

5. Data Load & Save

데이터 가져오기

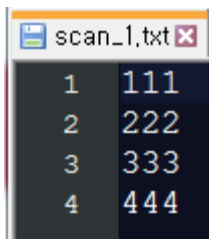
R에서 외부 파일의 데이터를 가져오기 위한 방법은 하기와 같다.

□ scan()

- 텍스트파일 읽어 배열에 저장하기
- c언어의 scanf()와 비슷한 역할
- 실수의 경우 소수점이 생략되어 입력됨
- 문자의 경우 what="" 옵션을 사용해야 함

□ 예제코드

```
setwd("d:")  
scan1<-scan('scan_1.txt')  
scan1
```



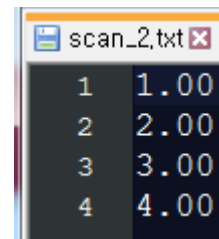
1	111
2	222
3	333
4	444

□ 결과

```
[1] 111 222 333 444
```

□ 예제코드

```
scan2<-scan('scan_2.txt')  
scan2  
scan2<-scan('scan_2.txt', what="")  
scan2
```



1	1.00
2	2.00
3	3.00
4	4.00

□ 결과

```
[1] 1 2 3 4  
[1] "1.00" "2.00" "3.00" "4.00"
```

□ 예제코드(숫자를 직접 입력하는 경우)

```
input<-scan()  
1: 1  
2: 2  
3: # 엔터키를 입력하면 입력 종료됨  
Read 2 items
```

데이터 가져오기

R에서 외부 파일의 데이터를 가져오기 위한 방법은 하기와 같다.

❑ `readline()`

- 함수로 한줄 읽어오기

```
> input<-readline("값을 입력하시오 : " )
```

```
값을 입력하시오. : R is very fun!!!
```

```
> input
```

```
[1] "R is very fun!!"
```

❑ `readLines()`

- 파일에서 데이터를 읽어 배열로 만들어주는 함수

```
> input<-readLines('scan_2.txt')
```

```
> input
```

```
[1] "1.00" "2.00" "3.00" "4.00"
```

❑ `read.table()`

- 텍스트 형태의 파일을 읽어서 데이터 프레임에 담는 함수

- 열구분이 공백인 데이터 파일 읽어오기

❑ `read.csv()`

- 열 구분이 ,(콤마)인 데이터 파일 읽어오기

- 열 구분이 tab으로 구분된 파일이면 `sep="\t"` 옵션 사용

데이터 가져오기

R에서 외부 파일의 데이터를 가져오기 위한 방법은 하기와 같다.

데이터 가져오기

□ 데이터 형태

- 읽어온 결과 값은 모두 Dataframe이 됨

□ 데이터 Package

- read.csv() : 열 구분이 콤마인 경우 사용
- read.table() : 열 구분이 공백인 경우 사용
- 엑셀파일은 지원 Package를 이용함
(32비트 : RODBC패키지, 64비트 : XLConnect 패키지 사용)

□ 예제코드

```
data <- read.csv("book.csv")  
data <- read.table("book.csv")  
data <- fread("book.csv")  
data <- read_excel("book.xlsx")
```

실행결과

```
Console Terminal x  
~/  
> a<-read.table('d:/del.csv',sep=' ',header=TRUE)  
> a  
      Sepal.Length Sepal.width Petal.Length Petal.width  Species  
1           5.1         3.5         1.4         0.2    setosa  
2           4.9         3.0         1.4         0.2    setosa  
3           4.7         3.2         1.3         0.2    setosa  
4           4.6         3.1         1.5         0.2    setosa  
5           5.0         3.6         1.4         0.2    setosa  
6           5.4         3.9         1.7         0.4    setosa  
7           4.6         3.4         1.4         0.3    setosa  
8           5.0         3.4         1.5         0.2    setosa  
9           4.4         2.9         1.4         0.2    setosa  
10          4.9         3.1         1.5         0.1    setosa  
11          5.4         3.7         1.5         0.2    setosa  
12          4.8         3.4         1.6         0.2    setosa
```

데이터 가져오기

R에서 클립보드의 데이터를 가져오기 위한 방법은 하기와 같다.

