ESMA 6836: Computational Statistics with R

Dr. Wolfgang Rolke

December 2018

Getting Started

Resma3.RData (Ver 3.1)

R

For a detailed introduction to R you can read the material of my course Computing with R

Installation and Updating

Installing R

You can get a free version of R for your computer from a number of sources. The download is about 70MB and setup is fully automatic. Versions for several operating systems can be found on the R web site

https://cran.r-project.org

Note

- the one item you should change from the defaults is to install R into a folder under the root, aka C:\R
- You might be asked at several times whether you want to do something (allow access, run a program, save a library, . . .), always just say yes!
- You will need to connect to a reasonably fast internet for these steps.
- This will take a few minutes, just wait until the > sign appears.

FOR MAC OS USERS ONLY

There are a few things that are different from MacOS and Windows. Here is one thing you should do:

Download XQuartz - XQuartz-2.7.11.dmg
Open XQuartz
Type the letter R (to make XQuartz run R)
Hit enter Open R Run the command .First()
Then, every command should work correctly.

RStudio

We will run R using an interface called **RStudio**. You can download it at RStudio.

Updating

R releases new versions about every three months or so. In general it is not necessary to get the latest version every time. Every now and then a package won't run under the old version, and then it is time to do so. In essence this just means to install the latest version of R from CRAN. More important is to now also update ALL your packages to the latest versions. This is done simply by running

```
update.packages(ask=FALSE, dependencies=TRUE)
```

R Markdown, HTML and Latex

R Markdown

R Markdown is a program for making dynamic documents with R. An R Markdown document is written in *markdown*, an easy-to-write plain text format with the file extension .Rmd. It can contain chunks of embedded R code. It has a number of great features:

- easy syntax for a number of basic objects
- code and output are in the same place and so are always synced
- several output formats (html, latex, word)

In recent years I (along with many others) who work a lot with R have made Rmarkdown the basic way to work with R. So when I work on a new project I immediately start a corresponding R markdown document.

---

to start writing an R Markdown document open RStudio, File > New File > R Markdown. You can type in the title and some other things.

The default document starts like this:

---
title: "My first R Markdown Document"
author: "Dr. Wolfgang Rolke"
date: "April 1, 2018"
output: html_document

This follows a syntax called YAML (also used by other programs). There are other things that can be put here as well, or you can erase all of it.

YAML stands for *Yet Another Markup Language*. It has become a standard for many computer languages to describe different configurations. For details go to yaml.org

Then there is other stuff you should erase. Next File > Save. Give the document a name with the extension .Rmd

Basic R Markdown Syntax

for a list of the basic syntax go to

https://rmarkdown.rstudio.com/articles_intro.html

or to

https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet

Embedded Code

There are two ways to include code chunks (yes, that's what they are called!) into an R Markdown document:

stand alone code

simultaneously enter CTRL-ALT-i and you will see this:

```{r}
```

Here ` is the *backtick*, on most keyboards on the ~ key, upper left below Esc.

you can now enter any R code you like:

```{r}
x <- rnorm(10)
mean(x)
```

which will appear in the final document as

```
x <- rnorm(10)
mean(x)
```

Actually, it will be like this:

```
x <- rnorm(10)
mean(x)
```

```
## [1] -0.1869992
```

so we can see the result of the R calculation as well. The reason it didn't appear like this before was that I added the argument eval=FALSE:

```{r eval=FALSE}
x <- rnorm(10)
mean(x)
```

which keeps the code chunk from actually executing (aka *eval*uating). This is useful if the code takes along time to run, or if you want to show code that is actually faulty, or ...

there are a number of useful arguments:

- eval=FALSE (shows but doesn't run the code)
- echo=FALSE (the code chunk is run but does not appear in the document)
- warning=FALSE (warnings are not shown)
- message=FALSE (messages are not shown)
- cache=TRUE (code is run only if there has been a change, useful for lengthy calculations)
- error=TRUE (if there are errors in the code R normally terminates the parsing (executing) of the markdown document. With this argument it will ignore the error, which helps with debugging)

inline code

here is a bit of text:

... and so the mean was -0.1869992.

Now I didn't type in the number, it was done with the chunk `` `r mean(x)` ``.

Many of these options can be set globally, so they are active for the whole document. This is useful so you don't have to type them in every time. I have the following code chunk at the beginning of all my Rmd files:

```
library(knitr)
opts_chunk$set(fig.width=6, fig.align = "center",
      out.width = "70%", warning=FALSE, message=FALSE)
```

We have already seen the message and warning options. The other one puts any figure in the middle of the page and sizes it nicely.

If you have to override these defaults just include that in the specific chunk.

Creating Output

To create the output you have to "knit" the document. This is done by clicking on the *knit* button above. If you click on the arrow you can change the output format.

HTML vs Latex(Pdf)

In order to knit to pdf you have to install a latex interpreter. My suggestion is to use Miktex, but if you already have one installed it might work as well.

There are several advantages / disadvantages to each output format:

- HTML is much faster
- HTML looks good on a webpage, pdf looks good on paper
- HTML needs an internet connection to display math, pdf does not
- HTML can use both html and latex syntax, pdf works only with latex (and a little bit of html)

I generally use HTML when writing a document, and use pdf only when everything else is done. There is one problem with this, namely that a document might well knit ok to HTML but give an error message when knitting to pdf. Moreover, those error messages are weird! Not even the line numbers are anywhere near right. So it's not a bad idea to also knit to pdf every now and then.

As far as this class is concerned, we will use HTML exclusively.

Tables

One of the more complicated things to do in R Markdown is tables. For a nice illustration look at

https://stackoverflow.com/questions/19997242/simple-manual-rmarkdown-tables-that-look-good-in-html-p

My preference is to generate a data frame and the use the *kable.nice* function:

```
Gender <- c("Male", "Male", "Female")
Age <- c(20, 21, 19)
kable.nice(data.frame(Gender, Age))
```

Gender	Age
Male	20
Male	21
Female	19

probably with the argument echo=FALSE so only the table is visible.

LATEX

You have not worked with latex (read: latek) before? Here is your chance to learn. It is well worthwhile, latex is the standard document word processor for science. And once you get

used to it, it is WAY better and easier than (say) Word.

A nice list of common symbols is found on https://artofproblemsolving.com/wiki/index.php/LaTeX:Symbols.

inline math

A LATEX expression always starts and ends with a $ sign. So this line:

The population mean is defined by $E[X] = \int_{-\infty}^{\infty} xf(x)dx$.

was done with the code

```
The population mean is defined by $E[X] = \int_{-\infty}^{\infty} xf(x) dx$.
```

displayed math

sometimes we want to highlight a piece of math:

The population mean is defined by

$$E[X] = \int_{-\infty}^{\infty} xf(x)dx$$

this is done with two dollar signs:

```
$$
E[X] = \int_{-\infty}^{\infty} xf(x) dx
$$
```

multiline math

say you want the following in your document:

$$E[X] = \int_{-\infty}^{\infty} xf(x)dx =$$
$$\int_0^1 xdx = \frac{1}{2}x^2|_0^1 = \frac{1}{2}$$

for this to display correctly in HTML and PDF you need to use the format

```
$$
\begin{aligned}
&E[X] = \int_{-\infty}^{\infty} xf(x) dx=\\
&\int_{0}^{1} x dx    = \frac12 x^2 |_0^1 = \frac12
\end{aligned}
$$
```

R Basics I

To start run

```
ls()
```

This shows you a "listing"" of the files (data, routines etc.) in the current project. (Likely there is nothing there right now)

Everything in R is either a data set or a function. It is a function if it is supposed to do something (maybe calculate something, show you something like a graph or something else etc.). If it is a function is ALWAYS NEEDS (). Sometimes the is something in between the parentheses, like in

```
mean(x)
```

```
## [1] 6
```

Sometimes there isn't like in the ls(). But the () has to be there anyway.

If you have worked for a while you might have things you need to save, do that by clicking on

File > Save

RStudio has a nice recall feature, using the up and down arrow keys. Also, clicking on the History tab shows you the recently run commands. Finally, typing the first three letters of a command in the console and then typing CTRL-^ shows you a list of when you ran commands like this the last times.

R is case-sensitive, so a and A are two different things.

Often during a session you create objects that you need only for a short time. When you no longer need them use **rm** to get rid of them:

```
x <- 10
x^2
```

```
## [1] 100
```

```
rm(x)
```

the **<-** is the *assignment* character in R, it assigns what is on the right to the symbol on the left. (Think of an arrow to the left)

Data Entry

For a few numbers the easiest thing is to just type them in:

```
x <-  c(10, 2, 6, 9)
x
```

```
## [1] 10  2  6  9
```

c() is a function that takes the objects inside the () and **c**ombines them into one single object (a vector).

Data Types in R

the most basic type of data in R is a **vector**, simply a list of values.

Say we want the numbers 1.5, 3.6, 5.1 and 4.0 in an R vector called x, then we can type

```
x <- c(1.5, 3.6, 5.1, 4.0)
x
```

```
## [1] 1.5 3.6 5.1 4.0
```

Often the numbers have a structure one can make use of:

```
1:10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
10:1
```

```
##  [1] 10  9  8  7  6  5  4  3  2  1
```

```
1:20*2
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40
```

```
c(1:10, 1:10*2)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10  2  4  6  8 10 12 14 16 18 20
```

Sometimes you need parentheses:

```
n <- 10
1:n-1
```

```
##  [1] 0 1 2 3 4 5 6 7 8 9
```

```
1:(n-1)
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

The *rep* ("repeat") command is very useful:

```
rep(1, 10)
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1
```

```
rep(1:3, 10)
```

```
##  [1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
```

```
rep(1:3, each=3)
```

```
## [1] 1 1 1 2 2 2 3 3 3
```

```
rep(c("A", "B", "C"), c(4,7,3))
```

```
##  [1] "A" "A" "A" "A" "B" "B" "B" "B" "B" "B" "B" "C" "C" "C"
```

what does this do?

```
rep(1:10, 1:10)
```

Commands for Vectors

To find out how many elements a vector has use the *length* command:

```
x <- c(1.4, 5.1, 2.0, 6.8, 3.5, 2.1, 5.6, 3.3, 6.9, 1.1)
length(x)
```

```
## [1] 10
```

The elements of a vector are accessed with the bracket [ ] notation:

```
x[3]
```

```
## [1] 2
```

```
x[1:3]
```

```
## [1] 1.4 5.1 2.0
```

```
x[c(1, 3, 8)]
```

```
## [1] 1.4 2.0 3.3
```

```
x[-3]
```

```
## [1] 1.4 5.1 6.8 3.5 2.1 5.6 3.3 6.9 1.1
```

```
x[-c(1, 2, 5)]
```

```
## [1] 2.0 6.8 2.1 5.6 3.3 6.9 1.1
```

Instead of numbers a vector can also consist of characters (letters, numbers, symbols etc.) These are identified by quotes:

```
c("A", "B", 7, "%")
```

```
## [1] "A" "B" "7" "%"
```

A vector is either numeric or character, but never both (see how the 7 was changed to "7").

You can turn one into the other (if possible) as follows:

```
x <- 1:10
x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
as.character(x)
```

```
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"
```

```
x <- c("1", "5", "10", "-3")
x
```

```
## [1] "1"  "5"  "10" "-3"
```

```
as.numeric(x)
```

```
## [1]  1  5 10 -3
```

A third type of data is logical, with values either TRUE or FALSE.

```
x <- 1:10
x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
x > 4
```

```
##  [1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

these are often used as conditions:

```
x[x>4]
```

```
## [1]  5  6  7  8  9 10
```

This, as we will see shortly, is EXTREMELY useful!

Data Frames

data frames are the basic format for data in R. They are essentially vectors of equal length put together as columns.

A data frame can be created as follows:

```
df <- data.frame(
  Gender=c("M", "M", "F", "F", "F"),
  Age=c(23, 25, 19, 22, 21),
  GPA=c(3.5, 3.7, 2.9, 2.8, 3.1)
)
df
```

```
##   Gender Age GPA
## 1      M  23 3.5
## 2      M  25 3.7
## 3      F  19 2.9
## 4      F  22 2.8
## 5      F  21 3.1
```

Lists

The most general data structures are lists. They are simply a collection of objects. There are no restrictions on what those objects are.

Example

```
lst <- list(
  Gender=c("M", "M", "F", "F", "F"),
  Age=c(23, 25, 19, 22, 21, 26, 34),
  f=function(x) x^2,
  list(A=c(1, 1), B=c("X", "X", "Y"))
)
lst
```

```
## $Gender
## [1] "M" "M" "F" "F" "F"
##
## $Age
## [1] 23 25 19 22 21 26 34
##
## $f
## function (x)
## x^2
##
## [[4]]
## [[4]]$A
## [1] 1 1
##
## [[4]]$B
## [1] "X" "X" "Y"
```

A data frame is a list with an additional requirement, namely that the elements of the list be of equal length.

Case Study: UPR Admissions

consider the **upr** data set . This is the application data for all the students who applied and were accepted to UPR-Mayaguez between 2003 and 2013.

```
dim(upr)
```

```
## [1] 23666    16
```

tells us that there were 23666 applications and that for each student there are 16 pieces of information.

```
colnames(upr)
```

```
##  [1] "ID.Code"        "Year"           "Gender"         "Program.Code"
##  [5] "Highschool.GPA" "Aptitud.Verbal" "Aptitud.Matem"  "Aprov.Ingles"
##  [9] "Aprov.Matem"    "Aprov.Espanol"  "IGS"            "Freshmen.GPA"
## [13] "Graduated"      "Year.Grad."     "Grad..GPA"      "Class.Facultad"
```

shows us the variables

```
head(upr, 3)
```

```
##      ID.Code Year Gender Program.Code Highschool.GPA Aptitud.Verbal
## 1 00C2B4EF77 2005      M          502           3.97            647
## 2 00D66CF1BF 2003      M          502           3.80            597
## 3 00AB6118EB 2004      M         1203           4.00            567
##   Aptitud.Matem Aprov.Ingles Aprov.Matem Aprov.Espanol IGS Freshmen.GPA
## 1           621          626         672           551 342         3.67
## 2           726          618         718           575 343         2.75
## 3           691          424         616           609 342         3.62
##   Graduated Year.Grad. Grad..GPA Class.Facultad
## 1        Si       2012      3.33           INGE
## 2        No         NA        NA           INGE
## 3        No         NA        NA       CIENCIAS
```

shows us the first three cases.

Let's say we want to find the number of males and females. We can use the table command for that:

```
table(Gender)
```

```
## Error: object 'Gender' not found
```

What happened? Right now R does not know what Gender is because it is "hidden" inside the upr data set. Think of **upr** as a box that is currently closed, so R can't look inside and see the column names. We need to open the box first:

```
attach(upr)
table(Gender)
```

```
## Gender
##     F     M
## 11487 12179
```

there is also a detach command to undo an attach, but this is not usually needed because the attach goes away when you close R.

Note: you need to attach a data frame only once in each session working with R.

Note: Say you are working first with a data set "students 2016" which has a column called Gender, and you attached it. Later (but in the same R session) you start working with a data set "students 2017" which also has a column called Gender, and you are attaching this one as well. If you use Gender now it will be from "students 2017".

Subsetting of Data Frames

Consider the following data frame (not a real data set):

```
students
```

```
##    Age GPA Gender
```

```
## 1   22 3.1   Male
## 2   23 3.2   Male
## 3   20 2.1   Male
## 4   22 2.1   Male
## 5   21 2.3 Female
## 6   21 2.9   Male
## 7   18 2.3 Female
## 8   22 3.9   Male
## 9   21 2.6 Female
## 10  18 3.2 Female
```

Here each single piece of data is identified by its row number and its column number. So for example in row 2, column 2 we have "3.2", in row 6, column 3 we have "Male".

As with the vectors before we can use the [ ] notation to access pieces of a data frame, but now we need to give it both the row and the column number, separated by a ,:

```
students[6, 3]
```

```
## [1] "Male"
```

As before we can pick more than one piece:

```
students[1:5, 3]
```

```
## [1] "Male"   "Male"   "Male"   "Male"   "Female"
```

```
students[1:5, 1:2]
```

```
##   Age GPA
## 1  22 3.1
## 2  23 3.2
## 3  20 2.1
## 4  22 2.1
## 5  21 2.3
```

```
students[-c(1:5), 3]
```

```
## [1] "Male"   "Female" "Male"   "Female" "Female"
```

```
students[1, ]
```

```
##   Age GPA Gender
## 1  22 3.1   Male
```

```
students[, 2]
```

```
##  [1] 3.1 3.2 2.1 2.1 2.3 2.9 2.3 3.9 2.6 3.2
```

```
students[, -3]
```

```
##    Age GPA
## 1   22 3.1
## 2   23 3.2
```

```
## 3   20 2.1
## 4   22 2.1
## 5   21 2.3
## 6   21 2.9
## 7   18 2.3
## 8   22 3.9
## 9   21 2.6
## 10  18 3.2
```

another way of subsetting a data frame is by using the $ notations:

```
students$Gender
```

```
##  [1] "Male"   "Male"   "Male"   "Male"   "Female" "Male"   "Female"
##  [8] "Male"   "Female" "Female"
```

Subsetting of Lists

The double bracket and the $ notation also work for lists:

Example

```
lst <- list(
  Gender=c("M", "M", "F", "F", "F"),
  Age=c(23, 25, 19, 22, 21, 26, 34),
  f=function(x) x^2,
  list(A=c(1, 1), B=c("X", "X", "Y"))
)
lst[[4]][[2]]
```

```
## [1] "X" "X" "Y"
```

```
lst$Gender
```

```
## [1] "M" "M" "F" "F" "F"
```

Vector Arithmetic

R allows us to apply any mathematical functions to a whole vector:

```
x <- 1:10
2*x
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

```
x^2
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

```
log(x)
```

```
##  [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101
##  [8] 2.0794415 2.1972246 2.3025851
```

```
sum(x)
```

```
## [1] 55
```

```
y <- 21:30
```

```
x+y
```

```
##  [1] 22 24 26 28 30 32 34 36 38 40
```

```
x^2+y^2
```

```
##  [1]  442  488  538  592  650  712  778  848  922 1000
```

```
mean(x+y)
```

```
## [1] 31
```

Let's try something strange:

```
c(1, 2, 3) + c(1, 2, 3, 4)
```

```
## [1] 2 4 6 5
```

so R notices that we are trying to add a vector of length 3 to a vector of length 4. This should not work, but it actually does!

When it runs out of values in the first vector, R simply starts all over again.

In general this is more likely a mistake by you, check that this is what you really wanted to do!

apply

A very useful routine in R is *apply*, and its brothers.

Let's say we have the following matrix:

```
Age <- matrix(sample(20:30, size=100, replace=TRUE), 10, 10)
Age[1:5, 1:5]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   20   27   27   20   29
## [2,]   25   23   26   25   27
## [3,]   25   26   20   21   24
## [4,]   25   30   23   22   20
## [5,]   23   26   30   30   26
```

and we want to find the sums of the ages in each column. Easy:

```
sum(Age[, 1])
```

```
## [1] 249
```

```
sum(Age[, 2])
```

```
## [1] 263
```

...

```
sum(Age[, 10])
```

```
## [1] 269
```

or much easier

```
apply(Age, 2, sum)
```

```
##  [1] 249 263 252 226 251 248 271 252 271 269
```

There are a number of apply routines for different data formats.

Case Study: upr admissions

Let's say we want to find the mean Highschool GPA:

```
mean(Highschool.GPA)
```

```
## [1] 3.65861
```

But what if we want to do this for each year separately? Notice that *apply* doesn't work here because the Years are not in separated columns. Instead we can use

```
tapply(Highschool.GPA, Year, mean)
```

```
##     2003     2004     2005     2006     2007     2008     2009     2010
## 3.646627 3.642484 3.652774 3.654729 3.628072 3.648552 3.642946 3.665298
##     2011     2012     2013
## 3.685485 3.695046 3.710843
```

R Basics II - Writing Functions

General Information

In R/RStudio you have several ways to write your own functions:

- In the R console type

```
myfun <- function(x) {
  out <- x^2
  out
}
```

- RStudio: click on File > New File > R Script. A new empty window pops up. Type fun, hit enter, and the following text appears:

name <- function(variables) {

}

change the name to *myfun*, save the file as myfun.R with File > Save. Now type in the code. When done click the Source button.

- fix: In the R console run

```
fix(myfun)
```

now a window with an editor pops up and you can type in the code. When you are done click on Save. If there is some syntax error DON'T run fix again, instead run

```
myfun <- edit()
```

myfun will exist only until you close R/RStudio unless you save the project file.

- Open any code editor outside of RStudio, type in the code, save it as myfun.R, go to the console and run

```
source('../some.folder/myfun.R')
```

Which of these is best? In large part that depends on your preferences. In my case, if I expect to need that function just for a bit I use the fix option. If I expect to need that function again later I start with the first method, but likely soon open the .R file outside RStudio because most code editors have many useful features not available in RStudio.

If *myfun* is open in RStudio there are some useful keyboard shortcuts. If the curser is on some line in the RStudio editor you can hit

- CTRL-Enter run current line or section
- CTRL-ALT-B run from beginning to line
- CTRL-Shift-Enter run complete chunk
- CTRL-Shift-P rerun previous

Testing

As always you can test whether an object is a function:

```
x <- 1
f <- function(x) x
is.function(x)
```

```
## [1] FALSE
```

```
is.function(f)
```

```
## [1] TRUE
```

Arguments

There are several ways to specify arguments in a function:

```
calc.power <- function(x, y, n=2) x^n + y^n
```

here n has a *default value*, x and y do not.

if the arguments are not named they are matched in order:

```
calc.power(2, 3)
```

```
## [1] 13
```

If an argument does not have a default it can be tested for

```
f <- function(first, second) {
  if(!missing(second))
      out <- first + second
  else out <- first
  out
}
f(1)
```

```
## [1] 1
```

```
f(1, s=3)
```

```
## [1] 4
```

There is a special argument ..., used to pass arguments on to other functions:

```
f <- function(x, which, ...) {
  f1 <- function(x, mult) mult*x
  f2 <- function(x, pow) x^pow
  if(which==1)
    out <- f1(x, ...)
  else
    out <- f2(x, ...)
  out
}
f(1:3, 1, mult=2)
```

```
## [1] 2 4 6
```

```
f(1:3, 2, pow=3)
```

```
## [1]  1  8 27
```

This is one of the most useful programming structures in R!

Note this example also shows that in R functions can call other functions. In many computer programs there are so called *sub-routines*, in R this concept does not exist, functions are just functions.

Return Values

A function can either return nothing or exactly one thing. It will automatically return the last object evaluated:

```
f <- function(x) {
  x^2
}
f(1:3)
```

```
## [1] 1 4 9
```

however, it is better programming style to have an explicit return object:

```
f <- function(x) {
  out <- x^2
  out
}
f(1:3)
```

```
## [1] 1 4 9
```

There is another way to specify what is returned:

```
f <- function(x) {
  return(x^2)
}
f(1:3)
```

```
## [1] 1 4 9
```

but this is usually used to return something early in the program:

```
f <- function(x) {
  if(!any(is.numeric(x)))
    return("Works only for numeric!")
  out <- sum(x^2)
  out
}
f(1:3)
```

```
## [1] 14
```

```
f(letters[1:3])
```

```
## [1] "Works only for numeric!"
```

If you want to return more than one item use a list:

```
f <- function(x) {
  sq <- x^2
  sm <- sum(x)
  list(sq=sq, sum=sm)
```

```
}
f(1:3)
```

```
## $sq
## [1] 1 4 9
##
## $sum
## [1] 6
```

Basic Programmming Structures in R

R has all the standard programming structures:

Conditionals (if-else)

```
f <- function(x) {
  if(x>0) y <- log(x)
  else y <- NA
  y
}
f(c(2, -2))
```

```
## [1] 0.6931472       NaN
```

A useful variation on the *if* statement is *switch*:

```
centre <- function(x, type) {
  switch(type,
         mean = mean(x),
         median = median(x),
         trimmed = mean(x, trim = .1))
}
x <- rcauchy(10)
centre(x, "mean")
```

```
## [1] -1.063426
```

```
centre(x, "median")
```

```
## [1] 0.4230036
```

```
centre(x, "trimmed")
```

```
## [1] -0.06293897
```

special R construct: *ifelse*

```
x <- sample(1:10, size=7, replace = TRUE)
x
```

```
## [1] 8 1 3 6 2 5 8
```

```
ifelse(x<5, "Yes", "No")
```

```
## [1] "No"  "Yes" "Yes" "No"  "Yes" "No"  "No"
```

Loops

there are two standard loops in R:

- for loop

```
y <- rep(0, 10)
for(i in 1:10) y[i] <- i*(i+1)/2
y
```

```
##  [1]  1  3  6 10 15 21 28 36 45 55
```

sometimes we don't know the length of y ahead of time, then we can use

```
for(i in seq_along(y)) y[i] <- i*(i+1)/2
y
```

```
##  [1]  1  3  6 10 15 21 28 36 45 55
```

If there is more than one statement inside a loop use curly braces:

```
for(i in seq_along(y)) {
  y[i] <- i*(i+1)/2
  if(y[i]>40) y[i] <- (-1)
}
y
```

```
##  [1]  1  3  6 10 15 21 28 36 -1 -1
```

You can nest loops:

```
A <- matrix(0, 4, 4)
for(i in 1:4) {
  for(j in 1:4)
    A[i, j] <- i*j
}
A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    2    4    6    8
## [3,]    3    6    9   12
## [4,]    4    8   12   16
```

- repeat loop

```
k <- 0
repeat {
  k <- k+1
```

```
  x <- sample(1:6, size=3, replace=TRUE)
  if(length(table(x))==1) break
}
k
```

```
## [1] 1
```

Notice that a repeat loop could in principle run forever. I usually include a counter that ensures the loop will eventually stop:

```
k <- 0
counter <- 0
repeat {
  k <- k+1
  counter <- counter+1
  x <- sample(1:6, size=3, replace=TRUE)
  if(length(table(x))==1 | counter>1000) break
}
k
```

```
## [1] 15
```

Random Numbers and Simulation

Random Numbers

Everything starts with generating X_1, X_2, .. iid U[0,1]. These are simply called random numbers. There are some ways to get these:

- random number tables
- numbers taken from things like the exact (computer) time
- quantum random number generators
- ...

The R package *random* has the routine *randomNumbers* which gets random numbers from a web site which generates them based on (truly random) atmospheric phenomena.

```
require(random)
randomNumbers(20, 0, 100)
```

```
##      V1 V2 V3 V4 V5
## [1,] 33 54 51 92  2
## [2,] 11 19  0 15 67
## [3,] 17 86  8 18 70
## [4,] 85 42 30 56 11
```

Most of the time we will use *pseudo-random numbers*, that is numbers that are not actually random but are indistinguishable from those. In R this is done with

```
runif(5)
```

```
## [1] 0.2349160 0.6613743 0.1086612 0.1734862 0.7146776
```

```
runif(5, 100, 300)
```

```
## [1] 204.2313 187.1649 270.9583 137.8778 129.0428
```

Standard Probability Distributions

Not surprisingly many standard distributions are part of base R. For each the format is

- dname = density
- pname = cumulative distribution function
- rname = random generation
- qname = quantile function

Note we will use the term *density* for both discrete and continuous random variable.

Example Poisson distribution

We have $X \sim \text{Pois}(\lambda)$ if

$$P(X = x) = \frac{\lambda^x}{x!} e^{-\lambda}, \; x = 0, 1, ...$$

```
# density
dpois(c(0, 8, 12, 20), lambda=10)
```

```
## [1] 4.539993e-05 1.125990e-01 9.478033e-02 1.866081e-03
```

```
10^c(0, 8, 12, 20)/factorial(c(0, 8, 12, 20))*exp(-10)
```

```
## [1] 4.539993e-05 1.125990e-01 9.478033e-02 1.866081e-03
```

```
# cumulative distribution function
ppois(c(0, 8, 12, 20), 10)
```

```
## [1] 4.539993e-05 3.328197e-01 7.915565e-01 9.984117e-01
```

```
# random generation
rpois(5, 10)
```

```
## [1] 12  7  9 12 11
```

```r
# quantiles
qpois(1:4/5, 10)
```

```
## [1]  7  9 11 13
```

Here is a list of the distributions included with base R:

- beta distribution: dbeta.
- binomial (including Bernoulli) distribution: dbinom.
- Cauchy distribution: dcauchy.
- chi-squared distribution: dchisq.
- exponential distribution: dexp.
- F distribution: df.
- gamma distribution: dgamma.
- geometric distribution: dgeom.
- hypergeometric distribution: dhyper.
- log-normal distribution: dlnorm.
- multinomial distribution: dmultinom.
- negative binomial distribution: dnbinom.
- normal distribution: dnorm.
- Poisson distribution: dpois.
- Student's t distribution: dt.
- uniform distribution: dunif.
- Weibull distribution: dweibull.

With some of these a bit of caution is needed. For example, the usual textbook definition of the geometric random variable is the number of tries in a sequence of independent Bernoulli trials until a success. This means that the density is defined as

$$P(X = x) = p(1-p)^{x-1},\ x = 1, 2, ..$$

R however defines it as the number of failures until the first success, and so it uses

$$P(X^* = x) = \text{dgeom}(x, p) = p(1-p)^{x},\ x = 0, 1, 2, ..$$

Of course this is easy to fix. If you want to generate the "usual" geometric do

```r
x <- rgeom(10, 0.4) + 1
x
```

```
##  [1]  2  1  1  9  1  2 11  1 12  3
```

if you want to find the probabilities or cdf:

```
round(dgeom(x-1, 0.4), 4)
```

```
##  [1] 0.2400 0.4000 0.4000 0.0067 0.4000 0.2400 0.0024 0.4000 0.0015 0.1440
```

```
round(0.4*(1-0.4)^(x-1), 4)
```

```
##  [1] 0.2400 0.4000 0.4000 0.0067 0.4000 0.2400 0.0024 0.4000 0.0015 0.1440
```

Another example is the Gamma random variable. Here most textbooks use the definition

$$f(x;\alpha,\beta) = \frac{1}{\Gamma(\alpha)\beta^{\alpha}} x^{\alpha-1} e^{-x/\beta}$$

but R uses

$$f^*(x;\alpha,\beta) = \frac{\beta^{\alpha}}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}$$

```
dgamma(1.2, 0.5, 2)
```

```
## [1] 0.06607584
```

```
2^0.5/gamma(0.5)*1.2^(0.5-1)*exp(-2*1.2)
```

```
## [1] 0.06607584
```

Again, it is easy to *re-parametrize*:

```
dgamma(1.2, 0.5, 1/(1/2))
```

```
## [1] 0.06607584
```

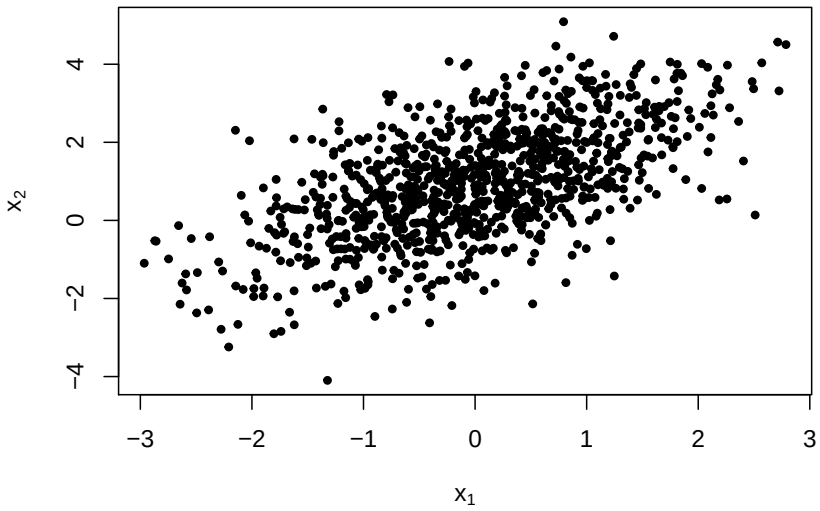
Other Variates

if you need to generate random variates from a distribution that is not part of base R you should first try to find a package that includes it.

Example multivariate normal

there are actually several packages, the most commonly used one is *mvtnorm*

```
library(mvtnorm)
x <- rmvnorm(1000,
             mean = c(0, 1),
             sigma = matrix(c(1, 0.8, 0.8, 2), 2, 2))
plot(x,
     pch=20,
     xlab = expression(x[1]),
     ylab = expression(x[2]))
```



sigma is the variance-covariance matrix, so in the above we have

$$\rho = Cor(X, Y) = \frac{Cov(X, Y)}{\sqrt{Var(X)\, Var(Y)}} = \frac{0.8}{\sqrt{1 * 2}} = 0.566$$

```
round(c(var(x[, 1]),
        var(x[, 2]),
        cor(x[, 1], x[, 2])), 3)
```

```
## [1] 1.033 2.020 0.587
```

Simulation

In a *simulation* we attempt to generate data just like what we might see in a real-live experiment, except that we control all the details. The we carry out some calculations on that artificial data, and we repeat this many times. Here are some examples:

Example

When rolling a fair die 5 times, what is the probability of no sixes? Of no more than one six?

```
B <- 10000 # number of simulation runs
num.sixes <- rep(0, B) # to store results
```

```r
for(i in 1:B) {
  x <- sample(1:6, size=5, replace=TRUE) # roll 5 dice
  num.sixes[i] <- length(x[x==6]) # how many sixes?
}
# Probability of no sixes
length(num.sixes[num.sixes==0])/B
```

```
## [1] 0.4015
```

```r
# Probability of no more than one sixes
length(num.sixes[num.sixes<=1])/B
```

```
## [1] 0.8041
```

Of course one can do this also analytically:

$$
\begin{aligned}
&P(\text{no sixes}) = P(\text{no six on any die}) = \\
&P(\text{no six on first die } \cap \text{ .. } \cap \text{ no six on fifth die}) = \\
&\prod_{i=1}^{5} P(\text{no six on } i^{th}\text{die}) = (\frac{5}{6})^5 = 0.402
\end{aligned}
$$

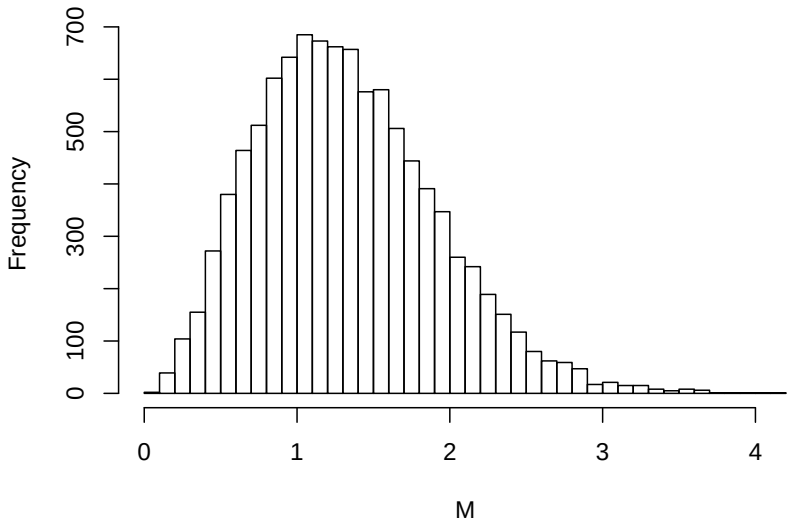
but already the second one is a bit harder to do analytically but not via simulation.

One issue we have with a simulation is the *simulation error*, namely that the simulation will always yield a slightly different answer.

Example

Say we have $X, Y, Z \sim N(0,1)$ and set $M = \max\{|X|, |Y|, |Z|\}$. What is the mean and standard deviation of M?

```r
B <- 10000
x <- matrix(abs(rnorm(3*B)), ncol=3)
M <- apply(x, 1, max)
hist(M, 50, main="")
```




```
round(c(mean(M), sd(M)), 3)
```

```
## [1] 1.329 0.587
```

Example Symmetric Random Walk in R

Let $P(Z_i = -1) = P(Z_i = 1) = \frac{1}{2}$ and $X_n = \sum_{i=1}^{n} Z_i$. Let A>0 some integer. Let's write a routine that finds the median number of steps the walk takes until it hits either -A or A.

One issue with simulations of *stochastic processes* is that in general they are very slow. Here I will use a little trick: I will generate part of the process, and then check whether the event of interest has already happened.

```
first.hit <- function(A) {
  B <- 10000
  num.steps <- rep(0, B)
  for(i in 1:B) {
    x <- 0
    k <- 0
    repeat {
      z <- sample(c(-1, 1), size=1000, replace=TRUE)
      x <- x + cumsum(z)
      if(max(abs(x))>=A) break
      x <- x[1000]
      k <- k+1000
    }
    k <- k+seq_along(x)[abs(x)>=A][1]
    num.steps[i] <- k
  }
```

```
  median(num.steps)
}
first.hit(100)
```

```
## [1] 7668
```

Example

The following you find in any basic stats course: A $100(1-\alpha)\%$ confidence interval for the success probability in a sequence of n Bernoulli trials is given by

$$\hat{p} \pm z_{\alpha/2}\sqrt{\hat{p}(1-\hat{p})/n}$$

where $\hat{p}$ is the proportion of successes. This method is supposed to work if n is at least 50.

Let's do a simulation to test this method.

```
ci.prop.sim <- function(p, n, conf.level=95, B=1e4) {
  z <- qnorm(1-(1-conf.level/100)/2)
  bad <- 0
  for(i in 1:B) {
    x <- sample(0:1, size=n, replace = TRUE, prob=c(1-p, p))
    phat <- sum(x)/n
    if(phat - z*sqrt(phat*(1-phat)/n)>p) bad<-bad+1
    if(phat + z*sqrt(phat*(1-phat)/n)<p) bad<-bad+1
  }
  bad/B
}
```

```
ci.prop.sim(0.5, 100)
```

```
## [1] 0.0557
```

and that is not so bad.

But

```
ci.prop.sim(0.1, 50)
```

```
## [1] 0.1258
```

and that is very bad indeed!

Soon we will consider a method that is guaranteed to give intervals with correct coverage, no matter what p and n are.

Example: Simultaneous Inference

There is a famous (infamous?) case of three psychiatrists who studied a sample of schizophrenic persons and a sample of non schizophrenic persons. They measured 77 variables for each subject - religion, family background, childhood experiences etc. Their goal was to discover

what distinguishes persons who later become schizophrenic. Using their data they ran 77 hypothesis tests of the significance of the differences between the two groups of subjects, and found 2 significant at the 2% level.They immediately published their findings.

What's wrong here? Remember, if you run a hypothesis test at the 2% level you expect to reject the null hypothesis of no relationship 2% of the time, but 2% of 77 is about 1 or 2, so just by random fluctuations they could (should?) have rejected that many null hypotheses! This is not to say that the variables they found to be different between the two groups were not really different, only that their method did not proof that.

In its general form this is known as the problem of simultaneous inference and is one of the most difficult issues in Statistics today. One general solution of used is called *Bonferroni's method*. The idea is the following:

Let's assume we carry out k hypothesis tests. All tests are done at α significance level and all the tests are all independent. Then the probability that at least one test rejects the null hypothesis although all null are true is given by

$$\begin{aligned}
&\alpha^* = P(\text{at least one null rejected} \mid \text{all null true}) = \\
&1 - P(\text{none of the nulls rejected} \mid \text{all null true}) = \\
&1 - \prod_{i=1}^{k} P(\text{ith null is not rejected} \mid \text{ith null true}) = \\
&1 - \prod_{i=1}^{k} [1 - P(\text{ith null is rejected} \mid \text{ith null true})] = \\
&1 - [1-\alpha]^k = \\
&1 - \left[1 - k\alpha + \binom{k}{2}\alpha^2 - +..\right] \approx k\alpha
\end{aligned}$$

so if each individual test is done with α/k, the *family-wise* error rate is the desired one.

Let's do a simulation to see how that would work in the case of our psychiatrists experiments. There many details we don't know, so we have to make them up a bit:

```
sim.shiz <- function(m, n=50, B=1e3) {
  counter <- matrix(0, B, 2)
  for(i in 1:B) {
    for(j in 1:77) {
      pval <- t.test(rnorm(n), rnorm(n))$p.value
      if(pval<0.02) counter[i, 1]<-1
      if(pval<0.05/m) counter[i, 2]<-1
    }
  }
  apply(counter, 2, sum)/B
}
sim.shiz(77)
```

```
## [1] 0.77 0.06
```

This works fine here. The main problem in real life is that it is rarely true that these test are independent, and then all we can say is that the needed α is between α/k and α.

Graphics with ggplot2

A large part of this chapter is taken from various works of Hadley Wickham. Among others The layered grammar of graphics and R for Data Science.

Why ggplot2?

Advantages of ggplot2

- consistent underlying grammar of graphics (Wilkinson, 2005)
- plot specification at a high level of abstraction
- very flexible
- theme system for polishing plot appearance
- mature and complete graphics system
- many users, active mailing list

but really, they are just so much nicer than base R graphs!

Grammar of Graphics

In 2005 Wilkinson, Anand, and Grossman published the book "The Grammar of Graphics". In it they laid out a systematic way to describe any graph in terms of basic building blocks. ggplot2 is an implementation of their ideas.

The use of the word *grammar* seems a bit strange here. The general dictionary meaning of the word grammar is:

the fundamental principles or rules of an art or science

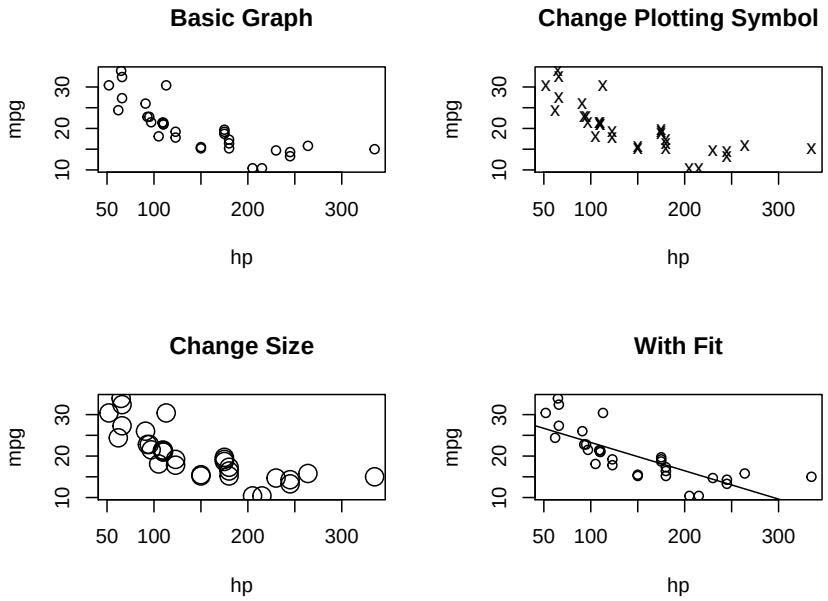
so it is not only about language.

As our running example we will use the *mtcars* data set. It is part of base R and has information on 32 cars:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

Say we want to study the relationship of hp and mpg. So we have two quantitative variables, and therefore the obvious thing to do is a scatterplot. But there are a number of different ways we can do this:

```
attach(mtcars)
par(mfrow=c(2, 2))
plot(hp, mpg, main="Basic Graph")
plot(hp, mpg, pch="x", main="Change Plotting Symbol")
plot(hp, mpg, cex=2, main="Change Size")
plot(hp, mpg, main="With Fit");abline(lm(mpg~hp))
```

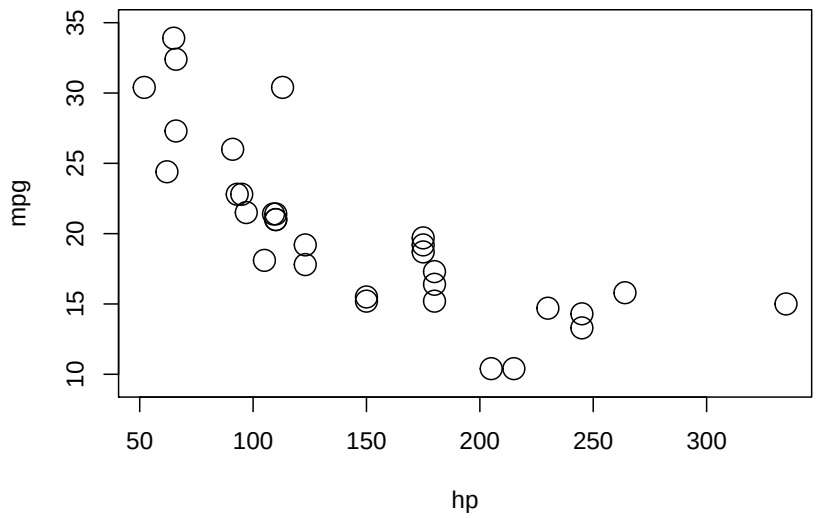


The basic idea of the grammar of graphs is to separate out the parts of the graphs: there is the basic layout, there is the data that goes into it, there is the way in which the data is displayed. Finally there are annotations, here the titles, and other things added, such as a fitted line. In ggplot2 you can always change one of these without worrying how that change effects any of the others.

Take the graph on the lower left. Here I made the plotting symbol bigger (with cex=2). But now the graph doesn't look nice any more, the first and the last circle don't fit into the graph.

The only way to fix this is to start all over again, by making the margins bigger:

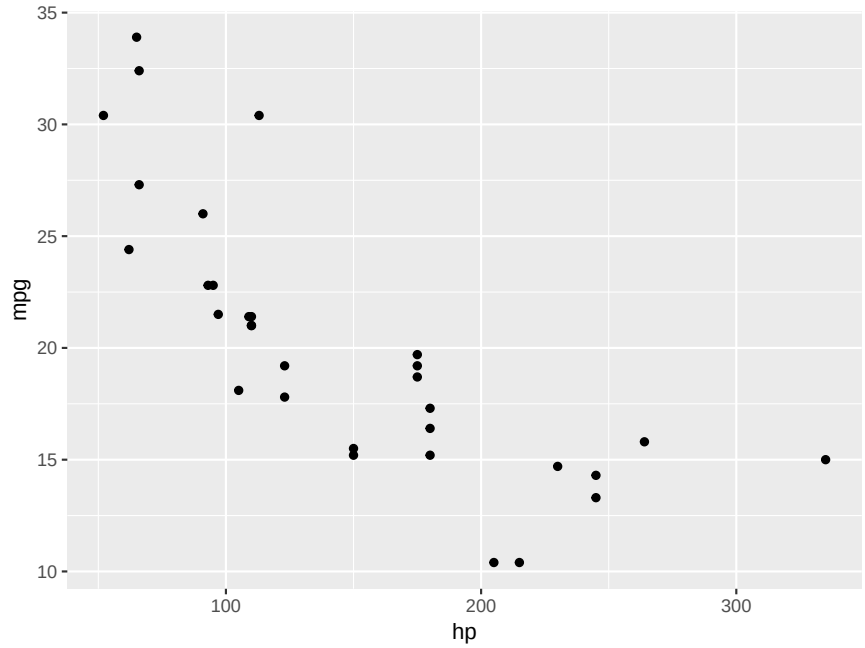
```
plot(hp, mpg, cex=2, ylim=range(mpg)+c(-1, 1))
```



and that is a bit of work because I have to figure out how to change the margins. In ggplot2 that sort of thing is taken care of automatically!

Let's start by recreating the first graph above.

```
ggplot(mtcars, aes(hp, mpg)) +
  geom_point()
```

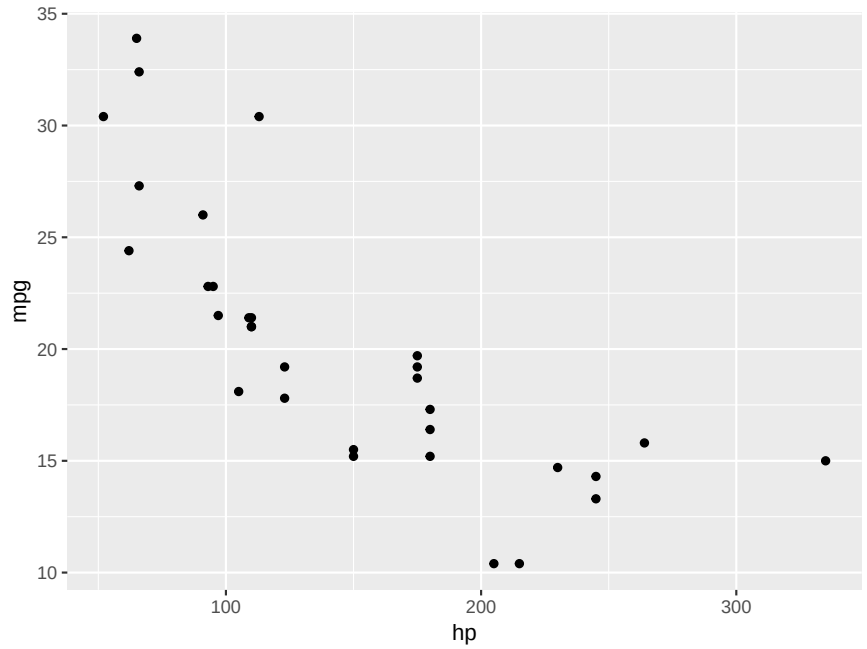


this has the following logic:

- *ggplot* sets up the graph
- it's first argument is the data set (which has to be a dataframe)
- *aes* is the *aestetic mapping*. It connects the data to the graph by specifying which variables go where
- *geom* is the geometric object (circle, square, line) to be used in the graph

Note ggplot2 also has the *qplot* command. This stands for *qick plot*

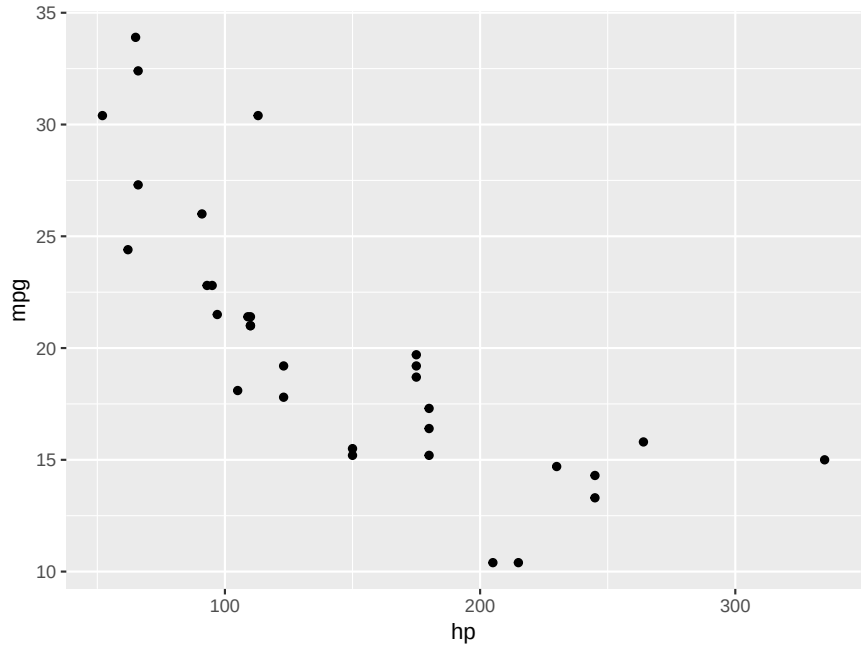
```
qplot(hp, mpg, data=mtcars)
```



This seems much easier at first (and it is) but the qplot command is also quite limited. Very quickly you want to do things that aren't possible with qplot, and so I won't discuss it further here.

Note consider the following variation:

```
ggplot(mtcars) +
  geom_point(aes(hp, mpg))
```

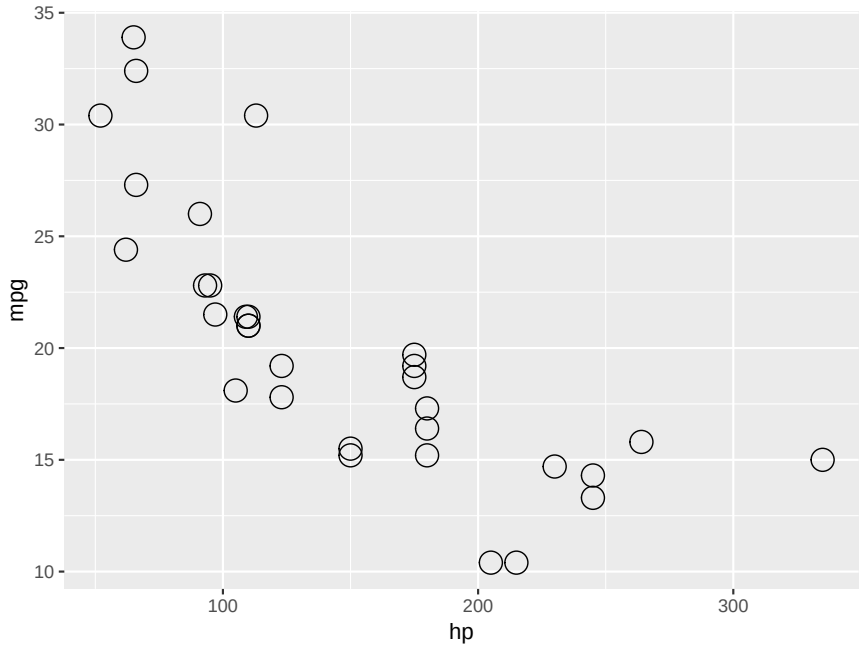


again it seems to do the same thing, but there is a big difference:

- if aes(x, y) is part of ggplot, it applies to all the geom's that come later (unless a different one is specified)
- an aes(x, y) as part of a geom applies only to it.

How about the problem with the graph above, where we had to increase the y margin?

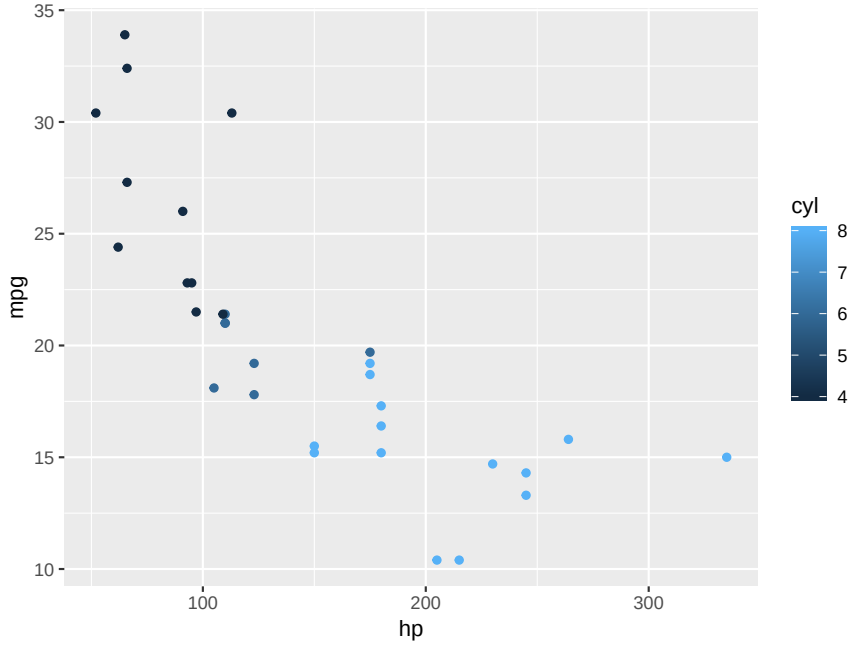
```
ggplot(mtcars, aes(hp, mpg)) +
  geom_point(shape=1, size=5)
```



so we see that here this is done automatically.

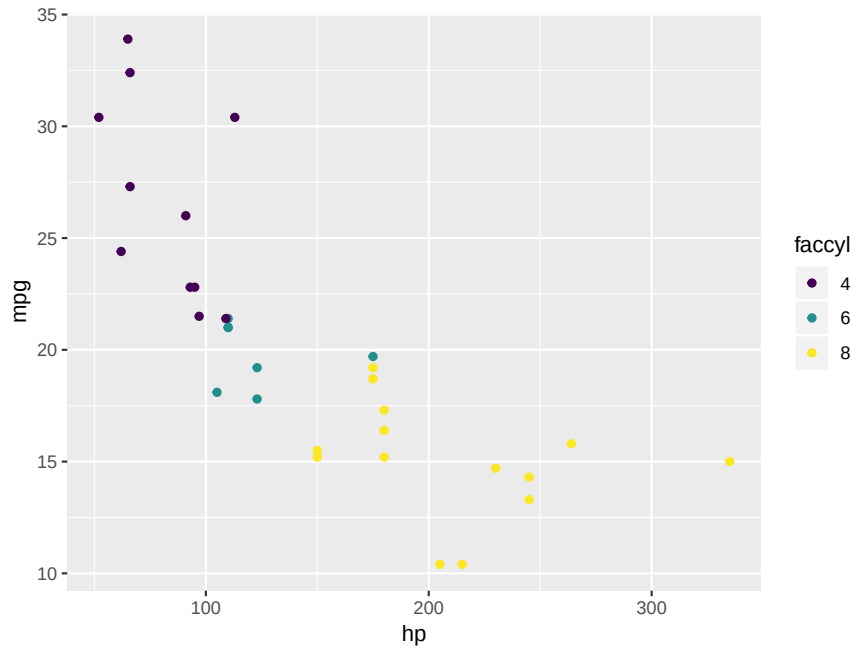
Let's say we want to identify the cars by the number of cylinders:

```
ggplot(mtcars, aes(hp, mpg, color=cyl)) +
  geom_point()
```



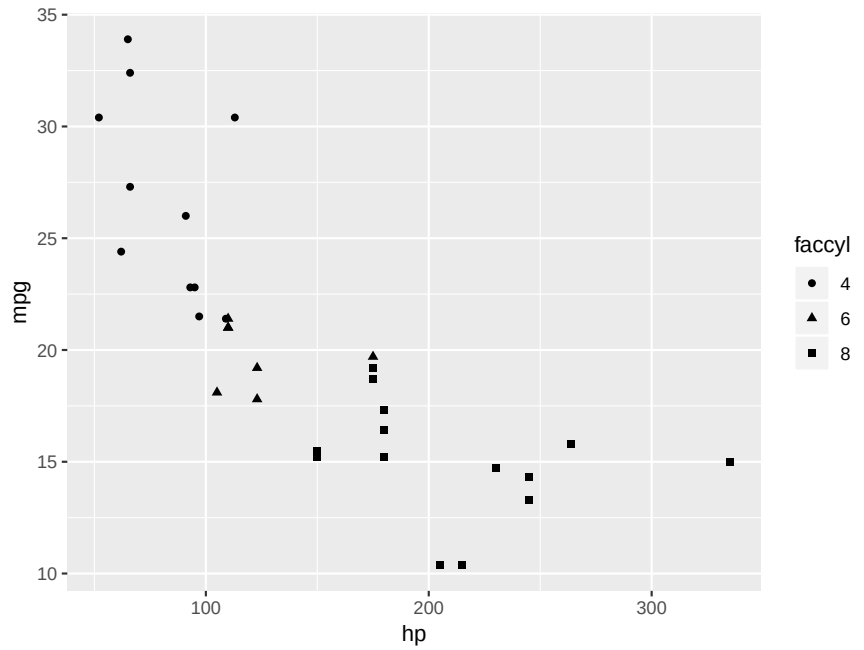
Notice that the legend is a continuous color scale. This is because the variable cyl has values 4, 6, and 8, and so is identified by R as a numeric variable. In reality it is categorical (ever seen a car with 1.7 cylinders?), and so we should change that:

```
mtcars$faccyl <- factor(cyl,
                        levels = c(4, 6, 8),
                        ordered = TRUE)
ggplot(mtcars, aes(hp, mpg, color=faccyl)) +
  geom_point()
```



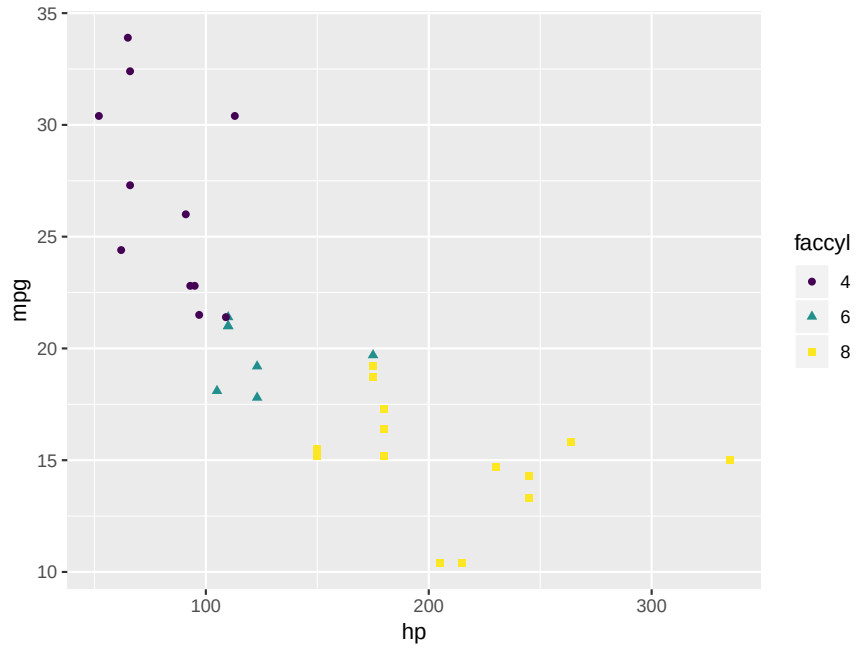
we can also change the shape of the plotting symbols:

```
ggplot(mtcars, aes(hp, mpg, shape=faccyl)) +
  geom_point()
```



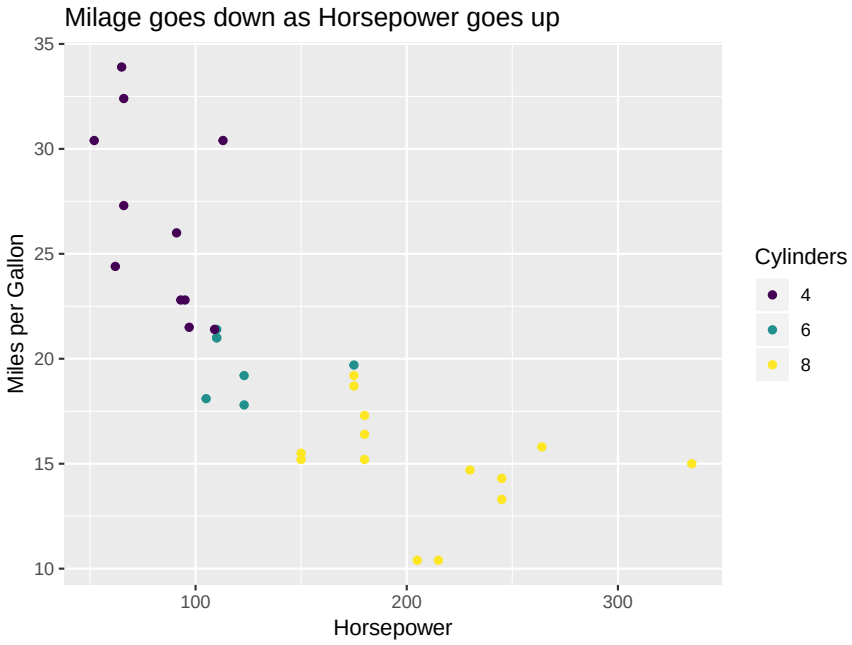
or both:

```
ggplot(mtcars, aes(hp, mpg, shape=faccyl, color=faccyl)) +
  geom_point()
```



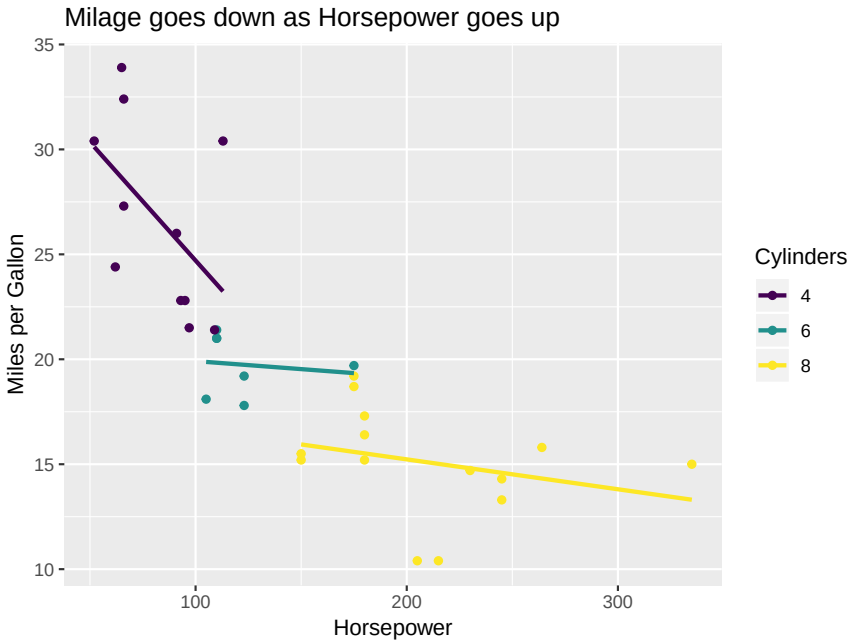
let's pretty up the graph a bit with some labels and a title. We will be playing around with this graph for a while, so I will save some intermediate versions:

```
plt1 <- ggplot(mtcars, aes(hp, mpg, color=faccyl)) +
  geom_point()
plt2 <- plt1 +
  labs(x = "Horsepower",
       y = "Miles per Gallon",
       color = "Cylinders") +
  labs(title = "Milage goes down as Horsepower goes up")
plt2
```



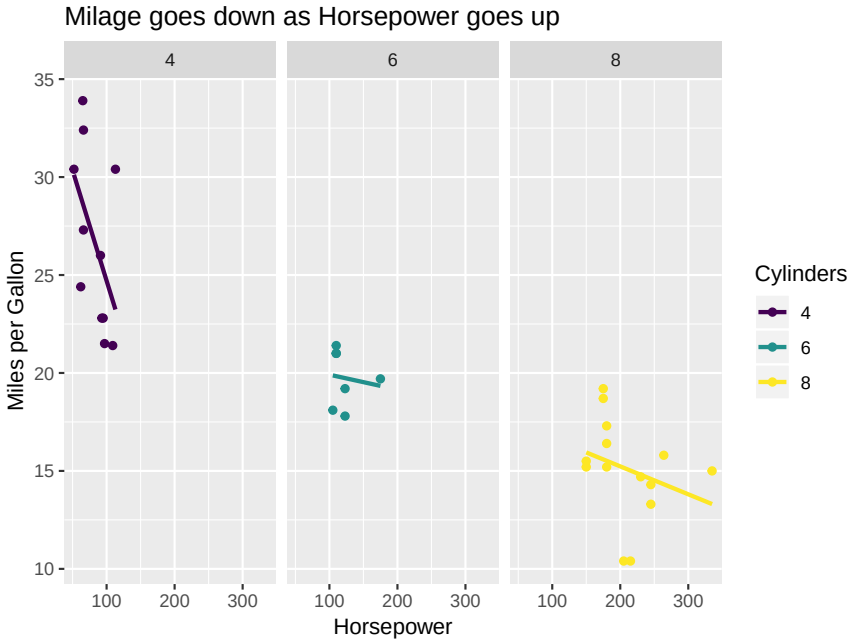
Say we want to add the least squares regression lines for cars with the same number of cylinders:

```
plt3 <- plt2 +
  geom_smooth(method = "lm", se = FALSE)
plt3
```



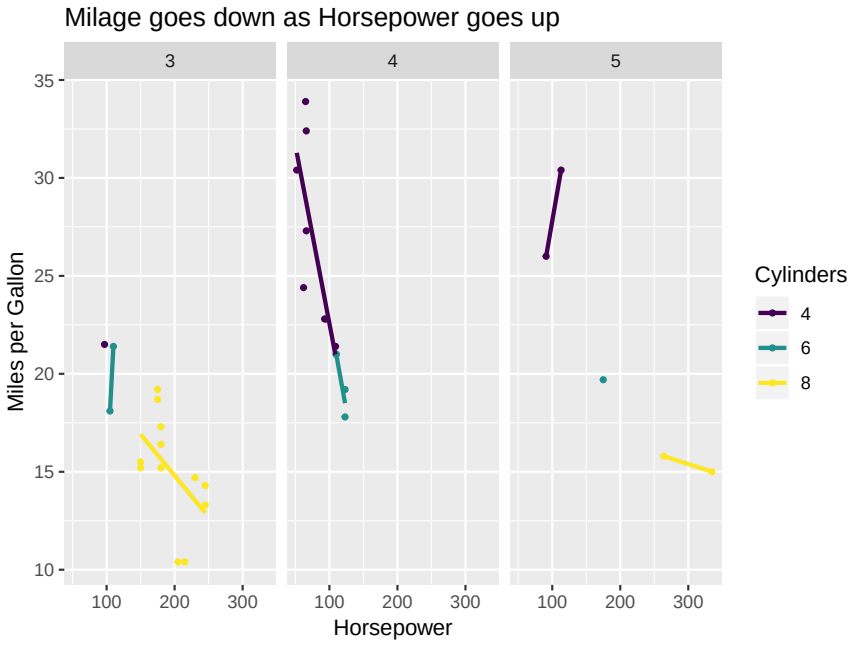
There is another way to include a categorical variable in a scatterplot. The idea is to do several graphs, one for each value of the categorical variable. These are called *facets*:

```
plt3 +
  facet_wrap(~cyl)
```



The use of facets also allows us to include two categorical variables:

```
mtcars$facgear <-
  factor(gear, levels = 3:5, ordered = TRUE)
plt4 <- ggplot(aes(hp, mpg, color=faccyl),
               data = mtcars) +
          geom_point(size = 1)
plt4 <- plt4 +
  facet_wrap(~facgear)
plt4 <- plt4 +
  labs(x = "Horsepower",
       y = "Miles per Gallon",
       color = "Cylinders") +
  labs(title = "Milage goes down as Horsepower goes up")
plt4 <- plt4 +
  geom_smooth(method = "lm", se = FALSE)
plt4
```

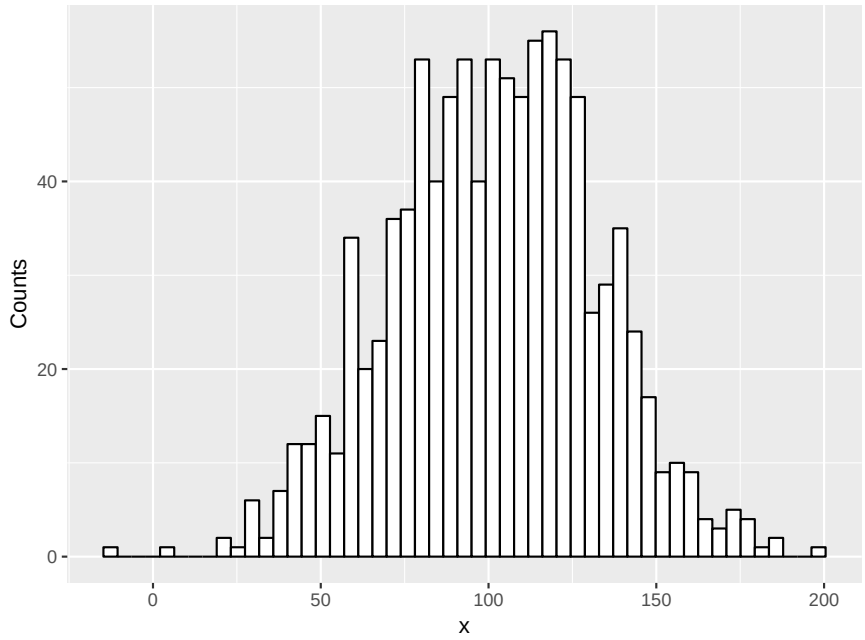


This is almost a bit to much, with just 32 data points there is not really enough for such a split.

Let's see how to use ggplot do a number of basic graphs:

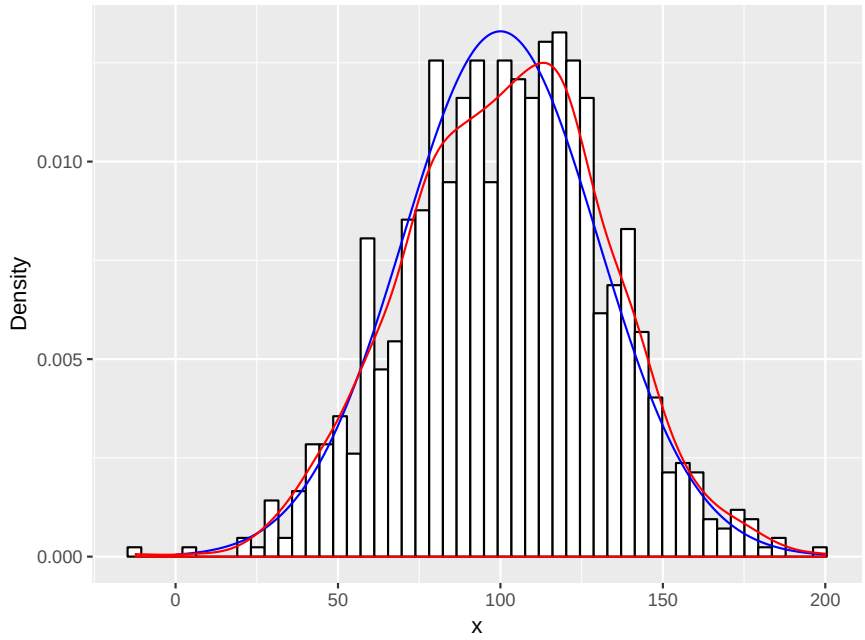
Histograms

```
x <- rnorm(1000, 100, 30)
df3 <- data.frame(x = x)
bw <- diff(range(x))/50 # use about 50 bins
ggplot(df3, aes(x)) +
  geom_histogram(color = "black",
                 fill = "white",
                 binwidth = bw) +
  labs(x = "x", y = "Counts")
```



Often we do histograms scaled to integrate to one. Then we can add the theoretical density and/or a nonparametric density estimate:

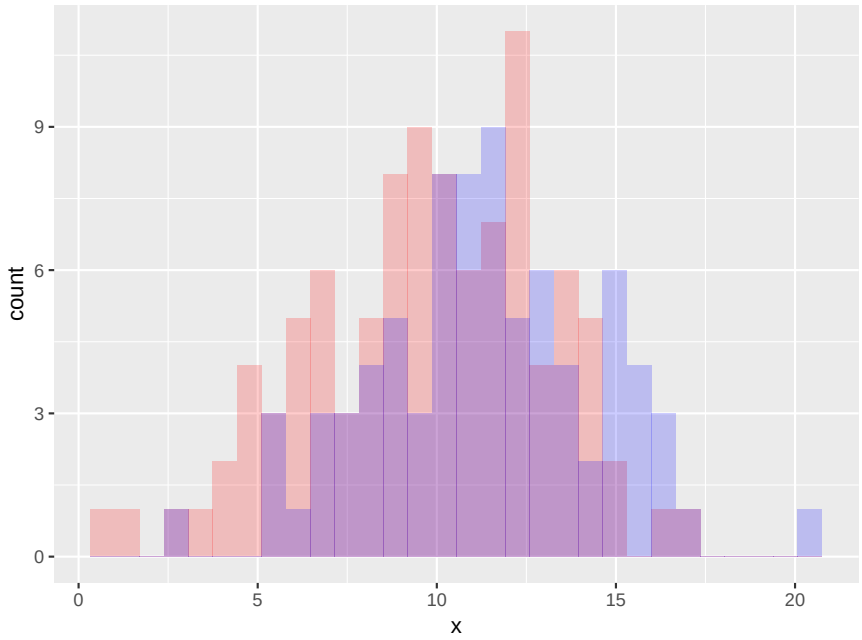
```
x <- seq(0, 200, length=250)
df4 <- data.frame(x=x, y=dnorm(x, 100, 30))
ggplot(df3, aes(x)) +
  geom_histogram(aes(y = ..density..),
        color = "black",
        fill = "white",
        binwidth = bw) +
  labs(x = "x", y = "Density") +
  geom_line(data = df4, aes(x, y),
            colour = "blue") +
  geom_density(color = "red")
```



Notice the red line on the bottom. This should not be there but seems almost impossible to get rid of!

Here is another interesting case: say we have two data sets and we wish to draw the two histograms, one overlaid on the other:

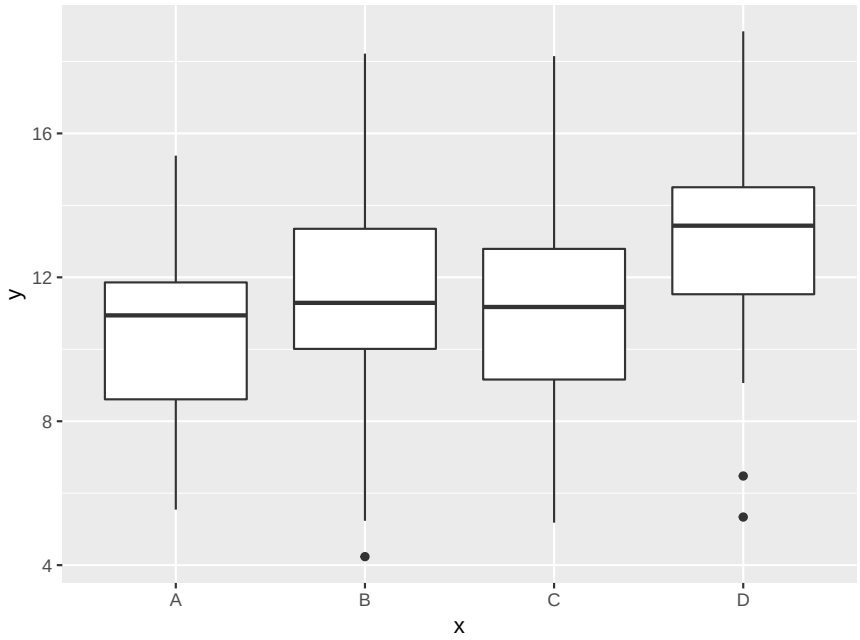
```
df5 <- data.frame(
  x = c(rnorm(100, 10, 3), rnorm(80, 12, 3)),
  y = c(rep(1, 100), rep(2, 80)))
ggplot(df5, aes(x=x)) +
    geom_histogram(data = subset(df5, y == 1),
        fill = "red", alpha = 0.2) +
    geom_histogram(data = subset(df5, y == 2),
        fill = "blue", alpha = 0.2)
```



Notice the use of alpha. In general this "lightens" the color so we can see "behind".

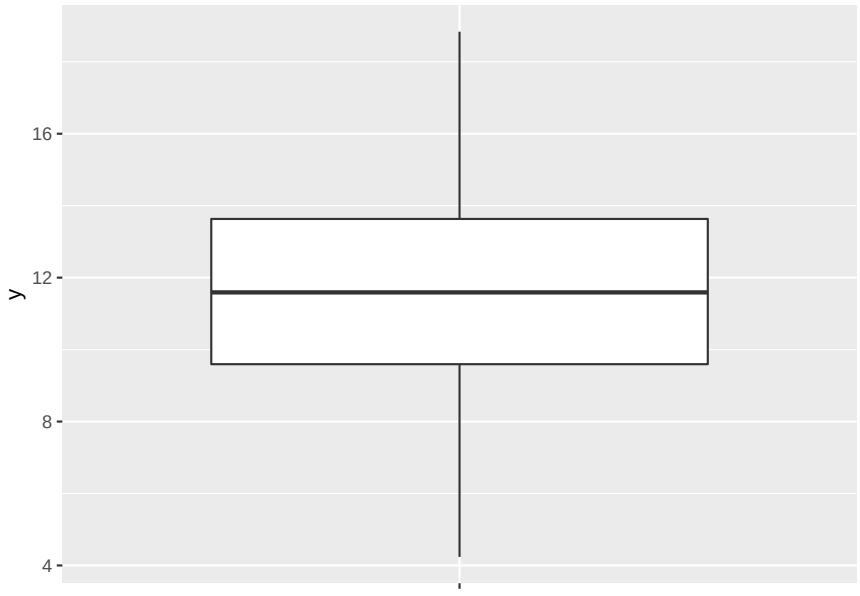
Boxplots

```
y <- rnorm(120, 10, 3)
x <- rep(LETTERS[1:4], each=30)
y[x=="B"] <- y[x=="B"] + rnorm(30, 1)
y[x=="C"] <- y[x=="C"] + rnorm(30, 2)
y[x=="D"] <- y[x=="D"] + rnorm(30, 3)
df6 <- data.frame(x=x, y=y)
ggplot(df6, aes(x, y)) +
  geom_boxplot()
```



strangely enough doing a boxplot without groups takes a bit of a hack. We have to "invent" a categorical variable:

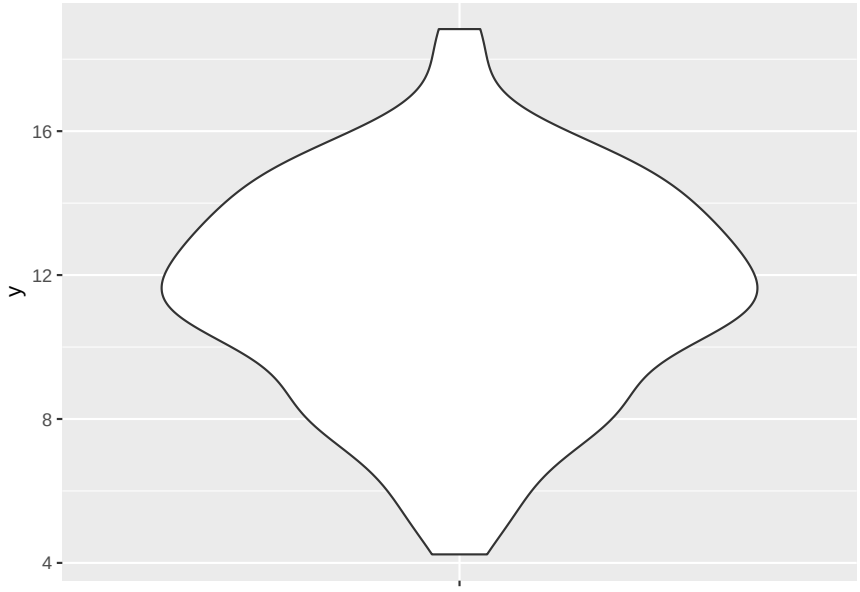
```
ggplot(df6, aes(x="", y)) +
  geom_boxplot() +
  xlab("")
```



There is a modern version of this graph called a violin plot:

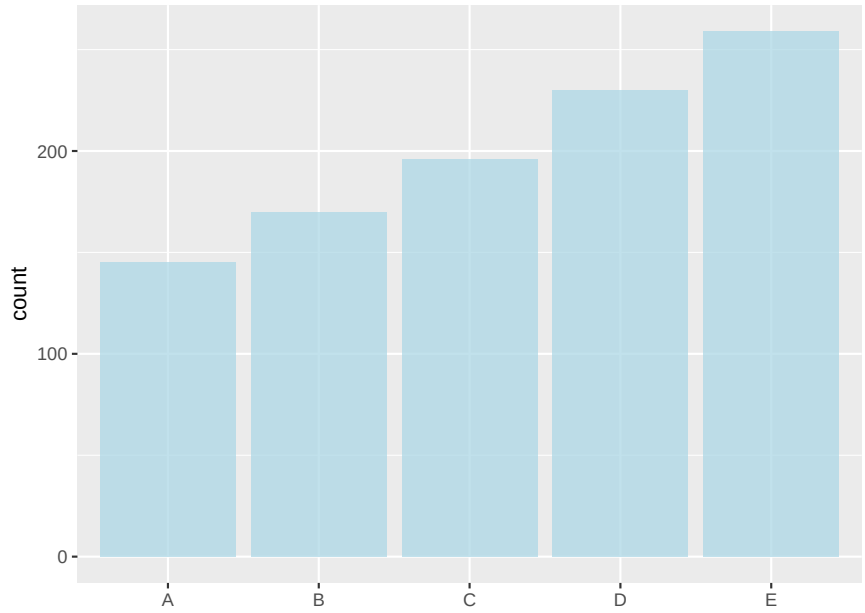
```
ggplot(df6, aes(x="", y)) +
  geom_violin() +
```

```
  xlab("")
```



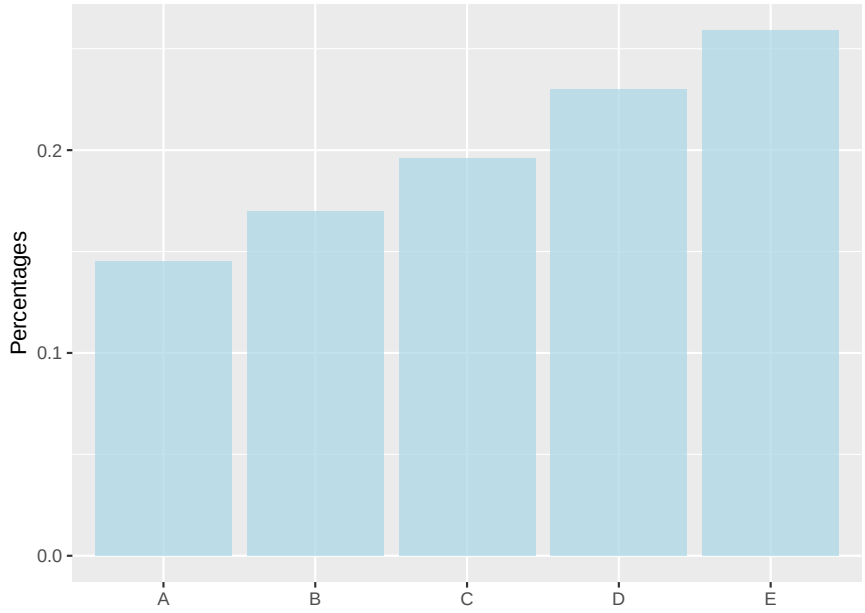
Barcharts

```
x <- sample(LETTERS[1:5],
            size = 1000,
            replace = TRUE,
            prob = 6:10)
df7 <- data.frame(x=x)
ggplot(df7, aes(x)) +
  geom_bar(alpha=0.75, fill="lightblue") +
  xlab("")
```



Say we want to draw the graph based on percentages. Of course we could just calculate them and then do the graph. Here is another way:

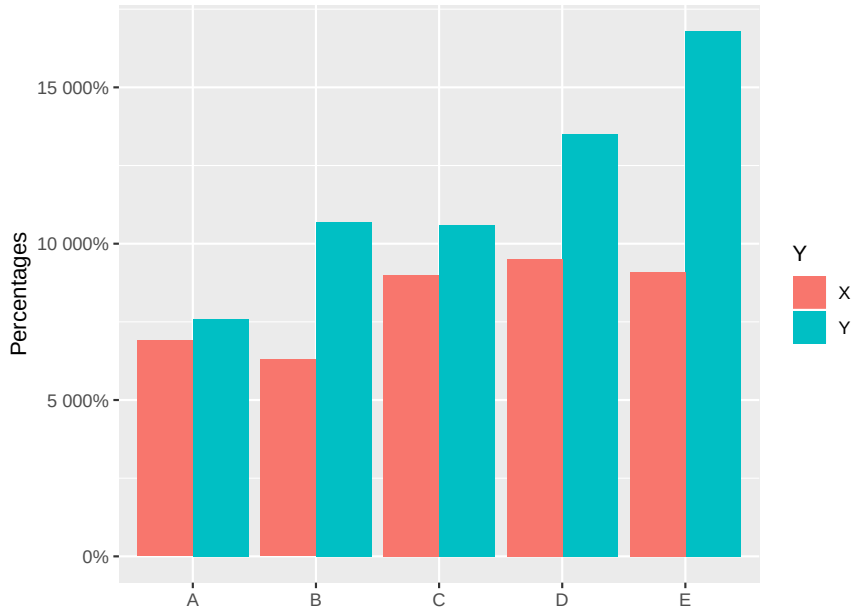
```
ggplot(df7, aes(x=x)) +
  geom_bar(aes(y=(..count..)/sum(..count..)),
      alpha = 0.75,
      fill = "lightblue") +
  labs(x="", y="Percentages")
```



Notice how this works: in geom_bar we use a new aes, but the values in it are calculated from the old data frame.

Finally an example of a contingency table:

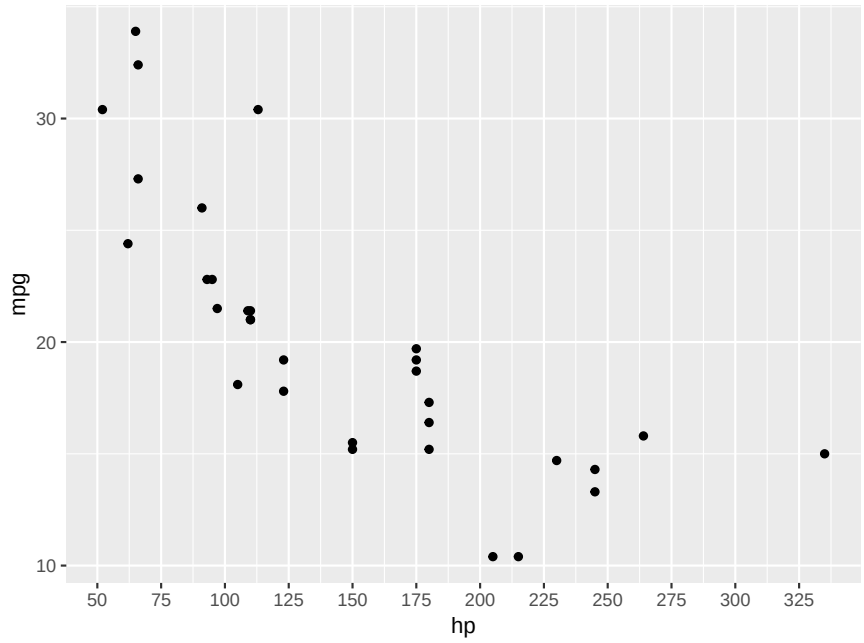
```
df7$y <- sample(c("X", "Y"),
                size = 1000,
                replace = TRUE,
                prob = 2:3)
ggplot(df7, aes(x=x, fill = y)) +
  geom_bar(position = "dodge") +
    scale_y_continuous(labels=scales::percent) +
    labs(x="", y="Percentages", fill="Y")
```



Axis Ticks and Legend Keys

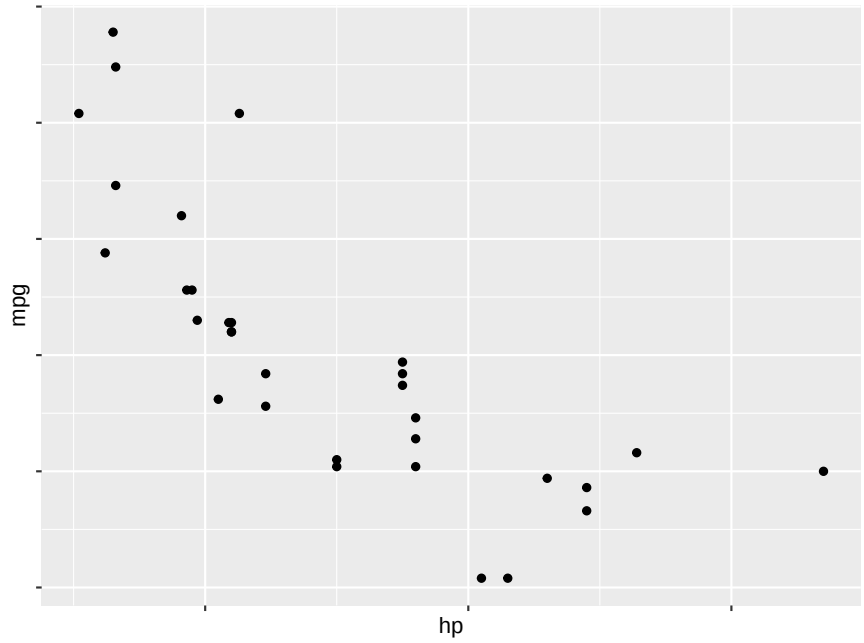
Let's return to the basic plot of mpg by hp. Let's say we want to change the axis tick marks:

```
ggplot(mtcars, aes(hp, mpg)) +
  geom_point() +
  scale_x_continuous(breaks = seq(50, 350, by=25)) +
  scale_y_continuous(breaks = seq(0, 50, by=10))
```



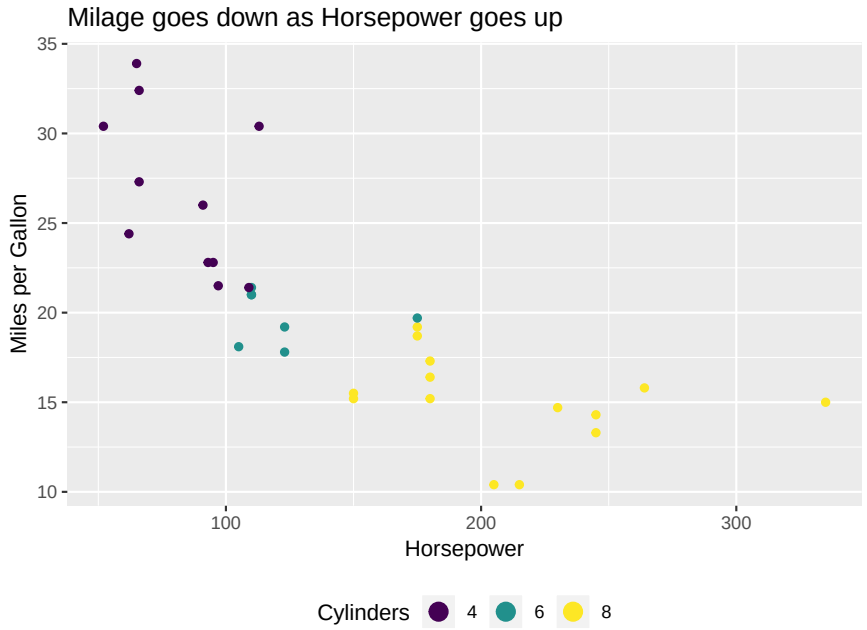
sometimes we want to do graphs without any tick labels. This is useful for example for maps and also for confidential data, so the viewer sees the relationship but can't tell the sizes:

```
ggplot(mtcars, aes(hp, mpg)) +
  geom_point() +
  scale_x_continuous(labels = NULL) +
  scale_y_continuous(labels = NULL)
```



By default ggplot2 draws the legends on the right. We can however change that. We can also change the appearance of the legend. Recall that the basic graph is in *plt2*. Then

```
plt2 +
  theme(legend.position = "bottom") +
  guides(color=guide_legend(nrow = 1,
                            override.aes = list(size=4)))
```



Saving the graph

It is very easy to save a ggplot2 graph. Simply run

```
ggsave("myplot.pdf")
```

it will save the last graph to disc.

One issue is figure sizing. You need to do this so that a graph looks "good". Unfortunately this depends on where it ends up. A graph that looks good on a webpage might look ugly in a pdf. So it is hard to give any general guidelines.

If you use R markdown, a good place to start is with the chunk arguments fig.with=6 and out.width="70%". In fact on top of every R markdown file I have a chunk with

```
library(knitr)
opts_chunk$set(fig.width=6,
               fig.align = "center",
               out.width = "70%",
               warning=FALSE,
               message=FALSE)
```

so that automatically every graph is sized that way. I also change the default behavior of the chunks to something I like better!

Important Commands

In the section I will list the most important commands in base R. The list is taken in large part from Hadley Wickham's book Advanced R. Most of them we already discussed. Those we have not you can read up on yourself.

The first functions to learn

? str

Important operators and assignment

%in%, match
=, <-, <<-
$, [, [[, head, tail, subset
with
assign, get

Comparison

all.equal, identical
!=, ==, >, >=, <, <=
is.na, complete.cases
is.finite

Random variables

(q, p, d, r) * (beta, binom, cauchy, chisq, exp, f, gamma, geom, hyper, lnorm, logis, multinom, nbinom, norm, pois, signrank, t, unif, weibull, wilcox, birthday, tukey)

Matrix algebra

crossprod, tcrossprod
eigen, qr, svd
%*%, %o%, outer
rcond
solve

Workspace

ls, exists, rm
getwd, setwd
q

source
install.packages, library, require

Help

help, ?
help.search
apropos
RSiteSearch
citation
demo
example
vignette

Debugging

traceback
browser
recover
options(error =)
stop, warning, message
tryCatch, try

Output

print, cat
message, warning
dput
format
sink, capture.output

Reading and writing data

data
count.fields
read.csv, write.csv
read.delim, write.delim
read.fwf
readLines, writeLines
readRDS, saveRDS
load, save
library

Files and directories

dir
basename, dirname, tools::file_ext
file.path
path.expand, normalizePath
file.choose
file.copy, file.create, file.remove, file.rename, dir.create
file.exists, file.info
tempdir, tempfile
download.file,

General Statistics

Descriptive Statistics

In general in Statistics we distinguish between quantitative (= numerical) and categorical (= qualitative) data. The main difference is that for quantitative data doing arithmetic makes sense, for example calculating the mean.

Note that just because a data set has digits, it is not necessarily quantitative. For example, digits are often used as labels.

Sometimes a data set can be treated as either categorical or quantitative. Say we have a variable "number of times a student failed a course". Now for some purposes one can treat this as quantitative, for example find the mean. For others we can treat it as categorical, for example do a table or a boxplot.

Consider the **upr admissions** data. Here are some simple things to do when looking at this kind of data:

Tables

```
Gender <- table(upr$Gender)
names(Gender) <- c("Female", "Male")
Percentage <- round(Gender/sum(Gender)*100, 1)
cbind(Gender, Percentage)
```

```
##        Gender Percentage
## Female  11487       48.5
## Male    12179       51.5
```

Contingency Tables

```
tbl <- table(upr$Gender, upr$Class.Facultad)
tbl
```

```
##
##     ADEM ARTES CIAG CIENCIAS INGE
##   F 1401  2570 1038     4127 2351
##   M 1091  1554 1288     2887 5359
```

In a contingency table percentages can be calculated in three ways:

```
# overall total
ot <- sum(tbl)
ot
```

```
## [1] 23666
```

```
# row total
rt <- apply(tbl, 1, sum)
rt
```

```
##     F     M
## 11487 12179
```

```
# column total
ct <- apply(tbl, 2, sum)
ct
```

```
##     ADEM    ARTES     CIAG CIENCIAS     INGE
##     2492     4124     2326     7014     7710
```

- by grand total

```
tmp <- cbind(tbl, Total=rt)
tmp <- rbind(tmp, Total=c(ct, sum(ct)))
round(tmp/ot*100, 1)
```

```
##       ADEM ARTES CIAG CIENCIAS INGE Total
## F      5.9  10.9  4.4     17.4  9.9  48.5
## M      4.6   6.6  5.4     12.2 22.6  51.5
## Total 10.5  17.4  9.8     29.6 32.6 100.0
```

- by row total

```
round(tmp/c(rt, ot)*100, 1)
```

```
##       ADEM ARTES CIAG CIENCIAS INGE Total
## F     12.2  22.4  9.0     35.9 20.5   100
## M      9.0  12.8 10.6     23.7 44.0   100
## Total 10.5  17.4  9.8     29.6 32.6   100
```

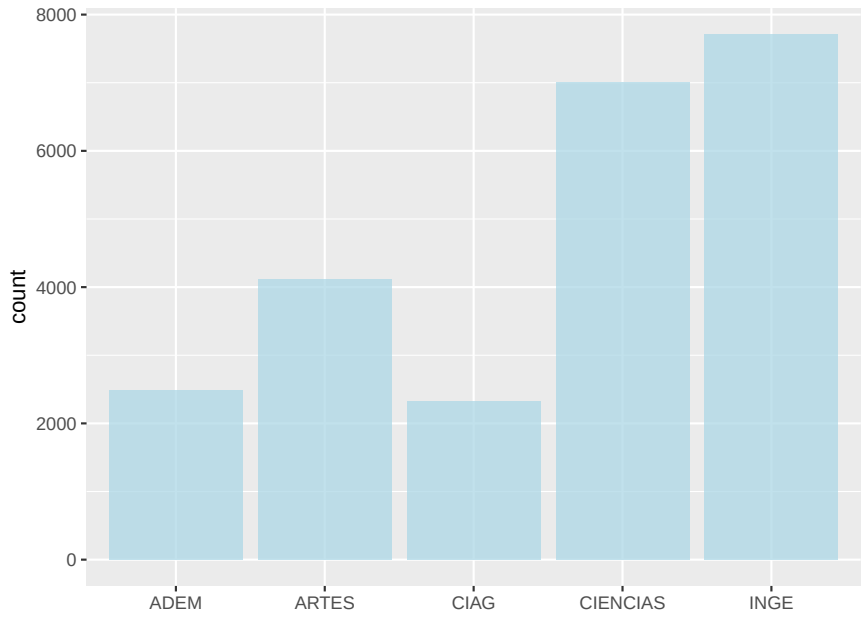
- by column total

```r
t(round(t(tmp)/c(ct, ot)*100, 1))
```

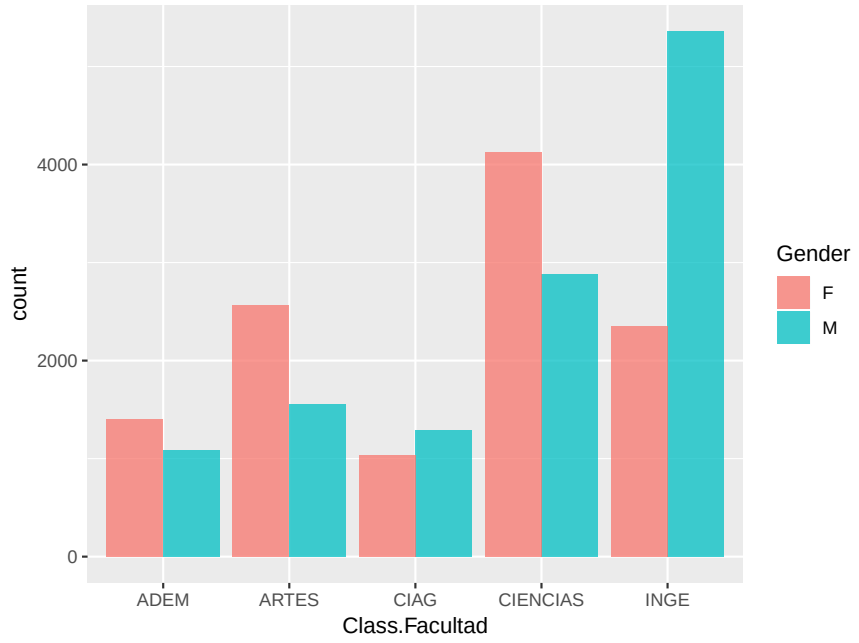
```
##         ADEM ARTES  CIAG CIENCIAS  INGE Total
## F       56.2  62.3  44.6     58.8  30.5  48.5
## M       43.8  37.7  55.4     41.2  69.5  51.5
## Total 100.0 100.0 100.0    100.0 100.0 100.0
```

Bar Charts

```r
ggplot(upr, aes(Class.Facultad)) +
  geom_bar(alpha=0.75, fill="lightblue") +
  xlab("")
```



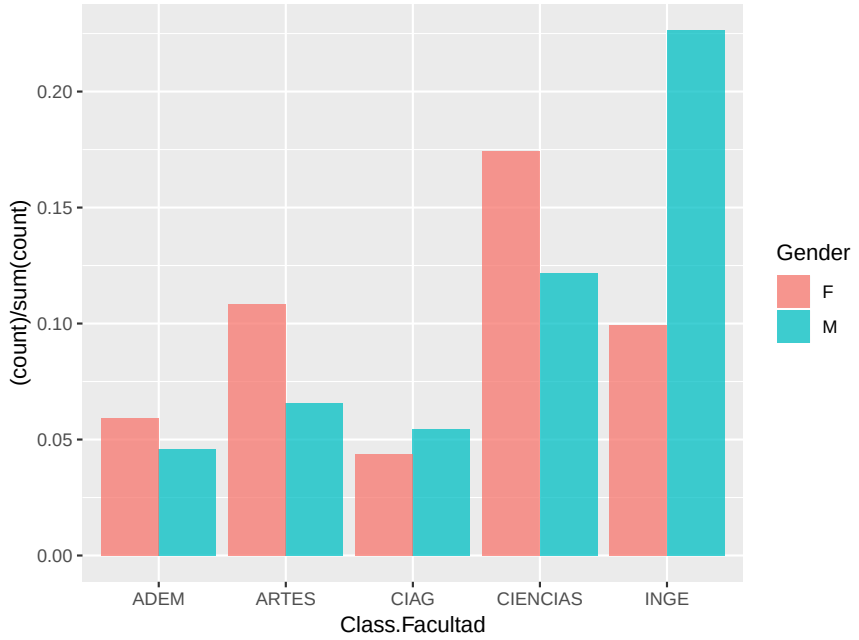

```r
ggplot(upr, aes(Class.Facultad, fill=Gender)) +
  geom_bar(position="dodge", alpha=0.75)
```



as with the tables, graphs can be done based on percentages:

- grand total

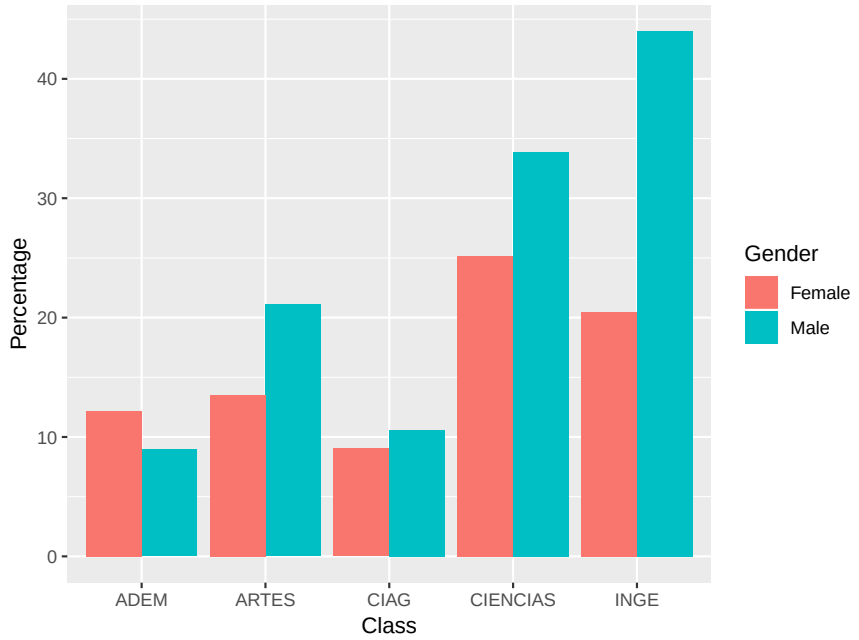
```
ggplot(upr, aes(Class.Facultad, fill=Gender)) +
  geom_bar(aes(y = (..count..)/sum(..count..)),
           position="dodge",
           alpha=0.75)
```



- row total

this one we have to work a bit:

```
tmp1 <- c(tmp[1, 1:5], tmp[2, 1:5])/c(rt, rt)*100
df <- data.frame(Percentage = tmp1,
                 Gender=rep(c("Female", "Male"), 5),
                 Class=names(tmp1))
ggplot(df, aes(x = Class,
               y = Percentage,
               fill = Gender)) +
    geom_bar(position = "dodge",
             stat = "identity")
```



Notice the use of *stat="identity"* if the data is already in the form of a table.

Numerical Summaries

```
round(mean(upr$Freshmen.GPA), 3)
```

```
## [1] NA
```

we get an error because there are missing values, so

```
round(mean(upr$Freshmen.GPA, na.rm=TRUE), 3)
```

```
## [1] 2.733
```

```
round(median(upr$Freshmen.GPA, na.rm=TRUE), 3)
```

```
## [1] 2.83
```

```
round(sd(upr$Freshmen.GPA, na.rm=TRUE), 3)
```

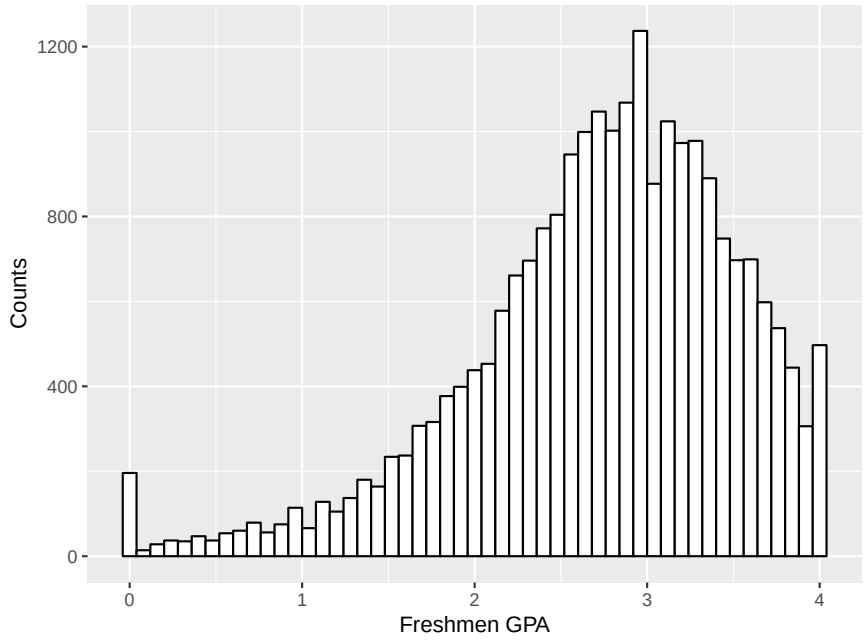
```
## [1] 0.779
```

```
round(quantile(upr$Freshmen.GPA,
               probs = c(0.1, 0.25, 0.75, 0.9),
               na.rm=TRUE), 3)
```

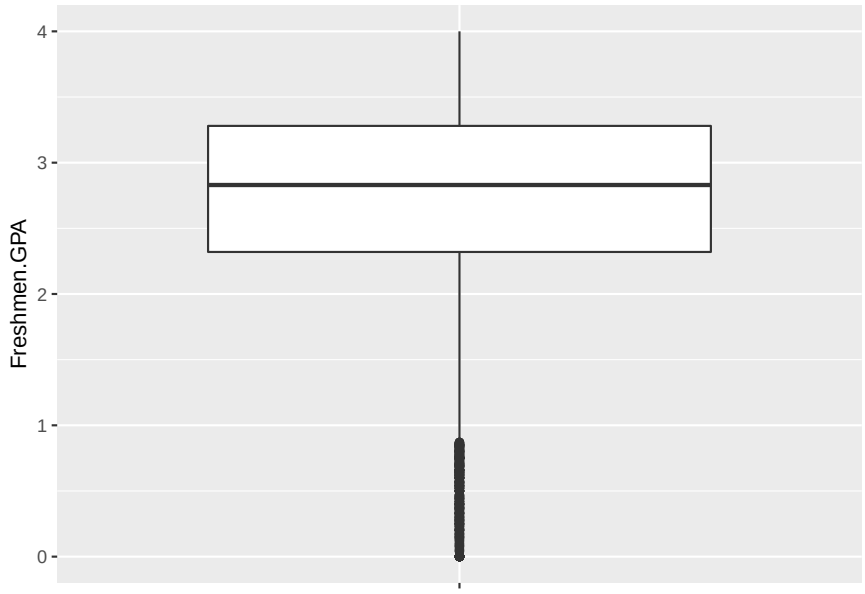
```
##  10%  25%  75%  90%
## 1.71 2.32 3.28 3.65
```

Histogram and Boxplot

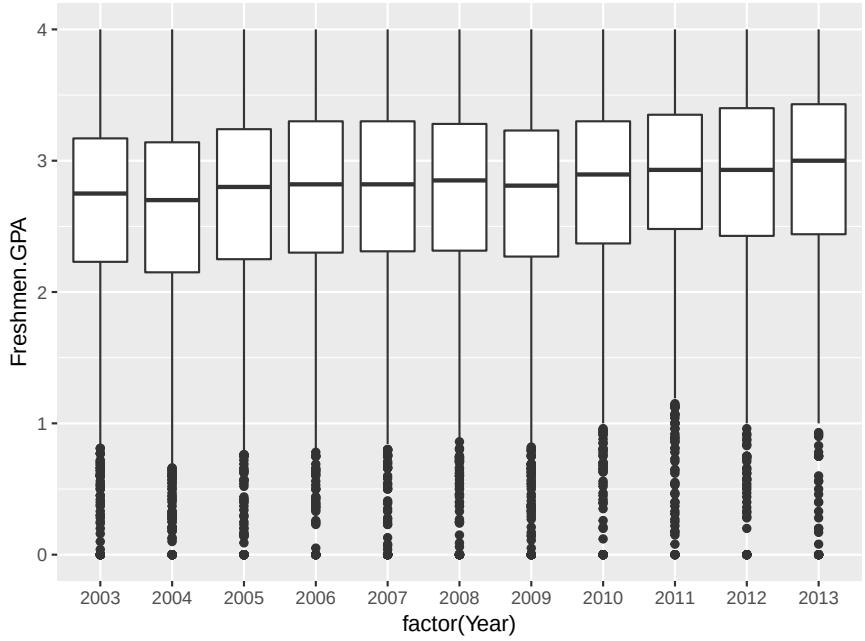
```
bw <- diff(range(upr$Freshmen.GPA, na.rm = TRUE))/50 # use about 50 bins
ggplot(upr, aes(Freshmen.GPA)) +
  geom_histogram(color = "black",
                 fill = "white",
                 binwidth = bw) +
  labs(x = "Freshmen GPA", y = "Counts")
```




```
ggplot(upr, aes(x="", y=Freshmen.GPA)) +
  geom_boxplot() +
  xlab("")
```




```
ggplot(upr, aes(factor(Year), Freshmen.GPA)) +
  geom_boxplot()
```

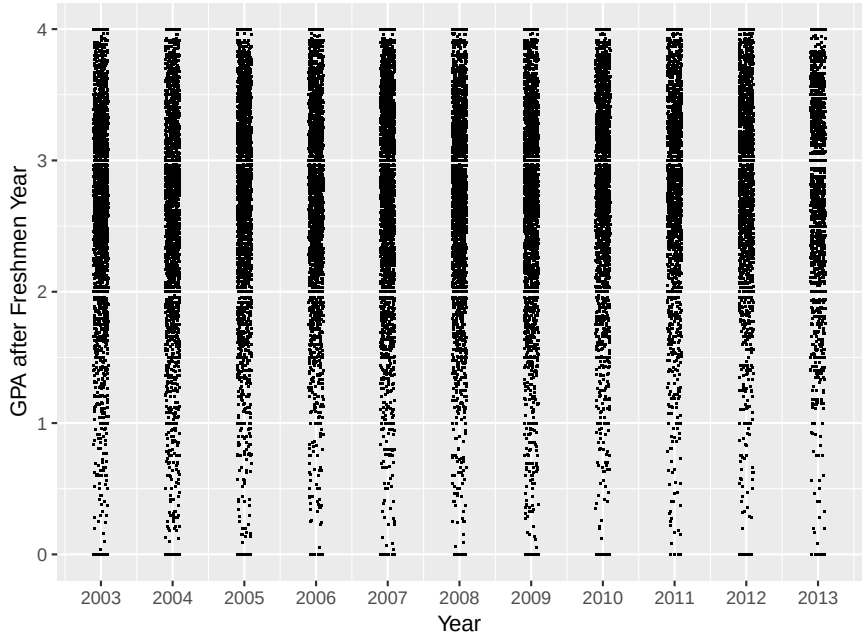


Two Quantitative Variables

```
round(cor(upr$Year, upr$Freshmen.GPA,
    use="complete.obs"), 3)
```

```
## [1] 0.097
```

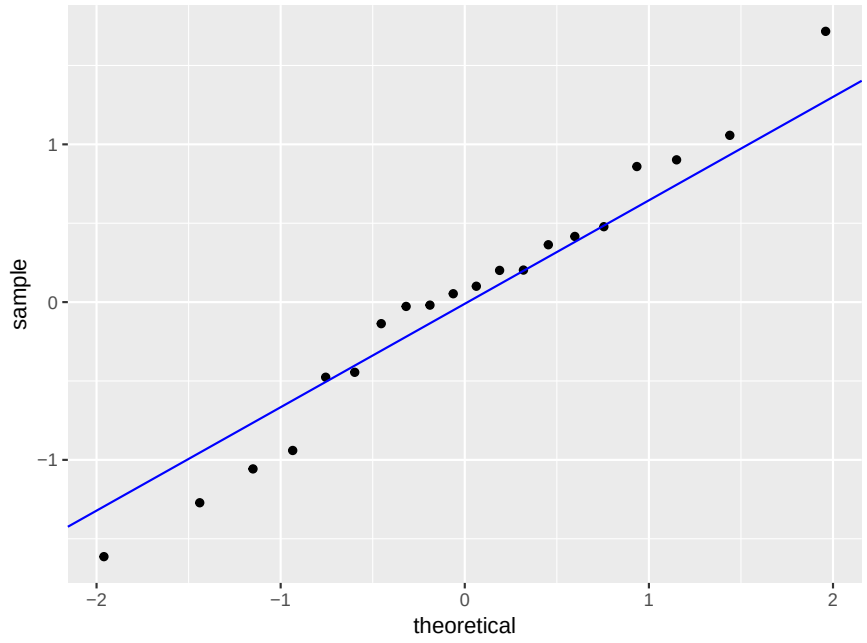
```
ggplot(upr, aes(Year, Freshmen.GPA)) +
  geom_jitter(shape=".", width=0.1, height = 0) +
  scale_x_continuous(breaks = 2003:2013) +
  labs(x="Year", y="GPA after Freshmen Year")
```



Normal Probability Plot

An important graph is the *normal probability plot*, which plots the sample quantiles vs the population quantiles of a normal distribution:

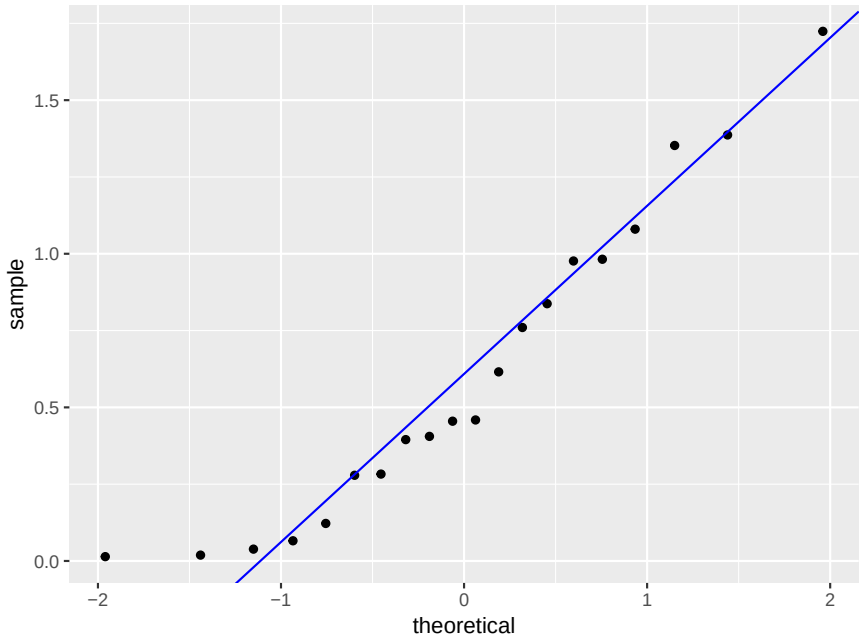
```
x <- rnorm(20)
df <- data.frame(x=x)
y1 <- quantile(x, c(0.25, 0.75))
x1 <- qnorm(c(0.25, 0.75))
slope <- diff(y1)/diff(x1)
int <- y1[1L] - slope * x1[1L]
ggplot(df, aes(sample=x)) +
  stat_qq() +
  geom_abline(slope = slope, intercept = int, color="blue")
```



Notice that adding the customary line through the quartiles takes a bit of work.

Here an example where the normal assumption fails:

```
x <- rexp(20)
df <- data.frame(x=x)
y1 <- quantile(x, c(0.25, 0.75))
x1 <- qnorm(c(0.25, 0.75))
slope <- diff(y1)/diff(x1)
int <- y1[1L] - slope * x1[1L]
ggplot(df, aes(sample=x)) +
  stat_qq() +
  geom_abline(slope = slope, intercept = int, color="blue")
```



Parameter Estimation

In this section we will study the problem of parameter estimation. In its most general form this is as follows: we have a sample $X_1, .., X_n$ from some probability density $f(x; \theta)$. Here both x and θ might be vectors. Also we will use the term density for both the discrete and the continuous case.

The problem is to find an estimate of θ based on the data $X_1, .., X_n$, that is a function (called a *statistic*) $T(X_1, .., X_n)$ such that in some sense $T(X_1, .., X_n) \approx \theta$.

Example: Binomial proportions

in a survey 567 people 235 said they prefer Coke over Pepsi. What is the percentage of people who prefer Coke over Pepsi?

The answer seems obvious: 235/567. However, let's work this out in detail. First, each person is a *Bernoulli trial* (yes/no) with some success probability π. So we have the density

$$P(X_i = 1) = 1 - P(X_i = 0) = \pi$$

which we can write as

$$f(x) = \pi^x (1 - \pi)^{1-x},\ x = 0, 1$$

the joint density is given by

$$
\begin{aligned}
&f(x_1, .., x_n) = \\
&\prod_{i=1}^{n} \pi_i^x (1-\pi)^{1-x_i} = \\
&\pi^{\sum x_i} (1-\pi)^{\sum (1-x_i)} = \\
&\pi^y (1-\pi)^{n-y}
\end{aligned}
$$

where $y = \sum x_i$ is the number of *successes*.

In our case this becomes $\pi^{235}(1-\pi)^{332}$ and the task is to estimate π.

Likelihood function

say we have $X_1, .., X_n \sim f(x;\theta)$ and independent. Then the joint density of $\mathbf{X} = (X_1, .., X_n)$ is given by

$$f(\mathbf{x};\theta) = \prod_{i=1}^{n} f(x_i;\theta)$$

The **likelihood function L** is defined by

$$L(\theta;\mathbf{x}) = \prod_{i=1}^{n} f(x_i;\theta)$$

this does not seem to be much: the right hand side is the same. However, it is a very different expression: in $f(\mathbf{x};\theta)$ $\mathbf{x}$ is the variable and θ is an (unknown) constant. In $L(\theta;\mathbf{x})$ θ is the variable and $\mathbf{x}$ is a (known) constant.

It is the essential difference of **before** the experiment, when one might ask questions of probability, and **after** the experiment, when one asks questions of statistics.

Closely related is the **log-likelihood function**

$$l(\theta;\mathbf{x}) = \log L(\theta;\mathbf{x}) = \sum_{i=1}^{n} \log f(x_i;\theta)$$

The log-likelihood is often easier to work with, not the least because it turns a product into a sum.

