R (BGU course)

Jonathan D. Rosenblatt

2017-10-30

# Preface

This book accompanies BGU's "R" course, at the department of Industrial Engineering and Management. It has several purposes:

* Help me organize and document the course material.
* Help students during class so that they may focus on listening and not writing.
* Help students after class, so that they may self-study.

At its current state it is experimental. It can thus be expected to change from time to time, and include mistakes. I will be enormously grateful to whoever decides to share with me any mistakes found.

I am enormously grateful to Yihui Xie, who's *bookdown* R package made it possible to easily write a book which has many mathematical formulae, and R output.

I hope the reader will find this text interesting and useful.

For reproducing my results you will want to run set.seed(1).

## Notation Conventions

In this text we use the following conventions: Lower case may be a vector or a scalar, random of fixed, as implied by the context. Upper case will stand for matrices. Equality is an equality, and is a definition. Norm functions are denoted with for vector norms, and for matrix norms. The type of norm is indicated in the subscript; e.g. for the Euclidean () norm. Tag, is a transpose. The distribution of a random vector is .

## Acknowledgements

I have consulted many people during the writing of this text. I would like to thank [Yoav Kessler](https://kesslerlab.wordpress.com/), [Lena Novack](http://fohs.bgu.ac.il/research/profileBrief.aspx?id=VeeMVried), Efrat Vilenski, Ron Sarfian, and Liad Shekel in particular, for their valuable inputs.

# Introduction

## What is R?

R was not designed to be a bona-fide programming language. It is an evolution of the S language, developed at Bell labs (later Lucent) as a wrapper for the endless collection of statistical libraries they wrote in Fortran.

As of 2011, half of R's libraries are [actually written in C](https://wrathematics.github.io/2011/08/27/how-much-of-r-is-written-in-r/).

For more on the history of R see [AT&T's site](http://www.research.att.com/articles/featured_stories/2013_09/201309_SandR.html?fbid=Yxy4qyQzmMa), John Chamber's talk at [UserR! 2014](https://www.youtube.com/watch?v=_hcpuRB5nGs) or the Introduction to the excellent Venables and Ripley ([2013](#ref-venables2013modern)).

## The R Ecosystem

A large part of R's success is due to the ease in which a user, or a firm, can augment it. This led to a large community of users, developers, and protagonists. Some of the most important parts of R's ecosystem include:

* [CRAN](https://cran.r-project.org/): a repository for R packages, mirrored worldwide.
* [R-help](https://www.r-project.org/mail.html): an immensely active mailing list. Noways being replaced by StackExchange meta-site. Look for the R tags in the [StackOverflow](http://stackoverflow.com/) and [CrossValidated](http://stats.stackexchange.com/) sites.
* [TakViews](https://cran.r-project.org/web/views/): part of CRAN that collects packages per topic.
* [Bioconductor](https://www.bioconductor.org/): A CRAN-like repository dedicated to the life sciences.
* [Neuroconductor](https://www.neuroconductor.org/): A CRAN-like repository dedicated to neuroscience, and neuroimaging.
* [Books](https://www.r-project.org/doc/bib/R-books.html): An insane amount of books written on the language. Some are free, some are not.
* [The Israeli-R-user-group](https://groups.google.com/forum/#!forum/israel-r-user-group): just like the name suggests.
* Commercial R: being open source and lacking support may seem like a problem that would prohibit R from being adopted for commercial applications. This void is filled by several very successful commercial versions such as [Microsoft R](https://mran.microsoft.com/open/), with its accompanying CRAN equivalent called [MRAN](https://mran.microsoft.com/), [Tibco's Spotfire](http://spotfire.tibco.com/discover-spotfire/what-does-spotfire-do/predictive-analytics/tibco-enterprise-runtime-for-r-terr), and [others](https://en.wikipedia.org/wiki/R_(programming_language)#Commercial_support_for_R).
* [RStudio](https://www.rstudio.com/products/rstudio/download-server/): since its earliest days R came equipped with a minimal text editor. It later received plugins for major integrated development environments (IDEs) such as Eclipse, WinEdit and even [VisualStudio](https://www.visualstudio.com/vs/rtvs/). None of these, however, had the impact of the RStudio IDE. Written completely in JavaScript, the RStudio IDE allows the seamless integration of cutting edge web-design technologies, remote access, and other killer features, making it today's most popular IDE for R.

## Bibliographic Notes

## Practice Yourself

# R Basics

We now start with the basics of R. If you have any experience at all with R, you can probably skip this section.

First, make sure you work with the RStudio IDE. Some useful pointers for this IDE include:

* Ctrl+Return to run lines from editor.
* Alt+Shift+k for RStudio keyboard shortcuts.
* Ctrl+r to browse the command history.
* Alt+Shift+j to navigate between code sections
* tab for auto-completion
* Ctrl+1 to skip to editor.
* Ctrl+2 to skip to console.
* Ctrl+8 to skip to the environment list.
* Code Folding:
  + Alt+l collapse chunk.
  + Alt+Shift+l unfold chunk.
  + Alt+o collapse all.
  + Alt+Shift+o unfold all.

## File types

The file types you need to know when using R are the following:

* **.R**: An ASCII text file containing R scripts only.
* **.Rmd**: An ASCII text file. If opened in RStudio can be run as an R-Notebook or compiled using knitr, bookdown, etc.

## Simple calculator

R can be used as a simple calculator. Create a new R Notebook (.Rmd file) within RStudio using File-> New -> R Notebook, and run the following commands.

10+5

## [1] 15

70\*81

## [1] 5670

2\*\*4

## [1] 16

2^4

## [1] 16

log(10)

## [1] 2.302585

log(16, 2)

## [1] 4

log(1000, 10)

## [1] 3

## Probability calculator

R can be used as a probability calculator. You probably wish you knew this when you did your Intro To Probability classes.

The Binomial distribution function:

dbinom(x=3, size=10, prob=0.5) # Compute P(X=3) for X~B(n=10, p=0.5)

## [1] 0.1171875

Notice that arguments do not need to be named explicitly

dbinom(3, 10, 0.5)

## [1] 0.1171875

The Binomial cumulative distribution function (CDF):

pbinom(q=3, size=10, prob=0.5) # Compute P(X<=3) for X~B(n=10, p=0.5)

## [1] 0.171875

The Binomial quantile function:

qbinom(p=0.1718, size=10, prob=0.5) # For X~B(n=10, p=0.5) returns k such that P(X<=k)=0.1718

## [1] 3

Generate random variables:

rbinom(n=10, size=10, prob=0.5)

## [1] 4 4 5 7 4 7 7 6 6 3

R has many built-in distributions. Their names may change, but the prefixes do not:

* **d** prefix for the *distribution* function.
* **p** prefix for the *cummulative distribution* function (CDF).
* **q** prefix for the *quantile* function (i.e., the inverse CDF).
* **r** prefix to generate random samples.

Demonstrating this idea, using the CDF of several popular distributions:

* pbinom() for the Binomial CDF.
* ppois() for the Poisson CDF.
* pnorm() for the Gaussian CDF.
* pexp() for the Exponential CDF.

For more information see ?distributions.

## Getting Help

One of the most important parts of working with a language, is to know where to find help. R has several in-line facilities, besides the various help resources in the R [ecosystem](#ecosystem).

Get help for a particular function.

?dbinom   
help(dbinom)

If you don't know the name of the function you are looking for, search local help files for a particular string:

??binomial  
help.search('dbinom')

Or load a menu where you can navigate local help in a web-based fashion:

help.start()

## Variable Asignment

Assignment of some output into an object named "x":

x = rbinom(n=10, size=10, prob=0.5) # Works. Bad style.  
x <- rbinom(n=10, size=10, prob=0.5)

If you are familiar with other programming languages you may prefer the = assignment rather than the <- assignment. We recommend you make the effort to change your preferences. This is because thinking with <- helps to read your code, distinguishes between assignments and function arguments: think of function(argument=value) versus function(argument<-value). It also helps understand special assignment operators such as <<- and ->.

Remark. **Style**: We do not discuss style guidelines in this text, but merely remind the reader that good style is extremely important. When you write code, think of other readers, but also think of future self. See [Hadley's style guide](http://adv-r.had.co.nz/Style.html) for more.

To print the contents of an object just type its name

x

## [1] 7 4 6 3 4 5 2 5 7 4

which is an implicit call to

print(x)

## [1] 7 4 6 3 4 5 2 5 7 4

Alternatively, you can assign and print simultaneously using parenthesis.

(x <- rbinom(n=10, size=10, prob=0.5)) # Assign and print.

## [1] 5 5 5 4 6 6 6 3 6 5

Operate on the object

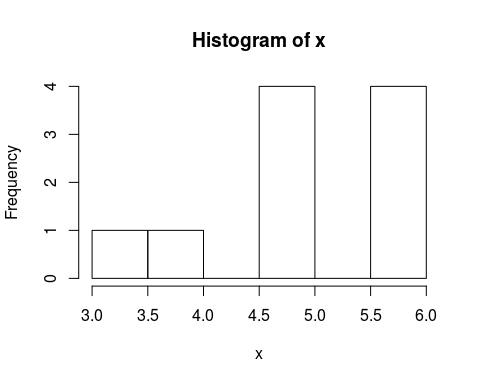
mean(x) # compute mean

## [1] 5.1

var(x) # compute variance

## [1] 0.9888889

hist(x) # plot histogram



R saves every object you create in RAM[[1]](#footnote-57). The collection of all such objects is the **workspace** which you can inspect with

ls()

## [1] "x"

or with Ctrl+8 in RStudio.

If you lost your object, you can use ls with a text pattern to search for it

ls(pattern='x')

## [1] "x"

To remove objects from the workspace:

rm(x) # remove variable  
ls() # verify

## character(0)

You may think that if an object is removed then its memory is freed. This is almost true, and depends on a negotiation mechanism between R and the operating system. R's memory management is discussed in Chapter 15.

## Missing

Unlike typically programming, when working with real life data, you may have **missing** values: measurements that were simply not recorded/stored/etc. *R* has rather sophosticated mechanisms to deal with missing values. It distinguishes between the following types:

1. NA: Not Available entries.
2. NaN: Not a number.

*R* tries to defend the analysit, and return an error, or NA when the presence of missing values invalidates the calculation:

missing.example <- c(10,11,12,NA)  
mean(missing.example)

## [1] NA

Most fnuctions will typically have an inner mechanism to deal with these. In the mean function, there is an na.rm argument, telling *R* how to Remove NAs.

mean(missing.example, na.rm = TRUE)

## [1] 11

A more general mechanism is removing these manually:

clean.example <- na.omit(missing.example)  
mean(clean.example)

## [1] 11

## Piping

Because R originates in Unix and Linux environments, it inherits much of its flavor. [Piping](http://ryanstutorials.net/linuxtutorial/piping.php) is an idea taken from the Linux shell which allows to use the output of one expression as the input to another. Piping thus makes code easier to read and write.

Remark. Volleyball fans may be confused with the idea of spiking a ball from the 3-meter line, also called [piping](https://www.youtube.com/watch?v=GWW15Nr1lQM). So: (a) These are very different things. (b) If you can pipe, [ASA-BGU](http://in.bgu.ac.il/sport/Pages/asa.aspx) is looking for you!

Prerequisites:

library(magrittr) # load the piping functions  
x <- rbinom(n=1000, size=10, prob=0.5) # generate some toy data

Examples

x %>% var() # Instead of var(x)  
x %>% hist() # Instead of hist(x)  
x %>% mean() %>% round(2) %>% add(10)

The next example[[2]](#footnote-63) demonstrates the benefits of piping. The next two chunks of code do the same thing. Try parsing them in your mind:

# Functional (onion) style  
car\_data <-   
 transform(aggregate(. ~ cyl,   
 data = subset(mtcars, hp > 100),   
 FUN = function(x) round(mean(x, 2))),   
 kpl = mpg\*0.4251)

# Piping (magrittr) style  
car\_data <-   
 mtcars %>%  
 subset(hp > 100) %>%  
 aggregate(. ~ cyl, data = ., FUN = . %>% mean %>% round(2)) %>%  
 transform(kpl = mpg %>% multiply\_by(0.4251)) %>%  
 print

Tip: RStudio has a keyboard shortcut for the %>% operator. Try Ctrl+Shift+m.

## Vector Creation and Manipulation

The most basic building block in R is the **vector**. We will now see how to create them, and access their elements (i.e. subsetting). Here are three ways to create the same arbitrary vector:

c(10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21) # manually  
10:21 # the `:` operator   
seq(from=10, to=21, by=1) # the seq() function

Let's assign it to the object named "x":

x <- c(10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21)

Operations usually work element-wise:

x+2

## [1] 12 13 14 15 16 17 18 19 20 21 22 23

x\*2

## [1] 20 22 24 26 28 30 32 34 36 38 40 42

x^2

## [1] 100 121 144 169 196 225 256 289 324 361 400 441

sqrt(x)

## [1] 3.162278 3.316625 3.464102 3.605551 3.741657 3.872983 4.000000  
## [8] 4.123106 4.242641 4.358899 4.472136 4.582576

log(x)

## [1] 2.302585 2.397895 2.484907 2.564949 2.639057 2.708050 2.772589  
## [8] 2.833213 2.890372 2.944439 2.995732 3.044522

## Search Paths and Packages

R can be easily extended with packages, which are merely a set of documented functions, which can be loaded or unloaded conveniently. Let's look at the function read.csv. We can see its contents by calling it without arguments:

read.csv

## function (file, header = TRUE, sep = ",", quote = "\"", dec = ".",   
## fill = TRUE, comment.char = "", ...)   
## read.table(file = file, header = header, sep = sep, quote = quote,   
## dec = dec, fill = fill, comment.char = comment.char, ...)  
## <bytecode: 0x67eb218>  
## <environment: namespace:utils>

Never mind what the function does. Note the environment: namespace:utils line at the end. It tells us that this function is part of the **utils** package. We did not need to know this because it is loaded by default. Here are some packages that I have currently loaded:

head(search())

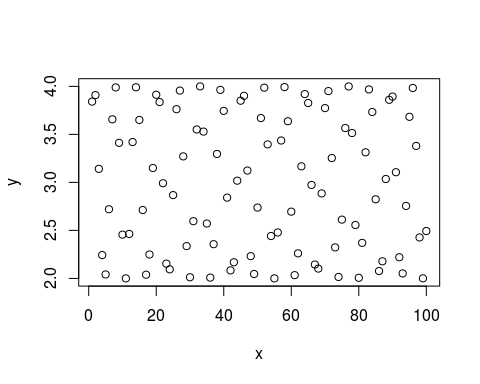
## [1] ".GlobalEnv" "package:doSNOW" "package:snow"   
## [4] "package:doParallel" "package:parallel" "package:iterators"

Other packages can be loaded via the library function, or downloaded from the internet using the install.packages function before loading with library. R's package import mechanism is quite powerful, and is one of the reasons for R's success.

## Simple Plotting

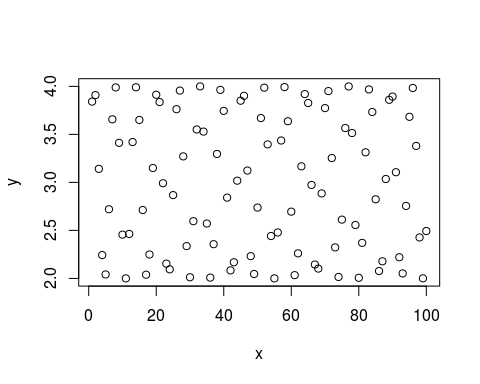
R has many plotting facilities as we will further detail in the Plotting Chapter 11. We start with the simplest facilities, namely, the plot function from the **graphics** package, which is loaded by default.

x<- 1:100  
y<- 3+sin(x)   
plot(x = x, y = y) # x,y syntax



Given an x argument and a y argument, plot tries to present a scatter plot. We call this the x,y syntax. R has another unique syntax to state functional relations. We call y~x the "tilde" syntax, which originates in works of G. Wilkinson and Rogers ([1973](#ref-wilkinson1973symbolic)) and was adopted in the early days of S.

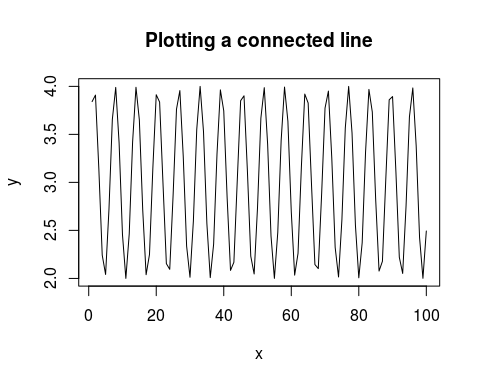
plot(y ~ x) # y~x syntax



The syntax y~x is read as "y is a function of x". We will prefer the y~x syntax over the x,y syntax since it is easier to read, and will be very useful when we discuss more complicated models.

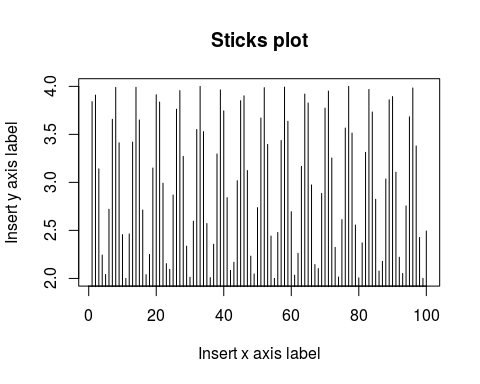
Here are some arguments that control the plot's appearance. We use type to control the plot type, main to control the main title.

plot(y~x, type='l', main='Plotting a connected line')



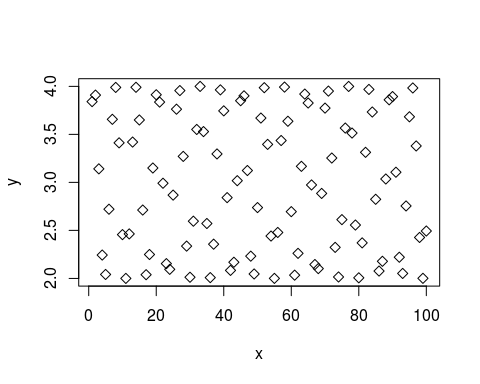
We use xlab for the x-axis label, ylab for the y-axis.

plot(y~x, type='h', main='Sticks plot', xlab='Insert x axis label', ylab='Insert y axis label')



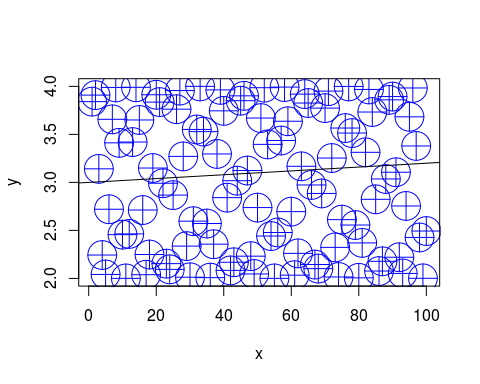
We use pch to control the point type.

plot(y~x, pch=5) # Point type with pcf



We use col to control the color, cex for the point size, and abline to add a straight line.

plot(y~x, pch=10, type='p', col='blue', cex=4)   
abline(3, 0.002)



For more plotting options run these

example(plot)  
example(points)  
?plot  
help(package='graphics')

When your plotting gets serious, go to Chapter 11.

## Object Types

We already saw that the basic building block of R objects is the vector. Vectors can be of the following types:

* **character** Where each element is a string, i.e., a sequence of alphanumeric symbols.
* **numeric** Where each element is a [real number](https://en.wikipedia.org/wiki/Real_number) in [double precision](https://en.wikipedia.org/wiki/Double-precision_floating-point_format) floating point format.
* **integer** Where each element is an [integer](https://en.wikipedia.org/wiki/Integer).
* **logical** Where each element is either TRUE, FALSE, or NA[[3]](#footnote-78)
* **complex** Where each element is a complex number.
* **list** Where each element is an arbitrary R object.
* **factor** Factors are not actually vector objects, but they feel like such. They are used to encode any finite set of values. This will be very useful when fitting linear model because they include information on contrasts, i.e., on the encoding of the factors levels. You should always be alert and recall when you are dealing with a factor or with a character vector. They have different behaviors.

Vectors can be combined into larger objects. A matrix can be thought of as the binding of several vectors of the same type. In reality, a matrix is merely a vector with a dimension attribute, that tells R to read it as a matrix and not a vector.

If vectors of different types (but same length) are binded, we get a data.frame which is the most fundamental object in R for data analysis. Data frames are brilliant, but a lot has been learned since their invention. They have thus been extended in recent years with the tbl class, pronounced [Tibble] (<https://cran.r-project.org/web/packages/tibble/vignettes/tibble.html>), and the data.table class.  
The latter is discussed in Chapter 21, and is strongly recommended.

## Data Frames

Creating a simple data frame:

x<- 1:10  
y<- 3 + sin(x)   
frame1 <- data.frame(x=x, sin=y)

Let's inspect our data frame:

head(frame1)

## x sin  
## 1 1 3.841471  
## 2 2 3.909297  
## 3 3 3.141120  
## 4 4 2.243198  
## 5 5 2.041076  
## 6 6 2.720585

Now using the RStudio Excel-like viewer:

frame1 %>% View()

We highly advise against editing the data this way since there will be no documentation of the changes you made. Always transform your data using scripts, so that everything is documented.

Verifying this is a data frame:

class(frame1) # the object is of type data.frame

## [1] "data.frame"

Check the dimension of the data

dim(frame1)

## [1] 10 2

Note that checking the dimension of a vector is different than checking the dimension of a data frame.

length(x)

## [1] 10

The length of a data.frame is merely the number of columns.

length(frame1)

## [1] 2

## Exctraction

R provides many ways to subset and extract elements from vectors and other objects. The basics are fairly simple, but not paying attention to the "personality" of each extraction mechanism may cause you a lot of headache.

For starters, extraction is done with the [ operator. The operator can take vectors of many types.

Extracting element with by integer index:

frame1[1, 2] # exctract the element in the 1st row and 2nd column.

## [1] 3.841471

Extract **column** by index:

frame1[,1]

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

Extract column by name:

frame1[, 'sin']

## [1] 3.841471 3.909297 3.141120 2.243198 2.041076 2.720585 3.656987  
## [8] 3.989358 3.412118 2.455979

As a general rule, extraction with [ will conserve the class of the parent object. There are, however, exceptions. Notice the extraction mechanism and the class of the output in the following examples.

class(frame1[, 'sin']) # extracts a column vector

## [1] "numeric"

class(frame1['sin']) # extracts a data frame

## [1] "data.frame"

class(frame1[,1:2]) # extracts a data frame

## [1] "data.frame"

class(frame1[2]) # extracts a data frame

## [1] "data.frame"

class(frame1[2, ]) # extract a data frame

## [1] "data.frame"

class(frame1$sin) # extracts a column vector

## [1] "numeric"

The subset() function does the same

subset(frame1, select=sin)   
subset(frame1, select=2)  
subset(frame1, select= c(2,0))

If you want to force the stripping of the class attribute when extracting, try the [[ mechanism instead of [.

a <- frame1[1] # [ extraction  
b <- frame1[[1]] # [[ extraction  
class(a)==class(b) # objects have differing classes

## [1] FALSE

a==b # objects are element-wise identical

## x  
## [1,] TRUE  
## [2,] TRUE  
## [3,] TRUE  
## [4,] TRUE  
## [5,] TRUE  
## [6,] TRUE  
## [7,] TRUE  
## [8,] TRUE  
## [9,] TRUE  
## [10,] TRUE

The different types of output classes cause different behaviors. Compare the behavior of [ on seemingly identical objects.

frame1[1][1]

## x  
## 1 1  
## 2 2  
## 3 3  
## 4 4  
## 5 5  
## 6 6  
## 7 7  
## 8 8  
## 9 9  
## 10 10

frame1[[1]][1]

## [1] 1

If you want to learn more about subsetting see [Hadley's guide](http://adv-r.had.co.nz/Subsetting.html).

## Non data.frame object classes

As previously mentioned, the data.frame class has been extended in recent years. The best known extensions are the data.table and the tbl. For beginners, it is important to know R's basics, so we keep focusing on data frames. For more advanced users, I recommend learning the (amazing) data.table syntax.

## Data Import and Export

For any practical purpose, you will not be generating your data manually. R comes with many importing and exporting mechanisms which we now present. If, however, you do a lot of data "munging", make sure to see Hadley-verse Chapter 13. If you work with MASSIVE data sets, read the Memory Efficiency Chapter 15.

### Import from WEB

The read.table function is the main importing workhorse. It can import directly from the web.

URL <- 'http://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/bone.data'  
tirgul1 <- read.table(URL)

Always look at the imported result!

head(tirgul1)

## V1 V2 V3 V4  
## 1 idnum age gender spnbmd  
## 2 1 11.7 male 0.01808067  
## 3 1 12.7 male 0.06010929  
## 4 1 13.75 male 0.005857545  
## 5 2 13.25 male 0.01026393  
## 6 2 14.3 male 0.2105263

Ohh dear. read.,table tried to guess the structure of the input, but failed to recognize the header row. Set it manually with header=TRUE:

tirgul1 <- read.table('data/bone.data', header = TRUE)   
head(tirgul1)

### Export as CSV

Let's write a simple file so that we have something to import

head(airquality) # examine the data to export

## Ozone Solar.R Wind Temp Month Day  
## 1 41 190 7.4 67 5 1  
## 2 36 118 8.0 72 5 2  
## 3 12 149 12.6 74 5 3  
## 4 18 313 11.5 62 5 4  
## 5 NA NA 14.3 56 5 5  
## 6 28 NA 14.9 66 5 6

temp.file.name <- tempfile() # get some arbitrary file name  
write.csv(x = airquality, file = temp.file.name) # export

Now let's import the exported file. Being a .csv file, I can use read.csv instead of read.table.

my.data<- read.csv(file=temp.file.name) # import  
head(my.data) # verify import

## X Ozone Solar.R Wind Temp Month Day  
## 1 1 41 190 7.4 67 5 1  
## 2 2 36 118 8.0 72 5 2  
## 3 3 12 149 12.6 74 5 3  
## 4 4 18 313 11.5 62 5 4  
## 5 5 NA NA 14.3 56 5 5  
## 6 6 28 NA 14.9 66 5 6

Remark. Windows users may need to use "\" instead of "/".

### Export non-CSV files

You can export your R objects in endlessly many ways: If insread of the comma delimiter in .csv you want other column delimiters, look into ?write.table. If you are exporting only for R users, you can consider exporting as binary objects with saveRDS, feather::write\_feather, or fst::write.fst. See (<http://www.fstpackage.org/>) for a comparison.

### Reading From Text Files

Some general notes on importing text files via the read.table function. But first, we need to know what is the active directory. Here is how to get and set R's active directory:

getwd() #What is the working directory?  
setwd() #Setting the working directory in Linux

We can now call the read.table function to import text files. If you care about your sanity, see ?read.table before starting imports. Some notable properties of the function:

* read.table will try to guess column separators (tab, comma, etc.)
* read.table will try to guess if a header row is present.
* read.table will convert character vectors to factors unless told not to using the stringsAsFactors=FALSE argument.
* The output of read.table needs to be explicitly assigned to an object for it to be saved.

### Writing Data to Text Files

The function write.table is the exporting counterpart of read.table.

### .XLS(X) files

Strongly recommended to convert to .csv in Excel, and then import as csv. If you still insist see the **xlsx** package.

### Massive files

The above importing and exporting mechanisms were not designed for massive files. See the section on Sparse Representation (14) and Out-of-Ram Algorithms (15) for more on working with massive data files.

### Databases

R does not need to read from text files; it can read directly from a database. This is very useful since it allows the filtering, selecting and joining operations to rely on the database's optimized algorithms. Then again, if you will only be analyzing your data with R, you are probably better of by working from a file, without the databases' overhead. See Chapter 15 for more on this matter.

## Functions

One of the most basic building blocks of programming is the ability of writing your own functions. A function in R, like everything else, is an object accessible using its name. We first define a simple function that sums its two arguments

my.sum <- function(x,y) {  
 return(x+y)  
}  
my.sum(10,2)

## [1] 12

From this example you may notice that:

* The function function tells R to construct a function object.
* The arguments of the function, i.e. (x,y), need to be named but we are not required to specify their type. This makes writing functions very easy, but it is also the source of many bugs, and slowness of R compared to type declaring languages (C, Fortran,Java,...).
* A typical R function does not change objects[[4]](#footnote-96) but rather creates new ones. To save the output of my.sum we will need to assign it using the <- operator.

Here is a (slightly) more advanced example.

my.sum.2 <- function(x, y , absolute=FALSE) {  
 if(absolute==TRUE) {  
 result <- abs(x+y)  
 }  
 else{  
 result <- x+y  
 }   
 result  
}  
my.sum.2(-10,2, TRUE)

## [1] 8

Things to note:

* if(condition){expression1} else{expression2} does just what the name suggests.
* The function will output its last evaluated expression. You don't need to use the return function explicitly.
* Using absolute=FALSE sets the default value of absolute to FALSE. This is overridden if absolute is stated explicitly in the function call.

An important behavior of R is the *scoping rules*. This refers to the way R seeks for variables used in functions. As a rule of thumb, R will first look for variables inside the function and if not found, will search for the variable values in outer environments[[5]](#footnote-98). Think of the next example.

a <- 1  
b <- 2  
x <- 3  
scoping <- function(a,b){  
 a+b+x  
}  
scoping(10,11)

## [1] 24

## Looping

The real power of scripting is when repeated operations are done by iteration. R supports the usual for, while, and repated loops. Here is an embarrassingly simple example

for (i in 1:5){  
 print(i)  
 }

## [1] 1  
## [1] 2  
## [1] 3  
## [1] 4  
## [1] 5

A slightly more advanced example, is vector multiplication

result <- 0  
for(i in 1:n){  
 result <- result+ x[i]\*y[i]  
}

Remark. **Vector Operations**: You should NEVER write your own vector and matrix products like in the previous example. Only use existing facilities such as %\*%, sum(), etc.

## Apply

For applying the same function to a set of elements, there is no need to write an explicit loop. This is such en elementary operation that every programming language will provide some facility to **apply**, or **map** the function to all elements of a set. R provides several facilities to perform this. The most basic of which is lapply which applies a function over all elements of a list, and return a list of outputs:

the.list <- list(1,'a',mean) # a list of 3 elements from different calsses  
lapply(X = the.list, FUN = class) # apply the function `class` to each elements

## [[1]]  
## [1] "numeric"  
##   
## [[2]]  
## [1] "character"  
##   
## [[3]]  
## [1] "standardGeneric"  
## attr(,"package")  
## [1] "methods"

R provides many variations on lapply to facilitate programming. Here is a partial list:

* sapply: The same as lapply but tries to arrange output in a vector or matrix, and not an unstructured list.
* vapply: A safer version of sapply, where the output class is pre-specified.
* apply: For applying over the rows or columns of matrices.
* mapply: For applying functions with various inputs.
* tapply: For splitting vectors and applying functions on subsets.
* rapply: A recursive version of lapply.
* eapply: Like lapply, only operates on environments instead of lists.
* Map+Reduce: For a [Common Lisp](https://en.wikipedia.org/wiki/Common_Lisp) look and feel of lapply.
* parallel::parLapply: A parallel version of lapply from the package **parallel**.
* parallel::parLBapply: A parallel version of lapply, with load balancing from the package **parallel**.

## Recursion

The R compiler is really not designed for recursion, and you will rarely need to do so.  
See the RCpp Chapter 19 for linking C code, which is better suited for recursion. If you really insist to write recursions in R, make sure to use the Recall function, which, as the name suggests, recalls the function in which it is place. Here is a demonstration with the Fibonacci series.

fib<-function(n) {  
 if (n < 2) fn<-1   
 else fn<-Recall(n - 1) + Recall(n - 2)   
 return(fn)  
}   
fib(5)

## [1] 8

## Dates and Times

[TODO]

## Bibliographic Notes

There are endlessly many introductory texts on R. For a list of free resources see [CrossValidated](http://stats.stackexchange.com/questions/138/free-resources-for-learning-r). I personally recommend the official introduction Venables et al. ([2004](#ref-venables2004introduction)), [available online](https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf), or anything else Bill Venables writes. For advanced R programming see Wickham ([2014](#ref-wickham2014advanced)), [available online](http://adv-r.had.co.nz/), or anything else Hadley Wickham writes.

For Imporitng and Exporting see (<https://cran.r-project.org/doc/manuals/r-release/R-data.html>). For working with databases see (<https://rforanalytics.wordpress.com/useful-links-for-r/odbc-databases-for-r/>).

## Practice Yourself

1. Save the numbers 1 to 100 into an object named object.
2. Write a loop that computes the mean of object.
3. Write a function that computes the mean of its input, and apply it on object.
4. Make a matrix of random numbers using A <- matrix(rnorm(40), ncol=10, nrow=4). Compute the mean of each columns. Do it using your own loop and then do the same with lapply or apply.

# Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a term cast by [John W. Tukey](https://en.wikipedia.org/wiki/John_Tukey) in his seminal book (Tukey [1977](#ref-tukey1977exploratory)). It is also (arguably) known as *Visual Analytics*, or *Descriptive Statistics*. It is the practice of inspecting, and exploring your data, before stating hypotheses, fitting predictors, and other more ambitious inferential goals. It typically includes the computation of simple *summary statistics* which capture some property of interest in the data, and *visualization*. EDA can be thought of as an assumption free, purely algorithmic practice.

In this text we present EDA techniques along the following lines:

* How we explore: with summary statistics, or visually?
* How many variables analyzed simultaneously: univariate, bivariate, or multivariate?
* What type of variable: categorical or continuous?

## Summary Statistics

### Categorical Data

Categorical variables do not admit any mathematical operations on them. We cannot sum them, or even sort them. We can only **count** them. As such, summaries of categorical variables will always start with the counting of the frequency of each category.

#### Summary of Univariate Categorical Data

# Make some data  
gender <- c(rep('Boy', 10), rep('Girl', 12))  
drink <- c(rep('Coke', 5), rep('Sprite', 3), rep('Coffee', 6), rep('Tea', 7), rep('Water', 1))   
age <- sample(c('Young', 'Old'), size = length(gender), replace = TRUE)  
# Count frequencies  
table(gender)

## gender  
## Boy Girl   
## 10 12

table(drink)

## drink  
## Coffee Coke Sprite Tea Water   
## 6 5 3 7 1

table(age)

## age  
## Old Young   
## 14 8

If instead of the level counts you want the proportions, you can use prop.table

prop.table(table(gender))

## gender  
## Boy Girl   
## 0.4545455 0.5454545

#### Summary of Bivariate Categorical Data

library(magrittr)  
cbind(gender, drink) %>% head # bind vectors into matrix and inspect

## gender drink   
## [1,] "Boy" "Coke"   
## [2,] "Boy" "Coke"   
## [3,] "Boy" "Coke"   
## [4,] "Boy" "Coke"   
## [5,] "Boy" "Coke"   
## [6,] "Boy" "Sprite"

table1 <- table(gender, drink) # count frequencies of bivariate combinations  
table1

## drink  
## gender Coffee Coke Sprite Tea Water  
## Boy 2 5 3 0 0  
## Girl 4 0 0 7 1

#### Summary of Multivariate Categorical Data

You may be wondering how does R handle tables with more than two dimensions. It is indeed not trivial to report this in a human-readable way. R offers several solutions: table is easier to compute with, and ftable is human readable.

table2.1 <- table(gender, drink, age) # A multilevel table.   
table2.1

## , , age = Old  
##   
## drink  
## gender Coffee Coke Sprite Tea Water  
## Boy 2 3 1 0 0  
## Girl 2 0 0 5 1  
##   
## , , age = Young  
##   
## drink  
## gender Coffee Coke Sprite Tea Water  
## Boy 0 2 2 0 0  
## Girl 2 0 0 2 0

table.2.2 <- ftable(gender, drink, age) # A human readable table.  
table.2.2

## age Old Young  
## gender drink   
## Boy Coffee 2 0  
## Coke 3 2  
## Sprite 1 2  
## Tea 0 0  
## Water 0 0  
## Girl Coffee 2 2  
## Coke 0 0  
## Sprite 0 0  
## Tea 5 2  
## Water 1 0

If you want proportions instead of counts, you need to specify the denominator, i.e., the margins. Think: what is the margin in each of the following outputs?

prop.table(table1, margin = 1)

## drink  
## gender Coffee Coke Sprite Tea Water  
## Boy 0.20000000 0.50000000 0.30000000 0.00000000 0.00000000  
## Girl 0.33333333 0.00000000 0.00000000 0.58333333 0.08333333

prop.table(table1, margin = 2)

## drink  
## gender Coffee Coke Sprite Tea Water  
## Boy 0.3333333 1.0000000 1.0000000 0.0000000 0.0000000  
## Girl 0.6666667 0.0000000 0.0000000 1.0000000 1.0000000

### Continous Data

Continuous variables admit many more operations than categorical. We can compute sums, means, quantiles, and more.

#### Summary of Univariate Continuous Data

We distinguish between several types of summaries, each capturing a different property of the data.

#### Summary of Location

Capture the "location" of the data. These include:

Definition 1 (Average) The mean, or average, of a sample , denoted is defined as

The sample mean is **non robust**. A single large observation may inflate the mean indefinitely. For this reason, we define several other summaries of location, which are more robust, i.e., less affected by "contaminations" of the data.

We start by defining the sample quantiles, themselves **not** a summary of location.

Definition 2 (Quantiles) The quantile of a sample , denoted , is (non uniquely) defined as a value above of the sample, and below .

We emphasize that sample quantiles are non-uniquely defined. See ?quantile for the 9(!) different definitions that R provides.

Using the sample quantiles, we can now define another summary of location, the **median**.

Definition 3 (Median) The median of a sample , denoted is the quantile of the sample.

A whole family of summaries of locations is the **alpha trimmed mean**.

Definition 4 (Alpha Trimmed Mean) The trimmed mean of a sample , denoted is the average of the sample after removing the proportion of largest and proportion of smallest observations.

The simple mean and median are instances of the alpha trimmed mean: and respectively.

Here are the R implementations:

x <- rexp(100) # generate some random data  
mean(x) # simple mean

## [1] 0.886218

median(x) # median

## [1] 0.5681976

mean(x, trim = 0.2) # alpha trimmed mean with alpha=0.2

## [1] 0.6537379

#### Summary of Scale

The *scale* of the data, sometimes known as *spread*, can be thought of its variability.

Definition 5 (Standard Deviation) The standard deviation of a sample , denoted , is defined as

For reasons of robustness, we define other, more robust, measures of scale.

Definition 6 (MAD) The Median Absolute Deviation from the median, denoted as , is defined as

where is some constant, typically set to so that MAD and have the same large sample limit.

Definition 7 (IQR) The Inter Quantile Range of a sample , denoted as , is defined as

Here are the R implementations

sd(x) # standard deviation

## [1] 0.9388276

mad(x) # MAD

## [1] 0.5584909

IQR(x) # IQR

## [1] 1.11348

#### Summary of Asymmetry

Summaries of asymmetry, also known as *skewness*, quantify the departure of the from a symmetric sample.

Definition 8 (Yule) The Yule measure of assymetry, denoted is defined as

Here is an R implementation

yule <- function(x){  
 numerator <- 0.5 \* (quantile(x,0.75) + quantile(x,0.25))-median(x)   
 denominator <- 0.5\* IQR(x)  
 c(numerator/denominator, use.names=FALSE)  
}  
yule(x)

## [1] 0.4051289

#### Summary of Bivariate Continuous Data

When dealing with bivariate, or multivariate data, we can obviously compute univariate summaries for each variable separately. This is not the topic of this section, in which we want to summarize the association **between** the variables, and not within them.

Definition 9 (Covariance) The covariance between two samples, and , of same length , is defined as

We emphasize this is not the covariance you learned about in probability classes, since it is not the covariance between two *random variables* but rather, between two *samples*. For this reasons, some authors call it the *empirical covariance*, or *sample covariance*.

Definition 10 (Pearson's Correlation Coefficient) Peasrson's correlation coefficient, a.k.a. Pearson's moment product correlation, or simply, the correlation, denoted r(x,y), is defined as

If you find this definition enigmatic, just think of the correlation as the covariance between and after transforming each to the unitless scale of z-scores.

Definition 11 (Z-Score) The z-scores of a sample are defined as the mean-centered, scale normalized observations:

We thus have that .

#### Summary of Multivariate Continuous Data

The covariance is a simple summary of association between two variables, but it certainly may not capture the whole "story" when dealing with more than two variables. The most common summary of multivariate relation, is the **covariance matrix**, but we warn that only the simplest multivariate relations are fully summarized by this matrix.

Definition 12 (Sample Covariance Matrix) Given observations on variables, denote the 'th observation of the 'th variable. The *sample covariance matrix*, denoted is defined as

where . Put differently, the 'th entry in is the sample covariance between variables and .

Remark. is clearly non robust. How would you define a robust covariance matrix?

## Visualization

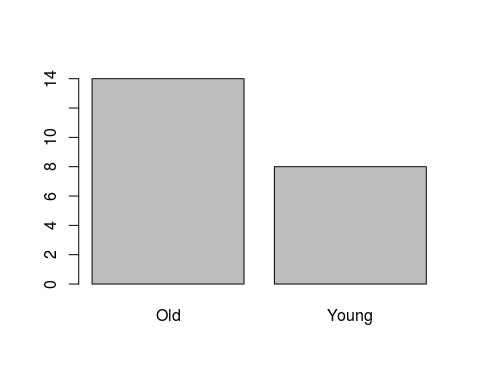
Summarizing the information in a variable to a single number clearly conceals much of the story in the sample. This is akin to inspecting a person using a caricature, instead of a picture. Visualizing the data, when possible, is more informative.

### Categorical Data

Recalling that with categorical variables we can only count the frequency of each level, the plotting of such variables are typically variations on the *bar plot*.

#### Visualizing Univariate Categorical Data

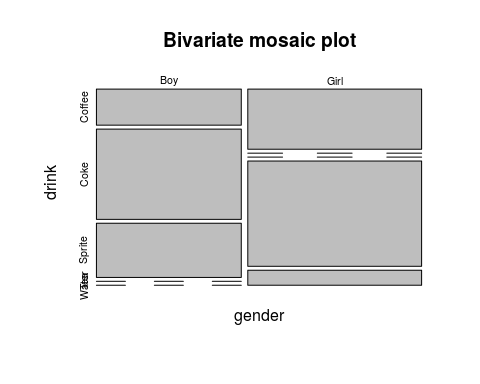
barplot(table(age))



#### Visualizing Bivariate Categorical Data

There are several generalizations of the barplot, aimed to deal with the visualization of bivariate categorical data. They are sometimes known as the *clustered bar plot* and the *stacked bar plot*. In this text, we advocate the use of the *mosaic plot* which is also the default in R.

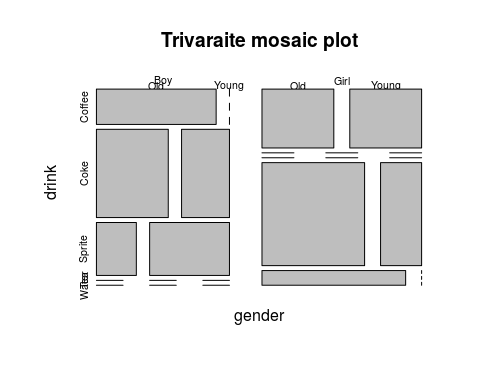
plot(table1, main='Bivariate mosaic plot')



#### Visualizing Multivariate Categorical Data

The *mosaic plot* is not easy to generalize to more than two variables, but it is still possible (at the cost of interpretability).

plot(table2.1, main='Trivaraite mosaic plot')

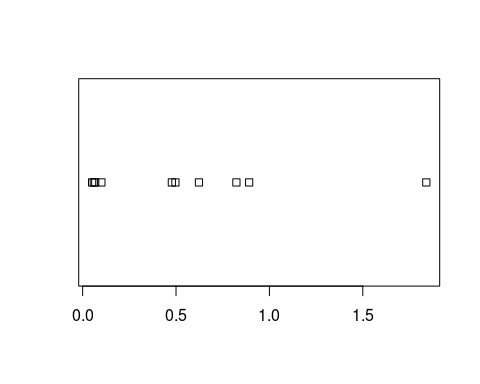


### Continuous Data

#### Visualizing Univariate Continuous Data

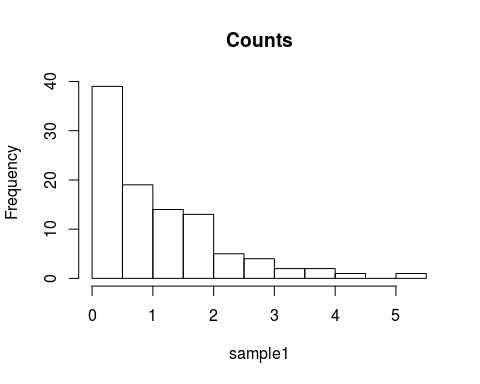
Unlike categorical variables, there are endlessly many way to visualize continuous variables. The simplest way is to look at the raw data via the stripchart.

sample1 <- rexp(10)   
stripchart(sample1)



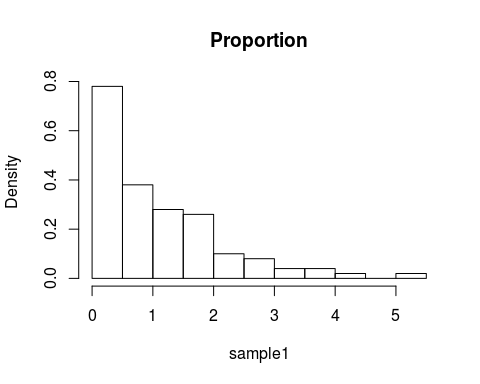
Clearly, if there are many observations, the stripchart will be a useless line of black dots. We thus bin them together, and look at the frequency of each bin; this is the *histogram*. R's histogram function has very good defaults to choose the number of bins. Here is a histogram showing the counts of each bin.

sample1 <- rexp(100)   
hist(sample1, freq=T, main='Counts')



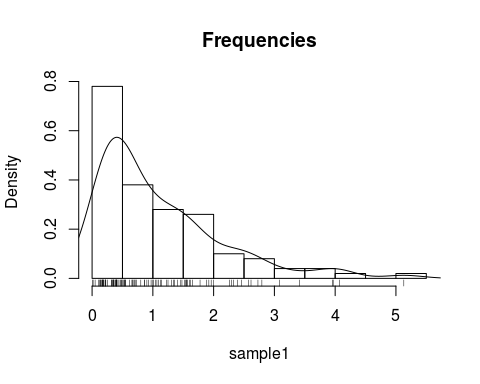
The bin counts can be replaced with the proportion of each bin using the freq argument.

hist(sample1, freq=F, main='Proportion')



The bins of a histogram are non overlapping. We can adopt a sliding window approach, instead of binning. This is the *density plot* which is produced with the density function, and added to an existing plot with the lines function. The rug function adds the original data points as ticks on the axes, and is strongly recommended to detect artifacts introduced by the binning of the histogram, or the smoothing of the density plot.

