Reproducibility in the social sciences

Rationale, tools & best practice

Thomas de Graaff

June 24, 2021

Department of Spatial Economics, Vrije Universiteit Amsterdam

Replication crisis (?)—roadmap

Open access, freely available online

Essay

Why Most Published Research Findings Are False

John P. A. Ioannidis

Summary

There is increasing concern that most current published research findings are false. The probability that a research claim is true may depend on study power and bias, the number of other studies on the same question, and, importantly, the ratio of true to no relationships among the relationships probed in each scientific field. In this framework, a research finding is less likely to be true when the studies conducted in a field are smaller; when effect sizes are smaller; when there is a greater number and lesser preselection of tested relationships; where there is greater flexibility in designs, definitions, outcomes, and analytical modes; when there is greater financial and other interest and prejudice; and when more teams are involved in a scientific field in chase of statistical significance. Simulations show that for most study designs and settings, it is more likely for a research claim to be false than true. Moreover, for many current scientific fields, claimed research findings may often be simply accurate measures of the prevailing bias. In this essay, I discuss the implications of these problems for the conduct and interpretation of research.

factors that influence this problem and some corollaries thereof.

Modeling the Framework for False Positive Findings

Several methodologists have pointed out [9–11] that the high rate of nonreplication (lack of confirmation) of research discoveries is a consequence of the convenient, yet ill-founded strategy of claiming conclusive research findings solely on the basis of a single study assessed by formal statistical significance, typically for a *p*-value less than 0.05. Research is not most appropriately represented and summarized by *p*-values, but, unfortunately, there is a widespread notion that medical research articles

It can be proven that most claimed research findings are false.

should be interpreted based only on *p*-values. Research findings are defined here as any relationship reaching formal statistical significance, e.g., effective interventions, informative predictors, risk factors, or associations. "Negative" research is also very useful.

is characteristic of the field and can vary a lot depending on whether the field targets highly likely relationships or searches for only one or a few true relationships among thousands and millions of hypotheses that may be postulated. Let us also consider, for computational simplicity, circumscribed fields where either there is only one true relationship (among many that can be hypothesized) or the power is similar to find any of the several existing true relationships. The pre-study probability of a relationship being true is $R/(R+1)$. The probability of a study finding a true relationship reflects the power $1-\beta$ (one minus the Type II error rate). The probability of claiming a relationship when none truly exists reflects the Type I error rate, α. Assuming that c relationships are being probed in the field, the expected values of the 2×2 table are given in Table 1. After a research finding has been claimed based on achieving formal statistical significance, the post-study probability that it is true is the positive predictive value, PPV. The PPV is also the complementary probability of what Wacholder et al. have called the false positive report probability [10]. According to the 2

- Why bother?
- Tools for organisation?
- How to document?
- Script everything. No, really!
- Versioning

Reproducibility in the social sciences

- In the social sciences few attention to what workflow to use (and why)
- more emphasis on transparency
    - scripts, data & additional analyses should be openly shared
- Increasing use of (large) datasets in the social sciences
    - more positivist & deductionist approach
    - more tools readily available
- Related work Healy (2011), Gandrud (2013), and Arribas-Bel and de Graaff (2015)

Why bother? Keep your sanity

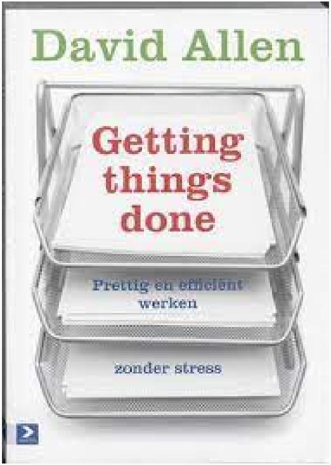
Because projects involve:

- whilst supervisor/referee not satisfied
    - whilst you are not satisfied
        1. formulate research topic;
        2. read literature;
        3. organise ideas;
        4. collect data;
        5. transform data;
        6. analyse data;
        7. present results.

- Circular? No, see this wonderful time-lapse video

Why bother part II? Do not loose your thoughts!

