DATA CARPENTRY

[*alpha* version]

DAY 1
Welcome!

Data Skills Workforce Development

Data Carpentry for Social Sciences Curriculum

Introduction to good data practices using **R/ RStudio**

Demonstrator: Robert Nagy

(PhD student)

The University of Edinburgh – Cancer Research UK Edinburgh Centre
+ Western General Hospital - Edinburgh Cancer Centre
+ Edinburgh BioQuarter – Edinburgh Health Economics Team

[**web:** https://www.ed.ac.uk/profile/robert-nagy]

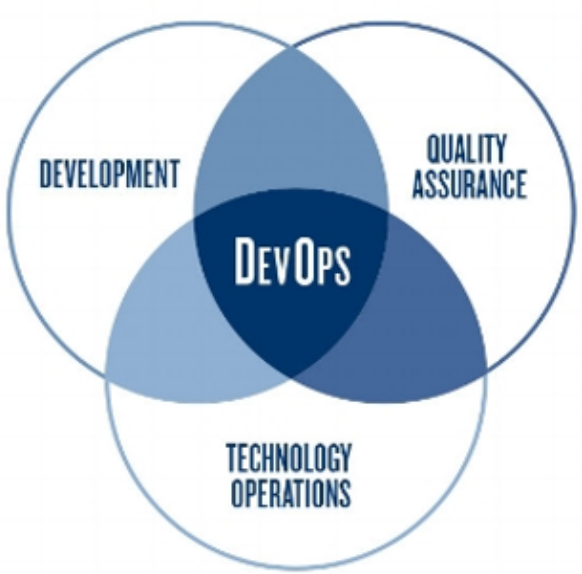
robert.nagy@ed.ac.uk

Lesson #1 – **'Before we start…'** https://datacarpentry.org/R-ecology-lesson/00-before-we-start.html

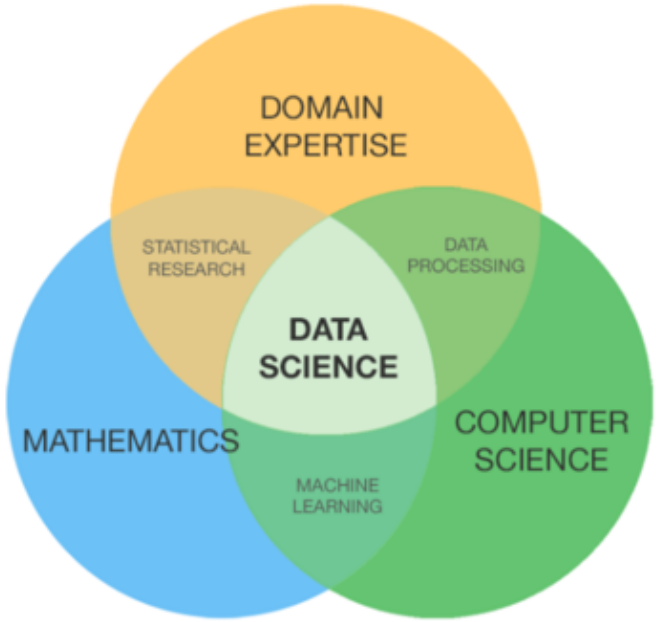
Learning Objectives

- Describe the purpose of the RStudio Script, Console, Environment, and Plots panes
- Organize files and directories for a set of analyses as an R Project, and understand the purpose of the working directory
- Use the built-in RStudio help interface to search for more information on R functions
- Demonstrate how to provide sufficient information for troubleshooting with the R user community

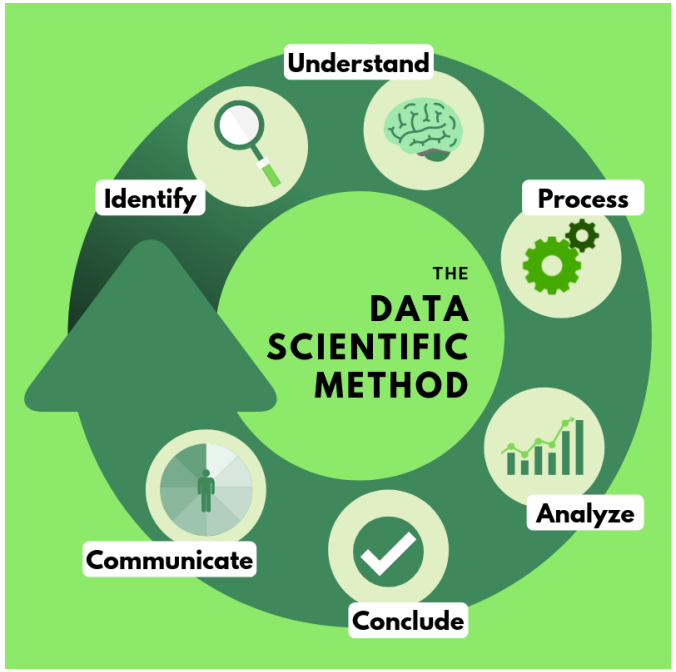
Lesson #1 – **'Ontology' of Data Science (the boarder scope..)**