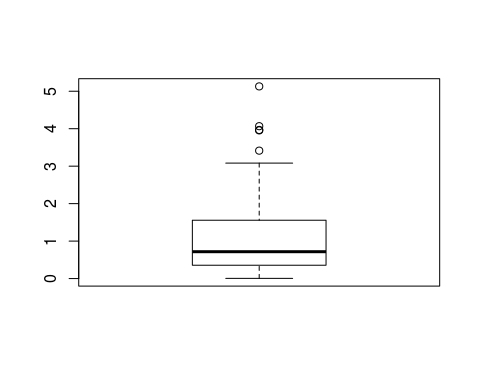
hist(sample1, freq=F, main='Frequencies')   
lines(density(sample1))   
rug(sample1)



Remark. Why would it make no sense to make a table, or a barplot, of continuous data?

One particularly useful visualization, due to John W. Tukey, is the *boxplot*. The boxplot is designed to capture the main phenomena in the data, and simultaneously point to outlines.

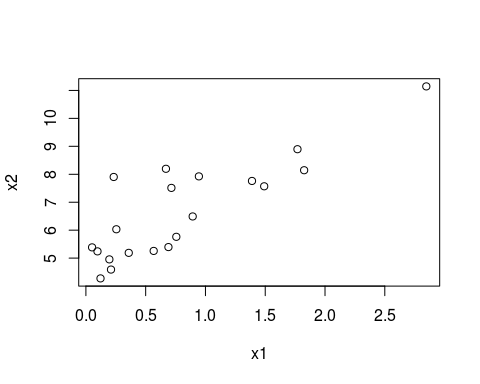
boxplot(sample1)



#### Visualizing Bivariate Continuous Data

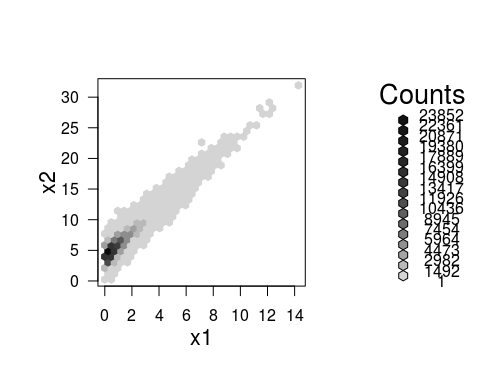
The bivariate counterpart of the stipchart is the celebrated scatter plot.

n <- 20  
x1 <- rexp(n)  
x2 <- 2\* x1 + 4 + rexp(n)  
plot(x2~x1)



Like the univariate stripchart, the scatter plot will be an uninformative mess in the presence of a lot of data. A nice bivariate counterpart of the univariate histogram is the *hexbin plot*, which tessellates the plane with hexagons, and reports their frequencies.

library(hexbin) # load required library  
n <- 2e5  
x1 <- rexp(n)  
x2 <- 2\* x1 + 4 + rnorm(n)  
plot(hexbin(x = x1, y = x2))

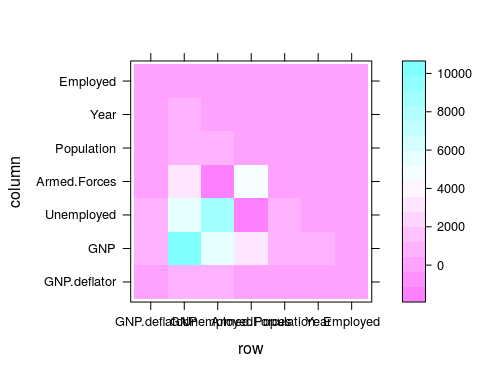


#### Visualizing Multivariate Continuous Data

Visualizing multivariate data is a tremendous challenge given that we cannot grasp dimensional spaces, nor can the computer screen present more than dimensional spaces. We thus have several options: (i) To project the data to 2D. This is discussed in the Dimensionality Reduction Section 10.1. (ii) To visualize not the raw data, but rather its summaries, like the covariance matrix.

Since the covariance matrix, is a matrix, it can be visualized as an image. Note the use of the :: operator, which is used to call a function from some package, without loading the whole package. We will use the :: operator when we want to emphasize the package of origin of a function.

covariance <- cov(longley) # The covariance of the longley dataset  
lattice::levelplot(covariance)



## Bibliographic Notes

Like any other topic in this book, you can consult Venables and Ripley ([2013](#ref-venables2013modern)). The seminal book on EDA, written long before R was around, is Tukey ([1977](#ref-tukey1977exploratory)). For an excellent text on robust statistics see Wilcox ([2011](#ref-wilcox2011introduction)).

## Practice Yourself

1. Read about the Titanic data set using ?Titanic. Inspect it with the table and with the ftable commands. Which do you prefer?
2. Inspect the Titanic data with a plot. Start with plot(Titanic) Try also lattice::dotplot. Which is the passenger category with most survivors? Which plot do you prefer? Which scales better to more categories?
3. Read about the women data using ?women.
   1. Compute the average of each variable. What is the average of the heights?
   2. Plot a histogram of the heights. Add ticks using rug.
   3. Plot a boxplot of the weights.
   4. Plot the heights and weights using a scatter plot. Add ticks using rug.
4. Choose to define a new symmetry measure: . Write a function that computes it, and apply it on women's heights data.
5. Compute the covariance matrix of women's heights and weights. Compute the correlation matrix. View the correlation matrix as an image using lattice::levelplot.
6. Pick a dataset with two LONG continous variables from ?datasets. Plot it using hexbin::hexbin.

# Linear Models

## Problem Setup

Example 1 (Bottle Cap Production) Consider a randomized experiment designed to study the effects of temperature and pressure on the diameter of manufactured a bottle cap.

Example 2 (Rental Prices) Consider the prediction of rental prices given an appartment's attributes.

Both examples require some statistical model, but they are very different. The first is a *causal inference* problem: we want to design an intervention so that we need to recover the causal effect of temperature and pressure. The second is a *prediction* problem, a.k.a. a *forecasting* problem, in which we don't care about the causal effects, we just want good predictions.

In this chapter we discuss the causal problem in Example 1. This means that when we assume a model, we assume it is the actual *data generating process*, i.e., we assume the *sampling distribution* is well specified. The second type of problems is discussed in the Supervised Learning Chapter 9.

Here are some more examples of the types of problems we are discussing.

Example 3 (Plant Growth) Consider the treatment of various plants with various fertilizers to study the fertilizer's effect on growth.

Example 4 (Return to Education) Consider the study of return to education by analyzing the incomes of individuals with different education years.

Example 5 (Drug Effect) Consider the study of the effect of a new drug, by analyzing the level of blood coagulation after the administration of various amounts of the new drug.

Let's present the linear model. We assume that a response[[6]](#footnote-150) variable is the sum of effects of some factors[[7]](#footnote-151). Denoting the dependent by , the factors by , and the effects by the linear model assumption implies that the expected response is the sum of the factors effects:

Clearly, there may be other factors that affect the the caps' diameters. We thus introduce an error term[[8]](#footnote-152), denoted by , to capture the effects of all unmodeled factors and measurement error[[9]](#footnote-153). The implied generative process of a sample of observations it thus

or in matrix notation

Let's demonstrate Eq.(2). In our cap example (1), assuming that pressure and temperature have two levels each (say, high and low), we would write if the pressure of the 'th measurement was set to high, and if the **pressure** was set to low. Similarly, we would write , and , if the **temperature** was set to high, or low, respectively. The coding with is known as *effect coding*. If you prefer coding with , this is known as *dummy coding*. The choice of coding has no real effect on the results, provided that you remember what coding you used when interpreting .

Remark. In [Galton's](https://en.wikipedia.org/wiki/Regression_toward_the_mean) classical regression problem, where we try to seek the relation between the heights of sons and fathers then , is the height of the 'th father, and the height of the 'th son.

There are many reasons linear models are very popular:

1. Before the computer age, these were pretty much the only models that could actually be computed[[10]](#footnote-155). The whole Analysis of Variance (ANOVA) literature is an instance of linear models, that relies on sums of squares, which do not require a computer to work with.
2. For purposes of prediction, where the actual data generating process is not of primary importance, they are popular because they simply work. Why is that? They are simple so that they do not require a lot of data to be computed. Put differently, they may be biased, but their variance is small enough to make them more accurate than other models.
3. For non continuous predictors, **any** functional relation can be cast as a linear model.
4. For the purpose of *screening*, where we only want to show the existence of an effect, and are less interested in the magnitude of that effect, a linear model is enough.
5. If the true generative relation is not linear, but smooth enough, then the linear function is a good approximation via Taylor's theorem.

There are still two matters we have to attend: (i) How the estimate ? (ii) How to perform inference?

In the simplest linear models the estimation of is done using the method of least squares. A linear model with least squares estimation is known as Ordinary Least Squares (OLS). The OLS problem:

and in matrix notation

Remark. Personally, I prefer the matrix notation because it is suggestive of the geometry of the problem. The reader is referred to Friedman, Hastie, and Tibshirani ([2001](#ref-friedman2001elements)), Section 3.2, for more on the geometry of OLS.

Different software suits, and even different R packages, solve Eq.(4) in different ways so that we skip the details of how exactly it is solved. These are discussed in Chapters 17 and 18.

The last matter we need to attend is how to do inference on . For that, we will need some assumptions on . A typical set of assumptions is the following:

1. **Independence**: we assume are independent of everything else. Think of them as the measurement error of an instrument: it is independent of the measured value and of previous measurements.
2. **Centered**: we assume that , meaning there is no systematic error.
3. **Normality**: we will typically assume that , but we will later see that this is not really required.

We emphasize that these assumptions are only needed for inference on and not for the estimation itself, which is done by the purely algorithmic framework of OLS.

Given the above assumptions, we can apply some probability theory and linear algebra to get the distribution of the estimation error:

The reason I am not too strict about the normality assumption above, is that Eq.(6) is approximately correct even if is not normal, provided that there are many more observations than factors ().

## OLS Estimation in R

We are now ready to estimate some linear models with R. We will use the whiteside data from the **MASS** package, recording the outside temperature and gas consumption, before and after an apartment's insulation.

library(MASS) # load the package  
data(whiteside) # load the data  
head(whiteside) # inspect the data

## Insul Temp Gas  
## 1 Before -0.8 7.2  
## 2 Before -0.7 6.9  
## 3 Before 0.4 6.4  
## 4 Before 2.5 6.0  
## 5 Before 2.9 5.8  
## 6 Before 3.2 5.8

We do the OLS estimation on the pre-insulation data with lm function, possibly the most important function in R.

lm.1 <- lm(Gas~Temp, data=whiteside[whiteside$Insul=='Before',]) # OLS estimation

Things to note:

* We used the tilde syntax Gas~Temp, reading "gas as linear function of temperature".
* The data argument tells R where to look for the variables Gas and Temp. We used Insul=='Before' to subset observations before the insulation.
* The result is assigned to the object lm.1.

Like any other language, spoken or programable, there are many ways to say the same thing. Some more elegant than others...

lm.1 <- lm(y=Gas, x=Temp, data=whiteside[whiteside$Insul=='Before',])   
lm.1 <- lm(y=whiteside[whiteside$Insul=='Before',]$Gas,x=whiteside[whiteside$Insul=='Before',]$Temp)   
lm.1 <- whiteside[whiteside$Insul=='Before',] %>% lm(Gas~Temp, data=.)

The output is an object of class lm.

class(lm.1)

## [1] "lm"

Objects of class lm are very complicated. They store a lot of information which may be used for inference, plotting, etc. The str function, short for "structure", shows us the various elements of the object.

str(lm.1)

## List of 12  
## $ coefficients : Named num [1:2] 6.854 -0.393  
## ..- attr(\*, "names")= chr [1:2] "(Intercept)" "Temp"  
## $ residuals : Named num [1:26] 0.0316 -0.2291 -0.2965 0.1293 0.0866 ...  
## ..- attr(\*, "names")= chr [1:26] "1" "2" "3" "4" ...  
## $ effects : Named num [1:26] -24.2203 -5.6485 -0.2541 0.1463 0.0988 ...  
## ..- attr(\*, "names")= chr [1:26] "(Intercept)" "Temp" "" "" ...  
## $ rank : int 2  
## $ fitted.values: Named num [1:26] 7.17 7.13 6.7 5.87 5.71 ...  
## ..- attr(\*, "names")= chr [1:26] "1" "2" "3" "4" ...  
## $ assign : int [1:2] 0 1  
## $ qr :List of 5  
## ..$ qr : num [1:26, 1:2] -5.099 0.196 0.196 0.196 0.196 ...  
## .. ..- attr(\*, "dimnames")=List of 2  
## .. .. ..$ : chr [1:26] "1" "2" "3" "4" ...  
## .. .. ..$ : chr [1:2] "(Intercept)" "Temp"  
## .. ..- attr(\*, "assign")= int [1:2] 0 1  
## ..$ qraux: num [1:2] 1.2 1.35  
## ..$ pivot: int [1:2] 1 2  
## ..$ tol : num 1e-07  
## ..$ rank : int 2  
## ..- attr(\*, "class")= chr "qr"  
## $ df.residual : int 24  
## $ xlevels : Named list()  
## $ call : language lm(formula = Gas ~ Temp, data = whiteside[whiteside$Insul == "Before", ])  
## $ terms :Classes 'terms', 'formula' language Gas ~ Temp  
## .. ..- attr(\*, "variables")= language list(Gas, Temp)  
## .. ..- attr(\*, "factors")= int [1:2, 1] 0 1  
## .. .. ..- attr(\*, "dimnames")=List of 2  
## .. .. .. ..$ : chr [1:2] "Gas" "Temp"  
## .. .. .. ..$ : chr "Temp"  
## .. ..- attr(\*, "term.labels")= chr "Temp"  
## .. ..- attr(\*, "order")= int 1  
## .. ..- attr(\*, "intercept")= int 1  
## .. ..- attr(\*, "response")= int 1  
## .. ..- attr(\*, ".Environment")=<environment: R\_GlobalEnv>   
## .. ..- attr(\*, "predvars")= language list(Gas, Temp)  
## .. ..- attr(\*, "dataClasses")= Named chr [1:2] "numeric" "numeric"  
## .. .. ..- attr(\*, "names")= chr [1:2] "Gas" "Temp"  
## $ model :'data.frame': 26 obs. of 2 variables:  
## ..$ Gas : num [1:26] 7.2 6.9 6.4 6 5.8 5.8 5.6 4.7 5.8 5.2 ...  
## ..$ Temp: num [1:26] -0.8 -0.7 0.4 2.5 2.9 3.2 3.6 3.9 4.2 4.3 ...  
## ..- attr(\*, "terms")=Classes 'terms', 'formula' language Gas ~ Temp  
## .. .. ..- attr(\*, "variables")= language list(Gas, Temp)  
## .. .. ..- attr(\*, "factors")= int [1:2, 1] 0 1  
## .. .. .. ..- attr(\*, "dimnames")=List of 2  
## .. .. .. .. ..$ : chr [1:2] "Gas" "Temp"  
## .. .. .. .. ..$ : chr "Temp"  
## .. .. ..- attr(\*, "term.labels")= chr "Temp"  
## .. .. ..- attr(\*, "order")= int 1  
## .. .. ..- attr(\*, "intercept")= int 1  
## .. .. ..- attr(\*, "response")= int 1  
## .. .. ..- attr(\*, ".Environment")=<environment: R\_GlobalEnv>   
## .. .. ..- attr(\*, "predvars")= language list(Gas, Temp)  
## .. .. ..- attr(\*, "dataClasses")= Named chr [1:2] "numeric" "numeric"  
## .. .. .. ..- attr(\*, "names")= chr [1:2] "Gas" "Temp"  
## - attr(\*, "class")= chr "lm"

In RStudio it is particularly easy to extract objects. Just write your.object$ and press tab after the $ for autocompletion.

If we only want , it can also be extracted with the coef function.

coef(lm.1)

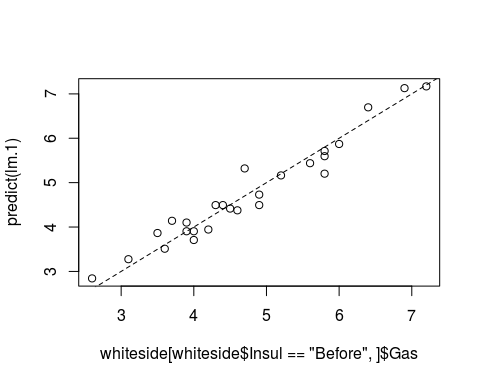
## (Intercept) Temp   
## 6.8538277 -0.3932388

Things to note:

* R automatically adds an (Intercept) term. This means we estimate and not . This makes sense because we are interested in the contribution of the temperature to the variability of the gas consumption about its **mean**, and not about zero.
* The effect of temperature, i.e., , is -0.39. The negative sign means that the higher the temperature, the less gas is consumed. The magnitude of the coefficient means that for a unit increase in the outside temperature, the gas consumption decreases by 0.39 units.

We can use the predict function to make predictions, but we emphasize that if the purpose of the model is to make predictions, and not interpret coefficients, better skip to the Supervised Learning Chapter 9.

plot(predict(lm.1)~whiteside[whiteside$Insul=='Before',]$Gas)  
abline(0,1, lty=2)



The model seems to fit the data nicely. A common measure of the goodness of fit is the *coefficient of determination*, more commonly known as the .

Definition 13 (R2) The coefficient of determination, denoted , is defined as

where is the model's prediction, .

It can be easily computed

R2 <- function(y, y.hat){  
 numerator <- (y-y.hat)^2 %>% sum  
 denominator <- (y-mean(y))^2 %>% sum  
 1-numerator/denominator  
}  
R2(y=whiteside[whiteside$Insul=='Before',]$Gas, y.hat=predict(lm.1))

## [1] 0.9438081

This is a nice result implying that about of the variability in gas consumption can be attributed to changes in the outside temperature.

Obviously, R does provide the means to compute something as basic as , but I will let you find it for yourselves.

## Inference

To perform inference on , in order to test hypotheses and construct confidence intervals, we need to quantify the uncertainly in the reported . This is exactly what Eq.(6) gives us.

Luckily, we don't need to manipulate multivariate distributions manually, and everything we need is already implemented. The most important function is summary which gives us an overview of the model's fit. We emphasize that that fitting a model with lm is an assumption free algorithmic step. Inference using summary is **not** assumption free, and requires the set of assumptions leading to Eq.(6).

summary(lm.1)

##   
## Call:  
## lm(formula = Gas ~ Temp, data = whiteside[whiteside$Insul ==   
## "Before", ])  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -0.62020 -0.19947 0.06068 0.16770 0.59778   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 6.85383 0.11842 57.88 <2e-16 \*\*\*  
## Temp -0.39324 0.01959 -20.08 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.2813 on 24 degrees of freedom  
## Multiple R-squared: 0.9438, Adjusted R-squared: 0.9415   
## F-statistic: 403.1 on 1 and 24 DF, p-value: < 2.2e-16

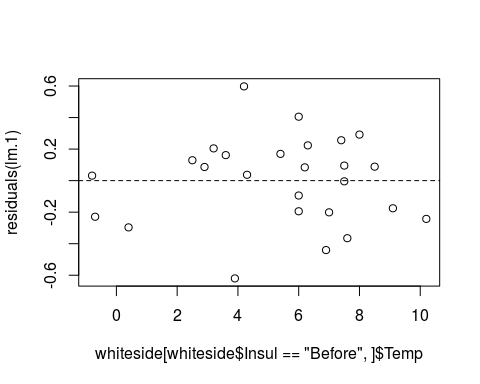
Things to note:

* The estimated is reported in the `Coefficients' table, which has point estimates, standard errors, t-statistics, and the p-values of a two-sided hypothesis test for each coefficient .
* The is reported at the bottom. The "Adjusted R-squared" is a variation that compensates for the model's complexity.
* The original call to lm is saved in the Call section.
* Some summary statistics of the residuals () in the Residuals section.
* The "residuals standard error"[[11]](#footnote-159) is . The denominator of this expression is the *degrees of freedom*, , which can be thought of as the hardness of the problem.

As the name suggests, summary is merely a summary. The full summary(lm.1) object is a monstrous object. Its various elements can be queried using str(sumary(lm.1)).

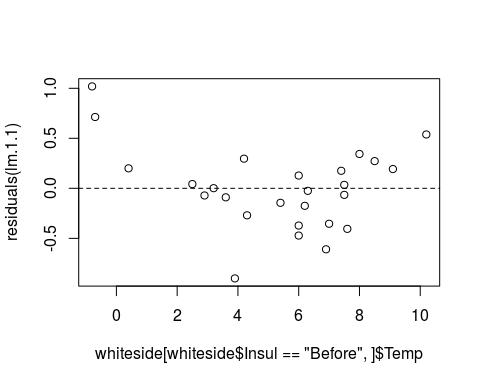
Can we check the assumptions required for inference? Some. Let's start with the linearity assumption. If we were wrong, and the data is not arranged about a linear line, the residuals will have some shape. We thus plot the residuals as a function of the predictor to diagnose shape.

plot(residuals(lm.1)~whiteside[whiteside$Insul=='Before',]$Temp)  
abline(0,0, lty=2)



I can't say I see any shape. Let's fit a **wrong** model, just to see what "shape" means.

lm.1.1 <- lm(Gas~I(Temp^2), data=whiteside[whiteside$Insul=='Before',])  
plot(residuals(lm.1.1)~whiteside[whiteside$Insul=='Before',]$Temp); abline(0,0, lty=2)



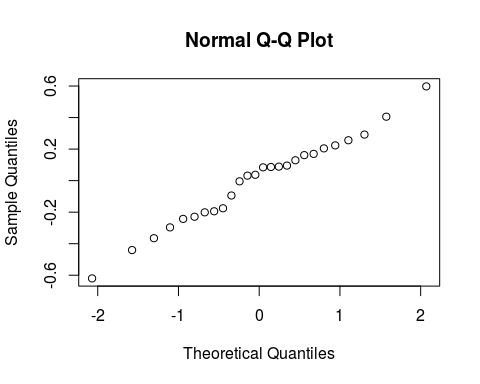
Things to note:

* We used I(Temp)^2 to specify the model .
* The residuals have a "belly". Because they are not a cloud around the linear trend, and we have the wrong model.

To the next assumption. We assumed are independent of everything else. The residuals, can be thought of a sample of . When diagnosing the linearity assumption, we already saw their distribution does not vary with the 's, Temp in our case. They may be correlated with themselves; a positive departure from the model, may be followed by a series of positive departures etc. Diagnosing these *auto-correlations* is a real art, which is not part of our course.

The last assumption we required is normality. As previously stated, if , this assumption can be relaxed. If is in the order of , we need to verify this assumption. My favorite tool for this task is the *qqplot*. A qqplot compares the quantiles of the sample with the respective quantiles of the assumed distribution. If quantiles align along a line, the assumed distribution if OK. If quantiles depart from a line, then the assumed distribution does not fit the sample.

qqnorm(resid(lm.1))

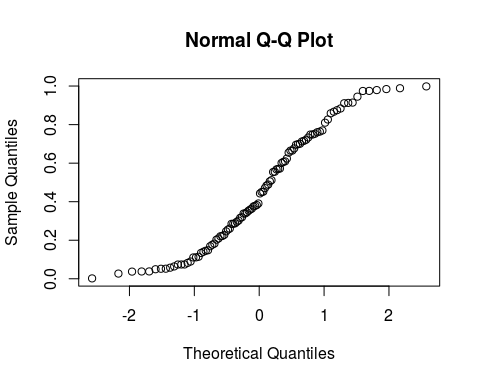


Things to note:

* The qqnorm function plots a qqplot against a normal distribution. For non-normal distributions try qqplot.
* resid(lm.1) extracts the residuals from the linear model, i.e., the vector of .

Judging from the figure, the normality assumption is quite plausible. Let's try the same on a non-normal sample, namely a uniformly distributed sample, to see how that would look.

qqnorm(runif(100))



### Testing a Hypothesis on a Single Coefficient

The first inferential test we consider is a hypothesis test on a single coefficient. In our gas example, we may want to test that the temperature has no effect on the gas consumption. The answer for that is given immediately by summary(lm.1)

summary.lm1 <- summary(lm.1)  
coefs.lm1 <- summary.lm1$coefficients  
coefs.lm1

## Estimate Std. Error t value Pr(>|t|)  
## (Intercept) 6.8538277 0.11842341 57.87561 2.717533e-27  
## Temp -0.3932388 0.01958601 -20.07754 1.640469e-16

We see that the p-value for against a two sided alternative is effectively 0, so that is unlikely to be .

### Constructing a Confidence Interval on a Single Coefficient

Since the summary function gives us the standard errors of , we can immediately compute to get ourselves a (roughly) confidence interval. In our example the interval is

coefs.lm1[2,1] + c(-2,2) \* coefs.lm1[2,2]

## [1] -0.4324108 -0.3540668

### Multiple Regression

Remark. *Multiple regression* is not to be confused with *multivariate regression* discussed in Chapter 8.

The swiss dataset encodes the fertility at each of Switzerland's 47 French speaking provinces, along other socio-economic indicators. Let's see if these are statistically related:

head(swiss)

## Fertility Agriculture Examination Education Catholic  
## Courtelary 80.2 17.0 15 12 9.96  
## Delemont 83.1 45.1 6 9 84.84  
## Franches-Mnt 92.5 39.7 5 5 93.40  
## Moutier 85.8 36.5 12 7 33.77  
## Neuveville 76.9 43.5 17 15 5.16  
## Porrentruy 76.1 35.3 9 7 90.57  
## Infant.Mortality  
## Courtelary 22.2  
## Delemont 22.2  
## Franches-Mnt 20.2  
## Moutier 20.3  
## Neuveville 20.6  
## Porrentruy 26.6

lm.5 <- lm(data=swiss, Fertility~Agriculture+Examination+Education+Education+Catholic+Infant.Mortality)  
summary(lm.5)

##   
## Call:  
## lm(formula = Fertility ~ Agriculture + Examination + Education +   
## Education + Catholic + Infant.Mortality, data = swiss)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -15.2743 -5.2617 0.5032 4.1198 15.3213   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 66.91518 10.70604 6.250 1.91e-07 \*\*\*  
## Agriculture -0.17211 0.07030 -2.448 0.01873 \*   
## Examination -0.25801 0.25388 -1.016 0.31546   
## Education -0.87094 0.18303 -4.758 2.43e-05 \*\*\*  
## Catholic 0.10412 0.03526 2.953 0.00519 \*\*   
## Infant.Mortality 1.07705 0.38172 2.822 0.00734 \*\*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 7.165 on 41 degrees of freedom  
## Multiple R-squared: 0.7067, Adjusted R-squared: 0.671   
## F-statistic: 19.76 on 5 and 41 DF, p-value: 5.594e-10

Things to note:

* The ~ syntax allows to specify various predictors separated by the + operator.
* The summary of the model now reports the estimated effect, i.e., the regression coefficient, of each of the variables.

Clearly, naming each variable explicitely is a tedios task if there are many. The use of Fertility~. in the next example reads: "Fertility as a function of all other variables in the swiss data.frame".

lm.5 <- lm(data=swiss, Fertility~.)  
summary(lm.5)

##   
## Call:  
## lm(formula = Fertility ~ ., data = swiss)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -15.2743 -5.2617 0.5032 4.1198 15.3213   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 66.91518 10.70604 6.250 1.91e-07 \*\*\*  
## Agriculture -0.17211 0.07030 -2.448 0.01873 \*   
## Examination -0.25801 0.25388 -1.016 0.31546   
## Education -0.87094 0.18303 -4.758 2.43e-05 \*\*\*  
## Catholic 0.10412 0.03526 2.953 0.00519 \*\*   
## Infant.Mortality 1.07705 0.38172 2.822 0.00734 \*\*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 7.165 on 41 degrees of freedom  
## Multiple R-squared: 0.7067, Adjusted R-squared: 0.671   
## F-statistic: 19.76 on 5 and 41 DF, p-value: 5.594e-10

### ANOVA (\*)

Our next example[[12]](#footnote-168) contains a hypothetical sample of participants who are divided into three stress reduction treatment groups (mental, physical, and medical) and two gender groups (male and female). The stress reduction values are represented on a scale that ranges from 1 to 5. The values represent how effective the treatment programs were at reducing participant's stress levels, with larger effects indicating higher effectiveness.

twoWay <- read.csv('data/dataset\_anova\_twoWay\_comparisons.csv')  
head(twoWay)

## Treatment Age StressReduction  
## 1 mental young 10  
## 2 mental young 9  
## 3 mental young 8  
## 4 mental mid 7  
## 5 mental mid 6  
## 6 mental mid 5

How many observations per group?

table(twoWay$Treatment, twoWay$Age)

##   
## mid old young  
## medical 3 3 3  
## mental 3 3 3  
## physical 3 3 3

Since we have two factorial predictors, this multiple regression is nothing but a *two way ANOVA*. Let's fit the model and inspect it.

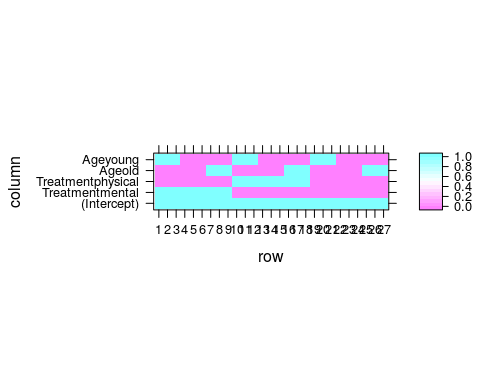
lm.2 <- lm(StressReduction~.,data=twoWay)  
summary(lm.2)

##   
## Call:  
## lm(formula = StressReduction ~ ., data = twoWay)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -1 -1 0 1 1   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 4.0000 0.3892 10.276 7.34e-10 \*\*\*  
## Treatmentmental 2.0000 0.4264 4.690 0.000112 \*\*\*  
## Treatmentphysical 1.0000 0.4264 2.345 0.028444 \*   
## Ageold -3.0000 0.4264 -7.036 4.65e-07 \*\*\*  
## Ageyoung 3.0000 0.4264 7.036 4.65e-07 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.9045 on 22 degrees of freedom  
## Multiple R-squared: 0.9091, Adjusted R-squared: 0.8926   
## F-statistic: 55 on 4 and 22 DF, p-value: 3.855e-11

Things to note:

* The StressReduction~. syntax is read as "Stress reduction as a function of everything else".
* All the (main) effects seem to be significant.
* The data has 2 factors, but the coefficients table has 4 predictors. This is because lm noticed that Treatment and Age are factors. Each level of each factor is thus encoded as a different (dummy) variable. The numerical values of the factors are meaningless. Instead, R has constructed a dummy variable for each level of each factor. The names of the effect are a concatenation of the factor's name, and its level. You can inspect these dummy variables with the model.matrix command.

model.matrix(lm.2) %>% lattice::levelplot()

 If you don't want the default dummy coding, look at ?contrasts.

If you are more familiar with the ANOVA literature, or that you don't want the effects of each level separately, but rather, the effect of **all** the levels of each factor, use the anova command.

anova(lm.2)

## Analysis of Variance Table  
##   
## Response: StressReduction  
## Df Sum Sq Mean Sq F value Pr(>F)   
## Treatment 2 18 9.000 11 0.0004883 \*\*\*  
## Age 2 162 81.000 99 1e-11 \*\*\*  
## Residuals 22 18 0.818   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Things to note:

* The ANOVA table, unlike the summary function, tests if **any** of the levels of a factor has an effect, and not one level at a time.
* The significance of each factor is computed using an F-test.
* The degrees of freedom, encoding the number of levels of a factor, is given in the Df column.
* The StressReduction seems to vary for different ages and treatments, since both factors are significant.

If you are extremely more comfortable with the ANOVA literature, you could have replaced the lm command with the aov command all along.

lm.2.2 <- aov(StressReduction~.,data=twoWay)  
class(lm.2.2)

## [1] "aov" "lm"

summary(lm.2.2)

## Df Sum Sq Mean Sq F value Pr(>F)   
## Treatment 2 18 9.00 11 0.000488 \*\*\*  
## Age 2 162 81.00 99 1e-11 \*\*\*  
## Residuals 22 18 0.82   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Things to note:

* The lm function has been replaced with an aov function.
* The output of aov is an aov class object, which extends the lm class.
* The summary of an aov does not like like the summary of an lm object, but rather, like an ANOVA table.

As in any two-way ANOVA, we may want to ask if different age groups respond differently to different treatments. In the statistical parlance, this is called an *interaction*, or more precisely, an *interaction of order 2*.

lm.3 <- lm(StressReduction~Treatment+Age+Treatment:Age-1,data=twoWay)

The syntax StressReduction~Treatment+Age+Treatment:Age-1 tells R to include main effects of Treatment, Age, and their interactions. Here are other ways to specify the same model.

lm.3 <- lm(StressReduction ~ Treatment \* Age - 1,data=twoWay)  
lm.3 <- lm(StressReduction~(.)^2 - 1,data=twoWay)

The syntax Treatment \* Age means "mains effects with second order interactions". The syntax (.)^2 means "everything with second order interactions"

Let's inspect the model

summary(lm.3)

##   
## Call:  
## lm(formula = StressReduction ~ Treatment + Age + Treatment:Age -   
## 1, data = twoWay)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -1 -1 0 1 1   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## Treatmentmedical 4.000e+00 5.774e-01 6.928 1.78e-06 \*\*\*  
## Treatmentmental 6.000e+00 5.774e-01 10.392 4.92e-09 \*\*\*  
## Treatmentphysical 5.000e+00 5.774e-01 8.660 7.78e-08 \*\*\*  
## Ageold -3.000e+00 8.165e-01 -3.674 0.00174 \*\*   
## Ageyoung 3.000e+00 8.165e-01 3.674 0.00174 \*\*   
## Treatmentmental:Ageold 4.246e-16 1.155e+00 0.000 1.00000   
## Treatmentphysical:Ageold 1.034e-15 1.155e+00 0.000 1.00000   
## Treatmentmental:Ageyoung -3.126e-16 1.155e+00 0.000 1.00000   
## Treatmentphysical:Ageyoung 5.128e-16 1.155e+00 0.000 1.00000   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 1 on 18 degrees of freedom  
## Multiple R-squared: 0.9794, Adjusted R-squared: 0.9691   
## F-statistic: 95 on 9 and 18 DF, p-value: 2.556e-13

Things to note:

* There are still main effects, but also interactions. This is because when allowing a different average response for every combination, we are effectively estimating cell means, even if they are not parametrized as cell means, but rather as main effect and interactions.
* The interactions do not seem to be significant.
* The assumptions required for inference are clearly not met in this example, which is there just to demonstrate R's capabilities.

Asking if all the interactions are significant, is asking if the different age groups have the same response to different treatments. Can we answer that based on the various interactions? We might, but it is possible that no single interaction is significant, while the combination is. To test for all the interactions together, we can simply check if the model without interactions is (significantly) better than a model with interactions. I.e., compare lm.2 to lm.3. This is done with the anova command.

anova(lm.2,lm.3, test='F')

## Analysis of Variance Table  
##   
## Model 1: StressReduction ~ Treatment + Age  
## Model 2: StressReduction ~ Treatment + Age + Treatment:Age - 1  
## Res.Df RSS Df Sum of Sq F Pr(>F)  
## 1 22 18   
## 2 18 18 4 -3.5527e-15

We see that lm.3 is **not** (significantly) better than lm.2, so that we can conclude that there are no interactions: different ages have the same response to different treatments.

### Testing a Hypothesis on a Single Contrast (\*) Returning to the model without interactions, lm.2.

coef(summary(lm.2))

## Estimate Std. Error t value Pr(>|t|)  
## (Intercept) 4 0.3892495 10.276186 7.336391e-10  
## Treatmentmental 2 0.4264014 4.690416 1.117774e-04  
## Treatmentphysical 1 0.4264014 2.345208 2.844400e-02  
## Ageold -3 0.4264014 -7.035624 4.647299e-07  
## Ageyoung 3 0.4264014 7.035624 4.647299e-07

We see that the effect of the various treatments is rather similar. It is possible that all treatments actually have the same effect. Comparing the effects of factor levels is called a *contrast*. Let's test if the medical treatment, has in fact, the same effect as the physical treatment.

library(multcomp)  
my.contrast <- matrix(c(-1,0,1,0,0), nrow = 1)  
lm.4 <- glht(lm.2, linfct=my.contrast)  
summary(lm.4)

##   
## Simultaneous Tests for General Linear Hypotheses  
##   
## Fit: lm(formula = StressReduction ~ ., data = twoWay)  
##   
## Linear Hypotheses:  
## Estimate Std. Error t value Pr(>|t|)   
## 1 == 0 -3.0000 0.7177 -4.18 0.000389 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
## (Adjusted p values reported -- single-step method)

Things to note:

* A contrast is a linear function of the coefficients. In our example , which justifies the construction of my.contrast.
* We used the glht function (generalized linear hypothesis test) from the package **multcompt**.
* The contrast is significant, i.e., the effect of a medical treatment, is different than that of a physical treatment.

## Bibliographic Notes

Like any other topic in this book, you can consult Venables and Ripley ([2013](#ref-venables2013modern)) for more on linear models. For the theory of linear models, I like Greene ([2003](#ref-greene2003econometric)).

## Practice Yourself

1. Inspect women's heights and weights with ?women.
   1. What is the change in weight per unit change in height? Use the lm function.
   2. Is the relation of height on weight significant? Use summary.
   3. Plot the residuals of the linear model with plot and resid.
   4. Plot the predictions of the model using abline.
   5. Inspect the normality of residuals using qqnorm.
   6. Inspect the design matrix using model.matrix.
2. Write a function that takes an lm class object, and returns the confidence interval on the first coefficient. Apply it on the height and weight data.
3. Use the ANOVA function to test the significance of the effect of height.

# Generalized Linear Models

Example 6 Consider the relation between cigarettes smoked, and the occurance of lung cancer. Do we expect the probability of cancer to be linear in the number of cigarettes? Probably not. Do we expect the variability of events to be constant about the trend? Probably not.

Example 7 Consider the relation between the length of the observation period, and the number of cars passing a cross-roads. Do you agree that the longer the period, the more cars are expected, but also more variability is expected?

## Problem Setup

In the Linear Models Chapter 5, we assumed the generative process to be

where the notation is read "the response given the predictors ".

This does not allow for the non-linear relations of Example 6, nor does it allow for the distrbituion of to change with , as in Example 7. *Generalize linear models* (GLM), as the name suggests, are a generalization of the linear models in Chapter 5 that allow that[[13]](#footnote-176).

To understand GLM, we recall that with the normality of , Eq.(7) implies that

For Example 6, we would like something in the lines of

For Example 7, we would like something in the lines of

More generally, for some distribution , with a parameter , we would like

Possible examples include

GLMs allow models of the type of Eq.(8), while imposing some constraints on and on the relation . GLMs assume the data distribution to be in a "well-behaved" family known as the [*Natural Exponential Family*](https://en.wikipedia.org/wiki/Natural_exponential_family) of distributions. GLMs also assume that is a rescaling of a linear combination of s. Formally

where

The function is called the *link* function, its inverse, is the *mean function*. This terminology will later be required to understand R's output.

## Logistic Regression

The best known of the GLM class of models is the *logistic regression* that deals with Binomial, or more precisely, Bernoulli-distributed data. The link function in the logistic regression is the *logit function*

implying that

Before we fit such a model, we try to justify this construction, in particular, the enigmatic link function in Eq.(9). Let's look at the simplest possible case: the comparison of two groups indexed by : for the first, and for the second. We start with some definitions.

Definition 14 (Odds) The *odds*, of a binary random variable, , is defined as

Odds are the same as probabilities, but instead of of telling me there is a of success, they tell me the odds of success are "2 to 1". If you ever placed a bet, the language of "odds" should not be unfamiliar to you.

Definition 15 (Odds Ratio) The *odds ratio* between two binary random variables, and , is defined as the ratio between their odds. Formally:

Odds ratios (OR) compare between the probabilities of two groups, only that it does not compare them in probability scale, but rather in odds scale. You can also think of ORs as a measure of distance between two Brenoulli distributions. ORs have better mathematical properties than other candidate distance measures, such as .

Under the logit link assumption, the OR between two conditions indexed by and , returns:

The last equality demystifies the choice of the link function in the logistic regression: **it allows us to interpret of the logistic regression as a measure of change of binary random variables, namely, as the (log) odds-ratios due to a unit increase in** .

Remark. Another popular link function is the normal quantile function, a.k.a., the Gaussian inverse CDF, leading to *probit regression* instead of logistic regression.

### Logistic Regression with R

Let's get us some data. The PlantGrowth data records the weight of plants under three conditions: control, treatment1, and treatment2.

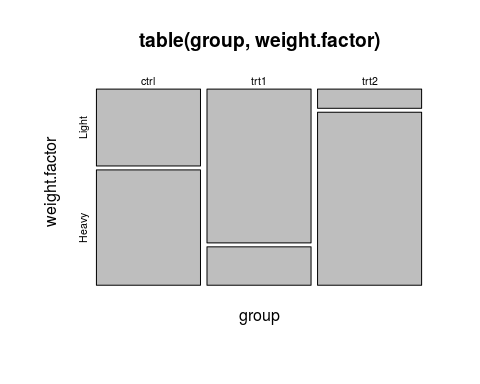
head(PlantGrowth)

## weight group  
## 1 4.17 ctrl  
## 2 5.58 ctrl  
## 3 5.18 ctrl  
## 4 6.11 ctrl  
## 5 4.50 ctrl  
## 6 4.61 ctrl

We will now attach the data so that its contents is available in the workspace (don't forget to detach afterwards, or you can expect some conflicting object names). We will also use the cut function to create a binary response variable for Light, and Heavy plants (we are doing logistic regression, so we need a two-class response). As a general rule of thumb, when we discretize continuous variables, we lose information. For pedagogical reasons, however, we will proceed with this bad practice.

Look at the following output and think: how many group effects do we expect? What should be the sign of each effect?

attach(PlantGrowth)  
weight.factor<- cut(weight, 2, labels=c('Light', 'Heavy')) # binarize weights  
plot(table(group, weight.factor))



Let's fit a logistic regression, and inspect the output.

glm.1<- glm(weight.factor~group, family=binomial)  
summary(glm.1)

##   
## Call:  
## glm(formula = weight.factor ~ group, family = binomial)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.1460 -0.6681 0.4590 0.8728 1.7941   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 0.4055 0.6455 0.628 0.5299   
## grouptrt1 -1.7918 1.0206 -1.756 0.0792 .  
## grouptrt2 1.7918 1.2360 1.450 0.1471   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 41.054 on 29 degrees of freedom  
## Residual deviance: 29.970 on 27 degrees of freedom  
## AIC: 35.97  
##   
## Number of Fisher Scoring iterations: 4

Things to note:

* The glm function is our workhorse for all GLM models.
* The family argument of glm tells R the respose variable is brenoulli, thus, performing a logistic regression.
* The summary function is content aware. It gives a different output for glm class objects than for other objects, such as the lm we saw in Chapter 5. In fact, what summary does is merely call summary.glm.
* As usual, we get the coefficients table, but recall that they are to be interpreted as (log) odd-ratios.
* As usual, we get the significance for the test of no-effect, versus a two-sided alternative. P-values are asymptotic, thus, only approximate (and can be very bad approximations in small samples).
* The residuals of glm are slightly different than the lm residuals, and called *Deviance Residuals*.
* For help see ?glm, ?family, and ?summary.glm.

Like in the linear models, we can use an ANOVA table to check if treatments have any effect, and not one treatment at a time. In the case of GLMs, this is called an *analysis of deviance* table.

anova(glm.1, test='LRT')

## Analysis of Deviance Table  
##   
## Model: binomial, link: logit  
##   
## Response: weight.factor  
##   
## Terms added sequentially (first to last)  
##   
##   
## Df Deviance Resid. Df Resid. Dev Pr(>Chi)   
## NULL 29 41.054   
## group 2 11.084 27 29.970 0.003919 \*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Things to note:

* The anova function, like the summary function, are content-aware and produce a different output for the glm class than for the lm class. All that anova does is call anova.glm.
* In GLMs there is no canonical test (like the F test for lm). LRT implies we want an approximate Likelihood Ratio Test. We thus specify the type of test desired with the test argument.
* The distribution of the weights of the plants does vary with the treatment given, as we may see from the significance of the group factor.
* Readers familiar with ANOVA tables, should know that we computed the GLM equivalent of a type I sum- of-squares. Run drop1(glm.1, test='Chisq') for a GLM equivalent of a type III sum-of-squares.
* For help see ?anova.glm.

