R Basics

# S.1.1 Downloading and installing R and RStudio

## S.1.1.1 How to download R

The process is really easy and will take you less than five minutes:

* Go to the [R homepage](http://www.r-project.org/).
* On the left-hand panel, just below the title ‘Download’, click on the word ‘CRAN’ to get a page of countries.
* Scroll down to the UK (or your current country) and click on one of the options, e.g. <https://www.stats.bris.ac.uk/R/>.
* Click on the version of R appropriate to your computer’s operating system (probably Mac or Windows).
* Click on the ‘base’ subdirectory.
* Click on the link to the R setup program (e.g. ‘Download R 4.0.5. for Windows’).
* When prompted, save the program to your computer’s hard drive.
* Open the folder, click on the ‘setup’ file, agree to everything, select ‘default installation’, and say ‘yes’ to a shortcut icon on your desktop.
* Click on the desktop icon (a big blue letter ‘R’ with a grey circle behind it) to familiarize yourself with what the base R console looks like, if you are interested.

## S.1.1.2 How to download RStudio

You will first need to have downloaded R (see section S.1.1.1). From there:

* Go to the [RStudio](https://www.rstudio.com/products/rstudio/download/) website.
* You will see the title ‘Choose your version’ and below that a series of options. Here, you only need to select the RStudio Desktop version with the ‘Open Source Licence’ by clicking on the ‘Download’ button below. Like the main program R, this version of RStudio is completely free.
* You will then be taken to a page with two steps—the first of which is to ‘Install R’, which you have already done. To follow the second step, you simply need to click on the big ‘Download RStudio’ button below, which will recommend the correct version for your system.
* When prompted, save the program to your computer’s hard drive.
* Agree to everything in the setup, select ‘default installation’, and say ‘yes’ to a shortcut icon on your desktop if it appears as an option. If the shortcut icon doesn’t appear as an option, you can search for ‘RStudio’ in your computer, and then right-click on its icon to pin it to your taskbar for easy access.
* Click on the desktop icon (a big white letter ‘R’ in a filled pale-blue circle) to start using RStudio.

# S.1.2 Operating RStudio and entering simple data

Once you have installed both R and RStudio, to fire up RStudio you just need to click on the icon on your desktop (or taskbar, if you have pinned it there), or open it from your program menu; the logo is a big white letter ‘R’ in a filled pale-blue circle. Note that you do not need to open the baseRprogram directly: RStudio will be fully functional so long as R is installed on your computer system.

When you first open RStudio, you will be met with three windows below your main task bar. On the left-hand side you should have one large window called the ‘Console’ and on the right-hand side there should be two smaller windows. In base R, you would only see one window, your ‘R Console’, set against a grey background, so make sure that it is RStudio you have opened. The Console is where we will input code to communicate directly with R. At the top of the Console, there is some text telling us about the version of R we are using, but below that you should find a blue ‘>’ cursor. The ‘>’ cursor tells you that R is ready and awaiting your instructions. The flashing ‘|’ is where we will type in commands. You can either type the following commands into R by hand, to get a feel for inputting the data yourself, or you can run them from the R script associated with this document that we also provide online (see section S.1.3.1 for details on opening R script files and running commands from them).

A very simple command we might want to get R to perform is a calculation. We can test this by typing in an easy sum at the flashing cursor. Try:

**27+64**

Now, if we hit ‘Enter’ on the keyboard, R will read your code (all commands in R are referred to as code), attempt to act on it, and print any required responses to your command as an output below your piece of code. No matter how complex your code, this is how Rfunctions every time it is given an instruction. Here, following our simple calculation command, R almost instantly presents us with the output:

[1] 91

While the answer 91 is correct, you will notice that R has prefaced its output with [1]. This [1] is simply a label R has assigned the value because it is the first (and in this case only) part of its output in response to our command. Beyond simple calculations, we can get R to do far more complicated work for us. We’ll next get R to keep track of multiple lists of data for us, which we might want to present or analyse.

