# Brief Inroduction to R - Examples

In the first exercises we are going to get familiar with R and practice some basic commands.

R is a full featured statistics package as well as a full programming language. In these classes we will only use simple commands. Any advanceds programming is not required in the course. You will be able to run statistical simulations and make plots of your data.

## Starting R

If you have installed R and RStudio then simply start the RStudio application. The command window is the one with the >. Usually the window in the lower left corner.

## R as a Calculator

The basic operations are +, -, *, /, ^.

```
2+3
```

```
## [1] 5
```

```
2*3
```

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

```
2/3
```

```
## [1] 0.6666667
```

```
2^3
```

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

```
2*(3+1)^2
```

```
## [1] 32
```

```
sqrt(3)
```

```
## [1] 1.732051
```

### Using Variables

Results can be stored in variables and can be used in calculations. For examle, the value of the variable x can be seen by using x as a command.

```
x= 2+3
x
```

```
## [1] 5
```

```
y= 2*3
y
```

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

```
z=x^y
z
```

```
## [1] 15625
```

Another notation for assignment: the arrow: <-.

```
a<- x*y-2*sin(z)+5
a
```

```
## [1] 36.91709
```

## Vectors

A vector is a type of list. Often it is a list of numbers, but it can be a list of other types such as characters. Vectors are created by using the c() function.

A vector with 4 entries:

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

```
## [1] 1 2 3 4
```

```
c(1:4)
```

```
## [1] 1 2 3 4
```

```
c(4:1)
```

```
## [1] 4 3 2 1
```

Sequences of integers:

```
1:4
```

```
## [1] 1 2 3 4
```

```
3:10
```

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

```
2*6:9
```

```
## [1] 12 14 16 18
```

```
10:3
```

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

Sequences of integers can be also obtained by using command seq().

```
seq(7)
```

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

```
seq(7,13)
```

```
## [1]  7  8  9 10 11 12 13
```

```
seq(7,13,2)
```

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

```
seq(14,7,-0.5)
```

```
##  [1] 14.0 13.5 13.0 12.5 12.0 11.5 11.0 10.5 10.0  9.5  9.0  8.5  8.0  7.5
## [15]  7.0
```

```r
seq(from=-2, to=0.7, by=0.1)
```

```
##  [1] -2.0 -1.9 -1.8 -1.7 -1.6 -1.5 -1.4 -1.3 -1.2 -1.1 -1.0 -0.9 -0.8 -0.7
## [15] -0.6 -0.5 -0.4 -0.3 -0.2 -0.1  0.0  0.1  0.2  0.3  0.4  0.5  0.6  0.7
```

We can store vectores in variables

```r
b= c(1:4, 2)
b
```

```
## [1] 1 2 3 4 2
```

```r
c<- c(1:4, 2:8)
c
```

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

A long vector will be displayed over several lines.

```r
d= 1:40
d
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

Vectors as lists of characters.

```r
m=c("klaus","karl","maria","anna", "sonja")
m
```

```
## [1] "klaus" "karl"  "maria" "anna"  "sonja"
```

```r
m[c(1,3)]
```

```
## [1] "klaus" "maria"
```

```r
m[c(2,7)]
```

```
## [1] "karl" NA
```

```r
length(m)
```

```
## [1] 5
```

### Vector arithmetic

A number can be added to a vector, i.e. the number is added to every element of the vector. The same holds for substraction, multiplication, division and powers.

```r
x1=c(3,6,5,2,0,12)
x1
```

```
## [1]  3  6  5  2  0 12
```

```r
x1+ 5.3
```

```
## [1]  8.3 11.3 10.3  7.3  5.3 17.3
```

```r
5*x1
```

```
## [1] 15 30 25 10  0 60
```

```r
x1^2
```

```
## [1]   9  36  25   4   0 144
```

```
5/x1
```

```
## [1] 1.6666667 0.8333333 1.0000000 2.5000000       Inf 0.4166667
```

```
x2=c(x1, pi, sqrt(x1), 3*x1+2)
x2
```

```
##  [1]  3.000000  6.000000  5.000000  2.000000  0.000000 12.000000  3.141593
##  [8]  1.732051  2.449490  2.236068  1.414214  0.000000  3.464102 11.000000
## [15] 20.000000 17.000000  8.000000  2.000000 38.000000
```

Minimum, maximum, lenght of a vetor as well as sum of all vector components can be obtained by using the commends min(), max() and lenght() respectively.

