Statistical Practice in Epidemiology with R

SPE practicals

Krista Fischer[1] Martyn Plummer[2] Janne Pitkäniemi[3] Bendix Carstensen[4]
Damien Georges[5] Esa Läärä[6]

3-7 June 2024

[1]Estonian Genome Center, University of Tartu, Estonia, Krista.Fischer@ut.ee
[2]University of Warwick, UK, martyn.plummer@warwick.ac.uk
[3]Finnish Cancer Registry, Helsinki, Finland, janne.pitkaniemi@cancer.fi
[4]Steno Diabetes Center Copenhagen, Gentofte, Denmark, Dept. of Biostatistics, University of Copenhagen, Denmark, b@bxc.dk, http://BendixCarstensen.com
[5]International Agency for Research on Cancer, Lyon, France, georgesd@iarc.who.int
[6]Department of Mathematical Sciences, University of Oulu, Finland, Esa.Laara@oulu.fi, http://www.oulu.fi/university/researcher/esa-laara

Contents

About

This booklet contains a set of exercises to apply concepts learned during the SPE-R course.

Chapter 1

Practice with basic R

The main purpose of this session is to give participants who have not had much (or any) experience with using `R` a chance to practice the basics and to ask questions. For others, it should serve as a reminder of some of the basic features of the language.

R can be installed on all major platforms (*i.e.* Windows, macOS, Linux). We do not assume in this exercise that you are using any particular platform. Many people like to use the RStudio graphical user interface (GUI), which gives the same look and feel across all platforms.

1.1 The working directory

A key concept in `R` is the *working directory* (or *folder* in the terminology of Windows). The working directory is where `R` looks for data files that you may want to read in and where `R` will write out any files you create. It is a good idea to keep separate working directories for different projects. In this course we recommend that you keep a separate working directory for each exercise.

If you are working on the command line in a terminal, then you can change to the correct working directory and then launch `R` by typing *R*.

If you are using a GUI then you will typically need to change to the correct working directory after starting `R`. In RStudio, you can change directory from the *Session* menu. However it is much more useful to create a new *project* to keep your source files and data. When you open a project in the RStudio GUI, your working directory is automatically changed to the directory associated with the project.

You can display the current working directory with the `getwd()` (*get working directory*) function and set it with the `setwd()` (*set working directory*) function. The function `dir()` can be used to see what files you have in the working directory.

1.2 The workspace

You can quit `R` by typing

```
q()
```

at the `R` command prompt. You will be asked if you want to save your workspace. We strongly recommend that you answer *no* to this question. If you answer *yes* then R will write a file named `.RData` into the working directory containing all the objects you created during your session. This file will be automatically loaded the next time you start `R` and this will restore all the objects in your workspace.

It may seem convenient to keep your R objects from one session to another. But this has many disadvantages.

- You may not remember how an object was created. This becomes a problem if you need to redo your analysis after the data has been changed or updated, or if you accidentally delete the object.

- An object might be modified or overwritten. In this case your analysis will give you a different answer but you will not know why. You may not even notice that the answer has changed.
- It becomes impossible to clean your workspace if you cannot remember which objects are required by which analyses. As a result, your workspace will become cluttered with old objects.

We strongly recommend that you follow some basic principles of reproducible research.

- Always start with a clean (empty) workspace.
- Read in the data you need from a pristine source.
- Put your R commands in a script file so that they can be run again in a future session.
- All modifications to the data that you need to make should be done in R using a script and not by editing the data source. This way, if the original data is modified or updated then you can run the script again on the updated data.

1.3 Using R as a calculator

Try using R as an interactive calculator by typing different arithmetic expressions on the command line. Pressing the return key on the command line finishes the expression. R will then evaluate the expression and print out the result.

Note that R allows you to recall previous commands using the vertical arrow key. You can edit a recalled command and then resubmit it by pressing the return key. Keeping that in mind, try the following:

```
12 + 16
(12 + 16) * 5
sqrt((12 + 16) * 5) # square root
round(sqrt((12 + 16) * 5), 2) # round to two decimal places
```

The hash symbol `#` denotes the start of a *comment*. Anything after the hash is ignored by R.

Round braces are used a lot in R. In the above expressions, they are used in two different ways. Firstly, they can be used to establish the order of operations. In the example

```
(12 + 16) * 5
```

they ensure that 12 is added to 16 before the result is multiplied by 5. If you omit the braces then you get a different answer

```
12 + 16 * 5
```

because multiplication has higher *precedence* than addition. The second use of round braces is in a function call (e.g. `sqrt`, `round`). To call a function in R, type the name followed by the arguments inside round braces. Some functions take multiple arguments, and in this case they are separated by commas.

You can see that complicated expressions in R can have several levels of nested braces. To keep track of these, it helps to use a syntax-highlighting editor. For example, in RStudio, when you type an opening bracket `(`, RStudio will automatically add a closing bracket `)`, and when the cursor moves past a closing bracket, RStudio will automatically highlight the corresponding opening bracket. Features like this can make it much easier to write R code free from syntax errors.

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

```
a <- round(sqrt((12 + 16) * 5), 2)
```

In this case `R` does not print anything to the screen. You can see the results of the calculation, stored in the object `a`, by typing `a` and also use `a` for further calculations, e.g:

```
exp(a)
log(a) # natural logarithm
log10(a) # log to the base 10
```

The left arrow expression `<-`, pronounced *gets*, is called the assignment operator, and is obtained by typing `<` followed by `-` (with no space in between). It is also possible to use the equals sign `=` for assignment.

Note that object names in R are case sensitive. So you can assign different values to objects named `A` and `a`.

1.4 Vectors

All commands in R are *functions* which act on *objects*. One important kind of object is a *vector*, which is an ordered collection of numbers, or character strings (e.g. *Charles Darwin*), or logical values (`TRUE` or `FALSE`). The components of a vector must be of the same type (numeric, character, or logical). 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 and assigns the result to the vector `v`.

A key feature of the R language is that many operations are *vectorized*, meaning that you can carry out the same operation on each element of a vector in a single operation. Try

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

and you will see that R understands what to do in each case.

R extends ordinary arithmetic with the concept of a *missing value* represented by the symbol `NA` (*Not Available*). Any operation on a missing value creates another missing value. You can see this by repeating the same operations on a vector containing a missing value:

```
v <- c(4, 6, NA)
3 + v
3 * v
```

The fact that every operation on a missing value produces a missing value can be a nuisance when you want to create a summary statistic for a vector:

```
mean(v)
```

While it is true that the mean of `v` is unknown because the value of the third element is missing, we normally want the mean of the non-missing elements. Fortunately the `mean` function has an optional argument called `na.rm` which can be used for this.

```
mean(v, na.rm = TRUE)
```

Many functions in R have optional arguments that can be omitted, in which case they take their default value (For example, the `mean` function has default `na.rm=FALSE`). You can explicitly values to optional arguments in the function call to override the default behaviour.

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. If you just want to know the length of a vector then it is much easier to use the `length` function.

```
length(v)
```

1.5 Sequences

There are short-cut functions for creating vectors with a regular structure. For example, if you want a vector containing the sequence of integers from 1 to 10, you can use

```
1:10
```

The `seq()` function allows the creation of more general sequences. For example, the vector (15, 20, 25, … ,85) can be created with

```
seq(from = 15, to = 85, by = 5)
```

The objects created by the `:` operator and the `seq()` function are ordinary vectors, and can be combined with other vectors using the combine function:

```
c(5, seq(from = 20, to = 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()`.

1. Create a vector `w` with components 1, -1, 2, -2
2. Display this vector
3. Obtain a description of `w` using `str()`
4. Create the vector `w+1`, and display it.
5. Create the vector `v` with components (5, 10, 15, … , 75) using seq().
6. Now add the components 0 and 1 to the beginning of `v` using c().
7. Find the length of this vector.

1.6 Displaying and changing parts of a vector (indexing)

Square brackets in R are used to extract parts of vectors. So `x[1]` gives the first element of vector `x`. Since R is vectorized you can also supply a vector of integer index values inside the square brackets. Any expression that creates an integer vector will work.

Try the following commands:

```
x <- c(2, 7, 0, 9, 10, 23, 11, 4, 7, 8, 6, 0)
x[4]
x[3:5]
x[c(1, 5, 8)]
```

Trying to extract an element that is beyond the end of the vector is, surprisingly, not an error. Instead, this returns a missing value

```
N <- length(x)
x[N + 1]
```

There is a reason for this behaviour, which we will discuss in the recap.

`R` also allows *logical subscripting*. Try the following

```
x > 10
x[x > 10]
```

The first expression creates a logical vector of the same length as `x`, where each element has the value `TRUE` or `FALSE` depending on whether or not the corresponding element of `x` is greater than 10. If you supply a logical vector as an index, `R` selects only those elements for which the conditions is `TRUE`.

You can combine two logical vectors with the operators `&` (logical and) and `|` (logical or). For example, to select elements of x that are between 10 and 20 we combine two one-sided logical conditions for $x \geq 10$ *and* $x \leq 20$:

```
x[x >= 10 & x <= 20]
```

The remaining elements of `x` that are *either* less than 10 *or* greater than 20 are selected with

```
x[x < 10 | x > 20]
```

Indexing can also be used to replace parts of a vector:

```
x[1] <- 1000
x
```

This replaces the first element of `x`. Logical subscripting is useful for replacing parts of a vector that satisfy a certain condition. For example to replace all elements that take the value 0 with the value 1:

```
x[x == 0] <- 1
x
```

If you want to replace parts of a vector then you need to make sure that the replacement value is either a single value, as in the example above, or a vector equal in length to the number of elements to be replaced. For example, to replace elements 2, 3, and 4 we need to supply a vector of replacement values of length 3.

```
x[2:4] <- c(0, 8, 1)
x
```

It is important to remember this when you are using logical subscripting because the number of elements to be replaced is not given explicitly in the R code, and it is easy to get confused about how many values need to be replaced. If we want to add 3 to every element that is less than 3 then we can break the operation down into 3 steps:

```
y <- x[x < 3]
y <- y + 3
x[x < 3] <- y
x
```

First we extract the values to be modified, then we modify them, then we write back the modified values to the original positions. R experts will normally do this in a single expression.

```
x[x < 3] <- x[x < 3] + 3
```

Remember, if you are confused by a complicated expression you can usually break it down into simpler steps.

If you want to create an entirely new vector based on some logical condition then use the `ifelse()` function. This function takes three arguments: the first is a logical vector; the second is the value taken by elements of the logical vector that are `TRUE`; and the third is the value taken by elements that are `FALSE`.

In this example, we use the remainder operator `%%` to identify elements of `x` that have value 0 when divided by 2 (i.e. the even numbers) and then create a new character vector with the labels *even* and *odd*:

```
x %% 2
ifelse(x %% 2 == 0, "even", "odd")
```

Now try the following:

1. Display elements that are less than 10, but greater than 4
2. Modify the vector x, replacing by 10 all values that are greater than 10
3. Modify the vector x, multiplying by 2 all elements that are smaller than 5 (Remember you can do this in steps).

1.7 Lists

Collections of components of different types are called *lists*, and are created with the `list()` function. Thus

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

creates a list with 3 components: the first is numeric, the second is logical and the third is character. A list element can be any object, including another list. This flexibility means that functions that need to return a lot of complex information, such as statistical modelling functions, often return a list.

As with vectors, single square brackets are used to take a subset of a list, but the result will always be another list, even if you select only one element

```
m[1:2] # A list containing first two elements of m
m[3] # A list containing the third element of m
```

If you just want to extract a single element of a list then you must use double square braces:

```
m[[3]] # Extract third element
```

Lists are more useful when their elements are named. You can name an element by using the syntax `name=value` in the call to the `list` function:

```
mylist <- list(
  name = c("Joe", "Ann", "Jack", "Tom"),
  age = c(34, 50, 27, 42)
)
mylist
```

This creates a new list with the elements *name*, a character vector of names, and *age* a numeric vector of ages. The components of the list can be extracted with a dollar sign `$`

```
mylist$name
mylist$age
```

1.8 Data frames

Data frames are a special structure used when we want to store several vectors of the same length, and corresponding elements of each vector refer to the same record. For example, here we create a simple data frame containing the names of some individuals along with their age in years, their sex (coded 1 or 2) and their height in cm.

```
mydata <- data.frame(
  name = c("Joe", "Ann", "Jack", "Tom"),
  age = c(34, 50, 27, 42),
  sex = c(1, 2, 1, 1),
  height = c(185, 170, 175, 182)
)
```

The construction of a data frame is just like a named list (except that we use the constructor function `data.frame` instead of `list`). In fact data frames are also lists so, for example, you can extract vectors using the dollar sign:

```
mydata$height
```

On the other hand, data frames are also two dimensional objects:

```
mydata
```

When you print a data frame, each variable appears in a separate column. You can use square brackets with two comma-separated arguments to take subsets of rows or columns.

```
mydata[1, ]
mydata[, c("age", "height")]
mydata[2, 4]
```

We will look into indexing of data frames in more detail below.

Now let's create another data frame with more individuals than the first one:

```
yourdata <- data.frame(
  name = c("Ann", "Peter", "Sue", "Jack", "Tom", "Joe", "Jane"),
  weight = c(67, 81, 56, 90, 72, 79, 69)
)
```

This new data frame contains the weights of the individuals. The two data sets can be joined together with the `merge` function.

```
newdata <- merge(mydata, yourdata)
newdata
```

The `merge` function uses the variables common to both data frames – in this case the variable *name* – to uniquely identify each row. By default, only rows that are in both data frames are preserved, the rest are discarded. In the above example, the records for Peter, Sue, and Jane, which are not in `mydata` are discarded. If you want to keep them, use the optional argument `all=TRUE`.

```
newdata <- merge(mydata, yourdata, all = TRUE)
newdata
```

This keeps a row for all individuals but since Peter, Sue and Jane have no recorded age, height, or sex these are missing values.

1.9 Working with built-in data frames

We shall use the births data which concern 500 mothers who had singleton births (i.e. no twins) in a large London hospital. The outcome of interest is the birth weight of the baby, also dichotomised as normal or low birth weight. These data are available in the Epi package:

```
library(Epi)
data(births)
objects()
```

The function `objects()` shows what is in your workspace. To find out a bit more about `births` try

```
help(births)
```

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

1. 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 workspace. 3. Get help on the object `diet`. 4. Remove the object `diet` with the command

```
remove(diet)
```

5. Check that the object `diet` is no longer in your workspace.

1.10 Referencing parts of the data frame (indexing)

Typing `births` will list the entire data frame – not usually very helpful. You can use the `head` function to see just the first few rows of a data frame

```
head(births)
```

Now try

```
births[1, ]
```

This will list all the values for the first row. Similarly,

```
births[2, ]
```

will list the value taken by the second row, and so on. To list the data for the first 10 subjects, try

```
births[1:10, ]
```

Often we want to extract rows of a data frame based on a condition. To select all subjects with height less than 180 cm from the data frame `mydata` we can use the `subset()` function.

```
subset(mydata, height < 180)
```

1.11 Summaries

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

```
summary(births)
```

This prints some summary statistics (minimum, lower quartile, mean, median, upper quartile, maximum). For variables with missing values, the number of `NA`s is also printed.

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)
```

Alternatively you can use

```
with(births, summary(hyp))
```

1.12 Generating new variables

New variables can be produced using assignment together with the usual mathematical operations and functions. For example

```
logbw <- log(births$bweight)
```

produces the variable `logbw` in your workspace, while

```
births$logbw <- log(births$bweight)
```

produces the variable `logbw` in the `births` data frame.

You can also replace existing variables. For example `bweight` measures birth weight in grams. To convert the units to kilograms we replace the original variable with a new one:

```
births$bweight <- births$bweight / 1000
```

1.13 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. Factors will become very important later in the course when we study modelling functions, where factors and numeric variables are treated very differently. For the moment, you can think of factors as *value labels* that are more informative than numeric codes.

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

```
births$hyp <- factor(births$hyp, labels = c("normal", "hyper"))
```

This takes the original numeric codes (0, 1) and replaces them with informative labels *normal'' and "hyper* for normal blood pressure and hypertension, respectively.

1. Convert the variable `sex` into a factor with labels `"M"` and `"F"` for values 1 and 2, respectively

1.14 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 simple frequency tables is `table`. The distribution of the factor `hyp` can be viewed using

```
with(births, table(hyp))
```

or by specifying the data frame as in

```
table(births$hyp)
```

For simple expressions the choice is a matter of taste, but `with` is shorter for more complicated expressions.

1. Find the frequency distribution of `sex`.
2. If you give two or more arguments to the `table` function then it produces cross-tabulations. Find the two-way frequency distribution of `sex` and `hyp`.
3. Create a logical variable called `early` according to whether `gestwks` is less than 30 or not. Make a frequency table of `early`.

1.15 Grouping the values of a numeric 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$agegrp <-
  cut(
    births$matage,
    breaks = c(20, 30, 35, 40, 45),
    right = FALSE
  )
with(births, table(agegrp))
```

By default the factor levels are labelled `[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.

Observations which are not inside the range specified by the `breaks` argument result in missing values for the new factor. Hence it is important that the first element in `breaks` is smaller than the smallest value in your data, and the last element is larger than the largest value.

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`.

Chapter 2

Reading data into R

2.1 Introduction

> *If you want to have rabbit stew, first catch the rabbit* – Old saying, origin unknown

R is a language and environment for data analysis. If you want to do something interesting with it, you need data.

For teaching purposes, data sets are often embedded in R packages. The base R distribution contains a whole package dedicated to data which includes around 100 data sets. This is attached towards the end of the search path, and you can see its contents with

```
objects("package:datasets")
```

A description of all of these objects is available using the `help()` function. For example

```
help(Titanic)
```

gives an explanation of the `Titanic` data set, along with references giving the source of the data.

The `Epi` package also contains some data sets. These are not available automatically when you load the `Epi` package, but you can make a copy in your workspace using the `data()` function. For example

```
library(Epi)
data(bdendo)
```

will create a data frame called `bdendo` in your workspace containing data from a case-control study of endometrial cancer. Datasets in the `Epi` package also have help pages: type `help(bdendo)` for further information.

To go back to the cooking analogy, these data sets are the equivalent of microwave ready meals, carefully packaged and requiring minimal work by the consumer. Your own data will never be able in this form and you must work harder to read it in to R.

This exercise introduces you to the basics of reading external data into R. It consists of reading the same data from different formats. Although this may appear repetitive, it allows you to see the many options available to you, and should allow you to recognize when things go wrong.

> **getting the data** You will need to download the zip file `data.zip` from the course web site (https://github.com/SPE-R/SPE/raw/gh-spe-material/SPE-all-material.zip) and unpack this in your working directory. This will create a sub-directory `data` containing (among other things) the files `fem.dat`, `fem-dot.dat`, `fem.csv`, and `fem.dta` (Reminder: use `setwd()` to set your working directory).

2.2 Data sources

Sources of data can be classified into three groups:

- Data in human readable form, which can be inspected with a text editor.
- Data in binary format, which can only be read by a program that understands that format (SAS, SPSS, Stata, Excel, …).
- Online data from a database management system (DBMS)

This exercise will deal with the first two forms of data. Epidemiological data sets are rarely large enough to justify being kept in a DBMS. If you want further details on this topic, you can consult the *R Data Import/Export* manual that comes with `R`.

2.3 Data in text files

Human-readable data files are generally kept in a rectangular format, with individual records in single rows and variables in columns. Such data can be read into a data frame in `R`.

Before reading in the data, you should inspect the file in a text editor and ask three questions:

- How are columns in the table separated?
- How are missing values represented?
- Are variable names included in the file?

The file `fem.dat` contains data on 118 female psychiatric patients. The data set contains nine variables.

Name	Description
ID	Patient identifier
AGE	Age in years
IQ	Intelligence Quotient (IQ) score
ANXIETY	Anxiety (1=none, 2=mild, 3=moderate,4=severe)
DEPRESS	Depression (1=none, 2=mild, 3=moderate or severe)
SLEEP	Sleeping normally (1=yes, 2=no)
SEX	Lost interest in sex (1=yes, 2=no)
LIFE	Considered suicide (1=yes, 2=no)
WEIGHT	Weight change (kg) in previous 6 months

Inspect the file `fem.dat` with a text editor to answer the questions above.

The most general function for reading in free-format data is `read.table()`. This function reads a text file and returns a data frame. It tries to guess the correct format of each variable in the data frame (integer, double precision, or text).

Read in the table with:

```
fem <- read.table("./data/fem.dat", header = TRUE)
```

Note that you must assign the result of `read.table()` to an object. If this is not done, the data frame will be printed to the screen and then lost.

You can see the names of the variables with

```
names(fem)
```

The structure of the data frame can be seen with

```
str(fem)
```

You can also inspect the top few rows with

```
head(fem)
```

Note that the IQ of subject 9 is -99, which is an illegal value: nobody can have a negative IQ. In fact -99 has been used in this file to represent a missing value. In `R` the special value `NA` (*Not Available*) is used to represent missing values. All `R` functions recognize `NA` values and will handle them appropriately, although sometimes the appropriate response is to stop the calculation with an error message.

You can recode the missing values with

```
fem$IQ[fem$IQ == -99] <- NA
```

Of course it is much better to handle special missing value codes when reading in the data. This can be done with the `na.strings` argument of the `read.table()` function. See below.

2.4 Things that can go wrong

Sooner or later when reading data into `R`, you will make a mistake. The frustrating part of reading data into `R` is that most mistakes are not fatal: they simply cause the function to return a data frame that is *not what you wanted.* There are three common mistakes, which you should learn to recognize.

2.5 Forgetting the headers

The first row of the file `fem.dat` contains the variable names. The `read.table()` function does not assume this by default so you have to specify this with the argument `header=TRUE`. See what happens when you forget to include this option:

```
fem2 <- read.table("data/fem.dat")
str(fem2)
head(fem2)
```

and compare the resulting data frame with `fem`. - What are the names of the variables in the data frame? - What is the class of the variables?

> **Explanation:** Remember that `read.table()` tries to guess the mode of the variables in the text file. Without the `header = TRUE` option it reads the first row, containing the variable names, as data, and guesses that all the variables are character, not numeric. By default, all character variables are coerced to factors by `read.table`. The result is a data frame consisting entirely of factors. (You can prevent the conversion of character variables to factors with the argument `as.is = TRUE`).

If the variable names are not specified in the file, then they are given default names `V1`, `V2`, … . You will soon realise this mistake if you try to access a variable in the data frame by, for example

```
fem2$IQ
```

as the variable will not exist

There is one case where omitting the `header = TRUE` option is harmless (apart from the situation where there is no header line, obviously). When the first row of the file contains **one less** value than subsequent lines, `read.table()` infers that the first row contains the variable names, and the first column of every subsequent row contains its **row name**.

2.6 Using the wrong separator

By default, `read.table` assumes that data values are separated by any amount of white space. Other possibilities can be specified using the `sep` argument. See what happens when you assume the wrong separator, in this case a tab, which is specified using the escape sequence `"\t"`

```
fem3 <- read.table("data/fem.dat", sep = "\t")
str(fem3)
```

- How many variables are there in the data set?

  Explanation: If you mis-specify the separator, `read.table()` reads the whole line as a single character variable. Once again, character variables are coerced to factors, so you get a data frame with a single factor variable.

2.7 Mis-specifying the representation of missing values

The file `fem-dot.dat` contains a version of the FEM dataset in which all missing values are represented with a dot. This is a common way of representing missing values, but is not recognized by default by the `read.table()` function, which assumes that missing values are represented by *NA*.

Inspect the file with a text editor, and then see what happens when you read the file in incorrectly:

```
fem4 <- read.table("data/fem-dot.dat", header = TRUE)
str(fem4)
```

You should have enough clues by now to work out what went wrong.

You can read the data correctly using the `na.strings` argument

```
fem4 <-
  read.table(
    "data/fem-dot.dat",
    header = TRUE,
    na.strings = "."
  )
```

2.8 Spreadsheet data

Spreadsheets have become a common way of exchanging data. All spreadsheet programs can save a single sheet in *comma-separated variable* (CSV) format, which can then be read into `R`. There are two functions in `R` for reading in CSV data: `read.csv()` and `read.csv2()`.

Both of these are wrappers around the `read.table()` function, *i.e.* the `read.table()` function is still doing the work of reading in the data but the `read.csv()` function provides default argument values for reading in CSV file so all you need to do is specify the file name.

You can see what these default arguments are with the `args()` function.

```
args(read.csv)
args(read.csv2)
```

See if you can spot the difference between `read.csv` and `read.csv2`.

> **Explanation:** The CSV format is not a single standard. The file format depends on the *locale* of your computer – the settings that determine how numbers are represented. In some countries, the decimal separator is a point `.` and the variable separator in a CSV file is a comma `,`. In other countries, the decimal separator is a comma `,` and the variable separator is a semi-colon `;`. This is reflected in the different default values for the arguments `sep` and `dec`. The `read.csv()` function is used for the first format and the `read.csv2()` function is used for the second format.

The file `fem.csv` contains the FEM dataset in CSV format. Inspect the file to work out which format is used, and read it into `R`.

2.9 Reading data from the Internet

You can also read in data from a remote web site. The `file` argument of `read.table()` does not need to be a local file on your computer; it can be a Uniform Resource Locator (URL), *i.e.* a web address.

A copy of the file `fem.dat` is held at (https://www.bendixcarstensen.com/SPE/data/fem.dat). You can read it in with

```
fem6 <-
  read.table(
    "http://www.bendixcarstensen.com/SPE/data/fem.dat",
    header = TRUE
```

```
  )
str(fem6)
```

2.10 Reading from the clipboard

On Microsoft Windows, you can copy values directly from an open Excel spreadsheet using the clipboard. Highlight the cells you want to copy in the spread sheet and select copy from the pull-down edit menu. Then type `read.table(file = "clipboard")` to read the data in.

There are two reasons why this is a bad idea

- It is not reproducible. In order to read the data in again you need to complete exactly the same sequence of mouse moves and clicks, and there is no record of what you did before.
- Copying from the clipboard loses precision. If you have a value `1.23456789` in your spreadsheet, but have formatted the cell so it is displayed to two decimal places, then the value read into `R` will be the truncated value `1.23`.

2.11 Binary data

The `foreign` package allows you to read data in binary formats used by other statistical packages. Since `R` is an open source project, it can only read binary formats that are themselves *open*, in the sense that the standards for reading and writing data are well-documented. For example, there is a function in the `foreign` package for reading SAS XPORT files, a format that has been adopted as a standard by the US Food and Drug Administration ((http://www.sas.com/govedu/fda/faq.html)). However, there is no function in the `foreign` package for reading native SAS binaries (`SAS7BDAT` files). Other packages are available from CRAN ((http://cran.r-project.org)) that offer the possibility of reading SAS binary files: see the `haven` and `sas7bdat` packages.

The file `fem.dta` contains the FEM dataset in the format used by Stata. Read it into `R` with

```
library(foreign)
fem5 <- read.dta("data/fem.dta")
head(fem5)
```

The Stata data set contains value and variable labels. Stata variables with value labels are automatically converted to factors.

There is no equivalent of variable labels in an `R` data frame, but the original variable labels are not lost. They are still attached to the data frame as an invisible *attribute*, which you can see with

```
attr(fem5, "var.labels")
```

A lot of *meta-data* is attached to the data in the form of attributes. You can see the whole list of attributes with

```
attributes(fem5)
```

or just the attribute names with

```
names(attributes(fem5))
```

The `read.dta()` function can only read data from Stata versions 5–12. The R Core Team has not been able to keep up with changes in the Stata format. You may wish to try the `haven` package and the `readstata13` package, both available from CRAN.

2.12 Summary

In this exercise we have seen how to create a data frame in `R` from an external text file. We have also reviewed some common mistakes that result in garbled data.

The capabilities of the `foreign` package for reading binary data have also been demonstrated with a sample Stata data set.

Chapter 3

Data manipulation with `tidyverse`

3.1 Introduction

In this chapter we will produce *more or less* the same outputs than in chapter 1.2 and 1.4 using `tidyverse` packages framework.

The main objective of this exercise is to get familiar you with some of the main `tidyverse` features.

This is an **optional practical** for participants having already good basic R skills. > All the rest of the course can be done without knowledge of `tidyverse`.

3.2 The `births` data

We will work with `births` data-set from `Epi` package. First of all, load the `Epi` and `tidyverse` packages. Then load the births data-set.

```
library(Epi)
suppressPackageStartupMessages(library(tidyverse))
data(births)
```

You can type `?birth` in the `R` console to get a description of the `birth` data-set. (Alternatively, you can refer to chapter 1.3.2)

3.3 `tibble` vs `data.frame`

Most `dplyr` functions outputs return `tibble` object instead of `data.frame`. Inspect the class and characteristics of the `births` object.

```
class(births)
head(births)
```

Note: As any `R` object this can be summarized using `str` function.

```
str(births)
```

`births` object is a 500 x 8 data.frame. Let's convert `births` to `tibble` format with `as_tibble` function.

```
births_tbl <- as_tibble(births)

class(births_tbl)
births_tbl

# another way to visualize data set is to use glimpse function
glimpse(births_tbl)
```

You can see that `tibble` objects inherits from `data.frame` which implies that all functions working with `data.frame` objects will work with `tibble`. The opposite is not necessary true. `tibble` has a couple of extra features compared to classical `data.frame`. One of them is a slightly more user-friendly console print. The main difference is probably that `tibble` objects supports grouping/nesting features. Some examples we be done will see latter on.

3.4 Piping functions

This is one of the most popular features of `tidyverse` grammar. It enables function chaining in `R`. Function output is transparently passed as input to the next function and so on. It can help to make the code more comprehensive and readable. Here is an example of classic vs piped functions.

```
head(births, 4)
births |> head(4)
```

Note: By default the chained object is given as the first argument to the following function. You can use `.` if this is not the case.

Here is a dummy example where we do not give the first argument to `head` function but the second one.

```
4 %>% head(births, .)
```

This can also be achieves with `base` pipe `|>` with two differences

- The place holder is an underscore `_` not a dot `.`. This is a deliberate design choice to help people with less acute eyesight
- The argument must be named

```
4 |> head(births, pos=_)
```

3.5 `mutate` columns

`mutate` will allow you to add and or modify columns in a `tibble`. Let's create 2 new variables :

- `agegrp` (5 years mother's age group)
- `gest4` (gestation time split in 4 categories)

And modify 2 others:

- `hyp` (factor version of `hyp`; `normal` vs `hyper`)
- `sex` (factor version of `sex`; `M` vs `F`)

```
births_tbl <-
  births_tbl |>
  mutate(
    # modify hyp varible (conversion into factor)
    hyp =
      factor(
        hyp,
        levels = c(0, 1),
        labels = c("normal", "hyper")
      ),
    # creating a new variable aggrep
    agegrp =
      cut(
        matage,
        breaks = c(20, 25, 30, 35, 40, 45),
        right = FALSE
      ),
    # modify sex variable (conversion into factor)
```

```
    sex =
      factor(
        sex,
        levels = c(1, 2),
        labels = c("M", "F")
      ),
    # creating a new variable gest4 with case_when instead of cut
    gest4 =
      case_when(
        gestwks < 25 ~ "less than 25 weeks",
        gestwks >= 25 & gestwks < 30 ~ "25-30 weeks",
        gestwks >= 30 & gestwks < 35 ~ "30-35 weeks",
        gestwks >= 35 ~ "more than 35 weeks"
      )
  )

births_tbl
```

You can see as header the type of data contained in each column. For instance `<dbl>` stands for double (i.e. numeric value) and `fct` stands for factor. In `R` `data.frame` (/ `tibble`) data type must be the same within a column (e.g. numeric only) but can be of different type across columns. (note: `matrix` object supports only one type of data)

Note that `case_when` function do not return a `factor` but a `character` variable in this case. You will have to force the conversion from `character` to `factor` if needed.

3.6 `select` columns, `filter` and `arrange` rows

`select` is used for column sub-setting while filter is for row sub-setting. They are equivalent to the `[]` in `R` base language. Let's display a table where only babies' `id`, `sex`, `bweight` and mothers' `agegrp` are kept for babies with a `bweight` above 4000g.

```
births_tbl |>
  # select only id, women age group, sex
  # and birth weight of the baby
  select(id, agegrp, sex, bweight) |>
  # keep only babies weighing more than 4000g
  filter(bweight > 4000)
```

`select` can also be useful to reorder and rename columns. `arrange` is a nice feature to reorder observations according to chosen attributes. Let's rename `agegrp`, `sex` and `bweight` with better looking labels (e.g. `Age group`, `Sex`, `Birth weight`) and reorder the table according to babies' decreasing birth weight.

```
births_tbl |>
  # select only id, women age group, sex
  # and birth weight of the baby
  select(
    id,
    "Age group" = agegrp,
    Sex = sex,
    "Birth weight" = bweight
  ) |>
  # rearrange rows to put the heaviest newborn on top
  arrange(desc(`Birth weight`))
```

Note: `tibble` supports blank spaces in the column names which can be handy for final table rendering. When you want to work with columns with blank spaces, do not forget to use the ` (back-quote). Try to produce the same table but arranging the rows by decreasing birth weights within each sex.

```
births_tbl |>
  # select only id, women age group, sex
  # and birth weight of the baby
  select(
    id,
    "Age group" = agegrp,
    Sex = sex,
    "Birth weight" = bweight
  ) |>
  # rearrange rows to put the heaviest newborn on top
  arrange(Sex, desc(`Birth weight`))
```

You can arrange the `tibble` according to more than one column.

3.7 group_by and summarise data

One greatest features of `dplyr` is the ability to aggregate data sharing a common attribute to process per group operations. Here we want to compute the number of boys and girls in the data-set. The idea here is to split the `births` table in two groups. One with the boys, the other with the girls and to count the number of rows in each group.

```
births.01 <-
  births_tbl |>
  # group the data according to the sex attribute
  group_by(sex) |>
  # count the number of rows/individuals in each group
  summarise(
    count = n()
  )
births.01
```

Note: `n` function is equivalent to `nrow`

Now we have the number of boys and girls, we can compute the distribution (in percentage) of newborns per sex.

```
births.02 <-
  births.01 |>
  mutate(
    percent = count / sum(count) * 100
  )
```

Trick: most of `dplyr` functions can be combined with a column selection execution statement using `across` function. This can be very handy in some cases. As an example below a code to compute the `sum` of every `birth.02` numerical columns (numerical columns only)

```
births.03 <-
  births_tbl |>
  select(gest4, sex, gestwks, bweight, matage) |>
  group_by(gest4, sex) |>
  summarise(
    across(
      where(is.numeric),
      ~ mean(.x, na.rm = TRUE)
    ),
    .groups = "drop"
  )
births.03
```

`across` function supports the purrr-style lambda format, e.g. `~ mean(.x, na.rm = TRUE)` where `.x` refers to the values from the data set to be passed to the function. This is a common notation you will find across several `tidyverse` functions.

Some other functions ending by `_with` can be used conditionally within `dplyr`. As an example we can rename only columns which are not numeric at once (here we want to code all column names using upper characters) using the combination of `rename_with` and `where`.

```
births.03 |>
  rename_with(toupper, where(~ !is.numeric(.x)))
```

Let's now compute the number of births and the mean birth weight according to newborn gender.

```
births.05 <-
  births_tbl |>
  group_by(sex) |>
  summarise(
    count = n(),
    bweight.mean = mean(bweight)
  )
births.05
```

With `births.05` table, compute the global mean birth weight.

Note: with such a table the mean baby's birth weight have to be weighted by number of boys and girls (see. `?weighted.mean`).

```
births.05 |>
  summarise(
    count.tot = sum(count),
    bweight.mean.tot = weighted.mean(bweight.mean, count)
  )

# this is equivalent to
births_tbl |>
  summarise(
    count.tot = n(),
    bweight.mean.tot = mean(bweight)
  )
```

3.8 Multiple grouping

In some cases, we can be interested in looking at more than a single strata. This can be achieved using multiple grouping. Let's count the number of births per gender and birth weight class (low vs not low)

```
births.06 <-
  births_tbl |>
  group_by(sex, lowbw) |>
  summarise(
    count = n()
  )
```

```
## `summarise()` has grouped output by 'sex'. You can override using
## the `.groups` argument.
```

```
births.06
```

Try then to compute the percentage of babies in each group. Look at the difference between the 2 following command lines:

```
births.06 |>
  mutate(
    percent = count / sum(count) * 100
  )

births.06 |>
  ungroup() |>
  mutate(
    percent = count / sum(count) * 100
  )
```

Are the results the same?

Note: summarizing a data-set will remove the last level of grouping but not the other ones if multiple grouping has been performed. In some cases you might have to explicitly ungroup your `data.frame` before doing further calculations. In the previous examples, if you do not ungroup the data-set, percentages are computed per gender. Ungrouping will let you compute the overall percentages.

Trick: a good practice is to always ungroup the summarized dataset in order to prevent form confusion. You can do it using the `.group = 'drop'` option in `summarize()`.

