

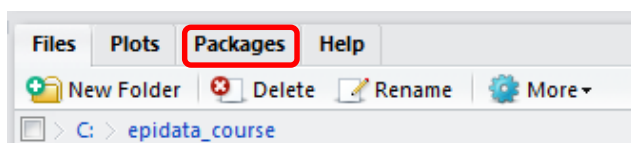
Exercise 2: Introduction to R software: data bases and functions

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

- Know how to import a data base from another format
- Create a function and apply it to an analytic problem

What we did in Exercise 1 is unusual in analysis: we entered aggregate data and analyzed them, while the usual currency of analysis is appropriate aggregation of individual observations. Commonly, datasets have been entered in other software as R is not a data entry tool. In the context of this course, data will come as an EpiData dataset with a *.REC and a *.CHK file. R accepts a multitude of data formats for import. In fact, a package has been developed that expands on the number of formats of datasets which can be imported into R, including for instance Epi Info / EpiData *.REC files and Stata *.dta files.

An important part of the versatility of R lies with the system of “packages”. What we have been using is the basic package, but if we want to utilize the capability of importing EpiData *.REC files or Stata *.dta files, the package “**Foreign**” should be added to our existing setup. In the lower right quadrant in RStudio, you see:



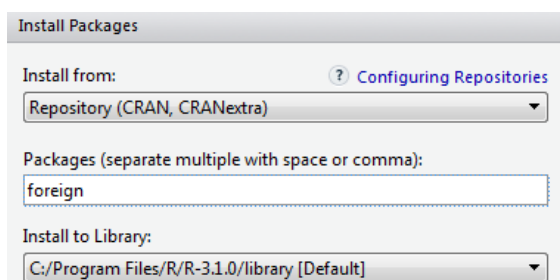
Click on Packages and you get a list of packages that are available with those loaded already ticked as for e.g. a basic statistical package:



The package we need is called foreign. You have to tick Install to get a menu:



You shouldn't need to do anything here except typing `foreign` into the line Packages and then Install as RStudio knows by definition where you have installed your copy of R. You need to be on the Internet though to get to the repository:



Alternatively, you can use the included Zip package in the course material, but then you need to change the Install from and get the downloaded zip file. But as always, to ensure obtaining

the most up-to-date version, get it whenever possible directly from the internet (<http://cran.r-project.org/web/packages/foreign/index.html>).

Having installed the package does not automatically put it at your disposal, you will have to call it. We start a new `*.r` script that we will name `e_ex02.r`.

In the required files on the course site you find the EpiData REC/CHK pair `e_ex02.rec` and `e_ex02.chk`. This is an abbreviated set of treatment results among patients with laboratory-confirmed multidrug resistance in Bangladesh.

The package `foreign` allows importing `*.REC` files with the command:

```
read.epiinfo("c:/epidata_course/e_ex02.rec")
```

However, this approach disregards the metadata (the `*.CHK` file) and only imports the field values from the `*.REC` file. We can write value labels in R, but we would first need to examine the `*.CHK` file and be careful about errors: tedious and error-prone. More efficient and safer is to use EpiData Entry 3.1 (or the EpiData Manager) to export the `e_ex02.*` file pair to a Stata `*.dta` file that contains both values and labels and then read the Stata file into R. Once you have exported the EpiData file pair to a Stata `e_ex02.dta` file, we are ready for import into R.

Being sticklers in labeling things, we start with a title, followed by actually invoking the now installed package **foreign**:

```
# Import an EpiData REC file

library(foreign)
e_ex02.dat <- read.dta("c:/epidata_course/e_ex02.dta")
```

Because we have made the “`epidata_course`” our project folder, it suffices to type:

```
e_ex02.dat <- read.dta("e_ex02.dta")
```

We read the Stata file by putting file name (and file path) in quotation marks inside a parenthesis (note the use of forward slashes) as in EpiData. We assign the imported file to an object that we will call `e_ex02.dat`. The `*.dat` is not required, it is just an object after all. We could call the object “`a`” if we wished to do so. The period separating the elements “`e_ex02`” and “`dat`” is not designating an extension, it is just one of the many ways R allows giving names to objects.

In a second step, we write the data to disk:

```
library(foreign)
e_ex02.dat <- read.dta("c:/epidata_course/e_ex02.dta")
write.table(e_ex02.dat, file="e_ex02.dat", row.names=TRUE)
```

Perhaps this is as good as any occasion to introduce the Help functions in R. If you know already “`write.table`”, then it is easy, type:

```
help(write.table)
```

and you get in the lower right-hand corner box quite extensive information (try it out!).

