An introduction to R

Rosario Barone

University of Rome Tor Vergata

Department of Economics and Finance
Centre for Economic and International Studies

(From the teaching material of Prof. Alessio Farcomeni, adapted from https://www.slideshare.net/bcbbslides/an-introduction-to-r)

Introduction to R

R basics

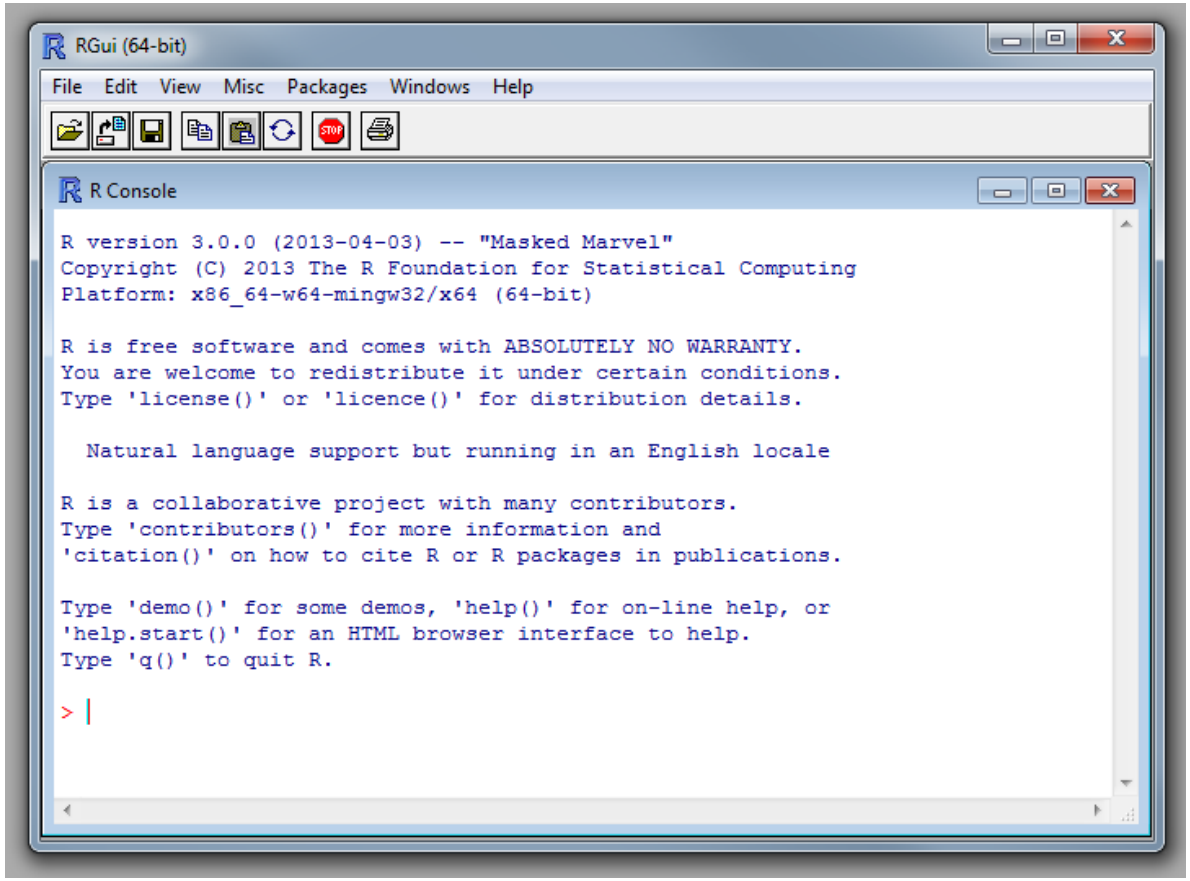
- Arithmetic
- Objects and structures
- Importing and exporting data
- Installing packages
- Getting help

