Second R meeting

Marco Smolla

November 6, 2013

How to handle data.frames and lists

Finishing the example from the last meeting

```
ID <- 15:6
Height <- round(runif(n = 10, min = 1.5, max = 2.2), digits = 2)
Sex <- sample(x = c("female", "male"), size = 10, replace = TRUE)
```

Remember logical operators
$a == b$ equal to
$a! = b$ unequal to
$a < b$ less than
$a > a$ bigger than
$a >= a$ bigger or equal to

Let us have a closer look at our data.

```
Sex == "male"
maleHeight <- Height[Sex == "male"]
femaleHeight <- Height[Sex != "male"]
mean(maleHeight)
mean(femaleHeight)
```

How does the mean change when you have 100 instead of 10 individuals?

The data frame - keeps your data organised

We will now create a data.frame that stores the three vectors ID, Height, Sex as columns would do in an ordinary table.

```
data <- data.frame(id = ID, height = Height, sex = Sex)
data
class(data)  # check the class of the data object
str(data)  # detailed information about all compartments of the object
summary(data)  # with summary() R tries to give you a quick statisitc of your values
```

Navigating through your data.frame

```
head(data)  # first five lines
tail(data)  # last five lines
names(data)  # returns the column names of the data.frame
dim(data)  # the dimensions of the data frame; try ncol() and nrow()
data$height  # the $ let's you call a specific column
data[1, ]  # shows the first line and all columns of data; also try data[5:7, ]
data[, 1]  # shows the first column and all lines of data; also try data[2:5, 2:3]
data[, -1]  # exclude the first column
data[, "sex"]  # try summary(data[ , 'sex'])
data[, c("id", "height")]  # include several columns defined by their name
data[Height >= 1.9, ]  # Why should you rather use this: data[data$height>=1.9, ]
data[data$height >= 1.9, "sex"]
```

Sometimes you understand your data better when you visualize it. Let us try to make three simple plots: a line plot, a histogram and a bar plot.

```
plot(data$height, type = "b")
# try also to sort the values with: plot(sort(data$height), type='b')
# Let us create a histogram that shows the distribution of body size
hist(data$height, xlab = "Height", main = "Distribution of body size")
# And a simple barplot that shows the differences between male and female
barplot(height = c(mean(Height[Sex != "male"]), mean(Height[Sex == "male"])),
    names.arg = c("female", "male"))
```

Importing and exporting data

We will import data from an Excel spreadsheet using the xlsx package.

```
library(xlsx)
```

Try to look up the help file of the package to learn how it works.

Now we need to locate the file we want to import. In my case I saved the file on the Desktop, but it could also be at any other place. When you start R it sets a place where it searches for files, the so called working directory. Let's see which directory this is.

```
getwd()
# returns the place where R is currently looking for a file
setwd("~/Desktop")
# tells R to look in a different place for files, in Windows you need a \
# instead of a /
```

We use the read function from the xlsx package to import data into R (what is the differnce between read.xlsx and read.xlsx2?)

```
read.xlsx2(file = "PlayerExpenses.xlsx", sheetIndex = 1)
```

Instead of setting the working directory, you could also import the file using file=' /Desktop/-PlayerExpenses.xlsx'. If you have a closer look at the data we just imported you recognize, that the first column looks strange. Have a look at the orginal data and you will see that the first column are dates. We can tell read.xlsx () to handle the first column as a date which in R is defined by the class 'POSIXct'. We do this by manipulating the argument colClasses.

```
expenses <- read.xlsx2(file = "PlayerExpenses.xlsx", sheetIndex = 1, colClasses = c("POSIXct",
    "character", "character", "numeric"))
```

Have a look at the data and use the functions we already know: str() and summary().

```
expenses
str(expenses)
summary(expenses)
```

Let's make a quick graph of the development of fees payed by the football club for a new player over the past century.

```
plot(expenses[, 1], expenses[, 4])
# Try the following two lines to get a nicer looking plot
plot(expenses[, 1], expenses[, 4]/1e+06, xaxt = "n", xlab = "Date", ylab = "Fee in Million GBP",
    pch = 20, type = "b")
axis.POSIXct(1, x = expenses$Date)
```

I sent you a file that includes the nationality of each player. Let's import this csv (comma separated values) file and add the information to our expenses data.frame.

```
nationality <- read.csv(file = "Nationality.csv")
```

I included the Fee column so you can **check whether the order of the two data.frames is correct**. We do this with the all() function. It checks whether all elements are TRUE and if so returns TRUE.

```
all(expenses$Fee == nationality$Fee)
```

To **add a vector to a data.frame** we just tell R how the new column shoud be named, here it is Nation.

```
expenses$Nation <- nationality[, 1]  # equal to: <- nationality$Nationality
expenses
summary(expenses)
```

By the way, if you would like to combine two data.frames, you can use the data.frame funciton

```r
data.frame(expenses, nationality)
```

Note, we have two columns with the same name (Fee). R will add an identifier to make all column names unique. If you need replicated column names use the argument check.names=F

Say, we are only interested in a **subset of the whole data set**, e.g. only the english player.

```r
expensesEnglish <- expenses[expenses$Nation == "England", ]
```

Inspect the new data.frame using the Workspace of RStudio and clicking at expensesEnglish.

For our future work we may only need the fees payed and the team names. We will **store this in a new csv file**. We use the write.csv() function to create a new csv file.

```r
write.csv(file = "expensesE.csv", expensesEnglish[, c("Sold.to", "Fee")])
write.xlsx(file = "expensesE.xlsx", x = expensesEnglish[, c("Sold.to", "Fee")],
    sheetName = "England")
```

Store data as an R object

Sometimes it is easier to **save your manipulated data the same way as it is currently stored in R**, as an R object. Try this:

```r
save(expenses, file = "myData")
```

As a result you will find a file in your current wokring directory (how do you know what the current working directory is?) with the name 'myData.'

Let's see how we **get this R object back into R** again.

```r
rm(expenses)  # let's get rid of the original object
expenses  # R tells us that the object does not exist
load("myData")  # we now load the R object myData, which includes the expenses data.frame
expenses  # and there it is again
```

Note: you can store several objects in a single file. Try for example `save(expenses, data, file='myData2')`.

Using a list to handle several data.frames

If you want to keep track of several values in different data.frames, you can store them in a list.

```r
fruit <- data.frame(apple = c(3, 2, 4, 5), banana = c(7, 7, 8, 1), strawberry = c(0,
    1, 1, 10))
vegetable <- data.frame(salad = c(15, 24, 17), tomato = c(10, 9, 2))
list(fruit, vegetable)
food <- list(fruits = fruit, vegetables = vegetable)
food
```

```r
class(food)
str(food)
summary(food)
```

Navigating through a list

```r
food$fruit
food[[1]]
class(food[[1]])
food[[1]]$n
food[[1]][1]
```