# R Techniques for Reproducible Research
# Day 1

Thomas Robinson
thomas.robinson@politics.ox.ac.uk
ts-robinson.com

April 2019

# Objectives of this course

1. Better understand what **reproducibility** means for social scientists using data
2. Understand why effective coding (within **R**) is crucial for reproducibility
3. Cover how to set up workflows and use packages that help us to ensure our research is reproducible...
4. ... and boost our coding proficiency in the process!

# Today's focus

- Discuss reproducibility and understand how R and coding proficiency are integral to it
- Go through basic coding etiquette (a style guide for intuitive and readable code)
- Understand some of R's limitations
- Tomorrow: nitty-gritty walkthroughs of key **tidyverse** packages/functions, and how to create open and accessible coding workflows for your research projects

# Motivation

- "Reproducibility" is an increasing priority across the (social) sciences
- But knowing how to make your research and workflow reproducible is often tricky
- Not least because the meaning of "reproduciblility" differs depending on your discipline
- So we need to establish some baselines:
  1. What is reproducibility (with respect to the social sciences)?
  2. Why should we care?
  3. What has R got to do with reproducibility?

# What is reproducibility in the social sciences?

- Many related terms, with different values to social science research:
    1. Reproducibility - the ability to generate the exact results of the initial analysis again

# What is reproducibility in the social sciences?

- Many related terms, with different values to social science research:
  1. Reproducibility - the ability to generate the exact results of the initial analysis again
  2. Replication - the ability to gather new data that confirms the results of the original study

# What is reproducibility in the social sciences?

- Many related terms, with different values to social science research:
  1. Reproducibility - the ability to generate the exact results of the initial analysis again
  2. Replication - the ability to gather new data that confirms the results of the original study
  3. Robustness - how sturdy your findings are to perturbations of the method/model/sampling etc.

# What is reproducibility in the social sciences?

- Many related terms, with different values to social science research:
  1. Reproducibility - the ability to generate the exact results of the initial analysis again
  2. Replication - the ability to gather new data that confirms the results of the original study
  3. Robustness - how sturdy your findings are to perturbations of the method/model/sampling etc.
  4. Repeatability - whether we can, in fact, gather new and commensurate data to replicate the original study

# What is reproducibility in the social sciences?

- Many related terms, with different values to social science research:
  1. Reproducibility - the ability to generate the exact results of the initial analysis again
  2. Replication - the ability to gather new data that confirms the results of the original study
  3. Robustness - how sturdy your findings are to perturbations of the method/model/sampling etc.
  4. Repeatability - whether we can, in fact, gather new and commensurate data to replicate the original study
- Depending on the specific discipline, project and research design, which of these four concepts are relevant will vary.

# What is reproducibility in the social sciences?

- Many related terms, with different values to social science research:
  1. Reproducibility - the ability to generate the exact results of the initial analysis again
  2. Replication - the ability to gather new data that confirms the results of the original study
  3. Robustness - how sturdy your findings are to perturbations of the method/model/sampling etc.
  4. Repeatability - whether we can, in fact, gather new and commensurate data to replicate the original study
- Depending on the specific discipline, project and research design, which of these four concepts are relevant will vary.

## Methods reproducibility

"The ability to implement, as exactly as possible, the experimental and computational procedures, with the **same data and tools**, to obtain the **same results**" (Goodman, Fanelli, and Ionnaidis 2016).[a]

[a] http://stm.sciencemag.org/content/8/341/341ps12

# More specifically...

We want to be able to write code where:

- We don't need to use Excel at all!

# More specifically...

We want to be able to write code where:

- We don't need to use Excel at all!
- Anyone can generate the same results as what is published

# More specifically...

We want to be able to write code where:

- We don't need to use Excel at all!
- Anyone can generate the same results as what is published
- It is easy to understand the assumptions you make in your workflow

# More specifically...

We want to be able to write code where:

- We don't need to use Excel at all!
- Anyone can generate the same results as what is published
- It is easy to understand the assumptions you make in your workflow

In turn, achieving these aims depends on two things:

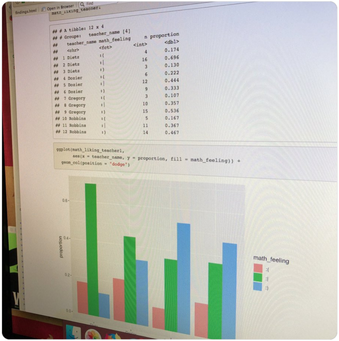
1. The clarity of your code - what in fact happens when we press 'run'?
2. Ensuring integrity of your workflow - do we have the necessary data/code/file structure? Have we ensured appropriate access to these resources?

