R for life sciences. Chapter 3: Basic graphics and data

true

2019-07-10

Cite as: Alfonso Garmendia (2019) R for life sciences. Chapter 2: Operations in R. http://personales.upv.es/algarsal/Documentation/Garmendia-R-Tutorial-03_Graphics.html

available in PDF and EPUB

Written in Rmarkdown, using Rstudio and pandoc.

Basic graphics and data

Basic graphics with one variable

One of the main reasons to use R instead of statistical programs is for its strong graphical capabilities. To see some of these capabilities, write **demo(graphics)**.

Basic graph types are density plots, dot plots, bar charts, line charts, pie charts, box-plots and scatter plots.

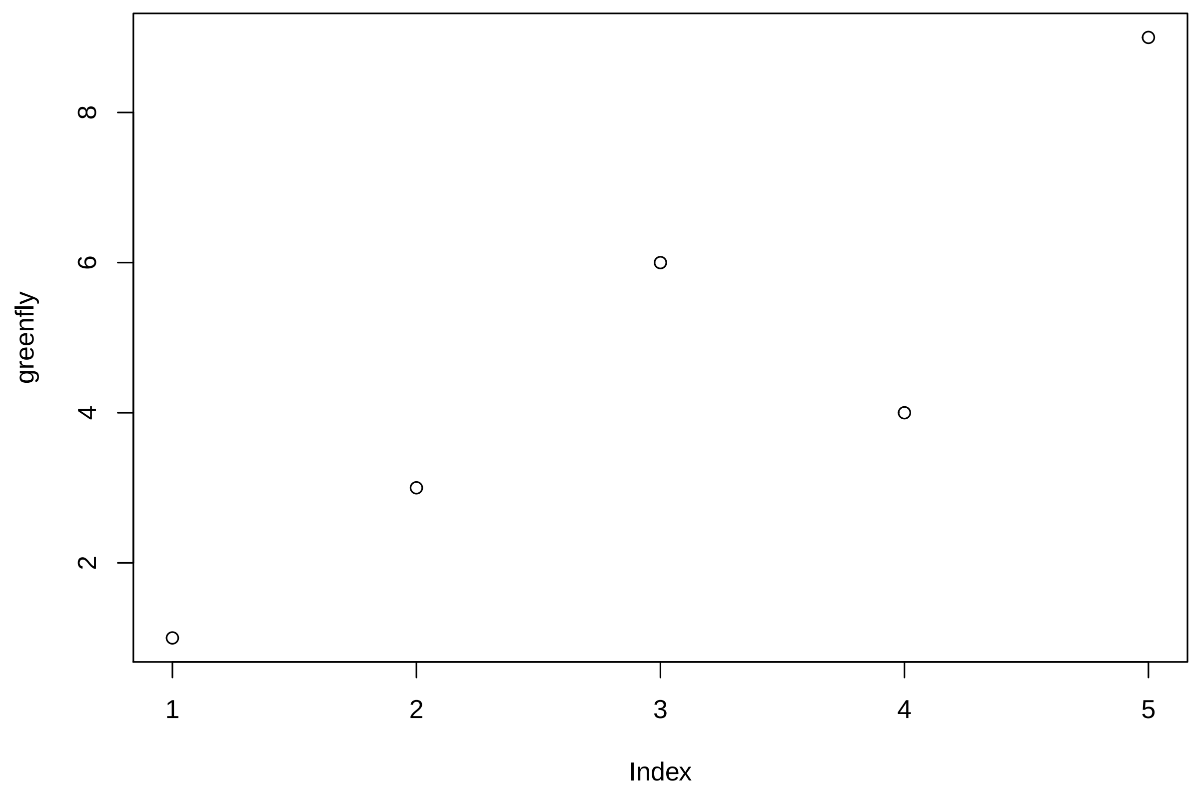
Plots in R have two types of commands, high-level commands to create the plot and low-level commands to add things to the plot, once it has been created. These low-level commands will do nothing if there is not an active plot.

Some of the most used low-level commands are:

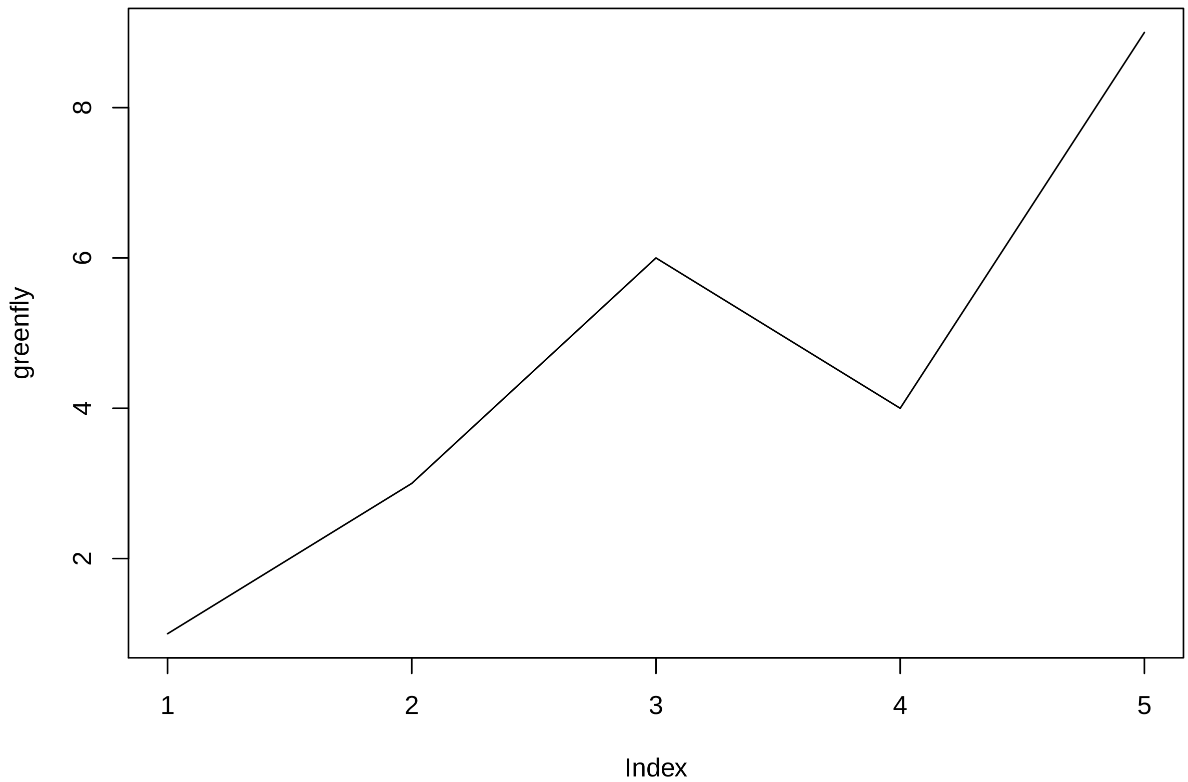
points () : Add points
lines() : Add a line graph
abline () : Add a straight line
title() : Add a title
legend() : Add a legend
text() : Add a text string at the desired coordinates into a figure.

The main primary command is **plot()**. Depending the input data, it will do the type of plot that best fit. But of course is possible to change. Looking at the help(plot) page is very advisable before start and take a look into the arguments. Changing for example the type of plot.

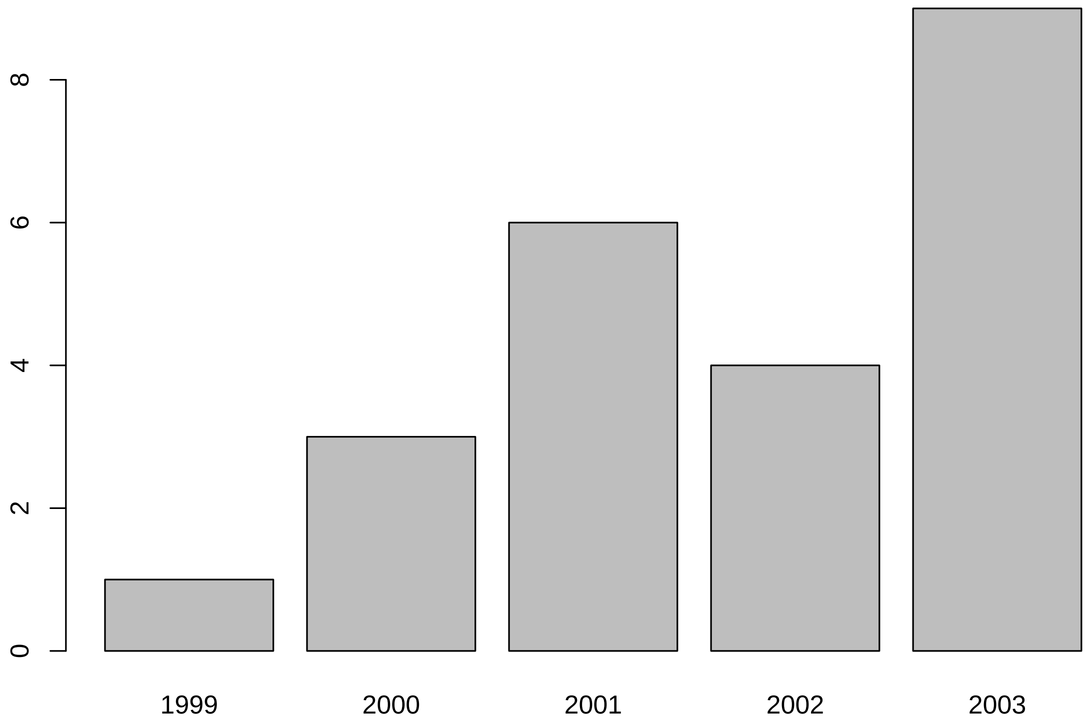
```
################ BASIC GRAPHS ###################
# Define the greenfly vector with 5 values
greenfly <- c(1, 3, 6, 4, 9)

################ POINTS ########################
# Graph the greenfly vector with all defaults
plot(greenfly)
```



