G54DMA - Lab 1: Introduction to R

Instructions

1. Introduction

'R' is a programming environment for data analysis and graphics, developed as an open source application. It is a powerful and flexible tool with a vast number of toolboxes. You can find copies of R on the windows machines in A32.

In this lab, you will enter sometimes unfamiliar/obscure R syntax into the command window. In order to get the best out of the lab sessions, you should try to understand how they work and what they do before moving on.

For a comprehensive reference manual on all things R, please refer to *An Introduction to R* by Venables et al., which can be found in our Moodle page and here:
https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf

I recommend that you print it and have it with you while you are working on the exercises.

Thorough the labs, we will be using *An Introduction to R* as a reference manual. Chapters or sections of interest will be signalled with "***Reading:***" in the Instruction sheets.

1.1 Learning outcomes

After this lab session, you will be able to use R to:

- **Use the R help function**
- **Perform simple arithmetic operations**
- **Perform simple vector calculations**
- **Perform simple matrix calculations**
- **Draw simple plots**
- **Install and use packages**

2. Getting Started

R is freely available open-source software, which is distributed in easy to install binary packages for most popular current operating systems, including Microsoft Windows, Apple Mac OS X, and versions of Linux. You can easily install R on your own computer, if you wish, by obtaining the current version from:

http://www.r-project.org

2.2. RStudio

RStudio is a powerful integrated development environment (IDE) for R. I recommend that you install and use it as your main R editor. You can download RStudio Desktop from:

https://www.rstudio.com/

The typical RStudio interface is shown in Figure 1. It is made up of different windows.

1. Script/Editor window (top left): Scripts are collections of commands that can be saved and edited. To run a command or a script in RStudio, you will need to get it into the command window. To do this, you can select the instructions that you want to execute and click Run or CTRL+ENTER. To open a new script, go to File -> New -> R script.

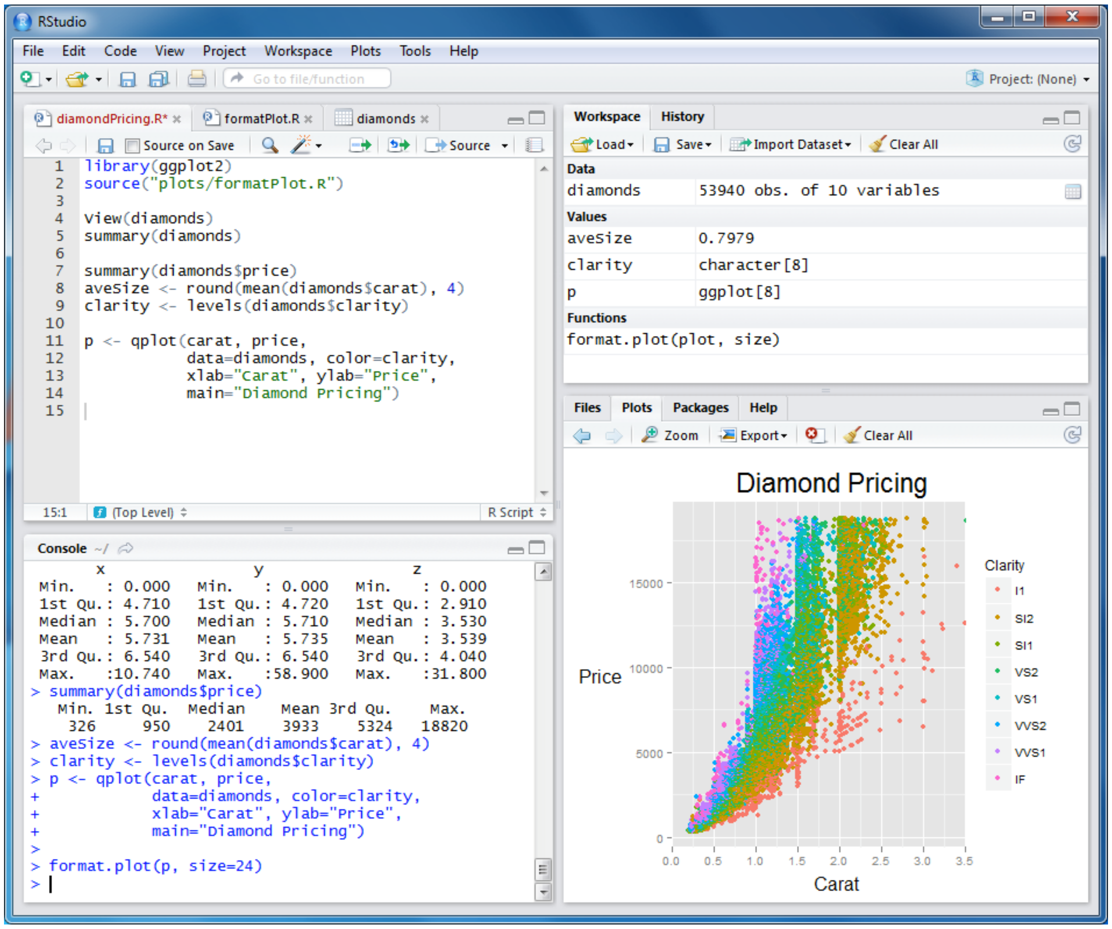
2. Console/Command window (bottom left): You can type R commands in this window. Results from your instructions will also appear in this window. The command line starts with **>** . You will type your commands after this symbol.