# Why should we care? (1/2)

- As research becomes increasingly sophisticated, and demands for reproducible workflows increase, we don't want to be left behind:

**Andrew Heiss, PhD**
@andrewheiss
Follow

Teaching my 11-yo #rstats tidyverse for her science fair project (diff-in-diff experiment to see if reading a math-related story prior to a math test can reduce math anxiety)




9:34 PM - 3 Feb 2019 from Spanish Fork, UT

# Why should we care? (2/2)

- Our research should be falsifiable, and that includes our code!

[1] https://www.nature.com/articles/s41562-018-0399-z

[2] https://www.vox.com/science-and-health/2018/8/27/17761466/psychology-replication-crisis-nature-social-science

# Why should we care? (2/2)

- Our research should be falsifiable, and that includes our code!
- Moreover, as the field is starting to realise, lots of social science research fails to *replicate* (Camerer et al 2018)[1]:
  - 13/21 major social science studies failed to replicate significance and direction of effect
  - Replicated effect sizes were on average 50% smaller than originally reported

[1] https://www.nature.com/articles/s41562-018-0399-z

[2] https://www.vox.com/science-and-health/2018/8/27/17761466/psychology-replication-crisis-nature-social-science

# Why should we care? (2/2)

- Our research should be falsifiable, and that includes our code!
- Moreover, as the field is starting to realise, lots of social science research fails to *replicate* (Camerer et al 2018)[1]:
  - 13/21 major social science studies failed to replicate significance and direction of effect
  - Replicated effect sizes were on average 50% smaller than originally reported
- More cynically, from a publication perspective, "journals are... increasingly insisting that **scientists share all the underlying data of their experiments for others to assess**" (Vox 2018)[2]

[1] https://www.nature.com/articles/s41562-018-0399-z

[2] https://www.vox.com/science-and-health/2018/8/27/17761466/psychology-replication-crisis-nature-social-science

# What has R got to do with reproducibility?

R (RStudio) is becoming the dominant statistical language (software) within social science research. It is free, open source and really powerful! BUT:

- Code is often private, esoteric and difficult to read
- Many researchers are picking up coding skills as and when they are needed
- And as a result miss out on many of the ‘best practice’ conventions that underpin coding proficiency

# What has R got to do with reproducibility?

R (RStudio) is becoming the dominant statistical language (software) within social science research. It is free, open source and really powerful! BUT:

- Code is often private, esoteric and difficult to read
- Many researchers are picking up coding skills as and when they are needed
- And as a result miss out on many of the 'best practice' conventions that underpin coding proficiency

Embedding effective R coding within our reproducibility framework helps:

# What has R got to do with reproducibility?

R (RStudio) is becoming the dominant statistical language (software) within social science research. It is free, open source and really powerful! BUT:

- Code is often private, esoteric and difficult to read
- Many researchers are picking up coding skills as and when they are needed
- And as a result miss out on many of the 'best practice' conventions that underpin coding proficiency

Embedding effective R coding within our reproducibility framework helps:

1. Make our code more readable (to ourselves and others)

# What has R got to do with reproducibility?

R (RStudio) is becoming the dominant statistical language (software) within social science research. It is free, open source and really powerful! BUT:

- Code is often private, esoteric and difficult to read
- Many researchers are picking up coding skills as and when they are needed
- And as a result miss out on many of the ‘best practice’ conventions that underpin coding proficiency

Embedding effective R coding within our reproducibility framework helps:

1. Make our code more readable (to ourselves and others)
2. Prevent mistakes

# What has R got to do with reproducibility?

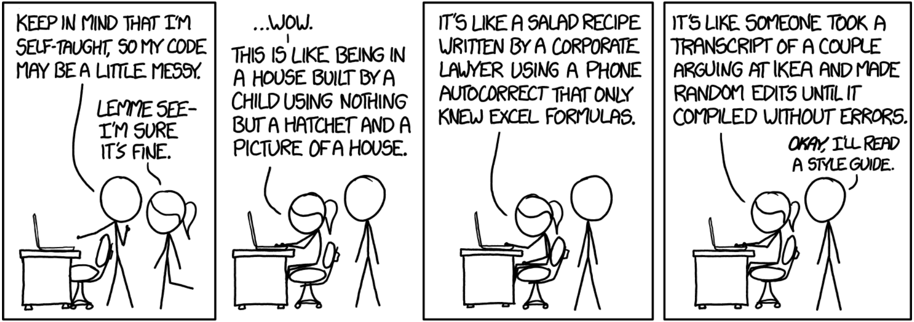
R (RStudio) is becoming the dominant statistical language (software) within social science research. It is free, open source and really powerful! BUT:

- Code is often private, esoteric and difficult to read
- Many researchers are picking up coding skills as and when they are needed
- And as a result miss out on many of the ‘best practice’ conventions that underpin coding proficiency

Embedding effective R coding within our reproducibility framework helps:

1. Make our code more readable (to ourselves and others)
2. Prevent mistakes
3. Ensures efficiency (across machines too)

# An unhelpful reviewer



# What has R got to do with reproducibility?

- *R* is complemented by other software/systems like RStudio, **tidyverse** and Git(Hub) that enable us to move away from bad practices:
    - Manually cleaning data (in Excel)
    - Manually creating LaTeXtables
    - Overwriting key scripts/data files
    - ... etc.

# Datasets for today and tomorrow (1/2)

- Today's focus: stylistic aspects of coding
- Tomorrow: nitty gritty substantive code and workflows
- Useful to have some example data to work with!
- My examples:

[3] https://injepijournal.biomedcentral.com/articles/10.1186/s40621-018-0174-7

# Datasets for today and tomorrow (1/2)

- Today's focus: stylistic aspects of coding
- Tomorrow: nitty gritty substantive code and workflows
- Useful to have some example data to work with!
- My examples:
  - A dataset released over the winter and cited by over 120 news outlets

[3] https://injepijournal.biomedcentral.com/articles/10.1186/s40621-018-0174-7

# Datasets for today and tomorrow (1/2)

- Today's focus: stylistic aspects of coding
- Tomorrow: nitty gritty substantive code and workflows
- Useful to have some example data to work with!
- My examples:
  - A dataset released over the winter and cited by over 120 news outlets
  - One of the most comprehensive datasets on human survival...

[3] https://injepijournal.biomedcentral.com/articles/10.1186/s40621-018-0174-7

# Datasets for today and tomorrow (1/2)

- Today's focus: stylistic aspects of coding
- Tomorrow: nitty gritty substantive code and workflows
- Useful to have some example data to work with!
- My examples:
  - A dataset released over the winter and cited by over 120 news outlets
  - One of the most comprehensive datasets on human survival...
  - ...in all of Westeros (and beyond)!

[3] https://injepijournal.biomedcentral.com/articles/10.1186/s40621-018-0174-7

# Datasets for today and tomorrow (1/2)

- Today's focus: stylistic aspects of coding
- Tomorrow: nitty gritty substantive code and workflows
- Useful to have some example data to work with!
- My examples:
  - A dataset released over the winter and cited by over 120 news outlets
  - One of the most comprehensive datasets on human survival...
  - ...in all of Westeros (and beyond)!
- Game of Thrones mortality dataset (Lystad and Brown 2018), kindly provided by Dr Reidar Lystad at the Macquaire University.[3]

[3] https://injepijournal.biomedcentral.com/articles/10.1186/s40621-018-0174-7

# Datasets for today and tomorrow (1/2)

- Today's focus: stylistic aspects of coding
- Tomorrow: nitty gritty substantive code and workflows
- Useful to have some example data to work with!
- My examples:
  - A dataset released over the winter and cited by over 120 news outlets
  - One of the most comprehensive datasets on human survival...
  - ...in all of Westeros (and beyond)!
- Game of Thrones mortality dataset (Lystad and Brown 2018), kindly provided by Dr Reidar Lystad at the Macquaire University.[3]
- Well-collated individual-level data with multiple grouping variables that allow us to explore how to code in a reproducible way!