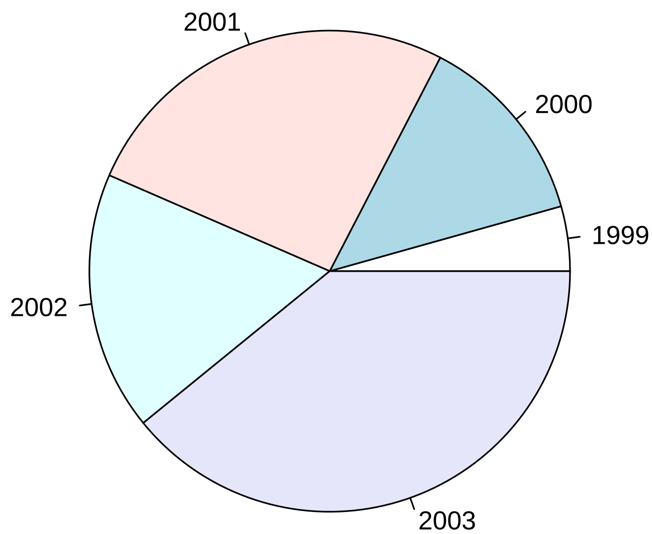

```
################ LINES  ########################
# Graph the greenfly vector with a line
plot(greenfly, type = "l")
```



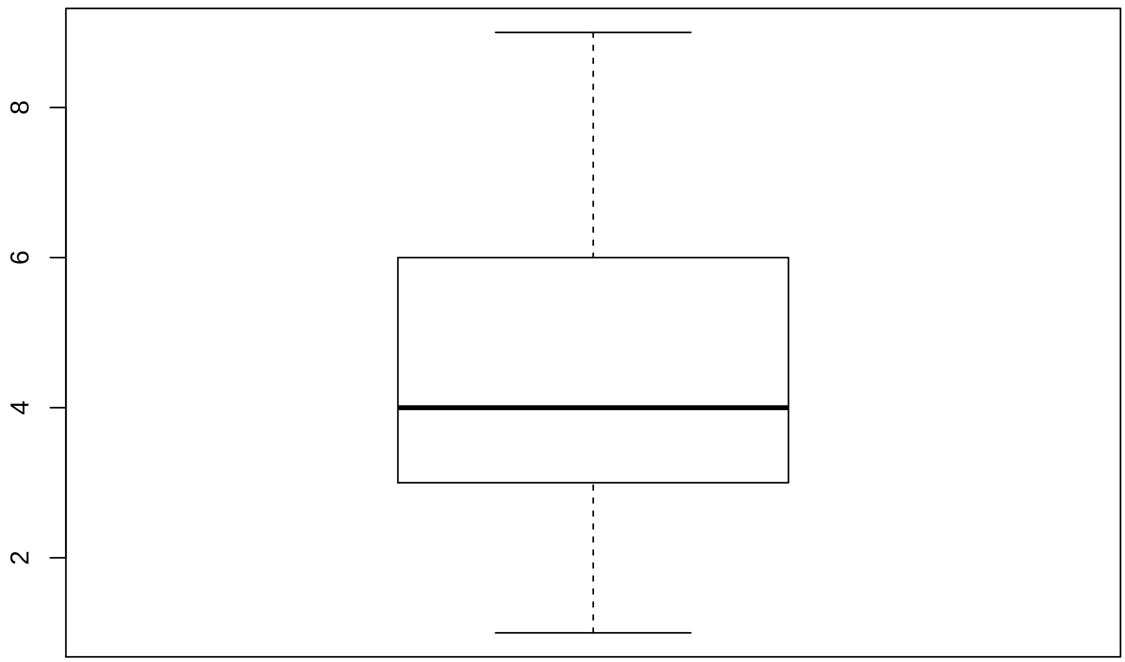

```
################ BARPLOT #######################
# Names for the bars
years <- as.character(1999:2003)
# Graph the greenfly vector with barplot
barplot(greenfly, names.arg = years)
```



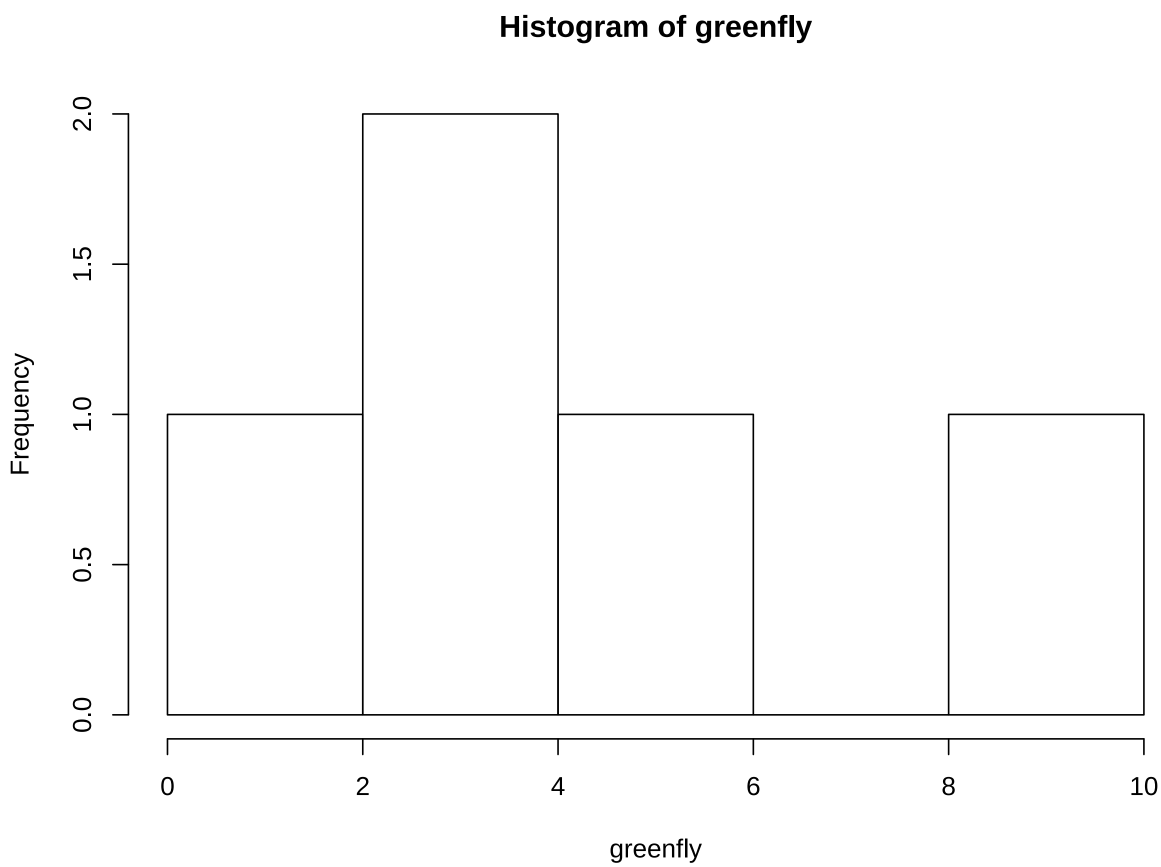

```
################ PIE PLOT #######################
# Graph the greenfly vector with a line
pie(greenfly, labels = years)
```



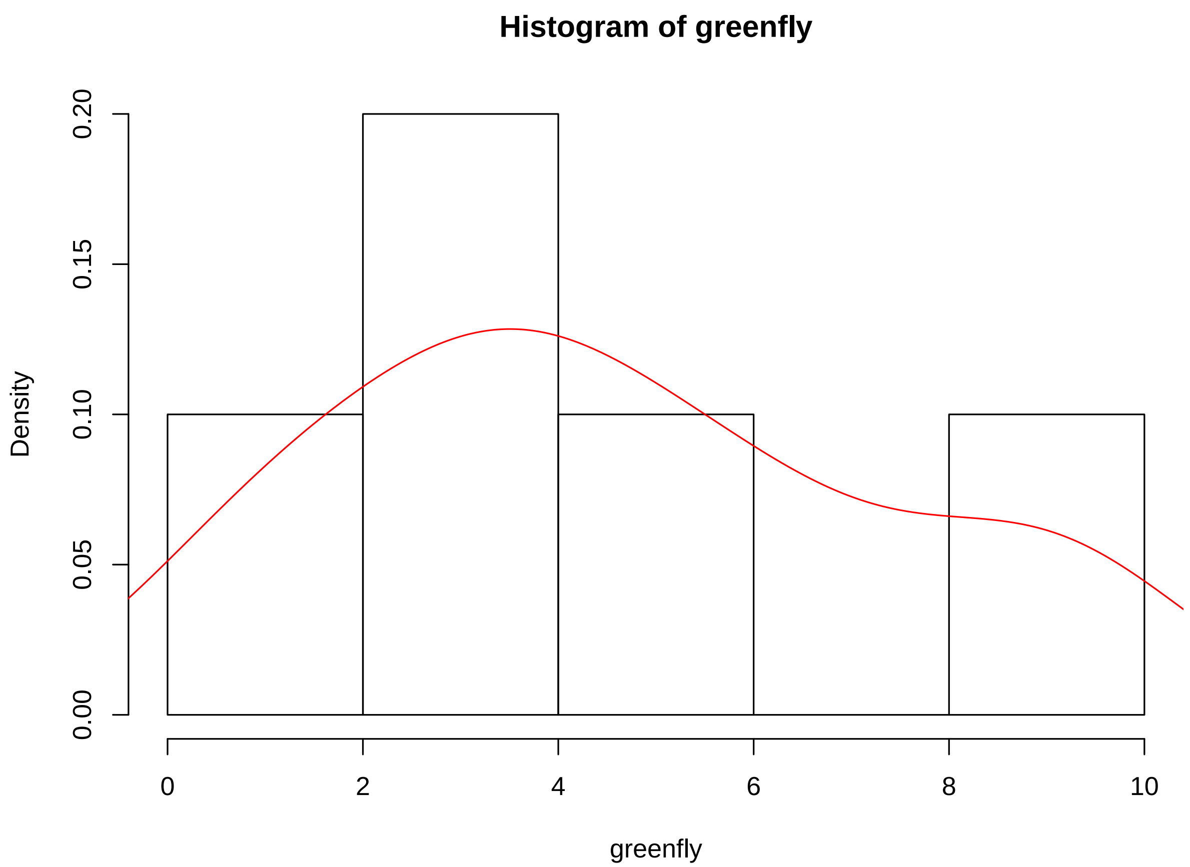

```
################ BOX PLOT #######################
# Graph the greenfly vector with a boxplot
boxplot(greenfly)
```



