Module 1. Database cross reference and functions

Functions

References

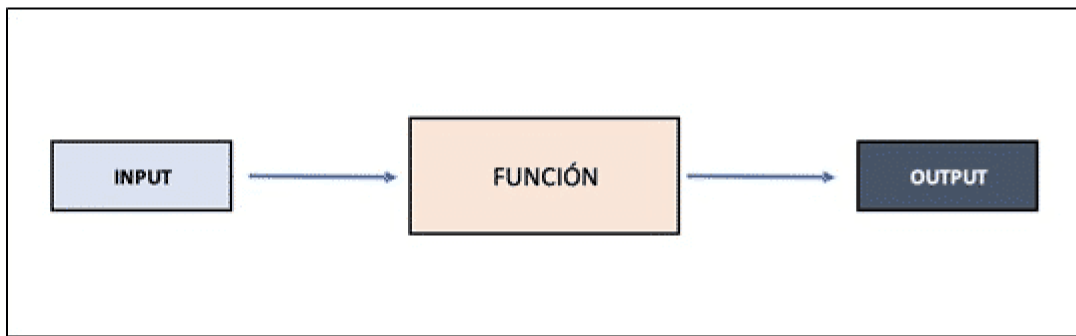
Functions

The concept of function has already been described, and there are multiple examples in the audiovisual material but this RStudio functionality will be developed even further, as it has great potential for data processing and it allows great user flexibility.

- **Functions**: these are elements that perform tasks in R. Exactly like in Excel, we use the name of the function, for example, SUM for sums, we add the values we want to add and it gives us a result after executing the code.

This definition is part of the first module of the first course, and it describes in simple terms the purpose of the function. Schematically, this can be represented as below:

Figure 1. Function definition



Source: prepared by the author

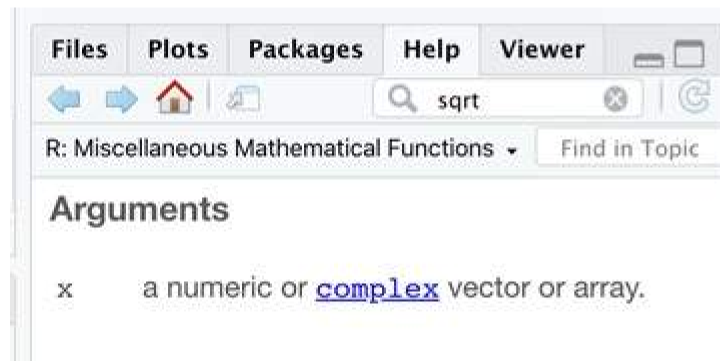
Input refers to values (they can be columns of data, unique values, lists, etc.) that will be indicated in the functions and *output* refers to the final result.

A more technical definition of a *function would* be that it is a code that uses inputs or arguments to perform calculations and returns an *output or result*.

Let's take a closer look at the features of functions.

As an example, we will use the sqrt() function. This function serves to calculate the square root of the numerical value indicated.

Figure 2. sqrt() function



Source: RStudio screenshot (Posit, 2011).

We can see how this function only needs an argument or *input* to return this result. The input may be a numerical value or structures that contain multiple values.

To use the function, we have to write its name in the code and define the arguments we want to use in parentheses.

Figure 2. sqrt() function

```
```{r}
sqrt(16)
```

[1] 4
```

Source: RStudio screenshot (Posit, 2011)

In this sqrt() function example, we can see that the input is "16" and the function automatically returns the output "4".

There are two types of functions:

- Incorporated into RStudio. They are part of the functionalities RStudio offers and can be used keeping the structure indicated in the documentation of each of them.
- Created by the user. Users may create functions that help automate the calculations they will perform repeatedly.

