JBS Quantitative Research Methods Module MPO1

Michaelmas 2010
Thilo Klein
http://thiloklein.de

Computer Lab Session 1
General Introduction to R; Descriptive Statistics

Contents

A) Introduction

Note: Commands you need to execute in R are in bold type, with an arrow prefixed
R is an open source statistical software package for use on Linux, Mac and Windows machines. Precompiled binary distributions are available for download at http://cran.r-project.org. For Windows, use: **base** distribution, R version 2.11.1. **Manuals.** Several manuals are provided with the software. Useful manuals can be accessed online: http://cran.r-project.org/manuals.html The ***help*** command (to be explored today) should be your first port of call. http://search.r-project.org provides a powerful search for R functions and mailing-list archives on the web.

This session introduces the R interface (see part B**);**
the menu bar (see part C**);**
the tool bar (see part D**);**
and script files, which allow you to define a sequence of instructions and execute them sequentially.

Exercise 1. Find help; R Commander; install/load packages

There should be an R icon at the bottom of the Quicklaunch section of the desktop sidebar (blue letter "R"). Click on it.

B) The R interface

In *Linux*, the R distribution comes with no pre-installed graphical user interface (GUI). After successful installation, you can run the program directly in the console by typing the upper case letter 'R'. The *Windows* distribution comes with a Java GUI. R opens up with the console window.

To begin with, it probably is convenient to install the 'R Commander' package from one of several R mirrors. Type the following command in the command line:

- **install.packages("Rcmdr")**

Choose a mirror (e.g. Bristol or London) and click the 'OK' button. After the package has been unpacked, you can load it in the active workspace by typing:

- **library(Rcmdr)** or **require(Rcmdr)**

The R Commander has two windows. You can use the **Script** window to write commands and send them to the console by pressing the 'Submit' button or the hot key 'Ctrl+R'. The **Output** window reproduces the command and gives the corresponding output. Try:

- **1+1**

Also, notice the *menu-bar* at the top, starting with **File, Edit**, … **Help**.

Type in at the script window:

- **memory.size(2000)**

This is the memory that will now be available for you to work in R (the maximum memory that your datasets may take up).

Case-sensitive. Commands in R are always given in lowercase letters. R is case sensitive, so variable names should be typed exactly the way they were created. Each command has, associated with it, a help file, which can be consulted using the help command. For example: ➢ **?memory.size** ➢ **help.search("memory size")**

C) The menu-bar

There are **several ways** to give **commands** to R.
1) Via the menu bar at the top of the screen. 2) You can specify commands in the command line. 3) You may create a text document (a script) with a series of commands and get them to be executed sequentially as a program.
We will learn all the above. The last one is used most common because it is the most flexible. To start we use the *menu bar*.

Exercise 2. Load a dataset and save it with a different name

Data is best read into R using the 'Comma Separated Value' format (file extension .csv). To convert an Excel (.xls) file to .csv, simply save the dataset as .csv in the spreadsheet editor of your choice. To load a dataset in R from the menu bar:

➢ **Data/Import data/from text file…/**

Enter 'dataset2' as the name of the dataset, select 'Commas' as the field separator, click 'OK' and then select the dataset.

The drive which contains the data is JBSroot on 'PROTON (ntdomain)' which has the alias V. If you explore that drive, you will be able to find the folder: V:\Public\MP01\data

Open the CSV dataset: dataset.csv

Convince yourself that the data was successfully imported:

➢ **ls()**

Save the dataset in your own file space, from the menu bar:

➢ **Data/Active data set/Save active data set… [your network directory for this module's lab sessions] \dataset2.RData**

If you do not have an account in the JBS network, you may use a temporary folder that you create in C:\. Copy the files you create in this folder into your own USB by the end of the session.

If you are not a student from the Judge, provide me with your name and crsid for the use of the JBS IT Services, who will create an account for you.

