Data Management with R (session 1)

Solution of the exercises

2021-09-07

Exercise 1.1

- Create object **my_vec** which contains a vector of numeric values: 28, 29, 35, 75, 40, 52, 23, 25, 10, 50.

We use function *c()*

```
my_vec <- c(28, 29, 35, 75, 40, 52, 23, 25, 10, 50)
```

- Compute the mean, min, max and standard deviation of **my_vec** by using the functions *mean()*, *min()*, *max()*

```
mean(my_vec)
```

```
## [1] 36.7
```

```
min(my_vec)
```

```
## [1] 10
```

```
max(my_vec)
```

```
## [1] 75
```

- Compute the variance of **my_vec** by using only the function *sum()*, *mean()*, *length()* (which gives the size of a vector). We remind that the variance of $x_1, ..., x_n$ is $\frac{1}{n}\sum(x_i - \bar{x})^2$ where $\bar{x}$ is the mean.

We store the mean in an object so that we can use this object in the formula

```
my_mean <- mean(my_vec)
1 / length(my_vec) * sum((my_vec - my_mean) ^ 2)
```

```
## [1] 306.41
```

- Compare it with the result obtained by *var()* function

```
var(my_vec)
```

```
## [1] 340.4556
```

Remark: the function *var()* $n - 1$ at the denominator (which gives an unbiased estimator of the variance for i.i.d. observations).

- Create object **my_vec_st** which substracts the mean and divide by the standard deviation:

```
my_vec_st <- (my_vec - mean(my_vec)) / sd(my_vec)
```

- Print the working directory (WD) and save objects **my_vec** and **my_vec_st** in a file "exo1.RData"

```
getwd()
```

```
## [1] "/media/thibault/My Passport/course/R_advanced/chapter_1/slides"
```

```
save(my_vec, my_vec_st, file = "exo1.RData")
```

Exercise 1.2

- What is the difference between *library()* and *require()* ?

If we print the code of the function *require()* we notice that it actually uses the function *library()* but this last one is included in the *tryCatch()* function.

When we apply the function *library()* on a package which is not installed on the machine, it causes an error and if the instruction is included in a file that we source (with function *source()*), the codes which appear after *library()* will not be executed.

When we use *tryCatch()*, we know that there could be an error but it will be ignored. In other term, even if a package is not installed, there will be a warning message (which is different than an error message) and the codes after *require()* will not be interrupted.

Moreover, *require()* returns a logical equal to TRUE if the package is loaded and FALSE otherwise

```
(require("this_package_does_not_exist"))
```

```
## Le chargement a nécessité le package : this_package_does_not_exist
```

```
## Warning in library(package, lib.loc = lib.loc, character.only = TRUE,
## logical.return = TRUE, : aucun package nommé 'this_package_does_not_exist' n'est
## trouvé
```

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

```
a <- 5
print(a)
```

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

- Why these two syntaxes are working ?

```
require("stringr")
require(stringr)
```

Actually, if we look at the code of the function *require()*, we can see:

```
if (!character.only)
  package <- as.character(substitute(package))
```

In other terms, when we do not use the "", it does not consider **package** argument as an object which explains why the following syntax is not working

```
nom_package <- "stringr"
require(nom_package)
```

Indeed, **nom_package** is not evaluated; instead of the function *substitute()* permits to keep **nom_package** as a **name** object which is then converted into strings with *as.character()*.

- Use the operator :: to use the function *str_to_title()* included in the package **stringr** without calling *library()* or *require()*.

It is not necessary to load a package for using one particular function inside it. Indeed, when we load a package, it imports lots of functions and new classes of object which is not necessaraly desired. That is why we can do:

```
stringr::str_to_title("data management with r (session 1)")
```

```
## [1] "Data Management With R (Session 1)"
```

Exercise 2.1

- What's happening here:

```
c(21, 180, "F", "DU", "FR", TRUE)
```

```
## [1] "21"   "180"  "F"    "DU"   "FR"   "TRUE"
```

Character dominates numeric/logical. All the elements of the vectors are converted in strings.

