

2022년 한국생명공학연구원 연구데이터  
분석과정 R

합성생물학전문연구소 김하성

2022-06-08



# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	강의 개요 . . . . .	5
1.2	강의 계획 . . . . .	5
1.3	참고 자료 . . . . .	5
<b>2</b>	<b>R/Rstudio basics</b>	<b>7</b>
2.1	What is R / Rstudio . . . . .	7
2.2	R / Rstudio Installation . . . . .	8
2.3	Rstudio interface . . . . .	10
2.4	Start a project . . . . .	11
2.5	Getting help . . . . .	12
2.6	R packages and Dataset . . . . .	14
<b>3</b>	<b>Rmarkdown</b>	<b>17</b>
3.1	Rmarkdown의 기본 작동 원리 . . . . .	18
3.2	코드 입력 . . . . .	20
3.3	Markdown 문법 . . . . .	21
3.4	스타일 . . . . .	23
3.5	테이블 . . . . .	24
3.6	YAML 헤더 . . . . .	24
3.7	Output format . . . . .	25
<b>4</b>	<b>R programming</b>	<b>27</b>
4.1	Console calculator . . . . .	27
4.2	What is a programming language . . . . .	28
4.3	Data and variables . . . . .	29
4.4	Object (Data structure) . . . . .	31
4.5	Functions . . . . .	40
4.6	Flow control . . . . .	45
4.7	Object Oriented Programming (Advanced) . . . . .	49
<b>5</b>	<b>Data transformation</b>	<b>51</b>
5.1	Reading and writing . . . . .	52
5.2	Subset . . . . .	54

5.3	Merging and Split	55
5.4	Transformation	56
5.5	Babies example	57
5.6	Useful functions	58
5.7	apply	60
<b>6</b>	<b>Lecture note 2</b>	<b>65</b>
6.1	numeric vector	65
6.2	matrix	69
6.3	data.frame	69
6.4	list	70
6.5	functions	71
6.6	for	72
6.7	Data transformation	73
<b>7</b>	<b>R basic graphics</b>	<b>75</b>
7.1	scatter plot	75
7.2	histogram	75
7.3	boxplot	76
7.4	barplot	77
7.5	Draw multiple graphs in the same plot	77
7.6	Usuful functions II	80
<b>8</b>	<b>tidyverse</b>	<b>83</b>
8.1	tibble object type	83
8.2	Tidy data structure	85
8.3	Pivoting	87
8.4	Separating and uniting	89
8.5	dplyr and pipe operator	89
8.6	dplyr - Important functions	91
8.7	Code comparison	94

# Chapter 1

## Introduction

### 1.1 강의 개요

- 목표: 생물 데이터 분석을 위한 R 사용법과 (Rstudio, Tidyverse, Bioconductor 포함) 프로그래밍 기술을 습득함
- 장소: 코빅 3층 전산교육장(1304호)
- 강사: 한국생명공학연구원 합성생물학전문연구단 김하성
- 연락처: 042-860-4372, haseong@kribb.re.kr
- 강의자료: <https://greendaygh.github.io/kribbr2022/>

### 1.2 강의 계획

1. R 사용법 및 데이터 분석 기초 5.19(목), 5.26(목)
2. R/Tidyverse 데이터 분석 중급 6.9(목), 6.16(목)
3. R/Tidyverse 활용 데이터 가시화 7.7(목), 7.14(목)
4. R/Bioconductor 활용한 바이오데이터 분석 기초 8.4(목), 8.11(목)
5. R/Bioconductor 활용한 NGS 데이터 분석 기초 9.1(목), 9.15(목)
6. R/Bioconductor 활용한 NGS 데이터 분석 및 Workflow 10.6(목), 10.13(목)

### 1.3 참고 자료

- R 홈페이지
- Rstudio 홈페이지
- Bioconductor
- R 기본 문서들
- R ebooks
- Cheat Sheets

- RStudio Webinars
- Shiny
- Hadley github
- R for Data Science
- Using R for Introductory Statistics by John Verzani
  - Free version of 1st Edition
  - Second edition
- Bioinformatics Data Skills by Vince Buffalo
- Introductory Statistics with R by Dalgaard
- 일반통계학 (영지문화사, 김우철 외)

## Chapter 2

# R/Rstudio basics

### 2.1 What is R / Rstudio



R은 통계나 생물통계, 유전학을 연구하는 사람들 사이에서 널리 사용되는 오픈소스 프로그래밍 언어입니다. Bell Lab에서 개발한 S 언어에서 유래했으며 많은 라이브러리 (다른 사람들이 만들어 놓은 코드)가 있어서 쉽게 가져다 사용할 수 있습니다. R은 복잡한 수식이나 통계 알고리즘을 간단히 구현하고 사용할 수 있으며 C, C++, Python 등 다른 언어들과의 병행 사용도 가능합니다. R은 IEEE에서 조사하는 Top programming languages에서 2018년 7위, 2019년 5위, 2020년 6위, 2021년 7위로 꾸준히 높은 사용자를 확보하며 빅데이터, AI 시대의 주요한 프로그래밍 언어로 사용되고 있습니다.

R은 데이터를 통계분석에 널리 사용되는데 이는 데이터를 눈으로 확인하기 위한 visualization이나 벡터 연산 등의 강력한 기능 때문에 점점 더 많은 사람들이 사용하고 있습니다. 기존에는 속도나 확장성이 다른 언어들에 비해 단점으로 지적되었으나 R 언어의 지속적인 개발과 업데이트로 이러한 단점들이 빠르게 보완되고 있습니다. R 사용을 위해서는 R 언어의 코어 프로그램을 먼저 설치하고 그 다음 R 언어용 IDE(Integrated Development Environment)인 RStudio 설치가 필요합니다.



Rstudio는 R 언어를 위한 오픈소스 기반 통합개발환경(IDE)으로 R 프로그래밍을 위한 편리한 기능들을 제공해 줍니다. R언어가 주목을 받고 두터운 사용자 층을

Rank	Language	Type	Score
1	Python	🌐 🖥️ 📱	100.0
2	Java	🌐 📱 🖥️	95.4
3	C	📱 🖥️ 📱	94.7
4	C++	📱 🖥️ 📱	92.4
5	JavaScript	🌐	88.1
6	C#	🌐 📱 🖥️ 📱	82.4
7	R	🖥️	81.7
8	Go	🌐 🖥️	77.7
9	HTML	🌐	75.4
10	Swift	📱 🖥️	70.4

Figure 2.1: <https://spectrum.ieee.org/top-programming-languages/>

확보할 수 있게된 핵심 동력이 Rstudio 입니다. 자체적으로 최고수준의 오픈소스 개발팀이 있으며 tidyverse, ‘shiny’ 등의 데이터 분석 관련 주요 패키지를 개발하였고 정기적으로 conference 개최를 하면서 기술 보급의 핵심 역할을 하고 있습니다.

## 2.2 R / Rstudio Installation

### 2.2.1 R 설치

- R 사이트에 접속 후 (<https://www.r-project.org/>) 좌측 메뉴 상단에 위치한 CRAN 클릭.
- 미러 사이트 목록에서 Korea의 아무 사이트나 들어감
- Download R for Windows를 클릭 후 base 링크 들어가서
- Download R x.x.x for Windows 링크 클릭으로 실행 프로그램 다운로드
- 로컬 컴퓨터에 Download 된 R-x.x.x-win.exe 를 실행 (2022.5 현재 R 버전은 4.2.0).
- 설치 프로그램의 지시에 따라 R 언어 소프트웨어 설치를 완료

### 2.2.2 Rstudio 설치

- 사이트에 접속 (<https://www.rstudio.com/>), 상단의 Products > RStudio 클릭
- RStudio Desktop 선택
- Download RStudio Desktop 클릭

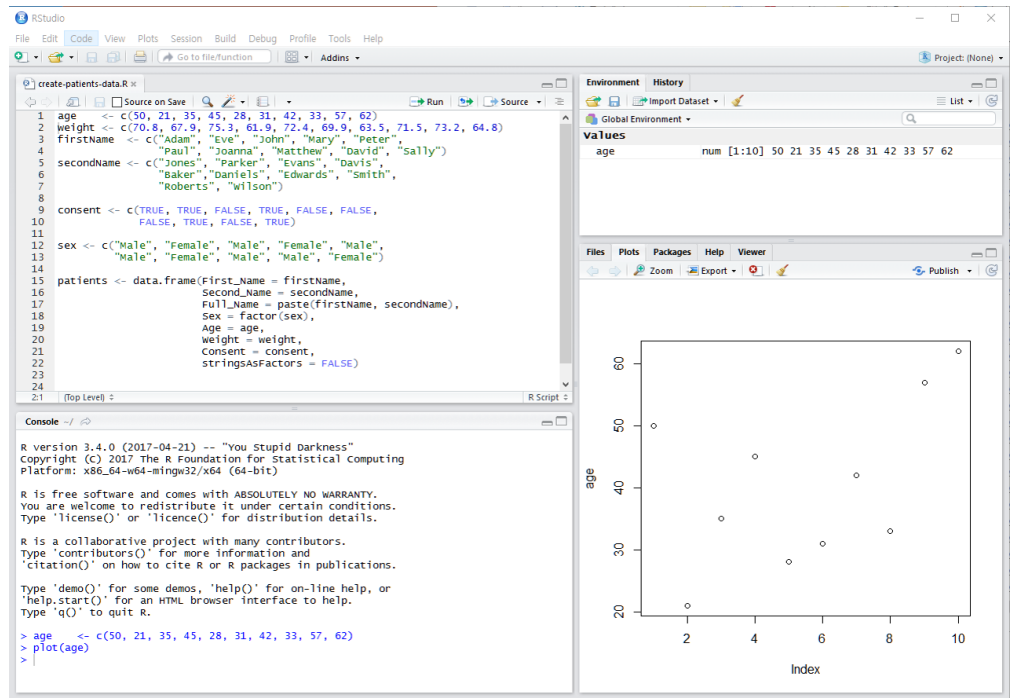


Products	About RStudio		Additional Websites	
<b>OPEN SOURCE</b>	<b>LEARNING</b>	<b>SUPPORT</b>	<b>ANALYSE &amp; EXPLORE</b>	<b>CONNECT &amp; INTEGRATE</b>
RStudio Desktop	Education	Frequently Asked Questions	Tidyverse	Professional Drivers
RStudio Server	Videos & Webinars	RStudio Support	ggplot2	Launcher Plugin SDK
Shiny Server	Cheatsheets	RStudio Community	dplyr	Databases
R Packages	rstudio::conf	Certified Partners	tidyr	Environments
		Product Security	purrr	Sparklyr
<b>HOSTED SERVICES</b>	<b>ABOUT US</b>	RStudio Documentation	<b>COMMUNICATE &amp; INTERACT</b>	Plumber
RStudio Academy	About the Company	Contact Us	Shiny	Reticulate
RStudio Cloud	What Makes Us Different	RStudio Legal Terms	rmarkdown	Ursa Labs
RStudio Public Package Manager	Analyst Reports	Email Subscription Management	flexdashboard	<b>MODEL &amp; PREDICT</b>
shinyapps.io	RStudio Swag			Tensorflow
	Careers			Tidymodels
<b>PROFESSIONAL</b>				Spark MLlib
RStudio Team				
RStudio Workbench				
RStudio Connect				
RStudio Package Manager				

Figure 2.2: <https://www.rstudio.com/>

- RStudio Desktop Free 버전의 Download를 선택하고
- Download RStudio for Windows 클릭, 다운로드
- 로컬 컴퓨터에 다운로드된 RStudio-x.x.x.exe 실행 (2022.5 현재 RStudio Desktop 2022.02.2+485)
- 설치 가이드에 따라 설치 완료

## 2.3 Rstudio interface



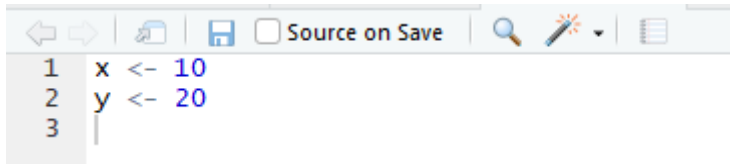
- 기본 화면에서 좌측 상단의 공간은 코드편집창, 좌측 하단은 콘솔창
- 각 위치를 기호에 따라서 바꿀 수 있음 (View -> Pane)

### 2.3.1 Keyboard shortcuts

- 참고사이트
  - <https://support.rstudio.com/hc/en-us/articles/200711853-Keyboard-Shortcuts>
  - Tools -> Keyboard shortcut Quick Reference (Alt + Shift + K)
- 코드편집창 이동 (Ctrl + 1) 콘솔창 이동 (Ctrl + 2)
- 한 줄 실행 (Ctrl + Enter)
- 저장 (Ctrl + S)
- 주석처리 (Ctrl + Shift + C)
  - 또는 #으로 시작하는 라인
- 탭 이동 (Ctrl + F11, Ctrl + F12)
- 코드편집창 확대 (Shift + Ctrl + 1) 콘솔창 확대 (Shift + Ctrl + 2)
- 컬럼 편집 (Alt + )
- 자동 완성 기능 (Tab completion) in RStudio

### Exercises

1. 코드편집창에서 다음을 입력/실행하고 단축키를 사용하여 주석을 넣으시오



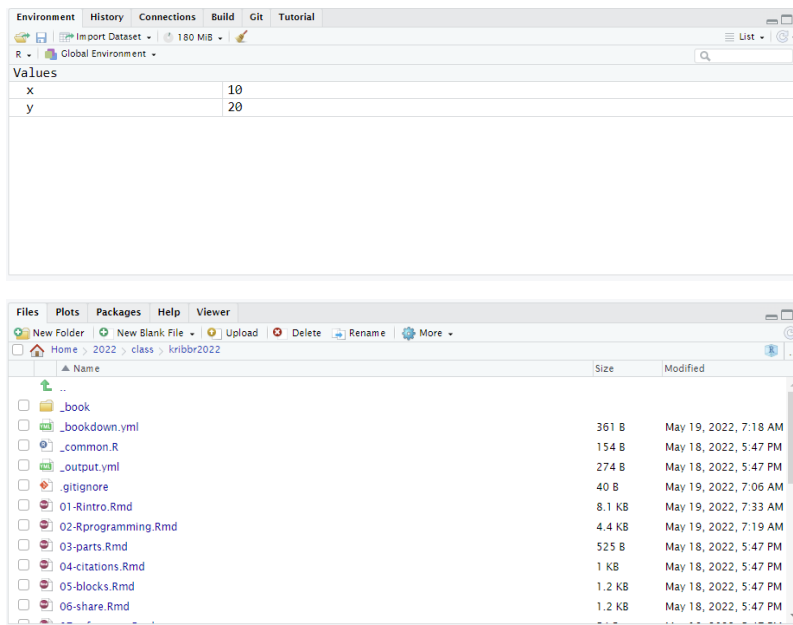
- 단축키 Ctrl + enter로 코드 실행
- 단축키 Ctrl + 2로 커서 콘솔창으로 이동
- x값 x+y값 확인
- 단축키 Ctrl + 1로 코드편집창 이동
- 단축키 Ctrl + Shift + C 사용

```

# x <- 10
# y <- 20

```

## 2.3.2 Environment and Files



## 2.4 Start a project

프로젝트를 만들어서 사용할 경우 파일이나 디렉토리, 내용 등을 쉽게 구분하여 사용 가능합니다. 아래와 같이 임의의 디렉토리에 kribbR 이라는 디렉토리를 생성하고 lecture1 프로젝트를 만듭니다.

File > New Project > New Directory > New Project > “kribbR” >  
Create Project

시작할 때는 해당 디렉토리의 `xxx.Rproj` 파일을 클릭합니다. Rstudio 오른쪽 상단 프로젝트 선택을 통해서 빠르게 다른 프로젝트의 작업공간으로 이동할 수 있습니다.

### 2.4.1 Hello world

File > New File > R markdown > OK

```
mystring <- "Hello \n world!"  
cat(mystring)  
print(mystring)
```

## 2.5 Getting help

R은 방대한 양의 도움말 데이터를 제공하며 다음과 같은 명령어로 특정 함수의 도움말과 예제를 찾아볼 수 있습니다. `?` 명령을 사용하면 되며 구글이나 웹에서도 도움을 얻을 수 있습니다.

```
help("mean")  
?mean  
example("mean")  
help.search("mean")  
??mean  
help(package="MASS")
```

또한 <https://www.rstudio.com/resources/cheatsheets/>에서는 다양한 R언어의 기능을 한 눈에 알아볼 수 있게 만든 cheatsheet 형태의 문서를 참고할 수 있습니다.

## Base R Cheat Sheet

### Getting Help

Accessing the help files

**?mean**  
Get help of a particular function.  
**help.search('weighted mean')**  
Search the help files for a word or phrase.  
**help(package = 'dplyr')**  
Find help for a package.

More about an object

**str(iris)**  
Get a summary of an object's structure.  
**class(iris)**  
Find the class an object belongs to.

### Using Packages

**install.packages('dplyr')**  
Download and install a package from CRAN.

**library(dplyr)**  
Load the package into the session, making all its functions available to use.

**dplyr::select**  
Use a particular function from a package.

**data(iris)**  
Load a built-in dataset into the environment.

### Working Directory

**getwd()**  
Find the current working directory (where inputs are found and outputs are sent).

**setwd('C://file/path')**  
Change the current working directory.

**Use projects in RStudio to set the working directory to the folder you are working in.**

### Vectors

#### Creating Vectors

c(2, 4, 6)	2 4 6	Join elements into a vector
2:6	2 3 4 5 6	An integer sequence
seq(2, 3, by=0.5)	2.0 2.5 3.0	A complex sequence
rep(1:2, times=3)	1 2 1 2 1 2	Repeat a vector
rep(1:2, each=3)	1 1 1 2 2 2	Repeat elements of a vector

#### Vector Functions

<b>sort(x)</b> Return x sorted. <b>table(x)</b> See counts of values.	<b>rev(x)</b> Return x reversed. <b>unique(x)</b> See unique values.
--	---

#### Selecting Vector Elements

By Position	
<b>x[4]</b>	The fourth element.
<b>x[-4]</b>	All but the fourth.
<b>x[2:4]</b>	Elements two to four.
<b>x[-(2:4)]</b>	All elements except two to four.
<b>x[c(1, 5)]</b>	Elements one and five.
By Value	
<b>x[x == 10]</b>	Elements which are equal to 10.
<b>x[x &lt; 0]</b>	All elements less than zero.
<b>x[x %in% c(1, 2, 5)]</b>	Elements in the set 1, 2, 5.
Named Vectors	
<b>x['apple']</b>	Element with name 'apple'.

### Programming

#### For Loop

```
for (variable in sequence){
  Do something
}
```

Example

```
for (i in 1:4){
  j <- i + 10
  print(j)
}
```

#### While Loop

```
while (condition){
  Do something
}
```

Example

```
while (i < 5){
  print(i)
  i <- i + 1
}
```

#### If Statements

```
if (condition){
  Do something
} else {
  Do something different
}
```

Example

```
if (i > 3){
  print('Yes')
} else {
  print('No')
}
```

#### Functions

```
function_name <- function(var){
  Do something
  return(new_variable)
}
```

Example

```
square <- function(x){
  squared <- x*x
  return(squared)
}
```

### Reading and Writing Data

Also see the [readr](#) package.

Input	Output	Description
df <- read.table('file.txt')	write.table(df, 'file.txt')	Read and write a delimited text file.
df <- read.csv('file.csv')	write.csv(df, 'file.csv')	Read and write a comma separated value file. This is a special case of read.table/write.table.
load('file.Rdata')	save(df, file = 'file.Rdata')	Read and write an R data file, a file type special for R.

Conditions

a == b	Are equal	a > b	Greater than	a >= b	Greater than or equal to	is.na(a)	Is missing
a != b	Not equal	a < b	Less than	a <= b	Less than or equal to	is.null(a)	Is null

RStudio® is a trademark of RStudio, Inc. • CC-BY-Mhairi McNeill • mhairimcneill@gmail.com

Learn more at [web page](#) or [vignette](#) • package version • Updated: 3/25

### Types

Converting between common data types in R. Can always go from a higher value in the table to a lower value.

as.logical	TRUE, FALSE, TRUE	Boolean values (TRUE or FALSE)
as.numeric	1, 0, 1	Integers or floating point numbers
as.character	'1', '0', '1'	Character strings. Generally preferred to factors.
as.factor	'1', '0', '1', levels: '1', '0'	Character strings with preset levels. Needed for some statistical models.

### Maths Functions

log(x)	Natural log.	sum(x)	Sum.
exp(x)	Exponential.	mean(x)	Mean.
max(x)	Largest element.	median(x)	Median.
min(x)	Smallest element.	quantile(x)	Percentage quantiles.
round(x, n)	Round to n decimal places.	rank(x)	Rank of elements.
signif(x, n)	Round to n significant figures.	var(x)	The variance.
cor(x, y)	Correlation.	sd(x)	The standard deviation.

### Variable Assignment

```
> a <- 'apple'
> a
[1] 'apple'
```

### The Environment

```
ls()
```

List all variables in the environment.

```
rm(x)
```

Remove x from the environment.

```
rm(list = ls())
```

Remove all variables from the environment.

**You can use the environment panel in RStudio to browse variables in your environment.**

### Matrices

```
m <- matrix(x, nrow = 3, ncol = 3)
```

Create a matrix from x.

**m[2, ]** - Select a row

**m[, 1]** - Select a column

**m[2, 3]** - Select an element

**t(m)** - Transpose

**m %\*% n** - Matrix Multiplication

**solve(m, n)** - Find x in: m \* x = n

### Lists

```
l <- list(x = 1:5, y = c('a', 'b'))
```

A list is a collection of elements which can be of different types.

