Basics in R

Marco Smolla

29.10.2013

1 Getting Started

1.1 Downloading R

- CRAN distributes the basic R for Linux, MAC OSX, and Windows
  http://cran.r-project.org
- RStudio is a graphical user interface for R that will make your life easier. It is freeware and it works across all three major operating systems
  http://www.rstudio.com

1.2 R Documentation and further links

- An Introduction to R, from CRAN
  http://cran.r-project.org/doc/manuals/R-intro.pdf
- The ultimate R Reference Card
  http://cran.r-project.org/doc/contrib/Short-refcard.pdf
- Quci-R gives short introductions to many different R topics
  http://www.statmethods.net
- The R help files (you also reach them when typing ?function) in R
  http://stat.ethz.ch/R-manual/R-devel/doc/html/
- An R lecture including exercises
  www.stat.columbia.edu/

1.3 Install R and RStudio

Both R and RStudio are executable files. Just double click the downloaded files and follow the instructions. After you installed R, install RStudio.

1.4 Installing additional packages

Packages (you can also call them libraries) are collections of functions that someone else wrote and thought it would be useful for others to access these functions. As we will see, there are basic functions that are part of every R distribution (e.g. the sum function `sum()`). Other functions are only included in external packages. CRAN (the Comprehensive R Archive Network) is the

largest repository for R packages. To install for example the plotting environment *ggplot2*, click in the 'Packages' window on 'Install Packages' then type ggplot2 and hit return (see Figure 1). R will now install the required package.
Note, the package is installed now, but if you want to use the package in the current session you need to tell R that you require it now. To that simply type the following command in the console and R loads all functions of the package so that you can use them:
`require(ggplot2)`

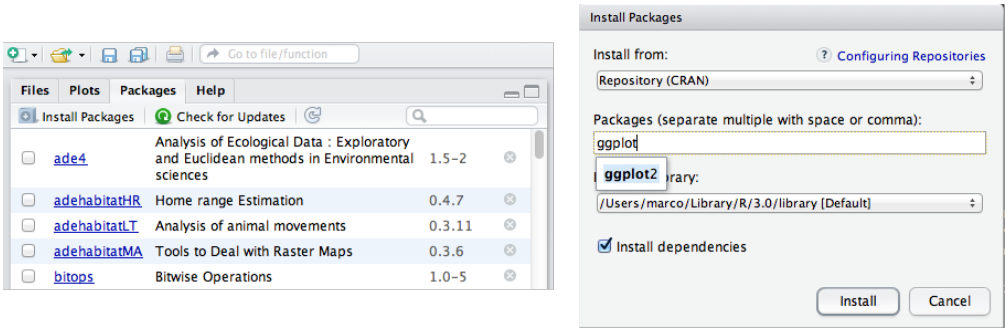

Figure 1: Installing packages

2 Help

There is always help at your hand. Using RStudio helps you remembering variable or function names. For example, type in the console `as.da` and than press your tab key (usually left from the 'Q' key). RStudio will give you recommendations of functions that start with 'as.da' and indeed, there is the function I was looking for `as.data.frame`. Next to the name RStudio give also a short overview of what the function does and which values it requires. Press your Return key to select the function.
It happens many times to me, that I forget how the function worked. If that is the case, simply put a '?' in front of the function name and hit Return (i.e. `?as.data.frame`). Now RStudio opens the complete help file of the function.
In other cases I don't know whether there is a function or not. In that case you use '??' combined with something you are looking for, e.g. `??bind` and RStudio runs a search through all help files of R to find help pages that matches your search phrase.

3 Using R

```
### 1.  Simple calculations and first objects Using basic operators
9 + 2 - 1
5 * 0.5
27/3
9^2
9^(1/2)
sqrt(9)
log(5)
```

```r
### Comparing values Will return a logical value, which is either 'TRUE'
### or 'FALSE'
54 > 45  # greater than
1 < 2  # less than
15 == 15  # exactly equal to
50 >= 51  # greater than or equal to

### Assigning values to a variable There are different ways to assign
### values to a variable (or placeholder) The most common is the arrow
### method <- that assigns a value to the variable it is pointing at.
x <- 2  # assigns a numeric value to a variable
y = 15  # -- same effect
assign("d", 7)  # -- same effect
color <- "green"  # assigns a character value to a variable

c(2, 3, 4)  # c is a function that combines values to a vector
c(d, color, "Flowers")  #

# test what happens when you ype this
c(1:10)

### Calculating with variables Once we created
b <- x + y
f <- d^x
g <- x < y


### Simple statistical measure
vector <- c(x, y, d, f, b, 50.3, 7^2)
length(vector)
sum(vector)
mean(vector)
max(vector)
min(vector)
median(vector)
summary(vector)

# try also the sort() and the order() function with the vector


### 2.  Understanding objects in R There are different so called classes
### of objects. We have already seen numeric characters, and logical
### values. The function class() helps you to find out which data type you
### are dealing with. Let's test this:
class(2)
class(c("a", "2"))
class(TRUE)
class(2 < 3)
class(vector)

# To an extend we can change the class of an object, for example, we can
# make a numeric value to a character value (characters are always
```

```r
# surrounded by ' '):
as.character(3)
# We can also make a logical value to a numeric
as.numeric(TRUE)
as.numeric(FALSE)
as.character(TRUE)
# But remember, by changing the class, you also change the behvaior of R
# regarding this object. But R tries hard to understand what you are doing
# see this:
"TRUE" == TRUE
"2" == 2
# BUT:
"color" == color


ls()  # returns all values that are in your workspace
rm(x, y, d, f, b, g)  # removes individual values
ls()

### 3.  Ways to organize and navigate through your data

### Vector
ID <- c(1:10)
Hight <- round(runif(n = 10, min = 1.5, max = 2.2), digits = 2)
Sex <- sample(x = c("female", "male"), size = 10, replace = TRUE)

Sex == "male"
maleHight <- Hight[Sex == "male"]
femaleHight <- Hight[Sex != "male"]
mean(maleHight)
mean(femaleHight)
# How does the mean change when you have 100 instead of 10 individuals?

### The data frame - keeps your data organised
data <- data.frame(id = ID, hight = Hight, sex = Sex)
summary(data)
class(data)

head(data)  # first five lines
tail(data)  # last five lines

names(data)  # returns the column names of the data.frame

data$hight  # the $ let's you call a specific column

data[1, ]  # show the first line and all columns of data
data[, "sex"]  #try summary(data[ , 'sex'])
data[Hight >= 1.9, ]  # Why should you rather use this: data[data$hight>=1.9, ]

# Let us create a histogram that shows the distribution of body size
hist(data$hight, xlab = "Hight", main = "Distribution of body size")
```