R-Bootcamp: Series 1

Dr. Matteo Tanadini

February 2022

Preparations

In this first part of this series we are going to move our first steps in **R**. Open **R** and notepad such that you can create a script. Please do not open Rstudio or any other editor yet. Editors will be introduced and used very soon.

As the audience of this course is mixed, we are providing exercises of different level of complexity. Those that are not novices to **R** and feel comfortable can try to solve the "Going further" exercises as well.

1 Exercise: Simple Arithmetics

1.1 Question:

*Use **R** to find the results of the following calculations.*

a) $2 * 10 + 4/(3 + 1)$

b) $2 * (37 - 3^3)$

c) $\sqrt{16} + (10 + 3)$

d) $2^4 + \frac{1*3}{2+3}$

Going further (*)

e) $3 + (\sqrt[3]{27} - 8)$

f) *Guess what the results of* $1/(2^3 - 8)$ *is. Then run this computation with **R***

g) $radius = 10 \qquad Area = radius^2 * \pi$

1.2 Question:

Use the results of the computations done in a) b) and c) to continue your analysis. To do that, you must store these results into objects. Keep in mind the naming rules previously seen.

```r
## example for a)
results.a <- 2 * 10 + 4 / (3 + 1)
```

Once you created the 3 objects, run the following computations:

```r
results.a * results.b / (results.c + results.a^2)
```

1.3 Question:

Which ones of the following are valid object names?

1. results.a
2. results_1

3. 1.results.a
4. results.&_a
5. results.computation.done.in.a
6. A.res
7. A_.res
8. A

---

2 Exercise: Vectors

From here onwards do make use of Rstudio (or any other editor you may find convenient).

2.1 Question:

Try to create the following vectors using the `c()` function and the ":" colon operator. When several options are possible, prefer the shortest one (short in terms of code length).

```
[1]  2 10 13
```

```
[1] 10 11 12 13 14 15
```

```
 [1]  10  20  30  40  50  60  70  80  90 100 110 120 130 140 150 160 170 180
```

2.2 Question:

*Another option to create sequence in **R** is to use the `seq()` function to create the following vectors. Note that the function `seq()` has the following arguments: `from`, `to`, `by` and `length.out`*

```
 [1]  0  1  2  3  4  5  6  7  8  9 10
```

```
[1]  0  2  4  6  8 10
```

```
 [1]  0.00  0.83  1.67  2.50  3.33  4.17  5.00  5.83  6.67  7.50  8.33  9.17
[13] 10.00
```

2.3 Question:

Look at the statements below and try to predict the outcome without running them. You may want to note the predicted results on a sheet of paper.

```r
c(10,20,10) * c(0,1,2)

1:10 * 3

c(1,2,100) / c(1,4,0)

c(1,2,100) + c(1:6)

c(1,2,100) + c(1:7)

c("A", "B", "C") * 2

c("1", "2", "3") * 2
```

Going further (*)

Look at the statements below and try to predict the outcome without running them. You may want to note the predicted results on a sheet of paper.

```
letters[1:12]

0 / 0

c(TRUE, true, FALSE, FALSE)

c(TRUE, FALSE, TRUE, TRUE) + 1

c(T, FALSE, TRUE)
```

2.4 Question:

Given the vector `seq.1`

```
seq.1 <- c(1,2,3, 1,2,3, 1,2,3)
```

Access the following elements

1. the very first element
2. the last element
3. the 3 central elements
4. all those elements that are equal or greater than 2

---

3 Exercise: Matrices

3.1 Question:

try to create the following matrices by using the `matrix()`*,* `rbind()` *or* `cbind()` *function.*

1.

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

2.

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

3.

```
     [,1] [,2] [,3]
[1,]    3    1    1
[2,]    3    2    2
[3,]    3    3    3
[4,]    3    1    4
[5,]    3    2    5
[6,]    3    3    6
```

4.

```
     [,1] [,2]
[1,] "a"  "c"
[2,] "b"  "d"
```

5.

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,] "a"  "a"  "a"  "a"  "a"  "a" 
[2,] "b"  "b"  "b"  "b"  "b"  "b" 
[3,] "c"  "c"  "c"  "c"  "c"  "c" 
[4,] "d"  "d"  "d"  "d"  "d"  "d" 
```

Going further (*)

Create the following matrices with whatever function you find more convenient. Obviously, when there are several viable options, prefer the shortest solution.

6.

```
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
[3,]    0    0    1    0
[4,]    0    0    0    1
```

7.

```
     [,1] [,2] [,3] [,4] [,5]
[1,]  999    0    0    0    0
[2,]    0  999    0    0    0
[3,]    0    0  999    0    0
[4,]    0    0    0  999    0
[5,]    0    0    0    0  999
```

8.

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

---

4 Exercise: Data frames

4.1 Question:

Try to create the following data frame by using the ***`data.frame()`*** *function.*

```
d.jobs
```

```
             job  salary satisfaction field.exists.since field.old
1 Data scientist      $$      depends                 15     FALSE
2         banker     $$$        money               1000      TRUE
3         artist      $?         high                Inf      TRUE
4           else unknown      unknown                 NA        NA
5       gardener       $       medium                500      TRUE
```

Going further (*)

4.1.1 Can two columns of a data frame have the same name? In theory, this is difficult to achieve, but in reality this is indeed feasible. Show how and explain why this is not a "smart" thing to do.

4.1.2 Can the name of a column contain empty spaces or parentheses?

4.2 Question:

Assume you created the following three vectors:

```
t.num <- 1:10
t.alph <- LETTERS[1:10]
t.logical <- sample(x = c(TRUE, FALSE), size = 10, replace = TRUE)
```

How can you create the following data frame just by using the ***`data.frame()`*** *function and the four vectors created above.*

```
        Numbers Alphabet   Log
case 1        1        A  TRUE
case 2        2        B FALSE
case 3        3        C FALSE
case 4        4        D  TRUE
case 5        5        E  TRUE
case 6        6        F  TRUE
case 7        7        G FALSE
case 8        8        H  TRUE
case 9        9        I  TRUE
case 10      10        J FALSE
```

4.3 Question:

Given the `d.jobs` *data frame, perform the following selections*

1. get the very first row
2. get the last column (via the "$" symbol)
3. get the second and third column (via the square brackets and column numbers)
4. get the second and third column (via the square brackets and column names)
5. get all rows for which the value of `field.old` is `TRUE`
6. get all rows, but the first one
7. get all rows, but those for which `field.old` is `TRUE`

Going further (*)

1. get the last row in a programmatic way. In other words, your code must return the last row regardless of the number of rows of the data frame
2. get those rows for which the the field exists since 500 or more years
3. create a list with 3 elements
   - the first element must be a logical vector of length 10
   - the second element must be the `d.jobs` dataset
   - the last element must be a named empty list
4. access the following of the list you just created
   - the logical vector (i.e. you must access the fist "slot")
   - the `salary` column present in the second slot
   - the fist and second slot
   - the named list. NB: you must access the list via its name, not its index

5 Exercise: Functions

5.1 Question:

The function `rep()` allows the user to create vector in which a value or some values are replicates. Use it arguments "x", "times", "each" and "length.out" to create the following vectors.

1.

```
[1] 1 1 1 1 1
```

2.

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

3.

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

4.

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

5.

```
[1] "A"  "Z"  "ZU" "A"  "Z"  "ZU"
```

Going further (*)

6.

```
 [1] "A" "B" "B" "C" "C" "C" "Z" "Z" "Z" "Z"
```

7.

```
 [1] 0 2 4 6 0 2 4 6 0 2 4 6
```

5.2 Question:

The `rnorm()` function allows the user to simulate normally distributed values. Look at its help page and answer the following questions:

1. does the help page only refers to `rnorm()`?
2. what are the arguments of `rnorm()`?
3. which one(s) are compulsory?
4. are there any arguments with default values?

Going further (*)

1. is it possible to provide a vector instead of a value (scalar) to the argument "mean". For example: `rnorm(n = 10, mean = 1:10))`? What would happen in this case?
2. without looking at the help page, how can I find out what the arguments of the `pnorm()` function are?
3. in which package is the `rnorm()` function available?

5.3 Question:

*Given the following **R** command*

```r
## orginal command:
plot(x = 1:10, y = 101:110, main = "First graph", type = "b")
```

Decide whether the commands below are fully equivalent to the "original command". Make use of the help page instead of running the commands.

1.

```r
plot(1:10, 101:110, main = "First graph", type = "b")
```

2.

```r
plot(1:10, 101:110, m = "First graph", ty = "b")
```

3.

```r
plot(main = "First graph", type = "b", x = 1:10, y = 101:110)
```

4.

```r
plot(x = 1:10, y = 101:110,
     main = "First graph",
     type = "b")
```

5.

```r
plot(1:10, 101:110, "b", main = "First graph")
```

---

6 Exercise: Vectors, matrices, data frame and functions

6.1 Question:

Write a short exercise that about one or more arguments seen up to this point. Solving the exercise should take at most a few minutes. Once you have the exercise ready, swap it with your neighbour. Finally discuss the solutions together and ask to the lecturers if needed

Go quickly over all arguments seen so far. Together with your neighbour prepare a question about a topic that is not yet 100% clear to you. Upload the question in the forum of this course (on ILIAS).

---

7 Exercise: Importing data

7.1 Question:

*Working directory: Assume you have two folders on your "Desktop". One is named "folder_A" and contains an **R**-script (i.e "AnalysisFebruary.R") and a data file (i.e. "DatasetCheese.csv"). The other folder, named "folder_B", contains a data file (i.e. "DatasetHospital.xlsx").*

1. *If you start Rstudio by opening the **R**-script file, where is you working directory?*

2. *Assume you started Rstudio by opening the **R**-script file in "folder_A" and you want to import the data file "DatasetCheese.csv". What path do you need to provide to a function to import the data?*

3. *Assume you started Rstudio by opening the **R**-script file in "folder_A" and you want to import the data file "DatasetHospital.xlsx". What path do you need to provide to a function to import the data?*

3. *Assume you started Rstudio by launching the application and you want to import the data file "DatasetHospital.csv". What path do you need to provide to a function to import the data?*

4. *Assume you started Rstudio by opening the **R**-script file in "folder_A". Would the following command work?*

```r
d.Cheese <- read.csv("DatasetCheese.csv", header = TRUE)
```

5. *Assume you started Rstudio by launching the application and your current directory happens to be the Desktop. Would the following command work?*

```
d.Cheese <- read.csv("folder_A/Datasetcheese.csv", header = TRUE)
```

7.2 Question:

Try to import the following datasets without using the Rstudio interface. NB: all datasets can be found in the "DataSets" folder. The focus of this exercise in not the path but rather the functions to import data and their arguments. Therefore, you can locate of data files where you prefer.

1. *Import the dataset "Framingham.dat"*
2. *Import the dataset "Blaueeier.txt"*
3. *Import the dataset "birthrates.csv" (hint: use the function `read.csv()` or `read.csv2()`)*
4. *Import the dataset "birthrates_WithDescription.csv". Before reading the data, open the file and decide on how to import the data (hint: you may want to ignore the first few rows).*

7.3 Question:

Try to import the following datasets using the Rstudio interface.

1. *Import the dataset "Framingham.dat"*
2. *Import the dataset "Blaueeier.txt"*
3. *Import the dataset "BlaueEier.xlsx"*

7.4 Question:

Going further (*)

1. *How can you read only the first 10 rows of the "BlaueEier.txt" file.*
2. *How can you import a very large dataset in a time-efficient way? Use your friend Google to find a solution*
3. *Try to import a your own dataset that is located on you computer. Discuss the results/difficulties you may encounter with the course instructors.*
4. *Get the dataset Credibility.dat at "http://stat.ethz.ch/Teaching/Datasets"*

R-Bootcamp: Series 1

Dr. Matteo Tanadini

February 2022

```
*** Solutions ***
```

Preparations

In this first part of this series we are going to move our first steps in **R**. Open **R** and notepad such that you can create a script. Please do not open Rstudio or any other editor yet. Editors will be introduced and used very soon.

As the audience of this course is mixed, we are providing exercises of different level of complexity. Those that are not novices to **R** and feel comfortable can try to solve the "Going further" exercises as well.

1 Exercise: Simple Arithmetics

1.1 Question:

*Use **R** to find the results of the following calculations.*

a) $2 * 10 + 4/(3 + 1)$

b) $2 * (37 - 3^3)$

c) $\sqrt{16} + (10 + 3)$

d) $2^4 + \frac{1*3}{2+3}$

Going further (*)

e) $3 + (\sqrt[3]{27} - 8)$

f) *Guess what the results of* $1/(2^3 - 8)$ *is. Then run this computation with **R***

g) $radius = 10 \qquad Area = radius^2 * \pi$

Answers

Note that the prompt sign (i.e. >) is not present here. This is done to facilitate the copying of code bits.

```
## a)
2 * 10 + 4 / (3 + 1)

[1] 21

## b)
2 * (37 - 3^3)

[1] 20

## c)
sqrt(16) + (10 + 3)

[1] 17
```

```
## Note: sqrt(16) == 16^(1/2)

## d)
2^4 + (1 * 3) / (2 + 3)

[1] 17
```

Going further

```
## e)
3 + 27^(1/3) - 8

[1] -2

## f)
1 / (2^3 -8)

[1] Inf

## g)
radius  <-  10
Area <- radius^2 * pi
Area

[1] 314
```

1.2 Question:

Use the results of the computations done in a) b) and c) to continue your analysis. To do that, you must store these results into objects. Keep in mind the naming rules previously seen.