**l[[2]]** - Second element of l.

**l[1]** - New list with only the first element.

**l\$x** - Element named x.

**l['y']** - New list with only element named y.

### Data Frames

```
df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))
```

A special case of a list where all elements are the same length.

**df\$x** - List subsetting

**df[[2]]** - List subsetting

**View(df)** - See the full data frame.

**head(df)** - See the first 6 rows.

**Matrix subsetting**

**df[, 2]** - Number of rows.

**df[2, 1]** - Number of columns.

**df[2, 2]** - Number of columns and rows.

**nrow(df)** - Number of rows.

**ncol(df)** - Number of columns.

**dim(df)** - Number of columns and rows.

**cbind** - Bind columns.

**rbind** - Bind rows.

### Strings

**paste(x, y, sep = ' ')** - Join multiple vectors together.

**paste(x, collapse = ' ')** - Join elements of a vector together.

**grep(pattern, x)** - Find regular expression matches in x.

**gsub(pattern, replace, x)** - Replace matches in x with a string.

**toupper(x)** - Convert to uppercase.

**tolower(x)** - Convert to lowercase.

**nchar(x)** - Number of characters in a string.

### Factors

**factor(x)** - Turn a vector into a factor. Can set the levels of the factor and the order.

**cut(x, breaks = 4)** - Turn a numeric vector into a factor by 'cutting' into sections.

### Statistics

**lm(y ~ x, data=df)** - Linear model.

**glm(y ~ x, data=df)** - Generalised linear model.

**t.test(x, y)** - Perform a t-test for difference between means.

**prop.test** - Test for a difference between proportions.

**summary** - Get more detailed information out a model.

**pairwise.t.test** - Perform a t-test for paired data.

**aov** - Analysis of variance.

### Distributions

	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	rnorm	dnorm	pnorm	qnorm
Poisson	rpois	dpois	ppois	qpois
Binomial	rbinom	dbinom	pbinom	qbinom
Uniform	runif	dunif	punif	qunif

### Plotting

**plot(x)** - Values of x in order.

**plot(x, y)** - Values of x against y.

**hist(x)** - Histogram of x.

### Dates

See the **lubridate** package.

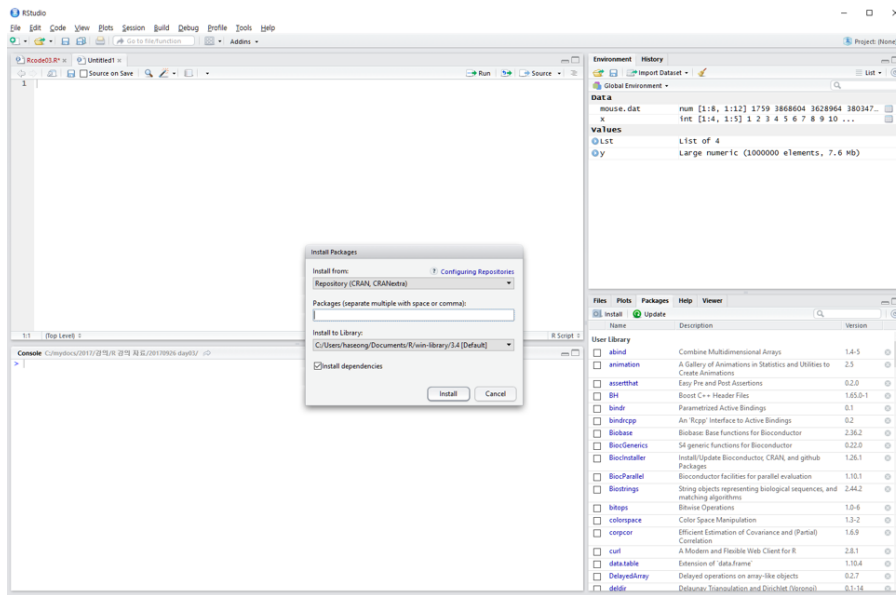
RStudio® is a trademark of RStudio, Inc. • CC BY-MHarr McNeill • mharr@mcneill.com • 844-448-1212 • rstudio.com

Learn more at [web page](#) or [vignette](#) • package version • Updated: 3/15

## 2.6 R packages and Dataset

R 패키지는 함수와 데이터셋의 묶음으로 다른 사람들이 만들어 놓은 코드나 기능을 가져와서 사용하므로써 코드 작성의 수고로움을 줄이고 편리하고 검증된 함수(기능)를 빠르게 도입하여 사용할 수 있다는 장점이 있습니다. 예를 들어 `sd()` 함수는 `stats` package에서 제공하는 함수로써 표준편차 계산을 위한 별도의 함수를 만들어서 사용할 필요가 없이 바로 (`stats` 패키지는 R 기본 패키지로) 별도 설치 없이 바로 사용 가능합니다.

이러한 패키지는 인터넷의 repository에서 구할 수 있으며 대표적인 repository는 The Comprehensive R Archive Network (CRAN) (<http://cran.r-project.org/web/views/>) 와 생물학자를 위한 Bioconductor (<http://www.bioconductor.org/>) 가 있습니다. 이러한 패키지의 설치와 같이 RStudio를 이용하거나 콘솔창에서 `install.packages()` 함수를 이용할 수 있습니다.



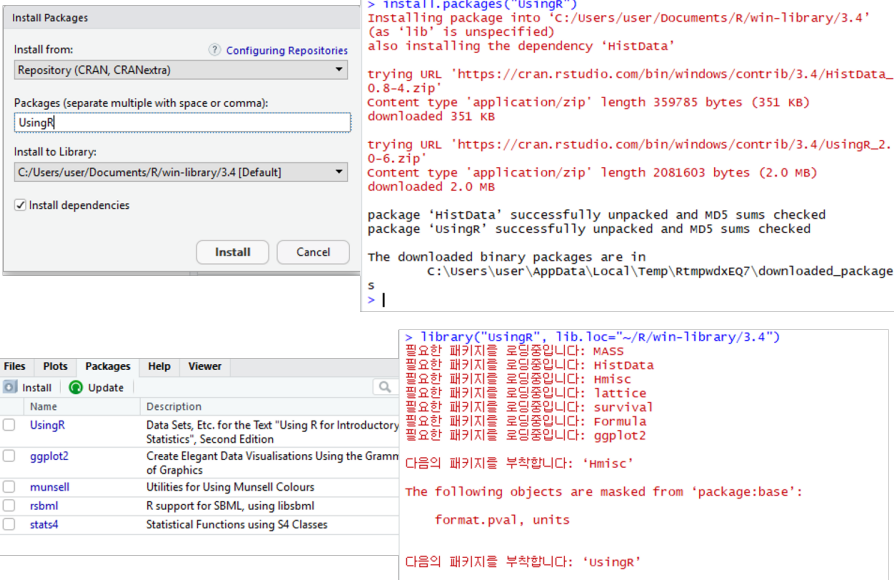
패키지를 설치하고 사용하기 위해서는 `library()` 함수를 사용해서 관련 명령어를 사용하기 전에 미리 loading 해 두어야 합니다. 한 번 로딩으로 작업 세션이 끝날때까지 관련된 함수를 사용할 수 있으나 R 세션이나 RStudio를 재시작 할 경우 다시 로딩해야 사용할 수 있습니다.

```
library(UsingR)
```

- R 설치 디렉토리
- R 패키지 설치 디렉토리

```
.libPaths()  
path.package()
```

## Packages → Install



The screenshot shows the 'Install Packages' dialog box in RStudio. The 'Repository' is set to 'CRAN, CRANextra'. The 'Packages' field contains 'UsingR'. The 'Install to Library' is set to 'C:/Users/user/Documents/R/win-library/3.4 [Default]'. The 'Install dependencies' checkbox is checked. The 'Install' button is highlighted.

The R console output shows the following commands and results:

```
> install.packages("usingR")
Installing package into 'C:/Users/user/Documents/R/win-library/3.4'
(as 'lib' is unspecified)
also installing the dependency 'HistData'

trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.4/HistData_0.8-4.zip'
Content type 'application/zip' length 359785 bytes (351 KB)
downloaded 351 KB

trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.4/UsingR_2.0-6.zip'
Content type 'application/zip' length 2081603 bytes (2.0 MB)
downloaded 2.0 MB

package 'HistData' successfully unpacked and MD5 sums checked
package 'usingR' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
C:\Users\user\AppData\Local\Temp\RtmpwdxEQ7\downloaded_packages
> |
```

The R console also shows the output of the `library()` command:

```
> library("usingR", lib.loc="~/R/win-library/3.4")
필요한 패키지를 로딩중입니다: MASS
필요한 패키지를 로딩중입니다: HistData
필요한 패키지를 로딩중입니다: Hmisc
필요한 패키지를 로딩중입니다: lattice
필요한 패키지를 로딩중입니다: survival
필요한 패키지를 로딩중입니다: formula
필요한 패키지를 로딩중입니다: ggplot2

다음의 패키지를 부하합니다: 'Hmisc'

The following objects are masked from 'package:base':
  format.pval, units

다음의 패키지를 부하합니다: 'usingR'
```

일반적으로 패키지 안에 관련된 데이터도 같이 저장되어 있으며 `data()` 함수를 이용해서 패키지 데이터를 사용자 작업공간에 복사해서 사용 가능합니다.

```
head(rivers)
length(rivers)
class(rivers)
data(rivers)
data(package="UsingR")
library(HistData)
head(Cavendish)
str(Cavendish)
head(Cavendish$density2)
```

이 저작물은 크리에이티브 커먼즈 저작자표시-비영리-변경금지 4.0 국제 라이선스에 따라 이용할 수 있습니다.



## Chapter 3

# Rmarkdown

Rmarkdown은 데이터를 분석하는 코드와 리포트를 동시에 수행할 수 있는 일종의 통합 문서입니다. 워드나 아래한글에서 프로그래밍과 데이터분석을 위한 코드를 작성할 수 있는 경우라고 생각해도 됩니다. Plain-text 기반의 markdown 문법을 사용하며 Rmarkdown으로 작성된 문서는 HTML, PDF, MS word, Beamer, HTML5 slides, books, website 등 다양한 포맷의 출력물로 변환할 수 있습니다.

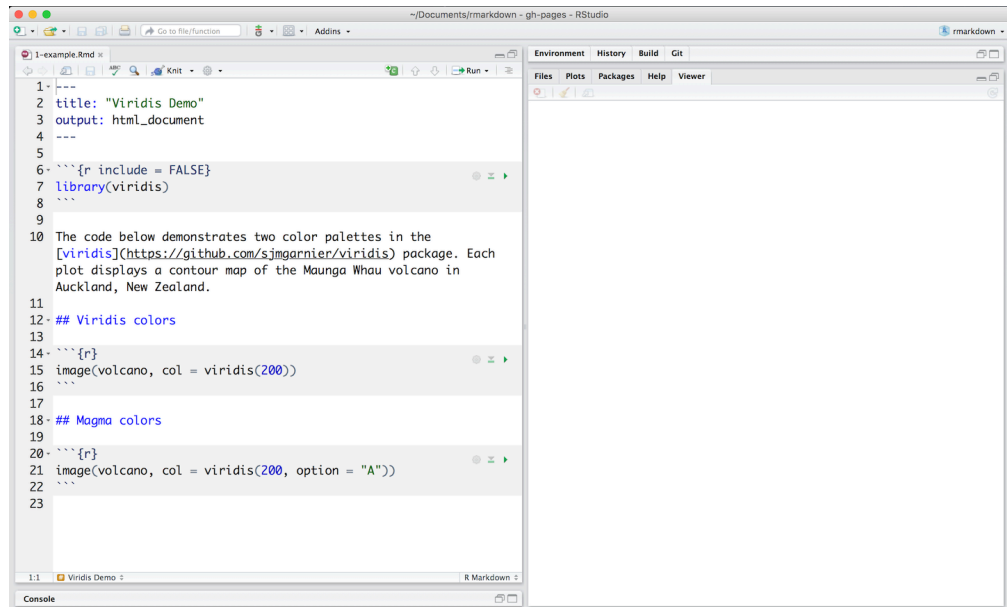


Figure 3.1: Image from [rmarkdown.rstudio.com](http://rmarkdown.rstudio.com)

Rmarkdown 웹사이트에 Rmarkdown 소개 동영상과 Rmarkdown 공식 사이트 메뉴얼 관련 서적 Rmarkdown: The Definitive Guide를 참고하세요. 또한 Rmarkdown을 사용할 때 cheatsheet를 옆에 두고 수시로 보면서 사용하시면 많은 도움이 될 수 있습니다.

### 3.1 Rmarkdown의 기본 작동 원리

Rmarkdown은 plain text 기반으로 작성되며 Rmd 라는 확장자를 갖는 파일로 저장됩니다. 다음과 같은 텍스트 파일이 Rmd 파일의 전형적인 예 입니다.



위 예제에서 네 가지 다른 종류의 콘텐츠를 볼 수 있습니다. 하나는 --- 으로 둘러싸인 내용으로 YAML 이라고 하며 JSON과 같은 데이터 직렬화를 수행하는 하나의 데이터 저장 포맷입니다. 백틱(`) 으로 둘러싸인 코드청그(Code Chunks)라고 하는 부분에는 R이나 python 등의 다양한 코드(실제 작동하는)를 넣어서 사용합니다. 그리고 ### 으로 표시된 글은 제목 글을 나타내며 나머지는 일반적인 텍스트를 나타냅니다.

이러한 Rmarkdown 파일은 render라는 명령어로 원하는 포맷의 문서로 변환할 수 있습니다. 다음 예의 파일을 pdf 형식으로 rendering 하기 위해서는 YAML에 pdf 임을 명시하고 아래와 같이 render함수를 사용하면 됩니다. 또는 Rstudio 코드 입력창 상단의 Knit 버튼으로 pdf나 html 문서를 생성할 수 있습니다.

```

1 ---
2 title: "My R markdown example"
3 output:
4   pdf_document: default
5 ---
6
7 ```{r setup, include=FALSE}
8 library(tidyverse)
9 ```
10
11 ## R Markdown
12
13 This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML,
14 PDF, and MS Word documents. When you click the Knit button a document will be generated
15 that includes both content as well as the output of any embedded R code chunks within the
16 document. You can embed an R code chunk like this:
17
18 ```{r}
19 cars %>% head
20 cars %>%
21   ggplot(aes(x=speed, y=dist)) +
22     geom_point()
23 ```

```

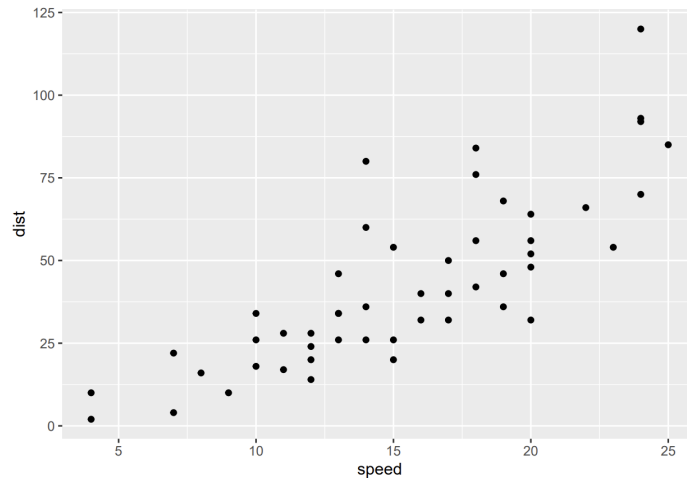
```
render("examples/test.Rmd", output_format = "pdf_document")
```

My R markdown example

#### R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
cars %>% head
cars %>%
  ggplot(aes(x=speed, y=dist)) +
    geom_point()
```



Rmarkdown의 작동 원리는 Rmd 파일을 만든 후 `render` 함수를 부르면 knitr 소프트웨어가 R 코드를 실행시킨 후 markdown (.md) 파일을 생성합니다. 이 후 .md 파일을 pandoc 이라는 문서변환기가 원하는 문서 형태로 변환해 줍니다.

## 3.2 코드 입력

Rmarkdown에서 사용하는 코드청크는 CTRL+ALT+i 단축키를 사용해서 넣을 수 있으며 다음과 같은 몇 가지 옵션으로 코드 스니펫들의 실행/숨김 여부를 결정할 수 있습니다.

- `include = FALSE` : 코드는 실행되지만 보고서에 결과와 코드가 보여지지 않음
- `echo = FALSE` : 코드는 실행되고 보고서에 결과가 포함되지만 코드는 보여지지 않음
- `eval = FALSE` : 코드가 실행되지 않지만 보고서에 코드는 보여짐
- `message = FALSE, warning=FALSE, error=FALSE` : 코드에 의해서 발생하는 메시지/경고/에러가 보고서에 보여지지 않음
- `fig.cap = "..."` : 코드로 그려지는 그래프에 캡션을 붙일 수 있음

```

43 ▾ ```{r}
44 # default
45 n <- c(1, 2, 3)
46 mean(n)
47 ▴ ```
48
49 ▾ ```{r, eval=F}
50 # eval=FALSE
51 n <- c(1, 2, 3)
52 mean(n)
53 ▴ ```
54
55 ▾ ```{r, echo=F}
56 # echo=FALSE
57 n <- c(1, 2, 3)
58 mean(n)
59 ▴ ```

```

Figure 3.2: 코드청크 옵션 예시

실행 결과는 아래와 같습니다.

```

# default
n <- c(1, 2, 3)
mean(n)
#> [1] 2

```

```

# eval=FALSE
n <- c(1, 2, 3)
mean(n)

```

```

#> [1] 2

```

Rmarkdown에서는 ‘r’을 사용해서 코드청크가 아닌 곳에 R 코드를 넣을 수 있습니다. 또한 R 언어 외에도 Python, SQL, Bash, Rcpp, Stan, JavaScript, CSS 등의 다양한 프로그래밍 언어에 대해서도 코드청크 기능을 사용할 수 있습니다. 그런데 이러한 언어들이 사용 가능해지기 위해서는 해당 언어들을 실행해주는

엔진이 있어야 하며 python의 경우 reticulate 라는 패키지가 이러한 기능을 담당합니다. 이 패키지를 설치할 경우 miniconda라는 가상환경 및 데이터 분석을 위한 오픈소스 패키지가 자동으로 설치됩니다.

```
library(reticulate)
```

```
x = "hello, python in R"
print(x.split(' '))
```

아래는 위에 해당하는 소스코드입니다.

```
```{r, eval=F}
library(reticulate)
```
```

```
```{python, eval=F}
x = "hello, python in R"
print(x.split(' '))
```
```

```
['hello,', 'python', 'in', 'R']
```

### 3.3 Markdown 문법

마크다운은 plain text 기반의 마크업 언어로서 마크업 언어는 태그 등을 이용해서 문서의 데이터 구조를 명시하는데 이러한 태그를 사용하는 방법 체계를 마크업 언어라고 합니다. 가장 대표적으로 html 이 있습니다.

```
<html>
  <head>
    <title> Hello HTML </title>
  </head>
  <body>
    Hello markup world!
  </body>
</html>
```

마크다운도 몇 가지 태그를 이용해서 문서의 구조를 정의하고 있으며 상세한 내용은 Pandoc 마크다운 문서를 참고하시기 바랍니다. 마크다운언어의 철학은 쉽게 읽고 쓸 수 있는 문서입니다. plain text 기반으로 작성되어 쓰기 쉬우며 (아직도 사람들이 메모장 많이 사용하는 이유와 같습니다) 태그가 포함되어 있어도 읽는데 어려움이 없습니다. 위 html 언어와 아래 markdown 파일의 예들을 보시면 그 차이를 어렵지 않게 이해할 수 있습니다.

마크다운에서는 Enter를 한 번 입력해서 줄바꿈이 되지 않습니다. <br> 또는 문장 마지막에 공백을 두 개 입력하면 되겠습니다.

이 문장은 줄바꿈이 되지 않습니다

```
<br>
```

이 문장은 줄바꿈이 됩니다

마크다운 태그를 몇 가지 살펴보면 먼저 # 을 붙여서 만드는 header 가 있습니다.

```
# A level-one header
```

```
## A level-two header
```

```
### A level-three header
```

```
# A level-one header {#11-1}
```

```
## A level-two header {#12-1}
```

```
### A level-three header {#13-1}
```

```
# A level-one header {#11-2}
```

```
## A level-two header {#12-2}
```

```
### A level-three header {#13-2}
```

Block quotations

This is block quote. This paragraph has two lines

```
> This is block quote. This paragraph has two lines
```

This is a block quote.

A block quote within a block quote.

```
> This is a block quote.
```

```
>
```

```
> > A block quote within a block quote.
```

Italic

```
*Italic*
```

**Bold**

```
**Bold**
```

Naver link

```
[Naver link](https://www.naver.com/)
```

이미지를 직접 삽입하고 가운데 정렬합니다.

```
<center>
```

```
! [ (images/rmarkdown/000002.png) {width="200"}
```

```
</center>
```

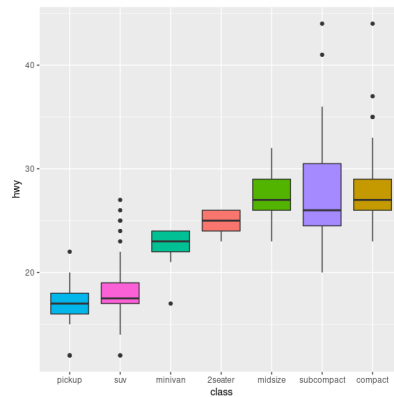


Figure 3.3: 자동차 모델에 따른 고속도로 연비 분포

1. 첫 번째
2. 두 번째
3. 세 번째

- 아이템 1
- 아이템 2
- 아이템 3
  - 아이템 3-1
  - 아이템 3-2

- 1.
- 2.
- 3.

- 1
- 2
- 3
  - 3-1
  - 3-2

참고로 소스코드 그대로 표현하기 위해서는 ~~~ 를 사용합니다.

## 3.4 스타일

아래와 같이 코드청크를 이용해서 css 코드를 삽입하고 해당되는 class 또는 id에 해당하는 내용에 스타일을 적용할 수 있습니다.

Table 3.1: A knitr kable.

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2

```

{css, echo=F}
#header1 {
color: red;
}

```

소스코드

```

<div id='header1'>

</div>

```

### 3.5 테이블

kable 함수를 이용하여 Rmarkdown 문서에 포함되는 표를 원하는 방향으로 작성할 수 있습니다. mtcars는 데이터프레임 형식의 데이터입니다.