```
# this tibble will still be grouped by sex
births_tbl |>
  group_by(sex, lowbw) |>
  summarise(
    count = n()
  )

# this tibble will be group free
births_tbl |>
  group_by(sex, lowbw) |>
  summarise(
    count = n(),
    .groups = "drop"
  )
```

The same exercise can be done using gestation time group (`gest4`) as stratifying variable. Lets compute number of births and mean birth weights according to gestation time category.

```
births_tbl |>
  group_by(gest4) |>
  summarise(
    count = n(),
    bweight.mean = mean(bweight)
  )
```

Any trend? It seems that birth weight increases with gestation time. We can also spot that in our data-set the gestation time is missing for 10 newborns. We will do not consider this observation for the rest of the exercise. Lets cross-tabulate the birth weight category and the gestation time groups.

```
births_tbl |>
  # keep only the newborn with defined gesational time category
  filter(
    !is.na(gest4)
  ) |>
  group_by(lowbw, gest4) |>
  # compute the number of babies in each cross category
  summarise(
    count = n()
  ) |>
```

```
  # compute the percentage of babies in each gestational
  # time category per birth weight category
  mutate(
    percent = count / sum(count, na.rm = TRUE)
  )
```

```
## `summarise()` has grouped output by 'lowbw'. You can override
## using the `.groups` argument.
```

Similarly we can be interested in the birth weight distribution per gestational time.

```
births_tbl |>
  filter(
    !is.na(gest4)
  ) |>
  group_by(gest4, lowbw) |>
  summarise(
    count = n()
  ) |>
  # compute the percentage of babies in each birth weight category
  # per gestational time category
  mutate(
    percent = count / sum(count, na.rm = TRUE)
  )
```

```
## `summarise()` has grouped output by 'gest4'. You can override
## using the `.groups` argument.
```

Note: grouping order matters! and can be confusing so think about ungrouping intermediate tables.

3.9 Bind and join tables

Another nice feature of `dplyr` is tables binding and joining. To practice we will create two `tibbles`:

- `age` an individual database which contains `pid` (unique individuals id) and their `age` in year
- `center` an study center database which contains `pid` (unique individuals id) and `center` (the center where an individual is registered coded as a letter)

```
age <-
  tibble(
    pid = 1:6,
    age = sample(15:25, size = 6, replace = TRUE)
  )

center <-
  tibble(
    pid = c(1, 2, 3, 4, 10),
    center = c("A", "B", "A", "B", "C")
  )

age
center
```

Now the tables are define we will try to make the linkage between individuals ages and the center they belong to. First of all let's have a look to `bind_rows` function.

```
bind_rows(age, center)
```

Is it useful? Here not really because we do not want to *bind* the data-set (but *join* them instead) but that can be in other situations (e.g. several individuals data base to merge..).

Note: in `bind_rows`, if columns names do not match, they are fill with `NA`.

Here we want to join the 2 `tibble` according to their common attribute `pid`. Depending on the context you can be interested in joining tables differently. Have a look at the differences between `left_join`, `full_join` and `inner_join`.

```
# all individuals from ages are kept
left_join(age, center, by = c("pid"))
# everithing is kept
full_join(age, center, by = c("pid"))
# only the individuals present in both dataset are kept
inner_join(age, center, by = c("pid"))
```

Can you spot the differences between the commands above? As an exercise, you can try to compute the individuals' mean age per center.

```
inner_join(age, center, by = c("pid")) |>
  group_by(center) |>
  summarise(
    mean_age = mean(age)
  )
```

Note: the `by` argument indicates which column should be use to make the *join*. In some cases, you might have to uses several columns to match (e.g. per sex and age group), this can be easily done specifying a vector of column names.

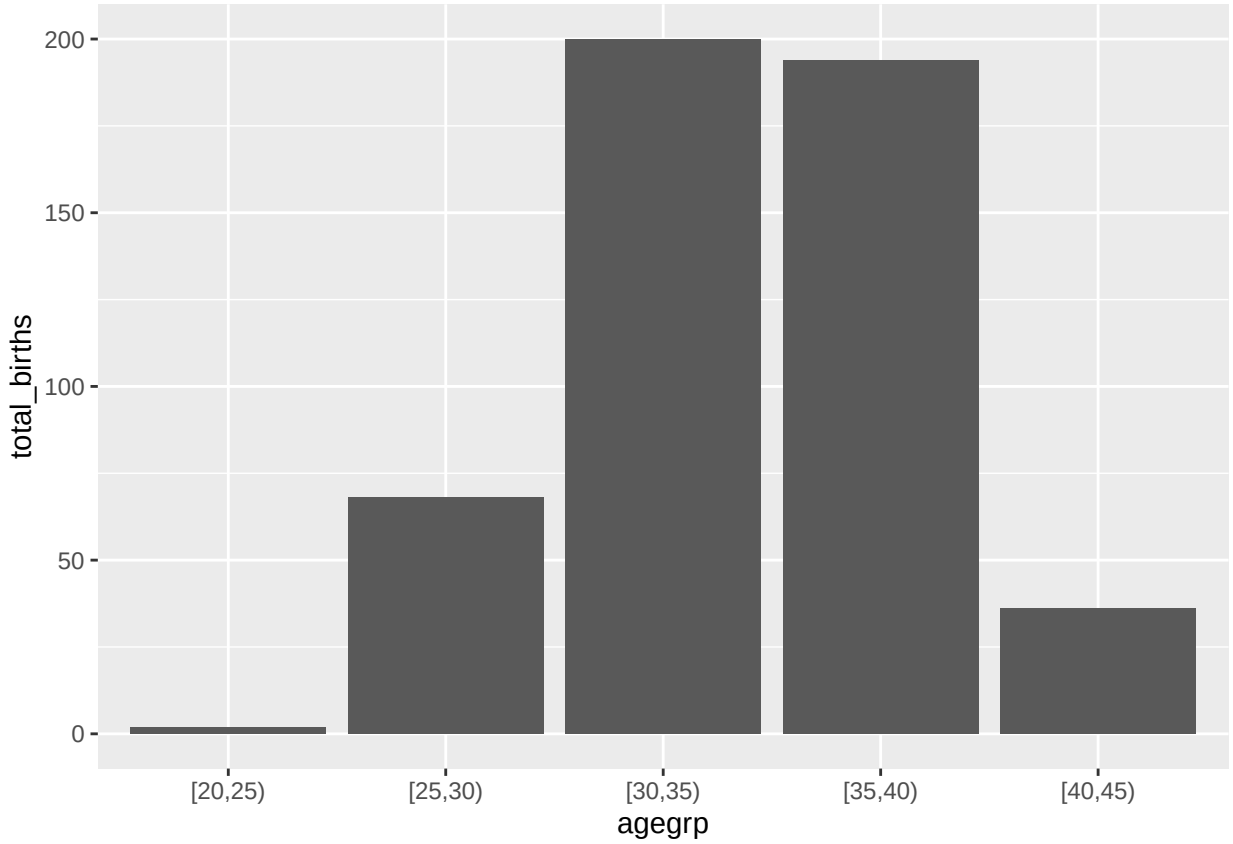
From now on, we will consider other packages than `dplyr` from the `tidyverse` suits.

3.10 Data Visualization with ggplot2

One of the package that have contributed to `tidyverse` success is for sure `ggplot2`. We will go more into the details on how to produce advanced graphs with `ggplot2` in another practical. Let's just have a quick example of graphic creation using `ggplot2`. Let's draw a bar plot to visualize the number of births by women age group. First you have to create a table with the number of birth per age group.

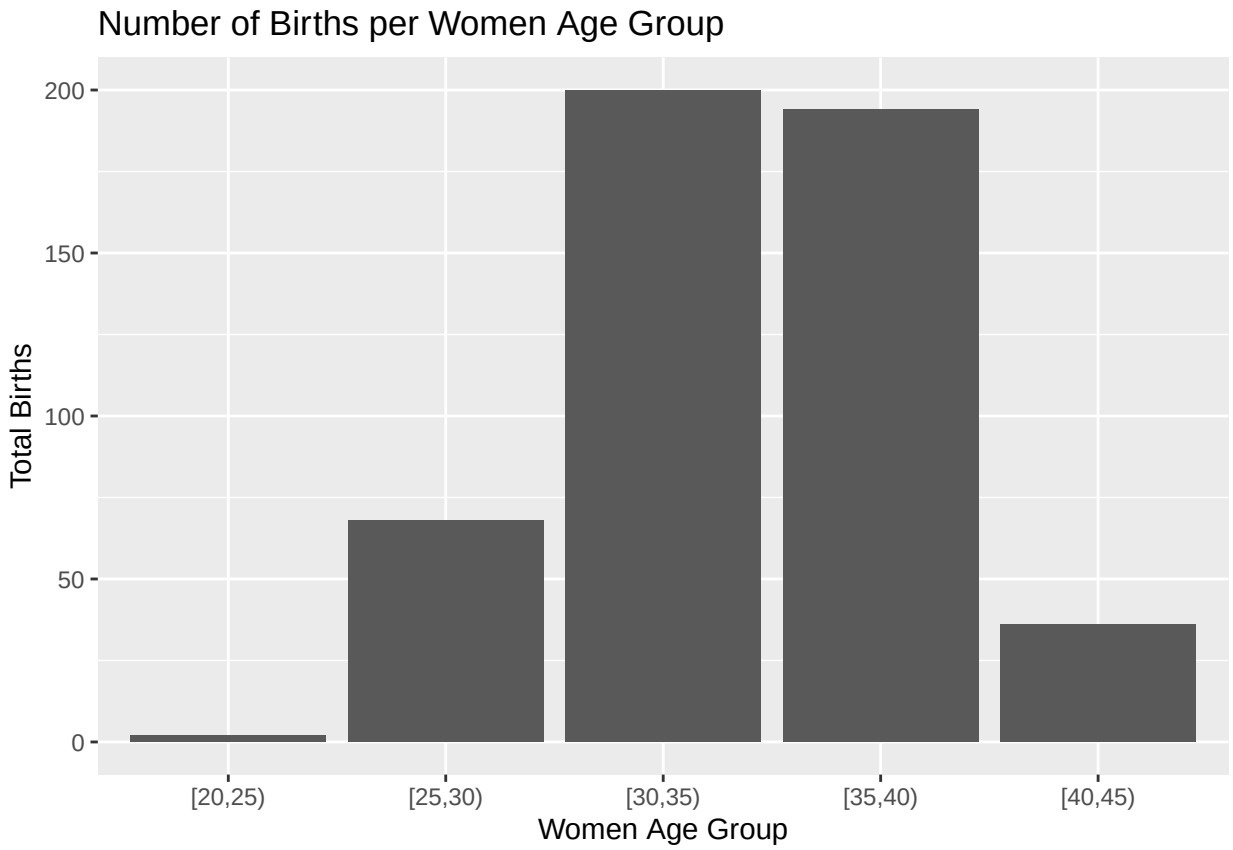
```
birth_per_ageg <- births_tbl |>
  group_by(agegrp) |>
  summarise(total_births = n())
```

```
(gg.01 <-
  ggplot(birth_per_ageg, aes(x = agegrp, y = total_births)) +
  geom_bar(stat = "identity"))
```



This graph can be customize adding labels and title to the plot:

```
(gg.02 <-
  gg.01 +
  xlab("Women Age Group") +
  ylab("Total Births") +
  ggtitle("Number of Births per Women Age Group"))
```



As you can see, plots from `ggplot` family are built incrementally using the `+` operator for each additional element.

3.11 pivoting data with tidyr

`dplyr` often comes with its good friend `tidyr` when we are performing data manipulation. `tidyr` main features is to reshape tables from long to wide format and vis-versa. Let's have an example. Let's transform in wide format the previously created `birth_per_ageg` table. We want to have a table with one column per age group containing the `total_births` numbers.

```
birth_per_ageg

birth_per_ageg_wide <-
  birth_per_ageg |>
  pivot_wider(
    names_from = "agegrp",
    values_from = "total_births"
  )

birth_per_ageg_wide
```

This table can easily be formatted back in long format using `pivot_longer` function:

```
birth_per_ageg_long <-
  birth_per_ageg_wide |>
  pivot_longer(
    cols = 1:5,
    names_to = "agegrp",
    values_to = "total_births"
  )
```

```
birth_per_ageg_long
```

Are the tables `birth_per_ageg` and `birth_per_ageg_long` identical?

```
identical(birth_per_ageg, birth_per_ageg_long)
```

Not really because the factor type of `agegrp` column has been lost during the transformation. Let's convert `agegrp` column into a factor. Is the new table identical to `birth_per_ageg` ?

```
birth_per_ageg_long_02 <-
  birth_per_ageg_long |>
  mutate(agegrp = as.factor(agegrp))

identical(birth_per_ageg, birth_per_ageg_long_02)
```

Here we have seen the simplest example you can have of table reshaping with `tidyr`. If you are interested check the dedicated vignette (`vignette("pivot")`) to learn how to perform more advanced tables reshaping.

3.12 reading files with readr

Another package from `tidyverse` that can be introduced here is `readr` that contains a set of functions equivalent to the core `R` data.frame reading functions (e.g. `read.table()`, `read.csv()`, `read.delim()`, …). The main change is that data are loaded in `R` as `tibble` instead of `data.frame`, type of variables (columns) are *guessed* if possible, and some extra data checking tests are performed.

Let's explore this differences with `fem` dataset available in `data` directory.

```
# read a csv using core R
fem.csv.core <- read.csv("data/fem.csv")
# read a csv using tidyverse
fem.csv.tidy <- read_csv("data/fem.csv")
```

```
## Rows: 118 Columns: 9
## -- Column specification --------------------------------------
## Delimiter: ","
## dbl (9): ID, AGE, IQ, ANXIETY, DEPRESS, SLEEP, SEX, LIFE, WEIGHT
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# compare
fem.csv.core
fem.csv.tidy
# table dimensions
dim(fem.csv.core)
dim(fem.csv.tidy)
# compare column types
map(fem.csv.core, class)
map(fem.csv.tidy, class)
```

note: in case you do not fully get the last lines and the `map()` call, it will be explained in the next section on `purrr` package.

Here we see that the only difference is the type of object loaded `data.frame` vs `tibble` and the default type chosen to cast numeric values (`integer` vs `numeric`).

What about loading `occoh.txt` you will be using in some other practical in the coming days.

```
# read a csv using core R
occoh.txt.core <- read.table("data/occoh.txt")
```

```
# read a csv using tidyverse
occoh.txt.tidy <- read_table("data/occoh.txt")
```

```
##
## -- Column specification --------------------------------------------
## cols(
##   id = col_double(),
##   birth = col_double(),
##   entry = col_date(format = ""),
##   exit = col_date(format = ""),
##   death = col_date(format = ""),
##   chdeath = col_double()
## )
```

```
## Warning: 1501 parsing failures.
## row col  expected    actual             file
##   1  -- 6 columns 7 columns 'data/occoh.txt'
##   2  -- 6 columns 7 columns 'data/occoh.txt'
##   3  -- 6 columns 7 columns 'data/occoh.txt'
##   4  -- 6 columns 7 columns 'data/occoh.txt'
##   5  -- 6 columns 7 columns 'data/occoh.txt'
## ... ... ......... ......... .................
## See problems(...) for more details.
```

```
occoh.txt.tidy <- read_table("data/occoh.txt")
```

```
##
## -- Column specification --------------------------------------------
## cols(
##   id = col_double(),
##   birth = col_double(),
##   entry = col_date(format = ""),
##   exit = col_date(format = ""),
##   death = col_date(format = ""),
##   chdeath = col_double()
## )
```

```
## Warning: 1501 parsing failures.
## row col  expected    actual             file
##   1  -- 6 columns 7 columns 'data/occoh.txt'
##   2  -- 6 columns 7 columns 'data/occoh.txt'
##   3  -- 6 columns 7 columns 'data/occoh.txt'
##   4  -- 6 columns 7 columns 'data/occoh.txt'
##   5  -- 6 columns 7 columns 'data/occoh.txt'
## ... ... ......... ......... .................
## See problems(...) for more details.
```

```
# compare
occoh.txt.core
occoh.txt.tidy
# table dimensions
dim(occoh.txt.core)
dim(occoh.txt.tidy)
# compare column types
map(occoh.txt.core, class)
map(occoh.txt.tidy, class)
```

As you can see, in addition to inferring the type of columns in the input data (here some dates), using `readr` to load you data-set can help you to detect inconsistencies in input data formatting (there are no true problem

here).

If you are interested, you can explore the other functions of `readr` and see how you can tune it.

3.13 String manipulation with stringr

Another popular `tidyverse` popular package is `stringr` package. This package is specialized in the string manipulation. Here are couple of examples.

Let's create a character vector with the following elements representing country names: "Estonia", "Finland", "Denmark", "United Kingdom", "France".

```
countries <-
  c("Estonia", "Finland", "Denmark", "United Kingdom", "France")
```

With `stringr` functions perform the following actions.

Extract the first three characters from each country name:

```
country_initials <- str_sub(countries, start = 1, end = 3)
```

Convert all country names to uppercase:

```
countries_upper <- str_to_upper(countries)
```

Replace "United" with "Utd" in each country name:

```
countries_modified <- str_replace(countries, "United", "Utd")
```

Find the positions of the letter "n" in each country name:

```
a_positions <- str_locate_all(countries, "n")
```

As you can see, the output of `str_locate_all` is a list (one element per character string) containing a 2 column table with one line for each match. The first column (start) being the position of the beginning of the match and the second one (end) being the end of the match. In our case, since we are searching for a single character match, this 2 indexes are always the same.

Count the number of characters in each country name:

```
character_counts <- str_length(countries)
```

These examples demonstrate various string manipulation operations using the `stringr` package. You can modify the exercises, combine several operations or explore other string manipulation functions provided by `stringr` to further practice and enhance your skills in manipulating and analyzing text data.

3.14 purrr package to apply functions to list

Among my favorite `tidyverse` packages, you will find `purrr`. This package contains several functions that are very similar to `lapply` function.

Apply a function to each element of the vector using map(). Here producing the mean of some grades per class:

```
# define the grade dataset
grades <-
  list(
    c1 = c(80, 85, 90),
    c2 = c(75, 70, 85, 88),
    c3 = c(90, 85, 95)
  )
# compute grades
mean_grades <- map(grades, mean)
```

By default `map()` return a list. One of the nice feature of `purrr` functions is to be able to specify the type of output you want (e.g. `_dbl` for numeric, `_chr` for characters, …). Check and try to explain the differences between the following command lines:

```
map(grades, mean)
map_dbl(grades, mean)
map_chr(grades, mean)
```

```
## Warning: Automatic coercion from double to character was deprecated in
## purrr 1.0.0.
## i Please use an explicit call to `as.character()` within
##   `map_chr()` instead.
## Call `lifecycle::last_lifecycle_warnings()` to see where this
## warning was generated.
```

```
map_df(grades, mean)
```

Other nice features of `map` like functions is he availability to support more than one argument. `map2()` for 2 arguments and `pmap()` for more than 2. This can be very handy in some conditions. If you are interested you can have a look to this function help file and play with the examples.

`purrr` package has also a set of functions that can be used to apply iteratively a function using `reduce` and/or `accumulate`. The 2 functions behave the same way, it takes the 2 first element of a list, apply a function taking 2 arguments. The results is combined with the third element of the list and given as input to the same function and so on.. The only difference is that accumulate return intermediate results while reduce return only the final results.

Here an example of the cumulative product of the 10 first numbers.

```
1:10 |> purrr::reduce(`*`)
1:10 |> purrr::accumulate(`*`)
```

`purrr` have many of others useful features. Please check the dedicated documentation if you want to go further with this package.

3.15 Bonus: Rendering tables

Once you have produced a nice data-set we can be interested in rendering it in a nice format that can meet presentation/publication expectations. The `kableExtra` table can be useful to achieve this goal.

```
# if(!require(kableExtra)) install.packages('kableExtra')
library(kableExtra)

births.08 <-
  births_tbl |>
  filter(
    !is.na(gest4)
  ) |>
  group_by(gest4) |>
  summarise(
    N = n()
  ) |>
  mutate(
    `(%)` = (N / sum(N)) |> scales::percent()
  )

# default
births.08

# create an html version of the table and save it on the hard drive
```

```
births.08 |>
  kable() |>
  kable_styling(
    bootstrap_options =
      c("striped", "hover", "condensed", "responsive"),
    full_width = FALSE
  ) |>
  save_kable(file = "births.08.html", self_contained = TRUE)
```

note: One other very cool package to produce advance formatted Excel spreadsheet I am using more and more is `openxlsx`. Check it out if you are interested.

Chapter 4

Tabulation

4.1 Introduction

R and its add-on packages provide several different tabulation functions with different capabilities. The appropriate function to use depends on your goal. There are at least three different uses for tables.

The first use is to create simple summary statistics that will be used for further calculations in R. For example, a two-by-two table created by the `table` function can be passed to `fisher.test`, which will calculate odds ratios and confidence intervals. The appearance of these tables may, however, be quite basic, as their principal goal is to create new objects for future calculations.

A quite different use of tabulation is to make *production quality* tables for publication. You may want to generate reports from for publication in paper form, or on the World Wide Web. The package `xtable` provides this capability, but it is not covered by this course.

An intermediate use of tabulation functions is to create human-readable tables for discussion within your workgroup, but not for publication. The `Epi` package provides a function `stat.table` for this purpose, and this practical is designed to introduce this function.

4.2 The births data

We shall use the births data which concern 500 mothers who had singleton births in a large London hospital. The outcome of interest is the birth weight of the baby, also dichotomised as normal or low birth weight. These data are available in the Epi package:

```
library(Epi)
data(births)
names(births)
head(births)
```

In order to work with this data set we need to transform some of the variables into factors. This is done with the following commands:

```
births$hyp <- factor(births$hyp, labels = c("normal", "hyper"))
births$sex <- factor(births$sex, labels = c("M", "F"))
births$agegrp <-
  cut(
    births$matage,
    breaks = c(20, 25, 30, 35, 40, 45),
    right = FALSE
  )
births$gest4 <-
  cut(
```

```
    births$gestwks,
    breaks = c(20, 35, 37, 39, 45),
    right = FALSE
  )
```

Now use `str(births)` to examine the modified data frame. We have transformed the binary variables `hyp` and `sex` into factors with informative labels. This will help when displaying the tables. We have also created grouped variables `agegrp` and `gest4` from the continuous variables `matage` and `gestwks` so that they can be tabulated.

4.3 One-way tables

The simplest table one-way table is created by

```
stat.table(index = sex, data = births)
```

This creates a count of individuals, classified by levels of the factor `sex`. Compare this table with the equivalent one produced by the `table` function. Note that `stat.table` has a `data` argument that allows you to use variables in a data frame without specifying the frame.

You can display several summary statistics in the same table by giving a list of expressions to the `contents` argument:

```
stat.table(
  index = sex,
  contents = list(count(), percent(sex)),
  data = births
)
```

Only a limited set of expressions are allowed: see the help page for `stat.table` for details.

You can also calculate marginal tables by specifying `margin=TRUE` in your call to `stat.table`. Do this for the above table. Check that the percentages add up to 100 and the total for `count()` is the same as the number of rows of the data frame `births`.

To see how the mean birth weight changes with `sex`, try

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

Add the count to this table. Add also the margin with `margin=TRUE`.

As an alternative to `bweight` we can look at `lowbw` with

```
stat.table(index = sex, contents = percent(lowbw), data = births)
```

All the percentages are 100! To use the `percent` function the variable `lowbw` must also be in the index, as in

```
stat.table(
  index = list(sex, lowbw),
  contents = percent(lowbw),
  data = births
)
```

The final column is the percentage of babies with low birth weight by different categories of gestation.

- Obtain a table showing the frequency distribution of `gest4`.
- Show how the mean birth weight changes with `gest4`.
- Show how the percentage of low birth weight babies changes with `gest4`.

Another way of obtaining the percentage of low birth weight babies by gestation is to use the ratio function:

```
stat.table(gest4, ratio(lowbw, 1, 100), data = births)
```

This only works because `lowbw` is coded 0/1, with 1 for low birth weight.

Tables of odds can be produced in the same way by using `ratio(lowbw, 1-lowbw)`. The `ratio` function is also very useful for making tables of rates with (say) `ratio(D,Y,1000)` where `D` is the number of failures, and `Y` is the follow-up time. We shall return to rates in a later practical.

4.4 Improving the Presentation of Tables

The `stat.table` function provides default column headings based on the `contents` argument, but these are not always very informative. Supply your own column headings using *tagged* lists as the value of the `contents` argument, within a `stat.table` call:

```
stat.table(gest4, contents = list(
  N = count(),
  "(%)" = percent(gest4)
), data = births)
```

This improves the readability of the table. It remains to give an informative title to the index variable. You can do this in the same way: instead of giving `gest4` as the `index` argument to `stat.table`, use a named list:

```
stat.table(index = list("Gestation time" = gest4), data = births)
```

4.5 Two-way Tables

The following call gives a 2×2 table showing the mean birth weight cross-classified by `sex` and `hyp`.

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

Add the count to this table and repeat the function call using `margin = TRUE` to calculate the marginal tables.

Use `stat.table` with the ratio function to obtain a 2×2 table of percent low birth weight by `sex` and `hyp`.

You can have fine-grained control over which margins to calculate by giving a logical vector to the `margin` argument. Use `margin=c(FALSE, TRUE)` to calculate margins over `sex` but not `hyp`. This might not be what you expect, but the `margin` argument indicates which of the index variables are to be `marginalized out`, not which index variables are to remain.

4.6 Printing

Just like every other `R` function, `stat.table` produces an object that can be saved and printed later, or used for further calculation. You can control the appearance of a table with an explicit call to `print()`

There are two arguments to the print method for `stat.table`. The `width` argument which specifies the minimum column width, and the `digits` argument which controls the number of digits printed after the decimal point. This table

```
odds.tab <-
  stat.table(
    gest4,
    list("odds of low bw" = ratio(lowbw, 1 - lowbw)),
    data = births
)
print(odds.tab)
```

shows a table of odds that the baby has low birth weight. Use `width=15` and `digits=3` and see the difference.

Chapter 5

Graphics in R

There are three kinds of plotting functions in R:

- Functions that generate a new plot, e.g. `hist()` and `plot()`.
- Functions that add extra things to an existing plot, e.g. `lines()` and `text()`.
- Functions that allow you to interact with the plot, e.g. `locator()` and `identify()`.

The normal procedure for making a graph in R is to make a fairly simple initial plot and then add on points, lines, text etc., preferably in a script.

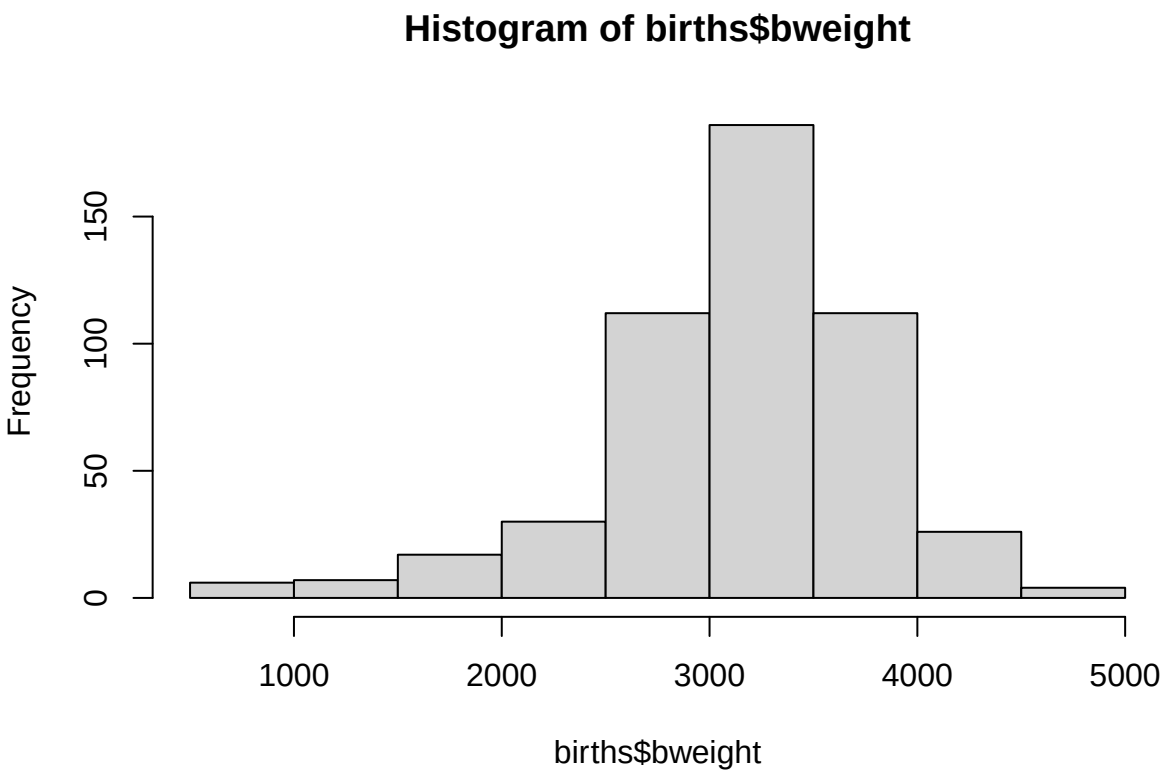
5.1 Simple plot on the screen

Load the births data and get an overview of the variables:

```
library(Epi)
data(births)
str(births)
```

Now look at the birth weight distribution with

```
hist(births$bweight)
```

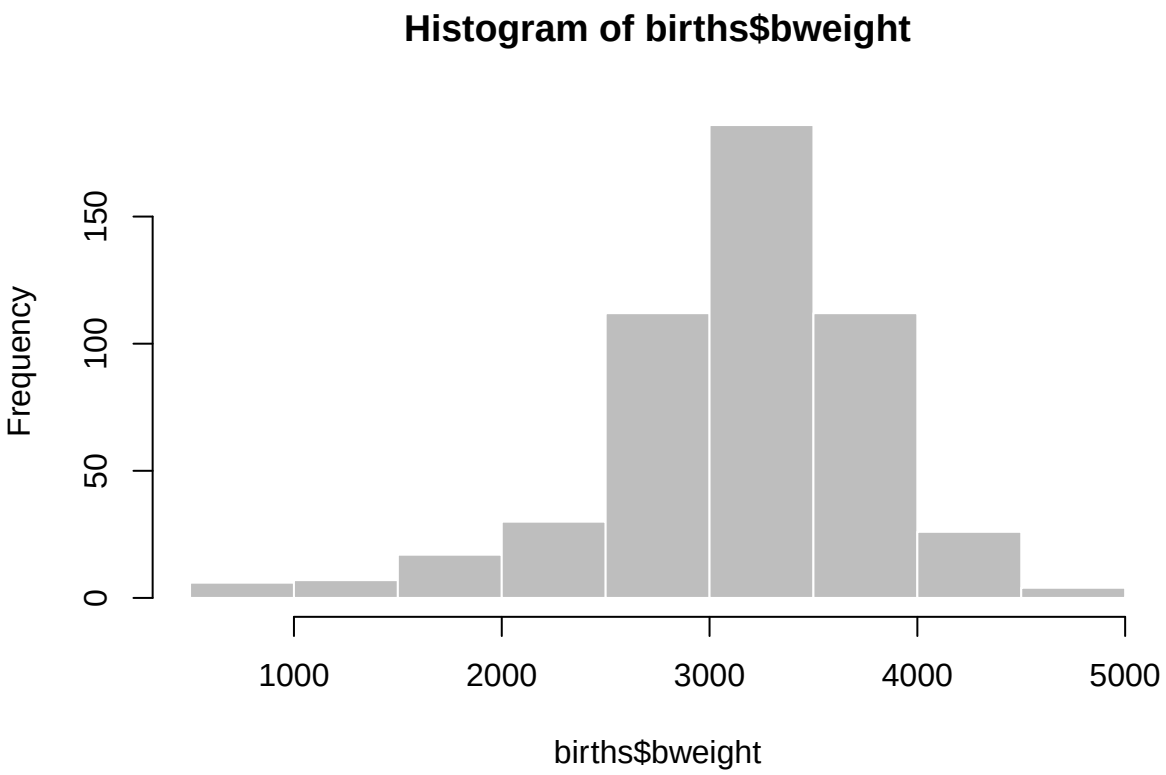


The histogram can be refined – take a look at the possible options with

```
help(hist)
```

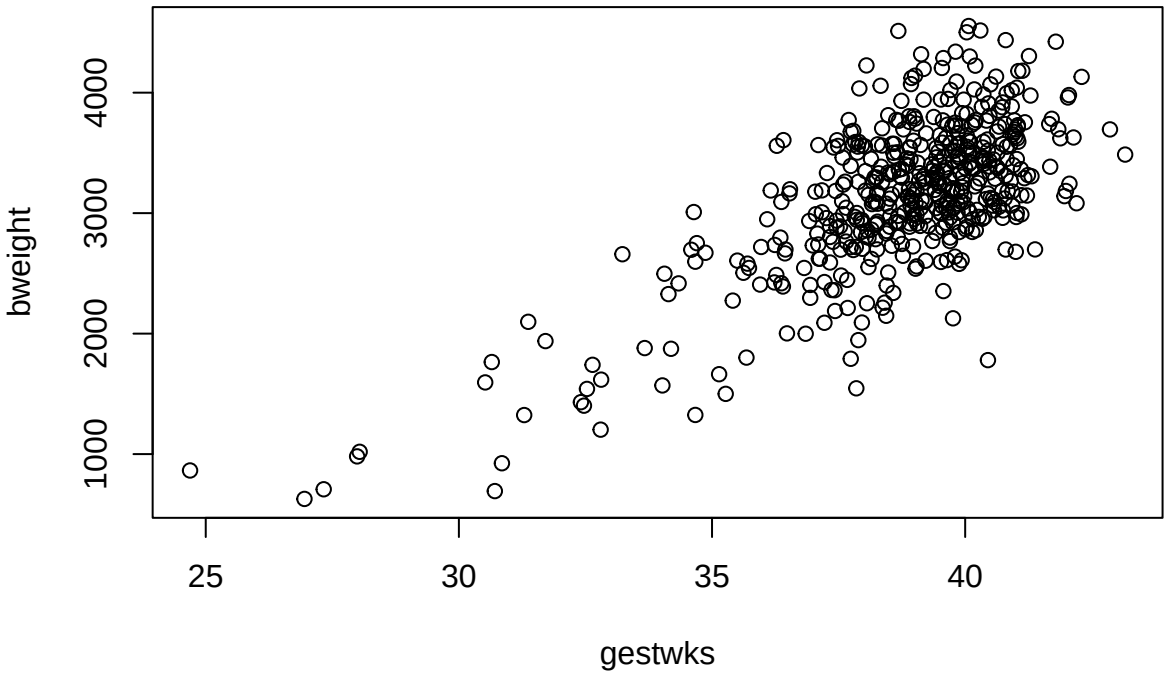
and try some of the options, for example:

```
hist(births$bweight, col = "gray", border = "white")
```



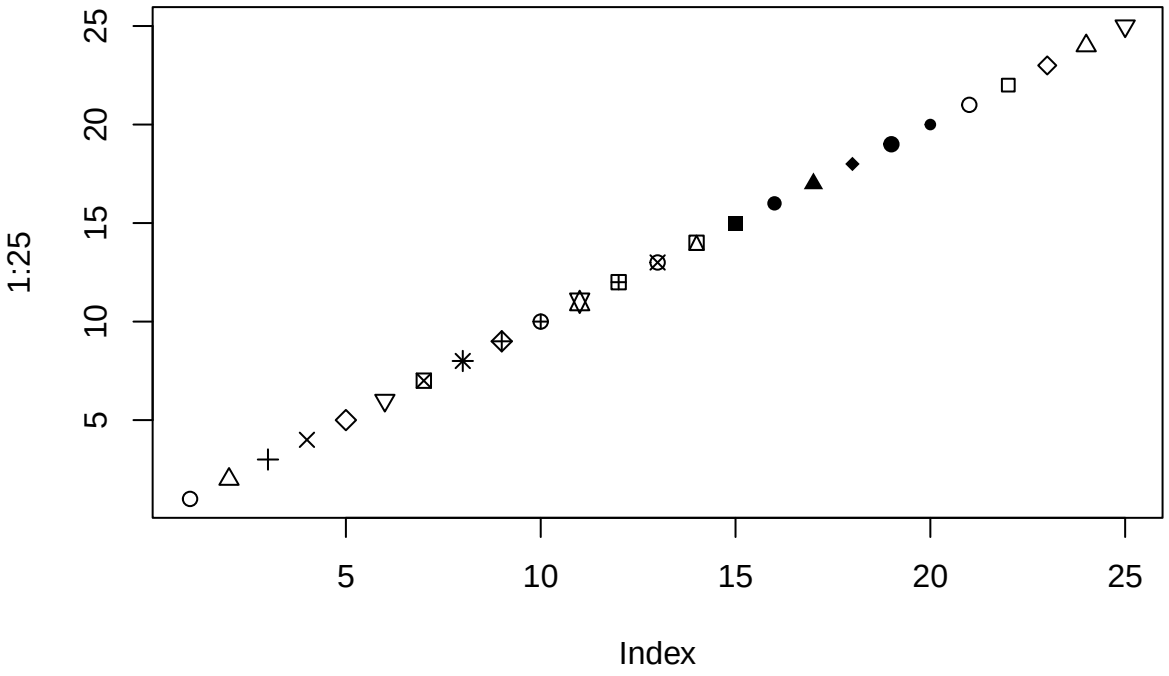
To look at the relationship between birthweight and gestational weeks, try

