"         "h4"         "h5"         "h6"
##     [46] "head"      "header"     "hgroup"     "hr"         "html"
##     [51] "i"         "iframe"     "img"        "input"      "ins"
##     [56] "kbd"       "keygen"     "label"      "legend"     "li"
##     [61] "link"      "mark"       "map"        "menu"       "meta"
##     [66] "meter"     "nav"        "noscript"   "object"     "ol"
##     [71] "optgroup"  "option"     "output"     "p"          "param"
##     [76] "pre"       "progress"   "q"          "ruby"       "rp"
##     [81] "rt"        "s"          "samp"       "script"     "section"
##     [86] "select"    "small"      "source"     "span"       "strong"
##     [91] "style"     "sub"        "summary"    "sup"        "table"
##     [96] "tbody"     "td"         "textarea"   "tfoot"      "th"
##    [101] "thead"     "time"       "title"      "tr"         "track"
##    [106] "u"         "ul"         "var"        "video"      "wbr"
```

# 반응형 함수들

위에서 shiny 코드들을 소개할 때 웹의 세상과 R의 세상으로 설명한 부분이 기억나실 겁니다. 이것은 웹의 세상의 데이터를 R의 세상으로 데려와서 여러 처리에 사용하고, 결과를 다시 웹의 세상으로 보내는 과정을 설명하면서 사용했습니다. 이 때 `input$`은 R의 세상에 있었고, `output$`은 웹의 세상에 있었습니다. `render*({})` 함수를 통해서 R의 세상의 결과물을 `output$`에 전달한다고도 했구요. 이 과정에서 `render*({})` 함수는 목적이 결과물을 전달하는데 있습니다. `input$`을 이용해서 계산하는 과정이 간단한 것들은 `render*({})` 함수 내부에 포함되어도 큰 문제가 되지 않지만, 계산을 여러 `output$`에서 사용한다면 고치는 것을 고려해 볼 수 있습니다.

개발에서는 반복되는 부분을 최대한 줄이거나 합치려는 경향이 있는데 2가지 장점이 있습니다. 하나는 효율화가 가능하구요, 다른 하나는 가독성이 좋아집니다.

우선 겹쳐서 사용하는 사례를 보겠습니다.

```
library(shiny)

ui<-fluidPage(
  titlePanel("test"),
  sidebarPanel(
    selectInput("dataSelect", "데이터셋 선택 :", choices=list("mtcars", "sleep", "iris", "co2"))
  ),
  mainPanel(
    verbatimTextOutput("out1"),
    verbatimTextOutput("out2")
  )
)
server<-function(input, output){
  output$out1 <- renderPrint({
    summary(get(input$dataSelect))
  })
  output$out2 <- renderPrint({
    str(get(input$dataSelect))
  })
}

shinyApp(ui, server)
```

?get으로 어떤 동작을 하는 함수인지 확인하세요. renderPrint({})가 두 번 있으면서 get함수가 각각 실행되어 같은 연산을 두 번하는 효과가 발생하였습니다. 그렇다면 그 동작을 한번만 하게끔 바꿔보겠습니다.

```
library(shiny)

ui<-fluidPage(
  titlePanel("test"),
  sidebarPanel(
    selectInput("dataSelect", "데이터셋 선택 :", choices=list("mtcars", "sleep", "iris", "co2"))
  ),
  mainPanel(
    verbatimTextOutput("out1"),
    verbatimTextOutput("out2")
  )
)
server<-function(input, output){

  selectedData<-get(input$dataSelect)

  output$out1 <- renderPrint({
    summary(selectedData)
  })
  output$out2 <- renderPrint({
    str(selectedData)
  })

}

shinyApp(ui, server)
```

# reactive context

get함수로 데이터를 부르는 부분을 앞으로 빼내고, 선언한 `selectedData`를 `summary`와 `str`함수로 사용하였습니다. 실행해 보시면 에러가 나는 것을 확인할 수 있습니다. 웹의 세상에서 R의 세상 문법으로 작성해서 생기는 문제인데요. 에러코드를 함께 보겠습니다.

```
Error in .getReactiveEnvironment()$currentContext() :  
  Operation not allowed without an active reactive context.  
  (You tried to do something that can only be done from inside  
  a reactive expression or observer.)
```

reactive context라는게 눈에 띄는데요. 이것이 R의 세상 문법이 동작하는 공간입니다.

```
selectedData<-get(input$dataSelect)
```

위에 코드는 R 문법으로 이루어져 있는데, 이것을 웹의 세상에서 작성했기 때문에 생기는 것이죠. 그럼 R의 세상 문법으로 작성할 수 있게 `render*({})`와 같은 환경을 제공하는 함수를 사용해 보겠습니다.