You can load multiple datasets into R. **A dataset is in the form of a matrix** with variables arranged in columns and a different observation in each row. To refer to a variable in R you must indicate which dataset it belongs to by typing dataset**$**variable. Using the **attach**(dataset) and **detach**(dataset) options, you can simply refer to a variable in the attached dataset by typing its name.

Clear the R workspace:

- **rm(list=ls())**

and open your newly saved file.

- **Data/Load data set/dataset2.RData**

The dataset
This file called dataset.csv contains a firm level panel dataset of firms in the Indian software industry over the period 1993-2002. It was put together by a PhD student, who compiled the data from IT software and services directories, published annually by the trade association (NASSCOM), from 1993, except for years 1996 and 2000. Almost all IT software and service companies in India that employ more than 20 professionals is a member of Nasscom; the membership profile ranges from privately owned companies to public sector companies, from domestic software companies to multinationals operating in India. This data gives you a flavor of a reasonably complex data that can actually be used in research. There are some limitations to this dataset. In 1996 and 2000 no directory was published by Nasscom; thus data are missing for these two years. Nasscom compiles data as reported by the firms and self reporting is sometimes incomplete - there are a few firms in the database with no information on the number of employees, year of establishment, revenue, etc.

For more comments regarding loading data **see Appendix 1**.

Exercise 3. Observe the displays in different windows and create a script file

Observe **the main transformations** that occurred in the different windows.

In the **'Script window'**, all the commands that we executed are stored in sequence.
In the **'Output window'** we observe commands AND results written in different colours. In particular, note that the commands that have been executed are presented in red, prefixed with a '>'.

Note the structure of the commands given:

```
dataset2 <- read.table("C:/…/dataset.csv", header=TRUE, sep=",",
na.strings="NA", dec=".", strip.white=TRUE)
```

Each dataset loaded must be given a name (here for example: 'dataset2') as in R there can be multiple dataset in use at a time. The above command reads the table stored in the file C:/…/dataset.csv into the current workspace. The options specified tell R that (1) the first line of the file contains names of the variables, (2) commas are field separator characters, (3) "NA" is the string that is to be interpreted as Not Applicable (NA) value, (4) "." is the character used in the file for decimal points, and (5) leading and trailing white space should be stripped from character fields.

Note that when the last item in the output window is a red '>' all by itself, the program is ready to receive a fresh command.

As a program, R functions by manipulating variables. The **notion of a variable** in R corresponds directly to the notion of a variable in statistics. Thus each variable has a certain number of observations associated with it (frequently all variables will have the same number of observations). Each observation corresponds to what we think of as a data point.

Now create a **Script-file**:

- **File\ New Script** or: press **Ctrl+O**

Save the new script in your own file space to document this session:

- **File\ Save as…** or: press **Ctrl+S**

Copy all commands used so far in the R Commander into the script file.

Note the advantage of using script compared to mouse and menu: it allows you to document your results and helps others to replicate them. Your script could look like this example:

```
# ------------------------------------------------------------------------------
# MPO1 Quantitative Research Methods
# Lab Session 1
# ------------------------------------------------------------------------------

# --- Ex 1: find help, R Commander, install/load packages ----------------

?memory.size                    # help if command is known
help.search("memory size")      # help if command is not known

install.packages("Rcmdr")       # install package Rcmdr
library(Rcmdr)                  # load package

# --- Ex 2: Load a dataset and save it with a different name ----------------

dataset2 <- read.table("C:/…/dataset.csv", header=TRUE, sep=",", na.strings="NA",
dec=".", strip.white=TRUE)      # read dataset from .csv file

ls()                            # display active objects in workspace

save("dataset2", file="C:/…/dataset2.RData")  # save active object dataset2

rm(dataset2)                    # clear object dataset2 from workspace
rm(list=ls())                   # clear workspace

load("C:/…/dataset2.RData")     # load object dataset2
```

The new Script-file is used in the same way as the 'Script window' in the R Commander. Commands are sent to the console (equivalent to the Rcmdr's 'Output window') using:

- **Ctrl+R**

You can switch between console and script using

- **Ctrl+Tab**

Exercise 4. Provide description(s) for one or more variables

Some characteristics of all variables will be shown if you invoke:

- **str(dataset2)**

R presents variable features contained in the dataset.

