

Exercises – Statistics and Experimental Design 2026-27

0 Introductory concepts in Statistics and the R programme

For this practical session, you can use R, but we recommend using RStudio, which is an intuitive and convenient interface for R.

To install:

<https://posit.co/download/rstudio-desktop/>

follow steps 1 (Install R) and 2 (Install RStudio).

When using RStudio, you will see four panes:

- **Source:** is where you create and edit R scripts. R scripts are simply text files with the extension “.R”. When you open RStudio, a new untitled script (“Untitled1”) is automatically created. Before you start writing, you should always save the file with a new name (for example, “AulasEDE.R”). This way, if something goes wrong with your computer while you are working, your R code will be available when you reopen RStudio.

You will notice that, when you write code in the Source pane, R does not execute it automatically. To execute the code, you need to send it to the Console. There are several ways to do this. The most time-consuming is to copy and paste. A more efficient option is to select the piece of code you want to run and click Run in the top-right corner of the Source pane. Alternatively, you can use the shortcuts Command + Return (Mac) or Control + Enter (Windows).

- **Console:** is the core of R. This is where the code is actually executed. At the beginning of the console you will see the symbol >, which indicates that R is ready to receive new commands. You can write code directly in the console and obtain an immediate response, or write it in Source and send the code to the console.

Although it is possible to execute code directly in the console, you should use Source. This is because code entered in the console is not saved (although it can be viewed in the history). Moreover, if you make a mistake, you will have to type everything again. Therefore, it is preferable to write the code in Source and run it when necessary.

- **Environment:** this pane displays all objects (such as vectors, matrices and data frames) defined in the current session. It also displays information such as the number of observations. It also includes useful options, such as “Import Dataset”, which opens a graphical interface for importing data into R.
- **Files / Plots / Packages / Help:**
 - **Files:** allows you to access files on your disk. You can also set the working directory by navigating to the desired folder, clicking on “More” and then on “Set As Working Directory”.
 - **Plots:** displays all generated plots.
 - **Packages:** shows all packages installed in R and indicates which ones are currently loaded. Loaded packages are marked.
 - **Help:** provides help on R functions. You can search for a function by name in the search bar.

There are several ways to import a data set into R:

- From the **Environment**, pane, use the “Import Dataset”. You can view the imported data and specify the type of the variables (numeric or text).
- Using the functions `read.table()` or `read.csv()` in R. You can use these functions even if you have an Excel file: you can first export/save the original file in `.csv` or `.txt` and then import it using `read.table()` (for `.txt` files) or `read.csv()` (for `.csv` files). The main arguments of these functions are:
 - **file**: the file path relative to the working directory, unless otherwise specified. For example, `file = "brix.txt"` looks for the file directly in the working directory.
 - **header**: logical value indicating whether the data have a header row, i.e. whether the first row contains the column names.
 - **sep**: character indicating how the columns are separated. For comma-separated files, use `sep = ","`; for tab-separated files, use `sep = "\t"`.
 - **dec**: character string indicating which symbol is used as the decimal separator. In most cases, use `dec = "."` or `dec = ","`.
 - **stringsAsFactors**: logical value indicating whether character strings should be converted to factors.

1. A farmer installed a rain gauge to measure rainfall on a given plot of land. Over one year, the following monthly totals (in mm) were obtained:

January	101.0	May	26.7	September	5.7
February	60.7	June	10.5	October	51.7
March	75.1	July	2.5	November	50.1
April	19.9	August	39.8	December	170.6

In an R working session, answer the following questions:

- (a) Create a vector containing the 12 monthly totals given above. Name the resulting object `precip`.

```
> precip <- c(101.0, 60.7, 75.1, 19.9, 26.7, 10.5, 2.5, 39.8, 5.7, 51.7, 50.1, 170.6)
```

The result can be displayed by typing the name of the object created.