There is a principle in Statistics that suggests that any inference should always be based on the likelihood function.

Example: Binomial proportions

We already found the joint density

$$\pi^y (1-\pi)^{n-y}$$

and so the log likelihood is

$$l(\pi; y, n) = y \log \pi + (n - y) \log(1 - \pi)$$

Example: Normal mean

Say $X_i \sim N(\mu, \sigma)$ so

$$\begin{aligned}
&l(\mu; \mathbf{x}, \sigma) = \\
&\sum_{i=1}^{n} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left\{ -\frac{1}{2\sigma^2} (x_i - \mu)^2 \right\} \right] = \\
&\frac{n}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (x_i - \mu)^2
\end{aligned}$$

Now let $\bar{x} = \frac{1}{n} \sum x_i$, then

$$\begin{aligned}
&\sum_{i=1}^{n} (x_i - \mu)^2 = \\
&\sum_{i=1}^{n} (x_i - \bar{x} + \bar{x} - \mu)^2 = \\
&\sum_{i=1}^{n} (x_i - \bar{x})^2 + 2 \sum_{i=1}^{n} (x_i - \bar{x})(\bar{x} - \mu) + \sum_{i=1}^{n} (\bar{x} - \mu)^2 = \\
&\sum_{i=1}^{n} (x_i - \bar{x})^2 + 2(\bar{x} - \mu) \sum_{i=1}^{n} (x_i - \bar{x}) + n(\bar{x} - \mu)^2 = \\
&\sum_{i=1}^{n} (x_i - \bar{x})^2 + 2(\bar{x} - \mu)(\sum_{i=1}^{n} x_i - n\bar{x}) + n(\bar{x} - \mu)^2 = \\
&\sum_{i=1}^{n} (x_i - \bar{x})^2 + n(\bar{x} - \mu)^2
\end{aligned}$$

and so

$$l(\mu; \mathbf{x}, \sigma) = \text{const} - \frac{1}{2\sigma^2/n} (\mu - \bar{x})^2$$

so as a function of μ the log-likelihood is a quadratic function with vertex at $\bar{x}$

Maximum Likelihood estimation

The idea of maximum likelihood estimation is to find that value of θ that maximizes the likelihood function. In an obvious way, this is the value of the parameter that best "agrees"" with the data.

Of course a function L has a maximum at x iff $\log L$ has a maximum at x, so we can also (and easier!) maximize the log-likelihood function

Example: Normal mean

$$\frac{dl(\mu; \mathbf{x}, \sigma)}{d\mu} = -\frac{1}{\sigma^2/n}(\mu - \bar{x}) = 0$$

so $\hat{\mu} = \bar{x}$

This is of course a maximum, and not a mimimum or an inflection point because $-\frac{1}{\sigma^2/n} < 0$.

Example: Binomial proportion

$$\frac{dl}{d\pi} = \frac{y}{\pi} - \frac{n-y}{1-\pi} = 0$$
$$\hat{\pi} = \frac{y}{n}$$

and for our numbers we find $\hat{\pi} = 235/527 = 0.4459$

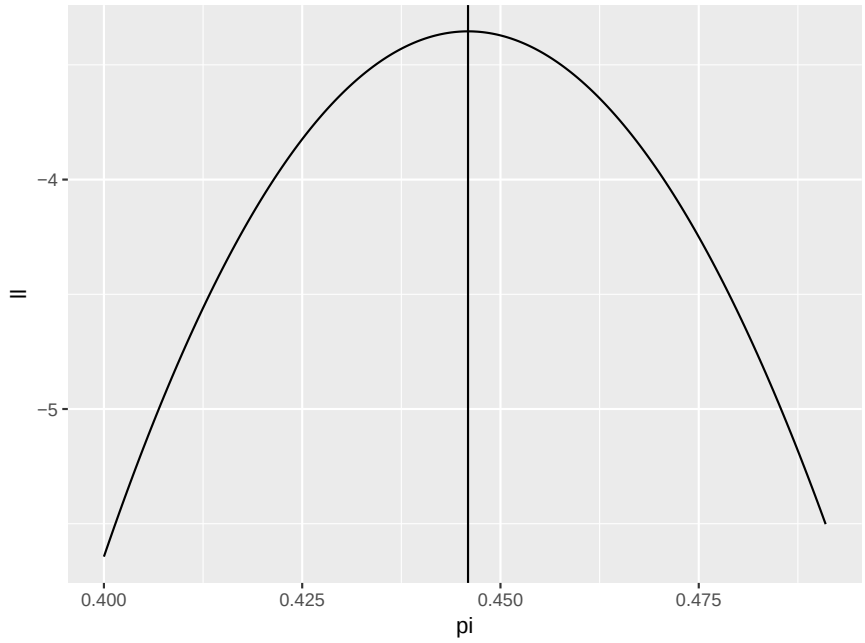
Numerical Computation

The above calculations require some calculus. Sometimes we can let R take care of this for us:

```
ll <- function(pi, y, n)
  log(dbinom(y, n, pi))
pi <- seq(0.4, 0.491, length=1000)
df <- data.frame(pi=pi, ll=ll(pi, 235, 527))
mle <- df$pi[df$ll==max(df$ll)]
mle
```

```
## [1] 0.4459099
```

```
ggplot(df, aes(x=pi, y=ll)) +
  geom_line() +
  geom_vline(xintercept = mle)
```



notice that the log-likelihood curve looks a lot like a parabola. This is not an accident, and it will come in handy soon!

Example: Beta distribution

A random variable is said to have a Beta density if

$$f(x;\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}$$

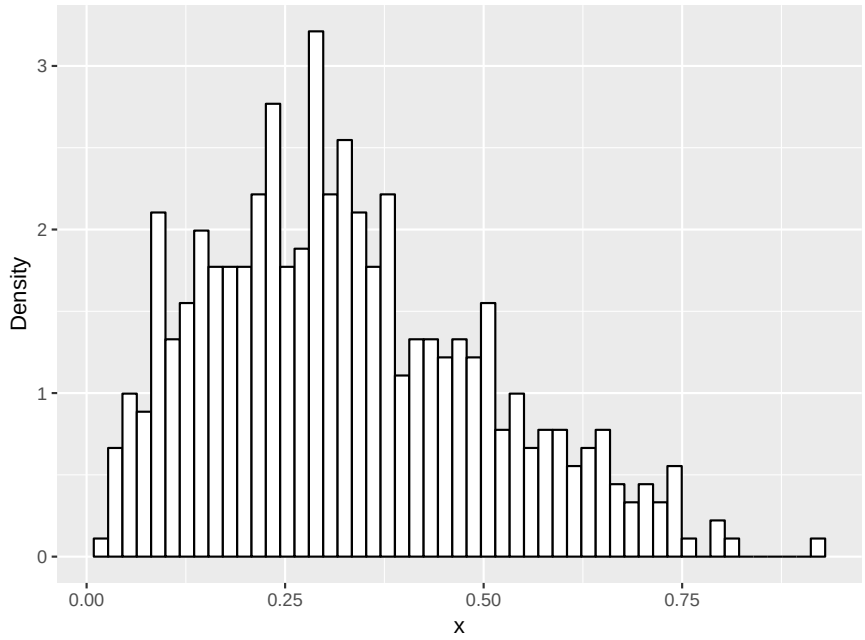
here Γ is the famous gamma function

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$$

Let's simulate a sample from a Beta distribution

```
set.seed(1111)
x <- sort(round(rbeta(500, 2, 4), 3))
beta.ex <- data.frame(x=x)
```

```
bw <- diff(range(x))/50
ggplot(beta.ex, aes(x)) +
  geom_histogram(aes(y = ..density..),
    color = "black",
    fill = "white",
    binwidth = bw) +
    labs(x = "x", y = "Density")
```



and we want to estimate α, β.

Now doing this with calculus is out because the log-likelihood function doesn't exist in closed form. Instead we will need to use a numerical method. Because the Beta distribution is a standard one, there is an R routine to do it for us. It is part of the package

```
library(MASS)
fitdistr(x,
         densfun="Beta",
         start=list(shape1=1,shape2=1))
```

```
##      shape1      shape2
##   2.0716029   4.2045833
##  (0.1227766) (0.2646915)
```

Example: linear density

here are observations from a linear density $f(x|a) = 2ax + 1 - a$, $0 < x < 1$ and $-1 < a < 1$:

```
0.005 0.011 0.025 0.03 0.031 0.031 0.05 0.059 0.061 0.064 0.067 0.068 0.072 0.075 0.082
```

We want to estimate a. So let's see:

$$f(x|a) = \prod_{i=1}^{n} [2ax_i + 1 - a] =$$

$$l(a) = \sum_{i=1}^{n} \log [2ax_i + 1 - a]$$

$$\frac{dl}{da} = \sum_{i=1}^{n} \frac{2x_i - 1}{2ax_i + 1 - a} = 0$$

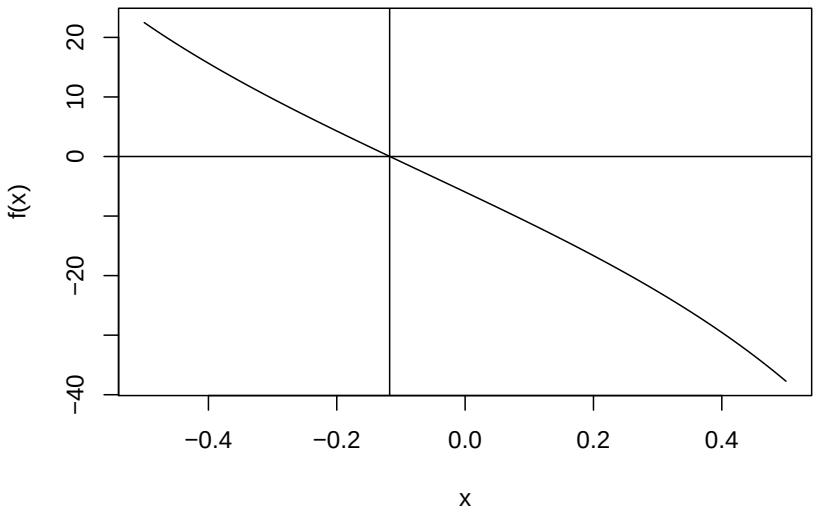
and this equation can not be solved analytically. Unfortunately this is not one of the distributions included in *fitdistr*, so we need to find a numerical solution ourselves.

Here are several:

- Simple Grid Search

Let's draw the curve of $\frac{dl}{da}$. In what follows x is the data above.

```
f <- function(a) {
  y <- a
  for(i in seq_along(a))
    y[i] <- sum( (2*x-1)/(2*a[i]*x+1-a[i]) )
  y
}
curve(f, -0.5, 0.5)
abline(h=0)
a <- seq(-0.5, 0.5, length=1000)
y <- f(a)
# find value of a where y is closest to 0
mle <- a[abs(y)==min(abs(y))]
abline(v=mle)
```



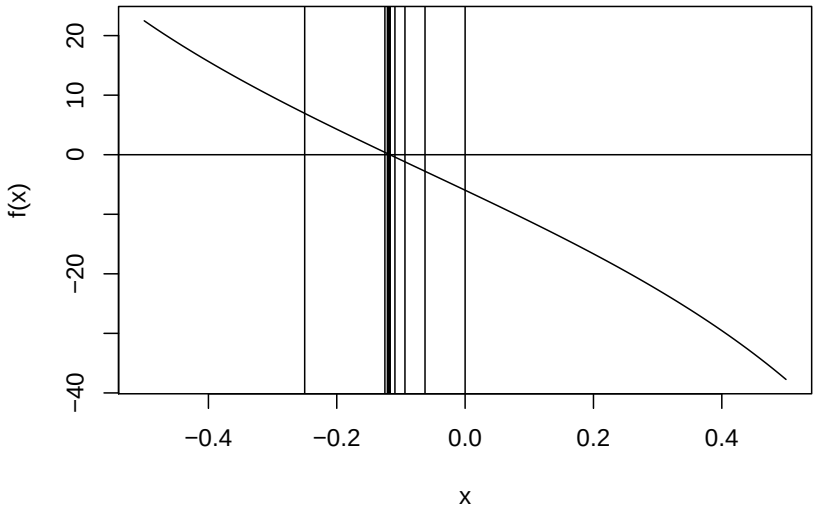

```
mle
```

```
## [1] -0.1176176
```

- Bisection Algorithm

The idea is this: our function is positive for a=-0.5 and negative for a=0.5. It is also continuous and decreasing. So we can find the zero by checking midpoints and adjusting the upper or lower limit accordingly:

```
curve(f, -0.5, 0.5)
abline(h=0)
low <- (-0.5)
high <- 0.5
repeat {
  mid <- (low+high)/2
  y <- f(mid)
  print(c(mid, y))
  abline(v=mid)
  if(y>0) low <- mid
  else high <- mid
  if(high-low<0.0001) break
}
```




```
## [1]  0.000 -5.962
## [1] -0.250000  6.946702
## [1] -0.1250000  0.3969489
## [1] -0.062500 -2.783515
## [1] -0.093750 -1.196284
## [1] -0.1093750 -0.4007672
## [1] -0.117187500 -0.002228235
## [1] -0.1210938  0.1972750
## [1] -0.11914062  0.09750275
## [1] -0.11816406  0.04763219
## [1] -0.11767578  0.02270072
## [1] -0.11743164  0.01023593
## [1] -0.117309570  0.004003769
## [1] -0.1172485352  0.0008877472
```

- Newton's Method

Isaak Newton invented the following algorithm: we want to solve the equation $f(x) = 0$. Let x_0 be some starting point. Then find successive points with

$$x_{n+1} = x_n - f(x_n)/f'(x_n)$$

notice that if this sequence converges to x we have

$$x = x - f(x)/f'(x)$$

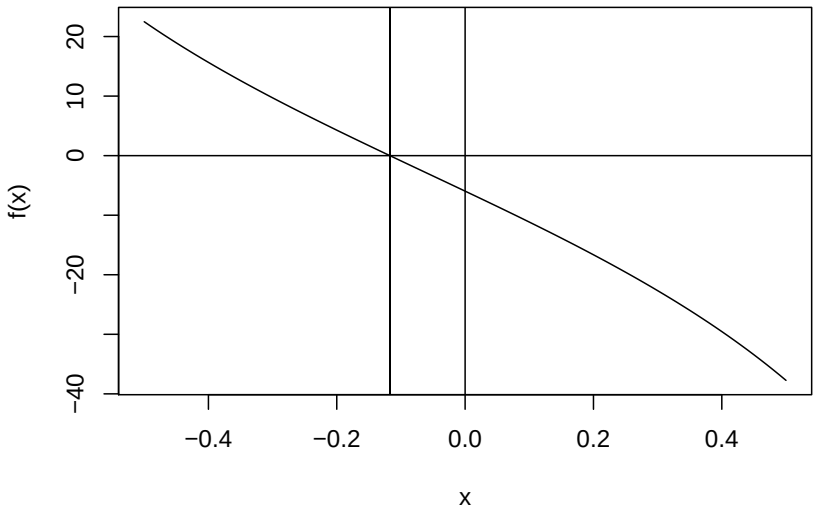
and so $f(x) = 0$.

In our case we have $f(a) = \frac{dl}{da}$ and so we also need

$$f'(a) = \frac{d^2l}{da^2} = -\sum_{i=1}^{n} \left(\frac{2x_i - 1}{2ax_i + 1 - a} \right)^2$$

we find

```
f.prime<- function(a) {
  y <- a
  for(i in seq_along(a))
    y[i] <- (-1)*sum( ((2*x-1)/(2*a[i]*x+1-a[i]))^2 )
  y
}
curve(f, -0.5, 0.5)
abline(h=0)
x.old <- 0
abline(v=x.old)
repeat {
  x.new <- x.old - f(x.old)/f.prime(x.old)
  print(x.new)
  abline(v=x.new)
  if(abs(x.old-x.new)<0.0001) break
  x.old <- x.new
}
```



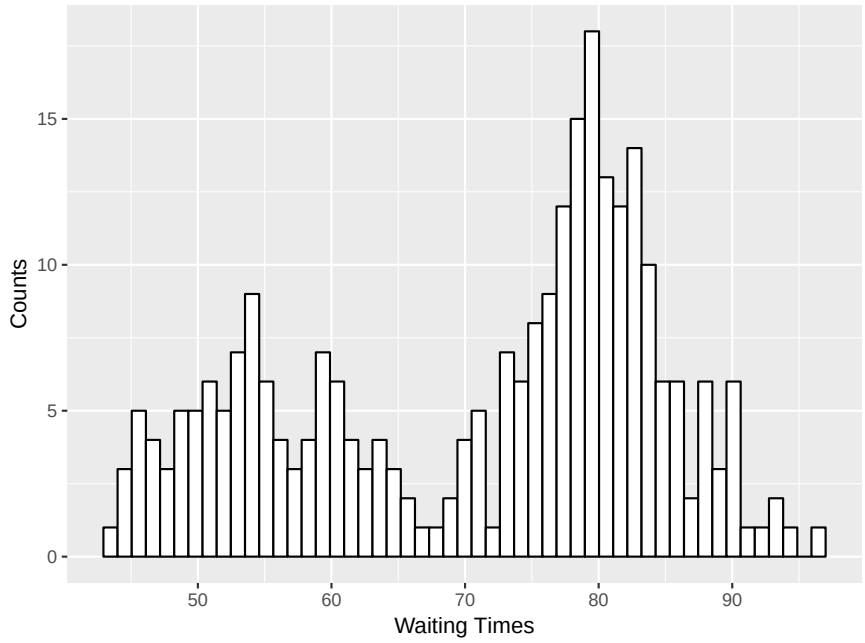

```
## [1] -0.1167333
## [1] -0.1172312
## [1] -0.1172311
```

Notice that his converges much faster, it only needed three "rounds". This is typically true, however Newton's method also can fail badly if the starting point is not good enough.

Example Old Faithful guyser

Consider the waiting times of the famous Old Faithful data:

```
bw <- diff(range(faithful$Waiting.Time))/50
ggplot(faithful, aes(Waiting.Time)) +
          geom_histogram(color = "black",
                 fill = "white",
                 binwidth = bw) +
  labs(x = "Waiting Times", y = "Counts")
```



What would be a useful model for this data? We can try a *normal mixture*:

$$\alpha N(\mu_1, \sigma_1) + (1 - \alpha) N(\mu_2, \sigma_2)$$

It seems that the two parts split at around 65, so we find

```
x <- faithful$Waiting.Time
round(c(mean(x[x<65]), mean(x[x>65])), 1)
```

```
## [1] 54.1 80.0
```

```
round(c(sd(x[x<65]), sd(x[x>65])), 1)
```

```
## [1] 5.4 5.9
```

How about α? Let's find the mle:

$$\begin{aligned}
\phi(x, \mu, \sigma) &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{1}{2\sigma^2}(x-\mu)^2} \\
l(\alpha) &= \sum \log\left[\alpha\phi(x_i, \mu_1, \sigma_1) + (1-\alpha)\phi(x_i, \mu_2, \sigma_2)\right] \\
\frac{dl}{d\alpha} &= \sum \frac{\phi(x_i, \mu_1, \sigma_1) - \phi(x_i, \mu_2, \sigma_2)}{\alpha\phi(x_i, \mu_1, \sigma_1) + (1-\alpha)\phi(x_i, \mu_2, \sigma_2)} \\
\frac{d^2l}{d\alpha^2} &= (-1) \sum \left(\frac{\phi(x_i, \mu_1, \sigma_1) - \phi(x_i, \mu_2, \sigma_2)}{\alpha\phi(x_i, \mu_1, \sigma_1) + (1-\alpha)\phi(x_i, \mu_2, \sigma_2)} \right)^2
\end{aligned}$$

```
f <- function(alpha, mu1=54.1, sigma1=5.4,
              mu2=80, sigma2=5.9,
              x=faithful$Waiting.Time) {
  u <- dnorm(x, mu1, sigma1)
```

```
  v <- dnorm(x, mu2, sigma2)
  y1 <- alpha
  y2 <- alpha
  for(i in 1:seq_along(alpha)) {
    tmp <- (u-v)/(alpha[i]*u+(1-alpha[i])*v)
    y1[i] <- sum(tmp)
    y2[i] <- (-1)*sum(tmp^2)
  }
  list(y1, y2)
}
alpha.old <- 0.5
repeat {
  tmp <- f(alpha.old)
  alpha.new <- alpha.old - tmp[[1]]/tmp[[2]]
  print(alpha.new)
  if(abs(alpha.old-alpha.new)<0.0001) break
  alpha.old <- alpha.new
}
```

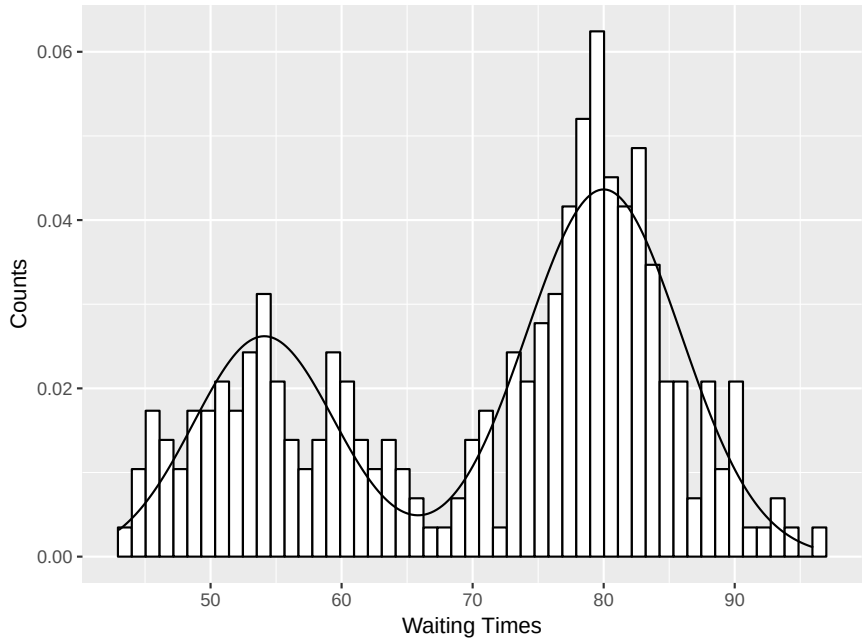
```
## [1] 0.3550228
## [1] 0.3545702
## [1] 0.3545704
```

```
alpha <- alpha.old
alpha
```

```
## [1] 0.3545702
```

Let's see what this looks like:

```
x <- seq(min(faithful$Waiting.Time),
         max(faithful$Waiting.Time),
         length=250)
y <- alpha*dnorm(x, 54.1, 5.4) +
     (1-alpha)*dnorm(x, 80, 5.9)
df <- data.frame(x=x, y=y)
bw <- diff(range(faithful$Waiting.Time))/50
ggplot(faithful, aes(Waiting.Time)) +
  geom_histogram(aes(y = ..density..),
                 color = "black",
                 fill = "white",
                 binwidth = bw) +
  labs(x = "Waiting Times", y = "Counts") +
  geom_line(aes(x, y),
            data=df,
            inherit.aes = FALSE)
```



How about the other parameters? Can we fit for them as well? What we need is a multivariate version of Newton's method:

Say we have the equation

$$f(x_1, .., x_n) = 0$$

Define the gradient Δ and the Hessian matrix H by

$$\Delta_i(x) = \frac{df}{dx_i}(x_1, .., x_n) H_{i,j}(x) = \frac{d^2 f}{dx_i dx_j}(x_1, .., x_n)$$

then the algorithm is

$$x_{new} = x_{old} - H^{-1}_{i,j}(x_{old}) \Delta_i(x_{old})$$

Let's fix $\alpha = 0.355$, $\sigma_1 = 5.4$ and $\sigma_2 = 5.9$ and fit for μ_1 and μ_2.

$$f(\mu) = \phi(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right)$$
$$f'(\mu) = \frac{d\phi}{d\mu} = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right) \frac{x-\mu}{\sigma^2} =$$
$$(x-\mu) f/\sigma^2$$
$$f''(\mu) = -f/\sigma^2 + (x-\mu)^2 f/\sigma^4 =$$
$$\left[(x-\mu)^2 - \sigma^2\right] f/\sigma^4$$

Let's use the following definitions (short-hands):

$$f_{i,j} = \phi(x_i, \mu_j, \sigma_j),\ j = 1, 2$$
$$\psi_i = \alpha f_{i,1} + (1-\alpha) f_{i,2}$$

so we have

$$l(\mu_1, \mu_2) = \sum \log \psi_i$$
$$\Delta_1 = \frac{dl}{d\mu_1} = \alpha \sum \frac{f'_{i,1}}{\psi_i}$$
$$\Delta_2 = (1-\alpha) \sum \frac{f'_{i,2}}{\psi_i}$$
$$H_{1,1} = \alpha \sum \frac{f''_{i,1} - \alpha f'^2_{i,1}}{\psi_i}$$
$$H_{1,2} = -(1-\alpha)\alpha \sum \frac{f'_{i,1} f'_{i,2}}{\psi_i}$$
$$H_{2,1} = H_{1,2}$$
$$H_{2,2} = (1-\alpha) \sum \frac{f''_{i,2} - (1-\alpha) f'^2_{i,2}}{\psi_i}$$

Let's implement this:

```
alpha <- 0.355
sigma <- c(5.4, 5.9)
mu.old <- c(50, 80)
grad <- rep(0, 2)
H <- diag(2)
k <- 0
x <- faithful$Waiting.Time
repeat {
  k <- k+1
  f1 <- dnorm(x, mu.old[1], sigma[1])
  f1.prime <- (x-mu.old[1])*f1/sigma[1]^2
  f1.doubleprime <-
    ((x-mu.old[1]^2-sigma[1]^2))*f1/sigma[1]^4
  f2 <- dnorm(x, mu.old[2], sigma[2])
  f2.prime <- (x-mu.old[2])*f2/sigma[2]^2
  f2.doubleprime <-
    ((x-mu.old[2]^2-sigma[2]^2))*f2/sigma[2]^4
  psi <- alpha*f1+(1-alpha)*f2
  grad[1] <- alpha*sum(f1.prime/psi)
  grad[2] <- (1-alpha)*sum(f2.prime/psi)
  H[1, 1] <- alpha*sum((f1.doubleprime-alpha*f1.prime^2)/psi)
  H[1, 2] <- (alpha-1)*alpha*sum((f1.prime*f2.prime)/psi)
  H[2, 1] <- H[2, 1]
```

```
  H[2, 2] <- (1-alpha)*sum((f2.doubleprime-(1-alpha)*f2.prime^2)/psi)
  mu.new <- c(mu.old-solve(H)%*%cbind(grad))
  print(c(mu.new, sum(log(psi))))
  if(sum(abs(mu.old-mu.new))<0.001) break
  if(k>10) break
  mu.old <- mu.new
}
```

```
## [1]    50.04450    79.99726 -1061.44499
## [1]    50.08849    79.99457 -1060.91574
## [1]    50.13196    79.99192 -1060.39760
## [1]    50.17492    79.98931 -1059.89032
## [1]    50.21739    79.98675 -1059.39364
## [1]    50.25937    79.98423 -1058.90732
## [1]    50.30086    79.98175 -1058.43113
## [1]    50.34187    79.97931 -1057.96484
## [1]    50.38242    79.97691 -1057.50821
## [1]    50.42250    79.97456 -1057.06105
## [1]    50.46213    79.97224 -1056.62312
```

Or we can make use of R:

```
x <- faithful$Waiting.Time
fun <- function(mu, alpha=0.355, sigma=c(5.4, 5.9)) {
  f1 <- dnorm(x, mu[1], sigma[1])
  f2 <- dnorm(x, mu[2], sigma[2])
  psi <- alpha*f1+(1-alpha)*f2
  -sum(log(psi))
}
optim(c(50,80), fun)
```

```
## $par
## [1] 54.44039 79.99045
##
## $value
## [1] 1034.465
##
## $counts
## function gradient
##       49       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

In fact why not fit for all?

```r
x <- faithful$Waiting.Time
fun <- function(par) {
  f1 <- dnorm(x, par[2], par[3])
  f2 <- dnorm(x, par[4], par[5])
  psi <- par[1]*f1+(1-par[1])*f2
  -sum(log(psi))
}
optim(c(0.5, 50, 5.4, 80, 5.9), fun)
```

```
## $par
## [1]  0.3610815 54.6234464  5.8871485 80.1082477  5.8562652
##
## $value
## [1] 1034.003
##
## $counts
## function gradient
##      211       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

EM Algorithm

There is another way to find the mle's in a problem of this kind called the EM or *Expectation-Maximization* algorithm. The idea is as follows.

Say there were a second variable Z_i which is 0 if the next waiting time is a short one and 1 otherwise. Now if we knew those Z_i's it would be easy to estimate the μ's:

$$\hat{\mu_i} = \text{mean}(X|Z = z_i)\hat{\sigma_i} = \text{sd}(X|Z = z_i)$$

On the other hand if we knew all the means and standard deviations it would also be easy to estimate α:

$$w_i = \frac{\alpha f_{i,1}}{\psi_i}\hat{\alpha} = \text{mean}(w_i)$$

The w_i are called the weights. These formulas can be verified easily using probability theory.

This suggests the following algorithm:

- choose a starting point for the parameters

- find the weights
- find the next estimates of the parameters
- iterate until convergence

```
alpha <- 0.355
mu <- c(50, 80)
sigma <- c(5.4, 5.9)
w <- rep(0, 40)
k <- 0
x <- faithful$Waiting.Time
repeat {
  k <- k+1
  psi <- (alpha*dnorm(x, mu[1], sigma[1]) +
        (1-alpha)*dnorm(x, mu[2], sigma[2]))
  w <- alpha*dnorm(x, mu[1], sigma[1])/psi
  alpha <- mean(w)
  mu[1] <- sum(w*x)/sum(w)
  mu[2] <- sum((1-w)*x)/sum(1-w)
  sigma[1] <- sqrt(sum(w*(x-mu[1])^2)/sum(w))
  sigma[2] <- sqrt(sum((1-w)*(x-mu[2])^2)/sum(1-w))
  psi1 <- (alpha*dnorm(x, mu[1], sigma[1]) +
        (1-alpha)*dnorm(x, mu[2], sigma[2]))
  cat(round(alpha,4), " ",
      round(mu, 1), " ",
      round(sigma, 2), " ",
      round(sum(log(psi1)), 5), "\n")
  if(sum(abs(psi-psi1))<0.001) break
  if(k>100) break
}
```

```
## 0.3345   53.8 79.5   5.21 6.46   -1035.686
## 0.3422   54 79.7   5.42 6.31   -1034.87
## 0.348   54.2 79.8   5.55 6.17   -1034.425
## 0.3522   54.3 79.9   5.65 6.07   -1034.199
## 0.3551   54.4 80   5.72 6   -1034.091
## 0.357   54.5 80   5.77 5.95   -1034.041
## 0.3583   54.5 80   5.8 5.92   -1034.019
## 0.3592   54.6 80.1   5.82 5.9   -1034.009
## 0.3598   54.6 80.1   5.84 5.89   -1034.005
## 0.3602   54.6 80.1   5.85 5.88   -1034.003
## 0.3604   54.6 80.1   5.86 5.88   -1034.002
## 0.3606   54.6 80.1   5.86 5.87   -1034.002
## 0.3607   54.6 80.1   5.87 5.87   -1034.002
## 0.3607   54.6 80.1   5.87 5.87   -1034.002
## 0.3608   54.6 80.1   5.87 5.87   -1034.002
```

Notice one feature of the EM algorithm: it guarantees that each iteration moves the parameters closer to the mle.

The EM algorithm was originally invented by Dempster and Laird in 1977 to deal with a common problem in Statistics called *censoring*: say we are doing a study on survival of patients after cancer surgery. Any such study will have a time limit after which we will have to start with the data analysis, but hopefully there will still be some patients who are alive, so we don't know their survival times, but we do know that the survival times are greater than the time that has past sofar. We say the data is censored at time T. The number of patients with survival times >T is important information and should be used in the analysis. If we order the observations into (x_1, .., x_n) the uncensored observations (the survival times of those patients that are now dead) and (x_{n+1}, .., x_{n+m}) the censored data, the likelihood function can be written as

$$L(\theta|x) = [1 - F(T|\theta)]^m \prod_{i=1}^{n} f(x_i|\theta)$$

where F is the distribution function of f.

Of course if we knew the survivial times of those m censored patients was (z_{n+1}, .., z_{n+m}) we could write the complete data likelihood:

$$L(\theta|x, z) = \prod_{i=1}^{n} f(x_i|\theta) \prod_{i=n+1}^{n+m} f(z_i|\theta)$$

This suggests the EM algorithm:

- in the M step assume you know the z's and estimate θ
- in the E step assume you know θ and estimate the z's

Example Censored exponential survival times

Say $X_i \sim \text{Exp}(\theta)$, we have data X_1, .., X_n and we know that m observations were censored at T. Now one can find that

$$\hat{\theta} = \frac{n+m}{\sum x_i + \sum z_i} z_i = \frac{1}{\theta} + T$$

```
em.exp <- function(x, m, T) {
  theta.old <- 1/mean(x)
  repeat {
    z <- rep(1/theta.old+T, m)
    theta.new <- 1/mean(c(x, z))
    print(theta.new, 5)
    if(abs(theta.new-theta.old)<0.0001) break
```

```
    theta.old <- theta.new
  }
}
```

```
x <- rexp(1000, 0.1)
1/mean(x)
```

```
## [1] 0.09817537
```

```
em.exp(x, 0, 0)
```

```
## [1] 0.098175
```

```
x <- x[x<20]
m <- 1000 - length(x)
m
```

```
## [1] 137
```

```
1/mean(x)
```

```
## [1] 0.1435548
```

```
em.exp(x, m, 20)
```

```
## [1] 0.10303
## [1] 0.099193
## [1] 0.09869
## [1] 0.098621
```

```
x <- x[x<10]
m <- 1000 - length(x)
m
```

```
## [1] 375
```

```
1/mean(x)
```

```
## [1] 0.2318416
```

```
em.exp(x, m, 10)
```

```
## [1] 0.12402
## [1] 0.1056
## [1] 0.10003
## [1] 0.098091
## [1] 0.097382
## [1] 0.097119
## [1] 0.097021
```

Confidence Intervals

In the last section we studied the problem of *point estimation*. Generally one also wants to have some idea of the accuracy of this estimate, that is one wants to calculate the standard error. Most commonly this is done by finding a *confidence interval*.

In essence this provides bounds for the likely values of the parameter. One can control the level of "likely".

Coverage

the defining property of a confidence interval is its *coverage*, that is the probability that over many repeated experiments the true parameter lies inside the interval with the nominal level.

Sometime this can be shown analytically:

Example Mean of normal distribution

a $(1-\alpha)100\%$ confidence interval for the mean μ of a normal distribution with known standard deviation σ is given by

$$\bar{X} \pm z_{\alpha/2}\sigma/\sqrt{n}$$

here z_α are the $(1-\alpha)100\%$ quantiles of a standard normal distribution. They are found with

```
#alpha = 0.05
qnorm(1-0.05)
```

```
## [1] 1.644854
```

Now let's denote the density and the cumulative distribution function of a standard normal random variable by ϕ and Φ, respectively. Then

$$\begin{aligned}
&P(\mu \in I) = \\
&P(\bar{X} - z_{\alpha/2}\sigma/\sqrt{n} < \mu < \bar{X} + z_{\alpha/2}\sigma/\sqrt{n}) = \\
&P(\mu - z_{\alpha/2}\sigma/\sqrt{n} < \bar{X} < \mu + z_{\alpha/2}\sigma/\sqrt{n}) = \\
&P(-z_{\alpha/2}\sigma/\sqrt{n} < \bar{X} - \mu < z_{\alpha/2}\sigma/\sqrt{n}) = \\
&P(-z_{\alpha/2} < \frac{\bar{X}-\mu}{\sigma/\sqrt{n}} < z_{\alpha/2}) = \\
&1 - 2\Phi(z_{\alpha/2}) = 1 - 2(\alpha/2) = 1 - \alpha
\end{aligned}$$

Sometimes one has to use simulation to check coverage:

Example Binomial proportion

Let $X_1, .., X_n \sim Ber(\pi)$ then a $(1-\alpha)100\%$ confidence interval for π can be found with the *binom.test* command. Let's check that this method has correct coverage. (This is actually not necessary here because this routine implements a method by Clopper and Pearson (1934), which is exact and has coverage by the way it is constructed.)

```
B <- 10000
p <- 0.5; n <- 10
ci <- matrix(0, B, 2)
x <- rbinom(B, n, p)
for(i in 1:B) {
  ci[i, ] <- binom.test(x[i], n)$conf.int
}
sum(ci[, 1]<p & p<ci[, 2])/B
```

```
## [1] 0.9778
```

Notice that $0.9768 > 0.95$, so this method has *over-coverage*. This is not a good thing but it is ok, and in fact as we will see soon in this case unavoidable.

Of course this should now be done for all (or at least many) values of n and π, and this can take a while. In the case of a discrete random variable there is a quicker way, though. Notice that in the case above (n=10), x can take at most 11 values:

```
table(x)
```

```
## x
##    0    1    2    3    4    5    6    7    8    9   10
##    9   97  428 1172 2081 2386 2158 1100  453  104   12
```

and in fact we even know their probabilities:

```
round(dbinom(0:10, n, p), 4)
```

```
##  [1] 0.0010 0.0098 0.0439 0.1172 0.2051 0.2461 0.2051 0.1172 0.0439 0.0098
## [11] 0.0010
```

so in the simulation the same 11 intervals are calculated 10000 times.

Now if we denote the interval by $(L(x), U(x))$ if x is observed we have

$$\text{Cov}(\pi, n) = P(L(X) < \pi < U(X)) =$$
$$\sum_{x=0}^{n} I_{(L(x),U(x))}(\pi)\text{dbinom}(x, n, \pi)$$

and so we have a much faster way to find the coverage:

```
cov.bernoulli <- function(n, p) {
  tmp <- 0
  for(i in 0:n) {
    ci <- binom.test(i, n)$conf.int
```

```
    if(ci[1]<p & p<ci[2]) tmp <- tmp + dbinom(i, n, p)
  }
  tmp
}
round(cov.bernoulli(10, 0.5), 3)
```

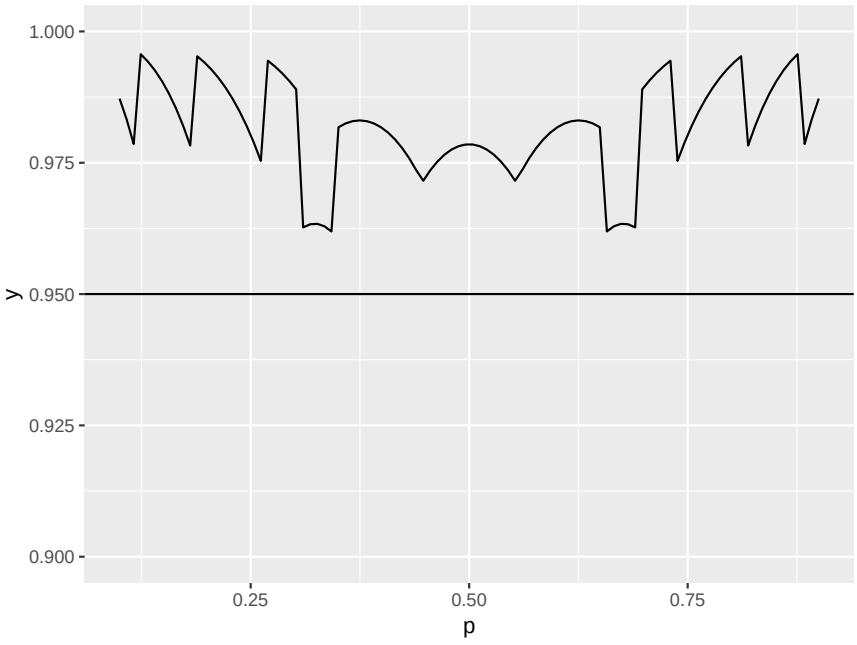
```
## [1] 0.979
```

and this is in fact no longer a simulation but an exact calculation!

Often we draw a *coverage graph*:

```
p <- seq(0.1, 0.9, length=100)
y <- p
for(i in 1:100)
  y[i] <- cov.bernoulli(10, p[i])
```

```
ggplot(data.frame(p=p, y=y), aes(p, y)) +
  geom_line() +
  geom_hline(yintercept = 0.95) +
  ylim(0.9, 1)
```



Notice the ragged appearance of the graph, which is quite typical for coverage graphs of discrete random variables.

Notice also that the actual coverage is always a bit higher than the nominal one. In fact it is well known that the Clopper-Pearson limits are somewhat *conservative* (aka a bit large). They are still generally a good choice because they don't depend on any assumptions.

Finding Confidence Interval

There are many ways to approach this problem, we will here discuss confidence intervals based on the method of maximum likelihood.

One major reason for the popularity of this method is the following celebrated theorem, due to Sir R.A. Fisher: under some regularity conditions

$$\sqrt{n}(\hat{\theta} - \theta) \sim N(0, \sqrt{I^{-1}})$$

where $N(\mu, \sigma)$ is the normal distribution and I is the *Fisher Information*, given by

$$I(\theta)_{ij} = -E\left[\frac{\partial^i \partial^j}{\partial\theta^i \partial\theta^j} \log f(x;\theta)\right]$$

and so it is very easy to find a $(1-\alpha)100\%$ confidence interval for (say) θ_i as

$$\hat{\theta} \pm z_{\alpha/2}\sqrt{I_{ii}^{-1}}$$

Example: Mean of a normal distribution, σ known.

we have

$$f(x;\mu) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}$$

Here we have only one parameter (μ), so the Fisher Information is given by

$$I(\mu) = -E\left[\frac{d^2 \log f(X;\mu)}{d\mu^2}\right]$$

and so we find

$$\begin{aligned}
&\frac{d}{d\mu}\log f(x;\mu) = \frac{1}{\sigma^2}(x-\mu)\\
&\frac{d^2}{d\mu^2}\log f(x;\mu) = -\frac{1}{\sigma^2}\\
&-E\left[\frac{d^2 f(X;\mu)}{d\mu^2}\right] = -E\left[-\frac{1}{\sigma^2}\right] = \frac{1}{\sigma^2}\\
&\sqrt{I(\mu)^{-1}} = \sqrt{\frac{1}{1/\sigma^2}} = \sigma\\
&\sqrt{n}(\hat{\mu} - \mu) \sim N(0, \sigma)\\
&\hat{\mu} \sim N(\mu, \sigma/\sqrt{n})
\end{aligned}$$

and we find the $(1-\alpha)100\%$ confidence interval to be

$$\hat{\mu} \pm z_{\alpha/2}\sigma/\sqrt{n}$$

this is of course the standard answer (for known σ).

Example: Binomial proportion

$$\log f(x;\pi) = x\log\pi + (1-x)\log(1-\pi)$$
$$\frac{d\log f}{d\pi} = \frac{x}{\pi} - \frac{1-x}{1-\pi}$$
$$\frac{d^2\log f}{d\pi^2} = -\frac{x}{\pi^2} - \frac{1-x}{(1-\pi)^2}$$
$$I(\pi) = -E\left[-\frac{X}{\pi^2} - \frac{1-X}{(1-\pi)^2}\right] =$$
$$\frac{EX}{\pi^2} + \frac{1-EX}{(1-\pi)^2} =$$
$$\frac{\pi}{\pi^2} + \frac{1-\pi}{(1-\pi)^2} =$$
$$\frac{1}{\pi} + \frac{1}{1-\pi} =$$
$$\frac{1}{\pi(1-\pi)}$$
$$\sqrt{I(\pi)^{-1}} = \sqrt{\pi(1-\pi)}$$
$$\hat{\pi} \sim N(\pi, \sqrt{\frac{\pi(1-\pi)}{n}})$$

and so a $(1-\alpha)100\%$ confidence interval would be given by

$$\hat{\pi} \pm z_{\alpha/2}\sqrt{\frac{\pi(1-\pi)}{n}}$$

But this does not help, we don't know π! The usual solution is to use a *plug-in* estimate:

$$\hat{\pi} \pm z_{\alpha/2}\sqrt{\frac{\hat{\pi}(1-\hat{\pi})}{n}}$$

and this is the standard textbook interval. Recall that we previously mentioned that this is NOT a good solution when n is small and π is either close to 0 or 1. This shows that this method gives us a way to find an interval but it does not guarantee that this interval is good.

If we apply it to our previous example we find a 95% confidence interval of

```
phat <- 235/567
round(phat + c(-1, 1)*qnorm(0.975)*sqrt(phat*(1-phat)/567), 3)
```

```
## [1] 0.374 0.455
```

Let's say for the moment that we couldn't do the math above, that is we have a case where we can't find the derivatives. We can however estimate it!

Notice that the Fisher Information is the (negative of the) expected value of the Hessian matrix of the log-likelihood function, and by the theorem of large numbers

$$\frac{1}{n}\sum H \to I$$

Now if we just replace I with the *observed* information we get:

```
binom.ll <- function(pi, y, n) {-log(dbinom(y, n, pi))}
fit <- nlm(binom.ll, 0.5, hessian = TRUE,y=235, n=567)
fit
```

```
## $minimum
## [1] 3.381578
##
## $estimate
## [1] 0.4144616
##
## $gradient
## [1] -2.178258e-06
##
## $hessian
##          [,1]
## [1,] 2336.051
##
## $code
## [1] 1
##
## $iterations
## [1] 4
```

a 95% confidence interval is given by

```
round(fit$estimate +
        c(-1, 1)*qnorm(1-0.05/2)/sqrt(fit$hessian[1, 1]), 3)
```

```
## [1] 0.374 0.455
```

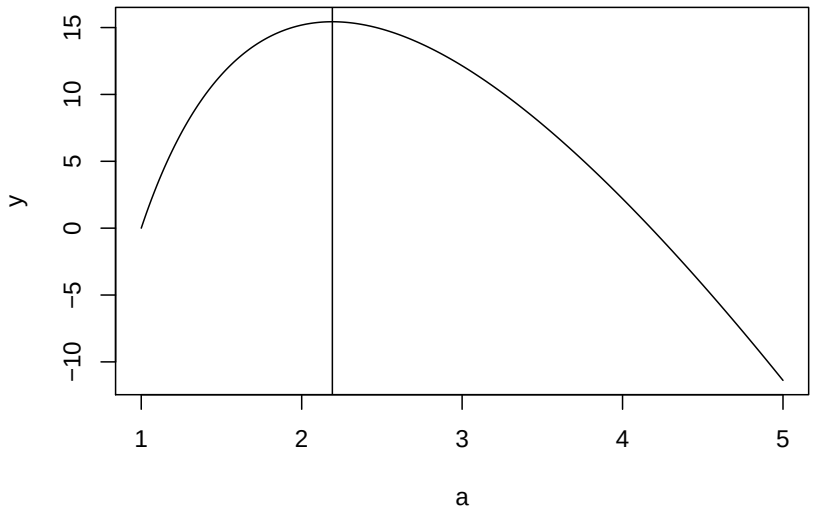
Let's put all of this together and write a "find a confidence interval" routine:

```
ci.mle <-
  function(f, # density
           param, # starting value for nlm
           dta, # data
           alpha=0.05, # desired confidence level
           rg, # range for plotting log-likelihood function
           do.graph=FALSE # TRUE if we want to look at the
```

```
                          # log-likelihood function
          )
{
  ll <- function(a, dta) { # log-likelihood function
     -sum(log(f(dta, a)))
  }
  tmp <- nlm(ll, param, hessian = TRUE, dta=dta)
  if(do.graph) {  # if you want to see the loglikelihood curve
     a <- seq(rg[1], rg[2], length=250)
     y <- rep(0, 250)
     for(i in seq_along(a))
        y[i] <- (-ll(a[i], dta))
     plot(a, y, type="l")
     abline(v=tmp$estimate)
  }
  if(length(param)==1) {
    ci <- tmp$estimate + c(-1, 1) *
      qnorm(1-alpha/2)/sqrt(tmp$hessian[1, 1])
    names(ci) <- c("Lower", "Upper")
  }
  else {
    I.inv <- solve(tmp$hessian) # find matrix inverse
    ci <- matrix(0, length(param), 2)
    colnames(ci) <- c("Lower", "Upper")
    if(!is.null(names(param)))
      rownames(ci) <- names(param)
    for(i in seq_along(param))
      ci[i, ] <- tmp$estimate[i] +
         c(-1, 1)*qnorm(1-alpha/2)*sqrt(I.inv[i, i])
  }
  list(mle=tmp$estimate, ci=ci)
}
```

Example: Beta(α, α)

```
x <- rbeta(100, 2.5, 2.5)
ci.mle(f = function(dta, a) {dbeta(dta, a, a)},
        param = 2.5,
        dta = x,
        rg = c(1, 5),
        do.graph = TRUE)
```




```
## $mle
## [1] 2.191294
##
## $ci
##    Lower    Upper
## 1.640830 2.741758
```

Example: Normal mean

Here we know the correct answer, so we can compare them:

```
x <- rnorm(25, 10, 1)
round(mean(x) + c(-1, 1)*qnorm(0.975)/sqrt(25), 2)
```

```
## [1]  9.47 10.26
```

```
tmp <- ci.mle(f = function(dta, a) {dnorm(dta, a)},
        param = 10,
        dta=x)
round(tmp$ci, 2)
```

```
## Lower Upper
##  9.47 10.26
```

More than one parameter

how about the multi dimensional parameter case?

Example: Normal, mean and standard deviation

```
x <- rnorm(200, 5.5, 1.8)
param <- c(5.5, 1.8)
names(param) <- c("mu", "sigma")
ci.mle(function(dta, a) {dnorm(dta, a[1], a[2])},
        param=param,
        dta=x)
```

```
## $mle
## [1] 5.457199 1.878691
##
## $ci
##          Lower    Upper
## mu    5.196831 5.717568
## sigma 1.694537 2.062845
```

Example: Beta(α, β)

```
x <- rbeta(200, 2.5, 3.8)
param <- c(2.5, 3.8)
names(param) <- c("alpha", "beta")
ci.mle(function(dta, a) {dbeta(dta, a[1], a[2])},
        param=param,
        dta=x)
```

```
## $mle
## [1] 2.514582 3.854701
##
## $ci
##          Lower    Upper
## alpha 2.046247 2.982916
## beta  3.111813 4.597589
```

Example: Old Faithful guyser

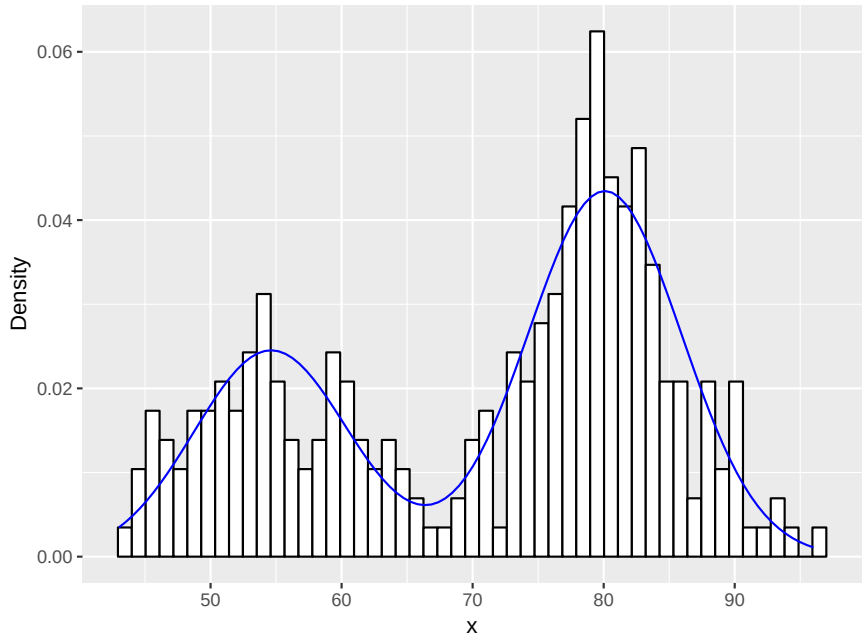
```
f <- function(dta, a)
  a[1]*dnorm(dta, a[2], a[3]) + (1-a[1])*dnorm(dta, a[4], a[5])
tmp <- ci.mle(f,
              param=c(0.35, 54, 5.4, 80, 5.9),
              dta=faithful$Waiting.Time)
tmp
```

```
## $mle
## [1]  0.360886 54.614854  5.871218 80.091067  5.867735
##
## $ci
##           Lower      Upper
```

```
## [1,]  0.2997968  0.4219752
## [2,] 53.2433113 55.9863960
## [3,]  4.8175843  6.9248509
## [4,] 79.1023581 81.0797757
## [5,]  5.0818840  6.6535865
```

and here is what this looks like:

```
bw <- diff(range(faithful$Waiting.Time))/50
ggplot(faithful, aes(x=Waiting.Time)) +
  geom_histogram(aes(y = ..density..),
    color = "black",
    fill = "white",
    binwidth = bw) +
    labs(x = "x", y = "Density") +
  stat_function(fun = f,
                colour = "blue",
                args=list(a=tmp$mle))
```



Now this sounds good, and it is, however this is based on having a *large enough* sample. In order to be sure ours is large enough one usually has to do some kind of coverage study.

Example: Hurricane Maria

How many people died due to Hurricane Maria when it struck Puerto Rico on September 20, 2017? Dr. Roberto Rivera and I tried to answer this question. In late November 2017 we got the following information from the Department of Health: during the time period September 1-19 there where 1582 deaths. During the period September 20 to October 31 there where 4319.

Now this means that during the time before the hurricane roughly $1587/19 = 83.5$ people died per day whereas in the 42 days after the storm it was $4319/42 = 102.8$, or $102.8 - 83.5 = 19.3$ more per day. This would mean a total of $42 \times 19.3 = 810.6$ deaths caused by Maria in this time period.

Can we find a 95% confidence interval? To start, the number of people who die on any one day is a Binomial random variable with n=3500000 (the population of Puerto Rico) and success(!!!) parameter π. Apparently before the storm we had $\pi = 83.5/3500000$. If we denote the probability to die due to Maria by μ, we find the probability model

$$f(x, y) = \text{dbinom}(1587, 19 \times 3500000, \pi)\text{dbinom}(4319, 42 \times 3500000, \pi + \mu)$$

Let's see:

```
N <- 3500000
f <- function(a) -log(dbinom(1582, 19*N, a[1])) -
    log(dbinom(4319, 42*N, a[1]+a[2]))
nlm(f, c(1582/19/3500000, (4319/42-1582/19)/3350000), hessian = TRUE)
```

```
## $minimum
## [1] 9.862462
##
## $estimate
## [1] 2.378947e-05 5.841843e-06
##
## $gradient
## [1] 0 0
##
## $hessian
##      [,1] [,2]
## [1,] -Inf -Inf
## [2,] -Inf -Inf
##
## $code
## [1] 1
##
## $iterations
## [1] 1
```

Oops, that didn't work. The problem is that the numbers for calculating the Hessian matrix become so small that it can not be done numerically.

What to do? First we can try to use the usual Poisson approximation to the Binomial:

```
f <- function(a)
  -log(dpois(1582, 19*a[1])) - log(dpois(4319, 42*(a[1]+a[2])))
res <- nlm(f, c(80, 20), hessian = TRUE)
res
```

```
## $minimum
```

```
## [1] 9.706561
##
## $estimate
## [1] 83.26316 19.57017
##
## $gradient
## [1] -3.840165e-12 -7.261486e-12
##
## $hessian
##           [,1]      [,2]
## [1,] 0.6365083 0.4083870
## [2,] 0.4083870 0.4084123
##
## $code
## [1] 1
##
## $iterations
## [1] 10
```

and now

```
round(42*(res$estimate[2] +
  c(-1, 1)*qnorm(1-0.05/2)*sqrt(solve(res$hessian)[2, 2])))
```

```
## [1]  607 1037
```

An even better solution is to do a bit of math:

$$\log\{\text{dpois}(x,\lambda)\} =$$
$$\log\left\{\frac{\lambda^x}{x!}e^{-\lambda}\right\} =$$
$$x\log(\lambda) - \log(x!) - \lambda$$

```
f <- function(a)
  -1582*log(19*a[1]) + 19*a[1] -
  4319*log(42*(a[1]+a[2])) + 42*(a[1]+a[2])
res <- nlm(f, c(20, 80), hessian = TRUE)
round(42*(res$estimate[2] +
  c(-1, 1)*qnorm(1-0.05/2)*sqrt(solve(res$hessian)[2, 2])))
```

```
## [1]  607 1037
```

By the way, in the paper we used a somewhat different solution based on the *profile likelihood*. In this case the answers are quite similar.

The paper is here

Notice the term $\pi + \mu$ above. This indicates that during the time after the storm there were

two sources of deaths, the "usual" one and Maria, and that they were additive. One can often consider different parametrizations for such problems. Maybe we should have used $(1+\epsilon)\pi$, indicating an increase in the base mortality rate after the storm. What parametrization is best is generally a question that should be answered by a subject expert.

UPDATE: After a long legal fight the Department of Health on June 1st 2018 finally updated the numbers:

Total de Defunciones por Mes

Mes	2015	2016	2017	2018
Jan	2744	2742	2894	2821
Feb	2403	2592	2315	2448
Mar	2427	2458	2494	2643
Apr	2259	2241	2392	2218
May	2340	2312	2390	1892
Jun	2145	2355	2369	0
Jul	2382	2456	2367	0
Aug	2272	2427	2321	0
Sep	2258	2367	**2928**	0
Oct	2393	2357	**3040**	0
Nov	2268	2484	2671	0
Dec	2516	2854	2820	0
Total	**28407**	**29645**	**31001**	**12022**

Notice how in general the number of deaths is much higher in the winter than in the summer. So it may be best to just use the data from February to November:

```
deaths.before <- 2315+2494+2392+2390+2369+2367+2321+2928-1317
deaths.after <- 1317+3040+2671
deaths.before/231 # Daily Deaths before Maria
```

```
## [1] 79.04329
```

```
deaths.after/72 # Daily Deaths after Maria
```

```
## [1] 97.61111
```

```
round(72*(deaths.after/72 - deaths.before/231)) # point estimate for total deaths due t
```

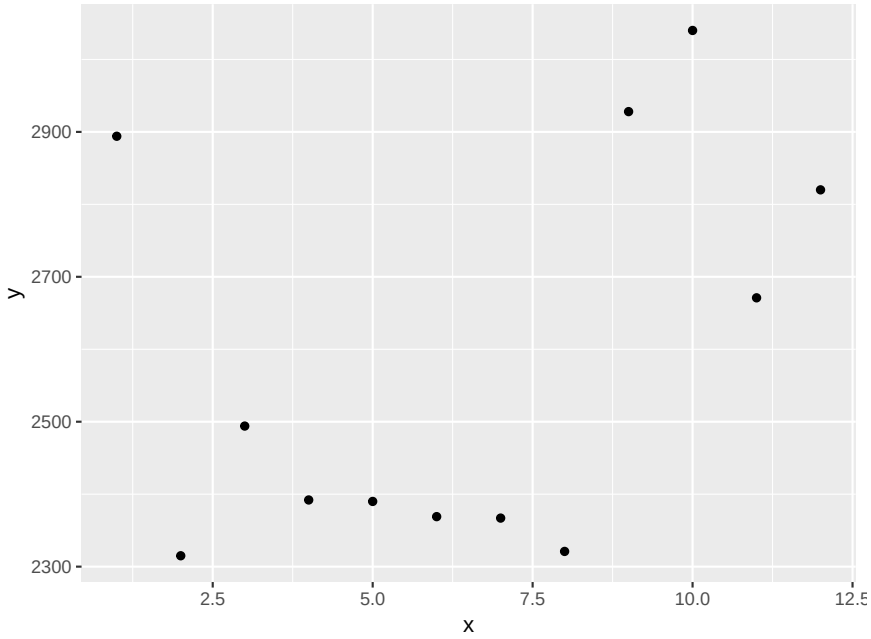
```
## [1] 1337
```

```
f <- function(a)
  -deaths.before*log(231*a[1]) + 231*a[1] -
  deaths.after*log(72*(a[1]+a[2])) + 72*(a[1]+a[2])
res <- nlm(f, c(20, 80), hessian = TRUE)
```

```
round(72*(res$estimate[2] +
  c(-1, 1)*qnorm(1-0.05/2)*sqrt(solve(res$hessian)[2, 2])))
```

```
## [1] 1153 1521
```

```
Months <- factor(unique(draft$Month), ordered=TRUE)
Deaths <- c(2894, 2315, 2494, 2392, 2390, 2369, 2367,
            2321, 2928, 3040, 2671, 2820)
ggplot(data=data.frame(x=1:12, y=Deaths), aes(x, y)) +
  geom_point()
```



There are other more subtle factors that one should consider. Doing so the best guess today is that around 1800 ± 200 people died in Puerto Rico during and after the Hurricane.

R Libraries

There are a number of packages available for maximum likelihood fitting:

```
library(maxLik)
x <- c(1582, 4319)
f <- function(param) {
  x[1]*log(19*param[1]) - 19*param[1] +
  x[2]*log(42*(param[1]+param[2])) - 42*(param[1]+param[2])
}
maxLik(logLik=f, start=c(20, 80))
```

```
## Maximum Likelihood estimation
## Newton-Raphson maximisation, 21 iterations
## Return code 2: successive function values within tolerance limit
```

```
## Log-Likelihood: 41906.11 (2 free parameter(s))
## Estimate(s): 83.17854 19.65237
```

In general these just provide wrappers for the routines mentioned above.

Hypothesis Testing

First off, hypothesis testing is a rather complicate business. We will here discuss just one method for developing a hypothesis tests. Also, there are many issues involved in testing that are not of a mathematical nature. Unless you have previously taken a course on Statistics I highly recommend that you read the discussion in ESMA 3101 - Hypothesis Testing.

General Problem Statement

As before we have the following general setup: we have data $x_1, .., x_n$ from some density $f(x|\theta)$. We want to test

$$H_0 : \theta \in \Theta_0 \text{ vs } H_a : \theta \notin \Theta_0$$

for some subset of the parameter space Θ_0.

Example: Coin Tossing

we flip a coin 1000 times and 549 heads. Is this a fair coin?

Here the parameter of interest θ is the probability of heads in one flip of this coin. Each flip is a Bernoulli trial, so we have

$$f(x|\theta) = \theta^x(1-\theta)^{1-x}, x = 0, 1$$

A fair coin has $\theta = 0.5$, so $\Theta_0 = \{0.5\}$ and we can write the hypotheses as

$$H_0 : \theta = 0.5 \text{ vs } H_a : \theta \neq 0.5$$

Of course this is a standard statistics problem, with a standard solution. Let's also consider two examples where the correct method is not obvious:

Example: Two Poisson means

A medical researcher carries out the following experiment: each day he gives 60 fruit flies either a poison type A or type B. In the evening he counts how many flies are dead. He finds:

```
kable.nice(poisons[c(1:5, 56:60), ])
```

	Dead	Poison
1	9	A
2	8	A
3	11	A
4	8	A
5	6	A
56	17	B
57	12	B
58	16	B
59	16	B
60	15	B

He wants to know whether the two poisons are equally effective.

Strictly speaking the number of dead flies follows a Binomial distribution with n trials and success probability π, but because n is large and π is small it is OK to treat them as Poisson rv's with rates λ_A and λ_B. Then the hypotheses are

$$H_0 : \lambda_A = \lambda_B \text{ vs } H_a : \lambda_A \neq \lambda_B$$

so here $\Theta_0 = \{(x, y) : 0 \leq x = y\}$

Example: Beta parameters

Below is a sample from the Beta(α, β) distribution and we wish to test

$$H_0 : \alpha \leq \beta \text{ vs } H_a : \alpha > \beta$$

so here we have $\Theta_0 = \{(x, y) : 0 \leq x \leq y\}$

```
beta.sample[c(1:5, 196:200)]
```

```
##  [1] 0.03 0.05 0.07 0.08 0.09 0.90 0.93 0.93 0.93 0.93
```

Critical Region of a Test

this is the set of points $(x_1, .., x_n)$ that if this were the observed data we would reject the null hypothesis.

Example: Coin Tossing

we want to test

$$H_0 : \theta = 0.5 \text{ vs } H_a : \theta \neq 0.5$$

the mle of π is $\hat{p} = x/n$ (here $549/1000 = 0.549$). Under H_0 $\hat{p}$ should be close to 0.5, so a sensible critical region would be of the form

$$|\hat{p} - 0.5| > c$$

for some number c. c is often called a *critical value*.

Example: Poisson means

the mle of a Poisson rate λ is the sample mean, so a test could be based on

$$|\bar{X}_A - \bar{X}_B| > c$$

Example: Beta parameters

Let $\hat{\alpha}, \hat{\beta}$ be the mle's of α and β, the a critical region could be

$$\hat{\alpha} - \hat{\beta} > c$$

Type I error, level of test α

How do we find a CR? This is done by first choosing α, the probability of the type I error. This in turn is the error to reject H_0 although it is true.

Example: Coin Tossing

From probability theory we know that

$$\sqrt{n}\frac{\hat{p} - 0.5}{\sqrt{p_0(1-p_0)}} \sim N(0,1)$$

so we find

$$
\begin{aligned}
&\alpha = P_{\pi=0.5}(|\hat{p} - 0.5| > c) = \\
&1 - P_{\pi=0.5}(|\hat{p} - 0.5| \leq c) = \\
&1 - P_{\pi=0.5}(-c \leq \hat{p} - 0.5 \leq c) = \\
&1 - P_{\pi=0.5}(-\frac{\sqrt{n}c}{\sqrt{p_0(1-p_0)}} \leq \sqrt{n}\frac{\hat{p} - 0.5}{\sqrt{p_0(1-p_0)}} \leq \frac{\sqrt{n}c}{\sqrt{p_0(1-p_0)}}) = \\
&1 - (1 - 2\Phi(\frac{\sqrt{n}c}{\sqrt{p_0(1-p_0)}})) \\
&\Phi(\frac{\sqrt{n}c}{\sqrt{p_0(1-p_0)}}) = \alpha/2 \\
&\frac{\sqrt{n}c}{\sqrt{p_0(1-p_0)}} = z_{\alpha/2} \\
&c = z_{\alpha/2}\sqrt{p_0(1-p_0)/n}
\end{aligned}
$$

If we use $\alpha = 0.05$ we find

```
cc <- qnorm(1-0.05/2)*sqrt(0.5*(1-0.5)/1000)
cc
```

```
## [1] 0.03098975
```

and so we would reject the null if

$$
\begin{aligned}
&|\hat{p} - 0.5| > 0.031 \\
&x/n < 0.5 - 0.031 \text{ or } x/n > 0.5 + 0.031 \\
&x < 469 \text{ or } x > 531
\end{aligned}
$$

we got x=549, so we would indeed reject the null.

Example: Poisson means

A CR could be constructed by noting that according to the *central limit theorem* the sample means have approximate normal distributions.

Example: Beta parameters

we don't even know how to find the mle's analytically, so this won't work.

The p value

The idea of the p value is as follows. Suppose we repeat the exact same experiment, how likely is it to observe the same outcome, or something even less likely, as what we just saw, assuming the null hypothesis is true?

If this probability is small (say $< \alpha$), then we just observed something very rare. Alternatively our assumption that the null hypothesis is true is false, and we should reject it!

Example: Coin Tossing

we got 549 heads in 1000 flips, so the p value would be the probability to flip a fair coin 1000 times and get 549 or more heads. Actually it would be $|\hat{p} - 0.5| \geq 0.049$ because under our alternative to few heads would also result in a rejection of the null.

Note that

$$|\hat{p}-0.5| \geq 0.049 \Leftrightarrow \hat{p}-0.5 < -0.049 \text{ or } \hat{p}-0.5 > 0.049 \Leftrightarrow x/n < 0.451 \text{ or } x/n > 0.549 \Leftrightarrow x < 451 \text{ or } x > 54$$

and so we can find the p value with

```
sum(dbinom(c(0:450, 550:1000), 1000, 0.5))
```

```
## [1] 0.001730536
```

$0.0017 < 0.05$, and so we do reject the null

Power of a Test

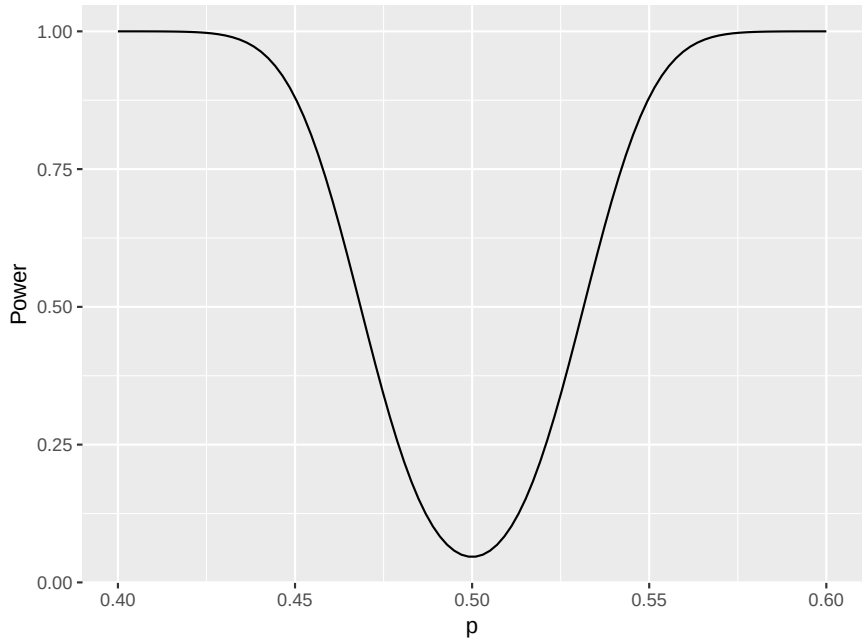
the power of a test is the probability to reject the null when it is false.

Under the null we know what θ is, namely θ_0. Under the alternative however there are many possibilities. So the power of a test is actually a function of the possible parameter values under the alternative hypothesis.

Example: Coin Tossing

our test is: *reject* H_0 *if* $x < 469$ *or* $x > 531$. So the power of the test is to do so if p is any value:

```
cr <- c(0:468, 532:1000)
p <- seq(0.4, 0.6, length=100)
power <- p
for(i in 1:100) {
  power[i] <- sum(dbinom(cr, 1000, p[i]))
}
ggplot(data.frame(p=p, y=power),
       aes(p, y)) +
  geom_line() +
  labs(x="p", y="Power")
```



so we see that if the π differs from 0.5 by more than 0.05, we will almost certainly be able to detect this with 1000 flips.

The power of a test has many uses:

- decide whether an experiment is worth doing. If it has a low power, it might be a waste of time.
- find what sample size would be required to have a reasonably powerful test.
- decide which of several methods is best, aka has the highest power.

Next we will discuss a general method for deriving hypothesis tests called the

Likelihood Ratio test

The *likelihood ratio test statistic* is defined by

$$\Lambda(\boldsymbol{x}) = \frac{\max_{\Theta_0} L(\theta)}{\max L(\theta)}$$

where L is the likelihood function.

From the definition it is clear that $0 \leq \Lambda \leq 1$.

In denominator the maximum is taken over all values of θ, so this is just like finding the maximum likelihood estimator!

Let's find out what we have in the three examples:

Example: Coin Tossing

we have previously found

$$L(\theta) = \theta^y (1-\theta)^{n-y}$$

where y was the number of successes and n the number of trials. Also, here $\Theta_0 = \{0.5\}$, so in the numerator the maximum is taken over just one value. The mle is y/n so

$$\Lambda(\boldsymbol{x}) = \frac{0.5^y (1-0.5)^{n-y}}{(y/n)^y (1-(y/n))^{n-y}} = \left(\frac{n}{2y}\right)^y \left(\frac{n}{2(n-y)}\right)^{n-y}$$

Note that under the null $\pi = 0.5$, so $y \sim n/2$, so $n/(2y) \sim 1$, $n/(2(n-y)) \sim 1$ and so $\Lambda \sim 1$.

Example: Poisson rates

Here we find

$$\begin{aligned} & f(x_1, .., x_n, y_1, .., y_n; \lambda_A, \lambda_B) = \\ & \prod_i \frac{\lambda_A^{x_i}}{x_i!} e^{-\lambda_A} \prod_i \frac{\lambda_B^{y_i}}{y_i!} e^{-\lambda_B} \\ & l(\lambda_A, \lambda_B) = \log f = \\ & \log \lambda_A \sum x_i - n\lambda_A + \log \lambda_B \sum y_i - n\lambda_B + K \\ & \frac{dl}{d\lambda_A} = \frac{\sum x_i}{\lambda_A} - n = 0 \end{aligned}$$

and so $\hat{\lambda}_A = \bar{X}$. Clearly also $\hat{\lambda}_B = \bar{Y}$. Now under H_0 $\lambda_A = \lambda_B =: \lambda$, and we find

$$l(\lambda) = \log \lambda \sum (x_i + y_i) - 2n\lambda$$

and so the value that maximizes the numerator is $\hat{\hat{\lambda}} = \frac{\sum (x_i + y_i)}{2n}$

Example: Beta parameters

the beta density is given by

$$f(x; \alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1} (1-x)^{\beta-1}$$

so the log likelihood function is given by

$$l(\alpha, \beta) = n \log \Gamma(\alpha+\beta) - n \log \Gamma(\alpha) - n \log \Gamma(\beta) + (\alpha-1) \sum \log x_i + (\beta-1) \sum \log(1-x_i)$$

now using calculus is certainly not going to work, so we need to use a numerical method:

```
beta.mle <- function(x) {
  log.like <- function(par) {
    -sum(log(dbeta(x, par[1], par[2])))
  }
  mle <- optim(c(1, 1), log.like)$par
  log.like <- function(par) {
    -sum(log(dbeta(x, par, par)))
  }
  mle.null <- optim(1, log.like)$par
  c(mle, mle.null)
}
beta.mle(beta.sample)
```

```
## [1] 1.974891 1.967810 1.971317
```

Wilks' Theorem

The usefulness of the likelihood ratio test statistic comes from the following famous theorem due to Wilks (1938):

Under some regularity conditions we find

$$-2\log\Lambda(\boldsymbol{X}) \sim \chi^2(p)$$

where p is the difference between the number of free parameters in the model and the number of free parameters under the null hypothesis.

Therefore we reject the null at the α level of significance if

$$-2\log\Lambda(\boldsymbol{X}) > \text{qchisq}(1-\alpha, p)$$

Notice that it usually more convenient to calculate

$$\begin{aligned} &-2\log\Lambda(\boldsymbol{X}) = \\ &-2\log\left\{\frac{l(\hat{\hat{\theta}})}{l(\hat{\theta})}\right\} = \\ &2\left\{\log l(\hat{\theta}) - \log l(\hat{\hat{\theta}})\right\} \end{aligned}$$

We said before that $0 < \Lambda < 1$, so $-2\log\Lambda \geq 0$. Also under H_0 $\Lambda \sim 1$, so $-2\log\Lambda \sim 0$

Example: Coin Tossing

$$-2\log\Lambda(\boldsymbol{x}) =$$
$$(-2)\log\left\{\left(\frac{n}{2y}\right)^{y}\left(\frac{n}{2(n-y)}\right)^{n-y}\right\} =$$
$$(-2)\left\{y\log\frac{n}{2y}+(n-y)\log\frac{n}{2(n-y)}\right\}$$

```
n <- 1000
y <- 549
lrt <- (-2)*(y*log(n/2/y)+(n-y)*log(n/2/(n-y)))
lrt
```

```
## [1] 9.619432
```

```
qchisq(1-0.05, 1)
```

```
## [1] 3.841459
```

and so we reject the null hypothesis, this does not seem to be a fair coin.

We can also easily find the p value of the test:

```
1-pchisq(lrt, 1)
```

```
## [1] 0.001925293
```

Example: Poisson rates

```
x <- poisons$Dead[poisons$Poison=="A"]
y <- poisons$Dead[poisons$Poison=="B"]
lrt <- 2*( sum(log(dpois(x, mean(x)))) +
           sum(log(dpois(y, mean(y)))) -
          sum(log(dpois(c(x, y), mean(c(x, y)))))
)
lrt
```

```
## [1] 12.03145
```

```
1-pchisq(lrt, 1)
```

```
## [1] 0.0005231045
```

Example: Beta parameters

```
p <- beta.mle(beta.sample)
lrt <- 2*(
  sum(log(dbeta(beta.sample, p[1], p[2]))) -
  sum(log(dbeta(beta.sample, p[3], p[3])))
```

```
)
lrt
```

```
## [1] 0.003293289
```

```
1-pchisq(lrt, 1)
```

```
## [1] 0.9542368
```

and here we have weak evidence that the null is false.

Bayesian Statistics

Our previous discussions focused on statistical methods that belong to the *Frequentist School*. There is however an entirely different approach to Statistics called *Bayesian*.

Prior and Posterior Distribution

Say we have a sample $\mathbf{x} = (x_1, .., x_n)$, iid from some probability density $f(.; \theta)$, and we want to do some inference on the parameter θ.

A Bayesian analysis begins by specifying a *prior* distribution $\pi(\theta)$. This prior is supposed to encode our knowledge of the parameter **before** an experiment is done. Then one uses Bayes' formula to calculate the *posterior distribution*:

$$f(\theta; \mathbf{x}) = f(\mathbf{x}; \theta)\pi(\theta)/m(\mathbf{x})$$

where $m(\mathbf{x})$ is the marginal distribution

$$m(\mathbf{x}) = \int .. \int f(\mathbf{x}|\theta)\pi(\theta)d\theta$$

Example: Binomial proportion, discrete prior

Let's say that we have two coins. One is a fair coin with $\pi = 0.5$ and the other is a loaded coin with $\pi = 0.6$. We randomly choose a coin and flip it 100 times. We get 58 heads. What can we say?

Now we have $X \sim Bin(100, \pi)$, or

$$P_\pi(X = 58) = \binom{100}{58}\pi^{58}(1-\pi)^{100-58} = K\pi^{58}(1-\pi)^{42}$$

the marginal is found using the *law of total probability*

$$
\begin{aligned}
&m(58) = \\
&P_{\pi=0.5}(X = 58)P(\pi = 0.5) + P_{\pi=0.6}(X = 58)P(\pi = 0.6) = \\
&\frac{1}{2}\left\{K0.5^{58}(1-0.5)^{42} + K0.6^{58}(1-0.6)^{42}\right\} = \\
&K/2\left\{0.5^{100} + 0.6^{58}0.4^{42}\right\}
\end{aligned}
$$

and the posterior distribution is given by

$$
\begin{aligned}
&P_{X=58}(\pi = 0.5) = \\
&\frac{P_{\pi=0.5}(X = 58)P(\pi = 0.5)}{m(58)} = \\
&\frac{K0.5^{58}(1-0.5)^{42}1/2}{K/2\left\{0.5^{100} + 0.6^{58}0.4^{42}\right\}} = \\
&\frac{0.5^{100}}{0.5^{100} + 0.6^{58}0.4^{42}}
\end{aligned}
$$

and a similar calculation shows

$$
\begin{aligned}
&P_{x=58}(\pi = 0.6) = \\
&\frac{0.6^{58}0.4^{42}}{0.5^{100} + 0.6^{58}0.4^{42}}
\end{aligned}
$$

or

```
round(c(0.5^100, 0.6^58*0.4^42)/(0.5^100+ 0.6^58*0.4^42), 4)
```

```
## [1] 0.231 0.769
```

and so the probability that this as the fair coin is 0.231.

Example: Binomial proportion, Uniform prior

Let's assume we have no idea what π might be, then a uniform distribution might make good sense as a prior:

$$
X \sim Bin(n, \pi), \pi \sim U[0, 1]
$$

now we find

$$
\begin{aligned}
&m(x) = \int_{-\infty}^{\infty} f(x|\mu)\pi(\mu)d\mu = \\
&\int_0^1 \binom{n}{x} p^x(1-p)^{n-x}1dp = \\
&K_1 \int_0^1 p^{(x+1)-1}(1-p)^{(n-x+1)-1}dp = K_2
\end{aligned}
$$

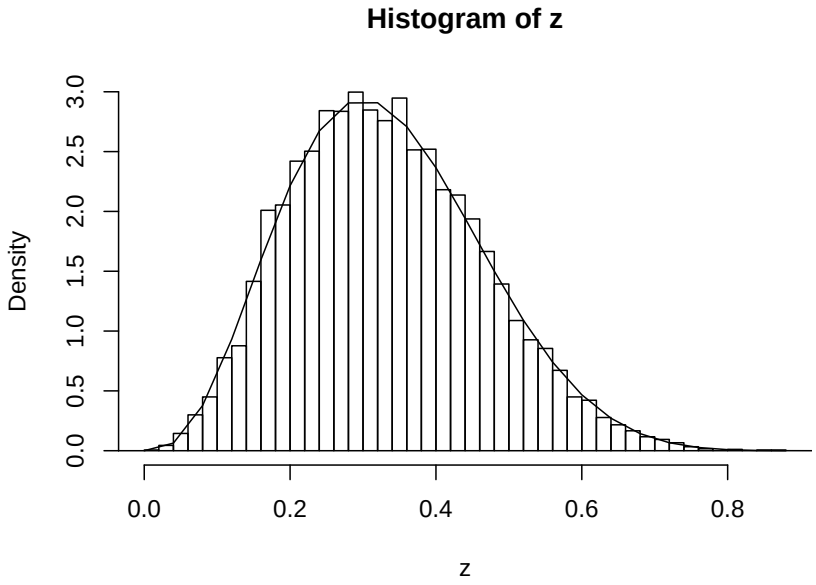
because this is (up to a constant) a Beta density which will integrate to 1. So

$$f(\theta;\mathbf{x}) = f(\mathbf{x};\theta)\pi(\theta)/m(\mathbf{x}) = K_3\pi^{(x+1)-1}(1-\pi)^{(n-x+1)-1}$$

and we find

$$\pi|X = x \sim \text{Beta}(x+1, n-x+1)$$

```
n <- 10
p <- runif(1e5)
x <-  rbinom(1e5, n, p)
x0 <- 3
z <- p[x==x0]
hist(z, 50, freq=FALSE)
curve(dbeta(x, x0+1, n-x0+1), -2, 2, add=T)
```



Example: Normal mean, normal prior

$X \sim N(\mu, \sigma)$ independent, σ known, $\mu \sim N(a, b)$.

Now

$$m(x) = \int_{-\infty}^{\infty} f(x|\mu)\pi(\mu)d\mu = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} \frac{1}{\sqrt{2\pi b^2}} e^{-\frac{1}{2b^2}(\mu-a)^2} d\mu$$

Note

$$
\begin{aligned}
&(x-\mu)^2/\sigma^2 + (\mu-a)^2/b^2 = \\
&x^2/\sigma^2 - 2x\mu/\sigma^2 + \mu^2/\sigma^2 + \mu^2/b^2 - 2a\mu/b^2 + a^2/b^2 \\
&(1/\sigma^2 + 1/b^2)\mu^2 - 2(x/\sigma^2 + a/b^2)\mu + K_1 = \\
&(1/\sigma^2 + 1/b^2)\left(\mu^2 - 2\frac{x/\sigma^2 + a/b^2}{1/\sigma^2 + 1/b^2}\mu\right) + K_2 = \\
&\frac{(\mu-d)^2}{c^2} + K_3
\end{aligned}
$$

where $d = \frac{x/\sigma^2+a/b^2}{1/\sigma^2+1/b^2}$ and $c = 1/\sqrt{1/\sigma^2 + 1/b^2}$

therefore

$$
m(x) = K_4 \int_{-\infty}^{\infty} e^{-\frac{1}{2c^2}(\mu-d)^2} d\mu = K_5
$$

because the integrand is a normal density with mean d and standard deviation c, so it will integrate to 1 as long as the constants are correct.

$$
\begin{aligned}
&f(\theta|\mathbf{x}) = f(\mathbf{x}|\theta)\pi(\theta)/m(\mathbf{x}) = \\
&K_6 e^{-\frac{1}{2c^2}(\mu-d)^2}
\end{aligned}
$$

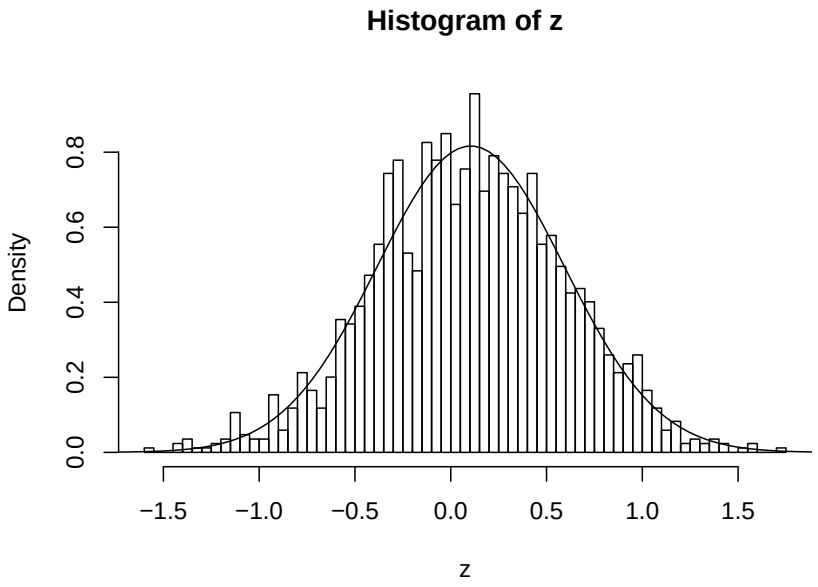
Notice that we don't need to worry about what exactly K_6 is, because the posterior will be a proper probability density, so K_6 will be what it has to be!

So we found

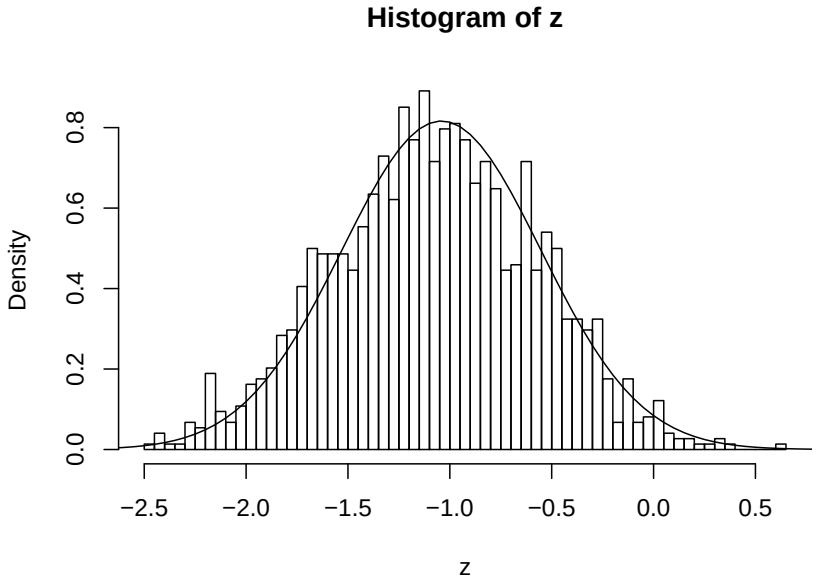
$$
\mu|X = x \sim N\left(\frac{x/\sigma^2 + a/b^2}{1/\sigma^2 + 1/b^2}, 1/\sqrt{1/\sigma^2 + 1/b^2}\right)
$$

Let's so a little simulation to see whether we got this right:

```
a <- 0.2 # just as an example
b <- 2.3
sigma <- 0.5
mu <- rnorm(1e5, a, b)
x <-  rnorm(1e5, mu, sigma)
x0 <- 0.1
cc <- 1/sqrt(1/sigma^2 + 1/b^2)
d <- (x0/sigma^2+a/b^2)/(1/sigma^2 + 1/b^2)
z <- mu[x>x0-0.05 & x<x0+0.05]
hist(z, 50, freq=FALSE)
curve(dnorm(x, d, cc), -2, 2, add=TRUE)
```




```
x0 <- (-1.1)
d <- (x0/sigma^2+a/b^2)/(1/sigma^2 + 1/b^2)
z <- mu[x>x0-0.05 & x<x0+0.05]
hist(z, 50, freq=FALSE)
curve(dnorm(x,d, cc), -3, 2, add=TRUE)
```



Note that one thing a frequentist and a Bayesian analysis have in common is the likelihood function.

Bayesian Inference

In a Bayesian analysis any inference is done from the posterior distribution. For example, point estimates can be found as the mean, median, mode or any other measure of central tendency.

Interval estimates (now called credible intervals) can be found using quantiles of the posterior distribution.

Example: Binomial proportion, uniform prior

we found the posterior distribution to be

$$\pi|X = x \sim \text{Beta}(x+1, n-x+1)$$

From probability theory we know that if $Y \sim \text{Beta}(\alpha, \beta)$ we have $EY = \frac{\alpha}{\alpha+\beta}$, so here we find $\hat{\pi} = \frac{y+1}{n+2}$. Recall that the frequentist solution (the mle) was $\hat{\pi} = \frac{y}{n}$.

Recall the survey of 567 people, 235 said they prefer Coke over Pepsi. A 95% credible interval for the true proportion is given by

```
ci <- qbeta(c(0.025, 0.975), 235+1, 567-235+1)
round(ci, 3)
```

```
## [1] 0.375 0.455
```

The frequentist confidence interval was

```
phat <- 235/567
round(phat + c(-1, 1)*qnorm(0.975)*sqrt(phat*(1-phat)/567), 3)
```

```
## [1] 0.374 0.455
```

and we see the two are quite close. This tends to be true as long as there is enough data.

Example: Normal mean, normal prior

say the following is a sample $x_1, .., x_n$ from a normal with standard deviation $\sigma = 2.3$:

```
dta.norm
```

```
##  [1] 2.2 2.5 2.6 2.7 3.9 4.1 4.5 4.5 4.6 4.8 5.1 5.1 5.2 6.7 6.9 7.6 7.6
## [18] 7.6 7.9 8.5
```

if we decide to base our analysis on the sample mean we have $\bar{X} \sim N(\mu, \sigma/\sqrt{n})$. Now if we use the posterior mean we find

$$E[\mu|X = x] = \frac{x/\sigma^2 + a/b^2}{1/\sigma^2 + 1/b^2}$$

now we need to decide what a and b to use. If we have some prior information we can use that. Say we expect a priori that $\mu = 5$, and of course we know $\sigma = 2.3$, then we could use $a = 5$ and $b = 3$:

```
d <- (mean(dta.norm)/(2.3^2/20) + 5/3^2)/(1/(2.3^2/20) + 1/3^2)
round(d, 2)
```

```
## [1] 5.22
```

A 95% credible interval is:

```
cc <- 1/sqrt(1/(2.3^2/20) + 1/3^2)
round(qnorm(c(0.025, 0.975), d, cc), 2)
```

```
## [1] 4.23 6.22
```

the standard frequentist solution would be

```
round(mean(dta.norm)+c(-1, 1)*qt(0.975, 12)*2.3/sqrt(20), 2)
```

```
## [1] 4.11 6.35
```

Example: Normal mean, Gamma prior

let's say that μ is a physical quantity, like the mean amount of money paid on sales. In that case it makes more sense to use a prior that forces μ to be non-negative. For example we could use $\mu \sim \text{Gamma}(\alpha, \beta)$. However, now we need to find

$$m(x) = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} \frac{1}{\Gamma(\alpha)\beta^\alpha} \mu^{\alpha-1} e^{-\mu/\beta} d\mu$$

and this integral does not exist. We will have to use numerical methods instead. Let's again find a point estimate based on the posterior mean. As prior we will use $\mu \sim \text{Gamma}(5, 1)$

```
fmu <- function(mu)
  dnorm(mean(dta.norm), mu, 2.3/sqrt(20))*
  dgamma(mu, 5, 1)
mx <- integrate(fmu, lower=0, upper=Inf)$value
posterior.density <- function(mu) fmu(mu)/mx
posterior.mean <-
  integrate(
    function(mu) {mu*posterior.density(mu)},
    lower = 0,
    upper = Inf)$value
round(posterior.mean, 2)
```

```
## [1] 5.17
```

how about a 95% credible interval? This we need to solve the equations

$$F(\mu) = 0.025,\ F(\mu) = 0.975$$

where F is the posterior distribution function. Again we need to work numerically. We can use a simple bisection algorithm:

```
pF <- function(t) integrate(posterior.density,
            lower=3, upper=t)$value
cc <- (1-0.95)/2
l <- 3
h <- posterior.mean
repeat {
    m <- (l+h)/2
    if(pF(m)<cc) l <- m
    else h <- m
    if(h-l<m/1000) break
}
left.endpoint <- m
h <- 8
l <- posterior.mean
repeat {
    m <- (l+h)/2
    if(pF(m)<1-cc) l <- m
    else h <- m
    if(h-l<m/1000) break
}
right.endpoint <- m
round(c(left.endpoint, right.endpoint), 2)
```

```
## [1] 4.19 6.16
```

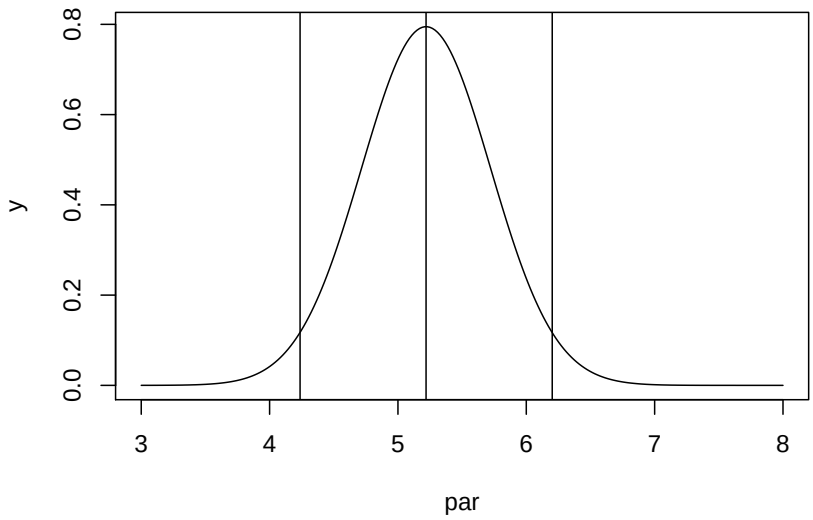
Let's generalize all this and write a routine that will find a point estimate and a $(1-\alpha)100\%$ credible interval for any problem with one parameter:

```
bayes.credint <- function(x, df, prior, conf.level=0.95, acc=0.001,
          lower, upper,  Show=TRUE) {
  if(any(c(missing(lower), missing(upper))))
    cat("Need to give lower and upper boundary\n")
  posterior.density <- function(par, x) {
    y <- 0*seq_along(par)
    for(i in seq_along((par)))
      y[i] <- df(x, par[i])*prior(par[i])/mx
    y
  }
  mx <- 1
  mx <- integrate(posterior.density,
          lower=lower, upper=upper, x=x)$value
  if(Show) {
     par <- seq(lower, upper, length=250)
     y <- posterior.density(par, x)
     plot(par, y, type="l")
```

```
  }
  f.expectation <- function(par, x) par*posterior.density(par, x)
  parhat <- integrate(f.expectation,
              lower=lower, upper=upper, x=x)$value
  if(Show) abline(v=parhat)
  pF <- function(t, x) integrate(posterior.density,
            lower=lower, upper=t, x=x)$value
  cc <- (1-conf.level)/2
  l <- lower
  h <- parhat
  repeat {
    m <- (l+h)/2
    if(pF(m, x)<cc) l <- m
    else h <- m
    if(h-l<acc*m) break
  }
  left.endpoint <- m
  h <- upper
  l <- parhat
  repeat {
    m <- (l+h)/2
    if(pF(m, x)<1-cc) l <- m
    else h <- m
    if(h-l<acc*m) break
  }
  right.endpoint <- m
  if(Show) abline(v=c(left.endpoint, right.endpoint))
  c(parhat, left.endpoint, right.endpoint)
}
```

Example: Normal mean, normal prior

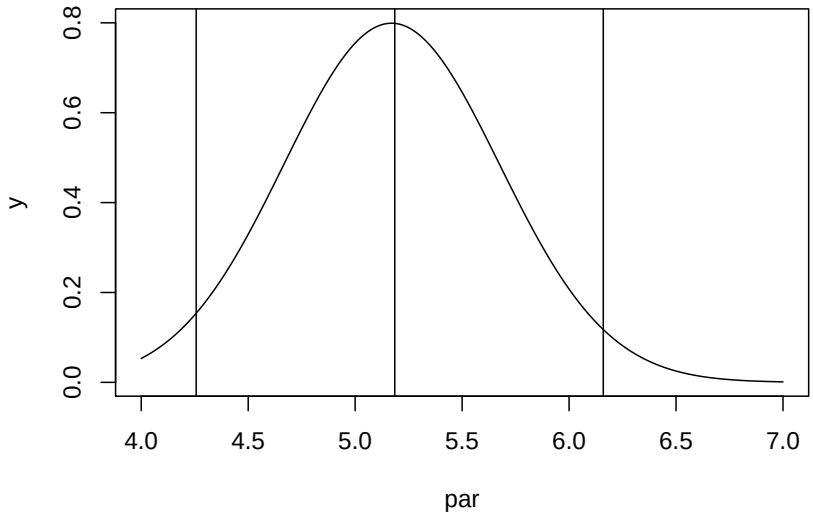
```
df <- function(x, par) dnorm(x, par, 2.3/sqrt(20))
prior <- function(par) dnorm(par, 5, 2.3)
round(bayes.credint(mean(dta.norm), df=df, prior=prior,
      lower=3, upper=8, Show=T), 2)
```




```
## [1] 5.22 4.24 6.20
```

Example: Normal mean, Gamma prior

```
df <- function(x, par) dnorm(x, par, 2.3/sqrt(20))
prior <- function(par) dgamma(par, 5, 1)
round(bayes.credint(mean(dta.norm), df=df, prior=prior,
      lower=4, upper=7, Show=TRUE), 2)
```



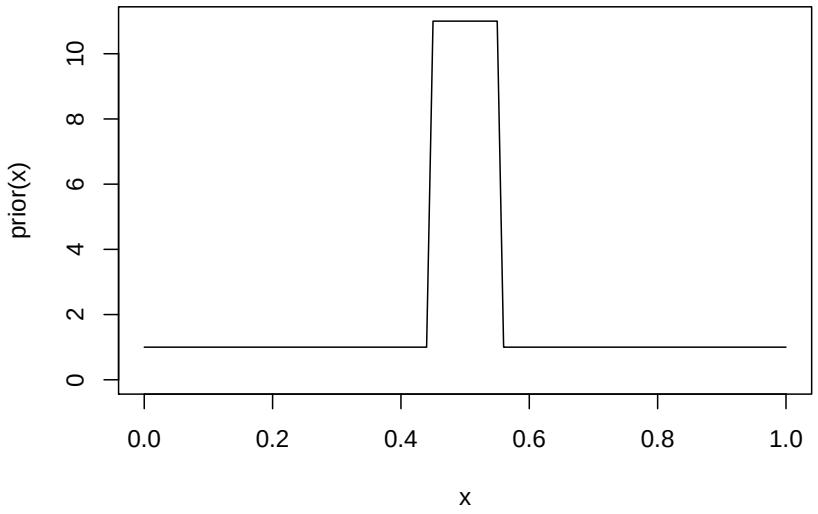

```
## [1] 5.19 4.26 6.16
```

Example: Binomial proportion, Lincoln's hat prior

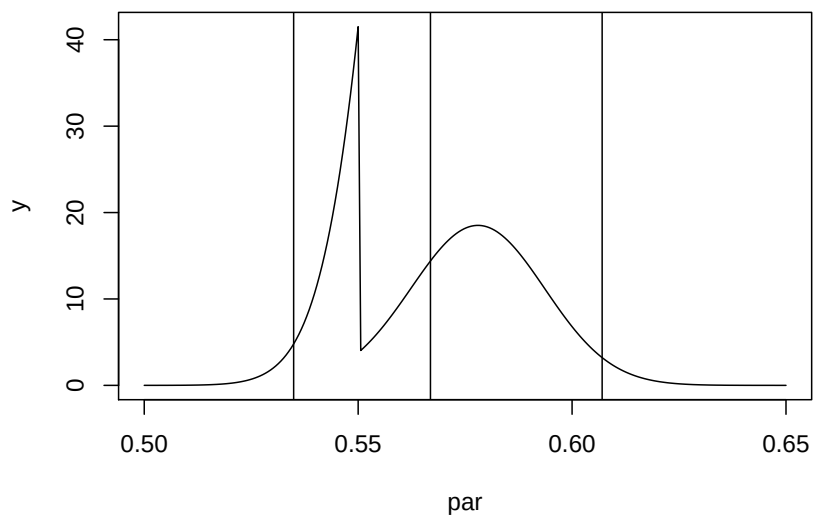
Say we pick a coin from our pocket. We flip it 1000 time and get 578 heads. We want to find a 95% credible interval for the proportion of heads.

What would be good prior here? We might reason as follows: on the one had we are quite sure that indeed it is an "almost" fair coin. On the other hand if it is not a fair coin we really don't know how unfair it might be. We can encode this in the *Lincoln's hat* prior:

```
prior <- function(x) dunif(x) + dunif(x, 0.45, 0.55)
curve(prior, 0, 1, ylim=c(0, 11))
```




```
df <- function(x, par) dbinom(x, 1000, par)
round(bayes.credint(x=578, df=df, prior=prior, acc=0.0001,
      lower=0.5, upper=0.65, Show=TRUE), 3)
```



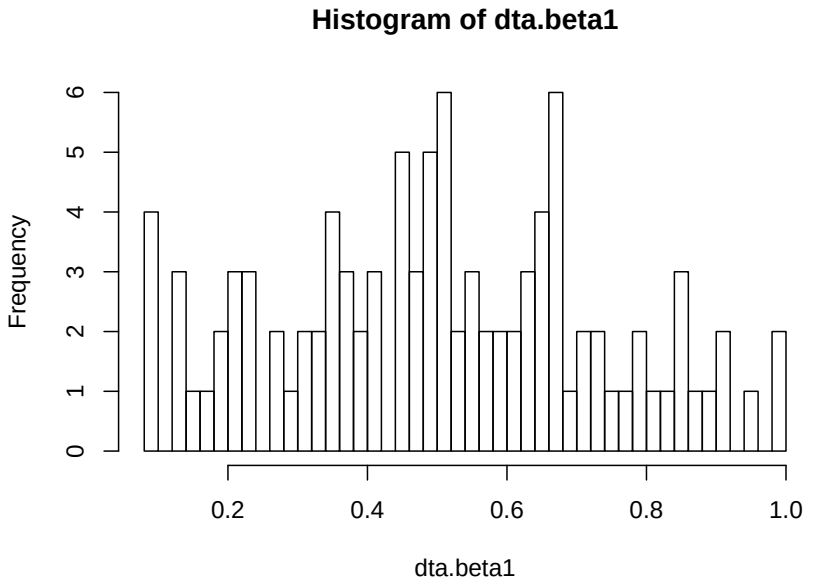

```
## [1] 0.567 0.535 0.607
```

So, have we just solved the Bayesian estimation problem for one parameter?

Example: Beta density, Gamma prior

consider the following sample:

```
dta.beta1 <- round(rbeta(100, 2, 2), 3)
hist(dta.beta1, 50)
```



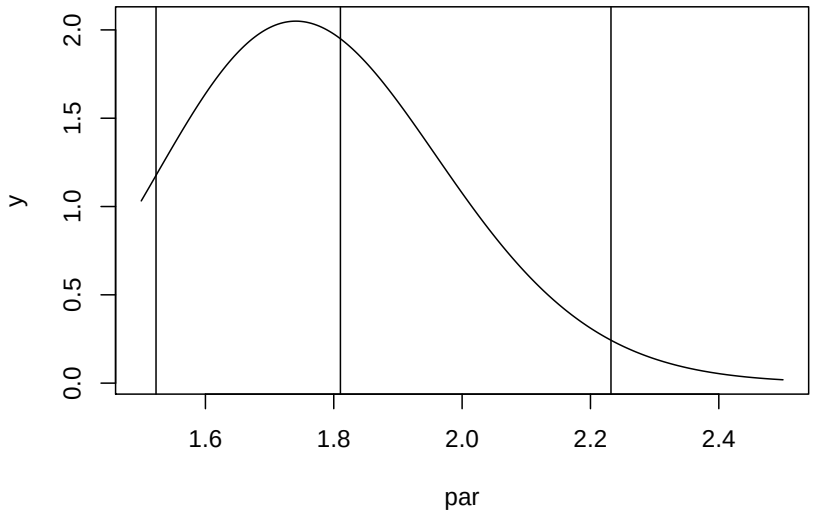
Let's say we know that this is from a Beta(a, a) distribution and we want to estimate a. As a prior we want to use Gamma(2, 1)

Now what is df? Because this is an independent sample we find

$$f(x, a) = \prod_{i=1}^{n} \text{dbeta}(x_i, a, a)$$

so

```
df <- function(x, par) prod(dbeta(x, par, par))
prior <- function(par) dgamma(par, 2, 1)
round(bayes.credint(dta.beta1, df=df, prior=prior,
      lower=1.5, upper=2.5, Show=TRUE), 2)
```




```
## [1] 1.81 1.52 2.23
```

so far, so good. But now

```
dta.beta2 <- round(rbeta(10000, 2, 2), 3)
bayes.credint(dta.beta2, df=df, prior=prior,
      lower=1.5, upper=2.5, Show=FALSE)
```

```
## Error in integrate(posterior.density, lower = lower, upper = upper, x = x): non-finit
```

Why does this not work? The problem is that the values is $\prod_{i=1}^{n} \text{dbeta}(x_i, a, a)$ get so small that R can't handle them anymore!

Occasionally one can avoid this problem by immediately choosing a *statistic* T(x), aka a function of the data, so that T(X) has a distribution that avoids the product. That of course is just what we did above by going to $\bar{X}$ in the case of the normal! In fact, it is also what we did in the case of the Binomial, because we replace the actual data (a sequence of Bernoulli

trials) with their sum. It is however not clear what one could use in the case of the Beta distribution.

Can we generalize this to more than one parameter? In principle yes, but in practice no, at least not if the number of parameters is much more than 3 or 4. The main problem is the calculation of the marginal $m(x)$, because numerical integration in higher-dimensional spaces is very difficult. In that case a completely different approach is used, namely sampling from the posterior distribution using so called MCMC (Markov Chain Monte Carlo) algorithms.

Another difficulty arises in the choice of priors. There are a number of different approaches known for low-dimensional problems, however these can fail badly in higher dimensions.

There are a number of R packages that allow Bayesian analysis, such as JAGS, OpenBUGS, WinBUGS and Stan. However, we don't have enough time to discuss these.

Simulation

Basic Idea

We already used simulation in a couple of situations. In this section we will take a closer look.

Example: Binomial proportion

Let's start with a simple example: say we flip a coin 100 times and get 62 heads. Is this a fair coin?

Of course we discussed this problem before and there is a standard solution: we want to test

$$H_0 : \pi = 0.5 \text{ vs. } H_a : \pi \neq 0.5$$

the test is done with

```
binom.test(62, 100)
```

```
##
##  Exact binomial test
##
## data:  62 and 100
## number of successes = 62, number of trials = 100, p-value =
## 0.02098
## alternative hypothesis: true probability of success is not equal to 0.5
## 95 percent confidence interval:
##  0.5174607 0.7152325
## sample estimates:
## probability of success
```

```
##                     0.62
```

and with a p value of 0.021 we would reject the null hypothesis at the 5% level.

But let's assume for the moment we do not know this solution. What could we do?

we can simulate flipping a fair coin and counting the number of heads with

```
smpl <- sample(c("Heads", "Tails"), size=100, replace=TRUE)
x <- sum(smpl=="Heads")
x
```

```
## [1] 45
```

Now we can do this many times:

```
B <- 1e4
x <- rep(0, B)
for(i in 1:B) {
  smpl <- sample(c("Heads", "Tails"),
                 size=100, replace=TRUE)
  x[i] <- sum(smpl=="Heads")
}
table(x)
```

```
## x
##  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50
##   3   5  10  17  29  39  80 101 166 231 319 410 509 548 669 720 802 797
##  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  68  69
## 797 735 665 535 468 407 310 211 149  97  67  49  27  11   6   7   2   2
```

and we can find the p-value with

```
(sum(x<=38)+sum(x>=62))/B
```

```
## [1] 0.0207
```

So here we did the test via simulation, without any theory!

Simulation Error

Every time we run a simulation, the computer generates different data and so the answer comes out a bit different as well. But how different?

At least in one case there is a simple way to estimate this simulation error. In the example above, each simulation run is essentially a Bernoulli trial ($x \leq 38, x \geq 62$ or not). An \$95% confidence interval for the true proportion is given by

$$\hat{\pi} \pm 2\sqrt{\hat{\pi}(1-\hat{\pi})/n}$$

Example: Binomial proportion, continued

```
0.02 + c(-2, 2)*sqrt(0.02*0.98/1e4)
```

```
## [1] 0.0172 0.0228
```

so the true p value is between 0.017 and 0.023, in either case < 0.05 and we would reject the null.

Example Exponential rate

the sample below is assumed to come from an exponential distribution rate λ. We wish to test

$$H_0 : \lambda = 1 \text{ vs. } H_a : \lambda > 1$$

```
exp.data
```

```
##  [1] 0.02 0.02 0.04 0.04 0.04 0.06 0.11 0.12 0.12 0.13 0.13 0.15 0.17 0.18
## [15] 0.19 0.21 0.30 0.30 0.38 0.39 0.41 0.41 0.49 0.51 0.51 0.54 0.59 0.60
## [29] 0.61 0.62 0.63 0.67 0.68 0.75 0.78 0.79 0.80 0.83 0.91 1.09 1.44 1.64
## [43] 1.68 1.88 2.05 2.06 2.20 3.02 3.11 4.70
```

here is a solution via simulation. We know from theory that the mle of λ is $1/\hat{X}$, so

```
B <- 1e4
sim.data <- rep(0, B)
for(i in 1:B) {
  sim.data[i] <-
    1/mean(rexp(length(exp.data), 1))
}
sum(sim.data>1/mean(exp.data))/B
```

```
## [1] 0.0733
```

Example Normal mean

below we have data from a normal distribution and we want to test

$$H_0 : \mu = 10 \text{ vs. } H_a : \mu > 10$$

```
norm.data
```

```
##  [1]  8.24  8.87  8.93  9.05  9.28  9.28  9.49  9.69  9.74  9.82  9.85
## [12] 10.15 10.45 10.47 10.65 11.15 11.31 11.44 11.87 12.40
```

Again we want to use simulation and we can use the sample mean as our test statistic, but here we have an additional problem: we will of course generate data from a normal distribution with mean 10, but what should we use as the standard deviation? It is not defined by the null hypothesis.

There is an obvious answer: use the sample standard deviation. It is not clear however if that is indeed legitimate.

```
B <- 1e4
n <- length(norm.data)
p <- sd(norm.data)
sim.data <- rep(0, B)
for(i in 1:B) {
  sim.data[i] <-
    mean(rnorm(n, 10, p))
}
sum(sim.data>mean(norm.data))/B
```

```
## [1] 0.3291
```

how does this compare to the standard answer?

```
t.test(norm.data, mu=10, alternative = "greater")$p.value
```

```
## [1] 0.3348097
```

pretty close, certainly within simulation error.

sometimes one varies the standard deviation a bit in the simulation step. R does not have a method for finding confidence intervals for variances, but here is how to find them:

```
v <- var(norm.data)
lower <- v*19/qchisq(0.05/2, 19,
                             lower.tail = FALSE)
upper <- v*19/qchisq(1-0.05/2, 19,
                             lower.tail = FALSE)
sqrt(c(lower = lower, upper = upper))
```

```
##    lower    upper
## 0.835776 1.605162
```

and so we can run the simulation also this way:

```
B <- 1e4
n <- length(norm.data)
sim.data <- rep(0, B)
for(i in 1:B) {
  sim.data[i] <-
    mean(rnorm(n, 10, runif(1, 0.836, 1.605)))
}
sum(sim.data>mean(norm.data))/B
```

```
## [1] 0.3475
```

In essence this has a bit of a Bayesian flavor, we just introduced a prior for σ!

Permutation Tests

There is a class of methods essentially built on simulation. Here is an

Example: Equal means

below are two data sets. Do they came from the same type of distribution but with different means?

So there is a distribution F, and we can assume without loss of generality that $E[X_F = 0]$. There are μ_1 and μ_2 such that

$$X_1 - \mu_1, .., X_n - \mu_1 \sim F$$
$$Y_1 - \mu_2, .., Y_m - \mu_2 \sim F$$

and we have the hypotheses

$$H_0 : \mu_1 = \mu_2 \text{ vs. } H_a : \mu_1 \neq \mu_2$$

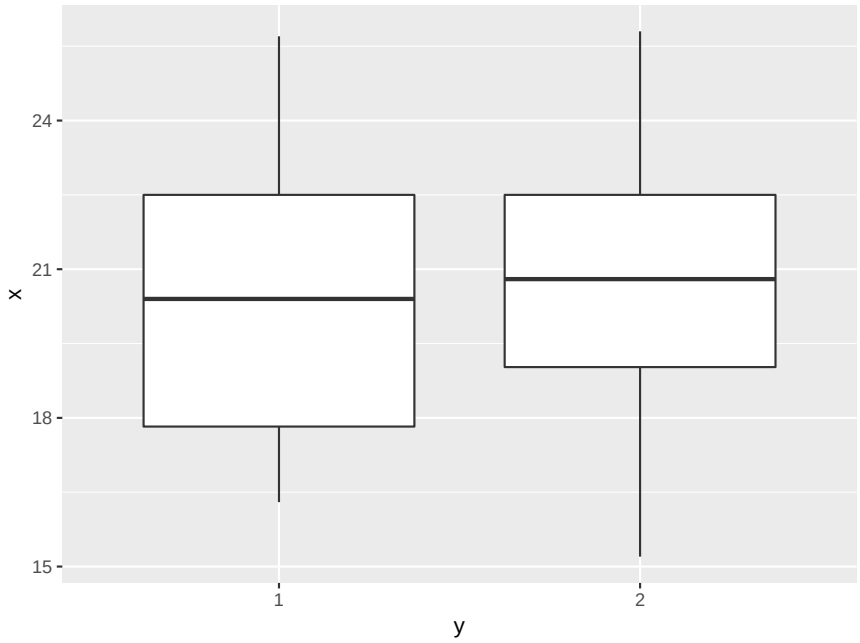
```
norm1.data
```

```
##  [1] 16.3 16.5 16.6 17.3 17.6 17.9 19.5 19.5 19.9 20.2 20.6 20.9 21.1 22.1
## [15] 22.4 22.8 22.8 23.3 24.1 25.7
```

```
norm2.data
```

```
##  [1] 15.2 16.2 17.1 17.2 17.4 17.7 18.4 19.0 19.1 19.2 19.2 19.3 19.8 20.7
## [15] 20.7 20.9 21.0 21.1 21.5 21.5 21.8 21.9 22.7 22.9 23.2 24.8 24.9 25.1
## [29] 25.3 25.8
```

```
df <- data.frame(
  x=c(norm1.data, norm2.data),
  y=rep(c("1", "2"), c(20, 30)))
ggplot(df, aes(y, x)) +
  geom_boxplot()
```



a reasonable test statistics would be

$$T = \frac{\bar{X} - \bar{Y}}{\sqrt{[(n-1)s_X^2 + (m-1)s_Y^2]/(n+m-2)}}$$

because under the null $E[\bar{X} - \bar{Y}] = 0$ and the denominator is the usual estimator of the standard deviation (called the *pooled* standard deviation).

```
x <- norm1.data
y <- norm2.data
T0 <- (mean(x)-mean(y))/sqrt((19*var(x)+29*var(y))/49)
T0
```

```
## [1] -0.1198418
```

Now the idea is as follows: under the null hypothesis all the X's and Y's are an independent sample from the same distribution. In this case the order is of no consequence, any reordering should give an equally valid answer:

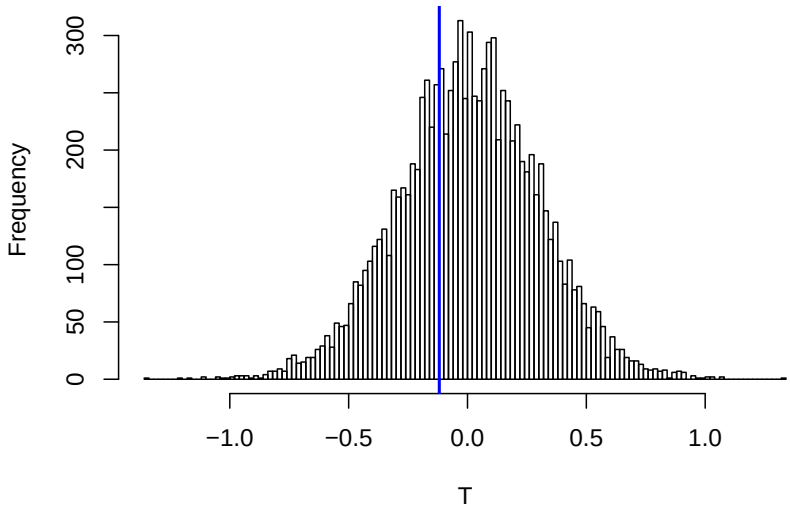
```
z <- sample(c(norm1.data, norm2.data)) #permutation
x <- z[1:20]
y <- z[21:50]
(mean(x)-mean(y))/sqrt((19*var(x)+29*var(y))/49)
```

```
## [1] -0.3628009
```

This is a perfectly legitimate value of T **IF** the null hypothesis is true.

Let's repeat this many times. In fact let's write a function that does it:

```
perm.test <- function(x, y,  B = 1e4, Show=FALSE) {
  n <- length(x)
  m <- length(y)
  T0 <- (mean(x) - mean(y))/
      sqrt(((n-1)*var(x)+(m-1)*var(y))/(n+m-2))
  xy <- c(x, y)
  T <- rep(0, B)
  for(i in 1:B) {
    z <- sample(xy)
    x <- z[1:n]
    y <- z[(n+1):(n+m)]
    T[i] <- (mean(x) - mean(y))/
      sqrt(((n-1)*var(x)+(m-1)*var(y))/(n+m-2))
  }
  if(Show) {
    hist(T, 100, main="")
    abline(v=T0, lwd=2, col="blue")
  }
  sum(abs(T)>abs(T0))/B
}
perm.test(norm1.data, norm2.data, Show=TRUE)
```

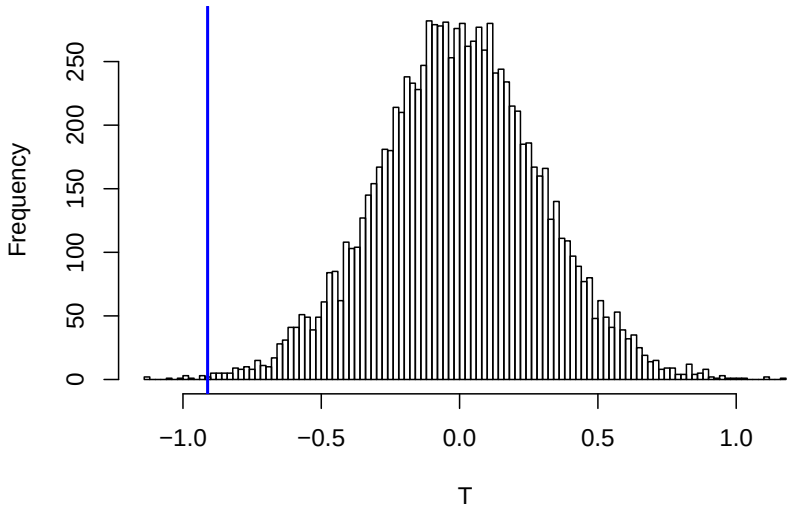



```
## [1] 0.6822
```

and we see that the value of T for the real data is in no way unusual.

Let's do this again for some data where the means are indeed different:

```
perm.test(x=rnorm(20, 10, 5),
          y=rnorm(30, 15, 5),
          Show = TRUE)
```




```
## [1] 0.0023
```

In our case we also know that the F is a normal distribution. In this case there is of course a classic solution, the so-called *two-sample-t test*:

```
t.test(norm1.data, norm2.data)$p.value
```

```
## [1] 0.6807857
```

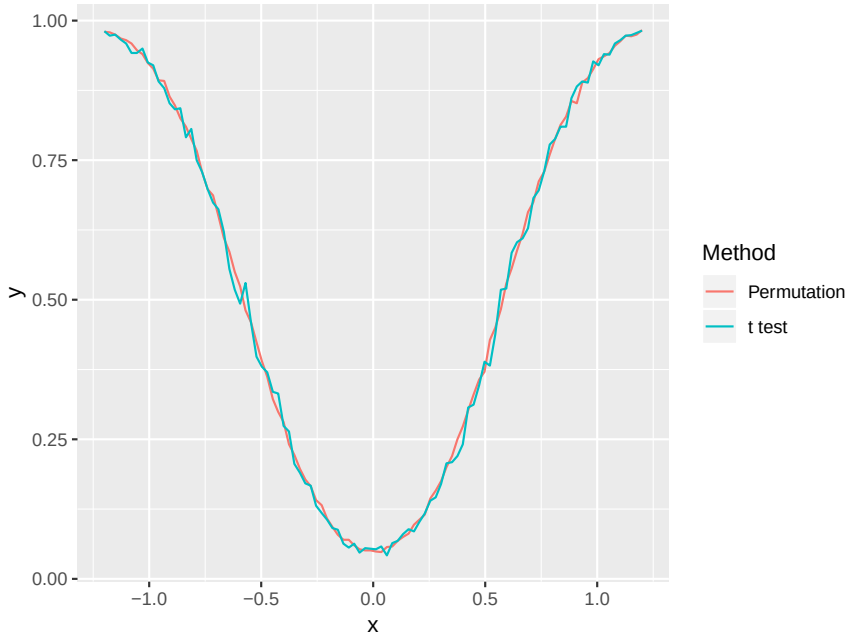
and notice that its p value is almost the same as the permutations tests!

How good a test is this? Let's find out:

```
pwr.sim <- function(mu2, n=20, m=30, B = 2500) {
  pvals <- matrix(0, B, 2)
  colnames(pvals) <- c("Permutation", "t test")
  for(i in 1:B) {
    x <- rnorm(n)
    y <- rnorm(m, mu2)
    pvals[i, 1] <- perm.test(x, y, B=2500)
    pvals[i, 2] <- t.test(x, y)$p.value
  }
  pvals
}
```

Let's do a whole power curve! This takes a while to run, though, so the result is saved as *pwr.tbl*

```
df <- data.frame(x=rep(pwr.tbl[, 1], 2),
       y=c(pwr.tbl[, 2], pwr.tbl[, 3]),
       Method=rep(c("Permutation", "t test"), each=100))
ggplot(data=df, aes(x, y, color=Method)) +
  geom_line()
```



Can't tell the difference? In fact the two methods have just about the same power, even so one depends strongly on the normal assumption, whereas the other one works without it.

