R for Beginners

International Proteomics Summer School 2024 in Greifswald Katharina J. Hoff[1]

[1] University of Greifswald, Institute for Mathematics and Computer Sciences, Bioinformatics Group; Contact: katharina.hoff@uni-greifswald.de

Contents

1 About R

Most sections in this workshop manuscript are adapted from *Hoff (2005) R-Manual for Biometry.* You find the original at e.g. http://bioinf.uni-greifswald.de/bioinf/katharina/bsc/RManual_English.pdf.

1.1 History

In 1976, John Chambers and his colleagues (Bell Laboratories) began to develop a programming languages called S. The new language should provide the possibility to program with data. Since then, S has been improved continuously.

The S language has been implemented in several ways. The commercial version, S-Plus, has been commonly used for data analysis by scientists.

Ross Ihaka and Robert Gentleman (University of Auckland, New Zealand) started working on an open source implementation that is similar to S. It is called – referring to the initial letters of their Christian names – R. R is covered by the GNU General Public License.

1.2 Advantages of R

- R is free and open source (you can see what's inside!)
- R is platform independent. It may be used on Unix, Linux, Windows and MacOS.
- R test outputs are far more advanced and comprehensive than the result of any Office Program. Confidence intervals, quantiles et cetera are usually automatically calculated along with p-value, degrees of freedom and many other values.
- In comparison to Office Programs, R is more powerful regarding huge data sets and complicated commands (e.g. nested functions).
- The knowledge of mathematical equations for statistical procedures is not an imperative necessity for the evaluation of data with R.
- R is an object oriented programming language. This has many advantages. It is e.g. possible to produce a graph with confidence intervals of an object containing the test output of `simint()` using the single, short command `plot(object.simint)`.
- R offers connectivity to other programs, e.g. to Microsoft Excel.

1.3 Getting and installing R

R is available at CRAN (Comprehensive R Archive Network) on the website http://www.R-project.org. Packages prepared for installation are provided for the operating systems Linux, Windows and Mac OS. Alternatively, you can compile R yourself. But in most cases, that's not necessary.

1.3.1 Installation on Windows

The installation will be started by a double click on the downloaded *.exe file. Simply follow the Installation Wizard. Usually it's okay if you don't change any of the defaults. After installation, you can start the program as you are used to on Windows: click on the desktop icon or on the link in the start menu. You can end R either the way you are used to exit Windows programs or by typing `q()` in the R console.

1.3.2 Installation of Add-on Packages on Windows

The R base system does not include all packages. If you want to use an add-on package, you need to install and to include it, first.

An internet connection is required for the installation of add-ons. You can start the installation process by clicking on the subentry **Install package(s) from CRAN...** in the **Packages** menu. A popup windows opens, presenting a list of available packages. Select the package of your choice and confirm with **OK**. The respective archive will be downloaded, unpacked and installed automatically. Afterwards, R asks the following question: `Delete downloaded files (y/N)?`. You can delete them with `y` (yes) because those files are only the sources for the preliminarily accomplished installation.

For usage of an add-on, you have to load it with the command `library(package name)` into your running R-system.

1.3.3 Installation on Linux

You need root permissions for the standard R installation on Linux. The installation procedure varies dependent on your distribution. On Ubuntu, you can use the graphical Synaptic Package manager to install R, or you type

```
sudo apt-get install R-base
```

On Suse, you should download R-base, first. Then, a click on the *.rpm packages in the Conquerer starts a simple GUI based installation with Yast. Alternatively, you can install R by typing the following command in the Shell:

```
su root
rpm -ih /path/to/package/packagename
```

After a successful installation, R can be called in the terminal window (Shell) by typing `R`. Typing `q()` in the R-Console (= terminal window while R is running) stops the program.

R GUIs (Graphical User Interfaces) are a mixed blessing on Linux. Sometimes, they work. Sometimes, they crash. Personally, I prefer to work with from a shell, instead. If you really need a GUI, I think the `Rcmdr` package might suit you. (You can also use `Rcmdr` on Windows.)

1.3.4 Installation of Add-ons on Linux

The easiest way to install an add-on package on Linux is to type

```
> install.packages(packagename)
```

in the R-console. A pop up windows will ask you to select a suitable mirror. If you fail to install a package this way, try to start R as root, instead (e.g. on Ubuntu `sudo R`).

Remember to include the add-on with `library(package.name)` before usage.

1.4 Documentation and Help System

Entering `help.start()` in the R-Console will open a Browser window on Linux, presenting different manuals and documentations. On Windows, the help pages are opening within the GUI. Handbooks are usually included in the R installation. If they are missing because you excluded them during a user defined installation, an active internet connection will be required.

The command `?function()` or `help(function)` calls for the help of individual functions.

On **Linux**, most help pages are opening within the terminal window. You navigate there with the arrow keys and return to the R command line by typing `q`.

If you do not know the name of the function you are looking for, try searching for a related word:

```
help.search("search.item")
```

It is possible to call examples for a certain function with `example(function)`. The simple entry of a function name will search for this function and return if it exists on the current system.

1.5 Editors

A **text editor** is a computer program for entering, processing and saving plain text. It is reasonable to use an editor while working with R if you want to recall certain preliminarily used functions after a longer period of time without complications.

For the usage of the standard Windows editor or another simple editor, you have to open the editor as well as R and arrange them somehow parallel on the screen. Type your commands into the editor first and copy & paste them into R. Finishing your session, remember to save the editor document as a .txt file somewhere (remember the directory and file name!).

There are many more advanced editors available. Those are able to do much more than only plain text editing. On Windows, WinEdt turned out to be a useful R editor (available at http://www.winedt.com). It can be adjusted in a way that you only have to press a button to hand marked source code over to the R machine. Emacs (available at emacs) combined with ESS (Emacs Speaks Statistics, available at http://ess.r-project.org is offering a similar service which is even platform independent. Both editors provide the user with a colorful highlighting for the source code.

2 First steps

2.1 Handling of the command line

Commands are always typed after > in the R command line. A command is verified by pressing the **ENTER** or **RETURN** key. R is calculating the input and gives an output if available. The arrow keys ↑ and ↓ provide a navigation through previously used commands. **POS1** sets the cursor to the beginning of a line, **END** sets the cursor the end of a line.

Comments are marked with Hash (#).

Blanks are usually ignored. `4 + 7` has the same meaning for R as `4+7`. However, blanks are not allowed to be used inside a command: `x <- 3` ⇒ three is alloted to x, but with a blank within the < and `-` it is getting the meaning " x is smaller than -3?".