+

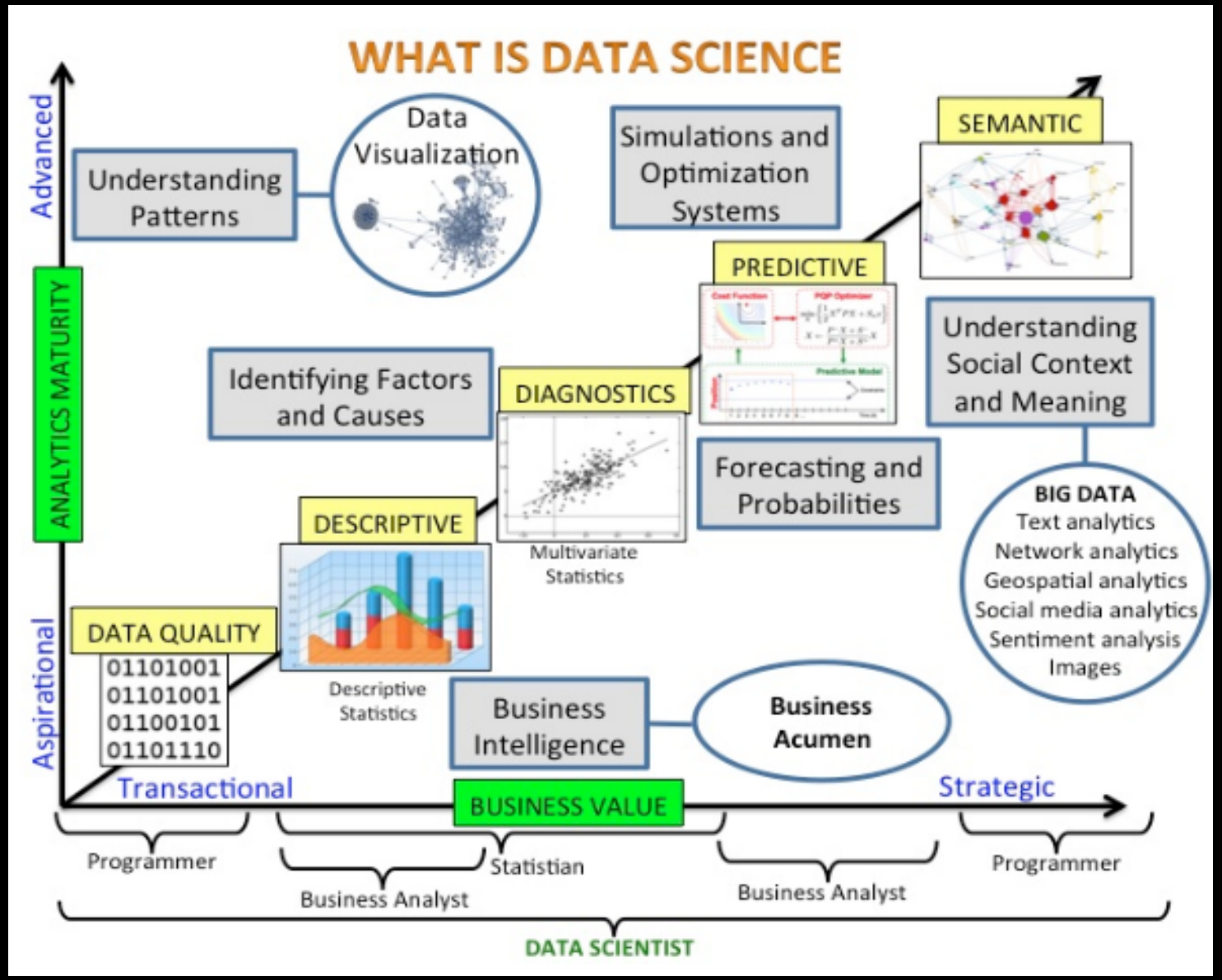


=



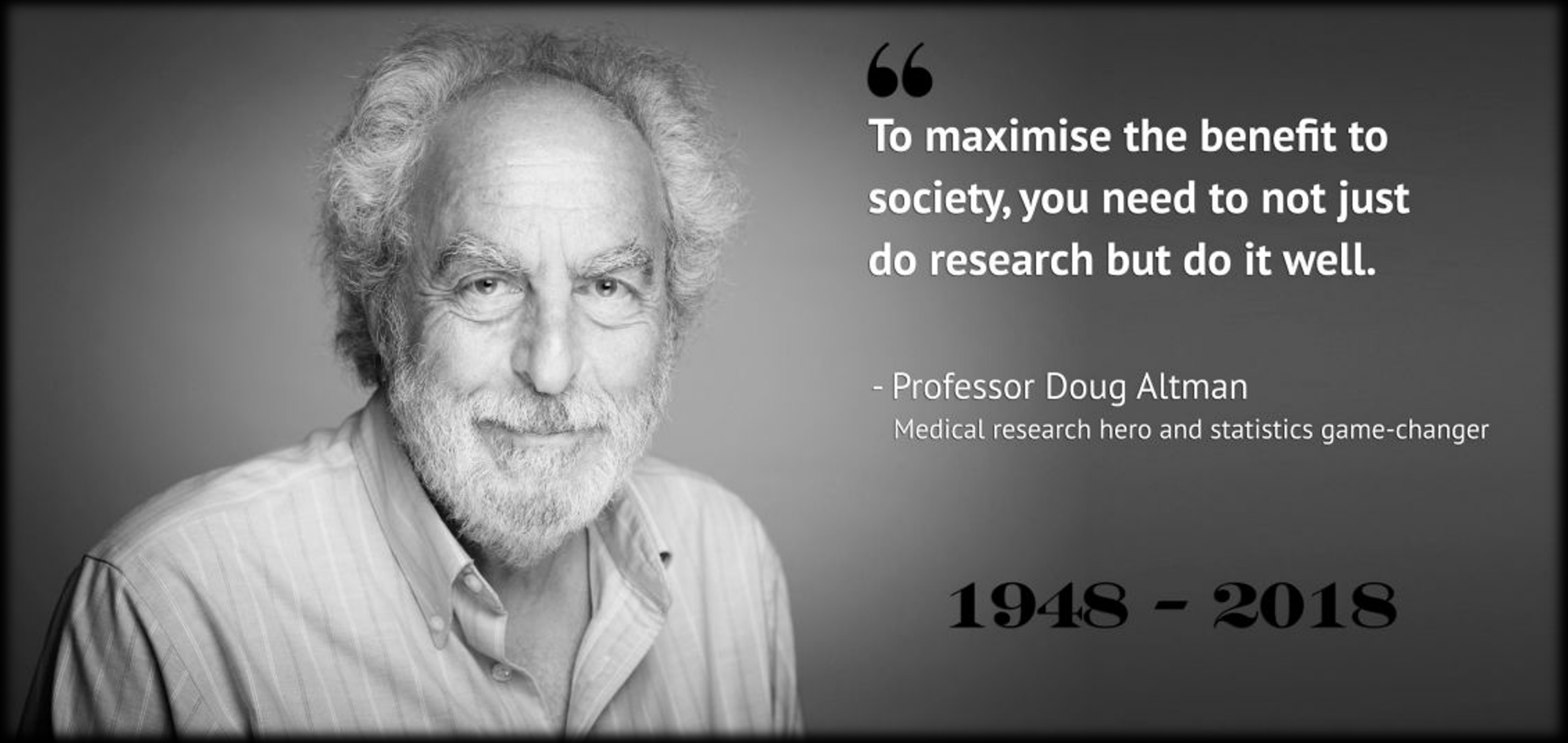
[source: https://towardsdatascience.com]

[**Keywords**: formal ontology; hermeneutic cycle; epistemology; knowledge/ data/ information re-cycling]



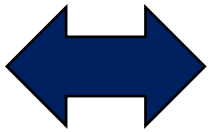
A robust , powerful and compact purpose-built tool for statistical computing and data visualisation that is self-containing/ standalone and optimised for vector operations.

[Source: https://www.datasciencecentral.com/profiles/blogs/data-science-summarized-in-one-picture



Lesson #1 – **Why R …?**

A complex decision: guarantee, reliability, standardisation, price, flexibility, …etc. considerations

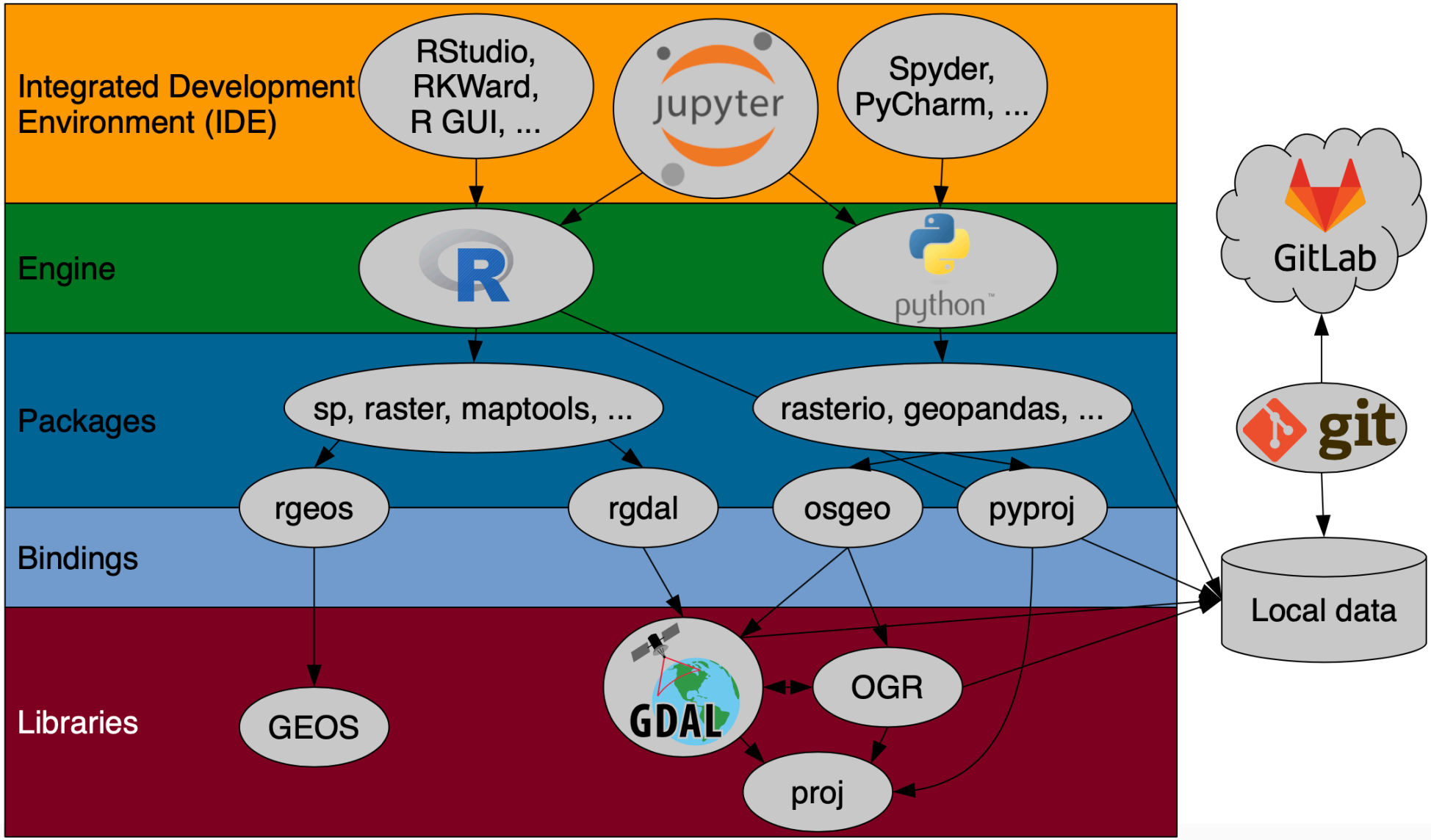


R is integral part of the Data Science workflow

- Raw data collection/ entry, preparation and formatting (e.g. Spreadsheets)
- Data cleansing and management (e.g. OpenRefine)
- Data analysis, visualization and publishing (e.g. R, Python, SQL)