```
## example for a)
results.a <- 2 * 10 + 4 / (3 + 1)
```

Once you created the 3 objects, run the following computations:

```
results.a * results.b / (results.c + results.a^2)
```

Answers

```
results.b <- 2 * (37 - 3^3)
results.c <- sqrt(16) + (10 + 3)
##
results.a * results.b / (results.c + results.a^2)

[1] 0.92
```

1.3 Question:

Which ones of the following are valid object names?

1. results.a
2. results_1
3. 1.results.a
4. results.&_a
5. results.computation.done.in.a
6. A.res
7. A_.res
8. A

Answers

1. results.a: valid

2. results_1: valid
3. 1.results.a: invalid, starts with a number
4. results.&_a: invalid, contains a special character (i.e. &)
5. results.computation.done.in.a: valid, but too long
6. A.res: valid
7. A_.res: valid, but weird
8. A: valid, however, not very informative

When choosing a name we must keep in mind the naming rules, but also that an object name must be **short**, but nevertheless **informative**.

If your analysis is very short and contains only a few objects, than names like A, B and C may be informative enough. However, most analyses contains dozen or even hundred of objects and therefore ”results.A” or ”res.A” would be a better name.

It is important to keep in mind that for virtually all analyses there will someone else that at some point will try to understand our code.

---

2 Exercise: Vectors

From here onwards do make use of Rstudio (or any other editor you may find convenient).

2.1 Question:

Try to create the following vectors using the `c()` *function and the ”:” colon operator. When several options are possible, prefer the shortest one (short in terms of code length).*

```
[1]  2 10 13
```

```
[1] 10 11 12 13 14 15
```

```
 [1]  10  20  30  40  50  60  70  80  90 100 110 120 130 140 150 160 170 180
```

Answers

```
c(2,10,13)

[1]  2 10 13

10:15

[1] 10 11 12 13 14 15

## same as c(10,11,12,13,14,15)
## same as seq(from = 10, to = 15, by = 1)

1:18 * 10

 [1]  10  20  30  40  50  60  70  80  90 100 110 120 130 140 150 160 170 180
```

2.2 Question:

*Another option to create sequence in **R** is to use the* `seq()` *function to create the following vectors. Note that the function* `seq()` *has the following arguments:* `from`*,* `to`*,* `by` *and* `length.out`

```
 [1]  0  1  2  3  4  5  6  7  8  9 10
```

```
[1]  0  2  4  6  8 10
```

```
 [1]  0.00  0.83  1.67  2.50  3.33  4.17  5.00  5.83  6.67  7.50  8.33  9.17
[13] 10.00
```

Answers

```
seq(from = 0, to = 10, by = 1)

 [1]  0  1  2  3  4  5  6  7  8  9 10

seq(from = 0, to = 10, by = 2)

[1]  0  2  4  6  8 10

seq(from = 0, to = 10, length.out = 13)

 [1]  0.00  0.83  1.67  2.50  3.33  4.17  5.00  5.83  6.67  7.50  8.33  9.17
[13] 10.00
```

2.3 Question:

Look at the statements below and try to predict the outcome without running them. You may want to note the predicted results on a sheet of paper.

```
c(10,20,10) * c(0,1,2)

1:10 * 3

c(1,2,100) / c(1,4,0)

c(1,2,100) + c(1:6)

c(1,2,100) + c(1:7)

c("A", "B", "C") * 2

c("1", "2", "3") * 2
```

Going further (*)

Look at the statements below and try to predict the outcome without running them. You may want to note the predicted results on a sheet of paper.

```
letters[1:12]

0 / 0

c(TRUE, true, FALSE, FALSE)

c(TRUE, FALSE, TRUE, TRUE) + 1

c(T, FALSE, TRUE)
```

Answers

```
c(10,20,10) * c(0,1,2)

[1]  0 20 20

1:10 * 3

 [1]  3  6  9 12 15 18 21 24 27 30

c(1,2,100) / c(1,4,0)

[1] 1.0 0.5 Inf

## dividing by zero yields Infinity

c(1,2,100) + c(1:6)

[1]   2   4 103   5   7 106
```

```
## note "recycling" here

c(1,2,100) + c(1:7)

Warning in c(1, 2, 100) + c(1:7):  longer object length is not a multiple of shorter object length

[1]   2   4 103   5   7 106   8

## note "recycling" here + a warning for non-matching length

c("A", "B", "C") * 2

Error in c("A", "B", "C") * 2:  non-numeric argument to binary operator

## strings and numeric vectors cannot be summed

c("1", "2", "3") * 2

Error in c("1", "2", "3") * 2:  non-numeric argument to binary operator

## same, strings and numeric vectors cannot be summed
```

Going further (*)

```
letters[1:12]

 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l"

0 / 0

[1] NaN

## note that "NaN" stand for "Not A Number"

c(TRUE, true, FALSE, FALSE)

Error in eval(expr, envir, enclos):  object 'true' not found

c(TRUE, FALSE, TRUE, TRUE) + 1

[1] 2 1 2 2

## logicals and numeric vectors can be summed
## in that case TRUE is evaluated as being the value 1,
## while FALSE 0

c(T, FALSE, TRUE, F)

[1]  TRUE FALSE  TRUE FALSE

## note that "T" and "F" are shortcuts for TRUE and FALSE respectively
## note that both "T" and "F" are not reserved names
T <- 6
## in general, is bad practice to abbreviate TRUE and FALSE
```

2.4 Question:

Given the vector `seq.1`

```
seq.1 <- c(1,2,3, 1,2,3, 1,2,3)
```

Access the following elements

1. the very first element
2. the last element
3. the 3 central elements
4. all those elements that are equal or greater than 2

Answers

```
## 1.
seq.1[1]

[1] 1

## 2.
seq.1[9]

[1] 3

## 3.
seq.1[c(4:6)]

[1] 1 2 3

## 4.
index.seq.1 <- seq.1 >= 2
index.seq.1

[1] FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE

seq.1[index.seq.1]

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

Note that the vector `seq.1` could more conveniently be created with the `rep()` function.

```
rep(x = 1:3, times = 3)

[1] 1 2 3 1 2 3 1 2 3

## or
rep(x = 1:3, length.out = 9)

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

---

3 Exercise: Matrices

3.1 Question:

try to create the following matrices by using the `matrix()`, `rbind()` or `cbind()` function.

1.

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

2.

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

3.

```
     [,1] [,2] [,3]
[1,]    3    1    1
[2,]    3    2    2
[3,]    3    3    3
[4,]    3    1    4
[5,]    3    2    5
[6,]    3    3    6
```

4.

```
     [,1] [,2]
[1,] "a"  "c"
[2,] "b"  "d"
```

5.

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,] "a"  "a"  "a"  "a"  "a"  "a" 
[2,] "b"  "b"  "b"  "b"  "b"  "b" 
[3,] "c"  "c"  "c"  "c"  "c"  "c" 
[4,] "d"  "d"  "d"  "d"  "d"  "d" 
```

Going further (*)

Create the following matrices with whatever function you find more convenient. Obviously, when there are several viable options, prefer the shortest solution.

6.

```
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
[3,]    0    0    1    0
[4,]    0    0    0    1
```

7.

```
     [,1] [,2] [,3] [,4] [,5]
[1,]  999    0    0    0    0
[2,]    0  999    0    0    0
[3,]    0    0  999    0    0
[4,]    0    0    0  999    0
[5,]    0    0    0    0  999
```

8.

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

Answers

```
## 1.
matrix(1:10, ncol = 5)

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

## 2.
rbind(1:5, 6:10)

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

## 3.
cbind(3, 1:3, 1:6)

     [,1] [,2] [,3]
[1,]    3    1    1
[2,]    3    2    2
[3,]    3    3    3
[4,]    3    1    4
[5,]    3    2    5
[6,]    3    3    6

## note recycling here

## 4.
matrix(data = c("a", "b", "c", "d"), nrow = 2)
```

```
     [,1] [,2]
[1,] "a"  "c"
[2,] "b"  "d"
```

```
## 5.
matrix(data = letters[1:4], nrow = 4, ncol = 6)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,] "a"  "a"  "a"  "a"  "a"  "a"
[2,] "b"  "b"  "b"  "b"  "b"  "b"
[3,] "c"  "c"  "c"  "c"  "c"  "c"
[4,] "d"  "d"  "d"  "d"  "d"  "d"
```

Going further (*)

```
## 6.
diag(nrow = 4)
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
[3,]    0    0    1    0
[4,]    0    0    0    1
```

```
## 7.
diag(x = 999, nrow = 5, ncol = 5)
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]  999    0    0    0    0
[2,]    0  999    0    0    0
[3,]    0    0  999    0    0
[4,]    0    0    0  999    0
[5,]    0    0    0    0  999
```

```
## 8.
matrix(data = 1:12, ncol = 4, nrow = 3, byrow = TRUE)
```

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

Note that another option to create the very last matrix is to use the `t()` function (i.e. transpose).

```
## same as 8.
M.8 <- matrix(data = 1:12, ncol = 3, nrow = 4)
t(M.8)
```

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

---

4 Exercise: Data frames

4.1 Question:

Try to create the following data frame by using the ***`data.frame()`*** *function.*

```
d.jobs
```

```
             job  salary satisfaction field.exists.since field.old
1 Data scientist      $$      depends                 15     FALSE
2         banker     $$$        money               1000      TRUE
3         artist      $?         high                Inf      TRUE
4           else unknown      unknown                 NA        NA
```

```
5       gardener        $       medium                    500       TRUE
```

Answers

```
job <- c("Data scientist", "banker", "artist", "else", "gardener")
salary <- c("$$", "$$$", "$?", "unknown", "$")
satisfaction <- c("depends", "money", "high", "unknown", "medium")
field.exists.since <- c(15, 1000, Inf, NA, 500)
field.old <- field.exists.since > 100
##
d.jobs <- data.frame(job, salary, satisfaction,
                     field.exists.since, field.old)
```

Going further (*)

4.1.1 Can two columns of a data frame have the same name? In theory, this is difficult to achieve, but in reality this is indeed feasible. Show how and explain why this is not a "smart" thing to do.

4.1.2 Can the name of a column contain empty spaces or parentheses?

Answers

4.1.1 "Jaein", two columns of a given data frame should not have the same name. When the `data.frame()` function is used, **R** makes sure that this does not happen by adding a suffix to columns with the same name.

```
df_sameName <- data.frame(job, satisfaction, satisfaction)
df_sameName

             job satisfaction satisfaction.1
1 Data scientist      depends        depends
2         banker        money          money
3         artist         high           high
4           else      unknown        unknown
5       gardener       medium         medium
```

This is a very sensible property. Naming must be unanbiguous to avoid confusion.

Note, however, that you can force the name of two columns of a data frame to be the same. This should never be done as it may lead to unnoticed mistakes. Indeed, in such a case it would not be clear any more to which of the two equally named cloumns you are trying to refer to.

```
colnames(df_sameName)[3] <- "satisfaction"
df_sameName$satisfaction <- NA
df_sameName

             job satisfaction satisfaction
1 Data scientist           NA      depends
2         banker           NA        money
3         artist           NA         high
4           else           NA      unknown
5       gardener           NA       medium

##
df_sameName$satisfaction

[1] NA NA NA NA NA
```

4.1.2. The answer is "Jaein". Again when the `data.frame()` function is used, **R** makes sure that a column name does not contain empty spaces or parentheses. Those are replaced with dots.

```
d.jobs.new <- data.frame(JobName = job,
                         "salary (categories)" = salary,
                         satisfaction,
                         field_old = field.old)
```

```
colnames(d.jobs.new)

[1] "JobName"              "salary..categories." "satisfaction"
[4] "field_old"
```

There are a few things to note here.

- the column name ”JobName” was passed to the `data.frame()` function without quotes as it does not contains spaces or parentheses
- empty spaces and parentheses are replaced with dots
- names can contain underscores

Note, however, that you can attribute names with space and parentheses by using the `colnames()` function.

```
colnames(d.jobs.new)[2] <- "salary (categories)"
colnames(d.jobs.new)

[1] "JobName"              "salary (categories)" "satisfaction"
[4] "field_old"
```

Further Note: keep in mind that empty spaces in a string are NOT ignored in **R**. This is especially relevant when the empty spaces are located at the very end of a string.

```
colnames(df_sameName) <- c("job", "satisfaction", "satisfaction ")
df_sameName[ ,3] <- 999
df_sameName

             job satisfaction satisfaction 
1 Data scientist           NA           999
2         banker           NA           999
3         artist           NA           999
4           else           NA           999
5       gardener           NA           999

df_sameName[ ,"satisfaction "]

[1] 999 999 999 999 999

df_sameName[ ,"satisfaction"]

[1] NA NA NA NA NA

##
"satistfaction" == "satistfaction "

[1] FALSE
```

4.2 Question:

Assume you created the following three vectors:

```
t.num <- 1:10
t.alph <- LETTERS[1:10]
t.logical <- sample(x = c(TRUE, FALSE), size = 10, replace = TRUE)
```

How can you create the following data frame just by using the `data.frame()` function and the four vectors created above.

```
       Numbers Alphabet   Log
case 1       1        A FALSE
case 2       2        B  TRUE
case 3       3        C  TRUE
case 4       4        D FALSE
case 5       5        E  TRUE
case 6       6        F  TRUE
case 7       7        G  TRUE
case 8        8        H FALSE
case 9        9        I  TRUE
case 10      10        J FALSE
```

Answers

The solution is to specify column names directly in the `data.frame()` call.

```
d.2 <- data.frame(Numbers = t.num, Alphabet = t.alph, Log = t.logical,
                  row.names = paste("case", 1:10))
d.2
```

```
        Numbers Alphabet   Log
case 1        1        A FALSE
case 2        2        B  TRUE
case 3        3        C  TRUE
case 4        4        D FALSE
case 5        5        E  TRUE
case 6        6        F  TRUE
case 7        7        G  TRUE
case 8        8        H FALSE
case 9        9        I  TRUE
case 10      10        J FALSE
```

Alternatively, you can create a "unnamed" data frame and the add the row- and column names afterwards.

```
d.2_again <- data.frame(t.num, t.alph, t.logical)
##
colnames(d.2_again) <- c("Numbers","Alphabet","Log")
row.names(d.2_again) = paste("case", 1:10)
##
d.2_again
```

```
        Numbers Alphabet   Log
case 1        1        A FALSE
case 2        2        B  TRUE
case 3        3        C  TRUE
case 4        4        D FALSE
case 5        5        E  TRUE
case 6        6        F  TRUE
case 7        7        G  TRUE
case 8        8        H FALSE
case 9        9        I  TRUE
case 10      10        J FALSE
```

You can test whether these two ways to create `d.2` are truly fully equivalent via the `identical()` function.

```
identical(d.2, d.2_again)
```

```
[1] TRUE
```

4.3 Question:

Given the `d.jobs` data frame, perform the following selections

1. get the very first row
2. get the last column (via the "$" symbol)
3. get the second and third column (via the square brackets and column numbers)
4. get the second and third column (via the square brackets and column names)
5. get all rows for which the value of `field.old` is `TRUE`
6. get all rows, but the first one

7. get all rows, but those for which `field.old` is `TRUE`

Going further (*)

1. get the last row in a programmatic way. In other words, your code must return the last row regardless of the number of rows of the data frame
2. get those rows for which the the field exists since 500 or more years
3. create a list with 3 elements
   - the first element must be a logical vector of length 10
   - the second element must be the `d.jobs` dataset
   - the last element must be a named empty list
4. access the following of the list you just created
   - the logical vector (i.e. you must access the fist "slot")
   - the `salary` column present in the second slot
   - the fist and second slot
   - the named list. NB: you must access the list via its name, not its index

