

Shiny의 또 다른 활용

RStudio Addin 함수 및 패키지의 제작

문건웅

2018/3/21

패키지 설치

이번 강의를 위해 다음 패키지의 설치가 필요하다.

- `install.packages(c("miniUI","dplyrAssist","editData","ggplotAssist","nycflights13"))`
- `install.packages("rstudioapi", type = "source")`
- `devtools::install_github("rstudio/addinexamples", type = "source")`

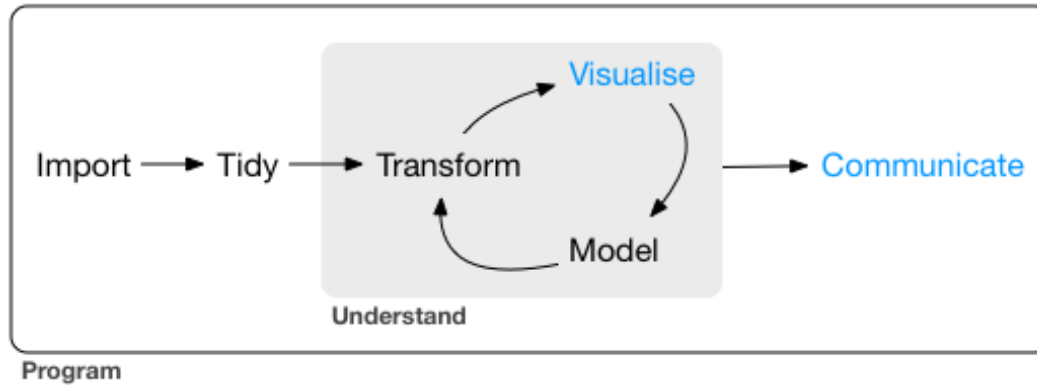
예제 소스 파일

이번 강의에 사용되는 예제 소스 파일들은 다음 `github`에서 다운로드 받을수 있다.

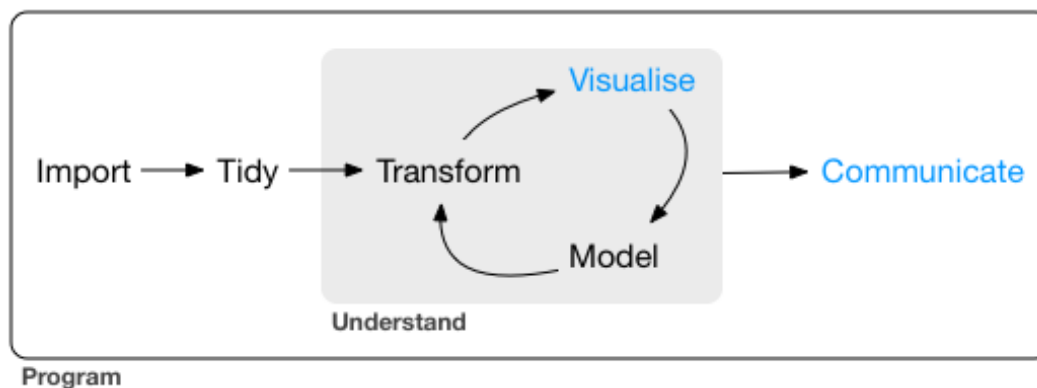
<https://github.com/cardiomoon/shinyLecture2>

