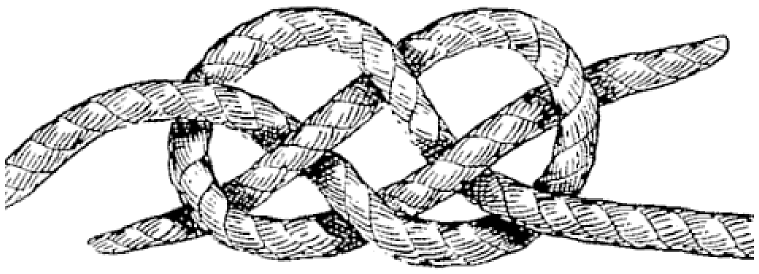
Coding togetheR

Alistair Bailey

April 22 2019

Contents

Welcome



A Carrick bend:[1] The Carrick bend is a type of sailor's knot used for joining two lines.

Coding togetheR is a series of collaborative workshops to teach foundational R codiing and data science skills at the University of Southampton in 2019. This book contains the materials covered over eight, two hour sessions.

The workshops are for anyone at the University of Southampton with data to analyse and who is struggling with their current tools. This series of eight weekly two hour workshops provides an introduction to working with data using R in a supported environment. Unlike traditional lessons, we all code together with the emphasis on participants learning by doing and by helping each other.

These materials are a mash-up of my own and many others. I've endevoured to credit everyone appropriately, but please message me[2] if I've messed up and I'll correct it. The main sources used here are: R for data science (R4DS)[3], the R4DS community[4], the Carpentries[5], Hands on Programming in R[6], swiRlstats[7] and Teaching Tech togetheR[8].

It was written using R (R Core Team, 2019) in RStudio (RStudio Team, 2018) using the bookdown package (Xie, 2018).

[1] https://en.wikipedia.org/wiki/File:Carrick-bend-Guten-Verrill-modified.png
[2] https://ab604.uk/
[3] https://r4ds.had.co.nz/
[4] https://www.rfordatasci.com/
[5] https://carpentries.org/
[6] https://rstudio-education.github.io/hopr/
[7] https://swirlstats.com/
[8] http://teachtogether.tech/en/

Preface

To follow these materials you will need an up to date version of R (R Core Team, 2019) and RStudio (RStudio Team, 2018). This may require requesting permission to install software from Isolutions if you have a University laptop.

If you are new to R, then the first thing to know is that R is a programming language and RStudio is a program for working with R called an integrated development environment (IDE). You can use R without RStudio, but not the other way around. Further details in Chapter 1.

Download R here[9] and Download RStudio Desktop here[10].

These materials were generated using R version 3.5.3.

Once you've installed R and RStudio, you'll also need a few R packages. Packages are collections of functions.

Open RStudio and put the code below into the `Console` window and press `Enter` to install the `tidyverse`,`janitor` and `here` packages.

```r
install.packages(c("tidyverse","janitor","here"))
```

[9] https://cran.r-project.org/
[10] https://www.rstudio.com/products/rstudio/download/

Chapter 1

Getting started

If, when faced with the thought of starting to learning to code you feel like this:

Then hopefully by the end of these workshops, you'll feel a bit more like this:

And if you like that, there is more at R for cats[3].

1.1 Philosohpy

In terms of the philosophy of learning to code:

1. The primary motivation for using tools such as R is to get more done, in less time and with less pain.
2. And the overall aim is to *understand and communicate* findings from our data.

As shown in Figure 1.3 of typical data analysis workflow, to acheive this aim we need to learn tools that enable us to perform the fundamental tasks of tasks of importing, tidying and often transforming the data. Transformation means for example, selecting a subset of the data to work with, or calculating the mean of a set of observations. We'll cover that in Chapter…

But first…

1.2 What are R and RStudio?

"There are only two kinds of languages: the ones people complain about and the ones nobody uses"

Bjarne Stroustrup, the inventor C++

R is a programming language that follows the philosophy laid down by it's predecessor S. The philosophy being that users begin in an interactive environment where they don't consciously think of themselves as programming. It was created in 1993, and documented in (Ihaka and Gentleman, 1996).

