

## Part E. Beyond EpiData Analysis using R

### Part E: Beyond EpiData Analysis using R

Exercise 1: Introduction to R software: basics

Exercise 2: Introduction to R software: data bases and functions

Exercise 3: Multivariable analysis in R part 1: A logistic regression

Exercise 4: Multivariable analysis in R part 2: A Cox proportional hazard model

### Introductory note

Part E addresses how to conduct a multivariable analysis for two situations for which EpiData Analysis is not designed to be operational. With our philosophy that there should preference be given to outstanding freely available software, we chose to introduce the open-source package R rather than any of the proprietary software solution packages.

The first exercise introduces the bare-bone basics of R software with a few simple exercises to get a feel for its working. Recommended texts for self-study are (available on the course CD-ROM):

Aragón T J. Applied epidemiology using R. University of California, Berkeley School of Public Health, and the San Francisco Department of Public Health. Version 14 October 2013, available from <http://medepi.com/courses/applied-epi-using-r/>.

Lumley T and the R Core Development Team and UW Department of Biostatistics. R fundamentals and programming techniques. Version Birmingham, 2006-2-27/28, available from <http://faculty.washington.edu/tlumley/Rcourse/R-fundamentals.pdf>.

Venables W N, Smith D M and the R Core Team. An introduction to R. Notes on R: a programming environment for data analysis and graphics, Version 3.1.0 (2014-04-10), available from: <http://cran.r-project.org/doc/manuals/R-intro.pdf>.

The second exercise extends on the basics by introducing data bases and functions. The recommended text for self-study is (available on the course CD-ROM):

Myatt M. Open source solutions – R. Nordic School of Public Health and University of Tartu. Computer software in epidemiology / statistical practice in epidemiology. Version 16 May 2005, available from <http://www.brixtonhealth.com>.

The third exercise gives the first application, the use of logistic regression to determine predictors that include continuous variables and variables with more than two levels. The recommended text for self-study is (available in the course material):

Manning C. Logistic regression (with R). Version 4 November 2007, available from <http://nlp.stanford.edu/~manning/courses/ling289/logistic.pdf>.

The fourth exercise gives the second application, survival analysis where stratification with the Kaplan-Meier approach does no more suffice and adjustments are preferred by using a Cox

proportional hazard model. The recommended texts for self-study are (available in the course material):

Tableman M. Survival analysis using S/R. Unterlagen für den Weiterbildungsgang in angewandter Statistik an der ETH Zürich. [Note: despite the German subtitle, the document is entirely in English, except for the subtitle which refers to the course at the Swiss Federal Institute of Technology in Zurich]. Version August-September 2012. It is an excerpt from the book *Survival Analysis Using S: Analysis of Time-to-Event Data* by Mara Tableman and Jong Sung Kim, published by Chapman & Hall/CRC, Boca Raton, 2004.

Therneau T M. Package 'survival'. Version 2.37-7, July 2, 2014, available from <http://cran.r-project.org/web/packages/survival/index.html>.

### **Acknowledgments:**

We used the following data in the exercises:

In **Exercise 1**, we use an example given by Altman in:

Altman D G, Machin D, Bryant T N, Gardner M J. Statistics with confidence. 2nd edition. 2 ed. Bristol: BMJ Group; 2000.

In **Exercises 2, 3, and 4** we use data from:

Aung K J M, Van Deun A, Declercq E, Sarker M R, Das P K, Hossain M A, Rieder H L. Successful “9-month Bangladesh regimen” for multidrug-resistant tuberculosis among over 500 consecutive patients. *Int J Tuberc Lung Dis* 2014;18: in press.

We are grateful to Drs Aung, Declercq, and Van Deun to grant permission to use an excerpt of individual data from the original dataset.

Nguyen Binh Hoa, Ajay M V Kumar, and Zaw Myo Tun went through the first draft of the exercises and gave valuable input into the development of Part E.

