# Analyzing and Documenting Data with Freeware

Miguel Alvarez, Bisrat Gebrekidan and George Ong'amo

Nairobi, 05-08-2019


## Contents

# 1 Introduction to R

Since **R** is a programming language, most of the communication with the interface happens through command lines. Herewith it is important to review the basics on the syntax of this language. For more details, consult Crawley (2007) and Burns (2011).

## 1.1 Basics on R Syntax

One of the most important tasks of **R** is mathematical calculations. Therefore this language is suitable for teaching on mathematics and related disciplines. You can thus apply basic operations

```
10 + 5*20
```

```
## [1] 110
```

```
(10 + 5)*20
```

```
## [1] 300
```

A list of mathemtical and logical operators are displayed in Table 1.

Table 1: Mathematical and logical operators.

| Operator | Description |
|---|---|
| `+`, `-` | Addition, subtraction. |
| `*`, `/` | Multiplication, division. |
| `^`, `**` | Exponentiation. |
| `sqrt()`, `log()`, `exp()` | Squared root, logarithm, exponential value. |
| `==`, `!=` | Equal to, different from. |
| `>`, `>=`, `<`, `<=` | Greater than, greater than or equal to, smaller than, smaller than or equal to. |
| `!`, `%in%` | Not, in. |
| `&`, `\|` | And, or. |
| `all()`, `any()` | All or any fulfilling a condition. |

## 1.2 R Objects

**R** is an object-oriented language, meaning that information can be structured in objects, while function will behave according to the kind (class) of object used as input (Genolini, 2008). Content or values of objects can be assigned by using `<-`.

```
A <- 5
B <- 2
A + B
```

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

```
A*B
```

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

The fundamental object in **R** is the `vector`. A `vector` is a sequence of values belonging to the same mode or class (Crawley, 2007). A further attribute of the vector is its `length` (its only dimension). Modes for data are **logical**, **numeric**, **factor**, **complex** and **character**, among others.

```
rep(5, times=10)
seq(from=10, to=100, by=10)
sample(letters, size=10, replace=TRUE)
c("Peter","Piper","picked","a","peck","of","pickled","peppers")
```

While single values, as in the first examples, are actually vectors of length equal to 1, operation can be applied to many values in vectors at once.

```
A <- 1:5
B <- seq(10, 50, by=10)
A + B
```

```
## [1] 11 22 33 44 55
```

There are several ways allowing access to elements contained in a vector. The most common is using square brackets as displayed in the console. In such brackets you can indicate the position of elements through integer values. Negative integers indicate elements to be excluded. It is also possible to use logical values, whereupon **TRUE** indicates elements to be included and **FALSE** indicates elements to be excluded.

```
B[5] # using number
B[1:5] # using numeric vector
B[-5] # using negative number
B[B != 7] # using logical vector
```

Additionally to it, there is also the possibility to name the elements of a vector and access to them using those names as identity.

```
names(B) <- LETTERS[1:length(B)]
B
B[c("C","E","A")]
```

The `matrix` is in **R** a `vector` with 2 dimensions assigned as attribute. Note that while in mathematics a vector is a special case of matrix, in **R** is the other way round. To produce a `matrix` you can use the functions `matrix()`, `rbind()` (binding rows), or `cbind()` (binding columns).

```
# matrix using function matrix matrix(1:20, nrow=5)
matrix(1:20, ncol=5, byrow=TRUE)

# matrix using functions cbind and rbind
cbind(A,B)
rbind(A,B)
```

The access to single elements in the matrix is analogous to the access for vectors, but two values (separated by commas) are required, the first value indicates the row and the second, the column.

```
# Creating the matrix in the workspace
M <- cbind(A,B)
M[1,] # access to first row
M[,1] # access to first column
M["A","B"] # access by names of rows and columns
```

Lists are more complex but at the same time more flexible. A `list` is an object with elements of different classes (even lists). The access to elements of a list is similar as for vectors. The use of double square brackets (`[[]]`) and the use of the dollar symbol (`$`) are two ways of access frequently used on lists.

```
# Creating a list
my_list <- list(First="Hello", Second=A, Third=M)
# Access to elements my_list[1:2] # using a numeric index
my_list[["Third"]] # using element's name
my_list$First # using dollar symbol
my_list$Second[3] # inside of an element
my_list$Third[5,2]
```

One of the most common objects used to handle data in **R** is the `data.frame`. Data frames resemble matrices, but the main difference is that a `matrix` can contain information belonging to only one class (e.g. `numeric`, `factor` or `logical`), while in the data frame every column can be of a different class.

```
E <- letters[1:5]
my_df <- data.frame(First=B, Third=E, stringsAsFactors=FALSE)
summary(my_df)
```

```
##      First       Third
##  Min.   :10   Length:5
##  1st Qu.:20   Class :character
##  Median :30   Mode  :character
##  Mean   :30
##  3rd Qu.:40
```

```
##  Max.   :50
```

The access to elements of a `data.frame` can be done in the same way as for a `matrix`. Additionally you can access to the columns by using the symbol `$`, as for lists.

As you can create objects by using assignments, you can also overwrite them by assigning new values to the object.

```
L <- letters
L[1:5] <- NA
L
```

```
##  [1] NA  NA  NA  NA  NA  "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
## [18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

Similarly, you can use logical operators for the modification of values.

```
L <- letters
L[L == "a"] <- "A"
L
```

```
##  [1] "A" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
## [18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

### 1.3 Use of Scripts and Editors

Besides the direct execution of commands at the console, there is the alternative of writing scripts. R scripts are usually text files using the extension `*.R` and collecting command lines. The advantage of using scripts is on the one side the possibility of executing many command lines at once in the console, to recycle routines and to use them as a report of processes and analyses done in **R**. Another great tool are comments in the script, which can be inserted with a hash symbol (`#`).

Since the script window in **R** does not provide much support for writting commands and big scripts may become confusing, a big help is provided by the use of editors. The next is a list of some editors available for the work with **R**:

- **RStudio** is an editor working in any operative system.
- **StatET** is a plugin developed for the eclipse workbench.
- **Tinn-R** Is an editor developed for Windows.
- **ESS** (Emacs Speaks Statistics)
- **JGR** (Java GUI for R)

### 1.4 Further Elements of R

Additional elements of **R** requiring a brief mention are, among others, the **Workspace** contains the objects created during a session. These objects can be saved in an **R image** by using `save.image()`. Furthermore, a selection of objects can be saved with the function `save()`. The output file can be then imported in a new session by using `load()`.

Commands executed in the console will be stored in a script called **history**, which may be accessible through the function `history()`.

**Exercises**

- Explore the properties of example data sets by using the functions `class()`, `mode()`, `summary()`, and `str()`.
- Manipulate values in data set **iris**.
- Check the applications of functions `split()` and `subset()` according to the help documentation.

## 2 Working with data

One of the major reasons for learning **R** is because of its applications on statistics. Thus, it is crucial to know, how to import and export data handled in R sessions.

## 2.1 Reading and writting data

Main funtions are `readLines()`, `read.table()` and its variants `read.csv()` and `read.csv2()`. Some packages may implement functions to import specific data formats, for instance **OpenOffice** and **LibreOffice** files can be imported using the function `read_ods()` from the package `readODS`, while **MS-Excel** spread sheets can be imported using `read.xlsx()` from the package `xlsx`.

## 2.2 Distributed Data Sets

While a list of data sets available or **R** can be called in sessions (type `data()` in your console), we distributed some additional ones for the purpose of this workshop.

**iris** is one of the most popular data sets available in **R**. The data set was collected by Anderson (1935) and displayed by Fisher (1936).

```
data(iris)
```

**Africa_env** is a data frame contains observations of 36 plots in four localities (two localities from Kenya and two localities from Tanzania). Plots are classified into three land uses, namely "unused fields" (semi-natural vegetation), "grazing lands" and "fallows" (long- and short-term abandoned crop fields).

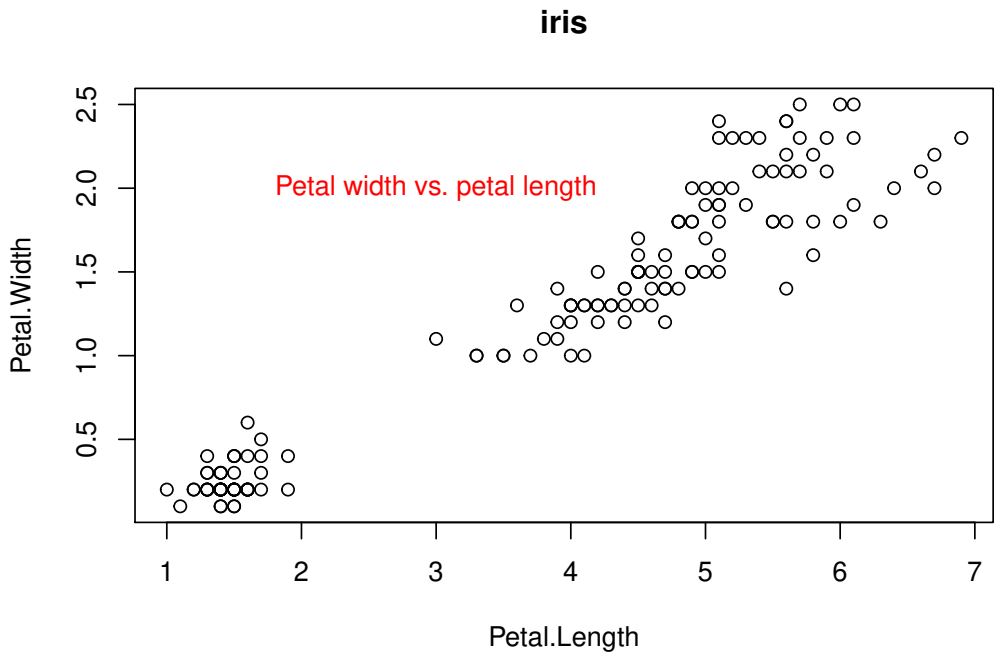
Additional to country, locality and land use membership, `Africa.env` contains soil chemical variables for each plot, namely organic Carbon content (in g $kg^{-1}$), total Nitrogen content (in g $kg^{-1}$), plant available Phosphorous (in mg $kg^{-1}$), exchangeable Potassium (in cmol $kg^{-1}$), electric conductivity (in dS $m^{-1}$) and pH (Kamiri et al., 2013).

```
Africa_env <- read.csv("Africa_env.csv")
```

