Advancing in R

Tutorial exercise: Introduction to R and RStudio

Introduction

In this tutorial, we will cover some of the very basics of the R programming language, and attempt some reasonably simple operations including the importing and manipulation of data. We will also make sure that everyone's copy of R is sufficiently up-to-date to run some of the packages we will be using during the course. Even if you already feel comfortable doing basic operations in R and RStudio, please go through the section on updating R and installing packages (find it on pages 5-7 of the tutorial). This will ensure that everyone on the Advancing in R course has what she or he needs without requiring a lot of downloading on an inconsistent wireless connection during the course itself. For those of you unfamiliar with RStudio, especially with RStudio projects (a way of organizing your data files, source files, graphics, etc), proceed through the tutorial in order, as this will ensure you are ready to use RStudio optimally during the course.

If you haven't already done so, you may also wish to review a chapter on the fundamentals of the R language in your favourite R textbook or manual (there are many freely available resources online if you don't have one to hand: try the CRAN page linked below if you don't know where to start). In the tutorial below we will try to define terms carefully, but it may help you if you already have a good grasp of what objects, vectors, lists, and data frames are, for example.

Learning outcomes

Install R and RStudio

Know how to update R

Know how to install R packages

Learn basics of RStudio client

Know how to save R scripts and RStudio projects

Know how to annotate scripts

Know how to import data into a data frame

Know how to summarize vectors and data frames

Know the basics of how to examine object structure and manipulate data frames

Know how to find help

Some useful commands

Below is a list of some (but not all) of the useful commands you will probably need to run at some stage of today's practical. Most of these are names of functions, sometimes with suggested usage, sometimes not. I will try to highlight the growing repertoire of commands you acquire as the module advances. If you need help on usage, pass ?*commandname* to the console as explained in the help section below.

getwd

names

order

read.csv (or read.table)

rm(list=ls())

setwd

str

? and ??

Installations

If you haven't already done so, the first step will be to install R and RStudio.

R is freely available for Linux, Mac OSX, and Windows from the comprehensive R archive network (CRAN) page: http://cran.r-project.org/index.html

Download and install the correct version for your operating system.

RStudio is also free, and available for a number of operating systems. Find it here: http://www.rstudio.com/products/rstudio/download/

Install RStudio, and you should be ready to go.

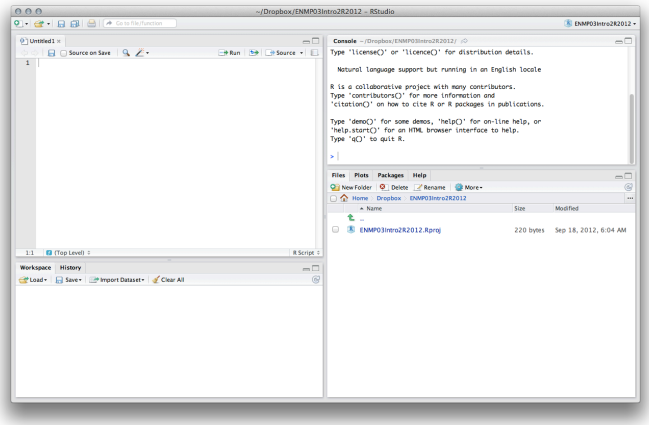
Working in RStudio: R projects

If it's not already running, start the RStudio software. RStudio is a client that operates R for you, and provides several advantages over coding in the native R environment that you will discover as we proceed through the course work.

One of the convenient features of RStudio is that it helps you organize your R projects by grouping saved files together in their own directory. Your saved files will probably include data files, R scripts, any plots you generate, and possibly your workspace. Let's start a new R project using the Project pulldown menu in the upper right hand corner of the R

interface. Select “New Project”, and then use the browser to select or create a folder where you want to store the files. Choose this carefully because later you will need to know where to dump data files belonging to this project. Give the project a descriptive name (e.g., AdvInRTutorial, or something snappier if you prefer), and save it. Normally, you will store all files associated with a project together. What you consider a project is up to you, but typically it might be one experiment, or one practical exercise.