## Exercise 1: Introduction to R software

At the end of this exercise you should be able to:

- Have a basic understanding of how R works and to expand on this basis in a self-learning process
- Be informed about resources on R software

R is a language and environment for statistical computing and graphing. It provides a suite of numerous packages for a virtually unlimited number of applications, including the most powerful tools that an epidemiologist might need. It has a large community of collaborators and contributors, it is entirely open to everybody who wants to use it and who wishes to contribute.



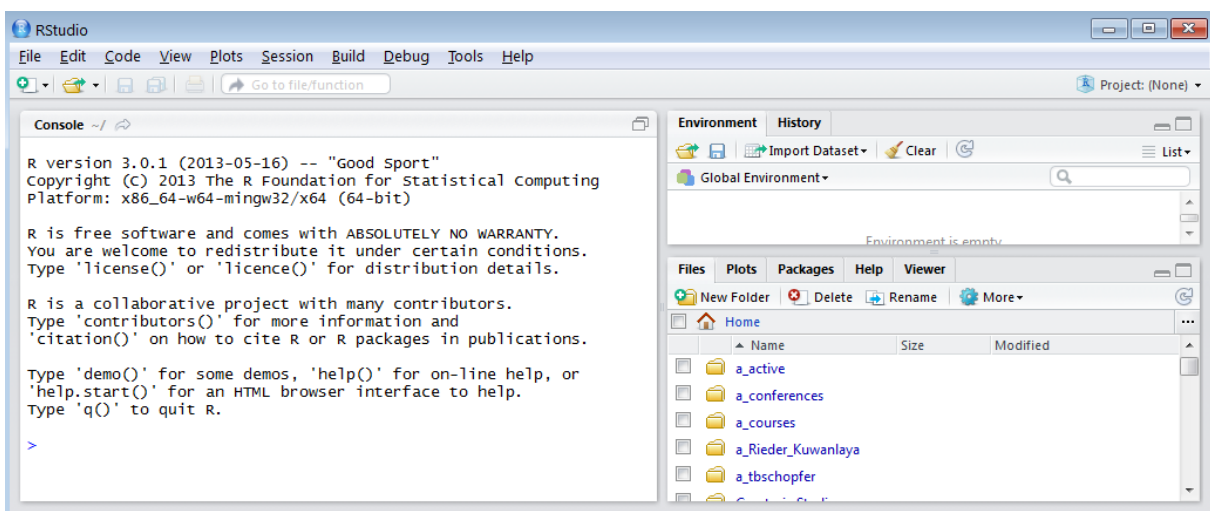
The starting point on the internet is the “The R Project for Statistical Computing” at its web site: <http://www.r-project.org/>. This is from where you access one if its mirror sites to download the software.



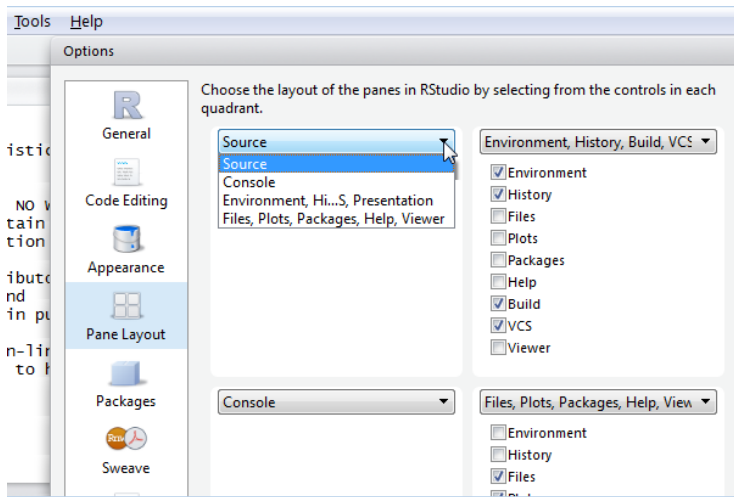
In tandem with R, we will also use RStudio, also open-source and free, a powerful and productive user interface for R. It can be found for download at its web site: <http://www.rstudio.com/>.

