

## 2. R 기본 문법

# 강의 목표

- R 프로그램의 기본 특징을 이해한다.
- R 로 할 수 있는 간단한 프로그램을 경험해 본다.
- R에서 object와 variable 개념을 이해한다.
- R 기본 자료형을 살펴본다.
- Vector에 대해 이해한다

# R 기본 특징

# R의 특징

## ▶ 인터프리터 언어

- 대화형 프로그래밍 언어로 프로그램 코드를 한 줄이 바뀔 때 마다 즉시 실행됨.
- R을 실행하면, input을 기다리는 console 창이 뜨게 됨. ‘>’는 명령프롬프트로 원하는 명령을 입력한 후 엔터를 치면 한줄 씩 실행 됨

```
> 1 + 2  
[1] 3
```

## ▶ 대소문자를 구분함

- R은 대소문자를 구분하므로 소스코드 작성 시 주의해야 함
- KoNLP 또는 stringsAsFactors=FALSE 등 다양한 함수나 패키지에 대소문자가 섞여있는 경우가 매우 많아서 자동완성기능을 활용하면 실수를 줄일 수 있음

# R의 특징

## ➤ 방대한 패키지

- 특정목적이나 연구분야에 사용되는 다양한 패키지들이 있음
- `install.packages("패키지명")` 입력시 설치가 됨

## ➤ 다양한 시각화 기능

- 데이터 분석 뿐만 아니라 데이터 및 분석결과를 시각화 하는 기능이 뛰어남
- `ggplot2`, `googleVis` 등과 같은 시각화 패키지가 대표적임

## ➤ 빅데이터 분석의 도구

- 현재 데이터분석 도구로 가장 많이 쓰고 있는 것은 R과 python이다.

**R로 뭘할까? – R을 계산기처럼!**

# 사칙연산

예를 들어

$$2.5 - \frac{3}{6} + \pi \times 2$$

를 계산해야 한다면 R로는 어떻게 할 수 있을까?

→ R에서 pi는 원주율  $\pi$  !!!

R의 프롬프트 (>) 다음에

`2.5 - 3 / 6 + pi * 2`

이렇게 쓰고 엔터를 친다. 그러면..

```
> 2.5 - 3 / 6 + pi * 2
[1] 8.283185
```

# 거듭제곱과 제곱근

▶ 그럼  $3^2 + \sqrt{16}$  을 계산해야 한다면 R로 어떻게 할 수 있을까?

- 거듭제곱과 제곱근sqrt는 영어 square root의 약자임.
- $3^2$ 은 9,  $\text{sqrt}(16)$ 은 4, 따라서 13임.
- $3^2$ 은  $3**2$ 로 쓸 수도 있음.
- $\text{sqrt}(16)$ 은  $16^{0.5}$ 로 쓸 수도 있음.

R의 프롬프트 (>) 다음에

$3^2 + \text{sqrt}(16)$

이렇게 쓰고 엔터를 친다. 그러면..

```
> 3^2 + sqrt(16)
[1] 13
```



# 로그 함수

➤  $\log e + \log_{10} 100 - \log_2 8$  을 계산해야 한다면 R로는 어떻게 할 수 있을까?

- $\log$ 는 자연 로그로, 밑이  $e$ 인 로그임.
- $\log_{10}$ 은 밑이 10인 로그,  $\log_2$ 는 밑이 2인 로그임.
- 그런데  $e$ 는  $\pi$ 와는 달리 따로 예약되어 있지 않아서 부득이  $\exp(1)$ 을 써야함.
- $\log(\exp(1)) = 1$ ,  $\log_{10}(100) = 2$ ,  $\log_2(8) = 3$   
그래서 답이 0이 됨.

R의 프롬프트 (>) 다음에  
 $\log(\exp(1)) + \log_{10}(100) - \log_2(8)$   
 이렇게 쓰고 엔터를 친다. 그러면..

```
> log(exp(1)) + log10(100) - log2(8)
[1] 0
```

# 정수 관련 편의 함수

- ▶ 몇몇 정수 관련 편의 함수는 알아 두면 좋을 수 있음.

<code>round(..)</code>	반올림
<code>ceiling(..)</code>	올림
<code>floor(..)</code>	버림
<code>%/%</code>	몫
<code>%%</code>	나머지

```

> round(2.4) ## 2.4의 반올림은?
[1] 2
> round(2.5) ## 2.5의 반올림은?
[1] 2
> round(2.51) ## 2.51의 반올림은?
[1] 3
> floor(2.4) ## 2.4의 버림은?
[1] 2
> floor(2.51) ## 2.51의 버림은?
[1] 2
> ceiling(2.4) ## 2.4의 올림은?
[1] 3
> ceiling(2.5) ## 2.5의 올림은?
[1] 3
> 5 %/% 3 ## 5를 3으로 나눈 몫은?
[1] 1
> 6 %/% 3 ## 6을 3으로 나눈 몫은?
[1] 2
> 5 %% 3 ## 5를 3으로 나눈 나머지는?
[1] 2
> 6 %% 3 ## 6을 3으로 나눈 나머지는?
[1] 0

```

# R 스타일 맛보기

# 데이터 소개

- ▶ 참고문헌: Dobson, A.J. (1983) An introduction to statistical modeling. (Table 7.4)
  - 농작물에 어떤 처리를 해서 과연 생산량이 늘었는지를 확인하고 싶어 두가지 처리를 한 경우와 아무런 처리도 하지 않은 경우에 대해 작물의 무게를 측정하였음.
  - 아무 처리도 하지 않은 경우: 4.17, 5.58, 5.18, 6.11, 4.5, 4.61, 5.17, 4.53, 5.33, 5.14
  - 첫번째 처리를 한 경우: 4.81, 4.17, 4.41, 3.59, 5.87, 3.83, 6.03, 4.89, 4.32, 4.69
  - 두번째 처리를 한 경우: 6.31, 5.12, 5.54, 5.5, 5.37, 5.29, 4.92, 6.15, 5.8, 5.26
  - 뭘 어떻게 하면 될까? 단계별로 해결해보자!

# 과제 1. 데이터 요약

- 우선은 무처리와 1, 2번 처리에 대해 각각 대표값(평균값과 중간값)과 데이터 분포를 살펴본다.
- 그런데, 예를 들어 평균을 계산하는 식을 다음과 같이 쓰면 어떻게 될까? R을 쓸 이유가 없게 된다.

```
> (4.17 + 5.58 + 5.18 + 6.11 + 4.5 + 4.61 + 5.17 + 4.53 + 5.33 + 5.14) / 10
[1] 5.032
> (4.81 + 4.17 + 4.41 + 3.59 + 5.87 + 3.83 + 6.03 + 4.89 + 4.32 + 4.69) / 10
[1] 4.661
> (6.31 + 5.12 + 5.54 + 5.5 + 5.37 + 5.29 + 4.92 + 6.15 + 5.8 + 5.26) / 10
[1] 5.526
```

- R은 항상 데이터를 먼저 넣어주고 그 데이터는 그대로 둔 채 거기에 함수를 적용해서 원하는 값을 얻도록 해야 함
- 즉, 앞에서 언급한 데이터를 R에 먼저 넣어주고 평균은 `mean()`, 중간값은 `median()`, 표준편차는 `sd()` 등의 함수를 그 데이터에 적용해 값을 구해야 함.