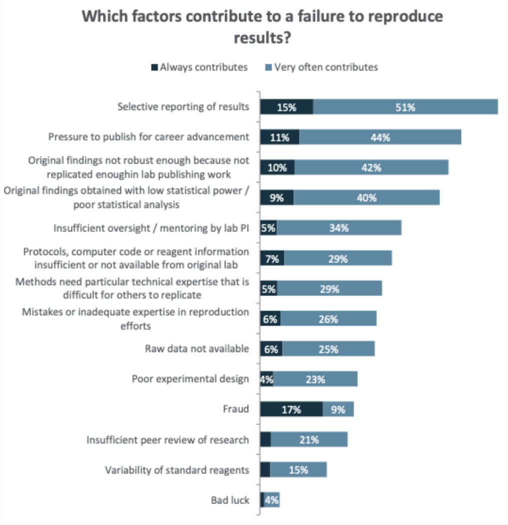
Notes aren't a record of my thinking process. They are my thinking process—Richard Feynman



More efficiency & creativity

- "Never Have The Same Thought Twice. Unless You Like That Thought" (Allen, 2001)
- connect literature with notes/ideas (Ahrens, 2017)

Why bother? The greater good (Nature, 2017)



Practical tips for reproducible research

As discussed by Gandrud (2013)

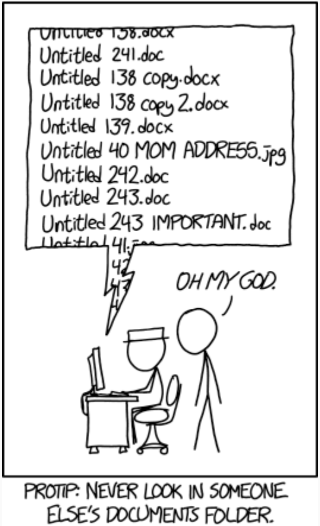
- document everything
- everything is a text file & humanly readable
  - yes: `txt`, `csv`, `R`, `html`, `md`, `tex`
  - no: `Rdata`, `docx`, `xlsx`, `dta`, `ppt`
- explicitly tie your files together
- have a plan to organise, store and makes your files available

My two-cents:

- use open-source as much as possible and make your work easily accessible

Organisation, R and RStudio

Organise your stuff!




```
/
└─project_X
   ├─data
   │  ├─source
   │  └─derived
   ├─docs
   │  ├─notes
   │  └─paper
   ├─literature
   └─code
      └─r
         ├─data
         └─figures
```

Use RStudio projects

- RStudio allows for projects: keep all the files associated with a project together—input data, R scripts, analytical results, figures
- Self contained package
    - searching within a project
    - use relative file names `"./code/read_data.R"`
        - even better in combination with package `here`
    - `renv` keeps track of versions of packages
        - but, simple sessionInfo()
    - use version control of package contents

Multiple code files by functionality

1. For medium-sized projects separate code in multiple R-files by functionality

```
# r code contents of file "main.R"

library("tidyverse")
source(read_data.R)
source(transform_data.R)
source(analyse_data.R)
source(process_output.R)
```

2. [Advanced] For large (complex) projects consider writing a package!

Documentation

Basic project documentation: README files

Write always a README file with information on:

- what/in what order to run code files
- what tools you need (e.g., LaTeX, `Make`, etc)
- README files are text files—often with some structure (markup language). Examples are Markdown and R Markdown.

```
/
└── project_X
    ├── README.md
    ├── project_X.Rproj
    ├── data
    │   ├── source
    │   └── derived
    └── ...
```

Code documentation: coding style

To understand and work with somebody else's code a good coding style is essential: `https://style.tidyverse.org/`

- Names of files/functions: use a verb (what they do) with (consistently) snake_case or camelCase
- Consider `dplyr`-package en pipes for working with dataframes

```
dat_inc <- dat %>%
        mutate(log_income) = log(income)) %>%
        group_by(neighborhood) %>%
        summarize(mean_inc = mean(log_income))
```

Creating documentation: use hashtags

- If you write accompanying documentation text, always express why you do something not what
- For functions you might want to specify the type of input and the type of output

```
# description of function
# input:  type of input
# output: type of output
my_func <- function(input){
    do_something_here
    return(output)
}
```

Creating documentation: docstring (advanced)

```r
library(docstring)

square <- function(x){
#' Square a number
#'
#' Calculates the square of the input
#'
#' @param x the input to be squared
    return(x^2)
}
```

Creating documentation: roxygen2 (advanced)

Only for packages (short-cut in R-studio): creates a skeleton

```r
#' Title
#'
#' @param x
#'
#' @return
#' @export
#'
#' @examples
square <- function(x){
    return(x^2)
}
```

Script everything!