Imagine we had information on the numbers of students who own different types of pets, and that this information was collected from students taking five different classes. We can enter the data for each pet type into R using commands like this one:

**dog <- c(7,9,5,6,2)**

Here, we have named our data set ‘dog’ (you can name data sets anything you like), and we then go on to list the values for how many students from each of the five classes have dogs. After the name, we type a *less than* sign (<) immediately followed by a *negative* sign (-) and then the list we want to assign to that name. You can think of these two symbols (<-) as making an arrow that points to the name that you want to assign to a list or another object in R—this is known as an ‘assignment character’, and it can also be entered with a simple keyboard shortcut (holding ‘Alt’ and pressing the ‘-‘ button). Next in the code, R likes you to give it a list as a series of numbers (or existing lists) separated by commas, with the whole list enclosed in round brackets, with the letter ‘c’ in front. The ‘c’ here stands for ‘concatenation’ or ‘bring together’, but we like to think of the ‘c’ as telling R to *connect* all the list of values. So, once we have typed in our command, we press ‘Enter/Return’. You will see the cursor again, telling you that R has completed the command and is ready for the next command. This time R does not respond by presenting an output, because we haven’t asked for R to do anything with that list yet, we have just told it the list to store for now.

Now we can add data on the other types of pets:

**cat <- c(5,3,6,9,7)**

**fish <- c(3,2,7,3,4)**

**other <- c(1,3,2,1,4)**

If we now ‘call’ one of these data sets by just typing its name and pressing ‘Enter’, R will tell us the values listed. Try it:

**dog**

[1] 7 9 5 6 2

**cat**

[1] 5 3 6 9 7

**fish**

[1] 3 2 7 3 4

**other**

[1] 1 3 2 1 4

You might have noticed that our lists of pets have now also appeared in the top right-hand window. This window shows us our ‘Global Environment’ and keeps track of the objects we create (i.e. information we have stored in R’s memory) and might want to use, such as our data set lists.

If we wanted to, we could now combine the data sets to one large data set for the numbers of pets of all types across all five classes. We do this in the same way we created the initial data sets, that is, by naming the new data and listing what we want it to contain:

**pet <- c(dog, cat, fish, other)**

‘pet’ now contains all the lists of pets in one long list:

**pet**

[1] 7 9 5 6 2 5 3 6 9 7 3 2 7 3 4 1 3 2 1 4

To find the total number of all pets across all five classes (assuming students don’t have more than one pet of a given type) we can then request:

**sum(pet)**

[1] 89

As you can see, it is pretty easy to input simple data sets like these intoR—and we do more of this manual data inputting in chapters 2 and 3. However, if we want to edit more complex code, it is easier to play around with it as a text or script file instead of typing directly into the Console each time. This is why we recommend using R scripts to store and edit your code, before running it in the Console (see section S.1.3). Also, you will more commonly use R to produce figures from larger data sets than the simple data we inputted manually here—we explain how to load existing data sets into R in section S.1.4.

# S.1.3 Saving R commands and graphics

## S.1.3.1 Using R scripts

In section S.1.2, we typed commands directly into the RStudio Console. While this is the most direct way to interact withR, most users of R instead use R scripts to write their code. Why would you want to do this? As you get more advanced in your use of R, sometimes you will want to perform a sequence of commands in R. Rather than having to type out each individual line and wait for R to work through it, we can write anR script, a set of commands that we can write and edit separately before running it through the Console together. More importantly, you can save your sequence of commands so that you can easily open that file in future and repeat your entire code exactly without lots of re-typing.

To open a new Rscript, click on the File menu on the top left of the RStudio taskbar and select ‘New File’. From here, select ‘R Script’ at the top of the choices that appear. You will see your R script window open on the left-hand side of your screen, occupying the top half of the screen that the Console previously spanned. The script window does not have a ‘>’ cursor because this is not where R will be running your code. However, it does have a flashing ‘|’ cursor of its own when you click in the window, as you can type out commands in here. And, in fact, the script window comes with a number of features that will make it easier to type in your code without making mistakes. For example, if you type out a piece of code involving brackets (such as our lists in section S.1.2), once you have typed the first open bracket, the R script window knows that you will need to match this with a closed bracket and automatically suggests that you add one to the end of your code. The R script window will also try to help you out when typing functions. For example, if we type ‘sum’ (as we did in section S.1.2) and pause, you will see that R suggests a range of functions beginning with ‘sum’. If you hover your mouse cursor over the different choices of function in the drop-down list, R offers you some information about the function and the option to tell you more if you wish. If we double-click on its first suggestion (the ‘sum’ we want), you’ll see that the brackets we’ll need to enclose our calculation have automatically appeared. While it might take you a little while to get used to these intelligent functions of the R script window, we think that they are ultimately really useful. Once you’ve gotten to grips with them, they can save you time and reduce mistakes in your coding—both valuable features when working on any project!

