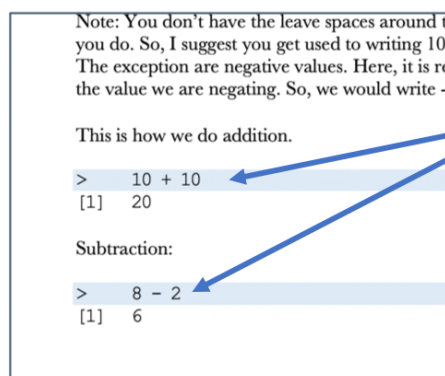


## FASS512: Second steps in R

Professor Patrick Rebuschat, [p.rebuschat@lancaster.ac.uk](mailto:p.rebuschat@lancaster.ac.uk)

This week, we will do our next steps in R. Please work through the following handout at your own pace.

As in the previous handout, please type the commands in your computer. That is, **don't just read the commands on the paper, please type every single one of them.**



Note: You don't have the leave spaces around t  
you do. So, I suggest you get used to writing 10  
The exception are negative values. Here, it is re  
the value we are negating. So, we would write -

This is how we do addition.

```
> 10 + 10
[1] 20
```

Subtraction:

```
> 8 - 2
[1] 6
```

Every time you see these shaded lines, please **type the commands** either in the console or the script editor, as appropriate.

If you don't complete the handout in class, please complete the rest at home. This is important as we will assume that you know the material covered in this handout. And again, the more you practice the better, so completing these handouts at home is important.

Finally, this handout assumes that you have installed R and RStudio and that you have completed all previous handouts. If you haven't please do this before working on the following handout. Handouts are available on [Moodle](#).

### References for this handout

Many of the examples and data files from our class come from these excellent textbooks:

- Andrews, M. (2021). *Doing data science in R*. Sage.
- Crawley, M. J. (2013). *The R book*. Wiley.
- Fogarty, B. J. (2019). *Quantitative social science data with R*. Sage.
- Winter, B. (2019). *Statistics for linguists. An introduction using R*. Routledge.

Are you ready? Then let's start on the next page! 📄

## Step 1: *Scripts*

A script is essentially a sequence of commands that we want R to execute. As Winter (2019) points out, we can think of our R script as the recipe and the R console as the kitchen that cooks according to this recipe. Let's try out the script editor and write our first script.

When working in R, try to work as much as possible in the *script*. This will be a summary of all of your analyses, which can then be shared with other researchers, together with your data. This way, others can reproduce your analyses.

### *Executing R code in scripts*

Thus far, you have typed your command lines in the console. This was useful to illustrate the functioning of our R, but in most of your analyses you won't type much in the console. Instead, we will use the script editor.

The script editor is the pane on the top left of your window. If you don't see it, you need to open a new script first. For this, press Cmd+Shift+N (Mac) or Ctrl+Shift+N (Windows). Alternatively, in the menu, click File > New File > RScript.)

In the script editor (not the console), type the following command in line 1 press Return (Mac) / Enter (Windows).

```
1 2 + 3
```

As you can see, nothing happened. There is no output in the Console pane; the cursor just moved to the next line in the script editor (line 2). This is because you did not execute the script.

To execute a command in the script editor, you need to place your cursor anywhere on the line you wish to execute and then click the Run icon in the Script editor pane. If you do this, then the following output will appear in your Console.

```
> 2 + 3
```

```
[1] 5
```

You can also run the current command line or selection in the script by pressing Cmd+Return (Mac) or Ctrl+Enter (Windows). This will also send your command from the script editor to the console. (I suggest using the shortcut, it's much more efficient.)