To see what the import did, type:

```
e_ex02.dat[1:5,]
```

This gives the first 5 records:

```

age      fq04      sex  totobstime  agequart      agemed  outcome07  outcome02
1  28 susceptible  Male      999 quartile 2  Below median  Cure  Success
2  45 susceptible  Male      482 quartile 4  Median or larger  Cure  Success
3  22 susceptible  Female    190 quartile 1  Below median  Default  Failure
4  26 susceptible  Female     72 quartile 2  Below median  Default  Failure
5  20 susceptible  Female   1000 quartile 1  Below median  Cure  Success

      pza02      kmy02      pth02      cxr02
1 PZA not known resistant KM not known resistant PTH not known resistant  Bilateral
2 PZA not known resistant KM not known resistant PTH not known resistant  Bilateral
3 PZA not known resistant KM not known resistant PTH not known resistant  Bilateral
4 PZA not known resistant KM not known resistant PTH resistant Not known bilateral
5 PZA not known resistant KM not known resistant PTH not known resistant  Bilateral

```

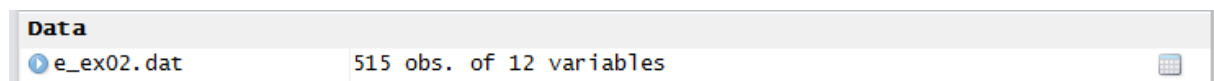
To see the variable names only, type:

```
# Get the list of variable names only
names(e_ex02.dat)
```

and get:

```
[1] "age"      "fq04"      "sex"      "totobstime" "agequart"  "agemed"   "outcome07"
[8] "outcome02" "pza02"     "kmy02"     "pth02"      "cxr02"
```

R has a special way to deal with missing information. It uses NA to denote any missing value, be it numeric or text. If NA is the assigned value, then a record in an analysis using the variable with such a value will be excluded. We will use different approaches to the analysis. In some analyses, we will include all 515 records – how many records we have, can be seen in the Environment lower quadrant of the left panel:



In another analysis, we will exclude missing observations. We will create three datasets:

- e_ex02_01.dat is the unaltered full set with missing values designated by NA, R's way to define missing values.
- e_ex02_02.dat is the dataset containing only records with information on initial fluoroquinolone resistance.
- e_ex02_03.dat is the subset of patients with initial ofloxacin resistance with either a bacteriologically unsuccessful outcome (failure or relapse) or a bacteriologically successful outcome (relapse-free cure or treatment completion).

The dataset

To get information on the structure of the dataset:

```
str(e_ex02.dat)
```

and we get (top only shown):

```
'data.frame': 515 obs. of 12 variables:
 $ age      : int  28 45 22 26 20 31 35 24 60 20 ...
 $ fq04     : Factor w/ 4 levels "Susceptible",...: 1 1 1 1 1 1 1 1 1 1 ...
 $ sex      : Factor w/ 2 levels "Male","Female": 1 1 2 2 2 1 2 1 1 1 ...
 $ tobstime: int  999 482 190 72 1000 1010 37 250 832 791 ...
 $ agequart : Factor w/ 4 levels "Quartile 1","Quartile 2",...: 2 4 1 2 1 3 3 2 4 1 ...
 $ agemed   : Factor w/ 2 levels "Below median",...: 1 2 1 1 1 2 2 1 2 1 ...
 $ outcome07 : Factor w/ 7 levels "Cure","Completion",...: 1 1 5 5 1 1 5 4 1 1 ...
 $ outcome02 : Factor w/ 2 levels "Success","Failure": 1 1 2 2 1 1 2 2 1 1 ...
 $ pza02    : Factor w/ 2 levels "PZA not known resistant",...: 1 1 1 1 1 1 1 1 1 1 ...
 $ kmy02    : Factor w/ 2 levels "KM not known resistant",...: 1 1 1 1 1 1 1 1 1 1 ...
 $ pth02    : Factor w/ 2 levels "PTH not known resistant",...: 1 1 1 2 1 1 1 1 1 1 ...
 $ cxr02    : Factor w/ 2 levels "Not known bilateral",...: 2 2 2 1 2 2 2 2 2 2 ...
```

The full dataset thus contains 515 records with 12 variables (rearranged):

AGE: Age in years as an integer.

AGEQUART: Age as a categorical variable in quartiles.

AGEMED: Age as a binary categorical variable (below the median, and median and larger).

SEX: Patient's sex as categorical variable (Female, Male).

OUTCOME07: 7-level treatment outcome (Cure, Completion, Failure, Death, Default, Relapse, Reinfection).

OUTCOME02: Outcome as a binary categorical variable, i.e. successful (relapse-free cure or completion) vs all other outcomes (Success, Failure).

TOTOBSTIME: Time of observation, an integer in days from treatment start until an event occurred (see later) or the observation time ended because of follow-up completion.

FQ04: Drug susceptibility test result for ofloxacin. If resistant, the minimum inhibitory concentration to gatifloxacin was determined. This categorical variable has four levels (Susceptible, Low-level resistance, High-level resistance, Missing).

PZA02: Drug susceptibility test result of molecular (*pncA*) for pyrazinamide as a categorical binary variable, resistant vs not known to be resistant (PZA not known resistant, PZA resistant).

KMY02: Drug susceptibility test result of phenotypic testing for kanamycin as a categorical binary variable, resistant vs not known to be resistant (KM not known resistant, KM resistant).

PTH02: Drug susceptibility test result of phenotypic testing for prothionamide as a categorical binary variable, resistant vs not known to be resistant (PTH not known resistant, PTH resistant).

CXR02: Radiographic disease extent as a categorical binary variable, bilateral vs not known to be bilateral (Not known bilateral, Bilateral).