Line breaks. If a command is overlapping a single line, `+` will indicate that the same command is continued in the next line. This character does **not** have to be typed! If a command is not complete, there will also show up a `+` in the next line. You have the possibility to complete your command after this sign. In many cases, brackets are missing!

2.2 Pocket calculator

```
> 1 + 2

[1] 3

> 6/3

[1] 2

> 4*5

[1] 20

> log(200)

[1] 5.298317

> 2^2

[1] 4
```

2.3 Assigning variables

Variables are assigned using the `<-` operator or `=`.

```
> my.first.variable <- 23
```

You get the content of a variable by simply typing it:

```
> my.first.variable

[1] 23
```

2.4 Data structure: vector

Vectors are a one dimensional data structures containing only one data type, e.g. numeric or character. Vectors with only one element can be created by simple allocation:

```
> my.first.variable <- 23
> my.first.variable
```

```
[1] 23
```

For creating vectors with multiple elements, the function `c()` ist mostly used to concatenate elements. (There are also other possibilities!)

```
> vec.1 <- c(2,3,4,5,6,3.4)
> vec.1
```

```
[1] 2.0 3.0 4.0 5.0 6.0 3.4
```

```
> vec.2 <- c("a", "b", "c")
> vec.2
```

```
[1] "a" "b" "c"
```

A vector can only contain entries of the same mode, i.e. only numerical entries, or only character, or only boolean, etc.

You get subsets of a vector by using edgy brackets `[]` and indicating the index of the desired vector positions:

```
> vec.1[1:3]
```

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

... returns elements 1, 2 and 3 of `vec.1`.

```
> vec.1[c(1,5,6)]
```

```
[1] 2.0 6.0 3.4
```

... returns elements 1, 5, and 6.

```
> vec.2[c(TRUE, FALSE, TRUE)]
```

```
[1] "a" "c"
```

... returns the first and the third element of `vec.2`.

2.5 Data structure: data frame

The data frame is a two dimensional data structure that might contain different data types in separated columns. It is most frequently used in biometry. All columns must have the same length:

```
> my.frame <- data.frame(group = c("A", "A", "B", "B", "B"), value = c(1, 2, 2, 4, 5))
> my.frame

  group value
1     A     1
2     A     2
3     B     2
4     B     4
5     B     5
```

In most cases, you will import your data into a data frame!

Single columns of the data frame accessible using the $-operator:

```
> my.frame$group

[1] A A B B B
Levels: A B

> my.frame$value

[1] 1 2 2 4 5
```

Columns themselves are vectors and can be treated as such:

```
> my.frame$group[1:3]

[1] A A B
Levels: A B
```

A useful function for subsetting dataframes is `subset()`:

```
> subset(my.frame, my.frame$group=="A")

  group value
1     A     1
2     A     2
```

✎ Exercise 1: vectors and data frames

Animal	Color	Age
rabbit	brown	2
dog	white-black	8
cat	red	7

- Create 3 vectors, one for each column of the above table.
- Create a data frame that looks like the table from those vectors.
- Subset all data for the animal Cat.
- Subset the Age column.

✎ Solution 1: vectors and data frames

```
> animal <- c("rabbit", "dog", "cat")
> color <- c("brown", "white-black", "red")
> age <- c(2,8,7)
> my.data <- data.frame(Animal = animal, Color = color, Age = age)
> my.data

  Animal       Color Age
1 rabbit       brown   2
2    dog white-black   8
3    cat         red   7

> subset(my.data, my.data$Animal=="cat")

  Animal Color Age
3    cat   red   7

> my.data$Age

[1] 2 8 7
```

3 Data import and export

There are several add-on packages that offer connectivity to other software, e.g. Microsoft Excel. However, the easiest way to use external data in R is to save the external data as a text-file! The import command is then:

```
> data <- read.table(file = "/path/to/file/filename.txt", header = TRUE,
+ sep = "\t", dec = ",", stringsAsFactors = FALSE, comment.char = "#")
```

The argument `sep` specifies the separator for the different columns. Tabulator is the default value.

`dec` defines if a dot or a comma is used as decimal sign. The default value in R is the international dot. In most European countries, commas are commonly used.

`stringsAsFactors` specify whether character vectors should automatically be converted into factors (factors are a data type in R that has so-called levels, e.g. the vector `c(1, 2, 3, 1)` could be converted to a factors with `as.factor(c(1, 2, 3, 1))` and would from then on have the levels `1, 2, and 3`).

`comment.car` defines a comment character in the input file. Default is hash (`#`). Use `""` to disable comment characters.

The function `write.table()` saves datasets from R in an external *.txt file:

```
> write.table(x = my.frame, file = "/path/to/file/filename.txt",
+ sep = "\t", dec=".", col.names = TRUE)
```

`col.names` has the same function as `header` in `read.table()`, it defines whether there exist column names (default) or not.

✎ Exercise 2: import a proteomics data set

The file `peptides.txt` contains data from MaxQuant (kindly provided by Miroslav Nikolov). Import this file into R as object `peptides`. Be aware that

- this file's columns are separated by tabulator,
- that the file has a header,
- that we do not want to convert character vectors into factors,
- and that we want to disable all comment characters.

✎ Solution 2: import a proteomics data set

```
> peptides <- read.table("../data/peptides.txt", header = TRUE, sep = "\t",
+ stringsAsFactors = FALSE, comment.char = "")
```

You can have a look at the first couple of lines like this (be aware that columns 16 and 17 have very long entries... you don't want to see them printed in this manuscript):

```
> peptides[1:3, c(1:15,18:30)]
```

```
   id Protein.Group.IDs Mod..Peptide.IDs Evidence.IDs MS.MS.IDs
1   0               191                0            0         0
2 651                56              674          757       780
3 652               366              675          758       781
  Oxidation..M..Site.IDs Carbamidomethyl..C..Site.IDs  Sequence R.Count K.Count
1                                                     AAAEELLAR       1       0
2                                                      FDDDVVSR       1       0
3                                                     FDDPLLGPR       1       0
  Length Missed.Cleavages      Mass                            Proteins
1      9                0  942.5135                         IPI00016932
2      8                0  951.4298 IPI00003965;IPI00646721;IPI00922910
3      9                0 1028.5291 IPI00103994;IPI00871954;IPI00939672
  Leading.Razor.Protein                         ENSEMBL Unique        PEP
1           IPI00016932                 ENST00000238112    yes 4.3018e-03
2           IPI00003965 ENST00000344836;ENST00000381886    yes 2.1176e-02
3           IPI00103994                 ENST00000394434    yes 1.3047e-06
  Mascot.Score Ratio.H.L Ratio.H.L.Normalized Ratio.H.L.Variability....
1        26.72    6.8814              0.27617                        NA
2        31.50        NA                   NA                        NA
3        49.71   12.2720             16.41300                        NA
  Ratio.H.L.Count Intensity Intensity.L Intensity.H Reverse Contaminant
1               1    525570       24236      501330
2               0    174100           0      174100
3               1    366430       27066      339370
```

