A Brief Guide to **R** for Beginners in Econometrics

Mahmood Arai
Department of Economics, Stockholm University
First Version: 2002-11-05, This Version: 2006-02-09

1 Introduction

About these pages

I wrote these pages as a means of facilitating the use of **R** for beginners in Econometrics. I try to be *brief* and thus consider few items and few aspects of every item. Items are discussed by means of *examples*. Comments are very welcome!

Conventions in these pages

The symbol `#` is used for comments. Thus all text after `#` in a line is a comment. Lines following `>` are **R**-commands that can be run at the **R** prompt which as standard looks like `>`. R-codes including comments of codes as well as R-output are displayed indented as follows.

```
# Example
> myexample <- "example"
> myexample
[1] "example"
```

The words in ‘ ’ refer to verbatim names of files, functions etc. when it is necessary for clarity. The names ‘`mysomething`’ such as ‘`mydata`’, ‘`myobject`’ are used to refer to a general dataframe, object etc. Truncated output is indicated by ‘`...`’.

2 General

R is published under the GPL (GNU Public License) and exists for all major platforms.

R is described on the **R** Home page as follows:

> *”R is ‘GNU S’, a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc. Please consult the R project homepage for further information.”*

See **R** Home page for manuals and documentations. There are a number of books on **R**. Fox (2002), An R and S-plus Companion to Applied Regression, Dalgaard (2002), Introductory Statistics with R, and Venables and Ripley (2002), Modern Applied Statistics with S, are excellent books on R. The first two cover more basic material while the third book is more advanced. See also Racine and Hyndman (2002) Using R to Teach Econometrics. To cite **R** in publications you can refer to `www.r-project.org` and R Development Core Team (2003). See also CRAN Task View: Computational Econometrics.

Objects and files

R regards things as *objects*. A dataset, vector, matrix, results of a regression, a plot etc. are all objects. One or several objects can be saved in a file. A file containing **R**-data is not an object but a set of objects. The file `'DataWageMacro.rda'` on the top of `http://people.su.se/~ma/R_intro/` is such a file. This file includes two data sets. This means that you can have several datasets in a file. This is contrary to many other standard statistics packages such as Stata where a data file contains one data-table (data set) only.

Basically all commands you use are *functions*. A command: something(object), does something on an object. This means that you are going to write lots of parentheses. Check that they are there and check that they are in the right place.

3 First things

Installation

R exists for several platforms and can be downloaded from [**R**'s Home page].

Working with R

It is a good idea to create a directory for a project and start **R** from there. This makes it easy to save your work and find it in later sessions.

If you want R to start in a certain directory in *MS-Windows*, you have to specify the `'start in directory'` to be your working directory. This is done by changing the `'properties'` by clicking on the right button of the mouse while pointing at your R-icon, and then going to `'properties'`.

*Displaying the working directory within **R**:*

```
> getwd()

[1] "/home/ma/mydirectory"
```

Changing the working directory to an existing directory `'/home/ma/mydirectory'`

```
> setwd("/home/ma/mydirectory")
NULL
```

Reading codes from a file

Edit a series of R-codes in a file for example `'Mycodes.R'`. You can then run your codes from a this file as follows.

```
> source("Mycodes.R", echo=TRUE)
```

Saving the output

It is also easy to save the output in a file e.g. `'file.txt'`.

```
> sink("file.txt") # creates the file and redirects the output to that
> 1:10 # the integers 1 to 10
> sink() # stops redirecting the output
```

For saving graphics, see the section on graphics below.

Naming in R

You should avoid using underscore '_'. Do not name an object as 'my_object' or 'my-object' use instead 'my.object'. Notice that in **R** 'my.object' and 'My.object' are two different names.

You should not use names of variables in a data-frame as names of objects. If you do so, the object will shadow the variable with the same name in another object. The problem is then that when you call this variable you will get the object – the object shadows the variable.

To avoid this problem:

1- Do not give a name to an object that is identical to the name of a variable in your data frames.

2- If you are not able to follow this rule, refer to variables by referring to the variable and the dataset that includes the variable. For example the variable 'wage' in the data frame 'lnu' is called by:

```
> lnu$wage.
```

The problem of "shadowing" concerns **R** functions as well. Do not use object names that are the same as **R** functions. 'conflicts(detail=TRUE)' checks whether an object you have created conflicts with another object in the **R** packages and lists them. You should only care about those that are listed under '.GlobalEnv' – objects in your workspace. All objects listed under '.GlobalEnv' shadows objects in **R** packages and should be removed in order to be able to use the objects in the **R** packages.

The following example creates an object with the name 'T' that should be avoided, checks conflicts and resolve the conflict by removing 'T'.

```
>  T <- "time"
>  conflicts(detail=TRUE)
$.GlobalEnv
[1] "T"

$"package:methods"
[1] "body<-"

$"package:base"
[1] "body<-" "T"

>  rm(T)
>  conflicts(detail=TRUE)
$"package:methods"
[1] "body<-"

$"package:base"
[1] "body<-"
```

Extensions for files

It is a good practice to use the extension 'R' for your files including R-codes. A file 'lab1.R' is then a text-file including R-codes.

The extensions 'rda' or 'RData' are appropriate for work images (i.e files created by 'save()'). The file 'lab1.rda' is then a file including R-objects.

The default name for the saved work image is '.RData'. Be careful not to name a file as '.RData' when you use 'RData' as extension, since you will then overwrite the '.Rdata' file.

Saving and loading objects and images of working spaces

Download the data from http://people.su.se/~ma/R_intro/.

You can load the file 'DataWageMacro.rda' containing the data frames 'lnu' and 'macro'.

Loading an image 'DataWageMacro.rda' (a file in the current directory).

```
> load("DataWageMacro.rda")
> ls() # lists the objects
```

The following command saves the object 'lnu' in a file 'mydata.rda'.

```
> save(lnu, file="mydata.rda")
```

To save an image of the current work space that is automatically loaded when you start **R** in the same directory.

```
> save.image()
```

You can also save your working image by answering 'yes' when you quit and are asked 'Save workspace image? [y/n/c]:'.

In this way the image of your workspace is saved in the hidden file '.RData'.

You can save an image of the current workspace and give it a name 'myimage.rda'.

```
> save.image("myimage.rda")
```

Overall options

'options()' can be used to set a number of options that governs various aspects of computations and displaying results.

Here are some useful options.

```
> options(prompt="   R> ")  # changes the prompt to '   R> '.
> options(width=100)        # changes the default line width to 100 characters.

> options(scipen=3)         # From R version 1.8. This option
                            # tells R to display numbers
                            # in fixed format instead of
                            # in exponential form, for example
                            # 1446257064291 instead of 1.446257e+12 as
                            # the result of 'exp(28)'.
> options()                 # displays the options.
```

To change options and keep them for future sessions, a set of options can be specified in a file named as '.Rprofile' in the users home directory. These options are then run when **R** is started. See ?options for details.

