R-exercise for Friday 31

October 2014
http://BendixCarstensen.com/Epi
Version 5

Compiled Thursday 30th October, 2014, 19:14
from: C:/Bendix/undervis/SPE/Intro/R-intro-SDC.tex

Bendix Carstensen Steno Diabetes Center, Gentofte, Denmark
& Department of Biostatistics, University of Copenhagen
bxc@steno.dk
www.pubhealth.ku.dk/~bxc

Edition 2014 by Bendix Carstensen

Contents

Chapter 1

Getting R running on your computer

1.1 What is R?

R is free program for data analysis and graphics. It contains all state of the art statistical methods, and has become the preferred analysis tool for most professional statisticians in the world. It can be used as simple calculator and as a very specialized statistical analysis and reporting machinery.

The special thing about R is that you enter commands from the keyboard into a console window, where you also see the results. This is an advantage because you end up with a script that you can use to *reproduce* your analyses—a requirement in any scientific endeavour.

The disadvantage is that you somehow have to find out what to type. The practicals will contain some hints, and you will mostly be using R as a calculator, as you just saw — type an expression, hit the return key and you get the result.

1.2 Getting R

You can obtain R, which is free, from CRAN (the **C**omprehensive **R** **A**rchive **N**etwork), at http://cran.r-project.org/. Under "`Download R for Windows`" click on "install R for the first time" and then on "Download R 3.0.2 for Windows", which is a self-extracting installer. This means that if you save it to your computer somewhere and click on it, it will install R for you.

Apart from what you have downloaded there are several thousand add-on packages to R dealing with all sorts of problems from ecology to fiance and incidentally, epidemiology. You must download these manually. In this course we shall only need the `Epi` package.

1.2.1 Starting R

You start R by clicking on the icon that the installer has put on your desktop. You should edit the properties of this, so that R starts in the folder that you have created on your computer for this course.

Once you have installed R, start it, and in the menu bar click on Packages → Install package(s)..., chose a mirror (this is just a server where you can get the stuff), and then the `Epi` package.

Once R (hopefully) has told you that it has been installed, you can type:

```
> library( Epi )
```

to get access to the `Epi` package. You can get an overview of the functions and datasets in the package by typing:

```
> library( help=Epi )
```

It should be apparent that you have version 1.1.49 of the `Epi` package. For documentauon purposes it is often useful to have the following at the beginning of your program:

```
> sessionInfo()
  R version 3.1.1 (2014-07-10)
  Platform: i386-w64-mingw32/i386 (32-bit)

  locale:
  [1] LC_COLLATE=Danish_Denmark.1252  LC_CTYPE=Danish_Denmark.1252
  [3] LC_MONETARY=Danish_Denmark.1252 LC_NUMERIC=C
  [5] LC_TIME=Danish_Denmark.1252

  attached base packages:
  [1] utils     datasets  graphics  grDevices stats     methods   base

  other attached packages:
  [1] Epi_1.1.68     foreign_0.8-61
```

1.2.2 Quitting R

Type `q()` in the console, and answer "No" when asked whether you want to save workspace image.

1.3 Working with the script editor

If you click on File → New script, R will open a window for you which is a text-editor very much like Notepad.

If you write a command in it you can transfer it to the R console and have it executed by pressing CTRL-r. If nothing is highlighted, the line where the cursor is will be transmitted to the console and the cursor will move to the next line. If a part of the screen is highlighted the highlighted part will be transmitted to the console. Highlighting can also be used to transmit only a part of a line of code.

1.3.1 Rstudio

This is an interface that allows you to have a slithly more flexible script-editor than the built-in, R-studio har syntax coloriung which can be very nice. You can obtain it from http://rstudio.com.

1.3.2 Try!

Now, **either** open a script by File → New script, and type (omit the ">" in the beginning of the line), **or** fire up R-studio and type in the editor window:

```
> 5+7
> pi
> 1:10
> N <- c(27,33,81)
> N
```

Run the lines one at a time by pressing CTRL-r, (in R-studio it is CTRL-ENTER) and see what happens.

You can also type the commands in the console directly. But then you will not have a record of what you have done. Well, you can press File → Save History and save all you typed in the console (including the 73.6% commands with errors).

1.4 Changing the looks ...

1.4.1 ... of standard R

If you want R to start up with a different font, different colors etc., the go to the folder where R is installed — most likely `Program Files\R\R-2.13.1`, then to the folder `etc`, and open the file `Rconsole` with Notepad. In the file are specifications on how R will look when you start it, pretty self-explanatory, except perhaps for MDI.

MDI means "**M**ultiple **D**isplay **I**nterface", which means you get a single R-window, and within that sub-windows with the console, the script editor, graphs etc. If this is set to "no", you get SDI which means "**S**ingle **D**isplay **I**nterface", which means that R will open the console, script editor etc. in separate windows of their own.

A withe background can be trying to look at so on my (BxC) computer I use a bold font and the following colors:

```
> background = gray5
> normaltext = yellow2
> usertext = green
> pagerbg = gray5
> pagertext = yellow2
> highlight = red
> dataeditbg = gray5
> dataedittext = red
> datadituser = yellow2
> editorbg = gray5
> editortext = lightblue
```

(If you want to know which colors are available in R, just give the command `colors()`).

1.4.2 ... of Rstudio

Click on Tools→Global options...→Apperance and choose Consolas font, 16 pt, Editor theme `Cobalt`

1.5 Further reading

On the CRAN web-site the last menu-entry on the left is "Contributed" and will take you to a very long list of various introductions to R, including manuals in esoteric languages such as Danish, Finnish and Hungarian.

Chapter 2

Some basic commands in R

2.1 Preliminaries

The purpose of these notes is to describe a small subset of the Rlanguage, sufficient to allow someone new to R to get started. The exercises are important because they reinforce basic aspects of R. For further details about R we refer the reader to **An Introduction to R** by W.N.Venables, D.M.Smith, and the R development team. This can be downloaded from the R website at http://www.r-project.org.

To start R click on the R icon. To change your working directory click on File → Change dir... and select the directory you want to work in. Alternatively you can write:

```
> setwd("c:/where/alll/my/files/are")
```

To get out of R click on the File menu and select Exit, or simpler just type "q()". You will be offered the chance to save the work space, but at this stage just exit without saving, then start R again, and change the working directory, as before.

R is case sensitive, so that A is different from a. Commands in R are generally separated by a newline, although a semi-colon can also be used. When using R it makes sense to avoid as much typing as possible by recalling previous commands using the vertical arrow key and editing them.

2.2 Using R as a calculator

Typing 2+2 will return the answer 4, typing 2^3 will return the answer 8 (2 to the power of 3), typing log(10) will return the natural logarithm of 10, which is 2.3026, and typing sqrt(25) will return the square root of 25.

Instead of printing the result you can store it in an object, say

```
> a <- 2+2
```

which can be used in further calculations. The expression <-, pronounced "gets", is called the assignment operator, and is obtained by typing < and then -. The assignment operator can also be used in the opposite direction, as in

```
> 2+2 -> a
```

The contents of `a` can be printed by typing `a`.

