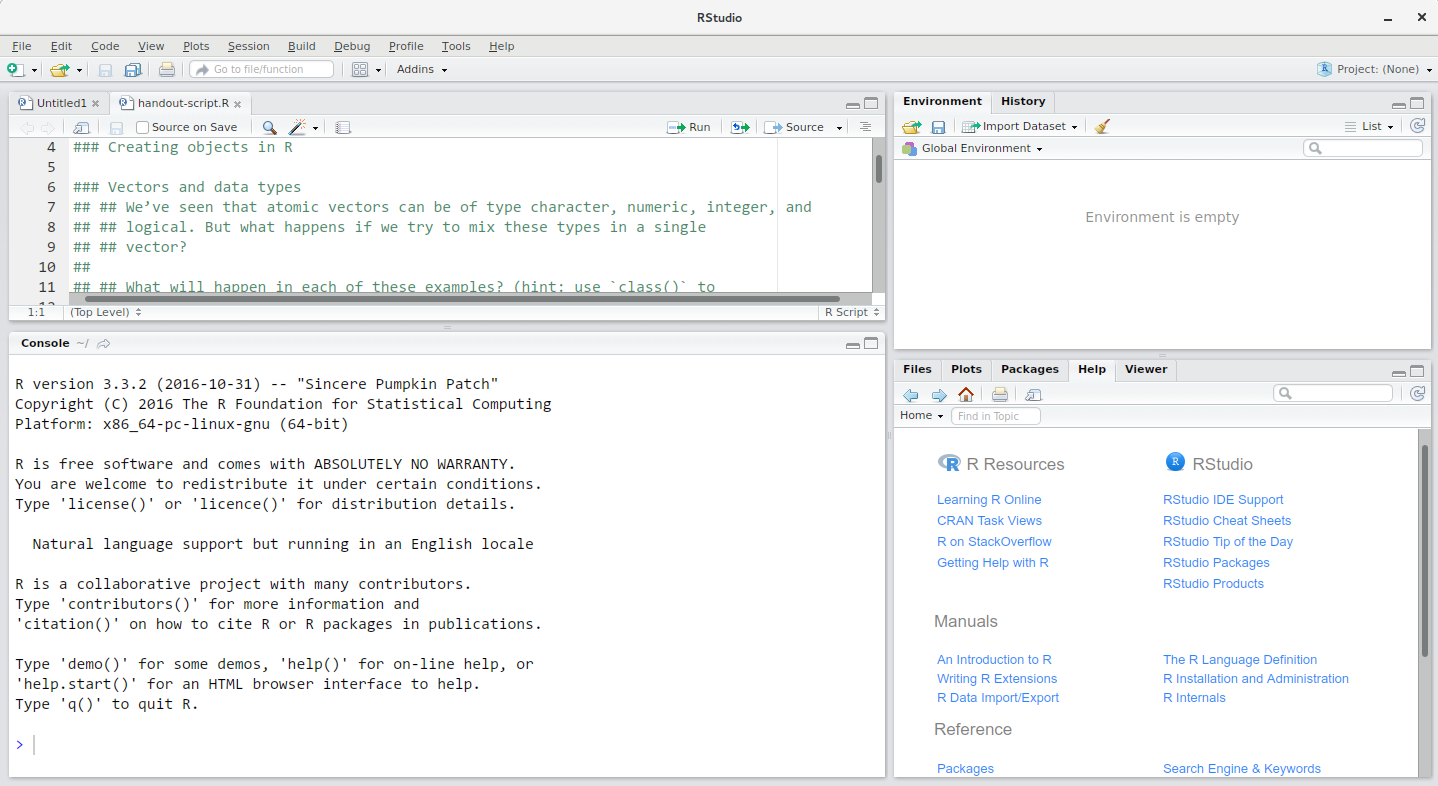
**Knowing your way around RStudio**



RStudio is divided into 4 “panes”:

* The **Source** for your scripts and documents (top-left, in the default layout)
* Your **Environment/History** (top-right) which shows all the objects in your working space (Environment) and your command history (History)
* Your **Files/Plots/Packages/Help/Viewer** (bottom-right)
* The R **Console** (bottom-left)

It is good practice to keep a set of related data, analyses, and text self-contained in a single folder, called the **working directory**. All of the scripts within this folder can then use *relative paths* to files that indicate where inside the project a file is located (as opposed to absolute paths, which point to where a file is on a specific computer). Working this way allows you to move your project around on your computer and share it with others without worrying about whether or not the underlying scripts will still work.