Reasons R has become popular include that it is both open source and cross platform, and that it has broad functionality, from the analysis of data and creating powerful graphical visualisations and web apps.

Like all languages though it has limitations, for example the syntax is initially confusing.

Take for example the word `environment`…

[3]https://www.rforcats.net/

Figure 1.1: Imposter syndrome[1].

Figure 1.2: R cat[2]

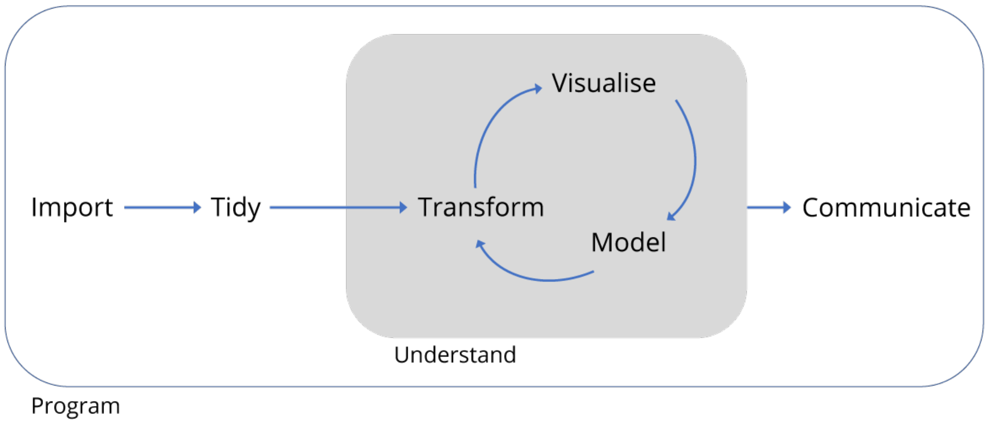


Figure 1.3: Data project workflow.

1.2.1 Environments

An environment is where we bring our data to work with it. Here we work in a R envrionment, using the R language as a set of tools. **RStudio** is an integrated development environment, or IDE for R programming. It is regularly updated, and upgrading enables access to the latest features.

The latest version can be downloaded here: http://www.rstudio.com/download

1.3 Why learn R, or any language ?

We can write R code without saving it, but it's generally more useful to write and save our code as a script. Working with scripts makes the steps you used in your analysis clear, and the code you write can be inspected by someone else who can give you feedback and spot mistakes.

Learning R (or any programming language) and working with scripts forces you to have deeper understanding of what you are doing, facilitates your learning and comprehension of the methods you use:

- Writing and publishing code is important for reproducible resarch
- R has many thousands of packages covering many disciplines.
- R can work with many types of data.
- They is a large R community for development and support.
- Using R gives you control over your figures and reports.