This test was first discussed by Fisher in the 1930's, but until fairly recently it was not doable. Nowadays it should be considered the go-to test for this kind of situation.

The Bootstrap

In the previous section we used simulation from the true distribution to derive statistical methods. But what do we do if we don't know the distribution?

The idea of the Bootstrap is rather strange: say we have some data from some distribution and we want to use it to estimate some parameter θ. We have a formula (a *statistic*) $T(x_1, .., x_n)$. What is the standard error in this estimate? That is, what is $sd[T(X_1, .., X_n)]$?

Sometimes we can do this mathematically: Let's assume that the X_i are *iid* (independent and identically distributed) and we are interested in the mean. Let's write $\mathbf{X} = (X_1, .., X_n)$, then

$$\theta = E[X_1]$$
$$T\mathbf{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$
$$E[T\mathbf{X}] = E[\frac{1}{n}\sum_{i=1}^{n} X_i] = \frac{1}{n}\sum_{i=1}^{n} E[X_i] = \frac{1}{n}n\theta = \theta$$
$$Var[T\mathbf{X})] = E\left[\left(\frac{1}{n}\sum_{i=1}^{n} X_i - \theta\right)^2\right] =$$
$$\frac{1}{n^2}E\left[\left(\sum_{i=1}^{n} X_i - n\theta\right)^2\right] =$$
$$\frac{1}{n^2}E\left[\left(\sum_{i=1}^{n} (X_i - \theta)\right)^2\right] =$$
$$\frac{1}{n^2}E\left[\sum_{i,j=1}^{n} (X_i - \theta)\,(X_j - \theta)\right] =$$
$$\frac{1}{n^2}\left[\sum_{i=1}^{n} E(X_i - \theta)^2 + \sum_{i,j=1, i\neq j}^{n} E(X_i - \theta)(X_j - \theta)\right] =$$
$$\frac{1}{n^2}\left[nE(X_1 - \theta)^2 + 0\right] = \frac{1}{n}Var[X_1]$$

because

$$E(X_i - \theta)^2 = E(X_1 - \theta)^2$$

(identically distributed) and

$$E(X_i - \theta)(X_j - \theta) = E(X_i - \theta)E(X_j - \theta) = 0$$

because of independence.

But let's say that instead of the mean we want to estimate θ with the median. Now what is the $sd[\text{median}(X_1, .., X_n)]$? This can still be done analytically, but is already much more complicated.

It would of course be easy if we could simulate from the distribution:

```
sim.theta <- function(B=1e4, n, mu=0, sig=1) {
  x <- matrix(rnorm(B*n, mu, sig), B, n)
  xbar <- apply(x, 1, mean)
  med <- apply(x, 1, median)
  round(c(sig/sqrt(n), sd(xbar), sd(med)), 3)
}
sim.theta(n=25)
```

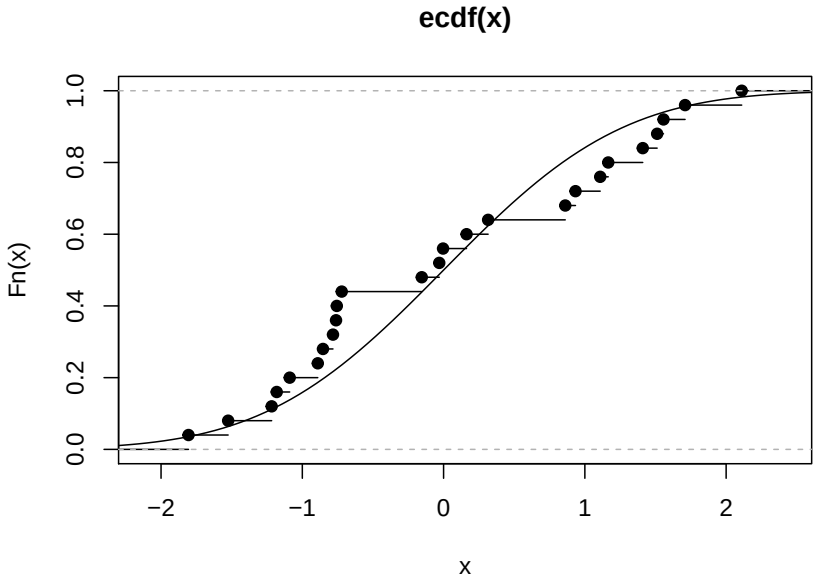
```
## [1] 0.200 0.198 0.247
```

But what do we do if didn't know that the data comes from the normal distribution? Then we can't simulate from F. We can, however simulate from the next best thing, namely the *empirical distribution function edf* $\hat{F}$. It is defined as:

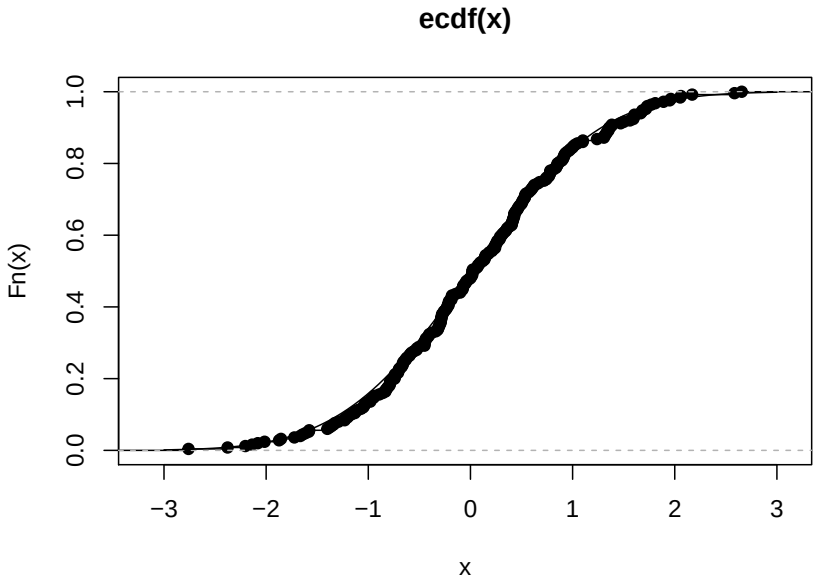
$$\hat{F}(x) = \frac{1}{n}\sum_{i=1}^{n} I_{(-\infty,x]}(X_i) = \frac{\#X_i \leq x}{n}$$

Here are two examples:

```
x <- rnorm(25)
plot(ecdf(x))
curve(pnorm(x), -3, 3, add=T)
```




```
x <- rnorm(250)
plot(ecdf(x))
curve(pnorm(x), -3, 3, add = TRUE)
```



There is a famous theorem in probability theory (Glivenko-Cantelli) that says that the empirical distribution function converges to the true distribution function uniformly.

How does one simulate from $\hat{F}$? It means to *resample* from the data, that is randomly select numbers from x *with replacement* such that each observation has an equal chance of getting picked:

```
x <- sort(round(rnorm(10, 10, 3), 1))
x
```

```
##  [1]  1.2  9.7 10.3 11.9 12.0 13.4 13.5 13.9 14.0 14.2
```

```
sort(sample(x, size=10, replace=TRUE))
```

```
##  [1]  1.2  1.2  9.7 10.3 11.9 12.0 12.0 13.5 14.0 14.2
```

```
sort(sample(x, size=10, replace=TRUE))
```

```
##  [1]  1.2  1.2  9.7  9.7 11.9 11.9 12.0 13.4 14.0 14.2
```

```
sort(sample(x, size=10, replace=TRUE))
```

```
##  [1] 10.3 10.3 13.4 13.4 13.4 13.9 14.0 14.2 14.2 14.2
```

Now the Bootstrap estimate of standard error is simply the sample standard deviation of the estimates of B such bootstrap samples:

```
x <- rnorm(250)
B <- 1000
z <- matrix(0, B, 2)
for(i in 1:B) {
  x.boot <- sample(x, size=length(x), replace=TRUE)
  z[i, 1]<- mean(x.boot)
```

```
  z[i, 2]<- median(x.boot)
}
round(c(1/sqrt(length(x)), apply(z, 2, sd)), 3)
```

```
## [1] 0.063 0.063 0.077
```

There is also a package that we can use:

```
library(bootstrap)
sd(bootstrap(x, 1000, mean)$thetastar)
```

```
## [1] 0.06633883
```

```
sd(bootstrap(x, 1000, median)$thetastar)
```

```
## [1] 0.08329715
```

Example: Skewness

the *skewness* of a distribution is a measure of it's lack of symmetry. It is defined by

$$\gamma_1 = E\left[\left(\frac{X-\mu}{\sigma}\right)^3\right]$$

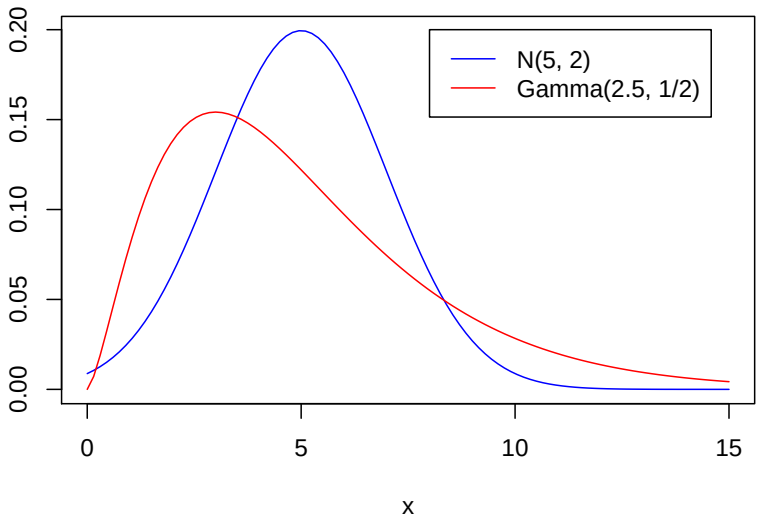
and for a symmetric distribution we should have $\gamma_1 = 0$.

a standard estimator of γ_1 is

$$\hat{\gamma}_1 = \frac{\frac{1}{n}\sum(x_i - \overline{x})^3}{[\frac{1}{n-1}\sum(x_i - \overline{x})^2]^{3/2}} = \frac{\frac{1}{n}\sum(x_i - \overline{x})^3}{[\text{sd(x)}]^3}$$

What is the standard error in this estimate? Doing this analytically would be quite an exercise, but:

```
curve(dnorm(x, 5, 2), 0, 15, col="blue", ylab="")
legend(8, 0.2, c("N(5, 2)",  "Gamma(2.5, 1/2)"),
    lty=c(1, 1), col=c("blue", "red"))
curve(dgamma(x, 2.5, 1/2), 0, 15, add=TRUE, col="red")
```




```
T.fun <- function(x) mean( (x-mean(x))^3 )/sd(x)^3
x <- rnorm(250, 5, 2)
x.boot <- bootstrap(x, 500, T.fun)$thetastar
round(c(mean(x.boot), sd(x.boot)), 3)
```

```
## [1] -0.042  0.134
```

```
x <- rgamma(250, 2.5, 1/2)
x.boot <- bootstrap(x, 500, T.fun)$thetastar
round(c(mean(x.boot), sd(x.boot)), 3)
```

```
## [1] 1.320 0.253
```

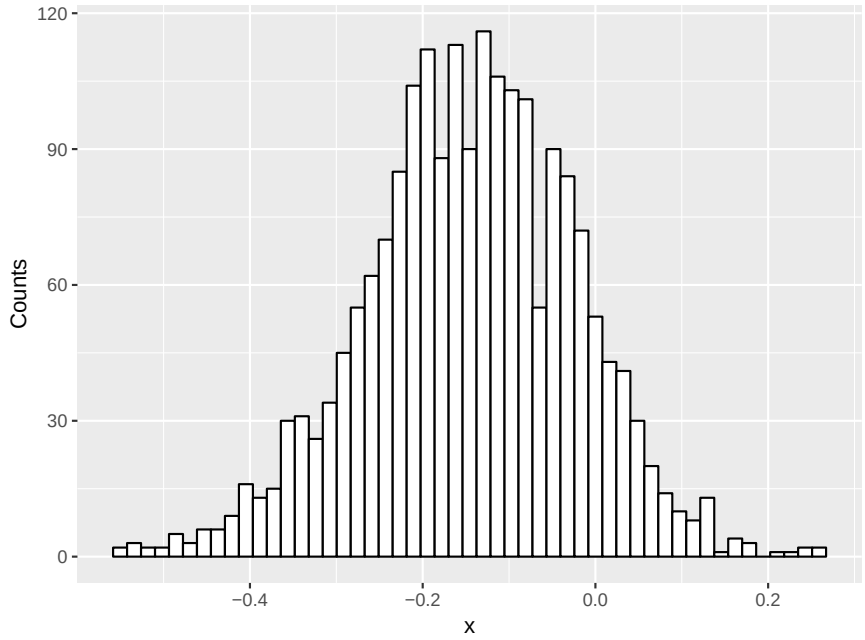
Bootstrap Confidence Intervals

There are two standard technics for using the Bootstrap to find confidence intervals:

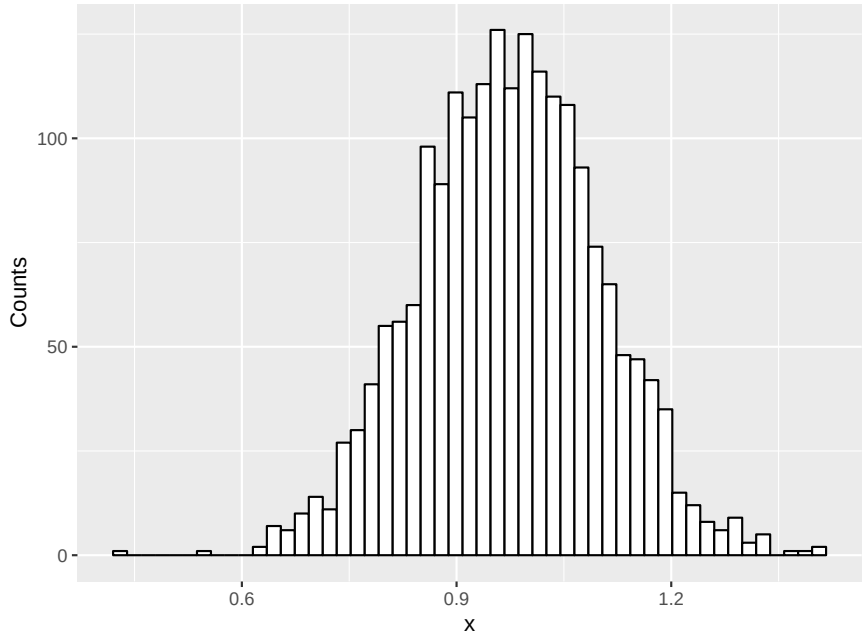
- **Normal Theory Intervals**

Let's continue the discussion of the skewness, and put a 95% confidence interval on the estimates:

```
x.normal <- rnorm(250, 5, 2)
T.fun <- function(x) mean( (x-mean(x))^3 )/sd(x)^3
thetastar.normal <- bootstrap(x.normal, 2000, T.fun)$thetastar
df <- data.frame(x = thetastar.normal)
bw <- diff(range(thetastar.normal))/50
ggplot(df, aes(x)) +
  geom_histogram(color = "black", fill = "white", binwidth = bw) +
  labs(x="x", y="Counts")
```




```
x.gamma <- rgamma(250, 2.5, 1/2)
thetastar.gamma <- bootstrap(x.gamma, 2000, T.fun)$thetastar
df <- data.frame(x = thetastar.gamma)
bw <- diff(range(thetastar.gamma))/50
ggplot(df, aes(x)) +
  geom_histogram(color = "black",
                 fill = "white",
                 binwidth = bw) +
  labs(x="x", y="Counts")
```



Note that I increased the number of Bootstrap samples to 2000, which is standard when calculating confidence intervals.

We can see that the bootstrap estimates are reasonably normally distributed, so we can find the confidence interval with

```
round(T.fun(x.normal) +
        c(-1, 1)*qnorm(0.975)*sd(thetastar.normal), 2)
```

```
## [1] -0.38  0.10
```

```
round(T.fun(x.gamma) +
        c(-1, 1)*qnorm(0.975)*sd(thetastar.gamma), 2)
```

```
## [1] 0.74 1.23
```

so in the normal case 0 is in the interval, indicating that this data set might well come from a symmetric distribution, whereas in the gamma case this is ruled out.

Notice that there is no $/\sqrt{n}$, because sd(thetastar) is already the standard deviation of the estimator, not of an individual observation.

- **Percentile Intervals**

An alternative way to find confidence intervals is by estimating the population quantiles of the bootstrap sample with the sample quantiles:

```
2000*c(0.025, 0.975)
```

```
## [1]   50 1950
```

```
round(sort(thetastar.normal)[2000*c(0.025, 0.975)], 2)
```

```
## [1] -0.40  0.08
```

```
round(sort(thetastar.gamma)[2000*c(0.025, 0.975)], 2)
```

```
## [1] 0.72 1.22
```

in our examples the two methods yield similar intervals.

- **More Advanced Intervals**

There are a number of ways to improve the performance of bootstrap based confidence intervals. One of the more popular ones is called *nonparametric bias-corrected and accelerated (BCa) intervals*. The package *bootstrap* has the routine *bcanon*. The intervals are the found via the percentile method but the percentiles are found with

$$\alpha_1 = \Phi\left(\widehat{z_0} + \frac{\widehat{z_0} + z_\alpha}{1 - \hat{a}(\widehat{z_0} + z_\alpha)}\right)$$
$$\alpha_2 = \Phi\left(\widehat{z_0} + \frac{\widehat{z_0} + z_{1-\alpha}}{1 - \hat{a}(\widehat{z_0} + z_{1-\alpha})}\right)$$

here

- Φ is the standard normal cdf
- α is the desired confidence level
- $\widehat{z_0}$ is a bias-correction factor
- $\hat{a}$ is called the acceleration

```
bcanon(x.normal, 2000, T.fun, alpha=c(0.025, 0.975))$conf
```

```
##      alpha   bca point
## [1,] 0.025 -0.39388296
## [2,] 0.975  0.07641379
```

```
bcanon(x.gamma, 2000, T.fun, alpha=c(0.025, 0.975))$conf
```

```
##      alpha bca point
## [1,] 0.025 0.7479726
## [2,] 0.975 1.2434498
```

Basic Inferences

In this section we will discuss some of the standard (frequentist) methods in Statistics.

Inference for a Population Mean

The basic R command for inference for a population mean is *t.test*.

- **Confidence Intervals**

Example: Mothers Cocain Use and Babies Health

Chasnoff and others obtained several measures and responses for newborn babies whose mothers were classified by degree of cocaine use.

The study was conducted in the Perinatal Center for Chemical Dependence at Northwestern University Medical School. The measurement given here is the length of the newborn.

Source: Cocaine abuse during pregnancy: correlation between prenatal care and perinatal outcome
Authors: SN MacGregor, LG Keith, JA Bachicha, and IJ Chasnoff
Obstetrics and Gynecology 1989;74:882-885

Let's ignore the drug status for the moment and find a 90% confidence interval for the length of a newborn baby

```
round(as.numeric(t.test(mothers$Length)$conf.int), 2)
```

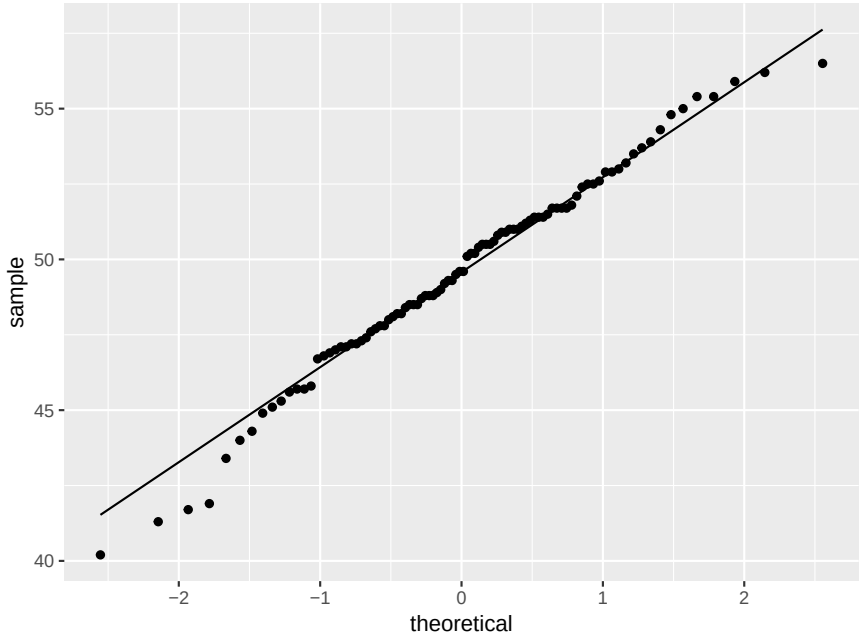
```
## [1] 48.86 50.24
```

The assumptions for this method are:

- data comes from a normal distribution
- or data set is large enough

Let's check:

```
df <- data.frame(x=mothers$Length)
ggplot(df, aes(sample=x)) +
  stat_qq() + stat_qq_line()
```



This is fine.

- **Hypothesis Testing**

Example: Resting Period of Monarch Butterflies

Some Monarch butterflies fly early in the day, others somewhat later. After the flight they have to rest for a short period. It has been theorized that the resting period (RIP) of butterflies flying early in the morning is shorter because this is a thermoregulatory mechanism, and it is cooler in the mornings. The mean RIP of all Monarch butterflies is 133 sec. Test the theory at the 10% level.

Research by Anson Lui, Resting period of early and late flying Monarch butterflies Danaeus plexippus, 1997

1. Parameter: mean μ
2. Method: 1-sample t

3. Assumptions: normal data or large sample

4. $\alpha = 0.1$

5. $H_0 : \mu = 133$ (RIP is the same for early morning flying butterflies as all others)

6. $H_0 : \mu < 133$ (RIP is the shorter for early morning flying butterflies)

7.

```
t.test(butterflies$RIP.sec.,
       mu=133,
       alternative = "less")$p.value
```

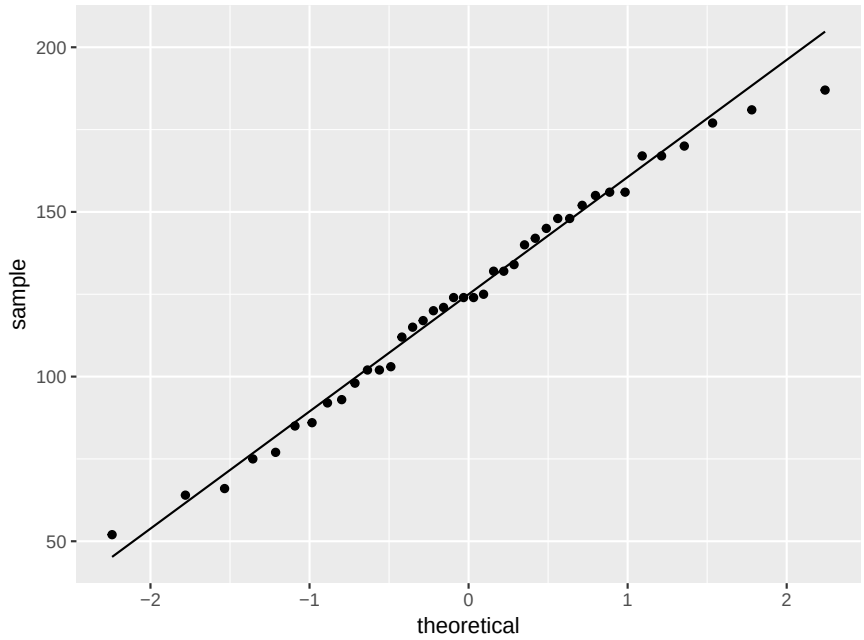
```
## [1] 0.05583963
```

8. $p = 0.0558 < \alpha = 0.1$, so we reject the null hypothesis

9. It appears the resting time is somewhat shorter, but the conclusion is not a strong one.

Checking the assumption:

```
df <- data.frame(x=butterflies$RIP.sec.)
ggplot(df, aes(sample=x)) +
  stat_qq() + stat_qq_line()
```



looks good.

- **Power Calculations**

The power of a test is the probability to correctly reject the null if the null is false. The power will always depend on an assumed value for the mean.

Let's say the true mean resting period is 120.7 seconds. What was the probability that Anson's experiment would have detected this? That is, what is

$$P_{\mu=120.7}(\text{reject null})$$

First we need to find the critical region of this test, so we know what *reject* H_0 actually means. Now if the test is done at the 5% level we reject the null if the p value is less than 0.05. How can we find out for what value of the sample mean this will happen? Let's do the following:

- generate data from a normal distribution with mean μ, standard deviation as in our data and the same number of observations
- find the p value and check whether it is < 0.05
- repeat many times and find average.
- use trial and error on μ until the probability is (about) 0.05:

```
mean.M <- function(M, true.mu=133) {
  B <- 10000
  pvals <- rep(0, B)
  for(i in 1:B) {
    x <- rnorm(length(butterflies$RIP.sec.), M,
               sd(butterflies$RIP.sec.))
    pvals[i] <- t.test(x, mu=true.mu,
       alternative = "less")$p.value
  }
  1-sum(pvals<0.05)/B
}
mean.M(120)
```

```
## [1] 0.2399
```

```
mean.M(110)
```

```
## [1] 0.0066
```

```
mean.M(115)
```

```
## [1] 0.0532
```

just about right! We now know that "reject H_0" means $\bar{X} < 115$.

Now we turn this around: if the true mean where 120.5, what is the probability that we would reject the null, that is get a sample mean of 115 or less? Actually, we can again use the same routine:

```
mean.M(115, true.mu = 120.7)
```

```
## [1] 0.724
```

Of course here this can also be done analytically:

$$P_{\mu=133}(\text{ reject null }) =$$
$$P_{\mu=133}(\bar{X} < \text{crit}) = 0.05$$

Now $\bar{X} \sim N(133, s/\sqrt{40})$, so

```
crit <- qnorm(0.05, 133, sd(butterflies$RIP.sec.)/sqrt(40))
crit
```

```
## [1] 124.0558
```

(Actually the distribution is t, not a normal, but we will ignore this here)

and now

$$P_{\mu=120.7}(\text{ reject null }) =$$
$$P_{\mu=120.7}P(\bar{X} < 124.06) =$$

```
pnorm(124.06, 120.7, sd(butterflies$RIP.sec.)/sqrt(40))
```

```
## [1] 0.7316819
```

and of course we get the same answer.

There are also a number of packages that can be used to find the power of a test:

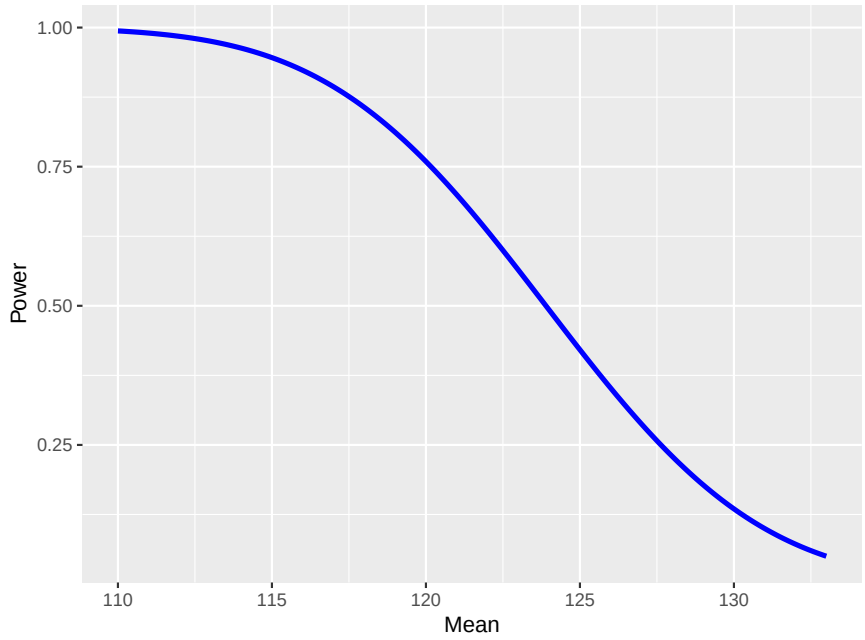
```
library(pwr)
pwr.t.test(40, d=(120.7-133)/sd(butterflies$RIP.sec.),
           alternative = "less",
           type = "one.sample")
```

```
##
##      One-sample t test power calculation
##
##               n = 40
##               d = -0.3576509
##       sig.level = 0.05
##           power = 0.7182354
##     alternative = less
```

Usually one wants to study the power for a whole range of values. This is done by drawing a *power curve*:

```
x <- seq(110, 133, 0.1)
y <- x
for(i in seq_along(x))
  y[i] <- pwr.t.test(40,
          d=(x[i]-133)/sd(butterflies$RIP.sec.),
           alternative = "less",
```

```
            type = "one.sample")$power
df <- data.frame(Mean=x, Power=y)
ggplot(df, aes(Mean, Power)) +
  geom_line(col="blue", size=1.2)
```



- **Sample Size**

so, if the true mean resting period is 120.7 seconds the power is 72%. What sample size would we need to have a power of 95%

```
round(pwr.t.test(power=0.95,
      d=(120.7-133)/sd(butterflies$RIP.sec.),
      alternative = "less",
      type = "one.sample")$n)
```

```
## [1] 86
```

The sample size issue also arises when we want to find a confidence interval. Here the number that corresponds to the power is the *error* E, that is half the length of the interval.

Analytically we find

$$\bar{X} \pm z_{\alpha/2} s/\sqrt{n}$$
$$E = z_{\alpha/2} s/\sqrt{n}$$
$$n = (\frac{z_{\alpha/2} s}{E})^2$$

let's see

```
I <- t.test(butterflies$RIP.sec.)$conf.int
diff(I)/2
```

```
## [1] 10.9988
```

so a 95% confidence interval has an error of 11. If we wanted an error of 5 we would need a sample size of

```
round((qnorm(0.975)*sd(butterflies$RIP.sec.)/5)^2)
```

```
## [1] 182
```

Inference for a Population Proportion

The R routine for inference for a proportion (or a probability or a percentage) is *binom.test*. This implements a method by Clopper and Pearson (1934). This method is exact and has no assumptions.

Note The formula discussed in many introductory statistic courses for the confidence interval is

$$\hat{p} \pm \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

where $\hat{p}$ is the proportion of success. This leads to confidence intervals that are now known to be quite wrong, and so this method should not be used anymore. The same is true for the corresponding hypothesis test. This method (actually a slight improvement due to Wilson (1927)) is implemented in R by *prop.test*.

Example: Jon Kerrichs Coin

The South African Jon Kerrich spent some time in a German prisoner of war camp during world war I. He used his time to flip a coin 10000 times, resulting in 5067 heads.

Test at the 5% level of significance whether 5067 heads in 10000 flips are compatible with a fair coin.

1. Parameter: proportion π
2. Method: exact binomial
3. Assumptions: None
4. $\alpha = 0.05$
5. $H_0 : \pi = 0.5$ (50% of flips result in "Heads", coin is fair)
6. $H_a : \pi \neq 0.5$ (coin is not fair)

7.

```
binom.test(x = 5067, n = 10000)$p.value
```

```
## [1] 0.1835155
```

8. $p = 0.1835 > \alpha = 0.05$, so we fail to reject the null hypothesis.

9. it appears Jon Kerrich's coin was indeed fair.

Example: Sample Size for Polling

Say some polling institute wants to conduct a poll for the next election for president. They will then find a 95% confidence interval and they want this interval to have an error of 3 percentage points (aka ± 0.03). What sample size do they need?

In American politics the two parties are always very close, so in a poll with n people about n/2 will vote for one or the other party. Let's do a little trial and error:

```
n <- 100
diff(as.numeric(binom.test(n/2, n)$conf.int)/2)
```

```
## [1] 0.1016789
```

Now that is to large, so

```
n <- 200
diff(as.numeric(binom.test(n/2, n)$conf.int)/2)
```

```
## [1] 0.07134157
```

```
n <- 400
diff(as.numeric(binom.test(n/2, n)$conf.int)/2)
```

```
## [1] 0.05009211
```

```
n <- 800
diff(as.numeric(binom.test(n/2, n)$conf.int)/2)
```

```
## [1] 0.03521797
```

```
n <- 1200
diff(as.numeric(binom.test(n/2, n)$conf.int)/2)
```

```
## [1] 0.02867679
```

```
n <- 1100
diff(as.numeric(binom.test(n/2, n)$conf.int)/2)
```

```
## [1] 0.02996843
```

```
n <- 1050
diff(as.numeric(binom.test(n/2, n)$conf.int)/2)
```

```
## [1] 0.03068294
```

There is something quite remarkable about this result!

Correlation

Example UPR Admissions data

What are the correlations between the various variables?

```
head(upr, 2)
```

```
##      ID.Code Year Gender Program.Code Highschool.GPA Aptitud.Verbal
## 1 00C2B4EF77 2005      M          502           3.97            647
## 2 00D66CF1BF 2003      M          502           3.80            597
##   Aptitud.Matem Aprov.Ingles Aprov.Matem Aprov.Espanol IGS Freshmen.GPA
## 1           621          626         672           551 342         3.67
## 2           726          618         718           575 343         2.75
##   Graduated Year.Grad. Grad..GPA Class.Facultad
## 1        Si       2012      3.33           INGE
## 2        No         NA        NA           INGE
```

Let's take out the those variables that are either not numerical or not useful for prediction:

```
x <- upr[, -c(1, 2, 3, 4, 13, 14, 16)]
head(x, 2)
```

```
##   Highschool.GPA Aptitud.Verbal Aptitud.Matem Aprov.Ingles Aprov.Matem
## 1           3.97            647           621          626         672
## 2           3.80            597           726          618         718
##   Aprov.Espanol IGS Freshmen.GPA Grad..GPA
## 1           551 342         3.67      3.33
## 2           575 343         2.75        NA
```

```
round(cor(x, use = "complete.obs") ,3)
```

```
##                Highschool.GPA Aptitud.Verbal Aptitud.Matem Aprov.Ingles
## Highschool.GPA          1.000          0.205         0.163        0.056
## Aptitud.Verbal          0.205          1.000         0.497        0.519
## Aptitud.Matem           0.163          0.497         1.000        0.474
## Aprov.Ingles            0.056          0.519         0.474        1.000
## Aprov.Matem             0.213          0.515         0.821        0.510
## Aprov.Espanol           0.268          0.604         0.408        0.439
## IGS                     0.666          0.734         0.769        0.462
## Freshmen.GPA            0.395          0.339         0.290        0.270
## Grad..GPA               0.385          0.349         0.316        0.280
##                Aprov.Matem Aprov.Espanol   IGS Freshmen.GPA Grad..GPA
## Highschool.GPA       0.213         0.268 0.666        0.395     0.385
## Aptitud.Verbal       0.515         0.604 0.734        0.339     0.349
## Aptitud.Matem        0.821         0.408 0.769        0.290     0.316
## Aprov.Ingles         0.510         0.439 0.462        0.270     0.280
```

```
## Aprov.Matem      1.000         0.421 0.712       0.326     0.351
## Aprov.Espanol    0.421         1.000 0.569       0.350     0.372
## IGS              0.712         0.569 1.000       0.474     0.485
## Freshmen.GPA     0.326         0.350 0.474       1.000     0.750
## Grad..GPA        0.351         0.372 0.485       0.750     1.000
```

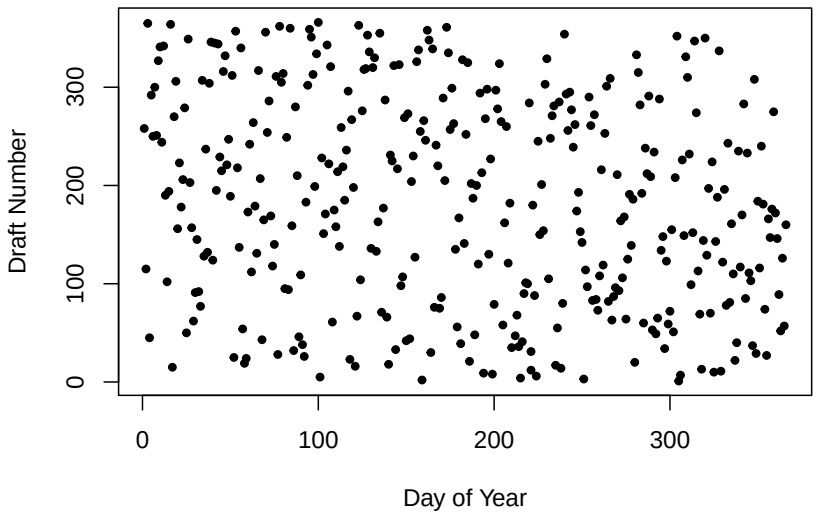
Example: The 1970's Military Draft

In 1970, Congress instituted a random selection process for the military draft. All 366 possible birth dates were placed in plastic capsules in a rotating drum and were selected one by one. The first date drawn from the drum received draft number one and eligible men born on that date were drafted first. In a truly random lottery there should be no relationship between the date and the draft number.

Question: was the draft was really random?

Here we have two quantitative variables, so we start with the scatterplot:

```
plot(draft$Draft.Number, draft$Day.of.Year,
     pch=20,
     xlab="Day of Year",
     ylab="Draft Number")
```



and this does not look like there is a problem with independence.

However:

1) Parameter: Pearson's correlation coefficient ρ
2) Method: Test for Pearson's correlation coefficient ρ
3) Assumptions: relationship is linear and that there are no outliers.

4) $\alpha = 0.05$

5) $H_0 : \rho = 0$ (no relationship between Day of Year and Draft Number)

6) $H_a : \rho \neq 0$ (some relationship between Day of Year and Draft Number)

7)

```
cor.test(draft$Draft.Number, draft$Day.of.Year)$p.value
```

```
## [1] 1.263829e-05
```

8) $p = 0.0000 < \alpha = 0.05$, so we reject the null hypothesis,
9) There is a statistically significant relationship between Day of Year and Draft Number.

Categorical Data Analysis - Tests for Independence

Example: Drownings in Los Angeles

Data is from O'Carroll PW, Alkon E, Weiss B. Drowning mortality in Los Angeles County, 1976 to 1984, JAMA, 1988 Jul 15;260(3):380-3.

Drowning is the fourth leading cause of unintentional injury death in Los Angeles County. They examined data collected by the Los Angeles County Coroner's Office on drownings that occurred in the county from 1976 through 1984. There were 1587 drownings (1130 males and 457 females) during this nine-year period

```
kable.nice(drownings)
```

	Male	Female
Private Swimming Pool	488	219
Bathtub	115	132
Ocean	231	40
Freshwater bodies	155	19
Hottubs	16	15
Reservoirs	32	2
Other Pools	46	14
Pails, basins, toilets	7	4
Other	40	12

Here we have two categorical variables (Method of Drowning and Gender), both categorical. We want to know whether the variables are independent. The most popular method of analysis for this type of problem is **Pearson's chi square test of independence**. It is done with the command *chisq.test* and it has the assumption of no expected counts less than 5.

1. Parameters of interest: measure of association

2. Method of analysis: chi-square test of independence

3. Assumptions of Method: all expected counts greater than 5

4. Type I error probability α=0.05

5. H_0: Classifications are independent = there is no difference in the method of drowning between men and women.

6. H_a: Classifications are dependent = there is some difference in the method of drowning between men and women.

7.

```
chisq.test(drownings)
```

```
##
##  Pearson's Chi-squared test
##
## data:  drownings
## X-squared = 144.48, df = 8, p-value < 2.2e-16
```

8. p = 0.000 < α=0.05, we reject the null hypothesis, there is a statistically significant difference between men and women and where they drown.

Let's see whether there is a problem with the assumptions:

```
round(chisq.test(drownings)$expected, 1)
```

```
##                          Male Female
## Private Swimming Pool  503.4  203.6
## Bathtub                175.9   71.1
## Ocean                  193.0   78.0
## Freshwater bodies      123.9   50.1
## Hottubs                 22.1    8.9
## Reservoirs              24.2    9.8
## Other Pools             42.7   17.3
## Pails, basins, toilets   7.8    3.2
## Other                   37.0   15.0
```

and we see that the expected counts of Pails, basins, toilets and Female is 3.2. In real life this would be considered ok, but it would also be easy to fix:

```
newmale <- c(drownings[1:7, 1], 7+40)
newfemale <- c(drownings[1:7, 2], 4+12)
```

```
newdrown <- cbind(newmale, newfemale)
newdrown
```

```
##                        newmale newfemale
## Private Swimming Pool      488       219
## Bathtub                    115       132
## Ocean                      231        40
## Freshwater bodies          155        19
## Hottubs                     16        15
## Reservoirs                  32         2
## Other Pools                 46        14
##                             47        16
```

```
out <- chisq.test(newdrown)
round(out$expected, 1)
```

```
##                        newmale newfemale
## Private Swimming Pool    503.4     203.6
## Bathtub                  175.9      71.1
## Ocean                    193.0      78.0
## Freshwater bodies        123.9      50.1
## Hottubs                   22.1       8.9
## Reservoirs                24.2       9.8
## Other Pools               42.7      17.3
##                           44.9      18.1
```

```
round(out$p.value, 4)
```

```
## [1] 0
```

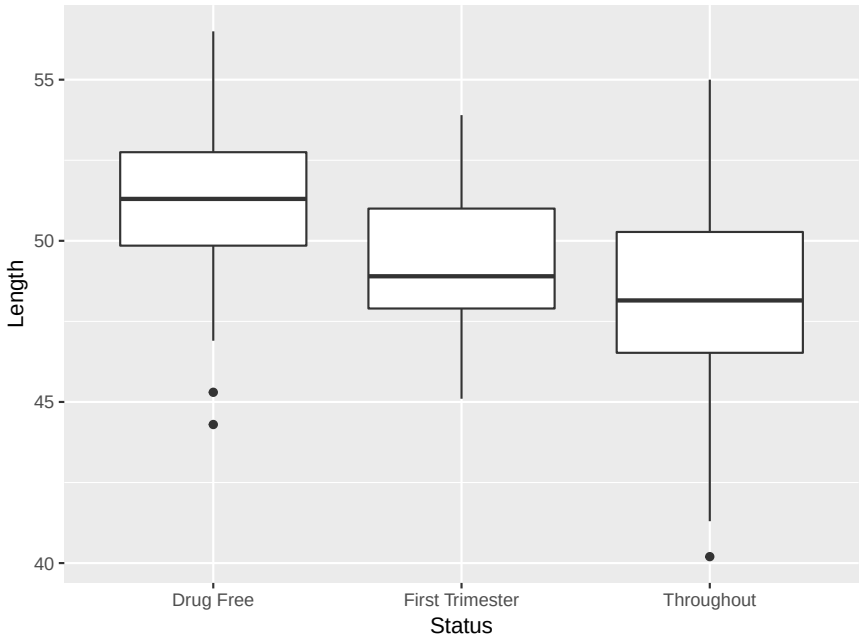
Comparing the Means of Several Populations - ANOVA

Basic Test

Example: Mothers Cocain Use and Babies Health

Are the mean lengths of the babies different depending on the drug use of the mother?

```
ggplot(mothers, aes(Status, Length)) +
        geom_boxplot()
```




```
out <- matrix(0, 3, 3)
colnames(out) <- c("Size", "Mean", "SD")
rownames(out) <- unique(mothers$Status)
out[, 1] <- tapply(mothers$Length,
                   mothers$Status, length)
out[, 2] <- round(tapply(mothers$Length,
                         mothers$Status, mean), 2)
out[, 3] <- round(tapply(mothers$Length,
                         mothers$Status, sd), 2)
```

	Size	Mean	SD
Drug Free	39	51.1	2.9
First Trimester	19	49.3	2.5
Throughout	36	48.0	3.6

```
kable.nice(out)
```

The standard method for this problem is called **ANOVA** (Analysis of Variance) and is run with the *aov* command.

```
fit <- aov(Length~Status, data=mothers)
summary(fit)
```

```
##             Df Sum Sq Mean Sq F value   Pr(>F)
## Status       2  181.4   90.69   9.319 0.000208
## Residuals   91  885.6    9.73
```

1. Parameters of interest: group means
2. Method of analysis: ANOVA
3. Assumptions of Method: residuals have a normal distribution, groups have equal variance

4. Type I error probability α=0.05

5. Null hypothesis H_0: $\mu_1 = \mu_2 = \mu_3$ (groups have the same means)

6. Alternative hypothesis H_a: $\mu_i \neq \mu_j$ (at least two groups have different means)

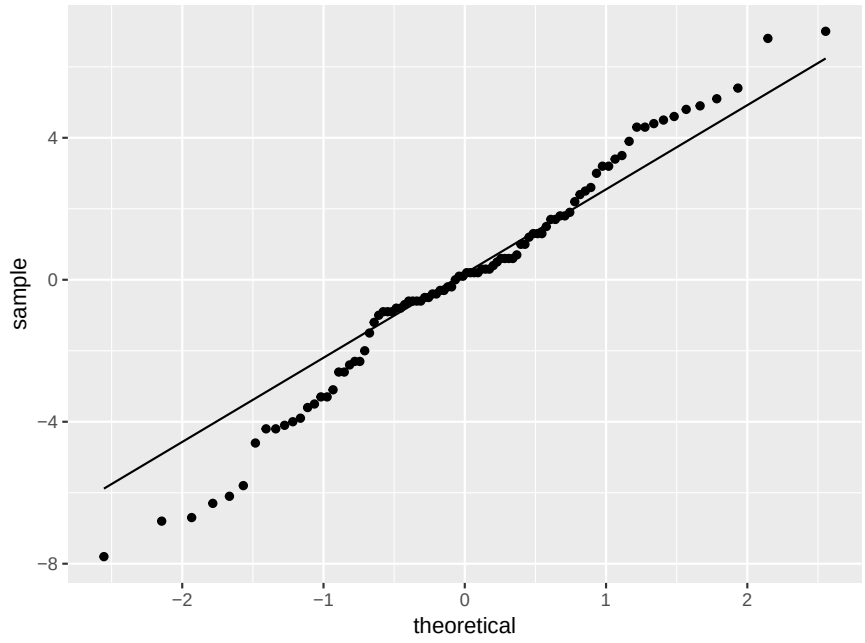
7. p=0.0002

8. $0.0002 < 0.05$, there is some evidence that the group means are not the same, the babies whose mothers used cocaine tend to be a little shorter (less healthy?)

In step 3 we have the assumptions

a. residuals have a normal distribution. W can check that with the normal plot. The residuals are the simply the observations minus their group means and are part of the aov object.

```
df <- data.frame(x=resid(fit))
ggplot(df, aes(sample=x)) +
  stat_qq() + stat_qq_line()
```



looks fine

b. groups have equal variance

Here one uses the rule of thumb: if the largest sample standard deviation is not more than three times the smallest, it is ok.

Here: 3*2.5 = 7.5 > 3.6, ok

Multiple Comparison

Often if the null of no difference is rejected, one wants to go a step further and do a pairwise comparison:

- is Drug Free different from First Trimester?
- is First Trimester different from Throughout?

There are a number of methods known for this problem, a popular one is by **Tukey**:

```
tuk <- TukeyHSD(fit)
plot(tuk)
```

this draws confidence intervals for the difference in means of all pairs. If an interval does not contain 0, the corresponding pair is statistically significantly different.

Here that is the case only for Drug Free - Throughout, so the other two pairs are not statistically significantly different. Remember, however that *failing to reject* H_0 is NOT the same as *accepting* H_0. The fact that those pairs are not statistically significantly different is almost certainly due to a lack of sample size.

Example: Cuckoo Eggs

That cuckoo eggs were peculiar to the locality where found was already known in 1892. A study by E.B. Chance in 1940 called The Truth About the Cuckoo demonstrated that cuckoos return year after year to the same territory and lay their eggs in the nests of a particular host species. Further, cuckoos appear to mate only within their territory. Therefore, geographical sub-species are developed, each with a dominant foster-parent species, and natural selection has ensured the survival of cuckoos most fitted to lay eggs that would be adopted by a

particular foster-parent species. The data has the lengths of cuckoo eggs found in the nests of six other bird species (drawn from the work of O.M. Latter in 1902).

Cuckoo Birds

Basic question: is there a difference between the lengths of the cuckoo eggs of different Foster species?

```
head(cuckoo)
```

```
##           Bird Length
## 1 Meadow Pipit  19.65
## 2 Meadow Pipit  20.05
## 3 Meadow Pipit  20.65
## 4 Meadow Pipit  20.85
## 5 Meadow Pipit  21.65
## 6 Meadow Pipit  21.65
```

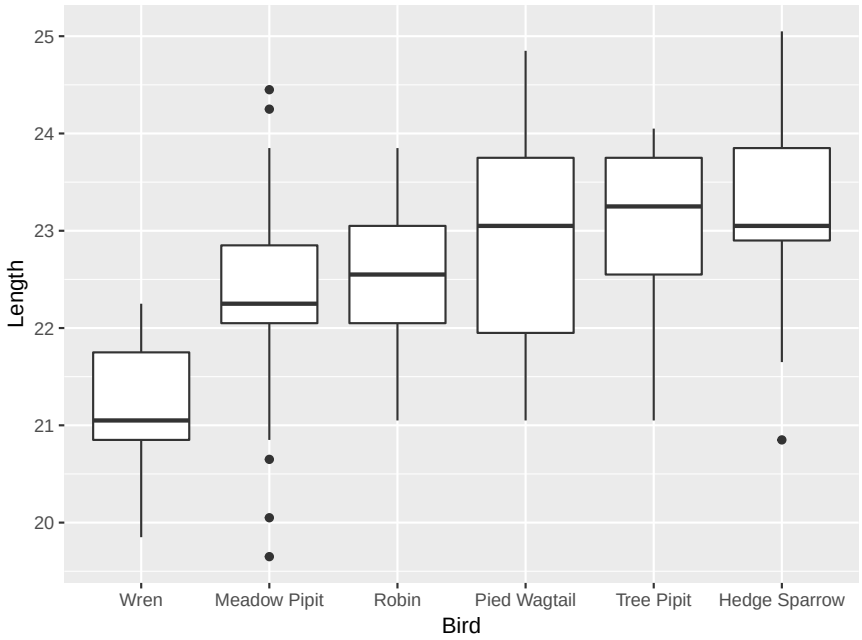
```
table(cuckoo$Bird)
```

```
##
## Hedge Sparrow  Meadow Pipit  Pied Wagtail         Robin    Tree Pipit
##            14            45            15            16            15
##          Wren
##            15
```

Here we have no obvious ordering of the groups. In this case the usual thing to do is to sort by the group means:

```
mn <- sort(tapply(cuckoo$Length, cuckoo$Bird, mean))
cuckoo$Bird <- factor(cuckoo$Bird,
                      levels = unique(names(mn)),
                      ordered = TRUE)
```

```
ggplot(data=cuckoo, aes(Bird, Length)) +
  geom_boxplot()
```



we have some outliers in the Meadow Pipit species, but not to bad and we will ignore that. Let's look at the table of summary statistics.

```
out <- matrix(0, 6, 3)
colnames(out) <- c("n", "Mean", "Sd")
rownames(out) <- as.character(levels(cuckoo$Bird))
out[, 1] <- tapply(cuckoo$Length,
                   cuckoo$Bird, length)
out[, 2] <- round(tapply(cuckoo$Length,
                   cuckoo$Bird, mean), 2)
out[, 3] <- round(tapply(cuckoo$Length,
                   cuckoo$Bird, sd), 2)
```

	n	Mean	Sd
Wren	15	21.13	0.74
Meadow Pipit	45	22.30	0.92
Robin	16	22.57	0.68
Pied Wagtail	15	22.90	1.07
Tree Pipit	15	23.09	0.90
Hedge Sparrow	14	23.12	1.07

```
kable.nice(out)
```

Both the graph and the table make it clear that there are some differences in the length, so the following is not really necessary:

```
fit <- aov(Length~Bird, data=cuckoo)
summary(fit)
```

```
##              Df Sum Sq Mean Sq F value   Pr(>F)
```

```
## Bird          5  42.94   8.588   10.39 3.15e-08
## Residuals   114  94.25   0.827
```

1) Parameters of interest: group means

2) Method of analysis: ANOVA

3) Assumptions of Method: residuals have a normal distribution, groups have equal variance

4) $\alpha = 0.05$

5) Null hypothesis H_0: $\mu_1 = ... = \mu_6$ (groups have the same means)

6) Alternative hypothesis H_a: $\mu_i \neq \mu_j$ (at least two groups have different means)

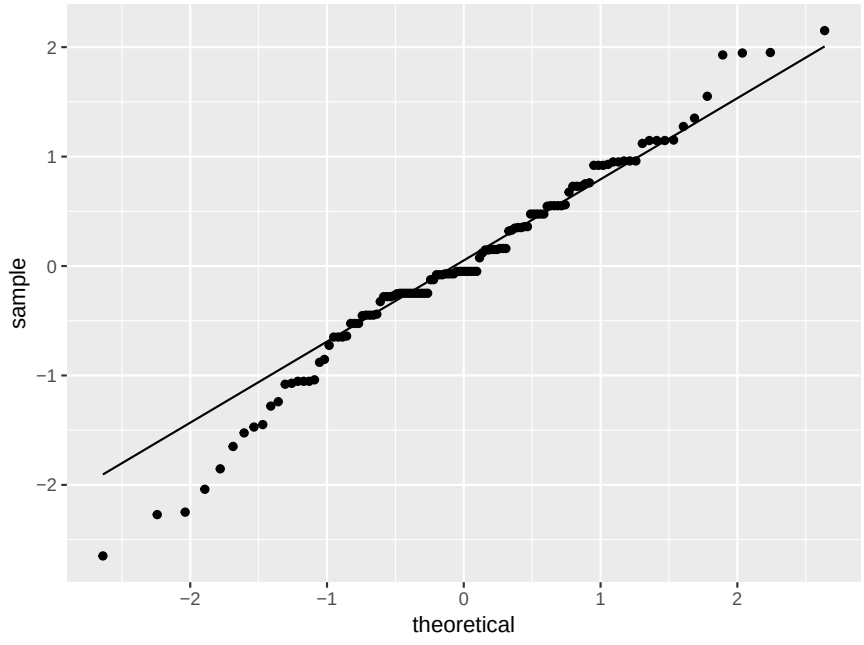
7) p value = 0.000

8) $0.000 < 0.05$, there is some evidence that the group means are not the same, the length are different for different foster species.

Assumptions of the method:

a) residuals have a normal distribution, plot looks (mostly) ok

```
df <- data.frame(Residuals=resid(fit),
            Fits = fitted(fit))
ggplot(data=df, aes(sample=Residuals)) +
           geom_qq() + geom_qq_line()
```



b) groups have equal variance

smallest stdev=0.7, largest stdev=1.1, 3*0.7=2.1>1.1, ok

So, how exactly do they differ?

```
tuk <- TukeyHSD(fit)
print(tuk)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = Length ~ Bird, data = cuckoo)
##
## $Bird
##                                  diff          lwr      upr     p adj
## Meadow Pipit-Wren          1.16888889  0.383069115 1.954709 0.0004861
## Robin-Wren                 1.44500000  0.497728567 2.392271 0.0003183
## Pied Wagtail-Wren          1.77333333  0.810904595 2.735762 0.0000070
## Tree Pipit-Wren            1.96000000  0.997571262 2.922429 0.0000006
## Hedge Sparrow-Wren         1.99142857  1.011964373 2.970893 0.0000006
## Robin-Meadow Pipit         0.27611111 -0.491069969 1.043292 0.9021876
## Pied Wagtail-Meadow Pipit  0.60444444 -0.181375330 1.390264 0.2324603
## Tree Pipit-Meadow Pipit    0.79111111  0.005291337 1.576931 0.0474619
## Hedge Sparrow-Meadow Pipit 0.82253968  0.015945760 1.629134 0.0428621
## Pied Wagtail-Robin         0.32833333 -0.618938100 1.275605 0.9155004
## Tree Pipit-Robin           0.51500000 -0.432271433 1.462271 0.6159630
## Hedge Sparrow-Robin        0.54642857 -0.418146053 1.511003 0.5726153
## Tree Pipit-Pied Wagtail    0.18666667 -0.775762072 1.149095 0.9932186
## Hedge Sparrow-Pied Wagtail 0.21809524 -0.761368960 1.197559 0.9872190
## Hedge Sparrow-Tree Pipit   0.03142857 -0.948035627 1.010893 0.9999990
```

so the eggs of Wrens are the smallest, and they are stat. significantly smaller than the eggs of all other birds.

Meadow Pipits are next, and they are stat. significantly smaller than the eggs of Tree Pipits and Hedge Sparrows.

no other differences are stat. significant!

This can get a bit hard to read, and it might be better to concentrate on those pairs that are stat. signif,. different at (say) the 5% level:

```
names(tuk[[1]][tuk[[1]][,4]<0.05, 4])
```

```
## [1] "Meadow Pipit-Wren"          "Robin-Wren"
## [3] "Pied Wagtail-Wren"          "Tree Pipit-Wren"
## [5] "Hedge Sparrow-Wren"         "Tree Pipit-Meadow Pipit"
## [7] "Hedge Sparrow-Meadow Pipit"
```

Transformations and Nonparametric Methods

Many classic methods of analysis (t tests, F tests etc) have an assumption of data that comes from a normal distribution. What can we do if we don't have that? There are two ways to proceed:

Transformations

A *data transformation* is any mathematical function applied to the data. Sometimes such a transformation can be used to make distribution more normal:

Example: Body and Brain Weight

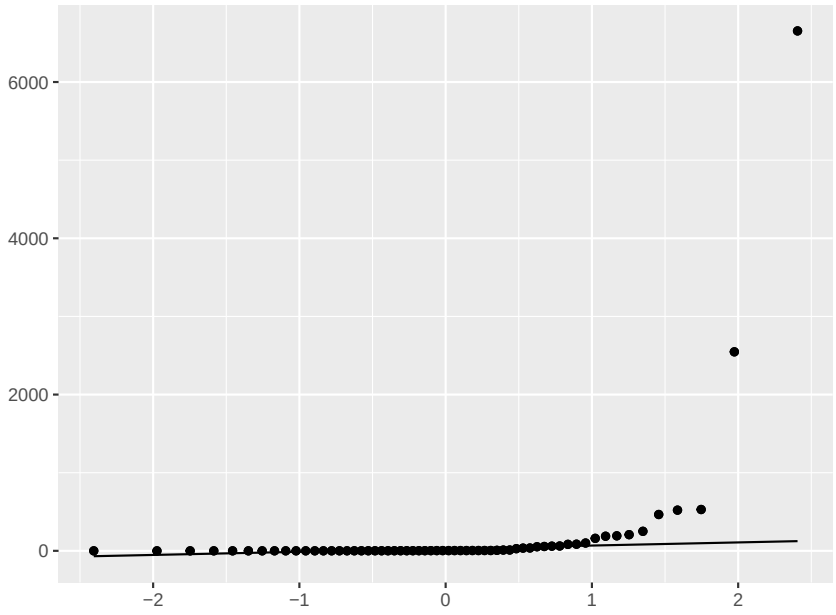
Consider the data set *brainsize*, which has the weights of the body (in kg) and of the brain (in gram) of 62 mammals:

```
kable.nice(brainsize)
```

Animal	body.wt.kg	brain.wt.g
African elephant	6654.000	5712.00
African giant pouched rat	1.000	6.60
Arctic Fox	3.385	44.50
Arctic ground squirrel	0.920	5.70
Asian elephant	2547.000	4603.00
Baboon	10.550	179.50
Big brown bat	0.023	0.30
Brazilian tapir	160.000	169.00
Cat	3.300	25.60
Chimpanzee	52.160	440.00
Chinchilla	0.425	6.40
Cow	465.000	423.00
Desert hedgehog	0.550	2.40
Donkey	187.100	419.00
Eastern American mole	0.075	1.20
Echidna	3.000	25.00
European hedgehog	0.785	3.50
Galago	0.200	5.00
Genet	1.410	17.50
Giant armadillo	60.000	81.00
Giraffe	529.000	680.00
Goat	27.660	115.00
Golden hamster	0.120	1.00
Gorilla	207.000	406.00
Gray seal	85.000	325.00
Gray wolf	36.330	119.50
Ground squirrel	0.101	4.00
Guinea pig	1.040	5.50
Horse	521.000	655.00
Jaguar	100.000	157.00
Kangaroo	35.000	56.00
Lesser short-tailed shrew	0.005	0.14
Little brown bat	0.010	0.25
Man	62.000	1320.00
Mole rat	0.122	3.00
Mountain beaver	1.350	8.10
Mouse	0.023	0.40
Musk shrew	0.048	0.33
N. American opossum	1.700	6.30
Nine-banded armadillo	3.500	10.80
Okapi	250.000	490.00

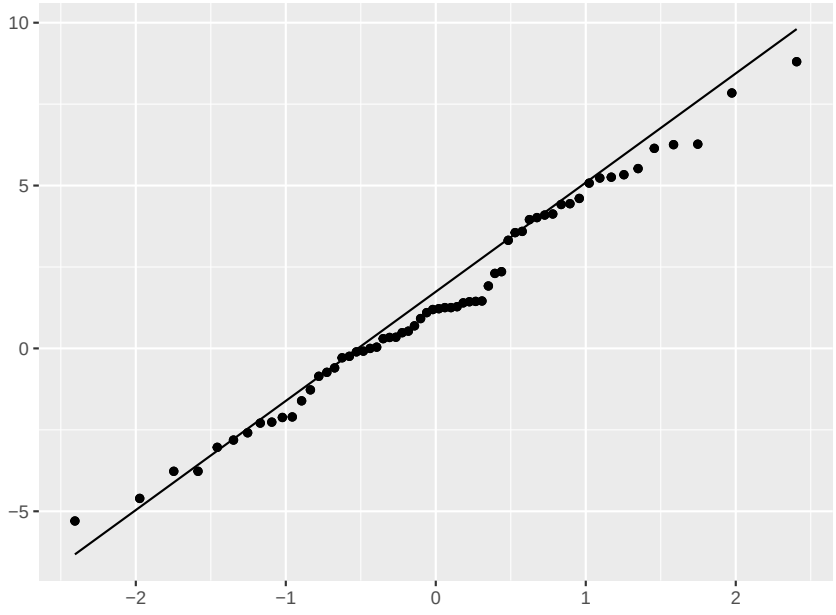
Let's say we want to find a 95% confidence interval for the mean body weight. If we want to use the *t.test* method we need to check normality:

```
qplot(data=brainsize, sample=body.wt.kg) +
  stat_qq() + stat_qq_line()
```



and this is clearly non-normal. However

```
qplot(data=brainsize, sample=log(body.wt.kg)) +
  stat_qq() + stat_qq_line()
```



shows that log(body.wt.kg) is indeed normally distributed. So now

```
t.test(log(brainsize$body.wt.kg))$conf.int
```

```
## [1] 0.5671413 2.1634911
## attr(,"conf.level")
## [1] 0.95
```

but this is confidence interval for log(body.wt.kg), we want one for body.wt.kg.

$$\begin{aligned} 0.567 \leq \log \mu \leq 2.163 \quad &\Leftrightarrow \\ \exp(0.567) \leq \mu \leq \exp(2.163) \quad &\Leftrightarrow \\ 1.763 \leq \mu \leq 8.697 & \end{aligned}$$

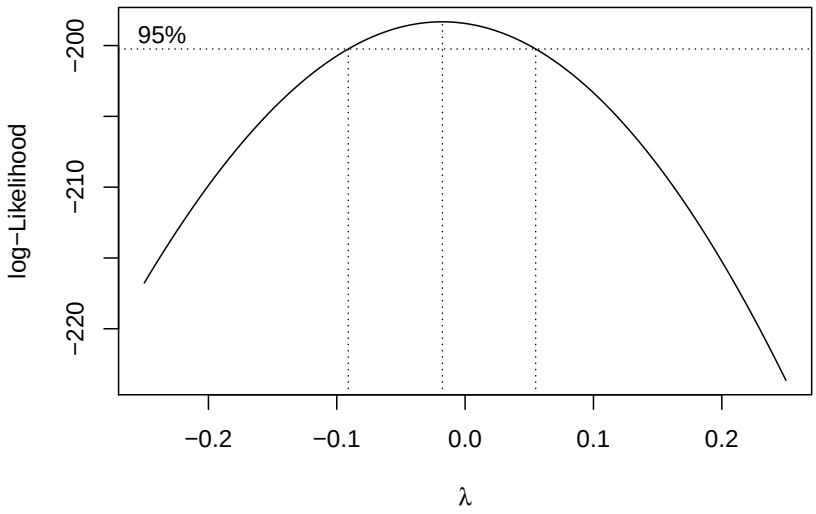
Box-Cox Transforms

Above we used a log transform. In principle any function might work, and there is even a way to pick the best from a list: In 1964 Box and Cox suggested a family of transformations of the form

$$T_\lambda(x) = \begin{cases} \frac{x^\lambda - 1}{\lambda} & \lambda \neq 0 \\ \log x & \lambda = 0 \end{cases}$$

Notice that this is continuous in λ: $\lim_{\lambda \to 0} T_\lambda(x) = T_0(x)$ and it includes $1/x, \sqrt{x}, x^k$ etc. To pick the best use

```
library(MASS)
fit <- lm(brainsize$body.wt.kg~rep(1, 62))
boxcox(fit, lambda=seq(-0.25, 0.25, 0.01))
```



the vertical lines give a confidence interval for λ, and any value inside the interval is acceptable. It seems that for our data $\lambda = 0$ or the log transform is indeed appropriate.

Note that the *boxcox* command requires a fit object generated by *lm* or *aov* but can also be used for single vectors as above.

A different way to proceed is to use a method that does not require a normal distribution. These are so called *nonparametric methods*. A number of them have been developed over the years as alternatives to the standard normal based methods.

In general nonparametric methods are based on the *ranks* of the observations and focus on the median instead of the mean.

Alternative to 1 Sample t

Example: Euro Coins

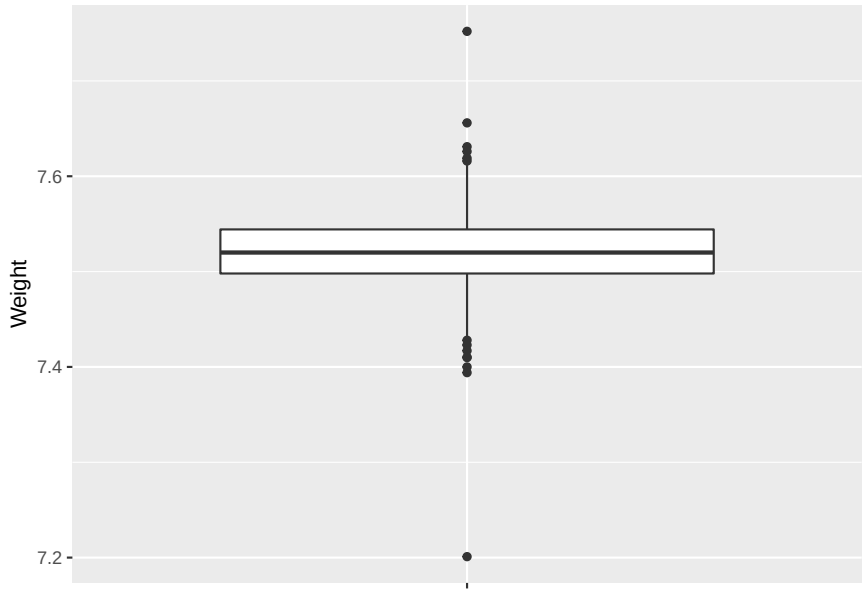
The data were collected by Herman Callaert at Hasselt University in Belgium. The euro coins were borrowed at a local bank. Two assistants, Sofie Bogaerts and Saskia Litiere weighted the coins one by one, in laboratory conditions on a weighing scale of the type Sartorius BP 310.

Say we are told that a one euro coin is supposed to weigh 7.5 grams. Does the data in support that claim?

```
head(euros)
```

```
##   Weight Roll
## 1  7.512    1
## 2  7.502    1
## 3  7.461    1
## 4  7.562    1
## 5  7.528    1
## 6  7.459    1
```

```
ggplot(euros, aes(y=Weight, x=rep("", 2000))) +
  geom_boxplot() +
  labs(x="")
```



The boxplot of Weight shows severe outliers, so the usual 1 sample t test won't work. Unfortunately the log transformation does not work here either. This is not a surprise, by the way, because the outliers are on both sides of the box.

The name of the test that works here is **Wilcoxon Signed Rank Test**.

The details are

```
wilcox.test(euros$Weight, mu=7.5)
```

```
##
##  Wilcoxon signed rank test with continuity correction
##
## data:  euros$Weight
## V = 1595000, p-value < 2.2e-16
## alternative hypothesis: true location is not equal to 7.5
```

1) Parameter of interest: 1 median

2) Method of analysis: Wilcoxon Signed Rank test

3) Assumptions of Method: **none**

4) $\alpha = 0.05$

5) Null hypothesis H_0: M=7.5 (median weight is 7.5 grams)

6) Alternative hypothesis H_a: M $\neq$ 7.5 (median weight is not 7.5 grams)

7) p value = 0.000

8) 0.000<0.05, so we reject H_0, it seems the median weight is not 7.5 grams.

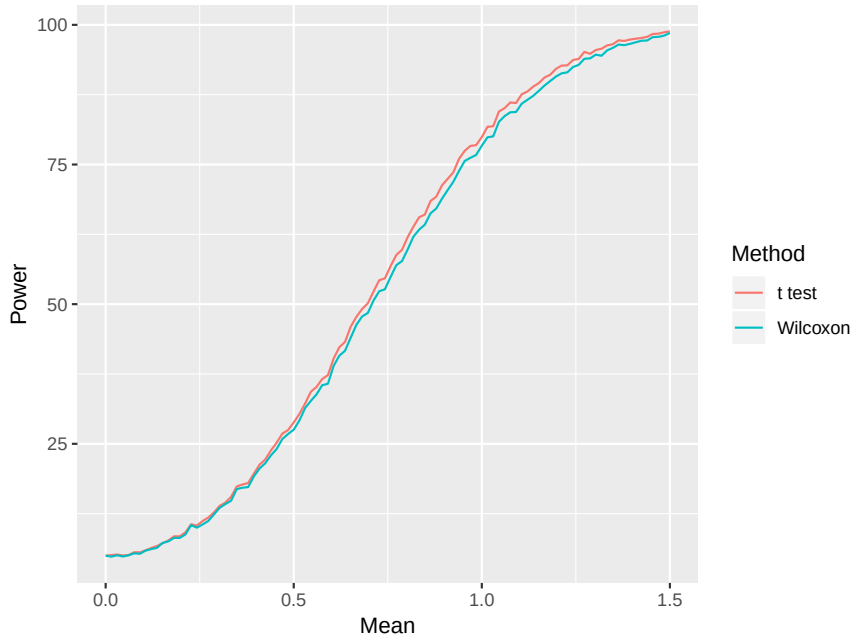
Actually, in this data set we could still have used the usual 1-sample t test (also with a p-value of 0.000) because we have a very large sample (n=2000), but in general it is never clear exactly how large a sample needs to be to "overcome" some outliers, so these non-parametric tests are always a safe alternative.

Why not always use the non-parametric test?

If using the t test sometimes is wrong but the Wilcoxon Rank Sum test always works, why not just always use this test and be safe? The answer is that the t test has a larger power:

```
mu <- seq(0, 1.5, length=100)
pw <- matrix(0, 100, 2)
colnames(pw) <- c("t test", "Wilcoxon")
B <- 10000
for(i in 1:100) {
  for(j in 1:B) {
    x <- rnorm(10, mu[i])
    pval <- t.test(x, mu=0)$p.value
    if(pval<0.05) pw[i, 1] <- pw[i, 1]+1
    pval <- wilcox.test(x, mu=0)$p.value
    if(pval<0.05) pw[i, 2] <- pw[i, 2]+1
  }
}
pw <- 100*pw/B
```

```
df <- data.frame(
        Mean=c(mu, mu),
        Power=c(pw[, 1], pw[, 2]),
        Method=rep(c("t test", "Wilcoxon"), each=100))
ggplot(df, aes(Mean, Power, color=Method)) +
  geom_line()
```



In real life the power of the nonparametric tests is often almost as high as the power of the standard tests, so they should always be used if there is a question about the normal assumption.

If we wanted a 90% confidence interval for median we could use

```
wilcox.test(euros$Weight,
            conf.int=TRUE,
            conf.level=0.9)$conf.int
```

```
## [1] 7.519540 7.522448
## attr(,"conf.level")
## [1] 0.9
```

Alternative to two sample t

just as with the t.test command, the wilcox.test command can also be used to compare the means of two populations:

Example: Euro Coins

Are the means of the weights of the coins in rolls 7 and 8 different?

```
x <- euros$Weight[euros$Roll==7]
y <- euros$Weight[euros$Roll==8]
wilcox.test(x, y)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
```

```
## data:  x and y
## W = 35619, p-value = 0.00684
## alternative hypothesis: true location shift is not equal to 0
```

Alternative to ANOVA

Example: Euro Coins

Say we want to know whether the coins in the 8 different rolls have the same average weight. The non-parametric alternative to the oneway ANOVA is the **Kruskal-Wallis test:**

```
kruskal.test(Weight~factor(Roll), data=euros)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  Weight by factor(Roll)
## Kruskal-Wallis chi-squared = 97.5, df = 7, p-value < 2.2e-16
```

1) Parameters of interest: **medians**
2) Method of analysis: Kruskal-Wallis
3) Assumptions of Method: **none**
4) $\alpha = 0.05$
5) Null hypothesis H_0: M_1=..=M_8 (group **medians** are the same)
6) Alternative hypothesis H_a: $M_i \neq M_j$ for some i, j(group medians are not the same)
7) p value = 0.00
8) $0.00 < 0.05$, so we reject H_0, it seems the group medians are not the same

Example: Cultural Differences in Equipment Use

A US company manufactures equipment that is used in the production of semiconductors. The firm is considering a costly redesign that will improve the performance of its equipment. The performance is characterized as mean time between failures (MTBF). Most of the companies customers are in the USA, Europe and Japan, and there is anecdotal evidence that the Japanese customers typically get better performance from the users in the USA and Europe.

```
head(culture)
```

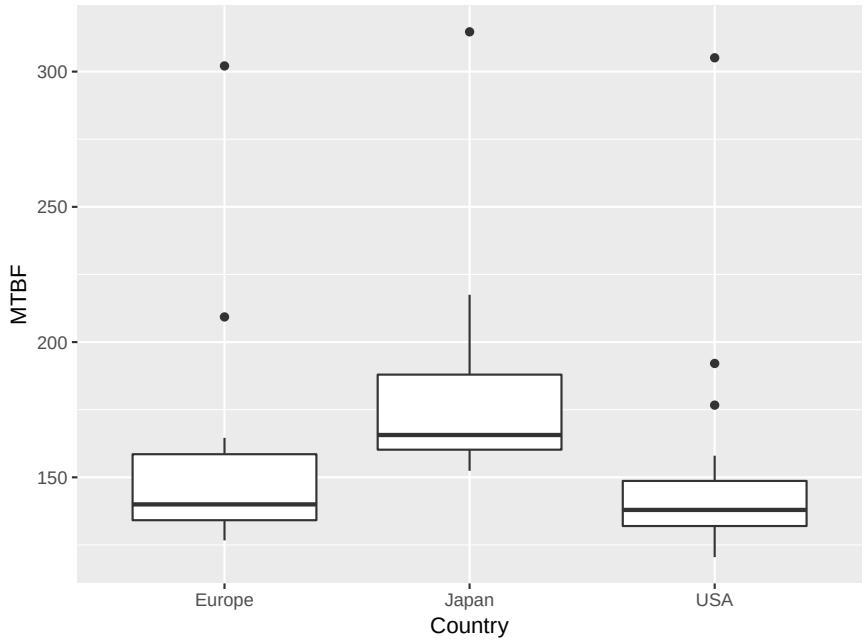
```
##   Country  MTBF
## 1     USA 120.5
## 2     USA 127.1
## 3     USA 128.1
```

```
## 4     USA 129.7
## 5     USA 130.8
## 6     USA 132.4
```

```
table(culture$Country)
```

```
##
## Europe  Japan    USA
##     15     12     20
```

```
ggplot(culture, aes(Country, MTBF)) +
  geom_boxplot()
```



There is a problem with the normal assumption. We can try to fix this with the log transform, but again this does not work.

Because none of the transformations worked we will use the non-parametric Kruskall-Wallis test:

```
kruskal.test(MTBF~factor(Country), data=culture)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  MTBF by factor(Country)
## Kruskal-Wallis chi-squared = 13.806, df = 2, p-value = 0.001005
```

1) Parameters of interest: medians

2) Method of analysis: Kruskal-Wallis

3) Assumptions of Method: **none**

4) $\alpha = 0.05$

5) Null hypothesis H_0: $M_1 = M_2 = M_3$ (group medians are the same)

6) Alternative hypothesis H_a: $M_i \neq M_j$ for some i, j (group medians are not the same)

7) p value = 0.001

8) $0.001 < 0.05$, so we reject H_0, it seems the group medians are not the same, the MTBF is different in different countries

If we had just done the ANOVA Country would not have been stat. significant (p-value = 0.098) but if you remember to check the normal plot you will see that there is a problem with this analysis.

Model Checking

Most discussions in Statistics start with a sentence like this:

we have observations $x_1, .., x_n$ *from a normal distribution. . .*

and so everything that follows depends on the assumption. But how do we know that a data set comes from a certain distribution in real life?

Graphical Checks

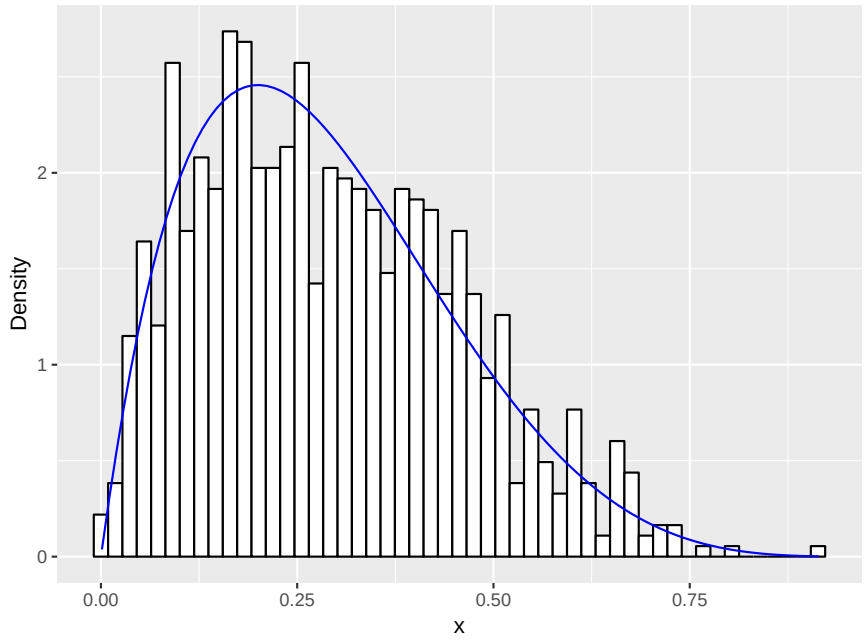
The most common checks we do are graphical.

- **Histogram with Fit**

we draw a histogram of the data, scaled to have total area 1, and overlay it with the theoretical curve:

```
x <- rbeta(1000, 2, 5)
```

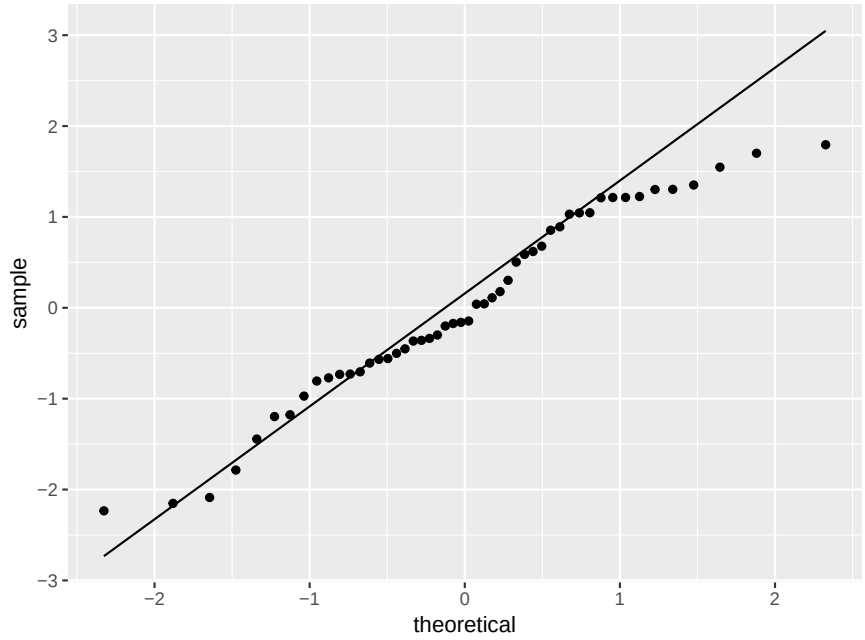
```
bw <- diff(range(x))/50
ggplot(data.frame(x=x), aes(x)) +
 geom_histogram(aes(y = ..density..),
    color = "black",
    fill = "white",
    binwidth = bw) +
    labs(x = "x", y = "Density") +
  stat_function(fun = dbeta, colour = "blue",
          args=list(shape1=2, shape2=5))
```



this works fine if we have sufficient data to do a histogram. If not we have

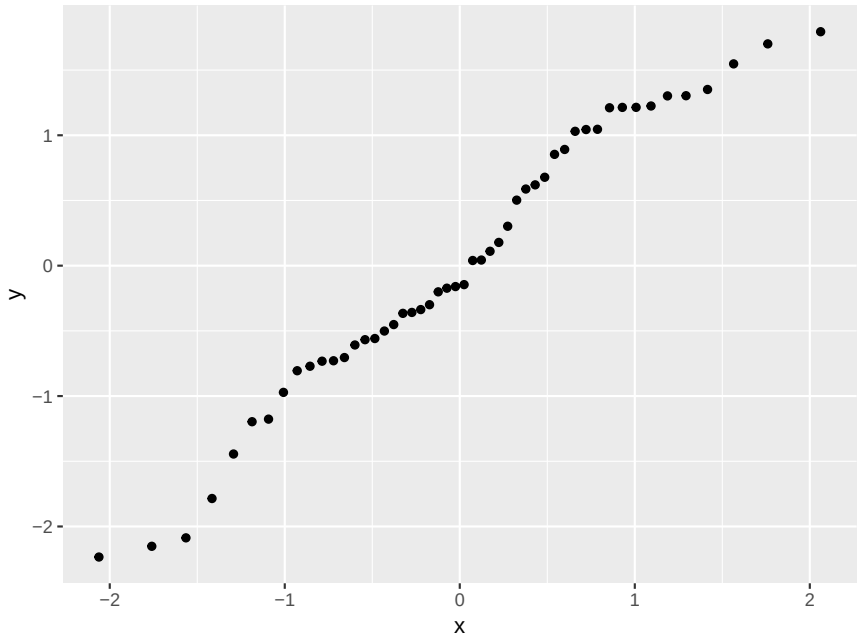
- **Probability Plot**

```
x <- rnorm(50)
ggplot(data=data.frame(x=x), aes(sample=x)) +
          geom_qq() + geom_qq_line()
```



What is drawn here? As the axis say, it is *sample* vs *theoretical*. Specifically it is the *quantiles* of the data set vs the quantiles of the distribution we are checking for:

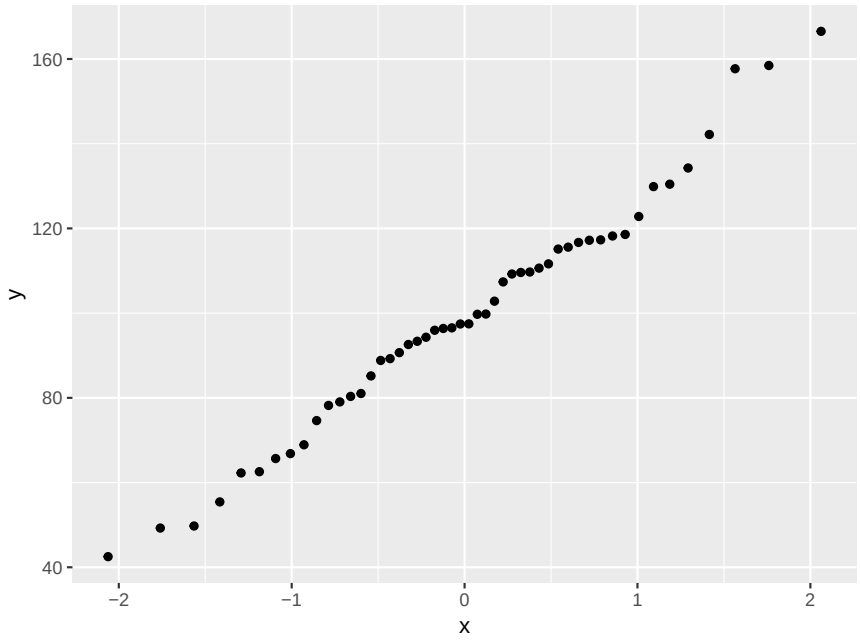
```
df <- data.frame(x=qnorm(1:50/51), y=sort(x))
ggplot(data=df, aes(x, y)) +
  geom_point()
```



and the line is drawn through the *quartiles*. It can be shown that if the data indeed comes from this distribution the points should fall along a straight line.

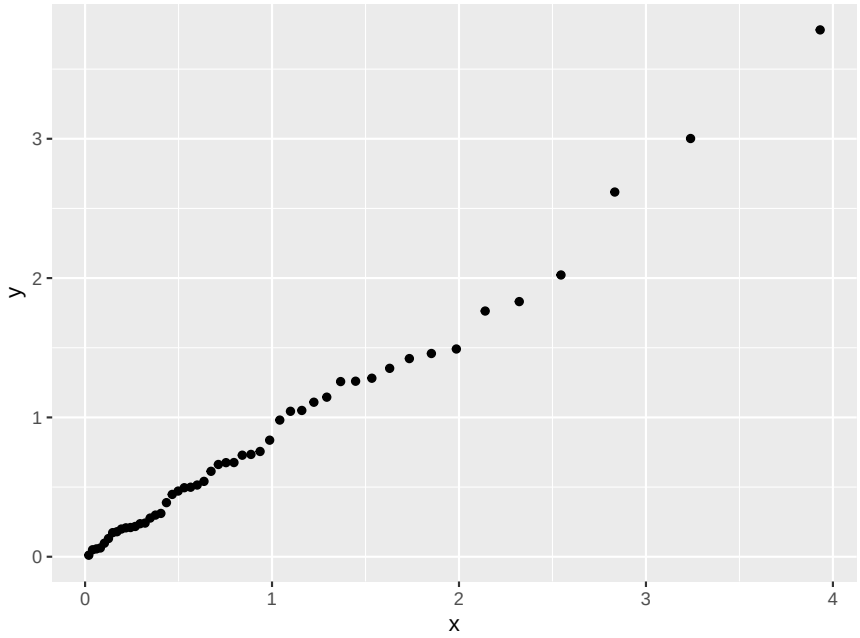
Note that this graph is *scale invariant*:

```
x <- rnorm(50, 100, 30)
df <- data.frame(x=qnorm(1:50/51), y=sort(x))
ggplot(data=df, aes(x, y)) +
  geom_point()
```



It works for other distributions as well:

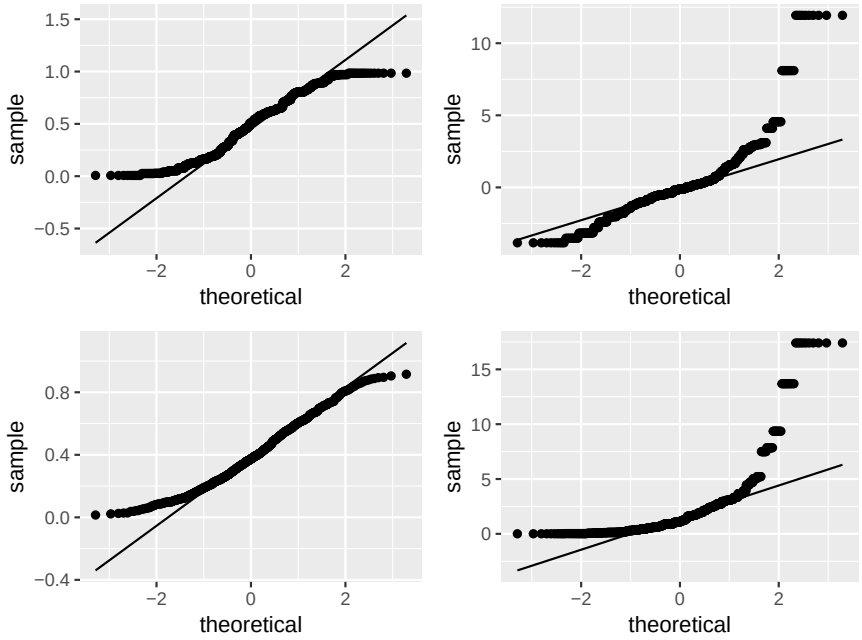
```
x <- rexp(50, 1)
df <- data.frame(x=qexp(1:50/51), y=sort(x))
ggplot(data=df, aes(x, y)) +
  geom_point()
```



here are some examples where the distribution is not normal:

```
df <- data.frame(x1=runif(100),
                 x2=rt(100, 2),
```

```
                x3=rbeta(1000, 2, 3),
                x4=rchisq(100, 2))
pushViewport(viewport(layout = grid.layout(2, 2)))
print(ggplot(data=df, aes(sample=x1)) +
          geom_qq() + geom_qq_line() ,
    vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(ggplot(data=df, aes(sample=x2)) +
          geom_qq() + geom_qq_line() ,
    vp=viewport(layout.pos.row=1, layout.pos.col=2))
print(ggplot(data=df, aes(sample=x3)) +
          geom_qq() + geom_qq_line() ,
    vp=viewport(layout.pos.row=2, layout.pos.col=1))
print(ggplot(data=df, aes(sample=x4)) +
          geom_qq() + geom_qq_line() ,
    vp=viewport(layout.pos.row=2, layout.pos.col=2))
```



with some experience it is possible to tell from the shape of the graph in which way the true distribution differs from the normal (maybe it has longer tails, is skewed etc.)

Formal Tests

There are a number of hypothesis tests one can use as well. The most important is the

- **Chisquare Goodness of Fit Test**

Example: Experiments in Plant Hybridization (1865)

by Gregor Mendel is one of the most famous papers in all of Science. His theory of genetics predicted that the number of Smooth yellow, wrinkled yellow, smooth green and wrinkled green peas would be in the proportions 9:3:3:1. In one of his experiments he observed 315, 101, 108 and 32. Does this agree with his theory?

How does this fit into our current discussion? Essentially his theory said that peas appear according to a *multinomial distribution* with parameters $m = 4, p = (9/16, 3/16, 3/16, 1/16)$.

One can show that the likelihood ratio test (together with some approximations) leads to the famous **chisquare statistic**

$$\chi^2 = \sum \frac{(O-E)^2}{E}$$

where O are the observed counts and E are the expected counts. Under the null hypothesis χ^2 has a chisquare distribution with m-1 degrees of freedom.

For Mendels data we find

```
O <- c(315, 101, 108, 32)
p <- c(9/16, 3/16, 3/16, 1/16)
E <- sum(O)*p
chi <- sum((O-E)^2/E)
c(chi, 1-pchisq(chi, 3))
```

```
## [1] 0.4700240 0.9254259
```

and so we fail to reject the null hypothesis, Mendels theory works.

This is a *large-sample* test (because of the approximations). The general requirement is that $E > 5$.

The chisquare statistic was already known in the mid 19th century but its distribution was derived by Karl Pearson in 1900. His argument was as follows: O is the sum of indicator random variables (X_i is of type i or not), so O has a binomial distribution, and if n is large enough

$$(O-E)/\sqrt{E} \sim N(0,1)$$

Therefore

$$(O-E)^2/E \sim \chi^2(1)$$

Finally

$$\sum (O-E)^2/E \sim \chi^2(m-1)$$

because there is one "restriction", namely $\sum O = n$.

Example: Death by kicks from a Horse

Number of deaths by horse kicks in the Prussian army from 1875-1894 for 14 Corps.

```
kable.nice(horsekicks)
```

Year	Deaths
1875	3
1876	5
1877	7
1878	9
1879	10
1880	18
1881	6
1882	14
1883	11
1884	9
1885	5
1886	11
1887	15
1888	6
1889	11
1890	17
1891	12
1892	15
1893	8
1894	4

Some theoretical arguments make it a reasonable guess that this data should follow a Poisson distribution, that is

$$P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda}$$

we want to test this. That is we have

$$H_0 : X \sim \text{Poisson}$$

Notice though that this does not say what λ is.

The idea is the following: if the Poisson model works at all, it should work for the value of λ that minimizes the chisquare statistic. So if we denote this number by $\hat{\lambda}$, we should test

$$H_0 : X \sim \text{Poisson}(\hat{\lambda})$$

However, this estimation will cost us a degree of freedom, so the distribution of the statistic is now chisquare with m-1-k degrees of freedom, where k is the number of parameters estimated.

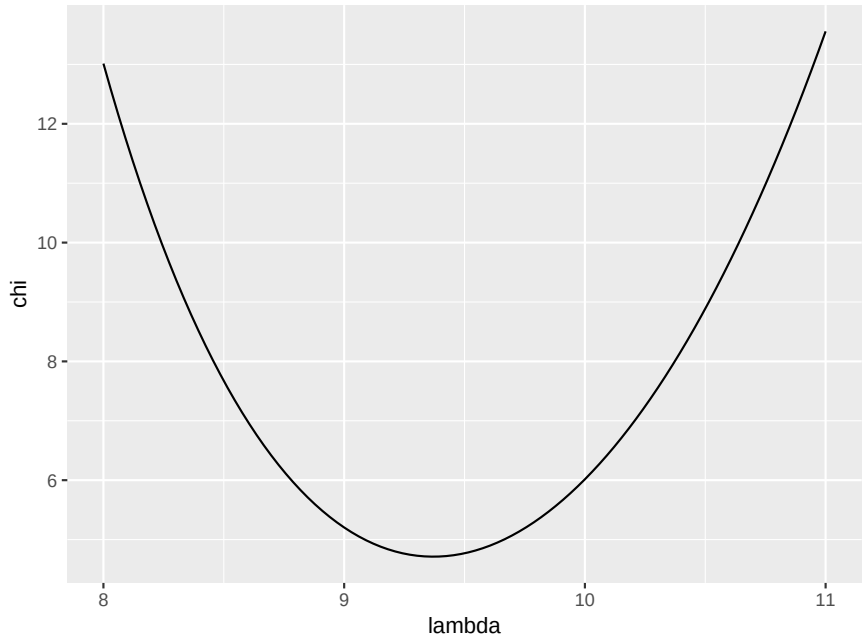
Note that this requires an unusual estimation technic, called *minimum chisquare*. In practice people just use *maximum likelihood*, but this is not always going to work.

We have another issue: in some years there were very few deaths, so the E would be small, less than 5. We can deal with this by grouping the data:

```
df <- data.frame(Period=c("0-6", "7-9", "10-12", "Over 12"),
                 Counts=c(6, 4, 5, 5))
kable.nice(df)
```

Period	Counts
0-6	6
7-9	4
10-12	5
Over 12	5

```
chi.fun <- function(lambda) {
  p <- c(ppois(6, lambda), sum(dpois(7:9, lambda)),
         sum(dpois(10:12, lambda)), 1-ppois(12, lambda))
  E <- 20*p
  sum((df$Counts-E)^2/E)
}
lambda <- seq(8, 11, length=100)
y <- lambda
for(i in 1:100) y[i] <- chi.fun(lambda[i])
df1 <- data.frame(lambda=lambda,
                  chi=y)
ggplot(data=df1, aes(lambda, chi)) +
  geom_line()
```




```
lambda[y==min(y)]
```

```
## [1] 9.363636
```

Notice that in this case the minimum chisquare estimate is quite different from the mle, which is

```
mean(horsekicks[, 2])
```

```
## [1] 9.8
```

now

```
tmp <- chi.fun(9.36)
c(tmp, 1-pchisq(tmp, 2))
```

```
## [1] 4.71293447 0.09475438
```

and so we find weak evidence for the Poisson distribution.

Notice that the binning we did of the data was completely arbitrary.

Notice that this test differs from those we discussed previously: it does not have an alternative hypothesis. Of course we could just have used

$$H_a : X \not\sim \text{Poisson}$$

but that seems rather pointless. In fact, this is part of a larger discussion, the difference between *Fisherian* and *Neyman-Pearson* hypothesis testing which would however lead us to far away.

The adjustment of the degrees of freedom for the number of estimated parameters has an interesting history. It does not appear in Pearson's original derivation. In fact, following Pearson's logic there should be no need for this adjustment, because if the sample size is large enough any parameter should be estimated with sufficiently high precision. The need for the adjustment was recognized only 20 years after the original publication of Pearson by none other than Egon Pearson (Karl's son) and by Sir Ronald Fisher and is now sometimes called the Fisher-Pearson statistic.

In the case of continuous distributions this becomes even more complicated because now there are infinitely many ways to bin the data. The are two main strategies:

- equal size
- equal probability

in general the second one is recommended.

Example: Euro coins

Do the weights follow a normal distribution?

Let's test this with k=10 (??) equal probability bins.

Again we need to estimate the parameters. Because this is a large sample we will use the mle's:

```
round(c(mean(euros$Weight), sd(euros$Weight)), 4)
```

```
## [1] 7.5212 0.0344
```

```
bins <- c(7, quantile(euros$Weight, 1:9/10), 8)
pb <- sprintf("%.3f", bins)
O <- hist(euros$Weight, breaks = bins, plot=FALSE)$counts
E <- round(2000*diff(pnorm(bins, 7.5212, 0.0344)), 1)
df <- cbind(Bins=paste0(pb[-11],"-",pb[-1]),
            O, E=sprintf("%.1f", E))
rownames(df) <- NULL
kable.nice(df)
```

Bins	O	E
7.000-7.480	204	231.0
7.480-7.492	200	164.9
7.492-7.503	210	200.8
7.503-7.512	206	192.4
7.512-7.520	190	183.0
7.520-7.529	202	207.2
7.529-7.538	192	195.3
7.538-7.551	220	238.9
7.551-7.566	179	193.5
7.566-8.000	197	192.8

```
chi <- sum((O-E)^2/E)
c(round(chi, 2), round(1-pchisq(chi, 10-1-2), 3))
```

```
## [1] 15.140  0.034
```

and so we reject the null hypothesis at the 5% level, this data does not come from a normal distribution.

It should be clear that there are many issues here:

- how to estimate the parameters
- how to bin
- how many bins

and all of these can lead to different results.

Tests based on the empirical distribution function

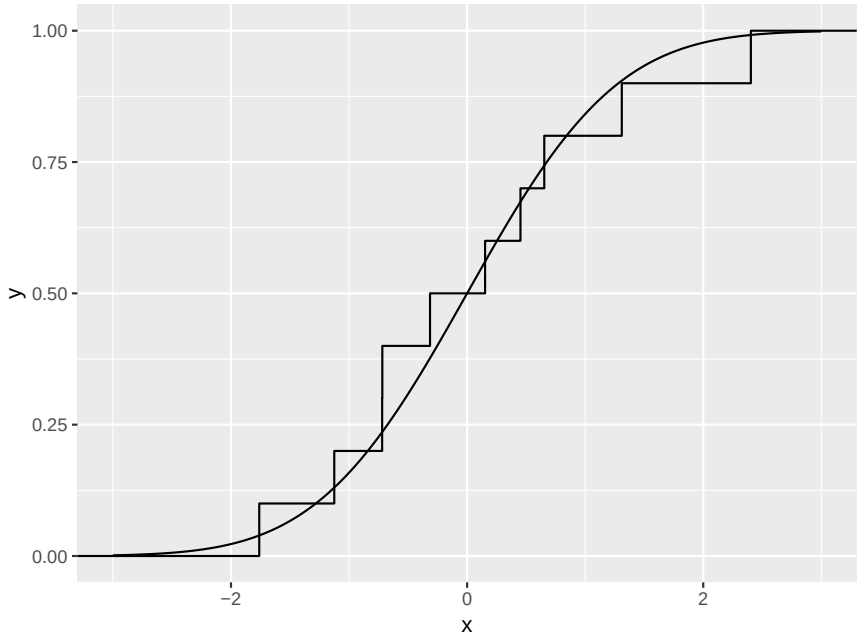
Recall the definition of the *empirical distribution function*:

$$\hat{F}(x) = \frac{1}{n} \sum I_{(-\infty, x)}(x_i)$$

Example: Artificial example

```
set.seed(112)
df <- data.frame(x=rnorm(10))
x=seq(-3, 3, length=250)
df1 <- data.frame(x=x,  y=pnorm(x))
ggplot(df, aes(x)) +
```

```
  stat_ecdf(geom = "step") +
  geom_line(data=df1, aes(x, y))
```



There are a number of tests based on some measure of "distance" between these two curves.

- **Kolmogorov-Smirnov**

$$D = \max\left\{|F(x) - \hat{F}(x)|; x \in R\right\}$$

this is implemented in *ks.test*

```
ks.test(euros$Weight, "pnorm", mean=7.521, sd=0.0344)
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  euros$Weight
## D = 0.022955, p-value = 0.2426
## alternative hypothesis: two-sided
```

notice however that this requires specific values of the parameters.

In the case of the normal distribution Lilliefors derived a test based on this statistic that allows estimation of the parameters:

```
library(nortest)
lillie.test(euros$Weight)
```

```
##
##  Lilliefors (Kolmogorov-Smirnov) normality test
##
```

```
## data:  euros$Weight
## D = 0.023354, p-value = 0.01311
```

and we see that this test correctly rejects the null.

An often better alternative to Kolmogorov-Smirnov is the

- **Anderson-Darling test**

it uses

$$D = n \int_{-\infty}^{\infty} \frac{(F(x) - \hat{F}(x))^2}{F(x)(1 - F(x))} dx$$

essentially this gives greater weight to the tail of the distribution, where F(x) and 1-F(x) are small.

The *ad.test* in R tests for composite normality:

```
ad.test(euros$Weight)
```

```
##
##  Anderson-Darling normality test
##
## data:  euros$Weight
## A = 1.6213, p-value = 0.0003646
```

Null distribution via simulation

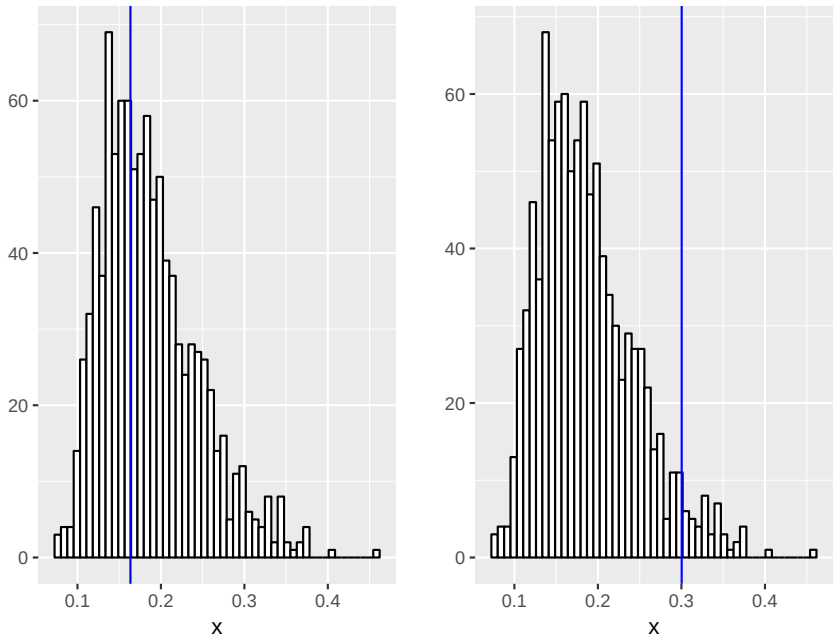
Let's say we wish to test whether a data set comes from an exponential distribution, and we want to use the Kolmogorov-Smirnov statistic. Now we need to estimate the rate, and so the basic test won't work. We can however do this:

- generate data from an exponential with the rate equal to the mle of the data.
- find the KS statistic
- repeat many time
- compare the results to the KS from the data

```
x1 <- rexp(20, 1)
x2 <- rgamma(20, 2, 1)
rt1 <- 1/mean(x1)
rt2 <- 1/mean(x2)
B <- 1000
ks.sim <- matrix(0, B, 2)
for(i in 1:B) {
  ks.sim[i, 1] <- ks.test(rexp(20, rt1),
                          "pexp", rate=rt1)$statistic
    ks.sim[i, 2] <- ks.test(rexp(20, rt2),
                          "pexp", rate=rt2)$statistic
```

```
}
ks.dat <- c(ks.test(x1, "pexp", rate=rt1)$statistic,
            ks.test(x2, "pexp", rate=rt2)$statistic)
```

```
pushViewport(viewport(layout = grid.layout(1, 2)))
bw1 <- diff(range(ks.sim[, 1]))/50
bw2 <- diff(range(ks.sim[, 2]))/50
print(ggplot(data.frame(x=ks.sim[, 1]), aes(x)) +
  geom_histogram(color = "black",
                 fill = "white",
                 binwidth = bw1) +
  labs(x = "x", y = "") +
    geom_vline(xintercept=ks.dat[1], color="blue"),
  vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(ggplot(data.frame(x=ks.sim[, 1]), aes(x)) +
  geom_histogram(color = "black",
                 fill = "white",
                 binwidth = bw2) +
  labs(x = "x", y = "") +
    geom_vline(xintercept = ks.dat[2], color="blue"),
  vp=viewport(layout.pos.row=1, layout.pos.col=2))
```




```
sum(ks.sim[, 1]>ks.dat[1])/B
```

```
## [1] 0.6
```

```
sum(ks.sim[, 2]>ks.dat[2])/B
```

```
## [1] 0.049
```

and this test does not rely on any probability theory.

Simple Regression (One Predictor)

In this chapter we will discuss so called linear models. These are models of the form

$$Y = \beta_0 + \beta_1 x_1 + .. + \beta_k x_k + \epsilon$$

at first it seems rather restrictive to only consider linear models, but in fact these are quite general. For one, the models are linear in the parameters, so for example

$$\begin{aligned} Y &= aX^b \\ \log Y &= \log(ax^b) = \\ \log a &+ b \log x \end{aligned}$$

is also a linear model!

In this section we discuss *simple regression*, which means models with just one predictor.

Least Squares Regression

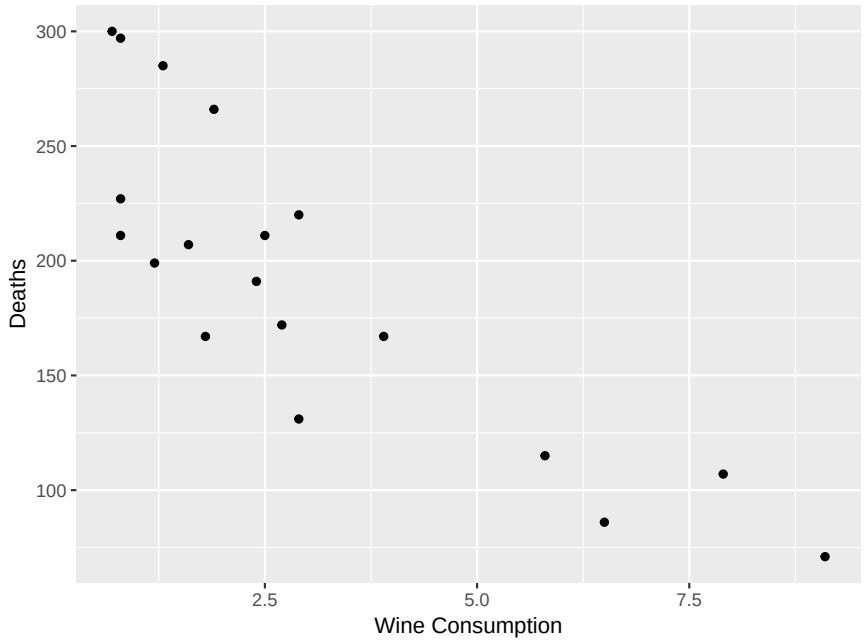
Example: Wine Consumption and Heart Desease

data on wine consumption per person per year and deaths from heart disease per 100000, by country

```
kable.nice(wine)
```

Country	Wine.Consumption	Heart.Disease.Deaths
Australia	2.5	211
Austria	3.9	167
Belgium	2.9	131
Canada	2.4	191
Denmark	2.9	220
Finland	0.8	297
France	9.1	71
Iceland	0.8	211
Ireland	0.7	300
Italy	7.9	107
Netherlands	1.8	167
New Zealand	1.9	266
Norway	0.8	227
Spain	6.5	86
Sweden	1.6	207
Switzerland	5.8	115
United Kingdom	1.3	285
United States	1.2	199
Germany	2.7	172

```
ggplot(wine,
       aes(Wine.Consumption, Heart.Disease.Deaths)) +
  geom_point() +
  labs(x="Wine Consumption", y="Deaths")
```



We want to fit a linear model, that is a straight line. We will use the **method of least squares**. The idea is as follows:

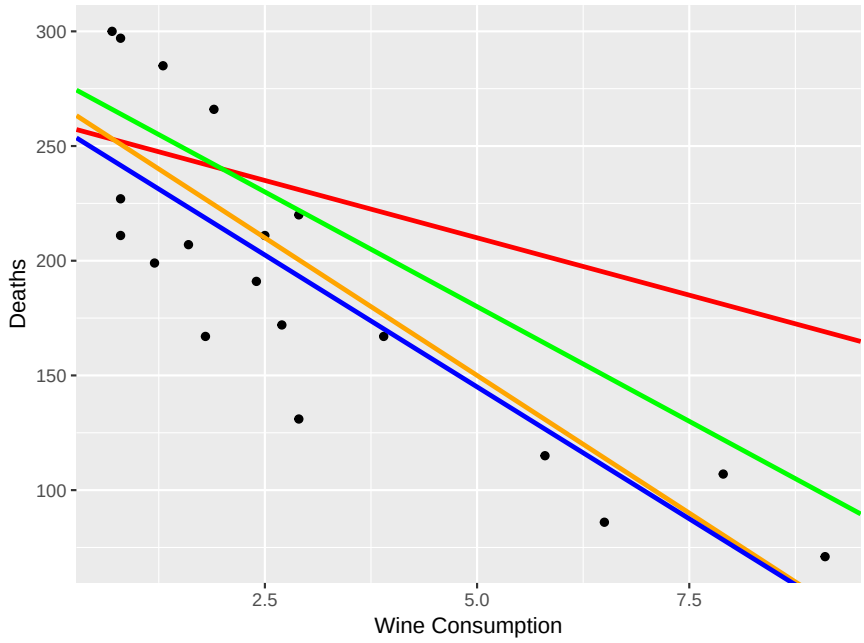
say the model is

$$\text{heart disease} = 260 - 10 \times \text{ wine consumption}$$

and we know that for a certain country (not in the data set) wine consumption is 3.7, then according to our model the heart disease rate should be about

$$\text{heart disease} = 260 - 10 \times 3.7 = 223$$

How do we find an equation? Well, to find some equation is easy:



clearly the red line is not very good (to flat), the green one is better but still a bit to flat, but how about the orange and blue ones? Both look reasonably good.

Is there a way to find a line that is "best" ? The answer is yes. In order to understand how we need to following:

Let's concentrate for a moment on the third line, which has the equation

$$\text{heart disease} = 270 - 24 * \text{ wine consumption}$$

or short $y = 270 - 24x$

The United States has a wine consumption of $x = 1.2$ liters and a heart disease rate of $y = 199$. Now if we did not know the heart disease rate we could use the equation and find

$$y = 270 - 24x = 270 - 24 * 1.2 = 241$$

Now we have 2 y's:

- the one in the data ($y = 199$)
- the one from the equation ($y = 241$)

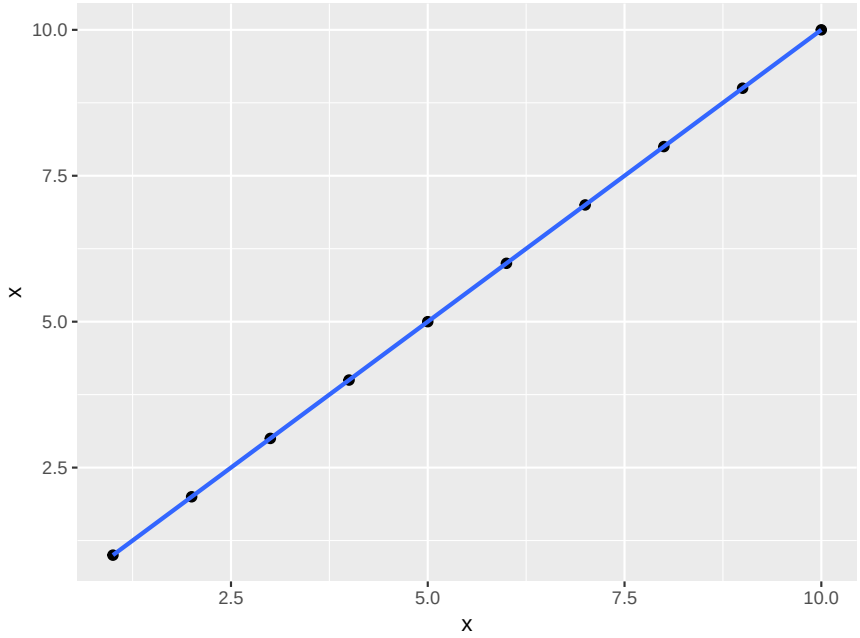
Let distinguish between them by calling the first the **observed value** and the second one the **fitted value**.

Think of it in these terms: the fitted value is our guess, the observed value is the truth. So the difference between them is the **error** in our guess. We call this the **residual**:

$$\epsilon = \text{fitted} - \text{observed} = 241 - 199 = 42$$

The line $y = 270 - 24x$ **overestimates** the heart disease rate in the US by 42.

If the line perfectly described the data, the residuals would all be 0:



This was done for the US, but of course we could do the same for all the countries in the data set:

Country	Consumption	Deaths	Fits	Residuals
Australia	2.5	211	210.0	1.0
Austria	3.9	167	176.4	-9.4
Belgium	2.9	131	200.4	-69.4
Canada	2.4	191	212.4	-21.4
Denmark	2.9	220	200.4	19.6
Finland	0.8	297	250.8	46.2
France	9.1	71	51.6	19.4
Iceland	0.8	211	250.8	-39.8
Ireland	0.7	300	253.2	46.8
Italy	7.9	107	80.4	26.6
Netherlands	1.8	167	226.8	-59.8
New Zealand	1.9	266	224.4	41.6
Norway	0.8	227	250.8	-23.8
Spain	6.5	86	114.0	-28.0
Sweden	1.6	207	231.6	-24.6
Switzerland	5.8	115	130.8	-15.8
United Kingdom	1.3	285	238.8	46.2
United States	1.2	199	241.2	-42.2
Germany	2.7	172	205.2	-33.2

so for each country our line makes an error. What we need is a way to find an **overall** error. The idea of least squares is to find the **sum of squares** of the residuals:

$$RSS = \sum \epsilon^2$$

In the case of our line we find

$$RSS = (-1.0)^2 + 9.4^2 + .. + 33.2^2 = 25269.8$$

In the same way we can find an RSS for any line:

- y = 280-10x , RSS = 71893
- y = 260-20x , RSS = 40738
- y = 260-23x , RSS = 24399.7

notice that the first two, which we said were not so good, have a higher RSS. So it seems that the lower the RSS, the better. Is there a line with the smallest RSS possible? The answer is again yes, using the method of **Least Squares** for which we have the routine:

```
fit <- lm(Heart.Disease.Deaths~Wine.Consumption,
    data=wine)
round(fit$coef, 2)
```

```
##      (Intercept) Wine.Consumption
##           260.56           -22.97
```

The least squares regression equation is:

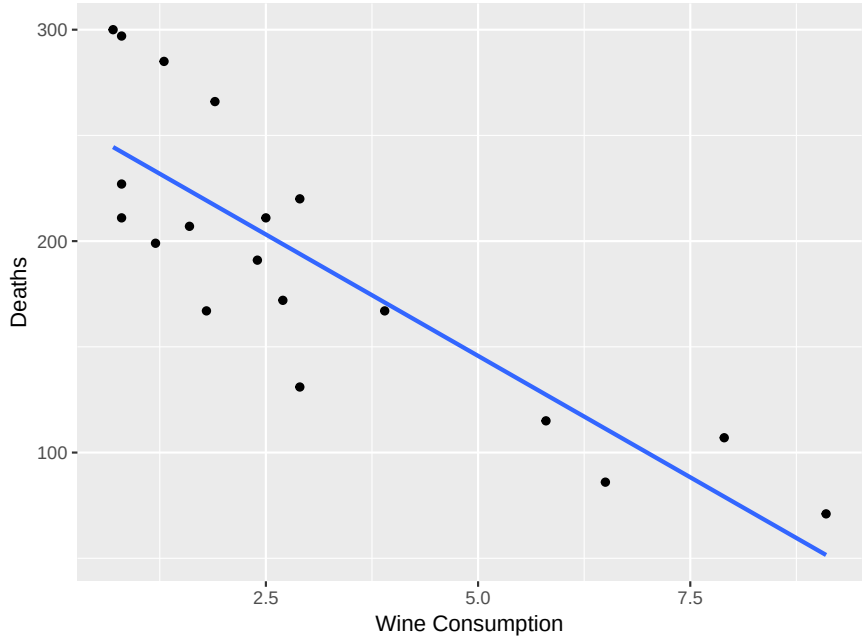
$$\text{heart disease} = 260.56 - 22.97 \text{ wine consumption}$$

very close to the last of our equations.

What is its RSS? It is not part of the output, but I can tell you it is 24391.

A nice graph to visualize the model is the scatterplot with the least squares regression line, called the **fitted line plot**

```
plt +
  geom_smooth(method = "lm", se=FALSE)
```



Alternatives to Least Squares

Instead of minimizing the sum of squares we could also have

- minimized the largest absolute residual
- minimized the sum of the absolute residuals
- some other figure of merit.

Historically least squares was use mostly because it could be done analytically:

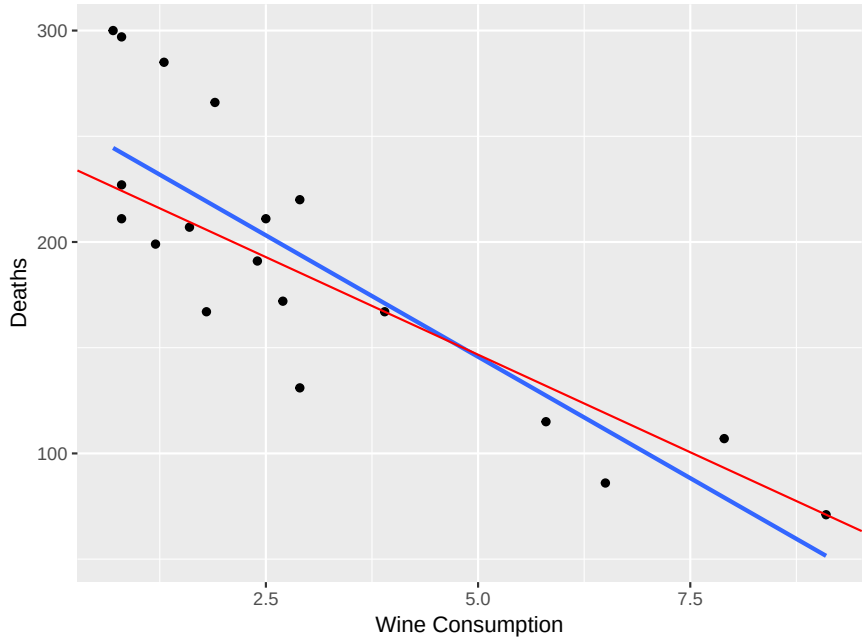
$$\frac{d}{d\beta_0}\sum(y_i-\beta_0-\beta_1x_i)^2=$$
$$(-2)\sum(y_i-\beta_0-\beta_1x_i)=$$
$$(-2)\left(\sum y_i-n\beta_0-\beta_1\sum x_i\right)=0$$
$$\hat{\beta}_0=\bar{y}-\beta_1\bar{x}$$

and the same for β_1. These days we can use pretty much any criterion we want:

```
fit.abs <- function(beta)
  sum(abs(wine$Heart.Disease.Deaths
          -beta[1]-beta[2]*wine$Wine.Consumption))
round(nlm(fit.abs, c(260, -23))$estimate, 2)
```

```
## [1] 239.01 -18.46
```

```
plt +
  geom_smooth(method = "lm", se=FALSE) +
  geom_abline(slope = -18.46,
              intercept = 239, color="red")
```



This is often called the L_1 regression. This also implemented in

```
library(robustbase)
X <- cbind(1, wine$Wine.Consumption)
lmrob.lar(X, wine$Heart.Disease.Deaths)$coefficients
```

```
## [1] 239.00000 -18.46154
```

which uses a much better algorithm based on the simplex method.

One way to understand the difference between these two is the following: let's use least squares/absolute value to estimate the mean!

So we have the model

$$Y = \beta_0 + \epsilon$$

Using least square (now with $\beta_1 = 0$) yields as above $\hat{\beta}_0 = \bar{y}$, the sample mean. What does absolute error give? It can be shown that it leads to the median!

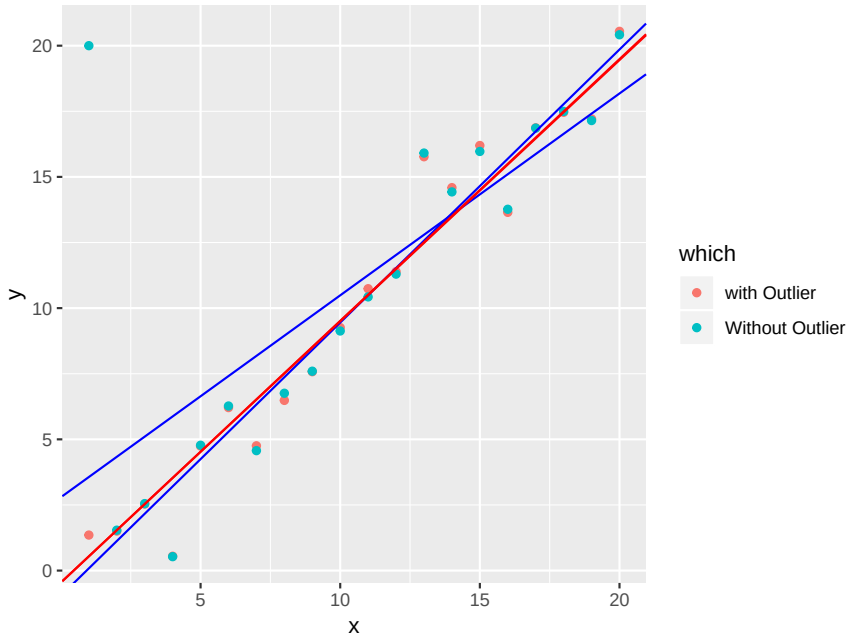
Just as the median is a robust (aka does not depend so much on outliers) estimator than the mean, L_1 estimation also is more robust.