- (b) Create the vector `meses` containing the names of the 12 months of the year.

```
meses <- c("Jan", "Fev", "Mar", "Abr", "Mai", "Jun", "Jul", "Ago", "Set", "Out", "Nov", "Dez")
```

- (c) Associate each measurement with the name of the corresponding month, using the R command `names`.

```
names(precip) <- meses
```

Result:

```
> precip
  Jan  Fev  Mar  Abr  Mai  Jun  Jul  Ago  Set  Out  Nov  Dez
101.0 60.7 75.1 19.9 26.7 10.5 2.5 39.8 5.7 51.7 50.1 170.6
```

- (d) Using the basic statistical commands available in R, calculate the following quantities:

- i. Total annual rainfall;

```
> sum(precip)
[1] 614.3
```

- ii. Mean monthly rainfall;

```
> mean(precip)
[1] 51.19167
```

iii. Median monthly rainfall;

```
> median(precip)
[1] 44.95
```

iv. Variance of the monthly rainfall values;

```
> var(precip)
[1] 2291.604
```

v. Standard deviation of the monthly rainfall values;

```
> sd(precip)
[1] 47.87071
```

or

```
> sqrt(var(precip))
[1] 47.87071
```

vi. Minimum monthly rainfall;

```
> min(precip)
[1] 2.5
```

vii. Maximum monthly rainfall;

```
> max(precip)
[1] 170.6
```

- (e) Apply the `summary` command to the `precip` object and inspect the result.

```
> summary(precip)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.50   17.55   44.95   51.19   64.30   170.60
```

- (f) Select the subvector of rainfall values for June to September (inclusive).

```
> precip[6:9]
Jun  Jul  Ago  Set
10.5  2.5 39.8  5.7
```

alternatively:

```
> precip[c(6,7,8,9)]
```

- (g) Apply the `plot` command to the vector `precip` that you created in question 1a. Comment on the result.

```
> plot(precip)
```

- (h) Run the following commands and comment on the result.

```
> plot(precip, type="l")
> plot(precip, type="h")
```

2. R provides several data sets. Their names and brief descriptions can be viewed using the command

```
> data()
```

Among these data sets is the vector `sunspots`, which records the average number of sunspots observed on each day of each month, between 1749 and 1983¹. The values can be viewed by simply typing the name of the object:

```
> sunspots
```

¹In fact, `sunspots` is not a vector, but an object of another type, called `ts` (the initials of *time series*, or seja, série cronológica). However, for what is required in this exercise the object behaves like a vector.

- (a) Determine the length of the vector `sunspots`, using the command `length`.

```
> length(sunspots)
[1] 2820
```

- (b) Create a histogram of the recorded values, using the command `hist`:

- i. letting the command define the value classes used; `hist(sunspots)`
- ii. requesting classes of width 10, starting at zero and ending at 260. `hist(sunspots, breaks=(0:26)*10)`

- (c) Using the command `quantile`:

- i. the three quartiles (first quartile, median and third quartile) of the data;

```
> quantile(sunspots)
 0%      25%      50%      75%     100%
0.000 15.700 42.000 74.925 253.800
```

- ii. the ninth decile of the data.

```
> quantile(sunspots, 0.9)
90%
112
```

- (d) Apply the `summary` command to the object `sunspots` and inspect the result.

```
> summary(sunspots)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.00   15.70   42.00   51.27   74.93  253.80
```

- (e) Construct a box-and-whisker plot of the data, using the `boxplot`.

```
> boxplot(sunspots)
```

3. A feeding experiment with rabbits used four different diets (labelled A, B, C and D) and five different treatments (numbered 1 to 5). After a certain period of time, the mean weights of the rabbits subjected to each diet-treatment combination were measured. The results obtained were:

peso	dieta	tratamento
1.5	A	1
1.4	A	2
1.4	A	3
1.2	A	4
1.4	A	5
2.7	B	1
2.9	B	2
2.1	B	3
3.0	B	4
3.3	B	5
2.1	C	1
2.2	C	2
2.4	C	3
2.0	C	4
2.5	C	5

1.3	D	1
1.0	D	2
1.1	D	3
1.3	D	4
1.5	D	5

Note that the first column is numeric, whereas the remaining columns are categorical variables (*factors*), although in one of them the labels of the different categories (*factor levels*) are numbers.

