

Up and Running with R

G.Janacek

UEA

May 12, 2009

Unlock the Secrets of R



G. Janacek

Introduction

There are three main kinds of statistics program available at UEA. They are

- SPSS - a program supported by the ITSC. It was designed to support the kinds of data analysis that you might need in the social sciences. It is fine for routine analysis.
- Stata - a complete, integrated statistical package which is popular amongst medics.
- \mathbb{R} which is (almost) a free clone of Splus, a program which implements the S language.

In what follows we aim to get you up and running with \mathbb{R} . *We will not be teaching you statistics or the details of Lisp*, our aim is to get you some way along the learning curve so that you can start to use \mathbb{R} for your own analysis.

In what follows we aim to get you up and running with \mathbb{R} . *We will not be teaching you statistics or the details of Lisp*, our aim is to get you some way along the learning curve so that you can start to use \mathbb{R} for your own analysis.

R is used by statistics departments and researchers around the world because it is an open source implementation of Splus. It allows the user to be flexible in their analysis and is much more up to date than SPSS. It has extensive and powerful graphics abilities, that are tightly linked with its analytic abilities. R is also developing rapidly and new features and abilities appear every few months. It is available on most machines at UEA and you can download your own copy to run on a personal machine.

R is free and I would urge you to get your own copy.

Binary versions and the source code (if you are very brave) for \mathbb{R} are available from the CRAN website. You will also find comprehensive documentation, Web Pages and Email Lists. Go to

<http://cran.r-project.org>

Details of the \mathbb{R} -help list, and of other lists that serve the \mathbb{R} community, are available from the web site for the \mathbb{R} project at

<http://www.R-project.org/>

Sadly there is no support for \mathbb{R} at UEA, which is why I am writing these notes. There is a *wiki* at

<http://cmpwiki/wiki/Pearson>

but I have only just set it up so the content is thin. I have asked ITSC to set up a mailing list, probably called Capital.R. You will be able to subscribe to the list and exchange problems and ideas with other subscribers.

Large Data Sets

One important point you should bear in mind is that \mathbb{R} was not designed for very large data sets. The consensus is that on the same modern system SAS is usually better able to handle large, dumb calculations than S-PLUS, which is (generally) better than \mathbb{R} . \mathbb{R} has a hard limit of 2 GB total memory, as I understand, and its data model requires holding an entire set in memory. This is very fast until it isn't. This limit applies even on 64 bit systems. Horses for courses!

Getting Up and Running

\mathbb{R} is a functional language, that is there is a language core that uses standard forms of algebraic notation, allowing the calculations such as $2 + 3$, or 3^{11} . Beyond this, most computation is handled using functions. Rather than go on about structure and syntax we will start to use \mathbb{R} .

Startup and Shut down

Start up \mathbb{R} , either by clicking on the ikon (PC and Mac) or typing R on a UNIX box, and you will get the welcome screen. What you will have loaded is the *base* package. You can load other packages that come with your installation or from elsewhere, see the header of the console window but for the moment we work with the base package.

Start up \mathbb{R} , either by clicking on the ikon (PC and Mac) or typing R on a UNIX box, and you will get the welcome screen. What you will have loaded is the *base* package. You can load other packages that come with your installation or from elsewhere, see the header of the console window but for the moment we work with the base package.

The action of quitting from an \mathbb{R} session uses the function call `q()`.

When you use \mathbb{R} you create a workspace in which all the objects you have created are stored. When you try and quit you will be asked if you wish to save your (workspace) session.

If you use a PC then you may be able to quit via the pull down menu on your screen.

Manipulating data

Most of the data sets in data are structured in some way so, for the moment, we will try something simpler to get us going. \mathbb{R} has lots of ways of simulating data from given distributions, for example

```
rnorm(n, mean=0, sd=1)
```

This command gives a vector of length `n` whose elements are random and have a Normal distribution mean zero and standard deviation 1. If you do not specify the `mean` and `sd` the values default to 0 and 1. So to create two `x` and `y` vectors of length 100

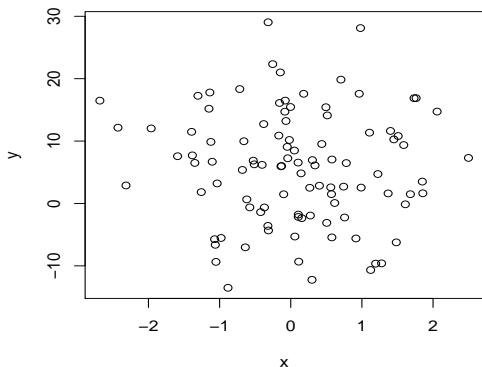
```
> x<-rnorm(100)
```

```
> y<-rnorm(100,5,9)
```

if you type `x` you can examine the numbers in `x`. If 100 is too many numbers try `x[20:50]` which gives a slice of the vector `x`. You can use the indices in many useful ways, eg `x[100:87]`

Note we use a left pointing arrow as the assignment operator. For versions of R after 1.4 you can, if you prefer, use `=`. I prefer to use `←` for compatibility.

To check the random numbers we could plot `x` and `y`. Try `?plot` or `help(plot)` This will explain the plotting command. You could try `plot(x,y,pch="+")` or `plot(x,y,pch="a")`



The obvious data summaries are the mean, median, standard deviation etc. try some of these functions.

```
hist(x) # computes and plots a histogram of x
mean(x) # computes the mean of the vector x
sd(x) # computes the standard deviation of the vector x
median(x) # computes the median of the vector x
summary(x) # summarises the vector x
print(x) # prints the object x
cat() # Prints multiple objects, one after the other
length() # Number of elements in a vector or of a list
unique() # Gives the vector of distinct values
diff() # Replace a vector by the vector of first differences
#N. B. diff(x) has one less element than x
sort() # Sort elements into order, but omitting NAs
order() # x[order(x)] orders elements of x, with NAs last
cumsum(x) # computes the cumulative sum of the vector x
cumprod(x) # computes the cumulative product of the vector x
rev() # reverse the order of vector elements
# is - as you have probably guessed - a comment symbol. Everyt
```

You are probably beginning to see a problem in using \mathbb{R} , *you have to know the name* of the function you would like to use. While there are some prototype GUI interfaces you will have to resign yourself to the UNIX command driven world. The appendices contain copies of some of the CRAN crib sheets but in addition the help system can be useful.

The `apropos()` command is convenient when you are not sure that you know the name of a function, for example if you were after a `stem` and `leaf` function but were not sure if the name was `stem` or `stemandleaf`. For example

```
> apropos(stem)
[1] "stem"      "system"    "system.file" "system.time"
```

The help command will help us find help on the given function or data-set *once we know the name*. For example `help(stem)` or the abbreviated `?stem` will display the documentation on the stem function. For example `stem` gives

stem {graphics} R Documentation

Stem-and-Leaf Plots

Description

stem produces a stem-and-leaf plot of the values in x.

The parameter scale can be used to expand the scale of the plot.

A value of scale=2 will cause the plot to be roughly twice as

Usage

```
stem(x, scale = 1, width = 80, atom = 1e-08)
```

Arguments

x a numeric vector.

scale This controls the plot length.

width The desired width of plot.

atom a tolerance.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988)

The New S Language. Wadsworth & Brooks/Cole.

Examples

```
stem(islands)
```

```
stem(log10(islands))
```

More help is available via a web browser, the command is `help.start()`. See figure 1.

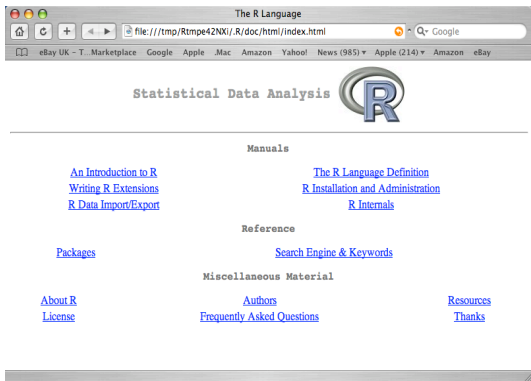


Figure: Help screen

You can follow the hyperlinks to the packages or use the search. In fact Google also works very well.

