

Using R at Grattan Institute

Will Mackey and Matt Cowgill

2019-09-07

Contents

Welcome	5
1 Introduction to R	7
1.1 What is R?	7
1.2 What is RStudio?	10
1.3 Installing R and RStudio	12
1.4 Packages	18
2 Why use R?	21
2.1 Why use script-based software?	21
2.2 Why use R specifically?	21
3 Using R at Grattan	23
3.1 Using R projects for a fully reproducible workflow.	23
3.2 Grattan coding style guide	24
3.3 What is the tidyverse and why do we use it?	24
3.4 An introduction to RMarkdown	24
3.5 Resources in this package	24
4 Data Visualisation	25
4.1 Introduction to data visualisation	25
4.2 Set-up and packages	26
4.3 Concepts	27
4.4 Exploratory data visualisation	31
4.5 Making Grattan-y charts	31
4.6 Adding labels	42
5 Reading data	45
5.1 Importing data	45
5.2 Reading common files:	46
5.3 Appropriately renaming variables	46
5.4 Getting to tidy data	46
6 Different data types	47

6.1	Tidy data	47
6.2	Dates with <code>lubridate::</code>	47
6.3	Strings with <code>stringr::</code>	47
6.4	Factors with <code>forcats::</code>	47
7	Data transformation	49
7.1	The pipe	49
7.2	Key <code>dplyr</code> functions:	49
7.3	Filter with <code>filter()</code>	50
7.4	Arrange with <code>arrange()</code>	50
7.5	Select variables with <code>select()</code>	50
7.6	Group data with <code>group_by()</code>	50
7.7	Edit and add new variables with <code>mutate()</code>	50
7.8	Summarise data with <code>summarise()</code>	50
7.9	Joining datasets with <code>*_join()</code>	50
8	Analysis	51
9	Creating functions	53
9.1	It can be useful to make your own function	53
9.2	Defining simple functions	53
9.3	More complex functions	53
9.4	Sets of functions	53
9.5	Using <code>purrr::map</code>	53
9.6	Sharing your useful functions with Grattan	53
10	Version control	55
10.1	Version control is important and intimidating	55
10.2	Github	55
10.3	Git	55

Welcome

This guide is designed for everyone who uses – or would like to use – R at Grattan Institute.

It does two main things:

1. Shows you how to use R to complete common analytical tasks you'll face at Grattan.
2. Sets out some guidelines and good practices when using R at Grattan.

As a guide to using R, this website is helpful but incomplete. We can't possibly cover - or anticipate - all the skills you might need to know. If you make it to the end of this guide and want to learn more, start by reading *R for Data Science* by Hadley Wickham and Garrett Golemund. It's free.

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

Chapter 1

Introduction to R

Most people reading this guide will know what R is. But if you don't - that's OK!

If you have used R before and are comfortable enough with it, you might want to skip to the next page. This page is intended for people who are unfamiliar with R.

1.1 What is R?

R is a programming language that is designed by and for statisticians, data scientists, and other people who work with data. It's free - you can download R at no charge. It's also open source - you can view and (if you're game) modify the code that underlies the R language. R is available for all major computing platforms including Windows, macOS, and Linux.

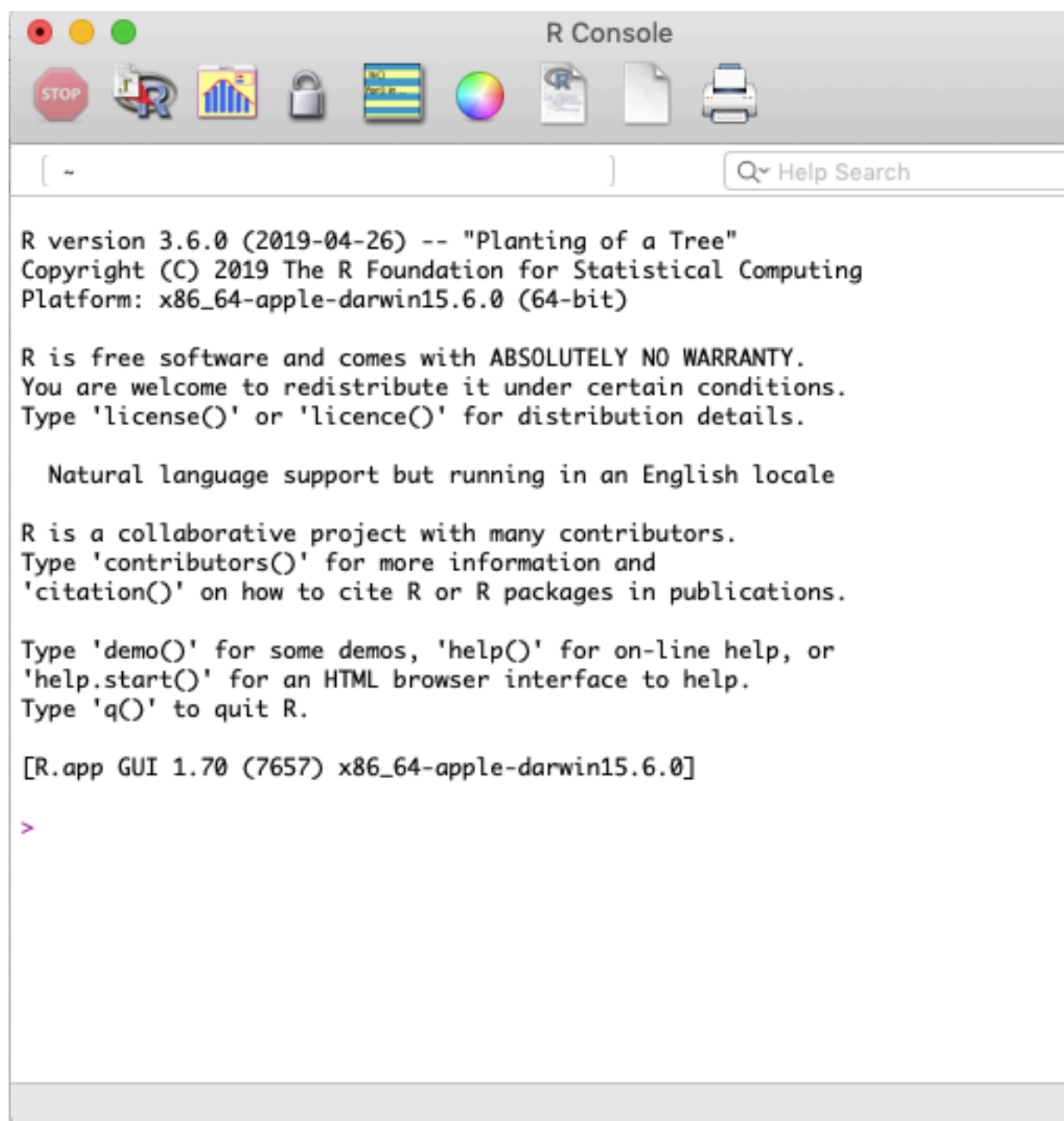
R has a lot in common with other statistical software like SAS, Stata, SPSS or Eviews. You can use those software packages to read data, manipulate it, generate summary statistics, estimate models, and so on. You can use R for all those things and more. You interact with R by writing code. This is a little different to Stata or SPSS, which allow you to do at least part of your analyses by clicking on menus and buttons. This means the initial learning curve for R can be a little steeper than for something like SPSS, but there are great benefits to a code-based approach to data analysis (see the next page for more on this).

R also has some overlap with general purpose programming languages like Python. But R is more focused on the sort of tasks that statisticians, data scientists, and academic researchers do.

R is quite old, having been first released publicly in 1995, but it's also growing and changing rapidly. A lot of developments in R come in the form of new

add-on pieces of software - known as ‘packages’ - that extend R’s functionality in some way. We cover packages more later in this page.

When you open R itself, you’re confronted with a few disclaimers and a command prompt, similar in appearance to the Terminal on macOS or command prompt in Windows.



This looks a bit intimidating, but you'll almost never open R directly and interact with it in that way.