[3] https://injepijournal.biomedcentral.com/articles/10.1186/s40621-018-0174-7

SpringerOpen | Injury Epidemiology

Inj Epidemiol. 2018 Dec; 5: 44. PMCID: PMC6286904
Published online 2018 Dec 10. doi: 10.1186/s40621-018-0174-7 PMID: 30535868

Inj Epidemiol

# "Death is certain, the time is not": mortality and survival in *Game of Thrones*

Reidar P. Lystad[1] and Benjamin T. Brown[2]

‣ Author information ‣ Article notes ‣ Copyright and License information Disclaimer

## Associated Data

▾ Data Availability Statement

The dataset is available from the corresponding author.

## Abstract

Go to:

### Background

*Game of Thrones* is a popular television series known for its violent and graphic portrayal of the deaths of its characters. This study aimed to examine the mortality and survival of important characters in *Game of*

# Datasets for today and tomorrow (2/2)

- To allow you to wrangle with some social science data yourselves, you can use:
  1. Your own data!
  2. Pittsburgh arrest data 2015, available at: `http://bit.ly/oxss_pittsburgh`
- The aim is to start writing R-scripts and conducting analyses that have reproducibility at their heart
- And to be critical of your own (existing) code to see how we can improve its readability and efficiency

# What is "good" code?

Fundamentals (in my opinion) of well-written code:

- We're looking for code that is readable, efficient and ubiquitous
- The code itself should be accessible to someone in the field with a reasonable degree of proficiency

We'll focus on three things today:

1. Basic coding conventions
2. Good code commenting
3. Understanding the limitations of base-R

# Basic coding conventions: Code Structure

- Do different things in different sections!
  - Keep your data-cleaning code separate from your models
  - Keep your models separate from your figures and tables
  - Keep your figures and tables separate from your robustness/appendix material!

# Basic coding conventions: Code Structure

- Do different things in different sections!
  - Keep your data-cleaning code separate from your models
  - Keep your models separate from your figures and tables
  - Keep your figures and tables separate from your robustness/appendix material!
- Two options:
  1. Reasonably short project: **one R script**, divide sections using a long line of commenting symbols (hashtags)
  2. Bigger project: **one R script per task** – cleaning, analysis, table/figure production, appendix material

# Basic coding conventions: Code Structure

The source(*filename*) command is your best friend:

```
#### Init ####
...
source("got_functions.R")
data <- data_clean("got_data_final.csv")
#############################################################
#### Data subsetting ####
data_lannister_female <- data %>%
                         filter(grepl("Lannister", name),
                                sex = 2)

data_stark_female <- data %>%
                     filter(grepl("Stark", name),
                            sex = 2)
#############################################################
##### Basic analysis ####
t.test(data_lanister_female$exp_time_sec,
      data_stark_female$exp_time_sec)
```

# How should I start a new R script?

The user should be able to see:

- What the project title is
- A description of the specific R script
- The initialisation code:
  - Packages (remember these are all installed locally!)
  - Code dependencies
  - Any cleaning or data loading
- Avoid hardcoding a working directory!

# Basic coding conventions: Code Structure

You can add your own coding shorthand into RStudio: Preferences - Code - Edit Snippets...

```
snippet header
  ######################################################################
  ##                                                                  ##
  ##      PROJECT TITLE:                                              ##
  ##      PROJECT AUTHOR:   THOMAS S. ROBINSON                        ##
  ##      EMAIL:            THOMAS.ROBINSON@POLITICS.OX.AC.UK         ##
  ##                                                                  ##
  ##      DESCRIPTION:                                                ##
  ##                                                                  ##
  ######################################################################

  #### Packages ####
  library(tidyverse)
  library(here)
```

# Basic coding conventions: Code Structure

- R is essentially whitespace agnostic

# Basic coding conventions: Code Structure

- R is essentially whitespace agnostic
- Use whitespace – linebreaks and indentation – to make the structure of your code clear

# Basic coding conventions: Code Structure

- R is essentially whitespace agnostic
- Use whitespace – linebreaks and indentation – to make the structure of your code clear
- Every new "block" of code – after a conditional statement, in a for loop, within a function etc. – should be indented

# Basic coding conventions: Code Structure

- R is essentially whitespace agnostic
- Use whitespace – linebreaks and indentation – to make the structure of your code clear
- Every new “block” of code – after a conditional statement, in a for loop, within a function etc. – should be indented
- But you can also align parameters within function calls to make them more readable

# Basic coding conventions: Code Structure

- R is essentially whitespace agnostic
- Use whitespace – linebreaks and indentation – to make the structure of your code clear
- Every new "block" of code – after a conditional statement, in a for loop, within a function etc. – should be indented
- But you can also align parameters within function calls to make them more readable
- Code should be **vertically biased** - it's easier to scroll down than across!