> It is important to distinguish the way a variable is stored in the dataset is different from how it is displayed on the screen. Computations are performed on the basis of how the variable is stored, but on the screen you will see something a little different.
>
> **There are essentially two types of variables in R: factor and numeric.**
>
> ***Factor*** variables are normally used when the content of the variables are names (for example, name or region) but they may also be numbers. The important characteristic of this type of variable is that it is not possible to do calculations with factor variables. They are characterized by their size (number of letters).
>
> ***Numeric*** variables may be stored as different types according to the amount of space we wish to preserve for the variable. The default is double and it works fine most of the time. Try:
>
> - **is.double(dataset2$rev)**

Exercise 5: Inspect and modify the dataset

To obtain an overview of the objects in the workspace, type:

- **ls()**

There are various ways to inspect the object "dataset2": edit(), browse(), cbind(). To *directly* call the variables from dataset2 (i.e. rev instead of using dataset2$rev), use:

- **attach(dataset2)**

a) with command **edit**:

- **edit(dataset2)**

allows for the dataset to be opened and modified.

Change the value in the second row in the variable year to 30. Close the R editor window by clicking on the white cross in the red box (top right corner).

> Entering data manually can be done in the data **Editor**. However, this is not very handy since the editor is a very primitive spreadsheet. We can modify data cell by cell and but we cannot apply cut and paste to enter a matrix of values created in Excel.

You can select the variables that you want to work with

- **edit(dataset2$year)** (or after attach(dataset2) simply: **edit(year)**)

Close the window

b) with command **browse** [do this exercise in your own time]

- **browse(dataset2)**

This command does not allow any modification of the data.

c) with command **cbind**

- **cbind(comp_name, year, level, capital)**

Missing values are displayed as "NA", but stored as arbitrarily big numbers!!

Attempt the following.

- **cbind(comp_name, year, level, capital)[level > 3 & is.na(level)==FALSE]**

Note that had we not included 'is.na(level)==FALSE' it would have included the missing values, as they are higher than 3!! So this is an issue to be aware of, especially if we are computing averages or some other more complex calculations.

'&' means 'and', '|' means 'or', '!' means 'not'. In particular '!=' means 'not equal'.

Exercise 6: Create (generate) a new variable

6a) Calculate output per employee for all enterprises.

- **str(dataset2)**

Examine revenue and employees. **rev** is revenue; **softemp** is software employees; **othemp** is other employees.

Now create a new variable which will be total employees.

- **totemp <- softemp + othemp** (alternatively: **totemp = softemp + othemp**)

You can then determine productivity by:

- **productivity <- rev / totemp**

Note: to bring up the name of a variable in any command you can write the first letters of the variable and the tab button in your keyboard. This way R finishes writing the name for you (only works in console, not in the script file or the R Commander!). If the variable name is long or you're very lazy, this can be useful!

6b) generate a variable containing revenue squared.

- **revsq <- rev^2**

generate a variable containing logarithm of rev; and generate a variable containing logarithm of totemp:

- **lrev <- log(rev)**
- **ltotemp <- log(totemp)**

Exercise 7: Generate an indicator variable

Allocate 1 to all observations of a variable with some specific characteristic. We wish to create a variable that flags up a characteristic among observations of a variable. Variables of this type are called indicator variables.
For example we may wish to indicate data collected for the year 2002.

- **year2002 <- ifelse(year == 2002, 1, 0)**

Note that to denote the logical condition in the "ifelse" function, you need to type two equal signs together (==).

Inspect the variables employed, selecting only year and year2002.

- **cbind(year, year2002)**

Observe: we flagged the cases that we needed to. The rest of the observations are assigned a value of zero.