Getting Help

```
> help.start()  # invokes the help pages.
> help(lm)      # help on 'lm', linear model.
> ?lm           # same as above.
```

4 Elementary commands

```
> summary(mydata)                    # Prints the simple statistics for ‘mydata’.
> hist(x, freq=TRUE)                 # Prints a histogram of the object ‘x’.
                                     # ‘freq=TRUE’ yields frequency
                                     # and ‘freq=FALSE’ yields probabilities.
> ls()                               # Lists all objects.
> ls.str()                           # Lists details of the objects in the memory.
> str(myobject)                      # Lists details of ‘myobject’.
> list.files()                       # Lists all files in the current directory.
> myobject                           # Prints simply the object.
> rm(myobject)                       # removes the object ‘myobject’.
> rm(list=ls())                      # removes all the objects in the working space.
> save.image("mywork.rda")           # saves current workspace named ‘mywork.rda’.
> save(myobject, file="myobject.rda")# saves the object ‘myobject’
                                     # in a file ‘myobject.rda’.
> load("mywork.rda")                 # loads "mywork.rda" into memory.
> load(".RData")                     # loads previously saved R-image saved by
                                     # answering yes to "Save workspace image? [y/n/c]:".
                                     # when quitting R. This is needed only when R
                                     # is started in one directory and the directory is
                                     # changed to another one in R.
> q()                                # Quits R.

# The output of a command can be directed in an object by using ‘<-’ ,
# an object is then assigned a value.

> x <- 1:2     # a vector named ‘x’ with a values 1 and 2.
               # <- is used to assign a value to an object (create an object).
> (x <- 1:2)   # Creates an object named ‘x’  and prints the contents of the object ‘x’.
```

5 Data management

Reading data in plain text format:

Data in columns

The data in this example are from a text file: `‘tmp.txt’`, containing the variable names in the first line (separated with a space) and the values of these variables (separated with a space) in the following lines.

```
The contents of the file ‘tmp.txt’
wage school public female
94      8       1      0
75      7       0      0
80      11      1      0
70      16      0      0
75      8       1      0
78      11      1      0
103     11      0      0
53      8       0      0
99      8       1      0
```

The following reads the contents of the file `‘tmp.txt’` and assigns it to an object named `‘dat’`.

```
> dat <- read.table("tmp.txt", header = TRUE)
```

The argument ‘header = TRUE’ indicates that the first line includes the names of the variables. The object ‘dat’ is a data-frame as it is called in **R**.

If the columns of the data in the file ‘tmp.txt’ were separated by ‘ , ’, the syntax would be:

```
> dat <- read.table("tmp.txt", header = TRUE, sep=",")
```

Data in rows

Time-series data often appear in a form where series are in rows. As an example I use data from the Swedish Consumer Surveys by the National Institute of Economic Research containing three series: consumer confidence index, a macro index and a micro index. First I saved the original data in a file in text-format. Before using the data as input in **R** I used a text editor and kept only the values for the first three series separated with spaces. The series are in rows. The values of the three series are listed in separate rows without variable names. Values are separated with spaces.

To read this file ‘macro.txt’, the following code puts the data in a matrix of 3 columns and 129 rows with help of ‘scan’ ‘matrix´ before defining a time-series object starting in 1993 with frequency 12 (monthly). The series are named as ‘cci’, ‘qmacro’ and ‘qmicro’. Notice that ‘matrix´ by default fills in data by columns.

```
> macro <-ts(matrix( scan("macro.txt"),129,3)    ,
               start=1993,frequency=12,
                  names=c("cci","qmacro", "qmicro"))

> macro  # The output is edited removing lines between April 1993 and June 2003.
         cci   qmacro   qmicro
Jan 1993    -38.3    -69.8    -21.3
Feb 1993    -28.0    -52.7    -16.0
Mar 1993    -31.6    -59.7    -17.0
Apr 1993    -28.0    -47.2    -17.8
...
...
...
Jun 2003      0.7    -26.3     10.1
Jul 2003      7.1    -15.5     12.1
Aug 2003      8.7    -13.3     13.7
Sep 2003     13.1     -7.7     15.1
```

Non-available and delimiters in tabular data

We have a file ‘data.txt´ with the following contents:

```
1 . 9
6 3 2
```

where the first observation on the second column (variable) is a missing value coded as ‘.’. To tell **R** that ‘.’ is a missing value, you use the argument: ‘na.strings="."’

```
 >  data1 <- read.table("data1.txt",na.strings="." )
 >  str(data1)
‘data.frame’:   2 obs. of  3 variables:
 $ V1: int  1 6
 $ V2: int  NA 3
 $ V3: int  9 2
```

Sometimes columns are separated by other separators than spaces. The separator might for example be ‘,’ in which case we have to use the argument ‘sep=","’.

Be aware that if the columns are separated by ‘,’ and there are spaces in some columns like the case below the ‘na.strings="."’ does not work. The NA is actually coded as two spaces, a point and two spaces, and should be indicated as: ‘na.strings=" . "’.

```
1,  .  ,9
6,  3  ,2
```

Sometimes missing value is simply ‘blank´ as follows.

```
1  9
6 3 2
```

Notice that there are two spaces between 1 and 9 in the first line implying that the value in the second column is blank. This is a missing value. Here it is important to specify ‘sep=" "´ along with ‘na.strings=""´ .

```
> data4 <- read.table("data4.txt",sep=" ", na.strings="" )
> str(data1)
‘data.frame’:   2 obs. of  3 variables:
$ V1: int  1 6
$ V2: int  NA 3
$ V3: int  9 2
```

Reading and writing data in other formats

Attach the library ‘foreign’ in order to read data in various standard packages data formats. Examples are SAS, SPSS, STATA, etc.

Download the data in Stata format: [‘wage.dta’]

```
# Reading data in Stata format:
> library(foreign)
> lnu <- read.dta("wage.dta") # reads the data and puts it in the object ‘lnu’
```

‘read.ssd()’ , ‘read.spss()’ etc. are other commands in the foreign package for reading data in SAS and SPSS format.

It is also easy to write data in a foreign format.

```
# Writing data in Stata format
> library(foreign)
# This writes the data object ‘lnu’ to the stata-format file ‘lnunew.dta’.
> write.dta(lnu,"lnunew.dta")
```

Examining the contents of a data-frame object

Read a dataset first.

