Notes on statistics and computing with data using R

PhD course; Aalborg, Spring, 2019

Compilation: Tuesday 23rd April, 2019

Søren Højsgaard
Department of Mathematical Sciences
Aalborg University, Denmark
http://people.math.aau.dk/~sorenh/

Contents

Chapter 1

Introduction to R

After starting R we see a prompt (>) where commands are typed followed by hitting the enter button. R will evaluate the commands and print the result.

Anything that exists in R is an OBJECT (sometimes also called a VARIABLE); anything we do in R involves calling functions. A function take zero, one or more objects as input arguments and returns an object as output.

We create two objects (or variables), m and n, holding the values 10 and 1.56. This can be done in different ways:

```
m <- 10
n = 1.56
```

We say that "<-" and "=" are ASSIGNMENT operators. They can be used interchangably and the effect is to create variables m and n and store some values in them. The "<-"–assignment operator (mimicing an arrow) is perhaps the most intuitive: It reads: "Take whatever is on the right hand side and store in the object on the left hand side". Typing the names at the command prompt causes R to print the values:

```
m
## [1] 10
n
## [1] 1.56
```

Objects can be modified. For example

```
m <- m * 10 + 7
m
## [1] 107
n = n + m
n
## [1] 108.6
```

Symbol	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
<	Less than
>	Greater than
<=	Less or equal
>=	Greater or equal
==	Logical equal
** or ^	Exponentiation
%%	Remainder after division (modulo)
%/%	Integer part
exp(x)	Exponential function
log(x)	Natural logarithm
sqrt(x)	Square root
abs(x)	Absolute value
sin(x) and cos(x)	Sine and cosine functions

Table 1.1: Mathematical operators and functions.

We can have multiple computations/assignments on one line but then they must be separated by semi colon (";"). Anything after a hashmark (#) is regarded as comments in R. For example:

```
## multiple instructions on the same line are separated by ";"
sqrt.n <- sqrt( n ); sqrt.n    # Another comment
## [1] 10.42
```

In the code above the sqrt() function takes a single argument as input and computes the square root.

Another example of a function in R is q() which is used for exititing (or quitting) R. If we simply type q then we see how the function is defined (which is not of interest to us at this stage). To actually invoke the function we must do

```
q()
```

1.1 R as a calculator

R functions nicely as a simple calculator. Table 1.1 shows examples of simple mathematical operations and functions. Below is an example on how to do some of the calculations.

```
-3^2                ## power
## [1] -9
sqrt(15)            ## square root
## [1] 3.873
sin(2)              ## sine function
## [1] 0.9093
log(5)              ## natural log with base e
## [1] 1.609
log(5, base=10)     ## log with base 10 (alternative: log10(5)) )
## [1] 0.699
exp(5)              ## exponential function
## [1] 148.4
a <- 1/0; a         ## Infinity
## [1] Inf
b <- 0/0; b         ## Not a number
## [1] NaN
7 %/% 2             ## integer division
## [1] 3
7 %%  2             ## modulo; remainder
## [1] 1
pi                  ## pi
## [1] 3.142
1e-6                ## 10^(-6)
## [1] 1e-06
```

1.2 Vectors and indexing

R has several data structures and vectors is the most fundamental one. The variables m and n created previously are actually vectors of length 1. Additional data structures will be discussed in Chapter **??**. We shall create more interesting vectors below: The function c() will concatenate its input into a vector. For example we can register the production of oil in Norway (in mio. tons) for different years as two different vectors.

```
year <- c(1971, 1976, 1981, 1986, 1991, 1996, 2001, 2005)
year
## [1] 1971 1976 1981 1986 1991 1996 2001 2005
production <- c(.3, 9.3, 24.0, 42.5, 93.3, 156.8, 162.1, 138.1)
production
## [1]   0.3   9.3  24.0  42.5  93.3 156.8 162.1 138.1
```

Computations in R are VECTORIZED which has as a consequence that we can easily convert the production to tons or calculate the square root of all elements in prod as

```
production * 100
## [1]    30   930  2400  4250  9330 15680 16210 13810
sqrt(production)
## [1]  0.5477  3.0496  4.8990  6.5192  9.6592 12.5220 12.7318 11.7516
```

One main virtue of R is that it is so easy to work with data using a simple indexing mechanism. For example, on the square root scale, oil production grows approximately linearly until the end of the 20th century. Suppose that we want to create new vectors only with the data from the 20th century by extracting sub-vectors of `year` and `production`.

A very simple approach is to notice that the relevant data are in the first 6 entries in the two data vectors. We can extract these data by creating a vector with the entries we want and put this into square brackets:

```
entries <- 1:6 # same as c(1, 2, 3, 4, 5, 6) but much shorter.
entries
## [1] 1 2 3 4 5 6
production2 <- production[ entries ]
production2
## [1]   0.3   9.3  24.0  42.5  93.3 156.8
```

This works fine in this small example but pinpointing specific entries in a longer vector becomes tedious and error prone. We can therefore do the following:

```
b <- year < 2000
b
## [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
```

The statement `year < 2000` means that for each element in `year` it is checked whether the year is larger than 2000 or not. Hence this is another example of a vectorized computation. The result is a logical vector which is stored in the variable `b`.

To extract the year and production corresponding to the condition that `year` is smaller than 2000 we can do

```
year[b]
## [1] 1971 1976 1981 1986 1991 1996
production[b]
## [1]   0.3   9.3  24.0  42.5  93.3 156.8
```

So we have seen that we can index a vector by specifying entries or via a logical vector (of the same length as the vector we want to index). We notice that we can easily get from the latter to the former using `which()`.[1]

[1]FiXme Note: Need any, all, NA, is.na(); need also saplly and lapply; also class and is()/as()...

```
which(b)
## [1] 1 2 3 4 5 6
```

The steps above do not change `year` and `production`, they only extract sub vectors of these vectors. Let us create new variables containing these vectors:

```
year2 <- year[b]
production2 <- production[b]
```

To continue with the indexing, consider this:

```
production
## [1]   0.3   9.3  24.0  42.5  93.3 156.8 162.1 138.1
b <- production > 150; b
## [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE
```

The statement `production > 150` implies that for each element in `production` it is checked whether it is larger than 150 or not.

```
which( b )  ## where in 'production' are these values
## [1] 6 7
production[ b ]   ## what are these values
## [1] 156.8 162.1
year[ b ]   ## in which year was this production
## [1] 1996 2001
```

We can find the productions and years for the cases where the production is smaller than 150 using logical negation `!` (which turns `TRUE` into `FALSE` and vice versa):

```
production[ !b ]
## [1]   0.3   9.3  24.0  42.5  93.3 138.1
year[ !b ]
## [1] 1971 1976 1981 1986 1991 2005
```

We can find data for production over 50 before the turn of the 20th century by combining logical expressions:

```
b <- (production > 50) & (year < 2000); b ## '&' is logical AND
## [1] FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE
production[ b ]; year[ b ]
## [1]  93.3 156.8
## [1] 1991 1996
```

Similarly, we can find production data before 1980 and after 2000:

```
b <- (year < 1980) | (year > 2000); b ## '|' is logical OR
## [1]  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE
production[ b ]; year[ b ]
## [1]   0.3   9.3 162.1 138.1
## [1] 1971 1976 2001 2005
```

Suppose we discover that all years before the end of the 20th century is off by one year. This is easy to repair by replacing some elements of a vector

```
b <- year < 2000
b
## [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
year[b]     ## The wrong years
## [1] 1971 1976 1981 1986 1991 1996
year[b] + 1 ## The correct years
## [1] 1972 1977 1982 1987 1992 1997
## Replace the relevant entries of year by the correct values:
year[b] <- year[b] + 1; year
## [1] 1972 1977 1982 1987 1992 1997 2001 2005
```

1.3 Data frames

A data frame is a list of vectors all of the same length and can be viewed as the "spreadsheet" of `R`. Data frames are usually formed by importing data from a file into `R` (more about this in Chapter **??**), but a data frame can also be created from vectors using the `data.frame()` function; for example:

```
oil <- data.frame(yr=year, prod=production)
```

Hence, a dataframe provides a handle on many vectors at one time. The first rows of `oil` are displayed with

```
head( oil, 4 )
##     yr prod
## 1 1972  0.3
## 2 1977  9.3
## 3 1982 24.0
## 4 1987 42.5
```

The indexing mechanism applies to dataframes as well; but here we use two indices:

```
oil[2:4, c(2,1)]
##   prod   yr
## 2  9.3 1977
## 3 24.0 1982
## 4 42.5 1987
```

The first index refers to rows and the second to columns. If we do not write anything for the first index then all rows are selected and likewise for columns.

We can EXTRACT A COLUMN in a data frame in different ways: either by through the $ operator or by indexing the column number.

```
oil$prod # Get the 'prod' variable from the 'oil' data frame
## [1]   0.3   9.3  24.0  42.5  93.3 156.8 162.1 138.1
## oil[["prod"]] ## same
oil[,2]  # Get the 2nd column from the 'oil' data frame
## [1]   0.3   9.3  24.0  42.5  93.3 156.8 162.1 138.1
## oil[[2]] ## same
```

We can ADD A COLUMN and DELETE A COLUMN with

```
oil$extra <- 1:8
oil$extra <- NULL
```

1.4 Using add–on packages

We can use a scatter plot to visuaize the relationship between year and production. Much of R's versatility comes from the many add-on packages available from CRAN (the Comprehensive R Archive Network), see `www.r-project.org` (at the time of writing this, there are some 10.000 packages on CRAN). One package which we shall use extensively is the **ggplot2** package. This package does not come with the default installation of R, so it must be installed separately on the computer which can be done as

```
install.packages("ggplot2")
```

This installation must be done only once. To make the package available in an R session the package must be loaded with the `library()` function

```
library(ggplot2)
```

1.5 Plotting

Once the **ggplot2** package is loaded functions in the package can be used, e.g.

```
qplot(yr, prod, data=oil)
```

Sometimes one does not wish to load an entire package just to call one function once. In this case one can use :: as

```
ggplot2::qplot(yr, prod, data=oil)
```

1.6 Simple (linear) regression

Next we look at a simple dataset: Age and fat percentage in 9 adults:

```
age <- c(23, 28, 38, 44, 50, 53, 57, 59, 60)
fatpct <- c(19.2, 16.6, 32.5, 29.1, 32.8, 42, 32, 34.6, 40.5)
```

Data is shown as dots in Figure 1.1. By LINEAR REGRESSION we find the best approximating straight line, using the method of ORDINARY LEAST SQUARES (OLS). The function in R is `lm()` (short for linear model).

```
reg <- lm(fatpct ~ age)
summary(reg)
##
## Call:
## lm(formula = fatpct ~ age)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -5.115 -3.599 -0.521  1.759  7.053
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    6.225      5.711    1.09   0.3118
## age            0.542      0.120    4.51   0.0028 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.61 on 7 degrees of freedom
## Multiple R-squared:  0.744,Adjusted R-squared:  0.707
## F-statistic: 20.3 on 1 and 7 DF,  p-value: 0.00277
```

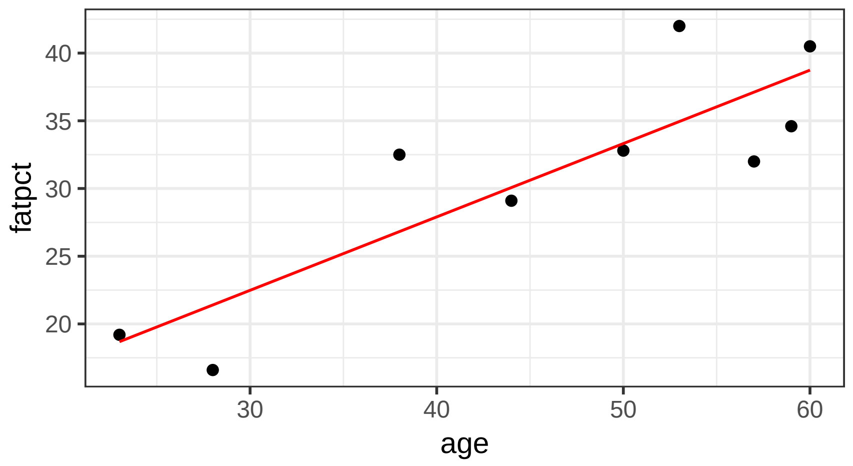


Figure 1.1:

The column `Estimate` shows the estimated regression coefficients. The interpretation is that when year increases by one year then the square root of the production increases by 0.54.

The plot in Figure 1.1 shows estimated regression line added on top of the data points. The plot is created as:

```
qplot(age, fatpct) +
    geom_line(aes(age, predict(reg)), color="red")
```

1.7 Getting help

Commands in `R` are really function calls. Example: `R` can be terminated by the function `q()`.

Help about a command is obtained with `help()` or `?`. Example: `rnorm()` simulates data from a normal distribution and help is obtained with

```
help( rnorm )
```

The possible arguments to a function are shown with

```
args( rnorm )
## function (n, mean = 0, sd = 1)
## NULL
```

The command `help( help )` will give you an overview over the help facilitites and `help.start()` will open a web–browser with the help facilities.

Most help pages have an examples-section at the end and these examples are often worthwhile to study for inspiration.

1.7.1 Getting help on a function that you know the name of

Use `?` or, equivalently, `help()`:

```
?mean
help(mean) # same
```

`args` shows you the arguments for a function.

```
args(mean)
## function (x, ...)
## NULL
args(mean.default)
## function (x, trim = 0, na.rm = FALSE, ...)
## NULL
args(read.csv)
## function (file, header = TRUE, sep = ",", quote = "\"", dec = ".",
##     fill = TRUE, comment.char = "", ...)
## NULL
```

For non-standard names use quotes or backquotes:

```
?`if`
?"if"       # same
help("if")  # same
```

There are also help pages for datasets, general topics and some packages:

```
?iris
?Syntax
?lubridate
```

Use the `example()` function to see examples of how to use it.

```
example(paste)
example(`for`)
```

The `demo()` function gives longer demonstrations of how to use a function.

```
demo()                           # all demos in loaded pkgs
demo(package = .packages(all.available = TRUE)) # all demos
demo(plotmath)
demo(graphics)
```

1.7.2 Finding a function that you do not know the name of

Use `??` or, equivalently, `help.search`:

```r
??regression
help.search("regression")
```

Again, non-standard names and phrases need to be quoted.

```r
??"logistic regression"
```

`apropos` finds functions and variables that match a regular expression.

```r
apropos("z$") # all fns ending with "z"
```

`RSiteSearch()` searches several sites directly from R. `findFn()` in **sos** wraps `RSiteSearch()` returning the results as a HTML table.

```r
RSiteSearch("logistic regression")
library(sos)
findFn("logistic regression")
```

`rseek.org` is an R search engine with a Firefox plugin.

1.7.3 Finding packages

`available.packages()` tells you all the packages that are available in the repositories that you set via `setRepositories()`.

`installed.packages()` tells you all the packages that you have installed in all the libraries specified in `.libPaths`. `library()` (without any arguments) is similar, returning the names and tag-line of installed packages.

```r
View(available.packages())
View(installed.packages())
library()
.libPaths()
```

Similarly, `data()` with no arguments tells you which datasets are available on your machine.

```r
data()
```

search() tells you which packages have been loaded.

```r
search()
```

packageDescription() shows you the contents of a package's DESCRIPTION file. Likewise news read the NEWS file.

```r
packageDescription("utils")
news(package = "ggplot2")
```

For finding which packages are loaded sessionInfo is quite nice.

```r
sessionInfo()
```

Help on a specific package is achieved with help():

```r
help(package="sos")
```

1.7.4 Getting help on variables

ls lists the variables in an environment.

```r
ls()                   # global environment
ls(all.names = TRUE) # including names beginning with '.'
ls("package:sp")      # everything for the sp package
```

Most variables can be inspected using str or summary:

```r
str(sleep)
summary(sleep)
```

ls.str is like a combination of ls and str.

```r
ls.str()
ls.str("package:grDevices")
lsf.str("package:grDevices")  # only functions
```

For large variables (particularly data frames), the head function is useful for displaying the first few rows.

```
head(sleep)
##   extra group ID
## 1   0.7     1  1
## 2  -1.6     1  2
## 3  -0.2     1  3
## 4  -1.2     1  4
## 5  -0.1     1  5
## 6   3.4     1  6
```

1.7.5 General learning about R

The Info page `https://stackoverflow.com/tags/r/info` is a very comprehensive set of links to free R resources.