Lesson #1 – **Why R …?**

Lesson #1 – **Systemic overview** [an illustrative example]

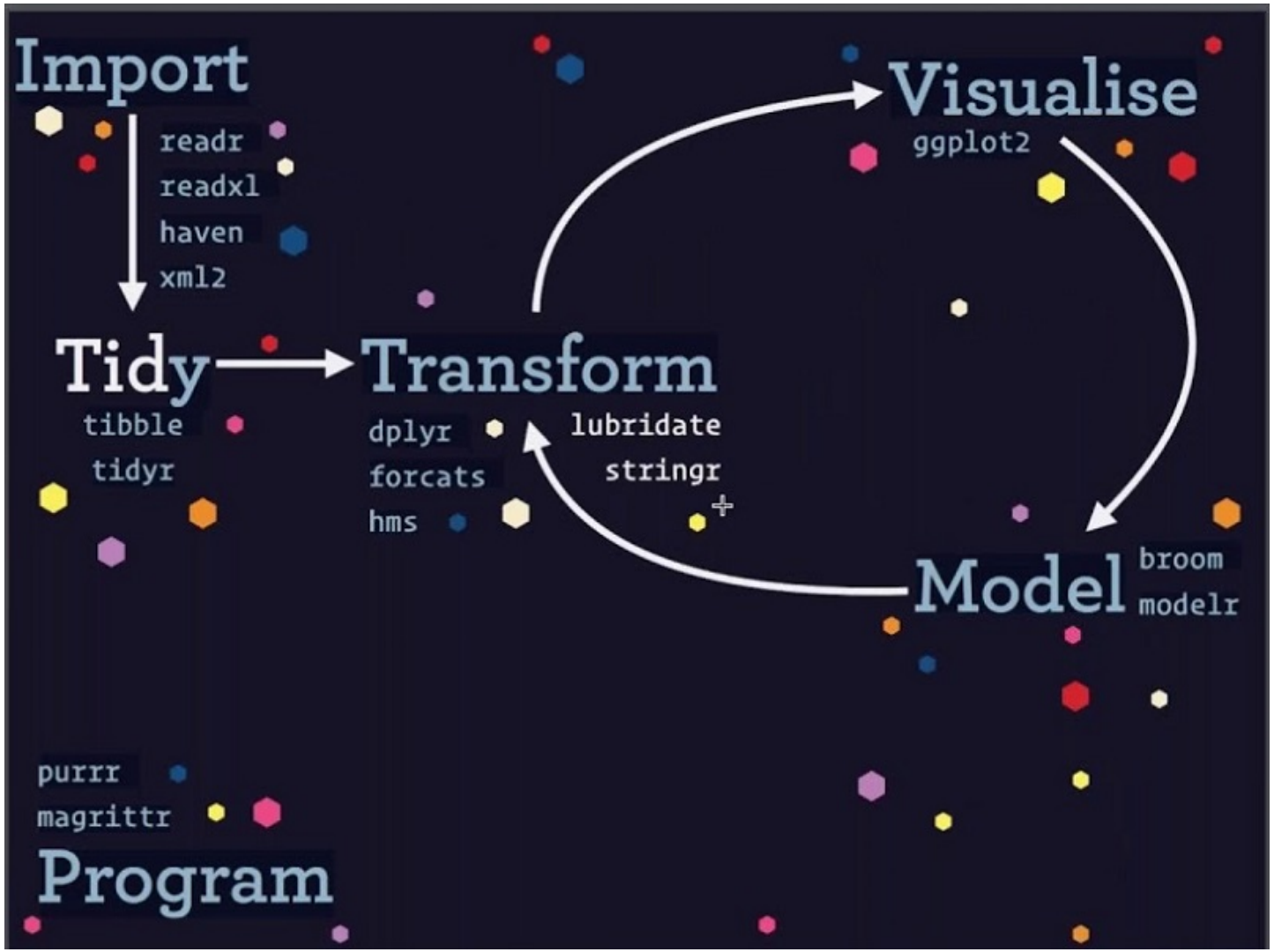


Lesson #1 – **The R syntax..** (‘The grammar of a programming language’)

“R is a high-level programming language used primarily for statistical computing and graphics. R does not have any well defined coding recommendations or de facto standards. This style guide provides some recommendations based on personal experience and expert opinions.”

Web: http://jef.works/R-style-guide/

Lesson #1 – **Tidyverse ecosystem overview & core R packages..**



Core packages we are going to use..

library(gridExtra) # ggplot

library(hexbin) # ggplot

library(dbplyr) # R and databases

library(RSQLite) # R and databases

library(tidyverse) # lesson 3 onwards

library(lubridate)

library(readr)

library(ggplot2)

library(dplyr)

library(magrittr)

library(tidyr)

Lesson #1 – **Working with data - fundamental concepts / elements..**

Introductory statistical package ‘mosaic’ overview > https://github.com/mlaviolet/Mosaic-cheatsheets/raw/master/mosaic-cheatsheet-gf.pdf

Base-R instructions > http://github.com/rstudio/cheatsheets/raw/master/base-r.pdf

ggplot data visualisation > https://github.com/rstudio/cheatsheets/raw/master/data-visualization-2.1.pdf

factors with ‘forcats’ > https://github.com/rstudio/cheatsheets/raw/master/factors.pdf

Dates and times > https://github.com/rstudio/cheatsheets/raw/master/lubridate.pdf

Working with strings > https://github.com/rstudio/cheatsheets/raw/master/strings.pdf

Data/ file import > https://github.com/rstudio/cheatsheets/raw/master/data-import.pdf

Data transformation with ‘dplyr’ > https://github.com/rstudio/cheatsheets/raw/master/data-transformation.pdf

