Prof. Dr. Kurt Schmidheiny
Universität Basel

A Short Guide to R with RStudio

Version: 22-9-2013, 21:42

1 Introduction

This guide introduces the basic commands of the statistical software R using the graphical interface RStudio. Examples are shown using Windows, the adaption to other platforms such as Mac OS and Linux is straightforward. This guide only introduces the basic commands for data management and estimation. More commands (on panel data, limited dependent variables, monte carlo experiments, etc.) will be described in the respective handouts.

R commands are set in `Courier`.

2 Installing R and RStudio

R is a free open-source statistical software for various platforms such as Windows, Mac OS and Linux. R can be download from http://www.r-project.org/. RStudio is a convenient environment to run R. It is also a free, open-source, cross-platform software available at http://rstudio.org/. Install R first, and then RStudio.

3 The RStudio Environment

When you start RStudio you will see a window with several panes: the Console pane where you type in your R commands and where results will be reported, the Workspace pane where open datasets and other objects are listed, the Plots pane where graphs are drawn, the History pane with a list of all your previously issued commands. More panes will be described below.

4 Additions to R: Packages

Many researchers provide their own R programs through the R project webpage. Many packages are already preinstalled in the basic R instal-

lation. They can be directly activated from RStudio: go to the Packages pane and tick the corresponding package. Alternatively, they are activated by issuing a command in the Console. For example,

```
> library("foreign")
```

activates additional commands to import data from foreing, i.e. other, software such as Stata and SAS.

New packages can be installed by clicking on Install Packages in the Packages pane or with the command

```
> install.packages("foreign")
```

You will need to activate the new package before using it.

5 Where to get help

The online help in R describes all basic R commands as well as commands in active packages. You can directly search the online help from the Help pane in RStudio. Alternatively, the help for a known command, e.g. `load`, is displayed in the Help pane by issuing the command

```
> ?load
```

or

```
> help("load")
```

If you don't know the exact expression for the command, you can search the R documentation. For example,

```
> help.search("read")
```

lists in the top-left pane all packages and commands that contain the word "read".

6 Opening and Saving Data

R will look for data or save data in the *drive* and working *directory*. The working directory is specified depending on the operation system by

```
> setwd("/Users/sck/mypath")          # example Mac OS
> setwd("c:/my documents/mydir")      # example Windows
> setwd("c:\\my documents\\mydir")    # alternative Windows
```

Note: R cannot handle the usual windows path `"c:\my documents\mydir"`.

Open an existing R datafile (extension .RData or .rda)

```
> load("example.RData")
```

or

```
> load("c:/my documents/mydir/example.RData")
```

The data is stored in an object called dataframe in the R workspace and appears in the *Workspace* pane. In the usual case where we are using variables from only *one* dataframe, it is convenient to apply the command

```
> attach(mydata)
```

which allows to access variables without specifying the dataframe.

You can view the dataframe by double-clicking on its name, e.g. `mydata`, in the *Workspace* pane. Alternatively, you can list the first lines with the command

```
> head(mydata)
```

Save data in R format:

```
> save(mydata, file="example.RData")
```

where `mydata` is the name of the dataframe to be saved.

7 Importing Data

Data from common statistical packages like SAS, SPSS, STATA can be imported using the package `foreign`. For example, the following com-

mand imports data from STATA:

```
> library(foreign)
> read.dta("c:/mypath/example.dta", convert.factors=FALSE)
```

where the option `convert.factors=FALSE` asks R not to automatically convert all categorical variables into factor variables (see section 8).

Comma-separated files can be imported with the following command:

```
> mydata <- read.csv("c:/mypath/example.csv", sep=",")
```

where `example.csv` is the name of the comma-separated file and `mydata` is the name of the dataframe to be created in the R workspace. You can set the comma separator (“,” or ‘;”) used in the datafile with the option `sep`. If you have already opened a dataframe with the same name in R, the old one will be automatically replaced.