## 2.3 Basic Statistics and Plotting Functions

The very basic function for plotting in **R** is `plot()` and most of the plotting parameters can be set by using `par()` (take a look in the help file for more details). Plotting functions are classified into two types, on the one side the **high level** functions that produce a whole graphic as in the case of `plot()`, on the other side the **low level** functions are able to introduce single elements in a drawn plot. Some high level functions offer the possibility to use them as low level, usually by setting the argument `add=TRUE`.

```
plot(iris[,c("Petal.Length", "Petal.Width")], main="iris") # High level function
text(3, 2, labels="Petal width vs. petal length", col="red") # Low level function
```




For basic statistics, a series of functions can be applied, for instance `mean()`, `median()`, `var()`, and `sd()`. Also for

extreme parameters the functions `max()`, `min()`, and `range()` are suitable. An useful functions providing many statistics at once is `summary()`.

```
summary(iris$Sepal.Length)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.300   5.100   5.800   5.843   6.400   7.900
```

Table 2: High and low level functions in R.

| Function | displayed graphic or added element |
|---|---|
| | *high level functions…* |
| `plot()` | x-y plot (default: scatter plot) |
| `barplot()` | bar plot |
| `pie()` | pie charts |
| `boxplot()` | box plot |
| `contour()` | contours |
| `curve()` | functions |
| `hist()` | histograms |
| `pairs()` | scatter plot matrices |
| `persp()` | perspective plots |
| | *low level functions…* |
| `points()` | points |
| `text()` | labels |
| `legend` | legend |
| `abline()` | straight line |
| `segment()` | segments |
| `lines()` | connected line segments |
| `arrows()` | arrows (vectors) |

You will realize that plotting functions are an important tool for data exploration (see next chapter).

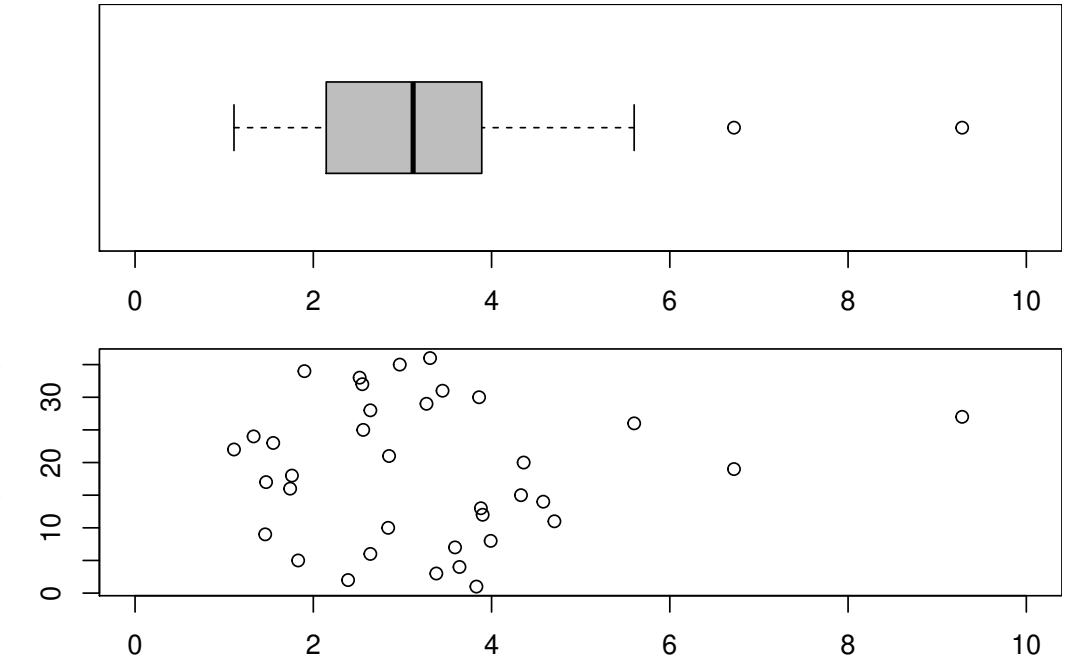
## 2.4 Data Exploration

Before starting statistical analizes, it is advisable to explore collected data and get confident with their content. In these steps, some assumtions set by further statistical tests should be also tested. Although data exploration is not usually mentioned in the methods of a research publication, it is a very important steps and may provide hints about expected results and interpretation of outcomes. We strongly recomment a look on the work of Zuur et al. (2010) for a more detailed overview.

The recognition of **outliers** is one of the starting steps applied in data exploration. Such extreme values may influence the outcomes of statistical assessments and obscure the real trends in the studied phenomenon. In the next example, we check the values of total Nitrogen in the `Africa_env` data set. We make a boxplot but compare it with the Cleveland's dotplot as suggested by Zuur et al. (2010). The Cleveland dotplot display the order of observations versus their values for the selected variable.

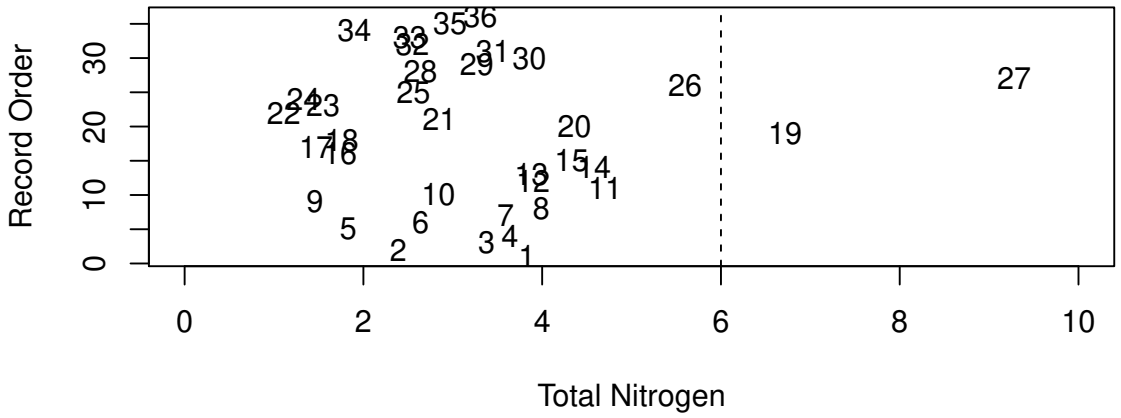
```
Africa_env <- read.csv("Africa_env.csv")

par(mfrow=c(2,1), mar=c(3,3,0,0))
boxplot(Africa_env$TotalN, col="grey", horizontal=TRUE, ylim=c(0,10))
plot(Africa_env$TotalN, 1:nrow(Africa_env), xlim=c(0,10))
```




In this case we could agree with the outcome of the boxplot and skip all values that are higher than 6 g $kg^{-1}$.

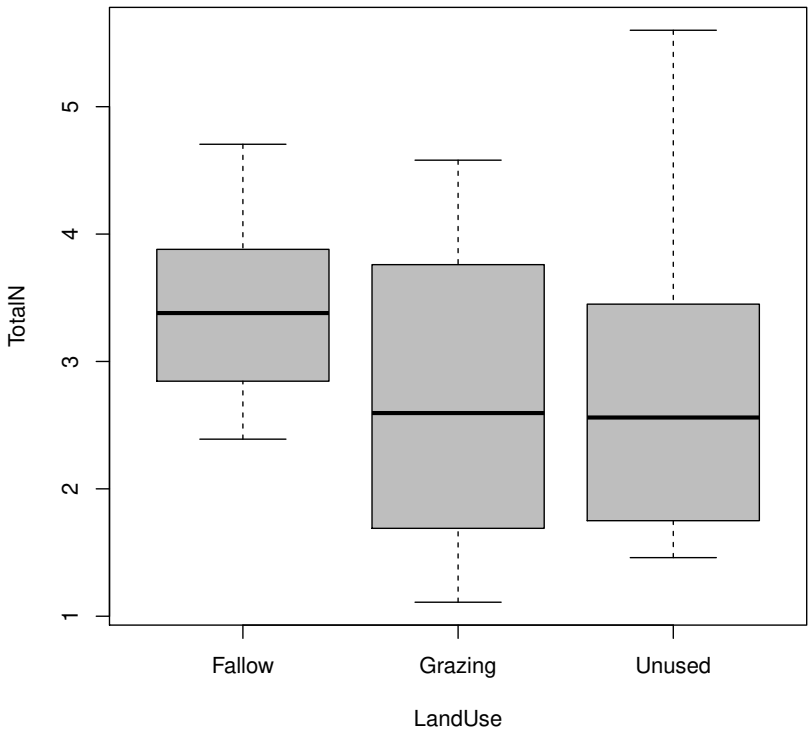
```
plot(Africa_env$TotalN, 1:nrow(Africa_env), xlim=c(0,10), xlab="Total Nitrogen",
        ylab="Record Order", type="n")
text(Africa_env$TotalN, 1:nrow(Africa_env), label=1:nrow(Africa_env))
abline(v=6, lty="dashed")
```




```
Africa_env <- subset(Africa_env, TotalN < 6)
```