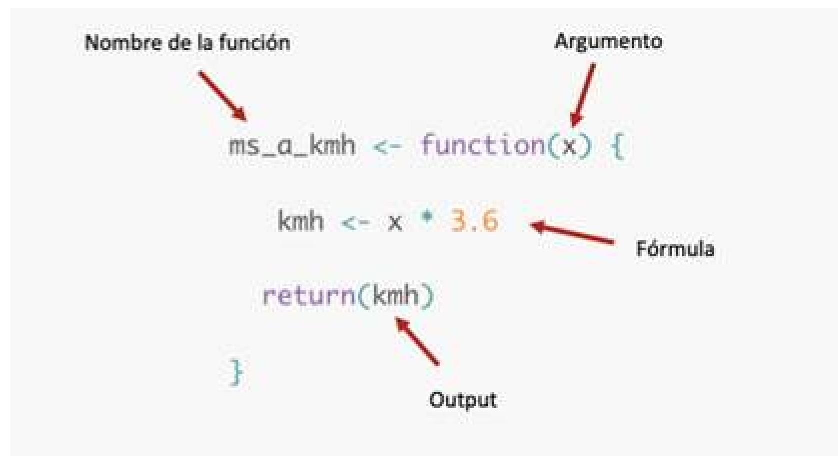
RStudio allows for the creation of functions following a suer-defined pattern; in this way, they can be used throughout the code to make calculations that we think are necessary and that are not within RStudio original functionalities.

For example, we may want to transform the values from meters per second to kilometers per hour as it is very common to have values in different units depending on the software used. In addition, we may want to communicate this information by referring to speed in terms of kilometers per houras this may be more understandable for certain staff members.

To create functions we have to use the structure required by RStudio:

- Choose a name for the function
- Use the function(){} structure
- Specify the argument(s)
- Create the formula, using the defined arguments
- Specify the output, as the formula or code we use can have multiple parts.

Figure 3. Creating functions on RStudio



Source: author's adaptation based on an RStudio screenshot (Posit, 2011)

As can be seen in the image, we have decided to call the function "ms_to_kmh" and have specified that it will only use an "x" argument.

- This argument can be specified with any other value (it could be "x" or "ms", for example).
- Since the argument is defined in the parenthesis, it must be present in the formula. If instead of "x" we had called it "ms", we would have to replace "x" with "ms" in the formula.

Finally, we state that the output will be the result of the calculation that we have indicated in the formula.

From now on, to use this function we will simp ytype "ms_to_kmh()" in the script to perform the desired calculations.

Figure 4. Creating functions on RStudio. Example

````
```{r}
ms_a_kmh(8)
```
````

```
[1] 28.8
```

Source: RStudio screenshot (Posit, 2011)

In the example above, we convert 8m/s (input) to km/h (output), using the previously created function.

Advantages of using functions:

- to automate calculations, that is, to reuse code in a simple way.
- to reduce the chance of making mistakes while writing code.

In the videos we will see examples of creating variables with multiple arguments and more complex features.

Self-assessment: The correct option is highlighted in yellow

Functions in R:

- are elements that perform specific tasks. Just like in Excel, we use the name of the function, for example,

sum() to perform sums.

- they can't perform tasks like in Excel. For example, we can't use a function called SUM to perform sums. Instead, we must write the entire code manually to sum the values without the help of predefined functions.

Main characteristics of database cross-referencing

In the previous course, we highlighted some of the characteristics of the process of extracting, transforming and uploading data. This process consists of properly storing the data collected in a particular environment and somewhat prepared for analysis, so that they can be easily accessible.

When we talk about storage, we refer to databases. These databases are organized datasets typically stored in the cloud. They are organized through the use of tables that may or may not be related to each other. In the context of the sports scientist, these types of data storage formats to get access to the information on a given frequency (daily, weekly, etc.) and perform the relevant analyses.

These formats offer a large number of advantages that have an impact on work efficiency and access to a great volume of information:

- Accumulated historical data. A differentiating factor in sports organizations, having data from previous seasons is a great advantage to assess procedures and changes. A frequent issue is that when clubs change staff, they usually lose all the information collected by the previous coaching staff, since they do not have automated data storage systems.

- Adding new variables/data: The structure of databases grants flexibility to store and create new variables according to the needs of the organization.

- Data updating: If there are changes in data processing due to updates of the different systems used and the variables need to be recalculated, we can perform this operation on the data stored in the past.

