Econometric modelling in finance and insurance with the R language

Arthur Charpentier

charpentier.arthur@uqam.ca

http ://freakonometrics.hypotheses.org/

Université du Québec à Montréal

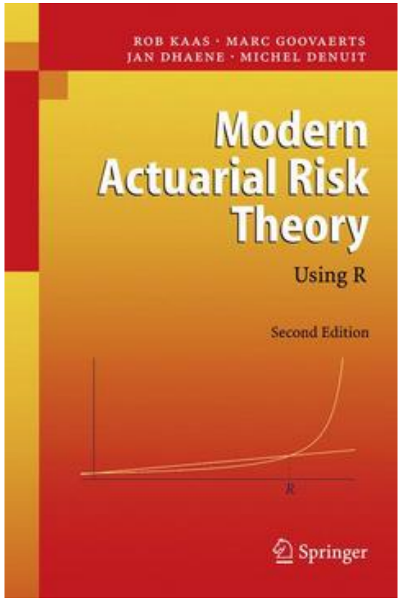
February 2013

Part I.
Introduction to the R language

R

"*R (and S) is the 'lingua franca' of data analysis and statistical computing, used in academia, climate research, computer science, bioinformatics, pharmaceutical industry, customer analytics, data mining, finance and by some insurers. Apart from being stable, fast, always up-to-date and very versatile, the chief advantage of R is that it is available to everyone free of charge. It has extensive and powerful graphics abilities, and is developing rapidly, being the statistical tool of choice in many academic environments.*"

Appendix A
The 'R' in Modern ART

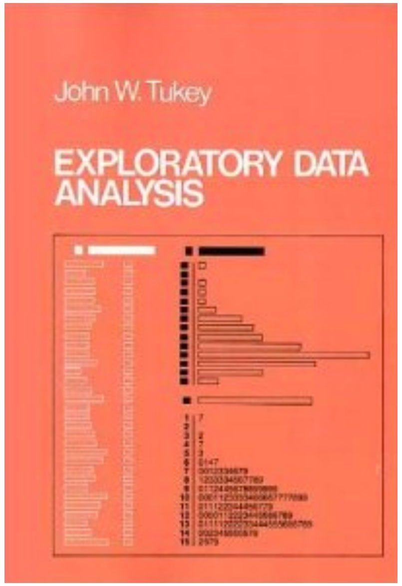


A brief history of R

R is based on the S statistical programming language developed by John Chambers at Bell labs in the 80's

R is an open-source implementation of the S language, developed by Robert Gentlemn and Ross Ihaka (released under the GPL license, *General Public License*).

Before R, and S

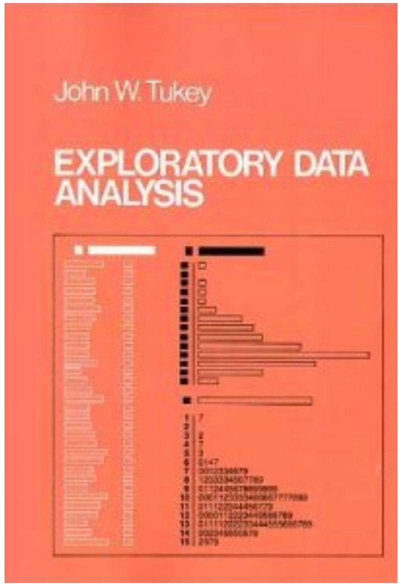


Exploratory Data Analysis (EDA) is an approach/philosophy for data analysis that employs a variety of techniques (mostly graphical) to

1. maximize insight into a data set ;
2. uncover underlying structure ;
3. extract important variables ;
4. detect outliers and anomalies ;
5. test underlying assumptions ;
6. develop parsimonious models ; and
7. determine optimal factor settings.

Source : : http ://www.itl.nist.gov/div898/handbook/eda/section1/eda11.htm

Before R, and S



EDA is an approach to data analysis that postpones the usual assumptions about what kind of model the data follow with the more direct approach of allowing the data itself to reveal its underlying structure and model. EDA is not a mere collection of techniques ; EDA is a philosophy as to how we dissect a data set ; what we look for ; how we look ; and how we interpret.

Most EDA techniques are graphical in nature with a few quantitative techniques.

Source : : http ://www.itl.nist.gov/div898/handbook/eda/section1/eda11.htm

The success of R, and S

"*The purpose of statistical software is to help in the process of learning from data*", Chambers (2000).

1998 : Chambers won the ACM (*Association for Computing Machinery*) Software System Award ; S has "*forever altered the way people analyze, visualize and manipulate data*"

Source : : http ://www.acm.org/announcements/ss99.html

R, and S, in 2010

Forbes

Steve McNally, Subscriber

+ Follow (17) Follow 719

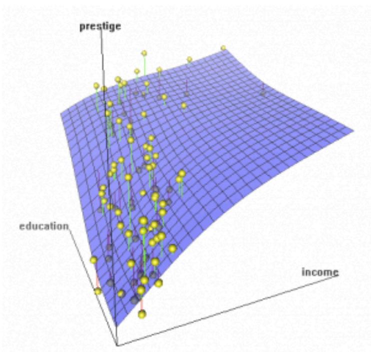
11/10/2010 @ 11:50AM | 21 152 views

Names You Need to Know in 2011: R Data Analysis Software

65 comments, 56 called-out + Comment Now + Follow Comments

Simply put by one of its staunchest advocates, "R is the most powerful statistical computing language on the planet; there is no statistical equation that cannot be calculated in R."

Beyond "just" a language, R is a toolset, a community, and a lot of free software.



"Everyone can, with open source R," Norman Nie says in Quentin Hardy's article, "afford to know exactly the value of their house, their automobile," their current business and prospects. Nie has built a successful business providing services and support for R. (Thanks to community member johnkolchak for this correction.)

Ross Ihaka and Robert Gentleman, then both at Auckland University in New Zealand, created the R Project informally around 1990. The R Core

The New York Times

Data Analysts Captivated by R's Power

Stuart Isett for The New York Times

R first appeared in 1996, when the statistics professors Robert Gentleman, left, and Ross Ihaka released the code as a free software package.

By ASHLEE VANCE
Published: January 6, 2009

TWITTER
LINKEDIN
SIGN IN TO E-MAIL
PRINT
REPRINTS
SHARE

To some people R is just the 18th letter of the alphabet. To others, it's the rating on racy movies, a measure of an attic's insulation or what pirates in movies say.

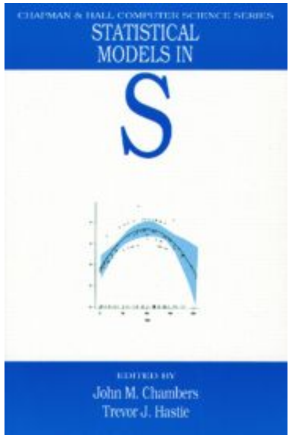
Related

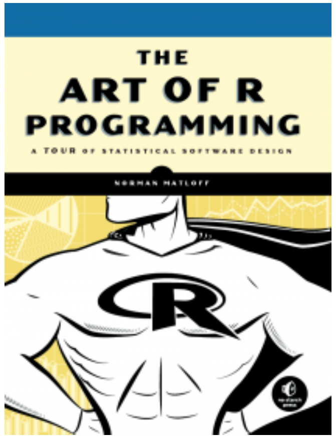
Bits: R You Ready for R?

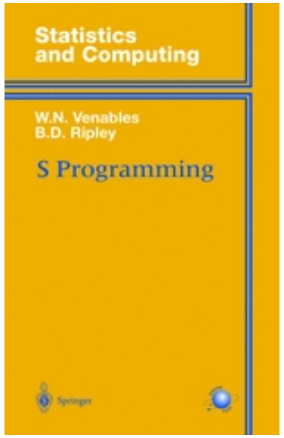
The R Project for Statistical Computing

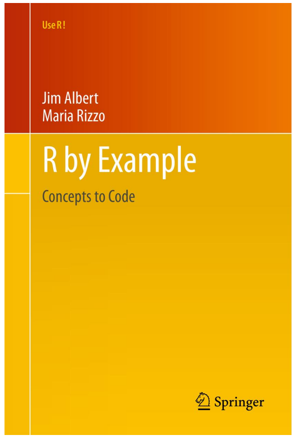
R is also the name of a popular programming language used by a growing number of data analysts inside corporations and academia. It is becoming their lingua franca partly because data mining has entered a golden age, whether being used to set ad prices, find new drugs more quickly or fine-tune financial models. Companies as diverse as Google, Pfizer, Merck, Bank of America, the InterContinental Hotels Group and Shell use it.

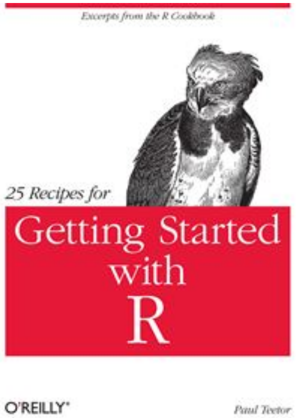
R, and S, in 2013

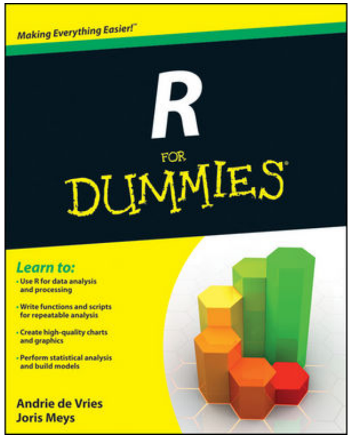


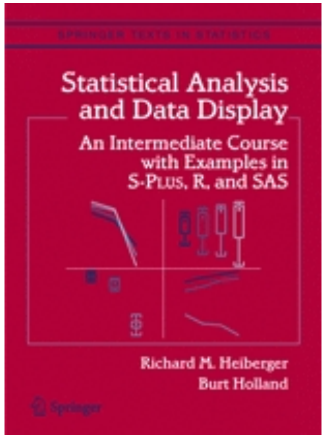






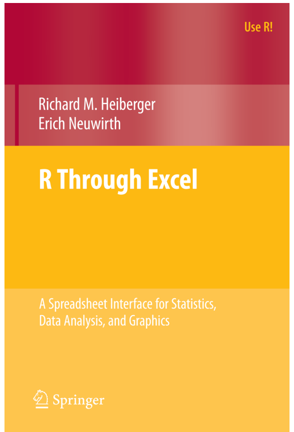


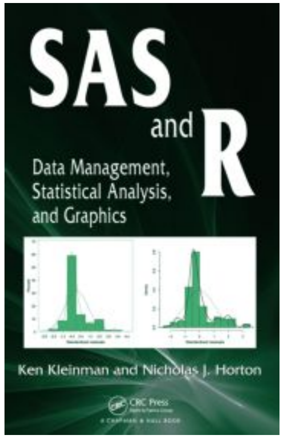


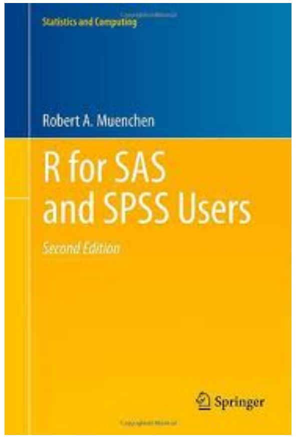


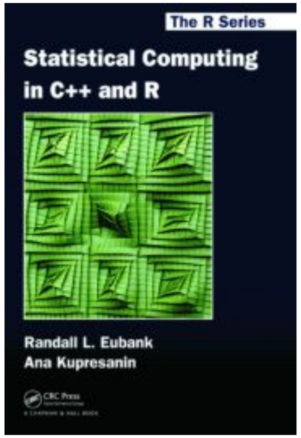
R, and S, in 2013

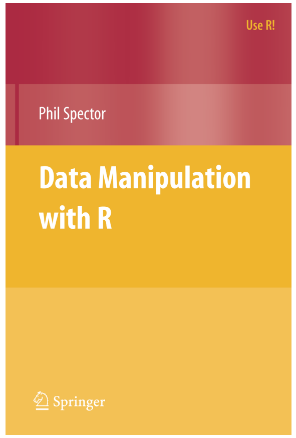
R, and S, in 2013

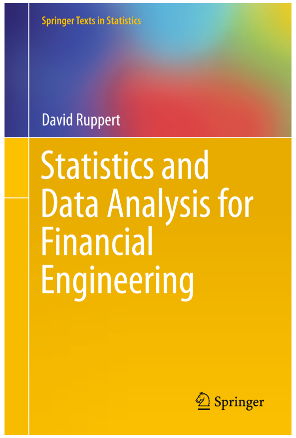


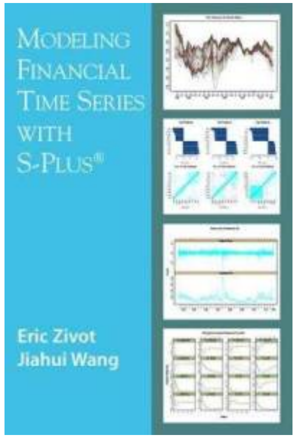


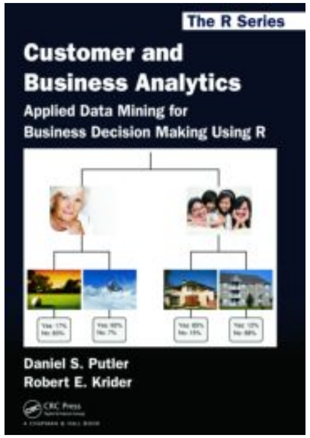


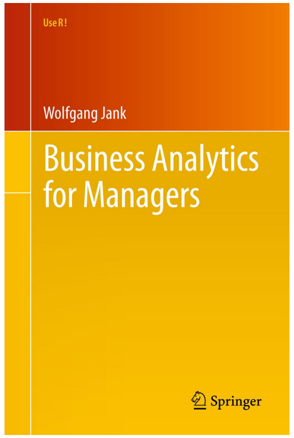


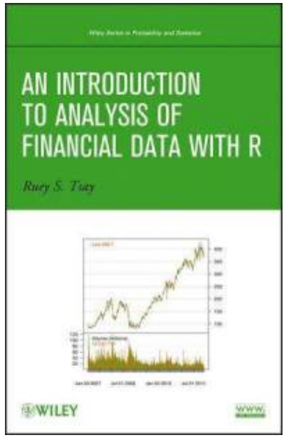


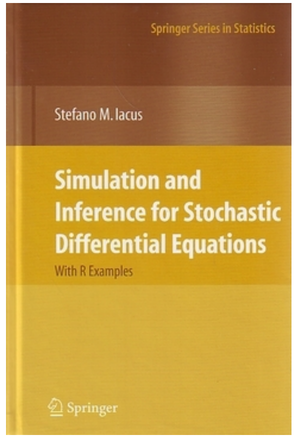






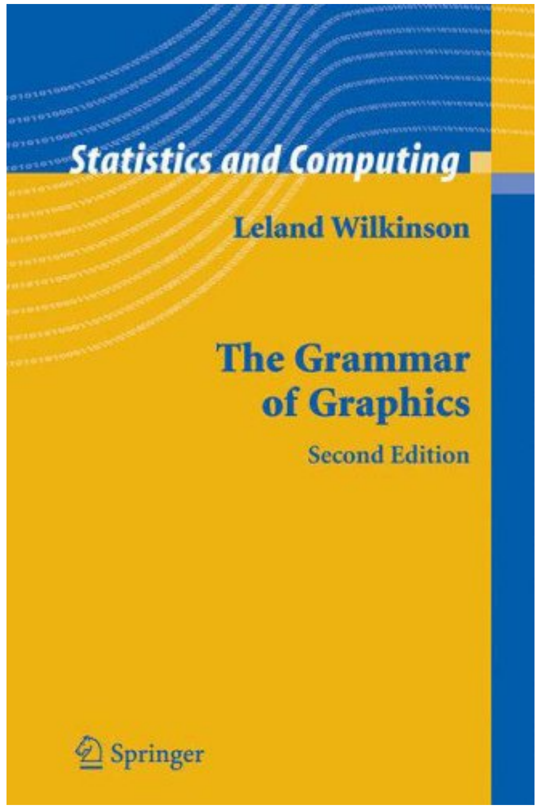


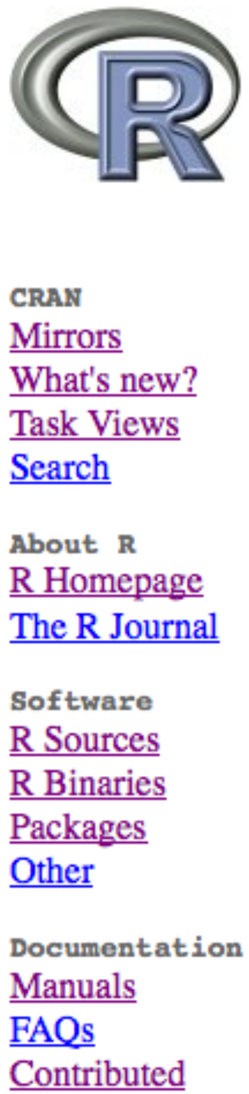




R, and S, in 2013

ggplot2 is based on a classic in the data visualization literature





The R community : http ://cran.r-project.org/

"*I can't think of any programming language that has such an incredible community of users. If you have a question, you can get it answered quickly by leaders in the field. That means very little downtime.*" Mike King, Quantitative Analyst, Bank of America.
"*The most powerful reason for using R is the community*" Glenn Meyers, in the Actuarial Review.
"*The great beauty of R is that you can modify it to do all sorts of things. And you have a lot of prepackaged stuff that's already available, so you're standing on the shoulders of giants*", Hal Varian, chief economist at Google.

Source : : http ://www.nytimes.com/2009/01/07/technology/business-computing/07program.html

R news and tutorials contributed by 425 R bloggers

Source : : http ://www.r-bloggers.com/

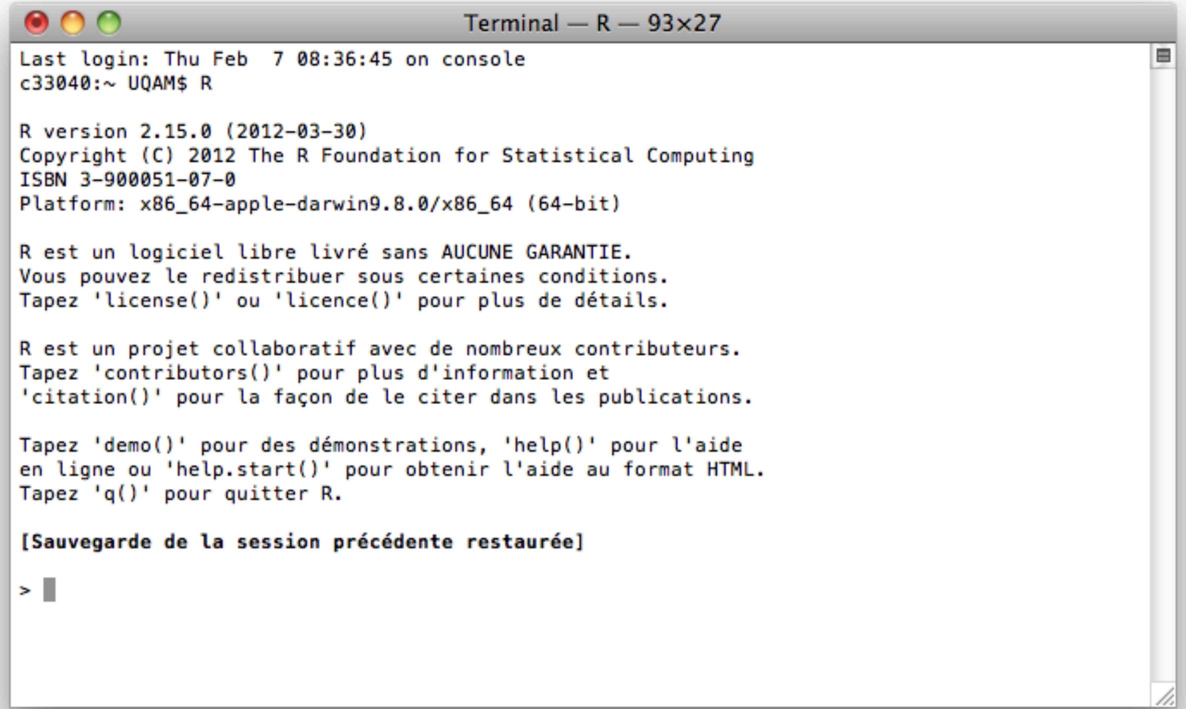
Agenda

- **The R language**
  - Opening R - or RStudio
  - Objects in R
  - Simple operations with R
  - Importing datasets with R
- **Functions with R**
- **Graphs with R**
- **R versus other softwares**

But the first step is to install R from : http ://cran.r-project.org/

R with Linux

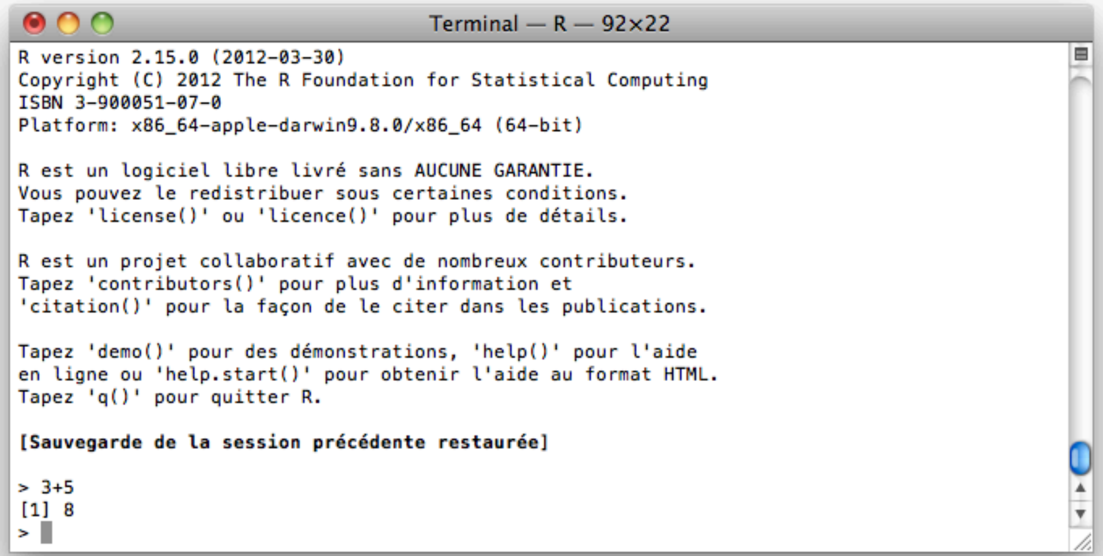
R can be started in a Unix terminal window, simply typing the command `R`.



One gets a prompt. R has a simple interface.

R with Linux

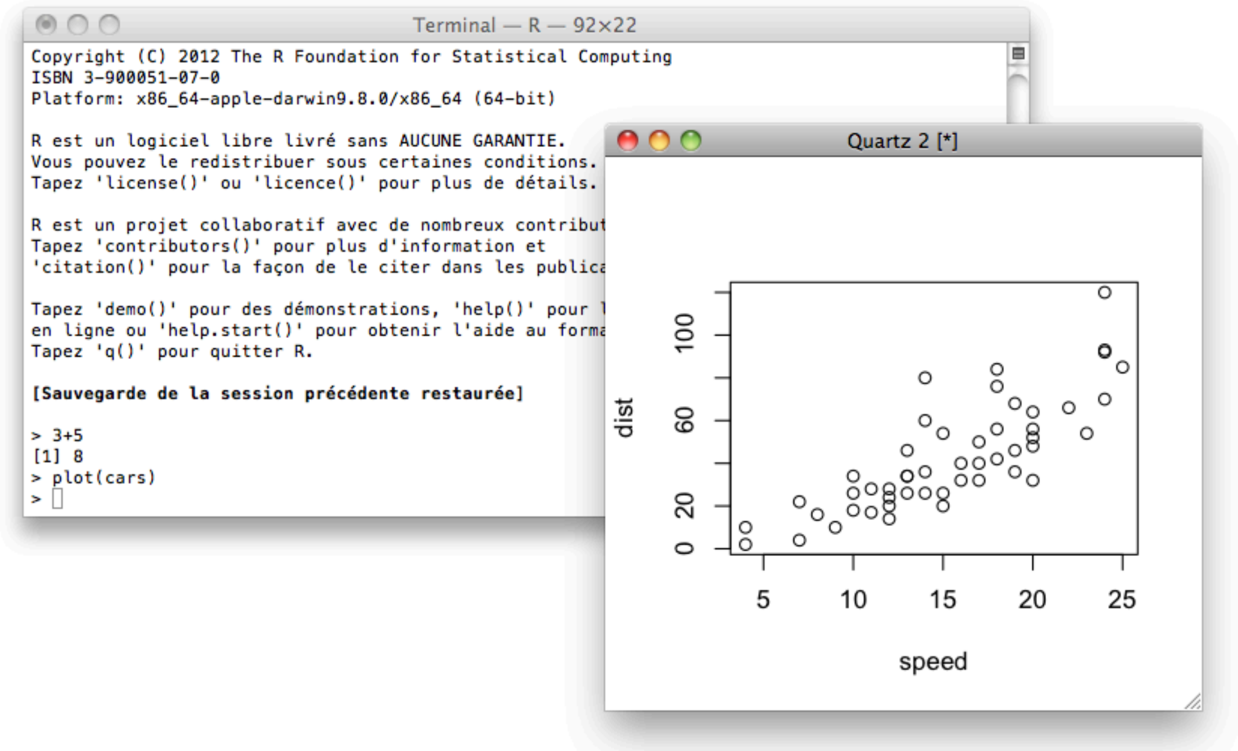
The most basic interaction is : entering expressions, the system will evaluate them, and then print a result.



Ris a calculator that can perform basic arithmetic operations.