The information for some variables was complete (age and sex), for some very few had missing information (fluoroquinolone drug susceptibility test result, missing assigned to a defined category). Some variables had quite a few missing (known *pncA* sequencing result for pyrazinamide), others were quite complete (like radiographic disease extent). One could argue to deal differently with the missing than what was done for this exercise in this simplified categorization as "not known resistant". In any case, we chose here certain simplifications that are unlikely to importantly bias the planned analysis in the wrong direction. For some key questions, a "purist's" approach is chosen to deal with missing values.

The easiest to create is the full set: the only manipulation required is an assignment of the only variable with a missing value, the variable FQ04 for ofloxacin drug susceptibility test result:

```
e_ex02.dat[e_ex02.dat$fq04=="Missing", "fluoroquinolone"] <- NA
```

Several new and important things are seen in this command line. Starting with the outermost right assignment NA, we note how R deals with missing data. In the data entry exercises with EpiData software we had taught (a bit dogmatically) that we wish to have a value (commonly designated as 9, 99, 9.99 etc) for all records with a missing value in a field. But this is by no means a universal standard. Rather more commonly such a field may remain empty or might have some value for missing or might be a mixture of both. Dealing correctly with missing values remains one of the most challenging tasks in any analysis. When we read the dataset and look at a frequency as follows:

```
table(e_ex02.dat$fq04)
```

we get (note the use of “table” rather than “tables” as in EpiData) before and after the NA assignment above:

Before:

```
Susceptible Low-level resistance High-level resistance Missing
      439              33              29              14
```

After:

```
Susceptible Low-level resistance High-level resistance Missing
      439              33              29              0
```

We could deal with the 14 missing as a category and leave it just as it is with these 4 category levels. In our, we will, however, pay special interest to this variable. As 14 of 515 is just 2.7% of all observations, there is no important bias if we later simply exclude these 14 cases. There are different ways to sub-setting, but we assign the R designation for missing, which is NA.

To the very left we have the name of the dataset e_ex02.dat. Inside the brackets we identify the variable within the dataset as in:

```
setname$varname
```

The “==” is new and differs what we learned in EpiData. The following are the logical operators used in R:

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x y	x OR y
x & y	x AND y
isTRUE(x)	test if X is TRUE

This is from the Quick-R website:

<http://www.statmethods.net/>

an extraordinarily useful website to get – as it says – quick answers to questions on R, highly recommendable! R (and Stata) uses the double == sign for equality rather than the single equal sign because the latter (=) can be used as “assign” instead of the <- we now got used to (and will stick to).

Thus, to get back:

```
e_ex02.dat$fq04=="Missing"
```

This identifies the variable `fq04` within the dataset `e_ex02.dat` and identifies records in which the category value is “Missing” (note that the value is not the “value” from the *.REC file, but the value is the label obtained from the Stata file for easy identification). After the comma, we state that we retain the variable name `fq04`:

```
e_ex02.dat[e_ex02.dat$fq04=="Missing", "fq04"] <- NA
```

we could check whether we have now records with NAs with the important command:

```
na.is(x)
```

which is used to find out which elements of `x` are recorded as missing (NA), i.e. here:

```
is.na(e_ex02.dat$fq04)
```

and we get (an excerpt from the list of all 515 records):

```
[289] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE TRUE
[305] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[321] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE
```

To assign this revised set to the new object with the desired name and to ensure that it is a “data frame”, we write:

```
e_ex02_01.dat <- data.frame(e_ex02.dat)
```

What is called a “data frame” in R is just what we EpiData users call a dataset. “It is a list of vectors and / or factors of the same length that are related “across” such that data in the same position come from the same experimental unit” (formulation copied from Peter Dalgaard, Introductory statistics with R, 2008). The distinction is made because R can deal with other types of data collection.

Sub-setting

The next dataset is trickier to create. An excellent source on the internet to find quick answers is Quick-R at <http://www.statmethods.net/index.html>. If we enter “subsets” into the Search box, we see among several option, option Nr 1:

```
1. Subsetting Data (36%)
   Description: Keeping or Deleting Variables, Observations, Random Samples ...
   URL: http://www.statmethods.net/management/subset.html
```

Clicking on the link we get all we need to know for starters. To select observations, the full explicit is:

```
newdata <- mydata[which(mydata$gender=='F' & mydata$age > 65),]
```

There is a short-cut allowed:

```
newdata <- mydata[which(gender=='F' & age > 65),]
```

Note here the single quotes to enclose text, but we can also use double quotes. In any case, we need to know the criteria. We mentioned above that we want to do a sub-analysis on patients with initial ofloxacin resistance. If we just write some table command, we might have the problem of ambiguity on which of the currently existing two datasets the command needs to be executed.

Our dataset to create is to be named “e_ex02_02.dat”. It will only contain records without missing data for the variable fq04. We are thus making a selection in EpiData language or sub-setting in R language. If we start from the originally read dataset, we write:

```
e_ex02_02.dat <- e_ex02.dat[which(e_ex02.dat$fq04 != "Missing"), ]
```

Alternatively, and perhaps simpler, we can use the subset function:

```
e_ex02_02.dat <- subset(e_ex02.dat, fq04 != "Missing")
```

Note (and see above) the “!=” denoting “unequal”.