```
min(x2)
```

```
## [1] 0
```

```
max(x2)
```

```
## [1] 38
```

```
length(x2)
```

```
## [1] 19
```

```
sum(x2)
```

```
## [1] 138.4375
```

Vectors of the same size can be added, subtracted, multiplied and divided.

```
b=c(3:8)
d=c(10:15)
b+d
```

```
## [1] 13 15 17 19 21 23
```

```
b-d
```

```
## [1] -7 -7 -7 -7 -7 -7
```

```
b*d
```

```
## [1]  30  44  60  78  98 120
```

```
b/d
```

```
## [1] 0.3000000 0.3636364 0.4166667 0.4615385 0.5000000 0.5333333
```

```
d^b
```

```
## [1]       1000      14641     248832    4826809  105413504 2562890625
```

### Accessing entries in a vector

Entries in vectors are found with x[j], i.e. by asking the j-th component of the vector x.

```
x1
```

```
## [1]  3  6  5  2  0 12
```

```
x1[1]
```

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

```r
x1[3]
```

```
## [1] 5
```

```r
x1[2:4]
```

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

```r
x1[8]
```

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

```r
x1[c(1,2)]
```

```
## [1] 3 6
```

```r
x1[c(1,3,5)]
```

```
## [1] 3 5 0
```

The same entry can be accesses multiple times.

```r
x1
```

```
## [1]  3  6  5  2  0 12
```

```r
x1[c(2,2,2,3)]
```

```
## [1] 6 6 6 5
```

```r
x1[3:7]
```

```
## [1]  5  2  0 12 NA
```

### Logical vectors

```r
k=rnorm(20)
k
```

```
##  [1]  1.50350498  1.36965700  1.41527376 -0.95015487 -0.60623215
##  [6]  0.05444119  0.27429067  0.02919109  0.86532904  1.64577427
## [11] -0.25578969 -0.06298012 -0.40578881  0.94287539  0.62096451
## [16]  0.72672259 -0.48639989 -0.14648467 -1.40717006  0.35384243
```

```r
pos=k>0
pos
```

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

```r
k.pos=k[pos]
k.pos
```

```
##  [1] 1.50350498 1.36965700 1.41527376 0.05444119 0.27429067 0.02919109
##  [7] 0.86532904 1.64577427 0.94287539 0.62096451 0.72672259 0.35384243
```

```r
k[k>0]
```

```
##  [1] 1.50350498 1.36965700 1.41527376 0.05444119 0.27429067 0.02919109
##  [7] 0.86532904 1.64577427 0.94287539 0.62096451 0.72672259 0.35384243
```

```
l=seq(10)
l
```

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

```
l==3
```

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

```
l[l==3]
```

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

```
l[l!=3]
```

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

```
l!=3 & l > 4
```

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

```
l[l!=3 & l > 4]
```

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

```
l.3= l==3
l.3
```

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

```
!l.3
```

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

### Functions of vectors

R has all the standard functions appearing in analysis. Most of them can be used on vectors. Also, for example, sum, mean, standard deviation, variance can be calculated in R.

```
sin(5)
```

```
## [1] -0.9589243
```

```
cos(1.3)
```

```
## [1] 0.2674988
```

```
pi
```

```
## [1] 3.141593
```

```
sin(pi/2)
```

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

```
cos(3*pi)
```

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

```
exp(1)
```

```
## [1] 2.718282
```

```
exp(-1)
```

```
## [1] 0.3678794
```

```
s=c(-2:2)
sin(s)
```

```
## [1] -0.9092974 -0.8414710  0.0000000  0.8414710  0.9092974
```

```
exp(s)
```

```
## [1] 0.1353353 0.3678794 1.0000000 2.7182818 7.3890561
```

```
sum(s)
```

```
## [1] 0
```

```
sum(s^2)
```

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

```
mean(s)
```

```
## [1] 0
```

```
sd(s)
```

```
## [1] 1.581139
```

```
var(s)
```

```
## [1] 2.5
```

## Matrices

Matrices are obtained by arranging the entries in a vector into rows and columns. For example, a vector of length 12 can be arranged as a 2x6 or a 6x2 matrix.