```
with(births, plot(gestwks, bweight))
```



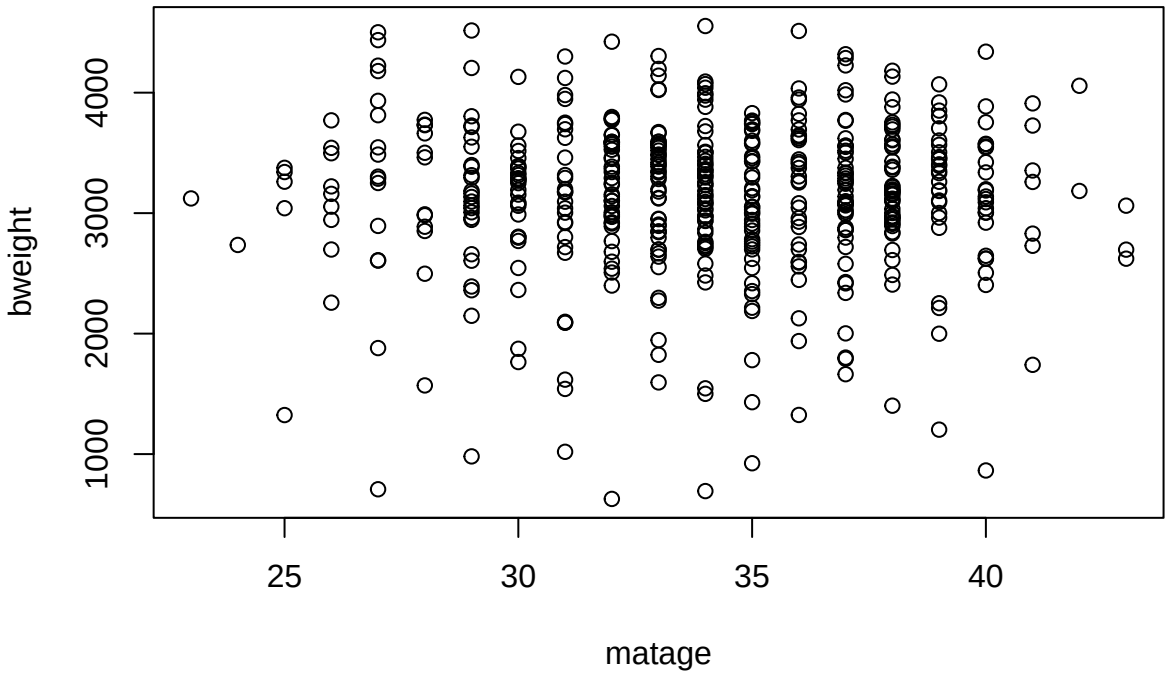
You can change the plot-symbol by the option `pch=`. If you want to see all the plot symbols try:

```
plot(1:25, pch = 1:25)
```



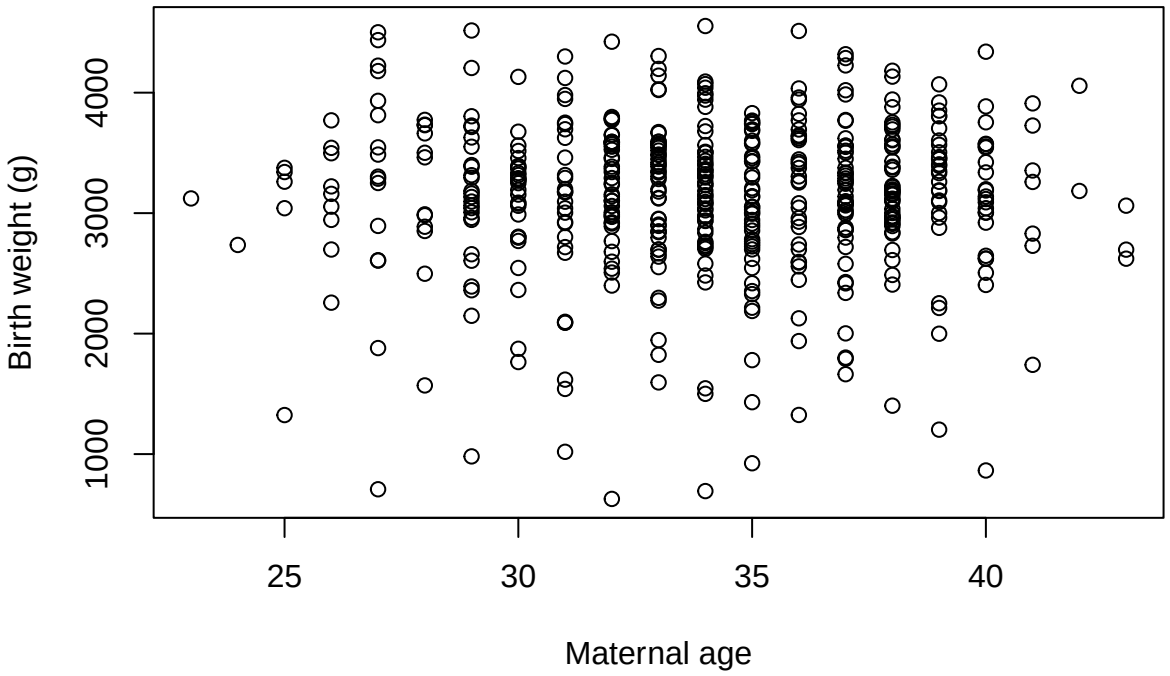
- Make a plot of the birth weight versus maternal age with

```
with(births, plot(matage, bweight))
```



Label the axes with

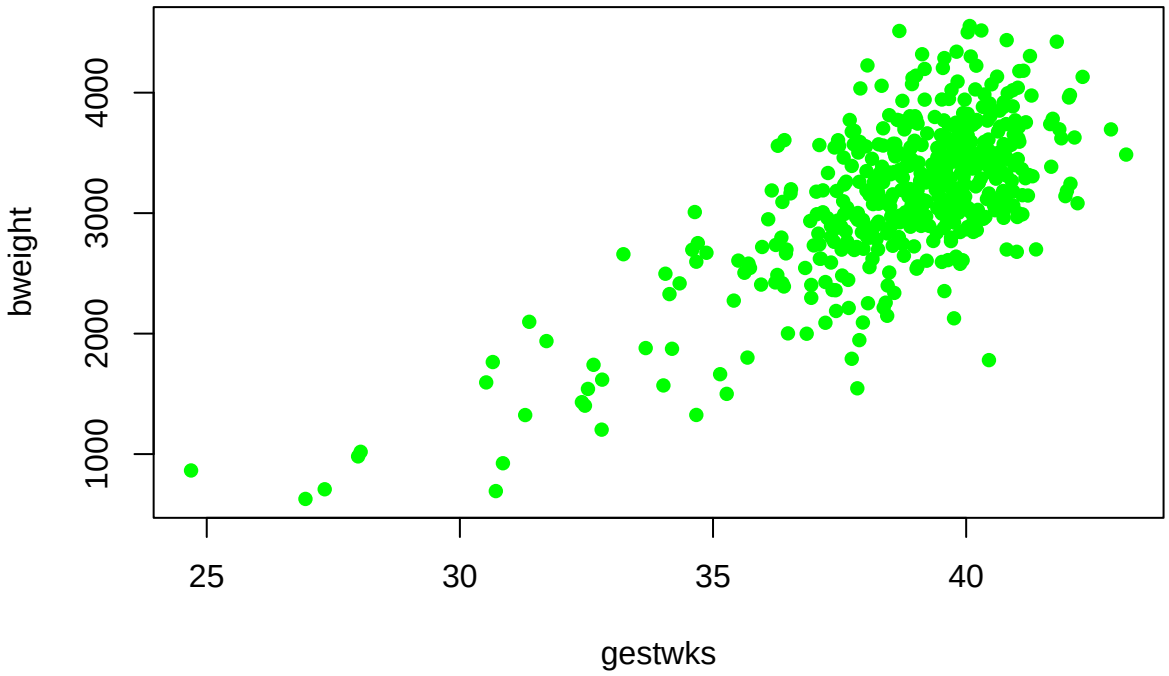
```
with(
  births,
  plot(
    matage,
    bweight,
    xlab = "Maternal age",
    ylab = "Birth weight (g)"
  )
)
```



5.2 Colours

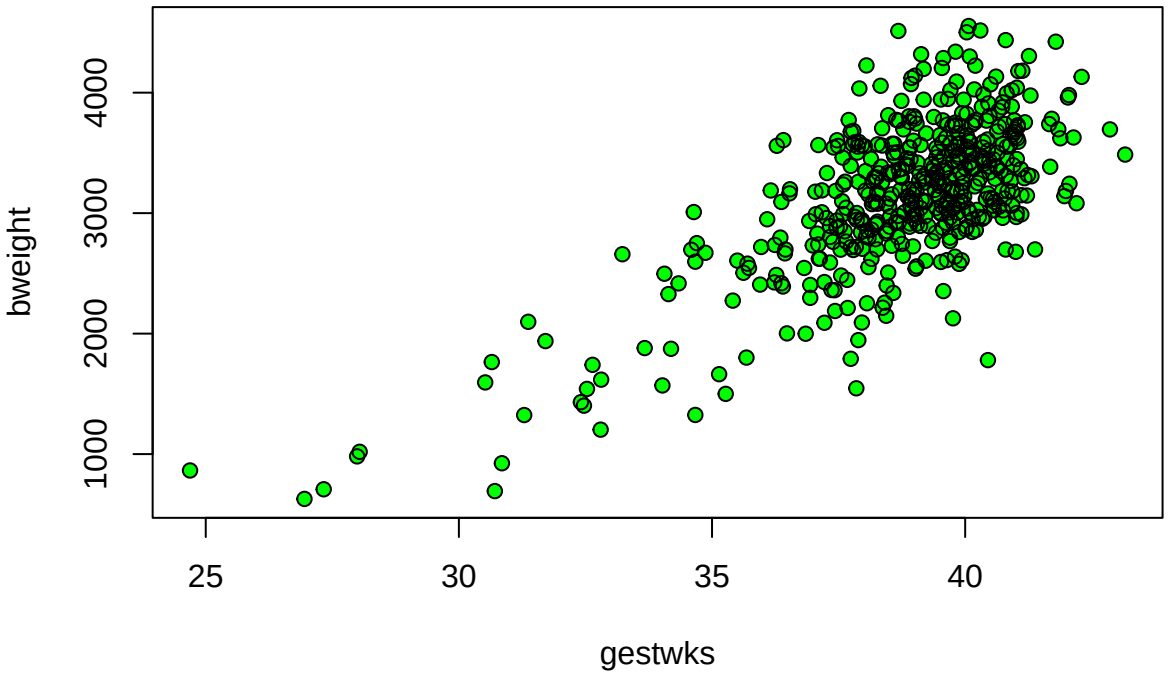
There are many colours recognized by `R`. You can list them all by `colours()` or, equivalently, `colors()` (`R` allows you to use British or American spelling). To colour the points of birthweight versus gestational weeks, try

```
with(births, plot(gestwks, bweight, pch = 16, col = "green"))
```



This creates a solid mass of colour in the centre of the cluster of points and it is no longer possible to see individual points. You can recover this information by overwriting the points with black circles using the `points()` function.

```
with(births, plot(gestwks, bweight, pch = 16, col = "green"))
with(births, points(gestwks, bweight, pch = 1))
```



Note: when the number of data points on a scatter plot is large, you may also want to decrease the point size: to get points that are 50% of the original size, add the parameter `cex=0.5` (or another number <1 for different sizes).

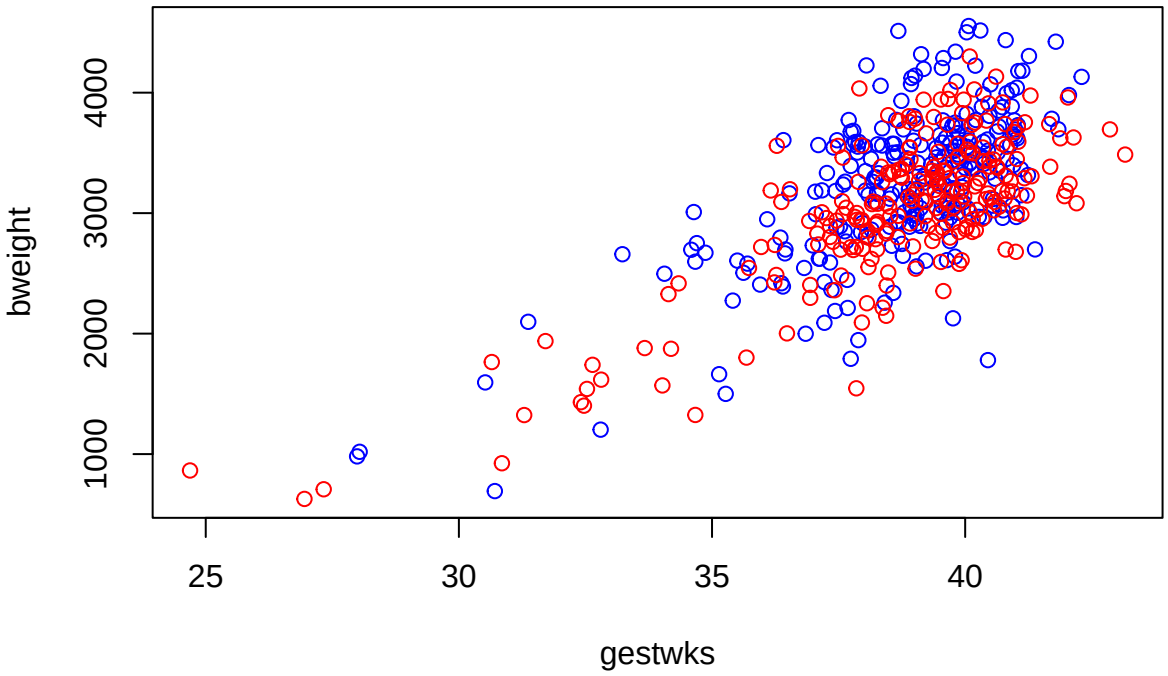
5.3 Adding to a plot

The `points()` function just used is one of several functions that add elements to an existing plot. By using these functions, you can create quite complex graphs in small steps.

Suppose we wish to recreate the plot of birthweight *vs* gestational weeks using different colours for male and female babies. To start with an empty plot, with `type='n'` argument.

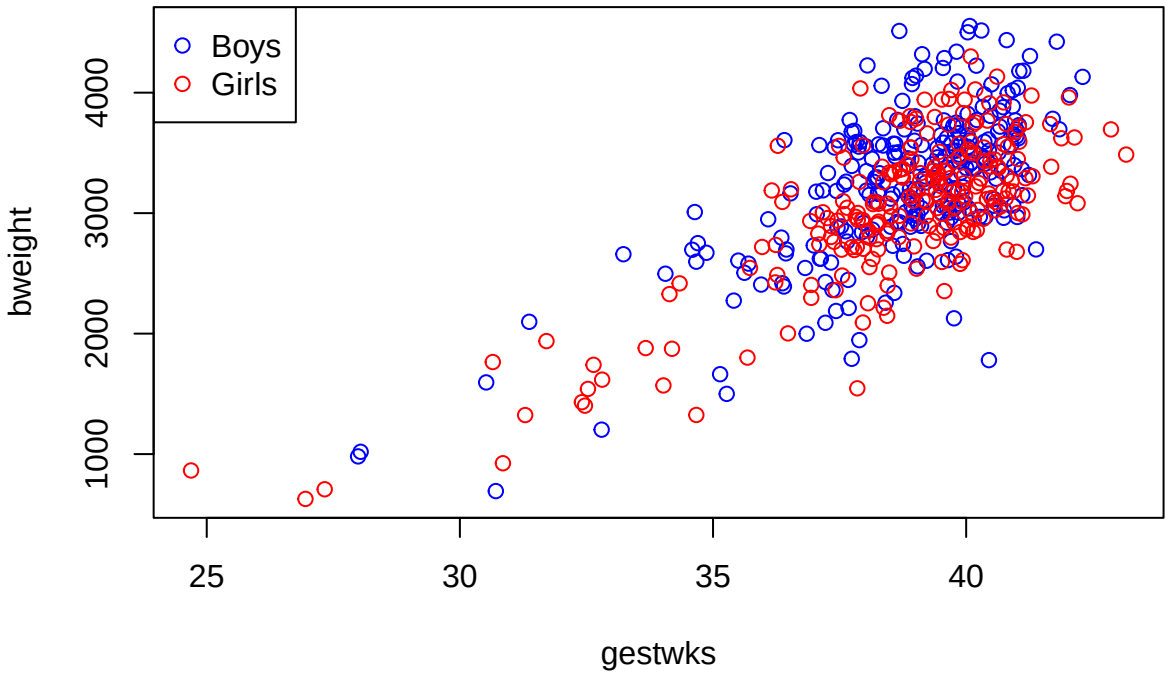
Then add the points with the `points` function.

```
with(births, plot(gestwks, bweight, type = "n"))
with(
  births,
  points(gestwks[sex == 1], bweight[sex == 1], col = "blue")
)
with(
  births,
  points(gestwks[sex == 2], bweight[sex == 2], col = "red")
)
```



To add a legend explaining the colours, try

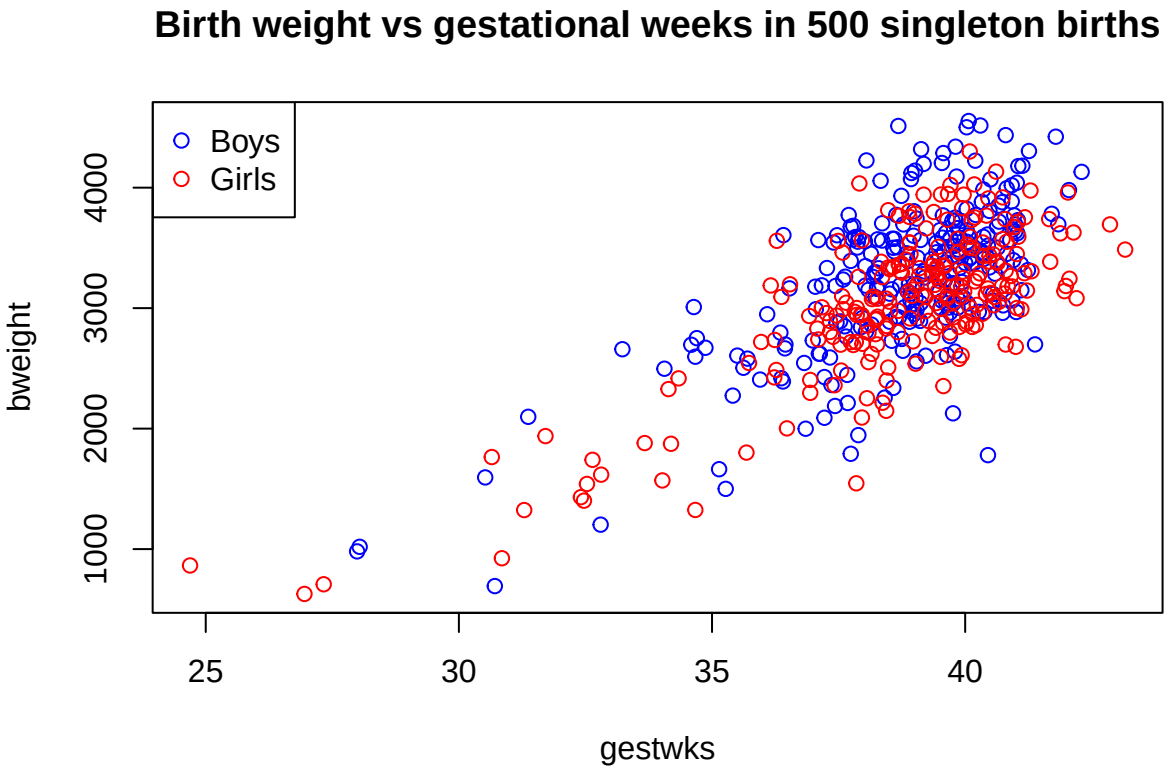
```
with(births, plot(gestwks, bweight, type = "n"))
with(
  births,
  points(gestwks[sex == 1], bweight[sex == 1], col = "blue")
)
with(
  births,
  points(gestwks[sex == 2], bweight[sex == 2], col = "red")
)
legend(
  "topleft",
  pch = 1,
  legend = c("Boys", "Girls"),
  col = c("blue", "red")
)
```



which puts the legend in the top left hand corner.

Finally we can add a title to the plot with

```
with(births, plot(gestwks, bweight, type = "n"))
with(
  births,
  points(gestwks[sex == 1], bweight[sex == 1], col = "blue")
)
with(
  births,
  points(gestwks[sex == 2], bweight[sex == 2], col = "red")
)
legend(
  "topleft",
  pch = 1,
  legend = c("Boys", "Girls"),
  col = c("blue", "red")
)
title(
  "Birth weight vs gestational weeks in 500 singleton births"
)
```



5.4 Using indexing for plot elements

One of the most powerful features of R is the possibility to index vectors, not only to get subsets of them, but also for repeating their elements in complex sequences.

Putting separate colours on males and female as above would become very clumsy if we had a 5 level factor instead of sex.

Instead of specifying one color for all points, we may specify a vector of colours of the same length as the `gestwks` and `bweight` vectors. This is rather tedious to do directly, but R allows you to specify an expression anywhere, so we can use the fact that `sex` takes the values 1 and 2, as follows:

First create a colour vector with two colours, and take look at `sex`:

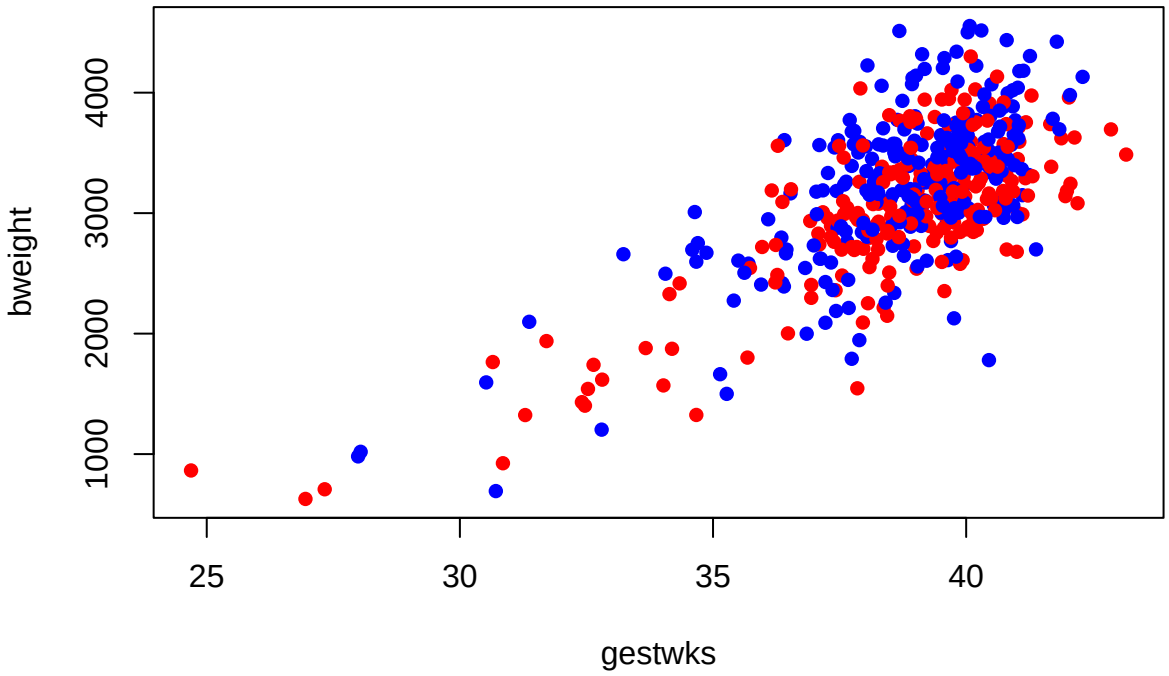
```
c("blue", "red")
births$sex
```

Now see what happens if you index the colour vector by sex:

```
c("blue", "red")[births$sex]
```

For every occurrence of a `1` in `sex` you get `"blue"`, and for every occurrence of `2` you get `"red"`, so the result is a long vector of `"blue"`s and `"red"`s corresponding to the males and females. This can now be used in the plot:

```
with(
  births,
  plot(gestwks, bweight, pch = 16, col = c("blue", "red")[sex])
)
```

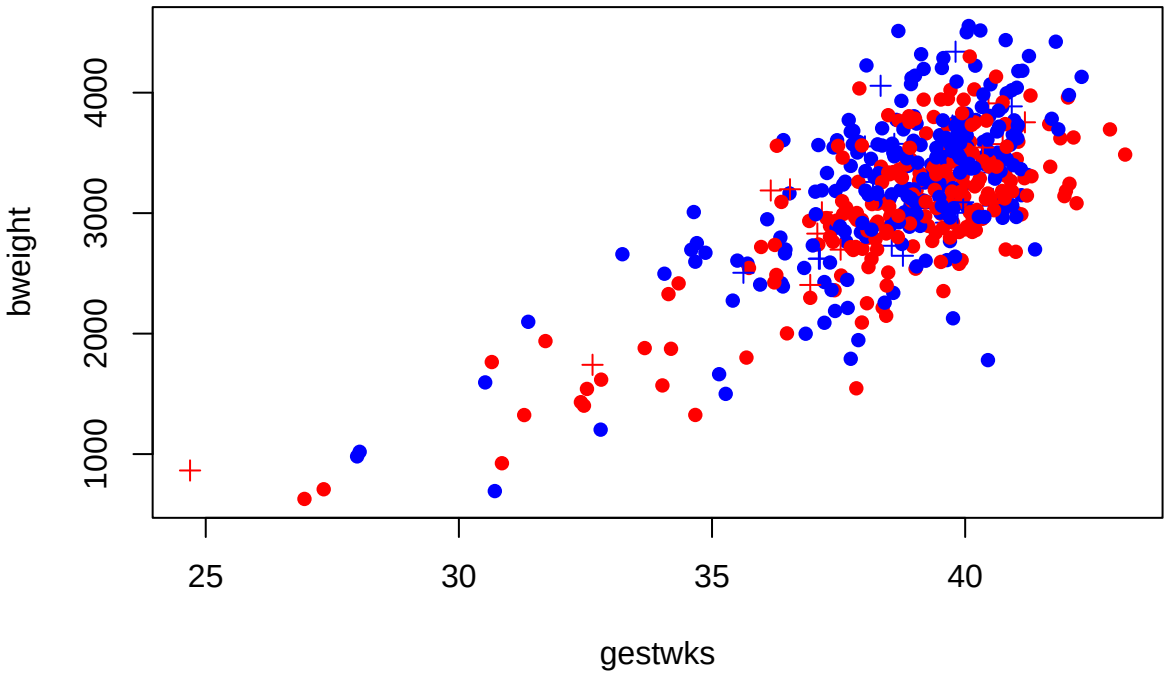


The same trick can be used if we want to have a separate symbol for mothers over 40 say. We first generate the indexing variable:

```
births$oldmum <- (births$matage >= 40) + 1
```

Note we add 1 because `( matage >= 40 )` generates a logic variable, so by adding 1 we get a numeric variable with values 1 and 2, suitable for indexing:

```
with(
  births,
  plot(
    gestwks,
    bweight,
    pch = c(16, 3)[oldmum],
    col = c("blue", "red")[sex]
  )
)
```

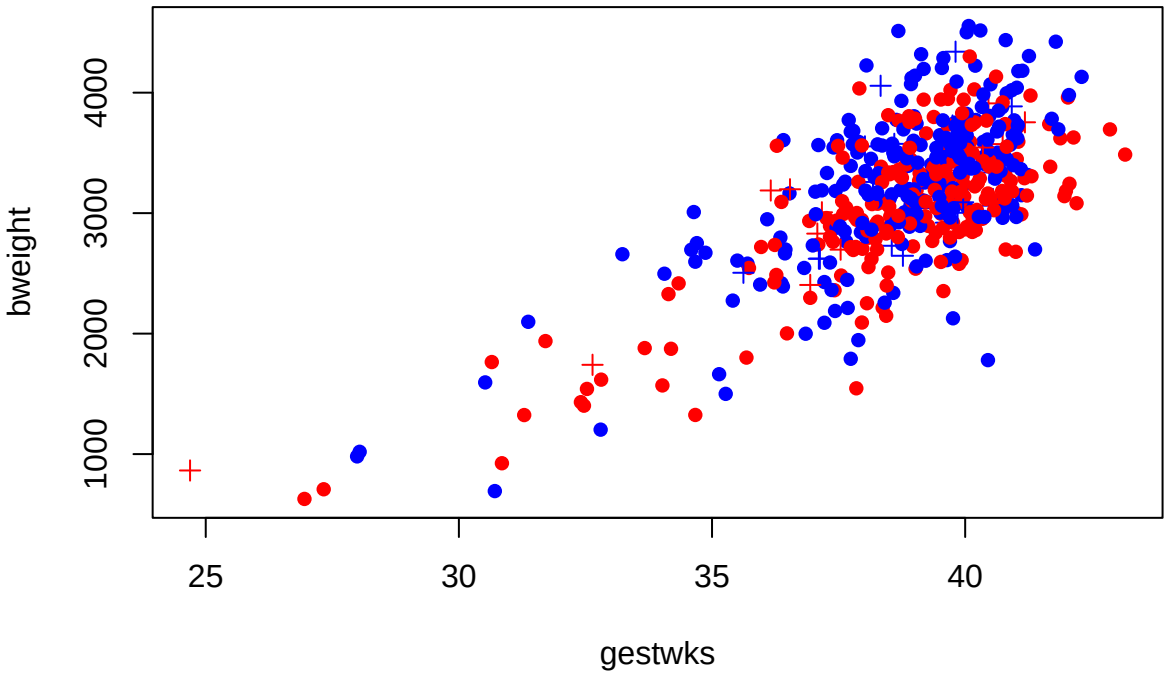


so

where `oldmum` is 1 we get `pch=16` (a dot) and where `oldmum` is 2 we get `pch=3` (a cross).

R will accept any kind of complexity in the indexing as long as the result is a valid index, so you don't need to create the variable `oldmum`, you can create it on the fly:

```
with(
  births,
  plot(
    gestwks,
    bweight,
    pch = c(16, 3)[(matage >= 40) + 1],
    col = c("blue", "red")[sex]
  )
)
```



5.5 Generating colours

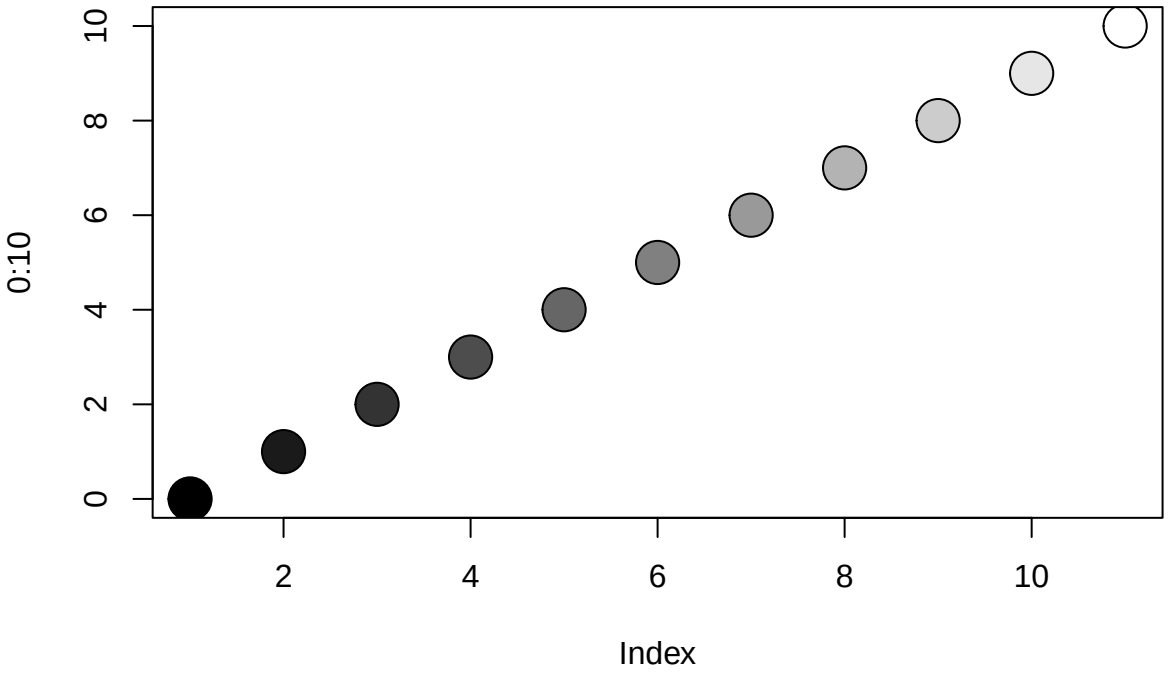
R has functions that generate a vector of colours for you. For example,

```
rainbow(4)
```

produces a vector with 4 colours (not immediately human readable, though). There are a few other functions that generates other sequences of colours, type `?rainbow` to see them. The `color` function (or `colour` function if you prefer) returns a vector of the colour names that R knows about. These names can also be used to specify colours.

Gray-tones are produced by the function `gray` (or `grey`), which takes a numerical argument between 0 and 1; `gray(0)` is black and `gray(1)` is white. Try:

```
plot(0:10, pch = 16, cex = 3, col = gray(0:10 / 10))
points(0:10, pch = 1, cex = 3)
```



5.6 Saving your graphs for use in other documents

If you need to use the plot in a report or presentation, you can save it in a graphics file. Once you have generated the script (sequence of R commands) that produce the graph (and it looks ok on screen), you can start a non-interactive graphics device and then re-run the script. Instead of appearing on the screen, the plot will now be written directly to a file. After the plot has been completed you will need to close the device again in order to be able to access the file. Try:

```
pdf(file = "bweight_gwks.pdf", height = 4, width = 4)
with(births, plot(gestwks, bweight, col = c("blue", "red")[sex]))
legend(
  "topleft",
  pch = 1,
  legend = c("Boys", "Girls"),
  col = c("blue", "red")
)
dev.off()
```

This will give you a portable document file `bweight_gwks.pdf` with a graph which is 4 inches tall and 4 inches wide.

Instead of *pdf*, other formats can be used (*jpg, png, tiff, …*). See `help(Devices)` for the available options.

In window-based environments (R GUI for Windows, R-Studio) you may also use the menu (`File`→`Save as ...` or `Export`) to save the active graph as a file and even copy-paste may work (from R graphics window to Word, for instance) – however, writing it manually into the file is recommended for reproducibility purposes (in case you need to redraw your graph with some modifications).

5.6.1 The 'par()' command

It is possible to manipulate any element in a graph, by using the graphics options. These are collected on the help page of `par()`. For example, if you want axis labels always to be horizontal, use the command `par(las=1)`. This will be in effect until a new graphics device is opened.

Look at the typewriter-version of the help-page with

```
help(par)
```

or better, use the the html-version through `Help` → `Html help` → `Packages` → `graphics` → `P`→ `par`.

It is a good idea to take a print of this (having set the text size to *smallest* because it is long) and carry it with you at any time to read in buses, cinema queues, during boring lectures etc. Don't despair, few R-users can understand what all the options are for.

`par()` can also be used to ask about the current plot, for example `par("usr")` will give you the exact extent of the axes in the current plot.

If you want more plots on a single page you can use the command

```
par(mfrow = c(2, 3))
```

This will give you a layout of 2 rows by 3 columns for the next 6 graphs you produce. The plots will appear by row, i.e. in the top row first. If you want the plots to appear columnwise, use `par( mfcol=c(2,3) )` (you still get 2 rows by 3 columns).

To restore the layout to a single plot per page use

```
par(mfrow = c(1, 1))
```

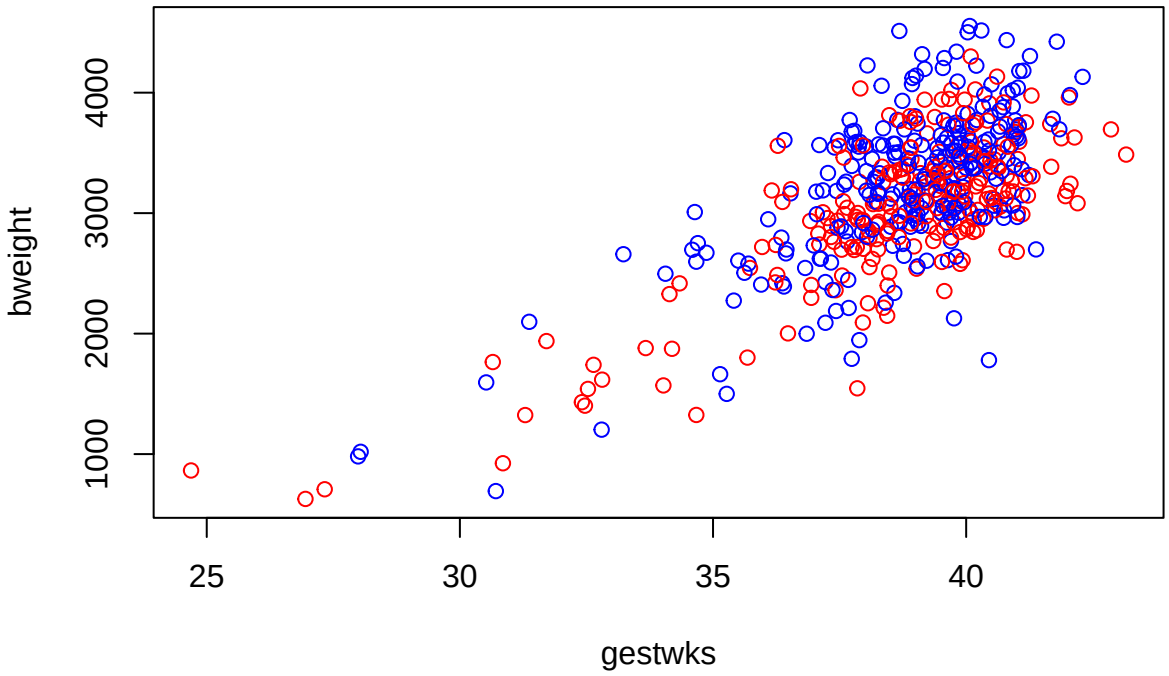
If you want a more detailed control over the layout of multiple graphs on a single page look at `?layout`.

5.7 Interacting with a plot

The `locator()` function allows you to interact with the plot using the mouse. Typing `locator(1)` shifts you to the graphics window and waits for one click of the left mouse button. When you click, it will return the corresponding coordinates.

You can use `locator()` inside other graphics functions to position graphical elements exactly where you want them. Recreate the birth-weight plot,

```
with(births, plot(gestwks, bweight, col = c("blue", "red")[sex]))
```



and then add the legend where you wish it to appear by typing

```
legend(
  locator(1),
  pch = 1,
  legend = c("Boys", "Girls"),
  col = c("blue", "red")
)
```

The `identify()` function allows you to find out which records in the data correspond to points on the graph. Try

```
with(births, identify(gestwks, bweight))
```

When you click the left mouse button, a label will appear on the graph identifying the row number of the nearest point in the data frame `births`. If there is no point nearby, `R` will print a warning message on the console instead. To end the interaction with the graphics window, right click the mouse: the `identify` function returns a vector of identified points.

- Use `identify()` to find which records correspond to the smallest and largest number of gestational weeks and view the corresponding records:

```
with(births, births[identify(gestwks, bweight), ])
```

Chapter 6

Analysis of hazard rates, their ratios and differences and binary regression

This exercise is *very* prescriptive, so you should make an effort to really understand everything you type into R. Consult the relevant slides of the lecture on *Poisson and Binary regression …*

6.1 Hand calculations for a single rate

Let λ be the true **hazard rate** or theoretical incidence rate of a given outcome event. Its estimator is the empirical **incidence rate** $\hat{\lambda} = D/Y$ = no. cases/person-years. Recall that the standard error of the empirical rate is $\mathrm{SE}(\hat{\lambda}) = \hat{\lambda}/\sqrt{D}$.

The simplest approximate 95% confidence interval (CI) for λ is given by

$$\hat{\lambda} \pm 1.96 \times SE(\hat{\lambda})$$

- Suppose 15 outcome events are observed during 5532 person-years in a given study cohort. Let's use R as a simple desk calculator to estimate the underlying hazard rate λ (in 1000 person-years; therefore 5.532) and to get the first version of an approximate confidence interval:

```
library(Epi)
options(digits = 4) #  to cut down decimal points in the output
```

```
D <- 15
Y <- 5.532 # thousands of years!
rate <- D / Y
SE.rate <- rate / sqrt(D)
c(rate, SE.rate, rate + c(-1.96, 1.96) * SE.rate)
```

6.2 Poisson model for a single rate with logarithmic link

You are able to estimate the hazard rate λ and compute its CI with a **Poisson regression model**, as described in the relevant slides in the lecture handout.

Poisson regression is a **generalized linear model** in which the **family**, *i.e.* the distribution of the response variable, is assumed to be the Poisson distribution. The most commonly applied **link function** in Poisson regression is the natural logarithm; log for short.

- A family object `poisreg`, a modified version of the original `poisson` family object, is available in package `Epi`. When using this, the response is defined as a *matrix* of two columns: numbers of cases D and person-years Y, these being combined into a matrix by `cbind(D,Y)`. No specification of `offset` is needed.