Answers

```
## 0. show the whole data frame
d.jobs

## 1. get the very first row
d.jobs[1, ]

## 2. get the last comlumn (via the dollar symbol)
d.jobs$field.old

## 3. get the second and third column  (via the square brackets and column numbers)
d.jobs[ , 2:3]
## same as
## d.jobs[ , c(2,3)]

## 4. get the second and third column  (via the square brackets and column names)
d.jobs[ , c("salary", "satisfaction")]

## 5. get all rows for which the value of "field.old" is "TRUE"
ind.jobs <- d.jobs$field.old
d.jobs[ind.jobs, ]
## note NAs (more comes later)
## same as
## d.jobs[d.jobs£field.old, ]

## 6. get all rows, but the first one
d.jobs[-1, ]

## 7. get all rows, but those for which "field.old" is "TRUE"
d.jobs[!d.jobs$field.old, ]
## note that the inverse of a logical vector is obtained with a leading "!"
```

Going further (*)

```
## 1. get the last row in a programmatic way. In other words, your code must return the last row regardless of t
tail(d.jobs, n = 1)
```

```
      job salary satisfaction field.exists.since field.old
5 gardener      $       medium                500      TRUE
```

```
## same as
nrows.d.jobs <- nrow(d.jobs)
d.jobs[nrows.d.jobs, ]

       job salary satisfaction field.exists.since field.old
5 gardener      $       medium                500      TRUE

## 2. get those rows for which the the field exists since 500 or more years
d.jobs[d.jobs$field.exists.since >= 500, ]

        job salary satisfaction field.exists.since field.old
2    banker    $$$        money               1000      TRUE
3    artist     $?         high                Inf      TRUE
NA     <NA>   <NA>         <NA>                 NA        NA
5  gardener      $       medium                500      TRUE

## 3. create a list with 3 elements
l.1 <- list(c(TRUE,FALSE,TRUE,TRUE),
            d.jobs,
            EmptyList = list())

## 4. access the following of the list you just created
# 4.1
l.1[[1]]

[1]  TRUE FALSE  TRUE  TRUE

## note that typing l.1[1] would also work, but returns a list not a vector.
str(l.1[1])

List of 1
 $ : logi [1:4] TRUE FALSE TRUE TRUE

str(l.1[[1]])

 logi [1:4] TRUE FALSE TRUE TRUE

##
## 4.2
l.1[[2]]$salary

[1] "$$"      "$$$"     "$?"      "unknown" "$"

##
## 4.3
l.1[1:2]

[[1]]
[1]  TRUE FALSE  TRUE  TRUE

[[2]]
             job  salary satisfaction field.exists.since field.old
1 Data scientist      $$      depends                 15     FALSE
2         banker     $$$        money               1000      TRUE
3         artist      $?         high                Inf      TRUE
4           else unknown      unknown                 NA        NA
5       gardener       $       medium                500      TRUE

##
## 4.4
l.1$EmptyList

list()
```

5 Exercise: Functions

5.1 Question:

The function ***rep()*** *allows the user to create vector in which a value or some values are replicates. Use it arguments "x", "times", "each" and "length.out" to create the following vectors.*

1.

```
[1] 1 1 1 1 1
```

2.

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

3.

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

4.

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

5.

```
[1] "A"  "Z"  "ZU" "A"  "Z"  "ZU"
```

Going further (*)

6.

```
 [1] "A" "B" "B" "C" "C" "C" "Z" "Z" "Z" "Z"
```

7.

```
 [1] 0 2 4 6 0 2 4 6 0 2 4 6
```

Answers

```
## 1.
rep(1, times = 5)

[1] 1 1 1 1 1

## 2.
rep(3:1, times = 5)

 [1] 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1

## 3.
rep(3:1, each = 4)

 [1] 3 3 3 3 2 2 2 2 1 1 1 1

## 4.
rep(1:3, length.out = 11)

 [1] 1 2 3 1 2 3 1 2 3 1 2

## 5.
rep(c("A", "Z", "ZU"), times = 2)

[1] "A"  "Z"  "ZU" "A"  "Z"  "ZU"
```

Going further (*)

```
## 6.
rep(c("A", "B", "C", "Z"), times = 1:4)

 [1] "A" "B" "B" "C" "C" "C" "Z" "Z" "Z" "Z"

## 7.
rep(0:3*2, times = 3)

 [1] 0 2 4 6 0 2 4 6 0 2 4 6
```

5.2 Question:

The `rnorm()` function allows the user to simulate normally distributed values. Look at its help page and answer the following questions:

1. does the help page only refers to `rnorm()`?
2. what are the arguments of `rnorm()`?
3. which one(s) are compulsory?
4. are there any arguments with default values?

Going further (*)

1. is it possible to provide a vector instead of a value (scalar) to the argument ”mean”. For example: `rnorm(n = 10, mean = 1:10))`? What would happen in this case?
2. without looking at the help page, how can I find out what the arguments of the `pnorm()` function are?
3. in which package is the `rnorm()` function available?

Answers

1. does the help page only refers to `rnorm()`?
   No, this page also contain the help about `dnorm()`, `pnorm()` and `qnorm()`. This is often the case for closely related function that share most arguments.
2. what are the arguments of `rnorm()`?
   ”n”, ”mean” and ”sd”:
3. which one(s) are compulsory?
   only ”n” is compulsory. Without setting ”n”, the function won’t work
4. are there any arguments with default values?
   Yes, default value for ”mean” is zero, while ”sd” is set to 1 by default

Going further (*)

1. is it possible to provide a vector instead of a value (scalar) to the argument ”mean” (e.g. `rnorm(n = 10, mean = 1:10)`). What would happen in this case?
   Yes, the argument ”mean” accepts vectors. In that case, the mean value of the first simulated value would be 1, for the second simulated value would be 2 and so on till the last simulated value that would be simulated from a normal distribution with mean 10.
2. without looking at the help page, how can I find out what the arguments of the `pnorm()` function are?
   Simply by typing `args(pnorm)`.
3. in which package is the `rnorm()` function available? In {*stats*}. This can be read at the top left of the help page.

5.3 Question:

*Given the following **R** command*

```r
## orginal command:
plot(x = 1:10, y = 101:110, main = "First graph", type = "b")
```

Decide whether the commands below are fully equivalent to the ”original command”. Make use of the help page instead of running the commands.

1.

```r
plot(1:10, 101:110, main = "First graph", type = "b")
```

2.

```r
plot(1:10, 101:110, m = "First graph", ty = "b")
```

3.

```r
plot(main = "First graph", type = "b", x = 1:10, y = 101:110)
```

4.

```r
plot(x = 1:10, y = 101:110,
     main = "First graph",
     type = "b")
```

5.

```r
plot(1:10, 101:110, "b", main = "First graph")
```

Answers

All 5 commands will indeed produce the same plot. However, some are more easy to read than others.

---

6 Exercise: Vectors, matrices, data frame and functions

6.1 Question:

Write a short exercise that about one or more arguments seen up to this point. Solving the exercise should take at most a few minutes. Once you have the exercise ready, swap it with your neighbour. Finally discuss the solutions together and ask to the lecturers if needed

Go quickly over all arguments seen so far. Together with your neighbour prepare a question about a topic that is not yet 100% clear to you. Upload the question in the forum of this course (on ILIAS).

---

7 Exercise: Importing data

7.1 Question:

*Working directory: Assume you have two folders on your "Desktop". One is named "folder_A" and contains an **R**-script (i.e "AnalysisFebruary.R") and a data file (i.e. "DatasetCheese.csv"). The other folder, named "folder_B", contains a data file (i.e. "DatasetHospital.xlsx").*

1. *If you start Rstudio by opening the **R**-script file, where is you working directory?*

2. *Assume you started Rstudio by opening the **R**-script file in "folder_A" and you want to import the data file "DatasetCheese.csv". What path do you need to provide to a function to import the data?*

3. *Assume you started Rstudio by opening the **R**-script file in "folder_A" and you want to import the data file "DatasetHospital.xlsx". What path do you need to provide to a function to import the data?*

3. *Assume you started Rstudio by launching the application and you want to import the data file "DatasetHospital.csv". What path do you need to provide to a function to import the data?*

4. *Assume you started Rstudio by opening the **R**-script file in "folder_A". Would the following command work?*

```r
d.Cheese <- read.csv("DatasetCheese.csv", header = TRUE)
```

5. *Assume you started Rstudio by launching the application and your current directory happens to be the Desktop. Would the following command work?*

```r
d.Cheese <- read.csv("folder_A/Datasetcheese.csv", header = TRUE)
```

Answers

1. "Desktop/folder_A"

2. "DatasetCheese.csv". Indeed, the data file is located in the working directory.

3. It depends. Very often the default working directory is "Documents" or the root folder "home". In my case the default working directory is "home". Therefore, I should read the data with the following path "Desktop/folder_B/DatasetHospital.csv"

4. Yes.

5. No. Again, **R** is case sensitive. The correct file name is "DatasetCheese.csv" (with a capital "C").

7.2 Question:

Try to import the following datasets without using the Rstudio interface. NB: all datasets can be found in the "DataSets" folder. The focus of this exercise in not the path but rather the functions to import data and their arguments. Therefore, you can locate of data files where you prefer.

1. *Import the dataset "Framingham.dat"*

2. *Import the dataset "Blaueeier.txt"*

3. *Import the dataset "birthrates.csv" (hint: use the function `read.csv()` or `read.csv2()`)*

4. *Import the dataset "birthrates_WithDescription.csv". Before reading the data, open the file and decide on how to import the data (hint: you may want to ignore the first few rows).*

Answers

Note that to get to the "DataSets" folder I must "go up" three time (i.e. "../../../"). This may differ for you (depends you where you located the "DataSets" folder).

1.

```
# getwd()
d.Fram <- read.table("../../../DataSets/Framingham.dat", header = TRUE)
str(d.Fram)
```

```
'data.frame': 16 obs. of  5 variables:
 $ sex : int  1 1 1 1 1 1 1 1 2 2 ...
 $ age : int  1 1 1 1 2 2 2 2 1 1 ...
 $ chol: int  1 2 3 4 1 2 3 4 1 2 ...
 $ y   : int  13 18 40 57 13 33 35 49 6 5 ...
 $ n   : int  340 408 421 362 123 176 174 183 542 552 ...
```

2.

```
d.BlauerEier <- read.table("../../../DataSets/Blaueeier.txt", header = TRUE)
str(d.BlauerEier)
```

```
'data.frame': 16 obs. of  2 variables:
 $ Jahr      : int  2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 ...
 $ Teilnehmer: int  42 65 98 147 99 182 223 327 234 252 ...
```

To be entirely correct, there is no dataset with the name specified above (remember case-sensitive).

3.

```
d.birthR <- read.csv2("../../../DataSets/birthrates.csv")
## same as
d.birthR <- read.csv("../../../DataSets/birthrates.csv", sep = ";")
str(d.birthR)
```

```
'data.frame': 182 obs. of  1 variable:
 $ fertility.fertTotal.infantMort.catholic.single24.single49.eAgric.eIndustry.eCommerce.eTransport.eAdmin.germa
```

Note that for `read.csv()` and `read.csv2()` functions the argument ”header” is, by default, set to TRUE . Therefore, we don’t need to specifcy it.

4.

```
d.birthR_no_des <- read.csv("../../../DataSets/birthrates_WithDescription.csv",
                            skip = 8)
str(d.birthR_no_des)

'data.frame': 182 obs. of  25 variables:
 $ fertility  : num  48 62.5 65.5 61.5 51.5 52.4 48.6 53.3 53 57 ...
 $ fertTotal  : num  22.4 29.2 30.3 32.9 25.1 23.3 21.6 24.8 23.5 28 ...
 $ infantMort : num  16.4 16.3 20.2 20.7 19.2 20 16.8 21.3 19.6 17.7 ...
 $ catholic   : num  7.74 6.68 2.78 3.98 9.58 ...
 $ single24   : num  80.5 85 83.3 78 76.7 80.2 81.1 80.5 81.7 81.5 ...
 $ single49   : num  16.2 19.3 15.1 8.5 12.3 19.9 19.8 11.1 15.2 10.4 ...
 $ eAgric     : num  60.9 66.7 57.3 71.7 36 33.7 52.5 51.6 52.1 28.9 ...
 $ eIndustry  : num  31.5 26 34.1 22.1 55.9 52.8 35.7 39.7 38.9 56.2 ...
 $ eCommerce  : num  3.4 1.8 2.7 1.4 3 5.7 4.8 2.8 3.2 6.2 ...
 $ eTransport : num  1.9 2.3 3.2 2.1 3.1 4.7 4 3.6 3.5 5.1 ...
 $ eAdmin     : num  2.1 3 2.6 2.7 1.9 2.5 2.8 2.1 2.2 3.2 ...
 $ german     : num  99.6 99.7 99.8 99.5 99.4 ...
 $ french     : num  0.26 0.13 0.07 0.19 0.14 0.39 0.32 0.09 0.16 0.45 ...
 $ italian    : num  0.1 0.05 0.07 0.26 0.44 0.65 0.47 0.2 0.53 0.55 ...
 $ romansh    : num  0.02 0.04 0.01 0 0.02 0.06 0.02 0.03 0.01 0.04 ...
 $ gradeHigh  : int  24 22 17 20 23 28 30 25 25 27 ...
 $ gradeLow   : int  14 9 14 13 12 13 11 12 9 9 ...
 $ educHigh   : int  30 29 24 24 23 43 32 22 27 38 ...
 $ bornLocal  : num  60 67.6 67.8 66.1 49.8 47.5 50.1 61.4 55 44.4 ...
 $ bornForeign: num  1.9 3.4 2.3 1.9 3.2 5.4 4 2.1 2.8 7.4 ...
 $ sexratio   : int  956 963 939 1024 896 879 898 907 875 978 ...
 $ canton     : chr  "Zurich" "Zurich" "Zurich" "Zurich" ...
 $ district   : chr  "Affoltern" "Andelfingen" "Bulaech" "Dielsdorf" ...
 $ altitude   : chr  "low" "low" "low" "low" ...
 $ language   : chr  "german" "german" "german" "german" ...
```

In none of these exercises we use absolute paths to import data. This was done on purpose as **good practise is to use relative paths**. The reason for that being reproducibility (ask the course instructors for more information or...more to come on day 4).

7.3 Question:

Try to import the following datasets using the Rstudio interface.