Another benefit of using an R script is that you can annotate your code for future reference, to remind yourself of what the different lines of commands mean. To do this, you just need to begin a line of annotation with the hashtag symbol ‘#’. When a line of text starts with a # and is read into R, R ignores the entire line. This means you can type a # and then write yourself a little note and then start a new line before typing your actual code. When you copy and paste both lines into R, only the code line will be read. You can also include a # at the end of a line of code, or after a particular argument or function as part of a larger command (see the ‘Glossary’ for what we mean by ‘argument’ and ‘function’), and the # will mean that any annotation added subsequently to that line will similarly be ignored when run in R. This can be useful when you want reminding of the role specific arguments/features play in an overall command. If you check out the accompanying R scripts for any of the chapters of this book (which we provide online as ESM), you will see that we have fully annotated them using the # symbol so that you can pick out what all the code does. The code included in chapters is not annotated, as we pick out key features in the accompanying text, but we strongly recommend you thoughtfully annotate code of your own as we do in our script files.

You can also include blank lines in scripts, whichR will also ignore. It also ignores spaces and tabs. These are useful in structuring the script, making it easier to follow by separating out commands that achieve different things. For example, the same code could be written out as one long command or written out over several separate lines (with annotations using the # symbol if desired); R will execute the commands identically (so long as the syntax is identical and correct in both formats). The automatic numbering of lines down the left-hand side of the script window is another useful feature of R scripts, particularly when you are discussing a script with someone else and need to refer to specific commands. An additional aesthetic benefit of using scripts in RStudio is the automatic colouring that RStudio’s editor highlights different components with; comments following hashtags are made green, numbers are made blue, and other text is kept black. This feature is especially helpful when looking over a script that is unfamiliar to you, enabling you to pick out what is going on more easily.

Once you have the commands in your script prepared, it is easy to run them in the Console. Simply highlight all the lines of the script you want to run and then click on the ‘Run’ icon at the top right of the script window. This will send your commands to the Console for R to implement, and you should see them appear here (in blue text), along with any outputs R produces in response to your instructions (in black text). You can run all of your code in one go, or one piece of code and then another after it, depending on whether you want to quickly reproduce a figure or output from code that you know functions perfectly or you need to take stock at the different steps. If there is a problem with your script, you will also see text in red appear in the Console, often stating an ‘Error’ R has encountered while trying to implement your script. The most common cause of such errors is a bit of code being mistyped, but you can easily refer back to your Rscript to amend whatever is causing the issue and run the script again once you have fixed this.

As with any piece of work, you should remember to regularly save your script while you are working on it, but you will definitely want to remember to save it before you close your session in RStudio. This will allow you to re-open your script at any point in the future without having to type out all your code again, so it is an absolute must! To save a new R script, you need to click on the little floppy disk ‘Save’ icon near the top left of the script window, and R will let you name and save the script as a file wherever you want on your computer. Once the name and location of the script have been saved once, if you subsequently click on the ‘Save’ icon again, R will update the saved file with the current version of your script. If you want to save a new version of your current script without replacing a previous version, you can click on your script, then go to the ‘File’ menu at the top left of the RStudio taskbar, and select ‘Save as’. This will allow you to give your current script a different name to the previous version and/or choose a different location for it to be saved. This may be particularly useful if you want to try out different variations of code, but without losing already-working versions of your instructions for R.

To load up an already-saved script, simply go to the ‘File’ menu at the top left of the RStudio taskbar, select ‘Open File’ and find the script file you want to open. Your saved commands will then appear again as a script window, ready to be run in the Console or edited.

# S.1.3.2. Producing and saving graphics in RStudio

When you run a line (or lines) of code that gets R to produce a figure, it should appear in the bottom-right window of RStudio under the ‘Plots’ tab. If the plot window is too small for your figure, you will get an error message in the RConsole and either your figure will not appear at all in the bottom-right window, or it will appear compressed and with components missing. If this happens, you simply need to adjust the size of the bottom-right window by hovering over the sides of the pane until a four-way cursor appears and you can click and drag to stretch the window out appropriately.

