R Lecture #2

October 16th, 2017

Hyeokkoo Eric Kwon (KAIST)

hkkwon7@business.kaist.ac.kr

Data Import and Export

Import from CSV

read.cvs(file, header = FALSE, sep=",")

- **file** : The name of the file which the data are to be read from. Each row of the table appears as one line of the file. If it does not contain an *absolute* path, the file name is *relative* to the current working directory
- **header** : a logical value indicating whether the file contains the names of the variables as its first line.
- **sep** : the field separator character. Values on each line of the file are separated by this character. For example:
  1. ,
  2. ;
  3. \t (tab)

Import from CVS (Cont'd)

- For example, open "a.csv"

  id, name, score

  1, "Kwon", 95

  2, "Lee", 90

  3, "Park", 70

1. *a <- read.csv("a.csv", header = TRUE, sep=",")*
2. *str(a)*
3. *View(a)*

Import from CSV(Cont'd)

- For example, open "b.csv"

  1, "Kwon", 95

  2, "Lee", 90

  3, "Park", 70

1. *b <- read.csv("b.csv", header = FALSE, sep=",")*
2. *View(b)*
3. *colnames(b) <- c("id", "name", "score")*
4. *View(b)*

CSV with Missing Values

- Open "c.csv"

  id, name, score

  1, "Kwon", 95

  2, "Lee", No Score

  3, "Park", 70

- ***na.strings*** : a character vector of strings which are to be interpreted as NA values.

1. *c <- read.csv("c.csv", header = TRUE, sep=",")*
2. *str(c)*
3. *c <- read.csv("c.csv", header = TRUE, sep=",",*

   na.strings=c("No Score"))

   str(c)

Export to CSV

write.csv(x, file = "", row.names=TRUE, sep=",")

- **x**: the object to be written, preferably a matrix or data frame
- **file** : file name
- **row.names** : either a logical value indicating whether the row names of x are to be written along with x, or a character vector of row names to be written

1. *write.csv(c, file = "d.csv", row.names=FALSE)*
2. *write.csv(c, file = "e.csv", row.names=TRUE)*

TXT Files

- Use *read.table* & *write.table* instead of *read.csv & write.csv*
- For example, open "a.txt"

  id, name, score

  1, "Kwon", 95

  2, "Lee", 90

  3, "Park", 70

1. *a1 <- read.table("a.txt", header = TRUE, sep=",")*
2. *str(a1)*
3. *write.table(a1, file = "d1.txt", row.names=FALSE)*

Import from Excel

- One of the best ways to read an Excel file is to export it to a comma delimited file (i.e., CSV) and import it using the method above.
- We can use the function *read_excel* from the *readxl* package. It reads from an Excel spreadsheet and returns a data frame.
  1. *install.packages("readxl")*
  2. *library(readxl)*

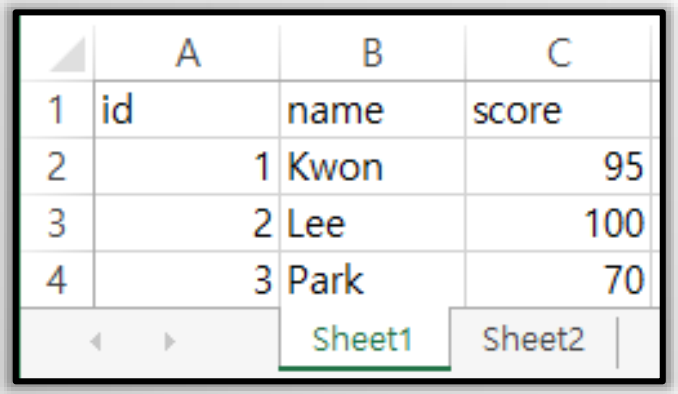
Import from Excel (Cont'd)

read_excel(path, sheet = 1)

- **path** : path to the Microsoft Excel file
- **sheet** : Sheet to read. Either a string (the name of a sheet), or an integer (the position of the sheet).