Or even better, check the structure like this (please ignore that this is running out of the page):

```
> str(peptides)
```

```
'data.frame':        1447 obs. of  30 variables:
 $ id                          : int  0 651 652 653 654 655 656 657 658 659 ...
 $ Protein.Group.IDs           : chr  "191" "56" "366" "64" ...
 $ Mod..Peptide.IDs            : chr  "0" "674" "675" "676" ...
 $ Evidence.IDs                : chr  "0" "757" "758" "759" ...
 $ MS.MS.IDs                   : chr  "0" "780" "781" "782" ...
 $ Oxidation..M..Site.IDs      : chr  "" "" "" "" ...
 $ Carbamidomethyl..C..Site.IDs: chr  "" "" "" "" ...
 $ Sequence                    : chr  "AAAEELLAR" "FDDDVVSR" "FDDPLLGPR" "FDEISFVNFAR" ...
 $ R.Count                     : int  1 1 1 1 1 1 0 1 1 1 ...
 $ K.Count                     : int  0 0 0 0 0 1 1 0 0 0 ...
 $ Length                      : int  9 8 9 11 11 8 10 10 11 16 ...
 $ Missed.Cleavages            : int  0 0 0 0 0 1 0 0 0 0 ...
 $ Mass                        : num  943 951 1029 1344 1177 ...
$ Proteins                  : chr  "IPI00016932" "IPI00003965;IPI00646721;IPI00922910" "IPI00103994;I
$ Leading.Razor.Protein     : chr  "IPI00016932" "IPI00003965" "IPI00103994" "IPI00005154" ...
$ Protein.Names             : chr  "Phosphatidylinositol-3,4,5-trisphosphate 5-phosphatase 2;SH2 doma
$ Uniprot                   : chr  "Q9UKF6;Q53F02;Q53RS2;A4UCU1;B4DQR2;Q05BZ5" "Q93009;B7Z7T5;B7Z815;
$ ENSEMBL                   : chr  "ENST00000238112" "ENST00000344836;ENST00000381886" "ENST000003944
$ Unique                    : chr  "yes" "yes" "yes" "yes" ...
$ PEP                       : num  4.30e-03 2.12e-02 1.30e-06 1.90e-12 1.98e-04 ...
$ Mascot.Score              : num  26.7 31.5 49.7 70.4 34.4 ...
$ Ratio.H.L                 : num  6.88 NA 12.27 15.49 NA ...
$ Ratio.H.L.Normalized      : num  0.276 NA 16.413 0.622 NA ...
$ Ratio.H.L.Variability.... : num  NA NA NA NA NA NA NA NA NA NA ...
$ Ratio.H.L.Count           : int  1 0 1 1 0 1 0 1 0 0 ...
$ Intensity                 : int  525570 174100 366430 803000 273650 42147 506490 1381500 71472 1379
$ Intensity.L               : num  24236 0 27066 0 0 ...
$ Intensity.H               : num  501330 174100 339370 803000 273650 ...
$ Reverse                   : chr  "" "" "" "" ...
$ Contaminant               : chr  "" "" "" "" ...
```

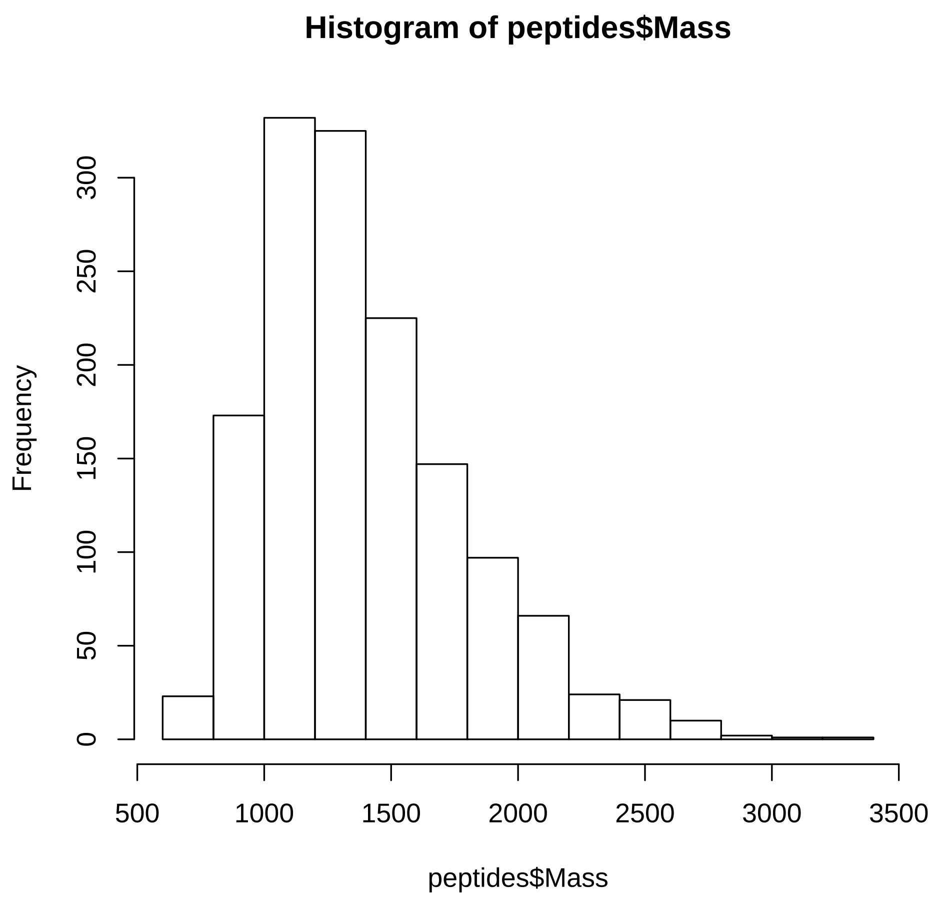
4 Plots

4.1 Histogram

A histogram shows frequency and might also be used to obtain the normal distribution of a sample. The function `hist()` creates a histogram. The argument `breaks` defines the number of cells displayed in the histogram.