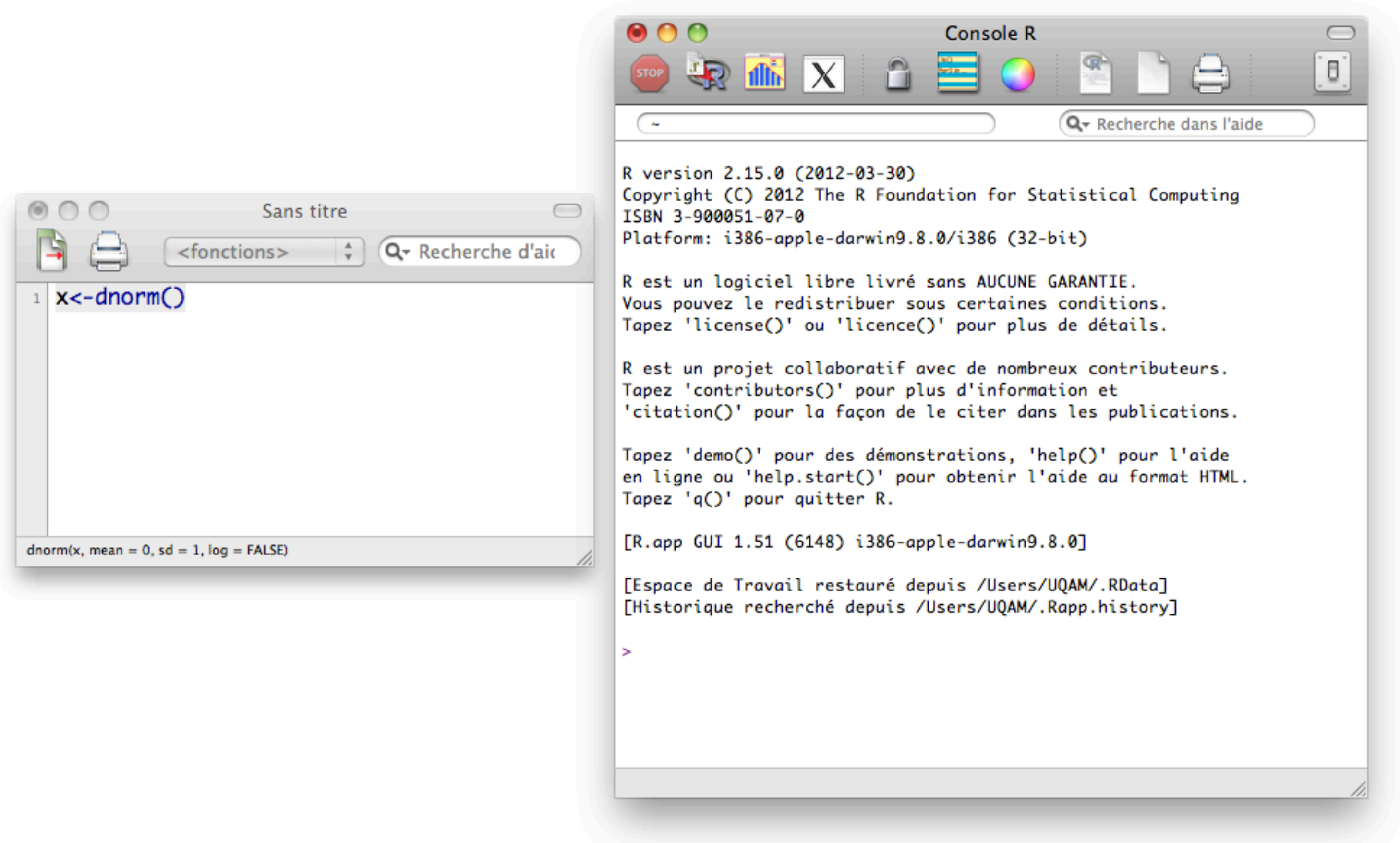
R with Linux

One should make a distinction between the command line shell and the graphical shell,



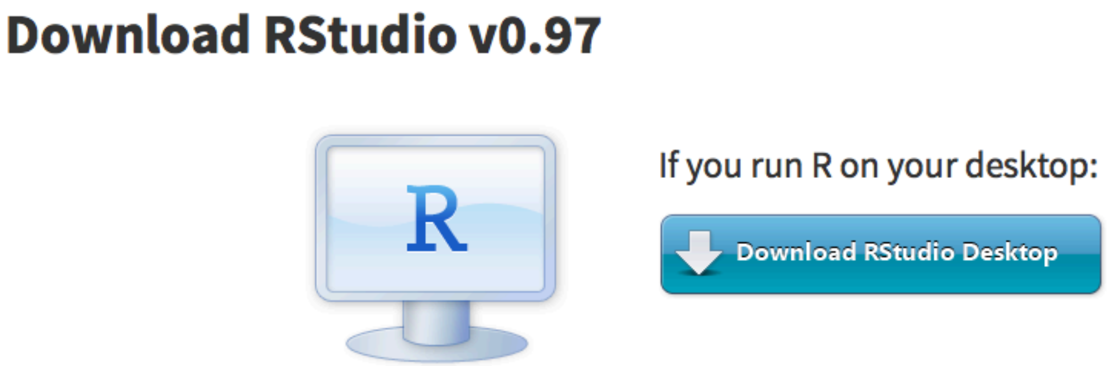
R with Mac, or Windows

With a Mac or Windows OS, one can get a more advanced R interface, with a console (the command line shell), a graphical shell, and a script shell,



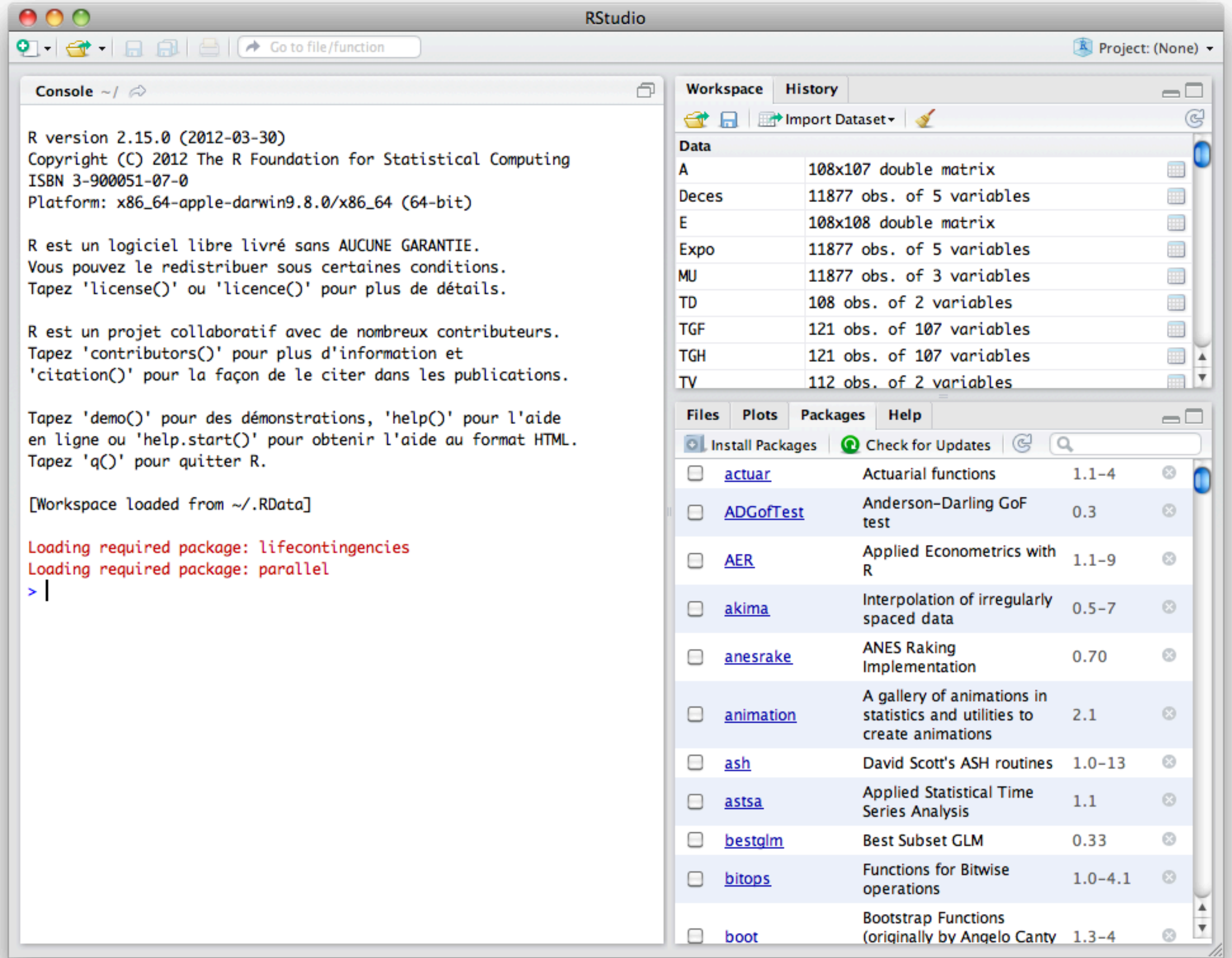
Integrated development environment for R

Note that it is possible possible to use some free and open source integrated development environment for R, e.g. RStudio



Source : : http ://www.rstudio.com/ide/download/

Integrated development environment for R

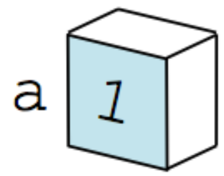


Simple operations with R

“*Everything in S is an object.*”

“*Every object in S has a class.*”

Simple operations with R




```
> a <- 1
> a
[1] 1
> class(a)
[1] "numeric"
> is.numeric(a)
[1] TRUE
> is.real(a)
[1] TRUE
> class(a==1)
[1] "logical"
> a+1
[1] 2
> ls()
 [1] "a"
 > A
Error : object 'A' not found
```

Simple operations with R

From a technical point of view, R uses 'copying' semantics, which makes R a 'pass by value' language

```
> a <- 1
> b <- a
> a <- 2
> a
[1] 2
> b
[1] 1
```

i.e. when we assign a value to another, it is not linked to the original one.

Those objects (that we created) are stored in a file called .RData (in the directory where we started R),

The R workspace

Our workspace is one of the several locations where R can find objects.

```
> find("a")
[1] ".GlobalEnv"
```

Remark : Our workspace is just an environment in R (i.e. a mapping between names, and values)

Note that predefined objects are stored elsewhere

```
> find("pi")
[1] "package:base"
```

The R workspace

Objects can be stored in several locations,

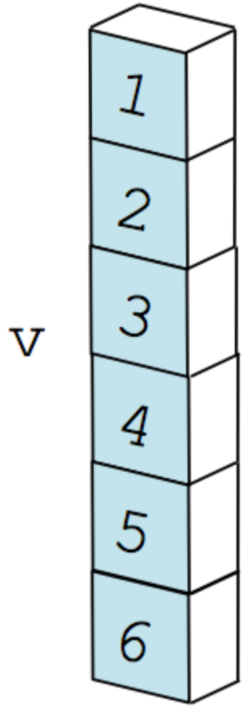
```
> search()
 [1] ".GlobalEnv"         "tools:RGUI"         "package:stats"
 [4] "package:graphics"   "package:grDevices" "package:utils"
 [7] "package:datasets"   "package:methods"    "Autoloads"
[10] "package:base"
```

Remark : To save our workspace use

```
> save.image()
```

Simple operations with R

```
> v <- c(1,2,3,4,5,6)
> v
[1] 1 2 3 4 5 6
> v=seq(from=1,to=6,by=1)
> v
[1] 1 2 3 4 5 6
> v=1:6
> v
[1] 1 2 3 4 5 6
> class(v)
[1] "numeric"
> v*3
[1]  3  6  9 12 15 18
> mean(v)
[1] 3.5
> sort(v,decreasing=TRUE)
[1] 6 5 4 3 2 1
```



Simple operations with R

When displaying a vector R lists the elements, from the left to the right, using (possibly) multiple rows (depending on the width of the display).

Each new row includes the index of the value starting that row, i.e.

```
> u <- 1:50
> u
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
[17] 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
[33] 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
[49] 49 50
```

Remark : singles values are interpreted as vectors of length 1

```
> a
[1] 2
```

Simple operations with R

Important functions to generate vectors are `c(...)` to concatenate series of elements (having the same type), but also `seq` to generate a sequence of elements evenly spaced

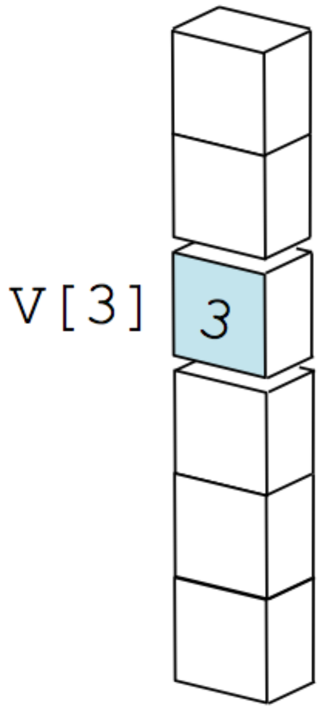
```
> seq(from=0, to=1, by=.1)
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
 > seq(5,2,-1)
[1] 5 4 3 2
> seq(5,2,length=9)
[1] 5.000 4.625 4.250 3.875 3.500 3.125 2.750 2.375
[9] 2.000
```

or `rep` which replicates elements

```
> rep(c(1,2,6),3)
[1] 1 2 6 1 2 6 1 2 6
> rep(c(1,2,6),each=3)
[1] 1 1 1 2 2 2 6 6 6
```

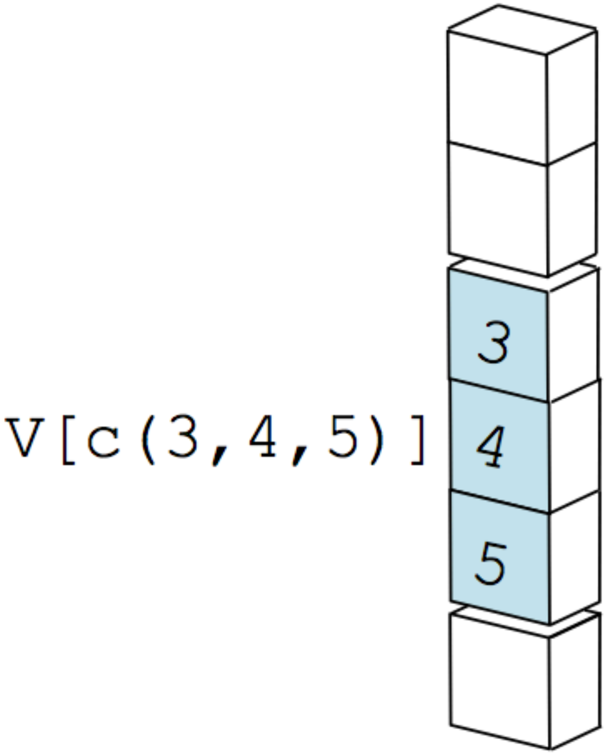
Simple operations with R

```
> v[3]
[1] 3
> v[3] <- 0
> v
[1] 1 2 0 4 5 6
> v[v==0] <- NA
> v
[1]  1  2 NA  4  5  6
> v[3] <- 3
> v
[1] 1 2 3 4 5 6
```



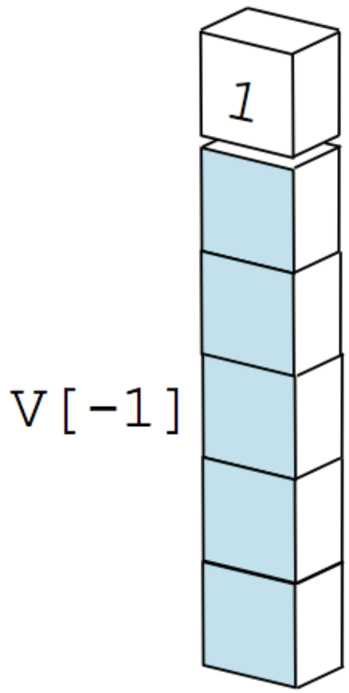
Simple operations with R

```
> v[c(3,4,5)]
[1] 3 4 5
> v[c(3,4,5)] <- v[c(3,4,5)]^2
> v
[1]  1  2  9 16 25  6
> v>5
[1] FALSE FALSE  TRUE  TRUE  TRUE  TRUE
>  which(v>5)
[1] 3 4 5 6
> v[v>5]
[1]  9 16 25  6
> v[v%%2==0]
[1]  2 16  6
> v <- 1:6
```



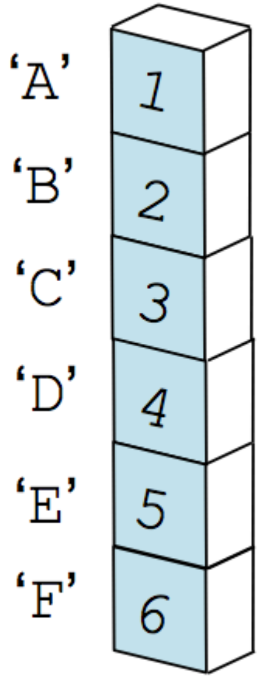
Simple operations with R

```
> v[-1]
[1] 2 3 4 5 6
> v[-c(1,5)]
[1] 2 3 4 6
> v[-which(v%%2==0)]
[1] 1 3 5
```



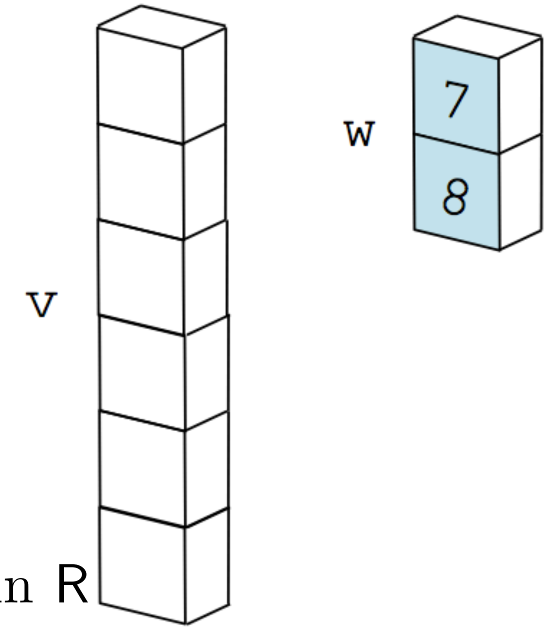
Simple operations with R

```
> names(v)
NULL
> names(v) <- c("A","B","C","D","E","F")
> v
A B C D E F
1 2 3 4 5 6
> names(v) <- letters[1:length(v)]
> v
a b c d e f
1 2 3 4 5 6
> names(v) <- toupper(letters[1:length(v)])
> names(v)
[1] "A" "B" "C" "D" "E" "F"
> v[c("B","F")]
B F
2 6
```



Simple operations with R

```
> w <- c(7,8)
> w
[1] 7 8
> c(v,w)
[1] 1 2 3 4 5 6 7 8
> v <- c(as.numeric(v),w)
[1] 1 2 3 4 5 6 7 8
```



Most standard functions for vector manipulation do exist in R

```
> sum(v)
[1] 36
> cumsum(v)
[1]  1  3  6 10 15 21 28 36
```

Simple operations with R

From a technical point of view, vectors are ordered collections of elements of the same type, which can be `numeric` (in $\mathbb{R}$), `complex` (in $\mathbb{C}$), `integer` (in $\mathbb{N}$), `character` for characters or strings, `logical` namely `FALSE` or `TRUE` (or in $\{0, 1\}$).

Remark vectors are collections of data of the same type. If not, R will coerce elements to a common type,

```
> x <- c(1:5,"yes")
> x
[1] "1"   "2"   "3"   "4"   "5"   "yes"
> y <- c(TRUE,TRUE,TRUE,FALSE)
> y
[1]  TRUE  TRUE  TRUE FALSE
> y+2
[1] 3 3 3 2
```

Simple operations with R

Keep in mind that $\mathbb{R}$ does not exist for computers.

```
> sqrt(2)^2
[1] 2
> sqrt(2)^2 == 2
[1] FALSE
> sqrt(2)^2 - 2
[1] 4.440892e-16
```

To compare numbers (properly) one should use

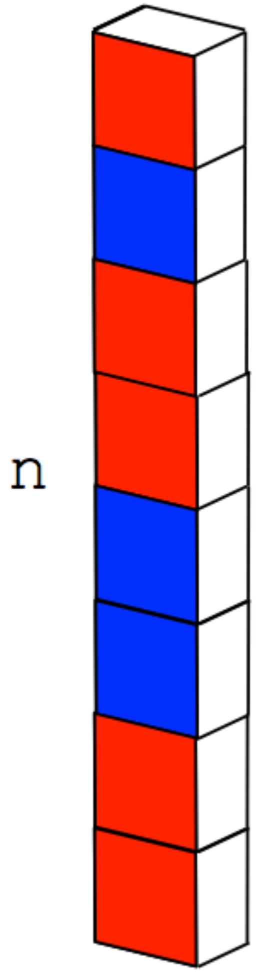
```
> all.equal(sqrt(2)^2,2)
[1] TRUE
```

Another example ?

```
> (3/10-1/10) == (7/10-5/10)
[1] FALSE
> (3/10-1/10) - (7/10-5/10)
[1] 2.775558e-17
```

Simple operations with R

```
> n <- c("R","B","R","R","B","B","R","R")
> n
[1] "R" "B" "R" "R" "B" "B" "R" "R"
> class(n)
[1] "character"
> paste(n,v,sep="-")
[1] "R-1" "B-2" "R-3" "R-4" "B-5" "B-6" "R-7" "R-8"
> n == "R"
[1]  TRUE FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE
> n <- as.factor(n)
> n
[1] R B R R B B R R
Levels: B R
```



Simple operations with R

Many functions can be used for factors (i.e. categorical variables)

```
> unclass(n)
[1] 2 1 2 2 1 1 2 2
attr(,"levels")
[1] "B" "R"
> new.n <- factor(n,labels=c("Male","Female"))
> new.n
[1] Female Male   Female Female Male   Male   Female Female
Levels: Male Female
> relevel(new.n,"Female")
[1] Female Male   Female Female Male   Male   Female Female
Levels: Female Male
```

Simple operations with R

Many functions can be used for characters or strings, e.g.

```
> cities <- c("New York, NY", "Los Angeles, CA", "Boston, MA")
> substr(cities, nchar(cities)-1, nchar(cities))
[1] "NY" "CA" "MA"
> unlist(strsplit(cities, ", "))[seq(2,6,by=2)]
[1] "NY" "CA" "MA"
```

or on dates

```
> dates <- c("16/Oct/2012:07:51:12","19/Nov/2012:23:17:12")
> some.dates <- strptime(dates,format="%d/%b/%Y:%H:%M:%S")
> some.dates
[1] "2012-10-16 07:51:12" "2012-11-19 23:17:12"
> diff(some.dates)
Time difference of 34.68472 days
```

Simple operations with R

Many functions can be used for characters or strings, e.g.

```
> some.dates <- as.Date(c("16/10/12","19/11/12"),format="%d/%m/%y")
> some.dates
[1] "2012-10-16" "2012-11-19"
> sequence.date <- seq(from=some.dates[1],to=some.dates[2],by=7)
> sequence.date
[1] "2012-10-16" "2012-10-23" "2012-10-30" "2012-11-06" "2012-11-13"
> format(sequence.date,"%b")
[1] "oct" "oct" "oct" "nov" "nov"
> weekdays(some.dates)
[1] "Tuesday" "Monday"
> Months <- months(sequence.date)
> Months
[1] "october"  "october"  "october"  "november" "november"
> Year <- substr(as.POSIXct(sequence.date), 1, 4)
> Year
[1] "2012" "2012" "2012" "2012" "2012"
```

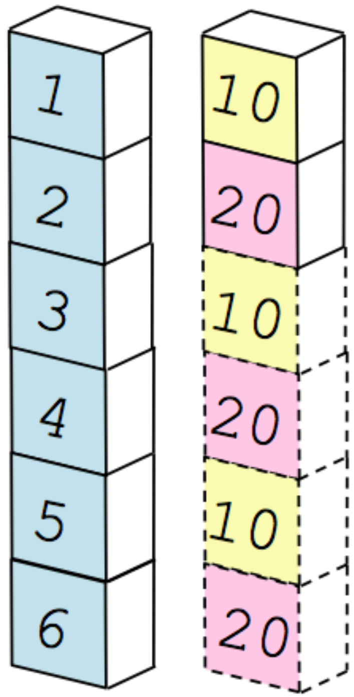
Simple operations with R

R has a recycling rule : when adding two vectors with different lengths, the shorter one is recycled,

```
> v+c(10,20)
[1] 11 22 13 24 15 26
```

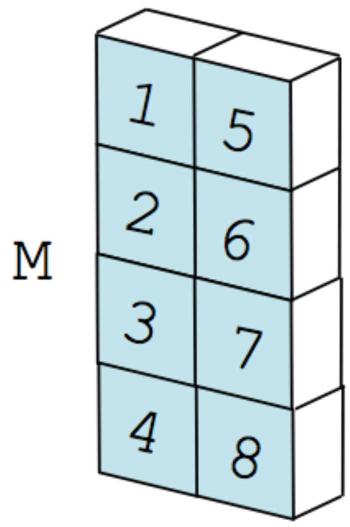
Remark : this rule is implicit when adding a numerical value (vector for length 1) to a vector

```
> v+10
[1] 11 12 13 14 15 16
```



Simple operations with R

```
> M <- matrix(v,nrow=4,ncol=2)
> M
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
> t(M)%*%M
     [,1] [,2]
[1,]   30   70
[2,]   70  174
> solve(t(M)%*%M)
         [,1]     [,2]
[1,]  0.54375 -0.21875
[2,] -0.21875  0.09375
```



Remark : `solve(A,B)` return matrix `X` solution of $\boldsymbol{AX} = \boldsymbol{B}$.

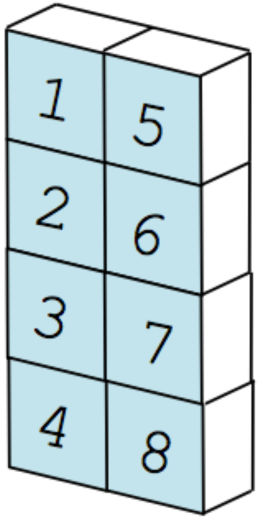
Simple operations with R

A matrix (or an array) is a rectangular collection of elements of the same type.

