

# 더욱 프로답게 만드는 동적 문서화 및 시각화

Step2. 클릭에 실시간으로 반응하는 Shiny Application 학습하기

[https://mrchypark.github.io/dabrp\\_classnote3/class8](https://mrchypark.github.io/dabrp_classnote3/class8)

[[pdf다운로드](#)] [[문의하기](#)] [[피드백하기](#)]

박찬엽

2017년 11월 16일

# 목차

## 1. 공부를 위한 자료

- 문서
- 예시 어플리케이션

## 2. shiny

- 웹 어플리케이션 기초
- 최소 동작 앱과 UI
- 입출력 구조
- 반응성 조절
- shinyapps에 배포

## 3. shiny-ext

- 지도를 그리는 leaflet
- 차트에 마우스 동작을 추가하는 ggriaph
- 네트워크 시각화의 끝판왕 visNetwork
- 종합 대쉬보드 패키지 shinydashboard

# 참고하면 좋은 자료

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

# shiny

shiny는 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.

# 설치

패키지를 설치하는 방법은 다음과 같습니다.

```
## cran에서 설치
install.packages("shiny")

## github에서 설치
if (!requireNamespace("devtools"))
  install.packages("devtools")
devtools::install_packages("rstudio/shiny")
```

# shiny 앱 예시

shiny는 개발이 어려운 만큼 생태계 활성화를 위해 다양한 예시를 쉽게 접할 수 있게 준비되어 있습니다.

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

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

위 코드는 실행해 볼 수 있는 예제를 보여줍니다. 아래처럼 실행하시면 바로 실행되는 shiny 앱 예제를 확인하실 수 있습니다.

[shiny github](#)에는 예제 repo가 있고, 활용할 수 있는 예제는 2017-11-16 기준 111개입니다.

# 예시 실행

runExample() 함수로 리스트 내의 예시들을 실행해 볼 수 있습니다.

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

# 다른 앱 실행방법

웹에서 제공하는 다른 사람의 소스를 바로 실행해 볼 수 있습니다.

```
# 기스트의 예시 앱 실행  
runGist()
```

```
# 깃헙의 예시 앱 실행  
runGitHub()
```

# runGist()

Gist란 스크립트 파일 몇 개만 공유할 때 유용한 서비스입니다. shiny 코드의 경우도 간단하게 한 두개 파일로만으로도 작성할 수 있으므로 Gist로 공유되는 경우도 있습니다.

```
runGist("https://gist.github.com/mrchypark/61283120fea4e392116503703547f39d")
```

# runGitHub()

github의 username, repo 이름, 폴더가 있다면 폴더명을 지정해주면 경로내의 shiny 앱을 가져와 실행해 줍니다. 물론 앱내 사용하는 패키지가 없다면 설치해줘야 합니다.

```
if (!requireNamespace("dplyr")){
  install.packages("dplyr")
}
if (!requireNamespace("shinydashboard")){
  install.packages("shinydashboard")
}

runGitHub(repo = "shiny-examples",username = "rstudio",subdir = "086-bus-dashboard")
```

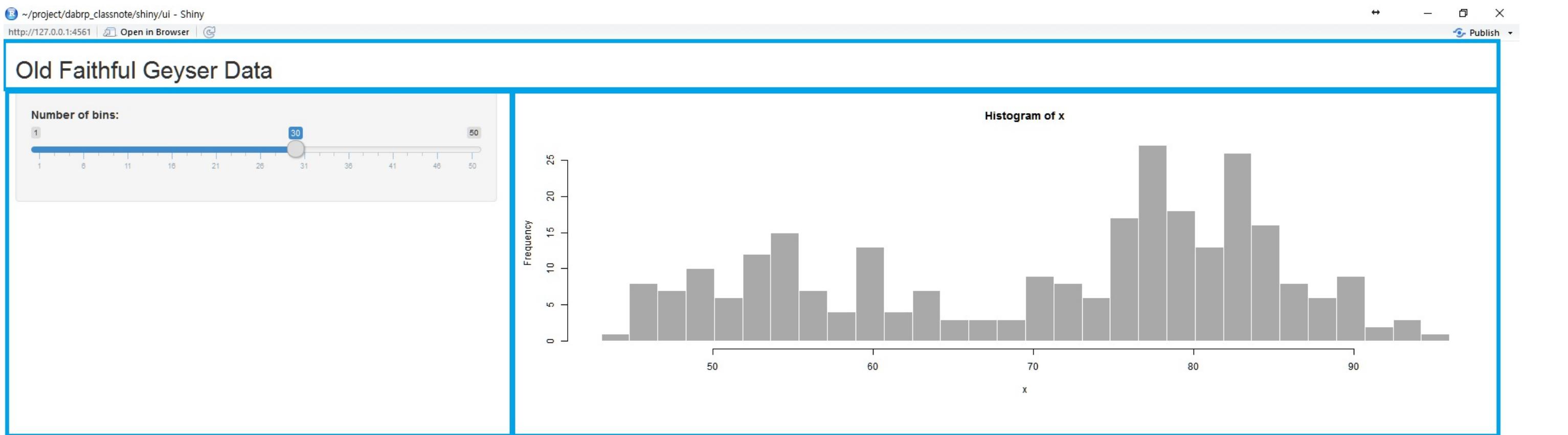
# 예시를 얻을 수 있는 곳

<https://github.com/rstudio/shiny-examples>은 공식적으로 잘된 사례를 모아 놓고 있습니다.

# shiny 패키지 구조

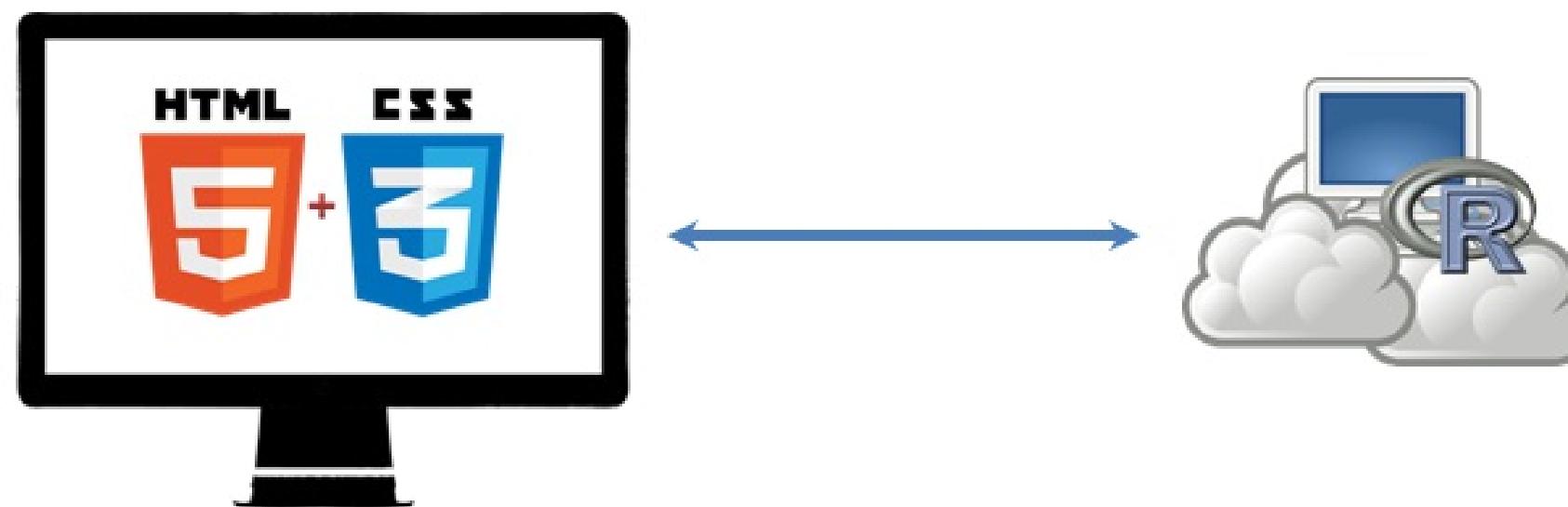
# shiny의 화면

ui라고 말하는 화면은 실제로 사용자가 보는 화면을 뜻합니다. shiny에서는 크게 titlePanel과 sidebarPanel, mainPanel의 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-server를 통해서 동작합니다. shiny-server는 shiny app이 동작할 수 있는 서버를 의미합니다. 오픈소스와 기업용 솔루션이 모두 준비되어 있으며 실습에는 <http://www.shinyapps.io/>를 이용해 보겠습니다.

# 최소 동작 앱 만들기

# shiny의 최소 코드

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

ui 만들기

# ui 만들기

ui는 화면의 위치를 나누는 함수와 입력을 받는 요소, 출력을 보여주는 요소로 구성되어 있습니다. \*는 여러 이름이 있다는 뜻입니다.

```
# *Page(), *Panel() 등의 함수로 위치를 나누고 모양을 결정
ui <- fluidPage(
  # *Input() 함수로 입력을 받는 요소들을 배치
  # *Output() 함수로 결과물을 출력하여 배치
)
```

# 페이지 구조 만들기

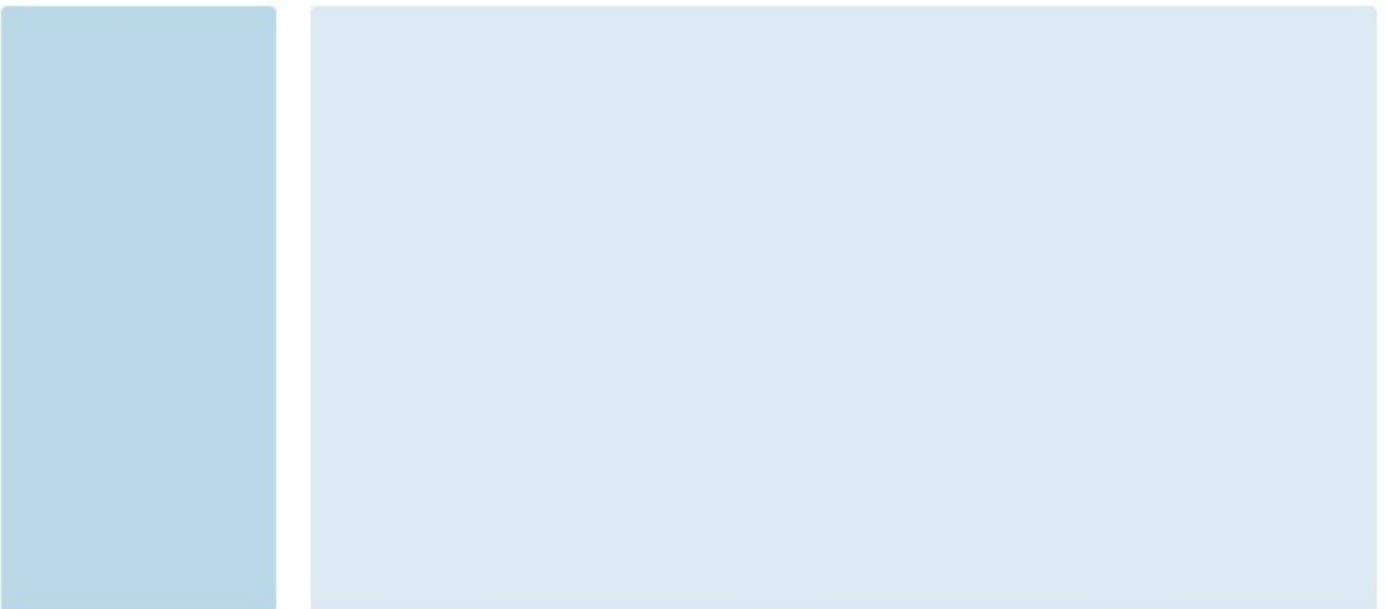
fluidPage()는 반응형 페이지를 만든다는 뜻입니다. fluidRow() 함수, column() 함수와 함께 사용합니다.

```
fluidPage(  
  fluidRow(  
    column(),  
    column()  
  )  
)
```

# column()

column() 은 총 화면 길이를 12개로 나누어서 사용합니다.

