

A quick introduction to R

Introduction

This is a short introduction to the R language, meant to be read as a primer to the R course arranged at CSC. This introduction covers only 10 commands, and 4 notations worth noting. These have been highlighted with green text when first encountered.

What is R?

R is both a programming language and a statistical environment. Don't be put off by the eerie sound of "programming language". It's just the way R has been constructed to work. You don't need to know much about programming in order to be able to use R effectively.

Where to acquire R?

R has already been installed on the computers in the CSC class room. If you need to install it on your own computer, you can copy the version on the class room computer, and paste it somewhere on your own machine. Assuming you are using Windows, that is. You can also download an installer or source codes from CRAN, the project website at <http://www.r-project.org/> .

What are all those packages?

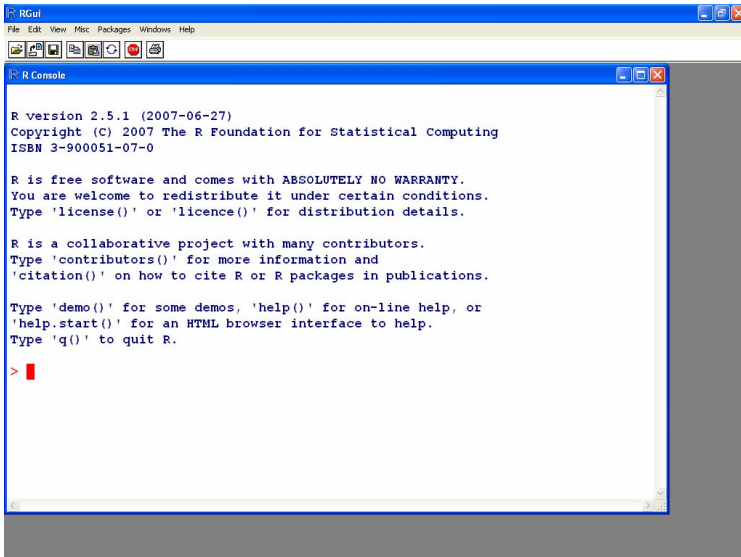
Packages are like computer accessories (joystick, scanner, etc.). They are not always necessary, but improve the experience or enable one to do things not possible without. Most of R functionality is scattered around in numerous packages that user can download and install on his own computer. Some packages come along with the default installation of R, but most of them need to be installed separately. The best place to study new packages is CRAN.

And Bioconductor – what's the hassle?

Bioconductor is a project that builds tools for analysis of genomic data, such as gene expression data. These tools are implemented as R packages. Bioconductor packages are distributed from the project's own website at <http://www.bioconductor.org/> .

User interface

The R user interface is rather utilitarian, consisting of a few menus and a command line with basic editing functionality. All the commands used in the analyses are typed to the command line starting with the sign `>` (see below). This sign is called prompt, since it prompts the user to write something.



```
RGui
File Edit View Misc Packages Windows Help
R Console
R version 2.5.1 (2007-06-27)
Copyright (C) 2007 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

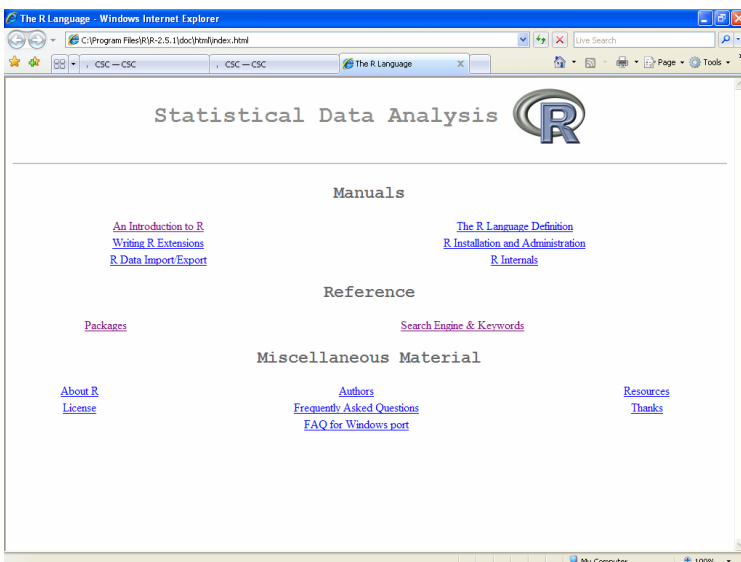
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> █
```

Help

Html-help can be invoked from the Help-menu. From the opening webpage, you can access manuals, frequently asked questions, references to help for individual packages, and most importantly, Search Engine. Help is the best place to find out new functions, and get descriptions on how to use them.