Trail of bread crumbs—reading data and data wrangling

Script everything you can!

- reading in data (e.g., `read_csv()`, or from API's)
- removing (empty) rows/columns
- (re)name variables
- transform variables (e.g., with `dplyr`)
- reshape data (e.g., `pivot_longer` & `pivot_wider`)

So, do not use excel—no nice breadcrumbs there!

Figures, diagrams and 3D pie charts

`ggplot2` works wonders as each element can scripted and saved

- add elements (e.g., data labels in scatterplots)
- combinations of elements
- use of functions (e.g., to loop over histograms as descriptives, or multiple descriptive maps—with the help of `sf`-package)

Script that output!

- save figures by `ggsave()` or open up a device and keep size constant

```
pdf(file = "../fig/my_w_plot.pdf",
    width = 4, # The width of the plot in inches
    height = 4)
    my_wonderfull_plot
dev.off()
```

- automatically save (regression) output with `stargazer`, `texreg`, `jtools`, ... ⟶ example?

Functions—why? (advanced)

```
df <- data.frame(
  a = rnorm(10),
  b = rnorm(10),
  c = rnorm(10)
)

df$a <- (df$a - min(df$a)) /
  (max(df$a) - min(df$a))
df$b <- (df$b - min(df$b)) /
  (max(df$b) - min(df$a))
df$c <- (df$c - min(df$c)) /
  (max(df$c) - min(df$c))
```

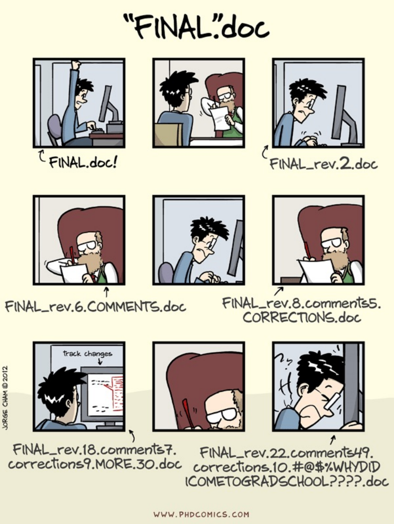
Functions—because (advanced)

Consider writing a function whenever you've copied and pasted a block of code more than twice

```
# input:  vector of numeric/integer values
# output: vector of numeric values scaled at
#         interval [0-1]
rescale <- function(x) {
  rng <- range(x)
  (x - rng[1]) / (rng[2] - rng[1])
}
df$a <- rescale(df$a)
```

Git and Github (advanced)

[Advanced] Git and Github: why-o-why should we do this?



- keep track of versions
- cooperate
- maintain a central repository
- Github: make your work public

[Note 1: a bit like Dropbox]
[Note 2: RStudio has Git functionality]

In conclusion

When should I adopt an open reproducable workflow?

- The sooner the better (now you have time—seriously)
- But think twice about which tools to invest time in
  - which tools go well together (reference management, editing, programming, versioning)?
  - advise: choose well-maintained open-source tools with large communities (R, Python)
  - advise: invest some time in markup languages ((R)Markdown, LaTeX—consider Overleaf platform)
    - bonus: combines wonderfull with Hugo/Jekyll
  - advise: really, really think about versioning (Git & Github)
- Baby steps: start one step at a time

You take the red pill—you stay in Wonderland, and I show you how deep the rabbit hole goes

Questions/comments?

Get the source of this presentation from

`https://github.com/Thdegraaff/reproducibility_nscr`

References i

Ahrens, S. (2017). *How to Take Smart Notes: One Simple Technique to Boost Writing, Learning and Thinking–for Students, Academics and Nonfiction Book Writers.* Sönke Ahrens.

Allen, D. (2001). *Getting Things Done: The Art of Stress-Free Productivity.* Penguin. 354 pp. Google Books: 7PoYBAAAQBAJ.

Arribas-Bel, D. and T. de Graaff (2015). "WooW-II: Workshop on Open Workflows". In: *REGION* 2.2 (2), R1–R2.

Gandrud, C. (2013). *Reproducible Research with R and R Studio.* CRC Press. 316 pp.

Healy, K. (2011). "Choosing Your Work Flow Applications". In: *The Political Methodologist* 18.2, pp. 9–18.