In the script, you can have as many lines of code as you wish. For example, you can add the following three commands to your script. (Don't type the numbers 1-3 on the left, they represent the lines in the script editor.)

```
1 scores <- c(145, 234, 653, 876, 456)
2 round(mean(scores))
3 round(sd(scores))
```

To execute each one separately, just go to the line in question and click the Run icon or, even better, press the keyboard shortcut. If you execute each line, you will get the following output. You first created a variable `scores`, which consists of a numeric vector. You then calculated the mean and standard deviation of observed scores, and rounded the output.

```
> scores <- c(145, 234, 653, 876, 456)
```

```
> round(mean(scores))
```

```
[1] 473
```

```
> round(sd(scores))
```

```
[1] 300
```

You can also run multiple commands in one go. For this, you either highlight several lines and then press the Run icon (or keyboard shortcut).

To execute *all* commands in the script, you click the Source icon (next to the Run icon) in the Script editor pane. Or just use the shortcut `Cmd+Option+R` (Mac) or `Ctrl+Alt+R` (Windows).

### *Multiline commands*

Using the script editor is particularly useful when we write long and complex commands. The example command below illustrates this nicely.

This is a fairly long command, written in the console in one line.

```
> df <- data.frame(name = c('jane', 'michaela', 'laurel', 'jaques'),
  age = c(23, 25, 46, 19), occupation = c('doctor', 'director',
  'student', 'spy'))
```

And here is the same command, split across multiple lines in the script editor.

As you can see, the use of indentations makes the structure of the data frame (names, age, occupation) much clearer. Presenting the command in one line or multiple lines does not make a difference to R.

```
1 df <- data.frame(name = c('jane', 'michaela', 'laurel', 'jaques'),
2   age = c(23, 25, 46, 19),
3   occupation = c('doctor', 'director', 'student', 'spy'))
```

### *Comments*

An important feature of R (and other programming languages) is the option to write comments in the code files. Comments are notes, written around the code, that are ignored when the script is executed. In R, anything followed by the `#` symbol on any line is treated as a comment. This

means that a line starting with `#` is ignored when the code is being run. And if we place a `#` at any point in a line, anything after the hash tag is also ignored. The following code illustrates this.

Comments are really useful for writing explanatory notes to ourselves or others.

```
1  # Here is data frame with three variables.

2  # The variables refer to the names, ages, and occupations of the
   participants.

3

4  df <- data.frame(name = c('jane', 'michaela', 'laurel', 'jaques'),
5                    age = c(23, 25, 46, 19),
6                    occupation = c('doctor', 'director', 'student',
                                   'spy'))
```

Another example:

```
1  2 + 3  # This is addition in R.

2  2 / 1  # This is division.
```

### *Code sections*

To make your script even clearer, you can use code sections. These divide up your script into sections as in the example below. To create a code section, go the line in the script editor where you would like to create the new section, then press `Cmd+Shift+R` (Mac) or `Ctrl+Shift+R` (Windows). Alternatively, in the Menu, select `Code > Insert Section`.

Lines 1 and 5 are code sections.

```
1  # Create vectors -----

2  scores_test1 <- c(1, 5, 6, 8, 10) # These are the scores on the pre-
   test.

3  scores_test2 <- c(25, 23, 52, 63) # These are the scores on the
   post-test.

4

5  # A few calculations -----

6  mean(scores_test1)

7  mean(scores_test2)
```

```
8 round(mean(scores_test2) - mean(scores_test1)) # The difference btw
pre and post-tests.
```

Once you have created a section, you can ask R to run only the code in a specific region. This is because R recognizes script sections as distinct regions of code.

To run the code in a specific section, first go to the section in question (e.g., the section called # A few calculations -----) and then either press Cmd+Option+T (Mac) or Ctrl+Alt+T (Windows). You can also use the menu, Code > Run Region > Run Section. Have a go to see if this works out well.

### *Saving a script*

Finally, you can also save your script. To do this, just click the Save icon in the Script editor pane or press Cmd+S (Mac) or Ctrl+S (Windows). The script can be named anything, but it is often recommended to use lowercase letters, numbers and underscores only. (That is, no spaces, hyphens, dots, etc.)

The script is saved in the .R format in your directory. If you later double click it, the file will open in RStudio by default, but you can also view and edit the file in Word and similar programs.