1.4 Finding your way around RStudio

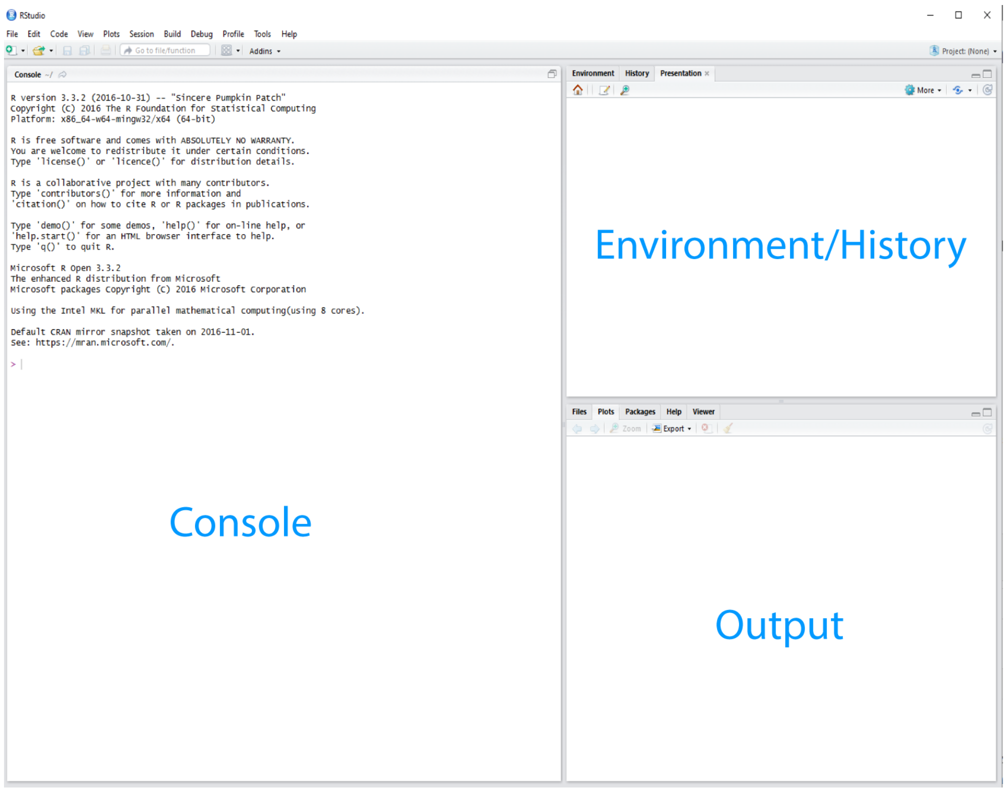
Let's begin by learning about RStudio[4], the Integrated Development Environment (IDE).

We will use R Studio IDE to write code, navigate the files found on our computer, inspect the variables we are going to create, and visualize the plots we will generate. R Studio can also be used for other things (e.g., version control, developing packages, writing Shiny apps) that we don't have time to cover during this workshop.

R Studio is divided into "Panes", see Figure 1.4.

When you first open it, there are three panes,the console where you type commands, your environment/history (top-right), and your files/plots/packages/help/viewer (bottom-right).

[4] https://www.rstudio.com/


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
Console ~/
Environment History Presentation
More
Files Plots Packages Help Viewer
Zoom Export
Environment/History
Console
Output

```
R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 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.

Microsoft R Open 3.3.2
The enhanced R distribution from Microsoft
Microsoft packages Copyright (C) 2016 Microsoft Corporation

Using the Intel MKL for parallel mathematical computing(using 8 cores).

Default CRAN mirror snapshot taken on 2016-11-01.
See: https://mran.microsoft.com/.

> 
```

Figure 1.4: The Rstudio Integrated Development Environment (IDE).

The enivronment shows all the R objects you have created or are using, such as data you have imported.

The output pane can be used to view any plots you have created.

Not opened at first start up is the fourth default pane: the script editor pane, but this will open as soon as we create/edit a R script (or many other document types). *The script editor is where will be typing much of the time.*

The placement of these panes and their content can be customized (see menu, R Studio -> Tools -> Global Options -> Pane Layout). One of the advantages of using R Studio is that all the information you need to write code is available ina single window. Additionally, with many shortcuts, auto-completion, and highlighting for the major file types you use while developing in R, R Studio will make typing easier and less error-prone.

Time for another philosphical diversion...