Vectors

Everything in R is a vector (but some have only one element) . As we have seen we can use `[]` to extract subsets. So

```
> p<-1:10 # gives a list
> p
 [1]  1  2  3  4  5  6  7  8  9 10
> p[4:9]
 [1] 4 5 6 7 8 9
> p[4:1]
 [1] 4 3 2 1
```

We can bind vectors together to get matrices

```
m1<-cbind(p,q)
```

```
> m1
```

	p	q
[1,]	1	-10
[2,]	2	-9
[3,]	3	-8
[4,]	4	-7
[5,]	5	-6
[6,]	6	-5
[7,]	7	-4
[8,]	8	-3
[9,]	9	-2
[10,]	10	-1

```
m2<-rbind(p,q)
> m2
  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
p    1    2    3    4    5    6    7    8    9   10
q  -10   -9   -8   -7   -6   -5   -4   -3   -2   -1
```

All of matrix algebra is available

```
> m1*2
```

```
      p  q
[1,]  2 -20
[2,]  4 -18
[3,]  6 -16
[4,]  8 -14
[5,] 10 -12
[6,] 12 -10
[7,] 14  -8
[8,] 16  -6
[9,] 18  -4
[10,] 20  -2
```

```
> m1-4
```

```
      p  q  
[1,] -3 -14  
[2,] -2 -13  
[3,] -1 -12  
[4,]  0 -11  
[5,]  1 -10  
[6,]  2  -9  
[7,]  3  -8  
[8,]  4  -7  
[9,]  5  -6  
[10,] 6  -5
```

```

> m2%*%m1
      p      q
p  385 -220
q -220  385
> m3<-m2%*%m1 ##### notice the percentage signs for mat
> solve(m3) ##### solve gives the inverse in this case
#####- it can be used to solve systems

      p      q
p 0.003856749 0.002203857
q 0.002203857 0.003856749

```

```
> m4<-solve(m3)
```

```
> m4%*%m3
```

```
           p           q
p  1.000000e+00 6.678685e-17
q -1.301043e-16 1.000000e+00
```

```
> eigen(m3)
```

```
$values
```

```
[1] 605 165
```

```
$vectors
```

```
           [,1]      [,2]
[1,]  0.7071068  0.7071068
[2,] -0.7071068  0.7071068
```

You can coerce vectors to matrices, for example

```
> p<-1:24
```

```
> p2<-p
```

```
> dim(p)<-c(4,6)
```

```
> p
```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]
[1,]	1	5	9	13	17	21
[2,]	2	6	10	14	18	22
[3,]	3	7	11	15	19	23
[4,]	4	8	12	16	20	24

```
> dim(p2) <- c(6,4)
> p2
```

	[,1]	[,2]	[,3]	[,4]
[1,]	1	7	13	19
[2,]	2	8	14	20
[3,]	3	9	15	21
[4,]	4	10	16	22
[5,]	5	11	17	23
[6,]	6	12	18	24

data input

Getting your own data into \mathbb{R} is reasonably easy. For a small amount of data try using `c`

```
mydata<-c(12,34,23,39,47)
```

This function combines, or concatenates terms together. As an example, suppose we have the following count of the number of typos per page of these notes:

```
2 3 0 3 1 0 0 1
```

To enter this into an \mathbb{R} session we do so with

```
typos<-c(2,3,0,3,1,0,0,1)
```

```
typos
```

```
[1] 2 3 0 3 1 0 0 1
```

We can also concatenate vectors

```
> a<-c(1,2,3)
> b<-c(4,3,2,1,0)
> c(a,b)
[1] 1 2 3 4 3 2 1 0
> d<-c(a,b)
> d
[1] 1 2 3 4 3 2 1 0
```

There are several other options

```
> w<--8:2
```

```
> w
```

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

```
> w<-1:3
```

```
> w
```

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

```
> t<-rep(w,each=3,length.out=12)
```

```
> t
```

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

```
> t<-rep(w,each=4,length.out=12)
```

```
> t
```

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

Missing values

If you are unfortunate enough to have a missing value you can put a NA in its place. R will understand that it denotes a missing value.

scan

if you have more data then you might prefer (as I do) to type in data from the keyboard, for example

```
mydata <-scan()
```

```
12
```

```
34
```

```
23 39 47
```

A blank line tells `scan` that input has ended. `?scan` will tell you how to read from a file.

I like to cut data and past into \mathbb{R} using `scan`. This can be a little tricky at times since cut and past from different applications may behave oddly - or not at all. If all else fails paste into a word processor and then to `.` This can be fairly convenient when entering in a few data points (10-40 say), but you might want to use a data file if you have more. You might, as mentioned above, cut and paste from a file to \mathbb{R} or you can use the `scan` options

If we have our numbers stored in a text file, then `scan` can be used to read them. You just need to tell `scan` to open the file and read from it `.`

Here are two examples

- Suppose the file `ReadWithScan.txt` has contents 1 2 3 4 then the command

```
> x <- scan(file = "ReadWithScan.txt")
```

will read the contents into your R session, putting the contents of the file into `x`.

- If you had some formatting between the numbers you want to get rid of, say your file `ReadWithScan.txt` is

```
1,2,3, 4
```

then

```
> x<-scan(file = "ReadWithScan.txt",sep=",")
```

is what you need.

- If you are like me and forget where the file is

```
> x <- scan(file =file.choose())
```

will open a dialogue box to help you find your file.

Data tables - the dataframe

If you want to enter multivariate sets of data, you can do any of the above for each variable. However, it may be more convenient to read in tables of data at once. \mathbb{R} has many built in data sets, you can see what they are by typing `data()`. If we choose the `women` data set we load it using `data(women)` and look at it

```
> women
  height weight
1     58   115
2     59   117
3     60   120
4     61   123
5     62   126
6     63   129
7     64   132
8     65   135
9     66   139
10    67   142
11    68   146
12    69   150
13    70   154
14    71   159
15    72   164
```

This kind of structured data is called a `data.frame`. You can read files of this type into `R` but as with the built in data frames they are not immediately available. To the height or weight data you have several possibilities

- 1 Use `women$height` or `women$weight`
- 2 Use `women[1]` or `women[2]`
- 3 Perhaps the simplest approach is to `attach` the file. If you type `attach(women)` `R` will realize you want to use the names within the table and you can just refer to `height` or `weight`. The reverse is of course `detach(women)`

If you want to enter multivariate sets of data, you can do any of the above for each variable. However, it may be more convenient to read in tables of data at once. Suppose your data is in tabular form such as this file `ReadWithReadTable.txt`.

```
gender weeks weight
  m      40    2968
  m      38    2795
  m      40    3163
  m      35    2925
  m      36    2625
```

Notice the first row supplies column names, the second and following rows the data. The command `read.table` will read this in and store the results in a *data frame*.