One should keep in mind that R is vector based, not matrix based,

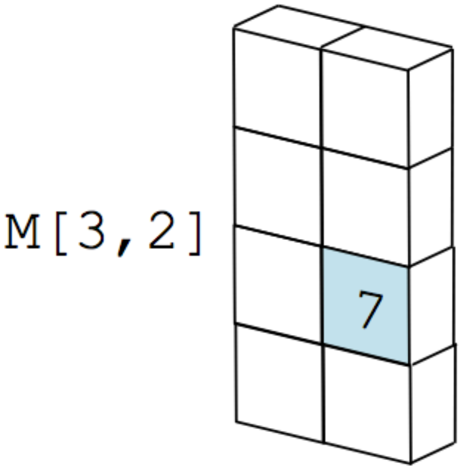
```
> M^2
     [,1] [,2]
[1,]    1   25
[2,]    4   36
[3,]    9   49
[4,]   16   64
```

M



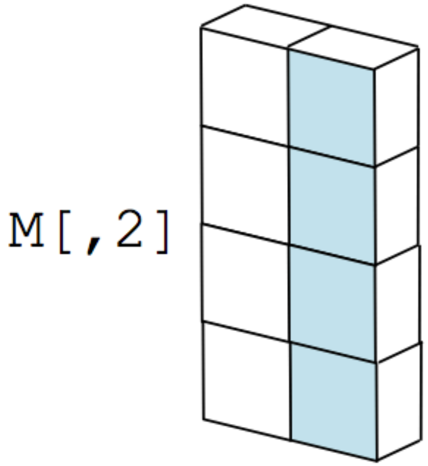
Simple operations with R

```
> M
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
> M[3,2]
[1] 7
> M==7
      [,1]  [,2]
[1,] FALSE FALSE
[2,] FALSE FALSE
[3,] FALSE  TRUE
[4,] FALSE FALSE
> which(M^2 > 10)
[1] 4 5 6 7 8
```



Simple operations with R

```
> M[,2]
[1] 5 6 7 8
```

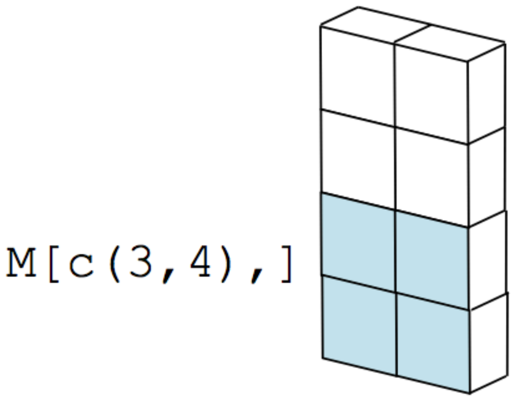


It is possible to use `rbind(...)` or `cbind(...)` to bind elements together, as columns or as rows

```
> N <- cbind(M,12:15)
> N
     [,1] [,2] [,3]
[1,]    1    5   12
[2,]    2    6   13
[3,]    3    7   14
[4,]    4    8   15
```

Simple operations with R

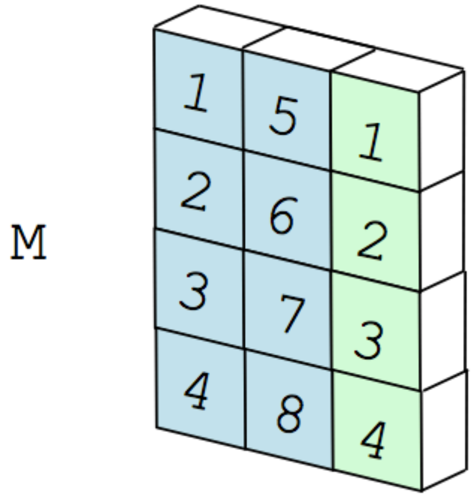
```
> M[c(3,4),]
     [,1] [,2]
[1,]    3    7
[2,]    4    8
> M[,1]<3
[1]  TRUE  TRUE FALSE FALSE
> M[M[,1]<3,]
     [,1] [,2]
[1,]    1    5
[2,]    2    6
```



Simple operations with R

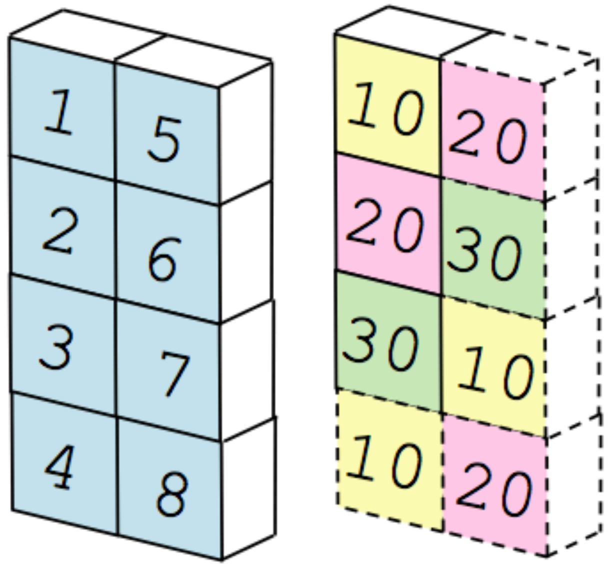
Remark : keep in mind that R has his recycling rule

```
> M <- matrix(v,nrow=4,ncol=3,byrow=FALSE)
> Warning :
In matrix(v, nrow = 4, ncol = 3) :
data length [8] is not a sub-multiple or
multiple of the number of rows [3]
> M
     [,1] [,2] [,3]
[1,]    1    5    1
[2,]    2    6    2
[3,]    3    7    3
[4,]    4    8    4
```



Simple operations with R

Remark : the recycling rule applies when adding a vector to a matrix (everything is a vector)

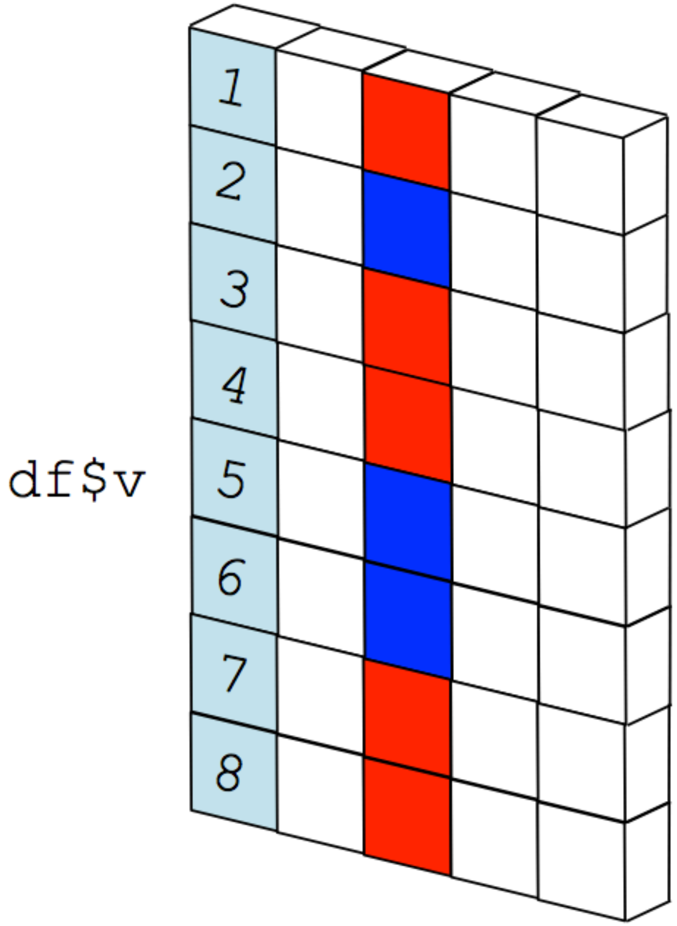



```
> M+c(10,20,30)
     [,1] [,2]
[1,]   11   25
[2,]   22   36
[3,]   33   17
[4,]   14   28
Warning :
In M + c(10, 20, 30) :
  longer object length is not a multiple of shorter object length
```

Simple operations with R

One can also define data frames

```
> set.seed(1)
> df <- data.frame(v,x=runif(8),n)
> df
  v         x n
1 1 0.2655087 R
2 2 0.3721239 B
3 3 0.5728534 R
4 4 0.9082078 R
5 5 0.2016819 B
6 6 0.8983897 B
7 7 0.9446753 R
8 8 0.6607978 R
> df$v
[1] 1 2 3 4 5 6 7 8
> df$x[1:3]
[1] 0.2655087 0.3721239 0.5728534
```



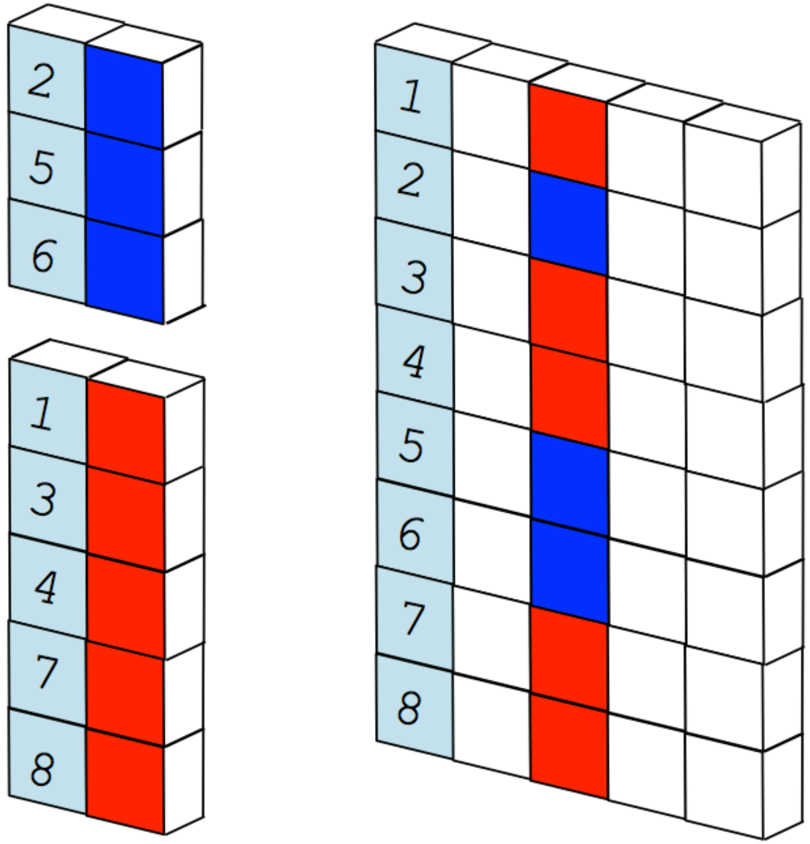
Simple operations with R

Each table has a unique name, each column within this table has a unique name, and each column has a unique type associated with it (a column is a vector).

```
> set.seed(1)
> df <- data.frame(v,x=runif(8),n)
> df
  v         x n
1 1 0.2655087 R
2 2 0.3721239 B
3 3 0.5728534 R
4 4 0.9082078 R
5 5 0.2016819 B
6 6 0.8983897 B
7 7 0.9446753 R
8 8 0.6607978 R
> df$v
[1] 1 2 3 4 5 6 7 8
> df$x[1:3]
[1] 0.2655087 0.3721239 0.5728534
```

Simple operations with R

```
> df2=data.frame(v=1:4,n=rnorm(4),z=rep("E",4))
> df2
  v          n z
1 1  0.3295078 E
2 2 -0.8204684 E
3 3  0.4874291 E
4 4  0.7383247 E
> merge(df,df2,"v")
  v         x n.x        n.y z
1 1 0.2655087   R  0.3295078 E
2 2 0.3721239   B -0.8204684 E
3 3 0.5728534   R  0.4874291 E
4 4 0.9082078   R  0.7383247 E
```



Simple operations with R

```
> merge(df,df2,"v",all.x=TRUE)
  v         x n.x        n.y    z
1 1 0.2655087   R  0.3295078    E
2 2 0.3721239   B -0.8204684    E
3 3 0.5728534   R  0.4874291    E
4 4 0.9082078   R  0.7383247    E
5 5 0.2016819   B         NA <NA>
6 6 0.8983897   B         NA <NA>
7 7 0.9446753   R         NA <NA>
8 8 0.6607978   R         NA <NA>
```

Simple operations with R

Finally, the most important objects in R are probably lists

```
> stored <- list(matrice = M, dates = some.dates, nom = "Arthur")
> stored
$matrice
     [,1] [,2] [,3]
[1,]    1    5    1
[2,]    2    6    2
[3,]    3    7    3
[4,]    4    8    4

$dates
[1] "2012-10-16" "2012-11-19"

$nom
[1] "Arthur"
> names(stored)
[1] "matrice" "dates"   "nom"
```

Importing datasets in R (for Windows)

```
> getwd()
[1] "C:\\Documents and Settings\\user\\arthurcharpentier\\"
> setwd("C:\\Documents and Settings\\user\\arthurcharpentier\\R\\datasets\\")

> file <- "extremedatasince1899.csv"
> StormMax <- read.table(file,header=TRUE,sep=",")
> tail(StormMax,3)
       Yr Region     Wmax       sst sun        soi split naofl naogulf
2098 2009  Basin 90.00000 0.3189293 4.3 -0.6333333     1  1.52   -3.05
2099 2009     US 50.44100 0.3189293 4.3 -0.6333333     1  1.52   -3.05
2100 2009     US 65.28814 0.3189293 4.3 -0.6333333     1  1.52   -3.05
> file <- "/Users/arthurcharpentier/R/datasets/extremedatasince1899.csv"
> StormMax <- read.table(file,header=TRUE,sep=",")
```

Importing datasets in R (for Mac)

```
> getwd()
[1] "/Users/arthurcharpentier"
> setwd("/Users/arthurcharpentier/R/datasets/")

> file <- "extremedatasince1899.csv"
> StormMax <- read.table(file,header=TRUE,sep=",")
> tail(StormMax,3)
       Yr Region     Wmax       sst sun        soi split naofl naogulf
2098 2009  Basin  90.00000 0.3189293 4.3 -0.6333333     1  1.52   -3.05
2099 2009     US  50.44100 0.3189293 4.3 -0.6333333     1  1.52   -3.05
2100 2009     US  65.28814 0.3189293 4.3 -0.6333333     1  1.52   -3.05
> file <- "/Users/arthurcharpentier/R/datasets/extremedatasince1899.csv"
> StormMax <- read.table(file,header=TRUE,sep=",")
```

Importing datasets in R

```
> file <- "http://freakonometrics.free.fr/extremedatasince1899.csv"
> StormMax <- read.table(file,header=TRUE,sep=",")

> filezip <- "http://freakonometrics.free.fr/extremedatasince1899.zip"
> temp = tempfile()
> download.file(filezip,temp);
trying URL 'http://freakonometrics.free.fr/extremedatasince1899.zip'
Content type 'application/zip' length 21241 bytes (20 Kb)
opened URL
==================================================
downloaded 20 Kb

>  StormMax <- read.table(unz(temp, "extremedatasince1899.csv"),
+  sep=",",header=TRUE,encoding="latin1")
```

Importing datasets in R

```
> mycols <- rep("NULL",11)
> mycols[c(1,2,3)] <- NA
> StormMax <- read.table(file,header=TRUE,sep=",",colClasses=mycols)
> tail(StormMax,3)
       Yr Region      Wmax
2098 2009  Basin  90.00000
2099 2009     US  50.44100
2100 2009     US  65.28814

> install.packages("RODBC",dependencies=TRUE)
> library(RODBC)

> sheet <- "c:\\Documents and Settings\\user\\excelsheet.xls"
> connection <- odbcConnectExcel(sheet)
> spreadsheet <- sqlTables(connection)

> query <- paste("SELECT * FROM",spreadsheet$TABLE_NAME[1],sep=" ")
> result <- sqlQuery(connection,query)
```

Coding functions with R

The function to compute is here

$$f : (\boldsymbol{x} = [x_i], \boldsymbol{p} = [p_i], \boldsymbol{d} = [d_i]) \mapsto \sum_{i=1}^{n} \frac{p_i \cdot x_i}{(1+d_i)^i}$$

We will define a function `f` with arguments vectors `x`, `p` and `d`

```
> f <- function(x,p,d){
+ s <- sum(p*x/(1+d)^(1:length(x)))
+ return(s)
+ }
```

Remark a statement of the form `d=0.05` will specify default values for that argument.

Remark functions always return values, either explicitly using `return(...)` or implicitly (using the last expression evaluated)

Coding functions with R

To call that function, the syntax is the same as R core functions,

```
> f(x=c(100,200,100),p=c(.4,.5,.3),d=.05)
[1] 154.7133
```

or equivalently

```
> f(c(100,200,100),c(.4,.5,.3),.05)
[1] 154.7133
```

Most R have default parameters, e.g.

```
> qnorm(.95)
[1] 1.644854
```

To get quantiles of a $\mathcal{N}(\mu, \sigma^2)$ distribution we use

```
> qnorm(.95,mean=1,sd=2)
[1] 4.289707
```

Side effects with R

R makes copies of the data supplied to a functions, i.e. operations that take place in the body of the function won't change original data (the so-called pass-by-value semantics, as opposed to passed-by-reference construction)

```
> s <- 0
> f <- function(x,p,d=.05){
+ s <- sum(p*x/(1+d)^(1:length(x)))
+ return(s)
+ }
> f(c(100,200,100),c(.4,.5,.3),.05)
[1] 154.7133
> s
[1] 0
```

Variables defined in the body of the function are local to that function.

Conditional evaluation : `if(...)`

The basic syntax is

```
if (condition1) {
    statement 1
} else if (condition2) {
    statement 2
} else {
    statement 3
}
```

Remark The `else` clause is optional here.

Loops with `for(...)` and `while(...)`

The basic syntaxes are here

```
for (variable in vector) {
    statement
}
```

and

```
while (condition) {
    statement
}
```

Remark : because many of R's operations are vectorized, you should think before you loop...

Functions within functions

One can define function within other functions

$$f : x \mapsto \frac{H(x)}{\int_x^\infty H(t)dt}$$

the code can be, if H is the survival function of the Gaussian distribution,

```
> f <- function(x,m=0,s=1){
+  H<-function(t) 1-pnorm(t,m,s)
+  integral<-integrate(H,lower=x,upper=Inf)$value
+  res<-H(x)/integral
+  return(res)
+ }
> f(0)
[1] 1.253314
```

The argument of function f is not a vector. If we want to compute $f(x_i)$ for some x_i's, one should vectorize the function

```
> f(x <- 0:1)
[1] 1.2533141 0.3976897
Warning :
In if (is.finite(lower)) { :
  the condition has length > 1 and only the first element will be used
> Vectorize(f)(x)
[1] 1.253314 1.904271
```

Remark : one can also use a loop (mentioned earlier)

```
> y <- rep(NA,2)
> x <- 0:1
> for(i in 1:2) y[i] <- f(x[i])
> y
[1] 1.253314 1.904271
```

Remark :one can also use the `sapply(...)` function (we'll also come back on that)

```
> y <- sapply(x,"f")
> y
[1] 1.253314 1.904271
```

More on functions $\mathbb{R}^d \to \mathbb{R}$

Consider now the joint density of the $\mathcal{N}(\mathbf{0}, \Sigma)$ distribution,

$$\varphi(x,y) = \frac{1}{2\pi\sqrt{1-\rho^2}} \exp\left(-\frac{1}{2(1-\rho^2)}\left[x^2+y^2-2\rho xy\right]\right), \forall x, y \in \mathbb{R}^2.$$

```
> binorm <- function(x1,x2,r=0){
+ exp(-(x1^2+x2^2-2*r*x1*x2)/(2*(1-r^2)))/(2*pi*sqrt(1-r^2))
+ }
```

Given vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, $\varphi(\boldsymbol{u}, \boldsymbol{v})$ is the vector $[\varphi(u_i, v_i)]$,

```
> u <- seq(-2,2)
> binorm(u,u)
[1] 0.002915024 0.058549832 0.159154943 0.058549832 0.002915024
```

More on functions $\mathbb{R}^d \to \mathbb{R}$

To compute the matrix $[\varphi(u_i, v_i)]$ use

```
> outer(u,u,binorm)
            [,1]       [,2]       [,3]       [,4]        [,5]
[1,] 0.002915024 0.01306423 0.02153928 0.01306423 0.002915024
[2,] 0.013064233 0.05854983 0.09653235 0.05854983 0.013064233
[3,] 0.021539279 0.09653235 0.15915494 0.09653235 0.021539279
[4,] 0.013064233 0.05854983 0.09653235 0.05854983 0.013064233
[5,] 0.002915024 0.01306423 0.02153928 0.01306423 0.002915024
```

Coding actuarial and functional functions with R

```
> alive <- read.table("http://freakonometrics.free.fr/TV8890.csv",header=TRUE,sep=";")$Lx
> alive[1:3]
[1] 100000  99352  99294
> death <- -diff(alive)
> death[1:3]
[1] 648  58  33
```

A standard mortality law is the one suggested by Makeham, with survival probability function

$$S(x) = \exp\left(-ax - \frac{b}{\log c}[c^x - 1]\right), \forall x \geq 0,$$

for some parameter $a \geq 0, b \geq 0$ and $c > 1$.

The R function to compute this function can be defined as

```
> sMakeham <- function(x,a,b,c){ ifelse(x<0,1,exp(-a*x-b/log(c)*(c^x-1))) }
```

Function `ifselse` can be used, to be sure that $S(x) = 1$ if $x < 0$. The probability function associated to this survival function can be computed as

```
> dMakeham <- function(x,a,b,c){ ifelse(x>floor(x),0,sMakeham(x,a,b,c)-sMakeham(x+1,a,b,c)
```

Based on that function, it is possible to use standard maximum likelihood techniques to estimate those parameters, based on the sample where death at birth are removed (this feature cannot be obtained using Makeham's distribution), as well as above 105,

```
> death <- death[-1]
> ages <- 1:(length(death))
> loglikMakeham <- function(abc){
- sum(log(dMakeham(ages,abc[1],abc[2],abc[3]))*death[ages])
+ }
```

The `optim` function can be used to obtain maximum likelihood estimators for parameters in Makeham's survival function (assuming that we can find adequate starting values for the algorithm)

```
> mlEstim <- optim(c(1e-5,1e-4,1.1),loglikMakeham)
> abcml <- mlEstim$par
```

Based on observed ages of deaths, it is possible to compute the average age-at-death

```
> sum((ages+.5)*death)/sum(death)
[1] 80.69235
```

which can be compare to the one obtained using Makeham's survival function

```
> integrate(sMakeham,0,Inf,abcml[1],abcml[2],abcml[3])
81.1661 with absolute error < 0.0032
```

Consider the expected discounted value of capital given if some insured is alive, i.e.

$$\sum_{k=1}^{n} \frac{C_k}{(1+i)^k} \mathbb{P}(T > x+k | T > x)$$

```
> f <- function(i,age,capital){
+ n <- length(capital)
+ capital.act <- capital*(1/(1+i))^(1:n)
+ probability <- alive[age+1+1:n]/alive[age+1]
+ return(sum(capital.act*probability))}

> g <- function(i) f(i,age = 45,capital = c(100,100,125,125,150,150))
> sum(c(100,100,125,125,150,150))
[1] 750
> g(.05)
[1] 621.3342
```

Let us now write a function which computes the actuarial discount rate, given some discounted value. A natural idea to find zeros would be to use the secant method (there is a `uniroot` function that searches roots)

```
> secant=function(fun, x0, x1, tolerence=1e-07, niter=500){
+ for ( i in 1:niter ) {
+         x2 <- x1-fun(x1)*(x1-x0)/(fun(x1)-fun(x0))
+         if (abs(fun(x2)) < tolerence)
```

```
+                   return(x2)
+           x0 <- x1
+           x1 <- x2
+ }}
```

There, we can write a function that searches $i_\star$ such that

$$\sum_{k=1}^{n} \frac{C_k}{(1+i_\star)^k} \mathbb{P}(T > x+k | T > x) = V$$

for some specific value of V.

```
> discount.rate = function(value,lower=0,upper=.1){
+ cat("With ",lower*100,"% interest rate, actuarial present value =",g(lower),"\n")
+ cat("With ",upper*100,"% interest rate, actuarial present value =",g(upper),"\n")
+ cat("Target value =",value,"\n")
+ f1=function(x){g(x)-value}
+ r=secant(f1,lower,upper)
+ cat("With ",r*100,"% interest rate, actuarial present value =",g(r),"\n")
+ return(r)
```

```
+ }

> discount.rate(600)
With  0 % interest rate, actuarial present value = 743.9027
With  10 % interest rate, actuarial present value = 526.6808
Target value = 600
With  6.022313 % interest rate, actuarial present value = 600
```

Programming efficiently in R

We want a function to generate random compound Poisson variables

$$S = X_1 + \cdots + X_N = \sum_{i=1}^{N} X_i, \text{ with } S = 0 \text{ if } N = 0.$$

Consider some specific distributions for N and X_i's.

```
> rN.Poisson <- function(n) rpois(n,5)
> rX.Exponential <- function(n) rexp(n,2)
```

A first (and natural idea) is to use a loop,

```
> rcpd1 <- function(n,rN=rN.Poisson,rX=rX.Exponential){
+ V <- rep(0,n)
+ for(i in 1:n){
+  N <- rN(1)
+  if(N>0){V[i] <- sum(rX(sum(N)))}
+ }
+ return(V)}
```

Programming efficiently in R

```
> set.seed(1)
> rcpd1(3)
[1] 0.9516067 1.9164197 2.5128117
```

Spltting and combining data.

Base function	`plyr` function	input	output
`aggregate`	`ddply`	data frame	data frame
`apply`	`aaply` (or `alply`)	array	array (or list)
`by`	`dlply`	data frame	list
`lapply`	`llply`	list	list
`mapply`	`maply` (or `mlply`)	array	array (or list)
`sapply`	`laply`	list	array

Programming efficiently in R

Remark there is also an `xtabs` function which computes sums of a specific vector given a factor

```
> v
[1] 1 2 3 4 5 6 7 8
> n
[1] R B R R B B R R
Levels: B R
> xtabs(v~n)
n
 B  R
13 23
```

With this function we get

```
> rcpd2 <- function(n,rN=rN.Poisson,rX=rX.Exponential){
+ N <- rN(n)
+ X <- rX(sum(N))
```

```
+ I <- factor(rep(1:n,N),levels=1:n)
+ return(as.numeric(xtabs(X ~ I)))}
```

The `tapply` can be used to compute any function (not only a sum)

```
> tapply(v,n,sum)
 B  R
13 23
```

Here, the code becomes

```
> rcpd3 <- function(n,rN=rN.Poisson,rX=rX.Exponential){
+ N <- rN(n)
+ X <- rX(sum(N))
+ I <- factor(rep(1:n,N),levels=1:n)
+ V <- tapply(X,I,sum)
+ V[is.na(V)] <- 0
+ return(as.numeric(V))}
```

To write more efficient functions, one can also `sapply` (mentioned earlier)

```
> rcpd4 <- function(n,rN=rN.Poisson,rX=rX.Exponential){
+  return(sapply(rN(n), function(x) sum(rX(x))))}
```

(or similarly - but the output will be a list)

```
> rcpd5 <- function(n,rN=rN.Poisson,rX=rX.Exponential){
+  return(unlist(lapply(lapply(t(rN(n)),rX),sum)))}
```

If we compare the efficiency of the codes, we get

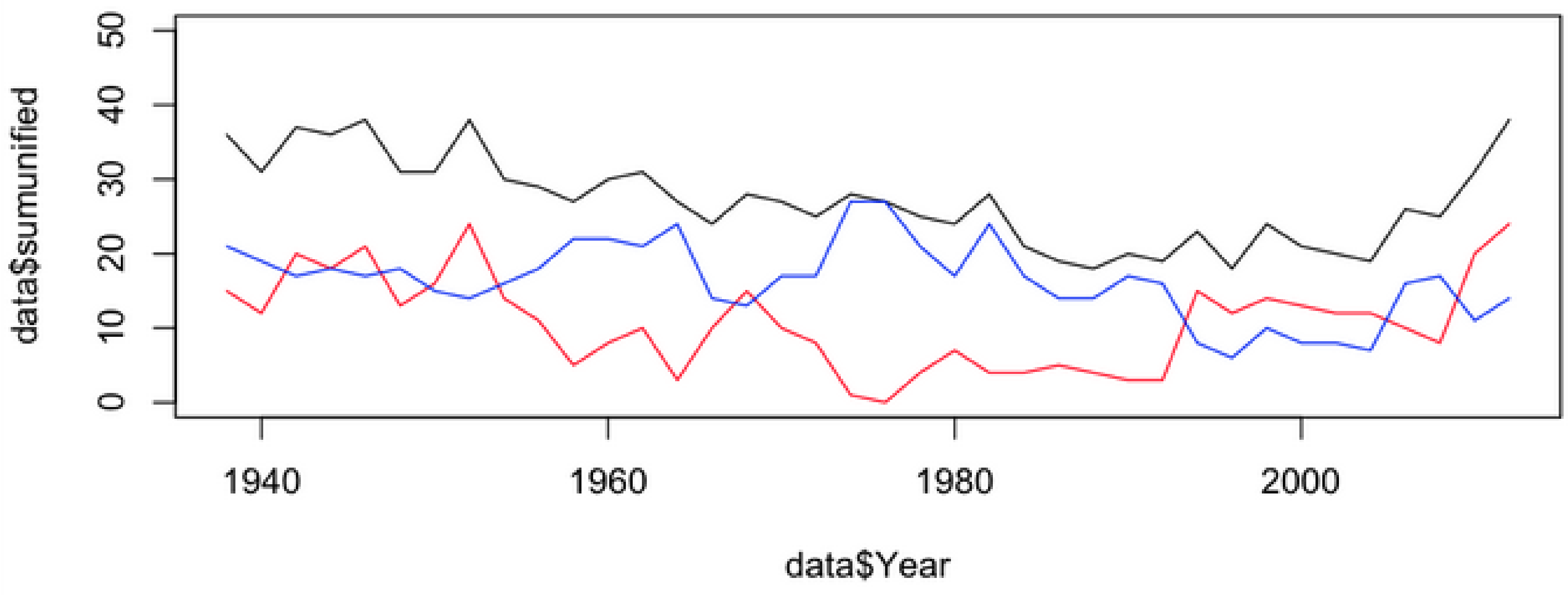
```
> library(microbenchmark)
> microbenchmark(rcpd1(n),rcpd2(n),rcpd3(n),rcpd4(n),rcpd5(n),times=1000)
Unit: microseconds
      expr     min       lq    median        uq        max
1 rcpd1(n) 232.447 273.6440  292.1250  324.5540   1961.836
2 rcpd2(n) 819.559 939.8615 1011.5625 1097.9500  25930.627
3 rcpd3(n) 303.330 347.7800  372.8095  411.5855   2439.751
4 rcpd4(n) 136.361 154.8520  166.3905  185.8655 107501.625
5 rcpd5(n) 119.019 138.3705  148.4900  164.4135  30560.373
```

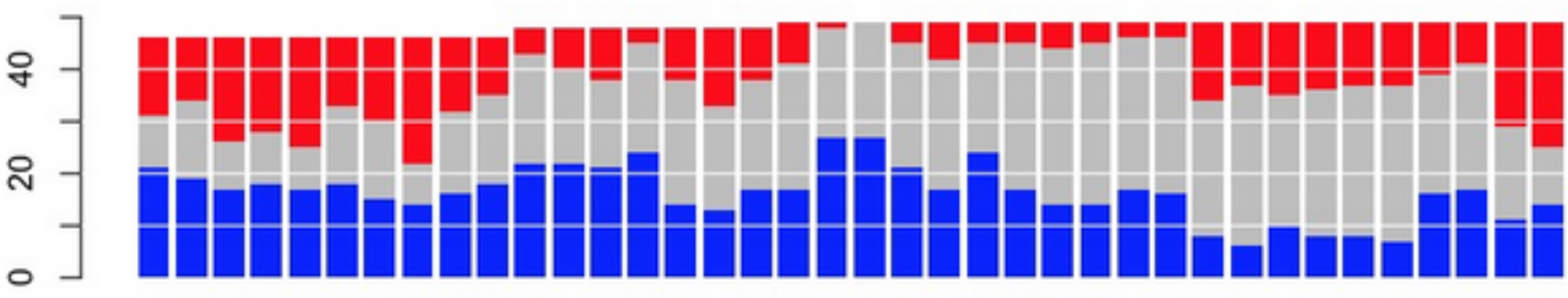
Graphs with R

"*If you can picture it in your head, chances are good that you can make it work in R. R makes it easy to read data, generate lines and points, and place them where you want them. It's very flexible and super quick. When you've only got two or three hours until deadline, R can be brilliant.*" Amanda Cox, a graphics editor at the New York Times. "*R is particularly valuable in deadline situations when data is scant and time is precious.*".