Let's predict the probability of a heavy plant for each treatment.

predict(glm.1, type='response')

## 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18   
## 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2   
## 19 20 21 22 23 24 25 26 27 28 29 30   
## 0.2 0.2 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9

Things to note:

* Like the summary and anova functions, the predict function is aware that its input is of glm class. All that predict does is call predict.glm.
* In GLMs there are many types of predictions. The type argument controls which type is returned.
* How do I know we are predicting the probability of a heavy plant, and not a light plant? Just run contrasts(weight.factor) to see which of the categories of the factor weight.factor is encoded as 1, and which as 0.
* For help see ?predict.glm.

Let's detach the data so it is no longer in our workspace, and object names do not collide.

detach(PlantGrowth)

We gave an example with a factorial (i.e. discrete) predictor. We can do the same with multiple continuous predictors.

data('Pima.te', package='MASS') # Loads data  
head(Pima.te)

## npreg glu bp skin bmi ped age type  
## 1 6 148 72 35 33.6 0.627 50 Yes  
## 2 1 85 66 29 26.6 0.351 31 No  
## 3 1 89 66 23 28.1 0.167 21 No  
## 4 3 78 50 32 31.0 0.248 26 Yes  
## 5 2 197 70 45 30.5 0.158 53 Yes  
## 6 5 166 72 19 25.8 0.587 51 Yes

glm.2<- step(glm(type~., data=Pima.te, family=binomial))

## Start: AIC=301.79  
## type ~ npreg + glu + bp + skin + bmi + ped + age  
##   
## Df Deviance AIC  
## - skin 1 286.22 300.22  
## - bp 1 286.26 300.26  
## - age 1 286.76 300.76  
## <none> 285.79 301.79  
## - npreg 1 291.60 305.60  
## - ped 1 292.15 306.15  
## - bmi 1 293.83 307.83  
## - glu 1 343.68 357.68  
##   
## Step: AIC=300.22  
## type ~ npreg + glu + bp + bmi + ped + age  
##   
## Df Deviance AIC  
## - bp 1 286.73 298.73  
## - age 1 287.23 299.23  
## <none> 286.22 300.22  
## - npreg 1 292.35 304.35  
## - ped 1 292.70 304.70  
## - bmi 1 302.55 314.55  
## - glu 1 344.60 356.60  
##   
## Step: AIC=298.73  
## type ~ npreg + glu + bmi + ped + age  
##   
## Df Deviance AIC  
## - age 1 287.44 297.44  
## <none> 286.73 298.73  
## - npreg 1 293.00 303.00  
## - ped 1 293.35 303.35  
## - bmi 1 303.27 313.27  
## - glu 1 344.67 354.67  
##   
## Step: AIC=297.44  
## type ~ npreg + glu + bmi + ped  
##   
## Df Deviance AIC  
## <none> 287.44 297.44  
## - ped 1 294.54 302.54  
## - bmi 1 303.72 311.72  
## - npreg 1 304.01 312.01  
## - glu 1 349.80 357.80

summary(glm.2)

##   
## Call:  
## glm(formula = type ~ npreg + glu + bmi + ped, family = binomial,   
## data = Pima.te)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.9845 -0.6462 -0.3661 0.5977 2.5304   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -9.552177 1.096207 -8.714 < 2e-16 \*\*\*  
## npreg 0.178066 0.045343 3.927 8.6e-05 \*\*\*  
## glu 0.037971 0.005442 6.978 3.0e-12 \*\*\*  
## bmi 0.084107 0.021950 3.832 0.000127 \*\*\*  
## ped 1.165658 0.444054 2.625 0.008664 \*\*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 420.30 on 331 degrees of freedom  
## Residual deviance: 287.44 on 327 degrees of freedom  
## AIC: 297.44  
##   
## Number of Fisher Scoring iterations: 5

Things to note:

* We used the ~. syntax to tell R to fit a model with all the available predictors.
* Since we want to focus on significant predictors, we used the step function to perform a *step-wise* regression, i.e. sequentially remove non-significant predictors. The function reports each model it has checked, and the variable it has decided to remove at each step.
* The output of step is a single model, with the subset of selected predictors.

## Poisson Regression

Poisson regression means we fit a model assuming . Put differently, we assume that for each treatment, encoded as a combinations of predictors , the response is Poisson distributed with a rate that depends on the predictors.

The typical link function for Poisson regression is . This means that we assume . Why is this a good choice? We again resort to the two-group case, encoded by and , to understand this model: . We thus see that this link function implies that a change in **multiples** the rate of events by .

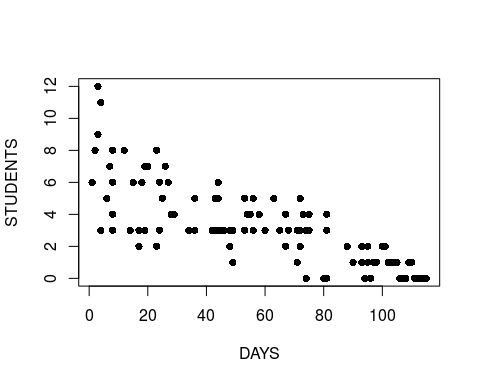
For our example[[14]](#footnote-184) we inspect the number of infected high-school kids, as a function of the days since an outbreak.

cases <-   
structure(list(Days = c(1L, 2L, 3L, 3L, 4L, 4L, 4L, 6L, 7L, 8L,   
8L, 8L, 8L, 12L, 14L, 15L, 17L, 17L, 17L, 18L, 19L, 19L, 20L,   
23L, 23L, 23L, 24L, 24L, 25L, 26L, 27L, 28L, 29L, 34L, 36L, 36L,   
42L, 42L, 43L, 43L, 44L, 44L, 44L, 44L, 45L, 46L, 48L, 48L, 49L,   
49L, 53L, 53L, 53L, 54L, 55L, 56L, 56L, 58L, 60L, 63L, 65L, 67L,   
67L, 68L, 71L, 71L, 72L, 72L, 72L, 73L, 74L, 74L, 74L, 75L, 75L,   
80L, 81L, 81L, 81L, 81L, 88L, 88L, 90L, 93L, 93L, 94L, 95L, 95L,   
95L, 96L, 96L, 97L, 98L, 100L, 101L, 102L, 103L, 104L, 105L,   
106L, 107L, 108L, 109L, 110L, 111L, 112L, 113L, 114L, 115L),   
 Students = c(6L, 8L, 12L, 9L, 3L, 3L, 11L, 5L, 7L, 3L, 8L,   
 4L, 6L, 8L, 3L, 6L, 3L, 2L, 2L, 6L, 3L, 7L, 7L, 2L, 2L, 8L,   
 3L, 6L, 5L, 7L, 6L, 4L, 4L, 3L, 3L, 5L, 3L, 3L, 3L, 5L, 3L,   
 5L, 6L, 3L, 3L, 3L, 3L, 2L, 3L, 1L, 3L, 3L, 5L, 4L, 4L, 3L,   
 5L, 4L, 3L, 5L, 3L, 4L, 2L, 3L, 3L, 1L, 3L, 2L, 5L, 4L, 3L,   
 0L, 3L, 3L, 4L, 0L, 3L, 3L, 4L, 0L, 2L, 2L, 1L, 1L, 2L, 0L,   
 2L, 1L, 1L, 0L, 0L, 1L, 1L, 2L, 2L, 1L, 1L, 1L, 1L, 0L, 0L,   
 0L, 1L, 1L, 0L, 0L, 0L, 0L, 0L)), .Names = c("Days", "Students"  
), class = "data.frame", row.names = c(NA, -109L))  
attach(cases)  
head(cases)

## Days Students  
## 1 1 6  
## 2 2 8  
## 3 3 12  
## 4 3 9  
## 5 4 3  
## 6 4 3

Look at the following plot and think: what is the sign of the effect of time on the number of sick students?

plot(Days, Students, xlab = "DAYS", ylab = "STUDENTS", pch = 16)



We now fit a model to check for the change in the rate of events as a function of the days since the outbreak.

glm.3 <- glm(Students ~ Days, family = poisson)  
summary(glm.3)

##   
## Call:  
## glm(formula = Students ~ Days, family = poisson)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.00482 -0.85719 -0.09331 0.63969 1.73696   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 1.990235 0.083935 23.71 <2e-16 \*\*\*  
## Days -0.017463 0.001727 -10.11 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for poisson family taken to be 1)  
##   
## Null deviance: 215.36 on 108 degrees of freedom  
## Residual deviance: 101.17 on 107 degrees of freedom  
## AIC: 393.11  
##   
## Number of Fisher Scoring iterations: 5

Things to note:

* We used family=poisson in the glm function to tell R that we assume a Poisson distribution.
* The coefficients table is there as usual. When interpreting the table, we need to recall that the effect, i.e. the , are **multiplicative** due to the assumed link function.
* Each day **decreases** the rate of events by a factor of about 0.02.
* For more information see ?glm and ?family.

## Extensions

As we already implied, GLMs are a very wide class of models. We do not need to use the default link function,but more importantly, we are not constrained to Binomial, or Poisson distributed response. For exponential, gamma, and other response distributions, see ?glm or the references in the Bibliographic Notes section.

## Bibliographic Notes

The ultimate reference on GLMs is McCullagh ([1984](#ref-mccullagh1984generalized)). For a less technical exposition, we refer to the usual Venables and Ripley ([2013](#ref-venables2013modern)).

## Practice Yourself

1. Try using lm for analyzing the plant growth data in weight.factor as a function of group in the PlantGrowth data.
2. Generate some synthetic data for a logistic regression:
   1. Generate two predictor variables of length . They can be random from your favorite distribution.
   2. Fix beta<- c(-1,2), and generate the response with:rbinom(n=100,size=1,prob=exp(x %\*% beta)/(1+exp(x %\*% beta))). Think: why is this the model implied by the logistic regression?
   3. Fit a Logistic regression to your synthetic data using glm.
   4. Are the estimated coefficients similar to the true ones you used?
   5. What is the estimated probability of an event at x=1,1? Use predict.glm but make sure to read the documentation on the type argument.
3. Read about the Acute Myelogenous Leukemia survival data with ?survival::aml.
   1. Use glm to estimate the effect of the maintenance chemotherapy on survival times. Assuming times are exponentially distributed, and recalling that the Gamma distribution generalizes the exponential, use family=Gamma.

# Linear Mixed Models

Example 8 (Fixed and Random Machine Effect) Consider the problem of testing for a change in the distribution of manufactured bottle caps. Bottle caps are produced by several machines. We could standardize over machines by removing each machine's average. This implies the within-machine variability is the only source of variability we care about. Alternatively, we could ignore the machine of origin. This second practice implies there are two sources of variability we care about: the within-machine variability, and the between-machine variability. The former practice is known as a *fixed effects* model. The latter as a *random effects* model.

Example 9 (Fixed and Random Subject Effect) Consider a crossover[[15]](#footnote-191) experimenal design where each subject is given 2 types of diets, and his health condition is recorded. We could standardize over subjects by removing the subject-wise average, before comparing diets. This is what a paired t-test does, and implies the within-subject variability is the only source of variability we care about. Alternatively, we could ignore the subject of origin. This second practice implies there are two sources of variability we care about: the within-subject variability and the betwee-subject variability.

The unifying theme of the above two examples, is that the variability we want to infer against has several sources. Which are the sources of variability that need to concern us? It depends on your purpose...

Mixed models are so fundamental, that they have earned many names:

* **Mixed Effects**: Because we may have both *fixed effects* we want to estimate and remove, and *random effects* which contribute to the variability to infer against.
* **Variance Components**: Because as the examples show, variance has more than a single source (like in the Linear Models of Chapter 5).
* **Hirarchial Models**: Because as Example 9 demonstrates, we can think of the sampling as hierarchical-- first sample a subject, and then sample its response.
* **Repeated Measures**: Because we make several measurements from each unit, like in Example 9.
* **Longitudinal Data**: Because we follow units over time, like in Example 9.
* **Panel Data**: Is the term typically used in econometric for such longitudinal data.
* **MANOVA**: Many of the problems that may be solved with a multivariate analysis of variance (MANOVA), may be solved with a mixed model for reasons we detail in 8.

We now emphasize:

1. Mixed effect models are a way to infer against the right level of variability. Using a naive linear model (which assumes a single source of variability) instead of a mixed effects model, probably means your inference is overly anti-conservative. Put differently, the uncertainty in your estimates is higher than the linear model from Chapter 5 may suggest.
2. A mixed effect models, as we will later see, is typically specified via its fixed and random effects. It is possible, however, to specify a mixed effects model by putting all the fixed effects into a linear model, and putting all the random effects into the covariance matrix of . This is known as *multivariate regression*, or *multivariate analysis of variance* (MANOVA). For more on this view, see Chapter 8 in (the excellent) Weiss ([2005](#ref-weiss2005modeling)).
3. Like in previous chapters, by "model" we refer to the assumed generative distribution, i.e., the sampling distribution.
4. If you are using the model merely for predictions, and not for inference on the fixed effects or variance components, then stating the generative distribution may be be useful, but not necessarily. See the Supervised Learning Chapter 9 for more on prediction problems. Also recall that machine learning from non-independent observations (such as mixed models) is a delicate matter that is rarely treated in the literature.

## Problem Setup

where are the factors with fixed effects, , which we may want to study. The factors , with effects , are the random effects which contribute to variability. In our diet exaple (9)

The reader may notice that we state merely as a convenient way to do inference on , instead of directly specifying .

Given a sample of observations from model (11), we will want to estimate . Under some assumption on the distribution of and , we can use *maximum likelihood* (ML). In the context of mixed-models, however, ML is typically replaced with *restricted maximum likelihood* (ReML), because it returns unbiased estimates of and ML does not.

### Relation to MANOVA (\*) {#manova} Multivariate analysis of variance (MANOVA) deals with the estimation of effect on vector valued outcomes. Put differently: in ANOVA the response, , is univariate In MANOVA, the outcome is multivariate. MANOVA is useful when there are correlations among the entries of . Otherwise- one may simply solve many ANOVA problems, instead of a single MANOVA.

Now assume that the outcome of a MANOVA is measurements of an individual at several time periods. The measurements are clearly correlated, so that MANOVA may be useful. But one may also treat time as a random effect, with a univariate response. We thus see that this seemingly MANOVA problem can be solved with the mixed models framework.

What MANOVA problems cannot be solved with mixed models? There may be cases where the covariance of the multivariate outcome, , is very complicated. If the covariance in may not be stated using a combination of random and fixed effects, then the covariance has to be stated explicitly in the MANOVA framework. It is also possible to consider mixed-models with multivariate outcomes, i.e., a *mixed MANOVA*, or *hirarchial MANOVA*. The R functions we present herein permit this.

## Mixed Models with R

We will fit mixed models with the lmer function from the **lme4** package, written by the mixed-models Guru [Douglas Bates](http://www.stat.wisc.edu/~bates/). We start with a small simulation demonstrating the importance of acknowledging your sources of variability, by fitting a linear model when a mixed model is appropriate. We start by creating some synthetic data.

n.groups <- 10  
n.repeats <- 2  
groups <- gl(n = n.groups, k = n.repeats) # create the group factor  
n <- length(groups)  
z0 <- rnorm(10,0,10) # generate group effects  
z <- z0[as.numeric(groups)] # assign effect to group  
epsilon <- rnorm(n,0,1) # generate measurement error  
beta0 <- 2 # set global mean  
y <- beta0 + z + epsilon # generate synthetic sample

We can now fit the linear and mixed models.

lm.5 <- lm(y~z) # fit a linear model  
library(lme4)  
lme.5 <- lmer(y~1|z) # fit a mixed-model

The summary of the linear model

summary.lm.5 <- summary(lm.5)  
summary.lm.5

##   
## Call:  
## lm(formula = y ~ z)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -2.48995 -0.51233 -0.04173 0.47603 2.05384   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 2.10630 0.23626 8.915 5.07e-08 \*\*\*  
## z 1.00652 0.02174 46.294 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 1.053 on 18 degrees of freedom  
## Multiple R-squared: 0.9917, Adjusted R-squared: 0.9912   
## F-statistic: 2143 on 1 and 18 DF, p-value: < 2.2e-16

The summary of the mixed-model

summary.lme.5 <- summary(lme.5)  
summary.lme.5

## Linear mixed model fit by REML ['lmerMod']  
## Formula: y ~ 1 | z  
##   
## REML criterion at convergence: 109.2  
##   
## Scaled residuals:   
## Min 1Q Median 3Q Max   
## -1.42573 -0.34773 -0.02314 0.38816 1.25231   
##   
## Random effects:  
## Groups Name Variance Std.Dev.  
## z (Intercept) 131.815 11.481   
## Residual 1.227 1.108   
## Number of obs: 20, groups: z, 10  
##   
## Fixed effects:  
## Estimate Std. Error t value  
## (Intercept) 1.195 3.639 0.328

Look at the standard error of the global mean, i.e., the intercept: for lm it is 0.2362628, and for lme it is 3.6390688. Why this difference? Because lm discounts the group effect[[16]](#footnote-197), while it should treat it as another source of variability. Clearly, inference using lm underestimates our uncertainty.

### A Single Random Effect

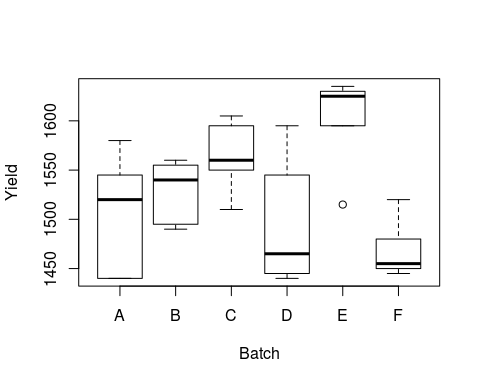
We will use the Dyestuff data from the **lme4** package, which encodes the yield, in grams, of a coloring solution (dyestuff), produced in 6 batches using 5 different preparations.

data(Dyestuff, package='lme4')  
attach(Dyestuff)  
head(Dyestuff)

## Batch Yield  
## 1 A 1545  
## 2 A 1440  
## 3 A 1440  
## 4 A 1520  
## 5 A 1580  
## 6 B 1540

And visually

plot(Yield~Batch)



If we want to do inference on the mean yield, we need to account for the two sources of variability: the batch effect, and the measurement error. We thus fit a mixed model, with an intercept and random batch effect, which means this is it not a bona-fide mixed-model, but rather, a simple random-effect model.

lme.1<- lmer( Yield ~ 1 | Batch , Dyestuff )  
summary(lme.1)

## Linear mixed model fit by REML ['lmerMod']  
## Formula: Yield ~ 1 | Batch  
## Data: Dyestuff  
##   
## REML criterion at convergence: 319.7  
##   
## Scaled residuals:   
## Min 1Q Median 3Q Max   
## -1.4117 -0.7634 0.1418 0.7792 1.8296   
##   
## Random effects:  
## Groups Name Variance Std.Dev.  
## Batch (Intercept) 1764 42.00   
## Residual 2451 49.51   
## Number of obs: 30, groups: Batch, 6  
##   
## Fixed effects:  
## Estimate Std. Error t value  
## (Intercept) 1527.50 19.38 78.8

Things to note:

* As usual, summary is content aware and has a different behavior for lme class objects.
* The syntax Yield ~ 1 | Batch tells R to fit a model with a global intercept (1) and a random Batch effect (|Batch). More on that later.
* The output distinguishes between random effects, a source of variability, and fixed effect, which's coefficients we want to study. The mean of the random effect is not reported because it is assumigly 0 for all random effects.
* Were we not interested in the variance components, and only in the coefficients or predictions, an (almost) equivalent lm formulation is lm(Yield ~ Batch).

Some utility functions let us query the lme object. The function coef will work, but will return a cumbersome output. Better use fixef to extract the fixed effects, and ranef to extract the random effects. The model matrix (of the fixed effects alone), can be extracted with model.matrix, and predictions made with predict. Note, however, that predictions with mixed-effect models are (i) a delicate matter, and (ii) better treated as prediction problems as in the Supervised Learning Chapter 9.

detach(Dyestuff)

### Multiple Random Effects

Let's make things more interesting by allowing more than one random effect. One-way ANOVA can be thought of as the fixed-effects counterpart of the single random effect. Our next example, involving two random effects, can be though of as the Two-way ANOVA counterpart.

In the Penicillin data, we measured the diameter of spread of an organism, along the plate used (a to x), and penicillin type (A to F).

head(Penicillin)

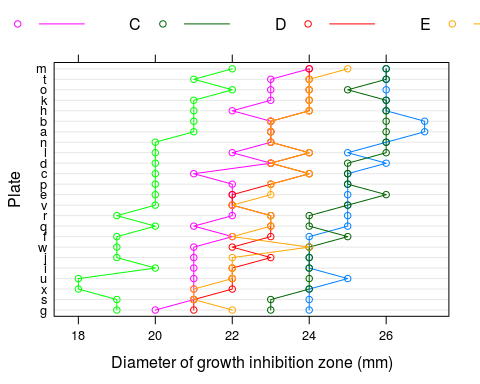
## diameter plate sample  
## 1 27 a A  
## 2 23 a B  
## 3 26 a C  
## 4 23 a D  
## 5 23 a E  
## 6 21 a F

One sample per combination:

attach(Penicillin)  
table(sample, plate) # how many observations per plate & type?

## plate  
## sample a b c d e f g h i j k l m n o p q r s t u v w x  
## A 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## B 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## C 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## D 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## E 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## F 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

And visually:



Let's fit a mixed-effects model with a random plate effect, and a random sample effect:

lme.2 <- lmer ( diameter ~ 1+ (1| plate ) + (1| sample ) , Penicillin )  
fixef(lme.2) # Fixed effects

## (Intercept)   
## 22.97222

ranef(lme.2) # Random effects

## $plate  
## (Intercept)  
## a 0.80454704  
## b 0.80454704  
## c 0.18167191  
## d 0.33739069  
## e 0.02595313  
## f -0.44120322  
## g -1.37551591  
## h 0.80454704  
## i -0.75264078  
## j -0.75264078  
## k 0.96026582  
## l 0.49310948  
## m 1.42742217  
## n 0.49310948  
## o 0.96026582  
## p 0.02595313  
## q -0.28548443  
## r -0.28548443  
## s -1.37551591  
## t 0.96026582  
## u -0.90835956  
## v -0.28548443  
## w -0.59692200  
## x -1.21979713  
##   
## $sample  
## (Intercept)  
## A 2.18705797  
## B -1.01047615  
## C 1.93789946  
## D -0.09689497  
## E -0.01384214  
## F -3.00374417

Things to note:

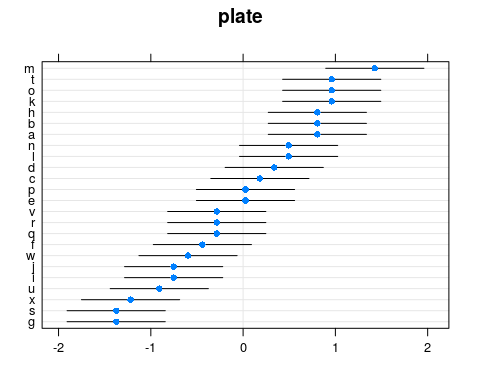
* The syntax 1+ (1| plate ) + (1| sample ) fits a global intercept (mean), a random plate effect, and a random sample effect.
* Were we not interested in the variance components, an (almost) equivalent lm formulation is lm(diameter ~ plate + sample).
* The output of ranef is somewhat controvertial. Think about it: Why would we want to plot the estimates of a random variable?

Since we have two random effects, we may compute the variability of the global mean (the only fixed effect) as we did before. Perhaps more interestingly, we can compute the variability in the response, for a particular plate or sample type.

random.effect.lme2 <- ranef(lme.2, condVar = TRUE)   
qrr2 <- lattice::dotplot(random.effect.lme2, strip = FALSE)

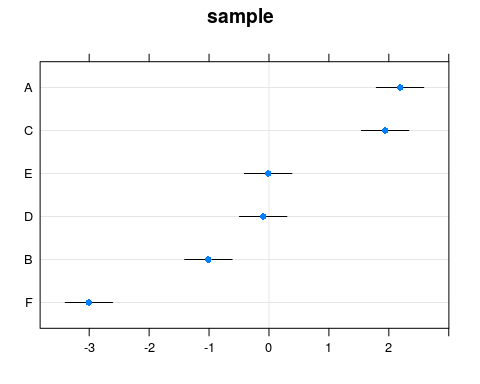
Variability in response for each plate, over various sample types:

print(qrr2[[1]])



Variability in response for each sample type, over the various plates:

print(qrr2[[2]])

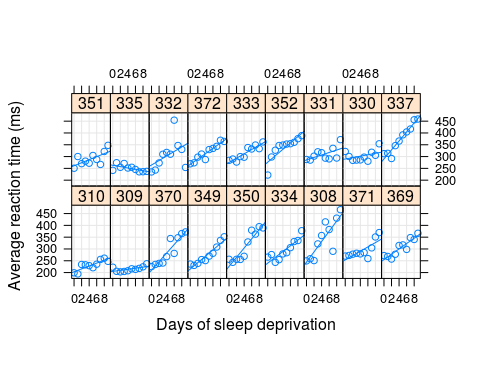


Things to note:

* The condVar argument of the ranef function tells R to compute the variability in response conditional on each random effect at a time.
* The dotplot function, from the **lattice** package, is only there for the fancy plotting.

### A Full Mixed-Model

In the sleepstudy data, we recorded the reaction times to a series of tests (Reaction), after various subject (Subject) underwent various amounts of sleep deprivation (Day).



We now want to estimate the (fixed) effect of the days of sleep deprivation on response time, while allowing each subject to have his/hers own effect. Put differently, we want to estimate a *random slope* for the effect of day. The fixed Days effect can be thought of as the average slope over subjects.

lme.3 <- lmer ( Reaction ~ Days + ( Days | Subject ) , data= sleepstudy )

Things to note:

* ~Days specifies the fixed effect.
* We used the Days|Subect syntax to tell R we want to fit the model ~Days within each subject.
* Were we fitting the model for purposes of prediction only, an (almost) equivalent lm formulation is lm(Reaction~Days\*Subject).

The fixed day effect is:

fixef(lme.3)

## (Intercept) Days   
## 251.40510 10.46729

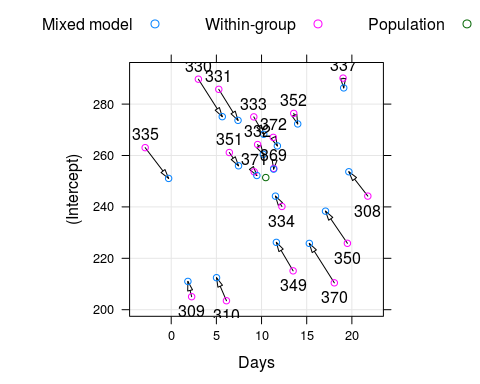
The variability in the average response (intercept) and day effect is

ranef(lme.3)

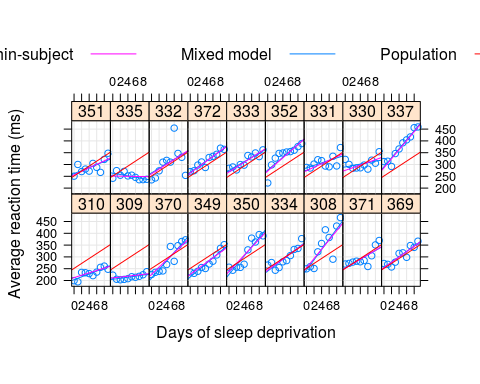
## $Subject  
## (Intercept) Days  
## 308 2.2585654 9.1989719  
## 309 -40.3985770 -8.6197032  
## 310 -38.9602459 -5.4488799  
## 330 23.6904985 -4.8143313  
## 331 22.2602027 -3.0698946  
## 332 9.0395259 -0.2721707  
## 333 16.8404312 -0.2236244  
## 334 -7.2325792 1.0745761  
## 335 -0.3336959 -10.7521591  
## 337 34.8903509 8.6282839  
## 349 -25.2101104 1.1734143  
## 350 -13.0699567 6.6142050  
## 351 4.5778352 -3.0152572  
## 352 20.8635925 3.5360133  
## 369 3.2754530 0.8722166  
## 370 -25.6128694 4.8224646  
## 371 0.8070397 -0.9881551  
## 372 12.3145394 1.2840297

Did we really need the whole lme machinery to fit a within-subject linear regression and then average over subjects? The answer is yes. The assumptions on the distribution of random effect, namely, that they are normally distributed, allows us to pool information from one subject to another. In the words of John Tukey: "we borrow strength over subjects". Is this a good thing? If the normality assumption is true, it certainly is. If, on the other hand, you have a lot of samples per subject, and you don't need to "borrow strength" from one subject to another, you can simply fit within-subject linear models without the mixed-models machinery.

To demonstrate the "strength borrowing", here is a comparison of the subject-wise intercepts of the mixed-model, versus a subject-wise linear model. They are not the same.



Here is a comparison of the random-day effect from lme versus a subject-wise linear model. They are not the same.



detach(Penicillin)

## The Variance-Components View

TODO

## Bibliographic Notes

Most of the examples in this chapter are from the documentation of the **lme4** package (Bates et al. [2015](#ref-lme4)). For a more theoretical view see Weiss ([2005](#ref-weiss2005modeling)) or Searle, Casella, and McCulloch ([2009](#ref-searle2009variance)). As usual, a hands on view can be found in Venables and Ripley ([2013](#ref-venables2013modern)).

## Practice Yourself

1. Return to the Penicillin data set. Instead of fitting an LME model, fit an LM model with lm. I.e., treat all random effects as fixed.
   1. Compare the effect estimates.
   2. Compare the standard errors.
   3. Compare the predictions of the two models.
2. [Very Advanced!] Return to the Penicillin data and use the gls function to fit a generalized linear model, equivalent to the LME model in our text.

# Multivariate Data Analysis

The term "multivariate data analysis" is so broad and so overloaded, that we start by clarifying what is discussed and what is not discussed in this chapter. Broadly speaking, we will discuss statistical *inference*, and leave more "exploratory flavored" matters like clustering, and visualization, to the Unsupervised Learning Chapter 10.

More formally, let be a variate random vector, with . Here is the set of problems we will discuss, in order of their statistical difficulty.

* **Signal detection**: a.k.a. *multivariate hypothesis testing*, i.e., testing if equals and for in particular.
* **Signal counting**: Counting the number of elements in that differ from , and for in particular.
* **Signal identification**: a.k.a. *multiple testing*, i.e., testing which of the elements in differ from and for in particular.
* **Signal estimation**: Estimating the magnitudes of the departure of from , and for in particular. If estimation follows a *signal detection* or *signal identification* stage, this is known as a *selective estimation* problem.
* **Multivariate Regression**: a.k.a. *MANOVA* in statistical literature, and *structured learning* in the machine learning literature.
* **Graphical Models**: Learning *graphical models* deals with the fitting/learning the multivariate distribution of . In particular, it deals with the identification of independencies between elements of .

Example 10 Consider the problem of a patient monitored in the intensive care unit. At every minute the monitor takes physiological measurements: blood pressure, body temperature, etc. The total number of minutes in our data is , so that in total, we have measurements, arranged in a matrix. We also know the typical measurements for this patient when healthy: .

Example 11 Consider the problem of the simplest multiple regression. The estimated coefficient, are a random vector. Regression theory tells us that its covariance is , and null mean of . We thus see that inference on the vector of regression coefficients, is nothing more than a multivaraite inference problem.

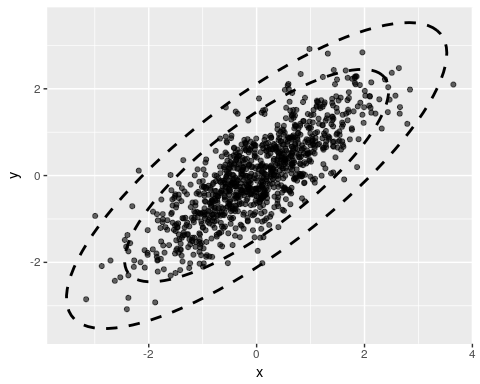
Demonstrating our terminology on Example 10: Signal detection means testing if the patient's measurement depart in any way from his healthy state, . Signal counting means measuring *how many* measurement depart from the healthy state. Signal identification means pin-pointing which of the physiological measurements depart from his healthy state. Signal estimation means estimating the magnitude of the departure from the healthy state. Multivariate regression means finding the factors which many explain the departure from the healthy state. Fitting a distribution means finding the joint distribution of the physiological measurements, and in particular, their dependence and independence.

Remark. In the above, "signal" is defined in terms of . It is possible that the signal is not in the location, , but rather in the covariance, . We do not discuss these problems here, and refer the reader to Nadler ([2008](#ref-nadler2008finite)).

Another possible question is: does a multivariate analysis gives us something we cannot get from a mass-univariate analysis (i.e., a multivariate analysis on each variable separately). In Example 10 we could have just performed multiple univariate tests, and sign an alarm when any of the univariate detectors was triggered. The reason we want a multivariate detector, and not multiple univariate detectors is that it is possible that each measurement alone is borderline, but together, the signal accumulates. In our ICU example is may mean that the pulse is borderline, the body temperature is borderline, etc. Analyzed simultaneously, it is clear that the patient is in distress.

The next figure[[17]](#footnote-212) illustrates the idea that some bi-variate measurements may seem ordinary univariately, while very anomalous when examined bi-variately.

Remark. The following figure may also be used to demonstrate the difference between Euclidean Distance and Mahalanobis Distance.



## Signal Detection

Signal detection deals with the detection of the departure of from some , and especially, . This problem can be thought of as the multivariate counterpart of the univariate hypothesis test. Indeed, the most fundamental approach is a mere generalization of the t-test, known as *Hotelling's test*.

Recall the univariate t-statistic of a data vector of length :

where , and is the unbiased variance estimator .

Generalizing Eq(12) to the multivariate case: is a -vector, is a -vector, and is a matrix of the covariance between the coordinated of . When operating with vectors, the squaring becomes a quadratic form, and the division becomes a matrix inverse. We thus have

which is the definition of Hotelling's test statistic. We typically denote the covariance between coordinates in with , so that . Using the notation, Eq.(13) becomes

which is the standard notation of Hotelling's test statistic.

For inference, we need the null distribution of Hotelling's test statistic. For this we introduce some vocabulary[[18]](#footnote-215):

1. **Low Dimension**: We call a problem *low dimensional* if , i.e. . This means there are many observations per estimated parameter.
2. **High Dimension**: We call a problem *high dimensional* if , where . This means there are more observations than parameters, but not many.
3. **Very High Dimension**: We call a problem *very high dimensional* if , where . This means there are less observations than parameter.

Hotelling's test can only be used in the low dimensional regime. For some intuition on this statement, think of taking measurements of physiological variables. We seemingly have observations, but there are unknown quantities in . Would you trust your conclusion that is different than based on merely observations.

Put formally: We cannot compute Hotelling's test when because is simply not invertible-- this is an algebraic problem. We cannot compute Hotelling's test when because the signal-to-noise is very low-- this is a statistical problem.

Only in the low dimensional case can we compute and trust Hotelling's test. When then is roughly distributed with degrees of freedom. The F distribution may also be found in the literature in this context, and will appear if assuming the -vectors are independent, and -variate Gaussian. This F distribution is non-robust the underlying assumptions, so from a practical point of view, I would not trust the Hotelling test unless .

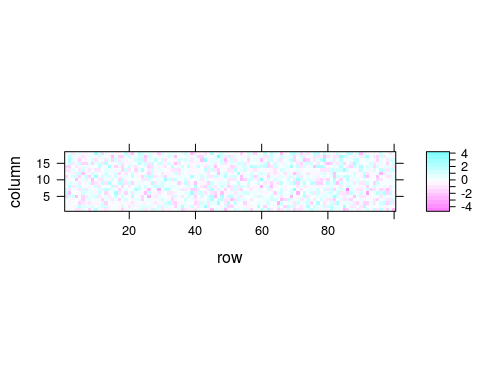
### Signal Detection with R

Let's generate some data with no signal.

library(mvtnorm)  
n <- 1e2 # observations  
p <- 1.8e1 # parameter dimension  
mu <- rep(0,p) # no signal  
x <- rmvnorm(n = n, mean = mu)  
dim(x)

## [1] 100 18

lattice::levelplot(x)



Now make our own Hotelling function.

hotellingOneSample <- function(x, mu0=rep(0,ncol(x))){  
 n <- nrow(x)  
 p <- ncol(x)  
 stopifnot(n > 5\* p)  
 bar.x <- colMeans(x)  
 Sigma <- var(x)  
 Sigma.inv <- solve(Sigma)  
 T2 <- n \* (bar.x-mu0) %\*% Sigma.inv %\*% (bar.x-mu0)  
 p.value <- pchisq(q = T2, df = p, lower.tail = FALSE)  
 return(list(statistic=T2, pvalue=p.value))  
}  
  
hotellingOneSample(x)

## $statistic  
## [,1]  
## [1,] 26.14718  
##   
## $pvalue  
## [,1]  
## [1,] 0.09644004

Things to note:

* stopifnot(n > 5 \* p) is a little verification to check that the problem is indeed low dimensional. Otherwise, the approximation cannot be trusted.
* solve returns a matrix inverse.
* %\*% is the matrix product operator (see also crossprod()).
* A function may return only a single object, so we wrap the statistic and its p-value in a list object.

Just for verification, we compare our home made Hotelling's test, to the implementation in the **rrcov** package. The statistic is clearly OK, but our approximation of the distribution leaves room to desire. Personally, I would never trust a Hotelling test if is not much greater than ...

rrcov::T2.test(x)

##   
## One-sample Hotelling test  
##   
## data: x  
## T2 = 26.1470, F = 1.2032, df1 = 18, df2 = 82, p-value = 0.2783  
## alternative hypothesis: true mean vector is not equal to (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)'   
##   
## sample estimates:  
## [,1] [,2] [,3] [,4] [,5]  
## mean x-vector -0.1212658 0.1555007 -0.09015709 -0.1353305 0.02612866  
## [,6] [,7] [,8] [,9] [,10]  
## mean x-vector 0.0204975 -0.1098174 0.02018985 0.0695986 -0.02264407  
## [,11] [,12] [,13] [,14] [,15]  
## mean x-vector 0.1729569 -0.1532554 0.02083236 0.1532461 6.771846e-05  
## [,16] [,17] [,18]  
## mean x-vector 0.07941955 -0.03240024 0.14604

## Signal Counting

There are many ways to approach the *signal counting* problem. For the purposes of this book, however, we will not discuss them directly, and solve the signal counting problem as a signal identification problem: if we know **where** departs from , we only need to count coordinates to solve the signal counting problem.

## Signal Identification

The problem of *signal identification* is also known as *selective testing*, or more commonly as *multiple testing*.

In the ANOVA literature, an identification stage will typically follow a detection stage. These are known as the *omnibus F test*, and *post-hoc* tests, respectively. In the multiple testing literature there will typically be no preliminary detection stage. It is typically assumed that signal is present, and the only question is "where?"

The first question when approaching a multiple testing problem is "what is an error"? Is an error declaring a coordinate in to be different than when it is actually not? Is an error an overly high proportion of falsely identified coordinates? The former is known as the *family wise error rate* (FWER), and the latter as the *false discovery rate* (FDR).

### Signal Identification in R

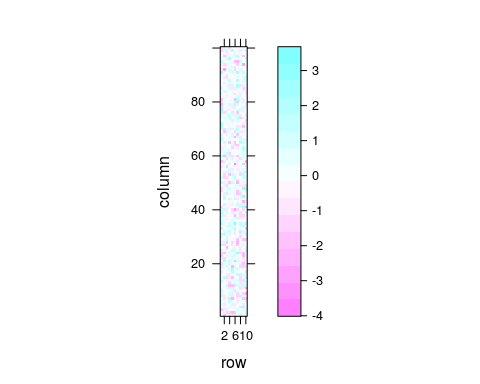
One (of many) ways to do signal identification involves the stats::p.adjust function. [TODO: clarify why use p.adjust?] The function takes as inputs a -vector of **p-values**. This implies that: (i) you are assumed to be able to compute the p-value of each the hypothesis tested; one hypothesis for every coordinate in . (ii) unlike the Hotelling test, we do not try to estimate the covariance between coordinates. Not because it is not important, but rather, because the methods we will use apply to a wide variety of covariances, so the covariance does not need to be estimated.

We start be generating some high-dimensional multivariate data and computing the coordinate-wise (i.e. hypothesis-wise) p-value.

library(mvtnorm)  
n <- 1e1  
p <- 1e2  
mu <- rep(0,p)  
x <- rmvnorm(n = n, mean = mu)  
dim(x)

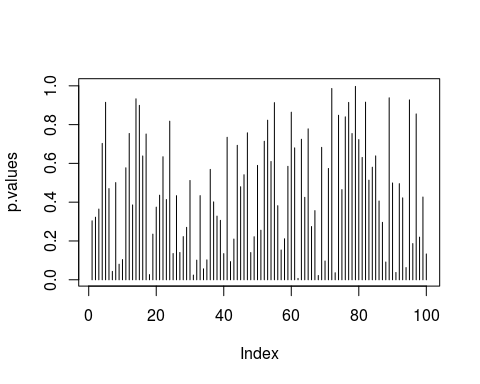
## [1] 10 100

lattice::levelplot(x)



We now compute the pvalues of each coordinate. We use a coordinate-wise t-test. Why a t-test? Because for the purpose of demonstration we want a simple test. In reality, you may use any test that returns valid p-values.

t.pval <- function(y) t.test(y)$p.value  
p.values <- apply(X = x, MARGIN = 2, FUN = t.pval)   
plot(p.values, type='h')



Things to note:

* t.pval is a function that merely returns the p-value of a t.test.
* We used the apply function to apply the same function to each column of x.
* MARGIN=2 tells apply to compute over columns and not rows.
* The output, p.values, is a vector of 100 p-values.

We are now ready to do the identification, i.e., find which coordinate of is different than . The workflow is: (i) Compute an adjusted p-value. (ii) Compare the adjusted p-value to the desired error level.

If we want , meaning that we allow a probability of making any mistake, we will use the method="holm" argument of p.adjust.

alpha <- 0.05  
p.values.holm <- p.adjust(p.values, method = 'holm' )  
table(p.values.holm < alpha)

##   
## FALSE   
## 100

If we want , meaning that we allow the proportion of false discoveries to be no larger than , we use the method="BH" argument of p.adjust.

alpha <- 0.05  
p.values.BH <- p.adjust(p.values, method = 'BH' )  
table(p.values.BH < alpha)

##   
## FALSE   
## 100

We now inject some strong signal in just to see that the process works. We will artificially inject signal in the first 10 coordinates.

mu[1:10] <- 2 # inject signal  
x <- rmvnorm(n = n, mean = mu) # generate data  
p.values <- apply(X = x, MARGIN = 2, FUN = t.pval)   
p.values.BH <- p.adjust(p.values, method = 'BH' )  
which(p.values.BH < alpha)

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

Indeed- we are now able to detect that the first coordinates carry signal, because their respective coordinate-wise null hypotheses have been rejected.

## Signal Estimation (\*)

The estimation of the elements of is a seemingly straightforward task. This is not the case, however, if we estimate only the elements that were selected because they were significant (or any other data-dependent criterion). Clearly, estimating only significant entries will introduce a bias in the estimation. In the statistical literature, this is known as *selection bias*. Selection bias also occurs when you perform inference on regression coefficients after some model selection, say, with a lasso, or a forward search[[19]](#footnote-224).

Selective inference is a complicated and active research topic so we will not offer any off-the-shelf solution to the matter. The curious reader is invited to read J. D. Rosenblatt and Benjamini ([2014](#ref-rosenblatt2014selective)), Javanmard and Montanari ([2014](#ref-javanmard2014confidence)), or [Will Fithian's](http://www.stat.berkeley.edu/~wfithian/) PhD thesis (Fithian [2015](#ref-fithian2015topics)) for more on the topic.

## Multivariate Regression (\*)

*Multivaraite regression*, a.k.a. *MANOVA*, similar to [*structured learning*](https://en.wikipedia.org/wiki/Structured_prediction) in machine learning, is simply a regression problem where the outcome, , is not scalar values but vector valued. It is not to be confused with *multiple regression* where the predictor, , is vector valued, but the outcome is scalar.

If the linear models generalize the two-sample t-test from two, to multiple populations, then multivariate regression generalizes Hotelling's test in the same way.

When the entries of are independent, MANOVA collapses to multiple univariate regressions. It is only when entries in are correlated that we can gain in accuracy and power by harnessing these correlations through the MANOVA framework.

### Multivariate Regression with R

TODO

## Graphical Models (\*)

Fitting a multivariate distribution, i.e. learning a *graphical model*, is a very hard task. To see why, consider the problem of continuous variables. In the simplest case, where we can assume normality, fitting a distributions means estimating the parameters in the expectation, , and parameters in the covariance, . The number of observations required for this task, , may be formidable.

A more humble task, is to identify **independencies**, known as *structure learning* in the machine learning literature. Under the multivariate normality assumption, this means identifying zero entries in , or more precisely, zero entries in . This task can be approached as a **signal identification** problem (8.3). The same solutions may be applied to identify non-zero entries in , instead of as discussed until now.

If multivariate normality cannot be assumed, then identifying independencies cannot be done via the covariance matrix and more elaborate algorithms are required.

