Notes edited from Data Carpentries Intro to R for Social Scientists

Getting set up

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. Relative paths 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 makes it a lot easier to move your project around on your computer and share it with others without having to directly modify file paths in the individual scripts.

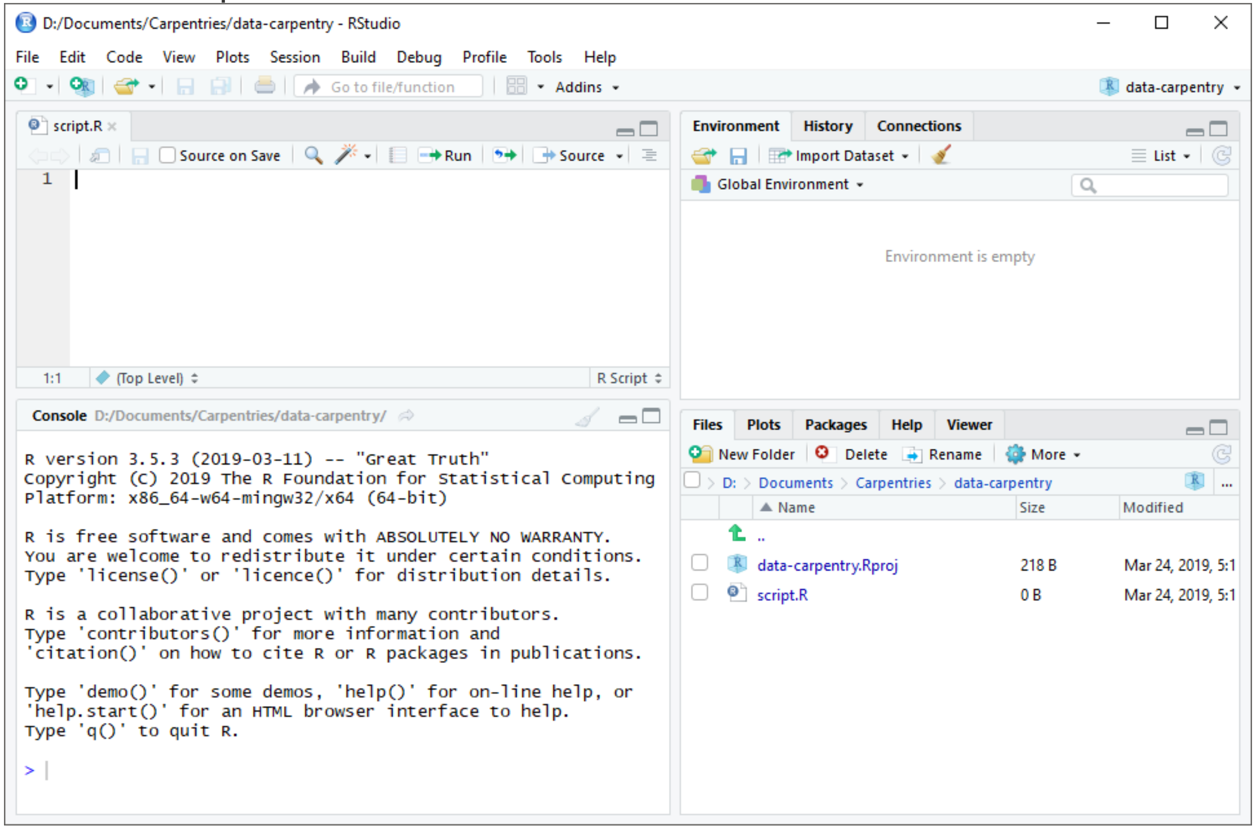
RStudio provides a helpful set of tools to do this through its "Projects" interface, which not only creates a working directory for you but also remembers its location (allowing you to quickly navigate to it). The interface also (optionally) preserves custom settings and open files to make it easier to resume work after a break.

Create a new project

- Under the `File` menu, click on `New project`, choose `New directory`, then `New project`
- Enter a name for this new folder (or "directory") and choose a convenient location for it. This will be your **working directory** for the rest of the day (e.g., `~/data-carpentry`)
- Click on `Create project`
- Create a new file where we will type our scripts. Go to File > New File > R script. Click the save icon on your toolbar and save your script as "`script.R`".

The RStudio Interface

Let’s take a quick tour of RStudio.