If your course comes on a CD-ROM, you find the two software packages on your CD-ROM, but as always with software, we strongly recommend that you obtain it from its original source on the internet.

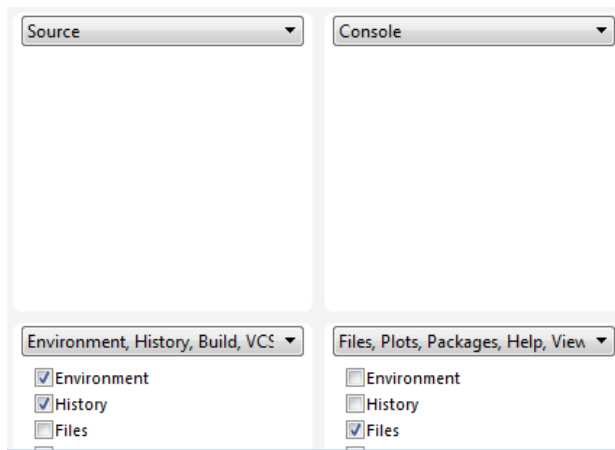
Install R on your computer and then install RStudio. If you open RStudio, you see that it is divided into a left-hand-sided panel, and a split right-handed panel:



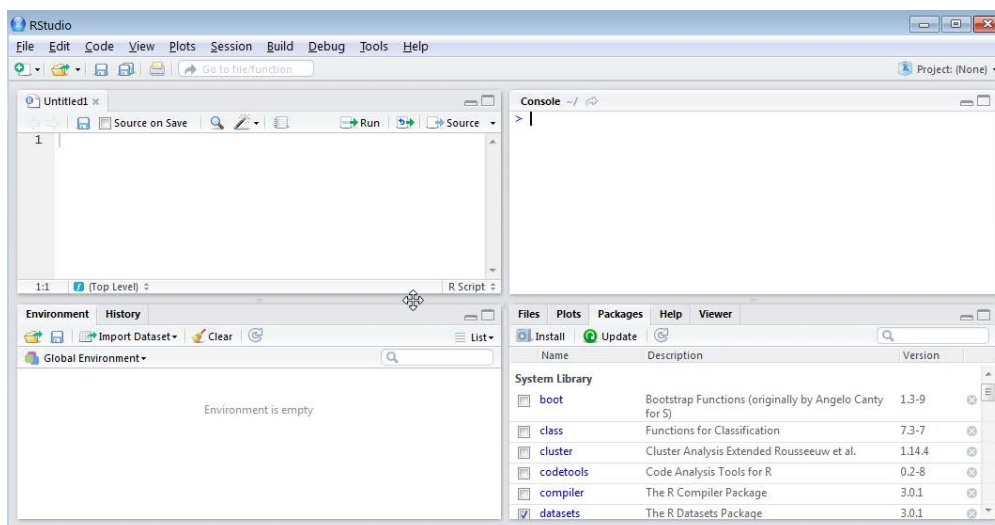
In Tools | Options we can adjust to the preferred layout:



We chose here (somewhat arbitrarily) to have the Source on the top left, the Console on the top right, the Environment, History, ... on the bottom left, and the Files, Plots, Packages, Help on the bottom left. and get altogether four quadrants:



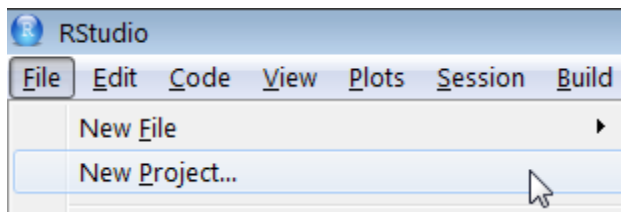
So that after these adjustments we get:



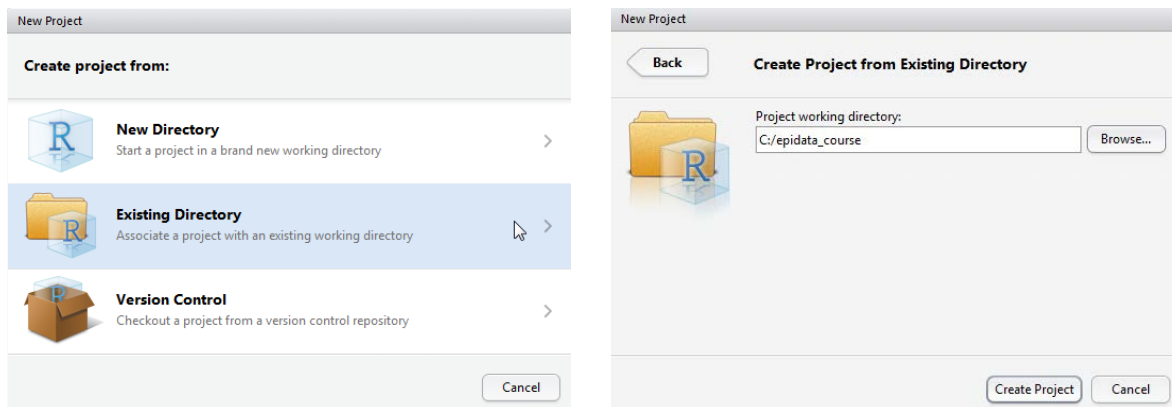
We cleared the screen of the Console by using the shortcut **CTRL+L**. To switch between the Console and the Source editor quadrants use **CTRL+1** to get to the Source editor, and **CTRL+2** to get to the Console.

Of course, you are free to arrange it at your liking, but for the time being in this course, we will refer to this arrangement.

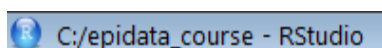
First we Create project from an Existing directory:



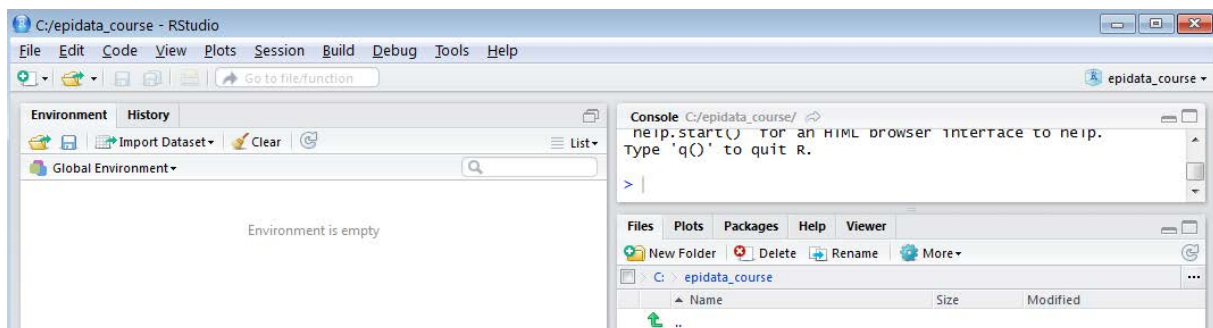
and browse for the EPIDATA\_COURSE project folder:



You note on the left top:



and note that the left side has been collapsed to show the environment only and on the right side we are in Files:



If you click on the toggle icon above Source:



You see the split screen again.

Note also that RStudio and R use forward slashes “C:/epidata\_course/” in the Console. RStudio actually created a sub-folder “.Rproj.user\” which is not public and by default hidden from view.

We are therefore all set to go. To introduce the workings of R we will create a two-by-two table as in a case-control study and calculate the odds ratio with a Woolf confidence interval.