3. Workspace/History window (top right): The Workspace tab has information about the data and functions that R has in memory. The History window has everything that has been typed or sent to the command window before.

4. Files/Plots/Packages/Help window (bottom right): One of the most useful features of RStudio, this window allows you to open files, view graphs, and install and load packages.

I recommend that you save all your work in the labs in scripts. That way, you can edit, expand and rerun your exercises.

Reading: For more information about the R environment, read Chapter 1 of *An Introduction to R* by Venables et al.


RStudio
File Edit Code View Project Workspace Plots Tools Help
Go to file/function
Project: (None)
diamondPricing.R* formatPlot.R diamonds
Source on Save
Source
15:1 (Top Level) R Script
Console ~/
Workspace History
Load Save Import Dataset Clear All
Files Plots Packages Help
Zoom Export Clear All
Diamond Pricing
Price
Carat
Clarity
I1
SI2
SI1
VS2
VS1
VVS2
VVS1
IF

```r
library(ggplot2)
source("plots/formatPlot.R")

view(diamonds)
summary(diamonds)

summary(diamonds$price)
aveSize <- round(mean(diamonds$carat), 4)
clarity <- levels(diamonds$clarity)

p <- qplot(carat, price,
           data=diamonds, color=clarity,
           xlab="Carat", ylab="Price",
           main="Diamond Pricing")
```

```
Data
diamonds      53940 obs. of 10 variables
Values
aveSize       0.7979
clarity       character[8]
p             ggplot[8]
Functions
format.plot(plot, size)
```

```
       x                y                z
 Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
 1st Qu.: 4.710   1st Qu.: 4.720   1st Qu.: 2.910
 Median : 5.700   Median : 5.710   Median : 3.530
 Mean   : 5.731   Mean   : 5.735   Mean   : 3.539
 3rd Qu.: 6.540   3rd Qu.: 6.540   3rd Qu.: 4.040
 Max.   :10.740   Max.   :58.900   Max.   :31.800
> summary(diamonds$price)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    326     950    2401    3933    5324   18820
> aveSize <- round(mean(diamonds$carat), 4)
> clarity <- levels(diamonds$clarity)
> p <- qplot(carat, price,
+            data=diamonds, color=clarity,
+            xlab="Carat", ylab="Price",
+            main="Diamond Pricing")
>
> format.plot(p, size=24)
> |
```

Figure 1. RStudio editor. The main interface consists of the **Script window** (top left), **Command window** (bottom left), the **Workspace window** (top right), and the **Plots/Files/Packages window** (bottom right).