To make the even more complex third dataset, the newcomer might best first stick to the fully explicit. We thus write three lines, making intermediary datasets, each line being self-explanatory:

We start with the dataset e_ex02_02.dat which has 501 records. Excluding 26 deaths defined in the variable outcome07, we get 475 observations in the new dataset e_ex02_02_03a.dat:

```
e_ex02_03a.dat <- subset(e_ex02_02.dat, outcome07 != "Death")
```

Excluding then 40 defaulters defined in the variable outcome07, we get 435 observations in the new dataset e_ex02_02_03b.dat:

```
e_ex02_03b.dat <- subset(e_ex02_03a.dat, outcome07 != "Default")
```

Excluding finally 382 patients with initial fluoroquinolone susceptibility defined in the variable fq04, we get 53 observations in the final desired dataset e_ex02_02_03.dat:

```
e_ex02_03.dat <- subset(e_ex02_03b.dat, fq04 != "Susceptible")
```

Of course, this can all be written into a single command line:

```
e_ex02_03.dat <- subset(e_ex02_02.dat, outcome07 != "Death" & outcome07 !=  
  "Default" & fq04 != "Susceptible")
```

and one gets the same result of 53 observations.

Finally, to save the three datasets to disk, we do as above:

```
write.table(e_ex02_01.dat, file="e_ex02_01.dat", row.names=TRUE)  
write.table(e_ex02_02.dat, file="e_ex02_02.dat", row.names=TRUE)  
write.table(e_ex02_03.dat, file="e_ex02_03.dat", row.names=TRUE)
```

A generically applicable function for the analysis of a 2-by-2 table

If we had a variable `fq02` to denote a binary outcome for initial fluoroquinolone (FQ) resistance (aggregating low and high resistance) and excluding those with missing fluoroquinolone test result, an analysis by the binary outcome in EpiData Analysis is as follows:

```
select fq02<>9 // exclude records with missing FQ result
tables outcome02 fq02 /o
```

which gives:

Outcome: Binomial outcome			
Binomial FQ resistance	Failure	Success	Total
Resistant	18	44	62
Susceptible	59	380	439
Total	77	424	501

Exposure: Binomial FQ resistance = Resistant
 Outcome: Binomial outcome = Failure

Odds Ratio = 2.63 (95% CI: 1.43-4.86) (Robins, Greenland, Breslow CI)

Second, we make a stratified analysis to look at the influence of SEX:

```
tables outcome02 fq02 sex /o
```

which gives a summary:

Binomial outcome by Binomial FQ resistance adjusted for Patient's sex			
	N = 501	N	OR (95% CI)
Crude		501	2.63 (1.43-4.86)
Adjusted		501	2.65 (1.43-4.93)
Patient's sex: Male	355		1.91 (0.85-4.29)
Patient's sex: Female	146		4.54 (1.65-12.54)

Summary Estimates
 Total 2 strata. 2 informative & 0 non-informative.
 Exposure: Binomial FQ resistance = Resistant
 Outcome: Binomial outcome = Failure

by the Mantel-Haenszel procedure.

The first table in R requires that we make a new variable for a binary result of fluoroquinolone resistance and then create the table:

```
e_ex02_02.dat[e_ex02_02.dat$fq04=="Susceptible", "fq02"] <- "1-Susceptible"
e_ex02_02.dat[e_ex02_02.dat$fq04=="Low-level resistance", "fq02"] <- "2-Resistant"
e_ex02_02.dat[e_ex02_02.dat$fq04=="High-level resistance", "fq02"] <- "2-Resistant"
e_ex02_fq02.dat <- data.frame(e_ex02_02.dat)
```

At this point let's introduce `attach`. Followed in parenthesis by the dataset name:

```
attach(e_ex02_02_fq02.dat)
```

it puts the dataset `e_ex02_02_fq02.dat` into the search path and we do thus not need to repeat the dataset name all the time, it suffices to write the variable names. It is equally important not to forget to detach a dataset to get it out of the path. Thus, we type here:

```
detach(e_ex02_02.dat)
```

If the data set is in the path, what would otherwise be:

```
table(e_ex02_fq02.dat$fq02, e_ex02_fq02.dat$outcome02)
```


becomes simplified:

```
table(fq02, outcome02)
```

This gives just the bare bone core of the correct output table:

	Success	Failure
1-Susceptible	380	59
2-Resistant	44	18

Note here that for the R command `table` the sequence is different from EpiData: the first variable gives the row and the second variable the column. We noted earlier that the default sequence in R is row, then column. Note also that we numbered “1-Susceptible” and “2-Resistant” to get these into our preferred alphabetical sequence (the default would be the inverse).

Could we somehow use what we did in Exercise 1, edit it a bit and get it functional for this situation? Open the `e_ex01_or.r` in the text editor and remove everything, except the following lines:

```
or <- (tab[1,1]/tab[2,1])/(tab[1,2]/tab[2,2])
se <- sqrt(1/a+1/b+1/c+1/d)
or.ci.lower <- exp(log(or)-1.96*se)
or.ci.upper <- exp(log(or)+1.96*se)
print(tab)
cat("\nOR:", round(or, digits=3), "\n95% CI:", round(or.ci.lower, digits=3),
    "to ", round(or.ci.upper, digits=3))
```

