Lecture 5 Data management: Part II - Reshaping data

This section describes how to prepare data for further analysis. There are situations when we need the **data frame** in a format that is different from the format in which we received it.

- Subsetting data
- Merging data
- aggregate( )

5.1 Subset data

Common tasks

- Select/delete columns
- Select/delete rows with or without conditions
- Select columns and rows with or without conditions

Using

- $ and [,]
- subset( ) Very powerful!
- dplyr package

Pick your favorite - one is enough.

In [1]:

```
df <- data.frame(names = c("Lucy", "John", "Mark", "Candy"),
                 score = c(67, 56, 87, 91))
for (i in 1:4){
    df$student.no[i] <- paste("student", i)
    df$pass[i] <- ifelse(df$score[i] >= 60, TRUE, FALSE)
}
df
```

A data.frame: 4 × 4

names	score	student.no	pass
<fct>	<dbl>	<chr>	<lgl>
Lucy	67	student 1	TRUE
John	56	student 2	FALSE
Mark	87	student 3	TRUE
Candy	91	student 4	TRUE

5.1.1 $ and [,]

Can only pick one variable.

In [2]:

```
names(df)
```

'names' 'score' 'student.no' 'pass'

In [3]:

```
# Recall the indexing system in R
df$names   # Select one variable
```

Lucy John Mark Candy

▶ **Levels**:

In [4]:

```
# Delete one variable
df.copy <- df
df.copy$names <- NULL
df.copy
```

A data.frame: 4 × 3

score	student.no	pass
<dbl>	<chr>	<lgl>
67	student 1	TRUE
56	student 2	FALSE
87	student 3	TRUE
91	student 4	TRUE

In [5]:

```
df[, 2]
```

67 56 87 91

In [6]:

```
df[ , "score"]
```

67 56 87 91

In [7]:

```
str(df[ , "score"])   # 1D vector
```

```
 num [1:4] 67 56 87 91
```

In [8]:

```
df[ , "score", drop = FALSE]
str(df[ , "score", drop = FALSE])   # 4 x 1 data frame
# The argument "drop = FALSE" maintains the original dimension
# The default is true
```

A

data.frame:

4 × 1

score
<dbl>
67
56
87
91

```
'data.frame':   4 obs. of  1 variable:
 $ score: num  67 56 87 91
```

In [9]:

```
df[1, ]
str(df[1, ])   # 1 x 4 data frame
# Can we drop a dimension here? Why?
```

A data.frame: 1 × 4

names	score	student.no	pass
<fct>	<dbl>	<chr>	<lgl>
Lucy	67	student 1	TRUE

```
'data.frame':   1 obs. of  4 variables:
 $ names     : Factor w/ 4 levels "Candy","John",..:
3
 $ score     : num 67
 $ student.no: chr "student 1"
 $ pass      : logi TRUE
```

In [10]:

```
df[1, , drop = TRUE]
```

$names

Lucy

▶ **Levels**:

$score

67

$student.no

'student 1'

$pass

TRUE

Any advantage of an n x 1 data frame over a vector of length n? <==> Is the drop argument useful?

In [11]:

```
# Delete variable "names" + reorder columns
df[ , c("student.no", "score", "pass")]
```

A data.frame: 4 × 3

student.no	score	pass
<chr>	<dbl>	<lgl>
student 1	67	TRUE
student 2	56	FALSE
student 3	87	TRUE
student 4	91	TRUE

In [12]:

```
# Select rows that passed
df[df$pass == TRUE, ]
```

A data.frame: 3 × 4

	names	score	student.no	pass
	<fct>	<dbl>	<chr>	<lgl>
1	Lucy	67	student 1	TRUE
3	Mark	87	student 3	TRUE
4	Candy	91	student 4	TRUE

In [13]:

```
# Delete variable
df[ , -c(1, 2)]   # Delete the 1st and 2nd
```

A data.frame: 4 × 2

student.no	pass
<chr>	<lgl>
student 1	TRUE
student 2	FALSE
student 3	TRUE
student 4	TRUE