> Note: In programming '=' and '<-' are used to define: 'allocate the content or value on the right hand side to the variable on the left hand side'. So we could command: **x <- x + 1** meaning change x by adding 1 to the value is currently held.
>
> ==, instead is used in the logical sense. Example: **if(x==2)** means: 'for those observations with x = 2'.

Working with strings

You can create variables that have the text included. The example below is only to show you what you could do:

- **compnameis <- paste("company name is:", comp_name, sep=" ")**
- **compnameis[1:10]**

Exercise 8: Calculate summary statistics. Identify missing values and outliers. Note that the latter may be the result of mistakes. Calculate proportions of individuals presenting a certain characteristic.

To calculate summary statistics (minimum, maximum, quartiles, mean):

- **summary(dataset2)**

> Note that for **factor** variables, a frequency table is presented in the summary.
>
> A quick way to **identify missing values**: if you know a variable that does not have any missing values at all (for example: year has no missing values) you can check if other variables have different numbers of observations compared to this variable. All variables having fewer observations than 3682 have missing values.
>
> - **NA%in%level**
>
> **How many** observations are **missing**? The difference between 3682 and the number of observations reported under Obs. Alternatively:
>
> - **length(level[is.na(level)])**
>
> **Outliers** and errors can be identified by looking at 'min' and 'max'. Note the very

odd value of 30 for one year (this is the one we entered in Exercise 7).

Proportions are calculated under the mean column if the variable is an indicator. For example 35% of the companies in the dataset are public. Note that if you don't have the indicator you could create it in order to calculate the proportion.

Logical statements (assertions) are very useful for checking your data.

Exercise 9: Calculate additional summary statistics but only for a few variables, or a subset of observations.

To calculate summary statistics have a look at the R Commander menu

- **Statistics\Summaries**

or type:

- **sd(rev)**
- **var(rev)**
- **help.search("Skewness"); library(timeDate); skewness(rev)**

Now calculate the standard deviation for the variable rev, if level > 2.

- **sd(rev[level>2])**

Exercise 10. Analysis of the frequency of discrete variables

- **table(pub)**

What is the proportion of public enterprises?

Exercise 11. Analysis of the frequency of a combination of discrete variables (two-way tables).

- **table(pub, level)**

And now try

- **t<-table(pub, level)**
- **t/sum(t)**

Identify what the numbers mean.

Exercise 12. Analysis of a variable conditioned on a discrete variable **[do this exercise in your own time]**

Which category of firms has a higher average of employees per company (those exporting to the US, or those who do not sell in the US)? Lookup:

- **?by**
- **by(data=totemp, INDICES=us, FUN=summary)**
- **by(totemp, us, summary)**

This gives the minimum, maximum, mean and quartiles for totemp for firms that export to each region.

Ask for some more statistics:

- **by(totemp, india, summary); by(rev, india, summary)**

Are firms selling in India relatively efficient? Answer: they obtain more revenues with fewer employees, so yes.

Exercise 13. Graphs.

13a) Scatter plots

Select *totemp* for the x variable and *rev* for the y variables.

- **plot(rev ~ totemp)** (or: > **plot(totemp, rev)**)

Now do the same but for the variables in logs:

- **plot(lrev ~ ltotemp)**

Can you see why logarithms are sometimes more revealing?

13b) Histograms

- **hist(rev)**

Not very revealing as not many companies report revenues above 10,000.

- **hist(rev[rev<1000])**

Still not… but can observe the shape.

- **hist(rev[rev<500], breaks=50)**

Better. **breaks** = #, indicates how many intervals we wish to define.
Try this:

- **par(mfrow=c(2,1))**
- **hist(rev[rev<200 & uk==1], breaks=50); hist(rev[rev<200 & uk==0], breaks=50)**

the same graph for both companies exporting to the UK and companies not exporting to the UK.