Creating a histogram of the column `Mass` in the data frame `peptides`:

```
> hist(peptides$Mass)
```



4.1.1 Some more graphics parameters

The following parameters work for all plotting functions:

`col` allows to specify a plot color

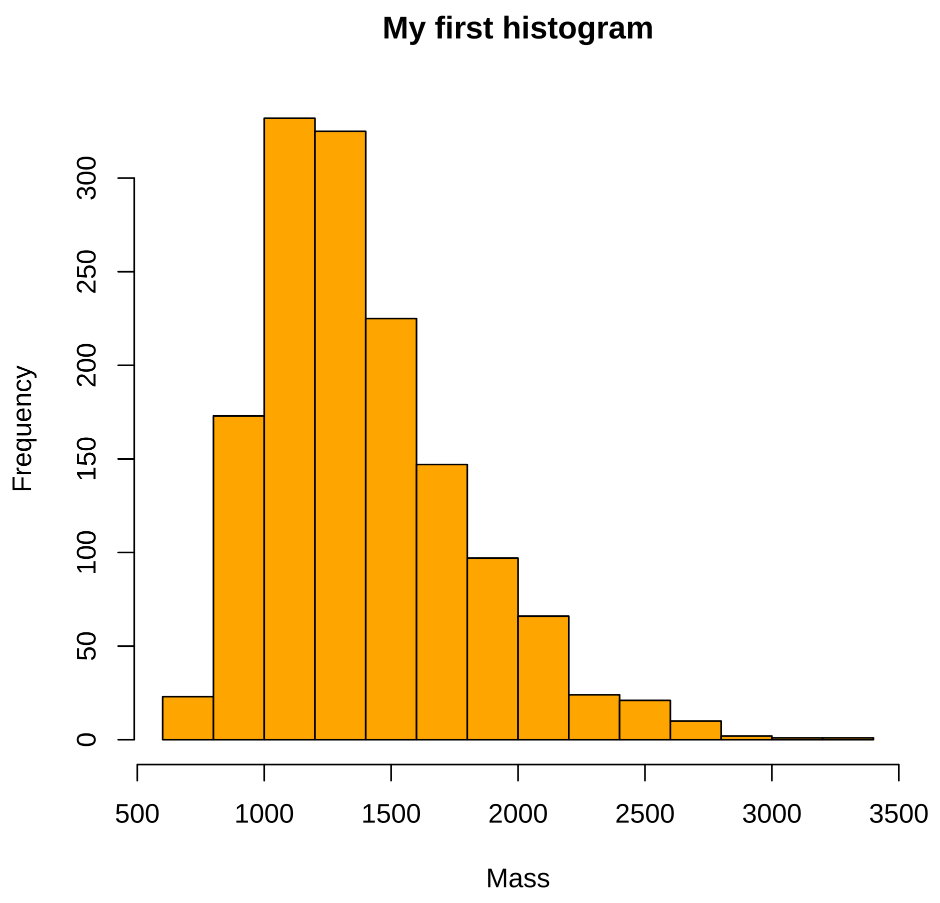
`main` sets a caption

`xlab` sets an x-axis label

`ylab` sets a y-axis label

Example:

```
> hist(peptides$Mass, col = "orange", main = "My first histogram", xlab = "Mass",
+ ylab = "Frequency")
```



4.2 Boxplot

A boxplot shows the distribution of one or more samples. It is often used to check the normal distribution. Several boxplots are helpful to estimate the homogeneity of variances between different samples.

Some parameters of the function `boxplot()`:

```
boxplot(x, col = NULL, xlab = "...", ylab = "...", main = "...", range = 1.5)
```

`x` is either a vector or a list containing several vectors. Alternatively, data might be specified with the `formula` construct:

```
formula = observations ~ grouping factor with two levels,
data = ..., subset = ..., na.action
```

Using the `formula` construct, group names are treated alphabetically (first position in the alphabet = first position in the function, e.g. first boxplot).

`col` specifies the color of the graph. The function `color()` calls all predefined colors.

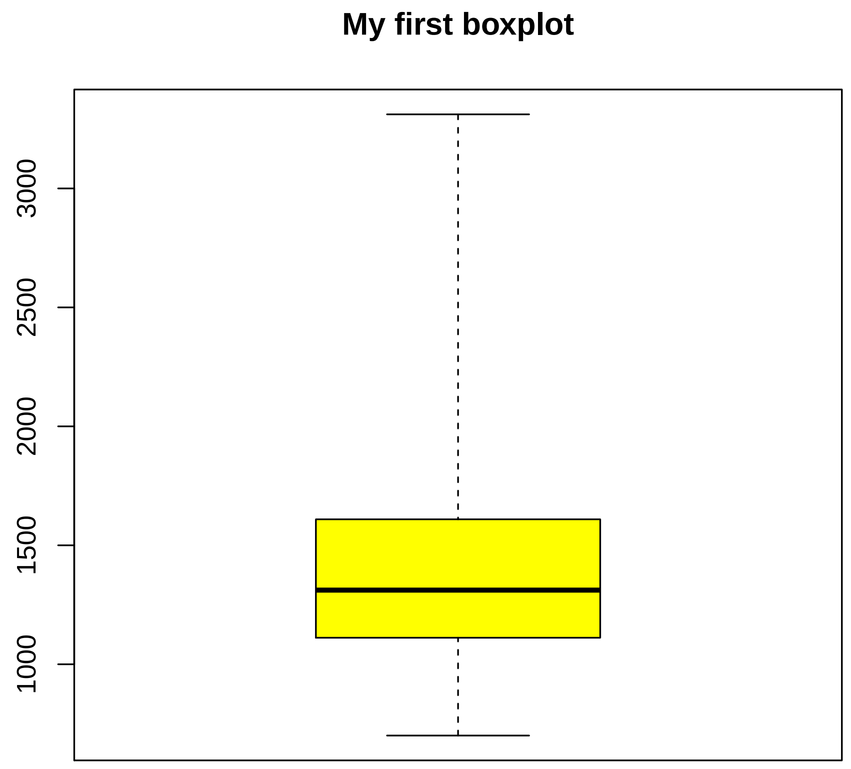
`xlab` and `ylab` set the axes labels. The group names will be displayed by default (if header of a data frame column).