```

knitr::kable(
  mtcars[1:5, ],
  caption = "A knitr kable."
)

```

### 3.6 YAML 헤더

Rmarkdown 파일에서 YAML의 가장 중요한 기능은 output 포맷을 지정하는 것이며 title, author, date, 등을 설정할수도 있습니다.

```

---
layout: page
title: "R "
subtitle: "Rmarkdown "
output:
  html_document:
    css: style.css

```



```

includes:
  in_header: header.html
  after_body: footer.html
theme: default
toc: yes
toc_float: true
highlight: tango
code_folding: show
number_sections: TRUE
mainfont: NanumGothic
---
```

## 3.7 Output format

주요 문서 포맷으로 다음과 같은 몇 가지가 있습니다. 상세한 내용은 Rmarkdown output format을 참고하시기 바랍니다.

- `html_document` - HTML document w/ Bootstrap CSS
- `pdf_document` - PDF document (via LaTeX template)
- `word_document` - Microsoft Word document (docx)
- `ioslides_presentation` - HTML presentation with ioslides
- `beamer_presentation` - PDF presentation with LaTeX Beamer
- `powerpoint_presentation` - PowerPoint presentation

### Exercises

“KRIBBR2022-Lecture2” 라는 이름의 다음과 같은 형태의 Rmarkdown 문서를 만들고 이 번 강의의 실습 코드 및 설명, 질문, 코멘트 등을 적어 보시기 바랍니다.

---

이 저작물은 크리에이티브 커먼즈 저작자표시-비영리-변경금지 4.0 국제 라이선스에 따라 이용할 수 있습니다.



## Chapter 4

# R programming

### 4.1 Console calculator

콘솔에서 바로 계산을 수행할 수 있습니다. 참고로 이전에 수행한 명령은 콘솔에 커서가 있는 상태에서 위 아래 화살표를 누르면 볼 수 있고 엔터를 눌러 재사용 할 수 있습니다. ;을 사용하면 두 개의 명령을 동시에 수행할 수 있습니다.

$$2 + 2$$

$$((2 - 1)^2 + (1 - 3)^2)^{1/2}$$

```
2 + 2
((2 - 1)^2 + (1 - 3)^2)^(1/2)
2 + 2; 2 - 2
```

#### Exercises

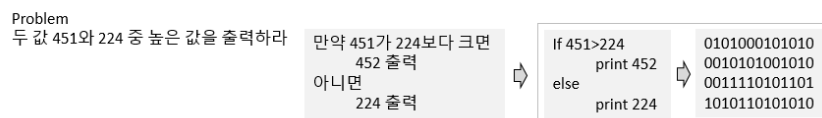
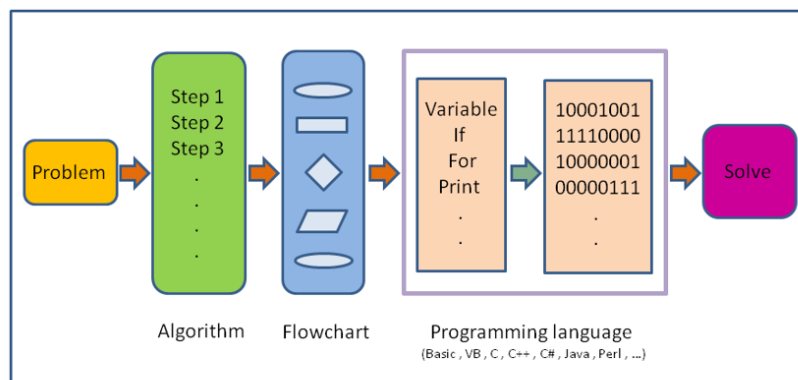
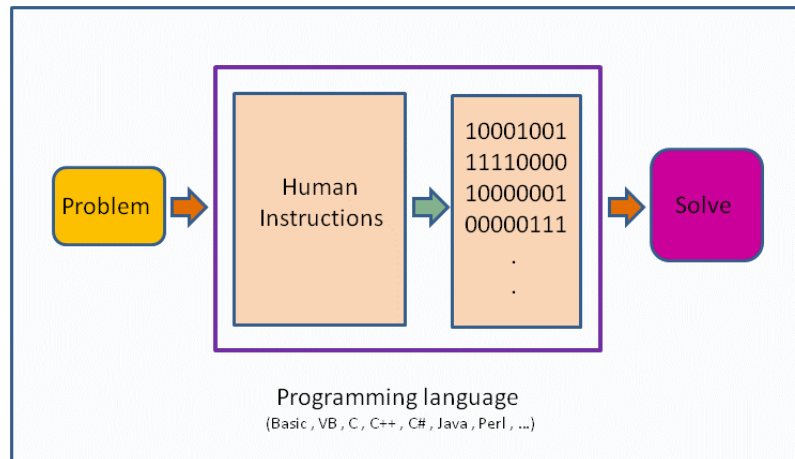
다음 공식들을 계산하는 R 코드를 작성하시오

$$\sqrt{(4 + 3)(2 + 1)}$$

$$2^3 + 3^2$$

$$\frac{0.25 - 0.2}{\sqrt{0.2(1 - 0.2)/100}}$$

## 4.2 What is a programming language



R은 programming language로서 다른 프로그래밍 언어와 같이 몇 가지 공통적 개념을 가집니다 ( , , , , )

### 4.2.1 Terminology

- Session: R 언어 실행 환경
- Console: 명령어 입력하는 창
- Code: R 프로그래밍 변수/제어문 모음
- Object: 변수, 함수 등 프로그래밍에서 사용되는 모든 객체 (Data structure)
  - array: 1D, 2D, 3D, ... 형태 값들의 모임
  - vector: 1차원 형태 값들의 모임 combine function c() EX: c(6, 11,

- 13, 31, 90, 92)
  - matrix: 2차원 형태 값들의 모임 (같은 타입 값으로 구성)
  - data frame: 2차원 형태 값들의 모임 (다른 타입 값 구성 가능)
  - list: vector, matrix, data.frame 및 list 등 다양한 객체를 원소로 가집
- function: 특정 기능 수행, [함수이름, 입력값 (arguments), 출력값 (return)]  
으로 구성
- Data (value): 값 - 자료형 (Data type)
  - Integers
  - doubles/numerics
  - logicals
  - characters
  - factor: 범주형
- Conditionals (조건, 제어):
  - if, ==, & (AND), | (OR) Ex: (2 + 1 == 3) & (2 + 1 == 4)
  - for, while: 반복 수

## 4.3 Data and variables

### 4.3.1 Data

일반적으로 데이터의 의미는 사실을 나타내는 수치입니다.

- 맥도너 정보경제학 (1963)
  - 지혜 (wisdom) : 패턴화된 지식
  - 지식 (knowledge) : 가치있는 정보
  - 정보 (information) : 의미있는 데이터
  - 데이터 (data) : 단순한 사실의 나열

```
library(UsingR)
exec.pay
?exec.pay
```

데이터는 속성에 따라서 다음과 같이 분류할 수 있습니다.

- 범주형 - 질적 데이터, 숫자로 나타낼 수 있으나 의미 없음
  - 명목형 (Nominal) - 사람 이름
  - 순서형 (Ordinal) - 달리기 도착 순서
- 수치형 - 숫자로 나타내며 데이터 속성을 그대로 지님
  - 구간형 (Interval) - 선수1, 선수2 종점통과 시간
  - 비율형 (Ratio) - 출발시간 기준 종점 통과 시간

이름	등수	도착	걸린시간
둘리	1	13:12	1:12
희동	5	14:30	2:30
길동	2	13:30	1:30
철수	4	14:00	2:00
영희	3	13:50	1:50

- Data type in R
  - Numeric (수치형)
    - \* Discrete (이산형) data - 카운트, 횟수
    - \* Continuous (연속형) data - 키, 몸무게, Cannot be shared
    - \* Date and time
  - Factors (범주형)
    - \* Categories to group the data
    - \* Character data - Identifiers (범주형)

### 4.3.2 Variables

변수는 데이터를 저장하는 공간으로 이해할 수 있습니다.

- Assignment operator ( <- OR = )
  - Valid object name <- value
  - 단축키: Alt + - (the minus sign)
- 내장 변수 Built-in variables

```
x <- 2
y <- x^2 - 2*x + 1
y
x <- "two"
some_data <- 9.8
pi
```

- 변수이름 작명법
  - Characters (letters), numbers, “\_”, “.”
  - A and a are different symbols
  - Names are effectively unlimited in length

```
i_use_snake_case <- 1
otherPeopleUseCamelCase <- 2
some.people.use.periods <- 3
And_aFew.People.RENOUNCEconvention <- 4
```

## 4.4 Object (Data structure)

변수, 함수 등 프로그래밍에서 사용되는 모든 개체를 말합니다.

### 4.4.1 vector

vector는 R의 기본 데이터 구조입니다. numeric vector, logical vector, character vector 등 저장되는 값의 타입에 따라 크게 세가지로 나눌 수 있습니다. `class()` 함수를 이용해서 값의 타입을 알아낼 수 있습니다. Combine function인 `c()`를 활용하여 만들며 값을 순차적으로 붙여갈 수 있습니다. 다음과 같은 Univariate (단변량, Single variable)을 표현할 때 사용됩니다.

$$x_1, x_2, \dots, x_n$$

```
x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
class(x)
y <- c("X1", "Y2", "X3", "Y4")
class(y)
z <- c(T, F, F, T)
class(z)
```

#### 4.4.1.1 numeric

numeric 형식의 벡터는 다음과 같은 다양한 편의 함수들을 사용해서 만들 수 있습니다.

```
1:5
seq(1,5, by=1)
seq(0, 100, by=10)
seq(0, 100, length.out=11)
?seq

rep(5, times=10)
rep(1:3, times=4)
rep(1:3, each=3)
```

#### Exercises

odds라는 이름의 변수에 1부터 100까지의 홀수만을 저장하시오 (`seq()` 함수 사용)

인덱싱은 배열형 (vector, matrix 등) 데이터의 일부 데이터를 참조할 때 사용하는 방법입니다. [와 ]를 사용하며 위치를 나타내는 수로 참조합니다.

```
x[1]
x[1:3]
i <- 1:3
x[i]
```

```
x[c(1,2,4)]
y[3]
```

또한 해당 위치의 이름으로 참조하기도 합니다.

```
head(precip)
precip[1]
precip[2:10]
precip[c(1,3,5)]
precip[-1]
precip["Seattle Tacoma"]
precip[c("Seattle Tacoma", "Portland")]
precip[2] <- 10
```

참고로 vector 들은 다음과 같은 builtin 함수들을 사용해서 해당 변수의 attribute를 알아낼 수 있습니다. attribute에는 원소 이름, 타입, 길이 등 vector형 변수가 가질 수 있는 특성을 말합니다.

```
head(precip)
class(precip)
length(precip)
names(precip)

test_scores <- c(100, 90, 80)
names(test_scores) <- c("Alice", "Bob", "Shirley")
test_scores
```

#### 4.4.1.2 logical

Logical 벡터는 True 또는 False를 원소로 갖는 벡터 입니다. 앞글자가 대문자로 시작하는 것을 기억하시고 T 또는 F와 같이 한 문자로 표현할 수도 있습니다. 특정 조건에 대한 판단 결과를 반환할 경우에도 논리값을 사용합니다. 이 경우 조건을 판단 후 인덱싱 방법으로 (which, any, all 등 사용) 해당 값들을 뽑아내기도 합니다. 또한 활용이 많은 sample 함수의 사용법을 익혀둡니다.

```
x <- 1:20
x > 13
temp <- x > 13
class(temp)

ages <- c(66, 57, 60, 41, 6, 85, 48, 34, 61, 12)
ages < 30
which(ages < 30)
i <- which(ages < 30)
ages[i]
any(ages < 30)
all(ages < 30)
```



```
random_number <- sample(c(1:10), 2)
```

**Exercises** 1. 1부터 100까지의 수를 `evens`이라는 이름의 변수에 저장하고 이 중 짝수만을 뽑아내서 출력하시오 (`which()` 함수 사용)

2. `sample` 함수를 사용하여 앞서 `odds`와 `evens` 변수에서 랜덤하게 1개씩의 샘플을 뽑아서 `mynumbers`에 저장하시오

3. 어떤 짝수가 뽑혔는지 찾아서 출력하시오 (`which`와 인덱싱 사용)

#### 4.4.1.3 character

Character(문자형) 벡터의 경우 문자열을 다루는데 자주 쓰이는 `paste()` 함수의 사용법을 알아두면 편리합니다. `paste()` 함수는 서로 다른 문자열을 붙이는데 주로 사용됩니다. 참고로 문자열을 나누는 함수는 `strsplit()` 입니다. `paste()`에서 붙이는 문자 사이에 들어가는 문자를 지정하는 파라미터는 `sep` 이고 `strsplit()` 함수에서 자르는 기준이 되는 문자는 `split` 파라미터로 지정해 줍니다 (?split 또는 ?paste 확인).

```
paste("X", "Y", "Z", sep="_")
paste(c("Four", "The"), c("Score", "quick"), c("and", "fox"), sep="_")
paste("X", 1:5, sep="")
paste(c("X", "Y"), 1:10, sep="")

x <- c("X1", "Y2", "X3", "Y4", "X5")
paste(x[1], x[2])
paste(x[1], x[2], sep="")
paste(x, collapse="_")

strsplit("XYZ", split="")
sort(c("B", "C", "A", "D"))
```

#### Exercises

1. `m`이라는 변수에 “Capital of South Korea is Seoul” 문자열을 저장하고 “Capital of South Korea”를 따로 뽑아내 `m2`에 저장하시오 (`substr()` 사용)
2. `LETTERS` 내장함수에서 랜덤하게 10개의 문자를 뽑아내 `myletters` 변수에 저장하고 이들을 연결하여 (`paste` 사용) 하나의 문장(String)을 만드시오
3. `myletters` 변수의 문자들을 알파벳 순서대로 정렬하고 (`sort` 사용) 이들을 연결하여 하나의 문장 (String)을 만드시오

#### 4.4.1.4 factor

Factor형은 범주형데이터를 저장하기 위한 object이며 R 언어에서 특별히 만들어져 사용되고 있습니다. `factor()` 함수를 이용해 생성하며 생성된 객체는 다음과 같이 `level`이라는 범주를 나타내는 특성값을 가지고 있습니다.

예를 들어 어린이 5명이 각각 빨강, 파랑, 노랑, 빨강, 파랑 색종이를 들고 있을 때 색의 종류를 나타내는 값들은 빨강, 파랑, 노랑 입니다. 다섯 명의 아이들이 어떤 색의 색종이를 들고 있는지와는 상관없이 세 가지 범주의 값을 가지는 것 입니다.

```
x <- c("Red", "Blue", "Yellow", "Red", "Blue")
y <- factor(x)
y
```

새로운 범주의 데이터를 추가할 경우 다음과 같이 해당되는 level을 먼저 추가하고 값을 저장해야 합니다.

```
levels(y)
y[1] <- "Gold"
y

levels(y) <- c(levels(y), "Gold")
levels(y)
y
y[1] <- "Gold"
y
```

`factor`는 기본적으로 level에 표시된 순서가 위치 (정렬) 순서입니다. 이를 바꾸기 위해서는 다음과 같이 `levels` 함수를 이용해서 순서를 바꿀 수 있습니다.

```
library(MASS)
str(Cars93)
x <- Cars93$Origin
plot(x)
levels(x) <- c("non-USA", "USA")
levels(x)
plot(x)
```

∴ rmdnote Exercises

UUU } Phe UUC } UUA } Leu UUG }	UCU } UCC } Ser UCA } UCG }	UAU } Tyr UAC } UAA } Stop UAG }	UGU } Cys UGC } UGA } Stop UGG } Trp
CUU } CUC } Leu CUA } CUG }	CCU } CCC } Pro CCA } CCG }	CAU } His CAC } CAA } Gln CAG }	CGU } CGC } Arg CGA } CGG }
AUU } Ile AUC } AUA } AUG } Met	ACU } ACC } Thr ACA } ACG }	AAU } Asn AAC } AAA } Lys AAG }	AGU } Ser AGC } AGA } Arg AGG }
GUU } GUC } Val GUA } GUG }	GCU } GCC } Ala GCA } GCG }	GAU } Asp GAC } GAA } Glu GAG }	GGU } GGC } Gly GGA } GGG }

1. 아미노산 Phe, Leu, Ser 를 값으로 갖는 범주형 변수 (factor)를 생성하시오 2. 각 아미노산과 해당 아미노산을 코딩하는 nucleotide triplets (codon)을 어떤 형태의 변수로 저장할 수 있을지 고민해 보시오

:::

#### 4.4.1.5 Missing values

특정 값이 “Not available” 이거나 “Missing value” 일 경우 벡터의 해당 원소 자리에 데이터의 이상을 알리기 위해 NA를 사용합니다. 따라서 일반적인 연산에서 NA가 포함되어 있는 경우 데이터의 불완전성을 알리기 위해 연산의 결과는 NA가 됩니다. `is.na()` 함수는 해당 변수에 NA 값이 있는지를 검사해주는 함수이며 R에는 이 외에도 다음과 같은 특수 값들이 사용되고 있습니다.

- NA: Not available, The value is missing
- NULL: a reserved value
- NaN: Not a number (0/0)
- Inf: (1/0)

```
hip_cost <- c(10500, 45000, 74100, NA, 83500)
sum(hip_cost)
sum(hip_cost, na.rm=TRUE)
?sum
```

#### 4.4.1.6 Useful functions

다음은 벡터형 변수와 같이 쓰이는 유용한 함수들입니다.

```
z <- sample(1:10, 100, T)
head(z)
sort(z)
```

```
order(z)
table(z)
p <- z/sum(z)
round(p, digits=1)
```

is 함수를 사용하여 데이터 타입이 사용자가 의도한 타입과 맞는지 검사할 수 있습니다. 콘솔창에서 is.를 타이핑한 후 잠시 기다리면 다양한 is 함수를 볼 수 있습니다.

```
is.na(1)
is.numeric(1)
is.logical(TRUE)
is.data.frame("A")
is.character("A")
```

as 함수는 데이터 타입을 변환해주는 함수입니다.

```
digits <- runif(10)*10
class(digits)
digits_int <- as.integer(digits)
class(digits_int)
digits_char <- as.character(digits_int)
class(digits_char)
digits_num <- as.numeric(digits_char)
class(digits_num)
```

## 4.4.2 matrix

매트릭스는 2차원 행렬로 같은 형식의 데이터 값 (numeric, character, logical) 으로부터 채워진 행렬을 말합니다. 매트릭스를 만드는 방법은 아래와 같으며 nrow 와 ncol 파라미터에 행과 열의 수를 넣고 각 셀에 들어갈 값은 가장 앞에 위치한 data 파라미터에 넣어 줍니다 (?matrix로 파라미터 이름 확인). 매트릭스 인덱싱은 매트릭스 안의 값을 저장하거나 참조할때 (빠올때) 사용하는 방법입니다. 매트릭스 변수이름 바로 뒤에 대괄호를 이용해서 제어를 하며 대괄호 안에 콤마로 구분된 앞쪽은 row, 뒷쪽은 column 인덱스를 나타냅니다.

```
mymat <- matrix(0, nrow=100, ncol=3) # 1
mymat[,1] <- 1:100 # 2
mymat[,2] <- seq(1,200,2) # 3
mymat[,3] <- seq(2,200,2) # 4
```

매트릭스의 row나 column에 이름이 주어져 있을 경우 이름을 따옴표(")로 묶은 후 참조가 가능합니다. row나 column의 이름은 rownames() 또는 colnames()로 생성하거나 변경할 수 있습니다. row나 column의 개수는 nrow() 또는 ncol() 함수를 사용합니다.

```
colnames(mymat)
colnames(mymat) <- c("A", "B", "C")
colnames(mymat)
colnames(mymat)[2] <- "D"
colnames(mymat)
rownames(mymat) <- paste("No", 1:nrow(mymat), sep="")
rownames(mymat)
```

여러 row나 column을 참조할 경우 아래와 같이 combine 함수를 사용하여 묶어줘야 하며 스칼라값을 (임의의 숫자 하나) 더하거나 뺄 경우 vector / matrix 연산을 기본으로 수행합니다.