Many topics in R are documented via vignettes, listed with `browseVignettes()`:

```
browseVignettes()
vignette("intro_sp", package = "sp")
```

By combining `vignette` with `edit`, you can get its code chunks in an editor.

```
edit(vignette("intro_sp",package="sp"))
```

1.8 Technicalities

1.8.1 On calling functions

Everything we do in `R` amounts to calling functions. For example addition of two numbers can be done as:

```
"+"(7, 9)
```

and typing `7 + 9` is just syntactic sugar on top of the function call above.

Consider this example: Add 10 to the value of elements number 1, 2 and 4 in the vector `v` below:

```
v <- c(7, 9, 0, 4, 8)
v[c(1, 2, 4)] <- v[c(1, 2, 4)] + 10
v
## [1] 17 19  0 14  8
```

The “square brackets” above have two meanings, so to speak and that can cause confusion. Extracting elements is done with the “square bracket extractor function” while replacement is done with the “square bracket assignment function”:

```
v <- c(7, 9, 0, 4, 8)
z <- v[c(1, 2, 4)]
v[c(1, 2, 4)] <- z + 10
v
## [1] 17 19  0 14  8
```

Under the hood, there is a call to the “square bracket extractor function” `"["` and the “square bracket assignment function” `"[<"`:

```
v <- c(7, 9, 0, 4, 8)
z <- "["(v, c(1, 2, 4))
v <- "[<-"(v, c(1, 2, 4), z + 10)
v
## [1] 17 19  0 14  8
```

Therefore, the calls below are equivalent

```
v <- c(7, 9, 0, 4, 8)
v[c(1, 2, 4)] <- v[c(1, 2, 4)] + 10
v
## [1] 17 19  0 14  8
v <- c(7, 9, 0, 4, 8)
v <- "[<-"(v, c(1, 2, 4), "["(v, c(1, 2, 4)) + 10)
v
## [1] 17 19  0 14  8
```

1.8.2 Vectors, lists, and data frames

Here we give some more information about data structures in `R`.

1.8.3 Vectors

The basic data structure in R is a vector. Think of a vector as a train wagon in which all passengers are of the same type.

Vectors can be created using the `c()` function (short for concatenate)

```
v1 <- c("here", "comes", "the", "sun") # Character vector
v2 <- c(7, 9, 13)                      # Numeric vector
v3 <- c(T, F, T)                       # Logical vector
v1; v2; v3
## [1] "here"  "comes" "the"   "sun"
## [1]  7  9 13
## [1]  TRUE FALSE  TRUE
```

Elements of vectors can be named (and used for indexing)

```
x <- c(a=123, b=234, c=345); x
##   a   b   c
## 123 234 345
x[c("a", "c")]
##   a   c
## 123 345
```

Elements are coerced to the least restrictive type:

```
x <- c(1, 2, 3, "hello world"); x
## [1] "1"           "2"           "3"           "hello world"
```

In this case the vector contains a character string and a string can represent both a number and a string so the full vector is coerced to a character vector.

1.8.4 Lists

If a vector is a train wagon in which all passengers have the same type, then a list is a train consisting of a sequence of train wagons. Wagons can be named.

```
x <- list(a=c(2, 3), b="hello world", c(T, F))
x
## $a
## [1] 2 3
##
## $b
## [1] "hello world"
##
## [[3]]
## [1]  TRUE FALSE
```

Indexing:

```
x[ c(1, 2) ] # The train consisting of wagons 1 and 2
## $a
## [1] 2 3
##
## $b
## [1] "hello world"
x[ c("a", "b") ]
## $a
## [1] 2 3
##
## $b
## [1] "hello world"
x[ 1 ]       # The train consisting of wagon 1 only
## $a
## [1] 2 3
x[ "a" ]
## $a
## [1] 2 3
x[[ 1 ]]     # Wagon 1
## [1] 2 3
x$a
## [1] 2 3
```

Remove and add wagons:

```
x$a <- NULL; x
## $b
## [1] "hello world"
##
## [[2]]
## [1]  TRUE FALSE
x$h <- 2:4; x    # Wagons are added at the end
## $b
## [1] "hello world"
##
## [[2]]
## [1]  TRUE FALSE
##
## $h
## [1] 2 3 4
x[5] <- 1000; x  # Empty wagons are added too
## $b
## [1] "hello world"
##
## [[2]]
## [1]  TRUE FALSE
##
```

```
## $h
## [1] 2 3 4
##
## [[4]]
## NULL
##
## [[5]]
## [1] 1000
```

The train analogy does not carry through all the way: An element of a list can be a list itself (i.e. a wagon can be a train itself):

```
x <- list(a=c(2, 3), b="hello world",
          c=c(T, F), d=list(g=23, 45)); x
## $a
## [1] 2 3
##
## $b
## [1] "hello world"
##
## $c
## [1]  TRUE FALSE
##
## $d
## $d$g
## [1] 23
##
## $d[[2]]
## [1] 45
str( x )
## List of 4
##  $ a: num [1:2] 2 3
##  $ b: chr "hello world"
##  $ c: logi [1:2] TRUE FALSE
##  $ d:List of 2
##   ..$ g: num 23
##   ..$  : num 45
```

1.8.5 Dataframes

A dataframe is basically a list in which each element has the same length.

```
d <- data.frame(x=5:8, y=c("here", "comes", "the", "sun"),
                z=c(T, F, T, T))
d
##   x     y     z
## 1 5  here  TRUE
## 2 6 comes FALSE
## 3 7   the  TRUE
## 4 8   sun  TRUE
```

Operate on dataframes as on lists. In addition, subscript as for matrices:

```
d[c(1, 4), c(1, 3)]
##   x    z
## 1 5 TRUE
## 4 8 TRUE
```

1.8.6 Matrices

Matrices can be constructed with the matrix() function. For example, a 3×3 matrix with the integers from one to nine is defined as follows.

```
A <- matrix(1:9, nrow=3, ncol=3); A;
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
dim( A )
## [1] 3 3
```

The numbers are read column-wise; if this is not what you want, use the optional argument byrow=TRUE to matrix(). Indexing matrices is simlar to indexing vectors except that an index vector defining rows and an index vector defining columns are needed.

```
A[1:2, c(1,3)]  ## extract submatrix by some rows and cols
##      [,1] [,2]
## [1,]    1    7
## [2,]    2    8
A[1:2, ]        ## extract submatrix by some rows and and all cols
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
A[2, ]          ## no longer a matrix, but a vector
## [1] 2 5 8
```

```
A[2, , drop=FALSE]   ## still a matrix
##      [,1] [,2] [,3]
## [1,]    2    5    8
```

1.8.7 Iterating over rows and columns of a matrix

To calculate e.g. the sum of each row we can use APPLY() or ROWSUMS()

```
rs <- apply(A, 1, sum) ## or rs <- rowSums(A)
rs
## [1] 12 15 18
```

Say we want to subtract from each column the mean of that column. We can use e.g. SWEEP

```
cm <- colMeans(A)
sweep(A, 2, cm, FUN="-")
##      [,1] [,2] [,3]
## [1,]   -1   -1   -1
## [2,]    0    0    0
## [3,]    1    1    1
```

1.8.8 Simulating random and systematic data

In addition to `c()`, vectors can be created in other ways, for example using `:` and `seq()` which both cretes sequences of numbers, and `rep()` which replicates a vector.

```
1:4    # Sequence of integer numbers from 1 to 4
## [1] 1 2 3 4
3:-3   # Sequence of integer numbers from 3 to -3
## [1]  3  2  1  0 -1 -2 -3
seq(1,3, by=0.5)   # Sequence from 1 to 3 with step size 0.5
## [1] 1.0 1.5 2.0 2.5 3.0
seq(1,3, length=4) # Sequence of length 4 starting at 1 and ending at 3
## [1] 1.000 1.667 2.333 3.000
rep(1:3, times=2)  # Replicate 1, 2, 3 twice
## [1] 1 2 3 1 2 3
rep(1:3, each=2)   # Replicate each of 1, 2 3 twice
## [1] 1 1 2 2 3 3
```

Random data can be generated as follows: Samples from 5 independent uniform random random variables on $[0, 10]$ are generated by `runif()`:

```
runif( n = 5, min = 0, max = 10 )
## [1] 5.038 0.522 9.856 0.773 3.733
```

Samples from 5 independent normal random random variables with mean 1 and standard deviation 2 are generated by `rnorm()`:

```
rnorm( n = 5, mean = 1, sd = 2 )
## [1] -1.2286  2.2959  2.3896  0.3233  0.9772
```

1.8.9 Getting data into functions

If we want to plot `production` against `year` we can do

```
plot(prod ~ yr, data=oil)
```

This works because there is a version of `plot()` that takes data as an argument; above `data=oil`. A function like `smooth.spline()`, however, does not take data as input:

```
args(smooth.spline)
## function (x, y = NULL, w = NULL, df, spar = NULL, lambda = NULL,
##     cv = FALSE, all.knots = FALSE, nknots = .nknots.smspl, keep.data = TRUE,
##     df.offset = 0, penalty = 1, control.spar = list(), tol = 1e-06 *
##         IQR(x), keep.stuff = FALSE)
## NULL
```

Hence we must provide data in another way. Can be done different ways:

- Using dollar sign

  ```
  sm <- smooth.spline(oil$yr, oil$prod, df=4)
  ```

- or - more elegantly - using `with()`

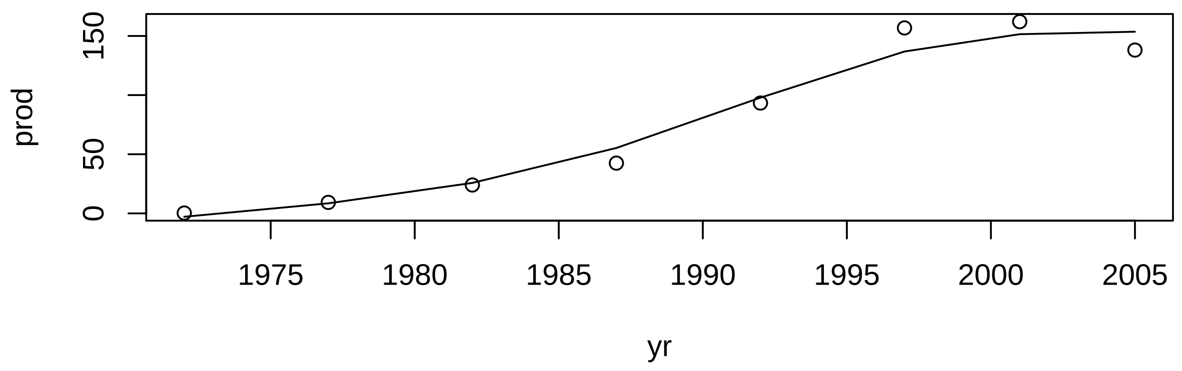
  ```
  sm <- with(oil, smooth.spline(yr, prod, df=4))
  ```

- or - but careful here - using ATTACH and DETACH

  ```
  attach(oil)
  sm <- smooth.spline(yr, prod, df=4)
  detach(oil)
  ```

The spline fit looks like:

```r
plot(prod ~ yr, data=oil)
lines(sm)
```



1.8.10 R as a programming language

In addition to the statstical and mathematical facilities, one virtue of R is that it is a programming language. While large scale programming with R is not the main focus of this book, we shall from time to time write small functions "on the fly" as we need them.

Example: Calculate the sum of the numbers $1, 2, 3, \ldots, N$. This sum can be computed as $N(N + 1)/2$, but if we did not know that we could simply do:

```r
N <- 100
result <- 0
for (i in 1:N)
    result <- result + i
cat("The sum of the numbers from 1 to", N, "is:", result, "\n")
## The sum of the numbers from 1 to 100 is: 5050
```

We can create a general purpose function that does the trick:

```r
add1toN <- function( N ){
    result <- 0
    for (i in 1:N)
        result <- result + i
    cat("The sum of the numbers from 1 to", N, "is:", result, "\n")
    result
}
add1toN( 1000 )
```

```r
## The sum of the numbers from 1 to 1000 is: 500500
## [1] 500500
add1toN( 10 )
## The sum of the numbers from 1 to 10 is: 55
## [1] 55
```

Chapter 2

Summarizing data

The following vectors are an excerpt from the `mtcars` dataset: Weight `wt` (in 1000 lbs) and fuel efficiency `mpg` (in miles per gallon).

```
x <- wt <- c(2.6, 2.9, 2.3, 3.2, 3.4, 3.5, 3.6, 3.2, 3.1, 3.4, 3.4, 4.1,
3.7, 3.8, 5.2, 5.4, 5.3, 2.2, 1.6, 1.8, 2.5, 3.5, 3.4, 3.8, 3.8,
1.9, 2.1, 1.5, 3.2, 2.8, 3.6, 2.8)

y <- mpg <- c(21, 21, 22.8, 21.4, 18.7, 18.1, 14.3, 24.4, 22.8, 19.2, 17.8,
16.4, 17.3, 15.2, 10.4, 10.4, 14.7, 32.4, 30.4, 33.9, 21.5, 15.5,
15.2, 13.3, 19.2, 27.3, 26, 30.4, 15.8, 19.7, 15, 21.4)
```

We shall look at measures of where is the "location" or "center" of the data and what is the "spread" of data.

2.1 Notation

We have n observations in the vector x. We denote these symbolically by

$$x_1, x_2, x_3, \dots, x_{n-1}, x_n$$

and they read “x one”, “x two” etc. For the sum $x_1 + x_2 + x_3 + \cdots + x_n$ we write

$$x_. = \Sigma_{i=1}^n x_i = x_1 + x_2 + x_3 + \cdots + x_n$$

and the left hand side reads “x dot” and $\Sigma_{i=1}^n x_i$ reads “the sum of x_i as i goes from 1 to n”. The symbol $x.$ is of course an arbitrary name for the sum.

2.2 Mesures of location

If we divide $x_.$ by the number of observations n we get the mean (or AVERAGE). Again, we can invent any name we like for this average, but it is quite common to write $\bar{x}$:

$$\bar{x} = \frac{1}{n} x_. = \frac{1}{n} \sum_{i=1}^{n} x_i = \frac{1}{n}(x_1 + x_2 + x_3 + \cdots + x_n)$$

The most commonly used MEASURE OF LOCATION is the SAMPLE MEAN (as opposed to a theoretical quantity defined later), (or EMPIRICAL MEAN or AVERAGE):

```
sum(x) / length(x)
## [1] 3.206
mean(x)
## [1] 3.206
```

The MEDIAN is a related measure: We sort the data:

```
x2 <- sort(x)
x2
##  [1] 1.5 1.6 1.8 1.9 2.1 2.2 2.3 2.5 2.6 2.8 2.8 2.9 3.1 3.2 3.2 3.2 3.4 3.4 3.4
## [20] 3.4 3.5 3.5 3.6 3.6 3.7 3.8 3.8 3.8 4.1 5.2 5.3 5.4
```

If the number of data points is odd, the median is the middle data point. If the number is even, the median is the average of the two points in the middle:

```
median(x)
## [1] 3.3
```

2.3 Measures of spread

The most common MEASURE OF SPREAD is the SAMPLE STANDARD DEVIATION (as oppposted to a theoretical quantity discussed later) (or EMPIRICAL STANDARD DEVIATION).

The squared distance of x_i to $\bar{x}$ is $(x_i - \bar{x})^2$. The average squared distance $\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$ is a measure of spread of data. For technical reasons we divide by $n - 1$ rather than n and obtain the SAMPLE VARIANCE

$$s_x^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2$$

If the units of the x_is are, say meters, then the unit of $\bar{x}$ is also meters but the unit of s_x^2 is "square meter". This leads to the <u>SAMPLE STANDARD DEVIATION</u> which also has unit meter:

$$s_x = \sqrt{s_x}$$

```
var( x )
## [1] 0.9496
sd( x )
## [1] 0.9745
```

There are alternative measures of spread. One is the interquartile tange (IQR). Consider this

```
quant <- quantile(x); quant
##    0%   25%   50%   75%  100% 
## 1.500 2.575 3.300 3.625 5.400
```

The 50% quantile is the median. The 75% quantile is the median of the highs and the 25% quantile is the median of the lows. The difference between these two quantiles is the IQR:

```
quant[4] - quant[2]
##  75% 
## 1.05
```

2.4 A practical interpretation and an empirical rule

A practical interpretation of the mean $\bar{x}$ and and standard deviation s_x is that for nearly symmetrical mound shaped datasets, about 68% of data is within one standard deviation of the mean and 95% is within two standard deviations of the mean.