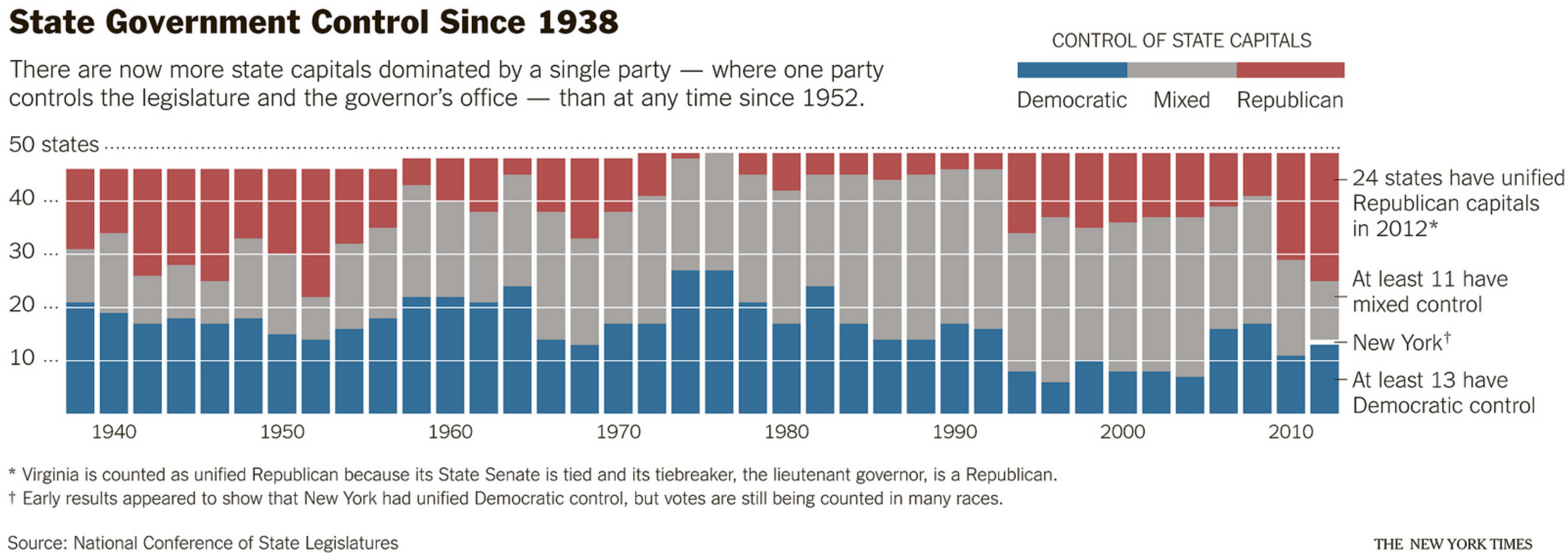
Source : http ://chartsnthings.tumblr.com/post/36978271916/r-tutorial-simple-charts

Graphs, R and The New York Times

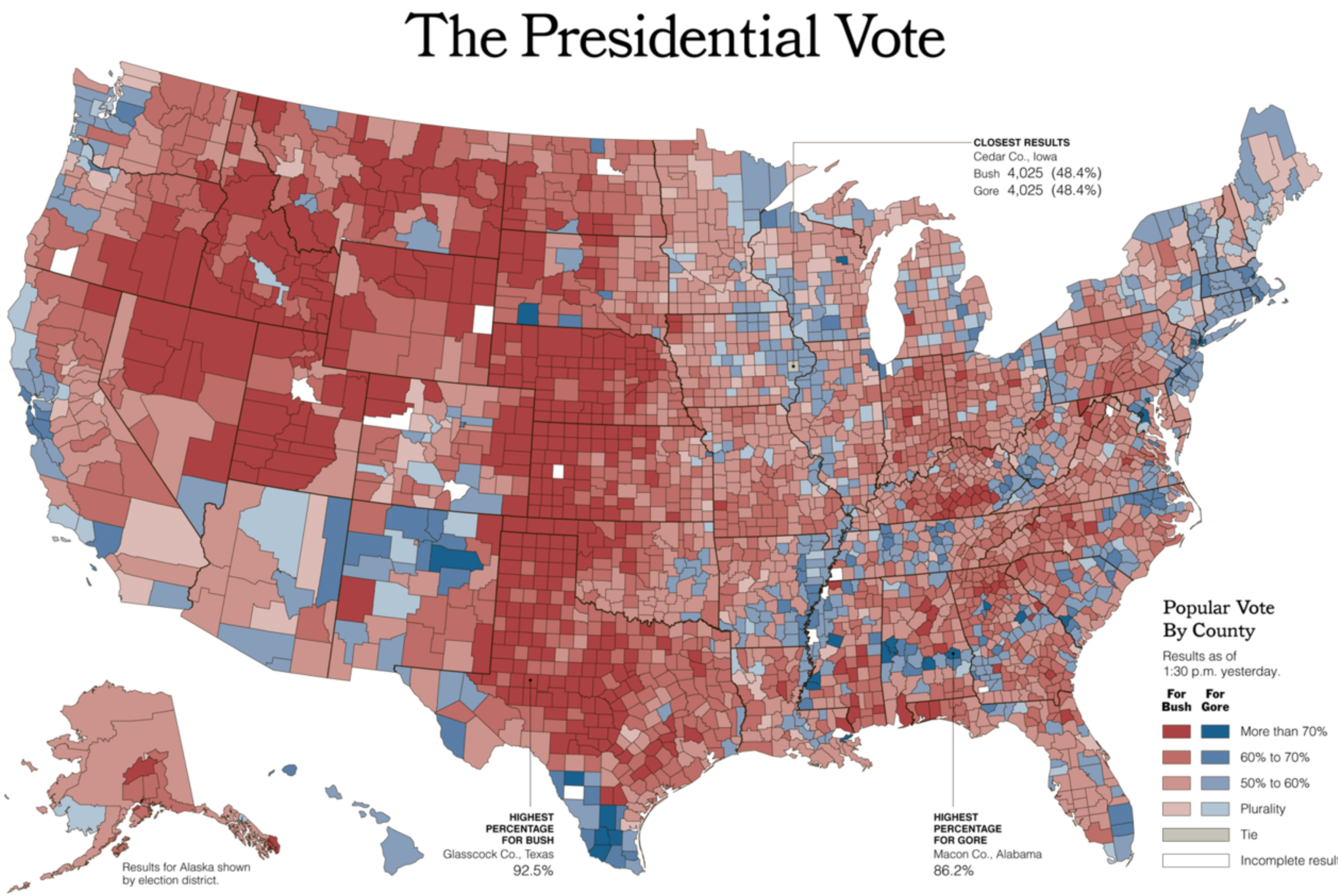




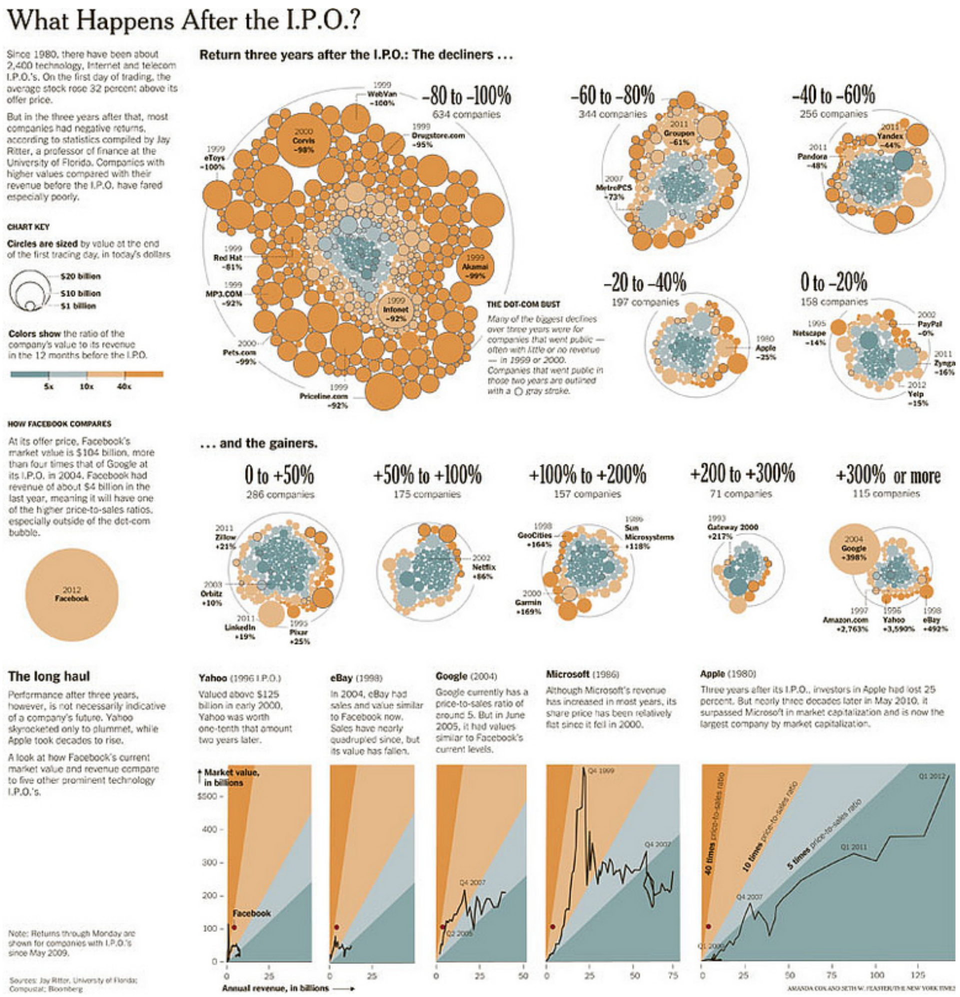
Graphs, R and The New York Times

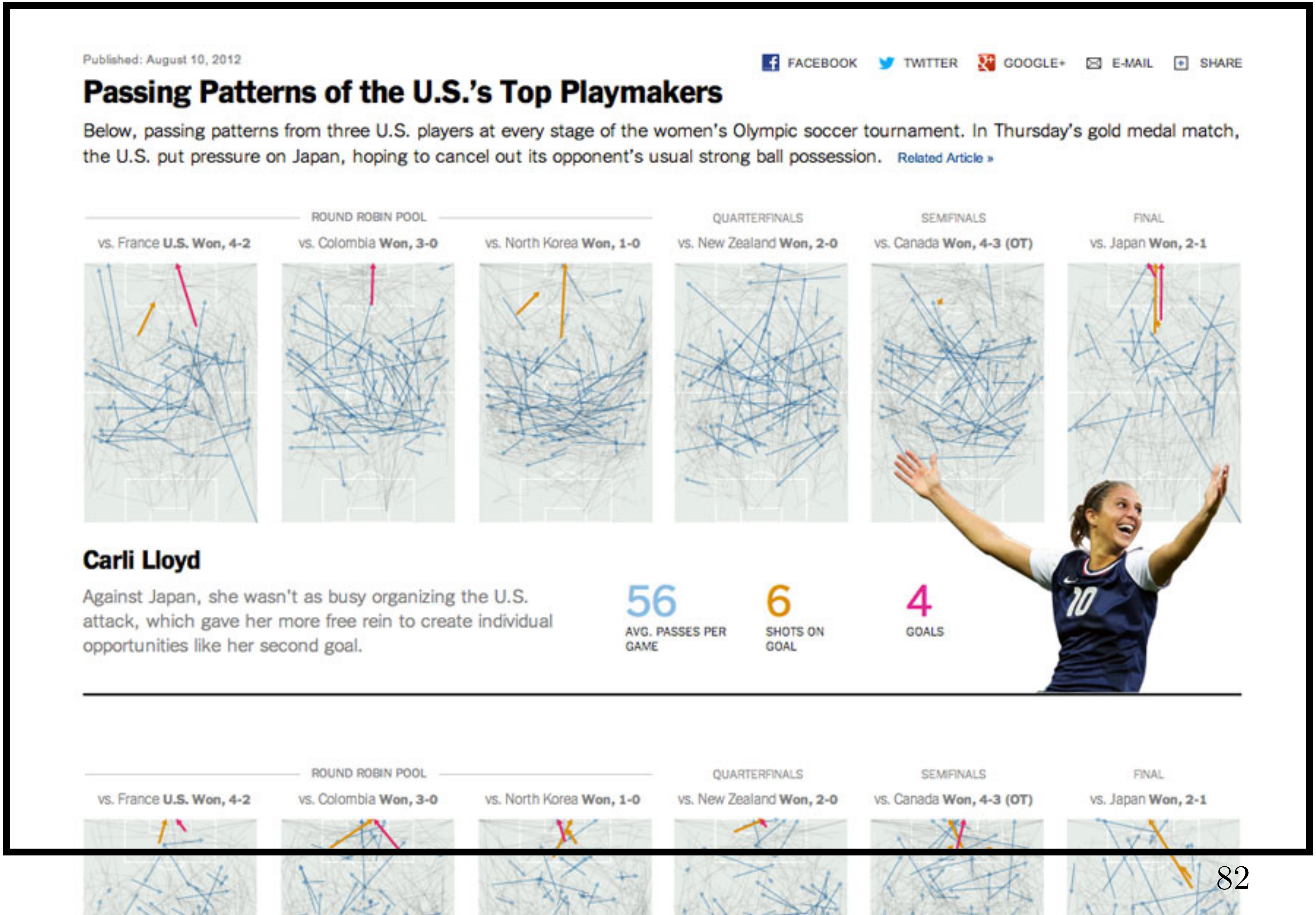


Graphs, R and The New York Times



Graphs, R and The New York Times




Published: August 10, 2012
FACEBOOK TWITTER GOOGLE+ E-MAIL SHARE
Passing Patterns of the U.S.'s Top Playmakers
Below, passing patterns from three U.S. players at every stage of the women's Olympic soccer tournament. In Thursday's gold medal match, the U.S. put pressure on Japan, hoping to cancel out its opponent's usual strong ball possession. Related Article »
ROUND ROBIN POOL
QUARTERFINALS
SEMIFINALS
FINAL
vs. France U.S. Won, 4-2
vs. Colombia Won, 3-0
vs. North Korea Won, 1-0
vs. New Zealand Won, 2-0
vs. Canada Won, 4-3 (OT)
vs. Japan Won, 2-1
Carli Lloyd
Against Japan, she wasn't as busy organizing the U.S. attack, which gave her more free rein to create individual opportunities like her second goal.
56
AVG. PASSES PER GAME
6
SHOTS ON GOAL
4
GOALS
ROUND ROBIN POOL
QUARTERFINALS
SEMIFINALS
FINAL
vs. France U.S. Won, 4-2
vs. Colombia Won, 3-0
vs. North Korea Won, 1-0
vs. New Zealand Won, 2-0
vs. Canada Won, 4-3 (OT)
vs. Japan Won, 2-1

Graphs in financial and actuarial communication

"*It's not just about producing graphics for publication. It's about playing around and making a bunch of graphics that help you explore your data. This kind of graphical analysis is a really useful way to help you understand what you're dealing with, because if you can't see it, you can't really understand it. But when you start graphing it out, you can really see what you've got.*" Peter Aldhous, San Francisco bureau chief of New Scientist magazine.

"*The commercial insurance underwriting process was rigorous but also quite subjective and based on intuition. R enables us to communicate our analytic results in appealing and innovative ways to non-technical audiences through rapid development lifecycles. R helps us show our clients how they can improve their processes and effectiveness by enabling our consultants to conduct analyses efficiently*". John Lucker, team of advanced analytics professionals at Deloitte Consulting Principal.

see also Gelman (2011).

Graphics in R

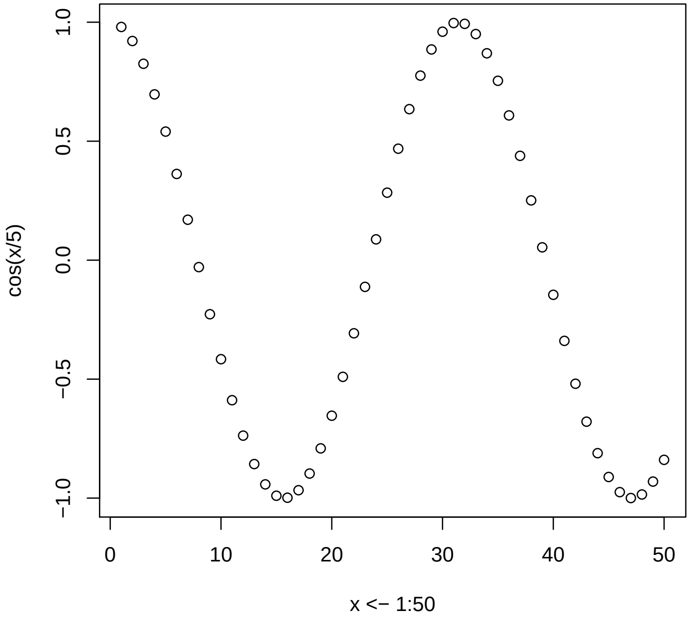
When working with R iteractively (i.e. typing commands into the interpreter), graphics output appears in a separate window.

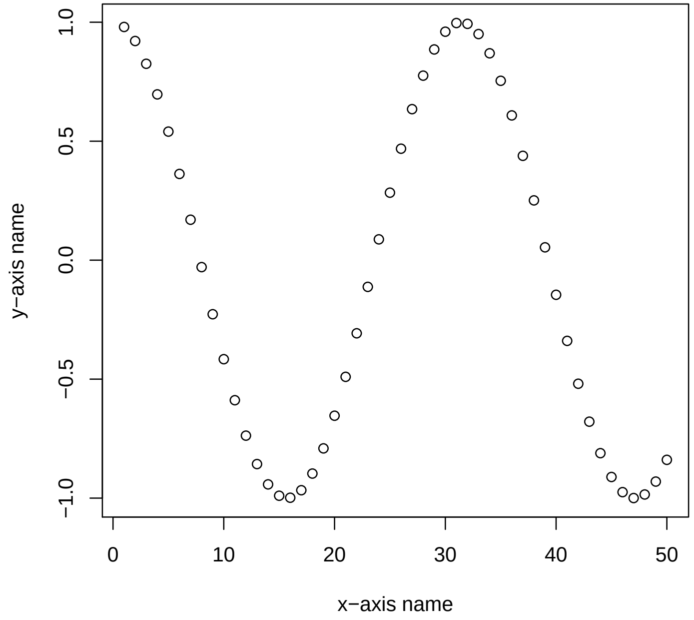
Remark : it is possible to catch the output into a file, (pdf, png, jpeg, etc).

As always, functions that produce graphical output rely on a series of arguments, e.g. `xlab` or `ylab)` for labels on the x and y axies, `xlim` and `xaxs` for bounds, and ranges for the x axis, `lty` for the type of line used, `pch` for the plotting character, and `col` for the color.

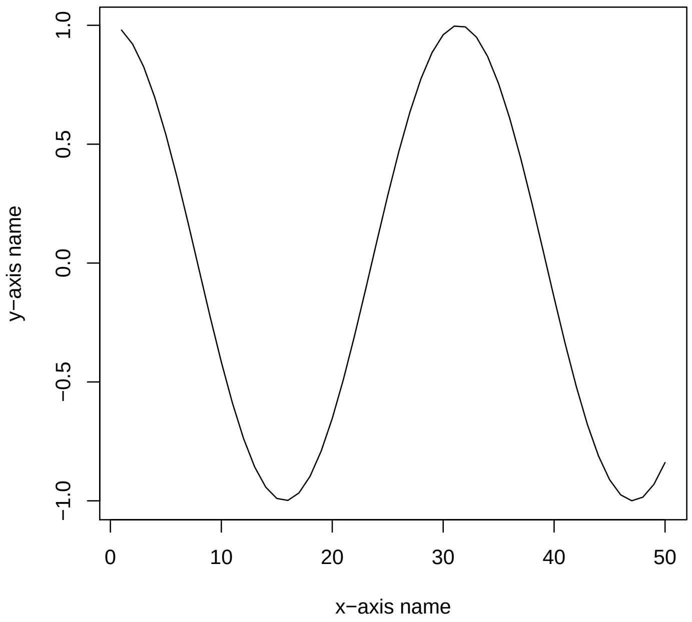
```
> x <- 1:50
> y <- cos(x/5)
> plot(x,y)
```

```
> plot(x<-1:50,cos(x/5),xlab="x-axis name",
+ ylab="y-axis name")
```

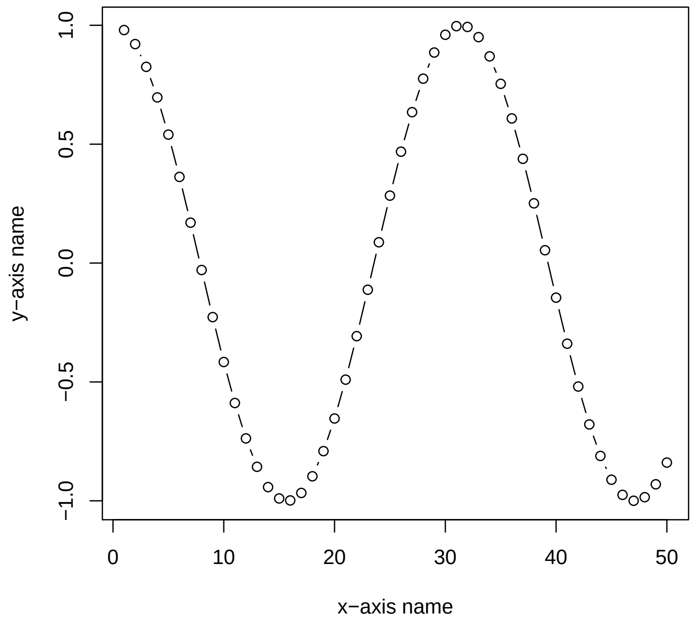




```
> plot(x<-1:50,cos(x/5),xlab="x-axis name",
+ ylab="y-axis name",type="l")
```

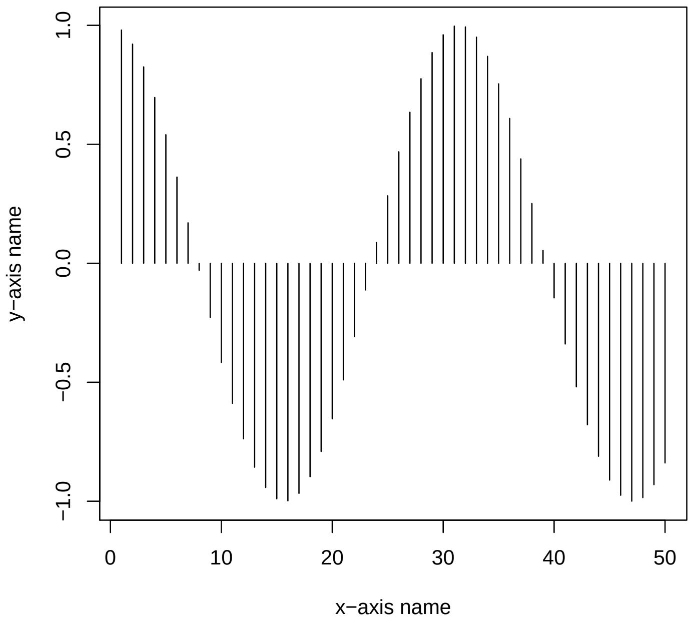


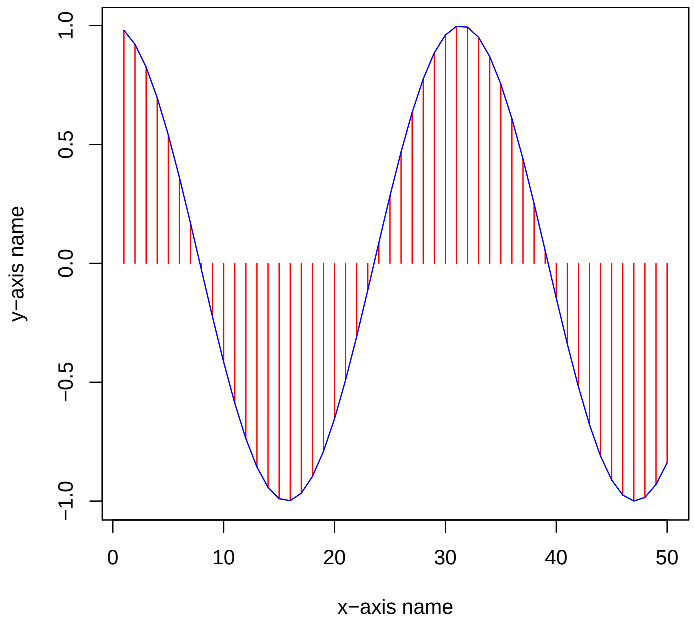

```
> plot(x<-1:50,cos(x/5),xlab="x-axis name"
+ ylab="y-axis name",type="b")
```



```
> plot(x<-1:50,cos(x/5),xlab="x-axis name",
+ ylab="y-axis name",type="h")
```

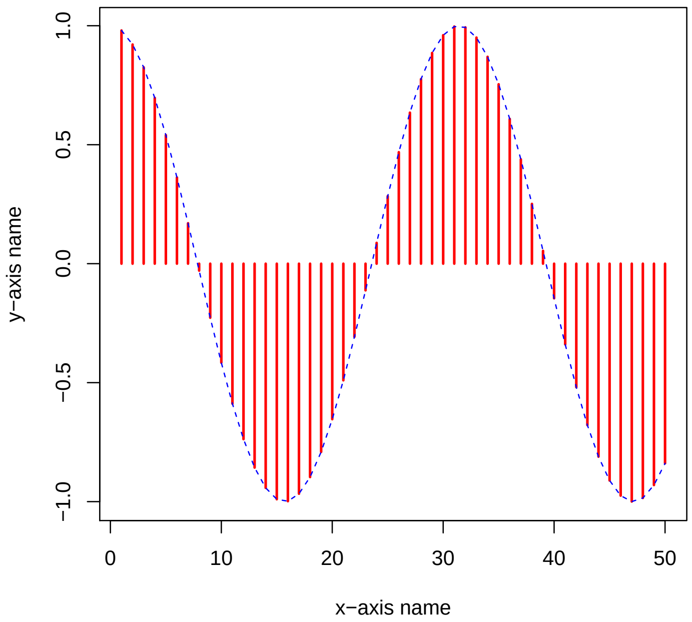
```
> plot(x<-1:50,cos(x/5),xlab="x-axis name",
+ ylab="y-axis name",type="h",col="red")
+ lines(x,cos(x/5),col="blue")
```

