충남대학교
Chungnam National University

투자론

- R과 Excel을 통한 금융데이터 분석 -

7주차
R(입문 및 데이터 포맷)과 Data Management

충남대학교
장호규 교수

Unit 02

Data with R

Overview

- Importing and Exporting Data
- Data Frame

Data with R

Importing and Exporting Data

- R works most easily with datasets stored as text files.
  Typically, values in text files are separated, or delimited,

```
gender id race ses schtyp prgtype read write math science socst
0 70 4 1 1 general 57 52 41 47 57
1 121 4 2 1 vocati 68 59 53 63 31
0 86 4 3 1 general 44 33 54 58 31
0 141 4 3 1 vocati 63 44 47 53 56
```

- Or by commas (CSV file):

```
gender,id,race,ses,schtyp,prgtype,read,write,math,science,socst
0,70,4,1,1,general,57,52,41,47,57
1,121,4,2,1,vocati,68,59,53,63,61
0,86,4,3,1,general,44,33,54,58,31
0,141,4,3,1,vocati,63,44,47,53,56
```

Reading in Text Data

- R provides several related functions to read data stored as files. Use `read.csv()` to read in data stored as CSV and `read.delim()` to read in text data delimited by other characters (such as tabs or spaces)

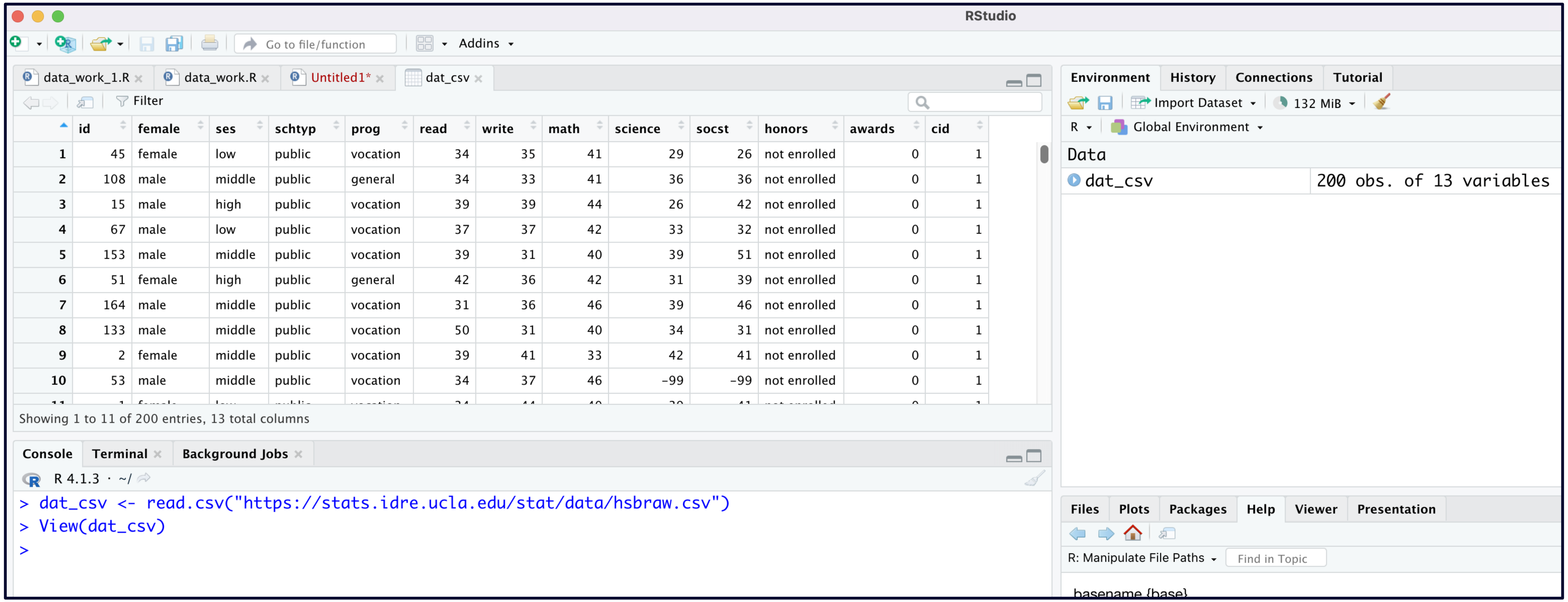
- For `read.delim()`, specify the delimiter in the sep= argument