# 과제 1. 데이터 요약

- ▶ 데이터를 입력할 때는 R 콘솔에 바로 입력할 수도 있고 파일에 데이터를 저장한 다음 그 파일을 읽어드릴 수도 있음.
- ▶ 콘솔에 바로 입력할 때는 `c()` 함수를 쓰고 변수에 저장함.
- ▶ 예를 들어 무처리, 처리 1, 2 등을 각각 변수 `ctrl`, `trt1`, `trt2` 등에 저장할 때 다음과 같이 함. (“<-“ 는 “<“와 “-“를 바로 이어 붙임.)
  - `ctrl <- c(4.17, 5.58, 5.18, 6.11, 4.5, 4.61, 5.17, 4.53, 5.33, 5.14)`
  - `trt1 <- c(4.81, 4.17, 4.41, 3.59, 5.87, 3.83, 6.03, 4.89, 4.32, 4.69)`
  - `trt2 <- c(6.31, 5.12, 5.54, 5.5, 5.37, 5.29, 4.92, 6.15, 5.8, 5.26)`
- ▶ 그런 다음 평균, 중간값, 표준편차 등은 다음과 같이 구함
  - `ctrl.avg <- mean(ctrl)`
  - `ctrl.med <- median(ctrl)`
  - `ctrl.sd <- sd(ctrl)`

# R Studio 화면

**Console**

```

During startup - Warning messages:
1: Setting LC_CTYPE failed, using "C"
2: Setting LC_COLLATE failed, using "C"
3: Setting LC_TIME failed, using "C"
4: Setting LC_MESSAGES failed, using "C"
5: Setting LC_PAPER failed, using "C"
>
>
>
> (4.17 + 5.58 + 5.18 + 6.11 + 4.5 + 4.61 + 5.17 + 4.53 + 5.33 + 5.14) / 10
[1] 5.032
> (4.81 + 4.17 + 4.41 + 3.59 + 5.87 + 3.83 + 6.03 + 4.89 + 4.32 + 4.69) / 10
[1] 4.661
> (6.31 + 5.12 + 5.54 + 5.5 + 5.37 + 5.29 + 4.92 + 6.15 + 5.8 + 5.26) / 10
[1] 5.526
>
>
> ctrl <- c(4.17, 5.58, 5.18, 6.11, 4.5, 4.61, 5.17, 4.53, 5.33, 5.14)
> trt1 <- c(4.81, 4.17, 4.41, 3.59, 5.87, 3.83, 6.03, 4.89, 4.32, 4.69)
> trt2 <- c(6.31, 5.12, 5.54, 5.5, 5.37, 5.29, 4.92, 6.15, 5.8, 5.26)
>
> ctrl.avg <- mean(ctrl)
> ctrl.med <- median(ctrl)
> ctrl.sd <- sd(ctrl)
>
> trt1.avg <- mean(trt1)
> trt1.med <- median(trt1)
> trt1.sd <- sd(trt1)
>
> trt2.avg <- mean(trt2)
> trt2.med <- median(trt2)
> trt2.sd <- sd(trt2)
>

```

**Environment**

Variable	Value
ctrl	num [1:10] 4.17 5.58 5.18 6.11 4.5 4.61 ...
ctrl.avg	5.032
ctrl.med	5.155
ctrl.sd	0.583091378392406
trt1	num [1:10] 4.81 4.17 4.41 3.59 5.87 3.83...
trt1.avg	4.661
trt1.med	4.55
trt1.sd	0.793675696434703
trt2	num [1:10] 6.31 5.12 5.54 5.5 5.37 5.29 ...
trt2.avg	5.526
trt2.med	5.435
trt2.sd	0.442573283322786

**Annotations:**

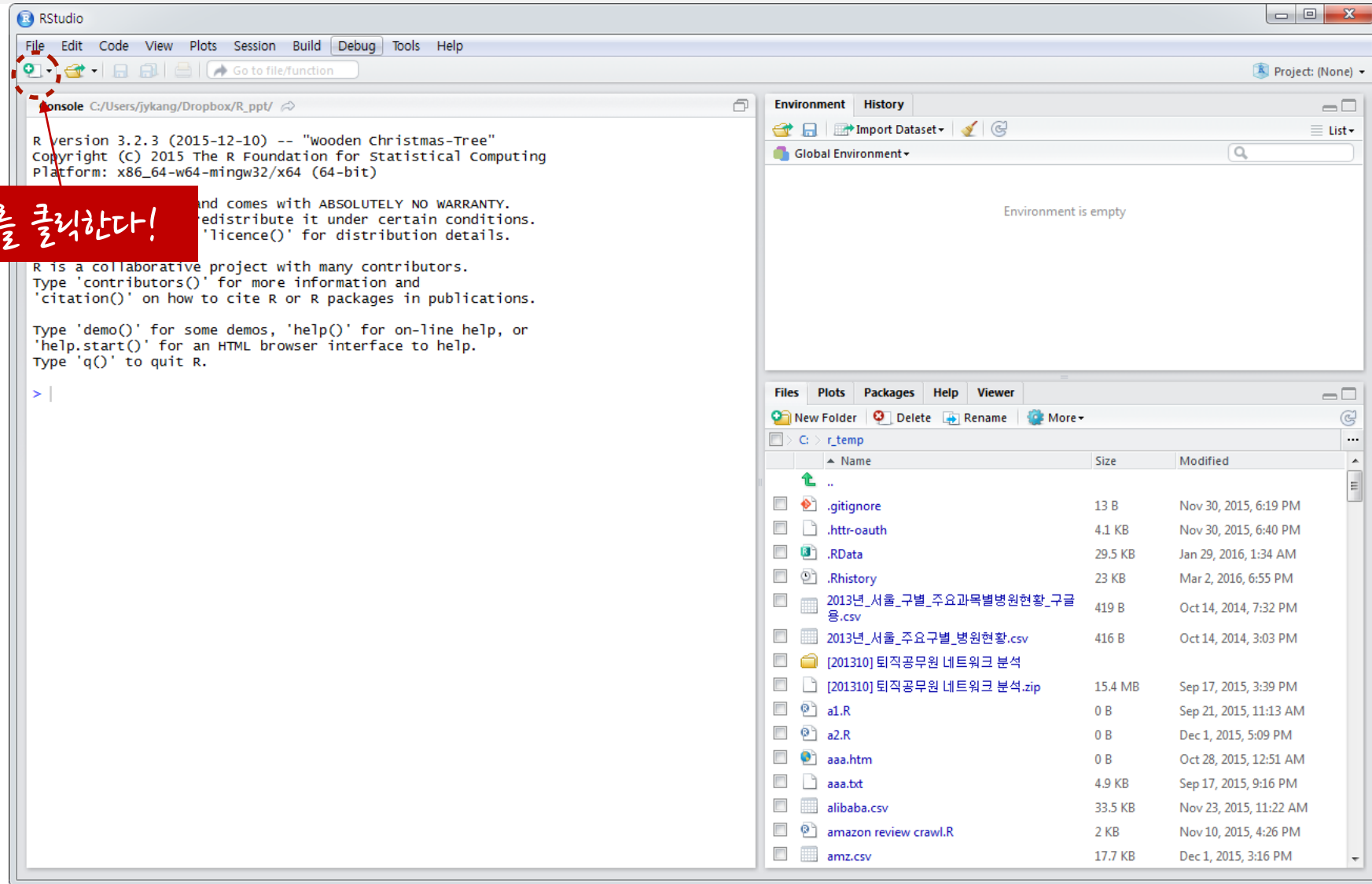
- 변수에 데이터를 저장하고 (Assigning data to variables)
- 원하는 값을 계산하면 (Calculate the values you want)
- 여기에 값이 나타난다! Rstudio라서 편리!! (The values appear here! Convenient because of Rstudio!!)

