Introduction to R

A. Blejec

andrej.blejec@nib.si

Ljubljana, July 30, 2010

Contents

1 Introduction

R is a comprehensive computer platform and language for modern statistical analysis of data. R supports a wide range of data structures, has excellent graphical capabilities, and a reach collection of statistical methods which are provided in many specialized packages.

R is based on S which is a very efficient high level language and an environment for data analysis and graphics. In 1998, the Association for Computing Machinery (ACM) presented its Software System Award to John M. Chambers, the principal designer of S, for

> the S system, which has forever altered the way people analyze, visualize, and manipulate data ...

A description and some history of R and S is quoted from "R FAQ" (**?**):

> R is a system for statistical computation and graphics. It consists of a language plus a run-time environment with graphics, a debugger, access to certain system functions, and the ability to run programs stored in script files.
>
> The design of R has been heavily influenced by two existing languages: Becker, Chambers & Wilks' S (see What is S?) and Sussman's Scheme. Whereas the resulting language is very similar in appearance to S, the underlying implementation and semantics are derived from Scheme. See "What are the differences between R and S?", for further details.
>
> The core of R is an interpreted computer language which allows branching and looping as well as modular programming using functions. Most of the user-visible functions in R are written in R. It is possible for the user to interface to procedures written in the C, C++, or FORTRAN languages for efficiency. The R distribution contains functionality for a large number of statistical procedures. Among these are: linear and generalized linear models, nonlinear regression models, time series analysis, classical parametric and nonparametric tests, clustering and smoothing. There is also a large set of functions which provide a flexible graphical environment for creating various kinds of data presentations. Additional modules ("add-on packages") are available for a variety of specific purposes (see R Add-On Packages).

R was initially written by Ross Ihaka and Robert Gentleman at the Department of Statistics of the University of Auckland in Auckland, New Zealand. In addition, a large group of individuals has contributed to R by sending code and bug reports.

Since mid-1997 there has been a core group (the “R Core Team”) who can modify the R source code archive. The group currently consists of Doug Bates, John Chambers, Peter Dalgaard, Robert Gentleman, Kurt Hornik, Stefano Iacus, Ross Ihaka, Friedrich Leisch, Thomas Lumley, Martin Maechler, Duncan Murdoch, Paul Murrell, Martyn Plummer, Brian Ripley, Duncan Temple Lang, Luke Tierney, and Simon Urbanek.

R has a home page at `http://www.R-project.org/`. It is free software distributed under a GNU-style copyleft, and an official part of the GNU project (“GNU S”).

Typesetting conventions

In this document, text is rendered as:

R commands in text: `mean(x)`
R commands in examples:

```
> mean(1:6)
```

Results (typed in R):

```
3.5
```

Menu options: **Edit**
Package names: **base**
File names: data.txt
Links (URL and file): `http://www.r-project.org`

2 Demonstration of R for data analysis

To show the flavor of R data analysis, we will analyze a small dataset of people's height and weight. People try to care about their body weight. It is common knowledge, that weight is increasing with height. To compensate the influence of height on weight, Body Mass Index (BMI) was invented and can be calculated as:

$$BMI = \frac{weight}{height^2}$$

where weight is measured in kilograms and height is measured in meters.

2.1 Reading data

Data for males and females were gathered and are listed in the file bmiall.txt in the simple tabular form: one case per line with tabulator separated values. For reading the Excel files, package **xlsReadWrite** should be installed (see **Packages | Install packages** and loaded (see **Packages | Load Package**).

First, we will read the data from file, list the first few lines and check the structure of data. Number of cases will be stored in variable `n`.

```
> bmiData <- read.table("../dat/bmiall.txt", header = TRUE)
> head(bmiData)

  gender age weight height
1      M  17   73.6  1.730
2      M  17   71.0  1.765
3      M  17   62.4  1.770
4      M  17   71.0  1.870
5      M  17   72.4  1.765
6      M  17  104.0  1.825

> str(bmiData)

'data.frame':        419 obs. of  4 variables:
 $ gender: Factor w/ 2 levels "F","M": 2 2 2 2 2 2 2 2 2 2 ...
 $ age   : int  17 17 17 17 17 17 17 17 17 17 ...
 $ weight: num  73.6 71 62.4 71 72.4 104 70.4 79.8 63.4 75.8 ...
 $ height: num  1.73 1.76 1.77 1.87 1.76 ...

> dim(bmiData)

[1] 419   4

> n <- dim(bmiData)[1]
```

Variables have different types, `gender` is descriptive, the rest are numeric. R adapts the default (and possible) operations according to variable types:

```
> summary(bmiData)
```

```
 gender        age              weight           height     
 F:205   Min.   :17.00   Min.   : 44.80   Min.   :1.502  
 M:214   1st Qu.:17.00   1st Qu.: 57.20   1st Qu.:1.652  
         Median :17.00   Median : 63.20   Median :1.720  
         Mean   :17.49   Mean   : 64.59   Mean   :1.720  
         3rd Qu.:18.00   3rd Qu.: 71.00   3rd Qu.:1.780  
         Max.   :18.00   Max.   :104.00   Max.   :1.970  
```

2.2 Gender and age

We would like to see the gender and age structure Attaching the `bmiData` means, that we want to use variables from `bmiData`. This is like unwrapping the data set for use. With `detach()`, the last attached data set is wrapped back, the variables are not recognized anymore.

```
> attach(bmiData)
> table(gender)
```

```
gender
  F   M 
205 214 
```

```
> table(gender, age)
```

```
      age
gender  17  18
     F 101 104
     M 112 102
```

Note: you can wrap back the dataset by `detach()`. Variables in a data set will be hidden to the system after `detach`.

Using the `prob.table` function, ne can get total, row, and column percentages:

```
> X <- table(gender, age)
> totP <- prop.table(X)
> round(totP * 100, 1)
```

```
      age
gender   17   18
     F 24.1 24.8
     M 26.7 24.3
```

```
> rowP <- prop.table(X, 1)
> round(rowP * 100, 1)
```

```
      age
gender   17   18
     F 49.3 50.7
     M 52.3 47.7
```

```
> colP <- prop.table(X, 2)
> round(colP * 100, 1)
```

```
      age
gender   17   18
     F 47.4 50.5
     M 52.6 49.5
```

Marginal distributions can be set using the `margin.table`:

Total number of cases

```
> margin.table(X)
```

```
[1] 419
```

and margins first by rows and then by columns

```
> margin.table(X, 1)
```

```
gender
  F   M
205 214
```

```
> margin.table(X, 2)
```

```
age
 17  18
213 206
```

Margins can be added to to the table

```
> Xm <- addmargins(X)
> Xm
```

```
      age
gender  17  18 Sum
   F   101 104 205
   M   112 102 214
   Sum 213 206 419
```

Although one can get marginal distributions with a function `margin.table`, we will do it to demonstrate the powerful function `apply`. For brevity, we will define two constants `byRow = 1` and `byColumn = 2`. They are saying, which index in a table to use for calculation of marginal sums.

```
> byRow <- 1
> byColumn <- 2
> (Rsum <- apply(X, byRow, sum))
```

```
  F   M
205 214
```

```
> (Csum <- apply(X, byColumn, sum))
```

```
 17  18
213 206
```

```
> X/Rsum
```

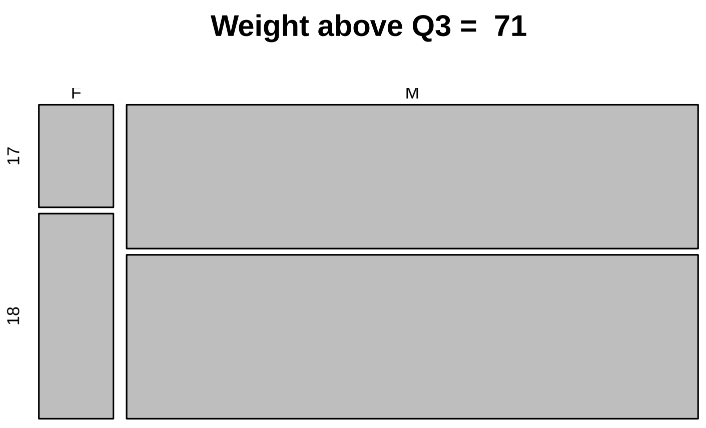
```
      age
gender        17        18
     F 0.4926829 0.5073171
     M 0.5233645 0.4766355
```

The first `apply` calculated the `sum` for each row, and the second one calculated the sum for each column. Don't be confused with a strange-looking details in the code, try to grasp the general idea. `apply` is powerful way to say that some function has to be applied to columns (2 in a call) or rows (1 in a call) of a table or matrix.

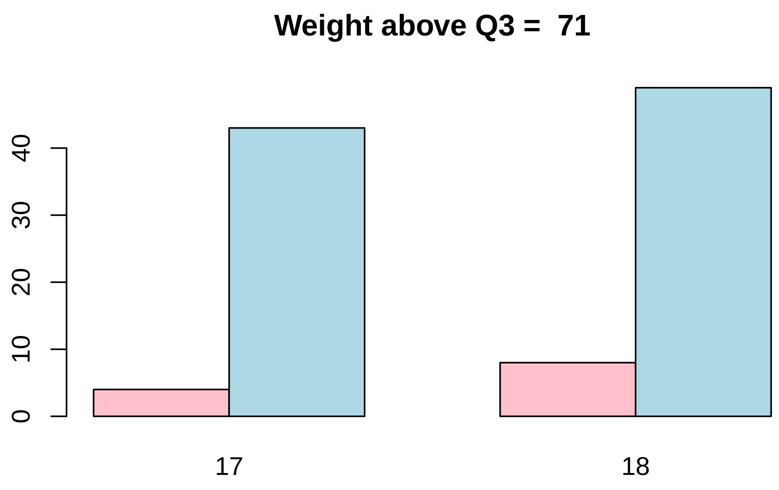
R is pretty smart in plotting. Objects carry *class* information, which helps R to decide how to plot the data. Beside the default plot, any other kind of plotting can be prepared. Let us graphically present some tables.

To do this, let us look at the distribution in the heavier part of population *i. e.* above the third quartile. First we calculate the third quartile and then prepare the selector variable. Then we use it to select cases, tabulate and plot.

```
> Q3 <- quantile(weight, 0.75)
> select <- (weight > Q3)
> X <- table(gender[select], age[select])
> print(X)

    17 18
  F  4  8
  M 43 49

> heading <- paste("Weight above Q3 = ", Q3)
> plot(X, main = heading)
```



```
> barplot(X, beside = TRUE, col = c("pink", "lightblue"), main = heading)
```



Both plots, the first one is known as *mosaic* plot, show that we have more males in the heavy part.

2.3 Height and weight

Variables `weight` and `height` are numeric, which means that one can calculate various summary statistics:

```
> mean(weight)
[1] 64.5883
> mean(height)
[1] 1.719964
> sd(weight)
[1] 10.53051
> sd(height)
[1] 0.08752747
> (V <- var(cbind(weight, height)))
             weight      height
weight 110.8916572 0.601565848
height   0.6015658 0.007661059
> cor(weight, height)
[1] 0.6526635
> my.cor <- V[1, 2]/(sd(weight) * sd(height))
> cat("Correlation r =", my.cor, "\n")
Correlation r = 0.6526635
```

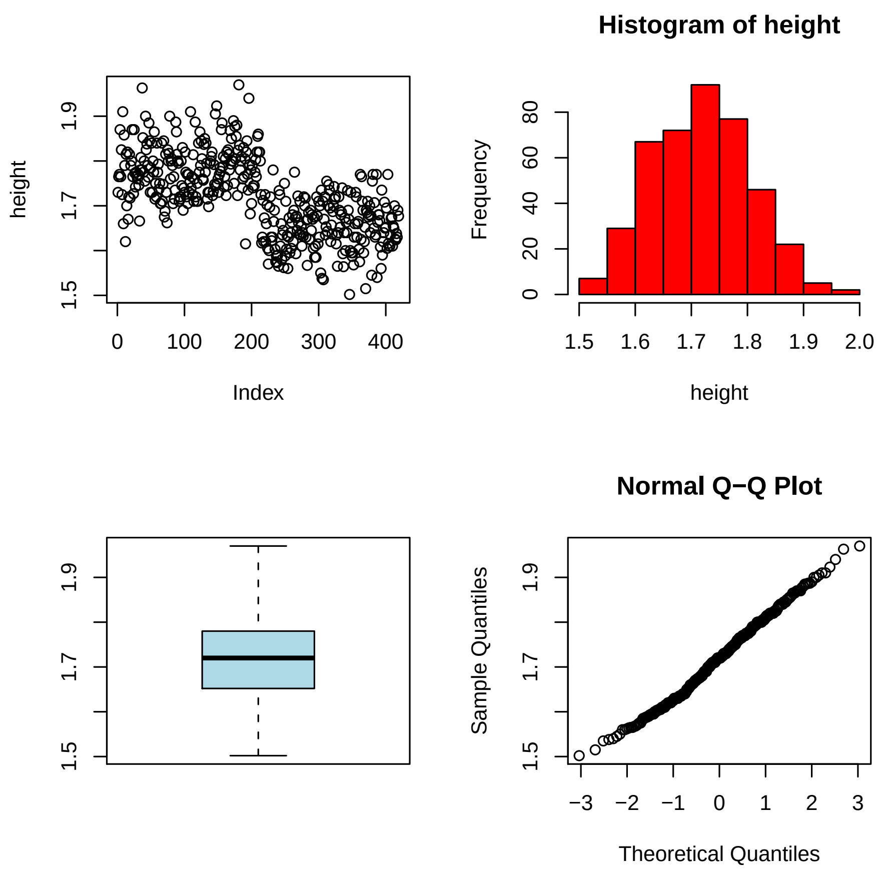
The last line shows one of the strengths of R : intermediate results are available for further calculation and printing! Correlation was calculated by taking the covariance `V[1,2]` from the covariance matrix `V` and dividing by standard deviations calculated by `sd` function. The `cat` concatenates and types the arguments; the argument `"\n"` instructs it to go to next line.

Are there differences in weight and height in gender age classes?

```
> aggregate(cbind(weight, height), list(age, gender), mean)
  Group.1 Group.2   weight   height
1      17       F 58.51881 1.650644
2      18       F 59.42500 1.656644
3      17       M 69.12500 1.775857
4      18       M 70.88137 1.791794
```

Let us do some plotting:

```
> oldpar <- par(mfrow = c(2, 2))
> plot(height)
> hist(height, col = 2)
> boxplot(height, col = "lightblue")
> qqnorm(height)
> par(oldpar)
```

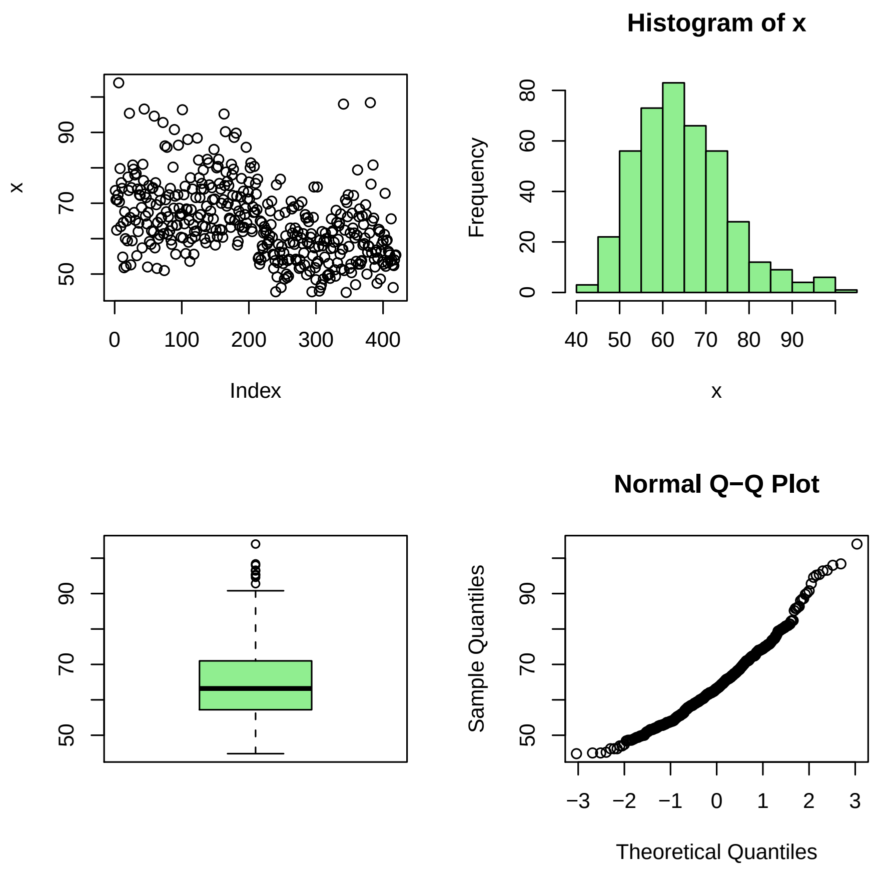


Such plot with four combined graphs (2 × 2) on the same panel that present the same data in different ways might be useful for future use. We can change it to a function, which can be called at any later time, by adding two lines, saying that a function with a name `gtour` is defined as our list of commands within the curly braces. (For brevity, we changed ”`height`” to ”`x`” and made color selectable; compare both command sets)

```
> gtour <- function(x, color = "lightblue") {
+     oldpar <- par(mfrow = c(2, 2))
+     plot(x)
+     hist(x, col = color)
+     boxplot(x, col = color)
+     qqnorm(x)
+     par(oldpar)
+ }
```

Now we can plot `weight` by calling the function `gtour`:

```
> gtour(weight, color = "lightgreen")
```

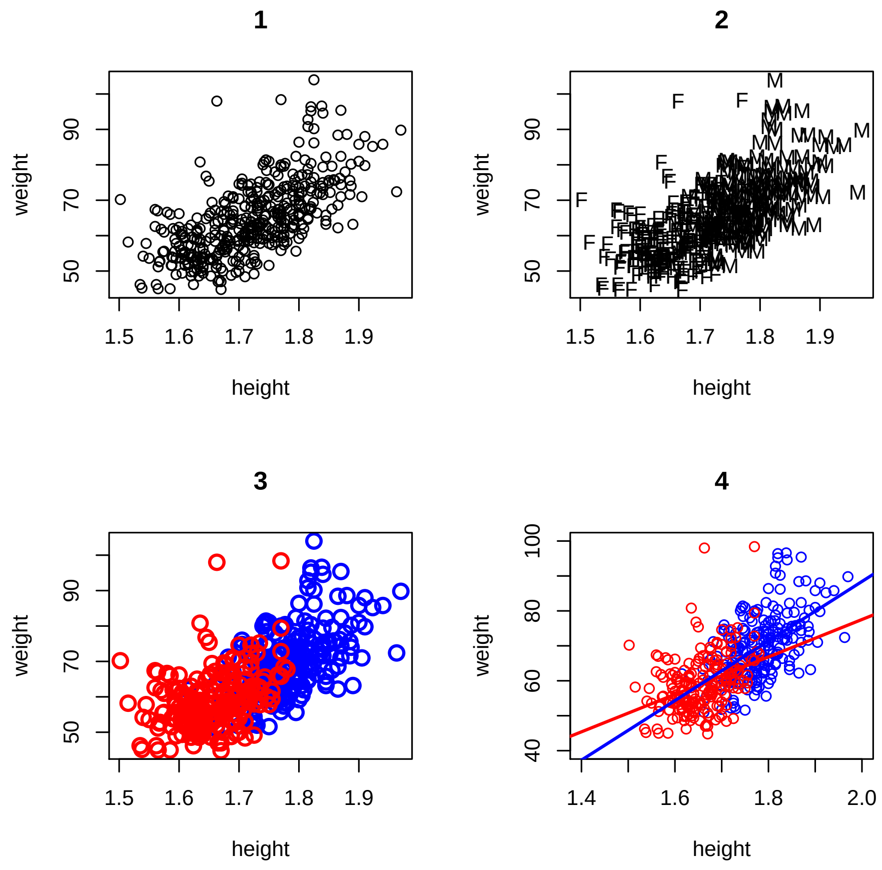


We have seen, that correlation between `height` and `weight` is pretty strong (r=0.65) and should be visible in the scatterplot. Four graphs in the panel show different refinements of the scatterplot.
First we plot the raw graph (1).
Since we assume that there are differences in height and weight of females and males, we want to mark the points according to gender (2).
There is considerable overlap and the graph is pretty ugly. Colors are usually better. Color is selected with a rather arcane (yet effective) line of code. we also try to make points bigger and lines wider (3).
For the last graph, points size is set back to default, but axes limits are controlled (4).

In the next step, linear regression lines are added to the plot.
for each gender:

- first select the data (`...[select]`),
- fit a linear model (`lm`),
- add the regression line to the plot (`abline`) and
- print the regression line coefficients (`print`

```
> oldpar = par(mfrow = c(2, 2))
> plot(height, weight)
> title(1)
> plot(height, weight, pch = as.character(gender))
> title(2)
> col <- c("red", "blue")[as.numeric(factor(gender))]
> plot(height, weight, col = col, cex = 1.5, lwd = 2)
> title(3)
> plot(height, weight, col = col, xlim = c(1.4, 2), ylim = c(40,
+     100))
> title(4)
> for (g in c("F", "M")) {
+     select <- (gender == g)
+     x <- height[select]
+     y <- weight[select]
+     rfit <- lm(y ~ x)
+     col <- c("red", "blue")[(g == "M") + 1]
+     abline(rfit, col = col, lwd = 2)
+     cat("Gender:", g, "\n")
+     print(rfit)
+ }
Gender: F

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
     -29.63        53.58

Gender: M

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
     -81.98        85.20
```



2.4 Inference

Are there any effects of gender and age on weight and height? We can test this by analysis of variance:

```
> summary(aov(height ~ gender + age))
             Df  Sum Sq Mean Sq  F value  Pr(>F)    
gender        1 1.76308 1.76308 514.1828 < 2e-16 ***
age           1 0.01282 0.01282   3.7395 0.05382 .  
Residuals   416 1.42642 0.00343                     
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
> summary(aov(weight ~ gender + age))
             Df Sum Sq Mean Sq  F value Pr(>F)    
gender        1  12631 12631.2 156.6954 <2e-16 ***
age           1    188   187.9   2.3305 0.1276    
Residuals   416  33534    80.6                    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Or, since there is no influence of age, by *t-test*:

```
> t.test(height ~ gender)

        Welch Two Sample t-test

data:  height by gender
t = -22.6415, df = 416.359, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.1410314 -0.1184996
sample estimates:
mean in group F mean in group M
       1.653688        1.783453
```

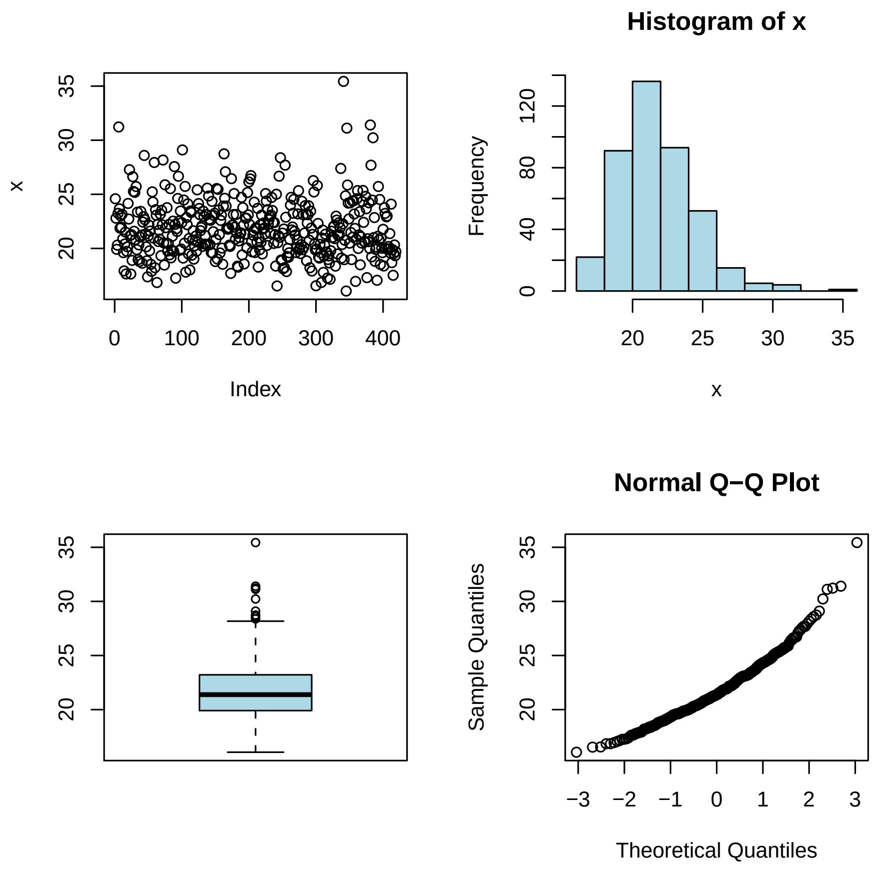
Sometimes functions return useful information, that can be used for further calculations or used for informative comments:

```
> (tweight <- t.test(weight ~ gender))

        Welch Two Sample t-test

data:  weight by gender
t = -12.5432, df = 410.525, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -12.70496  -9.26227
sample estimates:
mean in group F mean in group M
       58.97854        69.96215

> names(tweight)

[1] "statistic"   "parameter"   "p.value"     "conf.int"    "estimate"
[6] "null.value"  "alternative" "method"      "data.name"

> tweight$estimate

mean in group F mean in group M
       58.97854        69.96215

> cat("My comment about p-value (p =", tweight$p.value, ")\n")

My comment about p-value (p = 8.906156e-31 )
```

2.5 What about the BMI?

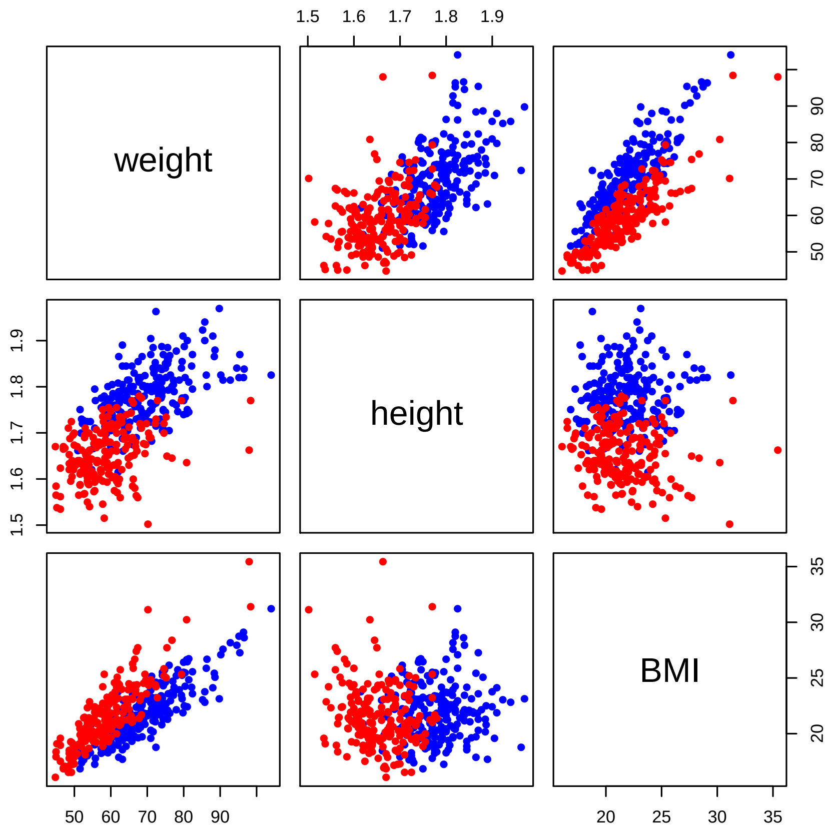
First, we calculate a new variable `BMI` and plot a `gtour`:

```
> BMI <- weight/(height^2)
> gtour(BMI, col = "lightblue")
```



Make a new dataset by combining columns for weight, height and BMI in table (data frame) and plot scatterplots for all possible pairs and calculate the correlation matrix:

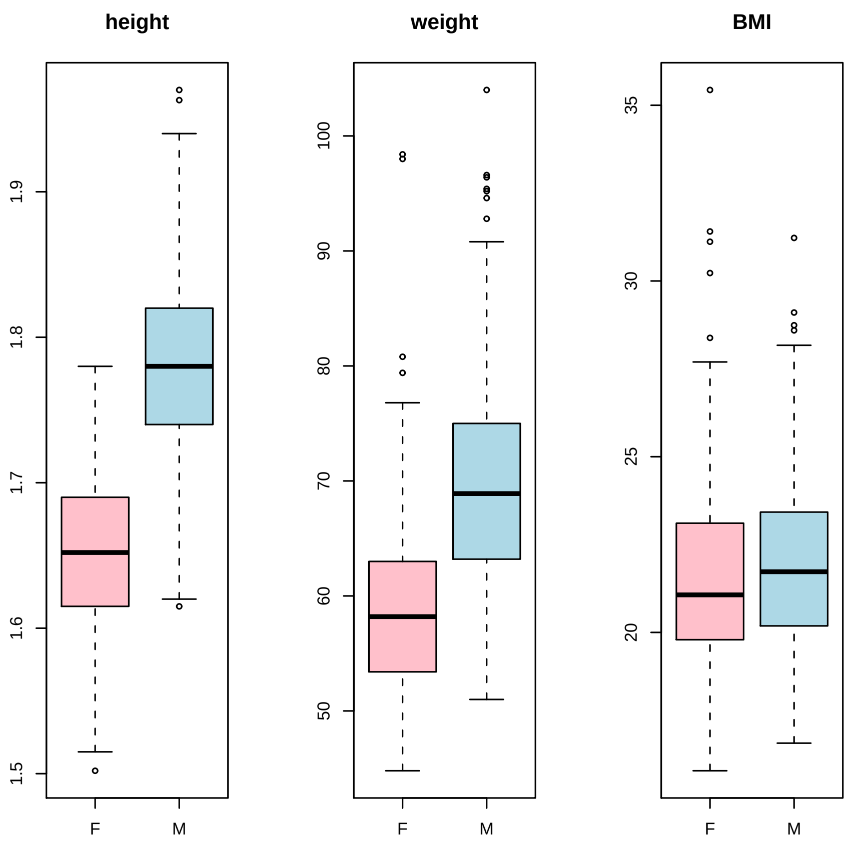
```
> X <- data.frame(weight, height, BMI)
> col <- c("red", "blue")[as.numeric(factor(gender))]
> pairs(X, col = col, pch = 16)
> cor(X)
          weight     height        BMI
weight 1.0000000 0.65266353 0.76766658
height 0.6526635 1.00000000 0.02193029
BMI    0.7676666 0.02193029 1.00000000
```



While weight is heavily influenced by height, BMI is almost independent of height and correlated with weight. In our case, it is also almost independent of gender and is a good general indicator of body massiveness.

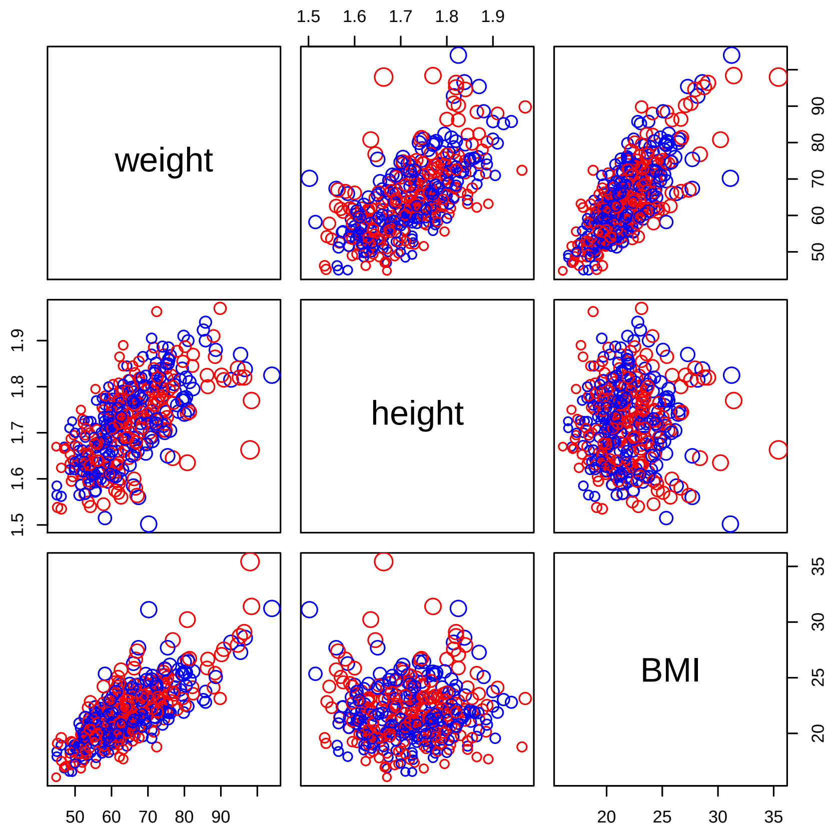
Let us show the data in some other ways. First the boxplots by gender.