A data frame is a special matrix where all the variables are stored as columns and each has the same length. Notice we need to specify that the headers are there in this case so we write `header=T` otherwise it is assumed that the columns do not have names, hence

```
> babies <-read.table(file="ReadWithReadTable.txt",header=T)
```

Because can hold many such dataframes **the variable headings in each table are initially hidden from the main program.** You can refer to them is several ways.

in case I forget the dimensions

```
> dim(babies)
[1] 24  3
```

now we have

```
> babies[,2] # the second column
 [1] 40 38 40 35 36 37 41 40 37 38 40 38 40 36 40 38 42 39 40
[23] 39 40
> babies[2,] # the second row
  gender weeks weight
2      m     38  2795
```

Alternatively

```
> babies[['gender']] # a factor, it prints the levels [1]
[1] m m m m m m m m m m m m m f f f f f f f f f f f f f
```

```
Levels: f m
```

```
> babies[['weight']] # a numeric vector
```

```
[1] 2968 2795 3163 2925 2625 2847 3292 3473 2628 3176 3421 29
```

Personally I would refer to the variables in a table using \$ as follows

```
> babies<-read.table(file="ReadWithReadTable.txt",header=T)
> babies$gender # a factor, it prints the levels [1]
> babies$gender
[1] m m m m m m m m m m m m f f f f f f f f f f f f
Levels: f m

> babies$weeks # a numeric vector
[1] 40 38 40 35 36 37 41 40 37 38 40 38 40 36 40 38 42 39 40
```

```
> x # default print out for a data.frame
```

```
  gender weeks weight
1      m    40  2968
2      m    38  2795
3      m    40  3163
4      m    35  2925
5      m    36  2625
6      m    37  2847
7      m    41  3292
8      m    40  3473
9      m    37  2628
10     m    38  3176
11     m    40  3421
12     m    38  2975
13     f    40  3317
```

```
.....
.....
```

`read.table` treats the variables as numeric or as factors. A *factor* is special class to R and has a special print method. You can think of a factor as a vector of *categories*. For example

```
(red,blue,blue,red)
```

or

```
(1,2,1,2,2,2,1)
```

The "levels" of the factor are displayed after the values are printed. Do note that it is common to read data as a vector and then to use a factor vector to indicate which class, type etc of observation.

The command `attach`, for example `attach(babies)` makes the column headers in the table available to `.`

The reverse being `detach`.

Be careful because an already existing variable **will not be overwritten..**

You should get an error message but it is easily missed.

Note: Some commands allow you to specify the data frame you are referring to.

Excel files

If you have Excel files you can always save them as csv files. Then they can be read into R using

```
x<-read.csv(file="fred".header=T).
```

The documentation says (I abbreviate):

The most common —R data import/export question seems to be ‘how do I read an Excel spreadsheet’. *The first piece of advice is to avoid doing so if possible!* If you have access to Excel, export the data you want from Excel in tab-delimited or comma-separated form, and use `read.delim` or `read.csv` to import it into `.` (You may need to use `read.delim2` or `read.csv2` in a continental European locale that uses comma as the decimal point.). Exporting a DIF file and reading it using `read.DIF` is another possibility.

Note that an Excel `.xls` is not just a spreadsheet: such files can contain many sheets, and the sheets can contain formulae, macros and so on. Not all readers can read other than the first sheet, and may be confused by other contents of the file.

- Windows users can use `odbcConnectExcel` in package `RODBC`. This can select rows and columns from any of the sheets in an Excel spreadsheet file.
- Also (only for Windows) the package `xlsReadWrite` has a function `read.xls` to read `.xls` files

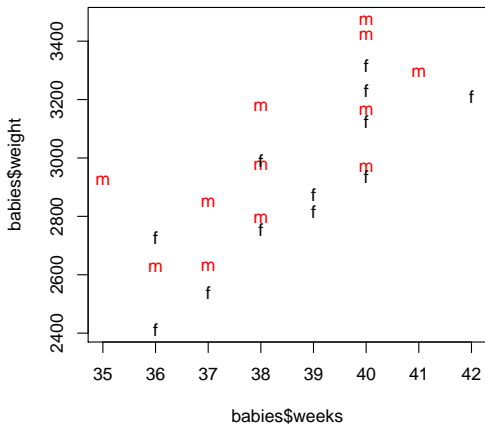
Perl users have contributed a module `OLE::SpreadSheet::ParseExcel` and a program `xls2csv.pl` to convert Excel spreadsheets to CSV files. Package `gdata` provides a basic wrapper in its `read.xls` function. The package `Foreign` has several useful functions for reading files from packages like `STATA` or `SPSS`- as we shall see.

Graphics

There are two graphic systems in R the traditional system and the grid system. For the moment we will look at the traditional one and return to the grid if we have time. As we have seen the `plot` command gives us basic plots. For the birth weight data we have

```
> babies<-read.table(file.choose(),header=T)
> names(babies)# I always forget what is in the tables so this
[1] "gender" "weeks" "weight"
> plot(babies$weeks,babies$weight,pch=as.character(babies$gender))
```

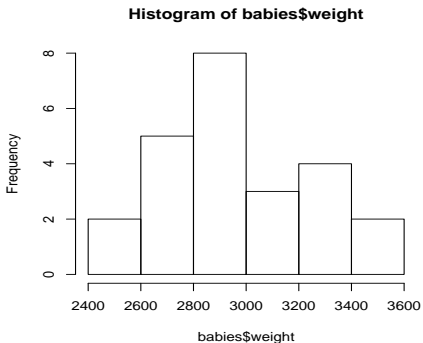
Here I have used the `pch` command to give different symbols for males and females. While I am at it I have use different colours. I have to use the `babies$gender` vector but `pch` expects characters. Hence we coerce `babies$gender` to characters. In the same way `col` expects integers so we coerce to integers.



You can add line and points to an already existing graph using `lines` or `points`.

We can look at histograms

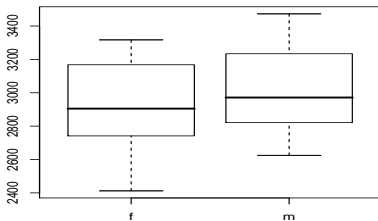
```
> hist(babies$weight)
```



Or boxplots

```
> boxplot(babies$weight~babies$gender)
```

Notice here we use the model formula `babies$gender` which enables us to perform boxplots by gender



We can have multiple plots on the one page. The parameter `mfrow` can be used to configure the graphics sheet so that subsequent plots appear row by row, one after the other in a rectangular layout, on the one page. For a column by column layout, use `mfcol` instead. In the example below we present four different transformations of the primates data, in a two by two layout:

```
par(mfrow=c(2,2), pch=16)
  data(Animals)      # Needed if Animals (MASS package) is not a
  attach(Animals)
  plot(body, brain)
  plot(sqrt(body), sqrt(brain))
  plot((body)^0.1, (brain)^0.1)
  plot(log(body),log(brain))
  detach(Animals)
par(mfrow=c(1,1),pch=1)      # Restore to 1 figure per page
```

Simple Stats

We can do simple stats with few problems. If we take the babies data we might test to see if the means differ for the sexes. If we we (naively) to do a t test then

```
> t.test(babies$weight~babies$gender)
```

Welch Two Sample t-test

```
data: babies$weight by babies$gender
t = -0.9775, df = 21.996, p-value = 0.3390
alternative hypothesis: true difference in means is not equal
95 percent confidence interval:
 -351.7111  126.3778
sample estimates:
mean in group f mean in group m
      2911.333      3024.000
```

Of course if you look up the help you will see we can specify the data file to get

```
> t.test(weight~gender,data=babies)
```

Welch Two Sample t-test

data: weight by gender

t = -0.9775, df = 21.996, p-value = 0.3390

alternative hypothesis: true difference in means is not equal

95 percent confidence interval:

-351.7111 126.3778

sample estimates:

mean in group f mean in group m

2911.333

3024.000

Notice we write `weight ~ gender`. This is a model formula and expresses our wish to think of the data vector `weight` in terms of the *factor* `gender`. In this case \mathbb{R} knows that `gender` is a factor but it is up to you to ensure that it is clear that a vector is a factor. Eg

```
> f<-rep(1:3,each=3,length.out=12)
> f
[1] 1 1 1 2 2 2 3 3 3
> f<-factor(f)
> f
[1] 1 1 1 2 2 2 3 3 3
Levels: 1 2 3
```

A more cautious person might use the non-paired Wilcoxon (Mann-Whitney)

```
> wilcox.test(babies$weight~babies$gender)
```

```
Wilcoxon rank sum test
```

```
data: babies$weight by babies$gender
```