## Step 2: *Installing and loading packages*

Packages are extensions to the R programming language. When you first install R, you automatically have access to a selection of core packages (“Base R”). For example, the base package contains the basic R functions, the graphics package has functions for basic graphics, and the stats package contains statistical functions.

One of the things that makes R so powerful is that there are currently over 16,000 additional packages that you can install on demand and for free.

What packages you will install depends on what you are planning to do. For example, if you are planning to run linear-mixed effect models, you are likely to install and load the `lme4` package.

One package that most textbooks recommend is `tidyverse`. This is actually a collection of packages to facilitate the data analysis and visualization. It includes well-known packages such as `ggplot2`, `dplyr`, and `tibble`. We will use the `tidyverse` package regularly, so you please remember to install it for each session.

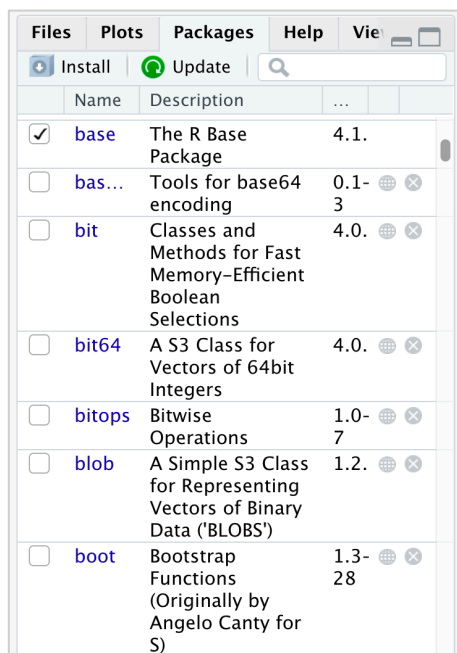
### *Installing and loading a package*

Let's begin by installing and loading the `tidyverse` package.

The fastest way to install a package is by using the `install.packages()` function, followed by the `library()` function to load it. I suggest you do this via the script editor. Simply type the two command lines, then run them both.

```
1 install.packages('tidyverse')
2 library(tidyverse)
```

Alternatively, you can see also install packages by using the Packages tab in the Files, Plots, Packages, etc. pane. As you see in the figure below, the base package is already installed. You can install more packages by scrolling through the list (or using the search option to narrow down the choices) and then selecting the tick box to the left of the package. If you do this, you will see that the click will run the `install.packages()` command in the console, so you might as well just type it yourself... ☺



### *Citing R and package developers*

It's important to acknowledge the important work done by the developers who make R packages available for free and open source. When you use a package for your analyses (e.g., `tidyverse` or `lme4`), you should acknowledge their work by citing them in your output (dissertation, presentation, articles, etc.).

You can find the reference for each package via the `citation()` function, as in the two examples below.

```
> citation("tidyverse")
```

```
Wickham H, Averick M, Bryan J, Chang W, McGowan LD, François R,
Grolemund G, Hayes A, Henry L, Hester J, Kuhn M, Pedersen TL, Miller E,
Bache SM, Müller K, Ooms J, Robinson D, Seidel DP, Spinu V, Takahashi K,
Vaughan D, Wilke C, Woo K, Yutani H (2019). "Welcome to the tidyverse."
_Journal of Open Source Software_, *4*(43), 1686. doi:
10.21105/joss.01686 (URL: https://doi.org/10.21105/joss.01686).
```

```
> citation("lme4")
```

To cite `lme4` in publications use:

Douglas Bates, Martin Maechler, Ben Bolker, Steve Walker (2015). Fitting Linear Mixed-Effects Models Using lme4. Journal of Statistical Software, 67(1), 1-48. doi:10.18637/jss.v067.i01.

And if you need to cite R itself, you can find out the reference for this as follows:

```
> citation()
```

To cite R in publications use:

R Core Team (2021). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.

```
> R.Version() $version.string
```

```
[1] "R version 4.1.2 (2021-11-01)"
```

### Step 3: *Working directories and clean workspaces*

Every R session has a working directory. This is essentially the directory or folder from which files are read and to which files are written.