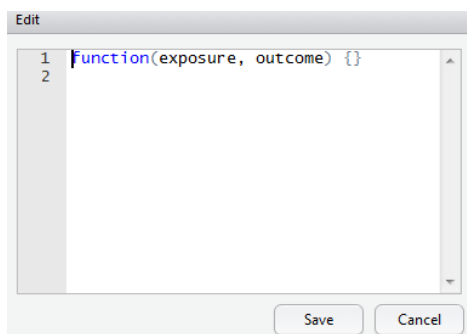
We are going to create a **Function** that can be applied in the future to the same setting. We have already used functions, `TABLE` is a function. To make our own function, type:

```
tab2by2 <- function(exposure, outcome) {}
```

This is the framework only, and it doesn’t yet do anything. To put it to useful work, type:

```
fix(tab2by2)
```

and a function editor window opens:



Our commands will go between the curly braces `{ }`. It is a custom (but no real need, but we will faithfully follow customs of those with more experience, they usually have their reasons) to put the opening brace `{` on a new line and then the commands after that on new lines, and have the closing brace `}` again at the end alone on a line. After we paste from the text editor what we have, we should thus get:

```

Edit
1 function(exposure, outcome)
2   {
3     or <- (tab[1,1]/tab[2,1])/(tab[1,
4     se <- sqrt(1/a+1/b+1/c+1/d)
5     or.ci.lower <- exp(log(or)-1.96*s
6     or.ci.upper <- exp(log(or)+1.96*s
7     print(tab)
8     cat("\nOR:", round(or, digits=3),
9   }

```

It is not yet quite correct, because there is no object `tab`. Thus type on the line before the “or”:

```
tab <- table(exposure, outcome)
```

That is, we assign the output of any two-by-two table of this format to an object `tab`. You may note that the RStudio short-cut key **ALT+-** to get the assign combination “<-” is non-functional in functions, you have to type it out. As we are already using `a`, `b`, `c`, and `d`, we might as well keep these, and then make in the “or” line assignments to these, so that in the revision we get:

```

function(exposure, outcome)
{
  tab <- table(exposure, outcome)
  a <- tab[1,1]; b <- tab[2,1]; c <- tab[1,2]; d <- tab[2,2]
  or <- (a/c)/(b/d)
  se <- sqrt(1/a+1/b+1/c+1/d)
  or.ci.lower <- exp(log(or)-1.96*se)
  or.ci.upper <- exp(log(or)+1.96*se)
  print(tab)
  cat("\nOR:", round(or, digits=3), "\n95% CI:", round(or.ci.lower,
  digits=3), "to ", round(or.ci.upper, digits=3))
}

```

Save it. We could save it later when quitting in the workspace, but it is better to also save in in a file and doing that now:

```
save(tab2by2, file = "tab2by2.r")
```

Conversely, when we open RStudio anew, it can be loaded whenever we need it with:

```
load("tab2by2.r")
```

Note that you actually need the full path, thus in our case:

```
load("C:/epidata_course/tab2by2.r")
```

Also note that this approach results in an `*.r` file that is not anymore a straight text file (you see gibberish in the text editor). It seems to be the price that has to be paid for a function we write by ourselves.

Before we apply it, we need to reopen and Run our `e_ex02.r` script which should read now:

```

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

#####
# 1) Import a Stata file

library(foreign)
e_ex02.dat <- read.dta("e_ex02.dta")
# e_ex02.dat <- read.dta("c:/epidata_course/e_ex02.dta")
write.table(e_ex02.dat, file="e_ex02.dat", row.names=TRUE)

```

```

# See the first 5 records
e_ex02.dat[1:5,]

# Get the list of variable names only
names(e_ex02.dat)

table(e_ex02.dat$fq04)

#####
# 2) Create full unaltered set (515 records)
e_ex02.dat[e_ex02.dat$fq04=="Missing", "fq04"] <- NA
is.na(e_ex02.dat$fq04)
e_ex02_01.dat <- data.frame(e_ex02.dat)

#####
# 3) Create a subset with records with known FQ DST result (401 records)
e_ex02_02.dat <- e_ex02.dat[which(e_ex02.dat$fq04 != "Missing"), ]
# or alternatively with the SUBSET function:
e_ex02_02.dat <- subset(e_ex02.dat, fq04 != "Missing")

#####
# 4) Create a subset with records with known FQ resistance and
# bacteriological success or failure (53 records)
# Commands on three lines
# e_ex02_03a.dat <- subset(e_ex02_02.dat, outcome07 != "Death")
# e_ex02_03b.dat <- subset(e_ex02_03a.dat, outcome07 != "Default")
# e_ex02_03.dat <- subset(e_ex02_03b.dat, fq04 != "Susceptible")
# Alternatively, commands on single line
e_ex02_03.dat <- subset(e_ex02_02.dat, outcome07 != "Death" & outcome07 != "Default"
  & fq04 != "Susceptible")

# Table of outcome by FQ resistance
# Create first a new variable FQ02

e_ex02_02.dat[e_ex02_02.dat$fq04=="Susceptible", "fq02"] <- "1-Susceptible"
e_ex02_02.dat[e_ex02_02.dat$fq04=="Low-level resistance", "fq02"] <- "2-Resistant"
e_ex02_02.dat[e_ex02_02.dat$fq04=="High-level resistance", "fq02"] <- "2-Resistant"
e_ex02_fq02.dat <- data.frame(e_ex02_02.dat)
table(e_ex02_fq02.dat$fq02, e_ex02_fq02.dat$outcome02)

write.table(e_ex02_01.dat, file="e_ex02_01.dat", row.names=TRUE)
write.table(e_ex02_02.dat, file="e_ex02_02.dat", row.names=TRUE)
write.table(e_ex02_03.dat, file="e_ex02_03.dat", row.names=TRUE)

#####
# 5) Make functions: 2-by-2 table
# Prepare making a function
# tab2by2 <- function(exposure, outcome) {}
# fix(tab2by2)
# save(tab2by2, file = "tab2by2.r")

attach(e_ex02_fq02.dat)

