

ETC1010: Introduction to Data Analysis

Week 1, part B

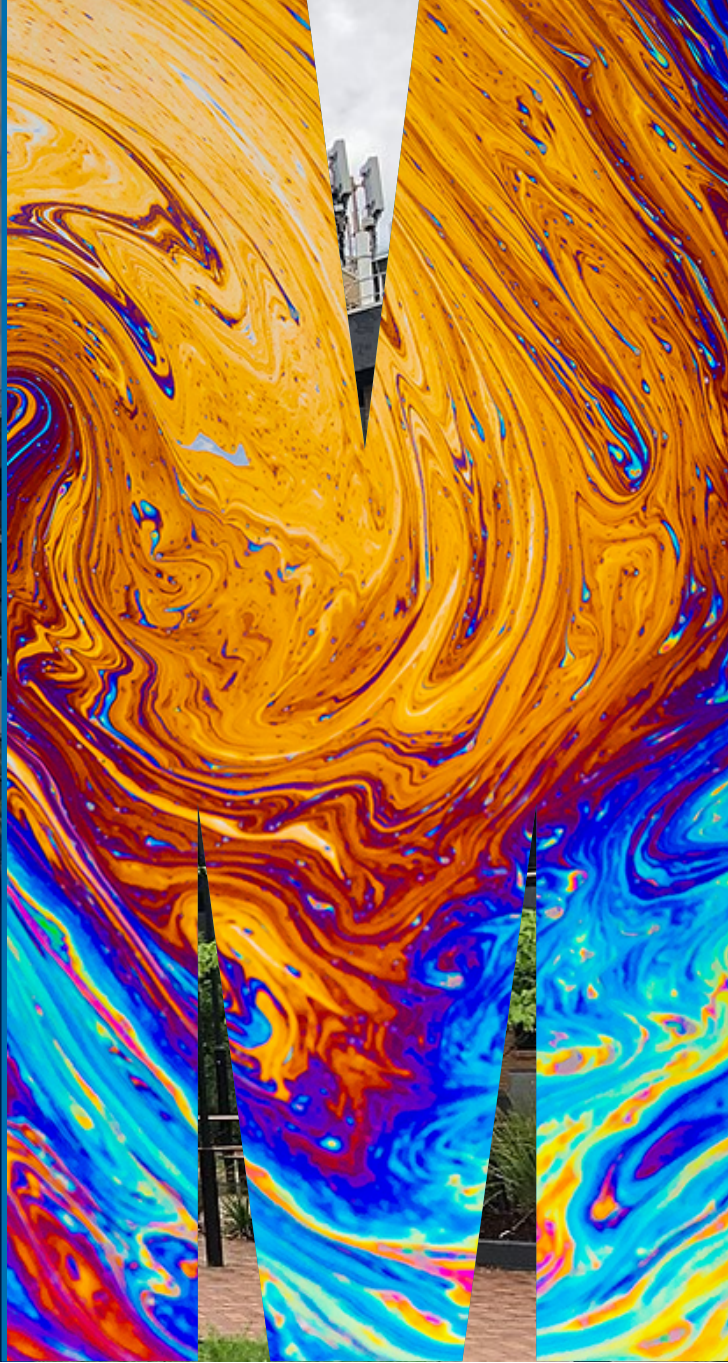
Week of introduction

Lecturer: *Nicholas Tierney*

Department of Econometrics and Business Statistics

✉ ETC1010.Clayton-x@monash.edu

11th Mar 2020



The screenshot shows the RStudio interface with several panels and annotations:

- Source Editor:** Contains R code for a 'Yummy pasta recipe'.

```
1 # Yummy pasta recipe -----
2 |
3 |
4 # get out the equipment we need (load the packages)
5 library(saucepan)
6 library(colander)
7
8 # get the ingredients out on to the counter (load data)
9 pasta<- read_csv("pasta.csv")
10 cheese<- read_csv("cheese.csv")
11 sauce<- read_csv("yummy_sauce.csv")
12 water<- read_csv("tap_water.csv")
13
14 #cook pasta then drain it and then add the cheese and the sauce
15 cooked_pasta<- Saucepan(pasta + water)
16 drained_pasta<-colander(cooked_pasta)
17 yummy_pasta <- c(drained_pasta, cheese, sauce)
18
19
```
- Environment Panel:** Shows 'Global Environment'.
- Files Panel:** Shows a file explorer.
- Console:** Shows the execution of the first few lines of the script.