# R Studio의 바른 사용법?

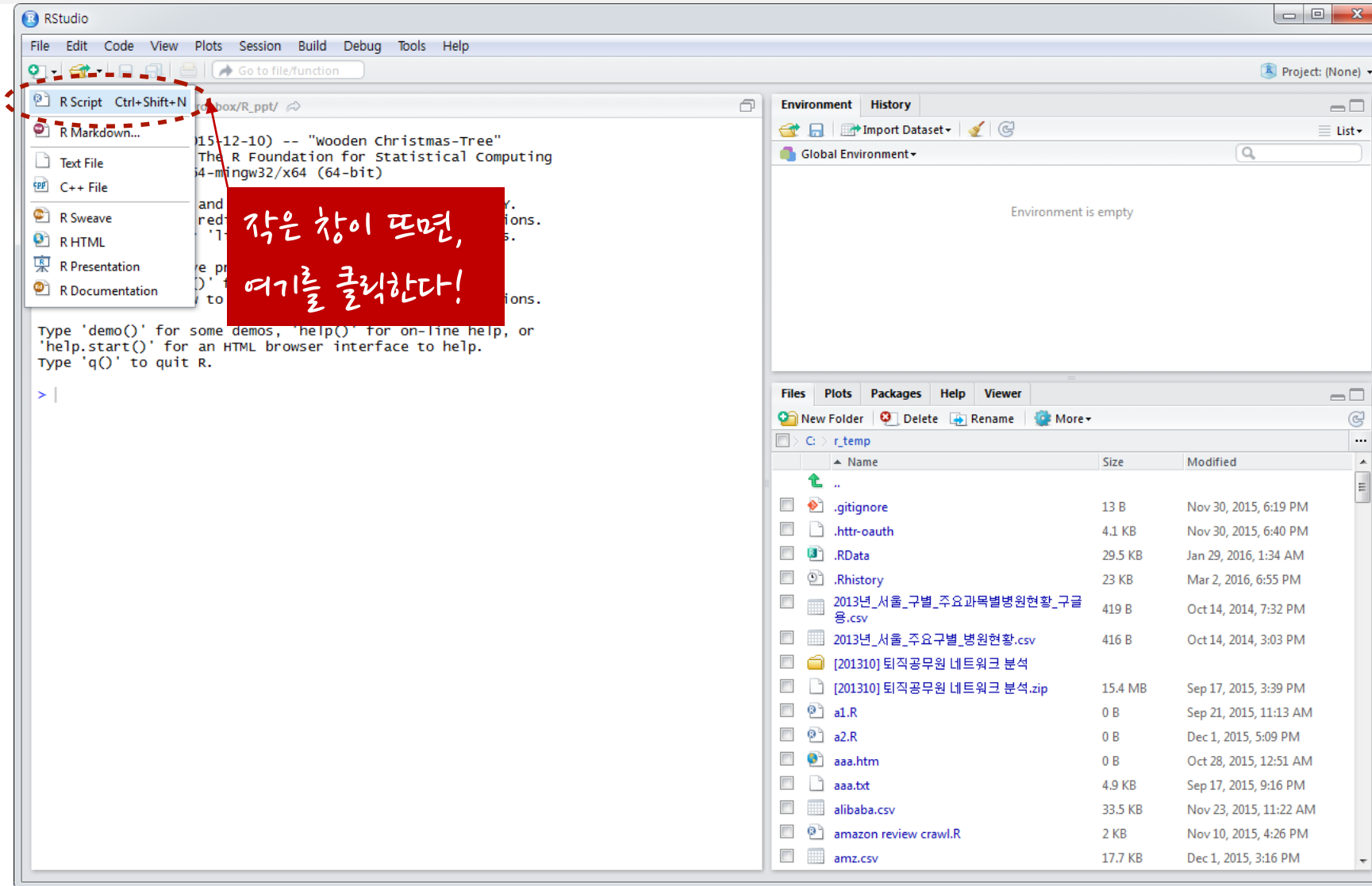
- ▶ 하지만 Rstudio를 쓰면서 R 콘솔 화면에서 작업하는 것이 바른 사용법일까?
- ▶ 항상 스크립트 파일을 열어서 거기다 하고 싶은 내용을 쓰고 그걸 R에 옮겨서 결과를 얻어야 한다! 스크립트의 내용을 R에 옮기는 일은 RStudio가 편리하게 해 주고, 사용자는 자신이 하고 싶은 일에만 집중할 수 있음.
- ▶ 또한 자신이 한 작업에 대한 온전한 기록이 남게 되어 작업 내용이나 오류를 확인할 수 있고 분석의 재현성을 높일 수 있음.