2.3. Working Directory

The *working directory* is the folder in your computer in which your scripts and data will be saved. The first thing you need to do is to set your working directory to the folder where you want to store your code and graphs.

You can check where your current working directory is with:

```
getwd()
```

And you can change your working directory with:

```
setwd("Path/to/your/working/directory")
```

Alternatively, you can use the **Files** tab in RStudio (bottom right window) to set your working directory. First, in the Files tab, navigate towards your desired working directory. Once there, click on “More -> Set as working directory”.

3. Help in R

The R help files are a useful source of information – simply type your keyword into the following syntax:

```
help(plot)
```

or with keywords:

```
help.search("line")
```

If you want to find out more about the help function then use:

```
help(help)
```

If you want to keep track of what you are doing try the history command (use the help to find out about it).

If you want to keep all your work, you can save the workspace either through the application (under ‘workspace’) or through the command prompt – again see help on save for the details.

To exit the program using the command line use the quit() command.

Reading: For more information about the *help* function, read point 1.7 from *An Introduction to R*.

4. Arithmetic and logical operations

Variables can be assigned by using the following syntax:

```
    num <- 2
    num = 2
```

Here ‘num’ is assigned the value of 2. A number of R commands use the <- notation for assignment. However, in nearly all cases, an equals sign (=) will also work.

The most common operators are shown in Table 1 and Table 2:

Table 1. Arithmetic Operators

Operator	**Description**
+	addition
-	subtraction
*****	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/%2 is 2

Table 2. Logical Operators

Operator	**Description**
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x | y	x OR y
x & y	x AND y
isTRUE(x)	test if X is TRUE

5. Vector and Matrix operations

Vectors are a special case of matrices, in which one of the dimensions (either rows or columns) is one.

Reading: For more information about Vectors, read Chapter 2 of *An Introduction to R* by Venables et al.

The R language has its origins in mathematical and statistical computing, and its main type of variable is the '*matrix*'. A matrix is a two dimensional list of numbers, similar to a two-dimensional array in other programming languages, but with properties similar to the mathematical notion of a matrix.

To input the matrix $\begin{pmatrix} 1, 2 \\ 3, 4 \end{pmatrix}$ input the following.

```
a <- matrix(c(1,2,3,4), nrow = 2, ncol = 2, byrow=TRUE)
```

where 'matrix' instructs R to accept a matrix, nrow/ncol are the number of rows and columns respectively, and 'byrow' is if the input is by row (true) or by column (false). The function *c* combines all its parameters into a vector.

If you want to find out more about these functions, type ***help(matrix)*** and ***help(c)***.

Type 'a' and the output should look something like:

```
> a
      [,1] [,2]
[1,]     1    2
[2,]     3    4
```

As with many open source packages like this there are often a number of ways to perform the same operation. Have a look at how the following commands can be used to enter a matrix. What are the differences if any?

```
a = rbind(c(1,2), c(3,4))
```

Now enter another matrix into variable b:

```
b = rbind(c(1,3), c(1,4))
```

We want to check that R knows how to multiply and add matrices correctly. The syntax for adding is simply '+'. The syntax for multiplying whole matrices is %*% – enter the following commands and check the answers.

```
a + b
```

```
a %*% b
```

Let's create another new matrix c:

```
c= rbind(c(1,2,3), c(3,4,5))
```

Recall that we can't always multiply matrices if they are incompatible in size. Lets try to multiply c*a. Does it return an error?

```
c %*% a
```

Can we get a * c though?

```
a %*% c
```

In order to multiply matrices element-by-element, the notation is different – simply a*a.