```
mymat[c(2,3,4,5),2] # 5
mymat-1 # 6
mysub <- mymat[,2] - mymat[,1] #7
sum(mysub) #8
sum(mysub^2) #8
```

### Exercises

1. score 라는 변수에 1부터 100까지 중 랜덤하게 선택된 20개의 수로 10 x 2 matrix를 만드시오 (sample() 사용)
2. score의 row 이름을 문자형으로 Name1, Name2, ..., Name10으로 지정하시오 (paste() 사용)
3. score의 column 이름을 문자형으로 math와 eng로 지정하시오
4. 이 matrix의 첫번째 컬럼과 두 번째 컬럼의 수를 각각 더한 후 total\_score라는 변수에 저장하시오
5. total\_score의 오름차순 순서를 나타내는 인덱스 (order() 함수 사용)를 o라는 변수에 저장하시오
6. score를 o순서로 재배치하고 score\_ordered 변수에 저장하시오

### 4.4.3 data.frame

데이터프레임은 형태는 매트릭스와 같으나 컬럼 하나가 하나의 vector형 변수로서 각 변수들이 다른 모드의 값을 저장할 수 있다는 차이가 있습니다. \$ 기호를 이용하여 각 구성 변수를 참조할 수 있습니다. 컬럼 한 줄이 하나의 변수 이므로 새로운 변수도 컬럼 형태로 붙여 넣을 수 있습니다. 즉, 각 row는 샘플을 나타내고 각 column은 변수를 나타내며 각 변수들이 갖는 샘플의 개수 (row의 길이, vector 의 길이)는 같아야 합니다. R 기반의 데이터 분석에서는 가장 선호되는 데이터 타입이라고 볼 수 있습니다.

```
## data.frame
ids <- 1:10
ids
idnames <- paste("Name", ids, sep="")
idnames
students <- data.frame(ids, idnames)
```

```

students
class(students$ids)
class(students$idnames)
students$idnames
str(students)

students <- data.frame(ids, idnames, stringsAsFactors = F)
class(students$idnames)
students$idnames
students[1,]
str(students)

```

데이터프레임에서는 \$를 사용하여 변수 이름으로 인덱싱이 가능합니다.

```

## data frame indexing
students$ids
students[,1]
students[, "ids"]

```

## Exercises

1. math라는 변수에 1부터 100까지 중 랜덤하게 선택된 10개의 수를 넣으시오
2. eng라는 변수에 1부터 100까지 중 랜덤하게 선택된 10개의 수를 넣으시오
3. students라는 변수에 문자형으로 Name1, Name2, ..., Name10으로 지정하시오 (paste() 사용)
4. math와 eng라는 벡터에 저장된 값들의 이름을 students 변수에 저장된 이름으로 지정하시오
5. math와 eng 벡터를 갖는 score 라는 data.frame을 만드시오
6. math와 eng 변수를 지우시오 (rm()사용)
7. score data frame의 math와 eng를 각각 더한 후 total\_score라는 변수에 저장 하시오

### 4.4.4 list

리스트는 변수들의 모임이라는 점에서 데이터프레임과 같으나 구성 변수들의 길이가 모두 같아야 하는 데이터프레임과는 달리 다른 길이의 변수를 모아둘 수 있는 점이 다릅니다. 즉, R언어에서 두 변수를 담을 수 있는 데이터 타입은 list와 data frame 두 종류가 있는데 list 변수 타입은 vector 형태의 여러개의 element를 가질 수 있으며 각 vector 길이가 모두 달라도 됩니다. list의 인덱싱에서 [ ]는 리스트를 반환하고 [[ ]]는 vector element들을 반환합니다.

### Lists

```
l <- list(x = 1:5, y = c('a', 'b'))
```

A list is a collection of elements which can be of different types.

<code>l[[2]]</code>	<code>l[1]</code>	<code>l\$x</code>	<code>l['y']</code>
Second element of l.	New list with only the first element.	Element named x.	New list with only element named y.

```
## list
parent_names <- c("Fred", "Mary")
number_of_children <- 2
child_ages <- c(4, 7, 9)
data.frame(parent_names, number_of_children, child_ages)
lst <- list(parent_names, number_of_children, child_ages)
lst[1]
lst[[1]]
class(lst[1])
class(lst[[1]])
lst[[1]][1]
lst[[1]][c(1,2)]
```

Also see the **dplyr** package.

### Data Frames

```
df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))
```

A special case of a list where all elements are the same length.

x	y
1	a
2	b
3	c

**Matrix subsetting**

`df[, 2]`

`df[2, ]`

`df[2, 2]`

**List subsetting**

`df$x`

`df[[2]]`

*Understanding a data frame*

`View(df)` See the full data frame.

`head(df)` See the first 6 rows.

**nrow(df)**  
Number of rows.

**ncol(df)**  
Number of columns.

**dim(df)**  
Number of columns and rows.

**cbind** - Bind columns.

**rbind** - Bind rows.

## Exercises

1. 위 아미노산 예제에서 Phe, Leu, Ser 각각의 코돈을 원소로 갖는 세 개의 vec-

tor 변수들을 만들고 이를 `aalist` 라는 이름의 하나의 리스트 변수로 만드시오

2. `aalist` 리스트를 `data.frame` 형식의 `aadf` 변수로 만드시오 (데이터 구조를 바꾸어 저장 가능)

## 4.5 Functions

함수(Function)란 사용자가 원하는 기능을 수행하는 코드의 모음으로서 반복적으로 쉽게 사용할 수 있도록 만들어 놓은 코드입니다.

### 4.5.1 A script in R

함수의 개념을 배우기 전에 스크립트를 활용한 명령어 수행을 알아보겠습니다. R 프로그래밍을 통해서 사용자가 원하는 기능을 수행하는 방법은 다음과 같이 스크립트를 만들어서 실행하는 것입니다. 일반적으로 R을 이용한 스크립트 명령을 어떻게 실행하는지 알아보겠습니다. 다음 예제는 입력 값들의 평균을 계산해서 출력해 주는 스크립트 명령입니다. R base 패키지에서 기본으로 제공되는 `mean()`이라는 함수가 있지만 사용하지 않고 `sum()`과 `length()` 함수를 사용했습니다.

```
numbers <- c(0.452, 1.474, 0.22, 0.545, 1.205, 3.55)
cat("Input numbers are", numbers, "\n")
numbers_mean <- sum(numbers)/length(numbers)
out <- paste("The average is ", numbers_mean, ".\n", sep="")
cat(out)
```

상황에 따라 다르긴 하지만 보통 위 스크립트를 실행할 때 R 파일을 하나 만들고 `source()`라는 함수를 사용해서 파일 전체를 한번에 읽어들이고 실행을 시킵니다. 위 코드를 `myscript.R` 이라는 새로운 R 파일을 하나 만들고 저장 후 다음과 같이 실행할 수 있습니다. 참고로 위 파일은 현재 Working directory와 같은 위치에 저장해야 합니다.

```
source("myscript.R")
```

그러나 위와 같은 식으로 실행할 경우 다음 몇 가지 문제가 있습니다. 하나는 입력 값이 바뀔 때마다 파일을 열어 바뀐 값을 저장해 줄 필요가 있습니다. 결과 값에 대해서 다른 처리를 하고 싶을 경우 또한 파일을 직접 수정해 주어야 합니다. 또한 모든 변수들이 전역변수로 사용되어 코드가 복잡해질 경우 변수간 간섭이 생길 가능성이 높습니다.

### 4.5.2 Build a function

함수는 특정 데이터를 입력으로 받아 원하는 기능을 수행한 후 결과 데이터를 반환하는 구조를 가집니다. 함수는 일반적으로 다음과 같은 포맷으로 구현할 수 있습니다.



```
my_function_name <- function(parameter1, parameter2, ... ){
  ##any statements
  return(object)
}
```

예를 들어 다음과 같은 my\_sine 함수를 만들 수 있으며 parameter (매개변수)는 x이고 y는 반환값을 저장하는 지역변수입니다.

```
my_sine <- function(x){
  y <- sin(x)
  return(y)
}
```

만들어진 함수는 다음과 같이 사용할 수 있습니다. 만들어진 함수는 처음에 한 번 실행해 주어 실행중인 R session에 등록한 후 사용할 수 있습니다. 여기서 함수로 전달되는 값 pi는 argument (전달인자) 라고 합니다. 전달인자는 함수에서 정의된 매개변수의 갯수와 같은 수의 전달인자를 입력해 주어야 합니다. 참고로 parameter와 argument는 많은 사람들이 혼동하는 단어입니다. 본 예에서 my\_sine함수의 괄호 안에 있는 변수 x는 parameter이고 x에 들어가는 값인 pi 나 90은 argument입니다.

```
my_sine(pi)
my_sine(90)
sin(90)
```

- Terminology
  - function name: my\_sine
  - parameter: x
  - argument: pi
  - return value: y

이제 위 스크립트 (myscript.R) 에서 사용된 코드를 함수로 바꿔봅니다. numbers (전달인자)를 받는 매개변수를 x로 하고 함수 이름은 mymean 이고 평균값 (numbers\_mean)을 반환하는 함수입니다.

```
numbers <- c(0.452, 1.474, 0.22, 0.545, 1.205, 3.55)

mymean <- function(x){
  cat("Input numbers are", x, "\n")
  numbers_mean <- sum(x)/length(x)
  out <- paste("The average is ", numbers_mean, ".\n", sep="")
  cat(out)
  return(numbers_mean)
}

retval <- mymean(numbers)
cat(retval)
```

myscript.R이라는 파일을 열고 작성된 스크립트에 더해서 아래처럼 함수 코드를 만들 경우 source() 함수로 함수를 세션으로 읽어오고 바로 사용할 수 있습니다. 위와 같이 함수를 만들 경우 입력 값을 언제든지 바꿔서 사용할 수 있고 반환값에 대한 추가적인 연산도 쉽게 수행 할 수 있습니다.

```
new_values <- c(1:10)
retval <- mymean(new_values)
retval
```

### Exercises

1. 변수 x에 1, 3, 5, 7, 9를, 변수 y에 2, 4, 6, 8, 10을 저장하는 코드를 작성하시오
2. x와 y를 더한 값을 z에 저장하는 코드를 작성하시오
3. mysum 이라는 이름의 함수를 작성하되 두 변수를 입력으로 받아 더한 후 결과를 반환하는 코드를 작성하시오
4. mymean 이라는 이름의 함수를 작성하되 두 변수를 입력으로 받아 평균을 구한 후 결과를 반환하는 코드를 작성하시오

### Exercises

- 1) mysd라는 이름의 (표본)표준편차를 구하는 함수를 myscript.R 파일에 구현하시오 (sd() 함수 사용하지 않고, 다음 표준편차 공식 이용)

$$\sigma = \sqrt{\frac{\sum (x - \text{mean}(x))^2}{\text{length}(x) - 1}}$$

코드는 아래와 같음

```
mysd <- function(x){
  numbers_sd <- sqrt(sum((x - mymean(x))^2)/(length(x)-1))
  return(numbers_sd)
}
```

- 2) 1부터 100까지의 값을 x에 저장하고 mysd 함수를 사용해서 표준편차를 구하시오
- 3) 앞서 작성한 mymean 함수와 mysd 함수를 같이 사용하여 x를 표준화 하고 z로 저장하시오. 표준화 공식은 다음과 같음

$$z = \frac{x - \text{mean}(x)}{\text{sd}(x)}$$

- 4) x 와 z 변수를 원소로 갖는 y라는 이름의 data.frame을 생성하시오

### 4.5.3 local and global variables

함수를 사용함에 따라서 함수 안에서 사용되는 변수와 함수 밖에서 사용되는 변수들의 경우를 명확히 이해할 필요가 있습니다. 다음 코드를 보면 전역변수  $x$ ,  $y$ 는 지역변수  $x$ ,  $y$ 와 독립적으로 사용되고 있습니다.

```
my_half <- function(x){
  y <- x/z
  cat("local variable x:", x, "\n")
  cat("local variable y:", y, "\n")
  cat("global variable z:", z, "\n")
  return(y)
}
y <- 100
x <- 20
z <- 30
cat("Global variable x:", x, "\n")
cat("Global variable y:", y, "\n")
cat("Global variable z:", z, "\n")
my_half(5)

my_half <- function(x, z){
  y <- x/z
  cat("local variable x:", x, "\n")
  cat("local variable y:", y, "\n")
  cat("local variable z:", z, "\n")
  return(y)
}

my_half(5, 10)
```

$\log$ ,  $\sin$  등의 함수들은 Built-in function으로 같은 이름의 함수를 만들지 않도록 주의합니다.

```
x <- pi
sin(x)
sqrt(x)
log(x)
log(x, 10)
x <- c(10, 20, 30)
x + x
mean(x)
sum(x)/length(x)
```

### 4.5.4 Vectorized functions

초기에 R이 다른 프로그래밍 언어에 비해서 경쟁력을 갖는 이유 중 하나가 바로 이 벡터 연산 기능 이였습니다. `vector` 변수에 들어있는 각 원소들에 대해서 특정 함수나 연산을 적용하고 싶을 경우 전통 방식의 C나 Java등의 언어에서는 원소의 개수만큼 반복문을 돌면서 원하는 작업을 수행 했습니다. 그러나 R의 벡터 연산 기능은 별도의 반복문 없이 `vector` 안에 있는 원소들에 대한 함수 실행 또는 연산을 수행할 수 있습니다.

```
x <- c(10, 20, 30)
x + x
sqrt(x)
sin(x)
log(x)
x-mean(x)

length(x)
test_scores <- c(Alice = 87, Bob = 72, James= 99)
names(test_scores)
```

#### Exercises

다음은 한 다이어트 프로그램의 수행 전 후의 다섯 명의 몸무게이다.

Before	78	72	78	79	105
after	67	65	79	70	93

- 1) 각각을 before 와 after 이름의 변수에 저장 후 몸무게 값의 변화량을 계산하여 `diff` 라는 변수에 저장하시오
- 2) `diff`에 저장된 값들의 합, 평균, 표준편차를 구하시오

#### Exercises

다음 네 학생이 있으며 “John”, “James”, “Sara”, “Lilly” 각 나이는 21, 55, 23, 53 이다. `ages` 라는 변수를 생성하고 각 나이를 저장한 후 `who`라는 이름의 함수를 만들어서 50살 이상인 사람의 이름을 출력하는 함수를 만드시오.

- `ages`라는 변수에 나이 저장, `c()` 함수 이용, `vector` 형태 저장
- `names()` 함수 이용해서 각 `ages` 벡터의 각 요소에 이름 붙이기
- `which()` 함수 사용해서 나이가 50보다 큰 인덱스 찾고 해당 인덱스 값들을 `idx`에 저장
- `ages`에서 `idx`에 해당하는 인덱스를 갖는 값을 `sel_ages`에 저장
- `names()` 함수를 이용해서 `sel_ages`의 이름을 `sel_names`에 저장
- 위 설명을 참고해서 `input`이라는 파라미터를 갖고 `sel_names`라는 50살 이상인 사람의 이름을 반환하는 `who50`이라는 이름의 함수 만들기
- `who50` 함수의 사용법은 `who50(ages)` 임

## 4.6 Flow control

### 4.6.1 if statements

R에서의 제어문의 사용은 다른 프로그래밍 언어와 거의 유사합니다. 먼저 `if` 는 다음과 같은 형식으로 사용되며 () 안에 특정 조건 판단을 위한 표현이 들어갑니다.

```
if(condition){
  expr_1
}else{
  expr_2
}
```

특히 `condition`은 하나의 원소에 대한 조건 판단문으로 T 또는 F 값 하나만을 반환하는 문장이어야 합니다. 위 코드는 만약 `condition` 조건이 True 이면 `expr_1`를 실행하고 False이면 `expr_2`를 실행하라는 명령입니다. `condition` 안에서 사용되는 비교 연산자들은 다음과 같습니다.

<code>!x</code>	logical negation, NOT x
<code>x &amp; y</code>	elementwise logical AND
<code>x &amp;&amp; y</code>	vector logical AND
<code>x   y</code>	elementwise logical OR
<code>x    y</code>	vector logical OR
<code>xor(x, y)</code>	elementwise exclusive OR
<code>&lt;</code>	Less than, binary
<code>&gt;</code>	Greater than, binary
<code>==</code>	Equal to, binary
<code>&gt;=</code>	Greater than or equal to, binary
<code>&lt;=</code>	Less than or equal to, binary

```
x <- 2
if(x%%2 == 1){
  cat("Odd")
}else{
  cat("Even")
}

x <- 5
if(x > 0 & x < 4){
  print("Positive number less than four")
}

if(x > 0) print("Positive number")

x <- -5
if(x > 0){
  print("Non-negative number")
} else if(x <= 0 & x > -5){
  print("Negative number greater than -5")
} else {
  print("Negative number less than -5")
}
```

```
if(x > 0)
  print("Non-negative number")
else
  print("Negative number")
```

### 4.6.2 ifelse statements

if는 하나의 조건만 비교하는데 사용할 수 있습니다. 그러나 변수에는 여러 값이 벡터형식으로 들어가고 벡터연산을 수행할 경우의 결과도 벡터형식으로 나오지만 if문은 이들을 한 번에 처리하기 어렵습니다. ifelse는 이러한 단점을 보완하여 여러 값을 한번에 처리할 수 있습니다.

```
ifelse (condition, True      , False      )
```

ifelse의 경우 빠르게 원하는 값을 반환할 수 있으나 조건별로 다른 추가적인 명령의 수행은 불가능하다는 단점이 있습니다.

```
x <- c(1:10)
if(x>10){
  cat("Big")
}else{
  cat("Small")
}

ifelse(x>10, "Big", "Small")
```

### Exercises

다음은 median (중간값)을 구하는 공식이며 x의 길이가 (n이) 홀수일 경우와 짝수일 경우에 따라서 다른 공식이 사용된다. 다음 공식과 코드를 이용하여 mymedian 이라는 이름의 함수를 만들고 입력 값들의 중간값을 구해서 반환하는 함수를 만드시오. (%% 나머지 연산, if문 사용, 아래 중간값 코드 참고)

$$\text{median}(X) = \begin{cases} \frac{1}{2}X[\frac{n}{2}] + \frac{1}{2}X[1 + \frac{n}{2}] & \text{if } n \text{ is even} \\ X[\frac{n+1}{2}] & \text{if } n \text{ is odd} \end{cases}$$

```
sorted_x <- sort(x)
#
retval <- sort_x[n/2]/2 + sort_x[1+(n/2)]/2
#
retval <- sort_x[(n+1)/2]
```

### 4.6.3 for, while, repeat

for 문은 반복적으로 특정 코드를 실행하고자 할 때 사용됩니다. 다음과 같은 형식으로 사용할 수 있습니다.

```
for(var in seq){
  expression
}
```

var는 반복을 돌 때마다 바뀌는 변수로 {} 안에서 사용되는 지역 변수입니다. seq는 vector 형식의 변수로 반복을 돌 때마다 순차적으로 var에 저장되는 값들입니다.

```
x <- 1:10
for(i in x){
  cat(i, "\n")
  flush.console()
}

sum_of_i <- 0
for(i in 1:10){
  sum_of_i <- sum_of_i + i
  cat(i, " ", sum_of_i, "\n");flush.console()
}
```

while문도 for문과 같이 반복적으로 특정 코드를 수행하고자 할 때 사용합니다. 사용하는 문법은 다음과 같으며 cond는 True 또는 False 로 반환되는 조건문을 넣고 True 일 경우 계속해서 반복하면서 expressions를 수행하며 이 반복은 cond가 False로 될 때 까지 계속됩니다.

```
while(cond){
  expression
}
```

while문을 사용할 경우 다음과 같이 indicator라 불리우는 변수를 하나 정해서 반복 할 때마다 값이 바뀌도록 해 주어야 합니다. 그렇지 않으면 무한 루프를 돌게 되는 문제가 발생합니다.

```
i <- 10
f <- 1
while(i>1){
  f <- i*f
  i <- i-1
  cat(i, f, "\n")
}
f
factorial(10)
```

repeat 명령은 조건 없이 블럭 안에 있는 코드를 무조건 반복하라는 명령입니다. 따라서 블럭 중간에 멈추기 위한 코드가 필요하고 이 명령이 break 입니다.

```
repeat{
  expressions
  if(cond) break
}

i <- 10
f <- 1
repeat {
  f <- i*f
  i <- i-1
  cat(i, f, "\n")
  if(i<1) break
}
f
factorial(10)
```

#### 4.6.4 Avoiding Loops

R에서는 가능하면 loop문을 사용하지 않는 것이 좋습니다. 이는 다른 언어들 보다 반복문이 느리게 수행된다는 이유 때문이기도 합니다. 그러나 R에서는 반복문을 수행하는 것 보다 훨씬 더 빠르게 반복문을 수행 한 것과 같은 결과를 얻을 수 있는 다양한 방법들이 제공되고 있습니다. 차차 그런 기법들에 대한 학습을 진행하도록 하겠습니다.