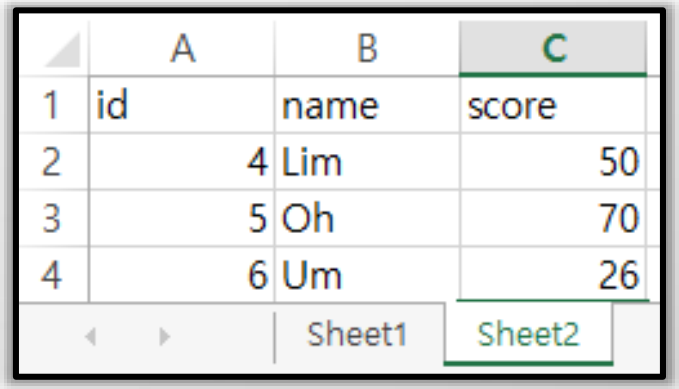
Import from Excel (Cont'd)

- Open "a.xlsx"



	A	B	C
1	id	name	score
2	1	Kwon	95
3	2	Lee	100
4	3	Park	70

Sheet1 Sheet2



	A	B	C
1	id	name	score
2	4	Lim	50
3	5	Oh	70
4	6	Um	26

Sheet1 Sheet2

1. *xls1 <- read_excel("a.xlsx", sheet = 1)*
2. *xls2 <- read_excel("a.xlsx", sheet = 2)*

Detection of Errors

Missing Values

- A missing value, represented by *NA* in *R*, is a placeholder for a datum of which the type is known but its value isn't. Therefore, it is impossible to perform statistical analysis on data where one or more values in the data are missing.

- A common choice, namely "leave out records with missing data" is supported by many base functions through the *na.rm* option.
- Functions such as *sum*, *prod*, *quantile*, *sd* and so on all have this option.

1. *age <- c(23, 16, NA)*
2. *mean(age)*
3. *mean(age, na.rm = TRUE)*

Missing Values (Cont'd)

- Besides the *is.na* function, *R* comes with a few other functions facilitating *NA* handling. The *complete.cases* function detects rows in a data.frame that do not contain any missing value.

1. *person <- read.csv("person.csv", sep=",", header=TRUE)*
2. *complete.cases(person)*

Missing Values (Cont'd)

- The resulting logical can be used to remove incomplete records from the data.frame. Alternatively the na.omit function, does the same.

