ADVANCED DATA ANALYSIS FOR PSYCHOLOGICAL SCIENCE

Extra slides: Introduction to R

Luca Menghini Ph.D.

luca.menghini@unipd.it

***

Master degree in Developmental and Educational Psychology

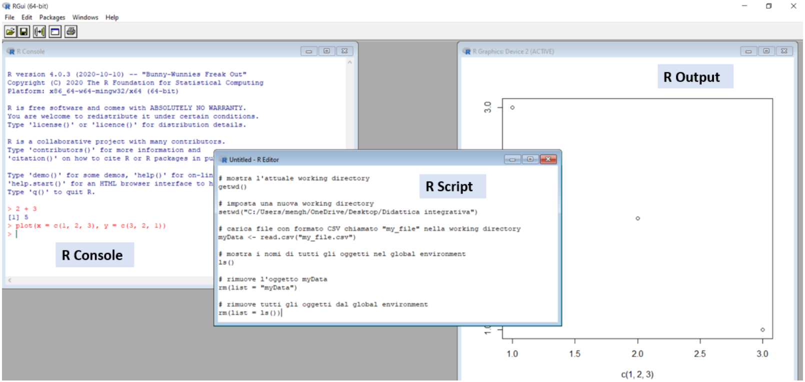
University of Padova

2023-2024

How to download and install R

1. Download `R` from https://www.r-project.org/
   CRAN (*Comprehensive R Archive Network*): network of servers that provide upated software versions and related documentation
2. Select CRAN from the *Download* menu on the left, then select a mirror mirror (e.g., the first or the uniPD's one), then select your OS (Linux, MacOS, or Windows)
3. Install `R` by opening the downloaded `.exe` (Windows) or `.pkg` (MacOS) or by following the commands required by your Linux version

The basic R interfance



```
R version 4.0.3 (2020-10-10) -- "Bunny-Wunnies Freak Out"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in pu

Type 'demo()' for some demos, 'help()' for on-lin
'help.start()' for an HTML browser interface to h
Type 'q()' to quit R.

> 2 + 3
[1] 5
> plot(x = c(1, 2, 3), y = c(3, 2, 1))
> 
```

```
# mostra l'attuale working directory
getwd()

# imposta una nuova working directory
setwd("C:/Users/mengh/OneDrive/Desktop/Didattica integrativa")

# carica file con formato CSV chiamato "my_file" nella working directory
myData <- read.csv("my_file.csv")

# mostra i nomi di tutti gli oggetti nel global environment
ls()

# rimuove l'oggetto myData
rm(list = "myData")

# rimuove tutti gli oggetti dal global environment
rm(list = ls())
```

- **Console**: to rapidly write (>) and execute (`Enter` key) the commands
- **Script** (`File` menu > `New R Script`): to write, modify, and save sequences of commands (saved with the `.R` format)
- **Outputs** (e.g., plot): graphical windows that opens when you run the command

RStudio

`RStudio` is a development environment for `R` using an **optimized graphical interface** to make it more simple (e.g., accessing files, objects, plots, datasets, etc.).

It was founded in 2009 by J. J. Allaire (written in Java and C++) and is managed and developed by an international research group.

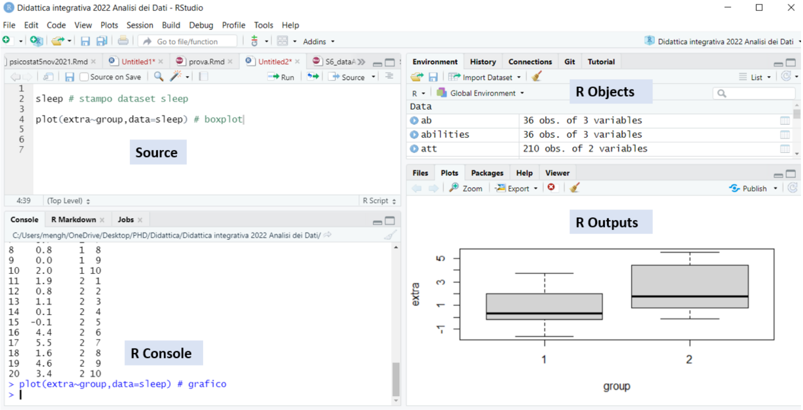
Like `R`, it is **free and open-source** (GNU General Public Licence) + paid premium versions.

How to install `RStudio`

(note: *after* installing `R`)

1. Download `RStudio` from https://rstudio.com
2. Select Download in the top menu, then select the "free" version of **RStudio Desktop**, then select your OS
3. Install RStudio by opening the downloaded file

The RStudio interface



```
sleep # stampo dataset sleep

plot(extra~group,data=sleep) # boxplot
```

Data

ab	36 obs. of 3 variables
abilities	36 obs. of 3 variables
att	210 obs. of 2 variables

```
8    0.8     1  8
9    0.0     1  9
10   2.0     1 10
11   1.9     2  1
12   0.8     2  2
13   1.1     2  3
14   0.1     2  4
15  -0.1     2  5
16   4.4     2  6
17   5.5     2  7
18   1.6     2  8
19   4.6     2  9
20   3.4     2 10
> plot(extra~group,data=sleep) # grafico
> 
```

- **Source**: R Scripts (`.R`), documents and presentations (`.Rmd`), applications (`.app`), etc. To run one or multiple commands, select them and tap on `Ctrl + Enter` or click on the `Run` button on the top-right of the window
- **Environment**: objects included in the workspace and **History** (history of executed commands)

Some elementary commands

Comments (#)

```
# this is a comment
```

Simple mathematical operations

```
2 + 2 # sum
[1] 4
2 * 2 # multiplication
[1] 4
log(3) # natural logarithm
[1] 1.098612
exp(1) # exponential function
[1] 2.718282
```

Longer expressions (with **round brackets**)

```
sqrt(5) * ( (4 - 1/2)^2 - pi/2^(1/3) )
[1] 21.81623
```

Assigning values to **objects** (<-)

```
x <- 3 # creates the 'x' object with value 3
x # pint the value of x
[1] 3
```

Objects' names can include letters, numbers, underscores, and, dots (e.g. tony, tony32, tony.32, tony_32)

```
tony_32 <- x / 3 # assign a value to the object
tony_32 # print the object's value
[1] 1
```

R is key-sensitive!

But it is not sensitive to spaces

```
3+2
[1] 5
3     +  2
[1] 5
```

Hands on R: Arithmetic operations

Perform the following operations using R. (solutions):

Source: Psicostat intro to R

1. $\frac{(45+21)^3+\frac{3}{4}}{\sqrt{32-\frac{12}{17}}}$
2. $\frac{\sqrt{7-\pi}}{3\ (45-34)}$
3. $\sqrt[3]{12-e^2}+\ln(10\pi)$
4. $\frac{\sin(\frac{3}{4}\pi)^2+\cos(\frac{3}{2}\pi)}{\log_7 e^{\frac{3}{2}}}$
5. $\frac{\sum_{n=1}^{10} n}{10}$

Extra: Assign the result of operation #4 to the object `x`, then assign the result of operation #3 to the object `y`, and sum `x` and `y`. The result should be 5.76.

