Week 3 workbook: Introduction to Data Analysis

Site:	Moodle VLE	Printed by:	Marta Shocket
Course:	24/25: LEC.243: Experimental Design and Analysis [1]	Date:	Monday, 14 October 2024, 21:31
Book:	Week 3 workbook: Introduction to Data Analysis		

Table of contents

3. RStudio introduction

Now we're going to start learning how to use R and RStudio to analyse data.

R is a free, open-source **programming language** designed for statistical computing and graphics. It is the most commonly used tool for statistical analysis in ecology and is gaining popularity in many other disciplines as well.

RStudio is a free, open-source **integrated development environment** (IDE) that makes coding in R much easier.

Why use R and RStudio?

- R and RStudio are both free and open-source. This means you'll be able to keep using them after you leave LU.
- R uses scripts of code (i.e. instructions) to perform analyses instead of pointing and clicking with a mouse. This makes it easier to re-run an analysis and share code, making science more *open* and *reproducible*.
- R is a bit harder to learn than Excel initially, but once you learn it, it is much more powerful and capable of doing complex tasks.
- A significant portion of modern tools for analysis in ecology, environmental science, and many other fields are written in R. This library of tools is being constantly updating as researchers create new ones and improve old ones.

Module learning outcomes and teaching approach

That said, this module is not really about coding per se. We do not expect you to become master coders during this 10 week module, most of which is focused on ideas about experimental design and statistics. We *do* expect that you will be able to interpret, modify, and use R code to perform some basic statistical analyses and visualise data.

Here are the learning outcomes for the module related to R. At the end of the module, you will be able to:

- Effectively use the RStudio coding environment
- Analyse example code and explain generally what the code is doing
- Take example code and modify it to apply to a new data set
- Conduct basic statistical analyses and data visualisations - with access to your past work from the practical sessions and other online resources

Starting from now, each week you will use the module workbooks to complete exercises in R.

These exercises will be a bit like an immersive foreign language program - you will **gradually learn how to code after repeated exposure and practice.**

Do get discouraged if it feels like it takes you longer than your "normal" to understand what is going on. Learning how to code is not just learning science content like you are used to, it is learning a whole new way to think. This process necessarily takes time, and we have incorporated this need for time and repetition into the module design.

An additional note on coding

Beginning coders are often under the impression that more advanced programmers have all of the commands they need memorized and write everything out perfectly the first time. In fact, nothing could be farther from the truth.

The two most important skills you need to be a successful coder are:

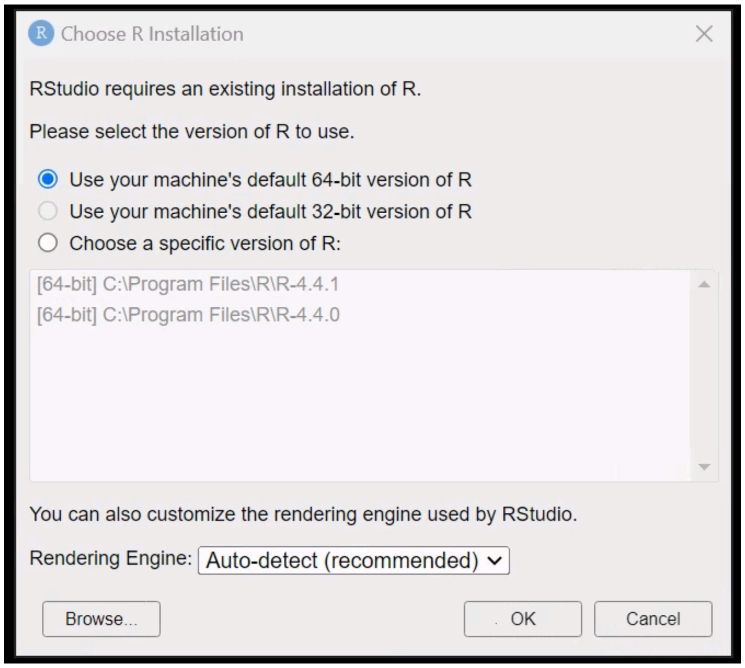
1. Looking things up as you code them, either online, in help files, or from previous work. (We often work by adapting example code.)
2. Debugging your code when it doesn’t work

These skills are key parts of the coding process, no matter how advanced you become. Embrace them.

3.1. The RStudio Interface

Open RStudio.

The first time you do this, you will get a message asking you which version of R you want RStudio to communicate with behind the scenes. (You will never have to interact with R directly - RStudio does all of that for you.)



Choose R Installation

RStudio requires an existing installation of R.

Please select the version of R to use.

