R tutorial

Contents

0.0.1 Basic R commands

First function in R will ask you whether you want to save data and then it will close R:

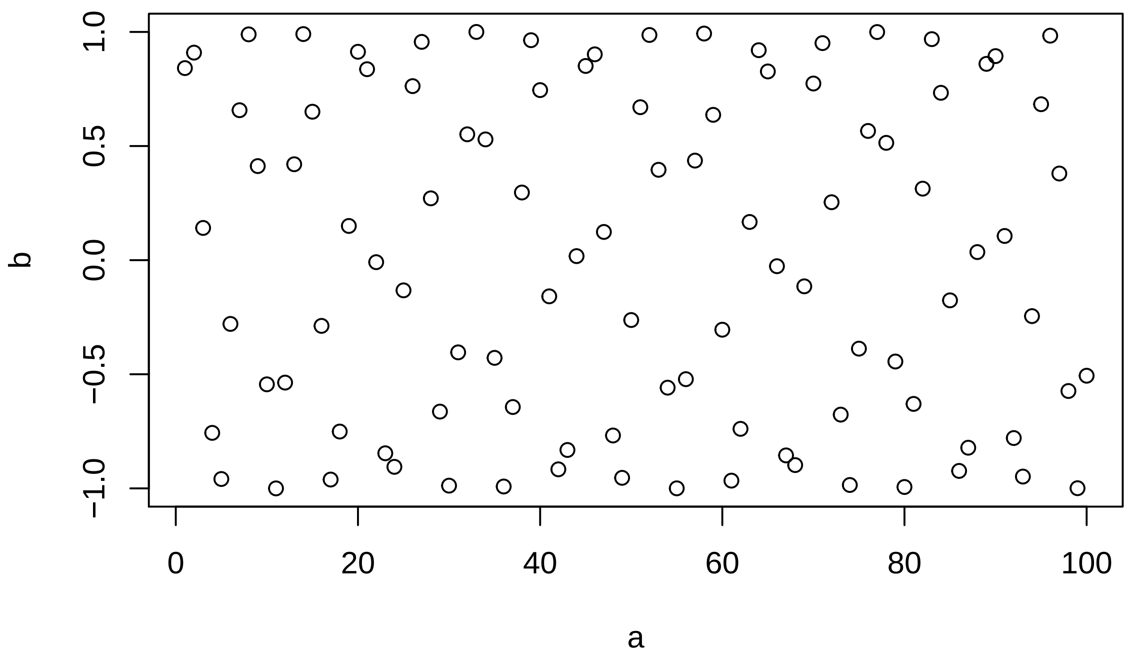
```
q()
```

This will close R without asking you:

```
q("n")
q(save="n")
```

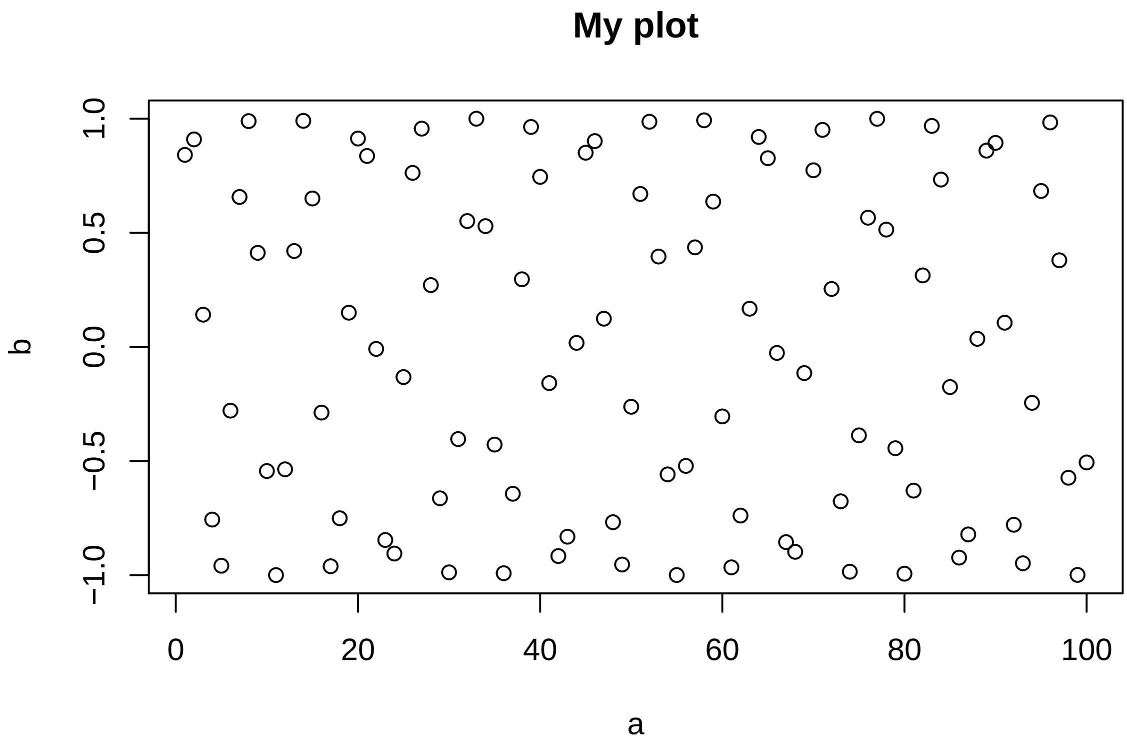
To show `plot` as an example of function, type:

```
a <- 1:100
b <- sin(a)
plot(a, b)
```



This will work:

```
plot(a, b, main="My plot")
```

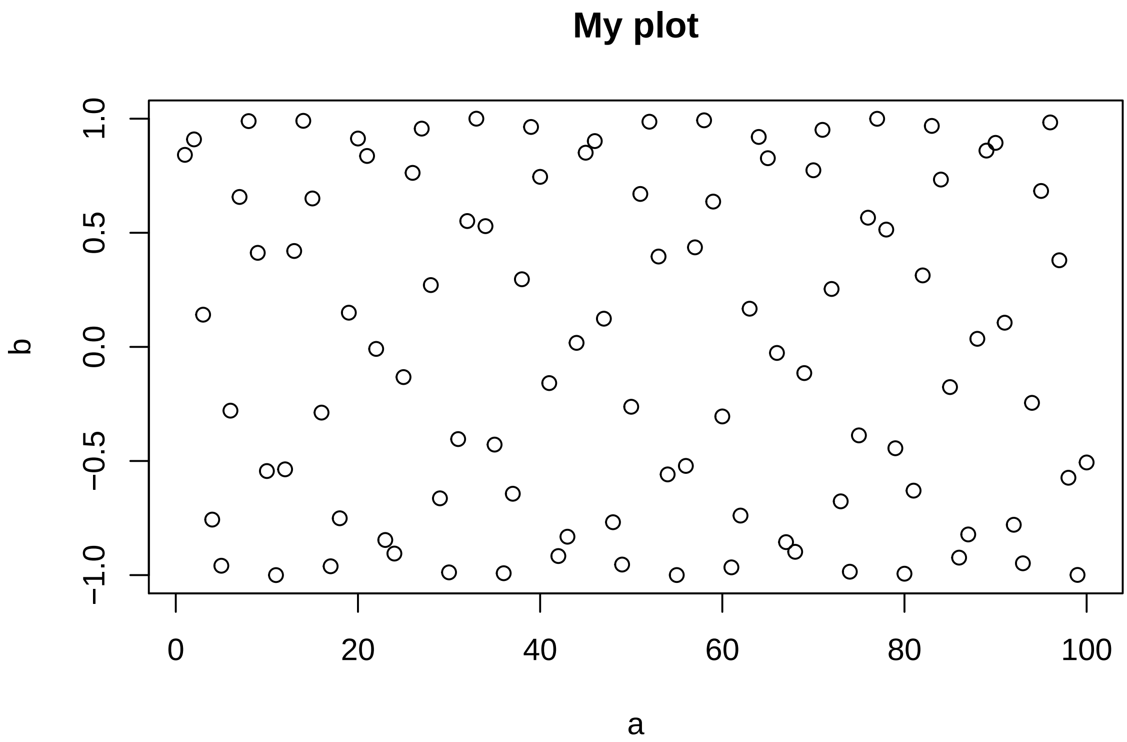


The R engine knows that first two parameters of the function `plot` are coordinates `x` and `y`, respectively. However, this will not work:

```
plot(a, b, "My plot")
```

because R does not know that `"My plot"` links to the parameter `main` (the main title). It is possible to specify all options:

```
plot(x=a, y=b, main="My plot")
```



Help could be found by:

```
help(plot)
?plot
apropos("svd")
help.search("svd")
```

Example or demo can be found as well:

```
example(image)
demo(graphics)
```

Basic operations:

```r
1+1
```

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

```r
2-1
```

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

```r
3*3
```

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

```r
6/3
```

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

R recognizes integers and floats from the context:

```r
5/2
```

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

```r
5.0/2.0
```

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

Power, modulo, integer division:

```r
3^3
```

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

```r
3**3
```

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

```r
5%%2
```

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

```r
5%/%2
```

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

R waits until command finished:

```r
1+
1
```

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

The hash sign `#` is used to add comments (code after `#` is ignored)

```r
1+1#+1
```

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

Spaces are ignored:

```r
1+1
```

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

```r
1 + 1
```

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

```r
1+         1
```

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

```r
1           +1
```

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

Basic constants and fuctions:

```r
pi
```

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

```r
cos(pi)
```

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

```r
sin(pi)
```

```
## [1] 1.224647e-16
```

```r
exp(1)
```

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

```r
abs(-4)
```

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

Logarithm is natural by default, decadic and binary are available as well:

```r
log(exp(2))
```

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

```r
log10(1000)
```

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

```r
log2(16)
```

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

Complex numbers are supported as well:

```r
2i
```

```
## [1] 0+2i
```

```r
2i*2i
```

```
## [1] -4+0i
```

One can assign a value to a variable:

```r
x <- 20
x
```

```
## [1] 20
```

```r
y <- 10
y
```

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

```r
x+y
```

```
## [1] 30
```

Logical and string variables are available as well:

```r
x<-FALSE
x
```

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

```r
y<-"string"
y
```

```
## [1] "string"
```

Equal works as well in most cases, but use `<-` to be on the safe side.

```r
x = 20
y = 10
x+y
```

```
## [1] 30
```

It is possible to change (e.g. increment) the value of a variable:

```r
x <- 10
x <- x + 1
x
```

```
## [1] 11
```

R recognizes capitals and small letters:

```r
a<-1
A<-2
a
```

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

```r
A
```

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

```r
a+A
```

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

Vectors can be defined by function `c()`:

```r
x <- c(1, 3, 2)
x
```

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

Vectors with increments of 1 can be easily generated by:

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

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

It can be used in the oposite way:

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

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

It is possible to print individual items from the vector:

```r
x <- c(1,5,2,3,4,7)
x
```

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

```r
x[1]
```

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

```r
x[2]
```

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

```r
x[3:6]
```

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

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

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

Another way how to make a vector:

```r
seq(from=6, to=21, by=2)
```

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

```r
rep((1:4), times=2)
```

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

```r
rep((1:4), each=2)
```

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

It is possible to add, subtract, multiply or divide a vector by a number:

```r
x<-1:5
x*2.5
```

```
## [1]  2.5  5.0  7.5 10.0 12.5
```

```r
x/2.5
```

```
## [1] 0.4 0.8 1.2 1.6 2.0
```

```r
x+2.5
```

```
## [1] 3.5 4.5 5.5 6.5 7.5
```

```r
x-2.5
```

```
## [1] -1.5 -0.5  0.5  1.5  2.5
```

It is possible to sum or multiply two vectors element-wise:

```r
x<-c(1,3,2)
y<-4:6
x+y
```

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

```r
x*y
```

```
## [1]  4 15 12
```

To get a dot product you need to type:

```r
x%*%y
```

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

You can apply a function item-wise on a vector:

```r
x<-1:4
exp(x)
```

```
## [1]  2.718282  7.389056 20.085537 54.598150
```

Matrix can be created by function `matrix` with two possible `byrow` option (`FALSE` is the default):

```r
x<-matrix(1:12, ncol=3)
x
```

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

```r
x<-matrix(1:12, ncol=3, byrow=TRUE)
x
```

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

Matrix can be transposes:

```r
t(x)
```

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

Matrix can be created by combining and adding columns or rows:

```r
x<-1:4
y<-c(3,2,6,5)
rbind(x, y)
```

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

```r
cbind(x, y)
```

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

Matrix elements can be accessed by indexes in square brackets (`[1,]` is for the first row, `[,1]` for the first column):

```r
x<-1:4
y<-c(3,2,6,5)
xy <- cbind(x, y)
xy
```

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

```r
xy[1,]
```

```
## x y
## 1 3
```

```r
xy[1,1]
```

```
## x
## 1
```

```r
xy[1,2]
```

```
## y
## 3
```

```r
xy[,1]
```

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

```r
xy[,2]
```

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

Special object for data analysis in R is `data.frame`:

```r
x<-1:4
y<-c(3,2,6,5)
mydata <- data.frame(x,y)
mydata
```

```
##   x y
## 1 1 3
## 2 2 2
## 3 3 6
## 4 4 5
```

```
mydata[1]
```

```
##   x
## 1 1
## 2 2
## 3 3
## 4 4
```

```
mydata[2]
```

```
##   y
## 1 3
## 2 2
## 3 6
## 4 5
```

```
mydata$x
```

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

```
mydata[1,1]
```

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

```
mydata[2,1]
```

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

```
mydata[2,]
```

```
##   x y
## 2 2 2
```

Logical operations can such as negation, or and and can be used in operations with `data.frame`, lets show some:

```
x<-TRUE
!x
```

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

```
y<-FALSE
x|x
```

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

```
x|y
```

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

```
y|y
```

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

```
y&y
```

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

```
x&y
```

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

```
x&x
```

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

```
1<2
```

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

```r
1>2
```

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

```r
1==1
```

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

```r
1==2
```

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

Loops are not as important R as in classical programming languages, but it is possible to use them:

```r
for(i in 1:3) {
  print(i)
}
```

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

In R it is possible to define a new function:

```r
sinpluscos <- function(x) {
 y<-sin(x)+cos(x)
 return(y)
}
sin(1)+cos(1)
```

```
## [1] 1.381773
```

```r
sinpluscos(1)
```

```
## [1] 1.381773
```

R contains many pre-defined datasets:

```r
head(faithful)
```

```
##   eruptions waiting
## 1     3.600      79
## 2     1.800      54
## 3     3.333      74
## 4     2.283      62
## 5     4.533      85
## 6     2.883      55
```

```
data()
```

Beside standard R functions and datasets it is possible to install new packages from R repositories:

```
installed.packages()
install.packages("igraph")
library(igraph)
```

That's all, lets try to read some data in the next lesson and then we will analyze them.

0.0.1.1 Tips and tricks

- you don't remember exactly the command (e.g. you are not sure whether it is `len` or `length`)? Just type the beginning of the command and double-click tabulator. You can use it also for parameters of a function.
- if the command is too long you can separate it on two or more lines (R waits until the command is closed).
- it is better not to use trivial names for variables because you can overwrite some existing variables.

- to generate a vector with constant increment (e.g. 2, 4, 6 … 10):

```
x<-2*1:5
```

- you can reverse the order of a vector by:

```
x<-2*1:5
x[5:1]
```

```
## [1] 10  8  6  4  2
```

0.0.2 Reading and writing files in R

You can read data from unformated text files by function `read.table`:

```
mydata<-read.table("https://raw.githubusercontent.com/spiwokv/Rtutorial/master/data/mydata.txt")
mydata
```

```
##   V1 V2 V3
## 1  1  2  3
## 2  4  5  6
```

Alternatively you can download the file to a working directory of your computer and open it:

```
mydata<-read.table("mydata.txt")
```

If you don't know which directory is the working one, you can type:

```
getwd()
```

or you can change it by:

```
setwd("C:/Documents")
```

The function `read.table` can be modified by the parameter `header`, it indicates that the first line of the file contains names of columns. Separators can be changed by parameter `sep`. By default, `sep` is set to `""`, i.e. one or multiple spaces can act as separators. Special separators such as tabulator can be defined by regular expressions, e.g. `sep="\t"`.

It is also possible to control quoting characters, strings which are to be interpreted as `NA` (not available) values or comment character.

There are special functions to read formated files, such as `read.csv`, `read.csv2`, `read.delim`, `read.delim2`, `read.fwf` and `read.ftable`. It is also possible to read data from Microsoft Excel, databases (MySQL, SQLite, Oracle, Microsoft SQL Server and others) XML, JSON and other files.

Data can be written by the function `write.table`:

```
mydata<-read.table("https://raw.githubusercontent.com/spiwokv/Rtutorial/master/data/mydata.txt")
head(mydata)
```

```
##   V1 V2 V3
## 1  1  2  3
## 2  4  5  6
```

```
write.table(mydata, "mydata2.txt")
```

R also contains sample data for demonstration of functions. You can see the list by typing:

```
data()
```

For example `faithful` dataset contains waiting times between eruptions and the duration of eruptions of the Old Faithful geyser in Yellowstone National Park (Wyoming, USA).

```
head(faithful)
```

```
##   eruptions waiting
## 1     3.600      79
## 2     1.800      54
```

```
## 3      3.333      74
## 4      2.283      62
## 5      4.533      85
## 6      2.883      55
```

0.0.2.1 Tips and tricks

- data in a file may be large and it is not useful to print it when you want to check whether data was read correctly. You can use `head` function instead to print the header with the first six lines.

```
head(faithful)
```

```
##   eruptions waiting
## 1     3.600      79
## 2     1.800      54
## 3     3.333      74
## 4     2.283      62
## 5     4.533      85
## 6     2.883      55
```

Analogously you can use the `tail` function. ### Analyzing data in R

We can show how to extract and manipulate data from a dataset. Let us create a simple "dataset" containing names:

```
jmena <- c("Karina","Radmila","Diana","Dalimil","Melichar","Vilma","Cestmir","Vladan","Bretislav")
```

It is possible to iterate this vector of names by typing square brackets with element ID (R counts from 1, not from 0). To get the first name type:

```
jmena[1]
```

```
## [1] "Karina"
```

(returns Karina). You can evaluate some expression within the square brackets, e.g.:

```
jmena[1+1]
```

```
## [1] "Radmila"
```

returns Radmila, not Karina Karina. You can use colon operator:

```
jmena[1:3]
```

```
## [1] "Karina"  "Radmila" "Diana"
```

This returns first three names. You can use more complicated operation:

```
jmena[1:3*2]
```

```
## [1] "Radmila" "Dalimil" "Vilma"
```

This returns element number 2, 4 and 6. Negative indexes can be used to remove some element:

```
jmena[-1]
```

```
## [1] "Radmila"   "Diana"     "Dalimil"   "Melichar"  "Vilma"     "Cestmir"
## [7] "Vladan"    "Bretislav"
```

(returns all elements except the first one). The last element can be obtained by `jmena[9]` if you know the number of elements or `jmena[length(jmena)]` in case you are lazy to count them. To inverse the order of elements you can use the colon operator as `jmena[9:1]` or `jmena[length(jmena):1]`.

Let us move to more complicated dataset. You can load the results of municipal elections 2015 in Prague:

```
volby <- read.table("https://raw.githubusercontent.com/spiwokv/Rtutorial/master/data/volby2013prahaA
                    sep=";", header=T, dec=",")
```

Parameter `sep=";"` indicates that individual items are separated by a semicolon. The option `header=T` indicates that the first line contains names of columns. In fact the first line contains the remark, because it starts with `#`. The option `dec=","` tells us that the Czech decimal is used. For users who can handle Czech characters you can replace "volby2013prahaASCII.txt" by "volby2013praha.txt".

For the first inspection you can type:

```
head(volby)
```

```
##   partyno    party order                 name age   cendid affiliation
## 1      15     SPOZ    23         Abt Jaroslav  40     SPOZ        SPOZ
## 2       1     CSSD     3 Adamek Frantisek Bc.  58     CSSD        CSSD
## 3      16 OBC_2011    10    Adamek Karel Ing.  61 OBC_2011       BEZPP
## 4      16 OBC_2011     1        Adamek Ludvik  58 OBC_2011    OBC_2011
## 5      16 OBC_2011    11     Adamek Lukas Bc.  28 OBC_2011    OBC_2011
## 6       4   TOP 09     6 Adamova Marketa Ing.  29   TOP 09      TOP 09
##    abs  rel mandate manord
## 1    8 0.14       -      -
## 2 2492 3.01       *      4
## 3   12 2.63       -      -
## 4   33 7.25       -      -
## 5   12 2.63       -      -
## 6 3626 2.68       *      6
```

This prints the first 10 lines. You can chose different number by `n=2` option. Similarly you can print last lines by function `tail`.

Function:

```
dim(volby)
```

```
## [1] 680  11
```

prints the dimension (number of lines and colums). You can get these values separately for the number of lines and columns by functions `nrow` and `ncol`, respectively.

You can iterate on the lines and columns similarly to the previoius example. For example, you can select the first candidate in the alphabetic order as:

```
volby[1,]
```

```
##   partyno party order         name age cendid affiliation abs  rel mandate
## 1      15  SPOZ    23 Abt Jaroslav  40   SPOZ        SPOZ   8 0.14       -
##   manord
## 1      -
```

Columns can be selected by:

```
head(volby[,4])
```

```
## [1] Abt Jaroslav         Adamek Frantisek Bc. Adamek Karel Ing.
## [4] Adamek Ludvik        Adamek Lukas Bc.     Adamova Marketa Ing.
## 680 Levels: Abt Jaroslav Adamek Frantisek Bc. ... Zubaty Jan Ing.
```

This prints names of candidates.

The function:

```
names(volby)
```

```
##  [1] "partyno"     "party"       "order"       "name"        "age"
##  [6] "cendid"      "affiliation" "abs"         "rel"         "mandate"
## [11] "manord"
```

prints names of columns, such as party number and name, candidate's name and age etc.

As an alternative to `volby[,4]` you can use `$` operator followed by the name of the column:

```r
head(volby$name)
```

```
## [1] Abt Jaroslav         Adamek Frantisek Bc. Adamek Karel Ing.
## [4] Adamek Ludvik        Adamek Lukas Bc.     Adamova Marketa Ing.
## 680 Levels: Abt Jaroslav Adamek Frantisek Bc. ... Zubaty Jan Ing.
```