- (a) Create vectors corresponding to each column (use the names given above for these vectors). The vector `peso` should be numeric, whereas `dieta` and `tratamento` should be defined as factors (using the `factor` command).

```
> peso <- c(1.5, 1.4, 1.4, 1.2, 1.4, 2.7, 2.9, 2.1, 3.0, 3.3, 2.1, 2.2, 2.4,
+          2.0, 2.5, 1.3, 1.0, 1.1, 1.3, 1.5)
> peso
[1] 1.5 1.4 1.4 1.2 1.4 2.7 2.9 2.1 3.0 3.3 2.1 2.2 2.4 2.0 2.5 1.3 1.0 1.1 1.3
[20] 1.5
```

To create `dieta` and `tratamento`, it is necessary to use the `factor` command. Since there are many repetitions, the command `rep`, which generates repeated values, will be used. The first argument of the command `rep` is always the vector of values to be repeated. The second argument, named `times`, can be a numeric vector of the same length, indicating how many times the corresponding value of the original vector should be repeated. This is what happens when creating the factor `dieta`, where each letter is specified to be repeated 5 times:

```
> dieta <- factor(rep( c("A","B","C","D") , times=c(5,5,5,5) ))
> dieta
[1] A A A A A B B B B B C C C C C D D D D D
Levels: A B C D
```

When the second argument of the `rep` command is a single integer, the entire vector is repeated that number of times. This is what happens when creating the vector `tratamento`, where the integer vector from 1 to 5 (generated by `1:5`) is repeated four times.

```
> tratamento <- factor(rep( 1:5 , times=4 ))
> tratamento
[1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
Levels: 1 2 3 4 5
```

- (b) The most common way of storing data in R is in objects of class *data frame*, created using the `data.frame`. Using the three vectors created in the previous question, create a *data frame* named `coelhos`.

```
> coelhos <- data.frame(peso, dieta, tratamento)
> coelhos
  peso dieta tratamento
1  1.5    A           1
2  1.4    A           2
3  1.4    A           3
4  1.2    A           4
5  1.4    A           5
6  2.7    B           1
7  2.9    B           2
8  2.1    B           3
9  3.0    B           4
10 3.3    B           5
11 2.1    C           1
12 2.2    C           2
```

13	2.4	C	3
14	2.0	C	4
15	2.5	C	5
16	1.3	D	1
17	1.0	D	2
18	1.1	D	3
19	1.3	D	4
20	1.5	D	5

- (c) Apply the `summary` to the *data frame* `coelhos`. Comment on the results, particularly on how the command handles columns of different types.

```
> summary(coelhos)
      peso      dieta tratamento
Min.   :1.000  A:5    1:4
1st Qu.:1.375  B:5    2:4
Median :1.750  C:5    3:4
Mean   :1.915  D:5    4:4
3rd Qu.:2.425      5:4
Max.   :3.300
```

As can be seen, in the numeric column `peso` the command `summary` produces the usual summary statistics (minimum, first quartile, median, mean, third quartile and maximum). For factors, the command `summary` lists the different categories (factor levels) and, to the right of the symbol ‘:’, the number of times each level is repeated.

- (d) Select only the column corresponding to the weights and calculate its mean.

Since the columns of the *data frame* `coelhos` have assigned names, the easiest way to select a column is to type the name of the *data frame* followed by a dollar sign (\$), and then the name of the column you want. Thus, to extract only the column `peso` and calculate its mean, type:

```
> coelhos$peso
[1] 1.5 1.4 1.4 1.2 1.4 2.7 2.9 2.1 3.0 3.3 2.1 2.2 2.4 2.0 2.5 1.3 1.0 1.1 1.3
[20] 1.5
> mean(coelhos$peso)
[1] 1.915
```

Alternatively, any subset of rows and/or columns of the *data frame* can be selected by specifying, in square brackets after the object name, the desired row and/or column numbers (separated by a comma). For example, if we want to select the first three rows of columns 1 e 3, we can write:

```
> coelhos[c(1,2,3),c(1,3)]
      peso tratamento
1  1.5          1
2  1.4          2
3  1.4          3
```

To obtain the *totalidade* of the first column, simply leave the row index blank (before the comma) and write only the column number after the comma:

```
> coelhos[,1]
[1] 1.5 1.4 1.4 1.2 1.4 2.7 2.9 2.1 3.0 3.3 2.1 2.2 2.4 2.0 2.5 1.3 1.0 1.1 1.3 1.5
> mean(coelhos[,1])
[1] 1.915
```

- (e) Select only the rows corresponding to diet C.