```
x <- 1:1E7
sum(x)
system.time(sum(x))

st <- proc.time()
total <- 0
for(i in 1:length(x)){
  total <- total + x[i]
}
ed <- proc.time()
ed-st
```

#### Exercises

1. 다음 네 사람의 이름과 나이를 데이터로 갖는 `users` 변수를 (data.frame) 만드시오

```
user_score <- c(90, 95, 88, 70)
user_names <- c("John", "James", "Sara", "Lilly")
```

2. 각 사람의 점수가 80보다 작으면 : Fail 크면 : Pass를 출력을 하는 코드를 작성하시오. 예를 들어 John의 점수는 80보다 크므로 John 90: Pass 출력 (for, print 함수 이용)



## 4.7 Object Oriented Programming (Advanced)

OOP는 객체지향 프로그래밍 이라고 합니다. OOP를 이용해서 프로그래밍으로 풀고자 하는 문제를 좀 더 명확하게 개념을 수립하고 복잡한 코드를 명료하게 만들 수 있습니다. 그런데 R에서 OOP는 다른 언어보다는 좀 더 어려운 개념적인 이해가 필요합니다. S3, S4, 그리고 `Reference class` 가 있으며 S3, S4는 `Generic function`을 이용하며 다른 언어에서 사용하는 OOP 개념과는 다릅니다. `Reference class`는 다른 언어에서 사용하는 OOP 개념과 유사하며 R6 패키지를 이용해서 사용할 수 있습니다.

---

이 저작물은 크리에이티브 커먼즈 저작자표시-비영리-변경금지 4.0 국제 라이선스에 따라 이용할 수 있습니다.



## Chapter 5

# Data transformation

일반적인 데이터 분석은 데이터 전처리(변환), 가시화, 모델링(통계분석)의 반복적인 수행으로 진행될 수 있습니다. R에서는 `data.frame` 형식의 데이터 타입이 주로 사용되며 (최근 `tibble` 형식) 따라서 `data.frame` 기반의 데이터를 다루기 위한 다양한 함수를 익힐 필요가 있습니다. 이번 강의에서는 `data.frame` 데이터를 읽거나 쓰는 함수들과 함께 데이터 전처리를 (변환) 위한 함수들을 배워보겠습니다.

앞에서 배웠던 데이터를 저장하는 object의 종류를 먼저 간략히 정리해 봅니다.

- **Vectors** - 같은 타입의 데이터를 (Numeric, character, factor, ...) 저장한 오브젝트로 인덱스는 `[, ]` 사용.
- **Lists** - 여러개의 vector를 원소로 가질 수 있으며 각 원소 vector들은 문자나 숫자 어떤 데이터 타입도 가능하고 길이가 달라도 됨. list의 인덱싱에서 `[ ]`는 리스트를 반환하고 `[[ ]]`는 vector를 반환함.
- **Matrices** - 같은 타입의 데이터로 채워진 2차원 행렬이며 인덱스는 `[i, j]` 형태로 `i`는 row, `j`는 column 을 나타냄. 매트릭스의 생성은 `matrix` 명령어를 사용하며 왼쪽부터 column 값을 모두 채우고 다음 컬럼 값을 채워 나가는 것이 기본 설정임. `byrow=T` 를 통해 row를 먼저 채울수도 있음. row와 column 이름은 `rownames`와 `colnames`로 설정이 가능하며 `rbind`와 `cbind`로 두 행렬 또는 행렬과 벡터를 연결할 수 있음 ( `rbind`와 `cbind`의 경우 행렬이 커지면 컴퓨터 리소스 많이 사용함)
- **data.frame** - list와 matrix의 특성을 모두 갖는 오브젝트 타입으로 list와 같이 다른 타입의 vector형 변수 여러개가 컬럼에 붙어서 matrix 형태로 구성됨. 단, list와는 다르게 각 변수의 길이가 (row의 길이) 같아야 함. `$` 기호로 각 변수들을 인덱싱(접근) 할 수 있고 matrix와 같이 `[i, j]` 형태의 인덱싱도 가능.

## 5.1 Reading and writing

파일에 있는 데이터를 R로 읽어들이거나 쓰는 일은 일반적인 데이터 분석 과정에서 필수적일 수 있습니다. 본 강의에서는 일반적으로 사용하는 텍스트 파일과 엑셀파일을 활용하는 방법을 알아보겠습니다.

### 5.1.1 Text file

편의상 데이터를 쓰는 과정을 먼저 살펴봅니다. UsingR 예제에 있는 데이터 중 `batting` 데이터는 2002 야구시즌에 선수들의 정보를 모아둔 데이터입니다. `str` 함수로 데이터 전체적인 구조를 파악한 일부 데이터만을 (홈런 개수와 스트라이크 아웃) 이용해 추가적인 데이터를 생성한 후 별도로 파일에 저장해 보겠습니다.

```
library(UsingR)
data(batting)
str(batting)

mydf <- data.frame(id = batting$playerID,
                  team = batting$teamID,
                  hr = batting$HR,
                  so = batting$SO,
                  soperhr = batting$SO/batting$HR)

head(mydf)
```

재미삼아 홈런과 삼진아웃간의 관계를 한 번 알아보겠습니다.

```
plot(mydf$hr, mydf$so)
mycor <- cor(mydf$hr, mydf$so)
fit <- lm(mydf$so ~ mydf$hr)
plot(mydf$hr, mydf$so); abline(fit); text(50, 170, round(mycor,2))
```

위 데이터를 파일로 저장하기 위해서는 `write.table` 또는 `write.csv` 함수를 사용할 수 있습니다. 패키지에 따라서 다양한 함수들이 제공되고 있지만 위 두 파일은 `utils`라는 R의 기본 패키지에 들어있는 함수들로서 가장 많이 사용되는 함수들입니다. `?write.table` 등으로 도움말을 보시고 특히 함수의 전달값 (Arguments) 들을 (`quote`, `row.names`, `col.names`, `sep`) 익혀두시기 바랍니다.

```
write.table(mydf, file="table_write.txt")
write.table(mydf, file="table_write.txt", quote=F)
write.table(mydf, file="table_write.txt", quote=F, row.names=F)
write.table(mydf, file="table_write.txt", quote=F, row.names=F, sep=",")
write.table(mydf, file="table_write.csv", quote=F, row.names=F, sep=",")
```

대부분의 텍스트 파일은 아래와 같이 csv 또는 txt 파일로 저장하여 메모장으로 열어 확인할 수 있으며 읽어올 때 구분자 (`sep` 파라미터) 나 header를 (`header` 파라미터) 읽을지 등을 옵션으로 지정할 수 있습니다.

```
dat <- read.csv("Dataset_S1_sub.txt")
head(dat)
```

Dataset\_S1\_sub.txt 파일을 열어보면 다음과 같이 header와 “,”로 구분되어 있는 것을 볼 수 있습니다. read.csv 함수의 도움말을 보면 이 함수의 파라미터 head와 sep이 기본값으로 T와 ,로 되어 있는 것을 볼 수 있습니다. read.csv 외에도 read.table, read.delim 등의 함수를 이용해서 텍스트 파일을 읽어올 수 있습니다.

```
str(dat)
```

### Exercises

1. 위 mydf에서 가장 홈런을 많이 순서로 데이터를 정렬하시오
2. 위 정렬된 데이터를 mydf2 로 저장한 후 csv 형태의 baseball.csv 파일로 저장하시오

## 5.1.2 Excel file

텍스트 파일 외에 엑셀파일은 readxl 이라는 R 패키지를 활용하여 읽거나 쓸 수 있습니다. 패키지는 다음과 같은 방법으로 설치할 수 있으며 read\_excel 이라는 함수를 사용해서 데이터를 읽어들이 수 있습니다. 참고로 이 후 강의에서 배우게 될 tidyverse 패키지들 중 readr 패키지를 사용하여 엑셀파일 데이터를 다룰 수도 있습니다.

```
install.packages("readxl")
library(readxl)
```

실습 파일은 형광 세포를 배양하여 형광리더기를 이용해 얻어진 실제 데이터이며 plate\_reader.xls 에서 다운로드 받을 수 있습니다. read\_excel 함수를 이용하여 파일의 내용을 읽어오면 기본 자료형이 tibble 입니다. tibble은 최근 많이 쓰이는 R object로 data.frame과 유사하나 입력값의 type, name, rowname을 임의로 바꿀 수 없다는 점이 다릅니다.

```
dat <- read_excel("plate_reader.xls", sheet=1, skip = 0, col_names=T)
```

엑셀파일에는 두 종류의 ( $OD_{600nm}$ , Fluorescence) 데이터가 저장되어 있습니다. 첫 번째 sheet에는 다음처럼 wide형 데이터가 저장되어 있습니다.

	A	B	C	D	E	F	G	H
1	Plate	Repeat	Well	Type	Time	595nm_kl	Time	EGFP_suli
2	1	1	B01	M	00:00:15.93	0.701	00:01:11.48	67809
3	1	1	B02	M	00:00:16.32	0.752	00:01:11.89	60025
4	1	1	B03	M	00:00:16.69	0.723	00:01:12.30	102745
5	1	1	B04	M	00:00:17.06	0.744	00:01:12.71	99979
6	1	1	B05	M	00:00:17.43	0.706	00:01:13.12	108175
7	1	1	B06	M	00:00:17.80	0.723	00:01:13.53	109575
8	1	1	B07	M	00:00:18.17	0.767	00:01:13.94	76531
9	1	1	B08	M	00:00:18.54	0.777	00:01:14.35	72137
10	1	1	B09	M	00:00:18.91	0.762	00:01:14.76	154549
11	1	1	B10	M	00:00:19.28	0.798	00:01:15.17	128498
12	1	1	B11	M	00:00:19.65	0.793	00:01:15.58	151693
13	1	1	B12	M	00:00:20.02	0.821	00:01:15.99	130526
14	1	1	C01	M	00:00:22.85	0.803	00:01:18.86	42654

프로토콜 상세 내역이 나온 세 번째 시트를 읽을 경우 `sheet` 옵션을 3로 설정하면 되며 `skip=3`으로 하고 컬럼 이름을 별도로 사용하지 않으므로 `col_names=T`로 하여 읽을 수 있습니다.

```
dat <- read_excel("plate_reader.xls", sheet=3, skip = 3, col_names=F)
```

참고로 엑셀파일로 저장하기 위해서는 csv 파일로 데이터를 writing 한 뒤 Excel로 해당 csv 파일을 열고xlsx 파일로 저장할 수 있습니다.

## 5.2 Subset

R에서 데이터 저장은 `data.frame`이나 `matrix` 타입을 일반적으로 사용합니다. 이 데이터의 일부 열 또는 행의 데이터만을 가져와서 별도로 저장하거나 분석이 필요할 경우가 있습니다. 이때 인덱싱을 사용해서 일부 데이터를 선택하고 사용할 수 있으며 `subset` 함수도 이러한 선별 기능을 제공합니다. `subset`은 행과 열 모두를 선별할 수 있는 함수입니다. 다음 `airquality` 데이터는 1973년 날짜별로 뉴욕의 공기질을 측정한 데이터입니다. `NA`를 제외한 나머지 데이터만으로 새로운 데이터셋을 만들어 봅시다. `is.na` 함수를 사용하면 해당 데이터가 `NA`일 경우 `TRUE`, `NA`가 아닐 경우 `FALSE`를 반환해 줍니다.

```
is.na(airquality$Ozone)
ozone_complete1 <- airquality[!is.na(airquality$Ozone),]
ozone_complete1 <- subset(airquality, !is.na(Ozone))
```

위 `ozone_complete1`와 `ozone_complete2`는 같은 결과를 보입니다. 그러나 `ozone_complete1` 보다는 `ozone_complete2` 코드가 훨씬 직관적이고 가독성이 높습니다. 특히 `airquality$Ozone`로 `$`를 사용하여 변수에 접근한 것이 아닌 `Ozone`이라는 변수 이름을 직접 사용해서 접근함으로써 코드의 간결성과 가독성을 유지할 수 있습니다. 또한 `subset`의 `select` 옵션을 이용해서 변수를 선택할 수도 있으며 `&`(AND)와 `|`(OR) 연산자를 사용해서 조건을 두 개 이상 설정할 수 있습니다. 아래 `select` 옵션에서 `-`는 해당 변수를 제외한다는 의미입니다.

```
ozone_complete3 <- subset(airquality, !is.na(ozone), select=c(ozone, temp, month, day))
ozone_complete4 <- subset(airquality, !is.na(ozone) & !is.na(solar.r), select=c(-month, -day))
```

### Exercises

airquality 데이터에서 Temp와 Ozone 변수로 이루어진 df라는 이름의 data.frame을 만드시오 (단 NA가 있는 샘플(열)은 모두 제외하시오)

## 5.3 Merging and Split

merge 함수는 두 개 이상의 데이터셋을 통합하는 기능을 수행하는 함수입니다. 특히 rbind나 cbind와는 다르게, 결합하는 두 데이터에 공통적이거나 한 쪽의 데이터를 기준으로 결합을 수행 합니다. ?merge를 참고하면 by, by.x, by.y, all, all.x, all.y 등의 옵션으로 이러한 설정을 수행할 수 있습니다. 간단한 예제를 통해서 이해해 보겠습니다.

10명의 사람이 있고 이 사람들의 나이와 성별을 각각 나타낸 두 데이터셋이 있습니다. 그런데 df1은 나이만을 df2는 성별 정보만을 가지고 있으며 두 정보 모두 제공된 사람은 3명 (인덱스 4,5,6) 뿐입니다. 이제 merge를 이용해서 두 데이터셋을 결합해 보겠습니다.

```
## merge
df1 <- data.frame(id=c(1,2,3,4,5,6), age=c(30, 41, 33, 56, 20, 17))
df2 <- data.frame(id=c(4,5,6,7,8,9), gender=c("f", "f", "m", "m", "f", "m"))

df_inner <- merge(df1, df2, by="id", all=F)
df_outer <- merge(df1, df2, by="id", all=T)
df_left_outer <- merge(df1, df2, by="id", all.x=T)
df_right_outer <- merge(df1, df2, by="id", all.y=T)
```

만약 두 데이터셋의 id가 다를 경우나 각각 다른 기준으로 결합해야 하는 경우는 by대신 by.x, by.y 옵션을 사용할 수 있습니다.

split 함수는 데이터를 특정 기준으로 나누는 역할을 하며 해당 기준은 factor 형 벡터 형태로 주어질 수 있습니다. 예를 들어 airquality 데이터의 month 변수를 기준으로 데이터를 분리해 보겠습니다.

```
str(airquality)
g <- factor(airquality$Month)
airq_split <- split(airquality, g)
class(airq_split)
str(airq_split)
```

위와 같이 airq\_split은 길이가 5인 (5, 6, 7, 8, 9월) list타입이 되었고 각 요소는 서로 다른 size의 data.frame형으로 구성 된 것을 확인할 수 있습니다.

## 5.4 Transformation

R에서 기존 가지고 있는 데이터의 변경은 새로운 변수의 추가, 삭제, 변형과 샘플의 추가, 삭제, 변형을 생각해 볼 수 있습니다. 이러한 기능은 앞에서 배운 `merge`, `split`이나 `rbind`, `cbind`, 그리고 인덱싱을 활용한 값 변경 등의 방법을 이용할 수 있습니다. 또한 가장 직관적으로 필요한 변수들을 기존 데이터셋에서 추출한 후 `data.frame` 명령어를 사용해서 새로운 데이터셋으로 만들어주면 될 것 입니다.

이러한 방법들 외에 `within`을 사용할 경우 특정 변수의 변형과 이를 반영한 새로운 데이터셋을 어렵지 않게 만들수 있습니다. `with` 함수의 사용 예와 함께 `within` 함수를 사용하여 데이터를 변형하는 예를 살펴봅니다. `with`나 `within` 함수는 R을 활용하는데 많이 사용되는 함수들은 아닙니다. 또한 이러한 기능들은 `dplyr` 등의 패키지에서 제공하는 경우가 많아서 필수적으로 익힐 부분은 아닙니다. 그러나 개념적인 이해를 돕기위한 좋은 도구들이며 여전히 고수준의 R 사용자들이 코드에 사용하고 있는 함수들이므로 알아두는 것이 좋습니다.

```
## without with
ozone_complete <- airquality[!is.na(airquality$Ozone), "Ozone"]
temp_complete <- airquality[!is.na(airquality$Temp), "Temp"]
print(mean(ozone_complete))
print(mean(temp_complete))

## with
with(airquality, {
  print(mean(Ozone[!is.na(Ozone)]))
  print(mean(Temp[!is.na(Temp)]))
})
```

위 `with` 함수에서 보는바와 같이 `$`를 이용한 변수 접근 대신 `with` 함수 내에서는 `{, }` 안에서 해당 `data.frame`에 있는 변수 이름을 직접 접근할 수 있으며 따라서 코드의 간결함과 가독성이 향상됩니다.

`within` 함수는 `with` 함수와 같이 `{, }` 안에서 변수의 이름만으로 해당 변수에 접근이 가능하나 입력된 데이터와 변경된 변수(들)을 반환한다는 점이 다릅니다. 아래 예는 `airquality` 데이터의 화씨 (Fahrenheit) 온도를 섭씨 (Celsius) 온도로 변환해서 새로운 데이터셋을 만드는 코드입니다. `data.frame`을 이용한 코드와 비교해 보시기 바랍니다. 데이터셋 내에서 참조할 변수들이 많아질 경우 `airquality$xxx` 식의 코드를 줄이는 것 만으로도 코드의 가독성과 간결성을 유지할 수 있습니다.

```
newairquality <- within(airquality, {
  celsius = round((5*(Temp-32))/9, 2)
})
head(newairquality)

## data.frame
celsius <- round((5*(airquality$Temp-32))/9, 2)
newairquality <- data.frame(airquality, celsius)
head(newairquality)
```



### Exercises

다음 df 의 hour, minute, second로 나누어진 값들을 초 단위로 변환하여 second-s라는 변수에 저장한 후 기존 df에 추가한 df2 데이터셋을 만드시오 (within 함수 이용)

```
df <- data.frame(hour=c(4, 7, 1, 5, 8),
                 minute=c(46, 56, 44, 37, 39),
                 second=c(19, 45, 57, 41, 27))
```

## 5.5 Babies example

UsingR 패키지의 babies 데이터를 이용해서 산모의 흡연 여부와 신생아 몸무게의 관계를 알아보는 분석을 수행해 보겠습니다. 본 강의를 통해 배우지 않은 내용들이 있지만 코드를 따라 가면서 참고하시기 바랍니다. 우선 UsingR 패키지를 로딩합니다. 산모의 임신 기간이 (gestation) 999로 표기된 데이터는 명백히 에러이며 이들을 NA로 처리합니다.

```
library(UsingR)
head(babies)
## a simple way to checkout the data
plot(babies$gestation)
babies$gestation[babies$gestation>900] <- NA
str(babies)
```

아래와 같이 within 함수를 사용해서 babies\$ 를 반복해서 입력해주는 불편함을 줄이고 가독성을 높입니다. 똑같은 방법으로 dwt (아빠의 몸무게) 변수의 에러값들에 대해서도 NA 처리를 할 수 있습니다.

```
new_babies <- within(babies, {
  gestation[gestation==999] <- NA
  dwt[dwt==999] <- NA
})
str(new_babies)
```

smoke 변수는 흡연 여부를 나타내는 범주형 변수로 0, 1, 2, 3 값은 의미가 없습니다. 사람이 읽을 수 있는 label을 붙인 factor 형 변수로 변환하는 코드도 함께 작성해 보겠습니다.

```
str(babies$smoke)
new_babies <- within(babies, {
  gestation[gestation==999] <- NA
  dwt[dwt==999] <- NA
  smoke = factor(smoke)
  levels(smoke) = list(
    "never" = 0,
    "smoke now" = 1,
    "until current pregnancy" = 2,
```

```
  "once did, not now" = 3)
})
str(new_babies$smoke)
```

이제 임신기간과 흡연 여부를 분석해 볼 수 있습니다. 흡연 그룹별로 기간에 차이가 있는지를 알아보는 분석은 t-test나 ANOVA를 사용할 수 있습니다.

```
fit <- lm(gestation~smoke, new_babies)
summary(fit) ## t-test
anova(fit)
```

간단히 결과를 보면 `summary(fit)`은 3가지 t-test의 결과를 보여줍니다. never vs. smoke new 의 경우 t값이 -1.657로 피우지 않은 경우에 비해서 피우는 사람의 임신 기간이 유의하게 줄어들었음을 알 수 있습니다. 그에 비해서 현재 흡연하지 않는 경우 (never vs. until current pregnancy 또는 never vs. once did, not now) 차이가 없는 것으로 나옵니다.

이제 smoke now 인 경우 또는 나이가 25세 미만인 경우의 샘플에 대해서 `newdf`를 만들어 봅시다 (subset 함수 사용, id, gestation, age, wt, smoke 변수 선택). 이후 `ggplot`을 이용하여 몸무게와 임신기간의 산점도를 그려보면 크게 다르진 않으나 흡연하는 여성 중 몸무게가 적게 나가는 여성에게서 짧은 임신기간을 갖는 경향을 볼 수 있습니다.

```
newdf <- subset(new_babies, (smoke=="smoke now" | smoke == "never") & age < 25, select=c("id", "gestation", "wt", "smoke"))
# ggplot(newdf, aes(x=wt, y=gestation, color=smoke)) +
#   geom_point(size=3, alpha=0.5) +
#   facet_grid(.~smoke) +
#   theme_bw()
```

## 5.6 Useful functions

지금까지 배운 여러 R 프로그래밍 기법이나 함수들과 같이 R을 활용한 데이터 분석에서 자주쓰이거나 유용하게 사용되는 함수들을 소개합니다. 먼저 원소들을 비교하여 공통적 또는 유일한 원소들만을 추출해내는 함수들입니다.

```
#match(), %in%, intersect()

x <- 1:10
y <- 5:15
match(x, y)
x %in% y
intersect(x, y)

#unique()
unique(c(x, y))
```

다음은 스트링 관련 함수들로서 서열데이터 분석 등에서 유용하게 활용되는 함수들입니다.