In [14]:

```
# I believe that this used to work, but not anymore.
# df[ , -c("names", "score")]

# Now
drop <- c("names", "score")
df[ , !names(df) %in% drop]
```

A data.frame: 4 × 2

student.no	pass
<chr>	<lgl>
student 1	TRUE
student 2	FALSE
student 3	TRUE
student 4	TRUE

In [15]:

```
select = c("student.no", "pass")
df[ , names(df) %in% select]
```

A data.frame: 4 × 2

student.no	pass
<chr>	<lgl>
student 1	TRUE
student 2	FALSE
student 3	TRUE
student 4	TRUE

In [16]:

```
# How does this work?
1 %in% c(1, 3, 5)
"b" %in% c("a", "c", "e")
1:10 %in% c(1, 3, 5)
```

TRUE

FALSE

TRUE FALSE TRUE FALSE TRUE FALSE FALSE FALSE
FALSE FALSE

a %in% b checks whether $a \in b$ for every single entry in a.

Exercise: show the name and score of those who passed except Lucy.

In [ ]:

```

```

5.1.2 subset( )

```
subset(x, subset, select, drop = FALSE, ...)
```

In [17]:

```
# "select" argument selects columns
subset(df, select = c(student.no, pass))
```

A data.frame: 4 × 2

student.no	pass
<chr>	<lgl>
student 1	TRUE
student 2	FALSE
student 3	TRUE
student 4	TRUE

In [18]:

```
# Can also delete unwanted columns
subset(df, select = -c(names, score))
```

A data.frame: 4 × 2

student.no	pass
<chr>	<lgl>
student 1	TRUE
student 2	FALSE
student 3	TRUE
student 4	TRUE

In [19]:

```
# "subset" argument selects rows
# Can apply conditions
subset(df, subset = (score > 80))
```

A data.frame: 2 × 4

	names	score	student.no	pass
	<fct>	<dbl>	<chr>	<lgl>
3	Mark	87	student 3	TRUE
4	Candy	91	student 4	TRUE

In [20]:

```
# Now use both select and subset arguments to apply conditions
# Select the names of those who passed
subset(df, select = names, subset = (pass == TRUE))
```

A data.frame: 3 × 1

	names
	<fct>
1	Lucy
3	Mark
4	Candy

Note that all subsets are still data frames.

Exercise: show the name and score of those who passed except Lucy.

In [21]:

```
# Show the name and score of those who passed except Lucy(s).
# Recall logical operators &, | and !
```

5.1.3 dplyr package

```
'dplyr is a grammar of data manipulation'
```

https://dplyr.tidyverse.org (https://dplyr.tidyverse.org)

I do not use this package, or any other packages within the whole `tidyverse`.
https://www.tidyverse.org (https://www.tidyverse.org)

In [22]:

```
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats
':

    filter, lag

The following objects are masked from 'package:base'
:

    intersect, setdiff, setequal, union
```

In [23]:

```
# Show the name and score of those who passed except Lucy(s).
df.col <- filter(df, names != "Lucy" & pass == TRUE)
df.col
```

A data.frame: 2 × 4

names	score	student.no	pass
<fct>	<dbl>	<chr>	<lgl>
Mark	87	student 3	TRUE
Candy	91	student 4	TRUE

In [24]:

```
df.final <- select(df.col, c(names, score))
df.final
```

A data.frame: 2 × 2

names	score
<fct>	<dbl>
Mark	87
Candy	91

dplyr cheetsheet

https://rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf (https://rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf)

5.2 Merge data

5.2.1 Add cases/observations to a data frame

This is basically adding rows.