1. *Import the dataset ”Framingham.dat”*

2. *Import the dataset ”Blaueeier.txt”*

3. *Import the dataset ”BlaueEier.xlsx”*

Answers

1. ”Import Dataset” → ”From text”

2. Fully correct answer would be: There is no such data file. ”Import Dataset” → ”From text”

3. ”Import Dataset” → ”From Excel”

7.4 Question:

Going further (*)

1. *How can you read only the first 10 rows of the ”BlaueEier.txt” file.*

2. *How can you import a very large dataset in a time-efficient way? Use your friend Google to find a solution*

3. *Try to import a your own dataset that is located on you computer. Discuss the results/difficulties you may encounter with the course instructors.*

4. *Get the dataset Credibility.dat at "http://stat.ethz.ch/Teaching/Datasets"*

Answers

1.

```
d.BlauerEier.10Rows <- read.table("../../../DataSets/BlaueEier.txt", header = TRUE,
                                  nrows = 10)
str(d.BlauerEier.10Rows)
```

```
'data.frame': 10 obs. of  2 variables:
 $ Jahr      : int  2004 2005 2006 2007 2008 2009 2010 2011 2012 2013
 $ Teilnehmer: int  42 65 98 147 99 182 223 327 234 252
```

2. There is a huge variety of functions that you could use to import large datasets in **R**. Very likely the most time-efficient functions are not to be found in the **R**-base functions, but rather in add-on packages that where specifically developed for these tasks (e.g. the `fread()` function found in `data.table` package).

Note that packages to import data are evolving very fast. The best package to import data today may be outdated in 1 one. As a matter of fact, also the functions that Rstudio uses/suggest to import data are changing quite frequently.

3. As you are reading a own dataset, there is non common solution here. The main point, was to try something else and discuss results/difficulties with the lecturers.

4.

```
d.cred <- read.table("http://stat.ethz.ch/Teaching/Datasets/NDK/Credibility.dat",
                     header = TRUE, sep = ";")
str(d.cred)
```

```
'data.frame': 22 obs. of  5 variables:
 $ Region : chr  "R1" "R2" "R3" "R4" ...
 $ Type   : chr  "NC" "NC" "NC" "NC" ...
 $ NoClaim: int  2595 763 3302 233 1670 3099 631 2428 116 8246 ...
 $ AvClaim: int  2889 2490 2273 2403 2480 2558 2471 2705 2836 2651 ...
 $ Sigma  : int  4867 4704 3681 4143 3885 4232 4493 4671 4724 4152 ...
```

R-Bootcamp: Series 2

Dr. Matteo Tanadini

February 2022

1 Exercise: Graphics I

1.1 Question:

Using the `plot()` function and built-in dataset `swiss` create a graph with the following properties

- do not use the formula interface
- plot "Fertility" on the y-axis
- plot "Catholic" on the x-axis
- set the colour of the observations to green
- add a title to the graph

1.2 Question:

Using the `plot()` function and built-in dataset `swiss` create a graph with the following properties

- use the formula interface
- plot "Education" on the y-axis
- plot "Agriculture" on the x-axis
- set the colour of the points to any colour of your preference[1]
- plot the observations as filled triangles
- set the axes labels as "edu" and "agri"
- set the limits of the x-axis to (-5, 100)
- add a title

Going further (*)

Using the `plot()` function and built-in dataset `swiss` create a graph with the following properties

- use the formula interface
- plot "Education" on the y-axis
- plot "Agriculture" on the x-axis
- plot the observations with "Fertility" equal or greater than 76 with a different colour to all the others
- use the letter "z" as plotting character
- set the expansion factor to be proportional to the variable "Infant.Mortality"
- add a title written on two lines

[1]You may want to type `colours()` or google "R + colours" to find all the available options.

- add a subtitle that contains the expression $\pi * 10 \neq \sqrt(10)$

1.3 Question:

We now want you to familiarise with other basic graphical functions. Create the following graphs.

1. an histogram of the "Fertility" variable found in `swiss` (hint: use `hist()`)
2. a boxplot of where the "Sepal.Length" variable found in `iris` is plotted against "Species" (hint: use `boxplot()`)
3. a graph where all variable of the `swiss` dataset are plotted against each other (hint: use `pairs()`)

---

2 Exercise: Graphics II

2.1 Question:

Consider the built-in dataset `Loblolly` (type `?Loblolly` to get more information). Using the `plot()`, `abline()` and `par()` functions create a graph with the following properties.

- the device region is divided in two part such that a graph can be accommodated on the left and one on the right-hand side
- the first plot on the left is a scatterplot of "height" against "age" (i.e. "height" is on the y-axis)
- the plot on the rights is a boxplot where the "height" of the trees is plotted against the variable "Seed"
- dots in the scatterplot must be rendered with the colour orange
- the inner area of the boxplots must be rendered with the colour purple
- the border of the boxplots and the whiskers must be rendered with the colour "cyan"
- both graphs must have a title
- on the scatterplot there is an horizontal dashed line flat on 40
- on the boxplot graph write some text above the very first boxplot on the left
- all graph titles must be written with the colour magenta. Use the function `par()` function to do that

2.2 Question:

Use the Rstudio interface to export this graph as a pdf file. Save the pdf on your Desktop.

3 Exercise: Graphics III

3.1 Question:

The dataset used in this exercise is `Orthodont`, which is about the growth of 29 children. Load the add-on package {`nlme`} such that the dataset is available (Then type `?Orthodont` if you wish more information about the dataset).

*To better understand the way multipanel conditioning and grouping work, run the following **R**-commands. You may try to guess what the resulting plot will be before running the command.*

```r
library(lattice)
library(nlme) ## for the dataset
# ? Orthodont

## 1. No multipanel conditioning, points only
xyplot(distance ~ age, data = Orthodont)
##
## 2. No conditioning, points and a regression line
xyplot(distance ~ age, data = Orthodont,
       type = c("p", "r"))
##
## 3. No conditioning, grouping (Sex), points and a regression line
xyplot(distance ~ age, data = Orthodont, groups = Sex,
       type = c("p", "r"))
##
## 4. multipanel conditining for Sex, points and a regression line
xyplot(distance ~ age | Sex, data = Orthodont,
       type = c("p", "r"))
```

3.2 Question:

We now do the same with the {ggplot2} package.

```r
library(ggplot2)

## 1. No multipanel conditioning, points only
ggplot(data = Orthodont,
       mapping = aes(y = distance,
                     x = age)) +
  geom_point()
##
## 2. No conditioning, points and a regression line
ggplot(data = Orthodont,
       mapping = aes(y = distance,
                     x = age)) +
  geom_point() +
  geom_smooth(method = "lm")
```

```
`geom_smooth()` using formula 'y ~ x'
```

```r
##
## 3. No conditioning, grouping (Sex), points and a regression line
ggplot(data = Orthodont,
       mapping = aes(y = distance,
                     x = age,
                     group = Sex,
                     colour = Sex)) +
  geom_point() +
  geom_smooth(method = "lm")
```

```
`geom_smooth()` using formula 'y ~ x'
```

```r
##
## 4. multipanel conditining for Sex, points and a regression line
ggplot(data = Orthodont,
       mapping = aes(y = distance,
                     x = age)) +
  geom_point() +
  geom_smooth(method = "lm") +
  facet_wrap(. ~ Sex)
```

```
`geom_smooth()` using formula 'y ~ x'
```

3.3 Question:

For the next exercise you can choose whether to work with {`lattice`}, {`ggplot2`} or even try with both.

1. Now try to produce a plot where each person is plotted in a different panel, use points and a regression line in each panel. If you use `ggplot()` make sure that the confidence interval around the regression line is turned off (makes little sense with 4 observations to estimate a CI).
2. than replace points and regression lines with lines connecting points for each Subject. In addition you should add a grid in the background. This comes as a default with `ggplot()` but not with `xyplot()` (you may want to type `?xyplot` for help).
3. finally, modify the previous plot and use different points and lines colours for the two genders.

3.4 Question:

NB: before starting to work on this exercise, read the whole assignment here. Indeed, this exercise involves some collaborative work.

*Using any of the built-in datasets (type `data()` for the whole list) produce a graph with the base **R** functions (i.e. `plot()`, `boxplot()`,...). Then save it as a jpeg file.*

Using the same dataset produce a graph with either `xyplot()` or `ggplot()`.

Exchange the two graphs with another participant in the room and without looking at his/her code try to reproduce the exact same graph.

Finally, discuss with the other participant your solution. Again, you may want to involve in this discussion the course instructors.

R-Bootcamp: Series 2

Dr. Matteo Tanadini

February 2022

```
*** Solutions ***
```

1 Exercise: Graphics I

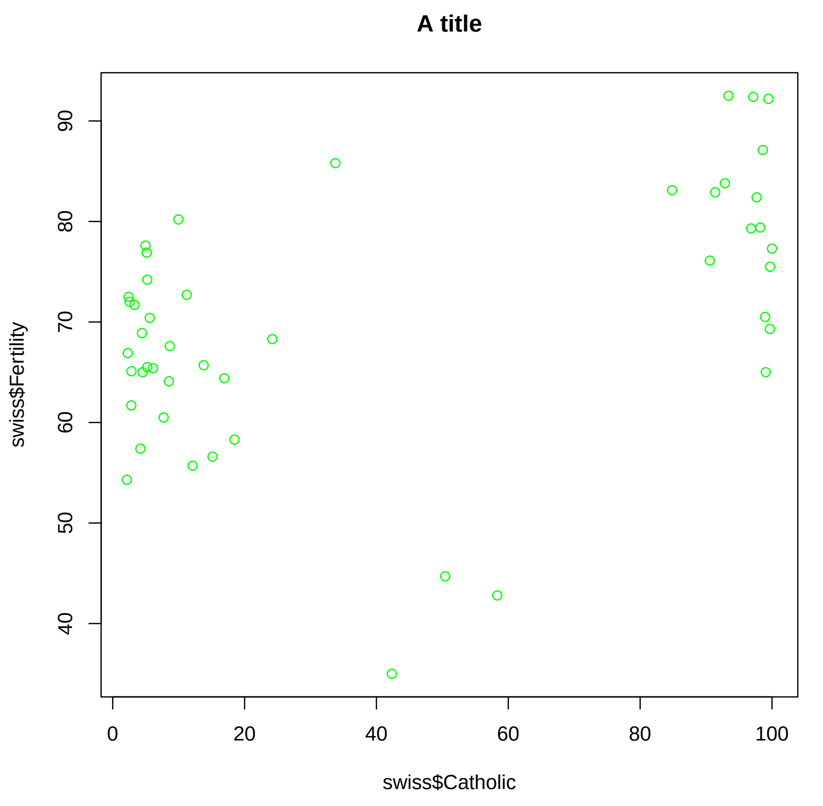
1.1 Question:

Using the `plot()` function and built-in dataset `swiss` create a graph with the following properties

- do not use the formula interface
- plot "Fertility" on the y-axis
- plot "Catholic" on the x-axis
- set the colour of the observations to green
- add a title to the graph

Answers

```r
plot(y = swiss$Fertility,
     x = swiss$Catholic,
     col = "green",
     main = "A title")
```



1.2 Question:

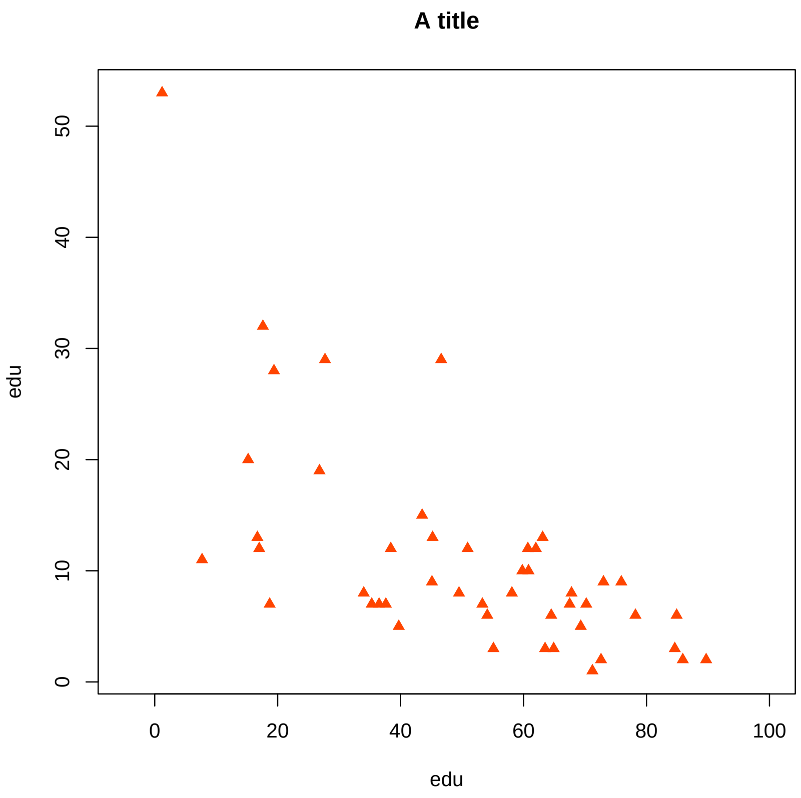
Using the `plot()` function and built-in dataset `swiss` create a graph with the following properties

- use the formula interface
- plot ”Education” on the y-axis
- plot ”Agriculture” on the x-axis
- set the colour of the points to any colour of your preference[1]
- plot the observations as filled triangles
- set the axes labels as ”edu” and ”agri”
- set the limits of the x-axis to (-5, 100)
- add a title

Answers

```
plot(Education ~ Agriculture,
     data = swiss,
     col = "orangered",
     pch = 17,
     ylab = "edu", xlab = "edu",
     xlim = c(-5, 100),
     main = "A title")
```

[1]You may want to type `colours()` or google ”R + colours” to find all the available options.