### Graphical Models in R

TODO

## Biblipgraphic Notes

For a general introduction to multivariate data analysis see Anderson-Cook ([2004](#ref-anderson2004introduction)). For an R oriented introduction, see Everitt and Hothorn ([2011](#ref-everitt2011introduction)). For more on the difficulties with high dimensional problems, see Bai and Saranadasa ([1996](#ref-bai1996effect)). For some cutting edge solutions for testing in high-dimension, see Goeman, Van De Geer, and Van Houwelingen ([2006](#ref-goeman2006testing)). For more on multiple testing, and signal identification, see Efron ([2012](#ref-efron2012large)). For more on the choice of your error rate see J. Rosenblatt ([2013](#ref-rosenblatt2013practitioner)). For an excellent review on graphical models see Kalisch and Bühlmann ([2014](#ref-kalisch2014causal)). Everything you need on graphical models, Bayesian belief networks, and structure learning in R, is collected in the [Task View](https://cran.r-project.org/web/views/gR.html).

## Practice Yourself

1. Generate multivariate data from two groups: rmvnorm(n = 100, mean = rep(0,10)) for the first, and rmvnorm(n = 100, mean = rep(0.1,10)) for the second.
   1. Do we agree the groups differ?
   2. Implement the two-group Hotelling test described in Wikipedia: (<https://en.wikipedia.org/wiki/Hotelling%27s_T-squared_distribution#Two-sample_statistic>).
   3. Verify that you are able to detect that the groups differ.
   4. Perform a two-group t-test on each coordinate. On which coordinates can you detect signal while controlling the FWER ? On which while controlling the FDR? Use p.adjust.
2. Return to the previous problem, but set n=10. Verify that you cannot compute your Hotelling statistic.

# Supervised Learning

Machine learning is very similar to statistics, but it is certainly not the same. As the name suggests, in machine learning we want machines to learn. This means that we want to replace hard-coded expert algorithm, with data-driven self-learned algorithm.

There are many learning setups, that depend on what information is available to the machine. The most common setup, discussed in this chapter, is *supervised learning*. The name takes from the fact that by giving the machine data samples with known inputs (a.k.a. features) and desired outputs (a.k.a. labels), the human is effectively supervising the learning. If we think of the inputs as predictors, and outcomes as predicted, it is no wonder that supervised learning is very similar to statistical prediction. When asked "are these the same?" I like to give the example of internet fraud. If you take a sample of fraud "attacks", a statistical formulation of the problem is highly unlikely. This is because fraud events are not randomly drawn from some distribution, but rather, arrive from an adversary learning the defenses and adapting to it. This instance of supervised learning is more similar to game theory than statistics.

Other types of machine learning problems include (Sammut and Webb [2011](#ref-sammut2011encyclopedia)):

* **Unsupervised learning**: See Chapter 10.
* **Semi supervised learning**: Where only part of the samples are labeled. A.k.a. *co-training*, *learning from labeled and unlabeled data*, *transductive learning*.
* **Active learning**: Where the machine is allowed to query the user for labels. Very similar to *adaptive design of experiments*.
* **Learning on a budget**: A version of active learning where querying for labels induces variable costs.
* **Weak learning**: A version of supervised learning where the labels are given not by an expert, but rather by some heuristic rule. Example: mass-labelling cyber attacks by a rule based software, instead of a manual inspection.
* **Reinforcement learning**:  
  Similar to active learning, in that the machine may query for labels. Different from active learning, in that the machine does not receive labels, but *rewards*.
* **Structure learning**: When predicting objects with structure such as dependent vectors, graphs, images, tensors, etc.
* **Manifold learning**: An instance of unsupervised learning, where the goal is to reduce the dimension of the data by embedding it into a lower dimensional manifold. A.k.a. *support estimation*.
* **Learning to learn**: Deals with the carriage of "experience" from one learning problem to another. A.k.a. *cummulative learning*, *knowledge transfer*, and *meta learning*.

## Problem Setup

We now present the *empirical risk minimization* (ERM) approach to supervised learning, a.k.a. *M-estimation* in the statistical literature.

Remark. We do not discuss purely algorithmic approaches such as K-nearest neighbour and *kernel smoothing* due to space constraints. For a broader review of supervised learning, see the Bibliographic Notes.

Given samples with inputs from some space and desired outcome, , from some space . Samples, have some distribution we denote . We want to learn a function that maps inputs to outputs. This function is called a *hypothesis*, or *predictor*, or *classifier*, denoted , that belongs to a hypothesis class such that . We also choose some other function that fines us for erroneous prediction. This function is called the *loss*, and we denote it by .

Remark. The *hypothesis* in machine learning is only vaguely related the *hypothesis* in statistical testing, which is quite confusing.

Remark. The *hypothesis* in machine learning is not a bona-fide *statistical model* since we don't assume it is the data generating process, but rather some function which we choose for its good predictive performance.

The fundamental task in supervised (statistical) learning is to recover a hypothesis that minimizes the average loss in the sample, and not in the population. This is know as the *risk minimization problem*.

Definition 16 (Risk Function) The *risk function*, a.k.a. *generalization error*, or *test error*, is the population average loss of a predictor : $$

The best predictor, is the risk minimizer:

To make things more explicit, may be a linear function, and a squared error loss, in which case problem (14) collapses to

Another fundamental problem is that we do not know the distribution of all possible inputs and outputs, . We typically only have a sample of . We thus state the *empirical* counterpart of (14), which consists of minimizing the average loss. This is known as the *empirical risk miminization* problem (ERM).

Definition 17 (Empirical Risk) The *empirical risk function*, a.k.a. *in-sample error*, or *train error*, is the sample average loss of a predictor : $$

A good candidate predictor, , is thus the *empirical risk minimizer*:

Making things more explicit again by using a linear hypothesis with squared loss, we see that the empirical risk minimization problem collapses to an ordinary least-squares problem:

When data samples are assumingly independent, then maximum likelihood estimation is also an instance of ERM, when using the (negative) log likelihood as the loss function.

If we don't assume any structure on the hypothesis, , then from (15) will interpolate the data, and will be a very bad predictor. We say, it will *overfit* the observed data, and will have bad performance on new data.

We have several ways to avoid overfitting:

1. Restrict the hypothesis class (such as linear functions).
2. Penalize for the complexity of . The penalty denoted by .
3. Unbiased risk estimation, where we deal with the overfitted optimism of the empirical risk by debiasing it.

### Common Hypothesis Classes

Some common hypothesis classes, , with restricted complexity, are:

1. **Linear hypotheses**: such as linear models, GLMs, and (linear) support vector machines (SVM).
2. **Neural networks**: a.k.a. *feed-forward* neural nets, *artificial* neural nets, and the celebrated class of *deep* neural nets.
3. **Tree**: a.k.a. *decision rules*, is a class of hypotheses which can be stated as "if-then" rules.
4. **Reproducing Kernel Hilbert Space**: a.k.a. RKHS, is a subset of "the space of all functions[[20]](#footnote-238)" that is both large enough to capture very complicated relations, but small enough so that it is less prone to overfitting, and also surprisingly simple to compute with.
5. **Ensembles**: a "meta" hypothesis class, which consists of taking multiple hypotheses, possibly from different classes, and combining them.

### Common Complexity Penalties

The most common complexity penalty applies to classes that have a finite dimensional parametric representation, such as the class of linear predictors, parametrized via its coefficients . In such classes we may penalize for the norm of the parameters. Common penalties include:

1. **Ridge penalty**: penalizing the norm of the parameter. I.e. .
2. **Lasso penalty**: penalizing the norm of the parameter. I.e.,
3. **Elastic net**: a combination of the lasso and ridge penalty. I.e. ,.

If the hypothesis class does not admit a finite dimensional parametric representation, we may penalize it with some functional norm such as .

### Unbiased Risk Estimation

The fundamental problem of overfitting, is that the empirical risk, , is downward biased to the population risk, . Formally:

Why is that? Think of estimating a population's mean with the sample minimum. It can be done, but the minimum has to be debiased for it to estimate the population mean. Unbiased estimation of broadly fall under: (a) purely algorithmic *resampling* based approaches, and (b) theory driven estimators.

1. **Train-Validate-Test**: The simplest form of validation is to split the data. A *train* set to train/estimate/learn . A *validation* set to compute the out-of-sample expected loss, , and pick the best performing predictor. A *test* sample to compute the out-of-sample performance of the selected hypothesis. This is a very simple approach, but it is very "data inefficient", thus motivating the next method.
2. **V-Fold Cross Validation**: By far the most popular risk estimation algorithm, in *V-fold CV* we "fold" the data into non-overlapping sets. For each of the sets, we learn with the non-selected fold, and assess ) on the selected fold. We then aggregate results over the folds, typically by averaging.
3. **AIC**: Akaike's information criterion (AIC) is a theory driven correction of the empirical risk, so that it is unbiased to the true risk. It is appropriate when using the likelihood loss.
4. **Cp**: Mallow's Cp is an instance of AIC for likelihood loss under normal noise.

Other theory driven unbiased risk estimators include the *Bayesian Information Criterion* (BIC, aka SBC, aka SBIC), the *Minimum Description Length* (MDL), *Vapnic’s Structural Risk Minimization* (SRM), the *Deviance Information Criterion* (DIC), and the *Hannan-Quinn Information Criterion* (HQC).

Other resampling based unbiased risk estimators include resampling **without replacement** algorithms like *delete-d cross validation* with its many variations, and **resampling with replacement**, like the *bootstrap*, with its many variations.

### Collecting the Pieces

An ERM problem with regularization will look like

Collecting ideas from the above sections, a typical supervised learning pipeline will include: choosing the hypothesis class, choosing the penalty function and level, unbiased risk estimator. We emphasize that choosing the penalty function, is not enough, and we need to choose how "hard" to apply it. This if known as the *regularization level*, denoted by in Eq.(16).

Examples of such combos include:

1. Linear regression, no penalty, train-validate test.
2. Linear regression, no penalty, AIC.
3. Linear regression, penalty, V-fold CV. This combo is typically known as *ridge regression*.
4. Linear regression, penalty, V-fold CV. This combo is typically known as *lasso regression*.
5. Linear regression, and penalty, V-fold CV. This combo is typically known as *elastic net regression*.
6. Logistic regression, penalty, V-fold CV.
7. SVM classification, penalty, V-fold CV.
8. Deep network, no penalty, V-fold CV.
9. Etc.

For fans of statistical hypothesis testing we will also emphasize: Testing and prediction are related, but are not the same. **It is indeed possible that we will want to ignore a significant predictor, and add a non-significant one!** (Foster and Stine [2004](#ref-foster2004variable)) Some authors will use hypothesis testing as an initial screening of candidate predictors. This is a useful heuristic, but that is all it is-- a heuristic.

## Supervised Learning in R

At this point, we have a rich enough language to do supervised learning with R.

In these examples, I will use two data sets from the **ElemStatLearn** package: spam for categorical predictions, and prostate for continuous predictions. In spam we will try to decide if a mail is spam or not. In prostate we will try to predict the size of a cancerous tumor. You can now call ?prostate and ?spam to learn more about these data sets.

Some boring pre-processing.

library(ElemStatLearn)   
data("prostate")  
data("spam")  
  
library(magrittr) # for piping  
  
# Preparing prostate data  
prostate.train <- prostate[prostate$train, names(prostate)!='train']  
prostate.test <- prostate[!prostate$train, names(prostate)!='train']   
y.train <- prostate.train$lcavol  
X.train <- as.matrix(prostate.train[, names(prostate.train)!='lcavol'] )  
y.test <- prostate.test$lcavol   
X.test <- as.matrix(prostate.test[, names(prostate.test)!='lcavol'] )  
  
# Preparing spam data:  
n <- nrow(spam)  
  
train.prop <- 0.66  
train.ind <- c(TRUE,FALSE) %>%   
 sample(size = n, prob = c(train.prop,1-train.prop), replace=TRUE)  
spam.train <- spam[train.ind,]  
spam.test <- spam[!train.ind,]  
  
y.train.spam <- spam.train$spam  
X.train.spam <- as.matrix(spam.train[,names(spam.train)!='spam'] )  
y.test.spam <- spam.test$spam  
X.test.spam <- as.matrix(spam.test[,names(spam.test)!='spam'])   
  
spam.dummy <- spam  
spam.dummy$spam <- as.numeric(spam$spam=='spam')   
spam.train.dummy <- spam.dummy[train.ind,]  
spam.test.dummy <- spam.dummy[!train.ind,]

We also define some utility functions that we will require down the road.

l2 <- function(x) x^2 %>% sum %>% sqrt   
l1 <- function(x) abs(x) %>% sum   
MSE <- function(x) x^2 %>% mean   
missclassification <- function(tab) sum(tab[c(2,3)])/sum(tab)

### Linear Models with Least Squares Loss

Starting with OLS regression, and a train-test data approach. Notice the better in-sample MSE than the out-of-sample. That is overfitting in action.

ols.1 <- lm(lcavol~. ,data = prostate.train)  
# Train error:  
MSE( predict(ols.1)- prostate.train$lcavol)

## [1] 0.4383709

# Test error:  
MSE( predict(ols.1, newdata = prostate.test)- prostate.test$lcavol)

## [1] 0.5084068

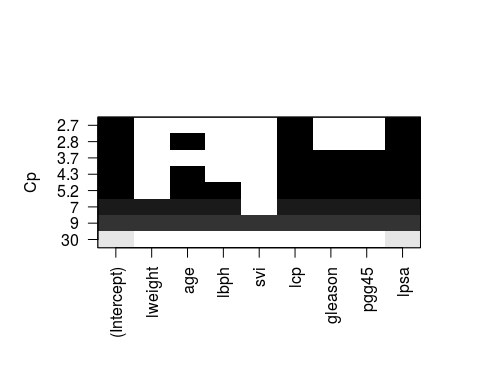
We now implement a V-fold CV, instead of our train-test approach. The assignment of each observation to each fold is encoded in fold.assignment. The following code is extremely inefficient, but easy to read.

folds <- 10  
fold.assignment <- sample(1:folds, nrow(prostate), replace = TRUE)  
errors <- NULL  
  
for (k in 1:folds){  
 prostate.cross.train <- prostate[fold.assignment!=k,] # train subset  
 prostate.cross.test <- prostate[fold.assignment==k,] # test subset  
 .ols <- lm(lcavol~. ,data = prostate.cross.train) # train  
 .predictions <- predict(.ols, newdata=prostate.cross.test)  
 .errors <- .predictions - prostate.cross.test$lcavol # save prediction errors in the fold  
 errors <- c(errors, .errors) # aggregate error over folds.  
}  
  
# Cross validated prediction error:  
MSE(errors)

## [1] 0.5484752

Let's try all possible models, and choose the best performer with respect to the Cp unbiased risk estimator. This is done with leaps::regsubsets. We see that the best performer has 3 predictors.

library(leaps)  
regfit.full <- prostate.train %>%   
 regsubsets(lcavol~.,data = ., method = 'exhaustive') # best subset selection  
plot(regfit.full, scale = "Cp")

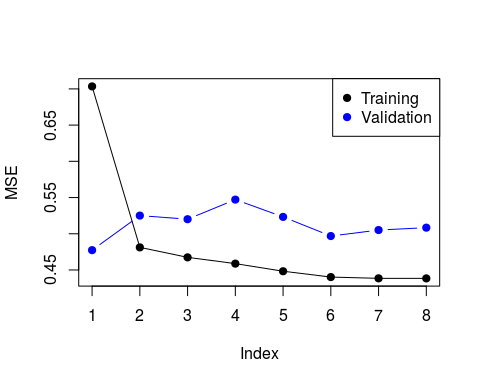


Instead of the Cp criterion, we now compute the train and test errors for all the possible predictor subsets[[21]](#footnote-245). In the resulting plot we can see overfitting in action.

model.n <- regfit.full %>% summary %>% length  
X.train.named <- model.matrix(lcavol ~ ., data = prostate.train )   
X.test.named <- model.matrix(lcavol ~ ., data = prostate.test )   
  
  
val.errors <- rep(NA, model.n)  
train.errors <- rep(NA, model.n)  
for (i in 1:model.n) {  
 coefi <- coef(regfit.full, id = i)  
   
 pred <- X.train.named[, names(coefi)] %\*% coefi # make in-sample predictions  
 train.errors[i] <- MSE(y.train - pred) # train errors  
  
 pred <- X.test.named[, names(coefi)] %\*% coefi # make out-of-sample predictions  
 val.errors[i] <- MSE(y.test - pred) # test errors  
}

Plotting results.

plot(train.errors, ylab = "MSE", pch = 19, type = "o")  
points(val.errors, pch = 19, type = "b", col="blue")  
legend("topright",   
 legend = c("Training", "Validation"),   
 col = c("black", "blue"),   
 pch = 19)



Checking all possible models is computationally very hard. *Forward selection* is a greedy approach that adds one variable at a time, using the AIC risk estimator. If AIC decreases, the variable is added.

ols.0 <- lm(lcavol~1 ,data = prostate.train)  
model.scope <- list(upper=ols.1, lower=ols.0)  
step(ols.0, scope=model.scope, direction='forward', trace = TRUE)

## Start: AIC=30.1  
## lcavol ~ 1  
##   
## Df Sum of Sq RSS AIC  
## + lpsa 1 54.776 47.130 -19.570  
## + lcp 1 48.805 53.101 -11.578  
## + svi 1 35.829 66.077 3.071  
## + pgg45 1 23.789 78.117 14.285  
## + gleason 1 18.529 83.377 18.651  
## + lweight 1 9.186 92.720 25.768  
## + age 1 8.354 93.552 26.366  
## <none> 101.906 30.097  
## + lbph 1 0.407 101.499 31.829  
##   
## Step: AIC=-19.57  
## lcavol ~ lpsa  
##   
## Df Sum of Sq RSS AIC  
## + lcp 1 14.8895 32.240 -43.009  
## + svi 1 5.0373 42.093 -25.143  
## + gleason 1 3.5500 43.580 -22.817  
## + pgg45 1 3.0503 44.080 -22.053  
## + lbph 1 1.8389 45.291 -20.236  
## + age 1 1.5329 45.597 -19.785  
## <none> 47.130 -19.570  
## + lweight 1 0.4106 46.719 -18.156  
##   
## Step: AIC=-43.01  
## lcavol ~ lpsa + lcp  
##   
## Df Sum of Sq RSS AIC  
## <none> 32.240 -43.009  
## + age 1 0.92315 31.317 -42.955  
## + pgg45 1 0.29594 31.944 -41.627  
## + gleason 1 0.21500 32.025 -41.457  
## + lbph 1 0.13904 32.101 -41.298  
## + lweight 1 0.05504 32.185 -41.123  
## + svi 1 0.02069 32.220 -41.052

##   
## Call:  
## lm(formula = lcavol ~ lpsa + lcp, data = prostate.train)  
##   
## Coefficients:  
## (Intercept) lpsa lcp   
## 0.08798 0.53369 0.38879

We now learn a linear predictor on the spam data using, a least squares loss, and train-test risk estimator.

# train the predictor  
ols.2 <- lm(spam~., data = spam.train.dummy)   
  
# make in-sample predictions  
.predictions.train <- predict(ols.2) > 0.5   
# inspect the confusion matrix  
(confusion.train <- table(prediction=.predictions.train, truth=spam.train.dummy$spam))

## truth  
## prediction 0 1  
## FALSE 1755 264  
## TRUE 80 919

# compute the train (in sample) misclassification  
missclassification(confusion.train)

## [1] 0.1139828

# make out-of-sample prediction  
.predictions.test <- predict(ols.2, newdata = spam.test.dummy) > 0.5   
# inspect the confusion matrix  
(confusion.test <- table(prediction=.predictions.test, truth=spam.test.dummy$spam))

## truth  
## prediction 0 1  
## FALSE 907 131  
## TRUE 46 499

# compute the train (in sample) misclassification  
missclassification(confusion.test)

## [1] 0.111813

The glmnet package is an excellent package that provides ridge, lasso, and elastic net regularization, for all GLMs, so for linear models in particular.

suppressMessages(library(glmnet))  
ridge.2 <- glmnet(x=X.train, y=y.train, family = 'gaussian', alpha = 0)  
  
# Train error:  
MSE( predict(ridge.2, newx =X.train)- y.train)

## [1] 1.006028

# Test error:  
MSE( predict(ridge.2, newx = X.test)- y.test)

## [1] 0.7678264

Things to note:

* The alpha=0 parameters tells R to do ridge regression. Setting will do lasso, and any other value, with return an elastic net with appropriate weights.
* The `family='gaussian' argument tells R to fit a linear model, with least squares loss.
* The test error is **smaller** than the train error, which I attribute to the variability of the risk estimators.

Remark. The variability of risk estimator is a very interesting problem, which recieved very little attention in the machine learning literature. If this topic interests you, talk to me.

We now use the lasso penalty.

lasso.1 <- glmnet(x=X.train, y=y.train, , family='gaussian', alpha = 1)  
  
# Train error:  
MSE( predict(lasso.1, newx =X.train)- y.train)

## [1] 0.5525279

# Test error:  
MSE( predict(lasso.1, newx = X.test)- y.test)

## [1] 0.5211263

We now use glmnet for classification.

logistic.2 <- cv.glmnet(x=X.train.spam, y=y.train.spam, family = "binomial", alpha = 0)

Things to note:

* We used cv.glmnet to do an automatic search for the optimal level of regularization (the lambda argument in glmnet) using V-fold CV.
* We set alpha=0 for ridge regression.

# Train confusion matrix:  
.predictions.train <- predict(logistic.2, newx = X.train.spam, type = 'class')   
(confusion.train <- table(prediction=.predictions.train, truth=spam.train$spam))

## truth  
## prediction email spam  
## email 1756 196  
## spam 79 987

# Train misclassification error  
missclassification(confusion.train)

## [1] 0.09111995

# Test confusion matrix:  
.predictions.test <- predict(logistic.2, newx = X.test.spam, type='class')   
(confusion.test <- table(prediction=.predictions.test, truth=y.test.spam))

## truth  
## prediction email spam  
## email 905 95  
## spam 48 535

# Test misclassification error:  
missclassification(confusion.test)

## [1] 0.09033481

### SVM

A support vector machine (SVM) is a linear hypothesis class with a particular loss function known as a *hinge loss*. We learn an SVM with the svm function from the **e1071** package, which is merely a wrapper for the **libsvm** C library, which is the most popular implementation of SVM today.

library(e1071)  
svm.1 <- svm(spam~., data = spam.train)  
  
# Train confusion matrix:  
.predictions.train <- predict(svm.1)   
(confusion.train <- table(prediction=.predictions.train, truth=spam.train$spam))

## truth  
## prediction email spam  
## email 1783 107  
## spam 52 1076

missclassification(confusion.train)

## [1] 0.0526839

# Test confusion matrix:  
.predictions.test <- predict(svm.1, newdata = spam.test)   
(confusion.test <- table(prediction=.predictions.test, truth=spam.test$spam))

## truth  
## prediction email spam  
## email 905 71  
## spam 48 559

missclassification(confusion.test)

## [1] 0.07517372

We can also use SVM for regression.

svm.2 <- svm(lcavol~., data = prostate.train)  
  
# Train error:  
MSE( predict(svm.2)- prostate.train$lcavol)

## [1] 0.3336868

# Test error:  
MSE( predict(svm.2, newdata = prostate.test)- prostate.test$lcavol)

## [1] 0.5633183

### Neural Nets

Neural nets (non deep) can be fitted, for example, with the nnet function in the **nnet** package. We start with a nnet regression.

library(nnet)  
nnet.1 <- nnet(lcavol~., size=20, data=prostate.train, rang = 0.1, decay = 5e-4, maxit = 1000, trace=FALSE)  
  
# Train error:  
MSE( predict(nnet.1)- prostate.train$lcavol)

## [1] 1.189798

# Test error:  
MSE( predict(nnet.1, newdata = prostate.test)- prostate.test$lcavol)

## [1] 1.29209

And nnet classification.

nnet.2 <- nnet(spam~., size=5, data=spam.train, rang = 0.1, decay = 5e-4, maxit = 1000, trace=FALSE)  
  
# Train confusion matrix:  
.predictions.train <- predict(nnet.2, type='class')   
(confusion.train <- table(prediction=.predictions.train, truth=spam.train$spam))

## truth  
## prediction email spam  
## email 1798 33  
## spam 37 1150

missclassification(confusion.train)

## [1] 0.02319417

# Test confusion matrix:  
.predictions.test <- predict(nnet.2, newdata = spam.test, type='class')   
(confusion.test <- table(prediction=.predictions.test, truth=spam.test$spam))

## truth  
## prediction email spam  
## email 903 53  
## spam 50 577

missclassification(confusion.test)

## [1] 0.06506633

### Classification and Regression Trees (CART)

A CART, is not a linear model. It partitions the feature space , thus creating a set of if-then rules for prediction or classification. This view clarifies the name of the function rpart, which *recursively partitions* the feature space.

We start with a regression tree.

library(rpart)  
tree.1 <- rpart(lcavol~., data=prostate.train)  
  
# Train error:  
MSE( predict(tree.1)- prostate.train$lcavol)

## [1] 0.4909568

# Test error:  
MSE( predict(tree.1, newdata = prostate.test)- prostate.test$lcavol)

## [1] 0.5623316

[TODO: plot with rpart.plot]

Tree are very prone to overfitting. To avoid this, we reduce a tree's complexity by *pruning* it. This is done with the prune function (not demonstrated herein).

We now fit a classification tree.

tree.2 <- rpart(spam~., data=spam.train)  
  
# Train confusion matrix:  
.predictions.train <- predict(tree.2, type='class')   
(confusion.train <- table(prediction=.predictions.train, truth=spam.train$spam))

## truth  
## prediction email spam  
## email 1751 185  
## spam 84 998

missclassification(confusion.train)

## [1] 0.08913188

# Test confusion matrix:  
.predictions.test <- predict(tree.2, newdata = spam.test, type='class')   
(confusion.test <- table(prediction=.predictions.test, truth=spam.test$spam))

## truth  
## prediction email spam  
## email 901 125  
## spam 52 505

missclassification(confusion.test)

## [1] 0.111813

### K-nearest neighbour (KNN)

KNN is not an ERM problem. For completeness, we still show how to fit such a hypothesis class.

library(class)  
knn.1 <- knn(train = X.train.spam, test = X.test.spam, cl =y.train.spam, k = 1)  
  
# Test confusion matrix:  
.predictions.test <- knn.1   
(confusion.test <- table(prediction=.predictions.test, truth=spam.test$spam))

## truth  
## prediction email spam  
## email 806 143  
## spam 147 487

missclassification(confusion.test)

## [1] 0.1831965

### Linear Discriminant Analysis (LDA)

LDA is equivalent to least squares classification 9.2.1. There are, however, some dedicated functions to fit it.

library(MASS)   
lda.1 <- lda(spam~., spam.train)  
  
# Train confusion matrix:  
.predictions.train <- predict(lda.1)$class  
(confusion.train <- table(prediction=.predictions.train, truth=spam.train$spam))

## truth  
## prediction email spam  
## email 1755 262  
## spam 80 921

missclassification(confusion.train)

## [1] 0.1133201

# Test confusion matrix:  
.predictions.test <- predict(lda.1, newdata = spam.test)$class  
(confusion.test <- table(prediction=.predictions.test, truth=spam.test$spam))

## truth  
## prediction email spam  
## email 906 130  
## spam 47 500

missclassification(confusion.test)

## [1] 0.111813

### Naive Bayes

A Naive-Bayes classifier is also not part of the ERM framework. It is, however, very popular, so we present it.

library(e1071)  
nb.1 <- naiveBayes(spam~., data = spam.train)  
  
# Train confusion matrix:  
.predictions.train <- predict(nb.1, newdata = spam.train)  
(confusion.train <- table(prediction=.predictions.train, truth=spam.train$spam))

## truth  
## prediction email spam  
## email 970 71  
## spam 865 1112

missclassification(confusion.train)

## [1] 0.3101392

# Test confusion matrix:  
.predictions.test <- predict(nb.1, newdata = spam.test)  
(confusion.test <- table(prediction=.predictions.test, truth=spam.test$spam))

## truth  
## prediction email spam  
## email 518 35  
## spam 435 595

missclassification(confusion.test)

## [1] 0.2969046

## Bibliographic Notes

The ultimate reference on (statistical) machine learning is Friedman, Hastie, and Tibshirani ([2001](#ref-friedman2001elements)). For a softer introduction, see James et al. ([2013](#ref-james2013introduction)). A statistician will also like Ripley ([2007](#ref-ripley2007pattern)). For an R oriented view see Lantz ([2013](#ref-lantz2013machine)). For a very algorithmic view, see the seminal Leskovec, Rajaraman, and Ullman ([2014](#ref-leskovec2014mining)) or Conway and White ([2012](#ref-conway2012machine)). For a much more theoretical reference, see Mohri, Rostamizadeh, and Talwalkar ([2012](#ref-mohri2012foundations)), Vapnik ([2013](#ref-vapnik2013nature)), Shalev-Shwartz and Ben-David ([2014](#ref-shalev2014understanding)). Terminology taken from Sammut and Webb ([2011](#ref-sammut2011encyclopedia)). For a review of resampling based unbiased risk estimation (i.e. cross validation) see the exceptional review of Arlot, Celisse, and others ([2010](#ref-arlot2010survey)).

## Practice Yourself

# Unsupervised Learning

This chapter deals with machine learning problems which are unsupervised. This means the machine has access to a set of inputs, , but the desired outcome, is not available. Clearly, learning a relation between inputs and outcomes is impossible, but there are still a lot of problems of interest. In particular, we may want to find a compact representation of the inputs, be it for visualization of further processing. This is the problem of *dimensionality reduction*. For the same reasons we may want to group similar inputs. This is the problem of *clustering*.

In the statistical terminology, and with some exceptions, this chapter can be thought of as multivariate **exploratory** statistics. For multivariate **inference**, see Chapter 8.

## Dimensionality Reduction

Example 12 Consider the heights and weights of a sample of individuals. The data may seemingly reside in dimensions but given the height, we have a pretty good guess of a persons weight, and vice versa. We can thus state that heights and weights are not really two dimensional, but roughly lay on a dimensional subspace of .

Example 13 Consider the correctness of the answers to a questionnaire with questions. The data may seemingly reside in a dimensional space, but assuming there is a thing as ``skill'', then given the correctness of a person's reply to a subset of questions, we have a good idea how he scores on the rest. Put differently, we don't really need a question questionnaire-- is more than enough. If skill is indeed a one dimensional quality, then the questionnaire data should organize around a single line in the dimensional cube.

Example 14 Consider microphones recording an individual. The digitized recording consists of samples. Are the recordings really a shapeless cloud of points in ? Since they all record the same sound, one would expect the -dimensional points to arrange around the source sound bit: a single point in . If microphones have different distances to the source, volumes may differ. We would thus expect the points to arrange about a **line** in .

### Principal Component Analysis

*Principal Component Analysis* (PCA) is such a basic technique, it has been rediscovered and renamed independently in many fields. It can be found under the names of Discrete Karhunen–Loève Transform; Hotteling Transform; Proper Orthogonal Decomposition; Eckart–Young Theorem; Schmidt–Mirsky Theorem; Empirical Orthogonal Functions; Empirical Eigenfunction Decomposition; Empirical Component Analysis; Quasi-Harmonic Modes; Spectral Decomposition; Empirical Modal Analysis, and possibly more[[22]](#footnote-259). The many names are quite interesting as they offer an insight into the different problems that led to PCA's (re)discovery.

Return to the BMI problem in Example 12. Assume you wish to give each individual a "size score", that is a **linear** combination of height and weight: PCA does just that. It returns the linear combination that has the largest variability, i.e., the combination which best distinguishes between individuals.

The variance maximizing motivation above was the one that guided Hotelling ([1933](#ref-hotelling1933analysis)). But years before him, Pearson ([1901](#ref-pearson1901liii)) derived the same procedure with a different motivation in mind. Pearson was also trying to give each individual a score. He did not care about variance maximization, however. He simply wanted a small set of coordinates in some (linear) space that approximates the original data well.

Before we proceed, we give an example to fix ideas. Consider the crime rate data in USArrests, which encodes reported murder events, assaults, rapes, and the urban population of each american state.

head(USArrests)

## Murder Assault UrbanPop Rape  
## Alabama 13.2 236 58 21.2  
## Alaska 10.0 263 48 44.5  
## Arizona 8.1 294 80 31.0  
## Arkansas 8.8 190 50 19.5  
## California 9.0 276 91 40.6  
## Colorado 7.9 204 78 38.7

Following Hotelling's motivation, we may want to give each state a "crimilality score". We first remove the UrbanPop variable, which does not encode crime levels. We then z-score each variable with scale, and call PCA for a sequence of criminality scores that best separate between states.

USArrests.1 <- USArrests[,-3] %>% scale   
pca.1 <- prcomp(USArrests.1, scale = TRUE)  
pca.1

## Standard deviations (1, .., p=3):  
## [1] 1.5357670 0.6767949 0.4282154  
##   
## Rotation (n x k) = (3 x 3):  
## PC1 PC2 PC3  
## Murder -0.5826006 0.5339532 -0.6127565  
## Assault -0.6079818 0.2140236 0.7645600  
## Rape -0.5393836 -0.8179779 -0.1999436

Things to note:

* Distinguishing between states, i.e., finding the variance maximizing scores, should be indifferent to the **average** of each variable. We also don't want the score to be sensitive to the measurement **scale**. We thus perform PCA in the z-score scale of each variable, obtained with the scale function.
* PCA is performed with the prcomp function. It returns the contribution (weight) of the original variables, to the new crimeness score.  
  These weights are called the *loadings* (or Rotations in the prcomp output, which is rather confusing as we will later see).
* The number of possible scores, is the same as the number of original variables in the data.
* The new scores are called the *principal components*, labeled PC1,...,PC3 in our output.
* The loadings on PC1 tell us that the best separation between states is along the average crime rate. Why is this? Because all the crime variables have a similar loading on PC1.
* The other PCs are slightly harder to interpret, but it is an interesting exercise.

**If we now represent each state, not with its original variables, but only with the first PCs (for example), we have reduced the dimensionality of the data.**

### Dimensionality Reduction Preliminaries

Before presenting methods other than PCA, we need some terminology.

* **Variable**: A.k.a. *dimension*, or *feature*, or *column*.
* **Data**: A.k.a. *sample*, *observations*. Will typically consist of , vectors of dimension . We typically denote the data as a matrix .
* **Manifold**: A generalization of a linear space, which is regular enough so that, **locally**, it has all the properties of a linear space. We will denote an arbitrary manifold by , and by a dimensional[[23]](#footnote-262) manifold.
* **Embedding**: Informally speaking: a "shape preserving" mapping of a space into another (see figure below).
* **Linear Embedding**: An embedding done via a linear operation.
* **Generative Model**: Known to statisticians as the **sampling distribution**. The assumed stochastic process that generated the observed .

Various embedding algorithms. Source: <http://scikit-learn.org/stable/auto_examples/manifold/plot_manifold_sphere.html#sphx-glr-auto-examples-manifold-plot-manifold-sphere-py>

Various embedding algorithms. Source: <http://scikit-learn.org/stable/auto_examples/manifold/plot_manifold_sphere.html#sphx-glr-auto-examples-manifold-plot-manifold-sphere-py>

There are many motivations for dimensionality reduction:

1. **Scoring**: Give each observation an interpretable, simple score (Hotelling's motivation).
2. **Latent structure**: Recover unobservable information from indirect measurements. E.g: Blind signal reconstruction, CT scan, cryo-electron microscopy, etc.
3. **Signal to Noise**: Denoise measurements before further processing like clustering, supervised learning, etc.
4. **Compression**: Save on RAM ,CPU, and communication when operating on a lower dimensional representation of the data.

### Latent Variable Generative Approaches

All generative approaches to dimensionality reduction will include a set of latent/unobservable variables, which we can try to recover from the observables . The unobservable variables will typically have a lower dimension than the observables, thus, dimension is reduced. We start with the simplest case of linear Factor Analysis.

#### Factor Analysis (FA)

FA originates from the psychometric literature. We thus revisit the IQ (actually g-factor[[24]](#footnote-266)) Example 13:

Example 15 Assume respondents answer quantitative questions: . Also assume, their responses are some linear function of a single personality attribute, . We can think of as the subject's ``intelligence''. We thus have

And in matrix notation:

where is the matrix of factor loadings, and the matrix of latent personality traits. In our particular example where , the problem is to recover the unobservable intelligence scores, , from the observed answers .

We may try to estimate by assuming some distribution on and and apply maximum likelihood. Under standard assumptions on the distribution of and , recovering from is still impossible as there are infinitely many such solutions. In the statistical parlance we say the problem is *non identifiable*, and in the applied mathematics parlance we say the problem is *ill posed*. To see this, consider an orthogonal *rotation* matrix (). For each such : . While both solve Eq.(17), and may have very different interpretations. This is why many researchers find FA an unsatisfactory inference tool.

Remark. The non-uniqueness (non-identifiability) of the FA solution under variable rotation is never mentioned in the PCA context. Why is this? This is because the methods solve different problems. The reason the solution to PCA is well defined is that PCA does not seek a single but rather a **sequence** of with dimensions growing from to .

Remark. In classical FA in Eq.(17) is clearly an embedding to a linear space: the one spanned by . Under the classical probabilistic assumptions on and the embedding itself is also linear, and is sometimes solved with PCA. Being a generative model, there is no restriction for the embedding to be linear, and there certainly exists sets of assumptions for which the FA returns a non linear embedding into a linear space.

The FA terminology is slightly different than PCA:

* **Factors**: The unobserved attributes . Akin to the *principal components* in PCA.
* **Loading**: The matrix; the contribution of each factor to the observed .
* **Rotation**: An arbitrary orthogonal re-combination of the factors, , and loadings, , which changes the interpretation of the result.

The FA literature offers several heuristics to "fix" the identifiability problem of FA. These are known as *rotations*, and go under the names of *Varimax*, *Quartimax*, *Equimax*, *Oblimin*, *Promax*, and possibly others.

#### Independent Component Analysis (ICA)

Like FA, *independent compoent analysis* (ICA) is a family of latent space models, thus, a *meta-method*. It assumes data is generated as some function of the latent variables . In many cases this function is assumed to be linear in so that ICA is compared, if not confused, with PCA and even more so with FA.

The fundamental idea of ICA is that has a joint distribution of **non-Gaussian**, **independent** variables. This independence assumption, solves the the non-uniqueness of in FA.

Being a generative model, estimation of can then be done using maximum likelihood, or other estimation principles.

ICA is a popular technique in signal processing, where is actually the signal, such as sound in Example 14. Recovering is thus recovering the original signals mixing in the recorded .

### Purely Algorithmic Approaches

We now discuss dimensionality reduction approaches that are not stated via their generative model, but rather, directly as an algorithm. This does not mean that they cannot be cast via their generative model, but rather they were not motivated as such.

#### Multidimensional Scaling (MDS)

MDS can be thought of as a variation on PCA, that begins with the graph[[25]](#footnote-271) of distances between data points, and not the original data.

MDS aims at embedding a graph of distances, while preserving the original distances. Basic results in graph/network theory (Graham [1988](#ref-graham1988isometric)) suggest that the geometry of a graph cannot be preserved when embedding it into lower dimensions. The different types of MDSs, such as *Classical MDS*, and *Sammon Mappings*, differ in the *stress function* penalizing for geometric distortion.

#### Local Multidimensional Scaling (Local MDS)

Example 16 Consider data of coordinates on the globe. At short distances, constructing a dissimilarity graph with Euclidean distances will capture the true distance between points. At long distances, however, the Euclidean distances as grossly inappropriate. A more extreme example is coordinates on the brain's cerebral cortex. Being a highly folded surface, the Euclidean distance between points is far from the true geodesic distances along the cortex's surface[[26]](#footnote-273).

Local MDS is aimed at solving the case where we don't know how to properly measure distances. It is an algorithm that compounds both the construction of the dissimilarity graph, and the embedding. The solution of local MDS, as the name suggests, rests on the computation of *local* distances, where the Euclidean assumption may still be plausible, and then aggregate many such local distances, before calling upon regular MDS for the embedding.

Because local MDS ends with a regular MDS, it can be seen as a non-linear embedding into a linear .

Local MDS is not popular. Why is this? Because it makes no sense: If we believe the points reside in a non-Euclidean space, thus motivating the use of geodesic distances, why would we want to wrap up with regular MDS, which embeds in a linear space?! It does offer, however, some intuition to the following, more popular, algorithms.

#### Isometric Feature Mapping (IsoMap)

Like localMDS, only that the embedding, and not only the computation of the distances, is local.

#### Local Linear Embedding (LLE)

Very similar to IsoMap 10.1.4.3.

#### Kernel PCA

TODO

#### Simplified Component Technique LASSO (SCoTLASS)

TODO

#### Sparse Principal Component Analysis (sPCA)

TODO

#### Sparse kernel principal component analysis (skPCA)

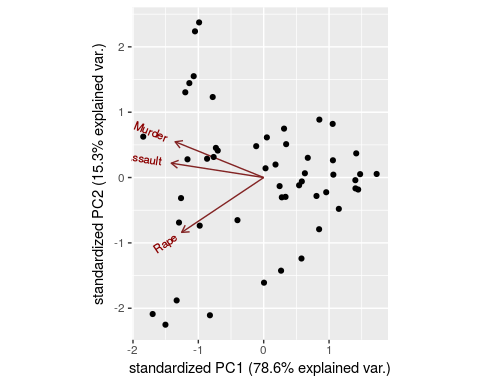
TODO

### Dimensionality Reduction in R

#### PCA

We already saw the basics of PCA in 10.1.1. The fitting is done with the prcomp function. The *bi-plot* is a useful way to visualize the output of PCA.

library(devtools)  
# install\_github("vqv/ggbiplot")  
ggbiplot::ggbiplot(pca.1)

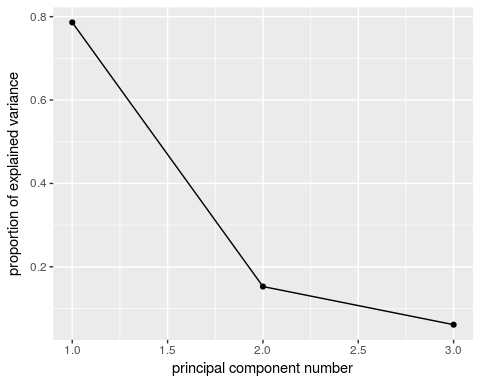


Things to note:

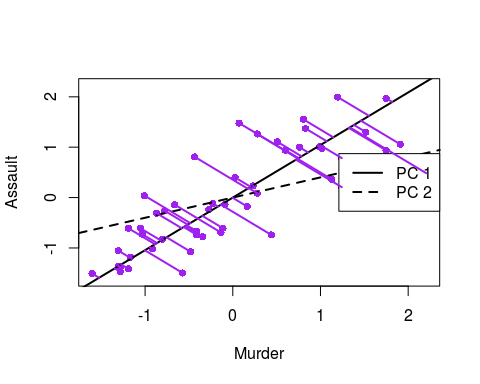
* We used the ggbiplot function from the **ggbiplot** (available from github, but not from CRAN), because it has a nicer output than stats::biplot.
* The bi-plot also plots the loadings as arrows. The coordinates of the arrows belong to the weight of each of the original variables in each PC. For example, the x-value of each arrow is the loadings on the first PC (on the x-axis). Since the weights of Murder, Assault, and Rape are almost the same, we conclude that PC1 captures the average crime rate in each state.
* The bi-plot plots each data point along its PCs.

The *scree plot* depicts the quality of the approximation of as grows. This is depicted using the proportion of variability in that is removed by each added PC. It is customary to choose as the first PC that has a relative low contribution to the approximation of .

ggbiplot::ggscreeplot(pca.1)



See how the first PC captures the variability in the Assault levels and Murder levels, with a single score.



More implementations of PCA:

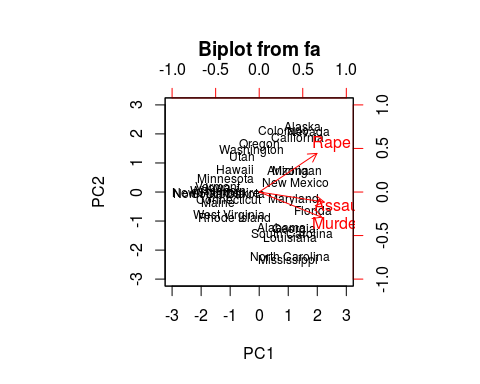
# FAST solutions:  
gmodels::fast.prcomp()  
  
# More detail in output:  
FactoMineR::PCA()  
  
# For flexibility in algorithms and visualization:  
ade4::dudi.pca()  
  
# Another one...  
amap::acp()

#### FA

fa.1 <- psych::principal(USArrests.1, nfactors = 2, rotate = "none")  
fa.1

## Principal Components Analysis  
## Call: psych::principal(r = USArrests.1, nfactors = 2, rotate = "none")  
## Standardized loadings (pattern matrix) based upon correlation matrix  
## PC1 PC2 h2 u2 com  
## Murder 0.89 -0.36 0.93 0.0688 1.3  
## Assault 0.93 -0.14 0.89 0.1072 1.0  
## Rape 0.83 0.55 0.99 0.0073 1.7  
##   
## PC1 PC2  
## SS loadings 2.36 0.46  
## Proportion Var 0.79 0.15  
## Cumulative Var 0.79 0.94  
## Proportion Explained 0.84 0.16  
## Cumulative Proportion 0.84 1.00  
##   
## Mean item complexity = 1.4  
## Test of the hypothesis that 2 components are sufficient.  
##   
## The root mean square of the residuals (RMSR) is 0.05   
## with the empirical chi square 0.87 with prob < NA   
##   
## Fit based upon off diagonal values = 0.99

biplot(fa.1, labels = rownames(USArrests.1))



# Numeric comparison with PCA:  
fa.1$loadings

##   
## Loadings:  
## PC1 PC2   
## Murder 0.895 -0.361  
## Assault 0.934 -0.145  
## Rape 0.828 0.554  
##   
## PC1 PC2  
## SS loadings 2.359 0.458  
## Proportion Var 0.786 0.153  
## Cumulative Var 0.786 0.939

pca.1$rotation

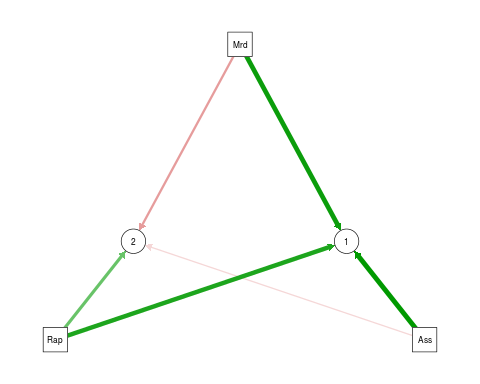
## PC1 PC2 PC3  
## Murder -0.5826006 0.5339532 -0.6127565  
## Assault -0.6079818 0.2140236 0.7645600  
## Rape -0.5393836 -0.8179779 -0.1999436

Things to note:

* We perform FA with the psych::principal function. The Principal Component Analysis title is due to the fact that FA without rotations, is equivalent to PCA.
* The first factor (fa.1$loadings) has different weights than the first PC (pca.1$rotation) because of normalization. They are the same, however, in that the first PC, and the first factor, capture average crime levels.