Hands on R: Relational & logical operators

Relational operators

```
3 == 3 # equal to
[1] TRUE
3 != 3 # different from
[1] FALSE
x >= 3 # higher than or equal to
[1] TRUE
5 %in% c(3, 5, 8) # included in
[1] TRUE
```

Logical operators

```
x <- TRUE
y <- !x # negation of
y
[1] FALSE
x & (5 < 2) # joined condition
[1] FALSE
x | (5 < 2) # inclusive disjoint
[1] TRUE
```

Exercises on relational & logical operators:

Source: Psicostat intro to R

1. Define a proposition to test the following condition: "*x is a number included between -4 and -2 or a number included between 2 and 4*"
2. Define two `TRUE` relationships and two `FALSE` relationship that let you test the results of all possible combinations using the logical operators `&` and `|`
3. Run the following operations: `4 ^ 3 %in% c(2,3,4)` and `4 * 3 %in% c(2,3,4)`. What do you note in the execution order of the operators?

Objects & functions

- **Objects**: identifying the values that have been *saved* in the workspace (`Environment`). Values are *assigned* to objects by using the `<-` symbol (minor and minus). To call an object, just write its name.

```
tony_32 <- 2 # assign a value to an object
tony_32 # print object
[1] 2
tony_32 <- tony_32 + 1 # update object
tony_32 # print again
[1] 3
```

- **Functions**: labels associated to sequences of commands that were programmed to return a specific output (called `value`) based on 1+ input values (called `argument`). The function name is always *followed by rounded brakes* that include the arguments. If no arguments are specified, default values are used.

```
sqrt(x = 9) # root square of the value assigned to the argument x
[1] 3
seq(from = 1, to = 5) # numerical sequence between the 'from' and 'to'
[1] 1 2 3 4 5
```

Types (classes) of objects

Logical

```
x <- TRUE
x <- T # writing "TRUE" or "T" is the same
class(x)
[1] "logical"
```

Numeric

```
x <- 1.4
class(x)
[1] "numeric"
```

Integer

```
as.integer(x)
[1] 1
```

Character

```
x <- "I    like R"
x # within "" the space matters!
[1] "I    like R"
```

Vector: series of values with the same class (e.g., all numeric) combined with the function `c()` (*combine*)

```
x <- c(1, 10.5, 3, 2)
x + 1
[1]  2.0 11.5  4.0  3.0
sqrt(x)
[1] 1.000000 3.240370 1.732051 1.414214
y <- c("I","like", "R")
```

Matrix: table with `nrow * ncol`

```
x <- matrix(1:12, nrow = 3, ncol = 4)
rownames(x) <- y # row names
x
     [,1] [,2] [,3] [,4]
I       1    4    7   10
like    2    5    8   11
R       3    6    9   12
```

Classes of objects: vector

An object of class *vector* is a sequence of values with the same class. It can be created with the combine function `c()` or with other functions.

```
x <- c(1, 10.5, 3, 2) # create a numeric vector
y <- 1:10 # create another numeric vector
(z <- rep(c(TRUE, FALSE), each = 2)) # create a logical vector
[1]  TRUE  TRUE FALSE FALSE
as.character(x) # convert the vector class from numeric to character
[1] "1"    "10.5" "3"    "2"
as.numeric(z) # from logical to numeric (FALSE = 0, TRUE = 1)
[1] 1 1 0 0
```

If we use a function on a vector, it applies to all the vectors' values.

```
y*2 # multiply all y values by 2
 [1]  2  4  6  8 10 12 14 16 18 20
round(sqrt(y), 2) # root squared of y values, rounded at 2 digits
 [1] 1.00 1.41 1.73 2.00 2.24 2.45 2.65 2.83 3.00 3.16
```

Classes of objects: vector

Some functions return a single value from a vector of values.

```
length(y) # return the number of vector elements
[1] 10
```

For instance, functions that compute **descriptive statistics**:

```
sum(y) # sum the elements of y
[1] 55
max(y) # maximum value
[1] 10
mean(y) # mean value
[1] 5.5
median(y) # median value
[1] 5.5
var(y) # variance
[1] 9.166667
sd(y) # standard deviation
[1] 3.02765
```

Classes of objects: vector

Squared brakes [ ] allow to **select 1+ elements** of the vector

```
tony32 <- c("one", "two", "three", "four", "five") # vector of characters
tony32[3] # select the third element
[1] "three"
tony32[3:5] # select from the third to the fifth element
[1] "three" "four"  "five"
tony32[c(4, 2)] # fourth and second (not "tony32[4,2]"!)
[1] "four" "two"
```

For instance, we can **select those elements meeting certain conditions** by using logical and relational operators:

```
y[y <= 3 | y > 8] # y values lower/equal to 3 or higher than 8
[1]  1  2  3  9 10
tony32[tony32 != "two"] # all values different from "two"
[1] "one"   "three" "four"  "five"
tony32[substr(tony32, 2, 2) == "w"] # values with the letter "w" in the 2nd position
[1] "two"
```

Classes of objects: vector

The `which()` function returns the **position** of the values meeting the condition

```
substr(tony32, 2, 2) == "w" # test equivalence for each value
[1] FALSE  TRUE FALSE FALSE FALSE
which(substr(tony32, 2, 2) == "w") # position of values meeting the cond.
[1] 2
tony32[substr(tony32,2,2)=="w"] == tony32[which(substr(tony32,2,2)=="w")]
[1] TRUE
```

To **replace** 1+ values of a vector, use the symbol `<-`

```
tony32[1] <- "three"
```

To **remove** 1+ values from a vector, use the symbol `-`

```
tony32[-c(2, 4)]
[1] "three" "three" "five"
```

Classes of objects: factor

An object of class *factor* is a special type of vector used in R to work with *categorical variables* (nominal and ordinal). The possible values of a factor are called *levels*, which are ordered increasingly by default (i.e., numeric or alphabetic order).

```
as.factor(tony32) # from character to factor
[1] three two   three four  five
Levels: five four three two

# summmarize the values of a vector
summary(tony32)
   Length     Class      Mode
        5 character character

# with factors, it shows the freq. by level
summary(as.factor(tony32)) # equivalent to table()
 five  four three   two
    1     1     2     1
```

```
(y <- rep(c(2,4,6),3)) # numeric vector
[1] 2 4 6 2 4 6 2 4 6

as.factor(y) # from numeric to factor
[1] 2 4 6 2 4 6 2 4 6
Levels: 2 4 6

summary(y) # summary of numeric vector
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      2       2       4       4       6       6

summary(as.factor(y)) # summary of factor
2 4 6
3 3 3
```