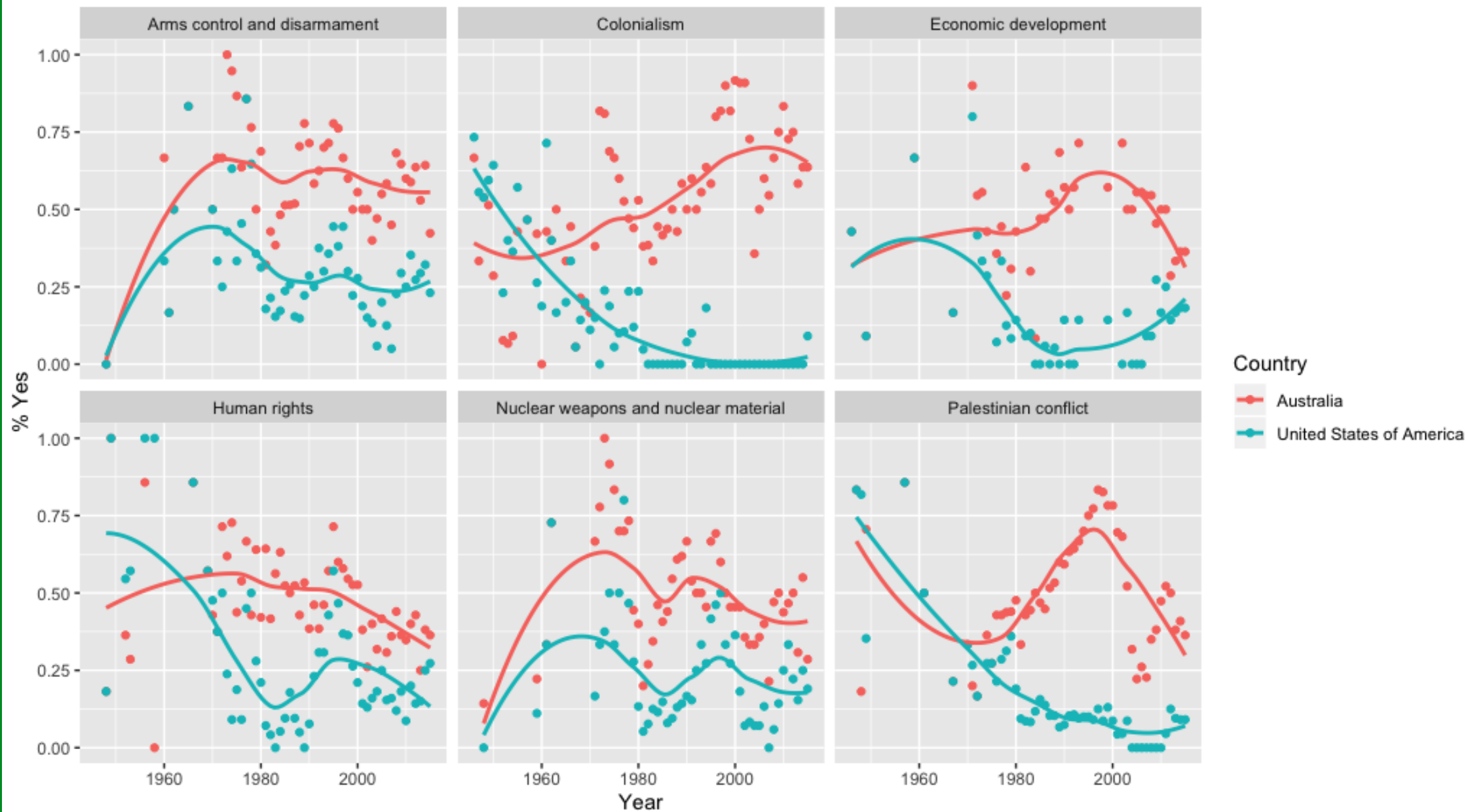
Annotations:

- Scripts are recipes – records of how to do things**
Write and save your recipes here so that R knows what to cook
- The environment is like the kitchen counter**
you can put ingredients(data) and finished dishes (model outputs) here to use while you cook
- Files are like ingredients in your cupboards –**
you need to get them out on to the kitchen counter (the environment) to use them.
The files that you need can be specified in the recipe so you know exactly what you need to get out
- Packages are like tools –**
when you need to use a saucepan you go out and buy one that someone has already designed and made (install.packages())
Each time you want to use that pan you just take it out of the cupboard (library())
- The console is where the cooking happens**
Send recipes here (run code) to cook them
You can cook here without using a recipe but you'll struggle to remember exactly how to recreate the dish in the future so it's better to use a recipe

From Jessica Ward (@JKRWard) of R Ladies Newcastle (UK) - @RLadiesNCL
<https://twitter.com/RLadiesNCL/status/1138812826917724160>

Percentage of 'Yes' votes in the UN General Assembly

1946 to 2015



New Thread

Search

Cancel

New Question

Post

- FILTERS
- All
 - Unread
 - Starred
 - Answered
 - Unanswered
 - Staff
 - Private

Pinned

Welcome to ETC1010!

General Nick Tierney STAFF 1d

- CATEGORIES
- General
 - Lectures
 - Tutorials
 - Quizzes
 - Assignments
 - Final Exam

Title

Input field for title

Type

Question Post Announcement

Category

General Lectures Tutorials Quizzes Assignments Final Exam

Paragraph

B *I* U <>

Rich text editor area


Pinned
Keep at top of thread list

Private
Visible to you and staff only

Anonymous
Hide your name from students

Post


R essentials: A short list (for now)

 Functions are (most often) verbs, followed by what they will be applied to in parentheses:

```
do_this(to_this)
```








 Columns (variables) in data frames are accessed with \$:

```
dataframe$var_name
```

 Packages are installed with `install.packages`, and loaded with `library`, once per session:

```
install.packages("package_name")  
library(package_name)
```

Today: Outline

-  Why we care about Reproducibility
-  R + markdown = Rmarkdown
-  Controlling output and input of rmarkdown
-  Exercises on creating rmarkdown reports on the humble platypus
-  Form up assignment groups
-  Quiz
-  Assignment 1 released next week.

