Introduction to R

Instructor: Christian Capezza
Course leader: Prof. Biagio Palumbo
Course: Statistics for Technology, a.y. 2019/2020
MSc in Mechanical Engineering for Design and Production

17 October 2019

Contents

Introduction

Why use R?

- Programming language for statistics
- Free and open source
- Packages
- Great support (Stackoverflow, R-Bloggers, etc.)
- Many free resources to learn

Install R and RStudio

Install R before RStudio

- R: https://www.r-project.org/
- RStudio: https://www.rstudio.com/

Rstudio is an integrated development environment (IDE) for R

RStudio

First thing to do: open a new script

- File -> New file -> R script
- CTRL/CMD + SHIFT + N

Four main panels

- Upper left: Script
- Lower left: Console
- Upper right: Environment variables, history
- Lower right: Files, plots, help

Set a new project

When you work with data stored in a file, you want to be able to load the data from the script.

- Create a new folder that will contain your project (scripts, data, plots you will generate, etc.)
- Save the new script into this folder
- Go to Session -> Set Working Directory -> to source file location
  - Alernatively,

```
setwd("C:/New project")
```

The very basics

Start programming: basic operations

You can write code lines in the console, better in the script.

Note: results appear in the console panel

```
1 + 2 # Sum. Note: everything after # is a comment
```

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

```
5 * 4 # Product
```

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

```
7 / 4 # Division
```

```
## [1] 1.75
```

```
3 ^ 2 # Power
```

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

Assign values to variables with <-

```
a <- 1.6
```

Note: the variable `a` appears in the environment panel

Now just type `a` to show its value

```r
a
```

```
## [1] 1.6
```

Access to variable values and use them

```r
a + 3
```

```
## [1] 4.6
```

```r
b <- a * 2
b
```

```
## [1] 3.2
```

```r
b <- 5 # You can overwrite b
b
```

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

```r
b <- b + 1 # You can overwrite b using b itself
b
```

```
## [1] 6
```

Functions

You can use

- functions available in R
- functions provided by packages
- used-defined functions: you define your own function

A word followed by round brackets calls a function. Example: the function `c()` concatenates elements in a single vector:

```r
c(1, 2.5, 4.5)
```

```
## [1] 1.0 2.5 4.5
```

Help

A function is characterized by:

- the name that identifies it (e.g. `c`)
- the required inputs (some are given by default)
- the outputs (in R it is always a single object)

Use help to understand how a specific function works. `help` and `?` are equivalent. Example:

```r
help(c)
?c
```

Data structures

Atomic vectors

Recommended: chapter 'Data structures' from (Wickham 2014).

Four types of atomic vectors:

```
dbl_var <- c(1, 2.5, 4.5) # double
class(dbl_var)
```

```
## [1] "numeric"
```

```
int_var <- c(1L, 6L, 10L) # integer
class(int_var)
```

```
## [1] "integer"
```

```
log_var <- c(TRUE, FALSE, T, F) # logical (T/TRUE equivalent)
class(log_var)
```

```
## [1] "logical"
```

```
chr_var <- c("these are", "some strings") # character
class(chr_var)
```

```
## [1] "character"
```

Factors

Categorical variable. Repeated values are considered as the same level. You can't use values that are not in the levels

```
x <- factor(c("a", "b", "a"))
x
```

```
## [1] a b a
## Levels: a b
```

```
class(x)
```

```
## [1] "factor"
```

```
levels(x)
```

```
## [1] "a" "b"
```

```
table(x) # absolute frequencies
```

```
## x
## a b
## 2 1
```

```
table(x) / length(x) # relative frequencies
```

```
## x
##         a         b
## 0.6666667 0.3333333
```

Numerical vectors: alternative ways to define them

```r
c(6.4, 4.3, 7, 3) # as seen before
```

```
## [1] 6.4 4.3 7.0 3.0
```

```r
1:10 # Sequence of numbers increasing by one
```

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

```r
seq(from = 1, to = 10, by = 3)
```

```
## [1]  1  4  7 10
```

```r
seq(from = 2, to = 10, length = 5)
```

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

```r
rep(3, length = 5)
```

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

Matrices

Different ways to define a matrix

```r
M <- matrix(data = c(11, 12, 13, 14, 15, 16),
            nrow = 2, ncol = 3, byrow = FALSE)
class(M)
```

```
## [1] "matrix"
```

```r
matrix(data = c(11, 12, 13, 14, 15, 16),
       nrow = 2, ncol = 3, byrow = TRUE)
```

```
##      [,1] [,2] [,3]
## [1,]   11   12   13
## [2,]   14   15   16
```