In the case of comparing a numerica response versus a categorical factor by using analysis of variance (ANOVA), one important condition is the homology of variances, a.k.a. **homoscedasticity**. This can be visualized through boxplots.

```
boxplot(TotalN ~ LandUse, data=Africa_env, col="grey")
```




In this plot it seems to be that the variability of values for total Nitrogen in soil is much lower within fallow plots than in the other dominant uses. Homoscedasticity can be also checked by the Bartlett test (function `bartlett.test()`).

```
bartlett.test(TotalN ~ Country, data=Africa_env)
```

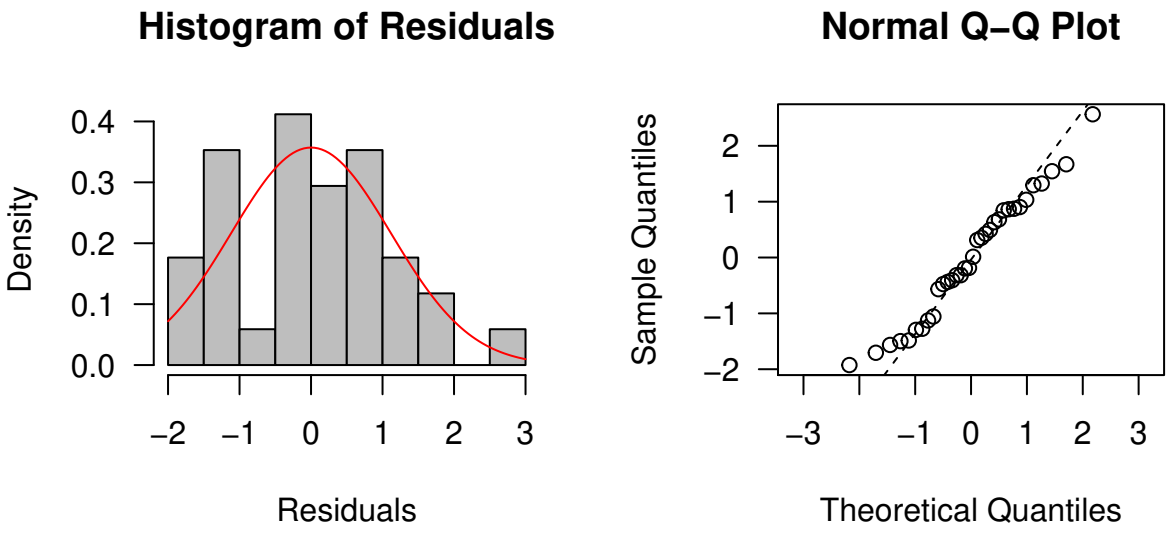
```
##
##  Bartlett test of homogeneity of variances
##
## data:  TotalN by Country
## Bartlett's K-squared = 6.3374, df = 1, p-value = 0.01182
```

This tests checked the null hypothesis that $H_0$: *"the variances of total Nitrogen in soil compared between dominant land uses are homogeneous"*. A P-value lower than 5%, as in this case, indicates a significant difference between variances.

Additionally, a test for **normality** in the distribution of the data have to be carried out previous to a parametric test. While this test should be carried out in every compared population (dominant land use), another approach can be testing the residuals of the model behind the ANOVA.

```
ANOVA <- aov(TotalN ~ Country, data=Africa_env)
Residuals <- resid(ANOVA) # Extracting model residuals

par(mfrow=c(1,2), las=1)
## Plotting histogram of residuals
hist(Residuals, freq=FALSE, col="grey")
curve(dnorm(x, mean=mean(Residuals), sd=sd(Residuals)), col="red", add=TRUE)
## Plotting q-q plots
qqnorm(Residuals, asp=1)
qqline(Residuals, lty=2)
```




Comparing the histogram with the curve showing the expected normal distribution, we can see that there is no much deviation although some gaps within the observed values.

As in the case of homoscedasticity, normality can be also checked by a statistical tests, for instance the Kolmogorov-Smirnov test (function `ks.test()`) and the Shapiro-Wilk test (function `shapiro.test()`).

```
ks.test(Residuals, "pnorm")
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  Residuals
## D = 0.11937, p-value = 0.7178
## alternative hypothesis: two-sided
```

```
shapiro.test(Residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Residuals
## W = 0.97218, p-value = 0.5242
```

According to both tests, the P-value is lower than 0.05 (5%) and thus the difference between a normal distributiona and the one of the observed values is significant.

Note that some authors critisize the use of statistical tests and the decission after a significance level and recommend a desicion according to graphical displays (e.g Zuur et al., 2010; Greenland et al., 2016).

**Data Transformation**

If you are not successful in the previous tests, you can still try them after data transformation. The most popular methods are relatively simple (see Table 3). Such transformations change not only the scale of the variables but they also modify the distribution of the values. Numerical variables are frequently transformed using the logarithmic, reciprocal and squared root transformations. The arcsine-squared root transformation is used to transform percentages and proportions.

If the mentioned transformations did not succeed, it remains the alternative of a Box-Cox transformation. For it the function `boxcox()` is available in the package `MASS`.

Some special cases are the dummy-transformation, applied for the analysis of nominal variables, and the standardization required mainly by multivariate statistics.

Table 3: Formulas and functions used for data transformation.

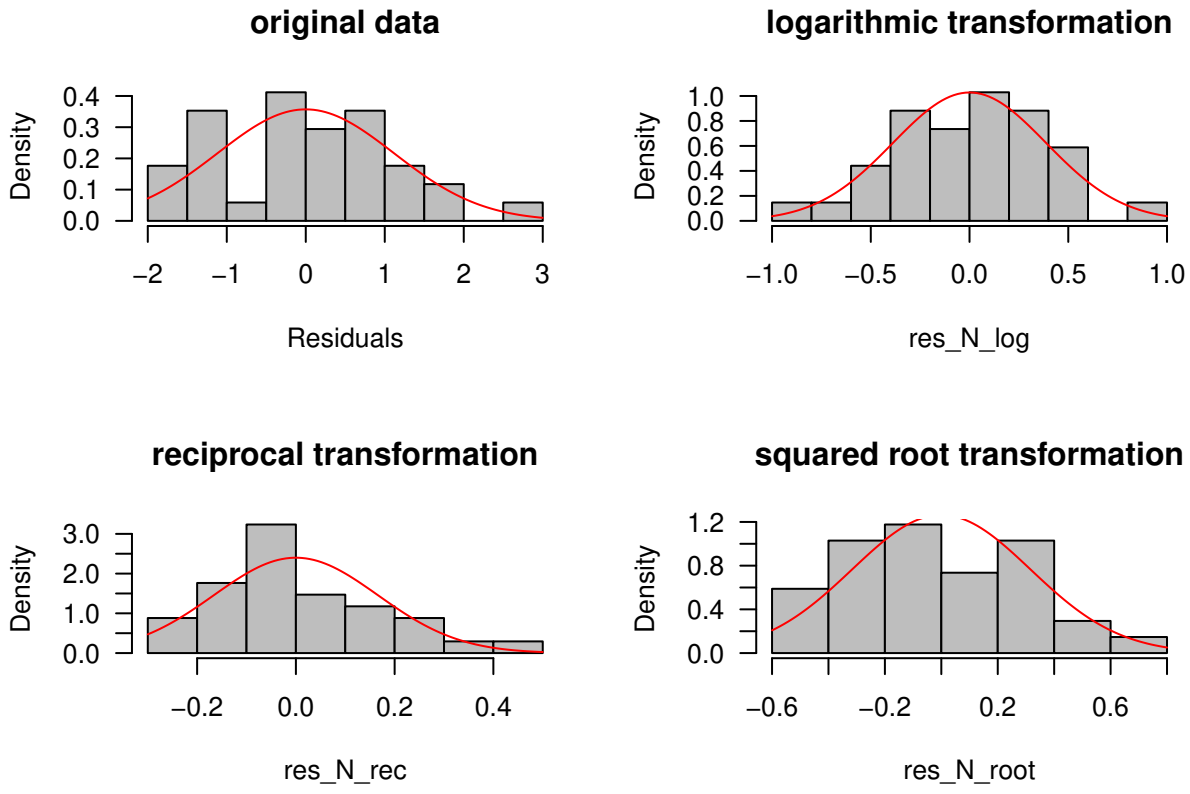
| Function or formula in R | description |
|---|---|
| `log(x+c, base=exp(1))` | logarithmic transformation |
| `1/(x+c)` | reciprocal transformation |
| `sqrt(x+c)` | squared root transformation |
| `asin(sqrt(x/c))*180/pi` | arcsine-squared root transformation |
| `(x-mean(x)/sd(x)), scale(x)` | standardisation |

```
N_log <- log(Africa_env$TotalN, base=exp(1)) # Logarithmic
N_rec <- 1/Africa_env$TotalN # Reciprocal
N_root <- sqrt(Africa_env$TotalN) # Squared Root
```

Since the normality test will be applied to the residuals, as mentioned before, we may calculate them.