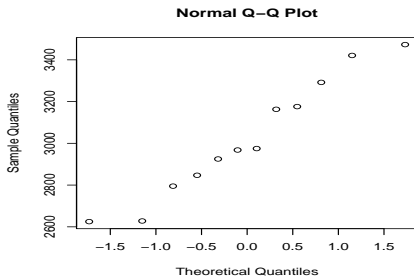
```
W = 58, p-value = 0.4428
```

```
alternative hypothesis: true location shift is not equal to 0
```

```
>
```

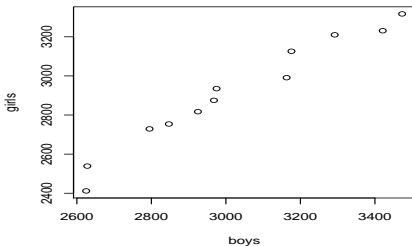
A really cautious person would check for normality. The boys are the first 12 observations, personally I would do a probability plot

```
> boys<-babies$weight[1:12] # makes boys vector  
> qqnorm(boys) #probability plot
```



To compare the boys and girls I would use a Q-Q plot

```
> girls<-babies$weight[13:24]  make a girls vector  
> qqplot(boys,girls)#q-q plot
```



Regression

We now look at some more complex statistics. We begin with regression. To start with we can use the babies data. If you were doing regression you would probably start by using the help system to find out the function and the parameters. The help output is included in these notes, see section ??

Babies

We start by attempting to fit a model which relates time of gestation to weight. The command we want is `lm`. Since it gets irritating to refer to `babies$weight` etc we attach the data frame

```
attach(babies)
```

so we try

```
r1<-lm( weight~weeks)
```

By this we mean we aim to explain the variable `weight` by `time`. In fact if we do this nothing appears to happen. In fact `R` does the computations and puts the output in the object `r1`. To get results we must summarize `r1`, thus

```
> summary(r1)
```

```
Call:
```

```
lm(formula = weight ~ weeks)
```

```
Residuals:
```

Min	1Q	Median	3Q	Max
-262.032	-158.292	8.355	88.147	366.496

```
Coefficients:
```

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-1485.0	852.6	-1.742	0.0955 .
weeks	115.5	22.1	5.228	3.04e-05 ***

```
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1 1
```

```
Residual standard error: 192.6 on 22 degrees of freedom
```

```
Multiple R-Squared: 0.554, Adjusted R-squared: 0.5338
```

```
F-statistic: 27.33 on 1 and 22 DF, p-value: 3.04e-05
```

The object `r1` has lots of information thus

```
> names(r1)
 [1] "coefficients" "residuals"      "effects"         "rank"
 [5] "fitted.values" "assign"         "qr"              "df.resid"
 [9] "xlevels"       "call"          "terms"           "model"
```

and we can plot it - try `plot(r1)`.

If you like/understand the ANOVA we have

```
> anova(r1)
```

```
Analysis of Variance Table
```

```
Response: weight
```

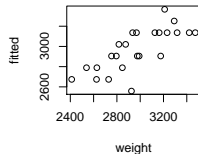
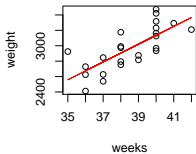
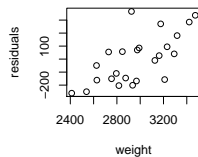
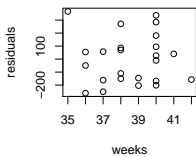
	Df	Sum Sq	Mean Sq	F value	Pr(>F)
weeks	1	1013799	1013799	27.330	3.04e-05 ***
Residuals	22	816074	37094		

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
>
```

I like to look at the residuals, plot the line (which we can do here), plot the residuals against the fitted,

```
> par(mfrow=c(2,2)) #tell R to expect 4 plots
> plot(weights,r1$resid,ylab="residuals") # plot the residuals
> plot(weeks,weight) # plot the data
> lines(weeks,r1$fitted,col=2)# over plot the regression as a
> plot(weight,r1$fitted,ylab="fitted") # plot the actual against
> par(mfrow=c(1,1))# back to single plot
```



Of course it is probable that the gender also has some input so we can try adding gender (a factor) to the model formula

```
> r2<-lm(weight~weeks +gender)
```

```
> summary(r2)
```

Call:

```
lm(formula = weight ~ weeks + gender)
```

Residuals:

Min	1Q	Median	3Q	Max
-257.49	-125.28	-58.44	169.00	303.98

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)	
(Intercept)	-1773.32	794.59	-2.232	0.0367	*
weeks	120.89	20.46	5.908	7.28e-06	***
genderm	163.04	72.81	2.239	0.0361	*

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 177.1 on 21 degrees of freedom

Multiple R-Squared: 0.64, Adjusted R-squared: 0.6057

F-statistic: 18.67 on 2 and 21 DF, p-value: 2.194e-05

```
> anova(r2)
```

Analysis of Variance Table

Response: weight

	Df	Sum Sq	Mean Sq	F value	Pr(>F)	
weeks	1	1013799	1013799	32.3174	1.213e-05	***
gender	1	157304	157304	5.0145	0.03609	*
Residuals	21	658771	31370			

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Here we see the gender has a significant effect, both from the anova and from the coefficient estimates. Our model thus has a slope and different intercepts depending on the gender.

Of course it is possible that the genders have differing slopes - that is we have an interaction between weeks and gender. To model this we use `weeks *gender` a shorthand for `weeks +gender+weeks:gender` where `weeks:gender` is the interaction term.

```

> r3<-lm(weight~weeks *gender)
> summary(r3)
Call:
lm(formula = weight ~ weeks * gender)
Residuals:
    Min       1Q   Median       3Q      Max
-246.69 -138.11  -39.13  176.57  274.28
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  -2141.67    1163.60  -1.841  0.080574 .
weeks         130.40      30.00   4.347  0.000313 ***
genderm       872.99     1611.33   0.542  0.593952
weeks:genderm -18.42      41.76  -0.441  0.663893
Residual standard error: 180.6 on 20 degrees of freedom
Multiple R-Squared:  0.6435, Adjusted R-squared:  0.59
F-statistic: 12.03 on 3 and 20 DF,  p-value: 0.0001010

```

```
> anova(r3)
```

Analysis of Variance Table

Response: weight

	Df	Sum Sq	Mean Sq	F value	Pr(>F)	
weeks	1	1013799	1013799	31.0779	1.862e-05	***
gender	1	157304	157304	4.8221	0.04006	*
weeks:gender	1	6346	6346	0.1945	0.66389	
Residuals	20	652425	32621			

>

For influence measures try

- `summary.lm` for summary and related methods.
- `influence.measures`.
- `hat` for the hat matrix diagonals
- `dfbetas`, `dffits`, `covratio`, `cooks.distance`, `lm`.

Fixed effects ANOVA

The table 1 gives the output of thermocouples made of 3 different materials at 3 different temperatures.

Temperature	material					
	M1		M2		M3	
T1	130	155	34	40	20	70
	74	80	80	75	82	58
T2	150	188	136	122	25	70
	159	126	108	115	58	45
T3	138	110	174	120	96	104
	168	160	150	139	82	60

Table: Output voltages

Possible models are

$$y_{ijk} = \mu + \alpha_i + \beta_j + \epsilon_{ijk} \quad (1)$$

that is just the main effects and

$$y_{ijk} = \mu + \alpha_i + \beta_j + (\alpha\beta)_{ij} + \epsilon_{ijk} \quad (2)$$

where $(\alpha\beta)_{ij}$ denotes the interaction between the main effects of material and temperature. I use a corner-point constraint

$$(\alpha\beta)_{1j} = 0 \quad i = 1, 2, 3 \text{ and } (\alpha\beta)_{i1} = 0 \quad j = 1, 2, 3$$

The table of means, see table 2 is and these are plotted in figure 2. We see that the response over temperature is different for the different materials - hence the interaction. How do we proceed?

	M1	M2	M3
T1	109.75	155.75	144.00
T2	57.25	120.25	145.75
T3	57.50	49.50	85.50

Table: Means of voltages

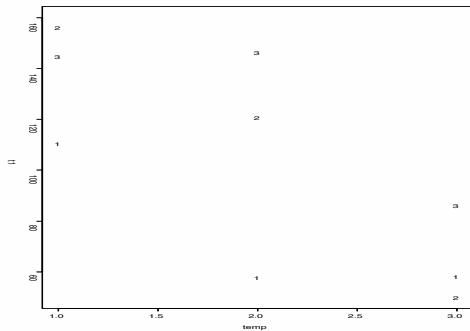


Figure: interaction example

How do we proceed? Type the values into R , call the values y . I also create two vectors `temp` and `mat`. You could just type them in or use the command `rep`