Classes of objects: factor

Create a factor with the function factor():

```
factor(x = c("C",rep("A",3),c("B","A","C")))
[1] C A A A B A C
Levels: A B C
(x <- factor(x = c("C","A","B","A"), # vector
             levels = c("C","A","B"))) # levels
[1] C A B A
Levels: C A B

levels(x) # levels() print the levels
[1] "C" "A" "B"
factor(x, levels=c("B","A","C")) # change order
[1] C A B A
Levels: B A C
levels(x) <- c("Uno","Due","Tre") # change names
x
[1] Uno Due Tre Due
Levels: Uno Due Tre
```

Factor levels can be **ordered** by setting the argument ordered = TRUE, which returns an ordrdinal variable based on the order defined by the levels argument.

```
# unordered factor (default)
vec <- c("Maria","Mauro","Teresa","Carlo")
factor(x = vec)
[1] Maria  Mauro  Teresa Carlo
Levels: Carlo Maria Mauro Teresa

# ordered factor (default alphabetical order)
factor(x = vec, ordered = TRUE)
[1] Maria  Mauro  Teresa Carlo
Levels: Carlo < Maria < Mauro < Teresa
# ordered factor (setting a different order)
factor(x = vec, ordered = TRUE,
       levels=c("Teresa","Mauro","Maria","Carlo"))
[1] Maria  Mauro  Teresa Carlo
Levels: Teresa < Mauro < Maria < Carlo
```

Classes of objects: matrix

A *matrix* is a **bidimensional structure** (`nrow * ncol`) **of values with the same class**, which can be created with the function:

`matrix(data, nrow = , ncol = , byrow = FALSE)`

```
(x <- matrix(1:12, nrow = 3, ncol = 4))

     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

matrix(1:12, nrow = 3, ncol = 4, byrow = TRUE)

     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12

matrix(c("Mar","Mau","Ter","Car"), nrow = 2)

     [,1]  [,2]
[1,] "Mar" "Ter"
[2,] "Mau" "Car"
```

To **select 1+ values** from a matrix, we use again **squared brackets**, but this time we use the syntax `matrix_name[row_number, col_number]`

```
x[1,2] # 1st row, 2nd column
[1] 4
x[2,1] # 2nd row, 1st column
[1] 2
x[1:3,2] # rows 1-3, second column
[1] 4 5 6
x[1,] # 1st row, all columns
[1]  1  4  7 10
x[,2] # 2nd column, all rows
[1] 4 5 6
```

Classes of objects: matrix

```
x # print the matrix x
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

Join 1+ matrices with cbind() and rbind()

```
cbind(x,matrix(rep(3,6),nrow=3)) # by column
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    4    7   10    3    3
[2,]    2    5    8   11    3    3
[3,]    3    6    9   12    3    3
rbind(x,matrix(rep(3,4),ncol=4)) # by row
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
[4,]    3    3    3    3
```

Setting row and column names:

```
rownames(x) <- c("a","b","c") # row names
colnames(x) <- 1:4 # column names
x
  1 2 3  4
a 1 4 7 10
b 2 5 8 11
c 3 6 9 12
c(nrow(x),ncol(x)) # No. of rows and cols = dim(x)
[1] 3 4
t(x) # transpose matrix (inverting rows and cols)
   a  b  c
1  1  2  3
2  4  5  6
3  7  8  9
4 10 11 12
```

Data structures: data.frame

A *dataframe* is a **bidimensional structure of vectors that can have different classes** (e.g., numeric, character, and factor) that can be created with the function: `data.frame(name_var1 = c(...), name_var2 = c(...), ...)`

```
(x <- data.frame(Num = 1:4,
                 Char = c("a","b","c","d"),
                 Logi = rep(c(TRUE,FALSE),2)))
  Num Char  Logi
1   1    a  TRUE
2   2    b FALSE
3   3    c  TRUE
4   4    d FALSE
str(x) # structure of the dataframe
'data.frame':   4 obs. of  3 variables:
 $ Num : int  1 2 3 4
 $ Char: chr  "a" "b" "c" "d"
 $ Logi: logi  TRUE FALSE TRUE FALSE
```

While `str(df_name)` returns the structure of a dataframe, `summary(df_name)` returns a summary for each column.

```
summary(x)
      Num           Char              Logi
 Min.   :1.00   Length:4           Mode :logical
 1st Qu.:1.75   Class :character   FALSE:2
 Median :2.50   Mode  :character   TRUE :2
 Mean   :2.50
 3rd Qu.:3.25
 Max.   :4.00
```

Data structures: data.frame

Manipulating a dataframe is very similar to what we saw for matrices.

Selecting values and joining two dataframes:

```
x[2, 2:3] # 2nd row, 2nd and 3rd column
  Char  Logi
2    b FALSE
cbind(x,data.frame(new=4:1)) # join by col
  Num Char  Logi new
1   1    a  TRUE   4
2   2    b FALSE   3
3   3    c  TRUE   2
4   4    d FALSE   1
rbind(x[1:3,],data.frame(Num=10, # join by row
                         Char="z",Logi=FALSE))
  Num Char  Logi
1   1    a  TRUE
2   2    b FALSE
3   3    c  TRUE
4  10    z FALSE
```

Name and number of rows and columns:

```
rownames(x) # default = 1:nrow(x)
[1] "1" "2" "3" "4"
colnames(x)[2] # 2nd column name
[1] "Char"
nrow(x) # no. of rows
[1] 4
ncol(x) # no. of cols
[1] 3
```

Transpose of a dataframe:

```
t(x)
     [,1]   [,2]    [,3]   [,4]
Num  "1"    "2"     "3"    "4"
Char "a"    "b"     "c"    "d"
Logi "TRUE" "FALSE" "TRUE" "FALSE"
```

Data structures: dataframe

To select a column (vector) from a dataframe, we can use the $ symbol with the syntax df_name$column_name:

```
x$Char # selecting the Char column
[1] "a" "b" "c" "d"
x$Char[2] # second value from the Char column
[1] "b"
x$Char[2] == x[2,2] # equivalent commands
[1] TRUE
x$Char <- NULL # removing the Char column
x[x$Num < 3,] # selecting cases with Num < 3
  Num  Logi
1   1  TRUE
2   2 FALSE
# same result with subset(x, Num < 3)
```

Alternative way to select the columns:

```
x[,"Logi"] # write col names within ""
[1]  TRUE FALSE  TRUE FALSE
x[1:2,c("Num","Logi")]
  Num  Logi
1   1  TRUE
2   2 FALSE
```

“Head” and “tail” of a dataframe:

```
head(x, n = 2) # first 2 rows
  Num  Logi
1   1  TRUE
2   2 FALSE
tail(x, 1) # last row
  Num  Logi
4   4 FALSE
```

Data structures: list

A *list* is a **collection of objects** that can have **different classes** (es. vector, matrix, and data.frame) e **different length** (contrarily to matrices and dataframes). It is the most complex and versatile structure in R, which can be created with the function `list(nome_oggetto1 = ..., nome_oggetto2 = ..., ...)`