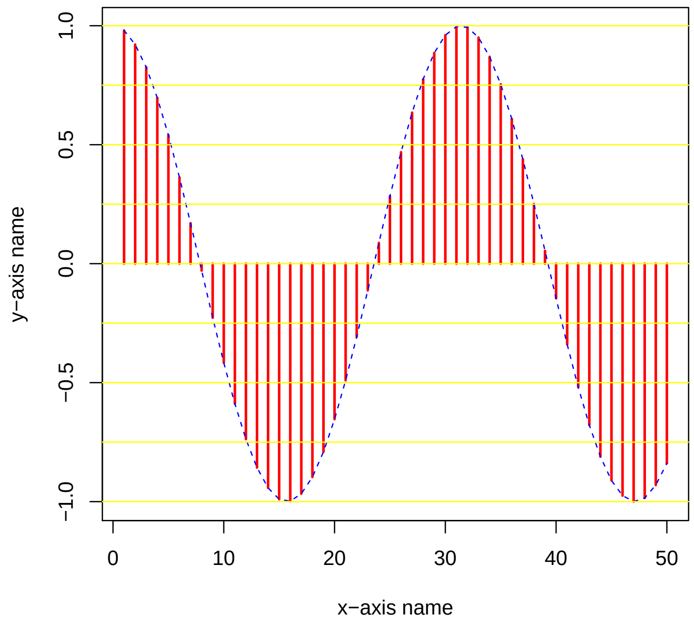




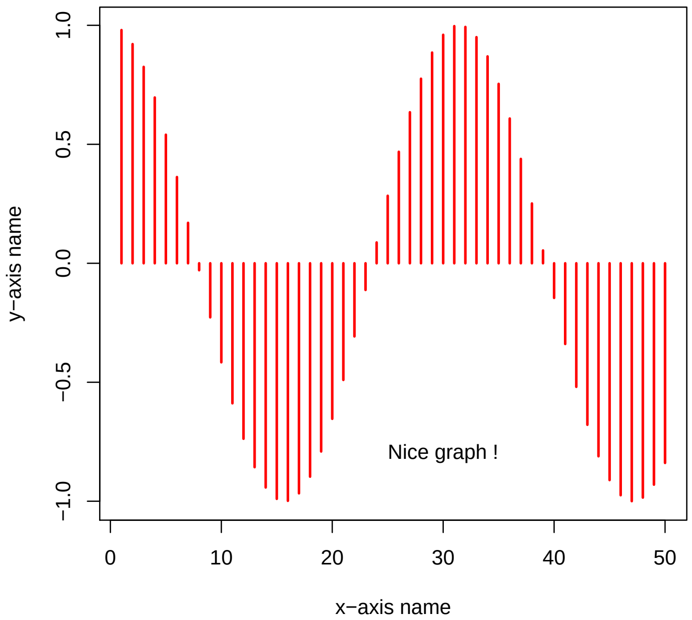
```
> plot(x<-1:50,cos(x/5),xlab="x-axis name",
+ ylab="y-axis name",type="h",col="red",lwd=2)
+ lines(x,cos(x/5),col="blue",lty=2)
```

```
> plot(x<-1:50,cos(x/5),xlab="x-axis name",
+ ylab="y-axis name",type="h",col="red",lwd
+ lines(x,cos(x/5),col="blue",lty=2)
+ abline(h=seq(-1,1,by=.25),col="yellow")
```

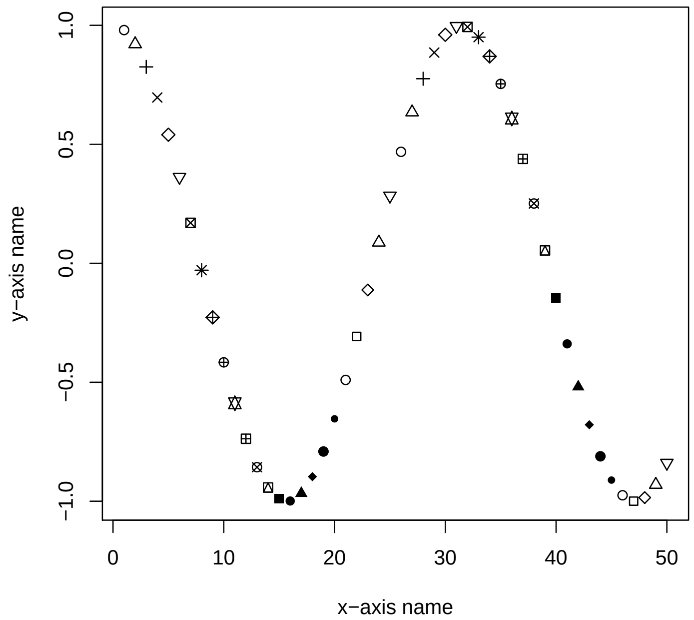




```
> plot(x<-1:50,cos(x/5),xlab="x-axis name",
+ ylab="y-axis name",type="h",col="red",lwd=2)
> text(30,-.8,"Nice graph !")
```

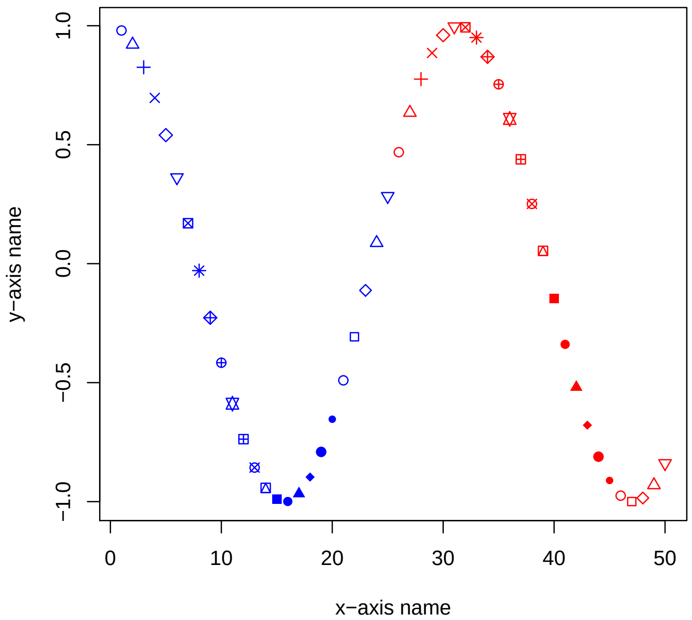


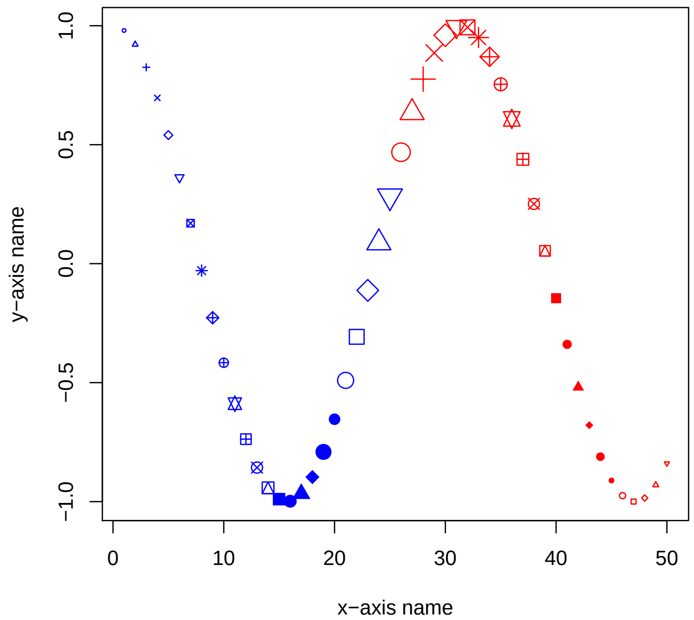

```
> plot(x<-1:50,cos(x/5),xlab="x-axis name",
+ ylab="y-axis name",type="p",pch=rep(1:25,
```



```
> plot(x<-1:50,cos(x/5),xlab="x-axis name",
+ ylab="y-axis name",type="p",pch=rep(1:25,2),
+ col=rep(c("blue","red"),each=25))
```

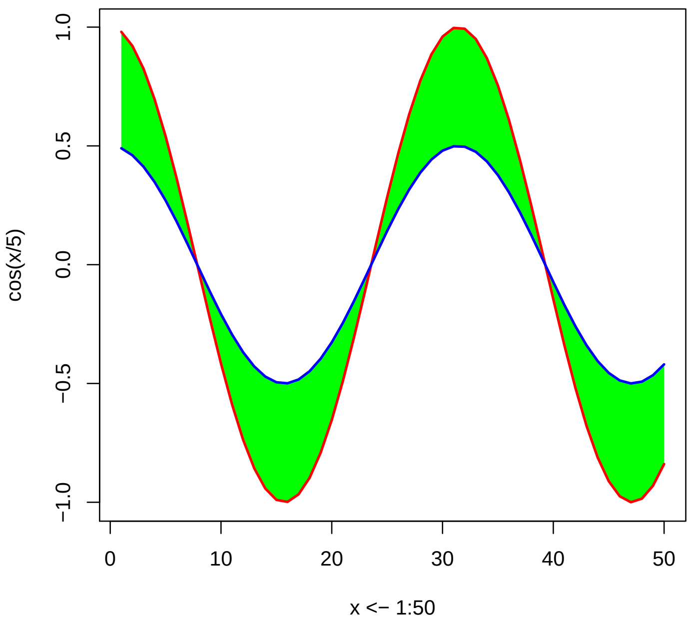
```
> plot(x<-1:50,cos(x/5),xlab="x-axis name",
+ ylab="y-axis name",type="p",pch=rep(1:25,
+ col=rep(c("blue","red"),each=25),
+ cex=c(seq(.4,2,length=25),seq(2,.4,length
```

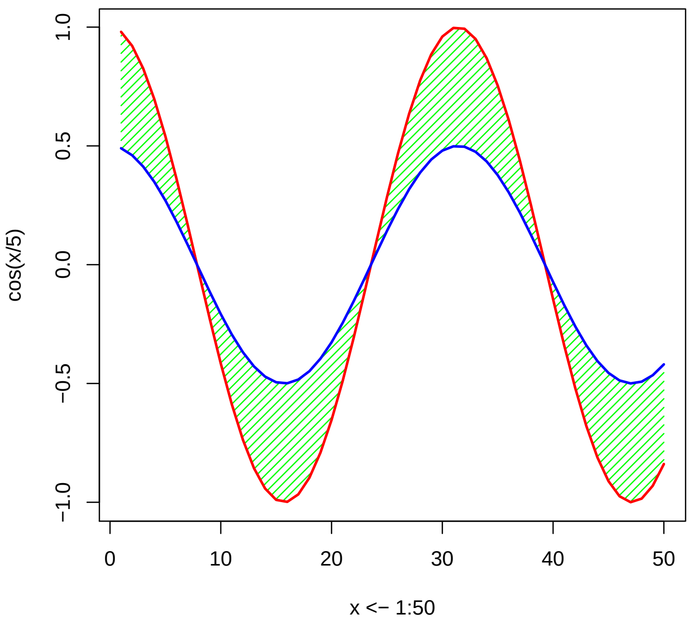




```
> plot(x<-1:50,cos(x/5),col="white")
> u <- c(x,rev(x))
> v <- c(cos(x/5),rev(cos(x/5)/2))
> polygon(u,v,col="green",border=NA)
> lines(x,cos(x/5),lwd=2,col="red")
> lines(x,cos(x/5)/2,lwd=2,col="blue")
```

```
> plot(x<-1:50,cos(x/5),col="white")
> u <- c(x,rev(x))
> v <- c(cos(x/5),rev(cos(x/5)/2))
> polygon(u,v,col="green",border=NA,density=20)
> lines(x,cos(x/5),lwd=2,col="red")
> lines(x,cos(x/5)/2,lwd=2,col="blue")
```





Displaying colors in graphs

Colors can be specified by names, e.g. `blue` or `light green`, see

```
> colors()
  [1] "white"           "aliceblue"       "antiquewhite"
  [4] "antiquewhite1"   "antiquewhite2"   "antiquewhite3"
  [7] "antiquewhite4"   "aquamarine"      "aquamarine1"
 [10] "aquamarine2"     "aquamarine3"     "aquamarine4"
 [13] "azure"           "azure1"          "azure2"
 [16] "azure3"          "azure4"          "beige"
 [19] "bisque"          "bisque1"         "bisque2"
 [22] "bisque3"         "bisque4"         "black"
```

(etc). One can also use RGB values, using function `rgb()`.

Displaying colors in graphs (palettes)

For sequential palettes, one can use `heat.colors()`

```
> cl <- heat.colors(n<-8)
> plot((1:n)/(n+1),rep(.5,n),col=cl[1:n],pch=15,cex=4,axes=FALSE)
```

and

```
> cl <- heat.colors(n<-50)
> plot((1:n)/(n+1),rep(.5,n),col=cl[1:n],pch=15,cex=4,axes=FALSE)
```

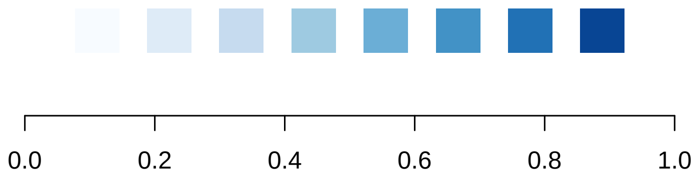
Displaying colors in graphs (palettes)

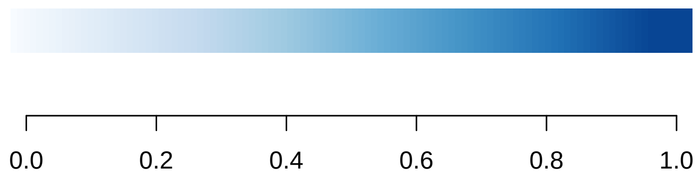
For sequential palettes, one can use `brewer.pal()` in `library(RColorBrewer)`

```
> library(RColorBrewer)
> cl <- brewer.pal(n<-8,"Blues")
> plot((1:n)/(n+1),rep(.5,n),col=cl[1:n],pch=15,cex=4,axes=FALSE)
```

and

```
> cl <- colorRampPalette(brewer.pal(8,"Blues"))(n<-100)
> plot((1:n)/(n+1),rep(.5,n),col=cl[1:n],pch=15,cex=4,axes=FALSE)
```





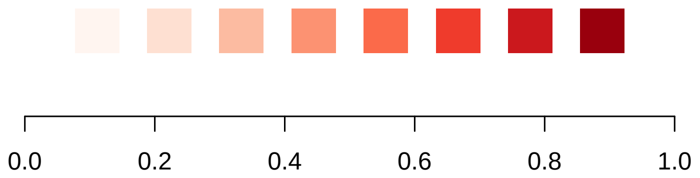
Displaying colors in graphs (palettes)

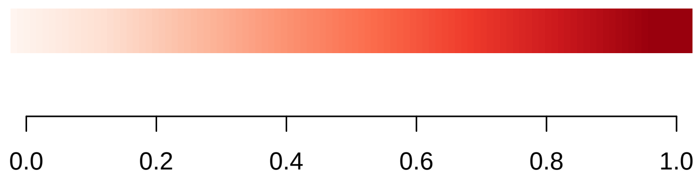
For sequential palettes, one can use `brewer.pal()` in `library(RColorBrewer)`

```
> library(RColorBrewer)
> cl <- brewer.pal(n<-8,"Reds")
> plot((1:n)/(n+1),rep(.5,n),col=cl[1:n],pch=15,cex=4,axes=FALSE)
```

and

```
> cl <- colorRampPalette(brewer.pal(8,"Reds"))(n<-100)
> plot((1:n)/(n+1),rep(.5,n),col=cl[1:n],pch=15,cex=4,axes=FALSE)
```





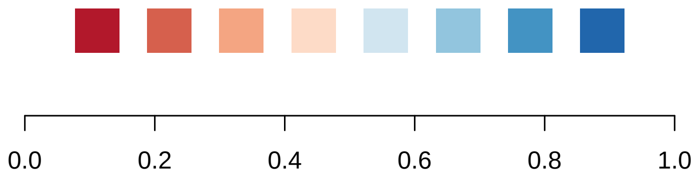
Displaying colors in graphs (palettes)

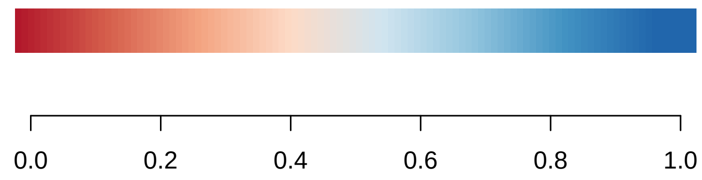
For sequential palettes, one can use `brewer.pal()` in `library(RColorBrewer)`

```
> library(RColorBrewer)
> cl <- brewer.pal(n<-8, "RdBu")
> plot((1:n)/(n+1),rep(.5,n),col=cl[1:n],pch=15,cex=4,axes=FALSE)
```

and

```
> cl <- colorRampPalette(brewer.pal(8, "RdBu"))(n<-100)
> plot((1:n)/(n+1),rep(.5,n),col=cl[1:n],pch=15,cex=4,axes=FALSE)
```

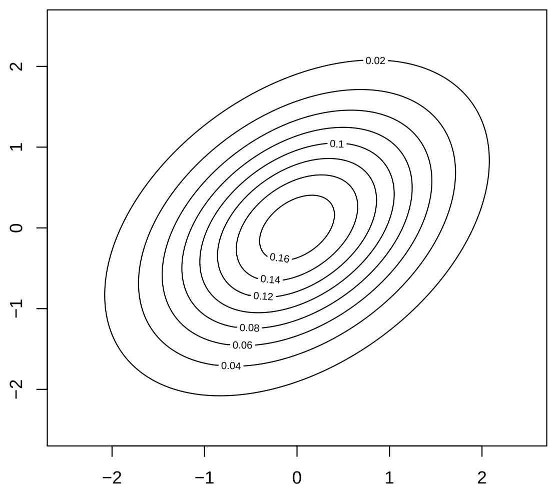




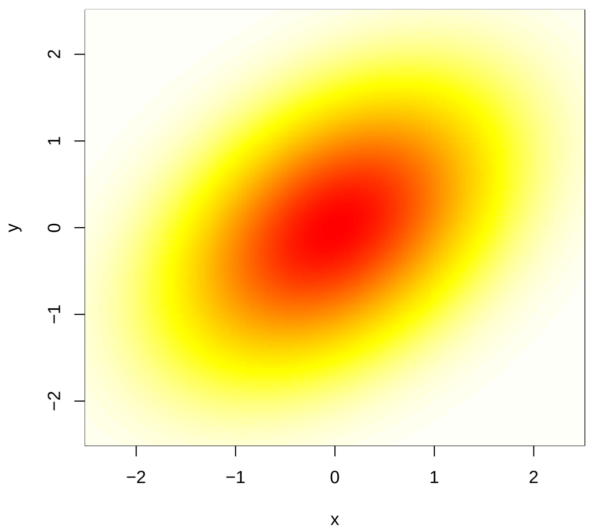
```
> x <- y <- seq(-2.5,2.5,by=.025)
> z <- outer(x,y,function(u,v) binorm(u,v,r=.4))
```

First, if we want to visualize only level curves

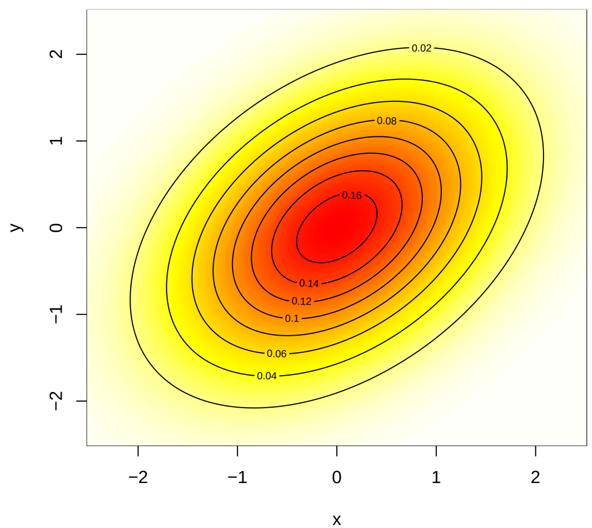
```
> contour(x,y,z)
```



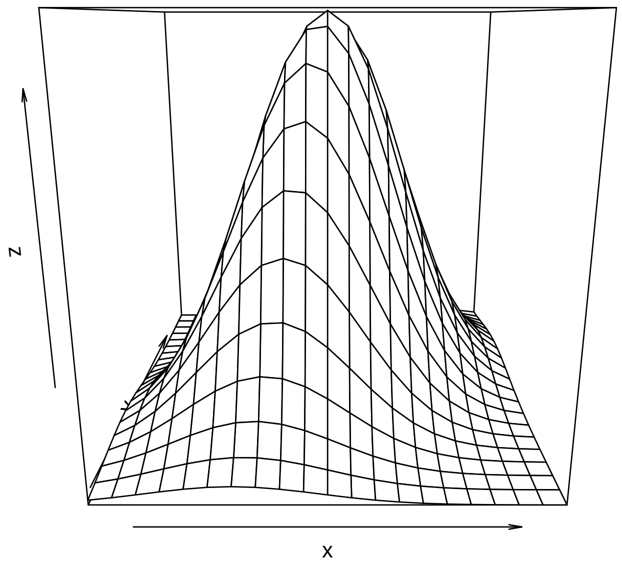

```
> image(x,y,z,col=
+ rev(heat.colors(101)))
```



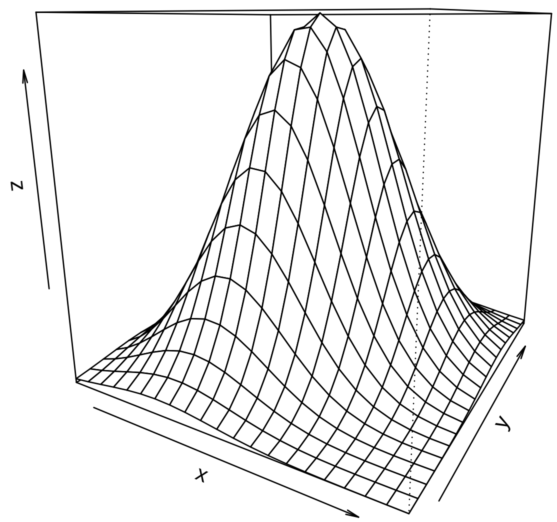

```
> image(x,y,z,col=
+ rev(heat.colors(101)))
> contour(x,y,z,add=TRUE)
```



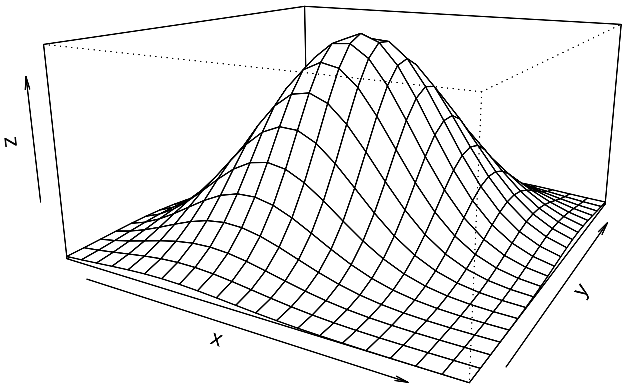
```
> persp(x,y,z)
```



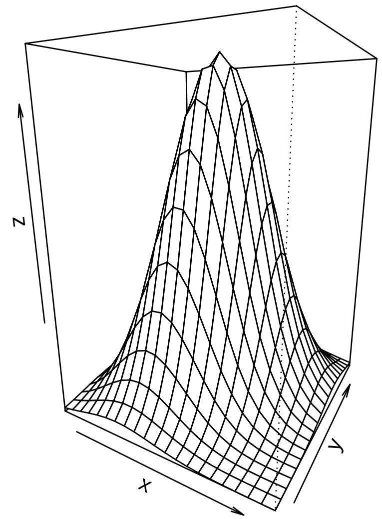

```
> persp(x,y,z,theta=30)
```



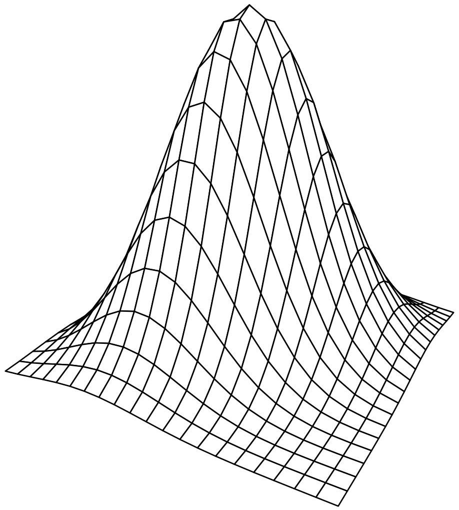
```
> persp(x,y,z,theta=30,expand=.5)
```



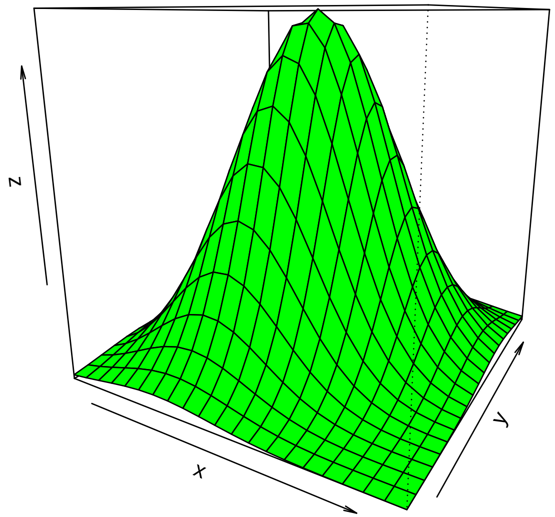

```
> persp(x,y,z,theta=30,expand=1.5)
```



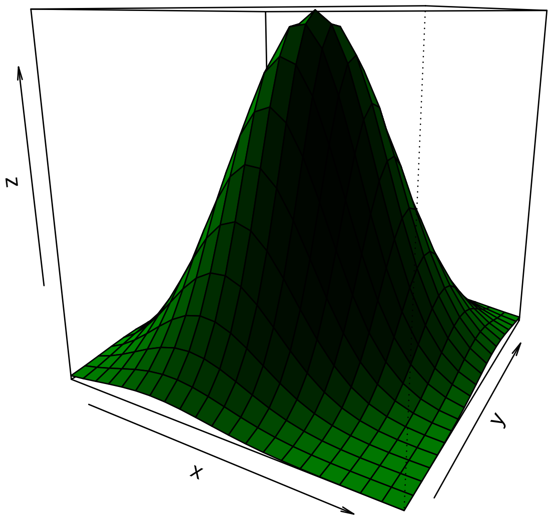
```
> persp(x,y,z,theta=30,box=FALSE)
```

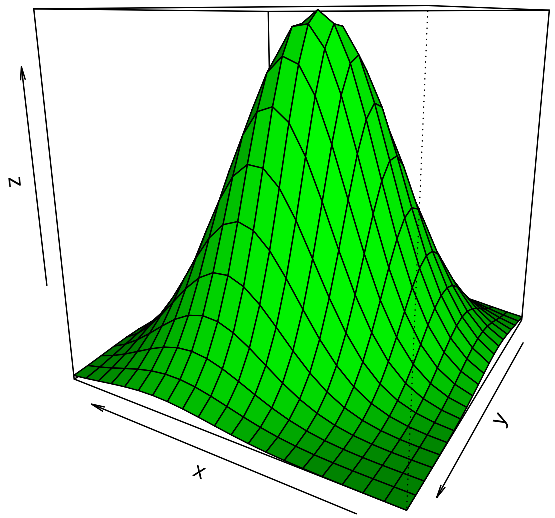


```
> persp(x,y,z,theta=30,col="green")
```



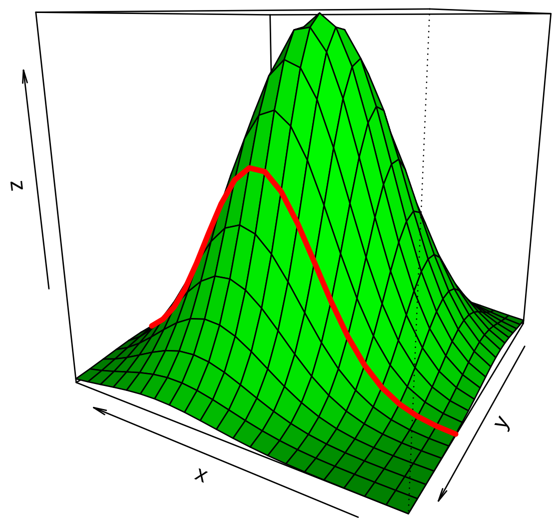
```
> persp(x,y,z,theta=30,col="green",shade=TRUE) > persp(x,y,z,theta=210,col="green",shade=T
```