You can find out your working directory by typing the following command. Your output will obviously look different from the one below, which refers to my machine. 😊

```
> getwd()
```

```
[1] "/Users/patrickrebuschat/Desktop/R working directory" # Output on mac
```

```
[1] "C:/Users/patrickrebuschat/Desktop/R working directory" # Output on Windows
```

You can also use a command to list the content in the working directory. (Alternatively, you can see your direct by using the Files tab in the Files, Packages, Plot, etc. pane.)

```
> list.files()
```

```
[1] "exam.csv" "nettle_1999_climate.csv" "Patrick R script.R"
```

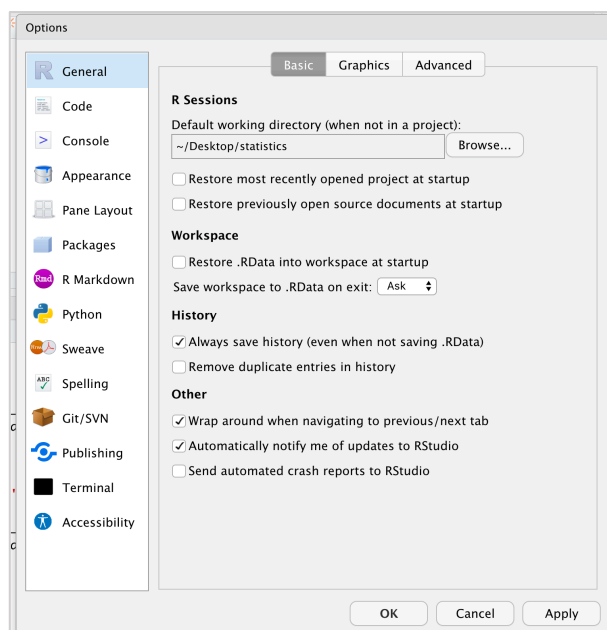
I suggest you create a new working directory on your computer desktop and then use it for the entire course. Important files related to your R tasks (scripts, data, etc.) should later be downloaded to this folder.

The first step is for you to create a folder called “statistics” (or similar) on your desktop. You can do this by going to the Files tab (in the Files, Packages, etc. pane) and clicking the “Create a new folder” icon.

Once you have created the “statistics” folder on the desktop, go to the menu to set the default working directory to the new “statistics” folder. The easiest way is to go to the menu, RStudio > Preferences. This should call up the following window.

In the window, click the Browse button and set the default working directory to the “statistics” folder in the desktop.

One more thing: I suggest you unclick the two boxes beneath the directory path (the ones saying “Restore...”). It’s better to start each R session with a blank space instead of loading all previous scripts and commands.



#### Step 4: *Loading data*

When we are dealing with data in our analyses, we usually begin by importing a data file. R allows you to import data files in many different formats, but the most likely ones are .csv and .xlsx.

I have uploaded several data files to our Moodle page. Please go to folder called “Data sets to download for this session” in the section for today’s session, then download the files in the folder and place them in your working directory (the “statistics” folder you just created on your desktop). The files are from Winter (2019) and Fogarty (2019).

Let’s try out loading data files. In the examples below, you will import three types of files: .csv, .txt, and .xlsx. Remember: You need to download the data files from our Moodle page and place them in our working directory first. Otherwise, you cannot import the files from our directory into R.