Lesson #1 – **Why learning R …?**

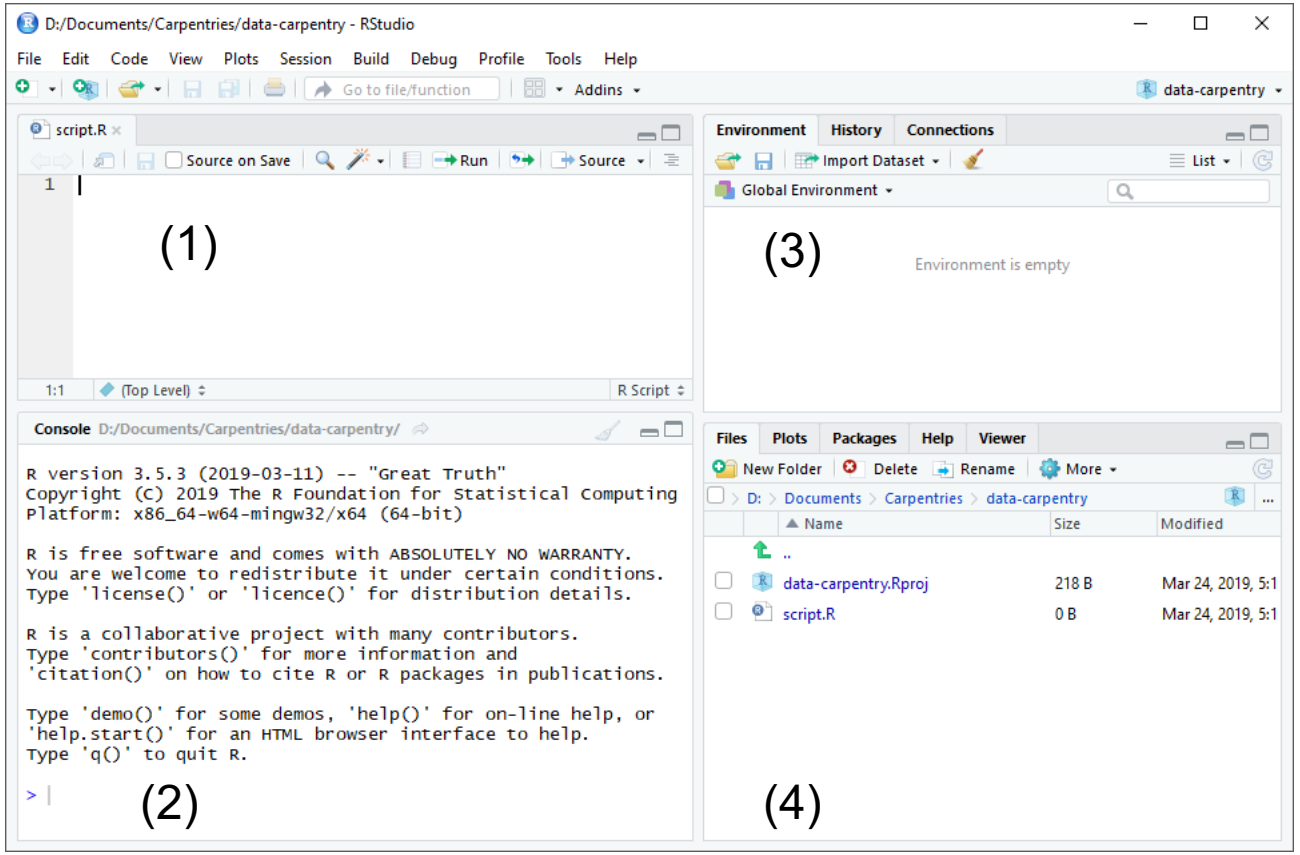
- R does not involve lots of pointing and clicking, and that's a good thing
- R code is great for reproducibility
- R is interdisciplinary and extensible
- R works on data of all shapes and sizes
- Advanced in handling missing values , date objects, etc.
- R produces high-quality graphics
- R has a large and welcoming community
- Not only is R free, but it is also open-source and cross-platform

Lesson #1 – The RStudio Script, Console, Environment, and Plots panes

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:

- **(1) Top Left** - *Source*: your scripts and documents
- **(2) Bottom Left** - *Console*: what R would look and be like without RStudio
- **(3) Top Right** - *Environment/ History*: look here to see what you have done
- **(4) Bottom Right** - *Files, plots, packages, documentation/ help, etc.*



RStudio’s default GUI arrangement on Windows 10.

Lesson #1 – Interacting with R/Studio

We call the instructions *commands* and we tell the computer to follow the instructions by *executing* (also called *running*) those commands.

Two main ways of interacting with R:

(1) by using the *Console*

- Commands can be directly typed and instantly executed (by pressing *Enter*)
- Also, results will be shown for executed commands
- Commands executed earlier on are preserved and can be recalled by pressing the *Up/ Down arrow* buttons

(2) by using *script files* (plain text files that contain your code). [*reproducibility considerations*]

- Because we want our code and workflow to be reproducible, it is better to type the commands we want in the *script editor* (*Source pane*), 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 us to *execute commands directly* from the script editor by pressing *Ctrl (Cmd)* + *Enter* key combination

Lesson #1 – Interacting with R/Studio (cont'd..)

- RStudio provides the **Ctrl + 1** and **Ctrl + 2** shortcuts allow you to jump between the *Script* and the *Console* panes

- If R is ready to accept commands, the R console shows a **>** prompt

- If R is still waiting for you to enter more data because it's incomplete yet, the console will show a *'**+**' prompt*.

- It means that you haven't finished entering a complete command. 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.

Lesson #1 – Setting up our working environment (*working directory*)

Working directory [*computing concept*] - The directory in which you are currently working. Path names that do not start with the root directory are assumed by the operating system to start from the working directory. 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.

Set and organizing your working directory

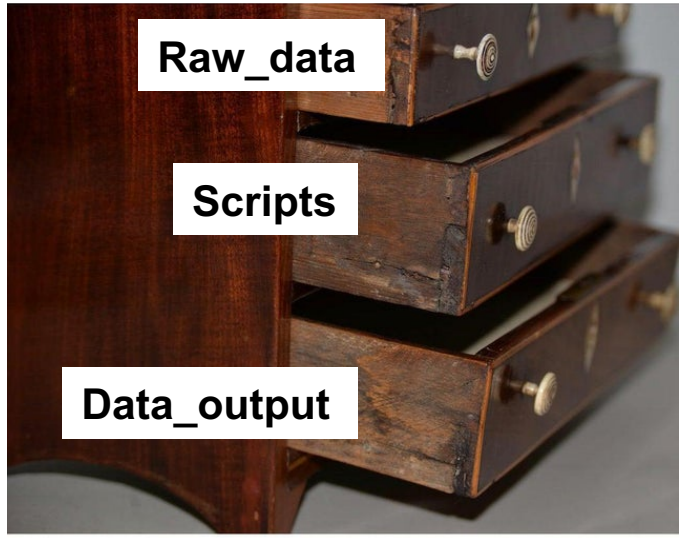
• **data/ raw_data.** Use this folder to store your raw data and intermediate datasets. For the sake of transparency and provenance, you should *always* keep a copy of your raw data accessible and do as much of your data cleanup and pre-processing programmatically (i.e., with scripts, rather than manually) 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.

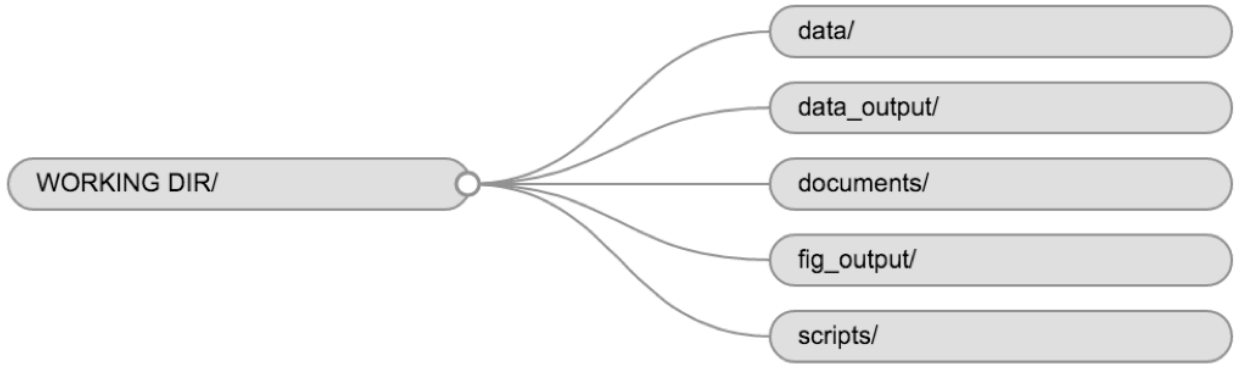
• **fig_output/** This folder can store the graphics that are generated by your scripts.

• **scripts/** A place to keep your R scripts for different analyses or plotting.