- Both `read.csv()` and `read.delim()` assume the first row of the text file is a row of variable names. If this is not true, use the argument header=FALSE

Example

- `data_ibes <- read.csv("C:\\Users\\Hogyu Jhang\\Desktop\\ibes.csv")`
- `data_cf <-read.csv("/Users/hogyujhang/Dropbox/Emmanuel and Hogyu/our working paper/code/cashflow.csv")`
- `dat.tab <- read.delim("/path/to/file.txt", sep="\t")`
- `dat_csv <- read.csv("https://stats.idre.ucla.edu/stat/data/hsbraw.csv")`

Example



	id	female	ses	schtyp	prog	read	write	math	science	socst	honors	awards	cid
1	45	female	low	public	vocation	34	35	41	29	26	not enrolled	0	1
2	108	male	middle	public	general	34	33	41	36	36	not enrolled	0	1
3	15	male	high	public	vocation	39	39	44	26	42	not enrolled	0	1
4	67	male	low	public	vocation	37	37	42	33	32	not enrolled	0	1
5	153	male	middle	public	vocation	39	31	40	39	51	not enrolled	0	1
6	51	female	high	public	general	42	36	42	31	39	not enrolled	0	1
7	164	male	middle	public	vocation	31	36	46	39	46	not enrolled	0	1
8	133	male	middle	public	vocation	50	31	40	34	31	not enrolled	0	1
9	2	female	middle	public	vocation	39	41	33	42	41	not enrolled	0	1
10	53	male	middle	public	vocation	34	37	46	-99	-99	not enrolled	0	1

Showing 1 to 11 of 200 entries, 13 total columns

Data

dat_csv	200 obs. of 13 variables

```
> dat_csv <- read.csv("https://stats.idre.ucla.edu/stat/data/hsbraw.csv")
> View(dat_csv)
>
```

Exporting Data

- We can export our data to a .csv file with `write.csv()`.
- If you need to save multiple objects from your session, you can save whatever objects you need with `save()`, which creates a binary `.Rdata` file, which can be loaded for later use with `load()`.
- Example
  - Write a csv file: `write.csv(dat_csv, file = "path/to/save/filename.csv")`
  - Save an .Rdata file: `save(dat_csv, mydata, file="path/to/save/filename.rda")`
  - Package to read and write data in other software formats:
    - `readxl`: Excel files
    - `haven`: Stata, SAS, and SPSS

Data Frames

- Data sets for statistical analysis are typically stored in data frames in R. The objects created by `read.csv()` and `read.table()` are data frames
- Data frames are rectangular, where the columns are variables and the rows are observations of those variables
- Data frame columns can be of different data types (some double, some character, etc.) – but they must be equal length
- Real datasets usually combine variables of different types, so data frames are well suited for storage

Data Frames

Name	Weight	Height	Age	Disease
John	185	69	34.5	TRUE
Emily	150	62	55.6	FALSE
Mary	120	65	21.1	TRUE
Dan	225	72	51.1	FALSE