We are in a tight spot with reproducibility

Only 6 out of 53 landmark results could be reproduced
-- Amgen, 2014*

An estimated 75% - 90% of preclinical results cannot be reproduced

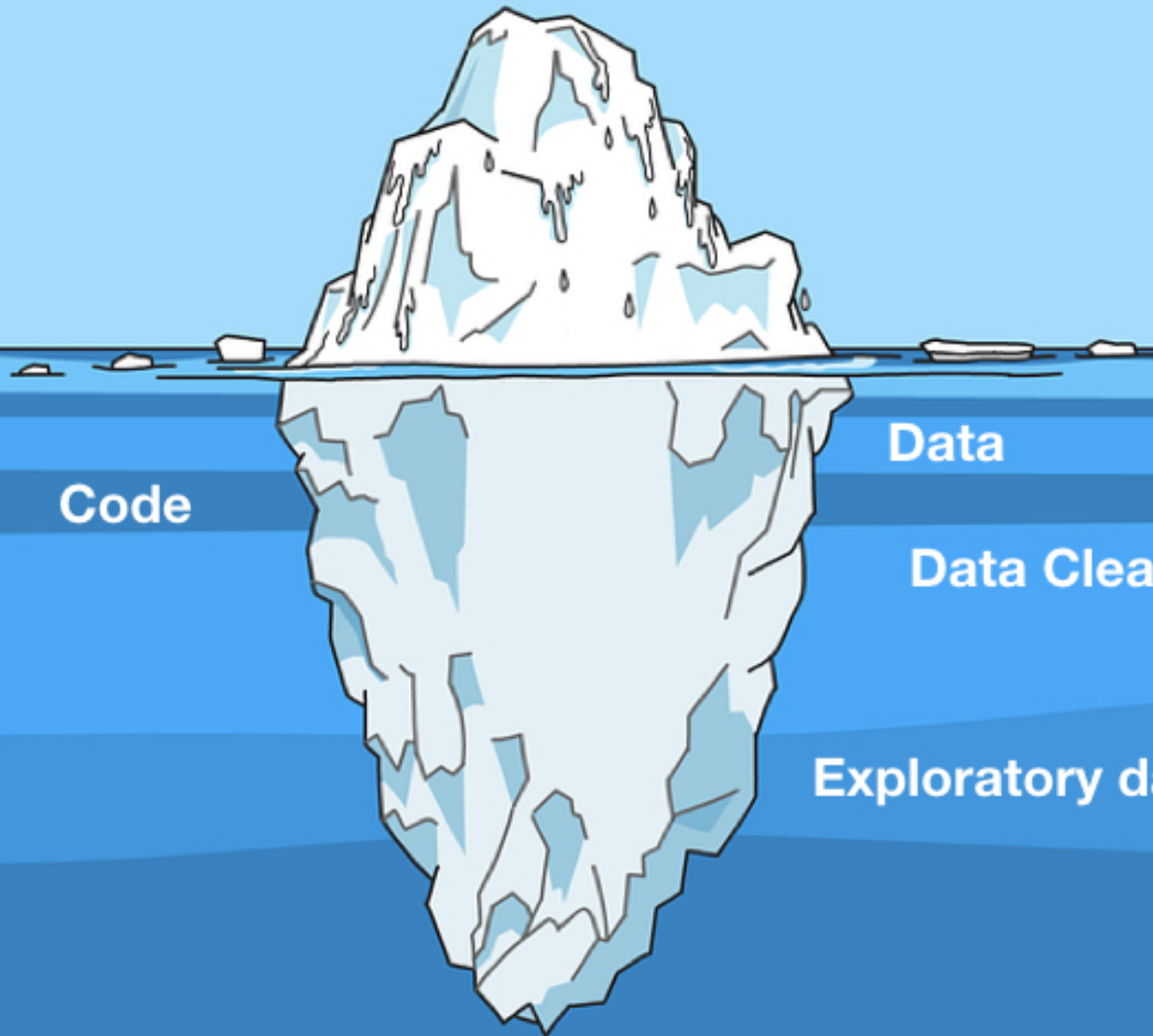
-- Begley, 2015*

* Heard via Garret Grolemond's great talk





The paper



Code

Data




Data Cleaning

Exploratory data analysis



**So what can we do about
it?**

Reproducibility checklist

Near-term goals:

-  Are the tables and figures reproducible from the code and data?
-  Does the code actually do what you think it does?
-  In addition to what was done, is it clear **why** it was done? (e.g., how were parameter settings chosen?)

Long-term goals:

-  Can the code be used for other data?
-  Can you extend the code to do other things?

Literate programming is a partial solution



Literate programming shines some light on this dark area of science.



An idea from [Donald Knuth](#) where you combine your text with your code output to create a document.



A *blend* of your literature (**text**), and your programming (**code**), to create something you can read from top to bottom.

Imagine a report:

Introduction, methods, results, discussion, and conclusion,

All the bits of code that make each section.

With rmarkdown, you can see all the pieces of your data analysis all together.

Each time you knit the analysis is ran from the beginning

Markdown as a new player to legibility

In 2004, [John Gruber](#), of [daring fireball](#) created [markdown](#), a simple way to create text that rendered into a HTML webpage.

