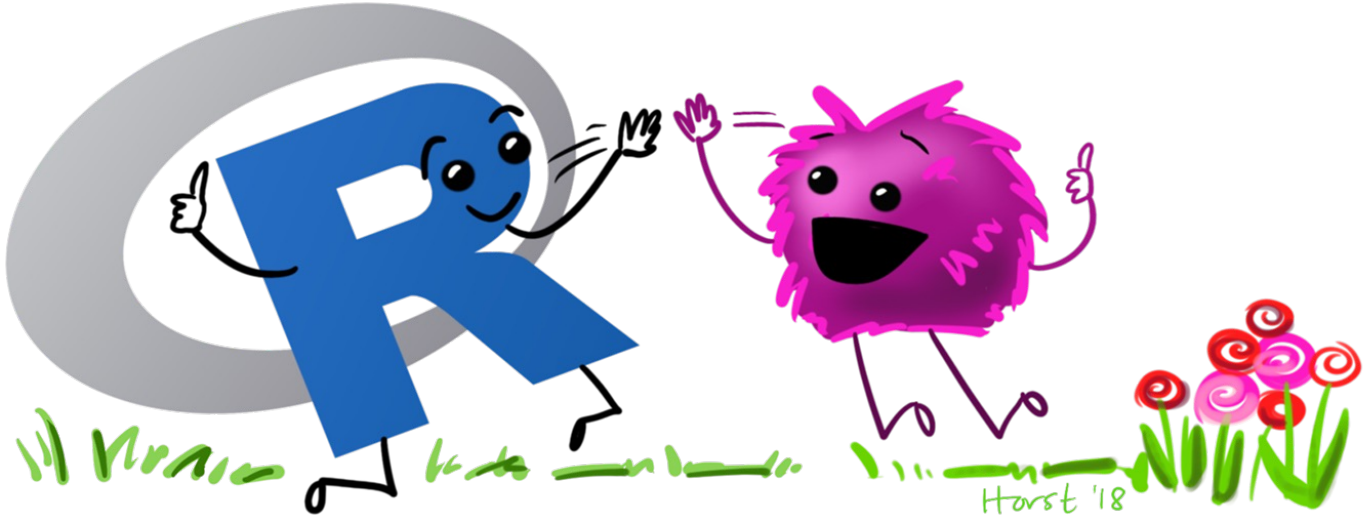
# Convert your code for collaboration




Dr. Fonti Kar

Illustrations by Allison Horst

**Workshop description:**

Want to share your code alongside a publication or with your colleagues? How do you best format your scripts so others can use it? Come along to learn about the best practices for translating your curated scripts into more usable functions for collaborations. We will talk about about functions, abstraction, modularisation, documenting your code using rendered reports to aid collaborations and reproducibility. This is a hands-on workshop, bring along your R script and computer.

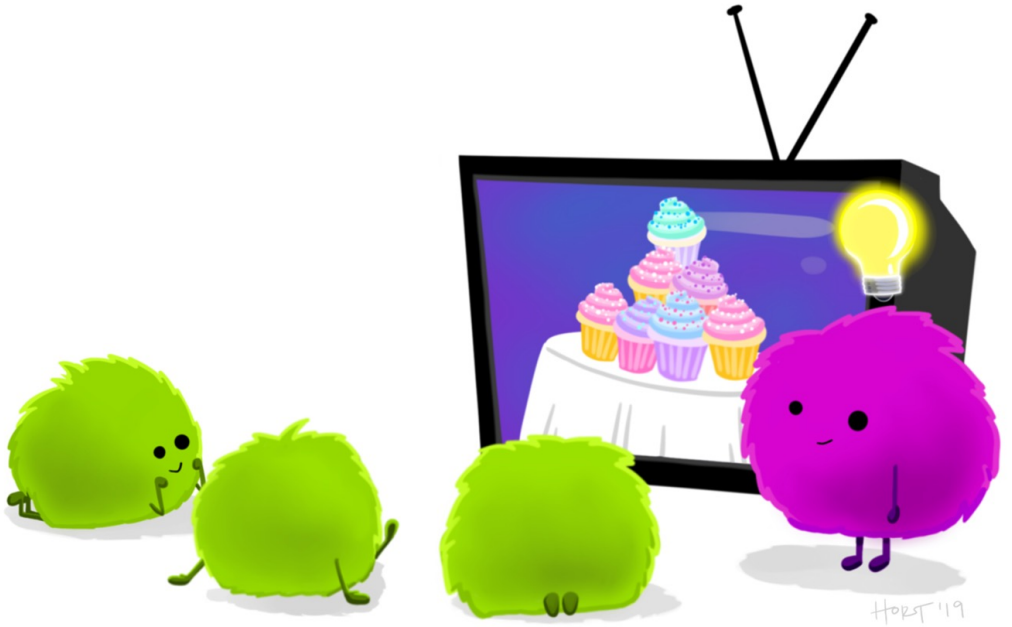
**Prerequisites:**

- You have written R functions before
- Install R/RStudio
- Install Rmarkdown
  `install.packages("rmarkdown")`