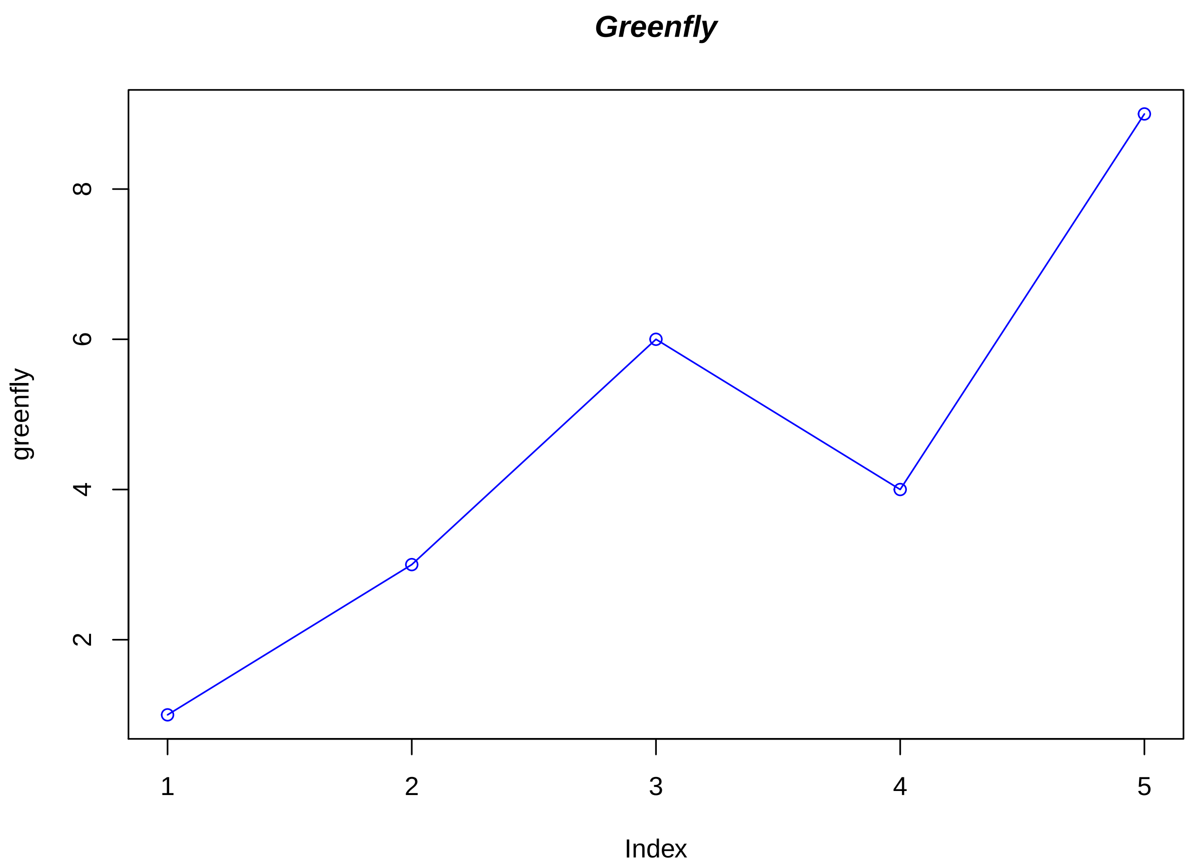

```
####### HISTOGRAM WITH FREQUENCIES ##############
# Graph the greenfly vector with a histogram with frequencies
hist(greenfly)
```



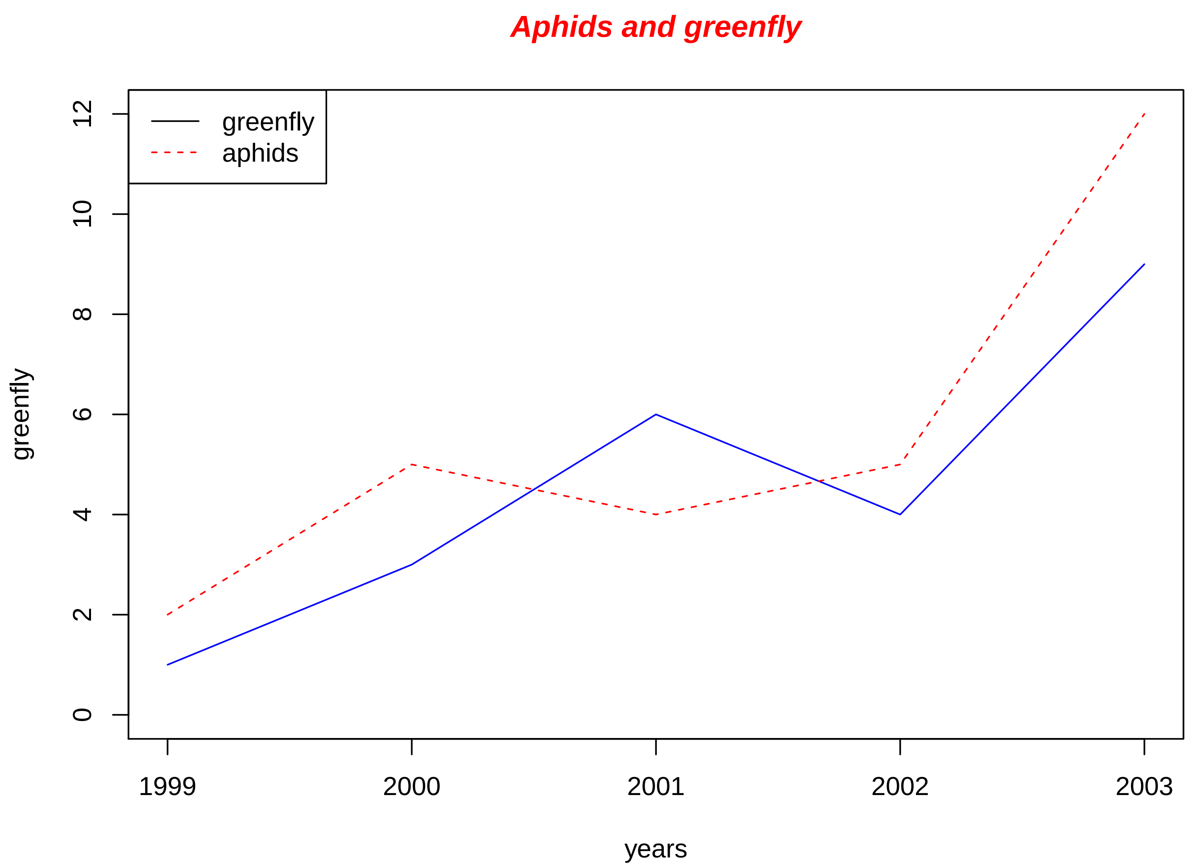

```
##### HISTOGRAM WITH PROBABILITY DENSITIES ######
# Graph the greenfly vector with a histogram with probability densities
hist(greenfly, freq = F)
# ADD other graph with density
lines(density(greenfly), col = "red")
```




```
################ LINES AND POINTS ###############
# Graph greenfly using blue points overlayed by a line
# and with bold/italic title
plot(greenfly, type = "o", col = "blue"
     , main = "Greenfly", font.main = 4)
```




```
####### TWO GRAPHS WITH LOW-LEVEL COMMANDS #######
# Define other vector
aphids <- c(2, 5, 4, 5, 12)
#
# Graph greenfly using a y axis that ranges from 0 to 12
plot(greenfly, type = "l", col = "blue"
     , ylim = c(0,12), x = years)
#
# Add graph for aphids with red dashed line
lines(aphids, type = "l", lty = 2, col = "red", x = years)

# Add a title with red and bold/italic font
title(main = "Aphids and greenfly"
      , col.main = "red", font.main = 4)
#
# Add legend
legend("topleft", c("greenfly", "aphids")
       , col = 1:2, lty = 1:2)
```



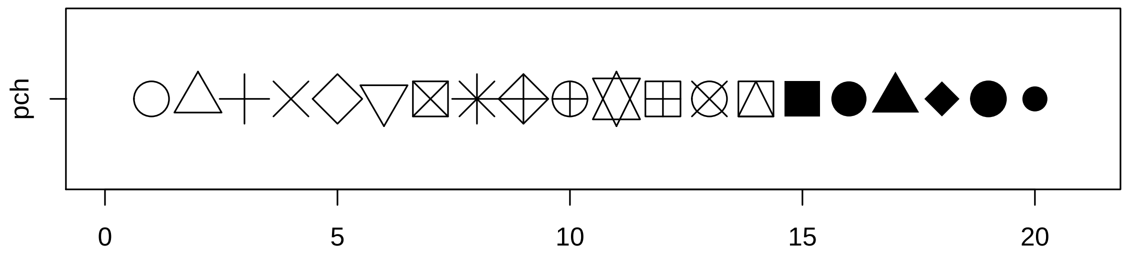
For more details on how to make graphs in R,there is a lot of accessible material into Internet, like tutorials[1], books[2] [3], and example graphics [4] [5] some of them very good ones.