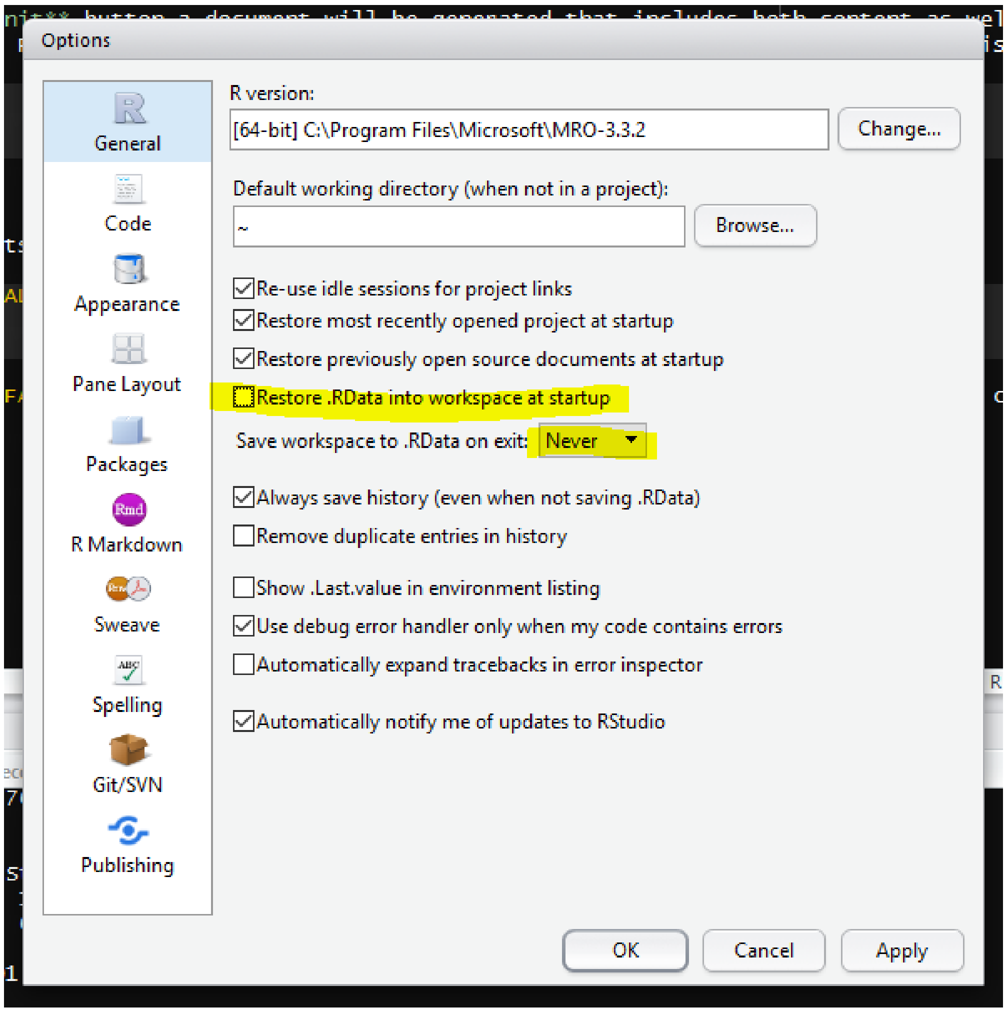
1.4.1 What is real?

At the start, we might consider our environment “real” - that is to say the objects we've created/loaded and are using are “real”. But it's much better in the long run to consider our scripts as “real” - our scripts are where we write down the code that creates our objects that we'll be using in our environment.

As a script is a document, it is reproducible

Or to put it another way: we can easily recreate an environment from our scripts, but not so easily create a script from an enivronment.

To support this notion of thinking in terms of our scripts as real, we recommend turning off the preservation of workspaces between sessions by setting the Tools > Global Options menu in R studio as shown in Figure



R version:
[64-bit] C:\Program Files\Microsoft\MRO-3.3.2

Default working directory (when not in a project):
~

- [✓] Re-use idle sessions for project links
- [✓] Restore most recently opened project at startup
- [✓] Restore previously open source documents at startup
- [ ] Restore .RData into workspace at startup

Save workspace to .RData on exit: Never

- [✓] Always save history (even when not saving .RData)
- [ ] Remove duplicate entries in history
- [ ] Show .Last.value in environment listing
- [✓] Use debug error handler only when my code contains errors
- [ ] Automatically expand tracebacks in error inspector
- [✓] Automatically notify me of updates to RStudio

Figure 1.5: Don't save your workspace, save your script instead.