# Overview

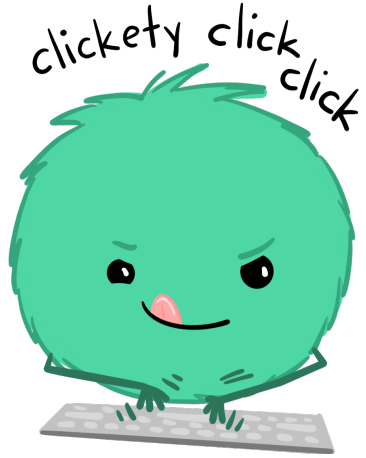
- Functions
  - Why write them?
  - Some tips
- Abstraction
- Modularisation
- Documenting for future us
  - Roxygen
  - Rmarkdown

Hands on edits on your own script!



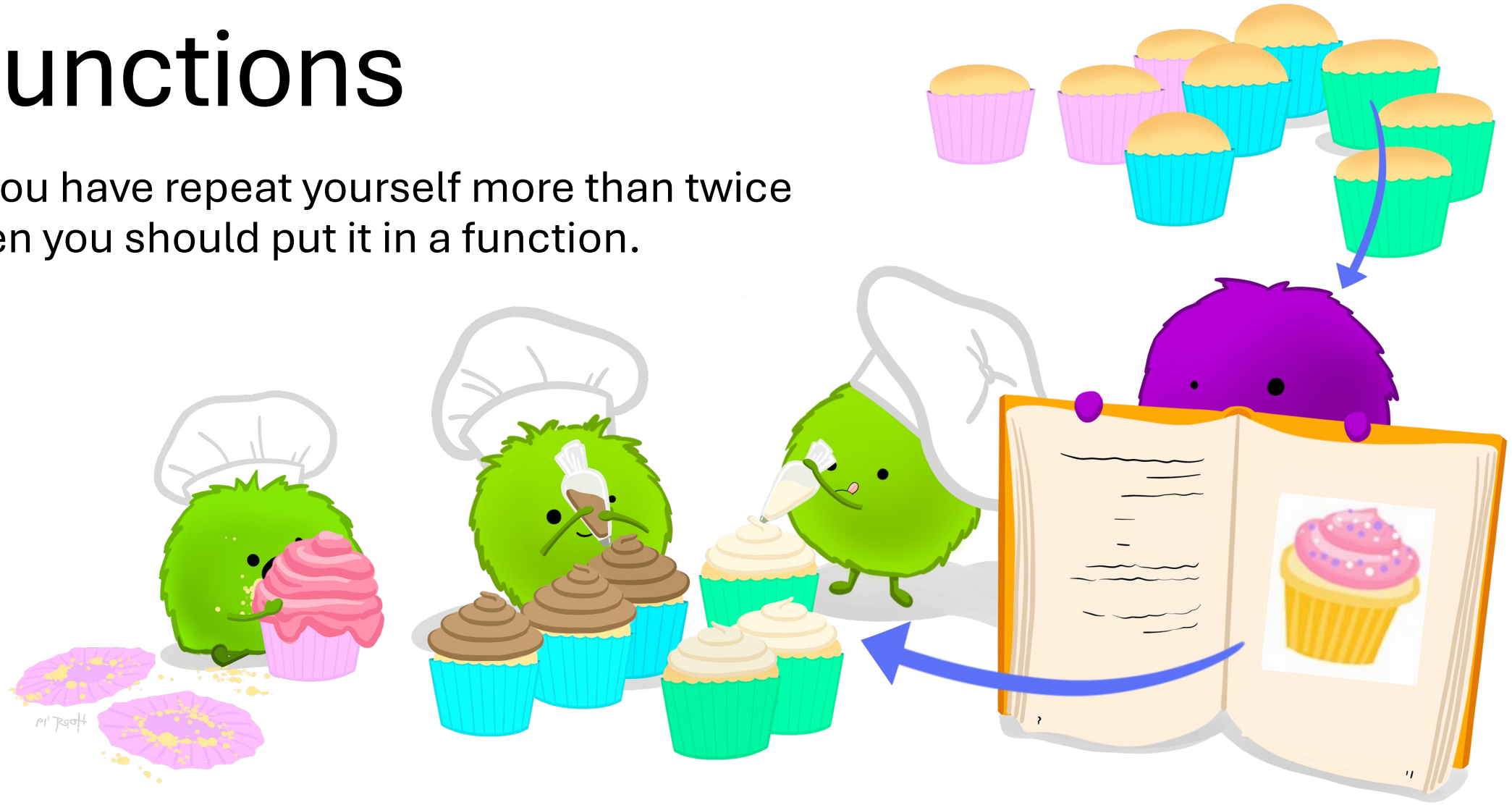
# Why write functions

- A quick group discussion 3 mins



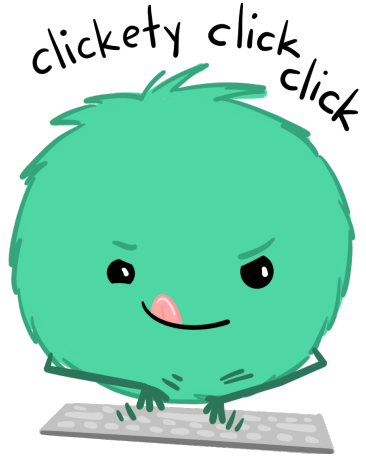
# Functions

If you have repeat yourself more than twice then you should put it in a function.