```
mreg <- glm(cbind(D, Y) ~ 1, family = poisreg(link = log))
ci.exp(mreg)
```

- If you want confidence interval for log rate

```
mreg <- glm(cbind(D, Y) ~ 1, family = poisreg(link = log))
ci.lin(mreg)[, c(1, 5, 6)]
```

In this course we endorse the use of family `poisreg` because of its advantages in more general settings.

6.3 Poisson model for a single rate with identity link

The approach leaning on having the number of cases D as the response and $\log(Y)$ as an offset, is limited only to models with log link. A major advantage of the `poisreg` family is that it allows a straighforward use of the *identity* link, too. With this link the response variable is the same, but the parameters to be directly estimated are now the rates itself and their differences, not the log-rates and their differences as with the log link.

- Fit a Poisson model with identity link to our simple data, and use `ci.lin()` to produce the estimate and the confidence interval for the hazard rate from this model:

```
mid <- glm(cbind(D, Y) ~ 1, family = poisreg(link = "identity"))
ci.lin(mid)
ci.lin(mid)[, c(1, 5, 6)]
```

How is the coefficient of this model interpreted? Verify that you actually get the same rate estimate and CI as in section 1.6.1, item 1.

6.4 Poisson model assuming the same rate for several periods

Now, suppose the events and person years are collected over three distinct periods.

- Read in the data and compute period-specific rates

```
Dx <- c(3, 7, 5)
Yx <- c(1.412, 2.783, 1.337)
Px <- 1:3
rates <- Dx / Yx
rates
```

- Using these data, fit the same model with log link as in section 1.6.2, assuming a common single hazard λ for the separate periods. Compare the result from the previous ones

```
m3 <- glm(cbind(Dx, Yx) ~ 1, family = poisreg(link = log))
ci.exp(m3)
```

- Now test whether the rates are the same in the three periods: Try to fit a model with the period as a factor in the model:

```
mp <- glm(cbind(Dx, Yx) ~ factor(Px), family = poisreg(link = log))
ci.exp(mp)
```

Compare the goodness-of-fit of the two models using `anova()` with the argument `test="Chisq"`:

```
anova(m3, mp, test = "Chisq")
```

Compare the test statistic to the deviance of the model `mp`. – What is the deviance indicating?

6.5 Analysis of rate ratio

We now switch to comparison of two rates λ_1 and λ_0, i.e. the hazard in an exposed group vs. that in an unexposed one.

Consider first estimation of the **hazard ratio** or the underlying *true* rate ratio $\rho = \lambda_1/\lambda_0$ between the groups. Suppose we have pertinent empirical data (cases and person-times) from both groups, (D_1, Y_1) and (D_0, Y_0). The point estimate of ρ is the empirical **incidence rate ratio**

$$\hat{\rho} = RR = \frac{\hat{\lambda}_1}{\hat{\lambda}_0} = \frac{D_1/Y_1}{D_0/Y_0}$$

Suppose you have 15 events during 5532 person-years in an unexposed group and 28 events during 4783 person-years in an exposed group:

- Calculate the incidence rates in the two groups, their ratio, and the CI of the true hazard ratio ρ by direct application of the above formulae:

```
D0 <- 15
D1 <- 28
Y0 <- 5.532
Y1 <- 4.783
```

- Now achieve this using a Poisson model. For that we first combine the group-specific numbers into pertinent vectors and specify a factor to represent the contrast between the exposed and the unexposed group

```
D <- c(D0, D1)
Y <- c(Y0, Y1)
expos <- 0:1
mm <- glm(cbind(D, Y) ~ factor(expos), family = poisreg(link = log))
```

What do the parameters mean in this model?

- You can extract the estimation results for exponentiated parameters in two ways, as before:

```
ci.exp(mm)
ci.lin(mm, Exp = TRUE)[, 5:7]
```

6.6 Analysis of rate difference

For the **hazard difference** $\delta = \lambda_1 - \lambda_0$, the natural estimator is the **incidence rate difference**

$$\hat{\delta} = \hat{\lambda}_1 - \hat{\lambda}_0 = D_1/Y_1 - D_0/Y_0 = \text{RD}.$$

Its variance is just the sum of the variances of the two rates

$$var(RD) = var(\hat{\lambda}_1) + var(\hat{\lambda}_0) = D_1/Y_1^2 + D_0/Y_0^2$$

- Use this formula to compute the point estimate of the rate difference λ and a 95% confidence interval for it:

```
R0 <- D0 / Y0
R1 <- D1 / Y1
RD <- diff(D / Y)
SED <- sqrt(sum(D / Y^2))
c(R1, R0, RD, SED, RD + c(-1, 1) * 1.96 * SED)
```

- Verify that this is the confidence interval you get when you fit an additive model (obtained by identity link) with exposure as a factor:

```
ma <- glm(cbind(D, Y) ~ factor(expos),
  family = poisreg(link = identity)
)
ci.lin(ma)[, c(1, 5, 6)]
```

6.7 Binary regression

Explore the factors associated with risk of low birth weight in 500 singleton births in a London Hospital. Indicator (lowbw) for birth weight less than 2500 g. Data available from the Epi package. Factors of interest are maternal hypertension (hyp), mother's age at birth over 35 years and sex of the baby.

Load the `Epi` package and the data set and look at its content

```
library(dplyr)
library(Epi)
data(births)
str(births)
```

- Because all variables are numeric we need first to do a little housekeeping. Two of them are directly converted into factors, and categorical versions are created of two continuous variables by function `cut()`.

```
births$hyp <- factor(births$hyp, labels = c("normal", "hyper"))
births$sex <- factor(births$sex, labels = c("M", "F"))
births$gest4 <- cut(births$gestwks,
  breaks = c(20, 35, 37, 39, 45), right = FALSE
)
births$maged <- ifelse(births$matage < 35, 0, 1)
```

- Cross tabulate (dplyr) counts of children by hypertension and low birth weight. calculate (mutate) proportions of low birth weigth children by hypertension.

```
births %>%
  count(hyp, lowbw) %>%
  group_by(hyp) %>% # now required with changes to dplyr::count()
  mutate(prop = prop.table(n))
```

- Estimate relative risk of low birth weight for mothers with hypertension compared to those without using binary regression.

```
m <- glm(lowbw ~ hyp, family = binomial(link = log), data = births)
ci.exp(m)
```

- Adjust relative risk of low birth and hypertension with the sex of children

```
m <- glm(lowbw ~ sex + hyp, family = binomial(link = log), data = births)
ci.exp(m)
```

- Adjust relative risk of low birth and hypertension with the sex of children and mother beeing over 35 years.

```
m <- glm(lowbw ~ maged + sex + hyp, family = binomial(link = log), data = births)
ci.exp(m)
```

6.8 Optional/Homework: Hand calculations and calculations using matrix tools

NB. This subsection requires some familiarity with matrix algebra. Do this only after you have done the other exercises of this session.

First some basic hand calculations.

- Suppose 15 outcome events are observed during 5532 person-years in a given study cohort. Let's use R as a simple desk calculator to estimate the underlying hazard rate λ (in 1000 person-years; therefore 5.532) and to get the first version of an approximate confidence interval:

```
library(Epi)
options(digits = 4) #  to cut down decimal points in the output
```

```
D <- 15
Y <- 5.532 # thousands of years!
rate <- D / Y
SE.rate <- rate / sqrt(D)
c(rate, SE.rate, rate + c(-1.96, 1.96) * SE.rate)
```

- Compute now the approximate confidence interval using the method based on log-transformation and compare the result with that in the previous item.

```
SE.logr <- 1 / sqrt(D)
EF <- exp(1.96 * SE.logr)
c(log(rate), SE.logr)
c(rate, EF, rate / EF, rate * EF)
```

- Calculate the incidence rates in the two groups, their ratio, and the CI of the true hazard ratio ρ by direct application of the above formulae:

```
D0 <- 15
D1 <- 28
Y0 <- 5.532
Y1 <- 4.783
R1 <- D1 / Y1
R0 <- D0 / Y0
RR <- R1 / R0
SE.lrr <- sqrt(1 / D0 + 1 / D1)
EF <- exp(1.96 * SE.lrr)
c(R1, R0, RR, RR / EF, RR * EF)
```

- Explore the function `ci.mat()`, which lets you use matrix multiplication (operator `'%*%'` in R) to produce a confidence interval from an estimate and its standard error (or CIs from whole columns of estimates and SEs):

```
ci.mat
ci.mat()
```

As you see, this function returns a 2×3 matrix (2 rows, 3 columns) containing familiar numbers.

- When you combine the single rate and its standard error into a row vector of length 2, i.e. a 1×2 matrix, and multiply this by the 2×3 matrix above, the computation returns a 1×3 matrix containing the point estimate and the confidence limit.

Apply this method to the single rate calculations in 1.6.1, first creating the 1×2 matrix and then performing the matrix multiplication.

```
rateandSE <- c(rate, SE.rate)
rateandSE
rateandSE %*% ci.mat()
```

- When the confidence interval is based on the log-rate and its standard error, the result is obtained by appropriate application of the exp-function on the pertinent matrix product

```
lograndSE <- c(log(rate), SE.logr)
lograndSE
exp(lograndSE %*% ci.mat())
```

- For computing the rate ratio and its CI as in 1.6.5, matrix multiplication with `ci.mat()` should give the same result as there:

```
exp(c(log(RR), SE.lrr) %*% ci.mat())
```

- The main argument in function `ci.mat()` is `alpha`, which sets the confidence level: $1 - \alpha$. The default value is `alpha = 0.05`, corresponding to the level $1 - 0.05 = 95\%$. If you wish to get the confidence interval for the rate ratio at the 90% level ($= 1 - 0.1$), for instance, you may proceed as follows:

```
ci.mat(alpha = 0.1)
exp(c(log(RR), SE.lrr) %*% ci.mat(alpha = 0.1))
```

- Now achieve this using a Poisson model. For that we first combine the group-specific numbers into pertinent vectors and specify a factor to represent the contrast between the exposed and the unexposed group

```
D <- c(D0, D1)
Y <- c(Y0, Y1)
expos <- 0:1
```

- Look again to the model used to analyse the rate ratio in. Often one would like to get simultaneously both the rates and the ratio between them. This can be achieved in one go using the *contrast matrix* argument `ctr.mat` to `ci.lin()` or `ci.exp()`. Try:

```
CM <- rbind(c(1, 0), c(1, 1), c(0, 1))
rownames(CM) <- c("rate 0", "rate 1", "RR 1 vs. 0")
CM
mm <- glm(D ~ factor(expos),
  family = poisson(link = log), offset = log(Y)
)
ci.exp(mm, ctr.mat = CM)
```

- Use the same machinery to the additive model to get the rates and the rate-difference in one go. Note that the annotation of the resulting estimates are via the column-names of the contrast matrix.

```
rownames(CM) <- c("rate 0", "rate 1", "RD 1 vs. 0")
ma <- glm(cbind(D, Y) ~ factor(expos),
  family = poisreg(link = identity)
)
ci.lin(ma, ctr.mat = CM)[, c(1, 5, 6)]
```

Chapter 7

Estimation of effects: simple and more complex

This exercise deals with analysis of metric and binary response variables. We start with simple estimation of effects of a binary, categorical or a numeric explanatory variable, the explanatory or exposure variable of interest. Then evaluation of potential modification and/or confounding by other variables is considered by stratification by and adjustment/control for these variables. For such tasks we utilize functions `lm()` and `glm()` which can be used for more general linear and generalized linear models. Finally, more complex spline modelling for the effect of a numeric exposure variable is illustrated.

7.1 Response and explanatory variables

Identifying the *response* or *outcome variable* correctly is the key to analysis. The main types are:

- Metric or continuous (a measurement with units).
- Binary ("yes" vs. "no", coded 1/0), or proportion.
- Failure in person-time, or incidence rate.

All these response variable are numeric.

Variables on which the response may depend are called *explanatory variables* or *regressors*. They can be categorical factors or numeric variables. A further important aspect of explanatory variables is the role they will play in the analysis.

- Primary role: exposure.
- Secondary role: confounder and/or effect-measure modifier.

The word **effect** is used here as a general term referring to ways of contrasting or comparing the expected values of the response variable at different levels of an explanatory variable. The main comparative measures or effect measures are:

- Differences in means for a metric response.
- Ratios of odds for a binary response.
- Ratios of rates for a failure or count response.

Other kinds of *contrasts* between exposure groups include (a) ratios of geometric means for positive-valued metric outcomes, (b) differences and ratios between proportions (risk difference and risk ratio), and (c) differences between incidence or mortality rates.

Note that in spite of using the causally loaded word *effect*, we treat *outcome regression* modelling here primarily with descriptive or predictive aims in mind. Traditionally, these types of models have also been used to estimate *causal effects* of exposure variables from the pertinent regression coefficients. More serious causal analysis is introduced in the lecture and practical on Tuesday morning, and modern approaches to estimate causal effects will be considered on Thursday afternoon.

7.2 Data set `births`

We shall use the `births` data to illustrate different aspects in estimating effects of various exposures on a metric response variable `bweight` = birth weight, recorded in grams.

1. Load the packages needed in this exercise and the data set, and look at its content

```
library(Epi)
library(mgcv)
data(births)
str(births)
```

2. We perform similar housekeeping tasks as in a previous exercise.

```
births$hyp <- factor(births$hyp, labels = c("normal", "hyper"))
births$sex <- factor(births$sex, labels = c("M", "F"))
births$maged <- cut(births$matage, breaks = c(22, 35, 44), right = FALSE)
births$gest4 <- cut(births$gestwks,
  breaks = c(20, 35, 37, 39, 45), right = FALSE)
```

3. Have a look at univariate summaries of the different variables in the data; especially the location and dispersion of the distribution of `bweight`.

```
summary(births)
with(births, sd(bweight))
```

7.3 Simple estimation with `lm()` and `glm()`

We are ready to analyze the effect of maternal hypertension `hyp` on `bweight`. A binary explanatory variable, like `hyp`, leads to an elementary two-group comparison of group means for a metric response.

1. Comparison of two groups is commonly done by the conventional t-test and the associated confidence interval.

```
with(births, t.test(bweight ~ hyp, var.equal = TRUE))
```

The P-value refers to the test of the null hypothesis that there is no effect of `hyp` on birth weight (somewhat implausible null hypothesis in itself!). However, `t.test()` does not provide the point estimate for the effect of `hyp`; only the test result and a confidence interval. – The estimated effect of `hyp` on birth weight, measured as a difference in means between hypertensive and normotensive mothers, is $3199 - 2768 = 431$ g.

2. The same task can easily be performed by `lm()` or by `glm()`. The main argument in both is the *model formula*, the left hand side being the response variable and the right hand side after $\sim$ defines the explanatory variables and their joint effects on the response. Here the only explanatory variable is the binary factor `hyp`. With `glm()` one specifies the `family`, i.e. the assumed distribution of the response variable. However, in case you use `lm()`, this argument is not needed, because `lm()` fits only models for metric responses assuming Gaussian distribution.

```
m1 <- glm(bweight ~ hyp, family = gaussian, data = births)
summary(m1)
```

3. Note the amount of output that `summary()` method produces. The point estimate plus confidence limits can, though, be concisely obtained by function `ci.lin()` found in `Epi` package.

```
round(ci.lin(m1)[, c(1, 5, 6)], 1)
```

7.4 Stratified effects, and interaction or effect-measure modification

We shall now examine whether and to what extent the *effect* of `hyp` on `bweight`, i.e. the mean difference between hypertensive and normotensive mothers, varies by `sex` without assigning causal interpretation to the estimated contrasts.

1. The following *interaction plot* shows how the mean `bweight` depends jointly on `hyp` and `gest4`

```
par(mfrow = c(1, 1))
with(births, interaction.plot(sex, hyp, bweight))
```

At face value it appears that the mean difference in `bweight` between hypertensive and normotensive mothers is somewhat bigger in boys than in girls.

2. Let us get numerical values for the mean differences in the two levels of `sex`. Stratified estimation of effects can be done by `lm()` as follows:

```
m3 <- lm(bweight ~ sex / hyp, data = births)
round(ci.lin(m3)[, c(1, 5, 6)], 1)
```

The estimated effects of `hyp` in the two strata defined by `sex` thus are −496 g in boys and −380 g among girls. The error margins of the two estimates are quite wide, though.

3. An equivalent model with an explicit *product term* or *interaction term* between `sex` and `hyp` is fitted as follows:

```
m3I <- lm(bweight ~ sex + hyp + sex:hyp, data = births)
round(ci.lin(m3I)[, c(1, 4, 5, 6)], 2)
```

From this output you would find a familiar estimate −231 g for girls vs. boys among normotensive mothers and the estimate −496 g contrasting hypertensive and normotensive mothers in the reference class of `sex`, i.e. among boys. The remaining coefficient is the estimate of the interaction effect such that $116.6 = -379.8 - (-496.4)$ g describes the contrast in the effect of `hyp` on `bweight` between girls and boys.

The P-value 0.46 as well as the wide confidence interval about zero of this interaction parameter suggest good compatibility of the data with the null hypothesis of no interaction between `hyp` and `sex`. Thus, there is insufficient evidence against the possibility of *effect(-measure) modification* by `sex` on the effect of `hyp`. On the other hand, this test is not very sensitive given the small sample size. Thus, in spite of obtaining a "non-significant" result, the possibility of a real effect-measure modification cannot be ignored based on these data only.

7.5 Controlling or adjusting for the effect of `hyp` for `sex`

The estimated effects of `hyp`: −496 in boys and −380 in girls, look quite similar (and the P-value against no interaction was quite large, too). Therefore, we may now proceed to estimate the overall effect of `hyp` *controlling for* – or *adjusting for* – `sex`.

1. Adjustment is done by adding `sex` to the model formula:

```
m4 <- lm(bweight ~ sex + hyp, data = births)
ci.lin(m4)[, c(1, 5, 6)]
```

The estimated effect of `hyp` on `bweight` adjusted for `sex` is thus −448 g, which is a weighted average of the sex-specific estimates. It is slightly different from the unadjusted estimate −431 g, indicating that there was no essential confounding by `sex` in the simple comparison of means. Note also, that the model being fitted makes the assumption that the estimated effect is the same for boys and girls.

Many people go straight ahead and control for variables which are likely to confound the effect of exposure without bothering to stratify first, but often it is useful to examine the possibility of effect-measure modification before that.

7.6 Numeric exposure, simple linear regression and checking assumptions

If we wished to study the effect of gestation time on the baby's birth weight then `gestwks` is a numeric exposure variable.

1. Assuming that the relationship of the response with `gestwks` is roughly linear (for a continuous response), % or log-linear (for a binary or failure rate response) we can estimate the linear effect of `gestwks` with `lm()` as follows:

```
m5 <- lm(bweight ~ gestwks, data = births)
ci.lin(m5)[, c(1, 5, 6)]
```

We have fitted a simple linear regression model and obtained estimates of the two regression coefficient: `intercept` and `slope`. The linear effect of `gestwks` is thus estimated by the slope coefficient, which is 197 g per each additional week of gestation.

At this stage it will be best to make some visual check concerning our model assumptions using `plot()`. In particular, when the main argument for the *generic function* `plot()` is a fitted `lm` object, it will provide you some common diagnostic graphs.

2. To check whether `bweight` goes up linearly with `gestwks` try

```
with(births, plot(gestwks, bweight))
abline(m5)
```

3. Moreover, take a look at the basic diagnostic plots for the fitted model.

```
par(mfrow = c(2, 2))
plot(m5)
```

What can you say about the agreement with data of the assumptions of the simple linear regression model, like linearity of the systematic dependence, homoskedasticity and normality of the error terms?

7.7 Penalized spline model

We shall now continue the analysis such that the apparently curved effect of `gestwks` is modelled by a *penalized spline*, based on the recommendations of Martyn in his lecture today.

You cannot fit a penalized spline model with `lm()` or `glm()`. Instead, function `gam()` in package `mgcv` can be used for this purpose. Make sure that you have loaded this package.

1. When calling `gam()`, the model formula contains expression '`s(X)`' for any explanatory variable `X`, for which you wish to fit a smooth function

```
mPs <- mgcv::gam(bweight ~ s(gestwks), data = births)
summary(mPs)
```

From the output given by `summary()` you find that the estimated intercept is equal to the overall mean birth weight in the data. The estimated residual variance is given by `Scale est.` or from subobject `sig2` of the fitted `gam` object. Taking square root you will obtain the estimated residual standard deviation: 445.2 g.

```
mPs$sig2
sqrt(mPs$sig2)
```

The degrees of freedom in this model are not computed as simply as in previous models, and they typically are not integer-valued. However, the fitted spline seems to consume only a little more degrees of freedom as an 3rd degree polynomial model would take.

2. A graphical presentation of the fitted curve together with the confidence and prediction intervals is more informative. Let us first write a short function script to facilitate the task. We utilize function `matshade()` in `Epi`, which creates shaded areas, and function `matlines()` which draws lines joining the pertinent end points over the x-values for which the predictions are computed.

```
plotFitPredInt <- function(xval, fit, pred, ...) {
  matshade(xval, fit, lwd = 2, alpha = 0.2)
  matshade(xval, pred, lwd = 2, alpha = 0.2)
  matlines(xval, fit, lty = 1, lwd = c(3, 2, 2), col = c("black", "blue", "blue"))
```

```
  matlines(xval, pred, lty = 1, lwd = c(3, 2, 2), col = c("black", "brown", "brown"))
}
```

3. Finally, create a vector of x-values and compute the fitted/predicted values as well as the interval limits at these points from the fitted model object utilizing function `predict()`. This function creates a matrix of three columns: (1) fitted/predicted values, (2) lower limits, (3) upper limits and make the graph:

```
nd <- data.frame(gestwks = seq(24, 45, by = 0.25))
pr.Ps <- predict(mPs, newdata = nd, se.fit = TRUE)
str(pr.Ps) # with se.fit=TRUE, only two columns: fitted value and its SE
fit.Ps <- cbind(
  pr.Ps$fit,
  pr.Ps$fit - 2 * pr.Ps$se.fit,
  pr.Ps$fit + 2 * pr.Ps$se.fit
)
pred.Ps <- cbind(
  pr.Ps$fit, # must add residual variance to se.fit^2
  pr.Ps$fit - 2 * sqrt(pr.Ps$se.fit^2 + mPs$sig2),
  pr.Ps$fit + 2 * sqrt(pr.Ps$se.fit^2 + mPs$sig2)
)
par(mfrow = c(1, 1))
with(births, plot(bweight ~ gestwks,
  xlim = c(24, 45),
  cex.axis = 1.5, cex.lab = 1.5
))
plotFitPredInt(nd$gestwks, fit.Ps, pred.Ps)
```

Compare this with the graph on slide 20 of the lecture we had. Are you happy with the end result?

7.8 Analysis of binary outcomes

Instead of investigating the distribution and determinants of birth weight as such, it is common in perinatal epidemiology to consider occurrence of low birth weight; whether birth weight is < 2.5 kg or not. Variable `lowbw` with values 1 and 0 in the `births` data represents that dichotomy. Some analyses on `lowbw` were already conducted in a previous practical. Here we illustrate further aspects of effect estimation and modelling binary outcome.

1. We start with simple tabulation of the prevalence of `lowbw` by maternal hypertension

```
stat.table(
  index = list(hyp, lowbw),
  contents = list(count(), percent(lowbw)),
  margins = TRUE, data = births
)
```

It seems that the prevalence for hypertensive mothers is about 18 percent points higher, or about three times as high as that for normotensive mothers

2. The three comparative measures of prevalences can be estimated by `glm()` with different link functions:

```
binRD <- glm(lowbw ~ hyp, family = binomial(link = "identity"), data = births)
round(ci.lin(binRD)[, c(1, 2, 5:6)], 3)
binRR <- glm(lowbw ~ hyp, family = binomial(link = "log"), data = births)
round(ci.lin(binRR, Exp = TRUE)[, c(1, 2, 5:7)], 3)
binOR <- glm(lowbw ~ hyp, family = binomial(link = "logit"), data = births)
round(ci.lin(binOR, Exp = TRUE)[, c(1, 2, 5:7)], 3)
```

Check that these results were quite compatible with the "about" estimates given in the previous item. How well is the odds ratio approximating the risk ratio here?

3. The prevalence of low birth weight is expected to be inversely related to gestational age (weeks), as is evident from simple tabulation

```
stat.table(
  index = list(gest4, lowbw),
  contents = list(count(), percent(lowbw)),
  margins = TRUE, data = births
)
```

4. Let's jump right away to spline modelling of this relationship

```
binm1 <- mgcv::gam(lowbw ~ s(gestwks), family = binomial(link = "logit"), data = births)
summary(binm1)
plot(binm1)
```

Inspect the figure. Would you agree, that the logit of the prevalence of outcome is almost linearly dependent on `gestwks`?

5. Encouraged by the result of the previous item, we continue the analysis with `glm()` and assuming logit-linearity

```
binm2 <- glm(lowbw ~ I(gestwks - 40), family = binomial(link = "logit"), data = births)
round(ci.lin(binm2, Exp = TRUE)[, c(1, 2, 5:7)], 3)
```

Inspect the results. How do you interpret the estimated coefficients and their exponentiated values?

6. Instead of fitted logits, it can be more informative to plot the fitted prevalences against `gestwks`, in which we utilize the previously created data frame `nd`

```
predm2 <- predict(binm2, newdata = nd, type = "response")
plot(nd$gestwks, predm2, type = "l")
```

The curve seems to cover practically the whole range of the outcome probability scale with a relatively steep slope between 33 to 37 weeks.

Chapter 8

Poisson regression & analysis of curved effects

This exercise deals with modelling incidence rates using Poisson regression. Our special interest is in estimating and reporting curved effects of continuous explanatory variables on the hazard rate, i.e. the theoretical incidence rate

We analyse the `testisDK` data found in the `Epi` package. It contains the numbers of cases of testis cancer and mid-year populations (person-years) in 1-year age groups in Denmark during 1943-96. In this analysis age and calendar time are first treated as categorical but finally, a penalized spline model is fitted.

8.1 Testis cancer: Data input and housekeeping

1. Load the packages and the data set, and inspect its structure:

```
library(Epi)
library(mgcv)
data(testisDK)
str(testisDK)
summary(testisDK)
head(testisDK)
```

2. There are nearly 5000 observations from 90 one-year age groups and 54 calendar years. To get a clearer picture of what's going on, we do some housekeeping. The age range will be limited to 15-79 years, and age and period are both categorized into 5-year intervals – following to the time-honoured practice in epidemiology.

```
tdk <- subset(testisDK, A > 14 & A < 80)
tdk$Age <- cut(tdk$A, br = 5 * (3:16), include.lowest = TRUE, right = FALSE)
nAge <- length(levels(tdk$Age))
tdk$Per <- cut(tdk$P,
  br = seq(1943, 1998, by = 5),
  include.lowest = TRUE, right = FALSE)
nPer <- length(levels(tdk$Per))
```

8.2 Some descriptive analysis

Computation and tabulation of incidence rates

1. Tabulate numbers of cases and person-years, and compute the incidence rates (per 100,000 y) in each 5 y × 5 y cell using `stat.table()`. Take a look at the structure of the thus created object

```
tab <- stat.table(
  index = list(Age, Per),
  contents = list(
    D = sum(D),
    Y = sum(Y / 1000),
    rate = ratio(D, Y, 10^5)
  ),
  margins = TRUE,
  data = tdk
)
str(tab)
```

The table is too wide to be readable as such. A graphical presentation is more informative.

2. From the saved table object `tab` you can plot an age-incidence curve for each period separately, after you have checked the structure of the table, so that you know the relevant dimensions in it. There is a function `rateplot()` in `Epi` that does default plotting of tables of rates (see the help page of `rateplot`)

```
str(tab)
par(mfrow = c(1, 1))
rateplot(
  rates = tab[3, 1:nAge, 1:nPer], which = "ap", ylim = c(1, 30),
  age = seq(15, 75, 5), per = seq(1943, 1993, 5),
  col = heat.colors(16), ann = TRUE
)
```

What can you conclude about the trend in age-specific incidence rates over calendar time? What about the effect of age; is there any common pattern in the age-incidence curves across the periods?

8.3 Age and period as categorical factors

We shall first fit a Poisson regression model with log link on age and period model in the traditional way, in which both factors are treated as categorical. The model is additive on the log-rate scale. It is useful to scale the person-years to be expressed in 10^5 y.

1. In fitting the model we utilize the `poisreg` family object found in package `Epi`.

```
tdk$Y <- tdk$Y / 100000
mCat <- glm(cbind(D, Y) ~ Age + Per,
  family = poisreg(link = log), data = tdk )
round(ci.exp(mCat), 2)
```

What do the estimated rate ratios tell about the age and period effects?

2. A graphical inspection of point estimates and confidence intervals can be obtained as follows. In the beginning it is useful to define shorthands for the pertinent mid-age and mid-period values of the different intervals

```
aMid <- seq(17.5, 77.5, by = 5)
pMid <- seq(1945, 1995, by = 5)
par(mfrow = c(1, 2))
matplot(aMid, rbind(c(1,1,1), ci.exp(mCat)[2:13, ]), type = "o", pch = 16,
   log = "y", cex.lab = 1.5, cex.axis = 1.5, col= c("black", "blue", "blue"),
  xlab = "Age (years)", ylab = "Rate ratio" )
matplot(pMid, rbind(c(1,1,1), ci.exp(mCat)[14:23, ]), type = "o", pch = 16,
  log = "y", cex.lab = 1.5, cex.axis = 1.5, col=c("black", "blue", "blue"),
  xlab = "Calendar year - 1900", ylab = "Rate ratio" )
```

3. In the fitted model the reference category for each factor was the first one. As age is the dominating factor, it may be more informative to remove the intercept from the model. As a consequence the age

effects describe fitted rates at the reference level of the period factor. For the latter one could choose the middle period 1968-72 using `Relevel()`.

```
tdk$Per70 <- Relevel(tdk$Per, ref = 6)
mCat2 <- glm(cbind(D, Y) ~ -1 + Age + Per70,
  family = poisreg(link = log), data = tdk )
round(ci.exp(mCat2), 2)
```

4. Let us also plot estimates from the latter model, too.

```
par(mfrow = c(1, 2))
matplot(aMid, rbind(c(1,1,1), ci.exp(mCat2)[2:13, ]), type = "o", pch = 16,
   log = "y", cex.lab = 1.5, cex.axis = 1.5, col=c("black", "blue", "blue"),
  xlab = "Age (years)", ylab = "Rate" )
matplot(pMid, rbind(ci.exp(mCat2)[14:18, ], c(1,1,1), ci.exp(mCat2)[19:23, ]),
        type = "o", pch = 16, log = "y", cex.lab = 1.5, cex.axis = 1.5,
        col=c("black", "blue", "blue"),
  xlab = "Calendar year - 1900", ylab = "Rate ratio" )
abline(h = 1, col = "gray")
```

8.4 Generalized additive model with penalized splines

It is obvious that the age effect on the log-rate scale is highly non-linear. Yet, it is less clear whether the true period effect deviates from linearity. Nevertheless, there are good reasons to try fitting smooth continuous functions for both time scales.

1. As the next task we fit a generalized additive model for the log-rate on continuous age and period applying penalized splines with default settings of function `gam()` in package `mgcv`. In this fitting an *optimal* value for the penalty parameter is chosen based on an AIC-like criterion known as UBRE ('Un-Biased Risk Estimator')

```
library(mgcv)
mPen <- mgcv::gam(cbind(D, Y) ~ s(A) + s(P),
  family = poisreg(link = log), data = tdk
)
summary(mPen)
```

The summary is quite brief, and the only estimated coefficient is the intercept, which sets the baseline level for the log-rates, against which the relative age effects and period effects will be contrasted. On the rate scale the baseline level 5.53 per 100000 y is obtained by `exp(1.7096)`.

2. See also the default plot for the fitted curves (solid lines) describing the age and the period effects which are interpreted as contrasts to the baseline level on the log-rate scale.