RStudio provides a helpful set of tools to do this through its “Projects” interface, which not only creates a working directory for you, but also remembers its location (allowing you to quickly navigate to it) and optionally preserves custom settings and (re-)open files to assist resume work after a break. Go through the steps for creating an “R Project” for this tutorial below.

A workspace is your current working environment in R which includes any user-defined object. By default, all of these objects will be saved, and automatically loaded, when you reopen your project.

**Interacting with R**

The basis of programming is that we write down instructions for the computer to follow, and then we tell the computer to follow those instructions. We write, or *code*, instructions in R because it is a common language that both the computer and we can understand. We call the instructions *commands* and we tell the computer to follow the instructions by *executing* (also called *running*) those commands.

There are two main ways of interacting with R: by using the console or by using script files (plain text files that contain your code). The console pane (in RStudio, the bottom left panel) is the place where commands written in the R language can be typed and executed immediately by the computer. It is also where the results will be shown for commands that have been executed. You can type commands directly into the console and press Enter to execute those commands, but they will be forgotten when you close the session.

Because we want our code and workflow to be reproducible, it is better to type the commands we want in the script editor, and save the script. This way, there is a complete record of what we did, and anyone (including our future selves!) can easily replicate the results on their computer.

RStudio allows you to execute commands directly from the script editor by using the Ctrl + Enter shortcut (on Macs, Cmd + Return will work, too). The command on the current line in the script (indicated by the cursor) or all of the commands in the currently selected text will be sent to the console and executed when you press Ctrl + Enter. You can find other keyboard shortcuts in this [RStudio cheatsheet about the RStudio IDE](https://raw.githubusercontent.com/rstudio/cheatsheets/main/rstudio-ide.pdf).

At some point in your analysis you may want to check the content of a variable or the structure of an object, without necessarily keeping a record of it in your script. You can type these commands and execute them directly in the console. RStudio provides the Ctrl + 1 and Ctrl + 2 shortcuts allow you to jump between the script and the console panes.

If R is ready to accept commands, the R console shows a > prompt. If it receives a command (by typing, copy-pasting or sent from the script editor using Ctrl + Enter), R will try to execute it, and when ready, will show the results and come back with a new > prompt to wait for new commands.

If R is still waiting for you to enter more data because it isn’t complete yet, the console will show a + prompt. It means that you haven’t finished entering a complete command. This is because you have not ‘closed’ a parenthesis or quotation, i.e. you don’t have the same number of left-parentheses as right-parentheses, or the same number of opening and closing quotation marks. When this happens, and you thought you finished typing your command, click inside the console window and press Esc; this will cancel the incomplete command and return you to the > prompt.

**Creating objects in R**

You can get output from R simply by typing math in the console:

3 + 5

12 / 7

However, to do useful and interesting things, we need to assign *values* to *objects*. To create an object, we need to give it a name followed by the assignment operator <-, and the value we want to give it:

weight\_kg <- 55

<- is the assignment operator. It assigns values on the right to objects on the left. So, after executing x <- 3, the value of x is 3. For historical reasons, you can also use = for assignments, but not in every context. Because of the [slight](https://blog.revolutionanalytics.com/2008/12/use-equals-or-arrow-for-assignment.html) [differences](https://renkun.me/2014/01/28/difference-between-assignment-operators-in-r/) in syntax, it is good practice to always use <- for assignments.

In RStudio, typing Alt + - (push Alt at the same time as the - key) will write <- in a single keystroke in a PC, while typing Option + - (push Option at the same time as the - key) does the same in a Mac.

Objects can be given almost any name such as x, current\_temperature, or subject\_id. Here are some further guidelines on naming objects:

* You want your object names to be explicit and not too long.
* They cannot start with a number (2x is not valid, but x2 is).
* R is case sensitive, so for example, weight\_kg is different from Weight\_kg.
* There are some names that cannot be used because they are the names of fundamental functions in R (e.g., if, else, for, see [here](https://stat.ethz.ch/R-manual/R-devel/library/base/html/Reserved.html) for a complete list). In general, even if it’s allowed, it’s best to not use other function names (e.g., c, T, mean, data, df, weights). If in doubt, check the help to see if the name is already in use.