```
t=1:12
t
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
```

```
M1= matrix(t,2)
M1
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    3    5    7    9   11
## [2,]    2    4    6    8   10   12
```

```
M2=matrix(t, 6)
M2
```

```
##      [,1] [,2]
## [1,]    1    7
## [2,]    2    8
## [3,]    3    9
## [4,]    4   10
## [5,]    5   11
## [6,]    6   12
```

In the previous examples the matrices are built one column at a time, i.e. the vector 1 to 12 runs in sequence down the columns. They can b also build by rows.

```
M3=matrix(t, 2, byrow=TRUE)
M3
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    2    3    4    5    6
## [2,]    7    8    9   10   11   12
```

This could be also done by applying the transpose matrix operation.

```
M4=t(M2)
M4
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    2    3    4    5    6
## [2,]    7    8    9   10   11   12
```

```
M3==M4
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,] TRUE TRUE TRUE TRUE TRUE TRUE
## [2,] TRUE TRUE TRUE TRUE TRUE TRUE
```

### Accessing entries in matrices

The square bracket notation is used by giving both the row and column indices.

```
M3[1,1]
```

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

```
M3[2,3]
```

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

### Sums and means on matrices

To sum each of the columns we use the colSums function.

```
colSums(M3)
```

```
## [1]  8 10 12 14 16 18
```

```
rowSums(M3)
```

```
## [1] 21 57
```

```
colMeans(M3)
```

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

```
rowMeans(M3)
```

```
## [1] 3.5 9.5
```

## Data sets

Data sets available in R.

```r
data ()
data(package="rpart")
head(trees)
```

```
##   Girth Height Volume
## 1   8.3     70   10.3
## 2   8.6     65   10.3
## 3   8.8     63   10.2
## 4  10.5     72   16.4
## 5  10.7     81   18.8
## 6  10.8     83   19.7
```

```r
tail(trees)
```

```
##    Girth Height Volume
## 26  17.3     81   55.4
## 27  17.5     82   55.7
## 28  17.9     80   58.3
## 29  18.0     80   51.5
## 30  18.0     80   51.0
## 31  20.6     87   77.0
```

```r
Girth=trees$Girth
Height=trees$Height
Volume=trees$Volume
trees2=data.frame(Girth, Height, Volume)
head(trees2)
```

```
##   Girth Height Volume
## 1   8.3     70   10.3
## 2   8.6     65   10.3
## 3   8.8     63   10.2
## 4  10.5     72   16.4
## 5  10.7     81   18.8
## 6  10.8     83   19.7
```

```r
trees2['Volume']
```

```
##    Volume
## 1    10.3
## 2    10.3
## 3    10.2
## 4    16.4
## 5    18.8
## 6    19.7
## 7    15.6
## 8    18.2
## 9    22.6
## 10   19.9
## 11   24.2
## 12   21.0
## 13   21.4
## 14   21.3
## 15   19.1
## 16   22.2
## 17   33.8
## 18   27.4
## 19   25.7
## 20   24.9
## 21   34.5
## 22   31.7
## 23   36.3
## 24   38.3
## 25   42.6
## 26   55.4
## 27   55.7
## 28   58.3
## 29   51.5
## 30   51.0
## 31   77.0
```

```
trees2[c('Volume','Girth')]
```

```
##    Volume Girth
## 1    10.3   8.3
## 2    10.3   8.6
## 3    10.2   8.8
## 4    16.4  10.5
## 5    18.8  10.7
## 6    19.7  10.8
## 7    15.6  11.0
## 8    18.2  11.0
## 9    22.6  11.1
## 10   19.9  11.2
## 11   24.2  11.3
## 12   21.0  11.4
## 13   21.4  11.4
## 14   21.3  11.7
## 15   19.1  12.0
## 16   22.2  12.9
## 17   33.8  12.9
## 18   27.4  13.3
## 19   25.7  13.7
## 20   24.9  13.8
## 21   34.5  14.0
## 22   31.7  14.2
## 23   36.3  14.5
## 24   38.3  16.0
## 25   42.6  16.3
## 26   55.4  17.3
## 27   55.7  17.5
## 28   58.3  17.9
## 29   51.5  18.0
## 30   51.0  18.0
## 31   77.0  20.6
```

Data sets can be read from a table

```r
# >read.table()
```

## Functions and loops

R is programming language. Some simple examples of functions are presented.