Graphs that help make graphs

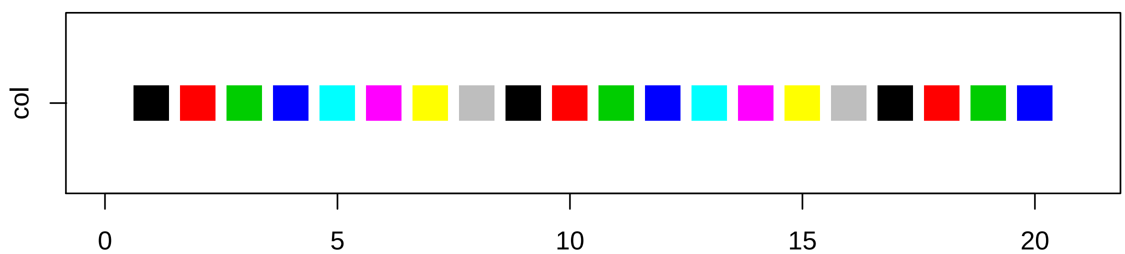
One of the problems everyone faces when making graphs in R is the use of numbers to name colors, point and line types, etc. A good idea to deal with this is to be able to make a tutorial graph like these ones:

```
##### Cheat-sheet for pch (point type) ######
plot(0, 0, xlim = c(0, 21), ylim = c(0.5, 1.5)
     , ylab = "", xlab = "", yaxt = "n")
axis(2, 1, labels = c("pch"))
for (i in 1:20) {
    points(i, 1, pch = i, cex = 3)
}
```

[1] Quick-R
[2] Chang 2013
[3] Murrel 2011
[4] r-graph-gallery
[5] Murrel



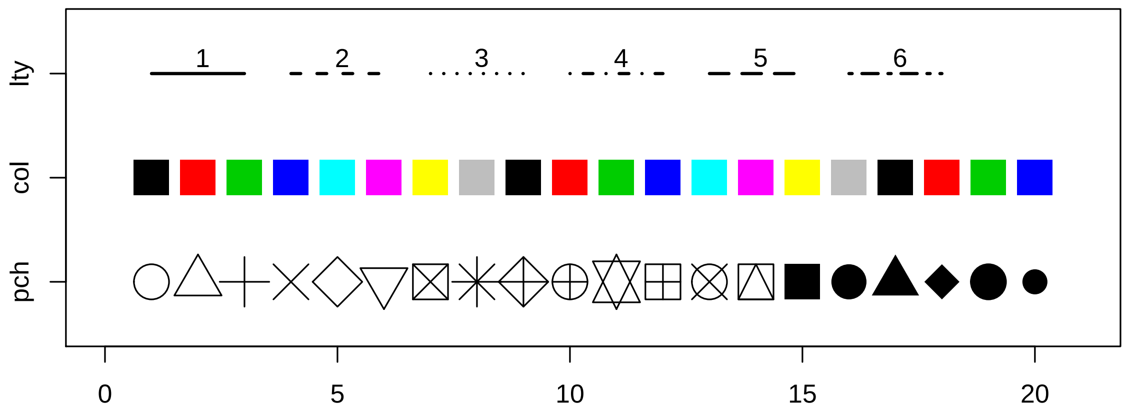

```
#### Cheat-sheet for colors ####
plot(0, 0, xlim = c(0, 21), ylim = c(0.5, 1.5)
     , ylab = "", xlab = "", yaxt = "n")
axis(2, 1, labels = c("col"))
for (i in 1:20) {
    points(i, 1, pch = 15, col = i, cex = 3)
}
```



Or combine several in one:

```
#### Cheat-sheet for ALL ####
num = 0 ; num1 = 0
plot(0, 0, xlim = c(0, 21), ylim = c(0.5, 3.5)
     , yaxt = "n", ylab = "", xlab = "")

### Add axis
axis(2, at = c(1, 2, 3), labels = c("pch", "col", "lty"))

### Fill the graph
for (i in seq(1,20)) {
  points(i, 1, pch = i, cex = 3)                        # pch
  points(i, 2, col = i, pch = 15 , cex = 3)             # col
  #lty
  if (i %in% c(seq(1, 18, 3))) {
```

```
        num = num + 1
    points(c(i, i + 2), c(3, 3)
           , col = "black"
           , lty = num, type = "l", lwd = 2)
        text(i + 1.1, 3.15, num)
  }
}
```



Data files inside R and graphs with two or more variables

We have already created data using c() [6], vector(), matrix(), data.frame() and list(), and also converted ones into others using as.factor(), as.data.frame(), etc. Other important source of data for training are the data files already housed into R packages and used for the examples in help() [7] or demo().

This data files are also very useful for training and teaching R. We have used some already (e.g. **iris**) and will use them more latter. They are also the easiest way for asking questions into forums and other webs, because it avoids all the problems of importing and exporting data.

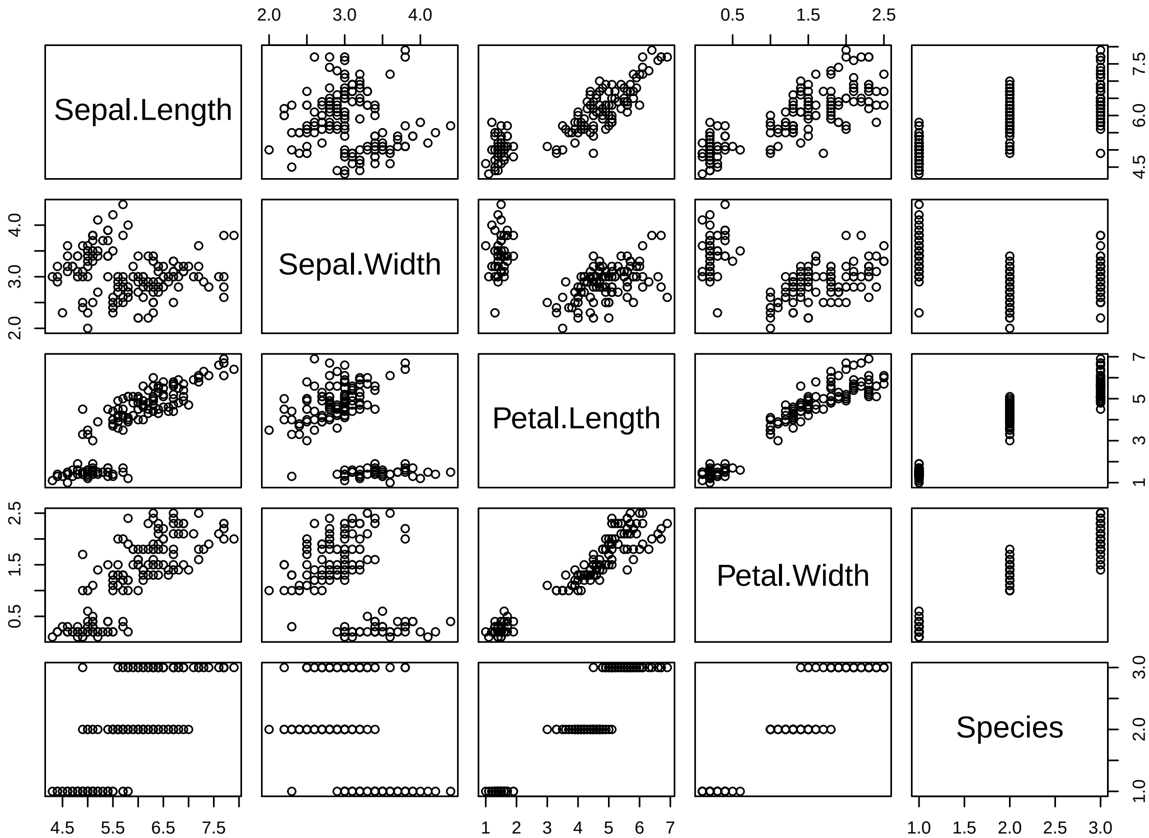
Use **data()** to see all data available from package "datasets" of from other packages.

We can explore the **plot()** possibilities with iris data frame. It uses different graphs depending on the input data.