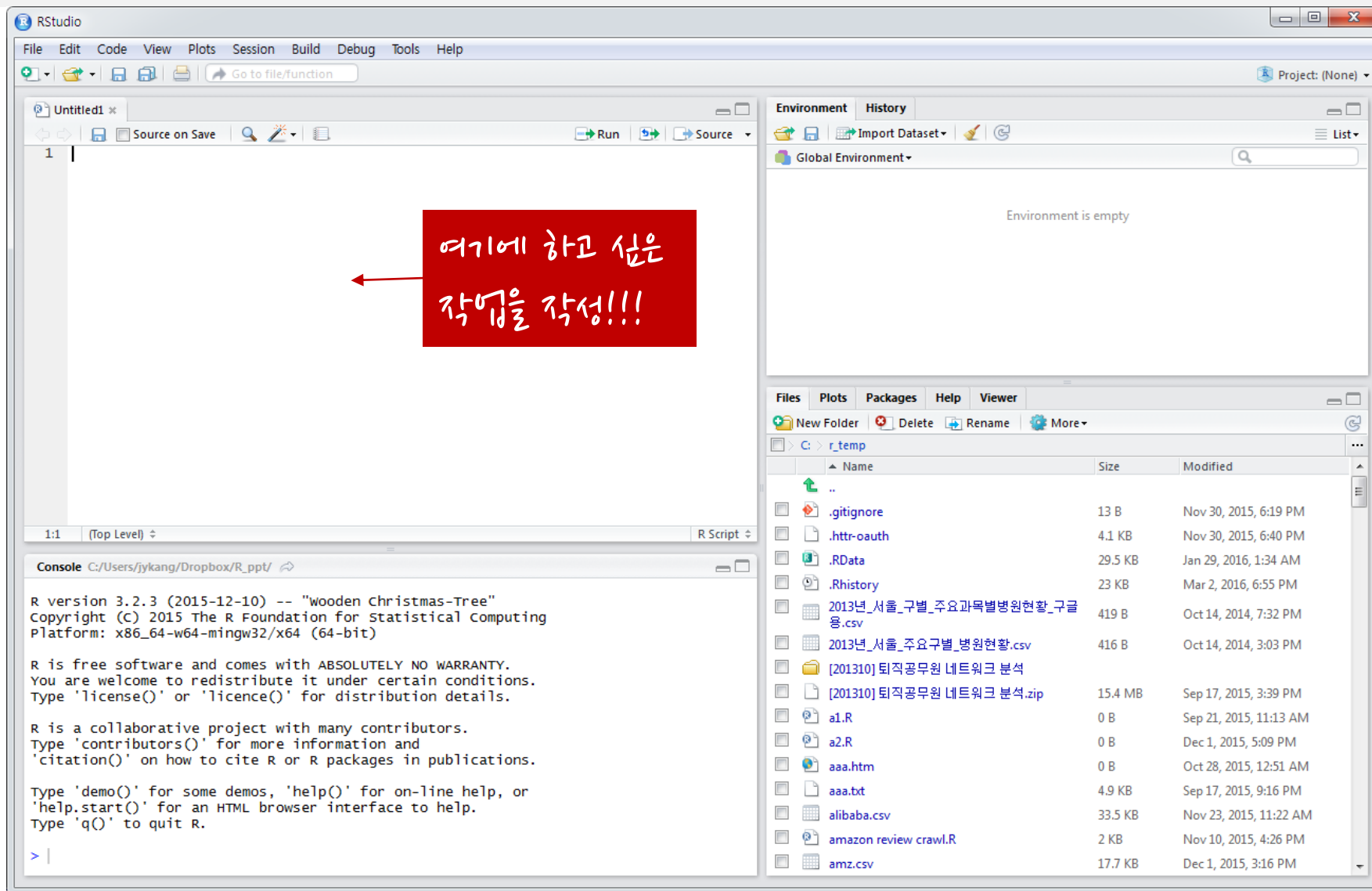
# R Studio 사용하기



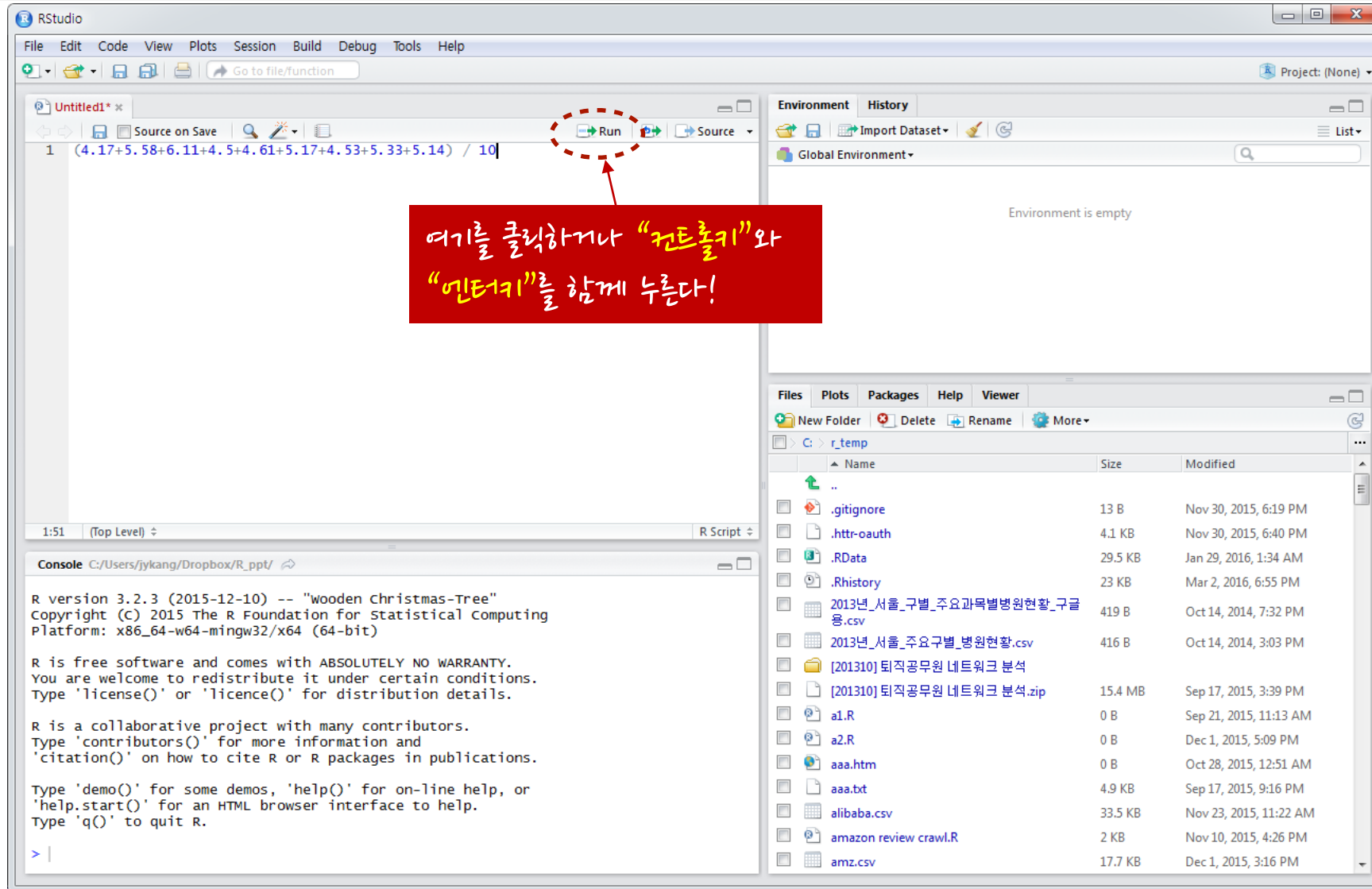
# R Studio 사용하기



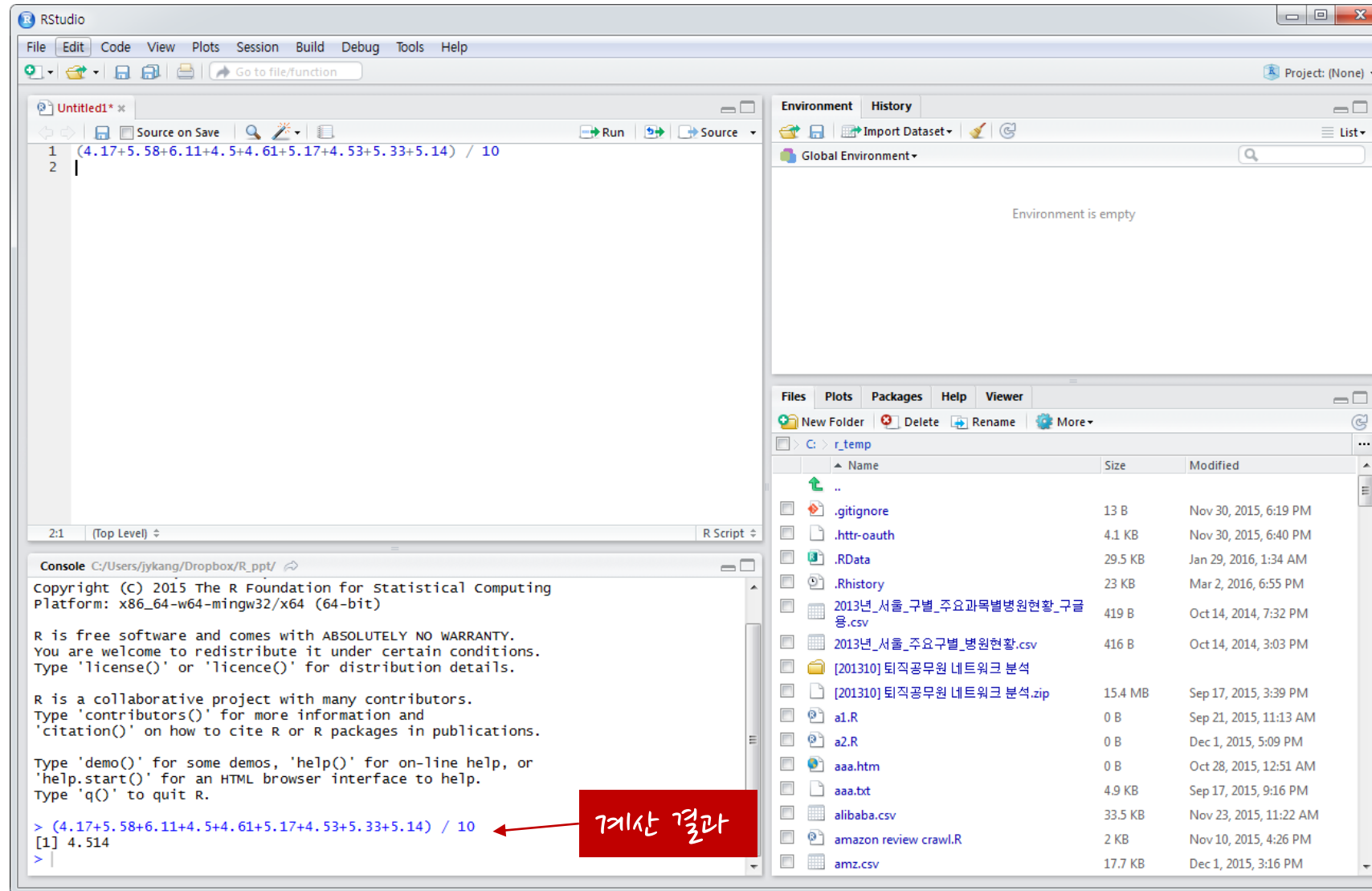
# R Studio 사용하기



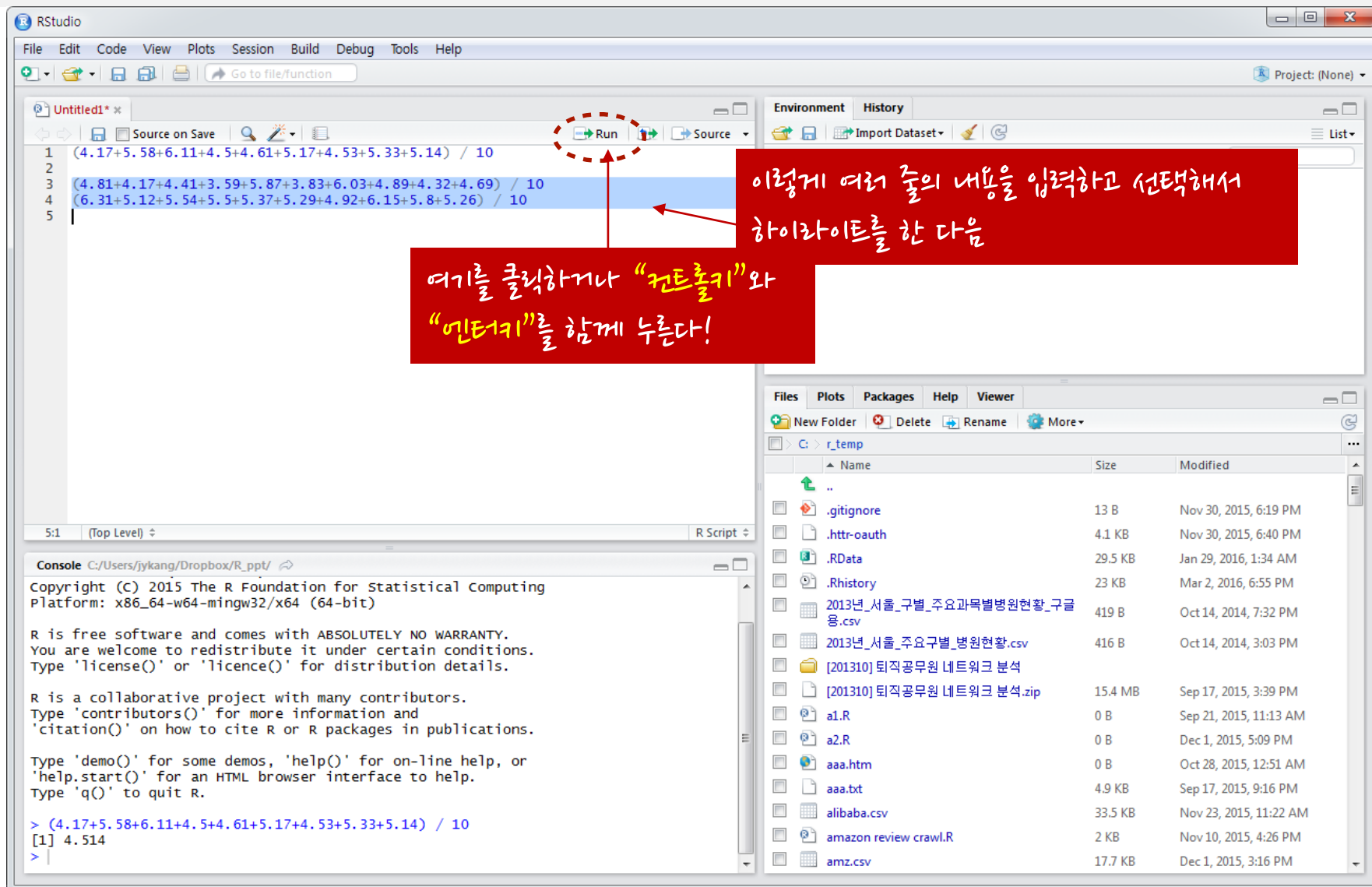
# R Studio 사용하기



# R Studio 사용하기



# R Studio 사용하기



# R Studio 사용하기

The screenshot displays the RStudio IDE interface. The main editor window shows a script file named 'Untitled1\*' with five lines of R code. The third line is highlighted in blue. The console window at the bottom shows the output of the first three lines of code. The environment window on the right is empty. The file explorer on the bottom right shows a directory named 'r\_temp' containing various files and folders.

**Script File (Untitled1\*):**

```

1 (4.17+5.58+6.11+4.5+4.61+5.17+4.53+5.33+5.14) / 10
2
3 (4.81+4.17+4.41+3.59+5.87+3.83+6.03+4.89+4.32+4.69) / 10
4 (6.31+5.12+5.54+5.5+5.37+5.29+4.92+6.15+5.8+5.26) / 10
5

```

**Console:**

```

You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> (4.17+5.58+6.11+4.5+4.61+5.17+4.53+5.33+5.14) / 10
[1] 4.514
> (4.81+4.17+4.41+3.59+5.87+3.83+6.03+4.89+4.32+4.69) / 10
[1] 4.661
> (6.31+5.12+5.54+5.5+5.37+5.29+4.92+6.15+5.8+5.26) / 10
[1] 5.526
>

```

**Environment:** Environment is empty

**Files:**

Name	Size	Modified
..		
.gitignore	13 B	Nov 30, 2015, 6:19 PM
.httr-oauth	4.1 KB	Nov 30, 2015, 6:40 PM
.RData	29.5 KB	Jan 29, 2016, 1:34 AM
.Rhistory	23 KB	Mar 2, 2016, 6:55 PM
2013년_서울_구별_주요과목별병원현황_구글용.csv	419 B	Oct 14, 2014, 7:32 PM
