Shared code

Ian D. Gow

15 January 2026

1 Introduction

Open-source software dominates in certain areas. Probably most internet sites run on Linux machines. Python and R are huge in data science and statistics. All of these systems rely on thousands of open-source packages that are continually being improved in part because anyone can see how they work.

Academic research is quite different. Most research papers are really software development projects in disguise. While the final output is a PDF rather than an app, a tremendous amount of coding is often involved and multiple authors can be involved.

Yet the open-source model has not taken off in academia. While some journals are including data-and-code repositories as part of their process, these do not yet dominate. Authors are generally reluctant to share their code and data. There are multiple reasons for this in my view. First, the code is often a "trade secret" of sorts; future papers may be produced and authors may want to retain a competitive edge in what is essentially a zero-sum game.[1] Second, having code available risks making it easy to show how fragile results are. It is difficult to replicate most research papers without access to the code and data, and many papers' results are "fragile" at best. Third, a lot of authors likely fear embarrassment if others could see their code. Academics' code is often inefficient, difficult to read, perhaps even wrong.

For some reason, a lot of the publicly available code in accounting research relates to two seemingly obscure topics: Fama-French industries and winsorization. As we cover both topics in *Empirical Research in Accounting: Tools and Methods*, I discuss these a little below.

Tip

In writing this note, I use several packages including those listed below.[2] This note was written using Quarto and compiled with RStudio, an integrated development environment (IDE) for working with R. The source code for this note is available here and the latest version of this PDF is here.

[1] Zero-sum because papers need to be published in a finite list of journals, which publish a finite number of papers. Publishing your paper on some topic often means not publishing mine.

```
library(dplyr)
library(farr)
```

2 Fama-French industries

Fama-French industry definitions are widely used in finance and accounting research to map SIC codes, of which there are hundreds, into a smaller number of industry groups for analysis.[3] For example, we might want to group firms into 48, 12, or even 5 industry groups.

The basic data on Fama-French industry definitions are available from Ken French's website at Tuck School of Business.

There are multiple classifications, starting with 5 industries, then 10, 12, 17, 30, 38, 48, and finally 49 industries. The data are supplied as zipped text files. For example, the 48-industry data can be found on this page, by clicking the link displayed as `Download industry definitions`.

If we download that linked file and unzip it, we can open it in a text editor or even Excel. The first ten lines of the file are as follows:

```
 1 Agric   Agriculture
           0100-0199 Agricultural production - crops
           0200-0299 Agricultural production - livestock
           0700-0799 Agricultural services
           0910-0919 Commercial fishing
           2048-2048 Prepared feeds for animals

 2 Food    Food Products
           2000-2009 Food and kindred products
           2010-2019 Meat products
```

Looking at the second row, we interpret this as saying that firms with SIC codes between `0100` and `0199` are assigned to industry group `1` (let's call this field `ff_ind`), which has a label or short description (`ff_ind_short_desc`) `Agric` and a full industry description (`ff_ind_desc`) of `Agriculture`.

One approach to this task might be to write a function like the following (this one is woefully incomplete, as it only covers the first two lines of data above):

[2] Execute `install.packages(c("dplyr", "farr", "haven"))` within R to install all the packages you need to run the code in this note.

[3] According to https://siccode.com, "Standard Industrial Classification Codes (SIC Codes) identify the primary line of business of a company. It is the most widely used system by the US Government, public, and private organizations."

```r
get_ff_ind_48 <- function(sic) {
  case_when(sic >= 100 & sic <= 199 ~ 1,
            sic >= 200 & sic <= 299 ~ 1)
}
```

In fact, this is essentially the approach taken in code you can find on the internet (e.g., SAS code here or here or Stata code here).

However, one can do better. The function `get_ff_ind()` in my R package `farr` gets all the data from Ken French's website. One simply specifies the classification desired (e.g., 48-industry using `48`) and gets back a table. As you can see, it takes a fraction of a second.

```r
library(farr)
ff_data <- get_ff_ind(48) |> system_time()
```

```
   user  system elapsed 
  0.094   0.014   0.259 
```

```r
ff_data
```

```
# A tibble: 598 x 6
   ff_ind ff_ind_short_desc ff_ind_desc   sic_min sic_max sic_desc              
    <int> <chr>             <chr>           <int>   <int> <chr>                 
 1      1 Agric             Agriculture       100     199 Agricultural producti~
 2      1 Agric             Agriculture       200     299 Agricultural producti~
 3      1 Agric             Agriculture       700     799 Agricultural services 
 4      1 Agric             Agriculture       910     919 Commercial fishing    
 5      1 Agric             Agriculture      2048    2048 Prepared feeds for an~
 6      2 Food              Food Products    2000    2009 Food and kindred prod~
 7      2 Food              Food Products    2010    2019 Meat products         
 8      2 Food              Food Products    2020    2029 Dairy products        
 9      2 Food              Food Products    2030    2039 Canned & preserved fr~
10      2 Food              Food Products    2040    2046 Flour and other grain~
# i 588 more rows
```