Attaching the 'lnu' data allows you to access the contents of the dataset 'lnu' by referring to the variable names in the 'lnu'.

```
> attach(lnu)
```

If you have not attached the 'lnu' you can use 'lnu$female' to refer to the variable 'female' in the data frame 'lnu'.

A description of the contents of the data frame lnu.

```
>  str(lnu) # Description of the data structure

'data.frame':   2249 obs. of  5 variables:
$ wage  : int  81 77 63 84 110 151 59 109 159 71 ...
$ school: int  15 12 10 15 16 18 11 12 10 11 ...
$ exp   : int  17 10 18 16 13 15 19 20 21 20 ...
$ public: int  0 1 0 1 0 0 1 0 0 0 ...
$ female: int  1 1 1 1 0 0 1 0 1 0 ...
> summary(lnu) # A summary description of the data
      wage            school           exp            public           female
Min.   : 17.00   Min.   : 4.00   Min.   : 0.00   Min.   :0.0000   Min.   :0.0000
1st Qu.: 64.00   1st Qu.: 9.00   1st Qu.: 8.00   1st Qu.:0.0000   1st Qu.:0.0000
Median : 73.00   Median :11.00   Median :18.00   Median :0.0000   Median :0.0000
Mean   : 80.25   Mean   :11.57   Mean   :18.59   Mean   :0.4535   Mean   :0.4851
3rd Qu.: 88.00   3rd Qu.:13.00   3rd Qu.:27.00   3rd Qu.:1.0000   3rd Qu.:1.0000
Max.   :289.00   Max.   :24.00   Max.   :50.00   Max.   :1.0000   Max.   :1.0000
```

Creating and removing variables in a data frame

Usually you do not need to create variables that are simple transformations of the original variables as is the case in many statistical packages. You can use transformation directly in your computations and estimations. See further the section on "Estimating parameters ..." below.

```
# Creating a new variable in a data-frame
> lnu$logwage <- log(lnu$wage)

# Removing a variable in a data-frame
> lnu$logwage <- NULL
```

Choosing a subset of variables in a data frame

```
 # Read a 'subset'  of variables (wage,female) in lnu.
 > lnu.female <- subset(lnu, select=c(wage,female))

 # Putting together two objects (or variables) in a data frame.
 > attach(lnu)
 > lnu.female <- data.frame(wage,female)

 # Read all variables in lnu but female.
 > lnux <- subset(lnu, select=-female)

 # The following keeps all variables from wage to public as listed above
 > lnuxx <- subset(lnu, select=wage:public)
```

Choosing a subset of observations in a dataset

```
> attach(lnu)

# Deleting observations that include missing value in a variable
> lnu <- na.omit(lnu)

# Keeping observations for female only.
> fem.data <- subset(lnu, female==1)

# Keeping observations for female and public employees only.
> fem.public.data <- subset(lnu, female==1 & public==1)

# Choosing all observations where wage> 90
> highwage <- subset(lnu, wage > 90)
```

Replacing values of variables

We create a variable indicating whether the individual has university education or not.

```
# Copy the schooling variable.
> lnu$university <- lnu$school

# Replace university value with 0 if years of schooling is less than 13 years.
> lnu$university <- replace(lnu$university, lnu$university<13, 0)

# Replace university value with 1 if years of schooling is greater than 12 years

> lnu$university <- replace(lnu$university, lnu$university>12, 1)

# The variable 'lnu$university'  is now a dummy for university education.
# Another way of achieving the same result but not by replacing values as we usually mean
# is to use 'cut' as follows. See also The section on 'factors' below.
> lnu$university <- cut(lnu$school,c(3,12,max(lnu$school))) # The lowest vale of school is 4.
```

NOTICE: Remember to reattach the data set after recoding.

```
> attach(lnu)
> table(university)
university
   0    1
1516  733
```

The above example is just an illustration of how to replace values of a variable. To create a dummy we could simply proceed as follows:

```
>  university <- school > 12
>  table(university)
university
FALSE  TRUE
1516   733
```

However, we usually do not need to create dummies. We can compute on `'school > 12'` directly,

```
# An example
>  table(school > 12)
FALSE  TRUE
1516   733
```

Factors

A variable can be defined as a factor as follows. Redefining years of schooling as factor:

```
> lnu$school <- factor(lnu$school)

> contrasts(lnu$school)
   5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
4  0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
5  1 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
6  0 1 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
. . .

22 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0
23 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0
24 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1
```

The factor defined as above can for example be used in a regression model. The reference category is the level with the lowest value. The lowest value is 4 and as you can see the column for 4 is not included above. Changing the base category will remove another column instead of this column.

Resetting the base category to those with 12 years of schooling (the 9th level):

```
> contrasts(lnu$school) <- contr.treatment(levels(lnu$school),base=9)
> contrasts(lnu$school)
   4 5 6 7 8 9 10 11 13 14 15 16 17 18 19 20 21 22 23 24
4  1 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
5  0 1 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
6  0 0 1 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
. . .
12 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
. . .
22 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  1  0  0
23 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  1  0
24 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  1
```

The following redefines `'school'` as a numeric variable.

```
> lnu$school <- as.numeric(lnu$school)
```

Sometimes our variable has to be redefined to be used as a category variable with appropriate levels that corresponds to various intervals. We might wish to have schooling categories that corresponds to schooling up to 9 years, 10 to 12 years and above 12 years. This could be coded by using `'cut()'`. Here the lowest value of `'school'` is 4.

```
>  SchoolLevel <- cut(school, c(3,9,12, max(school)))
>  table(SchoolLevel)
SchoolLevel
    (3,9]  (9,12] (12,24]
      608     908     733
```

```
Labels can be set for each level. Consider the university variable created in the previous section.
     > # Setting labels
     > SchoolLevel <- factor(SchoolLevel, labels=c("basic","gymnasium","university"))
      > table(SchoolLevel)
      SchoolLevel
           basic  gymnasium university
             608        908        733
```

Non-available in data

When **R** computes 'sum' , 'mean' etc on an object containing 'NA', it returns 'NA'. To be able to apply these functions on observations where data exists, you should add the argument 'na.rm=TRUE'. Another alternative is to remove all lines of data containing 'NA' by 'na.omit'.

```
> a <- c(1,NA, 3,4)
> sum(a)
[1] NA
> sum(a,na.rm=TRUE)
[1] 8
# or
> sum(na.omit(a))
[1] 8
```

Time-series data

Here I give an example for creating lag values of a variable and adding it to a time-series data set.See also 'diff´ for computing differences.