`main` adds a diagram title. This might be replaced by a separate function called `title()`.

`range` determines how far the whiskers extend from the box. Use zero to get whiskers that extend to extremes. Default is `1.5`.

Example:

```
> boxplot(peptides$Mass, main = "My first boxplot", col = "yellow", range = 0)
```



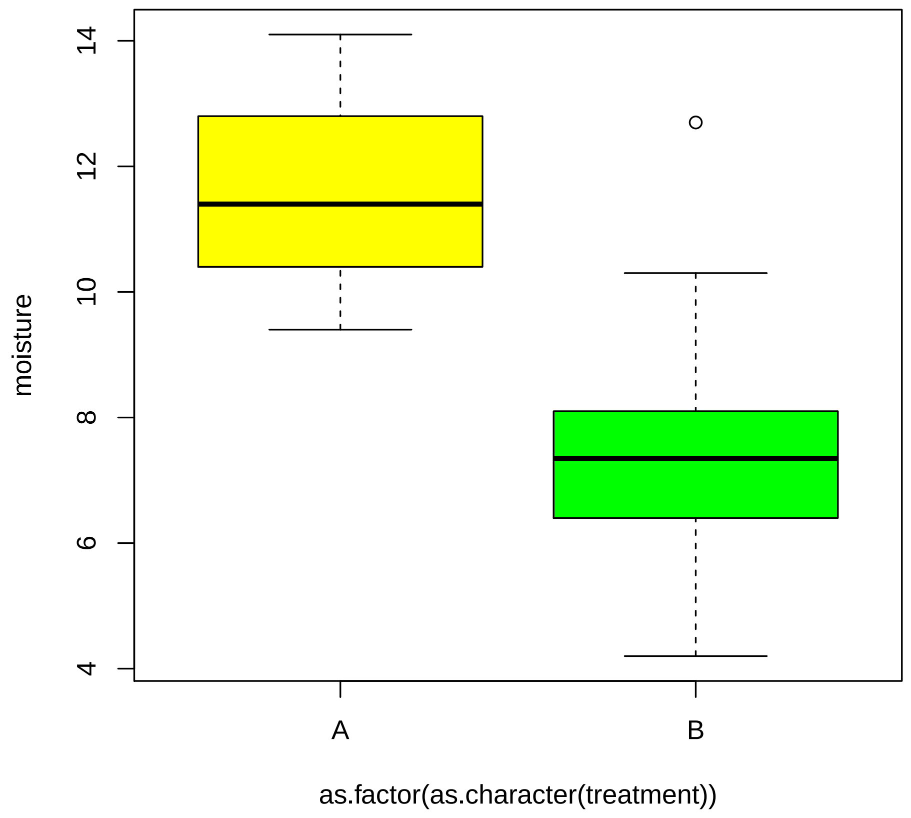
To demonstrate usage of the `formula`-construct, let's assume, we have a data set that looks like this:

```
> soil

   treatment moisture
1          A     12.8
2          A     13.4
3          A     11.2
4          A     11.6
5          A      9.4
6          A     10.3
7          A     14.1
```

```
8          A     11.9
9          A     10.5
10         A     10.4
11         B      8.1
12         B     10.3
13         B      4.2
14         B      7.8
15         B      5.6
16         B      8.1
17         B     12.7
18         B      6.8
19         B      6.9
20         B      6.4
```

```
> boxplot(formula = moisture~treatment, data = soil, col = c("yellow", "green"))
```



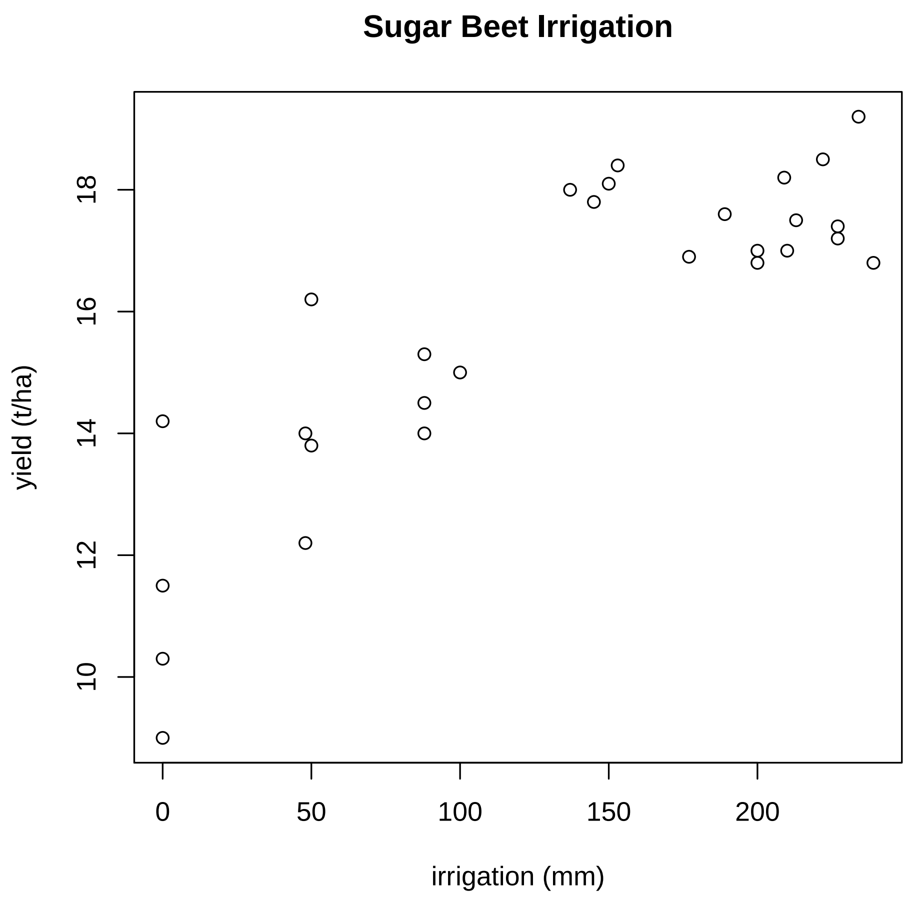
4.3 Scatterplot

The function `plot()` returns the graph of an empiric cumulative distribution in its basic functionality. Example:

```
> beets
```

```
   water yield
1      0   9.0
2      0  10.3
3      0  11.5
4      0  14.2
5     48  12.2
6     50  13.8
7     48  14.0
8     50  16.2
9     88  14.0
10    88  14.5
11   100  15.0
12    88  15.3
13   145  17.8
14   137  18.0
15   150  18.1
16   153  18.4
17   177  16.9
18   189  17.6
19   200  16.8
20   200  17.0
21   209  18.2
22   210  17.0
23   213  17.5
24   222  18.5
25   227  17.2
26   227  17.4
27   234  19.2
28   239  16.8
```

```
> plot(yield~water, data = beets, xlab = "irrigation (mm)", ylab = "yield (t/ha)",
+ main = "Sugar Beet Irrigation")
```



or alternatively with vectors `x` and `y` specified:

```
> plot(x = beets$water, y = beets$yield, xlab = "irrigation (mm)", ylab = "yield (t/ha)",
+ main = "Sugar Beet Irrigation")
```

Attention! The function `plot` accepts data input in form of a `formula`-construct, but only if the part `formula =` is **left out**!

