Start playing with R

Diana Santos

To prepare to Belgrade training school

1 First: go in and out

Start R (clicking on the icon, or writing commandoen “R” on your command line, or entering RStudio).

R shows it is ready with the prompt >. Then you can write commands in R.

The most import is how to leave it (the R interpreter):

```
> q()
```

When you issue this command, R asks whether you want to save your workspace. This can be useful when you are in the middle of a bif project, but for now you can asnwer no.

The second most important command is to ask for help:

```
> help()
```

If you want more specific help about a particular command, for example square root, use `help(sqrt)`.

It is also useful to know where you are (in our computer). In other words, where R expects data. For this there is the command (short for “get working directory”):

```
> getwd()
```

In different operating systems you get different kinds of place definitions, for example:

```
[1] "e:/work/dssantos"
[1] "/hf/sokrates/ilos-u1/dssantos"
```

If you want to change the place where you work to another place, just tell R where, using the command (short for “set working diurectory”):

```
> setwd("m:Rwork/")
```

or

```
> setwd("/linguateca/cursos/R/")
```

To get some acquaintance with R, you can start ysing the R environment as a calculator, and then learn some more complex datastructures:

a) arithmetic expressions and strings

```
> 3+4
[1] 7
> sqrt(49)
[1] 7
> value <- 8
> value
[1] 8
> (7+6)*3 - value
[1] 31
> "hi"
[1] "hi"
> noe
Error: object 'noe' not found
> 'sorry'
[1] "sorry"
> mas()
Error: could not find function "mas"
> help
function (topic, package = NULL, lib.loc = NULL, verbose = getOption("verbose"),
    try.all.packages = getOption("help.try.all.packages"), help_type = getOption
{
    types <- c("text", "html", "pdf")
    if (!missing(package))
 [...]
>
```

b) vectors

Take notice that the function `c` concatenates more than one object into an array:

```
> c("23","1","potato","carrots")
[1] "23"       "1"        "potato"  "carrots"
```

```
> idiotic <- c("23","1","potato","carrots")
> idiotic
[1] "23"      "1"       "potato"  "carrots"
> 24:100
 [1]  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42
[20]  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61
[39]  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80
[58]  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99
[77] 100
```

I hope that the way R answered explained the basic facts about R. Let me ellaborate:

- Each answer is numbered, numbering begins by 1, and when more then one answer is given, R writes only the first number in that line, leftmost mellom []
- When one issues a command in R, R executes it and shows the answer
- When one just writes a name, R tries to see whether it already has a meaning (as a container, called variable in a programming language). If yes, R provides us with the value stored in that container (which can even be the code for a function); if not, R complains with an error message
- When one writes a simple value, R repeats/returns the value
- There is a (quite important) difference between the name of a variable, and a string (which one uses quotes to write)
- Functions in R are spelled `NAME(arguments)`. When they have no arguments, as the help function, it is necessary to write () after the name, i.e. `help()`

2 Communicate with the environment

If we want to use R to do something real, and not just as a calculator or an oracle, we have to use data which already exists, and we will have to produce tables and figures. This means that one has to be able to work with files, reading and writing from them.

The easiest way to begin is to read from a file on the Web, so you will start reading files from URLs. Later you will learn to use files on your own machines.

CONDIVport CONDIVport is a corpus created by Augusto Soares da Silva's team in order to study the convergence and divergence of the Portuguese language across its two main varieties: Brazilian and Portuguese. The corpus contains texts in three domains or themes, taken from newspapers and magazines devoted to football, fashion and health, from three decades. See [13]. The data here belongs to a study of colour we did back in 2010. The "colour" column indicates the number of words with a colour meaning in the texts.[11]

```
> CondivSimples <-
 read.table("http://dinis2.linguateca.pt/Diana/UnivOslo/cursoR/condiv.txt")
> CondivSimples
              V1     V2       V3     V4      V5
1        dataset colour    theme decade variety
2    fashion50BR   1512  fashion     50      BR
3    fashion50PT   2351  fashion     50      PT
[...]
```

When one sees what has been read by R, one discovers that the first line is special: It is a header. There is an easy way to tell R this, but first I want to explain what R does when our command is longer than one line: R waits until we are ready, and shows it is waiting by exchanging the prompt to + instead of >.

```
> CondivSimples <- read.table("http://dinis2.linguateca.pt/Diana/UnivOslo/cursoR
+ header=TRUE)
> CondivSimples
         dataset colour    theme decade variety
1    fashion50BR   1512  fashion     50      BR
2    fashion50PT   2351  fashion     50      PT
[...]
```

The file contents show the most common way of writing values for statistical analysis: an observation per line, which includes values for different features, one per column. This is called a data frame.