Graphical model fans will like the following plot, where the contribution of each variable to each factor is encoded in the width of the arrow.

qgraph::qgraph(fa.1)



Let's add a rotation (Varimax), and note that the rotation has indeed changed the loadings of the variables, thus the interpretation of the factors.

fa.2 <- psych::principal(USArrests.1, nfactors = 2, rotate = "varimax")  
  
fa.2$loadings

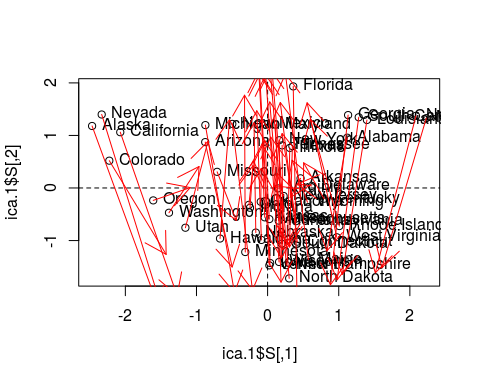
##   
## Loadings:  
## RC1 RC2   
## Murder 0.930 0.257  
## Assault 0.829 0.453  
## Rape 0.321 0.943  
##   
## RC1 RC2  
## SS loadings 1.656 1.160  
## Proportion Var 0.552 0.387  
## Cumulative Var 0.552 0.939

Things to note:

* FA with a rotation is no longer equivalent to PCA.
* The rotated factors are now called *rotated componentes*, and reported in RC1 and RC2.

#### ICA

ica.1 <- fastICA::fastICA(USArrests.1, n.com=2) # Also performs projection pursuit  
  
plot(ica.1$S)  
abline(h=0, v=0, lty=2)  
text(ica.1$S, pos = 4, labels = rownames(USArrests.1))  
  
# Compare with PCA (first two PCs):  
arrows(x0 = ica.1$S[,1], y0 = ica.1$S[,2], x1 = pca.1$x[,2], y1 = pca.1$x[,1], col='red', pch=19, cex=0.5)



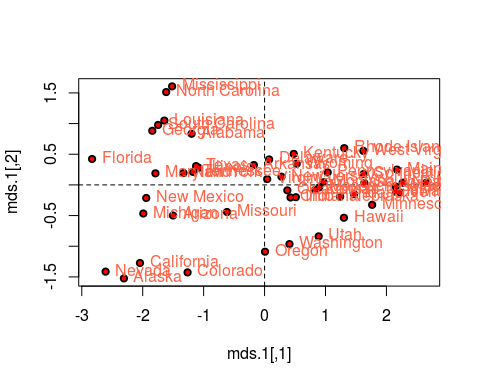
Things to note:

* ICA is fitted with fastICA::fastICA.
* The ICA components, like any other rotated components, are different than the PCA components.

#### MDS

Classical MDS, also compared with PCA.

# We first need a dissimarity matrix/graph:  
state.disimilarity <- dist(USArrests.1)  
  
mds.1 <- cmdscale(state.disimilarity)  
  
plot(mds.1, pch = 19)  
abline(h=0, v=0, lty=2)  
USArrests.2 <- USArrests[,1:2] %>% scale  
text(mds.1, pos = 4, labels = rownames(USArrests.2), col = 'tomato')  
  
# Compare with PCA (first two PCs):  
points(pca.1$x[,1:2], col='red', pch=19, cex=0.5)

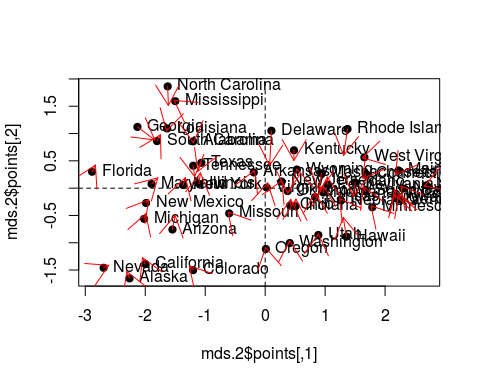


Things to note:

* We first compute a dissimilarity graph with dist. See the cluster::daisy function for more dissimilarity measures.
* We learn the MDS embedding with cmdscale.
* The embedding of PCA is the same as classical MDS with Euclidean distances.

Let's try other strain functions for MDS, like Sammon's strain, and compare it with the PCs.

mds.2 <- MASS::sammon(state.disimilarity, trace = FALSE)  
plot(mds.2$points, pch = 19)  
abline(h=0, v=0, lty=2)  
text(mds.2$points, pos = 4, labels = rownames(USArrests.2))  
  
# Compare with PCA (first two PCs):  
arrows(  
 x0 = mds.2$points[,1], y0 = mds.2$points[,2],   
 x1 = pca.1$x[,1], y1 = pca.1$x[,2],   
 col='red', pch=19, cex=0.5)



Things to note:

* MASS::sammon does the embedding.
* Sammon strain is different than PCA.

#### Sparse PCA

# Compute similarity graph  
state.similarity <- MASS::cov.rob(USArrests.1)$cov  
  
spca1 <- elasticnet::spca(state.similarity, K=2, type="Gram", sparse="penalty", trace=FALSE, para=c(0.06,0.16))  
spca1$loadings

## PC1 PC2  
## Murder -0.6683748 0  
## Assault -0.7345473 0  
## Rape -0.1171131 -1

#### Kernel PCA

kernlab::kpca()

## Clustering

Example 17 Consider the tagging of your friends' pictures on Facebook. If you tagged some pictures, Facebook may try to use a supervised approach to automatically label photos. If you never tagged pictures, a supervised approach is impossible. It is still possible, however, to group simiar pictures together.

Example 18 Consider the problem of spam detection. It would be nice if each user could label several thousands emails, to apply a supervised learning approach to spam detection. This is an unrealistic demand, so a pre-clustering stage is useful: the user only needs to tag a couple dozens of homogenous clusters, before solving the supervised learning problem.

In clustering problems, we seek to group observations that are similar.

There are many motivations for clustering:

1. **Understanding**: The most common use of clustering is probably as a an exploratory step, to identify homogeneous groups in the data.
2. **Dimensionality reduction**: Clustering may be seen as a method for dimensionality reduction. Unlike the approaches in the Dimensionality Reduction Section 10.1, it does not compress **variables** but rather **observations**. Each group of homogeneous observations may then be represented as a single prototypical observation of the group.
3. **Pre-Labelling**: Clustering may be performed as a pre-processing step for supervised learning, when labeling all the samples is impossible due to "budget" constraints, like in Example 18. This is sometimes known as *pre-clustering*.

Clustering, like dimensionality reduction, may rely on some latent variable generative model, or on purely algorithmic approaches.

### Latent Variable Generative Approaches

#### Finite Mixture

Example 19 Consider the distribution of heights. Heights have a nice bell shaped distribution within each gender. If genders have not been recorded, heights will be distributed like a *mixture* of males and females. The gender in this example, is a *latent* variable taking levels: male and female.

A *finite mixture* is the marginal distribution of distinct classes, when the class variable is *latent*. This is useful for clustering: We can assume the number of classes, , and the distribution of each class. We then use maximum likelihood to fit the mixture distribution, and finally, cluster by assigning observations to the most probable class.

### Purely Algorithmic Approaches

#### K-Means

The *K-means* algorithm is possibly the most popular clustering algorithm. The goal behind K-means clustering is finding a representative point for each of K clusters, and assign each data point to one of these clusters. As each cluster has a representative point, this is also a *prototype method*. The clusters are defined so that they minimize the average Euclidean distance between all points to the center of the cluster.

In K-means, the clusters are first defined, and then similarities computed. This is thus a *top-down* method.

K-means clustering requires the raw features as inputs, and not only a similarity graph. This is evident when examining the algorithm below.

The k-means algorithm works as follows:

1. Choose the number of clusters .
2. Arbitrarily assign points to clusters.
3. While clusters keep changing:
   1. Compute the cluster centers as the average of their points.
   2. Assign each point to its closest cluster center (in Euclidean distance).
4. Return Cluster assignments and means.

Remark. If trained as a statistician, you may wonder- what population quantity is K-means actually estimating? The estimand of K-means is known as the *K principal points*. Principal points are points which are *self consistent*, i.e., they are the mean of their neighbourhood.

#### K-Means++

*K-means++* is a fast version of K-means thanks to a smart initialization.

#### K-Medoids

If a Euclidean distance is inappropriate for a particular set of variables, or that robustness to corrupt observations is required, or that we wish to constrain the cluster centers to be actual observations, then the *K-Medoids* algorithm is an adaptation of K-means that allows this. It is also known under the name *partition around medoids* (PAM) clustering, suggesting its relation to [graph partitioning](https://en.wikipedia.org/wiki/Graph_partition).

The k-medoids algorithm works as follows.

1. Given a dissimilarity graph.
2. Choose the number of clusters .
3. Arbitrarily assign points to clusters.
4. While clusters keep changing:
   1. Within each cluster, set the center as the data point that minimizes the sum of distances to other points in the cluster.
   2. Assign each point to its closest cluster center.
5. Return Cluster assignments and centers.

Remark. If trained as a statistician, you may wonder- what population quantity is K-medoids actually estimating? The estimand of K-medoids is the median of their neighbourhood. A delicate matter is that quantiles are not easy to define for **multivariate** variables so that the "multivaraitre median", may be a more subtle quantity than you may think. See Small ([1990](#ref-small1990survey)).

#### Hirarchial Clustering

Hierarchical clustering algorithms take dissimilarity graphs as inputs. Hierarchical clustering is a class of greedy *graph-partitioning* algorithms. Being hierarchical by design, they have the attractive property that the evolution of the clustering can be presented with a *dendogram*, i.e., a tree plot.  
A particular advantage of these methods is that they do not require an a-priori choice of the number of cluster ().

Two main sub-classes of algorithms are *agglomerative*, and *divisive*.

*Agglomerative clustering* algorithms are **bottom-up** algorithm which build clusters by joining smaller clusters. To decide which clusters are joined at each iteration some measure of closeness between clusters is required.

* **Single Linkage**: Cluster distance is defined by the distance between the two **closest** members.
* **Complete Linkage**: Cluster distance is defined by the distance between the two **farthest** members.
* **Group Average**: Cluster distance is defined by the **average** distance between members.
* **Group Median**: Like Group Average, only using the median.

*Divisive clustering* algorithms are **top-down** algorithm which build clusters by splitting larger clusters.

#### Fuzzy Clustering

Can be thought of as a purely algorithmic view of the finite-mixture in Section 10.2.1.1.

### Clustering in R

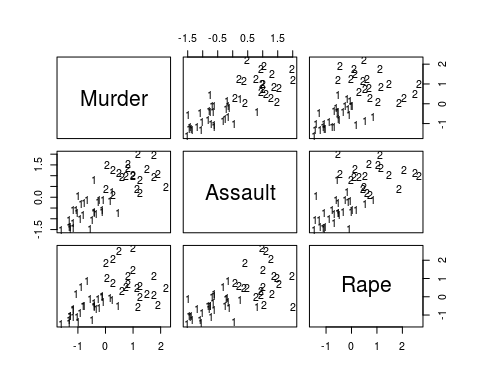
#### K-Means

The following code is an adaptation from [David Hitchcock](http://people.stat.sc.edu/Hitchcock/chapter6_R_examples.txt).

k <- 2  
kmeans.1 <- stats::kmeans(USArrests.1, centers = k)  
head(kmeans.1$cluster) # cluster asignments

## Alabama Alaska Arizona Arkansas California Colorado   
## 2 2 2 1 2 2

pairs(USArrests.1, panel=function(x,y) text(x,y,kmeans.1$cluster))



Things to note:

* The stats::kmeans function does the clustering.
* The cluster assignment is given in the cluster element of the stats::kmeans output.
* The visual inspection confirms that similar states have been assigned to the same cluster.

#### K-Means ++

*K-Means++* is a smart initialization for K-Means. The following code is taken from the [r-help](https://stat.ethz.ch/pipermail/r-help/2012-January/300051.html) mailing list.

# Write my own K-means++ function.  
kmpp <- function(X, k) {  
   
 n <- nrow(X)  
 C <- numeric(k)  
 C[1] <- sample(1:n, 1)  
   
 for (i in 2:k) {  
 dm <- pracma::distmat(X, X[C, ])  
 pr <- apply(dm, 1, min); pr[C] <- 0  
 C[i] <- sample(1:n, 1, prob = pr)  
 }  
   
 kmeans(X, X[C, ])  
}  
  
kmeans.2 <- kmpp(USArrests.1, k)  
head(kmeans.2$cluster)

## Alabama Alaska Arizona Arkansas California Colorado   
## 1 1 1 2 1 1

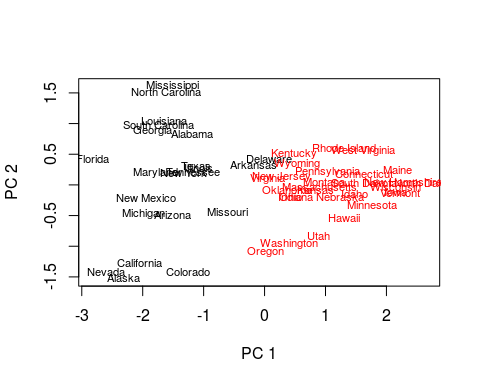
#### K-Medoids

Start by growing a distance graph with dist and then partition using pam.

state.disimilarity <- dist(USArrests.1)  
kmed.1 <- cluster::pam(x= state.disimilarity, k=2)  
head(kmed.1$clustering)

## Alabama Alaska Arizona Arkansas California Colorado   
## 1 1 1 1 1 1

plot(pca.1$x[,1], pca.1$x[,2], xlab="PC 1", ylab="PC 2", type ='n', lwd=2)  
text(pca.1$x[,1], pca.1$x[,2], labels=rownames(USArrests.1), cex=0.7, lwd=2, col=kmed.1$cluster)



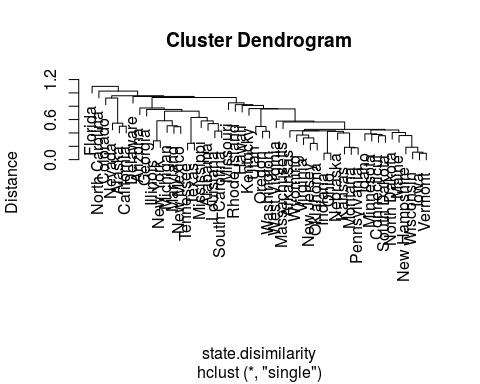
Things to note:

* K-medoids starts with the computation of a dissimilarity graph, done by the dist function.
* The clustering is done by the cluster::pam function.
* Inspecting the output confirms that similar states have been assigned to the same cluster.
* Many other similarity measures can be found in proxy::dist().
* See cluster::clara() for a big-data implementation of PAM.

#### Hirarchial Clustering

We start with agglomerative clustering with single-linkage.

hirar.1 <- hclust(state.disimilarity, method='single')  
plot(hirar.1, labels=rownames(USArrests.1), ylab="Distance")

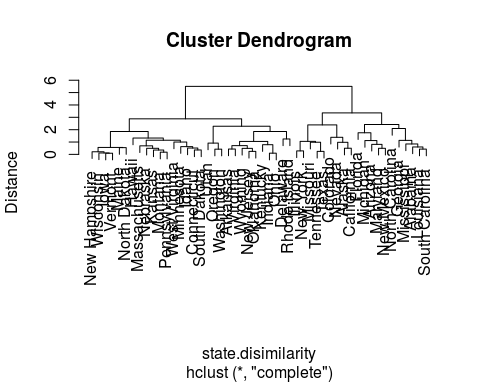


Things to note:

* The clustering is done with the hclust function.
* We choose the single-linkage distance using the method='single' argument.
* We did not need to a-priori specify the number of clusters, , since all the possible 's are included in the output tree.
* The plot function has a particular method for hclust class objects, and plots them as dendograms.

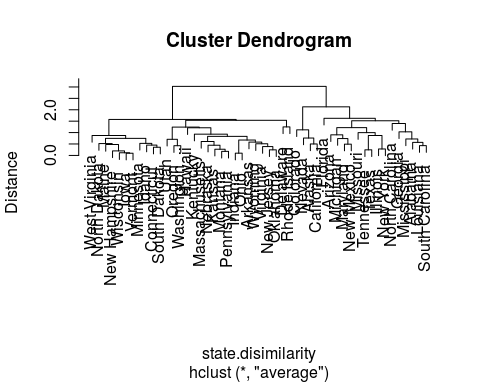
We try other types of linkages, to verify that the indeed affect the clustering. Starting with complete linkage.

hirar.2 <- hclust(state.disimilarity, method='complete')  
plot(hirar.2, labels=rownames(USArrests.1), ylab="Distance")



Now with average linkage.

hirar.3 <- hclust(state.disimilarity, method='average')  
plot(hirar.3, labels=rownames(USArrests.1), ylab="Distance")



If we know how many clusters we want, we can use cuttree to get the class assignments.

cut.2.2 <- cutree(hirar.2, k=2)  
head(cut.2.2)

## Alabama Alaska Arizona Arkansas California Colorado   
## 1 1 1 2 1 1

## Bibliographic Notes

For more on PCA see my [Dimensionality Reduction Class Notes](https://github.com/johnros/dim_reduce/blob/master/dim_reduce.pdf) and references therein. For more on everything, see Friedman, Hastie, and Tibshirani ([2001](#ref-friedman2001elements)). For a softer introduction, see James et al. ([2013](#ref-james2013introduction)).

## Practice Yourself

# Plotting

Whether you are doing EDA, or preparing your results for publication, you need plots. R has many plotting mechanisms, allowing the user a tremendous amount of flexibility, while abstracting away a lot of the tedious details. To be concrete, many of the plots in R are simply impossible to produce with Excel, SPSS, or SAS, and would take a tremendous amount of work to produce with Python, Java and lower level programming languages.

In this text, we will focus on two plotting packages. The basic **graphics** package, distributed with the base R distribution, and the **ggplot2** package.

Before going into the details of the plotting packages, we start with some high-level philosophy. The **graphics** package originates from the mainframe days. Computers had no graphical interface, and the output of the plot was immediately sent to a printer. Once a plot has been produced with the **graphics** package, just like a printed output, it cannot be queried nor changed, except for further additions.

The philosophy of R is that **everyting is an object**. The **graphics** package does not adhere to this philosophy, and indeed it was soon augmented with the **grid** package (R Core Team [2016](#ref-Rlanguage)), that treats plots as objects. **grid** is a low level graphics interface, and users may be more familiar with the **lattice** package built upon it (Sarkar [2008](#ref-lattice)).

**lattice** is very powerful, but soon enough, it was overtaken in popularity by the **ggplot2** package (Wickham [2009](#ref-ggplot2)). **ggplot2** was the PhD project of [Hadley Wickham](http://hadley.nz/), a name to remember... Two fundamental ideas underlay **ggplot2**: (i) everything is an object, and (ii), plots can be described by a simple grammar, i.e., a language to describe the building blocks of the plot. The grammar in **ggplot2** are is the one stated by L. Wilkinson ([2006](#ref-wilkinson2006grammar)). The objects and grammar of **ggplot2** have later evolved to allow more complicated plotting and in particular, interactive plotting.

Interactive plotting is a very important feature for EDA, and reporting. The major leap in interactive plotting was made possible by the advancement of web technologies, such as JavaScript. Why is this? Because an interactive plot, or report, can be seen as a web-site. Building upon the capabilities of JavaScript and your web browser to provide the interactivity, greatly facilitates the development of such plots, as the programmer can rely on the web-browsers capabilities for interactivity.

## The graphics System

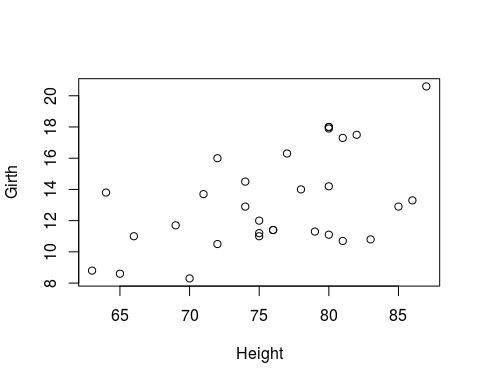
The R code from the Basics Chapter 3 is a demonstration of the **graphics** package and system. We make a quick review of the basics.

### Using Existing Plotting Functions

#### Scatter Plot

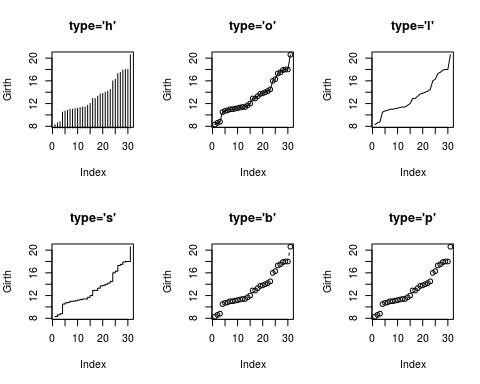
A simple scatter plot.

attach(trees)  
plot(Girth ~ Height)



Various types of plots.

par.old <- par(no.readonly = TRUE)  
par(mfrow=c(2,3))  
plot(Girth, type='h', main="type='h'")   
plot(Girth, type='o', main="type='o'")   
plot(Girth, type='l', main="type='l'")  
plot(Girth, type='s', main="type='s'")  
plot(Girth, type='b', main="type='b'")  
plot(Girth, type='p', main="type='p'")



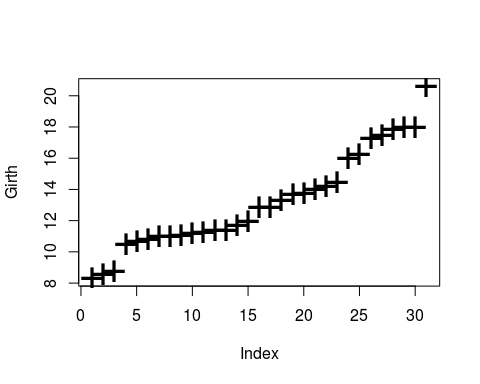
par(par.old)

Things to note:

* The par command controls the plotting parameters. mfrow=c(2,3) is used to produce a matrix of plots with 2 rows and 3 columns.
* The par.old object saves the original plotting setting. It is restored after plotting using par(par.old).
* The type argument controls the type of plot.
* The main argument controls the title.
* See ?plot and ?par for more options.

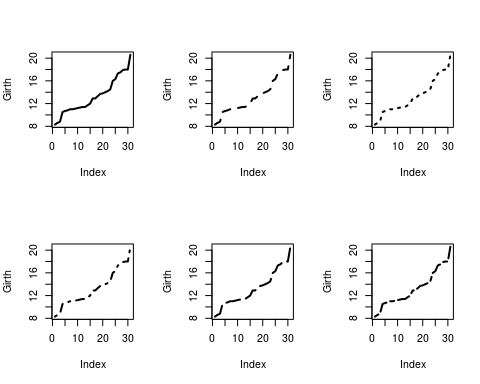
Control the plotting characters with the pch argument, and size with the cex argument.

plot(Girth, pch='+', cex=3)



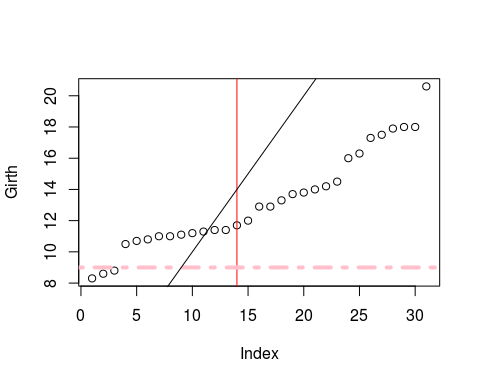
Control the line's type with lty argument, and width with lwd.

par(mfrow=c(2,3))  
plot(Girth, type='l', lty=1, lwd=2)  
plot(Girth, type='l', lty=2, lwd=2)  
plot(Girth, type='l', lty=3, lwd=2)  
plot(Girth, type='l', lty=4, lwd=2)  
plot(Girth, type='l', lty=5, lwd=2)  
plot(Girth, type='l', lty=6, lwd=2)

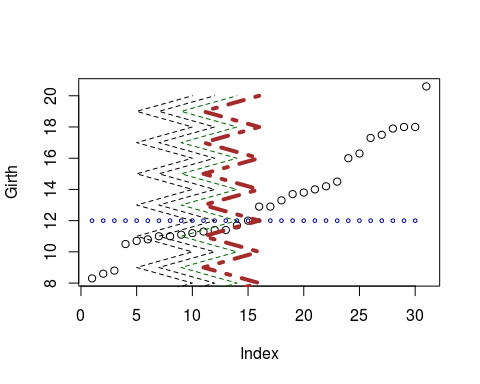


Add line by slope and intercept with abline.

plot(Girth)  
abline(v=14, col='red') # vertical line at 14.  
abline(h=9, lty=4,lwd=4, col='pink') # horizontal line at 9.  
abline(a = 0, b=1) # linear line with intercept a=0, and slope b=1.



plot(Girth)  
points(x=1:30, y=rep(12,30), cex=0.5, col='darkblue')  
lines(x=rep(c(5,10), 7), y=7:20, lty=2 )  
lines(x=rep(c(5,10), 7)+2, y=7:20, lty=2 )  
lines(x=rep(c(5,10), 7)+4, y=7:20, lty=2 , col='darkgreen')  
lines(x=rep(c(5,10), 7)+6, y=7:20, lty=4 , col='brown', lwd=4)

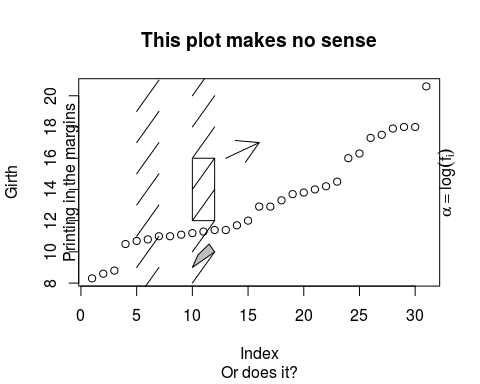


Things to note:

* points adds points on an existing plot.
* lines adds lines on an existing plot.
* col controls the color of the element. It takes names or numbers as argument.
* cex controls the scale of the element. Defaults to cex=1.

Add other elements.

plot(Girth)  
segments(x0=rep(c(5,10), 7), y0=7:20, x1=rep(c(5,10), 7)+2, y1=(7:20)+2 ) # line segments  
arrows(x0=13,y0=16,x1=16,y1=17) # arrows  
rect(xleft=10, ybottom=12, xright=12, ytop=16) # rectangle  
polygon(x=c(10,11,12,11.5,10.5), y=c(9,9.5,10,10.5,9.8), col='grey') # polygon  
title(main='This plot makes no sense', sub='Or does it?')   
mtext('Printing in the margins', side=2) # math text  
mtext(expression(alpha==log(f[i])), side=4)

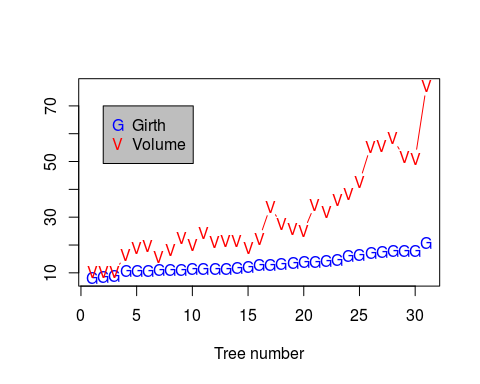


Things to note:

* The following functions add the elements they are named after: segments, arrows, rect, polygon, title.
* mtext adds mathematical text, which needs to be wrapped in expression(). For more information for mathematical annotation see ?plotmath.

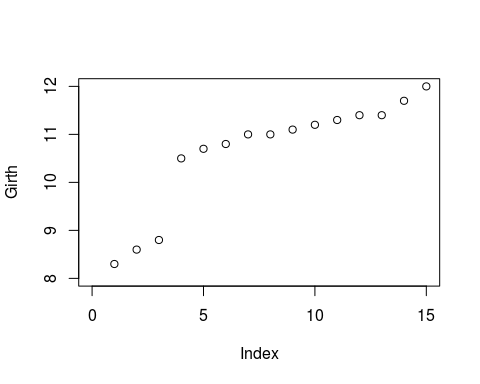
Add a legend.

plot(Girth, pch='G',ylim=c(8,77), xlab='Tree number', ylab='', type='b', col='blue')  
points(Volume, pch='V', type='b', col='red')  
legend(x=2, y=70, legend=c('Girth', 'Volume'), pch=c('G','V'), col=c('blue','red'), bg='grey')



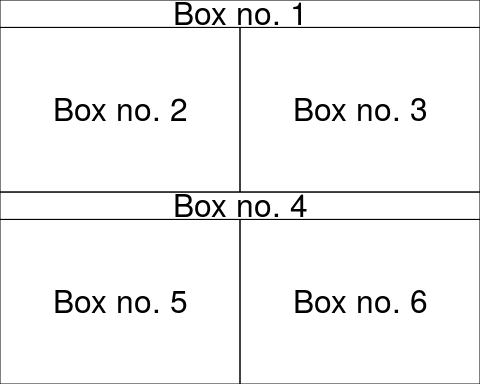
Adjusting Axes with xlim and ylim.

plot(Girth, xlim=c(0,15), ylim=c(8,12))



Use layout for complicated plot layouts.

A<-matrix(c(1,1,2,3,4,4,5,6), byrow=TRUE, ncol=2)  
layout(A,heights=c(1/14,6/14,1/14,6/14))  
  
oma.saved <- par("oma")  
par(oma = rep.int(0, 4))  
par(oma = oma.saved)  
o.par <- par(mar = rep.int(0, 4))  
for (i in seq\_len(6)) {  
 plot.new()  
 box()  
 text(0.5, 0.5, paste('Box no.',i), cex=3)  
}



Always detach.

detach(trees)

### Exporting a Plot

The pipeline for exporting graphics is similar to the export of data. Instead of the write.table or save functions, we will use the pdf, tiff, png, functions. Depending on the type of desired output.

Check and set the working directory.

getwd()  
setwd("/tmp/")

Export tiff.

tiff(filename='graphicExample.tiff')  
plot(rnorm(100))  
dev.off()

Things to note:

* The tiff function tells R to open a .tiff file, and write the output of a plot.
* Only a single (the last) plot is saved.
* dev.off to close the tiff device, and return the plotting to the R console (or RStudio).

If you want to produce several plots, you can use a counter in the file's name. The counter uses the [printf](https://en.wikipedia.org/wiki/Printf_format_string) format string.

tiff(filename='graphicExample%d.tiff') #Creates a sequence of files  
plot(rnorm(100))  
boxplot(rnorm(100))  
hist(rnorm(100))  
dev.off()

## png   
## 2

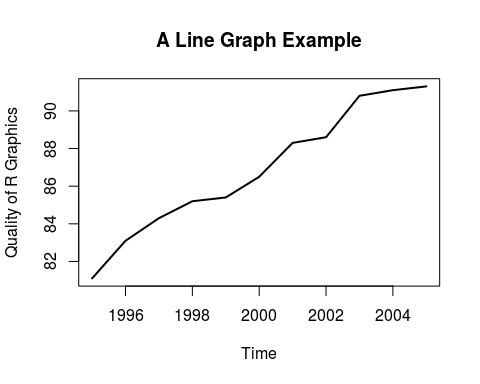
To see the list of all open devices use dev.list(). To close **all** device, (not only the last one), use graphics.off().

See ?pdf and ?jpeg for more info.

### Fancy graphics Examples

Building a line graph from scratch.

x = 1995:2005  
y = c(81.1, 83.1, 84.3, 85.2, 85.4, 86.5, 88.3, 88.6, 90.8, 91.1, 91.3)  
plot.new()  
plot.window(xlim = range(x), ylim = range(y))  
abline(h = -4:4, v = -4:4, col = "lightgrey")  
lines(x, y, lwd = 2)  
title(main = "A Line Graph Example",  
 xlab = "Time",  
 ylab = "Quality of R Graphics")  
axis(1)  
axis(2)  
box()

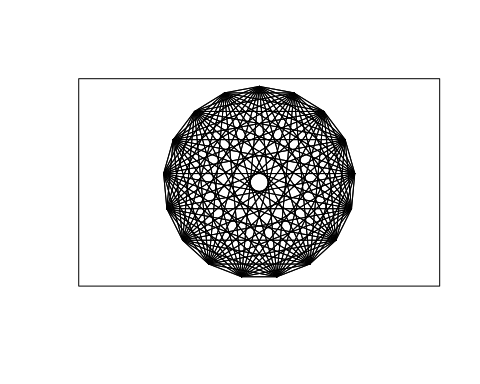


Things to note:

* plot.new creates a new, empty, plotting device.
* plot.window determines the limits of the plotting region.
* axis adds the axes, and box the framing box.
* The rest of the elements, you already know.

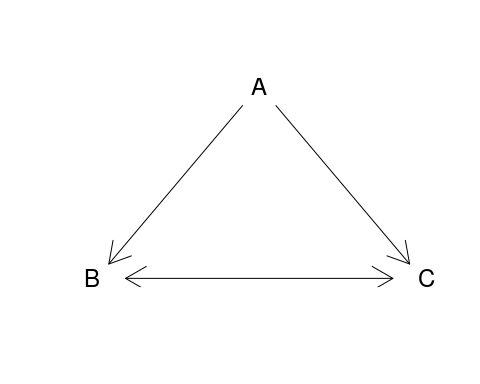
Rosette.

n = 17  
theta = seq(0, 2 \* pi, length = n + 1)[1:n]  
x = sin(theta)  
y = cos(theta)  
v1 = rep(1:n, n)  
v2 = rep(1:n, rep(n, n))  
plot.new()  
plot.window(xlim = c(-1, 1), ylim = c(-1, 1), asp = 1)  
segments(x[v1], y[v1], x[v2], y[v2])  
box()



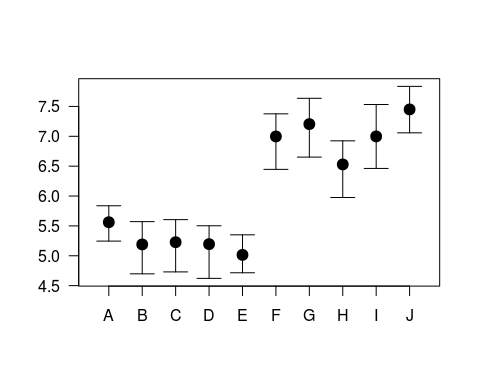
Arrows.

plot.new()  
plot.window(xlim = c(0, 1), ylim = c(0, 1))  
arrows(.05, .075, .45, .9, code = 1)  
arrows(.55, .9, .95, .075, code = 2)  
arrows(.1, 0, .9, 0, code = 3)  
text(.5, 1, "A", cex = 1.5)  
text(0, 0, "B", cex = 1.5)  
text(1, 0, "C", cex = 1.5)



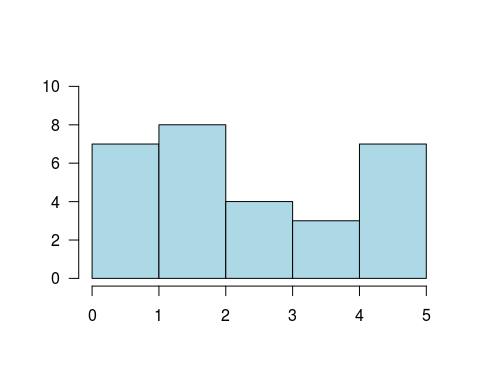
Arrows as error bars.

x = 1:10  
y = runif(10) + rep(c(5, 6.5), c(5, 5))  
yl = y - 0.25 - runif(10)/3  
yu = y + 0.25 + runif(10)/3  
plot.new()  
plot.window(xlim = c(0.5, 10.5), ylim = range(yl, yu))  
arrows(x, yl, x, yu, code = 3, angle = 90, length = .125)  
points(x, y, pch = 19, cex = 1.5)  
axis(1, at = 1:10, labels = LETTERS[1:10])  
axis(2, las = 1)  
box()



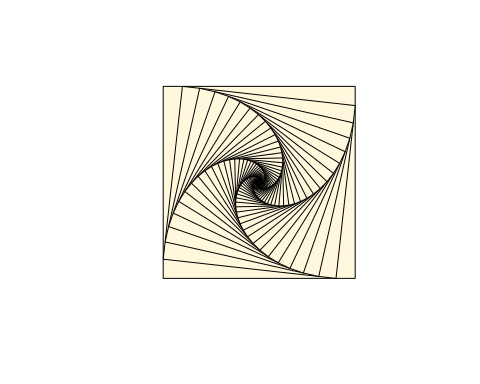
A histogram is nothing but a bunch of rectangle elements.

plot.new()  
plot.window(xlim = c(0, 5), ylim = c(0, 10))  
rect(0:4, 0, 1:5, c(7, 8, 4, 3), col = "lightblue")  
axis(1)  
axis(2, las = 1)



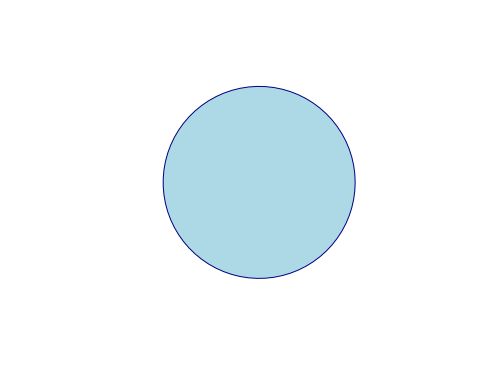
Spiral Squares.

plot.new()  
plot.window(xlim = c(-1, 1), ylim = c(-1, 1), asp = 1)  
x = c(-1, 1, 1, -1)  
y = c( 1, 1, -1, -1)  
polygon(x, y, col = "cornsilk")  
vertex1 = c(1, 2, 3, 4)  
vertex2 = c(2, 3, 4, 1)  
for(i in 1:50) {  
 x = 0.9 \* x[vertex1] + 0.1 \* x[vertex2]  
 y = 0.9 \* y[vertex1] + 0.1 \* y[vertex2]  
 polygon(x, y, col = "cornsilk")  
}



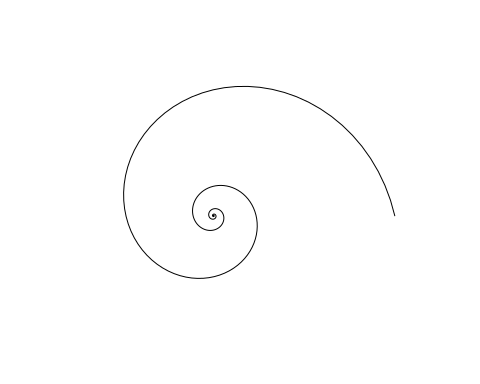
Circles are just dense polygons.

R = 1  
xc = 0  
yc = 0  
n = 72  
t = seq(0, 2 \* pi, length = n)[1:(n-1)]  
x = xc + R \* cos(t)  
y = yc + R \* sin(t)  
plot.new()  
plot.window(xlim = range(x), ylim = range(y), asp = 1)  
polygon(x, y, col = "lightblue", border = "navyblue")



Spiral- just a bunch of lines.

k = 5  
n = k \* 72  
theta = seq(0, k \* 2 \* pi, length = n)  
R = .98^(1:n - 1)  
x = R \* cos(theta)  
y = R \* sin(theta)  
plot.new()  
plot.window(xlim = range(x), ylim = range(y), asp = 1)  
lines(x, y)



## The ggplot2 System

The philosophy of **ggplot2** is very different from the **graphics** device. Recall, in **ggplot2**, a plot is a object. It can be queried, it can be changed, and among other things, it can be plotted.

**ggplot2** provides a convenience function for many plots: qplot. We take a non-typical approach by ignoring qplot, and presenting the fundamental building blocks. Once the building blocks have been understood, mastering qplot will be easy.

The following is taken from [UCLA's idre](http://www.ats.ucla.edu/stat/r/seminars/ggplot2_intro/ggplot2_intro.htm).

A **ggplot2** object will have the following elements:

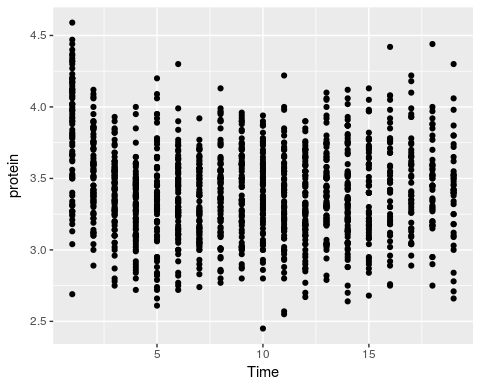
* **Data** the data frame holding the data to be plotted.
* **Aes** defines the mapping between variables to their visualization.
* **Geoms** are the objects/shapes you add as layers to your graph.
* **Stats** are statistical transformations when you are not plotting the raw data, such as the mean or confidence intervals.
* **Faceting** splits the data into subsets to create multiple variations of the same graph (paneling).

The nlme::Milk dataset has the protein level of various cows, at various times, with various diets.

library(nlme)  
data(Milk)  
head(Milk)

## Grouped Data: protein ~ Time | Cow  
## protein Time Cow Diet  
## 1 3.63 1 B01 barley  
## 2 3.57 2 B01 barley  
## 3 3.47 3 B01 barley  
## 4 3.65 4 B01 barley  
## 5 3.89 5 B01 barley  
## 6 3.73 6 B01 barley

library(ggplot2)  
ggplot(data = Milk, aes(x=Time, y=protein)) +  
 geom\_point()

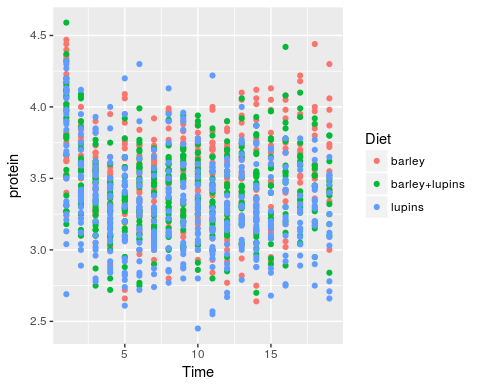


Things to note:

* The ggplot function is the constructor of the **ggplot2** object. If the object is not assigned, it is plotted.
* The aes argument tells R that the Time variable in the Milk data is the x axis, and protein is y.
* The geom\_point defines the **Geom**, i.e., it tells R to plot the points as they are (and not lines, histograms, etc.).
* The **ggplot2** object is build by compounding its various elements separated by the + operator.
* All the variables that we will need are assumed to be in the Milk data frame. This means that (a) the data needs to be a data frame (not a matrix for instance), and (b) we will not be able to use variables that are not in the Milk data frame.