[✓ Use your machine's default 64-bit version of R]
Use your machine's default 32-bit version of R
Choose a specific version of R:

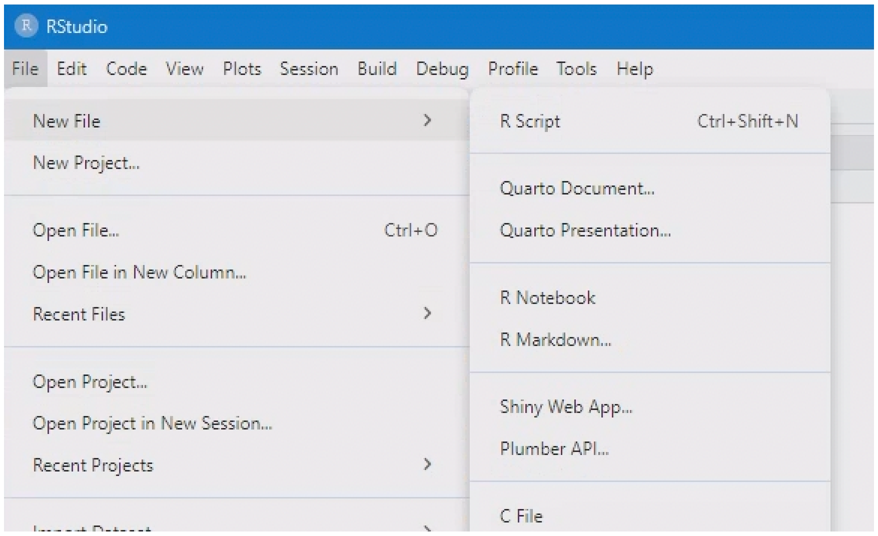
[64-bit] C:\Program Files\R\R-4.4.1
[64-bit] C:\Program Files\R\R-4.4.0

You can also customize the rendering engine used by RStudio.

Rendering Engine: Auto-detect (recommended)

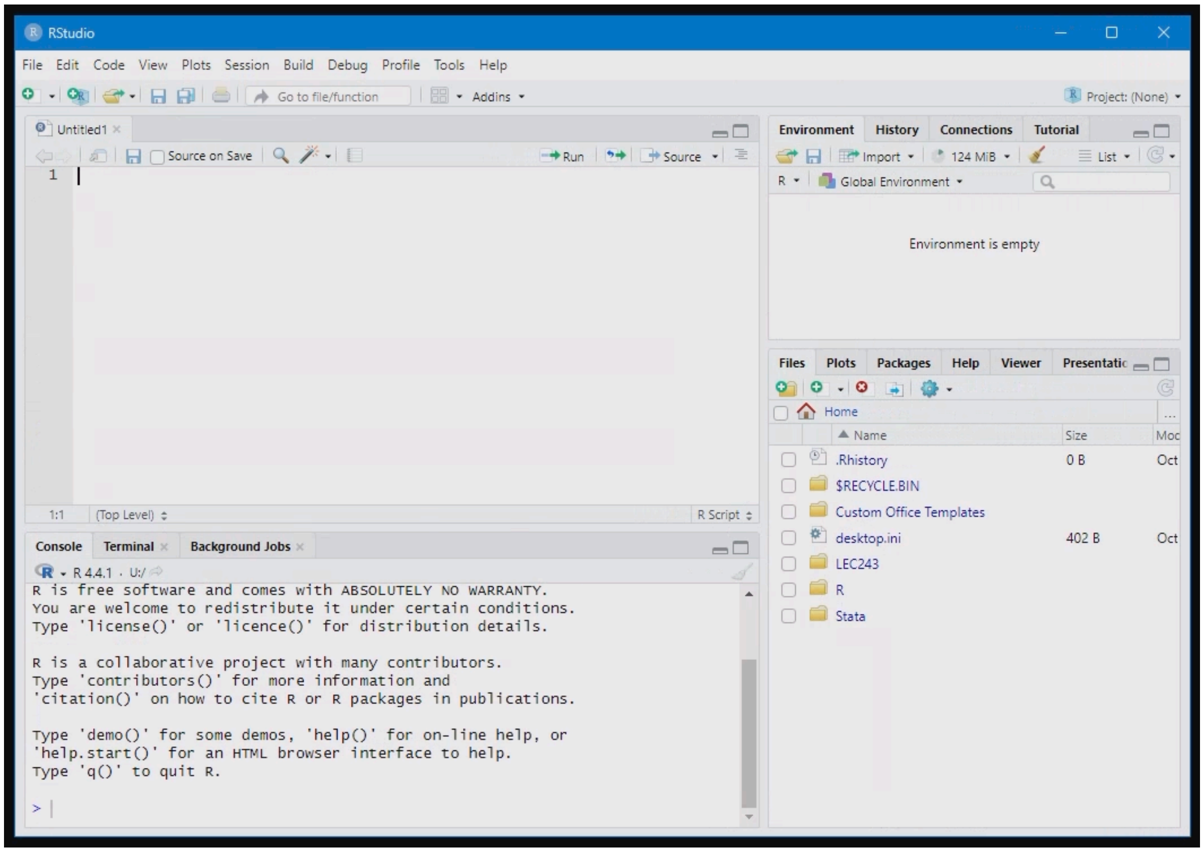
Select the default option (64-bit version of R) and click OK. You can also select OK for uploading crash reports.