2013년_서울_주요구별_병원현황.csv	416 B	Oct 14, 2014, 3:03 PM
[201310] 퇴직공무원 네트워크 분석		
[201310] 퇴직공무원 네트워크 분석.zip	15.4 MB	Sep 17, 2015, 3:39 PM
a1.R	0 B	Sep 21, 2015, 11:13 AM
a2.R	0 B	Dec 1, 2015, 5:09 PM
aaa.htm	0 B	Oct 28, 2015, 12:51 AM
aaa.txt	4.9 KB	Sep 17, 2015, 9:16 PM
alibaba.csv	33.5 KB	Nov 23, 2015, 11:22 AM
amazon review crawl.R	2 KB	Nov 10, 2015, 4:26 PM
amz.csv	17.7 KB	Dec 1, 2015, 3:16 PM

계산 결과

# R 기초문법



# 변수란?

```
> for (i in 1:10) i^2
```

:  $i^2$ 가 계산되지만 아무것도 저장되지도 프린트되지도 않음

```
> for (i in 1:5) print(i^2)
```

```
[1] 1
```

```
[1] 4
```

```
[1] 9
```

```
[1] 16
```

```
[1] 25
```

:  $i^2$ 가 계산되어 프린트 되지만 값을 보는 목적  
외에 나중에 이 값을 다시 사용할 수 없음

```
> a <- numeric(10)
```

```
> for (i in 1:10) a[i] <- i^2
```

```
> print(a)
```

```
[1] 1 4 9 16 25 36 49 64 81 100
```

*a 라는 변수를 사용해 값을 나중에 사용할 수 있게 됨!!*

# 변수란?

- ▶ 프로그램은 작업 처리 과정에서 필요에 따라 데이터를 메모리에 저장하는데, 이 때 변수를 사용함
- ▶ 변수 : 값을 저장할 수 있는 메모리의 공간
- ▶ R 변수명 규칙
  - 대소문자 구분
  - 영어와 숫자 모두 쓸 수 있으나 시작은 반드시 문자 또는 .으로 시작해야 함.
  - 예약어는 사용할 수 없음

메모리 memory

x	5
y	5

```
x<-3
x<-4
x<-x+1
y<-5
```

# 변수란?

The screenshot shows the RStudio interface with the following components:

- Source Editor:** Contains R code for variable assignment:
 

```

1 10000+9000 #음료수와 과자
2
3 음료수 = 10000 # 오른쪽 값을 왼쪽에 넣어라
4 과자 = 9000
5
6 총액 = 음료수 + 과자
7
8 총액
9
      
```
- Environment Pane:** Shows the 'Global Environment' with the following values:
 

values	
과자	9000
음료수	10000
총액	19000
- Console:** Shows the execution output:
 