```r
x=rnorm(20)
if(length(x[x<0])>10) print ('more than 10') else print ('< = 10')
```

```
## [1] "< = 10"
```

```r
x[x<0]
```

```
## [1] -1.30637755 -0.39883484 -2.00370795 -0.36388128 -0.64751790 -0.02651385
## [7] -0.18365301 -0.40999046 -1.82128821
```

```r
length(x[x<0])
```

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

```r
x=1:10
y=numeric()
for(t in x) y=c(y, t^2)
y
```

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

```r
for (t in x) print(t^2)
```

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

```r
x=0
repeat {x=x+1; print(x); if(x>10) break}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
```

```r
x=0
while(x<10){x=x+1; print(x)}
```

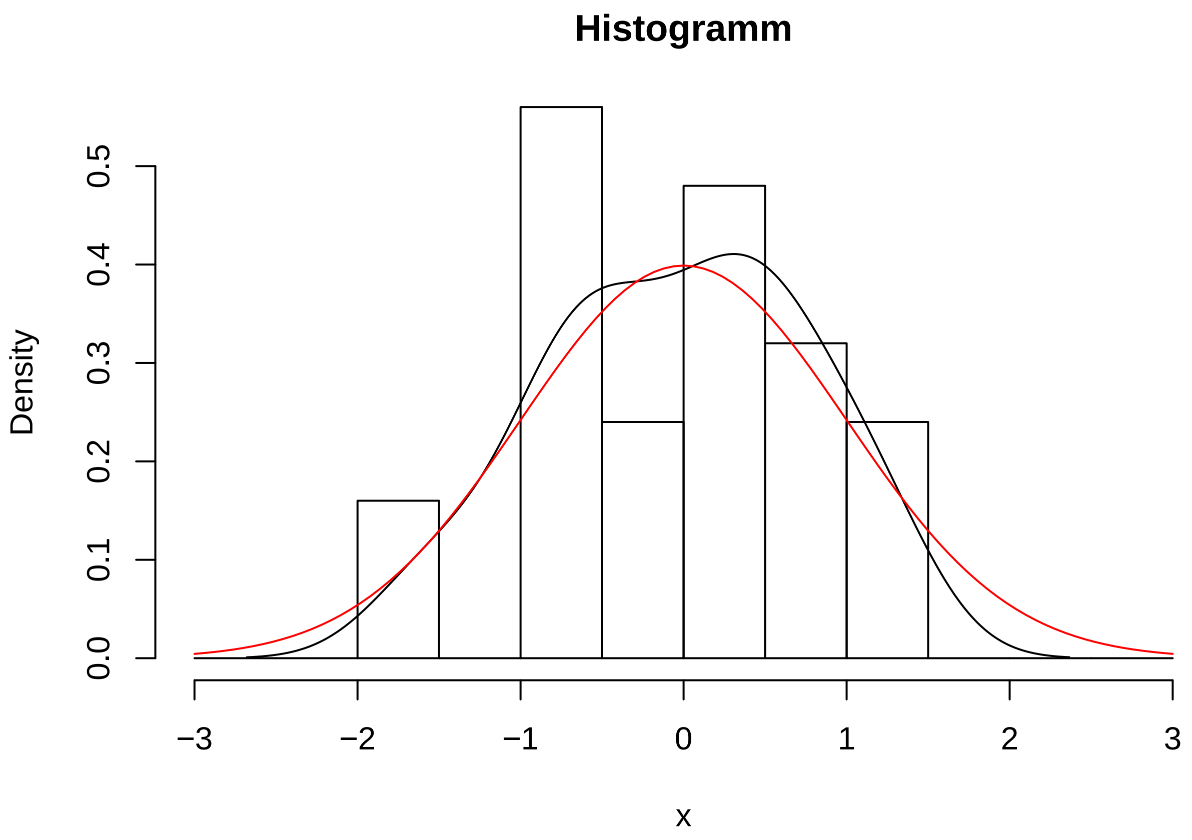
```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

```r
sum=function(x){y=0; for(t in x) y=y+t; y}
sum(1:45)
```