Exercise 14. Drop variables or observations

At times, you will want to get rid of some (redundant) existing variable or observation. To do so, use the following command. For example, to drop the variable *dom* do:

- **dataset2$dom <- NULL**

Exercise 15. Generate linear transformations

Create a **linear transformation** of the variable **othemp** (number of non-software employees) so that: othemp_lt = 1 + othemp/100

- **othemp_lt <- 1 + othemp/100**

This creates a new other employees variable giving the firms' employees in hundreds and adding one non-software employee to each firm.

Remember if $E(X) = \mu_X \quad and \quad Var(X) = \sigma_X^2$, then $Y=a+X/b$ is distributed with

$$E(Y) = a + \frac{\mu_X}{b} \quad and \quad Var(Y) = \frac{\sigma_X^2}{b^2}.$$

You can, if you wish, use R as a (very powerful) calculator.

- **sum(othemp)**
- **mean(othemp)**
- **1 + mean(othemp)/100**

Create a **quadratic transformation** of the othemp variable

- **othemp_qt <- 1 + othemp/100 + 5 * othemp^2**

Plot the variables

- **plot(othemp_lt ~ othemp)**

A linear transformation produces a line.

- **plot(othemp_qt ~ othemp)**

A quadratic transformation produces a parabola. The orientations of the lines and parabolas will be dependent on the parameters used.

Exercise 16. Some properties of the sample variance

In order to calculate the correlation matrix between all variables of interest (using only pair-wise complete observations):

- **cor(othemp, softemp, use=" pairwise.complete.obs")**

And to generate the covariance matrix:

- **cov(othemp, softemp, use="pairwise.complete.obs")**

Calculate the variance of the sum of all employees in a typical company. You can do it in two ways:

a) generate a variable which is the sum and apply the variance. You have created the variable (totemp), so you need to apply the variance.

b) Apply the formula: Var (X+Y) = Var (X) + Var (Y) + 2 * covariance (X, Y)

For a):

- **myvar <- function(x){var(x, na.rm=T)}**
- **myvar(totemp)**

For b):

- **myvar(othemp) + myvar(softemp) + 2*cov(othemp, softemp, use="pairwise.complete.obs")**

The same applies to the mean:

- **mymean <- function(x){mean(x,na.rm=T)}**

- **mymean(othemp) + mymean(softemp)**

Calculate the correlation between othemp and softemp using the definition of this statistic (in your own time).

D) The tool-bar

Exercise 17. Explore the tool-bar

Now we inspect some of the icons below the menu bar of the R GUI.
We have the standard options to open a script (*.R file extensions), load a workspace (*.rda), and save the current workspace. Icons four through six are copy/paste functions. The seventh (stop sign) is to stop the program at once (useful when the results take too much time and new commands cannot be typed and executed).

Exercise 18. Explore the help function

- **Help/R functions (text)**

allows the user to obtain a detailed description of how to use any entered command. The command line alternative is:

- **?<function>**

where <function> is the name of the function (e.g. sd, var, mean) that you want help on.

If you do not know the precise name of the command you need to use:

- **Help/Search help**

This allows you to introduce some key words. The program then searches for these through relevant packages. The command line alternative to this is:

- **help.search("<searchterm>")**

where <searchterm> is the term or function you want to search for.

Exercise 19. T-tests (in your own time)

We use the normal distribution when: i) the population is known to follow a normal distribution with known population variance; or ii) when the shape of the population distribution is not known but the size of the sample is bigger than 30. A company wishes to assess their newly introduced measures to control claim costs.

Costs used to be 60 pounds. They wish to test if they have been reduced. They are willing to work with a 5% level of significance. They extract a sample of 26 observations:

45	49	62	40	43	61	48	53	67	63
78	64	48	54	51	56	63	69	58	51
58	59	56	57	38	76				

a) Test the hypothesis that the company benefited from their new procedures.

b) Using R, perform this test and calculate the confidence interval at 99% significance level. (This part to be done in Lab-Session 1).