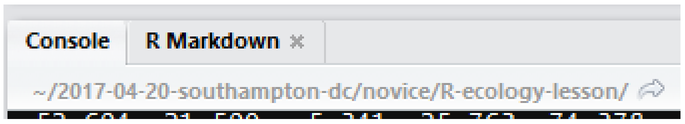
1.5:

1.5 Where am I?

R studio tells you where you are in terms of directory address like so:

If you are unfamiliar with how computers structure folders and files, then consider a tree with a root from which the trunk extends and branches divide. In the image above, the ~ symbol represents a contraction of the path from the root to the 'home' directory (in Windows this is 'Documents') and then the forward slashes are the branches. (Note: Windows uses backslashes, Unix type systems and R use forwardslashes).

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 makes it a lot easier 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.



```
~/2017-04-20-southampton-dc/novice/R-ecology-lesson/
```

Figure 1.6: Your working directory

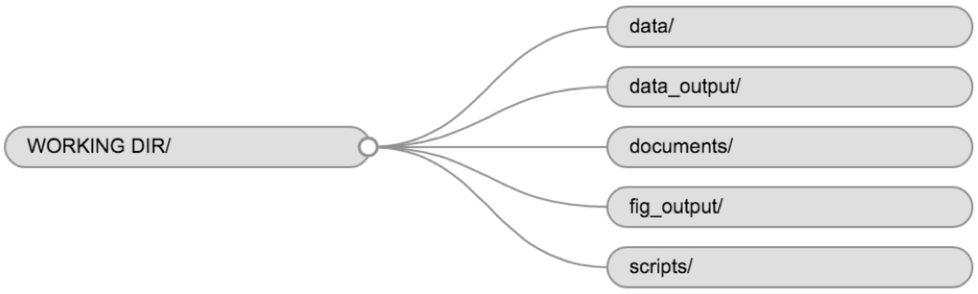
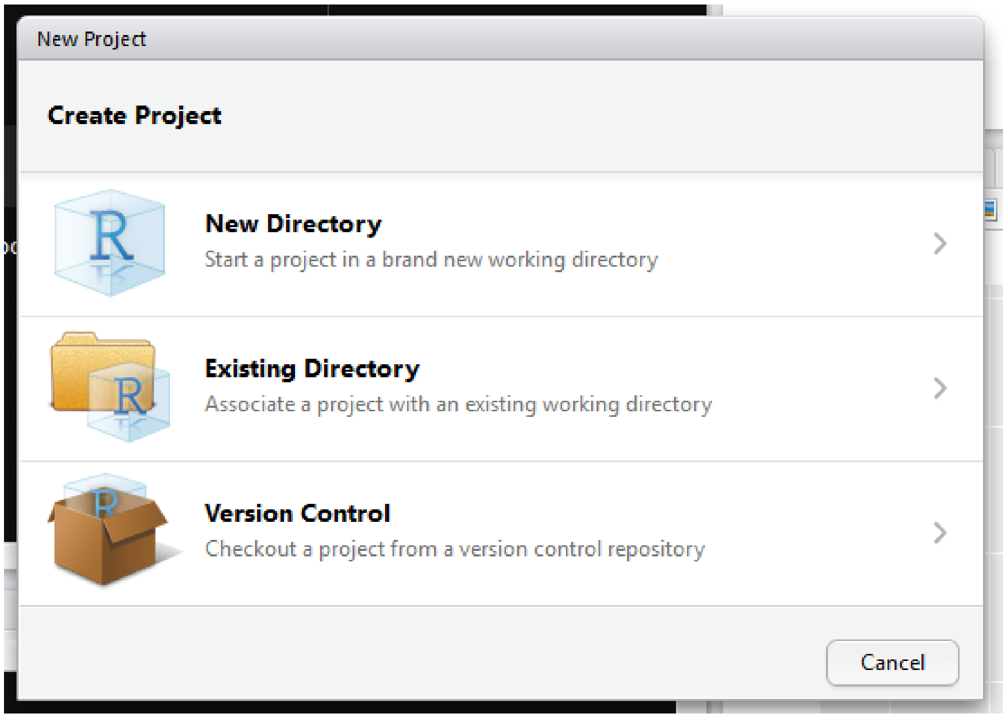


Figure 1.7: A typical directory structure


New Project

Create Project

New Directory
Start a project in a brand new working directory

Existing Directory
Associate a project with an existing working directory

Version Control
Checkout a project from a version control repository

Figure 1.8: Creating a R project

1.6 R projects

RStudio also has a facility to keep all files associated with a particular analysis together called a project.