Often we can get reasonable estimates of the sample mean and standard deviation by simply looking at data:

If data is not highly skewed then the mean and median are roughly the same. An estimate of the median is obtained by looking at the center of the (sorted) data vector:

```
x2 <- sort(x)
x2
##  [1] 1.5 1.6 1.8 1.9 2.1 2.2 2.3 2.5 2.6 2.8 2.8 2.9 3.1 3.2 3.2 3.2 3.4 3.4 3.4
## [20] 3.4 3.5 3.5 3.6 3.6 3.7 3.8 3.8 3.8 4.1 5.2 5.3 5.4
```

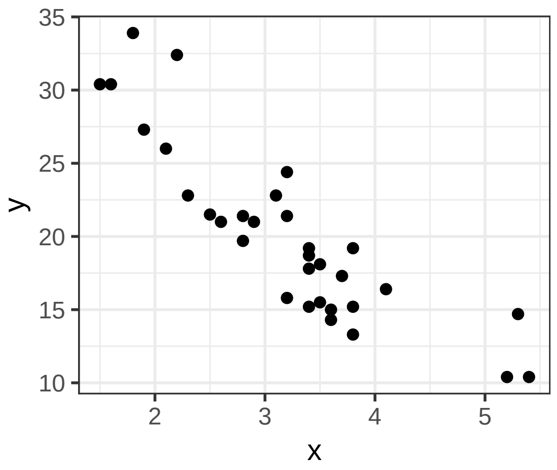


Figure 2.1: Correlated measurements.

```
median( x )
## [1] 3.3
mean( x )
## [1] 3.206
```

About 95% of the observations will fall in the interval $\bar{x} \pm 2s_x$.

If we say that 95% of the observations are "practically all observations" then practically all observations fall in this interval

Hence the largest minus the smallest value is about 4 standard deviations and hence that a crude estimate of the standard deviation is

```
(max(x) - min(x)) / 4
## [1] 0.975
sd(x)
## [1] 0.9745
```

2.5 Covariance and correlation

The measurements in `x` and `y` "co–vary"

```
library(ggplot2)
qplot(x, y)
```

A measure of how two variables co–vary is the SAMPLE COVARIANCE or EMPIRICAL COVARIANCE between x and y

$$s_{xy} = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})$$

Notice: If we replace y_i by x_i and $\bar{y}$ by $\bar{x}$ above then we get the sample covariance between x and x which is the (sample) variance of x.

Closely related to the (sample) covariance is the (sample) CORRELATION COEFFICIENT:

$$r_{xy} = \frac{s_{xy}}{s_x s_y} = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n} (x_i - \bar{x})^2 \sum_{i=1}^{n} (y_i - \bar{y})^2}}$$

The correlation is always in the interval ± 1. If the correlation is 1 or -1 there is a perfect linear association between x and y. If the correlation is 0 there is no linear association at all.

```
cov( x, y )
## [1] -5.126
cor( x, y )
## [1] -0.8728
```

2.6 The measurement unit matters – sometimes

The weight x is in 1000 pounds (lbs). One pound is about 0.45 kg, so to get the weight in kg we must multiply by 1000×0.45, so the weight in metric tonnes is $.45$ times the weight.

Now consider weight data on the new and old scale:

```
x.metric <- .45 * x
mean(x)
## [1] 3.206
mean(x.metric)
## [1] 1.443
sd(x)
## [1] 0.9745
sd(x.metric)
## [1] 0.4385
var(x)
## [1] 0.9496
var(x.metric)
## [1] 0.1923
cov(x, y)
```

```
## [1] -5.126
cov(x.metric, y)
## [1] -2.307
cor(x, y)
## [1] -0.8728
cor(x.metric, y)
## [1] -0.8728
```

More generally: Take four numbers, a, b, c and d and create new variables u_i and v_i as

$$u_i = a + bx_i, \quad v_i = c + dy_i$$

This corresponds to changing the scale and location of the data; for example changing temperature measurements from Celcius to Fahrenheit.

Then for the new variables we have

$$\bar{u} = a + b\bar{x}, \quad s_u^2 = b^2 s_x^2, \quad s_u = bs_x, \quad s_{uv} = bds_{xy}$$

but

$$\rho_{uv} = \rho_{xy}$$

Hence: mean, variance, standard deviation and covariance depends on the scale on which the variables are measured but correlation does not.

2.7 Correlation and regression

Fig 2.1 suggests that y (`mpg`) is linearly related to x (`wt`). In LINEAR REGRESSION we model variation ib y as a function of variation in x by asumming a linear relation:

$$y \approx \beta_0 + \beta_1 x \tag{2.1}$$

The parameters β_0 (the intercept) and β_1 (the slope) are to be estimated from data. The most common method is the method of LEAST SQUARES: The estimates for β_0 and β_1 are those values that minimizes the RESIDUAL SUM OF SQUARES

$$\sum_{i=1}^{n} [y_i - (\beta_0 + \beta_1 x_i)]^2$$

The estimated slope is closely related to the correlation coefficient:

$$\hat{\beta}_1 = r_{xy}\frac{s_y}{s_x} = \frac{s_{xy}}{s_x^2} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

The estimated intercept is

$$\hat{\beta} = \bar{y} - \hat{\beta}_1 \bar{x}$$

```
beta_1 <- cov(x, y) / var(x)
beta_1
## [1] -5.398
beta_0 <- mean(y) - beta_1 * mean(x)
beta_0
## [1] 37.4
```

The LINEAR REGRESSION model in (2.1) and many other models with a similar structure can be handled with the R function `lm()`:

```
lm(y ~ x)
##
## Call:
## lm(formula = y ~ x)
##
## Coefficients:
## (Intercept)            x
##        37.4         -5.4
```

The FITTED values from the regression model are the points on the estimated line:

```
y.hat <- beta_0 + beta_1 * x
y.hat
##  [1] 23.363 21.744 24.983 20.124 19.045 18.505 17.965 20.124 20.664 19.045
## [11] 19.045 15.266 17.425 16.885  9.328  8.248  8.788 25.523 28.762 27.682
## [21] 23.903 18.505 19.045 16.885 16.885 27.142 26.062 29.301 20.124 22.284
## [31] 17.965 22.284
```

We can plot data and fitted values as:

```
qplot(x, y) + geom_line(aes(x, y.hat))
```

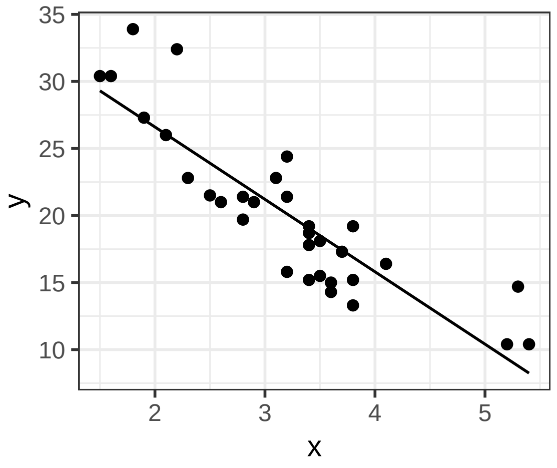


Figure 2.2: Correlation and regression are related topics.

Chapter 3

Linear models

Summary: Linear models are a super versatile tool and includes several classical analysis techiques such as linear regression, multiple regression, analysis-of-variance, and t-test as special cases.

Linear models is the most commonly used class of statistical models. Linear models focus on relating a QUANTITATIVE RESPONSE variable to one or more EXPLANATORY VARIABLES. Many specific types of models that are known under other names in the litterature are really linear models. Examples of this include LINEAR REGRESSION, MULTIPLE REGRESSION and POLYNOMIAL REGRESSION (Sec. 3.1.1), ANALYSIS OF VARIANCE or ANOVA, (Sec. 3.1.3) and ANALYSIS OF COVARIANCE or ANCOVA (Sec. 3.1.4).

The main difference between these models pertain to the nature of the explanatory variables, namely whether they are purely QUANTIATIVE (i.e. numerical), purely QUALITATIVE (i.e. categorical) or a combination of these. In the litterature, a LINEAR MODEL is also known as a GENERAL LINEAR MODEL.

3.1 What is a linear model?

Example 3.1.1 *[Potatoes]* This dataset contains `weight` and two sizes of 20 potatoes. Weight in grams; size in milimeter. There are two sizes: `length` is the longest length and `width` is the shortest length across a potato.

```
potatoes <- read.table("./data/potatoes.txt", header = TRUE)
head(potatoes, 4)
##   weight length width
## 1     22     38    29
## 2     41     46    34
## 3     24     40    31
## 4     16     33    28
```

Table 3.1: Some important Rfunctions for linear models introduced in this chapter.

Function	Purpose
`lm(y ~ x, data = ...)`	Fit a linear model (y is a quantitative response).
`summary(fit)`	Get parameter estimates and standard errors as well as e.g. R^2 value.
`predict(fit)`	Get the estimated response.
`confint(fit)`	Get the confidence interval for the parameters.
`residuals(fit)`	Raw residuals.

`residuals(fit, type = "pearson")`
`anova(fit)`
`drop1(fit, test = "F")`
`anova(fit1, fit2)`
	Standardised residuals.
	ANOVA. Test?
	Test dropping factors .
	ANOVA. Test?

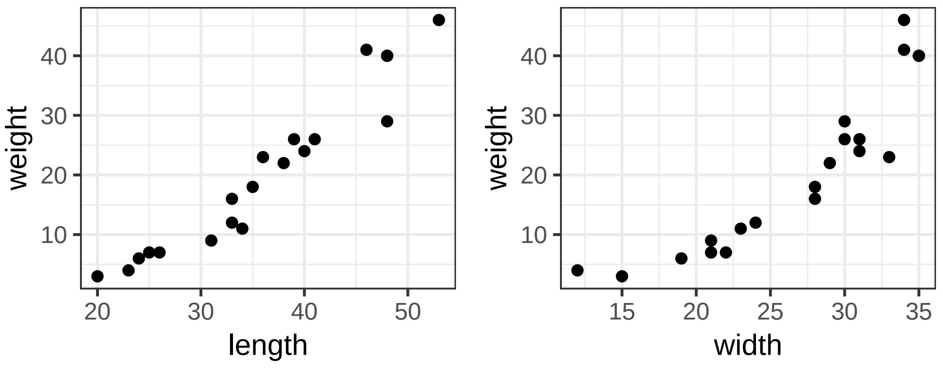


Figure 3.1: Plot of weight against length and against width of potatoes.

We are interested in modelling how `weight` changes when `length` and `width` changes. We say that `weight` is the <u>RESPONSE VARIABLE</u> while `length` and `width` are <u>EXPLANATORY VARIABLES</u>. A commonly used name for this setting is <u>MULTIPLE REGRESSION</u>. If only one variable is included as explanatory, then the the setting is called <u>LINEAR REGRESSION</u> (or <u>SIMPLE LINEAR REGRESSION</u>).

Initially, we plot weight against length and width to get an overview of the data, see Fig. 3.1. The relationship between weight and length seems to follow an approximately straight line. Likewise, it the relationship between weight and width appears to have a curvature, which can perhaps be captured by a parabola. However, in both plots it is clear that the relationships are not exact: Points scatter around a straight line and a parabola.

□ □

Linear models can be described informally and formally. In the following we take an informal approach and defer a more formal treatment to Chap. **??**, p. **??**. In a regression model, the aim is to model how variation in a <u>RESPONSE</u> y depends on variation in, say, three <u>PREDICTORS</u> or

EXPLANATORY VARIABLES or COVARIATES x_1, x_2, x_3. A LINEAR MODEL is a regression model where it is assumed that y is related linearly to the predictors as

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \varepsilon$$

where ε is an ERROR TERM. The terms $\beta_0, \dots, \beta_3$ are unknown PARAMETERS which are to be estimated from data. Ocasionally we shall refer to individual recordings by subscripting variables with an i, i.e.

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \varepsilon_i, \quad i = 1, \dots, n$$

Returning to the considerations Example 3.1.1, we may choose to consider the model

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_2^2 + \varepsilon$$

where y is `weight`, x_1 is `length` and x_2 is `width`. We shall think of $m(x_1, x_2) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_2^2$ as a function describing the SYSTEMATIC COMPONENT of the model whereas the error term ε refers to deviations between data and the systematic component.

A word about the term LINEAR in this context: Linear means that the parameters enter linearly, but the predictors need not do so (in the model above we had width and width–squared): In the setting above, suppose we have two sets of parameters $\beta_0, \dots, \beta_3$ and $\tilde\beta_0, \dots, \tilde\beta_3$. Then adding the corresponding systematic components $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_2^2$ and $\tilde\beta_0 + \tilde\beta_1 x_1 + \tilde\beta_2 x_2 + \tilde\beta_3 x_2^2$ gives $(\beta_0 + \tilde\beta_0) + (\beta_1 + \tilde\beta_1) x_1 + (\beta_2 + \tilde\beta_2) x_2 + (\beta_3 + \tilde\beta_3) x_2^2$ which has the same form as the systematic component we started out with. Thus $y = \beta_0 + \beta_1 x_1^{\beta_2} + \varepsilon$ is not a linear model (because β_0, β_1 and β_2 do enter linearly in the systematic component $m(x) = \beta_0 + \beta_1 x_1^{\beta_2}$).

3.1.1 Linear regression

First consider the simple case with one explanatory variable $x =$ `length` and the response variable is $y =$ `weight`. Mathematically, we write the relationship between y and x as

$$y = \beta_0 + \beta_1 x + \varepsilon. \tag{3.1}$$

The unknown parameters β_0 and β_1 (which we collect into a vector $\beta = (\beta_0, \beta_1)$) are estimated using the method of LEAST SQUARES, see Sec. 3.11.3, p. 79, using the `lm()` function. The expression `weight ~ length` is an example of a MODEL FORMULA, where `weight` depends on `length`. See Sec. 3.8, p. 64 for more about model formulae.

```
pot1_l <- lm(weight ~ length, data = potatoes)
tidy(pot1_l)  ## In the broom package
## # A tibble: 2 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)   -28.7     3.44       -8.35 1.33e- 7
## 2 length          1.37    0.0925     14.8  1.68e-11
```

Using R, we have now obtained, amongst other quantities, ESTIMATES of the unknown parameters β_0 and β_1. The estimate of β_0 (the intercept) is $\hat{\beta}_0 = -28.7$ and the estimate of β_1 (the slope) is $\hat{\beta}_1 = 1.4$. The intercept of -28.7 is interpreted as the expected weight of a potato with length 0 mm. Obviously, a negative value does not make sense. This is an example of so-called EXTRAPOLATION: We have only observed (collected) potatoes with length far away from 0, hence we cannot expect our model to predict well outside the data that was used to estimate the parameters. The interpretation of the slope is that when the length of a potato is increased by 1 mm, we expect its weight to be increased by 1.4 grams. We will now discuss this interpretation in more detail.

The model is illustrated in Fig. 3.2 (left). Above, $\beta_0 + \beta_1 x$ models the *expected* value of the outcome for an individual with an observed x-value, and ε represents random noise and/or measurement error. We can define $m(x) = \beta_0 + \beta_1 x$, and we can think of $m(x)$ as the SYSTEMATIC PART of the model, while ε denotes the RANDOM PART of the model. Note that $m(x)$ is a straight line with INTERCEPT β_0 and SLOPE β_1.

The interpretation of the parameters is as follows: If $x = 0$ then $m(x) = \beta_0$ so β_0 must be the average level of y for subjects with $x = 0$ (which also means that the unit of β_0 is grams). The interpretation of β_1 is that when x increases by one unit (milimeter in the potatoes example) from some value x_0 to $x_0 + 1$ then the average value of y changes by $m(x_0 + 1) - m(x_0) = \beta_1$ units (so the unit of β_1 in the potatoes example is grams per milimeter), and this change is the same for all initial values x_0.

Estimated parameters are often denoted with a hat as $\hat{\beta} = (\hat{\beta}_0, \hat{\beta}_1)$ to distinguish them from the true but unknown values. Plugging the estimated values into $\beta_1 + \beta_2 x_i$ gives the points on the estimated regression lines; these are also denoted FITTED VALUES or PREDICTED VALUES.