Before we do anything else, click on `File` > `New File` > `R Script` to create a new R script. This should open up a new panel in your window.



Now your window should be divided into four quadrants. This is what RStudio will usually look like.

- *Top left*: Code files - where you can see and edit your saved code files (called "**scripts**").
- *Bottom left*: `Console` - where you can input commands and see code that has been run. This is also where text-based output gets printed.
- *Top right*: `Environment` - where you can see the saved data objects in your current R workspace.
- *Bottom right*: This panel has four very useful tabs in it. 1) `Files` - where you can see your data and code files. 2) `Plots` - where image-based output appears. 3) `Packages` - where you can add and manage packages that contain extra features. 4) `Help` - where you can look up documentation about specific commands.



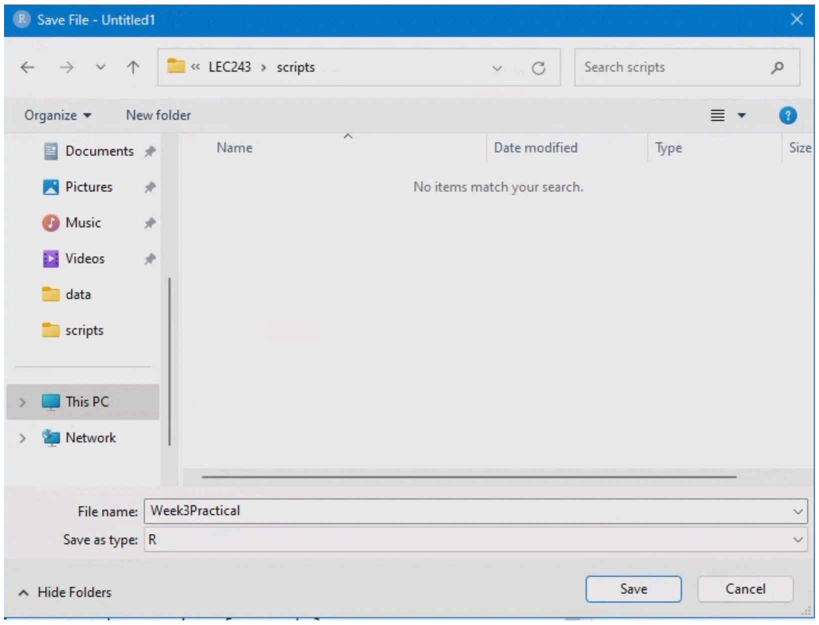
```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Click on the `Files` tab in the bottom right panel. You should see the folder `LEC243` that we created previously.

- Click on the `LEC243` folder to navigate inside it.
- Click on the `scripts` folder to navigate inside it.

Go back to the main toolbar at the top of the RStudio window.

- Select `File` > `Save`, and navigate to the `scripts` folder in the save file window.
- Enter the file name "Week3Practical", and click `Save`.
- You should see the file pop up in your files tab.



File name: Week3Practical
Save as type: R

Now that we have a script file saved, we're ready to get started using R!

IMPORTANT: You'll want to save the file you're working on periodically as you go through each practical to make sure you don't lose any work if you accidentally close your browser, etc.
To save your script, you can:

- Use the File menu.
- Use to hot keys: `Ctrl + S`.
- Press the save icon at the top of the script window tool bar.

Your saved files will be automatically stored on your Apporto U:/ drive and available the next time you log in.

3.2. The grammar of coding

Before we can talk about coding, we need to learn some vocabulary. Five essential building blocks of any coding language are:

1. operators
2. variables
3. functions
4. arguments
5. comments

Operators

Operators are typically 1 or 2 character commands that correspond to very simple, common tasks. For instance, all of the basic mathematical symbols are operators in most programming languages: `+` (addition), `-` (subtraction), `*` (multiplication), and `/` (division). In R, exponents can be expressed with either the up carrot, `^`, or a double asterisk, `**`. Thus, like Excel, R can function as a basic calculator.

- The input `84 + 22` will give the output `106.`

Type "`84 + 22`" directly into the `Console` - NOT the script file - and hit `Enter`.

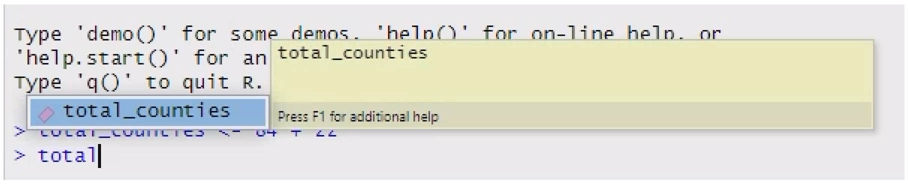
Variables (or "objects")

Data can be stored as named **variables**, which allows you to refer to and use them later. You can think of variables as analogous to nouns. From this point forward, we will refer to these saved variables as "**objects**". This is not the correct term according to computer science, but we want to be able to use "variable" in the biology/statistics sense of the word without being confusing.

- In R, we use the arrow operator `<-` to create and store objects in our R environment. (This is usually said aloud as "gets the value" or sometimes "gets" for short.)
- We can store the numbers that we used above as objects using the code:
  - `total_counties <- 84 + 22`
- Now whenever we use the name of the object (`total_counties`) R will substitute the value `106`.

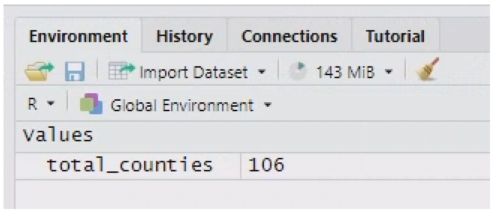
Create and store an object by typing "`total_counties <- 84 + 22`" into the `Console` and hit `Enter`. Now type "`total_counties`" and hit `Enter`.

As you type, RStudio will suggest objects that are stored in your environment that start with the same characters. You can either click on them, or use the arrow keys + Enter to select them. This will save you time and also prevent typos from causing bugs!