Standard probability functions are readily available. For example, the probability below 1.96 in a standard normal (i.e. Gaussian) distribution is obtained with

```
> pnorm(1.96)
```

while

```
> pchisq(3.84,1)
```

will return the probability below 3.84 in a χ^2 distribution on 1 degree of freedom, and

```
> pchisq(3.84,1,lower.tail=FALSE)
```

will return the probability above 3.84.

Exercise 2.1.

1. Calculate $\sqrt{3^2+4^2}$.
2. Find the probability above 4.3 in a chi-squared distribution on 1 degree of freedom.

2.3 Objects and functions

All commands in R are *functions* which act on *objects*. One important kind of object is a *vector*, which is an ordered collections of numbers, or an ordered collection of character strings. Examples of vectors are 4, 6, 1, 2.2, which is a numeric vector with 4 components, and "Charles Darwin", "Alfred Wallace" which is a vector of character strings with 2 components. The components of a vector must be of the same type (numeric or character). The combine function `c()`, together with the assignment operator, is used to create vectors. Thus

```
> v <- c(4, 6, 1, 2.2)
```

creates a vector `v` with components 4, 6, 1, 2.2 by first combining the 4 numbers 4, 6, 1, 2.2 in order and then assigning the result to the vector `v`. Collections of components of different types are called *lists*, and are created with the `list()` function. Thus

```
> m <- list(4, 6, "name of company")
```

creates a list with 3 components. The main differences between the numbers 4, 6, 1, 2.2 and the vector `v` is that along with `v` is stored information about what sort of object it is and hence how it is printed and how it is combined with other objects. Try

```
> v
> 3+v
> 3*v
```

and you will see that R understands what to do in each case. This may seem trivial, but remember that unlike most statistical packages there are many different kinds of object in R.

You can get a description of the structure of any object using the function `str()`. For example, `str(v)` shows that `v` is numeric with 4 components.

2.4 Sequences

It is not always necessary to type out all the components of a vector. For example, the vector (15, 20, 25, ... ,85) can be created with

```
> seq(15, 85, by=5)
```

and the vector (5, 20, 25, ... ,85) can be created with

```
> c(5,seq(20, 85, by=5))
```

You can learn more about functions by typing ? followed by the function name. For example `?seq` gives information about the syntax and usage of the function `seq()`.

Exercise 2.2.

1. Create a vector `w` with components 1, -1, 2, -2
2. Print this vector (to the screen)
3. Obtain a description of `w` using `str()`
4. Create the vector `w+1`, and print it.
5. Create the vector (0, 1, 5, 10, 15, ... , 75) using c() and seq().

2.5 The births data

Table 2.1: *Variables in the births dataset*

Variable	Units or Coding	Type	Name
Subject number	–	categorical	`id`
Birth weight	grams	metric	`bweight`
Birth weight < 2500 g	1=yes, 0=no	categorical	`lowbw`
Gestational age	weeks	metric	`gestwks`
Gestational age < 37 weeks	1=yes, 0=no	categorical	`preterm`
Maternal age	years	metric	`matage`
Maternal hypertension	1=hypertensive, 0=normal	categorical	`hyp`
Sex of baby	1=male, 2=female	categorical	`sex`

The most important example of a vector in epidemiology is the data on a variable recorded for a group of subjects. To introduce R we use the births data which concern 500 mothers who had singleton births in a large London hospital. These data are available as an R object called `births` in the Epi package. You can get them into your workspace by:

```
> library( Epi )
> data( births )
```

Try

```
> objects()
```

to make sure that you have an object called births in your working directory. A more detailed overview of the objects in your workspace is obtained by:

```
> lls()
```

The function

```
> str(births)
```

shows that the object `births` is a data frame with 500 observations of 8 variables. The names and types of the variables are also shown together with the first 10 values of each variable.

Some of the variables which make up these data take integer values while others are numeric taking measurements as values. For most variables the integer values are just codes for different categories, such as `"male"` and `"female"` which are coded 1 and 2 for the variable `sex`.

Exercise 2.3.

1. The dataframe `"diet"` in the Epi package contains data from a follow-up study with coronary heart disease as the end-point. Load these data with:

   ```
   > data(diet)
   ```

   and print the contents of the data frame to the screen.
2. Check that you now have two objects, `births`, and `diet` in your work space.
3. Obtain a description of the object `diet`.
4. Remove the object `diet` with the command

   ```
   > rm(diet)
   ```

5. Check that you only have the object `births` left.

2.6 Referencing parts of the data frame

Typing `births` will list the entire data frame - not usually very helpful. Now try

```
> births[1,"bweight"]
```

This will list the value taken by the first subject for the `bweight` variable. Similarly

```
> births[2,"bweight"]
```

will list the value taken by the second subject for `bweight`, and so on. To list the data for the first 10 subject for the `bweight` variable, try

```
> births[1:10, "bweight"]
```

and to list all the data for this variable, try

```
> births[, "bweight"]
```

Exercise 2.4.

1. Print the data on the variable `gestwks` for subject 7 in the `births` data frame.
2. Print all the data for subject 7.
3. Print all the data on the variable `gestwks`.

2.7 Summaries

A good way to start an analysis is to ask for a summary of the data by typing

```
> summary(births)
```

To see the names of the variables in the data frame try

```
> names(births)
```

Variables in a data frame can be referred to by name, but to do so it is necessary also to specify the name of the data frame. Thus `births$hyp` refers to the variable `hyp` in the `births` data frame, and typing `births$hyp` will print the data on this variable. To summarize the variable `hyp` try

```
> summary(births$hyp)
```

In most datasets there will be some missing values. These are usually coded using tab delimited blanks to mark the values which are missing. R then codes the missing values using the `NA` (not available) symbol. The summary shows the number of missing values for each variable.

2.8 Turning a variable into a factor

In R categorical variables are known as *factors*, and the different categories are called the levels of the factor. Variables such as `hyp` and `sex` are originally coded using integer codes, and by default R will interpret these codes as numeric values taken by the variables. For R to recognize that the codes refer to categories it is necessary to convert the variables to be factors, and to label the levels. To convert the variable `hyp` to be a factor, try

```
> hyp <- factor(births$hyp)
> str(births)
> objects()
```

which shows that `hyp` is both in your work space (as a factor), and in in the `births` data frame (as a numeric variable). It is better to use the transform function on the data frame, as in

```
> births <- transform(births, hyp=factor(hyp))
> str(births)
```

which shows that `hyp`, in the births data frame, is now a factor with two levels, labeled `"0"` and `"1"` which are the original values taken by the variable. It is possible to change the labels to (say) `"normal"` and `"hyper"` with

```
> births <- transform( births, hyp=factor(hyp,labels=c("normal","hyper")) )
> str(births)
```

Exercise 2.5.