Based on the graphs in Fig. 3.1, p. 36, we discussed in Sec. 3.1 the model

$$y_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_2^2 + \varepsilon. \qquad (3.2)$$

where x_1 is `length` and x_2 is `width`. The model can be fitted to data with

```
pot1_lw2 <- lm(weight ~ length + width + I(width^2), data = potatoes)
tidy(pot1_lw2)
## # A tibble: 4 x 5
##   term        estimate std.error statistic   p.value
##   <chr>          <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept)   8.84      7.26        1.22 0.241
## 2 length        0.831     0.135       6.16 0.0000137
## 3 width        -2.62      0.570      -4.59 0.000299
## 4 I(width^2)    0.0681    0.0121      5.62 0.0000386
```

where the notation `I(width^2)` ensures that a quadratic effect is included in the model. The notation will be explained in more detail in Sec. 3.8, p. 64, and corresponds to the model

Interpretation of the parameters become more involved: The function $m(x_1, x_2) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_2^2$ models the *expected* value of the outcome (`weight`) for

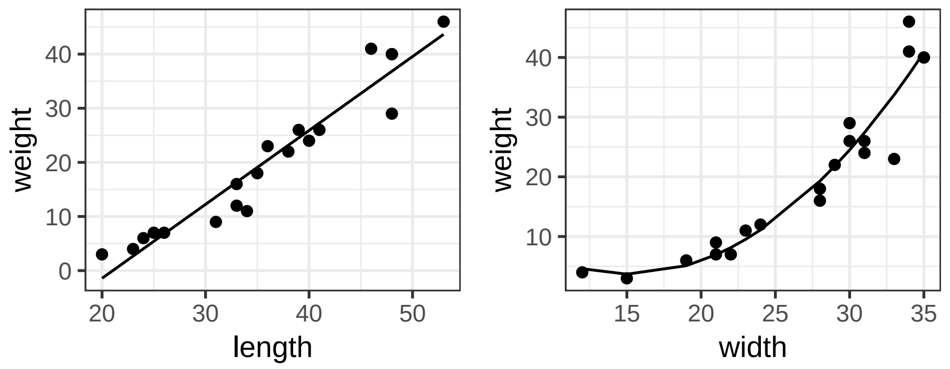


Figure 3.2: Left: `lm(weight ~ width, data = potatoes)` is illustrated. Right: `lm(weight ~ width + I(width^2), data = potatoes)` is illustrated.

an individual (a potato) with `length` $= x_1$ and `width` $= x_2$. First of all, β_0 is the expected weight of an infinitely small potato with zero length and zero width. The interpretation of β_1 is the effect on `weight` of changing `length` by one unit when `width` is fixed. The interpretation of β_2 and β_3 is more subtle: Increasing x_2 by one unit means a change in the expected value of `weight` by

$$m(x_1, x_2 + 1) - m(x_1, x_2) = \beta_2 + 2\beta_3 x_2 + \beta_3 = \beta_2 + \beta_3(2x_2 + 1).$$

This means that the change in the expected value of $x_2 =$ `width` to $x_2 + 1$ depends on the value of $x_2 =$ `width`. For example, if $x_2 =$ `width` $= 15$, then $m(x_1, x_2 + 1) - m(x_1, x_2) = \beta_2 + 31\beta_3$, and if $x_2 =$ `width` $= 30$, then $m(x_1, x_2 + 1) - m(x_1, x_2) = \beta_2 + 61\beta_3$.

There is an additional complicating fact: In practice, potatoes grow both in length and width. Therefore the interpretation of β_1 as the effect of increasing length by one unit while keeping width fixed is not directly applicable. We treat this topic in detail in Sec. 3.7, p. 64.

An simplification of the model (3.2) is to omit `length` as predictor. Notice also that the linear regression model in (3.1) can be seen as a simplification of (3.2) in which `width` and `width`-squared is omitted:

```
pot1_w2 <- lm(weight ~ width + I(width^2), data = potatoes)
tidy(pot1_w2)
## # A tibble: 3 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)  25.2      12.0         2.09 0.0521
## 2 width        -2.84      1.01       -2.80 0.0122
## 3 I(width^2)    0.0939    0.0203      4.63 0.000239
```

The model with just `width` as predictor and the model with `width` and `width`^{2} as predictors are illustrated in Fig. 3.2.

At this stage we shall make a very simple comparison of some of the models: We calculate the squared correlation between the observed and the fitted / predicted values. Calculating the correlation may seem very natural, and the reason for squaring these correlations will be discussed in Sec. 3.6.

```
##   pot1_l pot1_lw2  pot1_w2
##   0.9237   0.9771   0.9227
```

In all three models described above, the intercept (β_0) is estimated to be far away from zero. It is possible to force the intercept to be zero by including a "-1" in the model specification, i.e. we may write e.g. `lm(weight ~ -1 + length, data = potatoes)`.

3.1.2 Parameter estimates, standard errors etc

Consider again the model in (3.2)

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_2^2 + \varepsilon. \tag{3.3}$$

The model is fitted with

```
pot1_lw2 <- lm(weight ~ length + width + I(width^2), data = potatoes)
tidy(pot1_lw2)
## # A tibble: 4 x 5
##   term        estimate std.error statistic   p.value
##   <chr>          <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept)   8.84      7.26        1.22 0.241
## 2 length        0.831     0.135       6.16 0.0000137
## 3 width        -2.62      0.570      -4.59 0.000299
## 4 I(width^2)    0.0681    0.0121      5.62 0.0000386
```

The `Estimate` column

The parameter estimates (in the `Estimate` column) are the values of β_0 and β_1 that minimizes

$$\sum_{i=1}^{n} [y_i - (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3}^2]^2$$

The `std.error` column

The column `std.error` contains a measure of how much the estimates would vary if the study was to be redone again under identical conditions. It is not possible to do these replications: 2015 is long gone, but we can mimic the situation. Consider the original data:

```
head(potatoes, 6)
##   weight length width
## 1     22     38    29
## 2     41     46    34
## 3     24     40    31
## 4     16     33    28
## 5      7     25    21
## 6     40     48    35
```

In another (hypothetical) replication of the study it could have been that the first row (potato) would appear twice, the second row not at all, the third row could appear three times, row four and seven would not appear at all etc. Such a dataset can be generated by sampling the rows of the original dataset with replacement:

```
w <- sample(1:nrow(potatoes), replace=TRUE)
potatoes2 <- potatoes[w,]
head(potatoes2, 6)
##      weight length width
## 3        24     40    31
## 13       46     53    34
## 13.1     46     53    34
## 13.2     46     53    34
## 18       12     33    24
## 13.3     46     53    34
```

In this new dataset, row 13 from the original data set appears four times, so some rows from the original dataset do not appear at all. Fitting the regression model to this new dataset gives:

```
m2 <- lm(weight ~ length + width + I(width^2), data=potatoes2)
coef(m2)
## (Intercept)      length       width  I(width^2)
##    18.76106     1.14083    -3.64775     0.07742
```

Now we repeat this scheme many times

```
N <- 9999
coefmat <- matrix(0, nrow=N, ncol=4)
for (i in 1:N){
    potatoes2 <- potatoes[sample(1:nrow(potatoes), replace=TRUE),]
    m2 <- lm(weight ~ length + width + I(width^2), data=potatoes2)
    coefmat[i, ] <- coef(m2)
}
head(coefmat, 4)
##        [,1]   [,2]   [,3]    [,4]
## [1,] 18.76 1.1408 -3.648 0.07742
## [2,] 21.96 0.7606 -3.351 0.08023
## [3,] 24.62 0.9620 -4.064 0.09242
## [4,] 24.89 0.8711 -3.995 0.09438
```

Now calculate the standard errors of each of the set of parameter estimates

```
apply(coefmat, 2, sd)
## [1] 12.26527  0.19138  0.93802  0.01825
```

Hence the standard errors of the estimates across (pseudo) replications of the study are in the same ball park as the content of the `std.error` column:

```
tidy(pot1_lw2)
## # A tibble: 4 x 5
##   term        estimate std.error statistic   p.value
##   <chr>          <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept)   8.84      7.26        1.22 0.241
## 2 length        0.831     0.135       6.16 0.0000137
## 3 width        -2.62      0.570      -4.59 0.000299
## 4 I(width^2)    0.0681    0.0121      5.62 0.0000386
```

The `statistic` column

The `statistic` is simply the estimate divided by the standard error of the estimate. This quantity says how many standard errors the estimate is from zero. To illustrate this, suppose we change scale of the measurements: `weight` is measured in hundred grams instead of grams and length in meters instead of milimeters:

```
potatoes3 <- potatoes %>% mutate(weight=weight / 100, length = length / 1000)
m3 <- lm(weight ~ length + width + I(width^2), data=potatoes3)
tidy(m3)
## # A tibble: 4 x 5
##   term         estimate std.error statistic   p.value
##   <chr>           <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept)  0.0884    0.0726        1.22 0.241
## 2 length       8.31      1.35          6.16 0.0000137
## 3 width       -0.0262    0.00570      -4.59 0.000299
## 4 I(width^2)   0.000681  0.000121      5.62 0.0000386
```

The estimates and their standard errors change but the statistic remains unchanged: The statistic still measures how many standard errors the estimate is from zero – and this quantity does not depend on the scale of measurement.

The `p.value` column

Often we are interested in testing the HYPOTHESIS that a specific parameter is zero. For example, if β_1 is zero in the model, then this corresponds to that `length` has no effect in explaining the variation in `weight` (that is, when `width` and `width`–squared is in the model). The statistic can be positive or negative. However, what is important is the numerical value of the statistic, so we shall just assume the statistic is positive.

A large value of the estimate causes us to doubt this hypothesis, i.e. a large value provides EVIDENCE against the hypothesis. On the other hand, a value of the sestimate close to zero supports the hypothesis. But as shown above, the estimate can be made as small (or large) as we want them; all we have to do is change the scaling on which `length` is recorded. The statistic column, on the other hand, does not depend on the scaling; the statistic column shows how many standard deviations the parameter estimates are away from zero. A large value of the statistic makes us doubt that the corresponding parameter is zero. This poses the question: When is the value of a statistic large? An answer to this is as follows: Compute the probabilty of observing a value of the statistic larger or equal to the observed value of the statistic. If the statistic is large, then this probability is small. So a small probability can be seen as evidence against the hypotesis. This probability is called a p–value and is reported in the `p.value` column. For historical reasons there is a tradition of comparing the p–value with preset fixed values, such as 0.01 or 0.05: If the p–value is smaller than 0.05 one REJECTS the hypothesis that the parameter is zero.

3.1.3 Analysis of variance (ANOVA) models

Example 3.1.2 *[ToothGrowth] The response is the length of odontoblasts (cells responsible for tooth growth) in 60 guinea pigs. Each animal received one of three dose levels of vitamin C (0.5, 1, and 2 mg/day) by one of two delivery methods: orange juice (coded as OJ) or ascorbic acid a form of vitamin C and (coded as VC).*

```
head(ToothGrowth, 4)
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
```

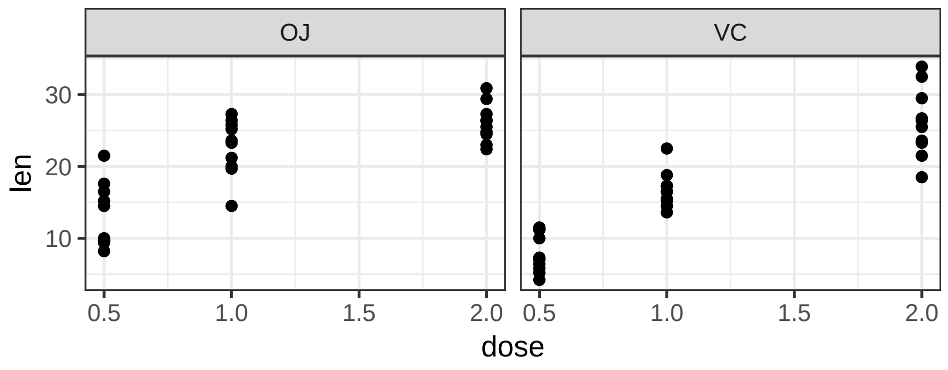
We can treat `dose` *as a numerical variable or we can think of* `dose` *as a factor with three levels. To accomodate both we add an extra column to data:*

```
ToothGrowth <- ToothGrowth %>%
    mutate(dosef = factor(dose,
                          levels = c(0.5, 1, 2),
                          labels = c("LO", "ME", "HI")))
```

```
head(ToothGrowth, 4)
##    len supp dose dosef
## 1  4.2   VC  0.5    LO
## 2 11.5   VC  0.5    LO
## 3  7.3   VC  0.5    LO
## 4  5.8   VC  0.5    LO
```

Such grouped data can be visualized in different ways:

```
ggplot(ToothGrowth, aes(dose, len)) + geom_point() + facet_grid(~ supp)
```



The average length in each of the $2 \times 3 = 6$ *groups is:*

```
tooth_avglen <- ToothGrowth %>%
  group_by(dose, supp) %>%
  summarise(val = mean(len))
tooth_avglen
## # A tibble: 6 x 3
## # Groups:   dose [3]
##    dose supp    val
##   <dbl> <fct> <dbl>
## 1   0.5 OJ    13.2
## 2   0.5 VC     7.98
## 3   1   OJ    22.7
## 4   1   VC    16.8
## 5   2   OJ    26.1
## 6   2   VC    26.1
```

An INTERACTION PLOT *is a graphical visualization of the group averages, see Fig. 3.3, where also variability in each group is displayed:*

```
ggplot(ToothGrowth, aes(x = factor(dose), y = len, colour = supp)) +
    geom_boxplot(outlier.shape = 4) +
    geom_point(data = tooth_avglen, aes(y = val)) +
    geom_line(data = tooth_avglen, aes(y = val, group = supp))
```

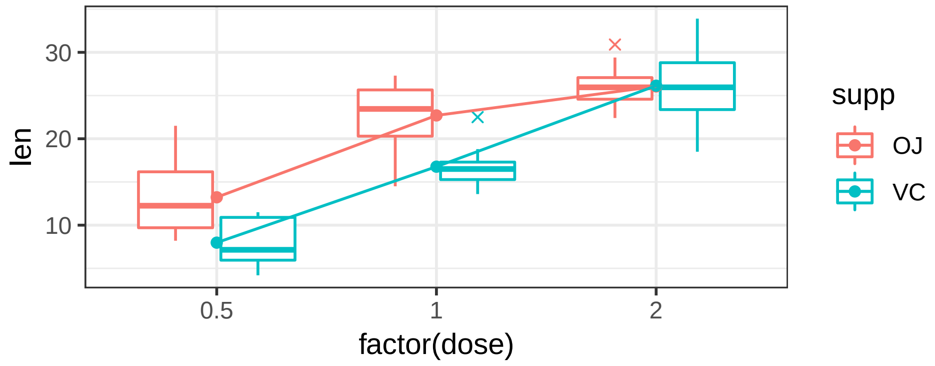


Figure 3.3: Interaction plot for the ToothGrowth data. The average length for each group is a dot. Boxplot outliers are crosses.

The plot shows us that the length increases with dose and that the sloop might be different for the two supplements: They start out at different lengths for dose 0.5 but end around the same length for dose 2.0.

□ □

We can model these data using a REGRESSION MODEL

$$y = \beta_0 + \beta_1 x_1 + \varepsilon,$$

where the predictor variable, x_{i1}, is a DUMMY VARIABLE: Create a vector x_1 which is 0 in entries with `supp = OJ` and 1 in entries with `supp = VC`. So the effect of the dummy variable is to "switch" the effect of β_1 on and off.

```
tg <- ToothGrowth %>% filter(dose == 0.5) ## data where dose is 0.5
lm(len ~ supp, data = tg)
##
## Call:
## lm(formula = len ~ supp, data = tg)
##
## Coefficients:
## (Intercept)       suppVC
##       13.23        -5.25
```

Since `supp` is coded as a FACTOR with two levels in the data frame, `lm()` will know that `supp` must be turned into a dummy variable (had `supp` had tree levels, then `lm()` would have created two dummy variables).