Some might object: "I don't want to install some package" or "I don't trust code I can't see or understand". You don't have to install the package. The code for the function is available here along with detailed explanation of what it is doing. Even the source code for the package is available here.

Others might wonder how to use this function. The SAS code variants above are in the form of SAS macros that take in a data set with SICs and return another data set with Fama-French industry variables added to it. My R variant can function in a similar way, but using more of an "SQL" approach.

In Chapter 19 of *Empirical Research in Accounting: Tools and Methods*, we join a data set `compustat` that contains SIC codes in `sich` with `ff_data` like this. In effect, this is quite close to the data-in-data-out approach of the SAS macros.

```
for_disc_accruals <-
  compustat |>
  inner_join(ff_data,
             join_by(between(sich, sic_min, sic_max))) |>
```

But I would argue that the approach used in my code is superior. It's simpler (not hundreds of lines of code), accurate (no risk of bad transcription in adapting from the data on Ken French's website), and versatile (one function handles classification into 10, 12, 17, 30, 38, 48, or 49 industries).

If you're not an R user, then just export to your chosen format and merge much as I do above.

```
library(haven)
get_ff_ind(48) |> write_dta("ff_data.dta")  # for Stata
get_ff_ind(48) |> write_xpt("ff_data.xpt")  # for SAS
```

3 Winsorization

Many SAS users probably have a SAS macro on their hard drives that they got from somewhere. I too once used such a file. One variant I found has 136 lines of code and uses the data-in-data-out paradigm taken above.

I recently saw an R function `winsorize()` that was 51 lines of dense code (some lines passing well past the conventional 80-character limit). What exactly is the function doing? Really hard to say. Can we do better?

I think so. The `farr` package contains the `winsorize()` function that is (effectively) four lines of code. Because it's such a small function, I can reproduce it here:

```
library(farr)
winsorize
```

```
function (x, prob = 0.01, p_low = prob, p_high = 1 - prob)
{
    cuts <- stats::quantile(x, probs = c(p_low, p_high), type = 2,
        na.rm = TRUE)
    x[x < cuts[1]] <- cuts[1]
    x[x > cuts[2]] <- cuts[2]
    x
}
<bytecode: 0x10789fb30>
<environment: namespace:farr>
```

Most of the work is done by the `quantile()` function from the built-in `stats` package. By choosing `type = 2`, I line up with the choices made in the standard SAS macro.[4] Then all values below the lower bound (`cuts[1]`) are set to that lower bound. All values above the upper bound (`cuts[2]`) are set to that upper bound. Then the result is returned; this is winsorization in a nutshell.

How to use this function?

In Chapter 22 of *Empirical Research in Accounting: Tools and Methods*, we replicate a paper that winsorizes various measures of β at the standard levels (i.e., 1% and 99%), which simply requires the following:

```
reg_data <-
  raw_data |>
  mutate(across(starts_with("beta"), winsorize))
```

In Chapter 19 of *Empirical Research in Accounting: Tools and Methods*, we replicate a paper that winsorizes six variables at the standard levels, but by fiscal year. This is also easily accomplished with `winsorize()`:[5]

```
win_vars <- c("at", "mtob", "leverage", "roa", "da_adj", "acc_at")

reg_data <-
  raw_data |>
  group_by(fyear) |>
  mutate(across(all_of(win_vars),
                \(x) winsorize(x, prob = 0.01))) |>
  ungroup()
```

More on winsorization is provided in Chapter 24 ("Extreme values and sensitivity analysis") of *Empirical Research in Accounting: Tools and Methods*.

[4] The choices relate to handling of ties and interpolating values. See `? quantile` in R for details.

[5] While the inclusion of `prob = 0.01` is not strictly necessary given that that's the default used by the function, this code does illustrate how you could choose `prob = 0.02` to get winsorization at 2% and 98% levels, a popular alternative choice.