Tutorial 5 The Working directory and data in R

Setting the working directory

In the first part of this tutorial we will learn how to set the working directory and explore its contents in R.

First create a folder with the name *r_workshop_data* in your desktop and save the data file for the workshop. This is a dataset of the distribution of Chytridomycosis in Australia. We will not use the complete data set for this workshop, but the all the files and meta data are available here.

To set the working directory you can use the *Misc* dropdown menu and *Change working directory* in Unix-like machines, or with *File* in Windows. An other option is to specify the path with the following command:

```
setwd("path/to/your/file")
```

Note that in Windows the paths are specified with backslash (), but R always uses the forward slash (/).

Set the working directory to the *r_workshop_data* folder.

We should make sure that we specified the working directory correctly. To get the working directory in R use the following command:

```
getwd()
```

```
## [1] "/Users/sebastianducheneAIr/Dropbox/PostersTalks/R_workshop_GSA_2014/intro_to_r/tutorial5"
```

Check the contents of the directory and make sure that it contains the data file.

```
dir()
```

```
##  [1] "Chytridiomycosis_data_1956_2007.txt"
##  [2] "Chytridomycosis_10.csv"
##  [3] "Chytridomycosis_10.xlsx"
##  [4] "Chytridomycosis_short.csv"
##  [5] "data_table_10.csv"
##  [6] "data_table_10.txt"
##  [7] "README.md"
##  [8] "README.md~"
##  [9] "sim_seqs.fasta"
## [10] "sim_tree.tre"
## [11] "tut5_seqs.fasta"
## [12] "tut5_tree.tre"
## [13] "tutorial5.rmd"
## [14] "tutorial5.rmd~"
## [15] "web_data.csv"
## [16] "webtable.csv"
```

Reading and writing text data

From this point in the workshop we will use the data files that we saved in the *r_workshop_data* folder.

Read the data as a text file with the command bellow.

```
data_text <- readLines("Chytridomycosis_short.csv")
```

This command reads the data as a vector, where each line in the file is an item in the vector. We will confirm the class of the object, and then determine its length with the length() function.

```
class(data_text)
```

```
## [1] "character"
```

```
length(data_text)
```

```
## [1] 2500
```

Open the file in a text editor (I recommend TextWrangler or NotePad++, but any other is sufficient in this case).

You can also open the file in Excel or any spreadsheet program, which should read it as a regular table of data.

Individual exercise: subset the vector to print on the screen the first ten items of data_text. You can use indexing as shown in tutorial 2.2. or search the help for the function head(). After inspecting the first six lines, create a vector called *data_text_10* for these first ten items. As an optional exercise, print on the screen the last ten items of the data set. Hint: Remember the length() function that we used earlier

Your code for the exercise should look something like this:

```
data_text_10 <- data_text[1:10]
```

We will now save the first ten lines in a file with the function writeLines(). This function takes several arguments. Use the help file for more information on how to use the function. The basics are explained bellow:

```
writeLines(text = data_text_10, con = "Chytridomycosis_10.csv")
# we could also use writeLines(data_text_10, 'Chytridomycosis_10.csv')
```

Check the *Chytridomycosis_10.csv* file in a text editor or spreadsheet program. The format should be identical to that of the original file.

Reading and writing tables

Reading data as text is only useful for text manipulations. For data analysis it is necessary to read the data like a spreadsheet and work with rows and columns. In R it is most convenient to have the data as data frames or matrices. To do this we can use the read.table() function.

Open the help file for the read.table function. It has many possible arguments with default values. In some cases it is not necessary to change the defaults, but sometimes this can cause problems, so it is useful to become familiar with this function.

Load the data file with the following command:

```r
data_table <- read.table(file = "Chytridomycosis_short.csv", sep = ",", head = T)
```

We specified sep="," because the default is tab delimited text, and we want R to use the commas to separate the values. Note that this is similar to using the function read.csv(), as shown in the help file for read.table().

The argument head = T is used to specify that the first row of the file is the names of the columns, which is useful if we want to treat our data like a data.frame object.