```
> oldpar <- par(mfrow = c(1, 3))
> cols <- c("pink", "lightblue")
> boxplot(split(height, gender), col = cols, main = "height")
> boxplot(split(weight, gender), col = cols, main = "weight")
> boxplot(split(BMI, gender), col = cols, main = "BMI")
> par(oldpar)
```



One can control the plotted symbols, their size can be calculated according to some variable, in our case the BMI. (You can try other divisors beside 15)

```
> cols <- c("red", "blue")
> pairs(X, cex = BMI/15, col = cols)
```

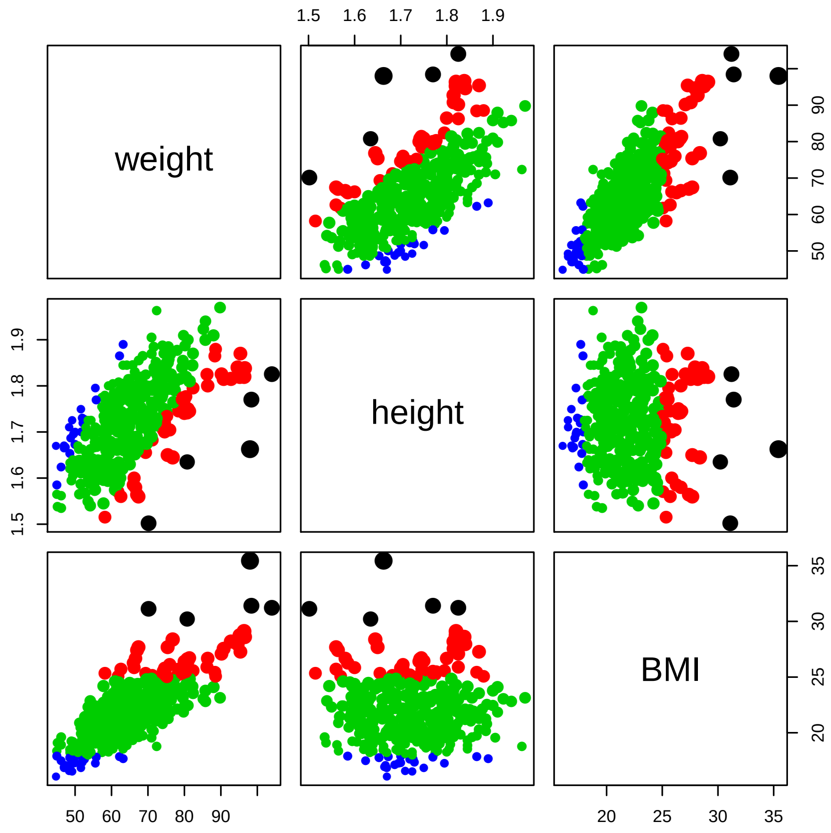


According to the health tables, BMI classes are defined. We will cut the `BMI` into classes, rename the classes and order the levels of the factor `bmic`. Then we will plot print and plot the table of coded BMI values.

```
> bmic <- cut(BMI, c(0, 13, 18, 25, 30, Inf))
> levels(bmic)
[1] "(0,13]"   "(13,18]"  "(18,25]"  "(25,30]"  "(30,Inf]"
> levels(bmic) <- c("S", "s", "N", "h", "H")
> bmic <- factor(bmic, levels = c("S", "s", "N", "h", "H"), ordered = T)
> is.ordered(bmic)
[1] TRUE
```

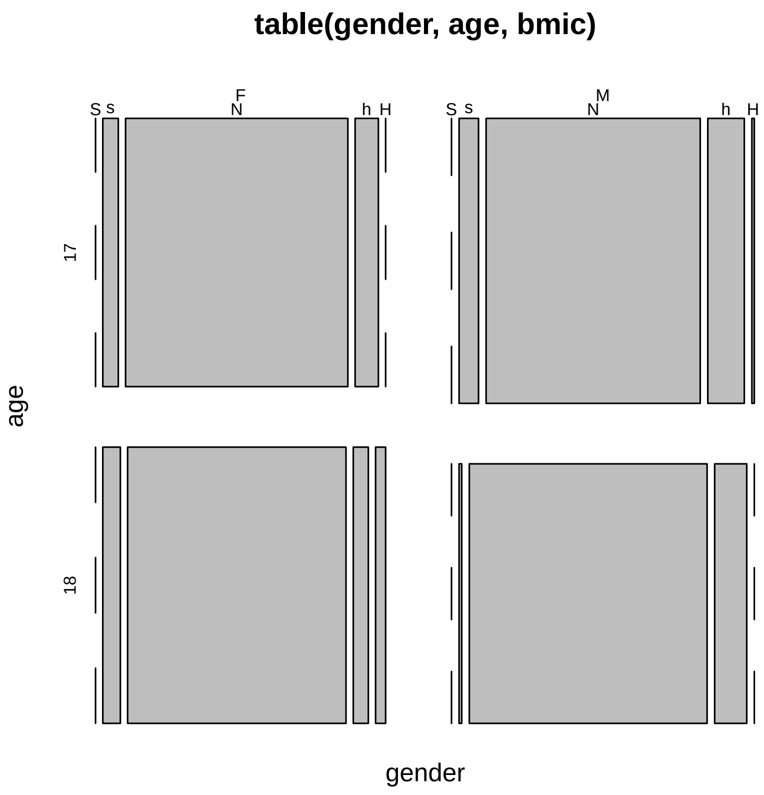
Plotting the scatterplot with coded BMI as color and size of points, we can see that BMI is a sort of principal component. The levels of BMI are running parallel to the main variation line.

```
> cols <- (6 - as.numeric(bmic))
> pairs(X, cex = BMI/15, col = cols, pch = 16)
```



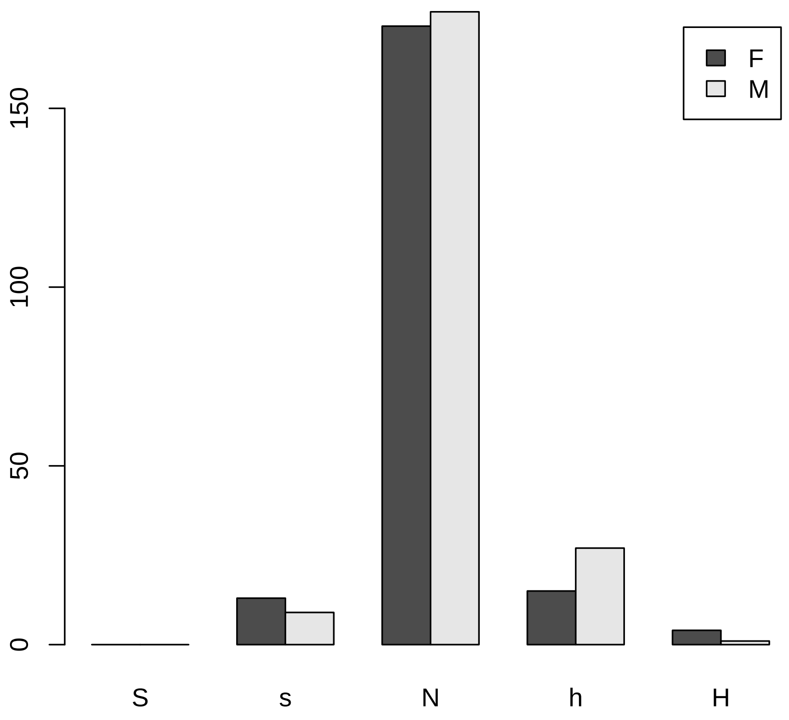
Flat contingency table shows the the structure of BMI levels by gender and age:

```
> gabTable <- ftable(gender, age, bmic)
> gabTable
           bmic  S  s  N  h  H
gender age
F      17        0  6 86  9  0
       18        0  7 87  6  4
M      17        0  8 88 15  1
       18        0  1 89 12  0
> plot(table(gender, age, bmic))
```



A more common plot of BMI by gender would be

```
> barplot(table(gender, bmic), beside = TRUE, legend = TRUE)
```


F
M
0
50
100
150
S
s
N
h
H

3 How to start

3.1 Installation

Installation of R for Windows users is pretty straightforward. Download the latest setup file (like R-2.7.1-win32.exe for version 2.7.1) from the `base` directory on the nearest `CRAN` site `http://www.r-project.org`, click on its icon to start installation, and follow instructions. Packages that do not come with the base distribution must be downloaded and installed separately. For more details. see the README file that is included with the R distribution.

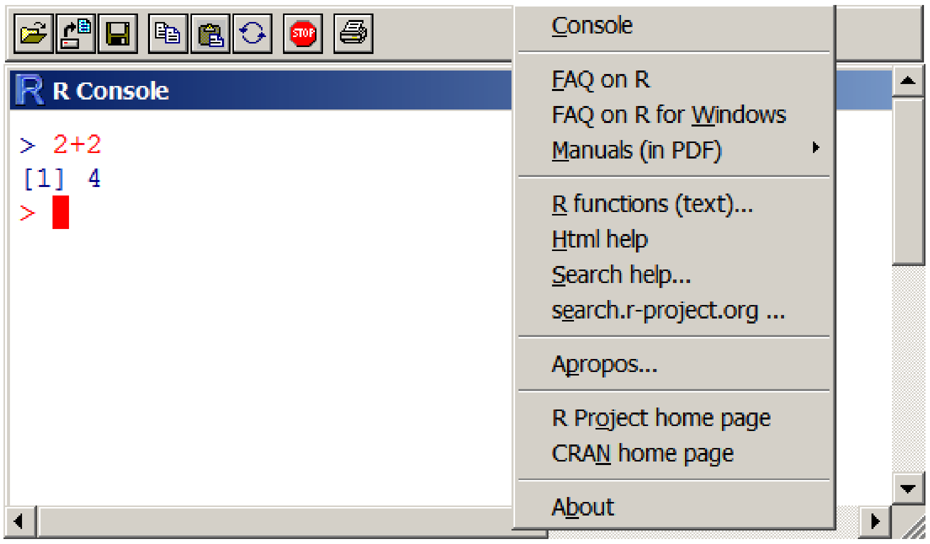
3.2 Invoking R

R can be started in any of the usual ways for starting programs under Windows. When R is started, it opens a canvas (named RGui) which will hold the command window (R console) and graphics windows (R graphics). Initially it opens the command window, graphics windows are opened on demand.

Figure 3.1 shows the R canvas with the command window. A simple command (2+2) giving the result 4 was given to R . Commands are preceded with a command prompt character > and typed in red. Commands can span more than one line, in which case command continuation lines are marked with + . Several commands, separated by a semicolon (`;`), can be typed in one line. Results are printed in blue. Some results are preceded with a number in square brackets (in our case `[1]`), which shows the position of the next value in a list. This will be more obvious in longer prints. Hash (#) is a comment character: anything after the # is a comment and is ignored by parser.

Commands can be literally typed in or pasted from clipboard. The **Copy-Paste** technique is very productive and can be used in connection with any text editor or word processing software. Certainly some editors are more suited for R than others (*e. g.* `Rwinedt`).

A very effective menu option **Edit | Paste commands only** (only available on Windows platform) is able to filter out any commands that are copied to clipboard (and mixed with any other text) and paste just commands (lines preceded with > and +) to the command window. One can select any portion of text in a .pdf file and paste it to R . For example, you can copy lines from this paragraph to the end of the next one, and use **Paste commands only** to get the result shown in the example.



```
> 2+2
[1] 4
>
```

Figure 3.1: R canvas

```
> 2 + 2
[1] 4
```

Commands (typed or pasted) are immediately interpreted and executed, showing the results as text or graphics. The immediate execution and display of results are a feature of so called ”interactive mode”. [1]

Results can be saved for later use. One of two *assign* operators, `<-` and `=`, can be used to name (save) results for later use:

```
> x <- 2 + 2
> x
[1] 4
> x + 3
[1] 7
```

The result of the operation in the first line is saved as an ”object” named `x`. Typing just the object name reveals it’s content. When object name is used in the commands, it’s value is used in calculation. In R everything you can type or plot is an object. You can get a list of objects created so far by typing `ls()` or `objects()`:

```
> ls()
 [1] "catln"          "Data"            "dstats"        "format.xtab"
 [5] "my.latex"       "my.prn"          "my.sd"         "my.var"
 [9] "pause"          "pcoord"          "peaks"         "purge"
[13] "read.clipboard" "resetWorkspace"  "savePdf"       "savePDF"
[17] "savePlot"       "saveWMF"         "sData"         "show.colors"
[21] "show.lty"       "show.pch"        "testing"       "write.xls"
[25] "x"              "xtab"
```

Among some predefined objects, there should be also the one named `x`.

[1] R can be used also in so called ”batch mode”

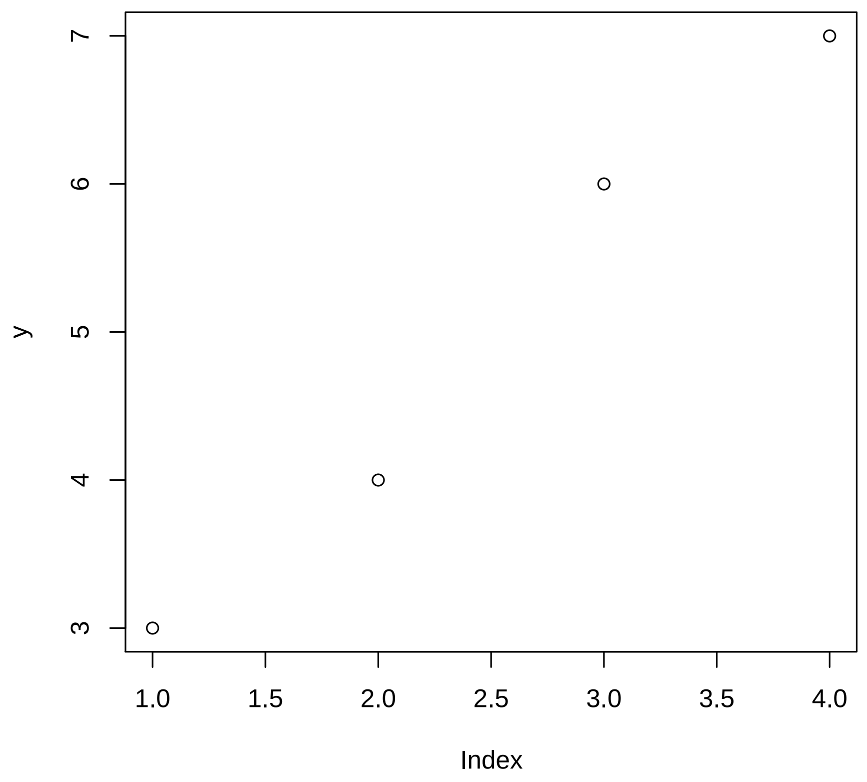
The `ls()` is an example of a *function* call, a name followed by a pair of parentheses. Typing just a `ls`, without the parenthesis pair, would list the content of the object named `ls`, which is not what we want at the moment. Yet, this shows that one can inspect, modify, and use any part of the objects in R .

The parenthesis pair tells R to call a function, with *arguments* that can be optionally typed within a parenthesis of a function call. For example, one can create a series of values using function `c` and calculate their mean value by a call to the function `mean`:

```
> y <- c(3, 4, 6, 7)
> y
[1] 3 4 6 7
> mean(y)
[1] 5
```

Plotting is performed with a call to the *plotting* functions, the minimal one being the `plot` function which opens a R graphics window with a shown graph:

```
> plot(y)
```



which opens a R graphics window with a shown plot.

3.3 Workspace and History

All user created objects are stored in a memory space called *workspace*. All typed commands are saved into the *history*. You can view the history with a call `history(100)`, which will show you the last 100 lines in a separate browser window. Workspace (and history) can be saved or loaded at any time by the use of menu commands **File | Save workspace** and **File | Load workspace** (**File | Save history** and **File | Load history** respectively). The images of the workspace and history are saved to disc when R session is closed. Saved workspace and history are loaded when you run R next time.

3.4 Working directory

It pays to create separate directory for every project and instruct R to use this directory as a *working directory*. The working directory is then a default directory for all file manipulations, including saving and loading of the workspace, history or any other files (*e. g.* data files). By default, R will start deep in the directory tree. You can check the working directory by `getwd()` and set a new one by `setwd()` or using the menu option (**File | Change dir ...**).

You can also exit R (**File | Exit** or type `q()`), save the workspace and history to the project directory and restart R by double-clicking the icon for `.Rimage` in the project directory. In this way, the previously saved workspace and history will be reloaded and working directory will be set to the project directory.

3.5 Packages

R is based on a number of *packages*. A package is basically a collection of functions and other objects, accompanied with the standardized description and some user manual (also called *vignette*). A collection of basic packages are loaded when R is invoked. Additional packages have to be first installed on your computer (**Packages | Install packages ...**) and then loaded to the workspace (**Packages | Load package ...**). The collection of packages is growing so it is worth checking it from time to time for the news. Packages are available `Package` section on CRAN web sites. Among many specialized packages, package **hmisc** (by Frank Harrell, miscellaneous functions) and **MASS** (Venables and Ripley, Modern Applied Statistics with S-PLUS) are pretty useful.

3.6 Getting help

R provides many ways of getting help. Every function in supported packages has a description page. This pages list function description, usage, arguments, returned value, references, links to associated functions, and examples. To get the help on some function, *e. g.* `mean`, one can type command `help(mean)` or simply `?mean` .

In addition, **Help** (Figure 3.2) provides a series of help resources. First is a help on *Console* (command window) shortcuts, like `UpArrow` $\boxed{\uparrow}$ and `DownArrow`$\boxed{\downarrow}$, that can be used to scroll through previously typed commands and $\boxed{\leftarrow}$ and $\boxed{\rightarrow}$, that can

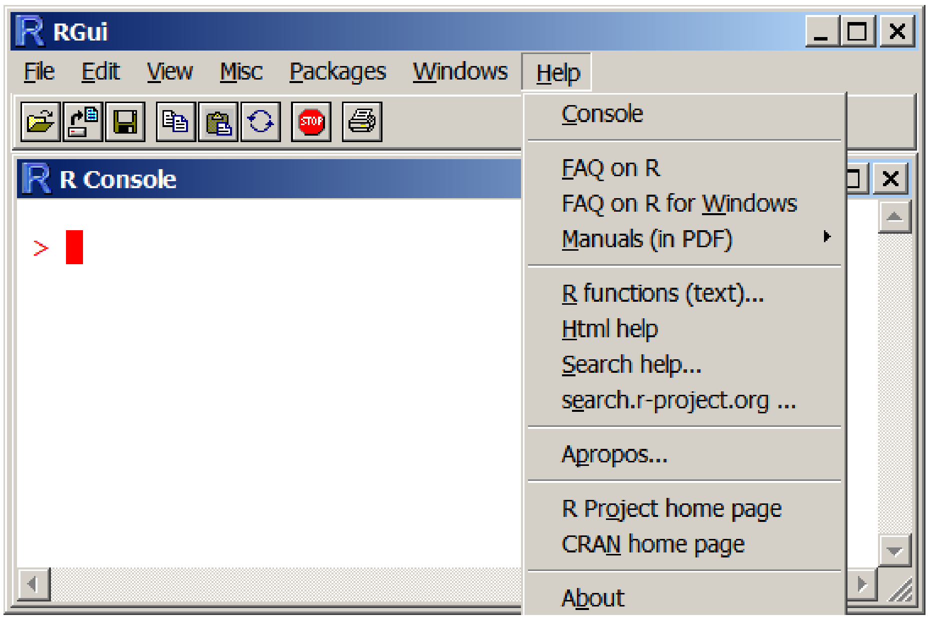


Figure 3.2: R **Help** menu

be used for correcting mistyped commands. A useful one might be `CTRL-X`, which has different meaning as usual: selected text in the console is pasted and eventually executed with a single keystroke. Tabulator key tab provides limited code completion of what you type and can be useful if you know just a beginning part of a function. If nothing happens, press the key twice to get a list of possible continuations.

Next you have a collection of **FAQ** and **Manuals** and entry to the individual functions help. **Help** | **R functions (text)...** is equivalent to function `help` while **Help** | **Search help ...** can be called via `help.search`:

```
> if (interactive()) {
+     help(mean)
+     help.search("mean")
+ }
```

Html help is an entry to different local manual, a search engine and package descriptions and documentation.
search R-project.org is a link to the R Site Search.
Apropos (and function `apropos("string")`) lists all object names that include the "string".

```
> apropos("mean")
 [1] "colMeans"        "kmeans"          "mean"            "mean.data.frame"
 [5] "mean.Date"       "mean.default"    "mean.difftime"   "mean.POSIXct"
 [9] "mean.POSIXlt"    "rowMeans"        "weighted.mean"
```

With function `example` you can access Examples part of some functions like:

```
> example(mean)
mean> x <- c(0:10, 50)

mean> xm <- mean(x)

mean> c(xm, mean(x, trim = 0.10))
[1] 8.75 5.50

mean> mean(USArrests, trim = 0.2)
  Murder  Assault UrbanPop     Rape
    7.42   167.60    66.20    20.16
```

Next we have links to R -project and CRAN home pages.In the `search` directory of R -project page you can find additional search tools on R-project page, like the very efficient `Rseek` http://www.rseek.org/.

In addition any web search engine like *Google* or *Copernicus* can be used to find bits and pieces of R system - just include "R-project" as one of the keywords.

3.7 ... and how to stop

To stop R , use the **File | Exit** or type `q()`. Unless you saved your work in some other way (*e. g.* as commands in ASCII form), don't forget to confirm saving of the workspace image. Next time you can double click on the saved .Rimage file and proceed where you stopped.

3.8 About the editors

For small analysis or experimenting with the code, typing R commands directly to the console is very efficient, since you get immediate response to your commands. However, it is rather easy to lose track and, what is more important, your work is difficulty reproducible. To have the possibility to repeat the analysis, it is advisable to type and edit commands in some text editor and save them for the later use. Editing is much easier in editors with parenthesis check and syntax highlighting. Possibility to copy the code from the text editor and paste it to the R console with one keystroke is also desirable. Hints for good editors can be found on R -project home page. R script editor (**File | New script**) has a shortcut `CTRL-R` to transfer line or selected area from the script window to R or execution, but no syntax highliting or parentheses checking or find and replace. Standard Windows Notepad is good enough, but not perfect. You have to use usual Copy-Paste technique (`CTRL-C`, `CTRL-V`) to transfer code to R console.

4 R objects and data structures

R can be used at different levels. First of all, it can be used as a calculator. It is not to difficult to calculate your BMI :)

In data analysis, data are usually collected into tables. In data tables, columns are usually data series, or variables and rows represent individual units on which measurements were made. In actual calculations, data series and some results are organized in vectors and matrices.

R supports organization of data in a form of *data.frame* and one can organize and manipulate data as *vectors* and *matrices*. Vector (1 dimension) and matrix (2 dimensions) are extended by additional dimensions into *arrays*. The richest structure in R is a *list*, which can hold a number of elements of any of above *classes*.

4.1 Data structures

4.1.1 Vector

R operates on named *data structures*. The simplest data structure is a *vector* which is a collection of single entities like numbers. Five numbers 5, 10, 4, 9 and 1 can be organized as a vector and *assigned* to a named object:

```
> x <- c(5, 10, 4, 9, 1)
> x
[1]  5 10  4  9  1
```

We used function `c()` which can take any number of arguments and combines them into a single entity. Assignment operator `'<-'` can be replaced with an equal sign `'='`. Assignment can also be made by a function `assign`, which will produce a vector named `y`:

```
> assign("y", c(2, 4, 5, -1, 7))
> y
[1]  2  4  5 -1  7
```

If the statement consist of just the name of an *object*, its content is printed, like `y` above. If the statement is an expression, its value is printed and lost, unless it is assigned to some name.

```
> 1/x
[1] 0.2000000 0.1000000 0.2500000 0.1111111 1.0000000
```

Vectors have one dimension, called `length`

```
> length(x)
[1] 5
```

4.1.2 Vector arithmetic

One of the R strengths is a vector arithmetic. Arithmetic operations are performed *component*-wise, which means that in a command

```
> z <- 2 * x + y - 5
> z
[1]  7 19  8 12  4
```

the first component of `z` is the sum of the first component of `x` multiplied by 2 and the first component of `y` decreased by 5. Then the second ones are used in calculation, and so forth:

$$z_i = 2x_i + y_i - 5, \qquad i = 1, \ldots, 5$$

.

The arithmetic operators are the usual ones:

- + addition
- - subtraction
- * multiplication
- / division
- ˆ power
- - unary minus sign for negative values

If an operand (vector) is shorter than the longest one, it is cyclically repeated and components are reused to fulfill the positional demands. Number 5 above was used five times.

```
> x
[1]  5 10  4  9  1
> x * c(-1, 1)
[1] -5 10 -4  9 -1
```

4.1.3 Generating regular sequences

Regular sequences are often needed and can be made by operator `':'` or function `seq()`. The first version can generate increasing or decreasing sequences of integer numbers - step is always 1 or -1.

