Data manipulation in R, Ch2

23 January 2019

Table of Contents

Reminder: Always use a R-script!

- *file* $->$ *New script* (Windows), or *file* $->$ *New document* (Mac)
- Write all your commands in the R-script
- Select a part of the scrip (or select the whole script) and run it by
    - **ctrl-r** or **cmd-enter** to run the selected part of the script
- Save your R-scrip under menu *file* or by **ctrl-s** or **cmd-s**

Data frames (data.frame)

Data frames

- Data frames or data.frame() is an R object deigned for handling different type of data
- Many functions in R handles data.frames

```
#Some data put into a data.frame
x<-c(2,3,4,4)
Names <- c("John", "Dial", "Mary","Oskar")
Male <- c(TRUE, TRUE,FALSE,TRUE) indicator<-
rep("Female",4) indicator[Male]<-"Male"
data<-data.frame(x,Names,sex=indicator)
```

Save data (export)

Save the data.frame 'data' as a plain-text file with write.table()

```
?write.table
# Save a space separated (delimited) version of the data ‘data’
write.table(data,
file = "/Users/danwik/desktop/data.txt", sep = " ", na = "NA",
dec = ".", row.names = FALSE, col.names = TRUE)

# Save a comma separated (delimited) version of the data ‘data’
write.table(data,
file = "/Users/danwik/desktop/data.csv", sep = ",", na = "NA",
dec = ".", row.names = FALSE, col.names = TRUE)
```

Exercise

Save the data.frame ‘data’ on your computer as a .csv-file and open it with excel

Import data

Import plain-text (ASCII) data-files with read.table()

Link: Prestige.txt

Link: Prestige.csv

```
# Open the space separated (delimited) text-file Prestige.txt
?read.table()
Prestige<-read.table(file=
"/Users/danwik/desktop/Prestige.txt",
header=TRUE,sep= " ",dec=".",na.string="NA")
rm(Prestige)
Prestige<-read.table(file=
"/Users/danwik/desktop/Prestige.csv",
header=TRUE,sep= ",",dec=".",na.string="NA")
```

Exercise

Import the ‘data’ on your computer which you saved as .csv-file

Import from Excel

- Importing from excel used to be a mess
  - One saved excel-spreadsheets as space or csv delimited and used read.table() to import the excel data
- However, now there are actually some packages to directly import & export excel-files and they work relatively well
- The package *openxlsx* works best! At least on a mac. However, it only works with xlsx-files.
- The package *XLConnect*, works with older excel-files as well. However, often causes memory issues if the file is large.
- Install both

Example: Import from Excel (XLConnect)

Link: Prestige.xls

```
library(XLConnect)
#Path to file
path<-"/Users/danwik/desktop/Prestige.xls"
#Read sheet
Prestige<-
readWorksheetFromFile(path,sheet=1,header=TRUE)
head(Prestige)
summary(Prestige)
```

Example: Save to Excel (XLConnect)

```
#Subset of Prestige data
DATA<-Prestige[,1:2]
# Load workbook (create if not existing)
bah <- loadWorkbook("/Users/danwik/Desktop/Subset.xlsx",
create = TRUE)
createSheet(bah, name = "subset")
writeWorksheet(bah,DATA, sheet = "subset",
startRow = 1, startCol = 1)
saveWorkbook(bah)
```

Example: Import from Excel (openxlsx)

```
library(openxlsx)
Prestige<-read.xlsx(xlsxFile=
"/Users/danwik/Desktop/Kurser/R2018/Data/Prestige.xlsx",
sheet=1, colNames=TRUE)
```

Example: Save to Excel (openxlsx)

```
path<-"/Users/danwik/Desktop/Kurser/R2018/Data/Prestige_ex
wb <- createWorkbook()
addWorksheet(wb, "Prestige")
#setColWidths(wb,sheet="Prestige",cols=1:6,widths = 20)
writeData(wb, "Prestige", lan_TDS)
saveWorkbook(wb, file = path, overwrite = TRUE)
```

Save and open R data (.RDat)

```
# Saving the frame DATA
save(DATA, file = "/Users/danwik/Desktop/DATA.RData")
#remove DATA
rm(DATA)
#Check DATA is removed
ls()
#load saved DATA
load(file="/Users/danwik/Desktop/DATA.RData")
#Check DATA is loaded
ls()
```

Debugging Data files

- Use summary() to control that the variables makes sense
    - Control means, variance, number of NA:s etc.
- Check certain values in R compared to the original data file.