### A case-control study example

Our reference text for statistical calculations like for EpiData Analysis is, whenever possible, the text by Altman and collaborators:

Altman D G, Machin D, Bryant T N, Gardner M J. Statistics with confidence. 2nd edition. Bristol: BMJ Group, 2000: pp 61-62.

The odds ratio is defined as the odds of exposure among cases divided by the odds of exposure among controls. Woolf’s logit method uses normal approximation to the distribution of the logarithm of the odds ratio in which the standard error is

$$SE(\log_e OR) = \text{SQRT}(1/a + 1/b + 1/c + 1/d)$$

To obtain the 95% confidence interval, we calculate the quantities:

$$95\% \text{ CI} = \log_e OR \pm 1.96 * SE(\log_e OR)$$

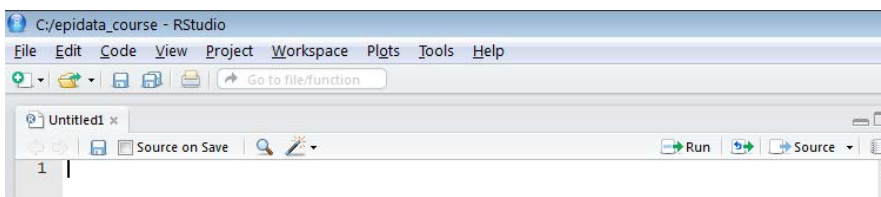
By exponentiating, we get the 95% confidence interval.

To enable verification of our numbers, we are going to use the same example as Altman uses in table 7.4 for the worked example:

ABO non-secretor state	Study group		Total
	Cases	Controls	
Yes	54	89	143
No	60	245	305
Total	114	334	448

We will be writing our commands into the upper left quadrant (Source editor). When we run it, it will show in the right upper quadrant (the Console). Objects that we create will be found in the left lower quadrant.

In the Source editor, you should now have the empty workspace like any text editor:



Type:

```
a <- 54
```

“a” is what we call an “object” to which we assign (using “<-”) the value 54. Some people say “Everything is an object in R”. There is (incomplete, though) truth to this. Anything and everything can be assigned to an object. Here we assign a number to an object “a”. We can name our objects whatever we want them to be called:

```
A  
a  
a.plus.b
```

but note importantly, they are case-sensitive: “A” is not the same as “a” in R. Another important note to make is that the value of an object can be overwritten without any warning.

You probably type the lesser sign “<”, followed by the minus sign “-”. RStudio allows you to use **ALT+** alone (the ALT key plus the minus sign) and you get the “assign” symbol combination including the spaces that we often like to make in R.

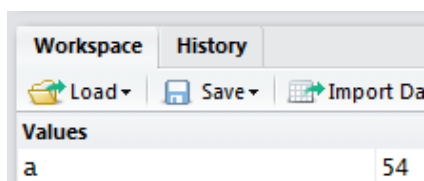
Nothing happened when you hit the carriage return key and this is how it is supposed to be in the Source editor. You only type here your stuff. To execute it, you need to tell it to do it. It is like writing commands in an EpiData Analysis program, and like in the latter you have:



a Run command: anywhere on a command line, it runs the current line, marking whatever you wish, it runs the marked section, nothing new here for the seasoned EpiData user. If we run this line, two things happen: in the right upper quadrant, the Console, you get:



This is confirmation that the command Run was executed. In the left lower quadrant, the Values, you get:



As we are going to use the Run command often, it is useful to know that the shortcut **CTRL+Enter** accomplishes the same thing as Run. **CTRL+Shift+Enter** runs the entire script.