Answer:

a) Ho: μ>=60; Ha: μ<60

The test is one-tailed. We decide the significance level at 5%. The test statistic is the t-student because the sample is small and we do not know the variance of the population.

The computed value for t is $t_{25} = \frac{\overline{X} - \mu}{s/\sqrt{n}} = \frac{56.42 - 60}{10.04/\sqrt{26}} = -1.818$. The critical value is -1.708, so we reject the null. Note that if we had decided to test it at 1% significance level, the result would have been the converse!

b) For this part, first highlight the data and copy them into your clipboard (Ctrl+C). Then read the data from the clipboard into R:

- **data <- scan("clipboard")**

Second, the confidence interval at 99% level:

- **mean(data) + c(-1,1)*sd(data)*qnorm(0.99)**

The test of hypothesis is performed using:

- **t.test(data, mu=60, alternative="less", conf.level=0.99)**

Observe the outcome: note that different results arise according to the type of test suggested (one-tail on the left, one-tail on the right, two-tail). Note that the confidence interval is built for a two-tail test.

Appendix 1

1. Loading datasets

Reading in a data file that is not in R format requires using the **read.csv** or the **read.xls** commands. Entering ASCII-tab delimited files (or comma-delimited files) is easy. For instance, we may want to use Excel to create a file. We save it as CSV (Save as type: comma separated values), say we name it as *filename*.csv and we read it into R by typing:

- **read.csv("[your network directory] /filename.txt", sep=",")**

2. R command general form

Knowing R essentially means knowing its commands and how to read the help files.

a) R functions have the following form:

function([*data object*] , [*options*])

To learn about a function such as the correlation function, you usually type help.search("correlation") or, given you already know the exact command: ?cor.

The R help documents first informs you which package the function belongs to, e.g. cor {stats}. Make sure you have that package installed. When in doubt, try to load it using: library(stats). If it does not load, first install it using: install.packages("stats").

The '[*data object*]' argument after 'command' means that after the command a variable list must be given. In the case of cor(), these are most likely two variable vectors, x and y, of same length, separated by a comma.

'[*options*]' add some instruction to complement the function that you're defining. They are presented after a comma. In the cor() example we may wish to specify which correlation coefficient is to be computed: Pearson? Spearman? Kendall? For Pearson's product-moment correlation coefficient, we would type:

- **cor(x, y, method="pearson")**

This is also the *default setting* in R. Another option we may wish to specify in an empirical analysis is how R treats missing values. The Spearman correlation coefficient will only work without missing values, so we add an additional option: use="complete obs".

b) Other useful commands

setwd() Every time it is opened, R has as default directory from where it loads and where it saves files (datasets, plots and others). To find out what the current working directory is type **getwd()**. This directory may be changed, using **setwd(),** to the folder defined for our current project in order not to be obliged to define the full path every time we operate with files.

xtable() from library **xtable** allows you to directly export your R output in LaTex. Try:

- **xtable(summary(lm))**

pdf() … dev.off from library **grDevices** to send R output to a pdf-file. Try:

- **pdf("myScattter.pdf"); plot(runif(100) ~ rnorm(100)); dev.off()**

3. Logical expressions

As it was mentioned these are used to restrict the orders given to R. A list of the operators follows:

|	or	/	divided by
&	and	^	raised to
==	equals	>	greater than
!=	not equal	>=	greater than or equal to
+	plus	<	less than
-	minus	<=	less than or equal to
*	multiplied by		

Note that logical equals need two equal signs to be defined. | and **&** are used when more than one condition is needed

Appendix 2

The R Reference Cards explain the main commands used here Check: http://cran.r-project.org/doc/contrib/Short-refcard.pdf. Other cards are available on the R project website http://cran.r-project.org. We will not cover some of them, but the help functionality may help you if you need them. Also as we've seen in Session 1, the menu bar of the R Commander may prove to be helpful to find more utilities.