The RStudio interface normally has four panels, but right now you may only see three of them, because our new project has no R script associated with it. Let’s fix this using the File menu (New -> R Script). Now that you have all four panels, you may wish to arrange them so that your layout matches my own preferred layout, as illustrated below. This is achieved by adjusting preferences or options or global options, depending on your operating system. In this layout, the “source” panel is top left, and the “console” panel is top right. While you have the options open, you may also want to check another box that will save you grief later on: select the “Code editing” tab under options, and make sure the box labelled “Soft-wrap R source files” is checked. This option means that long lines of code will naturally wrap within your source panel so that you don’t have to scroll horizontally to read them.



Think of the source panel as the “keyboard” through which you will always communicate with R. It is possible to enter commands directly into the console, but I strongly discourage you from doing this most of the time, as you cannot easily save the console (nor do you normally want to). The console is instead the “display screen” for the non-graphical results of your instructions. Similarly, it is possible to execute some functions using pulldown menus, but because these are never stored in the script, I will also discourage that.

The workspace and history panels in the lower left allow you to quickly monitor the objects in your current workspace as well as the history of commands you have passed to the console.

The last panel has four tabs: “Files”, “Plots”, “Packages”, and “Help”. The “Files” tab allows you to browse your computer for files, and helps you change the working directory if necessary (although if you consistently work in projects, the working directory will always be the one you nominated or created when you set up your project). We will use it shortly. The “Plots” panel is where graphical output from your commands will be displayed. The “Packages” panel will be used to add R add-ons called packages when you need access to specialist code (we will cover this in the first day of the course). Finally, the “Help” panel will display R’s standardized (and probably at first rather puzzling) help pages. More on that later.

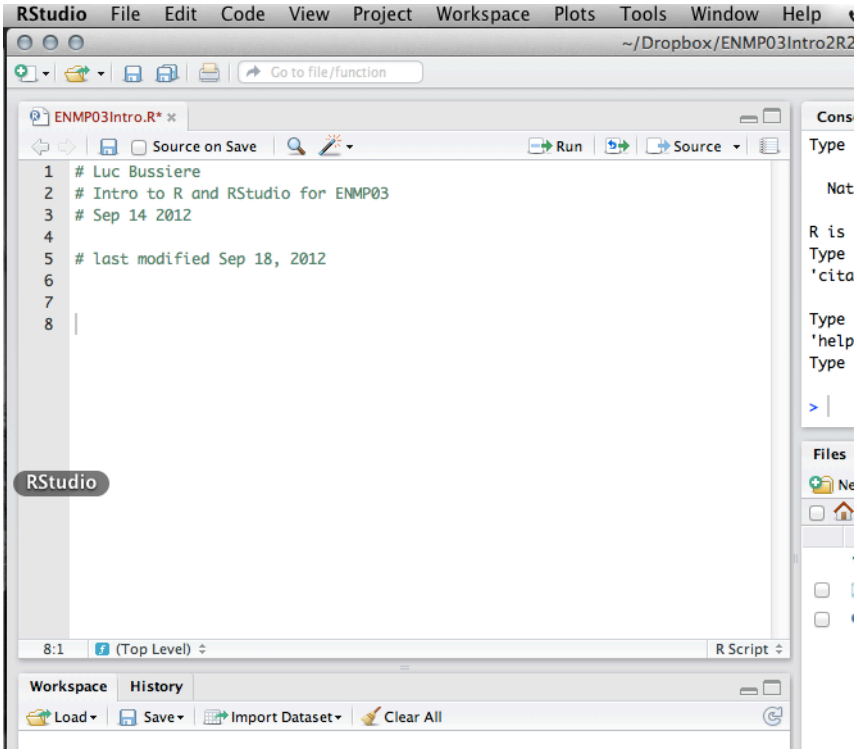
Let’s begin by focussing on the source panel. The source file, or script, is the permanent record of the work you do, so before we go any further, we should try saving it. There is more than one way to do this, but the easiest is to click the floppy disk icon in the upper left of the source panel. Name your script something descriptive. It will be saved into your project folder, and you may notice that your “Files” panel now shows the two files you have created within that folder: one project file (something principally useful for you when working on this project in RStudio) and one script file (a portable set of instructions for the R console that you can share with anyone in the world, regardless of what platform or client they use to operate in R). When you share your script with someone, they will want the file with the .R suffix, as that’s the one containing the instructions. Please don’t mistakenly send .Rproj files instead!