```
#substr()
x <- "Factors, raw vectors, and lists, are converted"
substr(x, 1, 6)

#grep()
grep("raw", x)

#grepl()
grepl("raw", x)
if(grepl("raw", x)){
  cat("I found raw!")
}

x <- paste(LETTERS, 1:100, sep="")
grep("A", x)
x[grep("A", x)]

grepl("A", x)
r <- grepl("A", x)
if(r){
  cat("Yes, I found A")
}else{
  cat("No A")
}

#strsplit()
x <- c("Factors, raw vectors, and lists, are converted", "vectors, or for, strings with")
y <- strsplit(x, split=", ")

#unlist()
unlist(y)

y <- strsplit(x, split="")
ychar <- unlist(y)
ycount <- table(y2)
ycount_sort <- sort(ycount)
ycount_sort <- sort(ycount, decreasing = T)
ycount_top <- ycount_sort[1:5]
ycount_top_char <- names(ycount_top)

#toupper(), tolower()
toupper(ycount_top_char)
```

## Exercises

built-in 데이터셋 중 `state.abb` 은 미국의 50개 주에대한 축약어임.

- 1) 이 중 문자 A 가 들어가는 주를 뽑아 x에 저장 하시오 (`grep` 또는 `grep1` 사용)
- 2) `state.abb` 중 위 x에 저장된 이름들을 빼고 y에 저장 하시오 (`match()` 또는 `%in%`사용)
- 3) `state.abb`에 사용된 알파벳의 갯수를 구하고 가장 많이 쓰인 알파벳을 구하시오 (`strsplit()`, `table()` 등 사용)

## 5.7 apply

`apply`는 데이터를 변형하기 위한 함수라기 보다는 데이터를 다룰 때 각 원소별, 그룹별, row, 또는 column 별로 반복적으로 수행되는 작업을 효율적으로 수행할 수 있도록 해주는 함수입니다. `apply` 계열의 함수를 적절히 사용하면 효율성이나 편리성 뿐만 아니라 코드의 간결성 등 많은 장점이 있습니다. 쉬운 이해를 위해 `colMean` 함수를 예로 들면 `colMean`은 column 또는 row 단위로 해당하는 모든 값들에 대해 평균을 계산해주는 함수이고 `apply`를 사용할 경우 다음과 같이 `apply` 함수와 `mean` 함수를 이용해서 같은 기능을 수행할 수 있습니다. 아래는 `babies` 데이터의 cleaning 된 (위에서 만들었던) `new_babies` 데이터에 이어서 수행되는 내용입니다.

```
library(UsingR)
head(babies)
df <- subset(babies, select=c(gestation, wt, dwt))
colMeans(df, na.rm=T)
apply(df, 2, mean, na.rm=T)
```

위와 같이 `colMeans`와 `apply`가 똑같은 결과를 보여주고 있습니다. 두 번째 인자인 `margin`의 값으로 (?`apply`참고) 여기서는 2가 사용되었으며 `margin` 값이 1인지 2인지에 따라서 다음과 같이 작동을 합니다.

	열 (2)		
행 (1)	gestation	wt	dwt
	284	120	110
	282	113	148
	279	128	NA
	NA	123	197
	282	108	NA
	286	136	130

mean외에도 다양한 함수들이 사용될 수 있으며 아래와 같이 임의의 함수를 만들어서 사용할 수 도 있습니다. 아래 코드에서는 `function(x)...`로 바로 함수의 정의를 넣어서 사용했으나 그 아래 `mysd` 함수와 같이 미리 함수 하나를 만들고 난 후 함수 이름을 이용해서 `apply`를 적용할 수 있습니다.

```
apply(df, 2, sd, na.rm=T)
apply(df, 2, function(x){
  xmean <- mean(x, na.rm=T)
  return(xmean)
})
```

`apply` 함수는 특히 R에서 느리게 작동하는 loop (`for`, `while` 등) 문 대신 사용되어 큰 행렬에 대해서도 빠른 계산 속도를 보여줄 수 있습니다.

```
n <- 40
m <- matrix(sample(1:100, n, replace=T), ncol=4)
mysd <- function(x){
  xmean <- sum(x)/length(x)
  tmpdif <- x-xmean
  xvar <- sum(tmpdif^2)/(length(x)-1)
  xsd <- sqrt(xvar)
  return(xsd)
}

## for
results <- rep(0, nrow(m))
for(i in 1:nrow(m)){
  results[i] <- mysd(m[i,])
}
print(results)
apply(m, 1, mysd)
```

```
apply(m, 1, sd)
```

`apply` 함수 외에도 `sapply`, `lapply`, `mapply` 등의 다양한 `apply` 계열 함수가 쓰일 수 있습니다. 먼저 `lapply`는 `matrix` 형태 데이터가 아닌 `list` 데이터에 사용되어 각 `list` 원소별로 주어진 기능을 반복해서 수행하며 `sapply`는 `lapply`와 유사하나 벡터, 리스트, 데이터프레임 등에 함수를 적용할 수 있고 그 결과를 벡터 또는 행렬로 반환합니다.

```
x <- list(a=1:10, b=exp(-3:3), logic=c(T,T,F,T))
mean(x$a)
lapply(x, mean)
sapply(x, mean)

x <- data.frame(a=1:10, b=exp(-4:5))
sapply(x, mean)

x <- c(4, 9, 16)
sapply(x, sqrt)
sqrt(x)

y <- c(1:10)
sapply(y, function(x){2*x})
y*2
```

마지막 예제에서처럼 `sapply`나 `lapply`도 임의의 함수를 만들어 적용시킬 수도 있습니다. 자세히 살펴 보면 `y`는 10개의 값을 갖는 벡터이고 이 벡터의 각 원소(값에) 함수를 반복해서 적용하는 것입니다. 함수에서 `x`는 각 원소의 값을 차례차례 받는 역할을 하므로 1부터 10까지 값이 함수로 들어가 2를 곱한 수가 반환됩니다. 따라서 벡터연산을 하는 `y*2`와 결과가 같으나 원하는 함수를 정의해서 자유롭게 사용할 수 있다는 장점이 있습니다. 리스트의 경우는 다음과 같이 사용합니다.

```
y <- list(a=1:10, b=exp(-3:3), logic=c(T,T,F,T))
myfunc <- function(x){
  return(mean(x, na.rm=T))
}
lapply(y, myfunc)
unlist(lapply(y, myfunc))
```

즉, `myfunc`의 `x`가 `list y`의 각 원소들, `y[[1]]`, `y[[2]]`, `y[[3]]`를 각각 받아서 `mean` 연산을 수행해 줍니다. 결과로 각 `list` 원소들의 평균 값이 반환되며 `unlist` 함수는 `list` 형태의 반환 값을 `vector` 형태로 전환해 줍니다.

### Exercises

다음은 앞에서 수행했던 `airquality` 데이터를 월별로 나눈 데이터셋임. 이 데이터셋을 이용하여 각 월별로 온도와 오존 농도의 평균값을 저장한 `data.frame` 형식의 데이터를 만들기 위하여 다음 단계별 과정에 적절한 코드를 작성하시오

```
## dataset
g <- factor(airquality$month)
airq_split <- split(airquality, g)
```

- 1) 다음 df의 ozone 평균을 구하는 ozone\_func 함수를 작성하시오 (단 입력은 data.frame 형식의 오브젝트를 받고 출력은 평균값 (정수 값 하나) 출력. mean 함수 사용시 데이터에 NA가 포함되어 있을 경우 na.rm=T 옵션 적용)

```
## May data.frame
df <- airq_split[[1]]
#
# write your code here for ozone_func function
#

## Usage
ozone_func(df)
## output
# 23.61538
```

- 2) lapply와 ozone\_func 함수를 사용하여 airq\_split list 데이터의 월별 ozone 평균 값을 구하고 ozone\_means에 vector 형식으로 저장하시오
- 3) 위 1), 2)와 같은 방법으로 temp\_func 함수를 만들고 월별 temp의 평균값을 temp\_means에 vector 형식으로 저장하시오.
- 4) 위에서 구해진 두 변수값들을 이용하여 air\_means 라는 이름의 data.frame으로 저장하시오

---

이 저작물은 크리에이티브 커먼즈 저작자표시-비영리-변경금지 4.0 국제 라이선스에 따라 이용할 수 있습니다.





# Chapter 6

## Lecture note 2

설명이나 실습을 위해 R 5.19( ), 5.26( ) 강의를 진행하며 작성한 코드입니다.

### 6.1 numeric vector

- Ctrl + Alt + i 누르면 코드청크 생성
- 커서를 해당 라인에 두구 Ctrl + Enter 누르면 해당 라인 실행

```
2 + 2
((2-1)^2 + (1-3)^2)^(1/2)
2 + 2; 2 - 2
```

- 기억할 단축키
  - Ctrl + 1 : 편집창
  - Ctrl + 2 : 콘솔창

```
sqrt((4+3)*(2+1))
2^3 + 3^2
```

- 변수에 값 저장하기

```
x <- 1
y <- 2
z <- x + y
```

- 변수 값 보기

```
x
y
z
print(x)
cat(x)
```

- numeric vector 만들기

```
x <- 1:5
x
y <- seq(1, 5, 1)
y <- seq(from=1, to=5, by=1)
y <- seq(to=5, from=1, by=1)
y <- seq(5, 1, -1)
?seq
y
```

- 연습문제 풀이

```
odds <- seq(1, 100, by=2)
odds
evens <- seq(2, 100, 2)
evens
```

- vector의 인덱싱 (첫번째 값의 인덱스는 1부터 시작)

```
odds[1]
odds[1:10]
i <- 1:10
i
odds[i]
dim(odds)
length(odds)
```

```
precip
?precip

head(precip)
str(precip)
dim(precip)

precip[1]
precip["Mobile"]
```

- logical vector 설명
- 기본 그래픽 함수 이용하는 방법은 필요할때만 설명

```
precip
length(precip)
plot(precip)
```

- which 함수 활용한 40 이상만 선택

```
precip > 40
precip[precip > 40]
```

```
idx <- precip > 40
which(idx)
myprecip <- precip[which(idx)]
myprecip
plot(myprecip)
```

- which함수 활용한 짝수 만들기

```
mynumbers <- 1:1000
mynumbers_res <- mynumbers %% 2
i <- which(mynumbers_res == 0)
evens <- mynumbers[i]
evens
```

- 홀수 값을 저장하는 벡터 만들고 하나씩 샘플링 (sample 함수 사용)

```
odds <- seq(1, 1000, 2)
length(evens)
length(odds)
?sample
mysample <- c(sample(evens, 1), sample(odds, 1))
print(mysample[1])
```

- 문자열 붙이기, ,로 나누어진 벡터들 각각의 원소를 붙여줌

```
paste("X", "Y", "Z", sep="_")
paste("X", "Y", "Z", sep=" ")
paste("X", "Y", "Z", "X", "Y", "Z", "X", sep="")
```

- 여러 벡터에서 각각의 원소를 붙여주는 기능

```
paste(c("X","Y"), 1:10, sep=" ")
gene_names <- paste("gene", 1:100, sep=" ")
```

- collapse (하나의 벡터에서 해당 벡터의 원소들을 붙여주는 기능)

```
paste(c("X", "Y"), collapse = "")
```

- 예제 (sample(), paste(), rep())

```
x <- sample(c("A", "C", "G", "T"), size=20, replace = T)
x2 <- paste(x, collapse = "")

myseq <- rep("", 5)
myseq[1] <- paste(sample(c("A", "C", "G", "T"), size=20, replace = T), collapse="")
myseq[2] <- paste(sample(c("A", "C", "G", "T"), size=20, replace = T), collapse="")
myseq[3] <- paste(sample(c("A", "C", "G", "T"), size=20, replace = T), collapse="")
myseq[4] <- paste(sample(c("A", "C", "G", "T"), size=20, replace = T), collapse="")
```

```
myseq[5] <- paste(sample(c("A", "C", "G", "T"), size=20, replace = T), collapse="")
myseq
```

- 예제 'TAAGTCT' 바코드를 각 서열의 3'에 붙여보기

```
paste(myseq, c("TAAGTCT"), sep="")
strsplit("XYZ", split="")
```

- split 함수 사용

```
x <- strsplit("XYZ", split="")
class(x)
x
y <- unlist(x)
class(y)
y
```

- factor 간단한 설명

```
x <- c("Red", "Blue", "Yellow", "Red", "Blue")
x
y <- factor(x)
y
y[1] <- "gold"

levels(y)[4] <- "gold"
y
y[1] <- "gold"
y
```

- 데이터에서 factor들 보기

```
library(MASS)
Cars93
str(Cars93)
```

- 아미노산 및 각 아미노산에 해당하는 코돈 표현 예제

```
aa <- c("Phe", "Leu", "Ser")
class(aa)
aa <- factor(aa)
class(aa)

aa <- list()
aa[[1]] <- c("UUU", "UUC")
aa
aa[[2]] <- c("UUA", "UUG", "CUA", "CUU", "CUG", "CUC")
```

```
aa
class(aa)
names(aa) <- c("Phe", "Leu")
aa
aa[[1]][1]
aa[[2]][3]
```

- useful functions

```
z <- sample(1:10, 100, T)
?sample
z
head(z)
sort(z)
order(z)
table(z)
```

## 6.2 matrix

- 연습문제 풀이 성적별 테이블 정렬

```
mynum <- sample(1:100, 20, T)
mynum
score <- matrix(mynum, nrow = 10, ncol = 2)
score
myrowname <- paste("Name", 1:10, sep="")
myrowname
rownames(score) <- myrowname
score
colnames(score) <- c("Math", "Eng")

total_score <- score[,1] + score[,2]
total_score <- score[, "Math"] + score[, "Eng"]
sort(total_score, decreasing = T)
score
o <- order(total_score, decreasing = T)
o
score[o,]
```

## 6.3 data.frame

```
math_score <- sample(1:100, 10, T)
eng_score <- sample(1:100, 10, T)
```

```
score <- data.frame(math_score, eng_score)
score
score$math_score
score$eng_score
```

- 추가 연습문제: 수학, 영어 성적을 더해서 total\_score를 만들고 이 값을 기준으로 내림차순으로 score 데이터프레임을 정렬 하시오.

```
total_score <- score$math_score + score$eng_score
o <- order(total_score, decreasing = T)
score[o,]
```

## 6.4 list

```
score
class(score)
mynum
class(mynum)

z <- list()
z[[1]] <- score
z[[2]] <- mynum
z
names(z) <- c("dataframe", "numericvector")
z
z$dataframe
z$numericvector
```

- 리스트 만들때 미리 원소의 개수를 알고 있으면 그 원소의 개수에 맞게 생성해 주는 것이 좋음 aa <- vector("list", 5)
- 각 아미노산에 해당하는 코돈 길이가 달라도 list 형태로 저장 가능

```
aa <- list()
aa[[1]] <- c("UUU", "UUC")
aa
aa[[2]] <- c("UUA", "UUG", "CUA", "CUU", "CUG", "CUC")
aa
class(aa)
names(aa) <- c("Phe", "Leu")
aa
as.data.frame(aa)
```

- 바람직한 데이터는 column은 변수, row는 샘플 구조의 데이터. 예를 들어서, 변수:먹이, 수명 -> 컬럼, 샘플: 마우스1, 마우스2 -> Row, 등

```
aa <- c("Phe", "Leu")
codon <- c("UUU", "UUC", "UUA", "UUG", "CUA", "CUU", "CUG", "CUC")
data.frame(aa, codon)
aa <- c(rep("Phe", 2), rep("Leu", 6))
codon <- c("UUU", "UUC", "UUA", "UUG", "CUA", "CUU", "CUG", "CUC")
data.frame(aa, codon)
```

## 6.5 functions

```
source("myscript.R")
```

- 함수만들기

```
my_function_name <- function(parameter1, parameter2, ... ){
  ##any statements
  return(object)
}
```

```
## mynumbers: numeric vector
mymean <- function(mynumbers){
  #cat("Input numbers are", mynumbers, "\n")
  numbers_mean <- sum(mynumbers)/length(mynumbers)
  #out <- paste("The average is ", numbers_mean, ".\n", sep="")
  #cat(out)
  return(numbers_mean)
}
```

- 함수 만든 후 loading 할 때 {, } 밖에서 Ctrl + Enter로 로딩

```
mymean(c(1,2,3))

x <- c(1, 2, 3, 0.452, 1.474, 0.22, 0.545, 1.205, 3.55)
mymean(x)

mymean()
```

- 데이터 표준화 예제

```
mysd <- function(x){
  numbers_sd <- sqrt(sum((x - mymean(x))^2)/(length(x)-1))
  return(numbers_sd)
}

#x <- sample(1:100, 1000, T)
x <- rnorm(1000, 10, 5)
z <- (x - mymean(x))/mysd(x)
```

```

mymean(z)
mysd(z)
mymean(x)
mysd(x)

plot(density(x))
density(x)

```

- 코드비교

```

x <- c(10, 20, 30)
x + 10

y <- rep(0, 3)
for(i in 1:3){
  y[i] <- x[i] + 10
}
y

```

## 6.6 for

```

x <- 1:10
for(i in x){
  cat(i, "\n")
  flush.console()
}

```

```

x <- 1
while(x <= 10){
  cat(x, "\n")
  flush.console()
  x <- x + 1
}

```

- 랜덤 서열만들기 예제

```

x <- sample(c("A", "C", "G", "T"), size=20, replace = T)
x2 <- paste(x, collapse = "")

myseq <- rep("", 5)
myseq[1] <- paste(sample(c("A", "C", "G", "T"), size=20, replace = T), collapse="")
myseq[2] <- paste(sample(c("A", "C", "G", "T"), size=20, replace = T), collapse="")
myseq[3] <- paste(sample(c("A", "C", "G", "T"), size=20, replace = T), collapse="")
myseq[4] <- paste(sample(c("A", "C", "G", "T"), size=20, replace = T), collapse="")
myseq[5] <- paste(sample(c("A", "C", "G", "T"), size=20, replace = T), collapse="")

```



```

numseq <- 7
myseq <- rep("", numseq)
for(i in 1:length(myseq)){
  x <- sample(c("A", "C", "G", "T"), size=20, replace = T)
  myseq[i] <- paste(x, collapse="")
}
myseq

```

## 6.7 Data transformation

- UsingR 패키지의 babies 데이터셋을 적절히 변환하는 예제. with와 within 활용법 알아두기 (설명 안 함)

```

library(UsingR)
head(babies)
str(babies)

new_babies <- within(babies, {
  gestation[gestation==999] <- NA
  dwt[dwt==999] <- NA
  smoke = factor(smoke)
  levels(smoke) = list(
    "never" = 0,
    "smoke now" = 1,
    "until current pregnancy" = 2,
    "once did, not now" = 3)
})
str(new_babies)

fit <- lm(gestation~smoke, new_babies)
summary(fit) ## t-test
anova(fit)

```



## Chapter 7

# R basic graphics

### 7.1 scatter plot

R에서 plot 함수는 가장 기본이 되는 그래프 함수입니다. 아래는 산포도를 그려주는 코드로서 myxy가 두 개의 변수(x1과 y1)를 가지고 있으므로 아래 명령들은 모두 같은 그림을 그려주게 됩니다.

```
x <- c(1:100)
y <- x*2 + rnorm(100)
myxy <- data.frame(x,y)
plot(myxy)
plot(myxy$x, myxy$y)
plot(x=myxy$x, y=myxy$y)
plot(y~x, data=myxy)
```

가장 마지막 명령은 formula를 사용한 plot으로 첫번째 파라미터 인자로 formula 타입이 전달되면 plot.formula 함수가 실행되며 x, y 값이 전달될 경우 plot.default 함수가 수행됩니다. R에서는 이렇게 전달되는 파라미터의 타입에 따라서 다른 기능을 하는 함수를 Generic function 이라고 합니다. 만약 기존 그림에 추가 데이터의 산포를 그리고 싶은 경우 points라는 함수를 사용합니다.

```
z <- sample(1:100, 100, replace =T)
points(x, z)
points(x, z, col="red")
```

### 7.2 histogram

hist 함수는 데이터들의 분포를 히스토그램으로 그려주는 함수입니다. 히스토그램은 데이터들이 갖는 값을 특정 구간으로 나누고 각 구간에 해당하는 데이터가 몇 개인지

빈도수를 계산하여 막대그래프로 보여줍니다.

```
x <- rnorm(100)
hist(x, br=20, xlim=c(-3,3), main="Main text", xlab="X label")

hist(airquality$Wind, br=50)
hist(airquality$Wind, br=10)
```

### 7.3 boxplot

boxplot (상자 수염 그림)은 데이터의 여러가지 대표값 (중간값 median, 첫번째 사분위수 1st quantile, 세번째 사분위수 3rd quantile, 최소 minimum, 최대값 maximum) 등을 한눈에 볼 수 있도록 만들어놓은 그래프 입니다. 수염이 나타내는 값은 최소값이나 최대값이 될 수 있고 또는 하위 1.5 IQR 에서 최소 데이터와 상위 1.5 IQR 내에 최고 데이터를 나타낼 수 있으며 이 경우 그 외에 존재하는 값들은 outlier가 됩니다.

```
x <- rnorm(100)
boxplot(x)

r <- boxplot(airquality$Wind)

airquality$Wind[which(airquality$Wind > (1.5*(r$stats[4]-r$stats[2])+r$stats[4]))]

with(airquality, {
  Wind[which(Wind > (1.5*(r$stats[4]-r$stats[2])+r$stats[4]))]
})

with(airquality, {
  val <- (1.5*(r$stats[4]-r$stats[2])+r$stats[4])
  Wind[which(Wind > val)]
})

with(airquality, {
  iqr <- quantile(Wind, 3/4) - quantile(Wind, 1/4)
  val <- 1.5 * iqr + quantile(Wind, 3/4)
  Wind[which(Wind > val)]
})
```

data.frame 타입의 오브젝트에 대해서 boxplot을 그릴 경우 여러 변수의 데이터들의 분포를 한눈에 비교할 수 있습니다.

```
y <- rnorm(100, 1, 1)
#boxplot(y)
xy <- data.frame(x, y)
```

```

boxplot(xy)
class(xy)

##
mean_vals <- sample(10)
mymat <- sapply(mean_vals, function(x){rnorm(100, x)})
dim(mymat)
boxplot(mymat)

```

## 7.4 barplot

막대그래프는 각 값들을 막대 형태로 나란히 배치하여 서로 비교가 용이하도록 만든 그래프입니다. `table` 함수는 같은 값을 갖는 데이터들이 몇 개나 있는지 테이블을 만들어주는 함수입니다. `rbind`는 두 변수를 row를 기준으로 붙여주는 역할의 함수입니다.