- `ggbrush.R`
- `pick_points.R`
- `tower_of_hanoi.R`

Typical Data Science Project



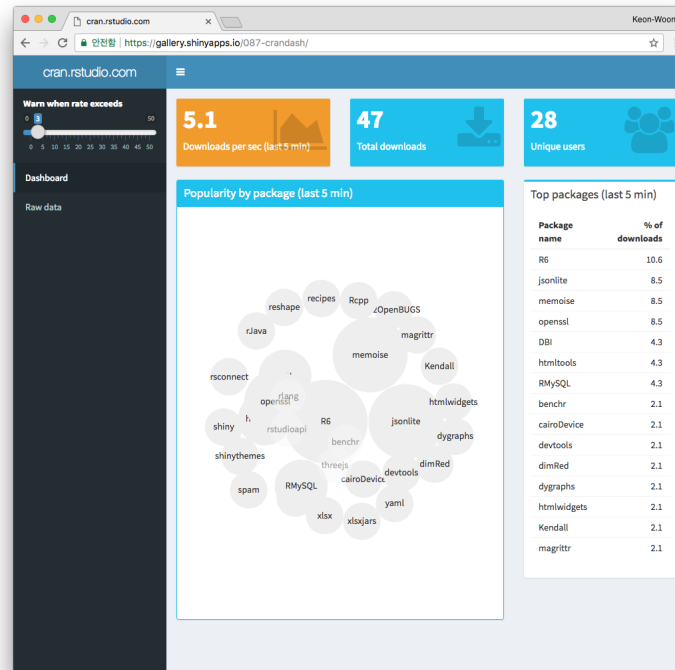
Typical Data Science Project



Shiny

A medium for interactively communicating *ideas* and *results*

A shiny app



<https://gallery.shinyapps.io/087-crandash/>

Shiny Applications

- 나의 데이터, 분석 결과, 모형 등을
- 다른 사람들 특히 R을 잘 모르는 사람들과 **communication**하기 위해 제작
- 주로 분석의 결과를 나타냄
- Server(Shiny server, ShinyApps.io)에 위치하여 웹 **browser**를 통해 접근

Shiny Applications

- 나의 데이터, 분석 결과, 모형 등을
- 다른 사람들 특히 R을 잘 모르는 사람들과 **communication**하기 위해 제작
- 주로 분석의 결과를 나타냄
- Server(Shiny server, ShinyApps.io)에 위치하여 웹 **browser**를 통해 접근

Shiny Gadgets

- R programming 을 도와주기 위한 **interactive tool**
- R을 모르는 **end user**가 아닌 R 을 사용하는 **user**가 사용하도록 제작
- 분석의 결과가 아니라 분석의 과정 중에 사용
- R console이나 R code로 불러오거나 RStudio 내에서 사용
- RStudio Addins 중 하나로 등록하여 GUI 환경에서 사용할 수 있다.

Potential Use of Shiny Gadgets

데이터 로딩, 데이터 정제, 데이터 조작, 데이터 시각화 등 거의 모든 업무에 대해 shiny gadget을 만들 수 있다.

- 복잡한 API를 통해 데이터를 다운로드하는 과정을 easy-to-use UI로 만듦
- 정규표현식을 이용한 검색/치환 등을 미리보기
- Visual Selection tools for subsetting or outlier exclusion

Shiny Gadgets by 문건웅

Install from CRAN

```
install.packages("dplyrAssist")  
install.packages("editData")  
install.packages("ggplotAssist")
```

Install from github

```
require(devtools)  
install_github("cardiomoon/dplyrAssist")  
install_github("cardiomoon/editData")  
install_github("cardiomoon/ggplotAssist")
```

A Shiny Gadget - ggplotAssist

```
devtools::install_github("cardiomoon/ggplotAssist")
```

The screenshot shows the 'Learn ggplot2' interface of the ggplotAssist Shiny application. The interface is organized into several panels:

- Data / Preprocessing:** Includes a 'Preprocessing' section with a text area for R codes and a 'show str' toggle. Below is the 'Enter data name' section with a text input containing 'msleep'.
- Aesthetics:** A list of variables (x, y, z, group, colour, fill, label, alpha, linetype, size, shape, xmin) with 'x' selected.
- mapping:** A section with a 'mapping' toggle and a list of variables (name, genus, vore, order, conservation, sleep_total, sleep_rem, sleep_cycle, awake, brainwt) with 'x' selected.
- R code for ggplot:** A text area containing 'ggplot(msleep)' and a 'reset' button.
- Add Layer(s):** A section with a 'Select' dropdown showing 'geom', 'stat', and 'coord'. Below it is a list of geom functions (geom_abline, geom_area, geom_bar, geom_bin2d, geom_blank, geom_boxplot, geom_cul, geom_contour, geom_count, geom_crossbar, geom_curve, geom_density2d, geom_density2d, geom_dotsplot, geom_errorbar, geom_errorbarh, geom_fgeom, geom_hex, geom_histogram, geom_hline, geom_jitter) with 'geom' selected.
- data:** A text input field.
- setting:** A section with a 'setting' toggle and a list of variables (x, y, z, group, colour, fill, label, alpha, linetype, size) with 'x' selected.
- Layer under construction:** A section with a text input field and 'Add Layer' and 'reset' buttons.
- Added layers:** A section with a list of layers and a 'Delete Layer' button.