In [25]:

```
df
```

A data.frame: 4 × 4

names	score	student.no	pass
<fct>	<dbl>	<chr>	<lgl>
Lucy	67	student 1	TRUE
John	56	student 2	FALSE
Mark	87	student 3	TRUE
Candy	91	student 4	TRUE

In [26]:

```
new.students <- data.frame(names = c("Name", "Nom"),
                           score = c(79, 48),
                           student.no = c("student 5", "student
6"),
                           pass = c(TRUE, FALSE))
new.students
```

A data.frame: 2 × 4

names	score	student.no	pass
<fct>	<dbl>	<fct>	<lgl>
Name	79	student 5	TRUE
Nom	48	student 6	FALSE

In [27]:

```
df.new <- rbind(df, new.students); df.new
```

A data.frame: 6 × 4

names <fct>	score <dbl>	student.no <chr>	pass <lgl>
Lucy	67	student 1	TRUE
John	56	student 2	FALSE
Mark	87	student 3	TRUE
Candy	91	student 4	TRUE
Name	79	student 5	TRUE
Nom	48	student 6	FALSE

5.2.2 Add variables to a dataset

This is adding columns.

In [28]:

```
# Option 1
df.copy$id1 <- 1:4
df.copy
```

A data.frame: 4 × 4

score <dbl>	student.no <chr>	pass <lgl>	id1 <int>
67	student 1	TRUE	1
56	student 2	FALSE	2
87	student 3	TRUE	3
91	student 4	TRUE	4

In [29]:

```
# Option 2
df.copy <- data.frame(df.copy, id2 = 1:4)
df.copy
```

A data.frame: 4 × 5

score	student.no	pass	id1	id2
<dbl>	<chr>	<lgl>	<int>	<int>
67	student 1	TRUE	1	1
56	student 2	FALSE	2	2
87	student 3	TRUE	3	3
91	student 4	TRUE	4	4

In [30]:

```
# Option 3
id3 <- 1:4
cbind(df.copy, id3)
```

A data.frame: 4 × 6

score	student.no	pass	id1	id2	id3
<dbl>	<chr>	<lgl>	<int>	<int>	<int>
67	student 1	TRUE	1	1	1
56	student 2	FALSE	2	2	2
87	student 3	TRUE	3	3	3
91	student 4	TRUE	4	4	4

Easily extend to adding multiple columns.

5.2.3 Merge data frames

Can be very useful when we link databases. For example,

1. Database 1 is the electronic health record.
2. Database 2 is the claims data for prescription drugs.

We can merge two databases using the unique patient ID.

In [31]:

```
# df stores student's EPIB 613 score
df
```

A data.frame: 4 × 4

names	score	student.no	pass
<fct>	<dbl>	<chr>	<lgl>
Lucy	67	student 1	TRUE
John	56	student 2	FALSE
Mark	87	student 3	TRUE
Candy	91	student 4	TRUE

In [32]:

```
# df.major stores student's program of study
df.major <- data.frame(student.no = c("student 1", "student 2",
"student 3", "student 4", "student 5", "student 6"),
                       major = c("MSc PH", "PhD Epi", "MSc Epi",
"MSc PH", "PhD Biostat", "MSc Biostat"))
df.major
```

A data.frame: 6 × 2

student.no	major
<fct>	<fct>
student 1	MSc PH
student 2	PhD Epi
student 3	MSc Epi
student 4	MSc PH
student 5	PhD Biostat
student 6	MSc Biostat

In [33]:

```
# See what does the argument 'all' do.
df.full <- merge(df, df.major, by = "student.no", all = F)
df.full
```

A data.frame: 4 × 5

student.no	names	score	pass	major
<chr>	<fct>	<dbl>	<lgl>	<fct>
student 1	Lucy	67	TRUE	MSc PH
student 2	John	56	FALSE	PhD Epi
student 3	Mark	87	TRUE	MSc Epi
student 4	Candy	91	TRUE	MSc PH

5.3 aggregate( )

- Very very very useful function!
- It does conditional operations.
  - that requires subsetting when you don't know aggregate( )

I need a big and complex dataset.