Plot a data frame

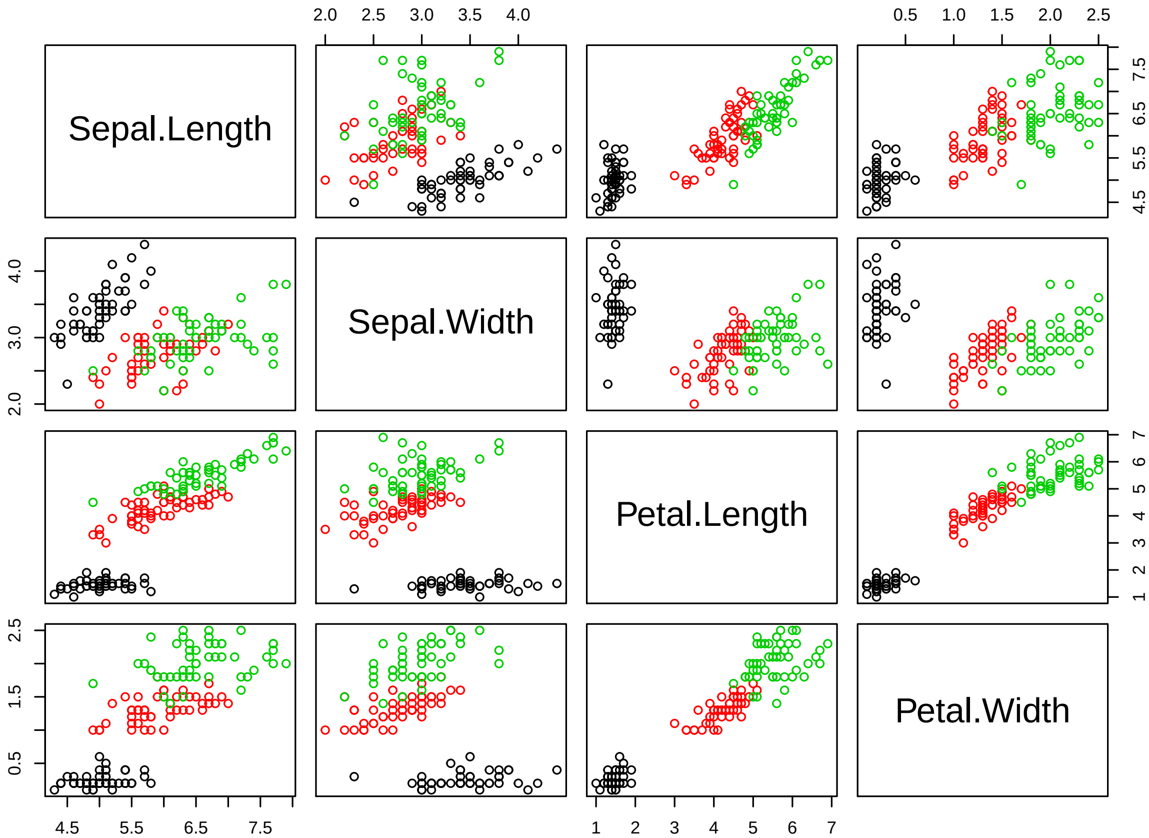
```
##### BASIC PLOTS WITH IRIS ####
# plot.data.frame() and pairs() will output same results.
plot(iris)                          # Data frame. All variables as.numeric
```

[6] **?c** to see the help page
[7] **?help** to see the help page



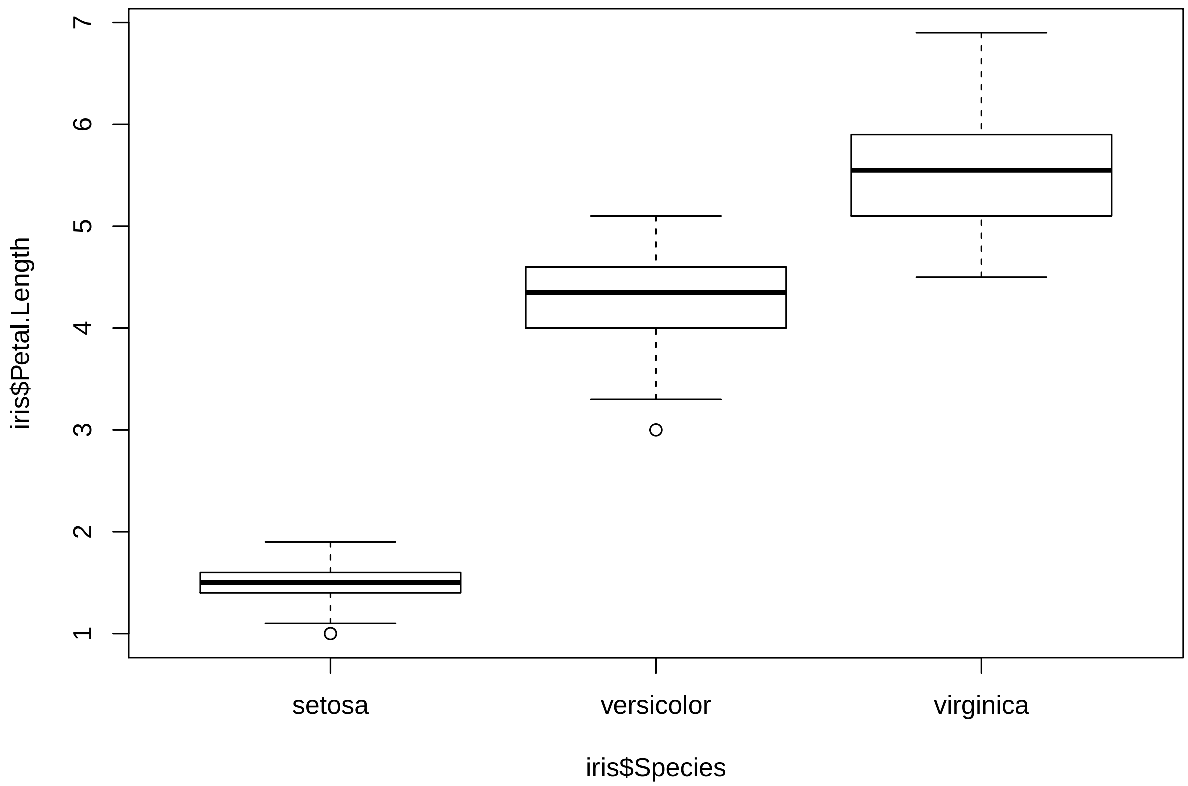
Observe that Species is a categorical variable. It has been converted to numeric, but obviously this is not the best way to represent a categorical variable. It is better this other way:

```
### Same plot with only 1:4 variables and colors by Species
plot(iris[1:4], col = iris$Species )                # colors by species
```

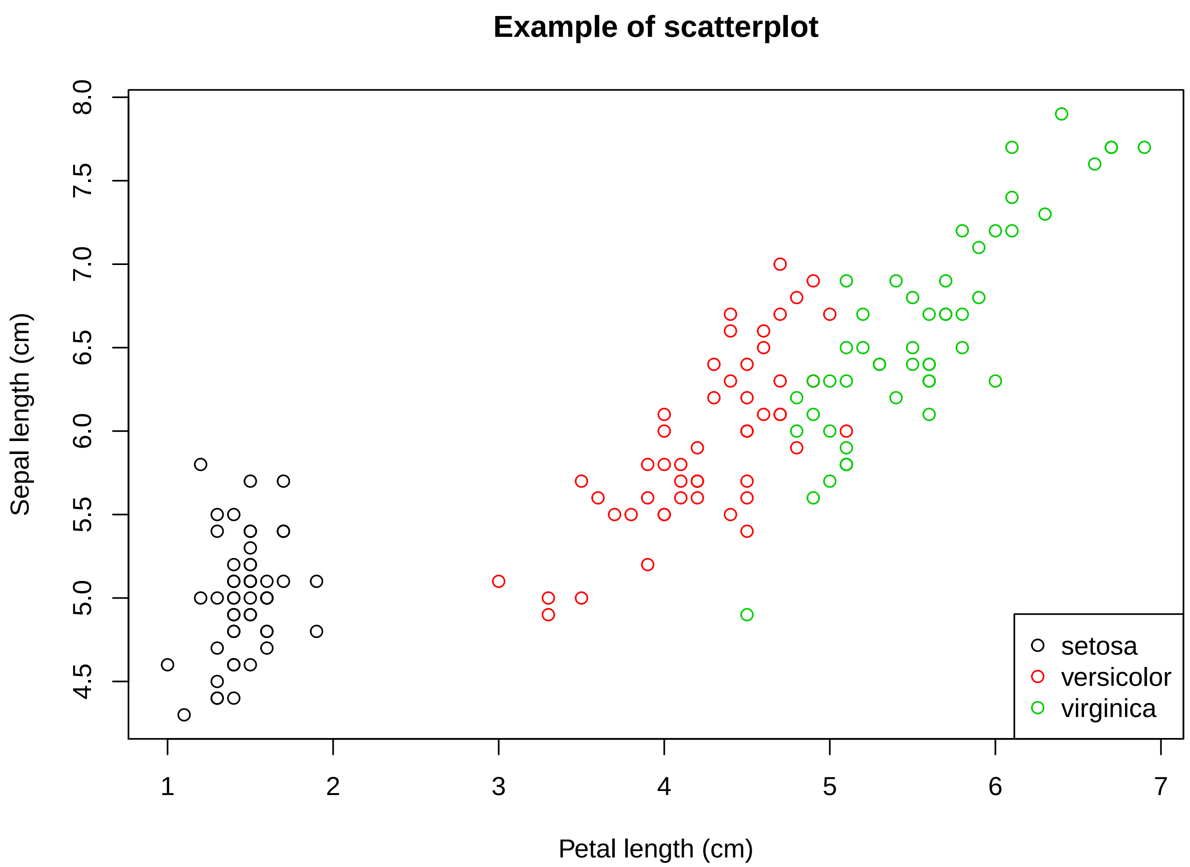


Plot two variables relationships

```
### Dependent variable numeric, independent variable categorical
plot(iris$Petal.Length ~ iris$Species) # Same that boxplot()
```




```
### Two numeric variables, color by species, title and legend
PETAL <- iris$Petal.Length                            # Numeric variable 1
SEPAL <- iris$Sepal.Length                            # Numeric variable 2
SP    <- iris$Species                   # Categorical variable for colors

### Scatterplot
plot(SEPAL~PETAL, col = SP, xlab = "Petal length (cm)",
     ylab = "Sepal length (cm)")

### low-levels
title("Example of scatterplot")               # low-level command title()
legend("bottomright", levels(SP), pch = T,col = c(1:3))      #low-level
```



Random data

Sometimes may be interesting to use random data. This can be done using the command **sample()** [8] to take a sample of the specified size from a vector.