To analyse data with R, you will typically write out a text file containing your code. This file - which we'll call a script - should be able to be read and executed by R from start to finish. The easiest way to write your code, run your script, and generate your outputs (whether that's a chart, a document, or a set of model results) is to use RStudio.

1.2 What is RStudio?

RStudio is another piece of free software you can download and run on your computer.¹ It's also available for Windows, macOS and Linux. In programmer jargon, RStudio is an “integrated development environment” or IDE. This means RStudio has a range of tools that help you work with R. It has a text editor for you to write R scripts, an R ‘console’ to interact directly with the language, and panes that let you see the objects you have stored in memory and any graphs you’ve created.

¹RStudio is, somewhat confusingly, a product made by a company called RStudio. Although the RStudio desktop software is free, RStudio makes money by charging for other services, like running R in the cloud. When we refer to RStudio, we’re referring to the desktop software unless we make it clear that we mean the company.

The screenshot shows the RStudio IDE interface. The main editor pane displays an R Markdown document titled 'Using_R_at_Grattan.Rmd'. The document content includes a title '## What is RStudio?', a paragraph about RStudio, and a code chunk for installing and loading the tidyverse package. The console pane at the bottom shows the output of the code chunk, including the list of attached packages and any conflicts.

Document Content:

```

'packages' - that extend R's functionality in some way. We cover packages more [later in this
page](#packages).
16
17 When you open R itself, you're confronted with a few disclaimers and a command prompt, similar in
appearance to the Terminal on macOS or command prompt in Windows.
18
19 `r knitr::include_graphics("atlas/r_screenshot.png")`
20
21 This looks a bit intimidating, but you'll almost never open R directly and interact with it in that
way.
22
23 To analyse data with R, you will typically write out a text file containing your code. This file -
which we'll call a script - should be able to be read and executed by R from start to finish. The
easiest way to write your code, run your script, and generate your outputs (whether that's a chart, a
document, or a set of model results) is to use RStudio.
24
25 ## What is RStudio?
26
27 RStudio is another piece of free software you can download and run on your computer.^[RStudio is,
somewhat confusingly, a product made by a company called RStudio. Although the RStudio desktop
software is free, RStudio makes money by charging for other services, like running R in the cloud.]
It's also available for Windows, macOS and Linux. In programmer jargon, RStudio is an "integrated
development environment" or IDE. This means RStudio has a range of tools that help you work with R. It
has a text editor for you to write R scripts, an R 'console' to interact directly with the language,
and panes that let you see the objects you have stored in memory and any graphs you've created.
28
29
30
31 You'll almost always interact with R by opening RStudio. You need to install R se
32
33 ## Installing R and RStudio
34
35 ## Packages {#packages}
36
37 ### What is a package?
38

```

Console Output:

```

> library(tidyverse)
— Attaching packages — tidyverse 1.2.1.9000
✓ ggplot2 3.2.1    ✓ purrr  0.3.2
✓ tibble  2.1.3    ✓ dplyr  0.8.3
✓ tidyr   0.8.3    ✓ stringr 1.4.0
✓ readr   1.3.1    ✓ forcats 0.4.0
— Conflicts — tidyverse_conflicts()
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
> ggplot(mtcars, aes(x = wt, y = mpg)) + geom_point() + grattantheme::theme_grattan()
>

```

Annotations:

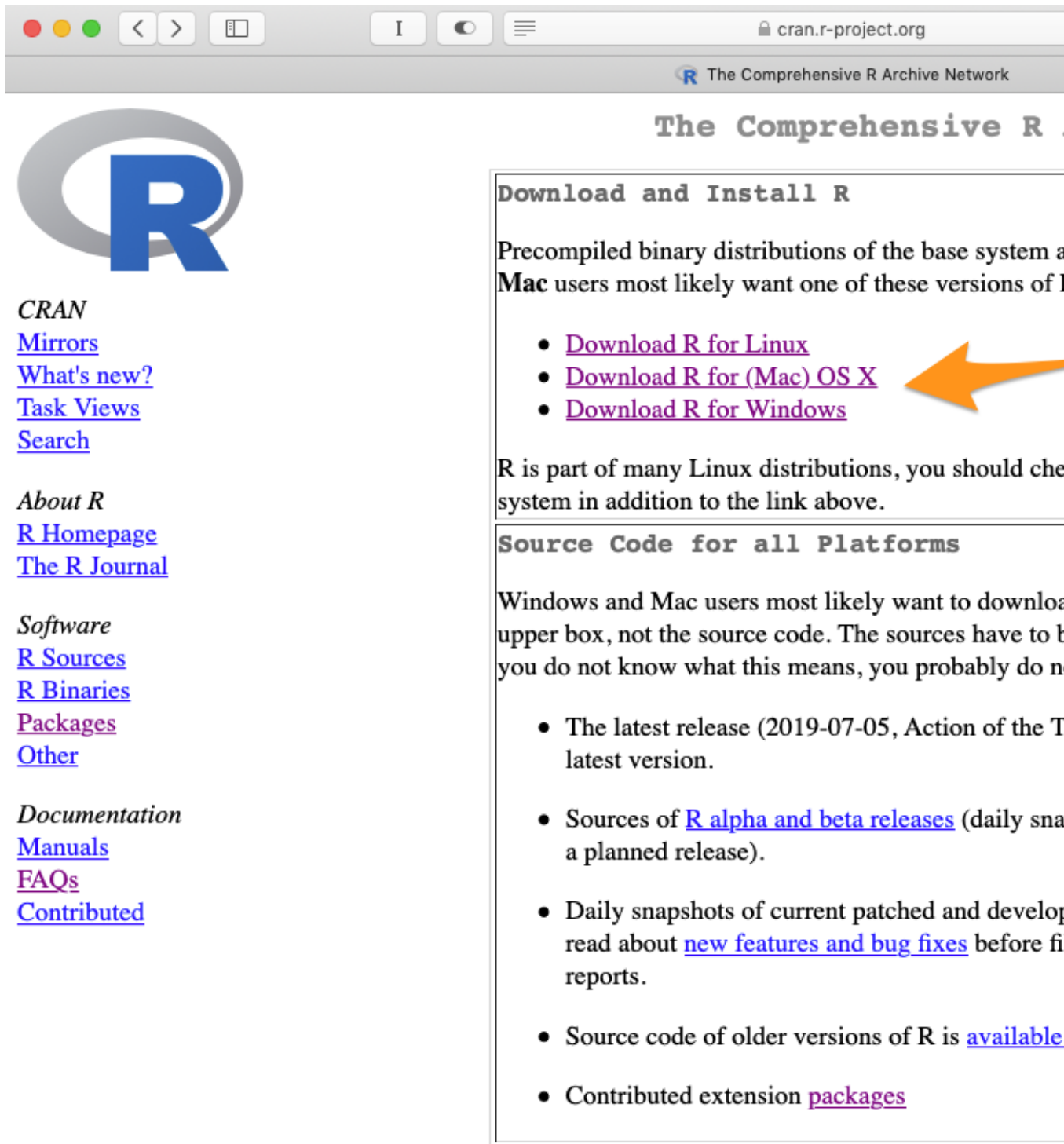
- This is where a text editor where you can write an R script - or an RMarkdown document like this one!** (points to the main editor pane)
- This is your 'console', where you can directly give commands to R and see the results** (points to the console pane)

You'll almost always interact with R by opening RStudio.

1.3 Installing R and RStudio

Although you'll usually work with R by opening RStudio, you need to install both R and RStudio separately.

Install R by going to CRAN, the Comprehensive R Archive Network. CRAN is a community-run website that houses R itself as well as a broad range of R packages.