Annotations

Because the R script is a permanent record of your instructions, it is essential that you annotate this list very well, so that when you try to run the script again two months from now (having received requests for revision from your colleagues or a scientific reviewer), your future self will know exactly what each section of code is for. Furthermore, as you become an adept R user, you will want to share bits of code with collaborators and future students, and they will definitely need help to navigate your script. R has a very simple annotation methodology: anything following a hash key (#) will not be treated as code by the R console. This means you can add lines of annotations (or alternatively short annotations at the end of lines) by preceding them with a hash key. Try this now by adding a few lines to the top of the script that describe the R script. I suggest your name, a description of the file, the date the script was first created, and perhaps the date it was last modified (see below). You can add empty lines anywhere in the R Script to improve its clarity or separate sections, and these will never affect the code. If you can’t find the hash key, try alt-3 (or right alt-3). Alternatively, you can select multiple lines of code and use command-shift-C (control-shift-C) to annotate (or un-annotate) blocks of text (C is for “comment”, to help you remember).

There is another extra function of the hash key in RStudio that you will find useful: try typing in '# Section 1 ####'. You will see that, at the bottom of the source panel, there is now a drop-down menu that says 'Section 1'. Using 4+ hashes at the end of a comment line denotes a different 'chunk' of code, and the drop-down menu enables you to skip between them quickly (rather than scrolling endlessly!). By using informative section headers (e.g., 'Overview', 'Data loading', 'Exploratory plots', etc), you will be able to quickly navigate through your scripts later on.

You may have noticed that the name of your R script has now turned red, and is followed by an asterisk. This is RStudio telling you that you have made alterations to the script file since your last save. If you are happy with the changes, save your script (using the floppy disk icon), and you will see the filename go back to black.



Updating R and installing packages

If you have not already done so, you may need to select a default repository where R can search for the package files you want. This setting is available under the Tools menu, then Global Options, then the Packages tab. You'll see a dialog for the CRAN mirror at the top of the Package Management options screen, and have the option to change or select one if there is nothing there. Since I live in Scotland, I normally choose the one in St. Andrews, but it makes virtually no difference which one you choose. Once you have selected a repository, click OK to exit the options dialog box.

To take advantage of the latest packages in R (including several that we will use in the course; see below), please make sure that you have the latest version of R. You can check your version by typing the following command in the source panel (upper left), and then selecting the text and hitting hit command-Enter (or control-Enter, depending on your operating system).

```
version
```

Note that you can also pass commands to the console one line at a time by simply positioning the cursor anywhere in the line you want "passed" to the console (i.e., you don't need to select the whole line if you just want to pass one of them). Either way, you should see that the text you have selected has been passed from the script to the console (the commands will appear in the console window), and executed.

If your R version needs updating (the current version as I write this is 3.2.2), one of the easiest ways to do this is simply via the {installr} package (although this does not work on Mac OS, where you instead need to download and install R from CRAN, as instructed below). You can load and use installr by entering the following commands in your console:

```
install.packages("installr")
```

```
library(installr)
```

Next, use the 'updateR()' function to install the latest version:

```
updateR()
```

When it asks whether you want to copy your packages over to the new installation, be sure to hit 'yes'! It's usually worth updating your packages (Tools -> Check for package updates) after installing a new version.