At the bottom, there is a 'R code for plot' section with a text area containing '1 ggplot(msleep)' and buttons for 'Preview' and 'Layer by layer'.

Writing Shiny Gadgets ... regular function

```
library(shiny)
library(miniUI)

myGadgetFunc <- function(inputValue1, inputValue2) {

  ui <- miniPage(
    gadgetTitleBar("My Gadget"),
    miniContentPanel(
      # Define layout, inputs, outputs
    )
  )

  server <- function(input, output, session) {
    # Define reactive expressions, outputs, etc.

    # When the Done button is clicked, return a value
    observeEvent(input$done, {
      returnValue <- ...
      stopApp(returnValue)
    })
  }

  runGadget(ui, server)
```

UI

```
ui <- miniPage(  
  gadgetTitleBar("My Gadget"),  
  miniContentPanel(  
    # Define layout, inputs, outputs  
  )  
)
```

UI

```
ui <- miniPage(  
  gadgetTitleBar("My Gadget"),  
  miniContentPanel(  
    # Define layout, inputs, outputs  
  )  
)
```

- Shiny Apps: fluidPage, sidebarLayout
- Shiny Gadgets: miniPage, miniContentPanel
- Shiny Gadget은 default로 RStudio의 Viewer pane에서 수행된다.
- Web browser보다 훨씬 적은 공간이므로 공간을 절약할 수 있는 miniUI 패키지를 사용

Server

```
server <- function(input, output, session) {  
  # Define reactive expressions, outputs, etc.  
  
  # When the Done button is clicked, return a value  
  observeEvent(input$done, {  
    returnValue <- ...  
    stopApp(returnValue)  
  })  
}  
  
runGadget(ui, server)
```

Server

```
server <- function(input, output, session) {  
  # Define reactive expressions, outputs, etc.  
  
  # When the Done button is clicked, return a value  
  observeEvent(input$done, {  
    returnValue <- ...  
    stopApp(returnValue)  
  })  
}  
  
runGadget(ui, server)
```

- UI의 `gadgetTitleBar("My Gadget")`에서 Cancel과 Done 버튼을 제공한다.
- Done 버튼은 위와 같이 다룬다.
- Cancel 버튼을 누르면 `runGadget()` 함수에서 자동으로 "User Cancel" 에러와 함께 앱을 종료해준다.

Cancel 버튼 이벤트 다루기

```
server <- function(input, output, session) {  
  observeEvent(input$cancel, {  
    stopApp(NULL)  
  })  
}  
  
runGadget(ui, server, stopOnCancel = FALSE)
```