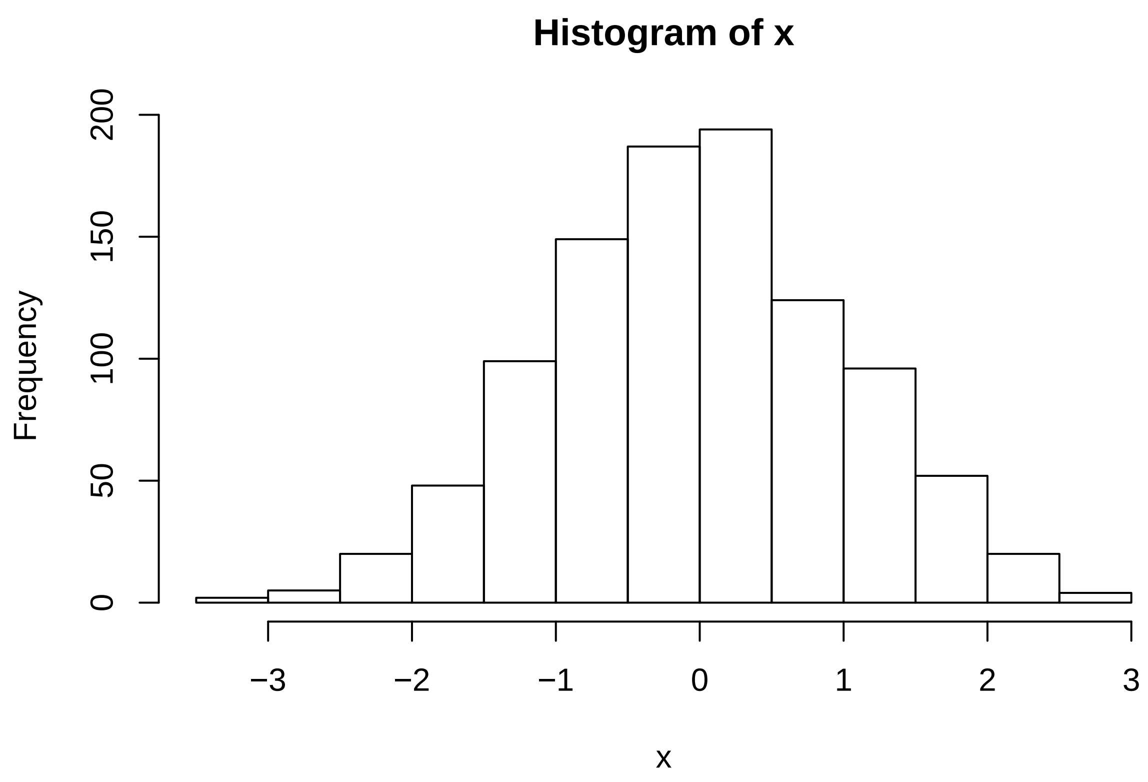
```
## [1] 1035
```

### Plots

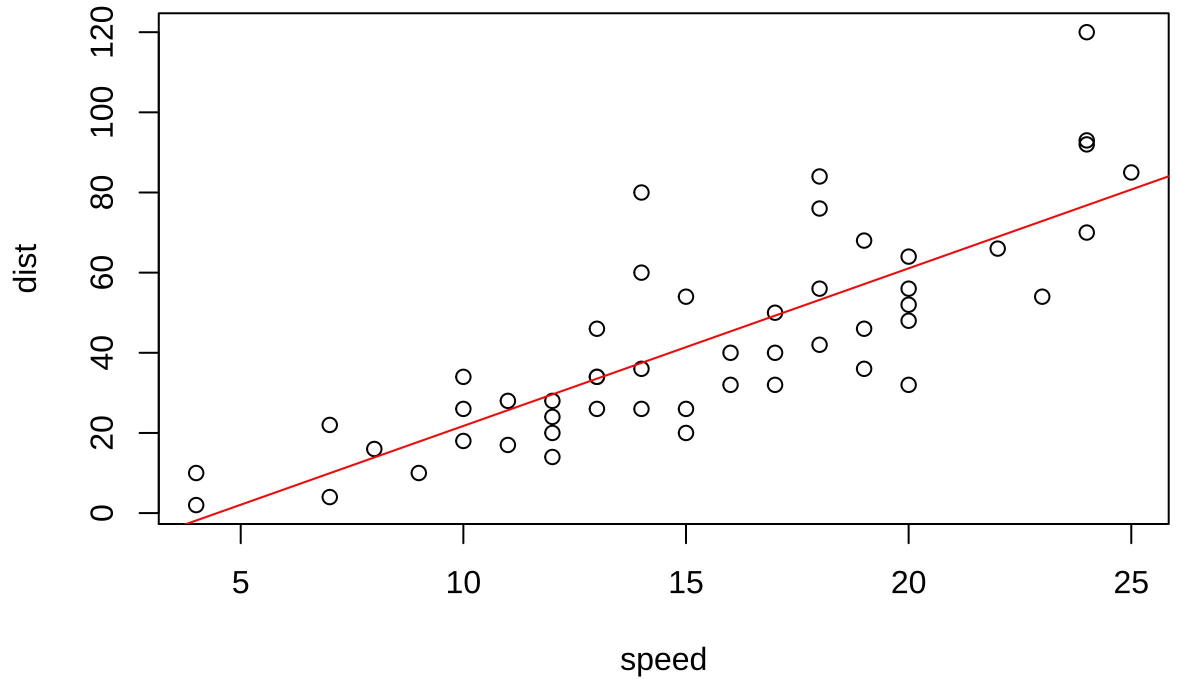
```r
x <- rnorm(25)
par(mar = c(4, 4, 3, 1))
hist(x, freq = FALSE, breaks = (-6:6)/2, main = "Histogramm")
lines(density(x))
curve(dnorm, -3, 3, add = TRUE, col = "red")
```



```
x <- rnorm(1000)
hist(x)
```



```
data(cars)
attach(cars)
plot(speed, dist)
a <- lm(dist ~ speed)
abline(a, col = "red")
```



## Random numbers

Random numbers can be gernerated in R by using the function sample(). This is used to run simulations of experiments with random outcomes. Namely, sample(k,n)generates a random permutation of k objects from the vector x, i.e. all k choices are different.

```
x=1:8
x
```

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

```
sample(x,5) #randomly sample 5 of the elements of x (every element is chosen at most once)
```

```
## [1] 8 5 6 3 4
```

```
sample(x,5)
```

```
## [1] 8 4 6 7 2
```

```
sample(x,5)
```

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

```
sample(x,8) #random permutation of all 8 elements
```

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

```
sample(x,8)
```

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

### Allowing repeated elements

We can get repeated values by setting the optional argument replace=TRUE.

```
sample(x,4,replace=TRUE)
```

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

```
sample(x,4,replace=TRUE)
```

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

```
sample(x,5,replace=TRUE)
```

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

```
sample(x,5,replace=TRUE)
```

```
## [1] 7 7 4 5 1
```

```
sample(x,12,replace=TRUE)
```