Example: artificial example

```
x <- 1:20
y1 <- x + rnorm(20, 0, 1.5)
y2 <- y1 + rnorm(20, 0, 0.1)
y2[1] <- 20
df <- data.frame(x=c(x, x), y=c(y1, y2),
          which=rep(c("with Outlier", "Without Outlier"), each=20))
lm1 <- coef(lm(y1~x))
lm2 <- coef(lm(y2~x))
l11 <- lmrob.lar(cbind(1, x), y1)$coefficients
l12 <- lmrob.lar(cbind(1, x), y2)$coefficients
print(lm1)
```

```
## (Intercept)           x
##  -0.9558222   1.0405002
```

```
ggplot(df, aes(x, y, color=which)) +
  geom_point() +
  geom_abline(intercept = lm1[1], slope = lm1[2], color="blue") +
  geom_abline(intercept = lm2[1], slope = lm2[2], color="blue") +
  geom_abline(intercept = l11[1], slope = l11[2], color="red") +
  geom_abline(intercept = l12[1], slope = l12[2], color="red")
```



and we see that the effect of the outlier is much larger on the least squares regression than on the L_1.

ANOVA

notice that ANOVA can also be viewed as a linear model, where the predictor variable is categorical. The main difference is that the "model" there is found via the likelihood ratio test rather than least squares and that the main interest is in hypothesis testing rather than prediction.

Assumptions of Least Squares Regression

This page explains the assumptions behind the method of least squares regression and how to check them.

Recall that we are fitting a model of the form

$$y = \beta_0 + \beta_1 x$$

there are **three assumptions**:

1) The model is good (that is, the relationship is linear and not, say, quadratic, exponential or something else)
2) The residuals have a normal distribution
3) The residuals have equal variance (are homoscadastic)

The second and third assumption we are already familiar with from ANOVA and correlation.

We can check these assumptions using two graphs:

- Residual vs. Fits plot: this is just what it says, a scatterplot of the residuals (on y-axis) vs. the fitted values.
- Normal plot of residuals

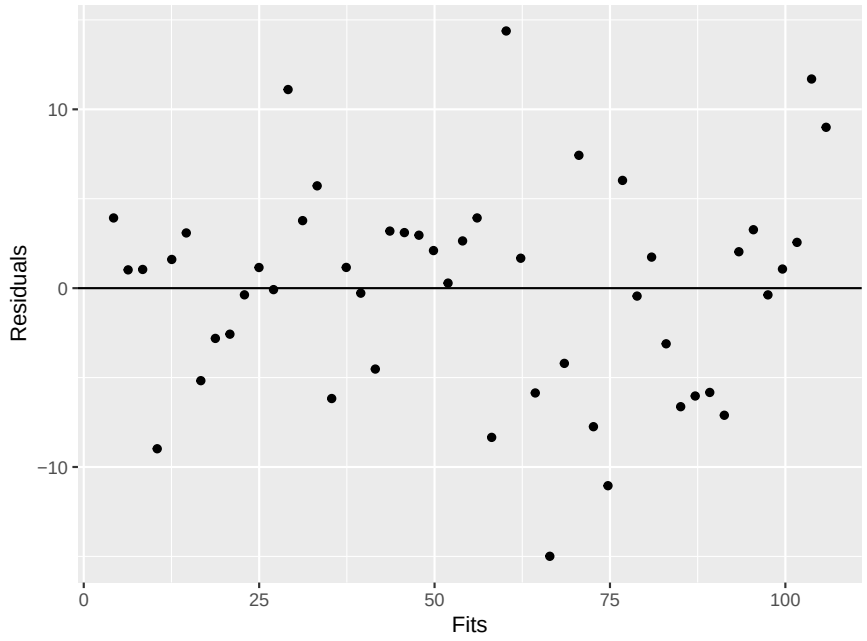
1) Good Model

For this assumption draw the Residuals vs. Fits plot and check for **any pattern**

Example:

Linear model is good:

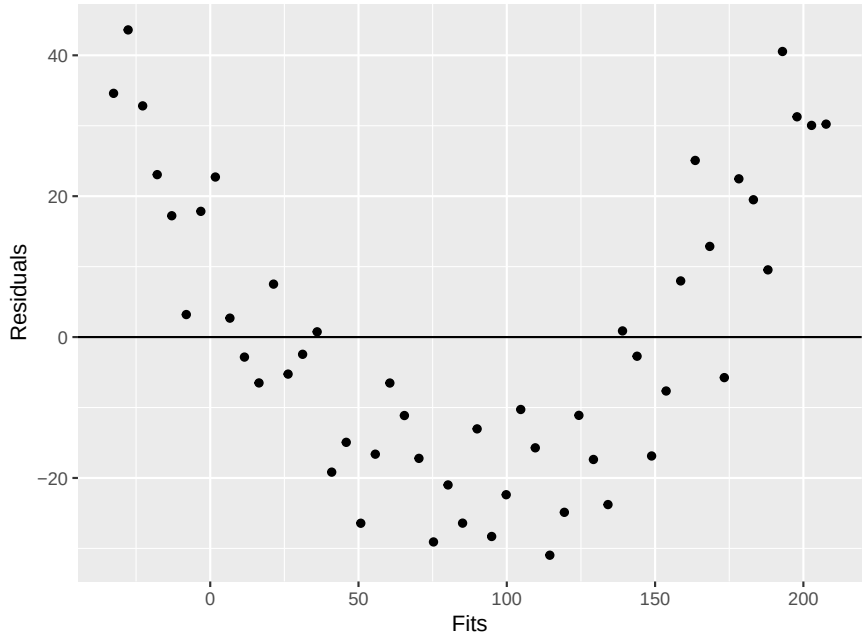
```
x <- 1:50
y <- 5 + 2*x + rnorm(50, 0, 5)
fit <- lm(y~x)
df <- data.frame(Fits=fitted(fit),
                 Residuals=residuals(fit))
ggplot(data=df, aes(Fits, Residuals)) +
            geom_point() +
            geom_hline(yintercept = 0)
```



Linear model is bad:

```
x <- 1:50
y <- 0.1*x^2+rnorm(50, 0, 10)
fit <- lm(y~x)
df <- data.frame(Fits=fitted(fit),
                 Residuals=residuals(fit))
```

```
ggplot(data=df, aes(Fits, Residuals)) +
            geom_point() +
            geom_hline(yintercept = 0)
```



The U shaped pattern in the residual vs. fits plot is a very common one if the linear model is bad.

2) Residuals have a Normal Distribution

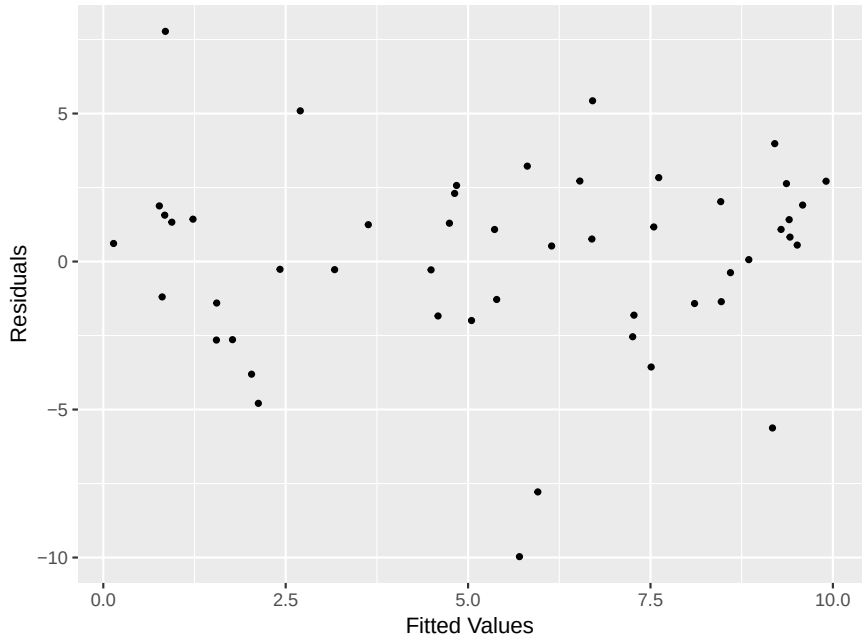
For this assumption draw the normal probability plot and see whether the **dots form a straight line**, just as we have done it many times by now.

3) Residuals have Equal Variance

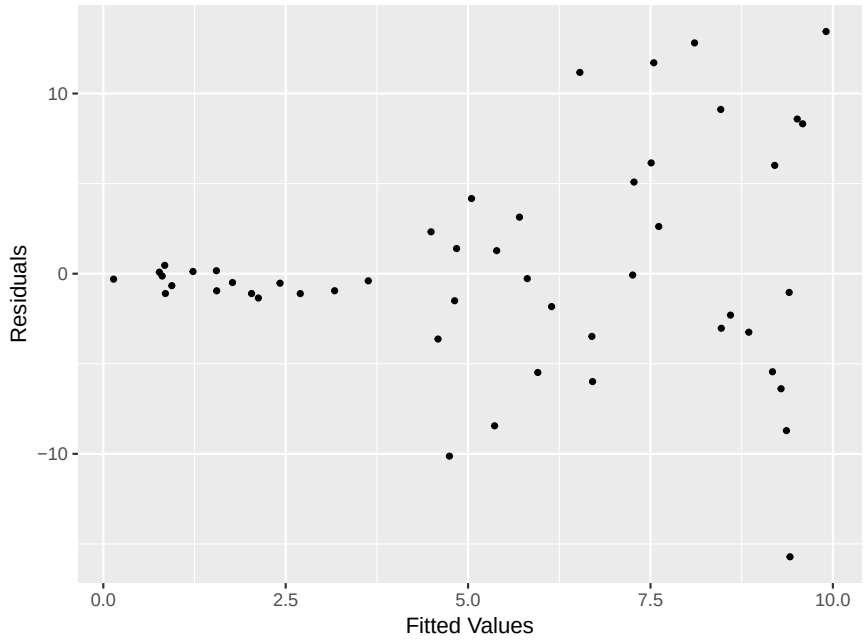
Previously we could check the stdev within the groups and see whether they differed by more than a factor of 3. Now, though we don't have groups. Instead we will again draw the Residuals vs. Fits plot and check whether the *variance (or spread) of the dots changes as you go along the x axis.*

Example:

Equal Variance ok:



Equal Variance not ok:



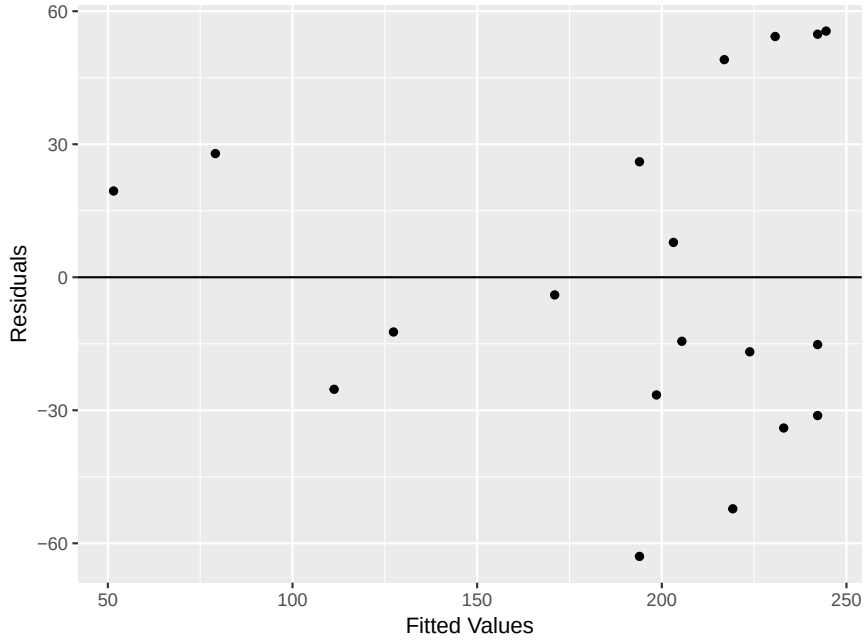
This can be a tricky one to decide, especially if there are few observations.

Example Case Study: Wine Consumption and Heart Disease

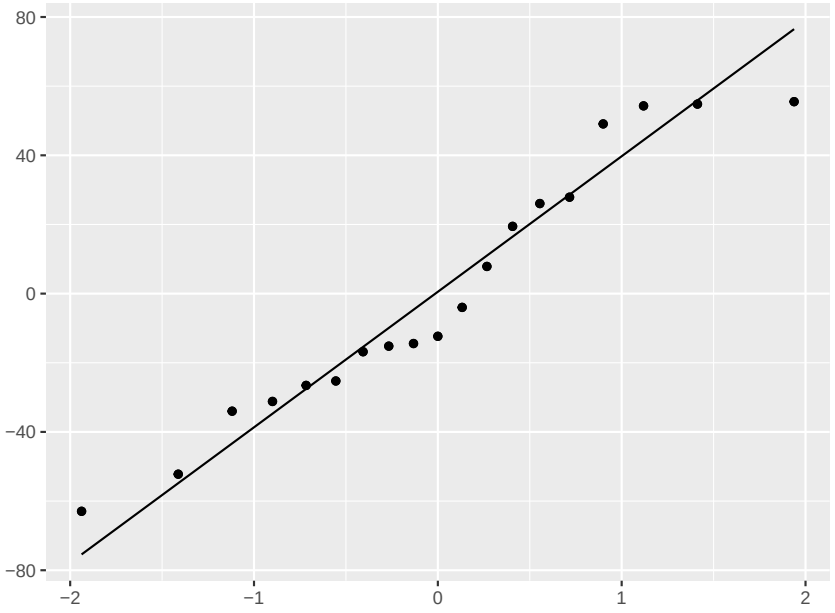
Let's check the assumptions for the wine consumption data:

```
fit <- lm(Heart.Disease.Deaths~Wine.Consumption,
          data=wine)
df <- data.frame(x=fitted(fit), y=resid(fit))
```

```
ggplot(df, aes(x, y)) +
  geom_point() +
  geom_hline(yintercept=0) +
  labs(x="Fitted Values", y="Residuals")
```




```
qplot(data=df, sample=y) +
  stat_qq() + stat_qq_line()
```



the normal plot is fine, and the residual vs. fits plot is fine as far the linear model assumption goes. There is, though, an appearance of unequal variance. This judgement is made more

difficult here, though, because there is very little data in the left half of the graph, and naturally a few dots won't have a large spread. It will take time for you to be able to judge these graphs properly. In fact this one is ok. Not great, but ok.

Note a final decision on whether the assumptions are justified is ALWAYS made based on the Residual vs. Fits Plot and the Normal plot of Residuals.

Prediction

Basic Concept

In this section we want to use a model to make predictions for y for some fixed x.

Example: Quality of Fish

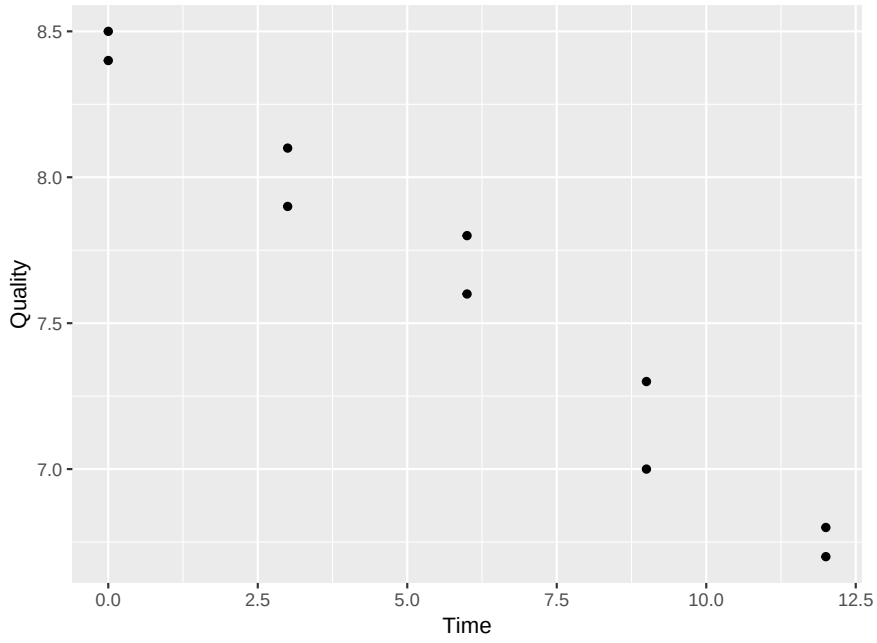
A study was conducted to examine the quality of fish after several days in ice storage. Ten raw fish of the same kind and quality were caught and prepared for storage. Two of the fish were placed in ice storage immediately after being caught, two were placed there after 3 hours, and two each after 6, 9 and 12 hours. Then all the fish were left in storage for 7 days. Finally they were examined and rated according to their "freshness".

Use this data set to estimate the quality of a fish that was put into ice 4 hours after being caught.

```
fish
```

```
##    Time Quality
## 1     0     8.5
## 2     0     8.4
## 3     3     7.9
## 4     3     8.1
## 5     6     7.8
## 6     6     7.6
## 7     9     7.3
## 8     9     7.0
## 9    12     6.8
## 10   12     6.7
```

```
plt <- ggplot(fish, aes(Time, Quality)) +
  geom_point()
plt
```




```
fit <- lm(Quality ~ Time, data=fish)
round(fit$coef, 3)
```

```
## (Intercept)        Time
##       8.460      -0.142
```

so we have

$$\text{Quality} = 8.46 - 0.142 * 4 = 7.9$$

We can also let R do the calculation for us:

```
round(predict(fit, newdata=data.frame(Time=4)), 2)
```

```
##    1
## 7.89
```

Confidence vs. Prediction Intervals

Again we want an idea of the "error" in our estimate. Previously we used confidence intervals to do this. Here we will again use confidence intervals, but in the context of regression there are two types of intervals:

Confidence Interval - used to predict the **mean** response of **many** observations with the desired x value.

Prediction Interval - used to predict the **individual** response of **one** observation with the desired x value.

Warning

The terminology is a little confusing here, with the same term meaning different things: Both confidence intervals and prediction intervals as found by the regression command are confidence intervals in the sense discussed before, and both are used for prediction!

They differ in what they are trying to predict, on the one hand an **individual response** (PI), on the other hand the **mean of many responses** (CI).

Example Fish

Use this data set to find a 95% interval estimate for the quality of a fish that was put into storage after 4 hours.

We are talking about **one** fish, so we want a **prediction** interval:

```
round(predict(fit, newdata=data.frame(Time=4),
        interval="prediction"), 2)
```

```
##    fit lwr  upr
## 1 7.89 7.6 8.19
```

so a 95% prediction interval for the rating of fish after 4 hours is (7.60, 8.19)

Example Again consider the Quality of Fish data. Use this data set to find a 90% interval estimate for the mean quality of fish that were put into storage after 4 hours.

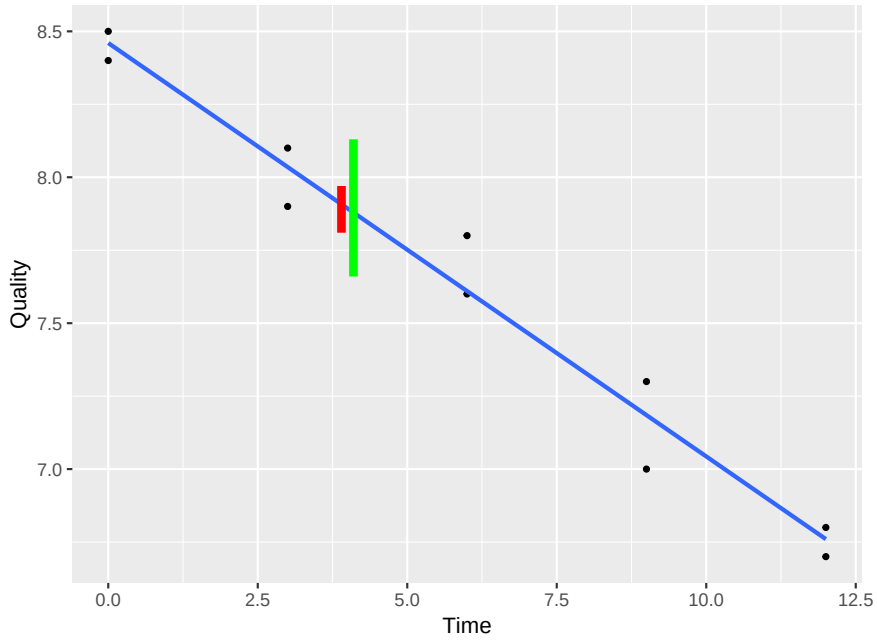
Now we are interested in the **mean** rating of many fish, so we want a **confidence** interval. Also we want a 90% interval instead of 95%:

```
round(predict(fit, newdata=data.frame(Time=4),
        interval="confidence",
        level = 0.90), 2)
```

```
##    fit  lwr  upr
## 1 7.89 7.81 7.97
```

so a 90% confidence interval for the mean rating of fish after 4 hours is (7.81, 7.97).

The two 90% intervals are shown in the next graph, the prediction interval in green and the confidence interval in red:



Notice that the prediction intervals are always wider than the confidence intervals.

The predict command can also be used to find a number of fits and intervals simultaneously:

```
round(predict(fit, newdata=data.frame(Time=1:10),
              interval="prediction",
              level = 0.99), 2)
```

```
##     fit  lwr  upr
## 1  8.32 7.87 8.77
## 2  8.18 7.74 8.62
## 3  8.04 7.60 8.47
## 4  7.89 7.46 8.32
## 5  7.75 7.33 8.18
## 6  7.61 7.19 8.03
## 7  7.47 7.04 7.89
## 8  7.33 6.90 7.76
## 9  7.19 6.75 7.62
## 10 7.04 6.60 7.48
```

If the newdata argument is left off the prediction is done for the data itself:

```
round(predict(fit), 2)
```

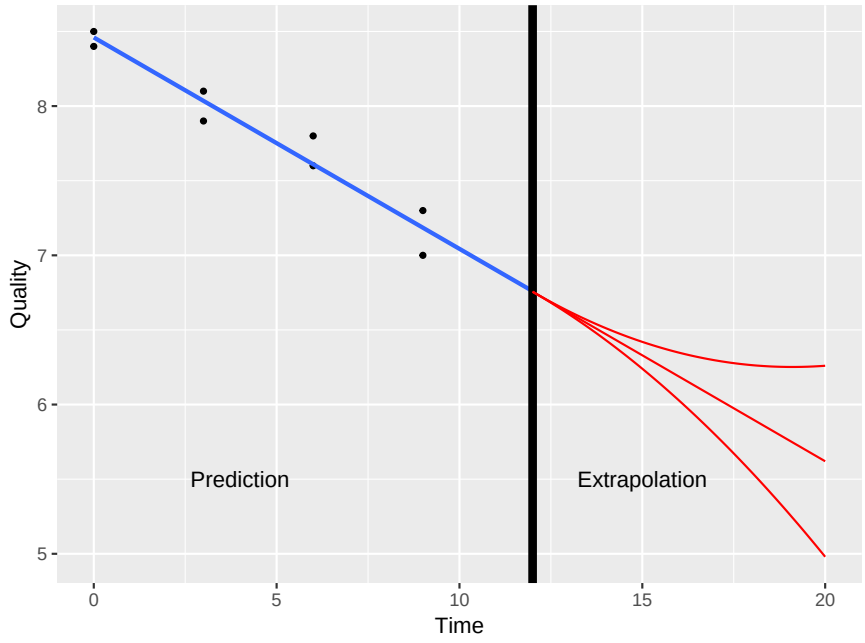
```
##    1    2    3    4    5    6    7    8    9   10
## 8.46 8.46 8.04 8.04 7.61 7.61 7.19 7.19 6.76 6.76
```

Prediction vs. Extrapolation

There is a fundamental difference between predicting the response for an x value **within** the range of observed x values (=Prediction) and for an x value **outside** the observed x values (=Extrapolation). The problem here is that the model used for prediction is only known to be good for the range of x values that were used to find it. Whether or not it is the same outside these values is generally impossible to tell.

Note Another word for prediction is **interpolation**

Example: Quality of Fish data



Nonlinear Models

Transformations and Polynomial Models

Example: Fabric Wear

Results from an experiment designed to determine how much the speed of a washing machine effects the wear on a new fabric. The machine was run at 5 different speeds (measured in rpm) and with six pieces of fabric each.

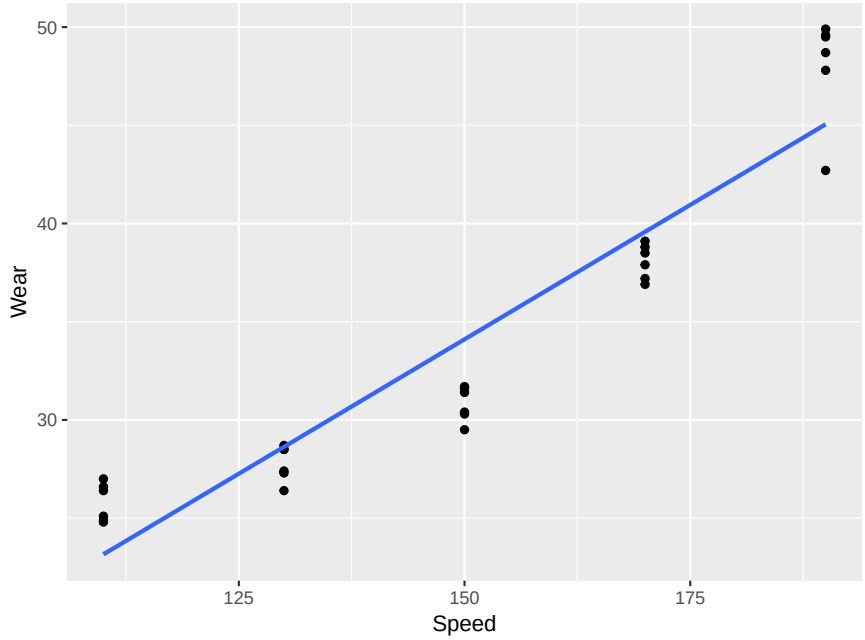
```
head(fabricwear)
```

```
##   Speed Wear
## 1   110 24.9
## 2   110 24.8
## 3   110 25.1
## 4   110 26.4
## 5   110 27.0
```

```
## 6   110 26.6
```

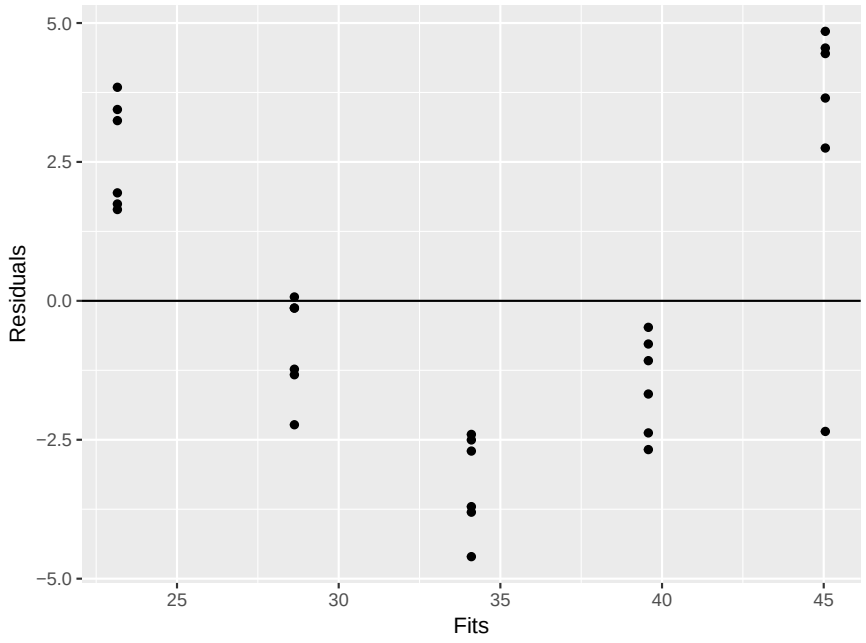
The scatterplot of wear by speed shows a strong but non-linear relationship:

```
ggplot(data=fabricwear, aes(Speed, Wear)) +
  geom_point()+
  geom_smooth(method = "lm", se=FALSE)
```



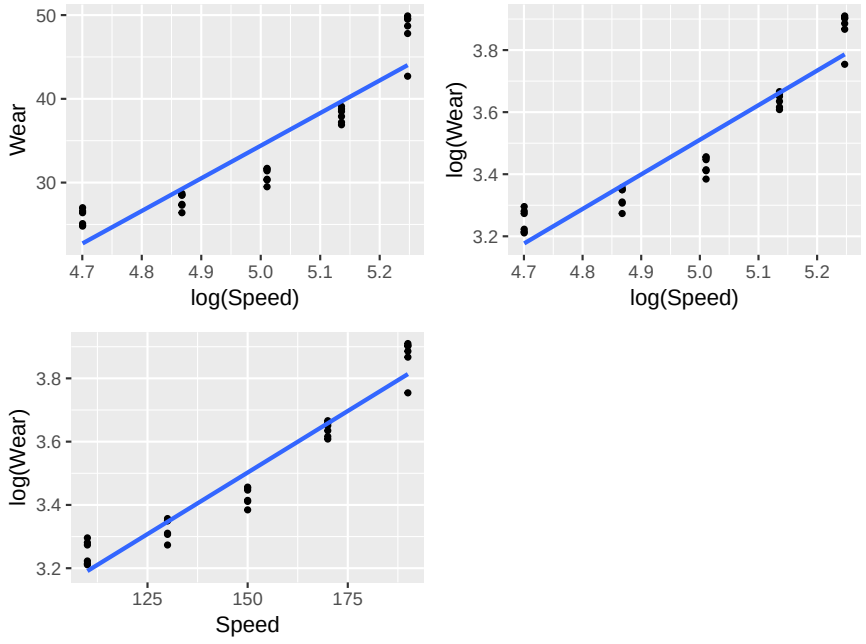
How strong is a difficult question, because Pearson's correlation coefficient won't work here. If we tried lm we would see in the residual vs fits plot that there is a problem with the assumption of a linear model:

```
fit <- lm(Wear~Speed, data=fabricwear)
df <- data.frame(Fits=fitted(fit),
                 Residuals=resid(fit))
ggplot(df, aes(Fits, Residuals)) +
  geom_point() +
  geom_hline(yintercept = 0)
```



So the question is: how do we fit models other than straight lines?

There are two basic things we can try. The first is something we have already done, namely the **log transformations**



unfortunately non of these looks very good

Some of these have names:

- log(y) vs. x is called an **exponential model**
- log(y) vs. log(x) is called a **power model**

The other solution to our problem is to fit a **polynomial model**:

Linear $y = \beta_0 + \beta_1 x$

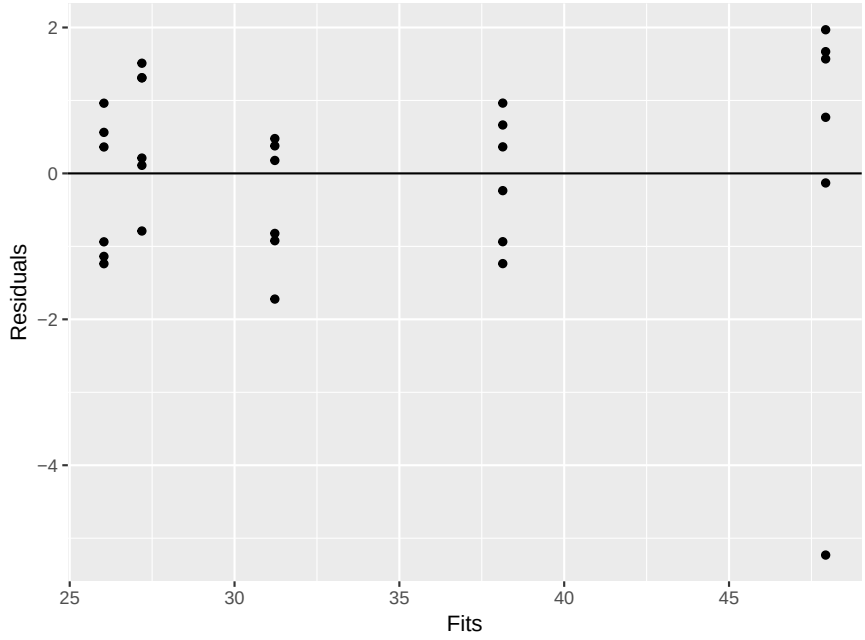
Quadratic $y = \beta_0 + \beta_1 x + \beta_2 x^2$

Cubic $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$

and so on

How do we fit such a model? We simply calculate the powers and use them in lm:

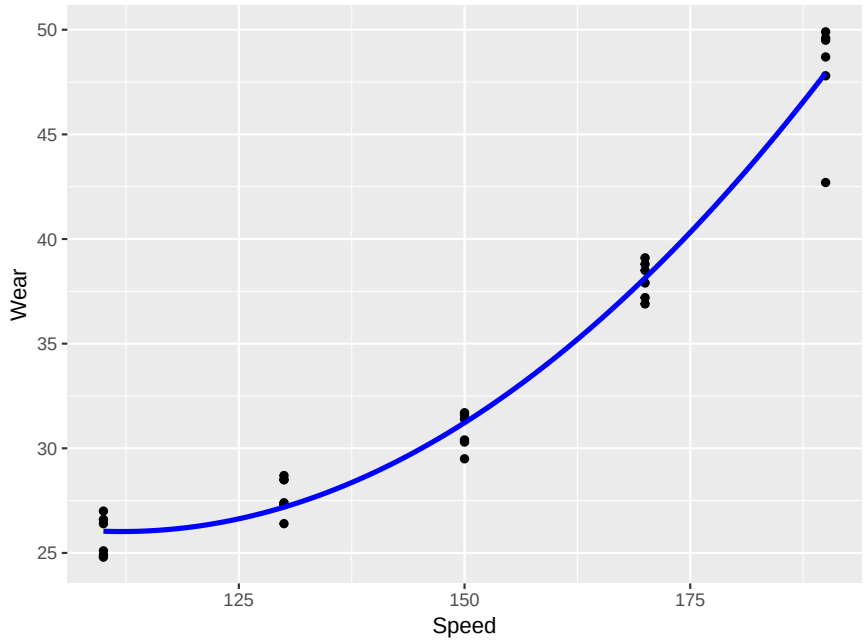
```
Speed2 <- Speed^2
quad.fit <- lm(Wear~Speed+Speed2, data=fabricwear)
quad.df <- data.frame(Fits=fitted(quad.fit),
                      Residuals=resid(quad.fit))
ggplot(quad.df, aes(Fits, Residuals)) +
  geom_point() +
  geom_hline(yintercept = 0)
```



What does such a curve look like?

```
x <- seq(min(fabricwear$Speed),
         max(fabricwear$Speed),
         length=250)
y <- coef(quad.fit)[1] +
     coef(quad.fit)[2]*x +
     coef(quad.fit)[3]*x^2
df.tmp <- data.frame(x=x, y=y)
ggplot(data=fabricwear, aes(Speed, Wear)) +
  geom_point() +
  geom_line(data=df.tmp, aes(x, y),
```

```
            inherit.aes = FALSE,
            color="blue", size=1.2)
```



There is however something not so good about this solution: our x values are of the size 100, their square is of order 10000. Using variables with very different sizes can lead to troubles in a regression. Also we have

```
cor(fabricwear$Speed, Speed2)
```

```
## [1] 0.9969033
```

and again using highly correlated predictors is an issue. One way around these problems is to use the *poly* command:

```
poly.fit <- lm(Wear~poly(Speed, 2), data=fabricwear)
coef(poly.fit)
```

```
##     (Intercept) poly(Speed, 2)1 poly(Speed, 2)2
##        34.10333        42.39626        13.20218
```

The poly command does two things: it scales all the variables so that their mean is 0 and their standard deviation is 1, and it changes them in such a way that they are uncorrelated. This makes the regression calculations much more stable.

If the goal is prediction, this is fine:

```
predict(quad.fit,
        newdata=data.frame(Speed=150, Speed2=150^2))
```

```
##        1
## 31.22238
```

```
predict(poly.fit, newdata=data.frame(Speed=150))
```

```
##        1
## 31.22238
```

but when it comes to getting the actual model, we would have to "reverse" the calculations done by poly, and we don't even know what those were, exactly.

There is an intermediate solution that sometimes works well: scale the x variable first:

```
mn <- mean(fabricwear$Speed)
s <- sd(fabricwear$Speed)
x <- (fabricwear$Speed-mn)/s
x2 <- x^2
round(c(mn, s, cor(x, x2)), 2)
```

```
## [1] 150.00  28.77   0.00
```

```
quad.scale.fit <- lm(fabricwear$Wear~x+x2)
coef(quad.scale.fit)
```

```
## (Intercept)           x          x2
##   31.222381    7.872787    2.980296
```

so first we also got uncorrelated predictors (that is just here, but in general the correlations will be low). Also:

$$x = \frac{\text{Speed} - 150}{28.77}$$
$$y = 31.2 + 7.87x + 2.98x^2$$
$$y = 31.2 + 7.87\frac{\text{Speed} - 150}{28.77} + 2.98\left(\frac{\text{Speed} - 150}{28.77}\right)^2 =$$
$$31.2 + 0.2735\ \text{Speed} - 41.03 +$$
$$0.0036\ \text{Speed}^2 - 1.08\ \text{Speed} + 81 =$$
$$71.17 - 0.806\ \text{Speed} + 0.0036\ \text{Speed}^2$$

and that is the same as

```
coef(quad.fit)
```

```
## (Intercept)       Speed      Speed2
## 71.19916667 -0.80669048  0.00360119
```

Prediction

Again we can use the predict command to do prediction, but there are some things we need to be careful with:

```
predict(quad.fit,
            newdata=data.frame(Speed=150,
                               Speed2=150^2))
```

```
##        1
## 31.22238
```

```
lwear <- log(Wear)
lspeed <- log(Speed)
log.fit <- lm(lwear~lspeed)
exp(predict(log.fit,
            newdata=data.frame(lspeed=log(150))))
```

```
##        1
## 33.87835
```

How about interval estimation? Let's do a simple simulation: consider the model $y = x^2$. It is both a quadratic model and linear in log-log, so we should be able to fit it either way:

```
B <- 1000
x <- 1:100/100
out <- matrix(0, 1000, 6)
lx <- log(x)
x2 <- x^2
for(i in 1:B) {
  y <- x^2 + rnorm(100,0, 0.07)
  pf <- lm(y~x+x2)
  out[i, 1:3] <- predict(pf,
        newdata=data.frame(x=0.5, x2=0.25),
        interval="confidence")
  ly <- log(y)
  lf <- lm(ly~lx)
  out[i, 4:6] <- exp(predict(lf,
        newdata=data.frame(lx=log(0.5)),
        interval="confidence"))
}
```

```
#quadratic model
sum(out[, 2]<0.5^2 & 0.5^2<out[, 3])/B
```

```
## [1] 0.949
```

```
#log transform
sum(out[, 5]<0.5^2 & 0.5^2<out[, 6])/B
```

```
## [1] 0.794
```

so this works fine for quadratic model but fails miserably for the log transform.

Why is that? Part of the problem is the error term. Note that what we really have is

$$y = x^2 + \epsilon$$

so taking logs leads to

$$\log y = \log(x^2 + \epsilon)$$

and not to what we are fitting, namely

$$\log y = \log(x^2) + \epsilon$$

Finding the best model - Overfitting

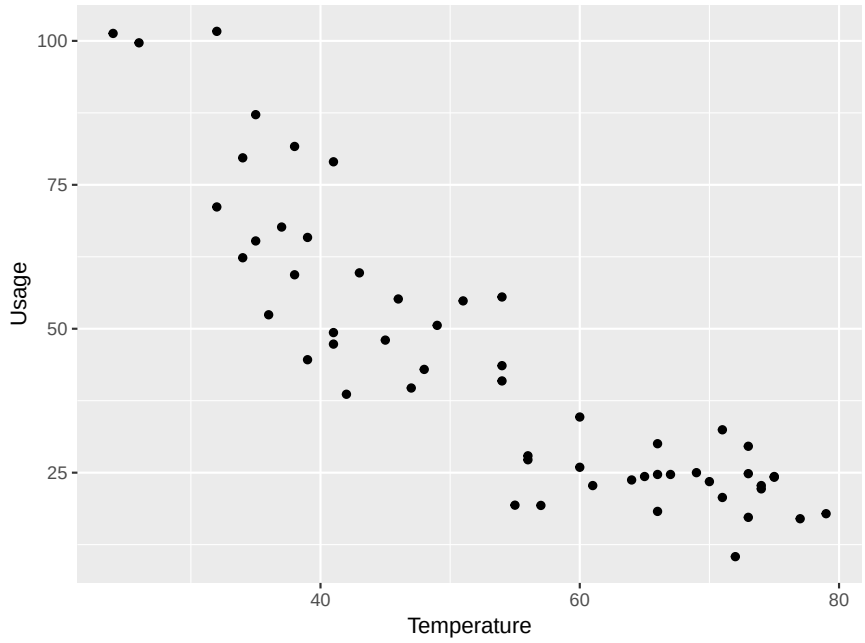
Example: Predicting the Usage of Electricity

In Westchester County, north of New York City, Consolidated Edison bills residential customers for electricity on a monthly basis. The company wants to predict residential usage, in order to plan purchases of fuel and budget revenue flow. The data includes information on usage (in kilowatt-hours per day) and average monthly temperature for 55 consecutive months for an all-electric home. Data on consumption of electricity and the temperature in Westchester County, NY.

```
kable.nice(elusage[1:10, ])
```

Month	Year	Usage	Temperature
8	1989	24.828	73
9	1989	24.688	67
10	1989	19.310	57
11	1989	59.706	43
12	1989	99.667	26
1	1990	49.333	41
2	1990	59.375	38
3	1990	55.172	46
4	1990	55.517	54
5	1990	25.938	60

```
ggplot(aes(Temperature, Usage), data=elusage) +
  geom_point()
```



Let's find the least squares model:

```
fit <- lm(Usage~Temperature, data=elusage)
round(fit$coef, 2)
```

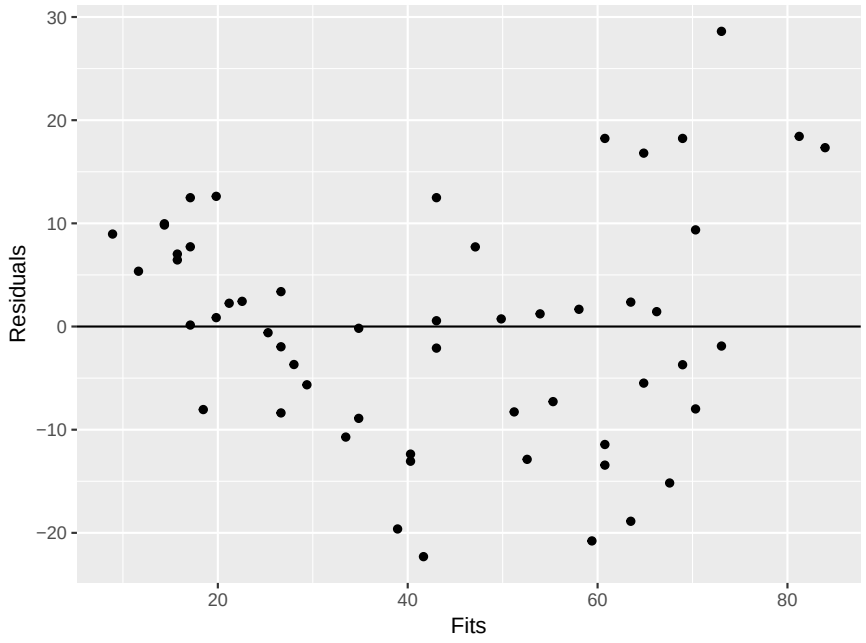
```
## (Intercept) Temperature
##      116.72       -1.36
```

gives the model as

$$\text{Usage} = 116.72 - 1.36\ \text{Temperature} + \epsilon$$

but

```
ggplot(data=data.frame(Fits=fitted(fit),
                       Residuals=residuals(fit)),
       aes(Fits, Residuals)) +
 geom_point() +
 geom_abline(slope = 0)
```



shows that this is a bad model.

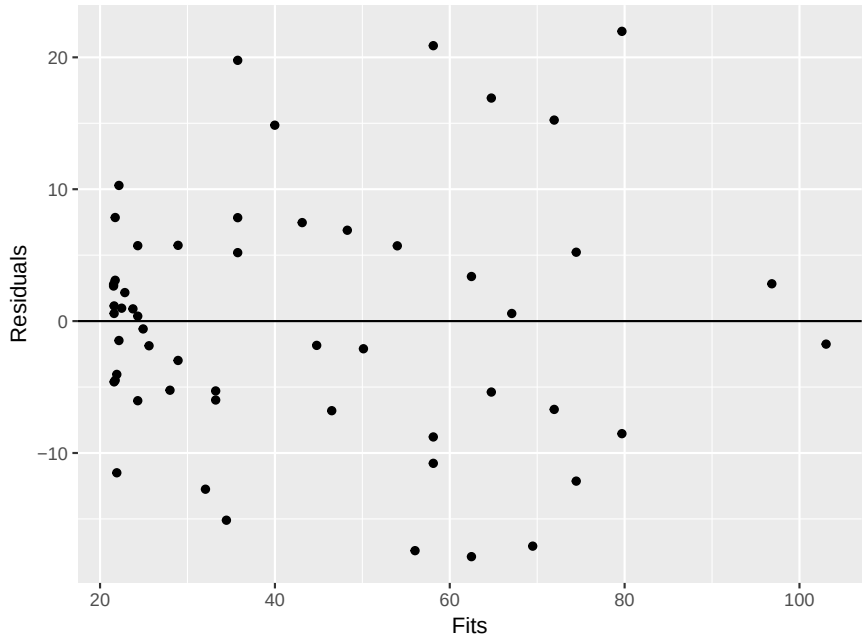
So now we try the

- quadratic model

```
quad.fit <- lm(Usage~poly(Temperature, 2),
               data=elusage)
```

the residual vs fits plot for this model is

```
ggplot(aes(Fits, Residuals),
       data=data.frame(Fits=fitted(quad.fit),
                       Residuals=residuals(quad.fit))) +
 geom_point() +
 geom_abline(slope = 0)
```

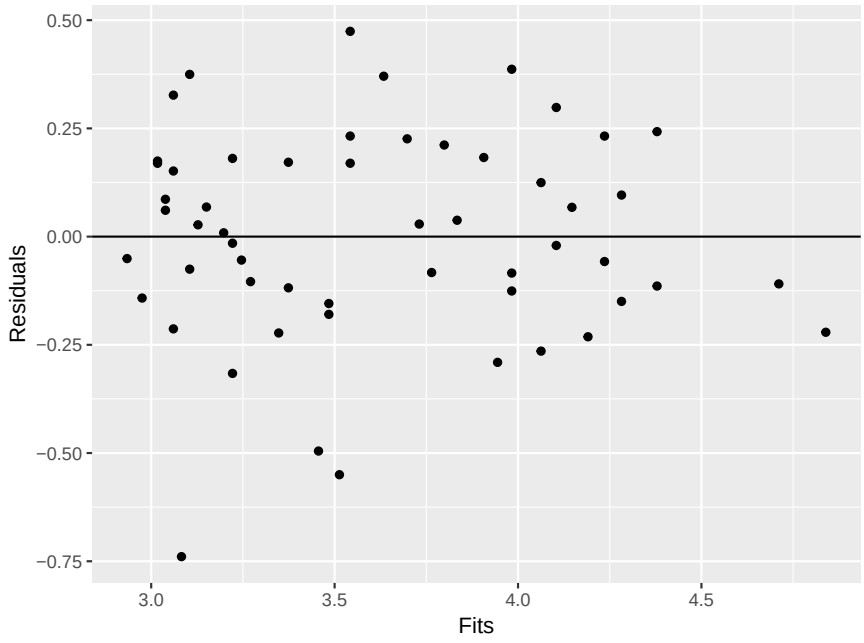


and that is much better.

- Transformations

```
log.usage <- log(Usage)
log.temp <- log(Temperature)
log.fit <- lm(log.usage~log.temp)
```

```
ggplot(aes(Fits, Residuals),
       data=data.frame(Fits=fitted(log.fit),
                       Residuals=residuals(log.fit))) +
 geom_point() +
 geom_abline(slope = 0)
```



Now we have to models with good residual vs fits plots. How do we choose among these models? A standard measure of the quality of the fit is the **Coefficient of Determination**. It is defined as

$$R^2 = \text{cor}(\text{Observed Values, Predicted Values})^2 100\%$$

the better a model is, the more correlated it's fitted values and the observed values should be, so if we have a choice of two model, the one with the higher R^2 is better.

Here we find

```
# Quadratic Model
round(100*summary(quad.fit)$r.squared, 2)
```

```
## [1] 84.69
```

```
# Log Transfrom Model
round(100*summary(log.fit)$r.squared, 2)
```

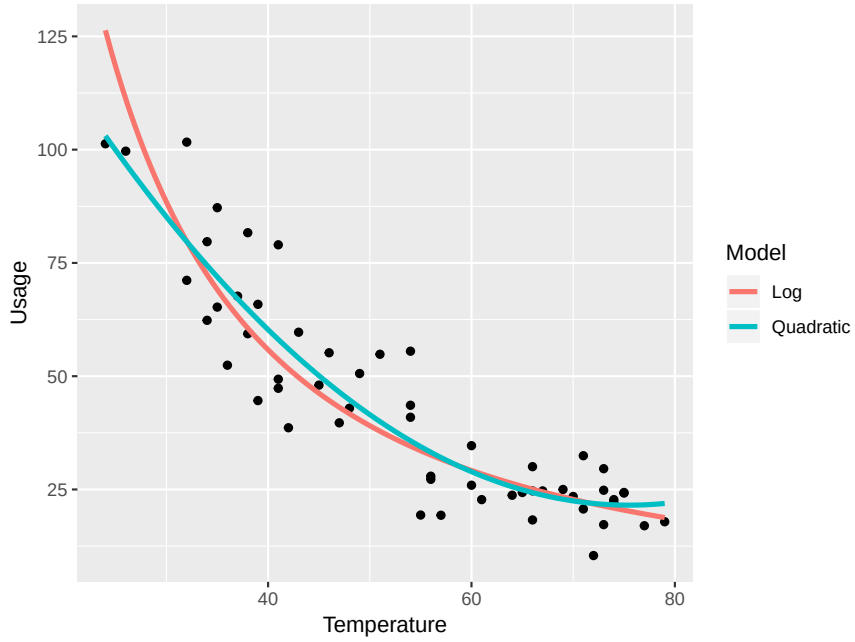
```
## [1] 81.12
```

Now the R^2 of the quadratic model is 84.69% and that of the log transform model is 81.12%, so the quadratic one is better.

Let's have a look what those models look like:

```
x <- seq(min(Temperature), max(Temperature), length=100)
y.quad <- predict(quad.fit,
                  newdata=data.frame(Temperature=x))
y.log <- exp(predict(log.fit,
```

```
                  newdata=data.frame(log.temp=log(x))))
dta <- data.frame(x=c(x, x),
                  y=c(y.quad, y.log),
                  Model=rep(c("Quadratic", "Log"),
                            each=100))
ggplot(data=elusage, aes(Temperature, Usage)) +
  geom_point() +
  geom_line(data=dta, aes(x, y, color=Model), size=1.2) +
  xlab("Temperature") +
  ylab("Usage")
```



Could we do even better? Let's check the cubic model:

```
cube.fit <- lm(Usage~poly(Temperature,3),
               data=elusage)
round(100*summary(cube.fit)$r.squared, 2)
```

```
## [1] 84.72
```

and yes, it's $R^2 = 84.72 > 84.69$!

but we need to be careful here: the quadratic model is a special case of the cubic model, and so it's R^2 can never be smaller.

The reason for this is simple: Say we find the best quadratic model, which is

$$\text{Usage} = \hat{\beta}_{02} - \hat{\beta}_{12}\ \text{T} + \hat{\beta}_{22}\ \text{T}^2$$

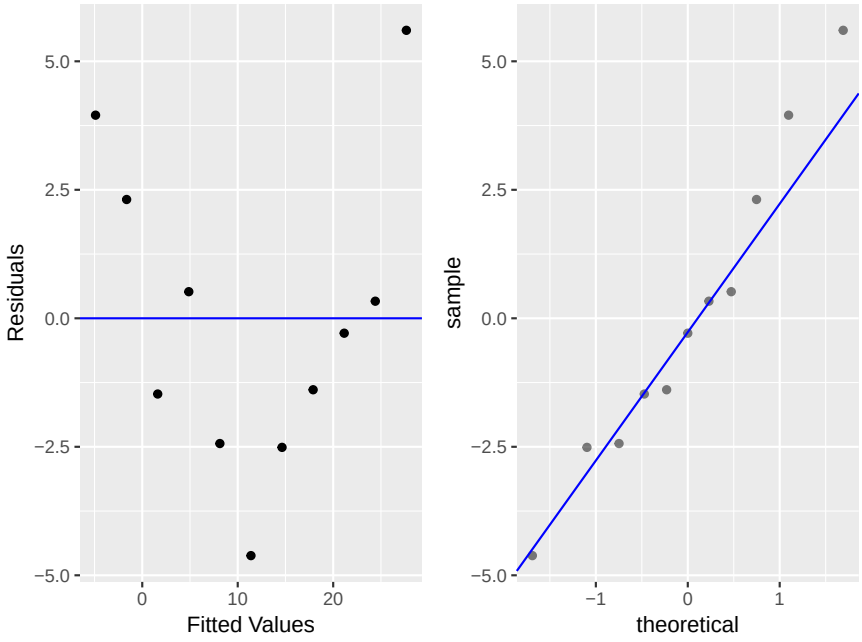
Now we add the cubic term T^3 as a predictor. One (of many) cubic models is

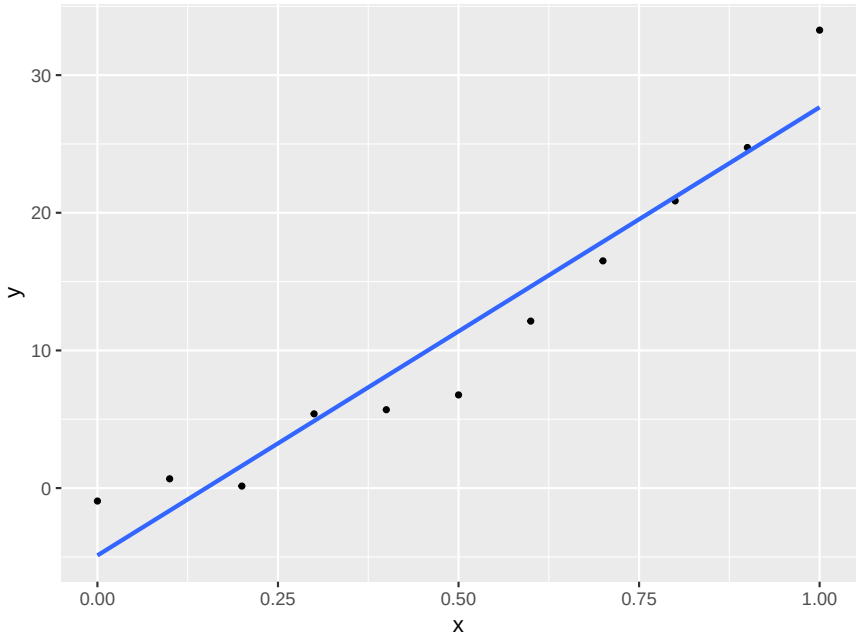
$$\text{Usage} = \hat{\beta}_{02} - \hat{\beta}_{12}\ \text{T} + \hat{\beta}_{22}\ \text{T}^2 + 0.0\ \text{T}^3$$

this is of course the same as the quadratic model above, so it has $R^2 = 84.69\%$. Only the least squares cubic model is the **best** cubic model, so it's R^2 cannot be smaller (and because of statistical fluctuation usually will be even a bit higher, even if the cubic term is not useful).

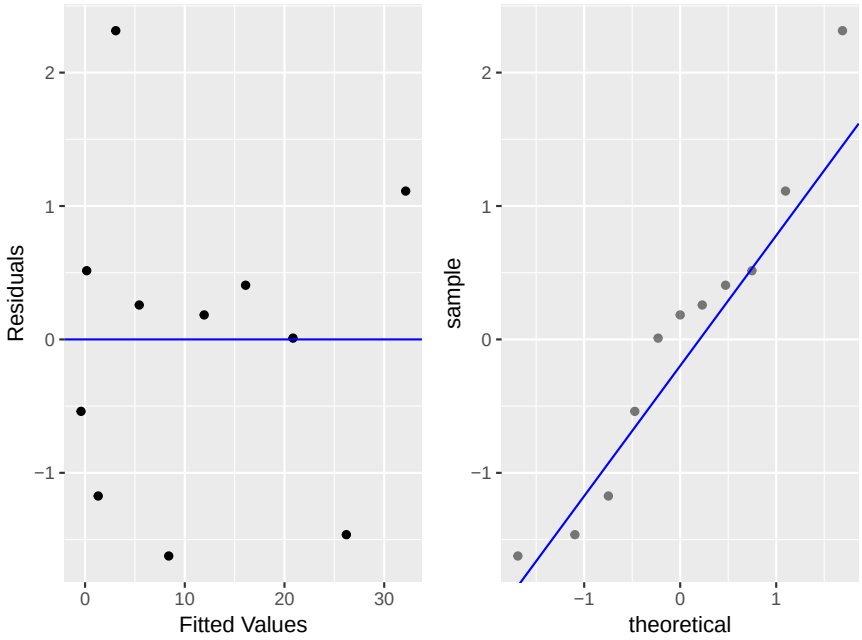
Question: which of these polynomial models should you use?

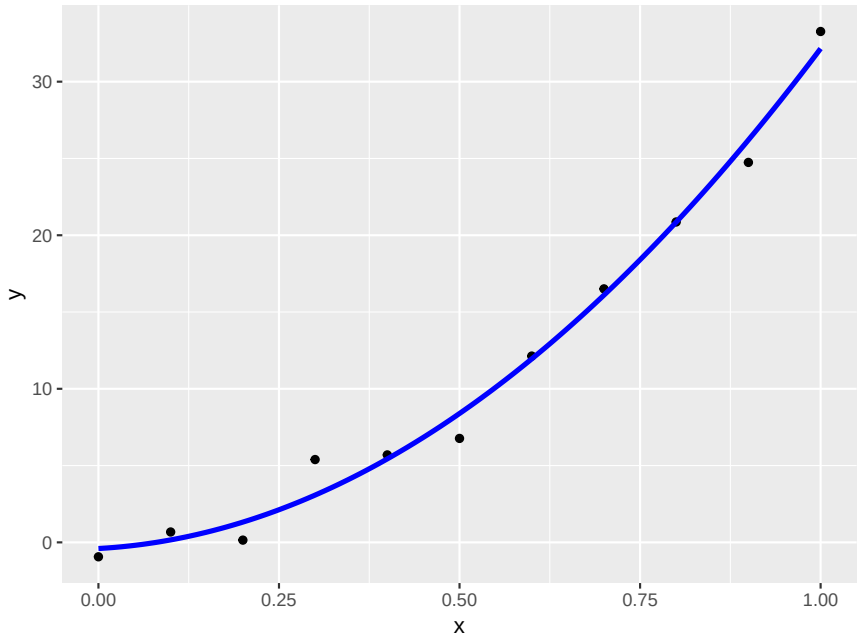
Linear Model



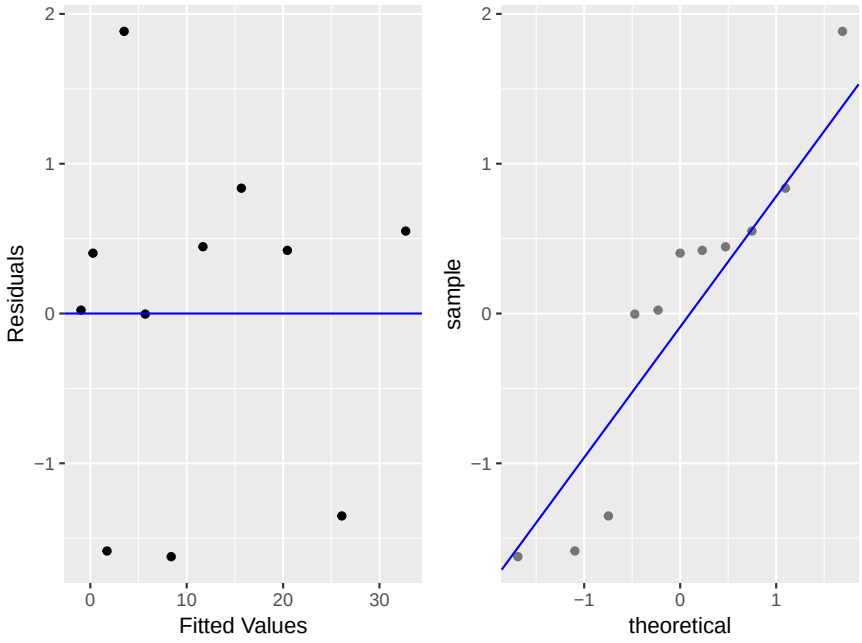


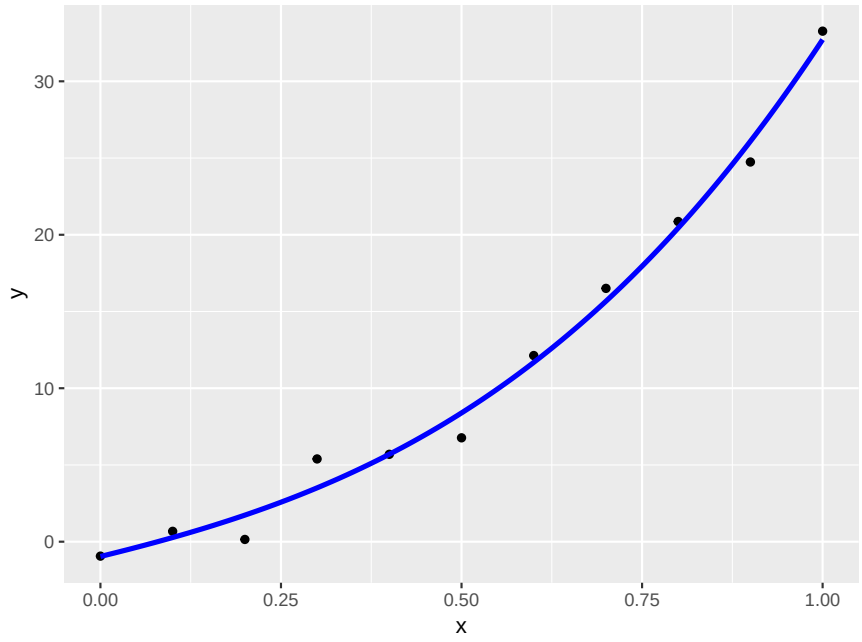
Quadratic Model



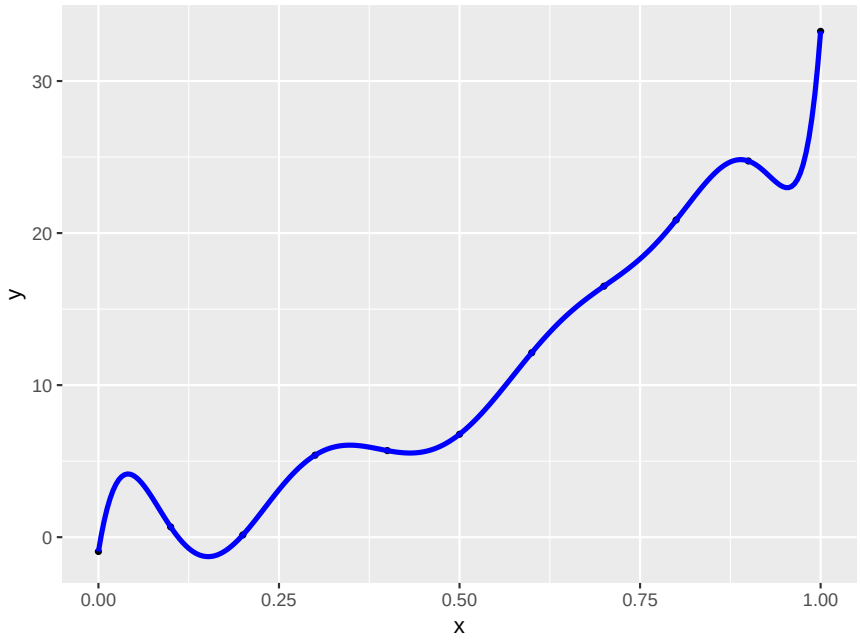


Cubic Model





Power 11 Model



and this one is perfect, it has $R^2 = 100\%$.

Actually, it is **always** possible to find a polynomial model which fits the data set perfectly, that is it has $R^2 = 100\%$! (Hint: look up Legendre polynomials)

But: we want our models to fit the relationship, not the random fluctuations in the data set.

A model should be **parsimoneous**, that is as simple as possible.

This is in agreement with one of the fundamental principles of science:

Ockham's razor, named after William of Ockham

Ockham's razor is the principle that "entities must not be multiplied beyond necessity" (entia non sunt multiplicanda praeter necessitatem). The popular interpretation of this principle is that the **simplest explanation is usually the correct one**.

For our problem this means: Use the polynomial model of **lowest degree** that can't be improved statistically significantly by adding another power.

Let's consider again the quadratic and the cubic models: the cubic model is better than the quadratic one (in terms of R^2), but is it **statistically significantly** better?

It turns out we can actually test this:

```
anova(cube.fit, quad.fit)
```

```
## Analysis of Variance Table
##
## Model 1: Usage ~ poly(Temperature, 3)
## Model 2: Usage ~ poly(Temperature, 2)
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1     51 4756.9
## 2     52 4766.4 -1   -9.4466 0.1013 0.7516
```

and so the answer is no, the cubic is not better. So we should use the quadratic one!

Warning as always a GOOD MODEL is one with a good residual vs. fits plot. It can happen that both the quadratic and the cubic are bad models and this test fails to reject the null because they are equally bad!

Note: if we have two models, one of which is a special case of the other, we say we have *nested* models.

Example: quadratic and cubic

Example: y vs x and y vs x, z

In all of those cases the model with more predictors will NEVER have a smaller R^2, so using R^2 would always lead to the model with all the terms, which may not be best.

Choosing between good Models

In choosing the best model (from our short list) proceed as follows: Model is "good" = no pattern in the Residual vs. Fits plot

1. If a linear model is good, use it, **you are done**

If the linear model is not good, proceed as follows

2. check the transformation models and see which of these (if any) are good.
3. find the best polynomial model using method described above.
4. Choose as the best of the **good** models in 2) and 3) the one which has the highest R^2.

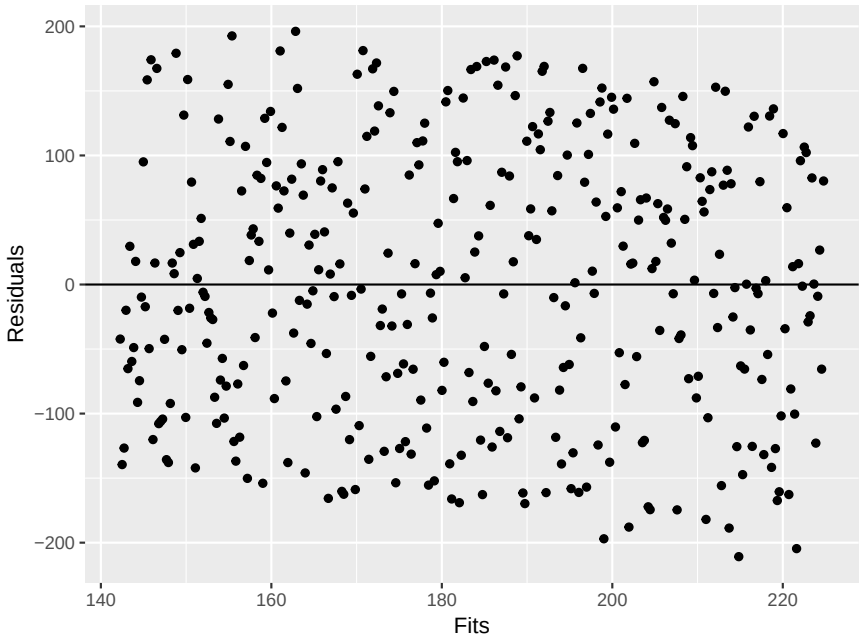
Back to Electricity usage. We have found:

- best transformation model is y vs log of x with $R^2 = 82.9\%$
- best polynomial model is the quadratic with $R^2 = 84.7\%$
- so best overall is quadratic.

An important comment

Having a high R^2 is desirable, but neither necessary for a model to be good, nor an indication that a model is good:

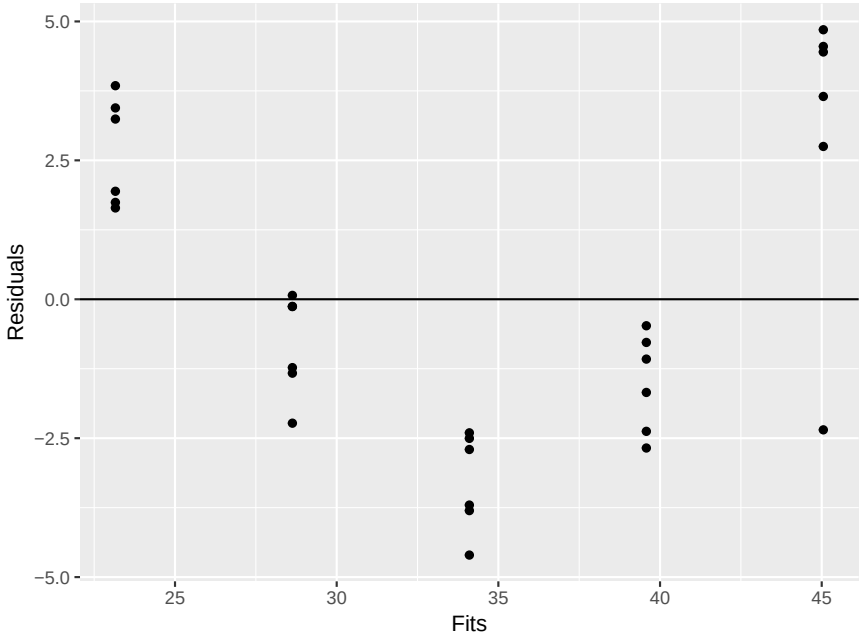
Example 1970's draft data:




```
## [1] 5.1
```

the linear model is a good one, even though it has a very low $R^2 = 5.1\%$

Example fabric wear data:




```
## [1] 88.6
```

the linear model is bad, even though it has a fairly high $R^2 = 88.6\%$.

Example: Lunatics in Massachusettes

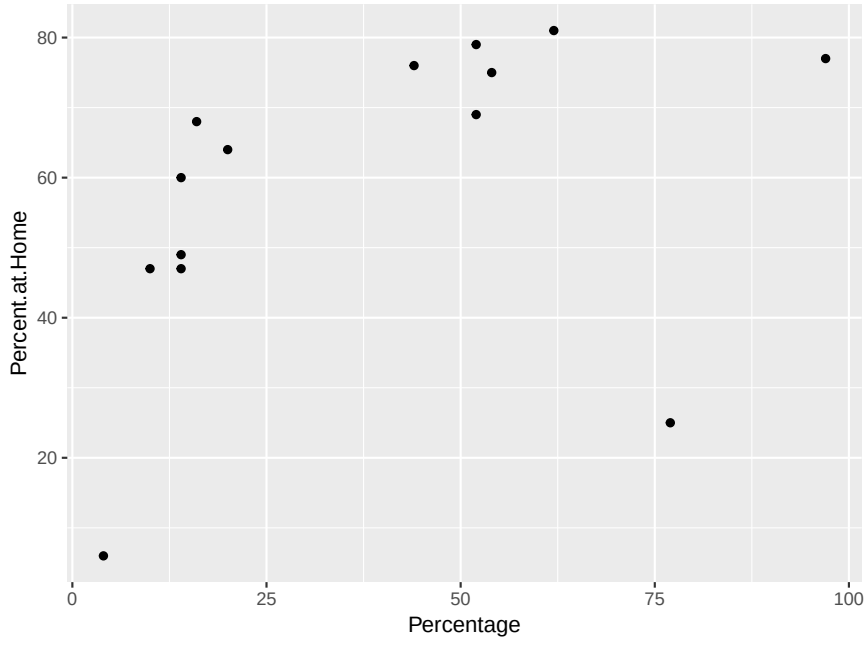
The data are from an 1854 survey conducted by the Massachusetts Commission on Lunacy (Mental disorder) under the leadership of Edward Jarvis. Dr. Jarvis was President of the American Statistical Association from 1852 to 1882.

```
lunatics
```

```
##        County Number Distance Percent.at.Home
## 1   BERKSHIRE    119       97               77
## 2    FRANKLIN     84       62               81
## 3   HAMPSHIRE     94       54               75
## 4     HAMPDEN    105       52               69
## 5   WORCESTER    351       20               64
## 6   MIDDLESEX    357       14               47
## 7       ESSEX    377       10               47
## 8     SUFFOLK    458        4                6
## 9     NORFOLK    241       14               49
## 10    BRISTOL    158       14               60
## 11   PLYMOUTH    139       16               68
## 12 BARNSTABLE     78       44               76
## 13  NANTUCKET     12       77               25
## 14      DUKES     19       52               79
```

We want to find a model to predict the percentage of lunatics kept at home by the distance to the nearest insane asylum.

```
ggplot(data=lunatics, aes(Distance, Percent.at.Home)) +
  geom_point() +
  labs(x= "Percentage")
```

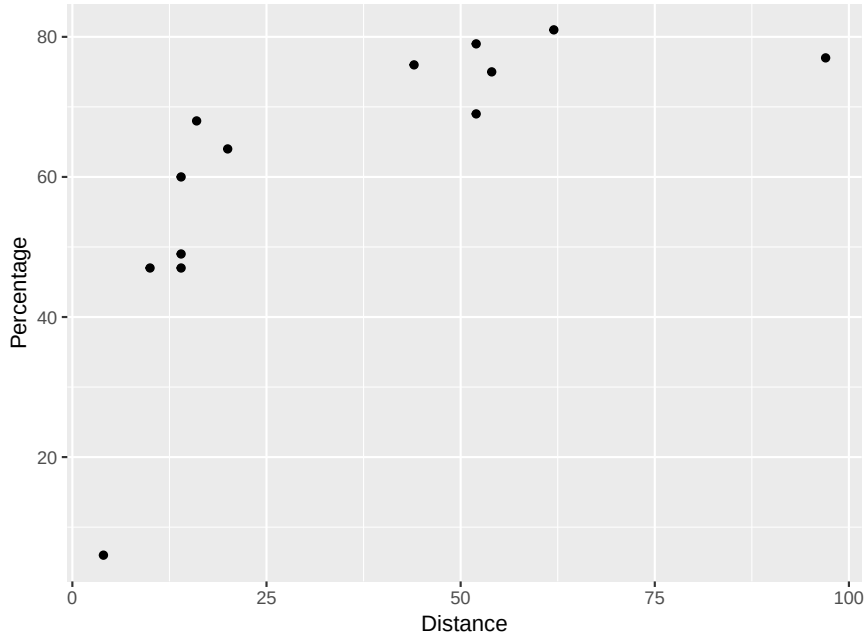


First we have a serious outlier. This turns out to be Nantucket, which is not a surprise because it is an island and people had to take a boat to get to the nearest asylum. We will

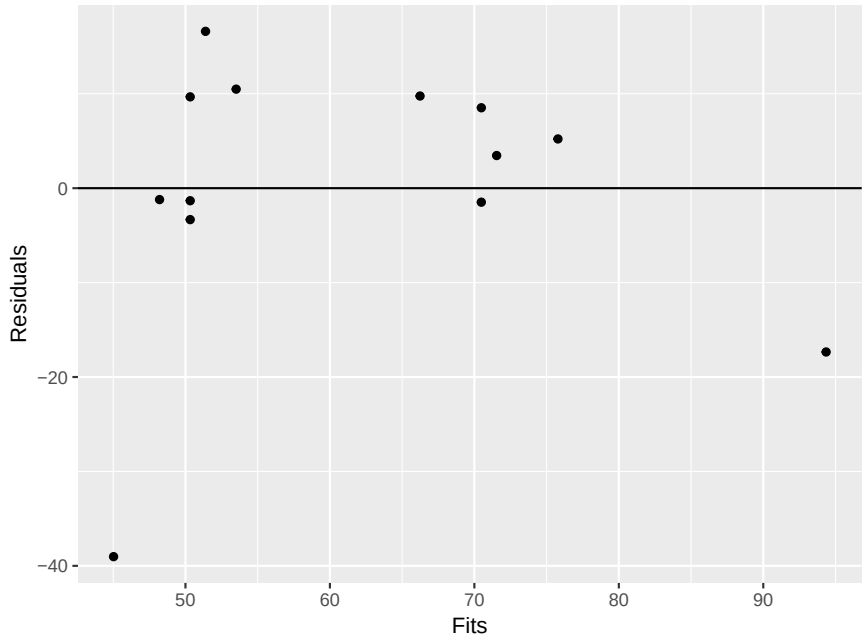
therefore take Nantucket out of the data set:

```
df <- data.frame(
  Distance=lunatics$Distance[-13],
  Percentage=lunatics$Percent.at.Home[-13])
df <- df[order(df$Distance), ]
```

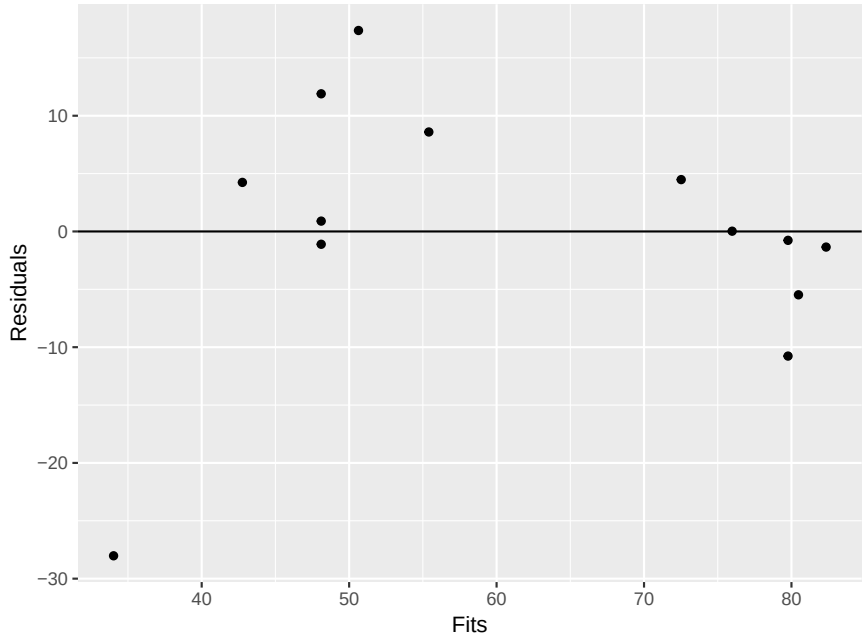
```
ggplot(data=df, aes(Distance, Percentage)) +
  geom_point()
```



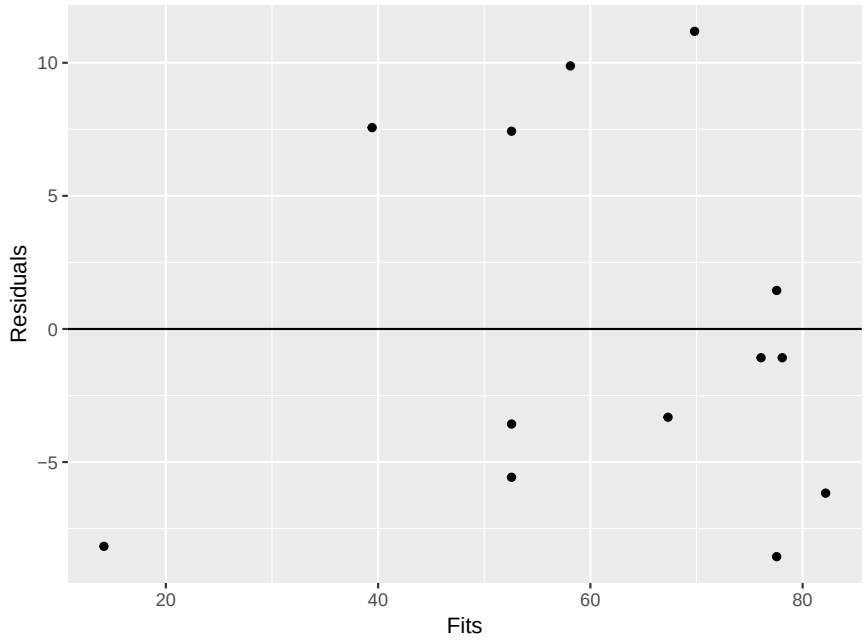

```
fits <- as.list(1:5)
for(i in 1:5) {
  fits[[i]] <- lm(Percentage~poly(Distance, i),
                  data=df)
  print(ggplot(data=data.frame(Fits=fitted(fits[[i]]),
                          Residuals=residuals(fits[[i]])),
        aes(Fits, Residuals)) +
     geom_point() +
     geom_abline(slope = 0))
  cat("Degree =", i, "\n")
  cat("R^2 =",
      round(100*summary(fits[[i]])$r.squared, 1), "\n")
  if(i>1)
    cat("p value =",
        round(anova(fits[[i]], fits[[i-1]])[[6]][2], 3))
}
```



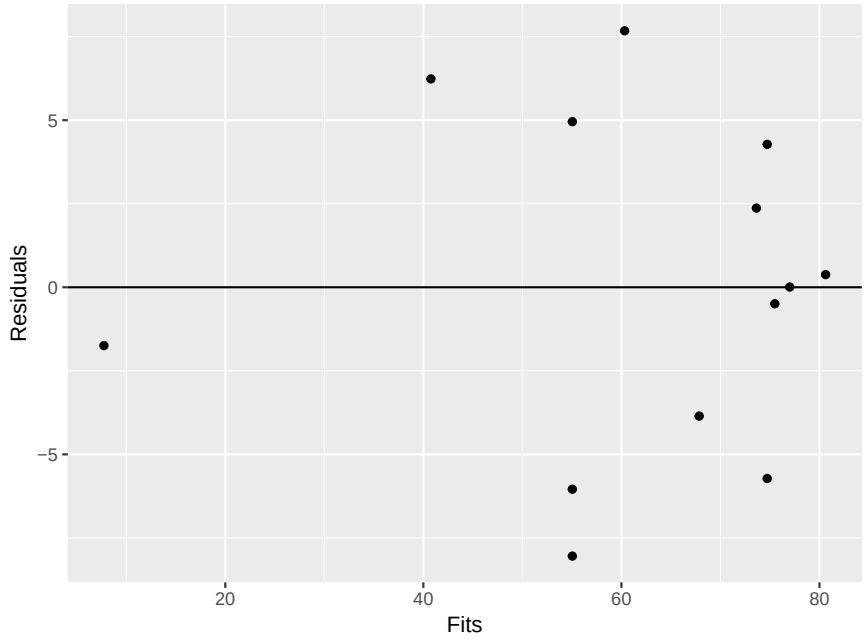

```
## Degree = 1
## R^2 = 50.3
```



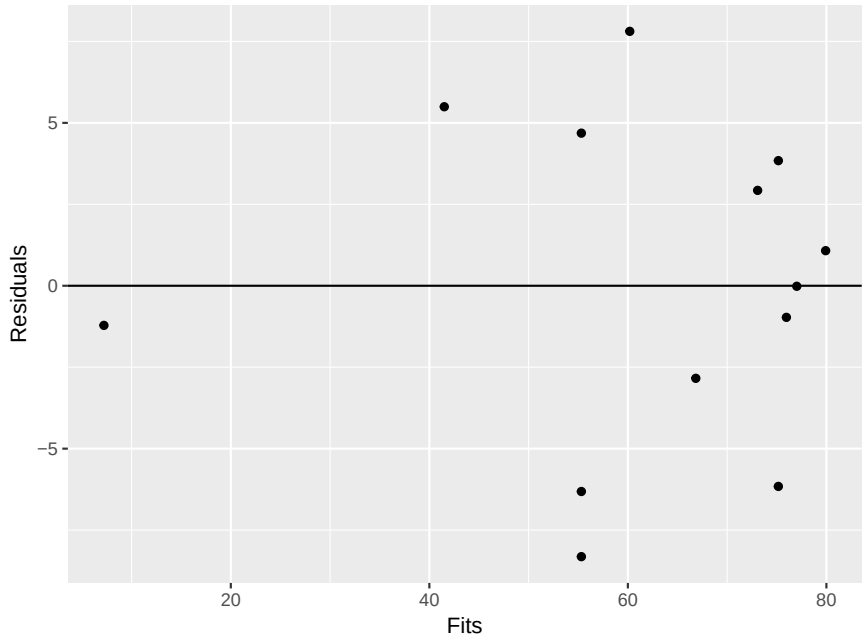

```
## Degree = 2
## R^2 = 70.7
## p value = 0.025
```




```
## Degree = 3
## R^2 = 88.7
## p value = 0.004
```




```
## Degree = 4
## R^2 = 94.1
## p value = 0.027
```

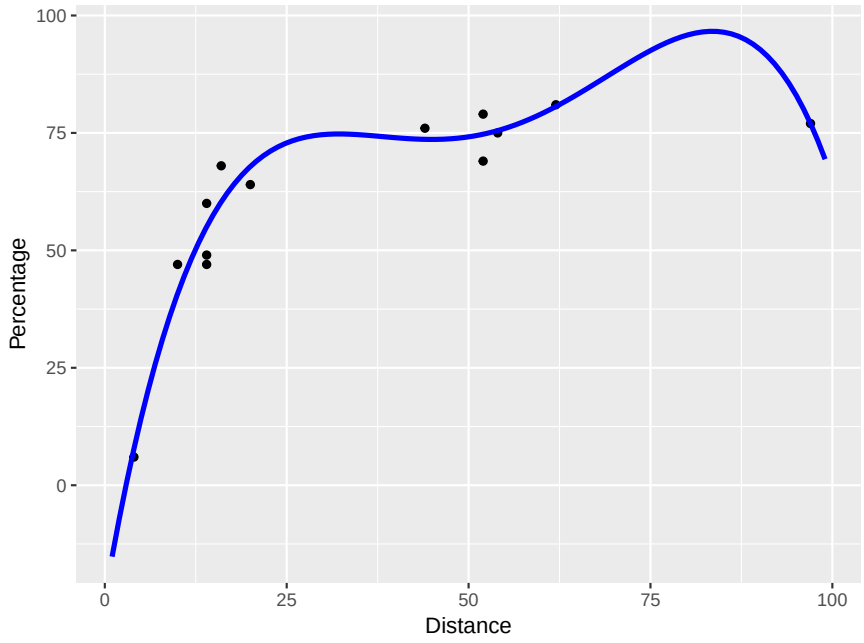



```
## Degree = 5
## R^2 = 94.2
## p value = 0.782
```

so we find that the polynomial of degree 4 is statistically significantly better than the cubic on (p=0.027) but the degree 5 is not stat. signif. better than the degree 4 one (p=0.782). So we will use degree 4.

What does this model look like?

```
fit <- lm(Percentage~poly(Distance, 4), data=df)
x <- seq(1, 99, length=100)
df1 <- data.frame(x=x,
        y=predict(fit, newdata=data.frame(Distance=x)))
ggplot(data=df, aes(Distance, Percentage)) +
  geom_point() +
  geom_line(data=df1, aes(x, y),
            color="blue", size=1.2)
```

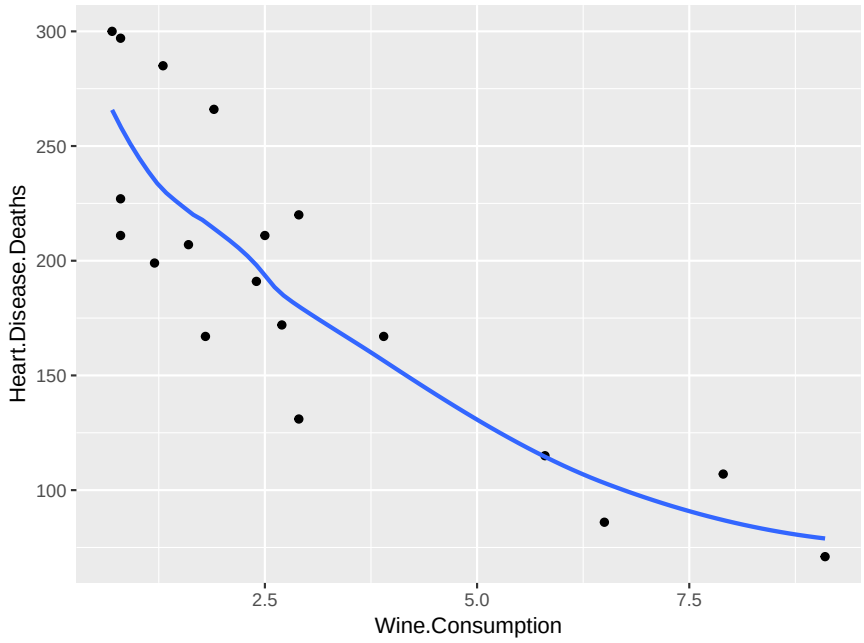


and this is not all that good either, the curve on the right of x=75 is basically determined by one observation and the bump at x=85 is an artifact of a power 4 model.

Nonparametric Regression, Smoothing

Consider the following graph:

```
ggplot(wine,
       aes(Wine.Consumption, Heart.Disease.Deaths)) +
  geom_point() +
  geom_smooth(se=FALSE)
```



Notice that this is the default of geom_smoooth, if you want the least squares line you need to use the argument method="lm".

What is this curve?

A completely different approach to fitting non-linear problems is via local and non-parametric regression. Say we have a model of the form $y = f(x)$ for some function f, and we want to estimate $y_0 = f(x_0)$. In this type of fitting procedure y_0 is estimated using only x values close to x_0, or at least the contribution of x values close to x_0 is weighted more heavily than others.

One popular method for smoothing is the function *loess*. It works as follows:

1) Find the k nearest neighbors of x_0, which constitute a neighborhood $N(x_0)$. The number of neighbors k is specified as a percentage of the total number of points in the data set. This percentage is called the span and is a tuning parameter of the method.
2) Calculate the largest distance $D(x_0)$ between x_0 and another point in the neighborhood.
3) Assign weights to each point in $N(x_0)$ using the tri-cube weight function:

$$W(\frac{x_0 - x_1}{\Delta(x_0)})$$

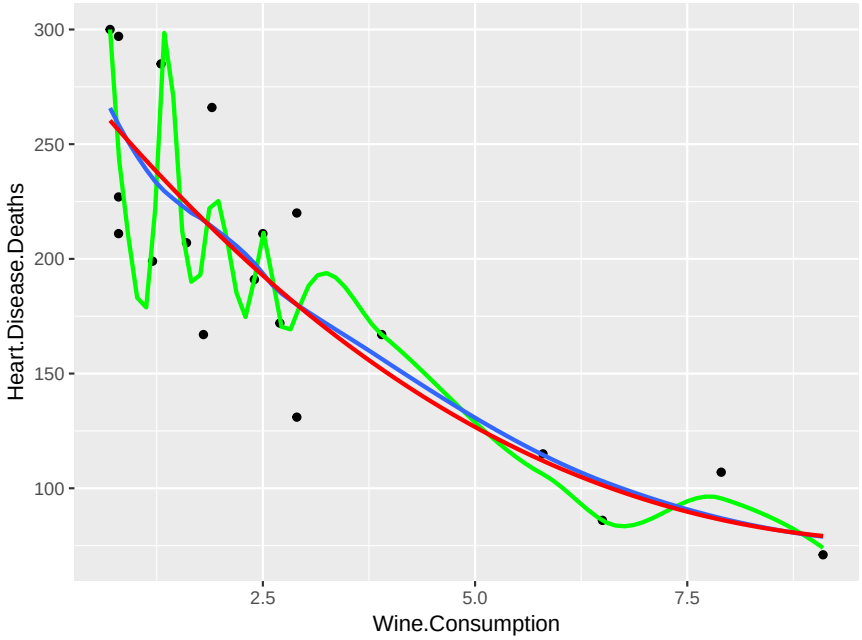
where

$$W(u) = (1 - u^3)^3 I_{[0,1]}(u)$$

4) Calculate the weighted least squares fit of x_0 on the neighborhood $N(x_0)$.

As we saw above, when it is supposed to be added to a ggplot it is easily done with the geom_smooth() command. The span can be changed with

```
ggplot(wine,
       aes(Wine.Consumption, Heart.Disease.Deaths)) +
  geom_point() +
  geom_smooth(se=FALSE, span=0.3, color="green") +
  geom_smooth(se=FALSE) +
  geom_smooth(se=FALSE, span=1.3, color="red")
```



so a larger value of span leads to a smoother curve. The default is 0.75.

To get actual predictions do this:

```
fit <- loess(Heart.Disease.Deaths~Wine.Consumption,
          data=wine)
predict(fit, data.frame(Wine.Consumption=5))
```

```
##        1
## 130.7324
```

There are a number of nonparametric regression methods implemented in R:

- **ksmooth** finds the *Nadaraya-Watson* kernel regression estimate, which is of the form

$$\hat{y}_i = \frac{\sum_{j=1}^{n} y_i K(\frac{x_i - x_j}{h})}{\sum_{j=1}^{n} K(\frac{x_i - x_j}{h})}$$

here K is the kernel function, usually one of the following:

- Normal density: $K(x) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}x^2)$
- Tukey's Biweight: $K(x) = \frac{15}{16}(1-x^2)^2 I_{[0,1]}(x)$
- Epanechnikov: $K(x) = \frac{3}{4}(1-x^2) I_{[0,1]}(x)$

and h is the equivalent to *span* in loess, a tuning parameter.

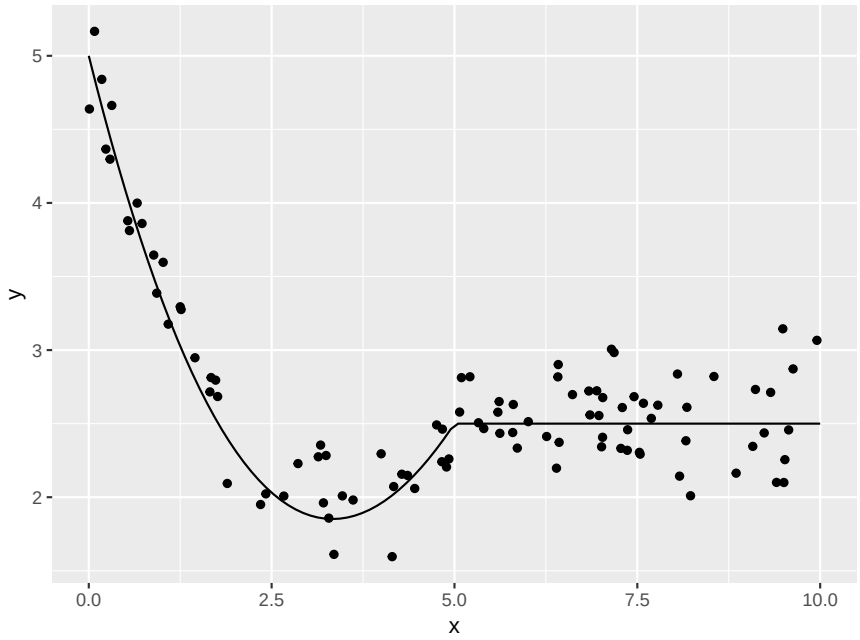
- **smooth.spline** fits a *cubic smoothing spline*.

Splines are smooth piecewise polynomial functions often used in numerical analysis. Cubic splines specifically use polynomials up to degree 3. The cubic functions change at points called *knots* and are chosen such that the whole function is continuous.

Let's consider the following artificial example: we generate some data, using the model

$$y = 5 - 2x + 0.35x^2 - 0.01x^3,\ x < 5$$
$$y = 2.5,\ x > 5$$

```
x <- sort(runif(100, 0, 10))
df <- data.frame(x=x,
     y=ifelse(x<5, 5-2*x+0.35*x^2-0.01*x^3, 2.5)+
     rnorm(100, 0, 0.25))
plt <- ggplot(df, aes(x, y)) + geom_point()
newx <- seq(0, 10, length=100)
df.true <- data.frame(x=newx,
     y=ifelse(newx<5,
              5-2*newx+0.35*newx^2-0.01*newx^3, 2.5))
plt + geom_line(data=df.true,  aes(x, y))
```



we wish to fit cubic splines to the data set. That is we fit a model of the form

$$y = \alpha_{0j} + \alpha_{1j}x + \alpha_{2j}x^2 + \alpha_{3j}x^3$$

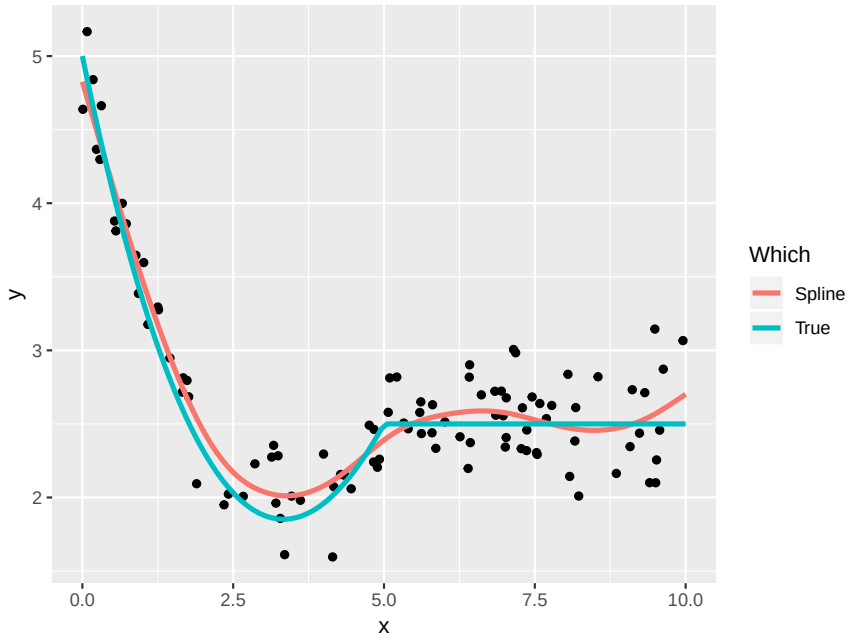
if $x_{j-1} < x < x_j$ (with $x_0 = -\infty$ and $x_{k+1} = \infty$). Here the x_j are the *knots*. Sometimes these are also estimated from the data, but for now we keep it simple and assume we know $k = 1$ and $x_1 = 5$.

We want to estimate the α_{ij} via least squares but with the condition that the resulting function be continuous, which results in the condition

$$\alpha_{0j} + \alpha_{1j}x_j + \alpha_{2j}x_j^2 + \alpha_{3j}x_j^3 = \alpha_{0j+1} + \alpha_{1j+1}x_j + \alpha_{2j+1}x_j^2 + \alpha_{3j+1}x_j^3$$

In general such *conditional optimization* problems can be solved with methods such as *Lagrange Multipliers*, but we don't need to worry about that, the routine *smooth.spline* takes care of it for us:

```
fit <- smooth.spline(df$x, df$y, spar = 0.8)
df1 <- data.frame(predict(fit,
        data.frame(x = seq(0, 10, length = 100))))
colnames(df1)[2] <- "y"
df1 <- rbind(df1, df.true)
df1$Which <- rep(c("Spline", "True"), each=100)
plt + geom_line(data=df1, aes(x, y, color=Which),
                size=1.2)
```



There is one drawback of this solution: it does not allow us to specify the exact location of the knot. Here is how to do this: first we introduce a new variable: $z = x - 5$ if $x > 5$ and 0 otherwise. Next we use *lm* to fit the model

$$y = \gamma_0 + \gamma_1 x + \gamma_2 x^2 + \gamma_3 x^3 + \gamma_4 z + \gamma_5 z^2 + \gamma_6 z^3$$

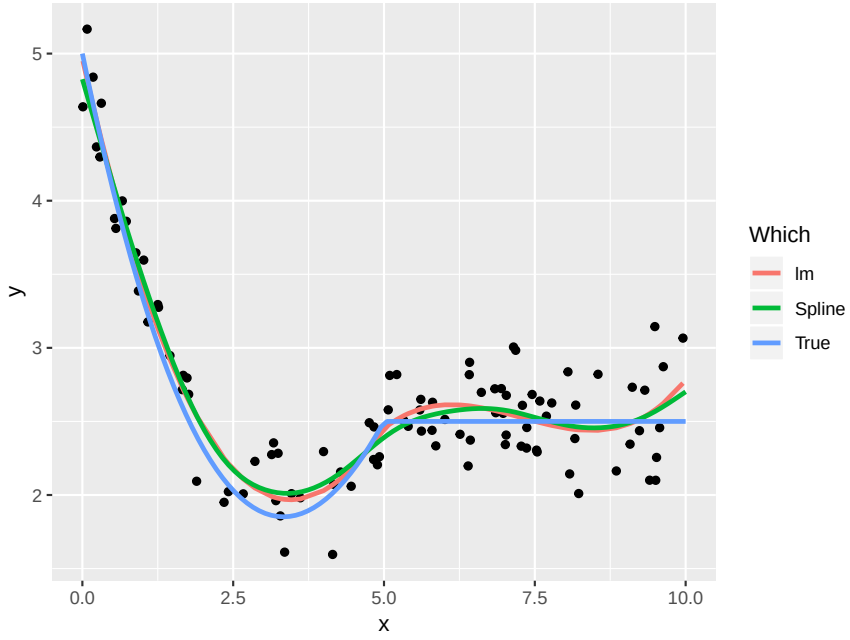
it is easy to see how to recover the $\alpha's$ from this fit.

```
x <- df$x
z <- ifelse(x < 5, 0, x - 5)
x2 <- x^2
```

```
x3 <- x^3
z2 <- z^2
z3 <- z^3
fit <- lm(df$y ~ x + x2 + x3 + z + z2 + z3)
g <- coef(fit)
a <- g[1:4]
b <- c(g[1] - g[5] * 5 + g[6] * 5^2 - g[7] * 5^3, g[2] +
            g[5] - 2 * g[6] * 5 + 3 * g[7] * 5^2, g[3] +
         g[6] - 3 * g[7] * 5, g[4] + g[7])
print(rbind(a, b))
```

```
##   (Intercept)         x         x2           x3
## a    4.966620 -1.846639  0.3156114 -0.009369247
## b   -7.468325  4.345847 -0.6125869  0.028050529
```

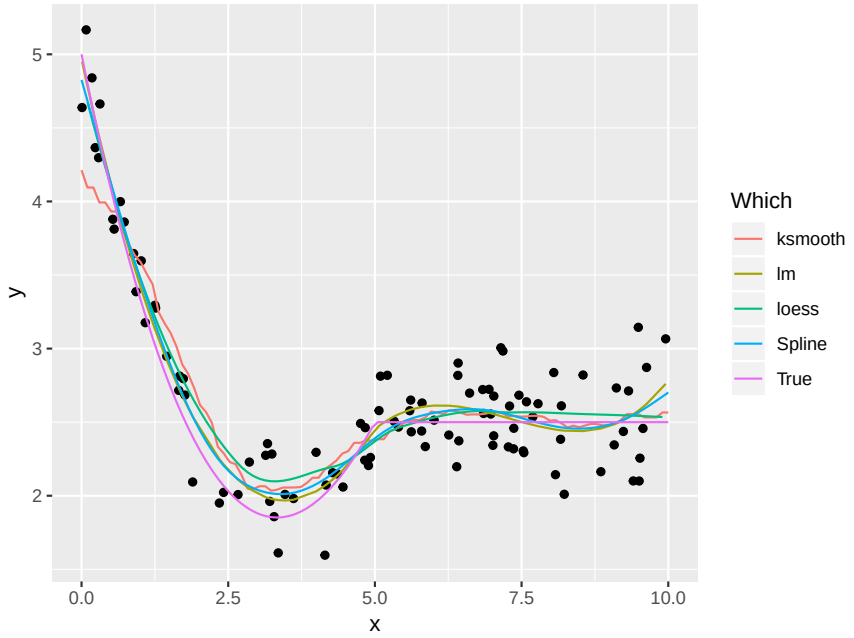
```
y <- rep(0, 100)
y[x <= 5] <- a[1] + a[2] * x[x <= 5] +
  a[3] * x[x <= 5]^2 + a[4] * x[x <= 5]^3
y[x > 5] <- b[1] + b[2] * x[x > 5] + b[3] *
          x[x > 5]^2 + b[4] * x[x > 5]^3
df.tmp <- data.frame(x=x, y=y, Which=rep("lm", 100))
df1 <- rbind(df1, df.tmp)
plt + geom_line(data=df1, aes(x, y, color=Which),
            size=1.1)
```



Finally, let's add the loess and the ksmooth solutions as well:

```
x <- seq(0, 10, length = 100)
fit <- loess(y~x, data=df)
```

```
df.loess <- data.frame(x=x,
                  y=predict(fit, data.frame(x = x)),
                  Which=rep("loess", 100))
df.ksmooth <- data.frame(x=x,
                  y=ksmooth(df$x, df$y, bandwidth = 2)$y,
                  Which=rep("ksmooth", 100))
df1 <- rbind(df1, df.loess, df.ksmooth)
plt + geom_line(data=df1, aes(x, y, color=Which))
```



Notice the different ways theses methods are called, mostly due to history of how and by whom they were added to R.

In general the choice of method is less important than the choice of:

Bandwidth (Smoothing Parameter)

One of the most active research areas in Statistics in the last 20 years has been the search for a method to find the "optimal" bandwidth for a smooother. There are now a great number of methods to do this, unfortunately non of them is fully satisfactory. We will briefly look at one method which is one of the main contenders: *cross-validation*.

In order to find an "optimal" solution we need to first decide what "optimal" means. Here we will consider the MISE (mean integrated square error):

$$\text{MISE } = E\left[\right] \int ||\hat{f}(x;b) - f(x;b)||^2 \Big] \, dx$$

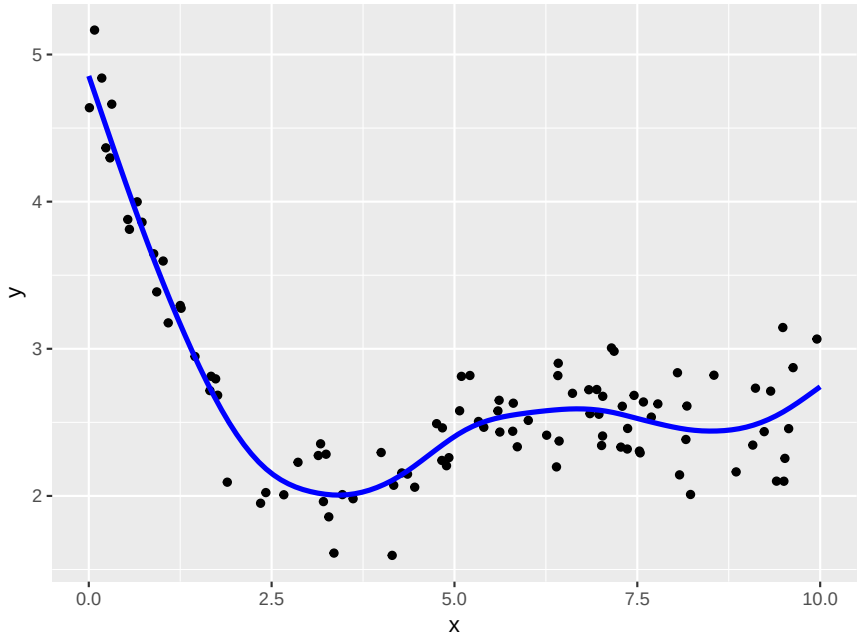
Of course the MISE depends on the unknown function f, and so we need to estimate it from the data. This is done via cross-validation, that is using the observations (x_1, y_1), ..,

(x_{i-1}, y_{i-1}), (x_{i+1}, y_{i+1}), .., (x_n, y_n) to fit the curve at x_i and get an estimate of y_i. Then you do this for all i and average over the error.

This specific description is often called *leave-one-out cross-validation*, for obvious reasons. One problem of this method is that for large data sets it is very computationally demanding.

cross-validation is already implemented in R for one of the smoothers, namely smooth.spline. If we do not specify a bandwidth (spar or df) the routine will invoke the cross-validation procedure and choose the bandwidth automatically.

```
fit <- smooth.spline(df$x, df$y)
df2 <- data.frame(predict(fit,
        data.frame(x = seq(0, 10, length = 100))))
colnames(df2)[2] <- "y"
plt + geom_line(data=df2, aes(x, y),
                color="blue", size=1.2)
```



Example: Lunatics

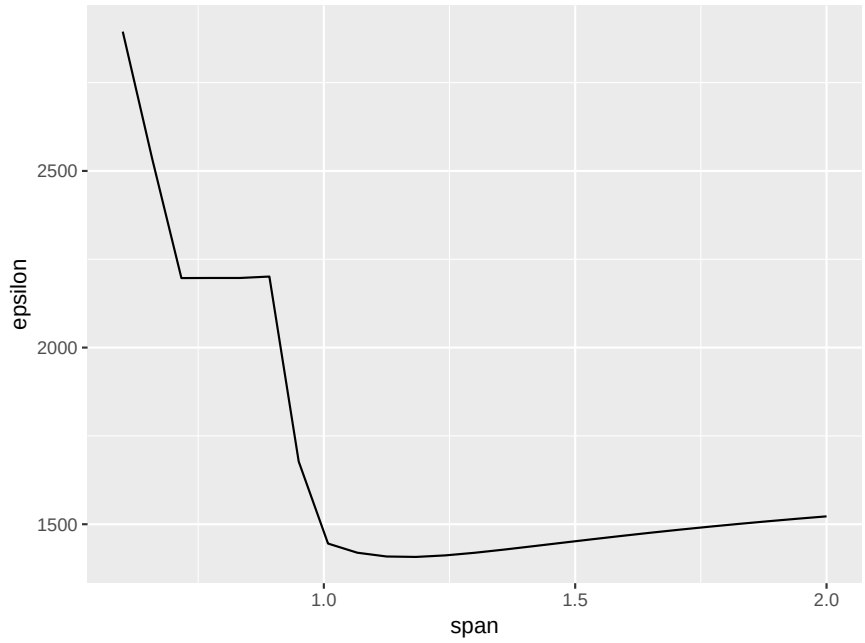
Let's implement leave-one-out cross-validation for the loess method and apply it to the lunatics data:

```
cr <- function(df, span) {
  n <- dim(df)[1]
  x <- df[[1]]
  y <- df[[2]]
  eps <- rep(0, n)
  for(i in 1:n) {
    fit <- loess(y[-i]~x[-i], span=span, surface="direct")
    yhat <- predict(fit, x[i])
```

```
    eps[i] <- (y[i]-yhat)^2
  }
  mean(eps)
}
```

```
span <- seq(0.6, 2, length=25)
eps <- span
for(i in 1:25) {
  eps[i] <- cr(lunatics[, 3:4], span[i])
}
```

```
ggplot(data=data.frame(span=span, epsilon=eps),
       aes(span, epsilon)) +
  geom_line()
```

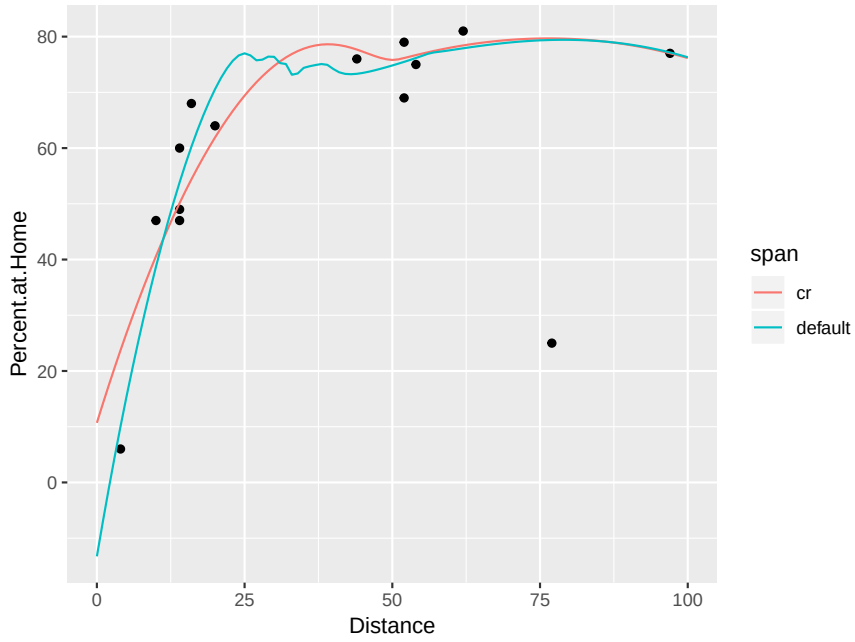



```
span.cr <- span[eps==min(eps)]
span.cr
```

```
## [1] 1.183333
```

```
fit <- loess(Percent.at.Home~Distance, span=span.cr,
             data = lunatics[-13, ],
             surface="direct")
fit1 <- loess(Percent.at.Home~Distance, span=0.75,
             data = lunatics[-13, ],
             surface="direct")
x <- 0:100
y <- c(predict(fit, x), predict(fit1, x))
df <- data.frame(x=c(x, x), y,
```

```
          span=rep(c("cr", "default"), each=101))
ggplot(data=lunatics,
            aes(Distance, Percent.at.Home)) +
  geom_point() +
  geom_line(data=df,  aes(x, y, color=span),
            inherit.aes = FALSE)
```



and we see that the smoother curve looks better.

Interval Estimation

How do we do interval estimation when using loess? As with the predict method for lm we can again get an estimate of the standard error by including *se=TRUE*. However, the predict command for loess does not have an *interval“* argument. So how do we know whether these are confidence or prediction intervals?

Let's find out. For this we do a little simulation study. We generate 100 x uniform on [0,10] and then 100 y=10+3x+N(0,3). We fit the loess model (using span=0.75) and predict yhat at x=5.0, then we do the same using lm.

```
B <- 1000
se.loess = rep(0, B)
se.lm = rep(0, B)
for (i in 1:B) {
  x <- runif(100, 0, 10)
  y <- 10 + 3 * x + rnorm(100, 0, 3)
  se.loess[i] <-  predict(loess(y ~ x),
                      newdata = data.frame(x = 5),
                      se = TRUE)$se.fit
```

```
  se.lm[i] <- predict(lm(y ~ x),
                      newdata = data.frame(x = 5),
                      se = TRUE)$se.fit
}
cat("loess  error: ", sd(se.loess), "\n")
```

```
## loess  error:  0.05266052
```

```
cat("lm  error: ", sd(se.lm), "\n")
```

```
## lm  error:  0.02182876
```

We see that the errors in loess are larger (with a larger standard deviation) than those of lm. This to be expected, after all we are using a lot more information in lm, namely the exact model for the data and the exact distribution of the residuals.

Notice also that

```
out <- predict(lm(y ~ x), newdata = data.frame(x = 5),
       se = TRUE, interval="confidence")
round(out$fit[2:3], 1)
```

```
## [1] 24.3 25.6
```

```
round(out$fit[1]+c(-1, 1)*qnorm(0.975)*out$se.fit, 1)
```

```
## [1] 24.3 25.6
```

so we see that these errors give us confidence intervals. But what if we want prediction intervals?

We have the following equations for the errors:

$$se_{fit} = \hat{\sigma}\sqrt{\frac{1}{n} + \frac{(x-\bar{x})^2}{s_{xx}}}$$
$$se_{pred} = \hat{\sigma}\sqrt{1 + \frac{1}{n} + \frac{(x-\bar{x})^2}{s_{xx}}}$$

where s_{xx} is the sum of squares and $\hat{\sigma}$ is an estimate of the standard deviation. It is part of the fit object with *as.numeric(fit[5])*. So

$$\frac{1}{n} + \frac{(x-\bar{x})^2}{s_{xx}} = \frac{se_{fit}}{\hat{\sigma}^2}$$
$$se_{pred} = \hat{\sigma}\sqrt{1 + \frac{se_{fit}^2}{\hat{\sigma}}} = \sqrt{\hat{\sigma}^2 + se_{fit}^2}$$

and so we can find prediction intervals with

```
fit <- loess(y ~ x)
yhat <- predict(fit)
sighat <- as.numeric(fit[5])
```

```
out <- predict(fit,
               newdata = data.frame(x = 5),
               se = TRUE)
se.pred <- sqrt(sighat^2 + out$se.fit^2)
round(out$fit[1]+c(-1, 1)*qnorm(0.975)*se.pred, 1)
```

```
## [1] 19.4 31.6
```

Example: Lunatics

Find a 90% prediction interval for the Percent.at.Home if the distance is 25miles, using the optimal span.

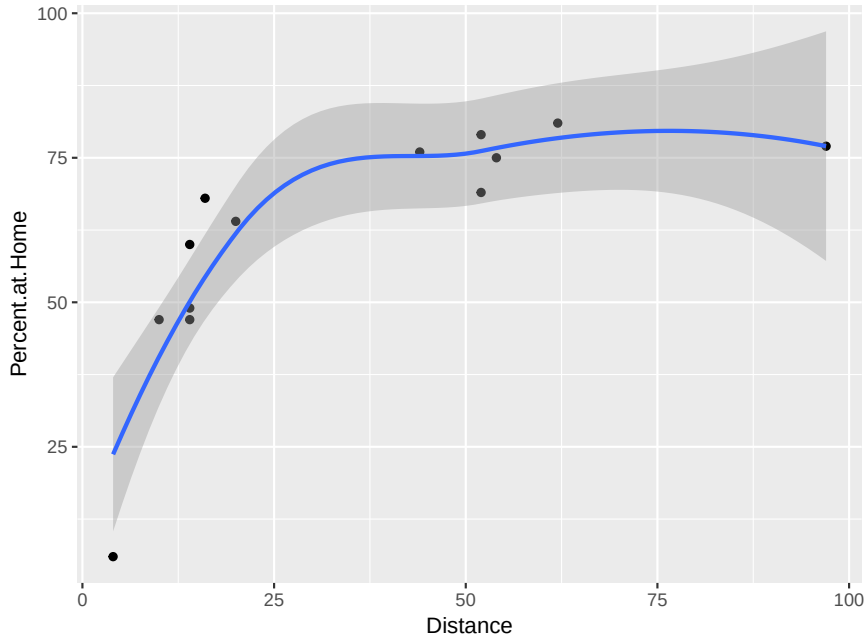
```
fit <- loess(Percent.at.Home~Distance, span=span.cr,
             data = lunatics[-13, ],
             surface="direct")
sighat <- as.numeric(fit[5])
out <- predict(fit,
               newdata = data.frame(Distance = 25),
               se = TRUE)
se.pred <- sqrt(sighat^2 + out$se.fit^2)
round(out$fit[1]+c(-1, 1)*qnorm(0.95)*se.pred, 1)
```

```
## [1] 53.1 85.7
```

Confidence Bands

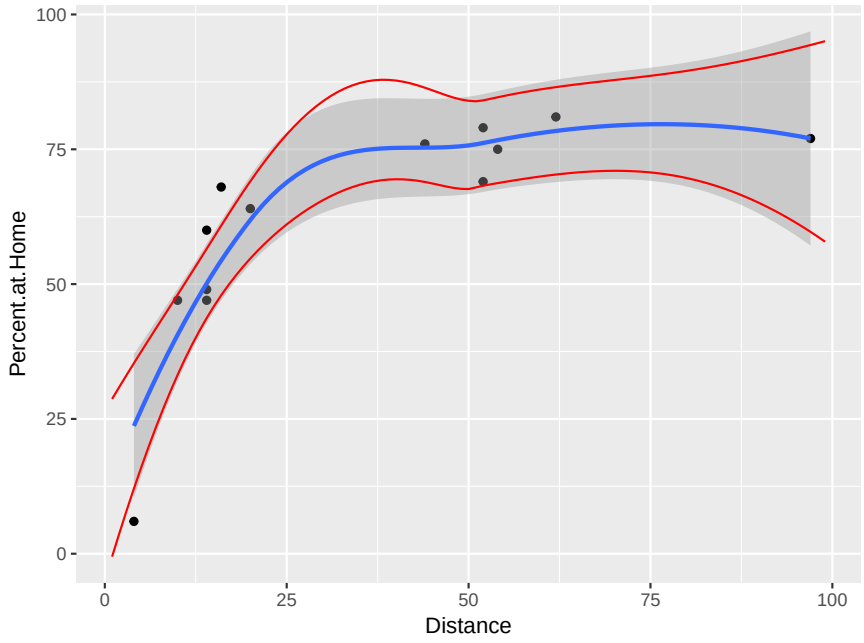
Let's have another look at geom_smooth:

```
ggplot(data=lunatics[-13, ],
            aes(Distance, Percent.at.Home)) +
  geom_point() +
  geom_smooth(span=span.cr)
```



what is this gray band? It is a shaded area between the lower and the upper 95% confidence intervals. We can recreate it ourselves:

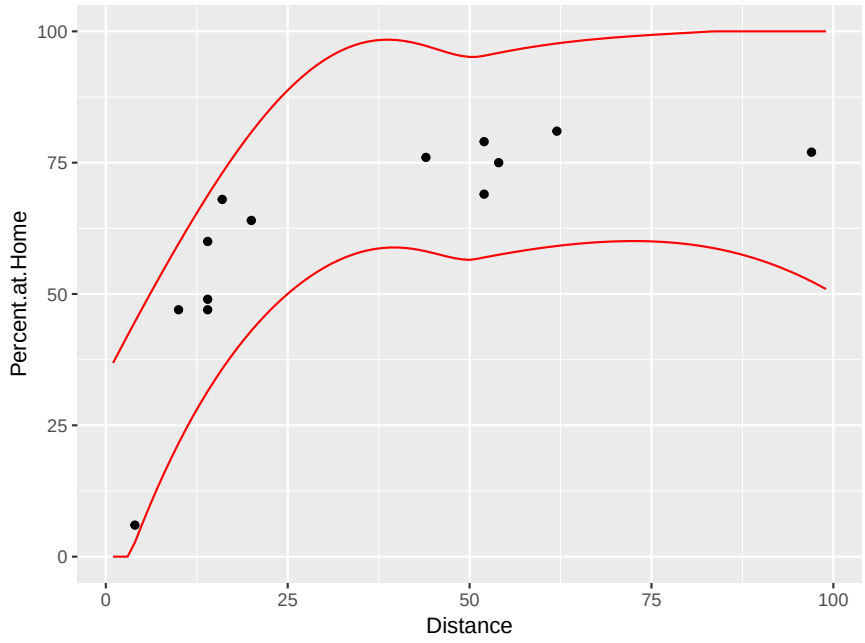
```
fit <- loess(Percent.at.Home~Distance, span=span.cr,
             data = lunatics[-13, ],
             surface="direct")
x <- 1:99
out <- predict(fit,
               newdata = data.frame(Distance = x),
               se = TRUE)
low <- out$fit-qnorm(0.975)*out$se.fit
high <- out$fit+qnorm(0.975)*out$se.fit
df.low <- data.frame(x=x, y=low)
df.high <- data.frame(x=x, y=high)
ggplot(data=lunatics[-13,],
            aes(Distance, Percent.at.Home)) +
  geom_point() +
  geom_smooth(method="loess", span=span.cr) +
  geom_line(data=df.low, aes(x, y), color="red")+
  geom_line(data=df.high, aes(x, y), color="red")
```



almost, except that geom_smooth does some additional adjustments. We won't worry about that for now.