Basic statistics with R

- Descriptive statistics & graphics
- Statistical tests
- Linear regression

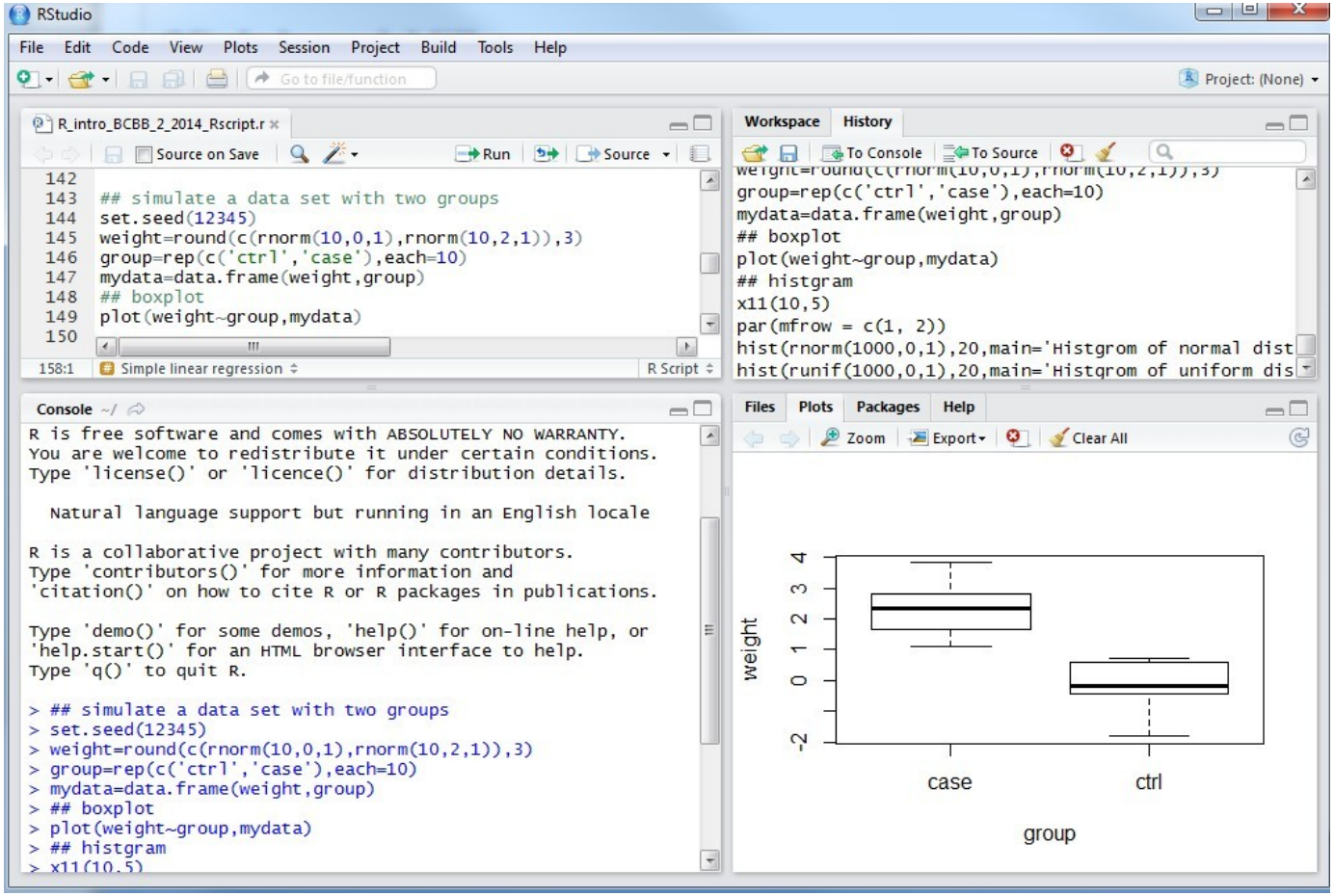
What's R?

- R is a language and environment for statistical computing



R and Rstudio

Rstudio (http://www.rstudio.com/)



Why R

R pros

- Free (as in free beer, and as in free speech)
- Versatile, flexible, powerful, complete
- Based on command-lines: reproducible

R cons

- Based on command-lines: not user friendly
- (Still) slightly cumbersome with big data

What can R do? -- Importing and manipulating data

Importing data.

- Basic text files. (read.table).
- Excel files. (read.xls in the gdata package).
- Several other formats (see library(foreign))

Manipulating data.

- Indexing and subsetting.
- Merging and Reshaping.