# Basic coding conventions: Code Structure

- R is essentially whitespace agnostic
- Use whitespace – linebreaks and indentation – to make the structure of your code clear
- Every new "block" of code – after a conditional statement, in a for loop, within a function etc. – should be indented
- But you can also align parameters within function calls to make them more readable
- Code should be **vertically biased** - it's easier to scroll down than across!
    - For instance, take a look at the code for plotting points using ggplot: `https://github.com/tidyverse/ggplot2/blob/master/R/geom-point.r`

# Basic coding conventions: Variable Names

- Create concise, understandable, and memorable names for your variables!
  - Be consistent

# Basic coding conventions: Variable Names

- Create concise, understandable, and memorable names for your variables!
  - Be consistent
  - Use **words**, not symbols

# Basic coding conventions: Variable Names

- Create concise, understandable, and memorable names for your variables!
  - Be consistent
  - Use **words**, not symbols
  - Keep names short

# Basic coding conventions: Variable Names

- Create concise, understandable, and memorable names for your variables!
  - Be consistent
  - Use **words**, not symbols
  - Keep names short
  - Avoid capital letters, but do use underscores for multi-word variables

# Basic coding conventions: Variable Names

- Create concise, understandable, and memorable names for your variables!
  - Be consistent
  - Use **words**, not symbols
  - Keep names short
  - Avoid capital letters, but do use underscores for multi-word variables
  - Avoid generic names like ‘x1’ or overly simplified names like ‘d_tf’

# Basic coding conventions: Variable Names

- Create concise, understandable, and memorable names for your variables!
  - Be consistent
  - Use **words**, not symbols
  - Keep names short
  - Avoid capital letters, but do use underscores for multi-word variables
  - Avoid generic names like 'x1' or overly simplified names like 'd_tf'
  - As a general rule, *will you remember what the variable name represents in six months time?*

# Basic coding conventions: Variable Names

- Create concise, understandable, and memorable names for your variables!
  - Be consistent
  - Use **words**, not symbols
  - Keep names short
  - Avoid capital letters, but do use underscores for multi-word variables
  - Avoid generic names like 'x1' or overly simplified names like 'd_tf'
  - As a general rule, *will you remember what the variable name represents in six months time?*
- Avoid using '.' in variable and function names

# Basic coding conventions: Variable Names

Aim to create a left-to-right hierarchy of variable names so it is clear how they relate to each other:

```
library(tidyverse)

data <- read_csv("got_data_final.csv")

data_s1 <- data %>%
           filter(exp_season == 1)

data_s1_lannister_female <- data_s1 %>%
                            filter(grepl("Lannister", name),
                                   sex = 2)
```

# Basic coding conventions: Naming iterators

- For loops (or similar) iterate through elements of a vector using a variable:
  **for (ELEMENT in VECTOR)** { **...** }
- Naming the iterator is key for future clarity
- As a general rule, only use $i, j, k$ when you have to, and only for numerical quantities:
  **for (i in 1:10000)** { **...** }
- For any other type of vector you wish to iterate over, use descriptive names:
  **for (house in unique(data$allegiance_last)** { **...** }
  **for (name in data$names)** { **...** }
  *etc.*

# Effective code commenting

# Commenting your code

- Clarity of your actual code is crucial for reproducibility, but commentary is just as important

# Commenting your code

- Clarity of your actual code is crucial for reproducibility, but commentary is just as important
- Using #'s in your script does two things:

# Commenting your code

- Clarity of your actual code is crucial for reproducibility, but commentary is just as important
- Using #'s in your script does two things:
  - Adds structure to your code

# Commenting your code

- Clarity of your actual code is crucial for reproducibility, but commentary is just as important
- Using #'s in your script does two things:
  - Adds structure to your code
  - Explains inherently unclear elements