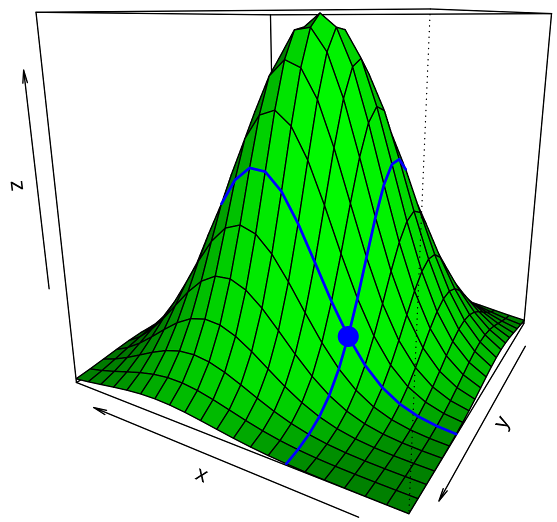




```
> pmat <- persp(x,y,z,theta=210,
+ col="green",shade=TRUE)
> u <- x; v <- rep(1,length(y))
> w <- binorm(u,v,r=.4)
> lines(trans3d(u,v,w, pmat),
+ lwd=4,col="red")
```

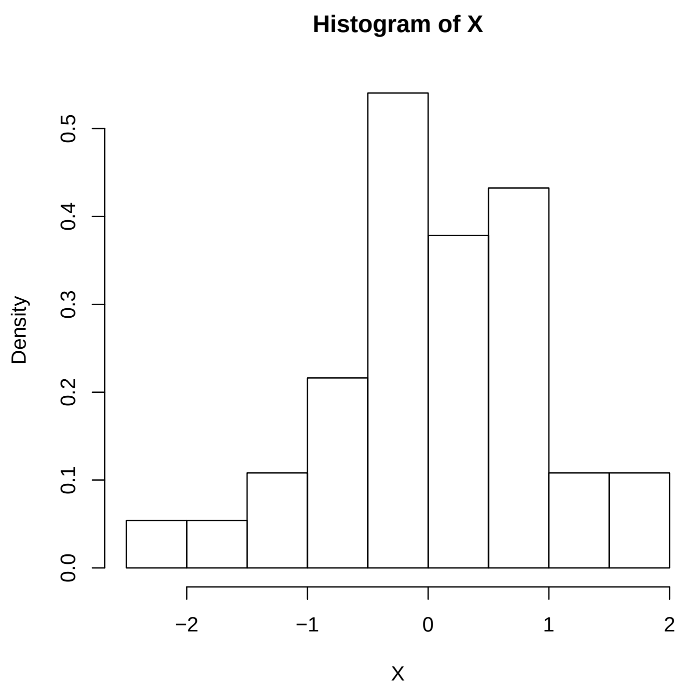
```
> u <- -1; v <- 1
> w <- binorm(u,v,r=.4)
> points(trans3d(u,v,w, pmat),
+ pch=19,col="blue")
```

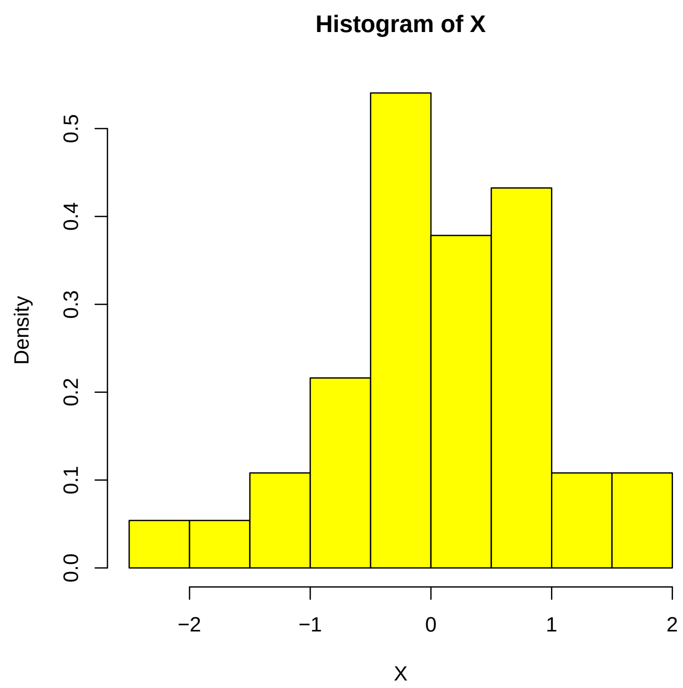




```
> X <- rnorm(37)
> hist(X,xlab="X",ylab="Density",
+ probability=TRUE)
```

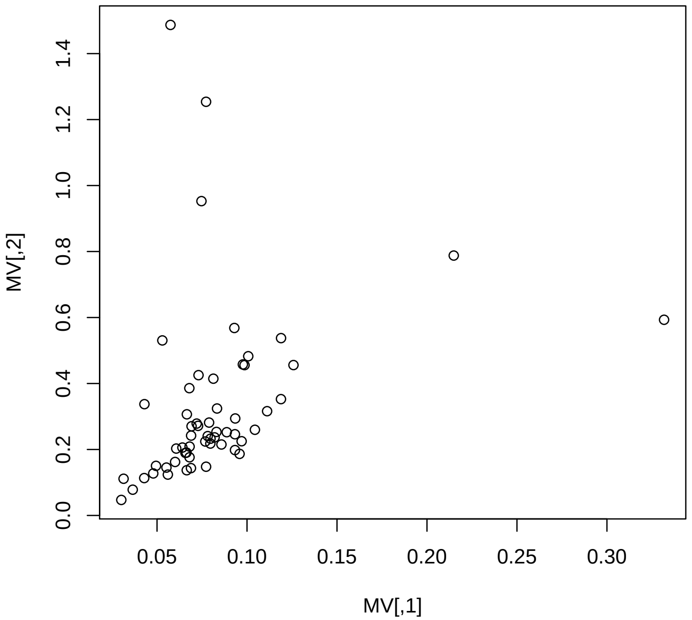
```
> hist(X,xlab="X",ylab="Density",
+ probability=TRUE,col="yellow")
```

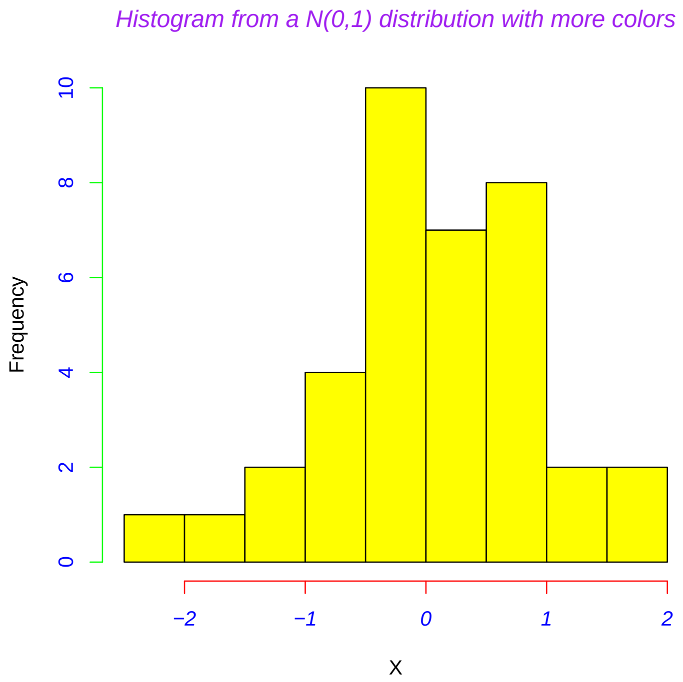




```
> hist(Histo,main="Histogram
  from a N(0,1) distribution")
```

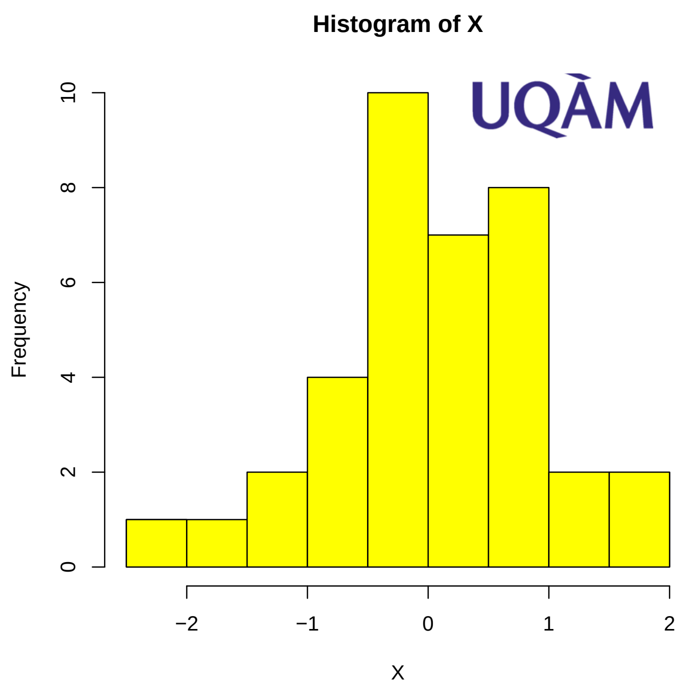
```
> plot(Histo,col="yellow",axes=FALSE,main="")
> title(main="Histogram from a N(0,1) distribution
  with more colors",font.main=3,col.main="purple")
> axis(1,col="red",col.axis="blue",font.axis=3)
> axis(2,col="green",col.axis="blue",font.axis=1)
```

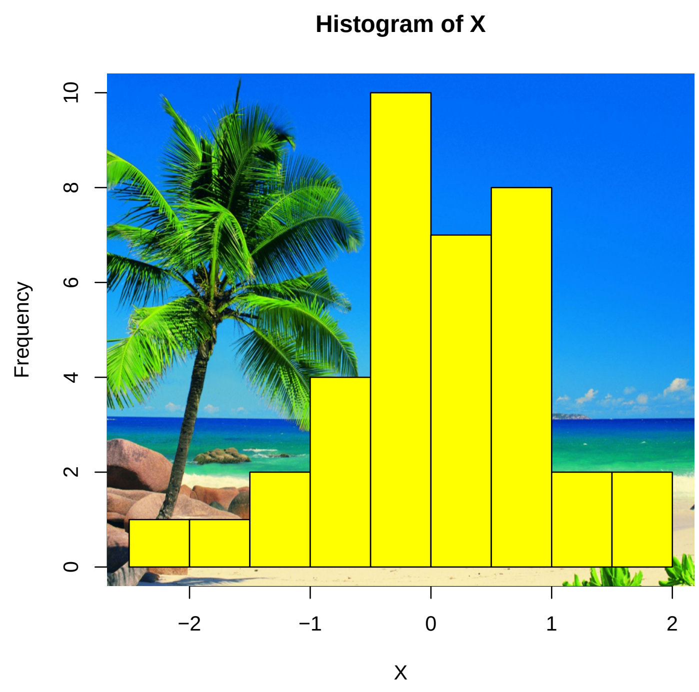




```
> library(png)
> img <- readPNG("backgroundgraph.png")
> r <- as.raster(img[,,1:3])
> hist(X)
> rasterImage(r,-2,0,2,11)
> lines(Histo,col="yellow")
```

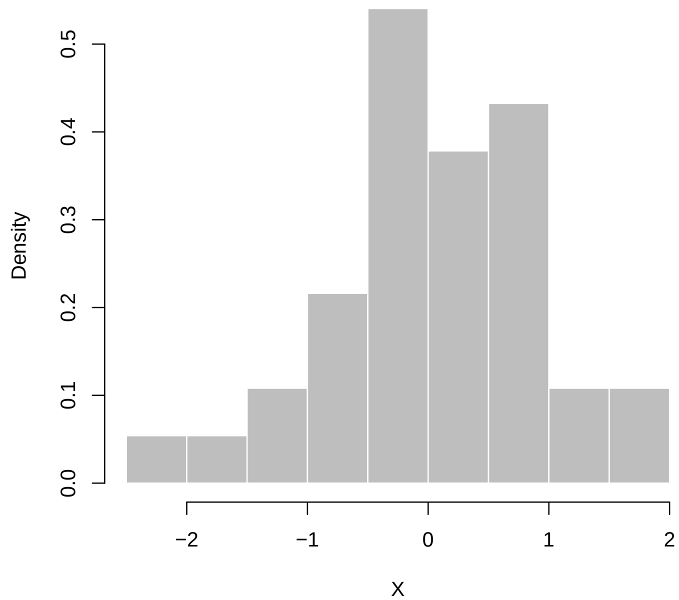
```
> img2 <- readPNG("backgroundgraph2.png")
> r <- as.raster(img2[,,1:3])
> hist(X)
> rasterImage(r2,-3,-.5,2.5,12)
> lines(Histo,col="yellow")
```

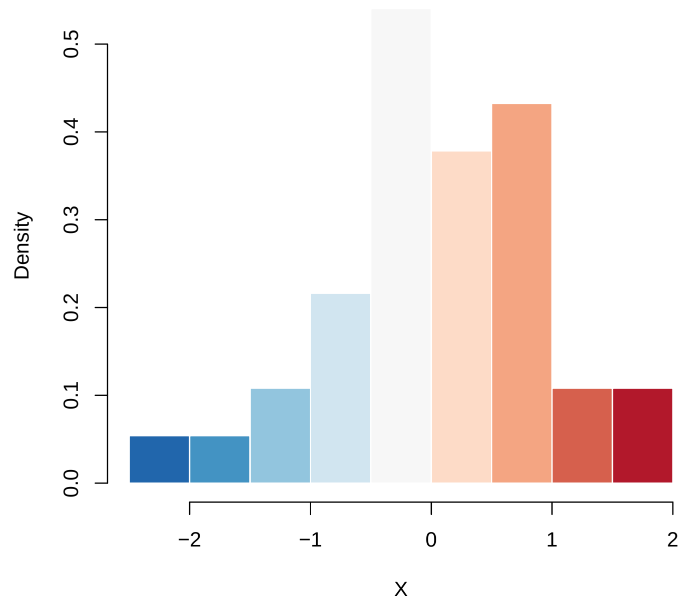




```
> hist(X,main="",col="grey",
+ border="white",probability=TRUE)
```

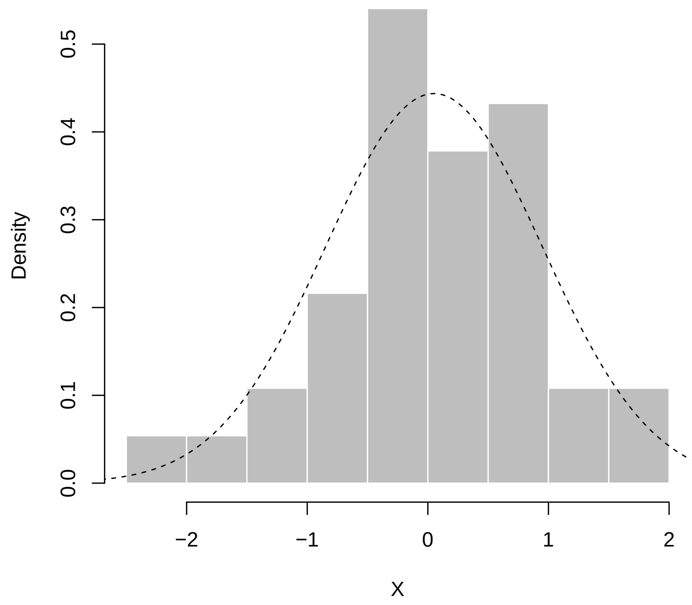
```
> library(RColorBrewer)
> rangecol=rev(brewer.pal(9, "RdBu"))
> hist(X,main="",col=rangecol,
+ border="white",probability=TRUE)
```

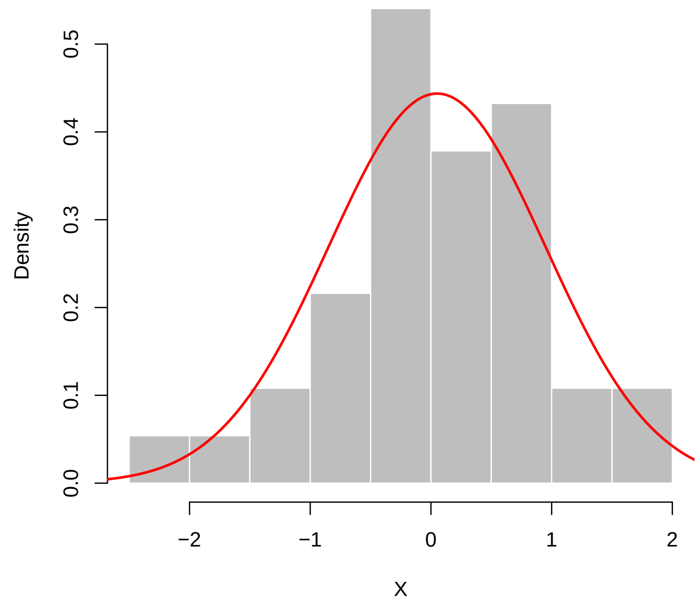




```
> hist(X,main="",col="grey",
+ border="white",probability=TRUE)
> u <- seq(min(X)-1,max(X)+1,by=.01)
> lines(u,dnorm(u,mean(X),sd(X)),lty=2)
```

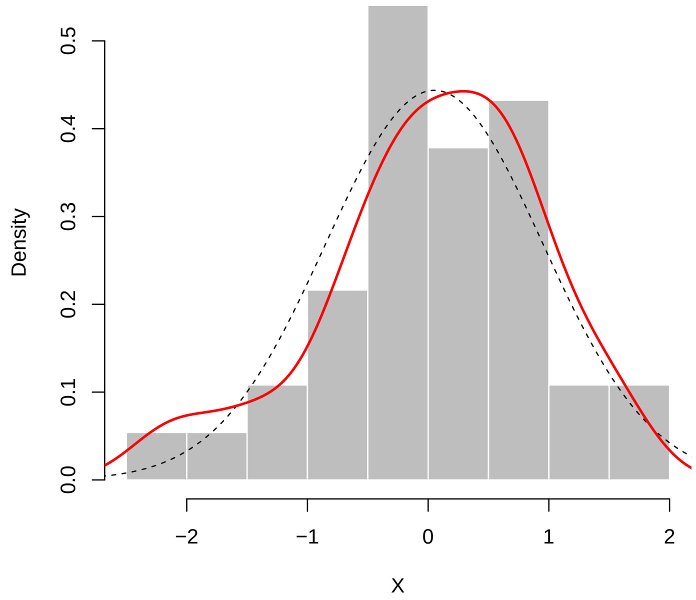
```
> hist(X,main="",col="grey",
+ border="white",probability=TRUE)
> u <- seq(min(X)-1,max(X)+1,by=.01)
> lines(u,dnorm(u,mean(X),sd(X)),lwd=2,col="red")
```

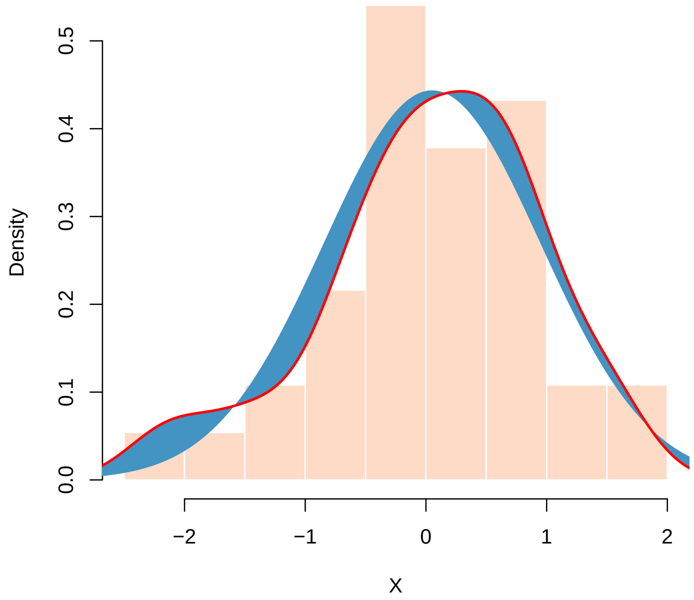




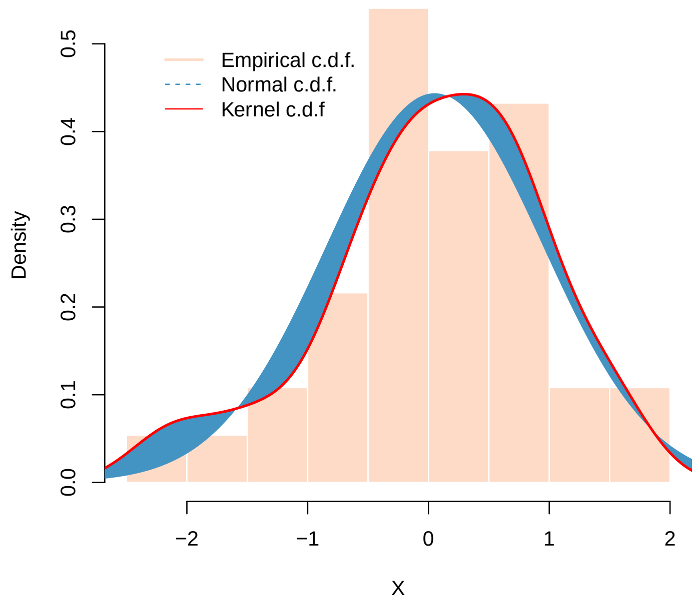
```
> hist(X,main="",col="grey",
+ border="white",probability=TRUE)
> u <- seq(min(X)-1,max(X)+1,by=.01)
> lines(u,dnorm(u,mean(X),sd(X)),lty=2)
> d <- density(X)
> lines(d$x,d$y,lwd=2,col="red")
```

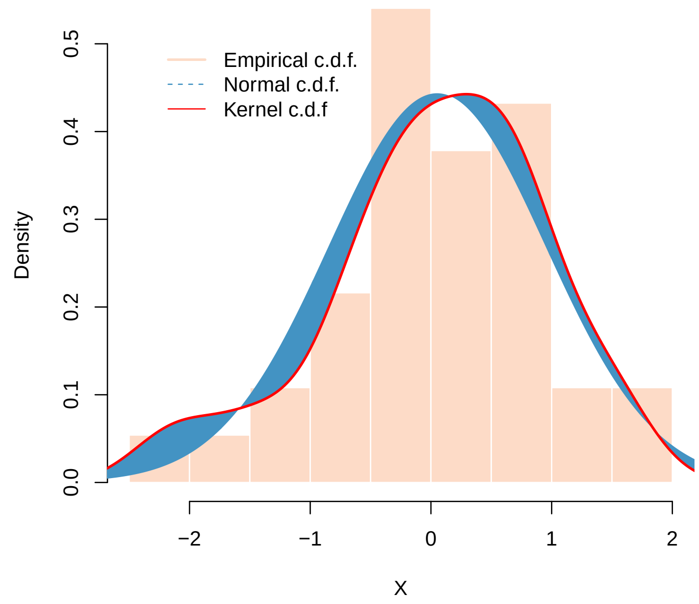
```
> hist(X,main="",col=rangecol[6],
+ border="white",probability=TRUE)
> polygon(c(d$x,rev(d$x)),c(d$y,
+ dnorm(rev(d$x),mean(X),sd(X))),
+ col=rangecol[2],border=NA)
+ lines(d$x,d$y,lwd=2,col="red")
```



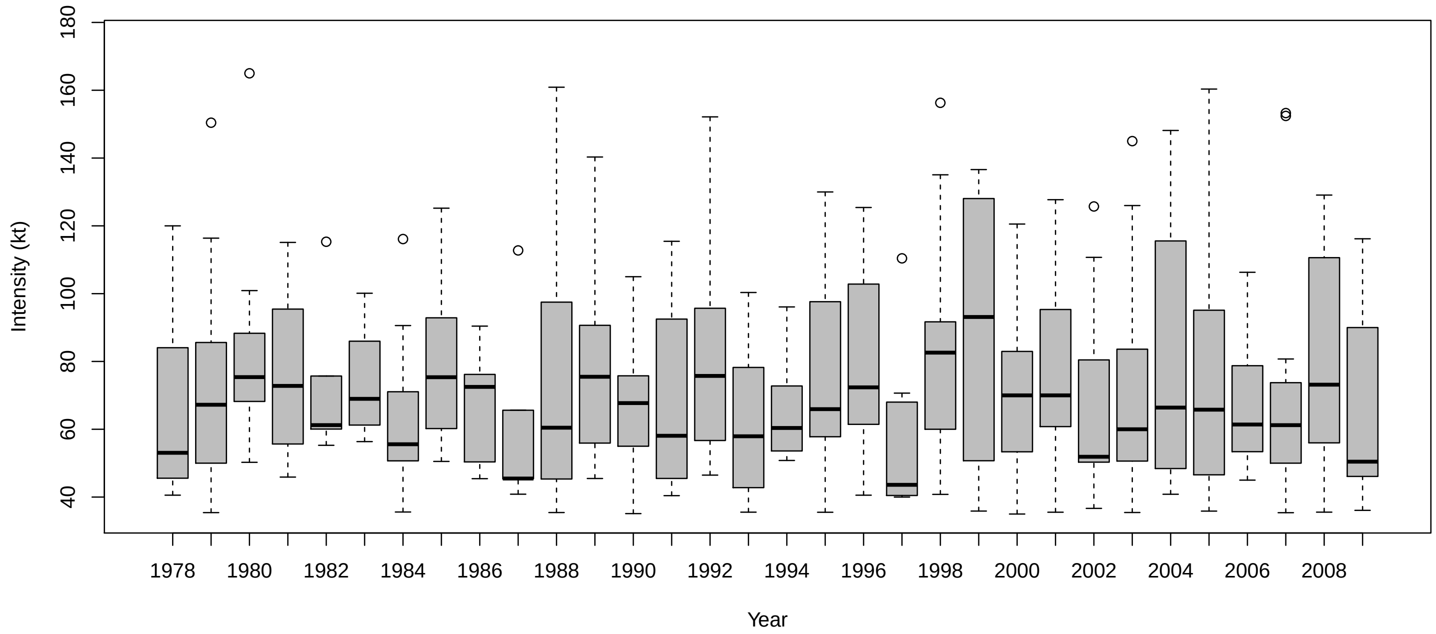


```
>  legend(locator(1),c("Empirical c.d.f.","Normal c.d.f.","Kernel c.d.f"),
+ col=c(rangecol[6],rangecol[2],"red"),lwd=c(2,1,1),lty=c(1,2,1),bty="n")
```