```
> temp<-rep( 1:3 ,each=12,length.out=36)
```

```
> temp
```

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

```
> temp<-factor(temp)
```

```
> mat<-rep( 1:3 ,each=2,length.out=36)
```

```
> mat
```

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

```
> mat<-factor(mat)
```

We must not forget to tell R that we are using factors. With the variables as shown use `lm` to get

```
> y
 [1] 130 155  34  40  20  70  74  80  80  75  82  58 150 188 1
[19] 159 126 108 115  58  45 138 110 174 120  96 104 168 160 1
> temp
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3
Levels: 1 2 3
> mat
 [1] 1 1 2 2 3 3 1 1 2 2 3 3 1 1 2 2 3 3 1 1 2 2 3 3 1 1 2 2 3
Levels: 1 2 3
```

```

> v1<-lm(y~mat*temp)
> summary(v1)
lm(formula = y ~ mat * temp)
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   109.75     12.41   8.845 1.85e-09 ***
mat2          -52.50     17.55  -2.992  0.00586 **
mat3          -52.25     17.55  -2.978  0.00607 **
temp2         46.00     17.55   2.621  0.01421 *
temp3         34.25     17.55   1.952  0.06141 .
mat2:temp2     17.00     24.82   0.685  0.49917
mat3:temp2    -54.00     24.82  -2.176  0.03848 *
mat2:temp3     54.25     24.82   2.186  0.03766 *
mat3:temp3     -6.25     24.82  -0.252  0.80307
Residual standard error: 24.82 on 27 degrees of freedom
Multiple R-Squared:  0.7706, Adjusted R-squared:  0.7026
F-statistic: 11.34 on 8 and 27 DF,  p-value: 7.018e-07

```

```
> anova(v1)
```

```
Analysis of Variance Table
```

```
Response: y
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)	
mat	2	31833	15916	25.8433	5.355e-07	***
temp	2	15734	7867	12.7736	0.0001247	***
mat:temp	4	8296	2074	3.3676	0.0232758	*
Residuals	27	16629	616			

Notice we have several observations per cell in the data table. We can thus estimate the interaction terms. If there was only one observation per cell then the interaction terms would be aliased. They would be inseparable from the main effect terms!

An alternative approach is to use the command `aov`. You might also consider looking at the package `car`. There is also `rem1.R`, it is up to you!

Introduction

In many situations we have complex data sets with several observations on each subject. For example x_1, x_2, \dots, x_6 are measurements made at different sites on the teeth of the following animals.

	x_1	x_2	x_3	x_4	x_5	x_6
modern	97	210	194	77	320	365
jackel	81	167	183	70	303	329
cwolf	135	273	268	106	419	481
iwolf	115	243	245	93	400	446
cuon	107	235	214	85	288	376
dingo	96	226	211	83	344	431
preh	103	221	191	81	323	350

One of the real problems in this case is the difficulty in drawing pictures and seeing what is going on. There are many specialized graphical methods that can be used in these situations but in my view they

- 1 Are difficult to draw, they require specialized software.
- 2 Are difficult to interpret. especially by non-technical users.

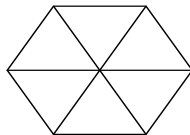
We can illustrate some ideas on the dog data



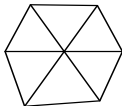
modern



jackel



cwolf



iwolf



cuon



dingo



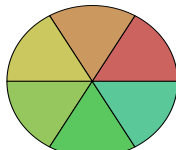
preh



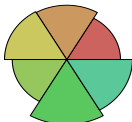
modern



jackel



cwoolf



iwoolf



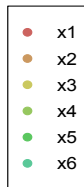
cuon



dingo



preh



modern



jackel



cwolf



iwolf



cuon

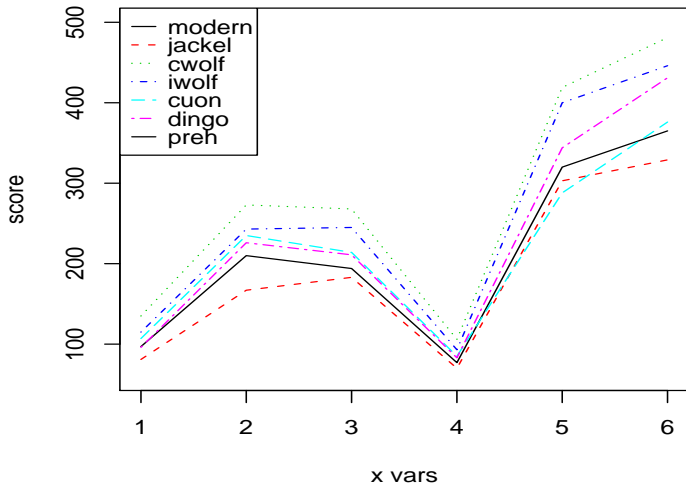


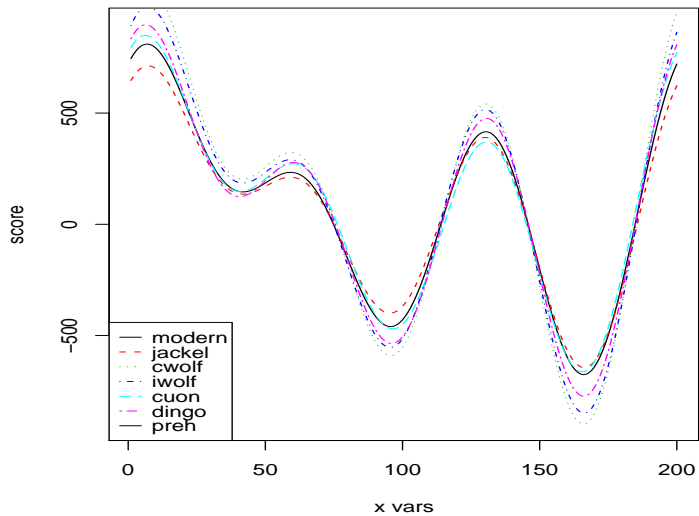
dingo



preh







Principal Components

To determine the interrelationships between the variables we can of course work out the correlations (or covariances) between the \mathbf{x}_j . If $\text{cov}(\mathbf{x}_i, \mathbf{x}_j) = c_{ij}$ then we can write these in a matrix \mathbf{C} whose ij th element is c_{ij} .

For the dogs data this gives the correlation matrix

	x_1	x_2	x_3	x_4	x_5	x_6
x_1	1.000	0.952	0.921	0.978	0.780	0.812
x_2	0.952	1.000	0.883	0.950	0.715	0.854
x_3	0.921	0.883	1.000	0.972	0.876	0.937
x_4	0.978	0.950	0.972	1.000	0.846	0.905
x_5	0.780	0.715	0.876	0.846	1.000	0.889
x_6	0.812	0.854	0.937	0.905	0.889	1.0001

While these correlation are useful they are still a complex structure and quite difficult to decipher. It would be very much simpler if the variables were uncorrelated. In this case we would have a diagonal matrix and from an intuitive view uncorrelated variables seem rather better. Converting to uncorrelated variables is a simple problem in linear algebra. The new variables PC1, PC2, etc are linear combinations of the original variables, here $x_1, x_2, x_3, x_4, x_5, x_6$.