Following what was seen in the previous question, and knowing that diet C corresponds to rows 11 a 15, the request in the question can be fulfilled by writing:

```
> coelhos[11:15,]
  peso dieta tratamento
11 2.1    C           1
12 2.2    C           2
13 2.4    C           3
14 2.0    C           4
15 2.5    C           5
```

However, R provides the powerful possibility of selecting elements of a vector or *data frame* by specifying a condition. In our example, the selection condition is that the column `dieta` has the value C. It is possible (as seen in the first exercise) to create a vector of logical values (true/false) corresponding to the condition:

```
> coelhos$dieta=="C"
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE  TRUE
[13] TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
```

This vector of logical values can be used to indicate which rows of the *data frame* we want to select (only those corresponding to the logical value TRUE):

```
> coelhos[coelhos$dieta=="C" , ]
  peso dieta tratamento
11 2.1    C           1
12 2.2    C           2
13 2.4    C           3
14 2.0    C           4
15 2.5    C           5
```

- (f) Using the `apply`, calculate the maximum value in each column. Comment.

The command `apply` allows the same function to be applied to all rows or all columns of a *data frame*. The name of the *data frame* is the first argument of the command `apply`. The second argument specifies the dimension to which the function is to be applied: 1 indicates rows and 2 indicates columns. The third argument is the name of the function to be applied. Thus, to obtain the maximum value of each column, the following command can be used:

```
> apply(coelhos, 2, max)
  peso      dieta tratamento
"3.3"      "D"          "5"
```

Note how the function `max` acts both on numeric values and on factor levels (in the latter case using the alphabetical order of the factor-level names).

4. In a study on raspberries carried out in the *Horticulture Section* of ISA, fruits from 14 different plants were analysed with respect to 6 different variables. The observed variables were: (i) fruit *diameter* (`Diametro` in *cm*); (ii) its *height* (`Altura` in *cm*); (iii) its *weight* (`Peso` in *g*); (iv) its soluble solids content, *brix* (`Brix` in degrees Brix); (v) its *pH*; (vi) its *sugar* content, excluding sucrose (`Acucar` in *g/100ml*). The data are in the file `brix.txt`, which is available on the course website (Support Materials section, Data subsection). The file should be downloaded and saved in the directory where the R working session is located, preferably the folder where you have been saving your work.

- (a) Read the data into R.

```
brix<-read.table("brix.txt", header=T)
```

- (b) Construct the scatter plots corresponding to every possible pair of variables. Comment on the result obtained.

```
plot(brix)
```

The "scatterplot matrix" produced by this command contains the scatter plots for every possible pair among the six variables in the data set. The scatter plots show that there are no particularly evident linear relationships. Another evident feature of the plots is the relatively porcas observações.

- (c) Obtain the variance-covariance matrix of the data. Interpret the result obtained.

```
> var(brix)
```

	Diametro	Altura	Peso	Brix	pH	Acucar
Diametro	0.011868132	0.0073626374	0.01184615	0.016703297	0.0060879121	0.068318681
Altura	0.007362637	0.0191758242	0.02926923	-0.009395604	0.0009120879	0.004142857
Peso	0.011846154	0.0292692308	0.12948846	-0.019576923	0.0150615385	0.054146154
Brix	0.016703297	-0.0093956044	-0.01957692	0.075659341	0.0190439560	0.250351648
pH	0.006087912	0.0009120879	0.01506154	0.019043956	0.0185054945	0.061236264
Acucar	0.068318681	0.0041428571	0.05414615	0.250351648	0.0612362637	1.627074725

A variance-covariance matrix is a symmetric matrix (i.e. each element in position ij is equal to the element in position ji for every i and j). The main diagonal contains the variances of the observed variables, while the off-diagonal elements are the covariances for the corresponding pairs of variables.

- (d) Obtain the correlation matrix for all pairs of variables. Interpret the result obtained.

```
> cor(brix)
```

	Diametro	Altura	Peso	Brix	pH	Acucar
Diametro	1.0000000	0.48805100	0.3021834	0.5574164	0.41079660	0.49163698
Altura	0.4880510	1.00000000	0.5873795	-0.2466701	0.04841828	0.02345412
Peso	0.3021834	0.58737946	1.0000000	-0.1977869	0.30768297	0.11796368
Brix	0.5574164	-0.24667005	-0.1977869	1.0000000	0.50895050	0.71353524
pH	0.4107966	0.04841828	0.3076830	0.5089505	1.00000000	0.35290238
Acucar	0.4916370	0.02345412	0.1179637	0.7135352	0.35290238	1.00000000

A correlation matrix is a symmetric matrix. The elements on the main diagonal are all 1 (corresponding to the correlation of a variable with itself), and the off-diagonal elements are the correlations for the corresponding pairs of variables.