	Subject	conc.0.25	conc.0.5
1	1	1.50	0.94
12	2	2.03	1.63
23	3	2.72	1.49
34	4	1.85	1.39
45	5	2.05	1.04
56	6	2.31	1.44

	Subject	time	conc
1.0.25	1	0.25	1.50
2.0.25	2	0.25	2.03
3.0.25	3	0.25	2.72
4.0.25	4	0.25	1.85
5.0.25	5	0.25	2.05
6.0.25	6	0.25	2.31
1.0.5	1	0.50	0.94
2.0.5	2	0.50	1.63
3.0.5	3	0.50	1.49
4.0.5	4	0.50	1.39
5.0.5	5	0.50	1.04
6.0.5	6	0.50	1.44

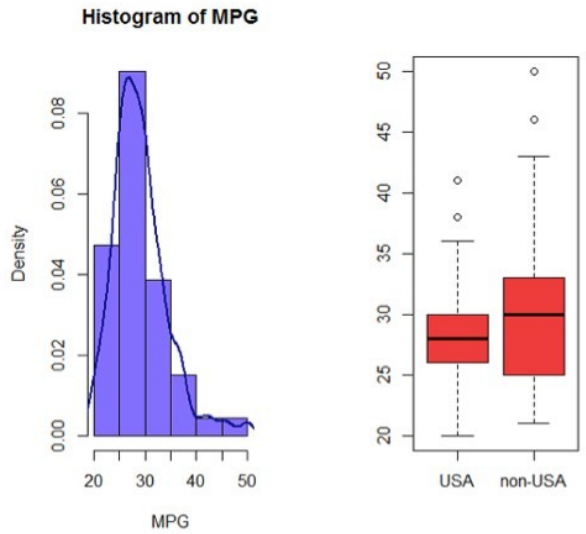
What can R do? -- Data analysis

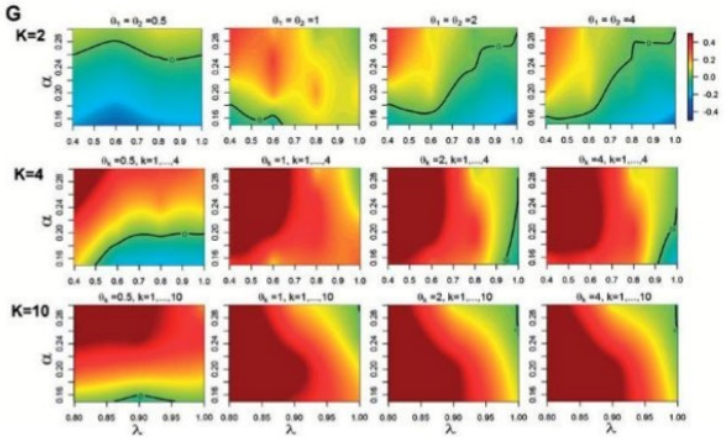
Basic statistical data analysis

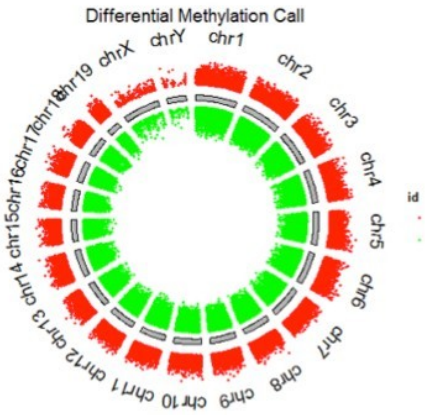
Advanced statistical data analysis

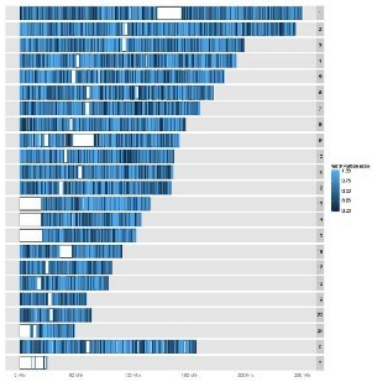
Cutting edge statistical data analysis: researchers often contribute with R extension packages (which can be installed for free using function install.packages or from the menu)

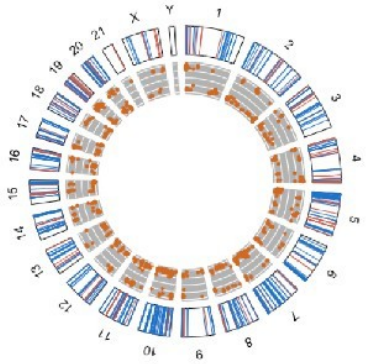
What can R do? -- Visualization











R programming: preliminaries

Install: http://www.r-project.org/

Manual: https://cran.r-project.org/manuals.html

Quit: q();

Getting help

- help('plot')
- ?plot
- Lots on the web (ask Google)

- Comments: # symbol (useful in text files with your commands to reproduce an analysis)

- Extended functionalities in libraries. Some come with the vanilla installation (e.g., library(MASS))

Basic Interaction with R

- Command prompt ‘>’ indicates where to type
- Enter to execute what you wrote
- Esc to break gibberish
- Workspace: temporary memory which can be saved in working directory
- Can source(“code.txt”)
- Can load(“workspace.RData”)
- Note: CaSE sEnSitiVe!