- What's happening here:

```
TRUE | this_object_does_not_exist
TRUE || this_object_does_not_exist
```

The first line gives an error message whereas it is not the case for the second line. Indeed || is a shortcut: the first verification is **TRUE** which means that we know that the result will be **TRUE** whatever the result of the second verification.

- What's happening here:

```
c(1, 1, 1, 1) ^ c(0, 1) + c(0, 1, 2)
```

The size of the vectors are all different. The bigger vector is the first one, so the other vectors are repeated until their sizes equal 4.

```
c(1, 1, 1, 1) ^ c(0, 1, 0, 1) + c(0, 1, 2, 0)
```

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

- **R** includes a lot of base functions which can be seen here. Choose 5 of them, describe and illustrate them.
  - *Sys.time()* returns the current date with time of the machine

    ```
    Sys.time()
    ```

    ```
    ## [1] "2021-09-07 10:53:48 CEST"
    ```

  - *is.na()* returns a vector of logical for identifying the elements with NA (Non Available) values

    ```
    x <- c(NA, 1, 2, 3)
    is.na(x)
    ```

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

  - **for** is a reserved word (like **if**, **else**, **repeat**, **while**, **function**, **for**, **in**, **next**, **break**, **TRUE**, **FALSE**, **NULL**, **Inf**, **NaN**, **NA**, **NA_integer_**, **NA_real_** **NA_complex_**, **NA_character_**, which means that it can not be used as a name object. It is used for doing a **for** loop

    ```
    for (i in 1:3)
      print(i)
    ```

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

  - *ceiling()* takes a single numeric argument x and returns a numeric vector containing the smallest integers not less than the corresponding elements of x.

```
ceiling(0.0000000000001)
```

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

- *floor()* takes a single numeric argument x and returns a numeric vector containing the largest integers not greater than the corresponding elements of x.

```
floor(0.999999999999)
```

```
## [1] 0
```

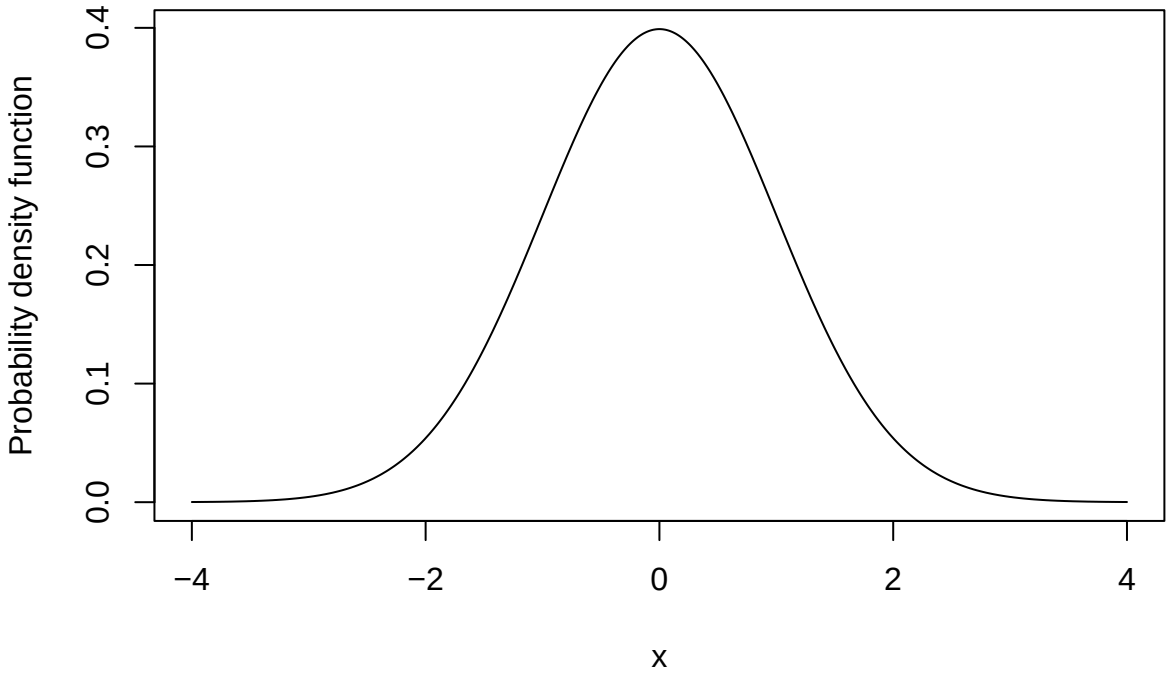
- *round()* rounds the values in its first argument to the specified number of decimal places (default 0). See 'Details' about "round to even" when rounding off a 5.

```
round(c(0.4999999999, 0.50000000001))
```

```
## [1] 0 1
```