We can describe a data frame as a matrix whose columns and lines can have names. The only difference between a mathematical matrix and a data frame is that a matrix must have values of the same type (generally numbers), while a data frame can have columns with different types (in the same column, though, you are allowed only one type). (By type I mean things like integers, real numbers, strings, factors, etc.)

To get the dimensions of a data frame (how many lines, how many columns) one uses the function `dim`:

```
> dim(CondivSimples)
[1] 18  5
```

To get only one column, one uses the data frame's name followed by a dollar sign and the name of the column:

```
CondivSimples$colour
 [1] 1512 2351 1342 6481 1525  669  719  694  468  318 1167  614  136  583  127
[16]  993   97  265
```

In this particular data frame, you can see that all values of the first column are different. This can apparently be used to identify unambiguously each line. This is not that rare, and so you can tell R that one column is used for row names by using the command `rownames=column number`. But note that most data frames, with thousands of lines, do not have a column for that.

```
> CondivSimples <- read.table("http://dinis2.linguateca.pt/Diana/UnivOslo/cursoR
+ header=TRUE, row.names=1)
> CondivSimples
               colour    theme decade variety
fashion50BR      1512  fashion     50      BR
fashion50PT      2351  fashion     50      PT
[...]
```

Notice that after executing the last command one gets a data frame with only four columns, not five as previously.

```
> dim(CondivSimples)
[1] 18  4
```

We could also create a new data frame with just a subset of the information from the original data frame, with the `subset` function. In the following example, we choose only the lines that deal with football, and the columns that concern colour and language variety.

```
> OnlyFoot<-subset(CondivSimples,theme=="football",c(1,4))
> OnlyFoot
               colour variety
```

```
football50BR      719      BR
football50PT      694      PT
football70BR      468      BR
football70PT      318      PT
football2000BR   1167      BR
football2000PT    614      PT
```

Lets us now learn how to write/save this new data frame in your working directory.

```
> write.table(BareFot, file="football.txt")
```

Go and see what is there in the file.

You can also read it back into R:

```
read.table("football.txt")
```

For completeness' sake, let us also create a table from scratch, and then write it to an extern file. This is not usually what one does, but it helps to get used to R. (Instead of a data frame, we create here a true matrix, which is a simple example of a contingency table.)

```
> exampleTable<-matrix(c(38,14,11,51),nrow=2)
> exampleTable
> colnames(exampleTable)<-c("BlueEyes","BrownEyes")
> rownames(exampleTable)<-c("BlondHair","BrownHair")
> write.table(exampleTable, file="HairEyes.txt")
```

3 Figures and graphics

Until now nothing shows R to be different from other programming languages, in what concerns text, matrices or other data structures.

The graphic capabilities are maybe the first area where users can see clear advantages of using R.

First, it is important to understand that R has several windows. If we don't say anything, R opens one window for all graphics. This means that when one asks R to create a picture, we "lose" the previous picture, because R writes in the same window. (But we can ask for another window...)

Follow the simplest commands to create figures, where the tilde ˜ reads as "as a function of".

```
> plot(colour ~ variety, data=CondivSimples)
> hist(CondivSimples$colour)
> plot(CondivSimples$colour)
```

If one is working mainly with one data frame, it is quite practical not to have to write its name all the time. By using the command `attach`, one can just use the names internal to the data frame.

```
> attach(CondivSimples)
```

Repeat the previous figures after attaching the data frame, to see how much simpler it gets.

One other interesting property of R is as follows: R makes invoking a function rather easy, because it allows it to be called wih different number of arguments (exactly like natural language, where a verb can be used without all its logical arguments).

Also, a function can behave differently depending on its arguments. Therefore the function `plot` can do different figures, depending on its argument.

In addition, one can (but does not need to) specify hundreds of different parameters, like figure title, form, caption, colour of the points, scale, etc. etc.

Just a few examples:

```
> plot(colour, col="green", pch=3, type="b")
> plot(colour, col="red", pch=2, las="1", tcl="0.2", xlab="cases",
+ ylab="How many colour words", type="h")
> plot(colour, col="red", pch=2, las="1", xlab="cases",
+ legend(4,6481,"Max")
```

To investigate this thoroughly (or at least more thorougly), try `help(plot)`, or read about the many possibilities in R [12].