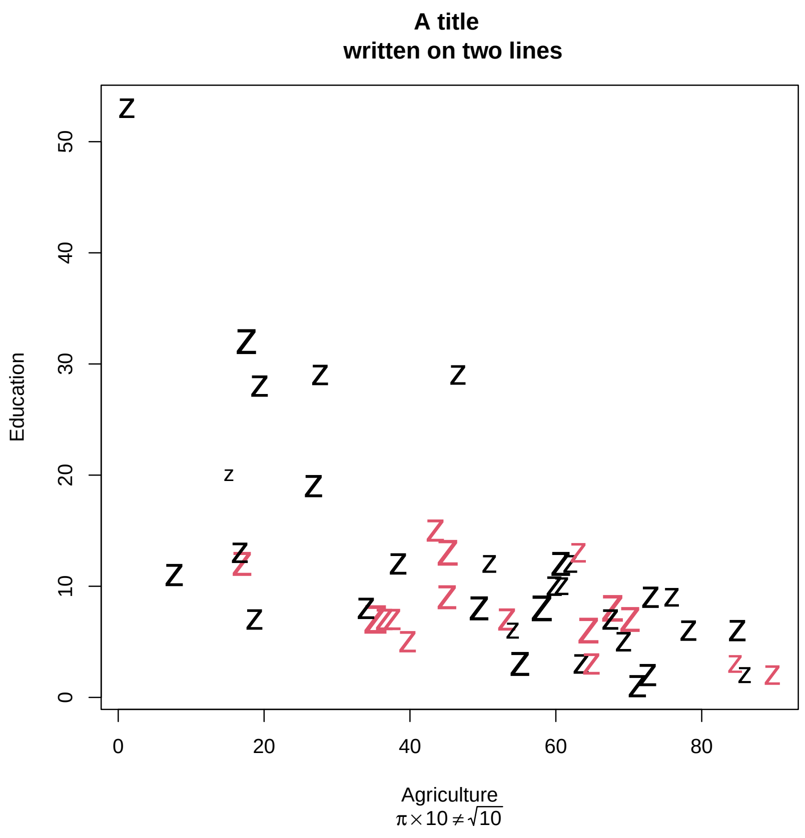
Going further (*)

Using the `plot()` function and built-in dataset `swiss` create a graph with the following properties

- use the formula interface
- plot "Education" on the y-axis
- plot "Agriculture" on the x-axis
- plot the observations with "Fertility" equal or greater than 76 with a different colour to all the others
- use the letter "z" as plotting character
- set the expansion factor to be proportional to the variable "Infant.Mortality"
- add a title written on two lines
- add a subtitle that contains the expression $\pi * 10 \neq \sqrt(10)$

Answers

```
v.colour <- (swiss$Fertility >= 76) + 1
plot(Education ~ Agriculture,
     data = swiss,
     col = v.colour,
     pch = "z",
     cex = Infant.Mortality / 10,
     main = "A title \n written on two lines",
     sub = expression(pi %*% 10 != sqrt(10)))
```



1.3 Question:

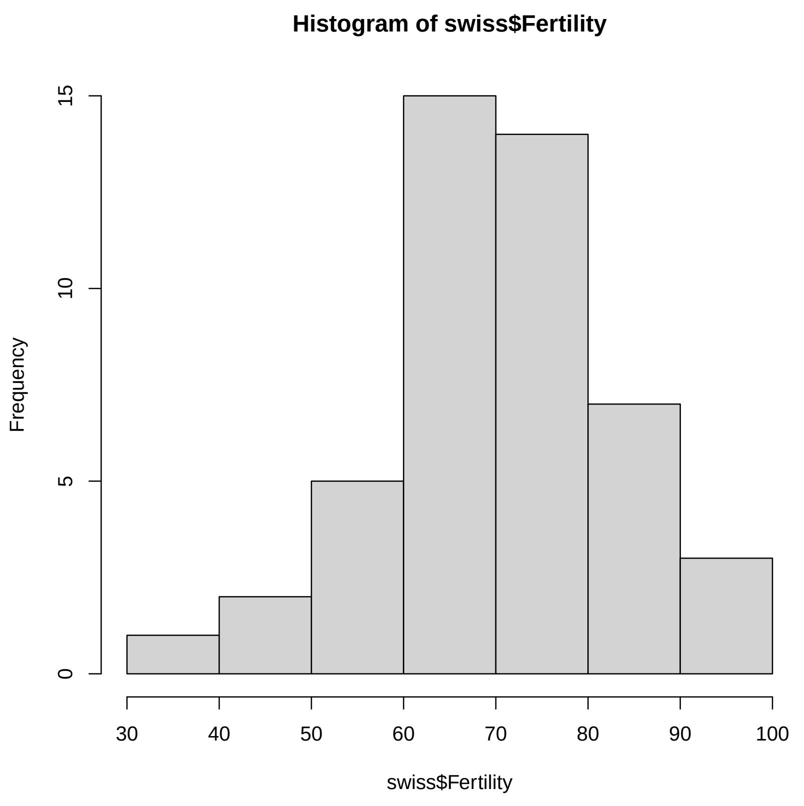
We now want you to familiarise with other basic graphical functions. Create the following graphs.

1. an histogram of the "Fertility" variable found in `swiss` (hint: use `hist()`)
2. a boxplot of where the "Sepal.Length" variable found in `iris` is plotted against "Species" (hint: use `boxplot()`)
3. a graph where all variable of the `swiss` dataset are plotted against each other (hint: use `pairs()`)

Answers

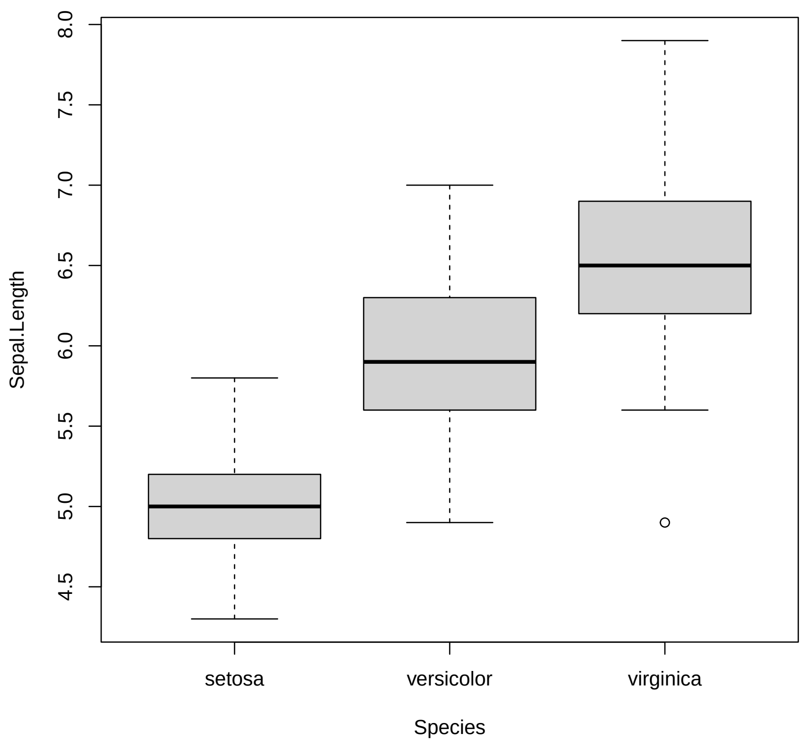
1.

```
hist(x = swiss$Fertility)
```



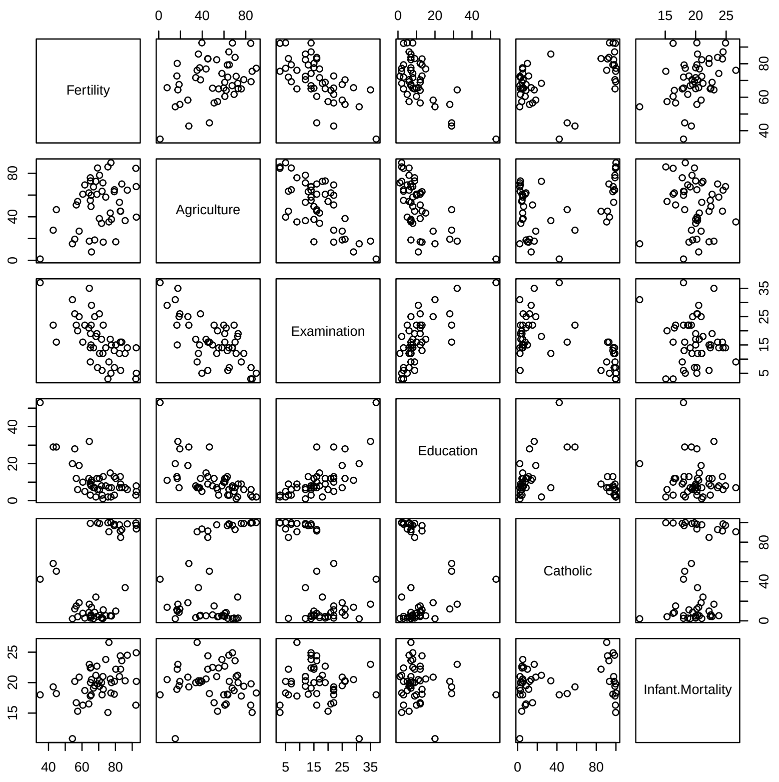
2.

```
boxplot(Sepal.Length ~ Species, data = iris)
```



3.

```
pairs(swiss)
```



---

2 Exercise: Graphics II

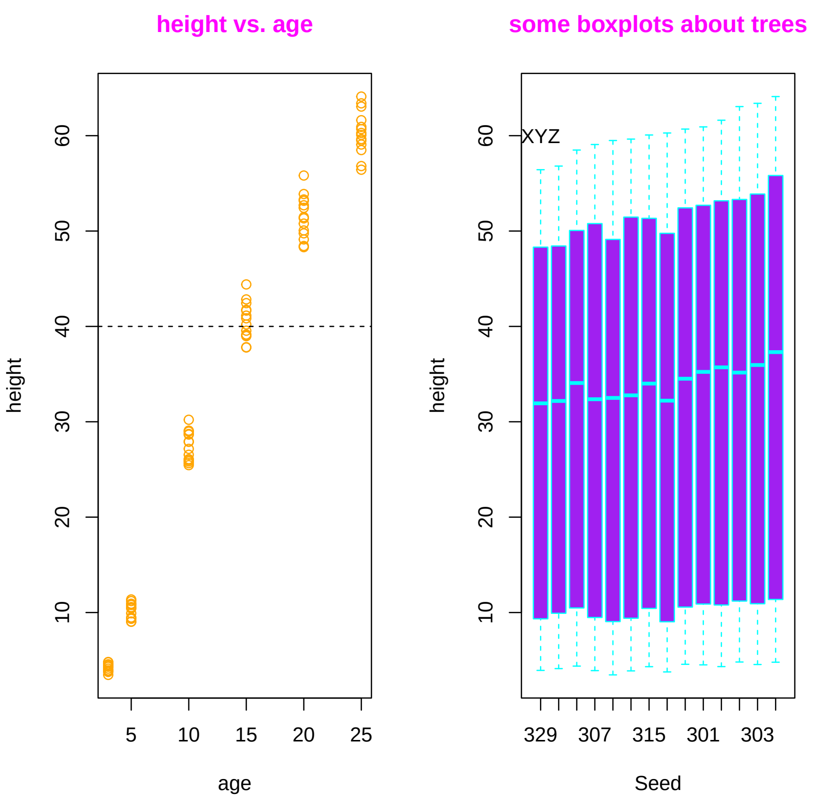
2.1 Question:

Consider the built-in dataset `Loblolly` *(type* `?Loblolly` *to get more information). Using the* `plot()`*,* `abline()` *and* `par()` *functions create a graph with the following properties.*

- the device region is divided in two part such that a graph can be accommodated on the left and one on the right-hand side
- the first plot on the left is a scatterplot of "height" against "age" (i.e. "height" is on the y-axis)
- the plot on the rights is a boxplot where the "height" of the trees is plotted against the variable "Seed"
- dots in the scatterplot must be rendered with the colour orange
- the inner area of the boxplots must be rendered with the colour purple
- the border of the boxplots and the whiskers must be rendered with the colour "cyan"
- both graphs must have a title
- on the scatterplot there is an horizontal dashed line flat on 40
- on the boxplot graph write some text above the very first boxplot on the left
- all graph titles must be written with the colour magenta. Use the function `par()` function to do that

Answers

```
par(mfrow = c(1, 2),
    col.main = "magenta")
plot(height ~ age, data = Loblolly,
     col = "orange",
     main = "height vs. age")
abline(h = 40, lty = "dashed")
boxplot(height ~ Seed, data = Loblolly,
        col = "purple", border = "cyan",
        main = "some boxplots about trees")
text(1, 60, "XYZ")
```



2.2 Question:

Use the Rstudio interface to export this graph as a pdf file. Save the pdf on your Desktop.

Answers

Create your graph and then simply click on "Export" → "Same as PDF". Set the directory where to save the file by clicking on "Directory" button.

3 Exercise: Graphics III

3.1 Question:

The dataset used in this exercise is `Orthodont`, which is about the growth of 29 children. Load the add-on package {`nlme`} such that the dataset is available (Then type `?Orthodont` if you wish more information about the dataset).

*To better understand the way multipanel conditioning and grouping work, run the following **R**-commands. You may try to guess what the resulting plot will be before running the command.*

```r
library(lattice)
library(nlme) ## for the dataset
# ? Orthodont

## 1. No multipanel conditioning, points only
xyplot(distance ~ age, data = Orthodont)
##
## 2. No conditioning, points and a regression line
xyplot(distance ~ age, data = Orthodont,
       type = c("p", "r"))
##
## 3. No conditioning, grouping (Sex), points and a regression line
xyplot(distance ~ age, data = Orthodont, groups = Sex,
       type = c("p", "r"))
##
## 4. multipanel conditining for Sex, points and a regression line
xyplot(distance ~ age | Sex, data = Orthodont,
       type = c("p", "r"))
```

Answers

For the sake of brevity the plots are not reported here. Simply run the code yourself. If you have any questions, do not hesitate to ask the course instructors.

3.2 Question:

We now do the same with the {`ggplot2`} package.

```r
library(ggplot2)

## 1. No multipanel conditioning, points only
ggplot(data = Orthodont,
       mapping = aes(y = distance,
                     x = age)) +
  geom_point()
##
## 2. No conditioning, points and a regression line
ggplot(data = Orthodont,
       mapping = aes(y = distance,
                     x = age)) +
  geom_point() +
  geom_smooth(method = "lm")
```

```
`geom_smooth()` using formula 'y ~ x'
```

```r
##
## 3. No conditioning, grouping (Sex), points and a regression line
ggplot(data = Orthodont,
       mapping = aes(y = distance,
                     x = age,
                     group = Sex,
                     colour = Sex)) +
  geom_point() +
  geom_smooth(method = "lm")
```

```
`geom_smooth()` using formula 'y ~ x'
```

```r
##
## 4. multipanel conditining for Sex, points and a regression line
ggplot(data = Orthodont,
       mapping = aes(y = distance,
                     x = age)) +
  geom_point() +
  geom_smooth(method = "lm") +
  facet_wrap(. ~ Sex)
```

```
`geom_smooth()` using formula 'y ~ x'
```

Answers

For the sake of brevity the plots are not reported here. Simply run the code yourself. If you have any questions, do not hesitate to ask the course instructors.

3.3 Question:

For the next exercise you can choose whether to work with {`lattice`}, {`ggplot2`} or even try with both.