If the spreadsheet to be imported contains missing value codes, e.g. -999, rather than empty cells, the following commands converts these into the R code for missing values:

```
> mydata <- read.csv("example.csv", na.strings="-999")
```

Comma-separated files can be generated with e.g. Excel:

- Make sure that missing data values are coded as empty cells or as numeric values (e.g., 999 or -1). Do not use strings (e.g “-”, “N/A”) to represent missing data.
- Make sure that there are only dots (“.”) but no commas (“,”) or semicolons (“;”) in numbers. You may need to change this under Format menu, then select Cells... .
- Make sure that variable names are included only in the first row of your spreadsheet.
- Excel uses a different separator depending on the country settings: with English settings, a comma (“,”) is typically used, with other languages a semicolon (“;”) may be used.

Under the File menu, select Save As... . Then Save as type Comma Separated(.csv). The file will be saved with a .csv extension.

8 Data Manipulation

All subsequent commands are shown using the example dataset `mtcars`. See `?mtcars` for a description of the data. The example data is loaded and stored in dataframe `mtcars` as

```
> data(mtcars)
```

A new variable in the dataframe `mtcars` is created by e.g.

```
> mtcars$logmpg <- log(mtcars$mpg)
```

where `logmpg` is the name of the new variable and `log(mpg)` is an example of a mathematical function of existing variables. If the dataframe is attached, the command is simply

```
> mtcars$logmpg <- log(mpg)
```

There are different ways of creating dummy variables in R:

```
> mtcars$heavy = (mtcars$wt>=3)
```

creates a *logical* variable which is `TRUE` if cars are heavier than 3,000 lb and `FALSE` otherwise. Logical variables are treated like a *factor* variable in estimation commands. Factor variables are categorical variables that can be either numeric or string variables. Categorical variables can only take a limited number of ordered or unordered values. A numeric dummy variable is created by

```
> mtcars$heavy = as.numeric(mtcars$wt>=3)
```

or

```
> mtcars$heavy = ifelse(mtcars$wt>=3,1,0)
```

in both cases the resulting variable is `1` for heavy cars and `0` otherwise. Numeric variables can be turned into factor variables by

```
> mtcars$heavyf <- factor(mtcars$heavy)
```

which also takes the two values 0 and 1. String labels can be attached to the two values by

```
> mtcars$heavyf <- factor(mtcars$heavy, labels = c("light", "heavy"))
```

Note: Variables are automatically replaced if a new assignment is made.

Conditional assignments can be made by choosing a subset of observations with the `[]` operator. For example,

```
> mtcars$hptype <- NA
> mtcars$hptype[mtcars$hp<100] <- "economy"
> mtcars$hptype[mtcars$hp>=100 & mtcars$hp < 200] <- "regular"
> mtcars$hptype[mtcars$hp>=200] <- "sport"
> mtcars$hptype = factor(mtcars$hptype)
```

creates a factor variable with 3 types of horsepower.

Variables are deleted from the dataframe by

```
> mtcars$heavy <- NULL
```

9 Descriptive Statistics

Display univariate summary statistics (mean, median, ...) of numeric variables `mpg` in dataframe `mtcars` *varlist*:

```
> summary(mtcars$mpg)
```

Or if the dataframe `mtcars` is attached, simply

```
> attach(mtcars)
> summary(mpg)
```

Statistics based on a subset of observations are given by e.g.

```
> summary(subset(mpg, wt>=3))
```

or equivalently

```
> summary(mpg[wt>=3])
```

where only observations with weight `wt` heavier than 3000 lb are used.

Report the frequency counts of variable `am`:

```
> table(am, useNA="ifany")
```

where the option `useNA="ifany"` requests to report missing values.

Display the correlation and covariance matrix, respectively, between all numeric variables in the dataframe `mtcars`

```
> cor(mtcars, use="pairwise.complete.obs")
> cov(mtcars, use="pairwise.complete.obs")
```

The option `use="pairwise.complete.obs"` asks to use all pairs of observations without missing values. The correlation between the two variables `mpg` and `wt` is computed by

```
> cor(mpg, wt, use="complete")
```

A basic two-way table of frequency counts of two variables `gear` and `am` is produced by:

```
> table(gear, am)
```

where the option `useNA = "ifany"` allows to include missing values.