Matrices

```r
rbind(c(11, 13, 15), c(12, 14, 16))    # bind vecs as rows
```

```
##      [,1] [,2] [,3]
## [1,]   11   13   15
## [2,]   12   14   16
```

```r
cbind(c(11, 12), c(13, 14), c(15, 16)) # bind vecs as cols
```

```
##      [,1] [,2] [,3]
## [1,]   11   13   15
## [2,]   12   14   16
```

Heterogeneous data structures: list

List, it is a container and can contain objects of any type and any length. Every object can be identified by a name

```
l <- list(num = 1:10, cha = c("a", "b", "c"),
          fac = factor(c("c1", "c2")))
class(l)
```

```
## [1] "list"
```

```
l
```

```
## $num
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $cha
## [1] "a" "b" "c"
##
## $fac
## [1] c1 c2
## Levels: c1 c2
```

Heterogeneous data structures: data frame

The most important data structure for storing data in R, they combine the behaviour of matrices and lists

- Like matrices, data frames have rows and columns, however
  - matrices require all elements are of the same type
  - data frames allow objects of different type for each column
- Like lists, they contain objects/columns of different types, however
  - lists allow different lengths for each contained object
  - data frames require the same length for each column

Heterogeneous data structures: data frame

```
df <- data.frame(nome = c("Gennaro Esposito"),
                 eta = c(24, 21, 32),
                 sesso = factor(c("M", "F"))) # Does not work!
```

```
df <- data.frame(name = c("Gennaro Esposito", "Maria Rossi"),
                 age = c(24, 21),
                 sex = factor(c("M", "F")))
class(df)
```

```
## [1] "data.frame"
```

```
df
```

```
##               name age sex
## 1 Gennaro Esposito  24   M
## 2      Maria Rossi  21   F
```

Access to elements in the data

Extract elements from vectors

```
a <- c(5, 23, 9)
a[2] # Indicate the position in the vector
```

```
## [1] 23
```

```
a[c(1, 3)] # Extract multiple elements
```

```
## [1] 5 9
```

```
a[c(TRUE, TRUE, FALSE)] # Extract only TRUE
```

```
## [1]  5 23
```

```
a[c(1, 1, 1, 3, 3, 3)] # Extract elements multiple times
```

```
## [1] 5 5 5 9 9 9
```

```
a[] # Take all
```

```
## [1]  5 23  9
```

```
b <- 2
a[b] # You can of course use variables as indexes
```

```
## [1] 23
```

Extract elements from vectors

You can also extract by removing undesired elements with minus sign:

```
a[- 2] # take everything except second
```

```
## [1] 5 9
```

```
a[- c(1, 2)] # take everything except first and second
```

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

Extract elements from matrices

```
M <- rbind(c(11, 13, 15), c(12, 14, 16))
M
```

```
##      [,1] [,2] [,3]
## [1,]   11   13   15
## [2,]   12   14   16
```

```
M[2, 3] # indicate row and column
```

```
## [1] 16
```

```
M[1, ] # take entire row as vector
```

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

```r
M[, 3] # take entire column as vector
```

```
## [1] 15 16
```

```r
M[1, , drop = FALSE] # take entire row as matrix
```

```
##      [,1] [,2] [,3]
## [1,]   11   13   15
```

Extract elements from list

(Results not shown in the slide)

```r
l <- list(num = 1:10, cha = c("a", "b", "c"))
l[1]        # extract object by position as list
l[[1]]      # extract object by position as it is
l["num"]    # extract object by name as list
l$num       # as above (advantageous in RStudio)
l[["num"]] # extract object by name as it is
```

Extract elements from data frame

It works in the same way as matrices (results not shown in the slide)

```r
df <- data.frame(name = c("Gennaro Esposito", "Maria Rossi"),
                 age = c(24, 21),
                 sex = factor(c("M", "F")))

df[2, 3] # indicate row and column
df[1, ]  # take entire row (remains data frame)
df[, 3]  # take entire column as vector
df$sex   # as above (advantageous in RStudio)
df[, 3, drop = FALSE]  # take entire column as data frame
```

Extract elements from data frame

It works in the same way as lists, too (results not shown in the slide)

```r
df[2]        # extract object by position as data frame
df[[2]]      # extract object by position as it is
df["sex"]    # extract object by name as data frame
df[["sex"]] # extract object by name as it is
```

Use logical variables to create conditions…

We can find positions in vectors/matrices/lists that satisfy a condition…

```r
a <- c(3, 2, 1, 4)
a > 2   # TRUE where condition satisfied, FALSE otherwise
```

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

```r
a >= 2 & a <=3  # AND condition
```

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

```r
a == 3 | a == 4 # OR condition
```

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

```r
which(a >= 2)   # Gives positions where condition satisfied
```

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

... and use them to extract elements

... and use them extract the corresponding elements

```r
a <- c(3, 2, 1, 4)
a[a > 2]
```

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

```r
a[a >= 2 & a <=3]
```

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

```r
a[a == 3 | a == 4]
```