R programming

Variable: information in an object, give it a name (can not start with number or special character)

Variables are case sensitive, so “A” are “a” are two different variables.

Assignment: <- and =

- A<-3; a<-4; X.1←9;

- Everything stays in a workspace which can be saved on the current directory (save.image())

- To list the workspace, ls()
- To erase everything, rm(list=ls())

Arithmetic operators

Addition: +, Subtraction: -
Division: /, Multiplication: *
Exponentiation: ^, exp function

Using R as a calculator:

```
> (3+2)^2*5-7
[1] 118
```

$$\int_0^\infty \frac{1}{(x+1)\sqrt{x}} dx$$

```
> integrand <- function(x) {1/((x+1)*sqrt(x))}
> integrate(integrand, lower = 0, upper = Inf)
3.141593 with absolute error < 2.7e-05
```

Function calls

Syntax: functionName(arg1, arg2)
(args(functionName) or help(functionName))

Arguments can be specified by position or by name, while name overrides position.

```
> log(x=16,base=2)
[1] 4
> log(16,2)
[1] 4
> log(2,16)
[1] 0.25
> log(base=2,x=16)
[1] 4
```

Data type

Three major data types: numeric, character, and logical.

- # numeric variables
- X<-3.1

- # character variables
- X<-"Male"

- # Logical variables
- X<-TRUE; Y<-FALSE

Data structure: vector, matrix and array

Vector: an ordered collection of data.

- x<-c(3.1,4.2,5.6); y<-c("F","M","F”); z<-c(T,F,T,F,T)
- c(…) is a function to concatenate its arguments end to end.

Matrix: two-dimensional generalizations of vectors

Array: multi-dimensional generalizations of vectors

```
> M=as.data.frame(M)
> M<-matrix(1:6,2,3)
> rownames(M)=paste("row",1:2,sep=".")
> colnames(M)=paste("col",1:3,sep=".")
> M
      col.1 col.2 col.3
row.1     1     3     5
row.2     2     4     6
```

Data structure: data frame

Data frames are matrix-like structures.

The columns in data frame can be of different data types.

Generate a new data frame

```
> D<-data.frame(weight=c(1.2,3.4,2.8),gender=c('F','M','F'))
> class(D)
[1] "data.frame"
> D
  weight gender
1    1.2      F
2    3.4      M
3    2.8      F
```

Convert a matrix to a data frame

```
> M<-as.data.frame(M)
> M
      col.1 col.2 col.3
row.1     1     3     5
row.2     2     4     6
```

- **Data Structures in R**

	Linear	Rectangular
All Same Type	VECTORS	MATRIX
Mixed	LIST	DATA FRAME

Importing data

Importing:

- Can import from any format
- Read.table(“file.txt”,header=TRUE,sep=”,”) (txt or csv file might need some manipulation first)
- library(foreign); read.spss(“file.sav”,to.data.frame=TRUE)
- library(gdata); read.xls(“file.xls”)

Indexing and selecting

Indexing: appending an index vector in square brackets to the name of a vector, matrix or data frame.

```
> X<-1:10
> X
 [1]  1  2  3  4  5  6  7  8  9 10
> X[c(1,3,5)]
[1] 1 3 5
```

The index vector can be a numeric vector or a character vector. Can also use $ followed by name of column.

```
> M
      col.1 col.2 col.3
row.1     1     3     5
row.2     2     4     6
> M[,1]
[1] 1 2
> M[2,]
      col.1 col.2 col.3
row.2     2     4     6
> M[2,3]
[1] 6
> M['row.1',]
      col.1 col.2 col.3
row.1     1     3     5
> M[,'col.1']
[1] 1 2
> M['row.1','col.2']
[1] 3
```

Deleting

Deleting: using negative subscripts

```
> X<-1:10
> X
 [1]  1  2  3  4  5  6  7  8  9 10
> X[-2]
[1]  1  3  4  5  6  7  8  9 10
>
> M
      col.1 col.2 col.3
row.1     1     3     5
row.2     2     4     6
> M[-1,]
      col.1 col.2 col.3
row.2     2     4     6
> M[,-2]
      col.1 col.3
row.1     1     5
row.2     2     6
> M[-1,-2]
      col.1 col.3
row.2     2     6
```

Loops

Syntax:

for(variable in sequence) {

statements

}

```
> for (i in 1:10){
+     print(i)
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

Loops

Avoid ‘for’ looping and use “apply” instead if possible.