The parameters in the model have a very simple interpretation: The intercept is simply the average value of `len` in the `OJ` group and `suppVC` is the *change* in average when moving from `OJ` to `VC`:

```
tg %>% group_by(supp) %>% summarise(mean(len))
## # A tibble: 2 x 2
##   supp  `mean(len)`
##   <fct>       <dbl>
## 1 OJ          13.2
## 2 VC           7.98
```

Next consider the whole dataset. We can treat dose as a numerical variable or we can think of dose as a factor with three levels. We shall do the latter here:

```
ToothGrowth <- ToothGrowth %>%
    mutate(dosef = factor(dose,
                          levels = c(0.5, 1, 2),
                          labels = c("LO", "ME", "HI")))
```

Because dosef is coded as a factor then the call

```
tooth2 <- lm(len ~ supp + dosef, data = ToothGrowth)
coef(tooth2)
## (Intercept)      suppVC     dosefME     dosefHI
##       12.45       -3.70        9.13       15.50
```

corresponds to the model

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \varepsilon,$$

where x_2 is 1 when dosef = ME and 0 otherwise and x_3 is 1 when dose = HI and 0 otherwise. The interpretation is straight forward: Think of a dummy variable as a way of switching a parameter on and off. The mean effect, $m(x_1, x_2, x_3)$, becomes

$$m(x_1, x_2, x_3) = \begin{cases} \beta_0 & \text{if dose = LO and supp = OJ} \\ \beta_0 + \beta_1 & \text{if dose = LO and supp = VC} \\ \beta_0 + \beta_2 & \text{if dose = ME and supp = OJ} \\ \beta_0 + \beta_1 + \beta_2 & \text{if dose = ME and supp = VC} \\ \beta_0 + \beta_3 & \text{if dose = HI and supp = OJ} \\ \beta_0 + \beta_1 + \beta_3 & \text{if dose = HI and supp = VC} \end{cases}$$

The effect of the parameters on the mean value in each group can also be illustrated as in Tab. 3.2.

Table 3.2: The effect of the parameters on the mean value in each group.

	LO	ME	HI
OJ	β_0	$\beta_0 + \beta_2$	$\beta_0 + \beta_3$
VC	$\beta_0 + \beta_1$	$\beta_0 + \beta_1 + \beta_2$	$\beta_0 + \beta_1 + \beta_3$

Hence β_1 is the effect of going from `OJ` to `VC` and that effect is the same for any value of `dosef`. Likewise, β_2 is the effect of going from `LO` to `ME` and that effect is the same for any value of `supp`.

Because of this structure where the parameter values are added together, we say we have an <u>ADDITIVE MODEL</u>. Parallel profiles in Fig. 3.3 correspond to additivity of the effect of the factors. As seen, the lines are not parallel for the high dose, hence the additive model may not be suitable.

We can add the mean values obtained from the additive model, `tooth2`, as follows:

```
ToothGrowth %>%
  mutate(pred2 = fitted(tooth2)) %>%
  group_by(dosef, supp) %>%
  summarise(mean(len), mean(pred2), mean(len - pred2))
## # A tibble: 6 x 5
## # Groups:   dosef [3]
##   dosef supp  `mean(len)` `mean(pred2)` `mean(len - pred2)`
##   <fct> <fct>       <dbl>         <dbl>               <dbl>
## 1 LO    OJ          13.2          12.5                0.775
## 2 LO    VC           7.98          8.76              -0.775
## 3 ME    OJ          22.7          21.6                1.11
## 4 ME    VC          16.8          17.9               -1.11
## 5 HI    OJ          26.1          28.0               -1.89
## 6 HI    VC          26.1          24.2                1.89
```

An alternative to the additive model is then an <u>INTERACTION MODEL</u>

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_1 x_2 + \beta_5 x_1 x_3 + \varepsilon$$

where $x_1 x_2$ is 1 when `supp = VC` and `dosef = ME`, and $x_1 x_3$ is 1 when `supp = VC` and `dosef = HI`.

The model parameters are estimated in `R`as follows:

```
tooth3 <- lm(len ~ supp + dosef + supp * dosef, data = ToothGrowth)
coef(tooth3)
##    (Intercept)         suppVC        dosefME        dosefHI suppVC:dosefME
##          13.23          -5.25           9.47          12.83          -0.68
## suppVC:dosefHI
##           5.33
ToothGrowth <- ToothGrowth %>% mutate(pred3 = fitted(tooth3))
ToothGrowth %>%
  group_by(dosef, supp) %>%
  summarise(mean(len), mean(pred3))
## # A tibble: 6 x 4
## # Groups:   dosef [3]
##   dosef supp  `mean(len)` `mean(pred3)`
##   <fct> <fct>       <dbl>         <dbl>
## 1 LO    OJ          13.2          13.2
## 2 LO    VC           7.98          7.98
## 3 ME    OJ          22.7          22.7
## 4 ME    VC          16.8          16.8
## 5 HI    OJ          26.1          26.1
## 6 HI    VC          26.1          26.1
```

We see that the fitted values under the interaction model is the same as the group averages, and as such the interaction model represents no simplification.

3.1.4 Analysis of covariance (ANCOVA) models

The purpose of ANCOVA is (usually) to compare two or more linear regression lines. It is a way of comparing the y variable among groups while statistically controlling for variation in y caused by variation in the x variable.

We could have chosen to think of `dose` as a numeric variable in the `ToothGrowth` data so that we would have had one factor and one numerical variable, but since there are only three different doses it would be a little speculative.

Example 3.1.3 *[cricks] Walker (1962)***??** *studied the mating songs of male tree crickets. Each wingstroke by a cricket produces a pulse of song, and females may use the number of pulses per second to identify males of the correct species. Walker (1962) wanted to know whether the chirps of the crickets* Oecanthus exclamationis *and* Oecanthus niveus *had different pulse rates. See* *http://www.biostathandbook.com/ancova.html* *for details. He measured the pulse rate of the crickets (variable* `pps`*) at a variety of temperatures (*`temp`*):*

```
crick <- read.table("data/cricket.txt", header = TRUE)
summary(crick)
##  species       temp            pps       
##  ex :14   Min.   :17.2   Min.   : 44.3  
##  niv:16   1st Qu.:20.8   1st Qu.: 59.2  
##           Median :24.0   Median : 76.2  
##           Mean   :23.6   Mean   : 72.5  
##           3rd Qu.:26.2   3rd Qu.: 84.5  
##           Max.   :30.4   Max.   :101.7  
head(crick, 4)
##   species temp  pps
## 1      ex 20.8 67.9
## 2      ex 20.8 65.1
## 3      ex 24.0 77.3
## 4      ex 24.0 78.7
```

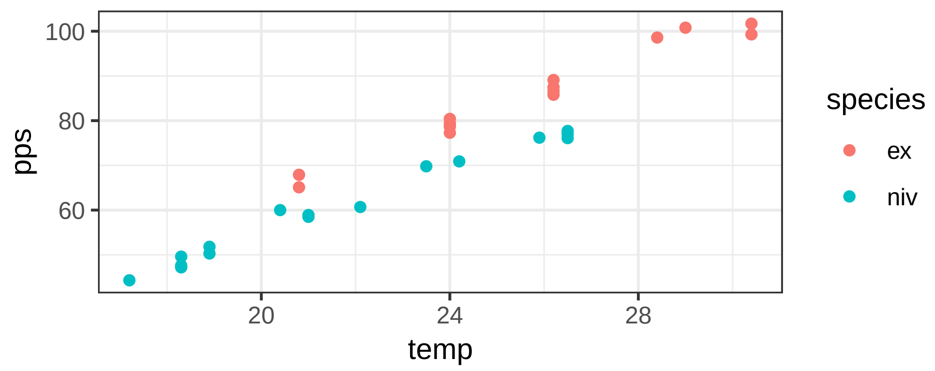


Figure 3.4: Sound of cricke: Pulse per second plotted against temperature for two species of cricks.

```r
ggplot(crick, aes(temp, pps, color = species)) + geom_point()
```

The main purpose here is to compare pps *(more precisely the mean of* pps*) for the two species when we take into account that the measurements were taken at different temperatures. Consider this summary of data:*

```r
crick %>%
  group_by(species) %>%
  summarise(m_pps = mean(pps), m_temp = mean(temp))
## # A tibble: 2 x 3
##   species m_pps m_temp
##   <fct>   <dbl>  <dbl>
## 1 ex       85.6   25.8
## 2 niv      61.0   21.7
```

□ □

A formal comparison (where we ignore temperature) can be made with a ONE–WAY ANOVA. Introduce a dummy variable x_1 with $x_1 = 1$ if `species = niv` and 0 when `species = ex`. The model and estimated parameters are then:

$$y = \beta_0 + \beta_1 x_1 + \varepsilon$$

```r
crm1 <- lm(pps ~ species, data = crick)
tidy(crm1)
## # A tibble: 2 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)     85.6      3.17     27.0  1.36e-21
## 2 speciesniv     -24.5      4.34     -5.66 4.64e- 6
```

The apparent large difference in group means of about 25 pps is, however, an artifact since the two species were largely measured at different temperatures, and because pulse rate is highly dependent on temperature as shown in Fig. 3.4.

1. This CONFOUNDING variable means that we should worry that any difference in mean pulse rate was caused by a difference in the temperatures at which you measured pulse rate, as the average temperature for the *O. exclamationis* measurements was 3.6 °C higher than for *O. niveus*.
2. We should also worry that *O. exclamationis* might have a higher rate than *O. niveus* at some temperatures but not others.

We can CONTROL FOR or ADJUST FOR temperature with ANCOVA as follows: Fit two parallel regression lines: Introduce x_2 as the temperature:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon$$

We can think of this as an ADDITIVE MODEL: The effect of changing temperature is the same for each species.

```
crm2 <- lm(pps ~ species + temp, data = crick)
tidy(crm2)
## # A tibble: 3 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)    -7.85     2.76      -2.85 8.36e- 3
## 2 speciesniv     -9.90     0.786    -12.6  8.19e-13
## 3 temp            3.63     0.106     34.4  7.91e-24
```

Hence after controlling for the effect of temperature the difference in mean pps is about 10 units (compared to 25 when not controlling for temperature) which is also what the graphics suggests.

```
ggplot(crick, aes(temp, pps, color = species)) +
  geom_point() +
  geom_line(aes(temp, predict(crm2)))
```

The interpretation of the model is that the mean difference in pps is about 10 for any temperature. In this sense we can say that species and temperature have an ADDITIVE EFFECT.

It is evident that the two regression lines are parallel, but suppose that was not the case: We introduce a third dummy variable x_3 which is 0 when `species = ex` and which is equal to temperature when `species = niv`. Hence x_3 is simply the elementwise product of x_1 and x_2, i.e. $x_{i3} = x_{i1}x_{i2}$.

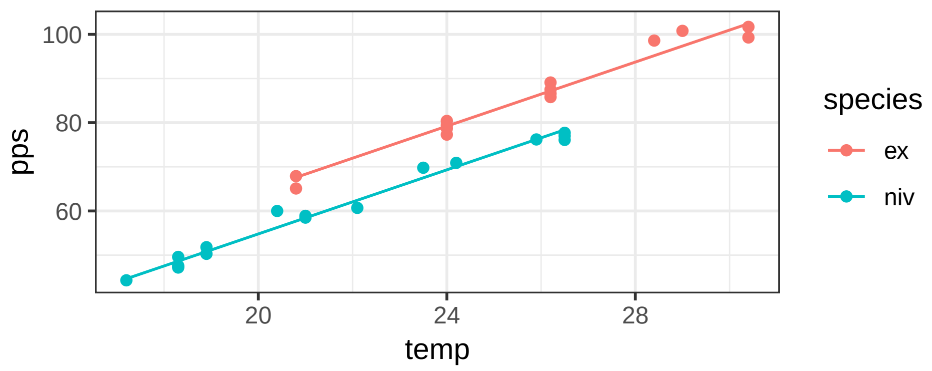


Figure 3.5: Two parallel regression lines fitted to cricks data.

While not really relevant in this example, we notice that we can also work with non–parallel regression lines:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 \underbrace{x_3}_{x_1 x_2} + \varepsilon \tag{3.4}$$

In this case, the effect of species depends on temperature. This can be seen by rearranging the model formula:

$$y = \beta_0 + (\beta_1 + \beta_3 x_2) x_1 + \beta_2 x_2 + \varepsilon \tag{3.5}$$

As seen, the effect of species is $\beta_1 + \beta_3 x_{i2}$. So in this case a comparison of the species must in general be made at specific temperatures (values of x_2), see Sec. 3.2. We fit the model with:

```
crm3 <- lm(pps ~ species + temp + species:temp, data = crick)
tidy(crm3)
## # A tibble: 4 x 5
##   term              estimate std.error statistic  p.value
##   <chr>                <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)        -11.0       4.22     -2.62  1.46e- 2
## 2 speciesniv          -4.77      5.20     -0.916 3.68e- 1
## 3 temp                 3.75      0.163    23.0   7.96e-19
## 4 speciesniv:temp     -0.213     0.214    -0.998 3.28e- 1
```

and notice that the parameter estimate for `speciesniv:temp` is very close to zero corresponding to that the two regression lines are close to being parallel (as the graphics suggests).

3.2 Prediction, estimates and contrasts, LSmeans

Suppose a linear model contains the regression parameters $\beta_1, \dots, \beta_p$. One is often interested in estimating a weighted sum of the model parameters

$$\theta = \lambda_1\beta_1 + \lambda_2\beta_2 + \cdots + \lambda_p\beta_p = \sum_{j=1}^{p} \lambda_j\beta_j$$

where $\lambda = (\lambda_1, \dots, \lambda_p)$ is a vector of known weights. We shall call such a weighted sum for a LINEAR PREDICTOR. After fitting the model the estimated linear predictor is

$$\hat\theta = \sum_{j=1}^{p} \lambda_j\hat\beta_j$$

Example: Calculate the estimated mean value on `supp = VC` and `supp = OJ` for `dosef = ME`. We compute the prediction by creating a new data frame which contains the values for the situations where we wish to compute the prediction:

```
coef(tooth2)
## (Intercept)      suppVC     dosefME     dosefHI
##       12.45       -3.70        9.13       15.50
new.data <- data.frame(supp = c("VC", "OJ"), dosef = "ME")
new.data
##   supp dosef
## 1   VC    ME
## 2   OJ    ME
predict(tooth2, newdata = new.data)
##     1     2
## 17.88 21.58
```

For the first observation where `supp = VC` for `dosef = ME` we get a predicted length of 17.88 while the predicted length is 21.58 for `supp = OJ` for `dosef = ME`.

An easier approach is as follows:

```
library(doBy)
at <- list(supp = c("VC", "OJ"), dosef = "ME")
K <- LE_matrix(tooth2, at = at)
K
##      (Intercept) suppVC dosefME dosefHI
## [1,]           1      1       1       0
## [2,]           1      0       1       0
linest(tooth2, K = K)
## Coefficients:
##              estimate     se     df t.stat p.value
## (Intercept)   12.455  0.988 56.000 12.603       0
## suppVC        -3.700  0.988 56.000 -3.744       0
## dosefME        9.130  1.210 56.000  7.543       0
## dosefHI       15.495  1.210 56.000 12.802       0
K2 <- LE_matrix(tooth2, at=new.data)
K2
##      (Intercept) suppVC dosefME dosefHI
## [1,]           1      1       1       0
## [2,]           1      0       1       0
## [3,]           1      1       1       0
## [4,]           1      0       1       0
```

Next, suppose the task is to compute the difference between `dosef = HI` and `dosef = ME`. This can be obtained as a difference between two linear predictors. We notice that this difference does not depend on `supp` and we get

```
lambda <- c(0, 0, -1, 1)
sum(coef(tooth2) * lambda)
## [1] 6.365
```

In practice we always want standard errors of such quantities:

```
library(doBy)
esticon(tooth2, L = lambda)
##         beta0 Estimate Std.Error  t.value       DF Pr(>|t|)    Lower Upper
## [1,] 0.00e+00 6.37e+00  1.21e+00 5.26e+00 5.60e+01 2.35e-06 3.94e+00  8.79
```

3.2.1 Estimation averaged across linear predictors

Suppose the task is to compute the mean response for `dosef = ME` averaged across the two `supp` methods.

