R Command Summary

Steve Ambler*
Département des sciences économiques
École des sciences de la gestion
Université du Québec à Montréal

April 2018

This document describes some of the basic *R* commands. It should prove useful as a short introduction to *R* with an emphasis on its use in econometrics.[1]

Note that **all** commands in *R* are **case-sensitive**. Linux and Unix users should be used to this. This is one of the most common sources of error for Windows users.

In what follows, "R>" is the command prompt. Type only what follows the command prompt.

Basic

Start an *R* session:	`R> R`
Start an *R* session as root (Linux):	`R> sudo R`
Quit an *R* session:	`R> quit()`
Quit an *R* session:	`R> q()`
Get working directory:	`R> getwd()`
Change working directory:	`R> setwd("/home/username/path")`
List files in current directory:	`R> list.files()`
List files in current directory:	`R> dir()`
Invoke help pages:	`R> help.start()`
Help on command `lm`:	`R> help(lm)`
Help on command `lm`:	`R> ?lm`
Example using `lm`	`R> example("lm")`
Arguments of command `lm`	`R> args(lm)`
Load package `AER`:	`R> library("AER")`
Install package `AER`:	`R> install.packages("AER")`

*ambler.steven@uqam.ca.
[1]It should cover many of the commands needed for the assignments in ECO4272.

The `example(·)` command is particularly useful. For any (almost any?) command available in *R*, the command will execute a simple example of the command and show the output it produces, with explanations.

Using Commands from Libraries

Some *R* commands are defined by default or included in the base version of the program when it is first loaded into memory. Others are contained in packages that must first be installed and then loaded into memory before they can be used. For example, the `kurtosis` and `skewness` commands in the section below on statistical functions are not defined by default. They are available in the "`moments`" package. If a needed package is not installed it can be with the command `install.packages(·)`.[2]

Note that under Linux packages are installed by default in directories that have system-wide access and that are not writable by ordinary users. It is better to start up a temporary *R* session as root (or administrator or super-user) before invoking the command `install.packages(·)`. To do this, start *R* using the command `sudo R`. If you install packages as an ordinary user, they will be installed in your home directory and will not in general be available to other users.

Other examples include the `coeftest(·)` command, which is from the "`lmtest`" package, using the heteroskedasticity-consistent (robust) variance-covariance matrix `vcovHC` from the "`sandwich`" package. Load such packages into memory using the `library(·)` command. See the previous section for usage.

Elementary Commands

Assignment operator:	$<-$
Assignment operator:	$=$
List all objects:	`R> ls()`
List details of all objects:	`R> ls.str()`
List details of `myobject`:	`R> str(myobject)`
List files in current directory:	`R> list.files()`
List files in current directory:	`R> dir()`
Print (to the screen) `myobject`:	`R> myobject`
Remove `myobject` from memory:	`R> rm(myobject)`
Remove all objects in working space:	`R> rm(list=ls())`

[2] In Linux, packages can be installed while running *R* using the `install.packages(·)` command. When running *R* as an ordinary user, the commands will be installed in a subdirectory of the user's home directory and will therefore only be available to that user. It is usually advisable to run *R* as root (superuser) and then install the needed packages. They will then be installed in a default sub-directory of a directory containing system-wide commands (generally, I believe, `/usr/bin/R/`), and will be available to all users on a shared machine. The other advantage is that the user's home directory does not become cluttered with extraneous sub-directories. In some distributions, for example *Debian* and *Ubuntu*, many (not all) packages are available in the distribution's own package repositories and can therefore be installed using the distribution's package manager, without starting *R* itself.

Data Management

In order to estimate an econometric model or to analyze the properties of a data set, the first step invariably is to load the data into memor. *R* has a native format for storing data in `data frames`. It also allows the user to import data that have been saved in other formats (see below). Some data frames are contained in *R* packages. For example, the package `AER` (for "Applied Econometrics in *R*") has several data frames, including `GrowthDJ`, a cross-country data set of macroeconomic variables related to economic growth. Say we want to load the data frame `GrowthDJ`: from the *R* library `AER`. We would use the command

```
R> data("GrowthDJ", package="AER")
```

In order to be able to perform functions on variables within a data set, it can be "attached". This adds the data set to the search path so, when you refer to a variable *R* searches the data set itself for the variable's name.

```
R> attach(GrowthDJ)
```

There is a possible drawback to doing this. All of the variable names in the data set become defined in *R*. If there is a conflict between a named variable and a pre-defined command in *R* (either in the default version of *R* or in a user-loaded library) this can lead to errors. Some guides to *R* recommend against using `attach(·)` for this reason.