All objects at your disposal will be listed here. In the Console, the value of object “a” is not visible. Some things you do will give you the result immediately, others will not, this assignment obviously belongs to the latter category. But type into the second line below the assignment:

```
365/7
```

and Run it. What you get now in your console is:

```
Console C:/epidata_course/ ↗
> a <- 54
> 365/7
[1] 52.14286
>
```

both the command you wrote and the result from the calculation. If you write into your working space:

a

and Run it you get:

```
Console C:/epidata_course/ ↗
> a <- 54
> 365/7
[1] 52.14286
> a
[1] 54
```

**CTRL+L** clears the Console.

Let's do the other assignments, writing each on a separate line:

```
b <- 89
c <- 60
d <- 245
```

then Run after marking. You should now have all four objects visible in the lower left quadrant:

Workspace		History
Load	Save	Import Dataset
Values		
a	54	
b	89	
c	60	
d	245	

While we have our four objects that make up the core of the 2-by-2 table, we would like to have the complete table including its marginals, and all assigned to one object.

To this end we will first create a **vector**, that is a one-dimensional object containing the values of a, b, and their sum. To keep with intuition, we will call this object "ab". Type:

```
ab <- c(a, b, a+b)
```

The "c" stands for "concatenate", that is putting a string consisting of sub-strings together. The elements of the vector are in parenthesis, separated by commas. They are, in the order in which we want to have them, our created two objects a and b, and the third element the sum of the two.

To see the result, type

```
ab
```

on a new line. Alternatively, you could also put a series of commands on one single line, separated by semi-colons, like in:

```
ab <- c(a, b, a+b); ab
```

and you should get:



```

Console C:/epidata_course/ ↵
> ab <- c(a, b, a+b); ab
[1] 54 89 143

```

Note, that in the left lower quadrant, you get now the object `ab` and a definition of what it is:

```

Values
a          54
ab        numeric[3]

```

Analogously, we create the second vector for the non-exposed:

```
cd <- c(c, d, c+d)
```

and get accordingly:

```

> cd <- c(c, d, c+d); cd
[1] 60 245 305

```

We have thus two vectors, one for the exposed and another for the non-exposed. Now we want them to be together in one single object that we will call `abcd`. The command to “bind” two rows together is:

```
abcd <- rbind(ab, cd); abcd
```

and we get:

```

> abcd <- rbind(ab, cd); abcd
  [,1] [,2] [,3]
ab  54  89 143
cd  60 245 305

```

The “`rbind`” is for “row binding”. Of course, if there is “`rbind`”, there must also be a “`cbind`” for column binding. Note also on the top `[,1]`, `[,2]` and `[,3]`: R labels (missing an actual label) things in a Cartesian way as `[row, col]`. R writes the indices to designate position in square brackets. As the column headers don’t have a row designation, it is thus the comma followed by the column number. What might be the internal designation for the location of the number 54? Yes, it is `[1,1]` for first row, first column, while 305 is in position `[2,3]`.

We have combined two **vectors** (one-dimensional objects) and the result is a **matrix** (a two-dimensional object). We could have created a matrix directly from the four objects `a`, `b`, `c`, and `d`. However, we have to pay attention to the correct the sequence:

```
abcd_m <- matrix(c(a, c, b, d, a+b, c+d), nrow=2, ncol=3); abcd_m
```

noting that the `matrix` function reads the data column-wise (!), we get:

```

> abcd_m <- matrix(c(a, c, b, d, a+b, c+d), nrow=2, ncol=3); abcd_m
  [,1] [,2] [,3]
[1,]  54  89 143
[2,]  60 245 305

```

We don’t need to write “`nrow=2, ncol=3`” to tell R that these 6 objects must be allocated to 2 rows and 3 columns, it suffices to stated:

```
abcd_m <- matrix(c(a, c, b, d, a+b, c+d), 2, 3); abcd_m
```

as R understands the first assignment (“2”) to be for the number of rows, and the second (“3”) for the number of columns. If we don’t pay attention to the default sequence row-column:

```
abcd_m <- matrix(c(a, c, b, d, a+b, c+d), 3, 2); abcd_m
```

we get a solution, but not the intended one:

```
> abcd_m <- matrix(c(a, c, b, d, a+b, c+d), 3, 2); abcd_m
      [,1] [,2]
[1,]   54  245
[2,]   60  143
[3,]   89  305
```

In contrast, if we specify:

```
abcd_m <- matrix(c(a, c, b, d, a+b, c+d), ncol=3, nrow=2); abcd_m
```

we will get what we intend:

```
> abcd_m <- matrix(c(a, c, b, d, a+b, c+d), ncol=3, nrow=2); abcd_m
      [,1] [,2] [,3]
[1,]   54   89  143
[2,]   60  245  305
```

Thus, as beginners, not yet fully conscious of default sequences, it is safer to be overly explicit.