1. Convert the variable `sex` into a factor
2. Label the levels of `sex` as `"male"` and `"female"`.

2.9 Frequency tables

When starting to look at any new data frame the first step is to check that the values of the variables make sense and correspond to the codes defined in the coding schedule. For categorical variables (factors) this can be done by looking at one-way frequency tables and checking that only the specified codes (levels) occur. The most useful function for making tables is `stat.table`. This is currently part of the `Epi` package, so you will need to load this package first with

```
> library(Epi)
```

The distribution of the factors `hyp` and `sex` can be viewed by typing

```
> stat.table(hyp,data=births)
> stat.table(sex,data=births)
```

Their cross-tabulation is obtained by typing

```
> stat.table(list(hyp,sex),data=births)
```

Cross-tabulations are useful when checking for consistency, but because no distinction is drawn between the response variable and any explanatory variables, they are not useful as a way of presenting data.

2.10 Grouping the values of a metric variable

For a numeric variable like `matage` it is often useful to group the values and to create a new factor which codes the groups. For example we might cut the values taken by `matage` into the groups 20–29, 30–34, 35–39, 40–44, and then create a factor called `agegrp` with 4 levels corresponding to the four groups. The best way of doing this is with the function `cut`. Try

```
> births <- transform(births,agegrp=cut(matage, breaks=c(20,30,35,40,45),right=FALSE))
> stat.table(agegrp,data=births)
```

By default the factor levels are labeled [20-25), [25-30), etc., where [20-25) refers to the interval which includes the left hand end (20) but not the right hand end (25). This is the reason for `right=FALSE`. When `right=TRUE` (which is the default) the intervals include the right hand end but not the left hand.

It is important to realize that observations which are not inside the range specified in the `breaks()` part of the command result in missing values for the new factor. For example, try

```
> births <- transform(births,agegrp=cut(matage, breaks=c(20,30,35),right=FALSE))
> summary(births)
```

Only observations from 20 up to, but not including 35, are included. For the rest, `agegrp` is coded missing. You can specify that you want to cut a variable into a given number of intervals of equal length by specifying the number of intervals. For example

```
> births <- transform(births,agegrp=cut(matage,breaks=5,right=FALSE))
> stat.table(agegrp,data=births)
```

shows 5 intervals of width 4.

Exercise 2.6.

1. Summarize the numeric variable `gestwks`, which records the length of gestation for the baby, and make a note of the range of values.
2. Create a new factor `gest4` which cuts `gestwks` at 20, 35, 37, 39, and 45 weeks, including the left hand end, but not the right hand. Make a table of the frequencies for the four levels of `gest4`.
3. Create a new factor `gest5` which cuts `gestwks` into 5 equal intervals, and make a table of frequencies.

2.11 Tables of means and other things

To obtain the mean of `bweight` by `sex`, try

```
> stat.table(sex, mean(bweight), data=births)
```

The headings of the table can be improved with

```
> stat.table(sex,list("Mean birth weight"=mean(bweight)),data=births)
```

To make a two-way table of mean birth weight by sex and hypertension, try

```
> stat.table(list(sex,hyp),mean(bweight),data=births)
```

and to tabulate the count as well as the mean, try

```
> stat.table(list(sex,hyp),list(count(),mean(bweight)),data=births)
```

Available functions for the cells of the table are `count, mean, weighted.mean, sum, min, max, quantile,median, IQR`, and `ratio`. The last of these is useful for rates and odds. For example, to make a table of the odds of low birth weight by hypertension, try

```
> stat.table(hyp, list("odds"=ratio(lowbw,1-lowbw,100)),data=births)
```

The scale factor 100 makes the odds per 100. Margins can be added to the tables, as required. For example,

```
> stat.table(sex, mean(bweight),data=births,margins=TRUE)
```

for a one-way table, and

```
> stat.table(list(sex,hyp),mean(bweight),data=births,margins=c(TRUE,FALSE))
> stat.table(list(sex,hyp), mean(bweight),data=births,margins=c(FALSE,TRUE))
> stat.table(list(sex,hyp), mean(bweight),data=births,margins=c(TRUE,TRUE))
```

for a two-way table.

Exercise 2.7.

1. Make a table of median birth weight by `sex`.
2. Do the same for gestation time, but include `count` as a function to be tabulated along with `median`. Note that when there are missing values for the variable being summarized the count refers to the number of non-missing observations for the row variable, not the summarized variable.
3. Create a table showing the mean gestation time for the baby by `hyp` and `lowbw`, together with margins for both.
4. Make a table showing the odds of hypertension by sex of the baby.

2.11.1 Other tabulation functions

You may want to take a look at the help pages for the functions:

- `table`
- `ftable`
- `xtabs`
- `addmargins`
- `array`
- `tapply`

One way to do this is to simply type:

```
> example( table )
```

2.12 Generating new variables

New variables can be produced using assignment together with the usual mathematical operations and functions:

`+` `-` `*` `log` `exp` `^` `sqrt`

The sign `^` means "to the power of", `log` means "natural logarithm", and `sqrt` means "square root".

The `transform()` function allows you to transform or generate variables in a data frame. For example, try

```
> births <- transform(births,
+           num1=1,
+           num2=2,
+           logbw=log(bweight))
```

The variable `logbw` is the natural logarithm of birth weight. Logs base 10 are obtained with `log10( )`.

2.13 Logical variables

Logical variables take the values TRUE or FALSE, and behave like factors. New variables can be created which are logical functions of existing variables. For example

```
> births <- transform(births, low=bweight<2000)
> str(births)
```

creates a logical variable `low` with levels TRUE and FALSE, according to whether `bweight` is less than 2000 or not. The logical expressions which R allows are

```
==      <      <=      >      >=      !=
```

The first is logical equals and the last is not equals. One common use of logical variables is to restrict a command to a subset of the data. For example, to list the values taken by `bweight` for hypertensive women, try

```
> births$bweight[births$hyp=="hyper"]
```

If you want the entire dataframe restricted to hypertensive women try:

```
> births[births$hyp=="hyper",]
```

The `subset()` function also allows you to take a subset of a data frame. Try

```
> subset(births, hyp=="hyper")
```

Exercise 2.8.

1. Create a logical variable called `early` according to whether `gestwks` is less than 30 or not.Make a frequency table of `early`.
2. Print the `id` numbers of women with `gestwks` less than 30 weeks.

Chapter 3

Working with R

3.1 Saving the work space

When exiting from R you are offered the chance of saving all the objects in your current work space. If you do so, the work space is re-instated next time you start R. It can be useful to do this, but before doing so it is worth tidying things up, because the work space can fill up with temporary objects, and it is easy to forget what these are when you resume the session.

3.2 Saving output in a file

To save the output from an R command in a file, for future use, the `sink()` command is used. For example,

```
> sink("output.txt")
> summary(births)
```

first instructs R to re-direct output away from the R terminal to the file `"output.txt"` and then summarizes the births data frame, the output from which goes to the sink. While a sink is open all output will go to it, replacing what is already in the file. To append output to a file, use the `append=TRUE` option with `sink()`. To close a sink, use

```
> sink()
```

Exercise 3.9.

1. Sink output to a file called `"output1.txt"`.
2. Make frequency tables of `hyp` and `sex`
3. Make a table of mean birth weight by sex
4. Close the sink
5. From windows, have a look inside the file `output1.txt` and check that the output you expected is in the file.