Each row is an observaion

Two-dimensional

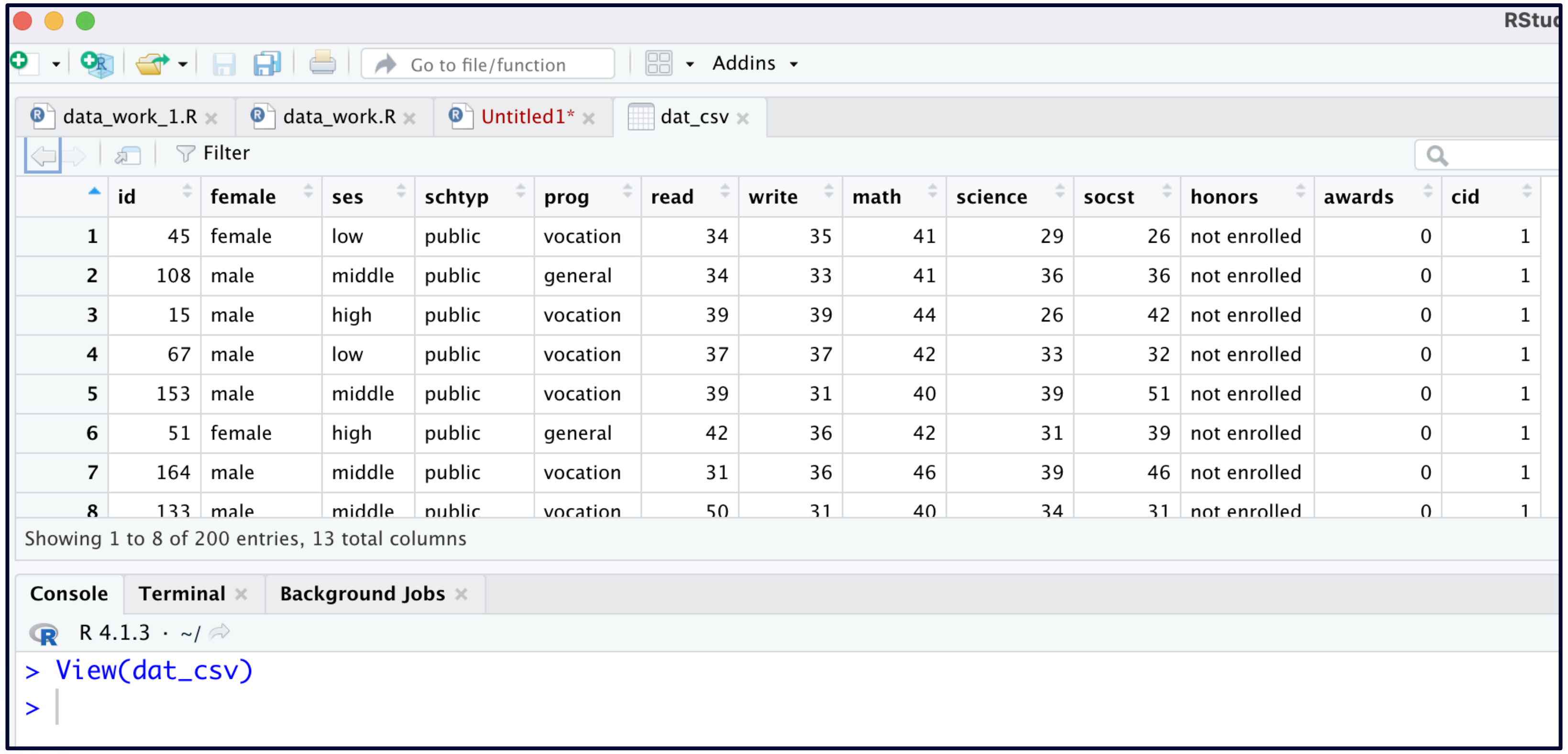
Heterogeneous

Rectangular

Each column vector is a variable

Viewing Data

- Use `View()` on a dataset to open a spreadsheet-style view of a dataset: `View(dat_csv)`



	id	female	ses	schtyp	prog	read	write	math	science	socst	honors	awards	cid
1	45	female	low	public	vocation	34	35	41	29	26	not enrolled	0	1
2	108	male	middle	public	general	34	33	41	36	36	not enrolled	0	1
3	15	male	high	public	vocation	39	39	44	26	42	not enrolled	0	1
4	67	male	low	public	vocation	37	37	42	33	32	not enrolled	0	1
5	153	male	middle	public	vocation	39	31	40	39	51	not enrolled	0	1
6	51	female	high	public	general	42	36	42	31	39	not enrolled	0	1
7	164	male	middle	public	vocation	31	36	46	39	46	not enrolled	0	1
8	133	male	middle	public	vocation	50	31	40	34	31	not enrolled	0	1