To get back to the matrix: A matrix is a “two-dimensional table of like elements” (Aragón). Remove the object `abcd_m` as we are not going down this way for the time being:

```
rm(abcd_m)
```

where `rm` is the command to remove an object which is specified within the following paranthesis.

To complete the table, we need to make a row that is the sum of the first and the second row. We will call it `rtot` and it must be:

```
rtot <- abcd[1,] + abcd[2,]; rtot
```

because we are adding values of the respective column in row 1 `[1, ]` to the values in row 2 `[2, ]`. The result:

```
> rtot <- abcd[1,] + abcd[2,]; rtot
[1] 114 334 448
```

The last thing we have to do to get our complete table with all the marginals is row binding to an object that we will call `tab`:

```
tab <- rbind(abcd, rtot); tab
```

giving:

```
> tab <- rbind(abcd, rtot); tab
      [,1] [,2] [,3]
ab     54   89  143
cd     60  245  305
rtot   114  334  448
```

Before we lose our hard work, let's save what we typed so far in our editing window (the Source as we named the upper left quadrant initially) as “`e_ex01_or.r`” (by going through File | Save or by clicking the diskette icon). To our knowledge, R is not very particular what extension we give things, but we keep anyhow to what is quite widely common usage, and that is to give the extension `*.r`. You note how the file name replaces the Untitled:



While we have everything we need, we could make it a bit more appealing with appropriate column and row labeling which is done as:

```
colnames(tab) <- c("Case", "Control", "Total")
rownames(tab) <- c("Exp+", "Exp-", "Total")
```

and check to verify with:

```
tab
```

	Case	Control	Total
Exp+	54	89	143
Exp-	60	245	305
Total	114	334	448

“colnames” and “rownames” are the respective commands to give labels to columns and rows, while what follows in the parenthesis ( tab ) is the object to which the command applies.

We can also label the dimensions:

```
names(dimnames(tab)) <- c("Exposure", "Status"); tab
```

and get:

```
Exposure Status
```

Exposure	Case	Control	Total
Exp+	54	89	143
Exp-	60	245	305
Total	114	334	448

So far so good, but all we have is input, but what we really need are calculations. The odds ratio should be simple enough, we need to define the location by row and column of a, b, c, and d to then calculate  $(a/c) / (b/d)$ , thus:

```
or <- (tab[1,1]/tab[2,1])/(tab[1,2]/tab[2,2]); or
```

Note again here the brackets (not parentheses): whenever something needs to be indexed in R, brackets are used, here for the Cartesian positioning [ row, col ] but also if one wishes to describe the position of a record.

We specify the object tab and the position of its numeric elements to get the correct calculation, and the result should be:

```
2.477528
```

For the calculation of the 95% confidence interval (CI) we do this preferably in more than one step, first calculating the standard error se:

```
se <- sqrt(1/a+1/b+1/c+1/d)
```

We take here directly our four input objects as this calculation is independent of the position in the table. Next, for the lower and upper 95% CI:

```
or.ci.lower <- exp(log(or)-1.96*se)
or.ci.upper <- exp(log(or)+1.96*se)
```

Let's remove the objects that we don't need, the command to remove objects is rm:

```
rm(ab, cd, abcd, rtot, se)
```

Finally, let's get an output that summarizes it all together:

```
print(tab)
cat("\nOR:", round(or, digits=3), "\n95% CI:", round(or.ci.lower,
  digits=3), "to ", round(or.ci.upper, digits=3))
```

With the function “cat” we avoid quotes where we don't need them, and then take control by writing them where we need them:

```
x <- c("Ajay", "Hoa", "Zaw")
x
```

gives:

```
"Ajay" "Hoa" "Zaw"
```

but

```
cat(x)
```

gives:

```
Ajay Hoa Zaw
```

The function “\n” is equivalent to a hard carriage return.