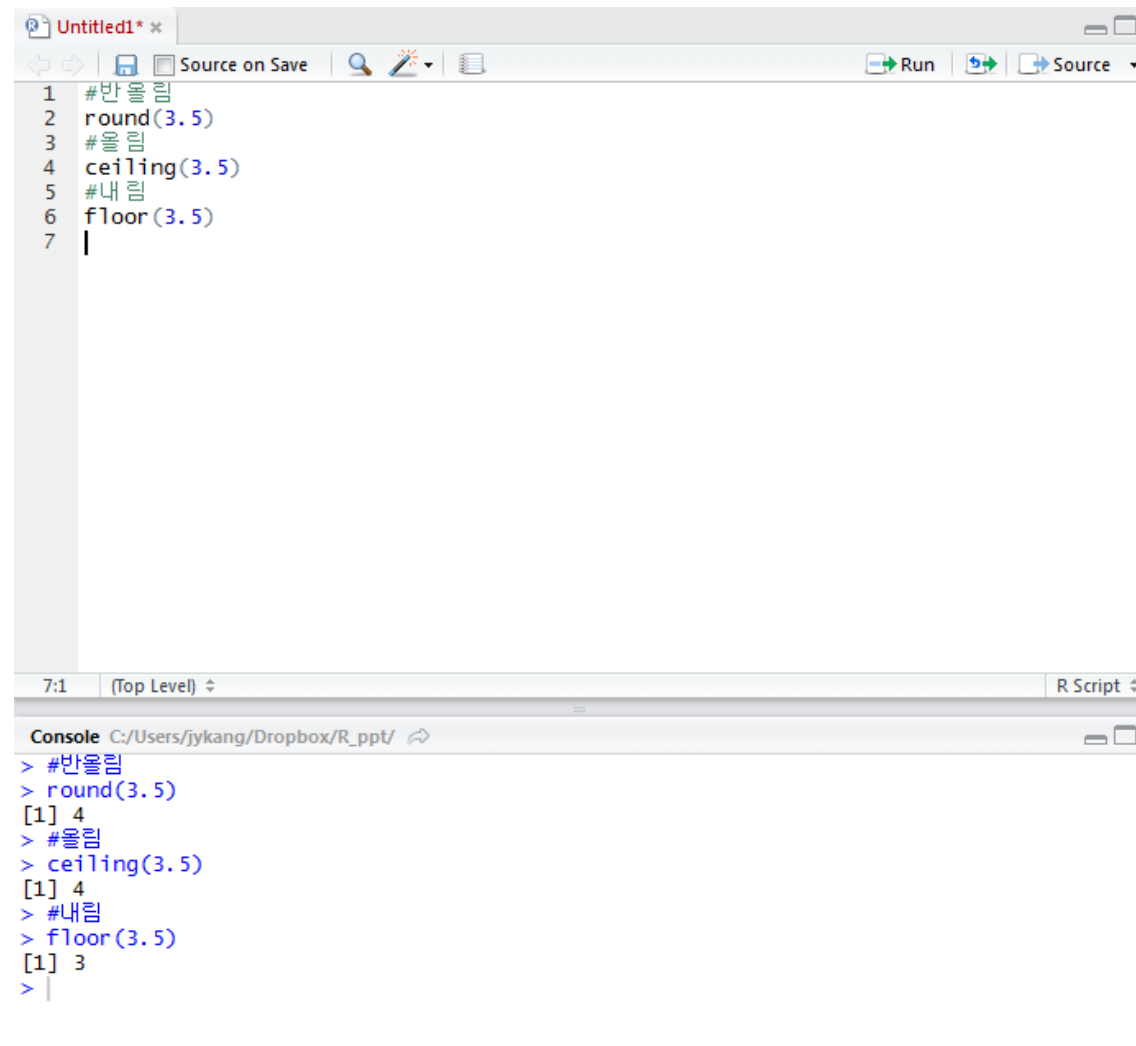
```

> 음료수 = 10000 # 오른쪽 값을 왼쪽에 넣어라
> 과자 = 9000
> 과자
[1] 9000
> 총액 = 음료수 + 과자
> 총액
Error: object '총' not found
> 총액
[1] 19000
      
```
- Files Pane:** Shows a directory listing for 'r\_temp' with files like .gitignore, .httr-oauth, .RData, .Rhistory, and various CSV/ZIP files.

# 함수란?

## ▶ 함수와 인수 (function 과 parameter)

- 함수 : 특정 기능을 수행하는 명령문을 함수라고 함
- 인수 : 함수의 기능을 세밀하게 조정하는 것이 인수
- 표현형태 : 함수 + (인수)
- $x \leftarrow c(1,2,3,4,5,6,7,8,9,10)$
- $y \leftarrow c(2,4,6,8,10,12,14,16,18,20)$
- `mean(x) ; mean(y)`
- `plot(x, y, main="function example")`



The screenshot shows an R script editor window titled 'Untitled1\*' with the following code:

```

1 #반올림
2 round(3.5)
3 #올림
4 ceiling(3.5)
5 #내림
6 floor(3.5)
7 |

```

Below the script editor is a console window showing the execution results:

```

> #반올림
> round(3.5)
[1] 4
> #올림
> ceiling(3.5)
[1] 4
> #내림
> floor(3.5)
[1] 3
> |

```

# 작업용 기본 디렉토리 설정

> `setwd("c:\\r_temp")` <-- 이렇게 지정하면 됩니다.

> `getwd()` <-- 현재 설정된 작업 디렉터리를 확인합니다.

[1] "c:/r\_temp"

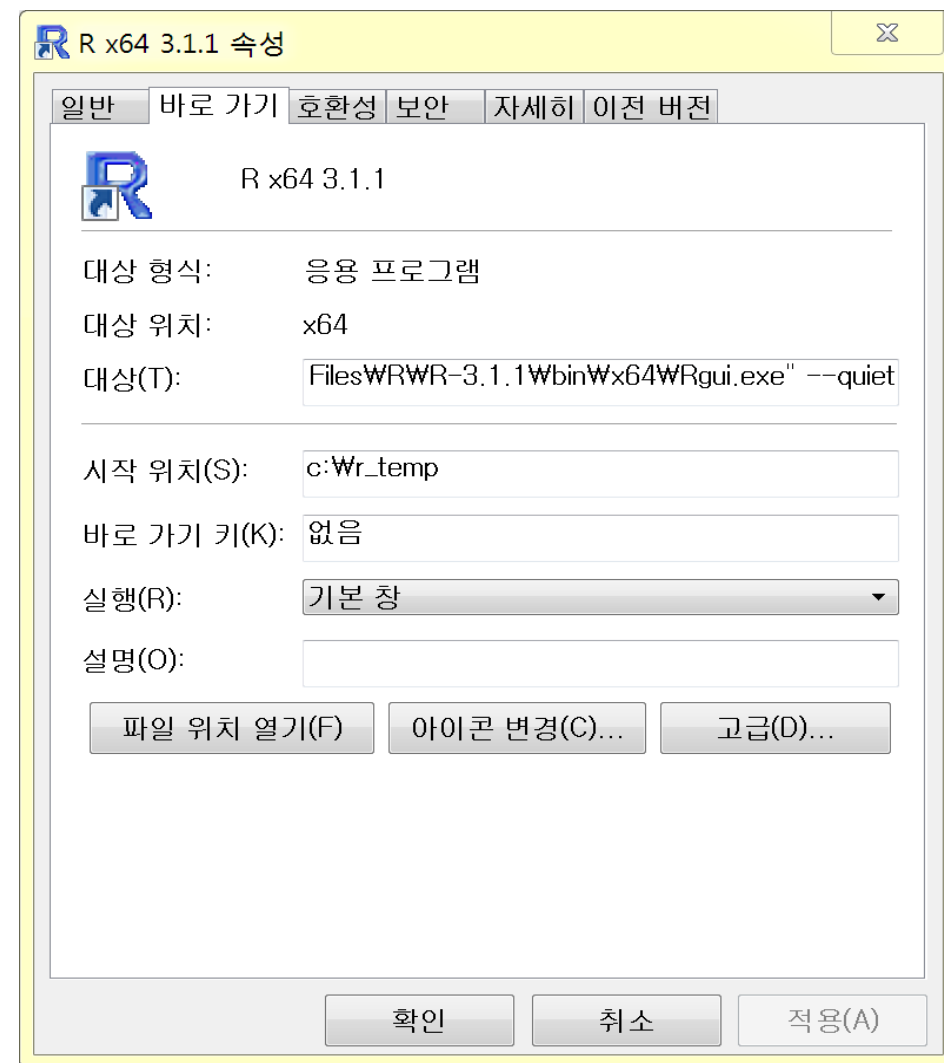
또는 아래와 같이 지정해도 됩니다.

> `setwd("c:/r_temp")` <-- 슬래쉬 기호인 것에 주의하세요

> `getwd()`

[1] "c:/r\_temp"

>



# 명령문과 참조문

## ➤ 명령문(instruction) 과 참조문 (주석, comment)

### ➤ 명령문: 직접명령문, 할당명령문

- `25*3/13` #직접명령문
- `print(25*3/13, digit=10)` #직접명령문
- `x<-c(1,2,3,4,5)` #할당명령문
- `assign("x", c(1, 2, 3, 4, 5))` #할당명령문

### ➤ 참조문(주석)

- `#(hash)` 기호 뒤에 따라오는 표현 또는 문장으로 보통 R 명령문에 대한 설명을 적기 위해 작성
- `x<- (1:10)` **# x에 1부터 10까지 할당**

# 화면에 결과 보여주기

## ➤ print () 함수와 cat () 함수

- print() – 한번에 하나씩 출력, cat() – 여러 개 출력

```
> print(1+2) <-- 이렇게 print 안에 출력하고 싶은 내용 쓰면 됩니다.
```

```
[1] 3
```

```
> 1+2 <-- 이렇게 해도 사실 print 명령이 생략된 것입니다.
```

```
[1] 3
```

```
> print('a') <-- 문자를 출력할 때는 홑따옴표를 붙여야 합니다.
```

```
[1] "a"
```

```
> 'a' <-- 그냥 문자만 입력해도 정상적으로 출력됩니다.
```

```
[1] "a"
```

```
> print(pi) <-- 소수점일 경우 총 7 자리로 출력합니다.
```

```
[1] 3.141593
```

```
> print(pi,digits=3) <-- digits 로 자리수 지정할 수 있습니다.
```

```
[1] 3.14
```

# 화면에 결과 보여주기

> **print(3,4)** <--- 두 개를 출력했는데 한 개만 나오죠??