It is very important to make sure that the data are read correctly. A very useful function is str(), which prints the structure of objects in R:

```r
str(data_table)
```

```
## 'data.frame':    2499 obs. of  18 variables:
##  $ Compiled_by     : Factor w/ 1 level "RichardRetallick": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Database_ID     : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Species         : Factor w/ 66 levels "Adelotus brevis",..: 59 59 59 59 59 59 59 59 59 59 ...
##  $ Sex             : Factor w/ 8 levels "Female","Female ?",..: 6 6 6 6 6 6 6 3 3 1 ...
##  $ Site            : Factor w/ 224 levels "1km east of Bogong Swamp, Kosciusko NP",..: 14 14 14 14 14
##  $ State           : Factor w/ 8 levels "ACT","missing data",..: 1 1 1 1 1 1 1 3 3 3 ...
##  $ Country         : Factor w/ 1 level "Australia": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Year            : int  1997 1997 1997 1997 1997 1997 1997 1997 1997 1997 ...
##  $ Diagnostic      : Factor w/ 1 level "Histology": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Individuals     : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Indivs_positive : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Collector_source: Factor w/ 38 levels "A. Beezley / P. Couper",..: 8 8 8 8 8 8 8 8 8 8 ...
##  $ Orig_database   : Factor w/ 5 levels "D. Mendez","D. Mendez / R. Speare",..: 1 1 1 1 1 1 1 1 1 1
##  $ Disease_status  : Factor w/ 3 levels "negative","no result",..: 1 1 1 1 2 2 2 1 1 1 ...
##  $ Accuracy        : Factor w/ 2 levels "acceptable","unacceptable": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Latitude        : num  -35.6 -35.6 -35.6 -35.6 -35.6 ...
##  $ Longitude       : num  149 149 149 149 149 ...
##  $ Dead_or_sick    : Factor w/ 2 levels "not noted","noted": 1 1 1 1 1 1 1 1 1 1 ...
```

As we can see, the data were read as a data.frame object, so we can use the indexing and column names like we did in tutorial 2.

We can also inspect the first six rows of the table:

```r
head(data_table)
```

```
##        Compiled_by Database_ID                Species          Sex
## 1 RichardRetallick           1 Pseudophryne pengilleyi not recorded
## 2 RichardRetallick           2 Pseudophryne pengilleyi not recorded
## 3 RichardRetallick           3 Pseudophryne pengilleyi not recorded
## 4 RichardRetallick           4 Pseudophryne pengilleyi not recorded
## 5 RichardRetallick           5 Pseudophryne pengilleyi not recorded
## 6 RichardRetallick           6 Pseudophryne pengilleyi not recorded
##                Site State   Country Year Diagnostic Individuals
## 1 Black Fellows Gap   ACT Australia 1997  Histology           1
## 2 Black Fellows Gap   ACT Australia 1997  Histology           1
## 3 Black Fellows Gap   ACT Australia 1997  Histology           1
## 4 Black Fellows Gap   ACT Australia 1997  Histology           1
## 5 Black Fellows Gap   ACT Australia 1997  Histology           1
## 6 Black Fellows Gap   ACT Australia 1997  Histology           1
##   Indivs_positive Collector_source Orig_database Disease_status   Accuracy
## 1               0        D. Hunter     D. Mendez       negative acceptable
## 2               0        D. Hunter     D. Mendez       negative acceptable
## 3               0        D. Hunter     D. Mendez       negative acceptable
## 4               0        D. Hunter     D. Mendez       negative acceptable
## 5               0        D. Hunter     D. Mendez      no result acceptable
## 6               0        D. Hunter     D. Mendez      no result acceptable
##   Latitude Longitude Dead_or_sick
## 1    -35.6     148.8    not noted
## 2    -35.6     148.8    not noted
## 3    -35.6     148.8    not noted
## 4    -35.6     148.8    not noted
## 5    -35.6     148.8    not noted
## 6    -35.6     148.8    not noted
```

To find the number of rows and columns we can use the following commands:

```r
dim(data_table)  # Prints the number of rows, followed by the number of columns.
```

```
## [1] 2499   18
```

```r
nrow(data_table)  # Prints the number of rows only.
```

```
## [1] 2499
```

```r
ncol(data_table)  # Prints the number of columns only.
```

```
## [1] 18
```

Individual exercise: Obtain some summary statistics from the columns in data_table. Try using the summary() function on the complete data frame and on one column each time

We can write tables in many formats from R. The most simple method is to save the data as tab delimited or comma separated values, instead of xlsx or other data formats (these require some additional R packages).