- Plot the function $f(x) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}x^2)$ for $x \in [-4, 4]$

```
x <- seq(from = -4, to = 4, by = 0.01)
plot(x, 1 /sqrt(2 * pi) * exp(-0.5 * x ^ 2), type = "l",
     ylab = "Probability density function")
```



- By using the function *sample()* draw a random sample of size 100 of a Bernoulli distribution with $p = 0.5$.

A bernoulli distribution has two possible values : 0 and 1. This is the first argument of the function *sample()*. We draw 100 values randomly (which is equivalent to choose $p = 0.5$) and use argument **replace = TRUE**.

- What are the differences between *sort()*, *order()* and *rank()*:

```
age <- c(25, 28, 30, NA, 21, 26, 29, 31, NA, 22, 27)
sort(age)
```

```
## [1] 21 22 25 26 27 28 29 30 31
```

```
rank(age)
```

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

```
order(age)
```

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

- *sort()* re-arrange the vector **age** by ordering the elements
- *rank()* gives the rank of each elements of **age** (ex-aquo are permitted)
- *order()* gives the indices of the elements ranked

Exercise 2.2

Let consider the vector of strings **my_word**:

```
my_word <- c("we went 2 times to warwick",
             "moi 1 fois 1 w-e")
```

- give the character position of "we" in **my_word**

```
gregexpr(pattern = "we", my_word)
```

```
## [[1]]
## [1] 1 4
## attr(,"match.length")
## [1] 2 2
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
##
## [[2]]
## [1] -1
## attr(,"match.length")
## [1] -1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
```

- give the character position of "w" or "e" in **my_word**

```
gregexpr(pattern = "[we]", my_word)
```

```
## [[1]]
## [1]  1  2  4  5 14 20 23
## attr(,"match.length")
## [1] 1 1 1 1 1 1 1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
##
## [[2]]
## [1] 14 16
## attr(,"match.length")
## [1] 1 1
```

```
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
```

- give the character position of "we" in **my_word**, knowing that there is a empty space before

```
gregexpr(pattern = " we", "we went to warwick")
```

```
## [[1]]
## [1] 3
## attr(,"match.length")
## [1] 3
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
```

- give the character position of any numbers in **my_word**

```
gregexpr(pattern = "[0-9]", my_word)
```

```
## [[1]]
## [1] 9
## attr(,"match.length")
## [1] 1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
##
## [[2]]
## [1]  5 12
## attr(,"match.length")
## [1] 1 1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
```

- count the number of times any numbers is appearing in **my_word**

```
stringr::str_count(my_word, "[0-9]")
```

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

Exercise 2.3

We consider the two vectors **weight** and **group**

```
ctl <- c(4.17, 5.58, 5.18, 6.11, 4.50, 4.61, 5.17, 4.53, 5.33, 5.14)
trt <- c(4.81, 4.17, 4.41, 3.59, 5.87, 3.83, 6.03, 4.89, 4.32, 4.69)
group <- gl(2, 10, 20, labels = c("Ctl", "Trt"))
weight <- c(ctl, trt)
```