Let us create a new time series data set, with the series in the data frame 'macro' adding lagged 'cci' (lagged by 1 month). The function 'ts.union' puts together the series keeping all observations while 'ts.intersect' would keep only the overlapping part of the series.

```
> macro2 <- ts.union(macro, l.cci = lag(macro[,1],-1) # the first lag of cci.
                                                      # 'macro[,1]' picks up the first column
                                                      # in the matrix 'macro')
> macro2 # The output is edited removing lines between March 1993 and August 2003.

         macro.cci macro.qmacro macro.qmicro l.cci
Jan 1993     -38.3        -69.8        -21.3    NA
Feb 1993     -28.0        -52.7        -16.0 -38.3
Mar 1993     -31.6        -59.7        -17.0 -28.0
...
...
...
Aug 2003       8.7        -13.3         13.7   7.1
Sep 2003      13.1         -7.7         15.1   8.7
Oct 2003        NA           NA           NA  13.1
```

Aggregating data by group

Let us create a simple dataset consisting of 3 variables V1, V2 and V3. V1 is the group identity and V2 and V3 are two numeric variables.

```
>  (df1 <- data.frame(V1=1:3, V2=1:9, V3=11:19))
  V1 V2 V3
1  1  1 11
2  2  2 12
3  3  3 13
4  1  4 14
5  2  5 15
6  3  6 16
7  1  7 17
8  2  8 18
9  3  9 19
```

By using the command 'aggregate' we can create a new data.frame consisting of group characteristics such as 'sum' , 'mean' etc. Here the function sum is applied to 'df1[,2:3]' that is the second and third columns of 'df1' by the group identity 'V1'.

```
>  (aggregate.sum.df1 <- aggregate(df1[,2:3],list(df1$V1),sum ) )
  Group.1 V2 V3
1       1 12 42
2       2 15 45
3       3 18 48
>
>  (aggregate.mean.df1 <-  aggregate(df1[,2:3],list(df1$V1),mean))
  Group.1 V2 V3
1       1  4 14
2       2  5 15
3       3  6 16
# The variable 'Group.1'  is a factor that identifies groups.
```

Using several data sets

We often need to use data from several datasets. In **R** it is not necessary to put these data together into a dataset as is the case in many statistical packages where only one data set is available at a time and all stored data are in the form of a table.

It is for example possible to run a regression using one variable from one data set and another variable from another dataset as long as these variables have the same length (same number of observations) and they are in the same order (the n:th observation in both variables correspond to the same unit).

```
# Consider the following two datasets:
> data1 <- data.frame(wage   = c(81,77,63,84,110,151,59,109,159,71),
                      female = c(1,1,1,1,0,0,1,0,1,0),
                      id     = c(1,3,5,6,7,8,9,10,11,12))

> data2 <- data.frame(experience = c(17,10,18,16,13,15,19,20,21,20),
                       id        = c(1,3,5,6,7,8,9,10,11,12))
```

We can use variables from both datasets without merging the datasets. Let us regress 'data1$wage' on 'data1$female' and 'data2$experience'.

```
> lm(log(data1$wage) ~ data1$female + data2$experience)
```

We can also put together variables from different data frames into a data frame and do our analysis on these data.

```
> (data3 <- data.frame(data1$wage,data1$female,data2$experience))

   data1.wage data1.female data2.experience
1          81            1               17
2          77            1               10
3          63            1               18
...
```

We can merge the datasets. If we have one common variable in both data sets, the data is merged according to that variable.

```
> (data4 <- merge(data1,data2))
    id wage female experience
1   1   81      1         17
2   3   77      1         10
3   5   63      1         18
...
```

This also works if the observations do not appear in the same order as defined by the `'id'`. Consider the following dataset that is not ordered by the variable `'id'`. Merging these data with `'data1'` yields a dataset that is identical to `'data4'`.

```
> data2biss <- data.frame(experience = c(10,13,15,16,17,18,19,20,20,21),
                          id         = c(3,7,8,6,1,5,9,10,12,11))

> data4biss <- merge(data1,data2biss)
> table(data4biss==data4) # check that 'data4biss' and 'data4' are identical.
TRUE
40
```

If the datasets have no common variables, the data sets must be ordered before matching. In this case you should not use `'merge'` (see ?merge for further information). Instead you can simply create a data frame by putting together the datasets in the existing order with help of `'data.frame'` or `'cbind'`. The data are then matched, observation by observation, in the existing order in the data sets. This is illustrated by the following example.

```
> data1.noid <- data.frame(wage   = c(81,77,63,84,110,151,59,109,159,71),
                           female = c(1,1,1,1,0,0,1,0,1,0))
> data2.noid <- data.frame(experience = c(17,10,18,16,13,15,19,20,21,20))
> cbind(data1.noid,data2.noid)
   wage female experience
1    81      1         17
2    77      1         10
3    63      1         18
...
```

If you want to add a number of observations at the end of a data set, you use `'rbind'`. The following example splits the `'data4'` in two parts and then puts them together by `'rbind'`.

```
> data.one.to.five <- data4[1:5,2:4]  # rows 1 to 5 and columns 2 to 4
> data.six.to.ten  <- data4[6:10,2:4] # rows 6 to 10 and columns 2 to 4
```

```
> data.one.to.five
  wage female experience
1   81      1         17
2   77      1         10
3   63      1         18
4   84      1         16
5  110      0         13
```

```
> data.six.to.ten
   wage female experience
6   151      0         15
7    59      1         19
8   109      0         20
9   159      1         21
10   71      0         20

> rbind(data.one.to.five,data.six.to.ten)
   wage female experience
1    81      1         17
2    77      1         10
3    63      1         18
4    84      1         16
5   110      0         13
6   151      0         15
7    59      1         19
8   109      0         20
9   159      1         21
10   71      0         20
```

6 Basic statistics

Statistic measures

Summary statistics for all variables in a data frame:

```
> summary(mydata)
```

Mean, Median, Standard deviation, Maximum, and Minimum of a variable:

```
> mean (myvariable)
> median (myvariable)
> sd (myvariable)
> max (myvariable)
> min (myvariable)
```

Average, Min, Max, Sum etc. by group.

The following computes the average by group creating a vector of the same length (See also 'tapply' in the section 'Tabulation'). Same length implies that for the group statistics is retained for all members of each group.

```
# Average wage for males and females:
> attach(lnu)
> ave(wage,female,FUN=mean)
```

The function `'mean'` can be substituted with `'min'`, `'max'`, `'length'` etc. yielding group-wise minimum, maximum, number of observations, etc.

Tabulation

Read a dataset first.