[1] 3

> **print('a','b')** <-- 문자의 경우는 아예 에러가 납니다.

다음에 오류가 있습니다 `print.default("a", "b")` : 'digits' 인자가 잘못되었습니다

추가정보: 경고메시지:

In `print.default("a", "b")` : 강제형변환에 의해 생성된 NA 입니다

> **cat(1,':','a','\n',2,':','b')** <-- 숫자와 문자 여러 개를 한꺼번에 출력시켰습니다.

1 : a

2 : b>

**\n** 문자란?



# 화면에 결과 보여주기

- ▶ 여러 개의 명령을 연속으로 보여줄 때 ‘;’을 사용

```
> 1;2;3 <-- 각 명령을 세미콜론으로 구분했습니다.
```

```
[1] 1
```

```
[1] 2
```

```
[1] 3
```

```
> 1+2 ; 2*3 ; 4/2 <--세미콜론 을 기준으로 순서대로 명령이 실행됩니다.
```

```
[1] 3
```

```
[1] 6
```

```
[1] 2
```

# R Data Type

# R Data Type

- R은 기본 데이터타입은 vector 를 포함하여 다음과 같은 다양한 데이터 타입을 제공하고 있음

Data type	Description
벡터(vector)	<ul style="list-style-type: none"> <li>a sequence of numbers or characters, or higher-dimensional arrays like matrices</li> <li>하나 이상의 값으로 구성된 1차원 데이터</li> </ul>
팩터(factor)	<ul style="list-style-type: none"> <li>a sequence assigning a category to each index, used to represent categorical data</li> <li>질적 자료를 다루는 벡터의 특수한 형태인 범주형 자료</li> </ul>
행렬(matrix)	<ul style="list-style-type: none"> <li>a vector with a dimension attribute. The dimension attribute is itself an integer vector of length 2 (nrow, ncol)</li> <li>2차원 배열형태</li> </ul>
배열(array)	<ul style="list-style-type: none"> <li>Arrays are similar to matrices but can have more than two dimensions.</li> <li>3차원 배열형태</li> </ul>
리스트(list)	<ul style="list-style-type: none"> <li>a collection of objects that may themselves be complicated</li> <li>다양한 속성을 가진 데이터들로 구성된 자료</li> </ul>
데이터프레임(dataframe)	<ul style="list-style-type: none"> <li>a table-like structure (experimental results often collected in this form)</li> <li>테이블 형태의 2차원 배열로 이뤄진 자료</li> </ul>

- Classes of objects, like expression data, are built from these. Most commands in R involve applying a function to an object.

# R Data Type

- A Variable is “Typed” by What it Contains
  - Unlike C variables do not need to be declared and typed. Assigning a sequence of numbers to `x` forces `x` to be a numeric vector
  - Given `x`, executing `class(x)` reports to the class. This indicates which functions can be used on `x`
  - R has five basic or “atomic” classes of objects:

# R Data Type

- In R the “base” type is a vector, not a scalar.
- A vector is an indexed set of values that are all of the same type. The type of the entries determines the class of the vector. The possible vectors are:
  - integer
  - numeric
  - character
  - complex
  - logical
- integer is a subclass of numeric
- Cannot combine vectors of different modes

# R Data Type

유형	기본형 데이터 유형	세부유형	예
수치형(numeric)	정수	integer	2, 5, 0, -7, -50
	실수	double	0.7, 1/2, pi( $\pi$ )
	지수	exponent	$2^3$
논리형(logical)	참	TRUE	T, TRUE, 1
	거짓	FALSE	F, FALSE, 0
복소수형(complex)	복소수	complex number	2+2i, 0+1i
문자형(character)	문자	character	"A", "a"
	문자열	character string	"character string", "abc"

# R 기본 자료형

## 숫자형 과 주요 산술 연산자

```
> 1+2
```

```
[1] 3
```

기호	의미	사용 예 -> 결과
+	더하기	5+6 -> 11
-	빼기	5-4 -> 1
*	곱하기	5*6 -> 30
/	나누기(실수 가능)	4/2 -> 2
%%	정수 나누기	위와 동일
%%	나머지 구하기	5%%4 -> 1
^, **	승수 구하기	3^2 -> 9, 3^3 -> 27

# R 기본 자료형

## 연산자의 우선순위 주의

> **1+2\*3** <-- 연산자의 우선순위에 따라 뒤의 \* (곱하기)부터 연산합니다

[1] 7

> **(1+2)\*3** <-- 만약 더하기부터 하려면 왼쪽처럼 괄호를 사용해야 합니다.

[1] 9



# R 기본 자료형 - 숫자

> 10000 <-- 0 이 4개 까지는 그대로 나옵니다

[1] 10000

> 100000 <-- 0 이 5개부터는 e로 표시됩니다.

[1] 1e+05

> 1000000 <-- 0이 6 개라서 결과에  $1 * 10^6$  으로 표시가 됩니다.

[1] 1e+06

> 1e2 <-- 이 말은  $1 * 10^2$  이라는 뜻이므로 100 이 나옵니다.

[1] 100

> 3e2 <-- 이 말은  $3 * 10^2$  아는 뜻이라서  $3 * 100$  의 결과인 300 이 나옵니다.

[1] 300

> 3e-1 <-- -1 은 소숫점 1 자리까지 표시하라는 뜻입니다.

[1] 0.3

> 3e-2 <-- -2 는 소수점 2 자리까지 표시하라는 뜻입니다.

[1] 0.03

## Tip!

> options(scipen = 100)

을 실행하게 되면 지수로 표현되는 한계치가 좀더 늘어납니다.

단, 이럴경우 10의 22제곱을 넘어가면 근사값이 나오게 됩니다.

# R 기본 자료형 - 숫자

## ▶ 강제로 숫자형으로 변환하기

```
> '1' + '2' <-- 숫자처럼 보여도 사실은 문자입니다.
```

다음에 오류가 있습니다 "1" + "2" : 이항연산자에 수치가 아닌 인수입니다

```
> as.numeric('1') + as.numeric('2') <-- 숫자로 강제로 변환했습니다.
```

```
[1] 3
```

# R 기본 자료형 - 문자

## 문자형

```
> 'First' <-- 문자라서 홑따옴표로 감싸고 출력했습니다.
```

```
[1] "First"
```

```
> "Second" <-- 이렇게 쌍따옴표로 감싸도 됩니다.
```

```
[1] "Second"
```

```
> First <-- 그냥 사용하면 변수 이름으로 인식이 되어서 에러가 발생합니다.
```

에러: 객체 'First'를 찾을 수 없습니다

```
> class('1') <-- 문자형 데이터를 검사합니다.
```

```
[1] "character"
```

```
> class(1) <-- 숫자형 데이터를 검사합니다.
```

```
[1] "numeric"
```

# R 기본 자료형 - 논리값

## ➤ Logical (논리값) – TRUE/FALSE or T/F

```
> 3 & 0 <-- 3 곱하기 0 의 뜻으로 거짓(FALSE)이 됩니다.
```

```
[1] FALSE
```

```
> 3 & 1 <-- 3 곱하기 1 의 뜻으로 참 곱하기 참이라서 참(TRUE)이 됩니다.
```

```
[1] TRUE
```

```
> 3 & 2 <-- 참 곱하기 참이라서 결과도 참(TRUE) 인 거 아시겠죠?
```

```
[1] TRUE
```

```
> 3 | 0 <-- 참 더하기 거짓 이라서 결과는 참(TRUE) 이 나옵니다.
```

```
[1] TRUE
```

```
> 3 | 1 <-- 참 더하기 참 이라서 결과는 참(TRUE) 이 나옵니다.
```

```
[1] TRUE
```

```
> !0 <-- 거짓이 아닌 것이라서 참(TRUE)이 나옵니다.
```

```
[1] TRUE
```

```
> !1 <-- 참이 아닌 것이라서 거짓(FALSE) 이 나옵니다.
```

```
[1] FALSE
```

```
> !3 <-- 참이 아닌 것이라서 거짓(FALSE) 이 나옵니다.
```

```
[1] FALSE
```

# R 기본 자료형 – Missing Value

## ➤ 결측치 (Missing value)

- One smart feature of R is that it allows for missing values in vectors and datasets; it denotes them as NA. You don't have to coerce missing values to, say 0, in order to have a meaningful vector. Many functions, like `cor()`, have options for handling missing values without just exiting. How to find NA values in a vector?
- NA는 일반적으로 결측값을 의미하며 제외하고 분석
- `is.na()` is used to test objects if they are NA
- Most R functions come with the optional argument, `na.rm`, which stands for NA remove. R will ignore NAs when it evaluates a function if you add the argument `na.rm = TRUE`:

```
vec <- c(1, 2, 3, NA)
is.na(vec)
## FALSE FALSE FALSE TRUE
```

> `sum(1,2,NA)` <-- NA 값이 연산 결과를 틀리게 만들었습니다.