Starting to work with data

The first thing to do is to change to the folder where the data is located. Go to File –menu, and select Change Dir... Browse to the correct folder, and click OK. Now R is able to see your datafile, and it can be loaded into R. When quitting R, contents of the R memory (workspace) are saved in the same folder, if needed.

Basic usage – commands and objects

As already pointed out, all data manipulations and analyses are carried out by typing the relevant commands. Let's first load in a datafile. Here, the data to be loaded is in a table (tab-delimited text file) like this (data from the R Help list):

A	B	C	Resp
1	1	1	34.12
1	1	2	32.45
1	1	3	44.55
1	2	1	20.88
1	2	2	22.32
1	2	3	27.71
2	1	6	38.2
2	1	7	31.62
2	1	8	38.71
2	2	6	18.93
2	2	7	20.57
2	2	8	31.55
3	1	9	40.81
3	1	10	42.23
3	1	11	41.26
3	2	9	28.41
3	2	10	24.07
3	2	11	21.16

Note that every column has a title, those being A, B, C and Resp. To load in this particular file, type:

```
> read.table("Book1.txt", header=T, sep="\t")
```

Now you should see the table appearing on the screen in the R window. What we just did was read in the table from the file Book1.txt. This table had column titles (`header=T`), and was separated by tabs (`sep="\t"`). The R command that did the job was `read.table`, and the filename, column titles, and separator were given as arguments to the command.

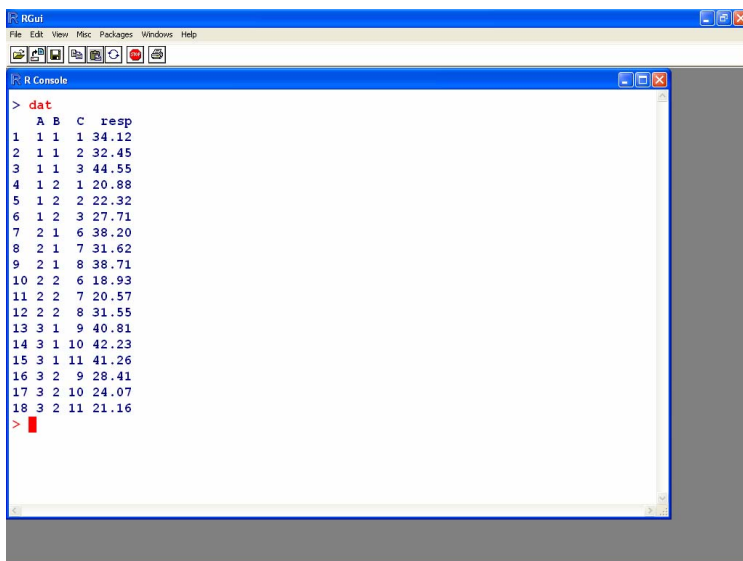
Note that the command was associated with brackets inside which the arguments were given. All R commands are like this: they are always coupled with a pair of brackets. If the command tells R what to do ("Read table!"), arguments tell which table to read, and how it should be read ("Note that the table is in file Book1.txt, it has column titles, and columns are separated by tabs").

The table we read in was immediately lost, because we did not save it inside R. In R all the variables and datasets are stored in memory as objects. To store the table as an object, we should modify the previous command as follows:

```
> dat<-read.table("Book1.txt", header=T, sep="\t")
```

Using this latter command for reading in the table, the dataset would be saved as an object named `dat` inside R. Note the notation "`<-`". It literally means that save everything on right side of the notation to the object that is named on the left side of the notation. When this command is issued nothing is displayed on the screen. This implies that everything went OK. In contrast to Windows, R typically only displays a message if something went wrong. We can see the table inside R by just typing its name to the command line:

```
> dat
```



```
> dat
  A B C resp
1  1 1 1 34.12
2  1 1 2 32.45
3  1 1 3 44.55
4  1 2 1 20.88
5  1 2 2 22.32
6  1 2 3 27.71
7  2 1 6 38.20
8  2 1 7 31.62
9  2 1 8 38.71
10 2 2 6 18.93
11 2 2 7 20.57
12 2 2 8 31.55
13 3 1 9 40.81
14 3 1 10 42.23
15 3 1 11 41.26
16 3 2 9 28.41
17 3 2 10 24.07
18 3 2 11 21.16
```

Accessing the data – extracting individual columns or rows from the table

OK, so we've got the data into R. Next we want to analyze it. In the case of our example, every column in the file is a variable. Column `resp` contains information on the response we have measured, and the columns `A` to `C` contain information on certain treatments. How can we, say, calculate a mean (arithmetic average) of the response variable?