```

x <- sample(1:12, 200, replace = T)
tab_x <- table(x)
y <- sample(1:12, 200, replace = T)
tab_y <- table(y)
tab_xy <- rbind(tab_x, tab_y)
barplot(tab_xy)
barplot(tab_xy, beside = T)
barplot(tab_xy, beside = T, col=c("darkblue","red"))
barplot(tab_xy, beside = T, col=c("darkblue","red"), xlab="Month")
barplot(tab_xy, beside = T, col=c("darkblue","red"), xlab="Month", horiz=TRUE)

```

### Exercises

- 1) iris 데이터의 꽃받침 (Sepal) 길이와 넓이를 각각 x와 y축으로 하는 산포도를 그리시오
- 2) iris 데이터에서 setosa 품종의 꽃받침의 (Sepal) 길이와 넓이 데이터를 빨간 점으로 나타내시오
- 3) iris 데이터에서 꽃받침과 (Sepal) 꽃잎의 (Petal) 길이의 분포를 그리시오 (hist 사용)
- 4) iris 데이터에서 꽃받침과 (Sepal) 꽃잎의 (Petal) 넓이의 분포를 그리시오 (boxplot 사용)
- 5) iris 데이터에서 품종별 꽃받침 (Sepal) 길이의 분포를 그리시오 (boxplot 사용)

## 7.5 Draw multiple graphs in the same plot

위 예제들에서 사용한 high level function들을 low level function (lines, points, ablines, axis 등)들과 함께 사용함으로써 원하는 도표 대부분을 그려낼

수 있습니다. 최근 널리 사용되는 ggplot2 패키지를 이용한 그래프 사용법 강의에서는 오늘 배우는 그래픽 명령어는 거의 사용하지 않습니다. 그러나 위 함수들은 R의 기본 그래프 함수들로서 단순한 도표에서부터 복잡한 그래픽까지 구현할 수 있는 다양한 유연성을 제공하므로 기본적인 사용법을 정확히 이해하는 것이 좋습니다.

아래 도표는 평균 0, 분산 1인 분포에서 500개의 랜덤한 수를 뽑아 x에 저장하고 x의 분포를 히스토그램으로 표현한 것 입니다. 그리고 x 값들과 상관성이 있는 y값들을 (x에 2를 곱하고 평균 5, 분산 1인 랜덤하게 뽑힌 수를 노이즈로 더함) 생성하고 모든 1000개 값들의 분포를 그린 히스토그램 입니다.

```
x <- rnorm(500)
hist(x, 100)
y <- 2*x + rnorm(500, mean=5, sd=1)
z <- c(x,y)
hist(z, br=100)
```

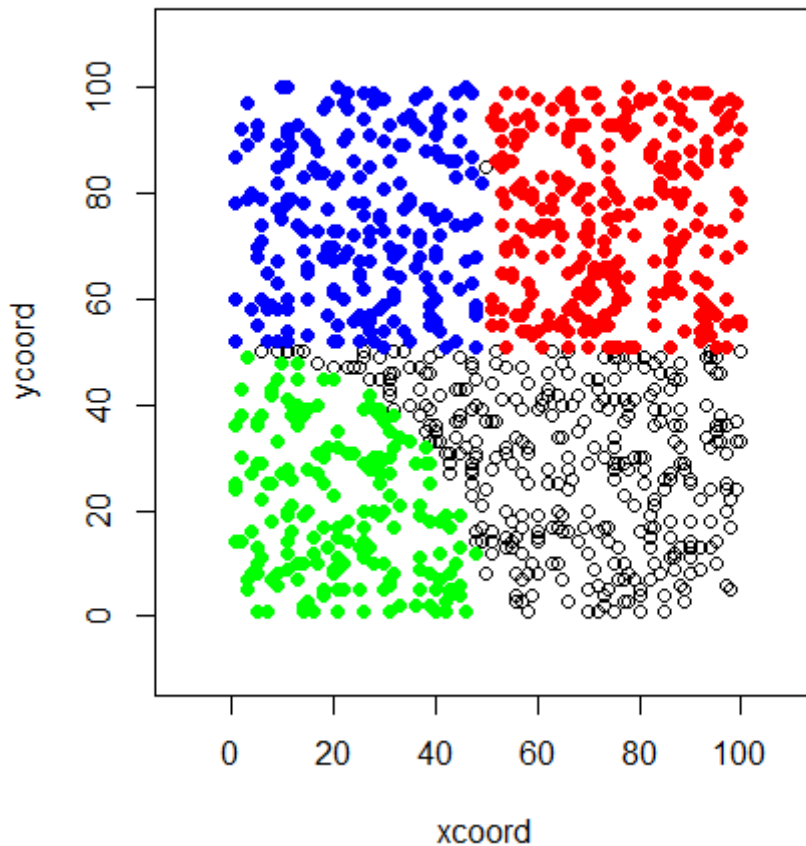
이제 위 histogram 그래프에 density 함수와 lines 함수를 조합하여 확률밀도함수 커브를 그려 넣을 수 있습니다. 이 때 hist 함수에 probability=T 옵션을 넣어 y축의 스케일을 확률밀도함수의 y 스케일과 맞춰주어 같은 화면에 그려지도록 했습니다.

```
hist(z, br=100)
hist(z, br=100, probability = T)
zd <- density(z)
lines(zd)
```

또한 아래 그래프는 위에서 생성한 x, y 값의 산포도를 그리고 x축과 y축 범위를 xlim, ylim 파라미터로 조절하는 예제 입니다. ?pch 도움말을 참고하여 다양한 포인트 모양을 선택할 수 있으며 x 값이 0 보다 작은 경우의 index를 뽑아 해당되는 x 값들과 그 값들의 짝이 되는 y값들에 대해서만 다시 포인트 그림을 red 색상으로 그려 넣었습니다. lm 은 linear model의 약자로 회귀 곡선을 구할 때 사용하는 함수이며 이 함수를 abline과 조합하여 회귀 직선을 그릴 수 있습니다.

```
plot(x,y, xlim=c(-5, 5), ylim=c(-5, 15), pch=3)
idx <- which(x<0)
points(x[idx], y[idx], col="red")
fit <- lm(y~x)
abline(fit)
```

## Exercises



- 1부터 100까지 수를 랜덤하게 1000개 생성해서 x좌표를 생성하고 xcoord에 저장 하시오 (중복허용)
- 1부터 100까지 수를 랜덤하게 1000개 생성해서 y좌표를 생성하고 ycoord에 저장 하시오 (중복허용)
- x, y 좌표 평면에 xcoord와 ycoord 값을 이용하여 좌표를 (산포도) 그리되 x와 y의 범위가 모두 -10부터 110까지 되도록 지정 하시오 (plot 이용)
- 앞서 문제와 같은 plot에 x가 50보다 크고 y가 50보다 큰 곳에 있는 좌표들에 red closed circle로 표현하시오 (which, points, pch parameter 등 이용, 아래 참고)

```
idx <- which(xcoord>50 & ycoord>50)
points(x=xcoord[idx], y=ycoord[idx], col="red", pch=19)
```

- 앞서 문제와 같은 plot에 x가 50보다 작고 y가 50보다 큰 곳에 있는 좌표들에 blue closed circle로 표현하시오 (which, points, pch parameter 등 이용)
- 앞서 문제와 같은 plot에 원점으로부터 거리가 50 이하인 좌표들을 green closed circle로 표현 하시오

## 7.6 Useful functions II

```
#match(), %in%, intersect()

x <- 1:10
y <- 5:15
match(x, y)
x %in% y
intersect(x, y)

#unique()
unique(c(x, y))

#substr()
x <- "Factors, raw vectors, and lists, are converted"
substr(x, 1, 6)

#grep()
grep("raw", x)

#grepl()
grepl("raw", x)
if(grepl("raw", x)){
  cat("I found raw!")
}

x <- paste(LETTERS, 1:100, sep="")
grep("A", x)
x[grep("A", x)]

grepl("A", x)
r <- grepl("A", x)
if(r){
  cat("Yes, I found A")
}else{
  cat("No A")
}

#strsplit()
```



```

x <- c("Factors, raw vectors, and lists, are converted", "vectors, or for, strings with")
y <- strsplit(x, split=" ", " ")

#unlist()
unlist(y)

y <- strsplit(x, split="")
ychar <- unlist(y)
ycount <- table(y2)
ycount_sort <- sort(ycount)
ycount_sort <- sort(ycount, decreasing = T)
ycount_top <- ycount_sort[1:5]
ycount_top_char <- names(ycount_top)

#toupper(), tolower()
toupper(ycount_top_char)

```

### Exercises

built-in 데이터셋 중 `state.abb` 은 미국의 50개 주에대한 축약어임.

- 1) 이 중 문자 A 가 들어가는 주를 뽑아 x에 저장 하시오 (`grep` 또는 `grep1` 사용)
- 2) `state.abb` 중 위 x에 저장된 이름들을 빼고 y에 저장 하시오 (`match()` 또는 `%in%`사용)
- 3) `state.abb`에 사용된 알파벳의 갯수를 구하고 가장 많이 쓰인 알파벳을 구하시오 (`strsplit()`, `table()` 등 사용)

### Exercises

`iris` 데이터셋의 각 Species 별로 꽃잎과 꽃받침의 길이와 넓이에 대한 평균값들을 구하고 막대그래프를 그리시오

---

이 저작물은 크리에이티브 커먼즈 저작자표시-비영리-변경금지 4.0 국제 라이선스에 따라 이용할 수 있습니다.



## Chapter 8

# tidyverse

tidyverse (<https://www.tidyverse.org/>)는 데이터 사이언스를 위한 R 기반의 독창적인 패키지들의 모음입니다. Rstudio의 핵심 전문가인 해들리위컴이 (Hadley Wickham) 중심이 되어 만들어 졌으며 기존의 툴보다 쉽고 효율적으로 데이터 분석을 수행할 수 있습니다.



### R packages for data science

The tidyverse is an opinionated **collection of R packages** designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

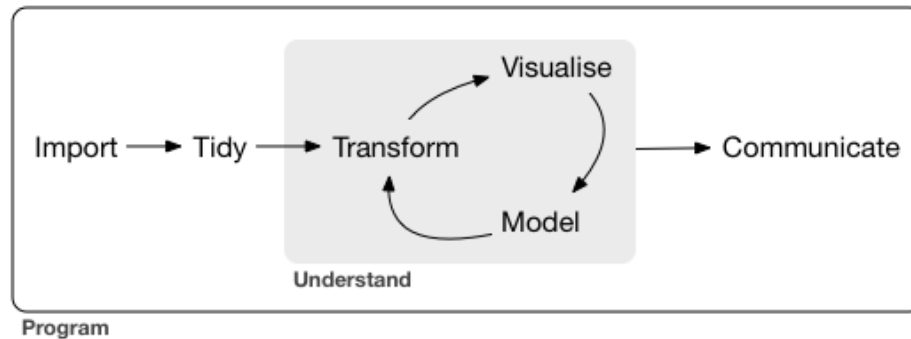
Install the complete tidyverse with:

```
install.packages("tidyverse")
```

데이터사이언스는 넓은 범위의 개념과 방법적인 정도가 있는 것은 아닙니다. 그러나 tidyverse의 목적은 데이터 분석을 위한 핵심이되는 고효율의 툴을 제공하는 것이며 그 철학은 다음과 같은 그림으로 요약할 수 있습니다.

### 8.1 tibble object type

R은 20년 이상된 비교적 오랜 역사를 가진 언어로서 `data.frame` 형태의 데이터 타입이 가장 많이 사용되고 있습니다. 그러나 당시에는 유용했던 기능이 시간이

Figure 8.1: from <https://r4ds.had.co.nz/>

흐르면서 몇몇 단점들이 드러나는 문제로 기존 코드를 그대로 유지한채 package 형태로 단점을 보완한 새로운 형태의 tibble 오브젝트 형식을 만들어 냈습니다. 대부분의 R 코드는 여전히 data.frame 형태의 데이터 타입을 사용하고 있으나 tidyverse에서는 tibble이 사용되는 것을 참고하시기 바랍니다.

```
library(tidyverse)
```

```
tb <- tibble(
  x = 1:5,
  y = 1,
  z = x ^ 2 + y
)
```

```
as_tibble(iris)
head(iris)
```

tibble은 data.frame과 다음 몇 가지 점이 다릅니다. data.frame의 경우 타입을 변환할 때 강제로 값의 타입을 바꾸거나 내부 변수의 이름을 바꾸는 경우가 있었으나 tibble은 이를 허용하지 않습니다. 샘플들 (row) 이름을 바꿀수도 없습니다. 또한 프린팅할 때 출력물에 나오는 정보가 다르며 마지막으로 data.frame은 subset에 대한 타입이 바뀔 경우가 있었지만 tibble은 바뀌지 않습니다.

```
x <- 1:3
y <- list(1:5, 1:10, 1:20)

data.frame(x, y)
tibble(x, y)
```

tibble은 컬럼 하나가 벡터형 변수가 아닌 리스트형 변수가 될 수 있다는 것도 data.frame과 다른 점 입니다.

```
names(data.frame(`crazy name` = 1))
names(tibble(`crazy name` = 1))
```

또한 다음과 같이 사용되는 변수의 (x) 참조 범위가 다릅니다.

```
data.frame(x = 1:5, y = x ^ 2)
tibble(x = 1:5, y = x ^ 2)

df1 <- data.frame(x = 1:3, y = 3:1)
class(df1)
class(df1[, 1:2])
class(df1[, 1])

df2 <- tibble(x = 1:3, y = 3:1)
class(df2)
class(df2[, 1:2])
class(df2[, 1])
class(df2$x)
```

## 8.2 Tidy data structure

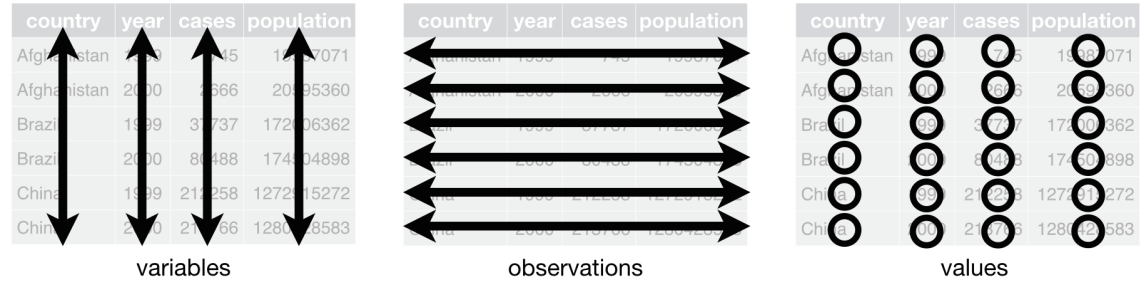
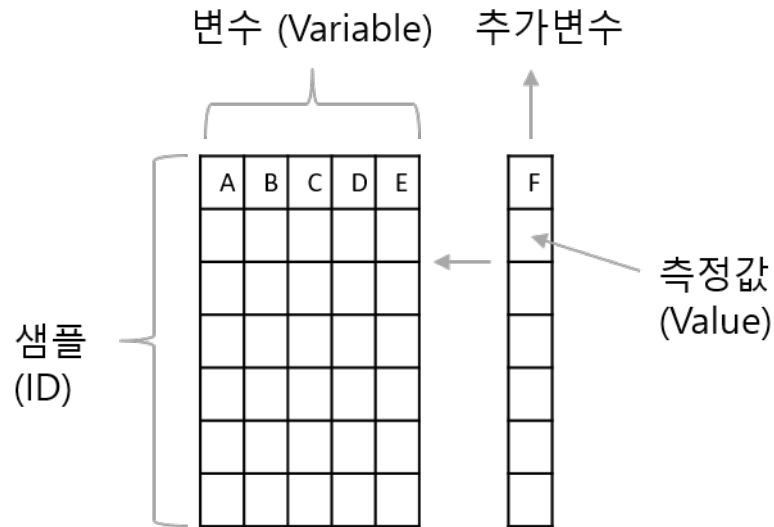
데이터의 변수와 값을 구분하는 일은 적절한 데이터 분석을 위해 필수적인 과정입니다. 특히 복잡하고 사이즈가 큰 데이터일 경우는 더욱 중요할 수 있으나 경험에 의존해서 구분을 하는 것이 대부분입니다. Tidy data는 이러한 변수와 값의 명확한 구분과 활용을 위한 데이터 구조중 하나입니다 (Hadley Wickham. Tidy data. The Journal of Statistical Software, vol. 59, 2014).

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

tidy data는 다음과 같은 특징이 있습니다.

- 각 변수는 해당하는 유일한 하나의 column을 가짐
- 각 샘플은 해당하는 유일한 하나의 row를 가짐
- 각 관측값은 해당하는 유일한 하나의 cell을 가짐

Tidy 데이터는 Long형 데이터로 알려져 있기도 합니다. 참고로 Wide형 데이터의 경우 샘플 데이터가 늘어날수록 row에 쌓이고 새로운 변수는 column에 쌓이는 방식으로 데이터가 확장되는 형태입니다. 엑셀에서 볼 수 있는 일반적인 형식으로 다음 그림과 같습니다.

Figure 8.2: from <https://r4ds.had.co.nz/>

Long형 데이터의 경우 ID, variable, value 세가지 변수만 기억하면 되겠습니다. 위 wide형 데이터 경우를 보면 ID, variable, 그리고 value 이 세가지 요인이 주요 구성 요소임을 알 수 있습니다. Long형으로 변환할 경우 샘플을 참조할 수 있는 어떤 변수 (variable)도 ID가 될 수 있으며 2개 이상의 변수가 ID로 지정될 수 있습니다. 참고로 ID를 지정할 경우 해당 ID는 가능하면 중복되지 않는 값들을 갖는 변수를 사용해야 식별자로서 기능을 적절히 수행할 수 있습니다. Long형을 사용할 경우 데이터의 변수가 늘어나도 행의 수만 늘어나므로 코딩의 일관성과 변수들의 그룹을 만들어서 분석하는 등의 장점이 있습니다. 아래는 새로운 변수 F가 추가될 때 long 형 데이터에 데이터가 추가되는 경우를 나타낸 그림 입니다.

ID	variable	values
1	B	
1	C	
...	...	
2	B	
2	C	
...	...	

+

1	F	
2	F	
3	F	
4	F	
...	...	

추가변수

### 8.3 Pivoting

일반적으로 얻어지는 데이터의 형태는 wide형이며 이를 Long형으로 변환하기 위해서는 tidyverse 패키지에 속한 tidyr 패키지의 pivot\_longer와 pivot\_wider를 사용합니다. 또한 reshape2 패키지의 melt 함수와 그 반대의 경우 dcast 함수를 사용할 수도 있습니다. 본 강의에서는 tidyr 패키지를 사용합니다. wide형 데이터를 long형으로 변환하거나 long형을 wide형으로 변환하는 작업을 pivoting 이라고 합니다.

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

→

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

name value

`airquality` 데이터는 전형적인 wide형 데이터로 특정 날짜에 네 개의 변수에 해당하는 값들을 측정했습니다. 이 데이터를 long형으로 바꿀 경우 ID를 날짜로 하면 데이터들을 식별 할 수 있습니다. 그런데 날짜는 변수가 Month와 Day두 개로 나누어져 있으므로 다음과 같이 두 변수를 식별 변수로 (ID로) 사용 합니다. 확인을 위해 상위 5개의 데이터만 가지고 형 변환을 진행해 보겠습니다.

```
airquality
myair <- airquality[1:5,]
myair_long <- pivot_longer(myair, c("Ozone", "Solar.R", "Wind", "Temp"))
myair_long
myair_long2 <- pivot_longer(myair, c(Ozone, Solar.R, Wind, Temp))
myair_long2
myair_long3 <- pivot_longer(myair, !c(Month, Day))
myair_long3
```

생성되는 long형 데이터의 변수 이름인 name과 value는 다음 파라미터를 지정하여 바꿀 수 있습니다.

```
myair_long <- pivot_longer(myair,
                           c(Ozone, Solar.R, Wind, Temp),
                           names_to = "Type",
                           values_to = "Observation")
myair_long
```

long형 데이터를 wide형 데이터로 변환 할 수도 있습니다.

```
pivot_wider(myair_long, names_from = Type, values_from = Observation)
```

### Exercises

- 1) 다음 데이터가 long형인지 wide형인지 판단하시오
- 2) long형이면 wide형으로 wide형이면 long형으로 변환하시오



```
stocks <- tibble(
  year   = c(2015, 2015, 2016, 2016),
  month  = c( 1,   2,   1,   2),
  profit = c(1.88, 0.59, 0.92, 0.17)
)
```

ggplot을 이용한 그래프 작성에는 위와 같은 long형 데이터가 주로 사용됩니다. R을 이용한 데이터 가시화는 dplyr 패키지로 wide형 데이터를 편집하고 pivot\_longer 함수로 long형 데이터로 변환 후 ggplot을 이용하는 방식으로 수행합니다. 두 데이터 포맷에 대한 좀 더 구체적인 내용은 다음 링크를 참고하시기 바랍니다. <https://www.theanalysisfactor.com/wide-and-long-data/>

## 8.4 Separating and uniting

데이터를 분석할 때 하나의 컬럼에 두 개 이상의 변수값이 저장되어 있거나 두 개의 변수를 하나의 컬럼으로 합해야 하는 경우가 종종 있습니다. 전자의 경우 separate() 함수를 사용해서 두 변수(컬럼)으로 나누어 줄 수 있으며 후자의 경우 unite() 함수를 사용하여 두 변수를 하나의 값으로 병합할 수 있습니다. 다음은 airquality 데이터에서 Month와 Day 변수를 하나의 컬럼으로 병합하여 Date라는 변수로 만들어 주는 경우의 예입니다.