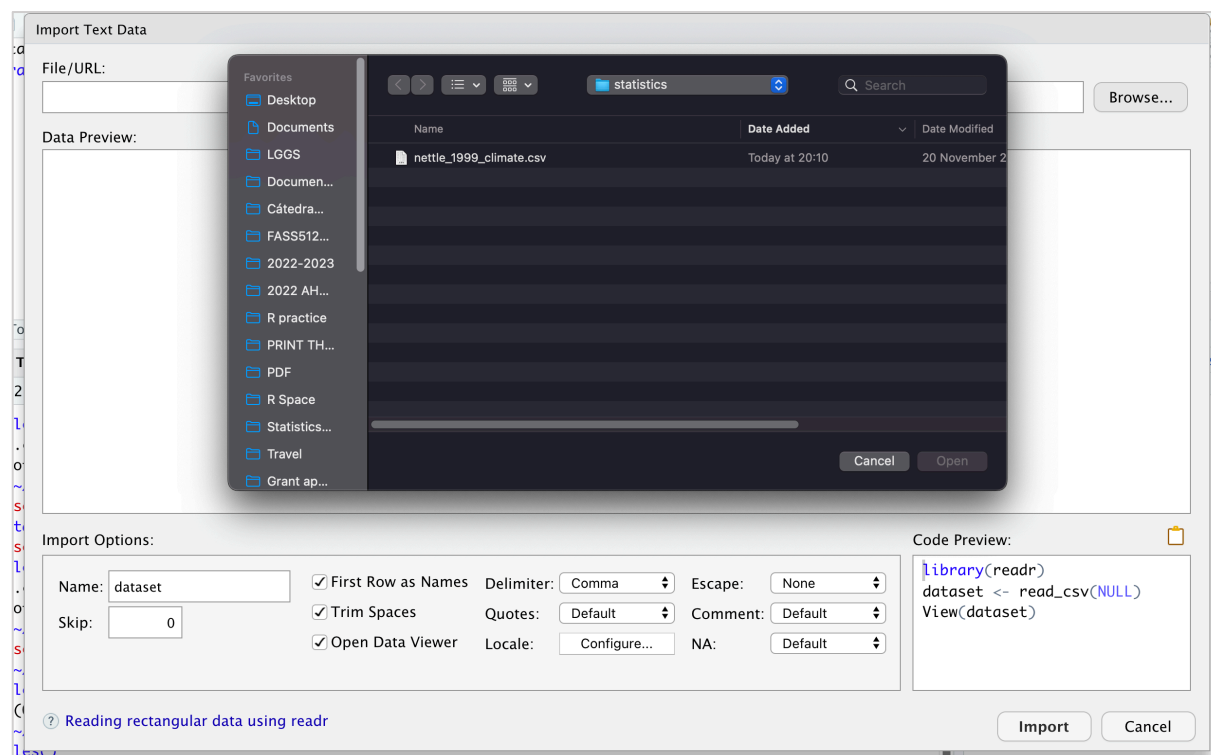
##### *Loading .csv files*

We can use the `read.csv()` function to load data that is in .csv format. The command below will load the data set ('nettle\_1999\_climate.csv') and create a new label for this data set (languages).



```
> languages <- read.csv('nettle_1999_climate.csv')
```

Alternatively, you can load data files by clicking File > Import Dataset > From Text (readr). In the dialogue window, then click browse and select the file `nettle_1999_climate.csv`. You can change the name of the data set in the text box at the bottom left, below Import Options, where it says Name.



Your Script editor should then display the data set `languages`.

### Loading .txt files

The data file you just imported is in the .csv format. You can import data from files in other formats, too. If the data is in .txt format, you can simply use the following command.

```
text_file <- read.table('example_file.txt', sep = "\t", header = TRUE)
```

(Note: Ignore the warning message in the console.)

The command creates a new data set called `text_file`. The `read.table()` function requires you to specify a separator (the argument `sep`). A separator is whatever divides your columns in your table. If your data is tab-delimited, then you include `sep = "\t"` in your command, as above. If you are using commas, then you need to include `sep = ","` and so forth. The `header = TRUE` argument indicates that your data set has a head (usually the column names).

### Loading .xlsx files

If the data is an Excel spreadsheet (e.g., .xlsx format), you can proceed as follows.

```
library(readxl)

spreadsheet_exl <- read_excel('simd.xlsx', sheet = 'simd')
```

First, you need to install the `readxl` package. Then, you create a new data set called `spreadsheet_exl` by using the `read_excel()` function.