Now try the following and make sure you understand the differences between a*a and a %*% a :

```
a * a
```

Similarly for powers of the numbers in the matrix:

```
a^3
```

If you want to store the answer, another variable, say *y*, can be set to be the answer. As R is a programming language, variable assignment is possible in the usual manner.

```
y= a%*%c
```

Now you can use *y* in a calculation

```
y + 4
```

Very frequently, you will need to access particular elements and indexes within a matrix. This can be done with the bracket [] operator. To access the element in the *i*th row and the *j*th column in matrix a, index it as *a[i,j]*. Remember that the row index goes before the column index.

```
a[1,2]

a[2,1]
```

To access a whole column or a whole row, use a space, like this:

```
a[1,] #returns first row

a[,1] #returns first column
```

You can also add conditions between the brackets, in that case, only the elements of the matrix that comply with such conditions are returned.

```
a[a>=2]
```

Returns:

```
[1] 3 2 4
```

What is the difference between the instructions a>=2 and a[a>=2]?

R has many built-in capabilities to perform various manipulations on matrices, such as the ability to transpose matrices. This can be performed using the aperm() function – this is a useful function with a range of transposition options. However, the quick notation for the plain transpose is:

```
t(y)
```

The transpose flips rows and columns. This is useful in situations where the parameters to a function require a column vector when our data is a row vector.

If we want to add an additional column to an existing matrix we can use the cbind( ) command.

```
z = cbind(c, c(1,2))
```

Reading: For more information about Matrices, read Chapter 5 of *An Introduction to R* by Venables et al.

6. Sequences and Plotting simple functions

Entering data by hand can be tedious so R allows creating uniformly spaced data points:

```
a= 0:100
```

Sequences of non-integer numbers can be done using this:

```
b= seq(from =0, to=1, by =.01) #by determines the interval
b1= seq(1,100,20)
b2 = seq(1,15, length.out=8) #length.out determines the length of
the output
```

You have probably realized by now that the output of assignments is automatically suppressed. However, just type 'b' now to show you the variable.

Now, just create something to plot, e.g.:

```
c= b ^ 2
```

R comes with built in graphics capability and is a useful tool for investigating data and for plotting advanced graphs and charts, including a wide variety of 3D plots.

```
plot(a, c)
```

See the help on plot to see the various variants of plotting. For example, see the effect of the following:

```
plot(a, c, col="red", pch="*")
```

While the graphical window is still open, then add a title and axes labels using the following command.

```
title(main = "A Title", xlab = "X Label", ylab = "Y Label")
```

You can actually do this all within the plot command.

```
plot(a, c, col="red", pch="*", main = "A Title", xlab = "X Label",
ylab = "Y Label")
```

We can use the lines command to plot more than one data series on one figure (or, as in this case, add a second set of symbols for the same data series):

```
lines(a, c, col=”blue”)
```

Reading: For more information about the *plot()* function, read points 12.1.1, 12.4, and 12.5 of *An Introduction to R* by Venables et al (we will cover more advanced plotting commands in the upcoming weeks).

7. Installing packages

From time to time, you might need to extend the functionality present in R with code from third parties. Part of the reason R has become so popular is the vast array of packages available at the cran and bioconductor repositories. In the last few years, the number of packages has grown exponentially!

The five most popular R packages are:

- *dplyr*, a grammar of data manipulation
- *devtools*, a collection of package development tools
- *foreign*, read data stored by Minitab, S, SAS, SPSS, Stata, and more
- *cluster*, methods for cluster analysis
- *ggplot2*, an implementation of the grammar of graphics in R

To install a package, you can use the *install* function:

```
install.packages("name")
```

Then, to load the package and use it, use the *library* function:

```
library ("name", "location")
```

The location of the library can sometimes be omitted.

You only need to install a package once but, in order to use it, you will need to load it at the beginning of each session.

Reading: For more information about Packages, read Chapter 13 of *An Introduction to R* by Venables et al.