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

```r
a[which(a >= 2)] # It is the same as a[a >= 2]
```

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

Missing values

In R, `NA` indicates missing values

```r
a <- c(6.7, NA, 4.2)
is.na(a) # a == NA does not work
```

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

```r
!is.na(a) # ! gives the opposite condition
```

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

```r
which(!is.na(a)) # gives where you have values
```

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

Work with data: commonly used functions and operations

Useful functions for vectors: element-wise operations

The following operations are applied element by element, then they return a vector with the same length of `a`

```r
a <- c(5.3, 6, 1.5, 9)
b <- c(2, 5.1, 4, 1)
a + b
2 * a
a ^ 2 # inverse is a ^ 0.5 or sqrt(a)
exp(a) # inverse is log(a)
sin(a) # analogously use cos tan etc.
```

Useful functions for vectors

The following return a single value (summary statistics)

```r
length(a)
a[length(a)] # gives the last element
sum(a) # sum all elements
mean(a) # mean:  equivalent to sum(a) / length(a)
sd(a) # Standard deviation (obtained dividing by (N-1))
sum(a ^ 2) # sum of squares
min(a) # max(a)
which.min(a) # which.max(a)
crossprod(a, b) # inner product between two vectors
crossprod(a, a) # equivalent to sum of squares
```

Useful functions for logical

When mathematical operations are applied to logical, FALSE are converted to 0 and TRUE to 1. Very useful to calculate proportions:

```r
a <- c(T, F, F, T, T, T, F, T)
sum(a) # total number of trues
sum(!a) # total number of falses
mean(a) # proportion of trues
mean(!a) # proportion of falses
```

Useful functions for matrices

```r
M <- rbind(c(11, 13, 15), c(12, 14, 16), c(5, 7, 2))
N <- rbind(c(14, 10, 17), c(4, 3, 9), c(12, 24, 8))
nrow(M) # number of rows
ncol(M) # number of columns
dim(M)  # two-dim. vector with n. of rows and cols
M[nrow(M), ] # last row
M[, ncol(M)] # last column
M %*% N # Matrix product (check dimensions!)
M * N # Product elementwise (check dimensions!)
det(M) # determinant
solve(M) # inverse matrix (if possible)
```

`apply`: apply function to each row/column

```r
rowMeans(M) # mean of each row
apply(M, 1, function(x) mean(x)) # equivalent
apply(M, 1, mean) # equivalent
colMeans(M) # mean of each colum
apply(M, 2, mean) # equivalent
rowSums(M) # sum of each row
apply(M, 1, sum) # equivalent
colSums(M) # sum of each column
apply(M, 2, sum) # equivalent
rowSd <- apply(M, 1 sd)
```

Useful functions for lists and data frames

```r
l <- list(num = 1:10, cha = c("a", "b", "c"),
          fac = factor(c("c1", "c2")))
names(l) # list element names: useful in loops or to extract
length(l) # number of objects in list
l$new # if you assign something, appends to end of list
summary(l) # summary statistics, useful with data frames
str(l)     # returns the structure, useful with data frames
```

`lapply`: apply function to each element

Works with vectors, lists, and data feames

```r
lapply(l, function(x) length(x)) # returns a list
```

```
## $num
## [1] 10
##
## $cha
## [1] 3
##
## $fac
## [1] 2
```

```r
sapply(l, function(x) length(x)) # returns a vector or matrix, if possible
```

```
## num cha fac
##  10   3   2
```

Loops

For loop

Avoids writing code many times: prevents propagation of errors and changes are simpler. After `in` you put the vector over which you want to iterate (results not shown).

```r
print(1); print(2); print(3)
for (ii in 1:3) print(ii) # the same as above
for (ii in c(1, 5, 2)) print(ii)
for (name in c("hi", "hey")) print(name) # not only numbers!
```

Work with data stored in a file

Packages

```r
install.packages("MASS") # install a new package
library(MASS)            # load package to use its functions/datasets
```

Load data

Once you load the library `ISLR`, the data frame Auto is automatically available in the variable `Auto`

```r
library(ISLR)            # Contains Auto data
data(Auto)               # load the data
summary(Auto)
View(Auto)               # View the data.frame in a new sheet
fix(Auto)                # View the data.frame in a new window
```

Load data from file

The same data are available at the following link http://faculty.marshall.usc.edu/gareth-james/ISL/Auto.csv as indicated in (James et al. 2013). Dowload the csv file and put the file in the working directory (or, better, in a subfolder of the working directory that we call data/). Then, load it to get a data frame

```r
df <- read.csv("data/Auto.csv")
str(df)
```