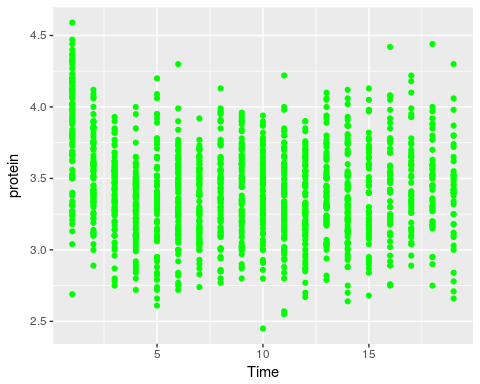
Let's add some color.

ggplot(data = Milk, aes(x=Time, y=protein)) +  
 geom\_point(aes(color=Diet))



The color argument tells R to use the variable Diet as the coloring. A legend is added by default. If we wanted a fixed color, and not a variable dependent color, color would have been put outside the aes function.

ggplot(data = Milk, aes(x=Time, y=protein)) +  
 geom\_point(color="green")

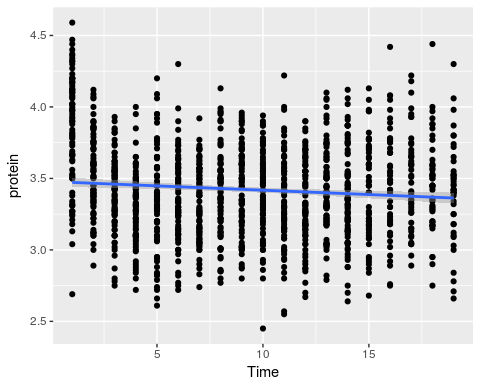


Let's save the **ggplot2** object so we can reuse it. Notice it is not plotted.

p <- ggplot(data = Milk, aes(x=Time, y=protein)) +  
 geom\_point()

We can add *layers* of new *geoms* using the + operator. Here, we add a smoothing line.

p + geom\_smooth(method = 'gam')

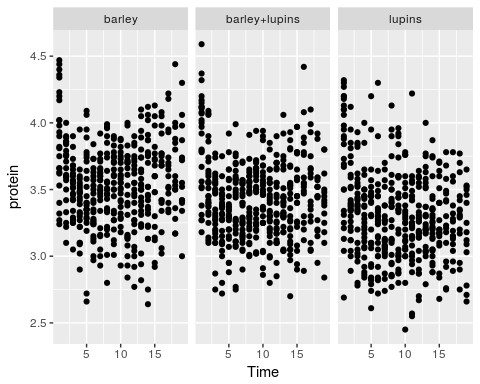


Things to note:

* The smoothing line is a layer added with the geom\_smooth() function.
* Lacking any arguments, the new layer will inherit the aes of the original object, x and y variables in particular.

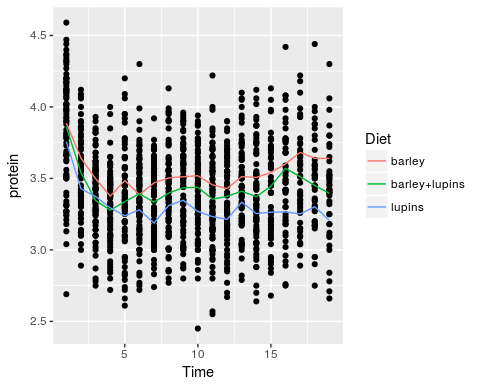
To split the plot along some variable, we use faceting, done with the facet\_wrap function.

p + facet\_wrap(~Diet)



Instead of faceting, we can add a layer of the mean of each Diet subgroup, connected by lines.

p + stat\_summary(aes(color=Diet), fun.y="mean", geom="line")



Things to note:

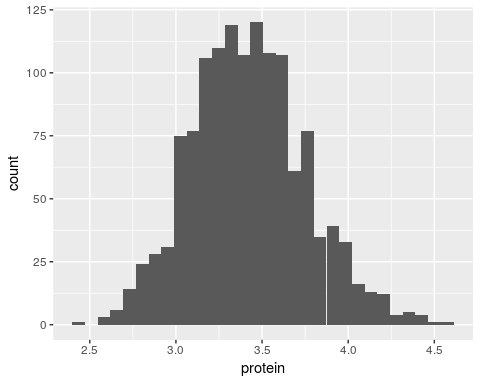
* stat\_summary adds a statistical summary.
* The summary is applied along Diet subgroups, because of the color=Diet aesthetic.
* The summary to be applied is the mean, because of fun.y="mean".
* The group means are connected by lines, because of the geom="line" argument.

What layers can be added using the **geoms** family of functions?

* geom\_bar: bars with bases on the x-axis.
* geom\_boxplot: boxes-and-whiskers.
* geom\_errorbar: T-shaped error bars.
* geom\_histogram: histogram.
* geom\_line: lines.
* geom\_point: points (scatterplot).
* geom\_ribbon: bands spanning y-values across a range of x-values.
* geom\_smooth: smoothed conditional means (e.g. loess smooth).

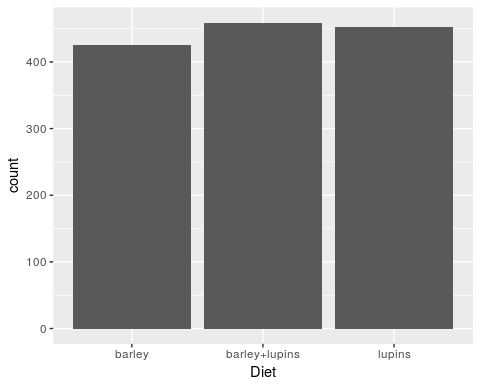
To demonstrate the layers added with the geoms\_\* functions, we start with a histogram.

pro <- ggplot(Milk, aes(x=protein))  
pro + geom\_histogram(bins=30)



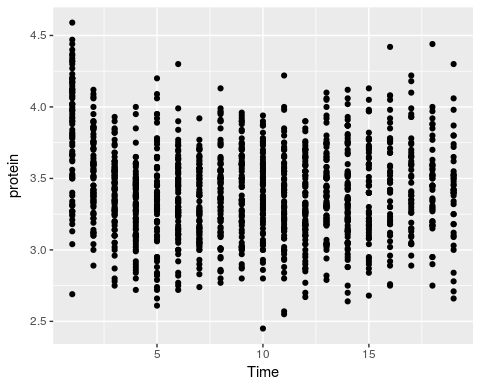
A bar plot.

ggplot(Milk, aes(x=Diet)) +  
 geom\_bar()



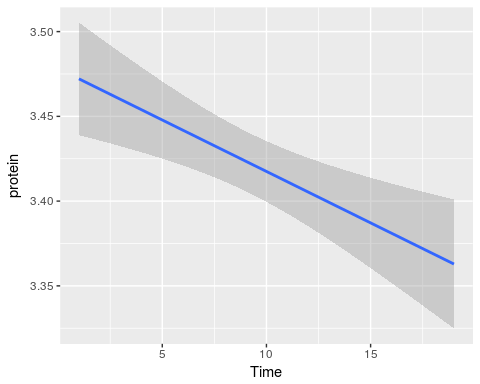
A scatter plot.

tp <- ggplot(Milk, aes(x=Time, y=protein))  
tp + geom\_point()



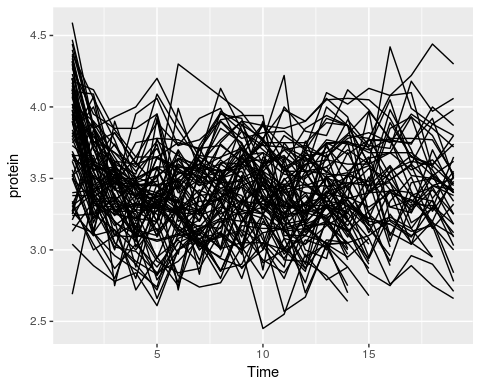
A smooth regression plot, reusing the tp object.

tp + geom\_smooth(method='gam')



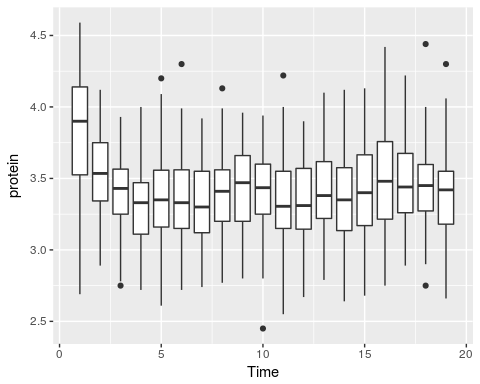
And now, a simple line plot, reusing the tp object, and connecting lines along Cow.

tp + geom\_line(aes(group=Cow))



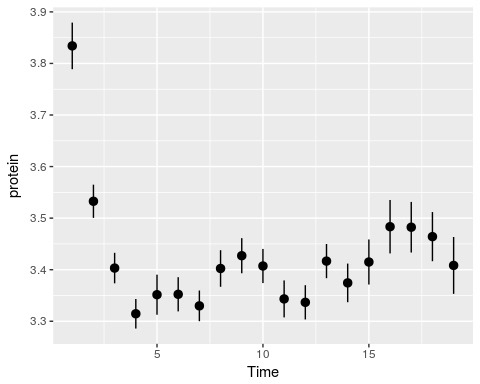
The line plot is completely incomprehensible. Better look at boxplots along time (even if omitting the Cow information).

tp + geom\_boxplot(aes(group=Time))



We can do some statistics for each subgroup. The following will compute the mean and standard errors of protein at each time point.

ggplot(Milk, aes(x=Time, y=protein)) +  
 stat\_summary(fun.data = 'mean\_se')

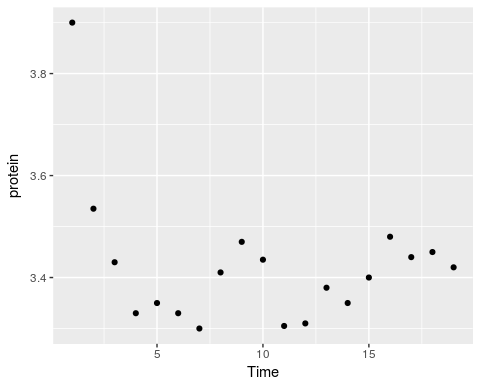


Some popular statistical summaries, have gained their own functions:

* mean\_cl\_boot: mean and bootstrapped confidence interval (default 95%).
* mean\_cl\_normal: mean and Gaussian (t-distribution based) confidence interval (default 95%).
* mean\_dsl: mean plus or minus standard deviation times some constant (default constant=2).
* median\_hilow: median and outer quantiles (default outer quantiles = 0.025 and 0.975).

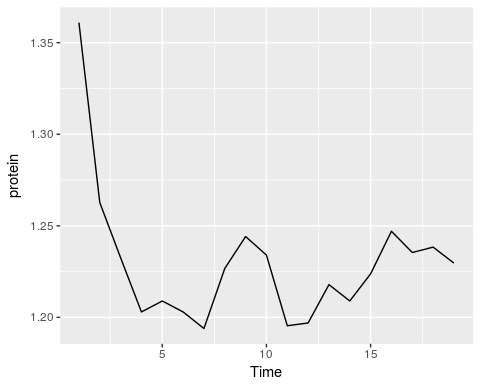
For less popular statistical summaries, we may specify the statistical function in stat\_summary. The median is a first example.

ggplot(Milk, aes(x=Time, y=protein)) +  
 stat\_summary(fun.y="median", geom="point")



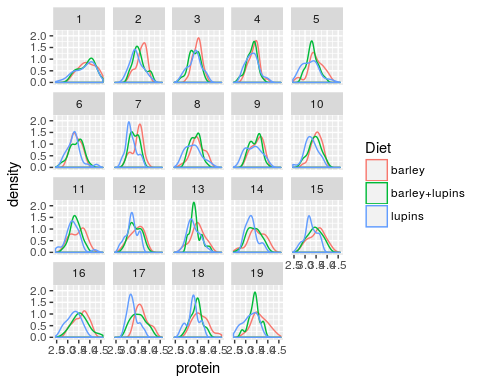
We can also define our own statistical summaries.

medianlog <- function(y) {median(log(y))}  
ggplot(Milk, aes(x=Time, y=protein)) +  
 stat\_summary(fun.y="medianlog", geom="line")



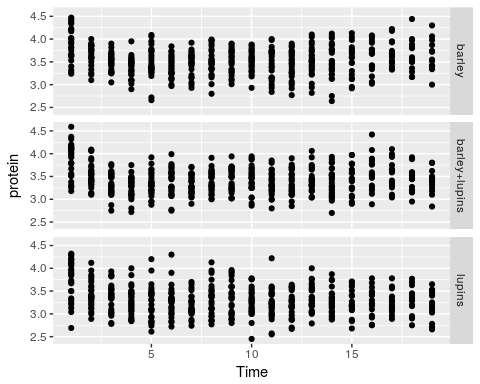
**Faceting** allows to split the plotting along some variable. face\_wrap tells R to compute the number of columns and rows of plots automatically.

ggplot(Milk, aes(x=protein, color=Diet)) +  
 geom\_density() +  
 facet\_wrap(~Time)



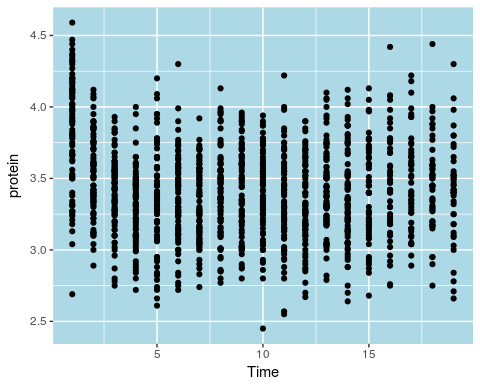
facet\_grid forces the plot to appear allow rows or columns, using the ~ syntax.

ggplot(Milk, aes(x=Time, y=protein)) +  
 geom\_point() +  
 facet\_grid(Diet~.) # `.~Diet` to split along columns and not rows.

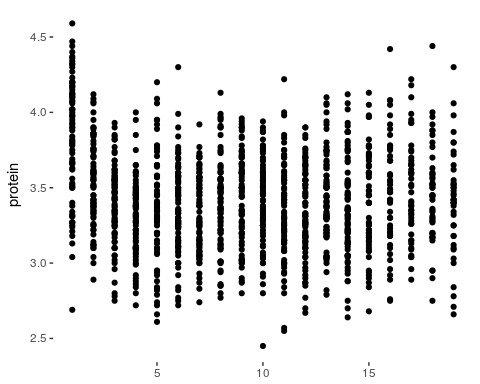


To control the looks of the plot, **ggplot2** uses **themes**.

ggplot(Milk, aes(x=Time, y=protein)) +  
 geom\_point() +  
 theme(panel.background=element\_rect(fill="lightblue"))



ggplot(Milk, aes(x=Time, y=protein)) +  
 geom\_point() +  
 theme(panel.background=element\_blank(),  
 axis.title.x=element\_blank())



Saving plots can be done using ggplot2::ggsave, or with pdf like the **graphics** plots:

pdf(file = 'myplot.pdf')  
print(tp) # You will need an explicit print command!  
dev.off()

Finally, what every user of **ggplot2** constantly uses, is the (excellent!) online documentation at <http://docs.ggplot2.org>.

## Interactive Graphics

As already mentioned, the recent and dramatic advancement in interactive visualization was made possible by the advances in web technologies, and the [D3.JS](https://d3js.org/) JavaScript library in particular. This is because it allows developers to rely on existing libraries designed for web browsing instead of re-implementing interactive visualizations. These libraries are more visually pleasing, and computationally efficient, than anything they could have developed themselves.

Some noteworthy interactive plotting systems are the following:

* **plotly**: The **plotly** package (Sievert et al. [2016](#ref-plotly)) uses the (brilliant!) visualization framework of the Plotly company to provide local, or web-publishable, interactive graphics.
* **dygraphs**: The [dygraphs](http://dygraphs.com/) JavaScript library is intended for interactive visualization of time series (xts class objects). The **dygraphs** R package is an interface allowing the plotting of R objects with this library. For more information see [here](https://rstudio.github.io/dygraphs/).
* **rCharts**: If you like the **lattice** plotting system, the **rCharts** package will allow you to produce interactive plots from R using the **lattice** syntax. For more information see [here](https://ramnathv.github.io/rCharts/).
* **clickme**: Very similar to **rCharts**.
* **googleVis**: TODO
* Highcharter: TODO
* Rbokeh: TODO
* **HTML Widgets**: The **htmlwidgets** package does not provide visualization, but rather, it facilitates the creation of new interactive visualizations. This is because it handles all the technical details that are required to use R output within JavaScript visualization libraries. It is available [here](http://www.htmlwidgets.org/), with a demo gallery [here](http://gallery.htmlwidgets.org/).

### Plotly

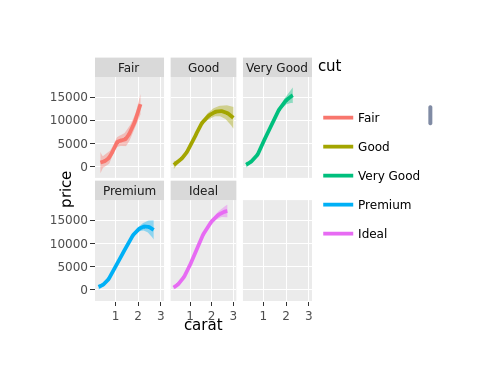
You can create nice interactive graphs using plotly::plot\_ly:

library(plotly)  
set.seed(100)  
d <- diamonds[sample(nrow(diamonds), 1000), ]

plot\_ly(data = d, x = ~carat, y = ~price, color = ~carat, size = ~carat, text = ~paste("Clarity: ", clarity))

More conveniently, any **ggplot2** graph can be made interactive using plotly::ggplotly:

p <- ggplot(data = d, aes(x = carat, y = price)) +  
 geom\_smooth(aes(colour = cut, fill = cut), method = 'loess') +   
 facet\_wrap(~ cut) # make ggplot  
ggplotly(p) # from ggplot to plotly



How about exporting **plotly** objects? Well, a **plotly** object is nothing more than a little web site: an HTML file. When showing a **plotly** figure, RStudio merely servers you as a web browser. You could, alternatively, export this HTML file to send your colleagues as an email attachment, or embed it in a web site. To export these, use the plotly::export or the htmlwidgets::saveWidget functions.

For more on **plotly** see <https://plot.ly/r/>.

### HTML Widgets

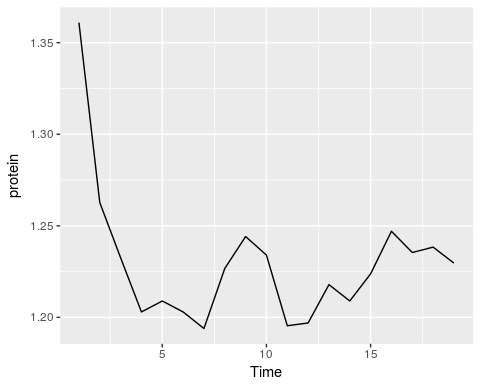
TODO

## Bibliographic Notes

For the **graphics** package, see R Core Team ([2016](#ref-Rlanguage)). For **ggplot2** see Wickham ([2009](#ref-ggplot2)). A [video](https://www.youtube.com/watch?v=9Objw9Tvhb4&feature=youtu.be) by one of my heroes, [Brian Caffo](http://www.bcaffo.com/), discussing **graphics** vs. **ggplot2**.

## Practice Yourself

1. Go to the Fancy Graphics Section 11.1.3. Try parsing the commands in your head.
2. Recall the medianlog example and replace the medianlog function with a [harmonic mean](https://en.wikipedia.org/wiki/Harmonic_mean).

* medianlog <- function(y) {median(log(y))}  
  ggplot(Milk, aes(x=Time, y=protein)) +  
   stat\_summary(fun.y="medianlog", geom="line")
* 

1. Write a function that creates a boxplot from scratch. See how I built a line graph in Section 11.1.3.
2. Export my plotly example using the RStudio interface and send it to yourself by email.

# Reports

If you have ever written a report, you are probably familiar with the process of preparing your figures in some software, say R, and then copy-pasting into your text editor, say MS Word. While very popular, this process is both tedious, and plain painful if your data has changed and you need to update the report. Wouldn't it be nice if you could produce figures and numbers from within the text of the report, and everything else would be automated? It turns out it is possible. There are actually several systems in R that allow this. We start with a brief review.

1. **Sweave**: *LaTeX* is a markup language that compiles to *Tex* programs that compile, in turn, to documents (typically PS or PDFs). If you never heard of it, it may be because you were born the the MS Windows+MS Word era. You should know, however, that *LaTeX* was there much earlier, when computers were mainframes with text-only graphic devices. You should also know that *LaTeX* is still very popular (in some communities) due to its very rich markup syntax, and beautiful output. *Sweave* (Leisch [2002](#ref-leisch2002sweave)) is a compiler for *LaTeX* that allows you do insert R commands in the *LaTeX* source file, and get the result as part of the outputted PDF. It's name suggests just that: it allows to weave S[[27]](#footnote-387) output into the document, thus, Sweave.
2. **knitr**: *Markdown* is a text editing syntax that, unlike *LaTeX*, is aimed to be human-readable, but also compilable by a machine. If you ever tried to read HTML or *LaTeX* source files, you may understand why human-readability is a desirable property. There are many *markdown* compilers. One of the most popular is Pandoc, written by the Berkeley philosopher(!) Jon MacFarlane. The availability of Pandoc gave [Yihui Xie](https://yihui.name/), a name to remember, the idea that it is time for Sweave to evolve. Yihui thus wrote **knitr** (Xie [2015](#ref-xie2015dynamic)), which allows to write human readable text in *Rmarkdown*, a superset of *markdown*, compile it with R and the compile it with Pandoc. Because Pandoc can compile to PDF, but also to HTML, and DOCX, among others, this means that you can write in Rmarkdown, and get output in almost all text formats out there.
3. **bookdown**: **Bookdown** (Xie [2016](#ref-xie2016bookdown)) is an evolution of **knitr**, also written by Yihui Xie, now working for RStudio. The text you are now reading was actually written in **bookdown**. It deals with the particular needs of writing large documents, and cross referencing in particular (which is very challenging if you want the text to be human readable).
4. **Shiny**: Shiny is essentially a framework for quick web-development. It includes (i) an abstraction layer that specifies the layout of a web-site which is our report, (ii) the command to start a web server to deliver the site. For more on Shiny see Chang et al. ([2017](#ref-shiny)).

## knitr

### Installation

To run **knitr** you will need to install the package.

install.packages('knitr')

It is also recommended that you use it within RStudio (version>0.96), where you can easily create a new .Rmd file.

### Pandoc Markdown

Because **knitr** builds upon *Pandoc markdown*, here is a simple example of markdown text, to be used in a .Rmd file, which can be created using the *File-> New File -> R Markdown* menu of RStudio.

Underscores or asterisks for \_italics1\_ and \*italics2\* return *italics1* and *italics2*. Double underscores or asterisks for \_\_bold1\_\_ and \*\*bold2\*\* return **bold1** and **bold2**. Subscripts are enclosed in tildes, like~this~ (likethis), and superscripts are enclosed in carets like^this^ (likethis).

For links use [text](link), like [my site](www.john-ros.com). An image is the same as a link, starting with an exclamation, like this ![image caption](image path).

An itemized list simply starts with hyphens preceeded by a blank line (don't forget that!):

- bullet  
- bullet  
 - second level bullet  
 - second level bullet

Compiles into:

* bullet
* bullet
  + second level bullet
  + second level bullet

An enumerated list starts with an arbitrary number:

1. number  
1. number  
 1. second level number  
 1. second level number

Compiles into:

1. number
2. number
   1. second level number
   2. second level number

For more on markdown see <https://bookdown.org/yihui/bookdown/markdown-syntax.html>.

### Rmarkdown

*Rmarkdown*, is an extension of *markdown* due to RStudio, that allows to incorporate R expressions in the text, that will be evaluated at the time of compilation, and the output automatically inserted in the outputted text. The output can be a .PDF, .DOCX, .HTML or others, thanks to the power of *Pandoc*.

The start of a code chunk is indicated by three backticks and the end of a code chunk is indicated by three backticks. Here is an example.

```{r eval=FALSE}  
rnorm(10)  
```

This chunk will compile to the following output (after setting eval=FALSE to eval=TRUE):

rnorm(10)

## [1] -1.4462875 0.3158558 -0.3427475 -1.9313531 0.2428210 -0.3627679  
## [7] 2.4327289 0.5920912 -0.5762008 0.4066282

Things to note:

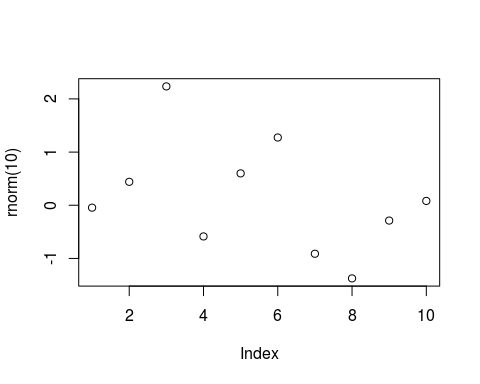
* The evaluated expression is added in a chunk of highlighted text, before the R output.
* The output is prefixed with ##.
* The eval= argument is not required, since it is set to eval=TRUE by default. It does demonstrate how to set the options of the code chunk.

In the same way, we may add a plot:

```{r eval=FALSE}  
plot(rnorm(10))  
```

which compiles into

plot(rnorm(10))



TODO: more code chunk options.

You can also call r expressions inline. This is done with a single tick and the r argument. For instance:

`r rnorm(1)` is a random Gaussian

will output

0.3378953 is a random Gaussian.

### Compiling

Once you have your .Rmd file written in RMarkdown, **knitr** will take care of the compilation for you. You can call the knitr::knitr function directly from some .R file, or more conveniently, use the RStudio (0.96) Knit button above the text editing window. The location of the output file will be presented in the console.

## bookdown

As previously stated, **bookdown** is an extension of **knitr** intended for documents more complicated than simple reports-- such as books. Just like **knitr**, the writing is done in **RMarkdown**. Being an extension of **knitr**, **bookdown** does allow some markdowns that are not supported by other compilers. In particular, it has a more powerful cross referencing system.

## Shiny

**Shiny** (Chang et al. [2017](#ref-shiny)) is different than the previous systems, because it sets up an interactive web-site, and not a static file. The power of Shiny is that the layout of the web-site, and the settings of the web-server, is made with several simple R commands, with no need for web-programming. Once you have your app up and running, you can setup your own Shiny server on the web, or publish it via [Shinyapps.io](https://www.shinyapps.io/). The freemium versions of the service can deal with a small amount of traffic. If you expect a lot of traffic, you will probably need the paid versions.

### Installation

To setup your first Shiny app, you will need the **shiny** package. You will probably want RStudio, which facilitates the process.

install.packages('shiny')

Once installed, you can run an example app to get the feel of it.

library(shiny)  
runExample("01\_hello")

Remember to press the **Stop** button in RStudio to stop the web-server, and get back to RStudio.

### The Basics of Shiny

Every Shiny app has two main building blocks.

1. A user interface, specified via the ui.R file in the app's directory.
2. A server side, specified via the server.R file, in the app's directory.

You can run the app via the **RunApp** button in the RStudio interface, of by calling the app's directory with the shinyApp or runApp functions-- the former designed for single-app projects, and the latter, for multiple app projects.

shiny::runApp("my\_app")

The site's layout, is specified via *layout functions* in the iu.R file. For instance, the function sidebarLayout, as the name suggest, will create a sidebar. More layouts are detailed in the [layout guide](http://shiny.rstudio.com/articles/layout-guide.html).

The active elements in the UI, that control your report, are known as *widgets*. Each widget will have a unique inputId so that it's values can be sent from the UI to the server. More about widgets, in the [widget gallery](http://shiny.rstudio.com/gallery/widget-gallery.html).

The inputId on the UI are mapped to input arguments on the server side. The value of the mytext inputId can be queried by the server using input$mytext. These are called *reactive values*. The way the server "listens" to the UI, is governed by a set of functions that must wrap the input object. These are the observe, reactive, and reactive\* class of functions.

With observe the server will get triggered when any of the reactive values change. With observeEvent the server will only be triggered by specified reactive values. Using observe is easier, and observeEvent is more prudent programming.

A reactive function is a function that gets triggered when a reactive element changes. It is defined on the server side, and reside within an observe function.

We now analyze the 1\_Hello app using these ideas. Here is the io.R file.

library(shiny)  
  
shinyUI(fluidPage(  
  
 titlePanel("Hello Shiny!"),  
  
 sidebarLayout(  
 sidebarPanel(  
 sliderInput(inputId = "bins",  
 label = "Number of bins:",   
 min = 1,  
 max = 50,  
 value = 30)  
 ),  
  
 mainPanel(  
 plotOutput(outputId = "distPlot")  
 )  
 )  
))

Here is the server.R file:

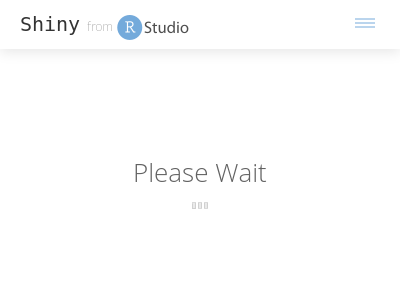
library(shiny)  
  
shinyServer(function(input, output) {  
  
 output$distPlot <- renderPlot({  
 x <- faithful[, 2] # Old Faithful Geyser data  
 bins <- seq(min(x), max(x), length.out = input$bins + 1)  
  
 hist(x, breaks = bins, col = 'darkgray', border = 'white')  
 })  
})

Things to note:

* ShinyUI is a (deprecated) wrapper for the UI.
* fluidPage ensures that the proportions of the elements adapt to the window side, thus, are fluid.
* The building blocks of the layout are a title, and the body. The title is governed by titlePanel, and the body is governed by sidebarLayout. The sidebarLayout includes the sidebarPanel to control the sidebar, and the mainPanel for the main panel.
* sliderInput calls a widget with a slider. Its inputId is bins, which is later used by the server within the renderPlot reactive function.
* plotOutput specifies that the content of the mainPanel is a plot (textOutput for text). This expectation is satisfied on the server side with the renderPlot function (renderText).
* shinyServer is a (deprecated) wrapper function for the server.
* The server runs a function with an input and an output. The elements of input are the inputIds from the UI. The elements of the output will be called by the UI using their outputId.

This is the output.

knitr::include\_url('http://shiny.rstudio.com/gallery/example-01-hello.html')

[](http://shiny.rstudio.com/gallery/example-01-hello.html)

Here is another example, taken from the RStudio [Shiny examples](https://github.com/rstudio/shiny-examples/tree/master/006-tabsets).

ui.R:

library(shiny)  
  
fluidPage(  
   
 titlePanel("Tabsets"),  
   
 sidebarLayout(  
 sidebarPanel(  
 radioButtons(inputId = "dist",   
 label = "Distribution type:",  
 c("Normal" = "norm",  
 "Uniform" = "unif",  
 "Log-normal" = "lnorm",  
 "Exponential" = "exp")),  
 br(),  
   
 sliderInput(inputId = "n",   
 label = "Number of observations:",   
 value = 500,  
 min = 1,   
 max = 1000)  
 ),  
   
 mainPanel(  
 tabsetPanel(type = "tabs",   
 tabPanel(title = "Plot", plotOutput(outputId = "plot")),   
 tabPanel(title = "Summary", verbatimTextOutput(outputId = "summary")),   
 tabPanel(title = "Table", tableOutput(outputId = "table"))  
 )  
 )  
 )  
)

server.R:

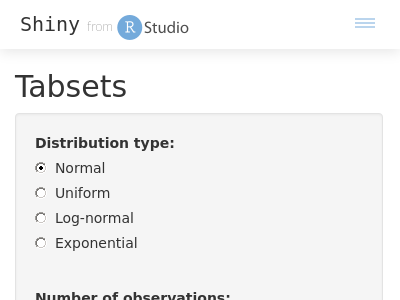
library(shiny)  
  
# Define server logic for random distribution application  
function(input, output) {  
   
 data <- reactive({  
 dist <- switch(input$dist,  
 norm = rnorm,  
 unif = runif,  
 lnorm = rlnorm,  
 exp = rexp,  
 rnorm)  
   
 dist(input$n)  
 })  
   
 output$plot <- renderPlot({  
 dist <- input$dist  
 n <- input$n  
   
 hist(data(), main=paste('r', dist, '(', n, ')', sep=''))  
 })  
   
 output$summary <- renderPrint({  
 summary(data())  
 })  
   
 output$table <- renderTable({  
 data.frame(x=data())  
 })  
   
}

Things to note:

* We reused the sidebarLayout.
* As the name suggests, radioButtons is a widget that produces radio buttons, above the sliderInput widget. Note the different inputIds.
* Different widgets are separated in sidebarPanel by commas.
* br() produces extra vertical spacing.
* tabsetPanel produces tabs in the main output panel. tabPanel governs the content of each panel. Notice the use of various output functions (plotOutput,verbatimTextOutput, tableOutput) with corresponding outputIds.
* In server.R we see the usual function(input,output).
* The reactive function tells the server the trigger the function whenever input changes.
* The output object is constructed outside the reactive function. See how the elements of output correspond to the outputIds in the UI.

This is the output:

knitr::include\_url('https://shiny.rstudio.com/gallery/tabsets.html')

[](https://shiny.rstudio.com/gallery/tabsets.html)

### Beyond the Basics

Now that we have seen the basics, we may consider extensions to the basic report.

#### Widgets

* actionButton Action Button.
* checkboxGroupInput A group of check boxes.
* checkboxInput A single check box.
* dateInput A calendar to aid date selection.
* dateRangeInput A pair of calendars for selecting a date range.
* fileInput A file upload control wizard.
* helpText Help text that can be added to an input form.
* numericInput A field to enter numbers.
* radioButtons A set of radio buttons.
* selectInput A box with choices to select from.
* sliderInput A slider bar.
* submitButton A submit button.
* textInput A field to enter text.

See examples [here](https://shiny.rstudio.com/gallery/widget-gallery.html).

knitr::include\_url('https://shiny.rstudio.com/gallery/widget-gallery.html')

[](https://shiny.rstudio.com/gallery/widget-gallery.html)

#### Output Elements

The ui.R output types.

* htmlOutput raw HTML.
* imageOutput image.
* plotOutput plot.
* tableOutput table.
* textOutput text.
* uiOutput raw HTML.
* verbatimTextOutput text.

The corresponding server.R renderers.

* renderImage images (saved as a link to a source file)
* renderPlot plots
* renderPrint any printed output
* renderTable data frame, matrix, other table like structures
* renderText character strings
* renderUI a Shiny tag object or HTML

Your Shiny app can use any R object. The things to remember:

* The working directory of the app is the location of server.R.
* The code before shinyServer is run only once.
* The code inside `shinyServer is run whenever a reactive is triggered, and may thus slow things.

To keep learning, see the RStudio's [tutorial](http://shiny.rstudio.com/tutorial/), and the Biblipgraphic notes herein.

## Flexdashboard

<http://rmarkdown.rstudio.com/flexdashboard/>

TODO: write section

## Bibliographic Notes

For RMarkdown see [here](http://rmarkdown.rstudio.com/). For everything on **knitr** see [Yihui's blog](https://yihui.name/knitr/), or the book Xie ([2015](#ref-xie2015dynamic)). For a **bookdown** manual, see Xie ([2016](#ref-xie2016bookdown)). For a Shiny manual, see Chang et al. ([2017](#ref-shiny)), the [RStudio tutorial](http://shiny.rstudio.com/tutorial/), or [Zev Ross's](http://zevross.com/blog/2016/04/19/r-powered-web-applications-with-shiny-a-tutorial-and-cheat-sheet-with-40-example-apps/) excellent guide. Video tutorials are available [here](https://www.rstudio.com/resources/webinars/shiny-developer-conference/).

## Practice Yourself

1. Generate a report using **knitr** with your name as title, and a scatter plot of two random variables in the body. Save it as PDF, DOCX, and HTML.
2. Recall that this book is written in **bookdown**, which is a superset of **knitr**. Go to the source .Rmd file of the first chapter, and parse it in your head: (<https://raw.githubusercontent.com/johnros/Rcourse/master/02-r-basics.Rmd>)

# The Hadleyverse

The *Hadleyverse*, short for "Hadley Wickham's universe", is a set of packages that make it easier to handle data. If you are developing packages, you should be careful since using these packages may create many dependencies and compatibility issues. If you are analyzing data, and the portability of your functions to other users, machines, and operating systems is not of a concern, you will LOVE these packages. The term Hadleyverse refers to **all** of Hadley's packages, but here, we mention only a useful subset, which can be collectively installed via the **tidyverse** package:

* **ggplot2** for data visualization. See the Plotting Chapter 11.
* **dplyr** for data manipulation.
* **tidyr** for data tidying.
* **readr** for data import.
* **stringr** for character strings.
* **anytime** for time data.

## readr

The **readr** package (Wickham, Hester, and Francois [2016](#ref-readr)) replaces base functions for importing and exporting data such as read.table. It is faster, with a cleaner syntax.

We will not go into the details and refer the reader to the official documentation [here](http://readr.tidyverse.org/articles/readr.html) and the [R for data sciecne](http://r4ds.had.co.nz/data-import.html) book.

## dplyr

When you think of data frame operations, think **dplyr** (Wickham and Francois [2016](#ref-dplyr)). Notable utilities in the package include:

* select() Select columns from a data frame.
* filter() Filter rows according to some condition(s).
* arrange() Sort / Re-order rows in a data frame.
* mutate() Create new columns or transform existing ones.
* group\_by() Group a data frame by some factor(s) usually in conjunction to summary.
* summarize() Summarize some values from the data frame or across groups.
* inner\_join(x,y,by="col")return all rows from ‘x’ where there are matching values in ‘x’, and all columns from ‘x’ and ‘y’. If there are multiple matches between ‘x’ and ‘y’, all combination of the matches are returned.
* left\_join(x,y,by="col") return all rows from ‘x’, and all columns from ‘x’ and ‘y’. Rows in ‘x’ with no match in ‘y’ will have ‘NA’ values in the new columns. If there are multiple matches between ‘x’ and ‘y’, all combinations of the matches are returned.
* right\_join(x,y,by="col") return all rows from ‘y’, and all columns from ‘x’ and y. Rows in ‘y’ with no match in ‘x’ will have ‘NA’ values in the new columns. If there are multiple matches between ‘x’ and ‘y’, all combinations of the matches are returned.
* anti\_join(x,y,by="col") return all rows from ‘x’ where there are not matching values in ‘y’, keeping just columns from ‘x’.

The following example involve data.frame objects, but **dplyr** can handle other classes. In particular data.tables from the **data.table** package (Dowle and Srinivasan [2017](#ref-datatable)), which is designed for very large data sets.

**dplyr** can work with data stored in a database. In which case, it will convert your command to the appropriate SQL syntax, and issue it to the database. This has the advantage that (a) you do not need to know the specific SQL implementation of your database, and (b), you can enjoy the optimized algorithms provided by the database supplier. For more on this, see the [databses vignette](https://cran.r-project.org/web/packages/dplyr/vignettes/databases.html).

The following examples are taken from [Kevin Markham](https://github.com/justmarkham/dplyr-tutorial/blob/master/dplyr-tutorial.Rmd). The nycflights13::flights has delay data for US flights.

library(nycflights13)  
flights

## # A tibble: 336,776 x 19  
## year month day dep\_time sched\_dep\_time dep\_delay arr\_time  
## <int> <int> <int> <int> <int> <dbl> <int>  
## 1 2013 1 1 517 515 2 830  
## 2 2013 1 1 533 529 4 850  
## 3 2013 1 1 542 540 2 923  
## 4 2013 1 1 544 545 -1 1004  
## 5 2013 1 1 554 600 -6 812  
## 6 2013 1 1 554 558 -4 740  
## 7 2013 1 1 555 600 -5 913  
## 8 2013 1 1 557 600 -3 709  
## 9 2013 1 1 557 600 -3 838  
## 10 2013 1 1 558 600 -2 753  
## # ... with 336,766 more rows, and 12 more variables: sched\_arr\_time <int>,  
## # arr\_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,  
## # origin <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,  
## # minute <dbl>, time\_hour <dttm>

The data is of class tbl\_df which is an extension of the data.frame class, designed for large data sets. Notice that the printing of flights is short, even without calling the head function. This is a feature of the tbl\_df class ( print(data.frame) would try to load all the data, thus take a long time).

class(flights) # a tbl\_df is an extension of the data.frame class

## [1] "tbl\_df" "tbl" "data.frame"

Let's filter the observations from the first day of the first month. Notice how much better (i.e. readable) is the **dplyr** syntax, with piping, compared to the basic syntax.

flights[flights$month == 1 & flights$day == 1, ] # old style  
  
library(dplyr)   
filter(flights, month == 1, day == 1) #dplyr style  
flights %>% filter(month == 1, day == 1) # dplyr with piping.

More filtering.

filter(flights, month == 1 | month == 2) # First OR second month.  
slice(flights, 1:10) # selects first ten rows.  
  
arrange(flights, year, month, day) # sort  
arrange(flights, desc(arr\_delay)) # sort descending  
  
select(flights, year, month, day) # select columns year, month, and day  
select(flights, year:day) # select column range  
select(flights, -(year:day)) # drop columns  
rename(flights, tail\_num = tailnum) # rename column  
  
# add a new computed colume  
mutate(flights,  
 gain = arr\_delay - dep\_delay,  
 speed = distance / air\_time \* 60)   
  
# you can refer to columns you just created! (gain)  
mutate(flights,  
 gain = arr\_delay - dep\_delay,  
 gain\_per\_hour = gain / (air\_time / 60)  
)  
  
# keep only new variables, not all data frame.  
transmute(flights,  
 gain = arr\_delay - dep\_delay,  
 gain\_per\_hour = gain / (air\_time / 60)  
)  
  
# simple statistics  
summarise(flights,  
 delay = mean(dep\_delay, na.rm = TRUE)  
 )  
# random subsample   
sample\_n(flights, 10)   
sample\_frac(flights, 0.01)

We now perform operations on subgroups. we group observations along the plane's tail number (tailnum), and compute the count, average distance traveled, and average delay. We group with group\_by, and compute subgroup statistics with summarise.

by\_tailnum <- group\_by(flights, tailnum)  
  
delay <- summarise(by\_tailnum,  
 count = n(),  
 avg.dist = mean(distance, na.rm = TRUE),  
 avg.delay = mean(arr\_delay, na.rm = TRUE))  
  
delay

## # A tibble: 4,044 x 4  
## tailnum count avg.dist avg.delay  
## <chr> <int> <dbl> <dbl>  
## 1 D942DN 4 854.5000 31.5000000  
## 2 N0EGMQ 371 676.1887 9.9829545  
## 3 N10156 153 757.9477 12.7172414  
## 4 N102UW 48 535.8750 2.9375000  
## 5 N103US 46 535.1957 -6.9347826  
## 6 N104UW 47 535.2553 1.8043478  
## 7 N10575 289 519.7024 20.6914498  
## 8 N105UW 45 524.8444 -0.2666667  
## 9 N107US 41 528.7073 -5.7317073  
## 10 N108UW 60 534.5000 -1.2500000  
## # ... with 4,034 more rows

We can group along several variables, with a hierarchy. We then collapse the hierarchy one by one.

daily <- group\_by(flights, year, month, day)  
per\_day <- summarise(daily, flights = n())  
per\_month <- summarise(per\_day, flights = sum(flights))  
per\_year <- summarise(per\_month, flights = sum(flights))

Things to note:

* Every call to summarise collapses one level in the hierarchy of grouping. The output of group\_by recalls the hierarchy of aggregation, and collapses along this hierarchy.

We can use **dplyr** for two table operations, i.e., *joins*. For this, we join the flight data, with the airplane data in airplanes.

library(dplyr)   
airlines

## # A tibble: 16 x 2  
## carrier name  
## <chr> <chr>  
## 1 9E Endeavor Air Inc.  
## 2 AA American Airlines Inc.  
## 3 AS Alaska Airlines Inc.  
## 4 B6 JetBlue Airways  
## 5 DL Delta Air Lines Inc.  
## 6 EV ExpressJet Airlines Inc.  
## 7 F9 Frontier Airlines Inc.  
## 8 FL AirTran Airways Corporation  
## 9 HA Hawaiian Airlines Inc.  
## 10 MQ Envoy Air  
## 11 OO SkyWest Airlines Inc.  
## 12 UA United Air Lines Inc.  
## 13 US US Airways Inc.  
## 14 VX Virgin America  
## 15 WN Southwest Airlines Co.  
## 16 YV Mesa Airlines Inc.