**Objects**

When assigning a value to an object, R does not print anything. You can force R to print the value by using parentheses or by typing the object name:

weight\_kg <- 55 *# doesn't print anything*

(weight\_kg <- 55) *# but putting parenthesis around the call prints the value of `weight\_kg`*

weight\_kg *# and so does typing the name of the object*

Now that R has weight\_kg in memory, we can do arithmetic with it. For instance, we may want to convert this weight into pounds (weight in pounds is 2.2 times the weight in kg):

2.2 \* weight\_kg

We can also change an object’s value by assigning it a new one:

weight\_kg <- 57.5

2.2 \* weight\_kg

This means that assigning a value to one object does not change the values of other objects. For example, let’s store the animal’s weight in pounds in a new object, weight\_lb:

weight\_lb <- 2.2 \* weight\_kg

and then change weight\_kg to 100.

weight\_kg <- 100

**Functions and their arguments**

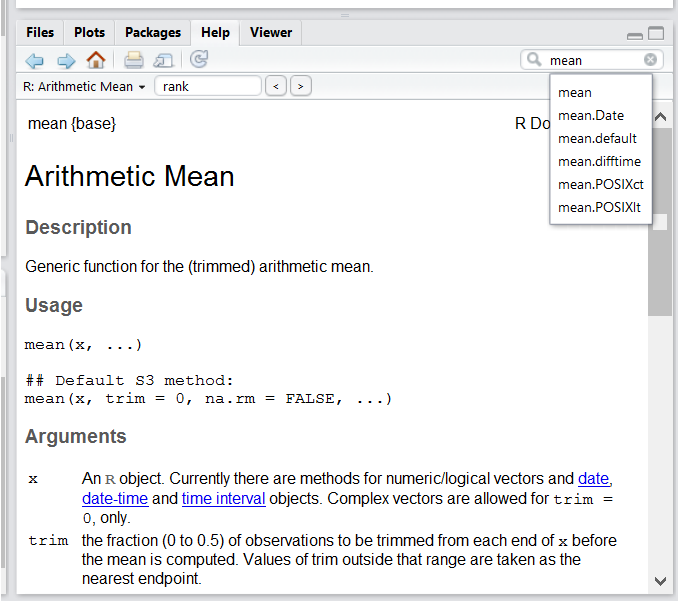
Functions are “canned scripts” that automate more complicated sets of commands including operations assignments, etc. Many functions are predefined, or can be made available by importing R *packages* (more on that later). A function usually takes one or more inputs called *arguments*. Functions often (but not always) return a *value*. A typical example would be the function sqrt(). The input (the argument) must be a number, and the return value (in fact, the output) is the square root of that number. Executing a function (‘running it’) is called *calling* the function. An example of a function call is:

weight\_kg <- **sqrt**(10)

Here, the value of 10 is given to the sqrt() function, the sqrt() function calculates the square root, and returns the value which is then assigned to the object weight\_kg. This function takes one argument, other functions might take several.

The return ‘value’ of a function need not be numerical (like that of sqrt()), and it also does not need to be a single item: it can be a set of things, or even a dataset. We’ll see that when we read data files into R.

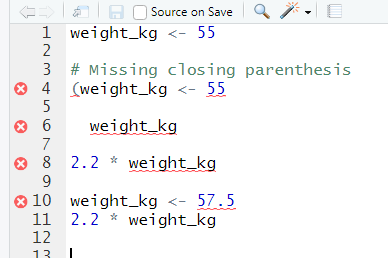
**Seeking help**

Searching function documentation with ? and ??

If you need help with a specific function, let’s say mean(), you can type ?mean or press F1 while your cursor is on the function name. If you are looking for a function to do a particular task, but don’t know the function name, you can use the double question mark ??, for example ??kruskall. Both commands will open matching help files in RStudio’s help panel in the lower right corner.

**Dealing with error messages**

Don’t get discouraged if your code doesn’t run immediately! Error messages are common when programming, and fixing errors is part of any programmer’s daily work. Often, the problem is a small typo in a variable name or a missing parenthesis. Watch for the red x’s next to your code in RStudio. These may provide helpful hints about the source of the problem.



RStudio shows a red x next to a line of code that R doesn’t understand.

If you can’t fix an error yourself, start by googling it. Some error messages are too generic to diagnose a problem (e.g. “subscript out of bounds”). In that case it might help to include the name of the function or package you’re using in your query