```
x <- list(Num = 1:4,
          Matr = matrix(1:12, nrow=3),
          df = x,
          lst = list(1:3,2:3))
str(x) # structure of the list
List of 4
 $ Num : int [1:4] 1 2 3 4
 $ Matr: int [1:3, 1:4] 1 2 3 4 5 6 7 8 9 10 ...
 $ df  :'data.frame':   4 obs. of  2 variables:
  ..$ Num : int [1:4] 1 2 3 4
  ..$ Logi: logi [1:4] TRUE FALSE TRUE FALSE
 $ lst :List of 2
  ..$ : int [1:3] 1 2 3
  ..$ : int [1:2] 2 3
```

To **select 1+ objects** from a list, we still use **squared brakets**

```
x[1] # single = create a sub-list
$Num
[1] 1 2 3 4
class(x[1])
[1] "list"
x[[1]] # double = extract the object
[1] 1 2 3 4
class(x[[1]])
[1] "integer"
x[[3]][2,1]
[1] 2
```

Hands on R

There are 5 students (Gina, Pina, Tina, Maria, and Lina), each with a registration number (507, 535, 566, 955, 515) and a score at the exam (19, 27, 28, 25, 25).

Source: Pastore, M. (2021). Master's degree course on "Data analysis in community psychology".

1. Create 3 *vectors* including, respectively, students' name, serial numbers, and exam score
2. Join the 3 vectors to create a *dataframe*
3. Delete the 3 original *vectors* from the workspace
   (only the dataframe should remain)
4. Select and print Maria's grade
5. Select the row including all the values belonging to Pina
6. Select all students with exam score equal to 25 (you can also use the `subset()` function - see `?subset`)
7. Select all students with an exam score higher than 20 () (you can also use the `subset()`)
8. Compute the mean and standard deviation of the exam scores

Functions & pakages

Many things in R can be done by using **functions**, which always include the following elements: *function name*, *arguments* (`arg_name = arg_value` or without name, based on the default position), and returned *value*.

```
sqrt(x = c(1,2,3))

[1] 1.000000 1.414214 1.732051

sqrt(c(1,2,3))

[1] 1.000000 1.414214 1.732051
```

R Help system: To know the details of any function (arguments, value, etc.), just add the `?` symbol before the function name

```
?sqrt
```

R packages: To get additional functions than those included in the base R packages, you need to install and open the related package

```
install.packages("pkg_name") # installing a package

library(pkg_name) # open a package

pkg_name::function_name() # using a function without loading the pkg
```

Objects, functions, and workspace

When we assign a value to an object, the latter is recorded in the **workspace**: the place that includes all objects and functions defined by the user (*Environment* tab in Rstudio).

The `ls()` function returns the name of any object and function included in the workspace, whereas the `rm()` function removes the object(s) selected:

```
x <- 1 # assign value to x
y <- 2 # assign value to y
ls() # show all objects in the workspace
[1] "x" "y"
rm(y) # remove object y
ls()
[1] "x"
```

Combining the two functions, the command **`rm(list=ls())` empties the workspace**, removing all objects and functions (very **useful at the beginning of any script!**)

```
rm(list = ls()) # remove all objects & functions
```

Objects & functions from inside R: Default packages

Some packages are **installed in R by default**, they do not require to be opened, and they are not shown in the workspace.

```
rm(list=ls()) # remove all objects & functions
head(sleep,4) # 'sleep' dataset from the 'datasets' pkg
  extra group ID
1   0.7     1  1
2  -1.6     1  2
3  -0.2     1  3
4  -1.2     1  4
mean(sleep$extra) # 'mean()' function from the 'base' pkg
[1] 1.54
letters[2] # constant values from the 'base' pkg
[1] "b"
ls() # yet the workpace looks empty!
character(0)
```

Objects & functions from outside R: The working directory

To read a file from a specific folder, you should first get (and possibly set) the **working directory**, that is the folder where input files are searched and where output files are saved.

```
getwd() # returns the current working directory
[1] "C:/Users/mengh/OneDrive/Desktop/PHD/Didattica/advancedDataAnalysisPsy"
dir()[1:3] # names of the first 3 files in the WD
[1] "~$img.pptx"
[2] "~$llabus corso ADVANCED DATA ANALYSIS FOR PSYCHOLOGICAL SCIENCE.docx"
[3] "1-course-intro.Rmd"
```

```
setwd("data") # moving WD to the 'data' subfolder
setwd("C:/Users/mengh/OneDrive/Desktop") # set new WD
```

Trick with RStudio: when you start a new project (e.g., data analysis for the thesis, report), create a new **R project** (`.Rproj`) from the menu `File > New R Project` by selecting an existing folder or by creating a new one, which will be the default WD for any file related to that project.

Reading & saving a dataset (1/2)

The first step in any data analysis with R is to read a dataset and save it in the workspace. To do so, we need to use a specific function based on the data format (e.g., CSV, xlsx, txt, sav). The default R data format is RData.

```r
# file .RData
load(file = "data/studqs.RData") # import
save(qs, file = "data/studqs.RData") # export
```

R can read and export further data formats, some of which require to install additional packages.

```r
# file .CSV (comma separated values)
qs <- read.csv(file = "data/studqs.csv") # import
write.csv(x = qs,"data/studqs.csv", row.names = FALSE) # export

# file .SAV (da SPSS)
library(foreign)
qs <- read.spss("data/studqs.sav", to.data.frame=TRUE) # import
```

Reading & saving a dataset (2/2)

A fast (but poorly reproducible!) way to read a file in R is by using the function `file.choose()` rather than the name of the file. This allows reading a file without setting the WD.

```
qs <- read.csv(file = file.choose())
```

If you have doubts on the function to be used for reading a file, you can use the Rstudio menu **`File`** > **`Import Dataset`**

Hands on R:

- Open or create a file `.xlsx` on your PC, save it in a `.csv` (comma separated values) format and read it in R
- Now try to directly read the `.xlsx` file (if you have doubts, just Google "*how to read xlsx file with R*" ☺)
- In both cases, note the class and the structure of the imported object

Hands on R: Course questionnaire

1. Download the files `studqs.RData` and `studqs.csv` from Github (`4-data` folder; select `File` > `Raw` > `Download` or right click > `Save as`) or Moodle ("Data" folder), save the file in a folder and set that folder as the working directory.
2. Read both files with R: what are their class? What is the class of the variables included in each of them?
3. Use the `describe()` function from the `psych` package to compute the descriptive statistics of the variable `numVar` and use the function `hist()` to visualize its histogram plot (try changing the argument `breaks`).
4. Use the function `table()` to create a frequency table of the variable `Q02` (i.e., "*On which weekday would you prefer to have the lectures?*")
5. Repeat point #4 but only consider the students that replied "Sì" (yes) to the question `Q01` (i.e., "*Do you plan to attend all lectures?*")
6. Cross the frequency of the variables `Q02` and `Q03` (i.e., "*Will you participate even if the lectures will be on a different day than the one you prefer?*") using `table()`

Descriptive statistics (univariate)