Once you have your figure as you want it, you can easily save it by clicking on the ‘Export’ button of the bar running below the ‘Plots’ tab of the bottom-right window. This will bring up a drop-down menu, where you can choose to save your figure as an image or a PDF, or copy it to the clipboard. If you copy it to your clipboard, you can then directly paste it into a Word document or other program by right-clicking and selecting ‘Paste’, but we recommend that you save a copy of any figure to your computer so that you have a version retained as a back-up. When saving a figure either as an image or a PDF, be sure to click the ‘Directory’ button near the top of the window that pops up so that you can select which folder it gets saved to—this will help you find it easily when needed! If you save your figure as an image, you can specify the image format at the top of the window that pops up (JPEG is good for most purposes, but sometimes journals or other publishers will request other formats, such as TIFF). If saving as an image or a PDF, you can also specify the size of the saved image near the top right of the window that pops up, but so long as your figure looks a decent size in this window and you have no particular reason to alter the size, the automatic measurements should suit you fine.

Occasionally you may come across a slight complication when saving plots produced in RStudio if you have stretched the Plots window in order to fit your full figure. Even if the plot looks as planned in the window, when you then click to export it you are sometimes presented with the default-sized window version of the plot to copy or save. A general method around this involves following these simple steps:

1. Create your plot and stretch the plotting window to however you want it.
2. Decide on the format that you want, e.g. JPEG, PNG, TIFF, PDF. For a JPEG file, you’ll need to use the **jpeg** driver.
3. Think of a name for the file that you will use to save your graph. Your plot will be stored under this name in the current directory (check this by typing **getwd()** in the R Console—likely just your ‘Documents’).
4. Run a line of code with the driver type and your file name, e.g.:

**jpeg('example\_plot.jpg')**

1. Now enter the code for the plot. It will not be re-plotted in R; the commands will be saved to a file instead.
2. Then run the **dev.off()** command. This saves the plot under the name you set, and you will then be able to find it in the directory.

This issue of stretched window plots not saving in the correct proportions does not occur in base R. There, if your plot fits within the window you have created, it will save in those proportions. So if our steps above seem too daunting, just run the code directly in base R rather than through RStudio. Alternatively, if you want more figure exporting options (e.g. ways to export figures of specific measurements, or as different file types, or in high resolution), see the Further Reading at the end of this document.

# S.1.4. Loading in data sets

## S.1.4.1. Setting up spreadsheets for use in R

In section S.1.2 we entered simple data into R manually (and we do this again in chapters 2 and 3). However, you may more commonly need to load in an existing, larger data set with which to produce a figure. Excel is the ideal tool to use to load larger data sets into R. When setting up an Excel data file for use in R, there are some simple rules you can follow to make your life easier:

* Give each column a short, informative name, ideally without capital letters or spaces to avoid typos later. R will use these names.
* Make sure all data is included in table form, and don’t have anything outside of labelled columns.
* Make sure you don’t press space after inputting some values but not others. This is particularly important for categorical data labels as R will interpret these as different values.

To save an Excel file for use in R you need to save it as a specific type of Excel file called a .csv file. When you save it:

* Give it whatever name you like.
* At the bottom of the saving box, there is a menu for choosing the file type. Always choose “CSV (Comma delimited)”.

**Note:** If using a Mac, sometimes saving a spreadsheet as a .csv in Excel can return a warning (which can be ignored), and troubleshooting the .csv can be done using textedit. For example, this might be needed if a space has been included after a value where it shouldn’t have been.

# S.1.4.2. Opening your data in R

If you follow the simple rules in section S.1.4.1, you can then easily load the data into R by typing one simple command into the RStudio Console (or running it from your script—see section S.1.3.1).