3.3 Saving R objects in a file

The command `read.table()` is relatively slow because it carries out quite a lot of processing as it reads the data. To avoid doing this more than once you can save the data frame, which includes the R information, and read from this saved file in future. For example,

```
> save(births, file="births.Rdata")
```

will save the births data frame in the file `births.Rdata`. By default the data frame is saved as a binary file, but the option `ascii=TRUE` can be used to save it as a text file. To load the object from the file use

```
> load("births.Rdata")
```

The commands `save()` and `load()` can be used with any R objects, but they are particularly useful when dealing with large data frames.

Exercise 3.10.

1. Use `read.table()` to read the data in the file `diet.txt` into a data frame called `diet`.
2. Save this data frame in the file `"diet.Rdata"`
3. Remove the data frame
4. Load the data frame from the file `"diet.Rdata"`.

3.4 Using a text editor with R

When working with R it is best to use a text editor to prepare a batch file (or script) which contains R commands and then to run them from the script. This means you can use the cut and paste facilities of the editor to cut down on typing. For Windows we recommend using the text editor Tinn-R, but you can use your favorite text editor instead if you prefer, and copy-paste commands from it into the R-console.

Alternatively you can use the built-in script-editor: Click on `File`→`New script`, or `File`→`Open script`, according to whether you are using an old script. You can move the current line from the script-editor to the console by `CTRL-R`. If you have highlighted a section of the script the highlighted part will be moved to the console.

Now start up the editor and enter the following lines:

```
> births <- transform( births,
+                      lowbw = factor(lowbw, labels=c("normal","low")),
+                        hyp = factor(hyp, labels=c("normal","hyper")),
+                        sex = factor(sex, labels=c("male","female")) )
```

Now save the script as `mygetbirths.R` and run it. One major advantage of running all your R commands from a script is that you end up with a record of exactly what you did which can be repeated at any time.

This will also help you redo the analysis in the (highly likely) event that your data changes before you have finished all analyses.

Exercise 3.11.

1. Create a script called mytab.R which includes the lines

   ```
   > stat.table(hyp,data=births)
   > stat.table(sex,data=births)
   ```

   and run just these two lines.
2. Edit the script to include the lines

   ```
   > stat.table(sex,mean(bweight),data=births)
   > stat.table(hyp,mean(bweight),data=births)
   ```

   and run these two lines.
3. Edit the script to create a factor cutting `matage` at 20, 30, 35, 40, 45 years, and run just this part of the script.
4. Edit the script to create a factor cutting `gestwks` at 20, 35, 37, 39, 45 weeks, and run just this part of the script.
5. Save and run the entire script.

3.5 The search path

R organizes objects in different positions on a search path. The command

```
> search()
```

shows these positions. The first is the work space, or global environment, the second is the Epi package, the third is a package of commands called methods, the fourth is a package called stats, and so on. To see what is in the work space try

```
> objects()
```

You should see just the objects `births` and `diet`. The command `objects(1)` does the same as `objects()`. A shorther name for the same function is `ls()`. In the Epi package is a function that gives a more detailed picture, `lls()`; try:

```
> lls()
```

To see what is in the Epi package, try

```
> ls(2)
```

When you type the name of an object R looks for it in the order of the search path and will return the first object with this name that it finds. This is why it is best to start your session with a clean workspace, otherwise you might have an object in your workspace that masks another one later in the search path.

3.6 Attaching a data frame

The function `objects(1)` shows that the only objects in the workspace are `births` and `diet`. To refer to variables in the `births` data frame by name it is necessary to specify the name of the data frame, as in `births$hyp`. This is quite cumbersome, and provided you are working primarily with one data frame, it can help to put a copy of the variables from a data frame in their own position on the search path. This is done with the function

```
> attach(births)
```

which places a copy of the variables in the `births` data frame in position 2. You can verify this with

```
> objects(2)
```

which shows the objects in this position are the variables from the `births` data frame. Note that the methods package has now been moved up to position 3, as shown by the `search()` function.

When you type the command:

```
> hyp
```

R will look in the first position where it fails to find `hyp`, then the second position where it finds `hyp`, which now gets printed.

Although convenient, attaching a data frame can give rise to confusion. For example, when you create a new object from the variables in an attached data frame, as in

```
> subgrp <- bweight[hyp==1]
```

the object `subgrp` will be in your workspace (position 1 on the search path) not in position 2. To demonstrate this, try

```
> objects(1)
> objects(2)
```

Similarly, if you modify the data frame in the workspace the changes will not carry through to the attached version of the data frame. The best advice is to regard any operation on an attached data frame as temporary, intended only to produce output such as summaries and tabulations.

Beware of attaching a data frame more than once - the second attached copy will be attached in position 2 of the search path, while the first copy will be moved up to position 3. You can see this with

```
> attach(births)
> search()
```

Having several copies of the same data set can lead to great confusion. To detach a data frame, use the command

```
> detach(births)
```

which will detach the copy in position 2 and move everything else down one position. To detach the second copy repeat the command `detach(births)`.

Exercise 3.12.

1. Use `search()` to make sure you have no data frames attached.
2. Use `objects(1)` to check that you have the data frame `births` in your work space.
3. Verify that typing `births$hyp` will print the data on the variable `hyp` but typing `hyp` will not.
4. Attach the `births` data frame in position 2 and check that the variables from this data frame are now in position 2.
5. Verify that typing `hyp` will now print the data on the the variable `hyp`.
6. Summarize the variable `bweight` for hypertensive women.

```
> setwd(sweave.wd)
```

Chapter 4

Sample size calculations

```
R version 3.1.1 (2014-07-10)
Platform: i386-w64-mingw32/i386 (32-bit)

attached base packages:
[1] utils     datasets  graphics  grDevices stats     methods   base

other attached packages:
[1] gRbase_1.7-0.1 Epi_1.1.68     foreign_0.8-61

loaded via a namespace (and not attached):
 [1] BiocGenerics_0.12.0 graph_1.44.0        grid_3.1.1          igraph_0.7.1
 [5] lattice_0.20-29     Matrix_1.1-4        parallel_3.1.1      RBGL_1.42.0
 [9] Rcpp_0.11.3         stats4_3.1.1        tools_3.1.1
```

4.1 Sample size calculations

4.1.1 Traditional calculations

Normally people have relied on various formulae or tabled when determining sample size in a study to obtain a given power to "detect an effect". The power of a study is the probability that the null hypotesis is rejected, so it depends on the null hypothesis, but in particular it depends on the alternative, or more precisely the scenario under which the data is gathered. So on how far from the null the true scenario is, measured by some sort of variability in data.

Hence in order to compute power or sample size it is necessary to specify the "truth".

4.1.2 Modern calculations

If you are able to specify the "true" scenario in which a given study will be conducted, it is also possible to *simulate* at dataset that will resemble the one that you anticipate to collect.

So the solution seems straigt forward: Simulate a dataset as you think it will be, analyze it and see what the result is. For example if the null is rejected or how wide the confidence limits are for the parameter (quantity of interest, e.g. the RR or slope of GFR by time or ...). One simulated dataset is of course prone to error (random variation), so normally you would simulate, say 1000 dataset and analyze then, extracting the results and summarizing them.

For eample for a given simulation scenario, the fraction of the analyses that reject the null is the power of the anticpated study under the conditions used for simulation.