```
x <- c(1,1,1,2,8,9) # create numeric vector
```

```
c(mean(x),median(x)) # mean & median
[1] 3.666667 1.500000
as.numeric(which.max(table(x))) # mode
[1] 1
c(var(x),sd(x)) # variance & standard dev.
[1] 14.266667  3.777124
quantile(x,probs=0.90) # 90° percentile
90%
8.5
quantile(x,probs=c(0.25,0.50,0.75,1)) # quartiles
 25%  50%  75% 100%
 1.0  1.5  6.5  9.0
round(rank(x)/length(x),2) # sample rank
[1] 0.33 0.33 0.33 0.67 0.83 1.00
```

```
table(x) # absolute frequencies
x
1 2 8 9
3 1 1 1
round(table(x)/length(x),2) # relative freq
x
   1    2    8    9
0.50 0.17 0.17 0.17
cumsum(x) # cumulative sum
[1]  1  2  3  5 13 22
cumsum(table(x)) # absolute cumulative sums
1 2 8 9
3 4 5 6
round(cumsum(table(x)/ # relative cumulative freq
               length(x)),2)
   1    2    8    9
0.50 0.67 0.83 1.00
```

Descriptive statistics (bivariate)

```
y <- -x - 1 # values inversely proport. to x
z <- round(rnorm(n=length(x)),1) # random values
(df <- data.frame(x,y,z)) # new data frame
  x   y    z
1 1  -2 -2.2
2 1  -2 -1.2
3 1  -2 -0.4
4 2  -3  0.4
5 8  -9 -2.2
6 9 -10  0.0
```

Correlation & covariance

```
cov(x,y) # covariance btw x & y
[1] -14.26667
cor(x,y) # correlation
[1] -1
```

‘

With more than 2 variables

```
cov(df) # covariance matrix
            x            y           z
x  14.26666667 -14.26666667 -0.01333333
y -14.26666667  14.26666667  0.01333333
z  -0.01333333   0.01333333  1.24266667

cor(df) # correlation matrix
            x            y            z
x  1.000000000 -1.000000000 -0.003166651
y -1.000000000  1.000000000  0.003166651
z -0.003166651  0.003166651  1.000000000
```

Graphics in R: Main functions

High-level functions (stand alone)

```
# basic function (plot depends on object class)
plot() # scatter plot

# distributions
boxplot() # boxplot
qqnorm() # quantile-quantile plot

# frequencies
barplot() # barplot (categorical variables)
hist() # histogram (continuous variables)
pie() # pie chart

# interactions
interaction.plot()
```

Lower-level functions
(add ellements to the former)

```
points() # add dots
lines() # add lines
text() # add text

# linear regression line
abline(a = ..., b = ...)

# more complex elements
rect() # add rectangles
polygon() # add polygons

# other graphical featurs
axis() # change the plot axes
legend() # add a legend
```

Graphical parameters

Run the command `?par` to see all graphical parameters that can be changed.

Some of them can be set by using the arguments of graphical functions:

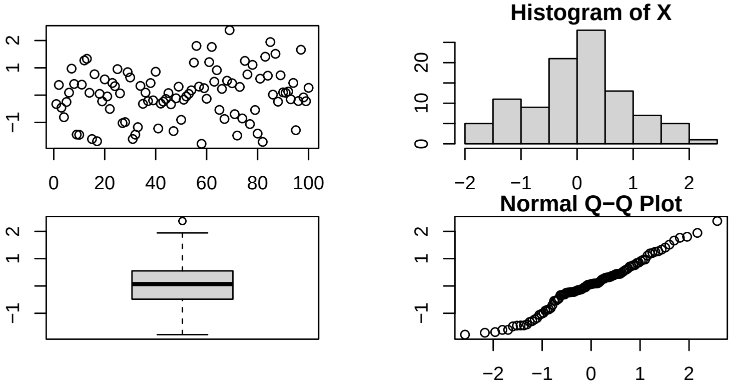
```
# dimension
cex = 2 # text & simbol dimension (multiplier)
lwd = 0.5 # line width

# color & shape
col = "red" # (see ?colors)
lty = 2 # type of line (1=solid, 2=dashed, ...)
pch = 19 # shape of dots (see ?points)

# titles
main = "Plot title"
xlab = "x-axis title"
```

Other parameters should be set within the `par()` function, which should be executed before generating the plot:

```
# margins (bottom,left,top,right)
mai # in inches
mar # in text lines

# multiple panels
mfrow=c(nrow, ncol)
# e.g. two plots next to each other
mfrow=c(2,1)
# es. one above and one below
mfrow=c(1,2)
```

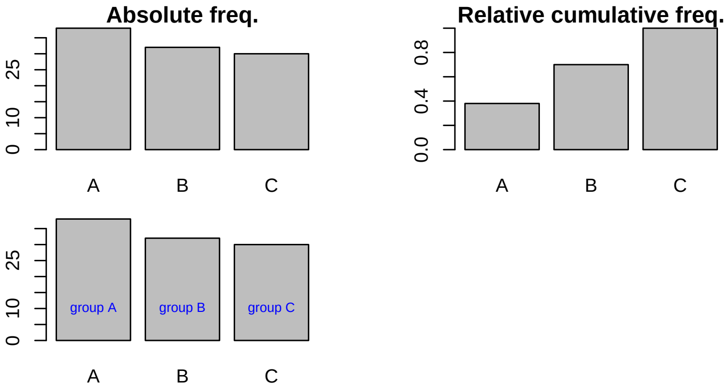
Univariate distributions: Continuous variables

```
X <- rnorm(n=100, mean=0, sd=1) # generating random values from the stand. norm. distr.
par(mfrow = c(2,2), mai = c(0.3,0.5,0.2,0.5)) # plot 4 graphs in a single window
plot(X) # scatter plot (variability)
hist(X) # histogram: frequency distr. by class
boxplot(X) # box plot: (1st & 3rd quartile +/- 1.5 IQR
qqnorm(X) # Q-Q plot: cumulative distribution of X vs. normal cumulative distr.
```



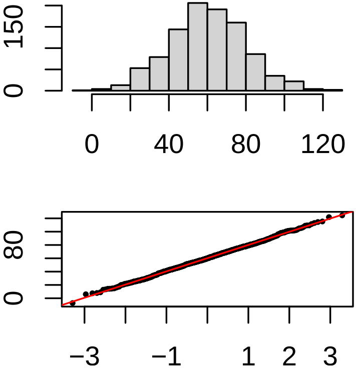
Univariate distributions: Categorical variables