```
library(shiny)

ui<-fluidPage(
  titlePanel("test"),
  sidebarPanel(
    selectInput("dataSelect", "데이터셋 선택 :", choices=list("mtcars", "sleep", "iris", "co2"))
  ),
  mainPanel(
    verbatimTextOutput("out1"),
    verbatimTextOutput("out2")
  )
)
server<-function(input, output){

  selectedData<-reactive({
    get(input$dataSelect)
  })

  output$out1 <- renderPrint({
    summary(selectedData())
  })
  output$out2 <- renderPrint({
    str(selectedData())
  })

}

shinyApp(ui, server)
```

# 웹의 세상과 R의 세상

`reactive({})`는 그 내부를 R의 세상으로 만듦으로써 선언 받는 변수를 함수화하여 반응형 데이터의 특성을 유지한채로 R의 세상에서 사용할 수 있게 해줍니다. 함수화라는 부분을 잘 보셔야 하는데, `selectedData`가 변수가 아니라 뒤에 `()`가 붙은 함수라는 사실을 확인하세요. 웹의 세상에 있는 `selectedData`는 변수를 선언한 것이 아니라 R의 세상에서 사용할 수 있게 함수화하여 함수로써 선언하였습니다. 이런 R의 세상과 웹의 세상을 넘나드는 부분을 관장하는 함수에는 `render*({})`, `reactive({})`, `isolate({})`, `observeEvent({})`, `eventReactive({})`, `reactiveValues({})`, `observe({})`가 있습니다. 앞에 두 개는 예시를 들어 설명을 했구요.

## 반응형 함수

- `isolate({})`: 즉각적인 반응이 아니라 어떤 다른 사용자의 입력(ex> action 버튼)을 인지하여 그 때 `input`에 해당하는 부분을 확인하여 결과물을 만듭니다. `render*({})` 함수 내부에서 사용합니다.
- `observeEvent({})`, `eventReactive({})`: 어떤 `input`을 인지하여 동작합니다. `observeEvent({})`는 서버 부분에서 실행해야 할 것들을 담당하기 때문에 선언의 형태가 아니라 독립적으로 작성됩니다. 예를 들어 서버에 파일을 저장한다거나, 로그로 남기는 글자를 출력하거나 할 때 사용합니다.
- `eventReactive({})`: `reactive({})`와 같이 웹의 세상의 변수에 선언하면서 함수화하여 R의 세상에서 반응형 데이터로써 사용할 수 있습니다.
- `reactiveValues({})`: `input$`이나 `output$`과 같은 역할을 하는 변수를 만드는 함수입니다. - `observe({})`: `observeEvent({})`와 `eventReactive({})`의 근간이 되는 함수로, `input$`의 입력을 인지하는 역할을 합니다.



# 실습 2

1. [https://mrchypark.github.io/dabrp\\_classnote2/class6#29](https://mrchypark.github.io/dabrp_classnote2/class6#29)의 4번 그래프 함수를 바탕으로 shiny 앱을 만들겠습니다.
    1. **Checkbox group input** 으로 나라를 선택하도록 만들겠습니다.
    2. 나라는 어떤 나라던 10개만 사용하겠습니다.
- 
- 그래프 코드는 [https://mrchypark.github.io/dabrp\\_classnote2/class6#30](https://mrchypark.github.io/dabrp_classnote2/class6#30) 입니다.
  - 데이터는 [https://mrchypark.github.io/dabrp\\_classnote2/class6#11](https://mrchypark.github.io/dabrp_classnote2/class6#11) 입니다.

# encoding 문제

이곳를 참고하세요. 이런 문제의 대부분은 중국어도 같이 가지고 있어서 중국쪽의 해결책이 많이 도움이 됩니다. 일본어 역시 마찬가지입니다.

shiny가 지원하는 js 패키지들의 활용

# 지도를 지원하는 leaflet

leaflet은 javascript 언어로 이루어진 지도에 대한 패키지입니다. shiny는 [htmlwidget](#)을 이용해서 javascript를 R로 변환하여 사용하는 것이 매우 많습니다. leaflet 역시 마찬가지로 [이곳](#)에 상세한 코드 예시들과 함께 소개가 준비되어 있습니다.

# 마우스 이벤트를 추가적으로 지원하는 ggiraph

**ggiraph**는 ggplot의 geom계 함수들에게 뒤에 `_interactive`를 붙여 활용함으로써 조금더 표가 사용자와 상호작용 할 수 있는 부분을 추가해줍니다. 효과로는 `click`, `hover`, `zoom`, `data_id` 등이 있습니다.

# 상호작용 network 화면을 만드는 visNetwork

**visNetwork**는 네트워크 시각화를 위한 패키지로 상호작용에 대해 유연하게 작성할 수 있게 설계되었습니다. node와 edge, group에 대한 개념만 이해하시면 멋진 네트워크 시각화 화면을 만들어 볼 수 있습니다.

# dashboard를 shiny로 만들어 보자 shinydashboard

**shinydashboard**는 대쉬보드를 shiny로 제작할 수 있게 대쉬보드 제작 프레임워크인 **AdminLTE**을 기반으로 작성했습니다. 유료 템플릿을 판매하는 기반을 가진 오픈소스인 만큼 프레임워크의 질이 product ready 수준이라 사용하기 매우 좋습니다.