The function `levels` determines levels of a vector. For example, if you type:

```r
volby[,2]
```

```
##   [1] SPOZ     CSSD     OBC_2011 OBC_2011 OBC_2011 TOP 09   KSCM
##   [8] SPOZ     LEV 21   ANO 2011 HLVZHURU Pirati   CSSD     SZ
##  [15] KC       SsCR     Pirati   HLVZHURU ODS      KDU-CSL  SsCR
##  [22] Suveren. PB       Zmena    Svobodni SPOZ     Suveren. ODS
##  [29] CSSD     KC       ODS      ANO 2011 Usvit    OBC_2011 KSCM
##  [36] OBC_2011 PB       SZ       OBC_2011 KC       LEV 21   Zmena
##  [43] SPOZ     Suveren. ODS      ANO 2011 KC       TOP 09   LEV 21
##  [50] KC       Usvit    KC       SPOZ     KSCM     OBC_2011 HLVZHURU
##  [57] SPOZ     KDU-CSL  KC       DSSS     SZ       DSSS     KC
##  [64] SsCR     KDU-CSL  Zmena    SZ       PB       TOP 09   HLVZHURU
##  [71] PB       SsCR     Svobodni KC       Usvit    ODS      PB
##  [78] Usvit    ANO 2011 Suveren. KDU-CSL  KSCM     ODS      SsCR
##  [85] SsCR     KDU-CSL  Pirati   Zmena    LEV 21   TOP 09   DSSS
##  [92] Svobodni KSCM     SZ       TOP 09   SPOZ     Svobodni PB
##  [99] KDU-CSL  ODS      SZ       Usvit    SZ       PB       Usvit
## [106] LEV 21   DSSS     SPOZ     LEV 21   SPOZ     Suveren. Svobodni
## [113] SsCR     Zmena    Pirati   KSCM     Usvit    KC       Zmena
## [120] KDU-CSL  PB       ODS      ANO 2011 HLVZHURU Pirati   Zmena
## [127] Zmena    SsCR     PB       LEV 21   SPOZ     TOP 09   KSCM
## [134] ODS      PB       Suveren. KC       ODS      Usvit    KSCM
## [141] Suveren. Zmena    KC       ODS      Usvit    DSSS     KSCM
## [148] ODS      ANO 2011 LEV 21   Zmena    DSSS     TOP 09   CSSD
## [155] DSSS     CSSD     Svobodni ODS      LEV 21   Svobodni HLVZHURU
## [162] PB       SPOZ     DSSS     Svobodni CSSD     PB       SsCR
## [169] KC       Zmena    KDU-CSL  Zmena    SZ       DSSS     KDU-CSL
## [176] Suveren. SsCR     Suveren. CSSD     KC       KSCM     ANO 2011
## [183] Suveren. DSSS     Pirati   Svobodni Usvit    KSCM     LEV 21
## [190] KSCM     KDU-CSL  KC       SZ       SsCR     ANO 2011 SZ
## [197] KSCM     Zmena    KSCM     KSCM     KDU-CSL  SZ       KSCM
## [204] TOP 09   Suveren. LEV 21   Usvit    HLVZHURU SPOZ     Svobodni
## [211] ODS      HLVZHURU SPOZ     ANO 2011 ODS      KSCM     ODS
## [218] SZ       ANO 2011 Usvit    DSSS     ODS      KDU-CSL  LEV 21
## [225] TOP 09   SsCR     HLVZHURU Suveren. KDU-CSL  Pirati   OBC_2011
## [232] Suveren. KC       HLVZHURU SPOZ     Zmena    PB       KC
## [239] LEV 21   SZ       Svobodni SsCR     Pirati   HLVZHURU Suveren.
## [246] Usvit    Zmena    CSSD     SPOZ     KDU-CSL  HLVZHURU Suveren.
## [253] Suveren. ODS      ODS      KSCM     KC       SZ       SZ
## [260] SsCR     Suveren. SPOZ     TOP 09   TOP 09   DSSS     ANO 2011
## [267] ODS      CSSD     Zmena    ODS      KC       KSCM     SPOZ
## [274] SZ       Svobodni TOP 09   ODS      Usvit    KSCM     KDU-CSL
## [281] Suveren. Svobodni KC       HLVZHURU SZ       Pirati   HLVZHURU
## [288] KC       Usvit    CSSD     PB       TOP 09   SPOZ     Usvit
## [295] DSSS     HLVZHURU KC       Svobodni SsCR     KDU-CSL  KDU-CSL
## [302] HLVZHURU PB       KSCM     ANO 2011 Svobodni PB       KSCM
## [309] Zmena    Zmena    ANO 2011 ANO 2011 TOP 09   TOP 09   LEV 21
## [316] KSCM     KDU-CSL  SPOZ     Zmena    LEV 21   HLVZHURU KC
## [323] Svobodni SsCR     Usvit    SZ       KSCM     PB       HLVZHURU
## [330] ODS      Usvit    PB       LEV 21   KC       KC       KSCM
## [337] PB       SPOZ     Pirati   Svobodni PB       CSSD     ODS
```

```
## [344] LEV 21   CSSD     SsCR     KDU-CSL  CSSD     KDU-CSL  KC
## [351] Usvit    SPOZ     OBC_2011 CSSD     PB       Pirati   Suveren.
## [358] Zmena    PB       ANO 2011 ODS      SsCR     CSSD     Pirati
## [365] ANO 2011 Pirati   PB       Pirati   Svobodni LEV 21   Svobodni
## [372] SZ       Usvit    DSSS     KSCM     LEV 21   DSSS     KDU-CSL
## [379] Suveren. LEV 21   PB       TOP 09   SsCR     TOP 09   SsCR
## [386] HLVZHURU LEV 21   SZ       ODS      Suveren. KC       Suveren.
## [393] SPOZ     KSCM     Zmena    PB       SZ       SPOZ     Usvit
## [400] Suveren. CSSD     LEV 21   ANO 2011 DSSS     TOP 09   KC
## [407] LEV 21   HLVZHURU Zmena    CSSD     ANO 2011 KC       CSSD
## [414] Zmena    SsCR     TOP 09   ODS      ODS      HLVZHURU HLVZHURU
## [421] KDU-CSL  TOP 09   KC       ANO 2011 LEV 21   ODS      Svobodni
## [428] CSSD     KSCM     TOP 09   HLVZHURU Zmena    CSSD     Usvit
## [435] DSSS     KDU-CSL  SZ       KDU-CSL  Suveren. ANO 2011 TOP 09
## [442] PB       Pirati   SZ       LEV 21   ANO 2011 ODS      Pirati
## [449] Svobodni TOP 09   LEV 21   PB       Svobodni SsCR     Pirati
## [456] KC       Suveren. KDU-CSL  HLVZHURU ANO 2011 LEV 21   PB
## [463] SsCR     SPOZ     Zmena    DSSS     LEV 21   HLVZHURU ANO 2011
## [470] Suveren. Suveren. SZ       Zmena    KC       KDU-CSL  ODS
## [477] OBC_2011 Zmena    SZ       SZ       SsCR     SZ       CSSD
## [484] CSSD     Zmena    LEV 21   KDU-CSL  Suveren. SZ       Usvit
## [491] ANO 2011 Svobodni Svobodni ANO 2011 KC       ANO 2011 Suveren.
## [498] KDU-CSL  KDU-CSL  HLVZHURU Svobodni SPOZ     Suveren. KSCM
## [505] ODS      TOP 09   TOP 09   Zmena    Svobodni KSCM     TOP 09
## [512] ANO 2011 Usvit    Svobodni Zmena    SsCR     SPOZ     Zmena
## [519] Usvit    DSSS     HLVZHURU TOP 09   Usvit    Usvit    KC
## [526] Usvit    Zmena    PB       TOP 09   Svobodni Svobodni Zmena
## [533] SZ       Svobodni Suveren. CSSD     Pirati   SsCR     Pirati
## [540] ODS      CSSD     SsCR     KDU-CSL  Svobodni KDU-CSL  DSSS
## [547] DSSS     KSCM     SsCR     Usvit    Zmena    Usvit    SsCR
## [554] Usvit    KDU-CSL  SZ       SPOZ     KSCM     HLVZHURU KSCM
## [561] SPOZ     ANO 2011 CSSD     DSSS     PB       DSSS     TOP 09
## [568] PB       PB       LEV 21   ANO 2011 Usvit    ANO 2011 CSSD
## [575] Svobodni KC       DSSS     KSCM     Svobodni HLVZHURU LEV 21
## [582] HLVZHURU Svobodni DSSS     HLVZHURU KSCM     SsCR     ODS
## [589] Zmena    SPOZ     PB       KC       SZ       Usvit    LEV 21
## [596] KDU-CSL  CSSD     SZ       KDU-CSL  DSSS     TOP 09   TOP 09
## [603] HLVZHURU SsCR     CSSD     SPOZ     SPOZ     CSSD     LEV 21
## [610] TOP 09   ANO 2011 PB       DSSS     TOP 09   HLVZHURU Suveren.
## [617] Suveren. LEV 21   ANO 2011 ANO 2011 Pirati   SZ       PB
## [624] Svobodni PB       ODS      Svobodni ODS      TOP 09   SPOZ
## [631] CSSD     SPOZ     CSSD     HLVZHURU Suveren. SsCR     DSSS
## [638] DSSS     DSSS     DSSS     Usvit    SsCR     TOP 09   TOP 09
## [645] CSSD     SPOZ     SPOZ     CSSD     LEV 21   Suveren. LEV 21
## [652] KSCM     SsCR     SsCR     SsCR     OBC_2011 Pirati   SZ
## [659] Suveren. ANO 2011 KDU-CSL  DSSS     CSSD     SZ       CSSD
## [666] KDU-CSL  DSSS     HLVZHURU DSSS     DSSS     Zmena    ANO 2011
## [673] CSSD     Usvit    ANO 2011 DSSS     HLVZHURU SZ       SPOZ
## [680] Usvit
## 20 Levels: ANO 2011 CSSD DSSS HLVZHURU KC KDU-CSL KSCM LEV 21 ... Zmena
```

it will print a vector with 680 items (one for each candidate) with political parties of candidates in the alphabet order of their family names. If you place this into the function `levels` it will print each party only once:

```
levels(volby[,2])
```

```
##  [1] "ANO 2011" "CSSD"     "DSSS"     "HLVZHURU" "KC"       "KDU-CSL"
##  [7] "KSCM"     "LEV 21"   "OBC_2011" "ODS"      "PB"       "Pirati"
```

```
## [13] "SPOZ"     "SsCR"     "Suveren." "Svobodni" "SZ"       "TOP 09"
## [19] "Usvit"    "Zmena"
```

By function `nlevels` you can get the number of political parties. Function `table` will print a table with numbers of candidates per party:

```
table(volby[,2])
```

```
##
## ANO 2011     CSSD     DSSS HLVZHURU       KC  KDU-CSL     KSCM   LEV 21
##       36       36       36       36       36       36       36       36
## OBC_2011      ODS       PB   Pirati     SPOZ     SsCR Suveren. Svobodni
##       11       36       36       21       36       36       36       36
##       SZ   TOP 09    Usvit    Zmena
##       36       36       36       36
```

In order to print only the lines containing candidates of "Piráti" you can use following expressions: The expression `volby[,2]` will print parties in the alphabet order of names of candidates. You can extend it by `volby[,2]=="Pirati"`. This will return the series of `TRUE` and `FALSE` values in the same order. For example, first 16 candidates in alphabet were not Pirates, so first 16 values are `FALSE`. The candidate number 17 is Pirate, so output number 17 is `TRUE`. You can apply `sum` function on the output. This function counts `TRUE` as 1 and `FALSE` as 0.

If you insert the previous expression `volby[,2]=="Pirati"` into the square brackets of `volby[]` you can select lines containing Pirates:

```
volby[volby[,2]=="Pirati",]
```

```
##     partyno  party order                         name age cendid
## 12        3 Pirati    20               Bakovsky Pavel  21 Pirati
## 17        3 Pirati     1        Bartos Ivan PhDr. PhD  33 Pirati
## 87        3 Pirati     6              Derer Ivan Ing.  46 Pirati
## 115       3 Pirati    15       Esner Vladislav Tobias  31 Pirati
## 125       3 Pirati    13               Findeis Lukas  28 Pirati
## 185       3 Pirati    17          Holovsky Sebastian  28 Pirati
## 230       3 Pirati     8              Kallasch Ondrej  22 Pirati
## 243       3 Pirati    19          Kheck Patrick Mgr.  32 Pirati
## 286       3 Pirati     9           Krausova Michaela  21 Pirati
## 339       3 Pirati    10               Mahrik Viktor  31 Pirati
## 356       3 Pirati    21              Matousek Josef  57 Pirati
## 364       3 Pirati     7              Michailidu Jana  23 Pirati
## 366       3 Pirati     3      Michalek Jakub Mgr. Bc.  24 Pirati
## 368       3 Pirati    11           Mikolas Ivan Ing.  48 Pirati
## 443       3 Pirati    12          Podhajsky Jan Mgr.  37 Pirati
## 448       3 Pirati    14               Polak Michael  39 Pirati
## 455       3 Pirati     4              Profant Ondrej  25 Pirati
## 537       3 Pirati    16          Sura Ales Ing. Bc.  36 Pirati
## 539       3 Pirati    18 Svetnicka Karel Mgr. et Bc.  53 Pirati
## 621       3 Pirati     5              Veverka Robert  36 Pirati
## 657       3 Pirati     2        Wagnerova Lenka PhDr.  52 Pirati
##     affiliation  abs   rel mandate manord
## 12       Pirati  314  1.47       -      -
## 17       Pirati 3772 17.68       -      -
## 87        BEZPP  416  1.95       -      -
## 115      Pirati   59  0.27       -      -
## 125      Pirati   54  0.25       -      -
## 185      Pirati  161  0.75       -      -
## 230      Pirati  257  1.20       -      -
## 243       BEZPP   89  0.41       -      -
## 286      Pirati  524  2.45       -      -
## 339      Pirati  216  1.01       -      -
```

```
## 356       BEZPP  450  2.11       -       -
## 364      Pirati  371  1.73       -       -
## 366      Pirati 1807  8.47       -       -
## 368      Pirati  249  1.16       -       -
## 443      Pirati  348  1.63       -       -
## 448      Pirati  201  0.94       -       -
## 455      Pirati  750  3.51       -       -
## 537      Pirati  225  1.05       -       -
## 539       BEZPP  208  0.97       -       -
## 621       BEZPP 1085  5.08       -       -
## 657      Pirati 1439  6.74       -       -
```

The square brackets contain comma inside, because we select lines or columns. The expression `volby[,2]=="Pirati"` is in front of the comma because we select lines. Lines with `TRUE` as the output of `volby[,2]=="Pirati"` are printed, others are not printed.

Let us look at numbers of votes in the column number 8. We can check the range by function `range`:

```
range(volby[,8])
```

```
## [1]     0 37794
```

This shows that the least successful candidate was not voted at all, the most successful got 37794 votes. You can print all votes sorted by the function `sort`:

```
head(sort(volby[,8]))
```

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

To get the reverse order use option:

```
head(sort(volby[,8], decreasing=TRUE))
```

```
## [1] 37794 26866 18955 14953 10590  9004
```

You may be interested who scored the best and worst in elections. You can use function `order`. This function prints the index of the lowest value, the index of the second lowest value and so forth. The expression:

```
head(volby[order(volby[,8]),])
```

```
##     partyno  party order             name age cendid affiliation abs  rel
## 341      12     PB    18  Machacek Vaclav  36     PB       BEZPP   0 0.00
## 467      22 LEV 21    32    Pucholt Milan  42 LEV 21      LEV 21   0 0.00
## 553      10   SsCR    30    Sefl Ladislav  64   SsCR        SsCR   0 0.00
## 77       12     PB    17 Cernota Miroslav  55     PB       BEZPP   1 0.08
## 121      12     PB    29 Fiedler Ladislav  45     PB       BEZPP   1 0.08
## 159      22 LEV 21    16    Harvanek Petr  64 LEV 21      LEV 21   1 0.26
##     mandate manord
## 341       -      -
## 467       -      -
## 553       -      -
## 77        -      -
## 121       -      -
## 159       -      -
```

will print the table sorted by the number of votes from the lowest to highest. You can revert the order by option `decreasing=T` in the order function.

Finally, we are interested in number of votes for each party. This can be obtained manually, party by party as:

```
sum(volby[volby[,2]=="Pirati",8])
```

```
## [1] 12995
```

and so forth for each party. As an alternative you can use function `aggregate`:

```
aggregate(x=volby[,8], by=list(volby[,2]), FUN=sum)
```

```
##     Group.1     x
## 1  ANO 2011 47723
## 2      CSSD 55212
## 3      DSSS  1039
## 4  HLVZHURU  1782
## 5        KC   638
## 6   KDU-CSL 36769
## 7      KSCM 32534
## 8    LEV 21   224
## 9  OBC_2011   260
## 10      ODS 91977
## 11       PB   480
## 12   Pirati 12995
## 13     SPOZ  3473
## 14     SsCR   457
## 15 Suveren.   400
## 16 Svobodni 11511
## 17       SZ 39345
## 18   TOP 09 95033
## 19    Usvit  8170
## 20    Zmena  3035
```

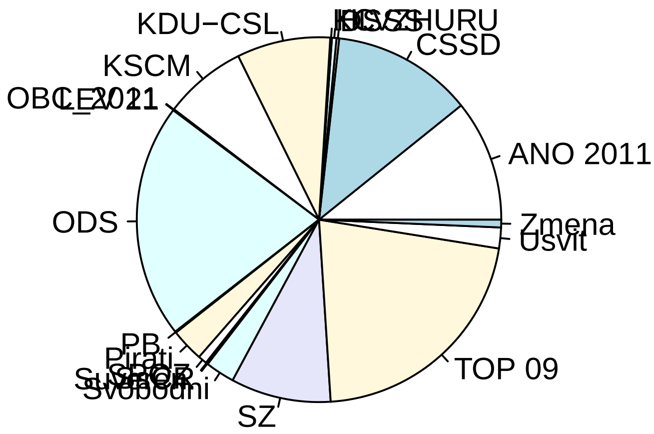
The function `list` is used because votes can be aggregated by multiple factors. Instead of function `sum` you can use other functions, for example average age of each party can be printed by:

```
aggregate(x=volby[,5], by=list(volby[,2]), FUN=mean)
```

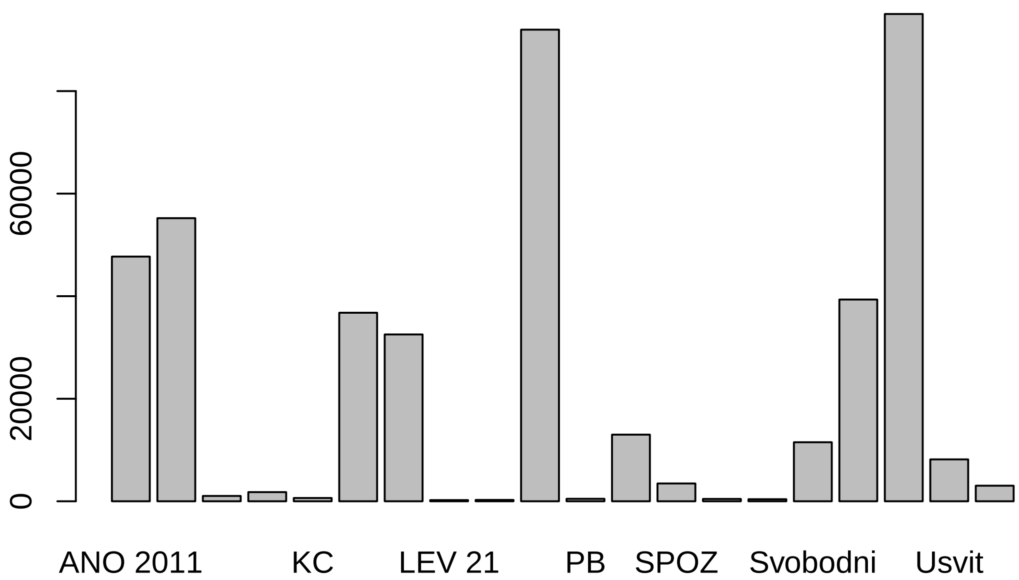
```
##     Group.1        x
## 1  ANO 2011 44.11111
## 2      CSSD 50.55556
## 3      DSSS 42.41667
## 4  HLVZHURU 42.41667
## 5        KC 54.33333
## 6   KDU-CSL 45.83333
## 7      KSCM 51.86111
## 8    LEV 21 53.86111
## 9  OBC_2011 49.36364
## 10      ODS 47.02778
## 11       PB 47.52778
## 12   Pirati 34.42857
## 13     SPOZ 47.08333
## 14     SsCR 52.63889
## 15 Suveren. 48.44444
## 16 Svobodni 38.50000
## 17       SZ 40.88889
## 18   TOP 09 44.33333
## 19    Usvit 45.58333
## 20    Zmena 44.19444
```

You can plot numbers of votes as a pie chart or bar plot:

```
vysledky<-aggregate(x=volby[,8], by=list(volby[, 2]), FUN=sum)
pie(vysledky[,2], labels=vysledky[,1])
```




```r
barplot(vysledky[,2], names.arg=vysledky[,1])
```



0.0.2.2 Tips and tricks

There is a family of "apply" functions. To calculate a sum for each row of an array or matrix use:

```
myarray <- matrix(1:10, nrow=5)
apply(myarray, 1, FUN=sum)
```

```
## [1]  7  9 11 13 15
```

If you want to calculate the same for columns replace `1` by `2`. You can use any other function with a single input, or even a user defined function defined by `function()`. For example you can count values higher than 5 per column as:

```
apply(myarray, 2, function(x) length(x[x>5]))
```

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

There are specialized packages for data analysis such as "dplyr". It uses a special "pipe" operator (`%>%`). The output of the operation before the pipe is used as an input of the operation after the pipe. Special functions `mutate`, `select`, `filter`, `summarise`, `arrange` and others are used in dplyr. You can replace the `aggregate` function from the previous example as:

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
volby %>% group_by(party) %>% summarise(abs=sum(abs))
```

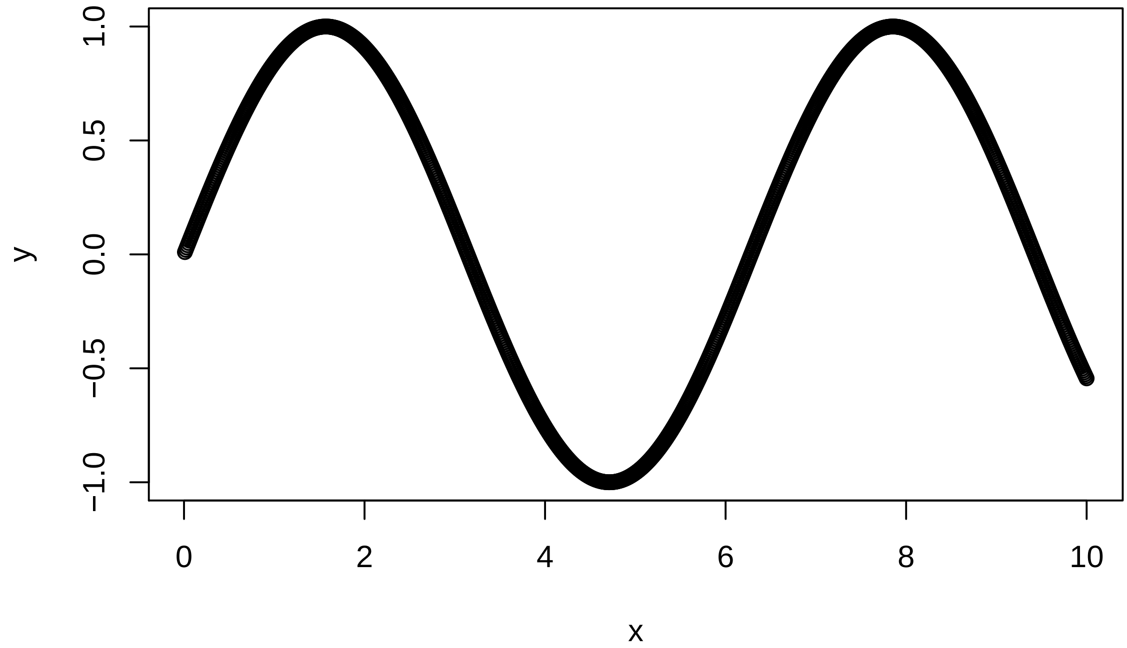
```
## # A tibble: 20 x 2
##    party      abs
##    <fct>    <int>
##  1 ANO 2011 47723
##  2 CSSD     55212
##  3 DSSS      1039
##  4 HLVZHURU  1782
##  5 KC         638
##  6 KDU-CSL  36769
##  7 KSCM     32534
##  8 LEV 21     224
##  9 OBC_2011   260
## 10 ODS      91977
## 11 PB         480
## 12 Pirati   12995
## 13 SPOZ      3473
## 14 SsCR       457
## 15 Suveren.   400
## 16 Svobodni 11511
## 17 SZ       39345
## 18 TOP 09   95033
## 19 Usvit     8170
## 20 Zmena     3035
```

Another useful package for data analysis is "TidyR". Both dplyr and TidyR are from a tidyverse package of packages for data analysis. TidyR uses functions `gather()`, `spread()`, `separate()`, `extract()` and others to reshape data from untidy to tidy datasets.

0.0.3 Plotting in R

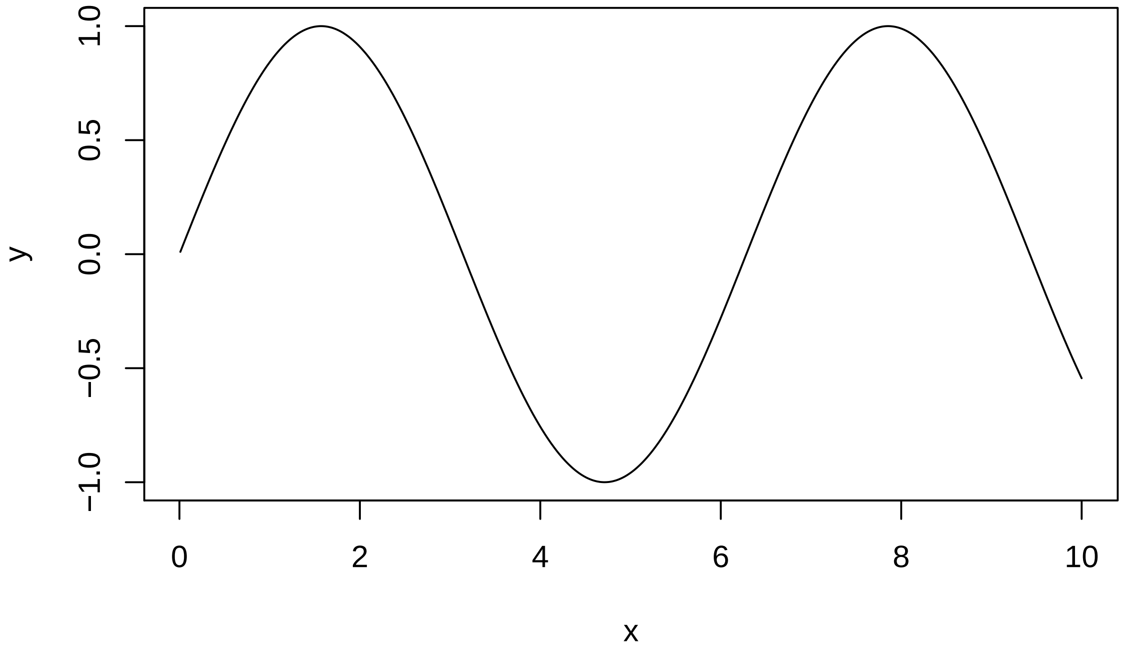
The basic plotting function in R is `plot`:

```
x<-1:1000/100
y<-sin(x)
plot(x,y)
```



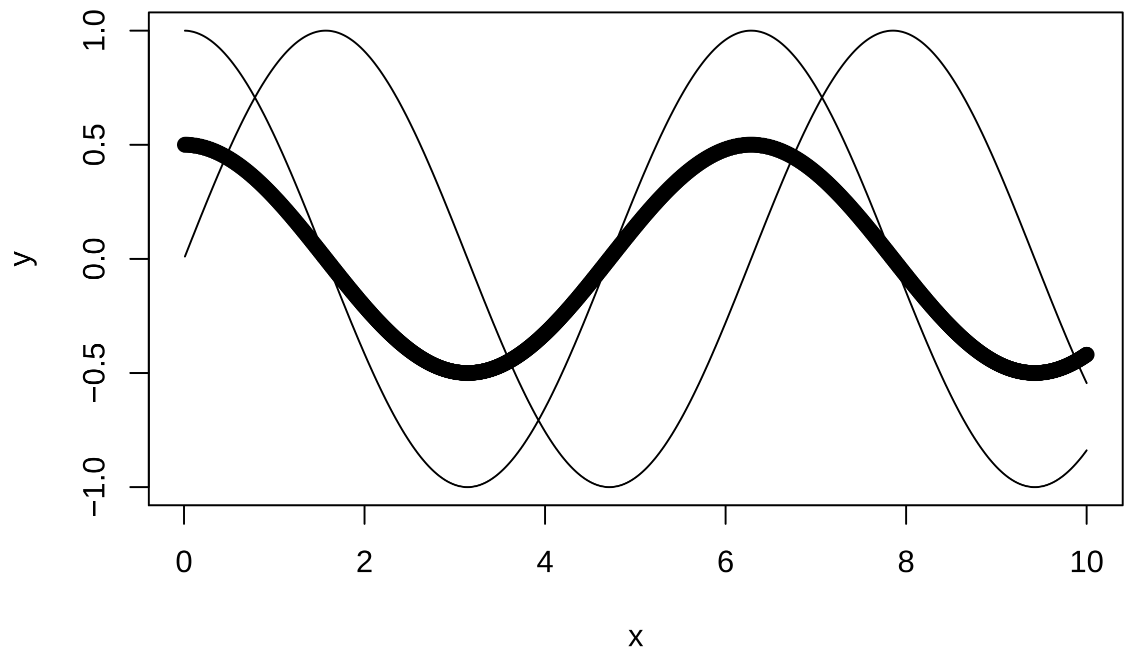
You can switch between points (default), lines (`type="l"`), both, histogram-like, steps, none and others:

```
plot( x, y, type="l")
```



The function `plot` always plots a new plot. If you want to add new lines or points into the existing plot use `lines` and `points`. Simply open new plot by function `plot` and then use `lines` and `points` without closing the plot:

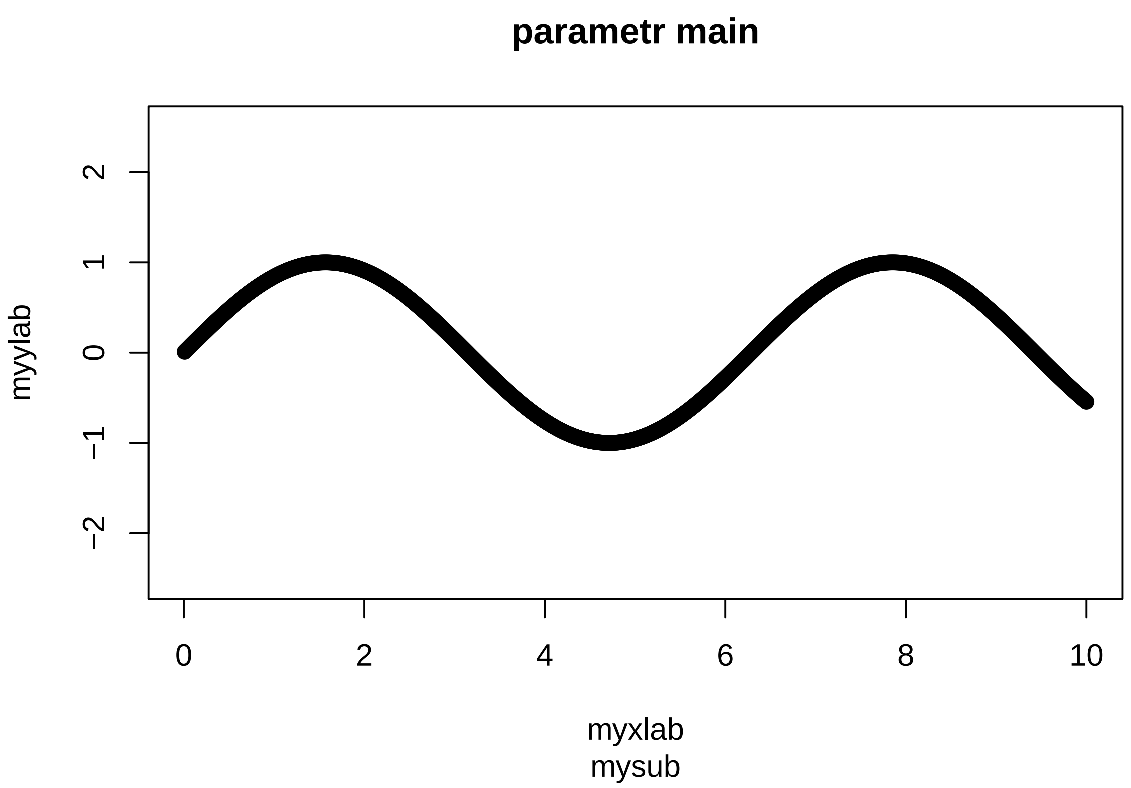
```
plot(x,y, type="l")
lines(x,cos(x))
points(x,0.5*cos(x))
```



Sometimes it is useful to create an empty plot canvas by `plot` with `type="n"` and add lines and points.

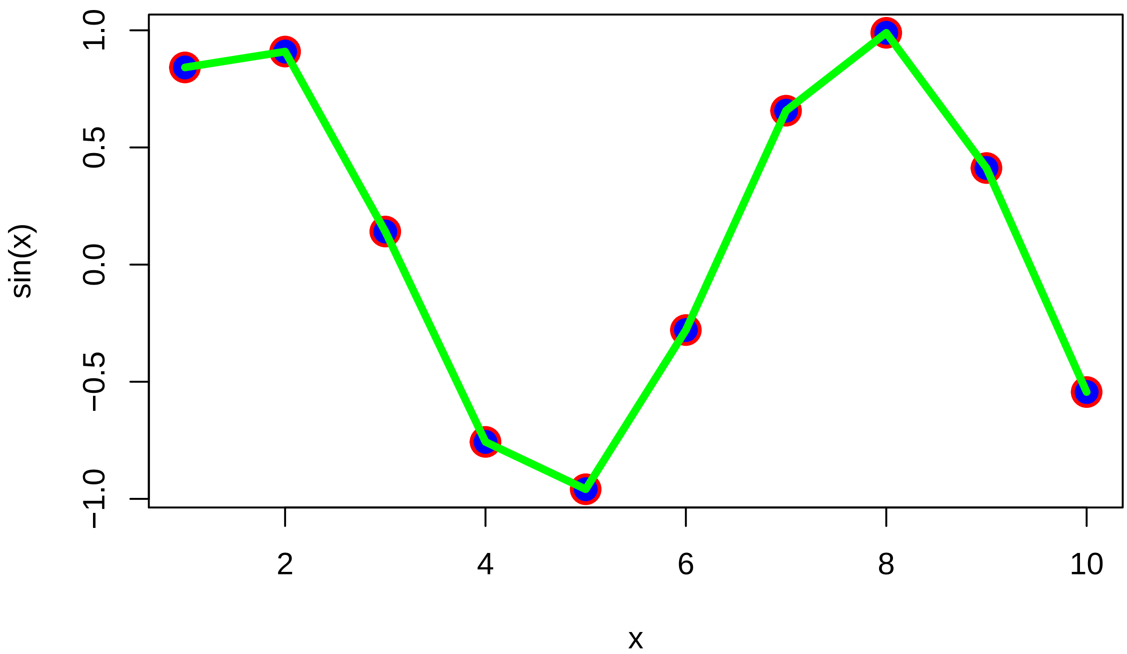
The function `plot` has many additional parameters:

```
plot( x, y, main="parametr main", sub="mysub",
     xlab="myxlab", ylab="myylab", asp=1)
```



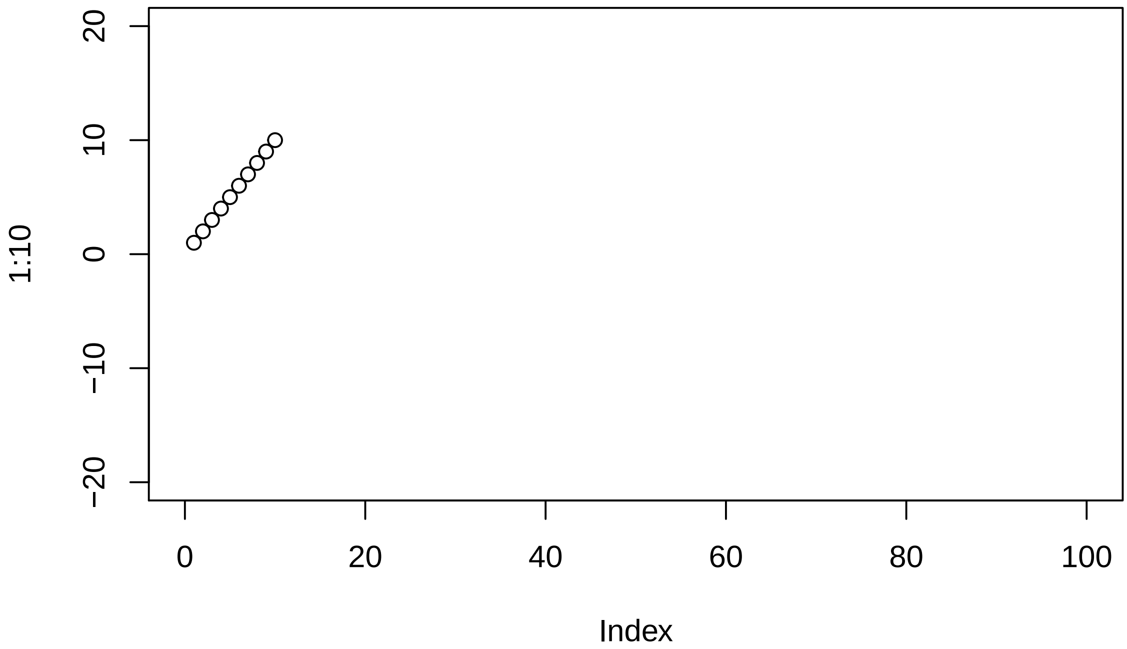
You can change colors of point and lines, shapes of points etc.:

```
x<-1:10
plot(x, sin(x), pch=21, col="red", bg="blue", cex=2, lwd=2)
lines(x, sin(x), col="green", lwd=4)
```



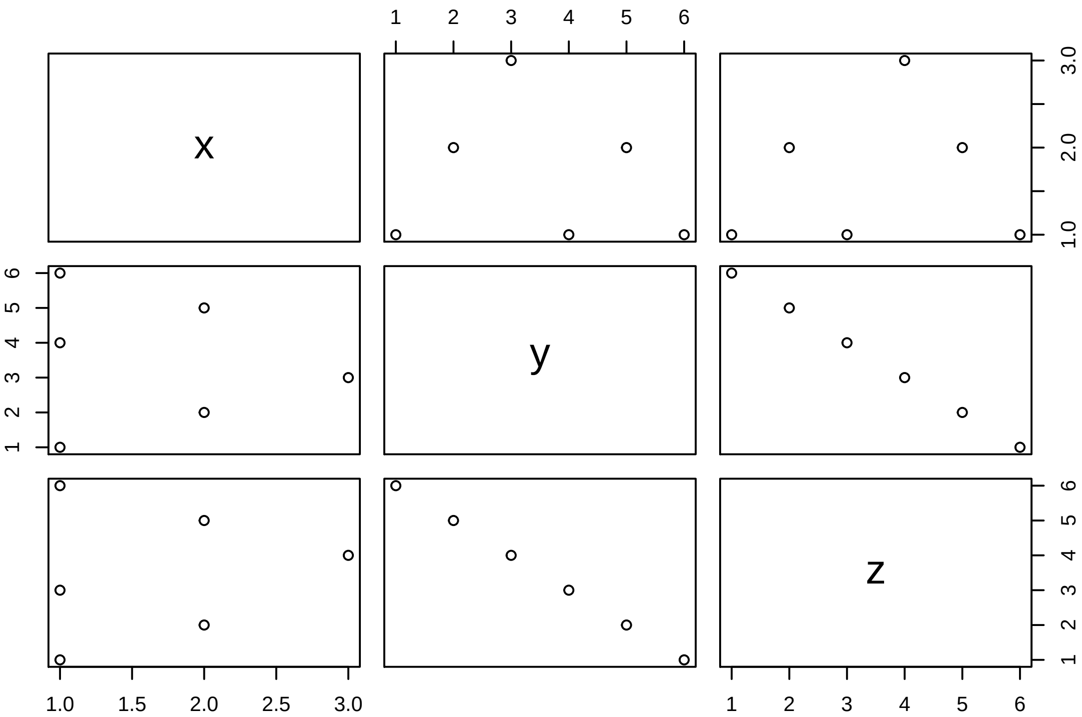
Range of the horizontal and vertical axis can be controlled by `xlim` and `ylim`:

```
plot(1:10, xlim=c(0,100), ylim=c(-20,20))
```



The function plot can be applied not only to a pair of vectors, but also to a single vector, `data.frame` and other objects. Lets try on `data.frame`:

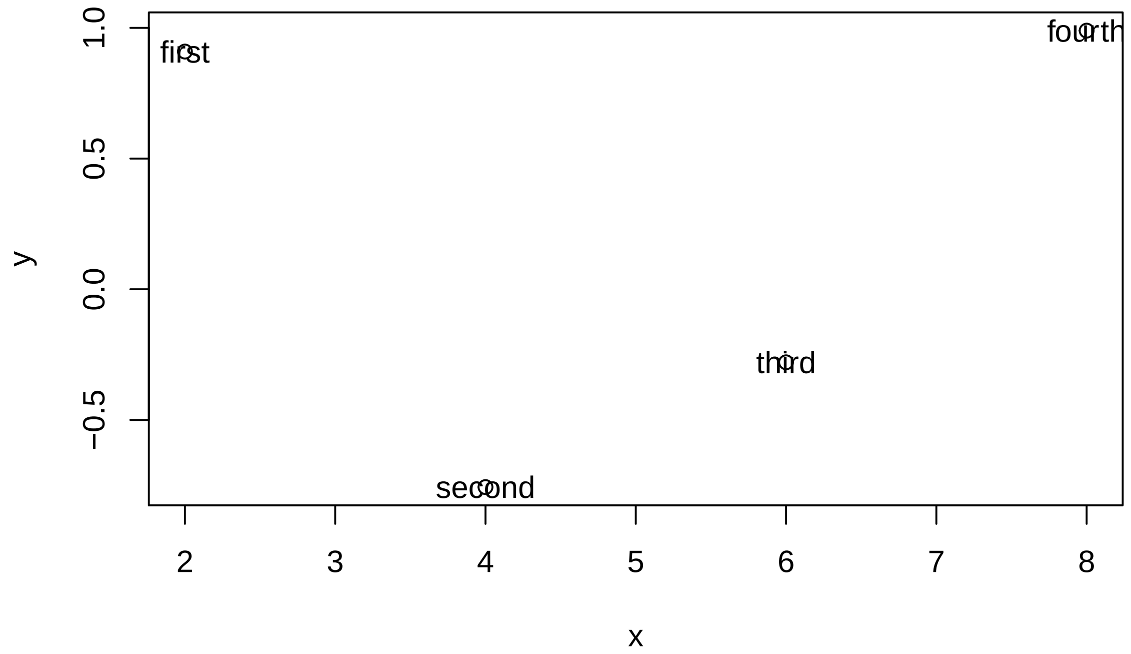
```
x<-c(1,2,3,1,2,1)
y<-1:6
z<-6:1
xyz<-data.frame(x,y,z)
plot(xyz)
```



The function `plot` can be used to other objects as will be shown later.

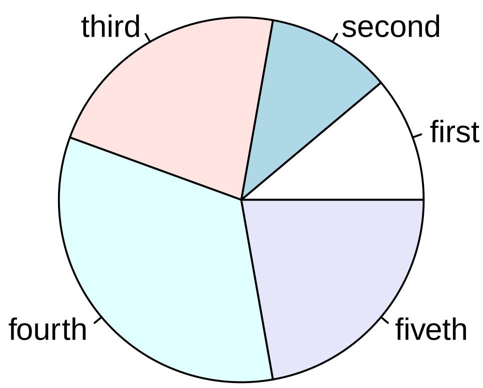
Using function `text` it is possible to add short strings to points with coordinates x and y:

```
x<-1:4*2
y<-sin(x)
pointnames<-c("first", "second", "third", "fourth")
plot(x,y)
text(x, y, labels=pointnames)
```



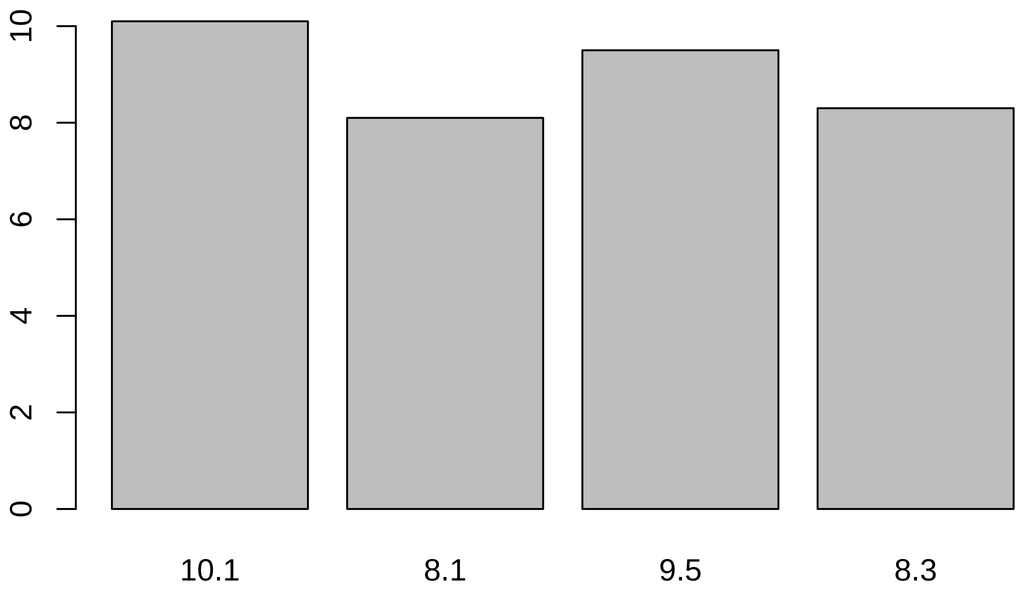
R can make pie charts:

```
x<-c(1,1,2,3,2)
nam<-c("first", "second", "third", "fourth", "fiveth")
pie(x, labels=nam)
```



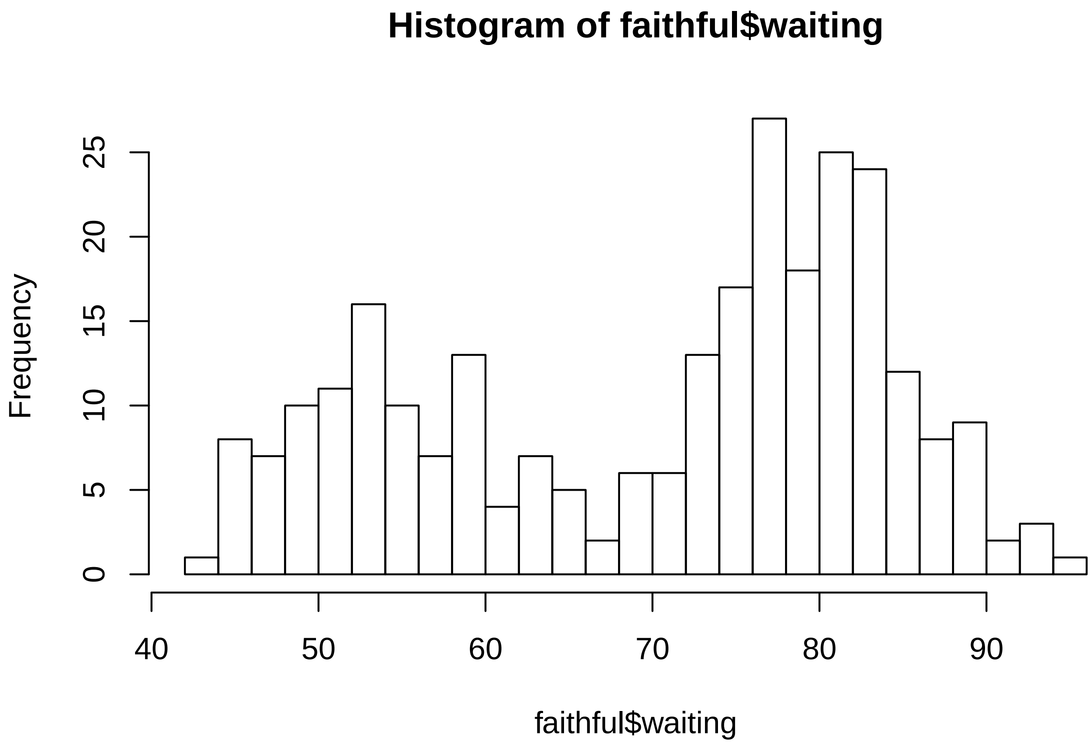
Barplots can be drawn using function barplot:

```
barplot(c(10.1,8.1,9.5,8.3), names.arg=c(10.1,8.1,9.5,8.3))
```



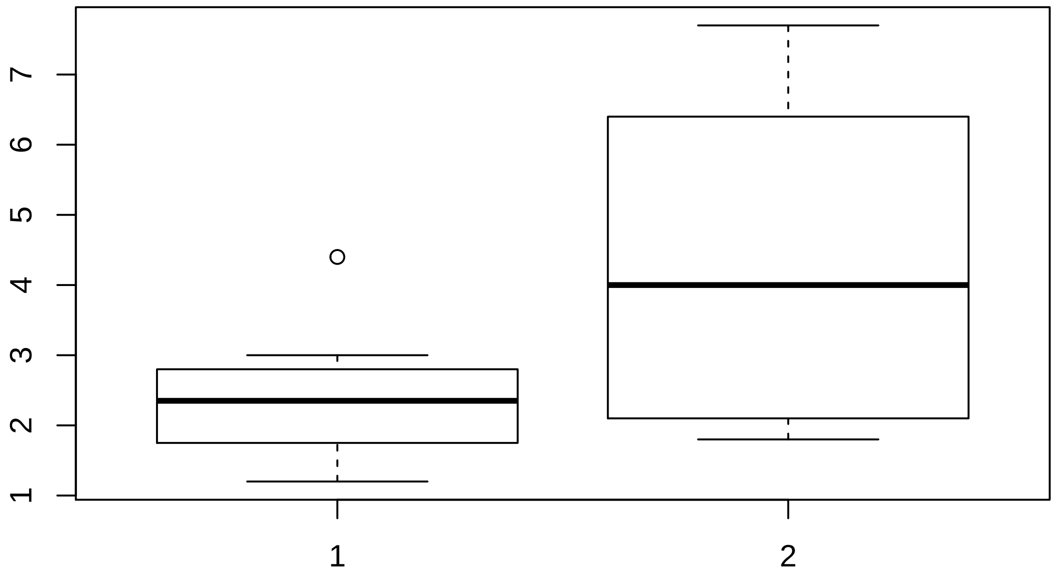
Histograms can be plotted by `hist` with breaks controllable by `breaks` parameter:

```
hist(faithful$waiting, breaks=20)
```



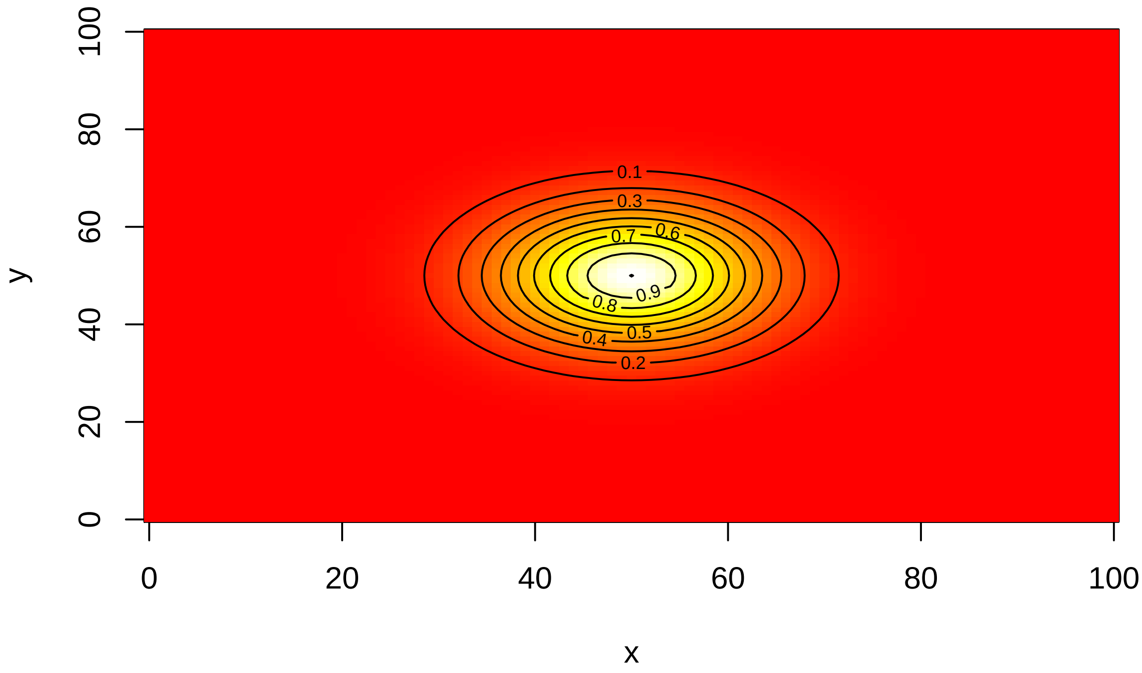
Tukey's boxplots can be ploted by `boxplot` function:

```
x<-c(1.2,2.2,1.3,4.4,3.0,2.2,2.5,2.6)
y<-c(3.3,2.3,1.8,5.5,7.7,7.3,1.9,4.7)
boxplot(x, y)
```

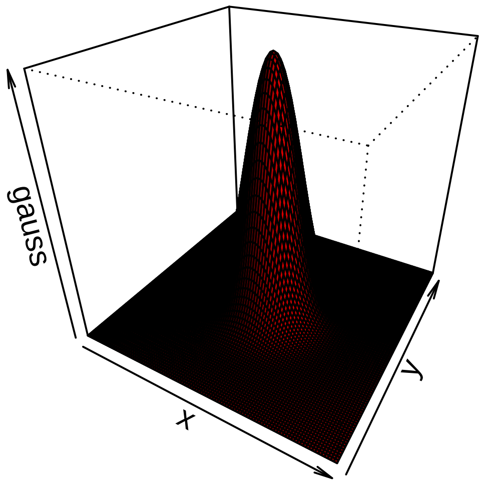


Three-dimensional plots can be presented using image, contour or persp function:

```
x<-0:100
y<-0:100
gauss<-exp(-outer((x-50)**2/200,(y-50)**2/200,"+"))
image(x, y, gauss, col=heat.colors(100))
contour(x, y, gauss, levels=0:10/10, add=TRUE)
```

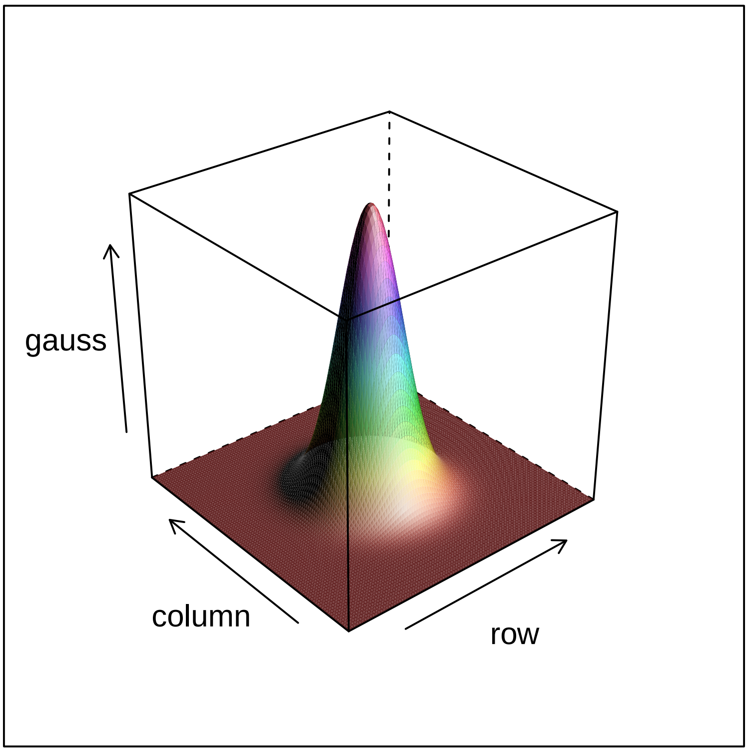



```
persp(x, y, gauss, col="red", theta=30, phi=30, shade=0.75, ltheta=100)
```



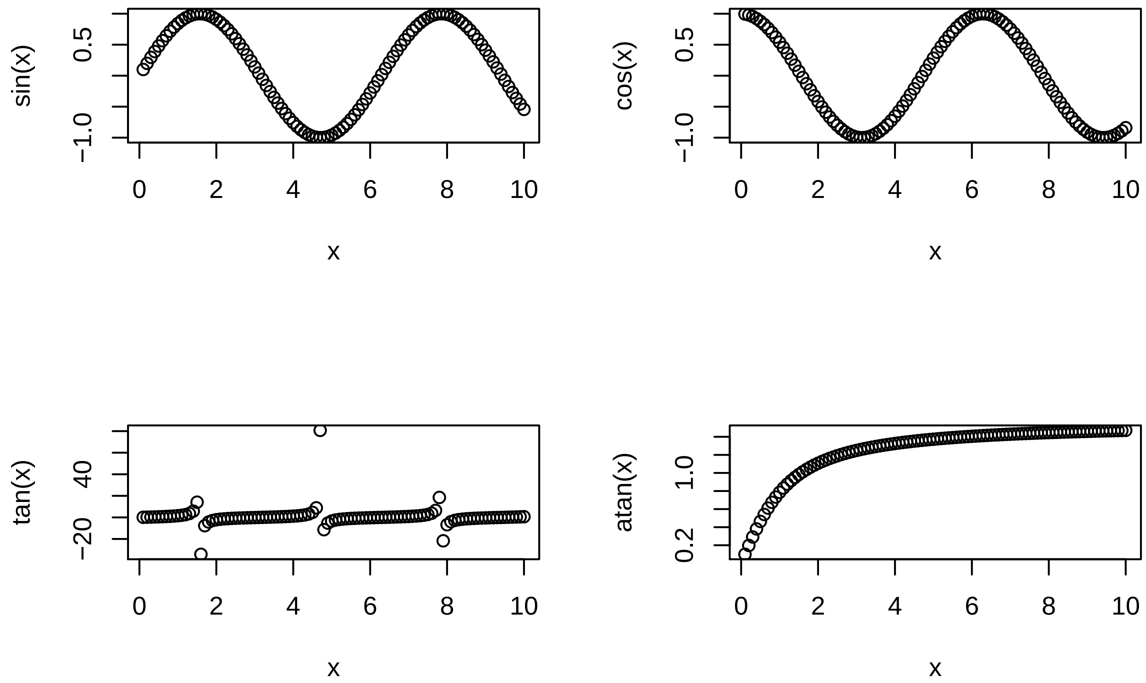
Nice 3D plots can be made by the `wireframe` function from the `lattice` library:

```
library(lattice)
wireframe(gauss, shade=TRUE,light.source = c(10,0,10))
```



The shape of plots can be modified by `par` function invoked before the function `plot` or other plotting functions. As an example we can show plotting of four plots on one canvas:

```
par(mfrow=c(2,2))
x<-1:100/10
plot(x, sin(x))
plot(x, cos(x))
plot(x, tan(x))
plot(x, atan(x))
```



R can use a wide range of colors. Pre-defined colors can be shown by functions `colors`:

```
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"
##  [25] "blanchedalmond"       "blue"                 "blue1"
##  [28] "blue2"                "blue3"                "blue4"
##  [31] "blueviolet"           "brown"                "brown1"
##  [34] "brown2"               "brown3"               "brown4"
##  [37] "burlywood"            "burlywood1"           "burlywood2"
##  [40] "burlywood3"           "burlywood4"           "cadetblue"
##  [43] "cadetblue1"           "cadetblue2"           "cadetblue3"
##  [46] "cadetblue4"           "chartreuse"           "chartreuse1"
##  [49] "chartreuse2"          "chartreuse3"          "chartreuse4"
##  [52] "chocolate"            "chocolate1"           "chocolate2"
##  [55] "chocolate3"           "chocolate4"           "coral"
##  [58] "coral1"               "coral2"               "coral3"
```

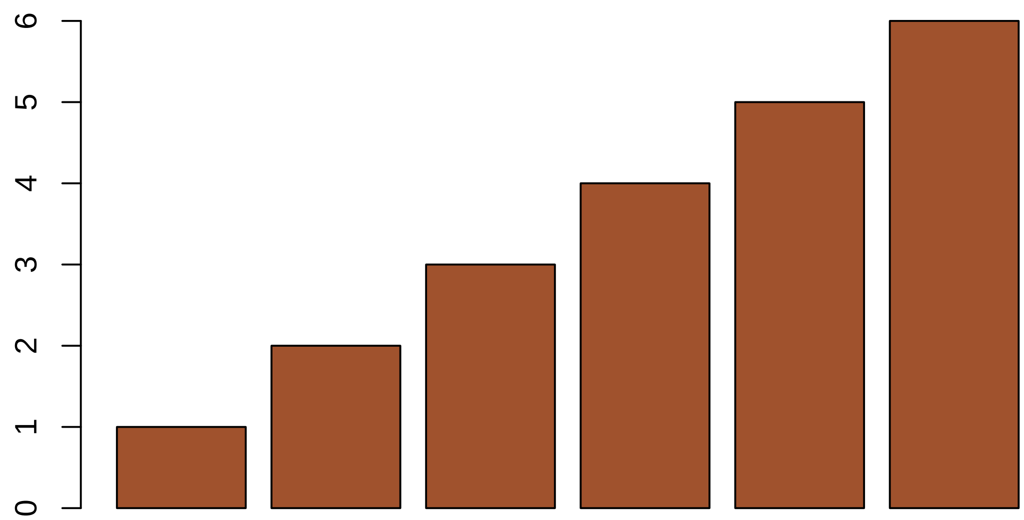
```
##  [61] "coral4"               "cornflowerblue"       "cornsilk"
##  [64] "cornsilk1"            "cornsilk2"            "cornsilk3"
##  [67] "cornsilk4"            "cyan"                 "cyan1"
##  [70] "cyan2"                "cyan3"                "cyan4"
##  [73] "darkblue"             "darkcyan"             "darkgoldenrod"
##  [76] "darkgoldenrod1"       "darkgoldenrod2"       "darkgoldenrod3"
##  [79] "darkgoldenrod4"       "darkgray"             "darkgreen"
##  [82] "darkgrey"             "darkkhaki"            "darkmagenta"
##  [85] "darkolivegreen"       "darkolivegreen1"      "darkolivegreen2"
##  [88] "darkolivegreen3"      "darkolivegreen4"      "darkorange"
##  [91] "darkorange1"          "darkorange2"          "darkorange3"
##  [94] "darkorange4"          "darkorchid"           "darkorchid1"
##  [97] "darkorchid2"          "darkorchid3"          "darkorchid4"
## [100] "darkred"              "darksalmon"           "darkseagreen"
## [103] "darkseagreen1"        "darkseagreen2"        "darkseagreen3"
## [106] "darkseagreen4"        "darkslateblue"        "darkslategray"
## [109] "darkslategray1"       "darkslategray2"       "darkslategray3"
## [112] "darkslategray4"       "darkslategrey"        "darkturquoise"
## [115] "darkviolet"           "deeppink"             "deeppink1"
## [118] "deeppink2"            "deeppink3"            "deeppink4"
## [121] "deepskyblue"          "deepskyblue1"         "deepskyblue2"
## [124] "deepskyblue3"         "deepskyblue4"         "dimgray"
## [127] "dimgrey"              "dodgerblue"           "dodgerblue1"
## [130] "dodgerblue2"          "dodgerblue3"          "dodgerblue4"
## [133] "firebrick"            "firebrick1"           "firebrick2"
## [136] "firebrick3"           "firebrick4"           "floralwhite"
## [139] "forestgreen"          "gainsboro"            "ghostwhite"
## [142] "gold"                 "gold1"                "gold2"
## [145] "gold3"                "gold4"                "goldenrod"
## [148] "goldenrod1"           "goldenrod2"           "goldenrod3"
## [151] "goldenrod4"           "gray"                 "gray0"
## [154] "gray1"                "gray2"                "gray3"
## [157] "gray4"                "gray5"                "gray6"
## [160] "gray7"                "gray8"                "gray9"
## [163] "gray10"               "gray11"               "gray12"
## [166] "gray13"               "gray14"               "gray15"
## [169] "gray16"               "gray17"               "gray18"
## [172] "gray19"               "gray20"               "gray21"
## [175] "gray22"               "gray23"               "gray24"
## [178] "gray25"               "gray26"               "gray27"
## [181] "gray28"               "gray29"               "gray30"
## [184] "gray31"               "gray32"               "gray33"
## [187] "gray34"               "gray35"               "gray36"
## [190] "gray37"               "gray38"               "gray39"
## [193] "gray40"               "gray41"               "gray42"
## [196] "gray43"               "gray44"               "gray45"
## [199] "gray46"               "gray47"               "gray48"
## [202] "gray49"               "gray50"               "gray51"
## [205] "gray52"               "gray53"               "gray54"
## [208] "gray55"               "gray56"               "gray57"
## [211] "gray58"               "gray59"               "gray60"
## [214] "gray61"               "gray62"               "gray63"
## [217] "gray64"               "gray65"               "gray66"
## [220] "gray67"               "gray68"               "gray69"
## [223] "gray70"               "gray71"               "gray72"
## [226] "gray73"               "gray74"               "gray75"
## [229] "gray76"               "gray77"               "gray78"
## [232] "gray79"               "gray80"               "gray81"
```

```
## [235] "gray82"            "gray83"            "gray84"
## [238] "gray85"            "gray86"            "gray87"
## [241] "gray88"            "gray89"            "gray90"
## [244] "gray91"            "gray92"            "gray93"
## [247] "gray94"            "gray95"            "gray96"
## [250] "gray97"            "gray98"            "gray99"
## [253] "gray100"           "green"             "green1"
## [256] "green2"            "green3"            "green4"
## [259] "greenyellow"       "grey"              "grey0"
## [262] "grey1"             "grey2"             "grey3"
## [265] "grey4"             "grey5"             "grey6"
## [268] "grey7"             "grey8"             "grey9"
## [271] "grey10"            "grey11"            "grey12"
## [274] "grey13"            "grey14"            "grey15"
## [277] "grey16"            "grey17"            "grey18"
## [280] "grey19"            "grey20"            "grey21"
## [283] "grey22"            "grey23"            "grey24"
## [286] "grey25"            "grey26"            "grey27"
## [289] "grey28"            "grey29"            "grey30"
## [292] "grey31"            "grey32"            "grey33"
## [295] "grey34"            "grey35"            "grey36"
## [298] "grey37"            "grey38"            "grey39"
## [301] "grey40"            "grey41"            "grey42"
## [304] "grey43"            "grey44"            "grey45"
## [307] "grey46"            "grey47"            "grey48"
## [310] "grey49"            "grey50"            "grey51"
## [313] "grey52"            "grey53"            "grey54"
## [316] "grey55"            "grey56"            "grey57"
## [319] "grey58"            "grey59"            "grey60"
## [322] "grey61"            "grey62"            "grey63"
## [325] "grey64"            "grey65"            "grey66"
## [328] "grey67"            "grey68"            "grey69"
## [331] "grey70"            "grey71"            "grey72"
## [334] "grey73"            "grey74"            "grey75"
## [337] "grey76"            "grey77"            "grey78"
## [340] "grey79"            "grey80"            "grey81"
## [343] "grey82"            "grey83"            "grey84"
## [346] "grey85"            "grey86"            "grey87"
## [349] "grey88"            "grey89"            "grey90"
## [352] "grey91"            "grey92"            "grey93"
## [355] "grey94"            "grey95"            "grey96"
## [358] "grey97"            "grey98"            "grey99"
## [361] "grey100"           "honeydew"          "honeydew1"
## [364] "honeydew2"         "honeydew3"         "honeydew4"
## [367] "hotpink"           "hotpink1"          "hotpink2"
## [370] "hotpink3"          "hotpink4"          "indianred"
## [373] "indianred1"        "indianred2"        "indianred3"
## [376] "indianred4"        "ivory"             "ivory1"
## [379] "ivory2"            "ivory3"            "ivory4"
## [382] "khaki"             "khaki1"            "khaki2"
## [385] "khaki3"            "khaki4"            "lavender"
## [388] "lavenderblush"     "lavenderblush1"    "lavenderblush2"
## [391] "lavenderblush3"    "lavenderblush4"    "lawngreen"
## [394] "lemonchiffon"      "lemonchiffon1"     "lemonchiffon2"
## [397] "lemonchiffon3"     "lemonchiffon4"     "lightblue"
## [400] "lightblue1"        "lightblue2"        "lightblue3"
## [403] "lightblue4"        "lightcoral"        "lightcyan"
## [406] "lightcyan1"        "lightcyan2"        "lightcyan3"
```

```
## [409] "lightcyan4"           "lightgoldenrod"       "lightgoldenrod1"
## [412] "lightgoldenrod2"      "lightgoldenrod3"      "lightgoldenrod4"
## [415] "lightgoldenrodyellow" "lightgray"            "lightgreen"
## [418] "lightgrey"            "lightpink"            "lightpink1"
## [421] "lightpink2"           "lightpink3"           "lightpink4"
## [424] "lightsalmon"          "lightsalmon1"         "lightsalmon2"
## [427] "lightsalmon3"         "lightsalmon4"         "lightseagreen"
## [430] "lightskyblue"         "lightskyblue1"        "lightskyblue2"
## [433] "lightskyblue3"        "lightskyblue4"        "lightslateblue"
## [436] "lightslategray"       "lightslategrey"       "lightsteelblue"
## [439] "lightsteelblue1"      "lightsteelblue2"      "lightsteelblue3"
## [442] "lightsteelblue4"      "lightyellow"          "lightyellow1"
## [445] "lightyellow2"         "lightyellow3"         "lightyellow4"
## [448] "limegreen"            "linen"                "magenta"
## [451] "magenta1"             "magenta2"             "magenta3"
## [454] "magenta4"             "maroon"               "maroon1"
## [457] "maroon2"              "maroon3"              "maroon4"
## [460] "mediumaquamarine"     "mediumblue"           "mediumorchid"
## [463] "mediumorchid1"        "mediumorchid2"        "mediumorchid3"
## [466] "mediumorchid4"        "mediumpurple"         "mediumpurple1"
## [469] "mediumpurple2"        "mediumpurple3"        "mediumpurple4"
## [472] "mediumseagreen"       "mediumslateblue"      "mediumspringgreen"
## [475] "mediumturquoise"      "mediumvioletred"      "midnightblue"
## [478] "mintcream"            "mistyrose"            "mistyrose1"
## [481] "mistyrose2"           "mistyrose3"           "mistyrose4"
## [484] "moccasin"             "navajowhite"          "navajowhite1"
## [487] "navajowhite2"         "navajowhite3"         "navajowhite4"
## [490] "navy"                 "navyblue"             "oldlace"
## [493] "olivedrab"            "olivedrab1"           "olivedrab2"
## [496] "olivedrab3"           "olivedrab4"           "orange"
## [499] "orange1"              "orange2"              "orange3"
## [502] "orange4"              "orangered"            "orangered1"
## [505] "orangered2"           "orangered3"           "orangered4"
## [508] "orchid"               "orchid1"              "orchid2"
## [511] "orchid3"              "orchid4"              "palegoldenrod"
## [514] "palegreen"            "palegreen1"           "palegreen2"
## [517] "palegreen3"           "palegreen4"           "paleturquoise"
## [520] "paleturquoise1"       "paleturquoise2"       "paleturquoise3"
## [523] "paleturquoise4"       "palevioletred"        "palevioletred1"
## [526] "palevioletred2"       "palevioletred3"       "palevioletred4"
## [529] "papayawhip"           "peachpuff"            "peachpuff1"
## [532] "peachpuff2"           "peachpuff3"           "peachpuff4"
## [535] "peru"                 "pink"                 "pink1"
## [538] "pink2"                "pink3"                "pink4"
## [541] "plum"                 "plum1"                "plum2"
## [544] "plum3"                "plum4"                "powderblue"
## [547] "purple"               "purple1"              "purple2"
## [550] "purple3"              "purple4"              "red"
## [553] "red1"                 "red2"                 "red3"
## [556] "red4"                 "rosybrown"            "rosybrown1"
## [559] "rosybrown2"           "rosybrown3"           "rosybrown4"
## [562] "royalblue"            "royalblue1"           "royalblue2"
## [565] "royalblue3"           "royalblue4"           "saddlebrown"
## [568] "salmon"               "salmon1"              "salmon2"
## [571] "salmon3"              "salmon4"              "sandybrown"
## [574] "seagreen"             "seagreen1"            "seagreen2"
## [577] "seagreen3"            "seagreen4"            "seashell"
## [580] "seashell1"            "seashell2"            "seashell3"
```

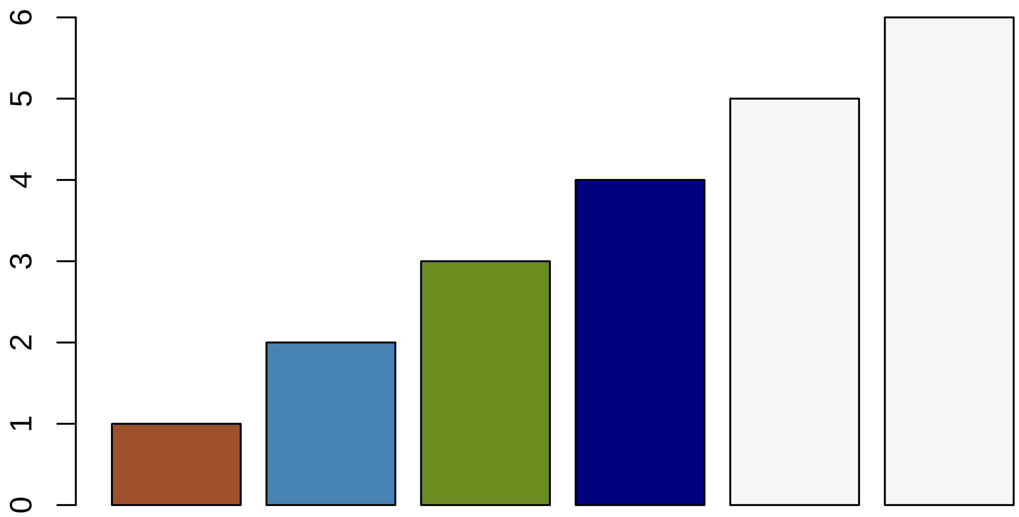
```
## [583] "seashell4"          "sienna"             "sienna1"
## [586] "sienna2"            "sienna3"            "sienna4"
## [589] "skyblue"            "skyblue1"           "skyblue2"
## [592] "skyblue3"           "skyblue4"           "slateblue"
## [595] "slateblue1"         "slateblue2"         "slateblue3"
## [598] "slateblue4"         "slategray"          "slategray1"
## [601] "slategray2"         "slategray3"         "slategray4"
## [604] "slategrey"          "snow"               "snow1"
## [607] "snow2"              "snow3"              "snow4"
## [610] "springgreen"        "springgreen1"       "springgreen2"
## [613] "springgreen3"       "springgreen4"       "steelblue"
## [616] "steelblue1"         "steelblue2"         "steelblue3"
## [619] "steelblue4"         "tan"                "tan1"
## [622] "tan2"               "tan3"               "tan4"
## [625] "thistle"            "thistle1"           "thistle2"
## [628] "thistle3"           "thistle4"           "tomato"
## [631] "tomato1"            "tomato2"            "tomato3"
## [634] "tomato4"            "turquoise"          "turquoise1"
## [637] "turquoise2"         "turquoise3"         "turquoise4"
## [640] "violet"             "violetred"          "violetred1"
## [643] "violetred2"         "violetred3"         "violetred4"
## [646] "wheat"              "wheat1"             "wheat2"
## [649] "wheat3"             "wheat4"             "whitesmoke"
## [652] "yellow"             "yellow1"            "yellow2"
## [655] "yellow3"            "yellow4"            "yellowgreen"
```

```r
barplot(1:6, col="sienna")
```

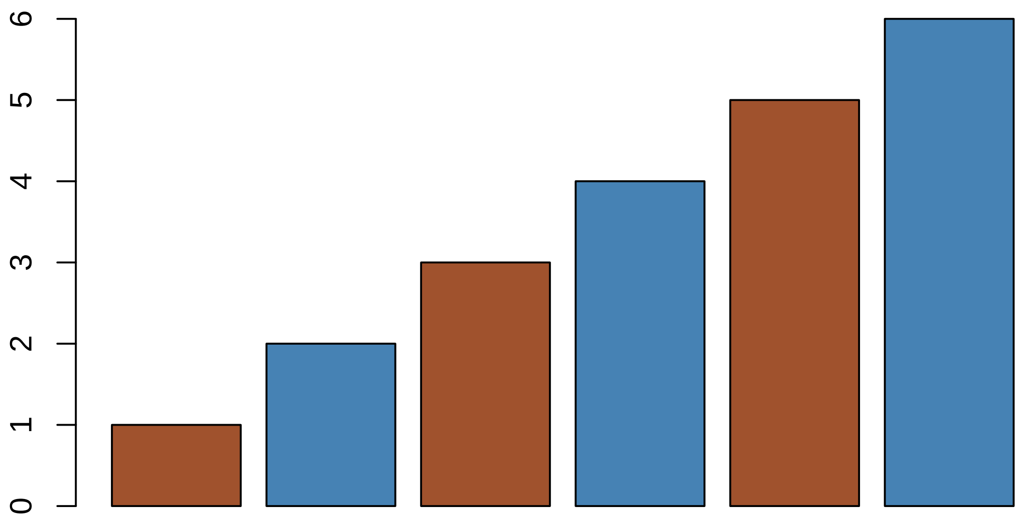


If colors are supplied as a vector, they alternate as shown bellow:

```r
barplot(1:6, col=c("sienna", "steelblue", "olivedrab",
                    "navy", "whitesmoke", "whitesmoke"))
```




```
barplot(1:6, col=c("sienna", "steelblue"))
```

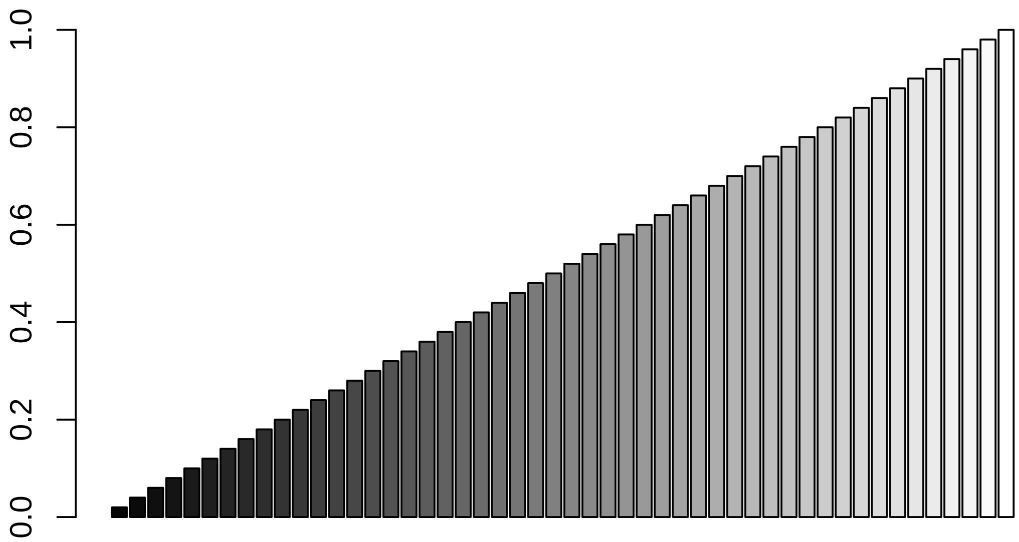


Shades of gray can be used by the function `gray`:

```
x<-1:50/50
gray(x)
```

```
##  [1] "#050505" "#0A0A0A" "#0F0F0F" "#141414" "#1A1A1A" "#1F1F1F" "#242424"
##  [8] "#292929" "#2E2E2E" "#333333" "#383838" "#3D3D3D" "#424242" "#474747"
## [15] "#4D4D4D" "#525252" "#575757" "#5C5C5C" "#616161" "#666666" "#6B6B6B"
## [22] "#707070" "#757575" "#7A7A7A" "#808080" "#858585" "#8A8A8A" "#8F8F8F"
## [29] "#949494" "#999999" "#9E9E9E" "#A3A3A3" "#A8A8A8" "#ADADAD" "#B3B3B3"
## [36] "#B8B8B8" "#BDBDBD" "#C2C2C2" "#C7C7C7" "#CCCCCC" "#D1D1D1" "#D6D6D6"
## [43] "#DBDBDB" "#E0E0E0" "#E6E6E6" "#EBEBEB" "#F0F0F0" "#F5F5F5" "#FAFAFA"
## [50] "#FFFFFF"
```