```

As the last command is ATTACH the proper file, let's try to apply it. Type:

```

load("C:/epidata_course/tab2by2.r")
tab2by2(fq02, outcome02)

```

and you should get:

```

      outcome
exposure Success Failure
1-Susceptible 380    59
2-Resistant   44    18

```

```

OR: 2.635
95% CI: 1.427 to 4.865

```

identical to the EpiData Analysis output (see above). Note that the R `load` function is necessary only after exiting R and reopening the script. Similarly, the three lines:

```

# tab2by2 <- function(exposure, outcome) {}
# fix(tab2by2)
# save(tab2by2, file = "tab2by2.r")

```

have been put as comments as they do not need repeat execution once done.

We can use this function for any other EXPOSURE-OUTCOME pair, such as:

```
tab2by2(sex, outcome02)
```

```

      outcome
exposure Success Failure
Male      308    47
Female    116    30

```

```

OR: 1.695
95% CI: 1.022 to 2.809

```

In EpiData Analysis:

```
tables outcome02 sex /o
```

Outcome: Binomial outcome			
Patient's sex	Failure	Success	Total
Female	30	116	146
Male	47	308	355
Total	77	424	501

Exposure: Patient's sex = Female
Outcome: Binomial outcome = Failure

Odds Ratio = 1.69 (95% CI: 1.02-2.81) (Robins, Greenland, Breslow CI)

We surely have to watch out for the variable `SEX`. You know from earlier that EpiData Analysis inverts (by design) the sequence of values for the exposure and outcome variables, while the net effect on the odds ratio thus remains unaffected.

To summarize: `TABLE` is an inbuilt function in R which cross tabulates the two variables. Replacing `TABLE` by our function ‘`tab2by2`’ runs our code creating the table and with the odds ratio and the 95% confidence interval, displayed it in the desired format. There are many functions that have been created already by R users and collaborators which we can make use of. Thus, a skill we have to master is to search for a function which suits our purpose and make correct and careful use of it. This example illustrates the basic mode of using and applying functions in R.

A generically applicable function for the analysis of a 2-by-2-by-2 table

In the previous paragraph we dealt with a **matrix** (a two-dimensional object of like elements). It was the special case of a 2-by-2 matrix. In this part, we move now forward to deal with **arrays**. An array is an n-dimensional table of like objects. In particular, we will concern

ourselves with the special case of a 2-by-2-by-2 table, i.e. a 3-dimensional table. We can construct such an object in R with the command:

```
z <- array(1:8, c(2, 2, 2)); z
```

and get:

```
, , 1
      [,1] [,2]
[1,]    1    3
[2,]    2    4

, , 2
      [,1] [,2]
[1,]    5    7
[2,]    6    8
```

What is the location of “7”? Try:

```
u <- z[1, 2, 2]; u
```

The third dimension (row is the first, the column the second, the added one the third) is also indicated by the the two leading commas:

```
, , 2
```

We mentioned before that the location [indexing] in R is defined within brackets []. The default sequence is by dimension, separated by commas, where the first dimension is always the row, the second the column and here the third is what we epidemiologists call the stratifying dimension. If we apply this concept of a multidimensional array to epidemiology, we thus have the concept of stratification.

The generic grammar for a table command for this basic stratification is:

```
table(exposure, outcome, stratvar)
```

or

```
table(rowvar, columnvar, stratvar)
```

where `stratvar` is the name of the stratifying variable. In our case, specifically, we try:

```
table(fq02, outcome02, sex)
```

to get, if the last `attach` command was `attach(e_ex02_fq02.dat)`:

```
, , sex = Male

      outcome02
fq02  Success Failure
1-Susceptible  274    38
2-Resistant    34     9

, , sex = Female

      outcome02
fq02  Success Failure
1-Susceptible  106    21
2-Resistant    10     9
```

In 1959, Mantel and Haenszel proposed an efficient method for estimating a summary odds ratio from a series of 2 by 2 tables. It is easy to apply and does not require iterative calculations. In the following introduction, we follow Schlesselman's explanations and use of example (Schlesselman J J. Case-control studies. Design, conduct, analysis. New York: Oxford University Press, 1982).