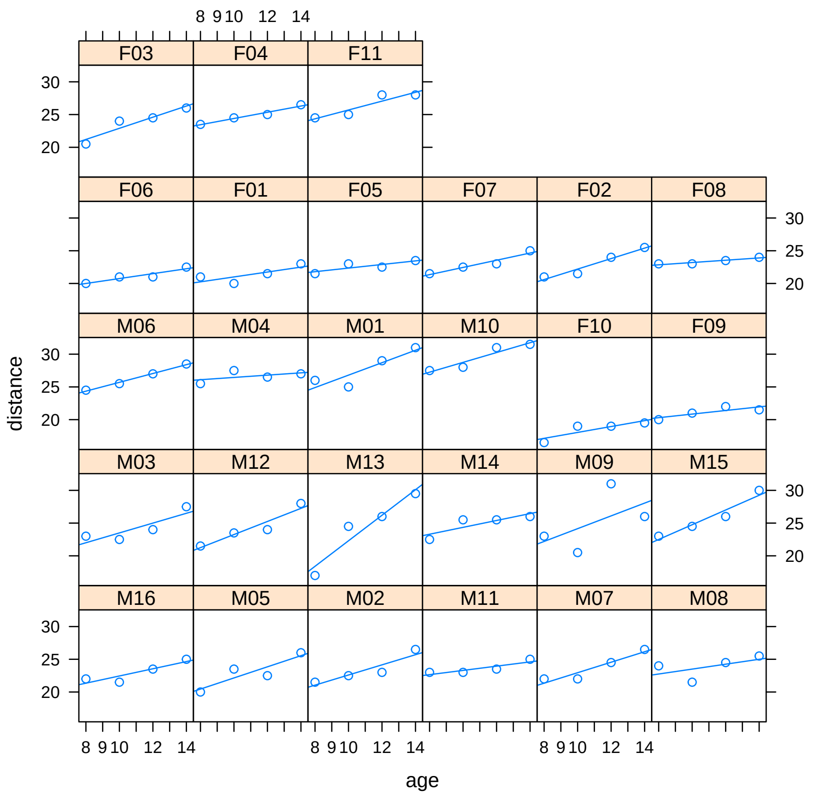
1. Now try to produce a plot where each person is plotted in a different panel, use points and a regression line in each panel. If you use `ggplot()` make sure that the confidence interval around the regression line is turned off (makes little sense with 4 observations to estimate a CI).
2. than replace points and regression lines with lines connecting points for each Subject. In addition you should add a grid in the background. This comes as a default with `ggplot()` but not with `xyplot()` (you may want to type `?xyplot` for help).
3. finally, modify the previous plot and use different points and lines colours for the two genders.

Answers

`xyplot()` solutions:

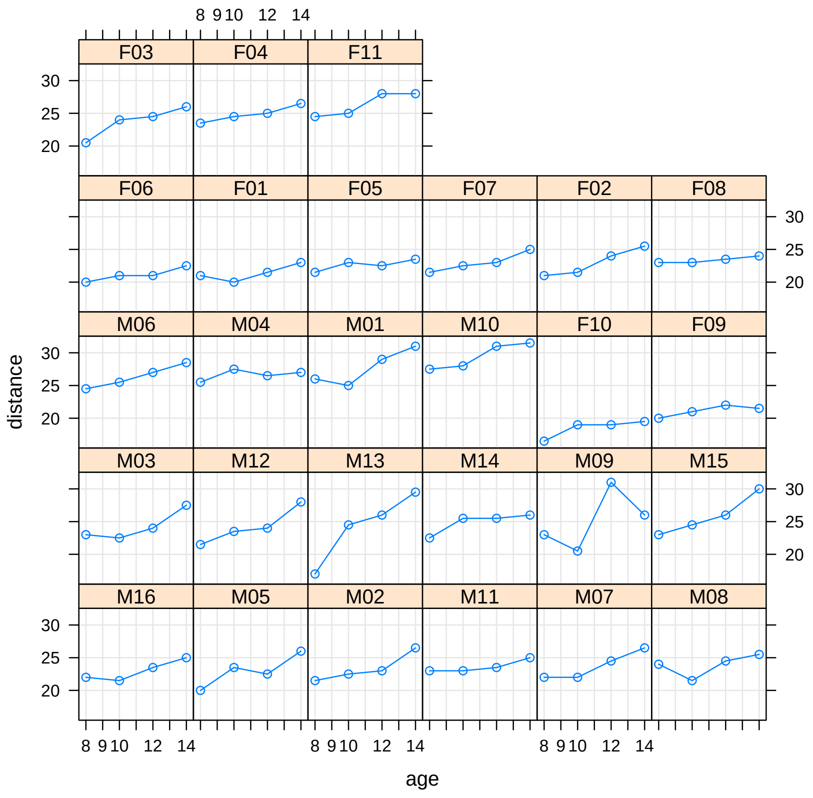
1.

```
xyplot(distance ~ age | Subject, data = Orthodont,
       type = c("p", "r"))
```



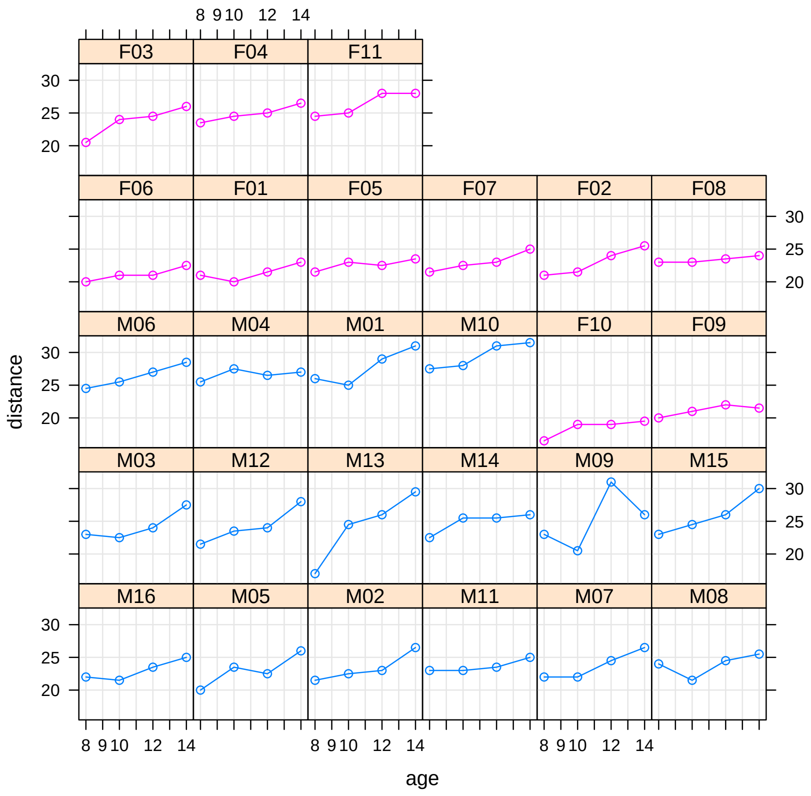
2.

```r
xplot(distance ~ age | Subject, data = Orthodont,
      type = c("b", "g"))
```



3.

```r
xyplot(distance ~ age | Subject, data = Orthodont,
       groups = Sex,
       type = c("b", "g"))
```

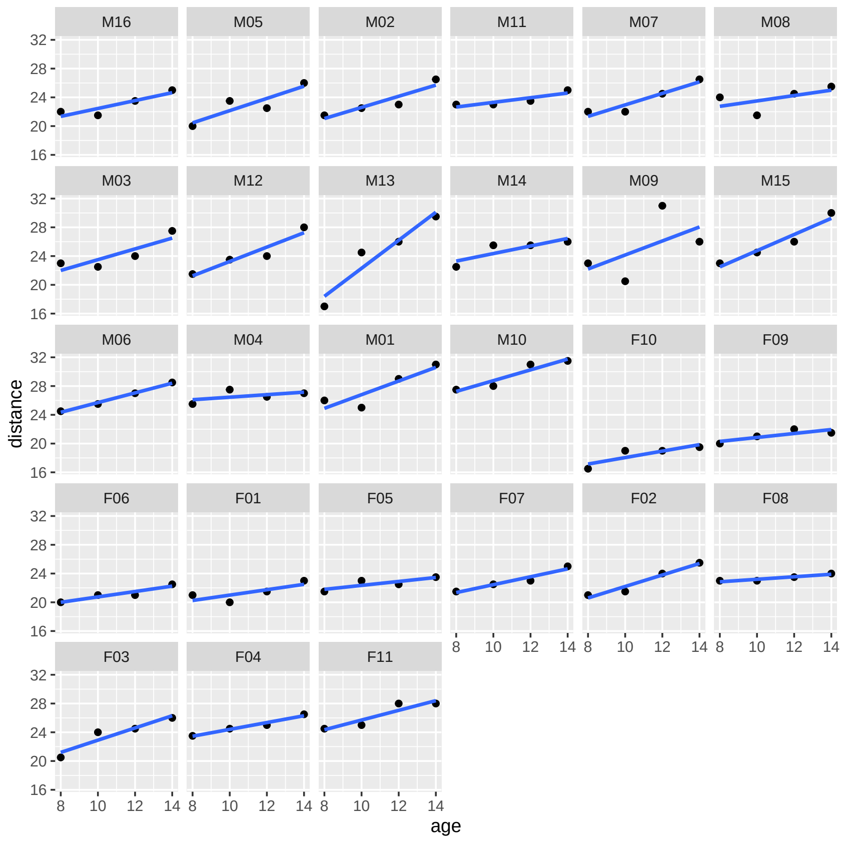


`ggplot()` solutions:

1.

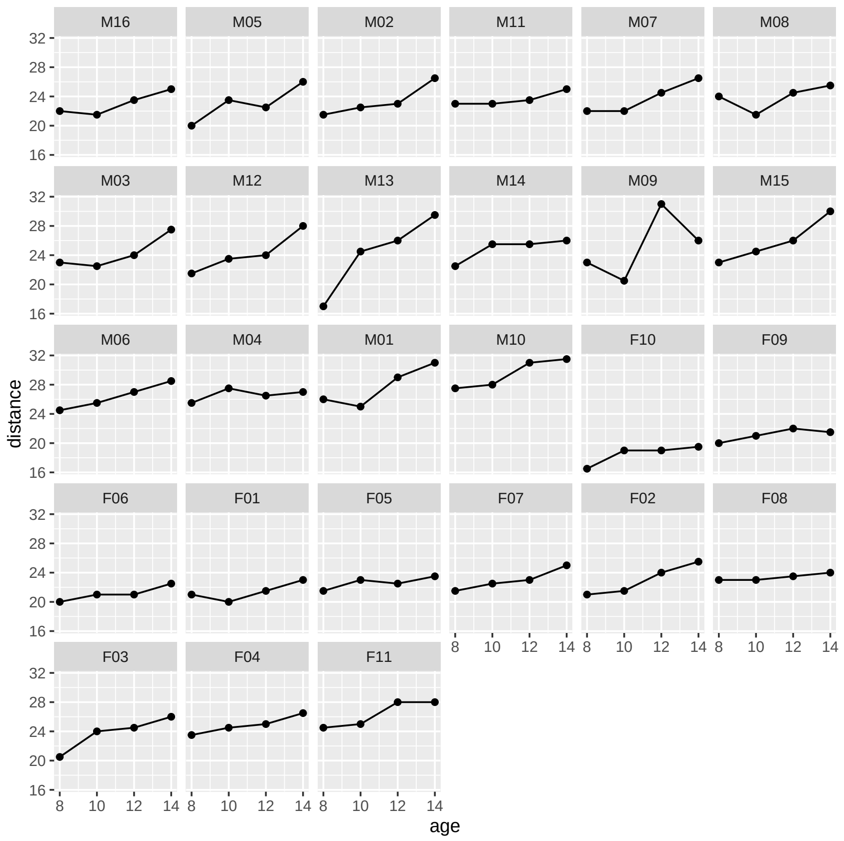
```
ggplot(data = Orthodont,
       mapping = aes(y = distance,
                     x = age)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(. ~ Subject)
```

```
`geom_smooth()` using formula 'y ~ x'
```



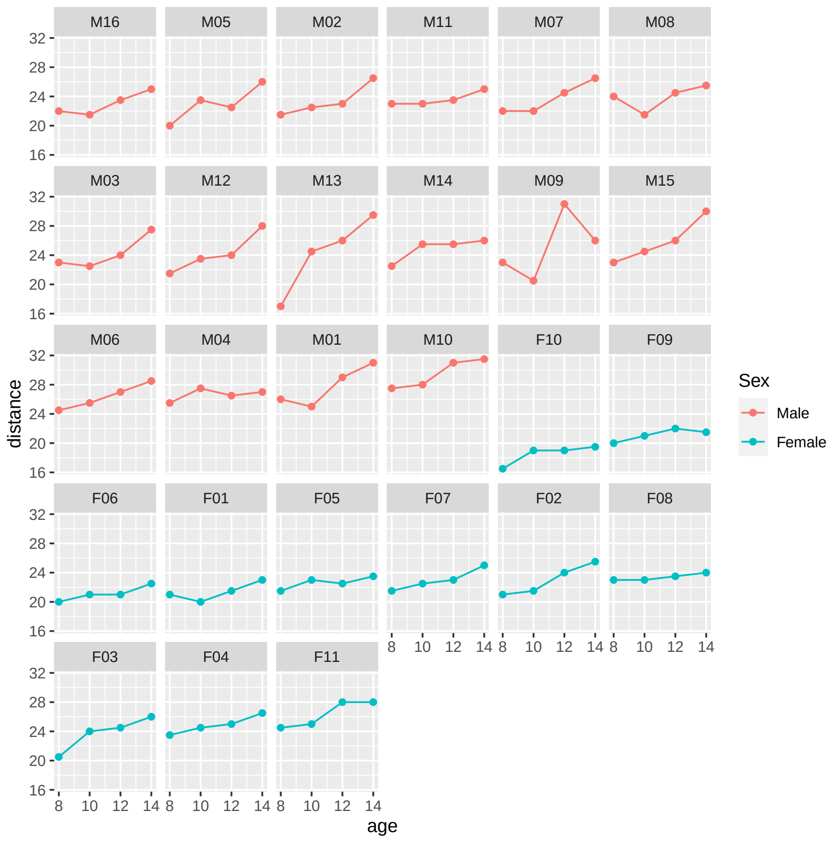
2.

```
ggplot(data = Orthodont,
       mapping = aes(y = distance,
                     x = age)) +
  geom_point() +
  geom_line() +
  facet_wrap(. ~ Subject)
```



3.

```
ggplot(data = Orthodont,
       mapping = aes(y = distance,
                     x = age,
                     colour = Sex)) +
  geom_point() +
  geom_line() +
  facet_wrap(. ~ Subject)
```



3.4 Question:

NB: before starting to work on this exercise, read the whole assignment here. Indeed, this exercise involves some collaborative work.