Other Formats

We often need to read data that have been stored in text format. The following reads the contents of the file `tmp.txt` and assigns it to an object named `mydata`.

```
R> FILE <- http://people.su.se/ ma/R_intro/data/tmp.txt
R> mydata <- read.table(file = FILE, header = TRUE)
R> mydata
```

The data file has a first row with variable names: hence the use of the `header = TRUE` option. If the columns in the file were separated by commas, the syntax would be

```
mydata <- read.table(file = FILE, header = TRUE, sep=",")
```

We often need to read data that have been saved in other formats by other econometrics or statistical packages, for example *STATA* data files. We can use the `foreign` library to do this. See the documentation for this library on all the formats that are supported. First enable the `foreign` package.

```
R> library(foreign)
```

Now import the data from the file `mydata.dta`.

```
R> lnu <- read.dta(file = "wage.dta")
```

We can also export *R* data into other formats. To export data into a *STATA* file, use

```
R> write.dta(mydata, file = "mydata.dta")
```

For further information on importing and exporting data from other data formats, see the article "R Data Import/Export" available at the following address.

https://cran.r-project.org/doc/manuals/R-data.pdf

Data are often available in the form of *Excel* spreadsheets. It is probably advisable to use *Excel* itself to write out the data to a text file and then use the `read.table(·)` command described above. In fact, the "R Data Import/Export" document referred to above advises, "The first piece of advice is to avoid doing so if possible!"

There is at least one package, called `xlsx`, that will allow the user to import *Excel* data files directly. Study the documentation carefully and check the results. See the following link for more information. Use at your own risk!

https://www.datacamp.com/community/tutorials/r-tutorial-read-excel-into-r#gs.3M349Ic

Saving Output

Once we have analyzed a data set and estimated an econometric model, we very often want to or need to save the results. Say we have estimated the model `cars` and want to save the output in a file. We can do this using `sink()`:

```
R> sink(file = "test.txt", type = "output")
R> summary(cars)
R> sink()
```

The last command turns off output to the file. Estimation output can also be saved in LaTeX format. Note that saving output in text form is different from saving a graph. On the latter, see the next section.

Saving Graphics Output

Quite often, after generating a graphic, we want to save it in its own file. For example, saving a graph in pdf format means it can easily be imported into a LaTeX document. When a graphics command is executed, its output is displayed by default in a separate graphics window that is opened on the screen. If we want the output into a file, we have to do the following.

```
R> pdf("filename.pdf")
R> plot(x,y)
R> dev.off()
```

The last command turns off output to the file. Many other graphics formats are available, such as png, jpeg, bmp, Postscript, etc.

Basic Descriptive Statistics

```
R> mean(myvariable)
R> median(myvariable)
R> var(myvariable)
R> sd(myvariable)
R> cov(myvariable1,myvariable2)
R> cor(myvariable1,myvariable2)
R> max(myvariable)
R> min(myvariable)
R> quantile(myvariable, 1:9/10)
R> kurtosis(myvariable)
R> skewness(myvariable)
```

The last two functions are not defined by default. They are available in the `moments` package, available from the *R* site and installable using the command `install.packages("moments")`. Note that the `kurtosis` command calculates the Pearson measure of kurtosis, which is **already normalized** by dividing by the square of the variance. This measure can be directly compared to the normalized kurtosis of the normal distribution (equal to three) to see whether a particular data series seems to come from a leptokurtic (kurtosis greater than three) distribution or not.

Use the following command for a basic set of summary statistics for all variables in a particular data set (dataframe).

```
R> summary(mydataframe)
```

The command `summary` can be applied to a single variable as well as an entire dataframe. As is typical in *R*, everything from a single variable to a dataframe is an **object** and many commands can be applied to objects of different types.

Functions Related to Probability Distributions

R contains very powerful functions to manipulate probability distributions. The `help(·)` command is very useful for learning what can be done in this regard. For a given probability distribution, *R* contains four basic functions. For example, with the univariate normal distribution we have

```
R> dnorm(x, mean=0, sd=1, log = FALSE)
R> pnorm(q, mean=0, sd=1, lower.tail=TRUE, log.p = FALSE)
```

```
R> qnorm(p, mean=0, sd=1, lower.tail=TRUE, log.p = FALSE)
R> rnorm(n, mean=0, sd=1)
```

The function `rnorm(·)` will generate n pseudo-random values drawn from the normal distribution with the specified mean and standard deviation. If the mean and standard deviation are not specified, the standard normal distribution is assumed.

The function that students in `ECO4272` will use most often is `pnorm(·)`. Once again, if the mean and standard deviation are not specified, the standard normal is assumed. For a standardized t statistic, this function can be used to calculate the area to the left of the statistic, so it is just the cumulative standard normal distribution function. For example, `pnorm(1.96)` gives approximately 0.975 and `pnorm(-1.96)` gives approximately 0.025.

The function `dnorm(·)` evaluates the normal density at a point or points. Students in `ECO4272` will not have much occasion to use this function.

The `qnorm(·)` function can be quite useful. Think of it as giving the value of the standardized t statistic for which the p-value is the argument of the function. For example, `qnorm(0.975)` gives approximately 1.96. If you need to know what is the (absolute) value of the t statistic you need to reject at 1% for a two-tailed test, use `qnorm(0.995)`, which gives approximately 2.576.