Normally you would vary the conditions you use for simulation, notably the sample size for each simulated dataset, and presumably also the effect size. Both of these are need as input for simulation.

This machinery is illustrated in a samplesize calculation for a study to be conducted in Greenland.

4.1.3 The Greenland complications study

The Greenland complications study is an extra follow-up of the B99 and the IHIT cohorts, where persons with diabetes or impaired glucose regulation (IGR) are followed up for microvascular complications, notably nephropathy and retinopathy.

The number of persons in the IGR is about 1200 (not know till the 2014 survey is completed), with an estimated follow-up time of 7,940 person-years *after* onset of DM/IGR.

We want to estimate:

- rates of microvascular complications
- rate-ratios of microvascular complications between carriers of different genotypes of interest.

4.1.3.1 Assumptions

Based on previous studies and sound clinical judgment the rate of newly onset nephropathy among persons with DM or IGR is taken to be 20 per 1000 PY and the corresponding rate of retinopathy to be 30 per 1000 PY. Moreove it is likely that it will be possible to recruit persons so that the total ampunt of follow-up time will be about 8,000 PY.

4.1.4 Precision and power

4.1.5 Rates

A bold calculation yields that a complication with a constant rate of 20/1000 during 7,940 PY will produce $20 \times 7.94 = 159$ events, and one with rates 30/1000 will produce 238 events:

```
> c(20,30) * 7.94
  [1] 158.8 238.2
```

The usual formula for the relative precision of a rate estimate based on these numbers is:

$$\text{erf} = \exp(1.96/\sqrt{D})$$

which in the cases here will produce rate estimates with a relative precision in the vicinity of $\pm$ 15%:

```
> exp( 1.96/sqrt(c(159,238)) )
  [1] 1.168170 1.135471
```

This is the analysis part of the study, based on the formula for an estimated rate $\hat{\lambda}$ besed on D observed cases:

$$\text{s.d.}(\log(\hat{\lambda})) = \sqrt{1/D}$$

and for a rate-ratio based on D_1 and D_0 events we have:

$$\text{s.d.}(\log(\hat{\text{RR}})) = \text{s.d.}(\log(\hat{\lambda}_1/\hat{\lambda}_0)) = \sqrt{1/D_0 + 1/D_1}$$

4.1.6 Rate-ratios

IN the Greenland study we are looking at genotypes with population prevalences between 5 and 10% separately and 44% in total, but we do not have any specific estimates of the likely risk elevation associated each of them.

Hence we will compute the precision of the RR estimates between carriers and non-carriers using a baseline rates of 20 and 30 per 1000 PY, for exposure prevalences 5 – 50 %, and exposure effects 1.05, . . . , 1.50. This will slightly overestimate the precision if the true overall rates is 20 resp. 30, because a fraction of the population having an elevated risk will of course result in an overall rate that is higher than the rate in the unexposed group. For the sake of completeness we also make the computations for person-years between 8,000 and 13,000.

4.1.7 Simulation of one dataset

In this situation the simulation is particularly simple, because for a fixed amount of follow-up time (person-years), the number of observed events is Poisson-distributed with mean equal to the person-years times the rate. SO we can simulate the number of events in the exposed (the carrier group) and the number of events in the non-carrier group, and the compute the RR and the percisoon of it as follows:

Define the incidence rate, the person-years, RR and the prevalence and work out the rate times PY in each group, and simulate two Poisson varaites with these means:

```
> inc <- 20      # Incidence per 1000 PY
> RR  <- 1.5     # RR associated with carrier status
> pr  <- 10      # Prevalence (%) of carrier status
> py  <- 8       # Person-years in 1000s
```

So with these data we can simulate a munber of events in the two groups:

Person-years in the two groups:

```
> ( PY <- py*1000 * c( pr/100, 1-pr/100 ) )
  [1]  800 7200
```

Incidence rates in the two groups (per 1 person-year):

```
> ( IR <- c(RR,1) * inc/1000 )
  [1] 0.03 0.02
```

Incidence rate times person-years:

```
> PY*IR
  [1]  24 144
```

This is the *expected* number of events in the two groups, so we can *simulate* a studyoutcome of events in the two groups by generation two Poisson variates with these means, note we need to give the number of variate we simulate:

```
> ( D <- rpois( 2, lambda=PY*IR ) )
  [1]  25 124
```

The estimated rates are now:

```
> ( rt <- D / PY )
  [1] 0.03125000 0.01722222
```

and the rate-ratio is then:

```
> ( RR <- rt[1]/rt[2] )
  [1] 1.814516
```

and the error-factor (relative precision):

```
> exp( 1.96 * sqrt( sum(1/D) ) )
  [1] 1.536801
```

and the p-value is looking up the log-RR squared divided by the variance of the log-RR in a χ^2-distribution with 1 d.f.:

```
> ( chtst <- log(rt[2]/rt[1])^2 / ( 1/D[1] + 1/D[2] ) )
  [1] 7.385908
> ( pval <- 1 - pchisq( chtst, 1 ) )
  [1] 0.006573688
```

4.1.8 SImulating many datasets

But we would like to simulate many datasets, so let us first try to do 5 datasets with these parameters. In this situation it is better to simulate the outcome in each of the two groups in different vectors. We simulate `N` datasets:

```
> nsim <- 5
```

The first is the same as before; first the person-years in the two groups:

```
> ( PY <- py*1000 * c( 1-pr/100, pr/100 ) )
  [1] 7200  800
```

Incidence rates in the two groups (per 1 person-year):

```
> ( IR <- c(RR,1) * inc/1000 )
  [1] 0.03629032 0.02000000
```

Incidence rate times person-years:

```
> ( cuminc <- PY*IR )
  [1] 261.2903  16.0000
```

But now we simulate 5 copies of outcomes in the exposed, resp non-exposed:

```
> # Events in exposed:
> ( nx <- rpois( nsim, cuminc[1] ) )
  [1] 268 266 282 225 286
> # Events in unexposed:
> ( nr <- rpois( nsim, cuminc[2] ) )
  [1] 20 24  9 16 21
```

SO then we can easily compute the 5 RRs:

```
> ( rr <-  (nx/PY[1]) / (nr/PY[2]) )
  [1] 1.488889 1.231481 3.481481 1.562500 1.513228


> ( erf <- exp( 1.96*sqrt(1/nx+1/nr) ) )
  [1] 1.575115 1.518530 1.941920 1.660507 1.557577
```

We are not really interested in the p-value but only if the results is significant, which is when the log-RR is larger than the log-error-factor:

```
> ( sgn <- abs(log(rr)) > log(erf) )
  [1] FALSE FALSE  TRUE FALSE FALSE
```

4.1.9 Summarizing over many scenarios

We will do this exercise for say 1000 or 10,000 datasets but also in may different scenarios. Thus, we set up an array to collect the median RR (for control), the precision (in terms of the error factor, the relative uncertainty of the RR estimates) and the power for combinations of these factors.