We do not need a very acute grasp of algebra as this is (has to be) done by software. What we need to focus on is that instead of our original measurement variables $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \dots, \mathbf{x}_p)$ we have a new set $(\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \dots, \mathbf{y}_p)$ called the principal components which are uncorrelated and which are made up of our original variables.

For example using R

```
> d1<-prcomp(dogs, scale = TRUE)
```

```
> summary(d1)
```

Importance of components:

	PC1	PC2	PC3	PC4	PC5	
Standard deviation	2.330	0.6077	0.3587	0.2545	0.07821	0.04
Proportion of Variance	0.905	0.0615	0.0215	0.0108	0.00102	0.00
Cumulative Proportion	0.905	0.9664	0.9878	0.9986	0.99964	1.00

We have a bit more detail

```
> d1
```

```
Standard deviations:
```

```
[1] 2.33002707 0.60767458 0.35872870 0.25448045 0.07821380 0.0
```

```
Rotation:
```

	PC1	PC2	PC3	PC4	PC5
x1	0.4099426	0.40138614	-0.45937507	-0.005510479	0.009871866
x2	0.4033020	0.48774128	0.29350469	-0.511169325	-0.376186947
x3	0.4205855	-0.08709575	0.02680772	0.737388619	-0.491604714
x4	0.4253562	0.16567935	-0.12311823	0.170218718	0.739406740
x5	0.3831615	-0.67111237	-0.44840921	-0.404660012	-0.136079802
x6	0.4057854	-0.33995660	0.69705234	-0.047004708	0.226871533

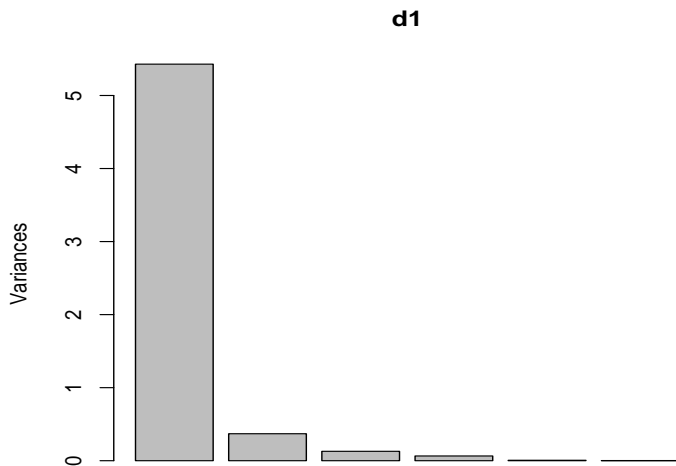
Here we have 4 new variables PC1,PC2, etc which are constructed from our original data,

$$PC1 = 0.4099426*x1+0.4033020*x2+ 0.4205855*x3+0.4253562*x4+0.3831615*x5+ 0.4057854*x6$$

Notice we have used the correlation matrix as the basis of our transformation. We could equally well have used the covariance matrix.

Reducing Dimension

If we plot the variances of the new components we get

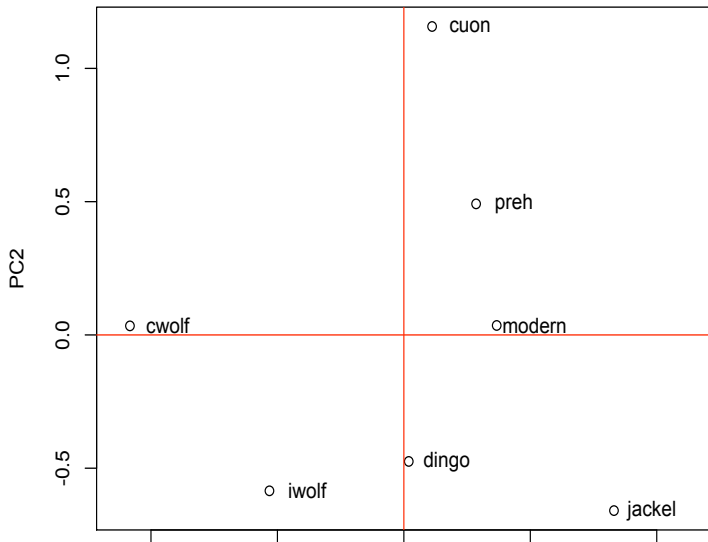


We see that in terms of the variation we have

k	1	2	3	4	5	6
Varn due to kth PC	0.905	0.0615	0.0215	0.0108	0.00102	0.0003
Cumulative variance	0.905	0.9664	0.9878	0.9986	0.99964	1.0000

It looks as though the first one or two principal components explain nearly all the variability. We could think in terms of just these two and reduce the dimensionality of our problem from 6 to 2 .

If we plot the values in the PC scale we have



Summary

- We use eigenvalue analysis to find the principal components.
- These new components are uncorrelated.
- We can use our eigen analysis on either the covariance or the correlation matrix.
- Plotting the variance or standard deviation of the principal components against order, a *scree plot* may help us reduce the dimension of the problem.
- Jolliffe suggests ignoring eigenvalues whose values are less than 1 when doing a correlation based extraction

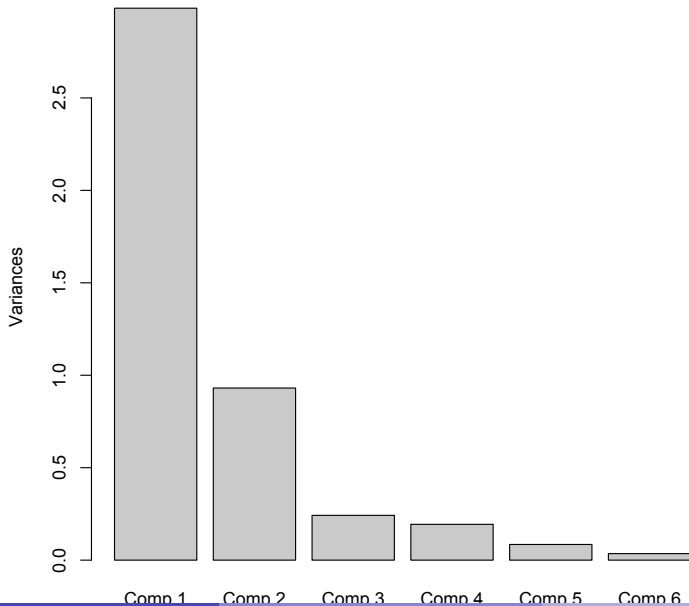
Swiss Bank Notes

Six variables measured on 100 genuine and 100 forged old swiss 1000-franc bills. The observations 1-100 are genuine, the other 100 observations are forged banknotes. The data is used in Applied Multivariate Statistical Analysis (2003) by Wolfgang Hrdle and Lopold Simar.

source: [1988] "Multivariate Statistics, A practical Approach", Cambridge University Press. Flury and Riedwyl

- Length of the bill
- Height of the bill, measured on the left
- Height of the bill, measured on the right
- Distance of inner frame to the lower border
- Distance of inner frame to the upper border
- Length of the diagonal