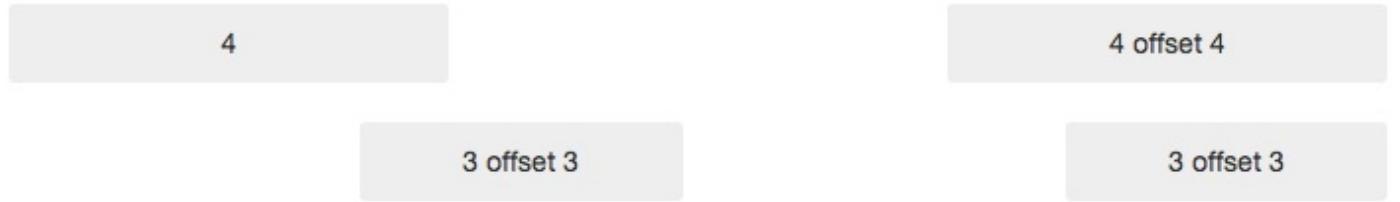
```
fluidPage(  
  fixedRow(  
    column(2), column(10)  
  )  
)
```



# offset 인자

offset 이란 그만큼 띄어낸다는 뜻입니다.

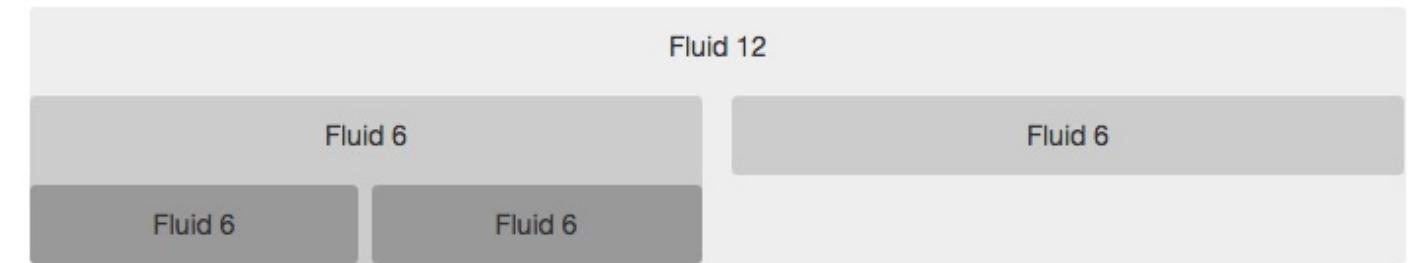
```
fluidPage(  
  fluidRow(  
    column(4,  
      "4"  
    ),  
    column(4, offset = 4,  
      "4 offset 4"  
    )  
  ),  
  fluidRow(  
    column(3, offset = 3,  
      "3 offset 3"  
    ),  
    column(3, offset = 3,  
      "3 offset 3"  
    )  
  )  
)
```



# column()내에 column() 사용

column()내에 column()을 추가할 수 있습니다. column()내에 column()은 다시 그 속한 column()을 통 12개로 나누어서 크기 를 지정합니다. column()내에 column()을 지정하기 전에 column()을 fluidRow()로 감싸야 동작합니다.

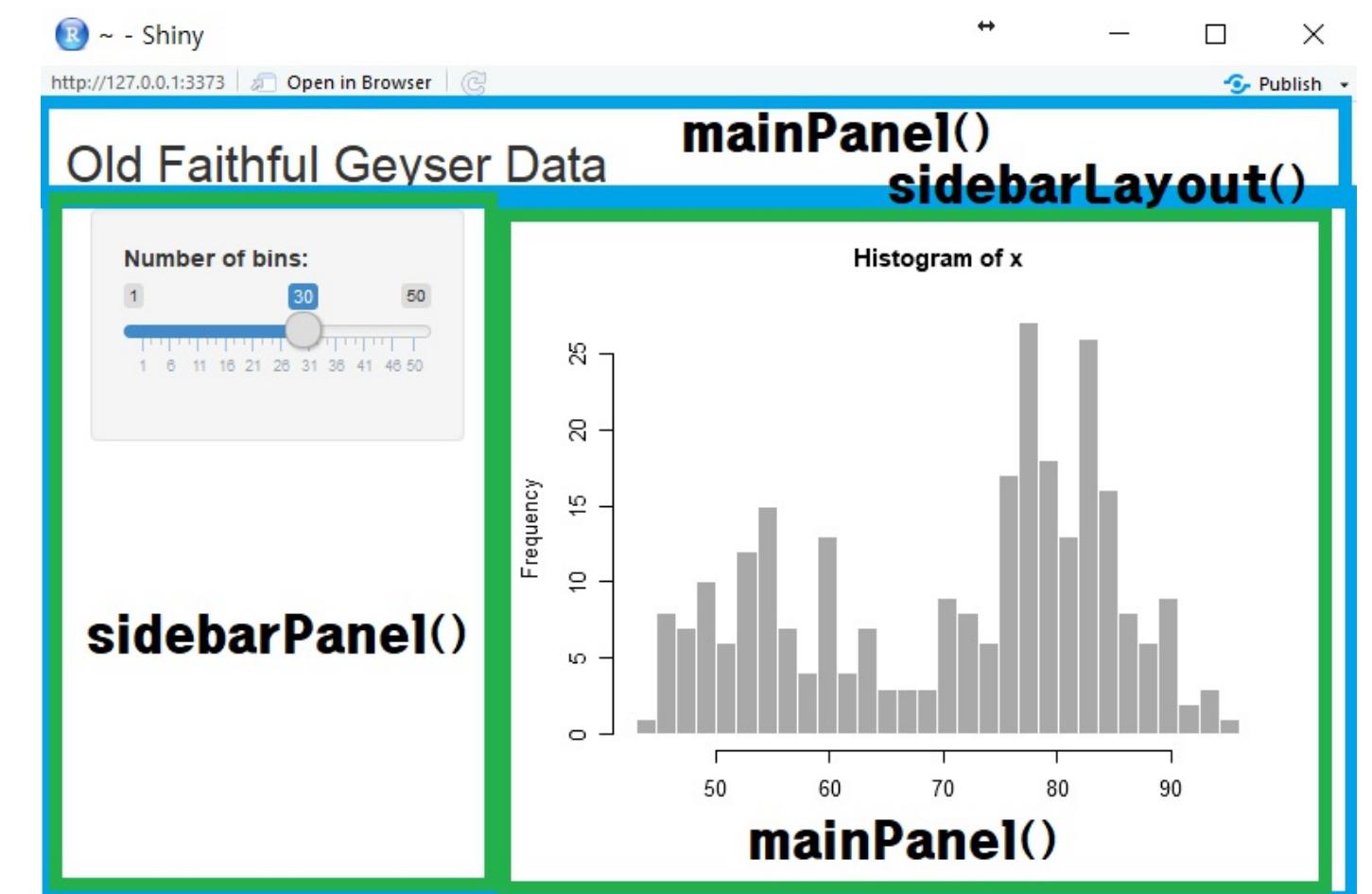
```
fluidPage(  
  fluidRow(  
    column(12,  
      "Fluid 12",  
      fluidRow(  
        column(6,  
          "Fluid 6",  
          fluidRow(  
            column(6,  
              "Fluid 6"),  
            column(6,  
              "Fluid 6"))  
        ),  
        column(width = 6,  
    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



# 어렵다면 \*Panel() 사용

\*Panel() 함수는 기본적으로 3가지(titlePanel(), sidebarPanel(), mainPanel())를 제공하며 구조는 아래와 같습니다.