Cross Tabulation

```
> attach(lnu)
> table(female,public) # yields frequencies
      public
female   0   1
     0 815 343
     1 414 677

> table(female,public)/length(female) # yields relative frequencies
      public
female         0         1
     0 0.3623833 0.1525122
     1 0.1840818 0.3010227
```

Creating a table by category.

```
# Average wage for females and males.
> tapply(wage, female, mean)

       0        1
88.75302 71.23190
```

Using `'length'`, `'min'`, `'max'`, etc yields number of observations, minimum, maximum etc for males and females.

```
>  tapply(wage, female, length)

   0    1
1158 1091
```

The following example yields average wage for females and males in the public and private sector.

```
> tapply(wage, list(female,public), mean)

         0        1
0 89.52883 86.90962
1 71.54589 71.03988
```

7 Matrixes

A matrix of order (dimension) $M \times N$ is a set of $M \cdot N$ elements in M rows of length N elements and thus N columns. Each element of a matrix $\mathbf{A}$ can be denoted as a_{ij} where $i = 1, 2, 3, ..., M$ denotes the position in rows and $j = 1, 2, 3, ..., N$ denotes the position in the columns.

$$\mathbf{A}_{M\times N} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1N} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2N} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{M1} & a_{M2} & a_{M3} & \cdots & a_{MN} \end{bmatrix}$$

A scalar is thus a degenerate case of 1×1 matrix with a real number as the only element. A column (row) vector is a degenerate matrix with one column (row).

In R we define a matrix as follows (see ?matrix in R):

```
# a matrix with 3 rows and 4 columns with elements 1 to 12 filled by columns:
# Note that columns are indexed as: [,1] ... and rows are indexed as [1,] ...
> (A <- matrix(1:12,3,4))
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

# a matrix with 3 rows and 4 columns with elements 1,2,3, ..., 12 filled by rows:

> (A <- matrix(1:12,3,4,byrow=TRUE))
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12

# Dimension of a matrix

> dim(A)
[1] 3 4

# Number of rows
  > nrow(A)
 [1] 3

# or
  > dim(A)[1]
 [1] 3

# Number of columns
  > ncol(A)
 [1] 4

# or
 > dim(A)[2]
[1] 4
```

```
# Indexation
# The elements of a matrix can be extracted by using brackets after the matrix name
# and referring to rows and columns separated by a comma.

# Extracting the third row:

 > A[3,]
[1]  9 10 11 12

# Extracting the third column:
 > A[,3]
[1]  3  7 11

# Extracting the element in the third row and the third column:
 > A[3,3]
[1] 11


# Extracting the matrix except the first row:

 > A[-1,]
     [,1] [,2] [,3]
[1,]    5    7    8
[2,]    9   11   12

# Extracting the matrix except the second column:
 > A[,-2]
     [,1] [,2] [,3]
[1,]    1    3    4
[2,]    5    7    8
[3,]    9   11   12

# Consider the square matrix AA
 > (AA <- matrix(1:9,3,3))
      [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

# The following extract the Minor matrix M_22 that is the matrix left
# when the second row and the second column is eliminated. Minor matrixes
# are used for computation of determinants by Laplace expansion.
  > AA[-2,-2]
      [,1] [,2]
 [1,]    1    7
 [2,]    3    9


# Evaluating some condition on all elements of a matrix

# Elements greater than 3
> A>3
      [,1]  [,2]  [,3] [,4]
[1,] FALSE FALSE FALSE TRUE
[2,]  TRUE  TRUE  TRUE TRUE
[3,]  TRUE  TRUE  TRUE TRUE
```

```
 >  A==3
      [,1]  [,2]  [,3]  [,4]
[1,] FALSE FALSE  TRUE FALSE
[2,] FALSE FALSE FALSE FALSE
[3,] FALSE FALSE FALSE FALSE


# Listing the elements fulfilling some condition
# All elements greater than 6
 > A[A>6]
[1]  9 10  7 11  8 12
# Listing the number of elements fulfilling some condition
 > length( A[A>6])
[1] 6
```

A column vector

$$\mathbf{x}_{M\times 1} = \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots a_{M1} \end{bmatrix}$$

A row vector

$$\mathbf{x}_{1\times N} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1N} \end{bmatrix}$$

In R you can either define row and column vectors as special matrixes as above or use the vector notion of R which is general in the sense that it only specifies the number of elements and does not generally bind it to a row or column. This features yields flexibility for multiplying vectors and matrixes and also ambiguity. See further page 24 specially footnote 1 in *An Introduction to R* http://cran.r-project.org/doc/manuals/R-intro.pdf.

Scalar Matrix

A special type of matrix is a scalar matrix which is a square matrix with the same number of rows and columns, all off-diagonal elements equal to zero and the same element in all diagonal positions. The following exercise demonstrates some matrix facilities regarding the diagonals of matrixes.

```
 R> (A <- matrix(0,3,3))
[,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
[3,]    0    0    0
 R> # Creates the special case that is the identity matrix.
 R> diag(A) <- 1
 R> diag(A)
[1] 1 1 1
 R> diag(1,3,3)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1

# see also ?upper.tri and ?lower.tri in R.
```

Matrix operators

Addition and subtraction

Addition and subtraction can be applied on matrixes of the same dimensions or a scalar and a matrix.

$$\mathbf{A} + \mathbf{B} = \mathbf{C1}$$

$$\mathbf{A} - \mathbf{B} = \mathbf{D1}$$

```
 > (A <- matrix(1:12,3,4))
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
 > (B <- matrix(-1:-12,3,4))
     [,1] [,2] [,3] [,4]
 [1,]   -1   -4   -7  -10
 [2,]   -2   -5   -8  -11
 [3,]   -3   -6   -9  -12
  > (C1 <- A+B)
      [,1] [,2] [,3] [,4]
 [1,]    0    0    0    0
 [2,]    0    0    0    0
 [3,]    0    0    0    0
  > (D1 <- A-B)
      [,1] [,2] [,3] [,4]
 [1,]    2    8   14   20
 [2,]    4   10   16   22
 [3,]    6   12   18   24
```

Try also 3*A, 3*A-2*B.

Scalar multiplication

$$\lambda\mathbf{A} = [\lambda a_{ij}]$$

```
 > (A <- matrix((1:9)^2,3,3))
      [,1] [,2] [,3]
[1,]    1   16   49
[2,]    4   25   64
[3,]    9   36   81
 > (AmultipliedByTwo <- 2*A)
      [,1] [,2] [,3]
[1,]    2   32   98
[2,]    8   50  128
[3,]   18   72  162
 > (AmultipliedByTwo/2==A)
      [,1] [,2] [,3]
[1,] TRUE TRUE TRUE
[2,] TRUE TRUE TRUE
[3,] TRUE TRUE TRUE
```

Matrix multiplication

For multiplying matrixes R uses $'\% * \%'$ and this works only when the matrixes are conform.