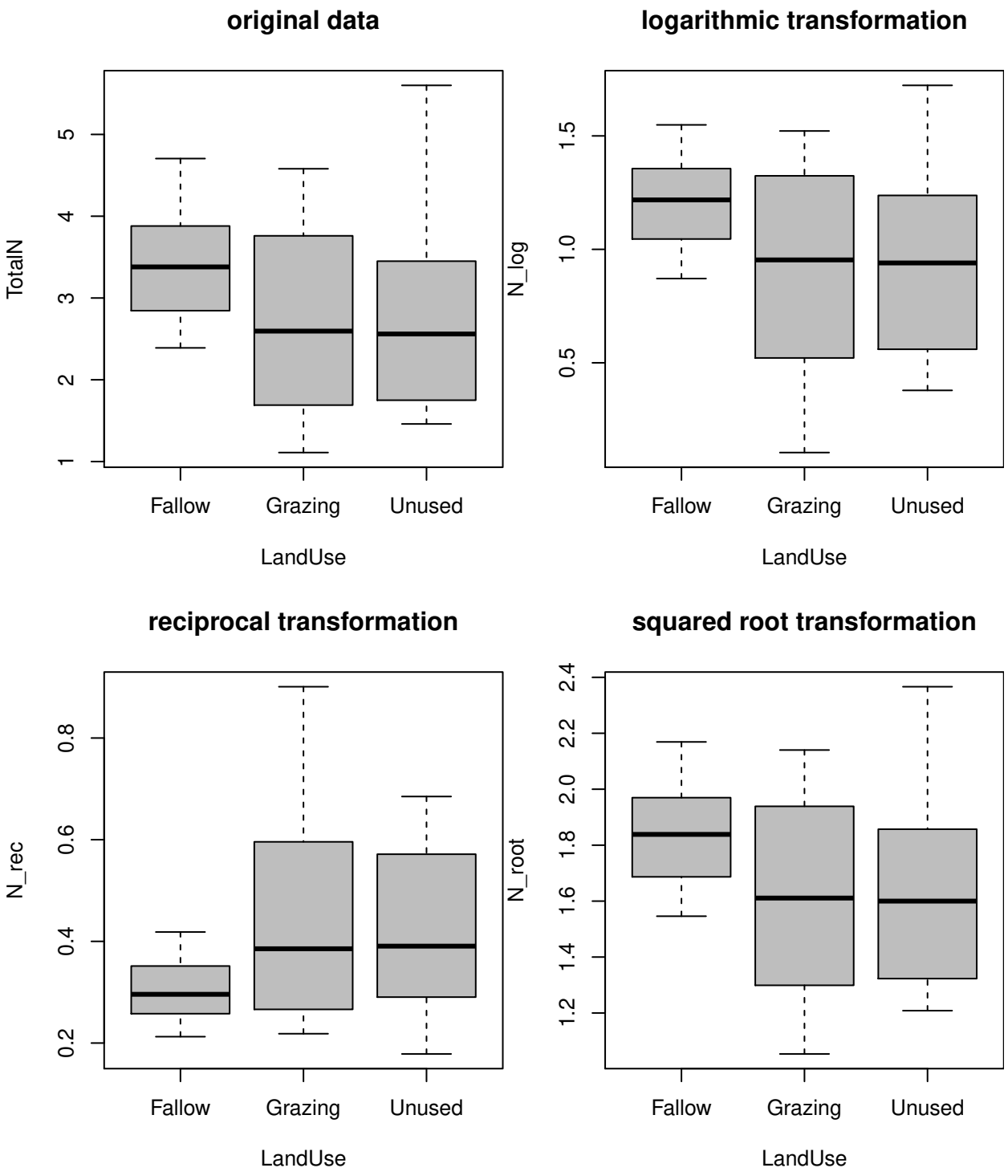
```
res_N_log <- resid(aov(N_log ~ LandUse, data=Africa_env))
res_N_rec <- resid(aov(N_rec ~ LandUse, data=Africa_env))
res_N_root <- resid(aov(N_root ~ LandUse, data=Africa_env))

par(mfrow=c(2,2), las=1)
hist(Residuals, freq=FALSE, col="grey", main="original data")
curve(dnorm(x, mean=mean(Residuals), sd=sd(Residuals)), col="red", add=TRUE)
hist(res_N_log, freq=FALSE, col="grey", main="logarithmic transformation")
curve(dnorm(x, mean=mean(res_N_log), sd=sd(res_N_log)), col="red", add=TRUE)
hist(res_N_rec, freq=FALSE, col="grey", main="reciprocal transformation")
curve(dnorm(x, mean=mean(res_N_rec), sd=sd(res_N_rec)), col="red", add=TRUE)
hist(res_N_root, freq=FALSE, col="grey", main="squared root transformation")
curve(dnorm(x, mean=mean(res_N_root), sd=sd(res_N_root)), col="red", add=TRUE)
```




And then an overview on the distribution of variances.

```
par(mfrow=c(2,2), mar=c(4,4,4,0))
boxplot(TotalN ~ LandUse, data=Africa_env, col="grey", main="original data")
boxplot(N_log ~ LandUse, data=Africa_env, col="grey", main="logarithmic transformation")
boxplot(N_rec ~ LandUse, data=Africa_env, col="grey", main="reciprocal transformation")
boxplot(N_root ~ LandUse, data=Africa_env, col="grey", main="squared root transformation")
```



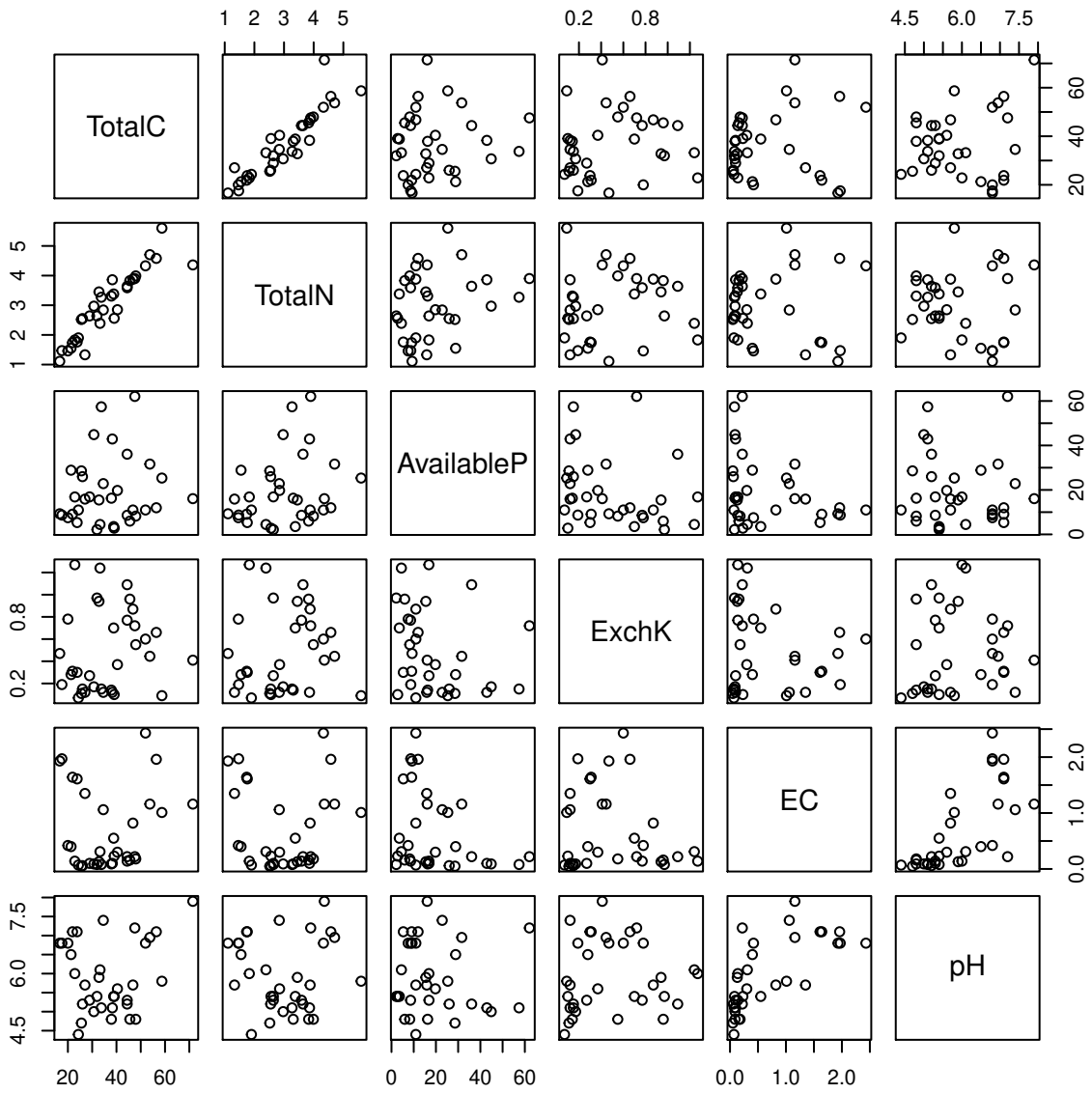

The **correlation** is the degree of association between variables. Causality may be one of the reasons, why two variables are correlated. In some cases it is necessary to discard some hihgly correlated values, since their effects to a responce may be overweighted.

```
# Extracting only numeric values
num_only <- Africa_env[,c("TotalC","TotalN","AvailableP","ExchK","EC","pH")]
```

```r
round(cor(num_only), digits=3)
```

```
##            TotalC TotalN AvailableP  ExchK     EC     pH
## TotalC      1.000  0.916      0.118  0.145  0.091  0.135
## TotalN      0.916  1.000      0.262  0.113 -0.022 -0.033
## AvailableP  0.118  0.262      1.000 -0.252 -0.289 -0.058
## ExchK       0.145  0.113     -0.252  1.000 -0.129  0.082
## EC          0.091 -0.022     -0.289 -0.129  1.000  0.703
## pH          0.135 -0.033     -0.058  0.082  0.703  1.000
```

```r
pairs(num_only)
```




## Exercises

- Make box plots for the variable **Height** in the data **trees**.
- Check **data(iris)** and draw box plots for the variable **Sepal.Length** separated by **Species**.
- Prepare a plot for publication.
- Check normal distribution and homoscedasticity in own data set.

# 3 Parametric Statistics

In this chapter we will briefly introduce to parametric statistics and hypothesis tests.