If you cannot use installr, simply download and install the latest version of R from CRAN, and RStudio should automatically detect this upon restart (again, use the 'version' command to check). If you have already installed custom packages to your own library, you may have to copy them over to the new one (then update your packages). You can use the following command to find where your R package library is:

```
.libPaths()
```

…after you have copied packages to your new library, remember to then update all packages as above.

Installing packages

One of R's great strengths is that it is open source software, and that thousands of expert users have freely contributed their own code to increase its abilities. As I write this in November 2015, there are 7488 R packages that you can download from CRAN (the comprehensive R archive network, which we will revisit later in the tutorial). By the time you complete this tutorial exercise, that number will undoubtedly have increased (you can check here if you like: http://cran.r-project.org/web/packages/).

During our course, we will exploit a tiny fraction of this expert coding, and to do so, you will need to install R packages on your machine. If you completed the exercise above, you may have already installed package "installr". Before you continue the tutorial, we recommend you also install the following packages (and/or ensure that all of these are up-to-date). We will not need them for today's tutorial, but the wireless network can be spotty during the course, especially if many people are trying to access it at once on multiple devices, and you will make your own life a lot easier if you have installed all the necessary packages before you begin the course.

You can install packages in a few ways. The first involves going to the 'Tools' menu and selecting 'Install packages…'. In the dialog box, type in the names of the packages (they should autofill). Make sure that you have selected 'Repository (CRAN)' under the 'Install from' section, and that the box indicating "Install dependencies" is checked. (You can also find this dialog by navigating to the "Packages" tab in the lower-right quadrant, or panel, of the RStudio interface (at least it is the lower right quadrant by default, unless you have changed this). Once you have selected the right tab, you should see an "Install" option, which will open the same dialog box as you can access via the Tools menu.)

You will notice that clicking "Install" in the dialog box causes a command to be automatically entered in the console, which is another way to install named packages. This is what we asked you to do with the command that installed the package "installr", above. You can modify this command to install more than one package at a time by "concatenating" package names using the c() function, nested within the install.packages() function. For example, the following command will install both tidyr and dplyr packages.

```
install.packages(c("tidyr", "dplyr"), dependencies = TRUE)
```

The 'dependencies = TRUE' part informs R that you want to install any additional packages that are required for these packages to run correctly; we strongly recommend installing your packages in this way so that you are not missing anything that's needed!

Using one or more of the methods described above, install each of the R packages in the list below, which we may use at some stage of the Advancing in R course. Note that you may need to wait some time for packages to download, especially if you are not accessing

the internet through a high speed connection. Make sure that "install dependencies" is checked in the dialog box if you use that method – this will ensure that other packages on which the focal one depends are also installed.

Packages to install for the Advancing in R course:

arm
boot
broom
car
dplyr
ggplot2
lme4
MASS
Matrix
MuMIn *(NB: the fourth letter is an uppercase "i")*
nlme
nlsTools
sjPlot
tidyr
visreg

Clearing the workspace

Because R is really a programming language, rather than merely a statistical package, one of the habits we'll need to get into is to adhere to good coding practice that will help your machine to run effectively. It is usually good practice to clear your workspace before beginning work on a project, so we tend to always add the following lines to the top of our scripts, just after the information on dates. (Note, from now on we will often add bits of code directly to the text instead of always taking screenshots.)

```
# clear R of all objects
rm(list=ls())
```

Note the first line, which is annotation (and therefore not executed if it is passed to the console). The second line contains two functions nested within one another and the arguments used by the functions. For now, it's perhaps best to just trust us and enter the code as written, but if you want to know more we will provide some more direction at the end of the tutorial. Briefly, this line clears R's workspace to make sure you aren't operating with any clutter in the memory.