```
barplot(x, col=gray(x))
```

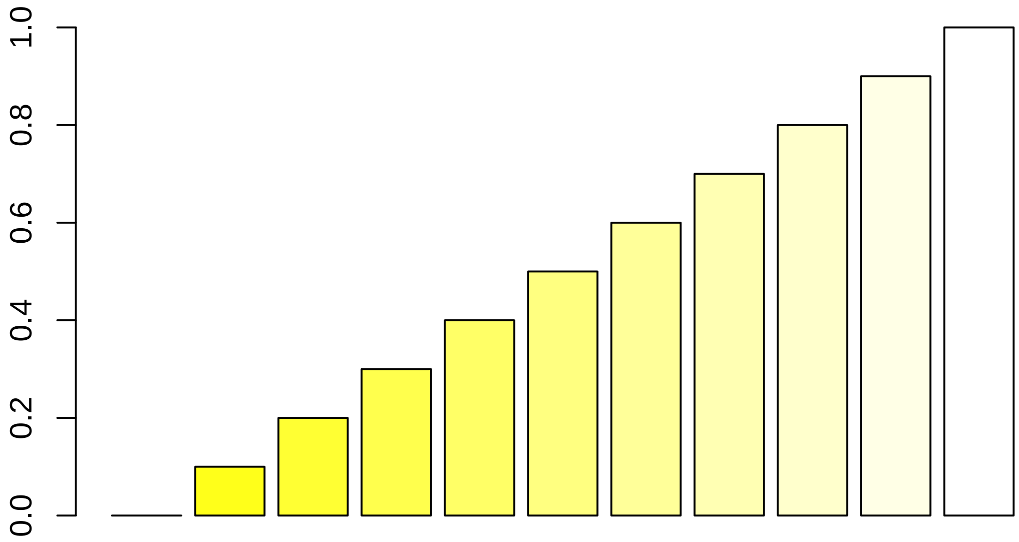


Colors can be also mixed from red, green and blue by the function `rgb`:

```
x<-0:10/10
rgb(1,1,x)
```

```
##  [1] "#FFFF00" "#FFFF1A" "#FFFF33" "#FFFF4D" "#FFFF66" "#FFFF80" "#FFFF99"
##  [8] "#FFFFB3" "#FFFFCC" "#FFFFE6" "#FFFFFF"
```

```
barplot(x, col=rgb(1,1,x))
```



You can try attractive palettes such as `rainbow`, `heat.colors`, `terrain.colors`, `topo.colors` and `cm.colors`.

Plots can be saved in many bitmap and vector graphical formats by functions `png`, `jpeg`, `pdf`, `svg` or `ps`. After invoking this function with file name as the argument no plot is shown. Instead it is saved to file. This property can be stopped by function `dev.off()`:

```
png("plot.png")
barplot(1:6)
dev.off()
```

```
## pdf
##   2
```

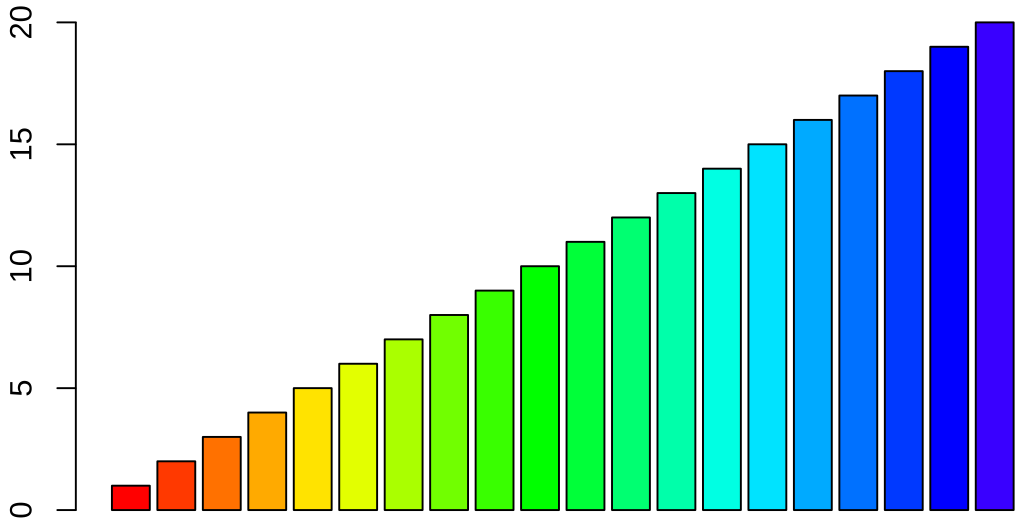
The plot is saved into working directory (see funcitons `getwd` and `setwd`).

R together with its packages makes it possible to plot graphs (in the sense of graph theory), heatmaps, word clouds, geographical maps and other special plot types.

0.0.3.1 Tips and tricks

- i often use function rainbow without the violet color:

```
barplot(1:20, col=rainbow(27)[1:20])
```



- high-resolution bitmap plots can be made in vector format and then converted to bitmap using your favorit graphical software
- alternatively, it is possible to use functions for bitmap plotting (e.g. `png`) with following modification:

```
x<-0:100
y<-0:100
png("plot.png", height=8, width=8, units='cm', res=600, pointsize=6)
gauss<-exp(-outer((x-50)**2/400,(y-50)**2/400,"+"))
image(x, y, gauss, col=heat.colors(100), axes=F)
contour(x, y, gauss, levels=0:10/10, add=TRUE, lwd=2, labcex=1.2)
axis(1, lwd=2)
axis(2, lwd=2)
box(lwd=2)
dev.off()
```

```
## pdf
##   2
```

This plots the plot in doubled size. In order to further increase the size it is possible to multiply `width`, `height` and `pointsize` in `png`. However, it keeps the same widths of lines and other parameters. To fix this, avoid plotting axes by function `image` (`axes=F`) and instead plot wide axes and box separately. It can be easily modified for other plotting functions.

- to make a movie, use the output file name with regular expression and a loop:

```
png("plot%03d.png")
x<-0:100
y<-0:100
for(i in 25:75) {
  gauss<-exp(-outer((x-i)**2/400,(y-i)**2/400,"+"))
  image(x, y, gauss, col=heat.colors(100))
  contour(x, y, gauss, levels=0:10/10, add=TRUE)
}
```

```r
dev.off()
```

```
## pdf
##   2
```

You can then use some video software (e.g. mencoder from Mplayer) to make a movie.

- a nice and popular plotting library from the "tidyverse" family is "ggplot2".

0.0.4 Random numbers in R

R can generate random numbers with different distributions. It is possible to generate ten random number with normal distribution with mean set to 20 and standard deviation set to 2 (default values are 0 and 1, respectively):

```r
x<-rnorm(10, mean=20, sd=2)
x
```

```
##  [1] 18.20982 20.95447 21.15001 18.33001 20.22682 22.18029 20.00108
##  [8] 22.02769 21.65802 16.95783
```

```r
mean(x)
```

```
## [1] 20.1696
```

```r
sd(x)
```

```
## [1] 1.790716
```

The true mean and standard deviation are not exactly equal to pre-set values, but you can try with larger sets:

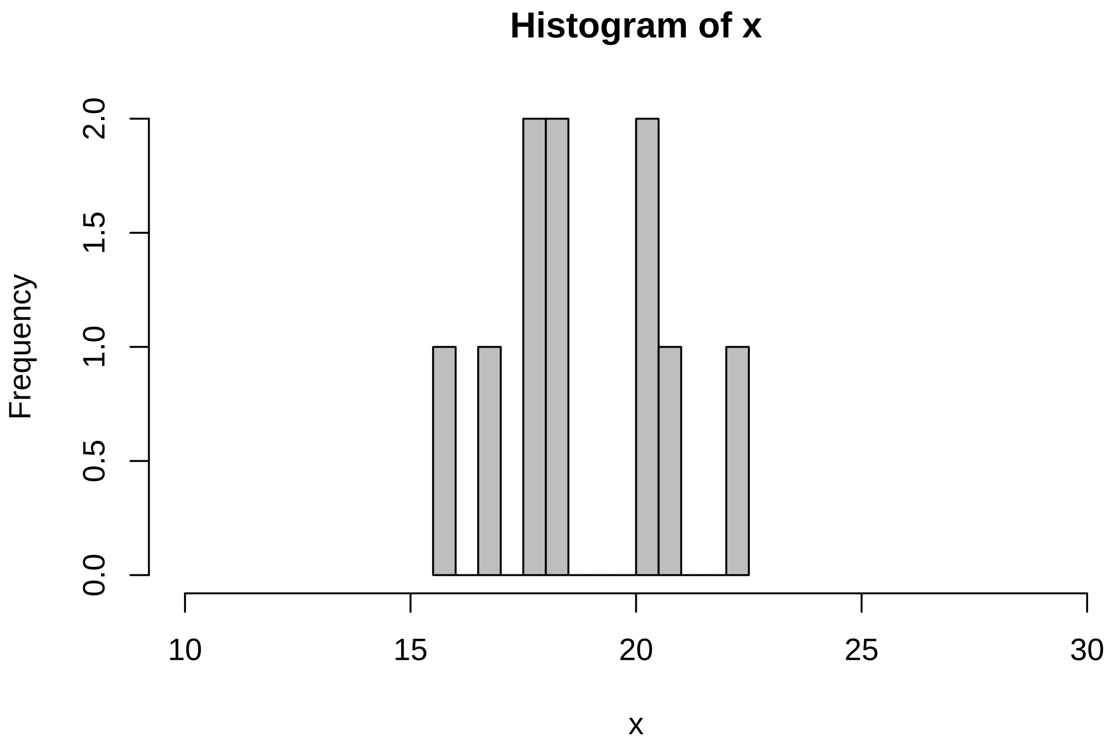
```r
x<-rnorm(10, mean=20, sd=2)
mean(x)
```

```
## [1] 18.79891
```

```r
sd(x)
```

```
## [1] 2.041641
```

```r
hist(x, br=20, xlim=c(10,30), col="gray")
```



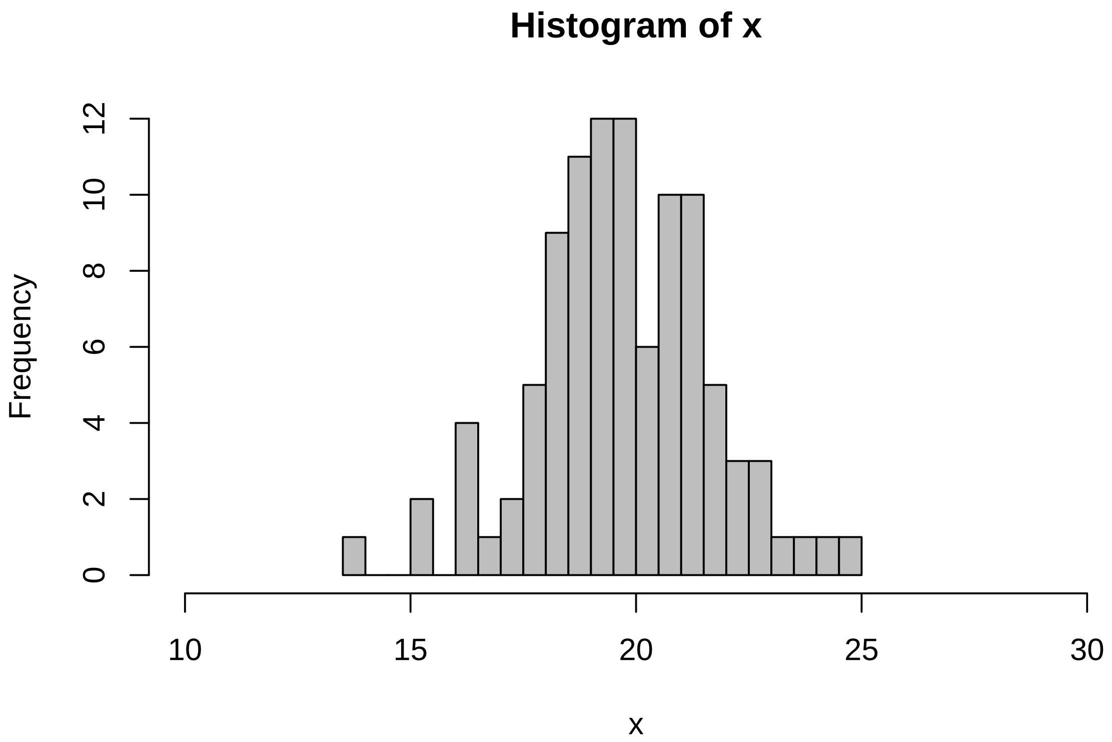

```r
x<-rnorm(100, mean=20, sd=2)
mean(x)
```

```
## [1] 19.692
```

```r
sd(x)
```

```
## [1] 1.948213
```

```r
hist(x, br=20, xlim=c(10,30), col="gray")
```



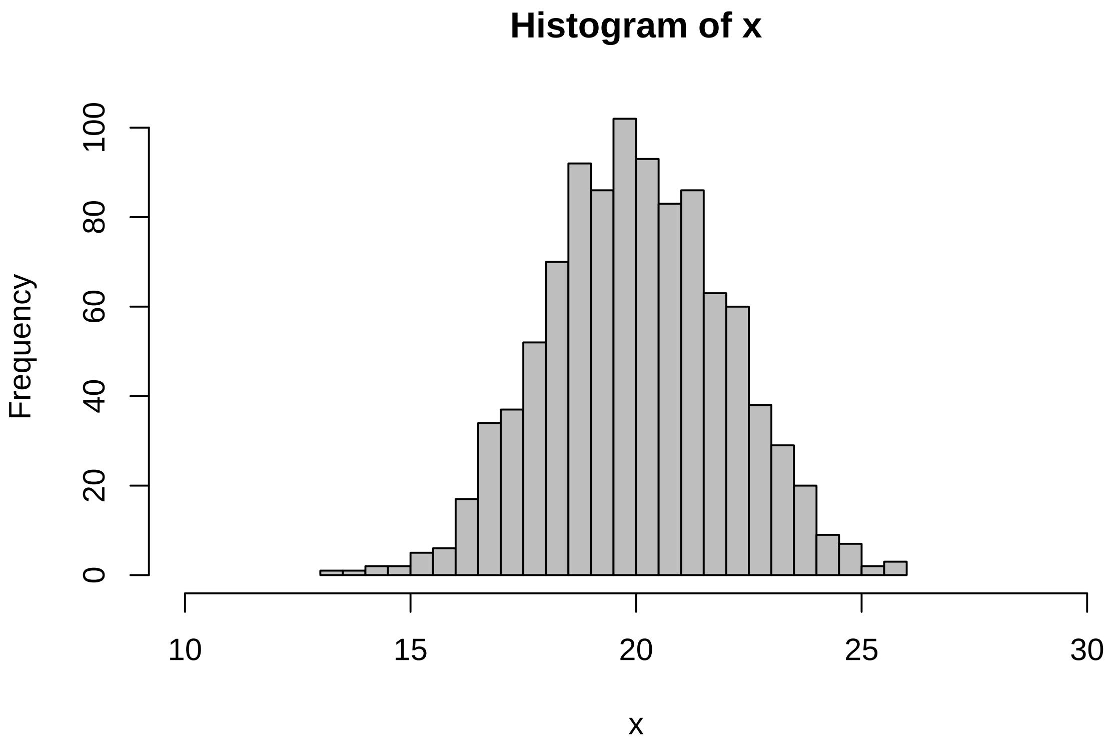

```r
x<-rnorm(1000, mean=20, sd=2)
mean(x)
```

```
## [1] 20.00122
```

```r
sd(x)
```

```
## [1] 1.999247
```

```r
hist(x, br=20, xlim=c(10,30), col="gray")
```



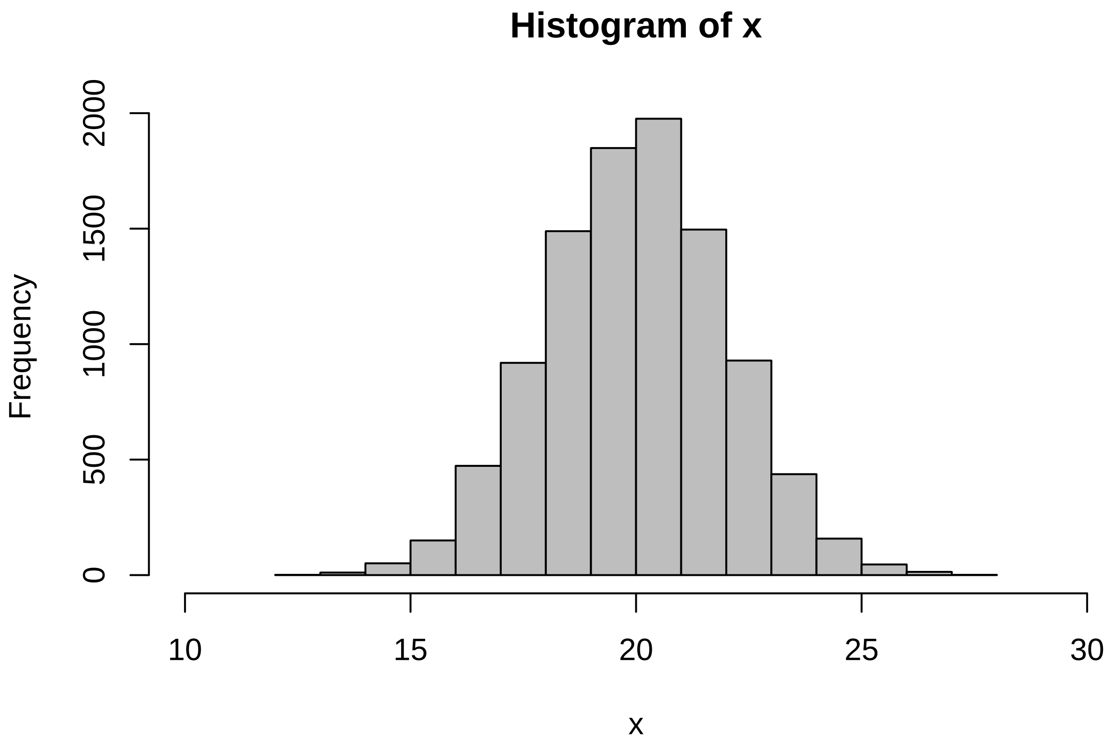

```
x<-rnorm(10000, mean=20, sd=2)
mean(x)
```

```
## [1] 20.00076
```

```
sd(x)
```

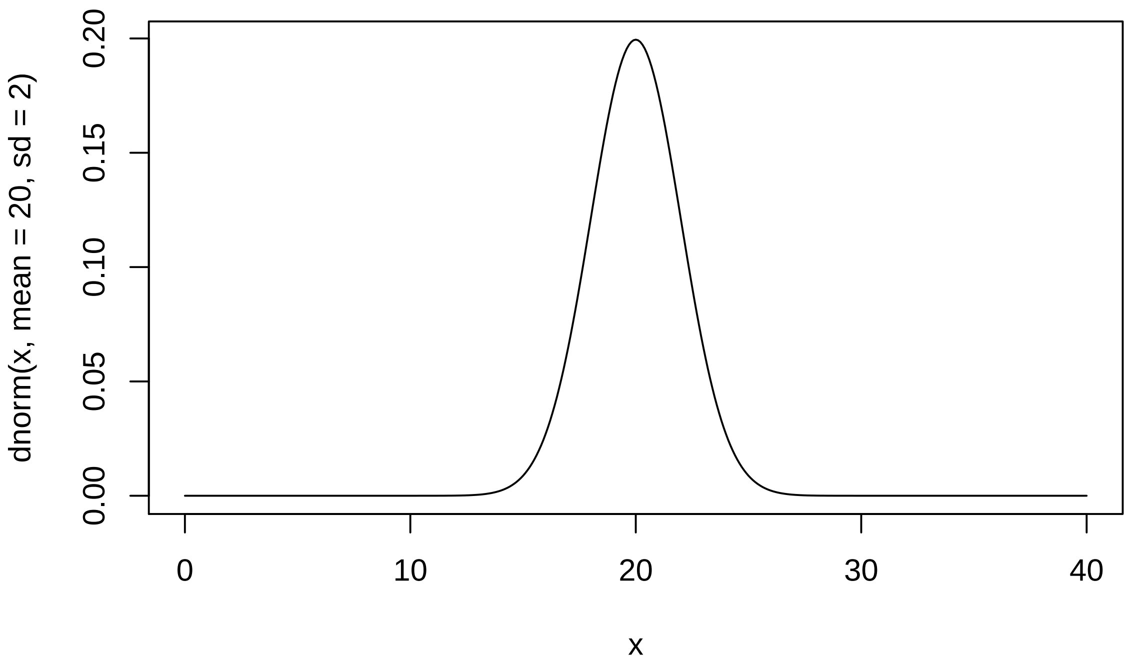
```
## [1] 1.997412
```

```
hist(x, br=20, xlim=c(10,30), col="gray")
```

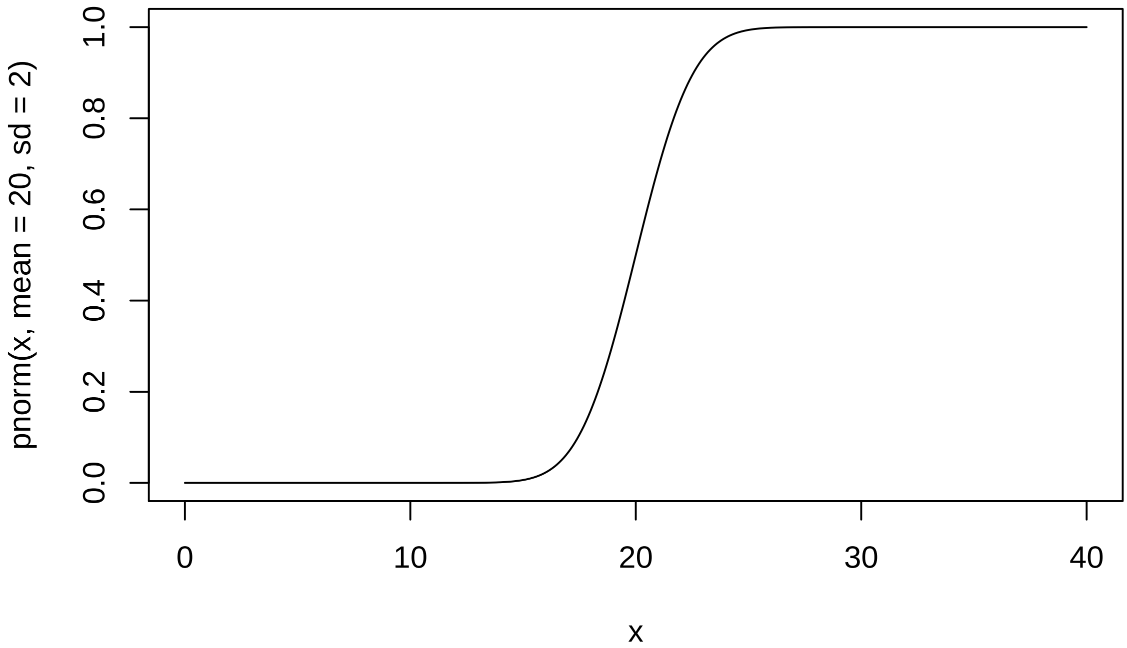


For normal distribution you can also calculate density by `dnorm`, distribution function by `pnorm` and quantile function by `qnorm`:

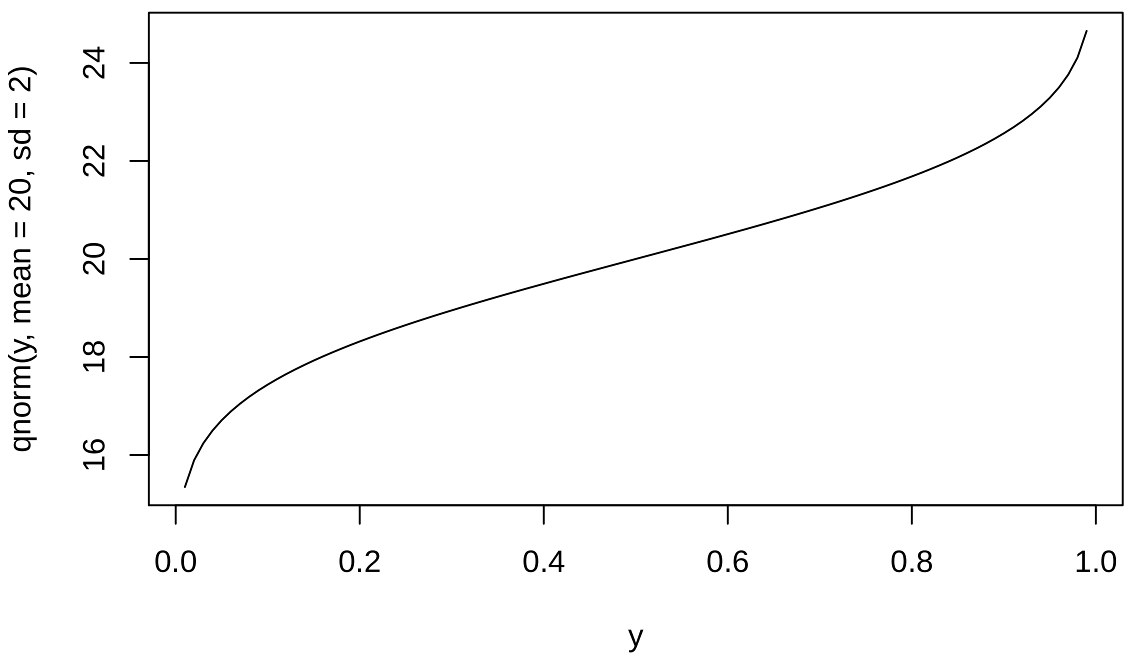
```
x<-0:400/10
plot(x, dnorm(x, mean=20, sd=2), type="l")
```