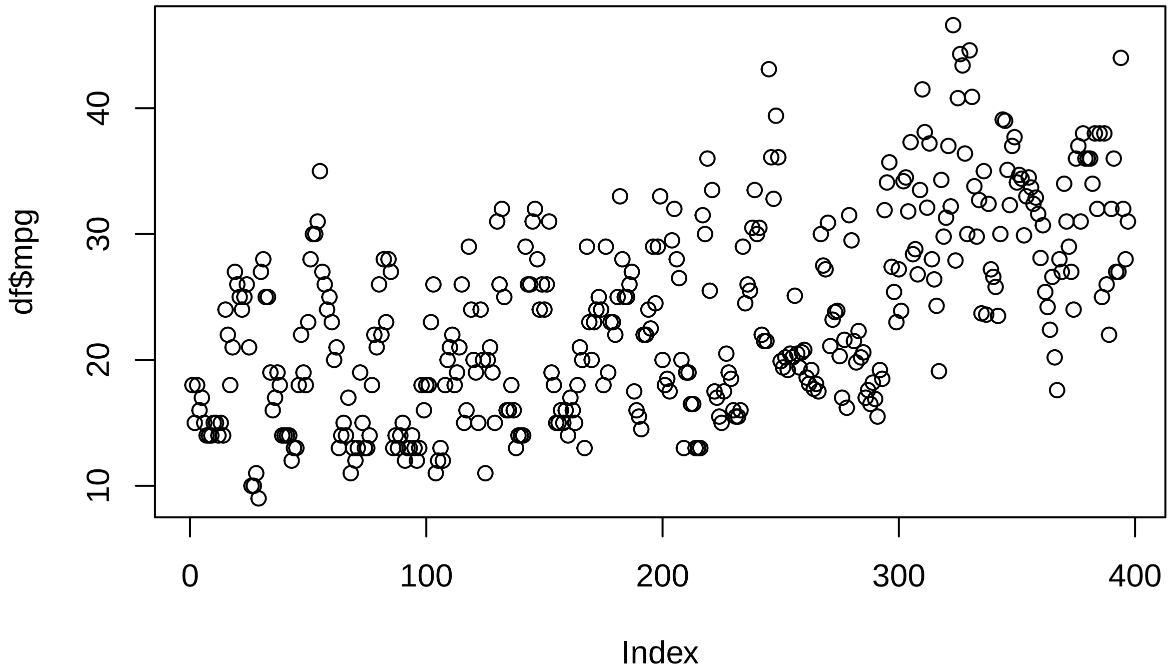
```
## 'data.frame':    397 obs. of  9 variables:
##  $ mpg         : num  18 15 18 16 17 15 14 14 14 15 ...
##  $ cylinders   : int  8 8 8 8 8 8 8 8 8 8 ...
##  $ displacement: num  307 350 318 304 302 429 454 440 455 390 ...
##  $ horsepower  : Factor w/ 94 levels "?","100","102",..: 17 35 29 29 24 42 47 46 48 40 ...
##  $ weight      : int  3504 3693 3436 3433 3449 4341 4354 4312 4425 3850 ...
##  $ acceleration: num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
##  $ year        : int  70 70 70 70 70 70 70 70 70 70 ...
##  $ origin      : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ name        : Factor w/ 304 levels "amc ambassador brougham",..: 49 36 231 14 161 141 54 223 241
```

```r
# View(df) # See the data
```

Plots: the very basics

Graphics: one variable (numeric)

```
plot(df$mpg) # Plots n vector values against 1:n
```