Debugging Data files

```
#Debugging: Control that the imported Prestige data set
#looks like it should
summary(Prestige)
# Check first and last values
Prestige[1,1]
Prestige[nrow(Prestige),ncol(Prestige)]
```

Load data from R packages

- Use *library()* to load the package and use *data()* to load the data from the package

```
#Load data 'Prestige' from the R package 'car'
#install.packages("car")
library(car)
help(package="car")
#data(Prestige)
Prestige
```

Work with data.frame in R

attach() and detach() data frames

- If you use the function *attach*() on a data frame you can directly access the variables in this data frame
- When you don't want to use the attached data frame you should detach it using the *detach*() function

```
# Try to print the ‘education’ variable in ‘Prestige’
education
#Attach the ‘Prestige’ data frame
attach(Prestige)
#Now try to print the ‘education’ variable in ‘Prestige’
education
#Detach Prestige
detach(Prestige)
```

attach() and detach() data frames

- If you use the function *attach*() on a data frame you can directly access the variables in this data frame
- When you don't want to use the attached data frame you should detach it using the *detach*() function

```
# Try to print the ‘education’ variable in ‘Prestige’
education
#Attach the ‘Prestige’ data frame
attach(Prestige)
#Now try to print the ‘education’ variable in ‘Prestige’
education
#Detach Prestige
detach(Prestige)
```

Problems with using attach() and alternative to using it

- Two problems
    1. Masked variables when using multiple data frames: If you have attached one data frame and attach a second data frame variables with equal variable names will be 'masked' from the first data frame.

       E.g. if 'income' is a variable in two data frames you have attached and you want to use the data you first attached, then you cannot call this variable just by writing 'income'. You will obtain the variable 'income' from the second data frame.
    2. If you make changes to a data frame which is attached, the changes are not made in the copy created by attach(). You have to attach the data frame again.
- Instead of attaching a data frame you can always reference a variable by name.frame$name.variable

Example: Problems with using attach() and alternative to using it

```
attach(Prestige)
#Change variable ‘income’ in attached copy of Prestige
income<-income/1000
#Compute the average of income in the attached copy
#of Prestige
mean(income)
#Compute the average of income in the
#data frame Prestige
mean(Prestige$income)
detach(Prestige)
```

factor

- Character data is usually stored as ‘factors’ in data frames
- You can create factors with the function *factor*()
- The function *levels*(*factor*) gives the categories of the factor
- Many functions in R know how to deal with factors

```
#Create a vector with character data
y<-c(rep("yes",2),rep("no",3),rep("maybe",4),"Don’t know")
y
# Define it as a factor
y<-factor(y)
y
#Check the categories
levels(y)
#Check the number of categories
length(levels(y))
#Check the numbers in each category
summary(y)
```

Matrices, Lists and missing values

Matrices

- The function matrix() is used to create matrices in R
- You can add a vector or a matrix to a matrix or joint two or more vectors into a matrix by cbind() (column-wise) and rbind() (row-wise)
- dim(), ncol(), nrow() are also useful functions (see below)

```
A<-matrix(c(1,2,3,4),nrow=3,ncol=4)
A
dim(A); ncol(A); nrow(A); length(A)
B<-matrix(c(1,2,3,4),3,4,byrow=TRUE)
B
AB<-cbind(A,B)
AB
AB<-rbind(A,B)
AB
```