A matrix 'B´ of dimension 2×3 cannot be multiplied with itself since for this to work given the rule of the matrix multiplication, the number of columns in the first element of the product must be equal to the number of rows in the second element. $B_{2\times3}\% * \%B_{2\times3}$ thus does not work.

```
# NOTICE that B*B yields element by element multiplication
  > B%*%B
 Error in B %*% B : non-conformable arguments
  > (E <- matrix(1:9,3,3)); (E2 <- matrix(1:9,3,3,byrow=TRUE))
      [,1] [,2] [,3]
 [1,]    1    4    7
 [2,]    2    5    8
 [3,]    3    6    9
      [,1] [,2] [,3]
 [1,]    1    2    3
 [2,]    4    5    6
 [3,]    7    8    9
  R> E2%*%E # Then E2%*%E works
      [,1] [,2] [,3]
 [1,]   14   32   50
 [2,]   32   77  122
 [3,]   50  122  194
  > matrix(
 + c(sum(E2[1,]*E[,1]), sum(E2[1,]*E[,2]), sum(E2[1,]*E[,3]),
 +   sum(E2[2,]*E[,1]), sum(E2[2,]*E[,2]), sum(E2[2,]*E[,3]),
 +   sum(E2[3,]*E[,1]), sum(E2[3,]*E[,2]), sum(E2[3,]*E[,3]))
 + ,3,3,byrow=TRUE)
      [,1] [,2] [,3]
 [1,]   14   32   50
 [2,]   32   77  122
 [3,]   50  122  194
 # Or another and more efficient way of obtaining crossproducts is:
  R>  crossprod(E) # equivalent to t(E)%*%E
      [,1] [,2] [,3]
 [1,]   14   32   50
 [2,]   32   77  122
 [3,]   50  122  194
```

Transpose of a matrix Interchanging the rows and columns of a matrix yields the transpose of a matrix.

```
 > (B <- matrix(1:6,2,3))
     [,1] [,2] [,3]
 [1,]    1    3    5
 [2,]    2    4    6
  > (Btranspose <- matrix(1:6,3,2,byrow=T))
      [,1] [,2]
 [1,]    1    2
 [2,]    3    4
 [3,]    5    6
  R> t(B)%*%B
      [,1] [,2] [,3]
 [1,]    5   11   17
 [2,]   11   25   39
 [3,]   17   39   61
```

Matrix inversion

The inverse of a square matrix $\mathbf{A}$ denoted as $\mathbf{A}^{-1}$ is defined as a matrix that when multiplied with $\mathbf{A}$ results in an Identity matrix (1's in the diagonal and 0's in all off-diagonal elements.)

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$$

```
> (FF <- matrix((1:9),3,3))
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
 > solve(FF)
Error in solve.default(FF) : Lapack routine dgesv: system is exactly
singular
 > (B <- matrix((1:9)^2,3,3))
     [,1] [,2] [,3]
         [1,]    1   16   49
         [2,]    4   25   64
[3,]    9   36   81
 > (Binverse <- solve(B))
           [,1]      [,2]       [,3]
[1,]  1.291667 -2.166667  0.9305556
[2,] -1.166667  1.666667 -0.6111111
[3,]  0.375000 -0.500000  0.1805556
 > B%*%Binverse
     [,1] [,2]          [,3]
[1,]    1    0  5.273559e-16
[2,]    0    1 -5.551115e-16
[3,]    0    0  1.000000e-00
 > round(B%*%Binverse)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

Determinants

How can vi evaluate whether a matrix has an inverse or not (is singular)? Matrixes with determinant equal to zero are singular i.e. has no inverse.

```
 > det(FF)        # |FF|
[1] 0

 > det(B)
[1] -216
```

See a textbook for determinants and associated concepts.

8 Estimating parameters of linear models using Least Squares

Some Examples

This section gives some examples of obtaining Ordinary Least Square estimates in a linear model.

The function for running a linear regression model is 'lm()'. In the following example the dependent variable is 'log(wage)' and the explanatory variables are 'school' and 'female'. An intercept is included by default. Notice that we do not have to specify the data since the data frame 'lnu' containing these variables is attached. The result of the regression is assigned to the object named 'reg.model'. This object includes a number of interesting regression results that can be extracted as illustrated further below after some examples for using 'lm'.

Read a dataset first.

```
> reg.model <- lm (log(wage)  ~   school + female)
> summary (reg.model)

Call:
lm(formula = log(wage) ~ school + female)

Residuals:
     Min       1Q   Median       3Q      Max
-1.46436 -0.15308 -0.01852  0.13542  1.10402

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  4.087730   0.022203  184.10   <2e-16 ***
school       0.029667   0.001783   16.64   <2e-16 ***
female      -0.191109   0.011066  -17.27   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2621 on 2246 degrees of freedom
Multiple R-Squared: 0.2101,     Adjusted R-squared: 0.2094
F-statistic: 298.8 on 2 and 2246 DF,  p-value: < 2.2e-16
```

Sometimes we wish to run the regression on a subset of our data.

```
> lm (log(wage)  ~   school + female, subset=wage>100)
```

Sometimes we need to use transformed values of the variables in the model. If you for example use the sum of two variables 'var1' and 'var2' as one of your explanatory variables. You should then use 'I(var1+var2)'. I() means Identify function.

```
> lm (log(wage) ~  school + female + exp + I(exp^2)) # exp^2 is 'exp' squared
```

Pairwise interaction of three variables 'female', 'school' and 'public'.

```
> lm (log(wage)  ~   female*school*public -school:public:female, data=lnu)
```

The above model could also be written as follows:

```
lm (log(wage) ~ female + school + public + female:school +
                      school:public + female:public, data= lnu)
```

A model with no intercept.

```
> reg.no.intercept <- lm (log(wage)  ~   female -1 )
```

A model with only an intercept.

```
> reg.only.intercept <- lm (log(wage)  ~   1 )
```

The following example runs 'lm' for females and males in the private and public sector separately as defined by the variables 'female' and 'public'. The data 'lnu' are split into four cells: males in private (0,0), females in private (1,0), males in public (0,1) and females in public (1,1). The obtained object 'by.reg' is a list and to display the 'summary()' of each element we use 'lapply' (list apply).

```
> by.reg <- by(lnu, list(female,public),function(x) lm(log(wage) ~ school, data=x))
> lapply(by.reg, summary)
> summary(by.reg[[2]]) # summary for the second element in the list i.e females in private
```

Notice that the function argument above can be any arbitrary function that can be applied to the subsets of the data. Try the following.