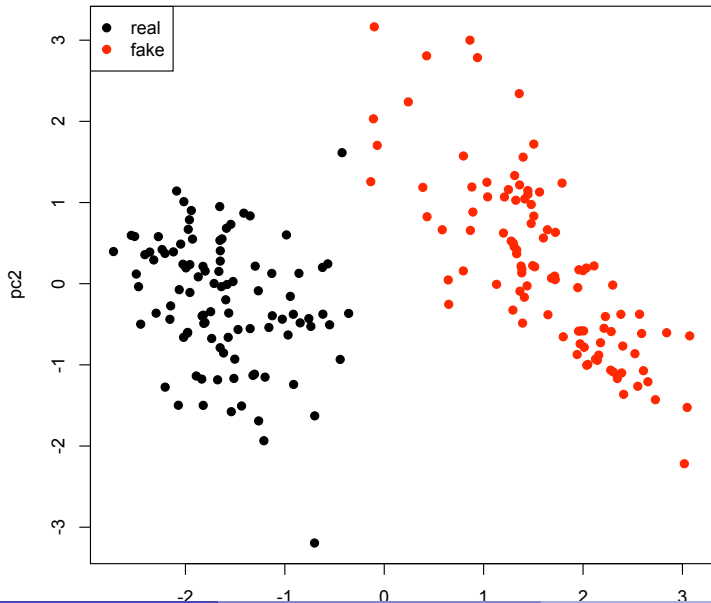
bx



```
summary(bx)
```

```
Importance of components:
```

	Comp.1	Comp.2	Comp.3	Comp.4	Comp.5	Comp.6
Standard deviation	1.7162629	1.1305237	0.9322192	0.6706479	0.4346031	0.4346031
Proportion of Variance	0.4909264	0.2130140	0.1448388	0.0749614	0.03147998	0.03147998
Cumulative Proportion	0.4909264	0.7039403	0.8487791	0.9237405	0.9552205	0.9867005

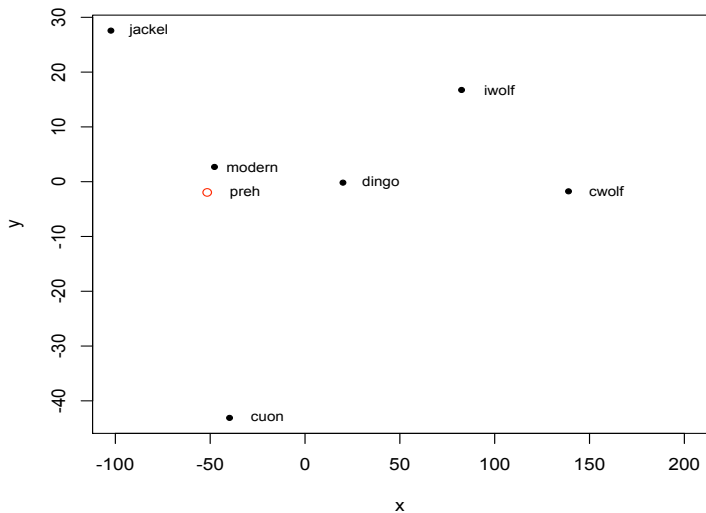


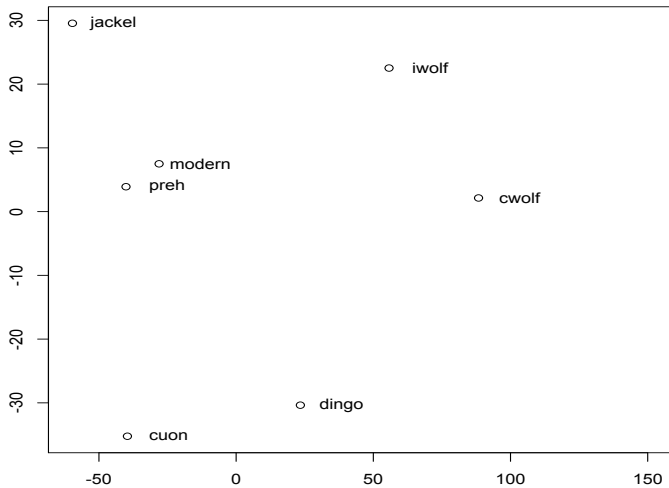
Suppose we take a matrix of distances between major cities. We construct a map to reproduce the distances we are given in the distance matrix. As a result of the MDS analysis, we would get a two-dimensional representation of the locations of the cities, that is, we would basically obtain a two-dimensional map. Of course it may be nothing like the true map

In general , MDS attempts to arrange "objects" (major cities in this example) in a space with a given number of dimensions (2 in our example) so as to reproduce the observed distances. As a result, we can "explain" the distances in terms of underlying dimensions; in our example, we could explain the distances in terms of the two geographical dimensions: north/south and east/west.

Clearly with just distance information the actual orientation of axes in the final solution is arbitrary. In our example, we could rotate the map in any way we want, the distances between cities remain the same. Thus, the final orientation of axes in the plane or space is mostly the result of a subjective decision by the researcher, who will choose an orientation that can be most easily explained.

MDS is not so much an exact procedure as rather a way to "rearrange" objects in an efficient manner, so as to arrive at a configuration that best approximates the observed distances. It actually moves objects around in the space defined by the requested number of dimensions, and checks how well the distances between objects can be reproduced by the new configuration. In more technical terms, it uses a function minimization algorithm that evaluates different configurations with the goal of maximizing the goodness-of-fit (or minimizing "lack of fit").





Projection pursuit

When examining data sets of any dimensionality, researchers are generally looking for subsets of the data that are “interesting”, i.e. that display some measure of structure or departure from normal distribution. This structure can take the form of trends, clusters, hypersurfaces, or anomalies. The traditional approach to examining these high-dimensional data sets is to reduce their dimensionality, usually by linear and/or nonlinear mapping or projection strategies. Humans are very good at visual pattern recognition, and projecting the data set down to one-, two-, or even three-dimensional space allows this ability to be utilized. However, bear in mind that projection is a data-smoothing operation.

The idea of projection pursuit is to locate the projection or projections from high- to low-dimensional space that reveal the most details about the structure of the data set. Once an interesting set of projections has been found, existing structures (clusters, surfaces, etc.) can be extracted and analyzed separately. There are two general approaches taken to projection pursuit: manual and automatic.

The most basic form of manual projection pursuit is the scatterplot, which is, in its most simple form, a two-dimensional display to indicate data characteristics over two selected dimensions at a time. It is quite simple to produce all $\binom{n}{2}$ pair-wise scatterplots for n -dimensional space and perform analysis on these. Unfortunately, this method only allows structure across the two plotted dimensions to be discovered. When the number of dimensions to analyze becomes very large, other projection methods must be considered.

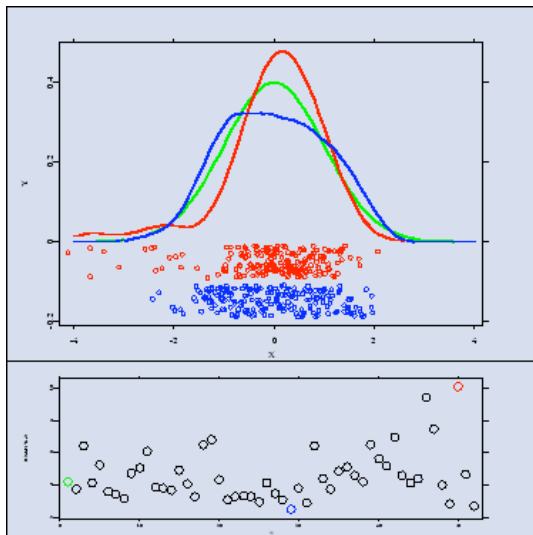
The main limitation of manual projection pursuit is the amount of time it takes to exhaustively explore a given space. If one were to use Asimov's Grand Tour concept which describes presenting projections of the data set in a sequence with a difference of views so slight as to make the sequence similar to watching a movie, and make a complete search of a high-dimensional space. —t may take three hours to completely explore a four-dimensional space! Clearly, touring spaces of even higher dimensionality would be out of the question.