- create a matrix $\mathbf{X}$ of dim 20×2 which contains in the first column 1 if group="Ctl", 0 otherwise, and contains in the second column 1 if group="Trt" and 0 otherwise.

```
X <- matrix(0, 20, 2)
X[, 1] <- group == "Ctl"
X[, 2] <- group == "Trt"
```

- What does the following command do?

```
split(weight, group)
```

```
## $Ctl
##  [1] 4.17 5.58 5.18 6.11 4.50 4.61 5.17 4.53 5.33 5.14
##
## $Trt
##  [1] 4.81 4.17 4.41 3.59 5.87 3.83 6.03 4.89 4.32 4.69
```

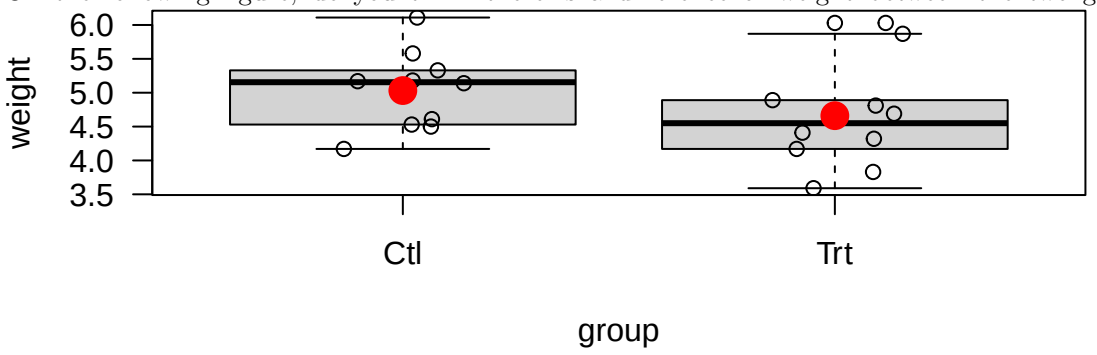
It splits the vector into two vectors with regards to the variable **group**

- Compute the mean of **weight** in the two groups "Ctl" and "Trt"

```
lapply(split(weight, group), mean)
```

```
## $Ctl
## [1] 5.032
##
## $Trt
## [1] 4.661
```

- On the following figure, do you think there is a difference of weight between the two groups ?



It seems that the "control" group has higher values but this not obvious.

- create matrix $A = (X'X)$ and $b = (X'y)$ where y is the **weight**
- solve the equation $A\beta = b$. We call **hat_beta** the solution of the equation

```
hat_beta <- solve(A, b)
```

- compute the adjusted values $\hat{y} = X\hat{\beta}$. We call **hat_y** this vector.

```
hat_y <- X %*% hat_beta
```

- compute the vector of residuals $\hat{\epsilon} = y - \hat{y}$. We call **hat_e** this vector

```
hat_e <- (weight - hat_y)
```

- Compute the sum of squares of the residuals. We call **sse** this value:

```
sse <- sum(hat_e ^ 2)
```

- compute the residual standard error which is equal to $\sqrt{SSE/(n-2)}$

```
sqrt(sse / 18)
```

```
## [1] 0.6963895
```

- compute **SST** which is equal to $\sum(y - \bar{y})^2$

```
sst <- sum((weight - mean(weight)) ^ 2)
```

- compute **SSR** which is equal to $\sum(\hat{y} - \bar{y})^2$

```
ssr <- sum((hat_y - mean(weight)) ^ 2)
```

- verify that $SST = SSR + SSE$

```
sst == (ssr + sse)
```

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

- compute the R^2 which is equal to SSR/SST

```
ssr/sst
```

```
## [1] 0.0730776
```

Exercise 2.4

- Execute the solution of exercice 3 and create a **list** object called **res_lm** which contains the residuals, the SSE value and the R^2.
- Create a **data.frame** called **pred_y** which contains the fitted values, the residuals and the Y variable.