# Commenting your code

- Clarity of your actual code is crucial for reproducibility, but commentary is just as important
- Using #'s in your script does two things:
  - Adds structure to your code
  - Explains inherently unclear elements
- Especially useful when research techniques are more complicated, functions have multiple parameters, or you are using unusual packages

# Commenting your code

- Clarity of your actual code is crucial for reproducibility, but commentary is just as important
- Using #'s in your script does two things:
  - Adds structure to your code
  - Explains inherently unclear elements
- Especially useful when research techniques are more complicated, functions have multiple parameters, or you are using unusual packages
- **Trade-off:** More detailed notes can impair the readability of your code

# Effective code commenting: structure

- Label sections of your code.
- In shorter projects, break up distinct sections using hashtag lines
- Use different numbers of hashtags to create a structure to your commentary:

```
#### This is a section ####

## This is a subsection

# This is a comment on a specific line of code
```

# Effective code commenting: when to comment

- Code first, comment later (but don't leave it too long!)
- Isolate the things you'll likely forget
- Don't make the reader infer your intentions
- Describe *why* you are doing things not *what* you are doing
- When writing functions, describe the inputs and the outputs at the beginning in a comment block
- Use verbs in your function names!

# Basic coding conventions: Style Guides

- These are my own suggestions, but different companies/institutions have their own style guides
- Google: `https://google.github.io/styleguide/Rguide.xml`
- tidyverse: `https://style.tidyverse.org/`
- From a reproducibility perspective, aim for clarity and consistency!
- And finally, remember that all code is code - it'll never look like a Monet!

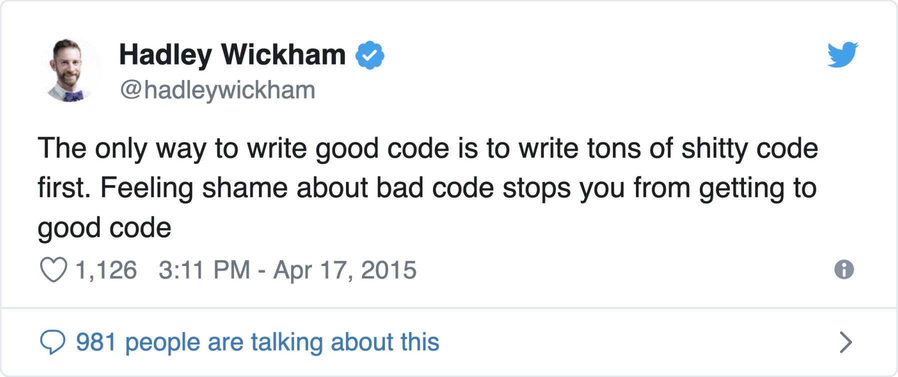
[**If you want to cheat**, use styler: install.packages("styler")]

# “Good code” recap

We’re looking for:

- Well-separated code (within and across scripts)
- Vertically-biased code...
- ... with a logical flow
- Variable names that are short, relevant and memorable!
- Comments that explain why, not what

# Coding with the right attitude



# Advantages of base R

- Base R is an object-orientated programming language
- Often faster than alternatives like MatLab (and indeed Python), with a statistics-orientated syntax
- Has an excellent complementary IDE in RStudio
- Vectorisation is built-in: `http://www.noamross.net/blog/2014/4/16/vectorization-in-r--why.html`
- And it is modular (through the use of packages)[4]

[4] Including **tidyverse** of course, but also machine-learning packages, more-complicated multinomial model functions, and even tools to convert the console into a web-browser `https://florianschaffner.com/websearchr/index.html`

# Limitations of any coding language

We'll focus on R-specific examples tomorrow, but there are some language-agnostic aspects of coding that we should be aware of:

# Limitations of any coding language

We'll focus on R-specific examples tomorrow, but there are some language-agnostic aspects of coding that we should be aware of:

1. Looping is "slow"
    - For-loops can be useful for iterating through data, particularly vectors and lists
    - But too many nested loops can become both slow and untidy
    - Each higher-level iterator has to wait for every iterator below it to finish (without parallelisation)
    - Lots of iterative processes can be solved with more efficient algorithms!