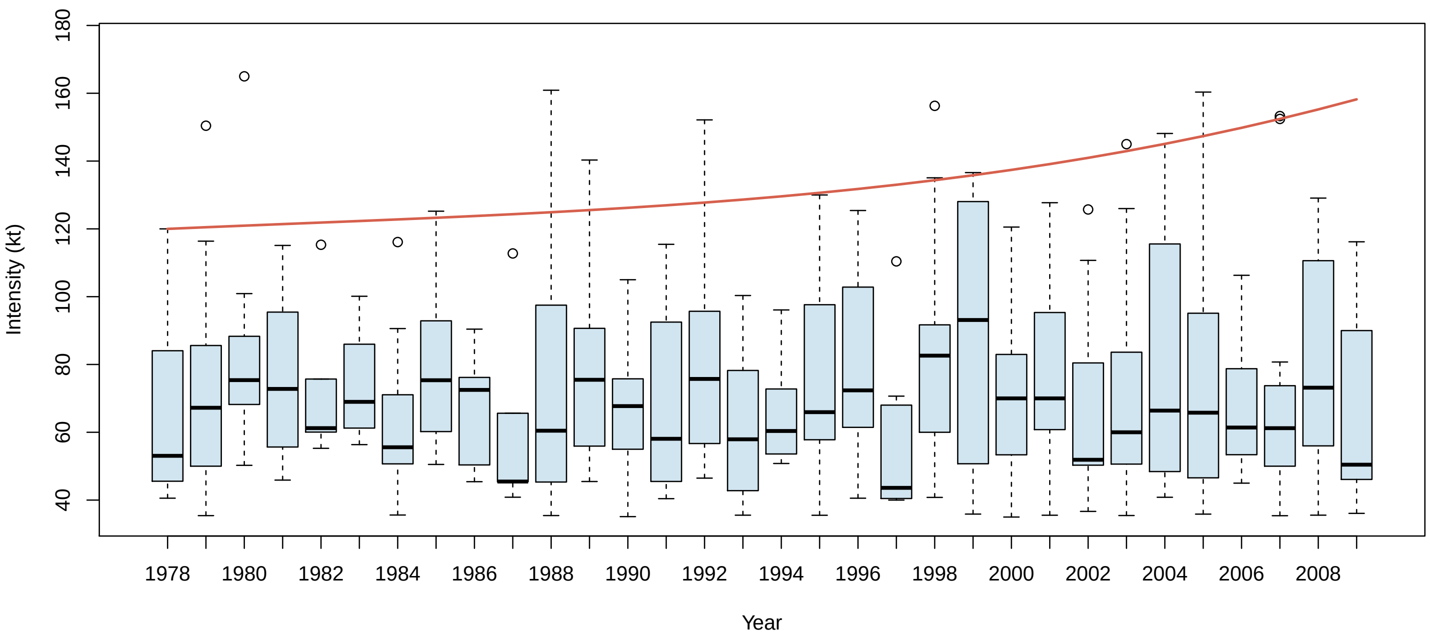




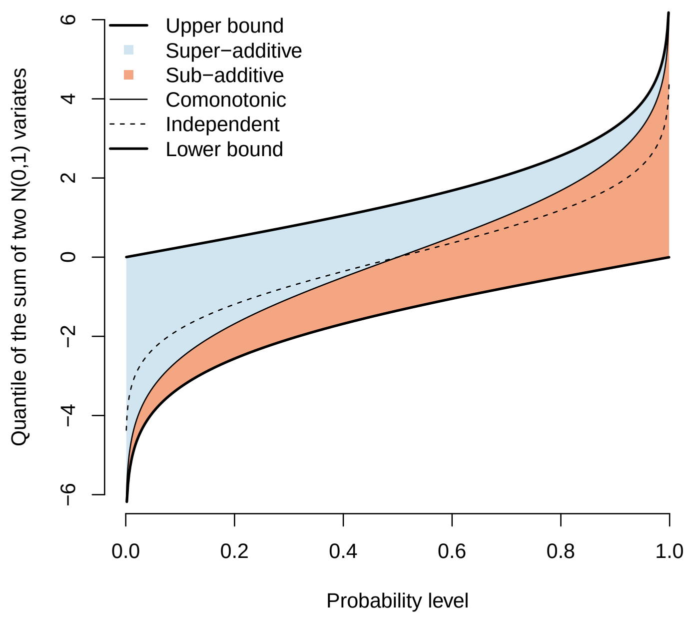
```
> StormMax <- read.table("extremedatasince1899.csv",header=TRUE,sep=",")
> StormMaxBasin <- subset(StormMax,(Region=="Basin")&(Yr>1977))
> attach(StormMaxBasin)
> boxplot(Wmax~as.factor(Yr),ylim=c(35,175),xlab="Year",
+ ylab="Intensity (kt)",col="grey")
```

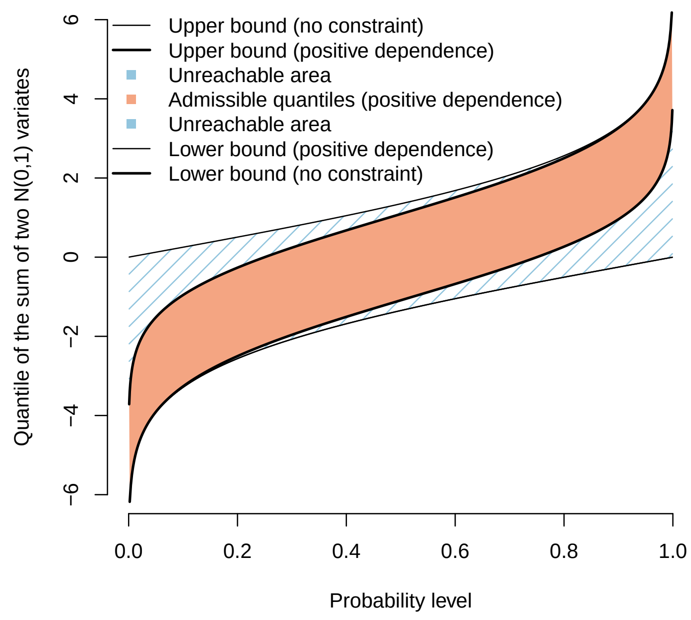


```
> boxplot(Wmax~as.factor(Yr),ylim=c(35,175),col=rangecol[4])
> library(quantreg); library(splines)
> reg <- rq(Wmax~bs(Yr,df=3),tau=.95,data=StormMaxBasin)
> yp <- predict(reg,newdata=data.frame(Yr=1978:2009))
> lines(1:32,yp,lwd=2,col=rangecol[8])
```



```
> polygon(c(x,rev(x)),c(Qsup,rev(Qsupind)),col=rangecol[3],border=NA,density=10)
> polygon(c(x,rev(x)),c(Qinf,rev(Qinfind)),col=rangecol[3],border=NA,density=10)
> polygon(c(x,rev(x)),c(Qinfind,rev(Qsupind)),col=rangecol[7],border=NA)
```

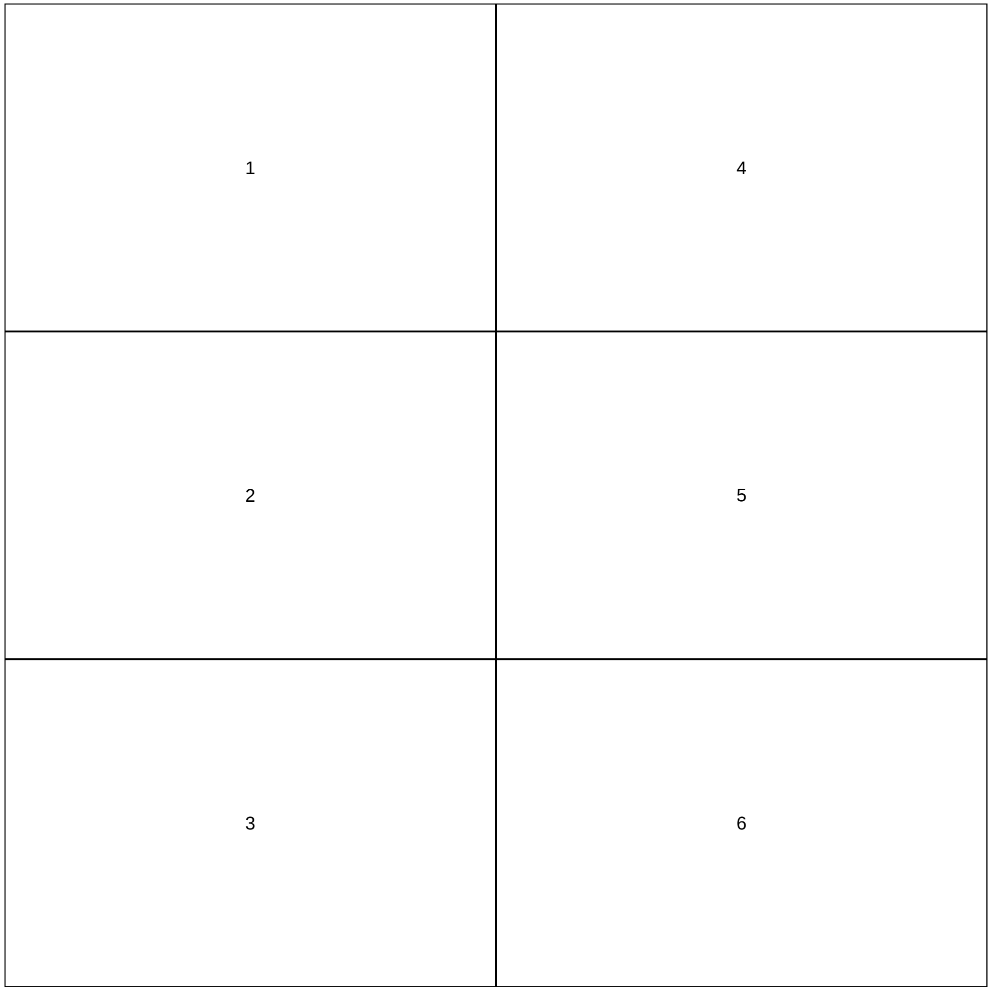




Geometry of plots

It is possible to define areas within a plot, via parameters `layout`.

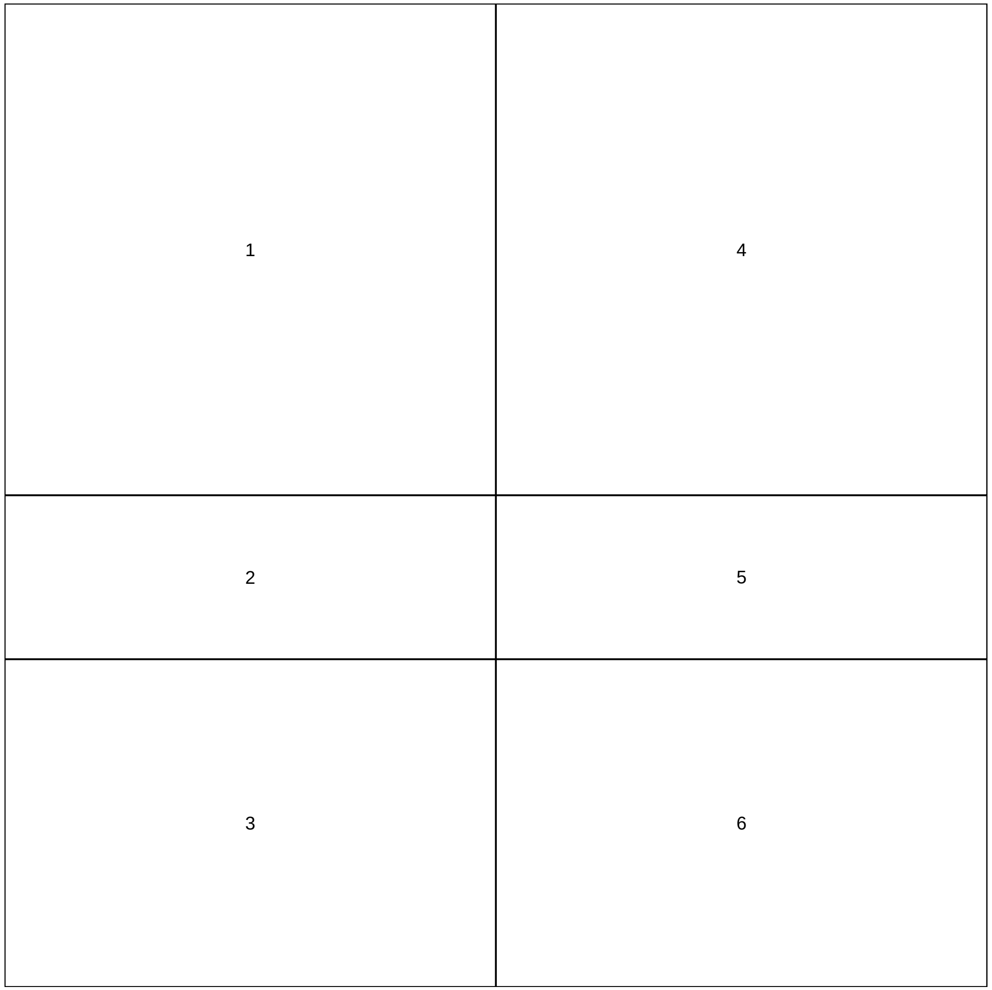
```
> mat <- matrix(1:6,3,2)
> mat
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> layout(mat)
> layout.show(6)
```



Geometry of plots

It is possible to define areas within a plot, via parameters `layout`.

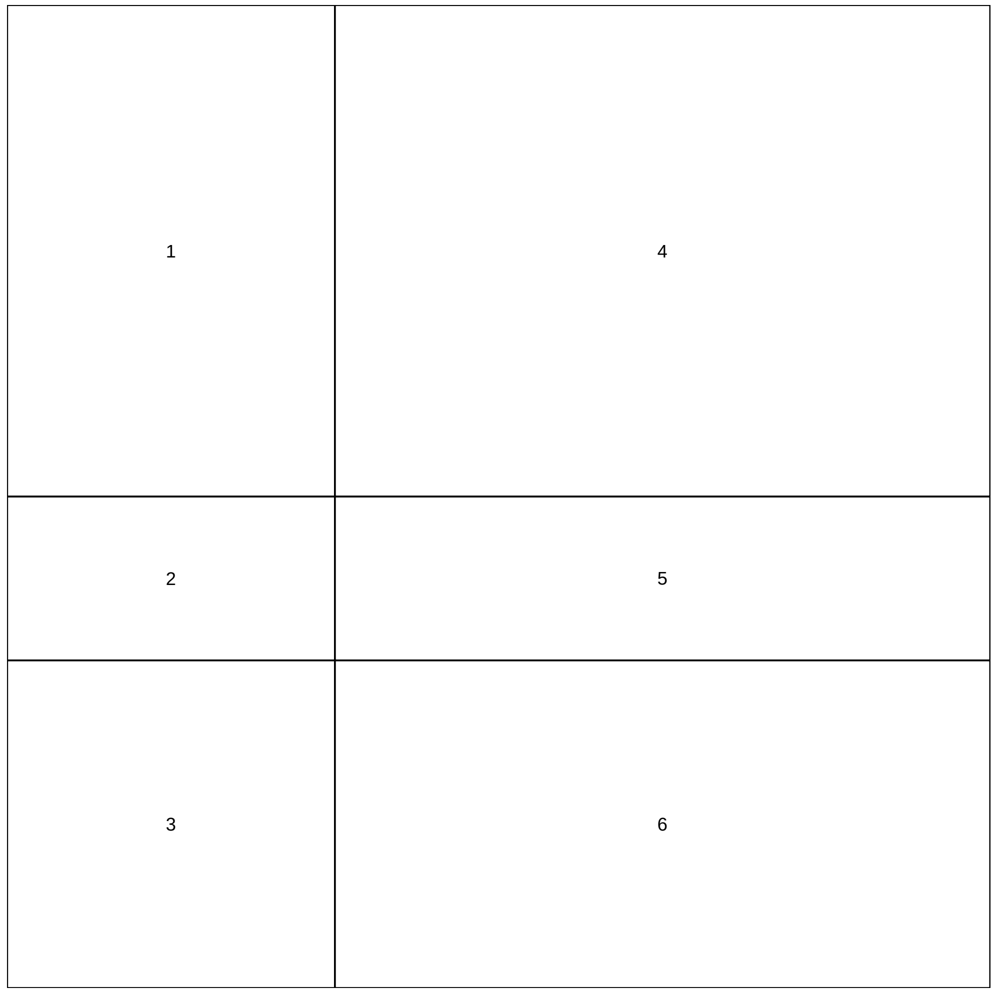
```
> mat <- matrix(1:6,3,2)
> mat
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> layout(mat,c(1,1),c(3,1,2))
> layout.show(6)
```



Geometry of plots

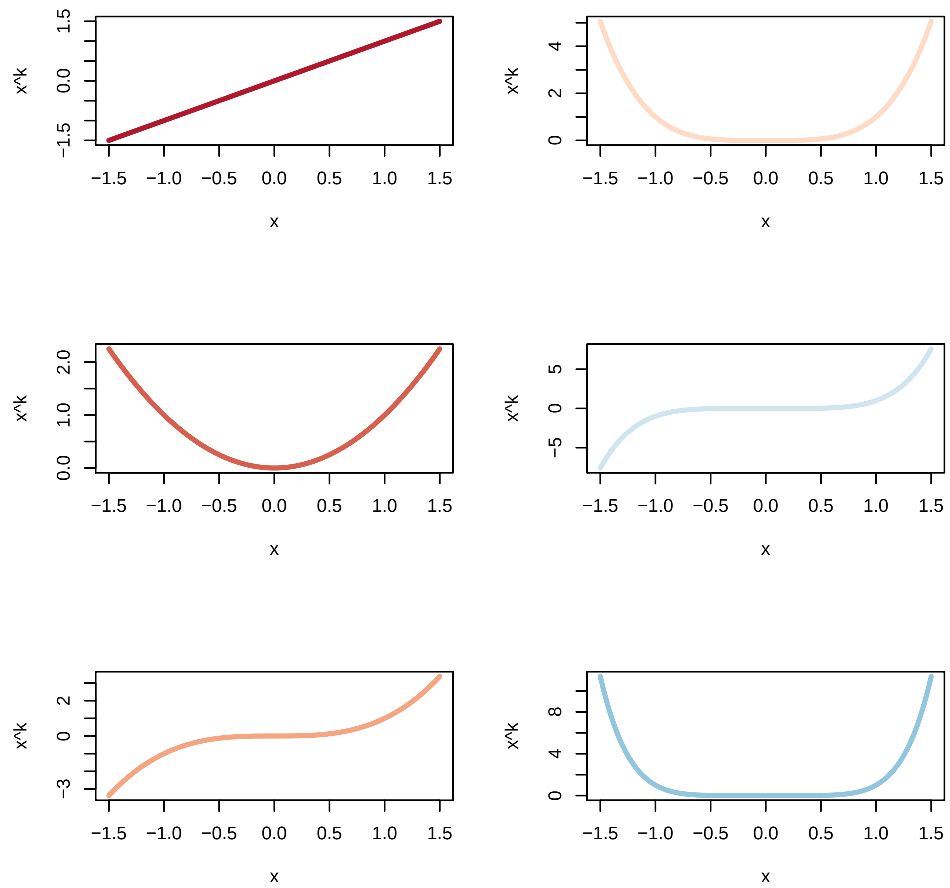
It is possible to define areas within a plot, via parameters `layout`.

```
> mat <- matrix(1:6,3,2)
> mat
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> layout(mat,c(1,2),c(3,1,2))
> layout.show(6)
```

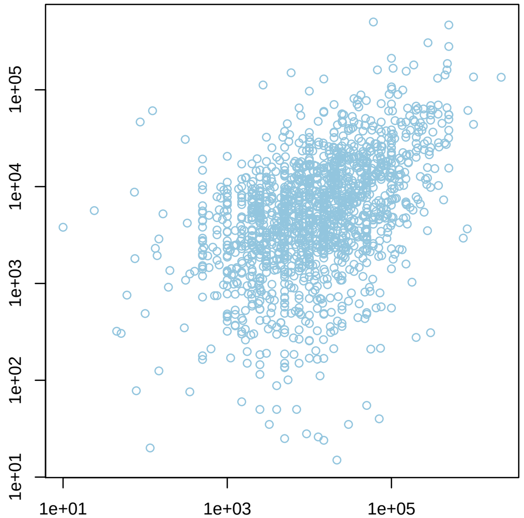


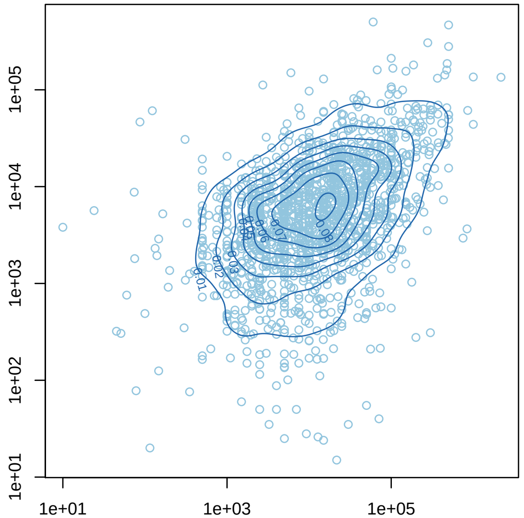
Geometry of plots

```
> layout(mat)
> x<-seq(-1.5,1.5,by=.02)
> for(k in 1:6){
+ plot(x,x^k,type="l",col=cl[k],lwd=3)
+ }
```

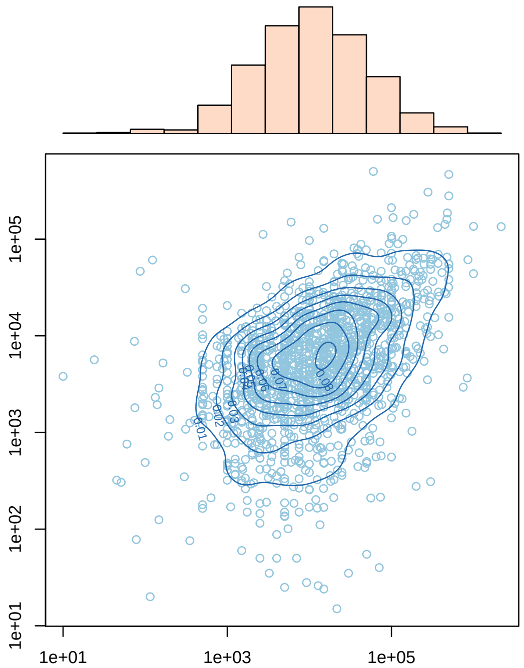


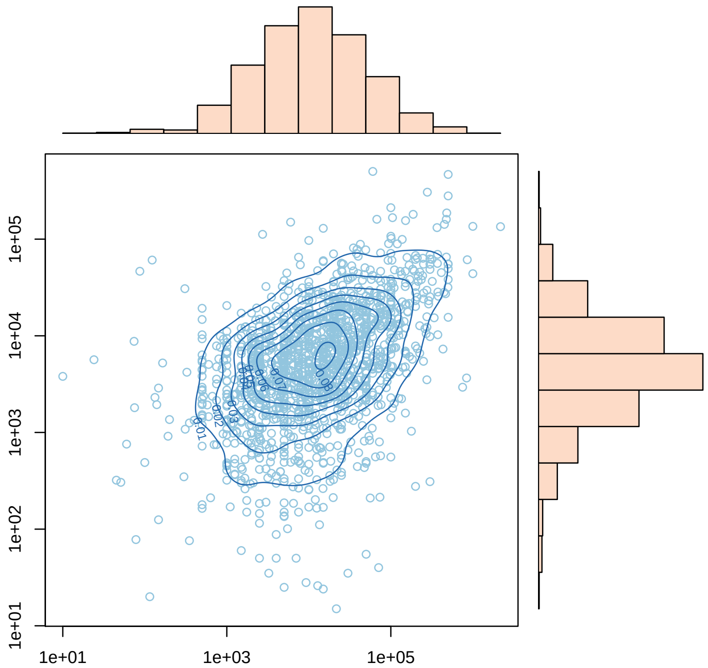
```
> library(evd); data(lossalae); library(MASS)
> xhist <- hist(log(X<-lossalae[,1]), plot=FALSE)
> yhist <- hist(log(Y<-lossalae[,2]), plot=FALSE)
> par(mar=c(3,3,1,1))
> layout(matrix(c(2,0,1,3),2,2,byrow=TRUE),
> c(3,1), c(1,3), TRUE)
> plot(X,Y, xlab="", ylab="",log="xy",col=rangecol[3])
```





```
> kernel <- kde2d(log(X),log(Y),n=201)
> contour(exp(kernel$x),exp(kernel$y),kernel$z,add=TRUE,col=rangecol[1])
> par(mar=c(0,3,1,1))
> barplot(xhist$counts, axes=FALSE, ylim=c(0, top),space=0,col=rangecol[6])
> par(mar=c(3,0,1,1))
> barplot(yhist$counts, axes=FALSE, xlim=c(0, top),space=0, horiz=TRUE,col=rangecol[6])
```

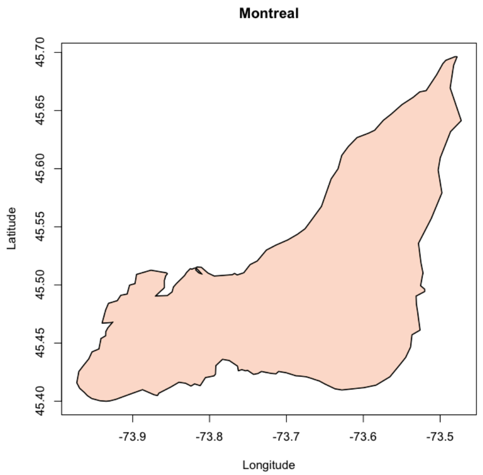




Maps

Maps can be plotted from shapefiles, via http ://gadm.org/download

```
> require(ggplot2); load("CAN_adm2.RData")
> plot(gadm)
> montreal=fortify(gadm[gadm$NAME_2 == "Communaute-Urbaine-de-Montreal",])
> plot(montreal[,c("long","lat")],t="l")
> polygon(x=montreal[,"long"],y=montreal[,"lat"],col=cl[4])
```



R versus other (statistical) softwares

"*The power of the language R lies with its functions for statistical modelling, data analysis and graphics ; its ability to read and write data from various data sources ; as well as the opportunity to embed R in excel or other languages like VBA. In the way SAS is good for data manipulations, R is superior for modelling and graphical output*"

Source : http ://www.actuaries.org.uk/system/files/documents/pdf/actuarial-toolkit.pdf

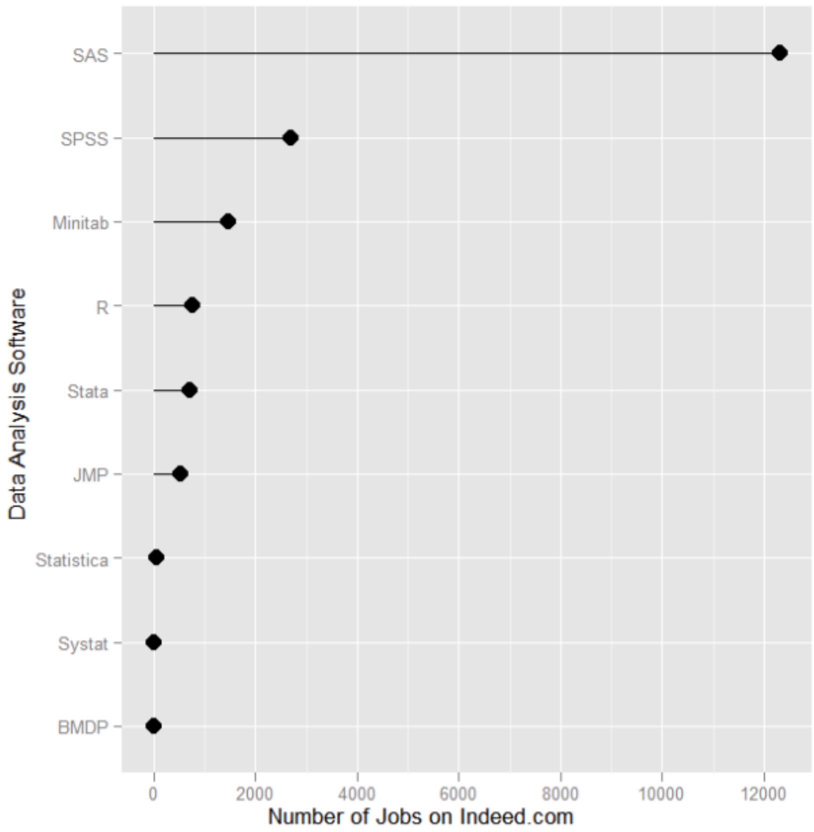
R versus other (statistical) softwares

SAS	PC : $ 6,000 per seat - server : $28,000 per processor
Matlab	$ 2,150 (*commercial*)
Excel	
SPSS	$ 4,975
EViews	$ 1,075 (*commercial*)
RATS	$ 500
Gauss	-
Stata	$ 1,195 (*commercial*)
S-Plus	$ 2,399 per year

Source : http ://en.wikipedia.org/wiki/Comparison_of_statistical_packages

R in the non-academic world

What software skills are employers seeking ?