```
Y <- factor(sample(x=c("A","B","C"),size=100,replace=TRUE)) # generating random factor

par(mfrow = c(2,2), mai = c(0.3,0.5,0.2,0.5)) # plot 4 graphs in a single window

barplot(table(Y), main="Absolute freq.") # bar plot: absolute frequencies

barplot(round(cumsum(table(Y)/length(Y)),2), main="Relative cumulative freq.")

barplot(table(Y)) # same graph but adding blue text

text(x=c(0.7,1.9,3.1),y=10,labels=c("group A","group B","group C"),col="blue",cex=0.7)
```

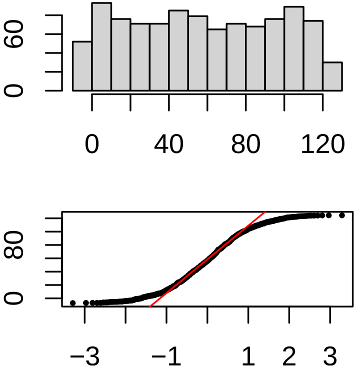


Assessing the normality of a distribution (1/3)

```
X <- rnorm(n = 1000, mean = 60, sd = 20) # norm
par(mfrow=c(2,1),mai=rep(0.3,4),mar=rep(1.8,4))
hist(X,main="",xlab="",ylab="") # histogram
qqnorm(X,cex=0.5,main="",pch=20) # Q-Q plot
qqline(X,col="red") # adding normal line
```



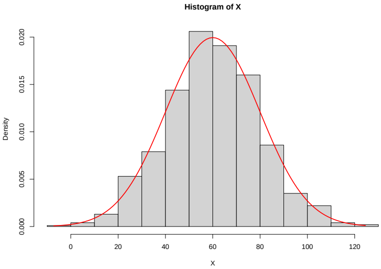
```
Y <- runif(n=1000,min=min(X),max=max(X)) # unif
par(mfrow=c(2,1),mai=rep(0.3,4),mar=rep(1.8,4))
hist(Y,main="",xlab="",ylab="") # histogram
qqnorm(Y,cex=0.5,main="",pch=20) # Q-Q plot
qqline(Y,col="red") # adding normal line
```

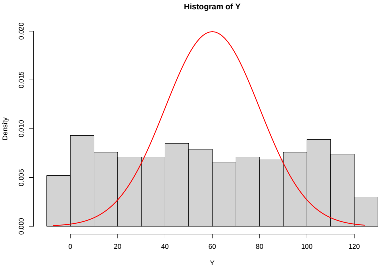


Assessing the normality of a distribution (2/3)

```
# Empirical distribution
hist(X, freq = FALSE)
# Theoretical (expected) distribution
curve(dnorm(x, mean = 60, sd = 20),
      from = min(X), to = max(X),
      col = "red", lwd = 2,
      add = TRUE)
```

```
# Empirical distribution
hist(Y, freq = FALSE, ylim = c(0,0.02))
# Theoretical (expected) distribution
curve(dnorm(x, mean = 60, sd = 20),
      from = min(Y), to = max(Y),
      col = "red", lwd = 2,
      add = TRUE)
```





Assessing the normality of a distribution (3/3)

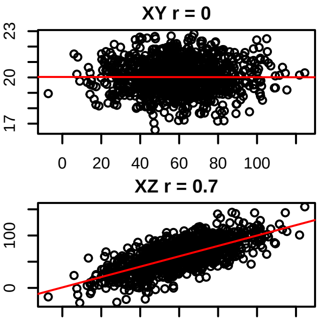
```
library(moments) # skewness & kurtosis
c(skewness(X), kurtosis(X)) # (ok if ~ 0)
[1] 0.0264805 3.0280811
# Kolmogorov-Smirnov test (H0: X is normal)
ks.test(X, y="pnorm", mean=mean(X), sd=sd(X))

    Asymptotic one-sample Kolmogorov-Smirnov test

data:  X
D = 0.015772, p-value = 0.9647
alternative hypothesis: two-sided
# Shapiro-Wilk test (H0: X is normal)
shapiro.test(X)

    Shapiro-Wilk normality test

data:  X
W = 0.99911, p-value = 0.9249
```

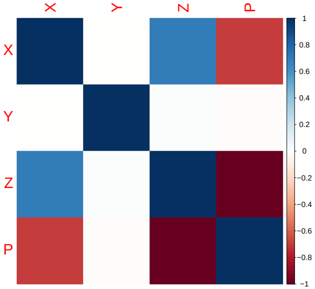
```
 # skewness & kurtosis
c(skewness(Y), kurtosis(Y)) # (ok if ~ 0)
[1] 0.02750876 1.76147352
# Kolmogorov-Smirnov test (H0: X is normal)
ks.test(Y, y="pnorm", mean=mean(Y), sd=sd(Y))

    Asymptotic one-sample Kolmogorov-Smirnov test

data:  Y
D = 0.069329, p-value = 0.0001337
alternative hypothesis: two-sided
# Shapiro-Wilk test (H0: X is normal)
shapiro.test(Y)

    Shapiro-Wilk normality test

data:  Y
W = 0.95032, p-value < 2.2e-16
```

Bivariate distributions: Continuous variables

```
Y <- rnorm(1000, mean=20, sd=1) # Y (independent from X)
Z <- X + rnorm(1000, sd = 20) # Z (proportional to X)
# Scatter plot
par(mfrow=c(2,1),mai=rep(0.3,4),mar=rep(1.8,4),cex=.5)
plot(X, Y, main = paste("XY r =",round(cor(X, Y),2)))
abline(lm(Y~X),col="red") # linear regression line
plot(X, Z, main = paste("XZ r =",round(cor(X, Z),2)))
abline(lm(Z~X),col="red") # tilde (~) with Alt + 1-2-6
```

```
df <- data.frame(X, Y, Z, P = -Z)

# Correlation plot (or heat plot)
library(corrplot)
corrplot(cor(df),method="color",
         tl.cex=2,cl.cex=1,
         mar = c(0,0,0,15))
```

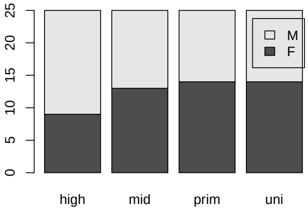




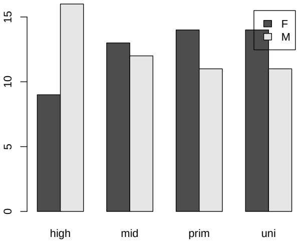
Bivariate distributions: Categorical variables

```r
# create data.frame with sex (M/F) & education level (primary, middle, highsch, univ.)
df <- data.frame(sex = sample(x = rep(c("F","M"),50), size = 100),
                 edu = sample(x = rep(c("prim","mid","high","uni"),25), size = 100))
```