4.4 Saving plots

In principle, R offers two approaches to saving graphics:

1. saving the content of the currently active graphical device
2. recording a graphic directly into a graphical device that is a file

We will here cover the first option.

4.4.1 EPS-format

1. Create a plots
2. type

```
dev.copy2eps(file = "filename.eps")
```

4.4.2 PDF-format

Identical to eps-format, except that you need to change the command to:

```
dev.copy2pdf(file = "filename.pdf")
```

✎ Exercise 3: Ratio vs. Intensity Scatterplot

Data and commands for this exercise were kindly provided by Miroslav Nikolov.

1. Import the MaxQuant file `proteinGroups.txt` into the R object `proteinGroups`, check the object structure with `str()`.
2. Use the following command to exclude lines that contain reverse hits:

   ```
   > proteinGroups <- read.table("proteinGroups.txt", quote = "\"", header = TRUE, +
   + sep = "\t", stringsAsFactors = FALSE, comment.char = "")
   > proteinGroups <- proteinGroups[proteinGroups$Reverse != "+",]
   ```

3. Use the following command to exclude lines with contaminants:

   ```
   > proteinGroups <- proteinGroups[proteinGroups$Contaminant != "+",]
   ```

4. Use the following command to exclude peptides that do not report a ratio:

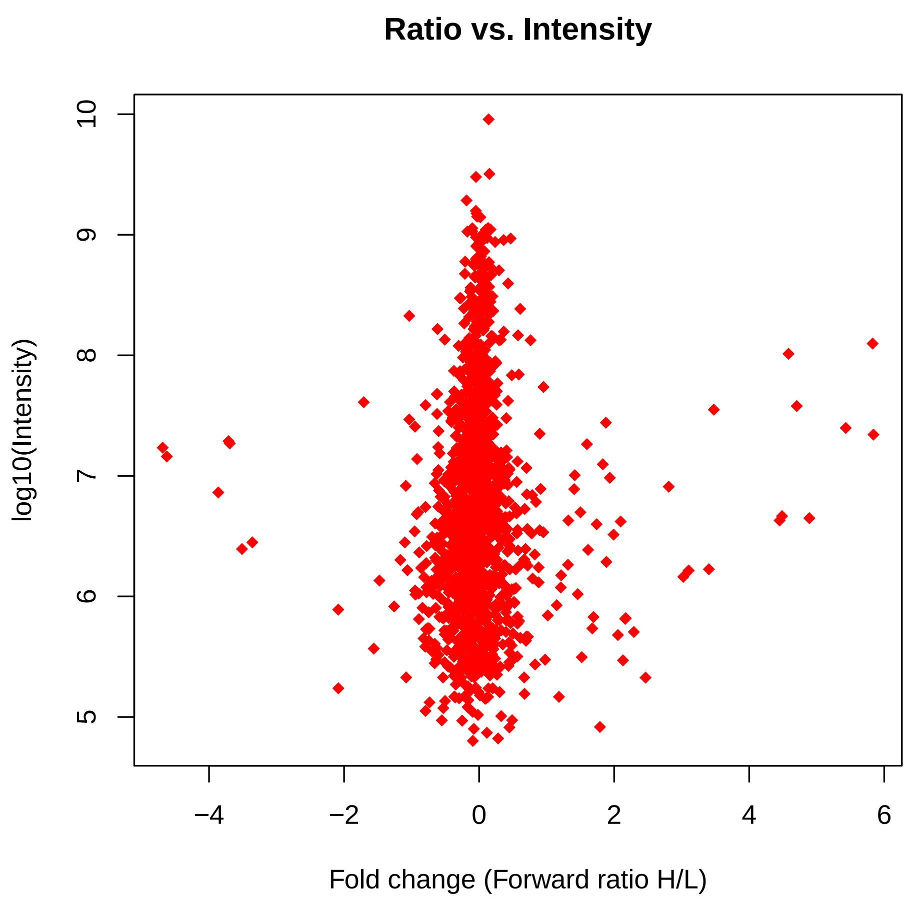
   ```
   > proteinGroups <- proteinGroups[is.na(proteinGroups$Ratio.H.L.Normalized) == FALSE,]
   ```

5. A ratio vs. intensity plot has the log_2 of the column `Ratio.H.L.Normalized` on the x-axis and log_{10} of the column `Intensity` on the y-axis. You get the logarithms in R by using `log2()` and `log10()`, respectively. Create a scatterplot using the `plot()` function.
6. Add a title and axes-labels to the plot.
7. Modify `color` and `pch` to your taste.
8. Save the plot as a pdf file!

In the other R workshop, you will learn more about beautifying plots!

✎ Solution 3: Ratio vs. Intensity Scatterplot

```
> proteinGroups <- read.table("proteinGroups.txt", quote = "\"", header = TRUE, sep = "\t",
+ stringsAsFactors = FALSE, comment.char = "")
> proteinGroups <- proteinGroups[proteinGroups$Reverse != "+",]
> proteinGroups <- proteinGroups[proteinGroups$Contaminant != "+",]
> proteinGroups <- proteinGroups[is.na(proteinGroups$Ratio.H.L.Normalized) == FALSE,]
> plot(x = log2(proteinGroups$Ratio.H.L.Normalized), y = log10(proteinGroups$Intensity),
+ col = "red", pch = 18, main = "Ratio vs. Intensity",
+ xlab = "Fold change (Forward ratio H/L)", ylab = "log10(Intensity)")
> # dev.copy2pdf()
```



5 t-Test

The parametric t-Test compares the mean of two samples.