The First Shiny Gadget - ggbrush.R

```
library(shiny)
library(miniUI)
library(ggplot2)

ggbrush <- function(data, xvar, yvar) {

  ui <- miniPage(
    gadgetTitleBar("Drag to select points"),
    miniContentPanel(
      plotOutput("plot", height = "100%", brush = "brush")
    )
  )

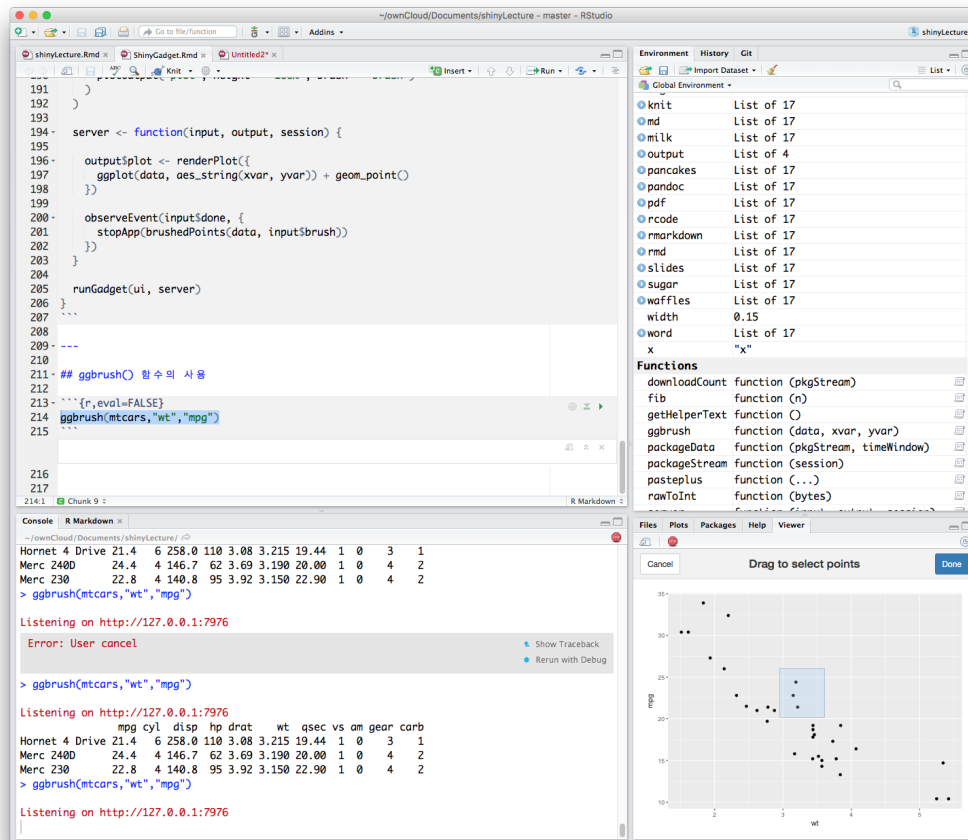
  server <- function(input, output, session) {

    output$plot <- renderPlot({
      ggplot(data, aes_string(xvar, yvar)) + geom_point()
    })

    observeEvent(input$done, {
      stopApp(brushedPoints(data, input$brush))
    })
  }
}
```

ggbrush() 함수의 사용

```
ggbrush(mtcars, "wt", "mpg")
```



The 2nd Example - pick_points.R

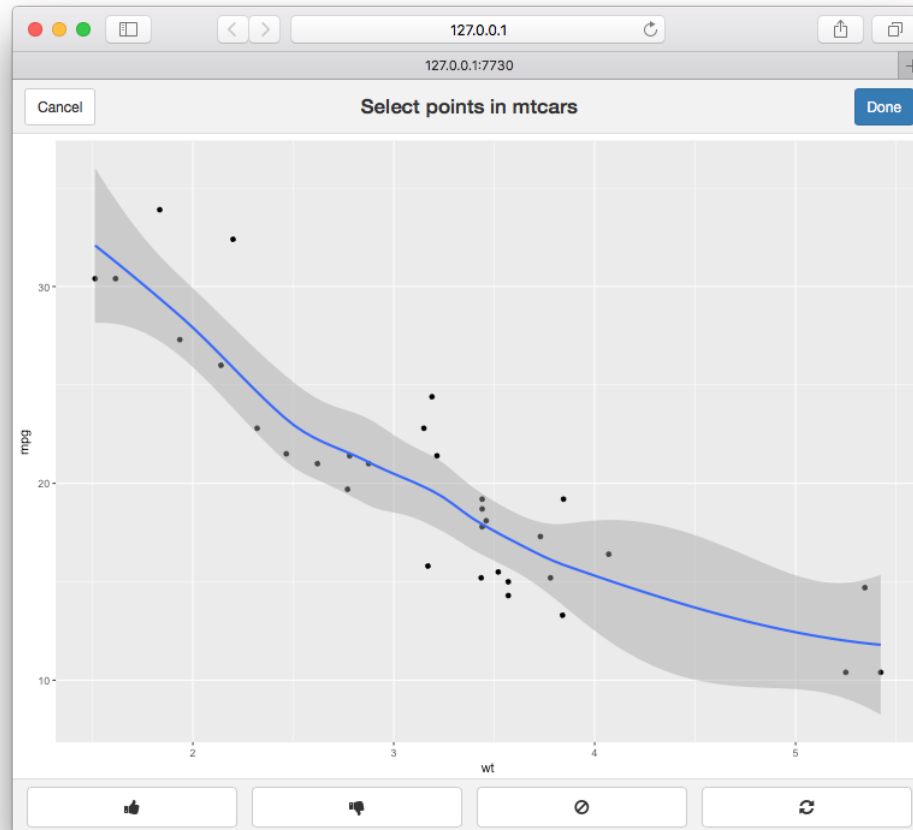
The screenshot displays the RStudio interface with the `pick_points.R` script open in the editor. The script defines a `pick_points` function that takes `data`, `x`, and `y` as arguments. It uses `miniPage` to create a user interface with a title bar, a plot output, and several action buttons. The `server` function handles the input, updates the plot, and returns the selected points.