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

To generate an m x n array of random values we can use the function sample function followed by the matrix function. For example, in order to simulate rolling a die, we use 1:6 to make vector (1,2,3,4,5,6). We generate a 2 x 6 array of random dice rolls.

```
y = sample(1:6, 12, replace=TRUE)
matrix(y,nrow=2,ncol=6)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    5    5    6    4    3    4
## [2,]    5    5    4    1    5    4
```

One could make it a 3 x 4 array.

```
y = sample(1:6, 12, replace=TRUE)
matrix(y,nrow=3,ncol=4)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    4    6    5    5
## [2,]    1    2    3    4
## [3,]    6    3    2    5
```

### Simulation

Let us first simulate rolling one die 5 times and check if one of the rolls was 6.

```
s1 = sample(1:6, 5, replace=TRUE)
s1
```

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

```
s1==6
```

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

Let us now roll the die 5000 times and see what fraction of the rolls give 6. We expect about 1/6 of them.

```
s2 = sample(1:6, 1000, replace=TRUE)
sum(s2==6)
```

```
## [1] 173
```

```
sum(s2==6)/1000
```

```
## [1] 0.173
```

The probability of getting a sum of 10 when rolling two dice can be estimated in the following way:

```
n=1000 # number of trials
x = matrix(sample(1:6, 2*n, replace=TRUE), 2, n)
y=colSums(x)
mean(y==10)
```

```
## [1] 0.095
```

## Getting help

R and RStudio have complete documentation on all R functions. It is often faster to ask for help from the command line using a question mark. The result will appear in the help window.

```
?mean
```