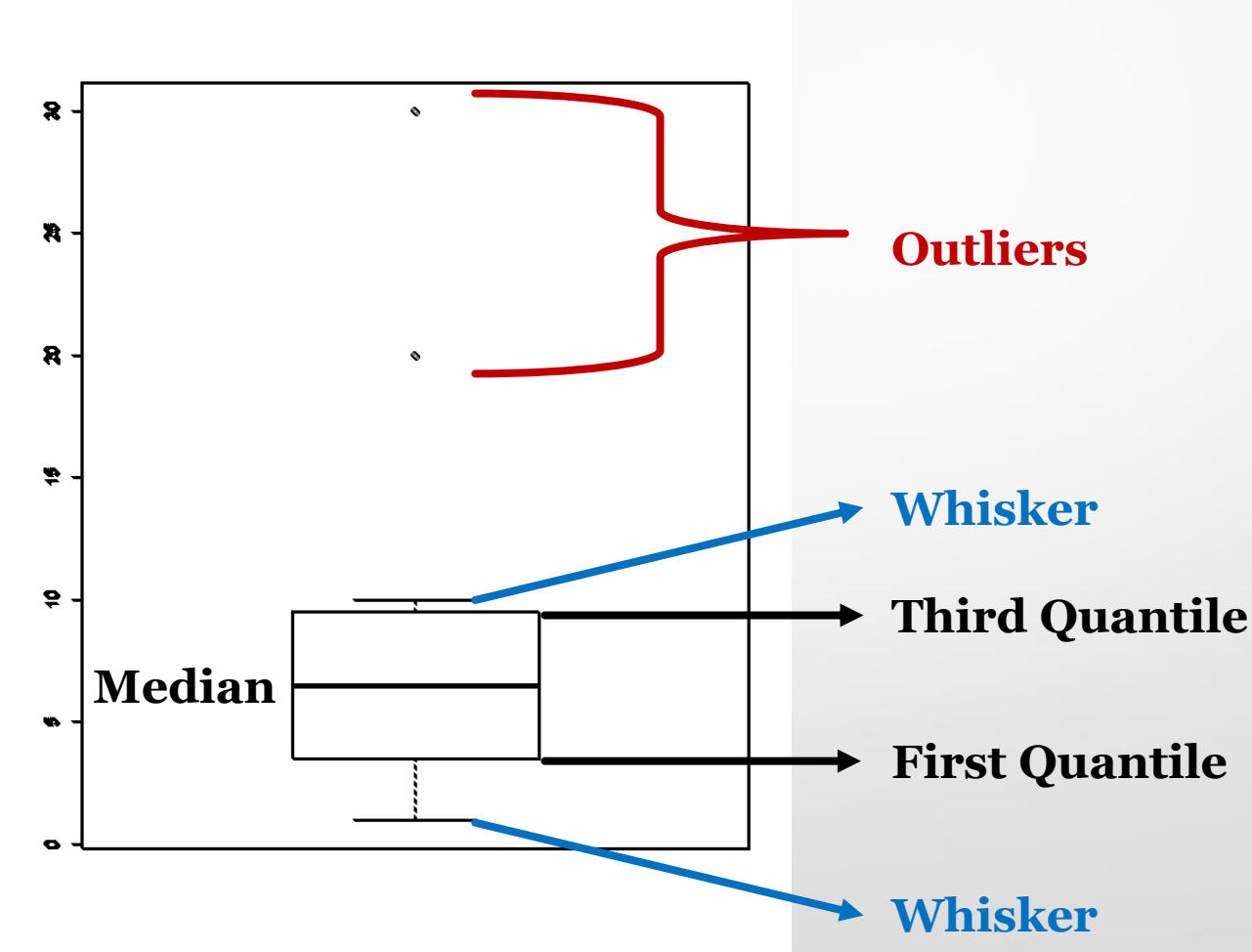
1. *person[complete.cases(person),]*
2. *na.omit(person)*

Outliers

- There is a vast body of literature on outlier detection, and several definitions of outlier exist. For example, Tukey's box-and-whisker method for outlier detection is often appropriate.
- In this method, an observation is an outlier when it is larger than the so-called "whiskers"' of the set of observations. The upper whisker is computed by adding 1.5 times the interquartile range to the third quartile and rounding to the nearest lower observation.

1. *x <- c(1:10, 20, 30)*
2. *boxplot(x)*

Outliers (Cont'd)



Outliers (Cont'd)

- The base R installation comes with function *boxplot.stats*, which, amongst other things, list the outliers.
  1. *ex <- read.csv("outlier.csv", head=TRUE, sep = ",")*
  2. *boxplot.stats(ex$score)$out*
  3. *ex[ex$score>40,]*

- The factor *1.5* used to compute the whisker is to an extent arbitrary and it can be altered by setting the *coef* option of *boxplot.stats*.
- A higher coefficient means a higher outlier detection limit
  1. *boxplot.stats(ex$score, coef=10)$out*

Obvious Inconsistency

- An obvious inconsistency occurs when a record contains a value or combination of values that cannot correspond to a real-world situation.
- For example, a person's age cannot be negative, a man cannot be pregnant and an under-aged person cannot possess a drivers license.
- The *editrules* package allows one to define rules on categorical, numerical or mixed-type data sets which each record must obey.

Obvious Inconsistency (Cont'd)

- We read this data into a variable called *people* and define some restrictions on age using *editset*.