[1] NA

> `sum(1,2,NA,na.rm=T)` <-- `na.rm=T` 로 NA 값을 제거하고 올바른 계산을 했습니다.

[1] 3

# Vector

# 벡터 생성하기

## ➤ 벡터의 생성 방법

- `x <- c(1, 2, 3, 4, 5)`
- `x <- 1:5`
- `x <- 5:1`

## ➤ 연속된 숫자 생성을 위한 `seq()` 함수

- `x <- seq(from, to, by, length.out)`
- `x <- seq(from=1, to=10, by=1)`
- `x <- seq(from=1, to=2, length.out =10)`
- `x <- seq(1, 10, 1)`
- `x <- seq(1, 2, length.out=10)`

```
> x <- seq(from=1, to=10, by=1);x
[1] 1 2 3 4 5 6 7 8 9 10
> x <- seq(from=1, to=2, length.out =10);x
[1] 1.000000 1.111111 1.222222 1.333333 1.444444 1.555556
[7] 1.666667 1.777778 1.888889 2.000000
> x <- seq(1, 10, 1);x
[1] 1 2 3 4 5 6 7 8 9 10
> x <- seq(1, 2, length.out=10);x
[1] 1.000000 1.111111 1.222222 1.333333 1.444444 1.555556
[7] 1.666667 1.777778 1.888889 2.000000
```

# 벡터 사용하기

## ➤ 수열 생성을 위한 rep() 함수

- `x <- rep(x, each, times, length.out)`
- `x <- rep(c(1, 2, 3), times=3)`
- `x <- rep(1:5, each=3, times=2)`

```
> x <- rep(c(1, 2, 3), times=3);x
[1] 1 2 3 1 2 3 1 2 3
> x <- rep(1:5, each=3, times=2);x
[1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5 1 1 1 2 2 2 3 3 3 4 4 4
[28] 5 5 5
```

## ➤ 벡터의 원소 추출 : [ ] 사용

- `a <- c("한국", "중국", "일본")`
- `b <- c(1, 2, 3, 4, 5)`
- `a[1] ; a[1:3]`
- `b[b>3]`

```
> a[1] ; a[1:3]
[1] "한국"
[1] "한국" "중국" "일본"
> b[b>3]
[1] 4 5
```



# 벡터 함수들

## ➤ 벡터에 원소추가 append 함수

- `append(x, value, after)`
- `append(a, "미국", 2) ; a`
- `append(a, "독일") ; a`

```
> append(a, "미국", 2)
[1] "한국" "중국" "미국" "일본"
> append(a, "독일")
[1] "한국" "중국" "일본" "독일"
```

## ➤ 개별원소에 이름부여 names 함수

- `b <- c(5, 6, 7, 8, 9, 10)`
- `names(b) <- c(2015:2020)`
- `b["2015"]`

```
> b["2015"]
2015
     5
```

# 벡터 함수들

## ▶ 문자결합을 위한 함수 paste()

- `paste(000, "00", sep= "")`
- `name<-paste(2015:2019, "y", sep= "") ;`
- `names(b) <- name`
- `b["2015y"]`

```
> name
[1] "2015y" "2016y" "2017y" "2018y" "2019y"

> b["2015y"]
2015y
      5
```

## ▶ 주의할 점 : 문자, 숫자를 섞어서 할당할 때! (문자로 변환됨)

- `MIX<-c(1,2,3,4, "아대")`
- `as.numeric(MIX[1:4])`
- `MIX[1:4]<-as.numeric(MIX[1:4])`

```
> MIX<-c(1,2,3,4, "아대");MIX
[1] "1"      "2"      "3"      "4"      "아대"
> class(MIX)
[1] "character"
> as.numeric(MIX[1:4])
[1] 1 2 3 4
> MIX[1:4]<-as.numeric(MIX[1:4]);MIX
[1] "1"      "2"      "3"      "4"      "아대"
> class(MIX)
[1] "character"
```

# Vector Subsetting

## ➤ Extracting Subsequences of a Vector

- Getting elements of a vector with desired properties is extremely common, so there are robust tools for doing it.
- An element of a vector `v` is assigned an index by its position in the sequence, starting with **1**. The basic function for subsetting is **`[]`**.
- **`v[1]` is the first element, `v[length(v)]` is the last.**
- The subsetting function takes input in many forms.

# Vector Subsetting

## ➤ Subsetting with Positive Integral Sequences

- `> v <- c("a", "b", "c", "d", "e")`

```
> J <- c(1, 3, 5)
```

```
> v[J]
```

```
[1] "a" "c" "e"
```

```
> v[1:3]
```

```
[1] "a" "b" "c"
```

```
> v[2:length(v)]
```

```
[1] "b" "c" "d" "e"
```

# Vector Subsetting

## ➤ Subsetting with Negated Integral Sequences

- This is a tool for removing elements or subsequences from a vector.

- `> v`

```
[1] "a" "b" "c" "d" "e"
```

```
> J
```

```
[1] 1 3 5
```

```
> v[-J]
```

```
[1] "b" "d"
```

- `> v[-1]`

```
[1] "b" "c" "d" "e"
```

```
> v[-length(v)]
```

```
[1] "a" "b" "c" "d"
```

# Vector Subsetting

## ➤ Subsetting with Logical Vector (Important!)

- Given a vector  $x$  and a logical vector  $L$  of the same length as  $x$ ,  $x[L]$  is the vector of entries in  $x$  matching a TRUE in  $L$ .

```
> L <- c(TRUE, FALSE, TRUE, FALSE, TRUE)
```

```
> v[L]
```

```
[1] "a" "c" "e"
```

```
> x <- seq(-3, 3);x
```

```
[1] -3 -2 -1  0  1  2  3
```

```
> x >= 0
```

```
[1] FALSE FALSE FALSE TRUE TRUE TRUE TRUE
```

```
> x[x >= 0]
```

```
[1] 0 1 2 3
```

# Vector Subsetting

## ➤ Subsetting by Names

```
> names(x) <- paste("N", 1:length(x), sep = "");x
```

```
  N1 N2 N3 N4 N5 N6 N7
```

```
 -3 -2 -1  0  1  2  3
```

```
> names(x)[x < 0]
```

```
[1] "N1" "N2" "N3"
```

```
> y <- c(x, NA, NA)
```

```
> z <- y[!is.na(y)]
```

```
> z
```

```
 N1 N2 N3 N4 N5 N6 N7
```

```
-3 -2 -1  0  1  2  3
```

```
> x
[1] -3 -2 -1  0  1  2  3
```

# Vector Subsetting

- If  $x$  is a vector with names and  $A$  is a subsequence of  $\text{names}(x)$ , then  $x[A]$  is the corresponding subsequence of  $x$ .

```
> x
```

```
N1 N2 N3 N4 N5 N6 N7
```

```
-3 -2 -1 0 1 2 3
```

```
> x[c("N1", "N3")]
```

```
N1 N3
```

```
-3 -1
```



# Vector Subsetting

- A subset expression can be on the receiving end of an assignment, in which case the assignment only applies the subset and leaves the rest of the vector alone.

```
> z <- 1:4
```

```
> z[1] <- 0
```

```
> z
```

```
[1] 0 2 3 4
```

```
> z[z <= 2] <- -1
```

```
> z
```

```
[1] -1 -1 3 4
```

```
> w <- c(1:3, NA, NA)
```

```
> w[is.na(w)] <- 0
```

```
> w
```

```
[1] 1 2 3 0 0
```

# 참고문헌

- R을 활용한 데이터 분석 - 김성근
- R 리뷰 - 서진수
- 생초짜를 위한 R - 정태훈

END

Thank you for your attention



경청해주셔서 감사합니다