## 3.1 Analysis of Variance

The analysis of variance (ANOVA) assumes that the variability of measurements is influenced by determined factors. Additional effects by unknown factors are considered as random (non-systematic) and called "experimental errors". For an easy explanation we will consider the effect of just one factor (independent variable) on a dependent variable, the so called one-way ANOVA. Just to contextualize this example in a scientific problematic, our question will be *"Is the electric conductivity of the soils depending on their country of origin?"* or in other words *"Is the country of origin influencing the electric conductivity of the soils?"* I do not really know, how interesting or logic this question can be but it works as a nice example. Then we may formulate a null hypothesis $H_0$: *"electric conductivity of soils is not depending on their country of origin"*. The alternative hypothesis will be $H_1$: *"electric conductivity is depending on country of origin and therefore different between countries"*.

To carry out the analysis of variance we will use the function `aov()`, which is internally calling a linear model (function `lm()`). Alternatively you can use the function `anova()`, but then you may previously fit the linear model.

```
Africa_env <- read.csv("Africa_env.csv")
ANOVA <- aov(EC ~ Country, data=Africa_env)
summary(ANOVA)
```

```
##             Df Sum Sq Mean Sq F value   Pr(>F)
## Country      1 11.447  11.447   39.94 3.32e-07 ***
## Residuals   34  9.745   0.287
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

According to the result of `aov()` there is a singnificant difference in the soil electric conductivity between countries. Nevertheless, the conditions of normality and homoscedasticity are not checked.

```
shapiro.test(resid(ANOVA))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  resid(ANOVA)
## W = 0.90258, p-value = 0.004017
```

```
bartlett.test(EC ~ Country, data=Africa_env)
```

```
##
##  Bartlett test of homogeneity of variances
##
## data:  EC by Country
## Bartlett's K-squared = 34.712, df = 1, p-value = 3.823e-09
```

After exploration, we came out that logarithmic transformation does the magic.

```
EC_log <- log(Africa_env$EC, base=exp(1))
ANOVA <- aov(EC_log ~ Country, data=Africa_env)
shapiro.test(resid(ANOVA))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  resid(ANOVA)
## W = 0.9831, p-value = 0.8442
```

```
bartlett.test(EC_log ~ Country, data=Africa_env)
```

```
##
##  Bartlett test of homogeneity of variances
##
```

```
## data:  EC_log by Country
## Bartlett's K-squared = 0.52083, df = 1, p-value = 0.4705
```

```
summary(ANOVA)
```

```
##             Df Sum Sq Mean Sq F value   Pr(>F)
## Country      1  38.21   38.21   72.11 6.44e-10 ***
## Residuals   34  18.02    0.53
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Thus, there is a significant effect of the country of origin to the values of electric conductivity in the soil for this data set.

Previous to finish the parametric test, some advices regarding **ANOVA**. In the example we applied the ANOVA to a factor with just two levels (Kenya and Tanzania). In such case may be enough the use of a paired two-sample t-test (function `pairwise.t.test()`). We can apply the one-way ANOVA to factors with more than two levels instead, for example comparing soil reaction in different localities.

```
ANOVA <- aov(pH ~ Locality, data=Africa_env)
summary(ANOVA)
```

```
##             Df Sum Sq Mean Sq F value   Pr(>F)
## Locality     3  17.17   5.724   14.15 4.82e-06 ***
## Residuals   32  12.94   0.404
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In other cases we may require the addition of a new explanatory (independent) variable, for example the land use type.

```
ANOVA <- aov(pH ~ Locality + LandUse, data=Africa_env)
summary(ANOVA)
```

```
##             Df Sum Sq Mean Sq F value  Pr(>F)
## Locality     3 17.172   5.724  14.410 5.4e-06 ***
## LandUse      2  1.024   0.512   1.289    0.29
## Residuals   30 11.916   0.397
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Or even more, we may also need to know the interaction between factors.

```
ANOVA <- aov(pH ~ Locality * LandUse, data=Africa_env)
summary(ANOVA)
```

```
##                  Df Sum Sq Mean Sq F value   Pr(>F)
## Locality          3 17.172   5.724  14.865 1.12e-05 ***
## LandUse           2  1.024   0.512   1.329    0.283
## Locality:LandUse  6  2.675   0.446   1.158    0.361
## Residuals        24  9.242   0.385
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

## 3.2 Post-hoc Contrast

We come back to the first example and we test the effect of locality in the soil reaction (measured as pH). Of course, you may check previously the normality of residuals and the homoscedasticity of variances, but this is just one example. Here the ANOVA tell us "there is a difference in the soil reaction depending on sampling locality" but we do not know, which localities have soil pH values differetn to which ones. To know it, we require a *post-hoc* contrast. In this session we will use the Tukey's honest significant difference (function `TukeyHSD()`).

```
ANOVA <- aov(pH ~ Locality, data=Africa_env)
summary(ANOVA)
```

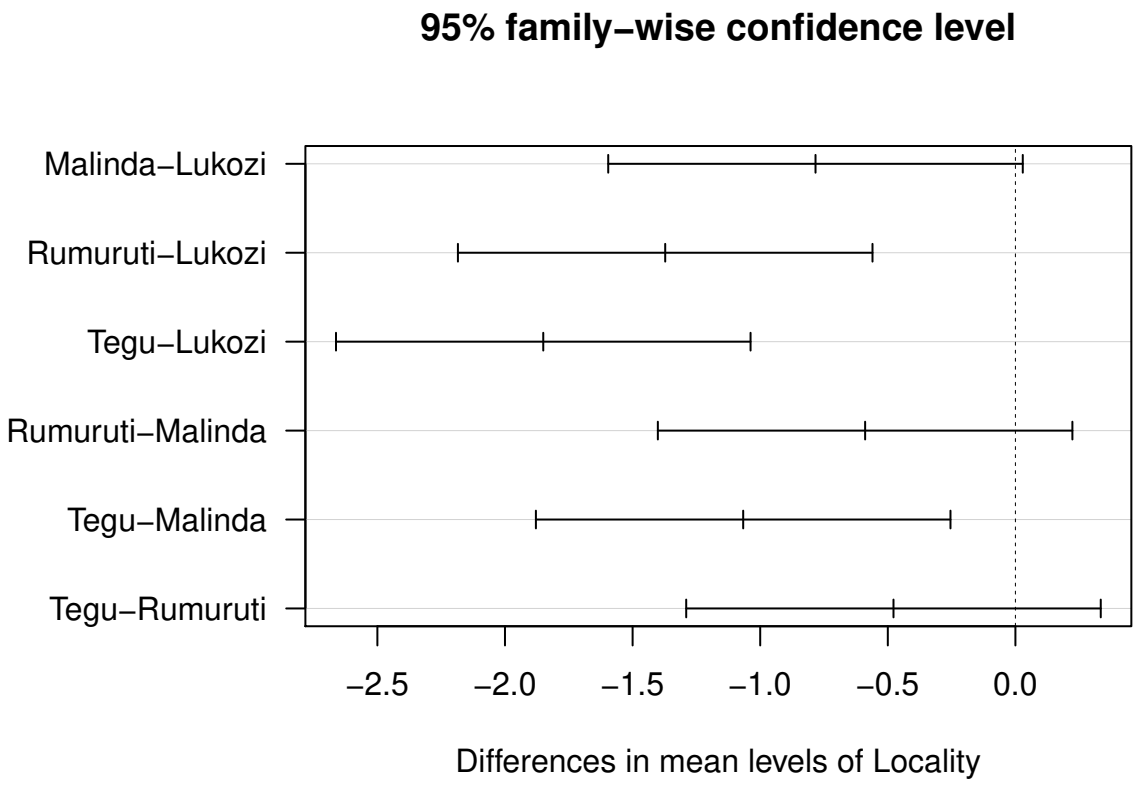
```
##             Df Sum Sq Mean Sq F value   Pr(>F)
## Locality     3  17.17   5.724   14.15 4.82e-06 ***
## Residuals   32  12.94   0.404
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Tukey <- TukeyHSD(ANOVA)
Tukey
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = pH ~ Locality, data = Africa_env)
##
## $Locality
##                          diff       lwr         upr     p adj
## Malinda-Lukozi    -0.7833333 -1.595514  0.02884776 0.0619263
## Rumuruti-Lukozi   -1.3722222 -2.184403 -0.56004113 0.0003764
## Tegu-Lukozi       -1.8500000 -2.662181 -1.03781891 0.0000038
## Rumuruti-Malinda -0.5888889 -1.401070  0.22329220 0.2225102
## Tegu-Malinda      -1.0666667 -1.878848 -0.25448558 0.0062137
## Tegu-Rumuruti     -0.4777778 -1.289959  0.33440331 0.3963421
```

In the output of `TukeyHSD()` you may look for those pairwise comparissons where both, the upper and the lower limit of the difference are either negative or positive. If the mentioned values have different symbols, the difference is not significant. That can be also checked in the column showing the p-values (**p adj**). There is also a graphic option to show the results of `TukeyHSD()`.

```
par(las=1, mar=c(5,10,5,1))
plot(Tukey)
```




For more details on those topics, review Logan (2010) and Hörmann and Unkel (2015).

## Exercises

- ”Is there a difference in the soil pH between sampling localities?” This time you will check first the normal distribution of residuals and the homogeneity of variances between localities.
- Is there differences between applied insecticides in the amount of insects (survivors) found in experimental plots? Use the data set **InsectSprays**.
- Apply a factorial ANOVA to the data set **ToothGrowth**.
- Now try the same with your own data set.

# 4 Optimihed R Codes

This session is regarding diverse alternatives to get automatical procedures in **R**.

## 4.1 Source Code

One strategy for automatizing process is to write command lines in a script and execute it via `source()`. For this option, you have to make sure, all required objects are either available in your session or created by the source code.

## 4.2 Loops

Loops are another way to optimize codes, especially when repeating routines or inserting switches or conditions. The most common loop is using the `for()` statement. Note the use of curly brakets.