1. *people <- read.csv("inconsistency.csv", header=TRUE, sep = ",")*

	age	agegroup	height	status	yearsmarried
1	21	adult	6.0	single	1
2	2	child	3.0	married	0
3	18	adult	5.7	married	20
4	**221**	elderly	5.0	widowed	2
5	34	child	7.0	married	3

Obvious Inconsistency (Cont'd)

- The *editset* function parses the textual rules and stores them in an editset object.
  1. *install.packages("editrules")*
  2. *library(editrules)*
  3. *age_rule <- editset(c("age >=0", "age <=100"))*

- The data can be checked against these rules with the *violatedEdits* function. violatedEdits returns a *logical array* indicating for each row of the data, which rules are violated.
  1. *violatedEdits(age_rule, people)*
  2. *summary(violatedEdits(age_rule, people))*

Obvious Inconsistency (Cont'd)

1. *drop1 <- violatedEdits(age_rule, people)[,1] |*
   violatedEdits(age_rule, people)[,2]
2. *people[!drop1,]*

1. *drop2 <- apply(violatedEdits(age_rule, people), 1, any)*
2. *people[!drop2,]*

Simple Transformation

- In practice, data cleaning procedures involve a lot of ad-hoc transformations. Consider as an example the following (fictitious) dataset listing the body length of some brothers.

1. *trans1<- read.csv("transform.csv", header=TRUE, sep=" ")*

	name	height	unit
1	Gaucho	170.00	cm
2	Zeppo	1.74	m
3	Chico	70.00	inch
4	Gummo	168.00	cm

Simple Transformation (Cont'd)

- The task here is to standardize the lengths and express all of them in meters and the *deducorrect* package helps to perform it. One first specifies a file with correction rules as follows:

```
1.  # convert centimeters
2.  if ( unit == "cm" ){
3.  height <- height/100
4.  }
5.  # convert inches
6.  if (unit == "inch" ){
7.  height <- height/39.37
8.  }
9.  # set all units to meter
10. unit <- "m"
```

transform rule.txt

Simple Transformation (Cont'd)

1. *install.packages("deducorrect")*
2. *library(deducorrect)*
3. *height_rule <- correctionRules("transform rule.txt")*

- *correctionRules* has parsed the rules and stored them in a *correctionRules* object. We may now apply them to the data.

1. *trans2 <- correctWithRules(height_rule, trans1)*
2. *trans2$corrected*

Data Transformation

Split and Subset

split(x, f) : divides the data in the vector *x* into the groups defined by *f*

- **x**: vector or data frame containing values to be divided into groups.
- **f** : a factor or a list of such factors used for the grouping.

1. *people <- read.csv("inconsistency.csv", header=TRUE, sep = ",")*
2. *split(people, people$agegroup)*

Split and Subset (Cont'd)

subset(x, condition) : Return subsets of vectors, matrices or data frames which meet conditions.

1. *people <- read.csv("inconsistency.csv", header=TRUE, sep = ",")*
2. *subset(people, height >5)*
3. *subset(people, status == "married" & yearsmarried >2)*

Merge

merge(x, y, by) : Merge two data frames by common columns

- **x, y** : data frames, or objects to be coerced to one.
- **by** : specifications of the columns used for merging

1. *x <- read.csv("x.csv", header=TRUE, sep=" ")*
2. *y<- read.csv("y.csv", header=TRUE, sep=" ")*
3. *merged <- merge(x, y, by = c("id"))*

Sort and Order

- sort(x, decreasing= FALSE) : returns an ordered vector
- order(x, decreasing= FALSE) : returns order of each element in a vector

1. *z<-c(20, 11, 33, 50, 47)*
2. *sort(z)*
3. *sort(z, decreasing = TRUE)*

1. *order(z)*
2. *order(z, decreasing = TRUE)*

1. *x <- read.csv("x.csv", header=TRUE, sep=" ")*
2. *x[order(x$score),]*

Group-Level Transformation

summaryBy(formula, data, FUN) : Function to calculate groupwise summary statistics

- ***formula*** *: columns to be calculated and indicating group*
- ***data*** *: a data frame*
- ***FUN*** *: a list of functions to be applied*

1. *install.packages("doBy")*
2. *library(doBy)*
3. *summaryBy(score + age ~ gender, merged, FUN = mean)*
4. *summaryBy(score ~ id, merged, FUN = max)*
5. *summaryBy(score ~ semester, merged, FUN = c(max, min, mean))*