

Starting with *R*

Please be aware that this document is NOT intended to teach *R*, but solely to explain certain actions and particular commands so that students are able to perform the required analyses in their BEMO computer classes.


If you wish to learn *R* you can take a free online course, download manuals and other materials freely available on the internet, so that you can most effectively use *R* in your work.

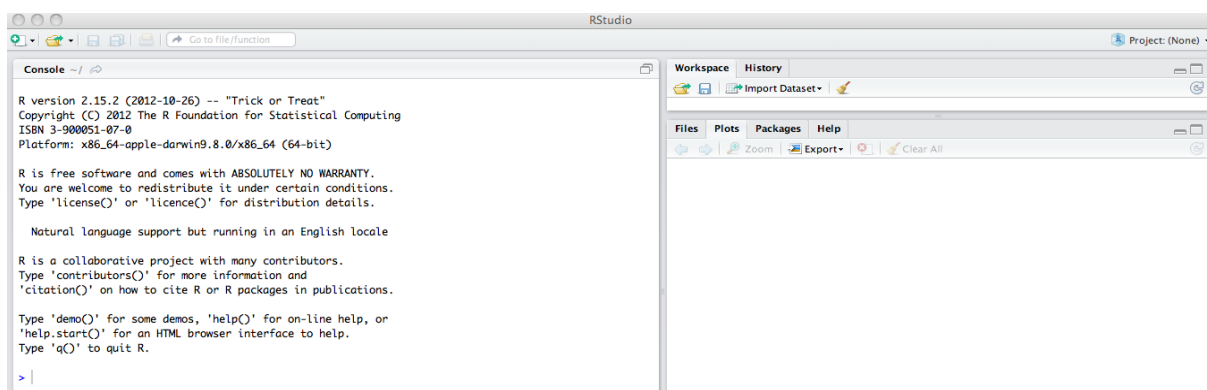
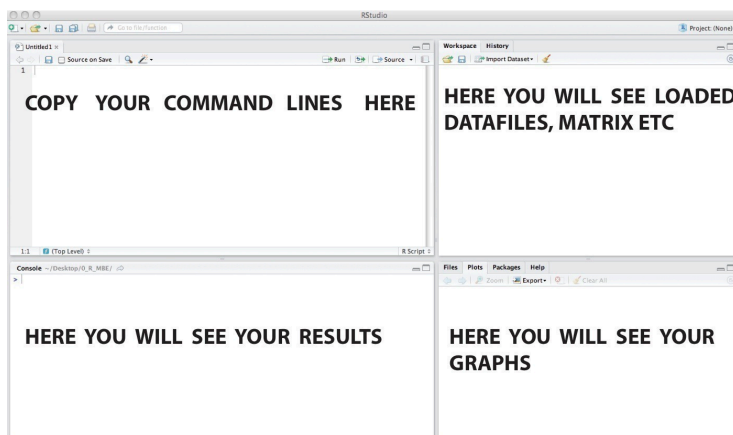
Rita Castilho
Faro, Portugal
2017

Introduction

If you are reading this document, it means that you are not proficient in R, but even if you are, I would suggest that instead of using the “normal” R console, downloadable from [CRAN](http://CRAN.R-project.org/), you use [RStudio](http://www.rstudio.com/), which will make your life much, much easier. I also suggest you read the document attentively. If you skip a step you may end not being able to load your data file, or do your analysis, simply because some really simple settings need to be done in order for R to work properly!

1. HOW TO GET THE SCRIPT ON THE SOURCE WINDOW?

You may write your own scripts directly on the source window, however, most of R unexperienced users will use pre-made scripts. These usually have two extensions: *.txt or *.R. You can open them both by pressing on the  icon, and choose, R Script.



2. HOW DO I SET UP THE WORKING DIRECTORY?

In most cases, the scripts you are using need a datafile. So in order for R to find that file you need to set up the working directory. This can be done in two ways:

I) Go to Session/Set Working Directory and choose the directory where you have the datafile.

ii) alternatively you can write in the console: `setwd("~/Desktop/R")`, if the directory is called R and is on the desktop, or any other appropriate indication.

There is also another command that does not need this previous step, for instance, when you need to choose the datafile you can write (`file.choose()`). However, at this point we want to minimize the sources of possible mistakes.

3. WHAT FORMAT SHOULD I SAVE THE PLOTS?

If you plan, as you should, to edit the figures obtained, in order to customize them, then you must save the plots in Plot/Export/Save Plot as image. Choose EPS format, which will allow you to save the file in a format that can be edit in [Inkscape](#) (open source) or [Adobe Illustrator](#) or the likes. Remember, if your work is being evaluated, extra credit will be given to creative presentation of data.