We can obtain this by specifying λ manually (remembering that the coefficient for `supp = OJ` is set to zero):

```
lambda <- c(1, 1 / 2, 1, 0)
esticon(tooth2, lambda)
##       beta0 Estimate Std.Error t.value     DF Pr(>|t|)  Lower Upper
## [1,]  0.000   19.735     0.856  23.058 56.000    0.000 18.020  21.4
```

Such an average is called POPULATION AVERAGED MEAN or POPULATION MEAN or LEAST SQUARES MEAN or LSMEAN

```
library(doBy)
at <- list(dosef = "ME")
LSmeans(tooth2, at = at)
## Coefficients:
##          estimate     se     df t.stat p.value
## estimate   19.735  0.856 56.000 23.058       0
```

The rule is that

1. covariates not listed in the `at`–argument are fixed at their average and
2. an average is calculated over the levels of factors not listed in the `at`–argument.

To obtain the population means for all 3 levels of `dosef` when `supp = OJ` we do:

```
LSmeans(tooth2, effect = "dosef", at = list(supp = "OJ"))
## Coefficients:
##      estimate     se     df t.stat p.value
## [1,]   12.455  0.988 56.000 12.603       0
## [2,]   21.585  0.988 56.000 21.841       0
## [3,]   27.950  0.988 56.000 28.281       0
```

```
LSmeans(crm2, effect = "species")
## Coefficients:
##      estimate      se      df  t.stat p.value
## [1,]   77.772   0.533  27.000 145.841       0
## [2,]   67.874   0.493  27.000 137.639       0
LSmeans(crm2, effect = "species", at = list(temp = 27))
## Coefficients:
##      estimate      se      df  t.stat p.value
## [1,]   90.094   0.500  27.000 180.218       0
## [2,]   80.197   0.717  27.000 111.836       0
```

3.2.2 Summary information – `summary()`

```
summary(crm2)
##
## Call:
## lm(formula = pps ~ species + temp, data = crick)
##
## Residuals:
##     Min      1Q Median      3Q     Max
## -3.129 -1.296 -0.175  0.939  3.747
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -7.855      2.761   -2.85   0.0084 **
## speciesniv    -9.898      0.786  -12.59  8.2e-13 ***
## temp           3.628      0.106   34.38  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.81 on 27 degrees of freedom
## Multiple R-squared:  0.99,Adjusted R-squared:  0.989
## F-statistic: 1.28e+03 on 2 and 27 DF,  p-value: <2e-16
```

There is a detail about summary(): A call to summary() returns an object (which is a list):

```
crm2.sum <- summary(crm2)
class(crm2.sum)
## [1] "summary.lm"
names(crm2.sum)
##  [1] "call"          "terms"         "residuals"     "coefficients"
##  [5] "aliased"       "sigma"         "df"            "r.squared"
##  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
```

These elements of the list can be accessed directly using $ and in some cases by accessor methods:

```
coef(crm2)
## (Intercept)  speciesniv        temp
##      -7.855      -9.898       3.628
coef(crm2.sum)
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept)   -7.855     2.7605  -2.845 8.365e-03
## speciesniv    -9.898     0.7861 -12.591 8.193e-13
## temp           3.628     0.1055  34.378 7.912e-24
```

Table contains

- Parameter estimates
- Standard errors of estimates
- Test statitics for testing whether these parameters are zero
- Corresponding p–values

Notice that `coef()` acts differently when applied to a `summary.lm` and a `lm` object.

```
sigma(crm2)
## [1] 1.805
```

3.3 Prediction and confidence intervals

3.3.1 Confidence interval – `confint()`

Confidence interval $\hat{\beta}_i \pm 1.96 se(\hat{\beta}_i)$:

```
confint(crm2)
##               2.5 % 97.5 %
## (Intercept) -13.519 -2.190
## speciesniv  -11.511 -8.285
## temp          3.411  3.844
```

3.4 Testing hypotheses for linear models

Effects of factors are described by the "differences" their levels "cause" in the response. Classically, the term "contrast" is used for such differences. Consider again the additive model for the bacteria data

```
tooth2
##
## Call:
## lm(formula = len ~ supp + dosef, data = ToothGrowth)
##
## Coefficients:
## (Intercept)       suppVC      dosefME      dosefHI
##       12.45        -3.70         9.13        15.50
```

An initial step is usually to test for removal of each effect by `drop1()`; i.e.

```
drop1(tooth2, test = "F")
## Single term deletions
##
## Model:
## len ~ supp + dosef
##        Df Sum of Sq  RSS AIC F value  Pr(>F)
## <none>               820 165
## supp    1       205 1026 176    14.0 0.00043 ***
## dosef   2      2426 3247 244    82.8 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We have concluded that there is a significant effect of having `dosef` in the model. Next we go one step further and try to find out where this difference comes from: The `glht()`multcomp function in the **multcomp** package can do this and so can the `esticon`doBy from **doBy**.

We can compute all pairwise differences as:

```
library(multcomp)
ddd <- glht(tooth2, mcp(dosef = "Tukey"))
ddd
```

```
## Multiple Comparisons of Means: Tukey Contrasts
## Linear Hypotheses:
##              Estimate
## ME - LO == 0     9.13
## HI - LO == 0    15.50
## HI - ME == 0     6.37
```

We can get confidence intervals for the contrasts with

```
## Multiplicity adjusted confidence intervals:
confint(ddd)
## Unadjusted confidence intervals:
## confint(ddd, calpha = univariate_calpha())
```

```
##   Simultaneous Confidence Intervals
## Multiple Comparisons of Means: Tukey Contrasts
## 95% family-wise confidence level
## Linear Hypotheses:
##              Estimate lwr    upr
## ME - LO == 0  9.130    6.215 12.045
## HI - LO == 0 15.495   12.580 18.410
## HI - ME == 0  6.365    3.450  9.280
```

If we want to control the FAMILYWISE ERROR RATE use

```
summary(ddd, test = adjusted())
##
##   Simultaneous Tests for General Linear Hypotheses
##
## Multiple Comparisons of Means: Tukey Contrasts
##
##
## Fit: lm(formula = len ~ supp + dosef, data = ToothGrowth)
##
## Linear Hypotheses:
##                Estimate Std. Error t value Pr(>|t|)
## ME - LO == 0     9.13       1.21    7.54   <1e-05 ***
## HI - LO == 0    15.50       1.21   12.80   <1e-05 ***
## HI - ME == 0     6.37       1.21    5.26   <1e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
## same as summary(ddd)
```

Similarly, we can compute all differences relative to the reference level:

```
ddd2 <- glht(tooth2, mcp(dosef = "Dunnett"))
summary(ddd2)
```

Model comparison with `lm()`

```
crm22 <- lm(pps ~ species + temp + species:temp, data = crick)
crm2
##
## Call:
## lm(formula = pps ~ species + temp, data = crick)
##
## Coefficients:
## (Intercept)   speciesniv         temp
##       -7.85        -9.90         3.63
crm22
##
## Call:
## lm(formula = pps ~ species + temp + species:temp, data = crick)
##
## Coefficients:
##     (Intercept)       speciesniv             temp  speciesniv:temp
##         -11.041           -4.766            3.751           -0.213
```

The additive model is a submodel of the interaction model: The additive model can be formed by setting certain parameters to zero in the interaction model.

Most statistical programs produce standard output tables providing tests of several hypotheses. We will consider three of these tables, which are available in R(Tab. 3.3).

Table 3.3: Three commonly used tables.

R-function	Description
`anova(model)`	Sequential ANOVA table.
`drop1(model)`	Dropping effects.
`coef(summary(model))`	Parameter estimate table.

3.4.1 Dropping each term in turn using `drop1()`

The `drop1()` function provides a table, where effects are removed in turn – assuming that all the other effects are contained in the model. Consider

```
drop1(crm2, test = "F")
## Single term deletions
##
## Model:
## pps ~ species + temp
##         Df Sum of Sq  RSS   AIC F value  Pr(>F)
## <none>                 88  38.3
## species  1       517  604  94.1     159 8.2e-13 ***
## temp     1      3850 3938 150.3    1182 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The first row is a test for removing `species` from the model `pps ~ species + temp`; the second row is a test for removing `temp` from the model `pps ~ species + temp`. Next consider

```
drop1(crm22, test = "F")
## Single term deletions
##
## Model:
## pps ~ species + temp + species:temp
##              Df Sum of Sq  RSS  AIC F value Pr(>F)
## <none>                    84.7 39.1
## species:temp  1      3.24 88.0 38.3       1   0.33
```

Notice: When applying the `drop1()` function to the interaction model only the test for the interaction is returned: This is because R obeys the principle of marginality for factors: Do not test for a main effect if this is contained in higher order interactions.

3.4.2 Investigating parameter estimates using `coef()`

Some insight can be gained by looking at the parameter estimates, their standard errors and derived quantities:

```
coef(summary(crm22))
##                  Estimate Std. Error t value  Pr(>|t|)
## (Intercept)      -11.0408     4.2219 -2.6152 1.465e-02
## speciesniv        -4.7658     5.2045 -0.9157 3.682e-01
## temp               3.7514     0.1628 23.0380 7.961e-19
## speciesniv:temp   -0.2133     0.2138 -0.9975 3.277e-01
```

However, care must be taken here: Each test is a test for dropping a term from the model assuming that all other parameters are present. For example `spieciesniv` is the the difference in intercept between the `niv` and the `ex` spicies. It appears that this parameter can be removed from the model with two different slopes. Hence the test is really a test for a model with common intercept and different slopes against a model with different intercepts and different slopes. But a model with common intercept (that is when `temp = 0`) depends on the temperature scale: If temperature is changed from Celcisus to Fahrenheit, then the intercept will change. To drive home this point further, consider

```
coef(summary(crm2))
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept)   -7.855     2.7605  -2.845 8.365e-03
## speciesniv    -9.898     0.7861 -12.591 8.193e-13
## temp           3.628     0.1055  34.378 7.912e-24
```

In a model with common slopes, there is evidently a difference in the intercepts - as Fig. 3.5, p. 51 also suggests.

3.4.3 Which table to use?

All three tables provide a collection of tests. The most useful table is the table (produced by `drop1()` in R) because it tests for each effect in turn whether it is necessary. This feature is especially useful, if one starts with a large model containing several effects.

The sequential anova table is less useful because of the dependence of the tests on the sequence of the model terms.

The least useful table (with respect to testing) is the parameter estimate table. It provides (Wald) tests that each parameter is equal to zero. This can only be used as a test that the complete effect can be dropped, if either the effect is a factor with only two levels or a continuous covariate.

3.5 Underlying assumptions about linear models

The `lm()` function will fit a linear model to data as has been illustrated above. Now we shall be more precise about what a linear model is.

We have observed data $y_1, \dots, y_n$, and for each y_i there are p EXPLANATORY VARIABLES $x_{i1}, x_{i2}, \dots, x_{ip}$.

We shall make assumptions about the RANDOM PROCESS that has generated these data (sometimes called the DATA GENERATING PROCESS):

1. We shall assume that each y_i is a REALIZATION of a NORMAL DISTRIBUTED random variable Y_i where
$$Y_i \sim N(\mu_i, \sigma^2),$$
such that the mean of Y_i is $\mathbb{E}(Y_i) = \mu_i$ and the variance of Y_i is $\mathbb{V}\text{ar}(Y_i) = \sigma^2$ for $i = 1, \dots, n$. Notice that we assume CONSTANT VARIANCE, i.e. the variance of each Y_i is the same (and does for example not depend on the explanatory variables), namely σ^2.

2. We shall assume that that the mean μ_i can be written as a WEIGHTED SUM of the covariates
$$\mu_i = \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} \tag{3.6}$$
Often we choose $x_{i1} = 1$ for all i such that β_1 is the intercept .

3. We shall assume that the Y_i's are INDEPENDENT such that $\mathbb{C}\text{ov}(Y_i, Y_j) = 0$.

These assumptions are very important as they make it possible to perform statistical inference because the assumptions enables us to estimate the uncertainty in the estimated parameters (the so-called STANDARD ERROR of a parameter estimate).

From a practical modelling perspective focus is almost always on the form of the mean μ_i in Eq. (3.6) but there are many other underlying assumptions: normality, constant variance, independence.

Notice that an equivalent way of writing the first two assumptions is that

$$Y_i = \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \varepsilon_i$$

where $\varepsilon_i \sim N(0, \sigma^2)$.

We have seen that linear, multiple and polynomial regression falls under the category of linear models as do ANOVA and ANCOVA models. Section Sec. 3.5.1 discusses how to check if these assumptions are fulfilled.

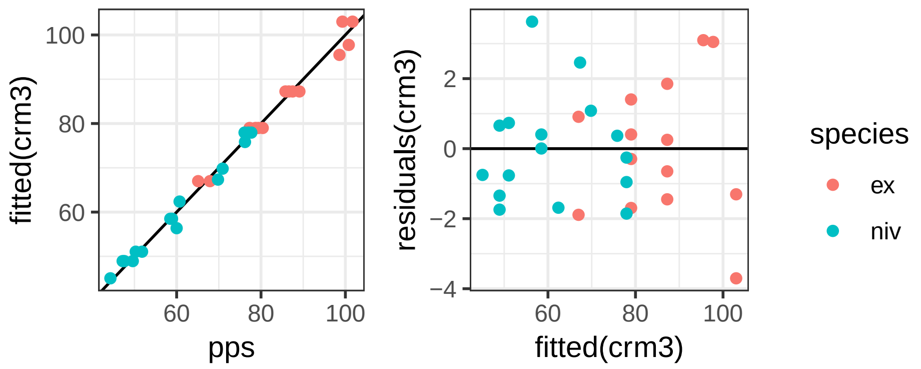


Figure 3.6: Diagnostic plots. On the left, the observed against the fitted values are shown. On the right, the fitted values against the raw residuals are shown.

3.5.1 Model checking – residuals etc

Checking a model is in many ways an art more than a science. Typically, it cannot be verified that the assumptions are satisfied. Instead, it can sometimes be demonstrated that the assumptions are not satisfied. Although it takes quite some experience to decide when a model does not fit adequately to data.

When we examine if a model fits the data we typically focus on the residuals. The RESIDUALS are computed as the observed minus the fitted values, and hence the residuals measure how far off the model is compared to the actual observed data.

$$r_i = y_i - \hat{\mu}_i$$

If the model fits to data, then it is possible to show that the r_i's are approximately $N(0, \sigma^2)$ distributed and approximately independent.

A minimum is to make the plots in Fig. 3.6.

Points should scatter randomly around the line (with no structure, e.g. the variance must not increase) if the models hold. This is not the case here, and moreover the residuals show a clustering according to treatment.

Figure Fig. 3.7 shows some example of residual plots when the model assumptions are not met.

(a) When things look right: A plot of what residuals could look like when the model assumptions are met.

(b) When the mean is not adequately specified: If the systematic part of the model is not adequate, i.e. if it does not account for all systematic variation in data, this can sometimes be seen when plotting the residuals against the fitted values. Remedy: add a covariate, here the square of the x-variable.