```r
plot(x, pnorm(x, mean=20, sd=2), type="l")
```




```r
y<-1:99/100
plot(y, qnorm(y, mean=20, sd=2), type="l")
```



The function pnorm is an integral of dnorm as you can see:

```
x<-0.1*0:230
sum(0.1*dnorm(x, mean=20, sd=2))
```

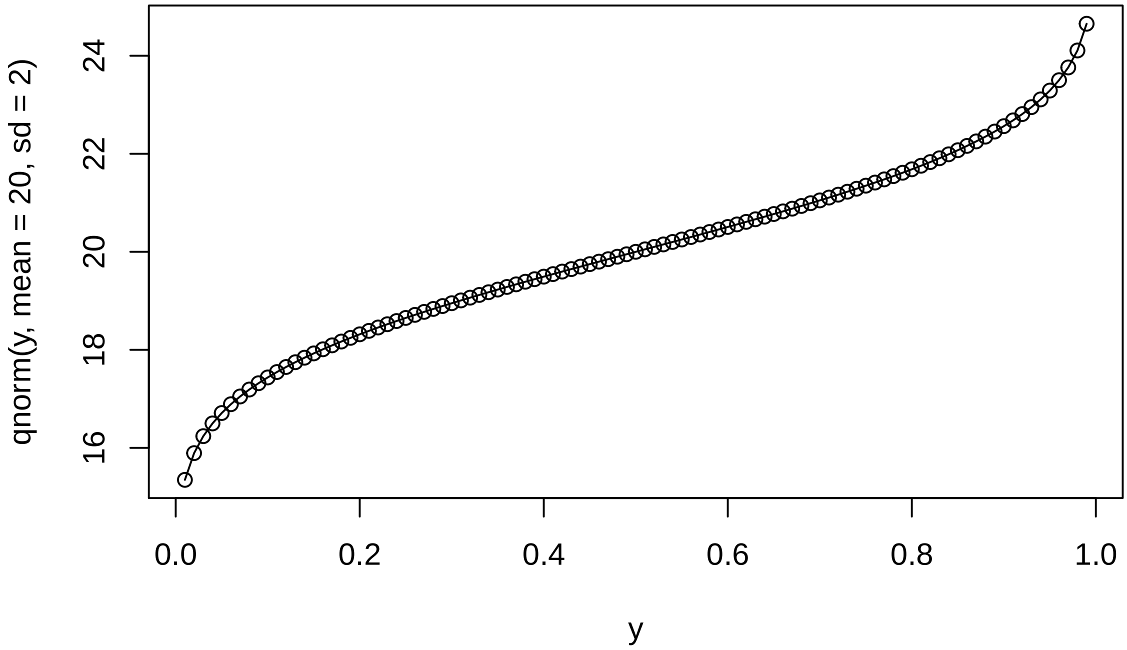
```
## [1] 0.9363903
```

```
pnorm(23, mean=20, sd=2)
```

```
## [1] 0.9331928
```

The function qnorm is an inverse function of pnorm:

```
y<-1:99/100
plot(y, qnorm(y, mean=20, sd=2), type="l")
x<-qnorm(y, mean=20, sd=2)
points(pnorm(x, mean=20, sd=2), x)
```



There are similar functions for other distributions such as chi-squared distribution (dchisq, pchisq, qchisq and rchisq), t-distribution (dt, pt, qt and rt), F-distribution (df, pf, qf and rf) and many others.

0.0.4.1 Tips and tricks

- you can set seed if you want to generate same random numbers:

```
set.seed(666)
rnorm(5)
```

```
## [1]  0.7533110  2.0143547 -0.3551345  2.0281678 -2.2168745
```

```
rnorm(5)
```

```
## [1]  0.75839618 -1.30618526 -0.80251957 -1.79224083 -0.04203245
```

```
set.seed(666)
rnorm(5)
```

```
## [1]  0.7533110  2.0143547 -0.3551345  2.0281678 -2.2168745
```

0.0.5 Univariate descriptive statistics in R

Lets create a sample with mean and standard deviation set to 20 and 2, respectively:

```
x<-rnorm(10, mean=20, sd=2)
```

Basic measures of descriptive statistics, namely minimum, lower quartile, median, mean, upper quartile and maximum, can be obtained by function summary:

```
summary(x)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   16.42   16.77   19.16   19.29   21.20   24.30
```

These values can be accessed by special functions:

```
min(x)
```

```
## [1] 16.41552
```

```
quantile(x, probs=0.25)
```

```
##      25%
## 16.76667
```

```
median(x)
```

```
## [1] 19.15545
```

```
mean(x)
```

```
## [1] 19.29477
```

```
quantile(x, probs=0.75)
```

```
##      75%
## 21.20466
```

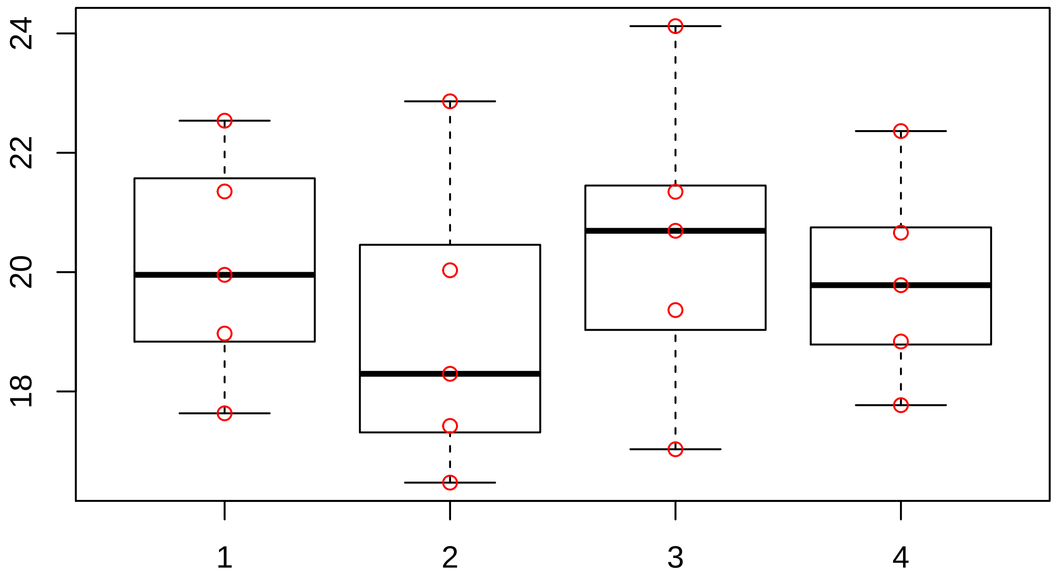
```
max(x)
```

```
## [1] 24.30009
```

Another useful function can be `range`.

Important plot of univariate descriptive statistics is Tukey's box plot. It plots a box with bottom and top at lower and upper quartile (exactly, at values nearest to lower and upper quartile). The horizontal line is located at median. The whiskers start from the bottom and top of the box. Each whisker goes to maximum/minimum, but it is not longer than 1.5-times of interquartile range. It some point is more distant it is depicted as a point. See bellow:

```
x1<-rnorm(10, mean=20, sd=2)
x2<-rnorm(10, mean=20, sd=2)
x3<-rnorm(10, mean=20, sd=2)
x4<-rnorm(10, mean=20, sd=2)
boxplot(x1, x2, x3, x4)
points(1:4, c(min(x1),min(x2),min(x3),min(x4)), col="red")
points(1:4, c(max(x1),max(x2),max(x3),max(x4)), col="red")
points(1:4, c(median(x1),median(x2),median(x3),median(x4)), col="red")
points(1:4, c(quantile(x1,0.25),quantile(x2,0.25),quantile(x3,0.25),quantile(x4,0.25)), col="red")
points(1:4, c(quantile(x1,0.75),quantile(x2,0.75),quantile(x3,0.75),quantile(x4,0.75)), col="red")
```



Mean estimate can be calculated by function `mean`, or manually:

```
mean(x)
```

```
## [1] 19.29477
```

```
sum(x)/length(x)
```

```
## [1] 19.29477
```

Variance estimate can be calculated by function `var`, or manually:

```
var(x)
```

```
## [1] 7.278577
```

```
sum((x-mean(x))**2/(length(x)-1))
```

```
## [1] 7.278577
```

Standard deviation estimate can be calculated by function `sd`, or manually:

```
sd(x)
```

```
## [1] 2.697884
```

```
sqrt(var(x))
```

```
## [1] 2.697884
```

```
sqrt(sum((x-mean(x))**2/(length(x)-1)))
```

```
## [1] 2.697884
```

Standard error of the mean does not have own function in (basic) R, so we can calculate it manually:

```
sd(x)/sqrt(length(x))
```

```
## [1] 0.8531457
```

0.0.5.1 Tips and tricks

- it is possible to index the function `summary`, e.g. to get minimum by index 1:

```
summary(x)[1]
```

```
##     Min.
## 16.41552
```

It is not really useful for this function, but you can use it later for other functions.

0.0.6 Confidence intervals in R

Confidence intervals can be calculated in R, for example, as mean +/- s.e.m. multiplied by quantile of t-distribution.

```
x<-rnorm(10, mean=20, sd=2)
sem<-sd(x)/sqrt(length(x))
mean(x)+sem*qt(p=c(0.025,0.975), df=(length(x)-1))
```

```
## [1] 18.31697 21.51018
```

The function `qt(p=c(0.025,0.975), df=(length(x)-1))` returns quantile of t-distribution for p=0.025 and 0.975, i.e. for 95 % probability. For 90 % use `p=c(0.05,0.95)`, for 99 % use `p=c(0.005,0.995)` etc.

If you generate 100 random samples (each with 10 items) with mean set to 20 and standard deviation set to 2, you should expect that 95 samples will contain 20 in the confidence interval and 5 will not. Let's try:

```
good<-0
for(i in 1:100) {
  x<-rnorm(10, mean=20, sd=2)
  sem<-sd(x)/sqrt(length(x))
  ci<-mean(x)+sem*qt(p=c(0.025,0.975), df=(length(x)-1))
  if((ci[1]<20)&&(ci[2]>20)) {
    good<-good+1
  }
}
good
```

```
## [1] 96
```

I obtained 97, close to expected 95.

0.0.6.1 Tips and tricks

- confidence interval can be obtained more easily by `t.test` as will be shown later:

```
x<-rnorm(10, mean=20, sd=2)
t.test(x)$conf.int
```

```
## [1] 18.72790 20.51576
## attr(,"conf.level")
## [1] 0.95
```

```
t.test(x)$conf.int[1:2]
```

```
## [1] 18.72790 20.51576
```

```
sem<-sd(x)/sqrt(length(x))
mean(x)+sem*qt(p=c(0.025,0.975), df=(length(x)-1))
```

```
## [1] 18.72790 20.51576
```

0.0.7 One-sample t-test in R

Confidence interval and one-sample t-test are two sides of the same coin. Let us calculate 95 % confidence interval for a sample generated by function `rnorm`:

```
x<-rnorm(10, mean=20, sd=2)
sem<-sd(x)/sqrt(length(x))
mean(x)+sem*qt(p=c(0.025,0.975), df=(length(x)-1))
```

```
## [1] 18.40536 21.46257
```

We can make a two-tailed t-test (at the significance level of 5 %) with the null hypothesis that the mean of x is equal to 20. The null hypothesis is rejected if 20 is outside the confidence interval. You can replace `p=c(0.025,0.975)` by `p=c(0.005, 0.995)` for the significance level 1 % etc.

Another option is to calculate criterion R and compare it with corresponding quantile of t-distribution:

```
R<-abs(mean(x)-20)*sqrt(length(x))/sd(x)
R
```

```
## [1] 0.09772072
```

```
qt(p=0.975, df=(length(x)-1))
```

```
## [1] 2.262157
```

The null hypothesis is rejected if R is bigger than the quantile of t-distribution.

The most convinient t-test option is to use the function `t.test`:

```
t.test(x, mu=20)
```

```
##
##  One Sample t-test
##
## data:  x
## t = -0.097721, df = 9, p-value = 0.9243
## alternative hypothesis: true mean is not equal to 20
## 95 percent confidence interval:
##  18.40536 21.46257
## sample estimates:
## mean of x
##  19.93397
```

The null hypothesis is rejected if p-value is lower than 0.05 (or 0.01 for the significance level of 1 %). The function `t.test` also provides the criterion R (called t), degrees of freedom, confidence interval and mean.

One-tailed t-test can be done similarly:

```
mean(x)+sem*qt(p=c(0,0.95), df=(length(x)-1))
```

```
## [1]     -Inf 21.17265
```

```
t.test(x, mu=20, alternative="less")
```

```
##
##  One Sample t-test
##
## data:  x
## t = -0.097721, df = 9, p-value = 0.4621
## alternative hypothesis: true mean is less than 20
## 95 percent confidence interval:
##      -Inf 21.17265
## sample estimates:
## mean of x
##  19.93397
```

```r
mean(x)+sem*qt(p=c(0.05,1), df=(length(x)-1))
```

```
## [1] 18.69528      Inf
```

```r
t.test(x, mu=20, alternative="greater")
```

```
##
##  One Sample t-test
##
## data:  x
## t = -0.097721, df = 9, p-value = 0.5379
## alternative hypothesis: true mean is greater than 20
## 95 percent confidence interval:
##  18.69528      Inf
## sample estimates:
## mean of x
##  19.93397
```

0.0.7.1 Tips and tricks

- you can iterate on the results of the function t.test:

```r
tt<-t.test(x, mu=20)
tt
```

```
##
##  One Sample t-test
##
## data:  x
## t = -0.097721, df = 9, p-value = 0.9243
## alternative hypothesis: true mean is not equal to 20
## 95 percent confidence interval:
##  18.40536 21.46257
## sample estimates:
## mean of x
##  19.93397
```

```r
tt[3]
```

```
## $p.value
## [1] 0.9242958
```

- you can change the significance level for the confidence interval by parameter `conf.level`:

```r
t.test(x, mu=20)
```

```
##
##  One Sample t-test
##
## data:  x
## t = -0.097721, df = 9, p-value = 0.9243
## alternative hypothesis: true mean is not equal to 20
## 95 percent confidence interval:
##  18.40536 21.46257
## sample estimates:
## mean of x
##  19.93397
```

```r
t.test(x, mu=20, conf.level=0.99)
```

```
##
##  One Sample t-test
```

```
##
## data:  x
## t = -0.097721, df = 9, p-value = 0.9243
## alternative hypothesis: true mean is not equal to 20
## 99 percent confidence interval:
##  17.73796 22.12997
## sample estimates:
## mean of x
##  19.93397
```

```r
t.test(x, mu=20, conf.level=0.999)
```

```
##
##  One Sample t-test
##
## data:  x
## t = -0.097721, df = 9, p-value = 0.9243
## alternative hypothesis: true mean is not equal to 20
## 99.9 percent confidence interval:
##  16.70337 23.16456
## sample estimates:
## mean of x
##  19.93397
```

0.0.8 Two-sample t-test in R

Let us skip a "manual" version of the t-test and proceed directly to the function `t.test`. There are two variants of two-sample t-test, one for equal variances and one for unequal variances. First let us test whether the variances are equal:

```r
healthy<-rnorm(10, mean=12.3, sd=3.3)
healthy
```

```
##  [1] 12.596588 13.572485 10.099827 10.561580  9.708413 11.412490 15.757506
##  [8]  8.584591 14.155399 14.302581
```

```r
sick<-rnorm(10, mean=8.5, sd=3.3)
sick
```

```
##  [1]  5.909912  3.763672 13.606515  9.789544  7.131921  9.517881  9.578317
##  [8]  9.001416 12.707699  9.659650
```

```r
var.test(healthy, sick)
```

```
##
##  F test to compare two variances
##
## data:  healthy and sick
## F = 0.64615, num df = 9, denom df = 9, p-value = 0.5256
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  0.1604936 2.6013805
## sample estimates:
## ratio of variances
##          0.6461461
```

The null hypothesis of the `var.test` is that variance of both samples are equal. We can reject the null hypothesis when p-value is lower than 0.05 (for the significance level of 5 %).

The null hypothesis of t-test is that both means are equal. We can reject the null hypothesis when p-value is lower than 0.05 (for the significance level of 5 %). For samples with equal variances we will use t-test with `var.equal=TRUE`. For samples with unequal variances we will use t-test with `var.equal=FALSE` (default):

```
t.test(healthy, sick, var.equal=TRUE)
```

```
##
##  Two Sample t-test
##
## data:  healthy and sick
## t = 2.5344, df = 18, p-value = 0.02077
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.5145616 5.5024248
## sample estimates:
## mean of x mean of y
## 12.075146  9.066653
```

```
t.test(healthy, sick, var.equal=FALSE)
```

```
##
##  Welch Two Sample t-test
##
## data:  healthy and sick
## t = 2.5344, df = 17.205, p-value = 0.02125
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.5062757 5.5107107
## sample estimates:
## mean of x mean of y
## 12.075146  9.066653
```

Paired t-test is used when values in both samples can be paired, e.g. one sample represents blood pressure of patients before and one sample after treatment. It is better to evaluate differences for individual patients one by one, rather than whole samples. The t-test can be switched to paired by `paired=TRUE`:

```
x<-rnorm(10, mean=20, sd=5)
x
```

```
##  [1] 21.857982 17.924920  7.460842 12.732419 21.253695 25.524013 21.579099
##  [8] 21.658019 22.710094 22.366305
```

```
y<-x+rnorm(10, mean=2, sd=0.5)
y
```

```
##  [1] 23.52077 19.22655 10.37449 14.20998 22.78327 27.48872 24.22182
##  [8] 24.61163 25.39220 24.07541
```

```
t.test(x,y)
```

```
##
##  Welch Two Sample t-test
##
## data:  x and y
## t = -0.8602, df = 17.999, p-value = 0.401
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -7.173033  3.005540
## sample estimates:
## mean of x mean of y
##  19.50674  21.59049
```

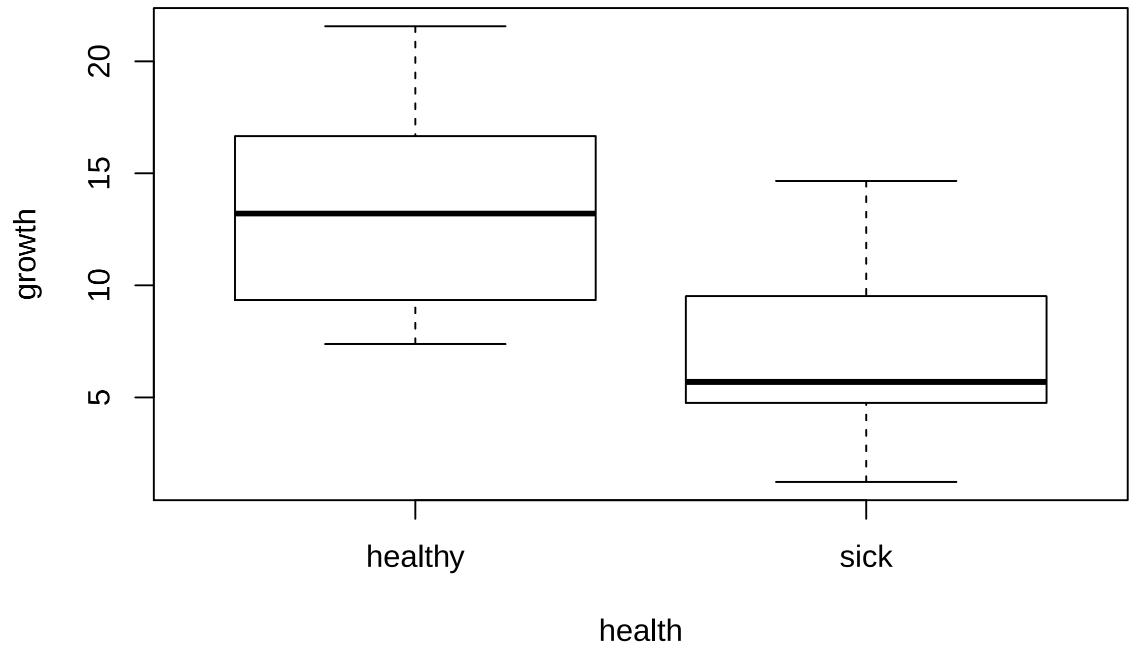
```
t.test(x,y, paired=TRUE)
```

```
##
##  Paired t-test
##
## data:  x and y
## t = -10.23, df = 9, p-value = 2.96e-06
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -2.544514 -1.622978
## sample estimates:
## mean of the differences
##               -2.083746
```

0.0.8.1 Tips and tricks

- the function `t.test` always gives a résumé on the alternative hypothesis, you can use it if you are not sure which variant of test should be used
- the function `t.test` (as well as `plot`) can use class 'formula' as the input. We will use it frequently in next lessons, so let us try it now:

```
healthy<-rnorm(10, mean=12.3, sd=3.3)
sick<-rnorm(10, mean=8.5, sd=3.3)
growth<-c(healthy, sick)
health<-rep(c("healthy", "sick"), each=10)
df<-data.frame(health, growth)
plot(growth~health, data=df)
```




```
t.test(growth~health, data=df)
```

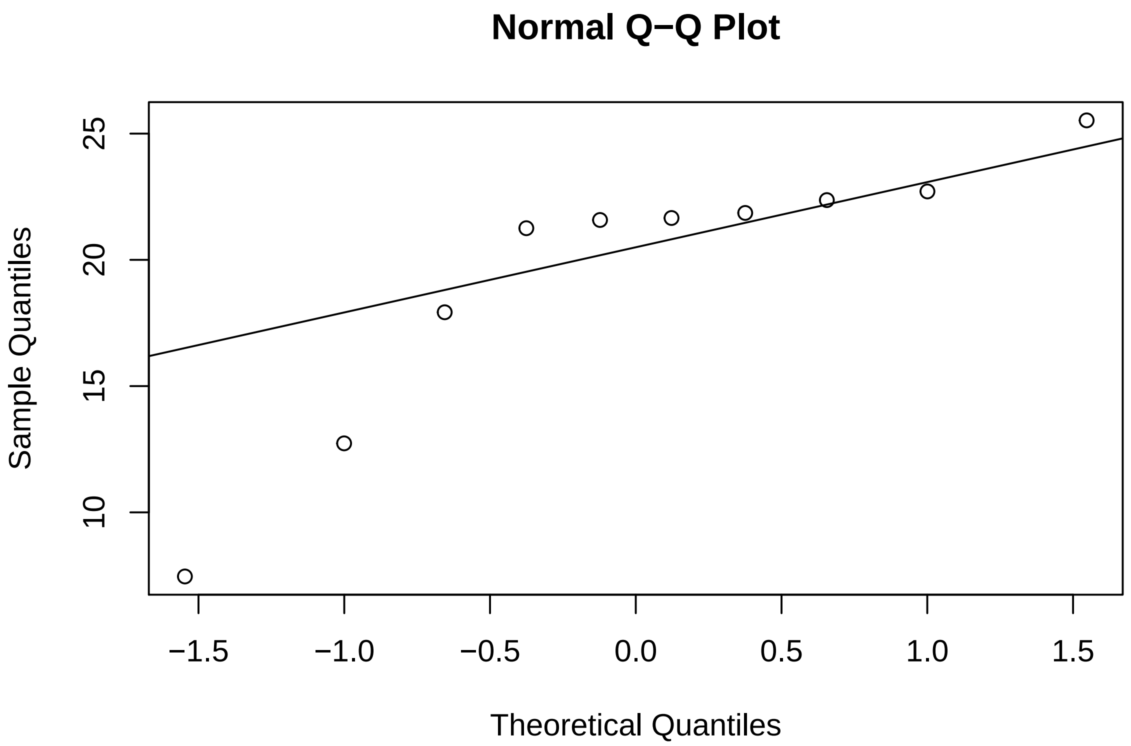
```
##
##  Welch Two Sample t-test
##
## data:  growth by health
```

```
## t = 3.1218, df = 17.969, p-value = 0.005901
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##   2.010189 10.287258
## sample estimates:
## mean in group healthy    mean in group sick
##             13.247098              7.098375
```

0.0.9 Non-parametric tests

Up to now we considered normal distribution of a variable. Here we will show how we can test whether the variable follows the normal distribution and what can we do if data do not follow normal distribution. There is a graphical tool to do that known as QQ-plot.