For other types of probability distributions, the arguments of the command are adapted to depend on the parameters of the distribution. For example, in the case of the uniform distribution, the following command generates n values drawn from the uniform distribution with the support $[a, b]$. (The only relevant parameters of the uniform distribution are its minimum and maximum values.)

```
R> runif(n, min=a, max=b)
```

For information on how to input the parameters of other distributions, use the `help(·)` and `args(·)` commands.

Executing a Script

By default, *R* starts an interactive section with input from the keyboard and output displayed in the terminal window. If you have a large number of commands to execute in a particular sequence, this can be inconvenient. It is possible to write these commands in a text file (script). The script can be executed with the command

```
R> source("myfile.R")
```

Any text appearing after "#" is ignored by the *R* interpreter, so comments and explanations can easily be included in an *R* script by prefacing them with the "#" character.

It is standard practice to use filenames with an "R" extension when saving scripts to files.

Quite often you will want to stop the execution of a script in order to examine some of the output produced by the script, such as a summary table of regression results or a plot. In order to stop the execution of a script until a key on the keyboard is pressed, use the command

```
R> readline(prompt="Press [enter] to continue")
```

Basic Graphs

```
R> x <- rnorm(100)
R> plot(x)
R> hist(x)
```

The first command generates 100 pseudo-random variables from a standardized normal distribution and places them in the vector x. The `plot(·)` command creates a basic plot, and the `hist(·)` command creates a basic histogram.

The most useful option of the `hist(·)` command is probably `freq=FALSE`, which restricts the total area under the histogram to be equal to one, which is often appropriate when plotting approximations to density functions. Use

```
R> hist(x, freq = FALSE)
```

The graphics capabilities of *R* are quite powerful. I would strongly advise consulting a reference book or article on *R* graphics to see what's available.

Linear Regression

```
R> model1 <- lm(Y   X1 + X2 + X3)
```

This is the basic command for estimating a linear regression model with the dependent variable Y and explanatory variables $X1$, $X2$ and $X3$. The default is to add a constant term. Here, the regression output is assigned to the object `model1`. Like most things in *R*, this is an **object**, which has certain default properties. A detailed summary of the regression output can be obtained with:

```
R> summary(model1)
```

By default, the standard errors that *R* calculates are **non-robust**. In order to generate **robust** standard errors, use:

```
R> coeftest(model1, vcov = vcovHC)
```

The `coeftest(·)` command is contained in the package `lmtest`, which must be installed and loaded into memory. Furthermore, the `vcov=vcovHC` option involves calculating the heteroscedasticity-consistent variance-covariance matrix which relies

on the package `sandwich`, which must be installed and loaded into memory. In turn, the `sandwich` package imports `zoo` and `stats`, which must be available (install them before using `sandwich`). The `stats` package should be part of almost any base *R* installation.

Since `model1` is an **object** defined in *R*, other commands can be applied to it. For example,

Some useful commands that operate on the object that results from estimating a linear model include the following. I assume here that `model1` is the object containing the regression results. Consult the help command for options.

Summary of regression results:	`R> summary(model1)`
Set of diagnostic plots:	`R> plot(model1)`
Model residuals:	`R> resid1 <- resids(model1)`
Model fitted values:	`R> fitted.values(model1)`
Coefficients table:	`R> coef(summary(model1))`
Variance-covariance matrix of coefficients:	`vcov(model1)`
Coefficient confidence intervals:	`confint(model1)`
Residual sum of squares:	`deviance(model1)`
Analysis of variance of results:	`anova(model1)`

Linear hypothesis testing can be conducted with the `linearHypothesis(·)` command, which works with both the non-robust and robust versions of the variance-covariance matrix od the coefficients.

I would strongly advise consulting the documentation on this command. The basic way of using it is relatively straightforward, but quite a few options and short cuts are available. The first argument of the command will typically be the name of a model estimated by linear regression using `lm(·)`. The second argument will typically be the R matrix and the third argument will be the r vector when the set of linear restrictions is written as $R\beta = r$, where β is the vector of model parameters. It is necessary to learn some basic commands for defining matrices and their elements. Here, I illustrate with an example.

```
R> linearHypothesis(unr,bigr,littler)
```

Here, `unr` is the name of the estimated (unrestricted) linear model, `bigr` is the R matrix, and `littler` is the r vector. To use the robust version of the variance-covariance matrix, use

```
R> linearHypothesis(unr,bigr,littler,white.adjust=HC)
```

Let's say the estimated model includes 4 explanatory variables in addition to the constant term, and we want to test the joint significance of the first two. In this case R is given by

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

and r is

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

We would set these up in *R* as follows.

```
R> bigr <- rbind(c(0,1,0,0,0),c(0,0,1,0,0))
R> littler <- rbind(0,0)
```

For more details use `help(rbind)`.

I strongly suggest reading through the commented scripts in the lecture notes on the simple regression model and the multiple regression model. These scripts illustrate how to load data, estimate a model, calculate robust standard errors, and save the output to a file.

This version: 05/04/2018