An array in R is just a multi-way table, and the trick is to set up the array to hold for example power for different combinations of sample-size, RR and prevalence of exposure. Initially we set up an array exclusively with `NA`s in it and then we fill in the relevant numbers, based on simulated datasets:

```
> library( Epi )
> prpw <- NArray( list( RR = seq(1.05,1.5,0.05),
+                       pr = seq(5,50,5),
+                       PY = 8:13,
+                      inc = c(20,30),
+                     what = c("RR","Erf","Pwr") ) )
> str( prpw )
   logi [1:10, 1:10, 1:6, 1:2, 1:3] NA NA NA NA NA NA ...
   - attr(*, "dimnames")=List of 5
    ..$ RR  : chr [1:10] "1.05" "1.1" "1.15" "1.2" ...
    ..$ pr  : chr [1:10] "5" "10" "15" "20" ...
    ..$ PY  : chr [1:6] "8" "9" "10" "11" ...
    ..$ inc : chr [1:2] "20" "30"
    ..$ what: chr [1:3] "RR" "Erf" "Pwr"
> dim( prpw )
    RR   pr   PY  inc what
    10   10    6    2    3
> prod( dim( prpw ) )
  [1] 3600
> dimnames( prpw )
```

```
$RR
 [1] "1.05" "1.1"  "1.15" "1.2"  "1.25" "1.3"  "1.35" "1.4"  "1.45" "1.5"

$pr
 [1] "5"  "10" "15" "20" "25" "30" "35" "40" "45" "50"

$PY
[1] "8"  "9"  "10" "11" "12" "13"

$inc
[1] "20" "30"

$what
[1] "RR"  "Erf" "Pwr"
```

With this array to collect the results we can simulate, by approximating the no. events by a Poisson variate with mean equal to the cumulative rate.

The array is filled by making a set of nested `for`-loops so that inside we address one particular combination of RR (`RR`), prevalence (`pr`), person-years (`PY`) and incidence rates (`inc`).

Note that the loop-variables are character variables, so for the calculations we make numerical counterparts:

```
> nsim <- 10000
> system.time(
+ for( ir in dimnames(prpw)[["RR"]] )
+ for( ip in dimnames(prpw)[["pr"]] )
+ for( iy in dimnames(prpw)[["PY"]] )
+ for( ii in dimnames(prpw)[["inc"]] )
+ {
+ nr <- as.numeric(ir)
+ np <- as.numeric(ip)/100
+ ny <- as.numeric(iy)
+ ni <- as.numeric(ii)
+ # Cumulative incidence in exposed and non-exposed:
+ cuminc <- ny * c(np,1-np) * ni * c(nr,1)
+ # Events in exposed:
+ nx <- rpois( nsim, cuminc[1] )
+ # Events in unexposed:
+ nr <- rpois( nsim, cuminc[2] )
+ # Rate Ratio
+ rr <-  (nx/np) / (nr/(1-np))
+ # Error factor
+ erf <- exp( 1.96*sqrt(1/nx+1/nr) )
+ # Significance
+ sgn <- abs(log(rr)) > log(erf)
+ # Store results - not we use the *character* variables for indexing:
+ prpw[ir,ip,iy,ii,"RR" ] <- median( rr )
+ prpw[ir,ip,iy,ii,"Erf"] <- median( erf )
+ prpw[ir,ip,iy,ii,"Pwr"] <- mean( sgn )
+ } )
   user  system elapsed
   9.96    0.00   10.03
```

For the sake of verification, we print the median RRs from the simulations, hopefully its is approximately equal to the simulated scenario:

```
> prpw[,,"12",,"RR" ]
  , , inc = 20

      pr
RR         5       10       15       20       25       30       35       40       45
```

```
   1.05 1.036364 1.049327 1.051414 1.048128 1.050761 1.047009 1.048665 1.051020 1.048994
   1.1  1.090909 1.088372 1.096774 1.096774 1.096774 1.094650 1.095238 1.100000 1.101721
   1.15 1.136752 1.145455 1.149371 1.145729 1.148237 1.151804 1.149024 1.149635 1.145265
   1.2  1.192469 1.195776 1.195980 1.201878 1.194030 1.200780 1.198505 1.201923 1.202186
   1.25 1.239130 1.245536 1.246667 1.251185 1.251429 1.246212 1.250000 1.248322 1.249585
   1.3  1.282700 1.296296 1.299187 1.301320 1.301587 1.301146 1.297362 1.298077 1.301359
   1.35 1.337533 1.339535 1.345118 1.347826 1.349112 1.351587 1.350649 1.352113 1.349917
   1.4  1.384615 1.393805 1.395310 1.395833 1.397727 1.401209 1.401003 1.400301 1.403646
   1.45 1.448430 1.443396 1.444444 1.450777 1.447236 1.449704 1.449770 1.445652 1.450249
   1.5  1.486957 1.492891 1.498397 1.500000 1.500000 1.500938 1.498747 1.500000 1.502002
        pr
RR           50
  1.05 1.051724
  1.1  1.101695
  1.15 1.153846
  1.2  1.201681
  1.25 1.245832
  1.3  1.302752
  1.35 1.354839
  1.4  1.398305
  1.45 1.448819
  1.5  1.500000

, , inc = 30

        pr
RR           5        10        15        20        25        30        35        40        45
  1.05 1.046832 1.049080 1.052007 1.049282 1.046512 1.047953 1.047619 1.049678 1.051174
  1.1  1.091954 1.099010 1.097354 1.100324 1.099315 1.101611 1.098175 1.100000 1.098825
  1.15 1.146552 1.145963 1.147679 1.150327 1.147727 1.149254 1.151899 1.154412 1.150327
  1.2  1.191223 1.196532 1.199387 1.197279 1.197556 1.196802 1.200348 1.197309 1.198257
  1.25 1.239130 1.243243 1.249482 1.247458 1.250000 1.249651 1.249199 1.252358 1.250063
  1.3  1.290123 1.289902 1.299003 1.299639 1.295455 1.297297 1.300000 1.298611 1.301932
  1.35 1.342391 1.347305 1.348339 1.351171 1.349740 1.347942 1.352000 1.351485 1.351527
  1.4  1.394988 1.396040 1.393743 1.398601 1.397727 1.401741 1.402652 1.399038 1.404926
  1.45 1.440233 1.448916 1.448049 1.446050 1.449739 1.451444 1.450098 1.448718 1.445862
  1.5  1.485175 1.495385 1.496150 1.501706 1.500000 1.497375 1.497942 1.500000 1.502315
        pr
RR           50
  1.05 1.050562
  1.1  1.098901
  1.15 1.150838
  1.2  1.198830
  1.25 1.251366
  1.3  1.298644
  1.35 1.346821
  1.4  1.400000
  1.45 1.450000
  1.5  1.502793
```

