Data Cleaning

# 1. Statistical analysis

We have encountered how data cleaning plays a role in the KDD process. Now, let us examine what is the place for data cleaning in statistical analysis

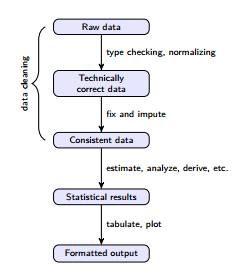


Figure 1: Statistical analysis value chain

The statistical analysis value chain is an example of a data processing pipeline. As we have seen during lecture, raw data is frequently untidy and may also suffers from the following problems:

* Lack of headers
* Wrong data types
* Wrong category labels
* Unknown encoding

Therefore, it is of utmost importance that Raw Data be cleansed and pre-processed into Technically Correct data.

Of course, subject matter discrepancies may still exist even after one has reached the stage of the data being Technically Correct – for example an Age variable having a negative value, or a mean temperature for a tropical country recording negative readings.

# 2. Reading text data into a R data.frame

## 2.1 read.table

We can use the following methods to read data that is technically correct :

read.delim

read.delim2

read.csv

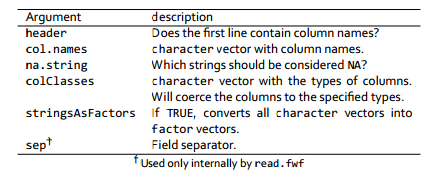
read.csv2

read.table

read.fwf

Using commands above, one will get an R object in the form of a data.frame. Use functions like head, str, and summary to get a feel for the created data frame.

Arguments for all the functions mentioned in Table 1 are:



A. Try the following script, assuming that the folder labelled files containing all the necessary text files are loaded in your current working directory

person <- read.csv('files/unnamed.txt')

person

#STOP! What happened to the headers, and how do we fix it?

person <- read.csv(file ='files/unnamed.txt', header = FALSE, col.names = c("age","height"))

person

str(person)

B. We can also try to interpret the columns in the way we want, as in the following two examples.

read.csv('files/unnamed.txt', header=FALSE, colClasses=c('numeric','numeric'))

dat <- read.csv( file = 'files/unnamed.txt' , header = FALSE , col.names = c('age','height') , stringsAsFactors=FALSE)

# What does the line above do?

str(dat) #answers the question above

dat$height <- as.numeric(dat$height)

dat

## 2.2 readLines

When the rows in a data file are not uniformly formatted we can read the text line-by-line and transform it into a rectangular shape by using the readLines function.

STEP 1 : Read data with readLines

txt<-readLines('files/daltons.txt')

txt

#What do you see here?

STEP 2: Selecting lines containing data

I <- grepl('^%', txt) # and throw them out

#What does grepl do?

(dat <- txt[!I]) #if you enclose an object which is being

# assigned it is equivalent to displaying the object immediately

#after assignment

STEP 3 : Split into separate fields

(fieldList <- strsplit(dat, split = ','))

STEP 4 : Standardize rows. Explain what the following code chunk does.

assignFields <- function(x){

out <- character(3)

# get names

i <- grepl('[[:alpha:]]',x)

out[1] <- x[i]

# get birth date (if any)

i <- which(as.numeric(x) < 1890)

out[2] <- ifelse(length(i)>0, x[i], NA)

# get death date (if any)

i <- which(as.numeric(x) > 1890)

out[3] <- ifelse(length(i)>0, x[i], NA)

out

}

(suppressWarnings(standardFields <- lapply(fieldList, assignFields)))

#What does this code above do?

STEP 5: Transform to data.frame

(M <- matrix( unlist(standardFields) , nrow=length(standardFields) , byrow=TRUE))

colnames(M) <- c('name','birth','death')

(daltons <- as.data.frame(M, stringsAsFactors=FALSE))

STEP 6: Normalize and coerce to correct types

daltons$birth <- as.numeric(daltons$birth)

daltons$death <- as.numeric(daltons$death)

daltons

## 2.3 Type conversion

Converting a variable from one type to another is called coercion.