# Why write functions

1. Improve coding skills
2. More readable code
   - Contain the messiness in an organised manner
3. Tidy R workspace
4. You only need to update code in one place
   - No more search and replace!
   - Reduce incidental mistakes when you copy and paste
     - i.e. updating a variable name in one place, but not in another.
5. Reuse work from project-to-project ♻️



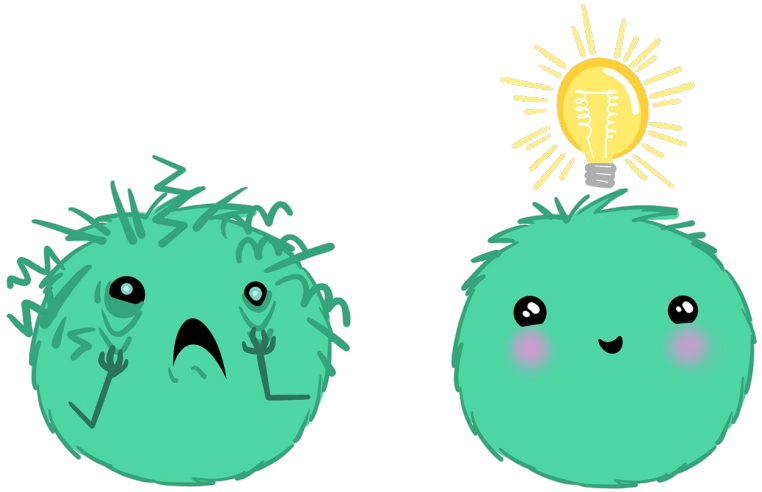
# Function writing tips

- A good **declarative** name
- Input names + defaults are **intuitive**
- **Comments** within function
- Dependencies are **explicit** using ::

```
get_latest_download <- function(path_to_galah_data = 'data/galah/'){
  # Grab all files in folder
  all_downloads <- list.files(path_to_galah_data)

  # Grab the dates only
  dates <- stringr::str_extract(all_downloads,
                                pattern = regex("[0-9]{4}\\-[0-9]{2}\\-[0-9]{2}"))

  # Determine the most recent date
  most_recent <- max(lubridate::ymd(dates)) |> as.character()

  # Identify the file name that is the most recent
  most_recent_file <- stringr::str_subset(all_downloads, most_recent)

  # Get path to most recent file
  paste0(path_to_galah_data, most_recent_file)
}
```

# Abstraction

- Identifying patterns, giving them a name to simplify the wider problem
- E.g.
  - Draw a square
  - Draw a rectangle with sides of equal length
  - Draw a triangle
  - Draw a house
- **Feature or task focused**
  - NOT details about your project, or data or variables
- Creates more readable + reusable code



Read more: https://www.reddit.com/r/learnprogramming/comments/2ite64/explain_like_im_34_years_old_abstraction/

# Abstraction

I want to add 2 to my values!

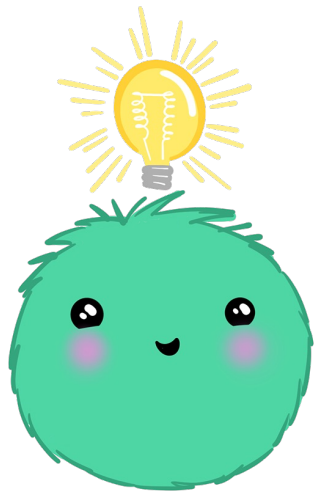
```
> add_2 <- function(y) {
+     2 + y
+ }
> add_2(1:10)
 [1]  3  4  5  6  7  8  9 10 11 12
```

Actually…I need to also add 7 to some other values

```
> add_7 <- function(y) {
+     7 + y
+ }
> add_7(1:10)
 [1]  8  9 10 11 12 13 14 15 16 17
```

Write a function for adding things!

```
> add_x <- function(y, x) {
+     y + x
+ }
> add_x(1:10, 7)
 [1]  8  9 10 11 12 13 14 15 16 17
> add_x(1:10, 30)
 [1] 31 32 33 34 35 36 37 38 39 40
```

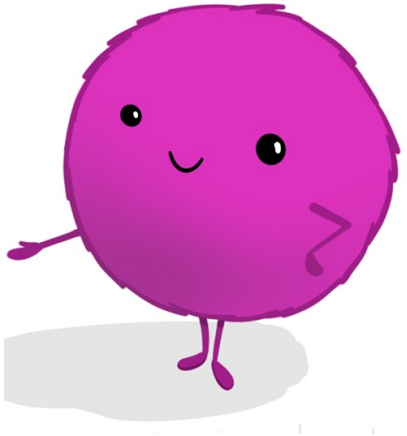


# Tips on being more abstract

- Keep it high level
- Zoom out, **generalise**
- Focus on the tasks/verbs
  - Creating a variable
  - Excluding values
  - Summarise by X
- Explain to someone what your function is doing without any context
  - Nothing about your question/methods/data