```
> 1:5
[1] 1 2 3 4 5
> 4:10
[1]  4  5  6  7  8  9 10
> 9:5
[1] 9 8 7 6 5
> -3:5
[1] -3 -2 -1  0  1  2  3  4  5
> -3:(-5)
[1] -3 -4 -5
> -(3:5)
[1] -3 -4 -5
```

Function `seq` can take any step (argument `by`):

```
> seq(1, 5)
[1] 1 2 3 4 5
> seq(1, 5, 0.5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
> seq(from = 1, to = 5, by = 0.5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
> seq(-3, 6, 2)
[1] -3 -1  1  3  5
> seq(6, -3, -3)
[1]  6  3  0 -3
> seq(length = 5, from = 2, by = 0.5)
[1] 2.0 2.5 3.0 3.5 4.0
```

Another useful function is `rep` which repeats arguments certain number of times.

```
> rep(1, 5)
[1] 1 1 1 1 1
> rep(c(1, 2), 3)
[1] 1 2 1 2 1 2
> rep(c(1, 2), each = 3)
[1] 1 1 1 2 2 2
```

4.1.4 Missing values

In some cases values of a vector are not known, are missing. Missing values in R have a code `NA`. In general, in any operation with `NA`, the value becomes `NA`. Function `is.na` tests which values are missing.

```
> u <- c(1:3, NA, 5)
> is.na(u)
[1] FALSE FALSE FALSE  TRUE FALSE
> which(is.na(u))
[1] 4
> u + x
[1]  6 12  7 NA  6
```

Function `which` returns the indices of TRUE values, in this case index 4 of a missing value. There are also other codes for missing or impossible values: `NaN` - not a number and `Inf` for infinite value (like 1/0).

4.1.5 Logical vectors

As you have seen, function `is.na` resulted in a series of `TRUE` and `FALSE`. This are two logical or Boolean constants (can also be abbreviated as `T` or `F`. `is.na` asked "is the component of a vector missing" and the answer was `TRUE` or `FALSE`. Apart to a series of functions `is.something` that ask if argument is of a kind *something*, logical values are results of comparisons.

Comparison and logical operators:

<	less
<=	less or equal
==	equal
>=	greater or equal
>	greater
&	and
|	or
!	not

```
> x
[1]  5 10  4  9  1
> y
[1]  2  4  5 -1  7
> x < y
[1] FALSE FALSE  TRUE FALSE  TRUE
> !(x < y)
[1]  TRUE  TRUE FALSE  TRUE FALSE
> x >= y
[1]  TRUE  TRUE FALSE  TRUE FALSE
> (x < y) & (y > 5)
[1] FALSE FALSE FALSE FALSE  TRUE
> x < y & y > 5
[1] FALSE FALSE FALSE FALSE  TRUE
```

To be on the safe side and have a clear code, use parentheses when in doubt what will happen (see last line of the example).

4.1.6 Logical vectors as filters

To show use of logical values as filters, we will use small dataset with four variables:

```
> smallData <- read.table("../dat/filter.txt", header = TRUE)
> smallData

  g xx yy zz
1 M  1  5  1
2 F  2  8  0
3 F  3  7  5
4 M  4  6  8
5 M  5  3  6

> attach(smallData)
```

Variable `g` represent gender of cases. For males, we want to add values of `x` and `y` and `z` and `y` for females and store new variable as `u`.

First we just calculate the sum of `z` and `y`

```
> u <- zz + yy
```

This is wrong for males, so we will correct the values for males. With variable `select` we will select which elements to change:

```
> select <- g == "M"
> u[select] <- (xx + yy)[select]
> u

[1]  6  8 12 10  8
```

Finally, add new variable to the dataset

```
> smallData <- cbind(smallData, u)
> smallData

  g xx yy zz  u
1 M  1  5  1  6
2 F  2  8  0  8
3 F  3  7  5 12
4 M  4  6  8 10
5 M  5  3  6  8


> select <- (g == "M") & (yy > 4)
> smallData[select, ]

  g xx yy zz  u
1 M  1  5  1  6
4 M  4  6  8 10
```

We do not need the attached data anymore:

```
> detach()
```

4.1.7 Character vectors

Character values and character vectors are used frequently in R , for example as plot labels. When needed they are denoted by a sequence of characters delimited by the double quote character, *e. g.* ”p-value”, ”Results of comparison”. In some prints and plots, they are used without the quote characters.

Character values can be entered using the double (̈) or single (’) quotes.

```
> varname <- c("X", "Y", "Z", "U", "V")
> varname
[1] "X" "Y" "Z" "U" "V"
```

Character values (strings) can be concatenated using `paste` function.

```
> paste(varname, x)
[1] "X 5"  "Y 10" "Z 4"  "U 9"  "V 1"
> paste(c("X", "Y"), x, sep = " = ")
[1] "X = 5"  "Y = 10" "X = 4"  "Y = 9"  "X = 1"
> paste("X", x, sep = " = ", collapse = " , ")
[1] "X = 5 , X = 10 , X = 4 , X = 9 , X = 1"
```

Function `paste` operates on components, and cyclically reuses shorter vector elements.

4.1.8 Index vectors

Use of index vectors is a way to select and modify a subset of values. Index vectors are added in square brackets as selectors to vectors. R recognizes several types of missing values.

Vector of positive values

Only the elements with indices in a vector of positive values are selected.

```
> x[2:4]
[1] 10  4  9
```

Vector of negative values

Negative value is a sign to omit the value with index equal to the opposite value.

```
> x[-(2:3)]
[1] 5 9 1
```

Logical vectors

They have to be of the same size as the subsetting vector. The values corresponding to `TRUE` are selected, those corresponding to `FALSE` are omitted.

```
> x
[1]  5 10  4  9  1
> u
[1]  6  8 12 10  8
> !is.na(u)
[1] TRUE TRUE TRUE TRUE TRUE
> x[!is.na(u)]
[1]  5 10  4  9  1
```

This selects elements with missing values:

```
> (miss <- which(is.na(u)))
integer(0)
> x[miss]
numeric(0)
```

Vector of character strings

Elements, corresponding the character strings are selected.

```
> names(x) <- varname
> x
 X  Y  Z  U  V
 5 10  4  9  1
> x[c("U", "V")]
U V
9 1
> names(x) <- NULL
```

4.1.9 Matrix

Matrix is a collection of vectors (or rows). Matrices can be generated in several ways:

Binding vectors

Vectors can be bind as rows (`rbind`) or as columns (`cbind`)

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

      x  y  z
[1,]  5  2  7
[2,] 10  4 19
[3,]  4  5  8
[4,]  9 -1 12
[5,]  1  7  4

> rbind(x, y, z)

  [,1] [,2] [,3] [,4] [,5]
x    5   10    4    9    1
y    2    4    5   -1    7
z    7   19    8   12    4
```

Reshaping a vector into a matrix

```
> m <- c(x, y, z)
> m

 [1]  5 10  4  9  1  2  4  5 -1  7  7 19  8 12  4

> X <- matrix(m, 5, 3)
> X

     [,1] [,2] [,3]
[1,]    5    2    7
[2,]   10    4   19
[3,]    4    5    8
[4,]    9   -1   12
[5,]    1    7    4

> dim(X)

[1] 5 3
```

A matrix with dimension (5,3) is produced, elements of a vector are columnwise filled into a matrix (can be changed by argument `byrow=TRUE`.

Matrices have two dimensions, number of rows and number of columns.

4.1.10 Some operations on matrices

extraction of diagonal `diag`

```
> diag(X)

[1] 5 4 8
```

Transposition (`t`)

```
> t(X)

     [,1] [,2] [,3] [,4] [,5]
[1,]    5   10    4    9    1
[2,]    2    4    5   -1    7
[3,]    7   19    8   12    4
```

4.1.11 Matrix multiplication %*%

Function `scale` is used for standardization of data matrix. In our case, just the column means are subtracted from each column. Then the sample covariance matrix is calculated in two ways: by explicit matrix multiplication and by function `var`.

```
> (Y <- scale(X, scale = FALSE))

     [,1] [,2] [,3]
[1,] -0.8 -1.4   -3
[2,]  4.2  0.6    9
[3,] -1.8  1.6   -2
[4,]  3.2 -4.4    2
[5,] -4.8  3.6   -6
attr(,"scaled:center")
[1]  5.8  3.4 10.0

> n <- dim(Y)[1]
> t(Y) %*% Y/(n - 1)

      [,1]  [,2]  [,3]
[1,] 13.70 -7.65 19.75
[2,] -7.65  9.30 -6.00
[3,] 19.75 -6.00 33.50

> var(Y)

      [,1]  [,2]  [,3]
[1,] 13.70 -7.65 19.75
[2,] -7.65  9.30 -6.00
[3,] 19.75 -6.00 33.50
```

4.1.12 Arrays

Arrays are extension of a matrix, they have more than two dimensions.

```
> array(1:12, c(2, 3, 2))

, , 1

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

, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12
```

4.1.13 Data frame

Vectors, matrices and arrays can only hold values of the same type: either numeric or character or logical. Data frames can have columns of different type. An example is `bmiData`.

Different types of data will be collected by column, but columns have to have the same length!

```
> case <- paste("Case00", 1:5, sep = "-")
> xData <- data.frame(case, first = y, second = u, z, miss.u = is.na(u))
> xData
```

```
      case first second  z miss.u
1 Case00-1     2      6  7  FALSE
2 Case00-2     4      8 19  FALSE
3 Case00-3     5     12  8  FALSE
4 Case00-4    -1     10 12  FALSE
5 Case00-5     7      8  4  FALSE
```

Use `cbind` and `rbind` to add additional columns and rows.

List

List is the richest structure in R . It can incorporate components of any kind.

```
> lst <- list(case, x, p = 0.05, data = xData)
> lst
```

```
[[1]]
[1] "Case00-1" "Case00-2" "Case00-3" "Case00-4" "Case00-5"

[[2]]
[1]  5 10  4  9  1

$p
[1] 0.05

$data
      case first second  z miss.u
1 Case00-1     2      6  7  FALSE
2 Case00-2     4      8 19  FALSE
3 Case00-3     5     12  8  FALSE
4 Case00-4    -1     10 12  FALSE
5 Case00-5     7      8  4  FALSE
```

Components may be named. In any case, one can extract the component by number.

```
> lst[[2]]
```

```
[1]  5 10  4  9  1
```

```
> lst$p
```

```
[1] 0.05
```

```
> lst[["p"]]
```

```
[1] 0.05
```

The first line prints the second component. The second and third line extract the `p` value.

Use use function combine `c` to add additional components to lists.

```
> extendedlst <- c(lst, added = "added component")
```

4.2 Getting information about data structures

We have the following objects: vector `x`, matrix `X`, data.frame `xData` and list `lst`.

```
> x

[1]  5 10  4  9  1

> X

     [,1] [,2] [,3]
[1,]    5    2    7
[2,]   10    4   19
[3,]    4    5    8
[4,]    9   -1   12
[5,]    1    7    4

> xData

      case first second  z miss.u
1 Case00-1     2      6  7  FALSE
2 Case00-2     4      8 19  FALSE
3 Case00-3     5     12  8  FALSE
4 Case00-4    -1     10 12  FALSE
5 Case00-5     7      8  4  FALSE

> lst

[[1]]
[1] "Case00-1" "Case00-2" "Case00-3" "Case00-4" "Case00-5"

[[2]]
[1]  5 10  4  9  1

$p
[1] 0.05

$data
      case first second  z miss.u
1 Case00-1     2      6  7  FALSE
2 Case00-2     4      8 19  FALSE
3 Case00-3     5     12  8  FALSE
4 Case00-4    -1     10 12  FALSE
5 Case00-5     7      8  4  FALSE
```

Several functions can give you information about the objects.

4.2.1 Number of elements in a structure `length`

```
> length(x)
[1] 5
> length(X)
[1] 15
> length(xData)
[1] 5
> length(lst)
[1] 4
```

4.2.2 Array organization of a structure `dim`

```
> dim(x)
NULL
> dim(X)
[1] 5 3
> dim(xData)
[1] 5 5
> dim(lst)
NULL
```

4.2.3 Variable names `names`

```
> names(x)
NULL
> names(X)
NULL
> names(xData)
[1] "case"   "first"  "second" "z"      "miss.u"
> names(lst)
[1] ""     ""     "p"    "data"
```

4.2.4 Dimension names of an array `dimnames`

```
> dimnames(x)
NULL
> dimnames(X)
NULL
> dimnames(xData)
[[1]]
[1] "1" "2" "3" "4" "5"

[[2]]
[1] "case"   "first"  "second" "z"      "miss.u"
> dimnames(lst)
NULL
```

4.2.5 Object structure `str`

```
> str(x)
 num [1:5] 5 10 4 9 1
> str(X)
 num [1:5, 1:3] 5 10 4 9 1 2 4 5 -1 7 ...
> str(xData)
'data.frame':        5 obs. of  5 variables:
 $ case  : Factor w/ 5 levels "Case00-1","Case00-2",..: 1 2 3 4 5
 $ first : num  2 4 5 -1 7
 $ second: int  6 8 12 10 8
 $ z     : num  7 19 8 12 4
 $ miss.u: logi  FALSE FALSE FALSE FALSE FALSE
> str(lst)
List of 4
 $     : chr [1:5] "Case00-1" "Case00-2" "Case00-3" "Case00-4" ...
 $     : num [1:5] 5 10 4 9 1
 $ p   : num 0.05
 $ data:'data.frame':        5 obs. of  5 variables:
  ..$ case  : Factor w/ 5 levels "Case00-1","Case00-2",..: 1 2 3 4 5
  ..$ first : num [1:5] 2 4 5 -1 7
  ..$ second: int [1:5] 6 8 12 10 8
  ..$ z     : num [1:5] 7 19 8 12 4
  ..$ miss.u: logi [1:5] FALSE FALSE FALSE FALSE FALSE
```

4.3 Selecting columns and rows

Rows and columns can be selected from matrices and data frames.

```
> X
     [,1] [,2] [,3]
[1,]    5    2    7
[2,]   10    4   19
[3,]    4    5    8
[4,]    9   -1   12
[5,]    1    7    4
> X[2,3]          # element in row 2, col 4
[1] 19
> X[2,]           # second row
[1] 10  4 19
> X[,c(1,3)]      # column 1 and 3
     [,1] [,2]
[1,]    5    7
[2,]   10   19
[3,]    4    8
[4,]    9   12
[5,]    1    4

> xData
      case first second  z miss.u
1 Case00-1     2      6  7  FALSE
2 Case00-2     4      8 19  FALSE
3 Case00-3     5     12  8  FALSE
4 Case00-4    -1     10 12  FALSE
5 Case00-5     7      8  4  FALSE
> xData[,3]       # column 3
[1]  6  8 12 10  8
> xData[,"second"]# column with name "second"
[1]  6  8 12 10  8
> xData[2:4,]     # rows 2,3, and 4
      case first second  z miss.u
2 Case00-2     4      8 19  FALSE
3 Case00-3     5     12  8  FALSE
4 Case00-4    -1     10 12  FALSE
```

4.4 Replacing values

Values can be replaced by assigning new values to an referenced element. Function `which` can give also the array indices, useful for replacement of missing data.

```
> y
> (y[3:4] <- NA)
> X
> X[2,] <- c(1,-1,NA)
> xData$first <- 1:5
> ismiss <- which(is.na(X),arr.ind=TRUE)
> ismiss
> X[which(is.na(X),arr.ind=TRUE)] <- 9999
> X
```

4.5 Factors

Descriptive, categorical variables can be either plain vectors of character strings or promoted to factors and ordered factors (nominal and ordinal scales of measurement).

In the BMI example, we had codes `S`, `s`, `N`, `h`, `H` as increasing BMI codes. In the data we can have a vector of character strings, which will be converted to a factor and ordered factor.

```
> b <- c("S","s","s","N","h","H","h","N")
> b

[1] "S" "s" "s" "N" "h" "H" "h" "N"

> #                 Factor
> fb <- factor(b)
> fb

[1] S s s N h H h N
Levels: h H N s S

> str(fb)

 Factor w/ 5 levels "h","H","N","s",..: 5 4 4 3 1 2 1 3

> levels(fb)

[1] "h" "H" "N" "s" "S"


> #                 Ordered factor
> ob <- factor(b,levels=c("S", "s", "N", "h", "H"),ordered=TRUE)
> ob

[1] S s s N h H h N
Levels: S < s < N < h < H

> str(ob)

 Ord.factor w/ 5 levels "S"<"s"<"N"<"h"<..: 1 2 2 3 4 5 4 3

> levels(ob)

[1] "S" "s" "N" "h" "H"

> # internal codes
> as.numeric(ob)

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

4.5.1 Recoding

Sometimes we need to combine or recode factor levels. For example, one wants to combine very small (`"S"`) and small (`"s"`) categories into one category (let's say with common code `"s"`). Levels can be recoded in the naive way by replacing the selection of old codes with new ones.

```
> levels(ob)[levels(ob) == "S"] <- "s"
> ob
```

```
[1] s s s N h H h N
Levels: s < N < h < H
```

You can combine several levels into one by a set operator `%in%`.

```
> levels(ob)[levels(ob) %in% c("H", "h")] <- "HighBMI"
> ob
```

```
[1] s       s       s       N       HighBMI HighBMI HighBMI N
Levels: s < N < HighBMI
```

You can drop unnecessary levels

```
> ob[, drop = T]
```

```
[1] s       s       s       N       HighBMI HighBMI HighBMI N
Levels: s < N < HighBMI
```

More elaborate procedures for recoding can be found in user packages, e.g. `recode()` in package **epicalc**.

4.6 Naming rows and columns

If possible, names are automatically added to columns and rows. Dimension names are stored as a list of vectors, so one can replace components of dimension names by other vectors. The only limitation is the uniqueness of the values in given names.

4.7 Removing objects

Objects can be removed from the workspace by the remove function `rm()`.

5 Data import/export

The main function for file reading and writing are functions `read.table` and `write.table` with modifications. Tab delimited text files are the most convenient form of data for input. R is capable of reading and writing data from many sources and data formats. You can use R to exchange files on the local machine, network or eve on the Internet. It can read data tables in various native formats for many statistical packages. Many data interface functions are described in the package **foreign**. Package **Hmisc** has enhanced versions of functions for reading and writing SPSS, STATA, and SAS data. Excel interface is available in package **xlsReadWrite**. Another option is to use ODBC interface available in package **RODBC**. You can also use function `read.xls()` from package **gdata**, which can be used on all platforms. lfn <- normalizePath("../dat/birthdeath.xls") X2<-read.xls(lfn,from=2,rowNames=TRUE) X2 You can find additional information on the *R-project* web page, accessible also through the **Help | Manuals (in PDF) | R Data Import/Export**. You can find some examples in the Appendix.

5.1 Reading data from a text file

```
> Dat <- read.table("../dat/Data.txt")
> dimnames(Dat[1:6, ])
[[1]]
[1] "1" "2" "3" "4" "5" "6"

[[2]]
 [1] "V1"  "V2"  "V3"  "V4"  "V5"  "V6"  "V7"  "V8"  "V9"  "V10" "V11"
> Dat <- read.table("../dat/Data.txt", header = T)
> dimnames(Dat[1:6, ])
[[1]]
[1] "1" "2" "3" "4" "5" "6"

[[2]]
 [1] "country"       "continent"     "infantMort"    "lifeExpec"
 [5] "population"    "birthRate"     "nPhysicians"   "physicians"
 [9] "gdp"           "internetUsers" "tvSets"
> Dat <- read.table("../dat/Data.txt", header = T, row.names = 1)
> dimnames(Dat[1:6, ])
[[1]]
[1] "Albania"    "Angola"     "Australia"  "Austria"    "Bangladesh"
[6] "Brazil"

[[2]]
 [1] "continent"     "infantMort"    "lifeExpec"     "population"
 [5] "birthRate"     "nPhysicians"   "physicians"    "gdp"
 [9] "internetUsers" "tvSets"
```

There are other functions for reading tables (see `help("read.table")`). Use `sep` argument to control separator character (comma, semicolon, tab (`\t`).

```
> sData <- read.delim("../dat/sData.txt", row.names = 1)
> dimnames(sData[1:6, ])
[[1]]
[1] "Albania"    "Angola"     "Australia"  "Austria"    "Bangladesh"
[6] "Brazil"

[[2]]
 [1] "continent"     "infantMort"    "lifeExpec"     "population"
 [5] "birthRate"     "nPhysicians"   "physicians"    "gdp"
 [9] "internetUsers" "tvSets"
```

5.2 Reading Excel files

Package **xlsReadWrite** has functions for reading and writing tables in Excel native (binary) format. The function `read.xls()` can not use relative paths. One can use function `normalizePath()` to convert relative file path to the absolute one:

```
> library("xlsReadWrite")
> (lfn <- normalizePath("../dat/bmiall.xls"))

[1] "C:\\_Y\\R\\I2R\\dat\\bmiall.xls"

> bmiData <- read.xls(lfn)
> head(bmiData)

  gender age weight height
1      M  17   73.6  1.730
2      M  17   71.0  1.765
3      M  17   62.4  1.770
4      M  17   71.0  1.870
5      M  17   72.4  1.765
6      M  17  104.0  1.825
```

To skip first few lines of the *Excel* table use argument `from`. By argument `rowNames=TRUE`, first column will be used for row names. Argument `sheet` determines which sheet to read.

```
> lfn <- normalizePath("../dat/birthdeath.xls")
> X2 <- read.xls(lfn, from = 2, rowNames = TRUE)
> X2
                X2007 X2006 X2005 X1990 X1985 X1980 X1975 X2007 X2006 X2005
Australia        12.0  12.1  12.3  15.4  15.7  15.3  16.9   7.6   7.5   7.4
Austria           8.7   8.7   8.8  11.6  11.6  12.0  12.5   9.8   9.8   9.7
Belgium          10.3  10.4  10.5  12.6  11.5  12.7  12.2  10.3  10.3  10.2
Czech Republic1   9.0   9.0   9.1  13.4  14.5  16.4  19.6  10.6  10.6  10.5
France           11.9  12.0  12.2  13.5  13.9  14.8  14.1   9.2   9.1   9.1
Germany2          8.2   8.3   8.3  11.4   9.6  10.0   9.7  10.7  10.6  10.6
Greece            9.6   9.7   9.7  10.2  11.7  15.4  15.7  10.3  10.2  10.2
Ireland          14.4  14.5  14.5  15.1  17.6  21.9  21.5   7.8   7.8   7.8
Israel           17.7  18.0  18.2  22.2  23.5  24.1  28.2   6.2   6.2   6.2
Italy             8.5   8.7   8.9   9.8  10.1  11.2  14.8  10.5  10.4  10.3
Japan             9.2   9.4   9.5   9.9  11.9  13.7  17.2   9.4   9.2   9.0
Mauritius        15.3  15.4  15.6  21.0  18.8  27.0  25.1   6.9   6.9   6.8
Netherlands      10.7  10.9  11.1  13.3  12.3  12.8  13.0   8.7   8.7   8.7
New Zealand      13.6  13.8  13.9  18.0  15.6   NaN  18.4   7.5   7.5   7.5
Norway           11.3  11.5  11.7  14.3  12.3  12.5  14.1   9.4   9.4   9.5
Panama           21.5  21.7  22.0  23.9  26.6  26.8  32.3   5.4   5.4   5.3
Poland            9.9   9.9   9.7  14.3  18.2  19.5  18.9   9.9   9.9   9.8
Portugal         10.6  10.7  10.8  11.8  12.8  16.4  19.1  10.6  10.5  10.4
Romania          10.7  10.7  10.7  13.6  15.8   NaN   NaN  11.8  11.8  11.7
Switzerland       9.7   9.7   9.8  12.5  11.6  11.3  12.3   8.5   8.5   8.5
Tunisia          15.5  15.5  15.5  25.8  31.3  35.2  36.6   5.2   5.1   5.1
United Kingdom   10.7  10.7  10.8  13.9  13.3  13.5  12.5  10.1  10.1  10.2
United States    14.2  14.1  14.1  16.7  15.7  16.2  14.0   8.3   8.3   8.2
                X1990 X1985 X1980 X1975
Australia         7.0   7.5   7.4   7.9
Austria          10.6  11.9  12.2  12.8
Belgium          10.6  11.2  11.6  12.2
Czech Republic1  11.7  11.8  12.1  11.5
France            9.3  10.1  10.2  10.6
Germany2         11.2  11.5  11.6  12.1
Greece            9.3   9.4   9.1   8.9
Ireland           9.1   9.4   9.7  10.6
Israel            6.2   6.6   6.7   7.1
Italy             9.4   9.5   9.7   9.9
Japan             6.7   6.2   6.2   6.4
Mauritius         6.5   6.8   7.2   8.1
Netherlands       8.6   8.5   8.1   8.3
New Zealand       7.9   8.4   NaN   8.1
Norway           10.7  10.7  10.1   9.9
Panama            NaN   NaN   NaN   NaN
Poland           10.2  10.3   9.8   8.7
Portugal         10.4   9.6   9.9  10.4
Romania          10.6  10.9   NaN   NaN
Switzerland       9.5   9.2   9.2   8.7
Tunisia           NaN   NaN   NaN   NaN
United Kingdom   11.2  11.8  11.8  11.9
United States     8.6   8.7   8.9   8.9
```

If you look at the file birthdeath.xls, you will see why we have to start reading in the second row and that the first seven columns describe the birth rates while the next seven record the death rates in seven years. When the data were read into R , column names (years) were prefixed by ”X” to make legal names. The following code makes more reasonable names.

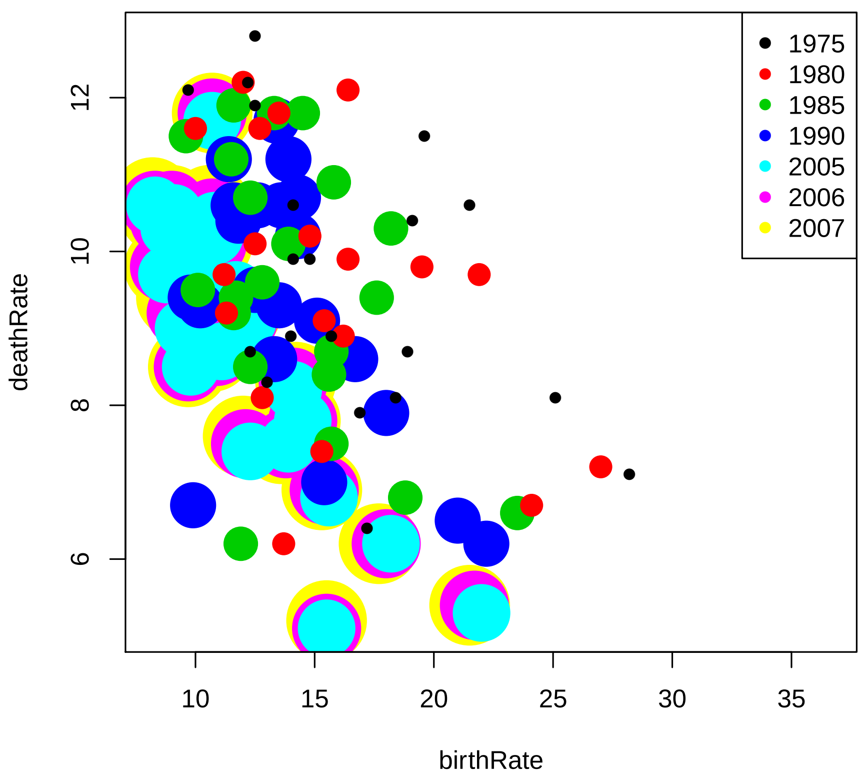
```
> (years <- substring(names(X2), 2, 5))

 [1] "2007" "2006" "2005" "1990" "1985" "1980" "1975" "2007" "2006" "2005"
[11] "1990" "1985" "1980" "1975"

> names(X2) <- paste(rep(c("birth", "death"), each = 7), years,
+     sep = ".")
> t(X2[c("Austria", "Italy"), ])

           Austria Italy
birth.2007     8.7   8.5
birth.2006     8.7   8.7
birth.2005     8.8   8.9
birth.1990    11.6   9.8
birth.1985    11.6  10.1
birth.1980    12.0  11.2
birth.1975    12.5  14.8
death.2007     9.8  10.5
death.2006     9.8  10.4
death.2005     9.7  10.3
death.1990    10.6   9.4
death.1985    11.9   9.5
death.1980    12.2   9.7
death.1975    12.8   9.9
```

Table `birthdeath.xls` is in a shape more common for results than data. It is summarizing two variables: birth rate and death rate across countries and time. One could reshape it into a data frame with four variables: `country`,`year`, `birthRate`, and `deathRate`

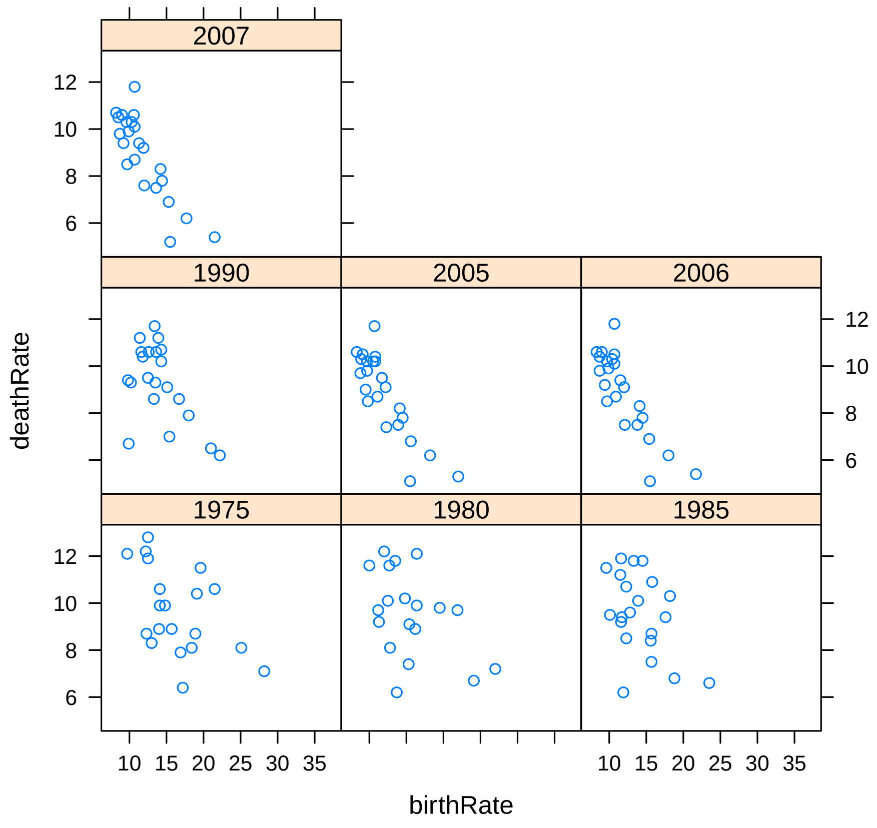
```
> lfn <- normalizePath("../dat/birthdeath.xls")
> X3 <- read.xls(lfn, from = 2)
> names(X3)

 [1] "Country" "X2007"   "X2006"   "X2005"   "X1990"   "X1985"   "X1980"
 [8] "X1975"   "X2007"   "X2006"   "X2005"   "X1990"   "X1985"   "X1980"
[15] "X1975"

> years <- as.numeric(sub("^X(.*)", "\\1", names(X3)[-1]))
> (years <- unique(years))

[1] 2007 2006 2005 1990 1985 1980 1975

> attach(X3)
> X3 <- data.frame(country = as.character(rep(X3[, 1], 7)), year = rep(years,
+     each = dim(X3)[1]), birthRate = as.vector(unlist(X3[, 2:8])),
+     deathRate = as.vector(unlist(X3[, 9:15])))
> detach()
> head(X3)

          country year birthRate deathRate
1       Australia 2007      12.0       7.6
2         Austria 2007       8.7       9.8
3         Belgium 2007      10.3      10.3
4 Czech Republic1 2007       9.0      10.6
5          France 2007      11.9       9.2
6        Germany2 2007       8.2      10.7
```

Are birth and death rate related? Try to explain the pattern on the following figure (size and color code years, small and black are first year, ...).