Other useful command to generate random numbers is **runif()** [9]. See its help page for more info.

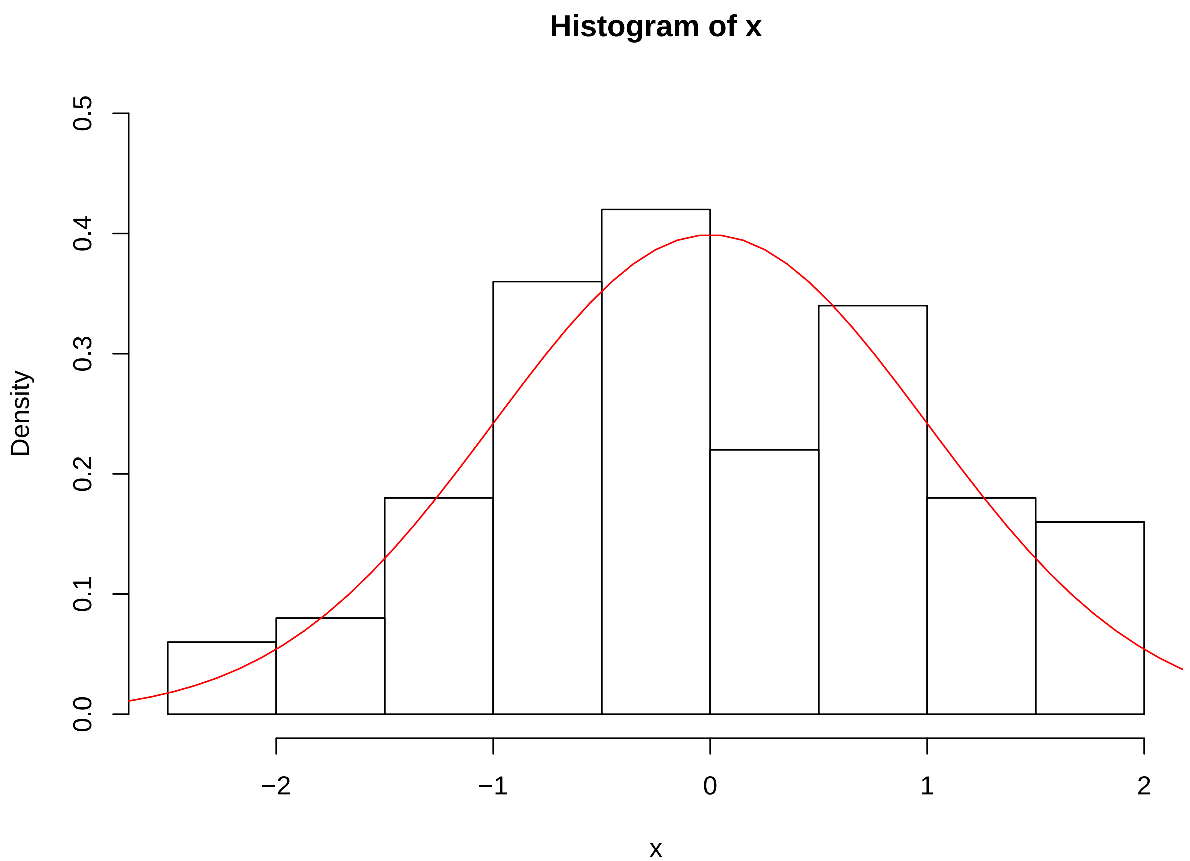
Random data following a particular distribution are very useful for simulations. Usually with distributions from package **stats** [10] there is enough, but if you need more you can look here.

Plotting distributions histograms

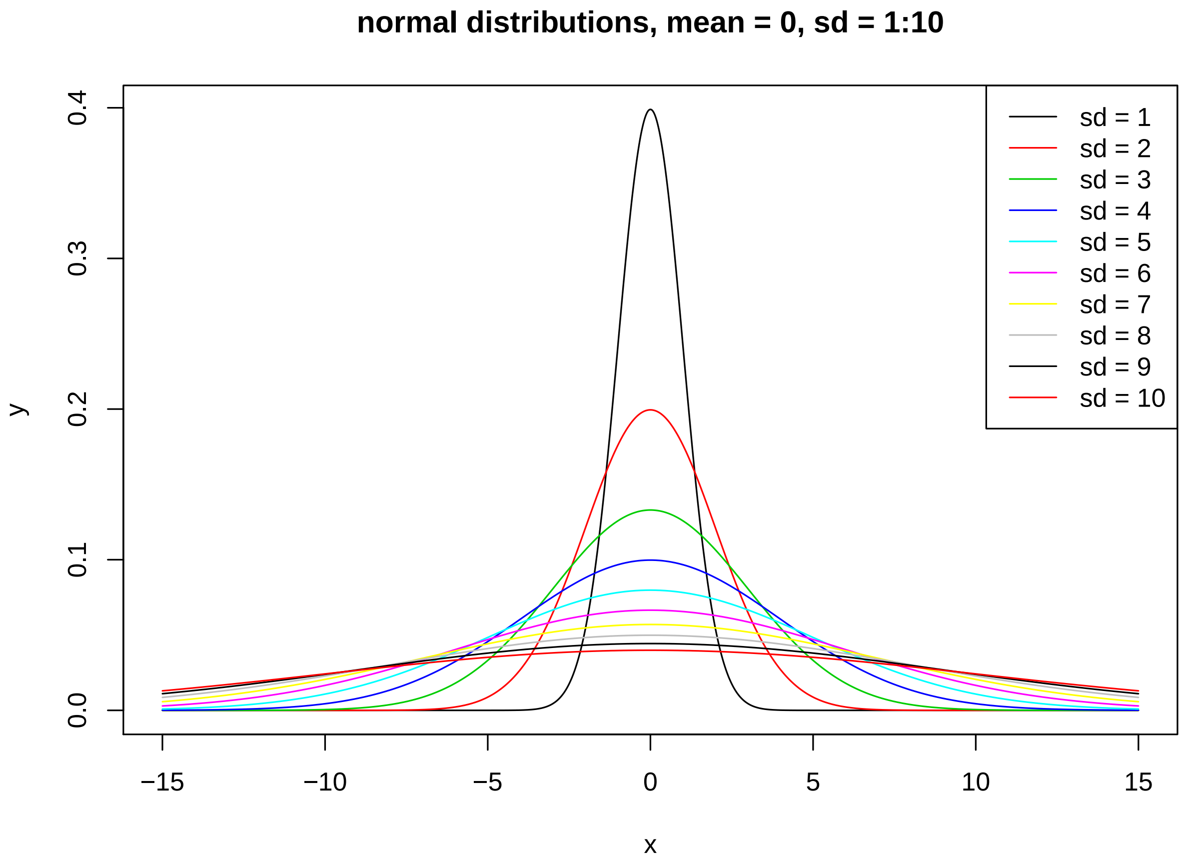
```
#### HISTOGRAMS OF DISTRIBUTIONS #####
?Distributions

### Plot with random normal (0,1) data
x <- rnorm(100, mean = 0, sd = 1)                # Random normal data
hist(x, freq = F, ylim = c(0, 0.5))              # Histogram
```

[8] **?sample** to see the help page
[9] **?runif** to see the help page
[10] see **?distributions**