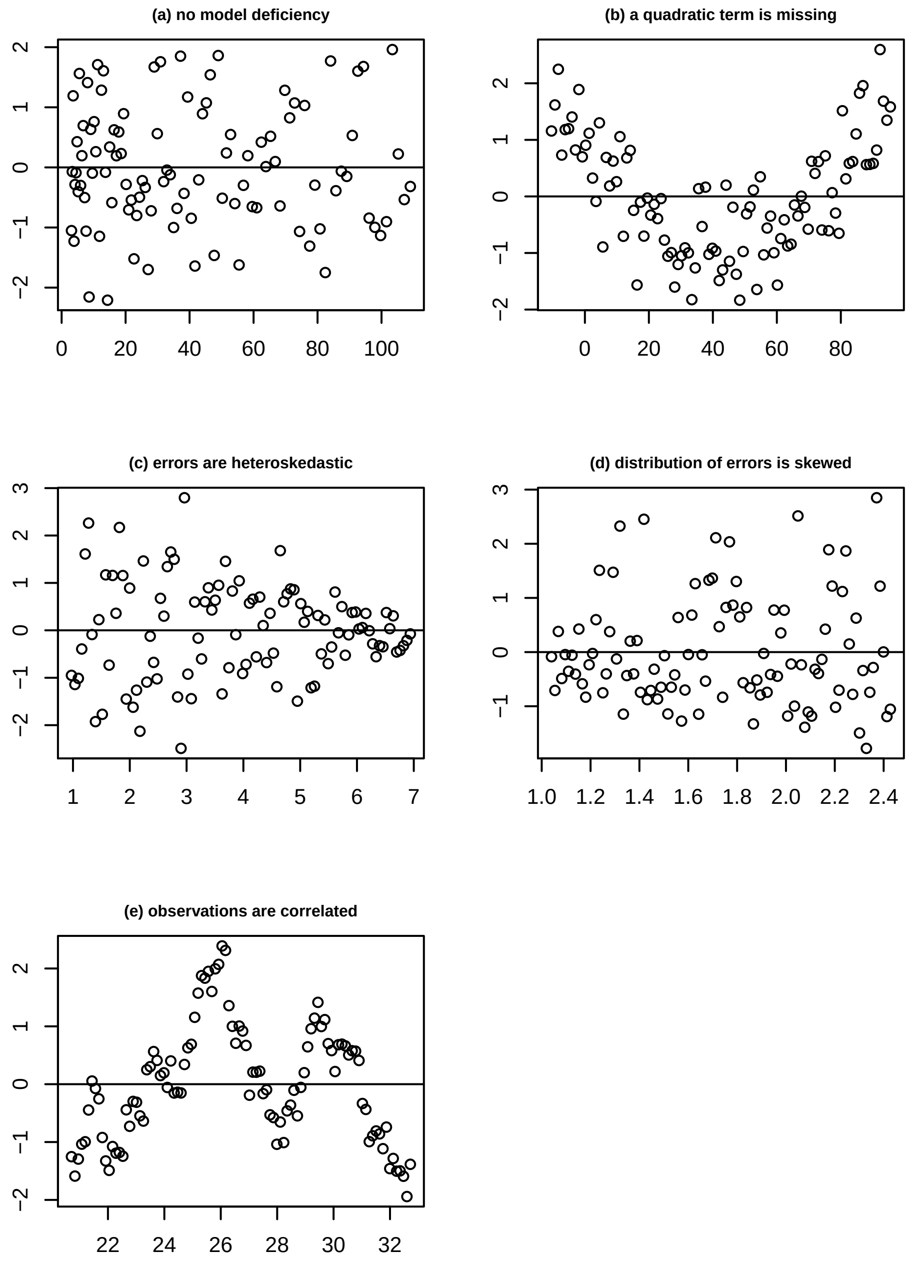


Figure 3.7: Examples of model deficiencies.

(c) When the variance is not constant: In some cases the variability of data increases with the mean such that the variance is not constant but increases with the mean. Figure 3.7, (c), shows what the residuals could look in this case. Remedy: Transform the response or use a GENERALIZED LINEAR MODEL with a variance function depending on the mean.

(d) When data are not normal: Fig. 3.7, (d), shows what residuals could look like if data are not normal (but, in this case, right skewed). Remedy: use a different distribution for the response with a generalized linear model or transform the response.

(e) When observations are not independent: Fig. 3.7, (e), shows what residuals could look like if data are not indpendent. Remedy: general advice: account for the dependency in the model, i.e. do not use a linear model, but for example a mixed model (see Chap. **??**).

3.6 Coefficient of determination etc

3.7 Colinearity

3.8 Model specification and model formulae

We have seen different specifications on the right hand side of a MODEL FORMULA in R, i.e. the part that comes after the tilde (˜).

We shall describe such specifications in more details based on this dataset where a and b are factors and x, y and z are variables:

```
dat2 <- cbind(expand.grid(a = c("a1", "a2"), b = c("b1", "b2")),
              x = 1:4, y = 11:14, z = 5:8 )
dat2
##    a  b x  y z
## 1 a1 b1 1 11 5
## 2 a2 b1 2 12 6
## 3 a1 b2 3 13 7
## 4 a2 b2 4 14 8
```

- The right hand side of a formula consists of a series of terms separated by "+" operators.
- A term consists of variable and factor names separated by ":" operators and a term is interpreted as the interaction of all the variables and factors appearing in the term.
- The "*" operator can be seen as a shortcut for creating many terms on the fly: Writing e.g. a*x is by R understood to be a+x+a:x.

From a formula and a dataset, the model matrix can be generated using model.matrix(). Below we use prmatrix() only to remove unimportant output.

```
prmatrix( model.matrix(~a + b:x, data = dat2) )
##   (Intercept) aa2 bb1:x bb2:x
## 1           1   0     1     0
## 2           1   1     2     0
## 3           1   0     0     3
## 4           1   1     0     4
```

Default is to include a column of ones in the model matrix. This can be overwritten with

```
prmatrix( model.matrix(~ -1 + a + b:x, data = dat2) )
##   aa1 aa2 bb1:x bb2:x
## 1   1   0     1     0
## 2   0   1     2     0
## 3   1   0     0     3
## 4   0   1     0     4
```

What does "a:a" now mean in this model language?

It means exactly the interaction between a and a which is a itself:

```
prmatrix( model.matrix(~ a : a, data = dat2) )
##   (Intercept) aa2
## 1           1   0
## 2           1   1
## 3           1   0
## 4           1   1
prmatrix( model.matrix(~ x : x, data = dat2) )
##   (Intercept) x
## 1           1 1
## 2           1 2
## 3           1 3
## 4           1 4
```

In addition to "a * b" being interpreted as "a + b + a : b" we have the following

- The "^" operator indicates crossing to the specified degree.

  For example "(a + b + c)^2" is identical to "(a + b + c)*(a + b + c)" which in turn expands to a formula containing the main effects for "a", "b" and "c" together with their second-order interactions.

- The "`-`" operator removes the specified terms, so that "`(a + b + c)^2 - a:b`" is identical to "`a + b + c + b:c + a:c`". It can also used to remove the intercept term: when fitting a linear model the form "`y ~ x - 1`" specifies a line through the origin. A model with no intercept can be also specified as "`y ~ x + 0`" or "`y ~ 0 + x`".

3.8.1 Formulae with arithmetic expressions

- Formulae can also involve arithmetic expressions. The formula "`log(y) ~ a + log(x)`" is legal.
- When such arithmetic expressions involve operators which are also used symbolically in model formulae, there can be confusion between arithmetic and symbolic operator use.
- To avoid confusion, the function `I()` is used to bracket those portions of a model formula where operators are used in their arithmetic sense. For example, in the formula "`y ~ a + I(x + z)`", the term "`x + z`" is to be interpreted as the sum of "`x`" and "`z`".

Example

```
prmatrix(
  model.matrix(~ x + z + I(x + z) + I(x^2) + sin(x + z) + log(x),
               data = dat2))
##   (Intercept) x z I(x + z) I(x^2) sin(x + z) log(x)
## 1           1 1 5        6      1    -0.2794 0.0000
## 2           1 2 6        8      4     0.9894 0.6931
## 3           1 3 7       10      9    -0.5440 1.0986
## 4           1 4 8       12     16    -0.5366 1.3863
```

3.9 Polynomial regression

The function `poly()` is convenient for specifying a polynomial:

```
p1 <- prmatrix(with(dat2, poly(x, 3)))
##            1    2       3
## [1,] -0.6708  0.5 -0.2236
## [2,] -0.2236 -0.5  0.6708
## [3,]  0.2236 -0.5 -0.6708
## [4,]  0.6708  0.5  0.2236
p2 <- prmatrix(with(dat2, poly(x, 3, raw = T)))
##      1  2  3
## [1,] 1  1  1
## [2,] 2  4  8
## [3,] 3  9 27
## [4,] 4 16 64
```

In the first case, the columns are orthogonal; in the second case they are not (but the two matrices span the same space).

A polynomium can put into a model formula in different ways:

```
prmatrix( model.matrix(~ x + I(x^2), data = dat2) )
##   (Intercept) x I(x^2)
## 1           1 1      1
## 2           1 2      4
## 3           1 3      9
## 4           1 4     16
prmatrix( model.matrix(~ poly(x, 2), data = dat2) )
##   (Intercept) poly(x, 2)1 poly(x, 2)2
## 1           1     -0.6708         0.5
## 2           1     -0.2236        -0.5
## 3           1      0.2236        -0.5
## 4           1      0.6708         0.5
```

Consider the polynomial model for the the `potatoes` data in Sec. 3.1.1

```
pot1_lw2
##
## Call:
## lm(formula = weight ~ length + width + I(width^2), data = potatoes)
##
## Coefficients:
## (Intercept)       length        width   I(width^2)
##      8.8400       0.8310      -2.6180       0.0681
```

corresponding to the mathematical formulation

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i2}^2 + \varepsilon_i.$$

If we have a linear relation between y and x, i.e. $y = f(x) = \alpha_0 + \alpha_1 x$ then the interpretation is that when x changes from x to $x + h$ then y changes from $\alpha_0 + \alpha_1 x$ to $\alpha_0 + \alpha_1(x+h) = \alpha_0 + \alpha_1 x + \alpha_1 h$. Hence the change on the y–scale is $f(x+h) - f(x) = \alpha_1 h$ which is caused by a change on the x–scale by h. This effect is the same no matter which x we start with.

If instead $f(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2$ the interpretation is more involved. We have

$$\begin{aligned} f(x+h) &= \alpha_0 + \alpha_1(x+h) + \alpha_2(x+h)^2 \\ &= \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \alpha_1 h + \alpha_2(h^2 + 2hx) \\ &= f(x) + \alpha_1 h + \alpha_2(h^2 + 2hx) \end{aligned}$$

Hence a change on the x-scale from x to $x+h$ means a change on the y scale of $f(x+h) - f(x) = \alpha_1 h + \alpha_2(h^2 + 2hx)$ and this amount depends (linearly) on the value of x.

This change (along with standard errors) can be calculated in different ways, for example as follows. First, let $\beta_3 = \alpha_2$ and $\beta_2 = \alpha_1$.

```
library(car)
deltaMethod(pot1_lw2, "b2*h + b3*(h^2 + 2*x*h)",
            parameterNames = c("b0", "b1", "b2", "b3"),
            constants = c(x = 5, h = 1))
##                                   Estimate    SE 2.5 %  97.5 %
## b2 * h + b3 * (h^2 + 2 * x * h)    -1.868 0.445 -2.74 -0.9963
x <- 5; h <-1
doBy::linest(pot1_lw2, c(0, 0, h, h^2 + 2*x*h))
## Coefficients:
##           estimate     se     df t.stat p.value
## estimate    -1.868  0.445 16.000 -4.199       0
```

Let us se the effect for a range of different x-values (speed)

```
f <- function(x.val){
    deltaMethod(pot1_lw2, "b2*h + b3*(2*x*h + h^2)",
                parameterNames = c("b0", "b1", "b2", "b3"),
                constants = c(x = x.val, h = 1))
}
widths <- seq(0, 50, by = 10)
out <- lapply(widths, f)
out <- do.call(rbind, out)
out <- as.data.frame(cbind(out, widths))
rownames(out) <- NULL
out
##   Estimate     SE   2.5 %  97.5 % widths
## 1  -2.5499 0.5583 -3.6442 -1.4556      0
## 2  -1.1869 0.3372 -1.8478 -0.5260     10
## 3   0.1761 0.1828 -0.1823  0.5345     20
## 4   1.5391 0.2664  1.0170  2.0612     30
## 5   2.9021 0.4756  1.9698  3.8343     40
## 6   4.2651 0.7066  2.8802  5.6500     50
names(out)[c(3, 4)] <- c("lwr", "upr")
out
##   Estimate     SE     lwr     upr widths
## 1  -2.5499 0.5583 -3.6442 -1.4556      0
## 2  -1.1869 0.3372 -1.8478 -0.5260     10
## 3   0.1761 0.1828 -0.1823  0.5345     20
## 4   1.5391 0.2664  1.0170  2.0612     30
## 5   2.9021 0.4756  1.9698  3.8343     40
## 6   4.2651 0.7066  2.8802  5.6500     50
```

The effect of increasing width by one unit from 10 to 11 is -1.1869 while the effect of increasing width by one unit from 20 to 21 is 0.1761.

Notice that the standard errors (confidence bands) of these predictions increase the further x moves away from the center of data. See Fig.3.8.

```
qplot(widths, Estimate, data = out) +
    geom_ribbon(aes(ymin = lwr, ymax = upr), alpha = .2)
```

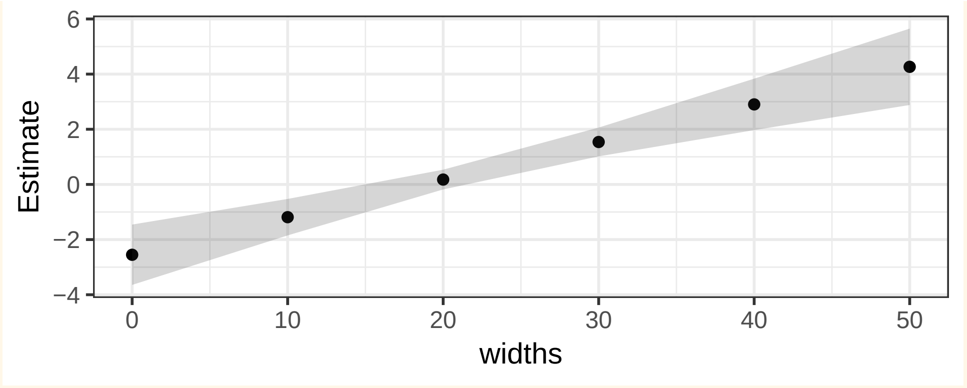


Figure 3.8: Applying the delta method to a polynomial regression. Gray indicate 95 % confidence bands.

3.9.1 Summaries using broom

The standard R`summary()` is as follows:

```
crm22.sum <- summary(crm22)
crm22.sum
##
## Call:
## lm(formula = pps ~ species + temp + species:temp, data = crick)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -3.703 -1.332 -0.124  0.867  3.628
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -11.041      4.222   -2.62    0.015 *
## speciesniv         -4.766      5.204   -0.92    0.368
## temp                3.751      0.163   23.04   <2e-16 ***
## speciesniv:temp    -0.213      0.214   -1.00    0.328
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.81 on 26 degrees of freedom
## Multiple R-squared:  0.99,Adjusted R-squared:  0.989
## F-statistic:  854 on 3 and 26 DF,  p-value: <2e-16
```

Shorter summaries using the **broom** package:

```
broom::tidy(crm22)
## # A tibble: 4 x 5
##   term            estimate std.error statistic  p.value
##   <chr>              <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)      -11.0       4.22     -2.62  1.46e- 2
## 2 speciesniv        -4.77      5.20     -0.916 3.68e- 1
## 3 temp               3.75      0.163    23.0   7.96e-19
## 4 speciesniv:temp   -0.213     0.214    -0.998 3.28e- 1
```

```
broom::glance(crm22)
## # A tibble: 1 x 11
##   r.squared adj.r.squared sigma statistic  p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>    <dbl> <int>  <dbl> <dbl> <dbl>
## 1     0.990         0.989  1.81      854. 4.39e-26     4  -58.1  126.  133.
## # ... with 2 more variables: deviance <dbl>, df.residual <int>
sigma(crm22)
## [1] 1.805
```

Output contains:

- Estimate $\hat{\sigma}$ of the residual standard error.
- Degrees of freedom are number of observations minus number of regression parameters
- Coefficient of determination (R^2 and adjusted R^2): Percentage of variation in data explained by the model.
- F–statistic: The result from comparing the model with the simplest possible model, namely the model with 1 on the right hand side.

3.10 What if the model assumptions are not satisfied?

Consider this setting: A one way anova model for investigating, say a treatment versus a control. We compare the means μ_1 and μ_2. Since we make the experiment by simulation, we know the answer: We choose μ_1 and μ_2 to be identical:

```
N <- 5
mu1 <- mu2 <- 7
sd1 <- 1; sd2 <- 1
sd <- c(rep(sd1, N), rep(sd2, N))
mu <- c(rep(mu1, N), rep(mu2, N))
fact <- factor(c(rep("ctl", N), rep("trt", N)))
fact
##  [1] ctl ctl ctl ctl ctl trt trt trt trt trt
## Levels: ctl trt
y <- rnorm(2 * N, mean = mu, sd = sd)
y
##  [1] 6.731 6.494 9.077 7.670 7.204 6.963 7.116 6.744 6.325 7.942
```

Compare treatment with control; is there a significant difference between means (short answer: no, because we generated data that way). Null hypothesis: $\mu_1 = \mu_2$; alternative hypothesis: $\mu_1 \neq \mu_2$.