# select the subset of interesting flight data.   
flights2 <- flights %>% select(year:day, hour, origin, dest, tailnum, carrier)   
  
# join on left table with automatic matching.  
flights2 %>% left\_join(airlines)

## Joining, by = "carrier"

## # A tibble: 336,776 x 9  
## year month day hour origin dest tailnum carrier  
## <int> <int> <int> <dbl> <chr> <chr> <chr> <chr>  
## 1 2013 1 1 5 EWR IAH N14228 UA  
## 2 2013 1 1 5 LGA IAH N24211 UA  
## 3 2013 1 1 5 JFK MIA N619AA AA  
## 4 2013 1 1 5 JFK BQN N804JB B6  
## 5 2013 1 1 6 LGA ATL N668DN DL  
## 6 2013 1 1 5 EWR ORD N39463 UA  
## 7 2013 1 1 6 EWR FLL N516JB B6  
## 8 2013 1 1 6 LGA IAD N829AS EV  
## 9 2013 1 1 6 JFK MCO N593JB B6  
## 10 2013 1 1 6 LGA ORD N3ALAA AA  
## # ... with 336,766 more rows, and 1 more variables: name <chr>

flights2 %>% left\_join(weather)

## Joining, by = c("year", "month", "day", "hour", "origin")

## # A tibble: 336,776 x 18  
## year month day hour origin dest tailnum carrier temp dewp humid  
## <dbl> <dbl> <int> <dbl> <chr> <chr> <chr> <chr> <dbl> <dbl> <dbl>  
## 1 2013 1 1 5 EWR IAH N14228 UA NA NA NA  
## 2 2013 1 1 5 LGA IAH N24211 UA NA NA NA  
## 3 2013 1 1 5 JFK MIA N619AA AA NA NA NA  
## 4 2013 1 1 5 JFK BQN N804JB B6 NA NA NA  
## 5 2013 1 1 6 LGA ATL N668DN DL 39.92 26.06 57.33  
## 6 2013 1 1 5 EWR ORD N39463 UA NA NA NA  
## 7 2013 1 1 6 EWR FLL N516JB B6 39.02 26.06 59.37  
## 8 2013 1 1 6 LGA IAD N829AS EV 39.92 26.06 57.33  
## 9 2013 1 1 6 JFK MCO N593JB B6 39.02 26.06 59.37  
## 10 2013 1 1 6 LGA ORD N3ALAA AA 39.92 26.06 57.33  
## # ... with 336,766 more rows, and 7 more variables: wind\_dir <dbl>,  
## # wind\_speed <dbl>, wind\_gust <dbl>, precip <dbl>, pressure <dbl>,  
## # visib <dbl>, time\_hour <dttm>

# join with named matching  
flights2 %>% left\_join(planes, by = "tailnum")

## # A tibble: 336,776 x 16  
## year.x month day hour origin dest tailnum carrier year.y  
## <int> <int> <int> <dbl> <chr> <chr> <chr> <chr> <int>  
## 1 2013 1 1 5 EWR IAH N14228 UA 1999  
## 2 2013 1 1 5 LGA IAH N24211 UA 1998  
## 3 2013 1 1 5 JFK MIA N619AA AA 1990  
## 4 2013 1 1 5 JFK BQN N804JB B6 2012  
## 5 2013 1 1 6 LGA ATL N668DN DL 1991  
## 6 2013 1 1 5 EWR ORD N39463 UA 2012  
## 7 2013 1 1 6 EWR FLL N516JB B6 2000  
## 8 2013 1 1 6 LGA IAD N829AS EV 1998  
## 9 2013 1 1 6 JFK MCO N593JB B6 2004  
## 10 2013 1 1 6 LGA ORD N3ALAA AA NA  
## # ... with 336,766 more rows, and 7 more variables: type <chr>,  
## # manufacturer <chr>, model <chr>, engines <int>, seats <int>,  
## # speed <int>, engine <chr>

# join with explicit column matching  
flights2 %>% left\_join(airports, by= c("dest" = "faa"))

## # A tibble: 336,776 x 15  
## year month day hour origin dest tailnum carrier  
## <int> <int> <int> <dbl> <chr> <chr> <chr> <chr>  
## 1 2013 1 1 5 EWR IAH N14228 UA  
## 2 2013 1 1 5 LGA IAH N24211 UA  
## 3 2013 1 1 5 JFK MIA N619AA AA  
## 4 2013 1 1 5 JFK BQN N804JB B6  
## 5 2013 1 1 6 LGA ATL N668DN DL  
## 6 2013 1 1 5 EWR ORD N39463 UA  
## 7 2013 1 1 6 EWR FLL N516JB B6  
## 8 2013 1 1 6 LGA IAD N829AS EV  
## 9 2013 1 1 6 JFK MCO N593JB B6  
## 10 2013 1 1 6 LGA ORD N3ALAA AA  
## # ... with 336,766 more rows, and 7 more variables: name <chr>, lat <dbl>,  
## # lon <dbl>, alt <int>, tz <dbl>, dst <chr>, tzone <chr>

Types of join with SQL equivalent.

# Create simple data  
(df1 <- data\_frame(x = c(1, 2), y = 2:1))

## # A tibble: 2 x 2  
## x y  
## <dbl> <int>  
## 1 1 2  
## 2 2 1

(df2 <- data\_frame(x = c(1, 3), a = 10, b = "a"))

## # A tibble: 2 x 3  
## x a b  
## <dbl> <dbl> <chr>  
## 1 1 10 a  
## 2 3 10 a

# Return only matched rows  
df1 %>% inner\_join(df2) # SELECT \* FROM x JOIN y ON x.a = y.a

## Joining, by = "x"

## # A tibble: 1 x 4  
## x y a b  
## <dbl> <int> <dbl> <chr>  
## 1 1 2 10 a

# Return all rows in df1.  
df1 %>% left\_join(df2) # SELECT \* FROM x LEFT JOIN y ON x.a = y.a

## Joining, by = "x"

## # A tibble: 2 x 4  
## x y a b  
## <dbl> <int> <dbl> <chr>  
## 1 1 2 10 a  
## 2 2 1 NA <NA>

# Return all rows in df2.  
df1 %>% right\_join(df2) # SELECT \* FROM x RIGHT JOIN y ON x.a = y.a

## Joining, by = "x"

## # A tibble: 2 x 4  
## x y a b  
## <dbl> <int> <dbl> <chr>  
## 1 1 2 10 a  
## 2 3 NA 10 a

# Return all rows.   
df1 %>% full\_join(df2) # SELECT \* FROM x FULL JOIN y ON x.a = y.a

## Joining, by = "x"

## # A tibble: 3 x 4  
## x y a b  
## <dbl> <int> <dbl> <chr>  
## 1 1 2 10 a  
## 2 2 1 NA <NA>  
## 3 3 NA 10 a

# Like left\_join, but returning only columns in df1  
df1 %>% semi\_join(df2, by = "x") # SELECT \* FROM x WHERE EXISTS (SELECT 1 FROM y WHERE x.a = y.a)

## # A tibble: 1 x 2  
## x y  
## <dbl> <int>  
## 1 1 2

## tidyr

## reshape2

## stringr

## anytime

## Biblipgraphic Notes

## Practice Yourself

# Sparse Representations

Analyzing "bigdata" in R is a challenge because the workspace is memory resident, i.e., all your objects are stored in RAM. As a rule of thumb, fitting models requires about 5 times the size of the data. This means that if you have 1 GB of data, you might need about 5 GB to fit a linear models. We will discuss how to compute *out of RAM* in the Memory Efficiency Chapter 15. In this chapter, we discuss efficient representations of your data, so that it takes less memory. The fundamental idea, is that if your data is *sparse*, i.e., there are many zero entries in your data, then a naive data.frame or matrix will consume memory for all these zeroes. If, however, you have many recurring zeroes, it is more efficient to save only the non-zero entries.

When we say *data*, we actually mean the model.matrix. The model.matrix is a matrix that R grows, converting all your factors to numeric variables that can be computed with. *Dummy coding* of your factors, for instance, is something that is done in your model.matrix. If you have a factor with many levels, you can imagine that after dummy coding it, many zeroes will be present.

The **Matrix** package replaces the matrix class, with several sparse representations of matrix objects.

When using sparse representation, and the **Matrix** package, you will need an implementation of your favorite model fitting algorithm (e.g. lm) that is adapted to these sparse representations; otherwise, R will cast the sparse matrix into a regular (non-sparse) matrix, and you will have saved nothing in RAM.

Remark. If you are familiar with MATLAB you should know that one of the great capabilities of MATLAB, is the excellent treatment of sparse matrices with the sparse function.

Before we go into details, here is a simple example. We will create a factor of letters with the letters function. Clearly, this factor can take only values. This means that of the model.matrix will be zeroes after dummy coding. We will compare the memory footprint of the naive model.matrix with the sparse representation of the same matrix.

library(magrittr)  
reps <- 1e6 # number of samples  
y<-rnorm(reps)  
x<- letters %>%   
 sample(reps, replace=TRUE) %>%   
 factor

The object x is a factor of letters:

head(x)

## [1] n x z f a i  
## Levels: a b c d e f g h i j k l m n o p q r s t u v w x y z

We dummy code x with the model.matrix function.

X.1 <- model.matrix(~x-1)  
head(X.1)

## xa xb xc xd xe xf xg xh xi xj xk xl xm xn xo xp xq xr xs xt xu xv xw xx  
## 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0  
## 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
## 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## 4 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## 5 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## 6 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## xy xz  
## 1 0 0  
## 2 0 0  
## 3 0 1  
## 4 0 0  
## 5 0 0  
## 6 0 0

We call **MatrixModels** for an implementation of model.matrix that supports sparse representations.

suppressPackageStartupMessages(library(MatrixModels))  
X.2<- as(x,"sparseMatrix") %>% t # Makes sparse dummy model.matrix  
head(X.2)

## 6 x 26 sparse Matrix of class "dgCMatrix"

## [[ suppressing 26 column names 'a', 'b', 'c' ... ]]

##   
## [1,] . . . . . . . . . . . . . 1 . . . . . . . . . . . .  
## [2,] . . . . . . . . . . . . . . . . . . . . . . . 1 . .  
## [3,] . . . . . . . . . . . . . . . . . . . . . . . . . 1  
## [4,] . . . . . 1 . . . . . . . . . . . . . . . . . . . .  
## [5,] 1 . . . . . . . . . . . . . . . . . . . . . . . . .  
## [6,] . . . . . . . . 1 . . . . . . . . . . . . . . . . .

Notice that the matrices have the same dimensions:

dim(X.1)

## [1] 1000000 26

dim(X.2)

## [1] 1000000 26

The memory footprint of the matrices, given by the pryr::object\_size function, are very very different.

pryr::object\_size(X.1)

## 264 MB

pryr::object\_size(X.2)

## 12 MB

Things to note:

* The sparse representation takes a whole lot less memory than the non sparse.
* The as(,"sparseMatrix") function grows the dummy variable representation of the factor x.
* The **pryr** package provides many facilities for inspecting the memory footprint of your objects and code.

With a sparse representation, we not only saved on RAM, but also on the computing time of fitting a model. Here is the timing of a non sparse representation:

system.time(lm.1 <- lm(y ~ X.1))

## user system elapsed   
## 3.252 0.128 3.380

Well actually, lm is a wrapper for the lm.fit function. If we override all the overhead of lm, and call lm.fit directly, we gain some time:

system.time(lm.1 <- lm.fit(y=y, x=X.1))

## user system elapsed   
## 1.092 0.024 1.118

We now do the same with the sparse representation:

system.time(lm.2 <- MatrixModels:::lm.fit.sparse(X.2,y))

## user system elapsed   
## 0.204 0.020 0.222

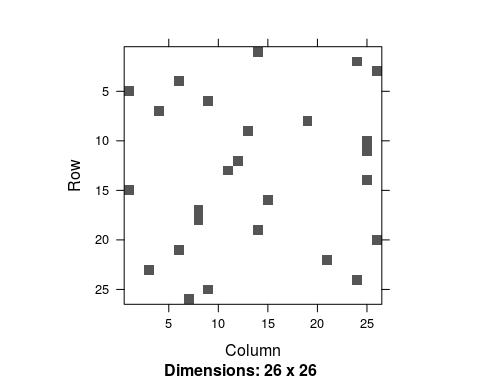
It is only left to verify that the returned coefficients are the same:

all.equal(lm.2, unname(lm.1$coefficients), tolerance = 1e-12)

## [1] TRUE

You can also visualize the non zero entries, i.e., the sparsity structure.

image(X.2[1:26,1:26])



## Sparse Matrix Representations

We first distinguish between the two main goals of the efficient representation: (i) efficient writing, i.e., modification; (ii) efficient reading, i.e., access. For our purposes, we will typically want efficient reading, since the model.matrix will not change while a model is being fitted.

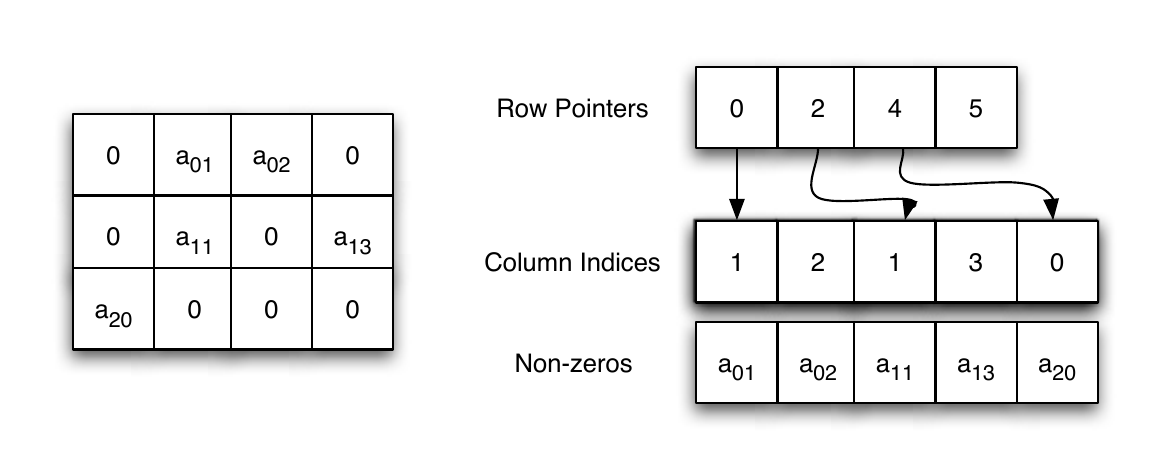
Representations designed for writing include the *dictionary of keys*, *list of lists*, and a *coordinate list*. Representations designed for efficient reading include the *compressed sparse row* and *compressed sparse column*.

### Coordinate List Representation

A *coordinate list representation*, also known as *COO*, or *triplet represantation* is simply a list of the non zero entries. Each element in the list is a triplet of the column, row, and value, of each non-zero entry in the matrix.

### Compressed Column Oriented Representation

A *compressed column oriented representation*, also known as *compressed sparse column*, or *CSC*, where the **column** index is similar to COO, but instead of saving the row indexes, we save the locations in the column index vectors where the row index has to increase. The following figure may clarify this simple idea.



The CSC data structure. From Shah and Gilbert ([2004](#ref-shah2004sparse)). Remember that MATLAB is written in C, where the indexing starts at , and not .

The nature of statistical applications is such, that CSC representation is typically the most economical, justifying its popularity.

### Compressed Row Oriented Representation

A *compressed row oriented representation*, also known as *compressed sparse row*, or *CSR*, is very similar to CSC, after switching the role of rows and columns. CSR is much less popular than CSC.

### Sparse Algorithms

We will go into the details of some algorithms in the Numerical Linear Algebra Chapter 17. For our current purposes two things need to be emphasized:

1. A mathematician may write . A computer, however, would **never** compute in order to find , but rather use one of many endlessly many numerical algorithms.
2. Working with sparse representations requires using a function that is aware of the representation you are using.

## Sparse Matrices and Sparse Models in R

### The Matrix Package

The **Matrix** package provides facilities to deal with real (stored as double precision), logical and so-called "pattern" (binary) dense and sparse matrices. There are provisions to provide integer and complex (stored as double precision complex) matrices.

The sparse matrix classes include:

* TsparseMatrix: a virtual class of the various sparse matrices in triplet representation.
* CsparseMatrix: a virtual class of the various sparse matrices in CSC representation.
* RsparseMatrix: a virtual class of the various sparse matrices in CSR representation.

For matrices of real numbers, stored in *double precision*, the **Matrix** package provides the following (non virtual) classes:

* dgTMatrix: a **general** sparse matrix of **doubles**, in **triplet** representation.
* dgCMatrix: a **general** sparse matrix of **doubles**, in **CSC** representation.
* dsCMatrix: a **symmetric** sparse matrix of **doubles**, in **CSC** representation.
* dtCMatrix: a **triangular** sparse matrix of **doubles**, in **CSC** representation.

Why bother with distinguishing between the different shapes of the matrix? Because the more structure is assumed on a matrix, the more our (statistical) algorithms can be optimized. For our purposes dgCMatrix will be the most useful.

### The glmnet Package

As previously stated, an efficient storage of the model.matrix is half of the story. We now need implementations of our favorite statistical algorithms that make use of this representation. At the time of writing, a very useful package that does that is the **glmnet** package, which allows to fit linear models, generalized linear models, with ridge, lasso, and elastic net regularization. The **glmnet** package allows all of this, using the sparse matrices of the **Matrix** package.

The following example is taken from [John Myles White's blog](http://www.johnmyleswhite.com/notebook/2011/10/31/using-sparse-matrices-in-r/), and compares the runtime of fitting an OLS model, using glmnet with both sparse and dense matrix representations.

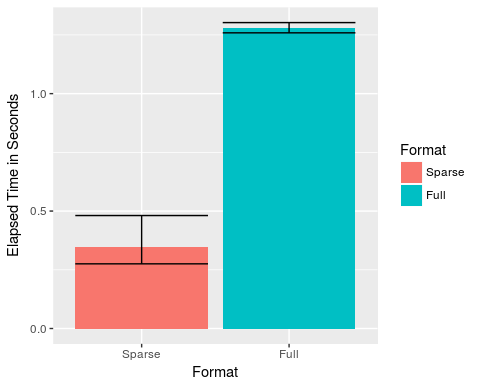
suppressPackageStartupMessages(library('glmnet'))  
  
set.seed(1)  
performance <- data.frame()  
   
for (sim in 1:10){  
 n <- 10000  
 p <- 500  
   
 nzc <- trunc(p / 10)  
   
 x <- matrix(rnorm(n \* p), n, p) #make a dense matrix  
 iz <- sample(1:(n \* p),  
 size = n \* p \* 0.85,  
 replace = FALSE)  
 x[iz] <- 0 # sparsify by injecting zeroes  
 sx <- Matrix(x, sparse = TRUE) # save as a sparse object  
   
 beta <- rnorm(nzc)  
 fx <- x[, seq(nzc)] %\*% beta  
   
 eps <- rnorm(n)  
 y <- fx + eps # make data  
   
 # Now to the actual model fitting:  
 sparse.times <- system.time(fit1 <- glmnet(sx, y)) # sparse glmnet  
 full.times <- system.time(fit2 <- glmnet(x, y)) # dense glmnet  
   
 sparse.size <- as.numeric(object.size(sx))  
 full.size <- as.numeric(object.size(x))  
   
 performance <- rbind(performance, data.frame(Format = 'Sparse',  
 UserTime = sparse.times[1],  
 SystemTime = sparse.times[2],  
 ElapsedTime = sparse.times[3],  
 Size = sparse.size))  
 performance <- rbind(performance, data.frame(Format = 'Full',  
 UserTime = full.times[1],  
 SystemTime = full.times[2],  
 ElapsedTime = full.times[3],  
 Size = full.size))  
}

Things to note:

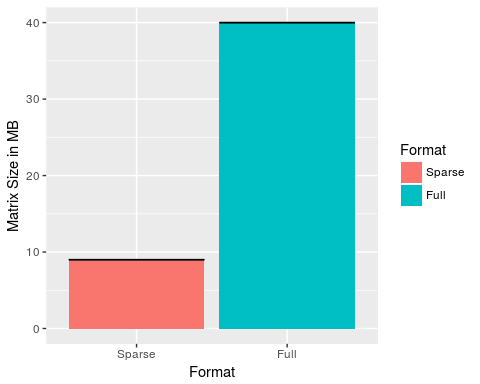
* The simulation calls glmnet twice. Once with the non-sparse object x, and once with its sparse version sx.
* The degree of sparsity of sx is . We know this because we "injected" zeroes in of the locations of x.
* Because y is continuous glmnet will fit a simple OLS model. We will see later how to use it to fit GLMs and use lasso, ridge, and elastic-net regularization.

We now inspect the computing time, and the memory footprint, only to discover that sparse representations make a BIG difference.

suppressPackageStartupMessages(library('ggplot2'))  
ggplot(performance, aes(x = Format, y = ElapsedTime, fill = Format)) +  
 stat\_summary(fun.data = 'mean\_cl\_boot', geom = 'bar') +  
 stat\_summary(fun.data = 'mean\_cl\_boot', geom = 'errorbar') +  
 ylab('Elapsed Time in Seconds')



ggplot(performance, aes(x = Format, y = Size / 1000000, fill = Format)) +  
 stat\_summary(fun.data = 'mean\_cl\_boot', geom = 'bar') +  
 stat\_summary(fun.data = 'mean\_cl\_boot', geom = 'errorbar') +  
 ylab('Matrix Size in MB')



How do we perform other types of regression with the **glmnet**? We just need to use the family and alpha arguments of glmnet::glmnet. The family argument governs the type of GLM to fit: logistic, Poisson, probit, or other types of GLM. The alpha argument controls the type of regularization. Set to alpha=0 for ridge, alpha=1 for lasso, and any value in between for elastic-net regularization.

## Bibliographic Notes

The best place to start reading on sparse representations and algorithms is the [vignettes](http://svitsrv25.epfl.ch/R-doc/library/Matrix/doc/) of the **Matrix** package. Gilbert, Moler, and Schreiber ([1992](#ref-gilbert1992sparse)) is also a great read for some general background. For the theory on solving sparse linear systems see Davis ([2006](#ref-davis2006direct)). For general numerical linear algebra see Gentle ([2012](#ref-gentle2012numerical)).

## Practice Yourself

# Memory Efficiency

As put by Kane et al. ([2013](#ref-kane2013scalable)), it was quite puzzling when very few of the competitors, for the Million dollars prize in the [Netflix challenge](https://en.wikipedia.org/wiki/Netflix_Prize), were statisticians. This is perhaps because the statistical community historically uses SAS, SPSS, and R. The first two tools are very well equipped to deal with big data, but are very unfriendly when trying to implement a new method. R, on the other hand, is very friendly for innovation, but was not equipped to deal with the large data sets of the Netflix challenge. A lot has changed in R since 2006. This is the topic of this chapter.

As we have seen in the Sparsity Chapter 14, an efficient representation of your data in RAM will reduce computing time, and will allow you to fit models that would otherwise require tremendous amounts of RAM. Not all problems are sparse however. It is also possible that your data does not fit in RAM, even if sparse. There are several scenarios to consider:

1. Your data fits in RAM, but is too big to compute with.
2. Your data does not fit in RAM, but fits in your local storage (HD, SSD, etc.)
3. Your data does not fit in your local storage.

If your data fits in RAM, but is too large to compute with, a solution is to replace the algorithm you are using. Instead of computing with the whole data, your algorithm will compute with parts of the data, also called *chunks*, or *batches*. These algorithms are known as *external memory algorithms* (EMA).

If your data does not fit in RAM, but fits in your local storage, you have two options. The first is to save your data in a *database management system* (DBMS). This will allow you to use the algorithms provided by your DBMS, or let R use an EMA while "chunking" from your DBMS. Alternatively, and preferably, you may avoid using a DBMS, and work with the data directly form your local storage by saving your data in some efficient manner.

Finally, if your data does not fit on you local storage, you will need some external storage solution such as a distributed DBMS, or distributed file system.

Remark. If you use Linux, you may be better of than Windows users. Linux will allow you to compute with larger datasets using its *swap file* that extends RAM using your HD or SSD. On the other hand, relying on the swap file is a BAD practice since it is much slower than RAM, and you can typically do much better using the tricks of this chapter. Also, while I LOVE Linux, I would never dare to recommend switching to Linux just to deal with memory contraints.

## Efficient Computing from RAM

If our data can fit in RAM, but is still too large to compute with it (recall that fitting a model requires roughly 5-10 times more memory than saving it), there are several facilities to be used. The first, is the sparse representation discussed in Chapter 14, which is relevant when you have factors, which will typically map to sparse model matrices. Another way is to use *external memory algorithms* (EMA).

The biglm::biglm function provides an EMA for linear regression. The following if taken from the function's example.

data(trees)  
ff<-log(Volume)~log(Girth)+log(Height)  
  
chunk1<-trees[1:10,]  
chunk2<-trees[11:20,]  
chunk3<-trees[21:31,]  
  
library(biglm)  
a <- biglm(ff,chunk1)  
a <- update(a,chunk2)  
a <- update(a,chunk3)  
  
coef(a)

## (Intercept) log(Girth) log(Height)   
## -6.631617 1.982650 1.117123

Things to note:

* The data has been chunked along rows.
* The initial fit is done with the biglm function.
* The model is updated with further chunks using the update function.

We now compare it to the in-memory version of lm to verify the results are the same.

b <- lm(ff, data=trees)  
rbind(coef(a),coef(b))

## (Intercept) log(Girth) log(Height)  
## [1,] -6.631617 1.98265 1.117123  
## [2,] -6.631617 1.98265 1.117123

Other packages that follow these lines, particularly with classification using SVMs, are **LiblineaR**, and **RSofia**.

### Summary Statistics from RAM

If you are not going to do any model fitting, and all you want is efficient filtering, selection and summary statistics, then a lot of my warnings above are irrelevant. For these purposes, the facilities provided by **base**, **stats**, and **dplyr** are probably enough. If the data is large, however, these facilities may be too slow. If your data fits into RAM, but speed bothers you, take a look at the **data.table** package. The syntax is less friendly than **dplyr**, but **data.table** is BLAZING FAST compared to competitors. Here is a little benchmark[[28]](#footnote-457).

First, we setup the data.

library(data.table)  
  
n <- 1e6 # number of rows  
k <- c(200,500) # number of distinct values for each 'group\_by' variable  
p <- 3 # number of variables to summarize  
  
L1 <- sapply(k, function(x) as.character(sample(1:x, n, replace = TRUE) ))  
L2 <- sapply(1:p, function(x) rnorm(n) )  
  
tbl <- data.table(L1,L2) %>%   
 setnames(c(paste("v",1:length(k),sep=""), paste("x",1:p,sep="") ))  
  
tbl\_dt <- tbl  
tbl\_df <- tbl %>% as.data.frame

We compare the aggregation speeds. Here is the timing for **dplyr**.

system.time( tbl\_df %>%   
 group\_by(v1,v2) %>%   
 summarize(  
 x1 = sum(abs(x1)),   
 x2 = sum(abs(x2)),   
 x3 = sum(abs(x3))   
 )  
 )

## user system elapsed   
## 9.268 0.008 9.285

And now the timing for **data.table**.

system.time(   
 tbl\_dt[ , .( x1 = sum(abs(x1)), x2 = sum(abs(x2)), x3 = sum(abs(x3)) ), .(v1,v2)]  
 )

## user system elapsed   
## 0.364 0.000 0.363

The winner is obvious. Let's compare filtering (i.e. row subsets, i.e. SQL's SELECT).

system.time(   
 tbl\_df %>% filter(v1 == "1")   
 )

## user system elapsed   
## 0.020 0.000 0.023

system.time(   
 tbl\_dt[v1 == "1"]   
 )

## user system elapsed   
## 0.016 0.000 0.014

## Computing from a Database

The early solutions to oversized data relied on storing your data in some DBMS such as *MySQL*, *PostgresSQL*, *SQLite*, *H2*, *Oracle*, etc. Several R packages provide interfaces to these DBMSs, such as **sqldf**, **RDBI**, **RSQite**. Some will even include the DBMS as part of the package itself.

Storing your data in a DBMS has the advantage that you can typically rely on DBMS providers to include very efficient algorithms for the queries they support. On the downside, SQL queries may include a lot of summary statistics, but will rarely include model fitting[[29]](#footnote-459). This means that even for simple things like linear models, you will have to revert to R's facilities-- typically some sort of EMA with chunking from the DBMS. For this reason, and others, we prefer to compute from efficient file structures, as described in Section 15.3.

If, however, you have a powerful DBMS around, or you only need summary statistics, or you are an SQL master, keep reading.

The package **RSQLite** includes an SQLite server, which we now setup for demonstration. The package **dplyr**, discussed in the Hadleyverse Chapter 13, will take care of translating the **dplyr** syntax, to the SQL syntax of the DBMS. The following example is taken from the **dplyr** [Databases vignette](https://cran.r-project.org/web/packages/dplyr/vignettes/databases.html).

library(RSQLite)  
library(dplyr)  
  
file.remove('my\_db.sqlite3')

## [1] TRUE

my\_db <- src\_sqlite(path = "my\_db.sqlite3", create = TRUE)  
  
library(nycflights13)  
flights\_sqlite <- copy\_to(  
 dest= my\_db,   
 df= flights,   
 temporary = FALSE,   
 indexes = list(c("year", "month", "day"), "carrier", "tailnum"))

Things to note:

* src\_sqlite to start an empty table, managed by SQLite, at the desired path.
* copy\_to copies data from R to the database.
* Typically, setting up a DBMS like this makes no sense, since it requires loading the data into RAM, which is precisely what we want to avoid.

We can now start querying the DBMS.

select(flights\_sqlite, year:day, dep\_delay, arr\_delay)

## # Source: lazy query [?? x 5]  
## # Database: sqlite 3.19.3 [my\_db.sqlite3]  
## year month day dep\_delay arr\_delay  
## <int> <int> <int> <dbl> <dbl>  
## 1 2013 1 1 2 11  
## 2 2013 1 1 4 20  
## 3 2013 1 1 2 33  
## 4 2013 1 1 -1 -18  
## 5 2013 1 1 -6 -25  
## 6 2013 1 1 -4 12  
## 7 2013 1 1 -5 19  
## 8 2013 1 1 -3 -14  
## 9 2013 1 1 -3 -8  
## 10 2013 1 1 -2 8  
## # ... with more rows

filter(flights\_sqlite, dep\_delay > 240)

## # Source: lazy query [?? x 19]  
## # Database: sqlite 3.19.3 [my\_db.sqlite3]  
## year month day dep\_time sched\_dep\_time dep\_delay arr\_time  
## <int> <int> <int> <int> <int> <dbl> <int>  
## 1 2013 1 1 848 1835 853 1001  
## 2 2013 1 1 1815 1325 290 2120  
## 3 2013 1 1 1842 1422 260 1958  
## 4 2013 1 1 2115 1700 255 2330  
## 5 2013 1 1 2205 1720 285 46  
## 6 2013 1 1 2343 1724 379 314  
## 7 2013 1 2 1332 904 268 1616  
## 8 2013 1 2 1412 838 334 1710  
## 9 2013 1 2 1607 1030 337 2003  
## 10 2013 1 2 2131 1512 379 2340  
## # ... with more rows, and 12 more variables: sched\_arr\_time <int>,  
## # arr\_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,  
## # origin <chr>, dest <chr>, air\_time <dbl>, distance <dbl>, hour <dbl>,  
## # minute <dbl>, time\_hour <dbl>

summarise(flights\_sqlite, delay = mean(dep\_time))

## # Source: lazy query [?? x 1]  
## # Database: sqlite 3.19.3 [my\_db.sqlite3]  
## delay  
## <dbl>  
## 1 1349.11

## Computing From Efficient File Structrures

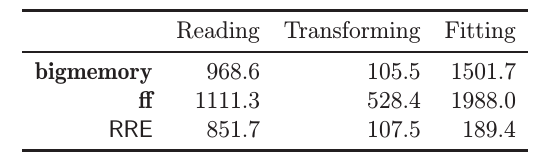
It is possible to save your data on your storage device, without the DBMS layer to manage it. This has several advantages:

* You don't need to manage a DBMS.
* You don't have the computational overhead of the DBMS.
* You may optimize the file structure for statistical modelling, and not for join and summary operations, as in relational DBMSs.

There are several facilities that allow you to save and compute directly from your storage:

1. **Memory Mapping**: Where RAM addresses are mapped to a file on your storage. This extends the RAM to the capacity of your storage (HD, SSD,...). Performance slightly deteriorates, but the access is typically very fast. This approach is implemented in the **bigmemory** package.
2. **Efficient Binaries**: Where the data is stored as a file on the storage device. The file is binary, with a well designed structure, so that chunking is easy. This approach is implemented in the **ff** package, and the commercial **RevoScaleR** package.

Your algorithms need to be aware of the facility you are using. For this reason each facility ( **bigmemory**, **ff**, **RevoScaleR**,...) has an eco-system of packages that implement various statistical methods using that facility. As a general rule, you can see which package builds on a package using the *Reverse Depends* entry in the package description. For the **bigmemory** package, for instance, [we can see](https://cran.r-project.org/web/packages/bigmemory/index.html) that the packages **bigalgebra**, **biganalytics**, **bigFastlm**, **biglasso**, **bigpca**, **bigtabulate**, **GHap**, and **oem**, build upon it. We can expect this list to expand.

Here is a benchmark result, from Wang et al. ([2015](#ref-wang2015statistical)). It can be seen that **ff** and **bigmemory** have similar performance, while **RevoScaleR** (RRE in the figure) outperforms them. This has to do both with the efficiency of the binary representation, but also because **RevoScaleR** is inherently parallel. More on this in the Parallelization Chapter 16. 

### bigmemory

We now demonstrate the workflow of the **bigmemory** package. We will see that **bigmemory**, with it's big.matrix object is a very powerful mechanism. If you deal with big numeric matrices, you will find it very useful. If you deal with big data frames, or any other non-numeric matrix, **bigmemory** may not be the appropriate tool, and you should try **ff**, or the commercial **RevoScaleR**.

# download.file("http://www.cms.gov/Research-Statistics-Data-and-Systems/Statistics-Trends-and-Reports/BSAPUFS/Downloads/2010\_Carrier\_PUF.zip", "2010\_Carrier\_PUF.zip")  
# unzip(zipfile="2010\_Carrier\_PUF.zip")  
  
library("bigmemory")  
x <- read.big.matrix("data/2010\_BSA\_Carrier\_PUF.csv", header = TRUE,   
 backingfile = "airline.bin",   
 descriptorfile = "airline.desc",   
 type = "integer")  
dim(x)

## [1] 2801660 11

pryr::object\_size(x)

## 616 B

class(x)

## [1] "big.matrix"  
## attr(,"package")  
## [1] "bigmemory"

Things to note:

* The basic building block of the **bigmemory** ecosystem, is the big.matrix class, we constructed with read.big.matrix.
* read.big.matrix handles the import to R, and the saving to a memory mapped file. The implementation is such that at no point does R hold the data in RAM.
* The memory mapped file will be there after the session is over. It can thus be called by other R sessions using attach.big.matrix("airline.desc"). This will be useful when parallelizing.
* pryr::object\_size return the size of the object. Since x holds only the memory mappings, it is much smaller than the 100MB of data that it holds.

We can now start computing with the data. Many statistical procedures for the big.matrix object are provided by the **biganalytics** package. In particular, the biglm.big.matrix and bigglm.big.matrix functions, provide an interface from big.matrix objects, to the EMA linear models in biglm::biglm and biglm::bigglm.

library(biganalytics)  
biglm.2 <- bigglm.big.matrix(BENE\_SEX\_IDENT\_CD~CAR\_LINE\_HCPCS\_CD, data=x)  
coef(biglm.2)

## (Intercept) CAR\_LINE\_HCPCS\_CD   
## 1.537848e+00 1.210282e-07

Other notable packages that operate with big.matrix objects include:

* **bigtabulate**: Extend the bigmemory package with 'table', 'tapply', and 'split' support for 'big.matrix' objects.
* **bigalgebra**: For matrix operation.
* **bigpca**: principle components analysis (PCA), or singular value decomposition (SVD).
* **bigFastlm**: for (fast) linear models.
* **biglasso**: extends lasso and elastic nets.
* **GHap**: Haplotype calling from phased SNP data.

## ff

The **ff** packages replaces R's in-RAM storage mechanism with on-disk (efficient) storage. Unlike **bigmemory**, **ff** supports all of R vector types such as factors, and not only numeric. Unlike big.matrix, which deals with (numeric) matrices, the ffdf class can deal with data frames.

Here is an example. First open a connection to the file, without actually importing it using the LaF::laf\_open\_csv function.

.dat <- LaF::laf\_open\_csv(filename = "data/2010\_BSA\_Carrier\_PUF.csv",  
 column\_types = c("integer", "integer", "categorical", "categorical", "categorical", "integer", "integer", "categorical", "integer", "integer", "integer"),   
 column\_names = c("sex", "age", "diagnose", "healthcare.procedure", "typeofservice", "service.count", "provider.type", "servicesprocessed", "place.served", "payment", "carrierline.count"),   
 skip = 1)

Now write the data to local storage as an ff data frame, using laf\_to\_ffdf.

data.ffdf <- ffbase::laf\_to\_ffdf(laf = .dat)  
head(data.ffdf)

## ffdf (all open) dim=c(2801660,6), dimorder=c(1,2) row.names=NULL  
## ffdf virtual mapping  
## PhysicalName VirtualVmode PhysicalVmode AsIs  
## sex sex integer integer FALSE  
## age age integer integer FALSE  
## diagnose diagnose integer integer FALSE  
## healthcare.procedure healthcare.procedure integer integer FALSE  
## typeofservice typeofservice integer integer FALSE  
## service.count service.count integer integer FALSE  
## VirtualIsMatrix PhysicalIsMatrix PhysicalElementNo  
## sex FALSE FALSE 1  
## age FALSE FALSE 2  
## diagnose FALSE FALSE 3  
## healthcare.procedure FALSE FALSE 4  
## typeofservice FALSE FALSE 5  
## service.count FALSE FALSE 6  
## PhysicalFirstCol PhysicalLastCol PhysicalIsOpen  
## sex 1 1 TRUE  
## age 1 1 TRUE  
## diagnose 1 1 TRUE  
## healthcare.procedure 1 1 TRUE  
## typeofservice 1 1 TRUE  
## service.count 1 1 TRUE  
## ffdf data  
## sex age diagnose healthcare.procedure typeofservice  
## 1 1 1 NA 99213 M1B   
## 2 1 1 NA A0425 O1A   
## 3 1 1 NA A0425 O1A   
## 4 1 1 NA A0425 O1A   
## 5 1 1 NA A0425 O1A   
## 6 1 1 NA A0425 O1A   
## 7 1 1 NA A0425 O1A   
## 8 1 1 NA A0425 O1A   
## : : : : : :  
## 2801653 2 6 V82 85025 T1D   
## 2801654 2 6 V82 87186 T1H   
## 2801655 2 6 V82 99213 M1B   
## 2801656 2 6 V82 99213 M1B   
## 2801657 2 6 V82 A0429 O1A   
## 2801658 2 6 V82 G0328 T1H   
## 2801659 2 6 V86 80053 T1B   
## 2801660 2 6 V88 76856 I3B   
## service.count  
## 1 1   
## 2 1   
## 3 1   
## 4 2   
## 5 2   
## 6 3   
## 7 3   
## 8 4   
## : :  
## 2801653 1   
## 2801654 1   
## 2801655 1   
## 2801656 1   
## 2801657 1   
## 2801658 1   
## 2801659 1   
## 2801660 1

We can verify that the ffdf data frame has a small RAM footprint.

pryr::object\_size(data.ffdf)

## 343 kB

The **ffbase** package provides several statistical tools to compute with ff class objects. Here is simple table.

ffbase:::table.ff(data.ffdf$age)

##   
## 1 2 3 4 5 6   
## 517717 495315 492851 457643 419429 418705

The EMA implementation of biglm::biglm and biglm::bigglm have their **ff** versions.

library(biglm)  
mymodel.ffdf <- biglm(payment ~ factor(sex) + factor(age) + place.served,   
 data = data.ffdf)  
summary(mymodel.ffdf)

## Large data regression model: biglm(payment ~ factor(sex) + factor(age) + place.served, data = data.ffdf)  
## Sample size = 2801660   
## Coef (95% CI) SE p  
## (Intercept) 97.3313 96.6412 98.0214 0.3450 0.0000  
## factor(sex)2 -4.2272 -4.7169 -3.7375 0.2449 0.0000  
## factor(age)2 3.8067 2.9966 4.6168 0.4050 0.0000  
## factor(age)3 4.5958 3.7847 5.4070 0.4056 0.0000  
## factor(age)4 3.8517 3.0248 4.6787 0.4135 0.0000  
## factor(age)5 1.0498 0.2030 1.8965 0.4234 0.0132  
## factor(age)6 -4.8313 -5.6788 -3.9837 0.4238 0.0000  
## place.served -0.6132 -0.6253 -0.6012 0.0060 0.0000

Things to note:

* biglm::biglm notices the input of of class ffdf and calls the appropriate implementation.
* The model formula, payment ~ factor(sex) + factor(age) + place.served, includes factors which cause no difficulty.
* You cannot inspect the factor coding (dummy? effect?) using model.matrix. This is because EMAs never really construct the whole matrix, let alone, save it in memory.

### matter

TODO

### iotools

TODO

## Computing from a Distributed File System

If your data is SOOO big that it cannot fit on your local storage, you will need a distributed file system or DBMS. We do not cover this topic here, and refer the reader to the **RHipe**, **RHadoop**, and **RSpark** packages and references therein.

## Bibliographic Notes

An absolute SUPERB review on computing with big data is Wang et al. ([2015](#ref-wang2015statistical)), and references therein (Kane et al. ([2013](#ref-kane2013scalable)) in particular). For an up-to-date list of the packages that deal with memory constraints, see the **Large memory and out-of-memory data** section in the High Performance Computing [R task view](https://cran.r-project.org/web/views/HighPerformanceComputing.html).

## Practice Yourself

# Parallel Computing

You would think that because you have an expensive multicore computer your computations will speed up. Well, no. At least not if you don't make sure they do. By default, no matter how many cores you have, the operating system will allocate each R session to a single core.

For starters, we need to distinguish between two types of parallelism:

1. **Explicit parallelism**: where the user handles the parallelisation.
2. **Implicit parallelism**: where the parallelisation is abstracted away from the user.

Clearly, implicit parallelism is more desirable, but the state of mathematical computing is such that no sufficiently general implicit parallelism framework exists. The [R Consortium](https://www.r-consortium.org/projects/awarded-projects) is currently financing a major project for a A "Unified Framework For Distributed Computing in R" so we can expect things to change soon. In the meanwhile, most of the parallel implementations are explicit.

## Implicit Parallelism

We will not elaborate on implicit parallelism except mentioning the following:

* You can enjoy parallel linear algebra by replacing the linear algebra libraries with BLAS and LAPACK as described [here](https://www.r-bloggers.com/faster-r-through-better-blas/).
* You should read the "Parallel computing: Implicit parallelism" section in the excellent [High Performance Computing](https://cran.r-project.org/web/views/HighPerformanceComputing.html) task view, for the latest developments in implicit parallelism.

## Explicit Parallelism

R provides many frameworks for explicit parallelism. Because the parallelism is initiated by the user, we first need to decide **when to parallelize?** As a rule of thumb, you want to parallelise when you encounter a CPU bottleneck, and not a memory bottleneck. Memory bottlenecks are released with sparsity (Chapter 14), or efficient memory usage (Chapter 15).

Several ways to diagnose your bottleneck include:

* Keep your Windows Task Manager, or Linux top open, and look for the CPU load, and RAM loads.
* The computation takes a long time, and when you stop it pressing ESC, R is immediately responsive. If it is not immediately responsive, you have a memory bottleneck.
* Profile your code. See Hadley's [guide](http://adv-r.had.co.nz/Profiling.html).

For reasons detailed in Kane et al. ([2013](#ref-kane2013scalable)), we will present the **foreach** parallelisation package (Analytics and Weston [2015](#ref-foreach)). It will allow us to:

1. Decouple between our parallel algorithm and the parallelisation mechanism: we write parallelisable code once, and can then switch the underlying parallelisation mechanism.
2. Combine with the big.matrix object from Chapter 15 for *shared memory parallisation*: all the machines may see the same data, so that we don't need to export objects from machine to machine.

What do we mean by "switch the underlying parallesation mechanism"? It means there are several packages that will handle communication between machines. Some are very general and will work on any cluster. Some are more specific and will work only on a single multicore machine (not a cluster) with a particular operating system. These mechanisms include **multicore**, **snow**, **parallel**, and **Rmpi**. The compatibility between these mechanisms and **foreach** is provided by another set of packages: **doMC** , **doMPI**, **doRedis**, **doParallel**, and **doSNOW**.

Remark. I personally prefer the **multicore** mechanism, with the **doMC** adapter for **foreach**. I will not use this combo, however, because **multicore** will not work on Windows machines. I will thus use the more general **snow** and **doParallel** combo. If you do happen to run on Linux, or Unix, you will want to replace all **doParallel** functionality with **doMC**.

Let's start with a simple example, taken from ["Getting Started with doParallel and foreach"](http://debian.mc.vanderbilt.edu/R/CRAN/web/packages/doParallel/vignettes/gettingstartedParallel.pdf).

library(doParallel)  
cl <- makeCluster(2)  
registerDoParallel(cl)  
result <- foreach(i=1:3) %dopar% sqrt(i)  
class(result)

## [1] "list"

result

## [[1]]  
## [1] 1  
##   
## [[2]]  
## [1] 1.414214  
##   
## [[3]]  
## [1] 1.732051

Things to note:

* makeCluster creates an object with the information our cluster. On a single machine it is very simple. On a cluster of machines, you will need to specify the i.p. addresses or other identifiers of the machines.
* registerDoParallel is used to inform the **foreach** package of the presence of our cluster.
* The foreach function handles the looping. In particular note the %dopar operator that ensures that looping is in parallel. %dopar% can be replaced by %do% if you want serial looping (like the for loop), for instance, for debugging.
* The output of the various machines is collected by foreach to a list object.
* In this simple example, no data is shared between machines so we are not putting the shared memory capabilities to the test.
* We can check how many workers were involved using the getDoParWorkers() function.
* We can check the parallelisation mechanism used with the getDoParName() function.

Here is a more involved example. We now try to make [Bootstrap](https://en.wikipedia.org/wiki/Bootstrapping_(statistics)) inference on the coefficients of a logistic regression. Bootstrapping means that in each iteration, we resample the data, and refit the model.

x <- iris[which(iris[,5] != "setosa"), c(1,5)]  
trials <- 1e4  
ptime <- system.time({  
 r <- foreach(icount(trials), .combine=cbind) %dopar% {  
 ind <- sample(100, 100, replace=TRUE)  
 result1 <- glm(x[ind,2]~x[ind,1], family=binomial(logit))  
 coefficients(result1)  
 }  
 })[3]  
ptime

## elapsed   
## 21.525

Things to note:

* As usual, we use the foreach function with the %dopar% operator to loop in parallel.
* The icounts function generates a counter.
* The .combine=cbind argument tells the foreach function how to combine the output of different machines, so that the returned object is not the default list.

How long would that have taken in a simple (serial) loop? We only need to replace %dopar% with %do% to test.

stime <- system.time({  
 r <- foreach(icount(trials), .combine=cbind) %do% {  
 ind <- sample(100, 100, replace=TRUE)  
 result1 <- glm(x[ind,2]~x[ind,1], family=binomial(logit))  
 coefficients(result1)  
 }  
 })[3]  
stime

## elapsed   
## 41.861

Yes. Parallelising is clearly faster.