```
> attach(X3)
> col <- as.numeric(factor(year))
> plot(birthRate, deathRate, col = col, pch = 16, cex = col)
> legend("topright", pch = 16, col = sort(unique(col)), legend = levels(facto
> detach(X3)
```



```
> library(lattice)
> print(xyplot(deathRate ~ birthRate | factor(year), data = X3))
```



5.3 Writing tables

This will write a tab delimited text file, which can be opened by Excel.

```
> X = sData[1:10, 1:5]
> write.table(X, "mytable.xls", sep = "\t")
> write.table(X, "mytable2.xls", sep = "\t", col.names = NA)
```

In the first file (`mytable.xls`), column names are shifted one cell to the left. Due to the `col.names` argument column names are positioned correctly in the second file (`mytable2.xls`).

5.4 Reading and writing data in statistical package formats

Functions for reading data in native formats of statistical packages (e.g. SPSS) are in **foreign**.

5.4.1 Reading SPSS (.SAV) file

Function `read.spss` returns a list with variable and value labels or a data frame:

```
> library(foreign)
> bmiData <- read.spss("../dat/bmi.sav")
> bmiData
```

```
$gender
 [1] Male   Male   Male   Male   Male   Male   Male   Male   Male   Male
[11] Male   Male   Male   Male   Male   Male   Male   Male   Male   Male
[21] Female Female Female Female Female Female Female Female Female Female
[31] Female Female Female Female Female Female Female Female Female Female
[41] Female Female Female Female Female
Levels: Female Male

$age
 [1] 17 17 17 17 17 17 17 17 17 18 18 18 18 18 18 18 18 18 18 18 17 17 17 17
[26] 17 17 17 17 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18

$weight
 [1] 64.2 74.8 55.8 68.4 68.2 88.0 59.0 65.2 53.6 77.2 68.2 60.0 74.0 64.0 55
[16] 62.0 60.6 71.6 62.2 88.4 46.2 47.0 62.0 58.0 48.6 59.4 54.8 61.6 60.0 59
[31] 49.6 49.8 53.0 50.0 48.8 57.2 65.6 61.8 62.2 57.4 59.0 51.2 49.4 68.0 64

$height
 [1] 1.770 1.705 1.770 1.730 1.753 1.910 1.740 1.765 1.725 1.814 1.710 1.760
[13] 1.887 1.720 1.710 1.750 1.710 1.840 1.775 1.865 1.535 1.670 1.720 1.635
[25] 1.653 1.755 1.643 1.700 1.747 1.735 1.695 1.620 1.657 1.637 1.687 1.700
[37] 1.743 1.715 1.720 1.615 1.635 1.565 1.640 1.720 1.690

attr(,"label.table")
attr(,"label.table")$gender
      Male     Female
"M       " "F       "

attr(,"label.table")$age
NULL

attr(,"label.table")$weight
NULL

attr(,"label.table")$height
NULL

attr(,"variable.labels")
              gender                  age               weight
            "Gender" "Age at measurement"        "Weight (kg)"
              height
        "Height (m)"
```

```
> attach(bmiData)
> table(gender, age)
```

```
        age
gender   17 18
  Female  9 16
  Male    9 11
```

```
> library(foreign)
> bmiData <- read.spss("../dat/bmi.sav", to.data.frame = TRUE)
> head(bmiData)
  gender age weight height
1   Male  17   64.2  1.770
2   Male  17   74.8  1.705
3   Male  17   55.8  1.770
4   Male  17   68.4  1.730
5   Male  17   68.2  1.753
6   Male  17   88.0  1.910
```

5.4.2 Reading Stata binary file

Function `read.dta` reads *Stata* binary files (version 5-10).

5.4.3 Writing in package formats

Function `write.foreign` exports simple data frames to other statistical packages by writing the data as free-format text and writing a separate file of instructions for the other package to read the data.

6 Elementary statistics

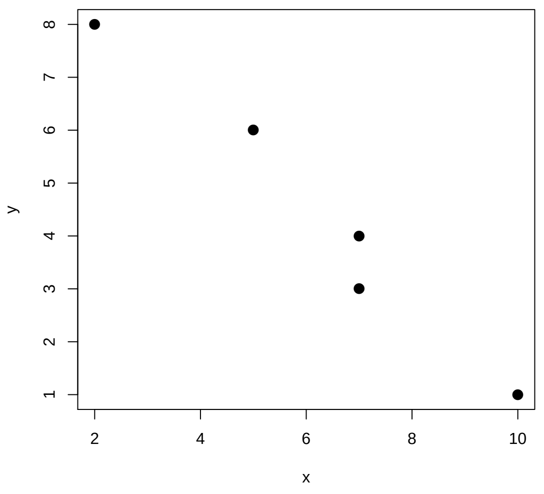
R has functions for almost all basic statistical operations. The use is quite intuitive, naming the operation to be performed on an object. Missing data handling is controlled by arguments `na.rm` (`NA` remove before calculation) or `na.omit` (see help information for details).

6.1 Basic statistical functions

Let us make a small dataset with five cases and two variables `x` and `y`:

```
> x <- c(7, 2, 10, 5, 7)
> y <- c(4, 8, 1, 6, 3)
> X <- cbind(x, y)
```

```
> plot(x, y, pch = 16, cex = 1.5)
```



6.1.1 Range and sum

```
> min(x)
[1] 2
> max(x)
[1] 10
> range(x)
[1]  2 10
> sum(x)
[1] 31
> cumsum(x)
[1]  7  9 19 24 31
> cumprod(x)
[1]    7   14  140  700 4900
> diff(x)
[1] -5  8 -5  2
```

6.1.2 Descriptives

```
> mean(x)
[1] 6.2
> median(x)
[1] 7
> sd(x)
[1] 2.949576
> sd(X)
       x        y
2.949576 2.701851
> var(x)
[1] 8.7
> var(X)
      x     y
x  8.70 -7.85
y -7.85  7.30
> cor(X)
          x         y
x  1.000000 -0.985028
y -0.985028  1.000000
```

6.1.3 Quantiles

```
> quantile(x, 0.9)
90%
8.8
> quartile = function(x) quantile(x, c(0.25, 0.5, 0.75))
> quartile(x)
25% 50% 75%
  5   7   7
```

6.1.4 Ranking and sorting

```
> rank(x)
[1] 3.5 1.0 5.0 2.0 3.5
> sort(x)
[1]  2  5  7  7 10
> rev(x)
[1]  7  5 10  2  7
> order(x)
[1] 2 4 1 5 3
```

6.1.5 Frequency bins and tables

Conversion of numeric variable to categories (frequency bins)

```
> x
[1]  7  2 10  5  7
> (bins <- cut(x, breaks = c(1, 3, 7, 10, 12)))
[1] (3,7]  (1,3]  (7,10] (3,7]  (3,7]
Levels: (1,3] (3,7] (7,10] (10,12]
> table(bins)
bins
  (1,3]   (3,7]  (7,10] (10,12]
      1       3       1       0
```

Conversion of bin values to numeric values:

```
> as.numeric(bins)
[1] 2 1 3 2 2
```

gives the bin numbers.

6.1.6 Outer product

Outer product function `outer` can be useful to calculate products of marginal distributions:

```
> a = c(2, 3)
> b = c(4, 5)
> outer(a, b)

     [,1] [,2]
[1,]    8   10
[2,]   12   15

> outer(a, b, paste, sep = "x")

     [,1]  [,2]
[1,] "2x4" "2x5"
[2,] "3x4" "3x5"
```

6.1.7 Marginal sums and means

```
> X

      x y
[1,]  7 4
[2,]  2 8
[3,] 10 1
[4,]  5 6
[5,]  7 3
```

```
> colMeans(X)

  x   y
6.2 4.4

> colSums(X)

 x  y
31 22
```

```
> rowSums(X)

[1] 11 10 11 11 10

> rowMeans(X)

[1] 5.5 5.0 5.5 5.5 5.0
```

6.2 apply(): Application of functions to matrix rows and columns

Function `apply` can be used to get similar results for arbitrary function that works on vectors: Column (note second argument) and row means

```
> apply(X, MARGIN = 2, FUN = mean)

  x   y
6.2 4.4

> apply(X, 1, mean)

[1] 5.5 5.0 5.5 5.5 5.0
```

The first argument is a matrix with data, second argument (`MARGIN`) determines along which margin (rows are first dimension or margin and columns are second dimension, thus value 1 for rows and value 2 for columns) we want to apply function in the third argument `FUN`. Any additional arguments for function FUN can be listed as well.

Column standard deviations

```
> apply(X, 2, sd)

       x        y
2.949576 2.701851
```

Minimal value in a row

```
> apply(X, 1, min)

[1] 4 2 1 5 3
```

Function `whichMin()` finds the index of a minimal value in a vector It is an adaptation of function `which()` that accepts a vector of logical values and returns indices of `TRUE` values.

```
> whichMin <- function(x) {
+     which(x == min(x))
+ }
> whichMin(c(5, 1, 6, 3, 2, 9))

[1] 2
```

Applying function `whichMin()` to columns (variables), gives the row number (case number) that has a minimal value for given variable (check in matrix `X`)

```
> (id <- apply(X, 2, FUN = whichMin))

x y
2 3

> X[id, ]

      x y
[1,]  2 8
[2,] 10 1

> diag(X[id, ])

[1] 2 1
```

6.3 Missing values

Handling of missing values (`NA` and `NaN`)is an important question in data analysis. Most of statistical functions in R have some default procedure and there are function arguments to control the behavior of functions when missing values are present. Be careful and examine the help pages for possible missing values options: in some elementary vector functions you will find `na.rm` (meaning `NA` remove) which can be either `TRUE` or `FALSE`: if TRUE, `NA`s will be removed prior to calculation. In functions that operate on matrices, you will find different options like `"everything"`, `"all.obs"`, `"complete.obs"`, `"na.or.complete"`, or `"pairwise.complete.obs"` or `na.omit` and `na.fail`.

Function `sum` (and many others) have defaulted argument `na.rm=FALSE`

```
> args(sum)
function (..., na.rm = FALSE)
NULL
```

and will return result `NA` if some values are missing.

```
> z <- c(8, 9, NA, 7, 6)
> sum(z)
[1] NA
```

When we set argument `na.rm=TRUE`, `NA`s will be removed prior to calculation:

```
> sum(z, na.rm = TRUE)
[1] 30
```

One can remove `NA`s by use of function `is.na`: Function `is.na` returns `TRUE` if value is missing (`NA`).

```
> z
[1]  8  9 NA  7  6
> is.na(z)
[1] FALSE FALSE  TRUE FALSE FALSE
> which(is.na(z))
[1] 3
> u <- z[-which(is.na(z))]
> u
[1] 8 9 7 6
> sum(u)
[1] 30
```

Missing values can be replaced by some other value:

```
> (u <- z)
[1]  8  9 NA  7  6
> u[which(is.na(z))] <- 999
> u
[1]   8   9 999   7   6
```

Mean value imputation:

```
> (u <- z)
[1]  8  9 NA  7  6
> u[which(is.na(z))] <- mean(u, na.rm = TRUE)
> u
[1] 8.0 9.0 7.5 7.0 6.0
```

Multiple `NA` replacement

```
> (u <- z)

[1]  8  9 NA  7  6

> u[1] <- NA
> u

[1] NA  9 NA  7  6

> u[which(is.na(u))] <- mean(u, na.rm = TRUE)
> u

[1] 7.333333 9.000000 7.333333 7.000000 6.000000
```

7 Writing functions

It pays to pack sequences of commands that are executed together and are used very often. You can always copy and paste them from some text editor, but it is a bit clumsy and rigid approach, requiring for example that data are always named in the same way. A much better style is to pack such commands as a `function`.

7.1 R functions

All standard R commands are functions. Functions are objects, that can get input parameters via arguments and perform some operation with a visible or (sometimes) invisible effect. Functions can, for example, receive some data as input; then perform some operation like calculate something, plot some graph, save data to disc; and possibly return an object with results. The structure of a function is

```
function( arguments ){
    statement 1
    statement 2 ...

    return( object )
}
```

A function is a group of expressions (grouped expressions are enclosed in curly braces `{...}`) introduced by an argument list. This structure can be assigned to an object with some name, which is used to invoke this function.

As an example, we will prepare a function that will write data to disk in a style of the last example in the section Writing tables. We would like to avoid thinking about the `sep` and `col.names` in future writings of data to disc. We will name the function `store.table`.

```
> store.table <- function(x, file) {
+     write.table(x, file = file, sep = "\t", col.names = NA)
+ }
```

The function has two arguments, the first is an object name (no double quotes needed!) and the second is the file name (string, quotes needed at function call!) Argument names (`x` and `file` are the placeholders and are sort of nickname for actual names that will be used at the function call. Inside the function, the first argument will be known as `x` wile the second argument is known as `file` and are used to pass actual arguments (`X` as `x` and ”`mytable3.xls`” as `file`.

Now the table `X` can be stored to disc via the function call:

```
> X <- data.frame(first = seq(1:5), chars = letters[1:5])
> store.table(X, "mytable3.xls")
```

Next, we will write a function that calculates confidence interval for given data `x` and selectable confidence `level`.

```
> CI <- function(x, level = 0.95) {
+     n <- length(x)
+     m <- mean(x)
+     SE <- sd(x)/sqrt(n)
+     q <- qnorm(c(0.5 - level/2, 0.5 + level/2))
+     ci <- m + q * SE
+     return(ci)
+ }
> xdat <- rnorm(n = 16, mean = 100, sd = 4)
> CI(xdat)

[1]  98.91263 103.08151
```

In this case, argument `x` is required (no default value) while `level` is optional since it has a default value `0.95`! This function returns a vector of two values stored in the object `ci`. The last statement in a function, namely `return(ci)`, is used to explicitly state which object will be returned by a function. If the returned value is not assigned to an object, it’s value is typed into the command window. In some cases (when the returned object is long or has a complex structure) printing of the returned value is undesirable and can be prevented by invisibly returning the function result. An example of the `invisible()` statement is given in the refined function below.

Since it is easy to forget the confidence level of the calculated interval, a more informative result can be returned; we can also print some report, if the user wants it:

```
> CI <- function(x, level = 0.95, verbose = FALSE) {
+     n <- length(x)
+     m <- mean(x)
+     SE <- sd(x)/sqrt(n)
+     q <- qnorm(0.5 + c(-level/2, level/2))
+     ci <- m + q * SE
+     if (verbose) {
+         cPerc <- paste(level * 100, "%", sep = "")
+         cat(cPerc, "confidence interval: (", ci[1], ",", ci[2],
+             ")\n")
+     }
+     invisible(list(limits = ci, confidence = level))
+ }
> xdat <- rnorm(n = 16, mean = 100, sd = 4)
> CI(xdat)
> CI(xdat, verbose = TRUE)

95% confidence interval: ( 97.10493 , 100.5006 )

> interval <- CI(xdat)
> interval

$limits
[1]  97.10493 100.50056

$confidence
[1] 0.95
```

Using the optional parameter `verbose` and `if` statement, we can control whether the function itself will report the interval limits or just be silent. We used function `cat` to print nicely formatted result. `print` can print just one object in plain format. In the second call of `CI(xdat,verbose=TRUE)`, the first argument was referenced by position, and the `verbose` part was argument was called by name: otherwise it would be treated as `level` on second position.

7.2 Control structures

R recognizes several control structures, that are usual in other programming languages. Control structures enable looping (repeating the same set of instructions, e.g. `for` structure) and branching (e.g. `if` structure).

7.2.1 Looping: `for` structure

A series of confidence intervals for a set of confidence levels can be produced by using the `for` loop:

```
for(var in seq) expr
```

Expression `expr` will be evaluated for each value in a sequence. In each evaluation, `var` will take corresponding value from sequence `seq`.

```
> for (level in c(0.9, 0.95, 0.99)) CI(xdat, level, TRUE)

90% confidence interval: ( 97.3779 , 100.2276 )
95% confidence interval: ( 97.10493 , 100.5006 )
99% confidence interval: ( 96.57143 , 101.0341 )
```

There are other control structures (see help: `?Control`).

7.2.2 Branching: `if` structure

Beside `for` you will mostly need also `if`, which can control the operation given the logical condition:

```
if(cond) expr

if(cond) expr  else  alt.expr
```

In the first case, expression `expr` will be evaluated only if `cond` is `TRUE`. In the second form, expression `expr` will be evaluated if `cond` is `TRUE`. otherwise `alt.expr` will be evaluated:

```
> p <- 0.02
> alpha <- 0.05
> if (p < alpha) cat("p less than alpha\n") else cat("p greater than alpha\n"

p less than alpha
```

8 Probability distributions

R has a rich family of functions for probability distributions.

Distribution	**R name**	**additional arguments**
beta	`beta`	shape1, shape2, ncp
binomial	`binom`	size, prob
Cauchy	`cauchy`	location, scale
chi-squared	`chisq`	df, ncp
exponential	`exp`	rate
F	`f`	df1, df2, ncp
gamma	`gamma`	shape, scale
geometric	`geom`	prob
hypergeometric	`hyper`	m, n, k
log-normal	`lnorm`	meanlog, sdlog
logistic	`logis`	location, scale
negative binomial	`nbinom`	size, prob
normal	`norm`	mean, sd
Poisson	`pois`	lambda
Student's t	`t`	df, ncp
uniform	`unif`	min, max
Weibull	`weibull`	shape, scale
Wilcoxon	`wilcox`	m, n

”R name” should be preceded by one of the prefix letter to get the actual function:

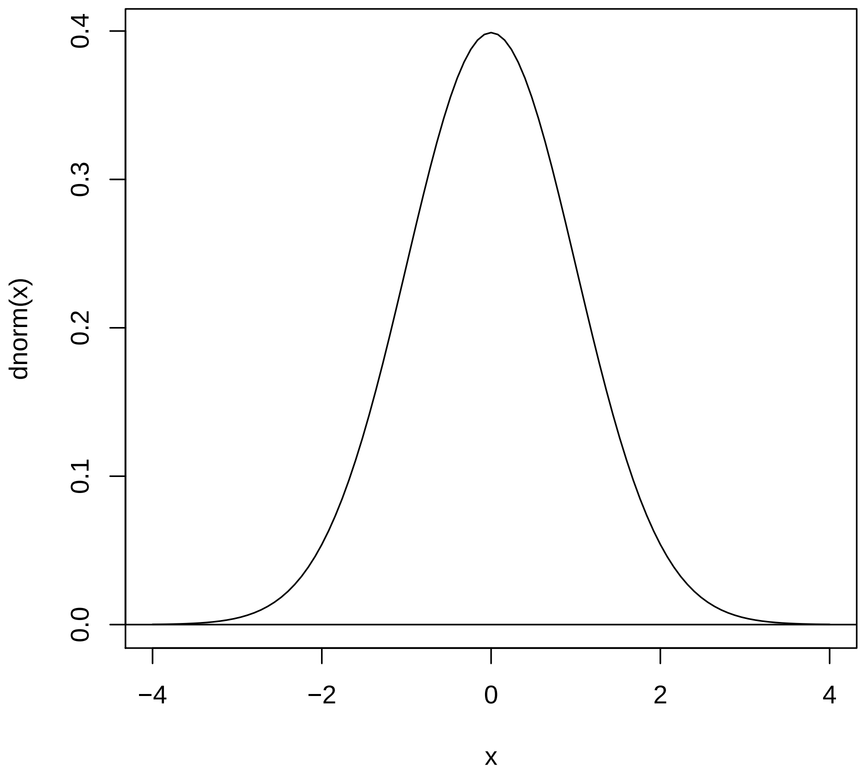
prefix	call	meaning
d	`dnorm(x,...)`	density at x
p	`pnorm(q, ...)`	cumulative probability up to q $P(x \leq q)$
c	`qnorm(p, ...)`	quantile x_p
r	`rnorm(n, ...)`	n random numbers

For normal distribution we get:

```
> dnorm(0)          # density at 0; standardized normal distribution
[1] 0.3989423
> pnorm(1.96)       # cumulative distribution P(Z<1.96)
[1] 0.9750021
> qnorm(0.975)      # quantile for p=0.975
[1] 1.959964
> rnorm(5)          # 5 random numbers
[1]  0.89980762  0.06208036 -1.06860811  0.76641523  0.57688349
```

By use of these functions, one can plot distributions, get probabilities for test, and generate random numbers for simulations.

```
> curve(dnorm(x),xlim=c(-4,4))     # plot a curve for a function
> abline(h=0)                      # horizontal line at 0
```



8.1 Sampling distribution

Sampling distribution is a distribution of estimates. To demonstrate its features, we can perform an repeated sampling experiment. First we will generate $N = 100$ samples with a sample size $n = 9$ normally distributed random numbers. We will organize them into a $N \times n$ matrix and compute mean values (by rows) for all samples.

Computation with `apply` is much more efficient than a use of `for` statement. Then we will plot the histogram of original data and the histogram of estimates.

```
> N=100
> n=9
> X=rnorm(N*n)          # generate random numbers
> X=matrix(X,N,n)       # reshape them into a N x n matrix
> M=apply(X,1,mean)     # calculate the vector of means
> head(M)               # display first few
```

```
[1]  0.57322707 -0.33556026 -0.17591202 -0.37841970  0.02554139  0.10454295
```

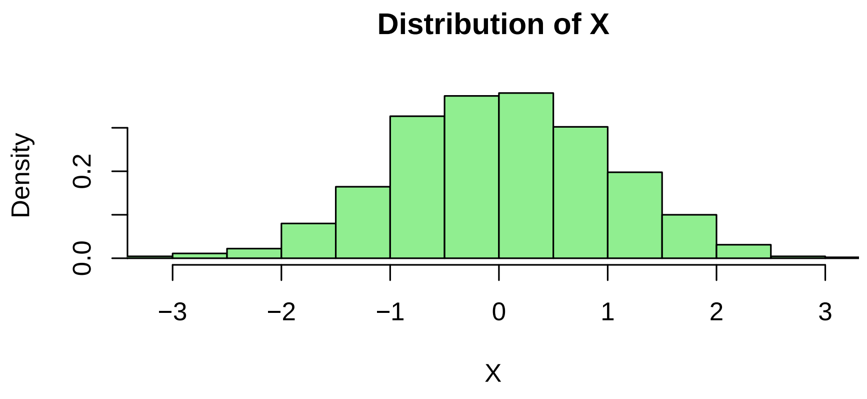
```
> range(M)              # display the range of estimates
```

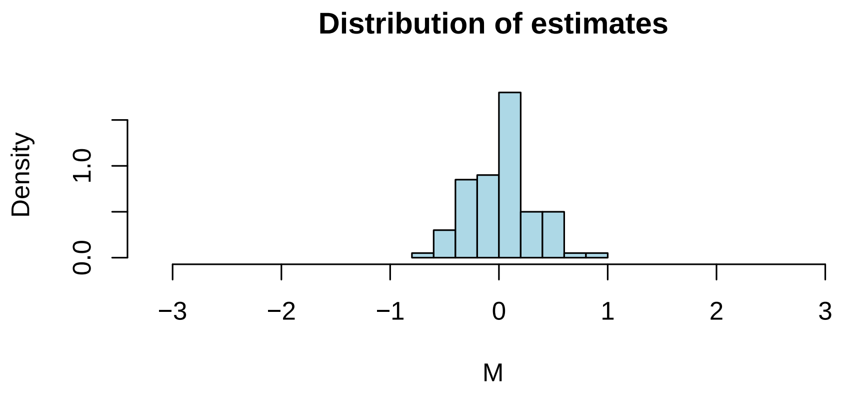
```
[1] -0.5862528  0.6972450
```

```
> #
> par(mfrow=c(2,1))     # organize space for plots in 2 rows, 1 column
> (xlimits=range(X))    # get the range of sampled values
```

```
[1] -2.878217  2.818998
```

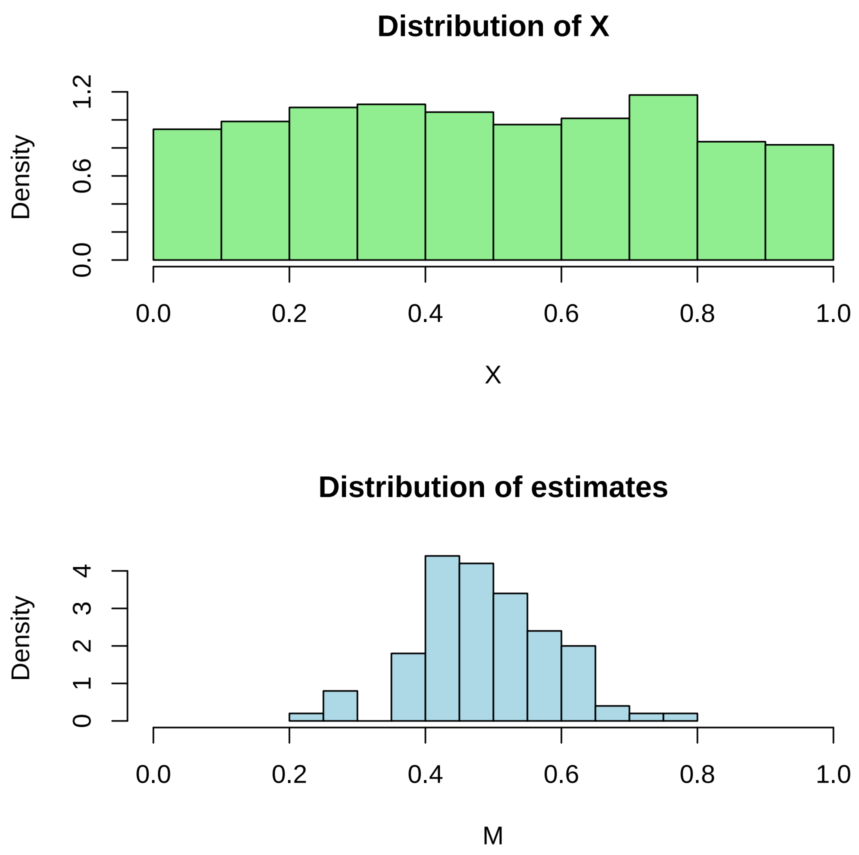
```
> hist(X,prob=TRUE,xlim=xlimits,col="lightgreen",main="Distribution of X")
> hist(M,prob=TRUE,xlim=xlimits,col="lightblue",main="Distribution of estimat
```





Converting this to a function, gives us a lot o possibilities for different experiments:

```
> sDist <- function(N=100,n=9,DIST=rnorm,FUN=mean,...){
+ X=DIST(N*n,...)       # generate random numbers
+ X=matrix(X,N,n)       # reshape them into a N x n matrix
+ M=apply(X,1,FUN)      # calculate the vector of means
+ head(M)               # display first few
+ range(M)              # display the range of estimates
+ #
+ par(mfrow=c(2,1))     # organize space for plots in 2 rows, 1 column
+ (xlimits=range(X))    # get the range of sampled values
+ hist(X,prob=TRUE,xlim=xlimits,col="lightgreen",
+      main="Distribution of X")
+ hist(M,prob=TRUE,xlim=xlimits,col="lightblue",
+      main="Distribution of estimates")
+ }
> sDist(100,9,DIST=runif)
```



Try it by changing the number of samples, sample size, distribution (`DIST`) and function (`FUN`; *e. g.* `median, sd, var,...`)

8.1.1 Function `apply`

Note the use of function `apply`, which is used for aplication of some function to the rows or columns of a matrix. The function `apply` essentially eliminates the usage of `for` loops:

Find the minimum value in each row:

```
> X <- matrix(round(runif(12, 0, 10)), 3, 4)
> X
```

```
     [,1] [,2] [,3] [,4]
[1,]    4    4    7    7
[2,]    6    0    8    4
[3,]    6    2    7    3
```

```
> apply(X, 1, min)
```

```
[1] 4 0 2
```

9 Plotting

R has a rich set of plotting functions. We will examine some predefined functions for basic data and statistical plots, that are customizable via a rich set of plotting parameters. It is advisable to examine the help pages for details and examples. In addition, complete freedom to make a custom graph is given through the so called high and low level plotting commands. High level commands prepare a new graph with basic plot. Low level commands can add additional elements details like points, lines, text and others.

There are galleries of R produced plots. One rich gallery is known as ”R Graphical Manual” http://bm2.genes.nig.ac.jp/RGM2/index.php, another is ”R graph gallery” http://addictedtor.free.fr/graphiques/.

There are also many graphical packages, among them **plotrix** has useful examples and functions.

The first plotting function will open a graphic window and al subsequent plots will be plotted in the same window replacing the previous plot. If you want to keep some plot, open a new graphic window. Function `windows()` will open a new graphic window and keep the previous plot untouched. You can close one graphic window at the time with a function `dev.off()` and all graphics windows at once with `graphics.off()`

9.0.2 Data

To start, we will get some data about different countries in the world. Object `col` holds numerical color codes of continents.

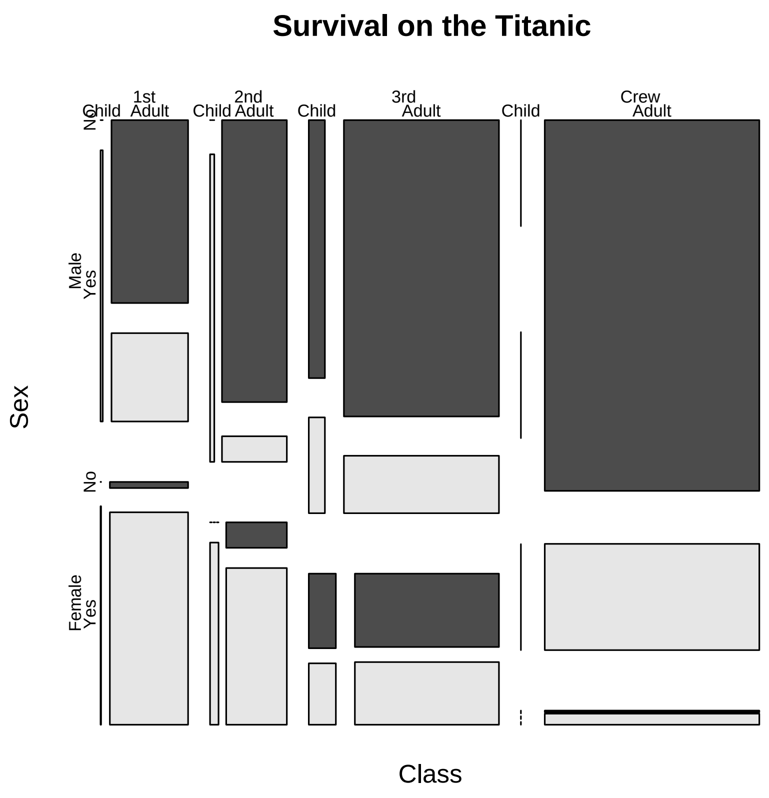
```
> names(sData)
 [1] "country"        "continent"      "infantMort"     "lifeExpec"
 [5] "population"     "birthRate"      "nPhysicians"    "physicians"
 [9] "gdp"            "internetUsers"  "tvSets"
> attach(sData)
> col = as.numeric(as.factor(continent))
```

There are many data sets (like Titanic data) available in R and used in examples in function help.

9.1 Plots for categorical variables

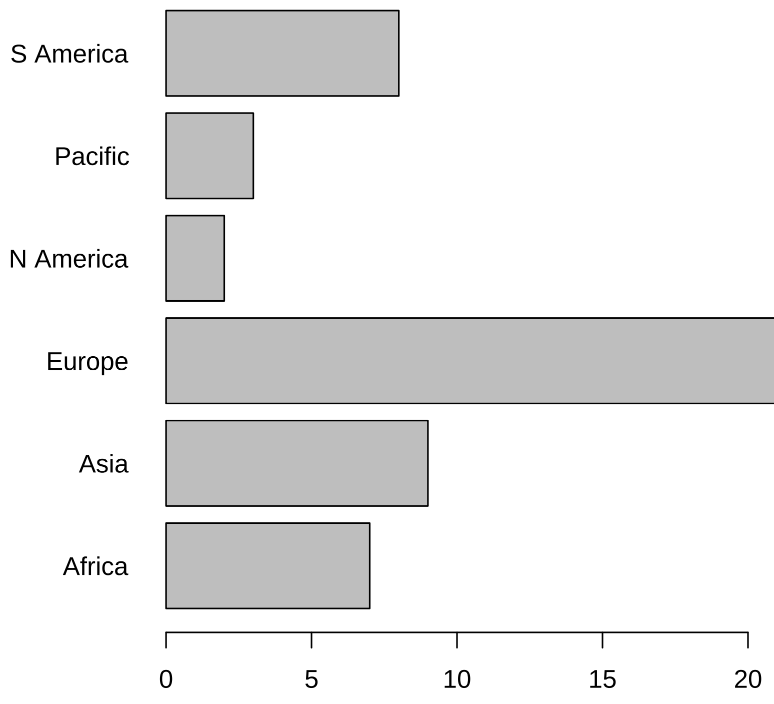
9.1.1 mosaicplot

```
> mosaicplot(Titanic, main = "Survival on the Titanic", color = TRUE)
```



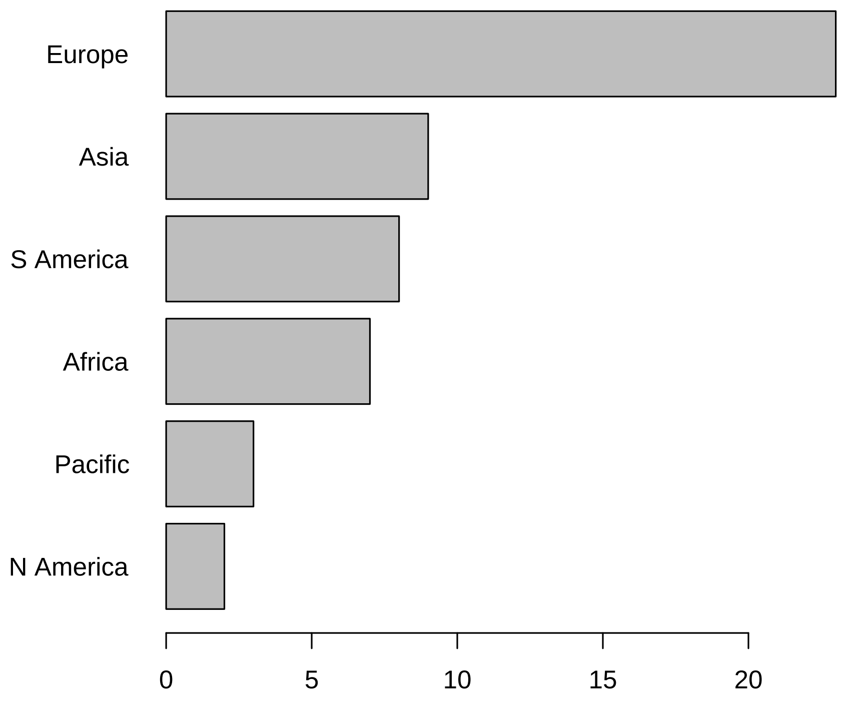
9.1.2 Barplot

```
> entriesPerContinent <- table(sData$continent)
> par(mar = c(5, 6, 4, 2))
> barplot(entriesPerContinent, horiz = TRUE, las = 1)
```



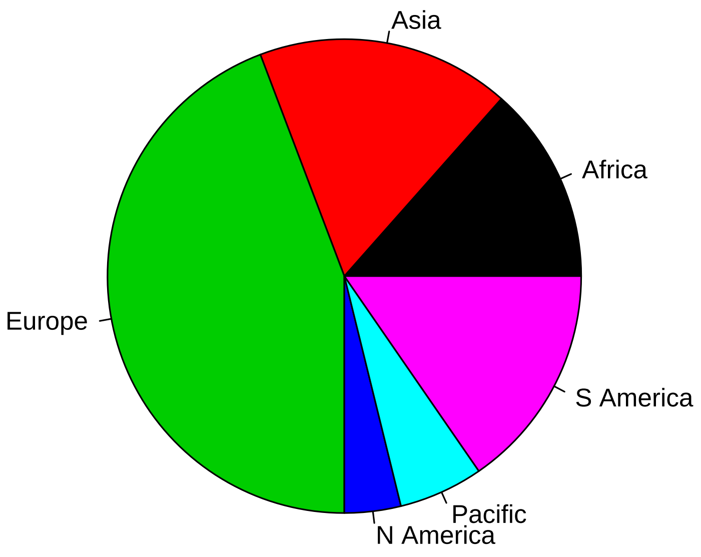
The sequence of plotted bars is determined by the levels of plotted variable (`factor`) - in this case they are ordered by the alphabet. Usually it is better to reorder bars in increasing/decreasing order of plotted values (counts).

```
> entriesPerContinent <- table(reorder(continent, continent, length))
> par(mar = c(5, 6, 4, 2))
> barplot(entriesPerContinent, horiz = TRUE, las = 1)
```


Europe
Asia
S America
Africa
Pacific
N America
0
5
10
15
20

9.1.3 Pie chart

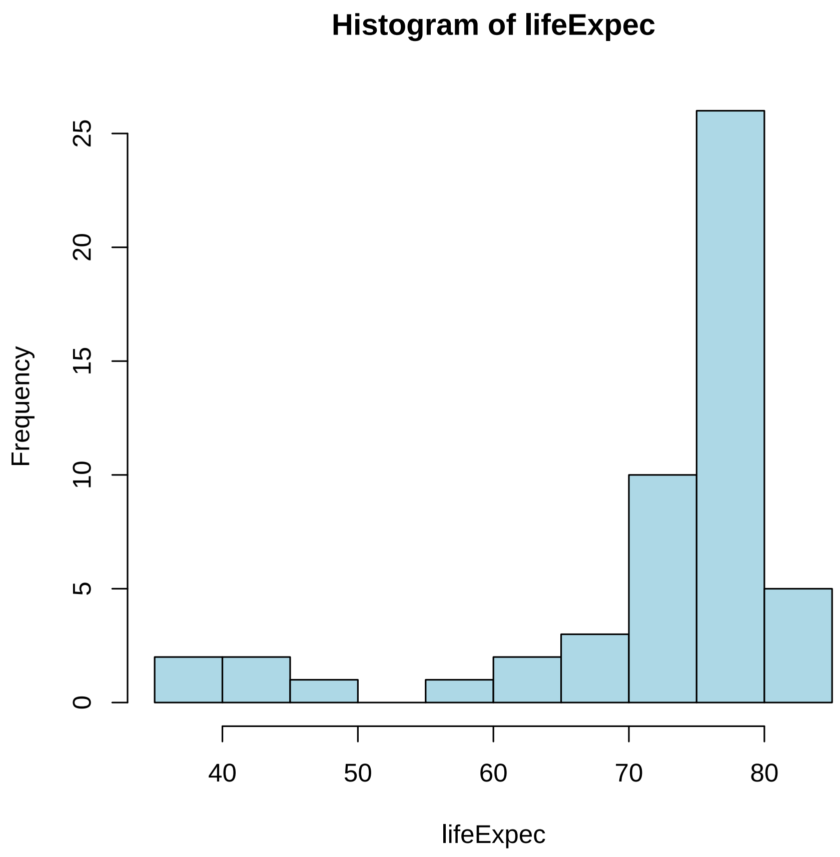
```
> pie(table(sData$continent), col = 1:6)
```



9.2 Plots for numerical variables

9.2.1 Histogram

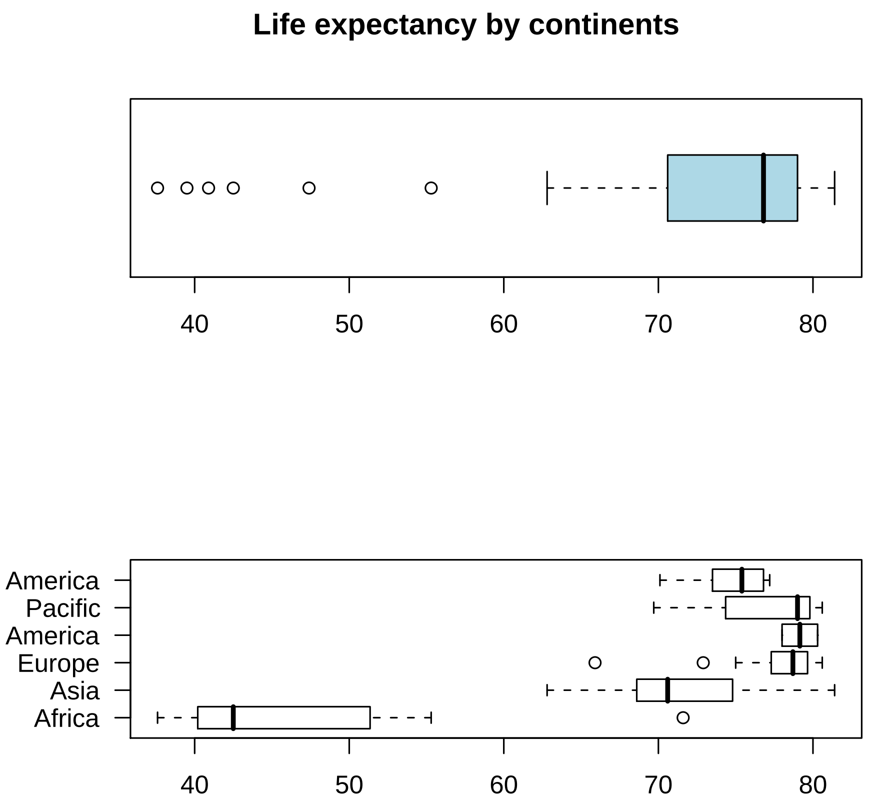
```
> hist(lifeExpec, col = "lightblue")
```



9.2.2 Boxplot

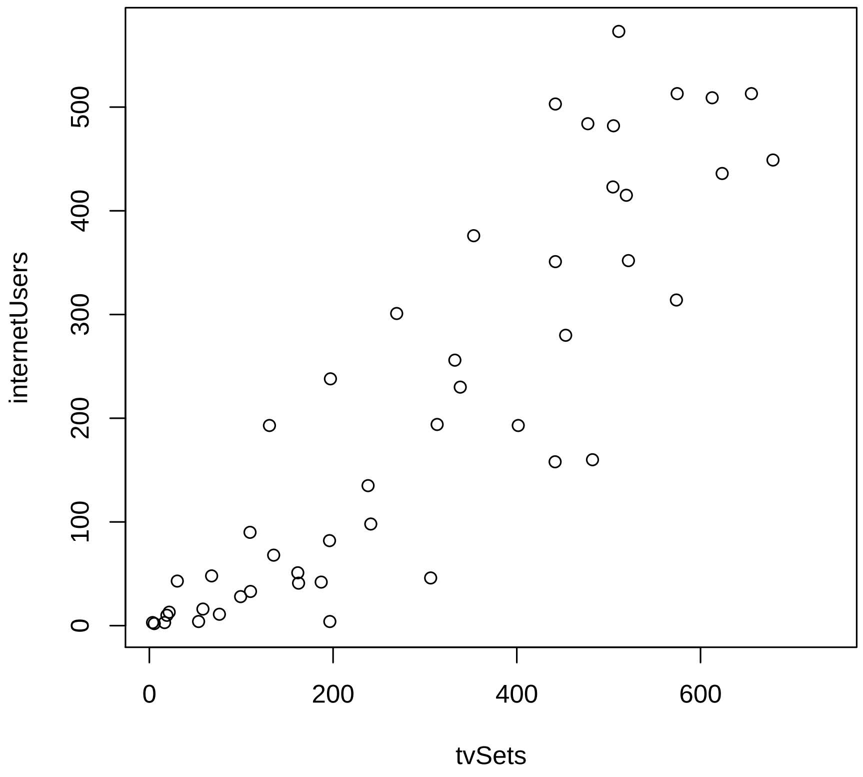
First we will prepare plotting parameters, do some plotting and reset parameters to old values.

```
> oldpar <- par(mfrow = c(2, 1))
> boxplot(lifeExpec, col = "lightblue", horizontal = TRUE)
> boxplot(split(lifeExpec, continent), horizontal = TRUE, las = 1)
> title(main = "Life expectancy by continents", outer = TRUE, line = -2)
> par(oldpar)
```



9.2.3 Scatterplot

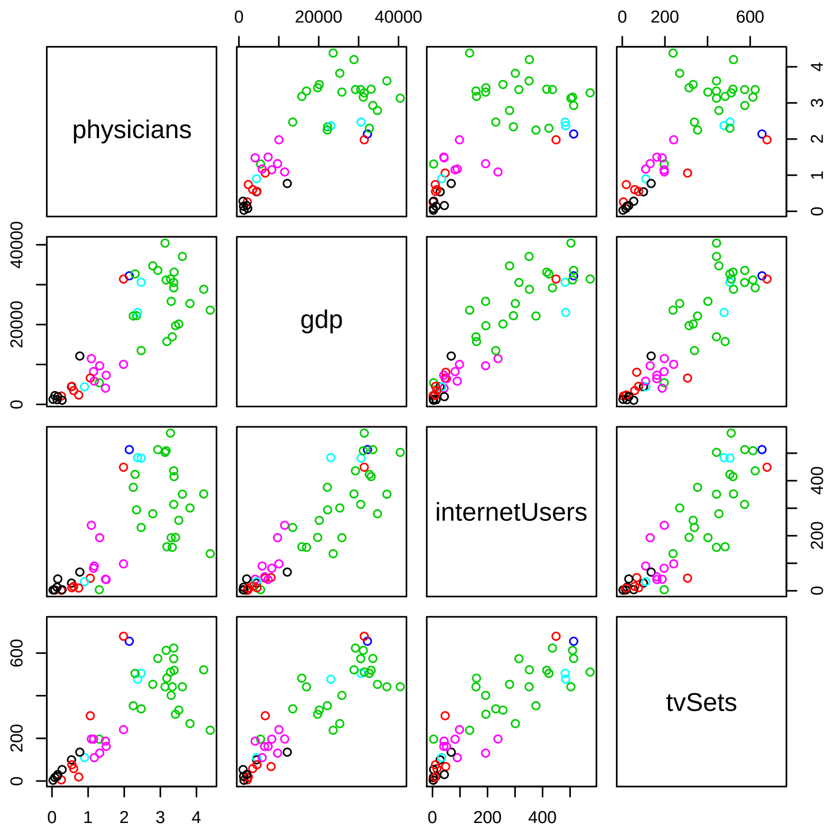
```
> plot(tvSets, internetUsers)
```



9.2.4 Pairs of scatterplots

Pairs of scatterplots are the default plot for matrices.

```
> par(mfrow = c(1, 1))
> pairs(sData[, 8:11], col = col)
```

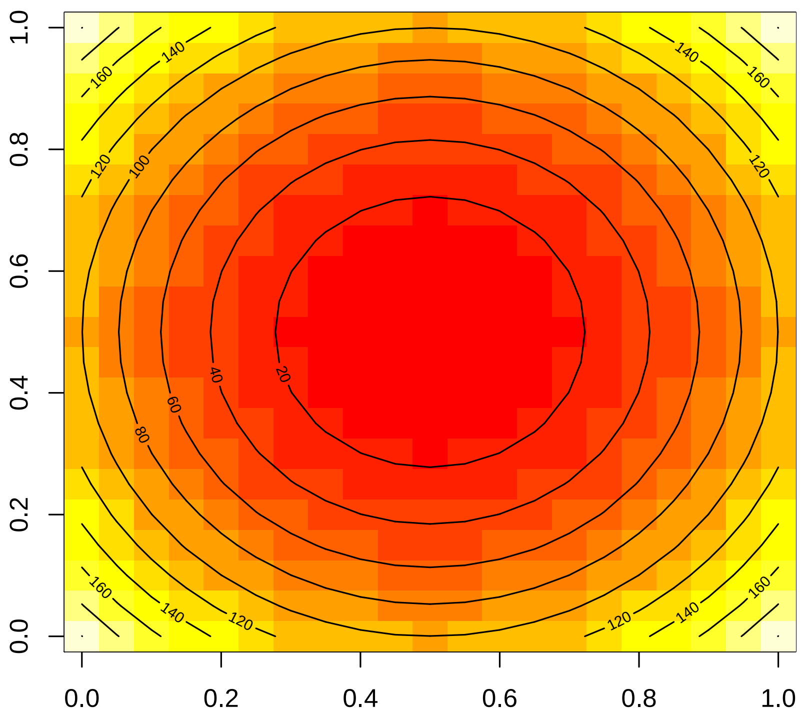


9.2.5 Three dimensional plots

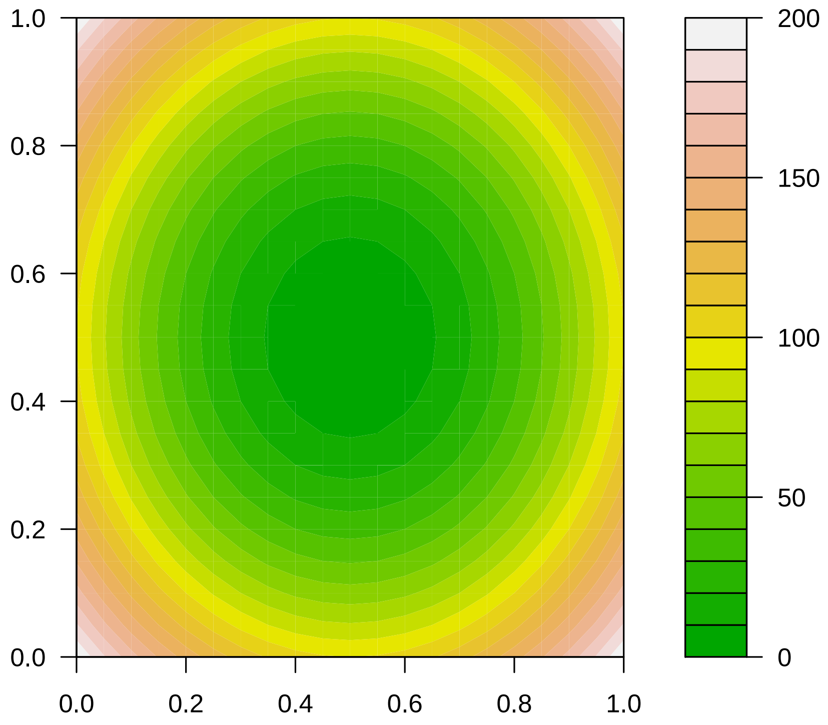
Basic 3d plotting commands are shown below, for additional possibilities see contributed packages.

Density plot: image and contours

```
> x <- (-10:10)
> y <- x
> z <- outer(x^2,y^2,"+") # perform add on all pairs
> image(z)
> contour(z,add=TRUE)
```

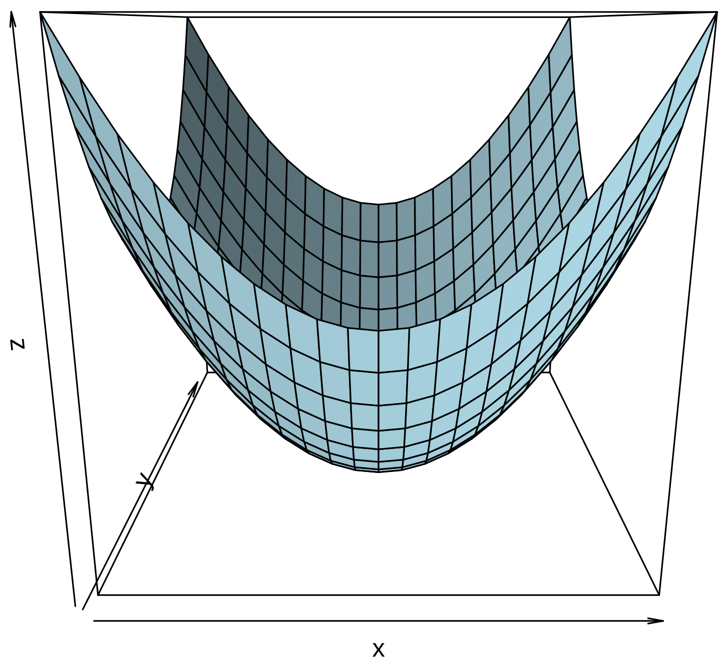


```
> filled.contour(z, color = terrain.colors)
```



Perpective plot

```
> persp(x, y, z, col = "lightblue", shade = 0.25)
```

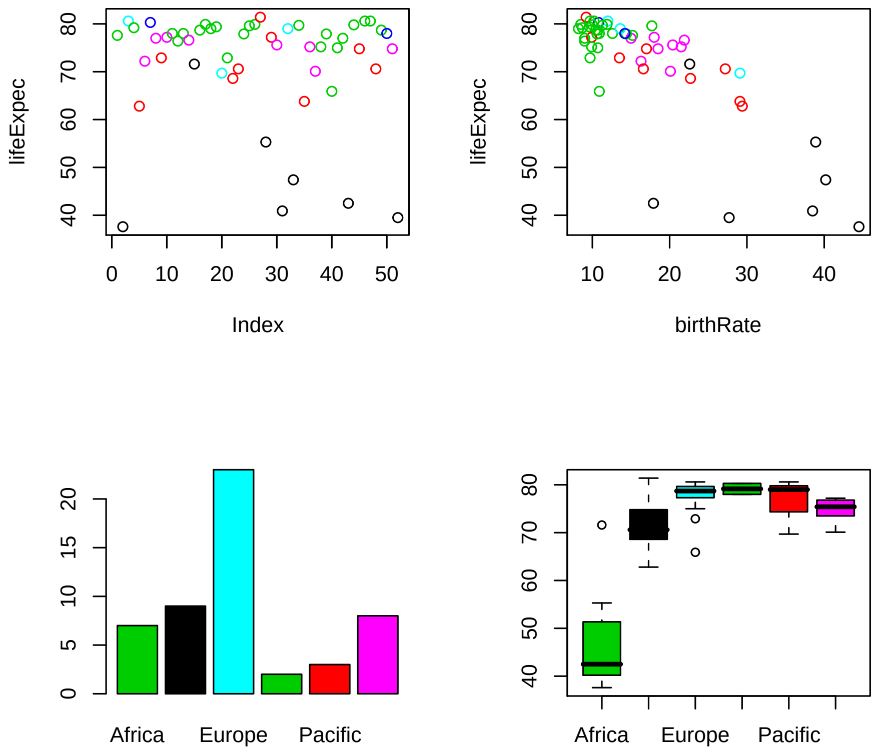


9.3 High level plotting commands

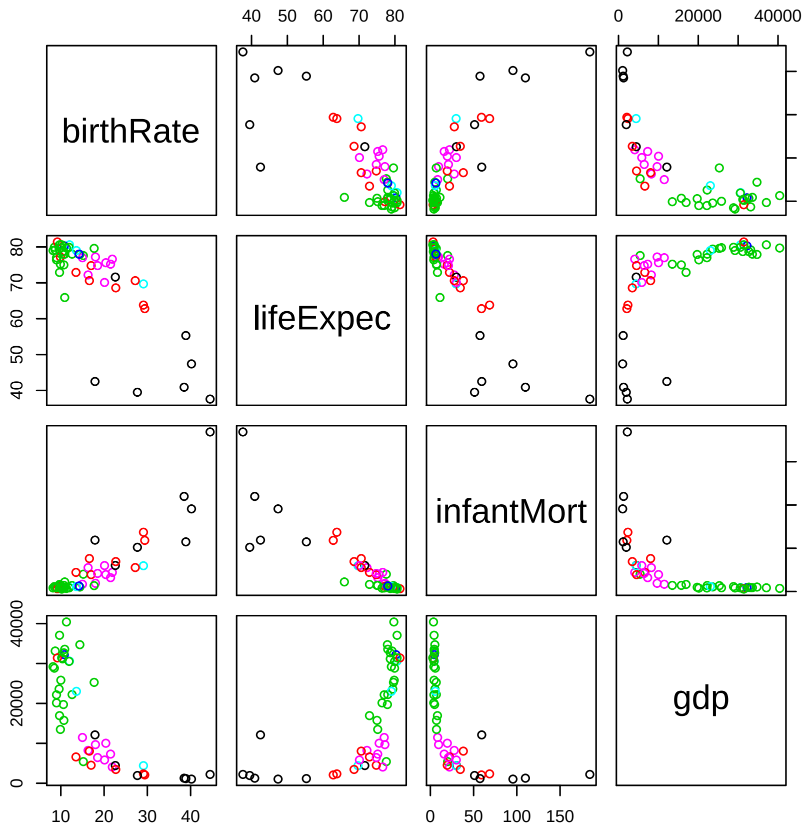
In previous section we used one of high level plotting commands. The most important feature of them is that they set up a new graph with new coordinate system.

We will try some graphs with variables from `sData`:

```
> names(sData)
 [1] "country"       "continent"     "infantMort"    "lifeExpec"
 [5] "population"    "birthRate"     "nPhysicians"   "physicians"
 [9] "gdp"           "internetUsers" "tvSets"
> attach(sData)
        The following object(s) are masked from sData ( position 3 ) :

         birthRate continent country gdp infantMort internetUsers lifeExpec n
> col = as.numeric(as.factor(continent))
> par(mfrow = c(2, 2))
> plot(lifeExpec, col = col)
> plot(birthRate, lifeExpec, col = col)
> plot(continent, col = col)
> plot(continent, lifeExpec, col = col)
```



```
> plot(~birthRate+lifeExpec+infantMort+gdp,col=col)
```


birthRate
lifeExpec
infantMort
gdp
40
50
60
70
80
0
20000
40000
10
20
30
40
0
50
100
150

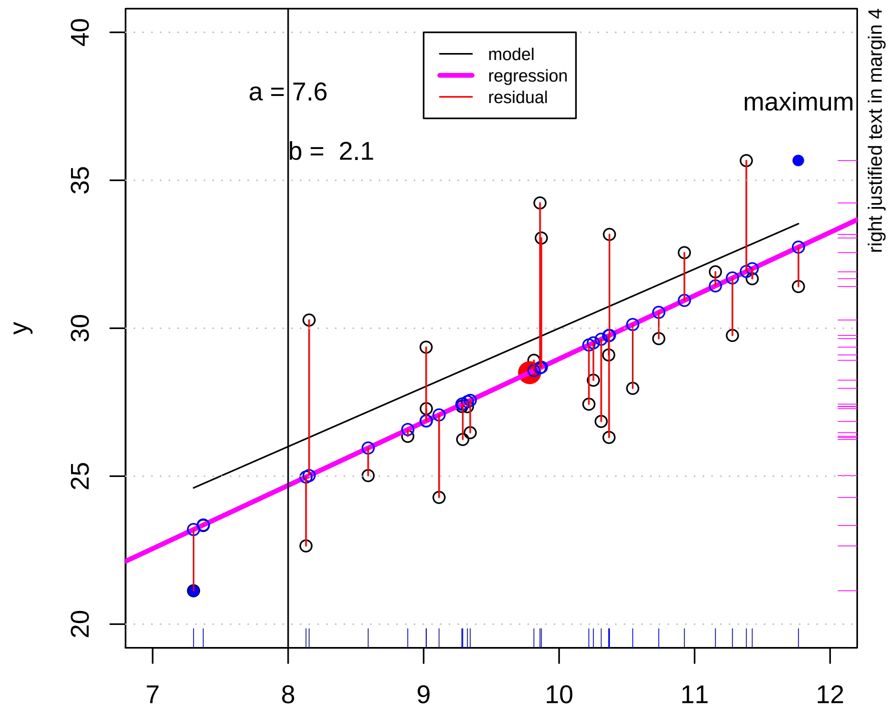
9.4 Low level commands

Low level commands add points, lines, texts, segments, and legend to existing plot.

- `points`
- `lines`
- `segments`
- `text`
- `mtext`
- `abline`
- `rug`
- `legend`

To show the use of plotting commands, we will simulate the situation in which x and y are normally distributed: $X \sim N(10,1)$ and $Y = a + bX + \epsilon$ where $\epsilon \sim N(0,3)$ and make a customized graph.

```
> set.seed(1357)
> n <- 30
> a <- 10
> b <- 2
> x <- rnorm(n,10)
> y <- a+b*x+rnorm(n,0,3)
> ##
> plot(x,y,xlim=c(7,12),ylim=c(20,40))
> points(mean(x),mean(y),pch=16,col="red",cex=2) # add centroid point
> range(x)
[1]  7.301743 11.766331
> range(y)
[1] 21.12839 35.66525
> points(range(x),range(y),pch=16,col="blue")    # mark extreme two points
> lines(range(x),a+b*range(x)) # add model line
> text(max(x),max(y)+2,"maximum")
> mtext("right justified text in margin 4",4,cex=0.75,adj=1)
> ab <- lsfit(x,y)
> abline(ab$coeff,col=6,lwd=3) # add regression line
> abline(v=8) # vertical line at 8
> aEst <- ab$coeff[1]
> bEst <- ab$coeff[2]
> text(8,38,paste("a =",round(aEst,1))) # centred text
> text(8,36,paste("b = ",round(bEst,1)),adj=0) # left justified
> rug(x,side=1,col="blue")
> rug(y,side=4,col=6)
> ## residuals
> yEst <- aEst + bEst *x
> points(x,yEst,col=4)
> segments (x, y, x, yEst,col=2)
> abline(h=seq(20,40,5),col="grey",lty=3) # add grey gridlines
> legend(9,40,col=c("black","magenta","red"),
+            lwd=c(1,3,1),
+            legend=c("model","regression","residual"),
+            cex=0.7)
```



9.5 Plotting parameters (par)

Plotting is controlled by a set of parameters, that can be set by a call to the function `par` or declared as an argument in the plotting function call. The most common parameters are presented, complete list and details is in the `par` help page (`help("par")` or simply `?par`).

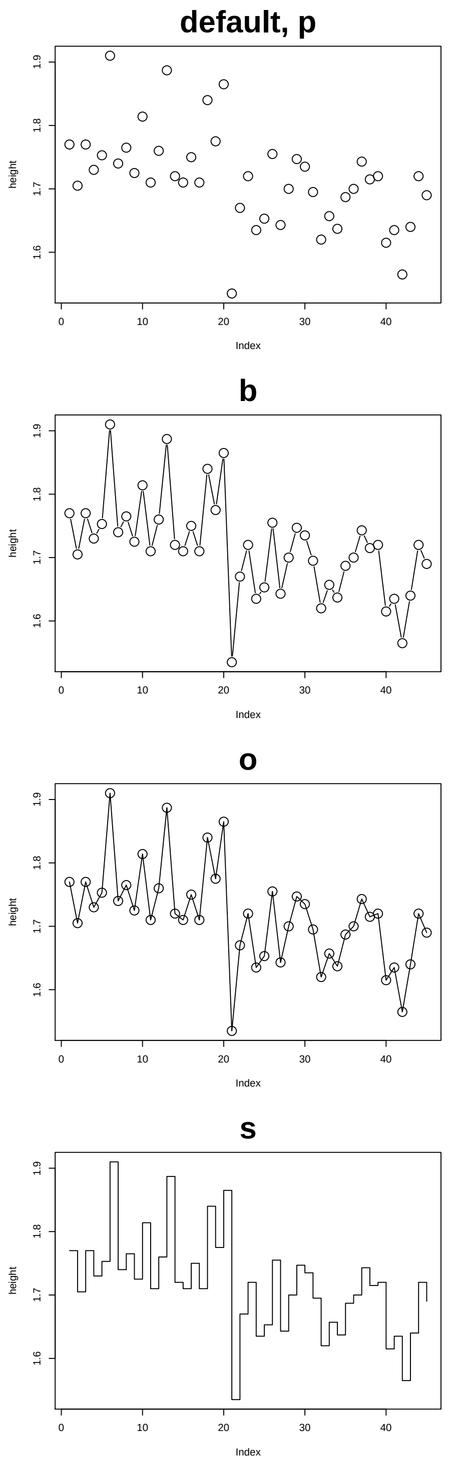
9.5.1 Plot types (type)

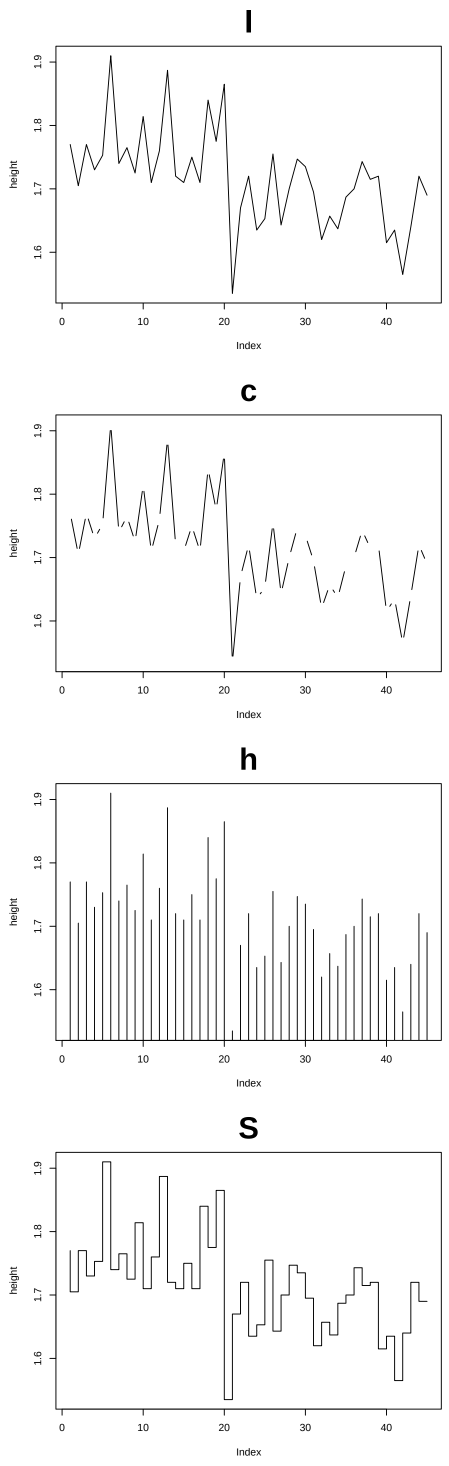
R supports different plotting types:

- ”p” for points
- ”l” for lines
- ”b” for both, points and lines
- ”c” for the lines part of ”b”
- ”o” for both, “overplotted”
- ”h” for “histogram” like (or “high-density”) vertical lines
- ”s” for stair steps
- ”S” for other steps
- ”n” for no plotting (just setting the axes)

In the following code, `cex.main` controls the size of title characters while `cex` controls the size of plotting characters (`characterexpansion`)

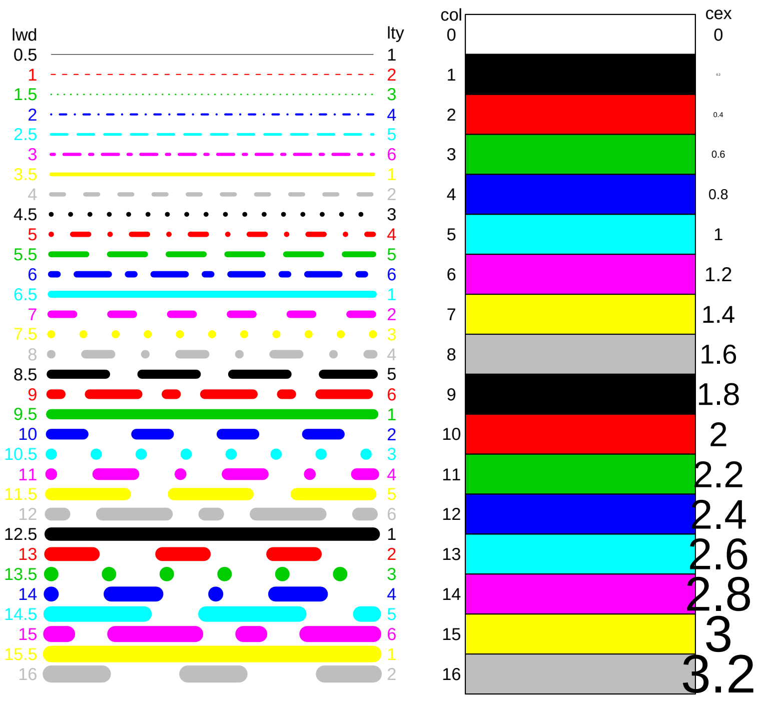
```
> bmiData <- read.table("../dat/bmi.txt", header = TRUE)
> height <- bmiData$height
> oldopt <- options(width = 100)
> oldpar <- par(mfrow = c(4, 2))
> plot(height, main = "default, p", cex.main = 3, cex = 2)
> plot(height, type = "l", main = "l", cex.main = 3)
> plot(height, type = "b", main = "b", cex.main = 3, cex = 2)
> plot(height, type = "c", main = "c", cex.main = 3, cex = 2)
> plot(height, type = "o", main = "o", cex.main = 3, cex = 2)
> plot(height, type = "h", main = "h", cex.main = 3)
> plot(height, type = "s", main = "s", cex.main = 3)
> plot(height, type = "S", main = "S", cex.main = 3)
> options(oldopt)
> par(oldpar)
```


default, p
b
o
s
height
Index
1.6
1.7
1.8
1.9
0
10
20
30
40


l
c
h
S
height
Index

9.5.2 Line type and width (`lty` and `lwd`)

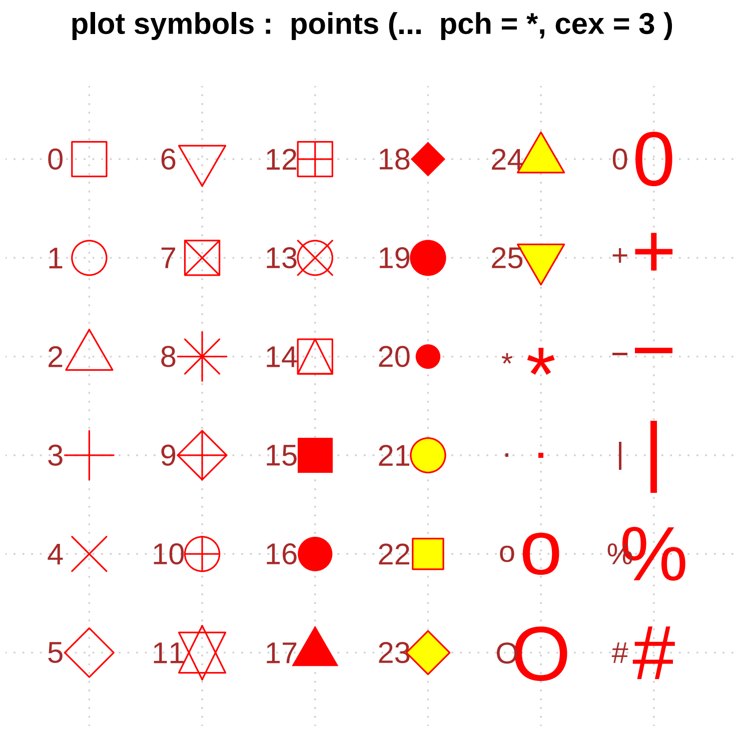
```
> show.lty()
```



9.5.3 Plotting character (`pch`) and size (`cex`)

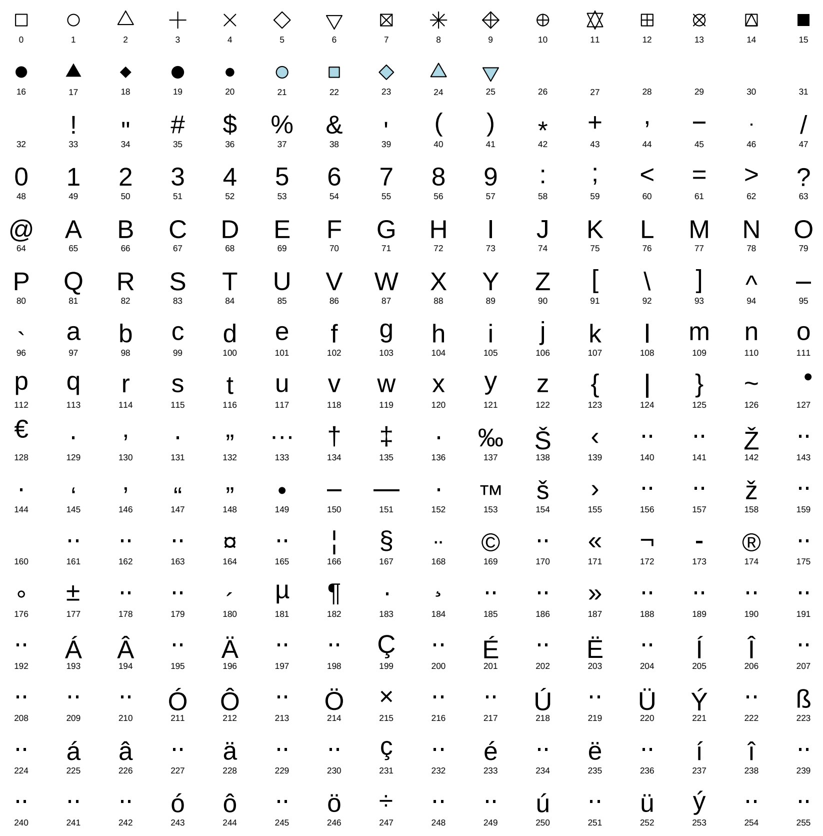
The character expansion (`cex`) parameter controls the size of plotted symbols (characters, `pch`). The effect of `cex` is shown on the figure of line types.

The following graph shows plotting characters an their positioning (from `help(par)`):



Plotting characters, listed by number:

```
> show.pch()
```



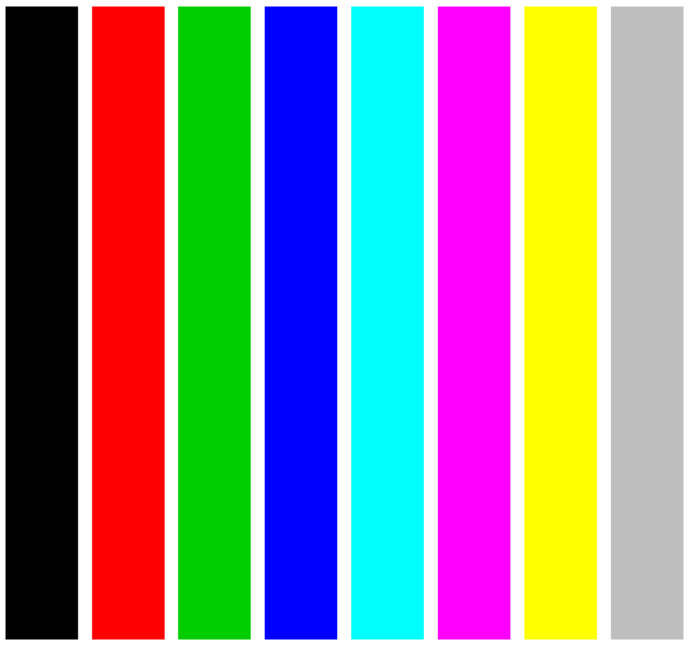
9.5.4 Colors (`col`)

Colors for plotting are passed via the `col` argument as numbers or as color names listed in function `colors`. There are also advanced color definition techniques as `rgb`, `hsv` or `color.brewer`. See `palette` for definition of sets of shading colors. Below is a list of first 10 color names and a table of colors with indices to color names.

Numbers of basic colors are associated by acronyms *RGB* and *CMYK*:

1	2	3	4	5	6	7	8	9	0
black	**R***ed*	**G***reen*	**B***lue*	**C***yan*	**M***agenta*	**Y***ellow*	**G***rey*	*blac***K**	no color

```
> barplot(rep(1, 8), yaxt = "n", col = 1:8, border = FALSE)
> palette()
[1] "black"   "red"     "green3"  "blue"    "cyan"    "magenta" "yellow"
[8] "gray"
```

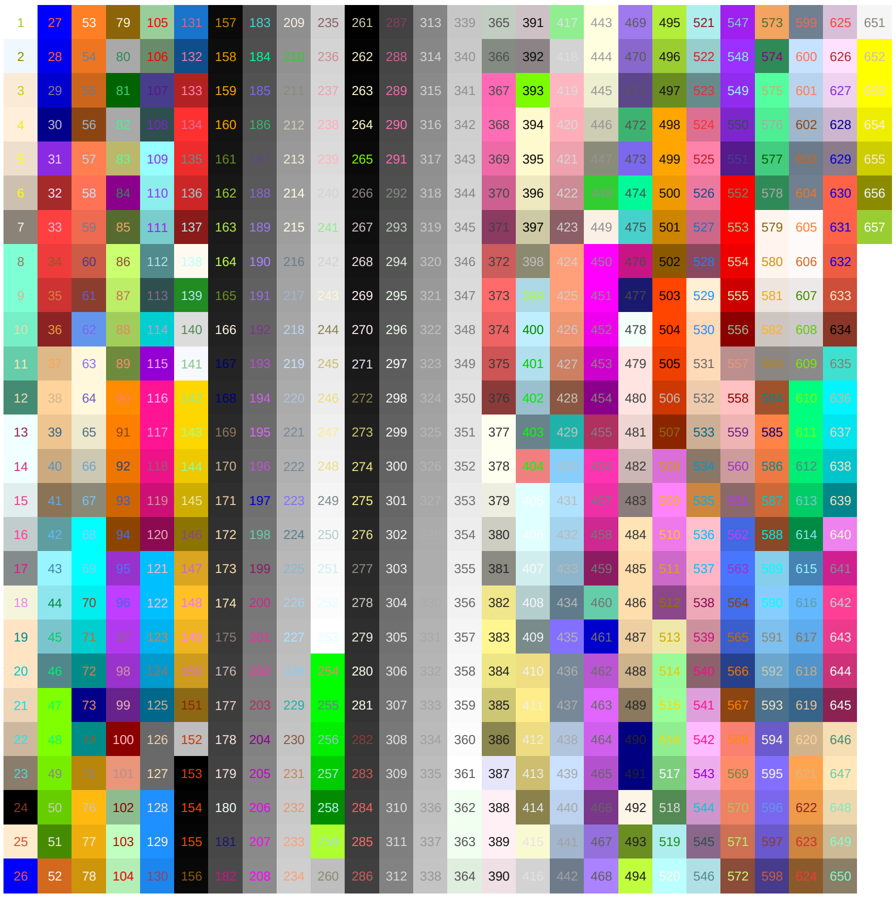


A list of 657 color names is available in `colors()` function. A list of color names that include the word ”red” is listed here.

```
> colors()[grep("red", colors())]
 [1] "darkred"          "indianred"         "indianred1"        "indianred2"
 [5] "indianred3"       "indianred4"        "mediumvioletred"   "orangered"
 [9] "orangered1"       "orangered2"        "orangered3"        "orangered4"
[13] "palevioletred"    "palevioletred1"    "palevioletred2"    "palevioletred3"
[17] "palevioletred4"   "red"               "red1"              "red2"
[21] "red3"             "red4"              "violetred"         "violetred1"
[25] "violetred2"       "violetred3"        "violetred4"
```

Colors and their indices are plotted below:

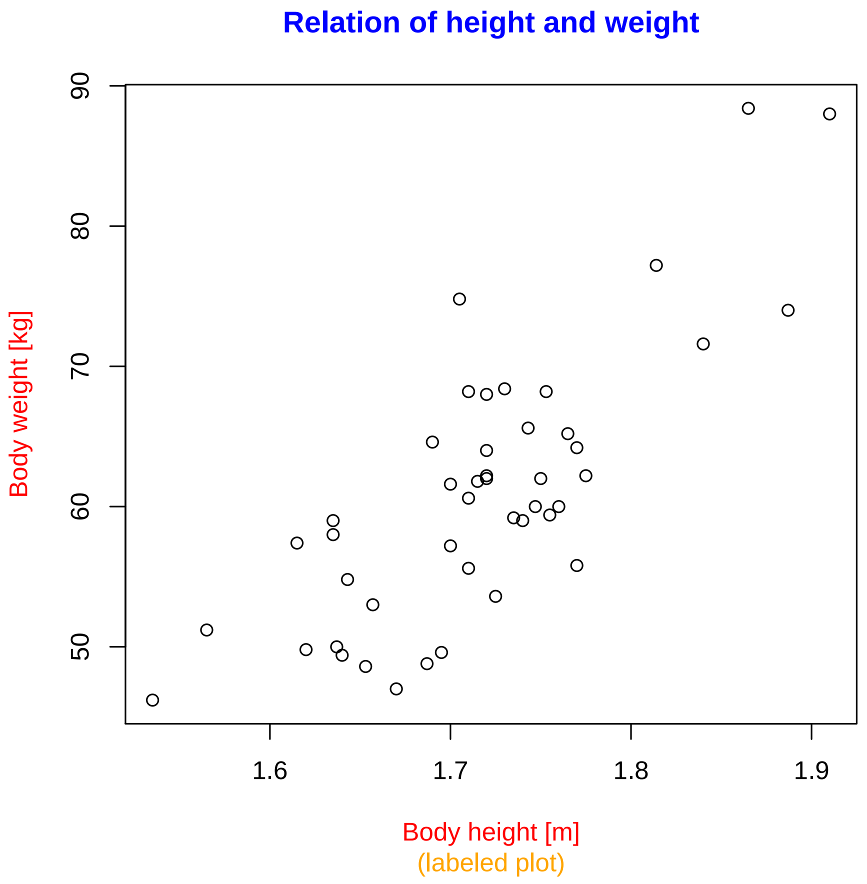
```
> show.colors(cex = 7)
```



9.5.5 Plot and axis labels

To overrule the default axis labeling by the variable names, `xlab` and `ylab` can be used. To put a title and subtitle use arguments `main` and `sub`. You can use `\n` to split the title into lines.