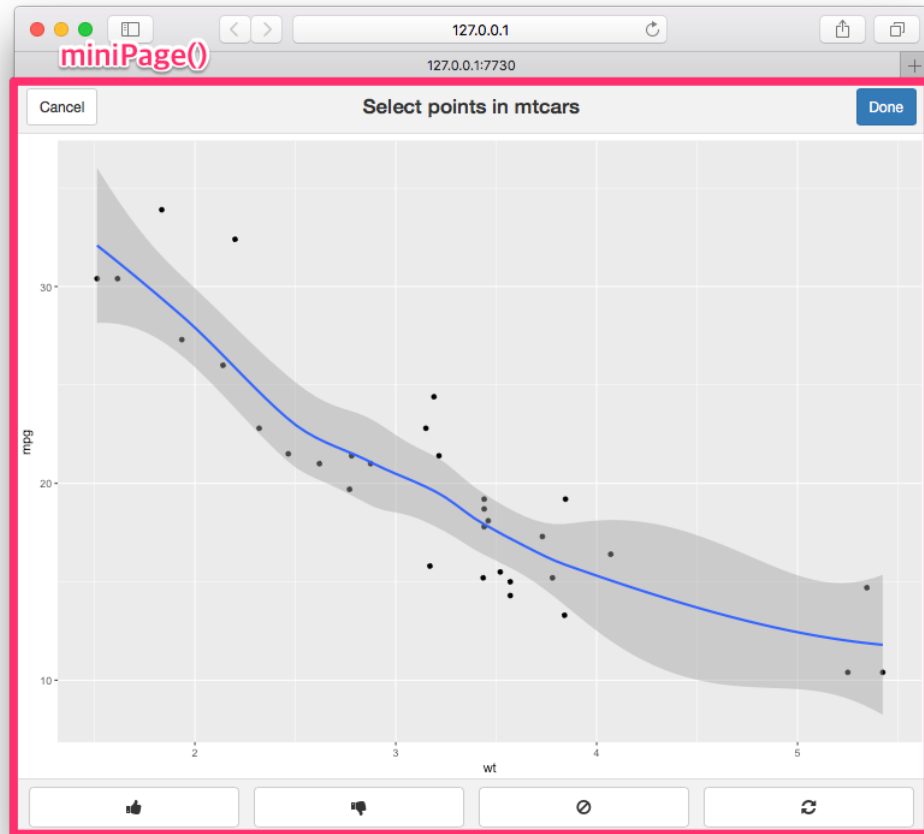
```
1 library(shiny)
2 library(miniUI)
3 library(ggplot2)
4
5 pick_points <- function(data, x, y) {
6
7   ui <- miniPage(
8
9     gadgetTitleBar(paste("Select points in", deparse(substitute(data)))),
10    miniContentPanel(padding = 0,
11    plotOutput("plot1", height = "100%", brush = "brush")
12    ),
13    miniButtonBlock(
14      actionButton("add", "", icon = icon("thumbs-up")),
15      actionButton("sub", "", icon = icon("thumbs-down")),
16      actionButton("none", "", icon = icon("ban")),
17      actionButton("all", "", icon = icon("refresh"))
18    )
19  )
20
21  server <- function(input, output, session) {
22    vals <- reactiveValues(keep = rep(TRUE, nrow(data)))
23
24    output$plot1 <- renderPlot({
25      exclude <- data[!vals$keep, , drop = FALSE]
26      include <- data[vals$keep, , drop = FALSE]
27
28      ggplot(data, aes_string(x, y)) +
29        geom_point() +
30        geom_point(data = exclude, color = "grey80") +
31      
```