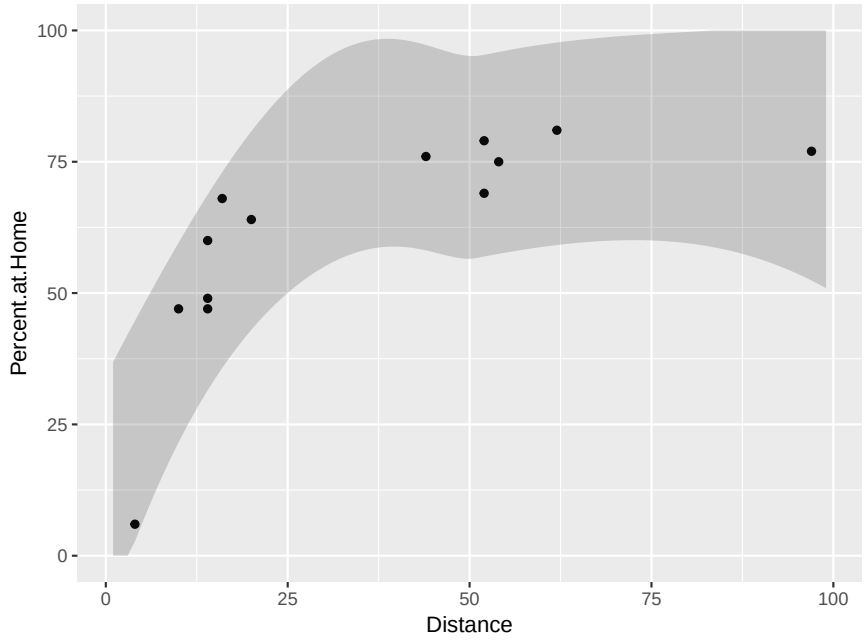
How about if we want the bands to show prediction intervals?

```
sighat <- as.numeric(fit[5])
se.pred <- sqrt(sighat^2 + out$se.fit^2)
low <- out$fit-qnorm(0.975)*se.pred
low[low<0] <- 0
high <- out$fit+qnorm(0.975)*se.pred
high[high>100] <- 100
df.low <- data.frame(x=x, y=low)
df.high <- data.frame(x=x, y=high)
ggplot(data=lunatics[-13,],
            aes(Distance, Percent.at.Home)) +
  geom_point() +
  geom_line(data=df.low, aes(x, y), color="red")+
  geom_line(data=df.high, aes(x, y), color="red")
```



or a bit nicer:

```
df1 <- data.frame(x, ymin=low, ymax=high)
ggplot(data=lunatics[-13,],
            aes(Distance, Percent.at.Home)) +
  geom_point() +
  geom_ribbon(data=df1,
              aes(x=x, ymin=ymin, ymax=ymax),
              alpha=0.2,
              inherit.aes = FALSE)
```



There is however an issue with these bands. From our recreation it is clear that they are *pointwise* confidence intervals, that is each is a 95% confidence interval for each x value. However, psychologically most people will look at them and interpret them as *simultaneous* confidence bands. That is, the 95% applies to the collection of intervals, not each interval alone.

Say we have n data points, and we find a $(1-\alpha)100\%$ confidence interval at each. If they are all independent we find

$$\begin{aligned}
&P(\text{at least one interval is wrong}) = \\
&1 - P(\text{no interval is wrong}) = \\
&1 - P\left(\cap_{k=1}^{n} I_k \text{ is right}\right) = \\
&1 - \prod_{k=1}^{n} P\left(I_k \text{ is right}\right) = \\
&1 - \prod_{k=1}^{n} (1-\alpha) = 1 - (1-\alpha)^n
\end{aligned}$$

and this goes to 1 as n grows larger.

To make matters worse, in a regression case intervals at neighboring points are clearly not independent, so we don't even know what the true simultaneous coverage might be.

Personally I am very reluctant to add such bands to graphs, but they are quite popular in many fields.

Nonlinear Parametric Models

Sometimes the model we wish to fit is known, up to parameters. Generally that is the case if there is a scientific theory that predicts the shape of the relationship. For example, radioactive decay is known to be exponential: $y = \alpha e^{-\beta t}$

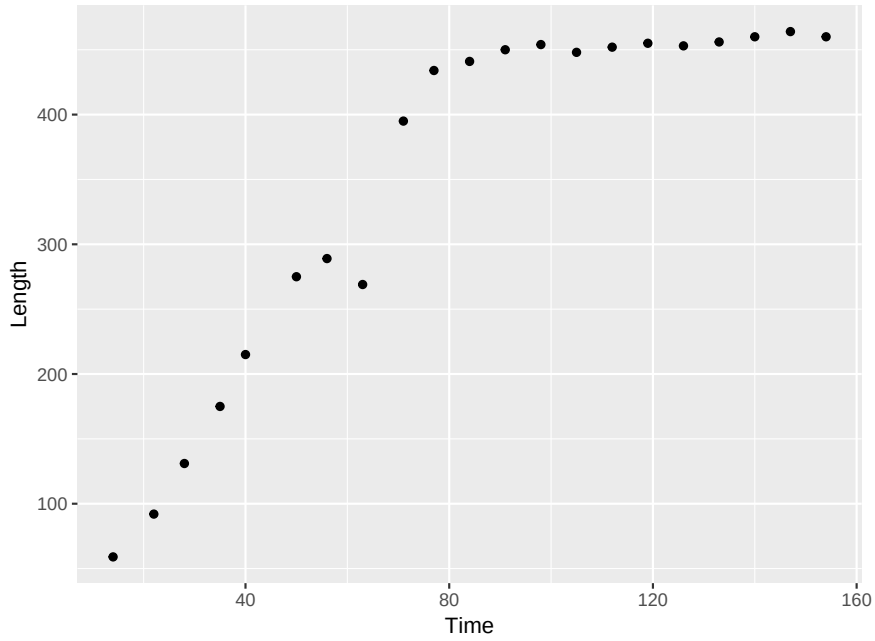
Example: Growth of Lobsters

Data from an experiment to raise Florida lobster in a controlled environment. The data shows the overall length and the age of a certain species of lobster.

```
kable.nice(lobster[1:10, ])
```

Time	Length
14	59
22	92
28	131
35	175
40	215
50	275
56	289
63	269
71	395
77	434

```
ggplot(data=lobster, aes(Time, Length)) +
  geom_point()
```



Now biology suggests that the relationship should be of the form

$$y = \frac{\beta_2}{1 + (\beta_2 - \beta_0)/\beta_0 \exp(\beta_1 t)} + \epsilon$$

where

- β_0 is the expected value of y at time t=0
- β_1 is a measure of the growth rate

- β_2 is the expected limit of y as $t \to \infty$

This is often called the **logistic** or **autocatalytic** model

How do we fit such a model, that is find "optimal" values of β_0, β_1 and β_2? Sometimes it is possible use transformations to "linearize" the model, for example we have of course $\log(y) = \log(\alpha) - \beta t$ for the radioactive decay model. This is not possible, though, for the logistic model, so we have to find a different solution.

Previously we have always used the method of least squares to estimate the parameters in our models, that is we minimized the "figure of merit"

$$\text{RSS} = \sum (y_i - \beta_0 - \beta_1 x_i)^2$$

the natural extension of this is to use

$$\text{RSS} = \sum (y_i - f(x_i; \boldsymbol{\beta}))^2$$

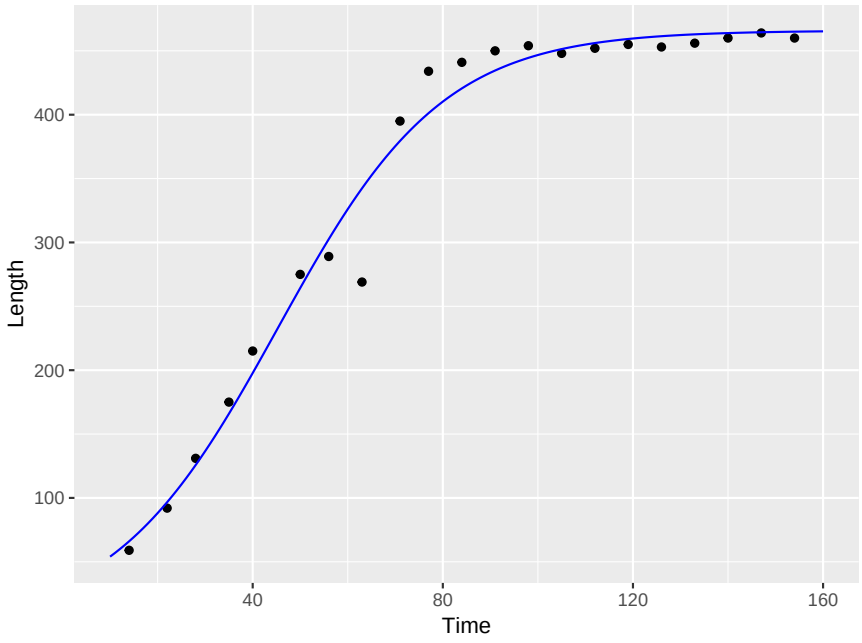
now for a linear model minimizing this expression could be done with *lm*. This however is still a minimization problem, and we can do it with

```
fit <- nls(Length ~ beta[3]/(1 + ((beta[3] -
          beta[1])/beta[1]) *  exp(beta[2] * Time)),
          start = list(beta = c(10, -0.1, 500)),
          data = lobster)
summary(fit)
```

```
##
## Formula: Length ~ beta[3]/(1 + ((beta[3] - beta[1])/beta[1]) * exp(beta[2] *
##     Time))
##
## Parameters:
##         Estimate Std. Error t value Pr(>|t|)
## beta1  32.008757   6.755720   4.738 0.000164
## beta2  -0.057557   0.004957 -11.612 8.55e-10
## beta3 465.884778   8.340739  55.857  < 2e-16
##
## Residual standard error: 21.63 on 18 degrees of freedom
##
## Number of iterations to convergence: 7
## Achieved convergence tolerance: 7.722e-06
```

```
x <- seq(10, 160, 1)
df <- data.frame(x=x,
                 y = predict(fit,
                  newdata = data.frame(Time = x)))
```

```
ggplot(data=lobster, aes(Time, Length)) +
 geom_point() +
 geom_line(data=df, aes(x, y), color="blue")
```



Example: Prime Number Theorem

That there were infinitely many prime numbers was first proven by the Greek mathematician Euclid at around 300BC. A serious study of how fast they grow was first begun by Adrienne-Marie Legendre. He studied the function N(k), which gives the number of primes less or equal to k. We can do the same. The primes up to 1,000,000 are available at

```
primes <- scan("C://Users//Wolfgang//dropbox//teaching//Computing-with-R//primes.txt")
primes <- as.integer(primes)
kable.nice(matrix(primes[1:100], ncol=10, byrow = TRUE))
```

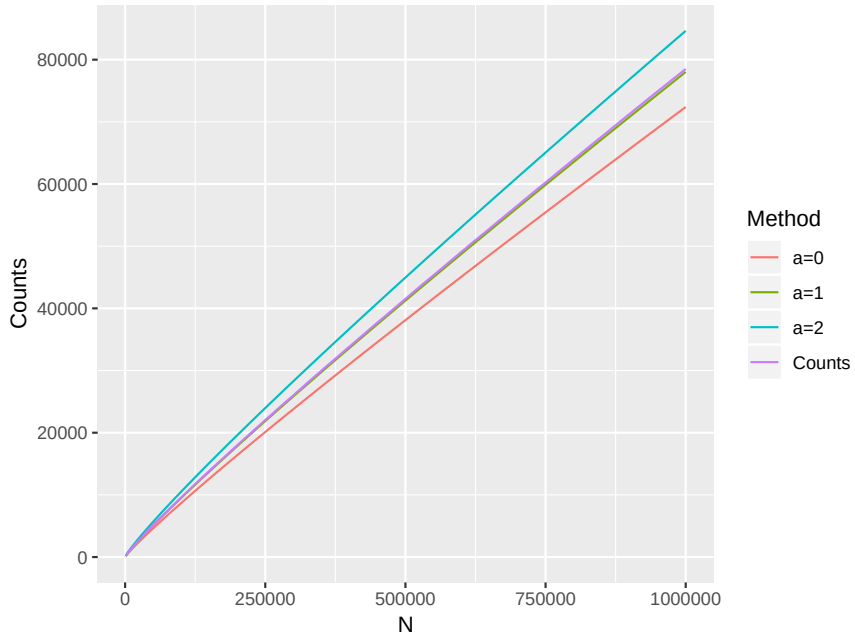
2	3	5	7	11	13	17	19	23	29
31	37	41	43	47	53	59	61	67	71
73	79	83	89	97	101	103	107	109	113
127	131	137	139	149	151	157	163	167	173
179	181	191	193	197	199	211	223	227	229
233	239	241	251	257	263	269	271	277	281
283	293	307	311	313	317	331	337	347	349
353	359	367	373	379	383	389	397	401	409
419	421	431	433	439	443	449	457	461	463
467	479	487	491	499	503	509	521	523	541

A detailed study of these primes led Legendre in 1798 to propose the function

$$N(k) = k/(\log k - \alpha)$$

Here is what that looks like for several values of α:

```
N <- function(k, alpha) {
  k/(log(k)-alpha)
}
k <- seq(1000, 1e6, length=250)
exact.counts <- k
for(i in 1:250)
  exact.counts[i] <- sum(primes<k[i])
df <- data.frame(N=c(k, k, k, k),
      Counts=c(exact.counts, N(k, 0), N(k, 1), N(k, 2)),
      Method=rep(c("Counts", "a=0", "a=1", "a=2"),
                 each=250))
ggplot(df, aes(N, Counts, color=Method)) +
  geom_line()
```



and so it seems a value of $\alpha = 1$ is good.

Legendre however was not satisfied with that, he wanted to find the optimal answer. So he found the least squares solution!

```
fit <- nls(exact.counts ~ k/(log(k) - alpha),
           start = list(alpha = 0))
coef(fit)
```

```
##    alpha
## 1.082001
```

and so he claimed that

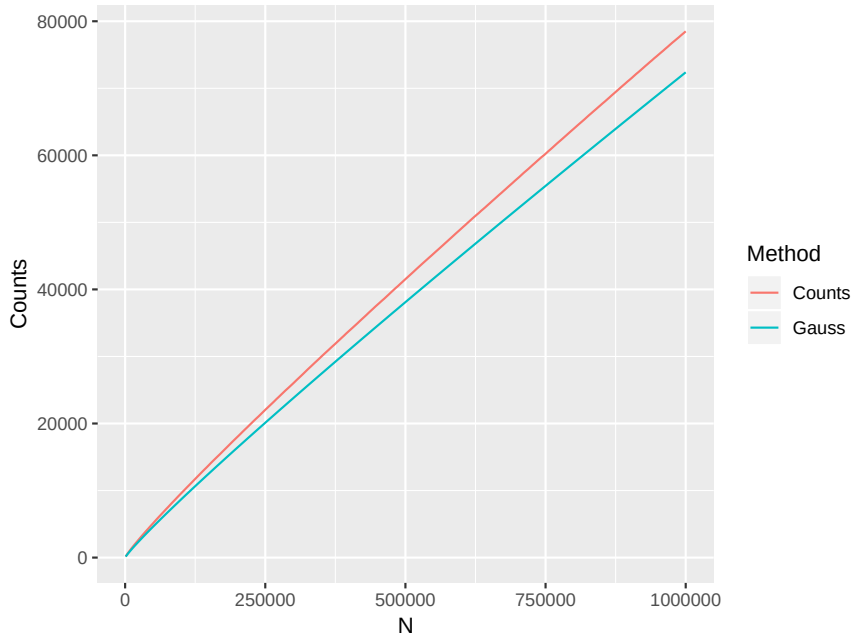
$$N(k) = k/(\log k - 1.08)$$

Around the same time German mathematician Carl Friedrich Gauss also looked at this problem, and he made a different conjecture. He said

$$N(k) = k/\log k$$

That was a rather strange guess, because it looks like this:

```
N <- function(k, alpha) {
  k/(log(k)-alpha)
}
k <- seq(1000, 1e6, length=250)
exact.counts <- k
for(i in 1:250)
  exact.counts[i] <- sum(primes<k[i])
df <- data.frame(N=c(k, k),
      Counts=c(exact.counts, N(k, 0)),
      Method=rep(c("Counts", "Gauss"),   each=250))
ggplot(df, aes(N, Counts, color=Method)) +
  geom_line()
```



and it surely looks like the two curves are growing further apart. However, almost 100 years later in 1896 the French mathematicians Jacques-Salomon Hadamard and Charles de la Valée Poussin independently showed that Gauss was right!

From our modern point of view we might say Legendre was guilty of **over-fitting**!

Logistic Regression - General Linear Models

In all the examples so far we always had a quantitative response. In this section we will study the case of a categorical response.

Example: Challenger shuttle disaster

We begin with a very famous dataset from the Challenger shuttle disaster. On Jan 28, 1986, at 11.38 am EST, the space shuttle challenger was launched from Cape Canaveral, Florida. The mission ended 73 seconds later when the Challenger exploded. All 7 crew members were killed.

Challenger Disaster Movie

What happened?

Hot propellant gases flew past the aft joint of the right solid rocket booster, burning through two rubber O-rings. An investigation ensued into the reliability of the shuttle's propulsion system. The explosion was eventually traced to the failure of one of the three field joints on one of the two solid booster rockets. Each of these six field joints includes two O-rings, designated as primary and secondary, which fail when phenomena called erosion and blowby both occur.

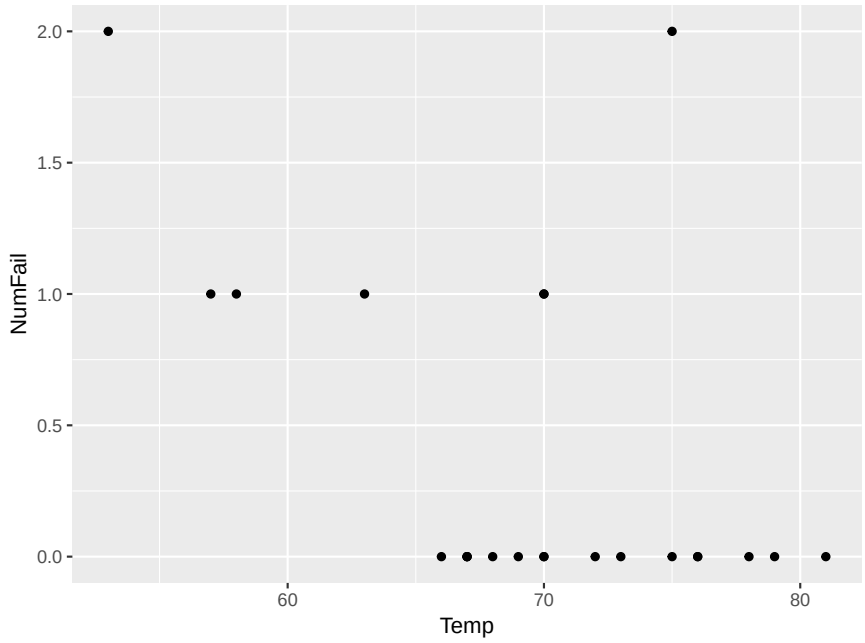
The night before the launch a decision had to be made regarding launch safety. The discussion among engineers and managers leading to this decision included concern that the probability of failure of the O-rings depended on the temperature t at launch, which was forecase to be 31 degrees F. There are strong engineering reasons based on the composition of O-rings to support the judgment that failure probability may rise monotonically as temperature drops.

The discussion centered on the following data from the previous 23 shuttle launches:

```
kable.nice(head(shuttle))
```

Temp	NumFail	Failure
66	0	0
70	1	1
69	0	0
68	0	0
67	0	0
72	0	0

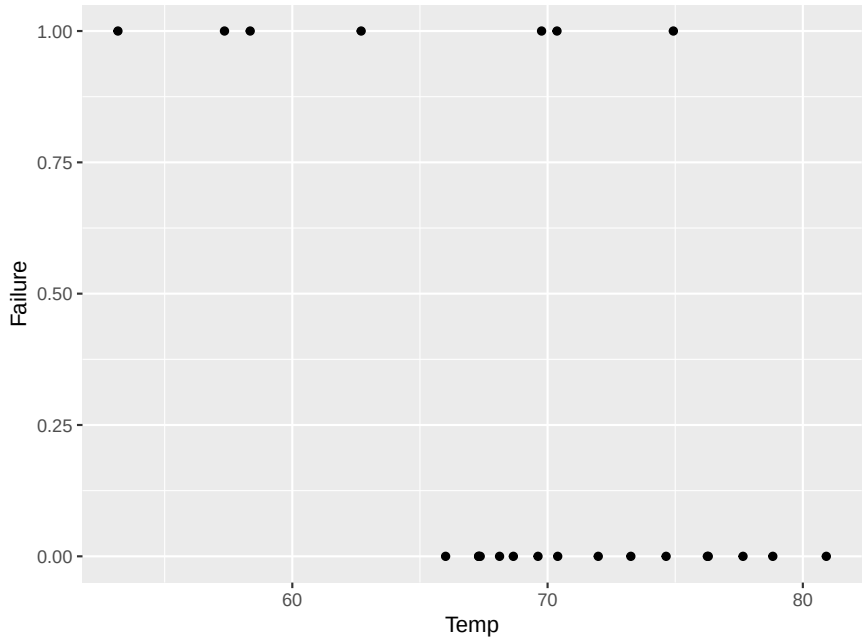
```
ggplot(data=shuttle, aes(Temp, NumFail)) +
  geom_point()
```



there seems to be a tendency for failures at lower temperatures.

The variable Failure is an indicator of failure or not:

```
plt <- ggplot(data=shuttle, aes(Temp, Failure)) +
  geom_jitter(height = 0)
plt
```



Again we want to use regression to study the relationship between temperature and failure. But now failure is categorical, and so the x and the y variable are no longer measured in the same way.

The way to connect them is to predict the probability of failure, which again is a quantitative variable. This is done as follows: we have responses $y_1, .., y_n$ which can be modeled as $Y_i \sim Ber(\pi_i)$. The π are related to the predictor variable x via the *link function*

$$\log\left(\frac{p}{1-p}\right) = \alpha + \beta x$$

this is called the *logit* link function. There are others as well.

We can invert the logit function:

$$\pi(x) = \frac{e^{\alpha+\beta x}}{1+e^{\alpha+\beta x}}$$

notice that this rules out $\pi(x) = 0$ or 1. There are other link functions that don't do that.

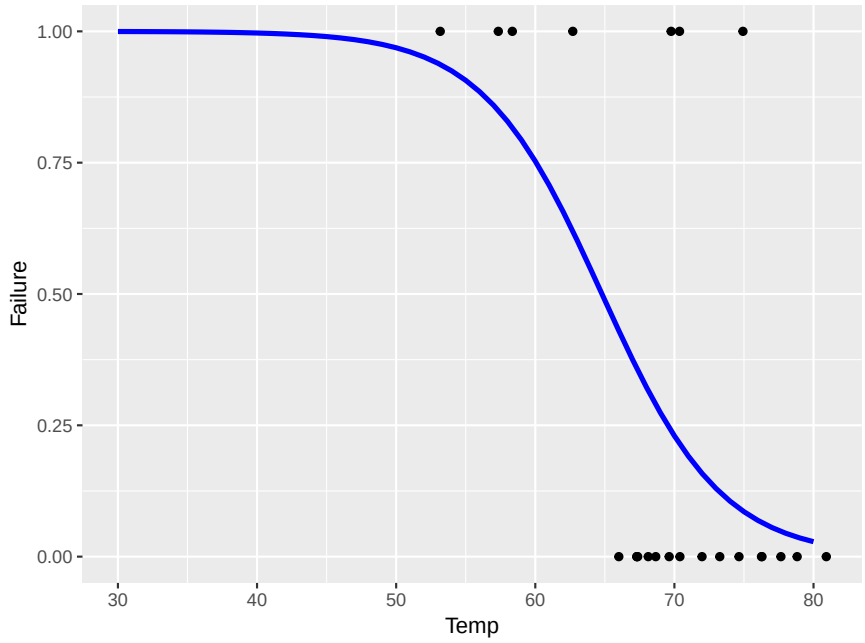
How do fit such a model, that is find α and β? For linear regression we used the method of least squares, which was possible because we could directly compare x and y. This is not the case here because p and x have different forms, which is why we needed a link function. Instead one uses maximum likelihood for the estimation. In R this is done using the command *glm* with the argument *family=binomial*:

```
fit <- glm(Failure~Temp,
           family=binomial,
           data=shuttle)
fit
```

```
##
## Call:  glm(formula = Failure ~ Temp, family = binomial, data = shuttle)
##
## Coefficients:
## (Intercept)         Temp
##     15.0429      -0.2322
##
## Degrees of Freedom: 22 Total (i.e. Null);  21 Residual
## Null Deviance:       28.27
## Residual Deviance: 20.32     AIC: 24.32
```

Let's see what this looks like

```
x <- seq(30, 80, 1)
df <- data.frame(x=x,
                 y=predict(fit, data.frame(Temp=x),
                                type="response"))
plt +
  geom_line(data=df, aes(x, y),
            color="blue", size=1.2)
```



we see that at the expected launch temperature of 32F the failure probability is 1.

What would be a 95% confidence interval for the probability at 32F?

```
tmp <- predict(fit, data.frame(Temp=32),
     type="response", se.fit=TRUE)
round(tmp$fit +c(-1, 1)*qnorm(0.975)*tmp$se.fit, 3)
```

```
## [1] 0.996 1.003
```

but there is something silly about this interval: it goes beyond 1! This is a consequence of using normal theory intervals. Here is a better solution:

```
tmp <- predict(fit, data.frame(Temp=32),
     type="link", se.fit=TRUE)
e <- tmp$fit
r <- tmp$se.fit
e
```

```
##        1
## 7.613694
```

```
r
```

```
## [1] 3.933421
```

```
cr <- qnorm(0.975)
round(c(exp(e-cr*r)/(1+exp(e-cr*r)),
        exp(e+cr*r)/(1+exp(e+cr*r))), 3)
```

```
##     1     1
## 0.476 1.000
```

but this has a much lower (likely to low) lower limit.

Warp Breaks

The data set gives the results of an experiment to determine the effect of wool type (A or B) and tension (low, medium or high) on the number of warp breaks per loom. Data was collected for nine looms for each combination of settings.

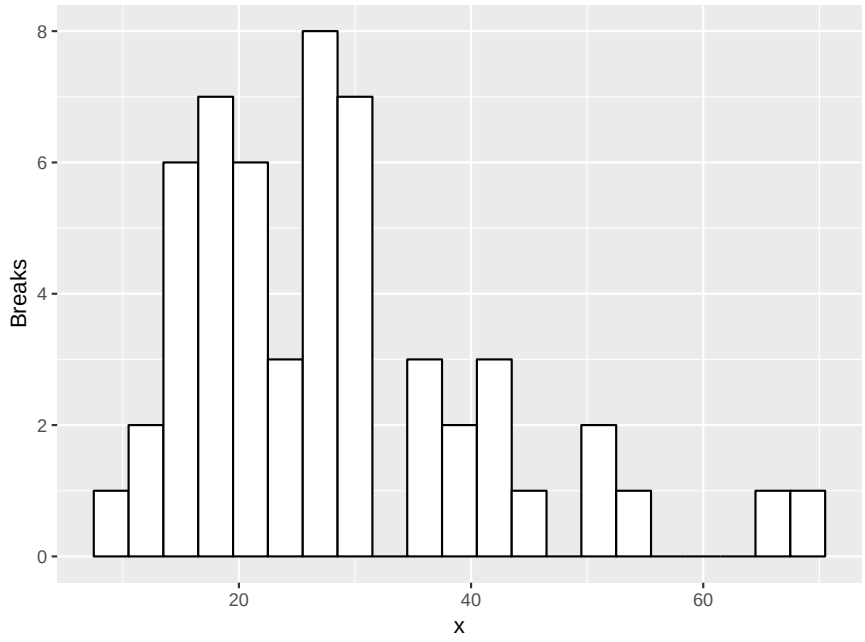
```
kable.nice(head(warpbreaks))
```

breaks	wool	tension
26	A	L
30	A	L
54	A	L
25	A	L
70	A	L
52	A	L

we want to build a model relating the wool type and tension to the number of breaks.

What distribution might be appropriate for *breaks*? First, let's have a look at them:

```
bw <- diff(range(warpbreaks$breaks))/20
ggplot(warpbreaks, aes(x=breaks)) +
  geom_histogram(color = "black",
    fill = "white",
    binwidth = bw) +
    labs(x = "x", y = "Breaks")
```



our data is counts with a bit of skew to the right. This is typical for data from a *Poisson* distribution.

Here is another argument in favor of a Poisson: Each loom could be considered as a series of small intervals. We then would have a large number of such intervals, each of which has a small probability of a break. The total number of breaks would be the sum of the breaks in each interval, and therefore would be Binomial. But in this case the Poisson approximation to the Binomial would be very good.

Again we want to use regression to relate type and tension to breaks. In the case of a Poisson response variable the link function is given by the logarithm.

```
fit <- glm(breaks~wool*tension,
           data=warpbreaks,
           family=poisson)
summary(fit)
```

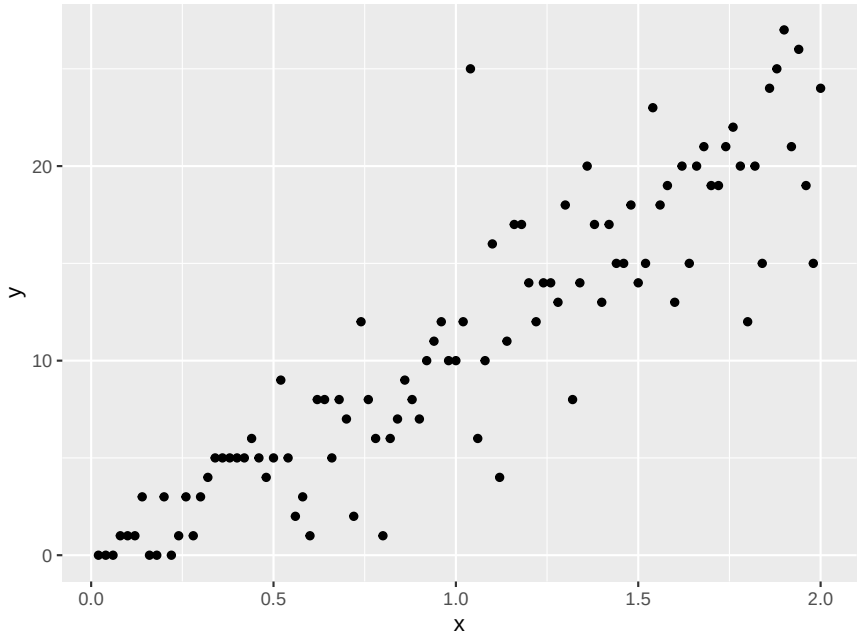
```
##
## Call:
## glm(formula = breaks ~ wool * tension, family = poisson, data = warpbreaks)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.3383  -1.4844  -0.1291   1.1725   3.5153
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)     3.79674    0.04994  76.030  < 2e-16
## woolB          -0.45663    0.08019  -5.694 1.24e-08
## tensionM       -0.61868    0.08440  -7.330 2.30e-13
## tensionH       -0.59580    0.08378  -7.112 1.15e-12
## woolB:tensionM  0.63818    0.12215   5.224 1.75e-07
## woolB:tensionH  0.18836    0.12990   1.450    0.147
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 297.37  on 53  degrees of freedom
## Residual deviance: 182.31  on 48  degrees of freedom
## AIC: 468.97
##
## Number of Fisher Scoring iterations: 4
```

and we see that all terms except one interaction term are stat. significant.

Let's do our own little study of Poisson regression. First we generate some data:

```
x <- 1:100/50
df <- data.frame(x=x, y=rpois(100, 10*x))
```

```
plt <- ggplot(data=df, aes(x, y)) +
  geom_point()
plt
```

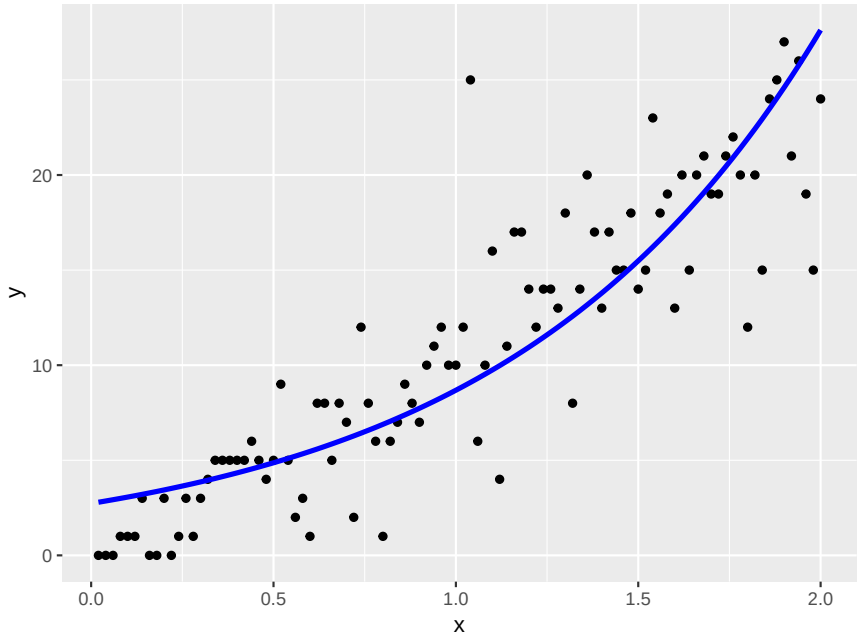



```
fit <- glm(y~x,
           data=df,
           family=poisson)
summary(fit)
```

```
##
## Call:
## glm(formula = y ~ x, family = poisson, data = df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8140  -0.7965   0.1136   0.5186   4.3300
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.00420    0.08665   11.59   <2e-16
## x            1.15719    0.05946   19.46   <2e-16
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 585.14  on 99  degrees of freedom
## Residual deviance: 156.51  on 98  degrees of freedom
## AIC: 539.39
##
```

```
## Number of Fisher Scoring iterations: 5
```

```
df1 <- df
df1$y <- predict(fit, type="response")
plt +
  geom_line(data=df1, aes(x, y), color="blue", size=1.2)
```



and that looks quite good!

Models with more than one predictor

ANOVA with more than one factor

ANOVA can be viewed as a linear model in the following way: Say we have a quantitative response variable y and a categorical predictor x. Then

$$y_{ij} = \mu + \alpha_i + \epsilon_{ij}$$

so the ij^{th} observation comes from an overall mean μ, a contribution α_i due to the fact that this observation comes from the i^{th} group and an error term.

With this formalism the null hypothesis is

$$H_0 : \alpha_1 = .. = \alpha_k = 0$$

Now this extends naturally to more than one predictor. Say there are two, then

$$y_{ijk} = \mu + \alpha_i + \beta_j + \gamma_{ij} + \epsilon_{ijk}$$

Notice the new term γ_{ij}. It is meant to measure *interaction*, that is a possible relationship (*association*) between the factors.

If there is no such relationship, the model simplifies to an *additive* model:

$$y_{ijk} = \mu + \alpha_i + \beta_j + \epsilon_{ijk}$$

Example: Testing Hearing Aids

Reference: Loven, Faith. (1981). A Study of the Interlist Equivalency of the CID W-22 Word List Presented in Quiet and in Noise. Unpublished MS Thesis, University of Iowa.

Description: Percent of a Standard 50-word list heard correctly in the presence of background noise. 24 subjects with normal hearing listened to standard audiology tapes of English words at low volume with a noisy background. They repeated the words and were scored correct or incorrect in their perception of the words. The order of list presentation was randomized.

The word lists are standard audiology tools for assessing hearing. They are calibrated to be equally difficult to perceive. However, the original calibration was performed with normal-hearing subjects and no noise background. The experimenter wished to determine whether the lists were still equally difficult to understand in the presence of a noisy background.

```
kable.nice(head(hearingaid))
```

Subject	List	Score
1	1	28
2	1	24
3	1	32
4	1	30
5	1	34
6	1	30

Notice that the values in both Subject and List are NOT numbers but labels, so both of them are categorical!

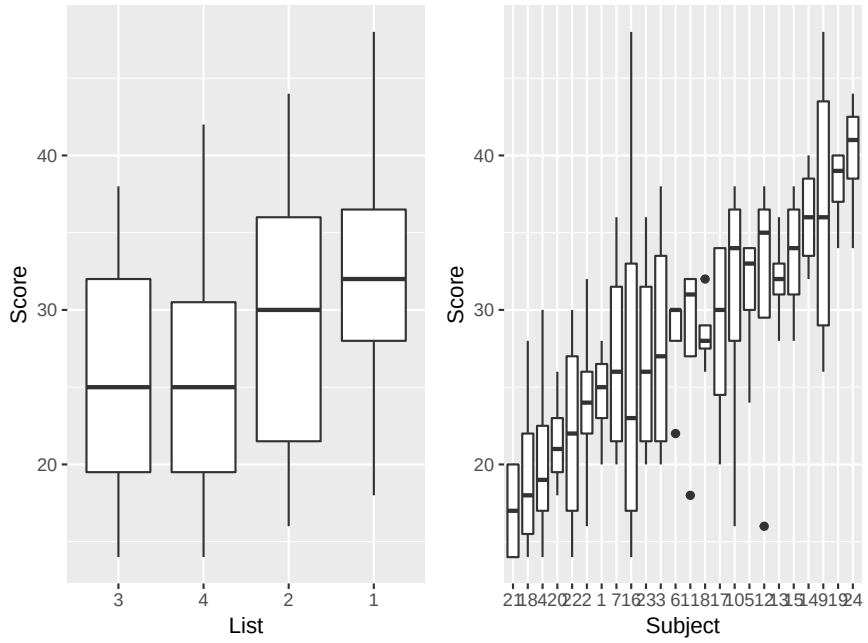
Because there are two factors this is called a **twoway ANOVA**. More specifically, this is a **Randomized Block design** with List as the factor and Subject as a blocking variable because the factor Subject is not of interest in itself.

Let's start by looking at the boxplots. Because neither List nor Subject has an obvious ordering we use size:

```
tmp <- tapply(hearingaid$Score, hearingaid$List, mean)
hearingaid$List <- factor(hearingaid$List,
                          levels=order(tmp),
                          ordered = TRUE)
tmp <- tapply(hearingaid$Score, hearingaid$Subject, mean)
```

```
hearingaid$Subject <- factor(hearingaid$Subject,
                             levels=order(tmp),
                             ordered = TRUE)
```

```
pushViewport(viewport(layout = grid.layout(1, 2)))
print(ggplot(data=hearingaid, aes(List, Score)) +
        geom_boxplot(),
  vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(ggplot(data=hearingaid, aes(Subject, Score)) +
       geom_boxplot() ,
  vp=viewport(layout.pos.row=1, layout.pos.col=2))
```



this shows why we should include Subject in the analysis: it has a large variation.

The summary statistics are:

```
sum.tbl <-
  data.frame(
    List=c(3, 4, 2, 1),
    n=as.integer(tapply(hearingaid$Score,
                hearingaid$List,length)),
    Mean=round(tapply(hearingaid$Score,
                      hearingaid$List, mean), 1),
    Sd=round(tapply(hearingaid$Score,
                    hearingaid$List, sd), 2)
)
rownames(sum.tbl) <- NULL
```

List	n	Mean	Sd
3	24	25.2	8.32
4	24	25.6	7.78
2	24	29.7	8.06
1	24	32.8	7.41

```
kable.nice(sum.tbl)
```

Because Subject is the blocking variable one would normally not include a table of summary statistics.

Now for the test, or better tests, because we can in general test for either Subject or List. The routine we will use is again *aov*:

```
fit <- aov(Score~., data=hearingaid)
summary(fit)
```

```
##             Df Sum Sq Mean Sq F value   Pr(>F)
## Subject     23   3232  140.51   3.868 6.96e-06
## List         3    920  306.82   8.446 7.41e-05
## Residuals   69   2507   36.33
```

So we have two tests, one for List and one for Subject. However, only the one for List is of interest:

1) Parameters of interest: List group means

2) Method of analysis: ANOVA

3) Assumptions of Method: residuals have a normal distribution, groups have equal variance

4) Type I error probability $\alpha = 0.05$

5) Null hypothesis H_0: $\mu_1 = .. = \mu_4$ (List groups have the same means)

6) Alternative hypothesis H_a: $\mu_i \neq \mu_j$ (at least two List groups have different means)

7) p value=0.00

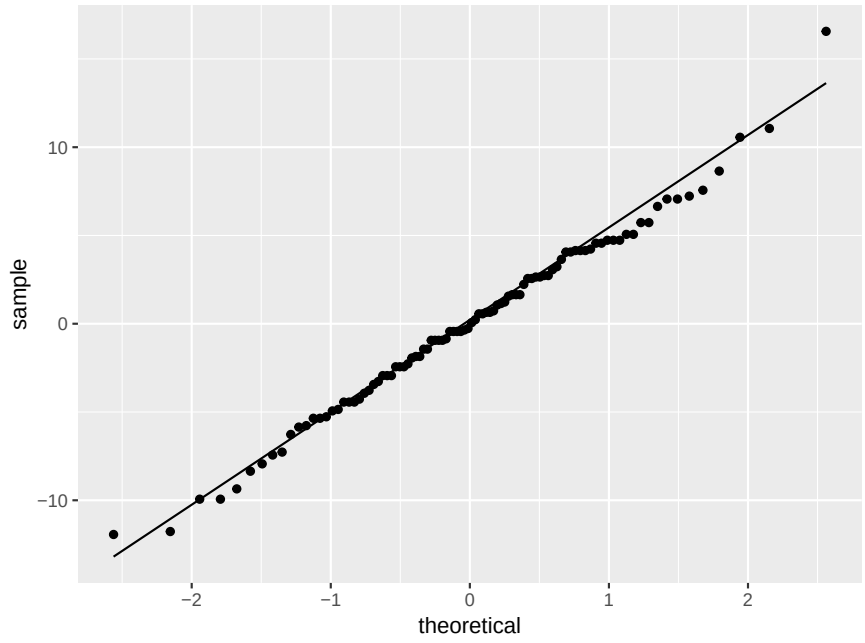
8) 0.000<0.05, there is some evidence that the group means are not the same, that List means are different)

As always we need to check the assumptions:

- **normal residuals**

The normal plot of residuals looks fine.

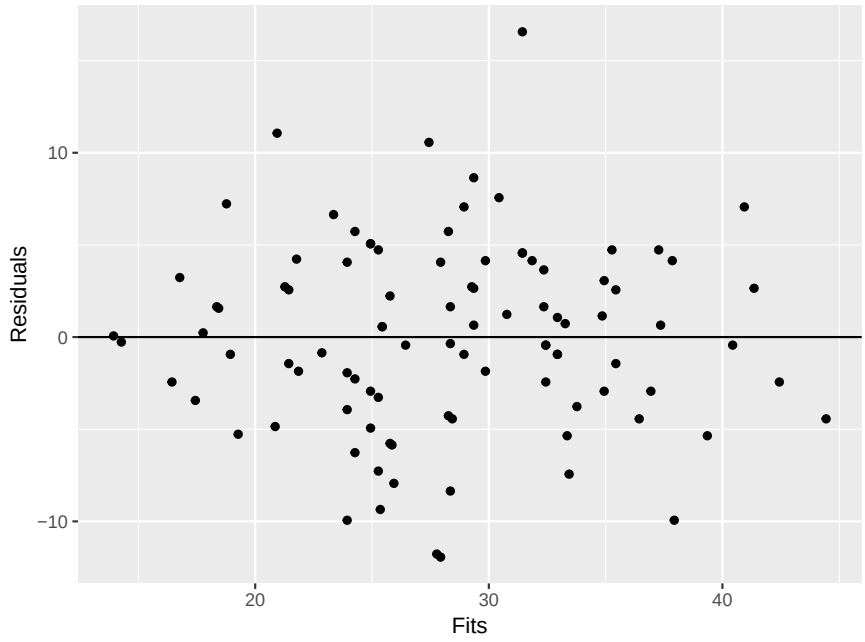
```
df <- data.frame(Residuals=resid(fit),
                 Fits = fitted(fit))
ggplot(data=df, aes(sample=Residuals)) +
  geom_qq() +geom_qq_line()
```



- **equal variance**

In a oneway ANOVA we could just find the group standard deviations and compare them. Now (and in general if there is more than one factor) this is no longer a good idea, mainly because there are to many factor level combinations (4*24 here) and not enough observations for each (one here). Instead we will do the same as in the regression case, namely check the residual vs. fits plot for a change in spread from left to right.

```
ggplot(data=df, aes(Fits, Residuals)) +
  geom_point() +
  geom_hline(yintercept = 0)
```

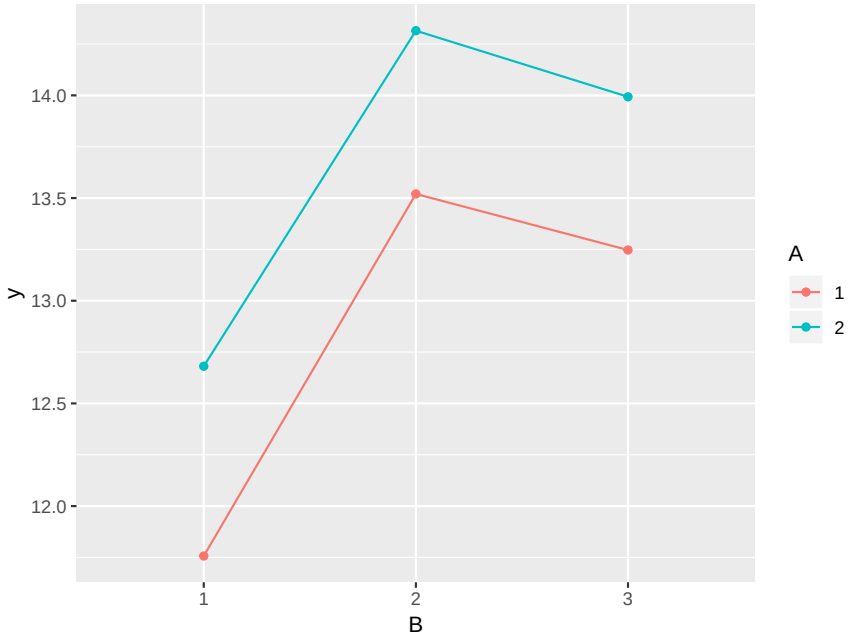


again, everything looks fine.

Notice that the ANOVA table also has the test for the Subject means. This is not very interesting, the boxplot already makes it clear that different subjects have very different hearing abilities. If that were not so, we would eliminate Subject and run a oneway ANOVA.

Because we now have two factors, we need to worry about an additional problem, namely whether or not there is a relationship between the two factors. This is called **interaction**.

To check we can draw the **interaction plot**.

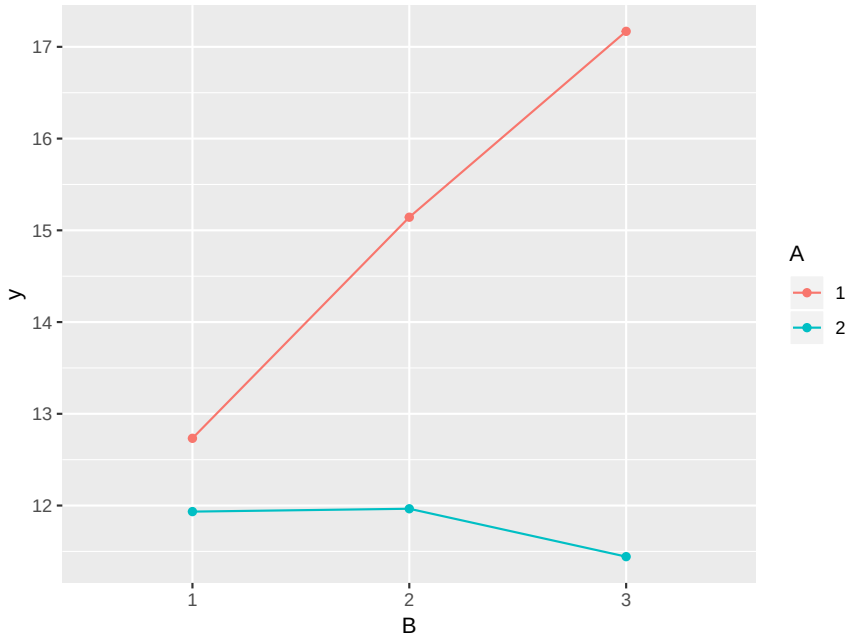


What is drawn here? First we find the *mean response* for each factor-level combination. Then plot those points vs. one of the factors and finally connect the dots that belong to the other factor.

Here the line segments are almost parallel. This implies that for any value of the factor A going from one value of B to the next adds **the same** amount to the response. So if we go from B=1 to B=2 **both** lines move up by about 2.0, and if we go from B=2 to B=3 **both** lines move down by 0.75.

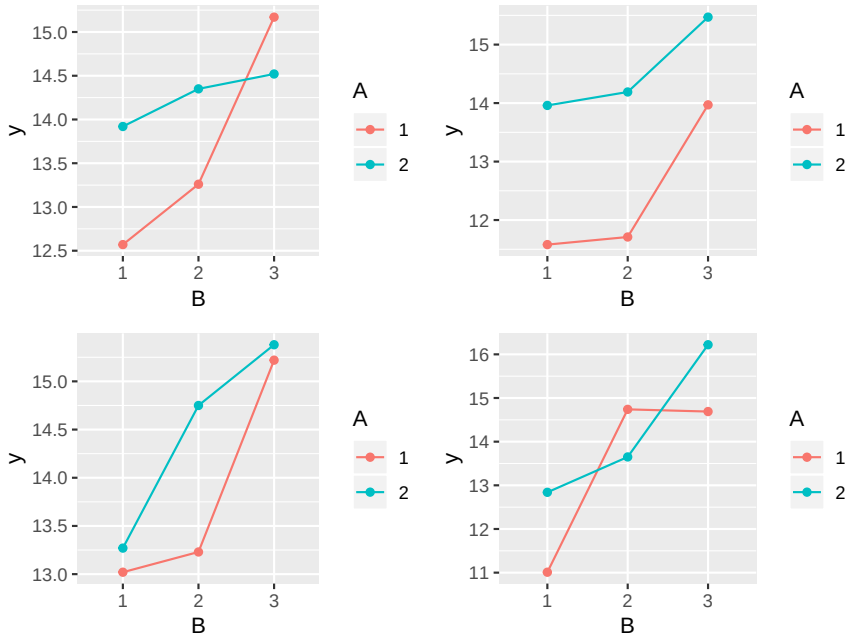
Because of this we call such a model **additive**

Now consider the following interactions plot:



Here as we go from B=2 to B=3 the line goes up by 4 **if A=1** but it goes down by 0.5 **if A=1**.

Deciding from the graph whether or not there is interaction is not always easy. Here are four interaction plots from a simulated data set, all guaranteed NOT to have any interaction:

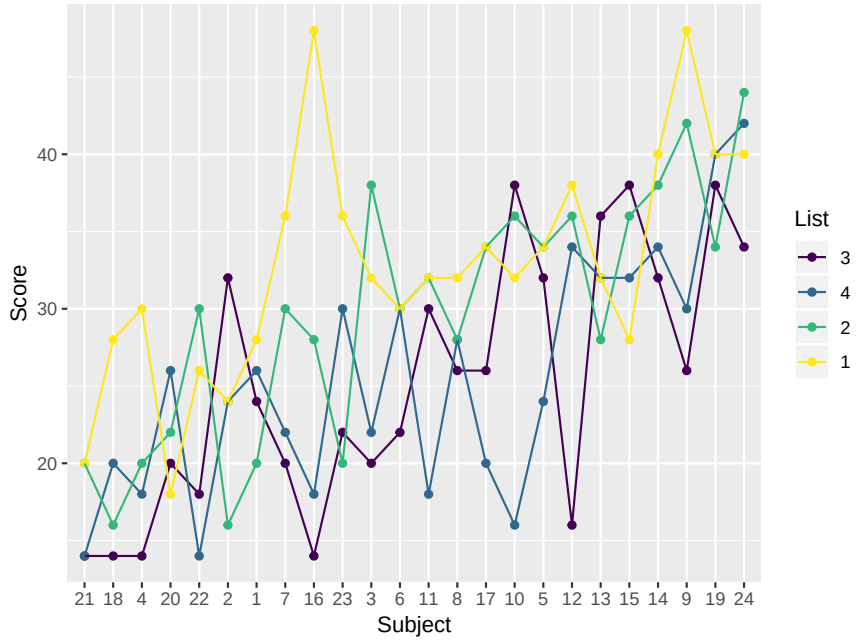


This is even worse because in ANOVA problems we often have very small data sets, so there is a great amount of variation in these graphs from sample to sample.

So it would be nice if we could actually test for interaction, but that requires **repeated measurements**.

In the hearing aid data we only have one observation for each combination of Subject and List, so we need to decide on the basis of the interaction plot:

```
ggplot(data = hearingaid,
       aes(Subject , Score,
           colour = List, group=List)) +
      stat_summary(fun.y=mean, geom="point")+
      stat_summary(fun.y=mean, geom="line")
```



There seems to be interaction between Lists and Subjects

Finally, it would of course be interesting to study just which lists are different, that is we could do a **multiple comparison**:

```
TukeyHSD(fit)$List
```

```
##          diff        lwr       upr        p adj
## 4-3 0.3333333 -4.2473895  4.914056 0.9974833454
## 2-3 4.4166667 -0.1640562  8.997390 0.0628040720
## 1-3 7.5000000  2.9192771 12.080723 0.0003038968
## 2-4 4.0833333 -0.4973895  8.664056 0.0974569856
## 1-4 7.1666667  2.5859438 11.747390 0.0005910298
## 1-2 3.0833333 -1.4973895  7.664056 0.2954408670
```

so List 1 is statistically significantly different from Lists 3 and 4.

No other differences are statistically significant.

Because Subject is only a blocking variable we won't a multiple comparison for it.

Example: Gasoline Type and Milage

In an experiment to study gas mileage four different blends of gasoline are tested in each of three makes of automobiles. The cars are driven a fixed distance to determine the mpg (miles per gallon) The experiment is repeated three times for each blend-automobile combination. (Taken from Lyman Ott)

Note that the interest here is indifferent gasoline blends, automobile is a blocking variable, so this is a randomized block design.

Gasoline is numbers, but these are just codes for different blends, so it is a categorical variable or factor.

```
kable.nice(head(gasoline))
```

MPG	Gasoline	Automobile
22.7	1	A
22.4	1	A
22.9	1	A
21.5	2	A
21.8	2	A
21.6	2	A

Here is an interesting calculation:

```
table(gasoline$Gasoline, gasoline$Automobile)
```

```
##
##     A B C
##   1 3 3 3
##   2 3 3 3
##   3 3 3 3
##   4 3 3 3
```

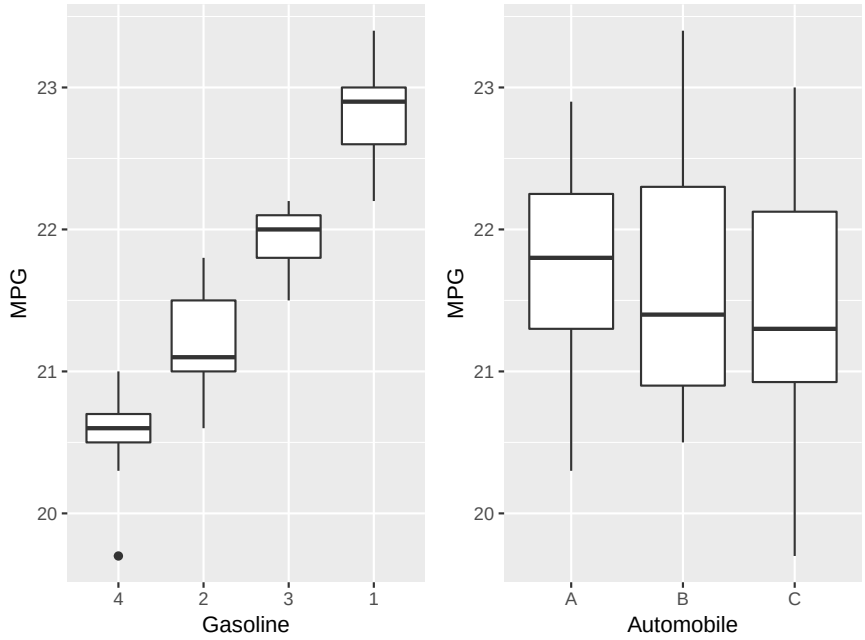
This shows us two things:

1. we have *repeated measurements* (several observations per factor-level combination)
2. we have a *balanced design* (the same number of repetitions in each factor-level combination)

This second feature used to be quite important because the calculations in a balanced design are much simpler. Nowadays with fast computers this is not important anymore. There are still good reasons why you want to design your experiment to have a balanced design if possible, though!

```
tmp <- tapply(gasoline$MPG, gasoline$Gasoline, mean)
gasoline$Gasoline <- factor(gasoline$Gasoline,
                            levels=order(tmp),
                            ordered = TRUE)
```

```
pushViewport(viewport(layout = grid.layout(1, 2)))
print(ggplot(data=gasoline, aes(Gasoline, MPG)) +
        geom_boxplot() ,
  vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(ggplot(data=gasoline, aes(Automobile, MPG)) +
       geom_boxplot() ,
  vp=viewport(layout.pos.row=1, layout.pos.col=2))
```



the boxplots suggest a difference between blends but not between automobiles.

The summary statistics are

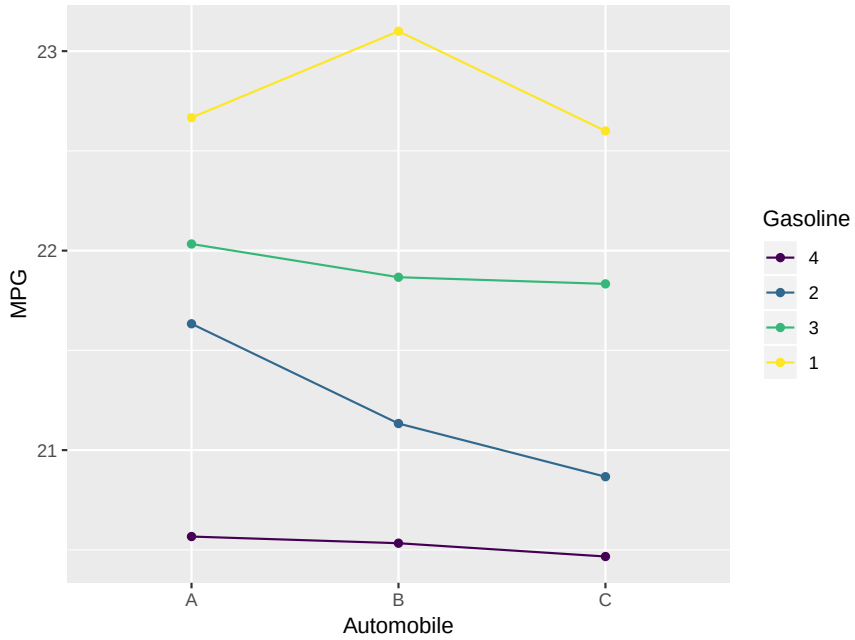
```
sum.tbl <-
  data.frame(
    Gasoline=c(4, 2, 3, 1),
    n=as.integer(tapply(gasoline$MPG,
                        gasoline$Gasoline,length)),
    Mean=round(tapply(gasoline$MPG,
                      gasoline$Gasoline, mean), 1),
    Sd=round(tapply(gasoline$MPG,
                    gasoline$Gasoline, sd), 2)
)
rownames(sum.tbl) <- NULL
```

Gasoline	n	Mean	Sd
4	9	20.5	0.36
2	9	21.2	0.37
3	9	21.9	0.25
1	9	22.8	0.36

```
kable.nice(sum.tbl)
```

Interaction:

```
ggplot(data = gasoline,
       aes(Automobile , MPG,
           colour = Gasoline, group=Gasoline)) +
      stat_summary(fun.y=mean, geom="point")+
      stat_summary(fun.y=mean, geom="line")
```



Lines are (almost) parallel, so there is no indication of interaction. We have **repeated measurements** (3 per factor-level combination), so we can test for this:

```
fit <- aov(MPG~Gasoline*Automobile, data=gasoline)
summary(fit)
```

```
##                     Df Sum Sq Mean Sq F value   Pr(>F)
## Gasoline             3 25.405   8.468  90.464 3.21e-13
## Automobile           2  0.527   0.263   2.813   0.0799
## Gasoline:Automobile  6  0.909   0.151   1.618   0.1854
## Residuals           24  2.247   0.094
```

1) Parameters of interest: Interaction

2) Method of analysis: ANOVA

3) Assumptions of Method: residuals have a normal distribution, groups have equal variance

4) Type I error probability α=0.05

5) Null hypothesis H_0 : no interaction

6) Alternative hypothesis H_a: some interaction

7) p value = 0.1854
8) 0.1854 > 0.05, there is no evidence of interaction.

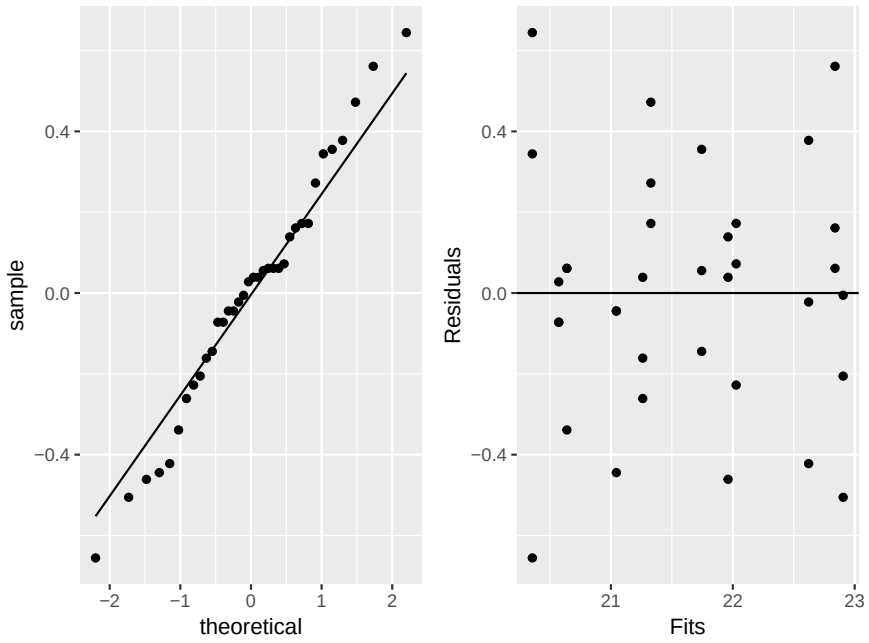
So we will now proceed without the interaction term:

```
fit <- aov(MPG~., data=gasoline)
summary(fit)
```

```
##              Df Sum Sq Mean Sq F value   Pr(>F)
## Gasoline      3 25.405   8.468  80.510 1.89e-14
## Automobile    2  0.527   0.263   2.504   0.0987
## Residuals    30  3.156   0.105
```

let's check the assumptions:

```
df <- data.frame(Residuals=resid(fit),
                 Fits = fitted(fit))
pushViewport(viewport(layout = grid.layout(1, 2)))
print(ggplot(data=df, aes(sample=Residuals)) +
  geom_qq() +geom_qq_line(),
  vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(ggplot(data=df, aes(Fits, Residuals)) +
  geom_point() +
  geom_hline(yintercept = 0),
  vp=viewport(layout.pos.row=1, layout.pos.col=2))
```



the plots look fine, so no problem with the assumptions.

Now let's test for the factors:

Test for Factor Gasoline:

1) Parameters of interest: means of gasoline groups

2) Method of analysis: ANOVA

3) Assumptions of Method: residuals have a normal distribution, groups have equal variance

4) Type I error probability $\alpha = 0.05$

5) Null hypothesis $H_0 : \mu_1 = .. = \mu_4$ (Gasoline groups have the same means)

6) Alternative hypothesis H_a: $\mu_i \neq \mu_j$ (Gasoline groups have different means)

7) p value=0.000

8) 0.000<0.05, there is some evidence of differences in gasoline blends

Test for Factor Automobile is not really needed because this is a blocking variable.

Notice that if we included the interaction the p-value for Automobile was 0.08, without the interaction it is 0.1. One advantage of being able to fit an additive model is that often it makes the conclusions stronger.

```
TukeyHSD(fit)$Gasoline
```

```
##          diff       lwr      upr        p adj
## 2-4 0.6888889 0.2731720 1.104606 5.176968e-04
## 3-4 1.3888889 0.9731720 1.804606 2.402867e-09
## 1-4 2.2666667 1.8509497 2.682384 6.139533e-14
## 3-2 0.7000000 0.2842831 1.115717 4.235469e-04
## 1-2 1.5777778 1.1620609 1.993495 1.293686e-10
## 1-3 0.8777778 0.4620609 1.293495 1.650229e-05
```

so all blends are stat. significantly different, with blend 1 having the highest miles per gallon.

Example: Film Thickness in Semiconductor Production

Chemical vapor deposition is a process used in the semiconductor industry to deposit thin films of silicon dioxide and photoresit on substrates of wafers as they are manufactured. The films must be as thin as possible and have a uniform thickness, which is measured by a process called infrared interference. A process engineer wants to evaluate a low-pressure chemical vapor deposition process that reduces costs and increases productivity. The engineer has set up an experiment to study the effect of chamber temperature and pressure on film thickness.

```
kable.nice(head(filmcoatings))
```

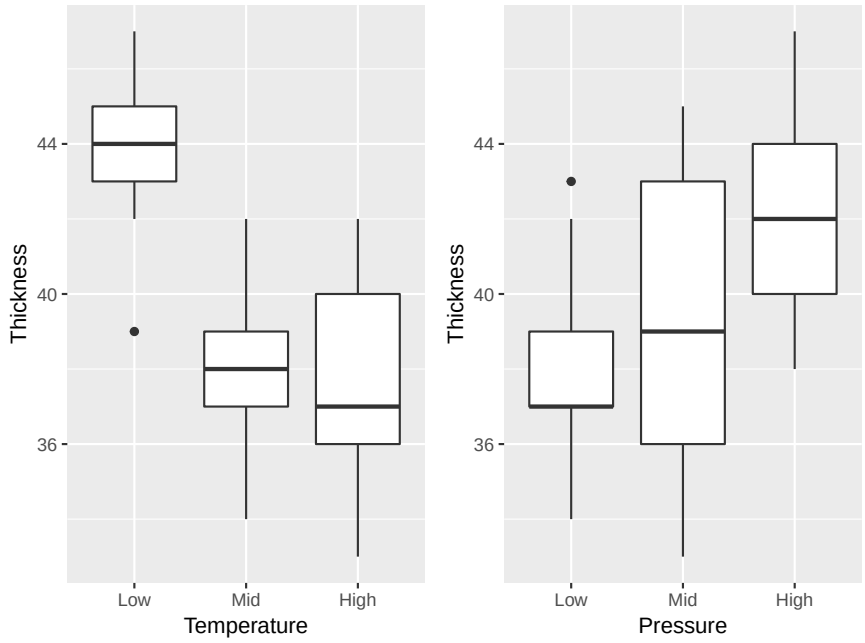
Thickness	Temperature	Pressure
42	Low	Low
43	Low	Low
39	Low	Low
45	Low	Mid
43	Low	Mid
45	Low	Mid

```
table(Temperature, Pressure)
```

```
##             Pressure
## Temperature High Low Mid
##        High    3   3   3
##        Low     3   3   3
##        Mid     3   3   3
```

so again we have balanced design with repeated measurements

```
filmcoatings$Temperature <-
  factor(filmcoatings$Temperature,
  levels=unique(filmcoatings$Temperature),
  ordered=TRUE)
filmcoatings$Pressure <-
  factor(filmcoatings$Pressure,
  levels=unique(filmcoatings$Pressure),
  ordered=TRUE)
pushViewport(viewport(layout = grid.layout(1, 2)))
print(ggplot(data=filmcoatings,
             aes(Temperature, Thickness)) +
        geom_boxplot(),
vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(ggplot(data=filmcoatings,
             aes(Pressure, Thickness)) +
        geom_boxplot(),
 vp=viewport(layout.pos.row=1, layout.pos.col=2))
```



Unlike in the hearing aid or gasoline experiments, here we equally interested in both factors. This type of experiment is called a **factorial design** problem.

For us there is no practical difference between a randomized block design and a factorial design but the distinction can be important in other analyses.

```
sum.tbl <-
  data.frame(
    Temperature=unique(filmcoatings$Temperature),
    n=as.integer(tapply(filmcoatings$Thickness,
             filmcoatings$Temperature,length)),
    Mean=round(tapply(filmcoatings$Thickness,
             filmcoatings$Temperature, mean), 1),
    Sd=round(tapply(filmcoatings$Thickness,
             filmcoatings$Temperature, sd), 2)
)
rownames(sum.tbl) <- NULL
```

Temperature	n	Mean	Sd
Low	9	43.7	2.29
Mid	9	38.0	2.35
High	9	37.7	2.92

```
kable.nice(sum.tbl)
```

```
sum.tbl <-
  data.frame(
    Pressure=unique(filmcoatings$Pressure),
    n=as.integer(tapply(filmcoatings$Thickness,
             filmcoatings$Pressure,length)),
    Mean=round(tapply(filmcoatings$Thickness,
```

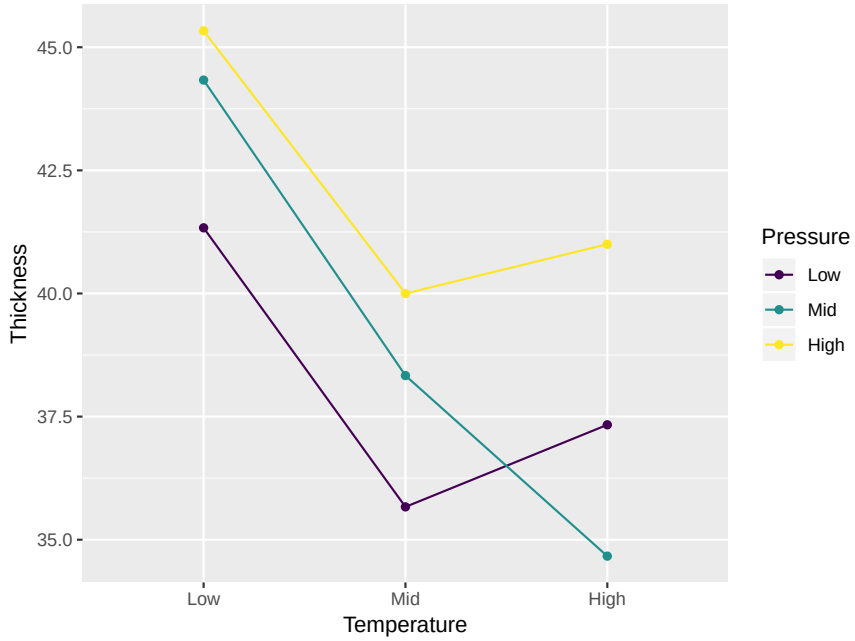
```
            filmcoatings$Pressure, mean), 1),
    Sd=round(tapply(filmcoatings$Thickness,
            filmcoatings$Pressure, sd), 2)
)
rownames(sum.tbl) <- NULL
```

Pressure	n	Mean	Sd
Low	9	38.1	2.85
Mid	9	39.1	4.37
High	9	42.1	2.80

```
kable.nice(sum.tbl)
```

Interaction

```
ggplot(data = filmcoatings,
       aes(Temperature, Thickness ,
           colour = Pressure, group=Pressure)) +
      stat_summary(fun.y=mean, geom="point")+
      stat_summary(fun.y=mean, geom="line")
```

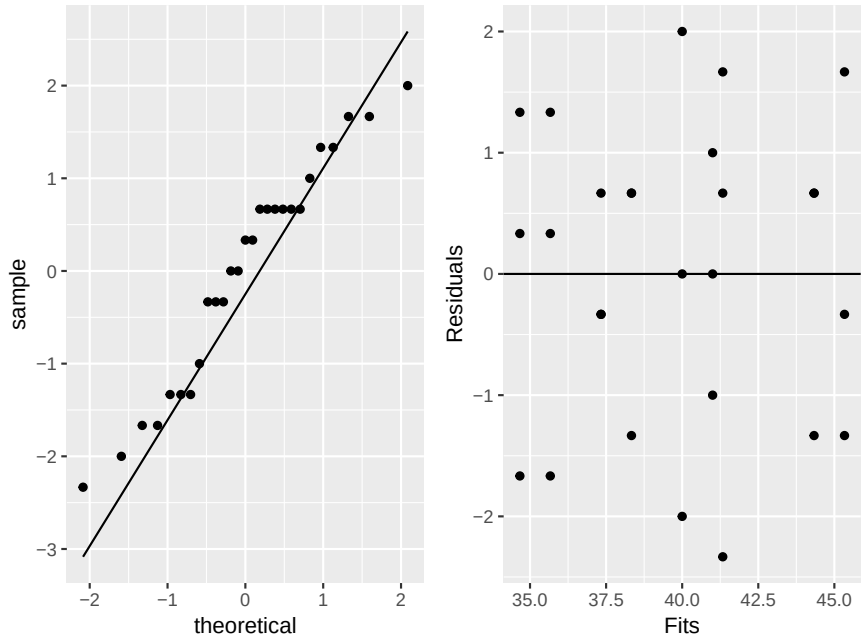


The lines are not all parallel, so there is likely some interaction. Again we have **repeated measurements** (3 per factor-level combination), so we can actually test for this:

```
fit <- aov(Thickness~Temperature*Pressure,
           data=filmcoatings)
```

```
pushViewport(viewport(layout = grid.layout(1, 2)))
df <- data.frame(Residuals=resid(fit),
            Fits = fitted(fit))
print(ggplot(data=df, aes(sample=Residuals)) +
```

```
            geom_qq() + geom_qq_line(),
    vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(ggplot(data=df, aes(Fits, Residuals)) +
            geom_point() +
            geom_hline(yintercept = 0),
    vp=viewport(layout.pos.row=1, layout.pos.col=2))
```



the graphs show that there are no problems with the assumptions.

```
summary(fit)
```

```
##                      Df Sum Sq Mean Sq F value   Pr(>F)
## Temperature           2 204.67  102.33  47.638 6.46e-08
## Pressure              2  78.00   39.00  18.155 4.83e-05
## Temperature:Pressure  4  37.33    9.33   4.345   0.0124
## Residuals            18  38.67    2.15
```

Test for Interaction:

1) Parameters of interest: Interaction

2) Method of analysis: ANOVA

3) Assumptions of Method: residuals have a normal distribution, groups have equal variance

4) Type I error probability $\alpha = 0.05$

5) Null hypothesis H_0 : no interaction

6) Alternative hypothesis H_a: some interaction

7) p value = 0.0124

8) 0.0124<0.05, there is some evidence of interaction

Test for Factor Temperature:

1) Parameters of interest: means of temperature groups

2) Method of analysis: ANOVA

3) Assumptions of Method: residuals have a normal distribution, groups have equal variance

4) Type I error probability $\alpha = 0.05$

5) Null hypothesis $H_0 : \mu_1 = \mu_2 = \mu_3$ (Temperature groups have the same means)

6) Alternative hypothesis H_a: $\mu_i \neq \mu_j$ (Temperature groups have different means)

7) p value = 0.000

8) 0.000 < 0.05, there is some evidence of differences in temperature

Test for Factor Pressure:

1) Parameters of interest: means of pressure groups

2) Method of analysis: ANOVA

3) Assumptions of Method: residuals have a normal distribution, groups have equal variance

4) Type I error probability $\alpha = 0.05$

5) Null hypothesis $H_0 : \mu_1 = \mu_2 = \mu_3$ (Pressure groups have the same means)

6) Alternative hypothesis H_a: $\mu_i \neq \mu_j$ (Pressure groups have different means)

7) p value = 0.000

8) 0.000<0.05, there is some evidence of differences in pressure

Finally, what we need is to find the best combination of pressure and temperature. So what we want is a multiple comparison for Temperature and Pressure (not either of them alone!). Easily done:

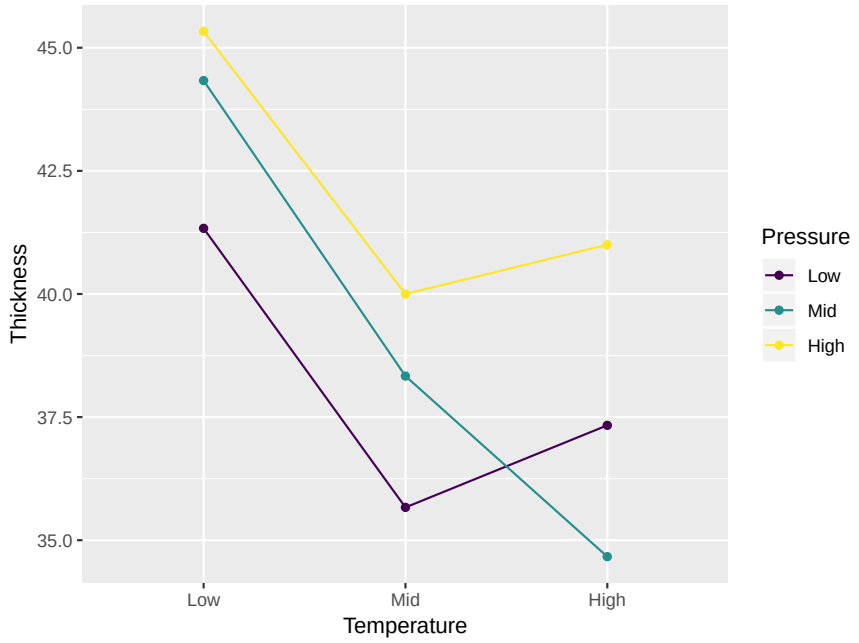
```
out <- TukeyHSD(fit)[[3]]
out
```

```
##                           diff         lwr        upr        p adj
## Mid:Low-Low:Low    -5.6666667  -9.8597499 -1.4735834 4.056833e-03
```

```
## High:Low-Low:Low   -4.0000000  -8.1930833  0.1930833 6.829422e-02
## Low:Mid-Low:Low     3.0000000  -1.1930833  7.1930833 2.908226e-01
## Mid:Mid-Low:Low    -3.0000000  -7.1930833  1.1930833 2.908226e-01
## High:Mid-Low:Low   -6.6666667 -10.8597499 -2.4735834 7.270485e-04
## Low:High-Low:Low    4.0000000  -0.1930833  8.1930833 6.829422e-02
## Mid:High-Low:Low   -1.3333333  -5.5264166  2.8597499 9.642909e-01
## High:High-Low:Low  -0.3333333  -4.5264166  3.8597499 9.999982e-01
## High:Low-Mid:Low    1.6666667  -2.5264166  5.8597499 8.866164e-01
## Low:Mid-Mid:Low     8.6666667   4.4735834 12.8597499 2.811245e-05
## Mid:Mid-Mid:Low     2.6666667  -1.5264166  6.8597499 4.291123e-01
## High:Mid-Mid:Low   -1.0000000  -5.1930833  3.1930833 9.938696e-01
## Low:High-Mid:Low    9.6666667   5.4735834 13.8597499 6.251132e-06
## Mid:High-Mid:Low    4.3333333   0.1402501  8.5264166 3.970300e-02
## High:High-Mid:Low   5.3333333   1.1402501  9.5264166 7.227536e-03
## Low:Mid-High:Low    7.0000000   2.8069167 11.1930833 4.142532e-04
## Mid:Mid-High:Low    1.0000000  -3.1930833  5.1930833 9.938696e-01
## High:Mid-High:Low  -2.6666667  -6.8597499  1.5264166 4.291123e-01
## Low:High-High:Low   8.0000000   3.8069167 12.1930833 8.029539e-05
## Mid:High-High:Low   2.6666667  -1.5264166  6.8597499 4.291123e-01
## High:High-High:Low  3.6666667  -0.5264166  7.8597499 1.147340e-01
## Mid:Mid-Low:Mid    -6.0000000 -10.1930833 -1.8069167 2.278849e-03
## High:Mid-Low:Mid   -9.6666667 -13.8597499 -5.4735834 6.251132e-06
## Low:High-Low:Mid    1.0000000  -3.1930833  5.1930833 9.938696e-01
## Mid:High-Low:Mid   -4.3333333  -8.5264166 -0.1402501 3.970300e-02
## High:High-Low:Mid  -3.3333333  -7.5264166  0.8597499 1.866153e-01
## High:Mid-Mid:Mid   -3.6666667  -7.8597499  0.5264166 1.147340e-01
## Low:High-Mid:Mid    7.0000000   2.8069167 11.1930833 4.142532e-04
## Mid:High-Mid:Mid    1.6666667  -2.5264166  5.8597499 8.866164e-01
## High:High-Mid:Mid   2.6666667  -1.5264166  6.8597499 4.291123e-01
## Low:High-High:Mid  10.6666667   6.4735834 14.8597499 1.512789e-06
## Mid:High-High:Mid   5.3333333   1.1402501  9.5264166 7.227536e-03
## High:High-High:Mid  6.3333333   2.1402501 10.5264166 1.284046e-03
## Mid:High-Low:High  -5.3333333  -9.5264166 -1.1402501 7.227536e-03
## High:High-Low:High -4.3333333  -8.5264166 -0.1402501 3.970300e-02
## High:High-Mid:High  1.0000000  -3.1930833  5.1930833 9.938696e-01
```

This is bit hard to read. Recall that we are only interested in small values of Thickness. Let's redo the interaction plot:

```
ggplot(data = filmcoatings,
       aes(Temperature, Thickness ,
           colour = Pressure, group=Pressure)) +
      stat_summary(fun.y=mean, geom="point")+
      stat_summary(fun.y=mean, geom="line")
```



so we see that that the best combination is Temperature=High, Pressure=Mid.

The next-best is Temperature=Mid, Pressure=Low. What does Tukey say about the comparison of these?

```
out["High:Mid-Mid:Low", ]
```

```
##       diff        lwr        upr      p adj
## -1.0000000 -5.1930833  3.1930833  0.9938696
```

so they are NOT statistically significant.

Let's check the next couple of combinations:

```
out["High:Mid-High:Low", ]
```

```
##       diff        lwr        upr      p adj
## -2.6666667 -6.8597499  1.5264166  0.4291123
```

```
out["High:Mid-Mid:Mid", ]
```

```
##       diff        lwr        upr      p adj
## -3.6666667 -7.8597499  0.5264166  0.1147340
```

```
out["Mid:High-High:Mid", ]
```

```
##        diff         lwr         upr       p adj
## 5.333333333 1.140250075 9.526416591 0.007227536
```

and now we have a stat. significant difference.

Notice that in the last one we have to change the order, because Tukey did as well.

So either of the four combinations (High Mid, Mid Low, High Low or Mid Mid), at least not at these sample sizes.

A simple idea for solving this problem seems to be this one:

1. find the best temperature:

```
sort(round(tapply(Thickness, Temperature, mean), 1))
```

```
## High  Mid  Low
## 37.7 38.0 43.7
```

so Temperature=High is best

2. find the best pressure:

```
sort(round(tapply(Thickness, Pressure, mean), 1))
```

```
##  Low  Mid High
## 38.1 39.1 42.1
```

so Pressure=Low is best

3. take the combination: Pressure=Low, Temperature=High is best! Except it is not: we saw before that Pressure=Mid, Temperature=High is best.

This simple idea does not work because of the presence of interaction.

Example: Water Quality and Mining

The effects of mining and rock type on water quality.

```
kable.nice(head(mines))
```

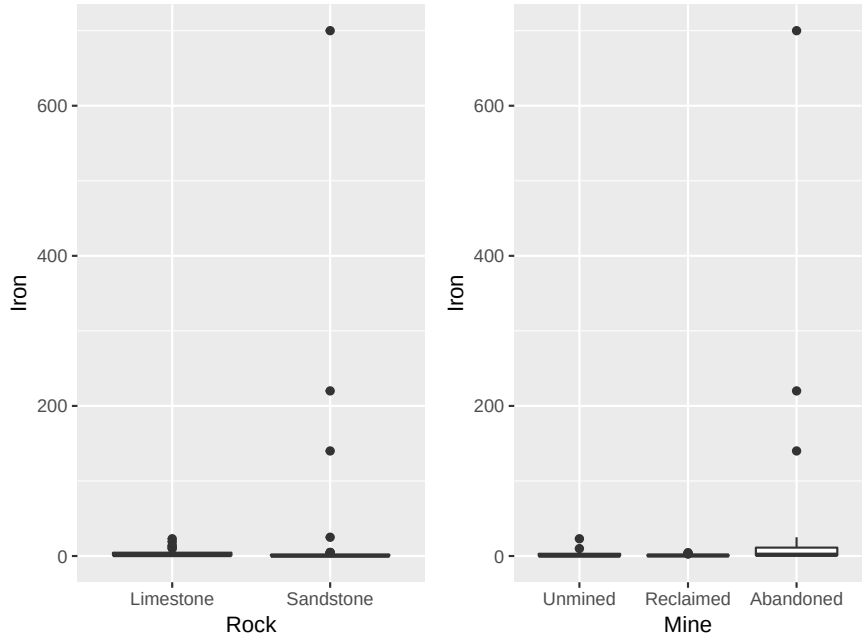
Rock	Mine	Iron
Sandstone	Unmined	0.20
Sandstone	Unmined	0.25
Sandstone	Unmined	0.04
Sandstone	Unmined	0.06
Sandstone	Unmined	1.20
Sandstone	Unmined	0.30

```
table(Rock, Mine)
```

```
##             Mine
## Rock         Abandoned Reclaimed Unmined
##   Limestone         13        13      13
##   Sandstone         13        13      13
```

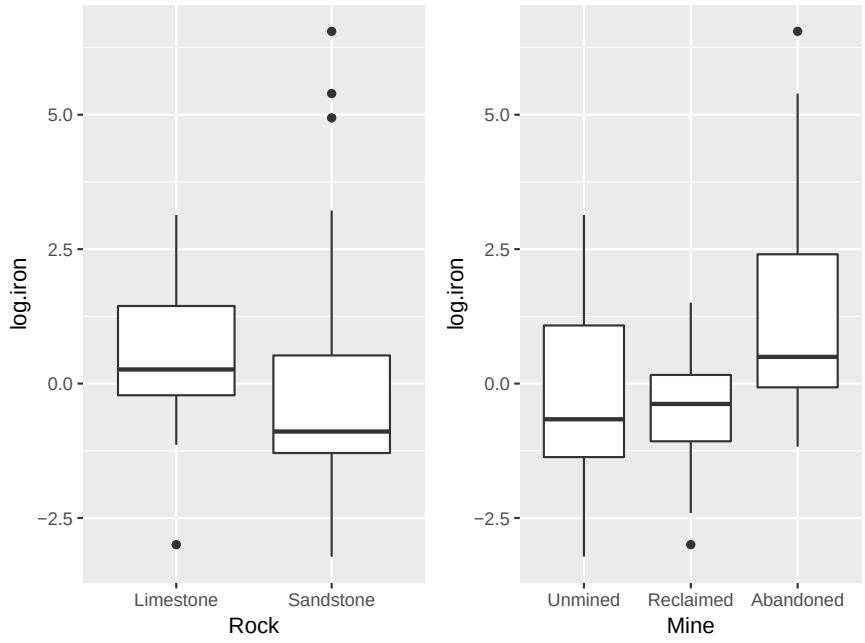
```
mines$Mine <- factor(mines$Mine,
        levels = c("Unmined", "Reclaimed", "Abandoned"),
        ordered = TRUE)
```

```
pushViewport(viewport(layout = grid.layout(1, 2)))
print(ggplot(data=mines, aes(Rock, Iron)) +
        geom_boxplot(),
vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(ggplot(mines, aes(Mine, Iron)) +
        geom_boxplot(),
 vp=viewport(layout.pos.row=1, layout.pos.col=2))
```



We have a clear problem with outliers (aka the normal assumption), so we try the log transform:

```
mines$log.iron <- log(mines$Iron)
pushViewport(viewport(layout = grid.layout(1, 2)))
print(ggplot(data=mines, aes(Rock, log.iron)) +
        geom_boxplot(),
vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(ggplot(mines, aes(Mine, log.iron)) +
        geom_boxplot(),
 vp=viewport(layout.pos.row=1, layout.pos.col=2))
```



This has solved the problem, so the analysis will be based on log.iron.

Summary Statistics

Because we use a transformation we will base the tables on Median and IQR

```
sum.tbl <-
  data.frame(
    Mine=levels(mines$Mine),
    n=as.integer(tapply(mines$Iron,
             mines$Mine,length)),
    Median=round(tapply(mines$Iron,
             mines$Mine, median), 1),
    IQR=round(tapply(mines$Iron,
             mines$Mine, IQR), 2)
)
rownames(sum.tbl) <- NULL
```

Mine	n	Median	IQR
Unmined	26	0.5	2.72
Reclaimed	26	0.7	0.83
Abandoned	26	1.6	10.24

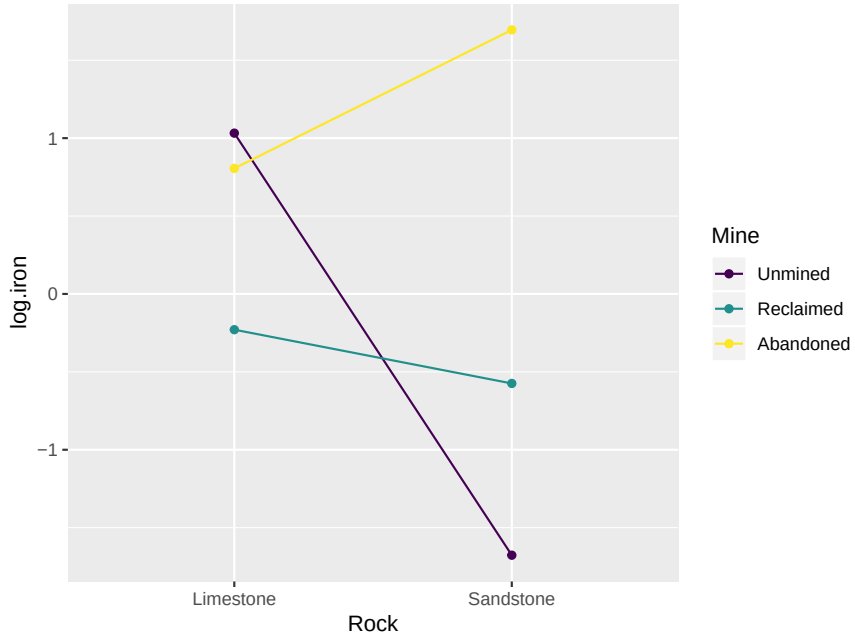
```
kable.nice(sum.tbl)
```

Note that the IQR's are very different. This is because this data set has a lot of outliers which still effect the IQR.

Interaction

```
ggplot(data = mines,
       aes(Rock , log.iron,
```

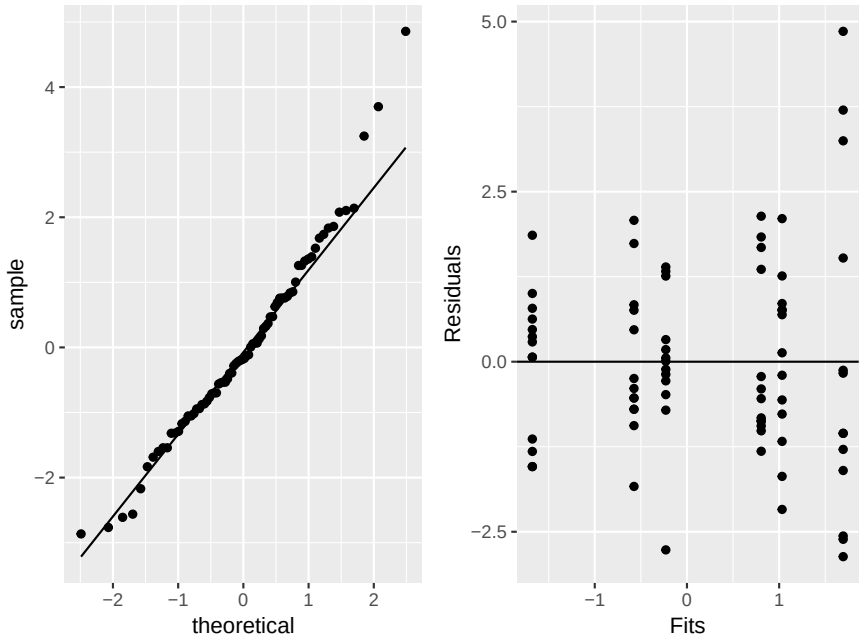
```
          colour = Mine, group=Mine)) +
      stat_summary(fun.y=mean, geom="point")+
      stat_summary(fun.y=mean, geom="line")
```



There seems to be some interaction. To confirm this test for it:

```
fit <- aov(log.iron~Rock*Mine, data=mines)
```

```
pushViewport(viewport(layout = grid.layout(1, 2)))
df <- data.frame(Residuals=resid(fit),
            Fits = fitted(fit))
print(ggplot(data=df, aes(sample=Residuals)) +
          geom_qq() + geom_qq_line(),
    vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(ggplot(data=df, aes(Fits, Residuals)) +
            geom_point() +
            geom_hline(yintercept = 0),
    vp=viewport(layout.pos.row=1, layout.pos.col=2))
```



assumptions are ok (after log transform!)

```
summary(fit)
```

```
##             Df Sum Sq Mean Sq F value   Pr(>F)
## Rock         1  10.15  10.154   4.629 0.034794
## Mine         2  45.13  22.566  10.287 0.000118
## Rock:Mine    2  43.45  21.727   9.904 0.000159
## Residuals   72 157.95   2.194
```

Test for Interaction:

1) Parameters of interest: Interaction

2) Method of analysis: ANOVA

3) Assumptions of Method: residuals have a normal distribution, groups have equal variance

4) Type I error probability $\alpha = 0.05$

5) Null hypothesis H_0 : no interaction

6) Alternative hypothesis H_a: some interaction

7) p value = 0.000

8) 0.000<0.05, there is some evidence of interaction
   Check the assumptions of ANOVA: both plots look ok

Test for Factor Rock:

1) Parameters of interest: means of pressure groups

2) Method of analysis: ANOVA

3) Assumptions of Method: residuals have a normal distribution, groups have equal variance

4) Type I error probability $\alpha = 0.05$

5) Null hypothesis H_0 : $\mu_1 = \mu_2$ (Rock groups have the same means)

6) Alternative hypothesis H_a: $\mu_1 \neq \mu_2$ (Rock groups have different means)

7) p value = 0.035

8) 0.035<0.05, there is some evidence of differences in Rock types.

Test for Factor Mine:

1) Parameters of interest: means of pressure groups

2) Method of analysis: ANOVA

3) Assumptions of Method: residuals have a normal distribution, groups have equal variance

4) Type I error probability $\alpha = 0.05$

5) Null hypothesis H_0 : $\mu_1 = \mu_2 = \mu_3$ (Mine groups have the same means)

6) Alternative hypothesis H_a: $\mu_i \neq \mu_j$ (Mine groups have different means)

7) p value = 0.000

8) 0.000<0.05, there is some evidence of differences in Mine types

Multiple Comparison The main interest is in mines, so

```
TukeyHSD(fit)$Mine
```

```
##                             diff        lwr       upr        p adj
## Reclaimed-Unmined   -0.07955136 -1.0626180 0.9035152 0.9795435606
## Abandoned-Unmined    1.57238141  0.5893148 2.5554480 0.0007902888
## Abandoned-Reclaimed  1.65193277  0.6688662 2.6349994 0.0004103178
```

Interpretation: There is a stat. signif. difference between the mean iron content of abandoned mines and the others. The difference between unmined and reclaimed mines is not stat. sign, at least not at these sample sizes.

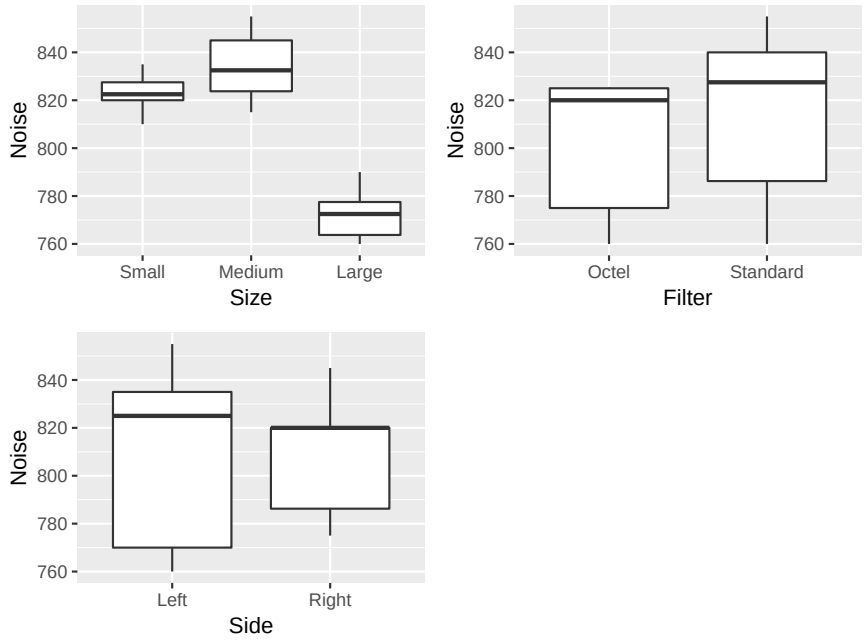
Example: Air Filters and Noise

The data are from a statement by Texaco, Inc. to the Air and Water Pollution Subcommittee of the Senate Public Works Committee on June 26, 1973. Mr. John McKinley, President of Texaco, cited the Octel filter, developed by Associated Octel Company as effective in reducing pollution. However, questions had been raised about the effects of pollution filters on aspects of vehicle performance, including noise levels. He referred to data presented in the datafile associated with this story as evidence that the Octel filter was was at least as good as a standard silencer in controlling vehicle noise levels.

```
kable.nice(head(airfilters))
```

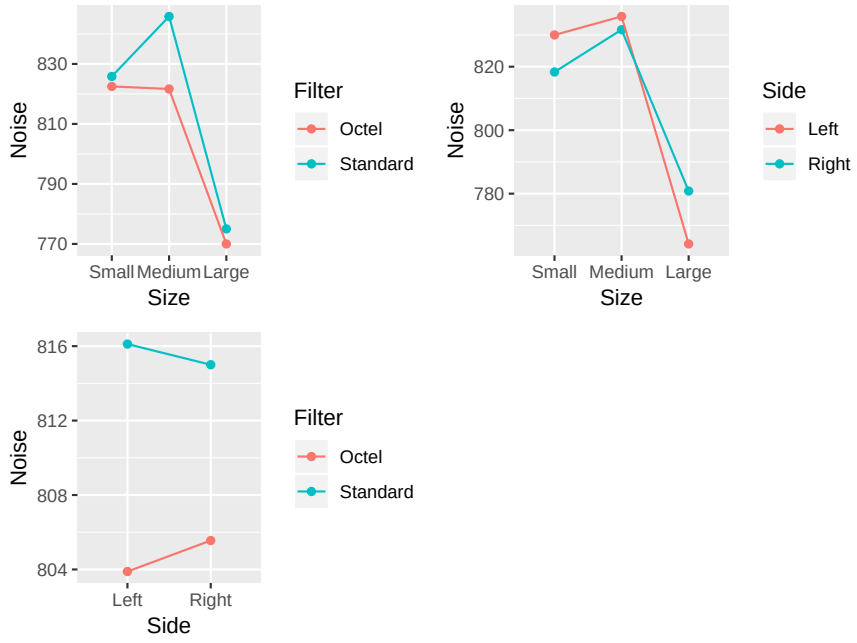
Noise	Size	Filter	Side
810	Small	Standard	Right
820	Small	Standard	Right
820	Small	Standard	Right
840	Medium	Standard	Right
840	Medium	Standard	Right
845	Medium	Standard	Right

```
airfilters$Size <- factor(airfilters$Size,
                  levels = unique(airfilters$Size),
                  ordered = TRUE)
plt1 <- ggplot(data=airfilters, aes(Size, Noise)) +
  geom_boxplot()
plt2 <- ggplot(data=airfilters, aes(Filter, Noise)) +
  geom_boxplot()
plt3 <- ggplot(data=airfilters, aes(Side, Noise)) +
  geom_boxplot()
pushViewport(viewport(layout = grid.layout(2, 2)))
print(plt1,
  vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(plt2,
  vp=viewport(layout.pos.row=1, layout.pos.col=2))
print(plt3,
  vp=viewport(layout.pos.row=2, layout.pos.col=1))
```



it seems large cars are more quiet. Not much of an effect due to either side or filter.

```
plt1 <- ggplot(data = airfilters,
       aes(Size , Noise,
           colour = Filter, group=Filter)) +
      stat_summary(fun.y=mean, geom="point")+
      stat_summary(fun.y=mean, geom="line")
plt2 <- ggplot(data = airfilters,
       aes(Size , Noise,
           colour = Side, group=Side)) +
      stat_summary(fun.y=mean, geom="point")+
      stat_summary(fun.y=mean, geom="line")
plt3 <- ggplot(data = airfilters,
       aes(Side , Noise,
           colour = Filter, group=Filter)) +
      stat_summary(fun.y=mean, geom="point")+
      stat_summary(fun.y=mean, geom="line")
pushViewport(viewport(layout = grid.layout(2, 2)))
print(plt1,
  vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(plt2,
  vp=viewport(layout.pos.row=1, layout.pos.col=2))
print(plt3,
  vp=viewport(layout.pos.row=2, layout.pos.col=1))
```



a possible interaction between Filter and Side

```
fit <- aov(Noise~.^3, data=airfilters)
summary(fit)
```

```
##                  Df Sum Sq Mean Sq F value   Pr(>F)
## Size              2  26051   13026 893.190  < 2e-16
## Filter            1   1056    1056  72.429 1.04e-08
## Side              1      1       1   0.048 0.829104
## Size:Filter       2    804     402  27.571 6.05e-07
## Size:Side         2   1293     647  44.333 8.73e-09
## Filter:Side       1     17      17   1.190 0.286067
## Size:Filter:Side  2    301     151  10.333 0.000579
## Residuals        24    350      15
```

the three-way interaction is significant (p=0.000579), so we can not simplify this model.

The main question here is whether there is a difference between the filters, and the answer is yeas (p=0.000). Because Filter has only two values a multiple comparison is not necessary.

Multiple Linear Regression

Example: House Prices

Prices of residencies located 30 miles south of a large metropolitan area with several possible predictor variables.

Notice the 1.7 baths!

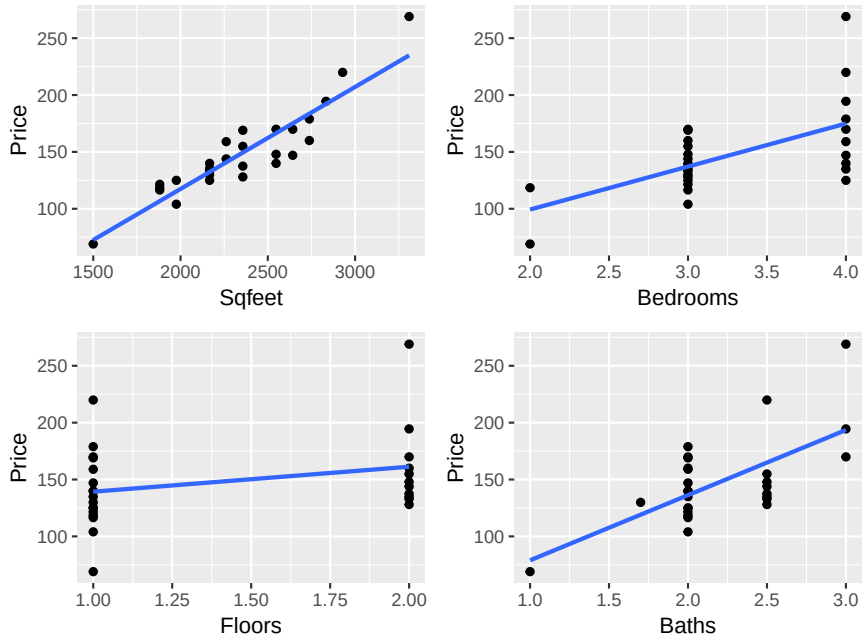
```
kable.nice(houseprice)
```

	Price	Sqfeet	Floors	Bedrooms	Baths
1	69.0	1500.000	1	2	1.0
3	118.5	1880.952	1	2	2.0
4	104.0	1976.190	1	3	2.0
5	116.5	1880.952	1	3	2.0
6	121.5	1880.952	1	3	2.0
7	125.0	1976.190	1	3	2.0
8	128.0	2357.143	2	3	2.5
9	129.9	2166.667	1	3	1.7
10	133.0	2166.667	2	3	2.5
11	135.0	2166.667	2	3	2.5
12	137.5	2357.143	2	3	2.5
13	139.9	2166.667	1	3	2.0
14	143.9	2261.905	2	3	2.5
15	147.9	2547.619	2	3	2.5
16	154.9	2357.143	2	3	2.5
17	160.0	2738.095	2	3	2.0
18	169.0	2357.143	1	3	2.0
19	169.9	2642.857	1	3	2.0
20	125.0	2166.667	1	4	2.0
21	134.9	2166.667	1	4	2.0
22	139.9	2547.619	1	4	2.0
23	147.0	2642.857	1	4	2.0
24	159.0	2261.905	1	4	2.0
25	169.9	2547.619	2	4	3.0
26	178.9	2738.095	1	4	2.0
27	194.5	2833.333	2	4	3.0
28	219.9	2928.571	1	4	2.5
29	269.0	3309.524	2	4	3.0

Let's go through the list of predictors one by one:

```
pushViewport(viewport(layout = grid.layout(2, 2)))
pos <- expand.grid(1:2, 1:2)
for(i in 1:4) {
  plt <-
    ggplot(data=houseprice,
           aes_(x = as.name(names(houseprice)[i+1]),
                y = as.name(names(houseprice)[1]))) +
```

```
    geom_point() +
    geom_smooth(method = "lm", se=FALSE)
  print(plt,
  vp=viewport(layout.pos.row=pos[i, 1],
              layout.pos.col=pos[i, 2]))
  cat("Price and ", colnames(houseprice)[i+1],
      " ", round(cor(houseprice$Price, houseprice[, i+1]), 3), "\n")
}
```




```
## Price and  Sqfeet   0.915
## Price and  Floors   0.291
## Price and  Bedrooms   0.605
## Price and  Baths   0.653
```

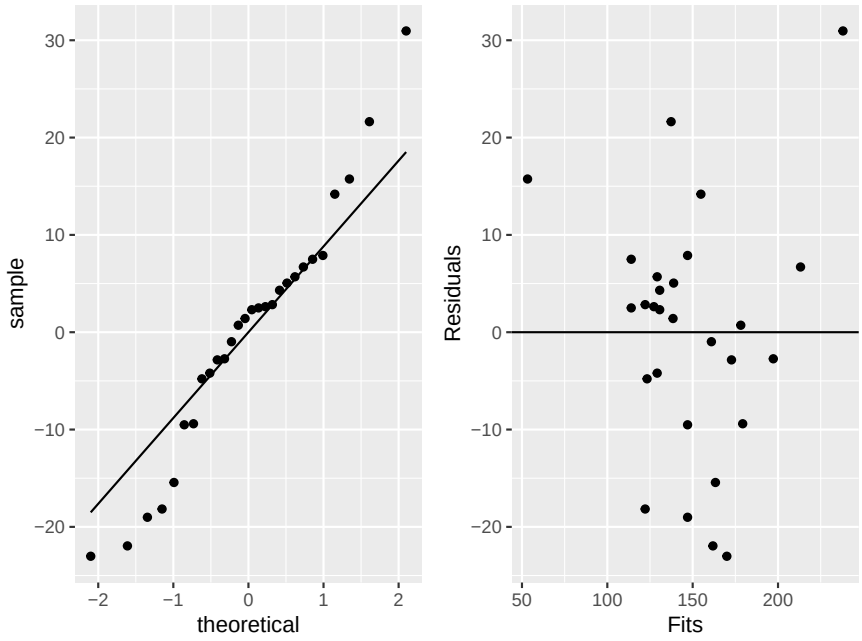
```
fit <- lm(Price~., data=houseprice)
summary(fit)
```

```
##
## Call:
## lm(formula = Price ~ ., data = houseprice)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -23.018  -5.943   1.860   5.947  30.955
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -67.61984   17.70818  -3.819 0.000882
## Sqfeet        0.08571    0.01076   7.966 4.62e-08
```

```
## Floors      -26.49306    9.48952  -2.792 0.010363
## Bedrooms     -9.28622    6.82985  -1.360 0.187121
## Baths        37.38067   12.26436   3.048 0.005709
##
## Residual standard error: 13.71 on 23 degrees of freedom
## Multiple R-squared:  0.8862, Adjusted R-squared:  0.8665
## F-statistic:  44.8 on 4 and 23 DF,  p-value: 1.558e-10
```

For the assumptions there is nothing new, as before we need to check the residual vs. fits plot and the normal plot of residuals:

```
pushViewport(viewport(layout = grid.layout(1, 2)))
df <- data.frame(Residuals=resid(fit),
            Fits = fitted(fit))
print(ggplot(data=df, aes(sample=Residuals)) +
          geom_qq() + geom_qq_line(),
    vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(ggplot(data=df, aes(Fits, Residuals)) +
            geom_point() +
            geom_hline(yintercept = 0),
    vp=viewport(layout.pos.row=1, layout.pos.col=2))
```



This appears to be a good model and the assumptions of normally distributed residuals with equal variance appears to be o.k.

Except,

Notice that there is something very strange about this model!

Let's have a look at the correlations between the predictors:

```
round(cor(houseprice[, -1]), 3)
```

```
##          Sqfeet Floors Bedrooms Baths
## Sqfeet    1.000  0.370    0.652 0.628
## Floors    0.370  1.000   -0.018 0.743
## Bedrooms  0.652 -0.018    1.000 0.415
## Baths     0.628  0.743    0.415 1.000
```

The highest correlation between predictors is r=0.743 (Floors-Baths)

As in the case of polynomial regression, highly correlated predictors are a potential problem. We will look at a solution called *principle components* at some point. Here we are ok.

Variable Selection

We have previously talked about the fact that we want our models to be as simple as possible. Often that means a model with as few predictors as possible. So the question becomes:

Can we eliminate any of our predictors without making the model (stat. signif.) worse?

There are several things one can think of:

Choose based on R^2

but we already know this will always lead to the model with all predictors, for the same reason that a cubic model always has an R^2 at least as high as the quadratic model.

Note:

Price by Sqfeet, Floors and Bedrooms: R^2=80.1%
Price by Floors, Bedrooms and Baths: R^2=68.4%
Price by Sqfeet, Bedrooms and Baths: R^2=83.5%
Price by Sqfeet, Floors, Bedrooms and Baths: R^2=88.2%
so model with all 4 has a higher R^2 than any of the models with just 3, **but this will always be so, even if one of the predictors is completely useless**.

Choose based on Hypothesis Tests

in the summary(fit) above we see that p_value of Bedrooms = 0.187121 > 0.05, so eliminate Bedrooms.

This sounds like a good idea AND IT IS WIDELY USED IN REAL LIFE, but it turns out to be a bad one ! The reason why is bit hard to explain, though.

Use nested models test

```
fit.without.bedrooms <- lm(Price~.-Bedrooms,
                           data=houseprice)
anova(fit, fit.without.bedrooms)
```

```
## Analysis of Variance Table
##
```

```
## Model 1: Price ~ Sqfeet + Floors + Bedrooms + Baths
## Model 2: Price ~ (Sqfeet + Floors + Bedrooms + Baths) - Bedrooms
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1     23 4321.9
## 2     24 4669.2 -1   -347.38 1.8487 0.1871
```

Again, this sounds like a good idea AND AGAIN IT IS WIDELY USED IN REAL LIFE, but it turns out to be a dangerous one! To start, if we have several predictors we might want to eliminate, we immediately face the issue of *simultaneous inference*.

There are several methods in wide use that are essentially based on this idea, such as *forward selection*, *backward selection* and *stepwise regression*. These are sometimes unavoidable but need to be done with great care!

What we need is new idea:

Best Subset Regression and Mallow's C_p

We will find ALL possible models and calculate Mallow's C_p statistic for each. The model with the lowest C_p is best.

```
library(leaps)
out <- leaps(houseprice[, -1], houseprice$Price,
      method = "Cp", nbest=1)
out
```

```
## $which
##       1     2     3     4
## 1 TRUE FALSE FALSE FALSE
## 2 TRUE FALSE FALSE  TRUE
## 3 TRUE  TRUE FALSE  TRUE
## 4 TRUE  TRUE  TRUE  TRUE
##
## $label
## [1] "(Intercept)" "1"           "2"           "3"           "4"
##
## $size
## [1] 2 3 4 5
##
## $Cp
## [1] 8.834171 8.812489 4.848657 5.000000
```

```
colnames(houseprice)[2:5][out$which[seq_along(out$Cp)[out$Cp==min(out$Cp)], ]]
```

```
## [1] "Sqfeet" "Floors" "Baths"
```

so the best model uses Sqfeet, Floors and Baths.

To find the model we rerun lm, now without Bedrooms:

```
summary(fit)
```

```
##
## Call:
## lm(formula = Price ~ ., data = houseprice)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -23.018  -5.943   1.860   5.947  30.955
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -67.61984   17.70818  -3.819 0.000882
## Sqfeet        0.08571    0.01076   7.966 4.62e-08
## Floors      -26.49306    9.48952  -2.792 0.010363
## Bedrooms     -9.28622    6.82985  -1.360 0.187121
## Baths        37.38067   12.26436   3.048 0.005709
##
## Residual standard error: 13.71 on 23 degrees of freedom
## Multiple R-squared:  0.8862, Adjusted R-squared:  0.8665
## F-statistic:  44.8 on 4 and 23 DF,  p-value: 1.558e-10
```

Note that the model with all four predictors has C_p=5.0. But C_p is a **statistic**, its exact value depends on the sample. So is the model with Sqfeet, Floors and Baths **statistically significantly** better than the model with all four predictors? We would need a hypothesis test to answer this question but this is not part of our course.

Prediction

Prediction works just as it did for simple regression. Say we want to find a 90% interval estimate for a house that has 2000 sqfeet, one floor and two baths. Then

```
predict(fit.without.bedrooms,
        newdata=data.frame(Sqfeet=2000,
                           Floors=1,
                           Bedrooms=0,
                           Baths=2),
        interval="prediction", level=0.9)
```

```
##        fit      lwr     upr
## 1 122.9797 98.07234 147.887
```

Example: Air Pollution and Mortality

The dependent variable for analysis is age adjusted mortality (called "Mortality"). The data include variables measuring demographic characteristics of the cities, variables measuring

climate characteristics, and variables recording the pollution potential of three different air pollutants.

```
x <- airpollution
rownames(x) <- NULL
kable.nice(head(airpollution[, 1:5], 3))
```

	Mortality	JanTemp	JulyTemp	RelHum	Rain
AkronOH	921.87	27	71	59	36
Albany-Schenectady-TroyNY	997.87	23	72	57	35
AllentownBethlehemPA-NJ	962.35	29	74	54	44

```
kable.nice(head(x[, 6:10], 3))
```

Education	PopDensity	NonWhite	WhiteCollar	Pop
11.4	3243	8.8	42.6	660328
11.0	4281	3.5	50.7	835880
9.8	4260	0.8	39.4	635481

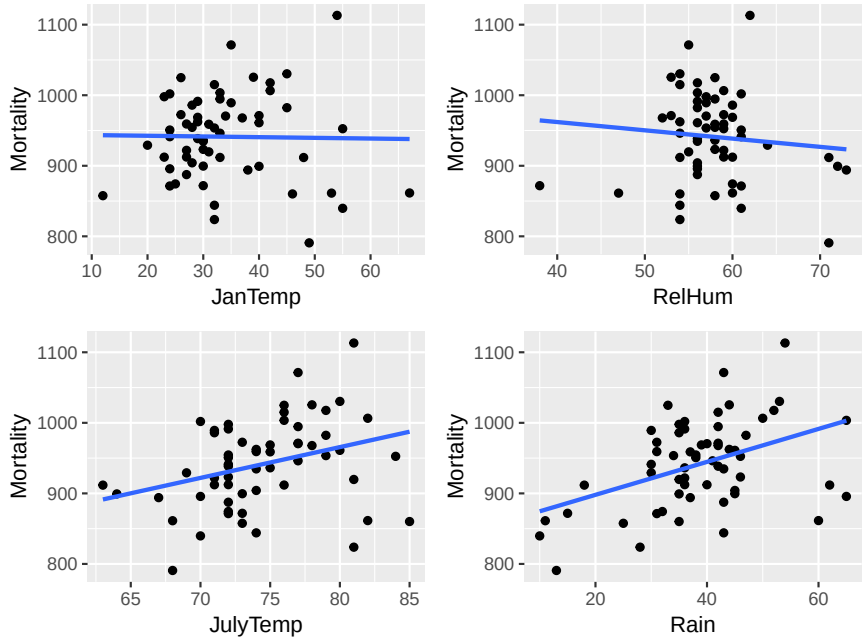
```
kable.nice(head(x[, 11:16], 3))
```

Pop.House	Income	HCPot	NOxPot	SO2Pot	NOx
3.34	29560	21	15	59	15
3.14	31458	8	10	39	10
3.21	31856	6	6	33	6

we want to look at the scatterplots and the correlations. There are 15 predictors, so there are 15 graphs and correlations.