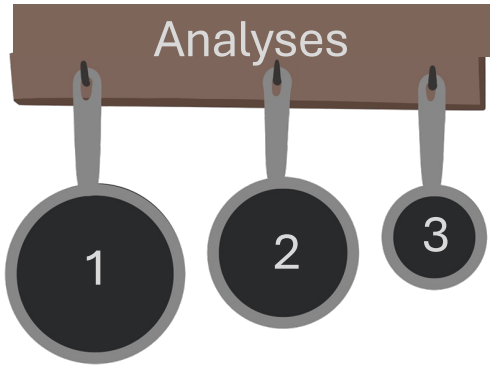
## Try it yourself!

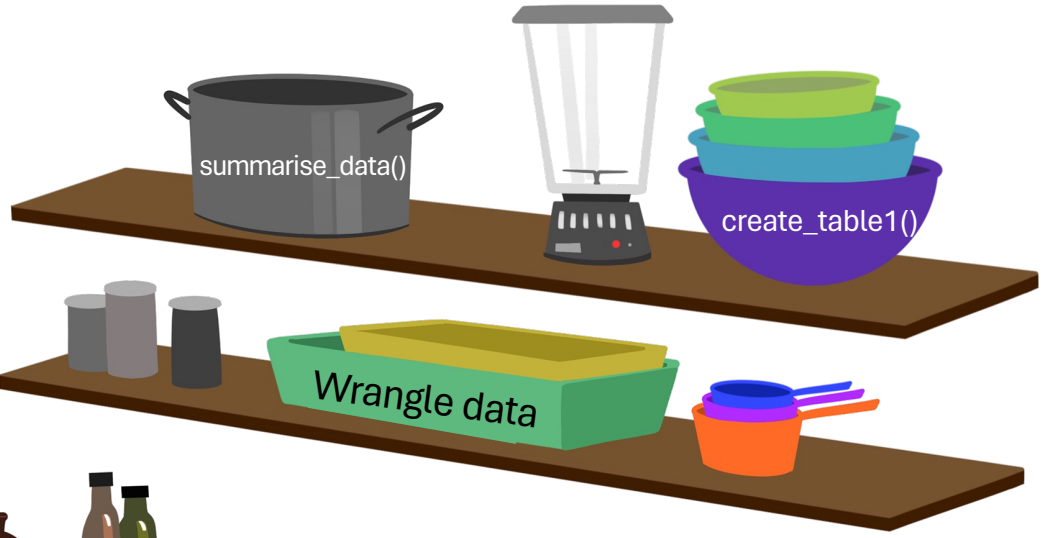
Look at your script, what are some patterns you can identify?
Can you pull these out and generalise it with a function?