# Limitations of any coding language

We'll focus on R-specific examples tomorrow, but there are some language-agnostic aspects of coding that we should be aware of:

1. Looping is "slow"
   - For-loops can be useful for iterating through data, particularly vectors and lists
   - But too many nested loops can become both slow and untidy
   - Each higher-level iterator has to wait for every iterator below it to finish (without parallelisation)
   - Lots of iterative processes can be solved with more efficient algorithms!
2. Vectorisation is typically much more efficient (and base-R is very good at handling it)
   - Use *ifelse(...)* rather than *if (x)* {...}
   - Avoid trying to do cell-specific operations - ask yourself whether this can be applied to a row/column?

# Limitations of base R

R also has some specific downsides...

- Base R syntax can be a little untidy, hindering the readability of code (we'll cover this more tomorrow)
- Some of the base functions are a little quirky and can upset your results (beware: stringsAsFactors = TRUE)
- Cleaning data with code can seem like a real pain!

# Questions from the session

1. How do I make sure my relative files work across Mac and Windows systems (and thus make my code even more reproducible)?
   - Use “file.path(folder1, folder2)
   - Works like “paste” for strings, but changes dependent on whether the user system is Windows or Unix-based (Mac/Linux)
2. How can I check if two objects (incl. dataframes) are the same?
   - Use “all.equal(OBJECT1, OBJECT2)”
   - This will not only return TRUE if they are equal, but will also tell how they differ (if they do!)
   - Check the documentation for some caveats!

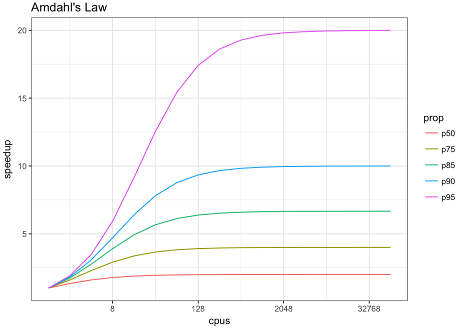
# Parallelization 1/3

- Vectorisation is one way to ensure the efficiency of your code (so too is mapping)
- Sometimes, however, we really do want to use for-loops etc.
- The fundamental issue with basic for-loops is that they run *sequentially*
- Parallelization is a way to get these for-loops to run in parallel
- It leverages the multi-core architecture of modern computers!
- **Key caveat:** You should only parallelize a for-loop which is entirely independent of any iteration of that loop
- Great for bootstrapping where every iteration is an independent draw etc.

# Amdahl's Law

The benefits of parallelization are dependent on:

- How intensive your computation is - does the code-time – run-time balance tip in favour of more efficient computation?
- Amdahl's law: the "speedup" of parallelization is dependent on the proportion of the computation that can be parallelized[5]




[5] Image credit: `https://nceas.github.io/oss-lessons/parallel-computing-in-r/parallel-computing-in-r.html`

# Parallelization 2/3

- Not enough time to go through this in any detail
- Jesus Fernandez-Villaverde (UPenn) has a great guide (for R and also other cool languages like Julia!): `https://www.sas.upenn.edu/~jesusfv/Guide_Parallel.pdf`
- Also, his GitHub has some good walkthroughs/code for comparing coding languages for social sciences: `https://github.com/jesusfv`
- The code in the following page is (lightly) adapted from this guide by Matt Jones: `https://nceas.github.io/oss-lessons/parallel-computing-in-r/parallel-computing-in-r.html`

# Parallelization 3/3

The basics to running parallelized for-loops in R:

- You need to install and load both **foreach** and **doparallel** packages
- *foreach* returns a list by default, so we need to set the ".combine =" parameter to a function that returns a vector (or you could use rbind for more complex output).

```
# Set the number of cores
registerDoParallel(numCores)

# Note: foreach, %dopar%, .combine
foreach (i=1:3, .combine=c) %dopar% {
  sqrt(i)
}

# End the parallelisation
stopImplicitCluster()
```

# And if you want to get really geeky...

- You can also parallelize over a GPU (graphics processing unit)
- GPUs have hundreds (if not thousands) of cores
- Much less powerful per-core, but great for doing large numbers of very small calculations
- This type of parallelization probably goes beyond R, but you can implement it using platforms like CUDA
- Google's neural networks and AI applications are now using TPUs (tensor processing units), designed for high-volume, low-precision calculations
- `https://www.tensorflow.org/`