```
> A<-matrix(rnorm(18),3,6)
> A
           [,1]      [,2]       [,3]       [,4]      [,5]      [,6]
[1,] -1.1481527 1.5523780  1.9058035  0.3911253 0.4454742  1.289636
[2,] -0.3308293 0.9995618 -0.7024981 -1.3493372 0.2288854 -1.146069
[3,]  0.3796588 0.4728620 -2.7539265 -0.9715068 0.4472836 -1.875507
> for( i in 1:ncol(A)){
+       print(mean(A[,i]))
+ }
[1] -0.3664411
[1] 1.008267
[1] -0.5168737
[1] -0.6432396
[1] 0.3738811
[1] -0.5773132
> apply(A,2,mean)
[1] -0.3664411  1.0082673 -0.5168737 -0.6432396  0.3738811 -0.5773132
```

The “apply” family of functions

```
apply(X, MARGIN, FUN, ...)
```

Arguments

`X`	an array, including a matrix.
`MARGIN`	a vector giving the subscripts which the function will be applied over. E.g., for a matrix `1` indicates rows, `2` indicates columns, `c(1, 2)` indicates rows and columns. Where `X` has named dimnames, it can be a character vector selecting dimension names.
`FUN`	the function to be applied: see ‘Details’. In the case of functions like `+`, `%*%`, etc., the function name must be backquoted or quoted.

```
> A
           [,1]       [,2]         [,3]       [,4]       [,5]        [,6]
[1,] -0.1784143 -1.7119119 -0.795967523 -0.7173456 -0.7757516  0.01387142
[2,]  0.7495703  0.3200711 -0.001902476 -0.6474069 -1.0387610 -0.42393931
[3,] -0.4642770  0.9435605 -1.402814741  0.6962816  1.6111892  0.30271782
> apply(A,1,mean)
[1] -0.6942532 -0.1737280  0.2811096
> apply(A,2,mean)
[1]  0.03562635 -0.14942675 -0.73356158 -0.22282362 -0.06777447 -0.03578335
```

Conditional execution

Comparison Operators

equal: == not equal: !
= greater: > less
than: <
greater or equal: >=
less than or equal:<=

Logical Operators

and: & or: | not: !

Conditional execution

Syntax:

if (condition) {
statement
} else
{ alternativ
e
}

```
> x <- rnorm(1)
> x
[1] -1.04468
> if(x > 0){
+     print("Postive number")
+ } else if (x==0) {
+     print("Zero")
+ } else {
+     print("Negative number")
+ }
[1] "Negative number"
```

User-defined functions

Syntax to define a function:

```
myfunction <- function(arg1, arg2, ...) {
    function_body
}
```

Syntax to call a function:

```
myfunction(arg1=..., arg2=...)
```

```
> myvar<-function(x){
+     y<-sum((x-mean(x))^2)/(length(x)-1)
+     return(y)
+ }
> A<-rnorm(6)
> A
[1]  1.85076258  0.27059280  0.22267241  0.43022576 -0.01934847 -1.31496070
> myvar(A)
[1] 1.023429
> var(A)
[1] 1.023429
```

Package installation

Install packages from CRAN network

- install.packages(“Qtools”)
- List functions in a package:
  library(help=packageName)

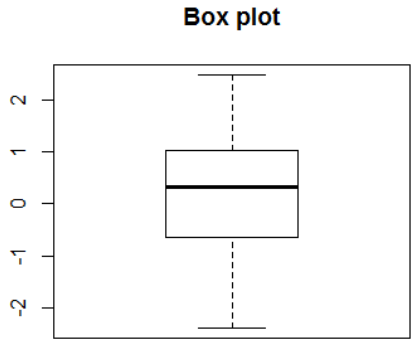
Reading and writing

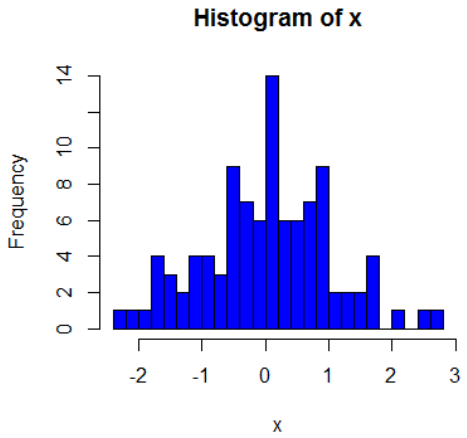
?read.table
?write.table

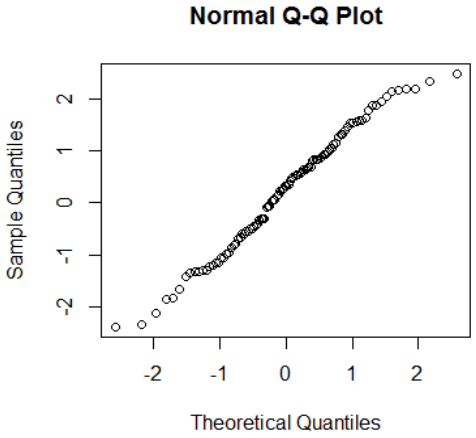
```
> M1<-matrix(rnorm(20),4,5)
> colnames(M1)<-paste('col',1:5,sep='-')
> rownames(M1)<-paste('row',1:4,sep='.')
> M1
            col-1      col-2      col-3       col-4      col-5
row.1  0.67867800 -0.3167286  1.7759049  0.12409765  0.8763414
row.2 -0.57490047  0.5515095 -0.8719302  0.90134290 -0.1009860
row.3  0.04489418 -0.5401848 -0.7451252  0.07305804 -1.9207843
row.4  0.32992309  0.1284177  1.1561778 -0.47675237  1.4029690
> write.table(M1,file='array.txt',sep='\t',
+             col.names = TRUE, row.names = TRUE,quote=F)
> M2<-read.table('array.txt',header=T,sep='\t')
> M2
            col.1      col.2      col.3       col.4      col.5
row.1  0.67867800 -0.3167286  1.7759049  0.12409765  0.8763414
row.2 -0.57490047  0.5515095 -0.8719302  0.90134290 -0.1009860
row.3  0.04489418 -0.5401848 -0.7451252  0.07305804 -1.9207843
row.4  0.32992309  0.1284177  1.1561778 -0.47675237  1.4029690
```

Basic Statistical Analysis

```r
## generate radnom numbers
x<-rnorm(100,0,1)
## summary statistics
summary(x)
## boxplot
boxplot(x,main='Box plot')
## histogram
hist(x,20,col='blue')
## qq plot
qqnorm(x)
```

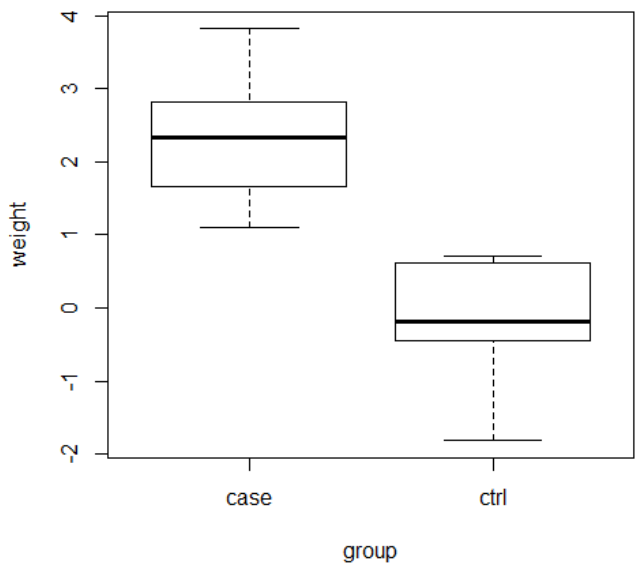






Visualization: Parallel boxplots

```r
## simulate a data set with two groups
set.seed(12345)
weight<-round(c(rnorm(10,0,1),rnorm(10,2,1)),3)
group<-rep(c('ctrl','case'),each=10)
mydata<-data.frame(weight,group)
## boxplot
plot(weight~group,mydata)
```