The console shows the execution of the script, including the loading of libraries and the definition of the `pick_points` function. The output window displays a plot titled "Select points in mtcars" with a blue loess smoothing line and a grey shaded confidence interval. The plot shows the relationship between `wt` (weight) and `mpg` (miles per gallon) for the `mtcars` dataset.

UI

```
ui <- miniPage(  
  gadgetTitleBar(paste("Select points in", deparse(substitute(data))),  
  miniContentPanel(padding = 0,  
    plotOutput("plot1", height = "100%", brush = "brush")  
  ),  
  miniButtonBlock(  
    actionButton("add", "", icon = icon("thumbs-up")),  
    actionButton("sub", "", icon = icon("thumbs-down")),  
    actionButton("none", "", icon = icon("ban")),  
    actionButton("all", "", icon = icon("refresh"))  
  )  
)
```





reactiveValues() 를 사용하여 현재 상태 저장

```
vals <- reactiveValues(keep = rep(TRUE, nrow(data)))

output$plot1 <- renderPlot({
  exclude <- data[!vals$keep, , drop = FALSE]
  include <- data[vals$keep, , drop = FALSE]

  ggplot(data, aes_string(x, y)) +
    geom_point() +
    geom_point(data = exclude, color = "grey80") +
    stat_smooth(data = include)
})
```


observeEvent() 로 현재상태를 update

```
selected <- reactive({  
  brushedPoints(data, input$brush, allRows = TRUE)$selected  
})  
  
observeEvent(input$add, vals$keep <- vals$keep | selected())  
observeEvent(input$sub, vals$keep <- vals$keep & !selected())  
observeEvent(input$all, vals$keep <- rep(TRUE, nrow(data)))  
observeEvent(input$none, vals$keep <- rep(FALSE, nrow(data)))
```

반환값

```
observeEvent(input$done, {  
  stopApp(data[vals$keep, , drop = FALSE])  
})
```

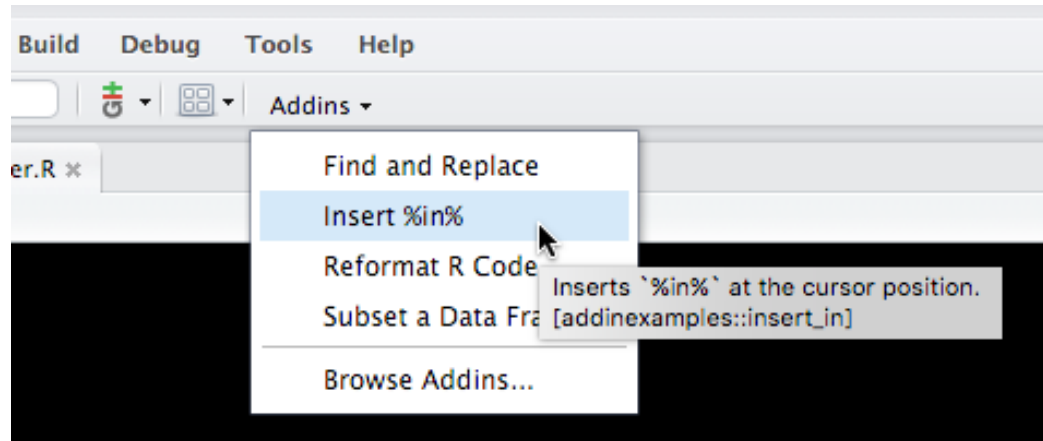
RStudio Addins

- RStudio ($\geq 0.99.878$)