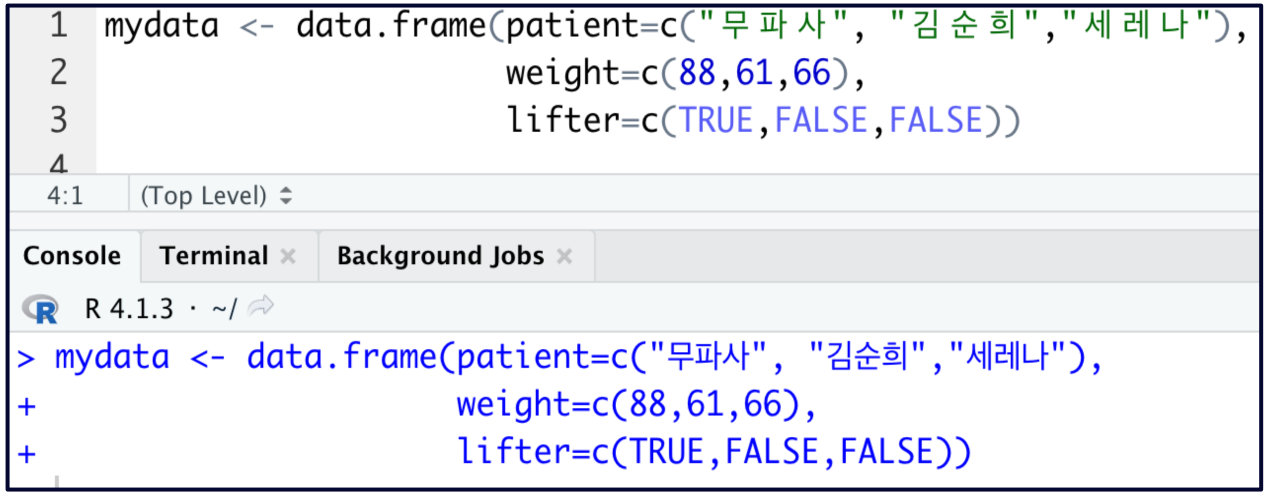
Showing 1 to 8 of 200 entries, 13 total columns

```
> View(dat_csv)
> 
```

Subsetting Data Frames

- With a two-dimensional structure, data frames can be subset with matrix notation `[rows, columns]`
- Use vectors to subset multiple rows/columns
- Omitting rows or columns specifies all rows and columns, respectively

Subsetting Data Frames



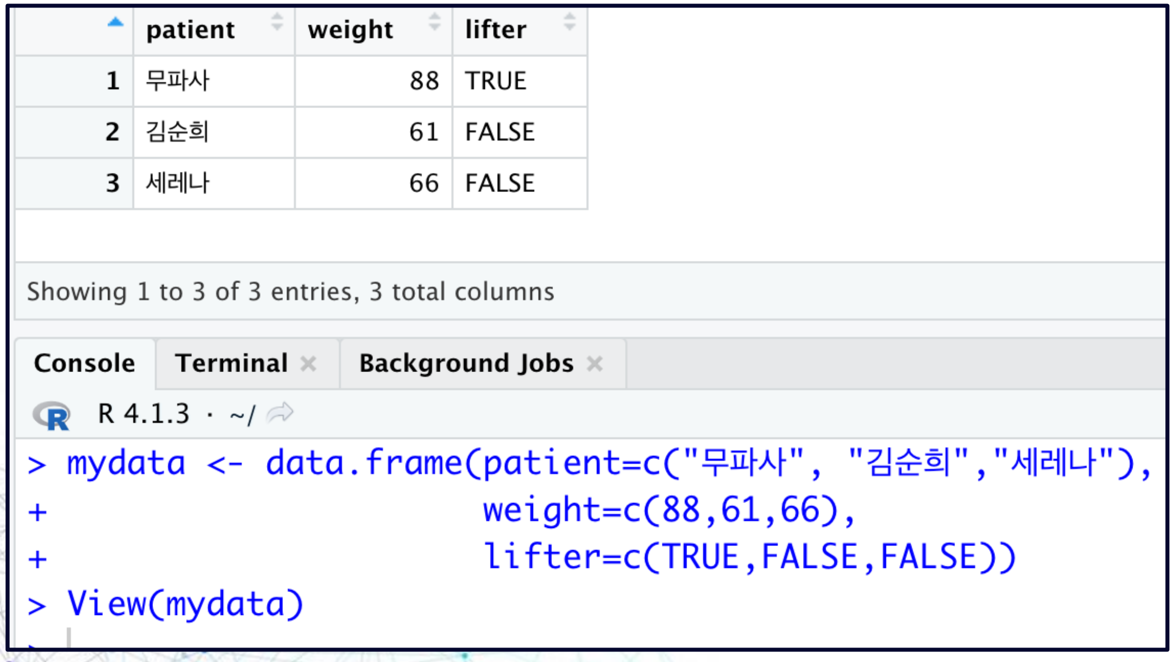
```
mydata <- data.frame(patient=c("무 파 사", "김 순 희","세 레 나"),
                     weight=c(88,61,66),
                     lifter=c(TRUE,FALSE,FALSE))

4:1   (Top Level) ⇕
```