```
fluidPage(  
  titlePanel(),  
  sidebarLayout(  
    sidebarPanel(),  
    mainPanel()  
)
```



sidebarLayout() 은 sidebarPanel()의 위치(왼쪽이 기본, 오른쪽)를 지정하기 위해 감싸는 함수로 왼쪽으로 되어 있다면 필요하지 않습니다.

# 다양한 \*Panel() 함수

\*Panel() 함수에는 navbarPage()과 navlistPanel(), tabsetPanel()도 tabPanel()과 함께 사용할 수 있습니다. 각각의 링크로 가서 예시코드로 동작을 확인하겠습니다.

# \*Input() 함수

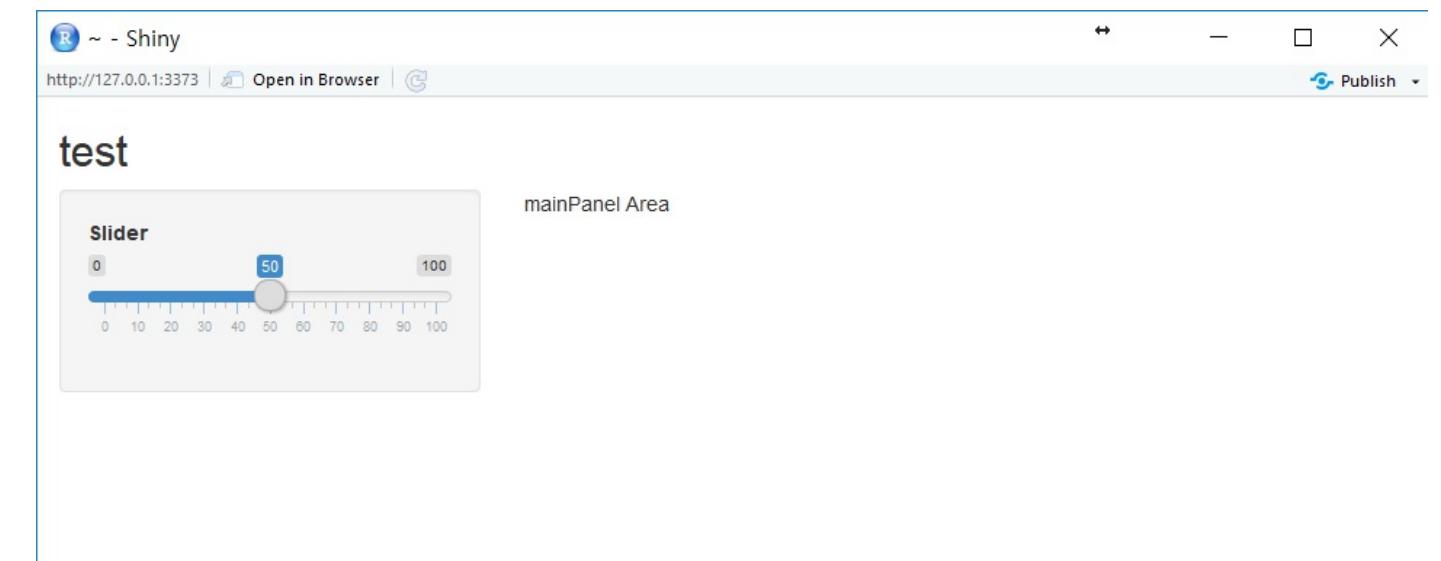
모양을 이쁘게 잡았다면, 이제 입력을 받을 도구들과 출력물을 배치할 차례입니다. 입력을 받을 도구는 [여기](#) 준비되어 있습니다.

The screenshot shows the Shiny Widgets Gallery interface. At the top, there is a navigation bar with the text "Shiny from R Studio" and links "Back to Gallery" and "Get Code". The main title "Shiny Widgets Gallery" is displayed prominently in a large white font on a blue background. Below the title, a descriptive text states: "For each widget below, the Current Value(s) window displays the value that the widget provides to shinyServer. Notice that the values change as you interact with the widgets." The interface is divided into three main sections, each demonstrating a different type of input:

- Action button**: This section contains a button labeled "Action". Below it, a "Current Value:" label is followed by a code block showing the result of `print(str(getClass(shinyActionButton)))` which outputs: [1] 0 attr(,"class") [1] "integer" "shinyActionButton". A "See Code" button is located at the bottom.
- Single checkbox**: This section contains a single checked checkbox labeled "Choice A". Below it, a "Current Value:" label is followed by a code block showing the result of `print(str(currentValue()))` which outputs: [1] TRUE. A "See Code" button is located at the bottom.
- Checkbox group**: This section contains three checkboxes labeled "Choice 1", "Choice 2", and "Choice 3", where "Choice 1" is checked. Below it, a "Current Values:" label is followed by a code block showing the result of `print(str(currentValues()))` which outputs: [1] "1". A "See Code" button is located at the bottom.

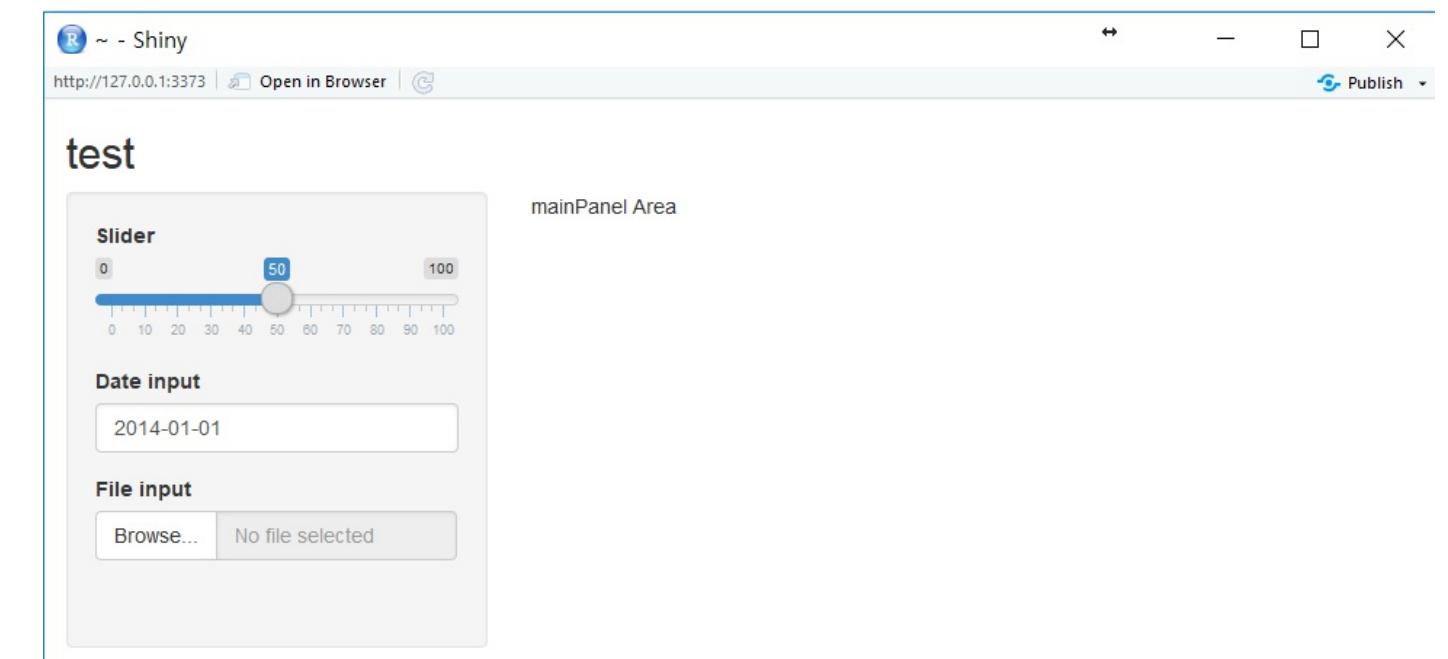
# \*Input()은 sidebarPanel()에 위치

```
fluidPage(  
  titlePanel("test"),  
  sidebarLayout(  
    sidebarPanel(  
      sliderInput(  
        inputId = "slider1"  
        , label = "Slider"  
        , min = 0  
        , max = 100  
        , value = 50  
      )  
    ),  
    mainPanel("mainPanel Area")  
  )  
)
```



# 여러개는 순차적으로 아래로 배치

```
fluidPage(  
  titlePanel("test"),  
  sidebarLayout(  
    sidebarPanel(  
      sliderInput(  
        inputId = "slider1"  
        , label = "Slider"  
        , min = 0  
        , max = 100  
        , value = 50  
      ),  
      dateInput(  
        inputId = "date"  
        , label = "Date input"  
        , value = "2014-01-01"  
      ),  
      fileInput(  
        inputId = "file"  
        , label = "File input"  
      )  
    ),  
    mainPanel("mainPanel Area")  
  )  
)
```



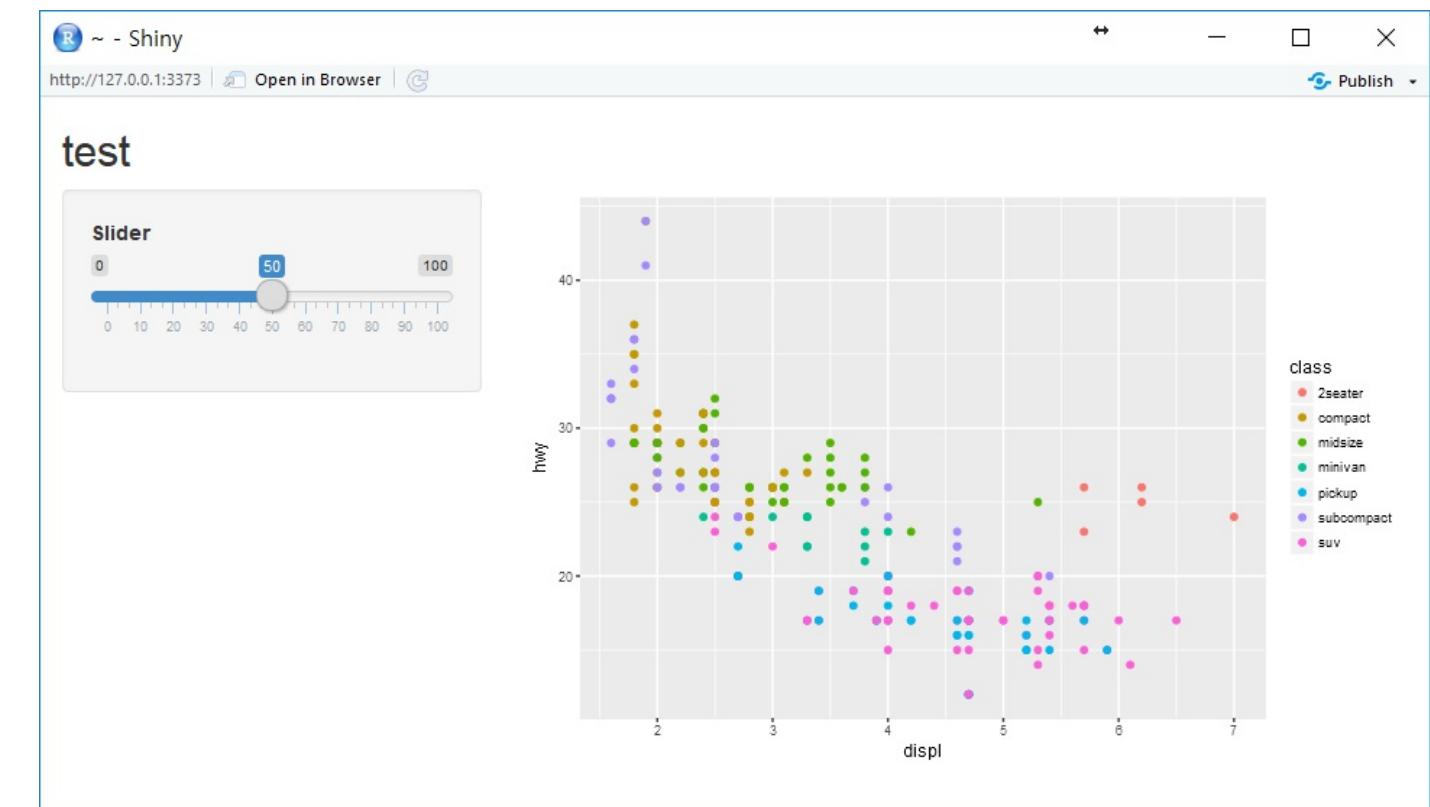
# \*outPut() 함수

input에서 입력값을 받아 서버가 처리한 결과물을 보여주는 곳으로 아래와 같은 함수들과 추가 패키지들이 제공하는 \*Output() 함수가 있습니다.

Function	Inserts
dataTableOutput()	인터렉티브 테이블
htmlOutput()	HTML 문서
imageOutput()	그림
plotOutput()	차트
tableOutput()	테이블
textOutput()	글자
verbatimTextOutput()	코드 글자

# \*Output()은 mainPanel()에 위치