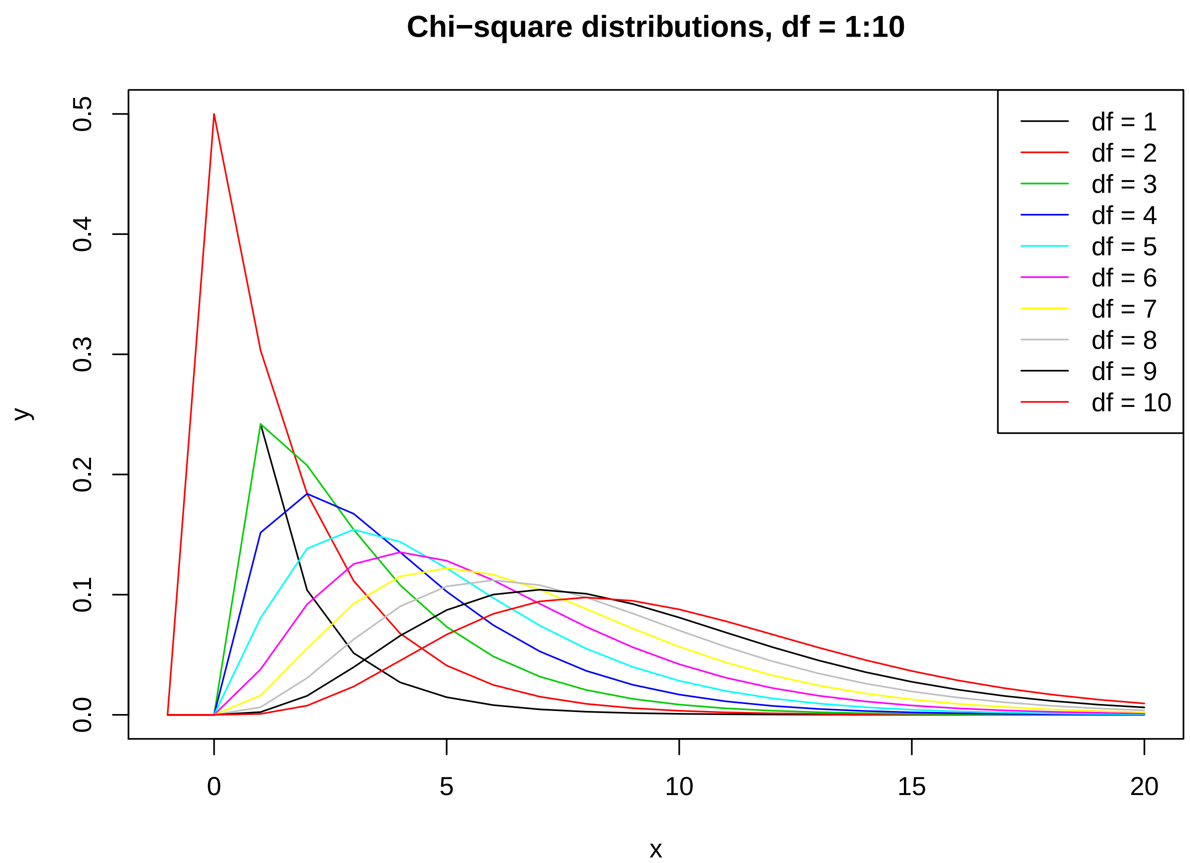
```r
### Plot normal density over the histogram
xl <- seq(-5, 5, length = 100)          # sequence of numbers
y <- dnorm(xl, mean = 0, sd = 1)        # Normal densities for x
lines(xl, y, col = "red")               # Red line with distribution
```



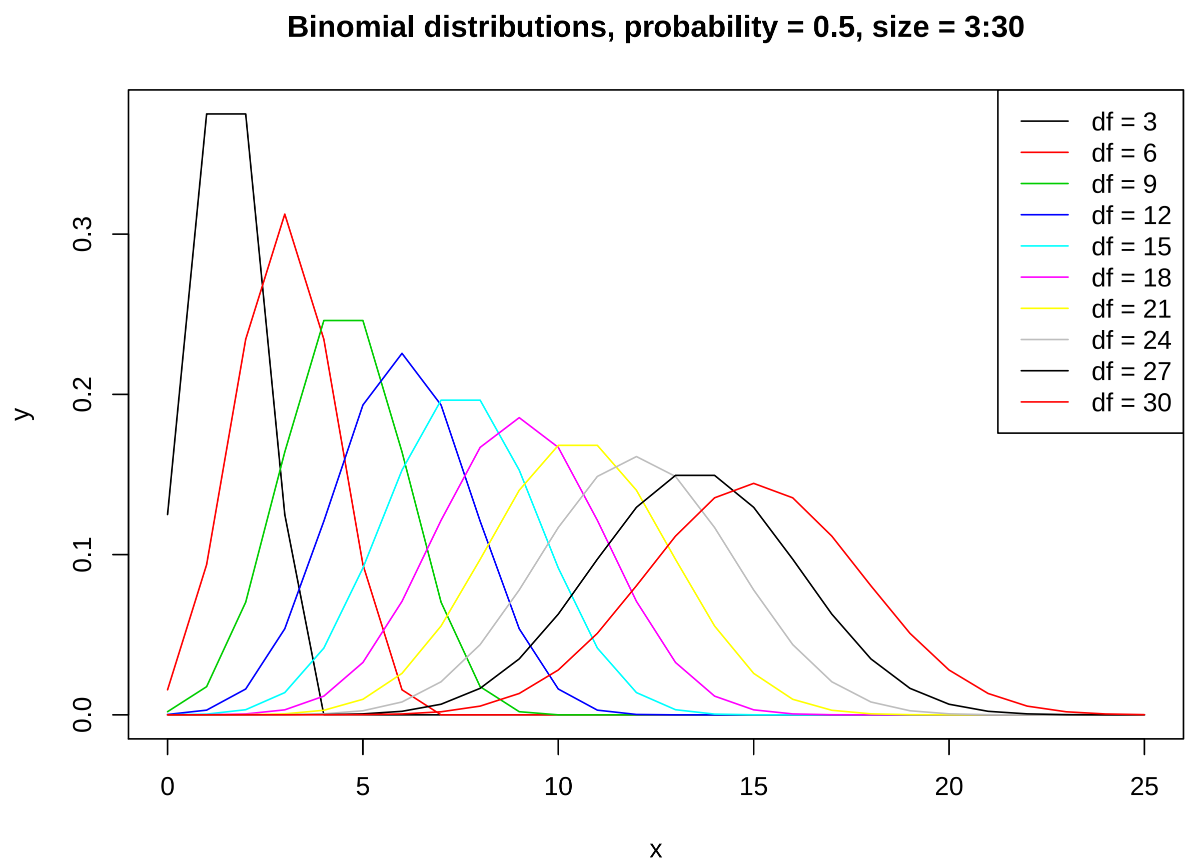

```r
### Plot graphic with ten normal distributions using for()
x   <- seq(-15, 15, length = 1000)      # Sequence of 1000 numbers
y   <- dnorm(x, mean = 0, sd = 1)       # Normal densities for x
plot(x, y, type = "n")                  # Empty plot (type = "n")

### Plot lines over first plot
for (i in 1:10) {                     # Ten lines with sd from 1 to 10
    y <- dnorm(x, 0, i)               # Calculate the normal density
    lines(x, y, col = i)            # Plot the line
}

### Legend
legend("topright", legend = paste("sd =", 1:10)
       , lty = 1, col = 1:10)
title(main = "normal distributions, mean = 0, sd = 1:10")
```




```
##### OTHER DISTRIBUTIONS #####

### Plot graphic with ten CHI-SQUARE distributions using for()
x   <- seq(-1, 20, 1)                    # Sequence
y   <- dchisq(x, df = 2)                 # densities for x
plot(x, y, type = "n")                   # Empty plot (type = "n")

### Plot lines over first plot
for (i in 1:10) {                        # Ten lines with DF from 1 to 10
    y <- dchisq(x, df = i)               # Calculate density
    lines(x, y, col = i)                  # Plot the line
}

### Legend and title
legend("topright", legend = paste("df =", 1:10),lty = 1, col = 1:10)
title(main = "Chi-square distributions, df = 1:10")
```




```r
#### Plot graphic with ten BINOMIALS distributions using for() ######
x   <- seq(0, 25, 1)                        # Sequence
y   <- dbinom(x, prob = 0.5, size = 3)  # densities for x
plot(x, y, type = "n")                      # Empty plot (type = "n")

### Plot lines over first plot
for (i in 1:10) {            # Ten lines with SIZES from 1 to 10
    y <- dbinom(x, prob = 0.5, size = i * 3)     # densities for x
    lines(x, y, col = i)                        # Plot the line
}
### Legend and title
legend("topright", legend = paste("df =", 1:10*3)
       , lty = 1, col = 1:10)
title(main = "Binomial distributions, probability = 0.5, size = 3:30")
```



Export figures

Once we have a nice picture, we may want it outside R, usually in a given format, size and resolution.

Possible output formats are: jpeg, bmp, png, tiff. All of them can be done with the chosen size and resolution. See the help page of **jpeg()**[11] for more information.

Other possible output format is pdf. See the help page for **pdf()**[12] for more information.

The main difference between pdf and the other formats is that in pdf() format we can introduce as many pictures as we want before closing the device, while in others (eg. jpeg) if we start a new plot, it will overwrite the previous one and we will only output the last one before closing the device.

With any of these commands what we are actually doing is opening a device or graphical window different from the default R one. Once we have finished our graph, we have to close this device, with **dev.off()**.

```
#### jpeg plot ####
jpeg(filename = "Plot1.jpeg")           # Open the device "Plot1.jpeg"
plot(iris[1:4], col = iris$Species)    # make the plot into the device
dev.off()                  # Close the device. Do not forget to put the ()
#### See the plot into the working directory. getwd() to see where.
```

It is possible to change the size, background color, resolution, etc.

[11]**jpeg()**: ?jpeg for help
[12]**pdf()**: ?pdf for help

```
#### tiff plot ####
tiff("Plot1.tiff", width = 12, height = 10, units = "cm",
     bg = "transparent", res = 150)       # Open the device "Plot1.tiff"
plot(iris[1:4], col = iris$Species)    # make the plot into the device
dev.off()              # Close the device. Do not forget to put the ()
#### See the plot into the working directory. getwd() to see where.
```

Look for the differences between Plot1.jpeg and Plot1.tiff. Change the parameters and see the results. A very useful background color is bg = "transparent".

Import - export data

The best format for data is always plain text. We can find data in plain text with different extensions: **.txt** for text; **.csv** for comma separated values. In Spain and other countries, the comma is used as decimal separator and therefore is not a good idea to use the comma also to separate values. A better way to separate values into a .csv is by semicolons ; or by tabs.

To see how to import and export data we will first export the data form iris into a file called IrisFile.csv separated by semicolons and with comma as decimal separators, and then import those data from the file.

```
#### Export iris
write.table(iris, file = "IrisFile.csv", dec = ",", sep = ";")
```

See the table into the working directory. Try to open it with a plain-text editor, with excel or with LibreOffice. Be careful, do not save changes without changing the name because some programs (especially Excel) usually change the format.