Markdown as a new player to legibility

```
- bullet list  
- bullet list  
- bullet list
```

 bullet list

 bullet list

 bullet list

Markdown as a new player to legibility

1. numbered list
2. numbered list
3. numbered list

`__bold__`, `**bold**`,

`_italic_`, `*italic*`

`> quote of something profound`

1. numbered list
2. numbered list
3. numbered list

bold, bold,

italic, italic

*quote of
something
profound*

Markdown as a new player to legibility

With very little marking up, we can create rich text, that **actually resembles** the text that we want to see.

Learn to use markdown Spend five minutes working through [markdowntutorial.com](https://www.markdowntutorial.com)

This the end of part 1 of lecture 1b

05 : 00

Start of part 2 of lecture 1b

Rmarkdown helps address the reproducibility problem



Q: How do we take markdown + R code = "literate programming environment"



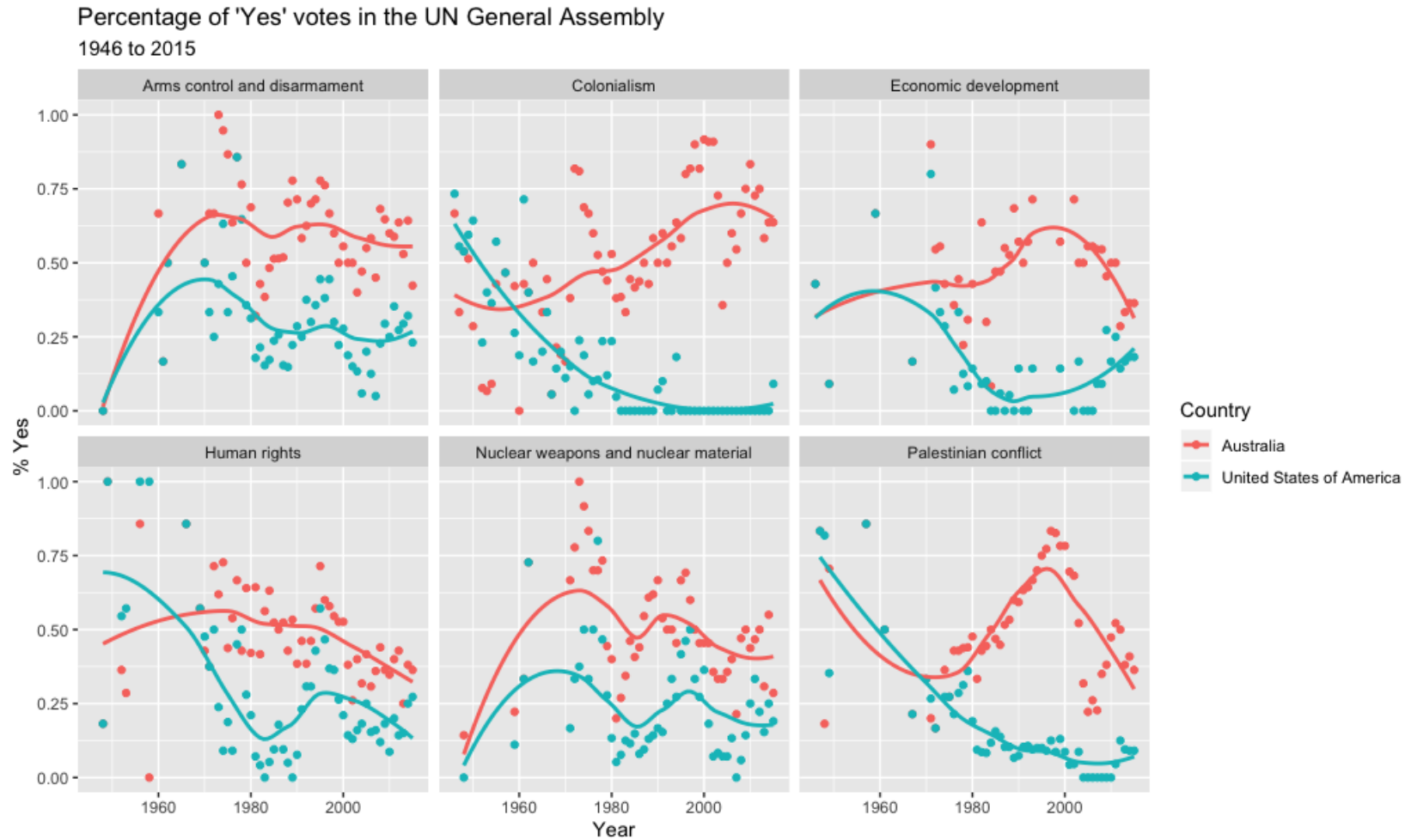
A: Rmarkdown

Rmarkdown...

Provides an environment where you can write your complete analysis, and marries your text, and code together into a rich document.

You write your code as code chunks, put your text around that, and then hey presto, you have a document you can reproduce.

Reminder: You've already used rmarkdown!






How will we use R Markdown?

- ▮ Every assignment + project / is an R Markdown document
- ▮ You'll always have a template R Markdown document to start with
- ▮ The amount of scaffolding in the template will decrease over the semester