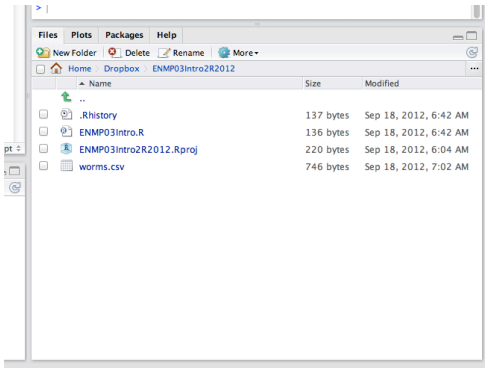
Once you have it entered, transfer the lines from the script to the console using command-Enter (or control-Enter). You should see again that the text you have selected has been "passed" from the script to the console, and executed. Since most of the text was annotation, R hasn't done anything with that. The only command was to clear the

workspace, which was probably already clear for you anyway. But it's always good practice to make sure!

Getting objects into R

Although it is possible to enter data manually into R, you will usually import your data from files created in other packages such as MS Excel. While R can read directly from .xls and .xlsx documents using dedicated packages (see: http://blog.rstudio.org/2015/04/15/readxl-0-1-0/), we typically convert them to .csv files for easy import. (NB R can also import .txt files – in some contexts it may be preferable to use tab-delimited text files, e.g., if you or your colleagues use a language that sometimes interprets the comma as a decimal point.)

In order to import .csv data into an object called a data frame, we will use the read.csv function. In preparation, make sure to move the data file you want to work on (worms.csv, provided along with this tutorial) to the folder you created for your project. If you have done this correctly, it should appear in the list of files displayed by default in the Files panel.



The read.csv() function is designed specifically for .csv data, so using it is relatively simple for this particular file:

```
# import the data; note that read.csv only works for .csv files
WORMS<-read.csv("worms.csv")
```

(From now on I will leave the annotations up to you, but make sure you continue to annotate!)

Note the "arrow" symbol that follows WORMS in that command: a less than symbol followed by a hyphen (<-). It is called the "**assignment operator**", which is the operator that assigns value to an object. It's important to realize that R only remembers the most recent assignment value, so if you make a mistake, you can simply correct it and re-execute the code to change the value of the object. In RStudio, the keyboard shortcut for the assignment operator is 'alt--' ('alt' and the '-' key). Take a look at other keyboard shortcuts that are available by going to the 'Help' menu and selecting 'Keyboard shortcuts'.

Once you have passed these lines from the script to the console, you should notice a new object appear in your workspace called WORMS. Note that I often name objects using capital letters to avoid any conflicts with R functions, which are almost never in allcaps.

You can inspect the data through the console. One option is to "call" the object itself.

```
# examine the object in the console
WORMS
```

(It is also possible to inspect the data by clicking on the object name in the source panel, which will open a new window that hopefully resembles a spreadsheet. But try to avoid this! Clicking on objects does not only display them; it may also try to edit or fix them, which can have disastrous consequences for the reproducibility of your analyses, because you will be changing the values of your data! I strongly urge you to always examine and modify objects using text commands recorded in the R script rather than with the mouse.)

Note that R is case-sensitive, so if you use uppercase letters to name an object, R won't be able to find it when written in lowercase letters. (Try this to see what kind of error message is generated. Note that error messages have information within them! This will often help you decipher what exactly went wrong.)

You will often find that your data file is too big to conveniently examine within the console, so you may instead wish to have the console show only the first six lines of the data frame. This is achieved using the following syntax:

```
head(WORMS)
```

The default number of lines is 6, but you can request a different number by passing that as an additional 'argument' to the 'head' function:

```
head(WORMS, 8)
```

You may not be surprised to find that you can examine the **last** 6 lines of the data frame by using the 'tail' function…!