```
for(i in letters[1:5]) {
    print(i)
}
```

```
## [1] "a"
## [1] "b"
## [1] "c"
## [1] "d"
## [1] "e"
```

By using the `if()` statement, you may be able to modify the behaviour of the loop according to a condition.

```
for(i in letters[1:5]) {
    if(i == "c")
        print("c is my favourite letter!") else
        print(i)
}
```

```
## [1] "a"
## [1] "b"
## [1] "c is my favourite letter!"
## [1] "d"
## [1] "e"
```

Despite the advantages of handling loops, for big data sets they may consume a lot of time and therefore it is advisable to avoid them, when possible (see Ligges and Fox, 2008).

## 4.3 Functions

As mentioned before, functions are also objects in **R**. You can define own functions, whereby the use of functions in functions is also possible. For example, defining a function for the logarithmic transformation.

```
log_trans <- function(x, ...) {
    if(any(x <= 0))
        x <- x + (0 - min(x)) + 0.1 # rescale to avoid zeros and negative values
    return(log(x, ...))
}
```

```r
# Fitive variable
Var <- c(-1, 0, 5, 10)
# Transforamtion (exp(1) as base is the default)
log_trans(Var)
```

```
## [1] -2.30258509  0.09531018  1.80828877  2.40694511
```

```r
# Without re-scaling
log(Var)
```

```
## Warning in log(Var): NaNs produced
```

```
## [1]      NaN     -Inf 1.609438 2.302585
```

```r
# Changing the base of logarithm
log_trans(Var, base=10)
```

```
## [1] -1.00000000  0.04139269  0.78532984  1.04532298
```

The settings of the functions are called **arguments**. Note that the three dots indicate arguments that will be passed to a function included in the definition.

In the context of object oriented programming, functions can be defined for specific objects. Object-oriented programming is implemented in **R** through **S3**, **S4**, and **R6**, for instance. Definitions of functions for specific object classes are called **methods** in **R** (Wickham, 2014; Genolini, 2008).

### 4.4 Packages

The most advanced way to optimize R code is creating packages. Packages may contain data and/or functions. The important addition of packages is that the developer of a package is oblied to document and provide examples on functions and data. For more details, take a look on Wickham (2014), Wickham (2015), and Ligges (2003).

**Exercises**

- Use a loop for calculating the mean value of every numeric variable in **iris**.

## 5 Documenting your work with R Markdown

R Markdown allows you to create documents that serve as a neat record of your analysis. It was designed for easier reproducibility, since both the computing code and narratives are in the same document, and results are automatically generated from the source code.

```
The idea should be simple enough: interweave narratives with code in a document,knit the
document to dynamically generate results from the code, and you will get a report.
```

`RMarkdown` uses Markdown syntax. `Markdown` is a very simple 'markup' language which provides methods for creating documents with headers, images, links etc. from plain text files, while keeping the original plain text file easy to read. You can convert Markdown documents to many other file types like `.html`,`.pdf`, or `.docx`.

For detailed information please refere to

- Xie et al. (2018) R Markdown: The Definitive Guide
- RStudio RMarkdown Documentation

### 5.1 Markdown Syntax

Section headers can be written after a number of pound signs, e.g.,

```
# First-level header

## Second-level header

### Third-level header
```

If you do not want a certain heading to be numbered, you can add `{-}` or `{.unnumbered}` after the heading, e.g.,

```
# Preface {-}
```

Unordered list items start with `*`, `-`, or `+`, and you can nest one list within another list by indenting the sub-list, e.g.,

```
- one item
- one item
- one item
    - one more item
    - one more item
    - one more item
```

The output is:

- one item
- one item
- one item
    - one more item
    - one more item
    - one more item

Ordered list items start with numbers (you can also nest lists within lists), e.g.,

```
1. the first item
2. the second item
3. the third item
    - one unordered item
    - one unordered item
```

The output does not look too much different with the Markdown source:

1. the first item
2. the second item
3. the third item
    - one unordered item
    - one unordered item

## 5.2 Download R Markdown

```
install.packages("rmarkdown")
library(rmarkdown)
```

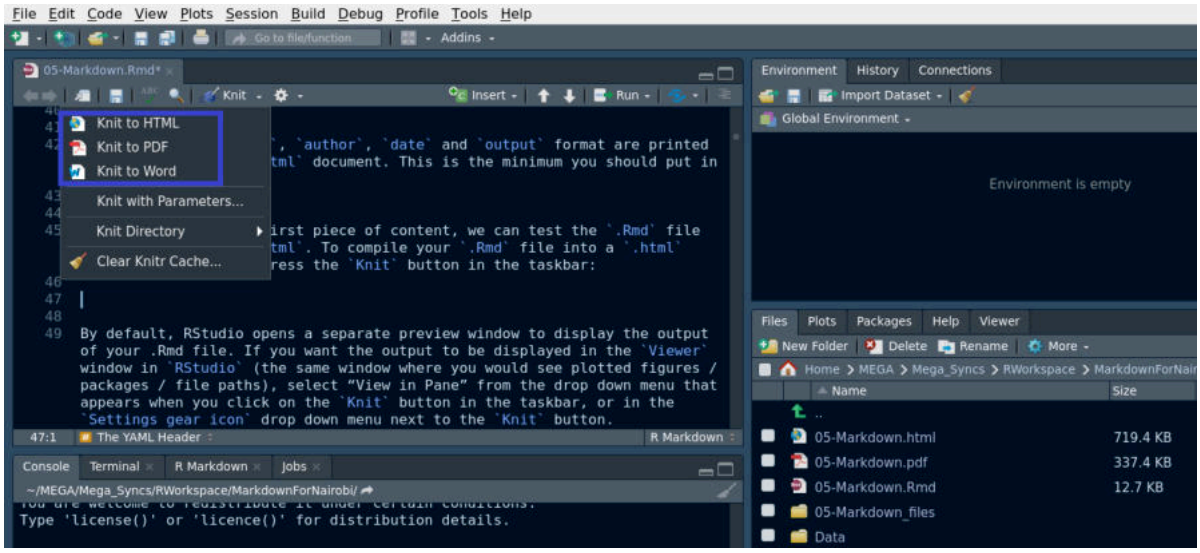
## 5.3 The different parts of an R Markdown file

### 5.3.1 The YAML Header

At the top of any `RMarkdown` script is a `YAML` header section enclosed by `---`. By default this includes a title, author, date and the file type you want to output to. Many other options are available for different functions and formatting, see here for `.html` options and here for `.pdf` options. Rules in the header section will alter the whole document.

```
---
title: "R Workshop in Nairobi"
author: Your Name
date: 05/08/2019
output: html_document
---
```

To compile your `.Rmd` file into a `.html` or `.pdf` or any other supported document, you should press the `Knit` button in the taskbar:




### 5.3.2 Code Chunks

Below the `YAML` header is the space where you will write your code, accompanying explanation and any outputs. Code that is included in your `.Rmd` document should be enclosed by three backwards apostrophes ``` (grave accents!). These are known as code chunks and look like this:

````
```r
norm <- rnorm(100, mean = 0, sd = 1)
```
````

for that code chunk. The code chunk above says that the code is R code.

You can insert an R code chunk either using the RStudio toolbar (the `Insert` button) or the keyboard shortcut `Ctrl + Alt + I` (`Cmd + Option + I` on macOS).

You have fine control over all these output via chunk options, which can be provided inside the curly braces (between ```{r and }). Chunk options are separated by commas, e.g.,

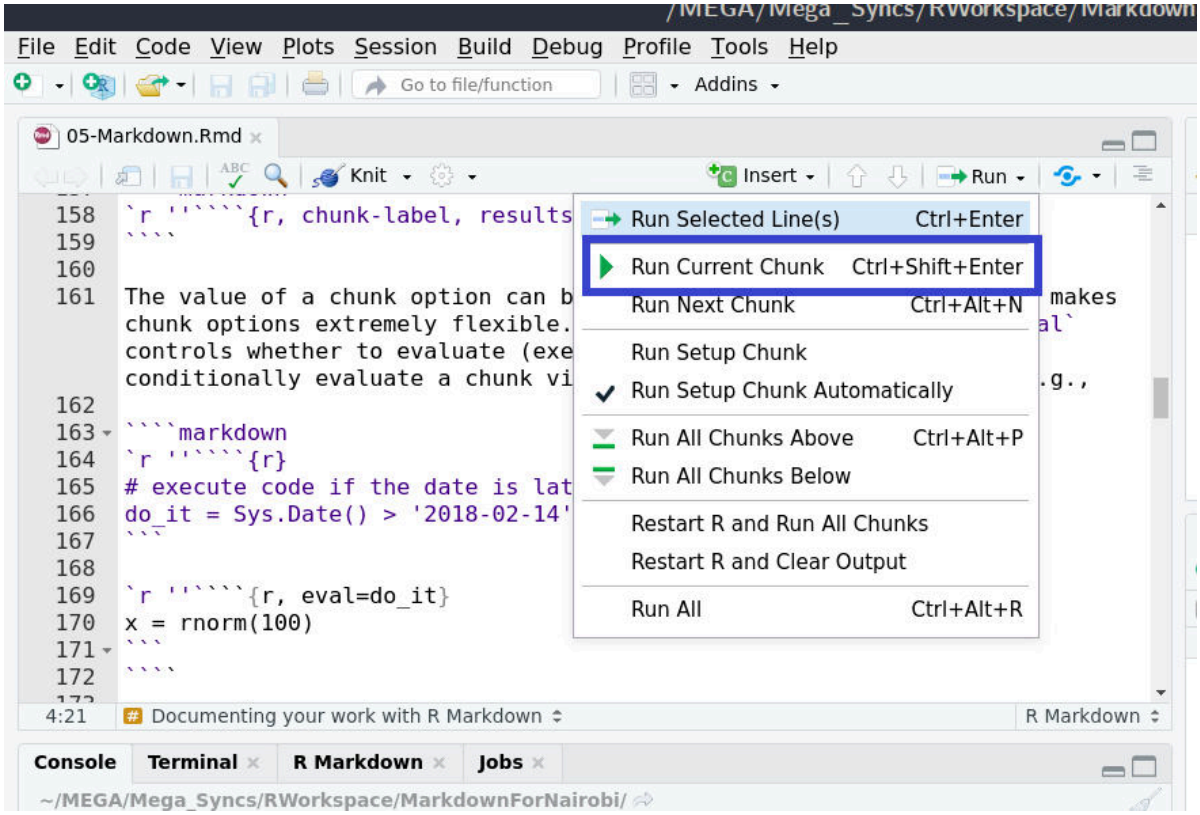
````
```{r, chunk-label, results='hide', fig.height=4}
```
````

The value of a chunk option can be an arbitrary R expression, which makes chunk options extremely flexible. For example, the chunk option `eval` controls whether to evaluate (execute) a code chunk, and you may conditionally evaluate a chunk via a variable defined previously, e.g.,