```
par(mfrow = c(1, 2))
plot(mPen, se=2, seWithMean = TRUE)
```

The dashed lines describe the approximate 95% confidence band for the pertinent curve. One could get the impression that year 1968 would be some kind of reference value for the period effect, like period 1968-72 chosen as the reference in the categorical model previously fitted. This is not the case, however, because `gam()` by default parametrizes the spline effects such that the reference level, at which the spline effect is nominally zero, is the overall *grand mean* value of the log-rate in the data. This corresponds to the principle of *sum contrasts* (`contr.sum`) for categorical explanatory factors.

From the summary you will also find that the degrees of freedom value required for the age effect is nearly the same as the default dimension $k - 1 = 9$ of the part of the model matrix (or basis) initially allocated for each smooth function. (Here k refers to the relevant argument that determines the basis dimension when specifying a smooth term by `s()` in the model formula). On the other hand the period effect takes just about 3 df.

3. It is a good idea to do some diagnostic checking of the fitted model

```
par(mfrow = c(2, 2))
gam.check(mPen)
```

The four diagnostic plots are analogous to some of those used in the context of linear models for Gaussian responses, but not all of them may be as easy to interpret. – Pay attention to the note given in the printed output about the value of `k`.

4. Let us refit the model but now with an increased `k` for age:

```
mPen2 <- mgcv::gam(cbind(D, Y) ~ s(A, k = 20) + s(P),
  family = poisreg(link = log), data = tdk
)
summary(mPen2)
par(mfrow = c(2, 2))
gam.check(mPen2)
```

With this choice of `k` the df value for age became about 11, which is well below $k - 1 = 19$. Let us plot the fitted curves from this fitting, too

```
par(mfrow = c(1, 2))
plot(mPen2, seWithMean = TRUE)
abline(v = 1968, h = 0, lty = 3)
```

There does not seem to have happened any essential changes from the previously fitted curves, so maybe 8 df could, after all, be quite enough for the age effect.

5. Graphical presentation of the effects using `plot.gam()` can be improved. For instance, we may present the age effect to describe the *mean* incidence rates by age, averaged over the whole time span of 54 years. This is obtained by adding the estimated intercept to the estimated smooth curve for the age effect and showing the antilogarithms of the ordinates of the curve. For that purpose we need to extract the intercept and modify the labels of the y-axis accordingly. The estimated period curve can also be expressed in terms of relative indidence rates in relation to the fitted baseline rate, as determined by the model intercept.

```
par(mfrow = c(1, 2))
icpt <- coef(mPen2)[1] #  estimated intecept
plot(mPen2,
  seWithMean = TRUE, select = 1, rug = FALSE,
  yaxt = "n", ylim = c(log(1), log(20)) - icpt,
  xlab = "Age (y)", ylab = "Mean rate (/100000 y)"
)
axis(2, at = log(c(1, 2, 5, 10, 20)) - icpt, labels = c(1, 2, 5, 10, 20))
plot(mPen2,
  seWithMean = TRUE, select = 2, rug = FALSE,
  yaxt = "n", ylim = c(log(0.4), log(2)),
  xlab = "Calendat year", ylab = "Relative rate"
)
axis(2, at = log(c(0.5, 0.75, 1, 1.5, 2)), labels = c(0.5, 0.75, 1, 1.5, 2))
abline(v = 1968, h = 0, lty = 3)
```

Homework You could continue the analysis of these data by fitting an age-cohort model as an alternative to the age-period model, as well as an age-cohort-period model utilizing function `apc.fit()` in `Epi`. See (http://bendixcarstensen.com/APC/) for details.

Chapter 9

Causal inference

9.1 Proper adjustment for confounding in regression models

The first exercise of this session will ask you to simulate some data according to pre-specified causal structure (don't take the particular example too seriously) and see how you should adjust the analysis to obtain correct estimates of the causal effects.

Suppose one is interested in the effect of beer-drinking on body weight. Let's *assume* that in addition to the potential effect of beer on weight, the following is true in reality:

- Beer-drinking has an effect on the body weight.
- Men drink more beer than women
- Men have higher body weight than women
- People with higher body weight tend to have higher blood pressure
- Beer-drinking increases blood pressure

The task is to simulate a dataset in accordance with this model, and subsequently analyse it to see, whether the results would allow us to conclude the true association structure.

- Sketch a DAG (not necessarily with R) to see, how should one generate the data
- Suppose the actual effect sizes are following:
- The probability of beer-drinking is 0.2 for females and 0.7 for males
- Men weigh on average $10kg$ more than women.
- One kg difference in body weight corresponds in average to $0.5mmHg$ difference in (systolic) blood pressures.
- Beer-drinking increases blood pressure by $10mmHg$ in average.
- Beer-drinking has **no** effect on body weight.

The `R` commands to generate the data are:

```
bdat <- data.frame(sex = c(rep(0, 500), rep(1, 500)))
# a data frame with 500 females, 500 males
bdat$beer <- rbinom(1000, 1, 0.2 + 0.5 * bdat$sex)
bdat$weight <- 60 + 10 * bdat$sex + rnorm(1000, 0, 7)
bdat$bp <-
  110 + 0.5 * bdat$weight + 10 * bdat$beer + rnorm(1000, 0, 10)
```

- Now fit the following models for body weight as dependent variable and beer-drinking as independent variable. Look, what is the estimated effect size:
- Unadjusted (just simple linear regression)

- Adjusted for sex
- Adjusted for sex and blood pressure
- What would be the conclusions on the effect of beer on weight, based on the three models? Do they agree? Which (if any) of the models gives an unbiased estimate of the actual causal effect of interest?
- How can the answer be seen from the graph?
- Now change the data-generation algorithm so, that in fact beer-drinking does increase the body weight by 2kg. Look, what are the conclusions in the above models now. Thus the data is generated as before, but the weight variable is computed as:

```
bdat$weight <-
  60 + 10 * bdat$sex + 2 * bdat$beer + rnorm(1000, 0, 7)
```

- Suppose one is interested in the effect of beer-drinking on blood pressure instead, and is fitting a) an unadjusted model for blood pressure, with beer as an only covariate; b) a model with beer and sex as covariates. Would either a) or b) give an unbiased estimate for the effect? (You may double-check whether the simulated data is consistent with your answer).

9.2 DAG tools in the package dagitty

There is a software *DAGitty* (http://www.dagitty.net/) and also an R package *dagitty* that can be helpful in dealing with DAGs. Let's try to get the answer to the previous exercise using this package.

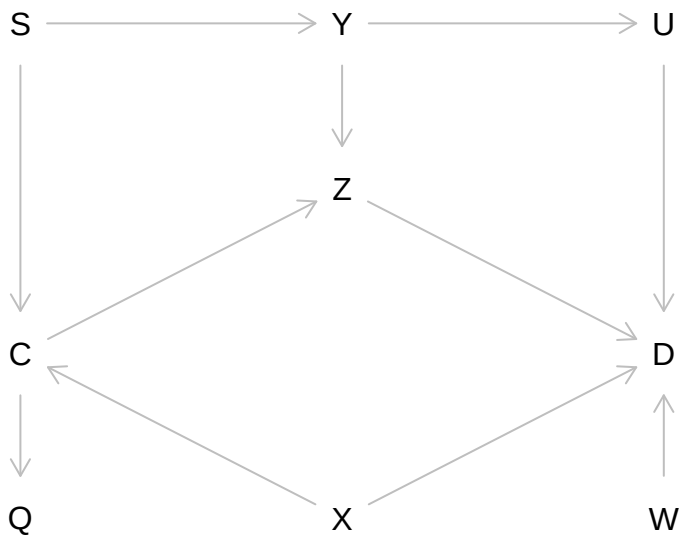
```
if (!("dagitty" %in% installed.packages())){
  install.packages("dagitty")
}
library(dagitty)
```

Let's recreate the graph on the lecture slide 28 (but omitting the direct causal effect of interest, $C \to D$):

```
g <- dagitty("dag {
    C <- S -> Y -> U -> D
    C -> Z <- Y
    Z -> D
    C <- X -> D
    C -> Q
    W -> D
  }")
plot(g)
```

To get a more similar look as on the slide, we must supply the coordinates (x increases from left to right, y from top to bottom):

```
coordinates(g) <-
  list(
    x =
      c(
        S = 1, C = 1, Q = 1, Y = 2, Z = 2,
        X = 2, U = 3, D = 3, W = 3
      ),
    y =
      c(
        U = 1, Y = 1, S = 1, Z = 2, C = 3,
        D = 3, X = 4, W = 4, Q = 4
      )
  )
plot(g)
```



Let's look at all possible paths from C to D:

```
paths(g, "C", "D")
```

As you see, one path contains a collider and is therefore a *closed* path and the others are *open*.

Let's identify the minimal sets of variables needed to adjust the model for D for, to obtain an unbiased estimate of the effect of C. You can specify, whether you want to estimate direct or total effect of C:

```
adjustmentSets(
  g, exposure = "C", outcome = "D", effect = "direct"
)
adjustmentSets(
  g, exposure = "C", outcome = "D", effect = "total"
)
```

Thus, for total effect estimation one should adjust for X and either Y or S, whereas for direct effect estimation, one would also need to adjust for Z.

You can verify that, these are the variables that will block all open paths from C to D.

Now try to do the *beer-weight* exercise using *dagitty*:

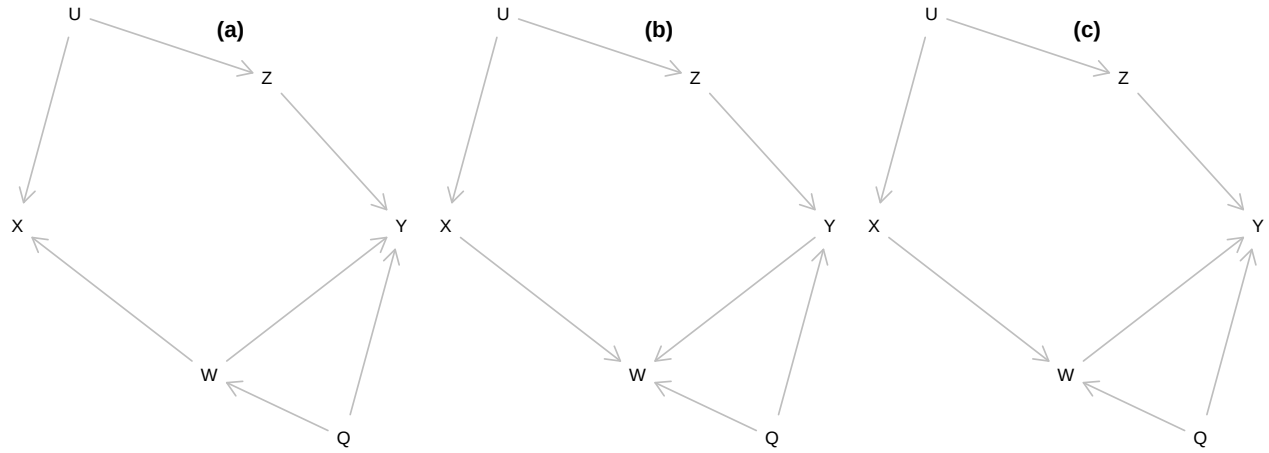
- Create the DAG and plot it
- What are the paths from WEIGHT to BEER?
- Will you get the same recommendation for the adjustment variable selection as you found before?

9.3 Identifying the true DAG for the data

The following code creates three DAGs

```
par(mfrow=c(1,3))
g1 <- dagitty("dag {
    U -> Z -> Y
    U -> X
    W -> Y
    Q -> W -> X
    Q -> Y
    }")
g2 <- dagitty("dag {
```

```
    U -> Z -> Y -> W
    U -> X -> W
    Q -> W
    Q -> Y
    }")
g3 <- dagitty("dag {
    U -> Z -> Y
    U -> X -> W  -> Y
    Q -> W
    Q -> Y
    }")
coord <-
  list( x = c(X=1, U = 1.3, W = 2, Z = 2.3, Q = 2.7, Y=3),
        y = c(U=1,   Z=1.3, X=2, Y=2,   W=2.7,  Q=3)
  )
coordinates(g1)<-coordinates(g2)<-coordinates(g3)<-coord
plot(g1)
title("(a)")
plot(g2)
title("(b)")
plot(g3)
title("(c)")
```



Now, the following script generates a dataset:

```
source("data/gendata.r")
head(dat)
```

```
##        Q      W      X      Z       Y
## 1 -0.397  1.056  0.199 -0.372  -4.198
## 2  0.460  2.393 -1.647 -3.004 -10.667
## 3  0.164 -0.364  0.205 -0.852  -1.830
## 4  1.584  3.285 -9.941  2.752  -4.121
## 5  1.048  1.637  0.783 -2.047  -4.378
## 6  1.895  3.272 -3.900 -2.313  -9.871
```

Exercise: Using linear models, try to identify, which of the three DAGs has been used to generate the data.

```
adjustmentSets(g1,"X","Y", effect="direct")
```

```
## { W, Z }
## { U, W }
```

```r
summary(lm(Y~X+W+Z,data=dat))
```

```
##
## Call:
## lm(formula = Y ~ X + W + Z, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.9888 -0.7441  0.0096  0.7657  3.5928
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.033168   0.024551   1.351    0.177
## X           -0.001691   0.014877  -0.114    0.909
## W           -2.599607   0.031987 -81.270   <2e-16 ***
## Z            1.014166   0.021197  47.846   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.097 on 1996 degrees of freedom
## Multiple R-squared:  0.9686, Adjusted R-squared:  0.9686
## F-statistic: 2.054e+04 on 3 and 1996 DF,  p-value: < 2.2e-16
```

```r
# no X effect: this DAG is possible

adjustmentSets(g2,"X","Y", effect="direct")
```

```
## { Z }
## { U }
```

```r
summary(lm(Y~X+Z,data=dat))
```

```
##
## Call:
## lm(formula = Y ~ X + Z, data = dat)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -7.480 -1.568 -0.107  1.580  8.730
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.10067    0.05092   1.977   0.0482 *
## X            1.13137    0.01077 105.040   <2e-16 ***
## Z            2.38032    0.02680  88.830   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.277 on 1997 degrees of freedom
## Multiple R-squared:  0.8648, Adjusted R-squared:  0.8647
## F-statistic:  6387 on 2 and 1997 DF,  p-value: < 2.2e-16
```

```r
# X effect is significant: this DAG is not likely

adjustmentSets(g3,"X","Y", effect="direct")
```

```
## { Q, W, Z }
## { Q, U, W }
```

```
summary(lm(Y~X+Q+W+Z,data=dat))
```

```
##
## Call:
## lm(formula = Y ~ X + Q + W + Z, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.7210 -0.6425  0.0031  0.6596  3.1985
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.019576   0.022168   0.883    0.377
## X           -0.006038   0.013429  -0.450    0.653
## Q            1.071881   0.050242  21.335   <2e-16 ***
## W           -3.033392   0.035312 -85.903   <2e-16 ***
## Z            1.003544   0.019138  52.437   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9903 on 1995 degrees of freedom
## Multiple R-squared:  0.9745, Adjusted R-squared:  0.9744
## F-statistic: 1.902e+04 on 4 and 1995 DF,  p-value: < 2.2e-16
```

```
# no X effect: this DAG is possible

adjustmentSets(g1,"Z","W")
```

```
##  {}
```

```
summary(lm(W~Z,data=dat))
```

```
##
## Call:
## lm(formula = W ~ Z, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.5057 -1.5539  0.0044  1.4534  8.0072
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.02154    0.04919  -0.438    0.661
## Z            0.01908    0.02238   0.852    0.394
##
## Residual standard error: 2.2 on 1998 degrees of freedom
## Multiple R-squared:  0.0003635,  Adjusted R-squared:  -0.0001368
## F-statistic: 0.7266 on 1 and 1998 DF,  p-value: 0.3941
```

```
# no Z effect: this DAG is possible

adjustmentSets(g3,"Z","W")
```

```
## { X }
## { U }
```

```
summary(lm(W~Z+X,data=dat))
```

```
##
## Call:
## lm(formula = W ~ Z + X, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.5471 -0.4981  0.0016  0.5336  2.3622
##
## Coefficients:
##              Estimate Std. Error  t value Pr(>|t|)
## (Intercept) -0.025965   0.017165   -1.513    0.131
## Z           -0.525522   0.009033  -58.178   <2e-16 ***
## X           -0.435859   0.003631 -120.044   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7676 on 1997 degrees of freedom
## Multiple R-squared:  0.8783, Adjusted R-squared:  0.8782
## F-statistic:  7208 on 2 and 1997 DF,  p-value: < 2.2e-16
```

```
# significant Z effect: this DAG is possible
```

If you have made your decision, you may check the script 'gendata.r' to see, whether your guess was right.

9.4 Instrumental variables estimation: Mendelian randomization

Suppose you want to estimate the effect of Body Mass Index (BMI) on blood glucose level (associated with the risk of diabetes). Let's conduct a simulation study to verify that when the exposure-outcome association is confounded, but there is a valid instrument (genotype), one obtains an unbiased estimate of the causal effect.

- Start by generating the genotype variable as *Binomial(2,p)*, with $p = 0.2$ (and look at the resulting genotype frequencies):

```
n <- 10000
mrdat <- data.frame(G = rbinom(n, 2, 0.2))
table(mrdat$G)
```

- Also generate the confounder variable U

```
mrdat$U <- rnorm(n)
```

- Generate a continuous (normally distributed) exposure variable BMI so that it depends on G and U. Check with linear regression, whether there is enough power to get significant parameter estimates. For instance:

```
mrdat$BMI <- with(mrdat, 25 + 0.7 * G + 2 * U + rnorm(n))
```

- Finally generate Y ("Blood glucose level") so that it depends on BMI and U (but not on G).

```
mrdat$Y <-
  with(mrdat, 3 + 0.1 * BMI - 1.5 * U + rnorm(n, 0, 0.5))
```

- Verify, that simple regression model for Y, with BMI as a covariate, results in a biased estimate of the causal effect (parameter estimate is different from what was generated)

How different is the estimate from 0.1?

- Estimate a regression model for Y with two covariates, G and BMI. Do you see a significant effect of G? Could you explain analytically, why one may see a significant parameter estimate for G there?
- Find an IV (instrumental variables) estimate, using G as an instrument, by following the algorithm in the lecture notes (use two linear models and find a ratio of the parameter estimates). Does the estimate get

closer to the generated effect size?

```
mgx <- lm(BMI ~ G, data = mrdat)
ci.lin(mgx) # check the instrument effect
bgx <- mgx$coef[2] # save the 2nd coefficient (coef of G)
mgy <- lm(Y ~ G, data = mrdat)
ci.lin(mgy)
bgy <- mgy$coef[2]
causeff <- bgy / bgx
causeff # closer to 0.1?
```

- A proper simulation study would require the analysis to be run several times, to see the extent of variability in the parameter estimates. A simple way to do it here would be using a `for`-loop. Modify the code as follows (exactly the same commands as executed so far, adding a few lines of code to the beginning and to the end):

```
n <- 10000
# initializing simulations:
# 30 simulations (change it, if you want more):
nsim <- 30
mr <- rep(NA, nsim) # empty vector for the outcome parameters
for (i in 1:nsim) { # start the loop
  ## Exactly the same commands as before:
  mrdat <- data.frame(G = rbinom(n, 2, 0.2))
  mrdat$U <- rnorm(n)
  mrdat$BMI <-
    with(mrdat, 25 + 0.7 * G + 2 * U + rnorm(n))
  mrdat$Y <-
    with(mrdat, 3 + 0.1 * BMI - 1.5 * U + rnorm(n, 0, 0.5))
  mgx <- lm(BMI ~ G, data = mrdat)
  bgx <- mgx$coef[2]
  mgy <- lm(Y ~ G, data = mrdat)
  bgy <- mgy$coef[2]
  # Save the i'th parameter estimate:
  mr[i] <- bgy / bgx
} # end the loop
```

Now look at the distribution of the parameter estimate:

```
summary(mr)
```

- (*optional*) Change the code of simulations so that the assumptions are violated: add a weak direct effect of the genotype G to the equation that generates Y:

```
mrdat$Y <-
  with(
    mrdat,
    3 + 0.1 * BMI - 1.5 * U + 0.05 * G + rnorm(n, 0, 0.5)
  )
```

Repeat the simulation study to see, what is the bias in the average estimated causal effect of BMI on Y.

- (*optional*) Using library `sem` and function `tsls`, one can obtain a two-stage least squares estimate for the causal effect and also the proper standard error. Do you get the same estimate as before?

```
if (!("sem" %in% installed.packages())) install.packages("sem")
library(sem)
summary(tsls(Y ~ BMI, ~G, data = mrdat))
```

(There are also several other R packages for IV estimation and Mendelian Randomization (*MendelianRandomization* for instance))

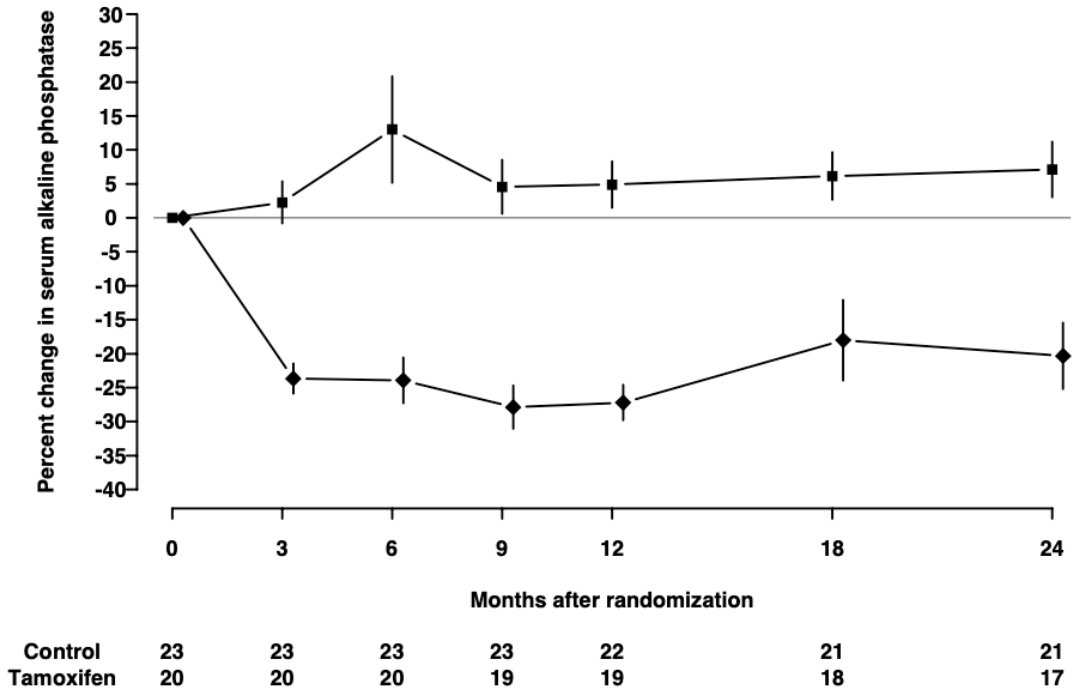
9.5 Why are simulation exercises useful for causal inference?

If we simulate the data, we know the data-generating mechanism and the *true* causal effects. So this is a way to check, whether an analysis approach will lead to estimates that correspond to what is generated. One could expect to see similar phenomena in real data analysis, if the data-generation mechanism is similar to what was used in simulations.

Chapter 10

Graphics with ggplot2

The plot below is from randomized study of the effect of Tamoxifen treatment on bone mineral metabolism in a group of patients who were treated for breast cancer (Kristensen et al, 1994). It is reproduced with permission from the author Peter Dalgaard who has also shared the original data.



The purpose of this exercise is to show you how to build a similar plot in R using the `ggplot2` package.

The outcome of the study is the percentage change in serum alkaline phosphate compared with the baseline when the participants first entered the study. The first measurement, on the left of the plot, is zero by definition. Then as we move from left to right, the plot shows the relative changes for the two groups of women: one receiving the anti-cancer drug tamoxifen; the other receiving standard treatment which is referred to as the *control* group. Each woman was randomly allocated to one of these groups on entry to the study. The women returned for follow-up visits until the end of the study at 24 months.

Take a moment to study the graph and try to identify the different graphical elements that we will need to reproduce. Write down a description of these elements and compare it to my summary below.

- The trajectory of the outcome variable is represented by points connected by straight lines. One point represents the mean value for one follow-up visit in one of the groups.
- Separate trajectories are plotted for the treatment and control groups.
- Vertical error bars give a visual impression of the uncertainty associated with each point.
- A horizontal line at zero is drawn as a reference so that we can see the size of the change relative to baseline.
- The x-axis has tick marks at follow-up visits, which are irregularly spaced. Visits occurred every 3 months for the first year, then every 6 months for the second year.
- The y-axis extends beyond the range of the data from -40% to +30%.
- A table underneath the x-axis shows the number of women in the treatment and control groups at each visit. These numbers go down as the study progresses. This phenomenon is extremely common in medical studies and is called "loss to follow-up".

These are the important graphical elements. You may have spotted some other features of the graph, but we will focus on trying to incorporate the features listed above.

10.1 Data

The data are available in the file `alkfos.csv`. This is a text file in comma-separated variable format. It can be read into R using the `read.csv()` function.

```
alkfos <- read.csv("./data/alkfos.csv")
```

There are 14 rows in the data frame `alkfos`: one for each combination of the 7 visits and the 2 groups. The variables in the data frame are:

- `time` The time since randomization for each follow-up visit.
- `mean` The outcome variable representing the mean percentage change in serum alkaline phosphate.
- `available` The number of women still participating in the study.
- `treat` The treatment group: a character vector taking values "tamoxifen" for the treatment group and "control" for the control group.
- `sem` The standard error of the mean used to calculate the length of the error bars.

10.2 Building the plot

10.2.1 The ggplot2 package

In this exercise, we will be using the R package ggplot2. The "gg" stands for "Grammar of Graphics", which is a conceptual framework for understanding statistical graphics created by Leland Wilkinson.

The core function in the ggplot2 package is `ggplot()`. This function requires two arguments:

- `data` is the name of the data frame that we will take the variables from.
- `mapping` is the *aesthetic mapping* between variables in the data frame and graphical elements that we want to plot. In this example:
  - y-axis is the mean of the percent change in alkaline phosphate represented by the variable `mean`;
  - x-axis is the visit time represented by the variable `time`.

We make the mapping in a call to the function `aes()`.

The code below creates a basic plot and assigns it to the object `p0`.

```
library(ggplot2, quietly=TRUE)
p0 <- ggplot(data=alkfos, mapping=aes(x=time, y=mean))
```

Running the code creates the plot but does not display it. This is an important difference between base graphics and ggplot graphics:

- Base graphics functions do not return an R object but will always display a plot on the current graphics device.
- GGplot graphics functions return an R graphical object, which will only be displayed if we ask for it.

There are two ways to display a graphical object in R:

1. If we have saved the result to an object, call the `print()` or `show()` functions.
2. If we have not saved the result, the graphical object will be displayed in the same way that a regular R object is printed to the screen.

The code chunk below displays the object `p0` created in the chunk above.

```
show(p0)
```

You may find the results a little disappointing. The `ggplot()` function has drawn x- and y-axes that span the range of the data, but otherwise has drawn no points or lines. This is another important difference between ggplot2 and base graphics. Whereas base plotting functions will usually create a sensible default plot, `ggplot()` makes no assumptions about what you want.

We will recreate the plot in *layers*. Layers are added to a ggplot object by calling additional functions and adding their return value to the basic plot using the `+` symbol.

10.2.2 Geometries

The most important layers are *geometries*, which describe how to display the data in the plot. Geometry functions in ggplot2 all start with the prefix `geom_`.

10.2.2.1 Points and lines

We want to plot points representing the mean of the outcome at each visit. This is done with the function `geom_point()`. We also want to plot lines between successive points in time. This is done with the function `geom_line()`. The code chunk below adds point and line geometries to our basic plot `p0` and saves the result to the new object `p1`.

```
p1 <- p0 + geom_point() + geom_line()
show(p1)
```

Again, this is not quite what we want. The line geometry is mixing up observations from the treatment and control groups, but we want separate lines for each group. To obtain this we need to add an additional aesthetic mapping inside the call to `ggplot()`. The aesthetic `group` gives the name of the variable in the data frame that distinguishes observations from different groups.

Modify the code chunk below so that it includes the `group` aesthetic to recreate plot `p1` correctly.

```
p1 <- ggplot(data=alkfos, mapping=aes(x=time, y=mean)) +
  geom_point() + geom_line()
show(p1)
```

10.2.2.2 Error bars

Now we can add the error bars. These can be added using the `linerange` geometry. This geometry needs some new aesthetic mappings. It will inherit the `x` mapping from the call to `ggplot()`. However, it needs new mappings `ymin` and `ymax` giving the lower and upper points of the error bar on the y-axis. We could go back and modify the aesthetic mappings in the call to `ggplot()`. However, we can also supply aesthetic mappings inside the call to a geometry function. It can be useful to do this when an aesthetic is only used by one geometry (which is the case here).

Modify the code chunk below so that the `linerange` geometry has its own aesthetic mapping for `ymin` and `ymax`.

- `ymin` is the mean minus the standard error
- `ymax` is the mean plus the standard error

Note that the error bars are *not confidence intervals*. In statistical graphics it is very common to use plus/minus the standard error to represent the uncertainty in the data without making any statistical claims about coverage.

```
p2 <- p1 + geom_linerange(...)
show(p2)
```

Compare the y-axes for plots `p1` and `p2`. Note that when we add the error bars, the y-axis automatically expanded the limits of the y-axis to include the highest and lowest values on the error bars.

10.2.2.3 Horizontal reference line

The last geometry we want to add is the horizontal reference line at zero. This is added using the `hline` geometry. We must supply the argument `yintercept` which determines where the horizontal line intersects the y-axis.

We should distinguish the reference line from the lines used to plot the data, otherwise it may distract the viewer. We could do this by changing the colour of the reference line. But here we will use a thinner line using the optional argument `linewidth` to the hline geometry. The default value of `linewidth` is 1. A value less than 1 draws a thinner line; a value greater than 1 draws a thicker line. Choose a suitable value for your plot.

```
p3 <- p2 + geom_hline(...)
show(p3)
```

10.2.3 Scales

Next we want to fix the x- and y-axes. The `ggplot()` function will provide default axes for you, which are often good enough. But in this case we want custom axes. These are provided by scale functions. In fact axes are just one kind of scale. In the grammar of graphics framework, a scale is a way to annotate any quantity that represents a dimension of the data. This also includes shapes, colours, and sizes.

Both axes are continuous. Therefore we use the `scale_x_continuous()` and `scale_y_continuous()` functions to create custom axes. These scale functions have the same arguments, but they affect the x- and y-axes, respectively. The arguments you want to supply are

- `breaks` A numeric vector of positions on the axis where you want tick marks.
- `limits` A numeric vector of length 2 giving the minimum and maximum values to be displayed on the axis.

Modify the code chunk below to create the custom axes:

```
p4 <- p3 + scale_x_continuous() + scale_y_continuous()
show(p4)
```

We should also supply informative axis labels. The default axis labels are taken from the names of the variables that we used in the aesthetic mappings for `x` and `y`. Use the `xlab()` and `ylab()` functions to provide informative axis labels.

```
p5 <- p4 + xlab(...) + ylab(...)
show(p5)
```

10.2.4 Themes

Our plot contains most of the same graphical elements as the reference plot. But it does not look the same. The original plot was drawn on a white background. But our plot has a grey background with white major and minor grid lines.

Themes control many aspects of the appearance of the plot that do not depend on the data. The default theme, which we have been using so far, is called `theme_gray()`. Use the `help()` function to look up `theme_gray()` and you will see the help page documents the other complete themes provided by the ggplot2 package.

- Modify the chunk below to use different themes until you find one that most closely matches the original plot in the file bentK.pdf.

- Find the parameter that changes to the font size and use it to reduce the font until the y-axis label fits.

```
p6 <- p5 + ...
show(p6)
```

10.2.5 Putting everything together

In this exercise, you have constructed the plot in steps, saving and displaying the plot in separate chunks to allow you to see the changes as we add each layer of the plot. In practice, we do not work like this. All the instructions for creating the plot should be together. Fill in the chunk below so that the plot is recreated in a single chunk.

```
p <- ...
show(p)
```

There is still one thing missing from our reproduction. In the original plot, the treatment and control groups were distinguished by different plotting symbols, whereas in our plot they both have circular marks. We can change this by adding the aesthetic mapping `shape`.

Modify the chunk above so that the aesthetic shape is mapped onto the variable treat. You can do this inside the call to `ggplot()` or in the call to `geom_point()`.

Note that adding the shape aesthetic automatically adds a legend to the plot, telling us which shape corresponds to which group.

10.3 Self study

Congratulations, you have mostly recreated the reference figure from Kristensen et al (1994). However, have you really learned anything? The best way to find out is to come back in a week's time and try to recreate the reference plot without looking at this exercise sheet.

You may find it useful to download the ggplot cheatsheet. You can also use R help to find out more about the functions in the ggplot2 package. Finally, you can refer to the book Ggplot2: elegant graphics for data analysis by Hadley Wickham, creator of the ggplot2 package.

10.4 Adding the table (advanced topic)

There is one important element missing from the graph. This is the table of the number of participants in each group. This is an advanced topic so instead of explaining each step I am just going to give you the solution. Do not be concerned if you do not understand this part.

Firstly we create a new graphical object to represent the table. This object has its own aesthetic mappings, geometries and scales. I will skip a detailed description of this step except to note that some of the arguments below will suppress graphical features (e.g. `NULL` and `element_blank()`)

```
tab <- ggplot(data=alkfos,
              mapping=aes(x=time, y=treat, label=available)) +
              geom_text(size=2) + xlab(NULL) + ylab(NULL) +
              scale_x_continuous(breaks=NULL) +
              theme_bw(base_size=9) +
              theme(panel.grid=element_blank())
tab
```

Next use the `plot_grid()` function from the cowplot package to stack the displayed objects vertically (`align="v"`), in a single column (`ncol=1`, `nrow=2`), with the left and right ends of the x-axes aligned (`axis="lr"`) and most of the space taken up by the plot (`rel_heights=c(5,1)`).

```
library(cowplot)
plot_grid(plotlist=list(p, tab), align="v", axis="lr",
          ncol=1, nrow=2, rel_heights=c(5,1))
```

10.5 References

Kristensen B, Ejlertsen B, Dalgaard P, Larsen L, Holmegaard SN, Transbøl I, Mouridsen HT. Tamoxifen and bone metabolism in postmenopausal low-risk breast cancer patients: a randomized study. *J Clin Oncol.* 1994 May;**12(5)**:992-7. doi: 10.1200/JCO.1994.12.5.992. PMID: 8164053.

Chapter 11

Survival analysis with competing risks: Oral cancer patients

11.1 Description of the data

File `oralca2.txt`, that you may access from a url address to be given in the practical, contains data from 338 patients having an oral squamous cell carcinoma diagnosed and treated in one tertiary level oncological clinic in Finland since 1985, followed-up for mortality until 31 December 2008.

The dataset contains the following variables:

Variable	Description
`sex`	sex, a factor with categories; `1 = "Female", 2 = "Male"`
`age`	age (years) at the date of diagnosing the cancer
`stage`	TNM stage of the tumour (factor): `1 = "I", ..., 4 = "IV", 5 = "unkn"`
`time`	follow-up time (in years) since diagnosis until death or censoring
`event`	event ending the follow-up (numeric): `0 = censoring alive, 1 = death from oral cancer, 2 = death from other causes.`

11.2 Loading the packages and the data

- Load the R packages `Epi`, and `survival` needed in this exercise.

```
library(Epi)
library(survival)
cB8  <- c("#000000", "#E69F00", "#56B4E9", "#009E73",
          "#F0E442", "#0072B2", "#D55E00", "#CC79A7") #colors chosen
options(digits=3)
```

- Read the datafile `oralca2.txt` from a website, whose precise address will be given in the practical, into an R data frame named `orca`. Look at the head, structure and the summary of the data frame. Using function `table()` count the numbers of censorings as well as deaths from oral cancer and other causes, respectively, from the `event` variable.

```
orca <-  read.table(file = 'https://raw.githubusercontent.com/SPE-R/SPE/master/pracs/data/oralca2.txt', he
head(orca)
orca$stage<-as.factor(orca$stage)
orca$sex<-factor(orca$sex,levels=c("Male","Female"))
str(orca)
summary(orca)
```

11.3 Total mortality: Kaplan–Meier analyses

- We start our analysis of total mortality pooling the two causes of death into a single outcome. First, construct a *survival object* `orca$suob` from the event variable and the follow-up time using function `Surv()`. Look at the structure and summary of `suob` .