```r
table(df) # crossed frequencies
   edu
sex high mid prim uni
  F    9  13   14  14
  M   16  12   11  11
par(mai=rep(0.4,4))
barplot(table(df),legend.text = TRUE) # bar plot
```

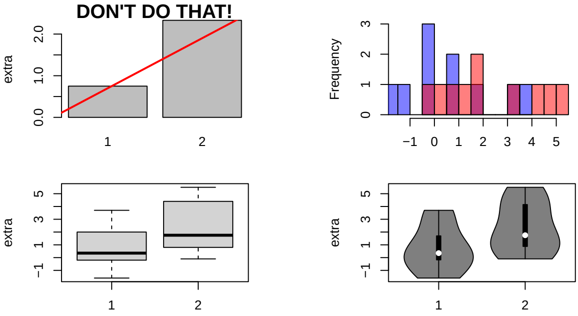


```r
# version with 'sided groups'
par(mai=rep(0.4,4))
barplot(table(df),legend.text = TRUE,
        beside = TRUE)
```



Bivariate distributions: Continuous & categorical

```r
par(mfrow = c(2,2), mai = c(0.5,1,0.2,0.5)) # setting graphical parameters
barplot(tapply(sleep$extra, sleep$group, FUN=mean), # barplot for means (DON'T DO THAT!)
        main="DON'T DO THAT!",cex.main=1.5, ylab = "extra"); abline(a=0,b=1,col="red",lwd=2)
hist(sleep[sleep$group==1,"extra"], breaks=10, col = rgb(0,0,1,alpha=0.5),
     xlim = c(min(sleep$extra), max(sleep$extra)), xlab="", main="") # hist by group
hist(sleep[sleep$group==2,"extra"], breaks=10, col = rgb(1,0,0,alpha=0.5), add = TRUE)
boxplot(extra ~ group, data = sleep) # box plot by group
vioplot::vioplot(extra ~ group, data = sleep) # violin plot: density by group
```



Saving a plot

```
# 1. open new file with PNG format
png(filename = "graFico.png", # file name
    width = 350, height = 350) # width & height

# 2. create the plot
plot(sleep$group, sleep$extra,
     xlab = "group", ylab = "extra", col = "#2E9FDF")

# 3. close the file
dev.off() # this will save the plot in your WD
pdf
  2
```

Alternative formats (see `?png`)

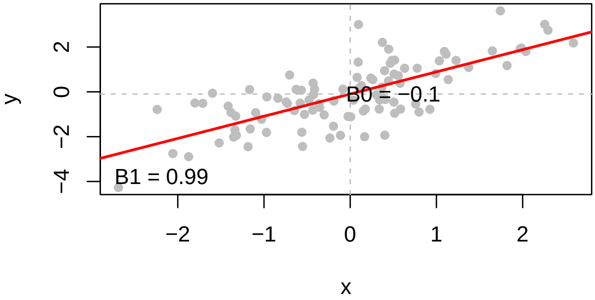
```
png() # Portable Network Graphics

bpm() # bitmap

jpeg() # Joint Photographic Experts Group

tiff() # Tagged Image File Format
```

Hands on R

1. Download and read the file `studqs.RData` (it's the same dataset used in the previous exercise)
2. Generate the histogram of the variable `numVar` separately for those that replied "Sì" (*yes*) to question `Q01` ("*Do you plan to attend all lectures?*") and those that replied "Non a tutti" (*not all lectures*); add a vertical red line corresponding to the mean value of `numVar` for the two groups
3. Generate a normal QQ plot of the variable `numVar`. Is it normally distributed?
4. Generate the box plot of the variable `numVar` by the variable `Q01`: which group has the highest median? Which has the hugest variability?
5. Create a new variable `numVar2` corresponding to `numVar2 - 2 * rnorm(nrow(qs), sd = 20)`
6. Generate the scatter plot of `numVar2` vs `numVar`, add the regression line with the `abline()`function, and compute the correlation coefficient between the two variables

Linear regression

- **Regression** allows to determinate the asymmetrical and functional relation between two variables, to quantify whether one variable (**predictor**) predicts another variable (**dependent** or **response**)
- **Linear regression** determines the link between two variables through a linear function: $y_i = \beta_0 + \beta_1 x_i + \epsilon_i$
- Such a function can be graphically represented as a **straight line**, where β_0 is the **intercept** (value assumed by `y` when `x` = 0) and β_1 is the **slope** (predicted change in `y` when `x` increases by 1 unit)



Fitting a linear regression model (1/3)

Using the dataset `gambling` from the `ADati` pkg. We want to fit models that predict `frequency` of gamig behavior based on `gender` and `risk` (risk perception).

```
data(gambling,package = "ADati")
```

R uses the `lm()` function with the arguments `formula` (syntax: `dependent ~ pred1 + pred2 +` . . ., where the symbol "~" is the **tilde**, on Windows: `Alt + 126`) and `data` (dataframe that includes the variables in the formula)

- **Null model**: `frequency` scores are only predicted by the **intercept** β_0, that is the mean value of `frequency` in the sample.

```
m0 <- lm(formula = frequency ~ 1, data = gambling)
```

```
coefficients(m0) # coefficient estimates (intercept)

(Intercept)

   10.55997

mean(gambling$frequency) # mean of the frequency variable

[1] 10.55997
```

Fitting a linear regression model (2/3)

- **Simple linear regression model**: `frequency` is predicted by the intercept β_0 (expected value of `frequency` when `gender` is `"f"`) and the slope coefficient β_1, expressing the ‘mean’ difference `m - f`.

```
m1 <- lm(formula = frequency ~ gender, data = gambling)
```

```
coefficients(m1)
(Intercept)     genderm
   9.336685    3.042012

tapply(gambling$frequency, gambling$gender, mean) # mean of frequency for f & m
        f         m
 9.336685 12.378697
```

Fitting a linear regression model (3/3)

- **Multiple linear regression model**: `frequency` is predicted by the intercept β_0 (expected `frequency` when `gender = f` and `risk = 0`) and the coefficients β_1 (‘mean’ gender difference when `risk = 0`) and β_2 (effect of `risk` net to gender differences)

```
m2 <- lm(formula = frequency ~ risk + gender, data = gambling)
```

```
coefficients(m2)

(Intercept)        risk     genderm

  18.951650   -1.393899    2.008555
```

- **Interactive model**: `frequency` is predicted by the intercept β_0 and the coefficients β_1 (‘media’ gender difference when `risk = 0`) and β_2 (effect of `risk` when `gender = "f"`) and β_3 (effect of `risk` when `gender = "m"`)

```
m3 <- lm(formula = frequency ~ risk * gender, data = gambling)
```

```
coefficients(m3)