```
> attach(bmiData)
        The following object(s) are masked _by_ .GlobalEnv :

         height
> plot(height, weight,
+         xlab = "Body height [m]",
+         ylab = "Body weight [kg]",
+         main = "Relation of height and weight",
+         sub  = "(labeled plot)",
+         col.lab  = "red",
+         col.main = "blue",
+         col.sub  = "orange")
> detach()
```

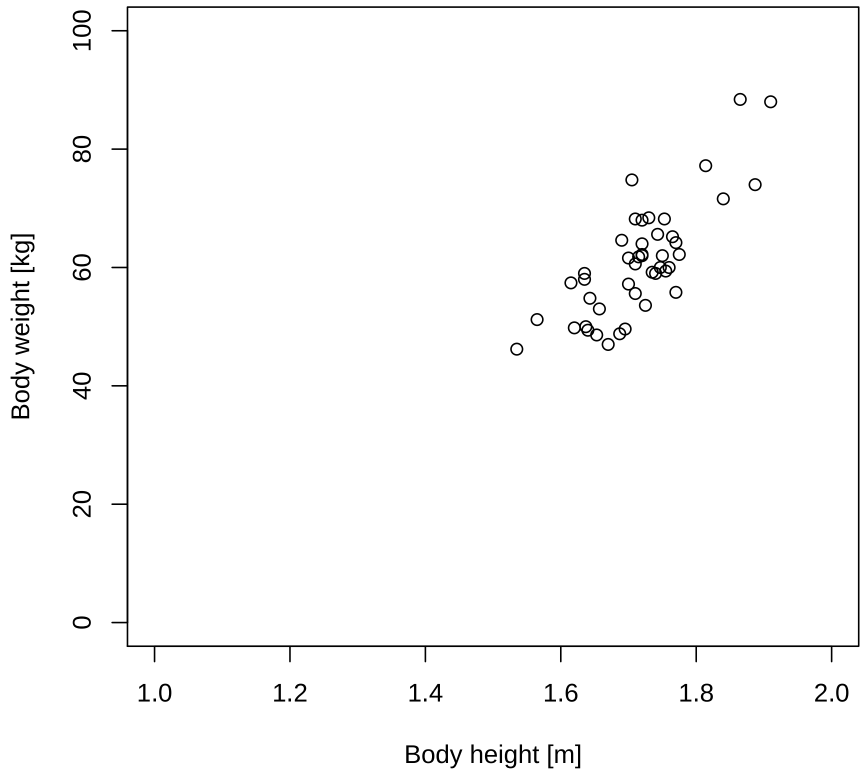


Function `title(...)` can be used for labeling of an existing plot. In addition to `col` and `cex` for plot color and plotting character size (expansion), dotted arguments (as in the preceding example) for labels, main and sub title are available *e. g.* `col.lab` and `cex.lab`.

9.5.6 Axis limits

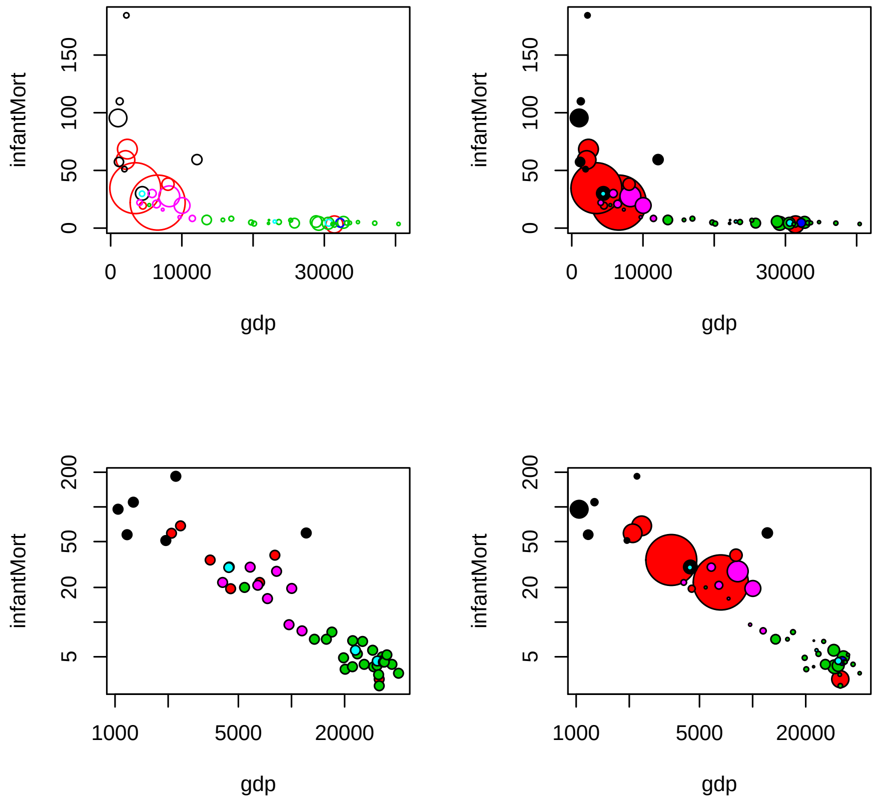
By default, data range is used for axis limits. To override the default, arguments `xlim` and `ylim` can be used:

```
> attach (bmiData)
          The following object(s) are masked _by_ .GlobalEnv :

           height
> plot(height, weight,
+         xlim = c(1, 2),
+         ylim = c(0, 100),
+         xlab = "Body height [m]",
+         ylab = "Body weight [kg]"
+         )
> detach ()
```



9.6 Tuning the graph

```
> ind <- rev(order(sData$population))
> X <- sData[ind, ]
> attach(X)
        The following object(s) are masked from sData ( position 3 ) :

         birthRate continent country gdp infantMort internetUsers lifeExpec nPhy


        The following object(s) are masked from sData ( position 4 ) :

         birthRate continent country gdp infantMort internetUsers lifeExpec nPhy
> oldpar <- par(mfrow = c(2, 2))
> cex <- 5 * sqrt(population/10^9)
> col <- as.numeric(continent)
> plot(gdp, infantMort, cex = cex, col = col)
> bg <- col
> ind <- rev(order(population))
> plot(gdp, infantMort, cex = cex, pch = 21, bg = bg, col = 1)
> plot(gdp, infantMort, cex = 1, pch = 21, bg = bg, col = 1, log = "xy")
> plot(gdp, infantMort, cex = cex, pch = 21, bg = bg, col = 1,
+     log = "xy")
> if (interactive()) identify(gdp, infantMort, log = "xy", country)
> par(oldpar)
> detach()
```



10 Hypothesis testing and statistical modeling

Almost all standard statistical functions and tests are provided in R . For models, *formula expressions* are used.

10.1 Hypothesis testing

You can get a list of available tests in basic packages by typing

```
> apropos("test")
 [1] ".valueClassTest"         "ansari.test"
 [3] "bartlett.test"           "binom.test"
 [5] "Box.test"                "chisq.test"
 [7] "cor.test"                "file_test"
 [9] "fisher.test"             "fligner.test"
[11] "friedman.test"           "kruskal.test"
[13] "ks.test"                 "mantelhaen.test"
[15] "mauchley.test"           "mauchly.test"
[17] "mcnemar.test"            "mood.test"
[19] "oneway.test"             "pairwise.prop.test"
[21] "pairwise.t.test"         "pairwise.wilcox.test"
[23] "power.anova.test"        "power.prop.test"
[25] "power.t.test"            "PP.test"
[27] "prop.test"               "prop.trend.test"
[29] "quade.test"              "shapiro.test"
[31] "t.test"                  "testing"
[33] "testPlatformEquivalence" "testVirtual"
[35] "var.test"                "wilcox.test"
```

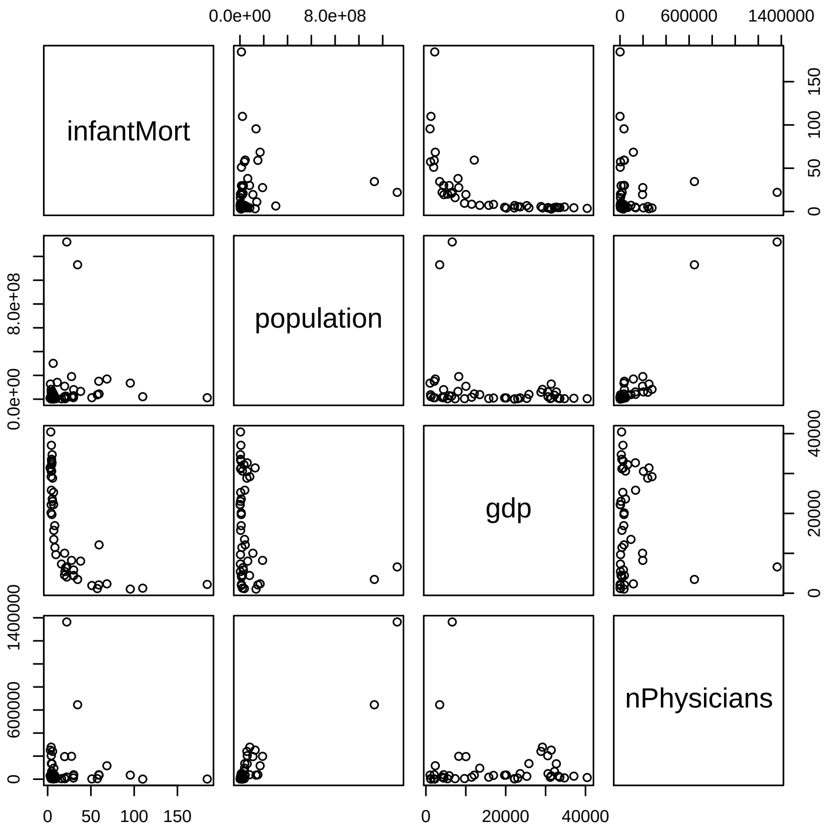
In help pages for specific tests are the details of use and examples. We used `t.test` in the first chapter.

10.2 Statistical models

Modern statistical models are provided in separate packages, many of them can be found in packages **Hmisc** and **MASS**.

Analysis of variance (`aov`) and general linear models (`glm`), linear models (`lm` and `lsfit` are provided in basic installation. We used `lm` and `aov` in the analysis of `BMI`.

```
> dimnames(sData)[[1]] <- sData$country
> attach(sData)
> R <- cor(sData[, -(1:2)], use = "pairwise.complete")
> round(R, 2)
              infantMort lifeExpec population birthRate nPhysicians physician
infantMort          1.00     -0.88       0.06      0.87       -0.06       -0.6
lifeExpec          -0.88      1.00      -0.05     -0.80        0.08        0.7
population          0.06     -0.05       1.00      0.05        0.93       -0.2
birthRate           0.87     -0.80       0.05      1.00       -0.11       -0.8
nPhysicians        -0.06      0.08       0.93     -0.11        1.00       -0.0
physicians         -0.69      0.71      -0.26     -0.80       -0.08        1.0
gdp                -0.62      0.63      -0.24     -0.74       -0.09        0.8
internetUsers      -0.58      0.59      -0.25     -0.68       -0.12        0.6
tvSets             -0.64      0.64      -0.08     -0.77        0.06        0.7
                gdp internetUsers tvSets
infantMort    -0.62         -0.58  -0.64
lifeExpec      0.63          0.59   0.64
population    -0.24         -0.25  -0.08
birthRate     -0.74         -0.68  -0.77
nPhysicians   -0.09         -0.12   0.06
physicians     0.83          0.69   0.77
gdp            1.00          0.92   0.90
internetUsers  0.92          1.00   0.89
tvSets         0.90          0.89   1.00
> lmFit = lm(infantMort ~ population + gdp + nPhysicians)
> lmFit

Call:
lm(formula = infantMort ~ population + gdp + nPhysicians)

Coefficients:
(Intercept)   population          gdp  nPhysicians
  5.535e+01    1.873e-08   -1.704e-03   -3.718e-05
> summary(lmFit)

Call:
lm(formula = infantMort ~ population + gdp + nPhysicians)

Residuals:
    Min      1Q  Median      3Q     Max
-29.234 -15.919  -1.932   7.323 132.632

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  5.535e+01  7.798e+00   7.097 1.06e-08 ***
population   1.873e-08  4.952e-08   0.378    0.707
gdp         -1.704e-03  3.738e-04  -4.559 4.39e-05 ***
nPhysicians -3.718e-05  5.416e-05  -0.686    0.496
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 27.79 on 42 degrees of freedom
  (6 observations deleted due to missingness)
Multiple R-squared: 0.4045,        Adjusted R-squared: 0.362
F-statistic: 9.509 on 3 and 42 DF,  p-value: 6.474e-05
> pairs(infantMort ~ population + gdp + nPhysicians)
> detach()
```



Try the log-transformation:

```
> lsData <- log(sData[, -(1:2)])
> attach(lsData)
> R <- cor(lsData[, -(1:2)], use = "pairwise.complete")
> round(R, 2)
              population birthRate nPhysicians physicians   gdp internetUsers
population          1.00      0.19        0.75      -0.28 -0.35         -0.33
birthRate           0.19      1.00       -0.38      -0.87 -0.91         -0.84
nPhysicians         0.75     -0.38        1.00       0.43  0.29          0.27
physicians         -0.28     -0.87        0.43       1.00  0.89          0.83
gdp                -0.35     -0.91        0.29       0.89  1.00          0.92
internetUsers      -0.33     -0.84        0.27       0.83  0.92          1.00
tvSets             -0.19     -0.87        0.42       0.91  0.89          0.88
              tvSets
population     -0.19
birthRate      -0.87
nPhysicians     0.42
physicians      0.91
gdp             0.89
internetUsers   0.88
tvSets          1.00
> lmFit = lm(infantMort ~ population + gdp + nPhysicians)
> lmFit

Call:
lm(formula = infantMort ~ population + gdp + nPhysicians)

Coefficients:
(Intercept)   population          gdp  nPhysicians
     6.5819       0.3045      -0.7109      -0.2557
> summary(lmFit)
Call:
lm(formula = infantMort ~ population + gdp + nPhysicians)

Residuals:
     Min       1Q   Median       3Q      Max
-0.66421 -0.21695 -0.03618  0.14591  1.50142

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   6.5819     2.0763   3.170  0.00284 **
population    0.3045     0.1236   2.463  0.01797 *
gdp          -0.7109     0.1232  -5.771 8.47e-07 ***
nPhysicians  -0.2557     0.1141  -2.241  0.03038 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3885 on 42 degrees of freedom
  (6 observations deleted due to missingness)
Multiple R-squared: 0.891,       Adjusted R-squared: 0.8832
F-statistic: 114.4 on 3 and 42 DF,  p-value: < 2.2e-16
> pairs(infantMort ~ population + gdp + nPhysicians)
```

For some models, you can plot informative analytical graphs

```
> plot(lmFit)
```

10.3 Multivariate analysis

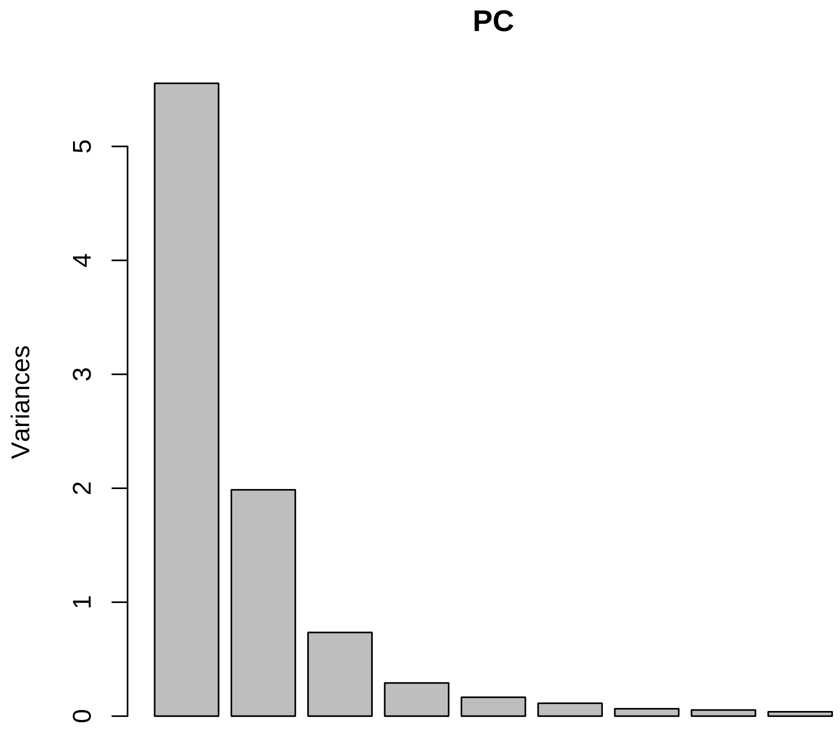
Complete set of functions and operators for linear algebra is provided and very efficient. Operator `%*%` is used for matrix multiplication and function `eigen` for eigenvalue analysis.

All standard multivariate procedures are available.

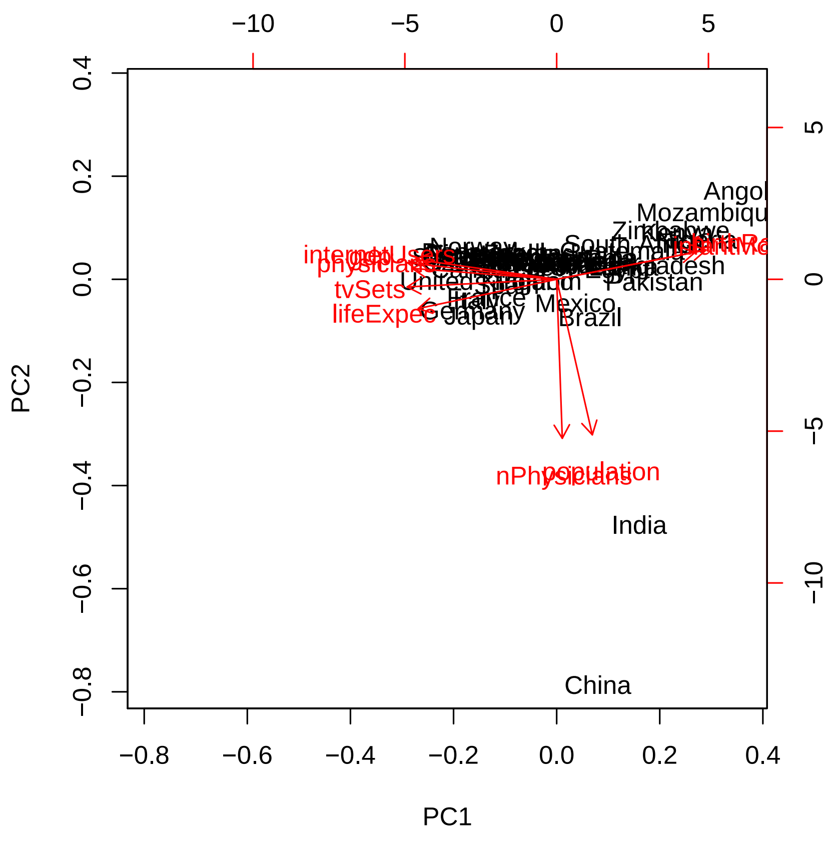
```
> X <- sData[-which(is.na(sData), arr.ind = TRUE)[, 1], ]
> PC <- prcomp(X[, -(1:2)], scale = TRUE)
> PC

Standard deviations:
[1] 2.3566503 1.4093063 0.8568253 0.5394988 0.4067815 0.3360706 0.2541269
[8] 0.2316664 0.1954032

Rotation:
                     PC1         PC2          PC3         PC4         PC5
infantMort    0.36055526  0.11772491 -0.507415576  0.24157403 -0.24082941
lifeExpec    -0.36046771 -0.12728796  0.444210090 -0.21332734 -0.67124972
population    0.09287870 -0.67709770 -0.118413259 -0.02714552 -0.11503028
birthRate     0.38909606  0.12747680 -0.218203284 -0.09280392 -0.62439138
nPhysicians   0.01465474 -0.69222156 -0.169854023  0.10083019  0.03504907
physicians   -0.37624408  0.04755503  0.003861519  0.81533348 -0.20365274
gdp          -0.38839773  0.07179842 -0.377203360  0.05937262 -0.12356898
internetUsers -0.36872737  0.08683875 -0.439835119 -0.44544110 -0.08706348
tvSets       -0.38917053 -0.03314868 -0.344920061 -0.09952732  0.15091795
                    PC6         PC7         PC8          PC9
infantMort   -0.2799792  0.63292712 -0.03872596 -0.042761930
lifeExpec    -0.2566611  0.30183790  0.05809919  0.009843672
population    0.3547632  0.05696294 -0.05578879 -0.610497831
birthRate    -0.0022118 -0.62084916  0.01915196  0.020945617
nPhysicians  -0.2355593 -0.10039528  0.02006753  0.643738205
physicians    0.1052340 -0.15627147 -0.33682124 -0.031262739
gdp           0.3537979  0.04411953  0.73914694  0.097250913
internetUsers 0.2871998  0.08337086 -0.57368423  0.196210115
tvSets       -0.6750229 -0.27604615  0.04839986 -0.401989818
```

```
> summary(PC)

Importance of components:
                         PC1   PC2    PC3    PC4    PC5    PC6     PC7     PC8
Standard deviation     2.357 1.409 0.8568 0.5395 0.4068 0.3361 0.25413 0.23167
Proportion of Variance 0.617 0.221 0.0816 0.0323 0.0184 0.0126 0.00718 0.00596
Cumulative Proportion  0.617 0.838 0.9193 0.9517 0.9701 0.9826 0.98979 0.99576
                           PC9
Standard deviation     0.19540
Proportion of Variance 0.00424
Cumulative Proportion  1.00000
```

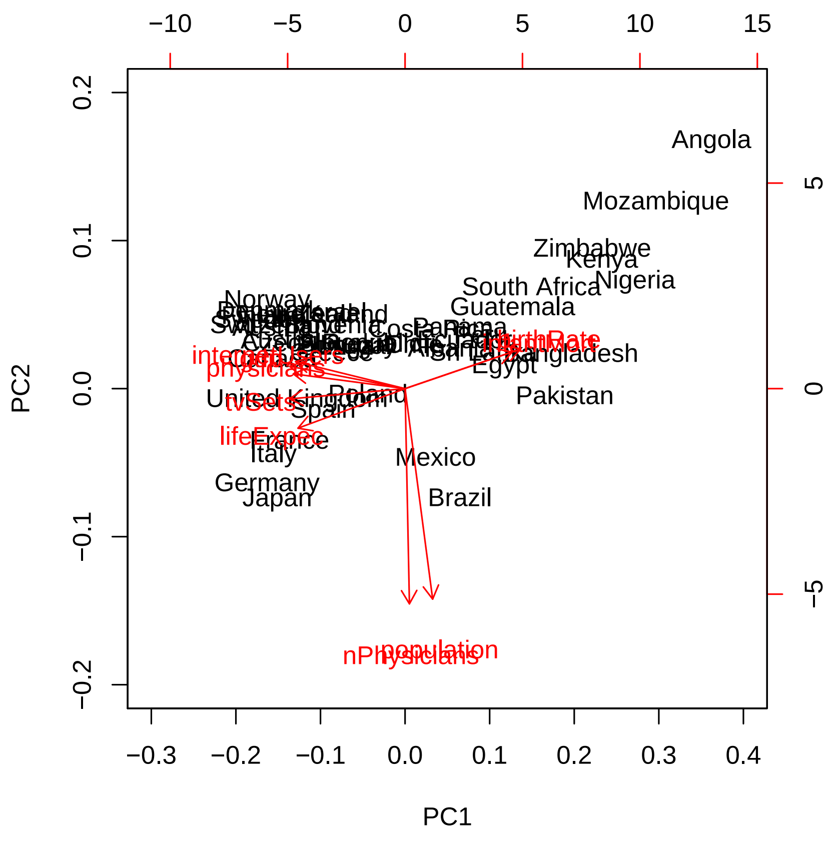
```
> plot(PC)
```



```
> biplot(PC)
```



```
> biplot(PC, xlim = c(-0.3, 0.4), ylim = c(-0.2, 0.2))
```


Angola
Mozambique
Zimbabwe
Kenya
Nigeria
South Africa
Guatemala
Norway
Panama
Bangladesh
Egypt
Pakistan
United Kingdom
Poland
Spain
France
Italy
Mexico
Germany
Japan
Brazil
birthRate
internetUsers
physicians
tvSets
lifeExpec
population
nPhysicians
PC1
PC2

A Practicals

A.1 Day 1

R environment

1. Browse to http://ablejec.nib.si/R/ECPR and download I2R.ZIP to directory <HOMEDIR>/ECPR you previously made. Unzip the downloaded file in the same directory.
2. start R and get familiar with the environment
3. change the working directory to <HOMEDIR>/ECPR/WD
4. Find help documentation for functions used in BMI analysis. Read documentation and try the examples
5. Browse http://www.r-project.org and CRAN. Go through the list of packages and look for some interesting packages. Read the descriptions and documentation. Look also at 'CRAN Task Views'.

R in action

1. calculate your BMI and determine your BMI code
2. Go through the material presented today, try the examples. Ask TA's about anything confusing.
3. Use **File | Open script** to open ../doc/I2R-bmi.R Use `CTRL-R` to step over the lines, try to predict what line does and check the results.

A.2 Day 2

1. Write a function that will add two numbers
2. Function `sd` returns the sample standard deviation. Write a function, that will 'correct' it to the population standard deviation (multiply by $(n-1)/n$)

A.3 Readings

Browse the R web pages `http://www.r-project.org`

Have a look at "R FAQs"

There is a section Windows Features

B Libraries and functions

B.1 Libraries

```
> library(Hmisc)
> library(xlsReadWrite)
> library(foreign)
> options(width = 80)
```

B.2 Functions

A collection of custom made functions is provided here.

B.2.1 `catln`

Concatenate and type, go to next line

```
> catln <- function(...) cat(..., "\n")
> catln("Text1", 1234, "Text2")

Text1 1234 Text2
```

B.2.2 `write.xls`

Write data frame to Excel file. It is a `write.table` function, with defaults that might be more practical.

```
> write.xls <- function(x, file = NULL, row.names = NULL, col.names = NULL,
+     sep = "\t", verbose = FALSE, ...) {
+     xName <- deparse(substitute(x))
+     if (is.null(file))
+         file <- paste(xName, "xls", sep = ".", collapse = "")
+     if (is.null(row.names))
+         row.names <- !(row.names(x)[1] == "1")
+     if (is.null(col.names)) {
+         if (row.names)
+             col.names <- NA
+         else col.names = TRUE
+     }
+     write.table(x, file = file, row.names = row.names, col.names = col.name
+         sep = sep, ...)
+     if (verbose)
+         cat(xName, "written to ", file, "\n")
+ }
> x <- data.frame(a = letters[1:3], x = 1:3, y = sqrt(1:3))
> dimnames(x)[[1]] = LETTERS[1:3]
> write.xls(x, verbose = TRUE)
```

```
x written to  x.xls
```

B.2.3 format.xtab

```
> format.xtab <- function(v1, v2, dataframe, dnn = NULL, fieldwidth = 10,
+     chisq = FALSE) {
+     if (!missing(v1) && !missing(v2)) {
+         if (!missing(dataframe))
+             basetab <- table(dataframe[[v1]], dataframe[[v2]])
+         else basetab <- table(v1, v2)
+         btdim <- dim(basetab)
+         row.pc <- col.pc <- vector("numeric", btdim[2])
+         row.sums <- apply(basetab, 1, sum)
+         col.sums <- apply(basetab, 2, sum)
+         row.names <- formatC(rownames(basetab), width = fieldwidth)
+         col.names <- formatC(colnames(basetab), width = fieldwidth)
+         grand.total <- sum(row.sums)
+         rowlabelspace <- paste(rep(" ", nchar(row.names[1])),
+             sep = "", collapse = "")
+         if (is.null(dnn))
+             dnn <- formatC(c(v1, v2), width = fieldwidth)
+         else dnn <- formatC(dnn, width = fieldwidth)
+         cat("Crosstabulation of", dnn[1], "by", dnn[2], "\n")
+         cat(rowlabelspace, dnn[2], "\n")
+         cat(rowlabelspace, col.names, "\n")
+         cat(dnn[1], "\n")
+         for (i in 1:btdim[1]) {
+             row.pc <- ifelse(row.sums[i], 100 * basetab[i, ]/row.sums[i],
+                 0)
+             for (j in 1:btdim[2]) col.pc[j] <- ifelse(col.sums[j],
+                 100 * basetab[i, j]/col.sums[j], 0)
+             cat(row.names[i], formatC(basetab[i, ], width = fieldwidth),
+                 formatC(row.sums[i], width = fieldwidth), "\n")
+             cat(rowlabelspace, formatC(round(row.pc, 2), width = fieldwidth),
+                 formatC(round(100 * row.sums[i]/grand.total,
+                   2), width = fieldwidth), "\n")
+             cat(rowlabelspace, formatC(round(col.pc, 2), width = fieldwidth),
+                 "\n\n")
+         }
+         cat(rowlabelspace, formatC(col.sums, width = fieldwidth),
+             formatC(grand.total, width = fieldwidth), "\n")
+         cat(rowlabelspace, formatC(round(100 * col.sums/grand.total,
+             2), width = fieldwidth), "\n\n")
+         if (chisq) {
+             chisq.obs <- chisq.test(basetab)
+             cat(names(chisq.obs$statistic), "=", round(chisq.obs$statistic,
+                 3), names(chisq.obs$parameter), "=", chisq.obs$parameter,
+                 "p =", round(chisq.obs$p.value, 5), "\n\n")
+         }
+         invisible(basetab)
+     }
+     else cat("Usage: ", "format.xtab(v1, v2,", "dataframe[, dnn=NULL, fieldwid
+ }
```

B.2.4 xtab

```
> xtab <- function(formula, dataframe, dnn = NULL, fieldwidth = 10,
+     chisq = FALSE) {
+     if (!missing(formula) && !missing(dataframe)) {
+         xt <- as.character(attr(terms(formula), "variables")[-1])
+         nxt <- length(xt)
+         if (nxt > 2) {
+             by.factor <- as.factor(dataframe[[xt[nxt]]])
+             factor.levels <- levels(by.factor)
+             nlevels <- length(factor.levels)
+             brkstats <- as.list(rep(0, nlevels))
+             names(brkstats) <- factor.levels
+             for (i in 1:nlevels) {
+                 currentdata <- subset(dataframe, by.factor ==
+                   factor.levels[i])
+                 currentcount <- length(currentdata[[nxt]])
+                 totalcount <- length(dataframe[[nxt]])
+                 cat("\nCount for", xt[nxt], "=", factor.levels[i],
+                   "is", currentcount, "(", round(100 * currentcount/totalcount
+                     1), "%)\n\n")
+                 rightside <- ifelse(nxt > 3, paste(xt[2:(nxt -
+                   1)], sep = "", collapse = "+"), xt[2])
+                 next.formula <- as.formula(paste(xt[1], rightside,
+                   sep = "~", collapse = ""))
+                 xtab(next.formula, currentdata, dnn, fieldwidth,
+                   chisq)
+             }
+         }
+         else format.xtab(xt[1], xt[2], dataframe, dnn, fieldwidth,
+             chisq)
+     }
+     else cat("Usage:", " xtab(formula, dataframe[, dnn=NULL, ",
+         "fieldwidth = 10, chisq = FALSE])\n")
+ }
```

B.2.5 dstats