Using any of the built-in datasets (type `data()` *for the whole list) produce a graph with the base* ***R*** *functions (i.e.* `plot()`*,* `boxplot()`*,...). Then save it as a jpeg file.*

Using the same dataset produce a graph with either `xyplot()` *or* `ggplot()`*.*

Exchange the two graphs with another participant in the room and without looking at his/her code try to reproduce the exact same graph.

Finally, discuss with the other participant your solution. Again, you may want to involve in this discussion the course instructors.

R-Bootcamp: Series 3

Dr. Matteo Tanadini

February 2022

1 Exercise: Fitting models

1.1 Question:

The `cars` *built-in dataset has two columns: "speed" (of the car) and "distance" (needed to stop the travelling car). Perform the following tasks.*

1. use the `str()` function to check that both columns are indeed numeric variables and to see how many observations are present in the data
2. create a new dataframe named `cars.short` that contains only the first 36 observations present in the original dataset.
3. add a new column that contains the values 1 to 6 repeated 6 times (for a total length of 36). Name this column "test_day"
4. display the data (you can use "base" graphs or the add-on packages {*lattice*} / {*ggplot2*}) (i.e. create a scatter plot for "dist" against "speed" and a boxplot for "dist" against "test_day"
5. fit a linear model where "dist" is the response variable, "speed" and "test_day" the predictors
6. look at the summary of the model, how good is the fit? (hint: look at the adjusted-r-squared value)
7. fit a model that only contains "speed" as predictor. Then compare the two models. What is the p-value of the ANOVA test?
8. plot the model diagnostics in one single device (hint: you may want to set the parameter "mfrow" via the `par()` function).
9. the very first plot produced by the command you just ran displays the residuals against the fitted values. Reproduce a equivalent graph with the `ggplot()` function

Going further (*)

10. get the regression coefficient from the fitted model. Use the corresponding extractor function first. Then use `str()` or `names()` to find out in which slot of the lm object the coefficients are stored and extract them as you would do when access a slot in a list.
11. get the adjusted-r-squared and the regression coefficients and store them in a list. Pay attention to the choice of the list name (remember must comprehensible and consistent)
12. use the `gam()` function in {*mgcv*} package to fit a Generalise Additive Model on the same data. Look at the summary of the fitted model and compare the adjusted-r-squared with the one of the linear model.

1.2 Question:

The `airquality` *built-in dataset has several columns. Fit a linear model where you try to predict "Ozone" concentration from "Solar.R" and "Wind". Comment on the data and results.*

2 Exercise: Missing values

2.1 Question:

*Create a dataset in a spreadsheet (e.g. excel) that contains empty cells. Make sure your dataset has both numeric and categorical variables that contain empty cells. Read the data in **R** and comment on how where the empty cells coded.*

3 Exercise: Packages

3.1 Question:

1. how many packages are available on CRAN?
2. search on CRAN for the "MASS" package
   - is this a recommended package?
   - how many authors are there?
   - who is the maintainer? Is that a good reference?
   - is it true that we can find this package in the "econometric" task view on CRAN?
   - is there any companion book to this package?
3. go to the "Machine Learning" task view on CRAN and find what is the reference implementation of the "Random Forest" algorithm
4. go to CRAN and click on the "Contributed" section. There search for a tutorial on how to create **R** packages. Name a document that seems to be good

3.2 Question:

In which package can you find the following objects:

1. the `ls()` function
2. the `plot()` function
3. a function that contains the word "sunflower"
4. a dataset named "swiss"
5. the `apropos()` function

3.3 Question:

*Assume you realised that loading a package, say package "nlme", creates a conflict. Assumed that you don't really need this package, how can you unload it from your **R**-session?*

3.4 Question:

Assume you wrote a few functions yourself and want to write a package to store them. It that feasible? Could you then pass this package to someone else or are you forced to put it on CRAN?

R-Bootcamp: Series 3

Dr. Matteo Tanadini

February 2022

```
*** Solutions ***
```

1 Exercise: Fitting models

1.1 Question:

The `cars` built-in dataset has two columns: "speed" (of the car) and "distance" (needed to stop the travelling car). Perform the following tasks.

1. use the `str()` function to check that both columns are indeed numeric variables and to see how many observations are present in the data
2. create a new dataframe named `cars.short` that contains only the first 36 observations present in the original dataset.
3. add a new column that contains the values 1 to 6 repeated 6 times (for a total length of 36). Name this column "test_day"
4. display the data (you can use "base" graphs or the add-on packages {*lattice*} / {*ggplot2*}) (i.e. create a scatter plot for "dist" against "speed" and a boxplot for "dist" against "test_day"
5. fit a linear model where "dist" is the response variable, "speed" and "test_day" the predictors
6. look at the summary of the model, how good is the fit? (hint: look at the adjusted-r-squared value)
7. fit a model that only contains "speed" as predictor. Then compare the two models. What is the p-value of the ANOVA test?
8. plot the model diagnostics in one single device (hint: you may want to set the parameter "mfrow" via the `par()` function).
9. the very first plot produced by the command you just ran displays the residuals against the fitted values. Reproduce a equivalent graph with the `ggplot()` function

Going further (*)

10. get the regression coefficient from the fitted model. Use the corresponding extractor function first. Then use `str()` or `names()` to find out in which slot of the lm object the coefficients are stored and extract them as you would do when access a slot in a list.
11. get the adjusted-r-squared and the regression coefficients and store them in a list. Pay attention to the choice of the list name (remember must comprehensible and consistent)
12. use the `gam()` function in {*mgcv*} package to fit a Generalise Additive Model on the same data. Look at the summary of the fitted model and compare the adjusted-r-squared with the one of the linear model.

Answers

1.

```
?cars
```

```
str(cars)
```

```
'data.frame': 50 obs. of  2 variables:
 $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
 $ dist : num  2 10 4 22 16 10 18 26 34 17 ...
```

There are 50 observations and both columns are numerical.

2.

```
cars.short <- cars[1:36, ]
nrow(cars.short)
```

```
[1] 36
```

3.

```
cars.short$test_day <- rep(1:6, times = 6)
```

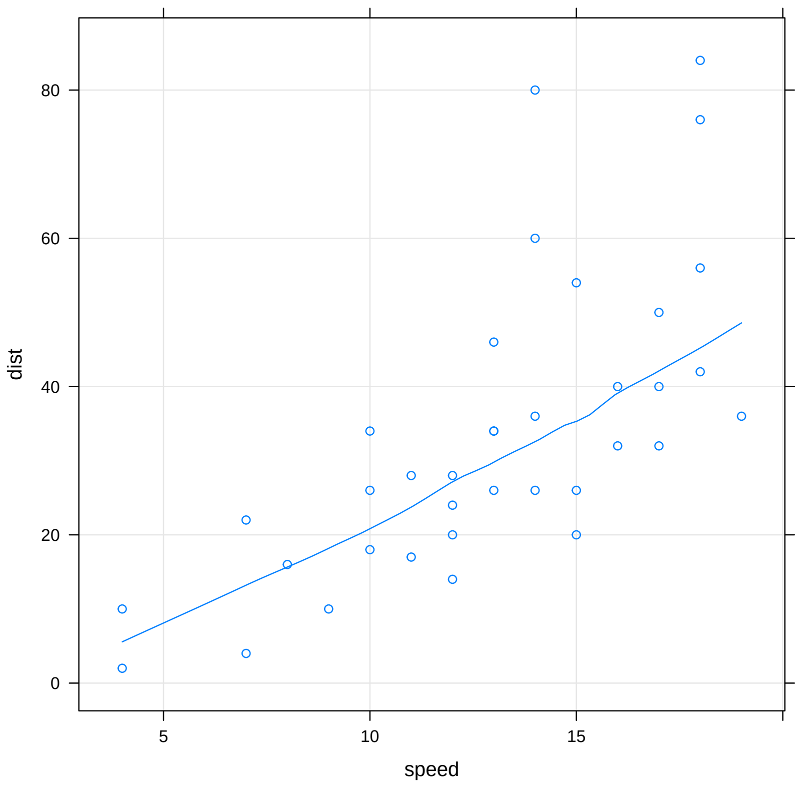
Actually, the argument "times" could be omitted and the `rep()` function would use "recycling to fill the column. Note also that there would be no warning message here, as the length of the final vector is a multiple of the shortest (i.e. 6 and 36).

```
cars.short$test_day_same <- rep(1:6)
head(cars.short, n = 13)
```

```
   speed dist test_day test_day_same
1      4    2        1             1
2      4   10        2             2
3      7    4        3             3
4      7   22        4             4
5      8   16        5             5
6      9   10        6             6
7     10   18        1             1
8     10   26        2             2
9     10   34        3             3
10    11   17        4             4
11    11   28        5             5
12    12   14        6             6
13    12   20        1             1
```

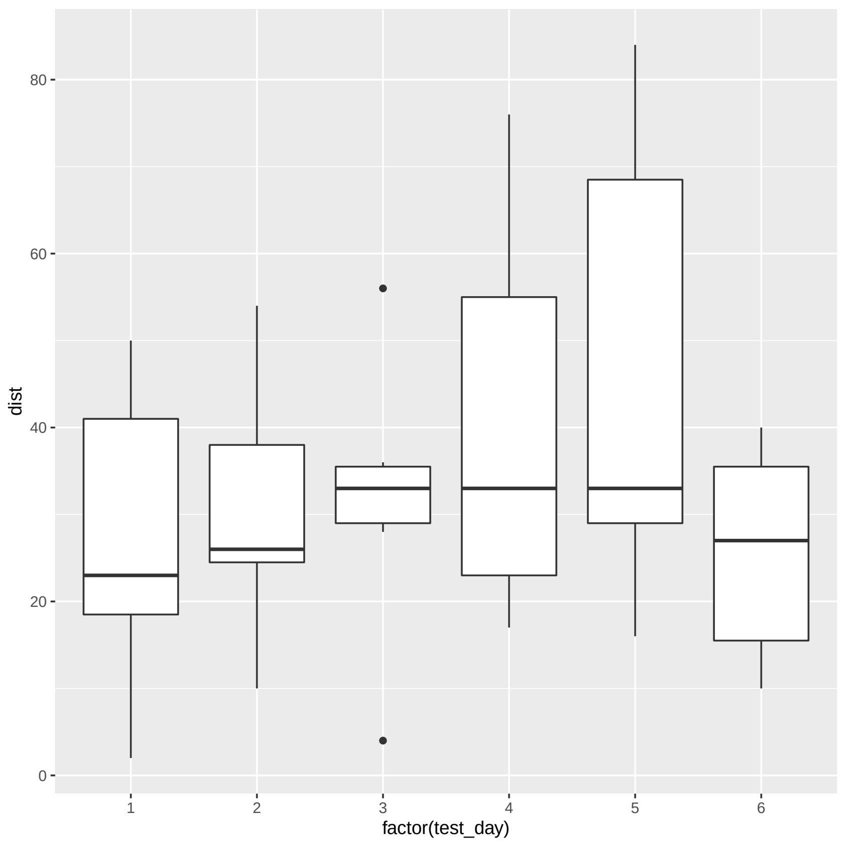
4.

```
library(lattice)
xyplot(dist ~ speed, data = cars.short,
       type = c("p", "smooth", "g"))
```



Here I decided to create a scatterplot with a smoother and a grid in the background.

```
library(ggplot2)
ggplot(data = cars.short,
       mapping = aes(y = dist,
                     x = factor(test_day))) +
  geom_boxplot()
```



Here I created a boxplot. Note that the function `ggplot()` requires factor to draw boxplots.

5.

```
lm.cars.short <- lm(dist ~ speed + test_day, data = cars.short)
```

6.

```
summary(lm.cars.short)


Call:
lm(formula = dist ~ speed + test_day, data = cars.short)

Residuals:
   Min     1Q Median     3Q    Max
-20.09  -9.47  -1.75   5.34  43.21

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -11.838      9.242   -1.28     0.21
speed          3.568      0.650    5.49  4.3e-06 ***
test_day      -0.264      1.462   -0.18     0.86
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 15 on 33 degrees of freedom
Multiple R-squared:  0.485,Adjusted R-squared:  0.454
F-statistic: 15.5 on 2 and 33 DF,  p-value: 1.76e-05
```

The adjusted-r-squared is about 0.45. Whether this is a good fit or not depends on the situation. Those of you that are familiar with the output of linear models, may have understood that day_test as been taken as a numeric variable here. We may want to consider this variable to be categorical here as there is no reason to believe that there is a linear trend in days. A simple solution to that would be to define a new varaible "test_day.factor" and refit the model.

```
cars.short$test_day.factor <- as.factor(cars.short$test_day)
str(cars.short)

'data.frame': 36 obs. of  5 variables:
 $ speed          : num  4 4 7 7 8 9 10 10 10 11 ...
 $ dist           : num  2 10 4 22 16 10 18 26 34 17 ...
 $ test_day       : int  1 2 3 4 5 6 1 2 3 4 ...
 $ test_day_same  : int  1 2 3 4 5 6 1 2 3 4 ...
 $ test_day.factor: Factor w/ 6 levels "1","2","3","4",..: 1 2 3 4 5 6 1 2 3 4 ...

##
lm.cars.short <- lm(dist ~ speed + test_day.factor, data = cars.short)
summary(lm.cars.short)


Call:
lm(formula = dist ~ speed + test_day.factor, data = cars.short)

Residuals:
   Min     1Q Median     3Q    Max
-26.26  -9.46  -1.54   6.57  32.53

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)       -15.570      9.012   -1.73    0.095 .
speed               3.597      0.601    5.99  1.7e-06 ***
test_day.factor2    2.134      7.834    0.27    0.787
test_day.factor3    1.069      7.855    0.14    0.893
test_day.factor4    8.370      7.873    1.06    0.296
test_day.factor5   12.671      7.896    1.60    0.119
test_day.factor6   -9.727      7.956   -1.22    0.231
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 14 on 29 degrees of freedom
Multiple R-squared:  0.613,Adjusted R-squared:  0.533
F-statistic: 7.66 on 6 and 29 DF,  p-value: 5.48e-05
```

Note that here we decided to overwrite the model because we won't need the old one any more as we realised it was not was we expected it to be. In other cases, we may want to keep the old model for future use and therefore overwriting is not a good option.