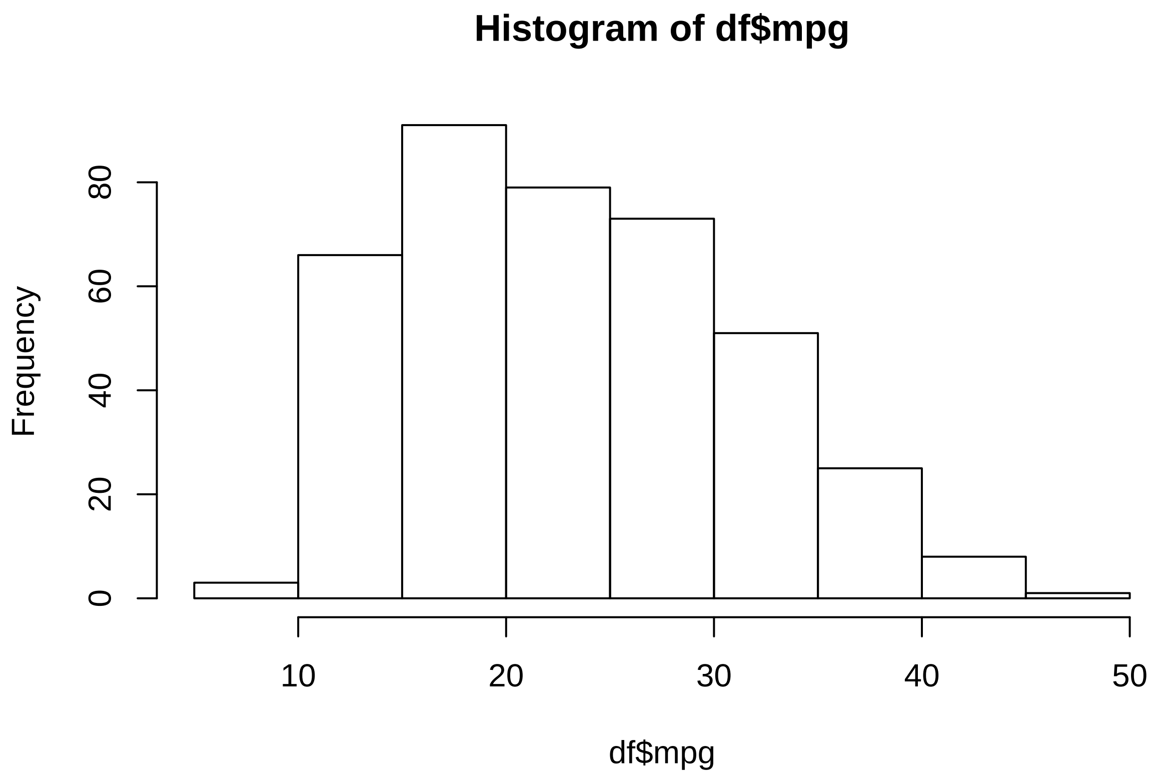
Graphics: one variable (numeric)

Results not shown: try them!

```
plot(df$mpg, type = "l") # Lines instead of points
plot(df$mpg, col = "red") # Set points/lines colours
plot(df$mpg, pch = 16) # Change point shape
plot(df$mpg, size = 3) # Change point/line size
plot(df$mpg, type = "l", lty = 2) # Change line type
plot(df$mpg, main = "Title") # Set plot title
plot(df$mpg, xlab = "x axis") # Set x axis title
plot(df$mpg, ylab = "y axis") # Set y axis title
plot(df$mpg, xlim = c(10, 50)) # Set limits of x axis
plot(df$mpg, ylim = c(0, 60)) # Set limits of y axis
```

Graphics: one variable (numeric)

```
hist(df$mpg) # histogram of frequencies
```



Graphics: one variable (numeric)

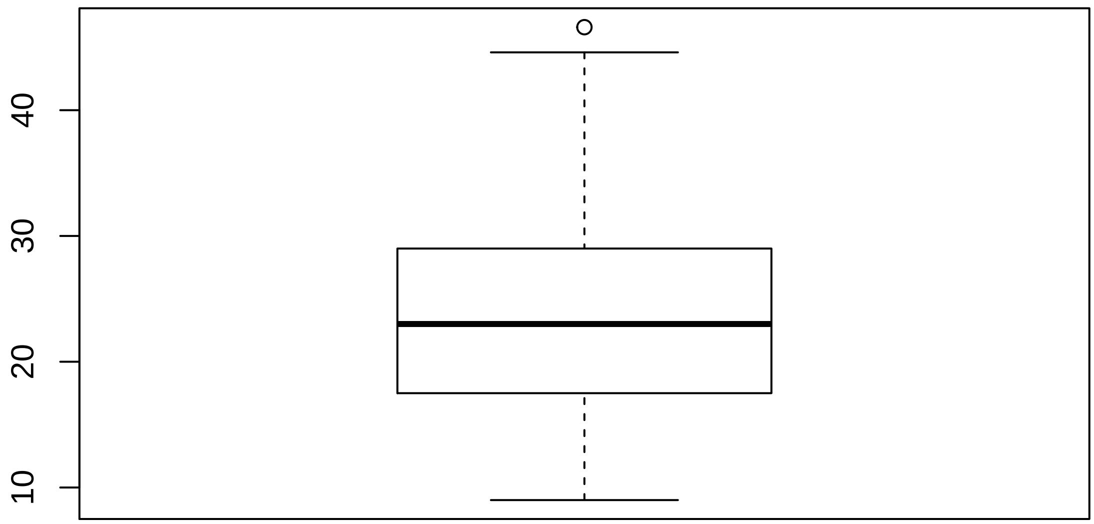
`probability = TRUE` is useful in cases you want to plot densities over histogram. Note: If you want to add a curve or points over an existing plot, after creating the first plot (e.g. `plot`, or `hist`), use `lines` or `points`.