Console | Terminal × | Background Jobs ×

R 4.1.3 · ~/

```
> mydata <- data.frame(patient=c("무파사", "김순희","세레나"),
+                      weight=c(88,61,66),
+                      lifter=c(TRUE,FALSE,FALSE))
```



	patient	weight	lifter
1	무파사	88	TRUE
2	김순희	61	FALSE
3	세레나	66	FALSE

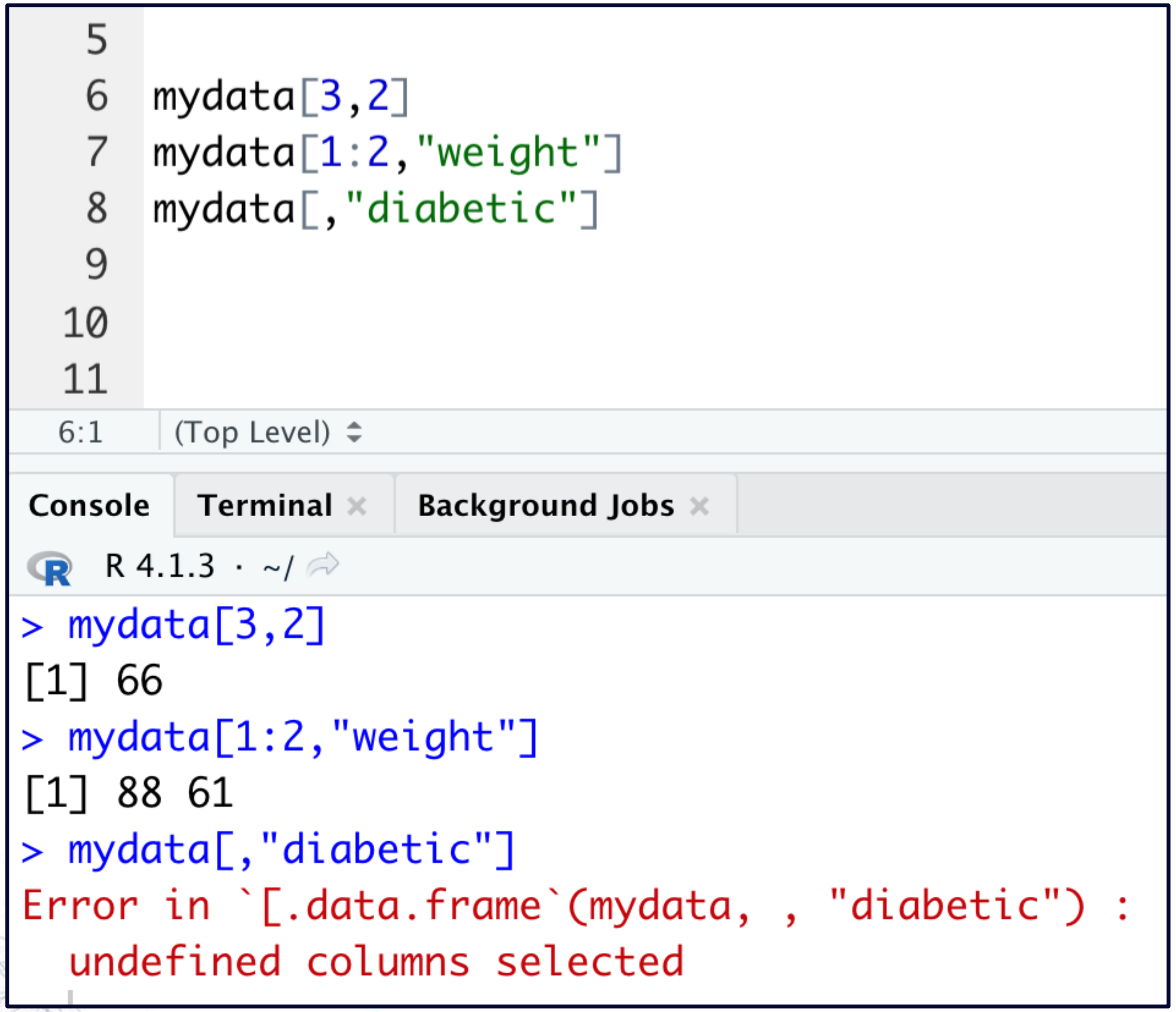
Showing 1 to 3 of 3 entries, 3 total columns

Console | Terminal × | Background Jobs ×

R 4.1.3 · ~/

```
> mydata <- data.frame(patient=c("무파사", "김순희","세레나"),
+                      weight=c(88,61,66),
+                      lifter=c(TRUE,FALSE,FALSE))
> View(mydata)
```