Note: Since spreadsheets have multiple sheets, you need to specify the name of the sheet you would like to import by using the `sheet` argument. In our case, the sheet is called `simd`, hence `sheet = 'simd'`.

RStudio can handle many other file extensions to import datasets. You can find out information on how to import other file types by using the R help function (or by searching on Google).

### Step 5: *Examining datasets*

If you have followed the steps above, you will have imported three data sets, `languages`, `spreadsheet_exl`, and `text_file`. You can now start exploring the data. We will focus on `languages` as an example.

Every time you import data, it's good to check the content, just to make sure you imported the correct file.

The easiest way to do this is by using the `View()` function. This allows you to inspect the data set in the script editor. Note: The function requires a capital V.

If you run the command below, you will see that this shows the data (a table) in a tab of the script editor. It will also be displayed in the console.

```
> View(spreadsheet_exl)

> View(languages)
```

languages

	Country	Population	Area	MGS	Langs
1	Algeria	4.41	6.38	6.60	18
2	Angola	4.01	6.10	6.22	42
3	Australia	4.24	6.89	6.00	234
4	Bangladesh	5.07	5.16	7.40	37
5	Benin	3.69	5.05	7.14	52
6	Bolivia	3.88	6.04	6.92	38
7	Botswana	3.13	5.76	4.60	27
8	Brazil	5.19	6.93	9.71	209
9	Burkina Faso	3.97	5.44	5.17	75

Showing 1 to 9 of 74 entries, 5 total columns

Console

```
R 4.1.2 ~ /Desktop/statistics/
58
59 South Africa 4.56 6.09 6.05 32
60 Sri Lanka 4.24 4.82 9.59 7
61 Sudan 4.41 6.40 4.02 134
62 Suriname 2.63 5.21 12.00 17
63 Tanzania 4.45 5.98 7.02 131
64 Thailand 4.75 5.71 8.04 82
65 Togo 3.56 4.75 7.91 43
66 UAE 3.21 4.92 0.83 9
67 Uganda 4.29 5.37 10.14 43
68 Vanuatu 2.21 4.09 12.00 111
69 Venezuela 4.31 5.96 7.98 40
70 Vietnam 4.83 5.52 8.80 88
71 Yemen 4.09 5.72 0.00 6
72 Zaire 4.56 6.37 9.44 219
73 Zambia 3.94 5.88 5.43 38
74 Zimbabwe 4.00 5.59 5.29 18
> languages <- read.csv('nettle_1999_climate.csv')
> View(languages)
>
```

You can also inspect your data by visiting the Environment tab in the Environment, History, Connections, etc. pane. As you can see in the figure below, this will tell you that languages has 74 observations (rows) and five variables (columns).

RStudio

Environment

languages

	Country	Population	Area	MGS	Langs
1	Algeria	4.41	6.38	6.60	18
2	Angola	4.01	6.10	6.22	42
3	Australia	4.24	6.89	6.00	234
4	Bangladesh	5.07	5.16	7.40	37
5	Benin	3.69	5.05	7.14	52
6	Bolivia	3.88	6.04	6.92	38
7	Botswana	3.13	5.76	4.60	27

Showing 1 to 7 of 74 entries, 5 total columns

Environment

languages 74 obs. of 5 variab...

Data

languages 74 obs. of 5 variab...

language... 17 obs. of 5 variab...

spreadsh... 6976 obs. of 22 var...

text\_file 3 obs. of 3 variabl...

If you would like to examine variables, you can start by using the `str()` function, as in the example below.

```
str(languages)
```

```
'data.frame':    74 obs. of  5 variables:
```

```

$ Country    : chr  "Algeria" "Angola" "Australia" "Bangladesh" ...
$ Population: num  4.41 4.01 4.24 5.07 3.69 3.88 3.13 5.19 3.97 3.5 ...
$ Area       : num  6.38 6.1 6.89 5.16 5.05 6.04 5.76 6.93 5.44 5.79 ...
$ MGS        : num  6.6 6.22 6 7.4 7.14 6.92 4.6 9.71 5.17 8.08 ...
$ Langs      : int  18 42 234 37 52 38 27 209 75 94 ...