- Changes tracking: Any changes made to the tables will be recorded, as well as previous versions. This functionality means that, if a mistake has been made during set-up, the data from the previous version can be recovered.

- Multiple access to information: Different users are able to safely and quickly use information at the same time.

However, in many environments it is impossible to have access to databases, but there are tools which have similar functionalities on a smaller scale. These are Excel workbooks or text files. To guarantee the best performance of these storage formats, we must adhere to the principles highlighted in the previous course for data tables structuring.

In data analytics and in the field of sports physical performance, it is fairly common to use more than one data table to perform analysis. It is very likely that a single data table will not allow us to have all the information we need. There is a chance of creating a table with all the data that we want to use on a daily basis, but this system may have long-term limitations. Below there is a common example in our field.

We want to have a data table that allows us to make analyses related to the athlete's physical workload in specific sessions or throughout the season(s). A common source of data for these types of analysis is the one provided by GPS. When we download this data from the specific software of the system we are using, the information provided is often not complete. We will get information such as:

- Date.

- Exercise/match
- Duration
- Default position
- Player name
- Locomotive variable 1 (e.g. distance in metres)
- Locomotive variable 2 (e.g. m/min)
- Variables 3,4,5...

This data is sufficient for certain types of analysis, as we saw in previous courses, but if we need complementary information, it will be necessary to add other variables. If we wanted to carry out a transversal study throughout the season in which we analyse the changes in conditional performance throughout the days, or a comparison according to the competition or the level of the opponent, to get that information, we would have to add it manually to the data table after each session. The same would happen if the players' positions varied throughout the days: even though in the software of the GPS devices the positions of each player are indicated, to get the appropriate information we would have to edit it manually after each match.

The list of examples is quite vast, depending on whether the athlete is injured or unavailable, the season, the match location, etc.

An alternative to make the process less expensive is to have multiple tables where to access the information and use a tool such as RStudio to join those tables when convenient - and in this way to be able to use them for the necessary analyses.

Self-assessment: The correct option is highlighted in yellow

When we talk about storage, we refer to:

- ○ databases
- ○ RAM memory
- ○ e-cards
- ○ programming languages

SUBMIT

This example is aimed at illustrating the need to join or cross multiple data tables in order for the analyses to be more time-efficient and so that there is a greater amount of information, all of which will enhance the value of our work.

RStudio allows you to cross-reference multiple data tables:

- by adding more variables to the original table
- by filtering observations of one table from the observations of another table (Grolemund & Wickham, 2017)

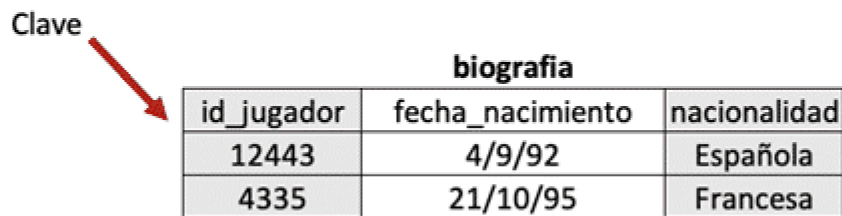
To understand how database cross-referencing works, the concept of "key” is essential.

The keys are variables within a table that identify a single observation. The keys will indicate that these observations are not repeated, that is, they are single occurrences.

In the image below we there is a simplified example of a table called "biography". There are 3 columns, The "key" will be the column

id_athlete, since it displays unique observations - a same athlete cannot have multiple dates of birth or nationalities.