 (Intercept)         risk      genderm risk:genderm

  16.9206907   -1.0994674    5.9744077   -0.6087177
```

Assessing linear model assumptions (1/3)

1. Independence of observations: all pairs of errors ϵ_i and ϵ_j are independent for any $i \neq j$. This assumption does not require particular procedures, just think: are my observations independent?

2. Idependence of predictors and errors: **residuals** (estimates of the error component ϵ_i) should not be associated with the predictors `gender` and `risk`.

But what are residuals?

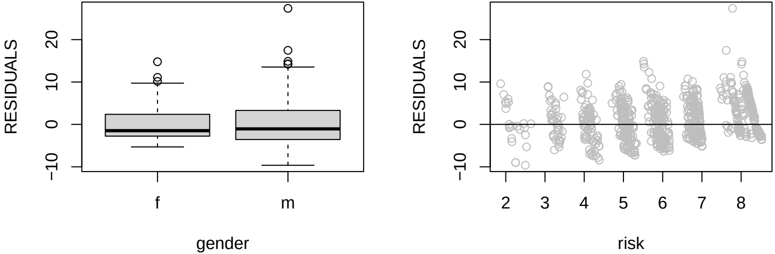
```
gambling$RESIDUALS <- residuals(m3) # model residuals with residuals()
gambling$PREDICTED <- fitted(m3) # predicted values with fitted()
head(gambling[,c("frequency","PREDICTED","RESIDUALS")],5) # RESIDUALS = OBSERVED - PREDICTED
   frequency PREDICTED  RESIDUALS
6       4.78  7.553228 -2.7732283
8      36.98  9.605418 27.3745820
10      4.78  7.553228 -2.7732283
18     11.16 11.335396 -0.1753962
19      5.62  7.828095 -2.2080952
```

Assessing linear model assumptions (2/3)

2. Idependence of predictors and errors: **residuals** (estimates of the error component ϵ_i) should not be associated with the predictors `gender` and `risk`. Here, we do not see substantal gender differences or a substantial relationship with `risk`.

3. Constant variance (Homogeneity of variances): residual variance is constant for any value of X. Here, we see that residual variance is slightly (but not substantially) higher in males than in females.

```
par(mfrow=c(1,2),mar=c(5,4,0,2)+0.1)
plot(RESIDUALS ~ gender, data=gambling)
plot(RESIDUALS ~ risk, data=gambling, col="gray")
abline(lm(RESIDUALS ~ risk, data=gambling))
```



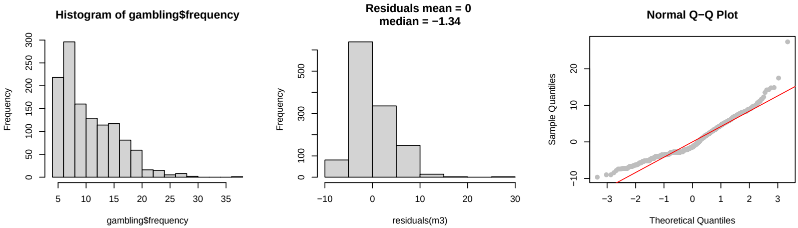
Assessing linear model assumptions (3/3)

4. Linearity: the expected value of residuals for a given X value is zero. Practically, residuals should have mean zero (as in this case).

5. Normality: residuals should be normally distributed.

Here, the fit is not very nice but we can accept it...

```
par(mfrow=c(1,3))
hist(gambling$frequency) # if residuals are normal, Y is normal as well
hist(residuals(m3),main=paste("Residuals mean =",round(mean(residuals(m3)),2),
                              "\nmedian =",round(median(residuals(m3)),2)))
qqnorm(residuals(m3), col="gray", pch=19)
qqline(residuals(m3), col="red")
```



Comparing multiple models

Likelihood ratio test: testing the ratio of the loglikelihood (probability of data functional to the parameter estimates)

```
library(lmtest)
lrtest(m0,m1,m2,m3)
Likelihood ratio test

Model 1: frequency ~ 1
Model 2: frequency ~ gender
Model 3: frequency ~ risk + gender
Model 4: frequency ~ risk * gender
  #Df  LogLik Df   Chisq Pr(>Chisq)
1   2 -3687.3
2   3 -3629.4  1 115.843  < 2.2e-16 ***
3   4 -3480.6  1 297.484  < 2.2e-16 ***
4   5 -3472.6  1  16.149  5.853e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Information criteria: penalizing models based on specific criteria (e.g., low likelihood and parsimony)

```
# Akaike information criterion (AIC)
AIC(m0,m1,m2,m3)$AIC # the lower the better
[1] 7378.599 7264.756 6969.272 6955.122

# Akaike weights: from 0 (-) to 1 (+)
library(MuMIn)
Weights(AIC(m0,m1,m2,m3)) # Aw
 model weights
[1] 0.000 0.000 0.001 0.999
```

Inference on regression coefficients

Based on the NHST approach, we can test the statistical significance of the regression coefficients. In R, such tests are conducted by default when we fit the model.

```
summary(m3)$coefficients # command summary() shows the model summary
                Estimate Std. Error    t value     Pr(>|t|)
(Intercept)   16.9206907  0.7416563  22.814733 3.337032e-96
risk          -1.0994674  0.1051711 -10.454087 1.484438e-24
genderm        5.9744077  1.0163116   5.878519 5.339754e-09
risk:genderm  -0.6087177  0.1512209  -4.025355 6.040343e-05
```

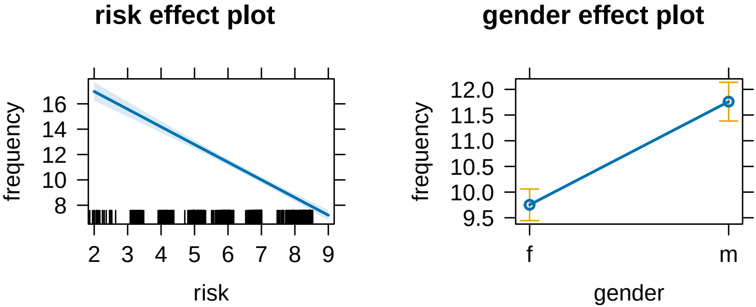
Determination coefficient R^2:

```
# the model explains the 29% of the variance in frequency
summary(m3)$r.squared
[1] 0.2965382
```

Visualizing the cofficient estimates

To visualize the estimated "effects" (regression line + 95% confidence intervals), we can use the `effects` package.

```
library(effects)
plot(allEffects(m2))
```



Credits

The present slides are partially based on:

- Pastore, M. (2015). Analisi dei dati in psicologia (e applicazioni in R). Il Mulino.
- Zandonella, C., Gambarota, F. (2021). Introduction to R (in Italian). Corso per imparare le basi di R

Other resources/books introducing R

- R Core Team. *The R Manuals*. Manuals in pdf format: https://cran.r-project.org/manuals.html (particularly: "*An Introduction to R (with many examples)*"; "*R Data Import/Export*")
- UniGlasgow - PsyTheacR. Interactive and free course on data analysis with R: https://psyteachr.github.io/
- Dalgaard, P. (2008). *Introductory statistics with R*. New York: Springer.
- Murrell (2011) R Graphics (II edition). New York: Chapman and Hall/CRC. Examples with code from the website: https://www.stat.auckland.ac.nz/~paul/RGraphics/rgraphics.html