As we see it does not print very nicely, but that can be remedied by using `ftable` (flat table):

```
> round( ftable( prpw[,,"12",,"RR" ], col.vars=1, row.vars=c(3,2) ), 2 )
         RR 1.05  1.1 1.15  1.2 1.25  1.3 1.35  1.4 1.45  1.5
inc pr
20  5     1.04 1.09 1.14 1.19 1.24 1.28 1.34 1.38 1.45 1.49
    10    1.05 1.09 1.15 1.20 1.25 1.30 1.34 1.39 1.44 1.49
    15    1.05 1.10 1.15 1.20 1.25 1.30 1.35 1.40 1.44 1.50
    20    1.05 1.10 1.15 1.20 1.25 1.30 1.35 1.40 1.45 1.50
    25    1.05 1.10 1.15 1.19 1.25 1.30 1.35 1.40 1.45 1.50
    30    1.05 1.09 1.15 1.20 1.25 1.30 1.35 1.40 1.45 1.50
    35    1.05 1.10 1.15 1.20 1.25 1.30 1.35 1.40 1.45 1.50
    40    1.05 1.10 1.15 1.20 1.25 1.30 1.35 1.40 1.45 1.50
    45    1.05 1.10 1.15 1.20 1.25 1.30 1.35 1.40 1.45 1.50
```

```
      50    1.05 1.10 1.15 1.20 1.25 1.30 1.35 1.40 1.45 1.50
  30  5     1.05 1.09 1.15 1.19 1.24 1.29 1.34 1.39 1.44 1.49
      10    1.05 1.10 1.15 1.20 1.24 1.29 1.35 1.40 1.45 1.50
      15    1.05 1.10 1.15 1.20 1.25 1.30 1.35 1.39 1.45 1.50
      20    1.05 1.10 1.15 1.20 1.25 1.30 1.35 1.40 1.45 1.50
      25    1.05 1.10 1.15 1.20 1.25 1.30 1.35 1.40 1.45 1.50
      30    1.05 1.10 1.15 1.20 1.25 1.30 1.35 1.40 1.45 1.50
      35    1.05 1.10 1.15 1.20 1.25 1.30 1.35 1.40 1.45 1.50
      40    1.05 1.10 1.15 1.20 1.25 1.30 1.35 1.40 1.45 1.50
      45    1.05 1.10 1.15 1.20 1.25 1.30 1.35 1.40 1.45 1.50
      50    1.05 1.10 1.15 1.20 1.25 1.30 1.35 1.40 1.45 1.50
```

Further more we see that the precision of the estimates (the error-factor) decreases only little with increasing RR, but much more so by increasing prevalence of the risk factor:

```
> round( ftable( prpw[,,c("8","12"),,"Erf"],
+                col.vars=1, row.vars=c(3,4,2) ), 2 )
            RR 1.05  1.1 1.15  1.2 1.25  1.3 1.35  1.4 1.45  1.5
  PY inc pr
  8  20  5     2.03 1.96 1.96 1.95 1.90 1.89 1.85 1.84 1.84 1.80
         10    1.66 1.65 1.63 1.61 1.60 1.58 1.58 1.56 1.55 1.54
         15    1.53 1.52 1.51 1.50 1.49 1.48 1.47 1.46 1.45 1.45
         20    1.46 1.45 1.44 1.44 1.43 1.42 1.41 1.41 1.40 1.39
         25    1.42 1.41 1.41 1.40 1.39 1.39 1.38 1.37 1.37 1.36
         30    1.40 1.39 1.38 1.38 1.37 1.36 1.36 1.35 1.35 1.35
         35    1.38 1.37 1.37 1.36 1.36 1.35 1.35 1.34 1.34 1.33
         40    1.37 1.36 1.36 1.35 1.35 1.34 1.34 1.33 1.33 1.33
         45    1.36 1.36 1.35 1.35 1.34 1.34 1.34 1.33 1.33 1.33
         50    1.36 1.35 1.35 1.35 1.34 1.34 1.34 1.33 1.33 1.33
     30  5     1.78 1.75 1.72 1.71 1.69 1.68 1.66 1.64 1.64 1.62
         10    1.51 1.50 1.49 1.48 1.47 1.46 1.45 1.44 1.43 1.42
         15    1.42 1.41 1.40 1.39 1.38 1.38 1.37 1.36 1.36 1.35
         20    1.36 1.36 1.35 1.34 1.34 1.33 1.33 1.32 1.32 1.31
         25    1.33 1.33 1.32 1.31 1.31 1.30 1.30 1.30 1.29 1.29
         30    1.31 1.31 1.30 1.30 1.29 1.29 1.28 1.28 1.28 1.27
         35    1.30 1.29 1.29 1.29 1.28 1.28 1.27 1.27 1.27 1.27
         40    1.29 1.29 1.28 1.28 1.27 1.27 1.27 1.27 1.26 1.26
         45    1.29 1.28 1.28 1.27 1.27 1.27 1.27 1.26 1.26 1.26
         50    1.28 1.28 1.28 1.27 1.27 1.27 1.27 1.26 1.26 1.26
  12 20  5     1.78 1.75 1.72 1.71 1.69 1.68 1.66 1.64 1.64 1.62
         10    1.51 1.50 1.49 1.48 1.47 1.46 1.45 1.44 1.43 1.42
         15    1.42 1.41 1.40 1.39 1.38 1.37 1.37 1.36 1.36 1.35
         20    1.36 1.36 1.35 1.34 1.34 1.33 1.33 1.32 1.32 1.31
         25    1.33 1.33 1.32 1.32 1.31 1.30 1.30 1.30 1.29 1.29
         30    1.31 1.31 1.30 1.30 1.29 1.29 1.28 1.28 1.28 1.27
         35    1.30 1.29 1.29 1.29 1.28 1.28 1.27 1.27 1.27 1.27
         40    1.29 1.29 1.28 1.28 1.27 1.27 1.27 1.27 1.26 1.26
         45    1.29 1.28 1.28 1.27 1.27 1.27 1.27 1.26 1.26 1.26
         50    1.28 1.28 1.28 1.27 1.27 1.27 1.27 1.26 1.26 1.26
     30  5     1.59 1.57 1.56 1.55 1.54 1.52 1.51 1.50 1.49 1.48
         10    1.40 1.39 1.38 1.37 1.37 1.36 1.35 1.35 1.34 1.33
         15    1.33 1.32 1.31 1.31 1.30 1.30 1.29 1.29 1.28 1.28
         20    1.29 1.28 1.28 1.27 1.27 1.26 1.26 1.26 1.25 1.25
         25    1.26 1.26 1.25 1.25 1.25 1.24 1.24 1.24 1.23 1.23
         30    1.25 1.24 1.24 1.24 1.23 1.23 1.23 1.22 1.22 1.22
         35    1.24 1.23 1.23 1.23 1.22 1.22 1.22 1.22 1.21 1.21
         40    1.23 1.23 1.22 1.22 1.22 1.22 1.21 1.21 1.21 1.21
         45    1.23 1.22 1.22 1.22 1.22 1.21 1.21 1.21 1.21 1.21
         50    1.23 1.22 1.22 1.22 1.22 1.21 1.21 1.21 1.21 1.21
```