The Mantel-Haenszel estimate of the odds ratio (OR_{mh}), adjusted for the effects of the stratification variables is calculated as:

$$\text{SUM}(a_i d_i / n_i) / \text{SUM}(b_i c_i / n_i)$$

An estimate of the variance of the OR_{mh} has been proposed. If we define:

$$w_i = b_i c_i / n_i$$

and

$$v_i = (a_i + c_i) / a_i c_i + (b_i + d_i) / b_i d_i$$

then the variance of the $\log_e OR_{mh}$ is given by:

$$\text{var}(\ln OR_{mh}) \approx \text{SUM} w_i^2 v_i / (\text{SUM} w_i)^2$$

The approximate 95% confidence interval is thus given by:

$$\ln OR_{mh} \pm 1.96 * \text{SQRT}[\text{var}(\ln OR_{mh})]$$

If we apply this to our data:

	Female		Male	
	Failure	Success	Failure	Success
Resistant	9	10	9	34
Susceptible	21	106	38	274
OR	4.543		1.909	
n_i	146		355	
W_i	1.438		3.639	
v_i	0.268		0.170	
$a_i d_i / n_i$	6.534		6.946	
$b_i c_i / n_i$	1.438		3.639	
$\text{SUM}(a_i d_i / n_i)$	13.481			
$\text{SUM}(b_i c_i / n_i)$	5.078			
OR_{mh}	2.655			
$w_i^2 * v_i$	0.555		2.258	
$\text{SUM}(w_i^2 * v_i)$	2.813			
$(\text{SUM} w_i)^2$	25.784			
$\text{SUM}(w_i^2 * v_i) / (\text{SUM} w_i)^2$	0.109			
$\text{var}(\ln OR_{mh})$	0.109			
OR _{mh} -lower	1.390			
OR _{mh} -upper	5.072			

EpiData Analysis gives:

$$OR_{mh}: 2.65 \quad (1.43-4.93)$$

The point estimate is the same, but the 95% CI interval is slightly different because EpiData Analysis uses consistently the Robins, Greenland, Breslow confidence intervals, while we followed here Schlesselman's example using Hauck's interval. As the exercise is about the

principle of learning on how to go about when creating a function, the choice of the confidence interval does not really matter here.

We will keep to this most simple example with only 8 cells total among three variables: each variable is binary.

Let's best start from scratch and make a new function:

```
ormh <- function(exposure, outcome, stratvar) {}
```

We create a function ORMH (short for Mantel-Heanszel odds ratio) that takes three parameters, in that given sequence.

We propose to draft the content that goes between the curly braces `{ }` in the text editor before we put into it with `fix`. We can modify what we had before, just slightly so:

```
tab <- table(exposure, outcome, stratvar)
a1 <- tab[1,1,1]; b1 <- tab[2,1,1]; c1 <- tab[1,2,1]; d1 <- tab[2,2,1]
a2 <- tab[1,1,2]; b2 <- tab[2,1,2]; c2 <- tab[1,2,2]; d2 <- tab[2,2,2]
```

This takes into account that we work now with a 3-dimensional array and have `a1`, `a2`, and `b1`, `b2`, etc.

We have to create the $a_i d_i / n_i$ and the $b_i c_i / n_i$ for each of the two tables:

```
adn1 <- (a1*d1)/n1
adn2 <- (a2*d2)/n2
bcn1 <- (b1*c1)/n1
bcn2 <- (b2*c2)/n2
```

and from these we calculate the object MHOR:

```
mhor <- (adn1+adn2)/(bcn1+bcn2)
```

Note that we chose the object name to be `mhor` to distinguish in from the function name `ormh`. Before we get too much carried away in our excitement, we should check whether we are on the right track:

```
fix(ormh)
```

and insert what we currently have:

```
function(exposure, outcome, stratvar)
{
  tab <- table(exposure, outcome, stratvar)
  a1 <- tab[1,1,1]; b1 <- tab[2,1,1]; c1 <- tab[1,2,1]; d1 <- tab[2,2,1]
  a2 <- tab[1,1,2]; b2 <- tab[2,1,2]; c2 <- tab[1,2,2]; d2 <- tab[2,2,2]
  n1 <- a1+b1+c1+d1
  n2 <- a2+b2+c2+d2
  adn1 <- (a1*d1)/n1
  adn2 <- (a2*d2)/n2
  bcn1 <- (b1*c1)/n1
  bcn2 <- (b2*c2)/n2
  mhor <- (adn1+adn2)/(bcn1+bcn2)
  print(tab)
  cat("\nOR:", round(mhor, digits=3))
}
```

and test it with the command:

```
ormh(fq02, outcome02, sex)
```

We get:

```
, , stratvar = Male
      outcome
exposure  Success Failure
1-Susceptible  274     38
2-Resistant    34      9
```

```
, , stratvar = Female
      outcome
exposure  Success Failure
1-Susceptible  106     21
2-Resistant    10      9
```

OR: 2.655

We are definitely on the right track! Therefore, we might continue requiring next the two components w and v that we calculate as intermediate steps for calculation of the variance and then the variance itself:

```
w1 <- (b1*c1)/n1
w2 <- (b2*c2)/n2
v1 <- ((a1+c1)/(a1*c1))+((b1+d1)/(b1*d1))
v2 <- ((a2+c2)/(a2*c2))+((b2+d2)/(b2*d2))
varor <- ((w1^2*v1)+(w2^2*v2))/((w1+w2)^2)
```

Then we calculate the confidence interval:

```
se <-sqrt(var.mhor)
mhor.lower <-exp(log(mhor)-1.96*se)
mhor.upper <-exp(log(mhor)+1.96*se)
```

