Getting Started in R

Objectives:

1. Download R and RStudio on your personal computer.
2. Create a single-value object, a vector, and a data frame.
3. Use number and character data.
4. Produce a Quarto report in PDF format to save a record of your work.

Install R and RStudio onto your computer

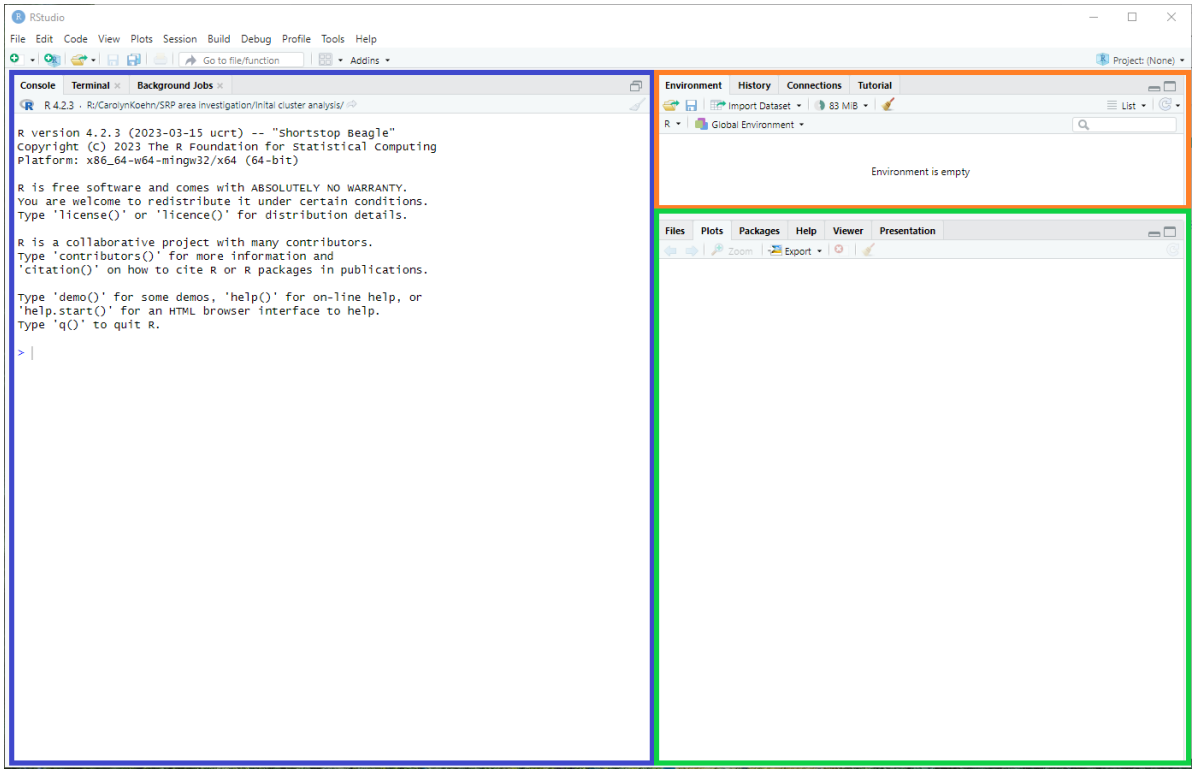
(1) Follow the instructions for "Install R and RStudio" at https://datacarpentry.org/R-ecology-lesson/index.html.

A free, online version of R can be used at posit.cloud. This is a backup solution if you cannot download this software to your computer. However, you are limited to 25 hours of use a month and will have limited processing power. If you use posit.cloud, make sure to log out when you are done working, or you will burn hours!

(2) Launch RStudio to make sure it is working.

Orienting yourself in RStudio

You should see a workspace for coding in R that looks similar to the image below.



The blue box is the console, where code is actually run. You can use the console as a simple calculator, for example. (3) Try typing `1+1` in the console next to the `>` arrow and pressing enter. The output will be `[1] 2`. We can ignore the number in brackets to see that the answer is 2.

The orange box is the Environment. The environment shows which datasets you have loaded into your current session. It also has tabs for History, Connections, and Tutorial. We will not be using those tabs.