The screenshot shows a web browser window with the address bar displaying `cran.r-project.org`. The page title is "The Comprehensive R Archive Network". The main heading is "The Comprehensive R". Below this, there is a large blue "R" logo. To the left of the logo, there is a sidebar with links: "CRAN", "Mirrors", "What's new?", "Task Views", "Search", "About R", "R Homepage", "The R Journal", "Software", "R Sources", "R Binaries", "Packages", "Other", "Documentation", "Manuals", "FAQs", and "Contributed". To the right of the logo, there is a section titled "Download and Install R". This section contains the text: "Precompiled binary distributions of the base system are available for Linux, Mac, and Windows. Mac users most likely want one of these versions of R". Below this text, there is a list of three links: "Download R for Linux", "Download R for (Mac) OS X", and "Download R for Windows". An orange arrow points to the "Download R for (Mac) OS X" link. Below the list, there is a paragraph: "R is part of many Linux distributions, you should check your distribution's documentation for more information on installing R on your system in addition to the link above." Below this paragraph, there is a section titled "Source Code for all Platforms". This section contains the text: "Windows and Mac users most likely want to download the precompiled binaries in the upper box, not the source code. The sources have to be compiled, and if you do not know what this means, you probably do not want to use them." Below this text, there is a list of four links: "The latest release (2019-07-05, Action of the Task Force on R's latest version)", "Sources of R alpha and beta releases (daily snapshots of R in development, a planned release)", "Daily snapshots of current patched and development versions of R, read about new features and bug fixes before finalizing a release", and "Source code of older versions of R is available". Below the list, there is a link: "Contributed extension packages".

CRAN

[Mirrors](#)

[What's new?](#)

[Task Views](#)

[Search](#)

About R

[R Homepage](#)

[The R Journal](#)

Software

[R Sources](#)

[R Binaries](#)

[Packages](#)

[Other](#)

Documentation

[Manuals](#)

[FAQs](#)

[Contributed](#)

The Comprehensive R

Download and Install R

Precompiled binary distributions of the base system are available for Linux, Mac, and Windows. Mac users most likely want one of these versions of R

- [Download R for Linux](#)
- [Download R for \(Mac\) OS X](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check your distribution's documentation for more information on installing R on your system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries in the upper box, not the source code. The sources have to be compiled, and if you do not know what this means, you probably do not want to use them.

- The latest release (2019-07-05, Action of the Task Force on R's latest version).
- Sources of [R alpha and beta releases](#) (daily snapshots of R in development, a planned release).
- Daily snapshots of current patched and development versions of R, read about [new features and bug fixes](#) before finalizing a release.
- Source code of older versions of R is [available](#).
- Contributed extension [packages](#)

You want to download the latest base R release, as a ‘binary’. Don’t worry, you don’t need to know what a binary is.

For macOS, the page will look like this:



CRAN

[Mirrors](#)

[What's new?](#)

[Task Views](#)

[Search](#)

About R

[R Homepage](#)

[The R Journal](#)

Software

[R Sources](#)

[R Binaries](#)

[Packages](#)

[Other](#)

Documentation

[Manuals](#)

[FAQs](#)

[Contributed](#)

R for M

This directory contains binaries for a base distribution and packages. OS 8.6 to 9.2 (and Mac OS X 10.1) are no longer supported for 32-bit systems (which is R 1.7.1) [here](#). Releases for old Mac OS X are found in the [old](#) directory.

Note: CRAN does not have Mac OS X systems and cannot cross-compile when assembling binaries, please use the normal precautions.

As of 2016/03/01 package binaries for R versions older than 3.0.0 such versions should adjust the CRAN mirror setting accordingly.

R 3.6.1 "Action of the To

Important: since R 3.4.0 release we are now providing binary toolkits to provide support for OpenMP and C++17 standard library tools from the [tools](#) directory and read the corresponding notes.

Please check the MD5 checksum of the downloaded file during the mirroring process. For example type `md5 R-3.6.1.pkg` in the *Terminal* application to print the MD5 checksum for the file. You can also validate the signature using `pkgutil --check-signature R-3.6.1.pkg`.

Click here

[R-3.6.1.pkg](#)

MD5-hash: 279e6662103dfe6a625b4573143cb995

SHA1-

hash: 4e932f8e5013870d2a9179b54eae277f41657b0

(ca. 76MB)

Latest

R 3.6.1 binary for OS X

Contains R 3.6.1 framework

Tcl/Tk 8.6.6 X11 libraries

are optional and can be

are only needed if you want

package documentation

For Windows, you'll need to click on the 'base' version, and then click again to start the download.



CRAN

[Mirrors](#)

[What's new?](#)

[Task Views](#)

[Search](#)

About R

[R Homepage](#)

[The R Journal](#)

Software

[R Sources](#)

[R Binaries](#)

[Packages](#)

[Other](#)

Documentation

[Manuals](#)

[FAQs](#)

[Contributed](#)

Subdirectories:

[base](#)

[contrib](#)

[old contrib](#)

[Rtools](#)

**click
here...**

Binaries for base distrib

Binaries of contributed C

There is also informatio

and corresponding envin

Binaries of contributed C

managed by Uwe Ligge

Tools to build R and R p

Windows, or to build R

Please do not submit binaries to CRAN. Pack
questions / suggestions related to Windows bi

You may also want to read the [R FAQ](#) and [R t](#)

Note: CRAN does some checks on these bina
downloaded executables.



[*CRAN*](#)

[Mirrors](#)

[What's new?](#)

[Task Views](#)

[Search](#)

[Download R 3.6.1 for](#)

[Installation and other inst](#)

[New features in this versi](#)

If you want to double-check t
compare the [md5sum](#) of the .
both [graphical](#) and [command](#)

Once you've installed R, you'll need to install RStudio. Go to the RStudio website and install the latest version of RStudio Desktop (open source license).

Once they're both installed, get started by opening RStudio.

1.4 Packages

R comes with a lot of functions - commands - built in to do a broad range of data tasks. You could, if you really wanted, import a dataset, clean it up, estimate a model, and make a plot all using the functions that come with R - known as 'base R'².

But a lot of our work at Grattan uses add-on software to base R, known as 'packages'. Some packages, like the popular 'dplyr', make it quicker and/or easier to do tasks that you could otherwise do in base R. Other packages expand the possibilities of what R can do - like fitting a machine learning model, for example.

Like R itself, packages are free and open source. You can install them from within RStudio.

²Technically some of the 'built-in' functions are part of packages, like the `tools`, `utils` and `stats` packages that come with R. We'll refer to all these as base R.

At Grattan, we make heavy use of a set of related packages known collectively as the `tidyverse`. We'll cover these more in a later chapter.

1.4.1 Installing packages

You'll typically install packages using the console in RStudio. That's the part of the window that, by default, sits in the bottom-left corner of the screen.

In our work at Grattan, we use packages from two different source: CRAN and Github. The main difference you need to know about is that we use different commands to install packages from these two sources.

To install a package from CRAN, we use the command `install.packages()`.

For example, this code will install the `ggplot2` package from CRAN:

```
install.packages("ggplot2")
```

To install a package from Github, we use the function `install_github()`. Unfortunately, this package doesn't come with R - it's part of the `devtools` package. First, we install `devtools` from CRAN:

```
install.packages("devtools")
```

Now we can install packages from Github using the `install_github()` function from the `devtools` package. For example, here's how we would install the Grattan `ggplot2` theme, which we'll discuss later in this website:

```
devtools::install_github("mattcowgill/grattantheme", dependencies = TRUE)
```

1.4.2 Using packages

Before using a function that comes from a package, as opposed to base R, you need to tell R where to look for the function. There are two main ways to do that.

We can either load (aka 'attach') the package by using the `library()` function:

```
library(devtools)
```

Now that the `devtools` package is loaded, we can use its `install_github()` function:

```
install_github("mattcowgill/grattantheme")
```

Or, we can use two colons - `::` - to tell R to use an individual function from a package without loading it:

```
devtools::install_github("mattcowgill/grattantheme")
```

It usually makes sense to load a package with `library()`, unless you only need to use one of its function once or twice. There's no harm to using the `::` operator even if you have already loaded a package with `library()`. This can remove ambiguity both for R and for humans reading your code, particularly if you're using an obscure function - it makes it clearer where the function comes from.

Chapter 2

Why use R?

We can break this question into two parts: 1. Why use script-based software to analyse data? 2. Why use R, specifically?

2.1 Why use script-based software?

1. Make your analysis reproducible by setting out the complete series of steps taken from raw data to final output.
2. Work with large data sets.

2.2 Why use R specifically?

```
library(tidyverse)
```


Chapter 3

Using R at Grattan

3.1 Using R projects for a fully reproducible workflow.

Finally adhering to the ‘hit by a bus’ rule.