```
> mydata
   weight group
1   0.586  ctrl
2   0.709  ctrl
3  -0.109  ctrl
4  -0.453  ctrl
5   0.606  ctrl
6  -1.818  ctrl
7   0.630  ctrl
8  -0.276  ctrl
9  -0.284  ctrl
10 -0.919  ctrl
11  1.884  case
12  3.817  case
13  2.371  case
14  2.520  case
15  1.249  case
16  2.817  case
17  1.114  case
18  1.668  case
19  3.121  case
20  2.299  case
```

T-test

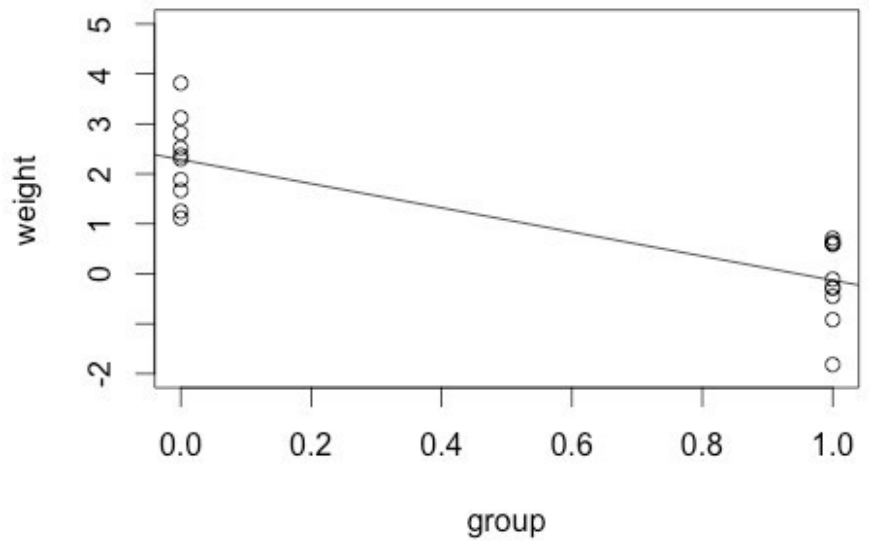
```
> t.test(weight[group == "ctrl"], weight[group == "case"])

        Welch Two Sample t-test

data:  weight[group == "ctrl"] and weight[group == "case"]
t = -6.5335, df = 17.979, p-value = 3.873e-06
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -3.196655 -1.640945
sample estimates:
mean of x mean of y
  -0.1328    2.2860

> t.test(weight~group,mydata)

        Welch Two Sample t-test

data:  weight by group
t = 6.5335, df = 17.979, p-value = 3.873e-06
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 1.640945 3.196655
sample estimates:
mean in group case mean in group ctrl
            2.2860            -0.1328
```

```
> mydata
   weight group
1   0.586  ctrl
2   0.709  ctrl
3  -0.109  ctrl
4  -0.453  ctrl
5   0.606  ctrl
6  -1.818  ctrl
7   0.630  ctrl
8  -0.276  ctrl
9  -0.284  ctrl
10 -0.919  ctrl
11  1.884  case
12  3.817  case
13  2.371  case
14  2.520  case
15  1.249  case
16  2.817  case
17  1.114  case
18  1.668  case
19  3.121  case
20  2.299  case
```

A t-test (and ANOVA) is a linear regression