- ▮ These lecture notes are created using R Markdown (!)

The anatomy of an rmarkdown document

There are three parts to an rmarkdown document.

-  Metadata (YAML)
-  Text (markdown formatting)
-  Code (code formatting)

DEMO

Metadata: YAML (YAML Ain't Markup Language)




The metadata of the document tells you how it is formed - what the **title** is, what **date** to put, and other control information.



If you're familiar with *TEX*, this is similar to how you specify document type, styles, fonts, options, etc in the front matter / preamble.

Metadata: YAML

 Rmarkdown documents use YAML to provide the metadata. It looks like this:

```
---  
title: "An example document"  
author: "Nicholas Tierney"  
output: html_document  
---
```

It starts and ends with three dashes ---, and has fields like the following: `title`, `author`, and `output`.

Text

Is markdown, as we discussed in the earlier section,
It provides a simple way to mark up text

```
1. bullet list  
2. bullet list  
3. bullet list
```

1. bullet list
2. bullet list
3. bullet list

Code

We refer to code in an rmarkdown document in two ways:

1. Code chunks, and
2. Inline code.

Code: Code chunks

Code chunks are marked by three backticks and curly braces with `r` inside them:

```
```{r chunk-name}  
a code chunk
```
```

Code: backtick?

A **backtick** is a special character you might not have seen before, it is typically located under the tilde key (~). On USA / Australia keyboards, is under the escape key:

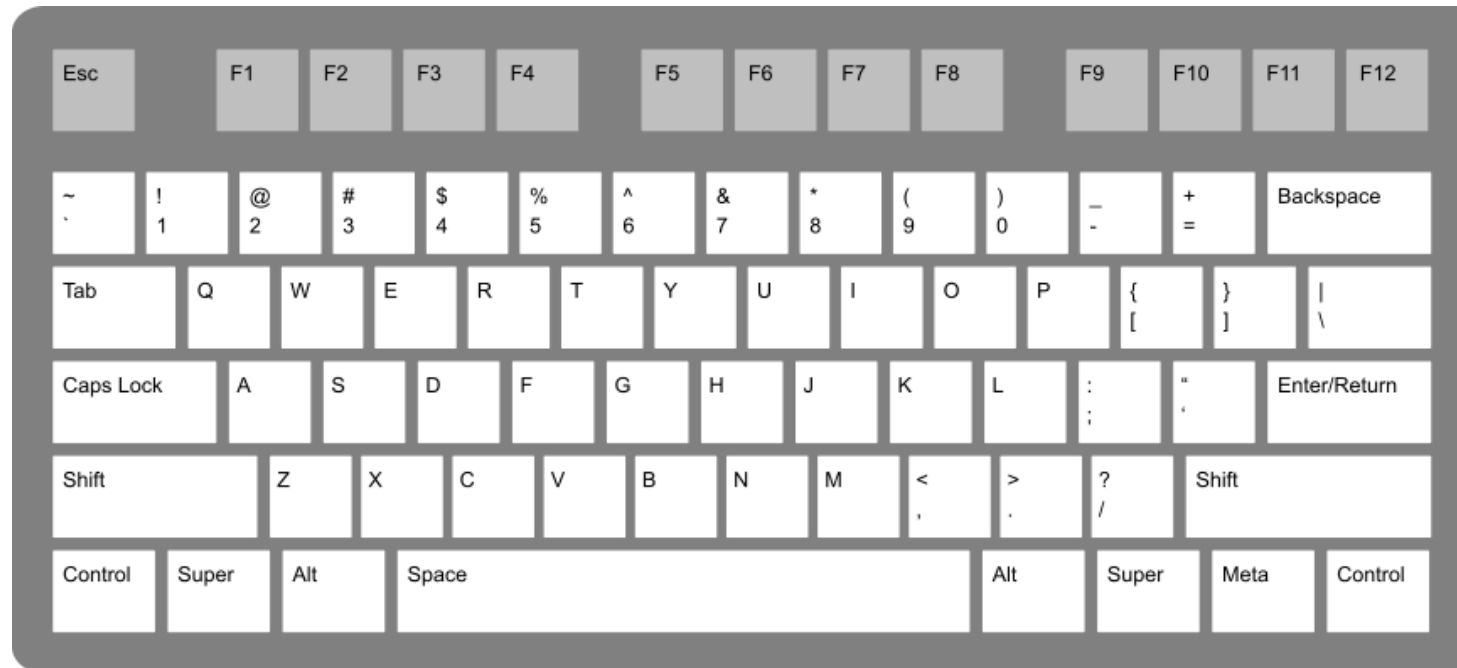


image from

https://commons.wikimedia.org/wiki/File:ANSI_Keyboard_Layout_Diagram_with_Form_Factor.svg

Code: Inline code

Sometimes you want to run the code inside a sentence. This is called running the code "inline".

You might want to run the code inline to name the number of variables or rows in a dataset in a sentence like:

There are XXX observations in the airquality dataset, and XXX variables.

Code: Inline code

You can call code "inline" like so:

```
There are `r nrow(airquality)` observations in the airquality dataset,  
and `r ncol(airquality)` variables.
```

Which gives you the following sentence

There are 153 observations in the airquality dataset, and 6 variables.

Code: Inline code



If your data changes upstream



You don't need to work out where you mentioned your data




You just update the document. 🎉

Your Turn: Put it together

Go to `rstudio.cloud` and go to "ida-exercise-1b"

 open the document "01-oz-atlas.Rmd"

 knit the document

 Change the data section at the top to be from a different state instead of "New South Wales"

 knit the document again

 How do the text and figures in the document change?

05 : 00

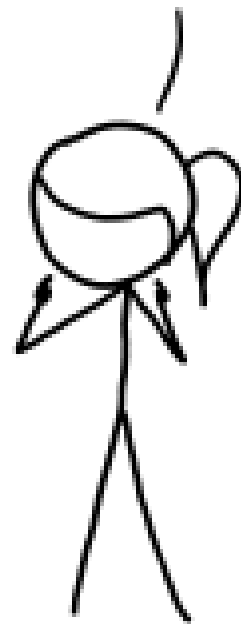
I'M DOING AN ART PROJECT WHERE I TAKE A PICTURE OF MYSELF EVERY HUNDRED YEARS.



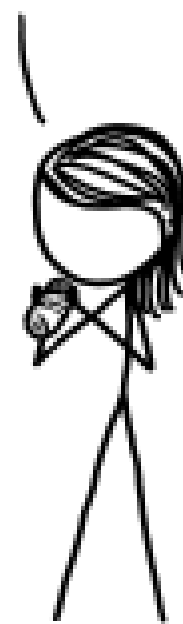
I'M DOING AN ART PROJECT WHERE I TAKE A PICTURE OF MYSELF EVERY $\frac{1}{24}$ TH OF A SECOND.