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.

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

3.1.1 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_csv("C:\\Users\\mcowgill\\Desktop\\hes1516.csv")
grattan_save("/Users/mcowgill/Desktop/images/expenditure_by_income.pdf")
```

3.1.2 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.

3.1.3 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.

3.2 Grattan coding style guide

Short summary of why

[Link to style guide](#)

3.3 What is the tidyverse and why do we use it?

Introduce following chapters

3.4 An introduction to RMarkdown

3.5 Resources in this package

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

Chapter 4

Data Visualisation

This chapter explores

4.1 Introduction to data visualisation

Data visualisation is used in two broad ways:

1. to examine and explore your data; and
2. to present a finding to your audience.

When you start using a dataset, you should *look at it*.¹ Plot histograms of variables-of-interest to spot outliers. Explore correlations with scatter plots and lines-of-best-fit. Check how many observations are in particular groups with bar charts. Identify variables that have missing or coded-missing values. Use faceting to explore differences in the above between groups, and do it interactively with non-static plots.

These **exploratory plots** are just for you and your team. They don't need to be perfectly labelled, the right size, in the Grattan palette or be particularly interesting. They're built and used to explore the data. Through this process, you can become confident your data is *what it says it is*.

When you **present a visualisation to a reader**, you make decisions about what they can and cannot see. You choose to highlight or omit particular things to help them better understand the message you are presenting.

¹From Kieran Healy's *Data Visualization: A Practical Introduction* (available free): 'You should look at your data. Graphs and charts let you explore and learn about the structure of the information you collect. Good data visualizations also make it easier to communicate your ideas and findings to other people.'

This requires important technical decisions: what data to use, what ‘stat’ to present it with — show every data point, show a distribution function, show the average or the median — and on what scale — raw numbers, on a log scale, as a proportion of a total.

It also requires *aesthetic* decisions. What colours in the Grattan palette would work best? Where should the labels be placed and how could they be phrased to succinctly convey meaning? Should data points be represented by lines, or bars, or dots, or balloons, or shades of colour?

All of these decisions need to be made with two things in mind:

1. Rigour, accuracy, legitimacy: the chart needs to be honest.
2. The reader: the chart needs to help the reader understand something, and it must convince them to pay attention.

At the margins, sometimes these two ideas can be in conflict: maybe a 70-word definition in the middle of your chart would improve its technical accuracy, but it could confuse the average reader.

Similarly, a bar chart is often the safest way to display data. But if the reader has stopped paying attention by your sixth consecutive bar chart, your point loses its punch.²

The way we design charts — much like our writing — should always be honest, clear and engaging to the reader.

This chapter shows how you can do this with R. It starts with the ‘grammar of graphics’ concepts of a package called `ggplot`, explains how to make those charts ‘Grattan-y’, then provides examples for all common (and some not-so-common) charts you should add to your box of data visualisation tools to impress your message on our readers.

4.2 Set-up and packages

This section uses the package `ggplot2` to visualise data, and `dplyr` functions to manipulate data. Both of these packages are loaded with `tidyverse`. The `scales` package helps with labelling your axes.

The `grattantheme` package is used to make charts look Grattan-y. The `absmapsdata` package is used to help make maps.

```
library(tidyverse)
library(grattantheme)
library(ggrepel)
library(absmapsdata)
```

²‘Bar charts are evidence that you are dead inside’ — Amanda Cox, data editor for the New York Times.

```
library(sf)
library(scales)
```

For most charts in this chapter, we'll use the `sa3_income` data summarised below³ It is a long dataset containing the median income and number of workers by SA3, occupation and sex between 2010 and 2015. We will also create a `professionals` subset that only includes people in professional occupations in 2015:

```
sa3_income <- read_csv("data/sa3_income.csv") %>%
  filter(!is.na(median_income),
         sex != "Persons")

professionals <- sa3_income %>%
  filter(year == 2015,
         occupation == "Professionals")

# Show the first six rows of the new dataset
head(sa3_income)
```

```
## # A tibble: 6 x 10
##   sa3 sa3_name sa3_sqkm sa4_name gcc_name occupation sex   year
##   <dbl> <chr>      <dbl> <chr>    <chr>    <chr>    <chr> <dbl>
## 1 10102 Queanbe~    6511. Capital~ Rest of~ Clerical ~ Fema~ 2010
## 2 10102 Queanbe~    6511. Capital~ Rest of~ Clerical ~ Fema~ 2011
## 3 10102 Queanbe~    6511. Capital~ Rest of~ Clerical ~ Fema~ 2012
## 4 10102 Queanbe~    6511. Capital~ Rest of~ Clerical ~ Fema~ 2013
## 5 10102 Queanbe~    6511. Capital~ Rest of~ Clerical ~ Fema~ 2014
## 6 10102 Queanbe~    6511. Capital~ Rest of~ Clerical ~ Fema~ 2015
## # ... with 2 more variables: median_income <dbl>, persons <dbl>
```

4.3 Concepts

The `ggplot2` package is based on the `grammar of graphics`. ...

The main ingredients to a `ggplot` chart are:

- **Data:** what data should be plotted.
 - e.g. `data`
- **Aesthetics:** what variables should be linked to what chart elements.
 - e.g. `aes(x = population, y = age)` to connect the `population` variable to the `x` axis, and the `age` variable to the `y` axis.
- **Geoms:** how the data should be plotted.

³From (ABS Employee income by occupation and sex, 2010-11 to 2015-16)[<https://www.abs.gov.au/AUSSTATS/abs@.nsf/DetailsPage/6524.0.55.0022011-2016?OpenDocument>]

- e.g. `geom_point()` will produce a scatter plot, `geom_col` will produce a column chart, `geom_line()` will produce a line chart.

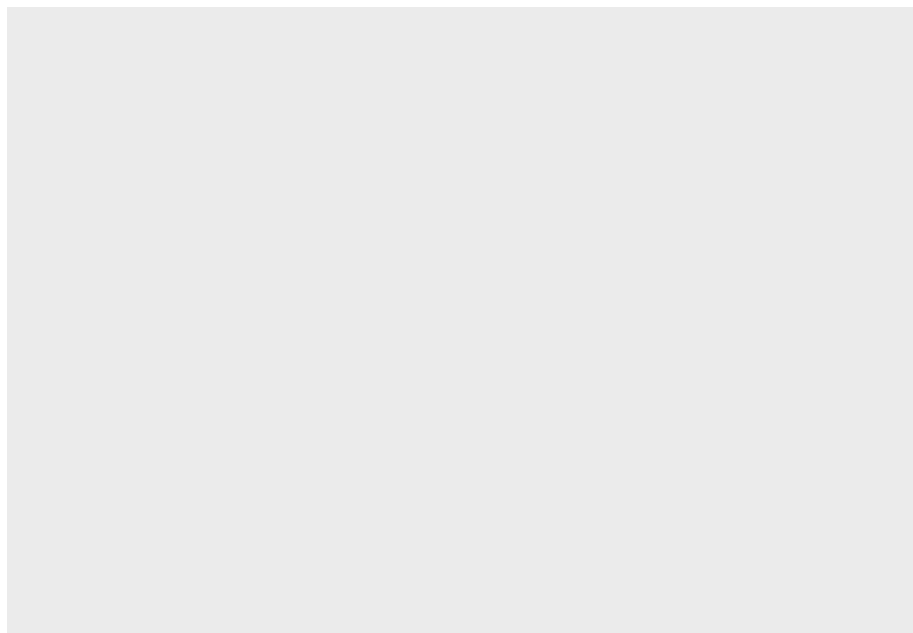
Each plot you make will be made up of these three elements. The full list of standard geoms is listed in the **tidyverse** documentation.

ggplot also has a ‘cheat sheet’ that contains many of the often-used elements of a plot, which you can download [here](#).



For example, you can plot a column chart by passing the `sa3_income` dataset into `ggplot()` (“make a chart with this data”). This completes the first step – data – and produces an empty plot:

```
professionals %>%  
  ggplot()
```

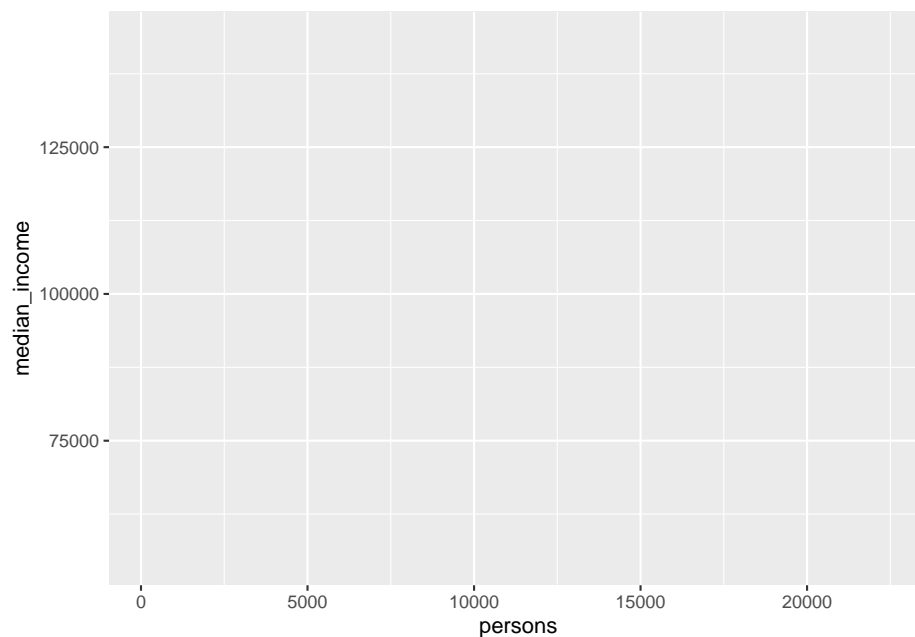


Next, set the **aes** (aesthetics) to `x = state` (“make the x-axis represent state”), `y = pop` (“the y-axis should represent population”), and `fill = year` (“the fill colour represents year”). Now **ggplot** knows where things should *go*.

If we just plot that, you’ll see that **ggplot** knows a little bit more about what we’re trying to do. It has the states on the x-axis and range of populations on

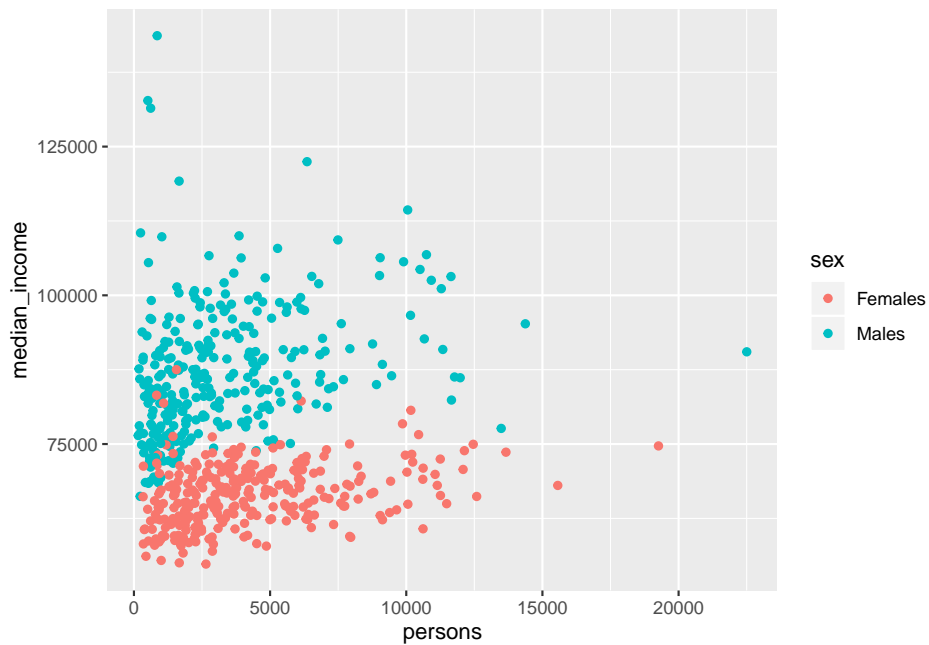
the y-axis:

```
professionals %>%  
  ggplot(aes(x = persons,  
             y = median_income,  
             colour = sex))
```



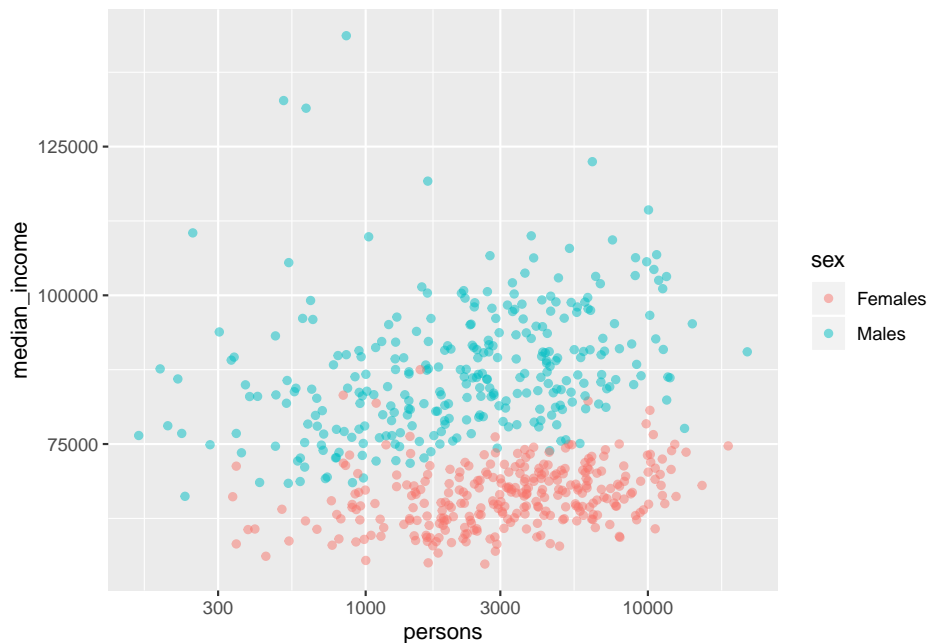
Now that `ggplot` knows where things should go, it needs to how to *plot* them on the chart. For this we use `geoms`. Tell `ggplot` to take the things it knows and plot them as a column chart by using `geom_col`:

```
professionals %>%  
  ggplot(aes(x = persons,  
             y = median_income,  
             colour = sex)) +  
  geom_point()
```



Great! There are a couple of quick things we can do to make the chart a bit clearer. There are points for each group in each year, which we probably don't need. So filter the data before you pass it to `ggplot` to just include 2015: `filter(year == 2015)`. There will still be lots of overlapping points, so set the opacity to below one with `alpha = 0.5`. The `persons` x-axis can be changed to a log scale with `scale_x_log10`.