Most important for now is to create figures in an appropriate format for other situations (articles, presentations, ...), and learn how to send figures to other programs. Or rather: redirect printing to a particular format. For a pdf:

```
> pdf("simple.pdf")
> plot(colour~theme)
> dev.list()
```

To come back to the original (print) window, it is enough to use the `dev.list` command to se which windows are open, and then choose the one you want to come back to.

```
> dev.list()
X11cairo      pdf
       2        3
> dev.set(2)
```

NB! Before one closes the window one is writing to, it is not possible to open it with other programs (like Adobe viewer, for pdf). Remember, therefore, to close the window with `dev.off()`. You can give this command the number of the window you want to close, or no arguments, in which case it closes the last window open, as in the next example:

```
> dev.off()
X11cairo
       2
```

The same happens for other formats, like `png`, `bmp`.

If you want to (keep) open more than one window at the same time, you can use the command `dev.new()`.

4 Take care of your work in R from session to session

Although this is not particularly relevant to beginners, it is very practical to be able to “freeze” everything you have, close R, and come back another day.

In these cases you can ask R to keep everything, so that R is in the same “state” when you come back. There is a special file format for this, `.Rdata`, and the command to create it, and to use it are as follows:

```
> save.image("everything.Rdata")
```

Go out of R and back again. Notice that R always asks you whether you want to keep what you have done so far. But if you write yes, it creates a file just called `.Rdata`. And you can have more than one if you want. So it is a good idea to save some things in a file `important.Rdata`, and then continue playing and trying, but not save after that.

You can also learn some cleaning commands in R: With `ls` you can see what exists in your R environment, and with `rm` you can remove what you want to remove.

```
> ls()
character(0)
> load("everything.Rdata")
> ls()
[1] "idiotic"       "CondivSimples" "onlyfb"        "OnlyFoot"
> rm(idiotic)
> ls()
```

Another possibility, which uses a lot less place, is an history-file, of type `.Rhistory`. One can give it a full name as well:

```
> savehistory("comandsMAthesis.Rhistory")
```

But it is important to know that it only has a size of 100 commands by default. (You can change it. Try to find how.)

To get theckommands back (in another session), it is enough to use `loadhistory`. (It is nice to have the commands available, because one can use the arrows to fetch the previous commands...)

```
> loadhistory("comandsMAthesis.Rhistory")
```

To finish, it is important to remind you that R expects all these files to be in your working directory. If it is not, one has to write the whole pathname, or use `setwd`.

5 Bibliography

There are dozens of books which give an introduction to R for humanists and others. In addition to the book by Baayen [1, 2], there is Johnson [8], Gries [6], Levshina [9], Brezina [3] for linguistics, and Jockers [7] for literature studies.

Crawley [4] and Dalgaard [5] are for science students, but provide very good overviews.

Since R is open source, there are many people who contribute to R, write documentation for R [10], and create new modules all the time. R's official homepage, `http://www.r-project.org/`, gives also access to a tutorial and many help pages.

References

[1] Harald Baayen. *Analyzing Linguistic Data: A practical introduction to Statistics using R*, Cambridge University Press, 2008.

[2] R. H. Baayen. *languageR: Data sets and functions with "Analyzing Linguistic Data: A practical introduction to statistics*, 2011. `http://CRAN.R-project.org/package=languageR`. R package version 1.2.

[3] Vaclav Brezina. *Statistics in Corpus Linguistics: A Practical Guide*, Cambridge University Press, 2018.

[4] Michael J. Crawley. *Statistics: An Introduction using R*, John Wiley and Sons, 2005.

[5] Peter Dalgaard. *Introductory Statistics with R*, Springer, 2008.

[6] Stefan Th. Gries. *Statistics for Linguistics with R: A Practical Introduction*, Mouton de Gruyter, 2009.

[7] Matthew L. Jockers. *Text analysis with R for Students of Literature*, Springer, 2014.

[8] Keith Johnson. *Quantitative Methods in Linguistics*, Wiley-Blackwell, 2008.

[9] Natalia Levshina. *How to do Linguistics with R: Data exploration and statistical analysis*, John Benjamins, 2015.

[10] R Development Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, 2010. `http://www.R-project.org`. ISBN 3-900051-07-0.

[11] Diana Santos, Rosário Silva e Cláudia Freitas. Pluralidades na cor: contrastando a língua do Brasil e de Portugal. Em Augusto Soares da Silva, Amadeu Torres e Miguel Gonçalves, editores, *Línguas Pluricêntricas: Variação Linguística e Dimensões Sociocognitivas. Pluricentric Languages: Linguistic Variation and Sociocognitive Dimensions*, 2011. p. 555–572.

[12] Tom Short. *R Reference Card*, 2004. `http://cran.r-project.org/doc/contrib/Short-refcard.pdf`.

[13] Augusto Soares da Silva. Measuring and parameterizing lexical convergence and divergence between european and brazilian portuguese. Em Dirk Geeraerts, Gitte Kristiansen e Yves Peirsman, editores, *Advances in Cognitive Sociolinguistics*, 2010.