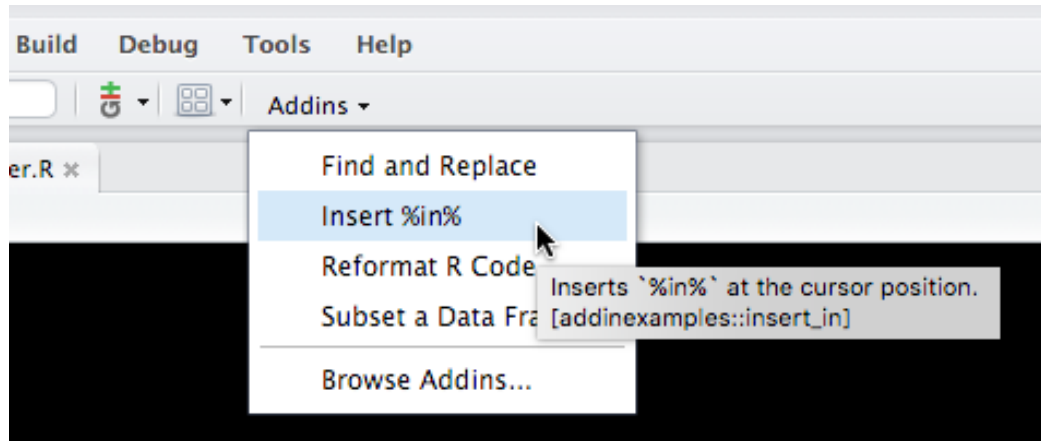
Install Addin Examples

```
devtools::install_github("rstudio/addinexamples", type = "source")
```

Running Addins



Running Addins



```
insertInAddin <- function() {  
  rstudioapi::insertText(" %in% ")  
}
```

RStudio API

- RStudio의 R code와 interact 할 수 있게 해준다.

```
install.packages("rstudioapi", type = "source")
```

Functions	Details
getActiveDocumentContext()	Returns information about the currently active RStudio document.
insertText(location, text, id = NULL)	Insert text at a specific location within a document.
setDocumentContext(text, id = NULL)	Set the contents of a document open in RStudio.

getActiveDocumentContext() returns list

- id — The unique document id.
- path — The path to the document on disk.
- contents — The contents of the document.
- selection — A list of selections.

Registering Addins

Registration file : located at **inst/rstudio/addins.dcf**

- Name: The name of the addin.
- Description: A description of the addin.
- Binding: The R function this addin is associated with.
- Interactive: Whether this addin is interactive (e.g. runs a Shiny application).

Registering Addins

Registration file : located at **inst/rstudio/addins.dcf**

- Name: The name of the addin.
- Description: A description of the addin.
- Binding: The R function this addin is associated with.
- Interactive: Whether this addin is interactive (e.g. runs a Shiny application).

For Example : insertInAddin

Name: Insert %**in**%

Description: Inserts ``%in%`` at the cursor position.

Binding: `insertInAddin`

Interactive: `false`

RStudio Addin 만들기

1. R 패키지 작성
2. R 함수 작성
3. **inst/rstudio/addins.dcf** 파일 작성
4. 패키지 설치

RStudio Addin 만들기

1. R 패키지 작성
2. R 함수 작성
3. **inst/rstudio/addins.dcf** 파일 작성
4. 패키지 설치

패키지를 설치할 때 RStudio 에서 addin 을 자동으로 등록한다. 하나의 패키지에 여러 개의 addin이 있는 경우 addins.dcf 파일에 여러 개의 addin을 등록할 수 있다. 이때 각 addin 사이에 빈줄을 넣는다.

Gadget Viewer

- `paneViewer()`: RStudio viewer pane 에 Gadget을 시작
- `dialogViewer()`: modal Dialog 로 Gadget을 시작
- `browserViewer()`: 별도의 `browser()`창을 띄워 gadget 시작

Shiny Gadget 설치

Imports:

```
shiny (>= 0.13),  
miniUI (>= 0.1.1),  
rstudioapi (>= 0.5)
```

editData 설치

- CRAN: <https://cran.r-project.org/web/packages/editData/index.html>

```
install.packages("editData")
```

- github: <https://github.com/cardiomoon/editData>

```
devtools::install_github("cardiomoon/editData")
```

editData 실행

```
library(editData)
```

```
result=editData("mtcars")
```