In [34]:

```
# Some simple simulation
# People who take the drug, that are obese and that are younger
are more likely to be cured.
# Setting seeds make random number generation reproducible.
set.seed(613)
n <- 100
drug <- sample(c(0, 1), size = n, replace = TRUE, prob = c(0.8,
0.2))
obesity <- sample(c(0, 1), size = n, replace = TRUE, prob = c(0.
5, 0.5))
age <- round(rnorm(n, mean = 60, sd = 10))
logit.p <- log(1.8)*drug + log(0.85)*(age - 60) + log(1.2)*obesi
ty + log(0.2)
p <- exp(logit.p)/(1 + exp(logit.p))
cured <- rbinom(n, size = 1, prob = p)
sim <- data.frame(drug, obesity, age, cured)
head(sim, 10)
```

A data.frame: 10 × 4

drug <dbl>	obesity <dbl>	age <dbl>	cured <int>
1	1	53	1
1	1	44	1
0	1	61	1
1	0	41	1
0	0	49	0
1	1	54	1
0	1	51	0
1	1	53	0
0	1	70	0
0	1	54	1

In [35]:

```
# Tabulate exposure and outcome
table(sim[, c("drug", "cured")])
# ~20% among unexposed to the drug are cured
# 40% among exposed are cured
```

```
    cured
drug  0  1
   0 47 23
   1 18 12
```

Quick exercise: calculate the mean age of the exposed group and the unexposed group

In [ ]:

```

```

aggregate( ) allows us to aggregate subgroups of the data frame by conditions and then apply a function to all the subgroups.

```
aggregate(x, by, FUN, ...)
```

In [36]:

```
# Syntax 1
aggregate(x = sim$age, by = list(drug = sim$drug), FUN = mean)
```

A data.frame: 2 × 2

drug <dbl>	x <dbl>
0	59.58571
1	61.03333

In [37]:

```
# Alternative syntax
# I highly recommend this one
aggregate(age~drug, data = sim, FUN = mean)
```

A data.frame: 2 × 2

drug <dbl>	age <dbl>
0	59.58571
1	61.03333

In [38]:

```
# Mean age by exposure-obesity group, so 2 binary conditions and
4 subgroups
aggregate(x = sim$age, by = list(drug = sim$drug, obesity = sim$
obesity), FUN = mean)
```

A data.frame: 4 × 3

drug	obesity	x
<dbl>	<dbl>	<dbl>
0	0	59.17500
1	0	61.53333
0	1	60.13333
1	1	60.53333

In [39]:

```
aggregate(age~drug+obesity, data = sim, FUN = mean)
```

A data.frame: 4 × 3

drug	obesity	age
<dbl>	<dbl>	<dbl>
0	0	59.17500
1	0	61.53333
0	1	60.13333
1	1	60.53333

In [40]:

```
# aggregate() can also take multiple target variables
aggregate(x = cbind(sim$age, sim$cured), by = list(drug = sim$dr
ug, obesity = sim$obesity), FUN = mean)
```

A data.frame: 4 × 4

drug	obesity	V1	V2
<dbl>	<dbl>	<dbl>	<dbl>
0	0	59.17500	0.3500000
1	0	61.53333	0.3333333
0	1	60.13333	0.3000000
1	1	60.53333	0.4666667

In [41]:

```
aggregate(cbind(age, cured)~drug+obesity, data = sim, FUN = mean
)
```

A data.frame: 4 × 4

drug	obesity	age	cured
<dbl>	<dbl>	<dbl>	<dbl>
0	0	59.17500	0.3500000
1	0	61.53333	0.3333333
0	1	60.13333	0.3000000
1	1	60.53333	0.4666667

With aggregate( ), we are already doing analysis.

Exercise: Get the count in each drug-obesity-cured group using aggregate( )

In [42]:

```
table(sim[, c("drug", "obesity", "cured")])
```

```
, , cured = 0

    obesity
drug  0  1
   0 26 21
   1 10  8

, , cured = 1

    obesity
drug  0  1
   0 14  9
   1  5  7
```

In [ ]:

```

```