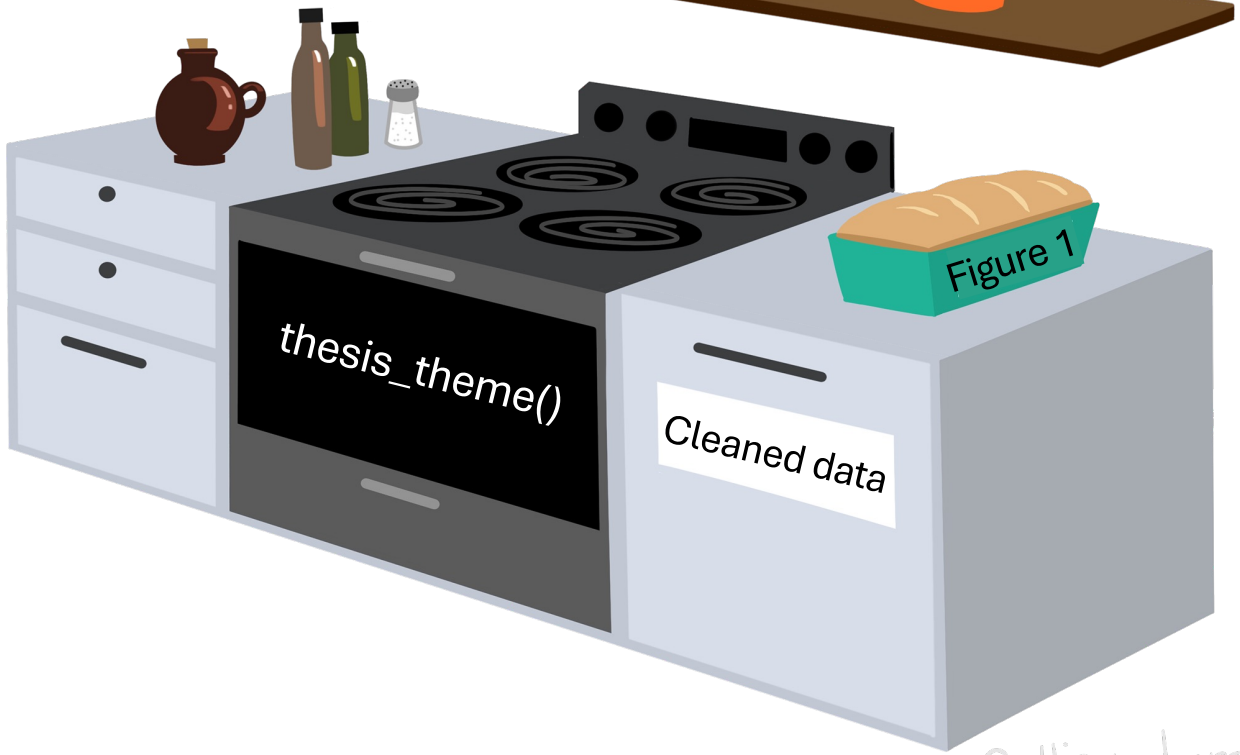
# Break up complexity

# Modularisation



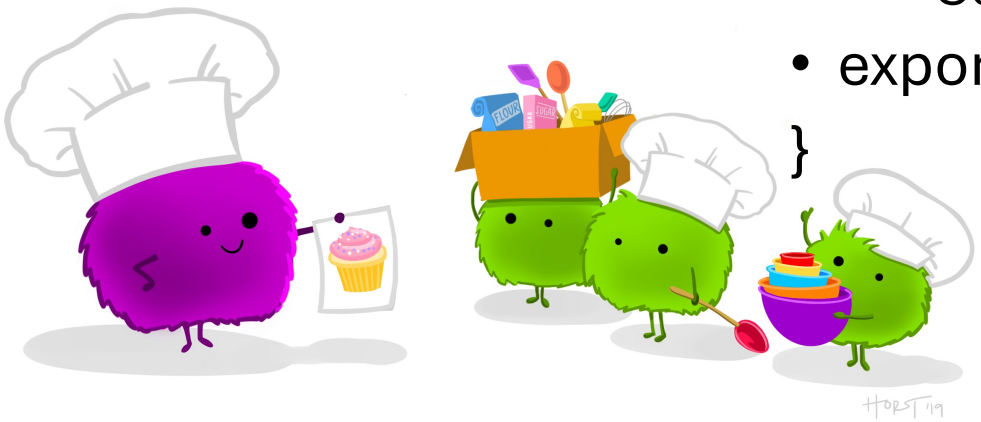





@allison_horst

# Isolate the main ‘steps’

- Make cupcake
  - Gather ingredients
  - Gather equipment
  - Measure our ingredients
  - Mix dry ingredients
  - Mix wet ingredients
  - Put into cake pan
  - Bake
  - Decorate

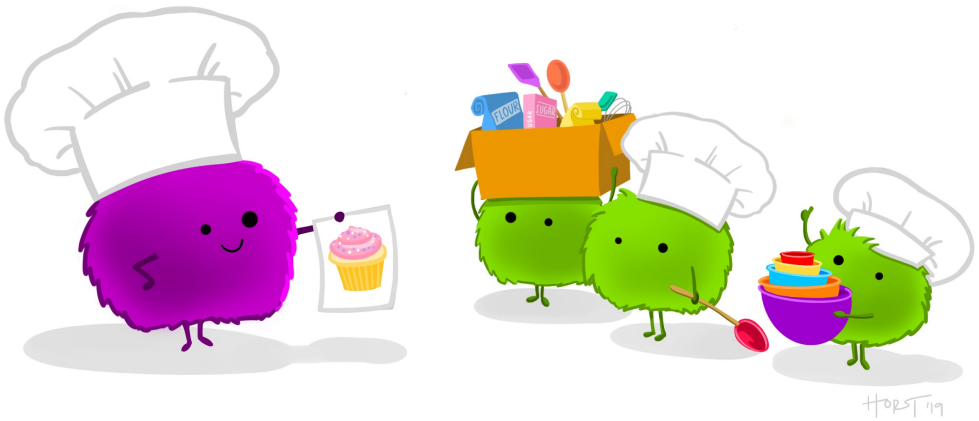


- create_fig1 {
  - clean_data()
  - fig1_wrangle_data()
  - my_phd_theme()
  - fig_base_plot()
    - Legends
    - Axis title
    - Colour of grouping variable
  - export_figure()

  }

# Try it yourself!

- Pull out the main tasks in your script
- Can you put these into separate functions
- Can you put these sub functions into a wrapper?