Exercise 3.1

- Import one data set from these different web pages by using the method of your choice:
  - link 1

```
covid_data <- readr::read_csv2("https://www.data.gouv.fr/fr/datasets/r/63352e38-d353-4b54-bfd1-f1b3ee1c
```

```
## i Using '\',\'' as decimal and '\'.\'' as grouping mark. Use `read_delim()` for more control.
```

```
## Warning: Unnamed `col_types` should have the same length as `col_names`. Using
## smaller of the two.
```

```
## Warning: 164090 parsing failures.
## row col  expected     actual
##   1  -- 7 columns 10 columns 'https://www.data.gouv.fr/fr/datasets/r/63352e38-d353-4b54-bfd1-f1b3ee1
##   2  -- 7 columns 10 columns 'https://www.data.gouv.fr/fr/datasets/r/63352e38-d353-4b54-bfd1-f1b3ee1
##   3  -- 7 columns 10 columns 'https://www.data.gouv.fr/fr/datasets/r/63352e38-d353-4b54-bfd1-f1b3ee1
##   4  -- 7 columns 10 columns 'https://www.data.gouv.fr/fr/datasets/r/63352e38-d353-4b54-bfd1-f1b3ee1
##   5  -- 7 columns 10 columns 'https://www.data.gouv.fr/fr/datasets/r/63352e38-d353-4b54-bfd1-f1b3ee1
## ... ... ......... .......... ..............................................................................
## See problems(...) for more details.
```

- link 2 (import if possible a ".xls" file)

```
download.file("https://www.data.gouv.fr/fr/datasets/r/94525672-4ec3-4699-a56b-607dfabb1b3c", destfile =
covid_data <- readxl::read_xlsx("covid.xlsx")
```

- link 3

```
download.file("https://www.data.gouv.fr/fr/datasets/r/021a7ffb-855f-498b-8ddc-7d9f299ba823", destfile =
maire_data <- readxl::read_xlsx("maire.xlsx", col_types = rep("text", 11))
```

Exercise 4.1

- In the data **admnrev** from package **wooldridge**, transform the data from the long to the wide form with respect to the variable year. We call **admnrev_wide** this object

```
library(wooldridge)
admnrev_wide <- tidyr::pivot_wider(admnrev, id_cols = "state",
                   names_from = 2,
                   values_from = c(3, 4, 5))
admnrev_wide
```

```
## # A tibble: 51 x 10
##    state admnrev_85 admnrev_90 admnrev_95 daysfrst_85 daysfrst_90 daysfrst_95
##    <chr>      <int>      <int>      <int>       <int>       <int>       <int>
##  1 AL             0          0          0           0           0           0
##  2 AK             1          1          1          30          30          30
##  3 AZ             0          1          1           0          30          30
##  4 AR             0          0          0           0           0           0
##  5 CA             0          1          1           0         120          30
##  6 CO             1          1          1         365          90          90
##  7 CT             0          1          1           0          90           0
##  8 DE             1          1          1          90          90          90
##  9 DC             1          1          1           0           0           0
## 10 FL             0          1          1           0          30           0
## # ... with 41 more rows, and 3 more variables: daysscnd_85 <int>,
## #   daysscnd_90 <int>, daysscnd_95 <int>
```

- Transform the object **admnrev_wide** from wide to long object.

```
tidyr::pivot_longer(admnrev_wide,
                    cols = 2:10,
                    names_to = c(".value", "year"),
                    names_pattern = "(.*)_(.*)"
             )
```

```
## # A tibble: 153 x 5
##    state year  admnrev daysfrst daysscnd
##    <chr> <chr>   <int>    <int>    <int>
##  1 AL    85          0        0        0
##  2 AL    90          0        0        0
##  3 AL    95          0        0        0
##  4 AK    85          1       30      365
##  5 AK    90          1       30      365
##  6 AK    95          1       30      365
##  7 AZ    85          0        0        0
##  8 AZ    90          1       30       90
##  9 AZ    95          1       30       90
## 10 AR    85          0        0        0
## # ... with 143 more rows
```