Try out the following commands as well

```
names(WORMS)
summary(WORMS)
str(WORMS)
```

The last function, str(), is a very useful command for examining the structure of your data frame (it is always worth looking at the structure of your imported data to check whether everything has been imported correctly). It tells you how many observations and variables there are, and it also gives you information on what kinds of 'vectors' you have in the data frame.

```
> str(WORMS)
'data.frame': 20 obs. of  7 variables:
 $ FIELD.NAME  : Factor w/ 20 levels "Ashurst","Cheapside",..: 8 17 10 16 7 11 3 1 19 15 ...
 $ AREA        : num  3.6 5.1 2.8 2.4 3.8 3.1 3.5 2.1 1.9 1.5 ...
 $ SLOPE       : int  11 2 3 5 0 2 3 0 0 4 ...
 $ VEGETATION  : Factor w/ 5 levels "Arable","Grassland",..: 2 1 2 3 5 2 2 1 4 2 ...
 $ SOIL.PH     : num  4.1 5.2 4.3 4.9 4.2 3.9 4.2 4.8 5.7 5 ...
 $ DAMP        : logi  FALSE FALSE FALSE TRUE FALSE FALSE ...
 $ WORM.DENSITY: int  4 7 2 5 6 2 3 4 9 7 ...
>
```

Note that R distinguishes between continuous numbers and integers, which is very useful for a statistical package because those two classes of vector have very different statistical properties.

It is possible to "call" a single vector from within an object by specifying its address:

```
WORMS$AREA
```

The dollar sign tells R to look within the first object, WORMS, for the vector AREA.

In addition to examining the object, you can easily perform operations on it, using operators or functions. Try the following:

```
mean(WORMS$AREA)

sd(WORMS$DENSITY)
```

The last command will generate a cryptic "NA" response. Why? What do you think NA means? Examine the data frame and the command carefully and see if you can find the error in my code. Correct the error and see if you can get a good measure of the standard deviation for worm density.

Now try the following command:

```
MEAN.SOIL.PH<-mean(WORMS$SOIL.PH)
```

Note that in the previous two commands, the output is generated and then forgotten by R. In the third, you have assigned the value to a new object (which is available for subsequent manipulation if you need it), but it is not displayed. How would you examine this new object? Do this now.

Examining and manipulating data (the basics)

You will often want to examine a specific part of a data frame or vector, rather than the whole thing. In this part of the tutorial, we will give you some basic tools for examining parts of your data frames, and we will expand on this toolkit during the course proper. You must first realize that all rows and columns of your data frame are numbered, and that individual cells in the data frame can be "called" by referring to their row and column coordinates using square brackets, [ ]. Omitting a coordinate means you want to look at all of the records. The number preceding the comma always refers to rows, and the number after the comma always refers to columns. So if you want to examine the AREA for Nursery Field, you need to translate this into coordinates. Nursery field is in the third row, and AREA is the second column. (You can check this by viewing the whole object WORMS again.)

```
WORMS[3,2]
```

I can translate the above line into an English expansion of the coding instructions:

```
WORMS   [          3       ,       2   ]
  ↑     ↑          ↑               ↑
```

"Give me the object known as WORMS where row number = 3 and column number = 2"

If you want to examine the whole vector for SLOPE, you could either name the vector

```
WORMS$SLOPE
```

Or you can use square brackets and omit the row information

```
WORMS[,3]
```

How would you request rows 5-15 of the data frame (hint – use ":" to specify a range of rows or columns)? How would you store this information for later use?

You can also include operators within square brackets to specify conditions that entries must meet to be included. Which rows are selected using the following code?

```
WORMS[WORMS$AREA>3,]
WORMS[WORMS$AREA>3 & WORMS$SLOPE<3,]
```

Note that I refer to WORMS twice in the first command – once to specify that R should look within the object WORMS, and once to specify the vector of WORMS that I want to satisfy a given condition. If I forget to give the full address for this vector (i.e., if I ask for `WORMS[AREA>3,])`, R will tell me that it cannot find the object AREA. Some tutorials and guides will suggest that you use the function attach() to circumvent having to provide full addresses, but this is a **seriously bad idea**, because in our more advanced exercises you will be moving back and forth across multiple objects and detaching and attaching all of them will become incredibly tedious and confusing.