The ”classical” **t-Test** is used with the following assumptions:

- **Approximate normal distribution of data** is read from the boxplots: The median lies in the middle of the box and both whiskers have an equal length. The normal distribution results in continuity of data, e.g. temperatures measured in Kelvin or lengths measured in metres.
- **Homogeneity of variances** is either read from the boxplots: The respective boxes including whiskers have the same length. Or the homogeneity of variances is checked with a statistical test.
- **Independence of data** is not fulfilled if one has e.g. taken data on the same fruit trees in two consecutive years. In vitro explants that originate in the same mother plant are not allowed to be treated as independent.

The **Welch t-test** is very similar to the ”classical” t-Test. Assumptions are normal distribution as well as independence of data. But the Welch t-test is more tolerant to heterogeneity in variances.

A **paired t-Test** implies:

- **Paired data**: A paired sample results from e.g. the investigation of the effect of two insecticides on different branches of the same tree.
- **Normal distribution of the differences in mean** (Boxplot).

5.1 The Function `t.test()`

```
t.test(x, y = NULL, alternative = c("two.sided", "less", "greater"),
            var.equal = FALSE, paired = FALSE, conf.level = 0.95, ...)
```

or

```
t.test(formula, data, subset, na.action, ...)
```

`x` and `y` represent two vectors that will be compared. `x` is the only essential variable while `y` is an optional argument (the function `t.test()` might be used for a one sample t-test). Alternatively, data can be implied with the `formula`-construct.

`data` specifies the data set for a `formula`-construct.

`subset` selects data that will be ex- or included regarding certain criteria.

`na.action` defines the treatment for values which are not available. Options for this argument are called with:

```
getOption("na.action").
```

`alternative` indicates whether a two-sided (H$_1$: $\mu_1 \neq \mu_2$), one-sided acceding (H$_1$: $\mu_1 > \mu_2$) or one-sided seceding (H$_1$: $\mu_1 < \mu_2$) test is calculated.

`var.equal` declares whether the variances are heterogeneous (`FALSE`) or homogeneous (`TRUE`). The default is `FALSE`, which stands for a t-Welch test. It has to be set on `TRUE` for a classical t-Test.

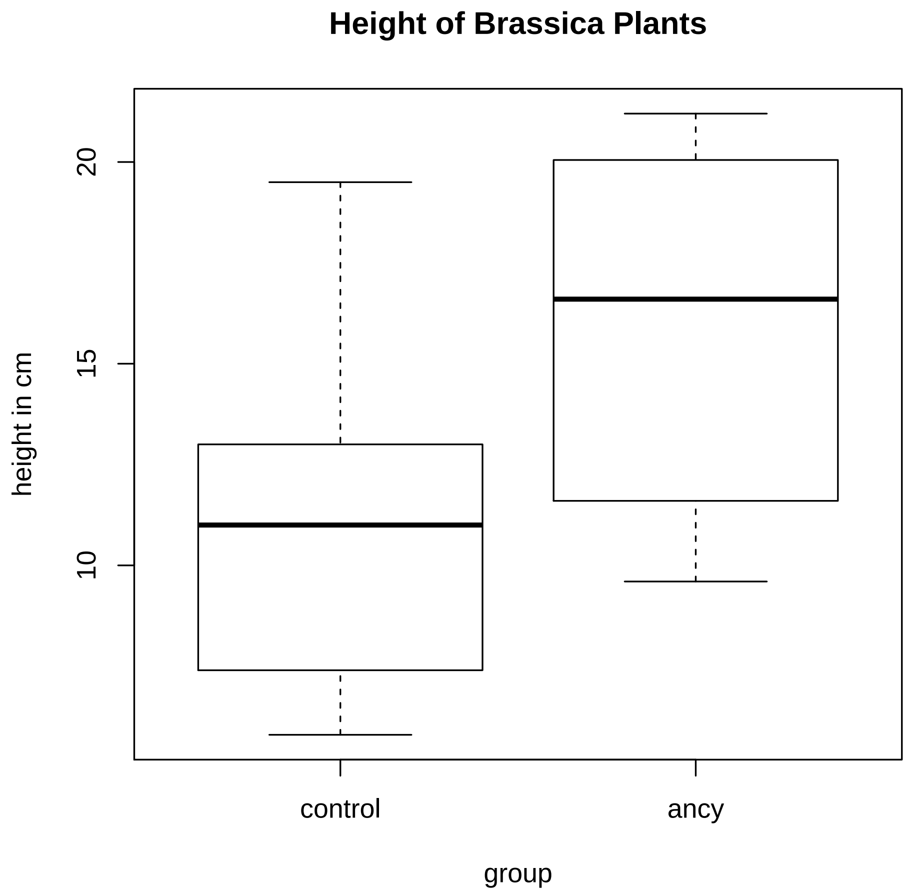
`conf.level` specifies the confidence level. The α error is calculated from 1 - `conf.level`. 0.95 is the default value (95% $\Rightarrow \alpha = 5\%$).

`paired` is set on `FALSE` by default. A paired t-Test is calculated if it is set on `TRUE`.

5.2 Example

Brassica campestris plants were split into two groups. The first group was treated with a substance called Ancymidol. Ancymidol is a herbicide. The other group was not treated and served as control group. At the end of the experiment, the height of each plant was measured in centimeters. The data looks like this:

```
> brassica <- read.table("brassica.txt", sep = "\t", header = TRUE)
> boxplot(formula = height~group, data = brassica, ylab = "height in cm",
+ main="Height of Brassica Plants", names = c("control", "ancy"))
```



- Approximate **normal distribution** is accepted because the median is located in the middle of both boxes.
- Approximate **homogeneity of variances**.
- **Continuous data** because height is indicated in cm
- **Independency of data** because the plants were treated independent from each other.

$\Longrightarrow$ Data is suitable for the analysis with a classical t-Test. Ancymidol is a growth repressor. Therefore, a one-sided test with the expectation that Ancymidol treated plants are smaller than the control group is calculated. Hypotheses:

$$H_0 : \mu_{control} \leq \mu_{ancy}$$
$$H_1 : \mu_{control} > \mu_{ancy}$$

```
> t.test(formula = height~group, data = brassica, var.equal = TRUE,
+ alternative = "less", conf.level = 0.95)

        Two Sample t-test

data:  height by group
t = -1.9919, df = 13, p-value = 0.03391
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
      -Inf -0.543402
sample estimates:
   mean in group ancy mean in group control
             11.01429              15.91250
```

5.2.1 Interpretation

```
        Two Sample t-test
```

The line presents the test header. If the variable `var.equal = TRUE` would not have been set, the function would return `Welch Two Sample t-test`.

```
data:  height by group
```

This says that the `formula`-construct compared heights dependent on the group.

```
t = -1.9919, df = 13, p-value = 0.03391
```

The test statistic `t` amounts 1.9919. This value is usually compared to a table value. The comparison of means is called "significant" if the t-value is more extreme than the table value for the respective quantile and degrees of freedom. Degrees of freedom are printed as `df = 13`. The `p-value` is compared to the respective α-error. The test result is significant if the p-value is smaller than α.