```
fluidPage(  
  titlePanel("test"),  
  sidebarLayout(  
    sidebarPanel(  
      sliderInput(  
        inputId = "slider1"  
        , label = "Slider"  
        , min = 0  
        , max = 100  
        , value = 50  
      )  
    ),  
    mainPanel(  
      plotOutput(  
        outputId = "plot1"  
      )  
    )  
  )
```



# 각 변수의 Id

Input() 함수와 Output() 함수는 사용할 변수에 대한 Id를 입력받습니다. 전부 글자형(character)로 구성됩니다. inputId는 서버에서 input\$inputId의 형태로 사용하며 아래의 경우 input\$slider1입니다. outputId는 서버에서 만든 결과물을 전달 받기 위해서 사용하며 서버에서 저장할 때는 output\$outputId로 사용합니다. 아래의 경우 output\$plot1입니다.

```
# inputId  
sliderInput(inputId = "slider1", label = "slider", min = 0, max = 100, value = 50)  
  
# outputId  
plotOutput(outputId = "plot1")
```

# server 만들기

# server 만들기

server는 아까 `Input()` 함수로 입력받은 데이터를 사용하기 위한 `input` 변수와 `Output()` 함수로 출력하기 위한 결과물을 저장하는 `output` 변수, 마지막으로 출력 결과물인 R 객체를 web의 세상에서 사용할 수 있는 상태로 `output` 변수에 저장하는 `render*({})` 함수로 구성됩니다.

```
server <- function(input, output){  
  output$plot1 <- renderPlot({  
    plot(input$slider1)  
  })  
}
```

# shiny의 입출력

# 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 보여줘야 할 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 Output이고, "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. 여기서는 ({ }) 안쪽은 R의 세계이고, 그 바깥은 웹의 세계라고 이해하겠습니다. 그래서 이런게 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.

# render\*() 함수

output() 함수에 대응되는 render() 함수가 모두 있습니다.

Output.func	render.func	Inserts
dataTableOutput()	renderDataTable()	인터랙티브 테이블
imageOutput()	renderImage()	그림
plotOutput()	renderPlot()	차트
tableOutput()	renderTable()	테이블
textOutput()	renderText()	글자
verbatimTextOutput()	renderText()	코드 글자

# 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$sdata1`이라는 곳에 웹 데이터를 저장하는 것을 뜻합니다. `*Input()` 패밀리 함수는 같은 규칙을 가지는데 입력형태명 `Input()`의 함수명을 가지고, 첫번째 인자는 `input$` 뒤로 붙을 컬럼명, 두번째 인자는 화면에 보여줄 글자를 의미합니다. 나머지 인자는 입력형태에 따라 다양하게 달라집니다.

# 실습 1

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

우선 서버쪽부터 보겠습니다.

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

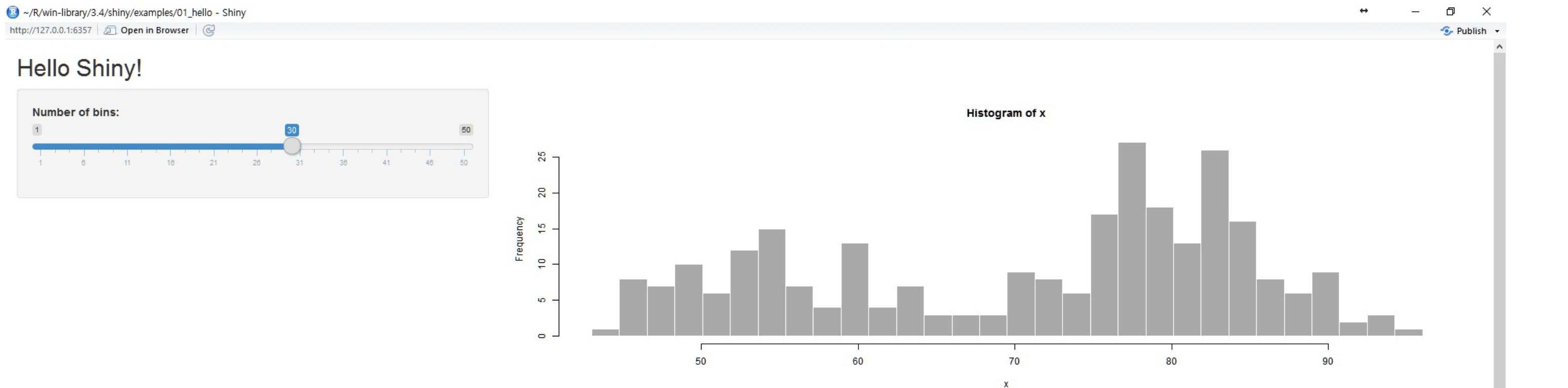
# server를 function으로 선언합니다. 웹의 세계에서 사용할 변수는 input과 output입니다.
function(input, output) {

  # 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 함수인지 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$bins`가 `render*`(`{}`) 함수 안에서 사용됐습니다. `output$`은 웹의 세계에서 사용하지만 `input$`은 R의 세계에서 사용합니다. 이부분은 `reactive`(`{}`) 함수에서 추가적으로 다루기로 하겠습니다. 이제 ui쪽 코드를 확인해 보겠습니다.



The screenshot shows a Shiny application interface. At the top, there's a browser header with the URL `http://127.0.0.1:6357` and a 'Publish' button. Below the header, the application title 'Hello Shiny!' is displayed. On the left, there's a sidebar with a slider titled 'Number of bins:' ranging from 1 to 50, with the value set to 30. To the right of the sidebar is a histogram titled 'Histogram of x'. The x-axis is labeled 'x' and ranges from 50 to 90. The y-axis is labeled 'Frequency' and ranges from 0 to 25. The histogram bars show a distribution that peaks around x=75. In the bottom right corner of the slide, there's a code editor window showing the 'ui.R' file of the Shiny app. The code defines the UI components, including the title and the sidebar with the slider. The 'server.R' tab is also visible in the code editor.

```
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")  
    )  
  )  
)
```

# 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.

# reactive한 현 상황

reactive는 반응성으로 번역하기도 합니다. 반응성이라고 하는 이유는 `input$bins`의 값이 변하는 순간을 `render({ })`가 감지해서 변한 값을 반영하여 `output$distPlot`에 저장하고, 그것을 `plotOutput("distPlot")`으로 전달하여 그림을 새로 그리기 때문입니다. `input`의 변화에 반응하여 `output`이 변화하니 간단한 반응성으로 연결된 것입니다.

# 반응형 함수

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

## 중요한 반응형 함수

`reactive({}), isolate()`

# reactive({})로 render\*({}) 외부로 연산 분리

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$`은 R의 세상에 있고, `output$`은 웹의 세상에 있습니다. `render*({})`함수를 통해서 R의 세상의 결과물을 `output$`에 전달한다고 했구요. 이 과정에서 `render*({})` 함수는 목적이 결과물을 전달하는데 있습니다. `input$`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 만듬으로써 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. `selectedData()` 함수는 앞에서 선언한 `reactive({ get(input$dataSelect) })` 부분이 변하는지를 보고 있다가 변하면 결과물을 R의 세상인 `render*({})` 함수 내부에서 사용할 있게 만들어 줍니다.

# reactive({})의 동작

1. input\$ 값을 주시
2. 변화한 값을 저장
3. 변화한 값을 전달

reactive({})와 render\*({})는 내부에 품고 있는 input\$ 변수의 변화를 계속 지켜보고 있습니다. 그리고 변화에 따라 순서대로 위 동작을 수행합니다. 전달 때문에 화면에 결과물이 변경되어서 보여지는 것입니다.

# isolate()로 대기

지금까지는 `input`이 변하는 대로 일일이 반응하는 형태의 반응성을 구현했습니다. 그런데 타이핑 같이 계속 빠른 변화가 있는 경우 오히려 성능의 저하를 가져오기 때문에 일일이 반응하지 말라고 해줘야 할 때도 있습니다. `isolate()`은 `render*({})`나 `reactive({})` 안쪽에서 사용해서 `input`의 변화가 `output` 쪽으로 전달되지 않고 값만 가지고 있게 할 수 있습니다. 즉, `reactive({})`의 동작 중에 `isolate()` 함수 내부의 `input$`은 2번 저장까지만 진행되고 3번의 전달은 진행되지 않습니다. `isolate()` 함수 밖의 다른 `input$`이 변화하여 값을 전달하는 일이 발생할 때 그 전까지 저장해둔 모든 값이 전달됩니다.

# 반응성과 isolate() 함수 적용 비교