```
> test<-lm(weight~group,mydata)
> summary(test)
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   2.2860     0.2618   8.733 6.89e-08 ***
groupctrl    -2.4188     0.3702  -6.534 3.85e-06 ***
```



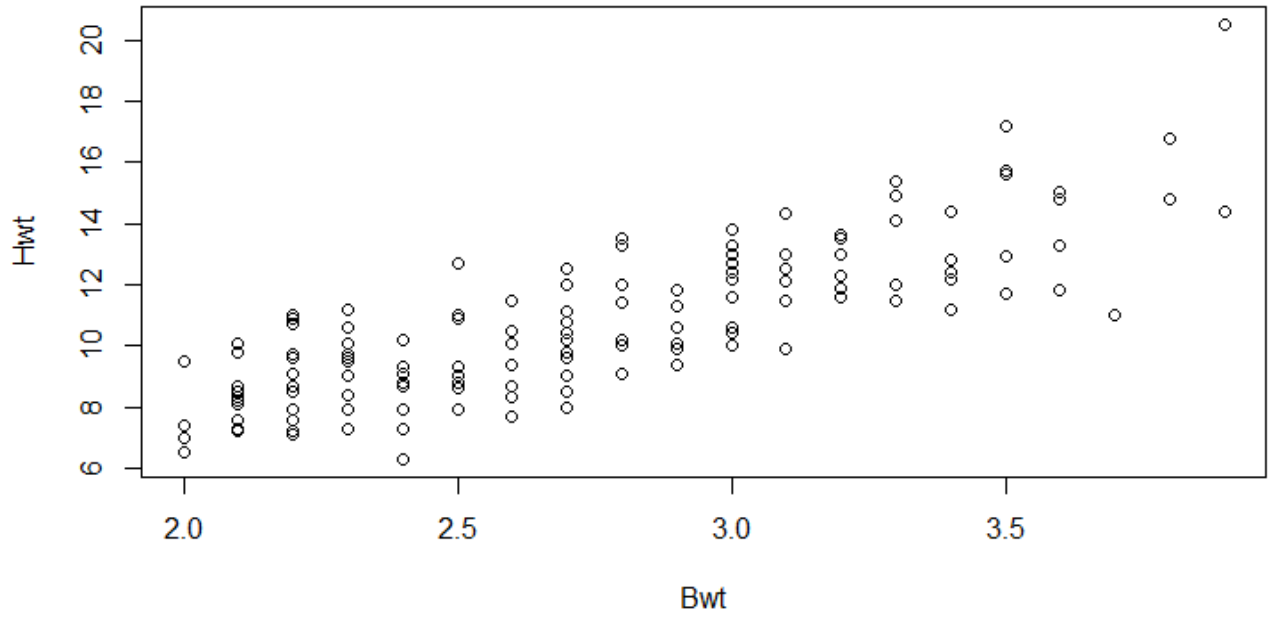
Data summary

```
> head(cats)
  Sex Bwt Hwt
1   F 2.0 7.0
2   F 2.0 7.4
3   F 2.0 9.5
4   F 2.1 7.2
5   F 2.1 7.3
6   F 2.1 7.6
> summary(cats)
 Sex          Bwt             Hwt       
 F:47   Min.   :2.000   Min.   : 6.30  
 M:97   1st Qu.:2.300   1st Qu.: 8.95  
        Median :2.700   Median :10.10  
        Mean   :2.724   Mean   :10.63  
        3rd Qu.:3.025   3rd Qu.:12.12  
        Max.   :3.900   Max.   :20.50  
```

Data visualization

plot(Hwt ~ Bwt, data=cats)

The tilde (~) operator is used to specify that Hwt is described by Bwt



Linear model

Express the relationship as formula: Hwt ~ Bwt

- It reads: Hwt predicted by Bwt, or Hwt as a function of Bwt.
- Variable on the left side: dependent variable. Variable on the right side: independent variable.

Implement in R:

- Mylm <- lm(Hwt ~ Bwt, data=cats)
- summary(mylm)

Linear regression results

```
> mylm <- lm(Hwt ~ Bwt,data=cats)
> summary(mylm)

Call:
lm(formula = Hwt ~ Bwt, data = cats)

Residuals:
    Min      1Q  Median      3Q     Max
-3.5694 -0.9634 -0.0921  1.0426  5.1238

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -0.3567     0.6923  -0.515    0.607
Bwt           4.0341     0.2503  16.119   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.452 on 142 degrees of freedom
Multiple R-squared:  0.6466,    Adjusted R-squared:  0.6441
F-statistic: 259.8 on 1 and 142 DF,  p-value: < 2.2e-16
```

Cheat sheet

- sum(vector), cusum(vector), min(vector), max(vector)
- length(vector), dim(matrix)
- names(dataFrame)
- is.na(vector), na.omit(matrix), levels(factor) (can be assigned, class(factor) must be factor)
- sort(vector), order(vector), seq(,by), rep(,each)
- Distributions: norm, unif, pois, binom, etc.; must use r, d, p, q for random generation, pmf/pdf, cdf, quantile
- quantile(vector)
- Plot, lines, points
- Barplot, hist, boxplot, table, prop.table mean, var, median, cor, cov
- Chisq.test, t.test, cor.test, lm

Final warnings

- You are working with a computer, so there might occur underflow and overflow (try exp(1000))
- Solution: work on log scale
- Note: to compute log(a+b) from log(a) and log(b) without exponentiating, there is function sumlog in library(snipEM)

- Arithmetic is not exact (try .3/3 == 0.1)

- Missing values are treated differently by different functions

- na.omit is your friend (usually), some functions have na.rm=TRUE option