Lesson #1 – Setting up our working environment (*working directory tree concept*)

Set and organizing your working directory (cont'd…)



***Directory tree* structure abstraction that is consists of the working directory and subdirectories in a hierarchical arrangement.**

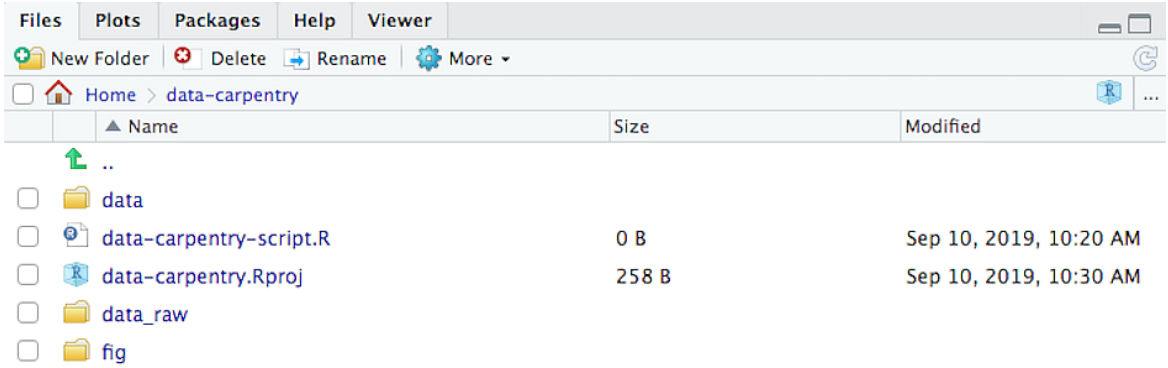
Creating the directory tree

Step #1: create/ set your working directory by creating your 'New Project'

Step #2. Add subdirectories either:

A. Through R commands

B. Via RStudio GUI



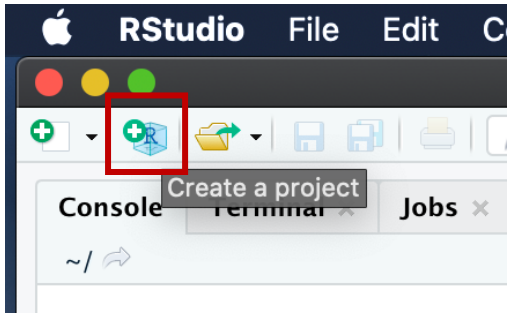
Lesson #1 – Setting up our working environment (Step #1: Create a *New Project*)

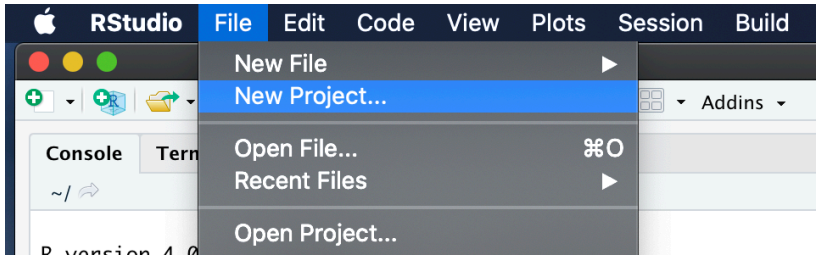
Using RStudio projects makes this easy and ensures that your working directory is set up properly.

1. Under the *File* menu, click on *New Project*. Choose *New Directory*, then *New Project*.

2. 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*).

3. Click on *Create Project*.





4. Download the *code handout*, place it in your *working directory* and rename it (e.g., *data-carpentry-script.R*). OR Create a *new R script* file.

5. (*Optional*) Set *Preferences* to '*Never*' save workspace in RStudio.

[* Code handout download URL: https://datacarpentry.org/R-ecology-lesson/code-handout.R]

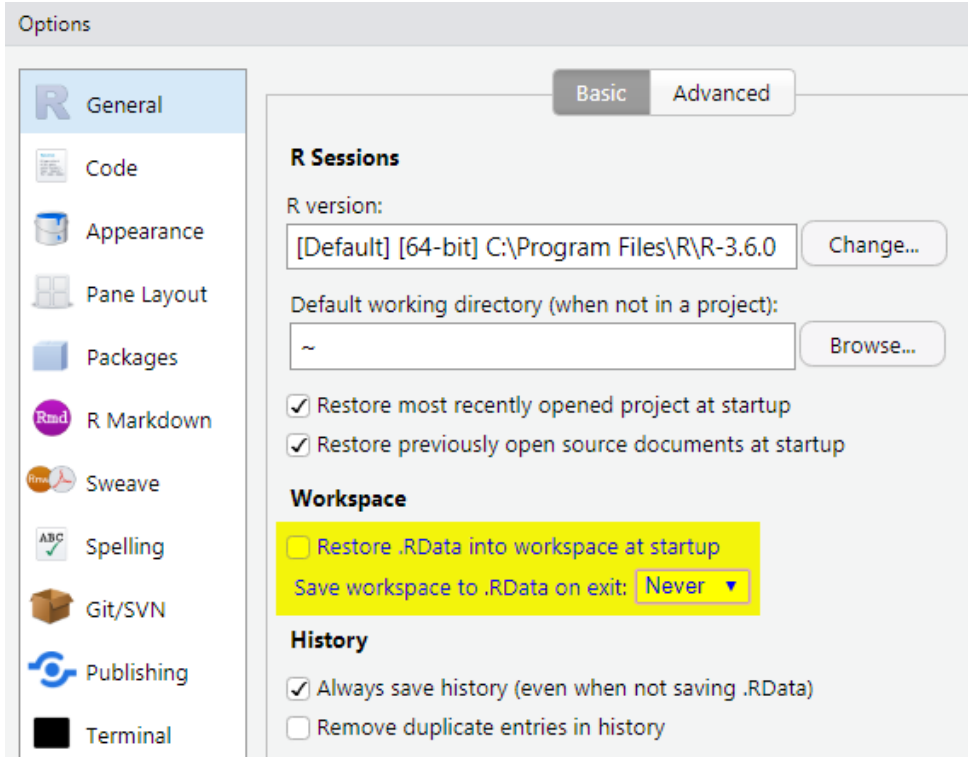
Lesson #1 – Setting up our working environment (Step #1: *Set up your workspace*)

Using RStudio projects makes this easy and ensures that your working directory is set up properly.

(*Optional*) Set *Preferences* to '*Never*' save workspace in RStudio.

A workspace is your current working environment in RStudio which includes any user-defined object.

- By default, all of these objects will be saved, and automatically loaded, when you reopen your project.
- Saving a workspace to .RData can be cumbersome, especially if you are working with larger datasets, and it can lead to hard to debug errors by having objects in memory you forgot you had.
- To turn this off, go to *Tools* –> '*Global Options*' and **select the 'Never' option for 'Save workspace to .RData' on exit.'**