```
qqnorm(x)
qqline(x)
```



The function calculates the mean and the standard deviation for the sample and from this it calculates quantiles. Then it plots the values of the sample vs. quantiles. This plot should be linear. If not, it means that the distribution is right or left skewed, bimodal or non-normal in some other way.

QQ-plot is good for visual evaluation, but for a quantitative evaluation it is useful to use some test of normality. One of them is the test developed by Shapiro and Wilk. You can run it by:

```
x<-rnorm(20)
shapiro.test(x)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  x
## W = 0.94727, p-value = 0.3276
```

Let us try with something non-normal

```r
x<-c(rnorm(10), rnorm(10, mean=4))
shapiro.test(x)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  x
## W = 0.90985, p-value = 0.06332
```

The null hypothesis is that the sample follows the normal distribution.

What about if data do not follow the normal distribution? A non-parametric (i.e. not requiring normal distribution) alternative to t-test is Wilcoxon test. The two-sample variant is also known as Mann-Whitney test. We can use the function `wilcox.test`. It is used the same way as t-test:

```r
x<-rnorm(10)
y<-rnorm(10, mean=2)
wilcox.test(x,y)
```

```
##
##  Wilcoxon rank sum test
##
## data:  x and y
## W = 1, p-value = 2.165e-05
## alternative hypothesis: true location shift is not equal to 0
```

Let us try to compare with t-test:

```r
t.test(x,y)
```

```
##
##  Welch Two Sample t-test
##
## data:  x and y
## t = -5.5487, df = 17.62, p-value = 3.107e-05
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -3.123008 -1.405660
## sample estimates:
##  mean of x  mean of y
## -0.3262392  1.9380948
```

0.0.9.1 Tips and tricks

- non-parametric variant of analysis of variance will be shown later.

0.0.10 Analysis of Variance

Analysis of Variance (ANOVA) is an extension of t-test to more than two samples. The null hypothesis is that all samples have the same mean. We simply cannot do a pairwise t-test because this increases probability of rejection of the null hypothesis simply by chance. First, let us show a "manual" version of ANOVA for three samples, one representing a biological parameter of patients who were administered a drug, one representing a control group and one representing patients administered a placebo:

```r
drug<-rnorm(10, mean=70, sd=30)
control<-rnorm(10, mean=100, sd=25)
placebo<-rnorm(10, mean=90, sd=25)
drug
```

```
##  [1]  66.72068  56.33683  81.62437  83.51047 116.77867  47.73066  84.01318
##  [8]  58.20104  57.83262  61.71645
```

```
control
```

```
##  [1]  61.57471 105.89935  94.36642 147.30065  96.95527 101.27382  98.30794
##  [8]  53.29537 106.48794  64.26199
```

```
placebo
```

```
##  [1] 100.11741  65.28788 105.92620  46.17389  85.25819  73.85201 119.71384
##  [8] 124.62775 110.25343  81.03393
```

First, let us calculate variances in each group:

```
sdrug<-sum((drug-mean(drug))^2)
scontrol<-sum((control-mean(control))^2)
splacebo<-sum((placebo-mean(placebo))^2)
```

We will sum this and we will call it sum of squares within the group (SSW):

```
SSW<-scontrol+sdrug+splacebo
```

Next, we will concatenate all samples and calculate mean of this supersample. We will call the variance of the supersample as sum of squares total (SST):

```
all<-c(control, drug, placebo)
SST<-sum((all-mean(all))^2)
```

In an extreme example that means of all samples are the same, the SSW and SST are the same, otherwise SST is bigger than SSW. The difference of SST and SSW is thus a measure of difference between the samples. We will cal this sum of squares between the groups (SSB):

```
SSB<-SST-SSW
```

The criterion with the F-distribution is calculated as:

```
FE<-(SSB*27)/(SSW*2)
FE
```

```
## [1] 2.373428
```

with two degrees of freedom, 27 is the total number of values minus number of samples (30-3), and 2 is number of samples minus 1. Finally, we will compare this with the value of the quantile of F-distribution:

```
qf(p=0.95, df1=2, df2=27)
```

```
## [1] 3.354131
```

If the `FE` is higher than `qf` we can reject the null hypothesis (i.e. that means of samples are the same).

In an automated way we can make use of a data frame:

```
labels<-rep(c("control", "drug", "placebo"), each=10)
all<-c(control, drug, placebo)
df<-data.frame(labels, all)
```

We will make a model by the function analysis of variance and we will obtain all results by summary of the model:

```
mymodel<-aov(all~labels, data=df)
mymodel
```

```
## Call:
##    aov(formula = all ~ labels, data = df)
##
## Terms:
##                    labels Residuals
## Sum of Squares   2858.616 16259.737
## Deg. of Freedom         2        27
```

```
##
## Residual standard error: 24.54002
## Estimated effects may be unbalanced
```

```
summary(mymodel)
```

```
##             Df Sum Sq Mean Sq F value Pr(>F)
## labels       2   2859  1429.3   2.373  0.112
## Residuals   27  16260   602.2
```

We can reject the null hypothesis on the basis of the p-value. The same result can be obtained by:

```
anova(lm(all~labels, data=df))
```

```
## Analysis of Variance Table
##
## Response: all
##           Df  Sum Sq Mean Sq F value Pr(>F)
## labels     2  2858.6 1429.31  2.3734 0.1123
## Residuals 27 16259.7  602.21
```

Data for ANOVA must follow normal distribution and there must be homogeneous variances of samples. For other than normal distribution try data transformation or Kruskal-Walis test (bellow). For different variances try transformation.

0.0.10.1 Tips and Tricks

- to test normality of data you can use the function `mshapiro.test`. It is a multivariate alternative to Shapiro and Wilk test.
- to test homogeneity of variances you can use `bartlett.test` or `fligner.test`.
- Kruskal-Wallis rank sum test (`kruskal.test`) is a non-parametric alternative to ANOVA:

```
kruskal.test(all~labels, data=df)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  all by labels
## Kruskal-Wallis chi-squared = 4.2916, df = 2, p-value = 0.117
```

- two-way ANOVA will be introduced together with linear models.

0.0.11 P-value adjustment and other approaches for multiple comparisons

In the previous chapter we have shown ANOVA on drug testing example:

```
drug<-rnorm(10, mean=70, sd=30)
control<-rnorm(10, mean=100, sd=25)
placebo<-rnorm(10, mean=90, sd=25)
labels<-rep(c("control", "drug", "placebo"), each=10)
all<-c(control, drug, placebo)
df<-data.frame(labels, all)
mymodel<-aov(all~labels, data=df)
mymodel
```

```
## Call:
##    aov(formula = all ~ labels, data = df)
##
## Terms:
##                   labels Residuals
## Sum of Squares    136.47  15732.58
```

```
## Deg. of Freedom        2        27
##
## Residual standard error: 24.13893
## Estimated effects may be unbalanced
```

```
summary(mymodel)
```

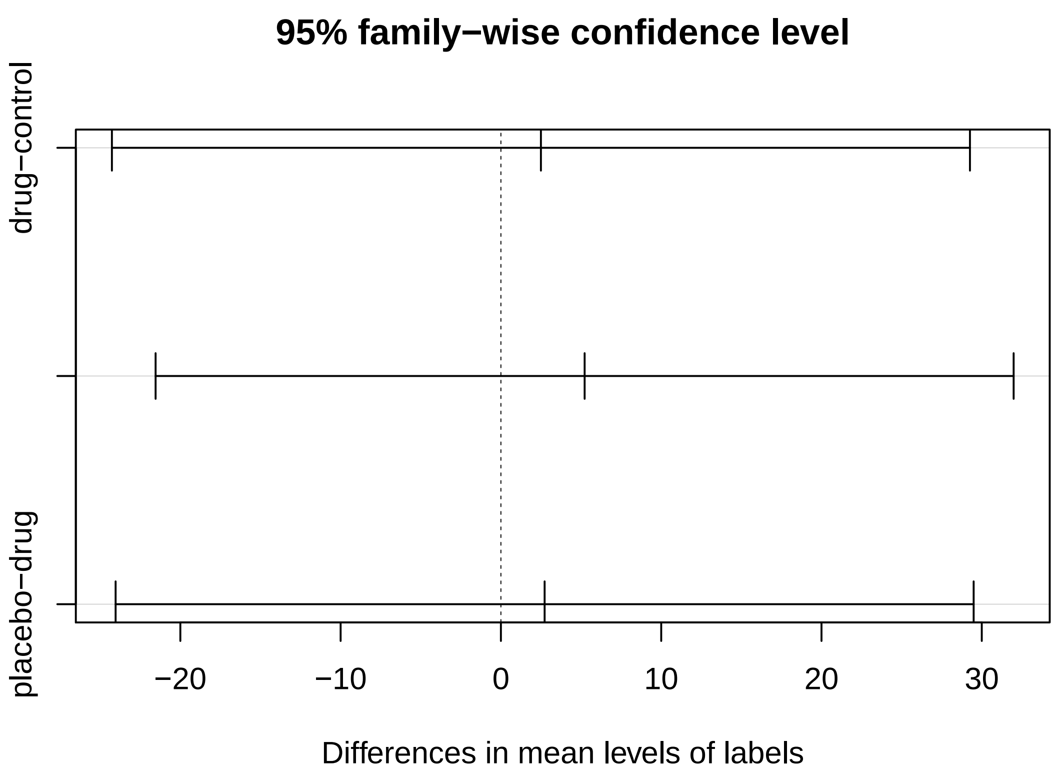
```
##             Df Sum Sq Mean Sq F value Pr(>F)
## labels       2    136    68.2   0.117   0.89
## Residuals   27  15733   582.7
```

This shows that there is a difference between the means. Next we want to know which samples are statistically significantly lower and higher. Again we cannot make a pairwise t-tests because of probability of rejection of the null hypothesis by chance. Instead we can use Tukey Honest Significance Test:

```
TukeyHSD(mymodel)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = all ~ labels, data = df)
##
## $labels
##                     diff       lwr      upr     p adj
## drug-control    2.495872 -24.27008 29.26183 0.9709940
## placebo-control 5.222671 -21.54328 31.98863 0.8795351
## placebo-drug    2.726799 -24.03916 29.49275 0.9654842
```

```
plot(TukeyHSD(mymodel))
```



Alternatively it is possible to do a pairwise t-test and adjust p-values by `pairwise.t.test`:

```
pairwise.t.test(all, labels, p.adjust.method="none")
```

```
##
```

```
##  Pairwise comparisons using t tests with pooled SD
##
## data:  all and labels
##
##         control drug
## drug    0.82    -
## placebo 0.63    0.80
##
## P value adjustment method: none
```

The option `p.adjust.method` can be "none" (no adjustment), "bonferroni" (Bonferroni correction), "holm" (Holm and Bonferroni) and "BH" or "fdr" (Benjamini and Hochberg). There is also an option `pool.sd` defining whether variances are homogeneous.

0.0.11.1 Tips and Tricks

- in biological sciences we often compare every sample with a single control. For this it is useful to use Dunnett test. It requires package `multicomp`:

```
require(multcomp)
```

```
## Loading required package: multcomp

## Loading required package: mvtnorm

## Loading required package: survival

## Loading required package: TH.data

## Loading required package: MASS

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select

##
## Attaching package: 'TH.data'

## The following object is masked from 'package:MASS':
##
##     geyser
```

```
mydata <- data.frame(labels, all)
```

We must define that the first sample is the control:

```
mydata$labels <- relevel(mydata$labels, ref=1)
```

Finally, we will do the Dunnett test:

```
mydata.aov <- aov(all~labels, data=mydata)
mydata.dunnett <- glht(mydata.aov, linfct = mcp(labels="Dunnett"))
summary(mydata.dunnett)
```

```
##
##   Simultaneous Tests for General Linear Hypotheses
##
## Multiple Comparisons of Means: Dunnett Contrasts
##
##
## Fit: aov(formula = all ~ labels, data = mydata)
##
```

```
## Linear Hypotheses:
##                        Estimate Std. Error t value Pr(>|t|)
## drug - control == 0       2.496     10.795   0.231    0.962
## placebo - control == 0    5.223     10.795   0.484    0.845
## (Adjusted p values reported -- single-step method)
```

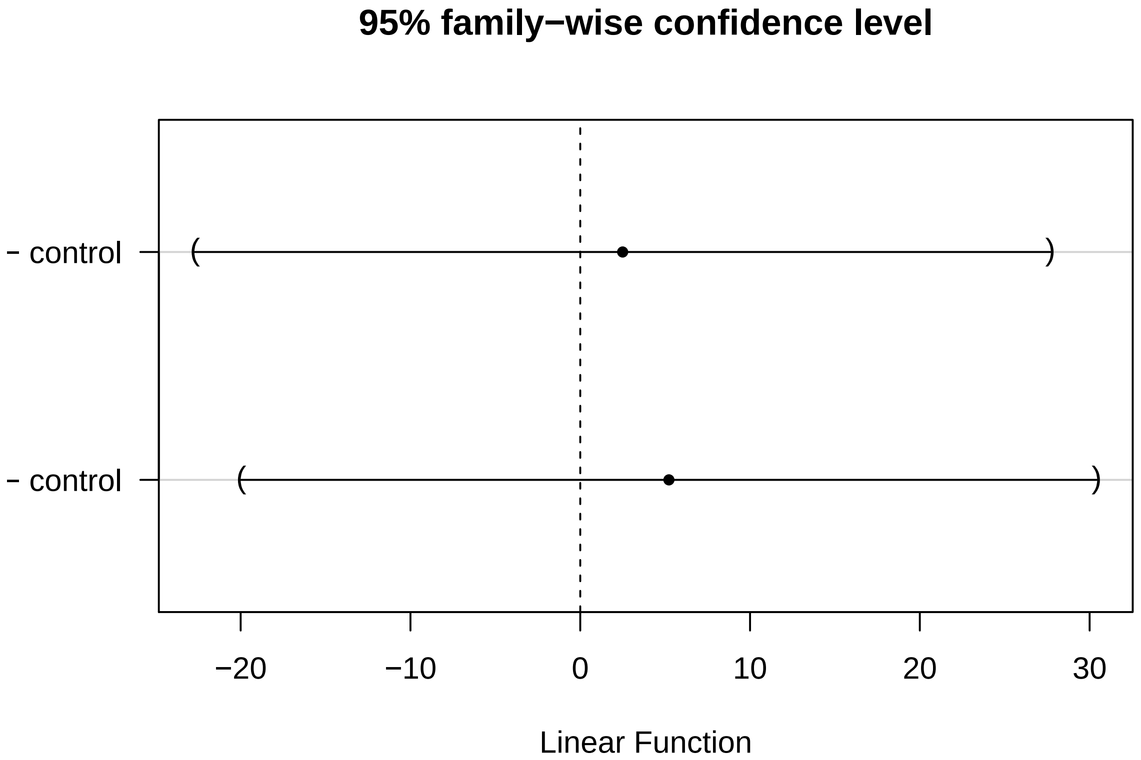
Confidence intervals can be calculated as:

```r
confint(mydata.dunnett)
```

```
##
##   Simultaneous Confidence Intervals
##
## Multiple Comparisons of Means: Dunnett Contrasts
##
##
## Fit: aov(formula = all ~ labels, data = mydata)
##
## Quantile = 2.3335
## 95% family-wise confidence level
##
##
## Linear Hypotheses:
##                        Estimate lwr      upr
## drug - control == 0      2.4959 -22.6953  27.6870
## placebo - control == 0   5.2227 -19.9685  30.4138
```

You can also make a plot:

```r
plot(mydata.dunnett)
```



- in biological sciences it is popular to use barplots and other similar plots with star symbols indicating the significance of a test, often with a horizontal lines that connect tested samples. In R you can generate such plots by the package "ggplot2" with "ggsignif" and "ggpubr".

0.0.12 Bi- and multivariate descriptive statistics

Let us generate a set of two well correlating variables x and y:

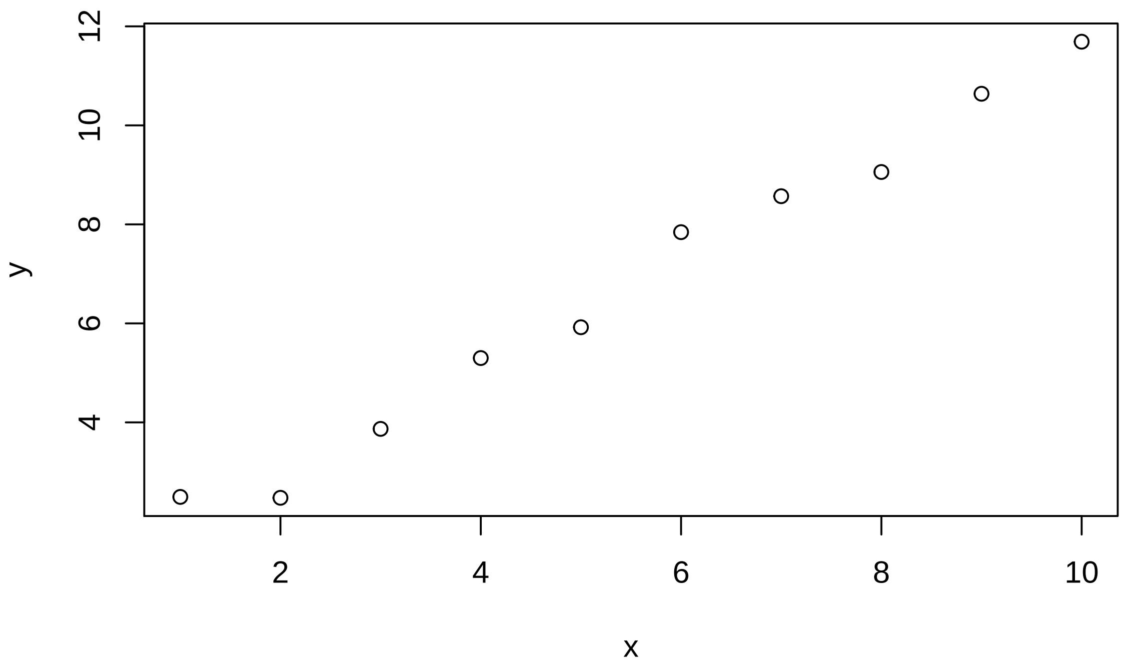
```
x<-1:10
y<-2:11+rnorm(10, sd=0.5)
x
```

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

```
y
```

```
##  [1]  2.494846  2.476638  3.869040  5.299605  5.921994  7.842900  8.569087
##  [8]  9.058114 10.638189 11.689659
```

```
plot(x,y)
```



Covariance can be calculated manually as:

```
sum((x-mean(x))*(y-mean(y)))/(length(x)-1)
```

```
## [1] 9.864383
```

Pearson correlation can be calculated as:

```
sum((x-mean(x))*(y-mean(y)))/sqrt(sum((x-mean(x))^2)*sum((y-mean(y))^2))
```

```
## [1] 0.9934884
```

In R they can be calculated as:

```
cov(x,y)
```

```
## [1] 9.864383
```

```
cor(x,y)
```

```
## [1] 0.9934884
```

Correlation can be calculated by dividing covariance by standard deviations of both variables:

```
cov(x,y)/(sd(x)*sd(y))
```

```
## [1] 0.9934884
```

0.0.12.1 Tips and Tricks

- it is possible to apply these functions on data frame or matrix. This will make a pairwise correlation of all columns.

0.0.13 Linear models

In R you can use function `lm` to build a linear model. It can fit a dependent variable by one or multiple independent variables. Independent variables can quantitative, categorical or both.

```
x<-1:10
y<-2:11+rnorm(10, sd=0.5)
linfit <- lm(y~x)
linfit
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Coefficients:
## (Intercept)            x
##      1.4602       0.9131
```

This will show that “y” grows with “x”. However, if you fit two noisy variables you will always obtain a result that “y” grows or descends with “x”, there almost no chance to get zero slope even if the two variables completely uncorrelated. The question is whether the non-zero slope is statistically significant. This you can learn by function `summary`:

```
summary(linfit)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.34555 -0.27653 -0.07201  0.24391  0.47553
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.46017    0.22603    6.46 0.000196 ***
## x            0.91309    0.03643   25.07 6.87e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3309 on 8 degrees of freedom
## Multiple R-squared:  0.9874, Adjusted R-squared:  0.9859
## F-statistic: 628.3 on 1 and 8 DF,  p-value: 6.869e-09
```

This is an extension of ANOVA in the way that the independent variable is not categorical (such as “control”, “placebo” and “drug”) but it is quantitative. The testing procedure is similar to that of ANOVA, the program calculates a sum of squares of error under assumption of null and alternative hypothesis and compares them.

There are several possibilities to describe models in the lm function:

function	expression in `lm`
f(x) = alpha	`y~1`
f(x) = alpha + beta x	`y~x`
f(x) = beta x	`y~-1+x`
f(x) = alpha + beta x + gamma x^2	`y~x+I(x^2)`
f(x) = alpha + beta_1 x_1 + beta_2 x_2	`y~x1+x2`
f(x) = alpha + beta_1 x_1 + beta_2 x_2 + gamma x_1 x_2	`y~x1*x2`

To get values of coefficients you can print coefficients:

```
linfit$coefficients
```

```
## (Intercept)           x
##   1.4601671   0.9130868
```

They can be iterated:

```
linfit$coefficients[1]
```

```
## (Intercept)
##    1.460167
```

```
linfit$coefficients[2]
```

```
##         x
## 0.9130868
```

Alternatively you may use the function `coef`:

```
coef(linfit)[1]
```

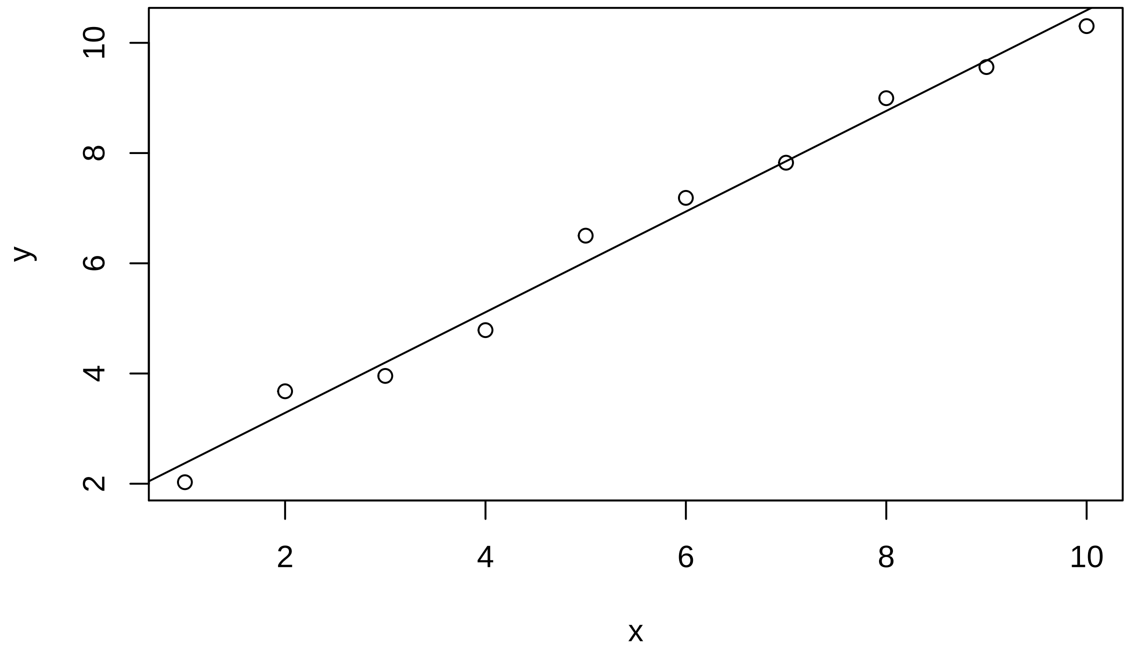
```
## (Intercept)
##    1.460167
```

```
coef(linfit)[2]
```

```
##         x
## 0.9130868
```

To plot a model into data you can use function `abline`:

```
plot(x,y)
abline(linfit)
```



The function `predict` predicts values of y for values of x based on the model. If you use:

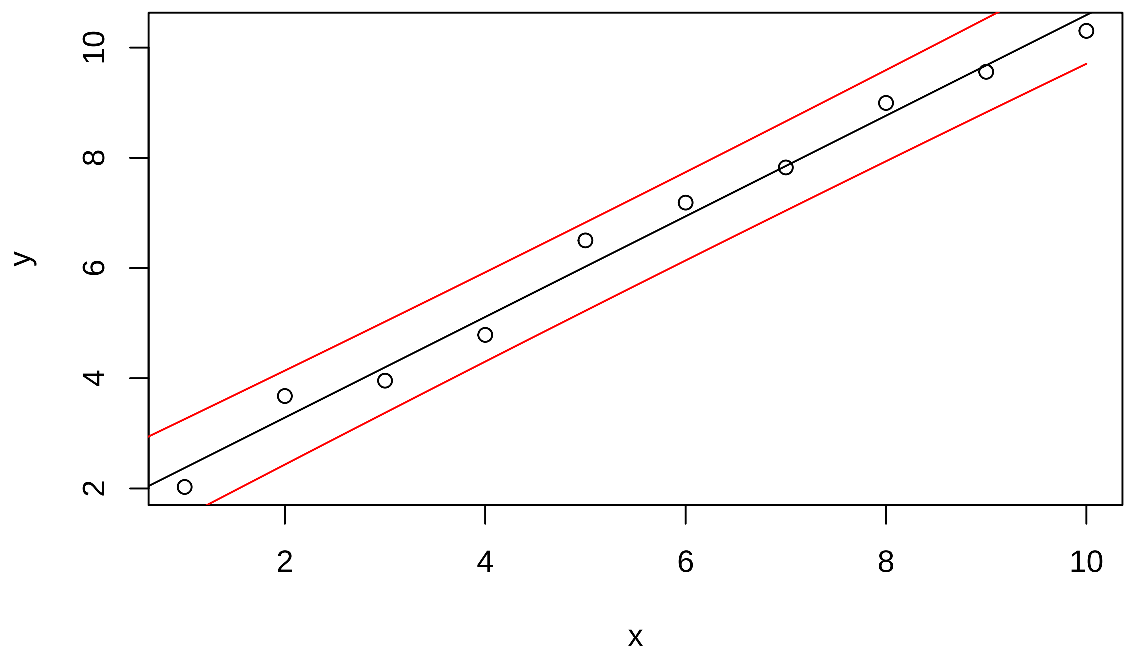
```
newy<-predict(object=linfit)
```

it will calculate values of y for each x by a linear model. If you want to calculate this for some other values of x (here called "newx") you can type:

```
newx<-0:100/10
newy<-predict(object=linfit, newdata=data.frame(x=newx))
```

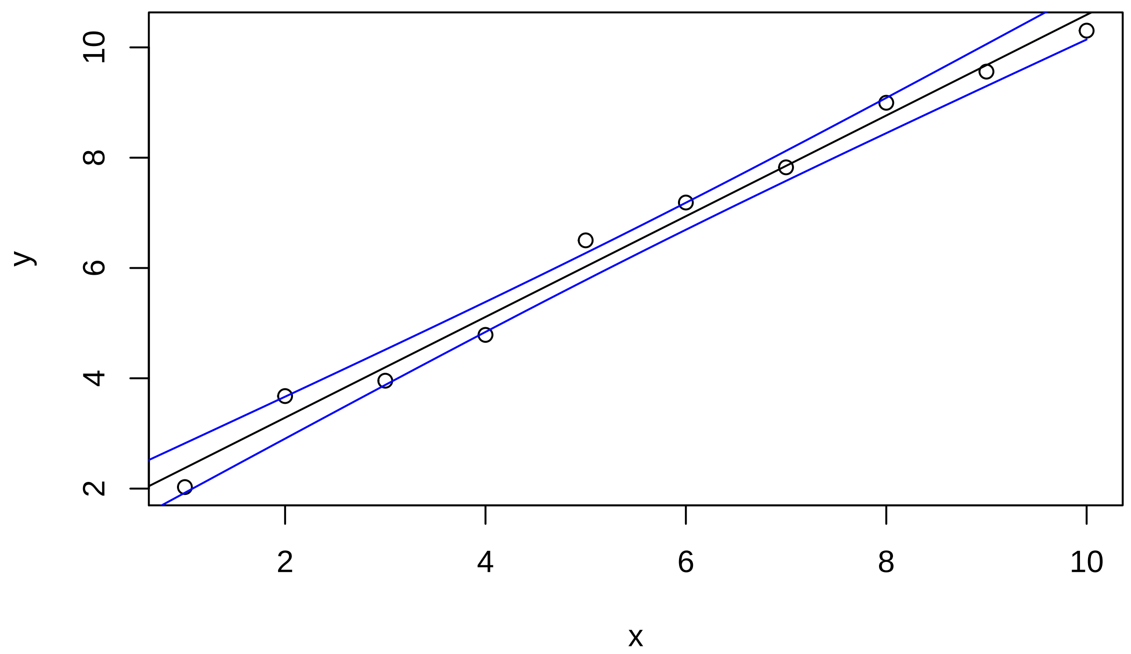
More interesting is calculation of confidence intervals. You can use:

```
newx<-0:100/10
newy<-predict(object=linfit, newdata=data.frame(x=newx), interval = 'prediction')
plot(x,y)
abline(linfit)
lines(newx, newy[,2], col="red")
lines(newx, newy[,3], col="red")
```



This will plot an interval to which 95 % of samples should fall (the level could be changed by option `level`). If you change "prediction" to "confidence" it will print confidence intervals for the model. Provided that there is some exact linear relationships between x and (inaccurately measured) y, we can accurately determine this relationship by doing an infinite number of measurements. If we do enough measurements we can get a vary narrow confidence interval:

```
newx<-0:100/10
newy<-predict(object=linfit, newdata=data.frame(x=newx), interval = 'confidence')
plot(x,y)
abline(linfit)
lines(newx, newy[,2], col="blue")
lines(newx, newy[,3], col="blue")
```



Prediction intervals are analogous to standard deviation, confidence intervals to standard error of the mean.

0.0.13.1 Tips and Tricks

- you can compare models "model1" and "model2" by `anova(model2, model1)`. This will tell you whether "model2" is significantly better than "model1".
- as already mentioned, linear models can use continuous as well as categorical independent variables. In order to do ANOVA with two factors use a linear model with `y~x1+x2`. In order to do ANOVA with two factors and their interactions use a linear model with `y~x1*x2`. Beside other ANOVA presumptions (normal distribution, homogeneity of variances) it is necessary to
- contingency tables are an alternative to ANOVA with categorical dependent and independent variables. To test statistical significance you can use `chisq.test` function.

0.0.14 Principal Component Analysis

Principal Component Analysis (PCA) is frequently used data analysis method. Let us demonstrate it on the results of Tour de France 2013. You can load the data by typing:

```
tdf <- read.table("https://raw.githubusercontent.com/spiwokv/Rtutorial/master/data/tourdefrance2013.
```

Every frame corresponds to a single rider (riders that did not finish the race were removed). The numbers in the table corresponds to their order in each stage. Each column corresponds to one stage. One stage was removed. It was a team time trial where order is given to teams rather than individual riders. It is very complicated to somehow visualize this data, because they are 20-dimensional (because of 20 stages). This is a great opportunity for PCA. Let us try on column 3-22 containing the results:

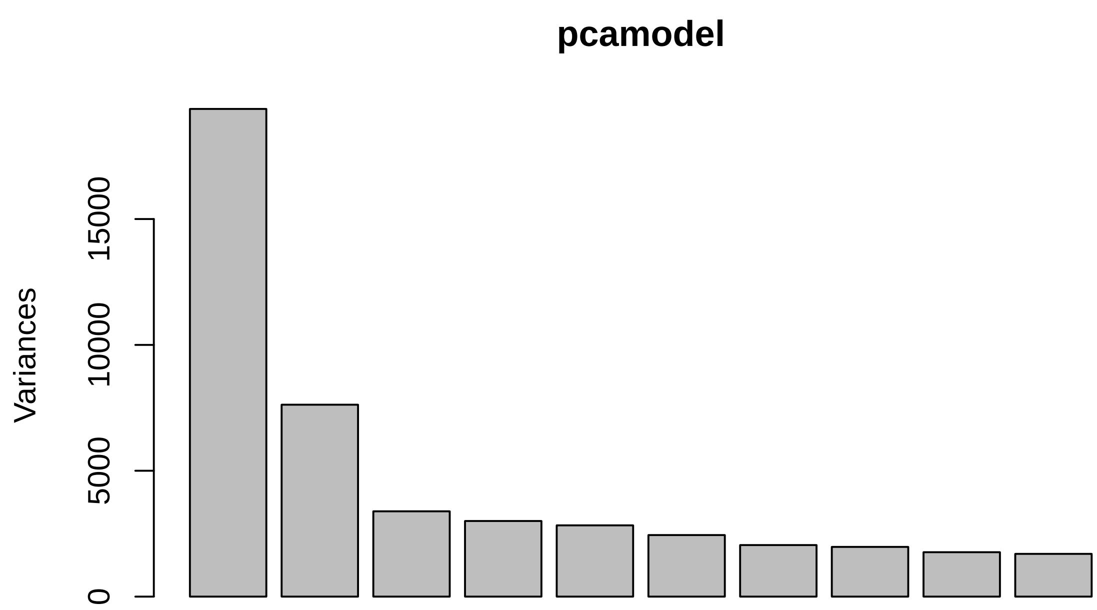
```
pcamodel <- prcomp(tdf[,3:22])
summary(pcamodel)
```

```
## Importance of components:
```

```
##                            PC1     PC2      PC3      PC4     PC5      PC6
## Standard deviation     139.1883 87.3118 58.21738 54.80780 53.2064 49.44327
## Proportion of Variance   0.3497  0.1376  0.06118  0.05422  0.0511  0.04413
## Cumulative Proportion    0.3497  0.4873  0.54851  0.60274  0.6538  0.69797
##                            PC7      PC8      PC9     PC10    PC11     PC12
## Standard deviation     45.24812 44.44331 42.01020 41.20537 39.0335 37.1382
## Proportion of Variance  0.03696  0.03566  0.03186  0.03065  0.0275  0.0249
## Cumulative Proportion   0.73493  0.77058  0.80244  0.83309  0.8606  0.8855
##                           PC13     PC14     PC15     PC16     PC17
## Standard deviation     36.58104 31.85483 30.65858 27.58704 26.54836
## Proportion of Variance  0.02416  0.01832  0.01697  0.01374  0.01272
## Cumulative Proportion   0.90965  0.92797  0.94493  0.95867  0.97140
##                           PC18     PC19     PC20
## Standard deviation     24.58404 23.35946 20.84627
## Proportion of Variance  0.01091  0.00985  0.00784
## Cumulative Proportion   0.98231  0.99216  1.00000
```

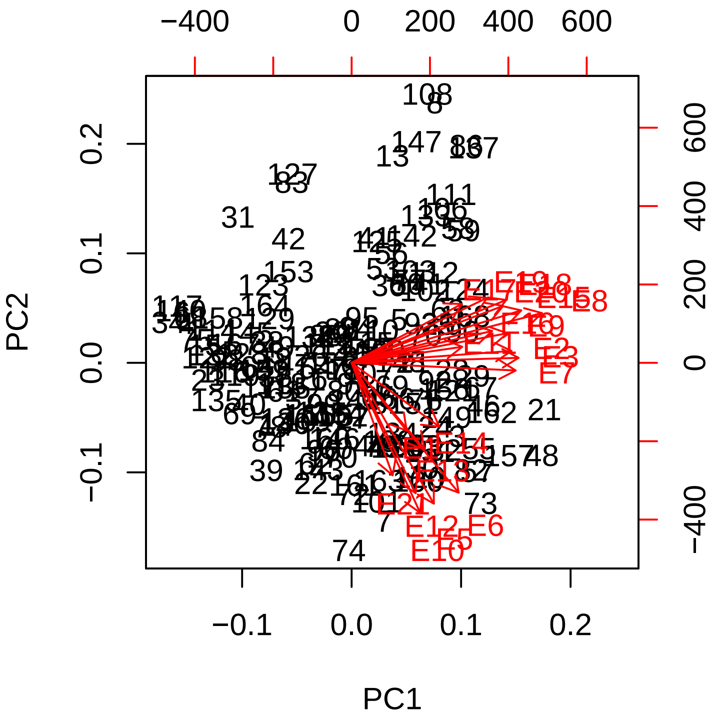
To plot importance of components you can use `plot` function on the model:

```
plot(pcamodel)
```



This shows that there is 1-2 important components describing the data very well. You can plot PC1 vs. PC2:

```
biplot(pcamodel)
```



Each number in black corresponds to the number of line in the `tdf` table. You can print their names as:

```
tdf[108,2]
```

```
## [1] Mark_Cavendish_GBr_Omega_Pharma_Quick_Step
## 170 Levels: Adam_Hansen_Aus_Lotto_Belisol ... Yury_Trofimov_Rus_Katusha
```

to reveal the identity of the rider 108. The red arrows show the relationships between the original data and the results of the PCA. For example, some arrows point right and some point to the bottom. The former correspond to mountain stages. The later to flat stages.

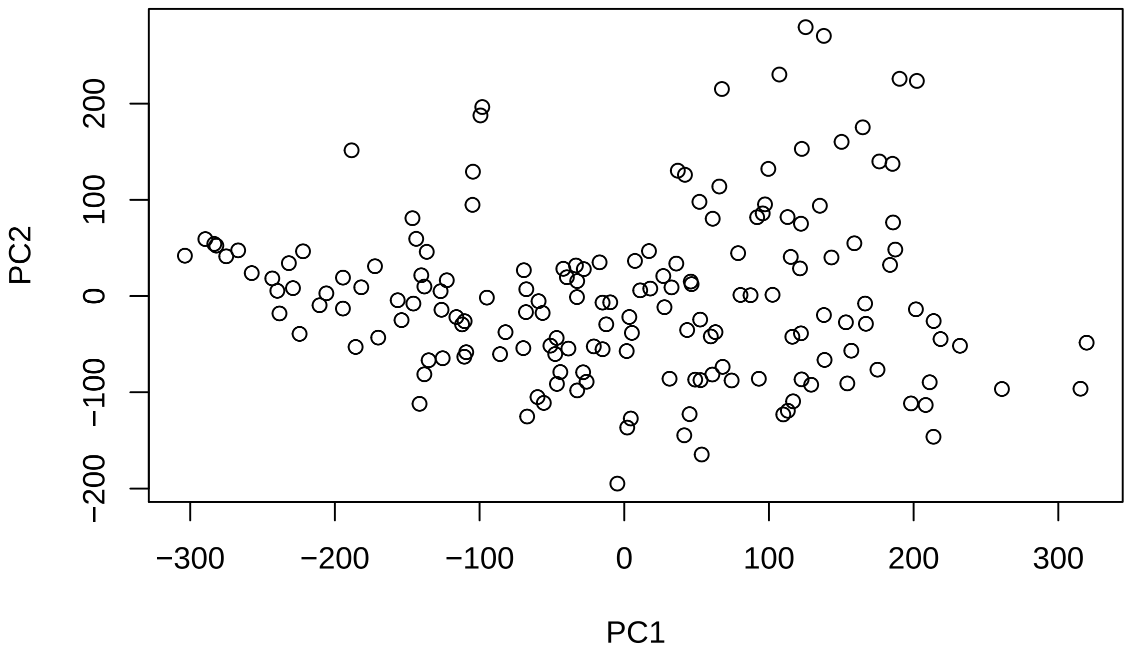
From the results of PCA you can recognize that rider on left side of the plot are those who reached best in the overall classification of the race. These are typically slim mountain stage specialists who scored well in mountain stages (the arrow points right, more right means higher place in a mountain stage).

Riders at the top reached good results in flat stages. These are typically masculine riders who perform poorly in mountain stages. Riders in the bottom right cloud are “domestiques” who don’t care much about their own results or rides who bet on just one stage and performed poorly in other stages.

0.0.14.1 Tips and Tricks

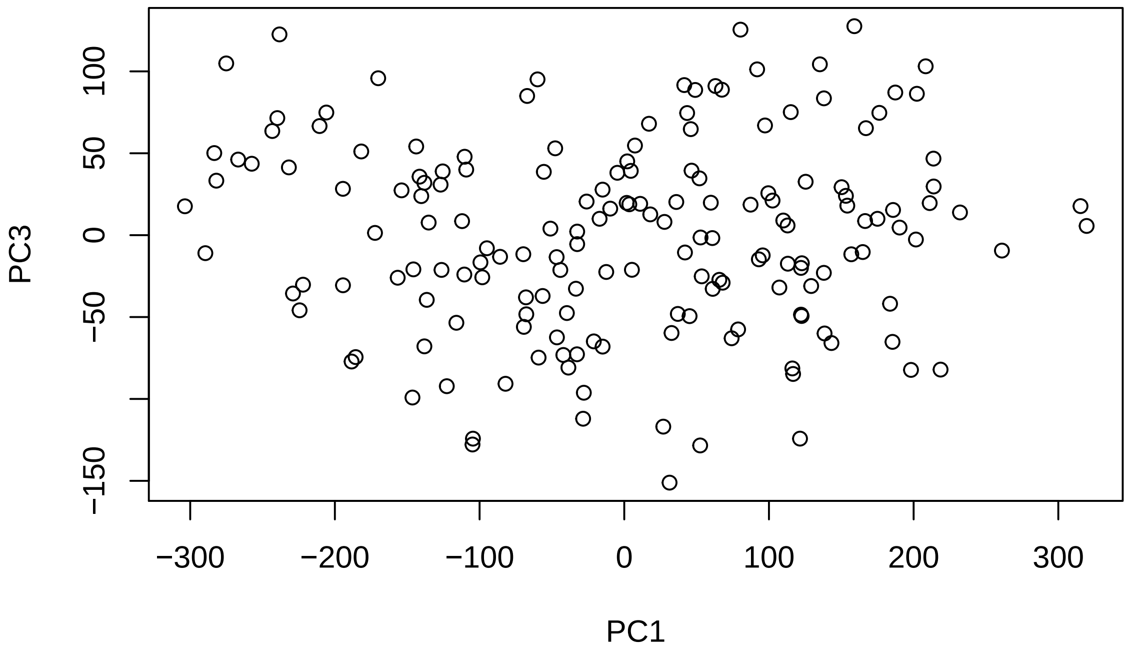
- the data can be centered and or normalized before PCA. Centering is a usual step in PCA and PCA without centering is rarely used. Normalization is used if you analyze apples and pears. They are controlled by: `center`, `scale.` (note: there is a dot after “scale”).
- you can use PC1 and PC2 values using `predict` function, e.g.:

```r
plot(predict(pcamodel))
```



To plot PC1 vs. PC3 use:

```r
plot(predict(pcamodel)[,c(1,3)])
```



0.0.15 Cluster Analysis

We can demonstrate cluster analysis on the same dataset as PCA:

```
tdf <- read.table("https://raw.githubusercontent.com/spiwokv/Rtutorial/master/data/tourdefrance2013.
```

First let us try non-hierarchical clustering by K-means method. For this we have to chose the number of cluster (the value of K). For example we have some feeling that there are 5 clusters of riders such as general classification specialists, sprinters, combined, domestiqeus and those who bet on just one or few stages. This can be done by:

```
kmodel <- kmeans(tdf[,3:22], 5)
kmodel
```

```
## K-means clustering with 5 clusters of sizes 31, 49, 16, 37, 37
##
## Cluster means:
##          E1        E2        E3        E5        E6        E7        E8
## 1 119.22581 140.16129 141.64516 148.22581 145.64516 142.96774 144.77419
## 2 114.73469  78.20408  69.61224 104.85714 106.87755  73.77551  64.59184
## 3  87.50000 155.62500 152.56250  35.62500  39.62500 146.56250 152.75000
## 4  72.02703  48.54054  43.29730  55.10811  45.37838  40.45946  33.37838
## 5 102.32432 111.97297 125.64865 104.70270 116.51351  99.78378 112.70270
##          E9       E10       E11       E12       E13       E14       E15
## 1 130.96774 138.16129 119.32258 133.32258 128.25806 136.90323 136.35484
## 2  75.59184 103.55102  90.67347 104.71429 113.48980 104.97959  63.22449
## 3 146.25000  22.06250 129.31250  53.62500  95.56250  85.75000 146.50000
## 4  40.32432  56.51351  48.45946  54.48649  49.75676  55.40541  36.91892
## 5  95.97297 102.86486  91.70270  80.00000  73.35135  63.97297 110.32432
##         E16       E17       E18       E19       E20       E21
## 1 126.87097  98.77419 127.64516 122.22581 117.58065 109.00000
## 2  66.69388  74.59184  62.02041  58.16327  70.79592  94.24490
## 3 126.56250 119.25000 146.62500 132.18750 142.00000  25.31250
## 4  44.56757  48.43243  41.40541  47.97297  34.16216  65.13514
## 5 109.97297 118.18919 104.16216 108.27027 105.00000 100.62162
##
## Clustering vector:
##   [1] 2 5 1 4 5 4 2 3 2 2 5 2 3 2 4 2 4 2 4 5 1 2 4 1 1 2 2 1 1 1 4 1 4 4 4
##  [36] 3 2 4 2 4 5 4 1 5 5 1 2 1 2 3 1 5 5 2 1 3 1 3 3 2 2 2 5 4 5 1 1 4 4 1
##  [71] 5 2 1 2 4 5 5 2 2 2 5 1 4 2 2 3 5 2 5 1 5 1 5 5 5 2 5 4 2 1 2 5 5 5 2
## [106] 3 5 3 5 5 3 3 2 4 5 4 4 2 4 1 1 4 4 5 5 5 4 4 4 2 2 2 3 5 4 2 3 4 1 4
## [141] 5 3 2 2 4 2 3 5 1 2 1 2 4 2 2 1 1 4 4 2 2 1 2 4 2 5 4 1 5 2
##
## Within cluster sum of squares by cluster:
## [1] 1109263.0 1561950.2  562254.8 1012247.7 1283058.7
##  (between_SS / total_SS =  40.9 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

Each rider was placed into one of 5 clusters and the vector with this results can be printed as:

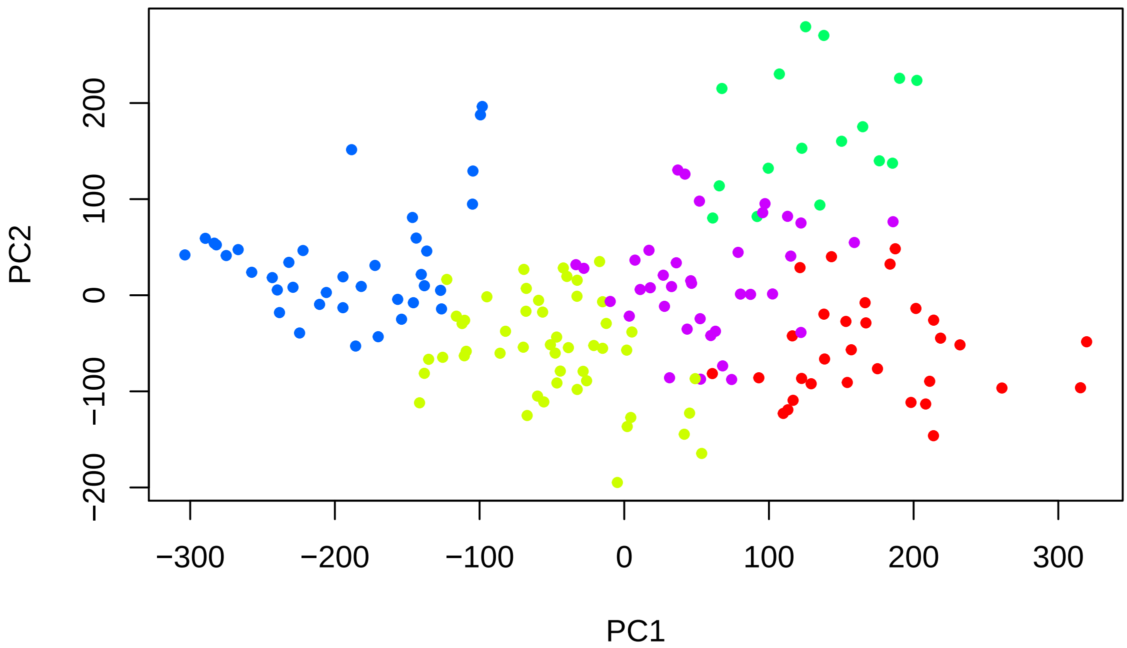
```
kmodel$cluster
```

```
##   [1] 2 5 1 4 5 4 2 3 2 2 5 2 3 2 4 2 4 2 4 5 1 2 4 1 1 2 2 1 1 1 4 1 4 4 4
##  [36] 3 2 4 2 4 5 4 1 5 5 1 2 1 2 3 1 5 5 2 1 3 1 3 3 2 2 2 5 4 5 1 1 4 4 1
##  [71] 5 2 1 2 4 5 5 2 2 2 5 1 4 2 2 3 5 2 5 1 5 1 5 5 5 2 5 4 2 1 2 5 5 5 2
## [106] 3 5 3 5 5 3 3 2 4 5 4 4 2 4 1 1 4 4 5 5 5 4 4 4 2 2 2 3 5 4 2 3 4 1 4
```

```
## [141] 5 3 2 2 4 2 3 5 1 2 1 2 4 2 2 1 1 4 4 2 2 1 2 4 2 5 4 1 5 2
```

We can illustrate the results on the plot from PCA:

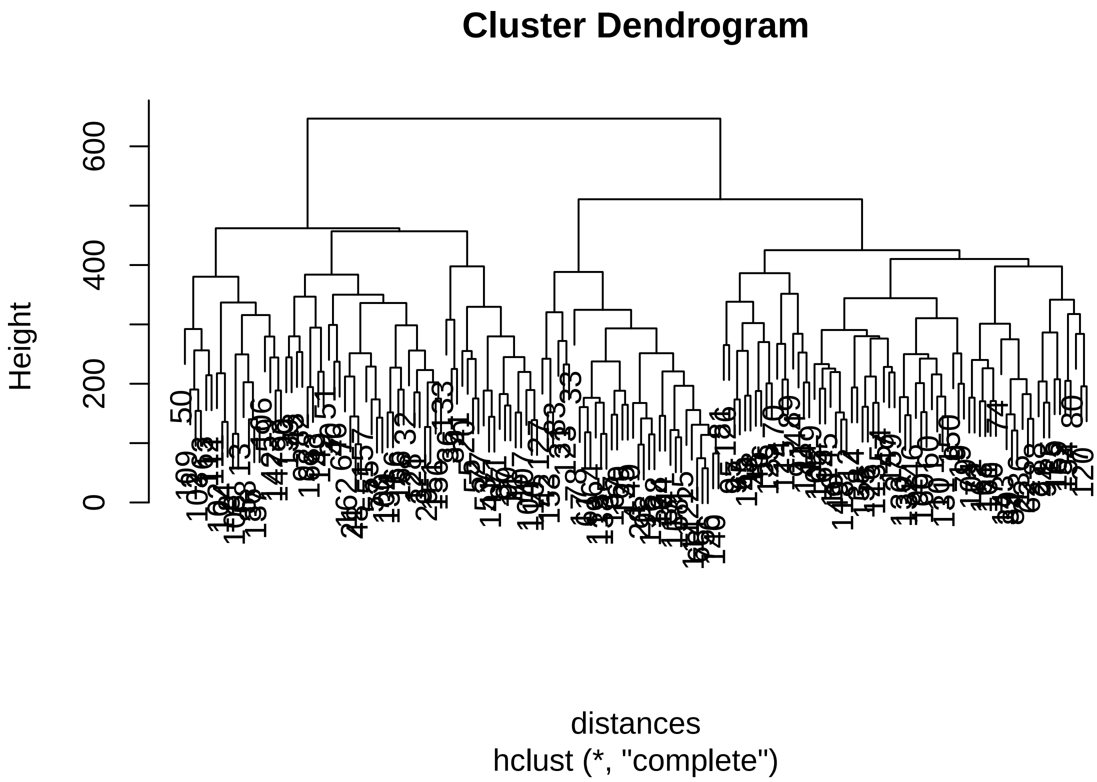
```
pcamodel <- prcomp(tdf[,3:22])
plot(predict(pcamodel), col=rainbow(5)[kmodel$cluster], pch=20)
```



You can see that the distribution into clusters works very well.

Hierarchial clustering can be done by typing:

```
distances <- dist(tdf[,3:22])
hmodel <- hclust(distances)
plot(hmodel)
```



You can again inspect identities of riders to find which branch corresponds to sprinters, which corresponds to general classification specialists etc.

0.0.15.1 Tips and Tricks

- the algorithm used for K-means clustering can be controlled by `algorithm` option.
- the type of distance can be changes by `method` option of `dist` function.
- the parameter `method` controls the method used to construct the tree in `hclust`.

0.0.16 Programming

R is a programming language. You can make loops and other programming constructions in R.

0.0.16.1 Tips and Tricks

- you can make a for-loop by:

```r
for(i in 1:10) {
  print(i)
}
```

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

- always use `print` function to print something in a loop
- you can “if” statement by:

```
for(i in 1:10) {
  if(i==3) {
    print(i)
  }
}
```

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

If-else statement and switch statement works as well.

- you can define a function:

```
sem <- function(x) {
  return(sd(x)/sqrt(length(x)))
}
sem(rnorm(10))
```

```
## [1] 0.3281054
```

- to run a program in R in a command line save the code to “program.R” and run it by typing:

  `Rscript program.R`

  or

  `R --no-save < program.R`

- you can use a kind of object oriented programming in R
- you can use generic function, such as plot, defined to an object you define

0.0.17 Next Steps

0.0.17.1 Tips and Tricks

- you can use any or packages available in CRAN
- you can use tidyverse family of packages for data analysis
- you can use Bioconductor family of packages for data analysis
- you can read R Journal to learn more about news in R community and new R packages
- you can make your own package and submit it to CRAN.
- there are special R user conferences (UseR!, eRum, . . .)
- there are special R user communities such as R ladies