```
>  by(lnu, list(female), mean)
>  by(lnu, list(female), function(x) length(x$wage))
>  by.female.lnu <- by(lnu, list(female), function(x) x)
>  summary(lm(log(wage) ~ school, data=by.female.lnu[[1]]))
>  summary(lm(log(wage) ~ school, data=by.female.lnu[[2]]))
```

Extracting the model formula and results

The model formula

```
> (equation1 <- formula(reg.model))
log(wage) ~ school + female
```

The estimated coefficients

```
> (bb <- coefficients(reg.model)) # 'coefficients' can be abbreviated as 'coef'
(Intercept)      school      female
 4.08772967  0.02966711 -0.19110920
```

The standard errors

```
 > coef(summary(reg.model))[,2] # coef(summary(reg.model))[,1:2] yields
                                # both 'Estimate' and 'Std.Error'
 (Intercept)      school      female
 0.022203334 0.001782725 0.011066176
 # or the square root of diagonal elements of the variance-covariance matrix:
 > sqrt(diag(vcov(reg.model)))
```

The t-values

```
> coef(summary(reg.model))[,3] # try also coef(summary(reg.model))
(Intercept)      school      female
  184.10432    16.64144   -17.26967
```

Analogously you can extract other elements of the lm-object by:

'summary(reg.model)$r.squared'

'summary(reg.model)$adj.r.squared'

'summary(reg.model)$df'

'summary(reg.model)$sigma'

'summary(reg.model)$fstatistic'

'summary(reg.model)$residuals' or 'residuals(reg.model)'

The residual sum of squares.

```
> RSS <-  deviance(reg.model)
```

The Predicted values

```
> pred.reg.model <- predict(reg.model, preddata=mydata) #
```

Computation of the coefficients as solutions to the Normal equations

A model with intercept and one explanatory variable

$$\hat{y}_i = a + bx_i$$

$$b = \sum x_i y_i - (1/N) \sum y_i \sum x_i) / \sum x_i^2 - (1/N)(\sum x_i)^2$$

$$a = (1/N) \sum y_i - b(1/N) \sum x_i$$

Read a dataset first.

```
> attach(lnu)
> fm1 <- lm(log(wage) ~  school)
> b <- (sum(school*log(wage)) - (1/length(log(wage)))*(sum(log(wage))*sum(school)))/
+ (sum(school^2) - (1/length(log(wage)))*(sum(school)^2))

> a <- mean(log(wage)) - b*mean(school)
> fm1; b; a

Call:
lm(formula = log(wage) ~ school)

Coefficients:
(Intercept)       school
     3.98170      0.03082

[1] 0.03081884
[1]  3.981698
```

Computation of the coefficients as solutions to the Normal equations using matrixes

$$\hat{y} = bX$$

Minimizing the sum of squared errors $e'e$ yields the normal equations: $X'y = (X'X)b$

Solving normal equations:

$$b = (X'X)^{-1}X'y$$

Read a dataset first.

Create a matrix of explanatory variables `'X'` (including a vector of 1's for the intercept) and a matrix of the dependent variable `'y'`. See also the Section on matrixes.

```
> X <- cbind(1, female, school , exp ,I(exp^2))
> y <- log(wage)

> Xprimey <- t(X)%*%y   # or crossprod(X,y)
> XprimeX <- t(X)%*%X   # or crossprod(X)
```

Solve for $X'X^{-1}$ in $X'X(X'X)^{-1} = \mathbf{i}$ where $\mathbf{i}$ is a vector of all elements equal to 1.

```
> XprimeX.inv <- solve(XprimeX)
> b <- c(XprimeX.inv%*%Xprimey)
> b <- round(b,digits=5)
```

To check our results we estimate the model by `'lm'` and compare the coefficients.

```
> multi.model <- lm(log(wage) ~  female + school + exp + I(exp^2))
> cbind(round(coefficients(multi.model),digits=5),b)
                             b
(Intercept)  3.70692  3.70692
female      -0.17626 -0.17626
school       0.03889  0.03889
exp          0.02338  0.02338
I(exp^2)    -0.00034 -0.00034
```

White's heteroscedasticity corrected standard errors

The package `'car'` has predefined functions for computing the White's and other corrections for non-constant variance. See `'hccm'` for different weighting options. See also `'robcov'` in the package 'Design'.

```
# The White's correction
> library(car)
> sqrt(diag(hccm(lm(log(wage) ~ female +school),type="hc0")))
(Intercept)      female      school
0.022356311 0.010920889 0.001929391
```

The following function computes the White's weighted var-covariance matrix and returns the White's standard errors.

```
> white.robust <-  function(y, X){
        # Whites standard error
        # y: dependent variable N times 1
        # X: matrix (or data-frame) with cbind(1, x1,x2,...)
        # of intercept and K explanatory variable N times (K+1)
   sqrt(  diag(  (solve(t(X) %*% X)) #(X'X)-1
                  %*% t(X) # X'
                  %*%  diag((resid(lm(y ~ X -1)))^2 )
                  %*%  X  %*%  (solve(t(X)%*%X))            )  )  }
> y <- log(wage)
> X <- cbind(1,female,school)
> white.robust(y,X)
                female      school
0.022356311 0.010920889 0.001929391
```

9 Test-statistics

t-test

```
> r1<- rnorm(100,mean=0,sd=1)
> r2 <- rnorm(100,mean=0,sd=3)
> library(ctest) # is not necessary for R version 1.9.0 or later
> t.test(r1,r2)
```

F-test

$$\frac{(RSS_r - RSS_u)/q}{RSS_u/N - K - 1}$$

where RSS_r and RSS_u are Residual sum of squares for the restricted (restricting the coefficients to be zero) and the unrestricted model (allowing non-zero as well as zero coefficients), q is number of restrictions and $N - K - 1$ is the degree of freedom for K explanatory variables.

```
# The restricted Model

 > mod.r <- lm(log(wage) ~ 1)
 > mod.u <- lm(log(wage) ~ female + school)
 > ((deviance(mod.r) - deviance(mod.u))/2) /
   ( deviance(mod.u)/ (length(wage) - 2 - 1) )
[1] 298.7589
```

You can also do the following:

```
> anova(mod.r,mod.u)
Analysis of Variance Table

Model 1: log(wage) ~ 1
Model 2: log(wage) ~ female + school
  Res.Df     RSS   Df Sum of Sq      F    Pr(>F)
1   2248 195.338
2   2246 154.291    2    41.047 298.76 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Durbin Watson