Results not shown: try them!

```
hist(df$mpg, breaks = 40) # set number of bins
hist(df$mpg, probability = TRUE) # histogram of densities
lines(density(df$mpg)) # adds kernel density estimate
```

Graphics: one variable (numeric)

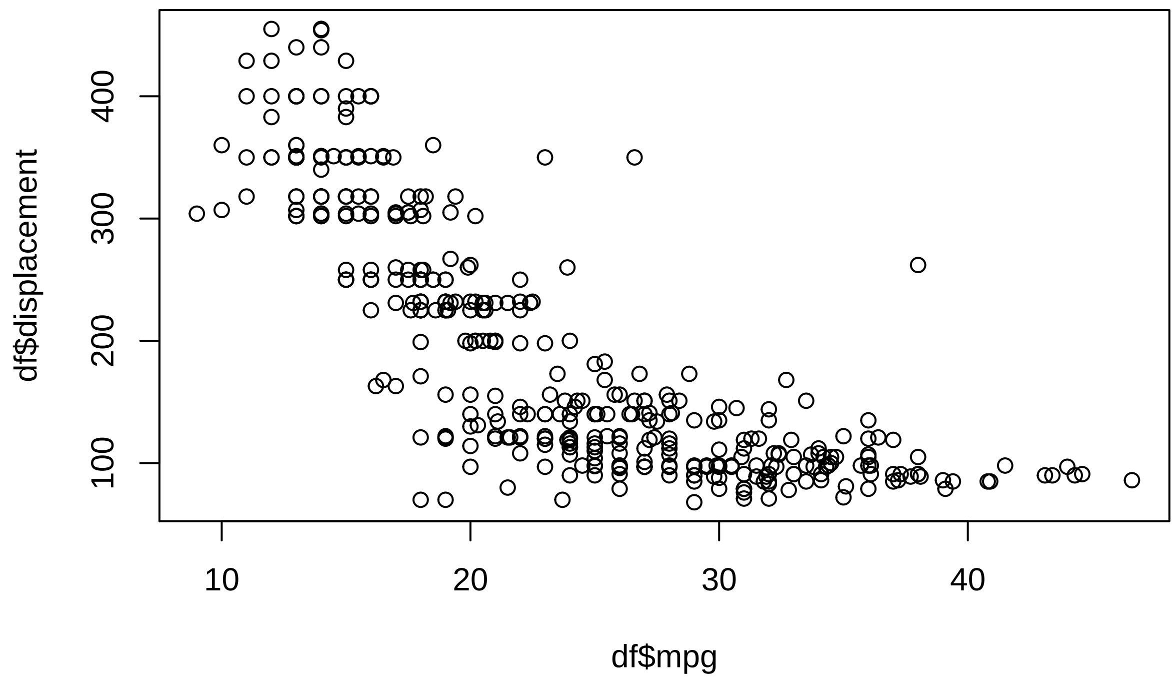
```
boxplot(df$mpg)
```



Graphics: two variables (numeric)

Scatterplot of two variables

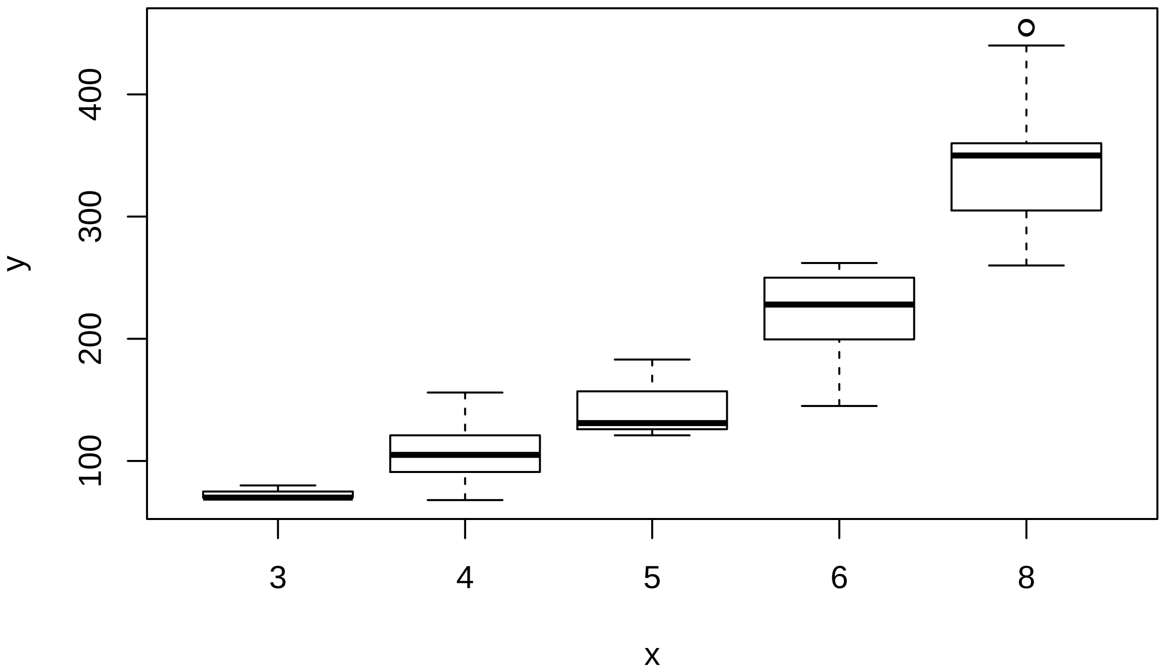
```
plot(df$mpg, df$displacement)
```



Graphics: two variables (one factor one numeric)

Each box plot is the one of all observations of displacement corresponding to the given level of the factor variable.