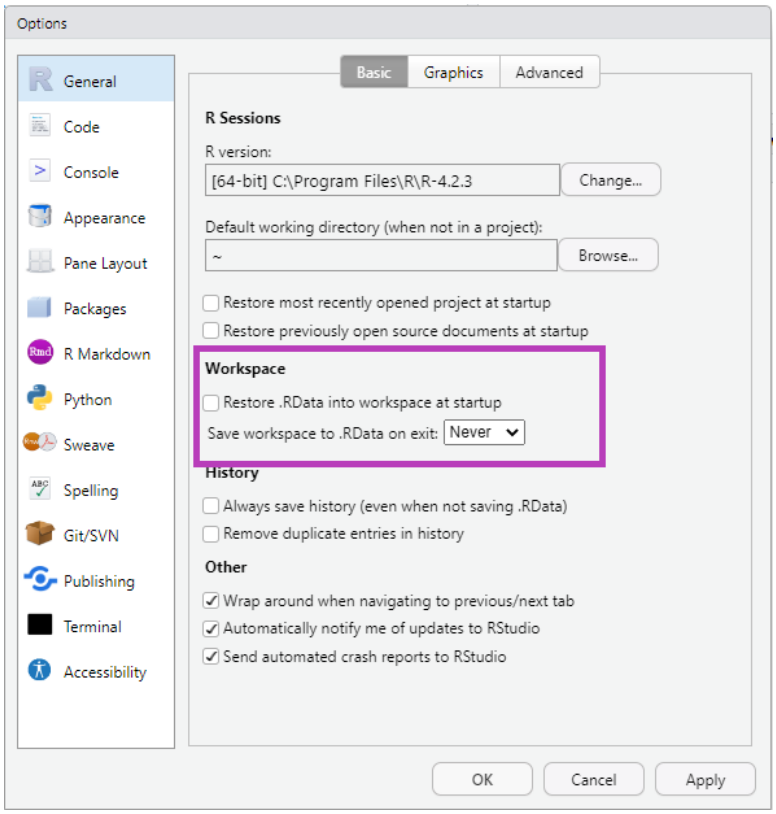
The green box shows the files in your current folder (similar to the file explorer on your computer). This is also where any plots you make or help files you look up will be shown. You can switch between file view and other views using the tabs at the top of this panel.

When you create a file, it will appear as a panel in the top left corner above the console. We will do most of our work within files because they allow us to save our work easily.

Getting set up for this semester's project

> 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 resuming work after a break. (4) Go through the steps for creating an "R Project" for this tutorial below.
>
> 1. Start RStudio.
> 2. Under the `File` menu, click on `New Project`. Choose `New Directory`, then `New Project`.
> 3. Enter a name (for example, "VIP-fall-2023") for this new folder/directory, and choose a convenient location for it (Desktop, school files folder, etc.). This will be your **working directory** for the rest of the semester.
> 4. Click on `Create Project`.
>
> (source: Data Carpentry Data Analysis and Visualization in R for Ecologists)

We are going to save our work by creating files, so we do not want R to save another record of all the code we run, which is a default setting in RStudio. (5) Under the `Tools` menu, click `Global Options`. Uncheck the box next to `Restore .RData into workspace at startup` and change the `Save workspace to .RData on exit` to `Never`. Accessibility options for RStudio are available in this menu at the bottom of the left panel.



It will also be helpful to create a folder to hold data. (6) In the `Files` panel in the bottom right corner, click `New Folder` and give it the name `data`.

Now, every time you work in `R` this semester, you can open this project by double-clicking the R project you've created here in your file explorer. You can also do this by opening RStudio, clicking the `Project: (None)` button in the upper right corner, choose `Open Project...`, and then find and select the R project file you just created. If you do one of these methods every time you work on this project, you should not run into pesky file pathway errors! (7) Practice opening the project file by closing RStudio now and re-opening it using one of the two methods described above.

