# Using R at Grattan Institute

*Will Mackey and Matt Cowgill*

*2019-08-01*

# Contents

# Introduction

R is good and cool. Do you want to be good and cool? Use R!

# Chapter 1

# Using R at Grattan

```
library(tidyverse)
```

This document sets out good practices for structuring your R analysis at Grattan Institute. Having a clear, consistent structure for our analyses means that our work is more easily checked and revised, including by ourselves in the future. A small investment of time up front to set up your analysis will save time (your own and others') down the track.

This guide is designed for *everyone* using R at Grattan. It includes a combination of rules and guidelines.

You should also be aware of the Grattan Institute R Style Guide, which lives in the same place as this document.

Any compaints or comments about this guide can be sent to Will or Matt, respectively.

## 1.1 Why use R?

It's good and cool!

## 1.2 Using R projects for a fully reproducible workflow.

*Finally adhering to the 'hit by a bus' rule.*

Cover: 1. setwd() and machine-speficic filepaths are bad 2. relative file paths are good 3. RStudio projects are an easy, reproducible way to set your wd

### 1.2.1 Filepaths

Filepaths should be relative to the working directory, and the working directory should be set by the project.

**Good**

```
hes <- read_csv("data/HES/hes1516.csv")
grattan_save("images/expenditure_by_income.pdf")
```

**Bad**

```
hes <- read_csv("/Users/mcowgill/Desktop/hes1516.csv")
hes <- read_csb("C:\Users\mcowgill\Desktop\hes1516.csv")
grattan_save("/Users/mcowgill/Desktop/images/expenditure_by_income.pdf")
```

### 1.2.2 Keep your scripts manageable

As a general rule of thumb, use one script per output. It should be clear what your script is trying to do (use comments!).

Consider breaking your analysis into pieces. For example:

- 01_import.R
- 02_tidy.R
- 03_model.R
- 04_visualise.R

**Don't** include interactive work (like `View(mydf)`, `str(mydf)`, `mean(mydf$variable)`, etc.) in your saved script.

### 1.2.3 Use subfolders of your project folder

Remember the hit-by-a-bus rule. It should be easy for any Grattan colleague to open your project folder and get up to speed with what it does. Putting all your files - raw data, scripts, output - in the one folder makes it harder to understand how your work fits together.

Use subfolders to clearly separate your code, raw data, and output.

## 1.3 Grattan coding style guide

Short summary of why

Link to style guide

## 1.4 What is the tidyverse and why do we use it?

Introduce following chapters

## 1.5 An introduction to RMarkdown

## 1.6 Resources in this package

- Starting a piece of analysis 'cheat sheet'.
- Updated style guide.
- Written guide/slides.

# Chapter 2

# Data Visualisation

## 2.1 Using `ggplot2` to create graphs in R

### 2.1.1 Concepts

Main ingredients to a `ggplot` chart: - Tidy data - Aesthetics - Geoms

Along with: - Facets - Colours - Labels

### 2.1.2 Bar charts

`geom_bar` if you have unit-record data and you want the geom to calculate something (count, sum, etc). `geom_col` if you want to plot numbers exactly as they are. This is how charts in Excel or Powerpoint work. It is the same as `geom_bar(stat = "identity")`

The `position` argument…

Remember: don't use too many colours (…and other viz tips from the Chart Style Guide)

### 2.1.3 Line charts

### 2.1.4 Scatter plots

`geom_point` `geom_smooth`

### 2.1.5 Distributions

`geom_histogram` `geom_density`

`ggridges::`

### 2.1.6 Maps

`absmapsdata`

## 2.2 Using grattantheme to make and export "Grattan-y" charts

(current text taken from an email I sent an intern once – will need to be updated)

The `grattantheme` package is hosted here: https://github.com/mattcowgill/grattantheme

You can install it with install_github from the `remotes` package:

```
install.packages("remotes")
remotes::install_github("mattcowgill/grattantheme")

library(grattantheme)
```

You can look at explanatory vignette with `vignette("using_grattantheme", "grattantheme")`.

Key functions are:

`theme_grattan()` to set the size and default single-colour. Set flipped = TRUE if you have used coord_flip() on the chart.

`grattan_fill_manual(n = 2)` to manually set the fill colours of a chart if you have mapped fill to an aesthetic, eg `aes(…, fill = gender)`. Set `n` to the number of colours you have.

`grattan_colour_manual(n = 2)` to manually set the colour if you have mapped colour to an aesthetic, eg `aes(…, colour = gender)`.

`grattan_y_continuous()` to properly style the Y-axis and align it with zero.

`grattan_save()` instead of `ggsave()` to export charts.

## 2.3 Creating simple interactive graphs with `plotly`

```
plotly::ggplotly()
```

# Chapter 3

# Reading data

## 3.1 Importing data

### 3.1.1 Reading CSV files

#### 3.1.1.1 `read_csv()`

The `read_csv()` function from the `tidyverse` is quicker and smarter than `read.csv` in base R.

Pitfalls: 1. read_csv is quicker because it surveys a sample of the data

We can also compress `.csv` files into `.zip` files and read them *directly* using `read_csv()`:

```
read_csv("data/my_data.zip")
```

This is useful for two reasons:

1. The data takes up less room on your computer; and
2. The original data, which shouldn't ever be directly edited, is protected and cannot be directly edited.

#### 3.1.1.2 `data.table::fread()`

The `fread` function from `data.table` is quicker than both `read.csv` and `read_csv`.

### 3.1.2 `readxl::read_excel()`

### 3.1.3 `rio`

### 3.1.4 `readabs`

## 3.2 Reading common files:

- TableBuilder CSVSTRINGs
- HES household file
- SIH
- LSAY and derivatives

See data directory for a list of microdata available to Grattan.

## 3.3 Appropriately renaming variables

As shown in the style guide

Add `rename_abs` function to a common Grattan package?

## 3.4 Getting to tidy data

`pivot_long()` and `pivot_wide()` *Make sure these are stable btw*

# Chapter 4

# Different data types

## 4.1 Tidy data

Other data structures

## 4.2 Dates with `lubridate::`

The `lubridate::` package

## 4.3 Strings with `stringr::`

- Replacing values
- Matching values
- Separating columns

## 4.4 Factors with `forcats::`

- Dangers with factors

# Chapter 5

# Data transformation

## 5.1 The pipe

## 5.2 Key `dplyr` functions:

All have the same syntax structure, which enable pipe-chains.

## 5.3 Filter with `filter()`

## 5.4 Arrange with `arrange()`

## 5.5 Select variables with `select()`

## 5.6 Group data with `group_by()`

## 5.7 Edit and add new variables with `mutate()`

### 5.7.1 Cases when you should use `case_when()`

## 5.8 Summarise data with `summarise()`

## 5.9 Joining datasets with `*_join()`

# Chapter 6

# Analysis

# Chapter 7

# Creating functions

## 7.1 It can be useful to make your own function

Why on earth would you create your own function?

## 7.2 Defining simple functions

## 7.3 More complex functions

## 7.4 Sets of functions

## 7.5 Using `purrr::map`

## 7.6 Sharing your useful functions with Grattan

# Chapter 8

# Version control

## 8.1 Version control is important and intimidating

Version control is great!

## 8.2 Github

We use Github to version-control and share reports in LaTeX, so you're already a bit set-up.

## 8.3 Git

Using Git within R Studio...