❑ 대상 데이터를 Clipboard에 복사

- 엑셀에서 원하는 부분을 복사 : ctrl+c

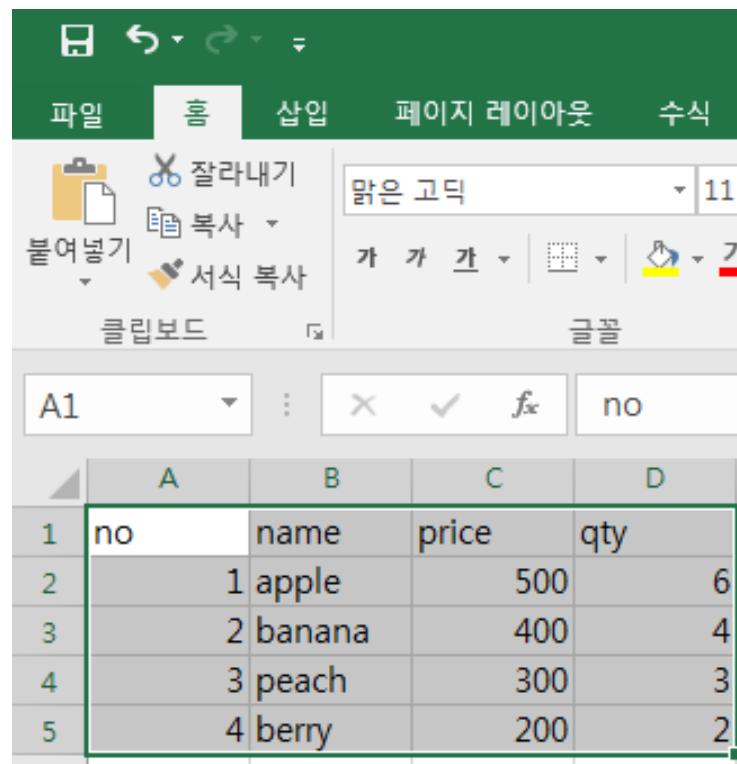
❑ 클립보드의 내용을 이용하여 데이터프레임 생성

> fruits<-read.delim("clipboard", header=T)

❑ 데이터 가져오기 결과

> fruits

```
no  name price qty
1 1 apple  500  6
2 2 banana 400  4
3 3 peach  300  3
4 4 berry  200  2
```



	A	B	C	D
1	no	name	price	qty
2	1	apple	500	6
3	2	banana	400	4
4	3	peach	300	3
5	4	berry	200	2

데이터 저장하기 - Flat파일

데이터를 Flat파일로 저장하기 위한 방법은 하기와 같다.

데이터 저장하기

□ 데이터 형태

- 읽어온 결과 값을 일반 파일로 저장함

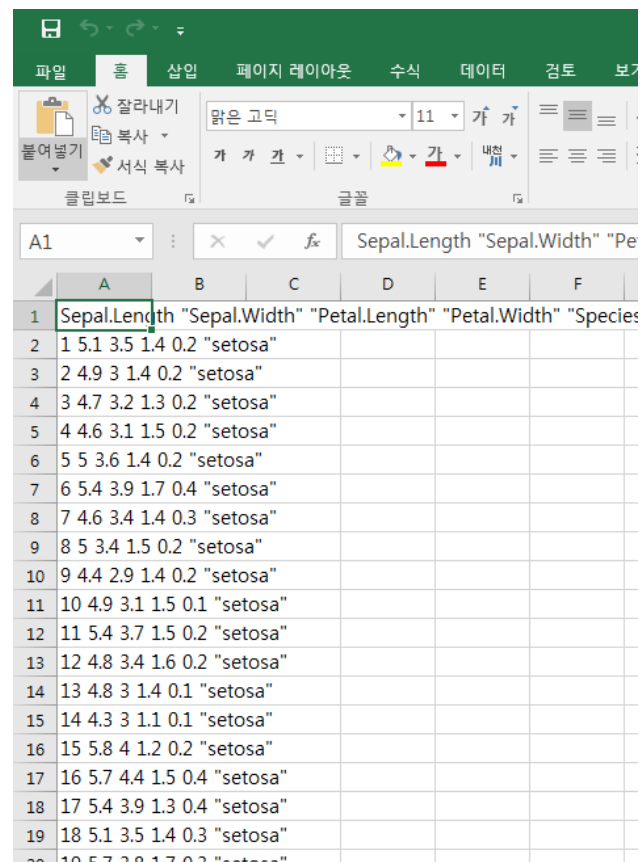
□ 명령

- write.table(파일명, 저장옵션)
- 자주사용하는 저장옵션
 - sep : 열 구분자(' ', ',')
 - header : 열명 출력 유무(header=TRUE)