Documenting your code

Reproducibility is an essential part of the scientific research process. Other scientists should be able to replicate your study and check your work. We can do reproducible data analysis by saving our code in documents or "scripts." This is *especially* important in VIP, as the code you write and your results may be part of published articles, handbooks for land planners, and references or work to build on for future VIP students. However, the scientist most supported

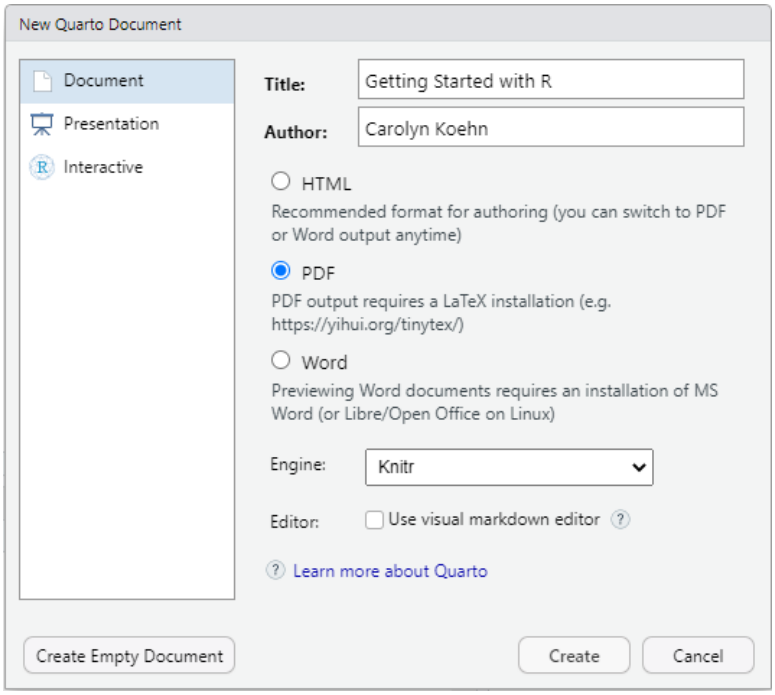
by reproducible coding is yourself! You will want to have a record of what you've done so you can go back and modify existing code or write new code based on work you've already done.

This semester, you are going to create a PDF report of the work you do for each module showing all the code you wrote and all your results.

Create a Quarto document

(8) In the `File` menu, select `New File` and then `Quarto Document`. You will see a set-up menu to help you create a document where you can write code, show code output, and present all your results in a PDF.

(9) Give your new Quarto document a title, add your name as the author, select the button next to the PDF option, and uncheck the "Use visual markdown editor" box if it is checked. Then select `Create Empty Document`. Your set-up menu should look like the image below:



Edit a Quarto document

Now that you have a document, you can start to write code that is saved. A general rule is that any code you want to save should go in the document, while any code you do not want to save can be run in the Console (as you tried earlier when you used the console as a calculator).

(10) In the *Console*, type the following and hit enter:

```
install.packages('rmarkdown')
```

(11) Next, in the Console panel, switch to the **Terminal** tab. In the Terminal, run the following code:

```
quarto install tinytex
```

These two lines of code install packages that will let you turn this document into a PDF. Since you only need to install packages once, we do not save this code in the document.

In a Quarto document, you can type out regular paragraphs, notes, and explanations that will be translated to the formatting you're looking at right now.