# Documenting for future us

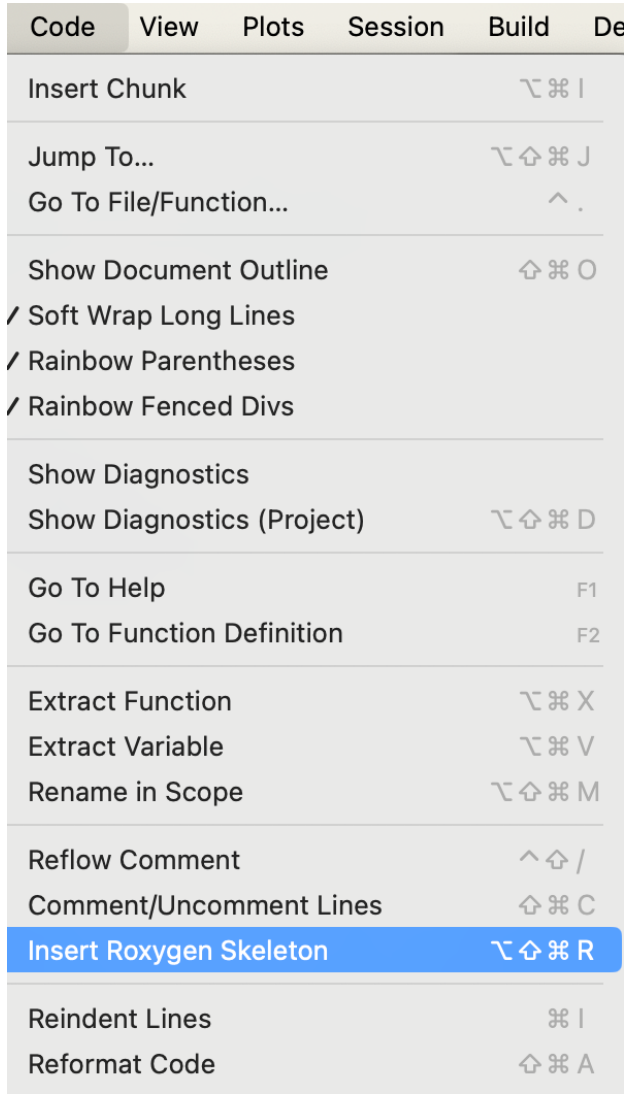
You will thank yourself later

# roxygen2 templates


```r
#' Get the latest galah download
#'
#' @param path_to_galah_data path where parquets of ALA downloads are stored
#'
#' @return path to most recent galah download
#'
#' @examples
#' arrow::open_dataset(get_latest_download())

get_latest_download <- function(path_to_galah_data = 'data/galah/'){
  # Grab all files in folder
  all_downloads <- list.files(path_to_galah_data)

  # Grab the dates only
  dates <- stringr::str_extract(all_downloads,
                                pattern = regex("[0-9]{4}\\-[0-9]{2}\\-[0-9]{2}"))

  # Determine the most recent date
  most_recent <- max(lubridate::ymd(dates)) |> as.character()

  # Identify the file name that is the most recent
  most_recent_file <- stringr::str_subset(all_downloads, most_recent)

  # Get path to most recent file
  paste0(path_to_galah_data, most_recent_file)
}
```

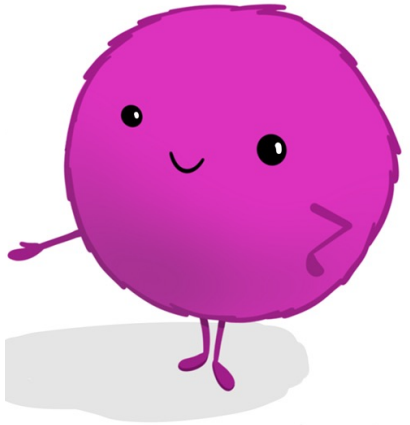
- Meta-data for your functions
- Breaks up long .R files
  - More keywords to Ctrl/CMD-F
- Quick transition to an R package
- Guides user to what they need to know about your function

# Try it yourself!



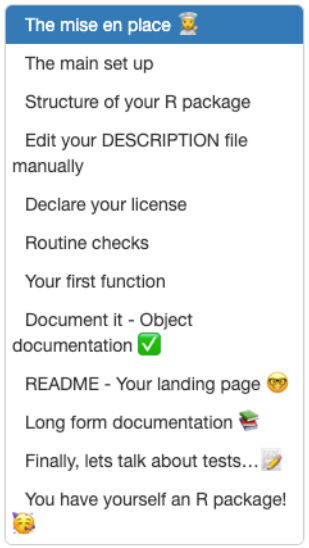

- Document 1 of your own functions using roxygen

```r
#' Get the latest galah download
#'
#' @param path_to_galah_data path where parquets of ALA downloads are stored
#'
#' @return path to most recent galah download
#'
#' @examples
#' arrow::open_dataset(get_latest_download())

get_latest_download <- function(path_to_galah_data = 'data/galah/'){
  # Grab all files in folder
  all_downloads <- list.files(path_to_galah_data)

  # Grab the dates only
  dates <- stringr::str_extract(all_downloads,
                                pattern = regex("[0-9]{4}\\-[0-9]{2}\\-[0-9]{2}"))

  # Determine the most recent date
  most_recent <- max(lubridate::ymd(dates)) |> as.character()

  # Identify the file name that is the most recent
  most_recent_file <- stringr::str_subset(all_downloads, most_recent)

  # Get path to most recent file
  paste0(path_to_galah_data, most_recent_file)
}
```