This can be done by accessing the column by using its name. Remember that the table was stored in the object called `dat`. To display the column `resp` from that object, we can use operator `$`:

```
> dat$resp
```

This syntax might look a bit frightening at first, but it really just means that read the column `resp` from the table `dat`. Note that this is not a command: Commands are always associated with brackets, so we are just accessing the table.

Mean of a variable is calculated using the function `mean()`. To calculate the mean of the column `resp` in table `dat`, we put the previous notation inside the brackets of the command `mean()`:

```
> mean(dat$resp)
```

And R gives us the answer:

```
[1] 31.08611
```

Now we know how to access different columns, but what about rows? If I want to get the 15th row of the table, how do I do that? Now, that is accomplished using subscripts. Subscripts are slightly hard to grasp at first, but very handy indeed. To access the 15th row, one would write:

```
> dat[15,]
```

That displays only fifteenth row of the table:

```
   A B  C  resp
15 3  1 11 41.26
```

The notation `[]` after the name of an object is a subscript. Note again that there are no commands involved, as there are no brackets anywhere. No brackets, no command, and hence we are trying to access an object.

In a similar fashion, one can actually use subscripts to access columns also. To get the column for response variable:

```
> dat[,4]
```

Note that subscripts are always written `[rows, columns]`. For example,

```
> dat[15,4]
```

will give the fifteenth observation from fourth column. If no column or row is specified that part of subscript is left empty (but note that the comma is needed).

Doing something more interesting – combining commands and setting up statistical models

Let's see if the treatments A and B have any effect on the response variable. This can be accomplished using a two-way ANOVA. The command that does the analysis is `aov()`. Inside the brackets we must type the formula of the test:

```
> a1<-aov(dat$resp~dat$A+dat$B)
```

This means that run a linear model where the values of response variable are modeled using variables A and B (and only their main effects). That's the formula inside the brackets. The results of this run are stored in an object `a1`. To get the final results, another command is needed:

```
> summary(a1)
```

That gives the results of the analysis:

```
      Df Sum Sq Mean Sq F value    Pr(>F)
dat$B   1  915.21   915.21  46.8213 5.576e-06 ***
dat$A   1   21.09    21.09   1.0792  0.3153
Residuals 15  293.20    19.55
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

As the generated table indicates, the main effect of treatment B is statistically significant, but the main effect of A isn't.

The two distinct steps above could have been combined:

```
> summary(aov(dat$resp~dat$A+dat$B))
```

Note that there are now two commands that are combined, as there are two pairs of brackets. The first pair of brackets goes with the command `summary()`, and the second pair with the command `aov()`. Brackets are always interpreted from the inside to the outside, meaning in this case that `aov()` is executed first, and its results are given to `summary()` that is executed second.

Looking at ANOVA in more details – loading libraries

The analysis run above was done using something called type I sums of squares. If you've never heard of the whole concept, never mind, that's just one method of calculating the statistical significance of the variables in the ANOVA model. We are now going to run the analysis again, but using another type of sums of squares (type III). The function with type III sums of squares is available in the library `car`. This library is installed in R, but not yet initialized (loaded in memory). Before being able to use this library it needs to be loaded into memory:

```
> library(car)
```

To get the type III sums of squares, the `summary()` command is replaced by `Anova()`:

```
> Anova(aov(dat$resp~dat$A+dat$B), type="III")
```

```
Response: dat$resp
      Sum Sq Df  F value    Pr(>F)
(Intercept) 2792.97  1 142.8864 4.558e-09 ***
dat$B       915.21  1  46.8213 5.576e-06 ***
dat$A       21.09  1   1.0792  0.3153
Residuals   293.20 15
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

If you compare the results given by type I and type III sums of squares, they do not differ, which is very nice, since they really shouldn't, because the experimental setup is orthogonal (explanatory variables are selected in such a way that they do not correlate with each other).