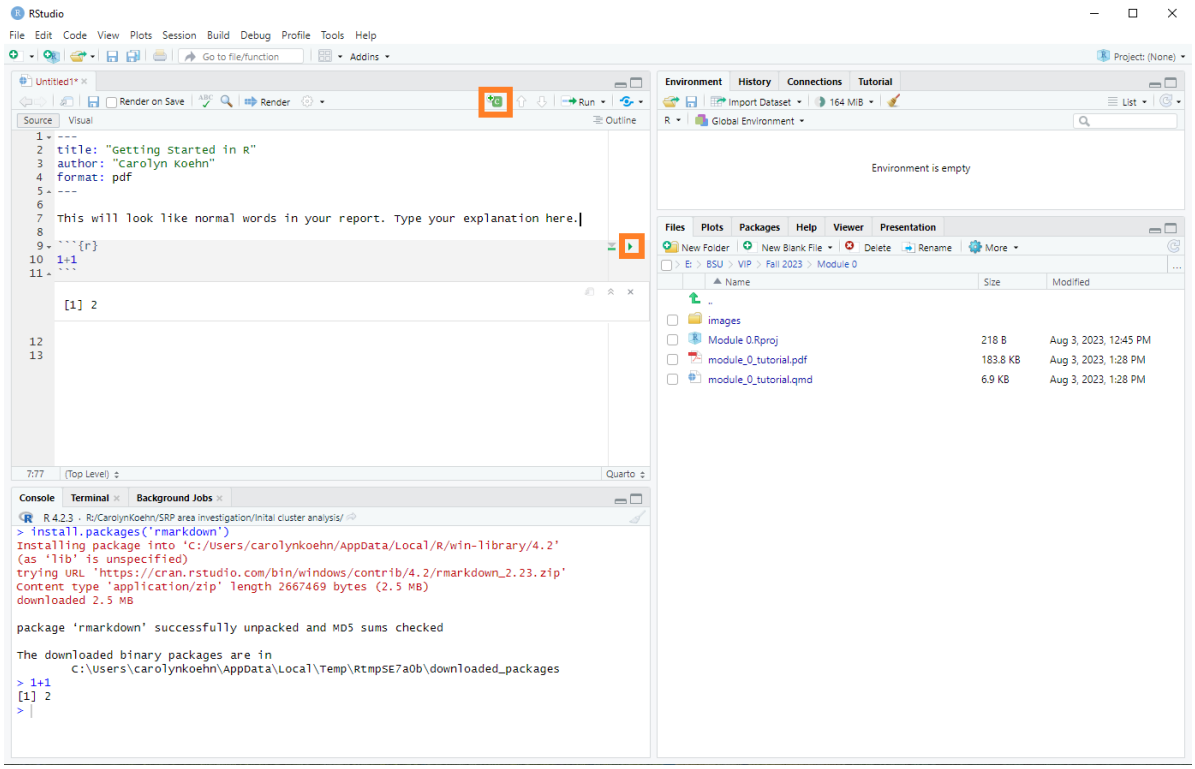
(12) Write a few sentences explaining why you are creating this document.

Next, let's run some code. **(13)** Hit the enter key twice to keep your notes and code visually separated. Then, click the icon in the top right corner of the document panel that looks like a C in a green box with a + next to it. This creates a new **code chunk**.

A code chunk always begins with three backticks, then the coding language in curly brackets. On the line under this, you can type your code. The code chunk ends with three more backticks. After the chunk is closed by these backticks, you can write normal text like you did at the top of this report. Code goes in the grey code chunks and regular text goes in the white space.

(14) Inside the code chunk, type `1+1` again. Then, press the green arrow in the top right of the code chunk to run your code. You will see the output below the code chunk and in the Console.

(15) Ensure that your RStudio document and Console look similar to the image below. The code chunk creator and the run code buttons are highlighted in orange.



Create the PDF

To turn a Quarto document into a PDF, you will need to **render** it. When you render a Quarto document, `R` will run all the code within it and produce a PDF with your writing, code, and code results. Make sure all the code you need is in the document itself. `R` will not remember anything you did in the Console.

(16) Test that your report will render. Click the `Render` button next to the blue arrow above your document. You may be prompted to save the file – make sure to give this document a meaningful name. Then, check the files panel on the bottom right to make sure that a PDF was created. Click that file to launch the PDF and see how it looks.

Basics of data in `R`

In order to analyze our data this semester, you will need to know a bit about how `R` handles data. When we do analysis, we assign *values* to *objects*. To create an object, you need to give it a name, then use the assignment operator `<-`, and then give the object a value. Use a name that makes sense so that you and I can understand your code!

(17) Create a new code chunk under the one you already have. Inside it, type

```
weight_kg <- 55
```

and hit the run arrow. You will not see any output, because all we have done is give `weight_kg` the value of `55`. Look in your Environment panel on the top right. You will see that `R` is remembering that `weight_kg` = `55`.

If you want to see output, you can `print` the value. **(18)** Create a new code chunk and run this code:

```
print(weight_kg)
```

```
[1] 55
```

You can do math with objects and create new objects out of them. **(19)** Try running this code in a code chunk, and then print the value of `weight_lb`.

```
weight_lb <- 2.2 * weight_kg
```

Doing math with objects instead of raw numbers is more reproducible. If I need to change the value of `weight_kg`, I only need to do it once and my code will automatically apply the change to the whole document. When we start working with code, this will allow us to change a data source if needed and still be able to use the same code.