````
```{r}
# execute code if the date is later than a specified day
do_it = Sys.Date() > '2019-07-01'
```

```{r, eval=do_it}
x = rnorm(100)
```
````

You can run an individual chunk of code at any time by placing your cursor inside the code chunk and selecting `Run -> Run Current Chunk`:




### 5.3.3 Chunk Options

There are a large number of chunk options in **knitr** documented at https://yihui.name/knitr/options. Some example Below:

- `eval`: Whether to evaluate a code chunk.
- `echo`: Whether to echo the source code in the output document.
- `warning`, `message`, and `error`: Whether to show warnings, messages, and errors in the output document.
- `cache`: Whether to enable caching. If caching is enabled, the same code chunk will not be evaluated the next time the document is compiled (if the code chunk was not modified), which can save you time.
- `fig.width` and `fig.height`: The (graphical device) size of R plots in inches. R plots in code chunks are first recorded via a graphical device in **knitr**, and then written out to files.
- `out.width` and `out.height`: The output size of R plots in the output document. These options may scale images. You can use percentages, e.g., `out.width = '80%'` means 80% of the page width.
- `fig.align`: The alignment of plots. It can be `'left'`, `'center'`, or `'right'`.
- `fig.cap`: The figure caption.

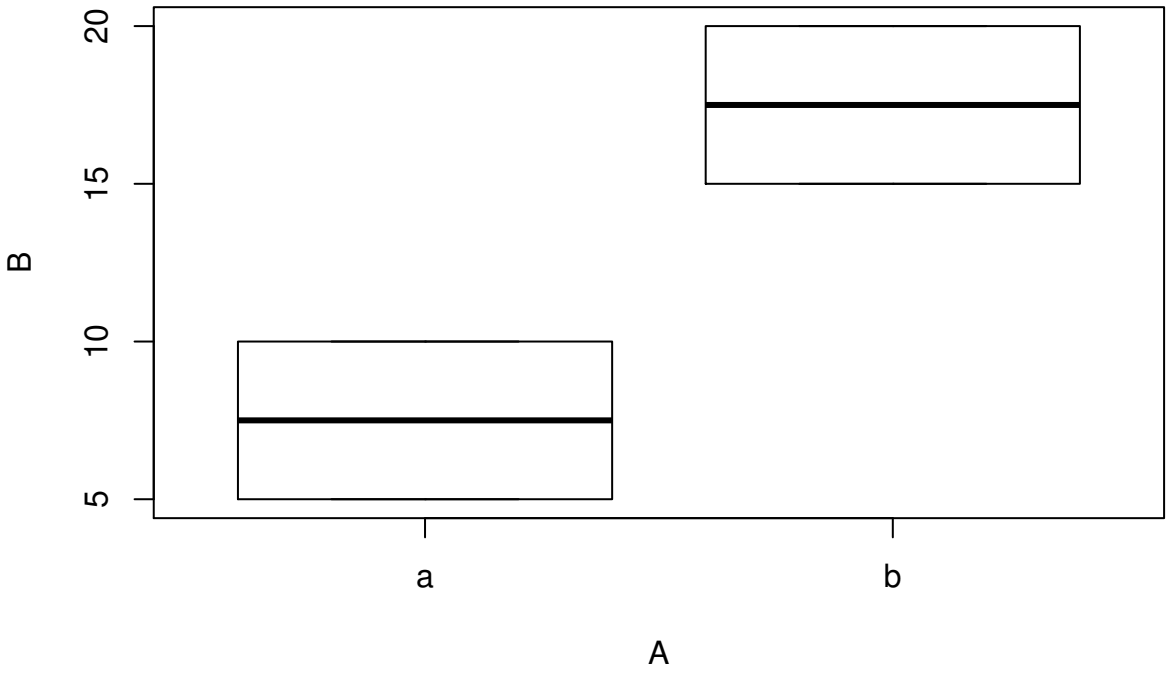
### 5.3.4 More on Code Chunks

It's important to remember when you are creating an `RMarkdown` file that if you want to run code that refers to an object, for example:

````
```r
plot(dataframe)
```
````

you must include instructions showing what `dataframe` is, just like in a normal R script. For example:

```
A <- c("a", "a", "b", "b")
B <- c(5, 10, 15, 20)
dataframe <- data.frame(A, B)
plot(dataframe)
```




Or if you are loading a dataframe from a `.csv` file, you must include the code in the `.Rmd`:

```
```r
dataframe <- read.csv("~/Desktop/MyCode/Mydataframe.csv")
```
```

Similarly, if you are using any packages in your analysis, you will have to load them in the `.Rmd` file using `library()` as in a normal `R` script.

```
```r
library(dplyr)
```
```

### 5.3.5 Inserting Figures

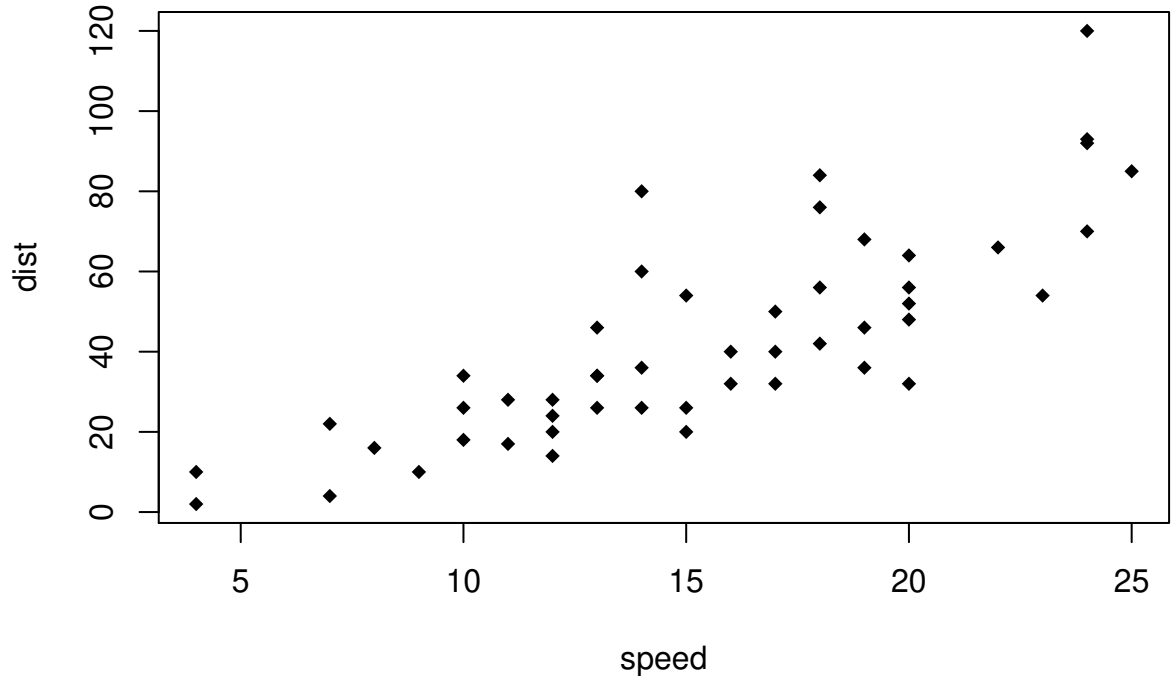
Inserting a graph into `RMarkdown` is easy, the more energy-demanding aspect might be adjusting the formatting.

By default, `RMarkdown` will place graphs by maximising their height, while keeping them within the margins of the page and maintaining aspect ratio. To manually set the figure dimensions, you can insert an instruction into the curly braces:

```
```{r, fig.width = 4, fig.height = 3}
A <- c("a", "a", "b", "b")
B <- c(5, 10, 15, 20)
dataframe <- data.frame(A, B)
plot(dataframe)
```
```

By default, figures produced by R code will be placed immediately after the code chunk they were generated from. For example:

```
plot(cars, pch = 18)
```




You can provide a figure caption using `fig.cap` in the chunk options. In the case of PDF output, such figures will be automatically numbered. If you also want to number figures in other formats (such as HTML), please see the **bookdown** package in Chapter **??** (in particular, see Section **??**).

To place multiple figures side-by-side from the same code chunk, you can use the `fig.show='hold'` option along with the `out.width` option. Figure 1 shows an example with two plots, each with a width of `50%`.