D:/Documents/Carpentries/data-carpentry - RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
data-carpentry
script.R
Source on Save
Run
Source
(Top Level)
R Script
Environment History Connections
Import Dataset
List
Global Environment
Environment is empty
Console D:/Documents/Carpentries/data-carpentry/
Files Plots Packages Help Viewer
New Folder Delete Rename More
D: Documents Carpentries data-carpentry
Name Size Modified
data-carpentry.Rproj 218 B Mar 24, 2019, 5:1
script.R 0 B Mar 24, 2019, 5:1

```
R version 3.5.3 (2019-03-11) -- "Great Truth"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

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.

>
```

RStudio is divided into four “panes”. The placement of these panes and their content can be customized (see menu, Tools -> Global Options -> Pane Layout).

The Default Layout is:

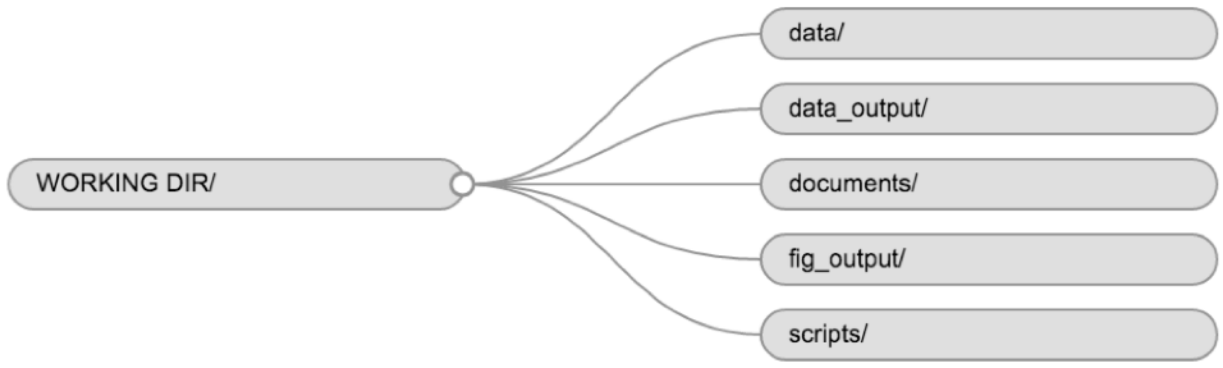
- Top Left - **Source**: your scripts and documents
- Bottom Left - **Console**: what R would look and be like without RStudio
- Top Right - **Enviornment/History**: look here to see what you have done
- Bottom Right - **Files** and more: see the contents of the project/ working directory here, like your Script.R file

Organizing your working directory

Using a consistent folder structure across your projects will help keep things organized and make it easy to find/file things in the future. This can be especially helpful when you have multiple projects. In general, you might create directories (folders) for **scripts**, **data**, and **documents**. Here are some examples of suggested directories:

- `data/` Use this folder to store your raw data and intermediate datasets. For the sake of transparency and provenance, you should *always* keep a copy of your raw data accessible and do as much of your data cleanup and preprocessing programmatically (i.e., with scripts, rather than manually) as possible.
- `data_output/` When you need to modify your raw data, it might be useful to store the modified versions of the datasets in a different folder.
- `documents/` Used for outlines, drafts, and other text.
- `fig_output/` This folder can store the graphics that are generated by your scripts.
- `scripts/` A place to keep your R scripts for different analyses or plotting.

You may want additional directories or subdirectories depending on your project needs, but these should form the backbone of your working directory.



The working directory

The working directory is an important concept to understand. It is the place where R will look for and save files. When you write code for your project, your scripts should refer to files in relation to the root of your working directory and only to files within this structure.

Using RStudio projects makes this easy and ensures that your working directory is set up properly. If you need to check it, you can use `getwd()`.

Downloading the data and getting set up

For this lesson we will use the following folders in our working directory: `data/`, `data_output/` and `fig_output/`. Let's write them all in lowercase to be consistent. We can create them using the RStudio interface by clicking on the "New Folder" button in the file pane (bottom right), or directly from R by typing at console:

```
dir.create("data")
dir.create("data_output")
dir.create("fig_output")
```

The direct download link for our data is: https://ndownloader.figshare.com/files/11492171. Place this downloaded file in the `data/` you just created. You can do this directly from R by copying and pasting this in your terminal:

```
download.file("https://ndownloader.figshare.com/files/11492171",
              "data/SAFI_clean.csv", mode = "wb")
```

Interacting with R

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

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

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

RStudio allows you to execute commands directly from the script editor by using the `Ctrl` + `Enter` shortcut (on Mac, `Cmd` + `Return` will work). The command on the current line in the script (indicated by the cursor) or all of the commands in selected text will be sent to the console and executed when you press `Ctrl` + `Enter`. If there is information in the console you do not need anymore, you can clear it with `Ctrl` + `L`.