```

As you can see above, the `str()` function will tell you many useful things about your dataset. For example, it will reveal the number of observations (rows, 74) and variables (columns, 5), and then list the variables (Country, Population, Area, MGS, Langs). For each variable, it will also indicate the variable type (chr = character strings, num = numeric, intd = integer). The `str()` function will also display the first observations of each variable (Algeria, Angola, Australia, Bangladesh, etc.).

You can also check the names of variables separately by using the `names()` function, or check the variable type by checking the `class()` function, but it's easier to just use the `str()` function as in the example above.

If you prefer, you can restrict your inspection of to the first or final rows of the data set. You can do this by using the `head()` and `tail()` function. This is helpful if you're tables has lots of rows.

```
> head(languages)      # This displays the first six rows.
```

	Country	Population	Area	MGS	Langs
1	Algeria	4.41	6.38	6.60	18
2	Angola	4.01	6.10	6.22	42
3	Australia	4.24	6.89	6.00	234
4	Bangladesh	5.07	5.16	7.40	37
5	Benin	3.69	5.05	7.14	52
6	Bolivia	3.88	6.04	6.92	38

```
> tail(languages)      # This displays the last six rows.
```

	Country	Population	Area	MGS	Langs
69	Venezuela	4.31	5.96	7.98	40
70	Vietnam	4.83	5.52	8.80	88
71	Yemen	4.09	5.72	0.00	6

```

72  Zaire      4.56 6.37 9.44 219
73  Zambia     3.94 5.88 5.43 38
74  Zimbabwe   4.00 5.59 5.29 18

```

There is also a very helpful function is called `summary()`. As you can see in the example below, this function will provide you with summary information for each of your variables.

For numeric/integer variables such as `Populations`, `Area`, `MGS`, and `Langs`, this command will calculate the minimum and maximum values, quartiles, median and mean. (We will discuss summary statistics in more detail later.)

For character variables, as in `Country`, the command will simply provide you with the number of observations (`length`) for this variable.

```
> summary(languages)
```

```

      Country      Population      Area      MGS      Langs
Length:74      Min.    :2.010   Min.    :4.090   Min.    : 0.000   Min.
: 1.00
Class :character 1st Qu.:3.607   1st Qu.:5.223   1st Qu.: 5.348   1st
Qu.: 17.25
Mode  :character Median :3.990   Median :5.640   Median : 7.355
Median : 40.00
Mean   :3.992   Mean   :5.618   Mean   : 7.029   Mean   : 89.73
3rd Qu.:4.393   3rd Qu.:6.032   3rd Qu.: 9.193   3rd Qu.: 93.75
Max.   :5.930   Max.   :6.930   Max.   :12.000   Max.   :862.00

```

In large datasets, you might want to examine only a specific variable. You can do this by using the `$` as an index. For example, if you would just like to examine the variable `Population` in the `languages` dataset, you could proceed as follows.

```
> str(languages)
```

```
> str(languages$Population)
```

```
num [1:74] 4.41 4.01 4.24 5.07 3.69 3.88 3.13 5.19 3.97 3.5 ...
```

```
> class(languages$Population)
```

```
[1] "numeric"
```

```
> head(languages$Population)
```

```
[1] 4.41 4.01 4.24 5.07 3.69 3.88
```

```
> tail(languages$Population)
```

```
[1] 4.31 4.83 4.09 4.56 3.94 4.00
```

```
> summary(languages$Population)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
2.010	3.607	3.990	3.992	4.393	5.930

### Step 6: *Closing your R session*

You have now completed this handout! The last step is to close your R session. When you quit RStudio, a prompt will ask whether you want to save the content of your workspace. It is better to **NOT** save the workspace. When you start RStudio again, you will have a clean workspace. You then just re-run your scripts.

So, I would save R scripts (especially if these are very long and are relevant to your analyses), but I would not the workspace contents.