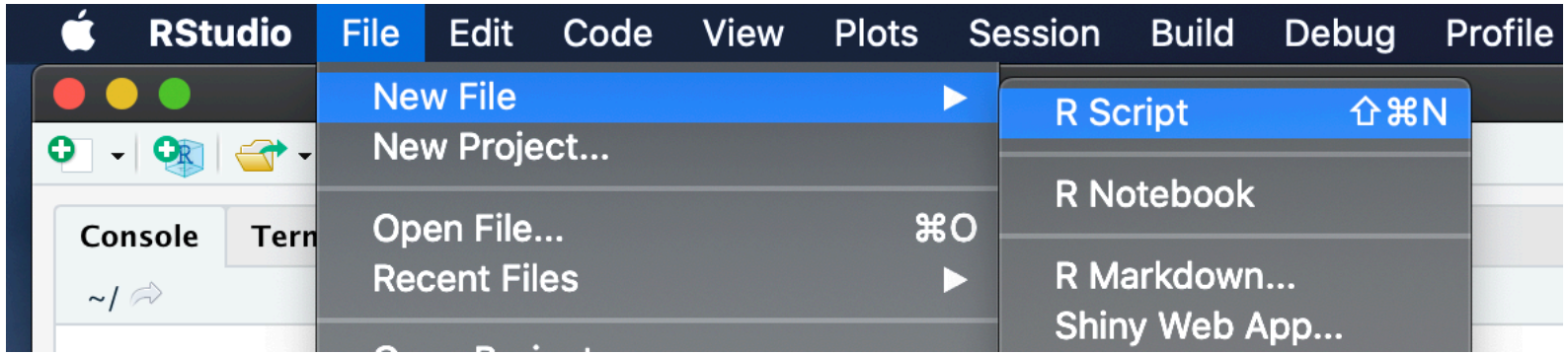


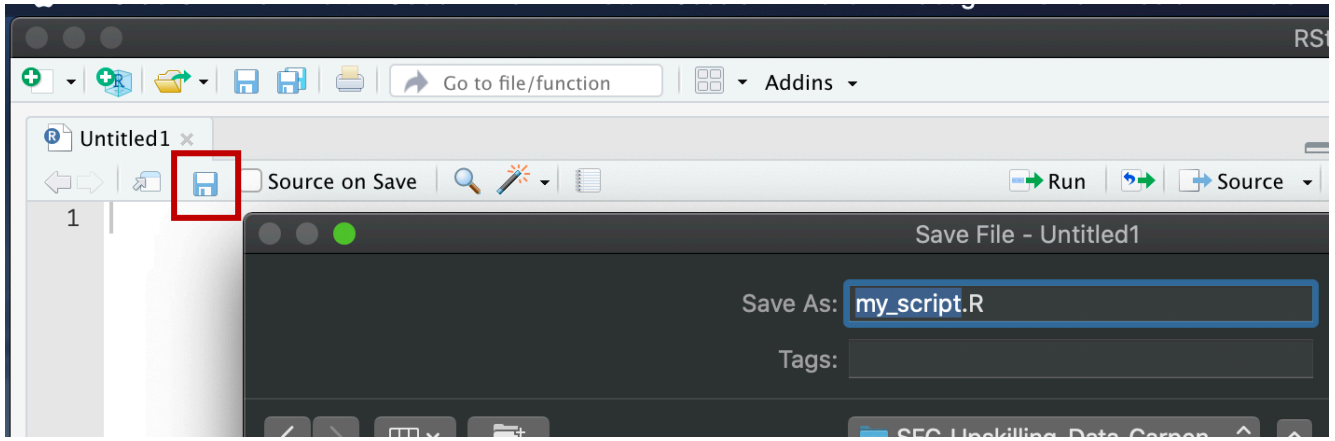
Lesson #1 – Setting up our working environment (Step #1: *Create your R script file*)

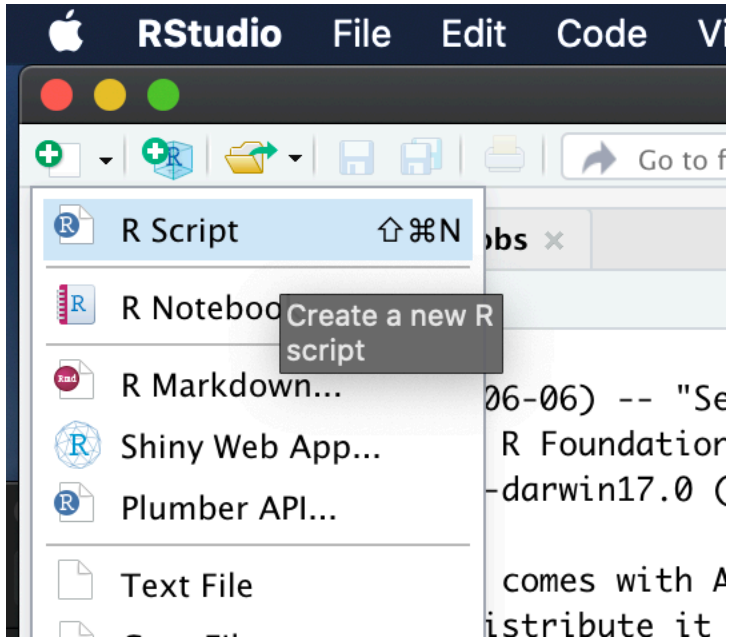
Create a new file where we will type our scripts

There are multiple ways around.. ☺

Go to *File* > *New File* > *R script*. Click the *save icon* on your toolbar and save your script as “script.R”.





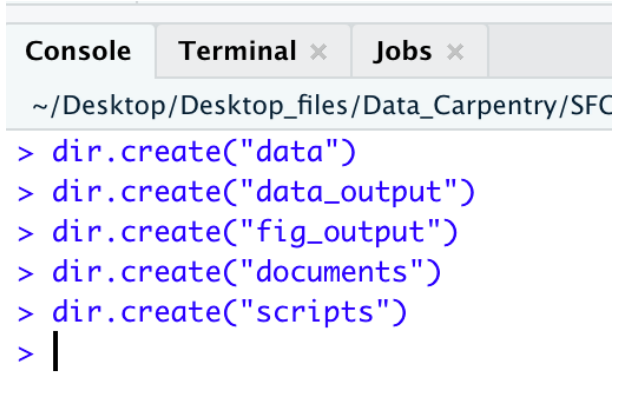


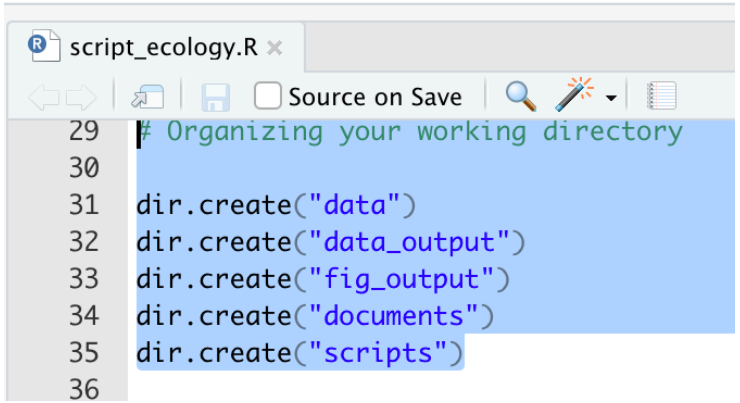
Lesson #1 – Setting up our working environment (*directory tree*)

Set and organizing your *directory tree*

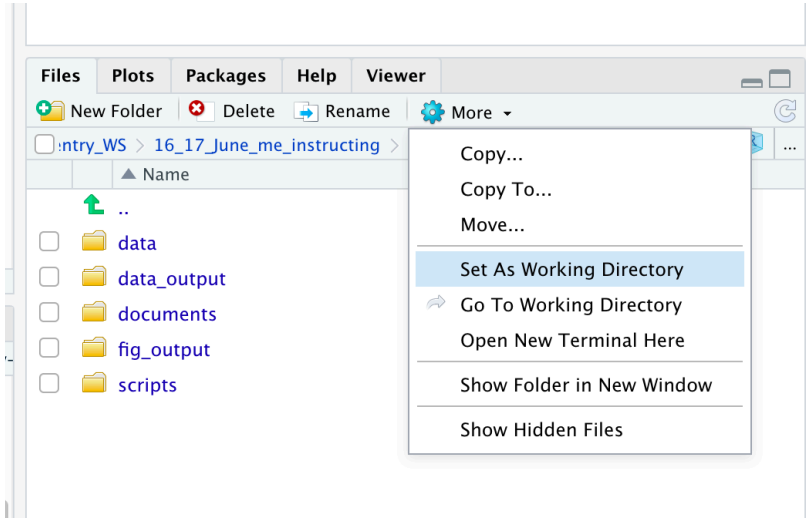
Under the *Files* tab on the right of the screen, click on *New Folder* and create a folder named *data_raw* within your newly created working directory (e.g., *~/data-carpentry/*)…

OR alternatively can type *R commands* in the *Console* or execute an *R script* (*Source* pane) – e.g.:





Hit *Enter* when wishing to execute commands on the *Console* (Left). And *Ctrl (Cmd)* + *Enter* will execute specific lines or highlighted code snippets in an R script (Right).



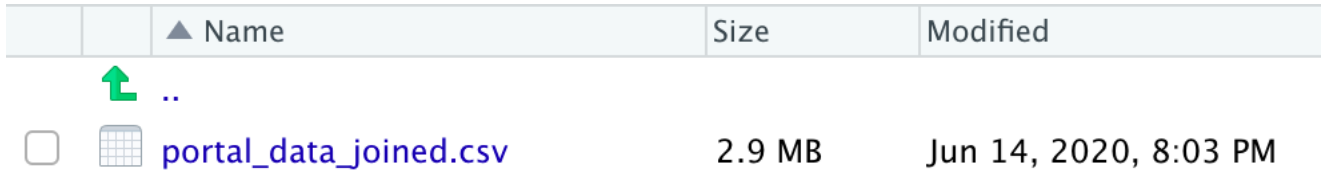
If your working directory is not what it should be, you can change it by navigating in the file browser to where your working directory should be, clicking on the *blue gear* icon "*More*", and selecting "*Set As Working Directory*".