```
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an
Type 'q()' to quit R.
> total_counties <- 84 + 22
> total
```

Now you should be able to see the object in your `Environment` panel.



```
Values
  total_counties   106
```

Functions

Functions are the commands that tell the program to do something. You can think of them as the verbs of a programming language. Functions are always followed by a set of brackets.

- The function `getwd()` will provide your current working directory (the location where R will read and save files).

Type the function "`getwd()`" into the `Console` and hit `Enter`.

> Note that as you type the command `getwd()`, RStudio will automatically add the ending bracket when you type the starting bracket. (You will see later that it also does the same thing with quotation marks.) This helps avoid introducing bugs from forgetting to close your curly brackets.

Arguments

Most functions also take inputs called **arguments** that go inside the brackets. Arguments are generally either:

1. Data to process
2. Settings that tell the function how to process the data

- The function `sqrt(x)` takes one argument, `x`, and calculates its square root. So `sqrt(4)` will return the value `2`.

Type "`sqrt(4)`" into the `Console` and hit `Enter`.

Comments

Comments are another critical part of coding, although not part of the code itself. Comments explain what the code is doing, and are a crucial part of making your code understandable to both other people and to yourself. (Every programmer hates the feeling of going back to old code days/weeks/months/years after you first wrote it and thinking "What the #$&! was I doing here????" Commenting your code in detail will make your life much easier.)

- In R, comments are preceded by the hash symbol #
- The # symbol tells R to ignore whatever else follows on that line in the script

Type the statement "`# Hello world!`" into the `Console` and hit `Enter`.

> **Q:** What output do you see? Why?

3.3. Running code from a script

Now that you've run some basic code via the `Console`, we're going to learn how to do it from an R script.

Two methods to run code from a script are:

1. Press `Ctrl + Enter`
2. Click the `Run` button with the green arrow near the top right of the script panel

For both methods, you can either move your cursor to a specific line of code to run just that line, or highlight a larger section of code to run the whole chunk.

Type the following code into your R script file. Run the code using both methods.

- First practice running each line separately, and watch what happens in the console after each command.
- Then try running all the lines at once.