4. MOVING FORWARD

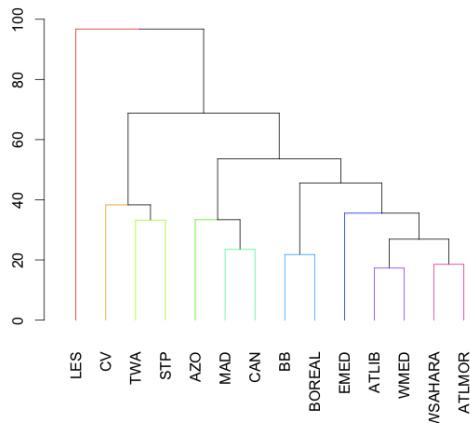
It is time to look at the command lines and make the appropriate changes. The yellow highlight indicates changes you need to make.

When you want to **execute a line**, you should place the mouse over that line and click `cmd+enter` (in Macs) and `ctrl+enter` (in windows). You can find many more shortcuts in https://www.rstudio.com/ide/docs/using/keyboard_shortcuts.

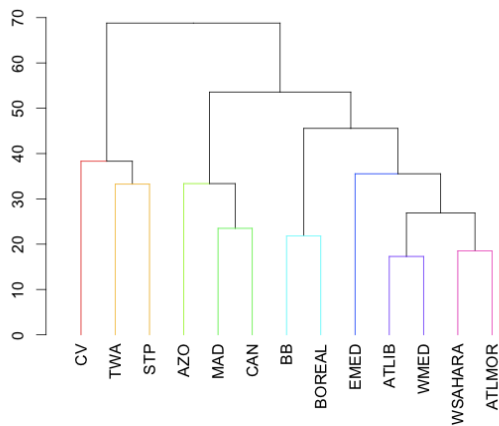
Building a cladogram

PART 1: clustsig

```
##### Bray-Curtis
#without installing the packages, R cannot know where to get function
install.packages("clustsig")
#without downloading library, R cannot access the functions of the package
library(clustsig)
#reading the txt file saved from excel as tab delimited (sep = "\t"), if you want to have a different separator
you need to change the sep parameter
#you need to either change the name of your file to Data.txt, or replace in the command line the name of the
file you have
data.tab = read.table("Data.txt", header=TRUE, sep = "\t")
#this command will give you the matrix dimension and you can check if your data is correctly important
dim(data.tab)
#this command will give you the matrix dimension and you can check if your data is correctly important
#you need to change the number of columns to be read, which equals the number of locations
data.mat = as.matrix(data.tab[,1:12])
#this is the main part of the command lines, read the package manual to find out what each part of the
command means
# for instance if you change method.distance="braycurtis" to
method.distance="euclidean" you will get an euclidean distance
#matrix
BC=simprof(data.mat, num.expected=10, num.simulated=9,
method.cluster="average", method.distance="braycurtis", alpha=0.05,
sample.orientation="column", const=0, silent=FALSE, increment=100)
#having the distance matrix, now we need to obtain the cladogram
simprof.plot(BC, leafcolors=NA, plot=TRUE, fill=TRUE,
leaflab="perpendicular", siglinetype=1)
#we can now obtain the number of groups which are found to be statistically significant
BC$numgroups
#Obtain a list of length numgroups with each element containing the sample IDs (row/column numbers in
the corresponding original data) that are in each
#significant cluster.
BC$significantclusters
# p-values for testing whether the two groups in that row are statistically different. The null hypothesis is
that there is no a priori group structure, so if p<0.05 means that there is structure
BC$pval
```



If you want to leave LES species out, because it is the last column, you can replace 14 by 13, in `data.mat`
`= as.matrix(data.tab[,1:13])`.



You will obtain this result:

```
[ ,1] [ ,2] [ ,3]
[1,] -9 -11 1 (ATLIB-WMED)
[2,] -7 -8 1 (WSAHARA-ATLMOR)
[3,] -10 -13 1 (BB-BOREAL)
[4,] -5 -6 1 (MAD-CAN)
[5,] 1 2 0 (ATLIB-WMED)-(WSAHARA-ATLMOR)
[6,] -1 -2 1 (TWA-STP)
[7,] -4 4 0 AZO-(MAD-CAN)
[8,] -12 5 0 EMED-(ATLIB-WMED)-(WSAHARA-ATLMOR)
[9,] -3 6 0 CV-(TWA-STP)
[10,] 3 8 0 (BB-BOREAL)-EMED-(ATLIB-WMED)-(WSAHARA-ATLMOR)
[11,] 7 10 0 AZO-(MAD-CAN)-(BB-BOREAL)-EMED-(ATLIB-WMED)-(WSAHARA-ATLMOR)
[12,] 9 11 0 CV-(TWA-STP)-AZO-(MAD-CAN)-(BB-BOREAL)-EMED-(ATLIB-WMED)-(WSAHARA-ATLMOR)
[13,] -14 12 0 LES-CV-(TWA-STP)-AZO-(MAD-CAN)-(BB-BOREAL)-EMED-(ATLIB-WMED)-(WSAHARA-ATLMOR)
```

1. TWA
2. STP
3. CV
4. AZO
5. MAD
6. CAN
7. WSAHARA

8. ATLMOR
9. ATLIB
10. BB
11. WMED
12. EMED
13. BOREAL
14. LES

This output shows how sites or clusters were merged in a hierarchical sequence of 13 steps, indicated between the square brackets.