```
par(mar = c(4, 4, .2, .1))
plot(cars, pch = 19)
plot(pressure, pch = 17)
```

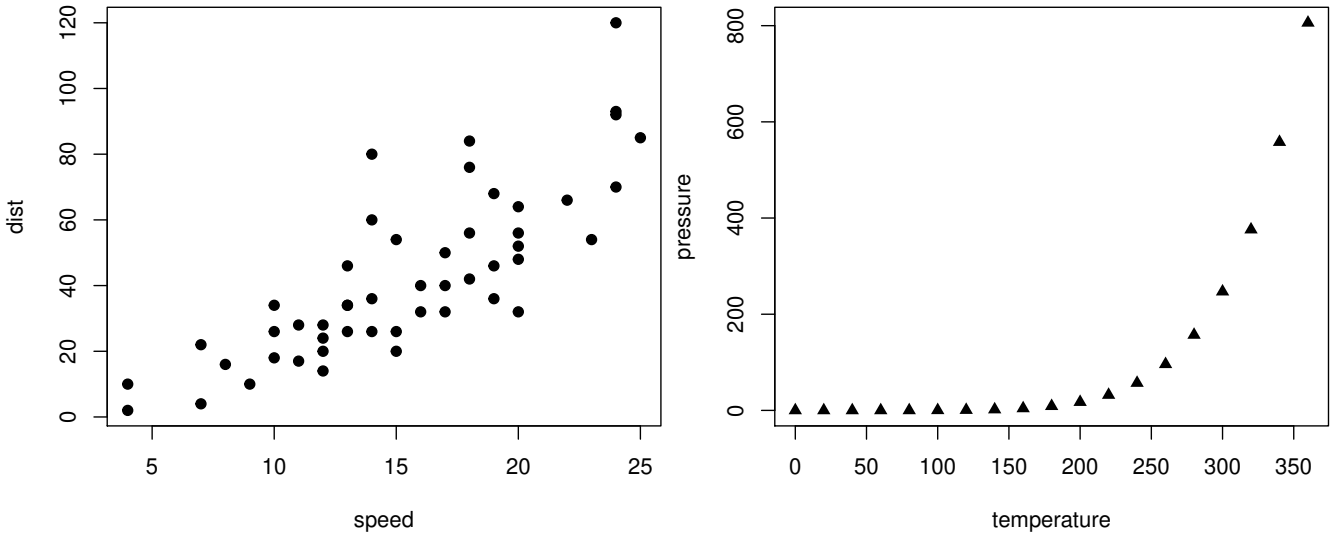



Figure 1: Two plots side-by-side.

If you want to include a graphic that is not generated from R code, you may use the `knitr::include_graphics()`

function.

```
```{r include-graphics,fig.cap='CRC logo', out.width='50%', fig.align='center'}
knitr::include_graphics('images/fig1.png')
```
```


Figure 2: CRC logo

### 5.3.6 Math expressions

Inline LaTeX equations can be written in a pair of dollar signs using the LaTeX syntax, e.g., `$f(k) = {n \choose k} p^{k} (1-p)^{n-k}$` (actual output: $f(k) = \binom{n}{k} p^k (1-p)^{n-k}$); math expressions of the display style can be written in a pair of double dollar signs, e.g., `$$f(k) = {n \choose k} p^{k} (1-p)^{n-k}$$`, and the output looks like this:

$$f\left(k\right) = \binom{n}{k} p^k \left(1-p\right)^{n-k}$$

### 5.3.7 Tables

The easiest way to include tables is by using `knitr::kable()`, which can create tables for HTML, PDF and Word outputs.[1] Table captions can be included by passing `caption` to the function, e.g.,

```
```{r tables-mtcars}
knitr::kable(iris[1:5, ], caption = 'A caption')
```
```

## 5.4 Presentations

For documents, the basic units are often sections. For presentations, the basic units are slides. A section in the Markdown source document often indicates a new slide in the presentation formats.

### 5.4.1 ioslides presentation

To create an ioslides presentation from R Markdown, you specify the `ioslides_presentation` output format in the YAML metadata of your document. You can create a slide show broken up into sections by using the `#` and `##` heading tags (you can also create a new slide without a header using a horizontal rule (`---`). For example here is a simple slide show:

```
---
title: "Nairobi R Workshop"
author: Mr Bond
date: August 05, 2019
output: ioslides_presentation
---

# In the morning

## Getting up

- Turn off alarm
- Get out of bed

## Breakfast
```

[1] You may also consider the **pander** package. There are several other packages for producing tables, including **xtable**, **Hmisc**, and **stargazer**, but these are generally less compatible with multiple output formats.

```
- Eat eggs
- Drink coffee

# In the evening

## Dinner

- Eat spaghetti
- Drink wine

---

```{r, cars, fig.cap="A scatterplot.", echo=FALSE}
plot(cars)
```

## Going to sleep

- Get in bed
- Count sheep
```

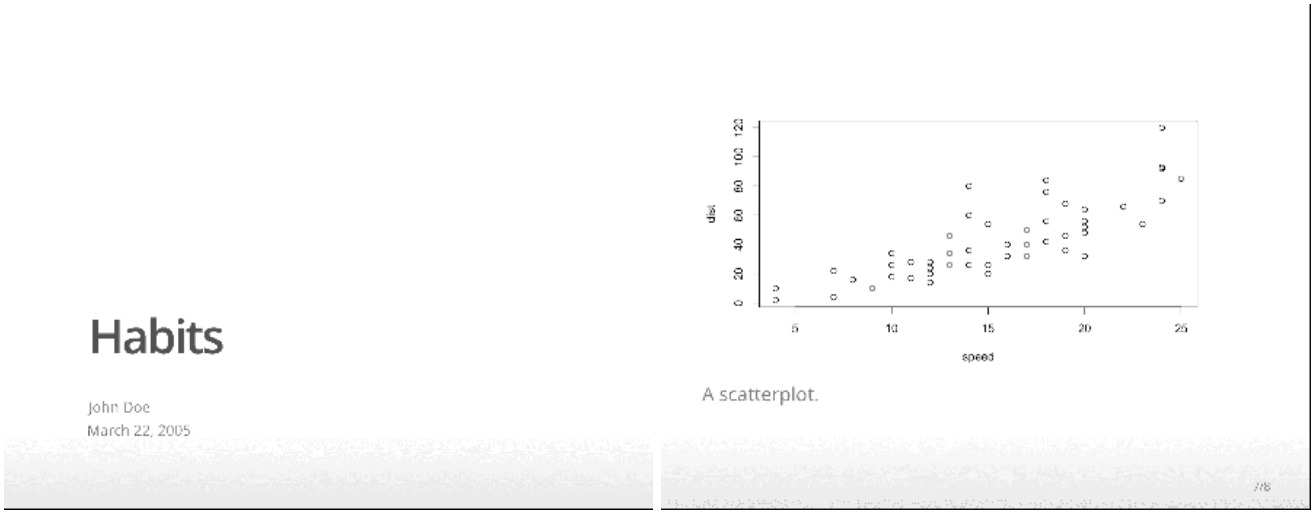



Figure 3: Two sample slides in an ioslides presentation.

You can add a subtitle to a slide or section by including text after the pipe (|) character. For example:

```
## Getting up | What I like to do first thing
```

### 5.4.2 Display modes

The following single character keyboard shortcuts enable alternate display modes:

- `'f'`: enable fullscreen mode
- `'w'`: toggle widescreen mode
- `'o'`: enable overview mode
- `'h'`: enable code highlight mode
- `'p'`: show presenter notes

Pressing `Esc` exits all of these modes. See the sections below on *Code Highlighting* and *Presenter Mode* for additional detail on those modes.

### 5.4.3 Incremental bullets

You can render bullets incrementally by adding the `incremental` option:

```
---
output:
  ioslides_presentation:
    incremental: true
---
```

If you want to render bullets incrementally for some slides but not others you can (ab)use this syntax for blockquotes:

```
> - Eat eggs
> - Drink coffee
```

### 5.4.4 Presentation size

You can display the presentation using a wider form factor using the `widescreen` option. You can specify that smaller text be used with the `smaller` option. For example:

```
---
output:
  ioslides_presentation:
    widescreen: true
    smaller: true
---
```

You can also enable the `smaller` option on a slide-by-slide basis by adding the `.smaller` attribute to the slide header:

```
## Getting up {.smaller}
```

# References


Anderson, E. (1935). The irises of the Gaspe Peninsula. *Bulletin of the American Iris Society*, 59:2–5.

Burns, P. (2011). *The R inferno*.

Crawley, M. J. (2007). *The R Book*. Wiley, Chichester.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188.

Genolini, C. (2008). A (not so) short introduction to s4.

Greenland, S., Senn, S. J., Rothman, K. J., Carlin, J. B., Poole, C., Goodman, S. N., and Altman, D. G. (2016). Statistical tests, P values, confidence intervals, and power: a guide to misinterpretations. *European Journal of Epidemiology*, 31(4):337–350.

Hörmann, G. and Unkel, I. (2015). *Analysis of ecological data with R*. Christian-Albrechts-Universität zu Kiel, Kiel.

Kamiri, H. W., Kreye, C., and Becker, M. (2013). Assessing selected soil properties responses to wetland cultivation in floodplain wetlands of east africa. *International Journal of AgriScience*, 3(11):825–837.

Ligges, U. (2003). Package management. *R News*, 3(3):37–39.

Ligges, U. and Fox, J. (2008). How can I avoid this loop or make it faster? *R News*, 8(1):46–50.

Logan, M. (2010). *Biostatistical design and analysis using R. A practical guide*. Wiley-Blackwell, Chichester.

Wickham, H. (2014). *Advanced R*. Routledge.

Wickham, H. (2015). *R Packages*. O'Reilly, Cambridge.

Xie, Y., Allaire, J., and Grolemund, G. (2018). *R markdown: The definitive guide*. Taylor & Francis Ltd.

Zuur, A. F., Ieno, E. N., and Elphick, C. S. (2010). A protocol for data exploration to avoid common statistical problems. *Methods in Ecology and Evolution*, 1(1):3–14.

## Notes