We will save the first ten lines of data_table in csv and txt (tab delimited text) by indexing the first ten rows. Open the help file for write.table() for more details on the arguments of the function.

```r
write.table(x = data_table[1:10, ], file = "data_table_10.csv", sep = ",", row.names = F)
write.table(x = data_table[1:10, ], file = "data_table_10.txt", sep = "\t",
    row.names = F)
```

Individual exercises: We used the argument row.names = F, try using T (the default) and look at the output files. Change argument for the col.names and look at the output files

Class exercise: Use the order function to sort the rows of data_table from the latest to the most recent year of collection of the sample. Write this table into a file called data_table_sorted.csv

Class exercise: Use the operation == to obtain a data frame only with samples from the year 1998. Save this to a file called data_table_1998.csv

Data from Excel

R can read data from Excel and other spreadsheet programs. We need to download and install additional tools for this purpose. In this example we will use the package gdata. Download and install it with this command:

```
install.packages("gdata")
```

```
##
## The downloaded binary packages are in
##  /var/folders/17/lpg5qqgx4237271yxywq5_m40000gn/T//RtmpE7WzKz/downloaded_packages
```

We need to load the package to make the functions available. Use the library() function:

```
library(gdata)
```

The data file 'Chytridomycosis_10.xlsx' is the same data that we used in a previous example, but it is in Excel format. We can read it as follows:

```
data_excel <- read.xls("Chytridomycosis_10.xlsx", sheet = 1)
```

Note that we use the argument sheet = 1 to specify that it should load the first sheet.

An other package that can handle spreadsheets is XLConnect. It is more diffult to use, but it has some more advanced functions.

Optional exercise: Reading data from the internet

R has many capabilities to handle data, including maps, phylogenetic trees, and other more complicated structures. We have loaded data from our local machine, but it is also possible to obtain data from the internet, such as a file in Dropbox or a remote server. We will try this by downloading some data from the github repository for the workshop.

Open a web browser and type in the following address:

```
https://github.com/sebastianduchene/intro_to_r/blob/master/tutorial5/Chytridomycosis_short.csv
```

You will note that this is one of the data sets that we have been using. It is stored in a remote server in GitHub. Click the link labeled 'Raw' to view the raw data in your browser. The address should be something like this:

```
https://raw.githubusercontent.com/sebastianduchene/intro_to_r/master/tutorial5/Chytridomycosis_short.cs
```

This is the URL for the raw data. We can read it directly into R with the following code. If you are using a windows machine you may not need to use the method=curl argument. The argument 'destfile' is used to specify the name of the file that we have downloaded.

```
download.file("https://raw.githubusercontent.com/sebastianduchene/intro_to_r/master/tutorial5/Chytridom
    method = "curl", destfile = "web_data.csv")
```

Check the working directory to see that the web_data.csv file was created. You can do this from R using the dir() function:

```
dir()
```

```
##  [1] "Chytridiomycosis_data_1956_2007.txt"
##  [2] "Chytridomycosis_10.csv"
##  [3] "Chytridomycosis_10.xlsx"
##  [4] "Chytridomycosis_short.csv"
##  [5] "data_table_10.csv"
##  [6] "data_table_10.txt"
##  [7] "README.md"
##  [8] "README.md~"
##  [9] "sim_seqs.fasta"
## [10] "sim_tree.tre"
## [11] "tut5_seqs.fasta"
## [12] "tut5_tree.tre"
## [13] "tutorial5.rmd"
## [14] "tutorial5.rmd~"
## [15] "web_data.csv"
## [16] "webtable.csv"
```

We can use this file for any analyses like we did in the previous example.

Optional exercise: Reading sequence data and phylogenetic trees

R can handle many specific applications. The code for these applications is not included in standard R distributions, but it can be loaded by installing packages. In this example we will read a sequence in fasta format and a phylogenetic tree. This can be done with the APE package.

Install it using the following command:

```
install.packages("ape")
```

R may prompt you to select a mirror. This is a remote server where the code and data are stored. Select one that is geographically close to you. To make the functions available, load the package with the library function:

```
library(ape)
```

Download the phylogenetic tree and the fasta file from the web:

```
download.file("https://raw.githubusercontent.com/sebastianduchene/intro_to_r/master/tutorial5/sim_seqs.
    method = "curl", destfile = "tut5_seqs.fasta")
download.file("https://raw.githubusercontent.com/sebastianduchene/intro_to_r/master/tutorial5/sim_tree.
    method = "curl", destfile = "tut5_tree.tre")
```

Use the functions read.dna, and read.tree from APE to read the data:

```
my_tree <- read.tree("tut5_tree.tre")
my_seqs <- read.dna("tut5_seqs.fasta", format = "fasta")
```

Use summary, str, and class to inspect these data. Try the following code:

```
plot(my_tree)
```

In the next section of the workshop we will learn more about plotting.