Finally, by adding what is to appear on the screen, we get the full function as:

```
function(exposure, outcome, stratvar)
{
  tab <- table(exposure, outcome, stratvar)
  a1 <- tab[1,1,1]; b1 <- tab[2,1,1]; c1 <- tab[1,2,1]; d1 <- tab[2,2,1]
  a2 <- tab[1,1,2]; b2 <- tab[2,1,2]; c2 <- tab[1,2,2]; d2 <- tab[2,2,2]
  a <- a1+a2; b <- b1+b2; c <-c1+c2; d <- d1+d2
  or <- (a/c)/(b/d)
  se <- sqrt(1/a+1/b+1/c+1/d)
  or.ci.lower <- exp(log(or)-1.96*se)
  or.ci.upper <- exp(log(or)+1.96*se)
  n1 <- a1+b1+c1+d1
  n2 <- a2+b2+c2+d2
  adn1 <- (a1*d1)/n1
  adn2 <- (a2*d2)/n2
  bcn1 <- (b1*c1)/n1
  bcn2 <- (b2*c2)/n2
  mhor <- (adn1+adn2)/(bcn1+bcn2)
  w1 <- (b1*c1)/n1
  w2 <- (b2*c2)/n2
  v1 <- ((a1+c1)/(a1*c1))+((b1+d1)/(b1*d1))
  v2 <- ((a2+c2)/(a2*c2))+((b2+d2)/(b2*d2))
  var.mhor <- ((w1^2*v1)+(w2^2*v2))/((w1+w2)^2)
```



```

se <-sqrt(var.mhor)
mhor.lower <-exp(log(mhor)-1.96*se)
mhor.upper <-exp(log(mhor)+1.96*se)
print(tab)
cat("\nOR adj:", round(mhor, digits=3), "\n95% CI:", round(mhor.lower,
  digits=3), "-", round(mhor.upper, digits=3))
}

```

If we test with:

```
ormh(oflres, outcome02, sex)
```

we get:

```

, , stratvar = Male

      outcome
exposure  Success Failure
1-Susceptible    274     38
2-Resistant      34      9

, , stratvar = Female

      outcome
exposure  Success Failure
1-Susceptible    106     21
2-Resistant      10      9

```

```

OR: 2.655
95% CI: 1.39 5.072

```

We can use this generically for other variables, such as:

```
ormh(fq02, outcome02, cxr02)
```

and get:

```

, , stratvar = Not known bilateral

      outcome
exposure  Success Failure
1-Susceptible    77     13
2-Resistant      7      3

, , stratvar = Bilateral

      outcome
exposure  Success Failure
1-Susceptible    303     46
2-Resistant      37     15

```

```

OR: 2.647
95% CI: 1.432 4.892

```

Do not forget to save the function to a file:

```
save(ormh, file = "ormh.r")
```

Admittedly, what we have done here is still quite amateurish. Fortunately, more professional contributors to R have written a function to calculate the Mantel-Haenszel estimate of the odds ratio. If you just type the function:

```
mantelhaen.test
```

you see the full function. We can use it for our example:

```
mantelhaen.test(table(fq02, outcome02, sex))
```

and we get:

```
Mantel-Haenszel chi-squared test with continuity correction

data: table(fq02, outcome02, sex)
Mantel-Haenszel X-squared = 8.8339, df = 1, p-value = 0.002957
alternative hypothesis: true common odds ratio is not equal to 1
95 percent confidence interval:
 1.430554 4.926883
sample estimates:
common odds ratio
 2.65484
```

To look what the author of that function actually wrote, you can type in analogy of what we did above:

```
fix(mantelhaen.test)
```


We note thereby that the producers of this function that is part of the basic R package has the same approach to the calculation of the confidence interval as was chosen for EpiData Analysis, which gave as a summary:


Binomial outcome by Binomial FQ resistance adjusted for Patient's sex				
	N = 501	N	OR	(95% CI)
Crude		501	2.63	(1.43-4.86)
Adjusted		501	2.65	(1.43-4.93)
Patient's sex: Male		355	1.91	(0.85-4.29)
Patient's sex: Female		146	4.54	(1.65-12.54)

In summary, we have demonstrated that we can create any function in R which has a generalizable applicability. Fortunately, others have already done most of what is probably needed and it is thus not usually necessary to write our own functions. This is good news, of course, as it proves rather tedious and definitely requires sometimes basic, and sometimes more sophisticated knowledge of statistics. It also requires a thorough knowledge of the S programming language. Functions abound for R, but one has to look out for them to know how they are used properly. How did we find that this one exists? At the R prompt, type:

```
??mantel
```

and you get in the right lower quadrant of RStudio:

Search Results 

The search string was "mantel" 

Help pages:

- [base::all.equal](#) Test if Two Objects are (Nearly) Equal
- [stats::Box.test](#) Box-Pierce and Ljung-Box Tests
- [stats::mantelhaen.test](#) Cochran-Mantel-Haenszel Chi-Squared Test for Count Data
- [survival::rats](#) Rat treatment data from Mantel et al

where you then click on the link for detailed information.

Tasks:

- o Append the `ormh.r` script to calculate and show also the stratum-specific and crude (unstratified) odds ratioa with 95% confidence intervals (you may use the same type of confidence interval as we used in the `tab2by2.r` script)*