Alternatively,
use *setwd("/path/to/working/directory")* to reset your working directory. However, your scripts should not include this line, because it will fail on someone else's computer (!).

Lesson #1 – Downloading the data directly into your *data* (or *raw_data*) folder

- The direct download link is: https://ndownloader.figshare.com/files/2292169.
- Place this downloaded file in the *data/* (OR *raw_data*) folder you just created.
- You can do this directly from R either from *R script* or as a *Console command*

```
download.file(url="https://ndownloader.figshare.com/files/2292169",
              destfile = "data/portal_data_joined.csv", mode = 'wb')
```

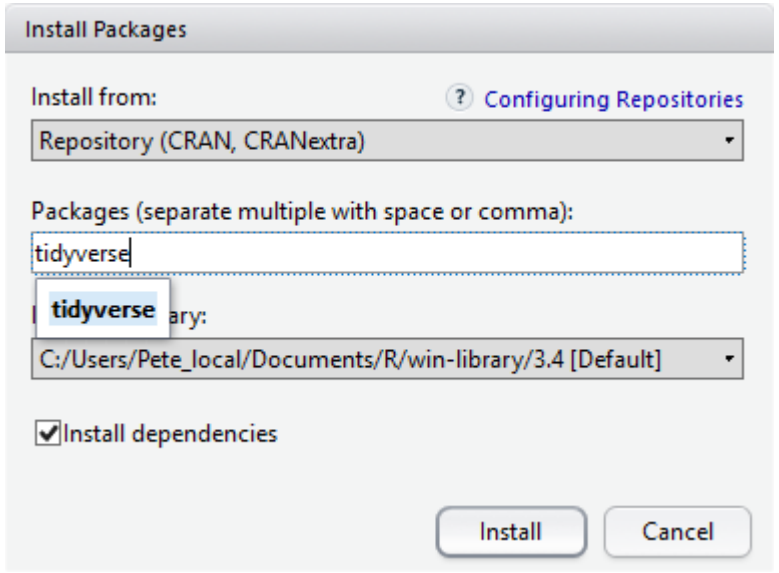


'mode' optional attribute: 'wb' (write binary) - relevant on Windows (and other Unix-alikes) it does distinguish between text and binary files..

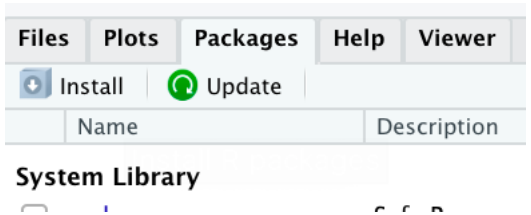
Lesson #1 – Installing and loading additional R packages – many ways around!

- Loads of R *packages* have been made available in central repositories, like the one hosted at *CRAN*, for anyone to download and install into their own R environment. Two main ways of installing and loading further packages.

1. Additional packages can be installed from the '*packages*' tab (RStudio GUI – Pane '4') → *Install* OR *Tools* → *Install Packages*…



Alternatively:



At the bottom of the Install Packages window is a check box to '*Install dependencies*'

2. Packages can be *installed* and *loaded* into RStudio by *Console command* OR *R script* – **syntax**: install.packages('*package_name*')

```
# tidyverse example
# install package alongside dependencies
install.packages('tidyverse', dependencies=T)

# load package
library(tidyverse)

search() # list packages which are loaded into RStudio
```

NB: Because the install process accesses the *CRAN repository*, you will need an Internet connection to install packages.

Lesson #1 – Installing and loading additional R packages **– Finger practice**

Select an arbitrary R package from *CRAN** Use the install option from the packages tab or use *install.packages()* and *library()* functions on the Console or Source (R script) panes to install and load another package into RStudio.

* https://cran.r-project.org/web/packages/available_packages_by_name.html

List of core packages for the workshop: https://github.com/datacarpentry/R-ecology-lesson/blob/master/needed_packages.R

Lesson #1 – Troubleshooting: Installing R packages (optional)

Sometimes a package will not install, try a different CRAN mirror

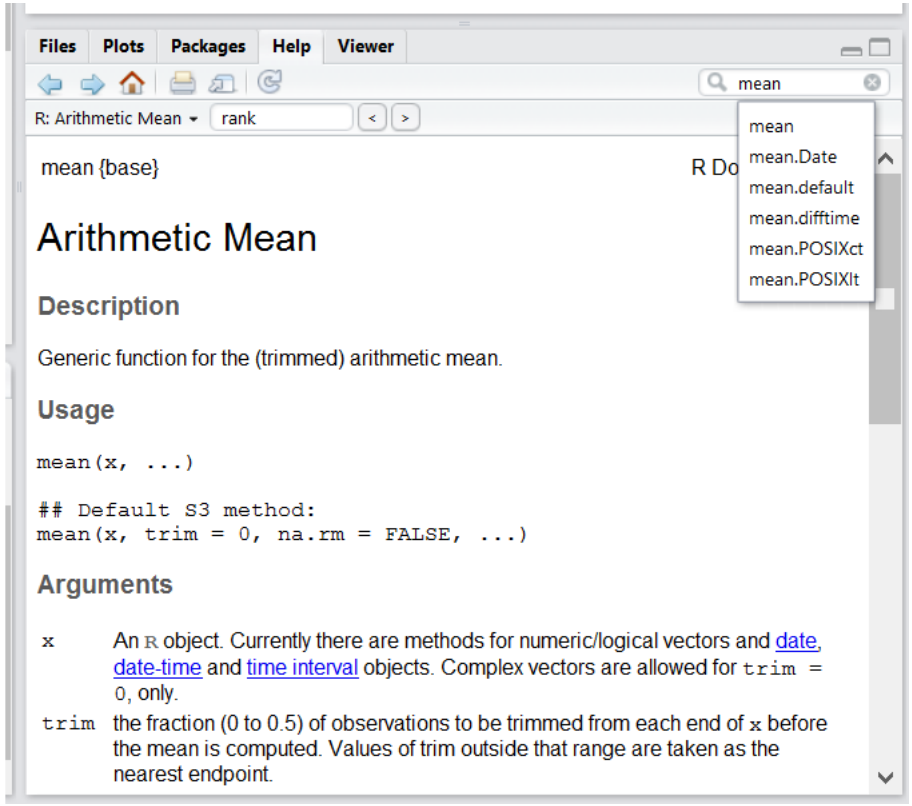
- *Tools > Global Options > Packages > CRAN Mirror*

Alternatively you can go to CRAN and download the package and install from ZIP file

- *Tools > Install Packages > set to 'from Zip/TAR'*

It is important that R, and the R packages be installed locally, not on a network drive.

Lesson #1 – **Looking for help? RStudio's built-in options..**

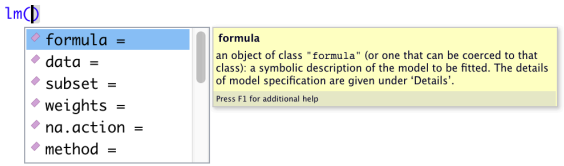


- **I know the name of the function I want to use, but I'm not sure how to use it**

```
> ?barplot
> help('barplot')
```

- **If you just need to remind yourself of the names of the arguments, you can use**

```
> args(lm)
```



- **I want to use a function that does X, there must be a function for it but I don't know which one…**

```
> help.search('mean')
> ??mean
```

- **I am stuck… I get an error message that I don't understand**

Google on the error message + 'r'

+1: If know the object's name but not sure what R package should be installed and loaded…

```
find('data.frame') # figure out what package contains the given object or function
```

Lesson #1 – **Looking for help? RStudio's built-in options..**

- **To share an object with someone else, if it's relatively small**, you can use the function *dput().* It will output R code that can be used to recreate the exact same object as the one in memory..

```
> dput(head(iris))
```

- **If the object is larger, provide the raw file** (i.e., your CSV file) **with your script up to the point of the error. Alternatively, in particular if your question is not related to a data frame, you can save any R object to a file**. It can be sent to someone by email who can read it with the *readRDS()*.

```
> saveRDS(iris, file='/tmp/iris.rds') # example dataset & file path
> some_data <- readRDS(file="~/Downloads/iris.rds") # assume .rds file in the Downloads folder
```