```
newairquality <- unite(airquality, Date, Month, Day, sep=".")
newairquality
```

separate() 함수를 사용하면 다음과 같이 해당 변수의 값을 나누어 다시 두 개의 변수(컬럼)으로 나누어 줄 수 있습니다.

```
separate(newairquality, col=Date, into = c("Month", "Day"), sep = "\\.")
```

## 8.5 dplyr and pipe operator

dplyr (<https://dplyr.tidyverse.org/>) 은 ggplot2을 개발한 해들리위컴이 (Hadley Wickham) 중심이 되어 만들어 졌으며 ggplot2와 함께 tidyverse의 (<https://www.tidyverse.org/>) 핵심 패키지 입니다. dplyr은 데이터를 다루는 크기나 분석의 속도, 편의성을 향상시켜 새롭게 만들어놓은 패키지 입니다. 기존 apply와 같은 행렬 연산 기능과 subset, split, group 와 같은 행렬 편집 기능을 더하여 만들어진 도구라고 할 수 있습니다.

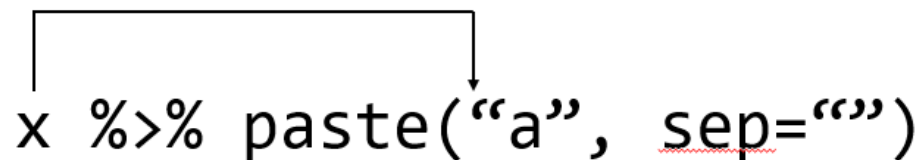
dplyr의 전신이라 할 수 있는 plyr 패키지는 다음과 같이 설명이 되어 있습니다. A set of tools for a common set of problems: you need to split up a big data structure into homogeneous pieces, apply a function to each piece and then combine all the results back together. 즉 split-apply-combine 세 가지 동작을 쉽게 할 수 있도록 만들어 놓은 툴 입니다. R이 다른 언어에 비해 데이터 분석에서 주목을 받는 이유로 split, apply 등의 행렬 연산 함수가 발달한 것을 내세우는데 dplyr은 이들을 보다 더 편리하게 사용할 수 있도록 만들어 놓은 것 입니다.

dplyr의 사용을 위해서는 여러 명령을 연속적으로 수행하도록 해주는 %>% 파이프 오퍼레이터의 이해가 필요합니다. 파이프 오퍼레이터의 작동법은 간단히 %>%의 왼쪽 코드의 결과를 출력으로 받아 오른쪽 코드의 입력 (첫번째 파라미터의 값)으로 받아들이는 작동을 합니다 (단축키: **Shift+Ctrl+m**). 다음 예에서 보면 `sin(pi)`와 같은 함수의 일반적인 사용법 대신 `pi %>% sin` 처럼 사용해도 똑같은 결과를 보여줍니다. `cos(sin(pi))`와 같이 여러 함수를 중첩하여 사용할 경우와 비교해서 코드의 가독성이나 효율 측면에서 크게 향상된 방법을 제공해 줍니다.

```
library(dplyr)

pi %>% sin
sin(pi)
pi %>% sin %>% cos
cos(sin(pi))
```

특히 %>%는 이후 설명할 dplyr의 `group_by`, `split`, `filter`, `summary` 등의 행렬 편집/연산 함수를 빈번히 다양한 조합으로 쓰게되는 상황에서 더 큰 효과를 발휘할 수 있습니다.



`x %>% paste("a", sep="")`

pipe operator의 왼쪽 구문의 결과가 오른쪽 구문의 첫 번째 파라미터의 입력 값으로 처리된다고 말씀 드렸습니다. 즉, 함수에서 사용되는 파라미터가 여러개일 경우가 있으므로 기본적으로 %>%의 왼쪽 구문의 출력 값은 오른쪽 구문 (함수)의 첫 번째 인자의 입력값으로 들어가는 것 입니다. 이는 다음 예들을 통해서 명확히 알 수 있습니다. 먼저 `paste`함수는 그 파라미터로 ,로 구분되는 여러개의 입력 값을 가질 수 있습니다. 따라서 다음 코드는 `x`가 `paste`의 첫 번째 파라미터로 들어가게 되어 "1a", "2a", "3a", "4a", "5a"로 a 앞에 x 값들이 붙어서 출력된 것을 알 수 있습니다.

```
x <- 1:5
x %>% paste("a", sep="")
```

특정 데이터셋의 컬럼별 평균을 구하고 각 평균의 합을 구할 경우를 생각해 봅시다. R에서는 `colMeans`라는 특별한 함수를 제공하여 컬럼별로 평균을 계산해 줍니다. 그 후 `sum` 함수를 사용하여 최종 원하는 값을 얻을 수 있습니다. 이러한 코드를 %>% 오퍼레이터를 사용한 경우의 코드와 비교해 볼 수 있습니다.

```
x <- data.frame(x=c(1:100), y=c(201:300))
sum(colMeans(x))
```

```
x <- data.frame(x=c(1:100), y=c(201:300))
x %>% colMeans %>% sum
```

그럼 만약 두 번째 파라미터에 입력으로 왼쪽 구문의 출력을 받아들이고 싶을 경우는 place holder . 을 사용하면 되겠습니다. round 함수는 두 개의 파라미터를 설정할 수 있으며 digits 라는 두 번째 파라미터에 값을 pipe operator로 넘겨주고 싶을 경우 아래와 같이 표현할 수 있습니다.

```
6 %>% round(pi, digits=.)
round(pi, digits=6)
```

### Exercises

- 1) pipe operator를 사용해서 airquality데이터를 long형으로 전환하는 코드를 작성하시오 (단 col 파라미터에는 Ozone, Solar.R, Wind, Temp 변수를 넣음)
- 2) pipe operator를 사용해서 airquality데이터의 Month와 Day 변수(컬럼)을 Date 변수로 병합하는 코드를 작성하시오

## 8.6 dplyr - Important functions

이제 dplyr 패키지에서 제공하는 함수를 사용해 보겠습니다. dplyr을 구성하는 중요한 함수는 다음과 같습니다.

- filter() - 샘플 (rows) 선택
- arrange() - 샘플들의 정렬 순서 변경
- select() - 변수 (columns) 선택
- mutate() - 새로운 변수 만들기
- summarise() - 대표값 만들기
- group\_by() - 그룹별로 계산 수행
- join() - 두 tibble 또는 data.frame을 병합할 때 사용

이 함수들은 %>%와 함께 쓰이면서 강력한 성능을 발휘합니다. summarise 함수는 특정 값들의 통계 값을 계산해 주는 함수이며 그 외 함수들은 행렬 편집을 위한 함수들로 보시면 되겠습니다. 간단한 예제를 수행하면서 각각의 기능을 살펴보고 왜 dplyr이 널리 사용되고 그 장점이 무엇인지 파악해 보도록 하겠습니다.

iris 데이터는 세 종류의 iris 품종에 대한 꽃잎과 꽃받침의 length와 width를 측정해 놓은 데이터 입니다. head와 str 명령어를 %>%를 이용해서 데이터를 살펴 봅니다.

```
library(tidyverse)

iris %>% head(10)
iris %>% str
```

### 8.6.1 filter

먼저 아래와 같이 `filter` 함수를 사용해서 원하는 조건의 데이터 (샘플)을 골라낼 수 있습니다.

```
library(dplyr)

head(iris)
iris %>% filter(Species=="setosa")
iris %>% filter(Species=="setosa" | Species=="versicolor")
iris %>% filter(Species=="setosa" & Species=="versicolor")
iris %>%
  filter(Species=="setosa" | Species=="versicolor") %>%
  dim
```

`filter`의 ,로 구분되는 매개변수는 and 로직으로 묶인 조건입니다. 지난 강좌에서 보셨듯 R에서 and는 `&`, or는 `|`, 그리고 not은 `!` 으로 사용하면 되며 `filter`에서 ,로 구분된 조건은 and와 같다고 보시면 되겠습니다.

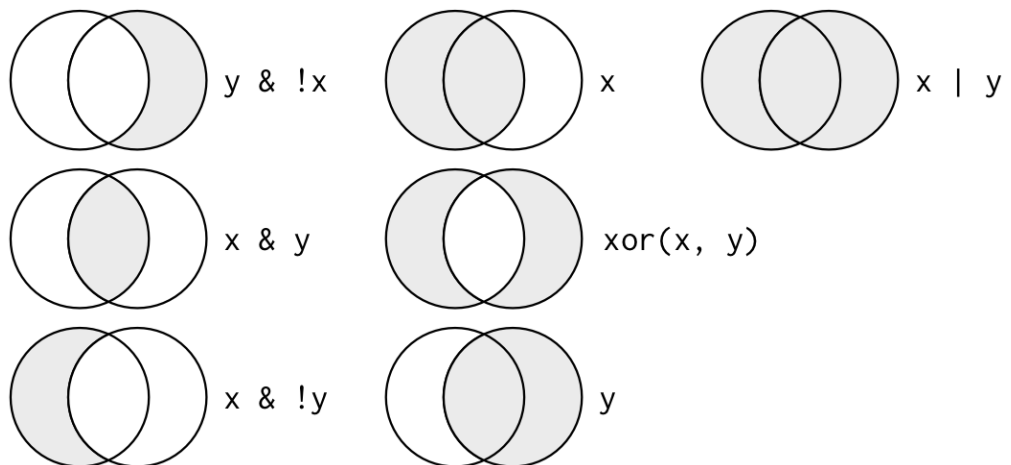


Image from (<https://r4ds.had.co.nz/>)

### 8.6.2 arrange

`arrange()`는 지정된 변수를 기준으로 값의 크기순서로 샘플들의 배열 순서 즉, row의 순서를 바꾸는 기능을 수행합니다. 기본으로 크기가 커지는 순서로 정렬이 진행되며 작아지는 순서를 원할 경우 `desc` 함수를 사용할 수 있습니다.

```
iris %>% arrange(Sepal.Length)
iris %>% arrange(desc(Sepal.Length))
iris %>% arrange(Sepal.Length, Sepal.Width)
```

### 8.6.3 select

`select()` 는 주어진 데이터셋으로부터 관심있는 변수를 (column) 선택하여 보여줍니다. 다음 helper 함수들은 `select` 함수와 같이 유용하게 쓰일 수 있습니다.

`starts_with("abc")` - "abc" 로 시작하는 문자열을 갖는 변수 이름  
`ends_with("xyz")` - "xyz"으로 끝나는 문자열을 갖는 변수 이름  
`contains("ijk")` - "ijk" 문자열을 포함하는 변수 이름  
`matches("(.)w1")` - 정규식, 반복되는 문자

```
head(iris)
iris %>% select(Species, everything()) %>% head(5)
iris %>% select(Species, everything())
iris %>% select(-Species)
iris %>% select(starts_with('S'))
iris %>% select(obs = starts_with('S'))
```

아래는 `matches` 함수를 사용한 방법입니다. 좀 더 복잡한 패턴을 적용하여 변수들을 선택할 수 있으며 `grep` 함수를 사용할 경우도 정규식 패턴을 적용할 수 있습니다.

```
iris2 <- rename(iris, aavar = Petal.Length)
select(iris2, matches("(.)\\1"))
tmp <- iris[,3:5]
colnames(iris)[grep("^S", colnames(iris))]
iris[,grep("^S", colnames(iris))]
tmp
```

아래 `(.)\\1`은 하나의 문자 .가 (어떤 문자든) 한 번 더 \\1 사용된 변수 이름을 말하며 이는 `aavar`의 `aa`밖에 없으므로 `aavar`가 선택됩니다. `grep`에서 `^` 표시는 맨 처음을 나타내므로 `^S`는 S로 시작하는 문자가 되겠습니다. 따라서 `grep("^S", colnames(iris))`의 경우 컬럼 이름 중 S로 시작하는 이름은 `True`로 그렇지 않으면 `False` 값을 리턴합니다.

### 8.6.4 mutate

`mutate()` 함수는 새로운 변수를 추가할 수 있는 기능을 제공하며 앞에서 배웠던 `within()`과 비슷하다고 볼 수 있습니다. 아래와 같이 `mutate` 함수는 `sepal_ratio`라는 변수를 새로 만들어서 기존 `iris` 데이터들과 함께 반환해 줍니다.

```
iris2 <- iris %>% mutate(sepal_ratio = Sepal.Length/Sepal.Width)
head(iris2)
```

### 8.6.5 summarise

`summarise()`는 `data.frame`내 특정 변수의 값들로 하나의 요약값/대푯값을 만들어 줍니다. `summarise` 함수는 단독으로 쓰이기 보다는 `group_by()` 기능과 병행해서 쓰이는 경우에 유용하게 쓰입니다. `summarise_all()` 함수를 사용하면 모든 변수에 대해서 지정된 함수를 실행합니다.

```
iris %>% summarise(mean(Sepal.Length), m=mean(Sepal.Width))
iris %>%
  group_by(Species) %>%
  summarise(mean(Sepal.Width))

iris %>%
  group_by(Species) %>%
  summarise_all(mean)

iris %>%
  group_by(Species) %>%
  summarise(across(everything(), mean))

iris %>%
  group_by(Species) %>%
  summarise_all(sd)

iris %>%
  group_by(Species) %>%
  summarise(across(everything(), sd))
```

### 8.6.6 join

join 함수는 데이터를 병합해주는 기능을 수행하는 함수입니다. 네 가지 종류의 함수가 있으며 (left\_join(), 'right\_join()', 'inner\_join()', 'full\_join()') (key) .by'에서 지정해준 파라미터의 값을 기준으로 기능이 수행 됩니다.

```
df1 <- data.frame(id=c(1,2,3,4,5,6), age=c(30, 41, 33, 56, 20, 17))
df2 <- data.frame(id=c(4,5,6,7,8,9), gender=c("f", "f", "m", "m", "f", "m"))

inner_join(df1, df2, by="id")
left_join(df1, df2, "id")
right_join(df1, df2, "id")
full_join(df1, df2, "id")

# vs.
cbind(df1, df2)
```

## 8.7 Code comparison

이제 split, apply, combine을 활용하여 평균을 구하는 코드와 dplyr 패키지를 사용하여 만든 코드를 비교해 보도록 하겠습니다. iris 데이터를 분석하여 품종별로

꽃받침의 길이 (Sepal.length)의 평균과 표준편차, 그리고 샘플의 수를 구해보는 코드입니다.

split은 factor형 변수인 Species를 기준으로 iris 데이터를 나누어 주는 역할을 하며 lapply는 list 형 데이터인 iris\_split을 각 리스트의 각각의 원소들에 대해서 임의의 함수 function(x)... 를 수행하는 역할을 합니다. 마지막 data.frame으로 최종 경로를 combine 합니다.

```
iris_split <- split(iris, iris$Species)
iris_means <- lapply(iris_split, function(x){mean(x$Sepal.Length)})
iris_sd <- lapply(iris_split, function(x){sd(x$Sepal.Length)})
iris_cnt <- lapply(iris_split, function(x){length(x$Sepal.Length)})
iris_df <- data.frame(unlist(iris_cnt), unlist(iris_means), unlist(iris_sd))
```

아래는 dplyr 패키지를 사용한 코드 입니다.

```
iris_df <- iris %>%
  group_by(Species) %>%
  summarise(n=n(), mean=mean(Sepal.Length), sd=sd(Sepal.Length))
```

위에서 보듯 dplyr 패키지를 사용할 경우 그 결과는 같으나 코드의 가독성과 효율성면에서 장점을 보여줍니다. iris 데이터를 받아서 Species에 명시된 그룹으로 나누고 원하는 함수를 타깃 컬럼에 대해서 적용하라는 의미 입니다. 다음은 모든 변수에 대한 평균을 구하는 코드 입니다.

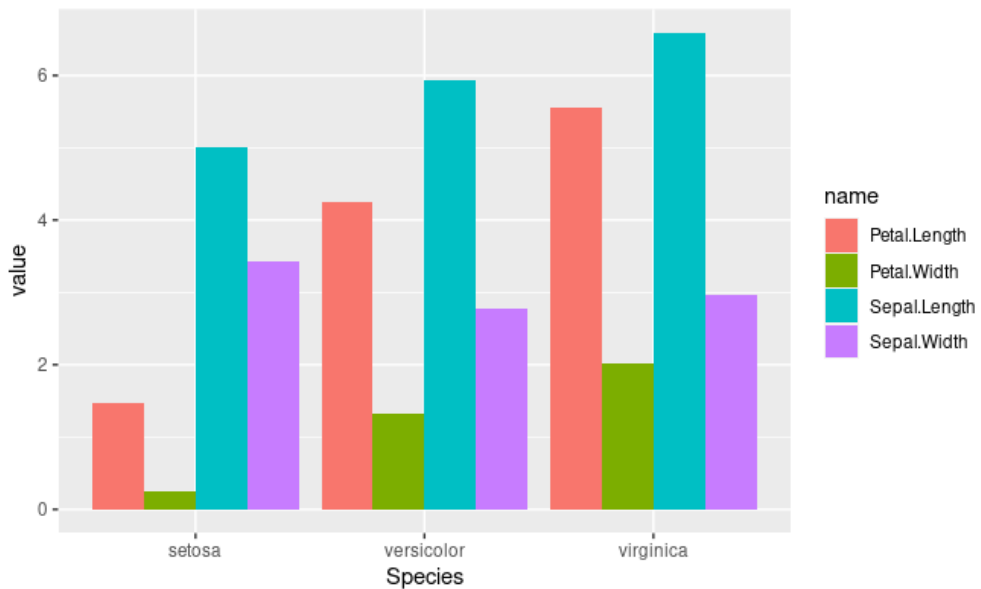
```
iris_mean_df <- iris %>%
  group_by(Species) %>%
  summarise(across(everything(), mean))
```

자세한 ggplot의 내용은 다음시간에 학습하겠지만 각 평균에 대한 막대그래프를 그려보겠습니다.

```
library(ggplot2)

iris_mean_df2 <- iris_mean_df %>%
  pivot_longer(-Species)

ggplot(iris_mean_df2, aes(x=Species, y=value, fill=name)) +
  geom_bar(stat="identity", position="dodge")
```



### Exercises

InsectSprays 데이터는 살충제 6종에 대한 살충력을 (죽은 벌래의 마릿수) 나타내는 데이터이다. 각 살충제별로 평균과 표준편차를 구하시오

### Exercises

dplyr 패키지의 starwars 는 스타워즈 영화에 나오는 등장인물들을 분석한 데이터셋이다. 종족에 따른 키의 평균과 표준편차를 구하시오. (NA 데이터는 제외하고 분석)

### Exercises

airquality 데이터는 뉴욕주의 몇몇 지점에서의 공기질을 측정한 데이터이다. 데이터에서 NA를 제거하고 각 월별로 평균 오존, 자외선, 풍속, 및 온도에 대한 평균과 표준편차를 구하시오

```
airmean <- airquality %>%
  filter(complete.cases(.)) %>%
  select(-Day) %>%
  group_by(Month) %>%
  summarise(across(everything(), mean)) %>%
  pivot_longer(-Month, values_to = "mean")

airsd <- airquality %>%
  filter(complete.cases(.)) %>%
  select(-Day) %>%
  group_by(Month) %>%
  summarise(across(everything(), sd)) %>%
```

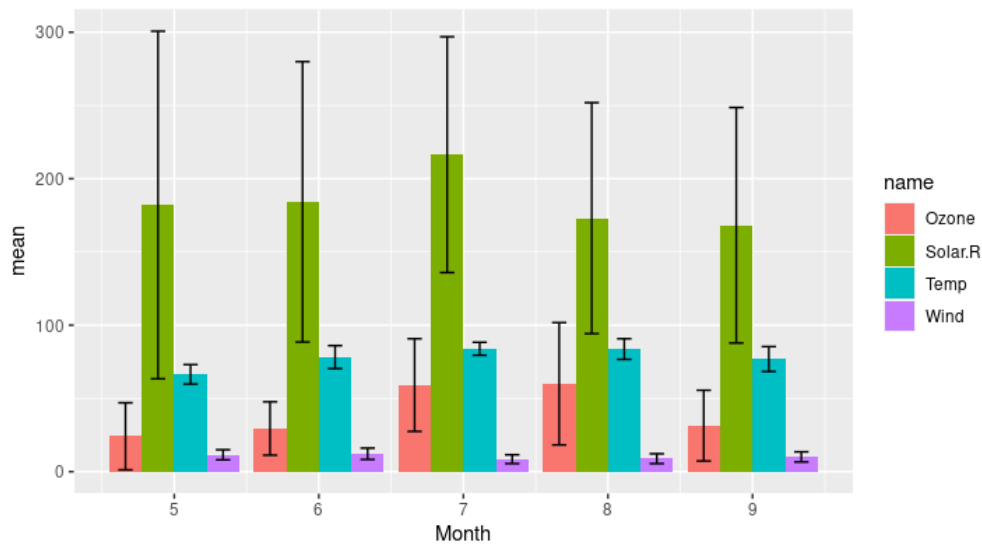


```
pivot_longer(~Month, values_to = "sd")
```

참고로 errorbar가 있는 막대그래프를 그려보겠습니다. 이를 위해서 먼저 두 테이블을 병합한 후 ggplot2 패키지의 ggplot 함수를 이용해서 그래프를 그립니다. 자세한 ggplot의 내용은 다음시간에 학습하겠습니다.

```
airdata <- left_join(airmean, airsd, by=c("Month", "name"))

ggplot(airdata, aes(x=Month, y=mean, fill=name)) +
  geom_bar(stat="identity", position="dodge") +
  geom_errorbar(aes(ymin=mean-sd, ymax=mean+sd), position=position_dodge(width=0.9), width=0.4)
```



이 저작물은 크리에이티브 커먼즈 저작자표시-비영리-변경금지 4.0 국제 라이선스에 따라 이용할 수 있습니다.