I'M DOING AN ART PROJECT WHERE YOU CAN COME TO MY HOUSE AND WATCH MY ACTUAL FACE AGE IN REAL TIME.



I'M DOING AN ART PROJECT WHERE YOU ALL DO THOSE THINGS WHILE I EAT A BURRITO.



End of part 2 of Lecture 1B

Make sure you finish the exercise on the rstudio.cloud

Start of part 3 of Lecture 1B

Code: Chunk names

Straight after the ````{r` you can use a text string to name the chunk:

```
```{r read-crime-data}  
crime <- read_csv("data/crime-data.csv")
```
```

Code: Chunk names

Naming code chunks has three advantages:

1. Navigate to specific chunks using the drop-down code navigator in the bottom-left of the script editor.
2. Graphics produced by chunks now have useful names.
3. You can set up networks of cached chunks to avoid re-performing expensive computations on every run.

Code: Chunk names

Every chunk should ideally have a name.

Naming things is hard, but follow these rules and you'll be fine:

1. One word that describes the action (e.g., "read")
2. One word that describes the thing inside the code (e.g., "gapminder")
3. Separate words with "-" (e.g., read-gapminder)

Code: Chunk options

You can control how the code is output by changing the code chunk options which follow the title.


```
```{r read-gapminder, eval = FALSE, echo = TRUE}  
gap <- read_csv("gapminder.csv")
```
```

What do you think this does?

00 : 30


Code: Chunk options

The code chunk options you need to know about right now are:

 `cache`: TRUE / FALSE. Do you want to save the output of the chunk so it doesn't have to run next time?


 `eval`: TRUE / FALSE Do you want to evaluate the code?

 `echo`: TRUE / FALSE Do you want to print the code?

 `include`: TRUE / FALSE Do you want to include code output in the final output document? Setting to FALSE means nothing is put into the output document, but the code is still run.

You can read more about the options at the official documentation: <https://yihui.name/knitr/options/#code-evaluation>

Your turn

 go to `rstudio.cloud`, open document `01-oz-atlas.Rmd` and change the document so that the code output is hidden, but the graphics are shown. (Hint: Google "rstudio rmarkdown cheatsheet" for some tips!)

 Re-Knit the document.

 Take a look at the [R Markdown Gallery](#).

05 : 00

End of Part 3 of Lecture 1B

Start of Part 4 of Lecture 1B


Global options: Set and forget

You can set the default chunk behaviour once at the top of the .Rmd file using a chunk like:

```
knitr::opts_chunk$set(  
  echo = FALSE,  
  cache = TRUE  
)
```

then you will only need to add chunk options when you have the occasional one that you'd like to behave differently.

Your turn

 Go to your `01-oz-atlas.Rmd` document on `rstudio.cloud` and change the global settings at the top of the `rmarkdown` document to `echo = FALSE`, and `cache = TRUE`

```
knitr::opts_chunk$set(  
  echo = FALSE,  
  cache = TRUE  
)
```

05 : 00

End of part 4 of lecture 1b

Start of part 5 of lecture 1b

DEMO

The many different outputs of rmarkdown


Your turn: Different types of documents

1. Change the output of your current R Markdown file to produce a **Word document**. Now try to produce pdf - this may not work! That's OK, we do'nt need it right now.
2. Create a new document that will produce a slide show `File > New R Markdown > Presentation`
3. Create a flexdashboard document - see this option in the `File > New R Markdown > From template list`.

Recap:

- ▮ There is a Reproducibility Crisis
- ▮ rmarkdown = YAML + text + code
- ▮ rmarkdown has many different output types
- ▮ Platypus are interesting!
- ▮ Assignment will be announced next week

Learning more:

 [R Markdown cheat sheet](#) and [Markdown Quick Reference](#) (Help -> Markdown Quick Reference) handy, we'll refer to it often as the course progresses

That's it!

This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).



Lecturer: Nicholas Tierney

Department of Econometrics and Business Statistics

✉ ETC1010.Clayton-x@monash.edu

11th Mar 2020