- always include the output of *sessionInfo()* or *R.version* function as it provides critical information about your platform, the versions of R and the packages that you are using, etc.

```
> sessionInfo()
R version 4.0.1 (2020-06-06)
Platform: x86_64-apple-darwin17.0 (64-bit)
Running under: macOS Catalina 10.15.5
```

```
> R.version
                _
platform       x86_64-apple-darwin17.0
arch           x86_64
os             darwin17.0
system         x86_64, darwin17.0
status
```

Lesson #1 – **Looking for help? Reach out Communities & User Forums**

RStudio Community

https://community.rstudio.com

RStudio Support

https://support.rstudio.com/hc/en-us/articles/201141096-Getting-Started-with-R

RDocumentation

https://www.rdocumentation.org

stackoverflow

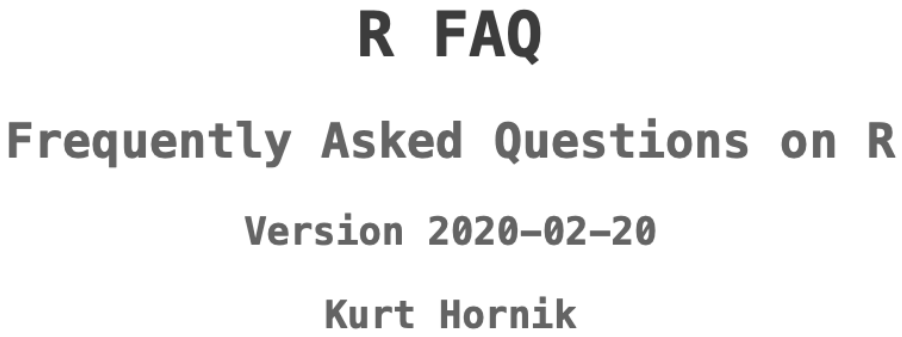
https://stackoverflow.com

http://edinbr.org

… etc

Lesson #1 – **Looking for help? More resources..**

- The R-help mailing list: it is read by a lot of people (including most of the R core team)
- The Posting Guide for the R mailing lists.
- How to ask for R help useful guidelines
- This blog post by Jon Skeet has quite comprehensive advice on how to ask programming questions.
- The reprex package is very helpful to create reproducible examples when asking for help. The [rOpenSci community call "How to ask questions so they get answered"], rOpenSci Blog and video recording includes a presentation of the *reprex* package and of its philosophy.



https://cran.r-project.org/doc/FAQ/R-FAQ.html

… etc

How to write a reproducible example

A reproducible example allows someone else to recreate your problem by just copying and pasting R code.

There are four things you need to include to make your example reproducible: required packages, data, code, and a description of your R environment.

•**Packages** should be loaded at the top of the script, so it's easy to see which ones the example needs.

•The easiest way to include **data** in an email is to use dput() to generate the R code to recreate it. For example, to recreate the mtcars dataset in R, I'd perform the following steps:

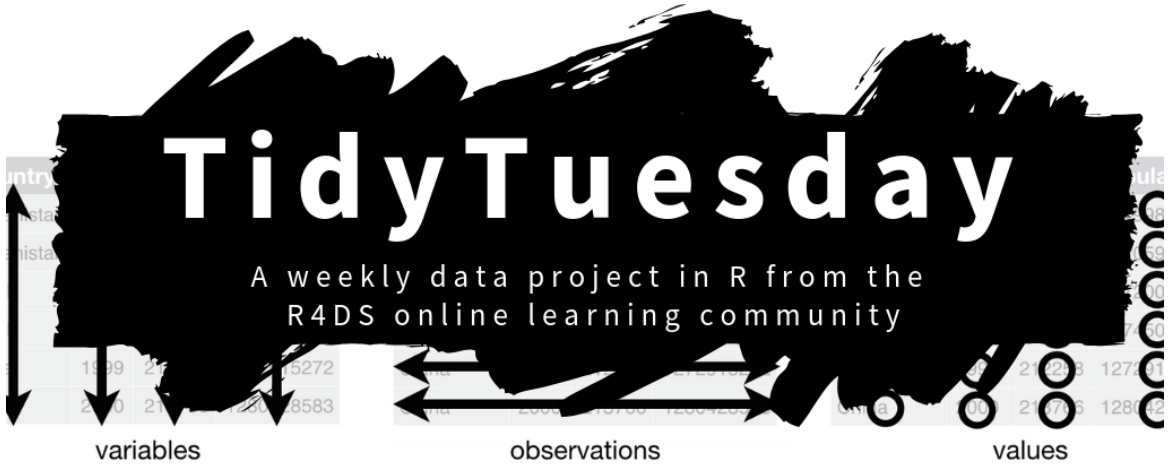
1.Run *dput(mtcars)* in R
2.Copy the output
3.In my reproducible script, type *mtcars* <- then paste.

•Spend a little bit of time ensuring that your **code** is easy for others to read:

•make sure you've used spaces and your variable names are concise, but informative
•use comments to indicate where your problem lies
•do your best to remove everything that is not related to the problem. The shorter your code is, the easier it is to understand.
•Include the output of *sessionInfo()* as a comment. This summarises your **R environment** and makes it easy to check if you're using an out-of-date package.

http://adv-r.had.co.nz/Reproducibility.html https://www.tidyverse.org/help/

Lesson #1 – **Getting involved.. Where to look? (plenty of sources out there!)**



https://www.tidytuesday.com

"We are a grassroots journal club initiative that helps young researchers create local Open Science journal clubs at their universities to discuss diverse issues, papers and ideas about improving science, reproducibility and the Open Science movement."

https://reproducibilitea.org

https://carpentries.org/community/

…etc

Lesson #1 – **Getting involved.. Where to look? (plenty of sources out there!)**

Explore exciting projects! ☺

https://towardsdatascience.com

Towards Data Science

https://www.kaggle.com/rtatman/welcome-to-data-science-in-r/

Publish your Open Source Software projects!

The Journal of Open Source Software

The Journal of Open Source Software is a **developer friendly**, open access journal for research software packages.

Committed to publishing quality research software with zero article processing charges or subscription fees.

https://joss.theoj.org

Lesson #1 – **Recommended readings**: a few hand-picked articles

Article Source: **Good enough practices in scientific computing**
Wilson G, Bryan J, Cranston K, Kitzes J, Nederbragt L, et al. (2017) Good enough practices in scientific computing. PLOS Computational Biology 13(6): e1005510.https://doi.org/10.1371/journal.pcbi.1005510

Article Source: **Best Practices for Scientific Computing**
Wilson G, Aruliah DA, Brown CT, Chue Hong NP, Davis M, et al. (2014) Best Practices for Scientific Computing. PLOS Biology 12(1): e1001745.https://doi.org/10.1371/journal.pbio.1001745

Wilkinson, M., Dumontier, M., Aalbersberg, I. *et al.* **The FAIR Guiding Principles for scientific data management and stewardship**. *Sci Data* 3, 160018 (2016). https://doi.org/10.1038/sdata.2016.18

Wickham H. **Tidy Data**. *Journal of Statistical Software* 59(10); (2014). 10.18637/jss.v059.i10

Lesson #1 – **Picking up the terminology - Glossary / Working definitions**

Engine (e.g. R) := R is a system for statistical computation and graphics. It provides, among other things, a programming language (compiling user-inputted commands), high level graphics, interfaces to other languages and debugging facilities.

IDE (e.g. RStudio) := *Integrated Development Environment*: An integrated development environment is a software application that provides comprehensive facilities to computer programmers for software development.

GUI := *Graphical User Interface*: A GUI (graphical user interface) is a system of interactive visual components for computer software.

CRAN := The Comprehensive R Archive Network. CRAN is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R. Please use the CRAN mirror nearest to you to minimize network load URL: https://cran.r-project.org .

Working directory := In computing, the working directory of a process is a directory of a hierarchical file system, if any, dynamically associated with each process. It is sometimes called the current working directory.

Any questions for Lesson #1 – ‘Before we start’?

Summary

How to find your way around RStudio?
How to interact with R?
How to manage your environment?
How to install packages?