```
library(shiny)
ui <- fluidPage(
  headerPanel("Click the button"),
  sidebarPanel(
    sliderInput("obs1", "Number of obs1:",
               min = 0, max = 1000
               , value = 500),
    sliderInput("obs2", "Number of obs2:",
               min = 0, max = 1000
               , value = 500),
    actionButton("goButton", "Go!")
  ),
  mainPanel(
    plotOutput("distPlot1"),
    plotOutput("distPlot2")
  )
)
```

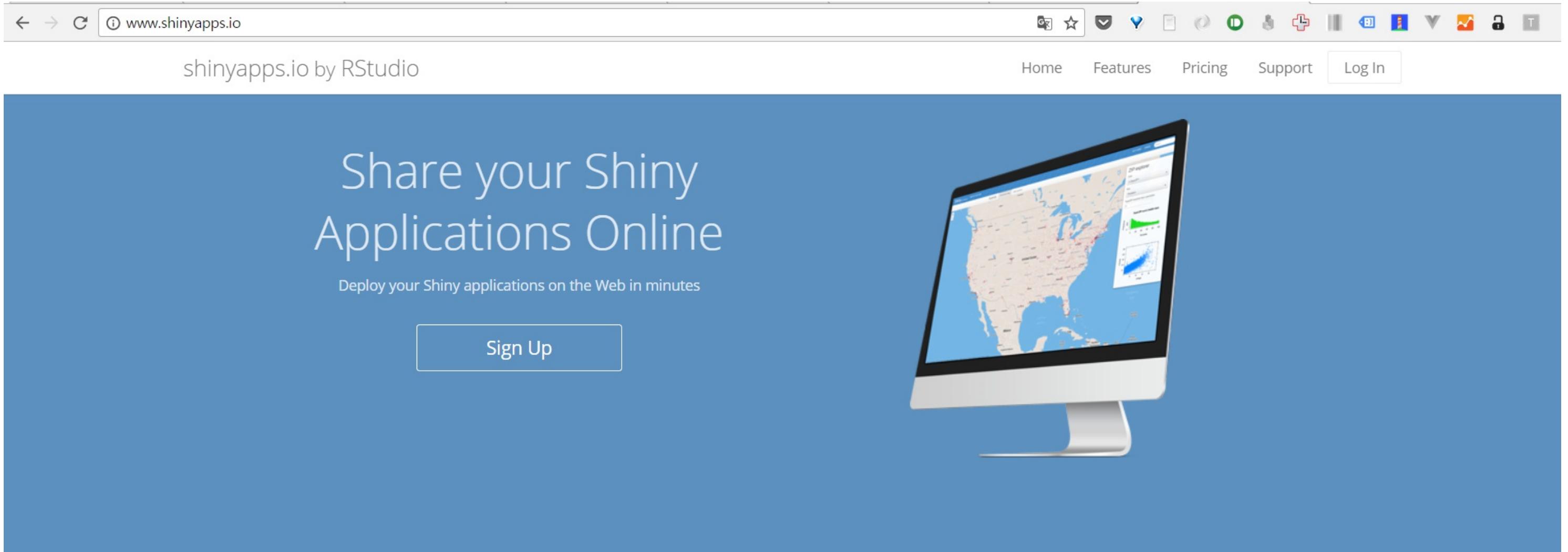
```
server<- function(input, output) {
  output$distPlot1 <- renderPlot({
    dist <- rnorm(input$obs1)
    hist(dist)
  })

  output$distPlot2 <- renderPlot({
    input$goButton

    dist <- isolate(rnorm(input$obs1))
    hist(dist)
  })
}
shinyApp(ui, server)
```

# shiny의 배포

# shinyapps.io



The screenshot shows the shinyapps.io homepage. At the top, there's a browser header with a back/forward button, a refresh icon, and the URL "www.shinyapps.io". To the right of the URL are various browser extensions and icons. Below the header, the page title "shinyapps.io by RStudio" is displayed. On the right side, there are navigation links for "Home", "Features", "Pricing", "Support", and a "Log In" button. The main content area has a blue background. It features the text "Share your Shiny Applications Online" in large white font, followed by "Deploy your Shiny applications on the Web in minutes". A "Sign Up" button is located below this text. To the right of the text is an image of a computer monitor displaying a Shiny application with a map of the United States. At the bottom of the page, there are three circular icons with text below them: "Easy To Use" (cloud icon), "Secure" (lock icon), and "Scalable" (lightning bolt icon). The page number "60 / 83" is visible in the bottom right corner.

shinyapps.io by RStudio

Home Features Pricing Support Log In

Share your Shiny Applications Online

Deploy your Shiny applications on the Web in minutes

Sign Up

Easy To Use

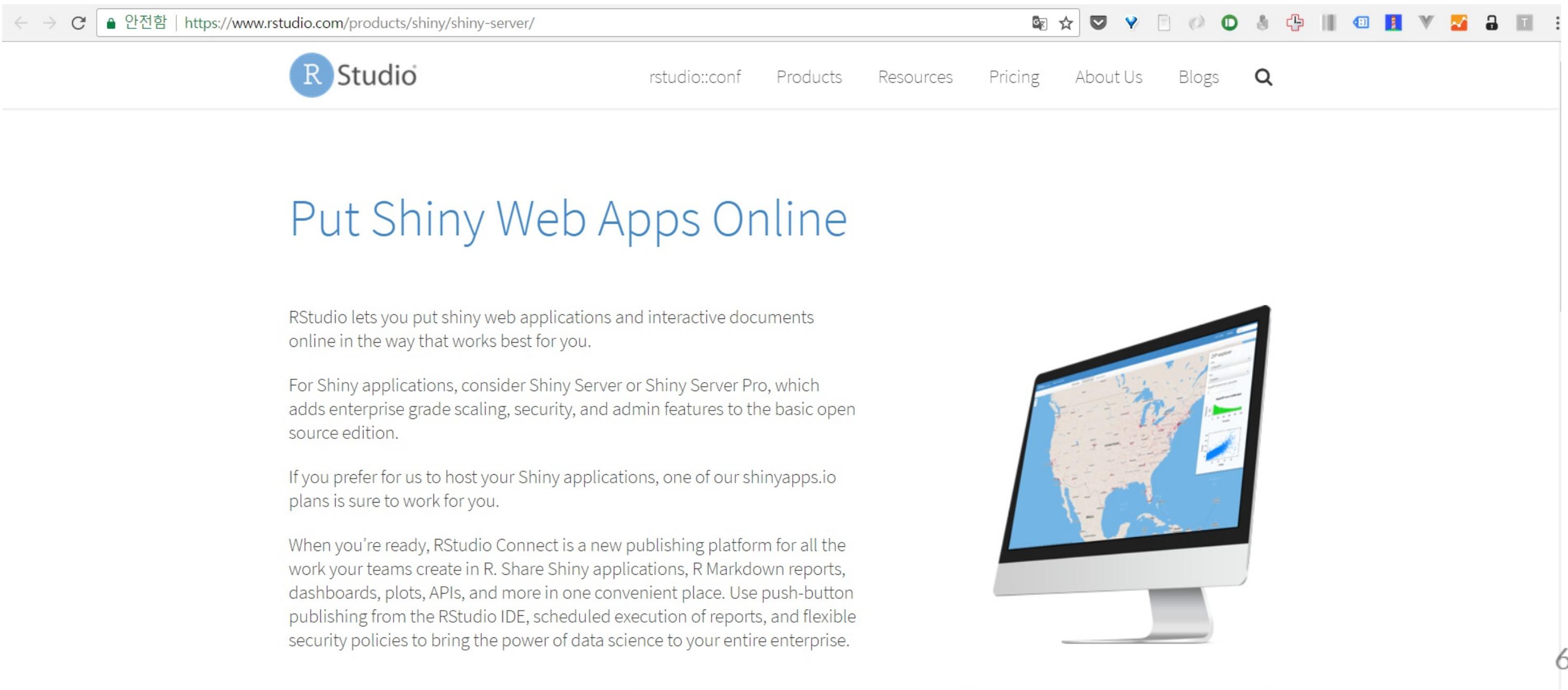
Secure

Scalable

60 / 83

# shiny server 구축

- 오픈소스 버전 [다운로드와 가이드](#)



The screenshot shows a web browser displaying the RStudio products page for Shiny Server. The URL in the address bar is <https://www.rstudio.com/products/shiny/shiny-server/>. The page features a large title "Put Shiny Web Apps Online". Below it, text explains that RStudio lets you put shiny web applications and interactive documents online. It mentions Shiny Server and Shiny Server Pro for enterprise needs, and shinyapps.io for hosting. It also introduces RStudio Connect as a publishing platform. To the right, there's an image of a computer monitor displaying a Shiny application with a map of the United States.

안전함 | <https://www.rstudio.com/products/shiny/shiny-server/>

R Studio

rstudio::conf Products Resources Pricing About Us Blogs

## Put Shiny Web Apps Online

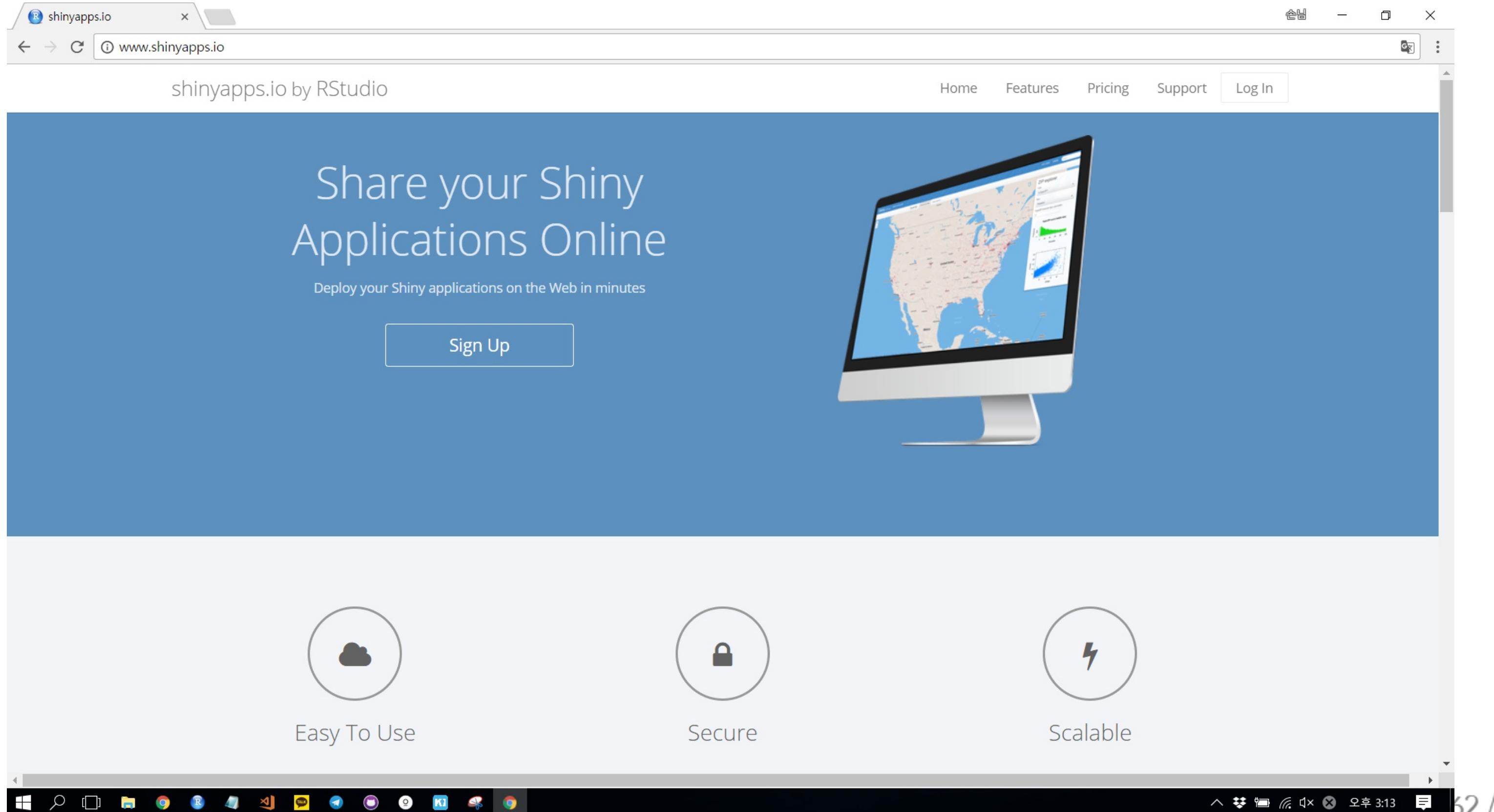
RStudio lets you put shiny web applications and interactive documents online in the way that works best for you.

For Shiny applications, consider Shiny Server or Shiny Server Pro, which adds enterprise grade scaling, security, and admin features to the basic open source edition.

If you prefer for us to host your Shiny applications, one of our shinyapps.io plans is sure to work for you.

When you're ready, RStudio Connect is a new publishing platform for all the work your teams create in R. Share Shiny applications, R Markdown reports, dashboards, plots, APIs, and more in one convenient place. Use push-button publishing from the RStudio IDE, scheduled execution of reports, and flexible security policies to bring the power of data science to your entire enterprise.

# shinyapps로 웹앱 배포 연습



## 회원가입

The screenshot shows a web browser window with the URL <https://www.shinyapps.io/admin/#/signup>. The page has a blue gradient background and features the shinyapps.io logo at the top center. Below the logo are three input fields: 'Email', 'Password', and 'Confirm Password'. A small text link below the fields reads: 'By clicking log in or sign up, you agree to the shinyapps.io terms of use.' At the bottom of the page are three large blue buttons labeled 'Sign Up', '— or —', 'Sign Up with Google', and 'Sign Up with GitHub'.

shinyapps.io

Email

Password

Confirm Password

By clicking log in or sign up, you agree to the shinyapps.io terms of use.

Sign Up

— or —

Sign Up with Google

Sign Up with GitHub

## 정보 입력

The screenshot shows a web browser window for shinyapps.io. The address bar displays the URL <https://www.shinyapps.io/admin/#/signup>. The page itself is a sign-up form for shinyapps.io. It features a large logo at the top center. Below the logo, there are input fields for email (containing rcoholic@gmail.com), password (containing a masked value), and confirmation password (also containing a masked value). A note below the fields states: "By clicking log in or sign up, you agree to the shinyapps.io terms of use." At the bottom of the form are three blue buttons: "Sign Up", "— or —", "Sign Up with Google", and "Sign Up with GitHub". The browser's taskbar at the bottom shows various pinned icons, and the system tray on the right indicates the date and time as "오늘 3:13".

shinyapps.io

✉ rcoholic@gmail.com

🔒 .....

🔒 .....

By clicking log in or sign up, you agree to the shinyapps.io terms of use.

Sign Up

— or —

Sign Up with Google

Sign Up with GitHub

로그인 화면

The screenshot shows the shinyapps.io dashboard with a blue header. The left sidebar has 'Dashboard', 'Applications', and 'Account' sections. The main content area is titled 'GETTING STARTED' with a message: 'Hi! You must be new here...'. It explains the need to install the `rsconnect` R package. Below is 'STEP 1 - INSTALL RSCONNECT' with the command `install.packages('rsconnect')`. 'STEP 2 - AUTHORIZE ACCOUNT' shows the command `rsconnect::setAccountInfo(name='rcoholic', token='D5CC64CECC5852B74004664571B63432', secret='<SECRET>')` and buttons for 'Show secret' and 'Copy to clipboard'. 'STEP 3 - DEPLOY' is partially visible at the bottom.

shinyapps.io

ggplot

안전함 | https://www.shinyapps.io/admin/#/dashboard

shinyapps.io

Help Account: rcoholic

Dashboard

Applications

Account

GETTING STARTED

Hi! You must be new here...

Thanks for trying out shinyapps.io! You'll need to install the `rsconnect` R package to get started. The `rsconnect` package enables you to deploy and manage your Shiny applications directly from your R console. To get started, fire up your favorite IDE, and follow the directions below.

STEP 1 – INSTALL RSCONNECT

The `rsconnect` package can be installed directly from CRAN. To make sure you have the latest version run following code in your R console:

```
install.packages('rsconnect')
```

STEP 2 – AUTHORIZE ACCOUNT

The `rsconnect` package must be authorized to your account using a token and secret. To do this, click the copy button below and we'll copy the whole command you need to your clipboard. Just paste it into your console to authorize your account. Once you've entered the command successfully in R, that computer is now authorized to deploy applications to your shinyapps.io account.

```
rsconnect::setAccountInfo(name='rcoholic',  
token='D5CC64CECC5852B74004664571B63432',  
secret='<SECRET>')
```

Show secret

Copy to clipboard

In the future, you can manage your tokens from the [Tokens](#) page the settings menu.

STEP 3 – DEPLOY

Once the `rsconnect` package has been configured, you're ready to deploy your first application. If you haven't written any applications yet, you can also checkout the [Getting Started](#).

## step1,2 진행

The screenshot shows a web browser window with the shinyapps.io dashboard. The browser tabs include 'shinyapps.io' and 'ggplot'. The address bar shows '안전함 | https://www.shinyapps.io/admin/#/dashboard'. The shinyapps.io logo is at the top left, and the user account 'rcoholic' is at the top right.

**STEP 1 - INSTALL RCONNECT**

The `rsconnect` package can be installed directly from CRAN. To make sure you have the latest version run following code in your R console:

```
install.packages('rsconnect')
```

**STEP 2 - AUTHORIZE ACCOUNT**

The `rsconnect` package must be authorized to your account using a token and secret. To do this, click the copy button below and we'll copy the whole command you need to your clipboard. Just paste it into your console to authorize your account. Once you've entered the command successfully in R, that computer is now authorized to deploy applications to your shinyapps.io account.

```
rsconnect::setAccountInfo(name='rcoholic',
                           token='D5CC64CECC5852B74004664571B63432',
                           secret='<SECRET>')
```

**Show secret**

**Copy to clipboard**

In the future, you can manage your tokens from the [Tokens](#) page the settings menu.

**STEP 3 - DEPLOY**

Once the `rsconnect` package has been configured, you're ready to deploy your first application. If you haven't written any applications yet, you can also checkout the [Getting Started Guide](#) for instructions on how to deploy our demo application. Run the following code in your R console.

```
library(rsconnect)
rsconnect::deployApp('path/to/your/app')
```

The bottom of the screen shows the Windows taskbar with various pinned icons and the system tray.

# Copy to clipboard 클릭 후 내용 복사

The screenshot shows a web browser window with the URL <https://www.shinyapps.io/admin/#/dashboard>. The browser has two tabs open: 'shinyapps.io' and 'ggplot'. The main content area displays the shinyapps.io dashboard with three main sections: 'STEP 1 - INSTALL RCONNECT', 'STEP 2 - AUTHORIZE ACCOUNT', and 'STEP 3 - DEPLOY'.

**STEP 1 - INSTALL RCONNECT**  
The `rsconnect` package can be installed directly from CRAN. Copying the command:

```
install.packages('rsconnect')
```

**STEP 2 - AUTHORIZE ACCOUNT**  
The `rsconnect` package must be authorized to your account using a token and secret. Click the copy button below and we'll copy the whole command you need to your clipboard. Just paste it into your console to authorize your account. Once you've entered the command successfully in R, that computer is now authorized to deploy applications to your shinyapps.io account.

```
rsconnect::setAccountInfo(name='rcoholic',  
                          token='D5CC64CECC5852B74004664571B63432',  
                          secret='<SECRET>')
```

Buttons: Show secret (green), Copy to clipboard (green)

In the future, you can manage your tokens from the [Tokens](#) page the settings menu.

**STEP 3 - DEPLOY**  
Once the `rsconnect` package has been configured, you're ready to deploy your first application. If you haven't written any applications yet, you can also checkout the [Getting Started Guide](#) for instructions on how to deploy our demo application. Run the following code in your R console.