Source : http ://r4stats.com/articles/popularity/

R in the insurance industry

From 2011, Asia Capital Reinsurance Group (ACR) uses R to Solve Big Data Challenges

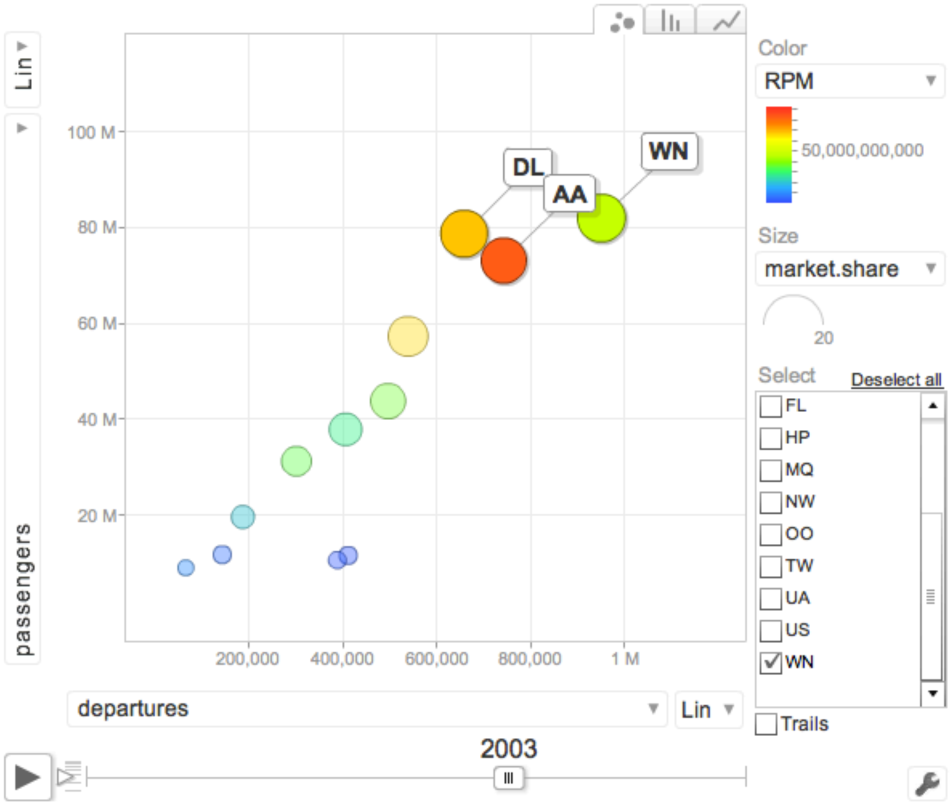
Source : http ://www.reuters.com/article/2011/07/21/idUS133061+21-Jul-2011+BW20110721

From 2011, Lloyd's uses motion charts created with R to provide analysis to investors.

Source : http ://blog.revolutionanalytics.com/2011/07/r-visualizes-lloyds.html

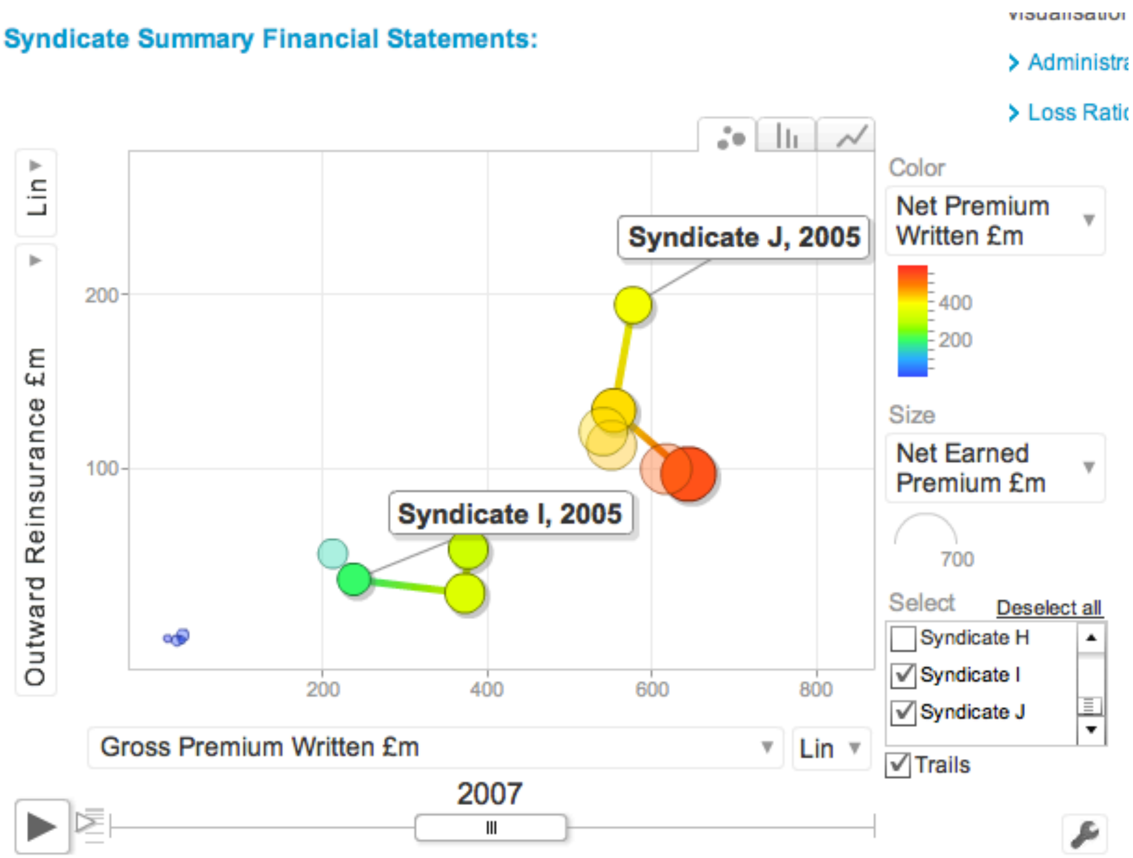
Source : http ://www.revolutionanalytics.com/what-is-open-source-r/companies-using-r.php

R in the insurance industry



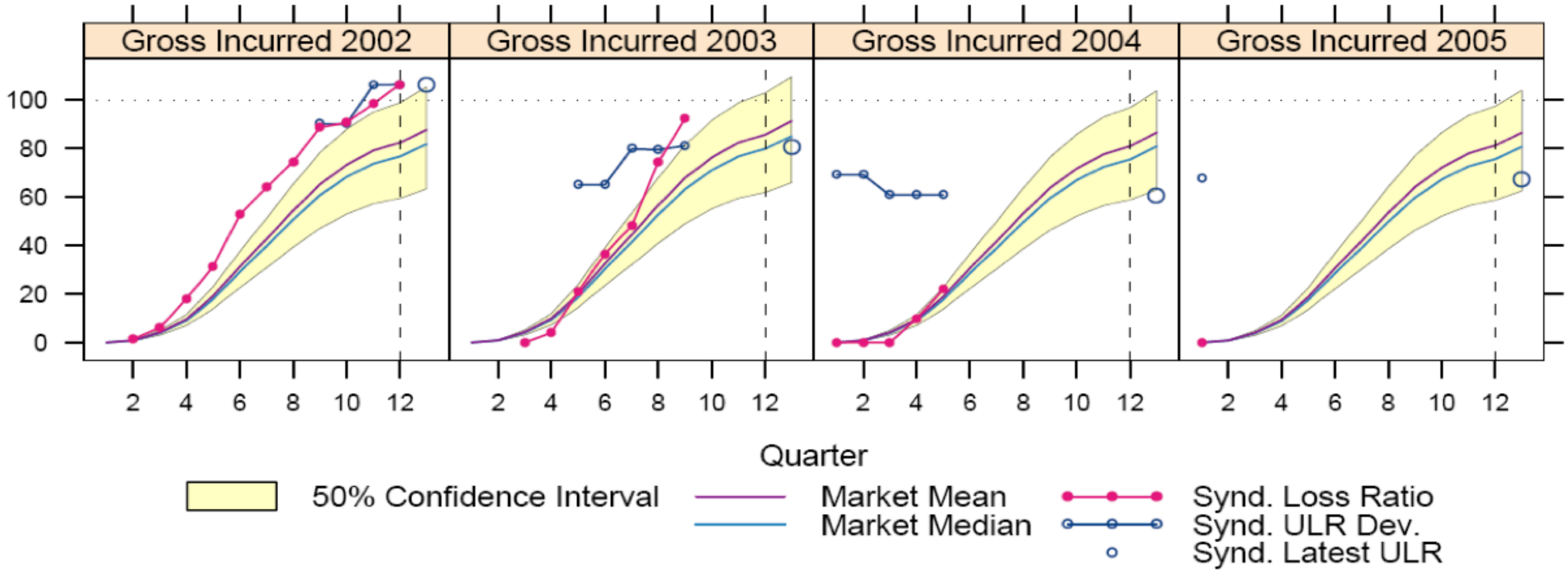
Source : http ://jeffreybreen.wordpress.com/2011/07/14/r-one-liners-googlevis/

R in the insurance industry



Source : http ://jeffreybreen.wordpress.com/2011/07/14/r-one-liners-googlevis/

R in the insurance industry



Source : http ://lamages.blogspot.ca/2011/09/r-and-insurance.html, i.e. Markus Gesmann's blog

Popularity of R versus other languages

as at January 2013,

Transparent Language Popularity

1.	C	17.780%
2.	Java	15.031%
8.	Python	4.409%
12.	R	1.183%
22.	Matlab	0.627%
27.	SAS	0.530%

Source : http ://lang-index.sourceforge.net/

TIOBE Programming Community Index

1.	C	17.855%
2.	Java	17.417%
7.	Visual Basic	4.749%
8.	Python	4.749%
17.	Matlab	0.641%
23.	SAS	0.571%
26.	R	0.444%

Source : http ://www.tiobe.com/index.php/

Popularity of R versus other languages

as at January 2013, tags

stackoverflow

C++	399,323
Java	348,418
Python	154,647
R	21,818
Matlab	14,580
SAS	899

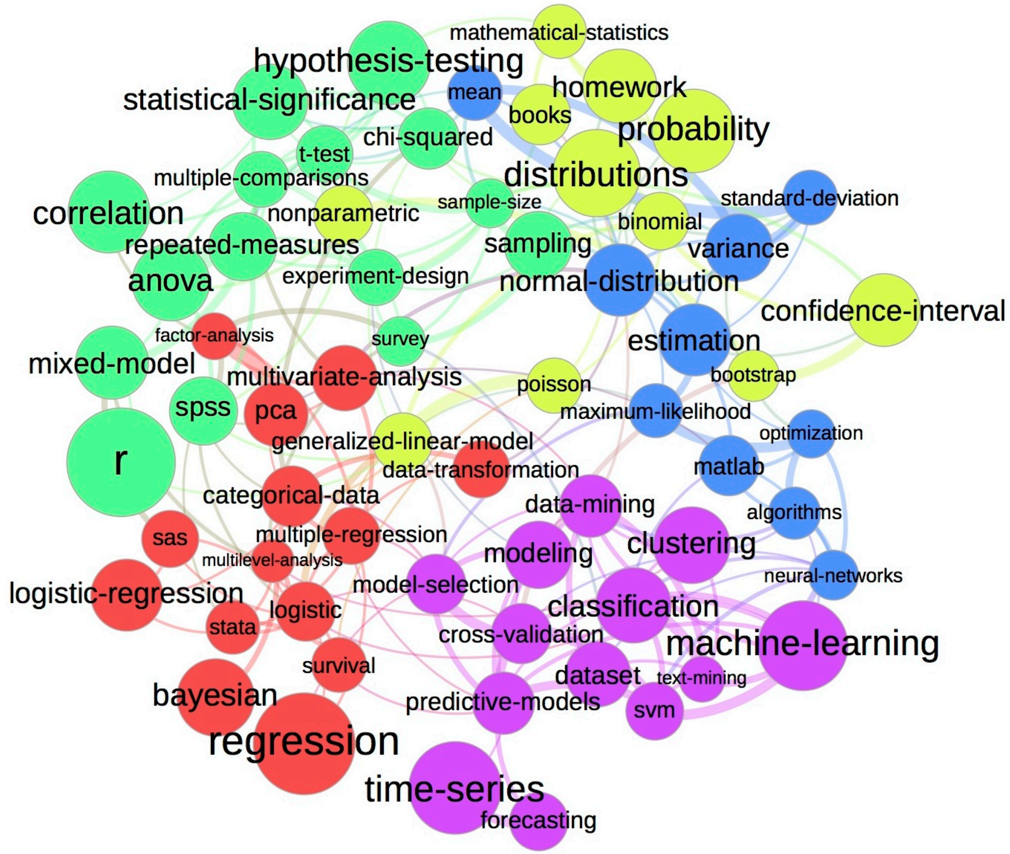
Source : http ://stackoverflow.com/tags ?tab=popular

Cross Validated

R	3,008
Matlab	210
SAS	187
Stata	153
Java	26

Source : http ://www.tiobe.com/index.php/

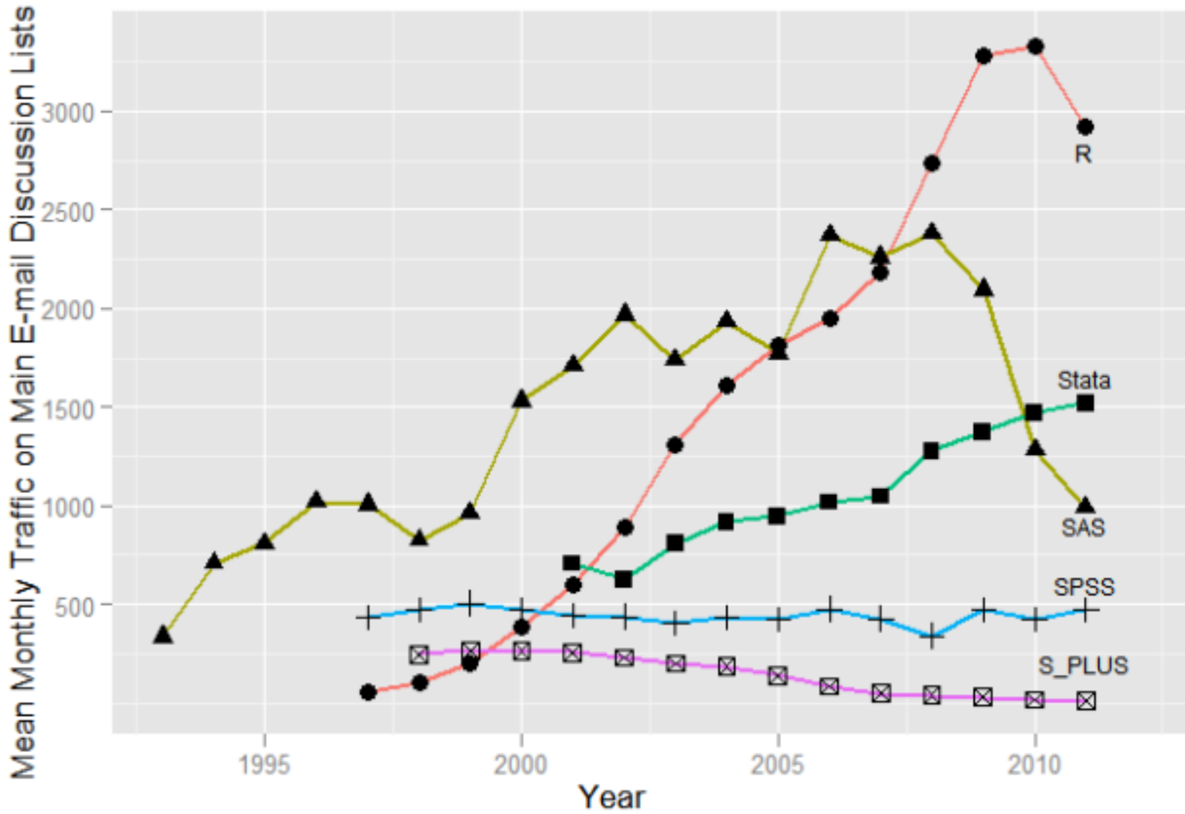
R versus other statistical languages



Source : http ://meta.stats.stackexchange.com/questions/1467/tag-map-for-crossvalidated

R versus other statistical languages

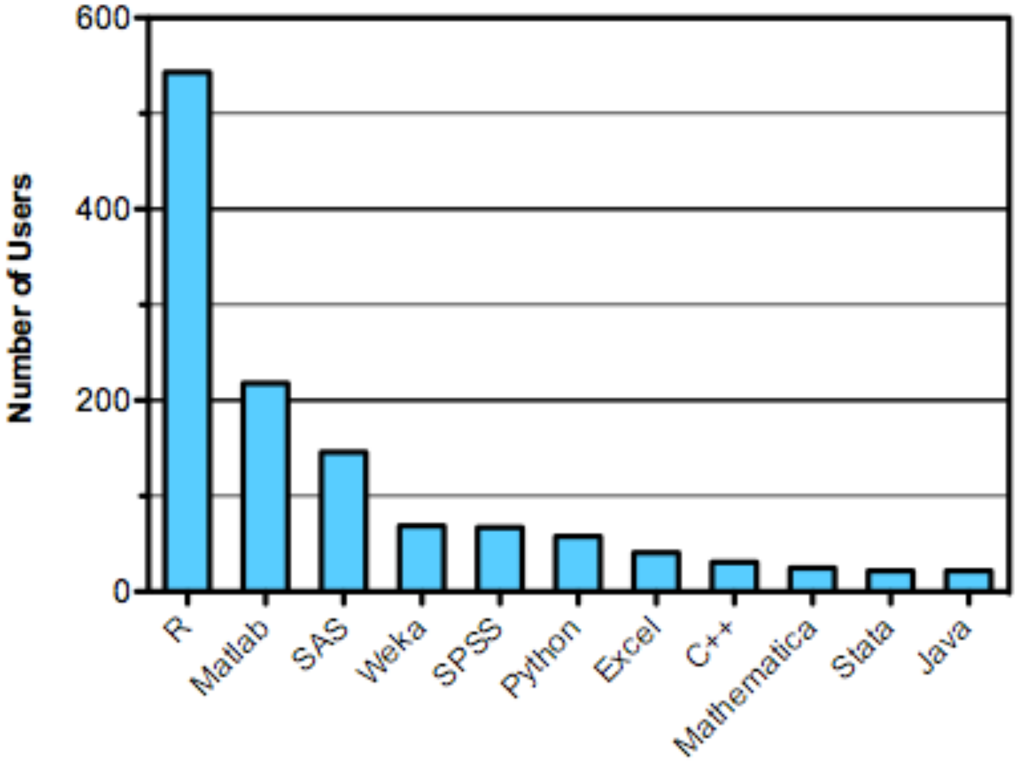
Plot of listserv discussion traffic by year (through December 31, 2011)



Source : http ://r4stats.com/articles/popularity/

R versus other statistical languages

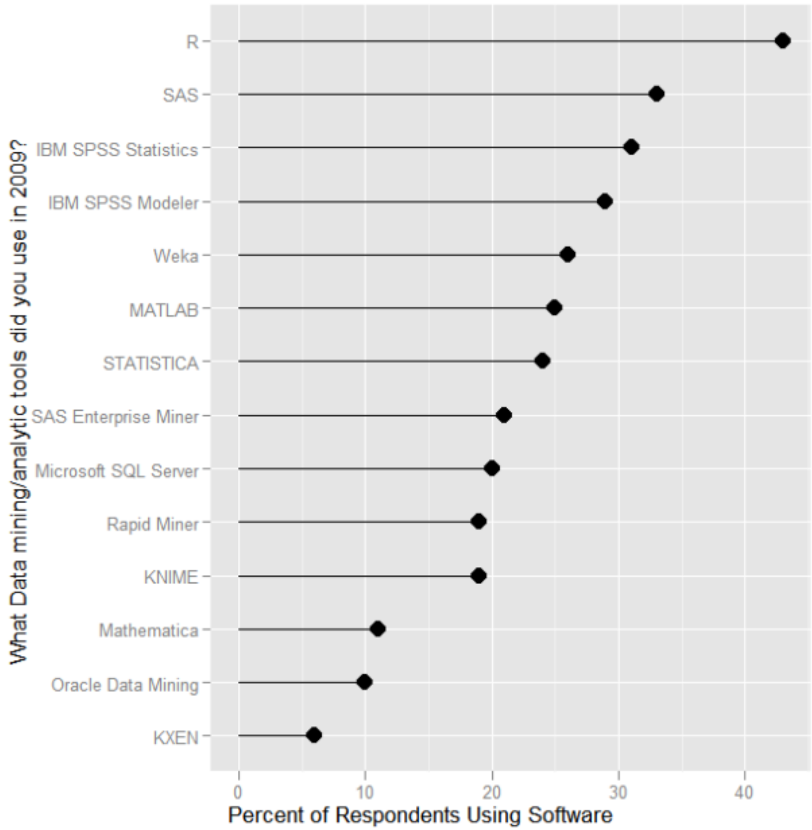
Software used by competitors on Kaggle



Source : http ://r4stats.com/articles/popularity/ and http ://www.kaggle.com/wiki/Software

R versus other statistical languages

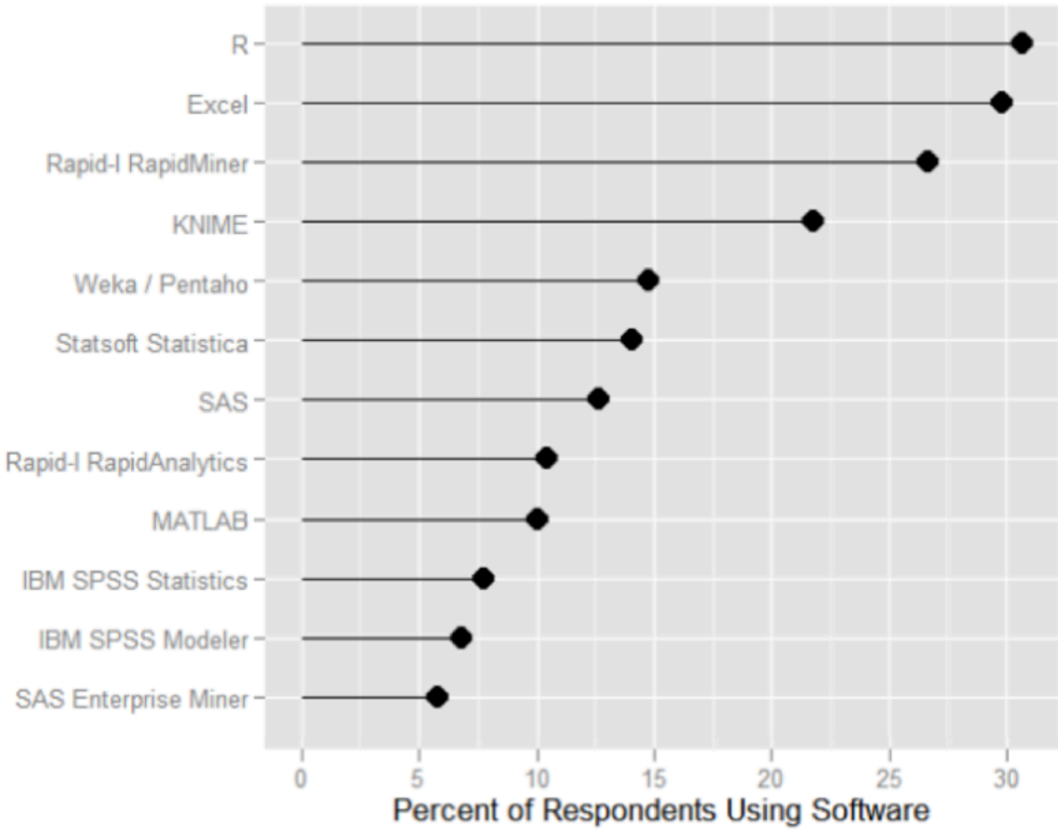
Data mining/analytic tools reported in use on Rexer Analytics survey, 2009.



Source : http ://r4stats.com/articles/popularity/

R versus other statistical languages

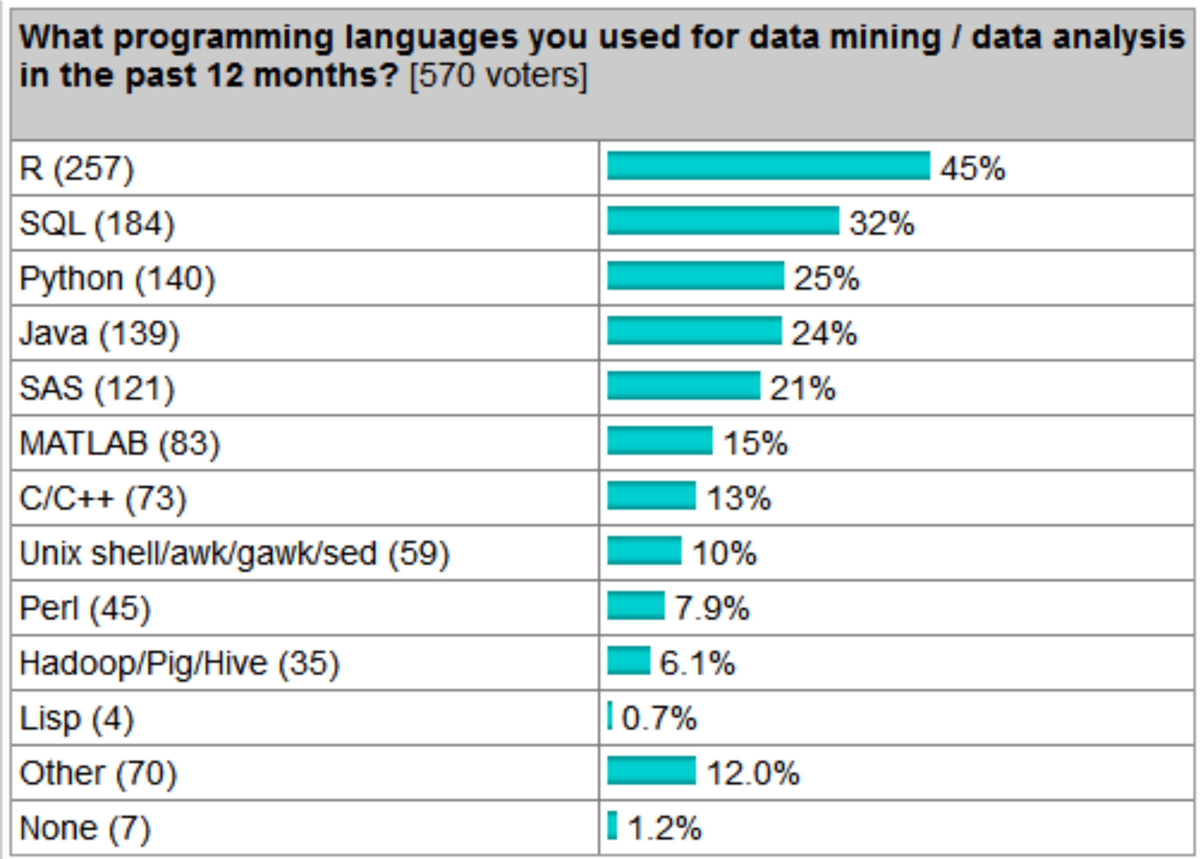
“What programming languages you used for data analysis in the past 12 months ?”



Source : http ://r4stats.com/articles/popularity/

R versus other statistical languages

"What programming languages you used for data analysis ?"



What programming languages you used for data mining / data analysis in the past 12 months? [570 voters]	
R (257)	45%
SQL (184)	32%
Python (140)	25%
Java (139)	24%
SAS (121)	21%
MATLAB (83)	15%
C/C++ (73)	13%
Unix shell/awk/gawk/sed (59)	10%
Perl (45)	7.9%
Hadoop/Pig/Hive (35)	6.1%
Lisp (4)	0.7%
Other (70)	12.0%
None (7)	1.2%

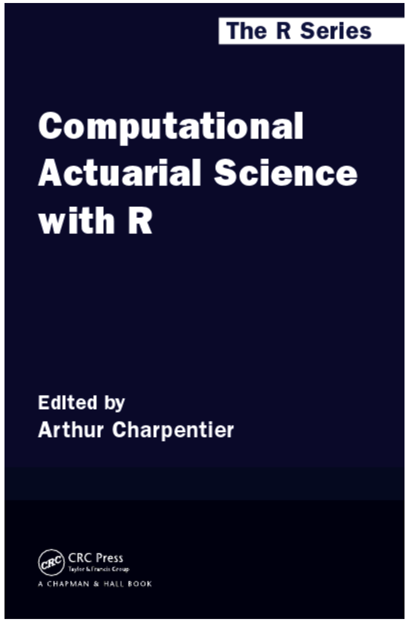
Source : http ://r4stats.com/articles/popularity/

Take-home message (for this first part)

“*The best thing about R is that it was developed by statisticians.*

The worst thing about R is that it was developed by statisticians.”

Bo Cowgill, Google



To go further...

forthcoming book on Computational Actuarial Science