```
professionals %>%  
  ggplot(aes(x = persons,  
             y = median_income,  
             colour = sex)) +  
  geom_point(alpha = .5) +  
  scale_x_log10()
```



That looks a bit better. The following sections in this chapter will cover a broad range of charts and designs, but they will all use the same building-blocks of `data`, `aes`, and `geom`.

The rest of the chapter will explore:

- Exploratory data visualisation
- Grattanising your charts and choosing colours
- Saving charts according to Grattan templates
- Making bar, line, scatter and distribution plots
- Making maps and interactive charts
- Adding chart labels

4.4 Exploratory data visualisation

Plotting your data early in the analysis stage can help you quickly identify outliers, oddities, things that don't look quite right.

4.5 Making Grattan-y charts

The `grattantheme` package contains functions that help *Grattanise* your charts. It is hosted here: <https://github.com/mattcowgill/grattantheme>

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

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

The key functions of `grattantheme` are:

- `theme_grattan`: set size, font and colour defaults that adhere to the Grattan style guide.
- `grattan_y_continuous`: sets the right defaults for a continuous y-axis.
- `grattan_colour_continuous`: pulls colours from the Grattan colour palette for colour aesthetics.
- `grattan_fill_continuous`: pulls colours from the Grattan colour palette for fill aesthetics.
- `grattan_save`: a save function that exports charts in correct report or presentation dimensions.

This section will run through some examples of *Grattanising* charts. The `ggplot` functions are explored in more detail in the next section.

4.5.1 Making Grattan charts

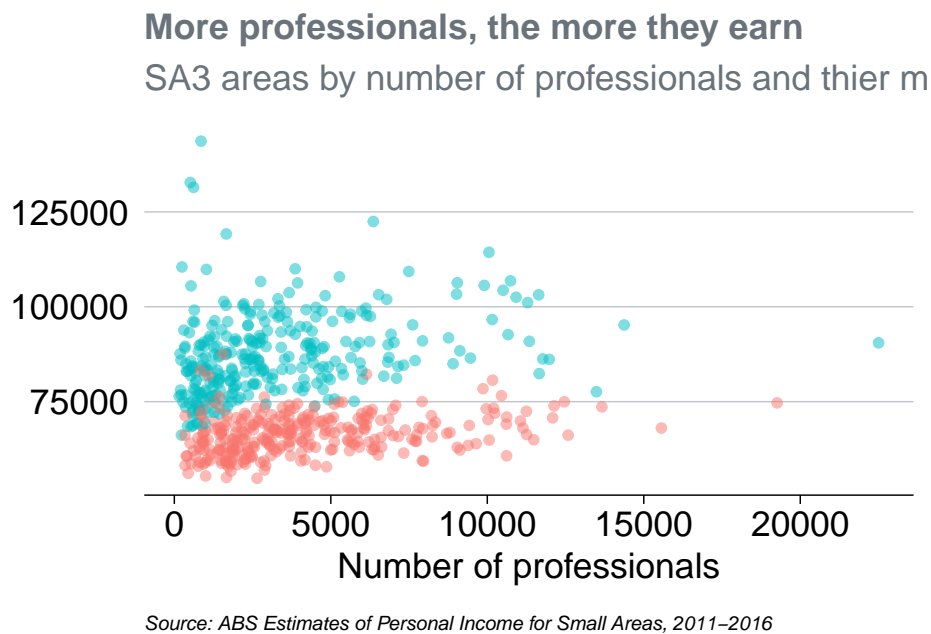
Start with a scatterplot, similar to the one made above:

```
base_chart <- professionals %>%
  ggplot(aes(x = persons,
             y = median_income,
             colour = sex)) +
  geom_point(alpha = .5) +
  labs(title = "More professionals, the more they earn",
       subtitle = "SA3 areas by number of professionals and thier median income",
       x = "Number of professionals",
       y = "Median income",
       caption = "Source: ABS Estimates of Personal Income for Small Areas, 2011")
base_chart
```




Let's make it Grattany. First, add `theme_grattan` to your plot:

```
base_chart +  
  theme_grattan()
```



Then `grattan_y_continuous` to align the x-axis with zero. This function

takes the same arguments as `scale_y_continuous`, so you can add `labels = comma()` to reformat the y-axis labels:

```
base_chart +
  theme_grattan() +
  grattan_y_continuous(labels = dollar) +
  scale_x_log10(labels = comma)
```



Source: ABS Estimates of Personal Income for Small Areas, 2011–2016

To define colour colours, use `grattan_colour_manual` with the number of colours you need (two, in this case):

```
prof_chart <- base_chart +
  theme_grattan() +
  grattan_y_continuous(labels = dollar) +
  scale_x_log10(labels = comma) +
  grattan_colour_manual(2)

prof_chart
```



Source: ABS Estimates of Personal Income for Small Areas, 2011–2016

Nice chart! Now you can save it and share it with the world.

4.5.2 Saving Grattan charts

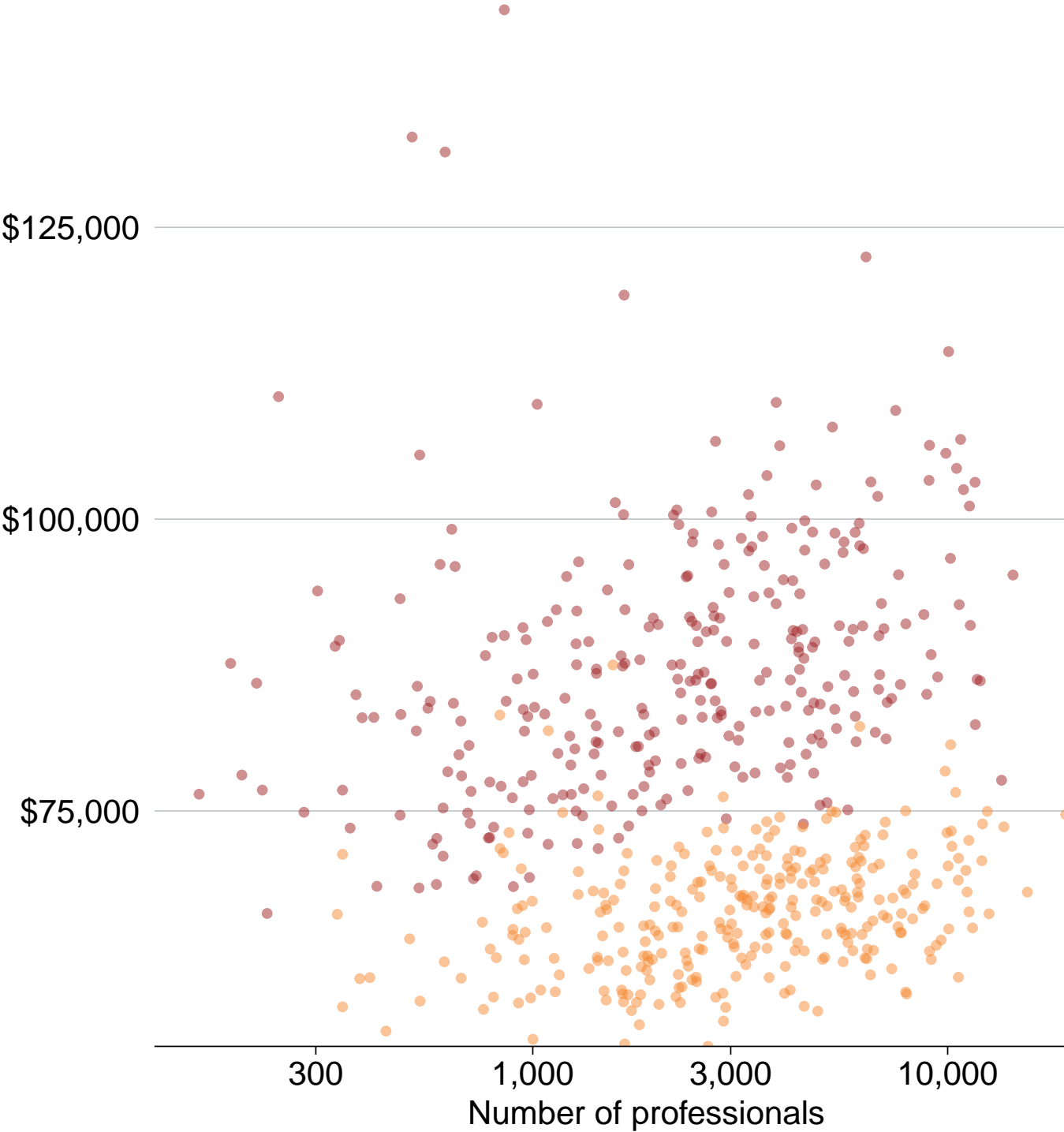
The `grattan_save` function saves your charts according to Grattan templates. It takes these arguments:

- **filename:** the path, name and file-type of your saved chart. eg: "atlas/professionals_chart.pdf".
- **object:** the R object that you want to save. eg: `prof_chart`. If left blank, it grabs the last chart that was displayed.
- **type:** the Grattan template to be used. This is one of:
 - "normal" The default. Use for normal Grattan report charts, or to paste into a 4:3 PowerPoint slide. Width: 22.2cm, height: 14.5cm.
 - "normal_169" Only useful for pasting into a 16:9 format Grattan PowerPoint slide. Width: 30cm, height: 14.5cm.
 - "tiny" Fills the width of a column in a Grattan report, but is shorter than usual. Width: 22.2cm, height: 11.1cm.
 - "wholecolumn" Takes up a whole column in a Grattan report. Width: 22.2cm, height: 22.2cm.
 - "fullpage" Fills a whole page of a Grattan report. Width: 44.3cm, height: 22.2cm.
 - "fullslide" Creates an image that looks like a 4:3 Grattan PowerPoint slide, complete with logo. Width: 25.4cm, height: 19.0cm.

- "fullslide_169" Creates an image that looks like a 16:9 Grattan PowerPoint slide, complete with logo. Use this to drop into standard presentations. Width: 33.9cm, height: 19.0cm
- "blog" Creates a 4:3 image that looks like a Grattan PowerPoint slide, but with less border whitespace than 'fullslide'."
- "fullslide_44" Creates an image that looks like a 4:4 Grattan PowerPoint slide. This may be useful for taller charts for the Grattan blog; not useful for any other purpose. Width: 25.4cm, height: 25.4cm.
- Set `type = "all"` to save your chart in all available sizes.
- `height`: override the height set by `type`. This can be useful for really long charts in blogposts.
- `save_data`: exports a `csv` file containing the data used in the chart.
- `force_labs`: override the removal of labels for a particular `type`. eg `force_labs = TRUE` will keep the y-axis label.

To save the `prof_chart` plot created above as a whole-column chart for a **report**:

```
grattan_save("atlas/professionals_chart_report.pdf", prof_chart, type = "wholecolumn")
```

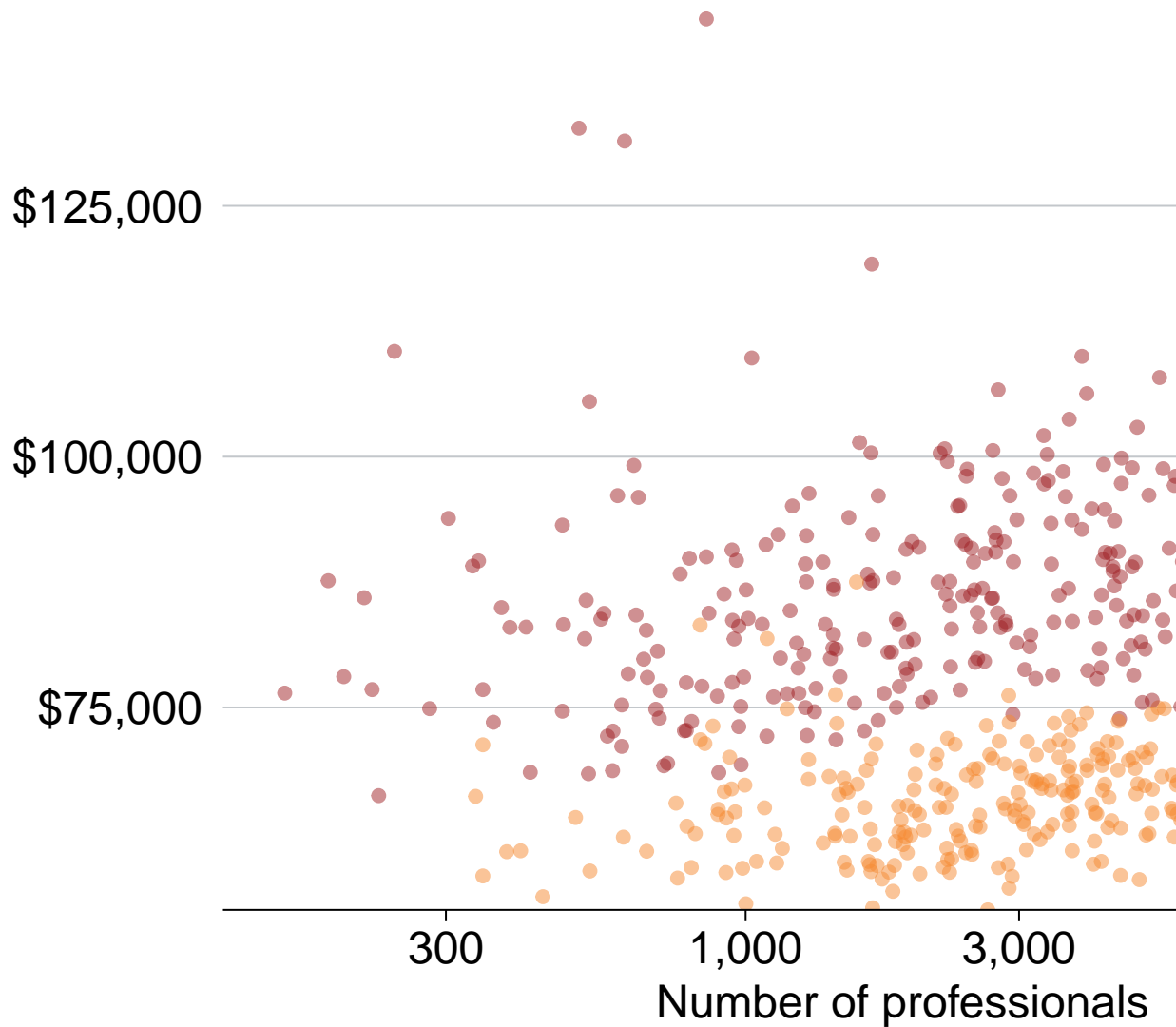


To save it as a **presentation** slide instead, use `type = "fullslide"`:

```
grattan_save("atlas/professionals_chart_presentation.pdf", prof_chart, type = "fullslide")
```

More professionals, the more they earn

SA3 areas by number of professionals and thier median in



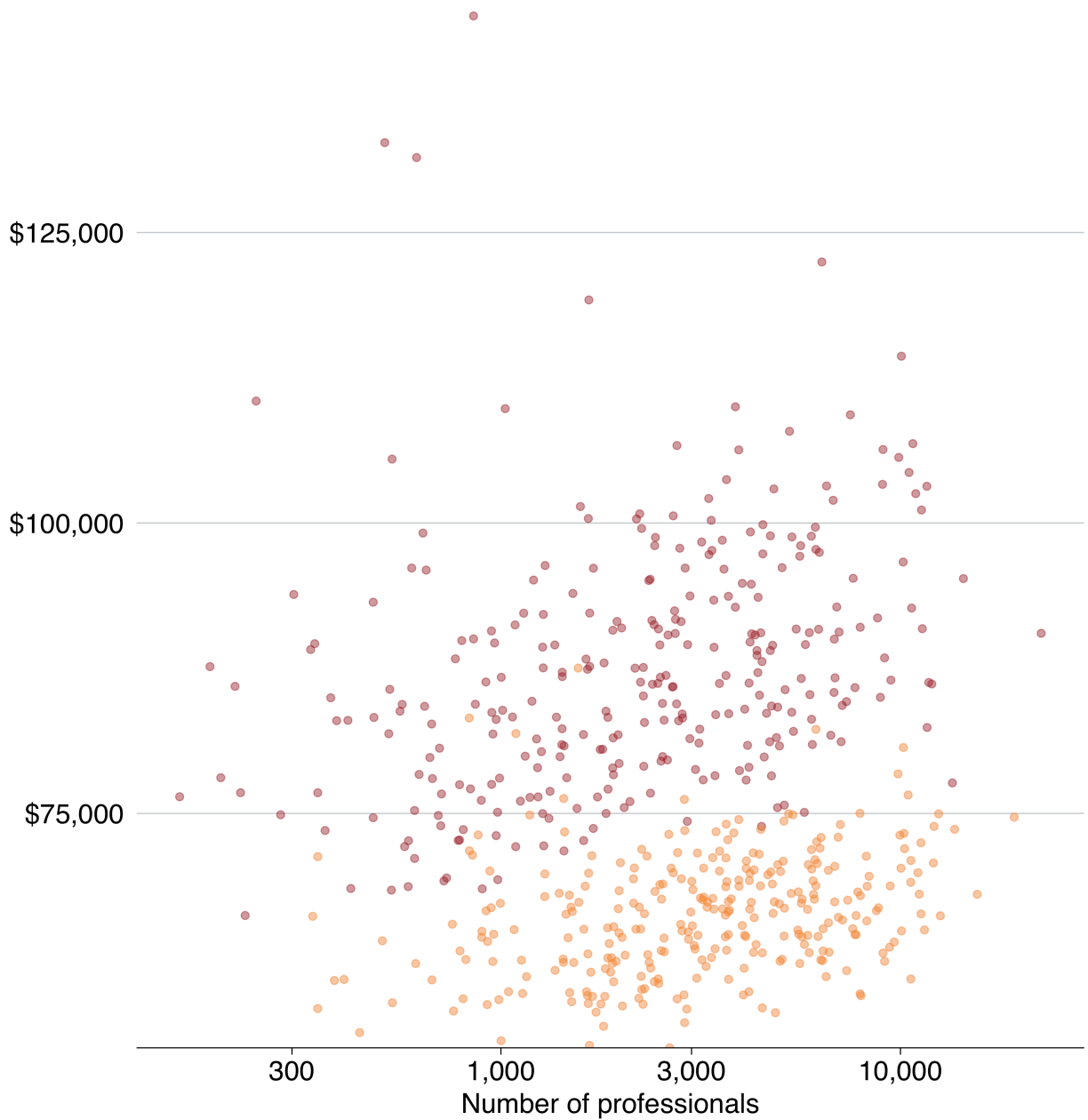
Source: ABS Estimates of Personal Income for Small Areas, 2011–2016

Or, if you want to emphasise the point in a *really tall* chart for a **blogpost**, you can use `type = "blog"` and adjust the `height` to be 50cm. Also note that because this is for the blog, you should save it as a `png` file:

```
grattan_save("atlas/professionals_chart_blog.png", prof_chart,  
             type = "blog", height = 30)
```


More professionals, the more they earn

SA3 areas by number of professionals and their median income

*Source: ABS Estimates of Personal Income for Small Areas, 2011-2016*

And that's it! The following sections will go into more detail about different chart types in R, but you'll mostly use the same basic **grattan** formatting you've used here.

4.6 Adding labels

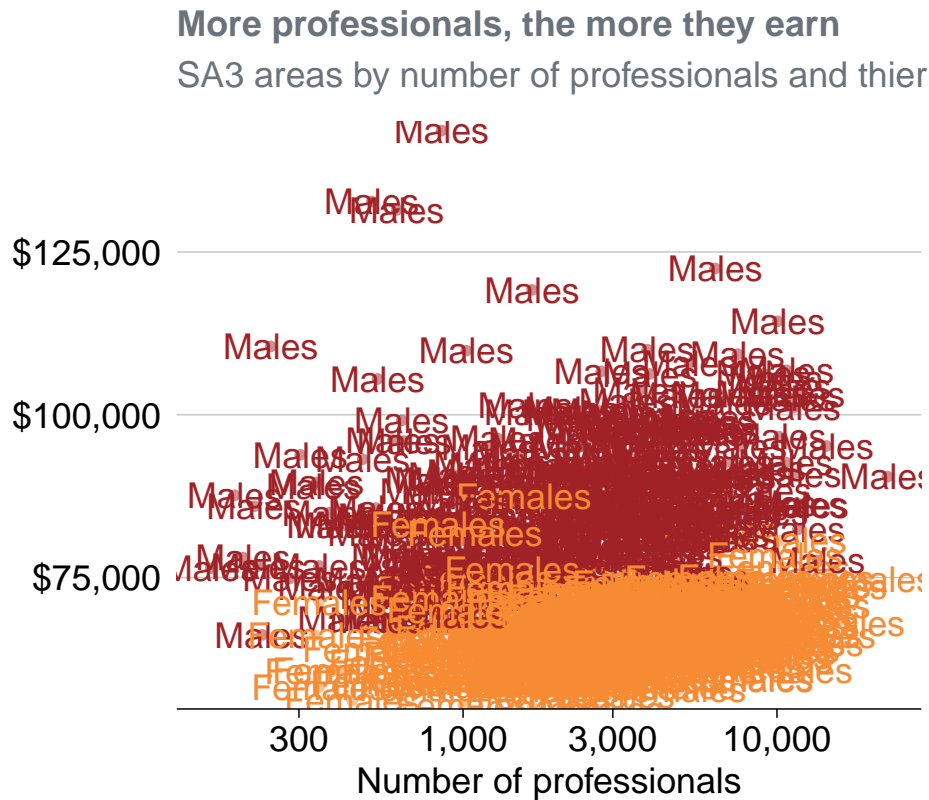
Labels can be a bit finicky – especially compared to labelling charts visually in PowerPoint. ...

Labels can be done in two broad ways:

1. Labelling every single data point on your chart. Grattan charts rarely do this.
2. Labelling some of the data points on your chart. This is how you label Grattan charts: label on item in a group and let the reader join the dots.

We'll look at the first approach so you can get a feel for how the labelling geoms – `geom_label` and `geom_text` (and some useful extensions) – work. It won't be pretty.

```
prof_chart +  
  geom_text(aes(label = sex))
```



Source: ABS Estimates of Personal Income for Small Areas, 2011–2016

Great! That looks *terrible*. `geom_text` is labelling each individual point because it has been told to do so. Just like `geom_point`, it takes the `x` and `y` aesthetics of each observation, then plots the `label` at that location. But we just want to label one of the points for `female` and one for `male`.

To do this, we can create a new dataset that just contains one observation each. Here, you're filtering the dataset to include *only* the female/male observations that have the most people:

```
label_data <- professionals %>%
  group_by(sex) %>%
  filter(persons == max(persons))
```

```
label_data
```

```
## # A tibble: 2 x 10
## # Groups:   sex [2]
##   sa3 sa3_name sa3_sqkm sa4_name gcc_name occupation sex   year
##   <dbl> <chr>      <dbl> <chr>   <chr>    <chr>    <chr> <dbl>
## 1 11703 Sydney ~    25.1 Sydney ~ Greater~ Professio~ Fema~ 2015
```

```
## 2 11703 Sydney ~      25.1 Sydney ~ Greater~ Professio~ Males  2015
## # ... with 2 more variables: median_income <dbl>, persons <dbl>
```

And then tell `geom_text` to look at *that* dataset:

```
prof_chart +
  geom_text(data = label_data,
            aes(label = sex))
```



Source: ABS Estimates of Personal Income for Small Areas, 2011–2016

Chapter 5

Reading data

5.1 Importing data

5.1.1 Reading CSV files

5.1.1.1 `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.

5.1.1.2 `data.table::fread()`

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

5.1.2 `readxl::read_excel()`

5.1.3 `rio`

5.1.4 `readabs`

5.2 Reading common files:

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

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

5.3 Appropriately renaming variables

As shown in the style guide

Add `rename_abs` function to a common Grattan package?

5.4 Getting to tidy data

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

Chapter 6

Different data types

6.1 Tidy data

Other data structures

6.2 Dates with `lubridate::`

The `lubridate::` package

6.3 Strings with `stringr::`

- Replacing values
- Matching values
- Separating columns

6.4 Factors with `forcats::`

- Dangers with factors

Chapter 7

Data transformation

7.1 The pipe

7.2 Key dplyr functions:

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

7.3 Filter with `filter()`

7.4 Arrange with `arrange()`

7.5 Select variables with `select()`

7.6 Group data with `group_by()`

7.7 Edit and add new variables with `mutate()`

7.7.1 Cases when you should use `case_when()`

7.8 Summarise data with `summarise()`

7.9 Joining datasets with `*_join()`

Chapter 8

Analysis

Chapter 9

Creating functions

9.1 It can be useful to make your own function

Why on earth would you create your own function?

9.2 Defining simple functions

9.3 More complex functions

9.4 Sets of functions

9.5 Using `purrr::map`

9.6 Sharing your useful functions with Grattan

Chapter 10

Version control

10.1 Version control is important and intimidating

Version control is great!

10.2 Github

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

10.3 Git

Using Git within R Studio...