At least two ways of making this test in R:

```
a <- t.test(y ~ fact); a
##
##  Welch Two Sample t-test
##
## data:  y by fact
## t = 0.79, df = 6.4, p-value = 0.5
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.8573  1.6919
## sample estimates:
## mean in group ctl mean in group trt
##             7.435             7.018
p <- a$p.value
lm(y ~ fact) %>% summary()  %>% coef()
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept)   7.4353     0.3744 19.8593 4.306e-08
## facttrt      -0.4173     0.5295 -0.7881 4.534e-01
```

Sometimes a hypothesis is rejected; even if the hypothesis is true. This is called an error of type I. If we us a 5% significance level, then there is 5% probability of make an error of type I.

If we redo a study 100 times, then even if the hypothesis is true, we will in 5 cases (on average) reject the hypothesis.

```
Nsim <- 1000
p.vec <- rep(0, Nsim)
for( i in 1:Nsim ){
    y <- rnorm(2 * N, mean = mu, sd = sd)
    a <- t.test(y ~ fact, var.equal = TRUE)
    p.vec[i] <- a$p.value
}
```

We see that we reject the null hypothesis about the right number of times:

```
c(sum(p.vec < 0.10), sum(p.vec < 0.05), sum(p.vec < 0.01)) / Nsim
## [1] 0.091 0.044 0.010
```

3.10.1 Non–normality

Let now the setting be that observations do not come from independent normal distributions but from independent Poisson distributions:

```
Nsim <- 1000
p.vec <- rep(0, Nsim)
for( i in 1:Nsim ){
    y <- rpois(2 * N, lambda = mu)
    a <- t.test(y ~ fact, var.equal = TRUE)
    p.vec[i] <- a$p.value
}
```

We see that we reject the null hypothesis about the right number of times:

```
c(sum(p.vec < 0.10), sum(p.vec < 0.05), sum(p.vec < 0.01)) / Nsim
## [1] 0.104 0.062 0.009
```

3.10.2 Non–constant variance

Now let us imagine that the standard deviation in the treatment group is much larger than in the control group.

```r
N <- 5
sd1 <- 1; sd2 <- 5
fact <- factor(c(rep("ctl", N), rep("trt", N)))
sd <- c(rep(sd1, N), rep(sd2, N))
```

```r
Nsim <- 1000
p.vec <- rep(0, Nsim)
for( i in 1:Nsim ){
    y <- rnorm(2 * N, mean = mu, sd = sd)
    a <- t.test(y ~ fact, var.equal = TRUE)
    p.vec[i] <- a$p.value
}
```

We still reject the hypthesis about the right number of times, but not quite: We reject the hypothesis too often. This means that we are more likely to erroneously claim that we found a significant difference that is not there.

```r
c(sum(p.vec < 0.10), sum(p.vec < 0.05), sum(p.vec < 0.01)) / Nsim
## [1] 0.126 0.075 0.025
```

Now make make the standard deviation in one group much much larger than in the other:

```r
N <- 5
sd1 <- 1; sd2 <- 25
fact <- factor( c(rep("ctl", N), rep("trt", N)) )
sd <- c(rep(sd1, N), rep(sd2, N))
```

```r
Nsim <- 1000
p.vec <- rep(0, Nsim)
for( i in 1:Nsim ){
    y <- rnorm(2 * N, mean = mu, sd = sd)
    a <- t.test(y ~ fact, var.equal = TRUE)
    p.vec[i] <- a$p.value
}
```

We reject the null hypothesis too often

```r
c(sum(p.vec < 0.10), sum(p.vec < 0.05), sum(p.vec < 0.01)) / Nsim
## [1] 0.127 0.075 0.031
```

Tentative conclusion: We need a somewhat extreme situation before the tests become really misleading (when it comes to tests).

3.10.3 Non–independence

Suppose the variance of all random variables is σ^2 (constant variance), observations in different groups are independent but observations within each group are all positively correlated with correlation ρ.

```
N <- 5
rho <- 0.8
Vi <- matrix(rho, nr = N, nc = N)
diag(Vi) <- 1
V <- kronecker(diag(1,2), Vi)
V
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]  1.0  0.8  0.8  0.8  0.8  0.0  0.0  0.0  0.0   0.0
##  [2,]  0.8  1.0  0.8  0.8  0.8  0.0  0.0  0.0  0.0   0.0
##  [3,]  0.8  0.8  1.0  0.8  0.8  0.0  0.0  0.0  0.0   0.0
##  [4,]  0.8  0.8  0.8  1.0  0.8  0.0  0.0  0.0  0.0   0.0
##  [5,]  0.8  0.8  0.8  0.8  1.0  0.0  0.0  0.0  0.0   0.0
##  [6,]  0.0  0.0  0.0  0.0  0.0  1.0  0.8  0.8  0.8   0.8
##  [7,]  0.0  0.0  0.0  0.0  0.0  0.8  1.0  0.8  0.8   0.8
##  [8,]  0.0  0.0  0.0  0.0  0.0  0.8  0.8  1.0  0.8   0.8
##  [9,]  0.0  0.0  0.0  0.0  0.0  0.8  0.8  0.8  1.0   0.8
## [10,]  0.0  0.0  0.0  0.0  0.0  0.8  0.8  0.8  0.8   1.0
Nsim <- 1000
Y <- MASS::mvrnorm(Nsim, mu = mu, Sigma = V)
head(Y)
##       [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10]
## [1,] 7.044 7.443 6.854 6.626 7.834 6.336 6.396 6.050 6.206 7.155
## [2,] 7.506 7.529 8.018 7.504 6.748 6.449 7.394 6.389 7.506 7.015
## [3,] 7.371 8.141 7.811 7.948 8.615 6.544 6.614 6.289 6.357 6.933
## [4,] 7.929 7.991 7.870 7.778 7.711 7.821 8.420 8.455 7.901 8.547
## [5,] 6.511 6.600 6.321 6.973 6.656 8.584 7.357 7.728 7.879 8.120
## [6,] 7.621 6.508 6.470 7.448 7.384 7.446 6.824 7.392 7.160 7.830
```

```
p.vec <- rep(0, Nsim)
for( i in 1:Nsim ){
    y <- Y[i, ]
    a <- t.test(y ~ fact, var.equal = T)
    p.vec[i] <- a$p.value
}
```

Now things go terribly wrong:

```
c(sum(p.vec < 0.10), sum(p.vec < 0.05), sum(p.vec < 0.01)) / Nsim
## [1] 0.704 0.643 0.497
```

We reject a true null hypothesis way too often, so the risk of erroneously concluding that there is a significant difference (when there is not) is very large.

The reason for this is that if we ignor are positive correlation then we pretend to have more information than we really have:

Just look at the sample mean from group i (treatment or control):

$$\bar{y}^i = \frac{1}{N}\sum_{j=1}^{N} y_{ij}$$

The variance of $\bar{y}^i$ is

$$\mathbb{V}\text{ar}(\bar{y}^i) = \frac{1}{N^2}\mathbb{V}\text{ar}(\sum_{j=1}^{N} y_{ij})$$

If observations are independent then the variance of the sum is the sum of the variances and we get

$$\mathbb{V}\text{ar}(\bar{y}^i) = \frac{1}{N^2}N\sigma^2 = \sigma^2/N$$

But this is not the case when observations are not independent.

It is always true that

$$\mathbb{V}\text{ar}(x_1 + x_2 + \cdots + x_n) = \sum_{i=1}^{N}\sum_{j=1}^{N}\mathbb{C}\text{ov}(x_i, x_j)$$

In the specific case we get

$$\mathbb{V}\text{ar}(\sum_{j=1}^{N} y_j^i) = N\sigma^2 + N(N-1)\sigma^2\rho/2 = N\sigma^2(1 + (N-1)\rho/2)$$

so

$$\mathbb{V}\text{ar}(\bar{y}^i) = \frac{1}{N^2}N\sigma^2 = \frac{1}{N}\sigma^2(1 + (N-1)\rho/2)$$

which is larger than σ^2/N.

It is interesting to ask the question: How many independent observations, say M, do the N correlated observations correspond to? That is, which M will give

$$\frac{\sigma^2}{M} = \frac{1}{N}\sigma^2(1 + (N-1)\rho/2)$$

and the answer is

$$M = \frac{N}{1 + (N-1)\rho/2}$$

So for $N = 5$ and $\rho = 0.8$ we get

```
N <- 5
rho <- .8
M <- N / (1 + (N - 1) * rho / 2)
M
## [1] 1.923
```

so we have about "two independent pieces of information" and not 5 as we pretended to have.

3.11 The mathematics of linear models

3.11.1 What is a linear model (II) - the assumptions

The `lm()` function will fit a linear model to data as has been illustrated above. Now we shall be more precise about what a linear model is.

From the perspective of data, the situation is: We have observed data or RESPONSES $y_1, \dots, y_N$. To each y_i there are p EXPLANATORY VARIABLES $x_{i1}, x_{i2}, \dots, x_{ip}$.

We shall make assumptions about the RANDOM PROCESS that has generated these data:

1. We shall assume that each y_i is a REALIZATION of a NORMAL DISTRIBUTED random variable Y_i where
$$Y_i \sim N(\mu_i, \sigma^2),$$
such that the mean of Y_i is $\mathbb{E}(Y_i) = \mu_i$ and the variance of Y_i is $\mathbb{V}\text{ar}(Y_i) = \sigma^2$ for $i = 1, \dots, N$. Notice that we assume CONSTANT VARIANCE, i.e. the variance of each Y_i is the same, namely σ^2.
2. We shall assume that that the mean μ_i can be written as a WEIGHTED SUM of the covariates
$$\mu_i = x_{i1}\beta_1 + x_{i2}\beta_2 + \cdots + x_{ip}\beta_p \qquad (3.7)$$
3. We shall assume that the Y_i's are INDEPENDENT such that $\mathbb{C}\text{ov}(Y_i, Y_j) = 0$.

From a practical modelling perspective focus is almost always on the form of the mean μ_i in (3.7) but there many other underlying assumptions: normality, constant variance, independence.

Notice that an equivalent way of writing the first two assumptions is that

$$Y_i = x_{i1}\beta_1 + x_{i2}\beta_2 + \cdots + x_{ip}\beta_p + e_i$$

where $e_i \sim N(0, \sigma^2)$.

We have seen that linear, multiple and polynomial regression falls under the category of linear models. So do ANCOVA models and so do ANOVA models.

When these assumptions are satisfied, there is a whole theory about "what to do". in terms of statistical inference.

This raises several questions:

1. What is this theory behind linear models?
2. How to verify that the assumptions are satisfied? This can usually not be done, but sometimes we can do the opposite: Convince ourselves that the assumptions are not satisfied. This requires subject matter knowledge and statistical techniques.
3. Which of the assumptions are most important? Normality, constant variance or independence? And what goes wrong when these assumtions are not satisfied?

3.11.2 Matrix representation of a linear model

Return again to the linear model

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + e$$

If we have N cases in a dataset we can organize the y's in the vector $y = (y_1, y_2, \ldots, y_N)$ and the predictors in the matrix X:

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & x_{13} \\ 1 & x_{21} & x_{22} & x_{23} \\ \vdots & \vdots & \vdots & \\ 1 & x_{N1} & x_{N2} & x_{N3} \end{bmatrix}$$

If we let $\beta = (\beta_0, \beta_1, \beta_2, \beta_3)$ and $e = (e_1, e_2, \ldots, e_N)$ then we can write

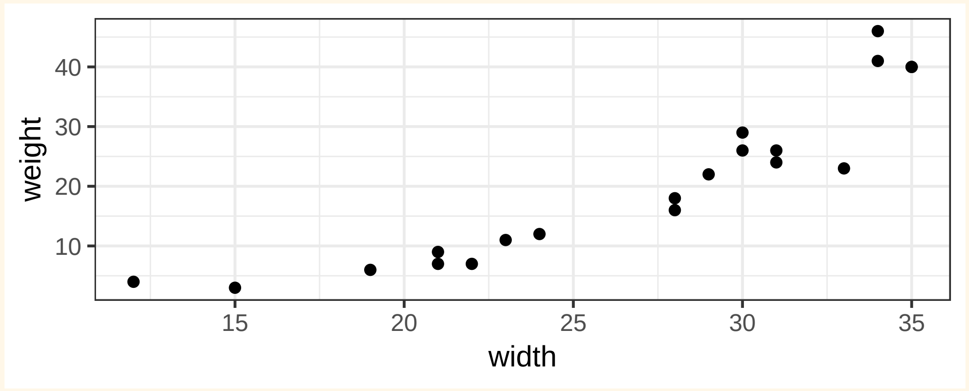
$$y = X\beta + e$$

Notice that what is observed is y and X.

3.11.3 Least squares – a minimization problem

Weight and width of potatoes:

```
library(ggplot2)
qplot(width, weight, data=potatoes)
```



Perhaps an approximately quadratic relation between y=`weight` and x=`width`:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + e_i$$

```
X <- cbind(1, potatoes$width, potatoes$width^2)
y <- potatoes$weight
head(X)
##      [,1] [,2] [,3]
## [1,]    1   29  841
## [2,]    1   34 1156
## [3,]    1   31  961
## [4,]    1   28  784
## [5,]    1   21  441
## [6,]    1   35 1225
head(y)
## [1] 22 41 24 16  7 40
```

Let $\beta = (\beta_0, \beta_1, \beta_2)$. One way of estimating β is by the method of LEAST SQUARES: The best β is the vector that minimizes the sum–of–squares:

$$\sum_{i=1}^{N} [y_i - (\beta_0 + \beta_1 x_i + \beta_2 x_i^2)]^2$$

In `R`, this vector can be found using the `lm()` function:

```
mm <- lm(weight ~ width + I(width^2), data=potatoes)
beta <- coef(mm); beta
## (Intercept)       width  I(width^2)
##    25.15450    -2.83987     0.09394
```

We can write

$$y_i \approx \beta_0 + \beta_1 x_i + \beta_2 x_i^2$$

in matrix form for all N observations:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \approx \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix}$$

or more compact

$$y \approx X\beta$$

Let $r = y - X\beta$. Then the LEAST SQUARES criterion can be formulated as: Choose as β the vector that makes the squared length of r (denoted by $||r||^2$) as small as possible.

The squared length is $r^\top r$ is called the RESIDUAL SUM OF SQUARES denoted RSS so the task is to minimize

$$RSS = (y - X\beta)^\top (y - X\beta)$$

One approach to doing so is to think of RSS as a function of β. We differentiate RSS with respect to β and set the derivatives to zero.

This leads to a system of equations called the NORMAL EQUATIONS

$$(X^\top X)\beta = X^\top y$$

and the solution to this system of equations is

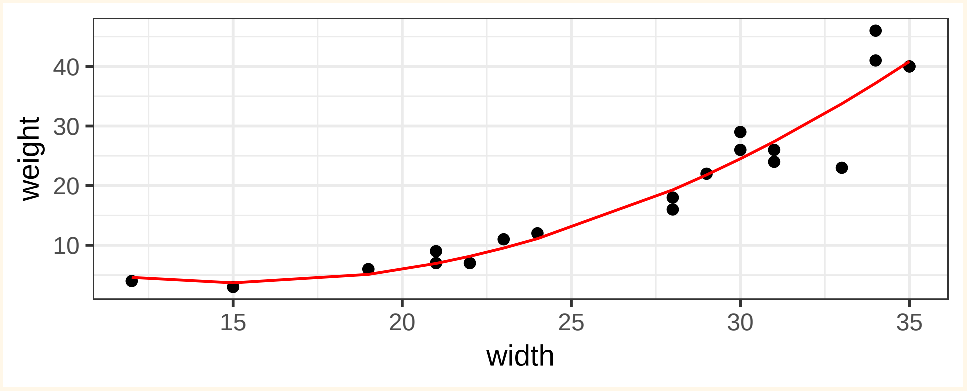
$$\beta = (X^\top X)^{-1} X^\top y$$

Define X and y and find β

Multiplying X and β gives the vector of FITTED VALUES:

```
fv <- X %*% beta; head(fv, 3)
##        [,1]
## [1,] 21.80
## [2,] 37.19
## [3,] 27.39
```

```
library(ggplot2)
qplot(width, weight, data=potatoes) +
    geom_line(aes(width, as.numeric(fv)), col="red")
```



Index

List of Corrections