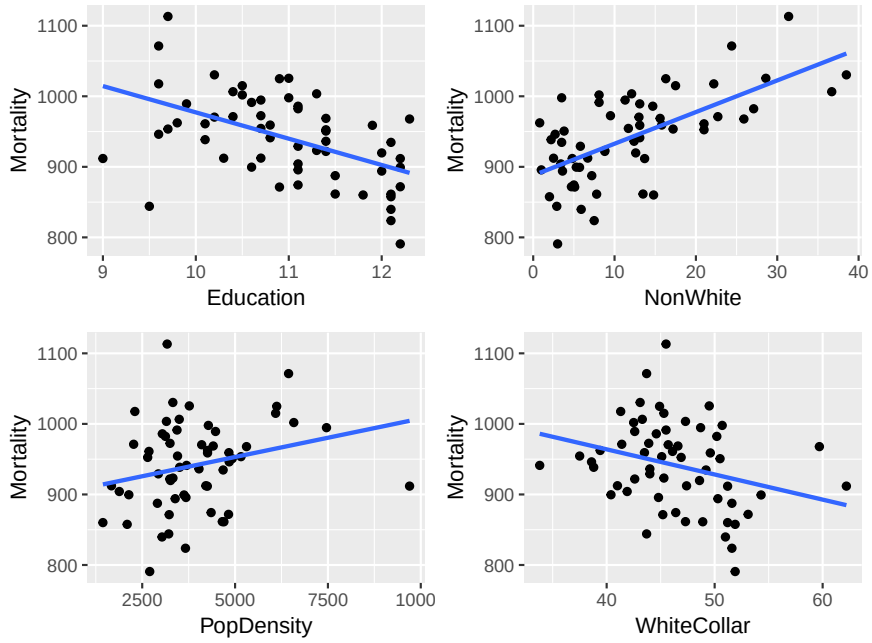
```
pos <- expand.grid(1:2, 1:2)
pushViewport(viewport(layout = grid.layout(2, 2)))
for(i in 1:4) {
   plt <-
       ggplot(data=airpollution,
           aes_(x = as.name(names(airpollution)[i+1]),
                y = as.name(names(airpollution)[1]))) +
    geom_point() +
    geom_smooth(method = "lm", se=FALSE)
    print(plt,
    vp=viewport(layout.pos.row=pos[i, 1],
              layout.pos.col=pos[i, 2]))
    cat("Mortality and ", colnames(airpollution)[i+1],
      " ", round(cor(airpollution$Mortality,
```

```
                    airpollution[, i+1]), 3), "\n")
}
```



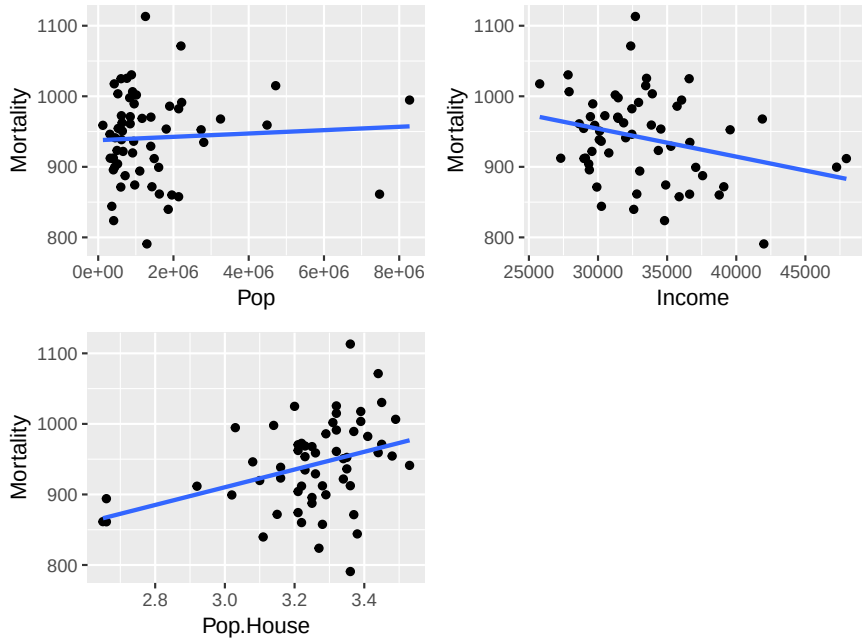

```
## Mortality and  JanTemp   -0.016
## Mortality and  JulyTemp   0.322
## Mortality and  RelHum   -0.101
## Mortality and  Rain   0.433
```

```
pushViewport(viewport(layout = grid.layout(2, 2)))
for(i in 5:8) {
   plt <-
       ggplot(data=airpollution,
           aes_(x = as.name(names(airpollution)[i+1]),
                y = as.name(names(airpollution)[1]))) +
    geom_point() +
    geom_smooth(method = "lm", se=FALSE)
    print(plt,
    vp=viewport(layout.pos.row=pos[i-4, 1],
              layout.pos.col=pos[i-4, 2]))
    cat("Mortality and ", colnames(airpollution)[i+1],
      " ", round(cor(airpollution$Mortality,
                     airpollution[, i+1]), 3), "\n")
}
```



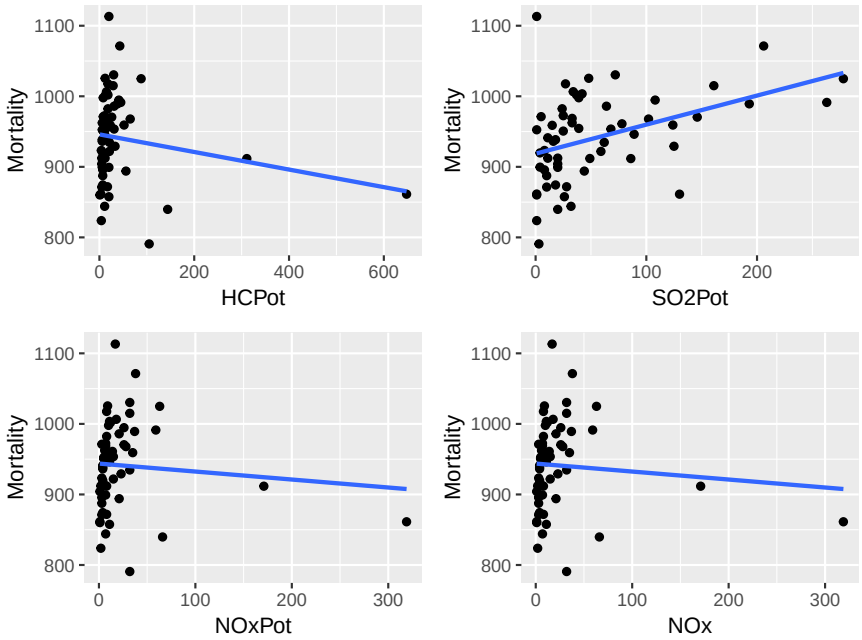

```
## Mortality and  Education   -0.508
## Mortality and  PopDensity   0.252
## Mortality and  NonWhite   0.647
## Mortality and  WhiteCollar   -0.289
```

```
pushViewport(viewport(layout = grid.layout(2, 2)))
for(i in 9:11) {
   plt <-
      ggplot(data=airpollution,
           aes_(x = as.name(names(airpollution)[i+1]),
                y = as.name(names(airpollution)[1]))) +
    geom_point() +
    geom_smooth(method = "lm", se=FALSE)
    print(plt,
    vp=viewport(layout.pos.row=pos[i-8, 1],
              layout.pos.col=pos[i-8, 2]))
    cat("Mortality and ", colnames(airpollution)[i+1],
      " ", round(cor(airpollution$Mortality,
                     airpollution[, i+1]), 3), "\n")
}
```




```
## Mortality and  Pop   0.059
## Mortality and  Pop.House   0.368
## Mortality and  Income   -0.283
```

```
pushViewport(viewport(layout = grid.layout(2, 2)))
for(i in 12:15) {
   plt <-
      ggplot(data=airpollution,
           aes_(x = as.name(names(airpollution)[i+1]),
                y = as.name(names(airpollution)[1]))) +
    geom_point() +
    geom_smooth(method = "lm", se=FALSE)
    print(plt,
    vp=viewport(layout.pos.row=pos[i-11, 1],
              layout.pos.col=pos[i-11, 2]))
    cat("Mortality and ", colnames(airpollution)[i+1],
      " ", round(cor(airpollution$Mortality,
                     airpollution[, i+1]), 3), "\n")
}
```



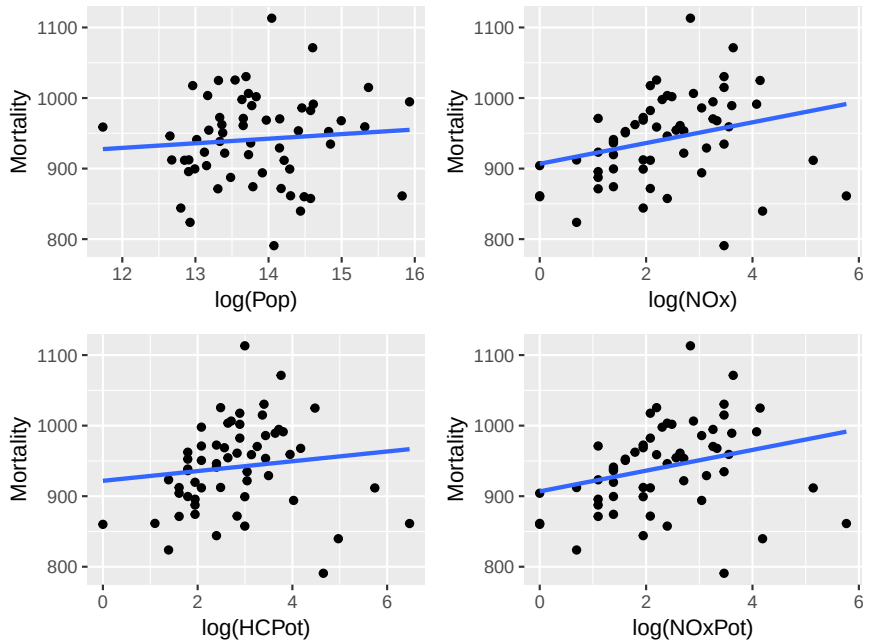

```
## Mortality and  HCPot   -0.185
## Mortality and  NOxPot   -0.085
## Mortality and  SO2Pot   0.419
## Mortality and  NOx   -0.085
```

There are problems with four predictors (Pop, HCPot, NOx, and NOxPot). Let's try the log transform and check again for those predictors:

The easiest way to do this is to make a new matrix:

```
newair <- airpollution
newair[ ,c("Pop", "HCPot", "NOx", "NOxPot")] <-
  log(newair[, c("Pop", "HCPot", "NOx", "NOxPot")])
colnames(newair)[c(10, 13, 14, 16)] <- c("log(Pop)", "log(HCPot)", "log(NOx)", "log(NOxP
pushViewport(viewport(layout = grid.layout(2, 2)))
k <- 0
for(i in c(10, 13, 14, 16)) {
    k <- k+1
    plt <-
      ggplot(data=newair,
           aes_(x = as.name(names(newair)[i]),
                y = as.name(names(newair)[1]))) +
    geom_point() +
    geom_smooth(method = "lm", se=FALSE)
    print(plt,
    vp=viewport(layout.pos.row=pos[k, 1],
              layout.pos.col=pos[k, 2]))
    cat("Mortality and ", colnames(newair)[i],
      " ", round(cor(newair$Mortality,
                     newair[, i]), 3), "\n")
```

```
}
```




```
## Mortality and  log(Pop)   0.085
## Mortality and  log(HCPot)   0.125
## Mortality and  log(NOx)   0.28
## Mortality and  log(NOxPot)   0.28
```

so in all cases the log transform worked, and we will use newair from now on.

Let's find the correlations in absolute value of the predictors with the response, in order:

```
cors <- round(cor(newair), 2)
sort(abs(cors[ ,"Mortality"]), decreasing = TRUE)[-1]
```

```
##    NonWhite   Education        Rain      SO2Pot   Pop.House    JulyTemp
##        0.65        0.51        0.43        0.42        0.37        0.32
## WhiteCollar      Income    log(NOx) log(NOxPot)  PopDensity  log(HCPot)
##        0.29        0.28        0.28        0.28        0.25        0.13
##      RelHum    log(Pop)     JanTemp
##        0.10        0.09        0.02
```

Next we look at the correlations between the predictors.

```
cors[-1, -1]
```

```
##             JanTemp JulyTemp RelHum  Rain Education PopDensity NonWhite
## JanTemp        1.00     0.32   0.09  0.06      0.11      -0.08     0.46
## JulyTemp       0.32     1.00  -0.44  0.47     -0.27      -0.01     0.60
## RelHum         0.09    -0.44   1.00 -0.12      0.19      -0.15    -0.12
## Rain           0.06     0.47  -0.12  1.00     -0.47       0.08     0.30
## Education      0.11    -0.27   0.19 -0.47      1.00      -0.24    -0.21
```

```
## PopDensity    -0.08    -0.01  -0.15  0.08     -0.24       1.00    -0.01
## NonWhite       0.46     0.60  -0.12  0.30     -0.21      -0.01     1.00
## WhiteCollar    0.21    -0.01   0.01 -0.11      0.49       0.25    -0.06
## log(Pop)       0.32     0.04  -0.02 -0.28      0.27       0.21     0.22
## Pop.House     -0.33     0.26  -0.14  0.20     -0.39      -0.17     0.35
## Income         0.20    -0.19   0.13 -0.36      0.51       0.00    -0.10
## log(HCPot)     0.23    -0.41   0.18 -0.48      0.18       0.26     0.15
## log(NOx)       0.18    -0.30   0.10 -0.39      0.03       0.34     0.21
## SO2Pot        -0.09    -0.07  -0.12 -0.13     -0.23       0.42     0.16
## log(NOxPot)    0.18    -0.30   0.10 -0.39      0.03       0.34     0.21
##             WhiteCollar log(Pop) Pop.House Income log(HCPot) log(NOx)
## JanTemp            0.21     0.32     -0.33   0.20       0.23     0.18
## JulyTemp          -0.01     0.04      0.26  -0.19      -0.41    -0.30
## RelHum             0.01    -0.02     -0.14   0.13       0.18     0.10
## Rain              -0.11    -0.28      0.20  -0.36      -0.48    -0.39
## Education          0.49     0.27     -0.39   0.51       0.18     0.03
## PopDensity         0.25     0.21     -0.17   0.00       0.26     0.34
## NonWhite          -0.06     0.22      0.35  -0.10       0.15     0.21
## WhiteCollar        1.00     0.28     -0.35   0.37       0.16     0.11
## log(Pop)           0.28     1.00     -0.26   0.41       0.48     0.50
## Pop.House         -0.35    -0.26      1.00  -0.30      -0.22    -0.12
## Income             0.37     0.41     -0.30   1.00       0.29     0.25
## log(HCPot)         0.16     0.48     -0.22   0.29       1.00     0.94
## log(NOx)           0.11     0.50     -0.12   0.25       0.94     1.00
## SO2Pot            -0.06     0.37     -0.01   0.07       0.57     0.68
## log(NOxPot)        0.11     0.50     -0.12   0.25       0.94     1.00
##             SO2Pot log(NOxPot)
## JanTemp      -0.09        0.18
## JulyTemp     -0.07       -0.30
## RelHum       -0.12        0.10
## Rain         -0.13       -0.39
## Education    -0.23        0.03
## PopDensity    0.42        0.34
## NonWhite      0.16        0.21
## WhiteCollar  -0.06        0.11
## log(Pop)      0.37        0.50
## Pop.House    -0.01       -0.12
## Income        0.07        0.25
## log(HCPot)    0.57        0.94
## log(NOx)      0.68        1.00
## SO2Pot        1.00        0.68
## log(NOxPot)   0.68        1.00
```

We find:

a) there are sizable correlations (for example cor(NonWhite,JulyTemp)=0.60).

Because of this interpreting (understanding) the final model will be difficult.

b) LOGT(NOxPot) and LOGT(NOx) are perfectly correlated.

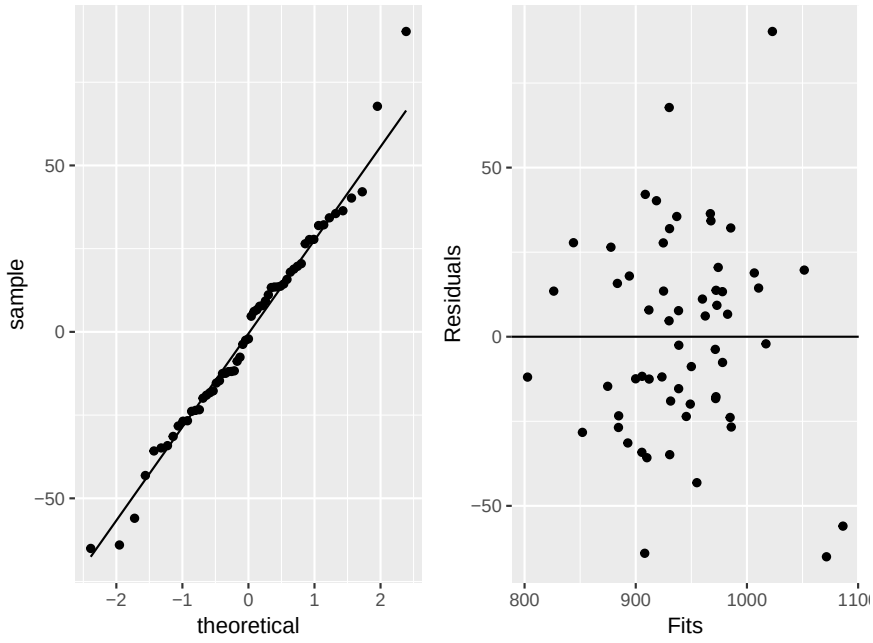
Using perfectly correlated predictors is not possible so we eliminate one of them, say log(NOx):

```
newair <- newair[, -16]
```

Next we fit a model with all the predictors and check the assumptions:

```
fit <- lm(Mortality~., data=newair)
```

```
pushViewport(viewport(layout = grid.layout(1, 2)))
df <- data.frame(Residuals=resid(fit),
            Fits = fitted(fit))
print(ggplot(data=df, aes(sample=Residuals)) +
           geom_qq() + geom_qq_line(),
    vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(ggplot(data=df, aes(Fits, Residuals)) +
            geom_point() +
            geom_hline(yintercept = 0),
    vp=viewport(layout.pos.row=1, layout.pos.col=2))
```



The residual vs fits plot looks fine, so there is no problem with the model.

The normal plot is ok, so no problem with the normal assumption. The residual vs fits plot looks fine, so there is no problem with the equal variance assumption.

Next we use the best subset regression to see whether we can find a model with fewer predictors.

```
out <- leaps(newair[, -1], newair$Mortality,
      method = "Cp", nbest=1)
colnames(newair)[-1][out$which[seq_along(out$Cp)[out$Cp==min(out$Cp)], ]]
```

```
## [1] "JanTemp"     "Rain"        "PopDensity"  "NonWhite"    "WhiteCollar"
## [6] "log(NOx)"
```

It suggests a model based on JanTemp, Rain, PopDensity, NonWhite, WhiteCollar and LOGT(NOx) with Mallow's C_p=4.32

```
df <-  newair[, c("Mortality",
                  "JanTemp",
                  "Rain",
                  "PopDensity",
                  "NonWhite",
                  "WhiteCollar",
                  "log(NOx)")]
fit <- lm(Mortality~., data=df)
summary(fit)
```

```
##
## Call:
## lm(formula = Mortality ~ ., data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -67.749 -24.087   3.249  19.473  93.474
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 944.274572  47.170880  20.018  < 2e-16
## JanTemp      -1.941998   0.516119  -3.763 0.000428
## Rain          1.924283   0.484198   3.974 0.000219
## PopDensity    0.006437   0.003608   1.784 0.080282
## NonWhite      4.194072   0.620038   6.764 1.18e-08
## WhiteCollar  -2.723255   0.947170  -2.875 0.005839
## `log(NOx)`   17.000178   4.879795   3.484 0.001012
##
## Residual standard error: 33.46 on 52 degrees of freedom
## Multiple R-squared:  0.7425, Adjusted R-squared:  0.7127
## F-statistic: 24.98 on 6 and 52 DF,  p-value: 1.028e-13
```

Because the best model does still include one of the pollution variables, we can conclude that pollution adds to the mortality rate.

And we are done!

Example: US Temperatures

The data gives the normal average January minimum temperature in degrees Fahrenheit with the latitude and longitude of 56 U.S. cities. (For each year from 1931 to 1960, the daily minimum temperatures in January were added together and divided by 31. Then, the averages for each year were averaged over the 30 years.)

Variables:
City: City
State: State postal abbreviation
JanTemp: Average January minimum temperature in degrees F.
Latitude: Latitude in degrees north of the equator
Longitude: Longitude in degrees west of the prime meridian

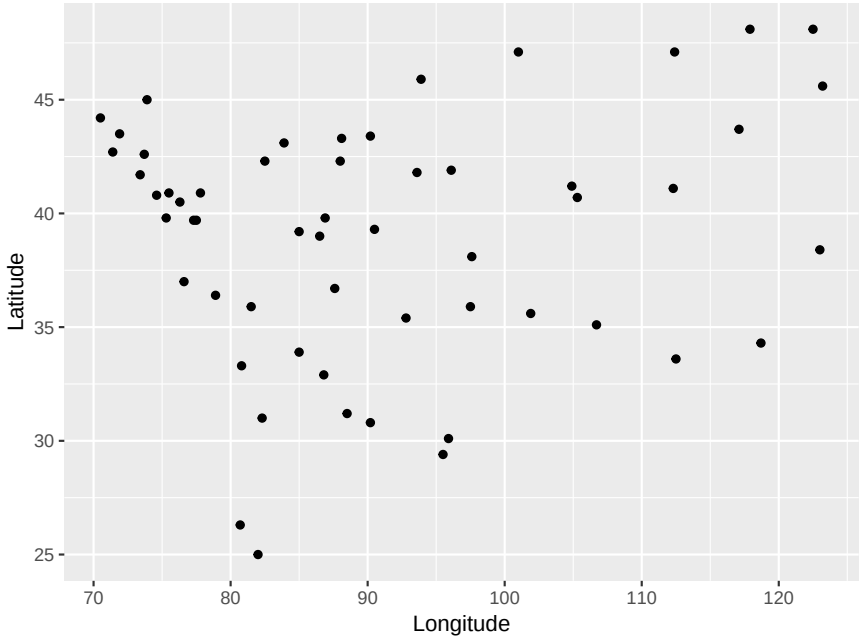
```
kable.nice(head(ustemperature))
```

City	State	JanTemp	Latitude	Longitude
Mobile	AL	44	31.2	88.5
Montgomery	AL	38	32.9	86.8
Phoenix	AZ	35	33.6	112.5
LittleRock	AR	31	35.4	92.8
LosAngeles	CA	47	34.3	118.7
SanFrancisco	CA	42	38.4	123.0

we want to develop a model that predicts the temperature from the longitude and latitude.

Let's begin by considering the predictors:

```
ggplot(data=ustemperature, aes(Longitude, Latitude)) +
  geom_point()
```

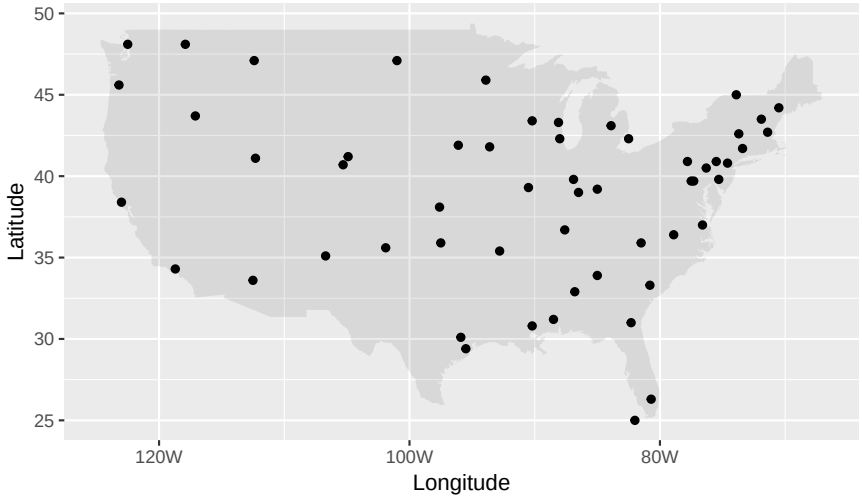


this however looks wrong, it is switched left to right. That is because every Longitude comes with East and West, and all of the US is in West. So we need to

```
ustemperature$Longitude <- -ustemperature$Longitude
```

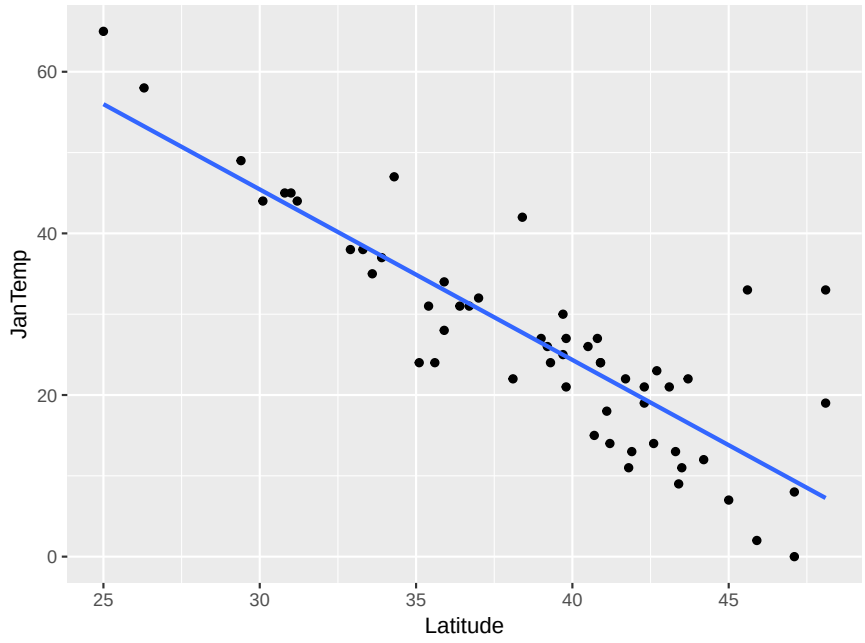
we can do even better than just the scatterplot:

```
library(maps)
usa <- map_data("usa")
ggplot() +
  geom_polygon(data = usa,
               aes(x=long, y = lat, group = group),
               alpha=0.1) +
  coord_fixed(1.3) +
  geom_point(data=ustemperature,
             aes(Longitude, Latitude)) +
  labs(x="Longitude", y="Latitude") +
  scale_x_continuous(breaks = c(-120, -100, -80),
                     labels = c("120W", "100W", "80W"))
```



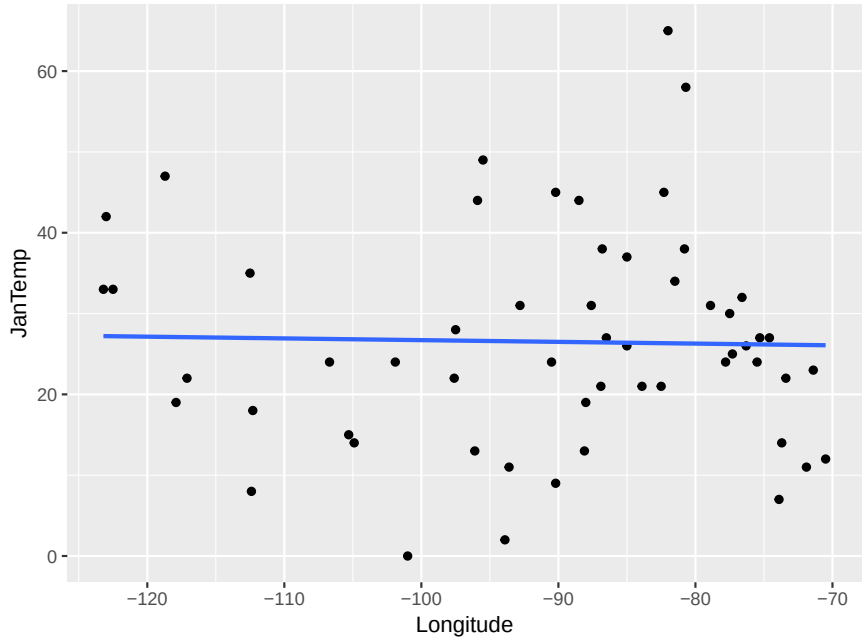
Now the relationship of Latitude to Temperature

```
ggplot(data=ustemperature, aes(Latitude, JanTemp)) +
  geom_point() +
  geom_smooth(method = "lm", se=FALSE)
```



seems fairly linear.

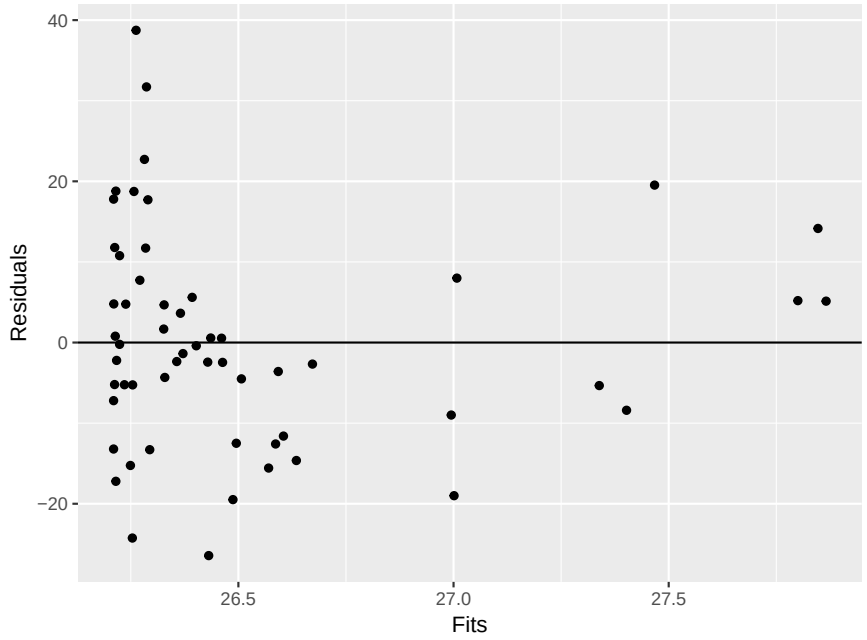
```
ggplot(data=ustemperature, aes(Longitude, JanTemp)) +
  geom_point() +
  geom_smooth(method = "lm", se=FALSE)
```



this does not. Let's fit a polynomial model in Longitude:

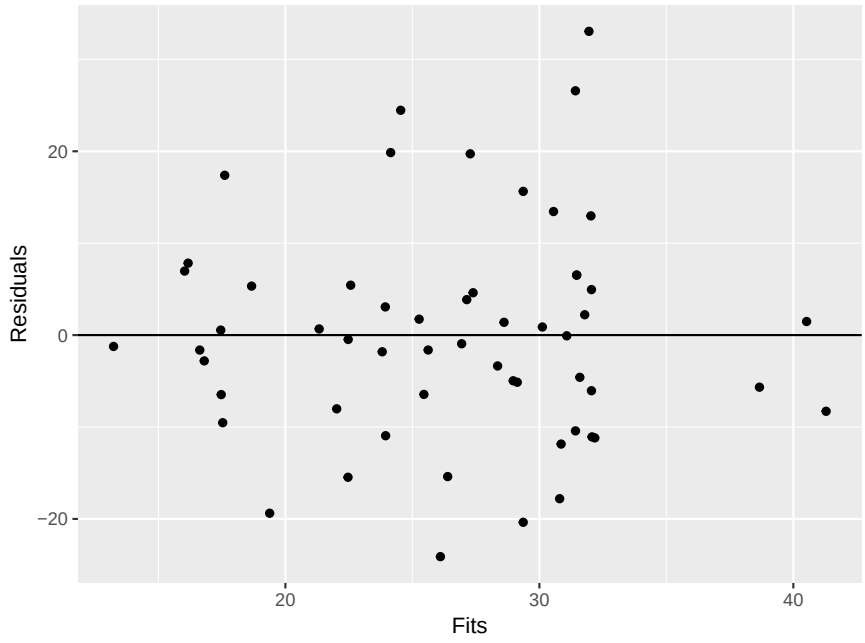
```
fit.quad <- lm(JanTemp~poly(Longitude, 2),
               data=ustemperature)
```

```
df <- data.frame(Residuals=resid(fit.quad),
            Fits = fitted(fit.quad))
ggplot(data=df, aes(Fits, Residuals)) +
            geom_point() +
            geom_hline(yintercept = 0)
```



not so good yet, so

```
fit.cube <- lm(JanTemp~poly(Longitude, 3),
               data=ustemperature)
df <- data.frame(Residuals=resid(fit.cube),
            Fits = fitted(fit.cube))
ggplot(data=df, aes(Fits, Residuals)) +
            geom_point() +
            geom_hline(yintercept = 0)
```



and that is fine.

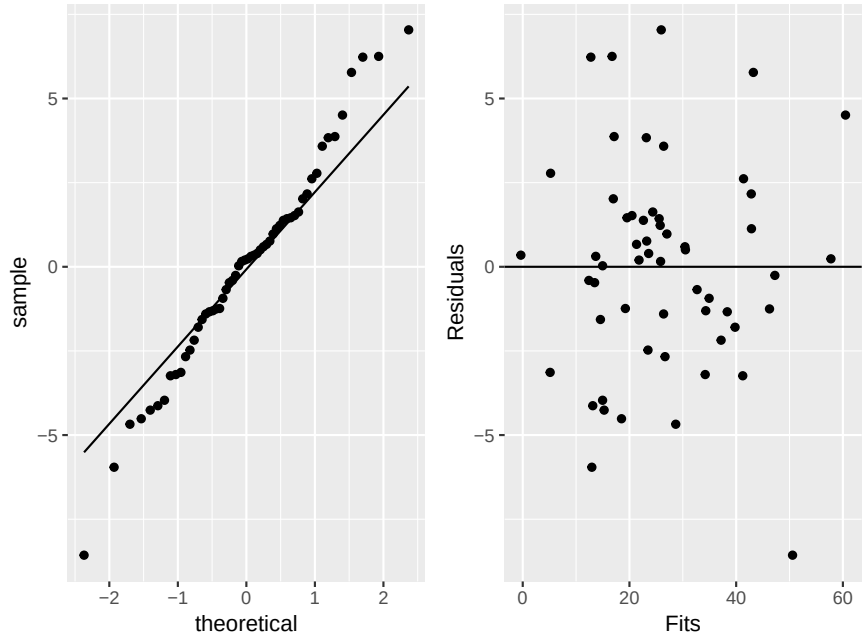
Now we put it together:

```
fit <- lm(JanTemp~Latitude + poly(Longitude, 3),
               data=ustemperature)
summary(fit)
```

```
##
## Call:
## lm(formula = JanTemp ~ Latitude + poly(Longitude, 3), data = ustemperature)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -8.569 -1.624  0.218  1.472  7.039
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)          118.39739    3.53301  33.512  < 2e-16
## Latitude              -2.35772    0.08998 -26.202  < 2e-16
## poly(Longitude, 3)1 -15.99052    3.26685  -4.895 1.03e-05
## poly(Longitude, 3)2  36.26524    3.47734  10.429 3.02e-14
## poly(Longitude, 3)3 -27.59874    3.30506  -8.350 4.13e-11
##
## Residual standard error: 3.225 on 51 degrees of freedom
## Multiple R-squared:  0.9461, Adjusted R-squared:  0.9419
## F-statistic: 223.9 on 4 and 51 DF,  p-value: < 2.2e-16
```

```
pushViewport(viewport(layout = grid.layout(1, 2)))
df <- data.frame(Residuals=resid(fit),
```

```
            Fits = fitted(fit))
print(ggplot(data=df, aes(sample=Residuals)) +
           geom_qq() + geom_qq_line(),
    vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(ggplot(data=df, aes(Fits, Residuals)) +
            geom_point() +
            geom_hline(yintercept = 0),
    vp=viewport(layout.pos.row=1, layout.pos.col=2))
```



shows that this indeed a good model.

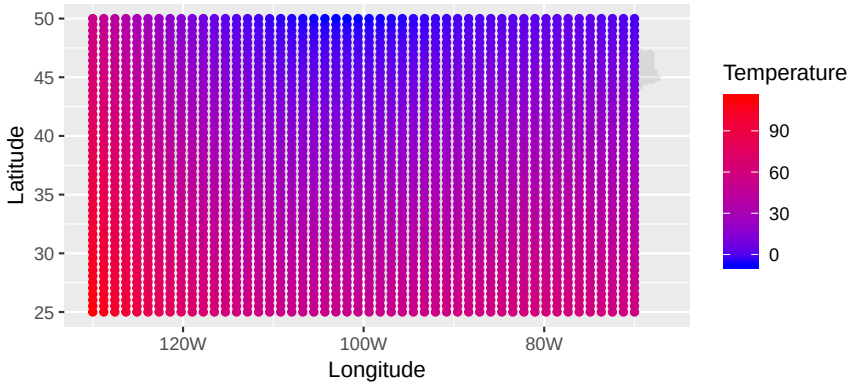
How can we visualize this model? Let's do the following:

- find a fine grid of points in the US
- for each such point use the model to predict the temperature - draw the map with these predictions

```
x <- seq(-130, -70, length=50)
y <- seq(25, 50, length=50)
xy <- expand.grid(x, y)
df <- data.frame(Longitude=xy[, 1],
                 Latitude=xy[, 2])
df$Temp <- predict(fit, df)
```

```
ggplot() +
  geom_polygon(data = usa,
               aes(x=long, y = lat, group = group),
               alpha=0.1) +
  coord_fixed(1.3) +
```

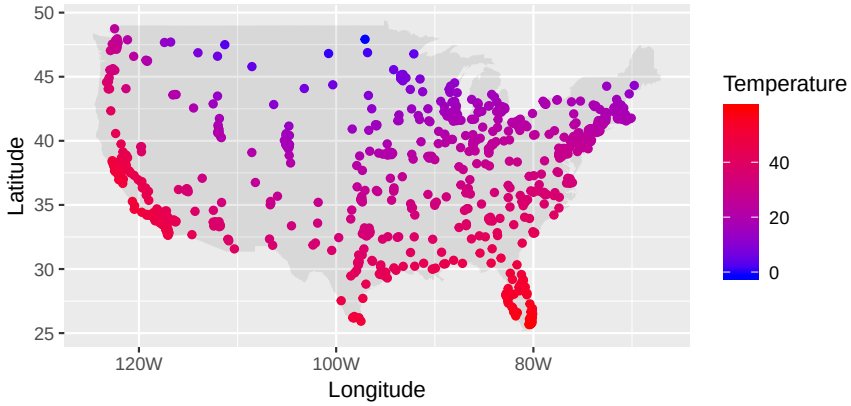
```
  geom_point(data=df, aes(Longitude, Latitude, color=Temp)) +
  scale_colour_gradient(low="blue", high="red") +
  labs(x="Longitude", y="Latitude", color="Temperature") +
  scale_x_continuous(breaks = c(-120, -100, -80),
                     labels = c("120W", "100W", "80W"))
```



It would be nicer, though, if we had points only in the US. We can use the data set us.cities in the map library to get coordinates:

```
df <- us.cities[, 5:4]
df <- df[df[, 2]<50, ] #not Alaska
df <- df[df[, 2]>25, ] #not Hawaii
colnames(df) <- c("Longitude", "Latitude")
df$Temp <- predict(fit, df)
```

```
ggplot() +
  geom_polygon(data = usa,
               aes(x=long, y = lat, group = group),
               alpha=0.1) +
  coord_fixed(1.3) +
  geom_point(data=df, aes(Longitude, Latitude, color=Temp)) +
  scale_colour_gradient(low="blue", high="red") +
  labs(x="Longitude", y="Latitude", color="Temperature") +
  scale_x_continuous(breaks = c(-120, -100, -80),
                     labels = c("120W", "100W", "80W"))
```



How about Puerto Rico?

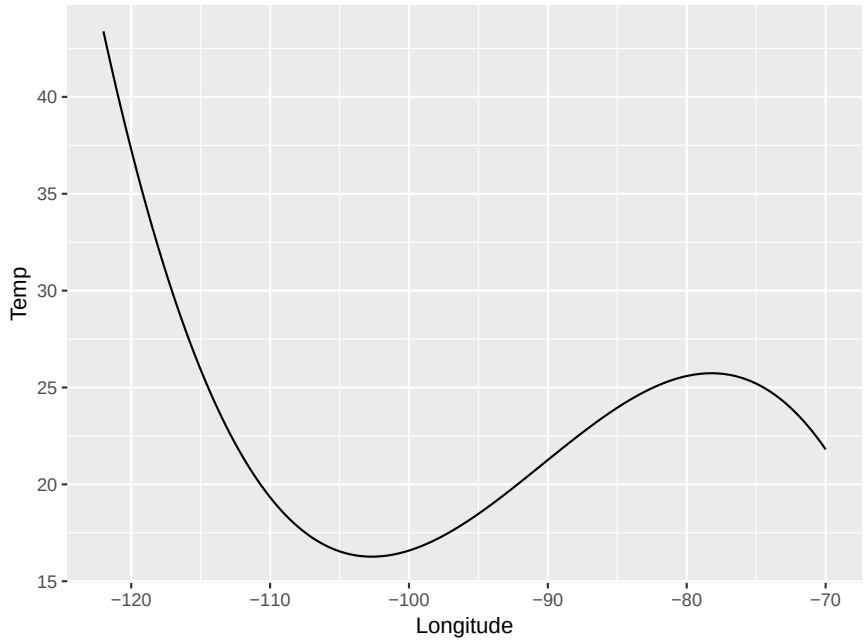
```
df <- data.frame(Longitude=-65.1, Latitude=18.1)
predict(fit, df)
```

```
##        1
## 66.26492
```

Clearly this is an extrapolation!

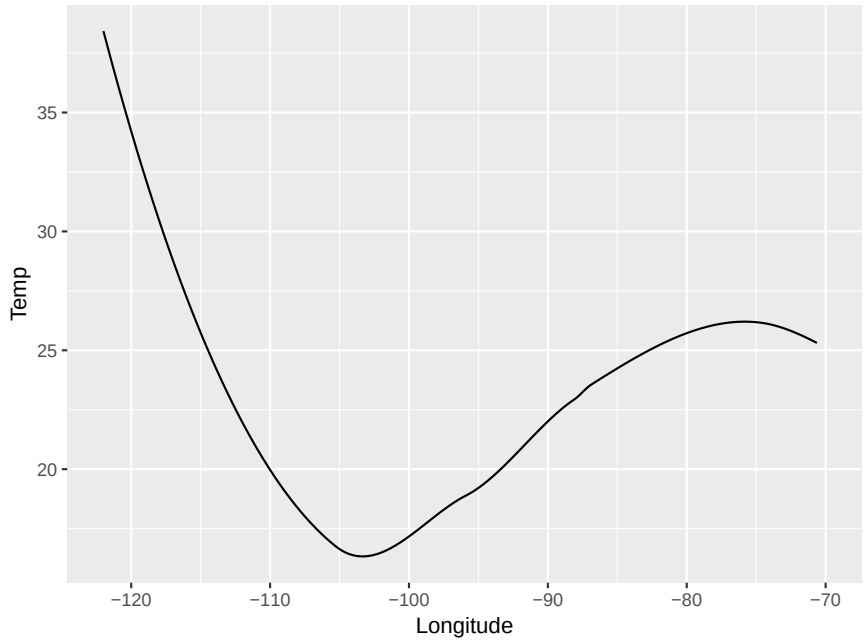
Here is another way to understand this model: image we take a trip from New York (Long=73.94, Lat=40.67) to San Francisco (Long-122.45, Lat=37.77). How does our model say the temperature changes?

```
df <- data.frame(Longitude=seq(-122, -70, length=250),
                 Latitude=rep(40, 250))
df$Temp <- predict(fit, df)
ggplot(df, aes(Longitude, Temp)) +
  geom_line()
```



How about a nonparametric fit?

```
fit.loess <- loess(JanTemp~Longitude+Latitude,
                   data=ustemperature)
df$Temp <- predict(fit.loess, df)
ggplot(df, aes(Longitude, Temp)) +
  geom_line()
```



which looks fairly similar.

Models with Categorical Predictors

Example: Environmental, Safety and Health Attitudes

Environment, Safety and Health Attitudes of employees of a laboratory. Employees are given a questionnaire, which is then collated into an average score from 1(bad) to 10(good). We also have available the length of service of the employee and their gender.

```
kable.nice(head(esh))
```

ES.H	Yrs.Serv	Sex
7.6	5	Female
9.0	30	Female
8.0	12	Female
6.8	7	Female
7.4	7	Female
9.8	27	Female

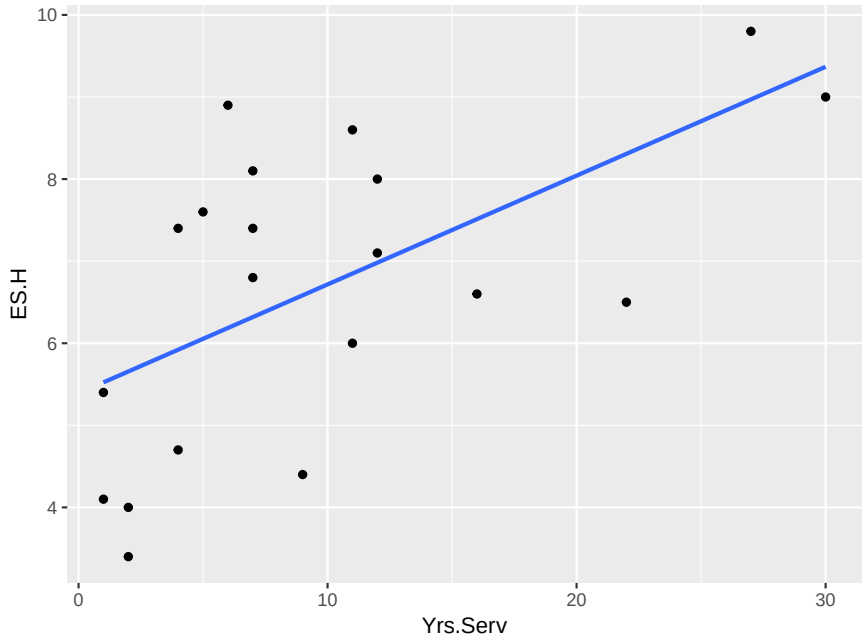
One of the predictor variables (Sex) is actually categorical. A categorical variable used in a regression model is often referred to as a *dummy* variable.

Let's start by looking at each predictor separately.

- Years is quantitative, so do the scatterplot:

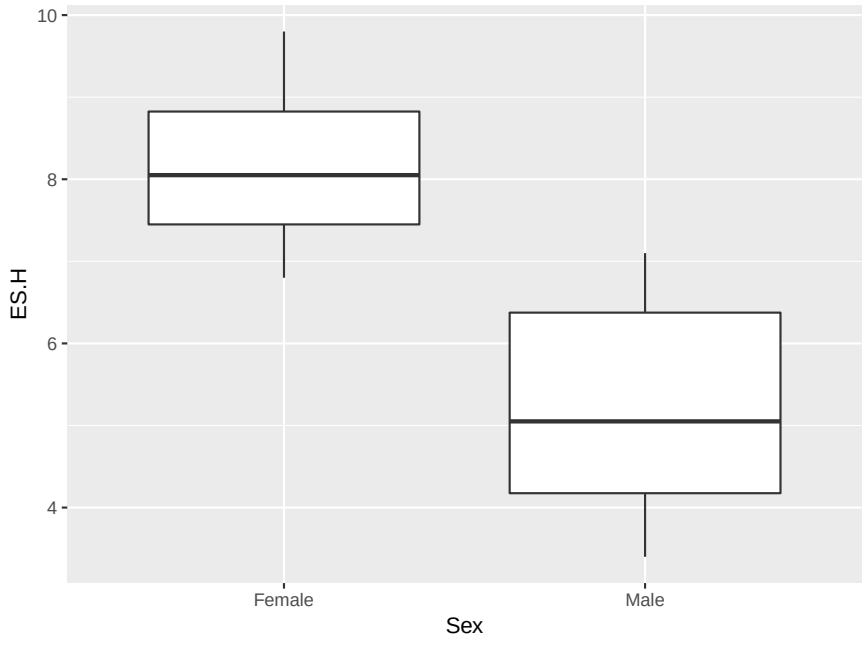
```
attach(esh)
```

```
ggplot(data=esh, aes(Yrs.Serv, ES.H)) +
    geom_point() +
  geom_smooth(method = "lm", se=FALSE)
```



- Sex is categorical, so do the boxplot:

```
ggplot(data=esh, aes(Sex, ES.H)) +
    geom_boxplot()
```



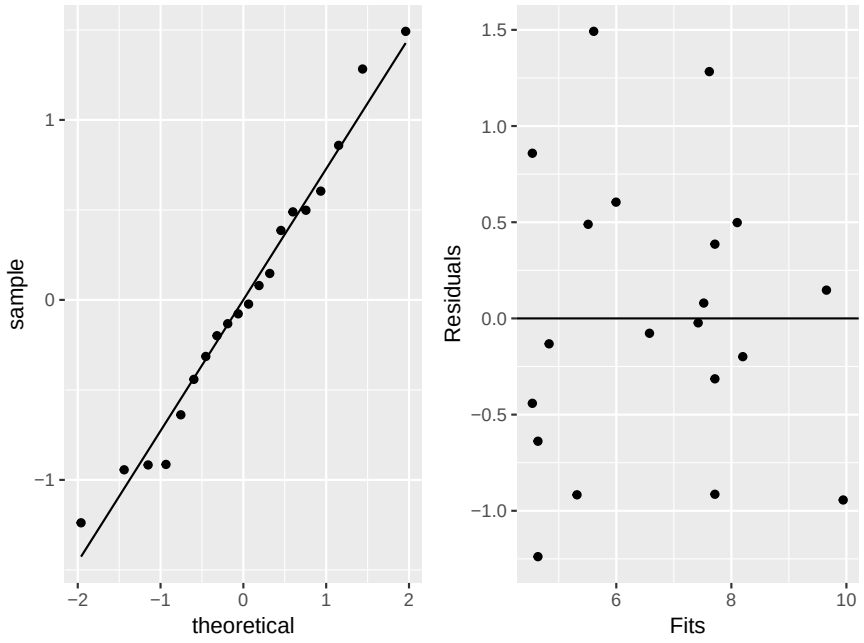
The values in Sex (Male, Female) are text but in a regression we need everything to be numeric, so in order to use Sex in a regression model we first have to *code* the variable as numbers, for example Female=0 and Male=1. Then

```
SexCode <- rep(0, length(Sex))
SexCode[Sex=="Male"] <- 1
```

```
esh1 <- data.frame(ESH=esh$ES.H,
                   YrsServ=esh$Yrs.Serv,
                   SexCode=SexCode)
fit <- lm(ESH~., data=esh1)
summary(fit)
```

```
##
## Call:
## lm(formula = ESH ~ ., data = esh1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.23832 -0.49061 -0.05023  0.49141  1.49221
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.03542    0.35862  19.618 4.10e-13
## YrsServ      0.09695    0.02228   4.351 0.000435
## SexCode     -2.59099    0.36058  -7.186 1.52e-06
##
## Residual standard error: 0.7861 on 17 degrees of freedom
## Multiple R-squared:  0.8394, Adjusted R-squared:  0.8205
## F-statistic: 44.44 on 2 and 17 DF,  p-value: 1.771e-07
```

```
  pushViewport(viewport(layout = grid.layout(1, 2)))
  df <- data.frame(Residuals=resid(fit),
            Fits = fitted(fit))
  print(ggplot(data=df, aes(sample=Residuals)) +
          geom_qq() + geom_qq_line(),
    vp=viewport(layout.pos.row=1, layout.pos.col=1))
  print(ggplot(data=df, aes(Fits, Residuals)) +
            geom_point() +
            geom_hline(yintercept = 0),
    vp=viewport(layout.pos.row=1, layout.pos.col=2))
```



The residual vs. fits and normal plot look good, so this is a good model.

Or is it?

Let's do the following: what would the equation look like if we knew the person was female? (or male). Well:

$$\begin{aligned}&\text{Female ES.H} = \\ &7.035 + 0.097\text{Yrs.Serv} - 2.591 \cdot 0 = \\ &7.035 + \mathbf{0.097}\text{Yrs.Serv}\end{aligned}$$

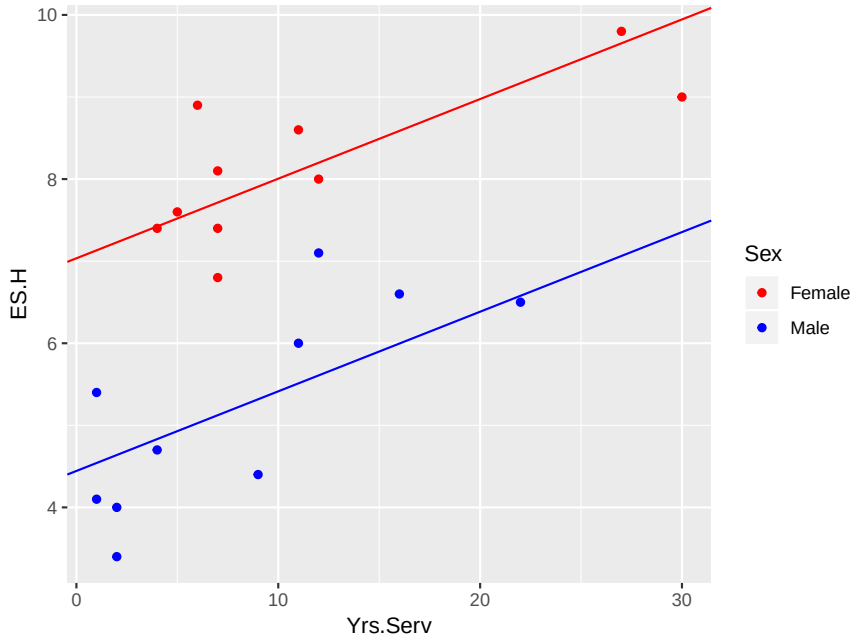
$$\begin{aligned}&\text{Male ES.H} = \\ &7.035 + 0.097\text{Yrs.Serv} - 2.591 \cdot 1 = \\ &4.444 + \mathbf{0.097}\text{Yrs.Serv}\end{aligned}$$

Notice that both equations have the same slope, so we have **parallel** lines.

Note such a model is also often called an *additive* model, similar to an ANOVA without interaction!

What does this look like? Here it is:

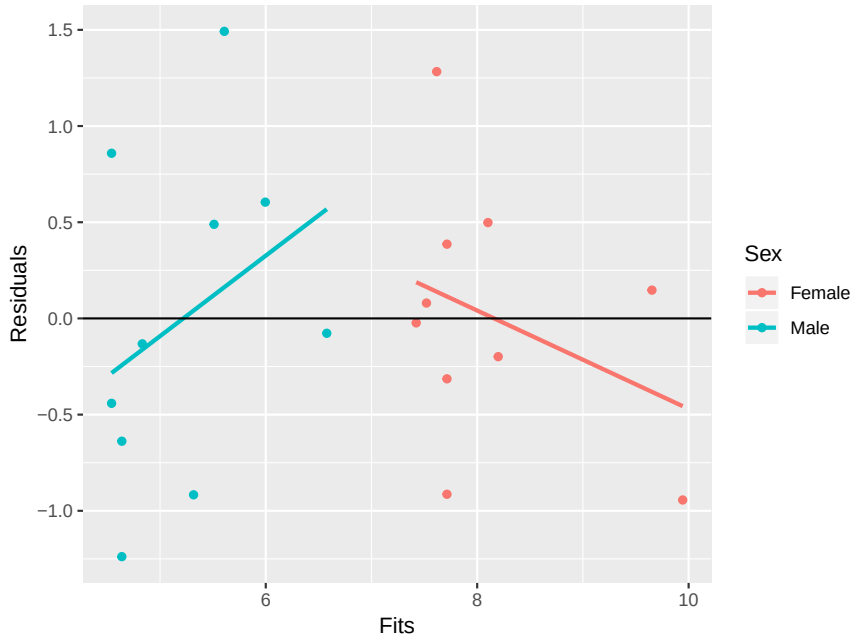
```
ggplot(data=esh, aes(Yrs.Serv, ES.H, color=Sex)) +
         geom_point() +
         scale_color_manual(values=c("red", "blue")) +
         geom_abline(intercept = c(7.035, 4.444),
                     slope = c(0.097, 0.097),
                     color=c("red", "blue"))
```



Now a model with parallel line may or may not make sense for our data, but it does not have to. Except that no matter what, the way we used the categorical variable (simply code it and use it) we will **always** result in parallel lines!

Is there a way to see whether this is ok here? Yes, what we need is a version of the residual vs fits plot that identifies the plotting symbols by Sex. If the model is good, this residual vs fits plot should also show no pattern.

```
ggplot(data=df, aes(Fits, Residuals, color=Sex)) +
  geom_point() +
  geom_smooth(method = "lm", se=FALSE) +
  geom_hline(yintercept = 0)
```



and as we can see there is a definite pattern in the colors.

So, how do we get away from parallel lines? This can be done by adding a variable Yrs.Serv*SexCode.

```
esh1$prod <- esh1$YrsServ*esh1$SexCode
fit.prod <- lm(ESH~., data=esh1)
summary(fit.prod)
```

```
##
## Call:
## lm(formula = ESH ~ ., data = esh1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.0280 -0.4430 -0.1206  0.3965  1.3300
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.32289    0.39824  18.388 3.48e-12
## YrsServ      0.07216    0.02736   2.637   0.0179
## SexCode     -3.20289    0.54300  -5.898 2.25e-05
## prod         0.06534    0.04443   1.470   0.1608
##
## Residual standard error: 0.7605 on 16 degrees of freedom
## Multiple R-squared:  0.8585, Adjusted R-squared:  0.832
## F-statistic: 32.37 on 3 and 16 DF,  p-value: 5.004e-07
```

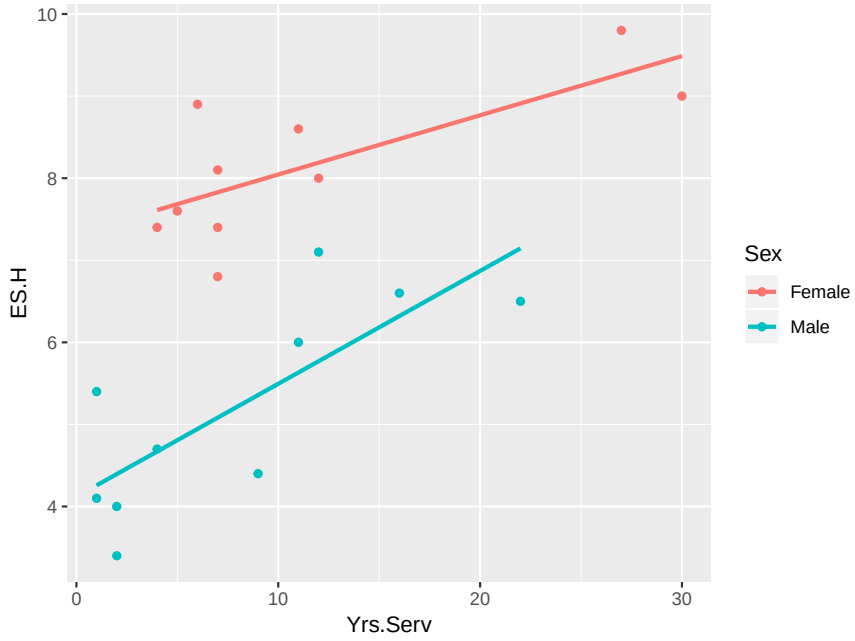
and now:

$$\begin{aligned}&\text{Female ES.H} = \\ &7.323 + 0.072\text{Yrs.Serv} - 3.203 \cdot 0 + 0.065 \cdot \text{Yrs.Serv*}0 = \\ &7.323 + 0.072\text{Yrs.Serv}\end{aligned}$$

$$\begin{aligned}&\text{Male ES.H} = \\ &7.323 + 0.072\text{Yrs.Serv} - 3.203 \cdot 1 + 0.065 \cdot \text{Yrs.Serv*}1 = \\ &4.120 + 0.138\text{Yrs.Serv}\end{aligned}$$

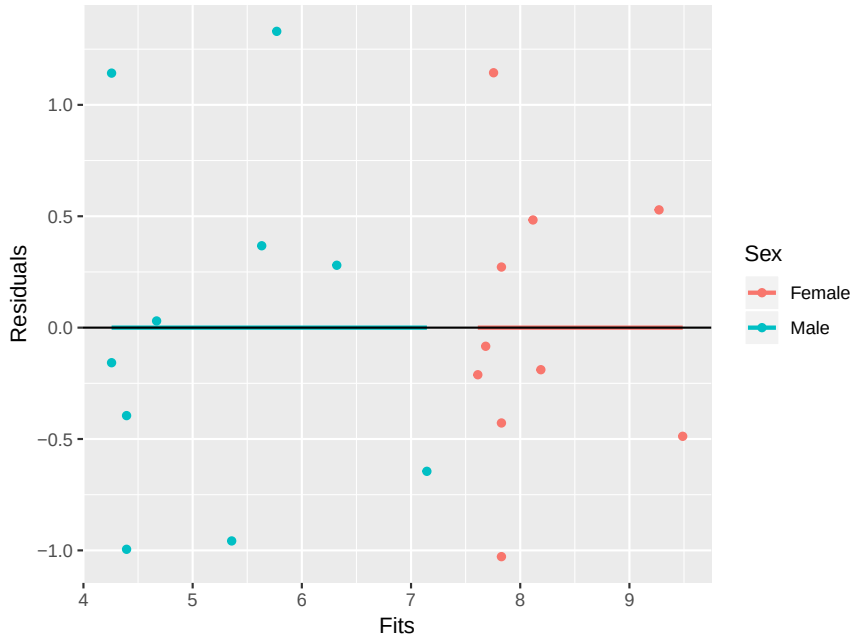
and so this fits *two separate lines*.

```
ggplot(data=esh, aes(Yrs.Serv, ES.H, color=Sex)) +
  geom_point() +
  geom_smooth(method = "lm", se=FALSE)
```



Now the residual vs. fits plot looks like this:

```
df <- data.frame(Residuals=resid(fit.prod),
                 Fits = fitted(fit.prod))
ggplot(data=df, aes(Fits, Residuals, color=Sex)) +
  geom_point() +
  geom_smooth(method = "lm", se=FALSE) +
  geom_hline(yintercept = 0)
```



Note you can get the same two equations by splitting up the data set into two parts, the score and years of the Females and the score and years of the Males, and then doing a simple regression for both:

```
round(coef(lm(ES.H[Sex=="Female"]~Yrs.Serv[Sex=="Female"])), 3)
```

```
##                (Intercept) Yrs.Serv[Sex == "Female"]
##                      7.323                      0.072
```

```
round(coef(lm(ES.H[Sex=="Male"]~Yrs.Serv[Sex=="Male"])), 3)
```

```
##              (Intercept) Yrs.Serv[Sex == "Male"]
##                    4.120                   0.137
```

Doing one multiple regression has some advantages, though. For example you get one R^2 for the whole problem, not two for each part. Moreover, usually this R^2 will be higher than either of the other two.

Above we fitted the independent lines model by explicitly calculating the product term. A better way is to do this:

```
esh2 <- esh
esh2$Sex <- SexCode
fit.prod <- lm(ES.H~.^2, data=esh2)
round(coef(fit.prod), 3)
```

```
##  (Intercept)     Yrs.Serv          Sex Yrs.Serv:Sex
##        7.323        0.072       -3.203        0.065
```

So now we have two models:

- parallel lines: ES.H = 7.035 + 0.097 Yrs.Serv - 2.591 Sex

R^2 = 83.9%

- separate lines: ES.H = 7.323 + 0.072 Yrs.Serv - 3.203 SexCode + 0.065 Yrs.Serv*SexCode

R^2=85.85%

Clearly the second one has a higher R^2, but then the first one is a special case of the second (nested models) and so the model with parallel lines will **never** have an R^2 higher than the model with separate lines, and usually always has an R^2 a bit lower.

Of course the parallel lines model has two terms while the other one has three, and the third one is more complicated, so we would prefer the parallel lines model, if possible.

What we want to know is whether the model with two separate lines is **statistically significantly** better than the model with parallel lines. So we need a hypothesis test with:

H_0: the two separate lines model is NOT statistically significantly better than the parallel lines model.

H_a: the two separate lines model is statistically significantly better than the parallel lines model.

Notice that the parallel lines model is a special case of the two independent lines model, and so we can again use the *anova* to decide which is better:

```
anova(fit.prod, fit)
```

```
## Analysis of Variance Table
##
## Response: ES.H
##              Df Sum Sq Mean Sq F value    Pr(>F)
## Yrs.Serv      1 23.009  23.009 39.7826 1.043e-05
## Sex           1 31.905  31.905 55.1642 1.434e-06
## Yrs.Serv:Sex  1  1.251   1.251  2.1623    0.1608
## Residuals    16  9.254   0.578
```

gives a p-value of 0.1608 > 0.05, so the parallel lines model is just as good as the model with separate lines.

Prediction

Let's find 95% interval estimates for female employees with 0, 1, 2,..,10 years of service, using the parallel lines model:

```
fit <- lm(ES.H~., data=esh2)
nw <- data.frame(Yrs.Serv=0:10, Sex=rep(0, 11))
round(predict(fit, nw, interval="prediction"), 2)
```

```
##     fit  lwr  upr
## 1  7.04 5.21 8.86
## 2  7.13 5.32 8.94
```

```
## 3  7.23 5.43 9.03
## 4  7.33 5.54 9.11
## 5  7.42 5.65 9.20
## 6  7.52 5.75 9.29
## 7  7.62 5.86 9.38
## 8  7.71 5.96 9.47
## 9  7.81 6.06 9.56
## 10 7.91 6.16 9.65
## 11 8.00 6.26 9.75
```

Lines and Interaction

Above we explained the problem of using categorical predictors in a regression model in terms of parallel lines vs. two independent lines. But in fact this another example of the issue of *interaction*, or more generally of a relationship between the predictors. Parallel lines are ok if the categorical and the continuous predictors are essentially independent. Often terms such as Yrs Serv*SexCode are also called *interaction terms*.

For your purposes in this class (and later when doing work such as this) simply remember to include product terms when you have categorical predictors. Then you can test if that term is really needed, and drop it if it is not.

Example: Sales of Shoes

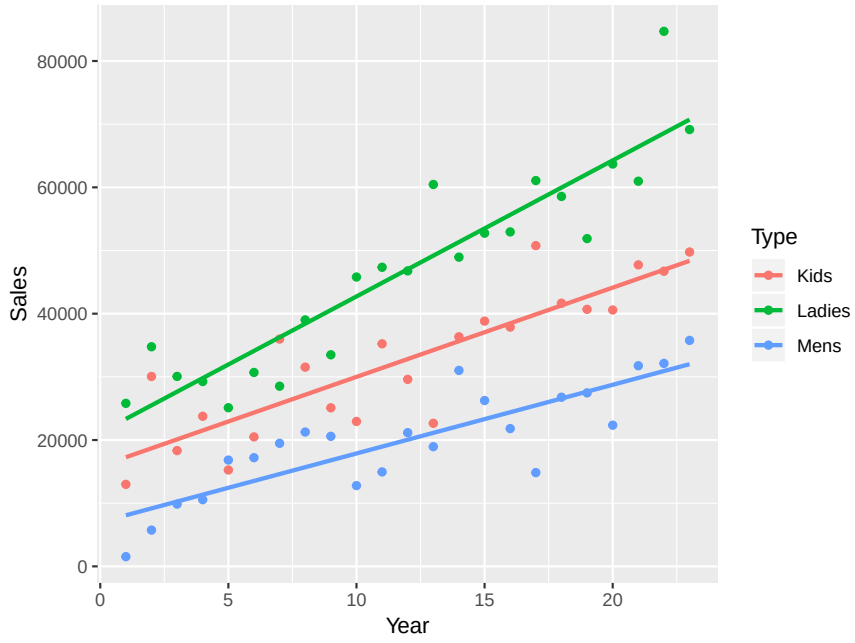
The number of shoes sold by year and type.

```
kable.nice(head(shoesales))
```

Sales	Year	Type
1539	1	Mens
12984	1	Kids
25809	1	Ladies
5742	2	Mens
30058	2	Kids
34764	2	Ladies

Let's have a look at the data:

```
ggplot(data=shoesales, aes(Year, Sales, color=Type)) +
  geom_point() +
  geom_smooth(method = "lm", se=FALSE)
```



We want to find a model for predicting Sales from Year and Type. Again Type is a categorical variable and so we need to code it. The most obvious thing to do would be to code:

- Mens= 0
- Kids= 1
- Ladies = 2

but that is dangerous. Unlike a categorical variable numbers always have an order and a size. So by coding in this way we are saying that Mens comes before Kids. Worse , we are saying that the "distance" from Mens to Kids is the same as the "distance" from Kids to Ladies!

Whether this matters or not depends on the specific problem. There is however a way to include such a variable without introducing order or size:

```
d1 <- rep(0, length(Type))
d1[Type=="Kids"] <- 1
d2 <- rep(0, length(Type))
d2[Type=="Ladies"] <- 1
```

Notice that by knowing d1 and d2 we now exactly what the type is:

- d1=0, d2=0 → Mens
- d1=1, d2=0 → Kids
- d1=0, d2=1 → Ladies

so we have not lost any information, but we have also not introduced any order or size!

Now

```
df <- shoesales[, 1:2]
df$d1 <- d1
df$d2 <- d2
fit <- lm(Sales~., data=df)
summary(fit)
```

```
##
## Call:
## lm(formula = Sales ~ ., data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -12963.7  -3433.5   -469.7   3349.1  22146.6
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1429.1     1917.8   0.745    0.459
## Year          1551.6      115.4  13.440  < 2e-16
## d1           12774.5     1875.7   6.811 3.74e-09
## d2           26986.9     1875.7  14.388  < 2e-16
##
## Residual standard error: 6361 on 65 degrees of freedom
## Multiple R-squared:  0.8565, Adjusted R-squared:  0.8498
## F-statistic: 129.3 on 3 and 65 DF,  p-value: < 2.2e-16
```

This is of course an additive model, again we should worry about interaction. But now we have two categorical predictors, so we need to add two product terms:

```
fit.prod <- lm(Sales~.^2-d1:d2, data=df)
summary(fit.prod)
```

```
##
## Call:
## lm(formula = Sales ~ .^2 - d1:d2, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11588.7  -3433.0   -256.7   2947.3  16121.3
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   7000.5     2443.6   2.865 0.005662
## Year          1087.3      178.2   6.101 7.14e-08
## d1            8862.2     3455.7   2.565 0.012728
## d2           14185.1     3455.7   4.105 0.000119
## Year:d1        326.0      252.0   1.294 0.200541
## Year:d2       1066.8      252.0   4.233 7.64e-05
```

```
##
## Residual standard error: 5669 on 63 degrees of freedom
## Multiple R-squared:  0.8895, Adjusted R-squared:  0.8807
## F-statistic: 101.4 on 5 and 63 DF,  p-value: < 2.2e-16
```

And again we can test whether the product terms are needed:

```
anova(fit.prod, fit)
```

```
## Analysis of Variance Table
##
## Model 1: Sales ~ (Year + d1 + d2)^2 - d1:d2
## Model 2: Sales ~ Year + d1 + d2
##   Res.Df        RSS Df  Sum of Sq      F    Pr(>F)
## 1     63 2024940688
## 2     65 2629827256 -2 -604886567 9.4096 0.0002656
```

and we find that here the interaction is needed ($p = 0.0003$).

Example: Headache and Pain Reliever

A pharmaceutical company set up an experiment in which patients with a common type of headache were treated with a new analgesic or pain reliever. The analgesic was given to each patient in one of four dosage levels: 2, 5, 7 or 10 grams. Then the time until noticeable relieve was recorded in minutes. In addition the sex (coded as Female=0 and Male=1) and the blood pressure of each patient was recorded. The blood pressure groups where formed by comparing each patients diastolic and systolic pressure reading with historical data. Based on this comparison the patients are assigned to one of three types: low (0.25), medium (0.5), high (0.75) according to the respective quantiles of the historic data.

```
head(headache)
```

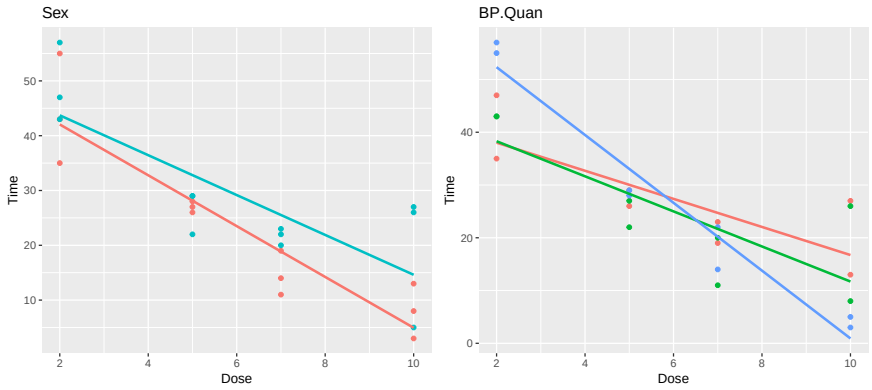
```
##   Time Dose Sex BP.Quan
## 1   35    2   0    0.25
## 2   43    2   0    0.50
## 3   55    2   0    0.75
## 4   47    2   1    0.25
## 5   43    2   1    0.50
## 6   57    2   1    0.75
```

here Sex and BP.Quan are already coded. BP.Quan is an interesting case because although it is categorical, it does have ordering and even a little bit of "size".

we want to determine the optimal dosage for each patient, possibly depending on sex and blood pressure.

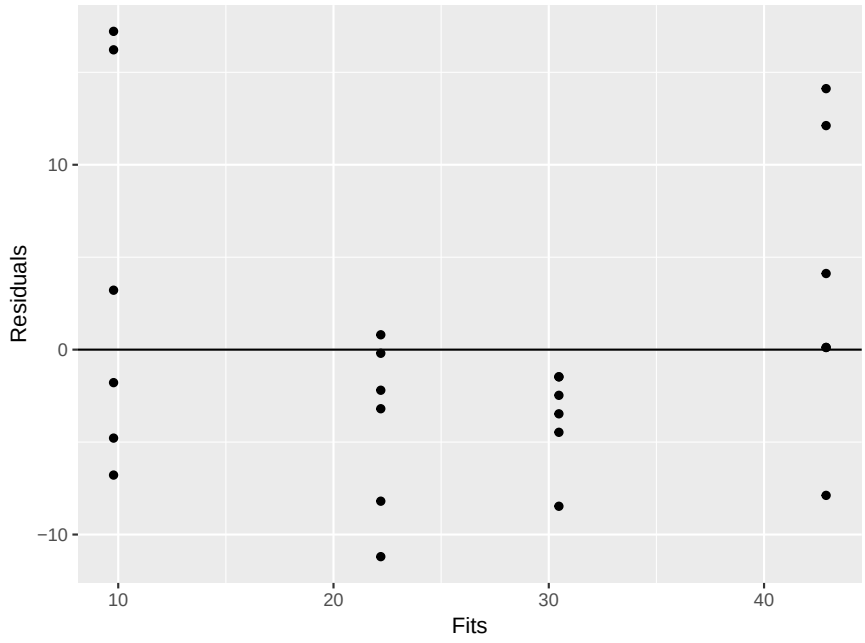
```
pushViewport(viewport(layout = grid.layout(1, 2)))
print(ggplot(data=headache,
             aes(Dose, Time, color=factor(Sex))) +
        geom_point() +
        geom_smooth(method = "lm", se=FALSE) +
```

```
        theme(legend.position="none") +
        labs(title="Sex"),
  vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(ggplot(data=headache,
             aes(Dose, Time, color=factor(BP.Quan))) +
        geom_point() +
        geom_smooth(method = "lm", se=FALSE) +
        theme(legend.position="none") +
        labs(title="BP.Quan"),
  vp=viewport(layout.pos.row=1, layout.pos.col=2))
```



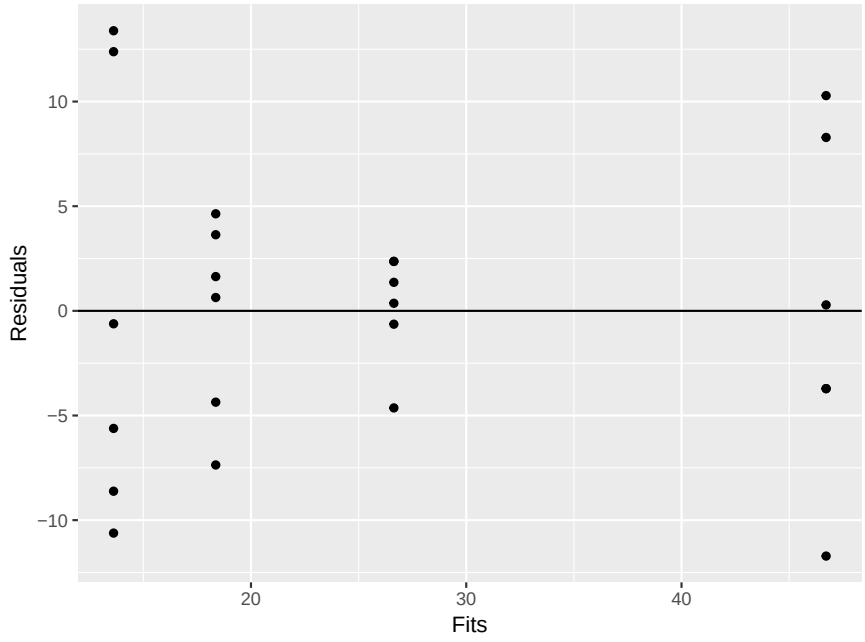
Let's start by fitting a linear model on Dose alone:

```
fit <- lm(Time~Dose, data=headache)
df <- data.frame(Residuals=resid(fit),
            Fits = fitted(fit))
ggplot(data=df, aes(Fits, Residuals)) +
            geom_point() +
            geom_hline(yintercept = 0)
```



There is a bit of a pattern here, so let's try a quadratic model. In this example we will eventually need the actual equations, so we won't use poly:

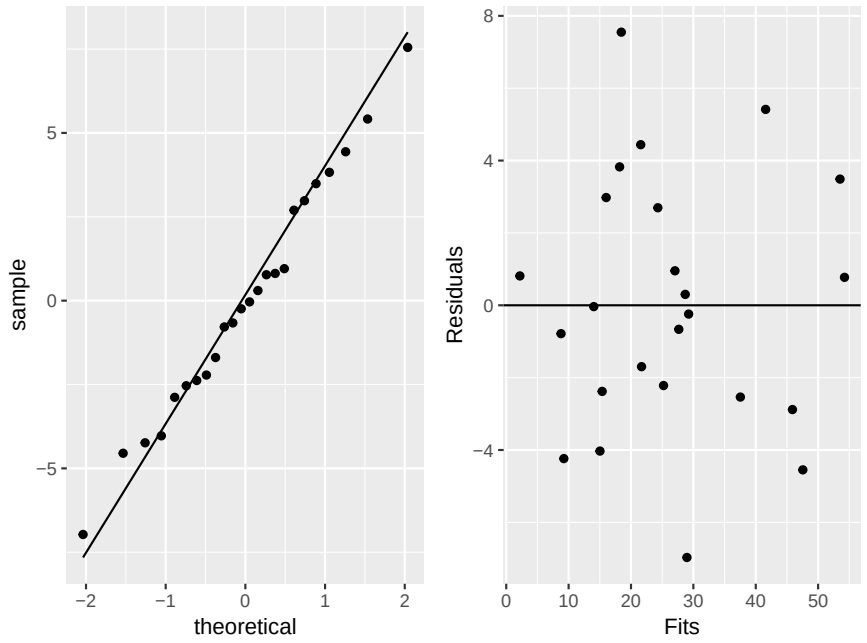
```
headache$Dose2 <- headache$Dose^2
fit <- lm(Time~Dose+Dose2, data=headache)
df <- data.frame(Residuals=resid(fit),
            Fits = fitted(fit))
ggplot(data=df, aes(Fits, Residuals)) +
            geom_point() +
            geom_hline(yintercept = 0)
```



and that looks better.

Now we will include the other two variables. One interesting question is whether BP.Quan is quantitative or categorical (in which case we should turn it into two dummy variables). The answer is not clear, and we will leave it alone. So

```
fit <- lm(Time~(Dose+Sex+BP.Quan)^3+Dose2, data=headache)
pushViewport(viewport(layout = grid.layout(1, 2)))
df <- data.frame(Residuals=resid(fit),
            Fits = fitted(fit))
print(ggplot(data=df, aes(sample=Residuals)) +
           geom_qq() + geom_qq_line(),
    vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(ggplot(data=df, aes(Fits, Residuals)) +
            geom_point() +
            geom_hline(yintercept = 0),
    vp=viewport(layout.pos.row=1, layout.pos.col=2))
```




```
summary(fit)
```

```
##
## Call:
## lm(formula = Time ~ (Dose + Sex + BP.Quan)^3 + Dose2, data = headache)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.9706 -2.4191 -0.1397  2.7665  7.5490
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)       41.2173     8.2720   4.983 0.000164
## Dose              -7.0353     1.8218  -3.862 0.001536
## Sex                4.3137    10.7536   0.401 0.693972
## BP.Quan           48.3235    14.0798   3.432 0.003705
## Dose2              0.5111     0.1184   4.316 0.000612
## Dose:Sex           1.0588     1.6120   0.657 0.521244
## Dose:BP.Quan      -7.4706     2.1106  -3.539 0.002973
## Sex:BP.Quan       -9.2941    19.9118  -0.467 0.647376
## Dose:Sex:BP.Quan  -0.1176     2.9849  -0.039 0.969080
##
## Residual standard error: 4.351 on 15 degrees of freedom
## Multiple R-squared:  0.9418, Adjusted R-squared:  0.9108
## F-statistic: 30.35 on 8 and 15 DF,  p-value: 6.567e-08
```

we see that the three-way interaction Dose:Sex:BP.Quan is not stat. significant (p=0.969), so we drop it:

```
fit <- lm(Time~(Dose+Sex+BP.Quan)^2+Dose2, data=headache)
summary(fit)
```

```
##
## Call:
## lm(formula = Time ~ (Dose + Sex + BP.Quan)^2 + Dose2, data = headache)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -6.971 -2.449 -0.125  2.763  7.549
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   41.0408     6.7350   6.094 1.55e-05
## Dose          -7.0059     1.6092  -4.354 0.000493
## Sex            4.6667     5.7655   0.809 0.430149
## BP.Quan       48.6765    10.5207   4.627 0.000280
## Dose2          0.5111     0.1147   4.457 0.000397
## Dose:Sex       1.0000     0.5900   1.695 0.109448
## Dose:BP.Quan  -7.5294     1.4451  -5.210 8.59e-05
## Sex:BP.Quan  -10.0000     8.4265  -1.187 0.252656
##
## Residual standard error: 4.213 on 16 degrees of freedom
## Multiple R-squared:  0.9418, Adjusted R-squared:  0.9164
## F-statistic:    37 on 7 and 16 DF,  p-value: 1.022e-08
```

again, two interactions are not significant, so

```
fit <- lm(Time~.+ Dose:BP.Quan, data=headache)
summary(fit)
```

```
##
```

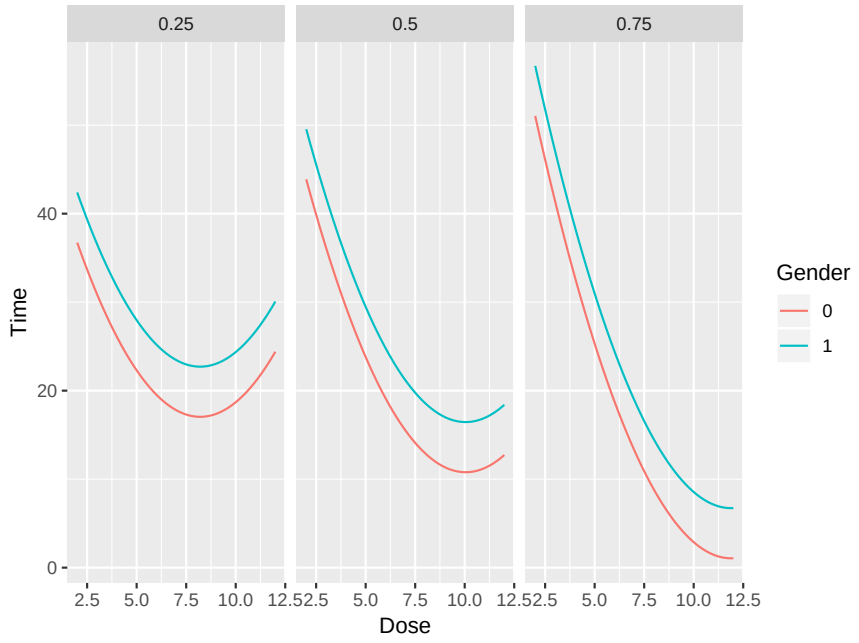
```
## Call:
## lm(formula = Time ~ . + Dose:BP.Quan, data = headache)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.4706 -2.1795  0.2083  2.7819  9.5490
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   40.5408     6.5253   6.213 7.31e-06
## Dose          -6.5059     1.6792  -3.874 0.001111
## Sex            5.6667     1.8258   3.104 0.006130
## BP.Quan       43.6765    10.2329   4.268 0.000463
## Dose2          0.5111     0.1217   4.199 0.000539
## Dose:BP.Quan  -7.5294     1.5340  -4.908 0.000113
##
## Residual standard error: 4.472 on 18 degrees of freedom
## Multiple R-squared:  0.9262, Adjusted R-squared:  0.9058
## F-statistic: 45.21 on 5 and 18 DF,  p-value: 1.437e-09
```

and now all terms are significant.

What does all of this look like?

```
x <- seq(2, 12, length=100)
y <- 0:1
z <- c(0.25, 0.5, 0.75)
xy <- expand.grid(x, y, z)
df <- data.frame(Dose=xy[, 1], Dose2=xy[, 1]^2,
                 Sex=xy[, 2], BP.Quan=xy[, 3])
df$Time <- predict(fit, df)
```

```
ggplot(data=df, aes(Dose, Time, color=factor(Sex))) +
  geom_line() +
  facet_wrap(~factor(BP.Quan)) +
  labs(color="Gender")
```



and so we can give the following advice:

- the same dosage will work for men and women
- for people with low blood pressure give 7.5mg
- for people with medium blood pressure give 11mg
- for people with high blood pressure give 9mg

Generalized Additive Models

For a linear regression we have a dependent variable Y and a set of predictors $x_1, .., x_n$ and a model of the form

$$Y = \alpha + \sum \beta_j x_j + \epsilon$$

Generalized additive models extend this in two ways: first we replace the linear terms $\beta_j x_j$ by non-linear functions, to get

$$Y = \alpha + \sum f(x_j; \beta_j) + \epsilon$$

Second we can take the same step as before in going from linear models to general linear models to fit problems where Y is a categorical variable.

A special case we have already discussed is where the functions are polynomials.

We can also "mix" linear and generalized additive models. Consider

$$Y = \alpha + \beta_1 x_1 + f(x_2; \beta_2) + \epsilon$$

here we have a model linear in X1 and additive in X2. Such a model is called "semiparametric".

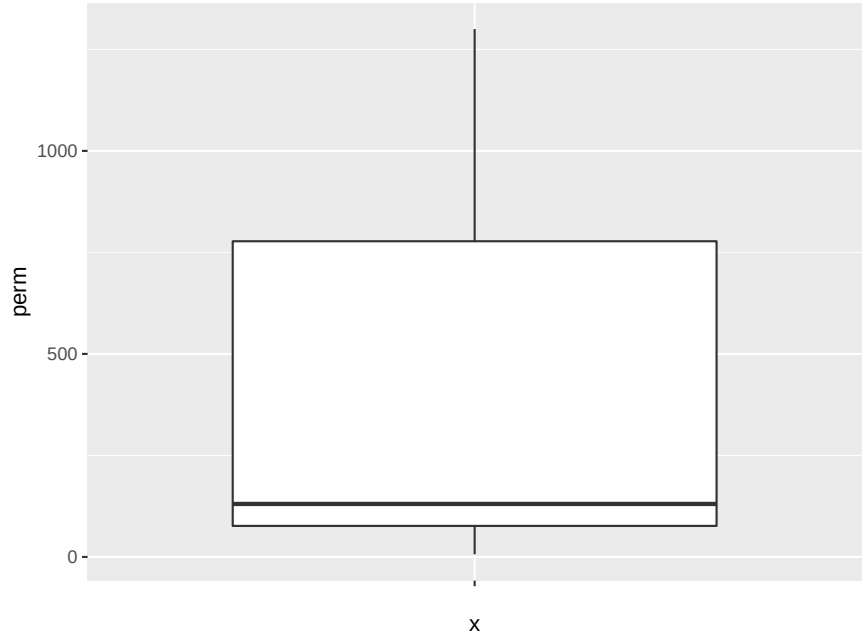
Example: Oil in Rocks

We have measurements on four cross-sections of each of 12 oil-bearing rocks. The measurements are the end product of a complex image-analysis and represent the total area, total perimeter and a measure of 'roundness' of the pores in the rock cross-section.

```
kable.nice(head(rock.oil))
```

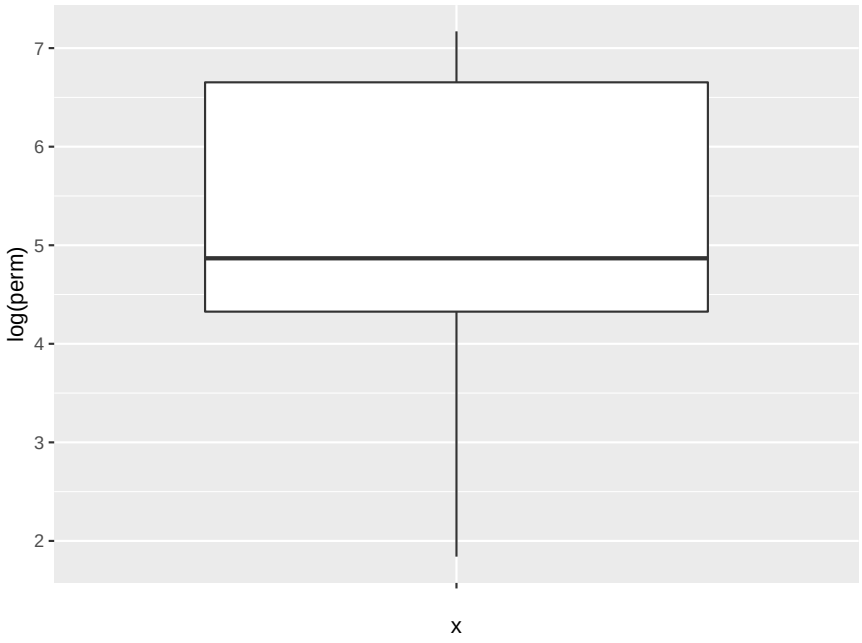
area	peri	shape	perm
4990	2791.90	0.0903296	6.3
7002	3892.60	0.1486220	6.3
7558	3930.66	0.1833120	6.3
7352	3869.32	0.1170630	6.3
7943	3948.54	0.1224170	17.1
7979	4010.15	0.1670450	17.1

```
ggplot(data=rock.oil, aes("", perm)) +
  geom_boxplot()
```



this looks a bit skewed, so we should try a log transform:

```
ggplot(data=rock.oil, aes("", log(perm))) +
  geom_boxplot()
```

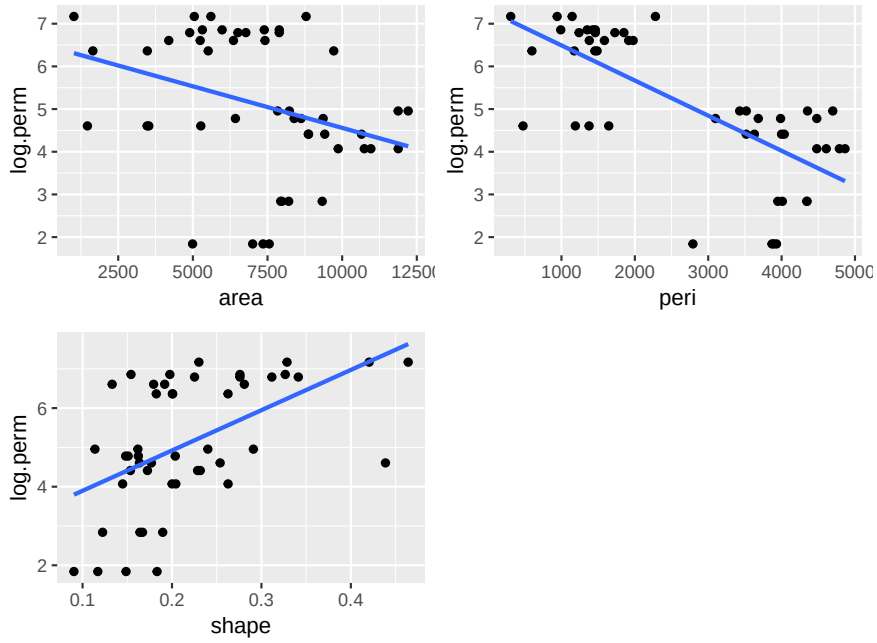


and that looks better, so

```
rock.oil$perm <- log(rock.oil$perm)
colnames(rock.oil)[4] <- "log.perm"
```

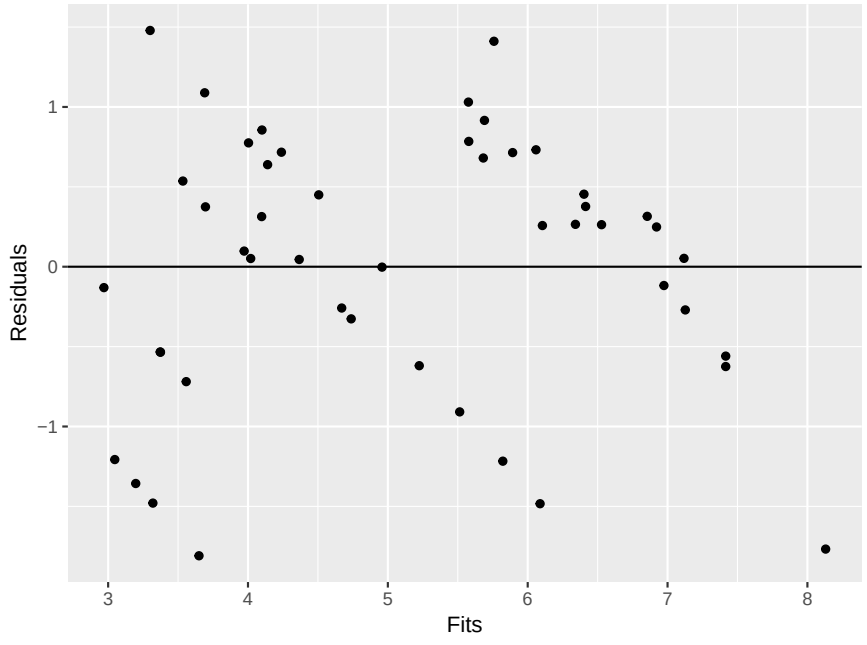
Next the predictors:

```
pushViewport(viewport(layout = grid.layout(2, 2)))
print(ggplot(data=rock.oil, aes(area, log.perm)) +
           geom_point() +
           geom_smooth(method = "lm", se=FALSE),
    vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(ggplot(data=rock.oil, aes(peri, log.perm)) +
           geom_point() +
           geom_smooth(method = "lm", se=FALSE),
    vp=viewport(layout.pos.row=1, layout.pos.col=2))
print(ggplot(data=rock.oil, aes(shape, log.perm)) +
           geom_point() +
           geom_smooth(method = "lm", se=FALSE),
    vp=viewport(layout.pos.row=2, layout.pos.col=1))
```



we begin with a linear model:

```
fit.lin <- lm(log.perm~., data=rock.oil)
df <- data.frame(Residuals=resid(fit.lin),
            Fits = fitted(fit.lin))
ggplot(data=df, aes(Fits, Residuals)) +
            geom_point() +
            geom_hline(yintercept = 0)
```



and that is not so bad.

Next let's fit the generalized additive model:

```
library(mgcv)
fit.gam <- gam(log.perm ~ s(area) + s(peri) + s(shape),
               data=rock.oil)
```

Notice the terms s() which means we are using splines.

Is this model better than the simple linear one? We can compare the two using ANOVA, done in

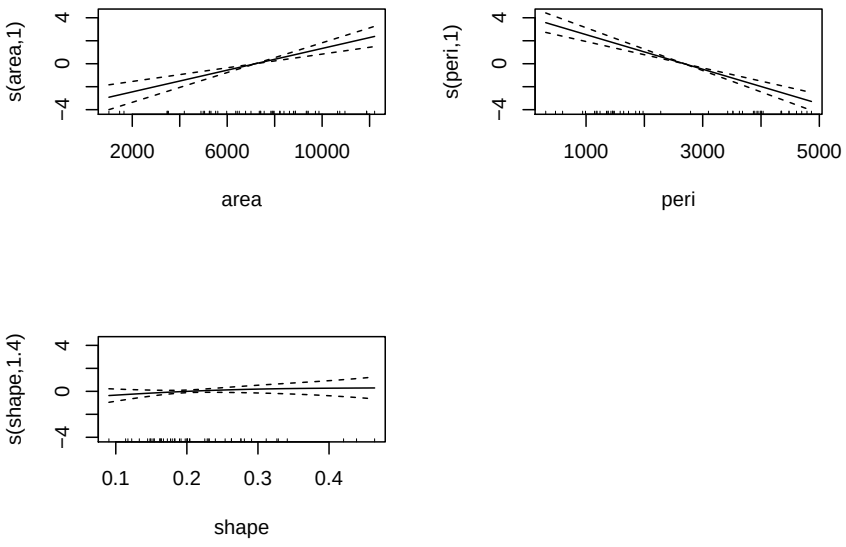
```
anova(fit.lin, fit.gam)
```

```
## Analysis of Variance Table
##
## Model 1: log.perm ~ area + peri + shape
## Model 2: log.perm ~ s(area) + s(peri) + s(shape)
##   Res.Df    RSS      Df Sum of Sq      F Pr(>F)
## 1 44.000 31.949
## 2 43.598 31.230 0.40237   0.71914 2.4951  0.125
```

It appears the more complicated model is not actually better than the old one (p=0.125).

What is this new model? In

```
par(mfrow=c(2, 2))
plot(fit.gam, se = TRUE)
```



we see the fitted line plots, which do look fairly linear.

Example: Kyphosis

This data set is about Kyphosis, a spinal deformity in children that occurs after certain surgeries on the spine.

The variables are:

1) Kyphosis: 1 if kyphosis is present, 0 otherwise.

2) Age: age of the child in month.

3) Number: the number of vertebrae involved in the spinal operation.

4) Start: the beginning of the range of the vertebrae involved in the spinal operation.

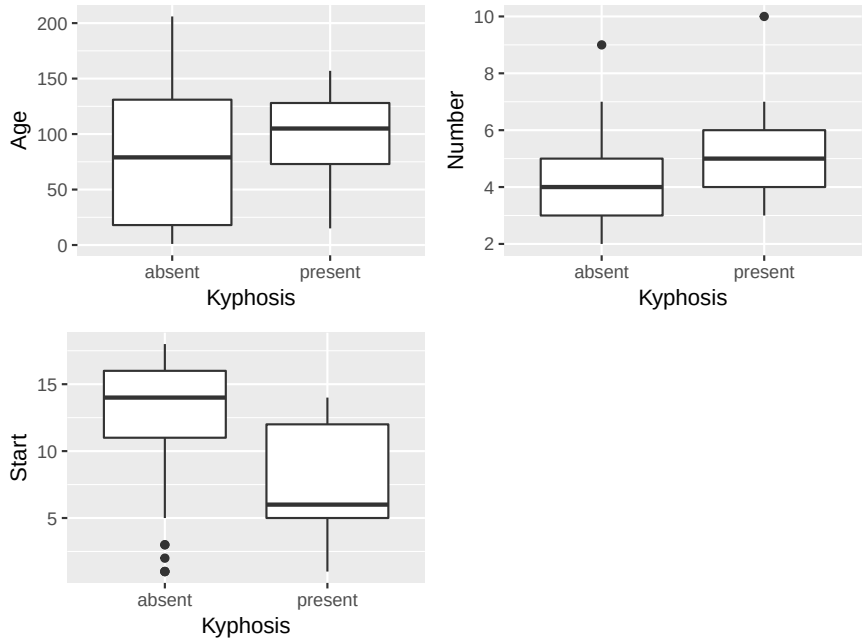
```
kable.nice(head(kyphosis))
```

Kyphosis	Age	Number	Start
absent	71	3	5
absent	158	3	14
present	128	4	5
absent	2	5	1
absent	1	4	15
absent	1	2	16

the goal is to predict whether a child will develop kyphosis. So this is a binary outcome, and we will use logistic regression.

Let's begin with some box plots:

```
pushViewport(viewport(layout = grid.layout(2, 2)))
print(ggplot(data=kyphosis, aes(Kyphosis, Age)) +
          geom_boxplot(),
    vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(ggplot(data=kyphosis, aes(Kyphosis, Number)) +
          geom_boxplot(),
      vp=viewport(layout.pos.row=1, layout.pos.col=2))
print(ggplot(data=kyphosis, aes(Kyphosis, Start)) +
          geom_boxplot(),
      vp=viewport(layout.pos.row=2, layout.pos.col=1))
```



so it seems all predictors are useful.

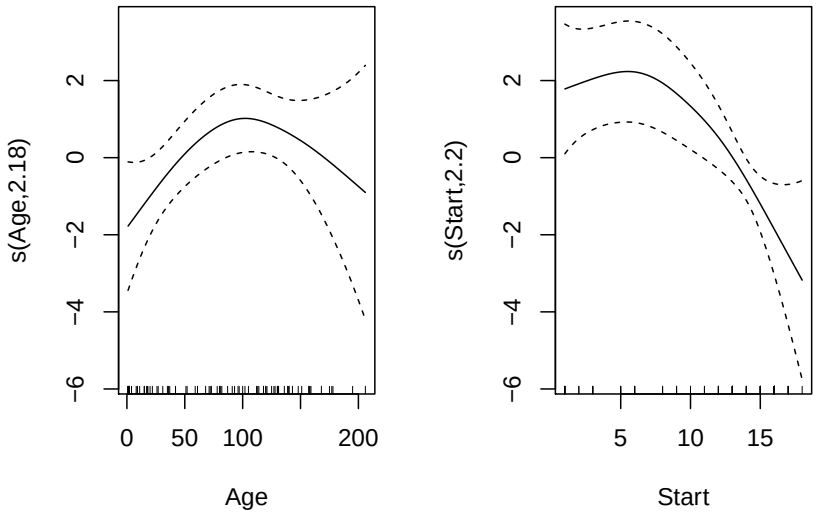
```
fit.glm <- glm(Kyphosis ~ ., family = binomial,
               data = kyphosis)
summary(fit.glm)
```

```
##
## Call:
## glm(formula = Kyphosis ~ ., family = binomial, data = kyphosis)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3124  -0.5484  -0.3632  -0.1659   2.1613
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.036934   1.449575  -1.405  0.15996
## Age          0.010930   0.006446   1.696  0.08996
## Number       0.410601   0.224861   1.826  0.06785
## Start       -0.206510   0.067699  -3.050  0.00229
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 83.234  on 80  degrees of freedom
## Residual deviance: 61.380  on 77  degrees of freedom
## AIC: 69.38
##
## Number of Fisher Scoring iterations: 5
```

which suggests that only Start is strongly predictive.

It is possible to show that Number is not very useful here (using anova), and we will continue with Age and Start:

```
fit.sa.gam <- gam(Kyphosis ~ s(Age) + s(Start),
                  family = binomial, data = kyphosis)
par(mfrow = c(1, 2))
plot(fit.sa.gam, se = TRUE)
```



From this it seems a model quadratic in Age might work. For Start we see that its spline appears piecewise linear, flat up to about 12 and then with a negative slope. This also makes sense from the background of the data, because values of Start up to 12 correspond to the thoracic region of the spine and values greater than 12 belong to the lumbar region. We will therefore try and fit a model of the form

$$f(x) = a + b(x - 12)I_{[12,\infty)}(x)$$

Notice that this model fits a continuous function. In R we can do this by including a term I((Start-12)*(Start>12)). The 'I' is needed so R does not interpret the '*' as meaning interaction. Comparing this with the gam model we see that this model is as good as the generalized additive one.

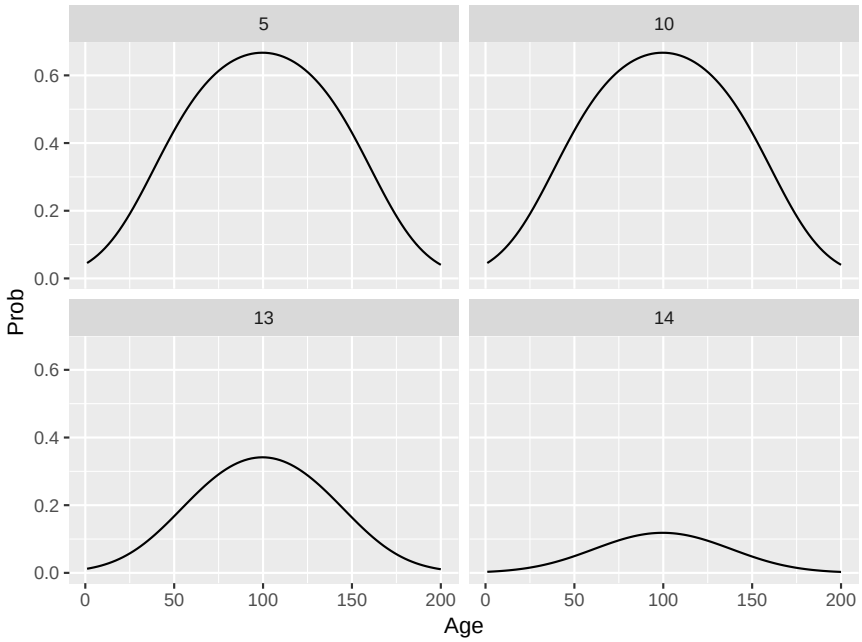
```
fit.sa.glm <- glm(Kyphosis ~ poly(Age, 2) +
                    I((Start - 12) * (Start > 12)),
                  family = binomial, data = kyphosis)
anova(fit.sa.gam, fit.sa.glm)
```

```
## Analysis of Deviance Table
##
## Model 1: Kyphosis ~ s(Age) + s(Start)
```

```
## Model 2: Kyphosis ~ poly(Age, 2) + I((Start - 12) * (Start > 12))
##   Resid. Df Resid. Dev      Df Deviance
## 1    74.498     52.383
## 2    81.000     51.953 -6.5018   0.4301
```

What does this new model look like?

```
x <- seq(1, 200, length = 100)
y <-  c(5, 10, 13, 14)
xy <- expand.grid(x, y)
df <- data.frame(Age=xy[, 1], Start=xy[, 2])
df$Prob <- predict(fit.sa.glm, df, type = "response")
ggplot(df, aes(Age, Prob)) +
  geom_line() +
  facet_wrap(~Start)
```



where we can see that the highest risk of kyphosis is for children around age 100 month (aka 8 years) but it diminishes the higher up the Start is.

Subset Selection and Ridge Regression

Subset Selection

We have previously discussed the issue of subset selection. There we use Mallow's C_p Statistic to find the best model. This calculates all possible models, and if there are k predictors there are 2^k such models. Although there are very fast algorithms available for this, it is not feasible to do it for much more than 30 predictors. So what do we do if we have more than that?

- **Forward/Backward Selection**

One idea is as follows:

1. Fit the model with no predictors.
2. Find which predictor improves the model the most (somehow)
3. If this predictor improves the fit statistically significantly, add it and go back to 2.
4. Stop

There are routines in R to do these steps.

Example: Pollution and Mortality

First we need to fix the non-normal predictors:

```
newair <- airpollution[, -16] #take out NOxPot
newair[, c(10, 13, 14)] <- log(newair[, c(10, 13, 14)])
colnames(newair)[c(10, 13, 14)] <-
  c("log.Pop", "log.HCPot", "log.NOx")
```

Next we fit the model with no predictor. Of course that just finds the mean of Mortality, but we need the corresponding *lm* object.

```
fit <- lm(Mortality~1, data=newair)
```

How do we decide which predictor (if any) to add to the model? We can use the *add1* command and the so called F statistic:

```
add1(fit, formula(newair), test="F")
```

```
## Single term additions
##
## Model:
## Mortality ~ 1
##             Df Sum of Sq    RSS    AIC F value    Pr(>F)
## <none>                   225993 488.79
## JanTemp      1        58 225935 490.78  0.0145 0.9045520
## JulyTemp     1     23407 202586 484.34  6.5858 0.0129321
## RelHum       1      2309 223684 490.19  0.5883 0.4462331
## Rain         1     42393 183599 478.54 13.1614 0.0006117
## Education    1     58340 167652 473.17 19.8352 3.991e-05
## PopDensity   1     14365 211627 486.92  3.8691 0.0540559
## NonWhite     1     94473 131520 458.85 40.9440 3.172e-08
## WhiteCollar  1     18920 207072 485.63  5.2081 0.0262337
## log.Pop      1      1646 224347 490.36  0.4182 0.5204471
## Pop.House    1     30608 195385 482.21  8.9292 0.0041348
## Income       1     18138 207855 485.86  4.9739 0.0296867
## log.HCPot    1      3553 222440 489.86  0.9103 0.3440525
## log.NOx      1     17696 208296 485.98  4.8425 0.0318330
## SO2Pot       1     39698 186295 479.40 12.1462 0.0009533
```

so the predictor with the highest F statistics (40.9) is NonWhite and it is statistically significant (p=0.000), so we add it to the list of predictors:

```
fit <- update(fit, .~.+NonWhite)
coef(fit)
```

```
## (Intercept)    NonWhite
##  887.901783    4.485521
```

Here and in what follows I use the F statistic as a criterion. There are others, some included in the *add1* routine such as *AIC* or *Akaike's information criterion*. Which criterion to use is a rather tricky question.

Next:

```
tmp <- add1(fit, formula(newair), test="F")
k <- seq_along(tmp[[5]][-1])[tmp[[5]][-1] ==max(tmp[[5]][-1])]
cat(rownames(tmp)[k+1], ", F = ", tmp[[5]][k+1], ", p value = ", tmp[[6]][k+1],"\n")
```

```
## Education , F =  18.66899 , p value =  6.428038e-05
```

```
fit <- update(fit, .~.+Education)
tmp <- add1(fit, formula(newair), test="F")
k <- seq_along(tmp[[5]][-1])[tmp[[5]][-1] ==max(tmp[[5]][-1])]
cat(rownames(tmp)[k+1], ", F = ", tmp[[5]][k+1], ", p value = ", tmp[[6]][k+1],"\n")
```

```
## JanTemp , F =  11.04183 , p value =  0.001588636
```

```
fit <- update(fit, .~.+JanTemp)
tmp <- add1(fit, formula(newair), test="F")
k <- seq_along(tmp[[5]][-1])[tmp[[5]][-1] ==max(tmp[[5]][-1])]
cat(rownames(tmp)[k+1], ", F = ", tmp[[5]][k+1], ", p value = ", tmp[[6]][k+1],"\n")
```

```
## SO2Pot , F =  7.428167 , p value =  0.008637386
```

```
fit <- update(fit, .~.+SO2Pot)
tmp <- add1(fit, formula(newair), test="F")
k <- seq_along(tmp[[5]][-1])[tmp[[5]][-1] ==max(tmp[[5]][-1])]
cat(rownames(tmp)[k+1], ", F = ", tmp[[5]][k+1], ", p value = ", tmp[[6]][k+1],"\n")
```

```
## Rain , F =  5.21367 , p value =  0.02644668
```

```
fit <- update(fit, .~.+Rain)
tmp <- add1(fit, formula(newair), test="F")
k <- seq_along(tmp[[5]][-1])[tmp[[5]][-1] ==max(tmp[[5]][-1])]
cat(rownames(tmp)[k+1], ", F = ", tmp[[5]][k+1], ", p value = ", tmp[[6]][k+1],"\n")
```

```
## log.NOx , F =  4.872501 , p value =  0.03172464
```

```
fit <- update(fit, .~.+log.NOx)
tmp <- add1(fit, formula(newair), test="F")
k <- seq_along(tmp[[5]][-1])[tmp[[5]][-1] ==max(tmp[[5]][-1])]
cat(rownames(tmp)[k+1], ", F = ", tmp[[5]][k+1], ", p value = ", tmp[[6]][k+1],"\n")
```

```
## WhiteCollar , F =  1.959068 , p value =  0.1676667
```

and the next predictor is not stat. significant, so we stop.

Notice that this is **not** the same model that we found using Mallow's C_p, which used JanTemp, Rain, PopDensity, NonWhite, WhiteCollar and LOGT(NOx).

An alternative to forward selection is its reverse, backward selection. Here we start with the full model and remove predictors until there are only significant predictors left:

Here is the backward solution:

```
fit <- lm(Mortality~., data=newair)
drop1(fit, test="F")
```

```
## Single term deletions
##
## Model:
## Mortality ~ JanTemp + JulyTemp + RelHum + Rain + Education +
##     PopDensity + NonWhite + WhiteCollar + log.Pop + Pop.House +
##     Income + log.HCPot + log.NOx + SO2Pot
##             Df Sum of Sq   RSS    AIC F value    Pr(>F)
## <none>                   51920 430.02
## JanTemp      1      7767 59687 436.24  6.5821   0.01379
## JulyTemp     1       946 52867 429.08  0.8018   0.37542
## RelHum       1       325 52246 428.38  0.2757   0.60215
## Rain         1      7070 58990 435.55  5.9912   0.01844
## Education    1       853 52773 428.98  0.7228   0.39984
## PopDensity   1      1112 53032 429.27  0.9422   0.33701
## NonWhite     1     31909 83830 456.28 27.0417 4.965e-06
## WhiteCollar  1      2637 54558 430.94  2.2348   0.14207
## log.Pop      1       162 52082 428.20  0.1369   0.71313
## Pop.House    1       991 52911 429.13  0.8394   0.36456
## Income       1       245 52165 428.29  0.2076   0.65087
## log.HCPot    1      2636 54557 430.94  2.2342   0.14213
## log.NOx      1      4750 56670 433.18  4.0253   0.05099
## SO2Pot       1       690 52611 428.80  0.5850   0.44843
```

now the predictor with the smallest F value is log.Pop, so we drop it:

```
fit <- update(fit, .~.-log.Pop)
tmp <- drop1(fit, test="F")
k <- seq_along(tmp[[5]][-1])[tmp[[5]][-1] ==min(tmp[[5]][-1])]
cat(rownames(tmp)[k+1], ", F = ", tmp[[5]][k+1], ", p value = ", tmp[[6]][k+1],"\n")
```

```
## Income , F =  0.1531485 , p value =  0.6973914
```

```
fit <- update(fit, .~.-Income)
tmp <- drop1(fit, test="F")
```

```
k <- seq_along(tmp[[5]][-1])[tmp[[5]][-1] ==min(tmp[[5]][-1])]
cat(rownames(tmp)[k+1], ", F = ", tmp[[5]][k+1], ", p value = ", tmp[[6]][k+1],"\n")
```

```
## RelHum , F =  0.272645 , p value =  0.6040685
```

```
fit <- update(fit, .~.-RelHum)
tmp <- drop1(fit, test="F")
k <- seq_along(tmp[[5]][-1])[tmp[[5]][-1] ==min(tmp[[5]][-1])]
cat(rownames(tmp)[k+1], ", F = ", tmp[[5]][k+1], ", p value = ", tmp[[6]][k+1],"\n")
```

```
## SO2Pot , F =  0.6219361 , p value =  0.4342887
```

```
fit <- update(fit, .~.-SO2Pot)
tmp <- drop1(fit, test="F")
k <- seq_along(tmp[[5]][-1])[tmp[[5]][-1] ==min(tmp[[5]][-1])]
cat(rownames(tmp)[k+1], ", F = ", tmp[[5]][k+1], ", p value = ", tmp[[6]][k+1],"\n")
```

```
## JulyTemp , F =  1.110323 , p value =  0.2972872
```

```
fit <- update(fit, .~.-JulyTemp)
tmp <- drop1(fit, test="F")
k <- seq_along(tmp[[5]][-1])[tmp[[5]][-1] ==min(tmp[[5]][-1])]
cat(rownames(tmp)[k+1], ", F = ", tmp[[5]][k+1], ", p value = ", tmp[[6]][k+1],"\n")
```

```
## PopDensity , F =  0.9241217 , p value =  0.3411151
```

```
fit <- update(fit, .~.-PopDensity)
tmp <- drop1(fit, test="F")
k <- seq_along(tmp[[5]][-1])[tmp[[5]][-1] ==min(tmp[[5]][-1])]
cat(rownames(tmp)[k+1], ", F = ", tmp[[5]][k+1], ", p value = ", tmp[[6]][k+1],"\n")
```

```
## log.HCPot , F =  1.562185 , p value =  0.2171643
```

```
fit <- update(fit, .~.-log.HCPot)
tmp <- drop1(fit, test="F")
k <- seq_along(tmp[[5]][-1])[tmp[[5]][-1] ==min(tmp[[5]][-1])]
cat(rownames(tmp)[k+1], ", F = ", tmp[[5]][k+1], ", p value = ", tmp[[6]][k+1],"\n")
```

```
## Pop.House , F =  1.678198 , p value =  0.2009972
```

```
fit <- update(fit, .~.-Pop.House)
tmp <- drop1(fit, test="F")
k <- seq_along(tmp[[5]][-1])[tmp[[5]][-1] ==min(tmp[[5]][-1])]
cat(rownames(tmp)[k+1], ", F = ", tmp[[5]][k+1], ", p value = ", tmp[[6]][k+1],"\n")
```

```
## WhiteCollar , F =  1.928731 , p value =  0.1708173
```

```
fit <- update(fit, .~.-WhiteCollar)
tmp <- drop1(fit, test="F")
k <- seq_along(tmp[[5]][-1])[tmp[[5]][-1] ==min(tmp[[5]][-1])]
cat(rownames(tmp)[k+1], ", F = ", tmp[[5]][k+1], ", p value = ", tmp[[6]][k+1],"\n")
```

```
## Education , F =  6.21418 , p value =  0.01583229
```

and now the p value is less than 0.05, so we stop.

This results in a model with predictors

```
rownames(tmp)[-1]
```

```
## [1] "JanTemp"   "Rain"      "Education" "NonWhite"  "log.NOx"
```

which is not the same as either best subset or forward selection.

- **stepwise selection**

Here in each step we either add or drop a variable. The *step* command does it for us. Notice that it uses AIC by default.

```
fit <- lm(Mortality~., data=newair)
step(fit)
```

```
## Start:  AIC=430.02
## Mortality ~ JanTemp + JulyTemp + RelHum + Rain + Education +
##     PopDensity + NonWhite + WhiteCollar + log.Pop + Pop.House +
##     Income + log.HCPot + log.NOx + SO2Pot
##
##                Df Sum of Sq   RSS    AIC
## - log.Pop       1       162 52082 428.20
## - Income        1       245 52165 428.29
## - RelHum        1       325 52246 428.38
## - SO2Pot        1       690 52611 428.80
## - Education     1       853 52773 428.98
## - JulyTemp      1       946 52867 429.08
## - Pop.House     1       991 52911 429.13
## - PopDensity    1      1112 53032 429.27
## <none>                      51920 430.02
## - log.HCPot     1      2636 54557 430.94
## - WhiteCollar   1      2637 54558 430.94
## - log.NOx       1      4750 56670 433.18
## - Rain          1      7070 58990 435.55
## - JanTemp       1      7767 59687 436.24
## - NonWhite      1     31909 83830 456.28
##
## Step:  AIC=428.2
## Mortality ~ JanTemp + JulyTemp + RelHum + Rain + Education +
##     PopDensity + NonWhite + WhiteCollar + Pop.House + Income +
##     log.HCPot + log.NOx + SO2Pot
##
##                Df Sum of Sq   RSS    AIC
## - Income        1       177 52259 426.40
## - RelHum        1       324 52406 426.57
## - Education     1       755 52837 427.05
```

```
## - SO2Pot       1       810 52892 427.11
## - JulyTemp     1       865 52947 427.17
## - Pop.House    1      1098 53180 427.43
## - PopDensity   1      1206 53288 427.55
## <none>                     52082 428.20
## - WhiteCollar  1      2622 54704 429.10
## - log.HCPot    1      2731 54813 429.21
## - log.NOx      1      5135 57217 431.75
## - Rain         1      6912 58994 433.55
## - JanTemp      1      7637 59719 434.27
## - NonWhite     1     32637 84719 454.90
##
## Step:  AIC=426.4
## Mortality ~ JanTemp + JulyTemp + RelHum + Rain + Education +
##     PopDensity + NonWhite + WhiteCollar + Pop.House + log.HCPot +
##     log.NOx + SO2Pot
##
##               Df Sum of Sq   RSS    AIC
## - RelHum       1       310 52569 424.75
## - SO2Pot       1       740 52999 425.23
## - JulyTemp     1       848 53107 425.35
## - Education    1      1110 53370 425.64
## - Pop.House    1      1128 53387 425.66
## - PopDensity   1      1198 53457 425.74
## <none>                     52259 426.40
## - log.HCPot    1      2631 54890 427.30
## - WhiteCollar  1      2825 55084 427.51
## - log.NOx      1      4995 57254 429.79
## - Rain         1      7196 59455 432.01
## - JanTemp      1      8281 60540 433.08
## - NonWhite     1     33147 85406 453.38
##
## Step:  AIC=424.75
## Mortality ~ JanTemp + JulyTemp + Rain + Education + PopDensity +
##     NonWhite + WhiteCollar + Pop.House + log.HCPot + log.NOx +
##     SO2Pot
##
##               Df Sum of Sq   RSS    AIC
## - SO2Pot       1       696 53265 423.52
## - Education    1      1066 53635 423.93
## - PopDensity   1      1116 53685 423.99
## - Pop.House    1      1128 53697 424.00
## - JulyTemp     1      1635 54204 424.55
## <none>                     52569 424.75
## - log.HCPot    1      2631 55200 425.63
## - WhiteCollar  1      2844 55413 425.86
```

```
## - log.NOx      1      4934 57503 428.04
## - Rain         1      7628 60197 430.74
## - JanTemp      1      7983 60552 431.09
## - NonWhite     1     34608 87177 452.59
##
## Step:  AIC=423.52
## Mortality ~ JanTemp + JulyTemp + Rain + Education + PopDensity +
##     NonWhite + WhiteCollar + Pop.House + log.HCPot + log.NOx
##
##               Df Sum of Sq   RSS    AIC
## - JulyTemp     1      1232 54497 422.87
## - PopDensity   1      1331 54595 422.98
## - Education    1      1350 54615 423.00
## - Pop.House    1      1540 54804 423.21
## <none>                     53265 423.52
## - log.HCPot    1      2651 55916 424.39
## - WhiteCollar  1      3186 56451 424.95
## - log.NOx      1      7406 60671 429.21
## - Rain         1      7719 60984 429.51
## - JanTemp      1     11176 64441 432.76
## - NonWhite     1     34745 88009 451.15
##
## Step:  AIC=422.87
## Mortality ~ JanTemp + Rain + Education + PopDensity + NonWhite +
##     WhiteCollar + Pop.House + log.HCPot + log.NOx
##
##               Df Sum of Sq   RSS    AIC
## - PopDensity   1      1028 55525 421.98
## - Education    1      1146 55643 422.10
## - Pop.House    1      1442 55939 422.41
## - log.HCPot    1      1609 56106 422.59
## <none>                     54497 422.87
## - WhiteCollar  1      3737 58234 424.79
## - log.NOx      1      6565 61062 427.58
## - Rain         1      8311 62808 429.25
## - JanTemp      1     13524 68021 433.95
## - NonWhite     1     41036 95533 453.99
##
## Step:  AIC=421.98
## Mortality ~ JanTemp + Rain + Education + NonWhite + WhiteCollar +
##     Pop.House + log.HCPot + log.NOx
##
##               Df Sum of Sq   RSS    AIC
## - log.HCPot    1      1735 57259 421.79
## <none>                     55525 421.98
## - Pop.House    1      2373 57898 422.44
```

```
## - Education    1      2558 58082 422.63
## - WhiteCollar  1      2763 58288 422.84
## - log.NOx      1      7827 63351 427.76
## - Rain         1      8886 64410 428.73
## - JanTemp      1     16942 72466 435.69
## - NonWhite     1     41873 97398 453.13
##
## Step:  AIC=421.79
## Mortality ~ JanTemp + Rain + Education + NonWhite + WhiteCollar +
##     Pop.House + log.NOx
##
##               Df Sum of Sq   RSS    AIC
## - Pop.House    1      1884 59143 421.70
## <none>                     57259 421.79
## - WhiteCollar  1      2609 59868 422.42
## - Education    1      3574 60834 423.36
## - Rain         1     11398 68658 430.50
## - log.NOx      1     16778 74037 434.95
## - JanTemp      1     18282 75541 436.14
## - NonWhite     1     41479 98739 451.94
##
## Step:  AIC=421.7
## Mortality ~ JanTemp + Rain + Education + NonWhite + WhiteCollar +
##     log.NOx
##
##               Df Sum of Sq    RSS    AIC
## <none>                      59143 421.70
## - WhiteCollar  1      2194  61337 421.85
## - Education    1      2621  61764 422.26
## - Rain         1     13263  72406 431.64
## - JanTemp      1     17593  76736 435.07
## - log.NOx      1     20607  79750 437.34
## - NonWhite     1     48049 107192 454.79
##
## Call:
## lm(formula = Mortality ~ JanTemp + Rain + Education + NonWhite +
##     WhiteCollar + log.NOx, data = newair)
##
## Coefficients:
## (Intercept)      JanTemp         Rain    Education     NonWhite
##    1031.949       -2.023        1.812      -10.746        4.040
## WhiteCollar      log.NOx
##      -1.451       19.248
```

this seems the easiest to use (certainly in terms of what we have to type into R) but it is important to understand that none of these methods works all the time. For each of them

there are examples were they lead to quite bad final models.

There has been a lot of research on the relative merits of these methods. There are in fact many Statisticians who advise against their use. As an alternative we can consider

Ridge Regression

One problem with the above methods is that they are all or nothing: a predictor either is in the final model or is not. Ridge regression takes a different approach: each variable gets a "weight" in how much it contributes to the final model.

Recall the least squares regression method: we minimize the least squares criterion

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j-1}^{k}\beta_j x_{ij}\right)^2$$

in ridge regression we use the criterion

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j-1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j-1}^{p}\beta_j^2$$

What does this do? The term $\sum_{j-1}^{p}\beta_j^2$ will depend mainly on the largest β's. Because this term is added in and we are minimizing this expression we essentially *penalize* large β's. Overall these coefficients will be *shrunk* towards 0. For this reason ridge regression is a *shrinkage* method. Such method have become quite popular in many areas of Statistics.