# Rendered magic!

- Traditional R scripts are hard to read
- Rmarkdown allows you to interweave **text, code, outputs, figures** into a static report
- E.g. https://fontikar.github.io/DIY_Rpkg/

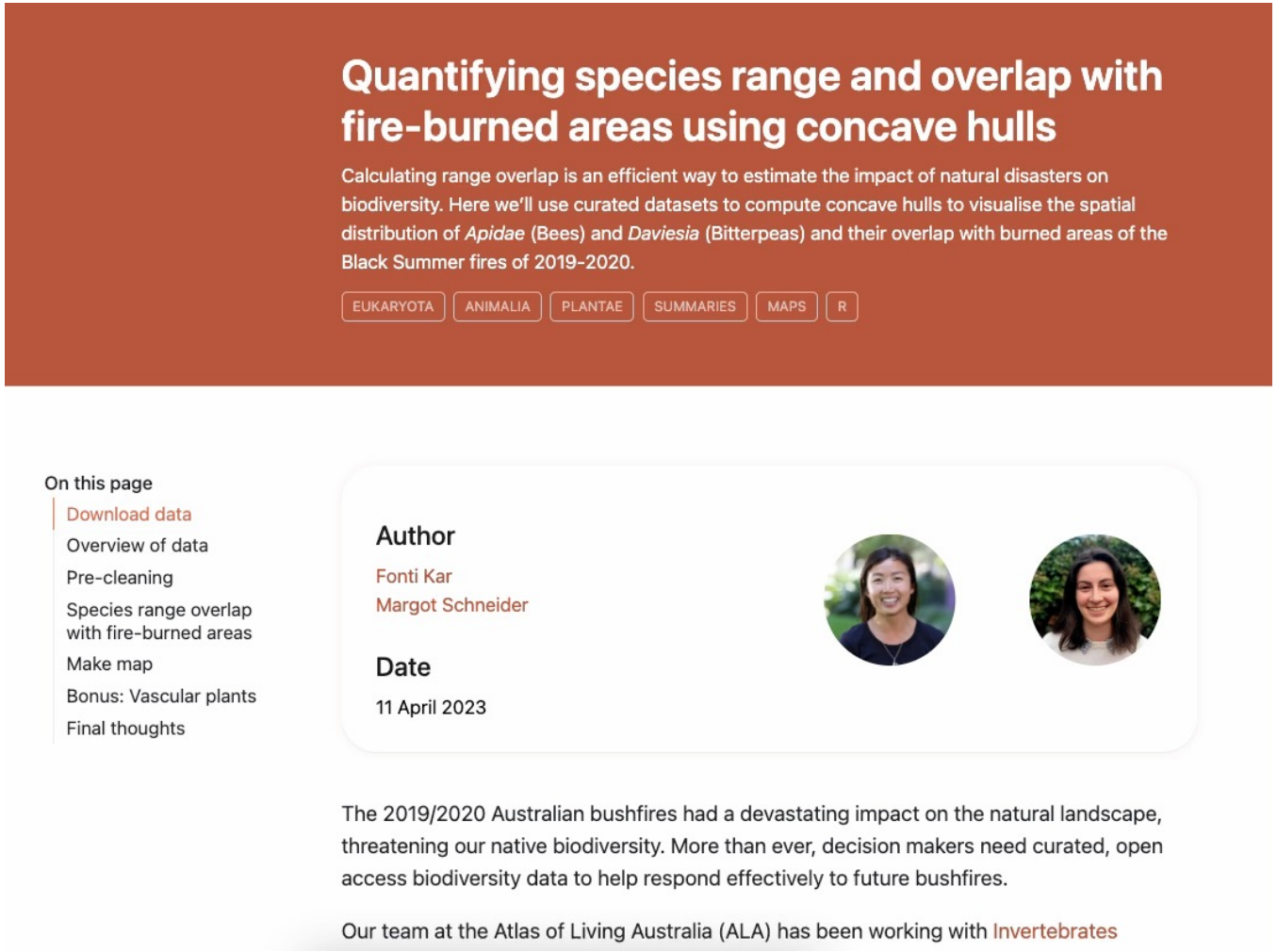





Read more: https://communities.springernature.com/posts/improving-code-reproducibility-small-steps-with-big-impacts