**alldata <- read.table(file.choose(), header = T, sep =** **",")**

This is not as spooky as it looks. ‘**alldata**’ is just a name that we want to use for our collection of data—we could have used any name. We have met the assignment character ‘**<-**‘ before. ‘**read.table**’ is a clever function for reading data into R. ‘**file.choose()**’ tells R to open a box and let you choose the file you want to open in RStudio, ‘**header = T**’ tells R to use the names you gave your columns to identify different parts of the data, and ‘**sep = ","**’ warns it to expect that numbers will be separated by commas (rather than, say, TABs) in the file. There are other variations of this code that could be used to load in data, such as versions that include specific file names and directories to save you from having to manually select the data (see the Further Reading of this document if you are interested in these), but we think that this universal command is the easiest to use, as it can be copied and pasted to be used in any scenario and all you have to edit is the name you want to assign the data. (And you do not even *have* to change that!)

Alternatively, data can be loaded into RStudio by using the ‘Import Dataset’ button under the ‘Environment’ tab of the top right-hand window. If you click on this button, you will get a drop-down list of places you can load data sets from. For .csv files, as we advise you to use above, you can select ‘From Text (base)’ and R will take you to your documents to select the appropriate file. As in the code version above, we still need to tell R that the column names are headers. So if this is not automatically selected in the window that pops up, then make sure you select ‘Yes’ for the ‘Heading’ section. Likewise, the ‘Separator’ should automatically be set to ‘Comma’, but select this if not. All the other settings R selects automatically should be fine, so you can then go ahead and press the ‘Import’ button at the bottom of the window. If you follow this method for loading in data, the data set will not be given the name ‘**alldata**’ as when using the above code—it will just be named the same as the data file. If you want to give the data set a shorter name than the file name itself, you can just run the code below with your data file’s name in place of ‘**filename**’:

**alldata<-filename**

Whichever method you follow, your data is now ready for you to look at and analyse. If you want to have a quick check that your data has loaded in properly, just type the following into the Console:

**View(alldata)**

Your data should pop up as a tab on your top left-hand window. In this book, we provide data sets as .csv files to be loaded in for the examples included in chapters 4, 5, 6, and 8.

# S.1.5. Installing and activating packages in RStudio

Although base R is powerful in itself, its functionality is continually being expanded as people write new (entirely free to use) add-on packages to carry out extra statistical or graphical functions. Sometimes you will want to achieve a particular goal that requires the use of one of these packages, but fortunately they are very easy to install into your version of RStudio. We will use a number of different add-on packages when producing more refined versions of figures throughout the book, but if you do not know the name of the package you need, then Googling the function you require is usually a good bet.

Once you know the name of the package you need, you first need to have a look at the bottom right-hand side window of RStudio. This is usually where your plots will appear when you are producing figures, but at the top of that window, you will see a ‘Packages’ tab to the right of the ‘Plots’ tab. Select this, and you will see a list of available packages. Next, find the ‘Install’ button below the ‘Packages’ tab and a pop-up window will appear. The ‘Install from’ field should automatically be set to ‘Repository (CRAN)’, and the ‘Install to Library’ field should also automatically suggest that the package be saved in the version of Ryou have installed, so you can leave both of these as they are. All you need to do is type the package name into the empty ‘Packages’ field and click on ‘Install’. Alternatively, you could just run the command line below through the R Console, with your required package substituted in where we have written ‘packagename’:

**install.packages("packagename")**

We use this latter approach several times throughout the book, but whichever method you follow, you should see the installation run in your Console and finish with a message telling you that the package has been successfully installed. Once a package has been installed onto your version of RStudio, it will remain there permanently, so you do not need to install it at the start of every session where you will need it. This is good to remember, because sometimes packages can take a little while to install.

Following installation, R requires a final, simple step before the package is ready for use in your Rsession. To do this, you simply need to substitute in the actual name of the package in the code below (note that no quotation marks are needed around the package name this time) and run it:

**library(packagename)**

Once this command has run, either directly in the Console or from a script, your package will be ready to use. However, unlike the installation, you do need to activate a package (using **library**) every time you want to use it in a new RStudio session.

# Further reading

* [‘Exporting plots’](https://intro2r.com/export-plots.html)
* [‘Exporting nice plots in R’](https://www.r-bloggers.com/2013/02/exporting-nice-plots-in-r/)
* [‘High Resolution Figures in R’](https://www.r-bloggers.com/2013/03/high-resolution-figures-in-r/)
* [‘Reading and Importing Excel Files into R’](https://www.datacamp.com/community/tutorials/r-tutorial-read-excel-into-r)
* [‘Importing Data’](https://www.statmethods.net/input/importingdata.html)