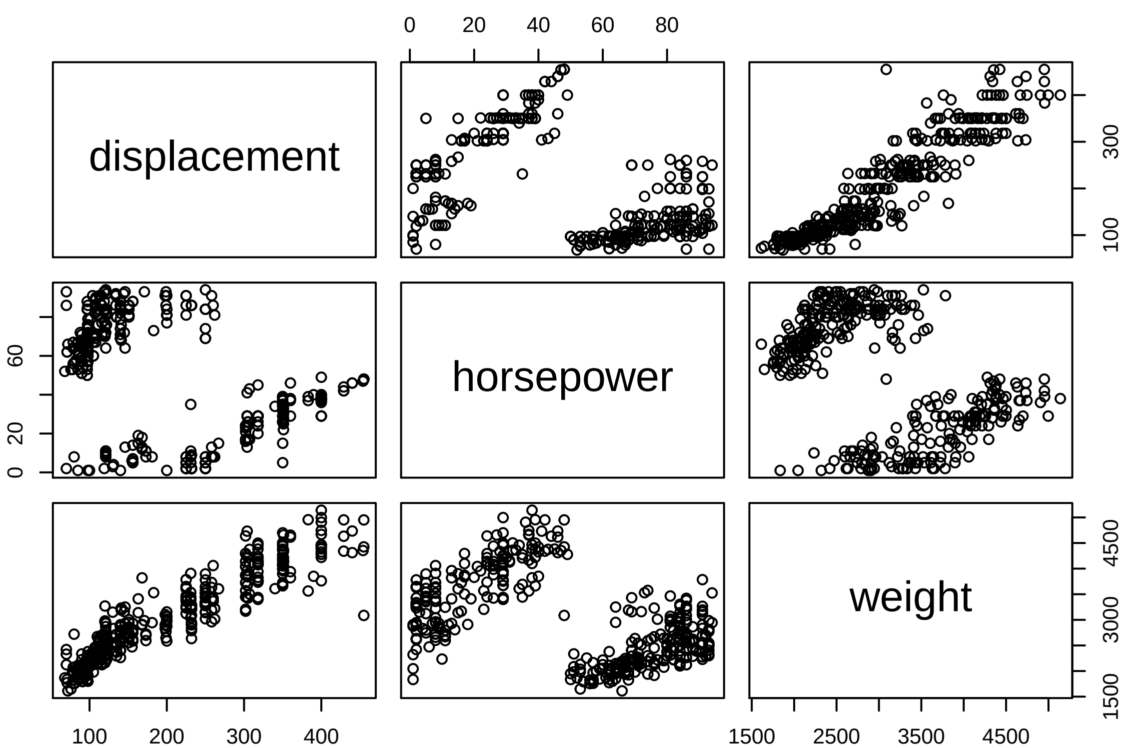
```
plot(factor(df$cylinders), df$displacement)
```



Graphics: more than two variables (numeric)

Scatterplot of three variables

```
pairs(df[, 3:5])
```



Random variables

Random variables: normal distribution

rnorm, dnorm, pnorm, qnorm are the four functions related to the normal distribution. Analogous functions are available for the other distributions (e.g. rexp, dexp, pexp, qexp for the exponential). The four functions, distinguished by the first letter, give

- `dnorm(x)`: give the density $f_X(x)$ of the random variable X in x
- `pnorm(x)`: give the cumulative distribution function (cdf) $F_X(x)$, i.e. the value of the probability $Pr(X \leq x)$
- `qnorm(p)`: give the quantile of the random variable X corresponding to the probability $p \in [0, 1]$, i.e. it returns the value x such that $F_X(x) = p$. In other words, it is the inverse of the cdf: $x = F_X^{-1}(p)$,
- `rnorm(n)`: generate n random observations of X

Random variables: normal distribution

Be careful that:

- For each distribution you must provide their parameters! For example, for the normal, you must provide mean and standard deviation. Check the help!
- Check also in the help the parametrization of the distribution, sometimes random variables are parametrized in more than one way.