Friedman and Tukey devised a method to automate the task of projection pursuit. They characterize a given projection by a numerical index that indicates the amount of structure that is present. This index can then be used as the basis for a heuristic search to locate the “interesting” projections. Different types of heuristic searches have been suggested

Once some structure has been found, it is then removed from the data. The data are then examined for further structures, which, if found, are also removed. This process continues until there is no remaining structure detectable within the data.

Projection pursuit methods are a great step forward in the problem of high-dimensional data analysis, but they do have many limitations. One of the most common problems is the difficulty in determining just what the solutions from automatic projection pursuit methods actually mean. Also, most projection pursuit software can get fooled by false structure.

Exploratory Projection Pursuit for the Swiss bank notes data (green = standard normal, red = best, blue = worst) gives



The figure shows the density for the standard, normally distributed data (green) and the estimated densities for the best (red) and the worst (blue) projections found. A dotplot of the projections is also presented. In the lower part of the figure we see the estimated value of the Friedman-Tukey index for each computed projection. From this information we can judge the non normality of the bank note data set since there is a lot of variation across the 50 random projections.

We have defines Exploratory Projection Pursuit is a technique used to find interesting structures in high-dimensional data via low-dimensional projections. Since the Gaussian distribution represents a standard situation, we define the Gaussian distribution as the most uninteresting. The search for interesting structures is done via a projection score like, for example, the Friedman-Tukey index $I_{\text{FT}}(\alpha) = \int f^2$. The parabolic distribution has the minimal score. We maximize this score over all projections.

The Jones-Sibson index maximizes

$$I_{\text{JS}}(\alpha) = \{\kappa_3(\alpha^\top X) + \kappa_4^2(\alpha^\top X)/4\}/12$$

as a function of α

The entropy index maximizes

$$I_{\text{E}}(\alpha) = \int f \log f$$

where f is the density of $\alpha^\top X$.

Projection pursuit regression

The recursive partitioning regression (RPR) basically operates as follows: A certain split coordinate giving the best variance reduction determines two hyper-rectangles on which constant regression surfaces are fitted. This splitting procedure is then applied recursively to each of the regions obtained. An obvious limitation of this RPR is that splits only occur parallel to particular coordinate projections. Regression functions that are piecewise constant, but in a different, rotated coordinate system, would not be approximated well. A simple function such as $m(x) = m(x_1, x_2) = x_1 x_2$ would not be well represented by the RPR technique.

Note that this particular m can be written as $\frac{1}{4}(x_1 + x_2)^2 - \frac{1}{4}(x_1 - x_2)^2$, a sum of two functions operating on projections

$$\beta_1^T x = (1, 1) \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

and

$$\beta_2^T x = (1, -1) \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}.$$

This motivates a generalization of RPR: Instead of using constant functions of projections along the coordinate axis, the regression surface is approximated by a sum of empirically determined univariate ridge functions $\{g_j\}$ of projections $\beta_j^T x$,

$$m(x) = \sum_{j=1}^p g_j(\beta_j^T x) .$$

(10.2.4)

This representation need not be unique. The ridge functions $\{g_j\}$ can be thought of as generalizations of linear functions: They are also constant on hyperplanes.

The idea of approximating high dimensional functions by simpler functions that operate on projections goes back at least to Kruskal (1969).

Friedman and Tukey (1974) applied this idea of searching for “interesting” projections in an analysis of a particle physics data set.

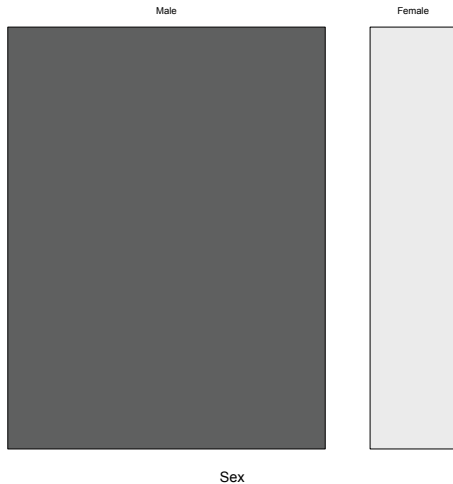
What is a mosaic plot?

A mosaic plot is a graphical display that allows you to examine the relationship among two or more categorical variables.

The mosaic plot starts as a square with length one. The square is divided first into horizontal bars whose widths are proportional to the probabilities associated with the first categorical variable. Then each bar is split vertically into bars that are proportional to the conditional probabilities of the second categorical variable. Additional splits can be made if wanted using a third, fourth variable, etc.

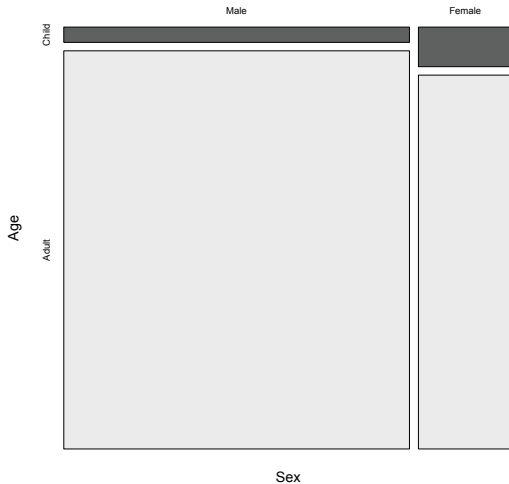
Here we take the data set on the mortality rates aboard the Titanic, which are influenced strongly by age, sex, and passenger class. If you wanted to compare the mortality rates between men and women using a mosaic plot, you would first divide the unit square according to the overall proportion of males and females.

Titanic

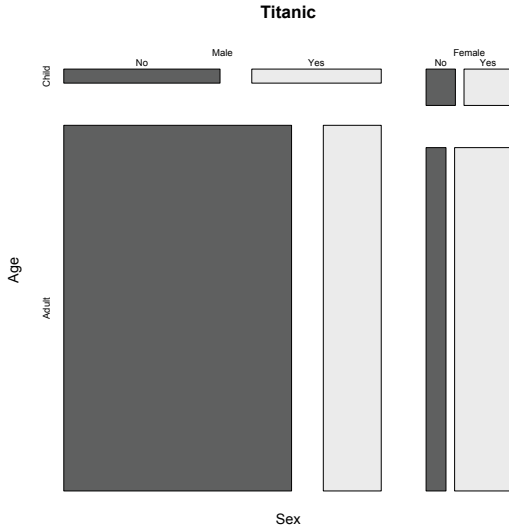


About 35% of the passengers were female, so the first split of the mosaic plot is 35/65. Next, split each bar vertically according to the proportion who were adult.

Titanic

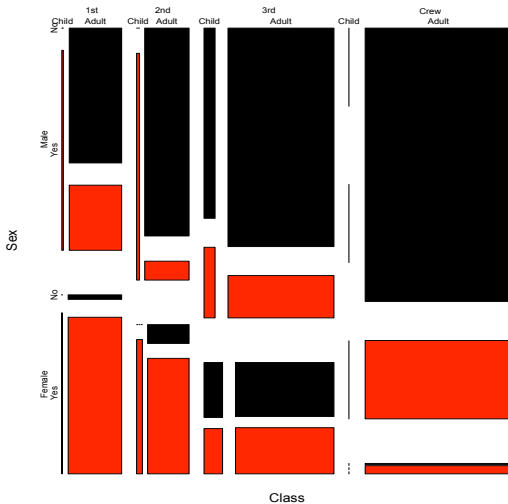


Now we split by survivors.



You can add a third split to examine the influence of the combination sex and passenger class on mortality.

Survival on the Titanic



This graph is worth staring at for quite a while. Notice that the mortality rate climbs very sharply for females when you move from first to third class. The rate climbs among males as well, but not as sharply. Also notice that females are found among 1st class passengers in numbers that are disproportionately large relative to their overall numbers. In contrast, more than half of the males were found among the 3rd class passengers.

For further reading see www.math.yorku.ca/SCS/friendly.html