Dealing with objects in memory

After working for a while one might get slightly confused as to what objects there are in R memory. Object names can be listed using the command `objects()`. It really only lists all the names of the objects in current R memory:

```
> objects()  
[1] "a1" "a2" "dat"
```

The command mentioned above is not always very helpful, since it might be interesting to know what type of an object the specific object is. There are a few commands that can help in assessing the type of an object. First, typing the name of an object usually prints the contents of an object to the screen:

```
> a1
```

```
Call:
```

```
aov(formula = dat$resp ~ dat$A * dat$B)
```

```
Terms:
```

	dat\$A	dat\$B	dat\$A:dat\$B	Residuals
Sum of Squares	21.0940	915.2068	9.1002	284.1018
Deg. of Freedom	1	1	1	14

```
Residual standard error: 4.504774  
Estimated effects may be unbalanced
```

It is often possible to decipher the contents of an object from this information.

It is also possible to study the class of an object. For example, class of the object `dat` is `data.frame`. In layman terms it means that the object is a table.

```
> class(a1)  
[1] "aov" "lm"
```

```
> class(dat)  
[1] "data.frame"
```

One can also study the innards of an object in much more details using the command `str()`. It prints a description of the object's structure to the screen:

```
> str(dat)  
'data.frame': 18 obs. of 4 variables:  
 $ A : int 1 1 1 1 1 1 2 2 2 2 ...  
 $ B : int 1 1 1 2 2 2 1 1 1 2 ...  
 $ C : int 1 2 3 1 2 3 6 7 8 6 ...  
 $ resp: num 34.1 32.5 44.5 20.9 22.3 ...
```

In this case, `str()` tells us, that `dat` is a data frame with 18 observations (rows) and 4 variables. The next lines list the variables (`A`, `B`, `C`, `resp`), and a few of the first observations of every variable. The dollar sign (\$) in front of every line describing a variable just reminds as about the fact that these variables can be accessed using `dat$A` -type of notation.

One can always get the column names of a table using the command `str()`, but there are two easier commands, `colnames()` and `rownames()` that list the column names and row names of a data frame, respectively.

Writing data to a disk

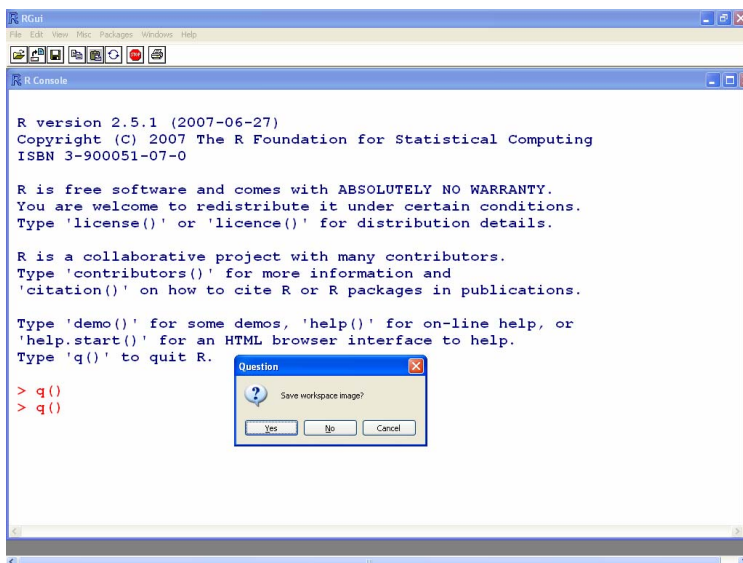
A data frame can be easily written to a file using the command `write.table()`. This command only works for data in a table format (for instance, as a data frame). If needed, we could write the table contained in the object `dat` to a file `dat.txt` as follows:

```
write.table(x=dat, file="dat.txt")
```

The same would not work with the object `a1`, since it is not a data frame. The best option is to save the whole R workspace (everything in R memory) to disk. Actually, this is what R automatically does, when the user quits the programs (see the next section).

Quitting R

R can be closed with the command `q()`. After issuing the quit command, R asks whether to save the workspace or not:



It is usually a good idea to save the workspace, since this creates a special file that can be directly read into R, and one can commence working with the same datasets and results already generated without a need to start from the scratch again.

Saved workspace is in a file called `.RData`, and all the commands given during the same R session are saved in a file called `.Rhistory`. To load the workspace into R again, one can simply double-click on the file `.Rdata`, and R should open automatically with all the data and results loaded. Note however that libraries are not loaded automatically, and these should be loaded (if needed) before commencing the work.