□ 예제코드

```
write.table('d:/temp/a.csv')
```

실행결과



	A	B	C	D	E	F
1	Sepal.Length	"Sepal.Width"	"Petal.Length"	"Petal.Width"	"Species"	
2	1	5.1	3.5	1.4	0.2	"setosa"
3	2	4.9	3	1.4	0.2	"setosa"
4	3	4.7	3.2	1.3	0.2	"setosa"
5	4	4.6	3.1	1.5	0.2	"setosa"
6	5	5	3.6	1.4	0.2	"setosa"
7	6	5.4	3.9	1.7	0.4	"setosa"
8	7	4.6	3.4	1.4	0.3	"setosa"
9	8	5	3.4	1.5	0.2	"setosa"
10	9	4.4	2.9	1.4	0.2	"setosa"
11	10	4.9	3.1	1.5	0.1	"setosa"
12	11	5.4	3.7	1.5	0.2	"setosa"
13	12	4.8	3.4	1.6	0.2	"setosa"
14	13	4.8	3	1.4	0.1	"setosa"
15	14	4.3	3	1.1	0.1	"setosa"
16	15	5.8	4	1.2	0.2	"setosa"
17	16	5.7	4.4	1.5	0.4	"setosa"
18	17	5.4	3.9	1.3	0.4	"setosa"
19	18	5.1	3.5	1.4	0.3	"setosa"
20	19	5.7	3.8	1.7	0.3	"setosa"

데이터 저장하기 - 엑셀파일

데이터를 엑셀파일로 저장하기 위한 방법은 하기와 같다.

데이터 저장하기

□ Package

- XLConnect

□ 데이터 Package 설치

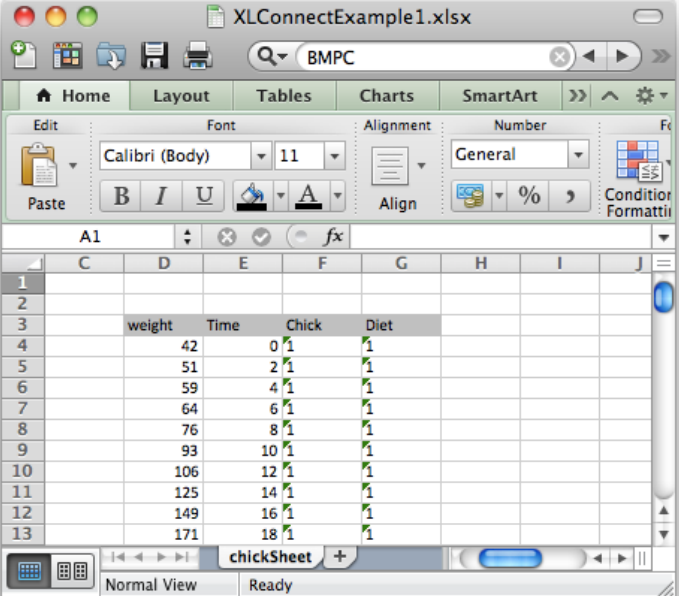
- `install.packages("XLConnect")`

□ 예제코드

```
require(XLConnect)
wb <- loadWorkbook("XLConnectExample1.xlsx", create = TRUE)
createSheet(wb, name = "chickSheet")
writeWorksheet(wb, ChickWeight, sheet = "chickSheet", startRow = 3, startCol = 4)
saveWorkbook(wb)
```

```
require(XLConnect) #보다 간단하게 코드를 작성한 사례
writeWorksheetToFile("XLConnectExample2.xlsx", data = ChickWeight,
  sheet = "chickSheet", startRow = 3, startCol = 4)
```

실행결과



	weight	Time	Chick	Diet
1				
2				
3				
4	42	0	1	1
5	51	2	1	1
6	59	4	1	1
7	64	6	1	1
8	76	8	1	1
9	93	10	1	1
10	106	12	1	1
11	125	14	1	1
12	149	16	1	1
13	171	18	1	1

R Cheat sheet - 데이터 불러오기/저장하기

R's **tidyverse** is built around **tidy data** stored in **tibbles**, which are enhanced data frames.