The basic coercion functions includes:

as.numeric

as.logical

as.integer

as.factor

as.character

as.ordered

Try the following lines of code

as.numeric(c('7', '7\*', '7.0', '7,0'))

as.numeric(c('7', '7\*', '7.0', '7,0'))

class(c('abc', 'def'))

## [1] "character"

class(1:10)

## [1] "integer"

class(c(pi, exp(1)))

## [1] "numeric"

class(factor(c('abc', 'def')))

## [1] "factor

sapply(dat, class) # this retrieves the classes of all columns in a data.frame called dat

The values of categorical variables are stored in factor variables. A factor is an integer vector with a table specifying what integer value corresponds to what level. The values in this translation table can be accessed using the levels function.

For instance, try the following code. Note that "abc" and 'abc' is interpreted the same way in R. Try the following code to see the effects of level and relevel

f <- factor(c("a", "b", "a", "a", "c"))

levels(f)

gender <- c(2, 1, 1, 2, 0, 1, 1)

# recoding table, stored in a simple vector

recode <- c(male = 1, female = 2)

(gender <- factor(gender, levels = recode, labels = names(recode)))

(gender <- relevel(gender, ref = "female"))

age <- c(27, 52, 65, 34, 89, 45, 68)

(gender <- reorder(gender, age))

attr(gender, "scores") <- NULL

gender

For the following section, you need to install the package called ‘lubridate’.

Base R has 3 objects to store time : data, POSIXlt and POSIXct. A PSIXct object stores the number of seconds that have passed since January 1, 1970.

Lets try the following code.

current\_time <- Sys.time()

class(current\_time)

current\_time

library(lubridate)

dates <- c("15/02/2013", "15 Feb 13", "It happened on 15 02 '13")

dmy(dates)

dmy("01 01 68")

dmy("01 01 69")

dmy("15 Febr. 2013")

dates <- c("15-9-2009", "16-07-2008", "17 12-2007", "29-02-2011")

as.POSIXct(dates, format = "%d-%m-%Y")

mybirth <- dmy("28 Sep 1976")

format(mybirth, format = "I was born on %B %d, %Y")

## 2.4 String normalization

## gender

## 1 M

## 2 male

## 3 Female

## 4 fem.

Without preprocessing, the data shown above will be saved as 4 classes. Text strings needs to classified to a number of fixed categories or in other words be ‘encoded’.

To do this, we can use the stringr package to perform string manipulation tasks.

Try the following code

library(stringr)

str\_trim(" hello world ")

str\_trim(" hello world ", side = "left")

str\_trim(" hello world ", side = "right")

toupper("Hello world")

tolower("Hello World")

Another aspect of string handling is that of string matching. One of it involves determining whether a range of substrings occurs within another string – which is known as a pattern. In the second way, we may want to see how different strings are. These two options are powered by the grep and grepl function.

Try the following code to search for string patterns

gender <- c("M", "male ", "Female", "fem.")

grepl("m", gender)

grep("m", gender)

grepl("m", gender, ignore.case = TRUE)

grepl("m", tolower(gender))

grepl("^", gender, fixed = TRUE) #What does this do?

Try the next block of code for string distances

adist("abc", "bac") # This results in 2 as turning abc-bac requires two character substitution

codes <- c("male", "female")

D <- adist(gender, codes)

colnames(D) <- codes

rownames(D) <- gender

D

i <- apply(D, 1, which.min)

data.frame(rawtext = gender, coded = codes[i])

library(stringdist)

stringdist("abc", "bac")

## Exercises

1. Predict the result of the following R statements. Only after your prediction, run it in R

a. exp(-Inf)

b. NA == NA

c. NA == NULL

d. NULL == NULL

e. NA & FALSE

2. In which step in Figure 1 would you perform the following?

a. Estimating values for empty ﬁelds.

b. Setting the font for the title of a histogram.

c. Rewrite a column of categorical variables so that they are all written in capitals.

d. Use the knitr package to produce a statistical report.

e. Exporting data from Excel to csv.

3.

a. Load the warpbreaks dataset. Find out which columns of warpbreaks are either numeric or integer in a single command

b. Is numeric a natural data type for the columns which are stored as such? If not change it to a character class.

c. Figure out what is the underlying type of an object for mean. Try the following command mean[1] and then try and use typeof() to discover the answer.

4. Type the following commands into your R terminal

v <- factor(c("2", "3", "5", "7", "11"))

a. Convert v to character with as.character

b. Convert v to numeric with as.numeric

c. Convert v to integers

5. We will try and work with an irregular text file

a. Read example.txt from the files folder. What is the content of example.txt?

b. Separate the vector of lines into a vector containing comments and a vector containing data. Hint : Use grep.