In the literature such methods are also refereed to as *penalized likelihood* methods.

λ is a parameter that controls the amount of shrinkage. If $\lambda = 0$ we are back at OLS.

How do we fit such a model?

```
library(ridge)
fit <- linearRidge(Mortality~., data=newair)
summary(fit)
```

```
##
## Call:
## linearRidge(formula = Mortality ~ ., data = newair)
##
##
## Coefficients:
##               Estimate Scaled estimate Std. Error (scaled)
## (Intercept)  8.750e+02              NA                   NA
## JanTemp     -9.412e-02      -7.277e+00            1.082e+01
## JulyTemp     6.604e-01       2.314e+01            9.865e+00
## RelHum      -4.164e-02      -1.706e+00            1.161e+01
## Rain         4.772e-01       4.206e+01            1.056e+01
```

```
## Education   -6.277e+00      -4.067e+01           1.023e+01
## PopDensity   2.059e-03       2.261e+01           1.121e+01
## NonWhite     9.271e-01       6.353e+01           1.025e+01
## WhiteCollar -6.699e-01      -2.586e+01           1.113e+01
## log.Pop      1.466e+00       9.127e+00           1.057e+01
## Pop.House    2.177e+01       3.032e+01           1.094e+01
## Income      -6.439e-04      -2.194e+01           1.096e+01
## log.HCPot    1.458e+00       1.253e+01           8.524e+00
## log.NOx      3.090e+00       2.796e+01           8.551e+00
## SO2Pot       7.871e-02       3.810e+01           1.024e+01
##             t value (scaled) Pr(>|t|)
## (Intercept)               NA       NA
## JanTemp                0.672 0.501310
## JulyTemp               2.346 0.018970
## RelHum                 0.147 0.883210
## Rain                   3.982 6.84e-05
## Education              3.976 7.01e-05
## PopDensity             2.016 0.043760
## NonWhite               6.198 5.73e-10
## WhiteCollar            2.325 0.020082
## log.Pop                0.864 0.387804
## Pop.House              2.770 0.005599
## Income                 2.001 0.045437
## log.HCPot              1.470 0.141584
## log.NOx                3.270 0.001077
## SO2Pot                 3.720 0.000199
##
## Ridge parameter: 2.861264, chosen automatically, computed using 1 PCs
##
## Degrees of freedom: model 2.995 , variance 1.031 , residual 4.959
```

Lasso

This is similar to ridge regression but it uses

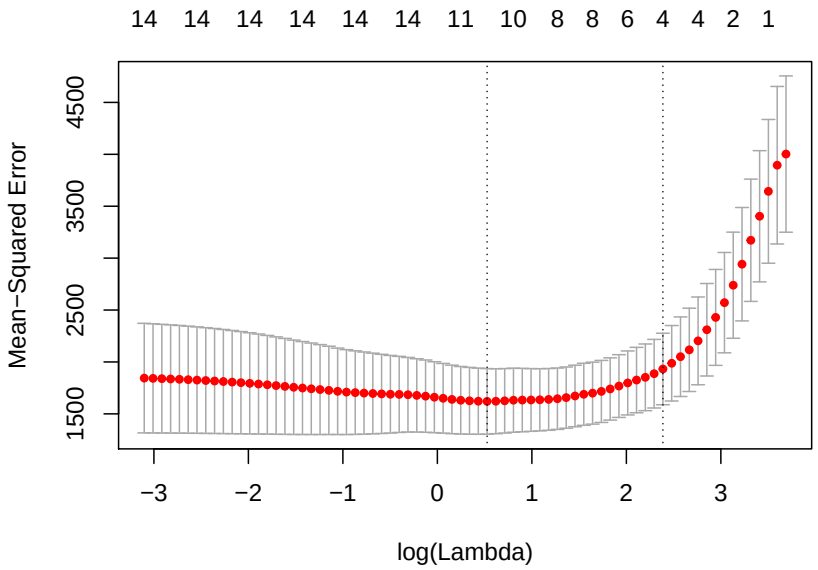
$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j-1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j-1}^{p}|\beta_j|$$

In modern terminology, the Lasso uses an L^1 penalty whereas ridge regression uses L^2.

The Lasso can be fit with

```
library(glmnet)
X <- data.matrix(newair[, -1])
y <- newair$Mortality
fit <- cv.glmnet(X, y, standardize=TRUE,
```

```
                type.measure="mse", nfolds = 5, alpha=1)
plot(fit)
```




```
cf <- as.numeric(coef(fit, s=fit$lambda.1se))
names(cf) <- c("Intercept", colnames(X))
cf
```

```
##    Intercept      JanTemp     JulyTemp       RelHum         Rain
## 1032.9753025    0.0000000    0.0000000    0.0000000    0.3963679
##    Education   PopDensity     NonWhite  WhiteCollar      log.Pop
##  -13.3991939    0.0000000    2.6816158    0.0000000    0.0000000
##    Pop.House       Income    log.HCPot      log.NOx       SO2Pot
##    0.0000000    0.0000000    0.0000000    0.0000000    0.1467875
```

One advantage of the lasso is that it can yield coefficients that are 0, and clearly any predictor whose coefficient is 0 can be dropped:

```
cf[abs(cf)>0]
```

```
##    Intercept         Rain    Education     NonWhite       SO2Pot
## 1032.9753025    0.3963679  -13.3991939    2.6816158    0.1467875
```

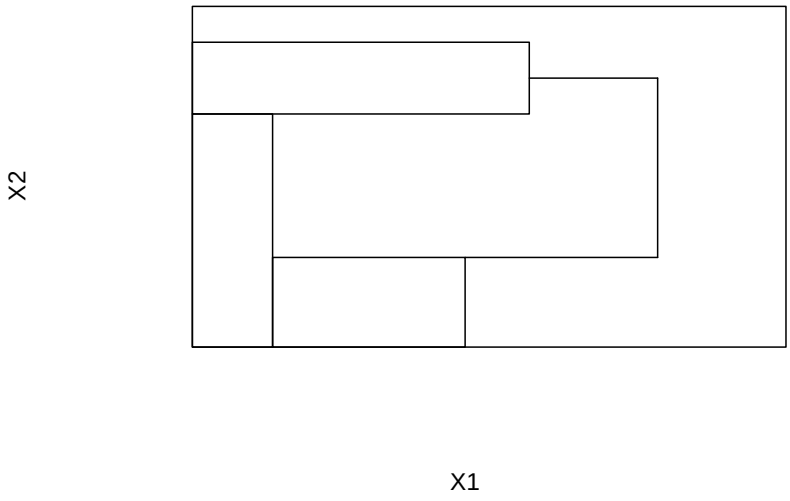
As a very general guideline, if the goal is subset selection use the lasso. If the goal is prediction use ridge regression.

Regression Trees

Tree-based methods partition the feature space into a set of rectangles, and then fit a simple model (for example a constant) in each one. They are conceptually simple yet powerful. They

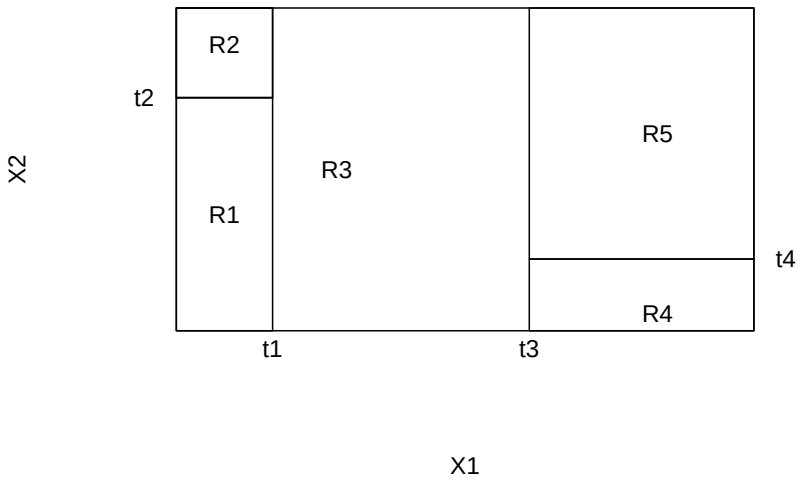
have been in use in many fields in the past, for example in Biology. (Animal or Vegetable?) In Statistics they became popular with the work of Breiman et al. in the 1980's

As an illustration consider a continuous response variable y and two continuous predictors X1 and X2, each with values in [0,1]. A partition of the feature space is given in

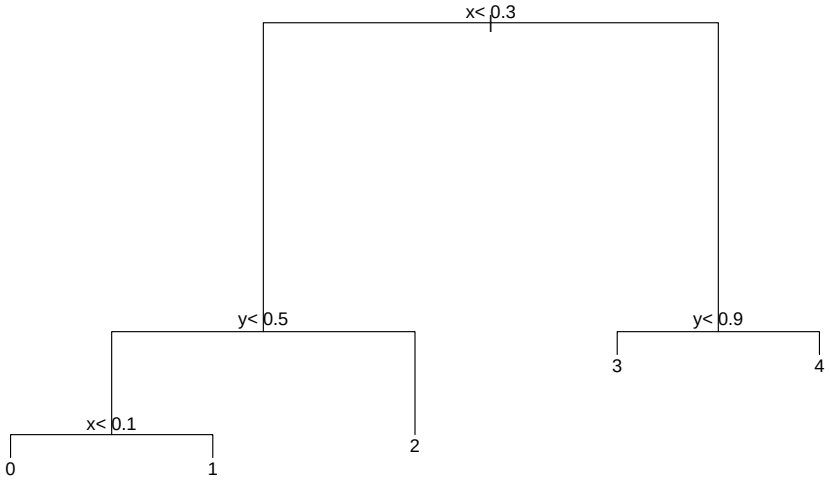


and the idea is to assign the same label to all the observations whose (x1, x2) falls into the same rectangle.

This is a perfectly legitimate partition but it has the problem that it is very difficult to describe. Instead we will restrict ourselves to the use of *recursive binary partitions* like the on shown in



Such a tree is very easy to describe using a tree diagram:



In this diagram each split is called a *node*. If it does not have a further split it is called a *terminal node*. The corresponding regression model predicts Y with a constant c_m in Region R_m, that is

$$\hat{f}(x) = \sum_{i=1}^{5} c_m I_{R_m}(x)$$

How to grow a tree

Say our data consists of p "inputs" and a response for each of n observations, that is (x_i, y_i) for i=1,..,n, with $x_i = (x_{i1}, .., x_{ip})$.

The algorithm needs to automatically decide on the splitting variables and split points, and also what topology (or shape) the tree should have. Suppose that first we partition into m regions R_1, .., R_m, and we model the response as a constant in each region.

If we adopt as our criterion minimization of the sum of squares

$$\sum (y_i - f(x_i))^2$$

it is easy to show that the best constants c_m are just the mean values of y's with corresponding x's in R_m:

$$\hat{c}_m = E[Y|x \in R_m]$$

Now finding the best binary partition in terms of minimum sum of squares is generally not possible for computational reasons, so we proceed as follows: Starting with all the data, consider a splitting variable j and split point s, and define the pair of half-planes

$$R_1(j, s) = \{x|x_j \leq s\}$$
$$R_2(j, s) = \{x|x_j > s\}$$

Then we seek the splitting variable j and split point s which are optimal.

Having found the best split we partition the data into the two resulting regions and repeat the splitting process on each of the two regions. Then this process is repeated.

How large should a tree grow? Clearly without any "stopping rule" the tree will grow until each observation has its own region, which would amount to overfitting. The size of the tree is in fact a tuning parameter such as span for loess, governing the bias-variance trade-off.

The most common strategy is to grow a large tree T_0, stopping only when some minimum node size (say 5) is reached. Then this large tree is pruned (made smaller) using *cost-complexity pruning*:

Say T_1 is a sub-tree of T_0, that is T1 can be obtained by pruning branches off T_0. Let |T| be the number of terminal nodes in T and index the terminal nodes by 1,..,m. Let

$$\hat{c}_m = \frac{1}{n_m} \sum_{x_i \in R_m} y_i$$
$$Q_m(T) = \frac{1}{n_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2$$

then we define the cost-complexity criterion

$$C_\alpha(T) = \sum_{m=1}^{|T|} n_m Q_m(T) + \alpha |T$$

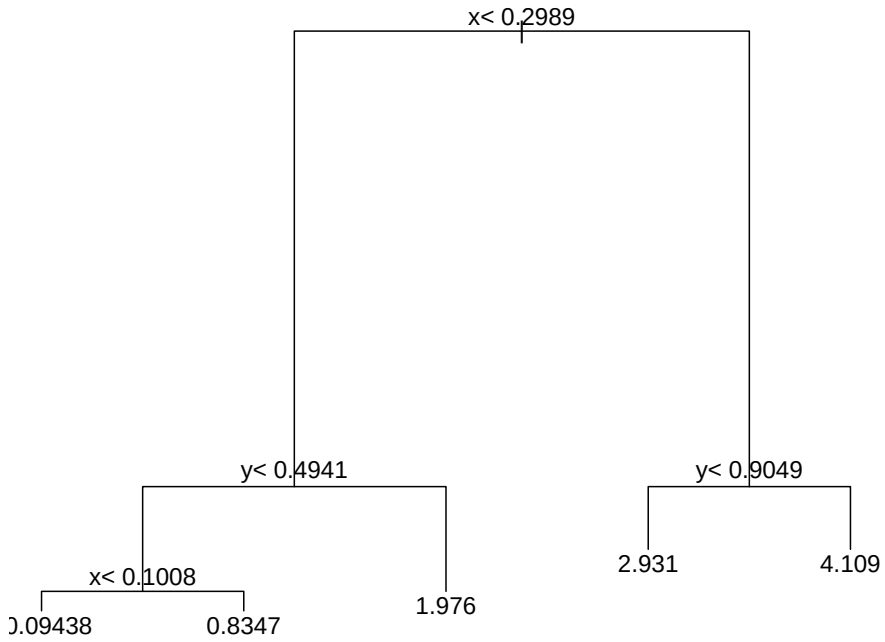
the idea is to find, for each α, the sub-tree T_α to minimize $C_\alpha(T)$. The tuning parameter α governs the trade-off between the size of the tree and the goodness-of-fit to the data. If $\alpha = 0$ the solution is the full tree T_0, and for larger values of α the tree becomes smaller. Methods to choose an "optimal" α automatically are known, for example cross-validation.

Example: Simulated data

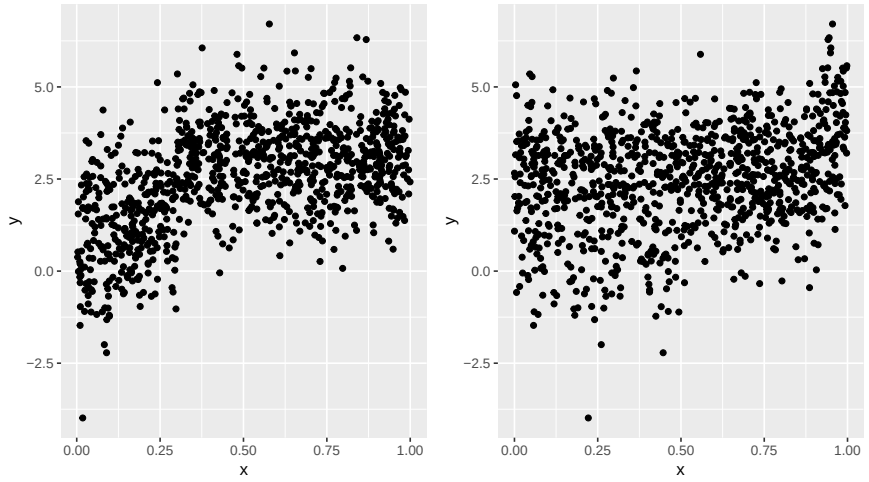
we will use the tree shown above

```
x <- runif(1000)
y <- runif(1000)
z <- rep(0, 1000)
for (i in 1:1000) {
  if (x[i] < 0.3) {
      if (y[i] < 0.5) {
          z[i]<- ifelse(x[i]<0.1, rnorm(1), rnorm(1, 1))
      }
      else {
          z[i]<- rnorm(1, 2)
      }
    }
    else {
```

```
      z[i]<- ifelse(y[i]<0.9, rnorm(1, 3), rnorm(1, 4))
    }
  }
fit <- rpart(z ~ x + y)
par(mar=c(1, 0, 0, 0))
plot(fit)
text(fit)
```



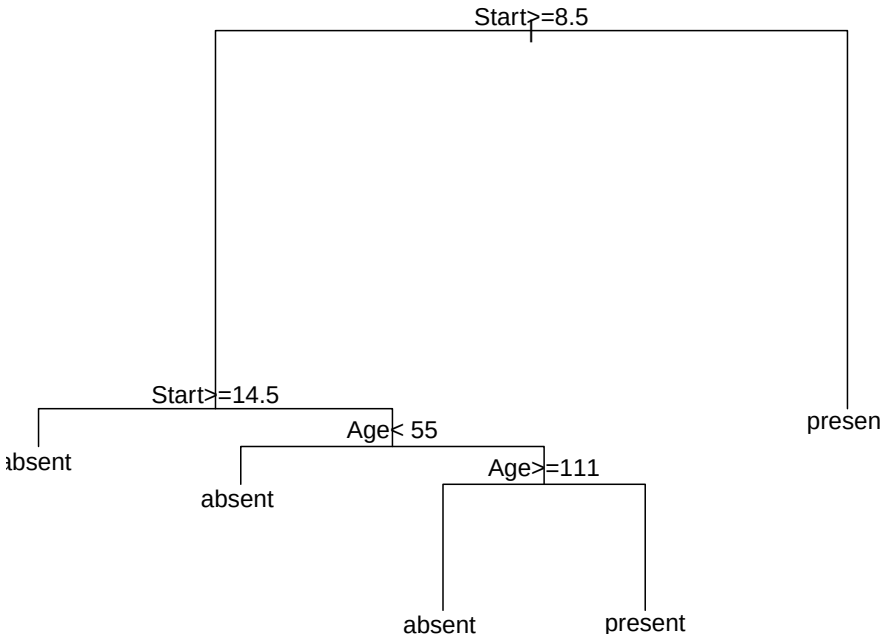

```
pushViewport(viewport(layout = grid.layout(1, 2)))
print(ggplot(data=data.frame(x=x, y=z), aes(x, y)) +
  geom_point()  ,
  vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(ggplot(data=data.frame(x=y, y=z), aes(x, y)) +
  geom_point()  ,
  vp=viewport(layout.pos.row=1, layout.pos.col=2))
```



Looking at these graphs it is not clear how one would fit a standard model to this data set.

Example: Kyphosis

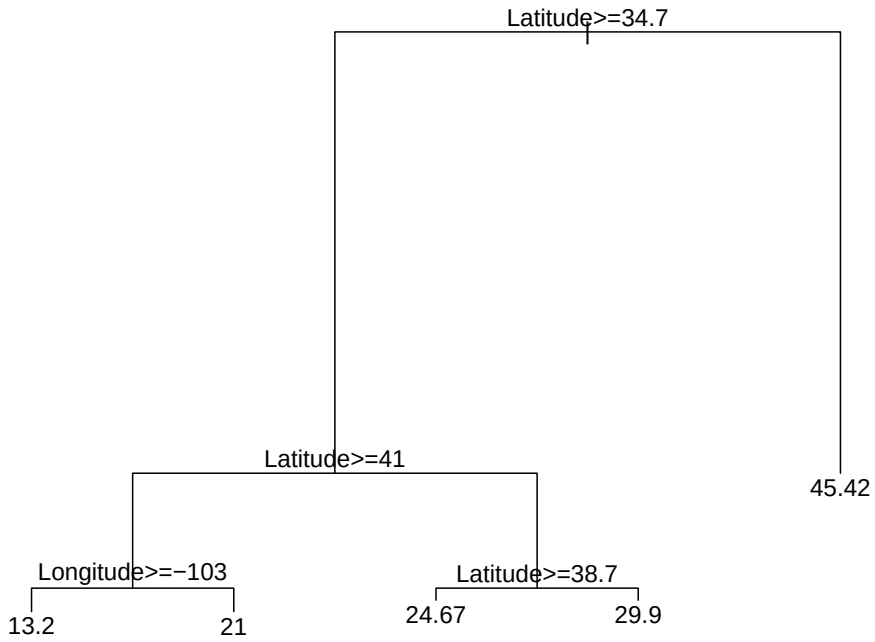
```
fit <- rpart(Kyphosis ~ Age + Number + Start,
             data = kyphosis)
par(mar=c(1, 0, 0, 0))
plot(fit)
text(fit)
```



notice that the method easily accepts a binary categorical response!

Example: US Temperature

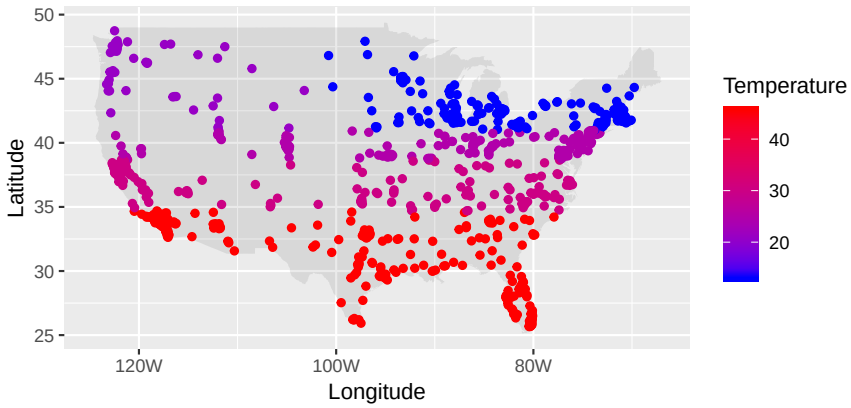
```
ustemperature$Longitude <- (-ustemperature$Longitude)
fit <- rpart(JanTemp~Longitude+Latitude,
             data = ustemperature)
par(mar=c(1, 0, 0, 0))
plot(fit)
text(fit)
```



here is the corresponding fitted line plot:

```
library(maps)
df <- us.cities[, 5:4]
df <- df[df[, 2]<50, ] #not Alaska
df <- df[df[, 2]>25, ] #not Hawaii
colnames(df) <- c("Longitude", "Latitude")
df$Temp <- predict(fit, df)
```

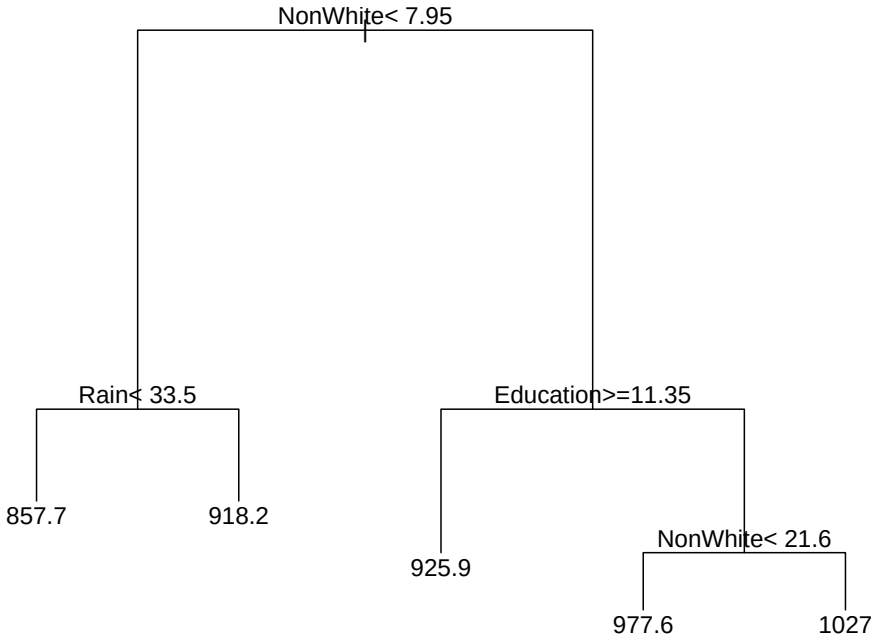
```
ggplot() +
  geom_polygon(data = usa,
               aes(x=long, y = lat, group = group),
               alpha=0.1) +
  coord_fixed(1.3) +
  geom_point(data=df, aes(Longitude, Latitude, color=Temp)) +
  scale_colour_gradient(low="blue", high="red") +
  labs(x="Longitude", y="Latitude", color="Temperature") +
  scale_x_continuous(breaks = c(-120, -100, -80),
                     labels = c("120W", "100W", "80W"))
```



Example: Air Pollution and Mortality

```
newair <- airpollution[, -16] #take out NOxPot
newair[, c(10, 13, 14)] <- log(newair[, c(10, 13, 14)])
colnames(newair)[c(10, 13, 14)] <-
  c("log.Pop", "log.HCPot", "log.NOx")
```

```
fit <- rpart(Mortality~., data = newair)
par(mar=c(1, 0, 0, 0))
plot(fit)
text(fit)
```



and we see that the only predictors used are NonWhite, Rain and Education

Principle Components Analysis

As we saw before, highly correlated predictors can cause difficulties in a regression analysis. We will now study a way to deal with this.

The idea of principle components is this: find a few linear combinations of $x_1, .., x_k$ that explain the variation in y. That is, find

$$z_1 = \sum \alpha_{i1} x_i ... z_m = \sum \alpha_{ik} x_i$$

and do a regression of y on $z_1, .., z_m$ instead of $x_1, .., x_k$.

This is a useful idea if the following happens:

- m is much smaller than k
- z_i and z_j are uncorrelated
- $z_j = \sum \alpha_{ij} x_i$ can be interpreted in some way, that is we can understand the meaning of z_j

Using matrix notation we can write $Z = XA$ where X is the data matrix and A is the m by n matrix of α's.

How should we choose A? The idea of principle components is as follows: Choose A such that

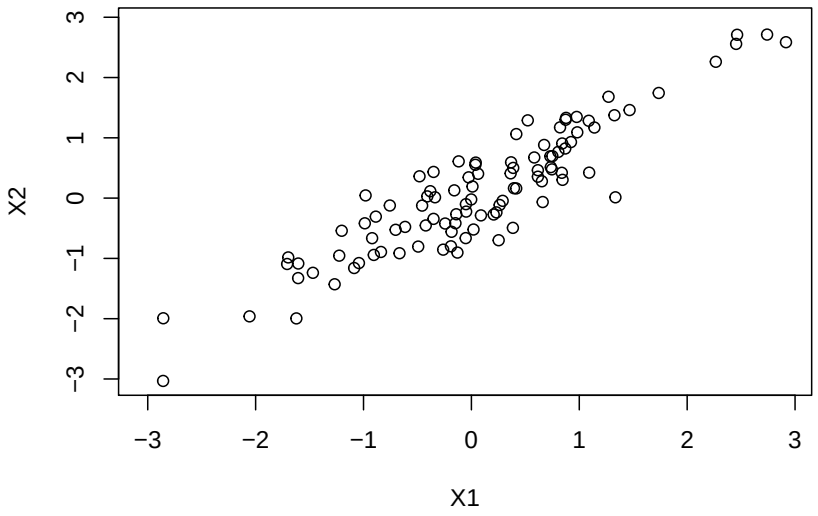
- the variables $z_1, .., z_m$ are uncorrelated (Z'Z is a diagonal matrix)
- z_1 has the largest possible variance, then z_2 has the second largest (subject to cor(z_1, z_2)=0), and so on.

So we want to find a matrix A such that Z'Z = (XA)'XA = A'(X'X)A = D

Now X'X is a symmetric k by k matrix. It could be singular but let's assume for the moment that it is not. Then using Linear Algebra it can be shown that the columns of A are the the eigenvectors of the matrix X'X.

Let's see how this works on an artificial example. We have a sample of 100 observations from a bivariate normal distribution with means (0,0), variances (1, 1) and correlation 0.9. First we plot x2 vs x1, then we find the matrix X'X and its eigenvectors (using the R function eigen). Next we find the new variables z1 and z2 as linear combinations of the eigenvectors and X.

```
library(mvtnorm)
x <- rmvnorm(100, mean = c(0, 0),
             sigma = matrix(c(1, 0.9, 0.9, 1), 2, 2))
xyr<- range(x)
plot(x, xlab = "X1", ylab = "X2", xlim = xyr, ylim = xyr)
```




```
y <- t(x) %*% x
print(y)
```

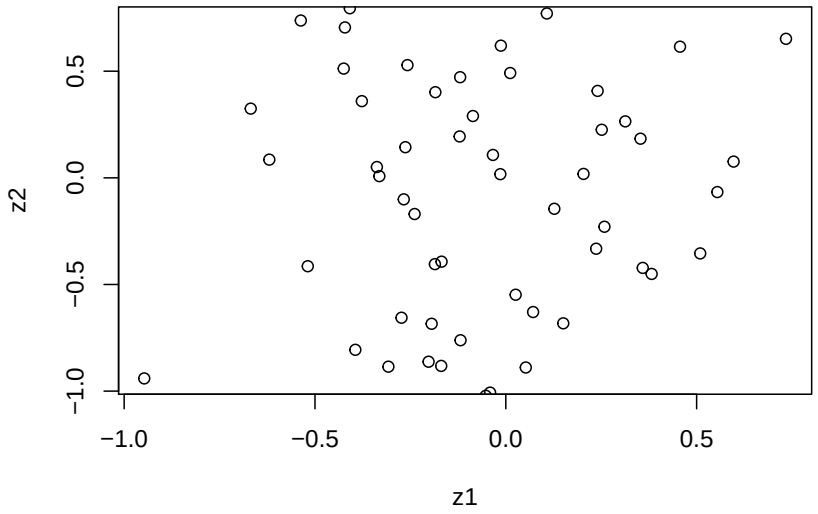
```
##          [,1]     [,2]
## [1,] 113.9399 101.5266
## [2,] 101.5266 108.3705
```

```
E <- eigen(y)
print(E)
```

```
## eigen() decomposition
## $values
## [1] 212.720001   9.590372
```

```
##
## $vectors
##             [,1]       [,2]
## [1,] -0.7167349  0.6973458
## [2,] -0.6973458 -0.7167349
```

```
z1 <- E$vector[1, 1] * x[, 1] + E$vector[1, 2] * x[, 2]
z2 <- E$vector[2, 1] * x[, 1] + E$vector[2, 2] * x[, 2]
plot(z1, z2, xlab = "z1", ylab = "z2", ylim = range(z1))
```

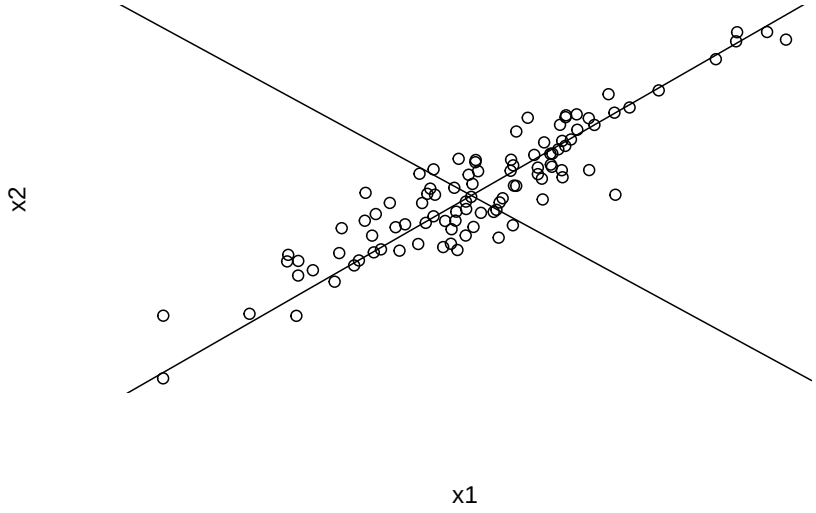


Notice

- z1 and z2 are uncorrelated (here, they are of course independent)
- the variance of z1 is much large than the variance of z2.

There is another, geometric way to see what principle components are: again we draw the scatterplot of x2 vs. x1, but without the axes. The first principle component transformation is y=e1,1x1+e1,2x2. In the x2 vs. x1 plot this describes a line with slope -e1,1/e1,2 and going through the origin, which we add to the plot. We do the same with the second principle component transformation.

```
plot(x, xlab = "x1", ylab = "x2",
     xlim = xyr, ylim = xyr,  axes = F)
abline(0, -E$vector[1, 1]/E$vector[1, 2])
abline(0, -E$vector[2, 1]/E$vector[2, 2])
```



Now we can see that the transformation is really a change of coordinate system, from x1, x2 to z1, z2.

In practice we can use the R function *princomp* to carry out the calculations for us.

```
pc <- princomp(x)
print(summary(pc))
```

```
## Importance of components:
##                            Comp.1     Comp.2
## Standard deviation     1.4529260 0.30876903
## Proportion of Variance 0.9567888 0.04321123
## Cumulative Proportion  0.9567888 1.00000000
```

we see that the first principle component explains about 95% of the variation in (x1, x2).

One of the components of the pc object is called "loadings"

```
pc$loadings
```

```
##
## Loadings:
##      Comp.1 Comp.2
## [1,]  0.718  0.696
## [2,]  0.696 -0.718
##
##                Comp.1 Comp.2
## SS loadings       1.0    1.0
## Proportion Var    0.5    0.5
## Cumulative Var    0.5    1.0
```

and we see that these are just the eigenvectors.

Example: Scores on math tests

consider the data in testscores. This is artificial data supposed to be the test scores of 25 mathematics graduate students in their qualifying exams. The differential geometry and complex analysis exams were closed book whereas the others were open book.

```
kable.nice(testscores)
```

diffgeom	complex	algebra	reals	statistics
36	58	43	36	37
62	54	50	46	52
31	42	41	40	29
76	78	69	66	81
46	56	52	56	40
12	42	38	38	28
39	46	51	54	41
30	51	54	52	32
22	32	43	28	22
9	40	47	30	24
32	49	54	37	52
40	62	51	40	49
64	75	70	66	63
36	38	58	62	62
24	46	44	55	49
50	50	54	52	51
42	42	52	38	50
2	35	32	22	16
56	53	42	40	32
59	72	70	66	62
28	50	50	42	63
19	46	49	40	30
36	56	56	54	52
54	57	59	62	58
14	35	38	29	20

```
test.pc <- princomp(testscores)
summary(test.pc, loadings = TRUE)
```

```
## Importance of components:
##                            Comp.1     Comp.2     Comp.3     Comp.4
## Standard deviation     28.4896795 9.03547104 6.60095491 6.13358179
```

```
## Proportion of Variance  0.8212222 0.08260135 0.04408584 0.03806395
## Cumulative Proportion   0.8212222 0.90382353 0.94790936 0.98597332
##                            Comp.5
## Standard deviation     3.72335754
## Proportion of Variance 0.01402668
## Cumulative Proportion  1.00000000
##
## Loadings:
##            Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
## diffgeom    0.598  0.675  0.185  0.386
## complex     0.361  0.245 -0.249 -0.829 -0.247
## algebra     0.302 -0.214 -0.211 -0.135  0.894
## reals       0.389 -0.338 -0.700  0.375 -0.321
## statistics  0.519 -0.570  0.607        -0.179
```

Looking at the summary we see that the first pc accounts for 82% of the variation in the data, and the first two account for 90%.

Let's have a look at the loadings: the first one is (0.6, 0.36, 0.3, 0.39, 0.52) and amounts to an average over all the exams. The second one is (-0.67, -0.25, 0.21, 0.34, 0.57). Notice that here the first two are negative and the others are positive. But the first two were the closed book exams and the others were open book!

Example: States

The data set *state.x77* has info of the 50 states of the United States of America.

'Population': population estimate as of July 1, 1975
'Income': per capita income (1974)
'Illiteracy': illiteracy (1970, percent of population)
'Life Exp': life expectancy in years (1969-71)
'Murder': murder and non-negligent manslaughter rate per 100,000 population (1976)
'HS Grad': percent high-school graduates (1970)
'Frost': mean number of days with minimum temperature below freezing (1931-1960) in capital or large city
'Area': land area in square miles

Source: U.S. Department of Commerce, Bureau of the Census (1977) Statistical Abstract of the United States.

```
kable.nice(head(state.x77))
```

	Population	Income	Illiteracy	Life Exp	Murder	HS Grad	Frost	Area
Alabama	3615	3624	2.1	69.05	15.1	41.3	20	50708
Alaska	365	6315	1.5	69.31	11.3	66.7	152	566432
Arizona	2212	4530	1.8	70.55	7.8	58.1	15	113417
Arkansas	2110	3378	1.9	70.66	10.1	39.9	65	51945
California	21198	5114	1.1	71.71	10.3	62.6	20	156361
Colorado	2541	4884	0.7	72.06	6.8	63.9	166	103766

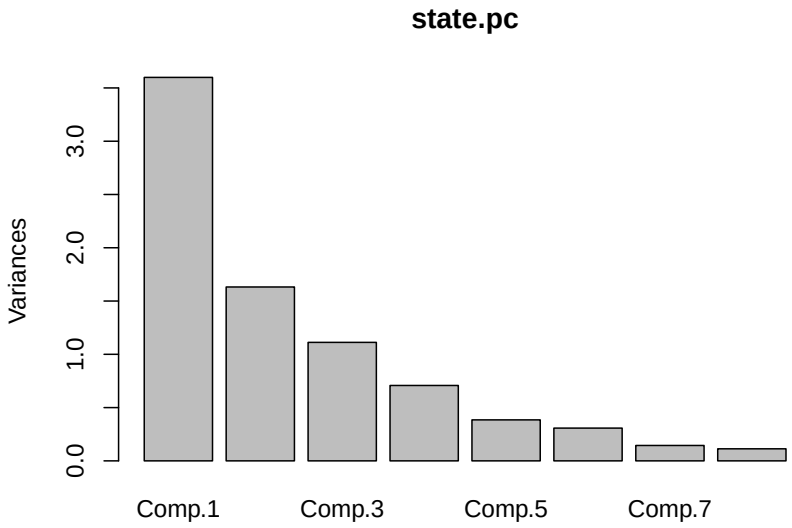
```
state.pc <- princomp(state.x77, cor = T)
summary(state.pc, loading = TRUE)
```

```
## Importance of components:
##                           Comp.1    Comp.2    Comp.3     Comp.4     Comp.5
## Standard deviation     1.8970755 1.2774659 1.0544862 0.84113269 0.62019488
## Proportion of Variance 0.4498619 0.2039899 0.1389926 0.08843803 0.04808021
## Cumulative Proportion  0.4498619 0.6538519 0.7928445 0.88128252 0.92936273
##                            Comp.6    Comp.7     Comp.8
## Standard deviation     0.55449226 0.3800642 0.33643379
## Proportion of Variance 0.03843271 0.0180561 0.01414846
## Cumulative Proportion  0.96779544 0.9858515 1.00000000
##
## Loadings:
##            Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
## Population  0.126  0.411  0.656  0.409  0.406                 0.219
## Income     -0.299  0.519  0.100        -0.638 -0.462
## Illiteracy  0.468                -0.353        -0.387  0.620  0.339
## Life Exp   -0.412         0.360 -0.443  0.327 -0.219  0.256 -0.527
## Murder      0.444  0.307 -0.108  0.166 -0.128  0.325  0.295 -0.678
## HS Grad    -0.425  0.299        -0.232         0.645  0.393  0.307
## Frost      -0.357 -0.154 -0.387  0.619  0.217 -0.213  0.472
## Area               0.588 -0.510 -0.201  0.499 -0.148 -0.286
```

It appears that the first PC contrasts "good" variables such as income and life expectancy with bad ones such as murder and illiteracy. This explains about 45% of the variation. The second PC contrasts 'Frost' with all the other variables. It accounts for an additional 20% but it seems difficult to understand exactly what that means.

One important question is how many PCs are needed to "reasonably" explain the data? One useful graph here is the screeplot, given in

```
plot(state.pc)
```



It is a simple barchart of the the variation explained by each PC. One popular method is to include enough PCs to cover at least 90% of the variation. In the states data this means 5, which seems a bit much.

Classification

Introduction to Classification

In general classification is concerned with the following problem: our population consists of several distinct groups. We have a set of data points from each group with associated measurements. We want to derive a procedure that tells us which group a new observation might belong to.

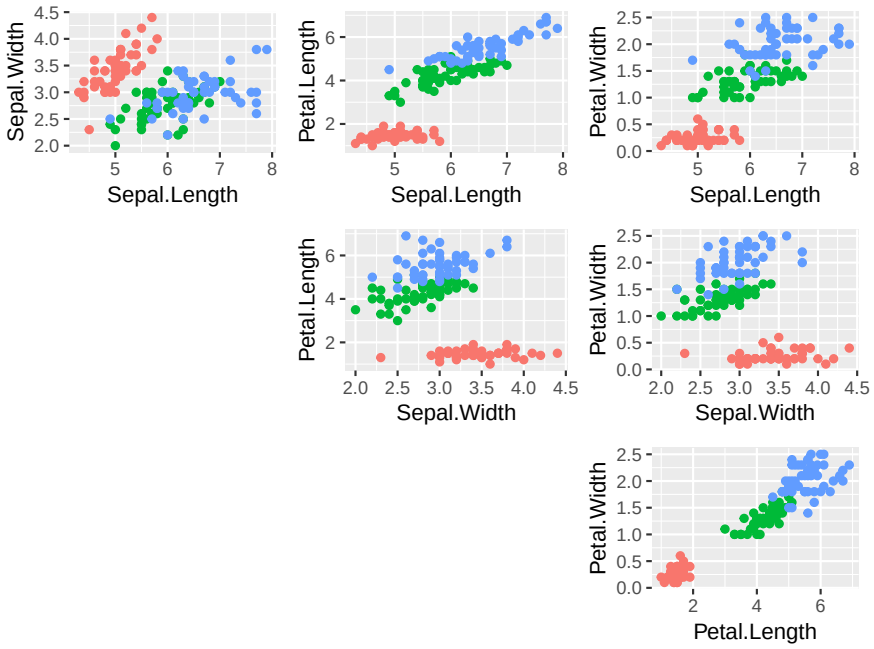
Example: Fisher's Iris Data

One of the most famous data sets in Statistics was first studied by Fisher, his iris data. For each of three types of iris flowers (Iris setosa, Iris virginica and Iris versicolor) we have four measurements: the lengths and the widths of the Petal and the Sepal. The goal is to determine from these measurements the type of flower.

```
kable.nice(head(iris))
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

```
pushViewport(viewport(layout = grid.layout(3, 3)))
print(ggplot(data=iris,
          aes(Sepal.Length, Sepal.Width, color=Species)) +
      geom_point() + theme(legend.position="none"),
  vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(ggplot(data=iris,
         aes(Sepal.Length, Petal.Length, color=Species)) +
      geom_point() + theme(legend.position="none"),
  vp=viewport(layout.pos.row=1, layout.pos.col=2))
print(ggplot(data=iris,
             aes(Sepal.Length, Petal.Width, color=Species)) +
      geom_point() + theme(legend.position="none"),
  vp=viewport(layout.pos.row=1, layout.pos.col=3))
print(ggplot(data=iris,
             aes(Sepal.Width, Petal.Length, color=Species)) +
      geom_point() + theme(legend.position="none"),
  vp=viewport(layout.pos.row=2, layout.pos.col=2))
print(ggplot(data=iris,
             aes(Sepal.Width, Petal.Width, color=Species)) +
      geom_point() + theme(legend.position="none"),
  vp=viewport(layout.pos.row=2, layout.pos.col=3))
print(ggplot(data=iris,
             aes(Petal.Length, Petal.Width, color=Species)) +
      geom_point() + theme(legend.position="none"),
  vp=viewport(layout.pos.row=3, layout.pos.col=3))
```



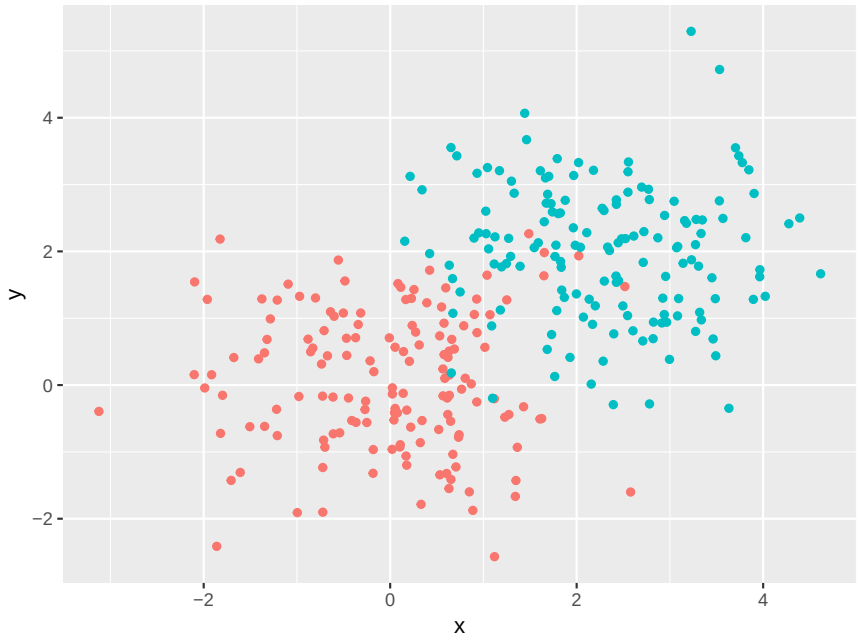
Example:

Consider the following artificial examples:

- two types, fairly simple separation

```
library(mvtnorm)
ex1 <- function(mu=2, n=50) {
  x1 <- rmvnorm(n, mean=c(0,0), sigma=diag(2))
  x2 <- rmvnorm(n, mean=c(mu,mu), sigma=diag(2))
  data.frame(x=c(x1[, 1], x2[, 1]),
             y=c(x1[, 2], x2[, 2]),
             group=rep(c("A", "B"), each=n))
}
```

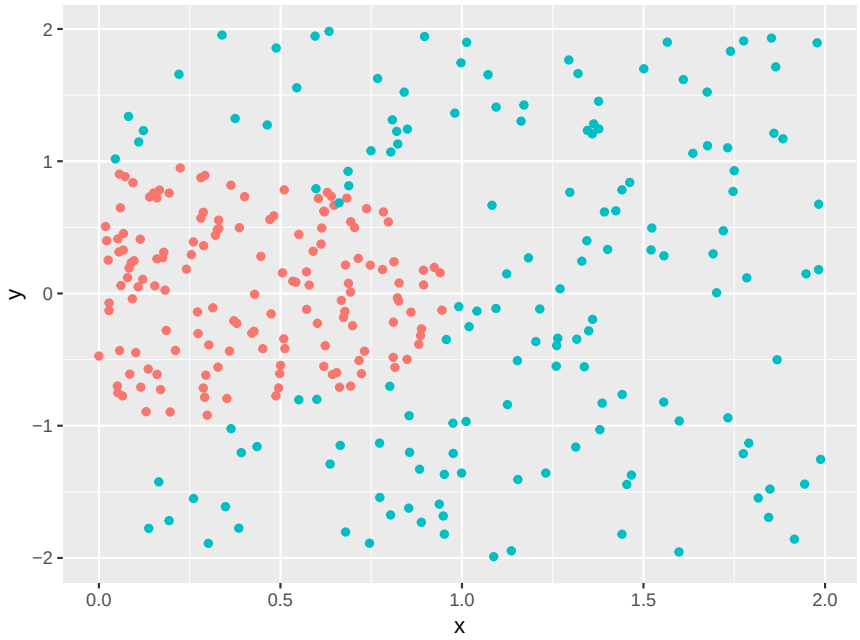
```
ggplot(data=ex1(n=150), aes(x, y,color=group)) +
  geom_point() +
  theme(legend.position="none")
```



- two types, more complicated separation

```
ex2 <- function(mu=2, n=50) {
  x <- cbind(runif(10000), runif(10000, -1, 1))
  x <- x[x[, 1]^2 + x[, 2]^2<1, ]
  x <- x[1:n, ]
  y <- cbind(runif(10000, 0, 2), runif(10000, -2, 2))
  y <- y[y[, 1]^2 + y[, 2]^2>0.9, ]
  y <- y[1:n, ]
  data.frame(x=c(x[, 1], y[, 1]),
             y=c(x[, 2], y[, 2]),
             group=rep(c("A", "B"), each=n))
}
```

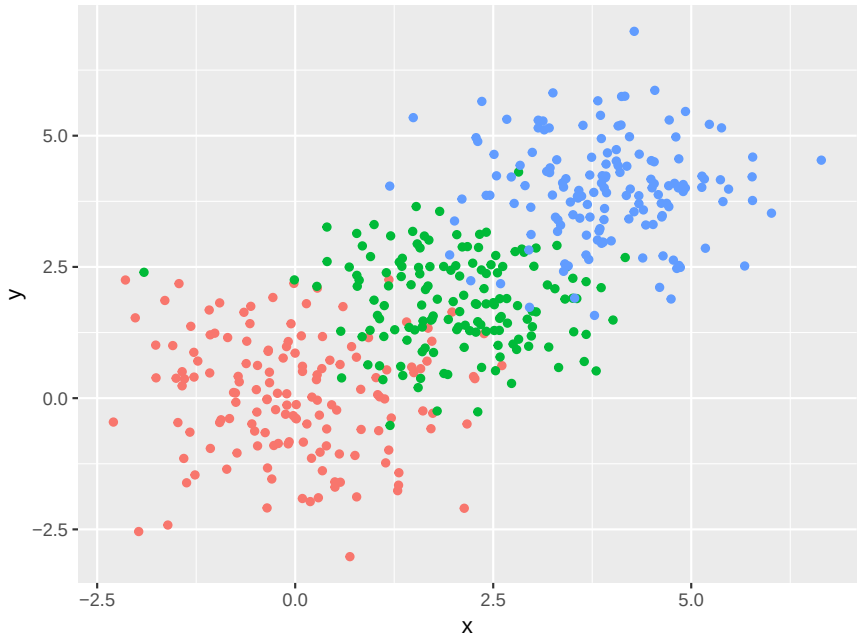
```
ggplot(data=ex2(n=150), aes(x, y,color=group)) +
  geom_point() +
  theme(legend.position="none")
```



- three types

```
ex3 <- function(mu=2, n=33) {
  x1 <- rmvnorm(n, mean=c(0, 0), sigma=diag(2))
  x2 <- rmvnorm(n, mean=c(mu, mu), sigma=diag(2))
  x3 <- rmvnorm(n, mean=2*c(mu, mu), sigma=diag(2))
  data.frame(x=c(x1[, 1], x2[, 1], x3[, 1]),
             y=c(x1[, 2], x2[, 2], x3[, 2]),
             group=rep(c("A", "B", "C"), each=n))
}
```

```
ggplot(data=ex3(n=150), aes(x, y,color=group)) +
  geom_point() +
  theme(legend.position="none")
```



In one sense this is not a new problem, it is simply a regression problem where the response variable is discrete.

For this we could code a response variable y as 0 if "green" and 1 if "red" if there are two groups (models 1 and 2) or with 0, 1 and 2 if there are three groups (model 3). Then we run the linear regression of y on x1 and x2.

Finally we assign a point (x1, x2) to "green" if its predicted response is <0.5, and to "red" otherwise for models 1 and 2, and depending on whether its predicted response is $<2/3$ or $>4/3$ for model 3.

Of course in the case of two groups we could also use *logistic regression*, but we won't pursue this idea here.

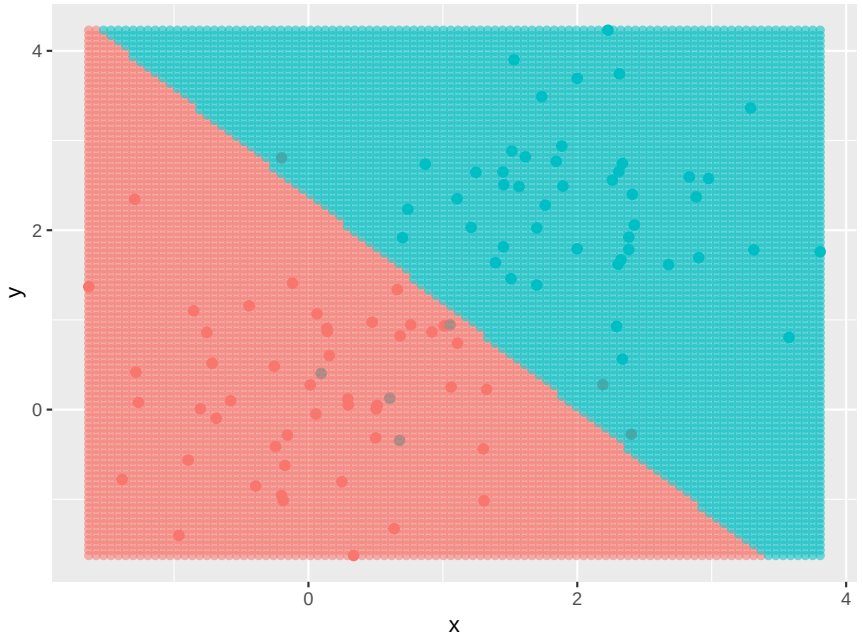
To see what this looks like we find an even spaced grid and predict the color for each point. Then we overlay that grid onto the graph. This is done in

```
make.grid <- function(df) {
  x <- seq(min(df$x), max(df$x), length=100)
  y <- seq(min(df$y), max(df$y), length=100)
  expand.grid(x=x, y=y)
}
do.graph <- function(df, df1) {
  print(ggplot(data=df, aes(x, y, color=group)) +
   geom_point(size=2) +
   theme(legend.position="none") +
   geom_point(data=df1,
              aes(x,y, color=group, alpha=0.1),
              inherits.aes=FALSE))
```

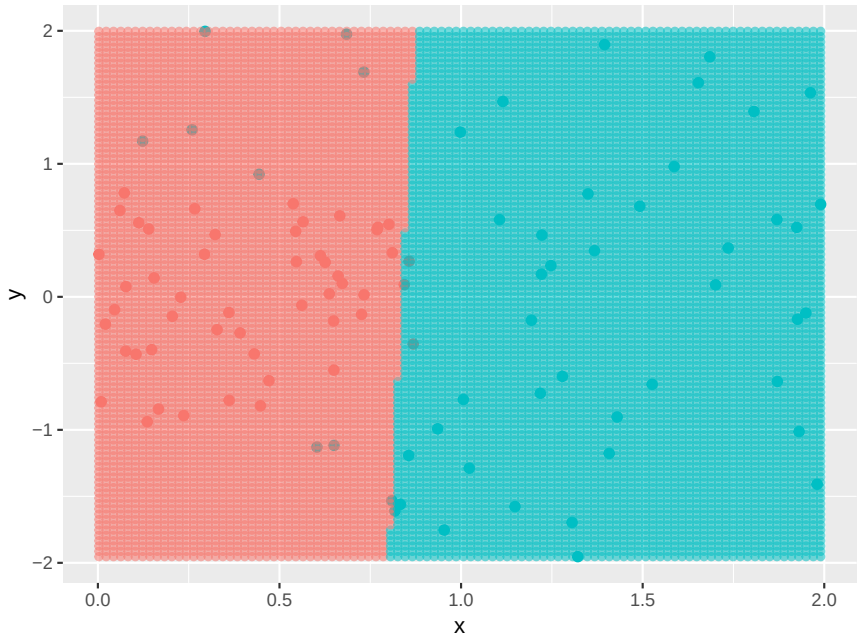
```
}
```

Here our three examples:

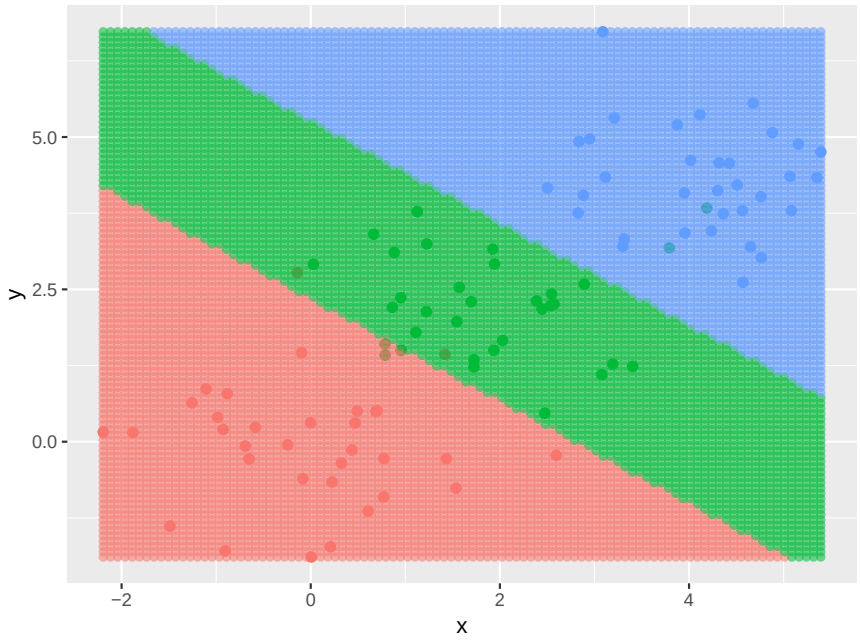
```
df <- ex1()
df$Code <- ifelse(df$group=="A", 0, 1)
fit <- lm(Code~x+y, data=df)
df1 <- make.grid(df)
df1$group <- ifelse(predict(fit, df1)<0.5, "A", "B")
do.graph(df, df1)
```




```
df <- ex2()
df$Code <- ifelse(df$group=="A", 0, 1)
fit <- lm(Code~x+y, data=df)
df1 <- make.grid(df)
df1$group <- ifelse(predict(fit, df1)<0.5, "A", "B")
do.graph(df, df1)
```




```
df <- ex3()
df$Code <- ifelse(df$group=="A", 0, 1)
df$Code[df$group=="C"] <- 2
fit <- lm(Code~x+y, data=df)
df1 <- make.grid(df)
tmp <- predict(fit, df1)
df1$group <- ifelse(tmp<2/3, "A", "B")
df1$group[tmp>4/3] <-"C"
do.graph(df, df1)
```



this seems to work ok for examples 1 and 3, not so much for 2.

we will use these examples quite a bit, so lets write a routine that generates data from any of them:

```
gen.ex <- function(which, n=50) {
  if(which==1)
      df <- ex1(n=n)
  if(which==2)
      df <- ex2(n=n)
  if(which==3)
      df <- ex3(n=n)
  df$Code <- ifelse(df$group=="A", 0, 1)
  if(which==3)
    df$Code[df$group=="C"] <- 2
  df
}
```

Let's have a closer look at example 1:

```
df <- gen.ex(1)
fit <- lm(Code~x+y, data=df)
coef(fit)
```

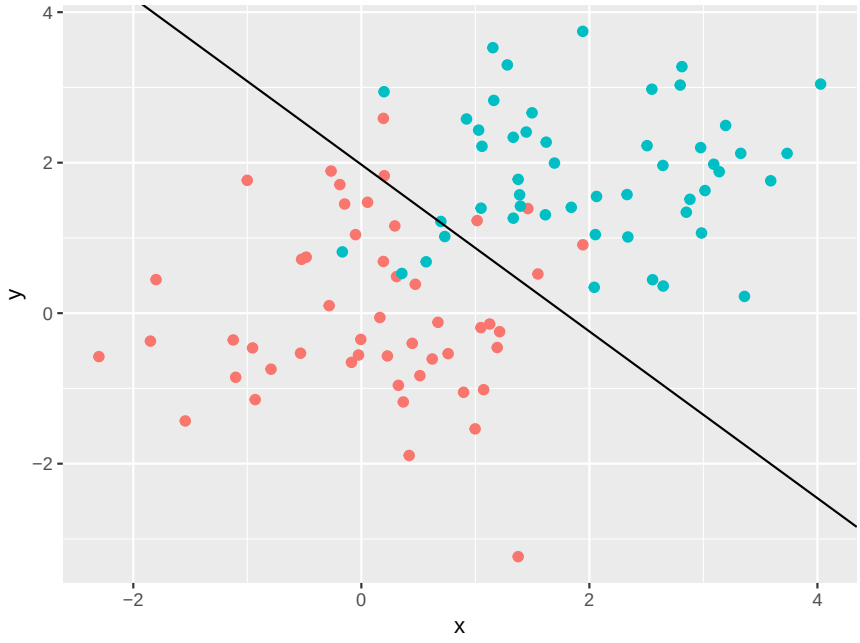
```
## (Intercept)           x           y
##   0.1612369   0.1797628   0.1622197
```

we assign the group depending if the fitted value is $<$ or $>$ than 0.5. What do we get if it is equal to 0.5?

$$
\begin{aligned}
0.5 &= \beta_0 + \beta_1 x + \beta_2 y \\
y &= (0.5 - \beta_0 - \beta_1 x)/\beta_2 \\
y &= \frac{0.5 - \beta_1}{\beta_2} - \frac{\beta_1}{\beta_2} x
\end{aligned}
$$

Let's add that line to the graph:

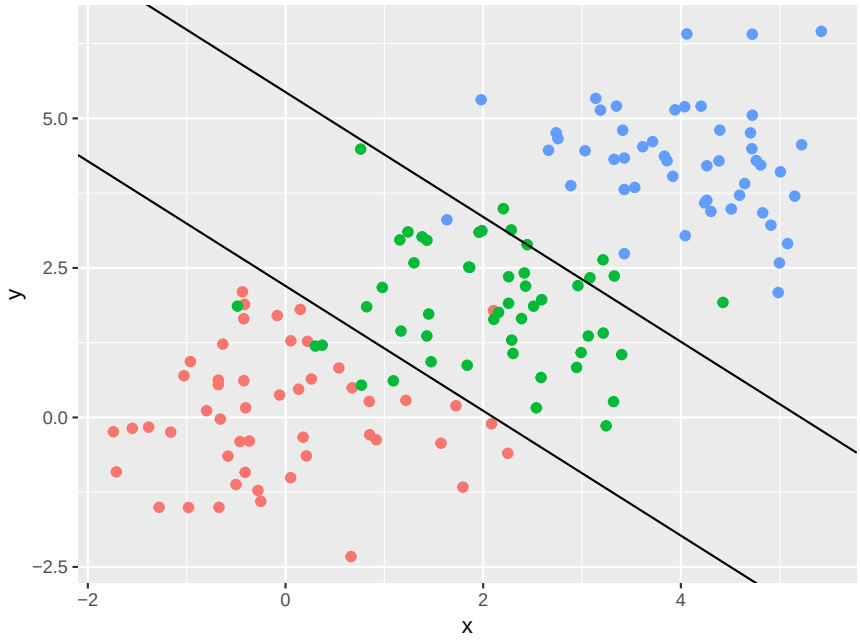
```
ggplot(data=df, aes(x, y, color=group)) +
  geom_point(size=2) +
  theme(legend.position="none") +
  geom_abline(intercept = (0.5-coef(fit)[2])/coef(fit)[3],
              slope=-coef(fit)[2]/coef(fit)[3])
```



and this is called the *decision boundary*.

It is easy to see that in example 3 it works like this:

```
df <- gen.ex(3)
fit <- lm(Code~x+y, data=df)
ggplot(data=df, aes(x, y, color=group)) +
  geom_point(size=2) +
  theme(legend.position="none") +
  geom_abline(intercept = (2/3-coef(fit)[2])/coef(fit)[3],
              slope=-coef(fit)[2]/coef(fit)[3]) +
  geom_abline(intercept = (4/3-coef(fit)[2])/coef(fit)[3],
              slope=-coef(fit)[2]/coef(fit)[3])
```



Misclassification Rate

One thing that sets a classification problem apart from regression is that here we have an obvious way to judge how good a method is, namely the **miss-classification rate**: What percentage of the observations are given the wrong label?

Let's see:

```
msr <- function(x, y) {
  z <- table(x, y)
  round((sum(z)-sum(diag(z)))/sum(z)*100, 1)
}
```

```
df <- gen.ex(1, n=1000)
fit <- lm(Code~x+y, data=df)
pred <- ifelse(predict(fit)<0.5, "A", "B")
table(df$group, pred)
```

```
##    pred
##       A   B
##   A 912  88
##   B  83 917
```

```
msr(df$group, pred)
```

```
## [1] 8.6
```

```
df <- gen.ex(2, n=1000)
fit <- lm(Code~x+y, data=df)
```

```
pred <- ifelse(predict(fit)<0.5, "A", "B")
msr(df$group, pred)
```

```
## [1] 20
```

```
df <- gen.ex(3, n=1000)
fit <- lm(Code~x+y, data=df)
tmp <- predict(fit)
pred <- ifelse(tmp<2/3, "A", "B")
pred[tmp>4/3] <- "C"
msr(df$group, pred)
```

```
## [1] 11.8
```

Overfitting and Cross-validation

Of course these misclassification rates are to optimistic: we calculated it on the same data set that we fit on. We should always *train* and *test* on different data sets, maybe using cross-validation:

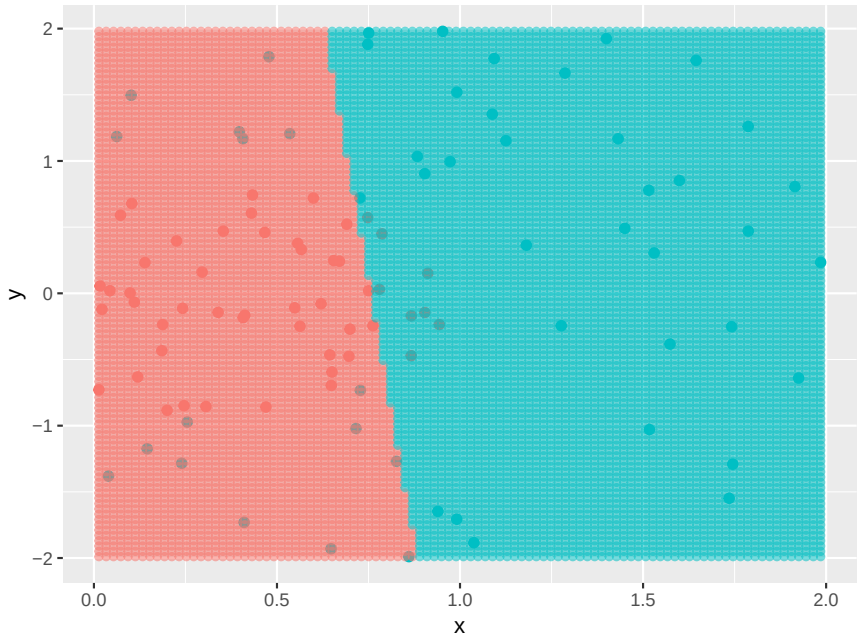
```
df <- gen.ex(1, n=1000)
out <- rep(0, 10)
for(i in 1:10) {
  I <- sample(1:2000, size=400)
  fit <- lm(Code~x+y, data=df[-I, ])
  pred <- ifelse(predict(fit, df[I, 1:2])<0.5, "A", "B")
  out[i] <- msr(df$group[I], pred)
}
mean(out)
```

```
## [1] 8.69
```

Here we split the data into 80% for training and 20% for evaluation. Is this a good split? Actually, nobody knows!

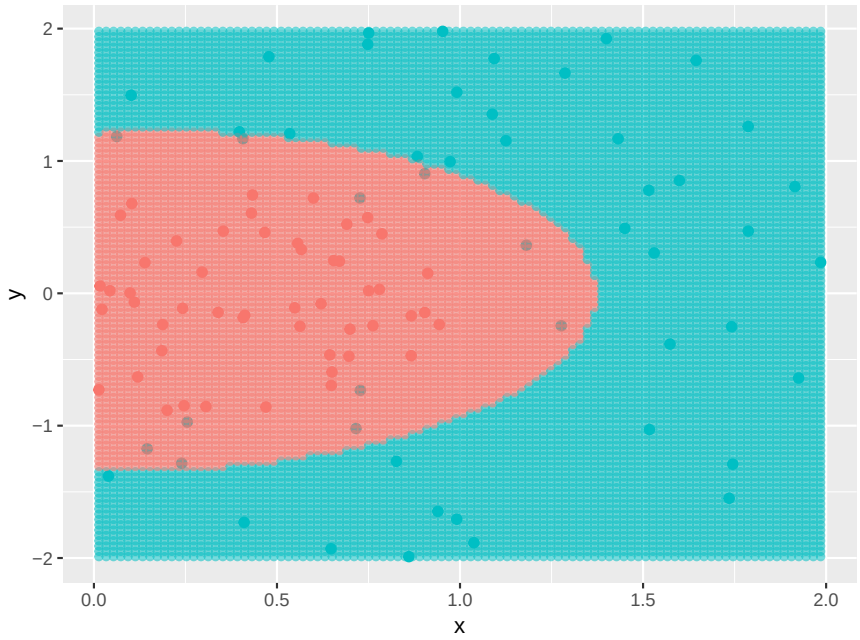
Our method works quite well for examples 1 and 3, but not so much for example 2.

```
df <- gen.ex(2)
df$Code <- ifelse(df$group=="A", 0, 1)
fit <- lm(Code~x+y, data=df)
df1 <- make.grid(df)
df1$group <- ifelse(predict(fit, df1)<0.5, "A", "B")
do.graph(df, df1)
```



shows us why: here a linear decision boundary clearly won't work. So how about a quadratic one?

```
df$x2 <- df$x^2
df$y2 <- df$y^2
df$xy <- df$x*df$y
fit <- lm(Code~x+y+x2+y2+xy, data=df)
df1 <- make.grid(df)
df1$x2 <- df1$x^2
df1$y2 <- df1$y^2
df1$xy <- df1$x*df1$y
df1$group <- ifelse(predict(fit, df1)<0.5, "A", "B")
do.graph(df, df1)
```



and that looks much better!

Here is the mcr based on cross-validation:

```
df <- df[, c(4, 1:2, 5:7)]
out <- rep(0, 10)
for(i in 1:10) {
  I <- sample(1:2000, size=400)
  fit <- lm(Code~x+y+x2+y2+xy, data=df[-I, ])
  pred <- ifelse(predict(fit, df[I, -1])<0.5, "A", "B")
  out[i] <- msr(df$Code[I], pred)
}
mean(out)
```

```
## [1] 16.4
```

Classification Methods

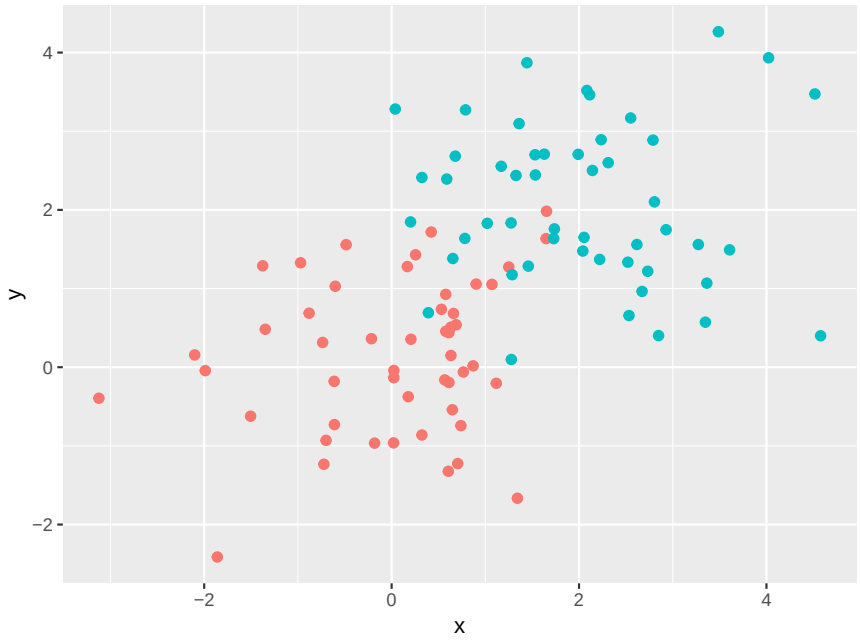
The two solutions we have discussed in the last section, linear and quadratic regression, are (slight variations of) what Fisher came up with back when he introduced the Iris data set. They are now called

Linear and Quadratic discriminants

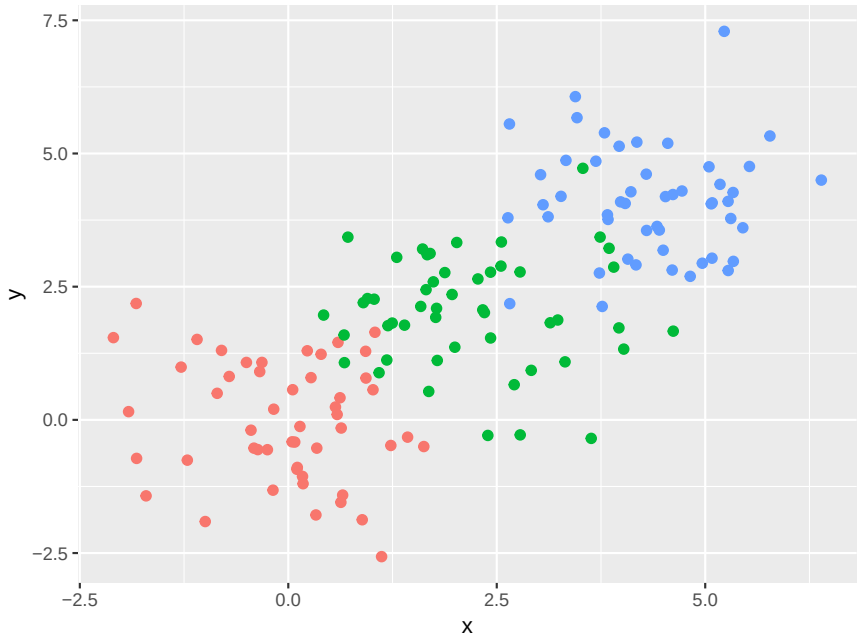
and are implemented in R with

```
library(MASS)
df <- gen.ex(1)
fit <- lda(df$group~x+y, data=df)
```

```
df1 <- make.grid(df)
df1$group <- predict(fit, df1)$class
do.graph(df, df1)
```




```
df <- gen.ex(3)
fit <- lda(group~x+y, data=df)
df1 <- make.grid(df)
df1$group <- predict(fit, df1)$class
do.graph(df, df1)
```



for example 2 we should use

```
df <- gen.ex(2)
fit <- qda(group~x+y, data=df)
df1 <- make.grid(df)
df1$group <- predict(fit, df1)$class
do.graph(df, df1)
```

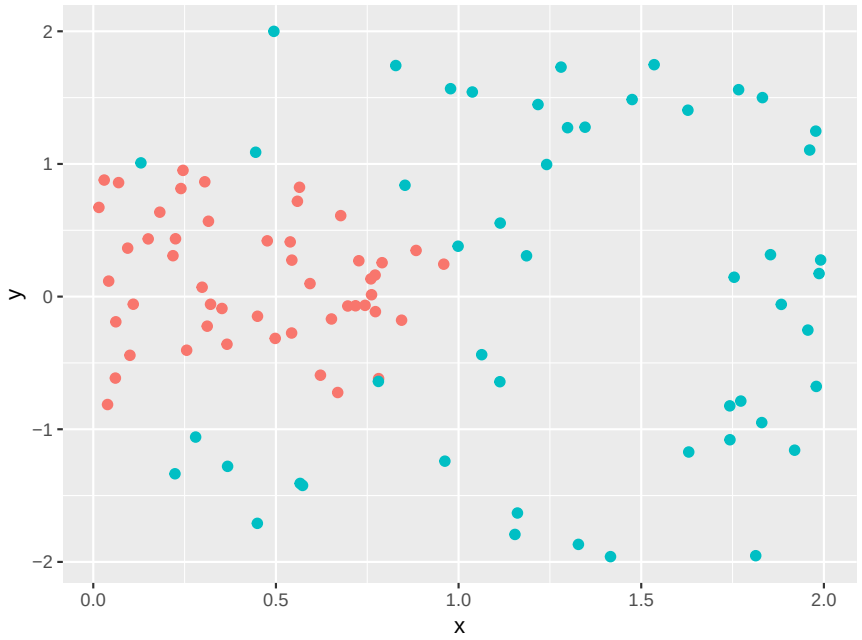
Notice a couple of differences between the lm and lda/qda solutions:

- in lda/qda we don't have to do any coding, they accept categorical variables as response.
- there is a difference between the lm and the lda/qda solutions of examples 2 and 3. Do you see what it is, and why?

Loess

Instead of using lm we could of course also have used *loess*:

```
df <- gen.ex(2)
fit <- loess(Code~x+y, data=df,
             control = loess.control(surface = "direct"))
df1$group <- c(ifelse(predict(fit, df1)<0.5, "A", "B"))
do.graph(df, df1)
```

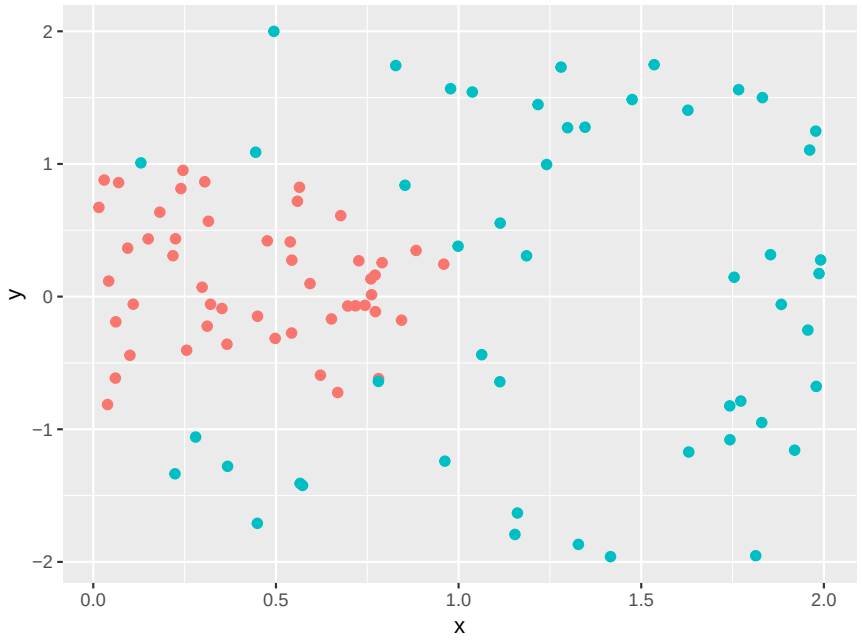


and in fact that looks quite a bit like the qda solution.

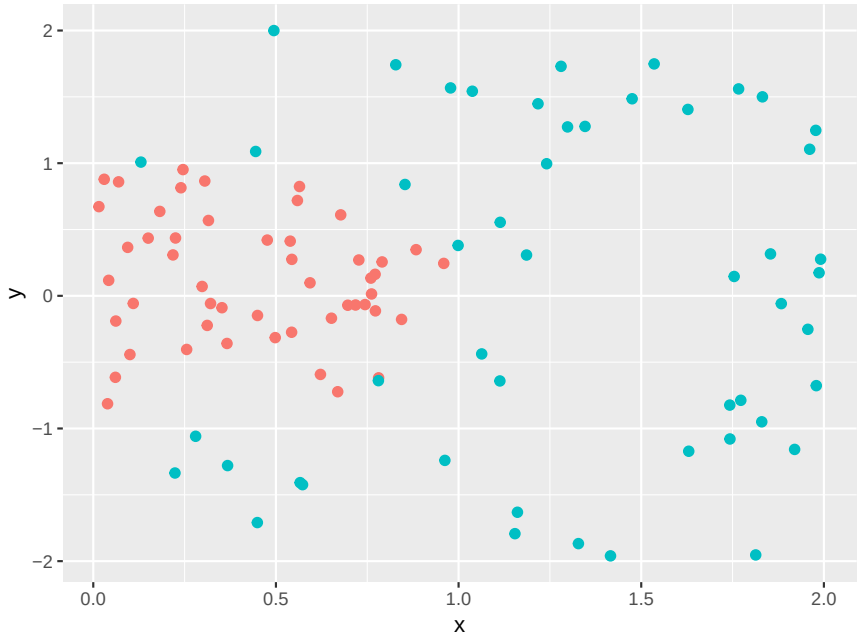
k-nearest neighbor

Here is an entirely different idea: we define a distance function, that is a function that calculates the distance between two points. In our examples we can just find Euclidean distance, but in other cases other distance functions can be useful. For a point x where we want to do prediction we find its k *nearest neighbors* and assign the label by majority rule. So if k=3 and if at least 2 of the three nearest neighbors are type "A", then we assign type "A" to x.

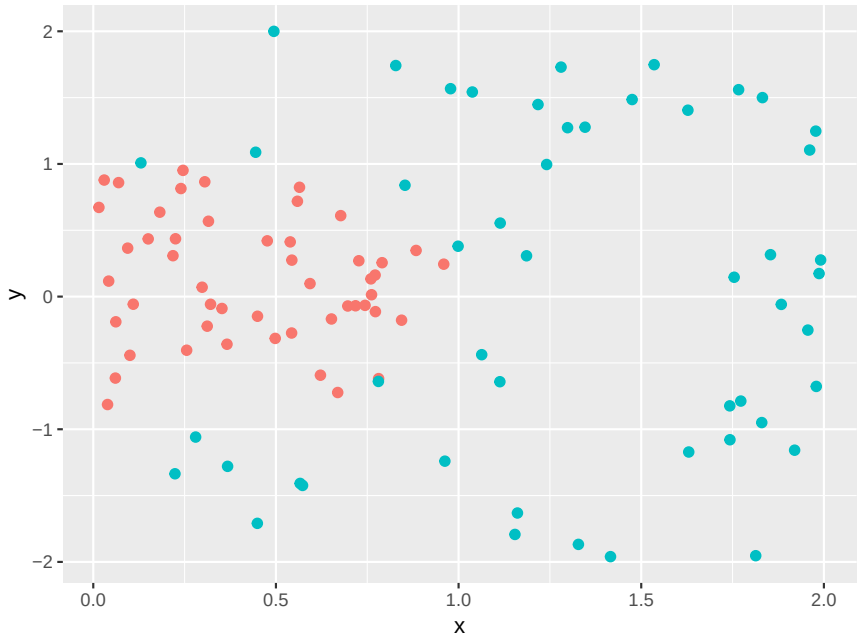
```
library(class)
df1$group <-  factor(
     knn(df[, 1:2], df1[, 1:2], cl=df$group, k=1))
do.graph(df, df1)
```




```
df1$group <-  factor(
     knn(df[, 1:2], df1[, 1:2], cl=df$group, k=3))
do.graph(df, df1)
```




```
df1$group <-  factor(
     knn(df[, 1:2], df1[, 1:2], cl=df$group, k=11))
do.graph(df, df1)
```

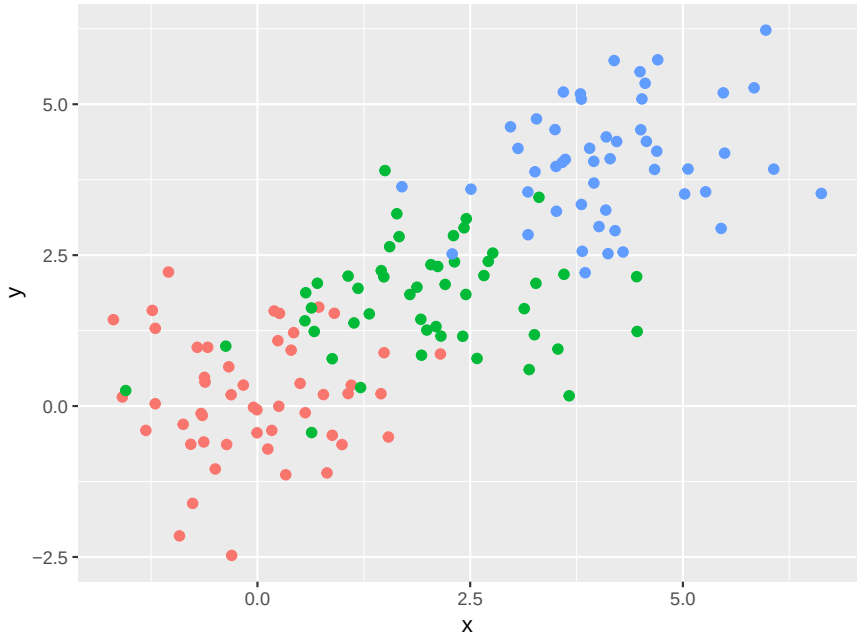


clearly the choice of k determines the bias variance trade-off.

Here is the knn solution for the other two cases:

```
df <- gen.ex(1)
df1 <- make.grid(df)
df1$group <-  factor(
     knn(df[, 1:2], df1, cl=factor(df$group), k=5))
do.graph(df, df1)
```

```
df <- gen.ex(3)
df1 <- make.grid(df)
df1$group <-  factor(
     knn(df[, 1:2], df1, cl=factor(df$group), k=5))
do.graph(df, df1)
```



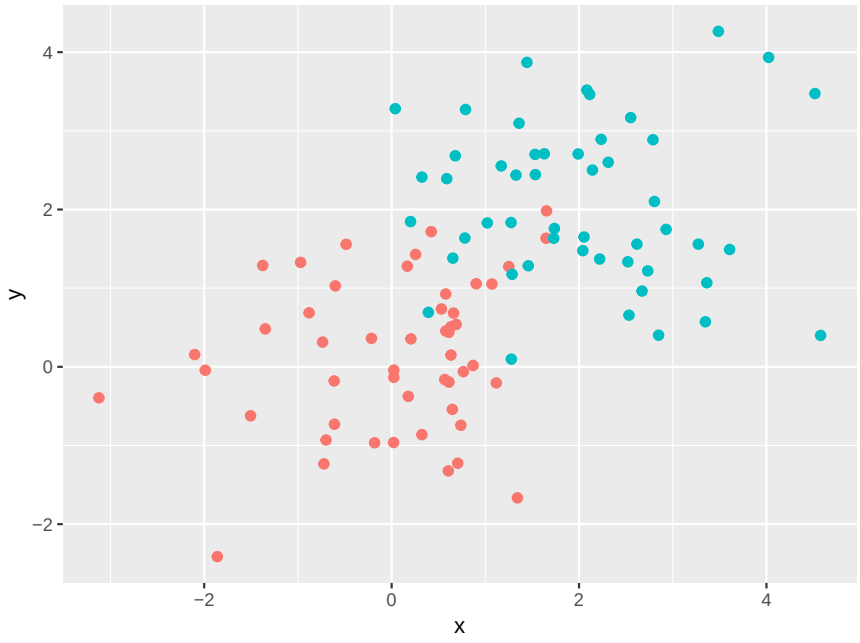
Our ability to apply this method clearly depends on how fast we can find the nearest neighbors. This issue has been studied extensively in Statistics and in Computer Science, and highly sophisticated algorithms are known that can handle millions of cases and hundreds of variables.

Classification Trees

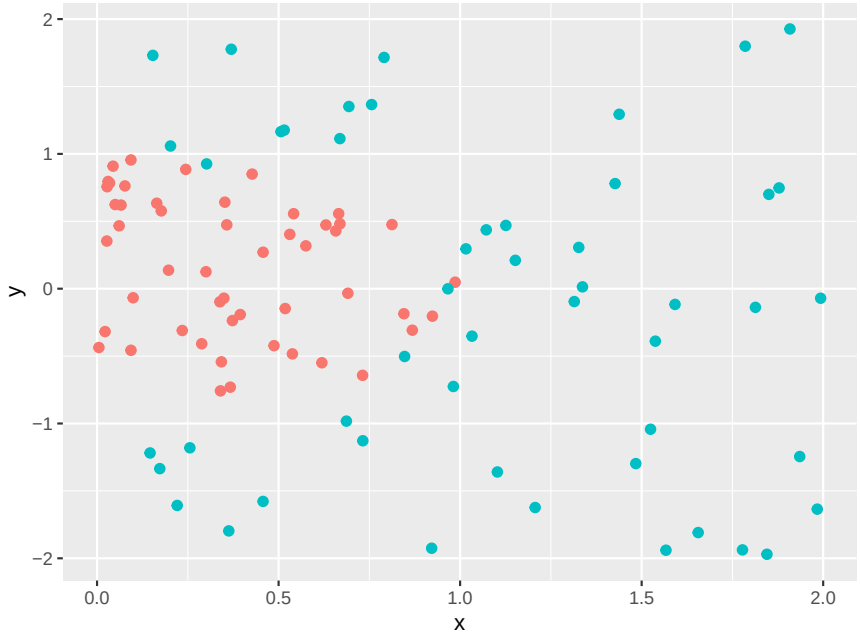
regression trees work very well for classification problems as well:

Basic Trees

```
library(rpart)
df <- gen.ex(1)
df1 <- make.grid(df)
fit <- rpart(group~., data=df[, 1:3], method = "class")
df1$group <- predict(fit, df1[, 1:2], type="class")
do.graph(df, df1)
```

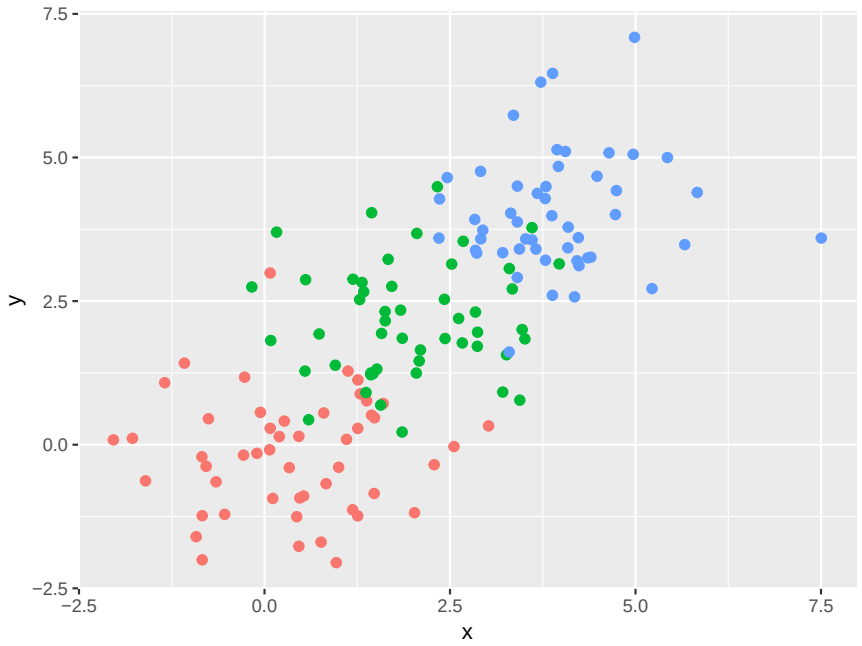



```
df <- gen.ex(2)
fit <-rpart(group~., data=df[, 1:3], method = "class")
df1 <- make.grid(df)
df1$group <- predict(fit, df1[, 1:2], type="class")
do.graph(df, df1)
```




```
df <- gen.ex(3)
fit <- rpart(group~., data=df[, 1:3], method = "class")
df1 <- make.grid(df)
df1$group <- predict(fit, df1[, 1:2], type="class")
```

```
do.graph(df, df1)
```



There are a number of ways to improve the performance of tree based methods. These are

- bagging (bootstrap aggregation)
- random forests

Both start with the same idea: take a random part of the data, fit a tree to it and use that for prediction. Then repeat this a number of times. Finally do the prediction by averaging.

Bagging

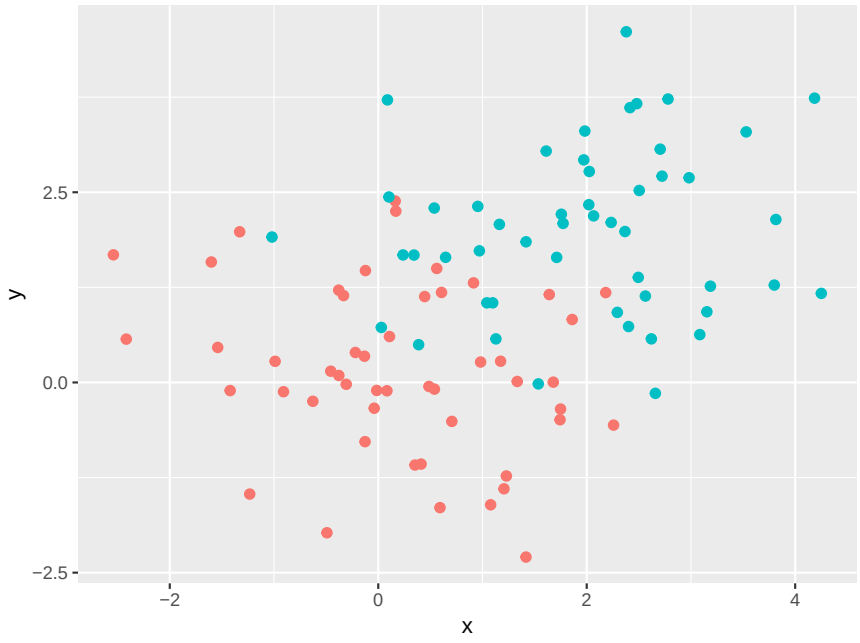
As the name suggests, here we apply the tree method to bootstap samples.

```
df <- gen.ex(1, n=50)[, 1:3]
n <- dim(df)[1]
L_tree <- list(1:100)
for(i in 1:100){
  I <- sample(1:n, size=n, replace=TRUE)
  L_tree[[i]] <- rpart(factor(group)~., data=df[I, ])
}
```

the aggregation is done by averaging:

```
df1 <- make.grid(df)
tmp <- predict(L_tree[[1]], df1)
df1$group <- ifelse(tmp[, 1]<0.5, "B", "A")
do.graph(df, df1)
```


```
for(i in 2:100)
  tmp <- tmp + predict(L_tree[[i]], df1)
df1$group <- ifelse(tmp[, 1]<50, "A", "B")
do.graph(df, df1)
```



alternatively we can predict the class directly and use majority rule:

```
df <- gen.ex(2, n=50)[, 1:3]
n <- dim(df)[1]
L_tree <- list(1:100)
```

```
for(i in 1:100){
  I <- sample(1:n, size=n, replace=TRUE)
  L_tree[[i]] <- rpart(factor(group)~., data=df[I, ],
                       method = "class")
}
df1 <- make.grid(df)
tmp <- matrix(nrow=10000, ncol=100)
for(i in 1:100)
  tmp[, i] <- predict(L_tree[[i]], df1, type="class")
df1$group <- ifelse(
  apply(tmp, 1, function(x) {sum(x==2)<50}),
        "A", "B")

do.graph(df, df1)
```

Random Forests

The major idea of random forests is that we only consider a random subset of predictors m each time we do a split on training examples. Whereas usually in trees we find all the predictors while doing a split and choose the best amongst them. Typically $m = \sqrt{p}$ where p are the number of predictors.

Now it seems crazy to throw away lots of predictors, but it makes sense because the effect of doing so is that each tree uses different predictors to split data at various times. This means that two trees generated on the same training data will have randomly different variables selected at each split, hence this is how the trees will get de-correlated and will be independent of each other.

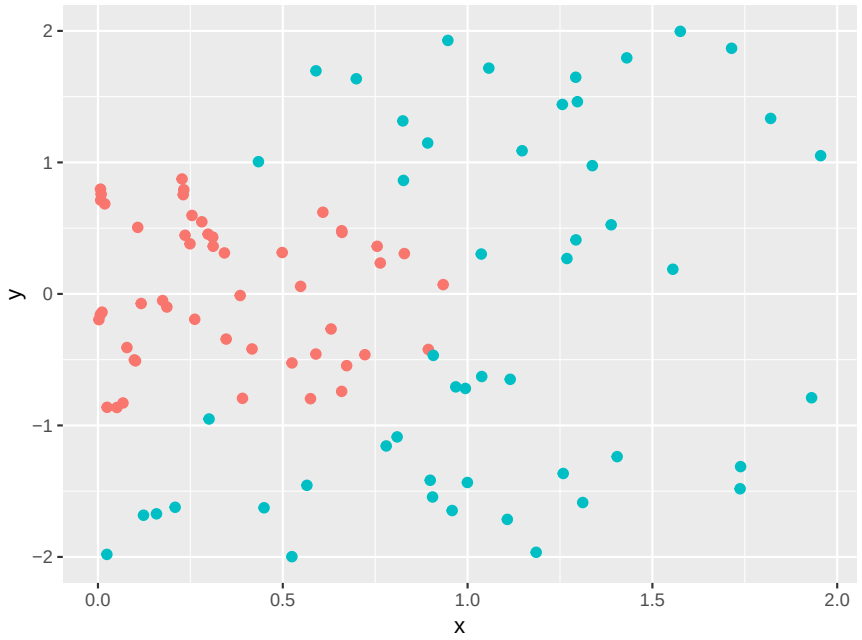
Another great thing about random forests and bagging is that we can keep on adding more and more big bushy trees and that won't hurt us because at the end we are just going to average them out which will reduce the variance by the factor of the number of Trees T itself.

So by doing this trick of throwing away predictors, we have de-correlated the Trees and the resulting average seems a little better.

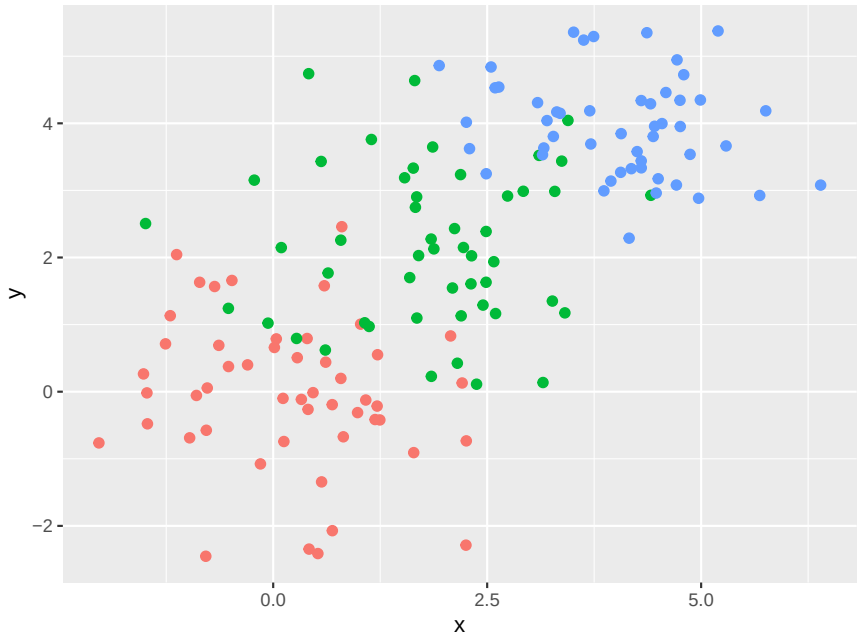
Here is how it works in R

```
library(randomForest)
df <- gen.ex(1)[, 1:3]
fit <- randomForest(factor(group)~., data=df)
df1 <- make.grid(df)
df1$group <- predict(fit, df1)
do.graph(df, df1)
```


```
df <- gen.ex(2)[, 1:3]
fit <- randomForest(factor(group)~., data=df)
df1 <- make.grid(df)
df1$group <- predict(fit, df1)
do.graph(df, df1)
```




```
df <- gen.ex(3)[, 1:3]
fit <- randomForest(factor(group)~., data=df)
df1 <- make.grid(df)
df1$group <- predict(fit, df1)
do.graph(df, df1)
```



Neural Networks and Support Vector Machines

Neural Networks

A *neural network* is a computer algorithm for fitting that is mostly a black box, that is understanding what it exactly does is not easy and beyond our level. For an explanation see Artifical Neural Networks.

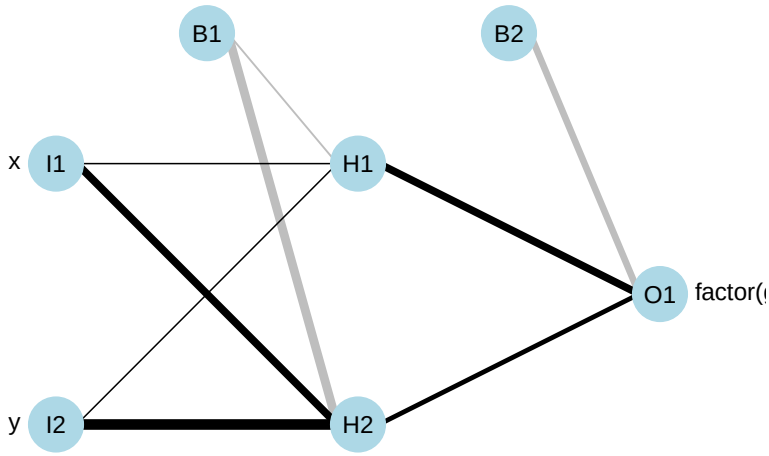
```
library(nnet)
df <- gen.ex(1)[ ,1:3]
fit <- nnet(factor(group)~., data=df, size=2)
```

```
## # weights:  9
## initial  value 75.481044
## iter  10 value 19.018153
## iter  20 value 18.850673
## iter  30 value 18.802697
## iter  40 value 18.703196
## iter  50 value 18.635669
## iter  60 value 18.593504
## iter  70 value 18.560864
## iter  80 value 18.549314
## iter  90 value 18.538397
## iter 100 value 18.525050
## final  value 18.525050
## stopped after 100 iterations
```

notice the use of class.ind because nnet requires a special format for the response variable.

to visualize a network we can use

```
library(NeuralNetTools)
par(mar=c(1, 1, 0, 1))
plotnet(fit)
```



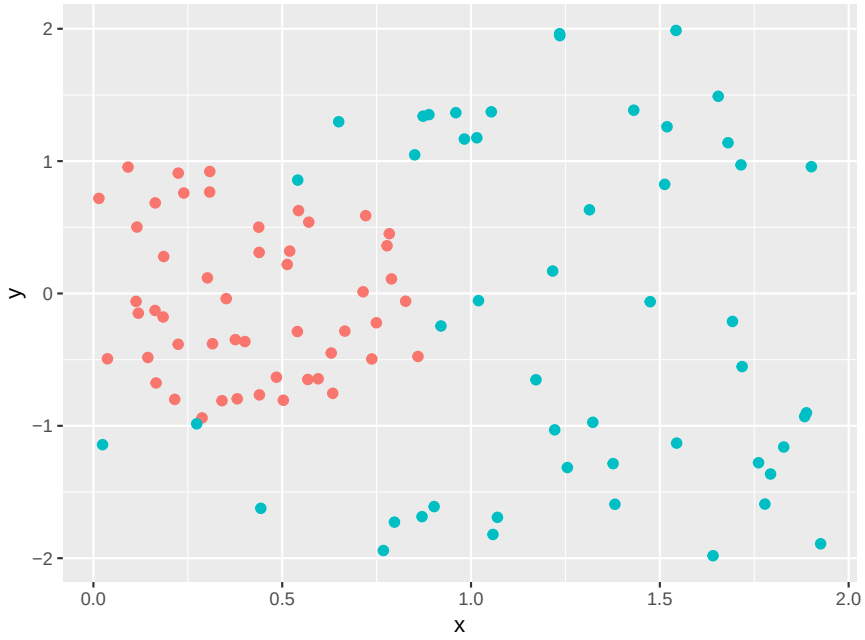
so the network has 2 **input layers** I1-I4, one for each predictor. It has two **hidden layers** because we chose size=2. It has two layers for **backward propagation** and finally it has two **output layers**.

```
df1 <- make.grid(df)
df1$group <- predict(fit, df1, type="class")
do.graph(df, df1)
```

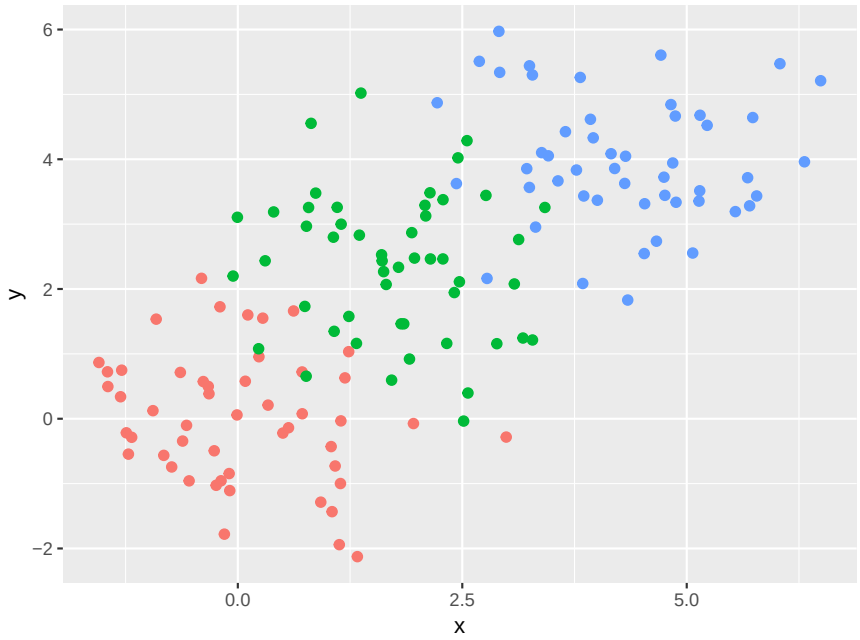

```
df <- gen.ex(2)[, 1:3]
fit <- nnet(factor(group)~., data=df, size=2)
```

```
## # weights:  9
## initial  value 77.164676
## iter  10 value 24.961422
## iter  20 value 21.508817
## iter  30 value 21.108154
## iter  40 value 20.558973
## iter  50 value 20.319070
## iter  60 value 20.287271
## iter  70 value 20.144824
## iter  80 value 20.111715
## iter  90 value 20.105023
## iter 100 value 19.964474
## final  value 19.964474
## stopped after 100 iterations
```

```
df1 <- make.grid(df)
df1$group <- predict(fit, df1, type="class", trace=0)
do.graph(df, df1)
```




```
df <- gen.ex(3)[, 1:3]
fit <- nnet(factor(group)~., data=df, size=2, trace=0)
df1 <- make.grid(df)
df1$group <- predict(fit, df1, type="class")
do.graph(df, df1)
```



Regression problems

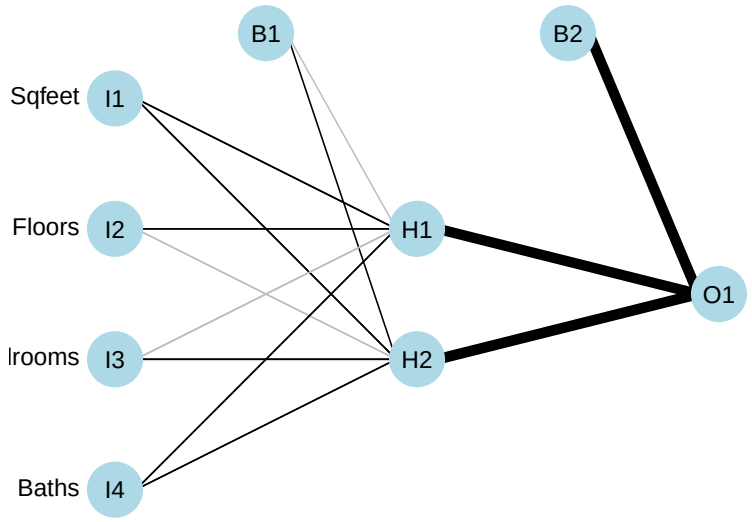
A *neural network* (also called a *perceptron*) is a general function fitter, and so can be used for regression as well. Here is an example

Example: House Prices

```
fit <- nnet(data.matrix(houseprice[, -1]),
            houseprice$Price, size=2, linout = 1)
```

```
## # weights:  13
## initial  value 653163.432931
## final  value 37992.517316
## converged
```

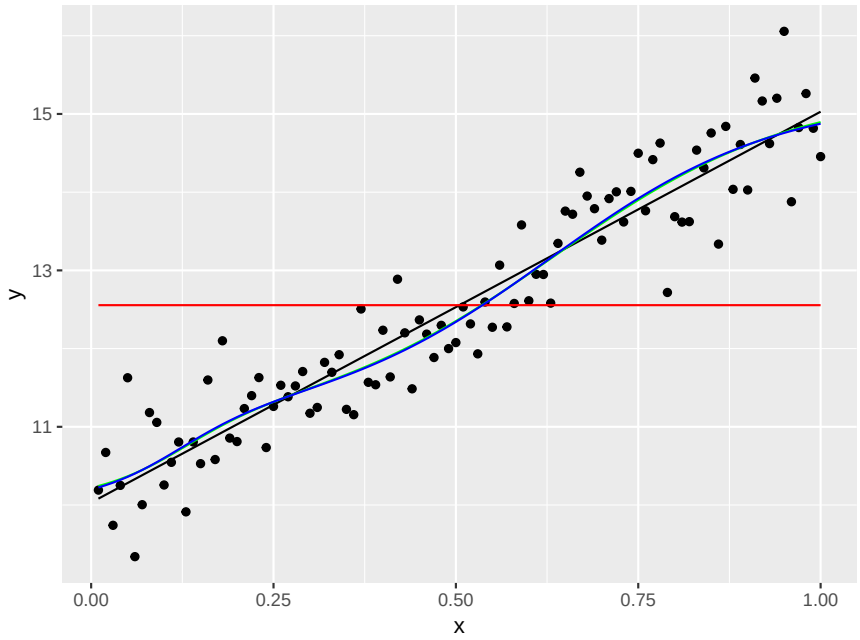
```
par(mar=c(1, 1, 0, 1))
plotnet(fit)
```



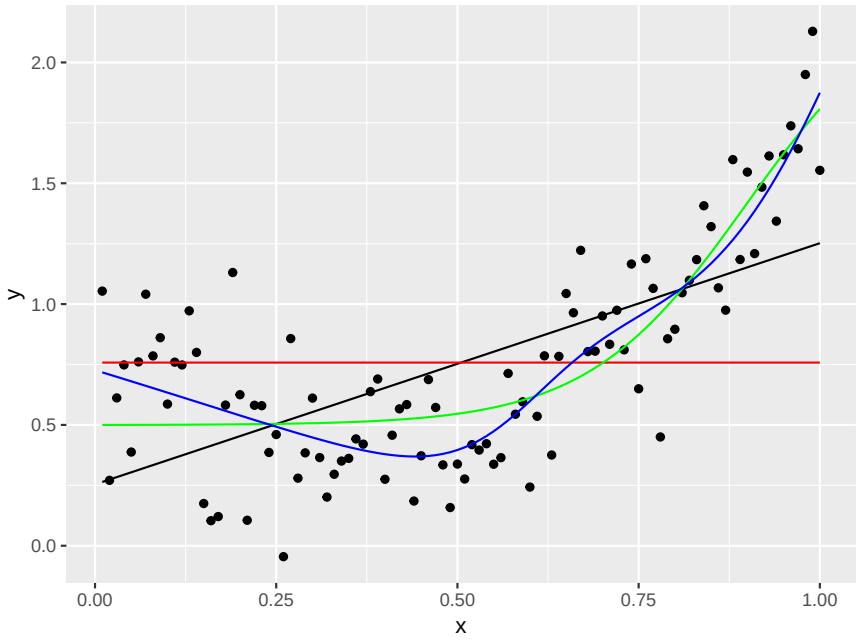
Example: Artificial Examples

Let's study this method using a few artificial examples:

```
x <- 1:100/100
y <- 10 + 5*x + rnorm(100, 0, 0.5)
df <- data.frame(x=x, y=y)
df$lm <- predict(lm(y~x))
df$nnet1 <- predict(nnet(x, y, size=1, linout = 1, trace = 0))
df$nnet2 <- predict(nnet(x, y, size=2, linout = 1, trace = 0))
df$nnet3 <- predict(nnet(x, y, size=3, linout = 1, trace = 0))
ggplot(data=df, aes(x, y)) +
  geom_point() +
  geom_line(data=data.frame(x=x, y= df$lm), aes(x, y),
            inherit.aes = FALSE) +
  geom_line(data=data.frame(x=x, y= df$nnet1), aes(x, y),
            inherit.aes = FALSE, color="red") +
  geom_line(data=data.frame(x=x, y= df$nnet2), aes(x, y),
            inherit.aes = FALSE,color="green") +
  geom_line(data=data.frame(x=x, y= df$nnet3), aes(x, y),
            inherit.aes = FALSE, color="blue")
```




```
y <- x + 3*(x-0.5)^2 + rnorm(100, 0, 0.25)
df <- data.frame(x=x, y=y)
df$lm <- predict(lm(y~x))
df$nnet1 <- predict(nnet(x, y, size=1, linout = 1, trace = 0))
df$nnet2 <- predict(nnet(x, y, size=2, linout = 1, trace = 0))
df$nnet3 <- predict(nnet(x, y, size=3, linout = 1, trace = 0))
ggplot(data=df, aes(x, y)) +
  geom_point() +
  geom_line(data=data.frame(x=x, y= df$lm), aes(x, y),
            inherit.aes = FALSE) +
  geom_line(data=data.frame(x=x, y= df$nnet1), aes(x, y),
            inherit.aes = FALSE, color="red") +
  geom_line(data=data.frame(x=x, y= df$nnet2), aes(x, y),
            inherit.aes = FALSE,color="green") +
  geom_line(data=data.frame(x=x, y= df$nnet3), aes(x, y),
            inherit.aes = FALSE, color="blue")
```



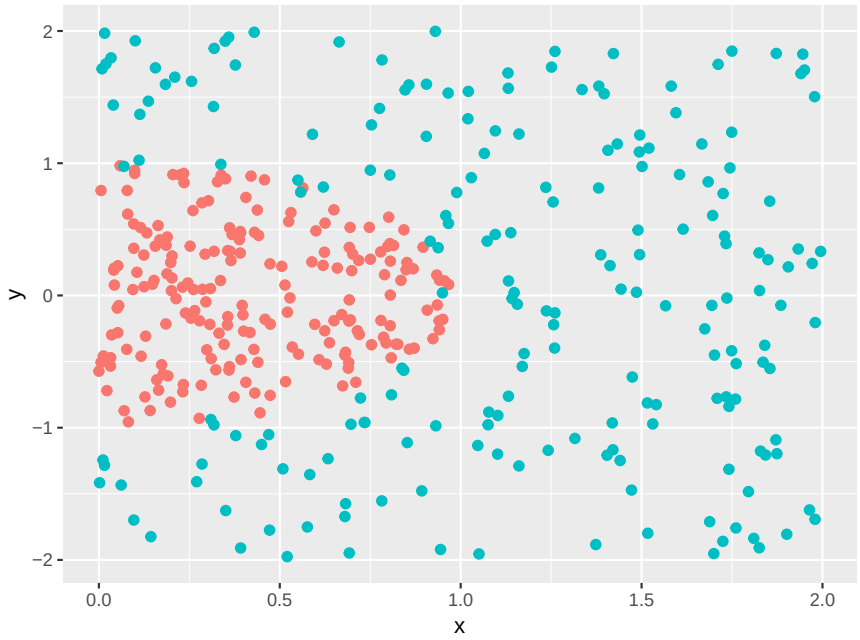
so higher number of hidden layers leads to a more complicated fit.

It is often tricky to know how many layers to use, and it is easy to overfit. Generally something like cross-validation is needed.

Support Vector Machines

This is another modern method for classification. Its idea is at first very strange. Let's have another look at example 2:

```
df <- gen.ex(2, 200)
ggplot(data=df, aes(x, y, color=group)) +
  geom_point(size=2) +
  theme(legend.position="none")
```

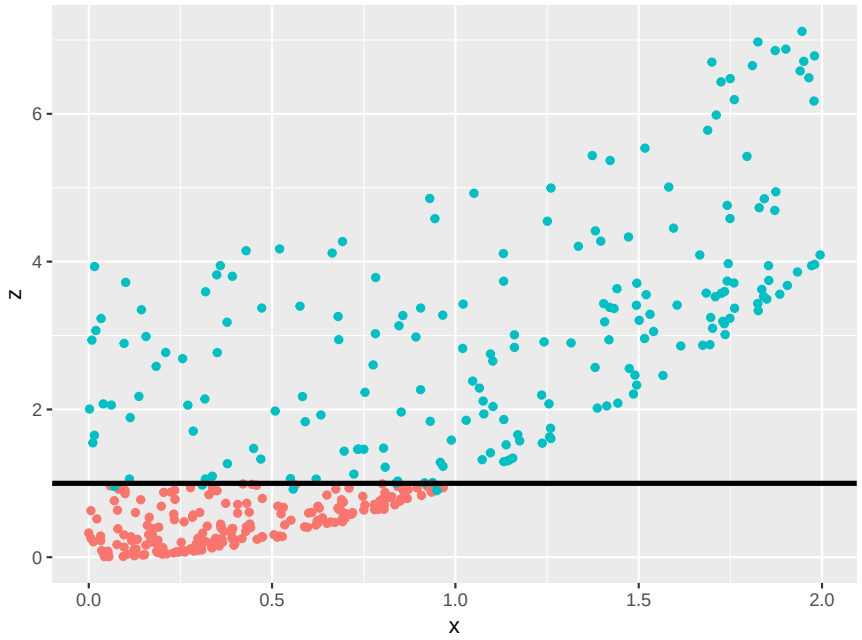


Let's say we defined a new variable z by

$$z = x^2 + y^2$$

then

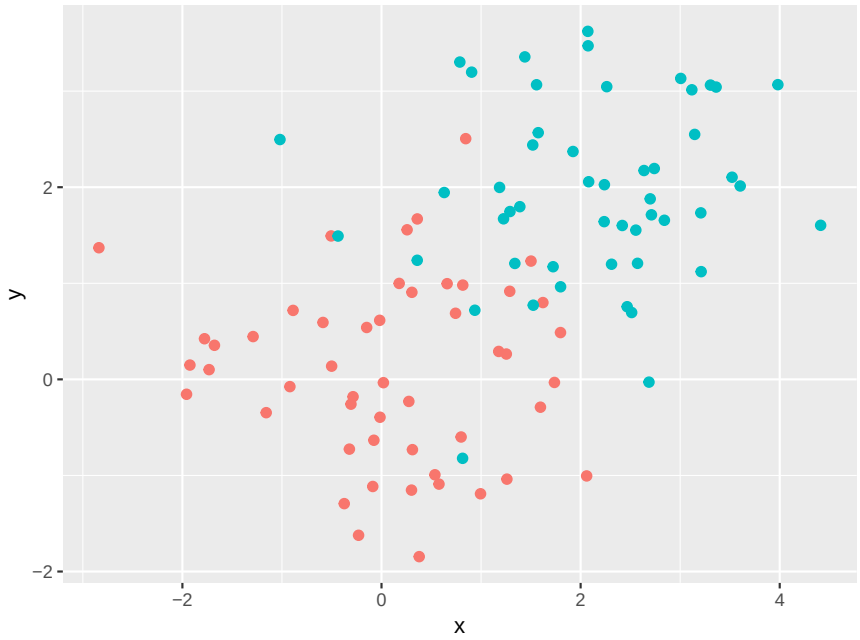
```
df$z <- df$x^2+df$y^2
ggplot(data=df, aes(x, z, color=group)) +
  geom_point() +
  geom_hline(yintercept = 1, size=1.2)+
  theme(legend.position="none")
```



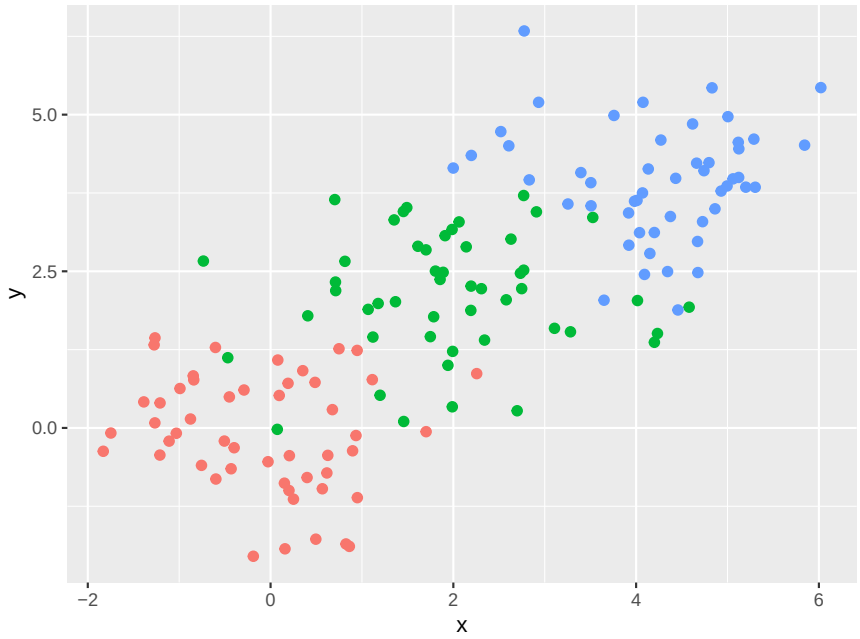
and so suddenly we would have a very simple decision rule: declare red is $z < 1$!

Adding a variable z is like adding an additional dimension. If we could display the data in (x, y, z) space there would a *separating hyperplane*. It can be shown that by adding sufficient dimensions there will eventually always be a hyperplane that perfectly separates the groups. SVM tries to find such a hyperplane without us having to specify a function, like above. It is implemented in R in the package

```
library(e1071)
df <- gen.ex(1)[, 1:3]
fit <- svm(factor(group)~., data=df)
df1 <- make.grid(df)
df1$group <- predict(fit, df1)
do.graph(df, df1)
```

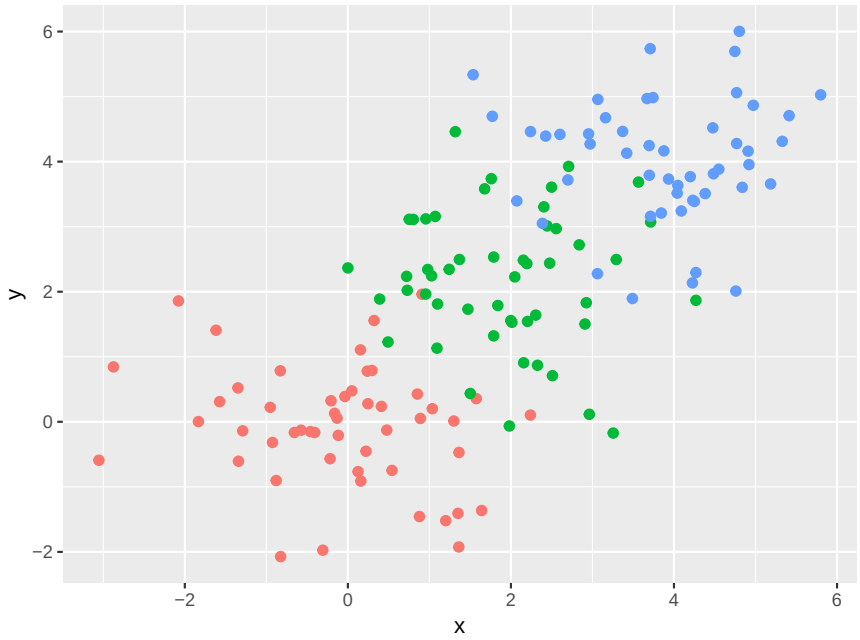



```
df <- gen.ex(3)[, 1:3]
fit <- svm(factor(group)~., data=df)
df1 <- make.grid(df)
df1$group <- predict(fit, df1)
do.graph(df, df1)
```




```
df <- gen.ex(3)[, 1:3]
fit <- svm(factor(group)~., data=df)
df1 <- make.grid(df)
df1$group <- predict(fit, df1)
```

```
do.graph(df, df1)
```




```
library(MASS)
library(rpart)
library(class)
library(nnet)
library(e1071)
```

Classification Examples

In this section we will study the performance of the classification methods discussed earlier. We will use the miss-classification rate and cross-validation.

As we saw, each of the methods has a slightly different list of arguments. It will therefore be worthwhile to write a single routine that does them all.