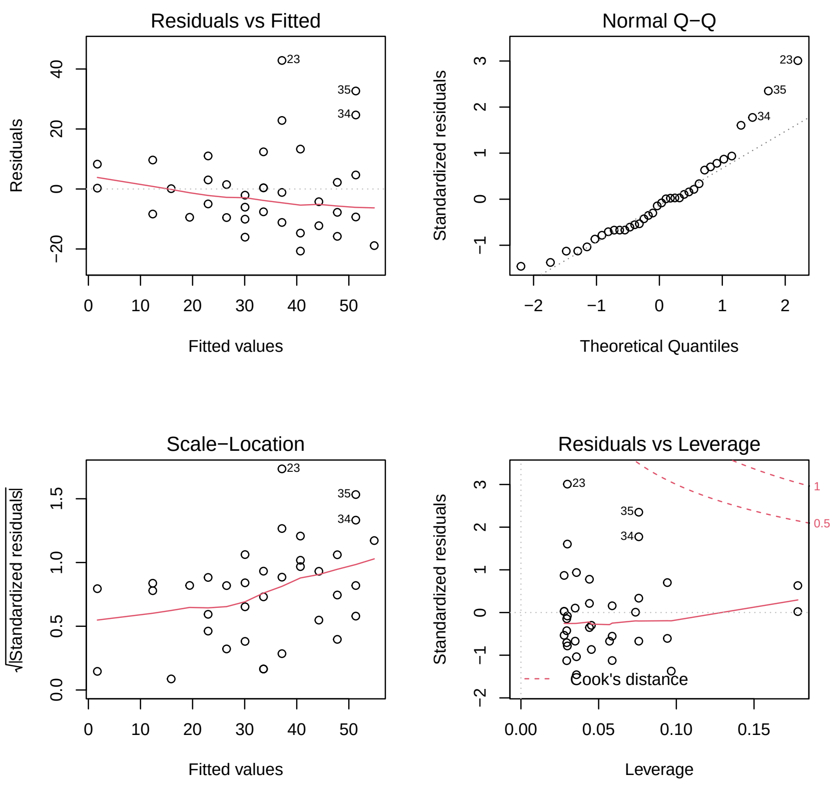
7.

```
lm.cars.short.speed.only <- update(lm.cars.short, . ~ . - test_day)
## same as
lm.cars.short <- lm(dist ~ speed, data = cars.short)
##
## compare the two models
anova(lm.cars.short.speed.only, lm.cars.short)

Analysis of Variance Table

Model 1: dist ~ speed + test_day.factor
Model 2: dist ~ speed
  Res.Df  RSS Df Sum of Sq    F Pr(>F)
1     29 5336
2     34 7110 -5     -1773 1.93   0.12
```

The p-value for this test is about 0.12.

8.

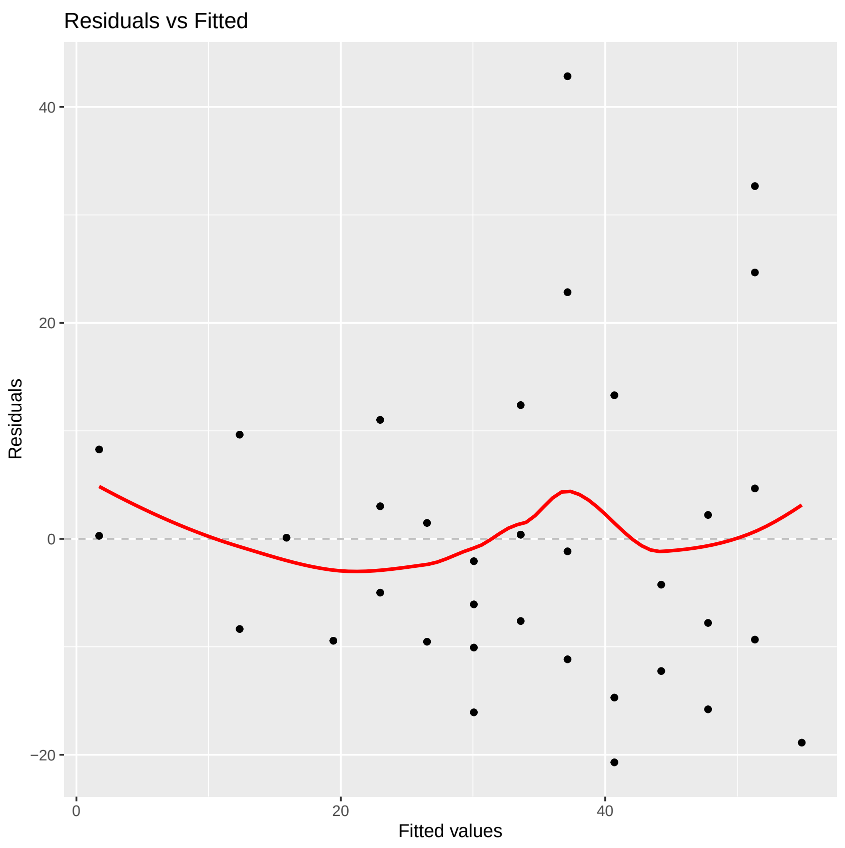
```
par(mfrow = c(2, 2))
plot(lm.cars.short)
```



9.

```
res.lm.cars.short <- resid(lm.cars.short)
fit.lm.cars.short <- fitted(lm.cars.short)
##
ggplot(mapping = aes(y = res.lm.cars.short,
                     x = fit.lm.cars.short)) +
  geom_hline(yintercept = 0, linetype = "dashed", col = "gray") +
  geom_point() +
  geom_smooth(se = FALSE, col = "red") +
  ylab("Residuals") +
  xlab("Fitted values") +
  ggtitle("Residuals vs Fitted")
```

```
`geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Note: here we named the residuals after the model.
Side note: You may have noticed that the smoother on the plot you created with the `ggplot()` function is more wiggly. The reason is simply that they use different methods.

Going further (*)

10. Let's get the regression coefficients with the extract function.

```
coef(lm.cars.short)

(Intercept)       speed
      -12.5         3.5
```

Let get them from the lm object.

```
names(lm.cars.short)
```

Ok, the first slot seems to contain the regression coefficients.

```
lm.cars.short$coefficients

(Intercept)       speed
      -12.5         3.5
```

11.

```
## where is the adjusted-r-squared?
names(lm.cars.short)

 [1] "coefficients" "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "xlevels"       "call"          "terms"         "model"

## mhmm, not there...
names(summary(lm.cars.short))

 [1] "call"          "terms"         "residuals"     "coefficients"
 [5] "aliased"       "sigma"         "df"            "r.squared"
 [9] "adj.r.squared" "fstatistic"    "cov.unscaled"

## ok, here it is
##
## put everything in a list
list.cars.short <- list("Regression coefficients" = coef(lm.cars.short),
                        "Adj-R-Squared" = summary(lm.cars.short)$adj.r.squared)
##
list.cars.short

$`Regression coefficients`
(Intercept)       speed
      -12.5         3.5

$`Adj-R-Squared`
[1] 0.47
```

12.

```
library(mgcv)

Loading required package:  nlme

This is mgcv 1.8-38.  For overview type 'help("mgcv-package")'.

gam.cars.short <- gam(dist ~ s(speed) + test_day.factor, data = cars.short)
summary(gam.cars.short)


Family: gaussian
Link function: identity

Formula:
dist ~ s(speed) + test_day.factor

Parametric coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)         31.00       5.58    5.56  5.4e-06 ***
test_day.factor2     2.13       7.83    0.27     0.79
test_day.factor3     1.07       7.85    0.14     0.89
test_day.factor4     8.37       7.87    1.06     0.30
test_day.factor5    12.67       7.90    1.60     0.12
test_day.factor6    -9.73       7.96   -1.22     0.23
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
         edf Ref.df    F p-value
s(speed)   1      1 35.9 1.9e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.533   Deviance explained = 61.3%
GCV = 228.42  Scale est. = 184.01    n = 36

##
list.cars.short[[2]]

[1] 0.47
```

The two adjusted-r-squared coefficients are the same. Indeed, the model fitted in this very case fully equivalent.

1.2 Question:

The `airquality` built-in dataset has several columns. Fit a linear model where you try to predict "Ozone" concentration from "Solar.R" and "Wind". Comment on the data and results.

Answers

Lets's give a look at the data

```
str(airquality)
```

```
'data.frame': 153 obs. of  6 variables:
 $ Ozone  : int  41 36 12 18 NA 28 23 19 8 NA ...
 $ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
 $ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
 $ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
 $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
 $ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
```

```
head(airquality)
```

```
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
```

There seems to be some values which are set to `NA`! Let's fit the model.

```
lm.airquality <- lm(Ozone ~ Solar.R + Wind, data = airquality)
summary(lm.airquality)
```

```

Call:
lm(formula = Ozone ~ Solar.R + Wind, data = airquality)

Residuals:
   Min     1Q Median     3Q    Max
-45.65 -18.16  -5.96  18.51  85.24

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  77.2460     9.0675    8.52  1.1e-13 ***
Solar.R       0.1004     0.0263    3.82  0.00022 ***
Wind         -5.4018     0.6732   -8.02  1.3e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 25 on 108 degrees of freedom
  (42 observations deleted due to missingness)
Multiple R-squared:  0.449,Adjusted R-squared:  0.439
F-statistic: 44.1 on 2 and 108 DF,  p-value: 1e-14
```

The sentence "(42 observations deleted due to missingness)" found in the summary output explains everything. These `NA` values are actually missing values. Indeed, NA stands for Not Available.

2 Exercise: Missing values

2.1 Question:

*Create a dataset in a spreadsheet (e.g. excel) that contains empty cells. Make sure your dataset has both numeric and categorical variables that contain empty cells. Read the data in **R** and comment on how where the empty cells coded.*

Answers

We get the dataset we used for the democode.

```
d.testNAs_2 <- read.csv("../../../DataSets/dataset_NAs.csv",
                        na.strings = -999,
                        header = TRUE)
str(d.testNAs_2)
```

```
'data.frame': 5 obs. of  5 variables:
 $ FirstName : chr  "Gianni" "Julia" "James " "Andrea" ...
 $ FamilyName: chr  "Pestalozzi" "Mueller" "Watts" "Pagani" ...
 $ Age       : int  20 23 43 19 NA
 $ Gender    : chr  "M" "F" "M" "" ...
 $ Salary    : num  4.5 NA 120 5.5 6.7
```

```
d.testNAs_2
```

```
  FirstName FamilyName Age Gender Salary
1    Gianni Pestalozzi  20      M    4.5
2     Julia    Mueller  23      F     NA
3    James       Watts  43      M  120.0
4    Andrea     Pagani  19           5.5
5    Judith Kartoffeln  NA      F    6.7
```

We note that `NA` are only created for numeric variables. For categorical variables (i.e. factors) empty cells are are not directly coded as missing values.

As a matter of fact, the variable `Gender` has three levels and no missing values.

```
levels(d.testNAs_2$Gender)
```

```
NULL
```

```
##
table(d.testNAs_2$Gender, useNA = "always")
```

```

        F    M <NA>
   1    2    2    0
```

3 Exercise: Packages

3.1 Question:

1. how many packages are available on CRAN?
2. search on CRAN for the "MASS" package
   - is this a recommended package?
   - how many authors are there?
   - who is the maintainer? Is that a good reference?
   - is it true that we can find this package in the "econometric" task view on CRAN?
   - is there any companion book to this package?
3. go to the "Machine Learning" task view on CRAN and find what is the reference implementation of the "Random Forest" algorithm
4. go to CRAN and click on the "Contributed" section. There search for a tutorial on how to create **R** packages. Name a document that seems to be good

Answers

1. how many packages are available on CRAN?
   About 15,000
2. search on CRAN for the "MASS" package

- is this a recommended package?
  Yes (see "priority" at the top of the package page)
- how many authors are there?
  6
- who is the maintainer? Is that a good reference?
  Brian Ripley. He is supposed to be a good reference as he is professor at a Dept. of Statistics (Uni Oxford). Btw: he is one the biggest names in the **R**-world (now retired).
- is it true that we can find this package in the "econometric" task view on CRAN?
  Yes
- is there any companion book to this package?
  YES, "Modern Applied Statistics with S" (4th edition, 2002). This used to be the reference book for **R**, is now outdated

3. go to the "Machine Learning" task view on CRAN and find what is the reference implementation of the "Random Forest" algorithm?
   The package $\{randomForest\}$
4. go to CRAN and click on the "Contributed" section. There search for a tutorial on how to create **R** packages. Name a document that seems to be good
   There you can find a document named "Creating R Packages: A Tutorial" by Friedrich Leisch (another big name in the **R**-world). The document is supposed to be good as it was written by one of the **R**-core members (i.e. Leisch).

   Note that in the "Contributed" section you can find very many introductory documents. They can be very helpful, however, keep in mind that everyone can write such a document and therefore the quality is again variable.

3.2 Question:

In which package can you find the following objects:

1. the `ls()` function
2. the `plot()` function
3. a function that contains the word "sunflower"
4. a dataset named "swiss"
5. the `apropos()` function

Answers

1. the `ls()` function
   `find("ls")` returns "package:base"
2. the `plot()` function
   "package:graphics"
3. a function that contains the word "sunflower"
   `apropos("sunflower")` returns "sunflowerplot". Don't ask me what this function is about. I just found cool that a function can have such a name :)
4. a dataset named "swiss" "package:datasets"
5. the `apropos()` function "package:utils"

3.3 Question:

*Assume you realised that loading a package, say package "nlme", creates a conflict. Assumed that you don't really need this package, how can you unload it from your **R**-session?*

Answers

You can unload a package by unticking its check box in the ”Packages” pane in Rstudio. Alternatively, you can use the `detach()` function directly. For example, `detach("package:nlme", unload = TRUE)`. NB: don’t forget to use the quotation marks around package:nlme.

If you were to prefer the Rstudio way to remove a package, make sure you will copy and paste the corresponding **R**-command in your script!

Finally, note that every time you restart `R` all package must be reloaded again. You can force some packages to load be default whenever starting a new session, **however, this is bad practice. Indeed, every new session must start ”clean” with no data or packages being loaded by default.**

3.4 Question:

Assume you wrote a few functions yourself and want to write a package to store them. It that feasible? Could you then pass this package to someone else or are you forced to put it on CRAN?

Answers

Absolutely, you can write a package with your own functions and datasets (note however, that this is usually done by (very) experienced people). Indeed, you can also write a script that contains your functions and data and load these with the `source()` function.

You can pass the package to anyone who wants to have it. You only must pay attention to the operating system. Packages come in two format: one that is suitable for mac and Linux users and one that is suitable to windows users.

Finally, there is no need to put your package on CRAN. CRAN is the official repository, but you could very well keep your package for yourself of put it on a another repository (e.g. github or sourceforge.net)