```
# Random practice code
a <- 3
b <- 5
a + b
a^2
b*a
b**a
z <- b/a
z
1:5
z <- c(1, 2, 3, 4, 5)
z
```

In R, the *colon operator* tells R to list out every integer number between A and B, when written as `A:B`.

Additionally, the small letter `c` (short for the computer science term *concatenate*) is used to group data together into a larger object called a **vector**.

You can think of a vector like a list, although the word *list* refers to a specific type of data object in R that we aren't using.

We'll use both of these operations in next week's practical.

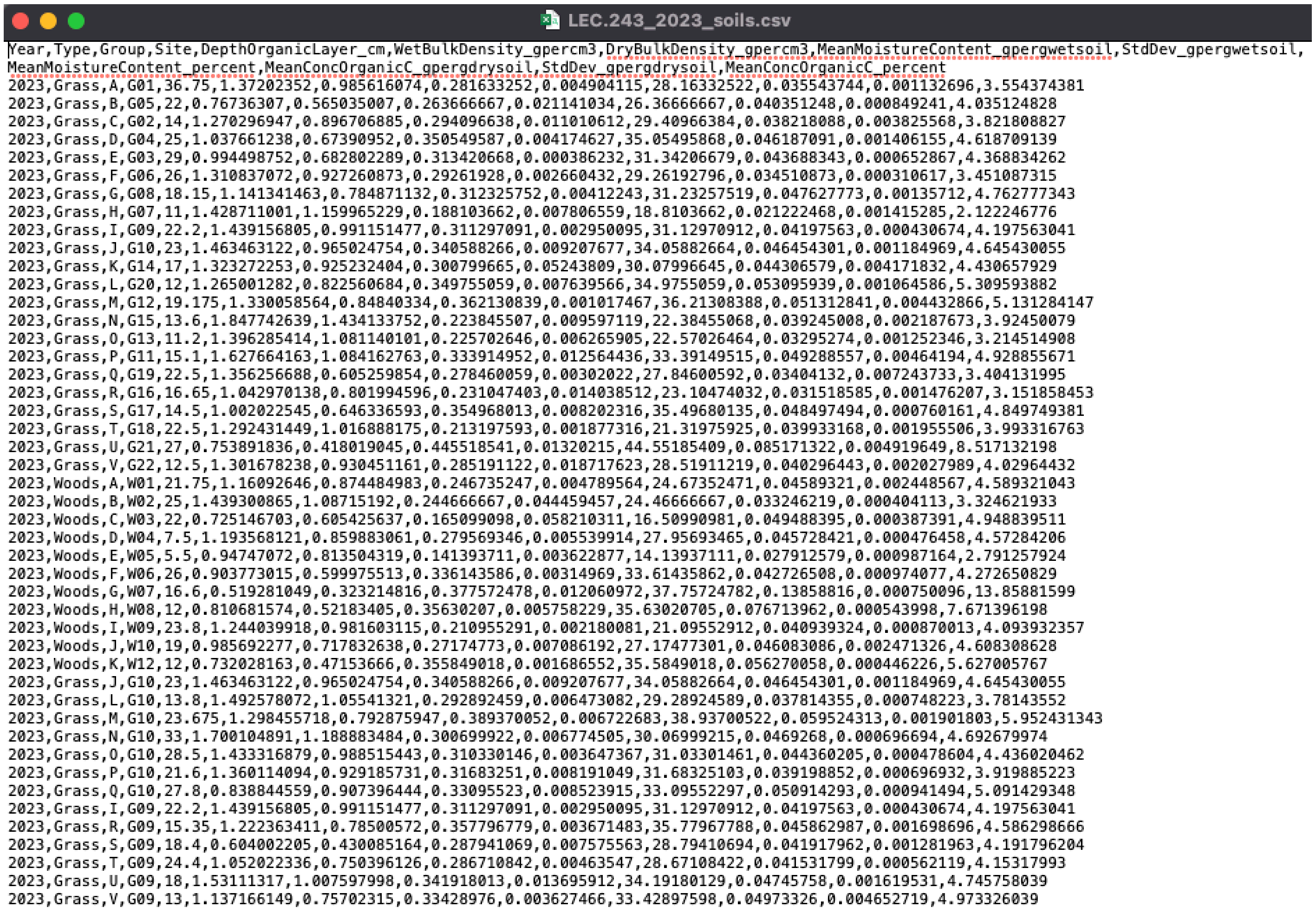
3.4. Loading data

In order to analyse data with R, first we need to load the data into R!

Many datasets are stored as Excel files, like the one you used earlier in this practical. However, R can't read Excel files directly - it needs to read the data as a comma separated value, or `.csv` file.

Any Excel file can be saved as a `.csv` file using the 'Save As' menu option. For today's exercise, we've already done that for you. (You can find the tabs that we used to create these `.csv` files in the Excel file.)

Unsurprisingly based on the name, a `.csv` file has values separated by commas. You can open one in a text editor and it will look something like this:



```
Year,Type,Group,Site,DepthOrganicLayer_cm,WetBulkDensity_gpercm3,DryBulkDensity_gpercm3,MeanMoistureContent_gpergwetsoil,StdDev_gpergwetsoil,
MeanMoistureContent_percent,MeanConcOrganicC_gpergdrysoil,StdDev_gpergdrysoil,MeanConcOrganicC_percent
2023,Grass,A,G01,36.75,1.37202352,0.985616074,0.281633252,0.004904115,28.16332522,0.035543744,0.001132696,3.554374381
2023,Grass,B,G05,22,0.76736307,0.565035007,0.263666667,0.021141034,26.36666667,0.040351248,0.000849241,4.035124828
2023,Grass,C,G02,14,1.270296947,0.896706885,0.294096638,0.011010612,29.40966384,0.038218088,0.003825568,3.821808827
2023,Grass,D,G04,25,1.037661238,0.67390952,0.350549587,0.004174627,35.05495868,0.046187091,0.001406155,4.618709139
2023,Grass,E,G03,29,0.994498752,0.682802289,0.313420668,0.000386232,31.34206679,0.043688343,0.000652867,4.368834262
2023,Grass,F,G06,26,1.310837072,0.927260873,0.29261928,0.002660432,29.26192796,0.034510873,0.000310617,3.451087315
2023,Grass,G,G08,18.15,1.141341463,0.784871132,0.312325752,0.00412243,31.23257519,0.047627773,0.00135712,4.762777343
2023,Grass,H,G07,11,1.428711001,1.159965229,0.188103662,0.007806559,18.8103662,0.021222468,0.001415285,2.122246776
2023,Grass,I,G09,22.2,1.439156805,0.991151477,0.311297091,0.002950095,31.12970912,0.04197563,0.000430674,4.197563041
2023,Grass,J,G10,23,1.463463122,0.965024754,0.340588266,0.009207677,34.05882664,0.046454301,0.001184969,4.645430055
2023,Grass,K,G14,17,1.323272253,0.925232404,0.300799665,0.05243809,30.07996645,0.044306579,0.004171832,4.430657929
2023,Grass,L,G20,12,1.265001282,0.822560684,0.349755059,0.007639566,34.9755059,0.053095939,0.001064586,5.309593882
2023,Grass,M,G12,19.175,1.330058564,0.84840334,0.362130839,0.001017467,36.21308388,0.051312841,0.004432866,5.131284147
2023,Grass,N,G15,13.6,1.847742639,1.434133752,0.223845507,0.009597119,22.38455068,0.039245008,0.002187673,3.92450079
2023,Grass,O,G13,11.2,1.396285414,1.081140101,0.225702646,0.006265905,22.57026464,0.03295274,0.001252346,3.214514908
2023,Grass,P,G11,15.1,1.627664163,1.084162763,0.333914952,0.012564436,33.39149515,0.049288557,0.00464194,4.928855671
2023,Grass,Q,G19,22.5,1.356256688,0.605259854,0.278460059,0.00302022,27.84600592,0.03404132,0.007243733,3.404131995
2023,Grass,R,G16,16.65,1.042970138,0.801994596,0.231047403,0.014038512,23.10474032,0.031518585,0.001476207,3.151858453
2023,Grass,S,G17,14.5,1.002022545,0.646336593,0.354968013,0.008202316,35.49680135,0.048497494,0.000760161,4.849749381
2023,Grass,T,G18,22.5,1.292431449,1.016888175,0.213197593,0.001877316,21.31975925,0.039933168,0.001955506,3.993316763
2023,Grass,U,G21,27,0.753891836,0.418019045,0.445518541,0.01320215,44.55185409,0.085171322,0.004919649,8.517132198
2023,Grass,V,G22,12.5,1.301678238,0.930451161,0.285191122,0.018717623,28.51911219,0.040296443,0.002027989,4.02964432
2023,Woods,A,W01,21.75,1.16092646,0.874484983,0.246735247,0.004789564,24.67352471,0.04589321,0.002448567,4.589321043
2023,Woods,B,W02,25,1.439300865,1.08715192,0.244666667,0.044459457,24.46666667,0.033246219,0.000404113,3.324621933
2023,Woods,C,W03,22,0.725146703,0.605425637,0.165099098,0.058210311,16.50990981,0.049488395,0.000387391,4.948839511
2023,Woods,D,W04,7.5,1.193568121,0.859883061,0.279569346,0.005539914,27.95693465,0.045728421,0.000476458,4.57284206
2023,Woods,E,W05,5.5,0.94747072,0.813504319,0.141393711,0.003622877,14.13937111,0.027912579,0.000987164,2.791257924
2023,Woods,F,W06,26,0.903773015,0.599975513,0.336143586,0.00314969,33.61435862,0.042726508,0.000974077,4.272650829
2023,Woods,G,W07,16.6,0.519281049,0.323214816,0.377572478,0.012060972,37.75724782,0.13858816,0.000750096,13.85881599
2023,Woods,H,W08,12,0.810681574,0.52183405,0.35630207,0.005758229,35.63020705,0.076713962,0.000543998,7.671396198
2023,Woods,I,W09,23.8,1.244039918,0.981603115,0.210955291,0.002180081,21.09552912,0.040939324,0.000870013,4.093932357
2023,Woods,J,W10,19,0.985692277,0.717832638,0.27174773,0.007086192,27.17477301,0.046083086,0.002471326,4.608308628
2023,Woods,K,W12,12,0.732028163,0.47153666,0.355849018,0.001686552,35.5849018,0.056270058,0.000446226,5.627005767
2023,Grass,J,G10,23,1.463463122,0.965024754,0.340588266,0.009207677,34.05882664,0.046454301,0.001184969,4.645430055
2023,Grass,L,G10,13.8,1.492578072,1.05541321,0.292892459,0.006473082,29.28924589,0.037814355,0.000748223,3.78143552
2023,Grass,M,G10,23.675,1.298455718,0.792875947,0.389370052,0.006722683,38.93700522,0.059524313,0.001901803,5.952431343
2023,Grass,N,G10,33,1.700104891,1.188883484,0.300699922,0.006774505,30.06999215,0.0469268,0.000696694,4.692679974
2023,Grass,O,G10,28.5,1.433316879,0.988515443,0.310330146,0.003647367,31.03301461,0.044360205,0.000478604,4.436020462
2023,Grass,P,G10,21.6,1.360114094,0.929185731,0.31683251,0.008191049,31.68325103,0.039198852,0.000696932,3.919885223
2023,Grass,Q,G10,27.8,0.838844559,0.907396444,0.33095523,0.008523915,33.09552297,0.050914293,0.000941494,5.091429348
2023,Grass,I,G09,22.2,1.439156805,0.991151477,0.311297091,0.002950095,31.12970912,0.04197563,0.000430674,4.197563041
2023,Grass,R,G09,15.35,1.222363411,0.78500572,0.357796779,0.003671483,35.77967788,0.045862987,0.001698696,4.586298666
2023,Grass,S,G09,18.4,0.604002205,0.430085164,0.287941069,0.007575563,28.79410694,0.041917962,0.001281963,4.191796204
2023,Grass,T,G09,24.4,1.052022336,0.750396126,0.286710842,0.00463547,28.67108422,0.041531799,0.000562119,4.15317993
2023,Grass,U,G09,18,1.53111317,1.007597998,0.341918013,0.013695912,34.19180129,0.04745758,0.001619531,4.745758039
2023,Grass,V,G09,13,1.137166149,0.75702315,0.33428976,0.003627466,33.42897598,0.04973326,0.004652719,4.973326039
```

Yuck. Pretty difficult for a human to work with, but it's perfect for R.

Type the following code into your R script and run it.

Notice that:

- the file path needs quotation marks around it.
- RStudio automatically types a second, closing quotation mark after your cursor, just like it does for brackets.

Remember that our working directory is automatically set to the top-level `U:/`, so we'll need to tell R exactly where to look to find the file.