The package `gmodels` adds a command to produce a detailed two-way table with counts, row and column percentages along with Pearson's chi-square statistic:

```
> install.packages("gmodels")
> library(gmodels)
> CrossTable(gear, am, chisq=TRUE)
```

Perform a two-sample t-test of the hypothesis that `mpg` has the same mean for the two groups defined by the dummy variable `am`

```
> t.test(mpg~am)
```

where the two-samples are by default not assumed to have equal variances.

10 Graphs

Draw a scatter plot of the variable `mpg` (y-axis) against `wt` (x-axis):

```
> plot(wt, mpg)
```

Add a red regression line after drawing the scatter plot

```
> abline(lm(mpg~wt), col="red")
```

Draw a scatter with connected points, title and labels

```
> plot(wt, mpg, type="l", main="Mileage and weight", xlab="weight",
ylab="mile per gallon")
```

Draw a histogram of the variable `mpg`

```
> hist(mpg, freq=FALSE)
```

11 OLS Regression

The multiple linear regression model

$$mpg_i = \beta_0 + \beta_1 wt_i + \beta_2 disp_i + u_i$$

is estimated by OLS with the `lm` function. For example,

```
> fm <- lm(mpg~wt+disp, data=mtcars)
> summary(fm)
```

regresses the mileage of a car (`mpg`) on weight (`wt`) and displacement (`disp`). A constant is automatically added if not suppressed by

```
> lm(mpg~wt+disp-1, data=mtcars)
```

Estimation based on a subsample is performed as

```
> lm(mpg~wt+disp, data=mtcars, subset=(wt>3))
```

where only cars heavier than 3000 lb are considered. Tranformations of variables are directly included with the `I()` function

```
> lm(I(log(mpg))~wt+I(wt^2)+disp, data=mtcars)
```

The Eicker-Huber-White covariance is reported after estimation with

```
> library(sandwich)
```



```
> library(lmtest)
> coeftest(fm, vcov=sandwich)
```

F-tests for one or more restrictions are calculated with the command `waldtest` which also uses the two libraries `sandwich` and `lmtest`

```
> waldtest(fm, "wt", vcov=sandwich)
```

tests $H_0 : \beta_1 = 0$ against $H_A : \beta_1 \neq 0$ with Eicker-Huber-White, and

```
> waldtest(fm, .~.-wt-disp, vcov=sandwich)
```

tests $H_0 : \beta_1 = 0$ and $\beta_2 = 0$ against $H_A : \beta_1 \neq 0$ or $\beta_2 \neq 0$.

New variables with residuals and fitted values are generated by

```
> mtcars$uhat <- resid(fm)
> mtcars$mpghat <- fitted(fm)
```

R treats factor variables different from numeric variables in estimation commands. See the handout on "Functional Form in the linear Model" for more details.

12 Important Functions and Operators

Some Mathematical Expressions

`abs(x)`	returns the absolute value of x.
`exp(x)`	returns the exponential function of x.
`log(x)`	returns the natural logarithm of x if x>0.
`log10(x)`	returns the log base 10 of x if x>0.
`sqrt(x)`	returns the square root of x if x>=0.
`floor(x)`	largest integers not greater than x
`ceiling(x)`	smallest integers not less than x
`trunc(x)`	integer by truncating x toward 0
`round(x, digits=n)`	rounds x to the nearest n decimal places

Logical and Relational Operators

`>`	greater than	`<`	less than
`>=`	greater or equal	`<=`	smaller or equal
`==`	equal	`!=`	not equal
`&`	and	`|`	or
`!`	not		

Some Probability distributions and density functions

`pnorm(q)`	cumulative standard normal distribution
`dnorm(x)`	returns the standard normal density
`qnorm(p)`	inverse cumulative standard normal distribution