```
suob <- Surv(orca$time, 1 * (orca$event > 0))
str(suob)
summary(suob)
```

- Create a `survfit` object `s.all`, which does the default calculations for a Kaplan–Meier analysis of the overall (marginal) survival curve.

```
s.all <- survfit(suob ~ 1, data = orca)
```

See the structure of this object and apply `print()` method on it, too. Look at the results; what do you find?

```
s.all
str(s.all)
```

- The `summary` method for a `survfit` object would return a lengthy life table. However, the `plot` method with default arguments offers the Kaplan–Meier curve for a conventional illustration of the survival experience in the whole patient group. Alternatively, instead of graphing survival proportions, one can draw a curve describing their complements: the cumulative mortality proportions. This curve is drawn together with the survival curve as the result of the second command line below.

```
plot(s.all,main="KM estimate of the survival
     and cum. mortality proportions",
     xlab="years", ylab="Survival")
```

```
plot(s.all)
lines(s.all, fun = "event", mark.time = F, conf.int = FALSE)
```

The effect of option `mark.time=F` is to omit marking the times when censorings occurred.

11.4 Total mortality by stage

Tumour stage is an important prognostic factor in cancer survival.

- stage I: cancer hasn't spread to lymph nodes or other tissue (local),
  stage II: cancer has grown but not spread
  stage III:cancer has grown larger but not spread
  stage IV cancer has spread to other organs or areas

- Plot separate cumulative mortality curves for the different stage groups marking them with different colors, the order which you may define yourself. Also find the median survival time for each stage.

```
s.stg <- survfit(suob ~ stage, data = orca)
col5 <- cB8[1:5]
plot(s.stg, col = col5, fun = "event", main="Cum. mortality by stage",mark.time = FALSE)
legend(15, 0.6, title="stage",legend=levels(orca$stage),col = col5,lty=rep(1,5))
```

- Create now two parallel plots of which the first one describes the cumulative hazards and the second one graphs the log-cumulative hazards against log-time for the different stages. Compare the two presentations with each other and with the one in the previous item.

```
par(mfrow = c(1, 2))
plot(s.stg, col = col5, fun = "cumhaz", main = "Cumulative hazards")
legend(1, 3.5, title="stage",legend=levels(orca$stage),col = col5,lty=rep(1,5), cex=0.8)
plot(s.stg, col = col5, fun = "cloglog",main = "cloglog: log cum.haz")
legend(3, -2, title="stage",legend=levels(orca$stage),col = col5,lty=rep(1,5), cex=0.7)
```

- If the survival times were *exponentially* distributed in a given (sub)population the corresponding cloglog-curve should follow an approximately linear pattern. Could this be the case here in the different stages?
- Also, if the survival distributions of the different subpopulations would obey the *proportional hazards* model, the vertical distance between the cloglog-curves should be approximately constant over the time axis. Do these curves indicate serious deviation from the proportional hazards assumption?
- In the lecture handouts it was observed that the crude contrast between males and females in total mortality appears unclear, but the age-adjustment in the Cox model provided a more expected hazard ratio estimate. We shall examine the confounding by age somewhat closer. First categorize the continuous age variable into, say, three categories by function `cut()` using suitable breakpoints, like 55 and 75 years, and cross-tabulate sex and age group:

```
orca$agegr <- cut(orca$age, br = c(0, 55, 75, 95))
stat.table(list(sex, agegr), list(count(), percent(agegr)),
  margins = TRUE,
  data = orca
)
```

Male patients are clearly younger than females in these data.

Now, plot Kaplan–Meier curves jointly classified by sex and age.

```
s.agrx <- survfit(suob ~ agegr + sex, data=orca)
par(mfrow=c(1,1))
plot(s.agrx, fun="event", main="Cumulative mortality (KM) by age and sex",xlab="Time since oral cancer dia
             col=rep(c(cB8[7], cB8[6]),3), lty=c(2,2, 1,1, 5,5),
     xaxs="i",yaxs="i")
legend(12,0.35, legend=c("(0,55] Female "," (0,55] Male",
                         "(55,75] Female "," (55,75] Male",
                         "(75,95] Female "," (75,95] Male" ),
       col=rep(c(cB8[7], cB8[6]),3), lty=c(2,2, 1,1, 5,5),cex=0.65)
```

In each age band the mortality curve for males is on a higher level than that for females.

11.5 Event-specific cumulative mortality curves

We move on to analysing cumulative mortalities for the two causes of death separately, first overall and then by prognostic factors.

- Use the `survfit`-function in `survival` package with option `type="mstate"`.

```
library(survival)
cif1 <- survfit(Surv(time, event, type = "mstate") ~ 1,
  data = orca
)
str(cif1)
```

- One could apply here the plot method of the survfit object to plot the cumulative incidences for each cause. However, we suggest that you use instead a simple function `plotCIF()` found in the `Epi` package. The main arguments are

`data`	data frame created by function }`survfit()`
`event`	indicator for the event: values 1 or 2.

Other arguments are like in the ordinary `plot()` function.

- Draw two parallel plots describing the overall cumulative incidence curves for both causes of death

```
par(mfrow = c(1, 2))
plotCIF(cif1, 1, main = "Cancer death",xlab="Time since oral cancer diagnosis (years)")
plotCIF(cif1, 2, main = "Other deaths",xlab="Time since oral cancer diagnosis (years)")
```

- Compute the estimated cumulative incidences by stage for both causes of death. Now you have to add variable `stage` to survfit-function.

See the structure of the resulting object, in which you should observe strata variable containing the stage grouping variable. Plot the pertinent curves in two parallel graphs. Cut the y-axis for a more efficient graphical presentation

```
col5 <- col5
cif2 <- survfit(Surv(time, event, type = "mstate") ~ stage,
  data = orca
)
str(cif2)

par(mfrow = c(1, 2))
plotCIF(cif2, 1,
  main = "Cancer death by stage",
  col = col5, ylim = c(0, 0.7)
)
plotCIF(cif2, 2,
  main = "Other deaths by stage",
  col = col5, ylim = c(0, 0.7)
)

legend(3, 0.7, title="stage",legend=levels(orca$stage),col = col5,lty=rep(1,5), cex=0.7)
```

Compare the two plots. What would you conclude about the effect of stage on the two causes of death?

- Using another function `stackedCIF()` in `Epi` you can put the two cumulative incidence curves in one graph but stacked upon one another such that the lower curve is for the cancer deaths and the upper curve is for total mortality, and the vertical difference between the two curves describes the cumulative mortality from other causes. You can also add some colours for the different zones:

```
par(mfrow=c(1,1),xaxs="i", yaxs="i") # make plot start 0,0
stackedCIF(cif1,xlim=c(0,20),
           col = c("black"),
           fill=c(cB8[6],cB8[8],cB8[2])) #choosing some colors
text( 10, 0.10, "Oral ca death ", pos = 4)
text( 10, 0.5, " Other death ", pos = 4)
text( 10, 0.80, " Alive ", pos = 4)
```

11.6 Regression modelling of overall mortality.

- Fit the semiparametric proportional hazards regression model, a.k.a. the Cox model, on all deaths including sex, age and stage as covariates. Use function `coxph()` in package `survival`. It is often useful to center and scale continuous covariates like `age` here. The estimated rate ratios and their confidence intervals can also here be displayed by applying `ci.lin()` on the fitted model object.

```
options(show.signif.stars = FALSE)
m1 <- coxph(suob ~ sex + I((age - 65) / 10) + stage, data = orca)
summary(m1)
round(ci.exp(m1),3)
```

Look at the results. What are the main findings?

- Check whether the data are sufficiently consistent with the assumption of proportional hazards with

respect to each of the variables separately as well as globally, using the `cox.zph()` function.

```
cox.zph(m1)
```

- No evidence against proportionality assumption could apparently be found. Moreover, no difference can be observed between stages I and II in the estimates. On the other hand, the group with stage unknown is a complex mixture of patients from various true stages. Therefore, it may be prudent to exclude these subjects from the data and to pool the first two stage groups into one. After that fit a model in the reduced data with the new stage variable.

```
orca2 <- subset(orca, stage != "unkn")
orca2$st3 <- Relevel(orca2$stage, list(1:2, 3, 4:5))
levels(orca2$st3) <- c("I-II", "III", "IV")
m2 <- coxph(Surv(orca2$time, 1 * (orca2$event > 0)) ~ sex + I((age - 65) / 10) + st3, data = orca2)
summary(m2)
#m2 <- update(m1, . ~ . - stage + st3, data = orca2) #do not work
round(ci.exp(m2), 3)
```

- Plot the predicted cumulative mortality curves by stage, jointly stratified by sex and age, focusing only on 40 and 80 year old patients, respectively, based on the fitted model `m2`. You need to create a new artificial data frame containing the desired values for the covariates.

```
newd <- data.frame(
  sex = c(rep("Male", 6), rep("Female", 6)),
  age = rep(c(rep(40, 3), rep(80, 3)), 2),
  st3 = rep(levels(orca2$st3), 4)
)
newd
col3 <- cB8[1:3] #pre-setting color palette
leg<-levels(interaction(levels(factor(orca2$sex)),levels(orca2$st3))) #legend labels by sex and stage
```

```
## Warning in ans * length(l) + if1: longer object length is not a
## multiple of shorter object length
```

```
par(mfrow = c(1, 2))
plot(
  survfit(m2, newdata = subset(newd, sex == "Male" & age == 40)),
  col = col3,
  lty= 1,
  fun = "event", mark.time = FALSE,
  main="Cum. mortality for M and F \n age 40"
  )
lines(
  survfit(m2, newdata = subset(newd, sex == "Female" & age == 40)),
  col = col3, lty=c(2,2,2),fun = "event", mark.time = FALSE
)
legend(0, 0.95, title="stage",legend=leg,
       col = c(col3[1],col3[1],col3[2],col3[2],col3[3],col3[3]),
        ,lty=c(2,1,2,1,2,1), cex=0.6)
plot(
  survfit(m2, newdata = subset(newd, sex == "Male" & age == 80)),
  ylim = c(0, 1), col = col3, fun = "event", mark.time = FALSE,
   main="Cum. mortality for M and F \n age 80"
)
lines(
  survfit(m2, newdata = subset(newd, sex == "Female" & age == 80)  ),
  col = col3, fun = "event", lty = 2, mark.time = FALSE
)
legend(10, 0.5, title="stage",legend=leg,
```

```
       col = c(col3[1],col3[1],col3[2],col3[2],col3[3],col3[3]),
        ,lty=c(2,1,2,1,2,1), cex=0.7)
```

11.7 Modelling event-specific hazards

- Fit the Cox model for the cause-specific hazard of cancer deaths with the same covariates as above. In this case only cancer deaths are counted as events and deaths from other causes are included into censorings.

```
m2haz1 <-
  coxph(
    Surv(time, event == 1) ~ sex + I((age - 65) / 10) + st3,
    data = orca2
  )
round(ci.exp(m2haz1), 4)
cox.zph(m2haz1)
```

Compare the results with those of model `m2`. What are the major differences?

- Fit a similar model for deaths from other causes and compare the results.

```
m2haz2 <-
  coxph(
    Surv(time, event == 2) ~ sex + I((age - 65) / 10) + st3,
    data = orca2
  )
round(ci.exp(m2haz2), 4)
cox.zph(m2haz2)
```

11.8 Lexis object with multi-state set-up

Before entering to multi-state analyses, it might be instructive to apply some Lexis tools to illustrate the competing-risks set-up. More detailed explanation of these tools will be given by Bendix later.

- Form a `Lexis` object from the data frame and print a summary of it. We shall name the main (and only) time axis in this object as `stime`.

```
orca.lex <- Lexis(
  exit = list(stime = time),
  exit.status = factor(event,
    labels = c("Alive", "Oral ca. death", "Other death")
  ),
  data = orca
)
summary(orca.lex)
```

- Draw a box diagram of the two-state set-up of competing transitions. Run first th e following command line

```
boxes(orca.lex,boxpos = TRUE)
```

Now, move the cursor to the point in the graphics window, at which you wish to put the box for *Alive*, and click. Next, move the cursor to the point at which you wish to have the box for *Oral ca. death*, and click. Finally, do the same with the box for *Other death*. If you are not happy with the outcome, run the command line again and repeat the necessary mouse moves and clicks.

Chapter 12

Time-splitting, time-scales and SMR

This exercise is about mortaity among Danish Diabetes patients. It is based on the dataset `DMlate`, a random sample of 10,000 patients from the Danish Diabetes Register (scrambeled dates), all with date of diagnosis after 1994.

Start by loading the relevant packages:

```
library(Epi)
library(popEpi)
library(mgcv)
library(tidyverse)
```

Then load the data and take a look at the data:

```
data(DMlate)
str(DMlate)
```

```
'data.frame':   10000 obs. of  7 variables:
 $ sex  : Factor w/ 2 levels "M","F": 2 1 2 2 1 2 1 1 2 1 ...
 $ dobth: num  1940 1939 1918 1965 1933 ...
 $ dodm : num  1999 2003 2005 2009 2009 ...
 $ dodth: num  NA NA NA NA NA ...
 $ dooad: num  NA 2007 NA NA NA ...
 $ doins: num  NA NA NA NA NA NA NA NA NA NA ...
 $ dox  : num  2010 2010 2010 2010 2010 ...
```

You can get a more detailed explanation of the data by referring to the help page:

```
?DMlate
```

1. Set up the dataset as a `Lexis` object with age, calendar time and duration of diabetes as timescales, and date of death as event. Make sure that you know what each of the arguments to `Lexis` mean:

   ```
   LL <- Lexis(entry = list(A = dodm - dobth,
                            P = dodm,
                          dur = 0),
               exit = list(P = dox),
         exit.status = factor(!is.na(dodth),
                              labels = c("Alive", "Dead")),
               data = DMlate)
   ```

   ```
   NOTE: entry.status has been set to "Alive" for all.
   NOTE: Dropping  4  rows with duration of follow up < tol
   ```

   Take a look at the first few lines of the resulting dataset, for example using `head()`.

2. Get an overview of the mortality by using `stat.table` to tabulate no. deaths, person-years (`lex.dur`) and the crude mortality rate by sex. Try:

```
stat.table(sex,
           list(D = sum(lex.Xst == "Dead"),
                Y = sum(lex.dur),
             rate = ratio(lex.Xst == "Dead",
                          lex.dur,
                          1000)),
          margins = TRUE,
             data = LL)
```

```
 ---------------------------------
 sex           D        Y    rate
 ---------------------------------
 M       1343.00 27614.21   48.63
 F       1156.00 26659.05   43.36

 Total   2499.00 54273.27   46.04
 ---------------------------------
```

```
# stat.table is more versatile than xtabs:
xtabs(cbind(D = lex.Xst == "Dead",
            Y = lex.dur)
      ~ sex,
      data = LL)
```

```
sex     D     Y
  M  1343 27614
  F  1156 26659
```

3. If we want to assess how mortality depends on age, calendar time and duration or how it relates to population mortality, we should in principle split the follow-up along all three time scales. In practice it is sufficient to split it along one of the time-scales and then use the value of each of the time-scales at the left endpoint of the intervals.

   Use `splitLexis` (or `splitMulti` from the `popEpi` package) to split the follow-up along the age-axis in sutiable intervals (here set to 1/2 year, but really immaterial as long as it is small):

```
SL <- splitLexis(LL,
                 breaks = seq(0, 125, 1 / 2),
             time.scale = "A")
summary(SL)
```

```
Transitions:
     To
From     Alive Dead  Records:  Events: Risk time:  Persons:
  Alive 115974 2499    118473     2499      54273      9996
```

   How many records are now in the dataset? How many person-years? Compare to the original `Lexis`-dataset.

12.1 Age-specific mortality

1. Now estimate age-specific mortality curves for men and women separately, using splines as implemented in `gam`. We use `k = 20` to be sure to catch any irregularities by age.

```
r.m <- mgcv::gam(cbind(lex.Xst == "Dead", lex.dur) ~ s(A, k = 20),
                 family = poisreg,
                   data = subset(SL, sex == "M"))
```

Make sure you understand all the components on this modeling statement. Fit the same model for women.

There is a convenient wrapper for this, exploiting the `Lexis` structure of data, but which does not have an update

```
r.m <- gam.Lexis(subset(SL, sex == "M"), ~ s(A, k = 20))
```

```
mgcv::gam Poisson analysis of Lexis object subset(SL, sex == "M") with log link:
Rates for the transition:
Alive->Dead
```

```
r.f <- gam.Lexis(subset(SL, sex == "F"), ~ s(A, k = 20))
```

```
mgcv::gam Poisson analysis of Lexis object subset(SL, sex == "F") with log link:
Rates for the transition:
Alive->Dead
```

2. Now, extract the estimated rates by using the wrapper function `ci.pred` that computes predicted rates and confidence limits for these.

   `glm.Lexis` and `gam.Lexis` use the `poisreg` family that will return the rates in the (inverse) units in which the person-years were given; that is the units in which `lex.dur` is recorded.

```
nd <- data.frame(A = seq(20, 90, 0.5))
p.m <- ci.pred(r.m, newdata = nd)
p.f <- ci.pred(r.f, newdata = nd)
str(p.m)
```

```
 num [1:141, 1:3] 0.00132 0.00137 0.00142 0.00147 0.00152 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:141] "1" "2" "3" "4" ...
  ..$ : chr [1:3] "Estimate" "2.5%" "97.5%"
```

3. Plot the predicted rates for men and women together - using for example `matplot` or `matshade`.

```
    par(mar = c(3.5,3.5,1,1),
        mgp = c(3,1,0) / 1.6,
        las = 1,
        bty = "n",
       lend = "butt")
matplot(nd$A, cbind(p.m, p.f) * 1000,
        type = "l",
         col = rep(c("blue", "red"), each = 3),
         lwd = c(3, 1, 1),
         lty = 1,
         log = "y", yaxt = "n",
        xlab = "Age",
        ylab = "Mortality per 1000 PY")
axis(side = 2,
     at = ll <- outer( c(5, 10, 20), -1:1, function(x,y) x * 10^y),
     labels = ll)
```

12.2 Further time scales: period and duration

4. We now want to model the mortality rates among diabetes patients also including current date and duration of diabetes, using penalized splines. Use the argument `bs = "cr"` to `s()` to get cubic splines instead of thin plate (`"tp"`) splines which is the default.

   As before specify the model exploiting the `Lexis` class of the dataset, try:

```
Mcr <- gam.Lexis(subset(SL, sex == "M"),
                 ~ s(A, bs = "cr", k = 10) +
                   s(P, bs = "cr", k = 10) +
                 s(dur, bs = "cr", k = 10))
```

```
mgcv::gam Poisson analysis of Lexis object subset(SL, sex == "M") with log link:
Rates for the transition:
Alive->Dead
```

```
summary(Mcr)
```

```
Family: poisson
Link function: log

Formula:
cbind(trt(Lx$lex.Cst, Lx$lex.Xst) %in% trnam, Lx$lex.dur) ~ s(A,
    bs = "cr", k = 10) + s(P, bs = "cr", k = 10) + s(dur, bs = "cr",
    k = 10)

Parametric coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -3.5407     0.0494   -71.7   <2e-16

Approximate significance of smooth terms:
        edf Ref.df Chi.sq p-value
s(A)   3.65   4.52 1013.2 < 2e-16
s(P)   1.02   1.05   17.6 3.5e-05
s(dur) 7.59   8.38   74.5 < 2e-16

R-sq.(adj) =  0.00333   Deviance explained = 9.87%
UBRE = -0.8054  Scale est. = 1         n = 60347
```

Fit the same model for women as well. Are the models reasonably fitting?

```
Fcr <- gam.Lexis(subset(SL, sex == "F"),
                 ~ s(A, bs = "cr", k = 10) +
                   s(P, bs = "cr", k = 10) +
                 s(dur, bs = "cr", k = 10))
```

```
mgcv::gam Poisson analysis of Lexis object subset(SL, sex == "F") with log link:
Rates for the transition:
Alive->Dead
```

```
summary(Fcr)
```

```
Family: poisson
Link function: log

Formula:
cbind(trt(Lx$lex.Cst, Lx$lex.Xst) %in% trnam, Lx$lex.dur) ~ s(A,
    bs = "cr", k = 10) + s(P, bs = "cr", k = 10) + s(dur, bs = "cr",
    k = 10)

Parametric coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -3.7848     0.0581   -65.2   <2e-16

Approximate significance of smooth terms:
```

```
        edf Ref.df Chi.sq p-value
s(A)   2.67   3.37  988.5 < 2e-16
s(P)   1.90   2.39   20.1 0.00014
s(dur) 5.97   6.97   39.0 < 2e-16

R-sq.(adj) =  0.00417   Deviance explained = 11.1%
UBRE = -0.82405  Scale est. = 1         n = 58126
```

5. Plot the estimated effects, using the default plot method for `gam` objects. Remember that there are three effects estimated, so it is useful set up a multi-panel display, and for the sake of comparability to set ylim to the same for men and women:

```
par(mfrow = c(2, 3))
plot(Mcr, ylim = c(-3, 3))
plot(Fcr, ylim = c(-3, 3))
```

```
par(mfcol = c(3, 2))
plot(Mcr, ylim = c(-3, 3))
plot(Fcr, ylim = c(-3, 3))
```

   What is the absolute scale for these effects?

6. Compare the fit of the naive model with just age and the three-factor models, using `anova`, e.g.:

```
anova(Mcr, r.m, test = "Chisq")
```

```
Analysis of Deviance Table

Model 1: cbind(trt(Lx$lex.Cst, Lx$lex.Xst) %in% trnam, Lx$lex.dur) ~ s(A,
    bs = "cr", k = 10) + s(P, bs = "cr", k = 10) + s(dur, bs = "cr",
    k = 10)
Model 2: cbind(trt(Lx$lex.Cst, Lx$lex.Xst) %in% trnam, Lx$lex.dur) ~ s(A,
    k = 20)
  Resid. Df Resid. Dev    Df Deviance Pr(>Chi)
1     60332      11717
2     60340      11812 -7.95    -95.1   <2e-16
```

   What do you conclude?

7. The model we fitted has three time-scales: current age, current date and current duration of diabetes, so the effects that we report are not immediately interpretable, as they are (as in any kind of multiple regressions) to be interpreted as *all else equal* which they are not, as the three time scales advance simultaneously at the same pace.

   The reporting would therefore more naturally be on the mortality scale as a function of age, but showing the mortality for persons diagnosed in different ages, using separate displays for separate years of diagnosis. This is most easily done using the `ci.pred` function with the `newdata =` argument. So a person diagnosed in age 50 in 1995 will have a mortality measured in cases per 1000 PY as:

```
pts <- seq(0, 12, 1/4)
nd <- data.frame(A = 50   + pts,
                 P = 1995 + pts,
               dur =        pts)
m.pr <- ci.pred(Mcr, newdata = nd)
```

   Note that because we used `gam.Lexis` which uses the `poisreg` family we need not specify `lex.dur` as a variable in the prediction data frame `nd`. Predictions will be rates in the same units as `lex.dur` (well, the inverse).

   Now take a look at the result from the `ci.pred` statement and construct prediction of mortality for men and women diagnosed in a range of ages, say 50, 60, 70, and plot these together in the same graph:

```
cbind(nd, ci.pred(Mcr, newdata = nd))[1:10,]
```

8. From figure it seems that the duration effect is over-modeled, so refit constraining the d.f. to 5:

```
Mcr <- gam.Lexis(subset(SL, sex == "M"),
                 ~ s(A, bs = "cr", k = 10) +
                   s(P, bs = "cr", k = 10) +
                 s(dur, bs = "cr", k = 5))
```

```
mgcv::gam Poisson analysis of Lexis object subset(SL, sex == "M") with log link:
Rates for the transition:
Alive->Dead
```

```
Fcr <- gam.Lexis(subset(SL, sex == "F"),
                 ~ s(A, bs = "cr", k = 10) +
                   s(P, bs = "cr", k = 10) +
                 s(dur, bs = "cr", k = 5))
```

```
mgcv::gam Poisson analysis of Lexis object subset(SL, sex == "F") with log link:
Rates for the transition:
Alive->Dead
```

Plot the estimated rates from the revised models. What do you conclude from the plots?

12.3 SMR

The SMR is the **S**tandardized **M**ortality **R**atio, which is the mortality rate-ratio between the diabetes patients and the general population. In real studies we would subtract the deaths and the person-years among the diabetes patients from those of the general population, but since we do not have access to these, we make the comparison to the general population at large, *i.e.* also including the diabetes patients.

So we now want to include the population mortality rates as a fixed variable in the split dataset; for each record in the split dataset we attach the value of the population mortality for the relevant sex, and and calendar time.

This can be achieved in two ways: Either we just use the current split of follow-up time and allocate the population mortality rates for some suitably chosen (mid-)point of the follow-up in each, or we make a second split by date, so that follow-up in the diabetes patients is in the same classification of age and data as the population mortality table.

12. We will use the former approach, using the dataset split in 6 month intervals, and then include as an extra variable the population mortality as available from the data set `M.dk`.

    First create the variables in the diabetes dataset that we need for matching with the population mortality data, that is sex and age and date at the midpoint of each of the intervals (or rather at a point 3 months after the left endpoint of the interval; recall we split the follow-up in 6 month intervals).

    We need to have variables of the same type when we merge, so we must transform the sex variable in `M.dk` to a factor, and must for each follow-up interval in the `SL` data have an age and a period variable that can be used in merging with the population data.

```
str(SL)
```

```
Classes 'Lexis' and 'data.frame':   118473 obs. of  14 variables:
 $ lex.id : int  1 1 1 1 1 1 1 1 1 1 ...
 $ A      : num  58.7 59 59.5 60 60.5 ...
 $ P      : num  1999 1999 2000 2000 2001 ...
 $ dur    : num  0 0.339 0.839 1.339 1.839 ...
 $ lex.dur: num  0.339 0.5 0.5 0.5 0.5 ...
 $ lex.Cst: Factor w/ 2 levels "Alive","Dead": 1 1 1 1 1 1 1 1 1 1 ...
 $ lex.Xst: Factor w/ 2 levels "Alive","Dead": 1 1 1 1 1 1 1 1 1 1 ...
 $ sex    : Factor w/ 2 levels "M","F": 2 2 2 2 2 2 2 2 2 2 ...
```

```
 $ dobth  : num  1940 1940 1940 1940 1940 ...
 $ dodm   : num  1999 1999 1999 1999 1999 ...
 $ dodth  : num  NA NA NA NA NA NA NA NA NA NA ...
 $ dooad  : num  NA NA NA NA NA NA NA NA NA NA ...
 $ doins  : num  NA NA NA NA NA NA NA NA NA NA ...
 $ dox    : num  2010 2010 2010 2010 2010 ...
 - attr(*, "breaks")=List of 3
  ..$ A  : num [1:251] 0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 ...
  ..$ P  : NULL
  ..$ dur: NULL
 - attr(*, "time.scales")= chr [1:3] "A" "P" "dur"
 - attr(*, "time.since")= chr [1:3] "" "" ""
```

```
SL$Am <- floor(SL$A + 0.25)
SL$Pm <- floor(SL$P + 0.25)
data(M.dk)
str(M.dk)
```

```
'data.frame':   7800 obs. of  6 variables:
 $ A   : num  0 0 0 0 0 0 0 0 0 0 ...
 $ sex : num  1 2 1 2 1 2 1 2 1 2 ...
 $ P   : num  1974 1974 1975 1975 1976 ...
 $ D   : num  459 303 435 311 405 258 332 205 312 233 ...
 $ Y   : num  35963 34383 36099 34652 34965 ...
 $ rate: num  12.76 8.81 12.05 8.97 11.58 ...
 - attr(*, "Contents")= chr "Number of deaths and risk time in Denmark"
```

```
M.dk <- transform(M.dk,
                  Am = A,
                  Pm = P,
                 sex = factor(sex, labels = c("M", "F")))
str(M.dk)
```

```
'data.frame':   7800 obs. of  8 variables:
 $ A   : num  0 0 0 0 0 0 0 0 0 0 ...
 $ sex : Factor w/ 2 levels "M","F": 1 2 1 2 1 2 1 2 1 2 ...
 $ P   : num  1974 1974 1975 1975 1976 ...
 $ D   : num  459 303 435 311 405 258 332 205 312 233 ...
 $ Y   : num  35963 34383 36099 34652 34965 ...
 $ rate: num  12.76 8.81 12.05 8.97 11.58 ...
 $ Am  : num  0 0 0 0 0 0 0 0 0 0 ...
 $ Pm  : num  1974 1974 1975 1975 1976 ...
```

Then match the rates from `M.dk` into `SL`; `sex`, `Am` and `Pm` are the common variables, and therefore the match is on these variables:

```
SLr <- merge(SL,
             M.dk[, c("sex", "Am", "Pm", "rate")])
dim(SL)
```

```
[1] 118473     16
```

```
dim(SLr)
```

```
[1] 118454     17
```

This merge (remember to `?merge`!) only takes rows that have information from both datasets, hence the slightly fewer rows in `SLr` than in `SL`.

- Compute the expected number of deaths as the person-time multiplied (`lex.dur`) by the correspond-

ing population rate, and put it in a new variable, `E`, say (Expected). Use `stat.table` to make a table of observed, expected and the ratio (SMR) by age (suitably grouped, look for `cut`) and sex.

13. Fit a poisson model with sex as the explanatory variable and log-expected as offset to derive the SMR (and c.i.). Some of the population mortality rates are 0, so you need to exclude those records from the analysis.

```
msmr <- glm((lex.Xst == "Dead") ~ sex - 1,
            offset = log(E),
            family = poisson,
              data = subset(SLr, E > 0))
ci.exp(msmr)
```

```
     exp(Est.) 2.5% 97.5%
sexM      1.69 1.60  1.78
sexF      1.54 1.46  1.63
```

Do you recognize the numbers?

- The same model can be fitted a bit simpler by the `poisreg` family, try:

```
msmr <- glm(cbind(lex.Xst == "Dead", E) ~ sex - 1,
            family = poisreg,
              data = subset(SLr, E > 0))
ci.exp(msmr)
```

```
     exp(Est.) 2.5% 97.5%
sexM      1.69 1.60  1.78
sexF      1.54 1.46  1.63
```

We can assess the ratios of SMRs between men and women by using the `ctr.mat` argument which should be a matrix:

```
(CM <- rbind(M = c(1, 0),
             W = c(0, 1),
         "M/F" = c(1, -1)))
```

```
    [,1] [,2]
M      1    0
W      0    1
M/F    1   -1
```

```
round(ci.exp(msmr, ctr.mat = CM), 2)
```

```
    exp(Est.) 2.5% 97.5%
M        1.69 1.60  1.78
W        1.54 1.46  1.63
M/F      1.09 1.01  1.18
```

What do you conclude about the mortality rates among men and women?

12.4 SMR modeling

14. Now model the SMR using age and date of diagnosis and diabetes duration as explanatory variables, including the expected-number instead of the person-years, using separate models for men and women.

Note that you cannot use `gam.Lexis` from the code you used for fitting models for the rates, you need to use `gam` with the `poisreg` family. And remember to exclude those units where no deaths in the population occur (that is where the population mortality rate is 0).

```
Msmr <- gam(cbind(lex.Xst == "Dead", E)
            ~ s(  A, bs = "cr", k = 5) +
```

```
              s(  P, bs = "cr", k = 5) +
              s(dur, bs = "cr", k = 5),
            family = poisreg,
              data = subset(SLr, E > 0 & sex == "M"))
ci.exp(Msmr)
```

```
            exp(Est.)  2.5% 97.5%
(Intercept)     2.167 2.000 2.347
s(A).1          0.596 0.518 0.686
s(A).2          0.893 0.865 0.921
s(A).3          0.366 0.282 0.476
s(A).4          0.459 0.376 0.559
s(P).1          0.989 0.973 1.006
s(P).2          0.961 0.909 1.016
s(P).3          0.903 0.783 1.041
s(P).4          0.917 0.813 1.035
s(dur).1        0.658 0.577 0.751
s(dur).2        0.845 0.761 0.937
s(dur).3        0.783 0.692 0.886
s(dur).4        1.818 1.104 2.992
```

```
Fsmr <- update(Msmr, data = subset(SLr, E > 0 & sex == "F"))
```

Plot the estimated smooth effects for both men and women using e.g. `plot.gam`. What do you see?

15. Plot the predicted SMRs from the models for men and women diagnosed in ages 50, 60 and 70 as you did for the rates. What do you see?

```
par(mfrow = c(1,1))
n50 <- nd
n60 <- mutate(n50, A = A + 10)
n70 <- mutate(n50, A = A + 20)
head(n70)
```

```
     A    P  dur
1 70.0 1995 0.00
2 70.2 1995 0.25
3 70.5 1996 0.50
4 70.8 1996 0.75
5 71.0 1996 1.00
6 71.2 1996 1.25
```

```
matshade(n50$A, cbind(ci.pred(Msmr, n50),
                      ci.pred(Fsmr, n50)), plot = TRUE,
         col = c("blue", "red"), lwd = 3,
         ylim = c(0.5, 5), log  = "y", xlim = c(50, 80))
matshade(n60$A, cbind(ci.pred(Msmr, n60),
                      ci.pred(Fsmr, n60)),
         col = c("blue", "red"), lwd = 3)
matshade(n70$A, cbind(ci.pred(Msmr, n70),
                      ci.pred(Fsmr, n70)),
         col = c("blue", "red"), lwd = 3)
abline(h = 1)
abline(v = 50 + 0:5, lty = 3, col = "gray")
```

Describe the shapes of the curves. What aspects of the shapes are induced by the model ?

16. Try to simplify the model to one with a simple sex effect, separate linear effects of age and date of follow-up for each sex, and a smooth effect of duration common for both sexes, giving an estimate of the change in

SMR by age and calendar time.

```
Bsmr <- gam(cbind(lex.Xst == "Dead", E)
            ~ sex / A +
              sex / P +
              s(dur, bs = "cr", k = 5),
            family = poisreg,
              data = subset(SLr, E > 0))
round(ci.exp(Bsmr)[-1,], 3)
```

```
         exp(Est.)  2.5%    97.5%
sexF      5.21e+04 0.000 2.38e+23
sexM:A    9.81e-01 0.977 9.86e-01
sexF:A    9.80e-01 0.975 9.86e-01
sexM:P    9.87e-01 0.972 1.00e+00
sexF:P    9.81e-01 0.965 9.98e-01
s(dur).1  6.63e-01 0.601 7.32e-01
s(dur).2  8.61e-01 0.795 9.31e-01
s(dur).3  8.85e-01 0.805 9.74e-01
s(dur).4  1.41e+00 0.945 2.11e+00
```

How much does SMR change by each year of age? And by each calendar year?

What is the meaning of the `sexF` parameter?

17. Use your previous code to plot the predicted mortality from this model too. Are the predicted SMR curves credible?

```
m50 <- mutate(n50, sex = "M")
f50 <- mutate(n50, sex = "F")
m60 <- mutate(m50, A = A + 10)
f60 <- mutate(f50, A = A + 10)
m70 <- mutate(m50, A = A + 20)
f70 <- mutate(f50, A = A + 20)
matshade(n50$A, cbind(ci.pred(Bsmr, m50),
                      ci.pred(Bsmr, f50)), plot = TRUE,
         col = c("blue", "red"), lwd = 3,
         ylim = c(0.5, 5), log  = "y", xlim = c(50, 80))
matshade(n60$A, cbind(ci.pred(Bsmr, m60),
                      ci.pred(Bsmr, f60)),
         col = c("blue", "red"), lwd = 3)
matshade(n70$A, cbind(ci.pred(Bsmr, m70),
                      ci.pred(Bsmr, f70)),
         col = c("blue", "red"), lwd = 3)
abline(h = 1)
abline(h = 1:5, lty = 3, col = "gray")
```

What is your conclusion about SMR for diabetes patients relative to the general popuation?

Chapter 13

Nested case-control study and case-cohort study: Risk factors of coronary heart disease

In this exercise we shall apply both the nested case-control (NCC) design and the case-cohort (CC) design in sampling control subjects from a defined cohort or closed study population. The case group comprises those cohort members who die from coronary heart disease (CHD) during a > 20 years follow-up of the cohort.
The risk factors of interest are cigarette smoking, systolic blood pressure, and total cholesterol level.

Our study population is an occupational cohort comprising 1501 men working in blue-collar jobs in one Nordic country. Eligible subjects had no history of coronary heart disease when recruited to the study in the early 1990s. Smoking habits and many other items were inquired at baseline by a questionnaire, and blood pressure was measured by a research nurse, the values being written down on the questionnaire. Serum samples were also taken from the cohort members at the same time and were stored in a freezer. For some reason, the data in the questionnaires were not entered to any computer file, but the questionnaires were kept in a safe storehouse for further purposes. Also, no biochemical analyses were initially performed for the sera collected from the participants. However, dates of birth and dates of entry to the study were recorded in an electronic file.

In 2010 the study was suddenly reactivated by those investigators of the original team who were still alive then. As the first step mortality follow-up of the cohort members was executed by record linkage to the national population register, from which the dates of death and emigration were obtained. Another linkage was performed with the national register of causes of death in order to get the deaths from coronary heard disease identified. As a result a data file `occoh.txt` was completed containing the following variables:

Variable	Description
`id`	identification number
`birth`	date of birth
`entry`	date of recruitment and baseline measurements
`exit`	date of exit from mortality follow-up
`death`	indicator for vital status at the end of follow-up; 1, if dead from any cause, and = 0, if alive
`chdeath`	indicator for death from coronary heart disease; 1, if *yes*, and 0, if *no*.

This exercise is divided into five main parts:

1. Description of the study base or the follow-up experience of the whole cohort, identification of the cases and illustrating the risk sets.

2. Nested case-control study within the cohort:

   (i) selection of controls by risk set or time-matched sampling using function `ccwc()` in package `Epi`,

(ii) collection of exposure data for cases and controls from the pertinent data base of the whole cohort to the case-control data set using function `merge()`, and
(iii) analysis of the case-control data set with stratified Cox model using function `clogit()` in package `survival()`,

3. Case-cohort study within the cohort:

(i) selection of a subcohort by simple random sampling from the cohort,
(ii) collection of exposure data for subcohort members and cases, and
(iii) analysis of the case-cohort data set with Cox model by weighted partial likelihood including appropriate weighting and correction of estimated covariance matrix for the model coefficients using function `cch()` in package `survival()`.

4. Comparison of results from all previous analyses, also with those from a full cohort design.

5. Further tasks and homework.

13.1 Reading the cohort data, illustrating the study base and risk sets

- Load the packages `Epi` and `survival`. Read in the cohort data file and name the resulting data frame as `oc`. See its structure and print the univariate summaries.