```
# Load data
data_area <- read.csv("LEC243/data/LEC243_2024_area.csv")
```

You should see the data pop up in your `Environment` panel.

> **Q:** Can you describe each part and what it does?

> **Q:** What happens if you deliberately add a typo into the file path telling R where to look for the .csv file?

> **Q:** What happens if you run the `read.csv()` function without the part of the code that creates and stores the object?

3.5. Data frames

Now you've loaded the area data into R, let's give a closer inspection.

R loads data as a specific type of object called a **data frame** that has *observations* (rows) of *variables* (columns). Each column has a name and the rows are numbered sequentially.

> **Q:** Look at the Environment panel. How many observations and variables does the area data frame have?

Click on the blue circle just to the left of the data frame in the `Environment` tab. It should drop down a preview of all the variables and their first few values.

Click on anywhere else on the data frame in the `Environment` tab. It should open a table in your scripts panel. Scroll down to examine the full table.

You can also examine data frames in R using the `head()` and `View()` functions.

Type the following code into your R script and run it.

```
head(data_area)
View(data_area)
```

To refer to a specific variable (column) in a data frame, we use the symbol `$`.

Type the following code into your R script and run it. Practice using the autocomplete feature to select the data frame and the variable name.

```
data_area$TotalCampusArea_m2
```

> **Q:** What happens? Why might this be useful as you are coding? Is there anything else that you notice about this data?

Now add the `print()` function around the previous code and run it again:

```
print(data_area$TotalCampusArea_m2)
```

> **Q:** What happens? Are there any differences to what happened previously?

> While you are using R line-by-line, if you run the name of an object, it will automatically print the stored value of that object in the Console, as if you had used the `print()` function.
>
> However, there are certain times (e.g. when running the a script from the command line, or even automatically!) when just running the name of the object will **not** print it. Therefore, it is generally a good idea to *explicitly* print values that you want to show up in the console as part of your analysis.

Nonetheless, it can be very quick and convenient to simply highlight an object and run it if you need to see what its contents are while you are still creating your analysis.

3.6. Basic calculations

Now we'll repeat a few of the calculations that we did in Excel earlier in R.

Unfortunately, most functions have different names in Excel and R.
You will probably get them confused - I still frequently get `mean()` and `average()` confused in both programs!

That's completely okay - if you use the wrong function, you should get an error message telling you that the function does not exist, and you can change your code.

If you can't remember the function you need, google (or other search engine) is your friend.

The R functions you'll need to use are:

- *average*: `mean()`
- *standard deviation*: `sd()`
- *length of an object*: `length()`
- *square root*: `sqrt()`

Type the following code into your R script and run it. Remember to use the auto-complete feature again!

```
# Basic calculations
mean(data_area$TotalCampusArea_m2)
sd(data_area$TotalCampusArea_m2)
length(data_area$TotalCampusArea_m2)
sd(data_area$TotalCampusArea_m2) / sqrt(length(data_area$TotalCampusArea_m2))
```

Q: What does the output look like? Are you surprised?

It turns out that R automatically replaces any missing values in a `.csv` file with the value `NA`, and if you try to perform a calculation with an `NA` value, R will simply return the value `NA`. Whoops!

Luckily there is an easy way to remove NA value from a data frame.

Type the following code into your R script (above the previous calculations) and run it. Then re-run your calculations from before.