Subsetting Data Frames



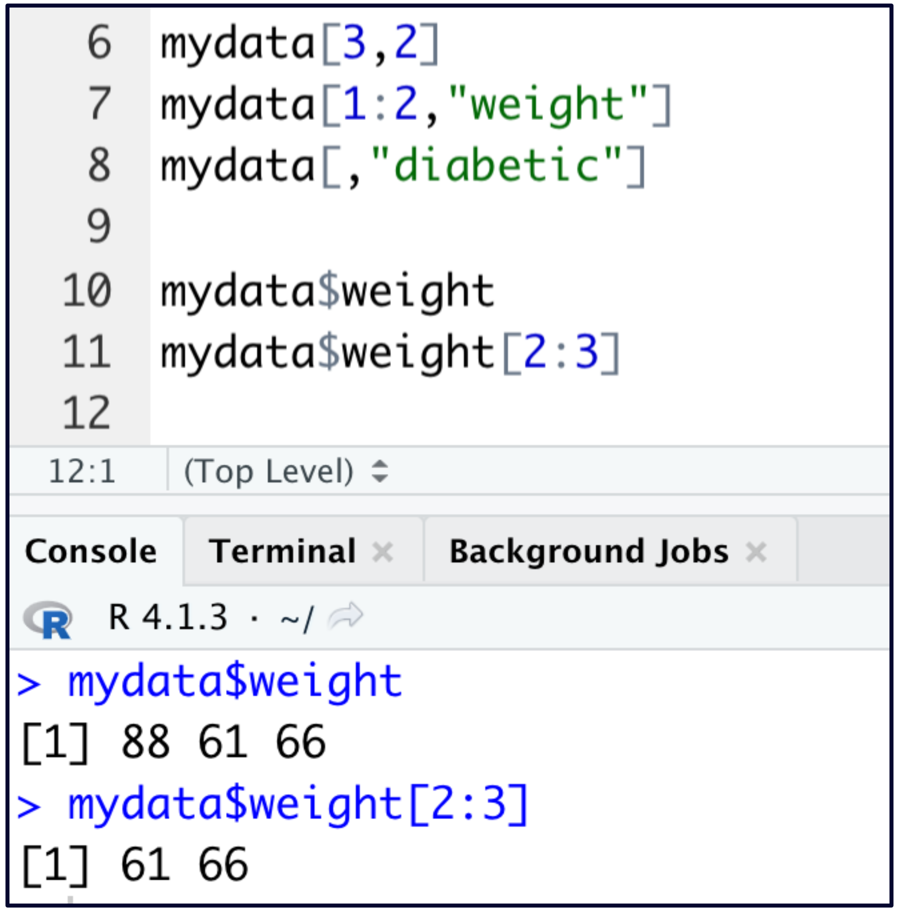
```
mydata[3,2]
mydata[1:2,"weight"]
mydata[,"diabetic"]
```

6:1 (Top Level)

Console Terminal Background Jobs

R 4.1.3 · ~/

```
> mydata[3,2]
[1] 66
> mydata[1:2,"weight"]
[1] 88 61
> mydata[,"diabetic"]
Error in `[.data.frame`(mydata, , "diabetic") : 
  undefined columns selected
```



```
mydata[3,2]
mydata[1:2,"weight"]
mydata[,"diabetic"]

mydata$weight
mydata$weight[2:3]
```

12:1 (Top Level)