```
do.class <- function(df, I, B=100, which=1:6) {
  miss.rate <- matrix(0, B, 7)
  n <- dim(df)[1]
  colnames(miss.rate) <- c("LDA", "QDA", "Tree", "NN",
                           "SVM", "knn 3", "knn 9")
  for(i in 1:B) {
    I <- sample(1:n, size=floor(n/2))
    train <- df[I, ]
    colnames(train)[1] <- "group"
    if(1 %in% which) {
      fit <- lda(group~., data=train)
      pred <- predict(fit, df[-I, -1])$class
```

```
      miss.rate[i, "LDA"] <- msr(factor(df[-I, 1]), pred)
    }
    if(2 %in% which) {
      fit <- qda(group~., data=train)
      pred <- predict(fit, df[-I, -1])$class
      miss.rate[i, "QDA"] <- msr(factor(df[-I, 1]), pred)
    }
    if(3 %in% which) {
      fit <-rpart(group~., data=train, method = "class")
      pred <- predict(fit, df[-I, -1], type="class")
      miss.rate[i, "Tree"] <- msr(factor(df[-I, 1]), pred)
    }
    if(4 %in% which) {
      fit <- nnet(factor(group)~., data=train, size=2,
                  rang = 0.1, trace=0,
                 decay = 5e-4, maxit = 200)
      pred <- predict(fit, df[-I, -1], type="class")
      miss.rate[i, "NN"] <- msr(df[-I, 1], pred)
    }
    if(5 %in% which) {
      fit <- svm(factor(group)~., data=train)
      pred <- predict(fit, df[-I, -1])
      miss.rate[i, "SVM"] <- msr(df[-I, 1], pred)
    }
    if(6 %in% which) {
      pred <-  factor(
       knn(df[I, -1], df[-I, -1], cl=df[I, 1], k=3))
      miss.rate[i, "knn 3"] <- msr(factor(df[-I, 1]), pred)
    }
    if(7 %in% which) {
      pred <-  factor(
       knn(df[I, -1], df[-I, -1], cl=df[I, 1], k=9))
      miss.rate[i, "knn 9"] <- msr(factor(df[-I, 1]), pred)
    }
  }
  apply(miss.rate[, which], 2, mean)
}
```

```
df <- gen.ex(1)
sort(do.class(df[, c(3, 1, 2)]))
```

```
##   LDA   QDA   SVM    NN knn 3  Tree
##  7.90  8.04  8.34 10.36 11.62 16.52
```

```
df <- gen.ex(2)
sort(do.class(df[, c(3, 1, 2)]))
```

```
##   SVM knn 3    NN   QDA   LDA  Tree
##  6.92  8.02 10.10 11.56 18.64 19.68
```

```
df <- gen.ex(3)
sort(do.class(df[, c(3, 1, 2)]))
```

```
##    LDA    QDA    SVM     NN  knn 3   Tree
## 14.123 14.468 15.131 15.774 15.963 21.921
```

Example: Fisher's Iris

```
sort(do.class(iris[, c(5, 1:4)]))
```

```
##   LDA   QDA    NN knn 3   SVM  Tree
## 2.640 2.788 4.425 4.547 4.859 6.184
```

Example: Kypthosis

```
sort(do.class(kyphosis))
```

```
##    LDA    SVM    QDA     NN   Tree  knn 3
## 19.400 19.712 20.326 23.200 23.424 23.883
```

QDA does not work here. Essentially there is not enough data to fit a quadratic model. So

```
sort(do.class(kyphosis, which=c(1, 3:7)))
```

```
##    LDA    SVM  knn 9     NN   Tree  knn 3
## 21.091 21.376 23.134 24.183 24.206 24.979
```

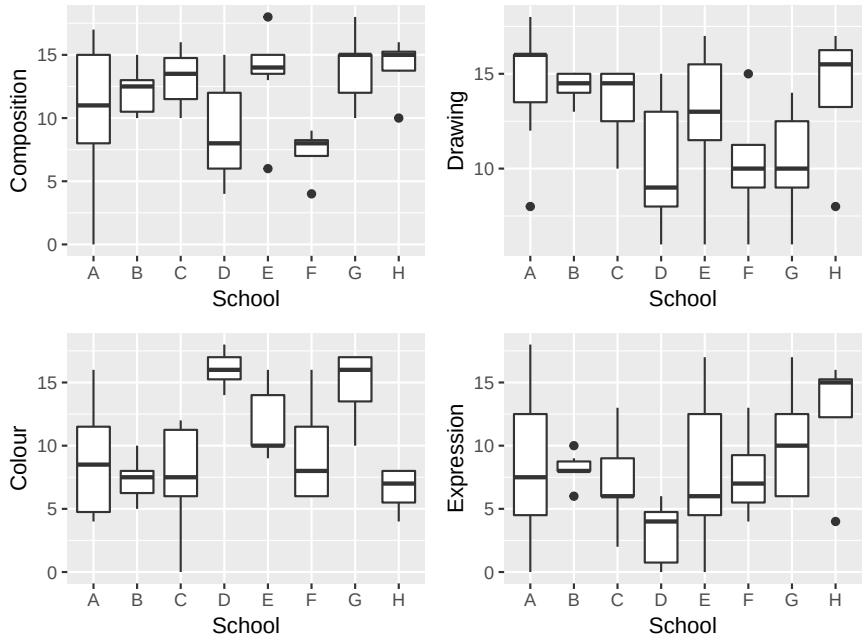
Example: Painters

The subjective assessment, on a 0 to 20 integer scale, of 54 classical painters. The painters were assessed on four characteristics: composition, drawing, colour and expression. They were also grouped in 8 "Schools". The data is due to the Eighteenth century art critic, de Piles.

```
kable.nice(head(painters))
```

	Composition	Drawing	Colour	Expression	School
Da Udine	10	8	16	3	A
Da Vinci	15	16	4	14	A
Del Piombo	8	13	16	7	A
Del Sarto	12	16	9	8	A
Fr. Penni	0	15	8	0	A
Guilio Romano	15	16	4	14	A

```
pushViewport(viewport(layout = grid.layout(2, 2)))
print(ggplot(data=painters, aes(School, Composition)) +
          geom_boxplot(),
    vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(ggplot(data=painters, aes(School, Drawing)) +
          geom_boxplot(),
    vp=viewport(layout.pos.row=1, layout.pos.col=2))
print(ggplot(data=painters, aes(School, Colour)) +
          geom_boxplot(),
    vp=viewport(layout.pos.row=2, layout.pos.col=1))
print(ggplot(data=painters, aes(School, Expression)) +
          geom_boxplot(),
    vp=viewport(layout.pos.row=2, layout.pos.col=2))
```




```
sort(do.class(painters[, c(5, 1:4)]))
```

```
## Error in qda.default(x, grouping, ...): some group is too small for 'qda'
```

Again QDA does not work here. So

```
sort(do.class(painters[, c(5, 1:4)], which=c(1, 3:7)))
```

```
##    LDA    SVM  knn 3   Tree  knn 9     NN
## 71.620 76.838 77.430 79.095 80.723 83.572
```

and this is clearly a very difficult classification problem, none of the methods does well.

It should also be pointed out that we have used all these methods essentially with their defaults. In real life one would play around with the tuning parameters to get better performance.

Special Topics

Survival Analysis

Survival Analysis is concerned with the distributions of lifetimes, often human as in Medical Sciences but also of components and machines.

Example: Life Table USA

Here is a *life table*, for the USA in 2016. Numbers are per 100,000 population:

```
life.us <- read.csv("us.life.table.csv")
kable.nice(life.us)
```

Age	Male	Female
<1 year	100000.00	100000.00
1-4 years	99378.94	99454.61
5-9 years	99281.21	99372.41
10-14 years	99216.42	99317.51
15-19 years	99133.12	99255.05
20-24 years	98772.20	99105.56
25-29 years	98039.19	98849.90
30-34 years	97148.16	98491.30
35-39 years	96152.90	98007.48
40-44 years	95006.17	97365.09
45-49 years	93692.59	96542.95
50-54 years	91897.80	95333.41
55-59 years	89193.00	93486.77
60-64 years	85086.59	90760.06
65-69 years	79496.85	87075.37
70-74 years	72184.16	81854.51
75-79 years	62329.23	74259.41
80-84 years	50291.00	63544.39
85+ years	35033.57	48766.34

so this tells us that of any 100,000 men 97148.16 survived to age 30, or about \$2850/100000*100% = 2.8% had died, compared to 1.5% of the women.

Tables like this are very important to insurance companies when trying to figure out what premiums to charge for a life insurance, or for a company to find out how much their employees need to pay into a retirement plan, etc.

Let's draw a curve that for each gender and age shows the proportion of the population alive.

First we need to reorganize the data as a data frame with numeric columns:

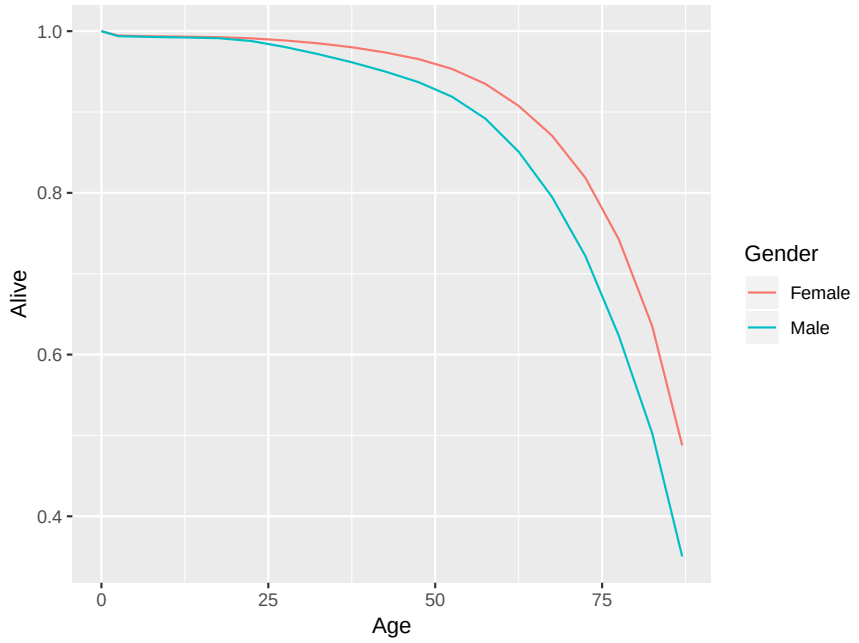
```
dim(life.us)
```

```
## [1] 19  3
```

```
life.us.1 <-
  data.frame(Age=rep(c(0, 2.5+5*0:16, 87), 2),
             Alive=c(life.us$Male, life.us$Female)/1e5,
             Gender=rep(c("Male", "Female"), each=19))
```

and now

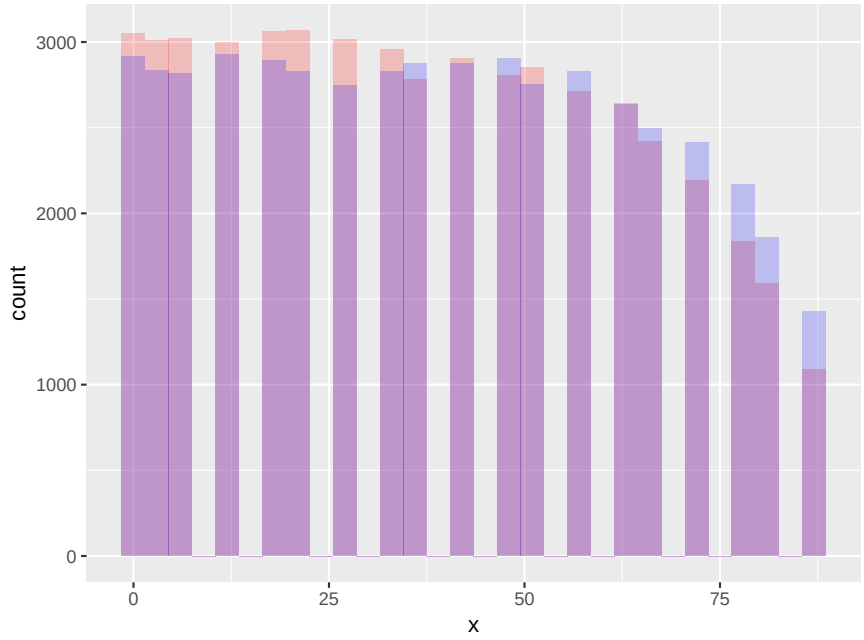
```
ggplot(data=life.us.1, aes(Age, Alive, color=Gender)) +
    geom_line()
```



What does this say how many newborn male babies we might have in a certain population? How may 1 year olds and so on? Let's assume our population consists of 100000 individuals, half male and female. Then the numbers in the table tell us the relative probabilities. For example for any 200 babies there should be about 98.8 20-24 year old males and 63.5 80-84 year old females. So we can generate a simulated population with

```
sim.pop <- list(Male=sample(c(0, 2.5+5*0:16, 87),
   size=50000, replace = TRUE, prob=life.us$Male),
                Female=sample(c(0, 2.5+5*0:16, 87),
   size=50000, replace = TRUE, prob=life.us$Female))
df <- data.frame(x=c(sim.pop$Male, sim.pop$Female),
      Gender=rep(c("Male", "Female"), each=50000))
ggplot(df, aes(x=x)) +
    geom_histogram(data = subset(df, Gender == "Male"),
        fill = "red", alpha = 0.2) +
```

```
    geom_histogram(data = subset(df, Gender == "Female"),
        fill = "blue", alpha = 0.2)
```



Let T denote a random variable describing a lifetime. Then we know that T takes values x>0 and T has a continuous distribution F with density f. In survival analysis several functions are of common interest:

- survivor function:

$$S(t) = 1 - F(t) = P(X > t)$$

  which is the probability to survive past time t.

- hazard function:

$$h(t) = \lim_{t \to \infty} \frac{P(t < T < t + h)}{h}$$

which is the probability to survive until time t, and then die.

- cumulative hazard function:

$$H(t) = \int_0^\infty h(t)dt$$

it is easy to check that

- $h(t) = f(t)/S(t)$
- $H(t) = -\log S(t)$

Here are some standard survival distributions and their associated functions:

- **Exponential**
  - $f(t) = \lambda \exp(-\lambda t)$
  - $S(t) = \exp(-\lambda t)$
  - $h(t) = \lambda$
  - $H(t) = \lambda t$

  λ is called rate parameter
- **Weibull**
  - $f(t) = \frac{\alpha}{\lambda}(\frac{t}{\lambda})^{\alpha-1} \exp\left(-(\frac{t}{\lambda})^{\alpha}\right)$
  - $S(t) = \exp\left(-(\frac{t}{\lambda})^{\alpha}\right)$
  - $h(t) = \frac{\alpha}{\lambda}(\frac{t}{\lambda})^{\alpha-1}$
  - $H(t) = (\frac{t}{\lambda})^{\alpha}$

  α is called the shape parameter and λ the scale parameter
- **Log-Logistic**
  - $f(t) = \frac{\lambda\tau(\lambda t)^{\tau-1}}{(1+(\lambda t)^{\tau})^2}$
  - $S(t) = \frac{1}{1+(\lambda t)^{\tau}}$
  - $h(t) = \frac{\lambda\tau(\lambda t)^{\tau-1}}{1+(\lambda t)^{\tau}}$
  - $H(t) = \log(1 + (\lambda t)^{\tau})$

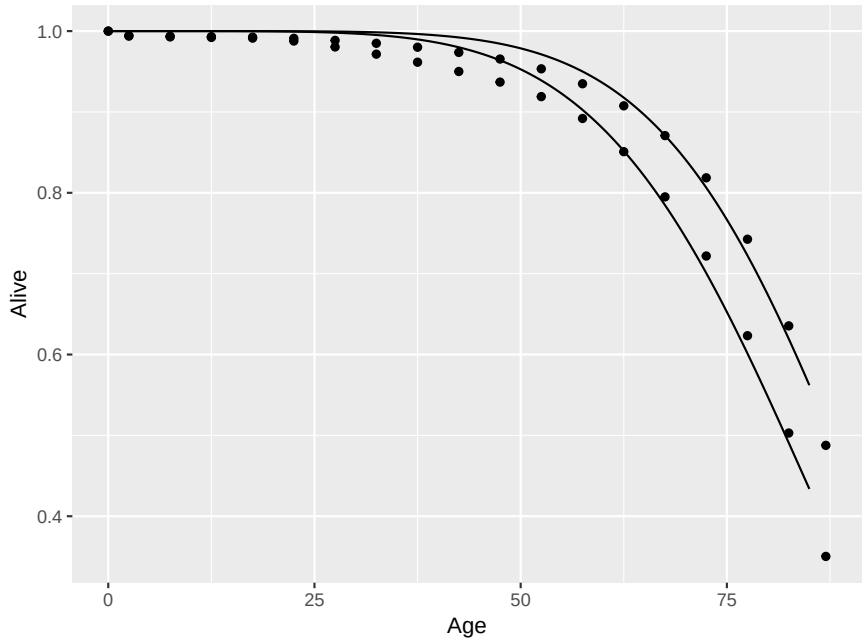
also often used are the log-normal and the gamma distributions.

Example: Life table USA

Let's see whether we can fit a Weibull distribution to our data:

```
y <- life.us.1$Alive[life.us.1$Gender=="Male"]
x <- life.us.1$Age[life.us.1$Gender=="Male"]
fit.male <- coef(nls(y~exp(-(x/l)^a),
                     start=list(l=100, a=4)))
y <- life.us.1$Alive[life.us.1$Gender=="Female"]
x <- life.us.1$Age[life.us.1$Gender=="Female"]
fit.female <- coef(nls(y~exp(-(x/l)^a),
                       start=list(l=100, a=4)))
x <- seq(0, 85, length=250)
df.male <- data.frame(x=x,
          y=exp(-(x/fit.male[1])^fit.male[2]))
df.female <- data.frame(x=x,
          y=exp(-(x/fit.female[1])^fit.female[2]))
```

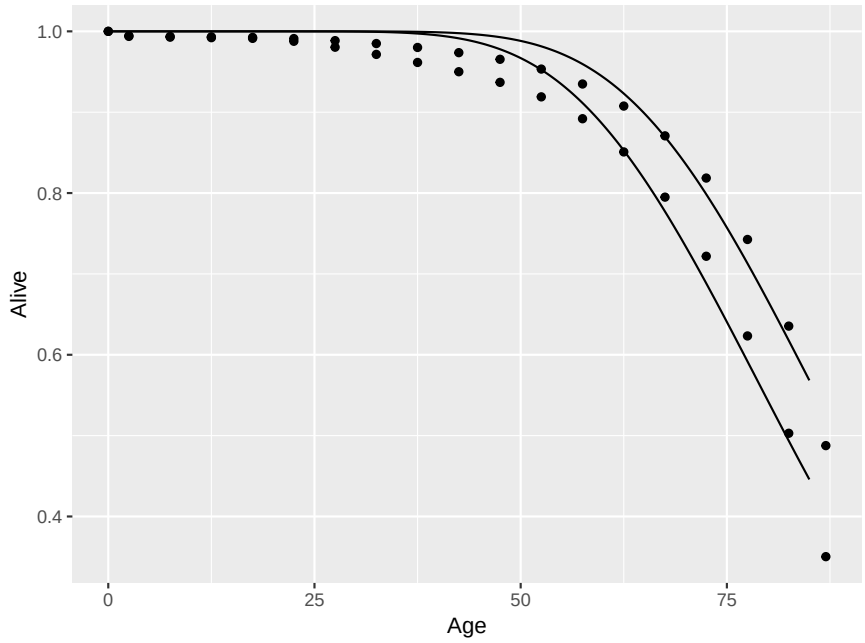
```
ggplot(data=life.us.1, aes(Age, Alive)) +
    geom_point() +
  geom_line(data=df.male, aes(x, y)) +
  geom_line(data=df.female, aes(x, y))
```



and these fits are ok but not great.

How about a gamma distribution?

```
y <- life.us.1$Alive[life.us.1$Gender=="Male"]
x <- life.us.1$Age[life.us.1$Gender=="Male"]
fit.male <- coef(nls(y~(1-pgamma(x, a, b)),
                     start=list(a=10, b=0.1)))
y <- life.us.1$Alive[life.us.1$Gender=="Female"]
x <- life.us.1$Age[life.us.1$Gender=="Female"]
fit.female <- coef(nls(y~(1-pgamma(x, a, b)),
                     start=list(a=10, b=0.1)))
x <- seq(0, 85, length=250)
df.male <- data.frame(x=x,
          y=1-pgamma(x, fit.male[1], fit.male[2]))
df.female <- data.frame(x=x,
          y=1-pgamma(x, fit.female[1], fit.female[2]))
ggplot(data=life.us.1, aes(Age, Alive)) +
    geom_point() +
  geom_line(data=df.male, aes(x, y)) +
  geom_line(data=df.female, aes(x, y))
```



and that looks a bit worse.

Example: Leukemia

This is survival times for leukemia, with covariates wbc, the white blood cell count and ag, a test result with values "present" or "absent".

```
kable.nice(head(leukemia))
```

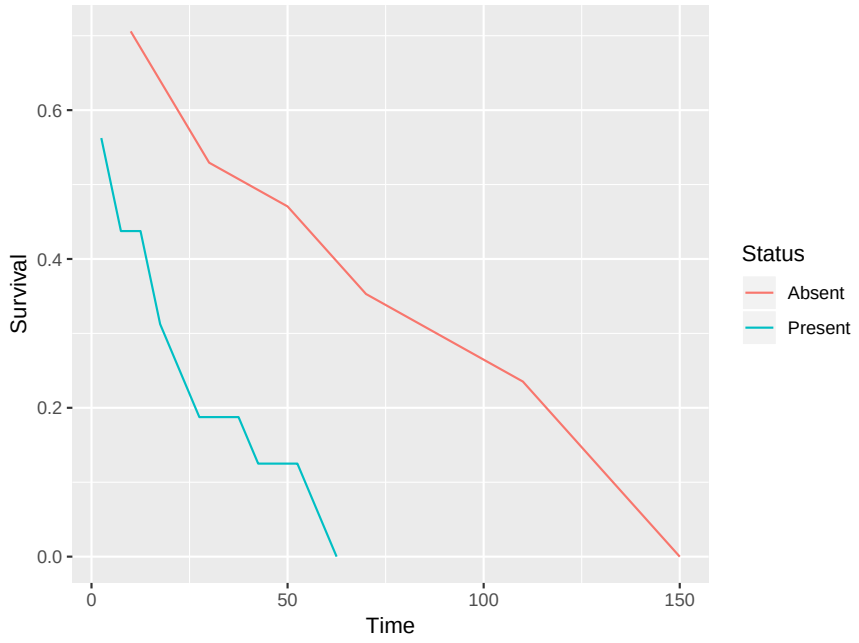
wbc	ag	time
2300	present	65
750	present	156
4300	present	100
2600	present	134
6000	present	16
10500	present	108

```
empdist <- function (data, npoints)
{
    a <- hist(data, breaks = npoints, plot = F)
    x <- a$mids
    y <- cumsum(a$counts)/length(data)
    list(x = x, y = y)
}
zp <- empdist(leukemia$time[leukemia$ag == "present"], 10)
za <- empdist(leukemia$time[leukemia$ag == "absent"], 10)
```

```r
df <- data.frame(Time=c(za$x, zp$x),
                 Survival=1-c(za$y, zp$y),
                 Status=rep(c("Present", "Absent"),
                c(length(za$x), length(zp$x))))
df
```

```
##     Time  Survival  Status
## 1    2.5 0.5625000 Present
## 2    7.5 0.4375000 Present
## 3   12.5 0.4375000 Present
## 4   17.5 0.3125000 Present
## 5   22.5 0.2500000 Present
## 6   27.5 0.1875000 Present
## 7   32.5 0.1875000 Present
## 8   37.5 0.1875000 Present
## 9   42.5 0.1250000 Present
## 10  47.5 0.1250000 Present
## 11  52.5 0.1250000 Present
## 12  57.5 0.0625000 Present
## 13  62.5 0.0000000 Present
## 14  10.0 0.7058824  Absent
## 15  30.0 0.5294118  Absent
## 16  50.0 0.4705882  Absent
## 17  70.0 0.3529412  Absent
## 18  90.0 0.2941176  Absent
## 19 110.0 0.2352941  Absent
## 20 130.0 0.1176471  Absent
## 21 150.0 0.0000000  Absent
```

```r
ggplot(data=df, aes(Time, Survival, color=Status)) +
    geom_line()
```



Censored Data

The most distinctive feature of survival analysis is **censoring**. Say we are testing a new treatment for late-stage stomach cancer. At the beginning of the study we select 100 cancer patients and begin treatment. After 1 year we wish to study the effectiveness of the new treatment. At that time 47 of the patients have died, and for them we know the number of days until death. For the other 53 patients we know that they have survived 365 days but we don't know how much longer they will live. How do we use all the available information?

Censoring comes in a number of different forms:

- right censoring: here "case i" leaves the trial at time C_i, and we either know T_i if $T_i \leq C_i$, or that $T_i > C_i$.
- we have random censoring if T_i and C_i are independent.
- type I censoring is when the censoring times are fixed in advance, for example by the end of the clinical trial.
- type II censoring is when the the experiment is terminated after a fixed number of failures.

In R we code the data as a pair (t_i, d_i) where t_i is the observed survival time and $d_i = 0$ if the observation is censored, 1 if a "death" is observed.

Survival data is often presented using a + for the censored observations, for example 35, 67+, 85+, 93, 101, etc.

Let $t_1 < t_2 < .. < t_m$ be the m distinct survival times. Let $Y_i(s)$ be an indicator function, which is 1 if person i is still at risk (that is alive) at time s and 0 otherwise, that is $Y_i(s) = I_{(0,t_i)}(s)$.

Then the number of patients at risk at time s is $r(s) = \sum Y_i(s)$. We can similarly define d(s) as the number of deaths occurring at time s.

There is also a modern approach to survival data based on stochastic processes, especially martingales and counting processes. In this setup one considers the counting process $N_i(t)$ associated with the ith subject, so $N_i(t)$ changes by 1 at each observed event (for example, a visit to the doctor, a heart attack, a death etc).

How can we estimate the survivor curve in the presence of censoring? One of the most commonly used estimators is the **Kaplan-Meier** estimator:

$$\hat{S}(x) = \prod_{t_i<t} \frac{r(t_i) - d(t_i)}{r(t_i)}$$

In R this is calculated by the routine *survfit*. It uses as its argument a "survival" object which in turn is generated using the Surv function.

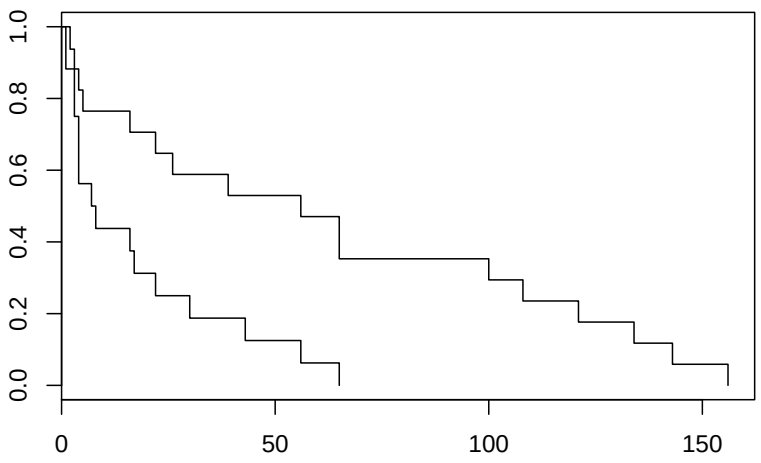
Example: Leukemia

This data does not have censoring, bit let's see what Kaplan Meier looks like anyway:

```
kable.nice(head(leukemia))
```

wbc	ag	time
2300	present	65
750	present	156
4300	present	100
2600	present	134
6000	present	16
10500	present	108

```
library(survival)
fit <- survfit(Surv(time) ~ ag, data = leukemia)
plot(fit)
```



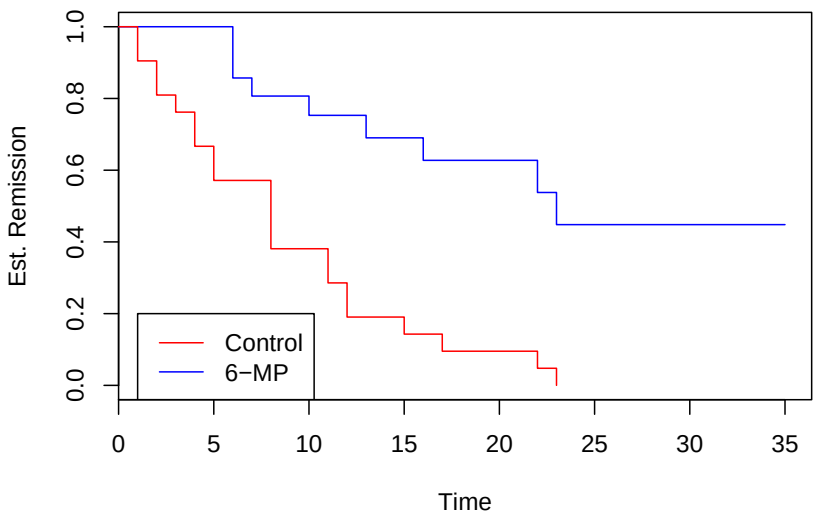
Example: Gehan data

A data frame from a trial of 42 leukaemia patients. Some were treated with the drug *6-mercaptopurine* and the rest are controls. The trial was designed as matched pairs, both withdrawn from the trial when either came out of remission.

```
kable.nice(head(gehan))
```

pair	time	cens	treat
1	1	1	control
1	10	1	6-MP
2	22	1	control
2	7	1	6-MP
3	3	1	control
3	32	0	6-MP

```
fit <- survfit(Surv(time, cens) ~ treat, data = gehan)
plot(fit, xlab = "Time", ylab = "Est. Remission",
     col = c("blue", "red"))
legend(1, 0.2, c("Control", "6-MP"),
       col = c("red", "blue"), lty = c(1, 1))
```



The obvious question is whether there are differences between the remission times of the two groups:

```
survdiff(Surv(time, cens) ~ treat, data = gehan)
```

```
## Call:
## survdiff(formula = Surv(time, cens) ~ treat, data = gehan)
##
##                 N Observed Expected (O-E)^2/E (O-E)^2/V
## treat=6-MP     21        9     19.3      5.46      16.8
## treat=control 21       21     10.7      9.77      16.8
##
##  Chisq= 16.8  on 1 degrees of freedom, p= 4e-05
```

and we see that there is.

Confidence intervals for the survival curve are automatically computed by the survfit function. By default it finds intervals of the form $S \pm 1.96se(S)$. Other options are:

- conf.type="log": $\exp(\log(S) \pm 1.96se(H))$
- conf.type="log-log": $\exp(-\exp(\log(-\log(S) \pm 1.96se(\log(H)))$

One advantage of log-log is that the limits are always in (0,1).

Let's find 90% confidence intervals based on these three methods for t=10 in the 6-MP group.

```
A <- matrix(0, 3, 2)
dimnames(A) <- list(c("plain", "log", "log-log"),
                    c("Lower", "Upper"))
fit <- survfit(Surv(time, cens) ~ treat, data = gehan,
            conf.type = "plain", conf.int = 0.9)
```

```
A[1, 1] <- fit$lower[fit$time == "10"]
A[1, 2] <- fit$upper[fit$time == "10"]
fit <- survfit(Surv(time, cens) ~ treat, data = gehan,
            conf.type = "log", conf.int = 0.9)
A[2, 1] <- fit$lower[fit$time == "10"]
A[2, 2] <- fit$upper[fit$time == "10"]
fit <- survfit(Surv(time, cens) ~ treat, data = gehan,
            conf.type = "log-log", conf.int = 0.9)
A[3, 1] <- fit$lower[fit$time == "10"]
A[3, 2] <- fit$upper[fit$time == "10"]
round(A, 2)
```

```
##         Lower Upper
## plain    0.59  0.91
## log      0.61  0.93
## log-log  0.55  0.87
```

An important question for new patients is of course the average survival time. As always "average" can be computed in different ways. R uses the median and the estimate is part of the output for a survfit object

```
survfit(Surv(time, cens) ~ treat, data = gehan)
```

```
## Call: survfit(formula = Surv(time, cens) ~ treat, data = gehan)
##
##                n events median 0.95LCL 0.95UCL
## treat=6-MP    21      9     23      16      NA
## treat=control 21     21      8       4      12
```

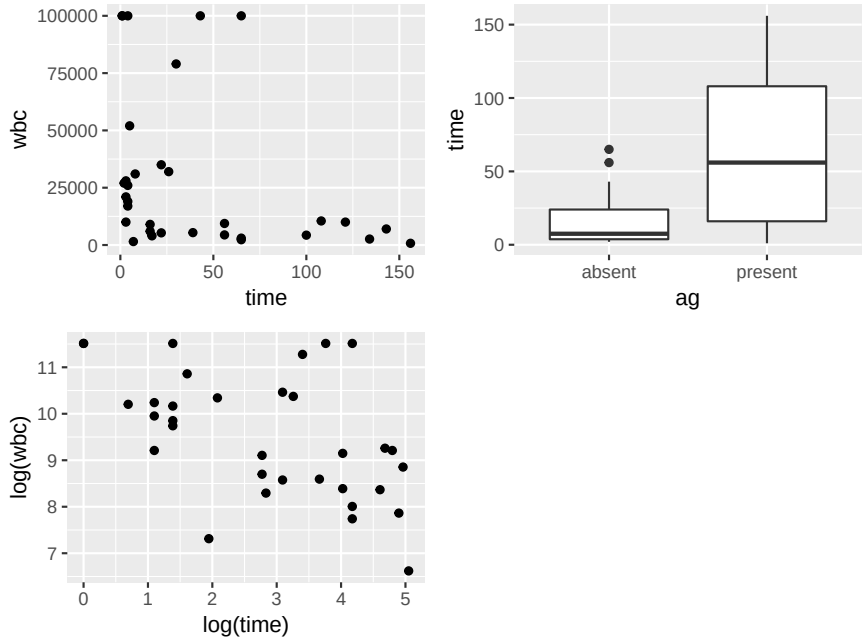
Sometimes it is possible to fit a parametric curve via generalized linear models to the survival curve. Say we wish to fit an exponential model, and suppose we have a covariate vector x for each case. Say λ_i is the rate of the exponential for case i. Then we can connect the rate to the covariates via the model $\lambda_i = \beta^T x$. This might on occasion be negative, and it is often better to use $\log(\lambda_i) = \beta^T x$.

Example: Leukemia

Let's see whether the Leukemia data can be modeled in this way. First we have a look at the relationship of time and its covariates:

```
pushViewport(viewport(layout = grid.layout(2, 2)))
print(ggplot(data=leukemia, aes(time, wbc)) +
        geom_point(),
  vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(ggplot(data=leukemia, aes(ag, time)) +
        geom_boxplot(),
  vp=viewport(layout.pos.row=1, layout.pos.col=2))
print(ggplot(data=leukemia, aes(log(time), log(wbc))) +
```

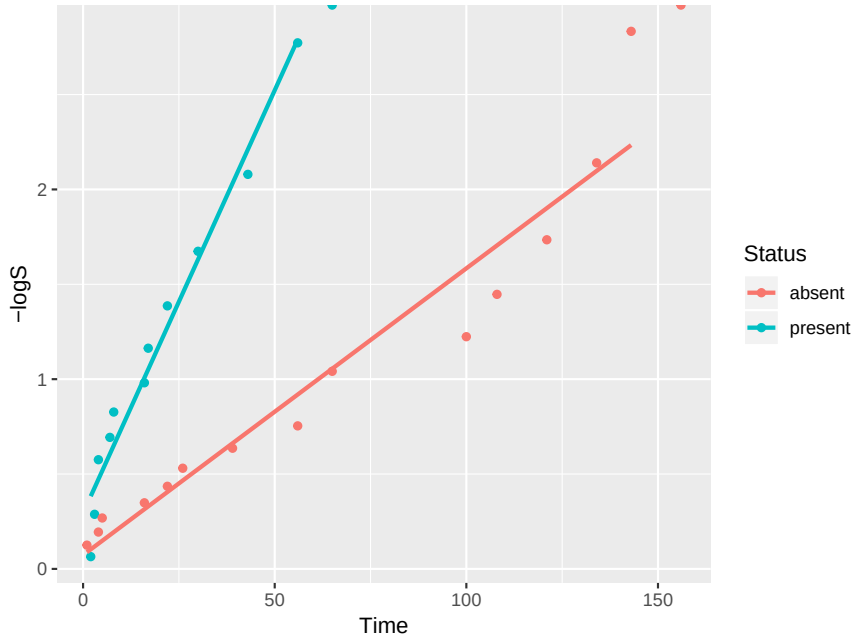
```
              geom_point(),
  vp=viewport(layout.pos.row=2, layout.pos.col=1))
```



The relationships do not appear to be linear, and we fix this by using a log transform on time and wbc.

What model might be appropriate for this data set? For the exponential model we have $-\log S(t) = H(t) = \lambda t$, so if we plot $-\log S(t)$ vs t we should see a linear relationship:

```
fit <- survfit(Surv(time, rep(1, 33)) ~ ag, data = leukemia)
df <- data.frame(Time=fit$time,
                 y=-log(fit$surv),
                 Status=rep(c("present", "absent"), c(12, 15)) )
ggplot(data=df, aes(Time, y, color=Status)) +
  geom_point() +
  geom_smooth(method="lm", se=FALSE) +
  labs(y= "-logS")
```



we fit this model via glm:

```
options(contrasts = c("contr.treatment", "contr.poly"))
fit <- glm(time ~ ag + log(wbc), family = Gamma(log),
           data=leukemia)
summary(fit, dispersion = 1)
```

```
##
## Call:
## glm(formula = time ~ ag + log(wbc), family = Gamma(log), data = leukemia)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1746  -1.2663  -0.4251   0.4962   1.9048
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   5.8154     1.2932   4.497 6.89e-06
## agpresent     1.0176     0.3492   2.914  0.00357
## log(wbc)     -0.3044     0.1319  -2.308  0.02097
##
## (Dispersion parameter for Gamma family taken to be 1)
##
##     Null deviance: 58.138  on 32  degrees of freedom
## Residual deviance: 40.319  on 30  degrees of freedom
## AIC: 301.49
##
## Number of Fisher Scoring iterations: 8
```

Cox Proportional Hazard Model

A nonparametric approach to survival data was first introduced by Cox in 1972. Here we model the hazard function h as follows: there is a baseline hazard $h_0(t)$ which is modified by covariates, so the hazard function for any individual case is

$$h(t) = h_0(t)\exp(\beta^T x)$$

and the interest is mainly in β.

The vector β is estimated via partial likelihood.

Suppose a death occurs at time t_j. Then conditionally on this event the probability that case i died is

$$\frac{h_0(t)\exp(\beta^T x_i)}{\sum_j I[T_j \geq t] h_0(t)\exp(\beta^T x_i)} = \frac{\exp(\beta^T x_i)}{\sum_j I[T_j \geq t]\exp(\beta^T x_i)}$$

which does not depend on the baseline hazard.

Example: Leukemia

```
fit <- coxph(Surv(time) ~ ag + log(wbc), data=leukemia)
summary(fit)
```

```
## Call:
## coxph(formula = Surv(time) ~ ag + log(wbc), data = leukemia)
##
##   n= 33, number of events= 33
##
##              coef exp(coef) se(coef)      z Pr(>|z|)
## agpresent -1.0691    0.3433   0.4293 -2.490  0.01276
## log(wbc)   0.3677    1.4444   0.1360  2.703  0.00687
##
##           exp(coef) exp(-coef) lower .95 upper .95
## agpresent    0.3433     2.9126     0.148    0.7964
## log(wbc)     1.4444     0.6923     1.106    1.8857
##
## Concordance= 0.726  (se = 0.065 )
## Rsquare= 0.377   (max possible= 0.994 )
## Likelihood ratio test= 15.64  on 2 df,   p=4e-04
## Wald test            = 15.06  on 2 df,   p=5e-04
## Score (logrank) test = 16.49  on 2 df,   p=3e-04
```

Example: Lung Cancer

Survival in patients with lung cancer at Mayo Clinic. Performance scores rate how well the patient can perform usual daily activities.

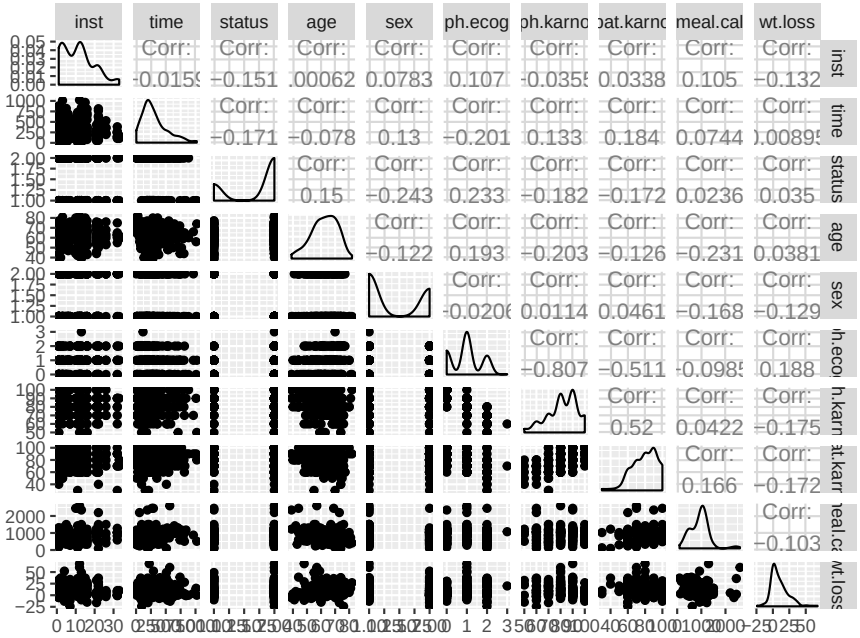
Variables:
inst: Institution code time: Survival time in days status: censoring status 1=censored, 2=dead
age: Age in years
sex: Male=1 Female=2
ph.ecog: ECOG performance score (0=good 5=dead) ph.karno: Karnofsky performance score (bad=0-good=100) rated by physician
pat.karno: Karnofsky performance score rated by patient
meal.cal: Calories consumed at meals
wt.loss: Weight loss in last six months

Source: Terry Therneau

```
kable.nice(head(lung.cancer))
```

inst	time	status	age	sex	ph.ecog	ph.karno	pat.karno	meal.cal	wt.loss
3	306	2	74	1	1	90	100	1175	NA
3	455	2	68	1	0	90	90	1225	15
3	1010	1	56	1	0	90	90	NA	15
5	210	2	57	1	1	90	60	1150	11
1	883	2	60	1	0	100	90	NA	0
12	1022	1	74	1	1	50	80	513	0

```
library(GGally)
ggpairs(lung.cancer)
```



Next we want to fit the Cox proportional hazards model, but there are two issues we need to

deal with:

- what to do with the missing values? we remove them from the data set using na.action=na.omit
- in this experiment sex was a stratification variable. We can include this fact in our model by using strata(sex) instead of just sex. This allows for non proportional hazards.

```
fit <- coxph(Surv(time, status) ~ strata(sex) + age +
       ph.ecog + ph.karno + pat.karno + meal.cal + wt.loss,
                 na.action = na.omit,
          data=lung.cancer)
summary(fit)
```

```
## Call:
## coxph(formula = Surv(time, status) ~ strata(sex) + age + ph.ecog +
##     ph.karno + pat.karno + meal.cal + wt.loss, data = lung.cancer,
##     na.action = na.omit)
##
##   n= 168, number of events= 121
##    (60 observations deleted due to missingness)
##
##                 coef  exp(coef)   se(coef)      z Pr(>|z|)
## age         9.054e-03  1.009e+00  1.160e-02  0.780   0.4352
## ph.ecog     7.073e-01  2.029e+00  2.228e-01  3.175   0.0015
## ph.karno    2.072e-02  1.021e+00  1.128e-02  1.836   0.0663
## pat.karno -1.330e-02  9.868e-01  8.050e-03 -1.652   0.0985
## meal.cal  -5.268e-06  1.000e+00  2.629e-04 -0.020   0.9840
## wt.loss   -1.520e-02  9.849e-01  7.890e-03 -1.927   0.0540
##
##           exp(coef) exp(-coef) lower .95 upper .95
## age          1.0091     0.9910    0.9864     1.032
## ph.ecog      2.0285     0.4930    1.3108     3.139
## ph.karno     1.0209     0.9795    0.9986     1.044
## pat.karno    0.9868     1.0134    0.9713     1.002
## meal.cal     1.0000     1.0000    0.9995     1.001
## wt.loss      0.9849     1.0153    0.9698     1.000
##
## Concordance= 0.638  (se = 0.041 )
## Rsquare= 0.121   (max possible= 0.994 )
## Likelihood ratio test= 21.58  on 6 df,   p=0.001
## Wald test            = 21.9  on 6 df,   p=0.001
## Score (logrank) test = 22.49  on 6 df,   p=0.001
```

Notice the three hypothesis tests at the bottom. They do the job of the F-test in regression, namely testing whether all the variables together are useful for predicting time. Hear clearly they are.

Checking the individual covariates we see that age and meal.cal are not significant. Let's

remove them

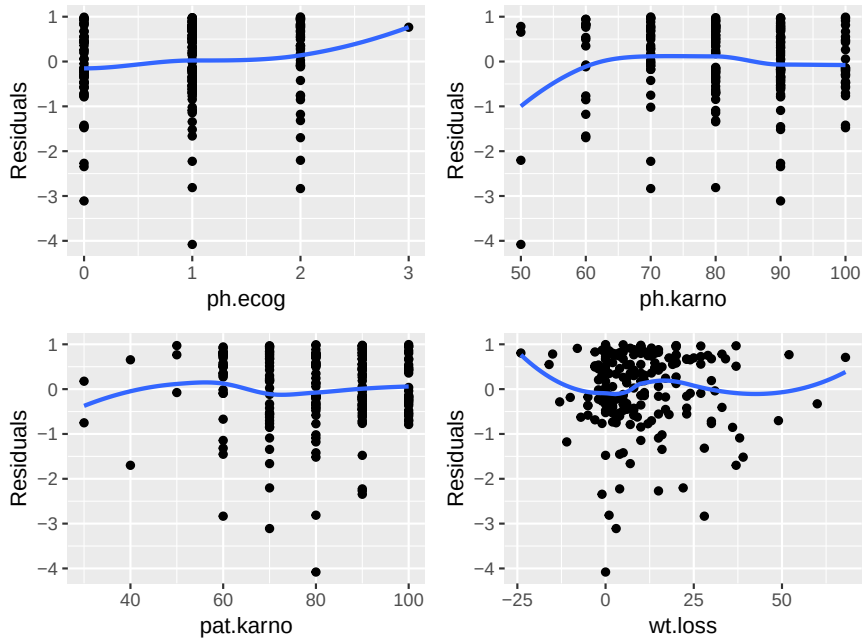
```
fit <- coxph(Surv(time, status) ~ strata(sex) + ph.ecog +
             ph.karno + pat.karno + wt.loss,
             na.action = na.omit, data=lung.cancer)
summary(fit)
```

```
## Call:
## coxph(formula = Surv(time, status) ~ strata(sex) + ph.ecog +
##     ph.karno + pat.karno + wt.loss, data = lung.cancer, na.action = na.omit)
##
##   n= 210, number of events= 148
##    (18 observations deleted due to missingness)
##
##                coef exp(coef)  se(coef)      z Pr(>|z|)
## ph.ecog    0.649467  1.914520  0.200699  3.236  0.00121
## ph.karno   0.017304  1.017454  0.010314  1.678  0.09342
## pat.karno -0.016679  0.983459  0.007255 -2.299  0.02151
## wt.loss   -0.013729  0.986365  0.006906 -1.988  0.04681
##
##           exp(coef) exp(-coef) lower .95 upper .95
## ph.ecog      1.9145     0.5223    1.2919    2.8372
## ph.karno     1.0175     0.9828    0.9971    1.0382
## pat.karno    0.9835     1.0168    0.9696    0.9975
## wt.loss      0.9864     1.0138    0.9731    0.9998
##
## Concordance= 0.633  (se = 0.038 )
## Rsquare= 0.115   (max possible= 0.995 )
## Likelihood ratio test= 25.71  on 4 df,   p=4e-05
## Wald test            = 26.57  on 4 df,   p=2e-05
## Score (logrank) test = 27.16  on 4 df,   p=2e-05
```

Next we want to try and assess whether this model is appropriate. In regression we use the residual vs. fits plot for this purpose. Here we have something similar. There are actually several kinds of residuals in a survival analysis, we will use what are called the martingale residuals. The plots are the residuals with the variable of interest vs. the residuals without the variable of interest. Again we need to deal with the missing values, and we remove them from the data set. Finally we add a loess curve to the graphs. All of this is done in

```
lung1 <- na.omit(lung.cancer[, -c(1, 4, 9)])
fit <- coxph(Surv(time, status) ~ strata(sex) + ph.karno +
             pat.karno + wt.loss, data=lung1)
df <- data.frame(ph.ecog=lung1$ph.ecog,
                 pat.karno=lung1$pat.karno,
                 ph.karno=lung1$ph.karno,
                 wt.loss=lung1$wt.loss,
                 Residuals=resid(fit))
```

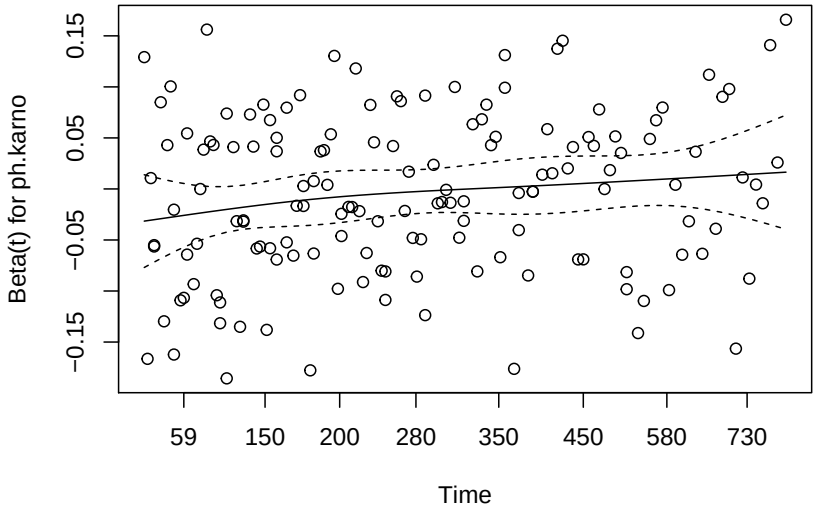
```
pushViewport(viewport(layout = grid.layout(2, 2)))
print(ggplot(data=df, aes(ph.ecog, Residuals)) +
        geom_point() + geom_smooth(se=FALSE),
  vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(ggplot(data=df, aes(ph.karno, Residuals)) +
        geom_point() + geom_smooth(se=FALSE),
  vp=viewport(layout.pos.row=1, layout.pos.col=2))
print(ggplot(data=df, aes(pat.karno, Residuals)) +
        geom_point() + geom_smooth(se=FALSE),
  vp=viewport(layout.pos.row=2, layout.pos.col=1))
print(ggplot(data=df, aes(wt.loss, Residuals)) +
        geom_point() + geom_smooth(se=FALSE),
  vp=viewport(layout.pos.row=2, layout.pos.col=2))
```

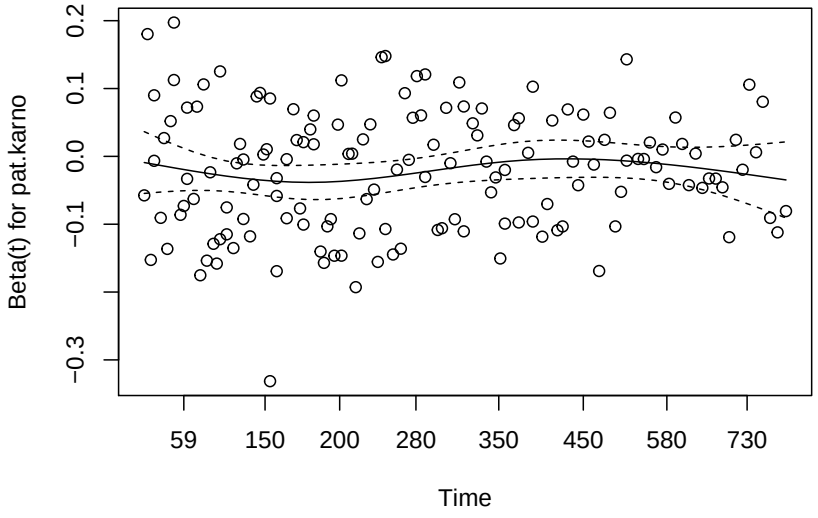


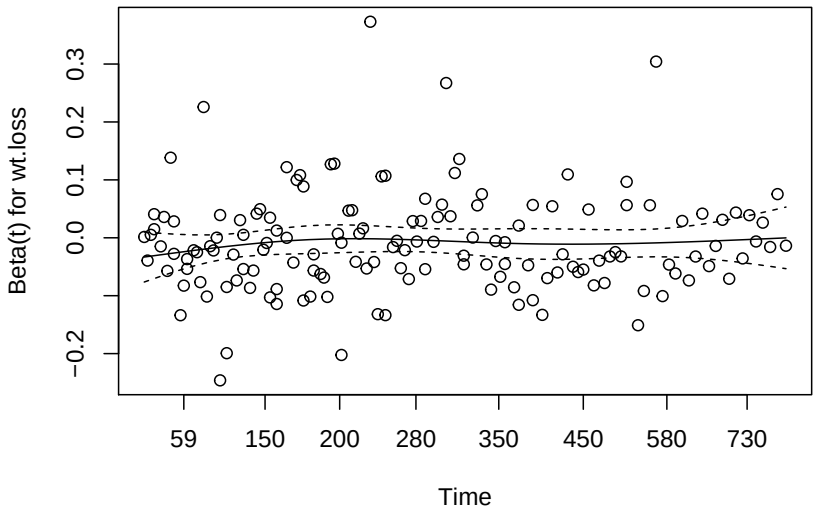
All of the relationships look reasonably linear.

Finally we can have a look whether the assumption of proportional hazards is justified. For this we use the cox.zph function. This plots the rescaled Shoenfeld residuals. A flat appearance indicates that the assumption is ok. The corresponding object carries out hypothesis tests for a significant slope in the scatterplots, which support our assessment.

```
fit.zph <- cox.zph(fit)
plot(fit.zph)
```


Beta(t) for ph.karno
0.15
0.05
−0.05
−0.15
59
150
200
280
350
450
580
730
Time


Beta(t) for pat.karno
0.2
0.1
0.0
−0.1
−0.3
59
150
200
280
350
450
580
730
Time




```
print(fit.zph)
```

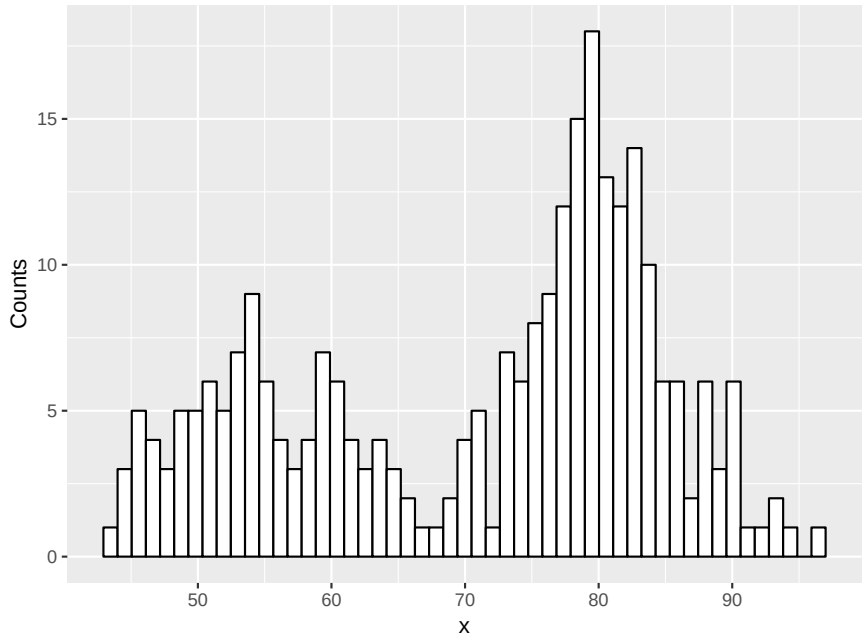
```
##              rho chisq      p
## ph.karno  0.1573 3.116 0.0775
## pat.karno 0.0540 0.472 0.4921
## wt.loss   0.0453 0.341 0.5591
## GLOBAL        NA 5.923 0.1154
```

Nonparametric Density Estimation

Example: Old Faithful Guyser

Let's have a look at the waiting times:

```
bw <- diff(range(faithful$Waiting.Time))/50
ggplot(faithful, aes(Waiting.Time)) +
geom_histogram(color = "black",
    fill = "white",
    binwidth = bw) +
    labs(x = "x", y = "Counts")
```



clearly the distribution of this data is non standard. We previously fit a mixture of normal densities to the data. In this section we will consider several non parametric methods. The basic problem is to estimate the density $f(x)$ by $\hat{f}(x)$ for each x.

Kernel Density Estimation.

The most commonly used method is called *kernel density estimation*. The idea is to average over the data points, giving greater weight to points near x. In general we have

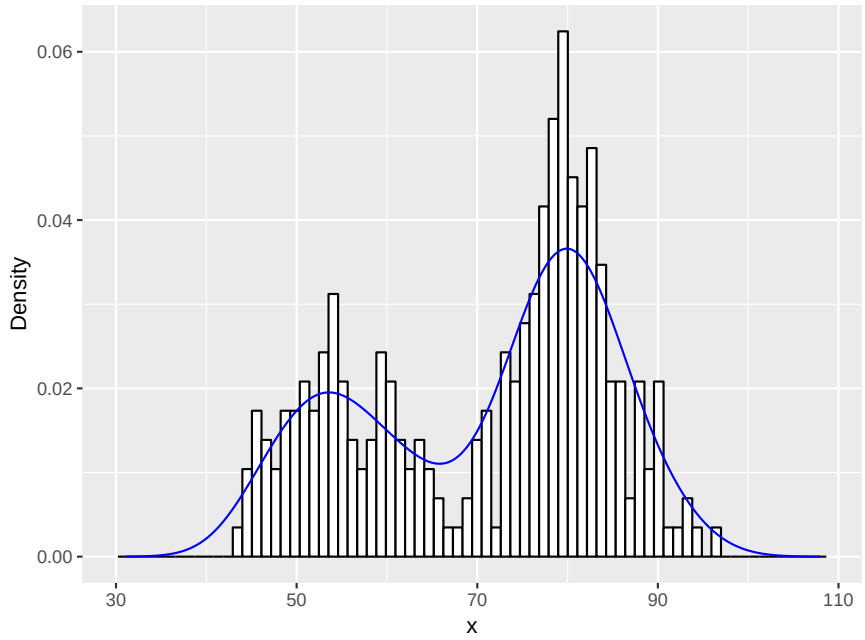
$$\hat{f}(x) = \frac{1}{n}\sum K(\frac{x - X_i}{h})$$

here K is kernel function and h is a tuning parameter.

This is implemented in

```
fit <- density(faithful$Waiting.Time)
```

```
ggplot(faithful, aes(Waiting.Time)) +
  geom_histogram(aes(y = ..density..),
    color = "black",
    fill = "white",
    binwidth = bw) +
    labs(x = "x", y = "Density") +
  geom_line(data=data.frame(x=fit$x, y=fit$y),
            aes(x, y), color="blue")
```



this uses a normal density as the kernel function. It has several formulas implemented for the choice of h. The default uses

```
bw.nrd0(faithful$Waiting.Time)
```

```
## [1] 3.987559
```

but it is often recommended to change that to

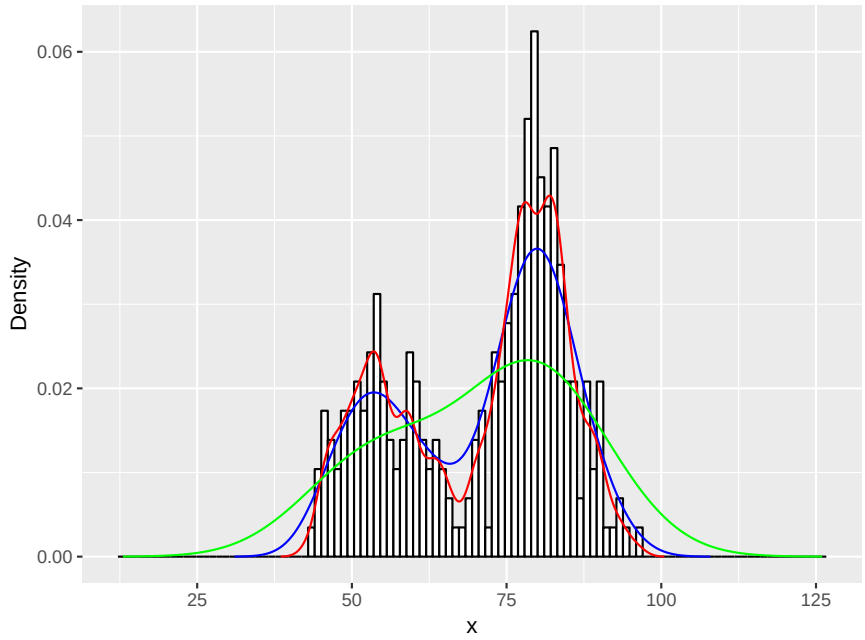
```
bw.SJ(faithful$Waiting.Time)
```

```
## [1] 2.504371
```

a method by Sheather and Jones

here are several:

```
fit1 <- density(faithful$Waiting.Time, bw=1.5)
fit2 <- density(faithful$Waiting.Time, bw=10)
ggplot(faithful, aes(Waiting.Time)) +
geom_histogram(aes(y = ..density..),
    color = "black",
    fill = "white",
    binwidth = bw) +
    labs(x = "x", y = "Density") +
  geom_line(data=data.frame(x=fit$x, y=fit$y),
            aes(x, y), color="blue") +
  geom_line(data=data.frame(x=fit1$x, y=fit1$y),
            aes(x, y), color="red")  +
  geom_line(data=data.frame(x=fit2$x, y=fit2$y),
            aes(x, y), color="green")
```



and it is clear that for small values of h we get a very ragged (under-smoothed) estimate whereas for an h to large it is the other way around.

There is a long list of density estimation methods and associated routines. For a review see Density estimation in R. Two libraries worth knowing about are

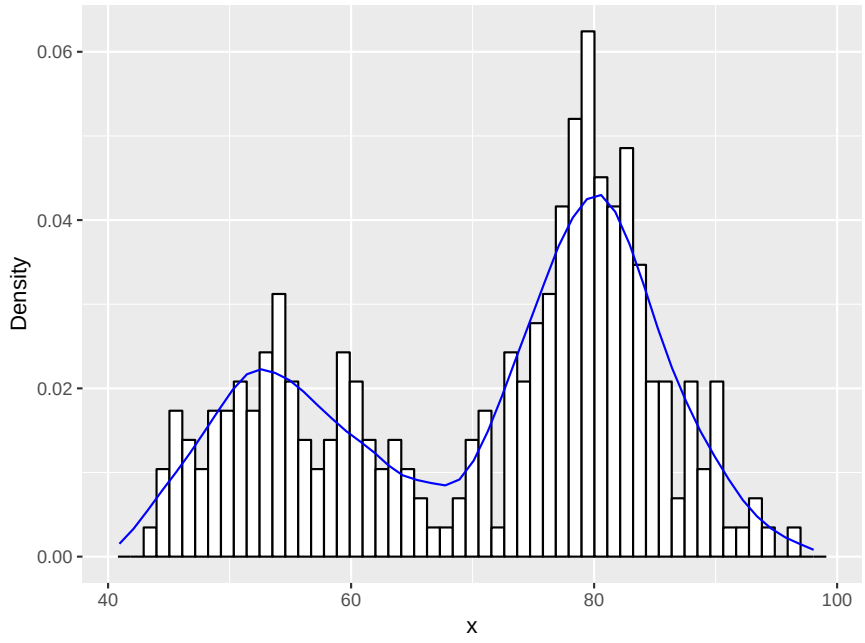
ASH

this stands for *averaged and shifted histograms*, an idea due to Scott (1992).

```
library(ash)
fit <- ash1(bin1(faithful$Waiting.Time))
```

```
## [1] "ash estimate nonzero outside interval ab"
```

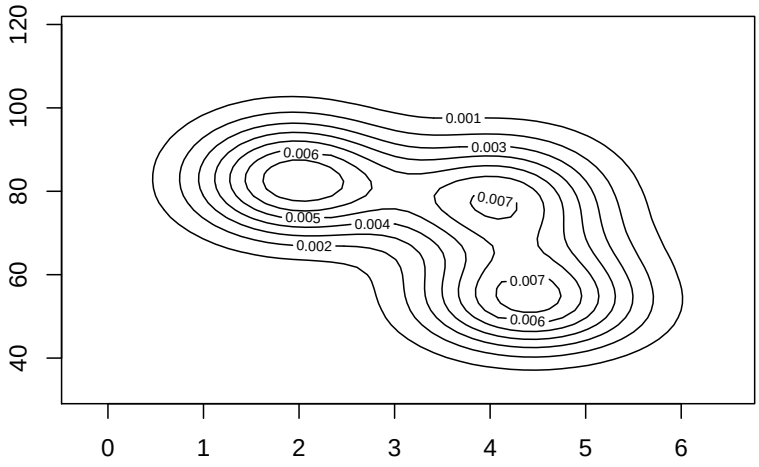
```
ggplot(faithful, aes(Waiting.Time)) +
  geom_histogram(aes(y = ..density..),
    color = "black",
    fill = "white",
    binwidth = bw) +
    labs(x = "x", y = "Density") +
  geom_line(data=data.frame(x=fit$x, y=fit$y),
            aes(x, y), color="blue")
```



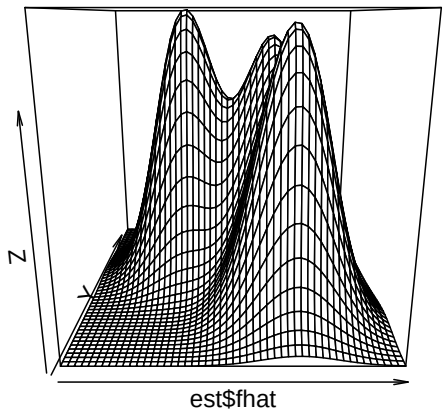
KernSmooth

a method due to Wand & Jones (1995). It's main advantage is that it also works for higher dimensional data.

```
library(KernSmooth)
data(geyser, package="MASS")
x <- cbind(geyser$duration, geyser$waiting)
est <- bkde2D(x, bandwidth=c(0.7, 7))
contour(est$x1, est$x2, est$fhat)
```




```
persp(est$fhat)
```

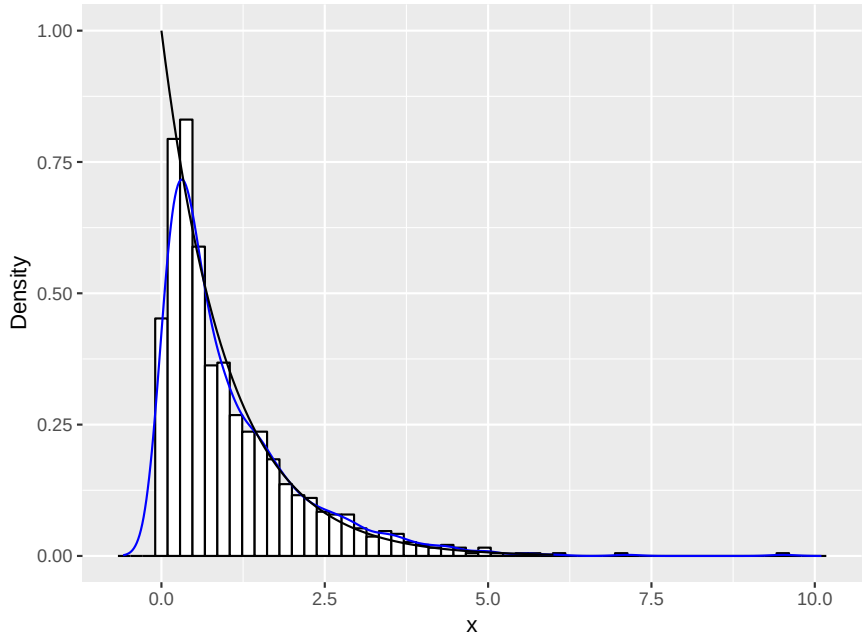


Boundary Problem

Most methods encounter problems when the true density does not go to 0 at the edges:

```
df <- data.frame(x=rexp(1000, 1))
fit <- density(df$x)
bw <- diff(range(df$x))/50
```

```
df1 <- data.frame(x=seq(0, 6, length=100),
                  y=exp(-seq(0, 6, length=100)))
ggplot(df, aes(x)) +
  geom_histogram(aes(y = ..density..),
    color = "black",
    fill = "white",
    binwidth = bw) +
    labs(x = "x", y = "Density") +
  geom_line(data=data.frame(x=fit$x, y=fit$y),
            aes(x, y), color="blue") +
  geom_line(data=df1,
            aes(x, y), color="black")
```

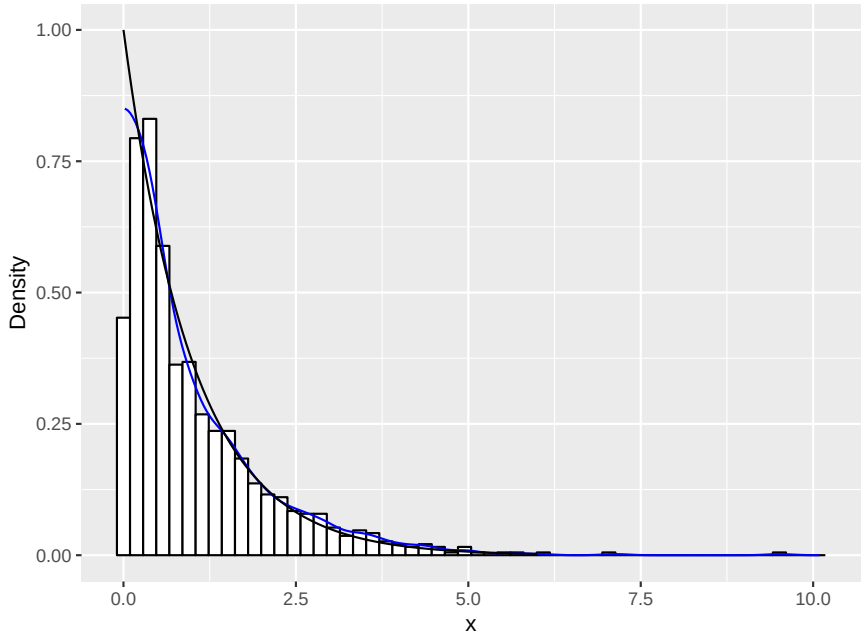


the reason is clear: The formula expects data on both sides of x, but in this case at x=0 there is data only on one side.

A simple solution is to "mirror" the data:

```
fit <- density(c(df$x, -df$x))
fit$y <- 2*fit$y[fit$x>0]
fit$x <- fit$x[fit$x>0]
bw <- diff(range(df$x))/50
ggplot(df, aes(x)) +
  geom_histogram(aes(y = ..density..),
    color = "black",
    fill = "white",
    binwidth = bw) +
    labs(x = "x", y = "Density") +
  geom_line(data=data.frame(x=fit$x, y=fit$y),
```

```
            aes(x, y), color="blue") +
  geom_line(data=df1,
            aes(x, y), color="black")
```



notice the factor of 2, which is needed to scale the density to integrate to 1. In general this rescaling can be difficult.

This would work perfectly fine for a density that has slope 0 at the boundary points, for example a uniform. It still is wrong in our case, because our density has slope -1.

Other more complicated solutions are known but none of them is very satisfactory.

Smooth Bootstrap

There is a version of the Bootstrap that can be helpful at times. The idea is to sample from a non-parametric density estimate instead of the empirical distribution function. For this however we would need to discuss how to simulate from a general distribution function. Come to ESMA 5015 Simulation next time!

Example: Hidalgo stamps

A well known data set in statistics has the thicknesses (espesor) in millimeters of 485 Mexican stamps (sello) printed in 1872-1874, from the 1872 Hidalgo issue.

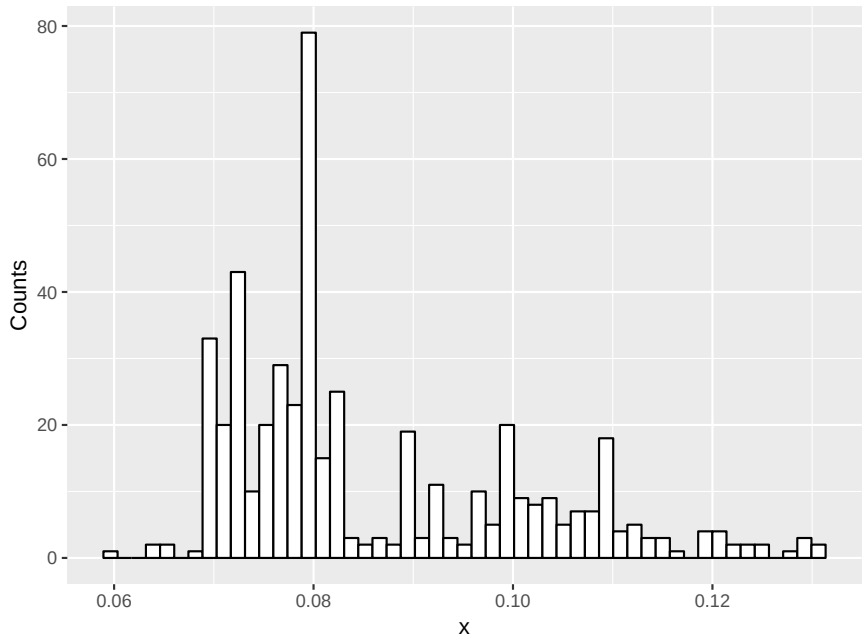
It is thought that the stamps from this issue are a "mixture" of different types of paper, of different thicknesses. Can we determine from the data how many different types of paper were used?

```
kable.nice(matrix(stamps[1:50], nrow=10))
```

0.060	0.069	0.07	0.070	0.071
0.064	0.069	0.07	0.070	0.071
0.064	0.069	0.07	0.070	0.071
0.065	0.070	0.07	0.070	0.071
0.066	0.070	0.07	0.070	0.071
0.068	0.070	0.07	0.070	0.071
0.069	0.070	0.07	0.070	0.071
0.069	0.070	0.07	0.070	0.071
0.069	0.070	0.07	0.070	0.071
0.069	0.070	0.07	0.071	0.071

Let's start with

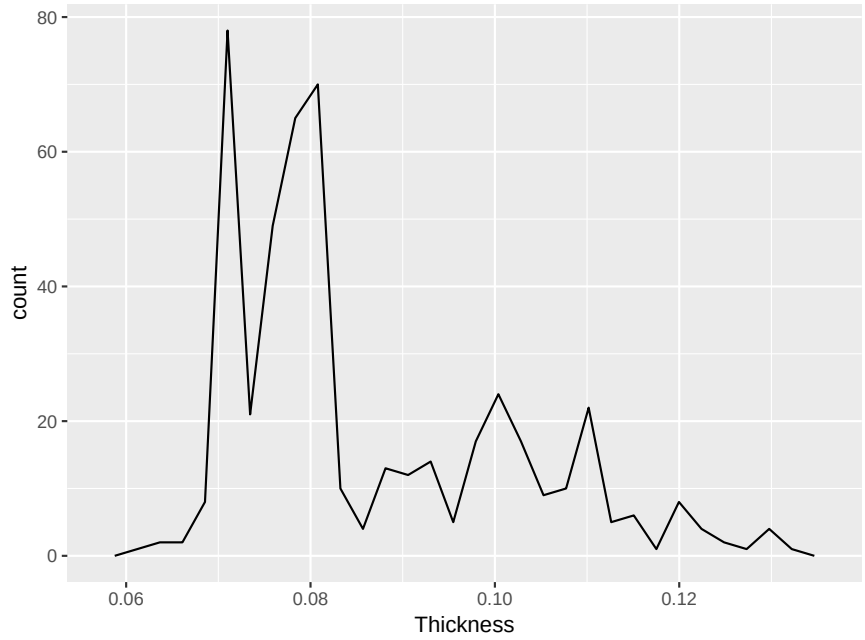
```
bw <- diff(range(stamps))/50
df <- data.frame(Thickness=stamps)
ggplot(df, aes(Thickness)) +
geom_histogram(color = "black",
    fill = "white",
    binwidth = bw) +
    labs(x = "x", y = "Counts")
```



which seems to have at least two modes. This judgement however is tricky because it depends on the number of bins we use.

An alternative is to use a frequency polygon

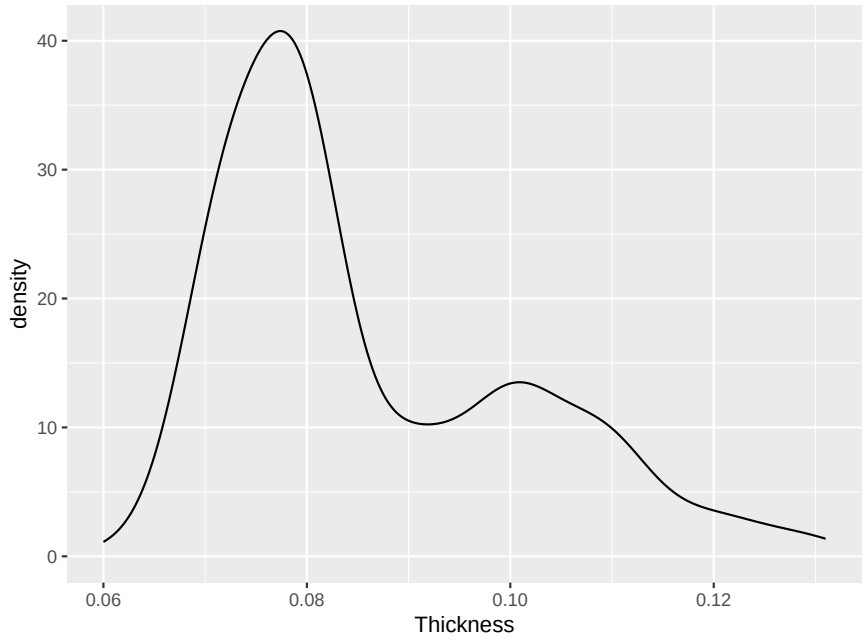
```
ggplot(df, aes(Thickness)) +
  geom_freqpoly()
```



which seems to suggest a much larger number of modes.

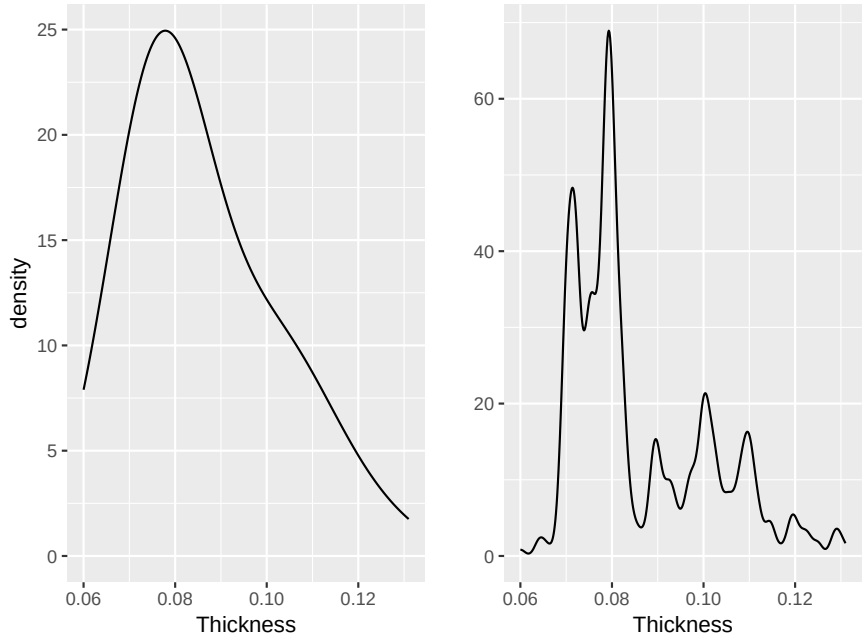
Let's instead draw the graph using a nonparametric density estimate:

```
ggplot(df, aes(Thickness)) +
  stat_density(geom="line")
```



here it seems again like there are two modes, but this depends largely on the chosen bandwidth:

```
pushViewport(viewport(layout = grid.layout(1, 2)))
print(ggplot(df, aes(Thickness)) +
   stat_density(geom="line", bw=0.01)  ,
  vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(ggplot(df, aes(Thickness)) +
   stat_density(geom="line", bw=0.001) +ylab("")  ,
  vp=viewport(layout.pos.row=1, layout.pos.col=2))
```



stat_density implements a kernel density estimator as discussed above, with choices of different kernels and bandwidth selectors. In what follows we will need to explicitly calculate these estimates and use the density routine.

From the above it is clear that the number of modes depends on the choice of h. It is possible to show that the number of modes is a non-increasing function of h. At the extremes we would have a simple normal distribution with one mode (h large and on the other a sharply peaked mode at each observation (h tiny)

Let's say we want to test

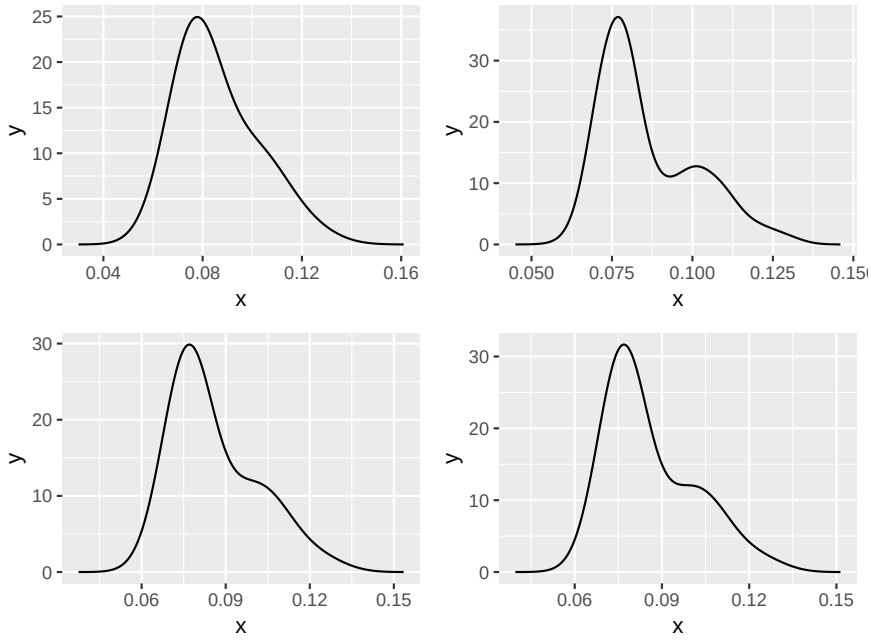
$$H_0 : \text{number of modes} = 1 \text{ vs. } H_a : \text{number of modes} > 1$$

Because the number of modes is a non-increasing function of h there exists an h_1 such that the density estimator has one mode for $h < h_1$ and two or more modes for $h > h_1$. Playing around with

```
fhat <- function(h, t, dta=stamps) {
  tmp <- density(dta, bw=h)
  df <- data.frame(x=tmp$x, y=tmp$y)
  if(missing(t)) return(df)
  out <- approx(df, xout=t)$y
```

```
  out[!is.na(out)]
}
draw.fhat <- function(h)
  ggplot(fhat(h), aes(x, y)) + geom_line()
```

```
pushViewport(viewport(layout = grid.layout(2, 2)))
print(draw.fhat(0.01) ,
  vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(draw.fhat(0.005) ,
  vp=viewport(layout.pos.row=1, layout.pos.col=2))
print(draw.fhat(0.0075) ,
  vp=viewport(layout.pos.row=2, layout.pos.col=1))
print(draw.fhat(0.0068) ,
  vp=viewport(layout.pos.row=2, layout.pos.col=2))
```



we find $h_1 \sim 0.0068$.

Is there a way to calculate the number of modes for a given h? here is one:

- calculate $y_i = \hat{f}(t_i; h)$ on a grid $t_1, ..t_k$
- calculate $z_i = y_i - 1 - y_i$ and note that at a mode z will change from positive to negative
- number of modes $= \sum I[z_i > 0 \text{ and } z_{i+1} < 0]$

Let's write a simple routine that automates the process. It uses a bisection algorithm.

```
x.points <- seq(min(stamps), max(stamps), length = 250)
calc.num.mode = function(y) {
  m <- length(y) - 1
```

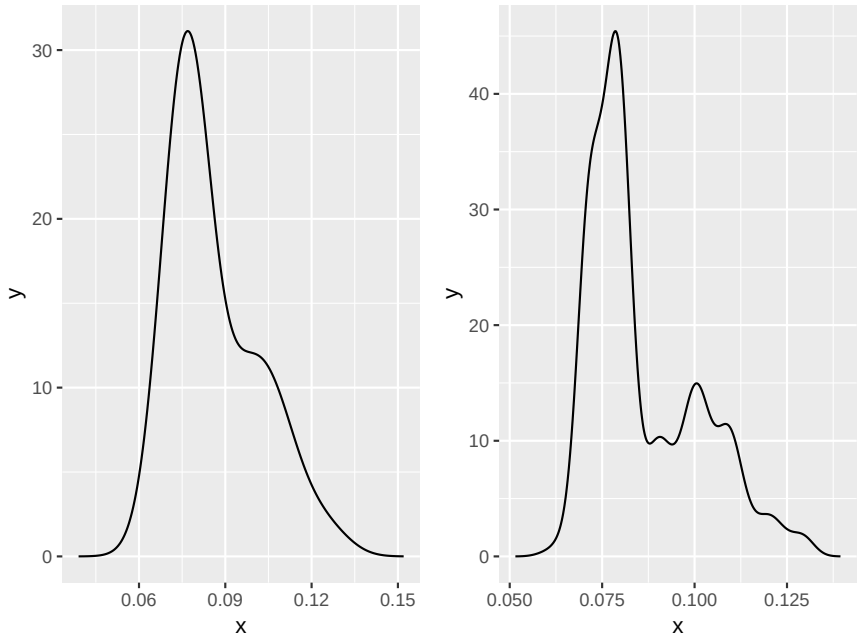
```
  z <- diff(y)
  sum(ifelse(z[-m] >= 0 & z[-1] < 0, 1, 0))
}
find.h <- function(num.modes, h=0.007, Show=FALSE) {
  repeat {
    h <- h-0.001
    if(Show)
      cat("h =", h, " modes=",
          calc.num.mode(fhat(h, x.points)), "\n")
    if(calc.num.mode(fhat(h, x.points)) >= num.modes)                break
  }
  low <- h
  high <- h + 0.001
  repeat {
    h <- (low+high)/2
    if(Show)
      cat("h =", h, " modes=",
          calc.num.mode(fhat(h, x.points)), "\n")
    if(calc.num.mode(fhat(h, x.points)) < num.modes)
      high <- h
    else
      low <- h
    if(high-low<10^-7)
      break
  }
  h
}
```

```
h1 <- find.h(1, Show = TRUE)
```

```
## h = 0.006  modes= 2
## h = 0.0065  modes= 2
## h = 0.00675  modes= 1
## h = 0.006875  modes= 1
## h = 0.0069375  modes= 1
## h = 0.00696875  modes= 1
## h = 0.006984375  modes= 1
## h = 0.006992188  modes= 1
## h = 0.006996094  modes= 1
## h = 0.006998047  modes= 1
## h = 0.006999023  modes= 1
## h = 0.006999512  modes= 1
## h = 0.006999756  modes= 1
## h = 0.006999878  modes= 1
## h = 0.006999939  modes= 1
```

```
h5 <- find.h(5)
```

```
pushViewport(viewport(layout = grid.layout(1, 2)))
print(draw.fhat(h1) ,
  vp=viewport(layout.pos.row=1, layout.pos.col=1))
print(draw.fhat(h5) ,
  vp=viewport(layout.pos.row=1, layout.pos.col=2))
```



So, how we can test

$$H_0 : \text{number of modes} = 1 \text{ vs. } H_a : \text{number of modes} > 1$$

Here it is:

- draw B bootstrap samples of size n from fhat(h1)
- for each find h_1^*, the smallest h for which this bootstrap sample has just 1 mode

- approximate p-value of test is the proportion of $h_1^* > h_1$.

the idea is this; if there is indeed just one mode, then in the bootstrap samples h_1^* should be around h_1 and so this proportions shouldn't be to small.

Notice we don't actually need h_1^*, we just need to check if $h_1^* > h_1$, which is the case if $\hat{f}(x^*; h_1^*)$ has at least two modes.

Note that we are not drawing bootstrap samples from "stamps" but from a density estimate, $\hat{f}$. So this is an example of the *smooth bootstrap* mentioned above.

How do we draw from fhat? It can be shown that if $y_1^*, .., y_n^*$ is a bootstrap sample from the data, then a smooth bootstrap sample is given by

$$x_i^* = \bar{y^*} + (1 + h_1^*/s^2)^{-1/2}(y_i^* - \bar{y^*} + h_1^*\epsilon_i)$$

where $\epsilon_i \sim N(0,1)$

```
test.modes <- function(k) {
  h <- find.h(k+1)
  q <- 1/sqrt((1 + h^2/var(stamps)))
  B <- 1000
  hstar <- rep(0, B)
  for (i in 1:B) {
    ystar <- sample(stamps, size = 485, replace = TRUE)
    xstar <- mean(ystar) + q*(ystar-mean(ystar) +
                h*rnorm(485))
    y <- fhat(h, x.points, dta=xstar)
    if (calc.num.mode(y) > k)
        hstar[i] <- 1
  }
  length(hstar[hstar > h])/B
}
test.modes(1)
```

```
## [1] 0
```

and so we find strong evidence against the null, there are more than one modes.

The same method works for testing

$$H_0 : \text{number of modes} = k \text{ vs. } H_a : \text{number of modes} > k$$

and we find

```
for(k in 2:9)
  cat("k =", k, ", p =", test.modes(k),"\n")
```

```
## k = 2 , p = 0.31
## k = 3 , p = 0.037
## k = 4 , p = 0.007
## k = 5 , p = 0
## k = 6 , p = 0
## k = 7 , p = 0.344
## k = 8 , p = 0.783
## k = 9 , p = 0.549
```

So there are certainly more than one mode, with a chance for as many 7.

Time Series Analysis

A *time series* is any data collected over a period of time. Examples are

- Dow Jones Industrial index
- Population of Earth
- Number of AIDS patients at some hospital
- ...

The main feature that sets time series apart is that the assumption of independence usually fails. If I wanted to guess the size of the human population in 2019, knowing what it is in 2018 might be useful.

Example: Births in New York

data set of the number of births per month in New York city, from January 1946 to December 1959 (originally collected by Newton)

```
kable.nice(matrix(births[1:50], nrow=10))
```

26663	21672	23105	23950	21761
23598	21870	23110	23504	22874
26931	21439	21759	22238	24104
24740	21089	22073	23142	23748
25806	23709	21937	21059	23262
24364	21669	20035	21573	22907
24477	21752	23590	21548	21519
23901	20761	21672	20000	22025
23175	23479	22222	22424	22604
23227	23824	22123	20615	20894

the first step is to store the data in a time series object in R, so that we can use R's many functions for analyzing time series data. One way to do that is the base R function *ts*.

by default ts assumes equal space time units of one year. For other time spans we can use the argument *frequency*. For example, if data is by month, use frequency=12. We can also change the starting point:

```
birth.ts <- ts(births, frequency=12, start=c(1946, 1))
head(birth.ts)
```

```
## [1] 26663 23598 26931 24740 25806 24364
```

Example: Deaths in Puerto Rico

number of deaths per month in Puerto Rico, according to the official deaths statistics of the government:

Total de Defunciones por Mes

Mes	2015	2016	2017	2018
Jan	2744	2742	2894	2821
Feb	2403	2592	2315	2448
Mar	2427	2458	2494	2643
Apr	2259	2241	2392	2218
May	2340	2312	2390	1892
Jun	2145	2355	2369	0
Jul	2382	2456	2367	0
Aug	2272	2427	2321	0
Sep	2258	2367	2928	0
Oct	2393	2357	3040	0
Nov	2268	2484	2671	0
Dec	2516	2854	2820	0
Total	28407	29645	31001	12022

```
Deaths <- c(2744, 2403, 2427, 2259, 2340, 2145,
            2382, 2272, 2258, 2393, 2268, 2516,
            2742, 2592, 2458, 2241, 2312, 2355,
            2456, 2427, 2367, 2357, 2484, 2854,
            2894, 2315, 2494, 2392, 2390, 2369,
            2367, 2321, 2928, 3040, 2671, 2820,
            2821, 2448, 2643, 2218)
pr.deaths.ts <-  ts(Deaths, frequency=12,
                    start=c(2015, 1))
```

Example: Dow Jones Industrial index

Data set has the daily closing values of the Dow Jones Industrial Index from January 2000 to November 2018 and the weekly closing values from January 1985 to November 2018.

the data set is available at dow.jones.rds

```
dow.jones <- readRDS("C:\\Users\\Wolfgang\\Dropbox\\teaching\\Computational-Statistics-w
kable.nice(head(dow.jones$Weekly))
```

Date	Close
1985-01-28	1277
1985-02-04	1289
1985-02-11	1282
1985-02-18	1275
1985-02-25	1299
1985-03-04	1269

Again we want to turn this into a time series object. Here however we have the problem that the time variable is not equal spaced (sometimes the New York stock exchange is not open on a Monday or a Friday). One way to do this is to use the package *zoo*:

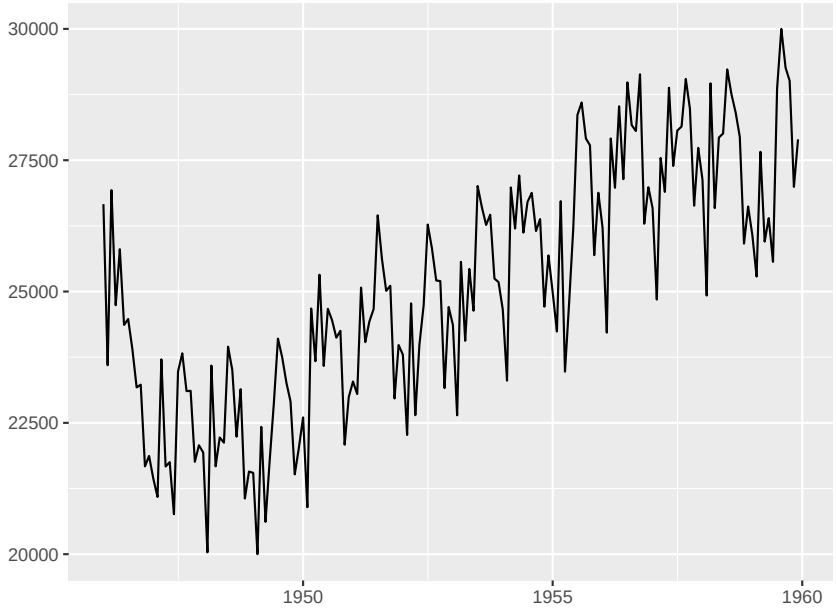
```
library(zoo)
dj.ts <- zoo(dow.jones$Weekly$Close,
     order.by=as.Date(as.character(dow.jones$Weekly$Date)))
```

Graphs

There is an obvious graph for a time series, namely a line graph by time. A ts object already has a plot function, so

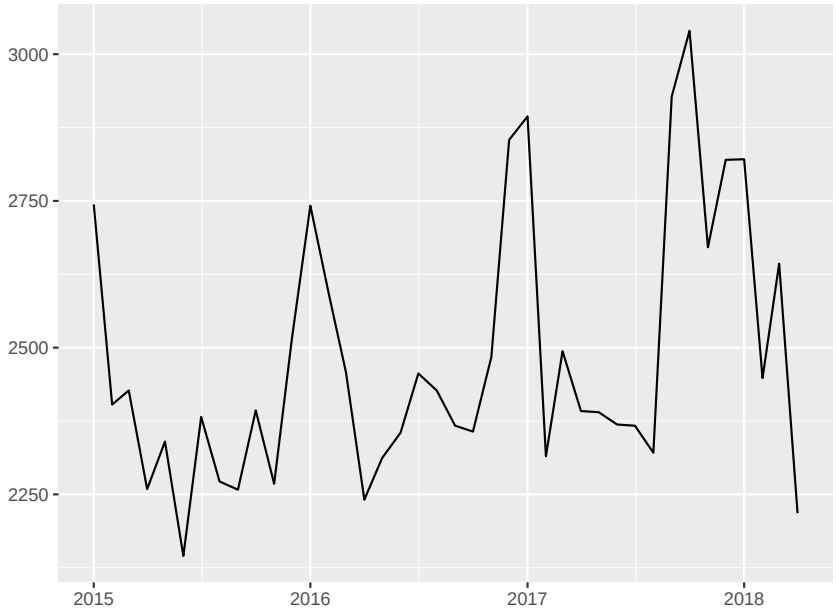
Example: Births in New York

```
library(ggfortify)
autoplot(birth.ts)
```



Example: Deaths in Puerto Rico

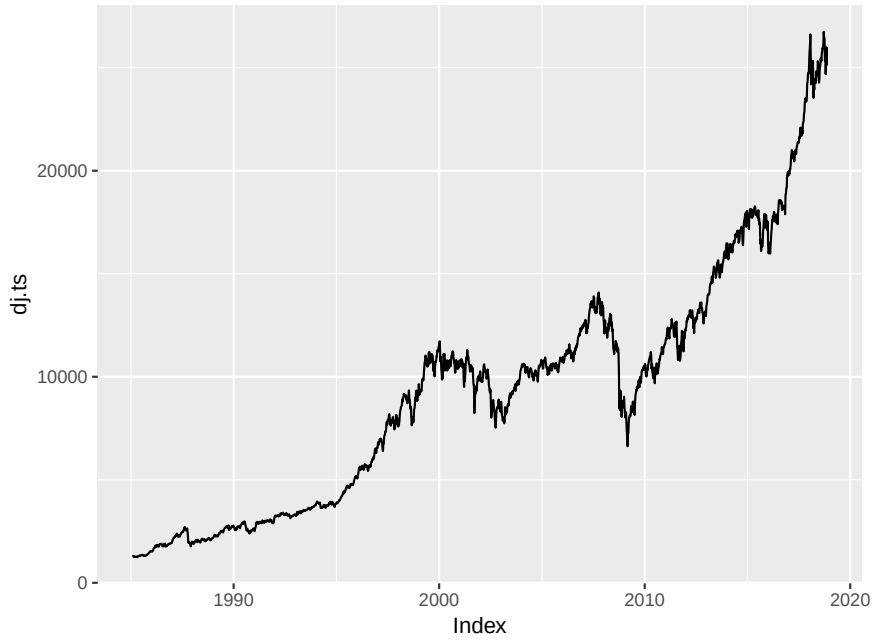
```
autoplot(pr.deaths.ts)
```



here we have a strong seasonal trend, deaths go up a lot in the winter. And of course we have a spike around September 2017!

Example: Dow Jones

```
autoplot(dj.ts)
```



Decomposition

In general a time series consists of some (or all) of these parts:

- a baseline
- a trend
- seasonal component(s)
- unusual parts
- random fluctuation

Example: Deaths in Puerto Rico

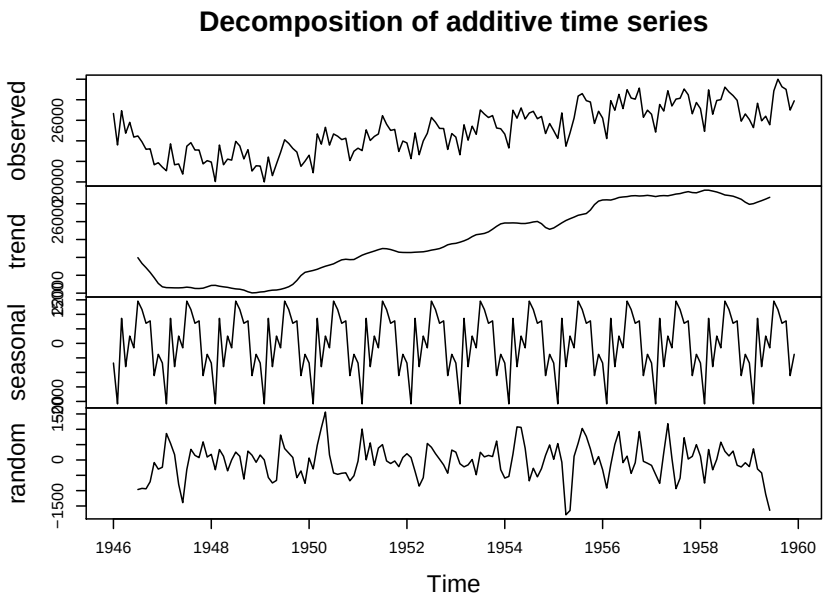
here we seem to have all of these:

- a baseline: roughly 2300 deaths per month
- a trend: from around 2250 in 2015 to 2500 in 2018
- seasonal component: winter month vs rest of year
- unusual parts: September 2017

One of the tasks in a time series analysis is to *decompose* the series into these parts.

Example: Births

```
births.dec <- decompose(birth.ts)
plot(births.dec)
```

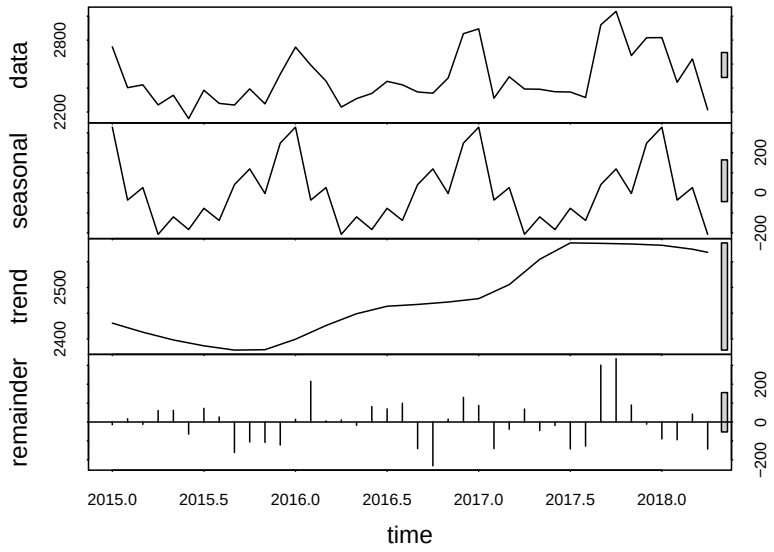


so we see a clear trend and seasonal component

Example: Deaths in PR

an alternative to decompose is the *stl* routine, which uses *loess* to do the fitting:

```
pr.deaths.dec <- stl(pr.deaths.ts, "periodic")
plot(pr.deaths.dec)
```



again a clear trend and seasonal component.

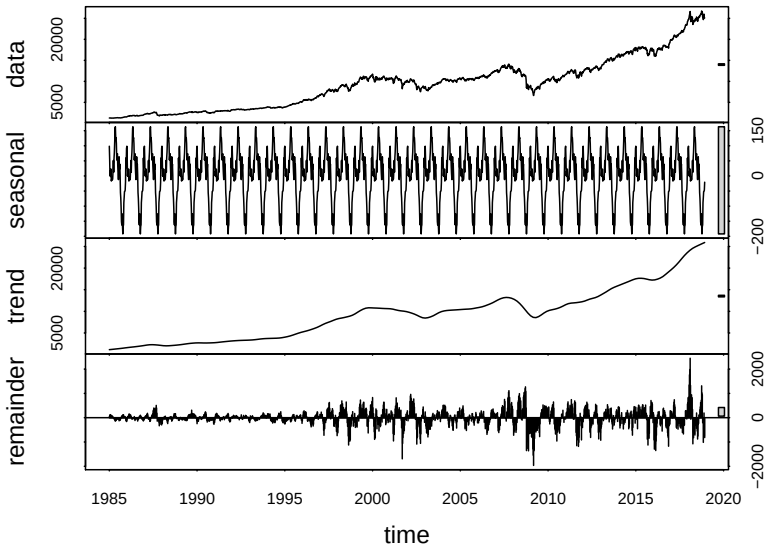
Example: Dow Jones

```
dj.dec <- stl(dj.ts, "periodic")
```

```
## Error in na.fail.default(as.ts(x)): missing values in object
```

and this gives an error. That is because this time series has irregular time points. It can be quite a chore to "fix" this problem. If the time periods are somewhat regular (in our case they are almost one week) it is easier to make a new series with a regular time scale:

```
dj.alt.ts <- ts(dow.jones$Weekly$Close, frequency = 52,
                start=c(1985, 1))
dj.alt.dec <- stl(dj.alt.ts, "periodic")
plot(dj.alt.dec)
```



which shows a clear trend but not any seasonality. We also see that the variation is increasing over time.

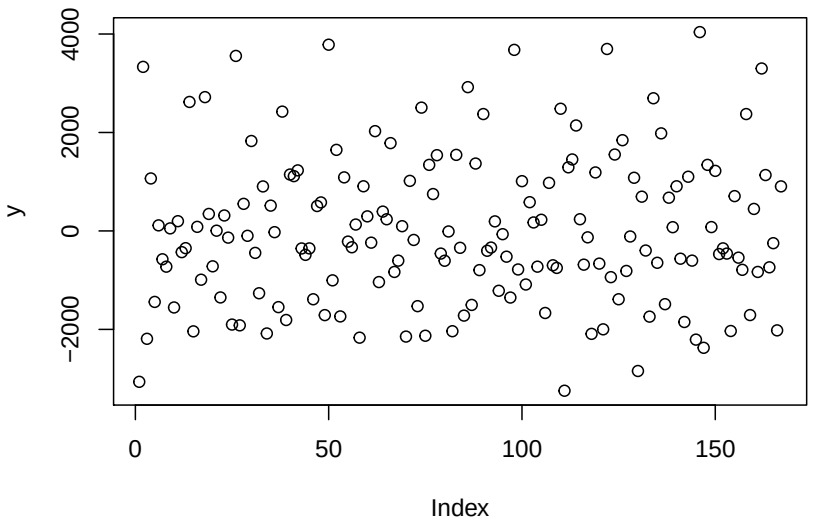
ARIMA models

As we said before, the special feature of a time series is a dependence between neighboring points. This suggests a way to analyse a time series by analyzing the correlation between time points.

Example: Births

Let's define a new variable: $Y_i = X_i - X_{i-1}$, that is the change in the number of births from one month to the next.

```
y <- diff(births)
plot(y)
```

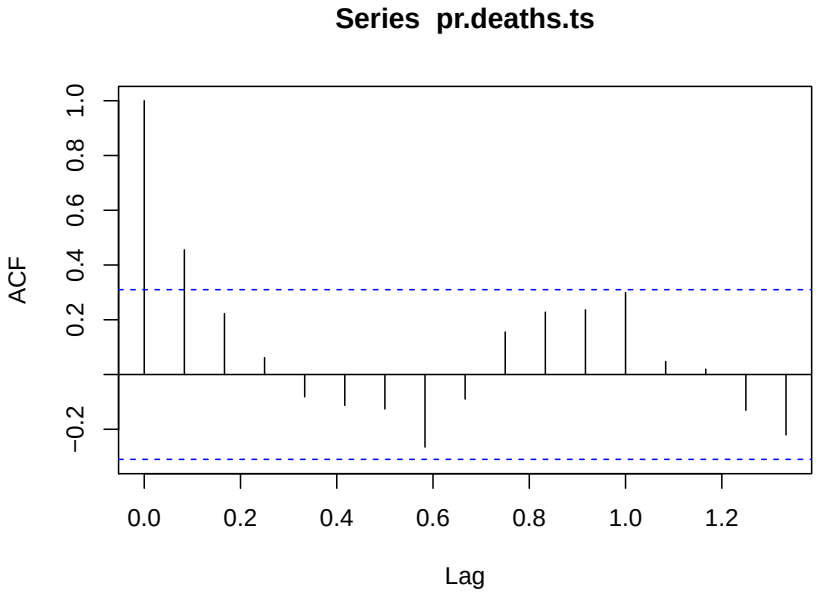


here we now longer have any pattern, which shows us that this series consisted only of a trend (which we eliminated in y).

There are some R routines to do this for us for a time series:

Example: PR Deaths

```
acf(pr.deaths.ts)
```



which shows a clear seasonal pattern.

---

the above is an example of a general model for time series called *ARIMA* (auto-regressive integrated moving average).

Here are a couple of examples:

- AR(1) = ARIMA(1, 0, 0)

$$X_i = c + \alpha_1 X_{i-1} + \epsilon_i$$

for the births data we saw that $X_i - X_{i-1}$ was essentially random noise, so it seems that series can be modeled as an AR(1).

- AR(2) = ARIMA(2, 0, 0)

$$X_i = c + \alpha_1 X_{i-1} + \alpha_2 X_{i-2} + \epsilon_i$$

- MA(1)=AR(0, 0, 1)

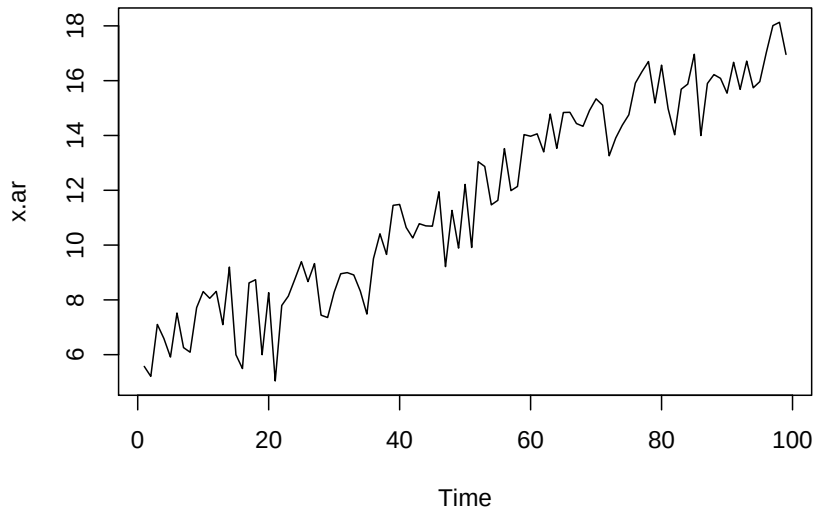
$$X_i = c + \beta_1 \epsilon_{i-1} + \epsilon_i$$

finally ARIMA combines both variations.

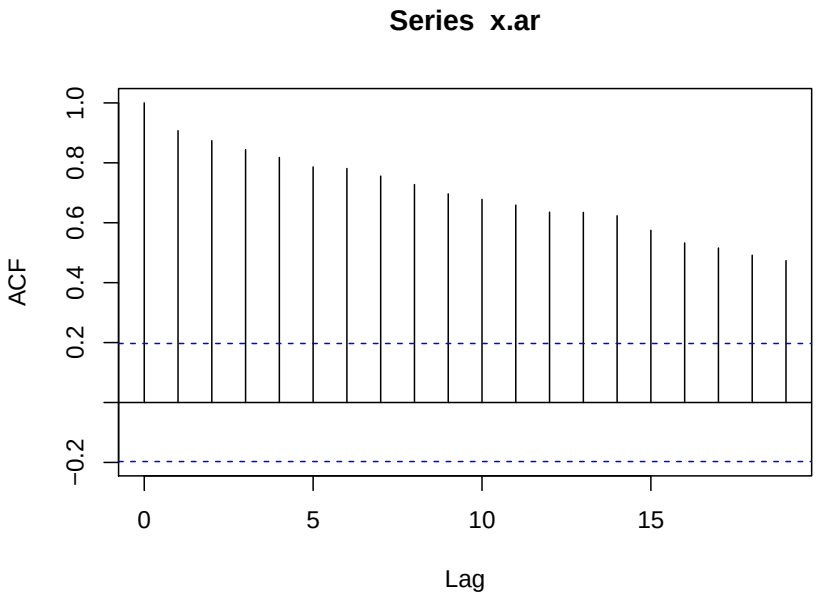
Example: Simulated data

let's create some artificial data to see what is going on:

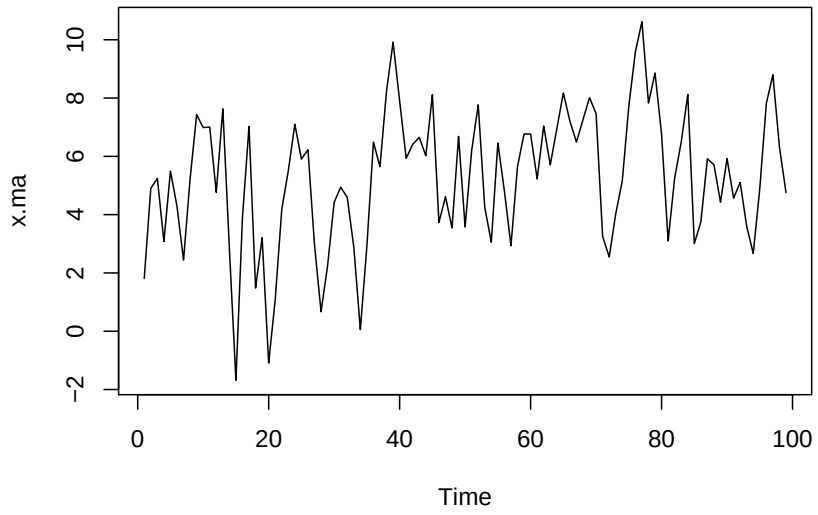
```
eps <- rnorm(100)
x <- sort(runif(100, 1, 10))
x.ar <- ts(5 +  x[-1] + 0.2* x[-100] + eps[-100])
plot(x.ar)
```



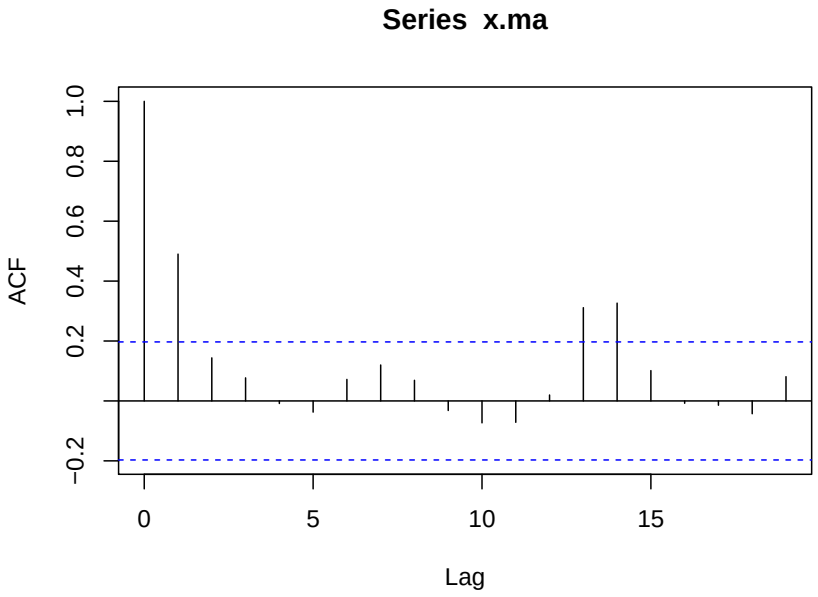

```
acf(x.ar)
```




```
x.ma <- ts(5 + 2* eps[-1] + eps[-100])
plot(x.ma)
```




```
acf(x.ma)
```



so the acf plot of x.ma shows us that indeed we have a moving average process, with a lag of 1 or 2.

Fitting an ARIMA model can be done with the *auto.arima* routine from the *forcast* package:

```
library(forecast)
auto.arima(x.ar)
```

```
## Series: x.ar
## ARIMA(0,1,1) with drift
```

```
##
## Coefficients:
##           ma1   drift
##       -0.8492  0.1159
## s.e.   0.0626  0.0168
##
## sigma^2 estimated as 1.096:  log likelihood=-143.16
## AIC=292.31   AICc=292.57   BIC=300.07
```

Spectral Analysis

The most general way to analyse a time series is a *spectral analysis*. In essence, we assume that

$$X_t = \sum_{j=1}^{k} (A_j \sin(2\pi\nu_j t) + B_j \cos(2\pi\nu_j t))$$

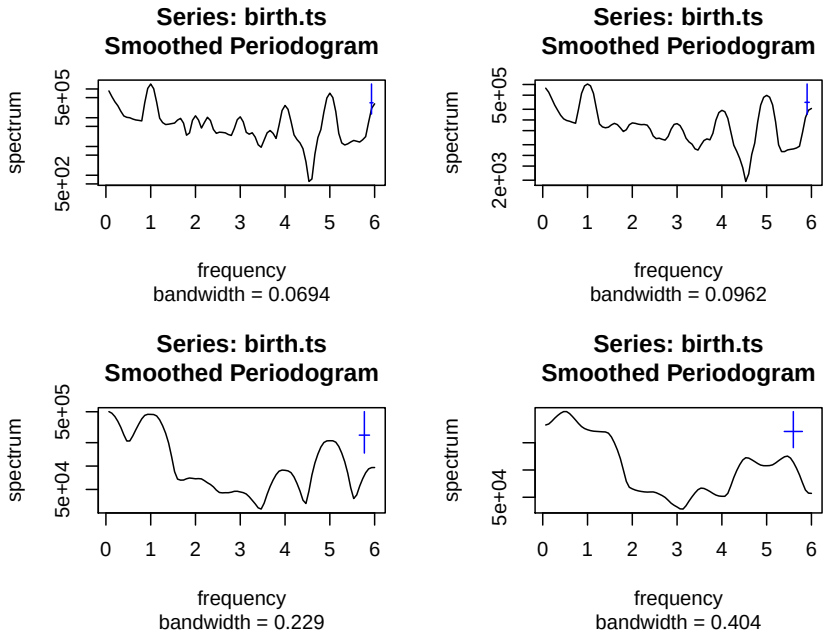
and a spectral analysis amounts to estimating the parameters A_j, B_j, ν_j.

A fuller discussion of this is beyond our course. It would require first a discussion *Fourier analysis*, one of the most widely used technics in mathematics and engineering.

The usual starting point is a look at the *periodogram*

Example: Births

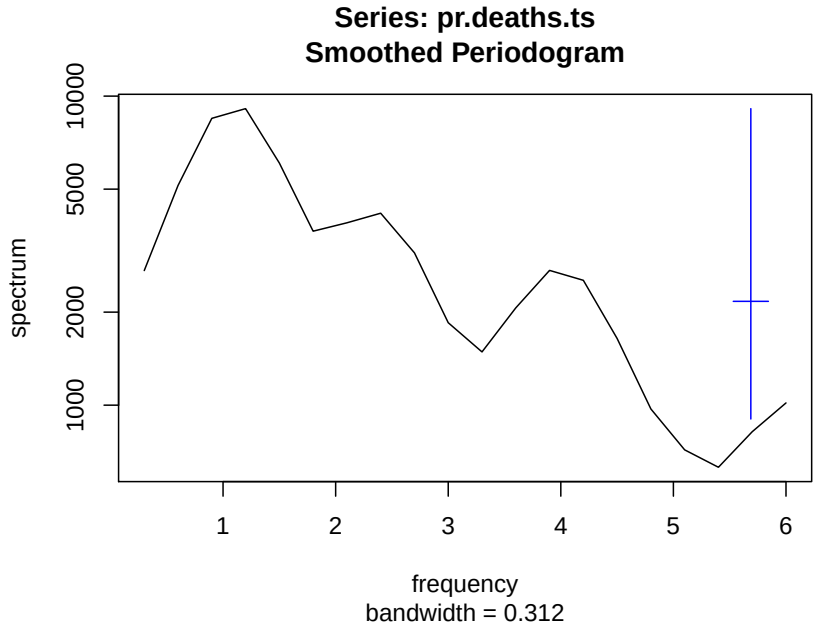
```
par(mfrow=c(2, 2))
spec.pgram(birth.ts, spans = c(3, 3))
spec.pgram(birth.ts, spans = c(3, 5))
spec.pgram(birth.ts, spans = c(7, 11))
spec.pgram(birth.ts, spans = c(7, 21))
```



and so we see that larger values of span do more smoothing and show the underlying patters

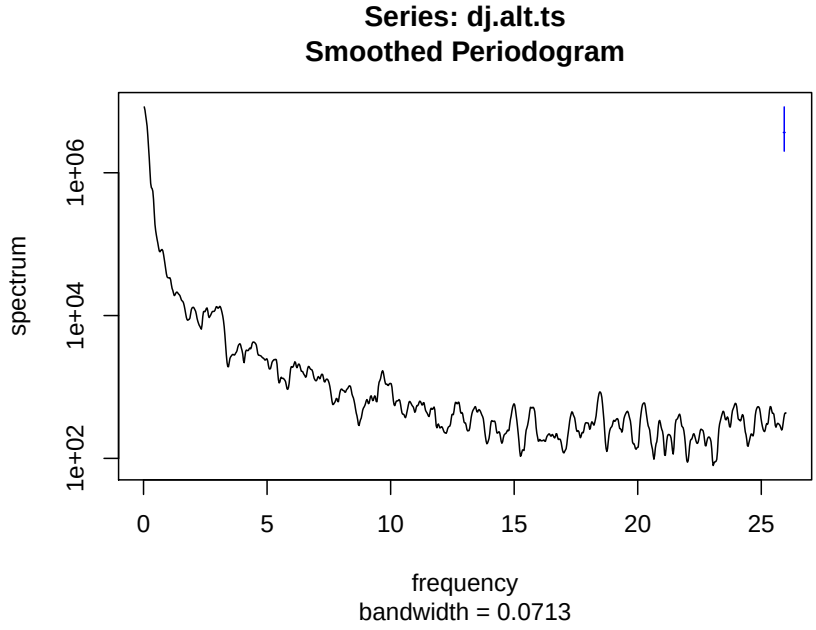
Example: PR Deaths

```
spec.pgram(pr.deaths.ts, span= c(3, 3))
```



Example: Dow Jones

```
spec.pgram(dj.alt.ts, span= c(3, 9))
```



and as we know there is no seasonal component here.