α must be set a priori before the test itself is calculated! In R, the default of α is 5%. The plants treated with Ancymidol are significantly shorter than the non treated control group because $0.03391 < 0.5$. The alternative hypothesis is accepted.

```
alternative hypothesis: true difference in means is greater than 0
```

This line returns the alternative hypothesis.

```
95 percent confidence interval:
      -Inf -0.543402
```

The 95% confidence interval for the difference of the true parameters $\mu_{control}$ - μ_{ancy} is displayed. If the experiment was repeated infinite times, the true difference would be located within the respective confidence interval in 95% of all cases . However, there is no statement about the current experiment in it.

Practice: If the confidence interval includes zero, the test result is counted as not significant. If the result is significant (zero not included), the difference to zero represents a measure of rejection of the

H_0-hypothesis. The interval width accounts for scattering and the number of observations. In general, confidence intervals are displayed in the original data's dimension: in this example measurements in centimeteres.

The given confidence interval `0.543402 Inf` indicates a significance to a confidence level of 0.95 because zero is excluded: $\mu_{control}$ - $\mu_{ancy} = 0$ can be rejected with an error probability of 5%. More detailed, the confidence interval indicates that the control plants are at least 0.542402 cm higher than the Ancymidol treated plants.

```
sample estimates:
mean in group ancy mean in group control
          11.01429              15.91250
```

Output of the mean values. Plants treated with Ancymidol have an average height of 11.0 cm whereas the control plants have a mean height of 15.9 cm.

The overall conclusion for this experiment is that the alternative hypothesis is accepted with a confidence level of 0.95.

✎ Exercise 4: sick mice

An experiment was conducted to measure the time that mice need to flee a certain distance from an obvious danger.The escape route was 10 cm wide and 2 m long tunnel. A big noise and vibrations were used to scare the mice. Actually, there were two groups of mice: one of them suffered from a neurodegenerative disease. The other group was healthy and served as control group.

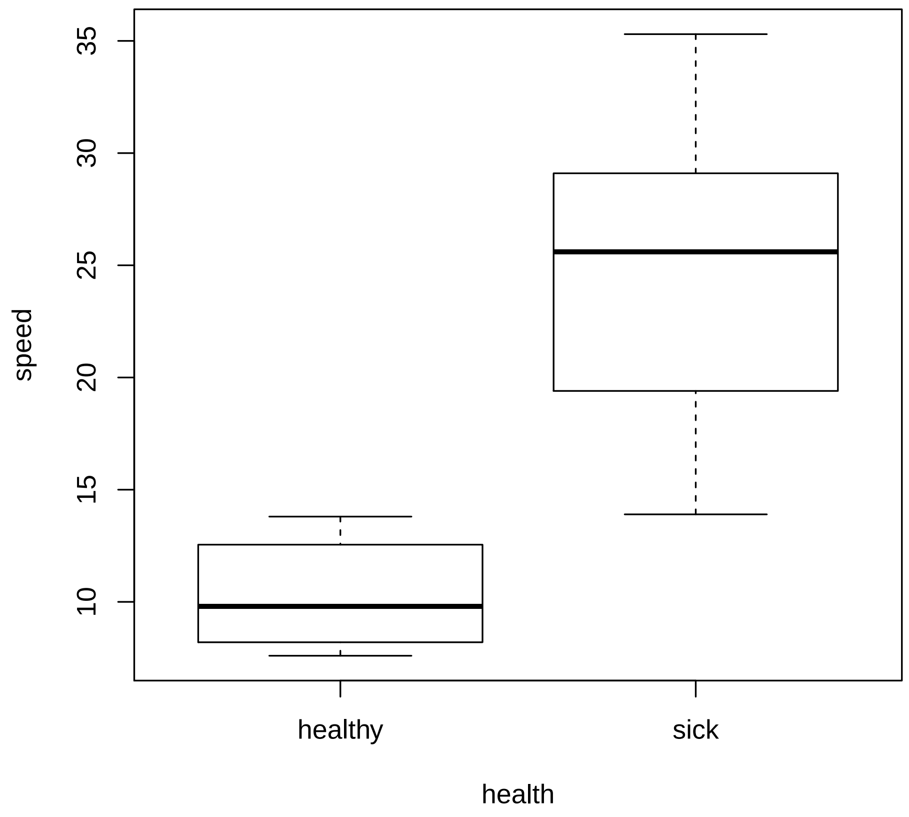
Sick	19.4	29.1	35.3	24.9	13.9	17.7	30.1	26.3	27.7	20.0	
Healthy	21.8	8.5	9.3	7.9	13.45	13.3	7.8	11.4	7.6	9.0	11.8

1. Read the data set `mice.txt` into an R object.
2. Create a boxplot. What are your assumptions about distribution and homogeneity of variances?
3. Form hypotheses for this experiment.
4. If you think, the t-test is a good choice for this experiment, conduct a suitable version of the t-test. How do you interpret the output?

✎ Solution 4: sick mice

```
> mice <- read.table("mice.txt", header = TRUE, sep = "\t", dec = ".")
```

```
> boxplot(formula = speed~health, data = mice)
```



Data of both groups is approximately normal, variances are heterogeneous.

$$H_0 : \mu_{healthy} \geq \mu_{sick}$$

$$H_1 : \mu_{healthy} < \mu_{sick}$$

```
> t.test(formula = speed~health, data = mice, alternative = "less",
+  var.equal = FALSE, conf.level = 0.95)

        Welch Two Sample t-test

data:  speed by health
t = -6.3816, df = 11.225, p-value = 2.376e-05
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
      -Inf -10.09944
sample estimates:
mean in group healthy    mean in group sick
             10.39545              24.44000
```

With a probability of error of 5%, healthy mice flee significantly faster through the tunnel than sick mice (because the p-value is smaller than 0.05). The confidence intervals mean that with a level of significance of 5%, healthy mice flee 10.09 to Inf seconds faster than sick mice.