Creating a project creates a working directory for you and also remembers its location (allowing you to quickly navigate to it) and optionally preserves custom settings and open files to make it easier to resume work after a break.

Below, we will go through the steps for creating an "R Project":

- Start R Studio (presentation of R Studio -below- should happen here)
- Under the `File` menu, click on `New project`, choose `New directory`, then `Empty project`
- Enter a name for this new folder (or "directory", in computer science), and choose a convenient location for it. This will be your **working directory** for the rest of the day (e.g., `~/coding-together`)
- Click on "Create project"
- Under the `Files` tab on the right of the screen, click on `New Folder` and create a folder named `data` within your newly created working directory. (e.g., `~/data`)
- Create a new R script (File > New File > R script) and save it in your working directory (e.g. `01-coding-together-worksho`

1.7 Using R projects with the `here` package

Jenny Bryan loves the `here`[5] package by Kirill Müller so much she wrote an ode to it[6].

In a nutshell, `here` sets the path implicitly to the top level of the R project you are working in. **But what does that mean, and why should I care?**

[5] https://here.r-lib.org/index.html
[6] https://github.com/jennybc/here_here

What it means is that your code becomes reproducible on another machine where the path to your project is different. For example…

1.8 Naming things

Jenny Bryan[7] has three principles for naming things[8] that are well worth remembering.

When you names something, a file or an object, ideally it should be:

1. Machine readable (no whitespace, punctuation, upper AND lowercase…)
2. Human readable (makes sense in 6 months or 2 years time)
3. Plays well with default ordering (numerical or date order)

1.9 Seeking help

If you need help with a specific R function, let's say `barplot()`, you can type:

```
?barplot
```

A Google or internet search "R <task>" will often either send you to the appropriate package documentation or a helpful forum question that someone else already asked, such as Stack Overflow[9] or the RStudio Community[10].

1.9.1 Asking for help

As well as knowing where to ask[11], the key to get help from someone is for them to grasp your problem rapidly. You should make it as easy as possible to pinpoint where the issue might be.

Try to use the correct words to describe your problem. For instance, a package is not the same thing as a library. Most people will understand what you meant, but others have really strong feelings about the difference in meaning. The key point is that it can make things confusing for people trying to help you. Be as precise as possible when describing your problem.

If possible, try to reduce what doesn't work to a simple *reproducible example* otherwise known as a *reprex*.

For more information on how to write a reproducible example see this article[12] using the `reprex` package.

[7] https://ropensci.org/blog/2017/12/08/rprofile-jenny-bryan/
[8] http://www2.stat.duke.edu/~rcs46/lectures_2015/01-markdown-git/slides/naming-slides/naming-slides.pdf
[9] http://stackoverflow.com/questions/tagged/r
[10] https://community.rstudio.com/
[11] https://www.tidyverse.org/help/#where-to-ask
[12] https://www.tidyverse.org/help/#reprex

Chapter 2

Data wrangling I

Chapter 3

Data wrangling II

Chapter 4

Functions

Chapter 5

Exploratory modelling

Chapter 6

Visualistion

Chapter 7

Reports

Using R for report writing and presentations.

References

Ihaka, R. and Gentleman, R. (1996). R: a language for data analysis and graphics. *Journal of computational and graphical statistics*, 5(3):299–314.

R Core Team (2019). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

RStudio Team (2018). *RStudio: Integrated Development Environment for R*. RStudio, Inc., Boston, MA.

Xie, Y. (2018). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.9.