Now, to import a data file:

```
#### Import IrisFile.csv
MyIris <- read.table("IrisFile.csv", dec = ",", sep = ";")
str(MyIris)                     # Check if it has been correctly imported
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

In both cases, export and import, we can specify the folder where we want (or have) the data table into the file name. By now it is a good idea to keep everything into the working directory.

Import data from web site

It is also possible to import data from a web site. For example if we want to import the original data from this article: https://peerj.com/articles/703/ We can download the data from the link and then import it with **read.table()** or read.csv(), but it would be easier to put directly the URL direction. Note: If the data are heavy and we are going to download them many times, it may be better the first option for downloading the data only once.

```
#### Import csv from web
# Original CSV
Webcsv <- "http://datadryad.org/bitstream/handle/10255/dryad.63704/Feeding_Assays.csv"
Data <- read.table(Webcsv, header = T, sep = ",")
str(Data)                       # Check if it has been correctly imported
```

```
## 'data.frame':    536 obs. of  22 variables:
##  $ Date                      : Factor w/ 33 levels "5/26/12","5/27/12",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ CupID                     : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Herb_Order                : Factor w/ 3 levels "Coleoptera","Hymenoptera",..: 3 3 3 3 3 3 3 3 3 3
##  $ Herbivore_Common          : Factor w/ 20 levels "Ailanthus Webworm",..: 3 3 3 3 19 3 3 6 3 13 ...
##  $ Herbivore_Scientific      : Factor w/ 20 levels "","Arge scapularis",..: 13 13 13 13 7 13 13 9 13
##  $ Food_Type                 : Factor w/ 15 levels "Acer negundo",..: 7 7 7 7 7 7 7 7 7 4 ...
##  $ Trial_Time_Days           : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Herb_PreMass_g            : num  0.172 0.169 0.067 0.069 0.088 0.036 0.039 0.074 0.101 0.34 ...
##  $ Herb_PostMass_g           : num  NA 0.183 0.112 0.082 0.137 0.041 0.065 NA 0.129 0.646 ...
##  $ Plant_PreMass_g           : num  0.34 0.497 0.533 0.354 0.439 0.253 0.549 0.432 0.277 0.999 ...
##  $ Plant_PostMass_g          : num  NA 0.315 0.341 0.204 0.313 0.246 0.469 NA 0.069 0.57 ...
##  $ Correction_Factor         : num  0.0112 0.0154 0.0164 0.0116 0.0139 ...
##  $ Temperature               : int  20 25 30 33 25 20 25 25 30 20 ...
##  $ Frass_Mass_g              : num  NA 0.159 0.058 0.032 0.061 0.002 0.034 NA 0.1 0.266 ...
##  $ Consumption               : num  NA 0.197 0.208 0.162 0.14 0.016 0.097 NA 0.217 0.518 ...
##  $ PctLoss                   : num  NA 0.397 0.391 0.456 0.319 ...
##  $ MassCorr_Consumption_Daily: num  NA 1.17 3.11 2.34 1.59 ...
##  $ Herb_Growth               : num  NA 0.014 0.045 0.013 0.049 0.005 0.026 NA 0.028 0.306 ...
##  $ Herb_MassCorr_Growth      : num  NA 0.083 0.672 0.188 0.557 0.139 0.667 NA 0.277 0.9 ...
##  $ Herb_RGR                  : num  NA 0.08 0.514 0.173 0.443 0.13 0.511 NA 0.245 0.642 ...
##  $ Herb_AssEff               : num  NA 0.195 0.722 0.802 0.564 0.874 0.649 NA 0.54 0.487 ...
##  $ Comments                  : Factor w/ 21 levels "","Ate everything - may need to re-run",..: 5 1
```

Other data formats

There are particular commands and packages in R for working with data in different formats:

"readxl" package, with read_excel(): While read_excel() auto detects the format from the file extension, read_xls() and read_xlsx() can be used to read files without extension.

"writexl" package, with write_xlsx(): Writes a data frame to an xlsx file.

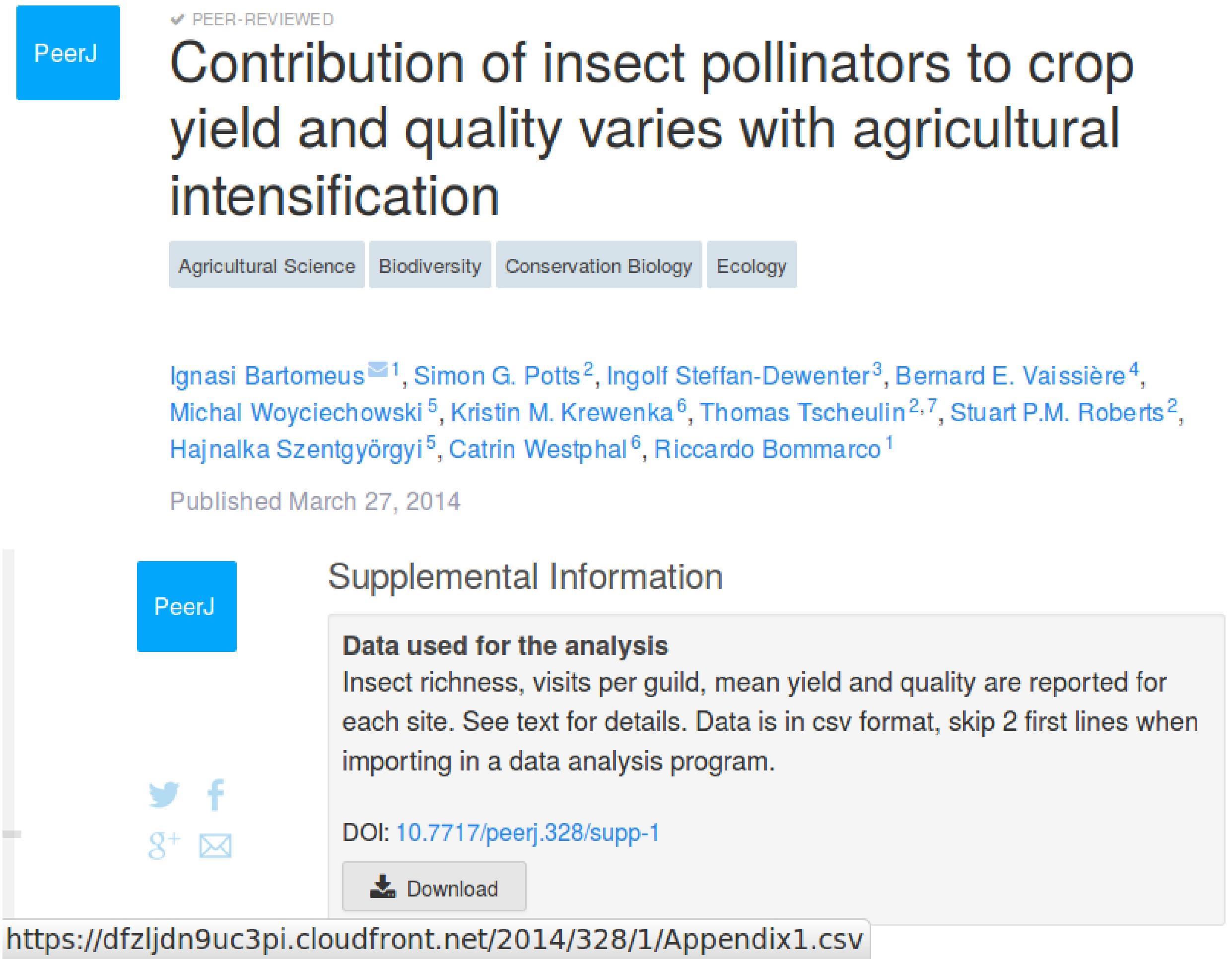
Other packages as **"xlsx"** sometimes are difficult to install because they need to have a good java version installed in the computer.

"shapefiles", "maptools", "rgdal", "spatstat" packages: To read and work with shape-files and maps.

"ape" package: To read DNA sequences.

Exercises

1. Plot a cheat-sheet with values of color and point type (col = , and pch =) from 1 to 25, and export it as a jpeg of 15 cm wide, 6 cm high and resolution 100 points per cm.
2. Plot into a graph ten Poisson distributions with lambda ranging from 1 to 10. Put legend and title. Export it as a .tiff file with size of 15x15 cm.
3. Import data from this article: https://peerj.com/articles/328/


PeerJ

PEER-REVIEWED

Contribution of insect pollinators to crop yield and quality varies with agricultural intensification

Agricultural Science | Biodiversity | Conservation Biology | Ecology

Ignasi Bartomeus[1], Simon G. Potts[2], Ingolf Steffan-Dewenter[3], Bernard E. Vaissière[4], Michal Woyciechowski[5], Kristin M. Krewenka[6], Thomas Tscheulin[2,7], Stuart P.M. Roberts[2], Hajnalka Szentgyörgyi[5], Catrin Westphal[6], Riccardo Bommarco[1]

Published March 27, 2014

PeerJ

Supplemental Information

Data used for the analysis

Insect richness, visits per guild, mean yield and quality are reported for each site. See text for details. Data is in csv format, skip 2 first lines when importing in a data analysis program.

DOI: 10.7717/peerj.328/supp-1

Download

https://dfzljdn9uc3pi.cloudfront.net/2014/328/1/Appendix1.csv

Be careful importing the data. Notice that you have to skip two first lines using "skip = 2"[13].

With these data, using for(), plot graphs to represent the effect of all the numerical variables, from "richness" to "mean_quality" on "yield". Choose the type of graph that you think better represents this effect for the different species. Create only one pdf with all the graphs inside.

[13] **?read.table** for help