You can also sort the data according to one of the vectors, using the command order(). The following code orders columns 1:6 by AREA (column 2). Note that I have 'nested' a function, order, within the call.

```
WORMS[order(WORMS[,2]),1:6]
```

You can nest more than one function at a time. The command rev() reverses the order of a vector. Try creating a new appropriately named object that contains rows sorted in decreasing order of SOIL.PH.

We have already used one kind of operator, >, to specify conditions that must be met. R has a long list of these that may be useful (<, <=, >, >=, == for exact equality and != for inequality). Note that the 'identity' operator is two equals signs together, ==, and that 'unequal to' is an exclamation point followed by an equals sign, !=. A single equal sign is usually equivalent to the assignment operator <- but because that can get very confusing I like to stick to the <- and avoid using = in my own coding.

Store the following subsets in well-named objects. Note that you don't necessarily need to do any of these in a single step! Store objects that break down the task if it helps.

All records where DAMP is TRUE
All records with Meadow VEGETATION
All records in Grasslands, sorted by DENSITY
The records where density < 5.5, sorted by SLOPE
The AREA and SOIL.PH for all plots not in Grassland

NB: Knowing how to use square brackets to manipulate data frames is very useful, but for complex data manipulation problems, this approach can become tedious or impractical. During the Advancing in R course we will highlight more flexible approaches to data manipulation using some recently released R packages.

Getting help

There are many alternative sources for help on using R. We can only teach you a small fraction of what you will need, so you must be sufficiently resourceful and confident to find and use help from various sources. R includes its own help documentation. If you know the name of the function for which you want help, just execute a command with a question mark preceding the function:

```
?mean
```

R includes its own help documentation, but the format takes some getting used to. It always includes a description of the object, a template for how to use the object, a description of the optional arguments, more details on usage, authorship information, and finally some example bits of code (which can incidentally always be copied directly into your console if you want to see the function illustrated).

Often you may not know the name of a function, so using two consecutive question marks allows you to search help files for a term. Suppose you didn't know that the function for standard deviation was sd, you might search for standard deviation (you can't include spaces in your search, so you must replace the space with a fullstop).

```
??standard.deviation
```

The results of your search (if there are any) will be a list of help pages containing the phrase specified. The name of each function begins with the package containing the function (e.g., the sd() function is from the 'stats' package. We'll cover packages in more detail later in the module.

Can you find out what the functions are for calculating the median, mode, maximum and minimum of a vector? What about the square root, or the natural log, or log base 10? Try these functions out on some of the vectors in your data frame. You may need to be resourceful in your search….

Another challenge: on page 8, we asked you to clear R's workspace by entering the following command:

```
# clear R of all objects
rm(list=ls())
```

Maybe now you are ready to interpret the details? There are two nested functions. Ask for the help file on rm() and ls(), and try to interpret what each of them is doing, and how they work together?

In addition to the help files, R functions are often but not always illustrated with examples (for short functions) and demonstrations (for more involved procedures). Use the example() or demo() functions to run these.

```
example(sd)
```

```
demo(lm.glm)
```

There are of course many resources for help outside the R environment that you may prefer to rely on. Textbooks, instructors, and colleagues may all be able to help, but you should also consult online resources including CRAN. Google may also help with some queries, as its ability to search specific error messages can often rescue an otherwise insoluble situation. However, many of the answers you find on Google will be dated – since R is a dynamically evolving language, some of the advice you get may be out of date by the tie you read it. Be persistent and resourceful, and patient in deciphering what seems like (and is!) a rather foreign language at the start of your learning curve.

Once you are satisfied with your ability to search for help, make sure you have saved your R script. If you want, you can also save the workspace, which means that next time R will open this project with all objects intact (rather than needing you to run the script to generate the objects again).

~ End of Tutorial ~