Figure 5. "Key" example



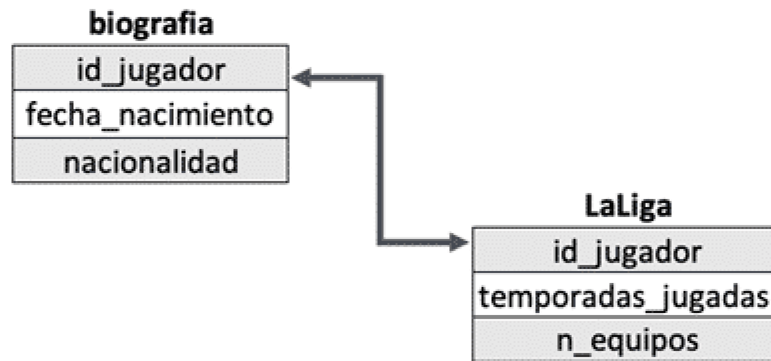
biografia

id_jugador	fecha_nacimiento	nacionalidad
12443	4/9/92	Española
4335	21/10/95	Francesa

Source: author's production based on an RStudio screenshot (Posit, 2011)

This key will allow the association/crossing with another table that has the same key and, in this way, they can be joined. In the example below we see it could be joined with a table called "LaLiga" that could have accumulated data of the different athletes who have participated in a competition.

Figure 6. Table merge Example



Source: author's production based on an RStudio screenshot (Posit, 2011)

However, in our field is very common to have tables with composite keys, due to the type of data we collect. These composite keys indicate that there are multiple variables/columns that are necessary to distinguish non-repeated observations (rows).

Below, for example, both variables "id_athlete" and "test_date" are necessary to identify unique observations, since players perform the tests multiple times, and if we were to select only the "id_athlete" column, we would get more than one observation.

Figure 7. Composite key Example

Source: author's production based on an RStudio screenshot (Posit, 2011)

To relate these data to another data table, it would also be necessary to relate it to composite keys from that table.

The following image is intended to exemplify multiple relationships between different keys, which have a variety of keys or composite keys that enable relationships to be made among them.

With these four tables we intend to show how we could get information on the line-ups during competition, the physical workload through GPS data and complementary information that may be of

interest for complementary analyses such as movement or distance covered and competition-relatetd content.

Figure 8. Multiple relationships between different keys

Source: author's production based on an RStudio screenshot (Posit, 2011)

Self-assessment: The correct option is highlighted in yellow

RStudio allows you to cross-reference multiple data tables:

- [ ] By adding more variables to the original table.
- [ ] By filtering observations from one table from observations in another table.
- [ ] By dividing the observations between different tables.
- [ ] RStudio does not allow cross-referencing multiple data tables without installing additional plugins.

SUBMIT

Types of data table joins

With RStudio we may use three main types of data table joins. Functions allow for these crossings to be carried out with great ease. Each type of function will achieve a different result, and therefore we need to determine the goal of the cross-referencing of data tables is before deciding which function to use.

The video material will be essential for the practical understanding of each of the functions. The descriptions shown below are for complementary reference.

For example, we could decide to join the following two tables (Atlassian, 2019):

Figure 9. Functions example

Source: author's production based on an RStudio screenshot (Posit, 2011)

- left_join(): This function allows you to keep all the observations of the first table (biography) and add the variables and observations of the values that match the second table (LaLiga).

- inner_join(): This function belongs to the type of joins that filter data, that is, it only keeps observations that share values (id_athlete) between tables.

- full_join(): This function includes all observations and variables in the two tables.

Figure 10. Functions example

left_join()

id_jugador	fecha_nacimiento	nacionalidad	temp_jugadas	n_equipos
12443	4/9/92	Española	5	3
4335	21/10/95	Francesa		

inner_join()

id_jugador	fecha_nacimiento	nacionalidad	temp_jugadas	n_equipos
12443	4/9/92	Española	5	3

full_join()

id_jugador	fecha_nacimiento	nacionalidad	temp_jugadas	n_equipos
12443	4/9/92	Española	5	3
4335	21/10/95	Francesa		
67899			8	1

Source: author’s production based on an RStudio screenshot (Posit, 2011)

This example is intended to highlight the need to decide which type of join we want to make in each particular case, since the results and the consequent analyses derived from these results will be conditioned by the way in which we have joined the data.

CONTINUE

References

Atlassian, (2019). SQL join types explained visually. https://www.atlassian.com/data/sql/sql-join-types-explained-visually

Grolemund, G., & Wickham, H. (2017). R for Data Science. O'Reilly Media.

Posit, (2011). RStudio [software para lenguaje de programación].

CONTINUE