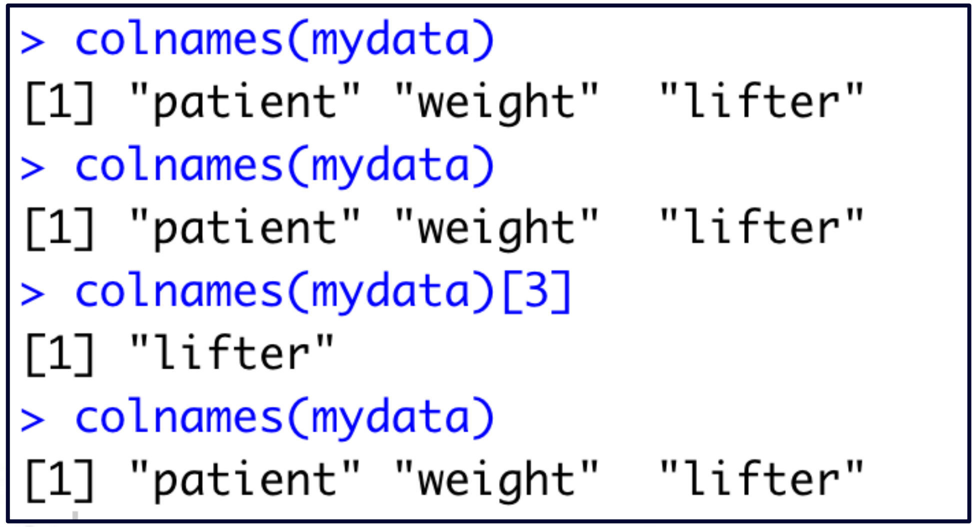
Console Terminal Background Jobs

R 4.1.3 · ~/

```
> mydata$weight
[1] 88 61 66
> mydata$weight[2:3]
[1] 61 66
```

Naming Data Frame Columns

- `colnames(data_frame)` returns the column names of data_frame (or matrix)
- `colnames(data_frame)<- c("some", "names")` assigns column names to data_frame

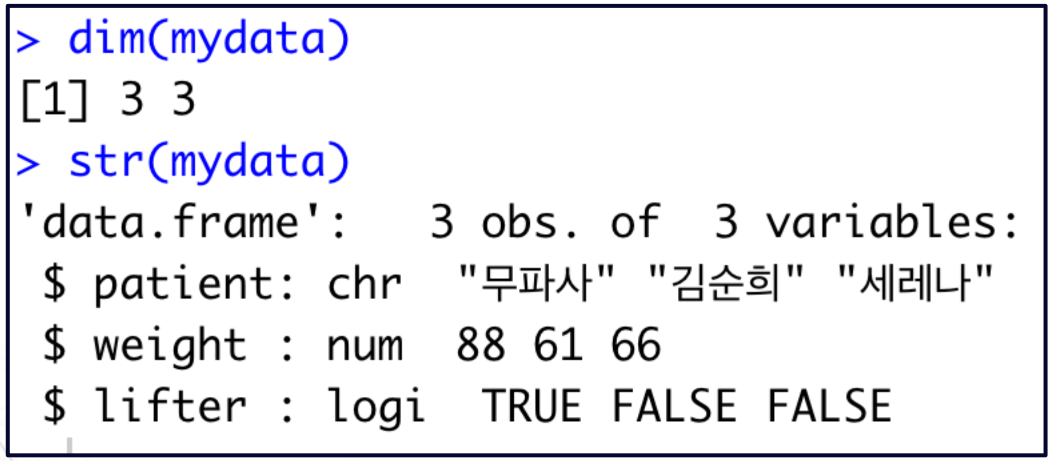


```
> colnames(mydata)
[1] "patient" "weight"  "lifter"
> colnames(mydata)
[1] "patient" "weight"  "lifter"
> colnames(mydata)[3]
[1] "lifter"
> colnames(mydata)
[1] "patient" "weight"  "lifter"
```

Examining the Structure of an Object

- Use `dim()` on two-dimensional objects to get the number of rows and columns

- Use `str()`, to see the structure of the object, including its class and the data types of elements. We also see the first few rows of each variable



```
> dim(mydata)
[1] 3 3
> str(mydata)
'data.frame':	3 obs. of  3 variables:
 $ patient: chr  "무파사" "김순희" "세레나"
 $ weight : num  88 61 66
 $ lifter : logi  TRUE FALSE FALSE
```

Adding New Variables to the Data Frame

- You can add variables to data frames by declaring them to be column variables of the data frame as they are created.
- Trying to add a column of the wrong length will result in an error.

```
> mydata$logWeight <- log(mydata$weight)
> colnames(mydata)
[1] "patient"   "weight"    "lifter"    "logWeight"
> mydata$z <- rep(0,5)
Error in `$<-.data.frame`(`*tmp*`, z, value = c(0, 0, 0, 0, 0)) :
  replacement has 5 rows, data has 3
```

Some Useful Functions to Create Variables from Existing Ones

- `log()`: logarithm
- `min_rank()`: rank values
- `cut()`: cut a continuous variable into intervals with new integer value signifying into which interval original value falls

  `scale()`: standardizes variable (substracts mean and divides by standard deviation)
- `lag()`, `lead()`: lag and lead a variable
- **cumsum():** cumulative sum
- `rowMeans()`, `rowSums()`: means and sums of several columns