Lists

- In Lists you can store heterogenous vectors, matrices and data types

```
#List
my_list<-list(A=A,B=B,C=matrix(c("a","b","c"),3,3),
vec=c(0,2,3,4))
#Print the whole list
my_list
#Only print the matrix A
my_list$A
```

Missing data

- Missing values is denoted as *NA* (not available)
- You have to think about this when you are exporting and importing data
- *is.na*() is used to identify missing values in R objects
- The default way to treat missing data in R is to remove missing values

```
#Na:s in data
y<-c(1,3,4,5,NA,5,NA)
#When you use the mean() function
#you have to specify the na.rm option
mean(y,na.rm=TRUE)
#With the summary function you don't have to do this.
summary(y)
#You can identify which elements are NA with is.na()
is.na(y)
```

Handle data by Indexing

Indexing

- This is maybe the most essential skill when working with R
- Indexing is used to extract or identify certain elements in different kinds of R objects: Matrices, lists, vectors, data frames and more.

Indexing: Vectors

```
names <- c("John", "Dial", 'Mary',"Oskar")
Male <- c(TRUE, TRUE,FALSE,TRUE)
#Select Dial
names[2]
#Or by a logical argument arg<-names=="Dial"
names[arg]
#"Dial" is actually "Diala". Change this.
names[arg]<-"Diala"
names
#Select the men
names[Male]
#Select the only woman (‘!’ is negation)
names[!Male]
```

Indexing: Factors

- Selecting certain elements works like for a vector

```
f_names<-factor(names)
f_names
#Select the first element
f_names[1]
# Select the boys
Male <- c(TRUE, TRUE,FALSE,TRUE)
f_names[Male]
```

Indexing: Matrices

- You can index both row- and column-wise

```
M<-matrix(c(1,2,3,4),4,3)
M
#Select the first row of matrix M
M[1,]
#Select the second column
M[,2]
#Remove row 2 and 3 (‘-’ is not)
M[-c(2,3),]
#Select all value less than 3 from column 1
arg<-M[,1]<3
M[arg,1]
```

Indexing: Data frames

- You can Index data frames as you do with matrices
- However, there is also a subset() function to make subsets.

```
# Data frames
#Check Prestige is a data.frame with class()
class(Prestige)
#Check the variable names of Prestige
names(Prestige)
#Compute the average of income using indexing
arg<-names(Prestige)=="income"
mean(Prestige[,arg])
#Select a the subsample consisting of "prof" type of jobs
sub_Prestige<-subset(Prestige,type=="prof")
summary(sub_Prestige)
#Select without subset
summary(Prestige[Prestige$type=="prof",])
```

Logical operators

- '$==$'': equal to; '$! =$': not equal to
- '$<$': less than; '$>$': greater than
- '$<=$': less or equal; '$>=$': greater or equal
- '$-$': not
- '&': and; '$|$': or
- '!' negation of a logical object

Example: ‘&’: and; ‘|’: or

```
vec1<-c(2,NA,1,3,4,6,NA)
vec2<-c(NA,2,3,4,5,4,NA)
Mat<-cbind(vec1,vec2)
#And
arg<-is.na(vec1) & is.na(vec2)
Mat[!arg,]
#or
arg<-is.na(vec1) | is.na(vec2)
Mat[!arg,]
```

Exersice: Subset of the Data frame Prestige

Use the Prestige data

1. Select a subsample consisting of only “prof” for variable “typ”
2. Select the subsample of individuals with “income” less than 10000
3. Finally select the subsample of occupations of type “prof” with income less than 10000

Indexing: Lists

- Indexing lists are a bit different
- You use double square brackets to obtain a listed object

```
#my_list<-list(A=A,B=B,C=matrix(c("a","b","c"),3,3),
#vec=c(0,2,3,4))
# Lists. Select the first object (matrix A) in my_list
my_list[[1]]
#Select the first row of matrix A in the my_list
my_list[[1]][1,]
```