Let's see how we can combine the power of **bigmemory** and **foreach** by creating a file mapped big.matrix object, which is shared by all machines. The following example is taken from Kane et al. ([2013](#ref-kane2013scalable)), and uses the big.matrix object we created in Chapter 15.

library(bigmemory)  
x <- attach.big.matrix("airline.desc")  
  
library(foreach)  
library(doSNOW)  
cl <- makeSOCKcluster(rep("localhost", 4)) # make a cluster of 4 machines  
registerDoSNOW(cl) # register machines for foreach()

Get a "description" of the big.matrix object that will be used to call it from each machine.

xdesc <- describe(x)

Split the data along values of BENE\_AGE\_CAT\_CD.

G <- split(1:nrow(x), x[, "BENE\_AGE\_CAT\_CD"])

Define a function that computes quantiles of CAR\_LINE\_ICD9\_DGNS\_CD.

GetDepQuantiles <- function(rows, data) {  
 quantile(data[rows, "CAR\_LINE\_ICD9\_DGNS\_CD"], probs = c(0.5, 0.9, 0.99),  
 na.rm = TRUE)  
}

We are all set up to loop, in parallel, and compute quantiles of CAR\_LINE\_ICD9\_DGNS\_CD for each value of BENE\_AGE\_CAT\_CD.

qs <- foreach(g = G, .combine = rbind) %dopar% {  
 require("bigmemory")  
 x <- attach.big.matrix(xdesc)  
 GetDepQuantiles(rows = g, data = x)  
}  
qs

## 50% 90% 99%  
## result.1 558 793 996  
## result.2 518 789 996  
## result.3 514 789 996  
## result.4 511 789 996  
## result.5 511 790 996  
## result.6 518 796 995

### Caution: Implicit with Explicit Parallelism

TODO

## Bibliographic Notes

For a brief and excellent explanation on parallel computing in R see Schmidberger et al. ([2009](#ref-schmidberger2009state)). For a full review see Chapple et al. ([2016](#ref-chapple2016mastering)). For an up-to-date list of packages supporting parallel programming see the High Performance Computing [R task view](https://cran.r-project.org/web/views/HighPerformanceComputing.html).

## Practice Yourself

# Numerical Linear Algebra

In your algebra courses you would write and solve . This is useful to understand the algebraic properties of , but a computer would never recover that way. Even the computation of the sample variance, is not solved that way in a computer, because of numerical and speed considerations.

In this chapter, we discuss several ways a computer solves systems of linear equations, with their application to statistics, namely, to OLS problems.

## LU Factorization

Definition 18 (LU Factorization) For some matrix , the LU factorization is defined as

where is unit lower triangular and is upper triangular.

The LU factorization is essentially the matrix notation for the [Gaussian elimination](https://en.wikipedia.org/wiki/Gaussian_elimination) you did in your introductory algebra courses.

For a square matrix, the LU factorization requires operations, and stores elements in memory.

## Cholesky Factorization

Definition 19 (Non Negative Matrix) A matrix is said to be *non-negative* if for all .

Seeing the matrix as a function, non-negative matrices can be thought of as functions that generalize the *squaring* operation.

Definition 20 (Cholesky Factorization) For some non-negative matrix , the Cholesky factorization is defined as

where is upper triangular with positive diagonal elements.

For obvious reasons, the Cholesky factorization is known as the *square root* of a matrix.

Because Cholesky is less general than LU, it is also more efficient. It can be computed in operations, and requires storing elements.

## QR Factorization

Definition 21 (QR Factorization) For some matrix , the QR factorization is defined as

where is orthogonal and is upper triangular.

The QR factorization is very useful to solve the OLS problem as we will see in 17.6. The QR factorization takes operations to compute. Three major methods for computing the QR factorization exist. These rely on *Householder transformations*, *Givens transformations*, and a (modified) *Gram-Schmidt procedure* (Gentle [2012](#ref-gentle2012numerical)).

## Singular Value Factorization

Definition 22 (SVD) For an arbitrary matrix , the *singular valued decomposition* (SVD), is defined as

$$ where is an orthonormal matrix, is an orthonormal matrix, and is diagonal.

The SVD factorization is very useful for algebraic analysis, but less so for computations. This is because it is (typically) solved via the QR factorization.

## Iterative Methods

The various matrix factorizations above may be used to solve a system of linear equations, and in particular, the OLS problem. There is, however, a very different approach to solving systems of linear equations. This approach relies on the fact that solutions of linear systems of equations, can be cast as optimization problems: simply find by minimizing .

Some methods for solving (convex) optimization problems are reviewed in the Convex Optimization Chapter @(convex). For our purposes we will just mention that historically (this means in the lm function, and in the LAPACK numerical libraries) the factorization approach was preferred, and now optimization approaches are preferred. This is because the optimization approach is more numerically stable, and easier to parallelize.

## Solving the OLS Problem

Recalling the OLS problem in Eq.(5): we wish to find such that

The solution, that solves this problem has to satisfy

Eq.(18) are known as the *normal equations*. The normal equations are the link between the OLS problem, and the matrix factorization discussed above.

Using the QR decomposition in the normal equations we have that

where is the

## Bibliographic Notes

For an excellent introduction to numerical algorithms in statistics, see Weihs, Mersmann, and Ligges ([2013](#ref-weihs2013foundations)). For an emphasis on numerical linear algebra, see Gentle ([2012](#ref-gentle2012numerical)), and Golub and Van Loan ([2012](#ref-golub2012matrix)).

## Practice Yourself

# Convex Optimization

## Bibliographic Notes

## Practice Yourself

# RCpp

## Bibliographic Notes

## Practice Yourself

# Debugging Tools

TODO

## Bibliographic Notes

## Practice Yourself

# data.table

data.table is an excellent extension of the data.frame class. If used as a data.frame it will look and feel like a data frame. If, however, it is used with it's unique capabilities, it will prove faster and easier to manipulate.

Let's start with importing some freely available car sales data from [Kaggle](https://www.kaggle.com/orgesleka/used-cars-database).

library(data.table)  
library(magrittr)  
auto <- fread('data/autos.csv')

View(auto)

dim(auto) # Rows and columns

## [1] 371824 20

names(auto) # Variable names

## [1] "dateCrawled" "name" "seller"   
## [4] "offerType" "price" "abtest"   
## [7] "vehicleType" "yearOfRegistration" "gearbox"   
## [10] "powerPS" "model" "kilometer"   
## [13] "monthOfRegistration" "fuelType" "brand"   
## [16] "notRepairedDamage" "dateCreated" "nrOfPictures"   
## [19] "postalCode" "lastSeen"

class(auto) # Object class

## [1] "data.table" "data.frame"

file.info('data/autos.csv') # File info on disk

## size isdir mode mtime ctime  
## data/autos.csv 68439217 FALSE 664 2016-11-28 15:47:00 2017-10-30 22:48:56  
## atime uid gid uname grname  
## data/autos.csv 2017-10-30 22:48:59 1000 1000 johnros johnros

object.size(auto) %>% print(humanReadable=TRUE, units = 'MB') # File size in memory

## 97.9 Mb

Things to note:

* The import has been done with fread instead of read.csv. This is more efficient, and directly creates a data.table object.
* The import is very fast.

Let's start with verifying that it behaves like a data.frame when expected.

auto[,2] %>% head

## name  
## 1: Golf\_3\_1.6  
## 2: A5\_Sportback\_2.7\_Tdi  
## 3: Jeep\_Grand\_Cherokee\_"Overland"  
## 4: GOLF\_4\_1\_4\_\_3T\xdcRER  
## 5: Skoda\_Fabia\_1.4\_TDI\_PD\_Classic  
## 6: BMW\_316i\_\_\_e36\_Limousine\_\_\_Bastlerfahrzeug\_\_Export

auto[1,2] %>% head

## name  
## 1: Golf\_3\_1.6

auto[[2]] %>% head

## [1] "Golf\_3\_1.6"   
## [2] "A5\_Sportback\_2.7\_Tdi"   
## [3] "Jeep\_Grand\_Cherokee\_\"Overland\""   
## [4] "GOLF\_4\_1\_4\_\_3T\xdcRER"   
## [5] "Skoda\_Fabia\_1.4\_TDI\_PD\_Classic"   
## [6] "BMW\_316i\_\_\_e36\_Limousine\_\_\_Bastlerfahrzeug\_\_Export"

But notice the difference between data.frame and data.table when subsetting multiple rows. You should always be explicit when you subset rows/columns!

auto[1:3] %>% dim # data.table will exctract \*rows\*

## [1] 3 20

as.data.frame(auto)[1:3] %>% dim # data.frame will exctract \*columns\*

## [1] 371824 3

Now let's do some data.table specific operations. The general syntax has the form DT[i,j,by]. SQL users may think of i as WHERE, j as SELECT, and by as GROUP BY. We don't need to name the arguments explicitly.

auto[,vehicleType,] %>% table # exctract column

## .  
## andere bus cabrio coupe kleinwagen   
## 37899 3362 30220 22914 19026 80098   
## kombi limousine suv   
## 67626 95963 14716

auto[vehicleType=='coupe',,] %>% dim # exctract rows

## [1] 19026 20

auto[,gearbox,] %>% table

## .  
## automatik manuell   
## 20223 77169 274432

auto[vehicleType=='coupe' & gearbox=='automatik',,] %>% dim # intersect conditions

## [1] 6008 20

auto[, mean(price), by=vehicleType] # average price per car group

## Warning in gmean(price): Group 9 summed to more than type 'integer'  
## can hold so the result has been coerced to 'numeric' automatically, for  
## convenience.

## vehicleType V1  
## 1: 20124.688  
## 2: coupe 25951.506  
## 3: suv 13252.392  
## 4: kleinwagen 5691.167  
## 5: limousine 11111.107  
## 6: cabrio 15072.998  
## 7: bus 10300.686  
## 8: kombi 7739.518  
## 9: andere 676327.100

auto[,gearbox:model,] %>% head # exctract column range

## gearbox powerPS model  
## 1: manuell 0 golf  
## 2: manuell 190   
## 3: automatik 163 grand  
## 4: manuell 75 golf  
## 5: manuell 69 fabia  
## 6: manuell 102 3er

The .N operator is very useful if you need to count the length of the result:

auto[.N-1,,] # will exctract the \*last\* row

## dateCrawled name seller offerType price  
## 1: 2016-03-20 19:41:08 VW\_Golf\_Kombi\_1\_9l\_TDI privat Angebot 3400  
## abtest vehicleType yearOfRegistration gearbox powerPS model kilometer  
## 1: test kombi 2002 manuell 100 golf 150000  
## monthOfRegistration fuelType brand notRepairedDamage  
## 1: 6 diesel volkswagen   
## dateCreated nrOfPictures postalCode lastSeen  
## 1: 2016-03-20 00:00:00 0 40764 2016-03-24 12:45:21

auto[,.N] # will count rows per type

## [1] 371824

You may concatenate results into a vector:

auto[,c(mean(price), mean(powerPS)),]

## [1] 17286.2996 115.5414

This c() syntax does not behave well is splitting:

auto[,c(mean(price), mean(powerPS)), by=vehicleType]

## vehicleType V1  
## 1: 20124.68801  
## 2: 71.23249  
## 3: coupe 25951.50589  
## 4: coupe 172.97614  
## 5: suv 13252.39182  
## 6: suv 166.01903  
## 7: kleinwagen 5691.16738  
## 8: kleinwagen 68.75733  
## 9: limousine 11111.10661  
## 10: limousine 132.26936  
## 11: cabrio 15072.99782  
## 12: cabrio 145.17684  
## 13: bus 10300.68561  
## 14: bus 113.58137  
## 15: kombi 7739.51760  
## 16: kombi 136.40654  
## 17: andere 676327.09964  
## 18: andere 102.11154

data.table' uses the.()` syntax for concatenating (actually, for listing):

auto[,.(mean(price), mean(powerPS)), by=vehicleType]

## Warning in gmean(price): Group 9 summed to more than type 'integer'  
## can hold so the result has been coerced to 'numeric' automatically, for  
## convenience.

## vehicleType V1 V2  
## 1: 20124.688 71.23249  
## 2: coupe 25951.506 172.97614  
## 3: suv 13252.392 166.01903  
## 4: kleinwagen 5691.167 68.75733  
## 5: limousine 11111.107 132.26936  
## 6: cabrio 15072.998 145.17684  
## 7: bus 10300.686 113.58137  
## 8: kombi 7739.518 136.40654  
## 9: andere 676327.100 102.11154

An equivalent formulation, replacing .() with an explicit list():

auto[,list(mean(price), mean(powerPS)), by=vehicleType]

You may include column names

auto[,.(Price=mean(price), Power=mean(powerPS)), by=vehicleType]

## Warning in gmean(price): Group 9 summed to more than type 'integer'  
## can hold so the result has been coerced to 'numeric' automatically, for  
## convenience.

## vehicleType Price Power  
## 1: 20124.688 71.23249  
## 2: coupe 25951.506 172.97614  
## 3: suv 13252.392 166.01903  
## 4: kleinwagen 5691.167 68.75733  
## 5: limousine 11111.107 132.26936  
## 6: cabrio 15072.998 145.17684  
## 7: bus 10300.686 113.58137  
## 8: kombi 7739.518 136.40654  
## 9: andere 676327.100 102.11154

Or split by multiple variables:

auto[,.(Price=mean(price), Power=mean(powerPS)), by=.(vehicleType,fuelType)]

## Warning in gmean(price): Group 37 summed to more than type 'integer'  
## can hold so the result has been coerced to 'numeric' automatically, for  
## convenience.

## vehicleType fuelType Price Power  
## 1: benzin 11820.443 70.144766  
## 2: coupe diesel 51170.248 179.487039  
## 3: suv diesel 15549.369 168.161149  
## 4: kleinwagen benzin 5786.514 68.743090  
## 5: kleinwagen diesel 4295.550 76.836660  
## 6: limousine benzin 6974.360 127.870246  
## 7: cabrio benzin 10673.413 147.063874  
## 8: bus benzin 5007.837 112.997127  
## 9: kleinwagen 6575.304 57.272879  
## 10: kombi diesel 7717.390 143.084948  
## 11: kombi benzin 8505.697 129.593180  
## 12: kombi 3194.817 123.230171  
## 13: bus diesel 13671.089 115.516662  
## 14: coupe benzin 22048.534 174.953055  
## 15: 31961.267 66.878570  
## 16: limousine diesel 13644.460 144.182170  
## 17: andere 37582.461 154.830556  
## 18: diesel 5076.690 87.696302  
## 19: andere diesel 4545.590 83.865398  
## 20: limousine 52270.634 106.347130  
## 21: suv 8100.394 116.529279  
## 22: suv benzin 10187.581 161.018952  
## 23: coupe 21512.372 120.224309  
## 24: bus 4122.915 82.868121  
## 25: cabrio 113710.680 114.892291  
## 26: bus lpg 3908.019 125.474383  
## 27: andere benzin 73975.337 107.286914  
## 28: limousine lpg 23217.197 164.060983  
## 29: kombi lpg 3215.888 144.016170  
## 30: limousine andere 4004.824 54.215686  
## 31: suv lpg 8320.890 211.598023  
## 32: andere 2182.677 23.806452  
## 33: cabrio diesel 13119.963 141.081687  
## 34: cabrio lpg 5411.757 145.855967  
## 35: limousine hybrid 13108.216 115.960784  
## 36: coupe lpg 5628.022 179.910256  
## 37: andere andere 48808689.682 100.977273  
## 38: kleinwagen lpg 2931.335 71.071869  
## 39: lpg 3055.327 106.801282  
## 40: bus cng 4830.621 97.370833  
## 41: kombi cng 5322.860 108.117647  
## 42: limousine elektro 37662.333 253.000000  
## 43: cng 2836.549 61.098039  
## 44: andere lpg 6534.400 157.433333  
## 45: kombi hybrid 16886.630 105.703704  
## 46: limousine cng 5320.725 150.025000  
## 47: kleinwagen cng 4381.025 89.425000  
## 48: andere cng 2653.357 97.857143  
## 49: kleinwagen andere 1477.345 38.310345  
## 50: kleinwagen elektro 9508.412 46.352941  
## 51: coupe elektro 19214.571 142.714286  
## 52: elektro 9875.000 84.750000  
## 53: kombi andere 2927.200 53.680000  
## 54: bus andere 9641.000 74.100000  
## 55: andere elektro 3948.250 6.083333  
## 56: coupe hybrid 34389.000 111.833333  
## 57: kleinwagen hybrid 10434.368 75.315789  
## 58: suv andere 5266.556 68.222222  
## 59: suv hybrid 22543.200 258.750000  
## 60: kombi elektro 6050.000 118.200000  
## 61: hybrid 4428.062 43.812500  
## 62: cabrio cng 4266.667 95.666667  
## 63: cabrio andere 4035.667 57.000000  
## 64: cabrio elektro 9180.833 50.333333  
## 65: suv cng 2599.750 48.250000  
## 66: suv elektro 179433.333 177.333333  
## 67: coupe cng 4833.333 84.333333  
## 68: coupe andere 5833.333 36.666667  
## 69: bus hybrid 7325.000 125.833333  
## 70: andere hybrid 10900.000 136.000000  
## 71: bus elektro 9800.000 60.000000  
## vehicleType fuelType Price Power

auto[,sum(price<1e4),] # Count prices higher than 10,000

## [1] 310497

auto[,mean(price<1e4),] # Proportion of prices larger than 10,000

## [1] 0.8350644

Notice the grouping by two variables with the .() syntax.

auto[,.(Price=mean(price)), by=.(vehicleType,gearbox)]

## Warning in gmean(price): Group 16 summed to more than type 'integer'  
## can hold so the result has been coerced to 'numeric' automatically, for  
## convenience.

## vehicleType gearbox Price  
## 1: manuell 8634.364  
## 2: coupe manuell 12400.354  
## 3: suv automatik 16784.747  
## 4: kleinwagen manuell 4604.674  
## 5: limousine manuell 8501.177  
## 6: cabrio manuell 14894.581  
## 7: bus manuell 6320.779  
## 8: kombi manuell 7462.864  
## 9: suv manuell 10038.929  
## 10: kleinwagen 33095.001  
## 11: 52170.035  
## 12: kleinwagen automatik 3887.873  
## 13: limousine automatik 18996.221  
## 14: kombi automatik 8908.798  
## 15: limousine 3904.094  
## 16: andere manuell 824344.396  
## 17: bus automatik 10599.586  
## 18: coupe automatik 35475.156  
## 19: cabrio automatik 16289.691  
## 20: coupe 233124.065  
## 21: kombi 3441.906  
## 22: automatik 7603.548  
## 23: cabrio 9751.342  
## 24: bus 124897.290  
## 25: andere 3661.203  
## 26: andere automatik 7366.262  
## 27: suv 7270.198  
## vehicleType gearbox Price

You may sort along one or more columns

auto[order(-price), price,] %>% head # Order along price. Descending

## [1] 2147483647 99999999 99999999 99999999 99999999 99999999

auto[order(price, -lastSeen), price,] %>% head# Order along price and last seen . Ascending and descending.

## [1] 0 0 0 0 0 0

You may group using a computed variable

auto[,.(Power=mean(powerPS)), by=.(PriceRange=price>1e4)]

## PriceRange Power  
## 1: FALSE 101.8838  
## 2: TRUE 185.9029

You may apply a function to ALL columns using a Subset of the Data using .SD

count.uniques <- function(x) length(unique(x))  
auto[,lapply(.SD, count.uniques), vehicleType]

## vehicleType dateCrawled name seller offerType price abtest  
## 1: 36714 32891 1 2 1378 2  
## 2: coupe 18745 13182 1 2 1994 2  
## 3: suv 14549 9707 1 1 1667 2  
## 4: kleinwagen 75591 49302 2 2 1927 2  
## 5: limousine 89352 58581 2 1 2986 2  
## 6: cabrio 22497 13411 1 1 2014 2  
## 7: bus 29559 19651 1 2 1784 2  
## 8: kombi 64415 41976 2 1 2529 2  
## 9: andere 3352 3185 1 1 562 2  
## yearOfRegistration gearbox powerPS model kilometer monthOfRegistration  
## 1: 101 3 374 244 13 13  
## 2: 75 3 414 117 13 13  
## 3: 73 3 342 122 13 13  
## 4: 75 3 317 163 13 13  
## 5: 83 3 506 210 13 13  
## 6: 88 3 363 95 13 13  
## 7: 65 3 251 106 13 13  
## 8: 64 3 393 177 13 13  
## 9: 81 3 230 162 13 13  
## fuelType brand notRepairedDamage dateCreated nrOfPictures postalCode  
## 1: 8 40 3 65 1 6304  
## 2: 8 35 3 51 1 5159  
## 3: 8 37 3 61 1 4932  
## 4: 8 38 3 68 1 7343  
## 5: 8 39 3 82 1 7513  
## 6: 7 38 3 70 1 5524  
## 7: 8 33 3 63 1 6112  
## 8: 8 38 3 75 1 7337  
## 9: 8 38 3 41 1 2220  
## lastSeen  
## 1: 32813  
## 2: 16568  
## 3: 13367  
## 4: 59354  
## 5: 65813  
## 6: 19125  
## 7: 26094  
## 8: 50668  
## 9: 3294

Things to note:

* .SD is the data subset after splitting along the by argument.
* Recall that lapply applies the same function to all elements of a list. In this example, to all columns of .SD.

If you want to apply a function only to a subset of columns, use the .SDcols argument

auto[,lapply(.SD, count.uniques), by=vehicleType, .SDcols=price:gearbox]

## vehicleType price abtest vehicleType yearOfRegistration gearbox  
## 1: 1378 2 1 101 3  
## 2: coupe 1994 2 1 75 3  
## 3: suv 1667 2 1 73 3  
## 4: kleinwagen 1927 2 1 75 3  
## 5: limousine 2986 2 1 83 3  
## 6: cabrio 2014 2 1 88 3  
## 7: bus 1784 2 1 65 3  
## 8: kombi 2529 2 1 64 3  
## 9: andere 562 2 1 81 3

## Make your own variables

It is very easy to compute new variables

auto[,log(price/powerPS),] %>% head # This makes no sense

## [1] Inf 4.567632 4.096387 2.995732 3.954583 1.852000

And if you want to store the result in a new variable, use the := operator

auto[,newVar:=log(price/powerPS),]

Or create multiple variables at once. The syntax c("A","B"):=.(expression1,expression2)is read "save the **list** of results from expression1 and expression2 using the **vector** of names A, and B".

auto[,c('newVar','newVar2'):=.(log(price/powerPS),price^2/powerPS),]

## Join

**data.table** can be used for joining. A *join* is the operation of aligning two (or more) data frames/tables along some index. The index can be a single variable, or a combination thereof.

Here is a simple example of aligning age and gender from two different data tables:

DT1 <- data.table(Names=c("Alice","Bob"), Age=c(29,31))  
DT2 <- data.table(Names=c("Alice","Bob","Carl"), Gender=c("F","M","M"))  
setkey(DT1, Names)  
setkey(DT2, Names)  
DT1[DT2,,]

## Names Age Gender  
## 1: Alice 29 F  
## 2: Bob 31 M  
## 3: Carl NA M

DT2[DT1,,]

## Names Gender Age  
## 1: Alice F 29  
## 2: Bob M 31

Things to note:

* A join with data.tables is performed by indexing one data.table with another. Which is the outer and which is the inner will affect the result.
* The indexing variable needs to be set using the setkey function.

There are several types of joins:

* **Inner join**: Returns the rows along the intersection of keys, i.e., rows that appear in **all** data sets.
* **Outer join**: Returns the rows along the union of keys, i.e., rows that appear in **any** of the data sets.
* **Left join**: Returns the rows along the index of the "left" data set.
* **Right join**: Returns the rows along the index of the "right" data set.

Assuming DT1 is the "left" data set, we see that DT1[DT2,,] is a right join, and DT2[DT1,,] is a left join. For an inner join use the nomath=0 argument:

DT1[DT2,,,nomatch=0]

## Names Age Gender  
## 1: Alice 29 F  
## 2: Bob 31 M

DT2[DT1,,,nomatch=0]

## Names Gender Age  
## 1: Alice F 29  
## 2: Bob M 31

## Reshaping data

Data sets (i.e. frames or tables) may arrive in a "wide" form or a "long" form. The difference is best illustrated with an example. The ChickWeight data encodes the weight of various chicks. It is "long" in that a variable encodes the time of measurement, making the data, well, simply long:

ChickWeight %>% head

## Grouped Data: weight ~ Time | Chick  
## weight Time Chick Diet  
## 1 42 0 1 1  
## 2 51 2 1 1  
## 3 59 4 1 1  
## 4 64 6 1 1  
## 5 76 8 1 1  
## 6 93 10 1 1

The mtcars data encodes 10 characteristics of 32 types of automobiles. It is "wide" since the various characteristics are encoded in different variables, making the data, well, simply wide.

mtcars %>% head

## 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

Most of *R*'s functions, with exceptions, will prefer data in the long format. There are thus various facilities to convert from one format to another. We will focus on the melt and dcast functions to convert from one format to another.

### Wide to long

melt will convert from wide to long.

dimnames(mtcars)

## [[1]]  
## [1] "Mazda RX4" "Mazda RX4 Wag" "Datsun 710"   
## [4] "Hornet 4 Drive" "Hornet Sportabout" "Valiant"   
## [7] "Duster 360" "Merc 240D" "Merc 230"   
## [10] "Merc 280" "Merc 280C" "Merc 450SE"   
## [13] "Merc 450SL" "Merc 450SLC" "Cadillac Fleetwood"   
## [16] "Lincoln Continental" "Chrysler Imperial" "Fiat 128"   
## [19] "Honda Civic" "Toyota Corolla" "Toyota Corona"   
## [22] "Dodge Challenger" "AMC Javelin" "Camaro Z28"   
## [25] "Pontiac Firebird" "Fiat X1-9" "Porsche 914-2"   
## [28] "Lotus Europa" "Ford Pantera L" "Ferrari Dino"   
## [31] "Maserati Bora" "Volvo 142E"   
##   
## [[2]]  
## [1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear"  
## [11] "carb"

mtcars$type <- rownames(mtcars)  
melt(mtcars, id.vars=c("type")) %>% head

## type variable value  
## 1 Mazda RX4 mpg 21.0  
## 2 Mazda RX4 Wag mpg 21.0  
## 3 Datsun 710 mpg 22.8  
## 4 Hornet 4 Drive mpg 21.4  
## 5 Hornet Sportabout mpg 18.7  
## 6 Valiant mpg 18.1

Things to note:

* The car type was originally encoded in the rows' names, and not as a variable. We thus created an explicit variable with the cars' type using the rownames function.
* The id.vars of the melt function names the variables that will be used as identifiers. All other variables are assumed to be measurements. These can have been specified using their index instead of their name.
* If not all variables are measurements, we could have names measurement variables explicitly using the measure.vars argument of the melt function. These can have been specified using their index instead of their name.
* By default, the molten columns are automatically named variable and value.

We can replace the automatic namings using variable.name and value.name:

melt(mtcars, id.vars=c("type"), variable.name="Charachteristic", value.name="Measurement") %>% head

## type Charachteristic Measurement  
## 1 Mazda RX4 mpg 21.0  
## 2 Mazda RX4 Wag mpg 21.0  
## 3 Datsun 710 mpg 22.8  
## 4 Hornet 4 Drive mpg 21.4  
## 5 Hornet Sportabout mpg 18.7  
## 6 Valiant mpg 18.1

### Long to wide

dcast will conver from long to wide:

dcast(ChickWeight, Chick~Time, value.var="weight")

## Chick 0 2 4 6 8 10 12 14 16 18 20 21  
## 1 18 39 35 NA NA NA NA NA NA NA NA NA NA  
## 2 16 41 45 49 51 57 51 54 NA NA NA NA NA  
## 3 15 41 49 56 64 68 68 67 68 NA NA NA NA  
## 4 13 41 48 53 60 65 67 71 70 71 81 91 96  
## 5 9 42 51 59 68 85 96 90 92 93 100 100 98  
## 6 20 41 47 54 58 65 73 77 89 98 107 115 117  
## 7 10 41 44 52 63 74 81 89 96 101 112 120 124  
## 8 8 42 50 61 71 84 93 110 116 126 134 125 NA  
## 9 17 42 51 61 72 83 89 98 103 113 123 133 142  
## 10 19 43 48 55 62 65 71 82 88 106 120 144 157  
## 11 4 42 49 56 67 74 87 102 108 136 154 160 157  
## 12 6 41 49 59 74 97 124 141 148 155 160 160 157  
## 13 11 43 51 63 84 112 139 168 177 182 184 181 175  
## 14 3 43 39 55 67 84 99 115 138 163 187 198 202  
## 15 1 42 51 59 64 76 93 106 125 149 171 199 205  
## 16 12 41 49 56 62 72 88 119 135 162 185 195 205  
## 17 2 40 49 58 72 84 103 122 138 162 187 209 215  
## 18 5 41 42 48 60 79 106 141 164 197 199 220 223  
## 19 14 41 49 62 79 101 128 164 192 227 248 259 266  
## 20 7 41 49 57 71 89 112 146 174 218 250 288 305  
## 21 24 42 52 58 74 66 68 70 71 72 72 76 74  
## 22 30 42 48 59 72 85 98 115 122 143 151 157 150  
## 23 22 41 55 64 77 90 95 108 111 131 148 164 167  
## 24 23 43 52 61 73 90 103 127 135 145 163 170 175  
## 25 27 39 46 58 73 87 100 115 123 144 163 185 192  
## 26 28 39 46 58 73 92 114 145 156 184 207 212 233  
## 27 26 42 48 57 74 93 114 136 147 169 205 236 251  
## 28 25 40 49 62 78 102 124 146 164 197 231 259 265  
## 29 29 39 48 59 74 87 106 134 150 187 230 279 309  
## 30 21 40 50 62 86 125 163 217 240 275 307 318 331  
## 31 33 39 50 63 77 96 111 137 144 151 146 156 147  
## 32 37 41 48 56 68 80 83 103 112 135 157 169 178  
## 33 36 39 48 61 76 98 116 145 166 198 227 225 220  
## 34 31 42 53 62 73 85 102 123 138 170 204 235 256  
## 35 39 42 50 61 78 89 109 130 146 170 214 250 272  
## 36 38 41 49 61 74 98 109 128 154 192 232 280 290  
## 37 32 41 49 65 82 107 129 159 179 221 263 291 305  
## 38 40 41 55 66 79 101 120 154 182 215 262 295 321  
## 39 34 41 49 63 85 107 134 164 186 235 294 327 341  
## 40 35 41 53 64 87 123 158 201 238 287 332 361 373  
## 41 44 42 51 65 86 103 118 127 138 145 146 NA NA  
## 42 45 41 50 61 78 98 117 135 141 147 174 197 196  
## 43 43 42 55 69 96 131 157 184 188 197 198 199 200  
## 44 41 42 51 66 85 103 124 155 153 175 184 199 204  
## 45 47 41 53 66 79 100 123 148 157 168 185 210 205  
## 46 49 40 53 64 85 108 128 152 166 184 203 233 237  
## 47 46 40 52 62 82 101 120 144 156 173 210 231 238  
## 48 50 41 54 67 84 105 122 155 175 205 234 264 264  
## 49 42 42 49 63 84 103 126 160 174 204 234 269 281  
## 50 48 39 50 62 80 104 125 154 170 222 261 303 322

Things to note:

* dcast uses a formula interface (~) to specify the row identifier and the variables. The LHS is the row identifier, and the RHS for the variables to be created.
* The measurement of each LHS at each RHS, is specified using the value.var argument.

## Bibliographic Notes

data.table has excellent online documentation. See [here](https://cran.r-project.org/web/packages/data.table/vignettes/datatable-intro.html). See [here](https://rstudio-pubs-static.s3.amazonaws.com/52230_5ae0d25125b544caab32f75f0360e775.html) for joining data.tables. See [here](https://cran.r-project.org/web/packages/data.table/vignettes/datatable-reshape.html) for more on reshaping data.tables. See [here](https://www.r-bloggers.com/intro-to-the-data-table-package/) for a comparison of the data.frame way, versus the data.table way.

## Practice Yourself

# Bibliography

Analytics, Revolution, and Steve Weston. 2015. *Foreach: Provides Foreach Looping Construct for R*. <https://CRAN.R-project.org/package=foreach>.

Anderson-Cook, Christine M. 2004. “An Introduction to Multivariate Statistical Analysis.” *Journal of the American Statistical Association* 99 (467). American Statistical Association: 907–9.

Arlot, Sylvain, Alain Celisse, and others. 2010. “A Survey of Cross-Validation Procedures for Model Selection.” *Statistics Surveys* 4. The author, under a Creative Commons Attribution License: 40–79.

Bai, Zhidong, and Hewa Saranadasa. 1996. “Effect of High Dimension: By an Example of a Two Sample Problem.” *Statistica Sinica*. JSTOR, 311–29.

Bates, Douglas, Martin Mächler, Ben Bolker, and Steve Walker. 2015. “Fitting Linear Mixed-Effects Models Using lme4.” *Journal of Statistical Software* 67 (1): 1–48. doi:[10.18637/jss.v067.i01](https://doi.org/10.18637/jss.v067.i01).

Chang, Winston, Joe Cheng, JJ Allaire, Yihui Xie, and Jonathan McPherson. 2017. *Shiny: Web Application Framework for R*. <https://CRAN.R-project.org/package=shiny>.

Chapple, Simon R, Eilidh Troup, Thorsten Forster, and Terence Sloan. 2016. *Mastering Parallel Programming with R*. Packt Publishing Ltd.

Conway, Drew, and John White. 2012. *Machine Learning for Hackers*. “ O’Reilly Media, Inc.”

Davis, Timothy A. 2006. *Direct Methods for Sparse Linear Systems*. SIAM.

Dowle, Matt, and Arun Srinivasan. 2017. *Data.table: Extension of ‘Data.frame‘*. <https://CRAN.R-project.org/package=data.table>.

Efron, Bradley. 2012. *Large-Scale Inference: Empirical Bayes Methods for Estimation, Testing, and Prediction*. Vol. 1. Cambridge University Press.

Everitt, Brian, and Torsten Hothorn. 2011. *An Introduction to Applied Multivariate Analysis with R*. Springer Science & Business Media.

Fithian, William. 2015. “Topics in Adaptive Inference.” PhD thesis, STANFORD UNIVERSITY.

Foster, Dean P, and Robert A Stine. 2004. “Variable Selection in Data Mining: Building a Predictive Model for Bankruptcy.” *Journal of the American Statistical Association* 99 (466). Taylor & Francis: 303–13.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. 2001. *The Elements of Statistical Learning*. Vol. 1. Springer series in statistics Springer, Berlin.

Gentle, James E. 2012. *Numerical Linear Algebra for Applications in Statistics*. Springer Science & Business Media.

Gilbert, John R, Cleve Moler, and Robert Schreiber. 1992. “Sparse Matrices in Matlab: Design and Implementation.” *SIAM Journal on Matrix Analysis and Applications* 13 (1). SIAM: 333–56.

Goeman, Jelle J, Sara A Van De Geer, and Hans C Van Houwelingen. 2006. “Testing Against a High Dimensional Alternative.” *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68 (3). Wiley Online Library: 477–93.

Golub, Gene H, and Charles F Van Loan. 2012. *Matrix Computations*. Vol. 3. JHU Press.

Graham, RL. 1988. “Isometric Embeddings of Graphs.” *Selected Topics in Graph Theory* 3. Academic Press San Diego, CA: 133–50.

Greene, William H. 2003. *Econometric Analysis*. Pearson Education India.

Hotelling, Harold. 1933. “Analysis of a Complex of Statistical Variables into Principal Components.” *Journal of Educational Psychology* 24 (6). Warwick & York: 417.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An Introduction to Statistical Learning*. Vol. 6. Springer.

Javanmard, Adel, and Andrea Montanari. 2014. “Confidence Intervals and Hypothesis Testing for High-Dimensional Regression.” *Journal of Machine Learning Research* 15 (1): 2869–2909.

Kalisch, Markus, and Peter Bühlmann. 2014. “Causal Structure Learning and Inference: A Selective Review.” *Quality Technology & Quantitative Management* 11 (1). Taylor & Francis: 3–21.

Kane, Michael J, John Emerson, Stephen Weston, and others. 2013. “Scalable Strategies for Computing with Massive Data.” *Journal of Statistical Software* 55 (14): 1–19.

Lantz, Brett. 2013. *Machine Learning with R*. Packt Publishing Ltd.

Leisch, Friedrich. 2002. “Sweave: Dynamic Generation of Statistical Reports Using Literate Data Analysis.” In *Compstat*, 575–80. Springer.

Leskovec, Jure, Anand Rajaraman, and Jeffrey David Ullman. 2014. *Mining of Massive Datasets*. Cambridge University Press.

McCullagh, Peter. 1984. “Generalized Linear Models.” *European Journal of Operational Research* 16 (3). Elsevier: 285–92.

Mohri, Mehryar, Afshin Rostamizadeh, and Ameet Talwalkar. 2012. *Foundations of Machine Learning*. MIT press.

Nadler, Boaz. 2008. “Finite Sample Approximation Results for Principal Component Analysis: A Matrix Perturbation Approach.” *The Annals of Statistics*. JSTOR, 2791–2817.

Pearson, Karl. 1901. “LIII. on Lines and Planes of Closest Fit to Systems of Points in Space.” *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2 (11). Taylor & Francis: 559–72.

R Core Team. 2016. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. <https://www.R-project.org/>.

Ripley, Brian D. 2007. *Pattern Recognition and Neural Networks*. Cambridge university press.

Rosenblatt, Jonathan. 2013. “A Practitioner’s Guide to Multiple Testing Error Rates.” *arXiv Preprint arXiv:1304.4920*.

Rosenblatt, Jonathan D, and Yoav Benjamini. 2014. “Selective Correlations; Not Voodoo.” *NeuroImage* 103. Elsevier: 401–10.

Sammut, Claude, and Geoffrey I Webb. 2011. *Encyclopedia of Machine Learning*. Springer Science & Business Media.

Sarkar, Deepayan. 2008. *Lattice: Multivariate Data Visualization with R*. New York: Springer. <http://lmdvr.r-forge.r-project.org>.

Schmidberger, Markus, Martin Morgan, Dirk Eddelbuettel, Hao Yu, Luke Tierney, and Ulrich Mansmann. 2009. “State of the Art in Parallel Computing with R.” *Journal of Statistical Software* 47 (1).

Searle, Shayle R, George Casella, and Charles E McCulloch. 2009. *Variance Components*. Vol. 391. John Wiley & Sons.

Shah, Viral, and John R Gilbert. 2004. “Sparse Matrices in Matlab\* P: Design and Implementation.” In *International Conference on High-Performance Computing*, 144–55. Springer.

Shalev-Shwartz, Shai, and Shai Ben-David. 2014. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge university press.

Sievert, Carson, Chris Parmer, Toby Hocking, Scott Chamberlain, Karthik Ram, Marianne Corvellec, and Pedro Despouy. 2016. *Plotly: Create Interactive Web Graphics via ’Plotly.js’*. <https://CRAN.R-project.org/package=plotly>.

Small, Christopher G. 1990. “A Survey of Multidimensional Medians.” *International Statistical Review/Revue Internationale de Statistique*. JSTOR, 263–77.

Tukey, John W. 1977. *Exploratory Data Analysis*. Reading, Mass.

Vapnik, Vladimir. 2013. *The Nature of Statistical Learning Theory*. Springer science & business media.

Venables, William N, and Brian D Ripley. 2013. *Modern Applied Statistics with S-Plus*. Springer Science & Business Media.

Venables, William N, David M Smith, R Development Core Team, and others. 2004. “An Introduction to R.” Network Theory Limited.

Wang, Chun, Ming-Hui Chen, Elizabeth Schifano, Jing Wu, and Jun Yan. 2015. “Statistical Methods and Computing for Big Data.” *arXiv Preprint arXiv:1502.07989*.

Weihs, Claus, Olaf Mersmann, and Uwe Ligges. 2013. *Foundations of Statistical Algorithms: With References to R Packages*. CRC Press.

Weiss, Robert E. 2005. *Modeling Longitudinal Data*. Springer Science & Business Media.

Wickham, Hadley. 2009. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. <http://ggplot2.org>.

———. 2014. *Advanced R*. CRC Press.

Wickham, Hadley, and Romain Francois. 2016. *Dplyr: A Grammar of Data Manipulation*. <https://CRAN.R-project.org/package=dplyr>.

Wickham, Hadley, Jim Hester, and Romain Francois. 2016. *Readr: Read Tabular Data*. <https://CRAN.R-project.org/package=readr>.

Wilcox, Rand R. 2011. *Introduction to Robust Estimation and Hypothesis Testing*. Academic Press.

Wilkinson, GN, and CE Rogers. 1973. “Symbolic Description of Factorial Models for Analysis of Variance.” *Applied Statistics*. JSTOR, 392–99.

Wilkinson, Leland. 2006. *The Grammar of Graphics*. Springer Science & Business Media.

Xie, Yihui. 2015. *Dynamic Documents with R and Knitr*. Vol. 29. CRC Press.

———. 2016. *Bookdown: Authoring Books and Technical Documents with R Markdown*. CRC Press.

1. S and S-Plus used to save objects on disk. Working from RAM has advantages and disadvantages. More on this in Chapter 15. [↑](#footnote-ref-57)
2. Taken from <http://cran.r-project.org/web/packages/magrittr/vignettes/magrittr.html> [↑](#footnote-ref-63)
3. R uses a [**three** valued logic](https://en.wikipedia.org/wiki/Three-valued_logic) where a missing value (NA) is neither TRUE, nor FALSE. [↑](#footnote-ref-78)
4. This is a classical *functional programming* paradigm. If you want an object oriented flavor of R programming, see Hadley's [Advanced R book](http://adv-r.had.co.nz/OO-essentials.html). [↑](#footnote-ref-96)
5. More formally, this is called [Lexical Scoping](https://darrenjw.wordpress.com/2011/11/23/lexical-scope-and-function-closures-in-r/). [↑](#footnote-ref-98)
6. The "response" is also know as the "dependent" variable in the statistical literature, or the "labels" in the machine learning literature. [↑](#footnote-ref-150)
7. The "factors" are also known as the "independent variable", or "the design", in the statistical literature, and the "features", or "attributes" in the machine learning literature. [↑](#footnote-ref-151)
8. The "error term" is also known as the "noise", or the "common causes of variability". [↑](#footnote-ref-152)
9. You may philosophize if the measurement error is a mere instance of unmodeled factors or not, but this has no real implication for our purposes. [↑](#footnote-ref-153)
10. By "computed" we mean what statisticians call "fitted", or "estimated", and computer scientists call "learned". [↑](#footnote-ref-155)
11. Sometimes known as the Root Mean Squared Error (RMSE). [↑](#footnote-ref-159)
12. The example is taken from <http://rtutorialseries.blogspot.co.il/2011/02/r-tutorial-series-two-way-anova-with.html> [↑](#footnote-ref-168)
13. Do not confuse *generalized linear models* with [*non-linear regression*](https://en.wikipedia.org/wiki/Nonlinear_regression), or [*generalized least squares*](https://en.wikipedia.org/wiki/Generalized_least_squares). These are different things, that we do not discuss. [↑](#footnote-ref-176)
14. Taken from <http://www.theanalysisfactor.com/generalized-linear-models-in-r-part-6-poisson-regression-count-variables/> [↑](#footnote-ref-184)
15. If you are unfamiliar with design of experiments, have a look at Chapter 6 of my Quality Engineering [class notes](https://github.com/johnros/qualityEngineering/blob/master/Class_notes/notes.pdf). [↑](#footnote-ref-191)
16. A.k.a. the *cluster effect* in the epidemiological literature. [↑](#footnote-ref-197)
17. My thanks to Efrat Vilneski for the figure. [↑](#footnote-ref-212)
18. This vocabulary is not standard in the literature, so when you read a text, you need to verify yourself what the author means. [↑](#footnote-ref-215)
19. You might find this shocking, but it does mean that you cannot trust the summary table of a model that was selected from a multitude of models. [↑](#footnote-ref-224)
20. It is even a subset of the Hilbert space, itself a subset of the space of all functions. [↑](#footnote-ref-238)
21. Example taken from <https://lagunita.stanford.edu/c4x/HumanitiesScience/StatLearning/asset/ch6.html> [↑](#footnote-ref-245)
22. <http://en.wikipedia.org/wiki/Principal_component_analysis> [↑](#footnote-ref-259)
23. You are probably used to thinking of the **dimension** of linear spaces. We will not rigorously define what is the dimension of a manifold, but you may think of it as the number of free coordinates needed to navigate along the manifold. [↑](#footnote-ref-262)
24. <https://en.wikipedia.org/wiki/G_factor_(psychometrics)> [↑](#footnote-ref-266)
25. The term Graph is typically used in this context instead of Network. But a graph allows only yes/no relations, while a network, which is a weighted graph, allows a continuous measure of similarity (or dissimilarity). *Network* is thus more appropriate than *graph*. [↑](#footnote-ref-271)
26. Then again, it is possible that the true distances are the white matter fibers connecting going within the cortex, in which case, Euclidean distances are more appropriate than geodesic distances. We put that aside for now. [↑](#footnote-ref-273)
27. Recall, S was the original software from which R evolved. [↑](#footnote-ref-387)
28. The code was contributed by Liad Shekel. [↑](#footnote-ref-457)
29. This is slowly changing. Indeed, Microsoft's SQL Server 2016 is already providing [in-database-analytics](https://blogs.technet.microsoft.com/dataplatforminsider/2016/03/29/in-database-advanced-analytics-with-r-in-sql-server-2016/), and other will surely follow. [↑](#footnote-ref-459)