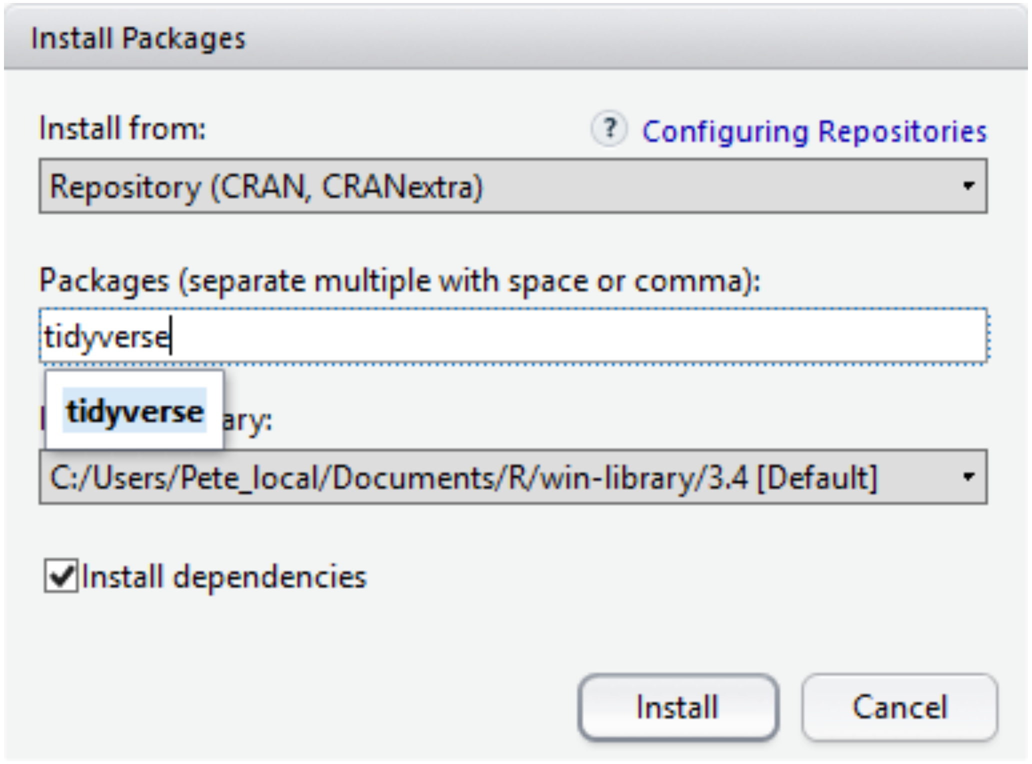
If R is ready to accept commands, the R console shows a `>` prompt. If R receives a command (by typing, copy-pasting, or sent from the script editor using `Ctrl` + `Enter`), R will try to execute it and, when ready, will show the results and come back with a new `>` prompt to wait for new commands.

If R is still waiting for you to enter more text, the console will show a `+` prompt. It means that you haven't finished entering a complete command. This is likely because you have not 'closed' a parenthesis or quotation, i.e. you don't have the same number of left-parentheses as right-parentheses or the same number of opening and closing quotation marks. When this happens, and you thought you finished typing your command, click inside the console window and press `Esc`; this will cancel the incomplete command and return you to the `>` prompt. You can then proofread the command(s) you entered and correct the error.

Installing additional packages using the packages tab

In addition to the core R installation, there are in excess of 10,000 additional packages which can be used to extend the functionality of R. Many of these have been written by R users and have been made available in central repositories, like the one hosted at CRAN, for anyone to download and install into their own R environment. In the course of this lesson we will be making use of several of these packages, such as 'ggplot2' and 'dplyr'.

Additional packages can be installed from the 'packages' tab. On the packages tab, click the 'Install' icon and start typing the name of the package you want in the text box. As you type, packages matching your starting characters will be displayed in a drop-down list so that you can select them.


Install Packages

Install from: Repository (CRAN, CRANextra)

Packages (separate multiple with space or comma): tidyverse

Install to Library: C:/Users/Pete_local/Documents/R/win-library/3.4 [Default]

☑ Install dependencies

At the bottom of the Install Packages window is a check box to 'Install' dependencies. This is ticked by default, which is usually what you want. Packages can (and do) make use of functionality built into other packages, so for the functionality contained in the package you are installing to work properly, there may be other packages which have to be installed with them. The 'Install dependencies' option makes sure that this happens.

Installing additional packages using R code

If you were watching the console window when you started the install of 'tidyverse', you may have noticed that the line

```
install.packages("tidyverse")
```

was written to the console before the start of the installation messages.

You could also have installed the `tidyverse` packages by running this command directly at the R terminal.