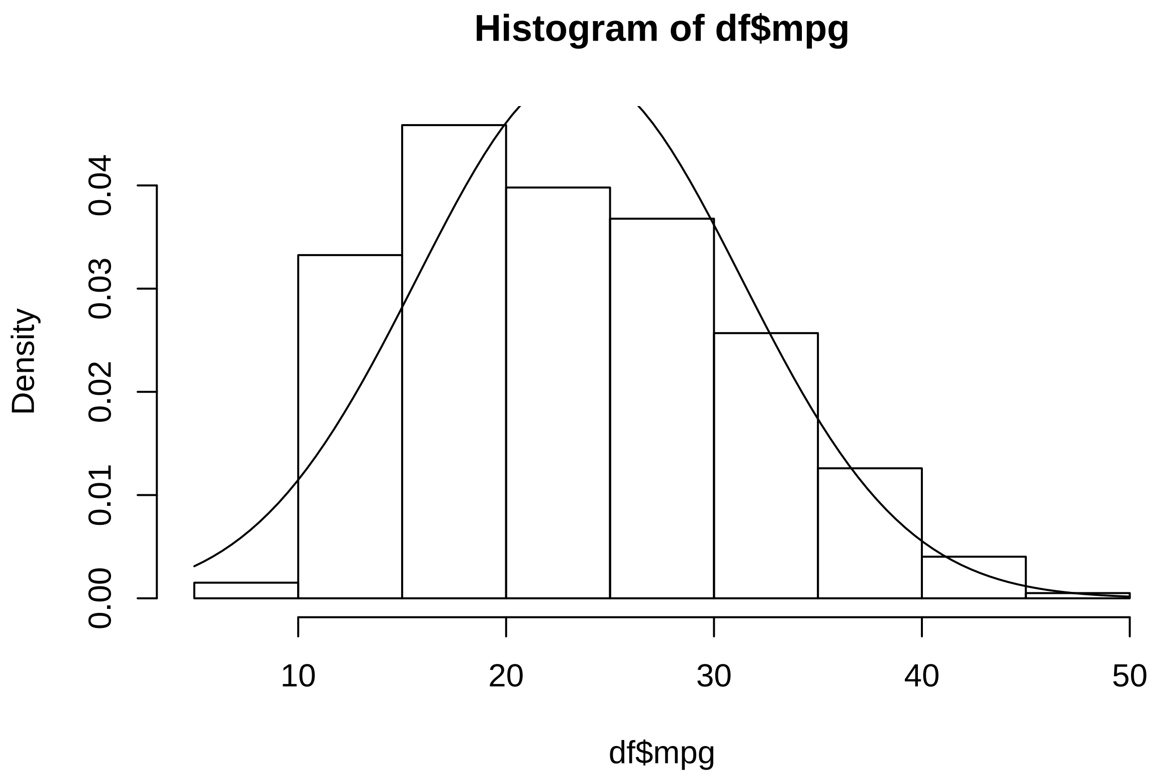
Plot of a function

Often you are interested in plotting a theoretical parametric density over a histogram, to see if data are fit well. More in general let us see how `plot` works

Plot a density over a histogram

You can also plot a parametric density over the histogram. Use `curve` with the argument `add = TRUE`.

```r
hist(df$mpg, probability = TRUE)
curve(dnorm(x, mean(df$mpg), sd(df$mpg)), add = TRUE)
```



How does `curve` work?

`curve` allows to build a plot of a function in a compact way. You can pass only the function name. If you need additional parameter to the function (e.g. `mean` and `sd` with `dnorm`), specify `x` where you pass the independent variable.

Results not shown: try them!

```
curve(dnorm) # Plots with x between 0 and 1
curve(dnorm, from = - 1, to = 1) # set x axis limit
curve(dnorm(x, mean = 0.2, sd = 0.5)) # Set mean and sd
curve(dnorm, add = TRUE) # works like lines/points
```

Some tricks to speed up coding

Keyboard shortcuts

- ALT+SHIFT+K show all keyboard shortcuts
- CTRL(CMD)+SHIFT+N open new script
- CTRL(CMD)+W close script
- ALT+- insert "<-"
- CTRL+1 move cursor to script
- CTRL+2 move cursor to console
- CTRL(CMD)+S save script
- CTRL(CMD)+SHIFT+C comment line/selected lines
- ALT+UP/DOWN move line upwards/downwards
- CTRL(CMD)+F find/replace

Other tricks

- CTRL(CMD)+F with lines selected: if you do replace all, you replace only in the selected lines
- Try to hold ALT, drag the cursor vertically in a script and type/delete with the keyboard! Useful with aligned lines of code!
- Double click on a word/variable to select it, triple click to select whole line, four clicks (OR CTRL+A) to select all
- Use CTRL(ALT)+left/right to move rapidly word by word
- Use auto-completion!!! if you want to load a file in the working directory, type a quote (" or ') and type TAB

References

References

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An Introduction to Statistical Learning*. Vol. 112. Springer.

Wickham, Hadley. 2014. *Advanced R*. Chapman; Hall/CRC.