The function round is followed by the object, then a comma, then the number of digits to which it is rounded, all in parentheses as is usual with functions.

Note that it is best if you write everything of the second item (starting with “cat”) on a single line. We should get:

```
      Status
Exposure Case Control Total
Exp+      54         89   143
Exp-      60        245   305
Total    114        334   448
```

```
OR: 2.478
```

```
95% CI: 1.595 to 3.849
```

Verifying by comparison with the results in Altman's shows that what we obtained is correct.

To make this more generically useful, we remove now all non-essential things, first of all the input objects a, b, c, and d. While not necessary (any object can be overwritten without warning), it is intuitively more appealing if we just keep the rest and then enter new values for a, b, c, and d and then run our “stripped” e\_ex01\_or.r. Thus, the stripped e\_ex01\_or.r may look like (note the hash “#” symbol at the beginning of the line correspondes to the asterisk “\*” in EpiData Analysis denoting a non-executed comment):

```
# Complete 2 by 2 table from
# a, b, c, d and calculate odds ratio with Wolf CI
# Assign below values for a, b, c, and d as in:
# a <- 100

ab <- c(a, b, a+b)
cd <- c(c, d, c+d)
abcd <- rbind(ab, cd)
```

```

rtot <- abcd[1,]+abcd[2,]
tab <- rbind(abcd, rtot)
colnames(tab) <- cbind("Case", "Control", "Total")
rownames(tab) <- rbind("Exp+", "Exp-", "Total")
names(dimnames(tab)) <- c("Exposure", "Status"); tab
or <- (tab[1,1]/tab[2,1])/(tab[1,2]/tab[2,2])
se <- 1.96*sqrt(1/a+1/b+1/c+1/d)
or.ci.lower <- exp(log(or)-se)
or.ci.upper <- exp(log(or)+se)

print(tab)
cat("\nOR:", round(or, digits=3), "\n95% CI:", round(or.ci.lower,
digits=3), round(or.ci.upper, digits=3), "\n")

```

You would now assign any value to your input objects a, b, c, and d, for instance directly in the console and then open this script and Run it, and there it will be.

To quit R, type:

```
q()
```

You will be prompted whether you wish to save the workspace image:

```
> q()
Save workspace image to C:/epidata_course/.RData? [y/n/c]:
```

to which you can safely answer with “n”: we have saved our scripts and do not need to save in addition the log file.

### Task

*In analogy to the calculation of the odds ratio, write an R script e\_ex01\_rr.r that calculates the relative risk from two proportions (thus not a ratio of incidence rates, but a ratio of two prevalence proportions) for a study used in Altman’s textbook. It has the isolation of Helicobacter pylori as the outcome and the history of an ulcer in the mother as the exposure. We use the following set-up of notations:*

Exposure	Outcome		Total
	Ill	Healthy	
Yes	A	B	A+B
No	C	D	C+D
Total	A+B	B+D	A+B+C+D

*The data provided by Altman in table 7.2 (page 59) are as follows:*

Mother with a history of ulcer	<i>H pylori</i> isolated		Total
	Yes	No	

Yes	6	16	22
No	112	729	841
Total	118	745	863

***The relative risk is calculated by:***

$$R = [A/(A+B)]/[C/(C+D)]$$

***The standard error of the log<sub>e</sub>R is:***

$$se = \text{SQRT}(1/A - 1/(A+B) + 1/C - 1/(C+D))$$

***The 95% CI is thus:***

$$95\%CI = \exp(\log_e R \pm 1.96 * se)$$