```
library(Epi)
library(survival)
url <- "https://raw.githubusercontent.com/SPE-R/SPE/master/pracs/data"
oc <- read.table(paste(url, "occoh.txt", sep = "/"), header = TRUE)
str(oc)
summary(oc)
```

- It is convenient to change all the dates into fractional calendar years

```
oc$ybirth <- cal.yr(oc$birth)
oc$yentry <- cal.yr(oc$entry)
oc$yexit <- cal.yr(oc$exit)
```

We shall also compute the age at entry and at exit, respectively, as age will be the main time scale in our analyses.

```
oc$agentry <- oc$yentry - oc$ybirth
oc$agexit <- oc$yexit - oc$ybirth
```

- As the next step we shall create a `lexis` object from the data frame along the calendar period and age axes, and as the outcome event we specify the coronary death.

```
oc.lex <- Lexis(
  entry = list(
    per = yentry,
    age = yentry - ybirth
  ),
  exit = list(per = yexit),
  exit.status = chdeath,
  id = id, data = oc
)
str(oc.lex)
summary(oc.lex)
```

- At this stage it is informative to examine a graphical presentation of the follow-up lines and outcome cases in a conventional Lexis diagram. Make use of the `plot` method for `Lexis` objects. Gray lifelines are drawn and a bullet is put at the exit point of those lifelines that end with the outcome event.

```
par(mfrow = c(1, 1))
plot(oc.lex, xlim = c(1990, 2010), grid = TRUE)
points(oc.lex, pch = c(NA, 16)[oc.lex$lex.Xst + 1])
```

- As age is here the main time axis, we shall graphically illustrate the **study base**, *i.e.* the follow-up lines and outcome events, only along the age scale, being ordered by age at exit. Vertical lines at those ages when new coronary deaths occur are drawn to identify the pertinent **risk sets**. For that purpose it is useful first to sort the data frame and the `Lexis` object jointly by age at exit & age at entry, and to give a new ID number according to that order.

```
oc.ord <- cbind(ID = 1:1501, oc[order(oc$agexit, oc$agentry), ])
oc.lexord <- Lexis(
  entry = list(age = agentry),
  exit = list(age = agexit),
  exit.status = chdeath,
  id = ID, data = oc.ord
)
plot(oc.lexord, "age")
points(oc.lexord, pch = ifelse(oc.lexord$lex.Xst == 1, 16, NA))
with(
  subset(oc.lexord, lex.Xst == 1),
  abline(v = agexit, lty = 3)
)
```

- For a closer look, we now zoom the graphical illustration of the risk sets into event times occurring between 50 to 58 years. – Copy the last four lines from the previous item and add arguments `xlim` and `ylim` to the call of `plot()`.

```
plot(oc.lexord, "age", xlim = c(50, 58), ylim = c(5, 65))
points(
  oc.lexord, "age", pch = ifelse(oc.lexord$lex.Xst == 1, 16, NA)
)
with(
  subset(oc.lexord, lex.Xst == 1),
  abline(v = agexit, lty = 3)
)
```

13.2 Nested case-control study

We shall now employ the strategy of **risk-set sampling** or **time-matched** sampling of controls, *i.e.* we are conducting a *nested case-control study* within the cohort.

- The risk sets are defined according to the age at diagnosis of the case. Further matching is applied for age at entry by 1-year agebands. For this purpose we first generate a categorical variable `agen2` for age at entry

```
oc.lex$agen2 <- cut(oc.lex$agentry, br = seq(40, 62, 1))
```

Matched sampling from risk sets may be carried out using function `ccwc()` found in the `Epi` package. Its main arguments are the times of `entry` and `exit` which specify the time at risk along the main time scale (here age), and the outcome variable to be given in the `fail` argument. The number of controls per case is set to be two, and the additional matching factor is given.

- After setting the RNG seed (with your own number), make a call of this function and see the structure of the resulting data frame `cactrl` containing the cases and the chosen individual controls.

```
set.seed(98623)
cactrl <-
  ccwc(
```

```
    entry = agentry, exit = agexit, fail = chdeath,
    controls = 2, match = agen2,
    include = list(id, agentry),
    data = oc.lex, silent = FALSE
  )
str(cactrl)
```

Check the meaning of the four first columns of the case-control data frame from the help page of function `ccwc()`.

- Now we shall start collecting data on the risk factors for the cases and their matched controls, including determination of the total cholesterol levels from the frozen sera! The storehouse of the risk factor measurements for the whole cohort is file `occoh-Xdata.txt`. It contains values of the following variables.

Variable	Description
`id`	identification number, the same as in `occoh.txt`
`smok`	cigarette smoking with categories; 1: `never`, 2: `former`, 3: `1-14/d`, 4: `15+/d`
`sbp`	systolic blood pressure (mmHg)
`tchol`	total cholesterol level (mmol/l)

```
ocX <-
  read.table(
    paste(url, "occoh-Xdata.txt", sep = "/"), header = TRUE
  )
str(ocX)
```

- In the next step we collect the values of the risk factors for our cases and controls by merging the case-control data frame and the storehouse file. In this operation we utilize function `merge()` to select columns of two data frames: `cactrl` (all columns) and `ocX` (four columns) and to merge these into a single file (see exercise 1.1, subsection 1.1.8, where `merge()` was introduced). The `id` variable in both files is used as the key to link each individual case or control with his own data on risk factors.

```
oc.ncc <- merge(cactrl, ocX[, c("id", "smok", "tchol", "sbp")],
  by = "id"
)
str(oc.ncc)
```

- We shall treat smoking as categorical and total cholesterol and systolic blood pressure as quantitative risk factors, but the values of the latter will be divided by 10 to get more interpretable effect estimates.

Variable	Description
`cholgrp`	cholesterol class; 1: `<5`, 2: `5-<6.5`, 3: `>=6.5`
`sbpgrp`	blood pressure class; 1: `<130`, 2: `130-<150`, 3: `150-<170`, 4: `>=170`

Convert the smoking variable into a factor.

```
oc.ncc$smok <- factor(oc.ncc$smok,
  labels = c("never", "ex", "1-14/d", ">14/d")
)
```

- It is useful to start the analysis of case-control data by simple tabulations by the categorized risk factors. Crude estimates of the rate ratios associated with them, in which matching is ignored, can be obtained as follows. We shall focus on smoking

```
stat.table(
  index = list(smok, Fail),
  contents = list(count(), percent(smok)),
```

```
  margins = TRUE,
  data = oc.ncc
)
smok.crncc <- glm(Fail ~ smok, family = binomial, data = oc.ncc)
round(ci.exp(smok.crncc), 3)
```

- A proper analysis takes into account matching that was employed in the selection of controls for each case from the pertinent risk set, further restricted to subjects who were about the same age at entry as the case was. Also, adjustment for the other risk factors is desirable. In this analysis function `clogit()` in `survival` package is utilized. It is in fact a wrapper of function `coxph()`.

```
m.clogit <- clogit(Fail ~ smok + I(sbp / 10) + tchol +
  strata(Set), data = oc.ncc)
summary(m.clogit)
round(ci.exp(m.clogit), 3)
```

Compare these with the crude estimates obtained above.

13.3 Case-cohort study

Now we start applying the second major outcome-selective sampling strategy for collecting exposure data from a big study population

- The subcohort is selected as a simple random sample ($n = 260$) from the whole cohort. The `id`-numbers of the individuals that are selected will be stored in vector `subcids`, and `subcind` is an indicator for inclusion to the subcohort.

```
N <- 1501
n <- 260
set.seed(15792)
subcids <- sample(N, n)
oc.lexord$subcind <- 1 * (oc.lexord$id %in% subcids)
```

- We form the data frame `oc.cc` to be used in the subsequent analysis selecting the union of the subcohort members and the case group from the data frame of the full cohort. After that we collect the data of the risk factors from the data storehouse for the subjects in the case-cohort data

```
oc.cc <- subset(oc.lexord, subcind == 1 | chdeath == 1)
oc.cc <- merge(oc.cc, ocX[, c("id", "smok", "tchol", "sbp")],
  by = "id"
)
str(oc.cc)
```

- We shall now create a graphical illustration of the lifelines contained in the case-cohort data. Lines for the subcohort non-cases are grey without bullet at exit, those for subcohort cases are blue with blue bullet at exit, and for cases outside the subcohort the lines are red and dotted with red bullets at exit.

```
plot(subset(oc.cc, chdeath == 0), "age")
lines(subset(oc.cc, chdeath == 1 & subcind == 1), col = "blue")
lines(subset(oc.cc, chdeath == 1 & subcind == 0), col = "red")
points(subset(oc.cc, chdeath == 1),
  pch = 16,
  col = c("blue", "red")[oc.cc$subcind + 1]
)
```

- Define the categorical smoking variable again.

```
oc.cc$smok <- factor(oc.cc$smok,
  labels = c("never", "ex", "1-14/d", ">14/d")
)
```

A crude estimate of the hazard ratio for the various smoking categories k vs. non-smokers ($k = 1$) can be obtained by tabulating cases (D_k) and person-years (y_k) in the subcohort by smoking and then computing the relevant exposure odds ratio for each category:

$$\mathrm{HR}_k^{\mathrm{crude}} = \frac{D_k/D_1}{y_k/y_1}$$

```
sm.cc <- stat.table(
  index = smok,
  contents = list(Cases = sum(lex.Xst), Pyrs = sum(lex.dur)),
  margins = TRUE,
  data = oc.cc
)
print(sm.cc, digits = c(sum = 0, ratio = 1))
HRcc <-
  (sm.cc[1, -5] / sm.cc[1, 1]) / (sm.cc[2, -5] / sm.cc[2, 1])
round(HRcc, 3)
```

Do these estimates resemble those obtained from nested case-control data?

- To estimate the rate ratios associated with smoking and adjusted for the other risk factors we now fit the pertinent Cox model applying the method of *weighted partial likelihood* as presented by Ling & Ying (1993) and Barlow (1994). This analysis can be done using function `cch()` in package `survival` with `method = "LinYing"`

```
oc.cc$survobj <- with(oc.cc, Surv(agentry, agexit, chdeath))
cch.LY <- cch(survobj ~ smok + I(sbp / 10) + tchol,
  stratum = NULL,
  subcoh = ~subcind, id = ~id, cohort.size = N, data = oc.cc,
  method = "LinYing"
)
summary(cch.LY)
```

13.4 Full cohort analysis and comparisons

Finally, suppose the investigators after all could afford to collect the data on risk factors from the storehouse for the whole cohort.

- Let us form the data frame corresponding to the full cohort design and convert again smoking to be categorical.

```
oc.full <- merge(oc.lex, ocX[, c("id", "smok", "tchol", "sbp")],
  by.x = "id", by.y = "id"
)
oc.full$smok <- factor(oc.full$smok,
  labels = c("never", "ex", "1-14/d", ">14/d")
)
```

Juts for comparison with the corresponding analysis in case-cohort data perform a similar crude estimation of hazard ratios associated with smoking.

```
sm.coh <- stat.table(
  index = smok,
  contents = list(Cases = sum(lex.Xst), Pyrs = sum(lex.dur)),
  margins = TRUE,
  data = oc.full
)
print(sm.coh, digits = c(sum = 0, ratio = 1))
HRcoh <-
```

```
  (sm.coh[1, -5] / sm.coh[1, 1]) / (sm.coh[2, -5] / sm.coh[2, 1])
round(HRcoh, 3)
```

- Fit now the ordinary Cox model to the full cohort. There is no need to employ extra tricks upon the ordinary `coxph()` fit.

```
cox.coh <- coxph(Surv(agentry, agexit, chdeath) ~
  smok + I(sbp / 10) + tchol, data = oc.full)
summary(cox.coh)
```

- Lastly, a comparison of the point estimates and standard errors between the different designs, including variants of analysis for the case-cohort design, can be performed.

```
betas <- cbind(coef(cox.coh), coef(m.clogit), coef(cch.LY))
colnames(betas) <- c("coh", "ncc", "cch.LY")
round(betas, 3)

SEs <- cbind(
  sqrt(diag(cox.coh$var)),
  sqrt(diag(m.clogit$var)),
  sqrt(diag(cch.LY$var))
)
colnames(SEs) <- colnames(betas)
round(SEs, 3)
```

You will notice that the point estimates of the coefficients obtained from the full cohort, nested case-control, and case-cohort analyses, respectively, are somewhat variable. However,
the standard errors from the NCC and CC analyses should be quite similar when the numbers of cases and non-cases are similar.

13.5 Further exercises and homework

- If you have time, you could run both the NCC study and CC study again but now with a larger control group or subcohort; for example 4 controls per case in NCC and $n = 520$ as the subcohort size in CC. Remember resetting the seed first. Pay attention in the results to how much closer will be the point estimates and the proper SEs to those obtained from the full cohort design.
- Instead of simple linear terms for `sbp` and `tchol` you could try to fit spline models to describe their effects.
- A popular alternative to weighted partial likelihood in the analysis of case-cohort data is the *pseudo-likelihood method* (Prentice 1986), which is based on *late entry* to follow-up of the case subjects not belonging to the subcohort. The way to do this is provided by function `cch()` which you can apply directly to the case-cohort data `oc.cc` as before but now with `method = "Prentice"`. – Try this and compare the results with those obtained by weighted partial likelihood in model `cch.LY`.
- Yet another computational solution for maximizing weighted partial likelihood is provided by a combination of functions `twophase()` and `svycoxph()` of the `survey` package. The approach is illustrated with an example in a vignette *Two-phase designs in epidemiology* by Thomas Lumley (see http://cran.r-project.org/web/packages/survey/vignettes/epi.pdf). – You can try this at home and check that you would obtain similar results as with model `cch.LY`.

Chapter 14

Causal inference 2: Model-based estimation of causal estimands

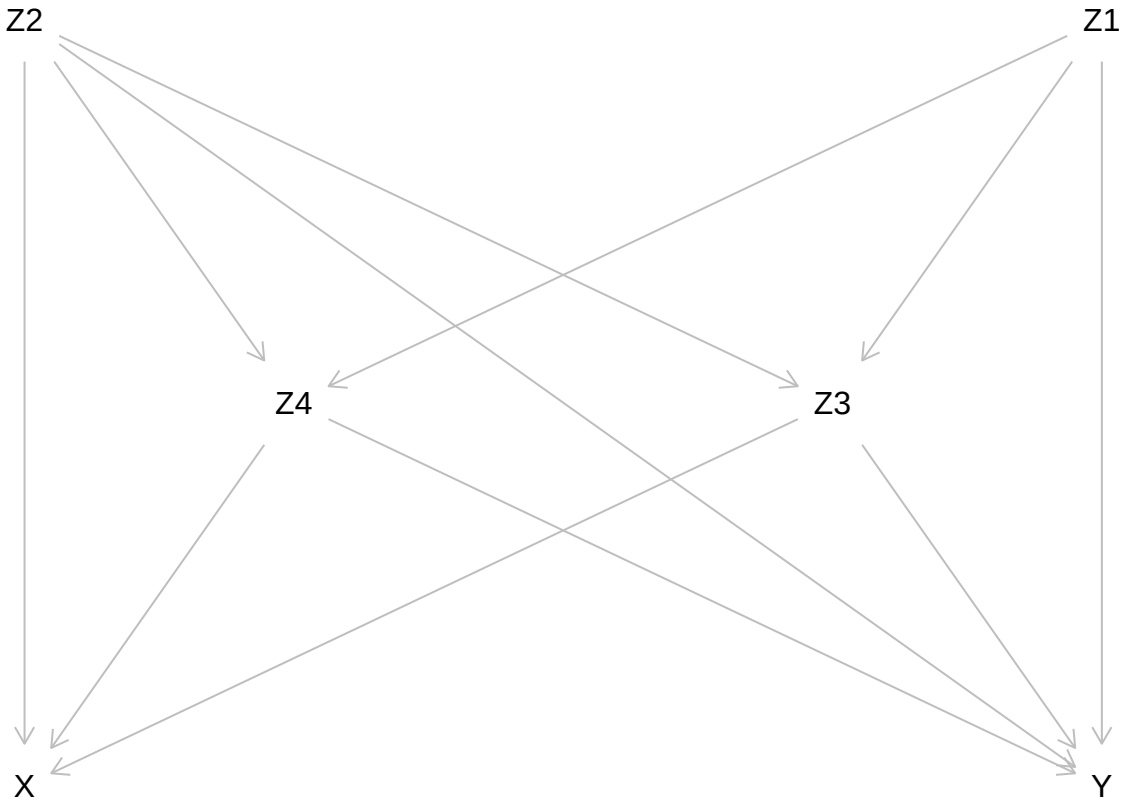
Sources of inspiration: Luque Fernandez, M.A. et al. (2018) *Stat Med* 2018;37(16):2530-2546 and Smith et al. (2022) *Stat Med* 2022;41(2):407-432.

14.1 Introduction

We shall illustrate with simulated data the estimation of causal effects of a binary exposure X when the outcome Y is also binary, and there is a set of four covariates $Z = (Z_1, Z_2, Z_3, Z_4)$. As a background story, we imagine a population of cancer patients, in whom the variables and the assumed marginal distributions of the covariates are

Variable	Description
X	treatment; 1: radiotherapy only, 0: radiotherapy + chemotherapy
Y	death during one year after diagnosis of cancer
Z_1	sex; 0: man, 1: woman; $Z_1 \sim$ Bern(0.5)
Z_2	age group 0; `young`, 1: `old`; $Z_2 \sim$ Bern(0.65)
Z_3	stage of cancer; 4 classes; $Z_3 \sim$ DiscUnif$(1, \dots, 4)$
Z_4	comorbidity score; 5 classes; $Z_3 \sim$ DiscUnif$(1, \dots, 5)$

For simplicity, covariates Z_3 and Z_4 are treated as continuous variables in the models. The assumed causal diagram is drawn using `dagitty` and is shown below.



For more generic notation, the probabilities of $Y = 1$ will be expressed as expectations, e.g.

$$E(Y^{X=x}) = P(Y^{X=x} = 1) \quad \text{and} \quad E(Y|X = x, Z = z) = P(Y = 1|X = x, Z = z),$$

where Z is the vector of relevant covariates.

The same principle is applied in expressing the conditional probability of $X = 1$, i.e. being exposed, given $Z = z$:

$$E(X|Z = z) = P(X = 1|Z = z).$$

The fitted or predicted probabilities of $Y = 1$ are denoted as fitted $\widehat{Y}$ or predicted values $\widetilde{Y}$ of Y with pertinent subscripts and/or superscripts. – Both X and Y are modelled by logistic regression. The expit-function or inverse of the logit function is defined: $\text{expit}(u) = 1/(1 + e^{-u})$, $u \in R$. This is equal to the cumulative distribution function of the standard logistic distribution, the values of which are returned in R by `plogis(u)`. The R function that returns values of the logit-function is `qlogis()`.

The true model assumed for the dependence of exposure X on covariates:

$$E(X|Z_1 = z_1, \ldots, Z_4 = z_4) = \text{expit}(-5 + 0.05z_2 + 0.25z_3 + 0.5z_4 + 0.4z_2z_4).$$

The assumed true model for the outcome is

$$E(Y|X = x, Z_1 = z_1, \ldots, Z_4 = z_4) = \text{expit}(-1 + x - 0.1z_1 + 0.35z_2 + 0.25z_3 + 0.20z_4 + 0.15z_2z_4)$$

Note that X does not depend on Z_1, and that in both models there is a product term Z_2Z_4, the effect of which appears weaker for the outcome model.

14.2 Control of confounding

1. Based on inspection of the causal diagram, can you provide justification for the claim that variables Z_2, Z_3 and Z_4 form a proper subset of the four covariates, which is sufficient to block all backdoor paths between X and Y and thus remove confounding?

2. Even though we have such a minimal sufficient set as indicated in item (a), why it still might be useful to include covariate Z_1, too, when modelling the outcome?

14.3 Generation of target population and true models

1. Load the necessary packages.

```
library(Epi)
library(stdReg)
library(PSweight)
```

2. Define two R-functions, which compute the expected values for the exposure and those for the outcome based on the assumed true exposure model and the true outcome model, respectively.

```
EX <- function(z2, z3, z4) {
  plogis(-5 + 0.05 * z2 + 0.25 * z3 + 0.5 * z4 + 0.4 * z2 * z4)
}
EY <- function(x, z1, z2, z3, z4) {
  plogis(-1 + x - 0.1 * z1 + 0.35 * z2 + 0.25 * z3 +
    0.20 * z4 + 0.15 * z2 * z4)
}
```

3. Define the function for the generation of data by simulating random values from pertinent probability distributions based on the given assumptions.

```
genData <- function(N) {
  z1 <- rbinom(N, size = 1, prob = 0.5) # Bern(0.5)
  z2 <- rbinom(N, size = 1, prob = 0.65) # Bern(0.65)
  z3 <- trunc(runif(N, min = 1, max = 5), digits = 0) # DiscUnif(1,4)
  z4 <- trunc(runif(N, min = 1, max = 6), digits = 0) # DiscUnif(1,5)
  x <- rbinom(N, size = 1, prob = EX(z2, z3, z4))
  y <- rbinom(N, size = 1, prob = EY(x, z1, z2, z3, z4))
  data.frame(z1, z2, z3, z4, x, y)
}
```

4. Generate a data frame `dd` for a big target population of 500000 subjects

```
N <- 500000
set.seed(7777)
dd <- genData(N)
```

14.4 Factual and counterfactual risks - associational and causal contrasts

1. Compute the factual risks of death for the two exposure groups

$$E(Y|X=x) = P(Y=1|X=x) = \frac{P(Y=1 \;\&\; X=x)}{P(X=x)}, \quad x = 0, 1,$$

in the whole target population, as well as their associational contrasts: risk difference, risk ratio, and odds ratio. Before that define a useful function

```
Contr <- function(mu1, mu0) {
  RD <- mu1 - mu0
  RR <- mu1 / mu0
  OR <- (mu1 / (1 - mu1)) / (mu0 / (1 - mu0))
  return(c(mu1, mu0, RD = RD, RR = RR, OR = OR))
}
Ey1fact <- with(dd, sum(y == 1 & x == 1) / sum(x == 1))
```

```
Ey0fact <- with(dd, sum(y == 1 & x == 0) / sum(x == 0))
round(Contr(Ey1fact, Ey0fact), 4)
```

How much bigger is the risk of death of those factually exposed to radiotherapy only as compared with those receiving chemotherapy, too?

2. Compute now the true **counterfactual** or **potential risks** of death

$$E(Y_i^{X_i=x}) = P(Y_i^{X_i=x} = 1) = \pi_i^{X_i=x}$$

for each individual under the alternative treatments or exposure values $x = 0, 1$ with given covariate values, then the average or overall counterfactual risks $E(Y^{X=1}) = \pi^1$ and $E(Y^{X=0}) = \pi^0$ in the population, and finally the true **marginal causal contrasts** for the effect of X:

$$\text{RD} = E(Y^{X=1}) - E(Y^{X=0}), \qquad \text{RR} = E(Y^{X=1})/E(Y^{X=0}),$$
$$\text{OR} = \frac{E(Y^{X=1})/[1 - E(Y^{X=1})]}{E(Y^{X=0})/[1 - E(Y^{X=0})]}$$

```
dd <- transform(dd,
  EY1.ind = EY(x = 1, z1, z2, z3, z4),
  EY0.ind = EY(x = 0, z1, z2, z3, z4)
)
EY1pot <- mean(dd$EY1.ind)
EY0pot <- mean(dd$EY0.ind)
round(Contr(EY1pot, EY0pot), 4)
```

3. Compare the associational contrasts computed in item 4.1 with the causal contrasts in item 4.2. What do you conclude about confoundedness of the associational contrasts?

14.5 Outcome modelling and estimation of causal contrasts by g-formula

As the first approach for estimating causal contrasts of interest we apply the method of **standardization** or **g-formula**. It is based on a hopefully realistic enough model for $E(Y|X = x, Z = z)$, i.e. how the risk of outcome is expected to depend on the exposure variable X and on a sufficient set Z of confounders. The potential or counterfactual risks $E(Y^{X=x}), x = 0, 1$, are marginal expectations of the above quantities, standardized over the joint distribution of the confounders Z in the target population.

$$E(Y^{X=x}) = E_Z[E(Y|X = x, Z)] = \int E(Y|X = x, Z = z)dF_Z(z), \quad x = 0, 1.$$

1. Assume now a *slightly misspecified* model `mY` for the outcome, which contains only main effect terms of the explanatory variables:

$$\pi_i = E(Y_i|X_i = x_i, Z_{i1} = z_{i1}, \ldots, Z_{i4} = z_{i4}) = \text{expit}\left(\beta_0 + \delta x_i + \sum_{j=1}^{4} \beta_j z_{ij}\right)$$

Fit this model on the target population using function `glm()`

```
mY <- glm(y ~ x + z1 + z2 + z3 + z4, family = binomial, data = dd)
round(ci.lin(mY, Exp = TRUE)[, c(1, 5)], 3)
```

There is not much idea in looking at the standard errors or confidence intervals in such a big population.

2. For each subject i, compute the fitted individual risk $\widehat{Y}_i$ as well as the predicted potential (counterfactual) risks $\widetilde{Y}_i^{X_i=x}$ for both exposure levels $x = 0, 1$ separately, keeping the individual values of the Z-variables as they are.

```
dd$yh <- predict(mY, type = "response") #  fitted values
dd$yp1 <- predict(mY, newdata = data.frame(
  x = rep(1, N), # x=1
  dd[, c("z1", "z2", "z3", "z4")]
), type = "response")
dd$yp0 <- predict(mY, newdata = data.frame(
  x = rep(0, N), # x=0
  dd[, c("z1", "z2", "z3", "z4")]
), type = "response")
```

3. Applying the method of **standardization** or **g-formula** compute now the point estimates

$$\widehat{E}_g(Y^{X=x}) = \frac{1}{n}\sum_{i=1}^{n} \widetilde{Y}_i^{X_i=x}, \quad x = 0, 1.$$

of the two counterfactual risks $E(Y^{X=1}) = \pi^1$ and $E(Y^{X=0}) = \pi^0$ as well as those of the marginal causal contrasts

```
EY1pot.g <- mean(dd$yp1)
EY0pot.g <- mean(dd$yp0)
round(Contr(EY1pot.g, EY0pot.g), 4)
```

The marginal expectations $E_Z[E(X = x, Z)]$ taken over the joint distribution of the confounders Z are empirically estimated from the actual data representing the target population by simply computing the arithmetic means of the individually predicted values $\widetilde{Y}_i^{X_i=x}$ of the outcome for the two exposure levels.

Compare the estimated contrasts with the true ones in item 4.2 above. How big is the bias due to slight misspecification of the outcome model? Compare in particular the estimate of the marginal OR here with the conditional OR obtained in item 5.1 from the pertinent coefficient in the logistic model. Which one is closer to 1?

4. Perform the same calculations using the tools in package `stdReg` (see Sjölander 2016)

```
mY.std <- stdGlm(fit = mY, data = dd, X = "x")
summary(mY.std)
round(summary(mY.std, contrast = "difference", reference = 0)$est.table, 4)
round(summary(mY.std, contrast = "ratio", reference = 0)$est.table, 4)
round(summary(mY.std, transform = "odds",
              contrast = "ratio", reference = 0)$est.table, 4)
```

Check that you got the same point estimates as in the previous item. Again, the confidence intervals are not very meaningful when analysing the data covering the whole big target population. Of course, when applied to real sample data they are relevant. In `stdReg` package, the standard errors are obtained by the multivariate delta method built upon M-estimation and robust sandwich estimator of the pertinent covariance matrix, and approximate confidence intervals are derived from these in the usual way.

14.6 Inverse probability weighting (IPW) by propensity scores

The next method is based on weighting each individual observation by the inverse of the probability of belonging to that particular exposure group, which was realized, this probability being predicted by the determinants of exposure.

1. Fit first a somewhat misspecified model for the exposure including the main effects of the Z-variables only.

$$p_i = E(X_i|Z_{1i} = z_{1i}, \dots, Z_{4i} = z_{4i}) = \text{expit}(\gamma_0 + \gamma_1 z_{1i} + \gamma_2 z_{2i} + \gamma_3 z_{i3} + \gamma_4 z_{4i}), \quad i = 1, \dots N$$

```
mX <- glm(x ~ z1 + z2 + z3 + z4,
  family = binomial(link = logit), data = dd
```

```
)
round(ci.lin(mX, Exp = TRUE)[, c(1, 5)], 4)
```

2. Extract the **propensity scores**, i.e. fitted probabilities of belonging to exposure group 1: $\text{PS}_i = \widehat{p_i}$, and compare their distribution between the two groups.

```
dd$PS <- predict(mX, type = "response")
summary(dd$PS)
with(subset(dd, x == 0), plot(density(PS), lty = 2))
with(subset(dd, x == 1), lines(density(PS), lty = 1))
```

How different are the distributions? Are they sufficiently overlapping?

3. Compute the weights

$$W_i = \frac{1}{\text{PS}_i}, \quad \text{when} \quad X_i = 1,$$
$$W_i = \frac{1}{1 - \text{PS}_i}, \quad \text{when} \quad X_i = 0.$$

Look at the sum as well as the distribution summary of the weights in the exposure groups. The sum of weights should be close to N in both groups.

```
dd$w <- ifelse(dd$x == 1, 1 / dd$PS, 1 / (1 - dd$PS))
with(dd, tapply(w, x, sum))
```

4. Compute now the weighted estimates of the potential or counterfactual risks for both exposure categories

$$\widehat{E}_w(Y^{X=x}) = \frac{\sum_{i=1}^n \mathbf{1}_{\{X_i=x\}} W_i Y_i}{\sum_{i=1}^n \mathbf{1}_{\{X_i=x\}} W_i} = \frac{\sum_{X_i=x} W_i Y_i}{\sum_{X_i=x} W_i}, \quad x = 0, 1,$$

and their causal contrasts, for instance

$$\widehat{\text{RD}}_w = \widehat{E}_w(Y^{X=1}) - \widehat{E}_w(Y^{X=0}) = \frac{\sum_{i=1}^n X_i W_i Y_i}{\sum_{i=1}^n X_i W_i} - \frac{\sum_{i=1}^n (1 - X_i) W_i Y_i}{\sum_{i=1}^n (1 - X_i) W_i}$$

```
EY1pot.w <- sum(dd$x * dd$w * dd$y) / sum(dd$x * dd$w)
EY0pot.w <- sum((1 - dd$x) * dd$w * dd$y) / sum((1 - dd$x) * dd$w)
round(Contr(EY1pot.w, EY0pot.w), 4)
```

These estimates seem to be somewhat downward biased when comparing to true values. Could this be because of omitting the relatively strong product term effect of Z_2 and Z_4?

14.7 Improving IPW estimation and using R package `PSweight`

We now try to improve IPW-estimation by a richer exposure model. In computations we shall utilize the R package `PSweight` (see PSweight vignette).

1. First, we compute the propensity scores and weights from a more flexible exposure model, which contains all pairwise product terms of the parents of X. According to the causal diagram, Z_1 is not in that subset, so it is left out. The exposure model is specified and the weights are obtained as follows using function `SumStat()` in `PSweight`:

```
mX2 <- glm(x ~ (z2 + z3 + z4)^2, family = binomial, data = dd)
round(ci.lin(mX2, Exp = TRUE)[, c(1, 5)], 3)
psw2 <- SumStat(
  ps.formula = mX2$formula, data = dd,
  weight = c("IPW", "treated", "overlap")
)
dd$PS2 <- psw2$propensity[, 2]
dd$w2 <- ifelse(dd$x == 1, 1 / dd$PS2, 1 / (1 - dd$PS2))
```

```
plot(density(dd$PS2[dd$x == 0]), lty = 2)
lines(density(dd$PS2[dd$x == 1]), lty = 1)
```

Note that apart from ordinary IPW, other types of weights can also also obtained. These are relevant when estimating other kinds of causal contrasts, like "average treatment effect among the treated" (ATT, see below) and "average treatment effect in the overlap (or equipoise) population" (ATO).

2. `PSweight` includes some useful tools to examine the properties of the distribution and to check the balance of the propensity scores, for instance

```
plot(psw2, type = "balance", metric = "PSD")
```

It is desirable that the horisontal values of these measures for given weights are less than 0.1.

3. Estimation and reporting of the causal contrasts. For relative contrasts, the summary method provides the results on the log-scale; therefore exp-transformation

```
ipw2est <- PSweight(ps.formula = mX2, yname = "y", data = dd, weight = "IPW")
ipw2est
summary(ipw2est)
(logRR.ipw2 <- summary(ipw2est, type = "RR"))
round(exp(logRR.ipw2$estimates[c(1, 4, 5)]), 3)
round(exp(summary(ipw2est, type = "OR")$estimates[c(1, 4, 5)]), 3)
```

Compare these with the previous IPW estimate as well as the true values. Have we obtained nearly unbiased results?

The standard errors provided by `PSweight` are by default based on the empirical sandwich covariance matrix and application of delta method as appropriate. Bootstrapping is also possible but is computationally very intensive and is recommended to be used only in relatively small samples.

14.8 Effect of exposure among those exposed

If we are interested in the causal contrasts describing the **effect of exposure among those exposed** (like ATE), the relevant factual and counterfactual risks in that subset are

$$\pi_1^1 = E(Y^{X=1}|X=1) = E(Y|X=1) = \pi_1,$$
$$\pi_1^0 = E(Y^{X=0}|X=1) = \sum_{X_i=1} E(Y|X=0, Z=z)P(Z=z|X=1)$$

We are thus making and "observed vs. expected" comparison, in which the z-specific risks in the unexposed are weighted by the distribution of Z in the exposed subset of the target population. The risks and their contrasts are estimated from the fit of the outcome model:

```
EY1att.g <- mean(subset(dd, x == 1)$yp1)
EY0att.g <- mean(subset(dd, x == 1)$yp0)
round(Contr(EY1att.g, EY0att.g), 4)
```

Compare the results here with those for the whole target population. What do you observe?

2. Have you any guess about the causal effect of exposure among the unexposed; is it bigger or smaller than among the exposed or among the whole population?

3. Incidentally, the true causal contrasts among the exposed based on the true model are similarly obtained from the quantities in item 4.2 above:

```
EY1att <- mean(subset(dd, x == 1)$EY1.ind)
EY0att <- mean(subset(dd, x == 1)$EY0.ind)
round(Contr(EY1att, EY0att), 4)
```

Compare the estimates in the previous item with the true values obtained here.

4. When wishing to estimate the effect of exposure among the exposed using the IPW approach, then the weights are $W_i = 1$ for the exposed and $W_i = \mathrm{PS}_i/(1-\mathrm{PS}_i)$ for the unexposed. Call again `PSweight` but with another choice of weight:

```
psatt <- PSweight(ps.formula = mX2, yname = "y", data = dd, weight = "treated")
psatt
round(summary(psatt)$estimates[1], 4)
round(exp(summary(psatt, type = "RR")$estimates[1]), 3)
round(exp(summary(psatt, type = "OR")$estimates[1]), 3)
```

Compare the results here with those obtained by g-formula in item 8.1 and with the true contrasts above.

14.9 Double robust estimation by augmented IPW

Let us attempt to correct the estimates by a **double robust** (DR) approach called **augmented IPW estimation** (AIPW), which combines the g-formula and the IPW approach. The classical AIPW-estimator can be expressed in two ways: either an IPW-corrected g-formula estimator, or a g-corrected IPW-estimator.

$$\begin{aligned}\widehat{E}_a(Y^{X=x}) &= \widehat{E}_g(Y^{X=x}) + \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{\{X_i=x\}} W_i (Y_i - \widetilde{Y}_i^{X_i=x}) \\ &= \widehat{E}_w(Y^{X=x}) + \frac{1}{n}\sum_{i=1}^{n} (1 - \mathbf{1}_{\{X_i=x\}} W_i) \widetilde{Y}_i^{X_i=x}.\end{aligned}$$

1. We shall first combine the results from the slightly misspecified outcome model with those from the more misspecified exposure model.