'dwtest' in the package 'lmtest' and 'durbin.watson' in the package 'car' can be used. See also 'bgtest´ in the package 'lmtest´ for Breusch-Godfrey test for higher order serial correlation.

```
# See the section on time-series data for details of creating 'macro2'.
> macro2 <- ts.union(
                      cci    = macro[,1],# column 1, series 1
                      qmacro = macro[,2],# column 2, series 2
                      qmicro = macro[,3],# column 3, series 3
                      l.cci = lag(macro[,1],-1)) # the first lag of cci


# Fitting the model.
> summary(lm(cci ~ l.cci, data=macro2))
Call:
lm(formula = cci ~ l.cci, data = macro2)

Residuals:
     Min       1Q   Median       3Q      Max
-11.5991  -1.9701   0.0977   2.4416   7.6339

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.60123    0.32851    1.83   0.0696 .
l.cci        0.93085    0.02321   40.10   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.639 on 126 degrees of freedom
Multiple R-Squared: 0.9273,     Adjusted R-squared: 0.9268
F-statistic:  1608 on 1 and 126 DF,  p-value: < 2.2e-16

> mod1 <- summary(lm(cci ~ qmacro, data=macro2))
> library(lmtest)      # Attaching the library 'lmtest'.
> dwtest(mod1,data=macro2)       # The durbin watson test.

  Durbin-Watson test

data:  mod1
DW = 0.063, p-value = < 2.2e-16
alternative hypothesis: true autocorrelation is greater than 0
```

10 Graphics

Save graphs in postscript

```
> postscript("myfile.ps")
> hist(1:10)
> dev.off()
```

Save graphs in pdf

```
> pdf("myfile.pdf")
> hist(1:10)
> dev.off()
```

Several plots in one figure

```
> a <- 100:1000
```

Set the layout in 4 cells in 1 row and 4 columns, plot some functions and save it in a postscript file.

```
> pdf("myfile.pdf")
> layout(matrix(1:4,1,4))
> plot(a, log(a))
> plot(a, (log(a))^2)
> plot(a, sqrt(a))
> plot(a, a^3)
> dev.off()
```

Plotting the observations and the Regression line

Read a dataset first.
Plot the data and the regression line.

```
>  pdf("regression.pdf")
```

Plot ‘school’ against ‘log(wage)’

```
>  plot(school,log(wage))
```

The regression line:

```
>  abline (lm(log(wage) ~  school ))
```

A vertical red line at ‘school’ mean:

```
>  abline(v = mean(school), col="red")
```

A horizontal red line at mean log ‘wage’ :

```
>  abline(h = mean(log(wage)), col="red")
>  dev.off()
```

Plotting time series

Let us start by reading a time-series data set. For details see the section on
‘Reading data in plain text format’.

```
> load("DataWageMacro.rda")
> ts.plot(macro) # plots the series in a diagram
> plot.ts(macro) # plots the series separately
```

11 Writing functions

The syntax is: myfunction ¡- function(x, a, ...) { ...} You write a function by `'function'` and assign it to an object. The arguments for a function are the variables used in the operations as specified in the body of the function i.e. the codes within { }.

Once you have written a function and saved it, you can use this function to perform the same operation as specified in{ ...} by referring to your function (saved in the object) and replacing the arguments by the arguments relevant for the actual computation.

The following function computes the squared of mean of a variable. By defining the function `'ms'` we can write `'ms(x)'` instead of `'(mean(x))^2)'` every time we want to compute the square of mean for a variable 'x'.

```
> ms <- function(x) {(mean(x))^2}
> a <- 1:100
> ms(a)
```

The arguments of a function:

The arguments of a function don't have to be given when default values of the arguments are defined. The following function prints a string of text, for example `'Welcome'`

```
> welc <- function() {print("Welcome")}
> welc()
[1] "Welcome"
```

If no default arguments are specified, the arguments of the function must be supplied.

```
> myprog.no.default <- function(x) print(paste("I use", x ,"for statistical computation."))
> myprog.no.default()
Error in paste("I use", x, "for statistical computation.") :
        Argument "x" is missing, with no default
```

If a default value is specified, the default value is assumed when no arguments are supplied.

```
> myprog <- function(x="R") print(paste("I use", x ,"for statistical computation."))
> myprog()
[1] "I use R for statistical computation"
> myprog("R and sometimes something else")
[1] "I use R and sometimes something else for statistical computation"
```

A function for computing percentile 'p' of a variable 'x'.

There is of course a function in **R** called `'quantile()'` that does the same thing. If we have a vector of numbers we can compute the 10th percentile by writing:

```
> numbers <- 1:100
> quantile(numbers,0.1)
```

The function has thus two arguments: the variable `'x'` and the percentile `'p'`.

```
> pct <- function(x,p) {

  x <- sort(x)             # Sort the vector in ascending order
  # Is i_p = p * N /100 is an integer or not? N is the number of observations.
  test <- ceiling(p*length(x)/100)>(p*length(x)/100) # 'ceiling'= upper closest integer

  # If 'test'  is true, i_p is not an integer,we compute the value
  # for the upper closest observation ('ceiling').
  yes <- x[ceiling(p*length(x)/100)]

  # If 'test' is false, i_p is an integer, we compute the weighted average value for
  # the upper and lower closest observations. The weights are p/100 and 1-p/100.
  no <- (p/100)*x[p*length(x)/100]+(1-p/100)*x[p*length(x)/100+1]

  # The function 'ifelse'  returns the value of 'yes' if test is true
  # and the value of the object 'no' if 'test'  is not true.

  ifelse(test, yes, no)    }
```

Now we can use the `'pct'` function to compute percentiles.

```
> numbers <- 1:100
> pct(numbers,10)
```

Notice that the function `'pct'` was written for a vector and the function 'sort()' works only for vectors. To sort the data frame `'df1'` by the variable `'var1'` and assign it to a sorted data frame we can do as follows:

```
> 'sorted.df1 <- df1[order(df1$var1),]'
```

12 Miscellaneous hints

Income Distribution	see ineq.
Logit	`'glm(formula, family=binomial(link=logit)'. See '?glm' & '?family'.`
Negative binomial	`?negative.binomial or ?glm.nb'` in MASS, VR.
Poisson regression	`'glm(formula, family=poisson(link=log)'. See '?glm' & '?family'.`
Probit	`'glm(formula,family=binomial(link=probit)'. See '?glm' & '?family'.`
Simultaneous Equations	see sem, systemfit.
Time Series	see ?ts, tseries, urca and strucchange.
Tobit	`see '?tobin'` in survival.

Acknowledgements

I am grateful to Michael Lundholm , Lena Nekby and Achim Zeileis for helpful comments.

References

Dalgaard, P. (2002), *Introductory Statistics with R*, Springer.

Fox, J. (2002), *An R and S-plus Companion to Applied Regression*, SAGE publications.

R Development Core Team (2003), R: A language and environment for statistical computing, R Foundation for Statistical Computing, Vienna, Austria, url = http://www.R-project.org.

Racine J. and Hyndman R. (2002), Using R to Teach Econometrics, *Journal of Applied Econometrics*, Vol. 17, No. 2, 2002, pp. 149-174.

Venables, W.N. & B.D. Ripley (2002), *Modern Applied Statistics with S*, Springer.