```
> dstats <- function(x, indices = 1:dim(x)[2]) {
+     if (!missing(x)) {
+         if (is.data.frame(x) | is.matrix(x)) {
+             indices <- sapply(x[indices], is.numeric)
+             d1 <- sapply(x[, indices], mean, na.rm = TRUE)
+             d2 <- sapply(x[, indices], var, na.rm = TRUE)
+             d3 <- sapply(x[, indices], validn)
+             dstat <- (rbind(d1, d2, d3))
+             rownames(dstat) <- c("Mean", "Variance", "n")
+             class(dstat) <- "dstat"
+             return(dstat)
+         }
+     }
+     cat("Usage: dstats(x, indices=1:dim(x)[2])\n")
+     cat("\twhere x is a data frame or matrix\n")
+ }
```

B.2.6 show.colors

```
> "show.colors" <- function(kaj = 1:length(colors()), cex = 4) {
+     n <- length(kaj)
+     nc <- sqrt(n) + 1
+     colid <- matrix(kaj, ncol = nc)
+     y <- outer(nc:1, rep(1, nc))
+     x <- outer(rep(1, nc), 1:nc)
+     kajc <- c(kaj, rep(1, nc * nc - n))
+     oldpar <- par(mar = c(0, 0, 0, 0))
+     plot(x, y, col = colors()[kajc], pch = 15, cex = cex, axes = F,
+         xlab = "", ylab = "")
+     kajc <- c(rev(kaj), rep(1, nc * nc - n))
+     text(x, y, kaj, col = colors()[kajc], cex = cex/6)
+     par(oldpar)
+ }
```

B.2.7 show.pch

```
> "show.pch" <- function() {
+     oldpar <- par(mar = c(1, 1, 1, 1) * 0.5)
+     on.exit(par(oldpar))
+     plot(c(0, 16), c(0, 32), type = "n", axes = F, xlab = "",
+         ylab = "")
+     for (i in 0:255) {
+         points(i%%16, 32 - (i%/%16) * 2, pch = i, cex = 1.5,
+             bg = "lightblue")
+         text(i%%16, 32 - (i%/%16) * 2 - 0.75, as.character(i),
+             cex = 0.5)
+     }
+ }
```

B.2.8 show.lty

```
> show.lty <- function() {
+     plot(c(0, 16), c(0, 34), type = "n", axes = F, xlab = "",
+         ylab = "")
+     text(0.7, 33, "lwd", adj = 1)
+     text(8.3, 33, "lty", adj = 0)
+     for (i in 1:32) {
+         segments(1, 33 - i, 8, 33 - i, col = i, lwd = i/2, lty = i)
+         text(0.7, 33 - i, i/2, adj = 1, col = i)
+         text(8.3, 33 - i, (i - 1)%%6 + 1, adj = 0, col = i)
+     }
+     text(9.7, 34, "col")
+     text(15.5, 34, "cex")
+     for (i in 0:16) {
+         rect(10, 32 - 2 * i, 15, 32 - 2 * i + 2, col = i, border = T)
+         rect(10, 32 - 2 * i, 15, 32 - 2 * i + 2, col = (i ==
+             0), density <- 0, border = T)
+         text(9.7, 32 - 2 * i + 1, i)
+         text(15.5, 32 - 2 * i + 1, i/5, cex = i/5)
+     }
+ }
```

```
> sessionInfo()
R version 2.8.0 (2008-10-20)
i386-pc-mingw32

locale:
LC_COLLATE=Slovenian_Slovenia.1250;LC_CTYPE=Slovenian_Slovenia.1250;LC_MONETAI

attached base packages:
[1] stats     graphics  grDevices datasets  utils     methods   base

other attached packages:
[1] foreign_0.8-29     xlsReadWrite_1.3.2 Hmisc_3.4-3

loaded via a namespace (and not attached):
[1] cluster_1.11.11 grid_2.8.0      lattice_0.17-15
```

C Data

C.1 Data sources

C.1.1 Statistical Yearbook of RS, Statistical Office of RS

http://www.stat.si/letopis/index_vsebina.asp?poglavje=14&leto=2006&jezik=si

C.1.2 NationMaster

http://www.nationmaster.com/statistics

C.1.3 Technology, Entertainment, Design (TED)

Hans Rosling lecture
http://www.ted.com/tedtalks/tedtalksplayer.cfm?key=hans_rosling

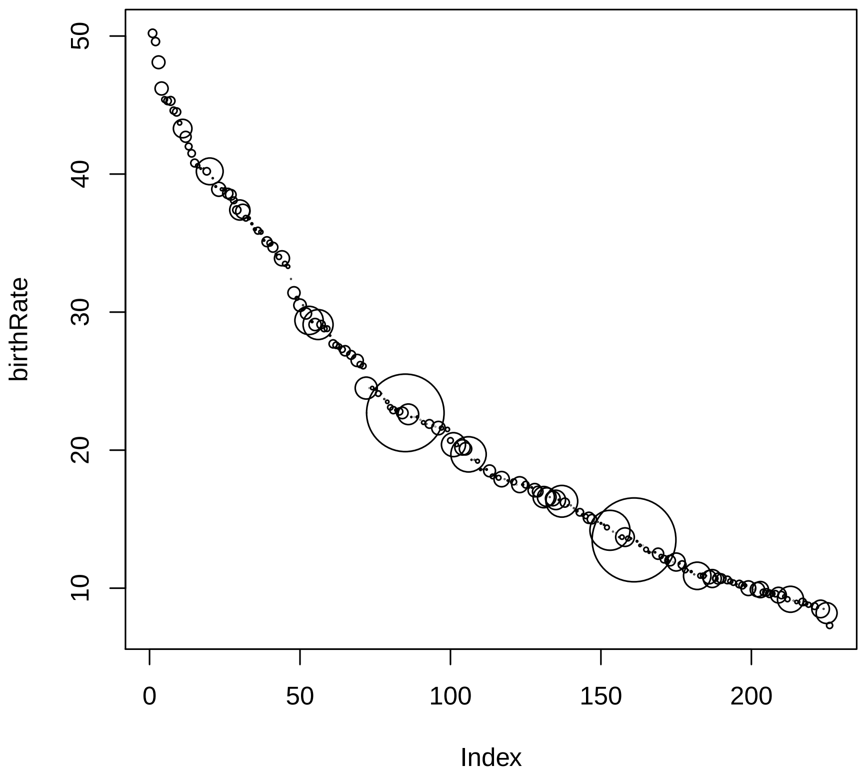
C.1.4 Birth rate by countries

http://www.infoplease.com/ipa/A0934668.html

```
> lfn <- normalizePath("../dat/birthbycountry.xls")
> births <- read.xls(lfn)
> names(births) <- c("country", "population", "birthRate")
> births <- births[rev(order(births$birthRate)), ]
> head(births)
```

```
          country population birthRate
150         Niger   12894865      50.2
126          Mali   11995402      49.6
209        Uganda   30262610      48.1
1     Afghanistan   31889923      46.2
182  Sierra Leone    6144562      45.4
40           Chad   10238807      45.3
```

```
> with(births, plot(birthRate, cex = 2 * sqrt(population/10^8)))
```

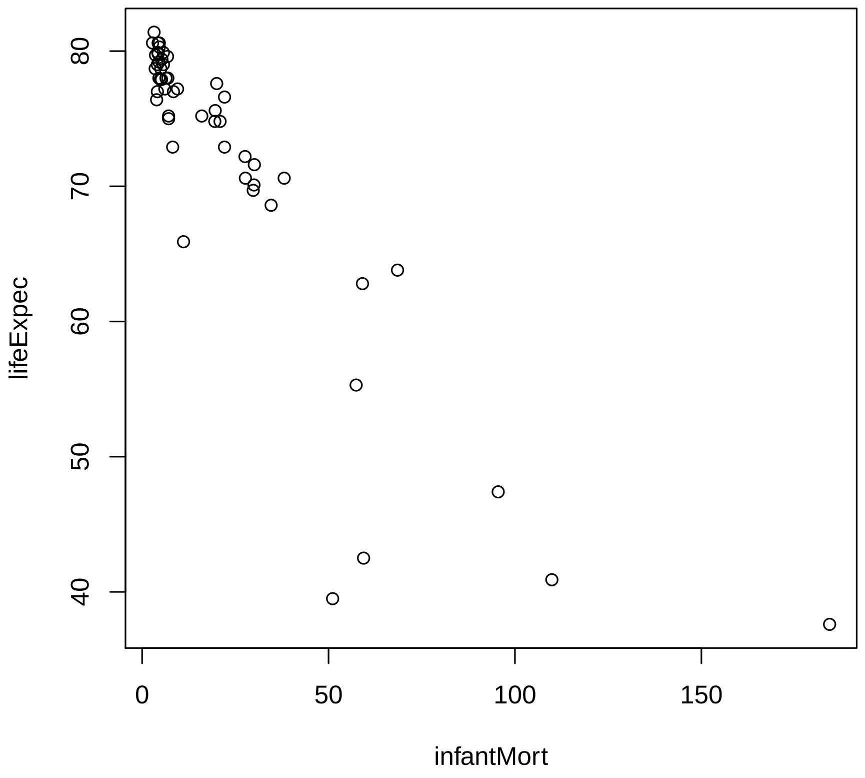


C.2 Data sets

C.2.1 Infant Mortality and Life Expectancy, 2007

http://www.infoplease.com/ipa/A0004393.html

```
> lfn <- normalizePath("../dat/mortality.xls")
> mortality <- read.xls(lfn)
> names(mortality)

[1] "continent"  "country"    "infantMort" "lifeExpec"

> head(mortality)

  continent    country infantMort lifeExpec
1    Europe    Albania       20.0      77.6
2    Africa     Angola      184.4      37.6
3   Pacific  Australia        4.6      80.6
4    Europe    Austria        4.5      79.2
5      Asia Bangladesh       59.1      62.8
6 S America     Brazil       27.6      72.2

> with(mortality, plot(infantMort, lifeExpec))
```



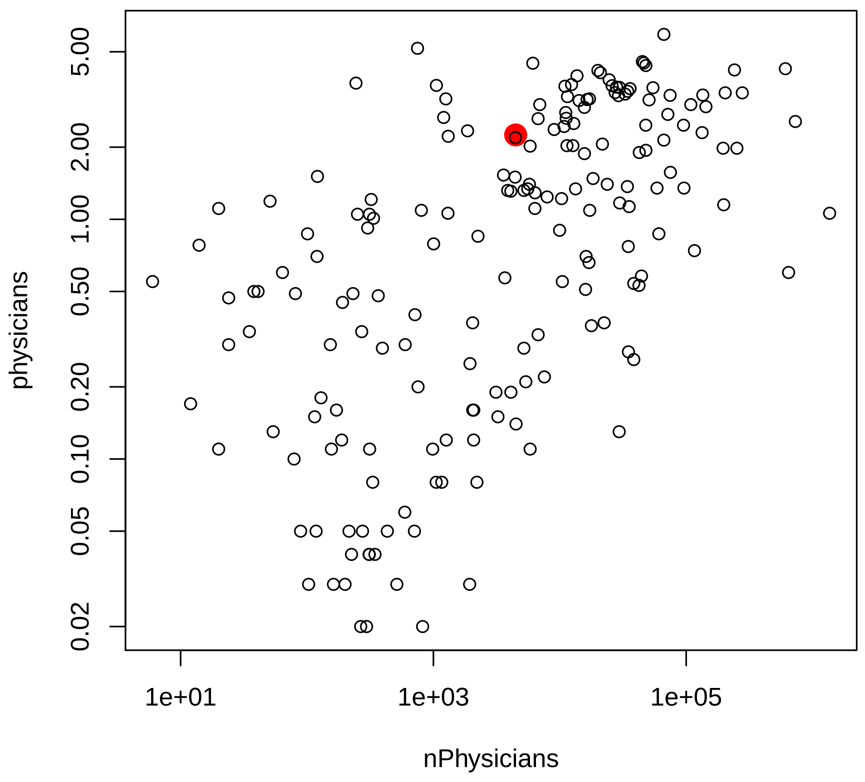
C.2.2 Number of Physicians and Physicians per 1000 inhabitants

http://www.who.int/whosis/en/

```
> lfn <- normalizePath("../dat/physicians.xls")
> physicians <- read.xls(lfn)
> head(physicians)
```

```
                country nPhysicians physicians
1           Afghanistan        4104       0.19
2               Albania        4100       1.31
3               Algeria       35368       1.13
4               Andorra         244       3.70
5                Angola        1165       0.08
6   Antigua and Barbuda          12       0.17
```

```
> with(physicians, {
+     select <- (country == "Slovenia") + 1
+     col = c(1, 2)[select]
+     pch = c(1, 16)[select]
+     plot(nPhysicians, physicians, col = col, pch = pch, cex = select,
+         log = "xy")
+ })
```

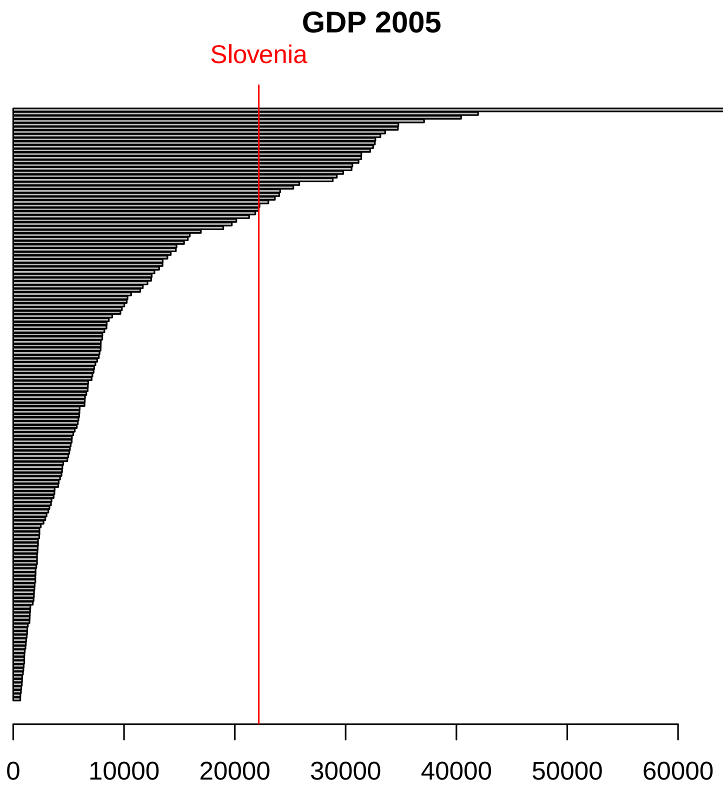


Both axes are logarithmic. Red dot represents Slovenia.

C.2.3 GDP per capita (USD), 2005

http://www.who.int/whosis/en/

```
> lfn <- normalizePath("../dat/gdp2005.xls")
> gdp2005 <- read.xls(lfn)
> head(gdp2005)

              country   gdp
1             Albania  5420
2             Algeria  6770
3              Angola  2210
4 Antigua and Barbuda 11700
5           Argentina 13920
6             Armenia  5060

> with(gdp2005, barplot(gdp[order(gdp)], hori = TRUE, main = "GDP 2005"))
> cName <- "Slovenia"
> slo <- gdp2005[gdp2005$country == cName, ][2]
> abline(v = slo, col = 2)
> text(slo, par("usr")[4] * 1.05, cName, xpd = TRUE, col = "red")
```

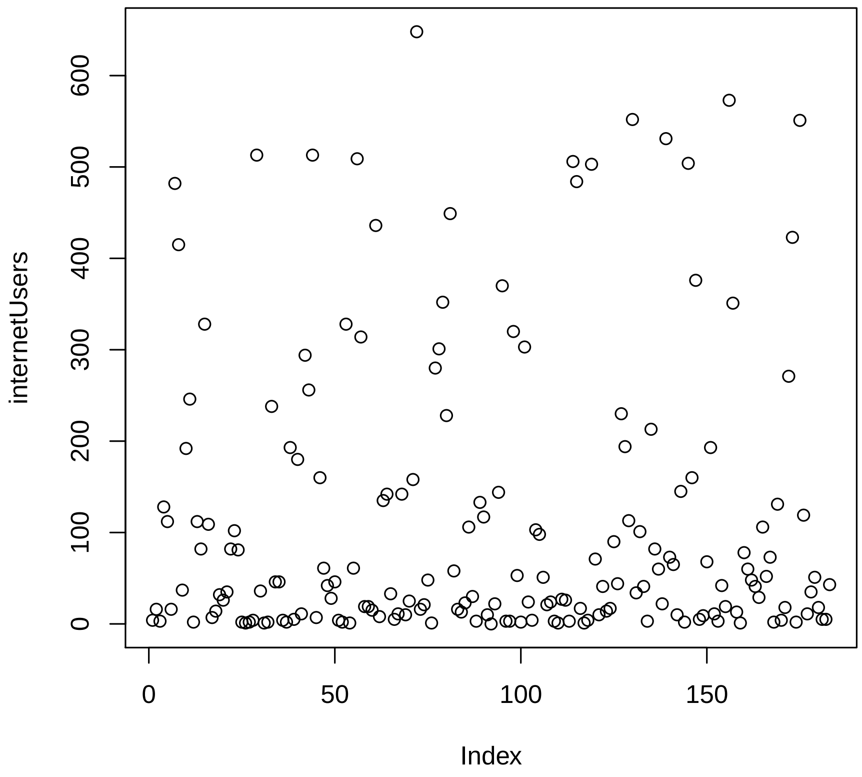


C.2.4 Internet users (per 1000), 2002

http://www.who.int/whosis/en/

```
> lfn <- normalizePath("../dat/internet2002.xls")
> internet2002 <- read.xls(lfn)
> head(internet2002)
              country internetUsers
1             Albania             4
2             Algeria            16
3              Angola             3
4 Antigua and Barbuda           128
5           Argentina           112
6             Armenia            16
```

```
> with(internet2002, plot(internetUsers))
```



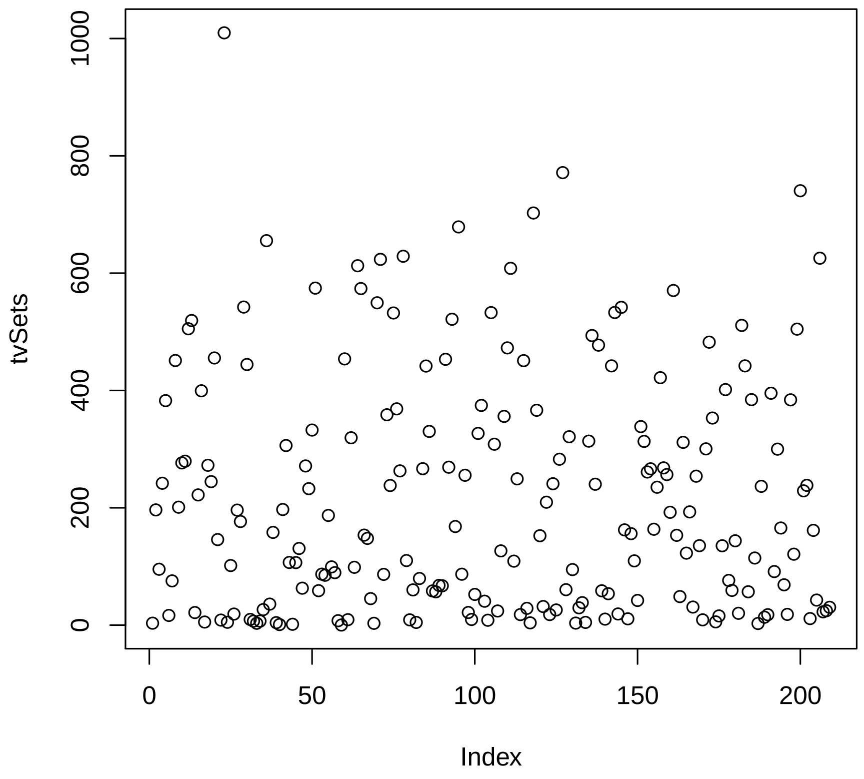
C.2.5 Number of TV sets per 1000

http://www.nationmaster.com/red/graph/med_tel_percap-media-televisions-per-capita&b_desc=1

```
> lfn <- normalizePath("../dat/tv.xls")
> tv <- read.xls(lfn)
> names(tv)[2] <- "tvSets"
> head(tv)
```

```
         country   tvSets
1    Afghanistan   3.3400
2        Albania 196.4640
3        Algeria  95.2908
4 American Samoa 241.8760
5        Andorra 382.7130
6         Angola  16.5723
```

```
> with(tv, plot(tvSets))
```



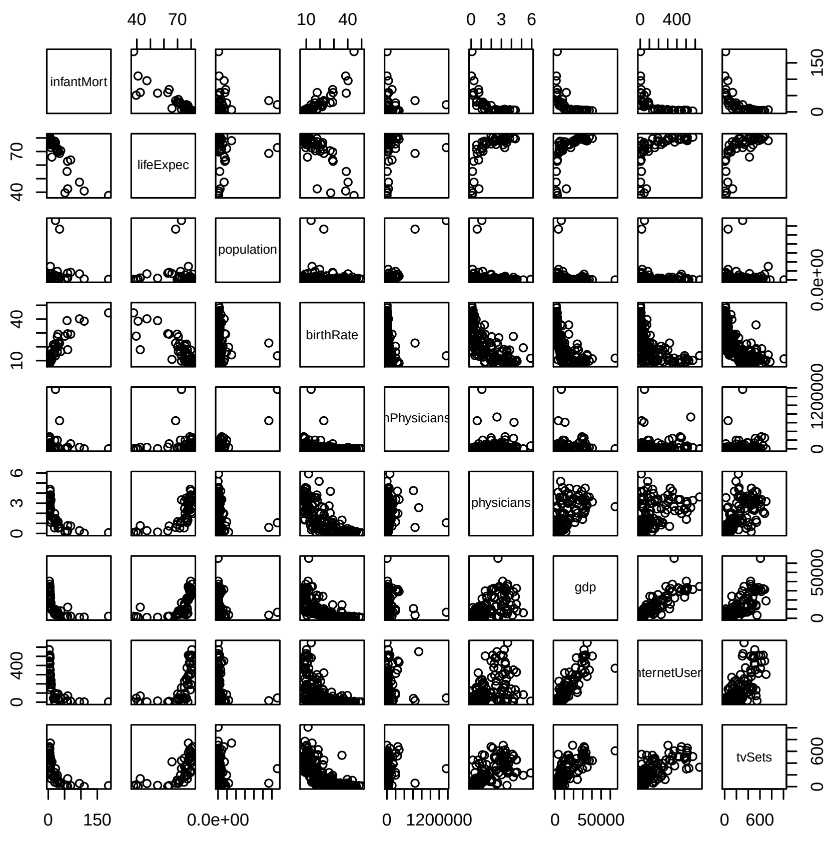
C.3 Merged data set

The data are merged into one dataset, based on the country names.

```
> Data <- merge(mortality, births, by.x = "country", by.y = "country",
+     all = TRUE, )
> Data <- merge(Data, physicians, by.x = "country", by.y = "country",
+     all = TRUE)
> Data <- merge(Data, gdp2005, by.x = "country", by.y = "country",
+     all = TRUE)
> Data <- merge(Data, internet2002, by.x = "country", by.y = "country",
+     all = TRUE)
> Data <- merge(Data, tv, by.x = "country", by.y = "country", all = TRUE)
> Data$continent <- as.factor(Data$continent)
> str(Data)
```

```
'data.frame':       260 obs. of  11 variables:
 $ country      : Factor w/ 260 levels "Albania","Angola",..: 1 2 3 4 5 6 7 8
 $ continent    : Factor w/ 6 levels "Africa","Asia",..: 3 1 5 3 2 6 4 6 2 6
 $ infantMort   : num  20 184.4 4.6 4.5 59.1 ...
 $ lifeExpec    : num  77.6 37.6 80.6 79.2 62.8 72.2 80.3 77 72.9 77.2 ...
 $ population   : num  3.60e+06 1.23e+07 2.04e+07 8.20e+06 1.50e+08 ...
 $ birthRate    : num  15.2 44.5 12 8.7 29.4 16.3 10.8 15 13.5 18 ...
 $ nPhysicians  : num  4100 1165 47875 27413 38485 ...
 $ physicians   : num  1.31 0.08 2.47 3.38 0.26 1.15 2.14 1.09 1.06 1.32 ...
 $ gdp          : num  5420 2210 30610 33140 2090 ...
 $ internetUsers: num  4 3 482 415 2 82 513 238 46 193 ...
 $ tvSets       : num  196.46 16.57 505.23 519.24 5.34 ...
```

```
> pairs(Data[, -c(1, 2)])
> write.xls(Data, "../dat/Data.xls")
```



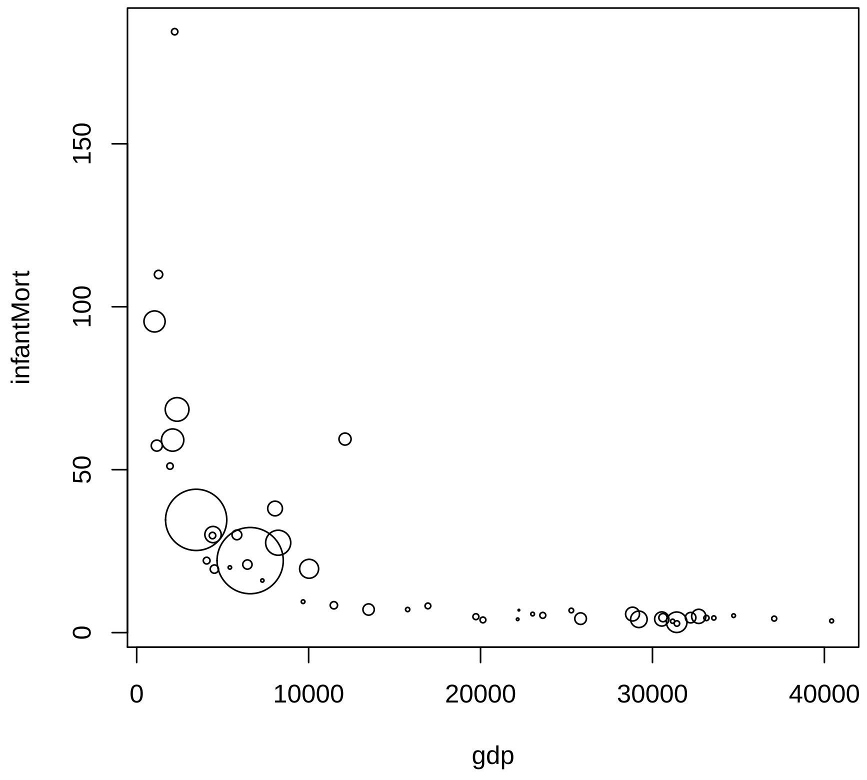
C.4 Small subset

For convenience a smaller subset with complete data on infant mortality is selected.

```
> select <- !is.na(Data$infantMort)
> sData <- Data[select, ]
> str(sData)
```

```
'data.frame':        52 obs. of  11 variables:
 $ country      : Factor w/ 260 levels "Albania","Angola",..: 1 2 3 4 5 6 7 8
 $ continent    : Factor w/ 6 levels "Africa","Asia",..: 3 1 5 3 2 6 4 6 2 6
 $ infantMort   : num  20 184.4 4.6 4.5 59.1 ...
 $ lifeExpec    : num  77.6 37.6 80.6 79.2 62.8 72.2 80.3 77 72.9 77.2 ...
 $ population   : num  3.60e+06 1.23e+07 2.04e+07 8.20e+06 1.50e+08 ...
 $ birthRate    : num  15.2 44.5 12 8.7 29.4 16.3 10.8 15 13.5 18 ...
 $ nPhysicians  : num  4100 1165 47875 27413 38485 ...
 $ physicians   : num  1.31 0.08 2.47 3.38 0.26 1.15 2.14 1.09 1.06 1.32 ...
 $ gdp          : num  5420 2210 30610 33140 2090 ...
 $ internetUsers: num  4 3 482 415 2 82 513 238 46 193 ...
 $ tvSets       : num  196.46 16.57 505.23 519.24 5.34 ...
```

```
> with(sData, plot(gdp, infantMort, cex = 5 * sqrt(population/10^9)))
> write.xls(sData, "../dat/sData.xls")
```



```
> dump(c("sData", "Data"), file = "../dat/I2R-dataset.r")
```

C.5 Interesting packages

prettyR Functions for conventionally formatted descriptive stats, and to format R output as HTML

Hmisc Harrell Miscellaneous

MAAS Modern Applied Statistics with S

SessionInfo

Windows XP (build 2600) Service Pack 3

- R version 2.8.0 (2008-10-20), `i386-pc-mingw32`
- Locale: `LC_COLLATE=Slovenian_Slovenia.1250;LC_CTYPE=Slovenian_Slovenia.1250;LC_MON`
- Base packages: base, datasets, graphics, grDevices, methods, stats, utils
- Other packages: foreign 0.8-29, Hmisc 3.4-3, xlsReadWrite 1.3.2
- Loaded via a namespace (and not attached): cluster 1.11.11, grid 2.8.0, lattice 0.17-15