The front side of this sheet shows how to read text files into R with **readr**.



The reverse side shows how to create tibbles with **tibble** and to layout tidy data with **tidyr**.

OTHER TYPES OF DATA

Try one of the following packages to import other types of files

- **haven** - SPSS, Stata, and SAS files
- **readxl** - excel files (.xls and .xlsx)
- **DBI** - databases
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)

Read Tabular Data - These functions share the common arguments:

```
read_*(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"),  
quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000,  
n_max), progress = interactive())
```

a,b,c
1,2,3
4,5,NA

A	B	C
1	2	3
4	5	NA

Comma Delimited Files

read_csv("file.csv")

To make file.csv run:

write_file(x = "a,b,c\n1,2,3\n4,5,NA", path = "file.csv")

a;b;c
1;2;3
4;5;NA

A	B	C
1	2	3
4	5	NA

Semi-colon Delimited Files

read_csv2("file2.csv")

write_file(x = "a;b;c\n1;2;3\n4;5;NA", path = "file2.csv")

a|b|c
1|2|3
4|5|NA

A	B	C
1	2	3
4	5	NA

Files with Any Delimiter

read_delim("file.txt", delim = "|")

write_file(x = "a|b|c\n1|2|3\n4|5|NA", path = "file.txt")

a b c
1 2 3
4 5 NA

A	B	C
1	2	3
4	5	NA

Fixed Width Files

read_fwf("file.fwf", col_positions = c(1, 3, 5))

write_file(x = "a b c\n1 2 3\n4 5 NA", path = "file.fwf")

Tab Delimited Files

read_tsv("file.tsv") Also **read_table()**.

write_file(x = "a\tb\tc\n1\t2\t3\n4\t5\tNA", path = "file.tsv")

USEFUL ARGUMENTS

a,b,c
1,2,3
4,5,NA

Example file

write_file("a,b,c\n1,2,3\n4,5,NA", "file.csv")
f <- "file.csv"

1	2	3
4	5	NA

Skip lines

read_csv(f, skip = 1)

A	B	C
1	2	3
4	5	NA

No header

read_csv(f, col_names = FALSE)

A	B	C
1	2	3

Read in a subset

read_csv(f, n_max = 1)

x	y	z
1	2	3
4	5	NA

Provide header

read_csv(f, col_names = c("x", "y", "z"))

A	B	C
NA	2	3
4	5	NA

Missing Values

read_csv(f, na = c("1", "!"))

Read Non-Tabular Data

Read a file into a single string

read_file(file, locale = default_locale())

Read each line into its own string

**read_lines(file, skip = 0, n_max = -1L, na = character(),
locale = default_locale(), progress = interactive())**

Read Apache style log files

read_log(file, col_names = FALSE, col_types = NULL, skip = 0, n_max = -1, progress = interactive())

Read a file into a raw vector

read_file_raw(file)

Read each line into a raw vector

**read_lines_raw(file, skip = 0, n_max = -1L,
progress = interactive())**

Data types

readr functions guess the types of each column and convert types when appropriate (but will NOT convert strings to factors automatically).

A message shows the type of each column in the result.

```
## Parsed with column specification:  
## cols(  
##   age = col_integer(),  
##   sex = col_character(),  
##   earn = col_double()  
## )
```

age is an integer

earn is a double (numeric)

sex is a character

1. Use **problems()** to diagnose problems
x <- read_csv("file.csv"); problems(x)

2. Use a **col_** function to guide parsing

- **col_guess()** - the default
- **col_character()**
- **col_double()**, **col_euro_double()**
- **col_datetime()** (format = "") Also **col_date()** (format = ""), **col_time()** (format = "")
- **col_factor()** (levels, ordered = FALSE)
- **col_integer()**
- **col_logical()**
- **col_number()**, **col_numeric()**
- **col_skip()**

```
x <- read_csv("file.csv", col_types = cols(  
  A = col_double(),  
  B = col_logical(),  
  C = col_factor()))
```

3. Else, read in as character vectors then parse with a **parse_** function.

- **parse_guess()**
- **parse_character()**
- **parse_datetime()** Also **parse_date()** and **parse_time()**
- **parse_double()**
- **parse_factor()**
- **parse_integer()**
- **parse_logical()**
- **parse_number()**

```
x$A <- parse_number(x$A)
```