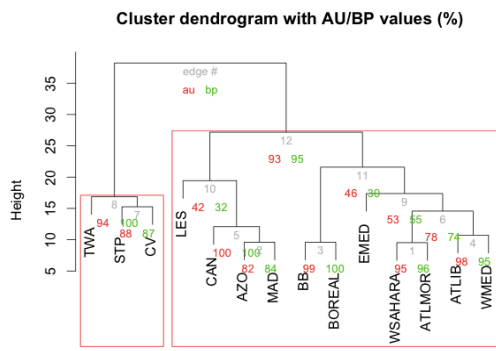
In case a number has a "-" sign, then this indicates that clusters are formed by an original category. In case a number has no "-" sign, then this indicates that clusters are joined that were formed at earlier steps with the actual numbers indicating the step. For instance, in the last step (step 13), clusters formed in step 12 and the fourteenth site are merged. At the first step, the ninth and the eleven site were joined. These are labelled -9 and -11 in the output. When you check the distance matrix, then you can see that these two sites have the smallest value of all pairwise distances. In the fifth step, the first and second clusters was merged. The last column indicates the probability value of the cluster.

PART 2: pvclust

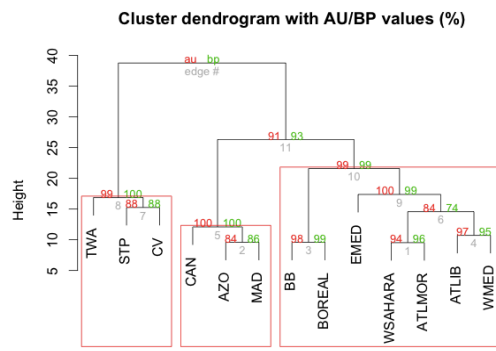
```
##### Ward Hierarchical Clustering with Bootstrapped p values
install.packages("pvclust")
library(pvclust)
#the agglomerative method used in hierarchical clustering. This should be (an abbreviation of) one of
"average", "ward", "single", "complete", "mcquitty",
# "median" or "centroid". The default is "average"
fit <- pvclust(data.mat, method.hclust="ward", method.dist="euclidean")
plot(fit) # dendogram with p values
# add rectangles around groups highly supported by the data
pvrect(fit, alpha=.95, pv="bp")
#####
```

From <http://www.is.titech.ac.jp/~shimo/prog/pvclust/>

pvclust is an [R](#) package for assessing the uncertainty in hierarchical cluster analysis. For each cluster in hierarchical clustering, quantities called p-values are calculated via multiscale bootstrap resampling. The p-value of a cluster is a value between 0 and 1, which indicates how strong the cluster is supported by data. **pvclust** provides two types of p-values: **AU** (Approximately Unbiased) p-value and **BP** (Bootstrap Probability) value. **AU** p-value, which is computed by multiscale bootstrap resampling, is a better approximation to unbiased p-value than **BP** value computed by normal bootstrap resampling. **pvclust** performs hierarchical cluster analysis via function `hclust` and automatically computes p-values for all clusters contained in the clustering of original data. It also provides graphical tools such as `plot` function or useful `pvrect` function which highlights clusters with relatively high/low p-values. Furthermore, parallel computation is available via `snow` package.



Distance: euclidean
Cluster method: ward



Distance: euclidean
Cluster method: ward

Building an MDS

If you started an MDS without having previously read the data you would need to go through those steps.

```
# Classical MDS
#Reading the data (from a txt file, tab delimited, 1st row: labels, no column labels)
data.df = read.table("data.txt", header=TRUE, sep = "\t")
#transform data.frame into matrix so that can be transposed
data.mat = as.matrix(data.df[,1:13])
#X Y
dim(data.mat)
#transposing matrix
tdata.mat<-t(data.mat)
#confirm if Y X
dim(tdata.mat)

#Obtaining distance matrix BC=Bray-Curtis

install.packages("clusterSim")
install.packages("calibrate")
library(clusterSim)
library(calibrate)
data.bcdist <- dist.BC(tdata.mat)
# k is the number of dimensions
fit <- cmdscale(data.bcdist,eig=TRUE, k=2)
# view results
fit
# define X and Y for plotting
x <- fit$points[,1]
y <- fit$points[,2]
# plot solution
plot(x, y, xlab="Coordinate 1", ylab="Coordinate 2", main="Bray-Curtis
MDS", type="p", pch=16)
# plot labels
textxy(x, y, labs = row.names(tdata.mat), cx = 0.5, dcol = "black", m =
c(0, 0))

#Obtaining distance matrix Euclidean distances
deu <- dist(tdata.mat,method="euclidean") # euclidean distances between the
rows
fit <- cmdscale(deu,eig=TRUE, k=2) # k is the number of dim
fit # view results
# plot solution
x <- fit$points[,1]
y <- fit$points[,2]
plot(x, y, xlab="Coordinate 1", ylab="Coordinate 2", main="Euclidean MDS",
type="p", pch=16)
textxy(x, y, labs = row.names(tdata.mat), cx = 0.5, dcol = "black", m =
c(0, 0))
```