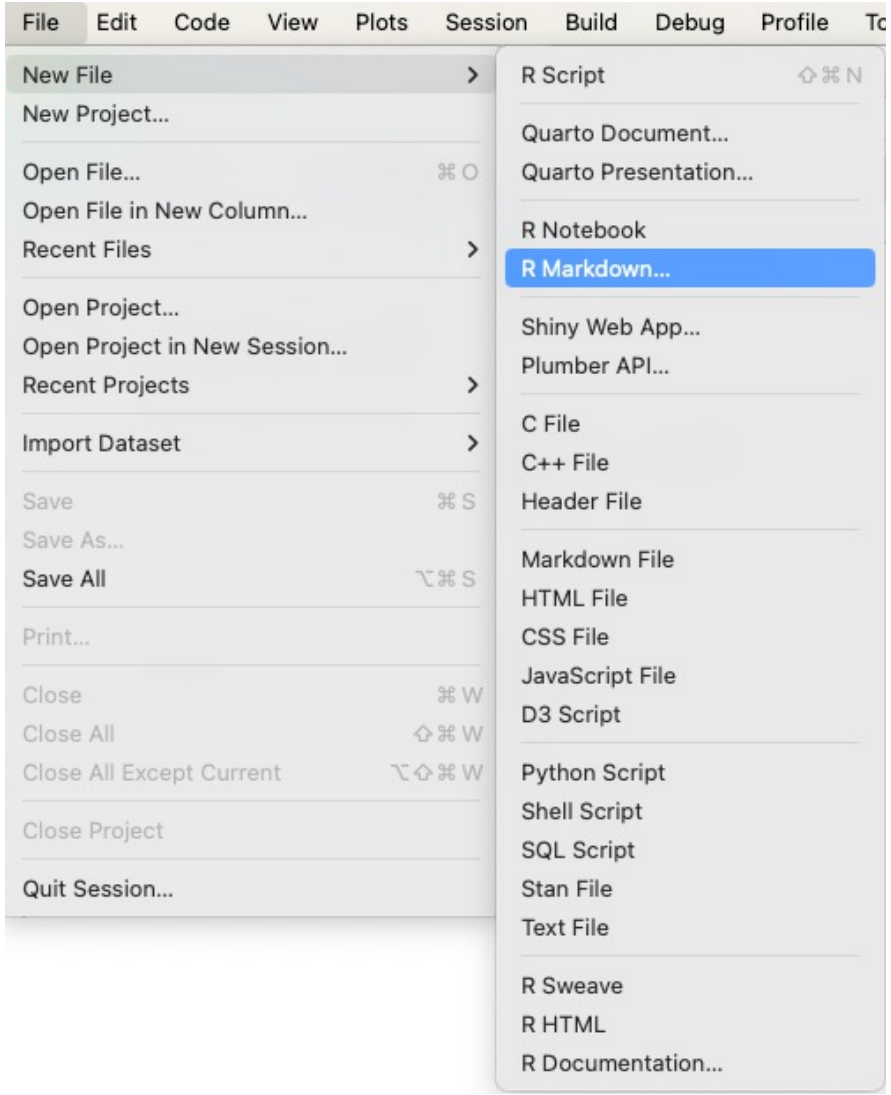
# Guide your users

- Do the heavy lifting for them
- Goal = aid reproducibility
- Explain:
  - What your function does
  - What input function requires
  - **Print out the input**
  - Execute your function
  - The output
  - **Print out the output**
- E.g. https://labs.ala.org.au/

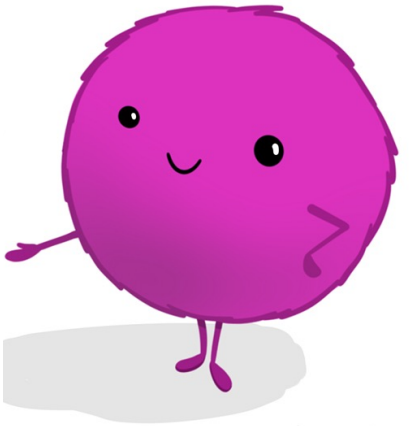



Read more: https://communities.springernature.com/posts/improving-code-reproducibility-small-steps-with-big-impacts

# Try it yourself!



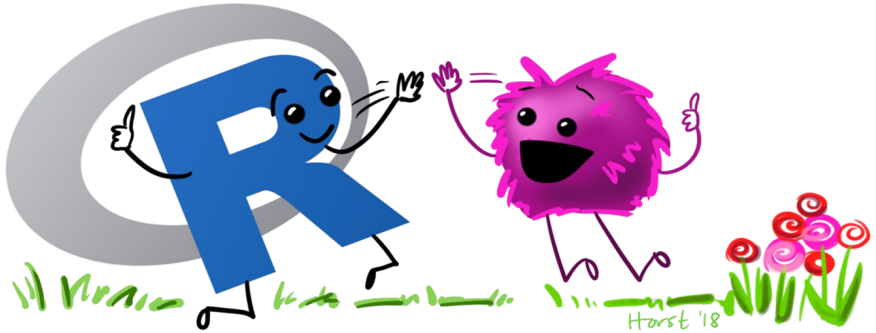

- Transform your script into a .Rmd
- Break up key parts of your script using headers, text, chunks
- Try knitting it



- Not an R user? Try a Quarto document

Learn more: https://quarto.org/docs/get-started/hello/rstudio.html

# Learn more




- On writing functions:
  - https://r4ds.hadley.nz/functions.html
- Functional programming/modularisation:
  - https://adv-r.hadley.nz/fp.html
- On communicating findings using Quarto
  - https://r4ds.hadley.nz/communicate
- On documenting code outputs for sustainability
  - https://youtu.be/OVM5Ok7W1NQ?si=cf6Zzl5jQIKYhCkg