```
library(rsconnect)  
rsconnect::deployApp('path/to/your/app')
```

At the top right of the dashboard, there is a modal dialog titled 'www.shinyapps.io 내용:' (Content of www.shinyapps.io) with the following text:  
Copy to clipboard: Ctrl+C, Enter  
`rsconnect::setAccountInfo(name='rcoholic', token='D5CC64CECC5852B74004664571B63432', secret='<SECRET>')`

Buttons: 확인 (Confirm), 취소 (Cancel)

At the bottom right of the screen, there is a status bar with icons for battery, signal, and volume, and the text '가 오후 3:21'.

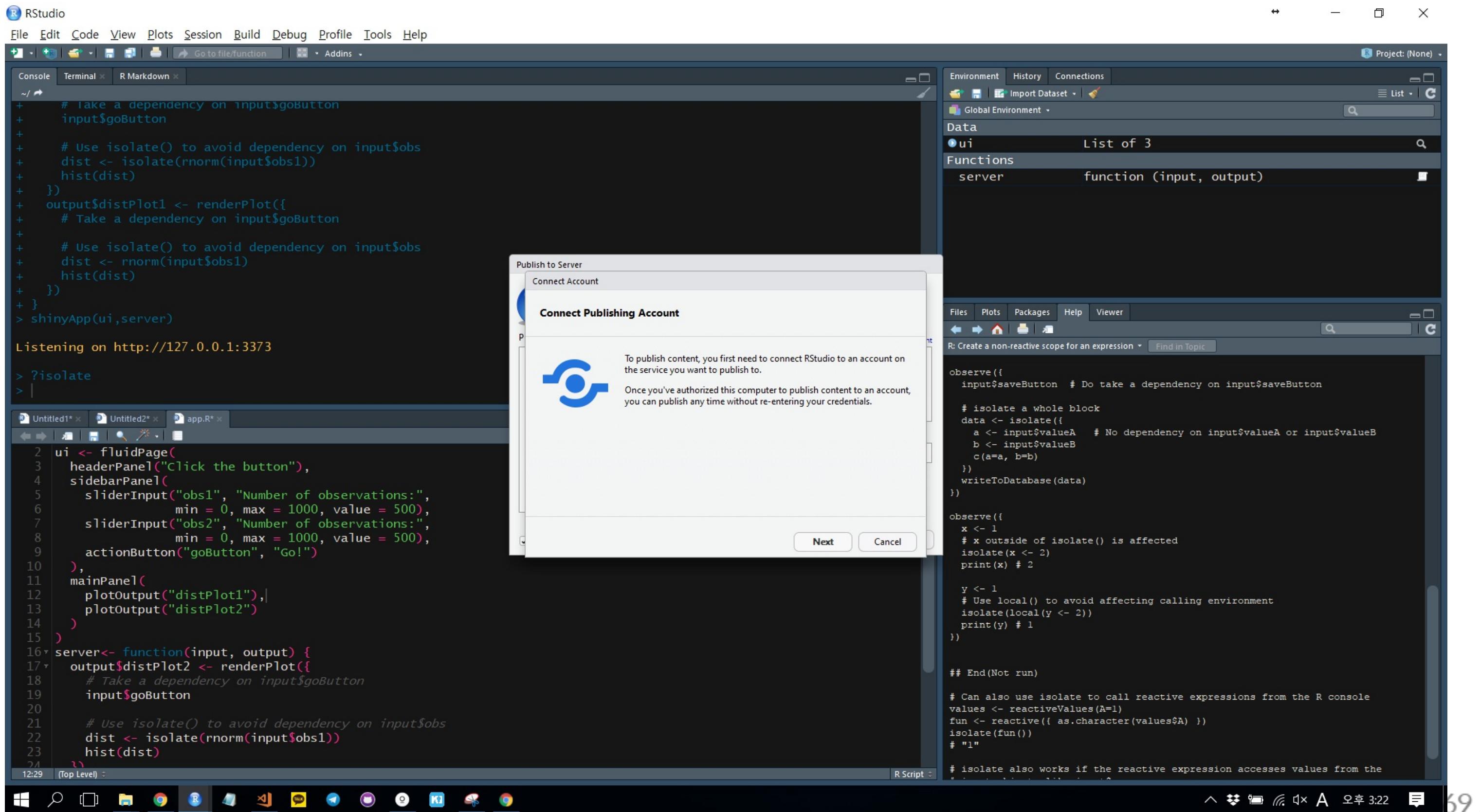
# publish 클릭

The screenshot shows the RStudio interface with the following components:

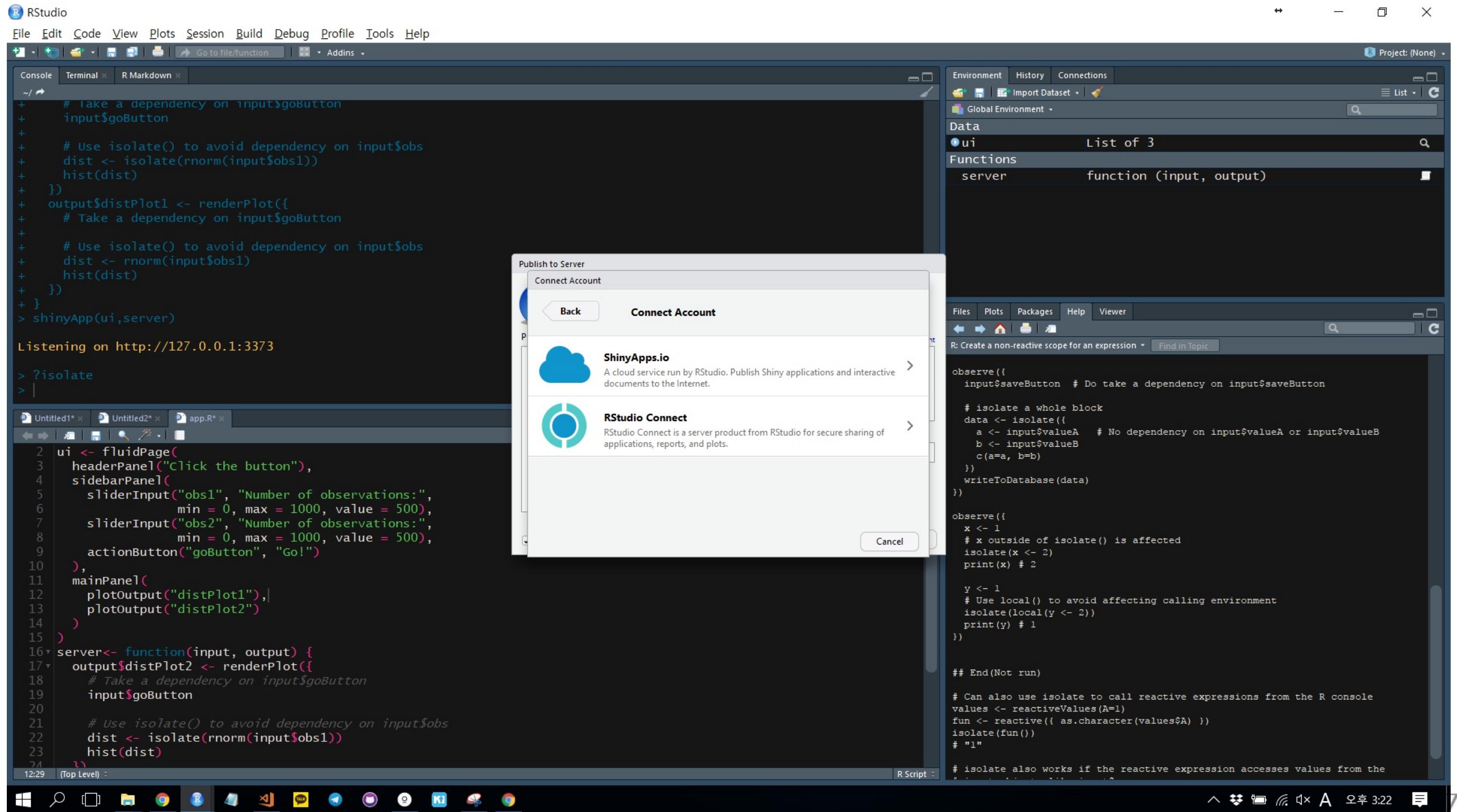
- Top Bar:** File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Help.
- Toolbar:** Go to file/function, Addins.
- Code Editor:** An R script titled "app.R" containing Shiny application code. A red box highlights the "Run App" button in the toolbar above the editor.
- Environment Tab:** Shows the global environment with objects like "data", "ID\_1", "ID\_2", "ui", "Data\_val", "date", "employee\_name", "ID\_no", and "IDchoices".
- Files Tab:** Shows the project structure with files like ".fontconfig", ".Rhistory", "FeedbackHub", "GitHub", "project", "R", "sdf", "test.Rmd", "TOBESOFT", "WindowsPowerShell", "사용자 지정 Office 서식 파일", "챗봇고객데이터.xlsx", and "카카오톡 받은 파일".
- Console:** Displays the R session output, including the server definition and the command "shinyApp(ui=ui, server=server)". It also shows the message "Listening on http://127.0.0.1:3373".
- Bottom Taskbar:** Windows taskbar with various icons.
- Page Footer:** 오후 4:02, 58 / 83.

A large watermark "publish 버튼" is overlaid on the center of the screen.

next



# shinyapps.io 선택



# 입력칸에 아까 복사한 내용 붙여넣기

The screenshot shows the RStudio interface with a 'Publish to Server' dialog box open in the foreground. The dialog is titled 'Connect ShinyApps.io Account' and contains instructions for connecting to a ShinyApps.io account. It includes a 'Back' button, a 'Connect Account' button, and a 'Cancel' button. The main content area features a blue cloud icon and text explaining the steps to log in and copy a token. Below the dialog, the RStudio environment is visible, showing the console output, code editor, and file browser.

RStudio Environment:

- Console tab active.
- Code editor shows R script content.
- File browser shows a 'Publish to Server' dialog.
- Environment tab shows global variables and functions.
- Plots tab shows a histogram.
- Packages tab shows available packages.
- Help tab shows help documentation.
- Viewer tab shows a preview of the Shiny app.

Console Output (Top Left):

```
# Take a dependency on input$goButton
input$goButton

# Use isolate() to avoid dependency on input$obs
dist <- isolate(rnorm(input$obs1))
hist(dist)

output$distPlot1 <- renderPlot({
  # Take a dependency on input$goButton

  # Use isolate() to avoid dependency on input$obs
  dist <- rnorm(input$obs1)
  hist(dist)
})

shinyApp(ui,server)
```

Listening on http://127.0.0.1:3373

> ?isolate

Code Editor (Bottom Left):

```
ui <- fluidPage(
  headerPanel("Click the button"),
  sidebarPanel(
    sliderInput("obs1", "Number of observations:",
               min = 0, max = 1000, value = 500),
    sliderInput("obs2", "Number of observations:",
               min = 0, max = 1000, value = 500),
    actionButton("goButton", "Go!")
  ),
  mainPanel(
    plotOutput("distPlot1"),
    plotOutput("distPlot2")
  )
)

server<- function(input, output) {
  output$distPlot2 <- renderPlot({
    # Take a dependency on input$goButton
    input$goButton

    # Use isolate() to avoid dependency on input$obs
    dist <- isolate(rnorm(input$obs1))
    hist(dist)
  })
}
```

File Browser (Bottom Right):

```
observe({
  input$saveButton # Do take a dependency on input$saveButton

  # isolate a whole block
  data <- isolate({
    a <- input$valueA # No dependency on input$valueA or input$valueB
    b <- input$valueB
    c(a=a, b=b)
  })
  writeToDatabase(data)
}

observe({
  x <- 1
  # x outside of isolate() is affected
  isolate(x <- 2)
  print(x) # 2
}

## End(Not run)

# Can also use isolate to call reactive expressions from the R console
values <- reactiveValues(A=1)
fun <- reactive({ as.character(values$A) })
isolate(fun())
# "1"

# isolate also works if the reactive expression accesses values from the
```

# Connect Account 클릭

The screenshot shows the RStudio interface with a 'Publish to Server' dialog box in the foreground. The dialog box is titled 'Connect Account' and contains instructions for connecting to ShinyApps.io. It includes a 'Back' button, a 'Connect' button, and a 'Cancel' button. A large blue cloud icon is present. The text in the dialog box reads:

Go to [your account on ShinyApps](#) and log in.  
Click your name, then choose **Tokens** from your account menu.  
Click **Show** on the token you want to use, then **Show Secret and Copy to Clipboard**. Paste the result here:

```
rsconnect::setAccountInfo(name='rcoholic',
token='[REDACTED]', # Redacted for security
secret='[REDACTED]')
```

Need a ShinyApps.io account? [Get started here.](#)

The background of the RStudio interface shows the following:

- Console Tab:** Displays R code for a shinyApp. The code includes comments about dependencies and the use of isolate(). It ends with a message: "Listening on http://127.0.0.1:3373".
- Code Editor:** Shows the same R code as the Console tab.
- Environment Tab:** Shows the global environment with objects: ui (List of 3), Functions (server), and Data (server).
- Plots Tab:** Shows a histogram plot.
- Packages Tab:** Shows a list of packages.
- Help Tab:** Shows help documentation for the observe() function.
- Viewer Tab:** Shows the R code for the shinyApp.

The status bar at the bottom shows the time as 12:29 and the page number as 72 / 83.

## 배포할 이름 결정 후 Publish 클릭

The screenshot shows the RStudio interface with a 'Publish to Server' dialog box in the foreground. The dialog box has the following fields:

- Publish Files From:** ~/sdf
- app.R** (checkbox checked)
- Publish From Account:** rcoholic: shinyapps.io
- Title:** test
- Launch browser** (checkbox checked)
- Publish** and **Cancel** buttons

In the background, the RStudio environment shows:

- Console:** Contains R code for a shinyApp, including comments about dependencies and isolate(). It ends with "Listening on http://127.0.0.1:3373".
- Code Editor:** Shows the app.R file with the same code as the console.
- Environment:** Shows the global environment with objects ui, server, and a list of 3.
- Plots:** Shows two plots labeled distPlot1 and distPlot2.
- Help:** A help topic for the observe() function.
- Bottom:** A taskbar with various icons and the status bar showing the date and time.

# 배포 진행중

The screenshot shows the RStudio interface with the following components:

- Top Bar:** RStudio, File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Help.
- Console Tab:** Preparing to deploy application...DONE  
Uploading bundle for application: 236082...Detecting system locale ... ko\_KO  
DONE  
Deploying bundle: 1078301 for application: 236082 ...  
Waiting for task: 496199272  
building: Processing bundle: 1078301  
building: Building image: 1078366  
building: Installing packages  
building: Installing files
- Environment Tab:** Global Environment, Data, Functions, server, List of 3
- Files Tab:** Run App, Untitled1\*, Untitled2\*, app.R\*
- Code Editor:** R Script, containing the following code:

```
ui <- fluidPage(  
  headerPanel("Click the button"),  
  sidebarPanel(  
    sliderInput("obs1", "Number of observations:",  
               min = 0, max = 1000, value = 500),  
    sliderInput("obs2", "Number of observations:",  
               min = 0, max = 1000, value = 500),  
    actionButton("goButton", "Go!")  
)  
  mainPanel(  
    plotOutput("distPlot1"),  
    plotOutput("distPlot2")  
)  
}  
server<- function(input, output) {  
  output$distPlot2 <- renderPlot({  
    # Take a dependency on input$goButton  
    input$goButton  
  
    # Use isolate() to avoid dependency on input$obs  
    dist <- isolate(rnorm(input$obs1))  
    hist(dist)  
  })  
}
```
- Help Tab:** R: Create a non-reactive scope for an expression, observe(), isolate()
- System Taskbar:** Windows icon, search, file, settings, network, battery, volume, clock, 74 / 83.

# 배포 완료 - 이름 클릭하여 세부사항 확인

The screenshot shows the shinyapps.io dashboard interface. On the left, a sidebar menu includes 'Dashboard', 'Applications' (with a sub-menu icon), and 'Account'. The main content area features a 'WHAT'S NEW?' section and a 'RECENT APPLICATIONS' table.

**WHAT'S NEW?**

**RECENT APPLICATIONS**

ID	Name	Status
236082	test ↗	Running

At the bottom, a footer note reads: © 2017 RStudio Inc. | All Rights Reserved | Terms Of Use

At the very bottom of the screen, a Windows taskbar displays various pinned icons and the system tray.

## url 클릭하여 결과 확인

The screenshot shows the shinyapps.io application management interface. The URL in the browser is <https://www.shinyapps.io/admin/#/application/236082>. The interface includes a sidebar with 'Dashboard', 'Applications' (selected), and 'Account' options. The main content area displays the 'APPLICATION 236082 - TEST' page with tabs for Overview, Metrics, URLs, Settings, Users, Logs, Restart, Archive, and Delete. The Overview section shows details for the application with ID 236082, name 'test', URL <https://rcoholic.shinyapps.io/test/>, status 'Running', size 'large', and deployment history from Nov 16, 2017. The Instances section shows one instance with ID 1101069. The Application Usage section indicates 'No Data'.

shinyapps.io

APPLICATION 236082 - TEST

Overview Metrics URLs Settings Users Logs Restart Archive Delete

OVERVIEW

<b>Id</b>	236082
<b>Name</b>	test
<b>URL</b>	<a href="https://rcoholic.shinyapps.io/test/">https://rcoholic.shinyapps.io/test/</a>
<b>Status</b>	Running
<b>Size</b>	large
<b>Deployed</b>	Nov 16, 2017
<b>Updated</b>	Nov 16, 2017
<b>Created</b>	Nov 16, 2017
<b>Bundle</b>	<a href="#">Download</a>

INSTANCES

<b>Id:</b> 1101069
--------------------

APPLICATION USAGE

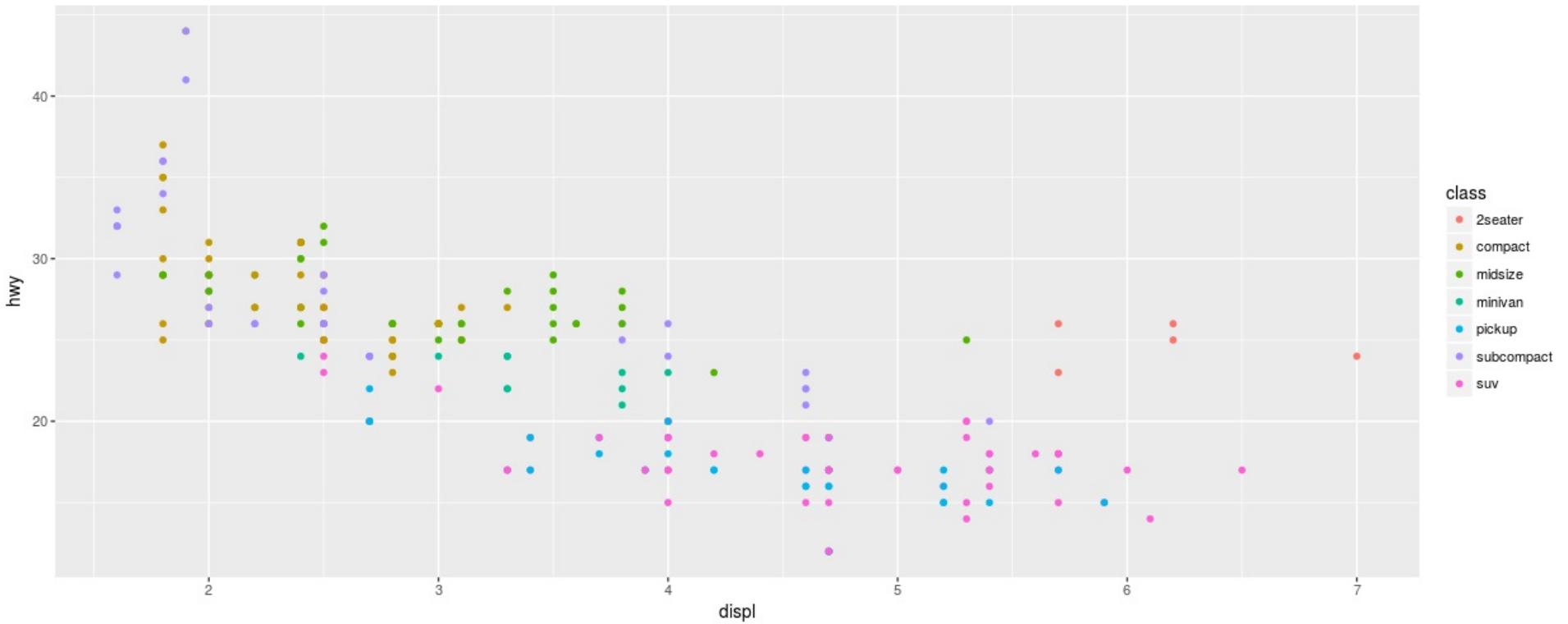
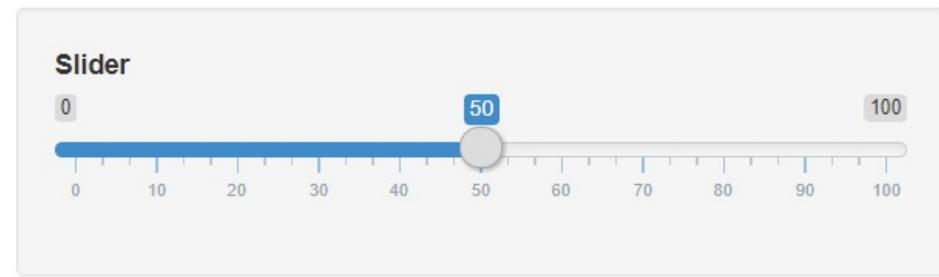
No Data

© 2017 RStudio Inc. | All Rights Reserved | Terms Of Use

# 온라인 배포 완료



test



# encoding 문제

이곳을 참고하세요. 이런 문제의 대부분은 중국어도 같이 가지고 있어서 중국쪽의 해결책이 많이 도움이 됩니다. 일본어 역시 마찬 가지입니다.

# shiny가 지원하는 js 패키지들의 활용

# 지도를 지원하는 leaflet

leaflet은 javascript 언어로 이루어진 지도에 대한 패키지입니다. shiny는 htmlwidget을 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

ggiraph는 ggplot의 geom계 함수들에게 뒤에 \_interactive를 붙여 활용함으로써 조금더 표가 사용자와 상호작용 할 수 있는 부분을 추가해줍니다. 효과로는 click, hover, zoom, data\_id 등이 있습니다.

# 상호작용 network 화면을 만드는 visNetwork

visNetwork는 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

shinydashboard는 대쉬보드를 shiny로 제작할 수 있게 대쉬보드 제작 프레임워크인 AdminLTE을 기반으로 작성했습니다. 유료 템플릿을 판매하는 기반을 가진 오픈소스인 만큼 프레임워크의 질이 product ready 수준이라 사용하기 매우 좋습니다.