```
EY1pot.a <- EY1pot.g + mean( 1*(dd$x==1) * dd$w * (dd$y - dd$yp1) )
EY0pot.a <- EY0pot.g + mean( 1*(dd$x==0) * dd$w * (dd$y - dd$yp0) )
round(Contr(EY1pot.a, EY0pot.a), 4)
```

Compare these results with those obtained by g-formula and by non-augmented IPW method. Was augmentation successful?

2. Let us then look, how close we get when combining the results from the slightly misspecified outcome model with the correct exposure model using the alternative AIPW-formula

```
EY1pot.w2 <- ipw2est$muhat[2]
EY0pot.w2 <- ipw2est$muhat[1]
EY1pot.a2 <- EY1pot.w2 + mean( (1 - 1*(dd$x==1) * dd$w2) * dd$yp1 )
EY0pot.a2 <- EY0pot.w2 + mean( (1 - 1*(dd$x==0) * dd$w2) * dd$yp0 )
round(Contr(EY1pot.a2, EY0pot.a2), 4)
```

Compare the results with previous ones. How successful was augmentation now?

AIPW-estimates and confidence intervals for the causal contrasts of interest can be obtained, for instance, using `PSweight` by adding the model formula of the outcome model as the value for the argument `out.formula`.

14.10 Double robust, targeted maximum likelihood estimation (TMLE)

We now consider now another double robust approach, known as **targeted maximum likelihood estimation** (TMLE). It also corrects the estimator obtained from the outcome model by elements that are derived from the exposure model. The corrections are, though, not as intuitive as those in AIPW. See Schuler and Rose (2017) for more details.

1. The first step is to utilize the propensity scores obtained above for the correct exposure model and define the "clever covariates"

```
dd$H1 <- dd$x / dd$PS2
dd$H0 <- (1 - dd$x) / (1 - dd$PS2)
```

2. Then, a working model is fitted for the outcome, in which the clever covariates are explanatory variables, but the model also includes the previously fitted linear predictor $\hat{\eta}_i = \text{logit}(\widehat{Y}_i)$ from the original outcome model `mY` as an offset term; see item 5.2. Moreover, the intercept is removed.

```
epsmod <- glm(y ~ -1 + H0 + H1 + offset(qlogis(yh)),
  family = binomial(link = logit), data = dd
)
eps <- coef(epsmod)
eps
```

3. The logit-transformed predicted values $\widetilde{Y}_i^{X_i=1}$ and $\widetilde{Y}_i^{X_i=0}$ of the potential or counterfactual individual risks from the original outcome model are now corrected by the estimated coefficients of the clever covariates, and the corrected predictions are returned to the original scale.

```
ypred0.H <- plogis(qlogis(dd$yp0) + eps[1] / (1 - dd$PS2))
ypred1.H <- plogis(qlogis(dd$yp1) + eps[2] / dd$PS2)
```

4. Estimates of the causal contrasts:

```
EY0pot.t <- mean(ypred0.H)
EY1pot.t <- mean(ypred1.H)
round(Contr(EY1pot.t, EY0pot.t), 4)
```

Compare these with previous results and with the true values.

14.11 Double robust estimation with `AIPW` and `tmle` packages

It may be difficult to specify conventional generalized linear models or even generalized additive models for exposure and outcome which are sufficiently realistic, yet which do not suffer from overfitting. Modern approaches of **statistical learning**, aka "machine learning" provide tools for flexible modelling, which may be used to reduce the risk of misspecification, if thoughtfully applied.

There are a few R packages in which some general algorithmic approaches for supervised learning are implemented for estimating causal parameters. For instance, package `AIPW` (see Zhong et al., 2021) utilizes several learning algorithms for exposure and outcome modelling and then performs AIPW estimation of the parameters of interest coupled with calculation of confidence intervals. Package `tmle` (see Karim and Frank, 2021) performs same tasks but uses the TMLE approach in estimation.

Both `AIPW` and `tmle` lean on the `SuperLearner` package, which uses multiple learning algorithms (e.g. GLM, GAM, Random Forest, Recursive Partitioning, Gradient Boosting, etc.) for constructing predictions of the counterfactual quantities, and then creates an optimal weighted average of those models, aka an "ensemble". These algorithms are computationally highly intensive. Fitting models with only 3 or 4 covariates as in this practical on our target cohort of 500,000 subjects would take hours on an ordinary laptop. With a study population of 5000 it takes several minutes.

Chapter 15

Time-dependent variables and multiple states

The following practical exercise is based on the data from paper:

> P Hovind, L Tarnow, P Rossing, B Carstensen, and HH Parving: Improved survival in patients obtaining remission of nephrotic range albuminuria in diabetic nephropathy. *Kidney Int*, **66**(3):1180–1186, Sept 2004.

You can find a `.pdf`-version of the paper here: http://BendixCarstensen.com/AdvCoh/papers/Hovind.2004.pdf

15.1 The renal failure dataset

The dataset `renal.dta` contains data on follow up of 125 patients from Steno Diabetes Center. They enter the study when they are diagnosed with nephrotic range albuminuria (NRA). This is a condition where the levels of albumin in the urine is exceeds a certain level as a sign of kidney disease. The levels may however drop as a consequence of treatment, this is called remission. Patients exit the study at death or kidney failure (dialysis or transplant).

Variable	Description
`id`	Patient id
`sex`	1=male, 2=female
`dob`	Date of birth
`doe`	Date of entry into the study (2.5 years after NRA)
`dor`	Date of remission. Missing if no remission has occurred
`dox`	Date of exit from study
`event`	Exit status: 1,2,3=event (death, ESRD), 0=censored

1. The dataset is in Stata-format, so you must read the dataset using `read.dta` from the `foreign` package (which is part of the standard `R`-distribution). At the same time, convert `sex` to a proper factor. Choose where to read the dataset.

```
library(Epi)
library(survival)
library(mgcv)
library(foreign)
# renal <- read.dta(
#  "https://raw.githubusercontent.com/SPE-R/SPE/master/pracs/data/renal.dta")
renal <- read.dta("http://BendixCarstensen.com/SPE/data/renal.dta")
renal$sex <- factor(renal$sex, labels = c("M", "F"))
head(renal)
```

2. Use the `Lexis` function to declare the data as survival data with age, calendar time and time since entry into the study as timescales. Label any event > 0 as *ESRD*, i.e. renal death (death of kidney (transplant or dialysis), or person). Note that you must make sure that the *alive* state (here `NRA`) is the first, as `Lexis` assumes that everyone starts in this state (unless of course `entry.status` is specified):

```
Lr <- Lexis(entry = list(per = doe,
                         age = doe - dob,
                         tfi = 0),
            exit = list(per = dox),
      exit.status = factor(event > 0, labels = c("NRA", "ESRD")),
            data = renal)
```

```
NOTE: entry.status has been set to "NRA" for all.
```

```
str(Lr)
```

```
Classes 'Lexis' and 'data.frame':   125 obs. of  14 variables:
 $ per    : num  1996 1990 1988 1995 1988 ...
 $ age    : num  28.1 30.2 25.8 44.5 26.6 ...
 $ tfi    : num  0 0 0 0 0 0 0 0 0 0 ...
 $ lex.dur: num  1.08 6.6 5.39 8.75 16.07 ...
 $ lex.Cst: Factor w/ 2 levels "NRA","ESRD": 1 1 1 1 1 1 1 1 1 1 ...
 $ lex.Xst: Factor w/ 2 levels "NRA","ESRD": 2 2 2 1 1 2 2 1 2 1 ...
 $ lex.id : int  1 2 3 4 5 6 7 8 9 10 ...
 $ id     : num  17 26 27 33 42 46 47 55 62 64 ...
 $ sex    : Factor w/ 2 levels "M","F": 1 2 2 1 2 2 1 1 2 1 ...
 $ dob    : num  1968 1959 1962 1951 1961 ...
 $ doe    : num  1996 1990 1988 1995 1988 ...
 $ dor    : num  NA 1990 NA 1996 1997 ...
 $ dox    : num  1997 1996 1993 2004 2004 ...
 $ event  : num  2 1 3 0 0 2 1 0 2 0 ...
 - attr(*, "time.scales")= chr [1:3] "per" "age" "tfi"
 - attr(*, "time.since")= chr [1:3] "" "" ""
 - attr(*, "breaks")=List of 3
  ..$ per: NULL
  ..$ age: NULL
  ..$ tfi: NULL
```

```
summary(Lr)
```

```
Transitions:
     To
From  NRA ESRD  Records:  Events: Risk time:  Persons:
  NRA  48   77       125       77       1085       125
```

Make sure you know what the variables in `Lr` stand for.

3. Visualize the follow-up in a Lexis-diagram, by using the `plot` method for `Lexis` objects.

```
plot(Lr, col = "black", lwd = 3)
```

```
subset(Lr, age < 0)
```

```
 lex.id  per   age tfi lex.dur lex.Cst lex.Xst  id sex  dob  doe dor  dox event
     88 1989 -38.8   0     3.5     NRA    ESRD 586   M 2028 1989  NA 1993     1
```

What is wrong here? List the data for the person with negative entry age.

4. Correct the data and make a new plot, for example by:

```
Lr <- transform(Lr, age = ifelse(dob > 2000, age + 100, age),
                    dob = ifelse(dob > 2000, dob - 100, dob))
subset(Lr, id == 586)
```

```
 lex.id  per  age tfi lex.dur lex.Cst lex.Xst  id sex  dob  doe dor  dox event
     88 1989 61.2   0     3.5     NRA    ESRD 586   M 1928 1989  NA 1993     1
```

```
plot(Lr, col = "black", lwd = 3)
```

5. Now make a Cox-regression analysis of ESRD occurrence with the variables sex and age at entry into the study, using time since entry to the study as time scale.

```
mc <- coxph(Surv(lex.dur, lex.Xst == "ESRD")
            ~ I(age / 10) + sex, data = Lr)
summary(mc)
```

```
Call:
coxph(formula = Surv(lex.dur, lex.Xst == "ESRD") ~ I(age/10) +
    sex, data = Lr)

  n= 125, number of events= 77

             coef exp(coef) se(coef)     z Pr(>|z|)
I(age/10)  0.551     1.736    0.140  3.93  8.4e-05
sexF      -0.182     0.834    0.273 -0.67     0.51

          exp(coef) exp(-coef) lower .95 upper .95
I(age/10)     1.736      0.576     1.319      2.28
sexF          0.834      1.199     0.489      1.42

Concordance= 0.612  (se = 0.036 )
Likelihood ratio test= 16.1  on 2 df,   p=3e-04
Wald test            = 16.4  on 2 df,   p=3e-04
Score (logrank) test = 16.8  on 2 df,   p=2e-04
```

What is the hazard ratio between males and females? Between two persons who differ 10 years in age at entry?

6. The main focus of the paper was to assess whether the occurrence of remission (return to a lower level of albumin excretion, an indication of kidney recovery) influences mortality. *Remission* is a time-dependent variable which is initially 0, but takes the value 1 when remission occurs. In order to handle this, each person who sees a remission must have two records:

   - One record for the time before remission, where entry is `doe`, exit is `dor`, remission is 0, and event is 0.
   - One record for the time after remission, where entry is `dor`, exit is `dox`, remission is 1, and event is 0 or 1 according to whether the person had an event at `dox`.

   This is accomplished using the `cutLexis` function on the `Lexis` object, where we introduce a remission state `Rem`. Also use `split.state=TRUE` to have different ESRD states according to whether a person had had remission or not prioer to ESRD. The statement to do this is:

```
Lc <- cutLexis(Lr, cut = Lr$dor, # where to cut follow up
               timescale = "per",  # what timescale are we referring to
               new.state = "Rem",  # name of the new state
             split.state = TRUE)   # different states depending on previous
summary(Lc)
```

```
Transitions:
     To
```

```
From  NRA Rem ESRD ESRD(Rem)  Records:  Events: Risk time:  Persons:
  NRA  24  29   69         0       122       98        825       122
  Rem   0  24    0         8        32        8        260        32
  Sum  24  53   69         8       154      106       1085       125
```

List the records from a few select persons (choose values for `lex.id`, using for example `subset(Lc, lex.id %in% c(5,7,9))`).

7. Now show how the states are connected and the number of transitions between them by using `boxes`. This is an interactive command that requires you to click in the graph window:

```
boxes(Lc)
```

It has a couple of fancy arguments, try:

```
boxes(Lc, boxpos = TRUE, scale.R = 100, show.BE = TRUE, hm = 1.5, wm = 1.5)
```

You may even be tempted to read the help page for `boxes.Lexis` …

8. Plot a Lexis diagram where different coloring is used for different segments of the follow-up. The `plot.Lexis` function draws a line for each record in the dataset, so you can index the coloring by `lex.Cst` and `lex.Xst` as appropriate — indexing by a factor corresponds to indexing by the *index number* of the factor levels, so you must be know which order the factor levels are in:

```
levels(Lc) # names and order of states in lex.Cst and lex.Xst
```

```
[1] "NRA"       "Rem"       "ESRD"      "ESRD(Rem)"
```

```
par(mai = c(3, 3, 1, 1) / 4, mgp = c(3, 1, 0) / 1.6)
plot(Lc, col = c("red", "limegreen")[Lc$lex.Cst],
        xlab = "Calendar time", ylab = "Age",
         lwd = 3, grid = 0:20 * 5, las = 1,
        xlim = c(1970, 2010), ylim = c(20, 70),
        xaxs = "i", yaxs = "i")
points(Lc, pch = c(NA, NA, 16, 16)[Lc$lex.Xst],
           col = c("red", "limegreen", "transparent", "transparent")[Lc$lex.Cst])
points(Lc, pch = c(NA, NA, 1, 1)[Lc$lex.Xst],
           col = "black", lwd = 2)
```

9. Make a Cox-regression of mortality rates (i.e. endpoint `ESRD` or `ESRD(Rem)`) with sex, age at entry and remission as explanatory variables, using time since entry as timescale, and include `lex.Cst` as time-dependent variable, and indicate that each record represents follow-up from `tfi` to `tfi+lex.dur`. Make sure that you know why what goes where here in the call to `coxph`.

```
(EP <- levels(Lc)[3:4])           # define EndPoint states
```

```
[1] "ESRD"      "ESRD(Rem)"
```

```
m1 <- coxph(Surv(tfi,             # entry time
                 tfi + lex.dur,   # exit time
                 lex.Xst %in% EP) # event
            ~ sex + I((doe - dob - 50) / 10) + # fixed covariates
              (lex.Cst == "Rem"),              # time-dependent variable
            data = Lc)
summary(m1)
```

```
Call:
coxph(formula = Surv(tfi, tfi + lex.dur, lex.Xst %in% EP) ~ sex +
    I((doe - dob - 50)/10) + (lex.Cst == "Rem"), data = Lc)

  n= 154, number of events= 77
```

```
                          coef exp(coef) se(coef)     z Pr(>|z|)
sexF                   -0.0553    0.9462   0.2750 -0.20  0.84052
I((doe - dob - 50)/10)  0.5219    1.6852   0.1366  3.82  0.00013
lex.Cst == "Rem"TRUE   -1.2624    0.2830   0.3848 -3.28  0.00104

                       exp(coef) exp(-coef) lower .95 upper .95
sexF                       0.946      1.057     0.552     1.622
I((doe - dob - 50)/10)     1.685      0.593     1.290     2.202
lex.Cst == "Rem"TRUE       0.283      3.534     0.133     0.602

Concordance= 0.664  (se = 0.033 )
Likelihood ratio test= 30.3  on 3 df,   p=1e-06
Wald test            = 27.1  on 3 df,   p=6e-06
Score (logrank) test = 29.4  on 3 df,   p=2e-06
```

What is the effect of of remission on the rate of ESRD?

15.2 Splitting the follow-up time

In order to explore the effect of remission on the rate of ESRD, we split the data further into small pieces of follow-up. To this end we use the function `splitLexis`. The rates can then be modeled using a Poisson-model, and the shape of the effect of the underlying *rates* be explored. Furthermore, we can allow effects of both time since NRA and current age. To this end we will use splines, so we need the `splines` and also the `mgcv` packages.

10. Now split the follow-up time every month after entry, and verify that the number of events and risk time is the same as before and after the split:

```
sLc <- splitLexis(Lc, "tfi", breaks = seq(0, 30, 1/12))
summary( Lc)
```

```
Transitions:
     To
From  NRA Rem ESRD ESRD(Rem)  Records:  Events: Risk time:  Persons:
  NRA  24  29   69         0       122       98        825       122
  Rem   0  24    0         8        32        8        260        32
  Sum  24  53   69         8       154      106       1085       125
```

```
summary(sLc)
```

```
Transitions:
     To
From   NRA  Rem ESRD ESRD(Rem)  Records:  Events: Risk time:  Persons:
  NRA 9854   29   69         0      9952       98        825       122
  Rem    0 3139    0         8      3147        8        260        32
  Sum 9854 3168   69         8     13099      106       1085       125
```

11. Now fit the Poisson-model corresponding to the Cox-model we fitted previously. The function `Ns()` produces a model matrix corresponding to a piece-wise cubic function, modeling the baseline hazard explicitly (think of the `Ns` terms as the baseline hazard that is not visible in the Cox-model). You can use the wrapper function `glm.Lexis`

```
mp <- glm.Lexis(sLc,
                ~ Ns(tfi, knots = c(0, 2, 5, 10)) +
                  sex + I((doe - dob - 40) / 10) +
                  I(lex.Cst == "Rem"))
```

```
stats::glm Poisson analysis of Lexis object sLc with log link:
Rates for transitions:
NRA->ESRD
Rem->ESRD(Rem)
```

```
ci.exp(mp)
```

```
                                exp(Est.)    2.5%   97.5%
(Intercept)                        0.0166 0.00396   0.070
Ns(tfi, knots = c(0, 2, 5, 10))1    5.1892 1.94920  13.815
Ns(tfi, knots = c(0, 2, 5, 10))2   34.2000 1.76482 662.755
Ns(tfi, knots = c(0, 2, 5, 10))3    4.4332 2.17998   9.015
sexF                                0.9175 0.53626   1.570
I((doe - dob - 40)/10)              1.7008 1.30081   2.224
I(lex.Cst == "Rem")TRUE             0.2793 0.13140   0.594
```

How does the effects of sex change from the Cox-model?

12. Try instead using the `gam` function from the `mgcv` package. There is convenience wrapper for this for `Lexis` objects as well:

```
mx <- gam.Lexis(sLc,
                ~ s(tfi, k = 10) +
                  sex + I((doe - dob - 40) / 10) +
                  I(lex.Cst == "Rem"))
```

```
mgcv::gam Poisson analysis of Lexis object sLc with log link:
Rates for transitions:
NRA->ESRD
Rem->ESRD(Rem)
```

```
ci.exp(mp, subset = c("Cst", "doe", "sex"))
```

```
                        exp(Est.)  2.5% 97.5%
I(lex.Cst == "Rem")TRUE     0.279 0.131 0.594
I((doe - dob - 40)/10)      1.701 1.301 2.224
sexF                        0.918 0.536 1.570
```

```
ci.exp(mx, subset = c("Cst", "doe", "sex"))
```

```
                        exp(Est.)  2.5% 97.5%
I(lex.Cst == "Rem")TRUE     0.278 0.131 0.592
I((doe - dob - 40)/10)      1.699 1.300 2.222
sexF                        0.931 0.544 1.595
```

We see that there is virtually no difference between the two approaches in terms of the regression parameters.

13. Extract the regression parameters from the models using `ci.exp` and compare with the estimates from the Cox-model:

```
ci.exp(mx, subset = c("sex", "dob", "Cst"), pval = TRUE)
```

```
                        exp(Est.)  2.5% 97.5%        P
sexF                        0.931 0.544 1.595 0.794539
I((doe - dob - 40)/10)      1.699 1.300 2.222 0.000107
I(lex.Cst == "Rem")TRUE     0.278 0.131 0.592 0.000897
```

```
ci.exp(m1)
```

```
                       exp(Est.)  2.5% 97.5%
sexF                       0.946 0.552 1.622
```

```
I((doe - dob - 50)/10)     1.685 1.290 2.202
lex.Cst == "Rem"TRUE       0.283 0.133 0.602
```

```
round(ci.exp(mp, subset = c("sex", "dob", "Cst")) / ci.exp(m1), 2)
```

```
                         exp(Est.) 2.5% 97.5%
sexF                          0.97 0.97  0.97
I((doe - dob - 40)/10)        1.01 1.01  1.01
I(lex.Cst == "Rem")TRUE       0.99 0.99  0.99
```

How large are the differences in estimated regression parameters?

14. The model has the same assumptions as the Cox-model about proportionality of rates, but there is an additional assumption that the hazard is a smooth function of time since entry. It seems to be a sensible assumption (well, restriction) to put on the rates that they vary smoothly by time. No such restriction is made in the Cox model. The `gam` model optimizes the shape of the smoother by general cross-validation. Try to look at the shape of the estimated effect of `tfi`:

    ```
    plot(mx)
    ```

    Is this a useful plot?

15. However, `plot` (well, `plot.gam`) does not give you the *absolute* level of the underlying rates because it bypasses the intercept. So in order to predict the rates as a function of `tfi` and the covariates, we set up a prediction data frame. Note that age in the model specification is entered as `doe-dob`, hence the prediction data frame must have these two variables and not the age, but it is only the difference that matters for the prediction:

    ```
    nd <- data.frame(tfi = seq(0, 20, 0.1),
                     sex = "M",
                     doe = 1990,
                     dob = 1940,
                 lex.Cst = "NRA")
    str(nd)
    ```

    ```
    'data.frame':   201 obs. of  5 variables:
     $ tfi    : num  0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 ...
     $ sex    : chr  "M" "M" "M" "M" ...
     $ doe    : num  1990 1990 1990 1990 1990 1990 1990 1990 1990 1990 ...
     $ dob    : num  1940 1940 1940 1940 1940 1940 1940 1940 1940 1940 ...
     $ lex.Cst: chr  "NRA" "NRA" "NRA" "NRA" ...
    ```

    ```
    matshade(nd$tfi, cbind(ci.pred(mp, newdata = nd),
                           ci.pred(mx, newdata = nd)) * 100,
             plot = TRUE,
             type = "l", lwd = 3:4, col = c("black", "forestgreen"),
             log = "y", xlab = "Time since entry (years)",
             ylab = "ESRD rate (per 100 PY) for 50 year old men")
    ```

    Try to overlay with the corresponding prediction from the `glm` model using `Ns`.

Prediction from the multistate model

`If we want to make proper statements about the survival and disease probabilities we must know not only how the occurrence of remission influences the rate of death/ESRD, but we must also model the occurrence rate of remission itself.`

17. The rates of ESRD were modelled by a Poisson model with effects of age and time since NRA — in the models `mp` and `mx`. But if we want to model whole process we must also model the remission rates transition from `NRA` to `Rem`, but the number of events is rather small so we restrict covariates in this model

to only time since NRA and sex. Note that only the records that represent follow-up in the `NRA` state should be used; this is most easily done using the `gam.Lexis` function

```
mr <- gam.Lexis(sLc, ~ s(tfi, k = 10) + sex,
                     from = "NRA",
                       to = "Rem")
```

```
mgcv::gam Poisson analysis of Lexis object sLc with log link:
Rates for the transition:
NRA->Rem
```

```
summary(mr)
```

```
Family: poisson
Link function: log

Formula:
cbind(trt(Lx$lex.Cst, Lx$lex.Xst) %in% trnam, Lx$lex.dur) ~ s(tfi,
    k = 10) + sex

Parametric coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   -3.703      0.258  -14.34   <2e-16
sexF           0.958      0.373    2.57     0.01

Approximate significance of smooth terms:
        edf Ref.df Chi.sq p-value
s(tfi) 1.01   1.03   0.07    0.81

R-sq.(adj) =  -5.65e-06   Deviance explained = 1.65%
UBRE = -0.96024  Scale est. = 1         n = 9952
```

```
ci.exp(mr, pval = TRUE)
```

```
            exp(Est.)   2.5%  97.5%        P
(Intercept)    0.0247 0.0149 0.0409 1.25e-46
sexF           2.6062 1.2550 5.4120 1.02e-02
s(tfi).1       1.0050 0.8913 1.1332 9.35e-01
s(tfi).2       0.9962 0.8078 1.2287 9.72e-01
s(tfi).3       0.9982 0.9191 1.0841 9.66e-01
s(tfi).4       1.0019 0.8901 1.1278 9.75e-01
s(tfi).5       0.9984 0.9228 1.0802 9.69e-01
s(tfi).6       0.9982 0.9014 1.1053 9.72e-01
s(tfi).7       1.0017 0.9262 1.0834 9.66e-01
s(tfi).8       0.9945 0.6845 1.4449 9.77e-01
s(tfi).9       0.9479 0.6335 1.4183 7.95e-01
```

What is the remission rate-ratio between men and women?

18. If we want to predict the probability of being in each of the three states using these estimated rates, we may resort to analytical calculations of the probabilities from the estimated rates, which is actually doable in this case, but which will be largely intractable for more complicated models. Alternatively we can *simulate* the life course for a large group of (identical) individuals through a model using the estimated rates. That will give a simulated cohort (in the form of a `Lexis` object), and we can then just count the number of persons in each state at each of a set of time points. This is accomplished using the function `simLexis`. The input to this is the initial status of the persons whose life-course we shall simulate, and the transition rates in suitable form:

    - Suppose we want predictions for men aged 50 at NRA. The input is in the form of a `Lexis` object (where `lex.dur` and `lex.Xst` will be ignored). Note that in order to carry over the `time.scales`

and the `time.since` attributes, we construct the input object using `subset` to select columns, and `NULL` to select rows (see the example in the help file for `simLexis`):

```
inL <- subset(sLc, select = 1:11)[NULL, ]
str(inL)
```

```
Classes 'Lexis' and 'data.frame':   0 obs. of  11 variables:
 $ lex.id : int
 $ per    : num
 $ age    : num
 $ tfi    : num
 $ lex.dur: num
 $ lex.Cst: Factor w/ 4 levels "NRA","Rem","ESRD",..:
 $ lex.Xst: Factor w/ 4 levels "NRA","Rem","ESRD",..:
 $ id     : num
 $ sex    : Factor w/ 2 levels "M","F":
 $ dob    : num
 $ doe    : num
 - attr(*, "time.scales")= chr [1:3] "per" "age" "tfi"
 - attr(*, "time.since")= chr [1:3] "" "" ""
 - attr(*, "breaks")=List of 3
  ..$ per: NULL
  ..$ age: NULL
  ..$ tfi: num [1:361] 0 0.0833 0.1667 0.25 0.3333 ...
```

```
timeScales(inL)
```

```
[1] "per" "age" "tfi"
```

```
inL[1, "lex.id"] <- 1
inL[1, "per"] <- 2000
inL[1, "age"] <- 50
inL[1, "tfi"] <- 0
inL[1, "lex.Cst"] <- "NRA"
inL[1, "lex.Xst"] <- NA
inL[1, "lex.dur"] <- NA
inL[1, "sex"] <- "M"
inL[1, "doe"] <- 2000
inL[1, "dob"] <- 1950
inL <- rbind(inL, inL)
inL[2, "sex"] <- "F"
inL
```

```
 lex.id  per age tfi lex.dur lex.Cst lex.Xst id sex  dob  doe
      1 2000  50   0      NA     NRA    <NA> NA   M 1950 2000
      1 2000  50   0      NA     NRA    <NA> NA   F 1950 2000
```

```
str(inL)
```

```
Classes 'Lexis' and 'data.frame':   2 obs. of  11 variables:
 $ lex.id : num  1 1
 $ per    : num  2000 2000
 $ age    : num  50 50
 $ tfi    : num  0 0
 $ lex.dur: num  NA NA
 $ lex.Cst: Factor w/ 4 levels "NRA","Rem","ESRD",..: 1 1
 $ lex.Xst: Factor w/ 4 levels "NRA","Rem","ESRD",..: NA NA
 $ id     : num  NA NA
 $ sex    : Factor w/ 2 levels "M","F": 1 2
 $ dob    : num  1950 1950
```

```
 $ doe    : num  2000 2000
 - attr(*, "breaks")=List of 3
  ..$ per: NULL
  ..$ age: NULL
  ..$ tfi: num [1:361] 0 0.0833 0.1667 0.25 0.3333 ...
 - attr(*, "time.scales")= chr [1:3] "per" "age" "tfi"
 - attr(*, "time.since")= chr [1:3] "" "" ""
```

The other input for the simulation is the models for the transitions. This is given as a list with an element for each transient state (that is `NRA` and `Rem`), each of which is again a list with names equal to the states that can be reached from the transient state. The content of the list will be `glm` objects, in this case the models we just fitted, describing the transition rates:

```
Tr <- list("NRA" = list("Rem"  = mr,
                        "ESRD" = mx),
           "Rem" = list("ESRD(Rem)" = mx))
```

With this as input we can now generate a cohort, using `N=5` to simulate life course of 10 persons (5 for each set of starting values in `inL`):

```
(iL <- simLexis(Tr, inL, N = 10))
```

```
 lex.id  per  age  tfi lex.dur lex.Cst   lex.Xst id sex  dob  doe cens
      1 2000 50.0 0.00    2.97     NRA      ESRD NA   M 1950 2000 2020
      2 2000 50.0 0.00    0.44     NRA       Rem NA   M 1950 2000 2020
      2 2000 50.4 0.44   12.00     Rem ESRD(Rem) NA   M 1950 2000 2020
      3 2000 50.0 0.00   10.63     NRA      ESRD NA   M 1950 2000 2020
      4 2000 50.0 0.00   10.60     NRA      ESRD NA   M 1950 2000 2020
      5 2000 50.0 0.00   14.29     NRA      ESRD NA   M 1950 2000 2020
      6 2000 50.0 0.00    3.00     NRA      ESRD NA   M 1950 2000 2020
      7 2000 50.0 0.00   20.00     NRA       NRA NA   M 1950 2000 2020
      8 2000 50.0 0.00   11.21     NRA      ESRD NA   M 1950 2000 2020
      9 2000 50.0 0.00   15.42     NRA      ESRD NA   M 1950 2000 2020
     10 2000 50.0 0.00    9.74     NRA      ESRD NA   M 1950 2000 2020
     11 2000 50.0 0.00   12.24     NRA      ESRD NA   F 1950 2000 2020
     12 2000 50.0 0.00    8.18     NRA      ESRD NA   F 1950 2000 2020
     13 2000 50.0 0.00    4.78     NRA      ESRD NA   F 1950 2000 2020
     14 2000 50.0 0.00    1.23     NRA       Rem NA   F 1950 2000 2020
     14 2001 51.2 1.23    6.32     Rem ESRD(Rem) NA   F 1950 2000 2020
     15 2000 50.0 0.00   14.64     NRA      ESRD NA   F 1950 2000 2020
     16 2000 50.0 0.00    2.51     NRA       Rem NA   F 1950 2000 2020
     16 2003 52.5 2.51    4.66     Rem ESRD(Rem) NA   F 1950 2000 2020
     17 2000 50.0 0.00    0.38     NRA       Rem NA   F 1950 2000 2020
     17 2000 50.4 0.38   10.51     Rem ESRD(Rem) NA   F 1950 2000 2020
     18 2000 50.0 0.00    6.73     NRA      ESRD NA   F 1950 2000 2020
     19 2000 50.0 0.00    0.09     NRA       Rem NA   F 1950 2000 2020
     19 2000 50.1 0.09    6.74     Rem ESRD(Rem) NA   F 1950 2000 2020
     20 2000 50.0 0.00    4.24     NRA      ESRD NA   F 1950 2000 2020
```

```
summary(iL, by = "sex")
```

```
$M

Transitions:
     To
From  NRA Rem ESRD ESRD(Rem)  Records:  Events: Risk time:  Persons:
  NRA   1   1    8         0        10        9       98.3        10
  Rem   0   0    0         1         1        1       12.0         1
  Sum   1   1    8         1        11       10      110.3        10
```

```
$F

Transitions:
     To
From  NRA Rem ESRD ESRD(Rem)  Records:  Events: Risk time:  Persons:
  NRA   0   4    6         0        10       10       55.0        10
  Rem   0   0    0         4         4        4       28.2         4
  Sum   0   4    6         4        14       14       83.2        10
```

What type of object have you got as `iL`?

19. Now generate the life course of, say, 5,000 persons, and look at the summary. The `system.time` command is just to tell you how long it took, you may want to start with 500 just to see how long that takes.

```
system.time(sM <- simLexis(Tr, inL, N = 500, t.range = 12))
```

```
   user  system elapsed
   2.48    3.09    1.94
```

```
summary(sM, by = "sex")
```

```
$M

Transitions:
     To
From  NRA Rem ESRD ESRD(Rem)  Records:  Events: Risk time:  Persons:
  NRA  30  72  398         0       500      470       2721       500
  Rem   0  35    0        37        72       37        415        72
  Sum  30 107  398        37       572      507       3136       500

$F

Transitions:
     To
From  NRA Rem ESRD ESRD(Rem)  Records:  Events: Risk time:  Persons:
  NRA  32 143  325         0       500      468       2358       500
  Rem   0  89    0        54       143       54       1023       143
  Sum  32 232  325        54       643      522       3380       500
```

Why are there so many ESRD-events in the resulting data set?

20. Now count how many persons are present in each state at each time for the first 10 years after entry (which is at age 50). This can be done by using `nState`. Try:

```
nStm <- nState(subset(sM, sex == "M"), time.scale = "age",
               at = seq(0, 10, 0.1),
              from = 50)
nStf <- nState(subset(sM, sex == "F"), time.scale = "age",
               at = seq(0, 10, 0.1),
              from = 50)
head(nStf, 15)
```

```
      State
when    NRA Rem ESRD ESRD(Rem)
  50    500   0    0         0
  50.1  493   3    4         0
  50.2  488   6    6         0
  50.3  483   8    9         0
  50.4  471  15   14         0
  50.5  466  16   18         0
```

```
  50.6 458  20   22         0
  50.7 452  22   26         0
  50.8 448  25   27         0
  50.9 443  29   28         0
  51   439  32   29         0
  51.1 429  35   36         0
  51.2 420  37   43         0
  51.3 417  39   43         1
  51.4 413  41   45         1
```

What is in the object `nStf`?

21. With the counts of persons in each state at the designated time points (in `nStm`), compute the cumulative fraction over the states, arranged in order given by `perm`:

```
ppm <- pState(nStm, perm = c(2, 1, 3, 4))
ppf <- pState(nStf, perm = c(2, 1, 3, 4))
head(ppf)
```

```
      State
when     Rem   NRA ESRD ESRD(Rem)
  50   0.000 1.000    1         1
  50.1 0.006 0.992    1         1
  50.2 0.012 0.988    1         1
  50.3 0.016 0.982    1         1
  50.4 0.030 0.972    1         1
  50.5 0.032 0.964    1         1
```

```
tail(ppf)
```

```
      State
when     Rem   NRA  ESRD ESRD(Rem)
  59.5 0.200 0.312 0.926         1
  59.6 0.198 0.308 0.924         1
  59.7 0.198 0.306 0.924         1
  59.8 0.194 0.302 0.920         1
  59.9 0.196 0.300 0.918         1
  60   0.196 0.298 0.918         1
```

What do the entries in `ppf` represent?

22. Try to plot the cumulative probabilities using the `plot` method for `pState` objects:

```
plot(ppf)
```

Is this useful?

23. Now try to improve the plot so that it is easier to read, and easier to compare between men and women, for example:

```
par(mfrow = c(1, 2))
# Men
plot(ppm, col = c("limegreen", "red", "#991111", "forestgreen"))
lines(as.numeric(rownames(ppm)), ppm[, "NRA"], lwd = 2)
text(59.5, 0.95, "Men", adj = 1, col = "white", font = 2, cex = 1.2)
axis(side = 4, at = 0:10 / 10)
axis(side = 4, at = 1:99 / 100, labels = NA, tck = -0.01)
# Women
plot(ppf, col = c("limegreen", "red", "#991111", "forestgreen"),
          xlim = c(60, 50)) # inverted x-axis
lines(as.numeric(rownames(ppf)), ppf[, "NRA"], lwd = 2)
```

```
text(59.5, 0.95, "Women", adj = 0, col = "white", font = 2, cex = 1.2)
axis(side = 2, at = 0:10 / 10)
axis(side = 2, at = 1:99 / 100, labels = NA, tck = -0.01)
```

What is the 10-year risk of remission for men and women respectively?