(20) Challenge

Try to guess the ending values of `mass`, `age`, and `mass_index` without running the code below code (you can use a calculator). Then run the code and check your results by using `print()` or looking in the Environment panel. Write your guesses and the correct answers in your report *outside* of a grey code chunk.

```
mass <- 47.5
age  <- 122
mass <- mass * 2.0
age  <- age - 20
mass_index <- mass/age
```

Functions

Functions are bits of code that `R` already knows how to run. All you have to do is use the command that runs the code you need. You've already used one function: `print`. Anything the function needs to know to run its code goes into the `()` after the function name. For example, `print` needs to know *what* to print, so you have to put the object name inside the parentheses.

If you need help figuring out what a function does or what to put in the `()`, you can search the help files from the console. **(21)** Try typing this in the Console and pressing enter:

```
?abs
```

(22) Write one sentence on why you would use the `abs` function (what does it calculate?).

Sometimes functions need more than one input (also called an *argument*). When this happens, each argument has a name. You can use an `=` to assign a value to each argument. If you do not use the name of an argument and an `=`, the arguments *must* be in the same order as they are listed on the help file. It is best to use the names of the arguments whenever possible, because it ensures that `R` will do what you want it to do and it will be easier for you and others to read your code.

(23) Use `?log` in the Console to see the information about the `log` function. Create a new code chunk and use the following code to calculate $log_{10}(5)$ (the base 10 logarithm of 5).

```
log(x = 5, base = 10)
```

```
[1] 0.69897
```

(24) In the same code chunk, hit enter and calculate $log_8(16)$.

How do we look at data?

Most of the time, our data will be in a *vector* or a *data frame*. A vector is a group of values. Data frames are like Excel spreadsheets, but each column is its own vector. Most of your work this semester will be with data frames and vectors.

You can create your own vector with the function `c()` (the `c` stands for concatenate). All the values in the vector go inside the `()`. Values inside vectors do not need argument names. **(25)** Create a new code chunk and run this code to create a vector and print it.

```r
weight_g <- c(100, 450, 5000)
print(weight_g)
```

```
[1]  100  450 5000
```

A vector can also have letters (called *characters*). **(26)** Create a new code chunk and run this code.

```r
animals <- c("mouse", "rat", "dog")
print(animals)
```

```
[1] "mouse" "rat"   "dog"
```

The quotes are essential for characters. If you don't use them, `R` will look for *objects* called `mouse`, `rat`, and `dog`.

(27) Create a vector called `animal_names` in a new code chunk. Choose three names, one for each animal, and put them inside the vector. Print the vector.

(28) Lastly, create a data frame from these vectors and print it using this code:

```r
animal_weight <- data.frame(animals, weight_g, animal_names)
print(animal_weight)
```

```
  animals weight_g animal_names
1   mouse      100       Mickey
2     rat      450         Remy
3     dog     5000        Goofy
```

Now each animal is associated with its weight. **(29)** Take a look at how `R` prints the data frame. This is how `R` would read an Excel spreadsheet that looked like this:

	A	B	C
1	animals	weight_g	animal_names
2	mouse	100	Mickey
3	rat	450	Remy
4	dog	5000	Goofy

(30) Finally, check the Environment panel to see how the different types of objects appear there.

Finishing up

(31) Go back through your report and write (in the "white space" of the document between code chunks) what each code chunk is doing in your own words. These explanations don't need to be long, but they should be clear enough that if you come back to this report in one year, you should be able to easily understand the purpose of each code chunk. Then, **(32)** render your report to PDF and **(33)** email it to: [INSERT EMAIL HERE].

Statement of original and referenced work:

This document is original work authored by Carolyn Koehn, created while consulting the Data Carpentry R for Ecology lesson, which is licensed under a Creative Commons Attribution 4.0 International License, as a guide. The order of topics, the "Getting set up for this semester's project" section, the "Basics of data in R" activities, and the "How do we look at data?" activities are directly from the Data Carpentry lesson. The rest of the module is Carolyn Koehn's original work.

License