We can then print the anticipated power to detect a significant RR of a given size, in percent:

```
> round( ftable( prpw[,,c("8","12"),,"Pwr"]*100,
+                col.vars=1, row.vars=c(3,4,2) ), 1 )
```

```
           RR 1.05  1.1 1.15  1.2 1.25  1.3 1.35  1.4 1.45  1.5
PY inc pr
8  20  5      4.5  6.2  8.0  9.9 12.0 15.2 18.1 21.7 25.4 29.0
       10     5.3  7.2  9.5 13.6 17.1 22.4 26.3 32.0 38.5 44.9
       15     6.3  8.0 10.6 15.6 21.6 27.0 34.4 41.5 48.7 56.6
       20     6.4  8.9 12.9 17.3 24.5 31.5 40.7 48.8 57.3 64.2
       25     5.9  9.5 13.3 19.4 27.1 36.3 44.9 53.7 63.3 69.8
       30     6.2  8.8 13.4 19.7 29.3 37.8 47.0 57.1 67.3 74.4
       35     6.3  9.2 14.1 21.8 31.4 40.4 50.9 60.9 70.2 79.0
       40     5.8  9.3 14.8 22.1 31.4 41.7 52.2 62.9 72.2 80.3
       45     5.8 10.2 15.2 22.8 32.0 42.1 53.4 63.5 73.3 81.0
       50     6.1  9.5 14.3 21.8 31.6 41.8 53.7 63.6 73.2 81.0
   30  5      5.7  7.0  9.4 11.6 15.7 19.4 23.6 28.0 32.6 37.4
       10     5.9  8.9 11.3 16.2 22.4 28.6 36.3 44.3 51.9 59.1
       15     6.2  9.8 13.4 20.4 28.0 36.8 46.4 55.8 64.8 72.4
       20     6.5  9.8 15.1 23.5 33.5 43.3 55.0 64.0 73.2 80.9
       25     7.0 10.7 17.7 25.8 37.2 48.8 60.1 71.6 79.3 85.8
       30     6.8 10.9 18.4 28.6 40.4 53.3 63.7 74.5 83.6 89.7
       35     7.2 11.5 20.0 29.4 41.9 55.8 66.8 77.2 86.3 91.5
       40     6.8 11.6 19.4 31.8 43.9 57.4 68.9 80.0 87.6 92.9
       45     7.2 11.5 20.0 31.8 44.9 58.8 70.6 80.7 88.4 93.5
       50     6.8 11.5 19.4 31.4 43.9 58.8 71.3 80.5 88.7 93.7
12 20  5      5.4  7.0  9.6 11.8 15.3 18.7 23.6 27.6 34.6 38.9
       10     6.3  8.0 11.8 16.4 22.0 29.1 35.9 44.2 51.6 59.2
       15     6.3  9.5 14.1 20.7 27.6 37.5 46.4 55.7 64.6 73.0
       20     6.1  9.6 15.6 23.8 33.7 44.3 54.4 64.2 74.2 81.2
       25     6.9 10.4 18.1 25.7 37.5 49.4 60.1 70.4 79.2 86.4
       30     6.5 11.2 18.7 28.6 39.8 53.2 65.3 75.3 83.4 89.5
       35     6.5 10.9 19.0 29.6 42.5 54.9 67.5 78.6 85.4 91.1
       40     6.7 11.4 20.4 31.2 43.4 56.5 70.2 79.9 87.3 93.3
       45     7.1 11.9 19.6 31.1 44.4 58.5 70.5 81.4 88.1 93.5
       50     6.8 11.9 19.8 31.7 43.5 58.9 71.8 80.0 88.3 93.3
   30  5      5.3  7.7 11.1 14.9 19.6 25.2 30.7 38.0 44.1 50.4
       10     6.0  9.8 14.5 21.8 29.3 38.5 49.2 58.4 67.9 74.8
       15     6.4 11.0 17.5 27.7 38.8 50.5 61.6 72.4 80.7 87.1
       20     7.0 12.7 21.4 32.2 45.0 59.2 71.2 80.8 87.8 93.2
       25     6.9 13.1 23.5 35.7 51.1 64.1 78.0 86.2 92.0 96.0
       30     7.5 14.9 25.0 38.5 55.0 68.7 81.8 89.5 94.4 97.6
       35     7.3 14.8 27.2 41.8 58.0 72.1 84.0 91.5 96.1 98.1
       40     7.8 15.0 28.6 42.2 61.0 73.5 85.6 92.6 96.7 98.6
       45     7.4 14.6 27.6 43.3 60.4 75.1 86.6 93.4 97.1 98.9
       50     8.1 15.5 28.3 43.2 61.1 75.7 86.3 93.2 97.2 98.9
```

And show where the power exceeds 80%; if we explicitly use the `print` function we can use the `zero.print` argument:

```
> print( ftable( (prpw[,,c("8","12"),,"Pwr"]>0.7)*1,
+                col.vars=1, row.vars=c(3,4,2) ),
+        zero.print="." )
           RR 1.05 1.1 1.15 1.2 1.25 1.3 1.35 1.4 1.45 1.5
PY inc pr
8  20  5        .   .    .   .    .   .    .   .    .   .
       10       .   .    .   .    .   .    .   .    .   .
       15       .   .    .   .    .   .    .   .    .   .
       20       .   .    .   .    .   .    .   .    .   .
       25       .   .    .   .    .   .    .   .    .   .
       30       .   .    .   .    .   .    .   .    .   1
       35       .   .    .   .    .   .    .   .    1   1
       40       .   .    .   .    .   .    .   .    1   1
       45       .   .    .   .    .   .    .   .    1   1
       50       .   .    .   .    .   .    .   .    1   1
   30  5        .   .    .   .    .   .    .   .    .   .
       10       .   .    .   .    .   .    .   .    .   .
       15       .   .    .   .    .   .    .   .    .   1
       20       .   .    .   .    .   .    .   .    1   1
```

```
          25      .  .  .  .  .  .  .  1  1  1
          30      .  .  .  .  .  .  .  1  1  1
          35      .  .  .  .  .  .  .  1  1  1
          40      .  .  .  .  .  .  .  1  1  1
          45      .  .  .  .  .  .  1  1  1  1
          50      .  .  .  .  .  .  1  1  1  1
12 20     5       .  .  .  .  .  .  .  .  .  .
          10      .  .  .  .  .  .  .  .  .  .
          15      .  .  .  .  .  .  .  .  .  1
          20      .  .  .  .  .  .  .  .  1  1
          25      .  .  .  .  .  .  .  1  1  1
          30      .  .  .  .  .  .  .  1  1  1
          35      .  .  .  .  .  .  .  1  1  1
          40      .  .  .  .  .  .  1  1  1  1
          45      .  .  .  .  .  .  1  1  1  1
          50      .  .  .  .  .  .  1  1  1  1
    30    5       .  .  .  .  .  .  .  .  .  .
          10      .  .  .  .  .  .  .  .  .  1
          15      .  .  .  .  .  .  .  1  1  1
          20      .  .  .  .  .  .  1  1  1  1
          25      .  .  .  .  .  .  1  1  1  1
          30      .  .  .  .  .  .  1  1  1  1
          35      .  .  .  .  .  1  1  1  1  1
          40      .  .  .  .  .  1  1  1  1  1
          45      .  .  .  .  .  1  1  1  1  1
          50      .  .  .  .  .  1  1  1  1  1
```

Broadly speaking, with 80% power we can only see effects of RR> 1.4 for exposures with prevalence over 30%, but the good news is that with 12,000 PY we have a relative precision of 1.3 and with 8,000 PY a relative precision in the vicinity of 1.5.