```
# Remove NAs
data_area <- na.omit(data_area)
```

Now what does the calculation output look like?

Notice that we have replaced the old object `data_area` with a new version of it without the NAs - we actually over-wrote our data!

That seems kinda bad, right?

Importantly, we are not changing the original data stored in the .csv file - we are only changing the version stored in our R environment.

The original data (with NAs) still exist, and if we want to see them again, all we have to do is re-run the line of code that loads the data from the .csv file.

This is a big reason why using code-based applications like R is good for reproducible and open science - anyone can re-run your analysis starting from the original data and they should get the same answer!

Q: If you wanted keep the old data (with NAs) and create a new version of it (without NAs) so you could compare them, how would you do that?

3.7. Help and documentation

Type the following code into your R script and run it.

```
?sd
```

Q: What happened? Is there anything that you notice as you read the help file?

It turns out there is another way to remove NAs from some calculations - you can do it in the function with the `na.rm` argument. (There are often multiple ways to do things in R.)

Reload our data from the .csv file - remember we didn't change the original data, just our version of it stored in the R environment!

See what happens if you use this other method for ignoring NAs in the function. Copy the first three lines of your old code and modify it as shown below.

```
# Basic calculations
mean(data_area$TotalCampusArea_m2, na.rm = TRUE)
sd(data_area$TotalCampusArea_m2, na.rm = TRUE)
length(data_area$TotalCampusArea_m2, na.rm = TRUE)
```

Q: What happened? Why do you think it didn't work for all of the functions?

Up until now, we've only given our functions a single, unnamed argument (a file, data frame, or number) for it to do something with.

Here, we are using a second, named argument with an equal sign (`na.rm`) to change the settings for what the function does with the data we provide as the first argument (i.e., ignore the NAs or not).

Many functions have settings like this that you can toggle on or off by providing an argument that is either `TRUE` or `FALSE`.

You can also search for functions directly in the Help tab's search bar.

Search for `read.csv` and skim the documentation.

Q: How many different arguments can `read.csv()` take? (Your answer can be approximate.)

You can see how this Help tab and documentation can be really helpful for more complex functions!

3.8. Tidying up

Before you finish today, let's tidy up the R script.

First, you'll want to add a comment header to the very top of your document describing what the script is, who wrote it, and when they they wrote it. Mine usually look something like this:

```
##### Marta Shocket
##### LEC 243 - Week 3 Practical
##### 21 Oct 2024
```

You should also:

- make sure that all of your code is thoroughly commented, so that someone else (or you in 12 months) can understand what you did
- delete spare lines of code that you don't need anymore

Make sure that you have saved your R script! Remember, it will automatically be stored on the R server and available the next time you log in.

When you close RStudio, it will ask if you want to save your workspace image to your `U:/`. This refers to all the objects you have stored in your environment. **Select 'Don't save'** - because we saved our data and scripts, we can easily recreate the workspace.

Here's what my final script looks like, with extra comments and formatting to make it easier to understand:

```
##### Marta Shocket
##### LEC 243 - Week 3 Practical
##### 21 Oct 2024


# Practice code
a <- 3
b <- 5
a + b
a^2
b*a
b**a
z <- b/a
z
1:5
z <- c(1, 2, 3, 4, 5)
z

# Load data
data_area <- read.csv("LEC-243/data/LEC243_2024_area.csv")

# View data
head(data_area)
View(data_area)
data_area

### Remove NAs
data_area <- na.omit(data_area)

### Basic calculations on total campus area
# Mean
mean(data_area$TotalCampusArea_m2)

# Standard deviation
sd(data_area$TotalCampusArea_m2)

# Number of observations
length(data_area$TotalCampusArea_m2)

# Standard error
sd(data_area$TotalCampusArea_m2) / sqrt(length(data_area$TotalCampusArea_m2))

### Alternate code for removing NAs
# This method doesn't work for length(),
# so I removed NAs from the full dataset instead
mean(data_area$TotalCampusArea_m2, na.rm = TRUE)
sd(data_area$TotalCampusArea_m2, na.rm = TRUE)
```

3.9. Glossary of R functions

The final task for today is to create a glossary of R functions that you'll update at the end of each practical.

This glossary will provide a reference sheet that you can use during this module, while preparing for the end-of-module exam, and during the rest of your studies.

Steps to create your glossary:

1. In RStudio, go to File > New File and select Markdown File. A markdown file is a type of simple text file that you can edit and read as a tab in RStudio, which will make it easy to reference as you work.

2. Save the file in your LEC243 folder as `RGlossary.md`.

3. Write "Glossary of R functions" (or your preferred title) at the top.

4. Make an entries for each function that we used today. I'll post a list of the new functions each week, in alphabetical order - but it is up to you how to best describe them in your own words.

Some may only need a few words (e.g. "calculates a mean") whereas for others you might want to add more information. You can also update your glossary to contain more information if you encounter a problem while using a function later on.

The functions for week 3 are:

- `?`
- `getwd()`
- `head()`
- `length()`
- `mean()`
- `na.omit()`
- `print()`
- `read.csv()`
- `sd()`
- `sqrt()`
- `View()`